
Flexiva

Overview

Flexiva

Flexiva is a flexible and infinitely versatile business software application which connects to all of your ReST API's and provides full CRUD forms workflow for all your data with no programming.

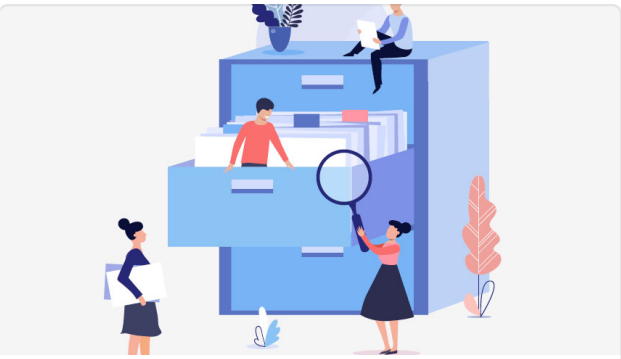
Getting started

Flexiva is available at app.flexiva.co.uk ↗

Simply sign-up from the login page for your free trial.

Explore our resources

We've put together a variety of helpful guides and links to support you.



Overview

A summary of the product, and processes involved in building your application.

App Studio

Configure the application for your specific business requirements.

Sample App

Browse our pre-configured sample app with ReST API data to test.

Beginners Guide

Create your first simple application one step at a time.

Production Designer Guide

Recommended approach to build a production application.

API

The Web API for creating your own Flexiva client-side apps.

Glossary

Here is a list of words, phrases and three-letter-acronyms (TLA's) which are used throughout the documentation.

Word/Phrase/TLA	Description/Meaning
API	Application Programming Interface
App	An application. App is simply shorthand. They are one and the same.
Application	A software application which normally runs on a client-side device. This could be a 'native' installable app, or one which runs in a web browser. See front-end.
Back-end	Usually the database and file server exposing data services via a ReST API. See server-side.
BOT	An automated robot software service which runs periodically and prepares or transfers data into and out of databases or back-end or front-end apps.
Client-Server	The concept of client-server used to be known as master-slave. The client is the device which the end-user operates to interact with the data centrally stored on the server.
Client-side	If something is client-side, it means it runs on devices such as computers, tablets or mobile phones, as opposed to server-side which runs on the data servers. See client-server and front-end.
CRUD	Create, Read, Update and Delete data-oriented operations typically provided by a GUI front-end application.
Designer	The administrator role permissioned to use App Studio to configure the application.
End-Point	The URL of the ReST API. Also known as 'endpoint' or 'end point'.
End-User	The non-designer role for typical users of the client-side application.

Front-end	The software or data exists on the client device where the end-user can interact with the data. See client-side.
GUI	A Graphical User Interface application is how the end-user interacts with the data via 'widgets' known as forms, menus, buttons, fields and grids. Modern GUI's support keyboards, mice and touch sensitive screens.
LoB	Line of Business software application which allows employees of the business to manage corporate data stored in the back-end data servers.
Low-code	A software builder tool which allows a developer to write less code than they would if they used a raw compiler.
Nav Bar	The Navigation Bar appears to the left of the application and is a hierarchical menu to open forms.
No-code	A software builder tool which allows a designer to drag and drop components to create applications without writing any code.
ReST	Representational State Transfer is a protocol for exposing data-centric API's to client-side applications.
Server-side	If something is server-side, it means it runs on the data servers such as API's, databases, file storage, BOT's or some applications. See client-server and back-end.
Toolbar	The toolbar appears on the top of the application and shows different 'elements' such as the Search text box, the History and Add menus, custom buttons, the help button and a drop down user profile menu.

URL	Uniform Resource Locator is a world-wide-web term for the address of a web site, or a web API. Derivatives include URI or End-Point. The term end-point is typically used to describer a URL which is specifically a ReST API.

Introduction to Flexiva

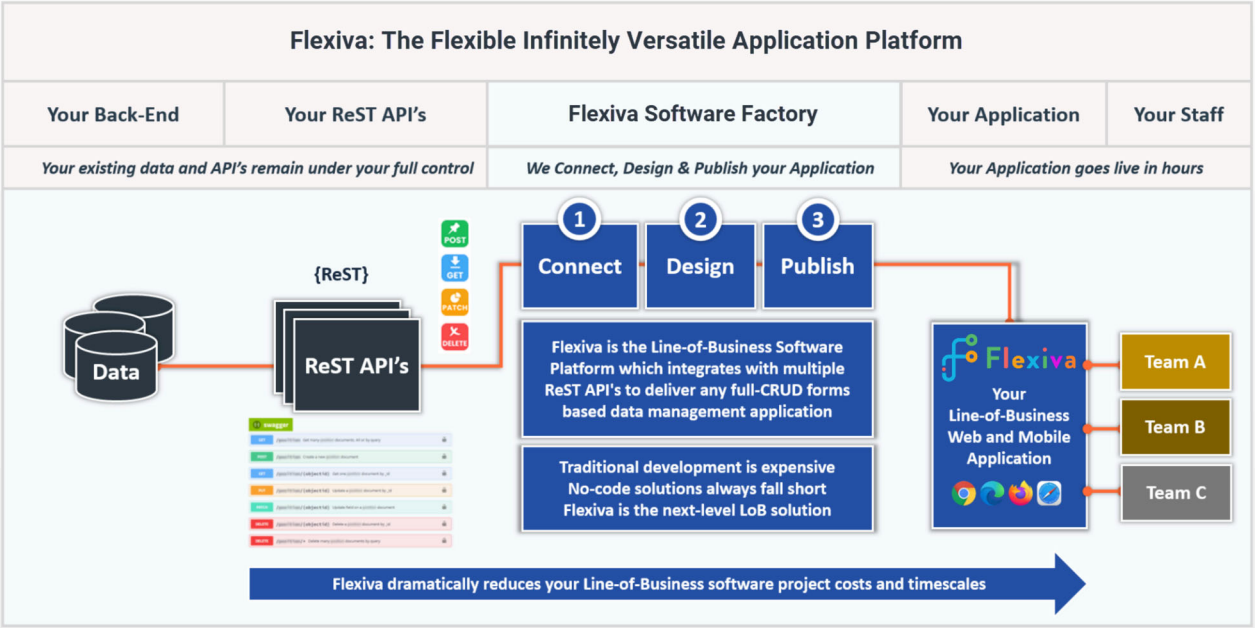
What is Flexiva?

Flexiva is a line-of-business (LoB) application which can connect to your Representational State Transfer (ReST) Application Programming Interface (API) and allow you to design create, read, update, delete (CRUD) forms and all essential management and security access using no-code tools i.e. you do not have to be a front-end software developer to create a web and mobile application for your end-users.

Flexiva is an infinitely versatile application platform which delivers a back-end independent front-end application to web browsers running on any device. You can connect multiple ReST API's to provide access to all your corporate computing resources to your end users in a single highly secure application. You can do this with no-code, in fact you can do this without knowing anything about the complexities of front-end software development. This allows business specialists, rather than software developers, to build LoB apps much faster and cheaper than any other no-code, low-code, full-code or AI vibe-coding solutions available today.

There are only three steps to getting started to bring your back-end ReST API to life for your end-user community:

1. ***Connect*** your ReST API
2. ***Design*** your app, forms, lookups, fields, user access
3. ***Publish*** your web and mobile app for your end-users


Flexiva: The Flexible Infinitely Versatile Application Platform
Your Back-End
Your ReST API's
Flexiva Software Factory
Your Application
Your Staff
Your existing data and API's remain under your full control
We Connect, Design & Publish your Application
Your Application goes live in hours
Data
{ReST}
ReST API's
POST
GET
PATCH
DELETE
1
Connect
2
Design
3
Publish
Flexiva is the Line-of-Business Software Platform which integrates with multiple ReST API's to deliver any full-CRUD forms based data management application
Traditional development is expensive
No-code solutions always fall short
Flexiva is the next-level LoB solution
Flexiva
Your Line-of-Business Web and Mobile Application
Team A
Team B
Team C
Flexiva dramatically reduces your Line-of-Business software project costs and timescales

Conceptual Overview

The Flexiva concept is to bring life to ReST API's and present the data to end-users for manipulation in a mature industrial strength back-end independent business software front-end application.

Your business will have dedicated servers for storing and managing data. Typically these will be databases, plus files. You will also have software applications which access this data, and possibly your web site and mobile app.

In an ideal scenario, your company will have invested in developing a Representational State Transfer (ReST) Application Programming Interface (API) to provide secure and performant create, read, update, delete (CRUD) access to manage your corporate databases.

Investing in a ReST API allows your business to take advantage of modern software applications allowing your end-users to use their desktop, tablet and mobile phone to conduct business flexibly, and also allows you to permit access to third parties to participate in your essential business processes.

The problem is that building powerful line-of-business (LoB) modern software applications which work securely on all devices is an expensive and time-consuming exercise. You will need to hire a team of software developers, business analysts, quality assurance specialists and project manage them. This costs time and money, and brings considerable risk.

Flexiva offers a highly cost-effective alternative.

Flexiva is a no-code LoB application which can be deployed by non-developers i.e. business specialists, saving a tremendous amount of both time and money. Unlike the previous generation of no-code products, the Flexiva app is already assembled, and does not require a 'brick-by-brick' approach.

Simply connect your ReST API's design your forms, and publish your app to your end-users.

Connect

Connect to your ReST API.

Your Application Programming Interface (API) will probably be engineered using the Representational State Transfer (ReST) protocols. This allows data to be Created, Read, Updated and Deleted (CRUD).

You connect each ReST API as a Flexiva data source using your security credentials, and test these to ensure that this works as expected in Flexiva.

Note that your ReST API is never accessed directly by the client-side application. It is only used via our Web API server-side proxy to ensure that all your data traffic remains secure by being hidden from prying eyes.

This is done by opening the App Studio and using the Data Sources configurator. Each of your ReST API CRUD endpoints should be used to create and test data source request so that data can be created, read, updated and deleted.

You will need ReST API endpoints for the following:

Search

The top toolbar has a search box. You will need a ReST API data source request which can take a text string and return a list of all matching records across all of your databases connected to this application. This allows end-users to easily locate records even with no training. Use this configurator to set this up.

Activity Metrics

Both the left navigation bar and the top right account summary drop down can show key performance indicators. You will need a ReST API data source request for each KPI you wish to display. This allows end-users to easily open groups of data even with no training. Use this configurator to set this up.

Custom Variables

In order to allow data to transfer between your front-end application and your back-end, you will need to define a custom variable for each field in each ReST API request for all CRUD methods. It is best to do these one entity at a time. Use this configurator to set this up.

Lookups

All lookup forms connect to a ReST API data source to return pages of records of a specific type. You will need one of these for each 'entity type' you wish to display before you create a lookup form to open from your navigation bar. Use this configurator to create your forms, and this configurator to link these to your navigation bar.

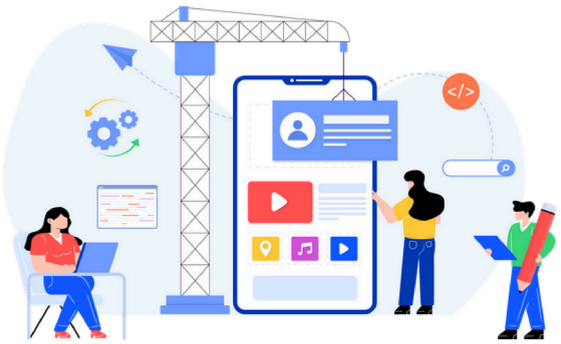
CRUD

In order to facilitate create, read, update and delete for each of your entity data types, you will need to configure a CRUD ReST API method so that you can bind these to a data entry form and design fields and components using the form designer to allow the user to drill down into these forms from lookup forms. You will also need custom variables to build your data dictionary to pass data between the front-end application and the back-end.

Once you have created and tested these data source requests for all necessary CRUD operations, you can now design your forms to start presenting this information for manipulation by your end-users.

Design

Design your application forms, lookups, fields to fit your business requirements.



Once you have connected your ReST API to data source requests, you can now commence the design of your web and mobile application, before publication.

This is done by opening the App Studio and using the following configurators:

- App Toolbar: Configure the application toolbar
- Activity Metrics: Specify what KPI's are displayed after login
- Navigation Bar: Configure your menus and items
- Custom Variables: Configure your own custom variables to share data securely amongst parts of your app
- Forms: Generate and customise CRUD data entry forms for your data sources
- Security: Set security of all system-wide objects per user account
- User/Company Accounts: Manage your end-user accounts and your company account.

Recommended Process

The design process begins after connecting and testing one or more data source requests. This is the recommended order to design your application.

Create Lookup Forms

Create blank lookup forms so you can put together a skeletal application to understand the various tools.

Navigation Bar

Build the navigation bar to open your forms when items are clicked to understand the user workflow.

Create Data Entry Forms

Create data entry forms to allow drill down from lookup forms.

Custom Variables

Create custom variables for each field in each ReST API request for all CRUD methods.

Design Forms with Data

Once your navigation is operating, connect your lookup forms to ReST API data sources via components to display data and allow drill down. Then design data entry CRUD forms to create, read, update and delete data.

Add Menu

In order to allow users to add new records, configure the app toolbar add menu.

History Menu

In order to allow users to locate previously opened records, configure the history menu to show the correct data.

Activity Metrics

To display KPI's, configure the activity metrics.

User Accounts

Invite your end-users to login using this configurator.

Security

Set up which functions and UI elements are visible or enabled for specific users or teams using this configurator.

Final Touches

Add custom F1 help for each form, and write a user guide similar to this one you are reading.

Publish

Publish your web/mobile business application to your end-users.

After you have connected, designed and configured your application, you can now publish the application for use by your private end-user community i.e. colleagues.

The publication process is actually automatic because your application is live the moment you start designing it. This allows you to both design and test it on all your devices at the same time.

Once you wish to share your application with your colleagues, simply invite them to login using the User/Company Accounts configurator.

App Studio

Introduction

What is App Studio?

App Studio is available to subscribers who are allocated the role of 'designer'. A designer can configure all aspects of the application such as data sources, navigation bar, toolbar, forms, metrics, variables, security, and user management.

Each *configurator* is purpose built to allow the designer to customise a specific sub-system of the application.

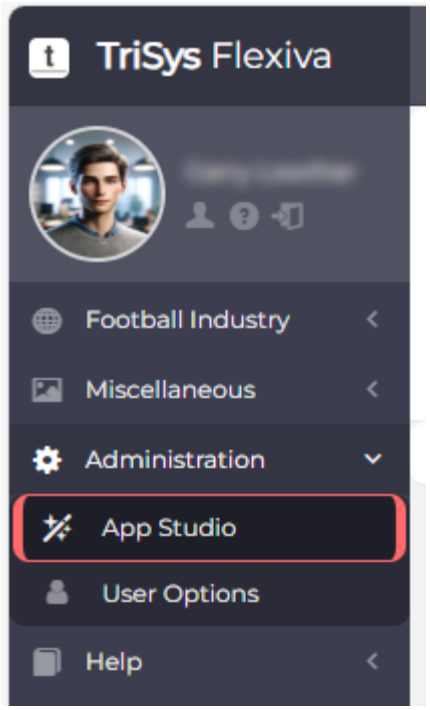
Because no developer experience is required, the Flexiva application is configured, rather than programmed, using standard point and click, or drag and drop. This eliminates a huge amount of complexity typically associated with traditional programming, or indeed the latest no-code 'brick-by-brick' software assembly platforms.

The Flexiva application is already built, and all complex engineering and design decisions have already been completed. This allows the designer to focus on addressing the business requirements, much faster, therefore much cheaper, than using previous generation no-code app builders.

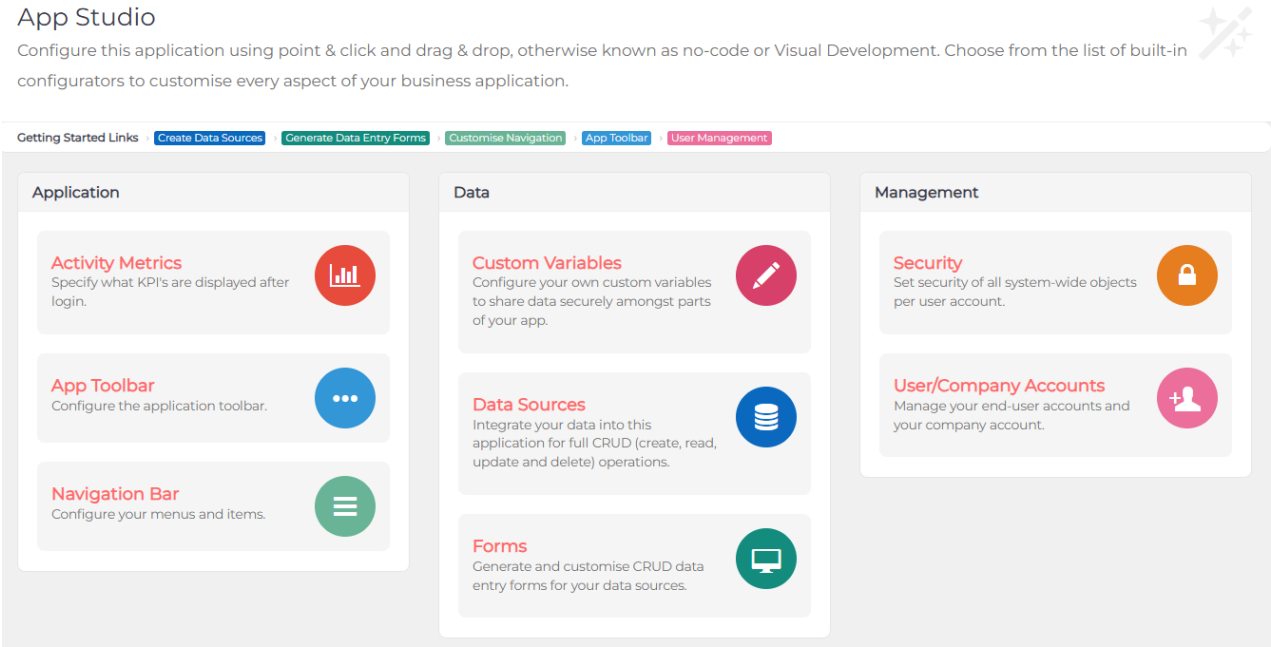
Opening App Studio

Getting started with App Studio

You will find App Studio inside the navigation bar under the **Administration** group:



Opening this form will show all of the application configurators available to you ordered alphabetically by category and configurator name:



There are three distinct groups of configurators:

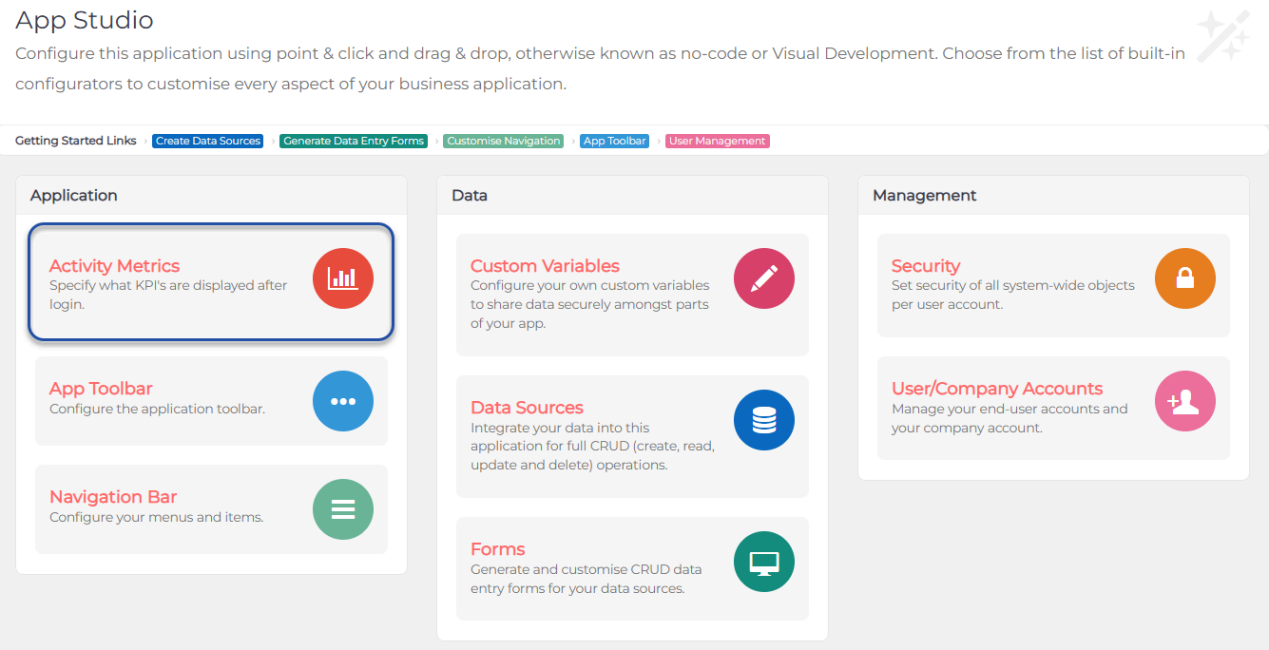
- **Application**: App-specific design
- **Data**: Configuring the use of data in forms
- **Management**: Accounts and security

The following pages discuss each of these configurators in detail.

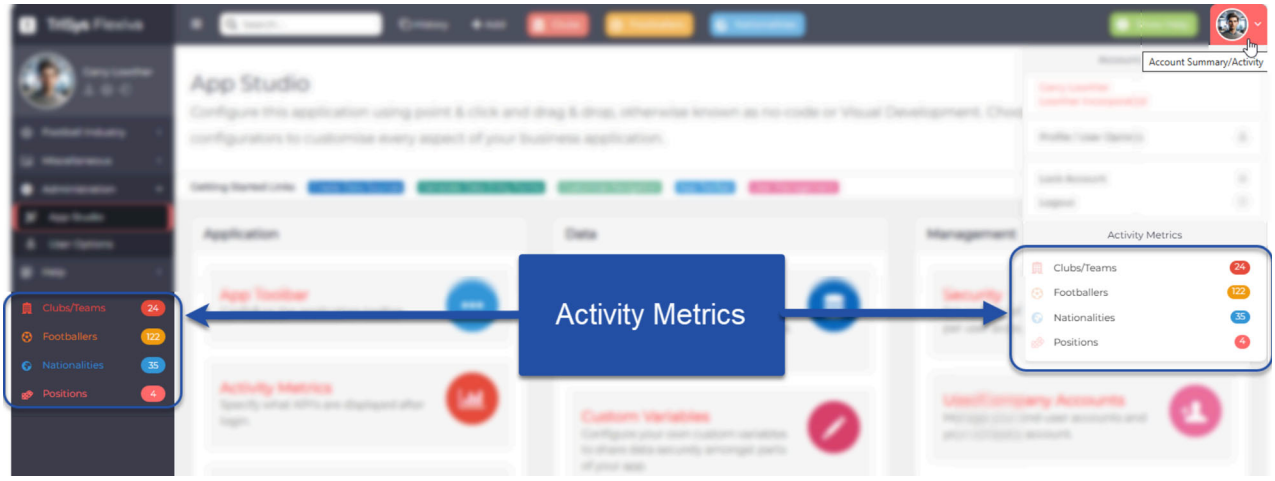
Activity Metrics

Configuring the display of essential key performance indicators.

The Activity Metrics configurator is available from the App Studio.



The Activity Metrics Configurator is a modal dialogue with a master hierarchical tree of activity metrics on the left, with metric properties displayed on the right. There are two places where activity metrics can be displayed.

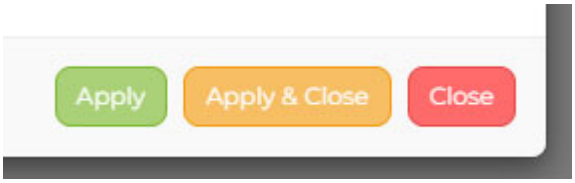


Each activity metric is linked to a ReST API data source request which simply returns a number which is displayed in your chosen colour on the navigation bar or drop-down account summary menu, or both.

When the user clicks on each metric, you can specify which form should open. This approach draws user attention to the important key performance indicators.

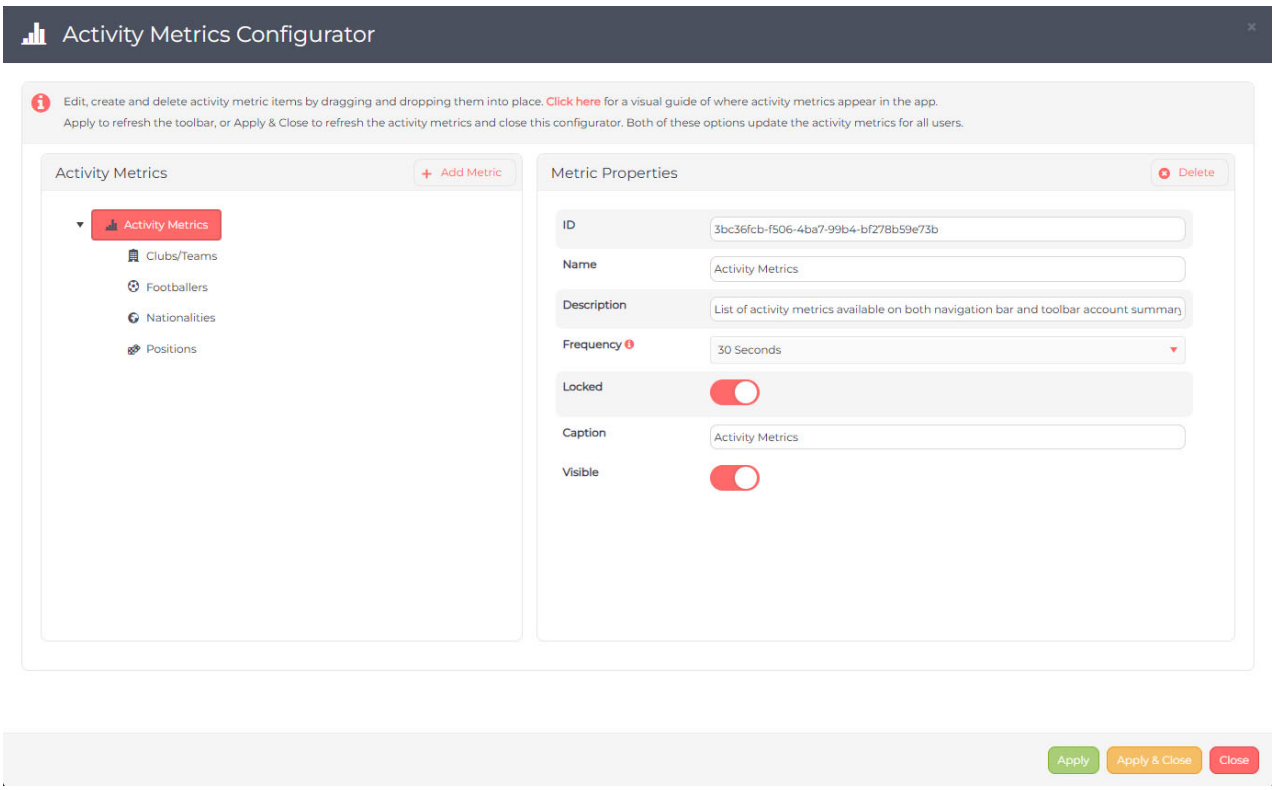
Each user can have their own specific KPI's controlled by the security subsystem.

When editing the properties of each toolbar element, the 3 buttons to the bottom right of the popup modal dialogue are used to either save or discard your changes:



Button	Action
Apply	Apply/save your changes. The app toolbar refreshes to reflect the changes. The popup remains open.
Apply & Close	Apply/save your changes. The app toolbar refreshes to reflect the changes. The popup closes.
Close	Cancel/discard all changes. The app toolbar does not refresh. The popup closes.

Activity Metrics



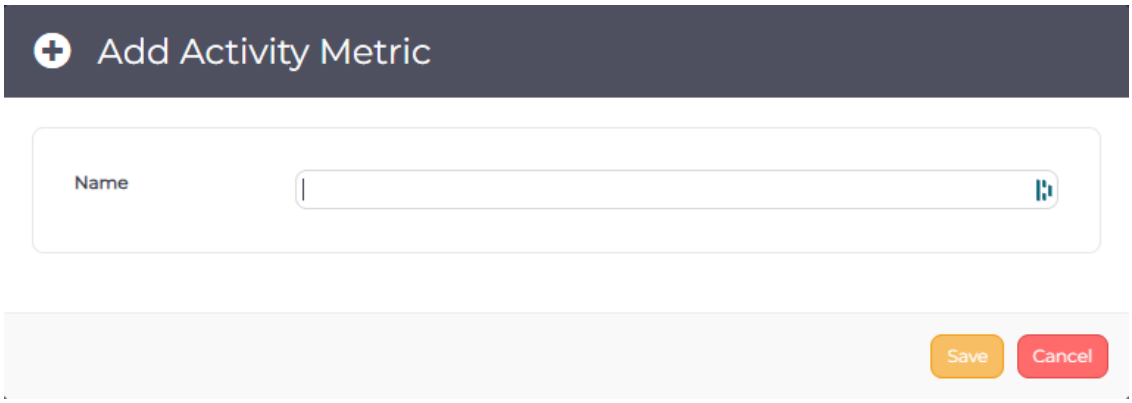
The Activity Metrics hierarchical tree appears to the left showing all configured metrics beneath. Each metric can be dragged and dropped into the preferred location when ordering items.

The Metric Properties are as follows:

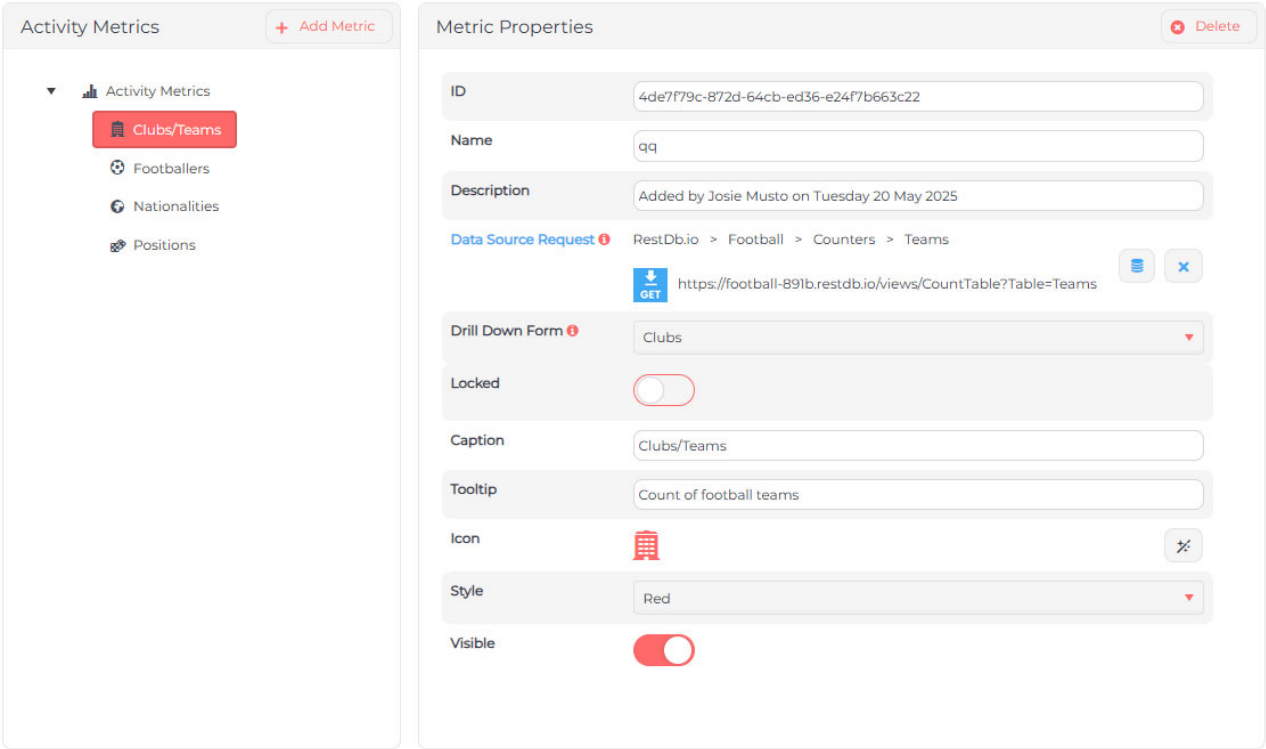
Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Frequency	How often the metric is refreshed in seconds. This lower this number, the more frequently the associated data source request is called.
Locked	Whether the item is locked. The Activity Metrics root is locked i.e. cannot be removed.
Caption	The text of the activity metrics. For example "KPI's".
Visible	Whether this is visible or not.

Add Metric

The add metric button appears in the left panel and is used to configure a new activity metric.



Type a descriptive name for your metric. This must be unique and you will not be able to change this later. Once saved, your new metric will appear selected in the tree on the left.



The Metric Properties can now be edited.

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the metric.
Description	A description about what this metric does.
Data Source Request	This is the ReST API data source request which is used to return a number which is displayed in the metric.
Drill Down Form	When the user clicks on the activity metric, this is the form which is opened.
Locked	Whether the item is locked. The History menu is locked i.e. cannot be removed.
Caption	The text of the menu item.
Tooltip	When the user hovers over the menu item you can remind them of any specific instructions.
Icon	The icon that appears to the left of the menu item.
Style	The custom button can be styled using the standard 'bootstrap' colours: Blue [info], Green [success], Grey [default], Red [danger], Themed, Yellow [warning].
Visible	Whether this is visible or not.

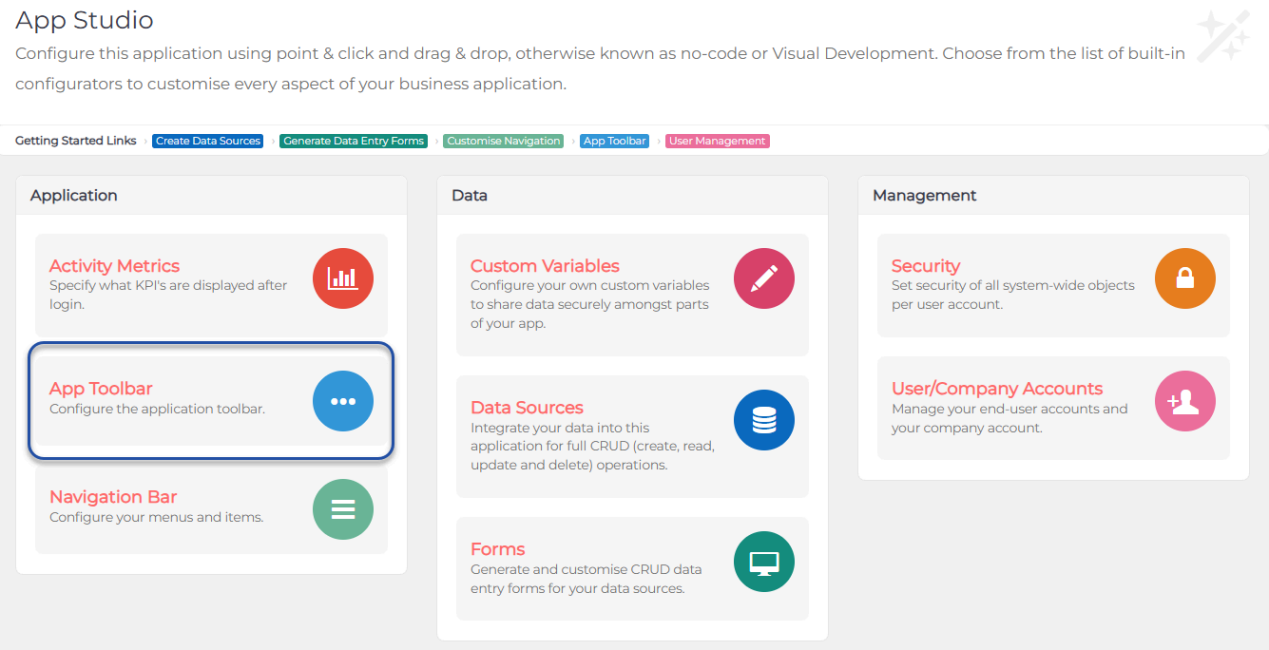
Delete Metric

The Delete button is used to delete the selected metric. This will remove it from the list of activity metrics displayed.

App Toolbar

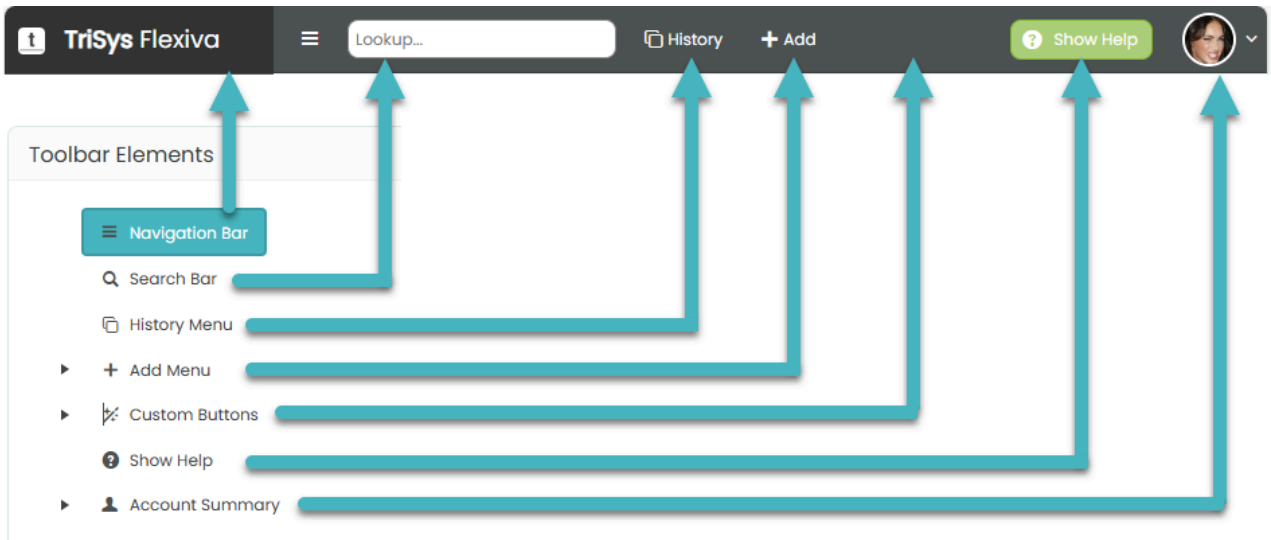
Configuring the application toolbar components

The App Toolbar configurator is available from the App Studio.

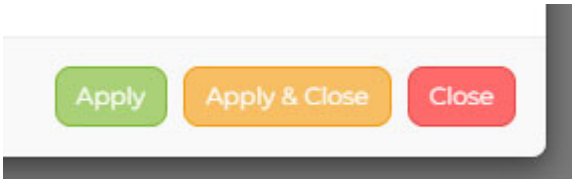


The App Toolbar Configurator is a modal dialogue with a master hierarchical tree of toolbar elements on the left, with element properties displayed on the right. There are 7 major elements to the toolbar.

Each of the toolbar elements configures one or more toolbar components:



When editing the properties of each toolbar element, the 3 buttons to the bottom right of the popup modal dialogue are used to either save or discard your changes:

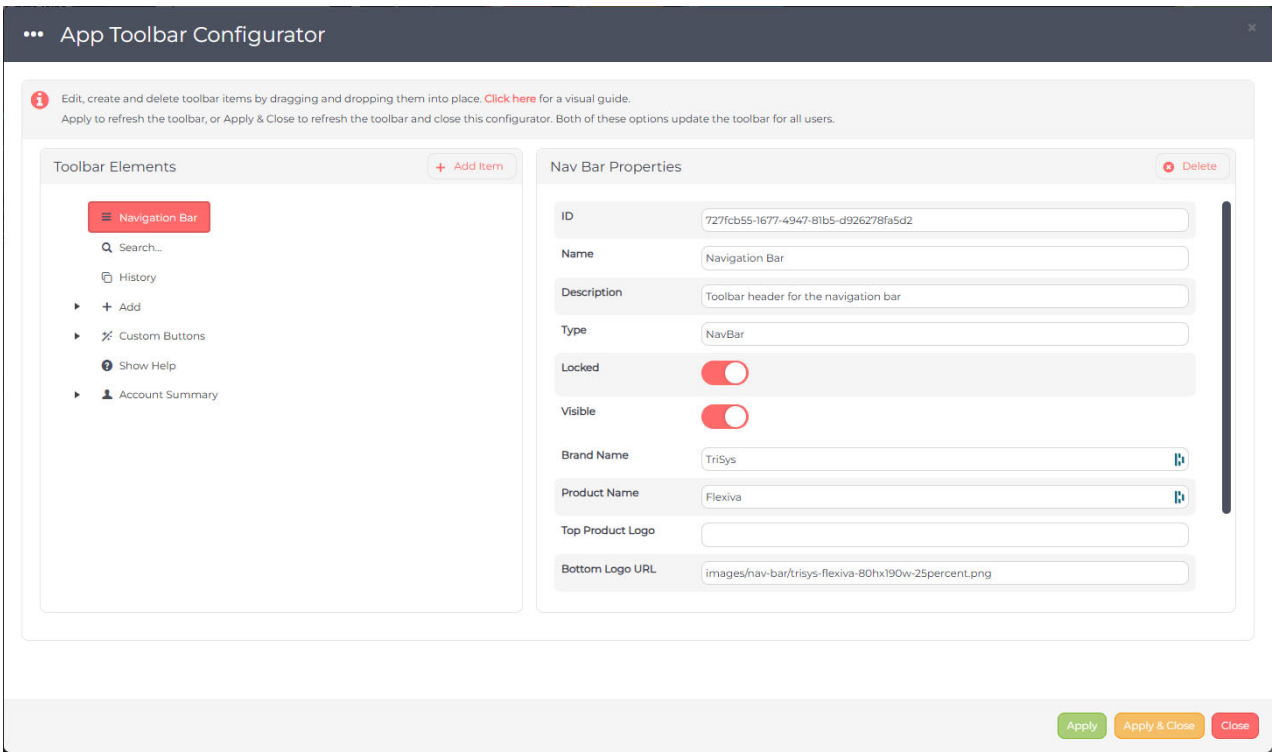


Button	Action
Apply	Apply/save your changes. The app toolbar refreshes to reflect the changes. The popup remains open.
Apply & Close	Apply/save your changes. The app toolbar refreshes to reflect the changes. The popup closes.
Close	Cancel/discard all changes. The app toolbar does not refresh. The popup closes.

There is also a small ***x*** to the top right of the popup dialogue. Clicking this will close the popup without saving your changes.

Each of the toolbar elements are documented below.

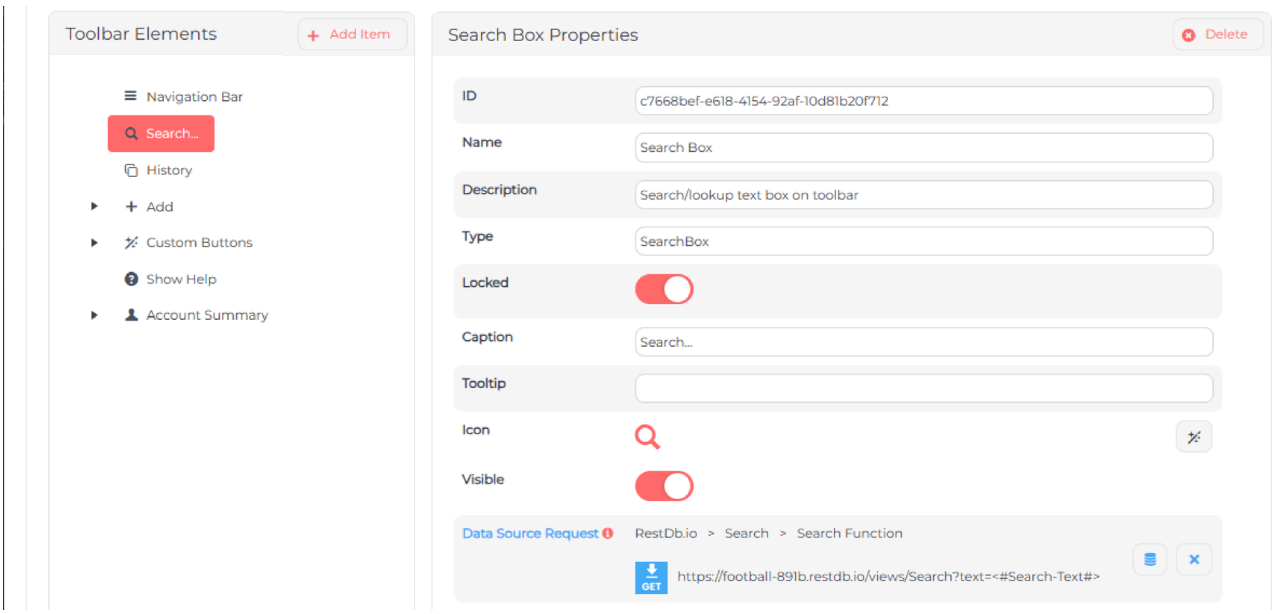
Navigation Bar



The navigation bar appears to the left of the application and it contains groups of menu items configured using its own configurator, however there are other non-menu elements, and these can be configured as follows:

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of the element. NavBar is shorthand for a navigation bar.
Locked	Whether the item is locked. NavBar is always locked.
Visible	Whether the navbar is visible or not.
Brand Name	The brand name of the application. You can brand your application to match your business.
Product Name	Whilst Flexiva is the name of our product, you can call yours whatever you like.
Top Product Logo	You can add your own company logo to the top of the navbar.
Bottom Logo URL	You can also add your own company logo to the bottom of the navbar.

Search...



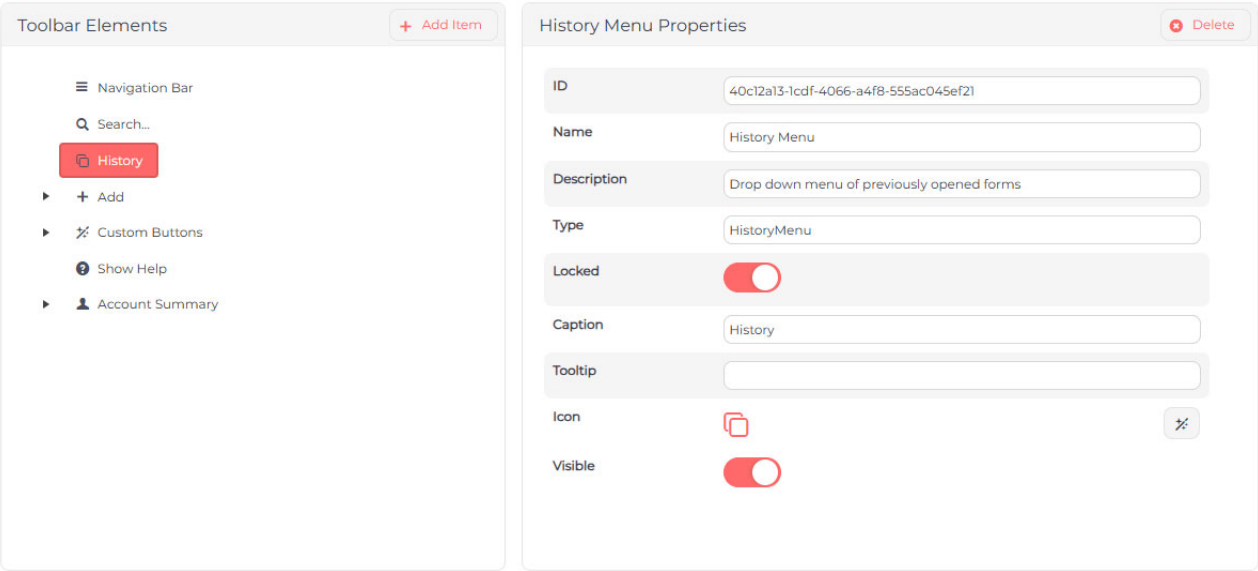
The search box appears on the top of the toolbar to the right of the navigation bar. It is used by end-users to search for all data in all of your ReST API's configured as data sources.

The most important item in this is therefore the Data Source Request which will be a Read ReST API which searches over all your ReST API's. Examples might be to allow end-users to search for products by a code, or a customer by name, or a vehicle by registration number.

Here is a list of all of the search box properties:

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of element.
Locked	Whether the item is locked. The search box is locked i.e. cannot be removed.
Caption	The watermark text of the search box.
Tooltip	When the user hovers over the search box you can remind them of any specific instructions.
Icon	The icon that appears to the left inside the search box.
Visible	Whether this is search box visible or not.
Data Source Request	This is the ReST API data source request which is used to search across all of your databases supplying data to this application.

History

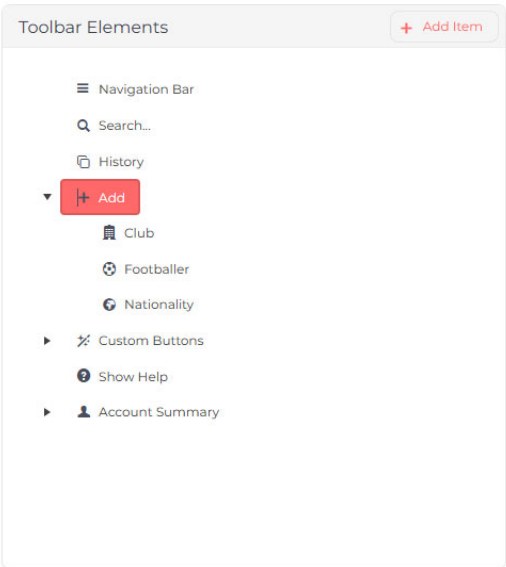


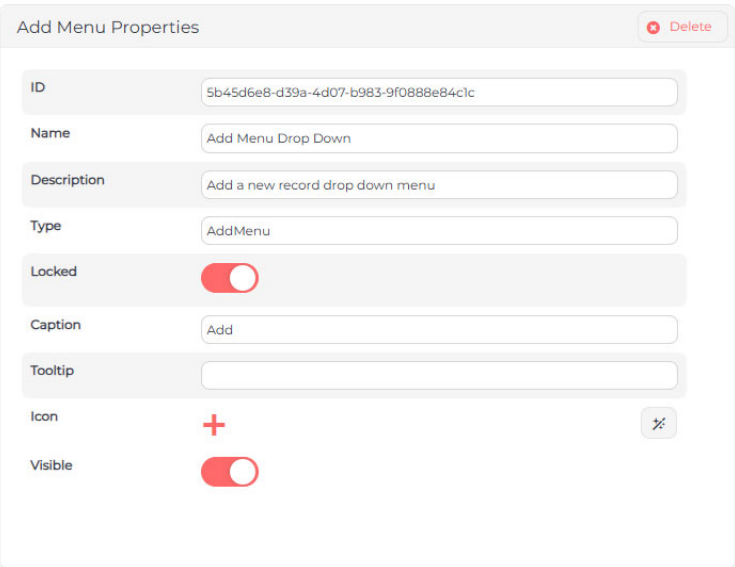
The history drop down menu appears on the top of the toolbar to the right of the search box. It shows all previously opened forms, allowing end-users to quickly resume working with a specific lookup or record.

Here is a list of all of the history properties:

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of element.
Locked	Whether the item is locked. The History menu is locked i.e. it cannot be removed.
Caption	The text of the history menu button.
Tooltip	When the user hovers over the history menu you can remind them of any specific instructions.
Icon	The icon that appears to the left of the menu caption.
Visible	Whether this is visible or not.

Add



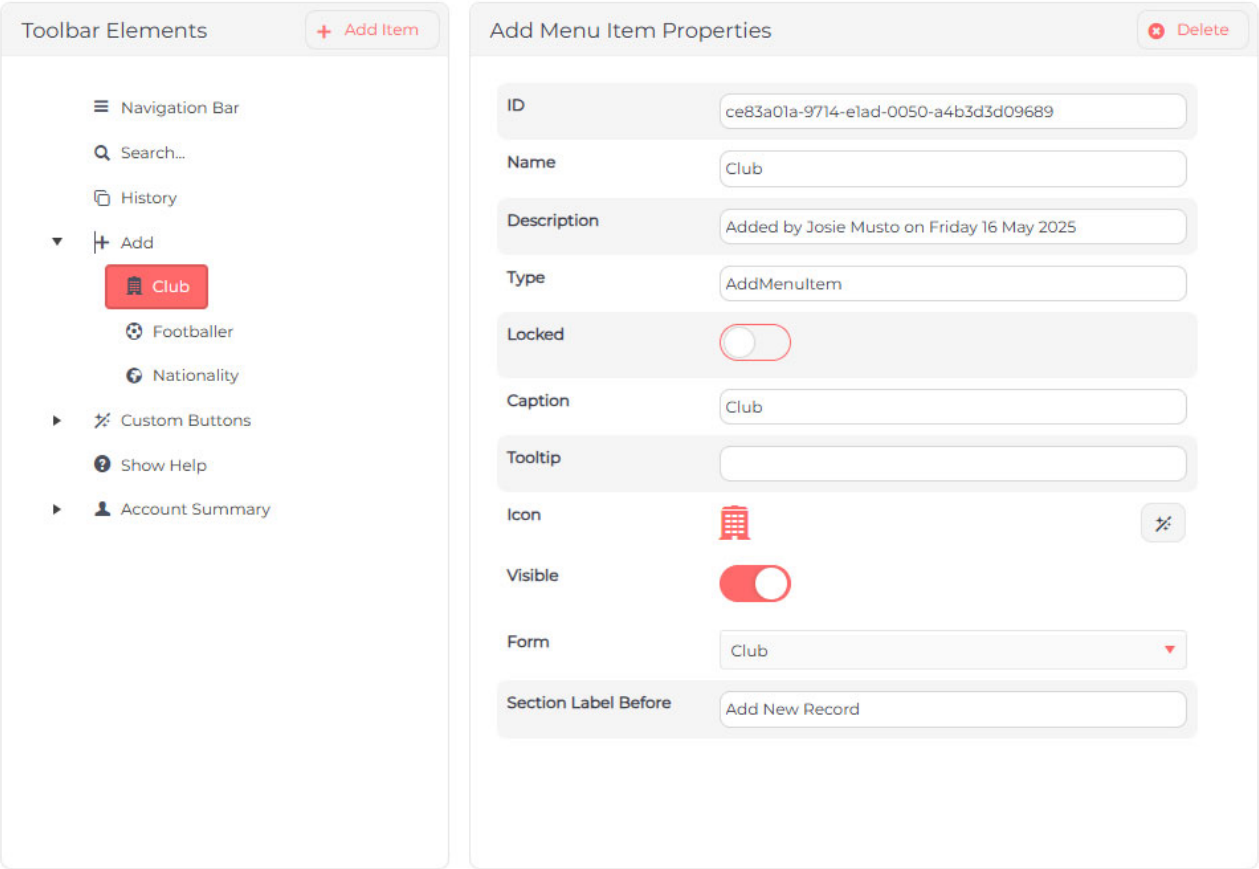


The Add toolbar element appears to the right of the History drop down menu. It is a drop down menu of all data entry forms allowing users to quickly add records.

Here is a list of all of the add menu properties:

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of element.
Locked	Whether the item is locked. The Add menu is locked i.e. it cannot be removed.
Caption	The text of the menu item.
Tooltip	When the user hovers over the menu item you can remind them of any specific instructions.
Icon	The icon that appears to the left of the menu item.
Visible	Whether this is visible or not.

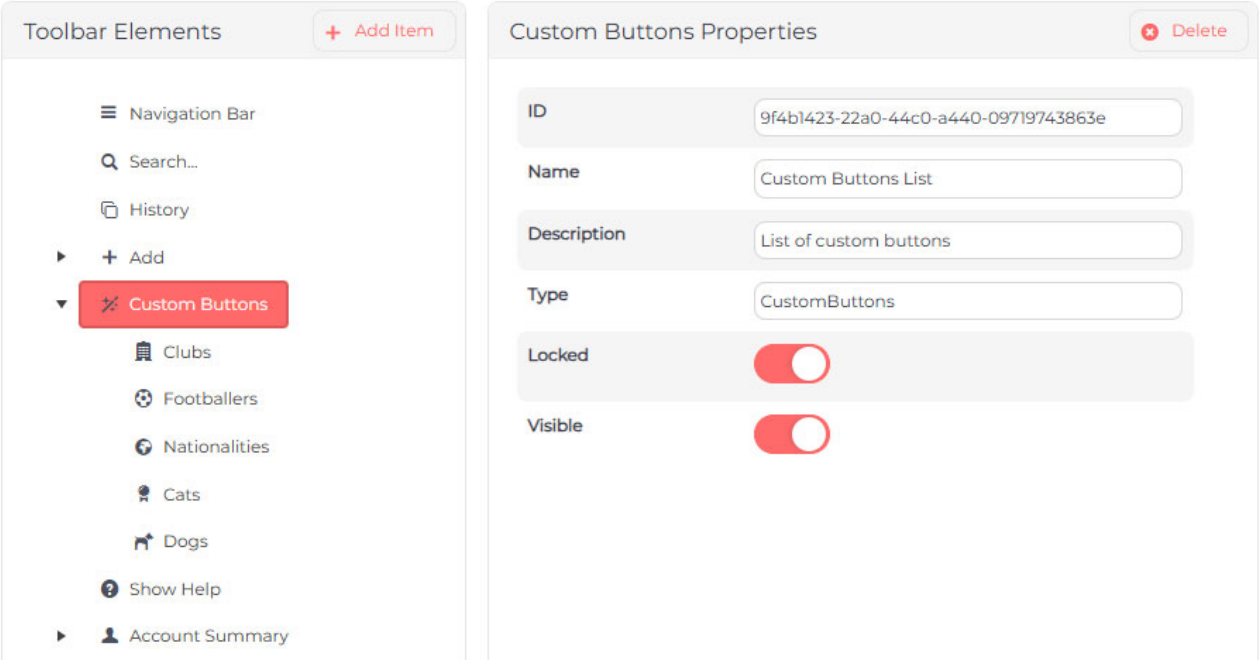
Any number of data entry forms can be added to this menu, and each item can be configured also using properties. Note that ordering the add menu items is done by dragging and dropping the menu item into the correct order.



Here are the properties of each add menu item:

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of element.
Locked	Whether the item is locked. The Add menu item is not locked i.e. it can be removed.
Caption	The text of the menu item.
Tooltip	When the user hovers over the menu item you can remind them of any specific instructions.
Icon	The icon that appears to the left of the menu item.
Visible	Whether this is visible or not.
Form	This a drop down to a form which is opened when the user selects this menu item.
Section Label Before	Menu items can be grouped for clarity. This text creates a section header above this menu item with this text.

Custom Buttons



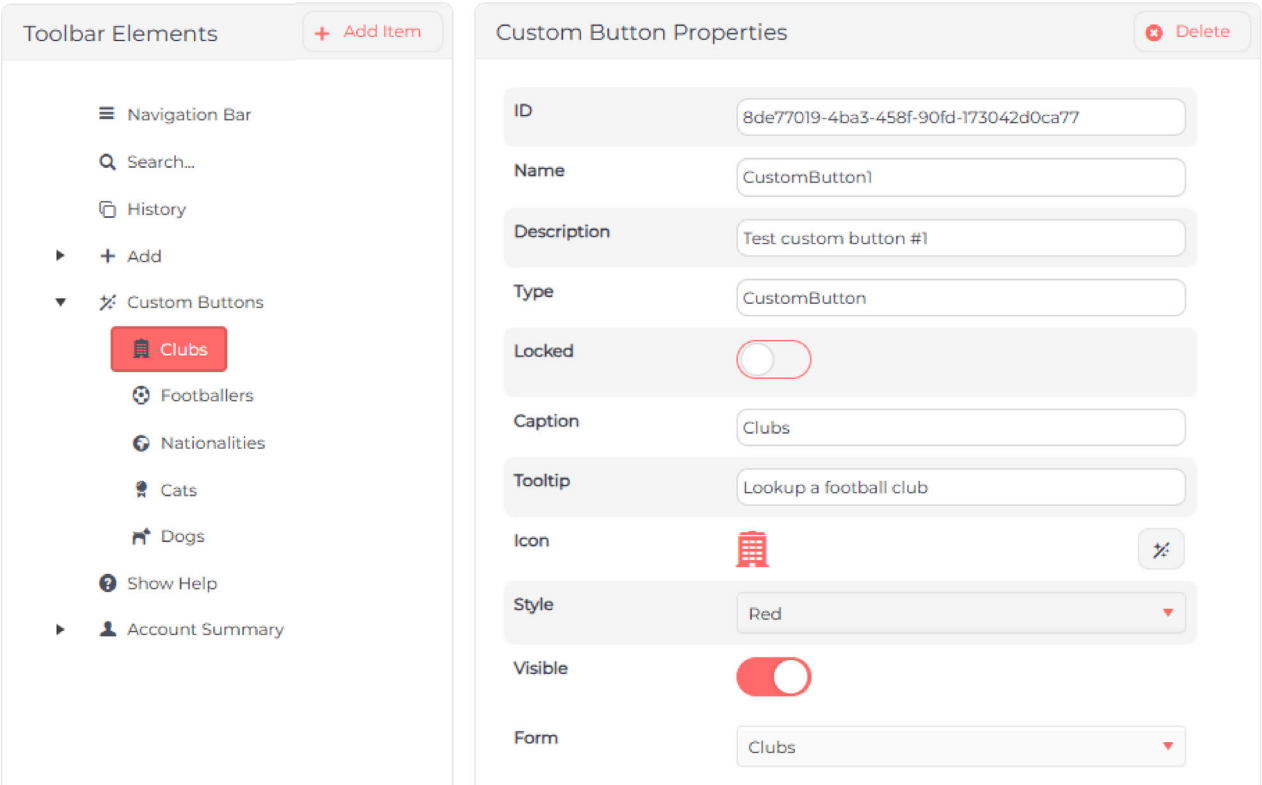
Custom buttons can be added to the toolbar to the right of the Add drop down menu.

Each custom button can be linked to a form.

The customs button group has these properties:

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of element.
Caption	Whether the item is locked. The custom button is not locked i.e. it can be removed.
Tooltip	When the user hovers over the button you can remind them of any specific instructions.
Icon	The icon that appears to the left of the button.
Style	The button can be styled using the standard 'bootstrap' colours: Blue [info], Green [success], Grey [default], Red [danger], Themed, Yellow [warning]
Visible	Whether this is visible or not.
Form	This a drop down to a form which is opened when the user selects this custom button.

Each specific custom button can be configured also:

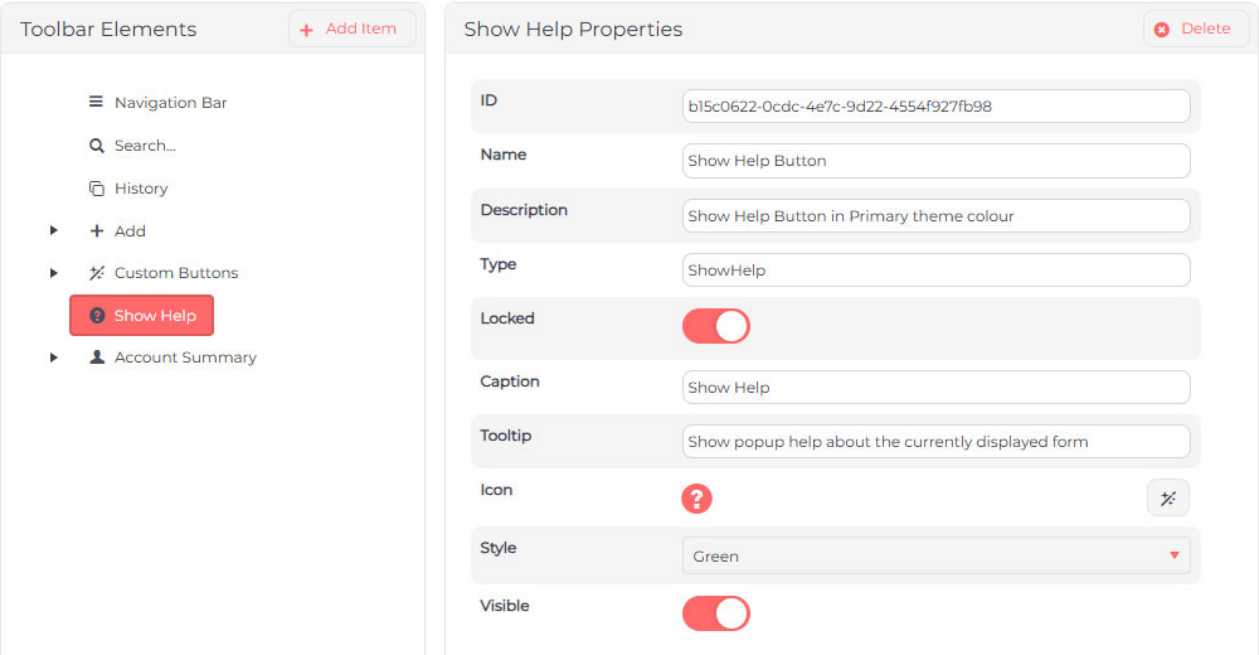


The ordering of each custom button can be repositioned by dragging and dropping the custom button into the desired location.

The custom button properties are:

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of element.
Locked	Whether the item is locked. The custom button is not locked i.e. can be removed.
Caption	The text of the custom button.
Tooltip	When the user hovers over the custom button you can remind them of any specific instructions.
Icon	The icon that appears to the left of the custom button.
Style	The custom button can be styled using the standard 'bootstrap' colours: Blue [info], Green [success], Grey [default], Red [danger], Themed, Yellow [warning].
Visible	Whether this is visible or not.
Form	This a drop down to a form which is opened when the user selects this custom button.

Show Help

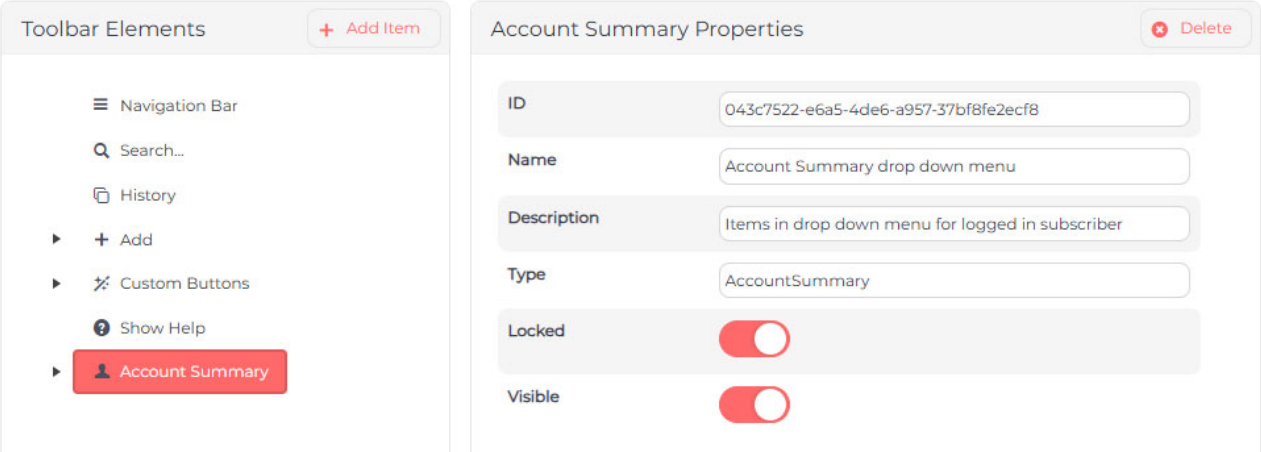


The show help button appears on the toolbar to the immediate left of the account summary drop down.

The following properties can be set:

Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of element.
Locked	Whether the item is locked. The show help button is locked i.e. cannot be removed.
Caption	The text of the show help button.
Tooltip	When the user hovers over the show help button you can remind them of any specific instructions.
Icon	The icon that appears to the left of the show help button.
Style	The button can be styled using the standard 'bootstrap' colours: Blue [info], Green [success], Grey [default], Red [danger], Themed, Yellow [warning]. The default is green.
Visible	Whether this is visible or not.

Account Summary



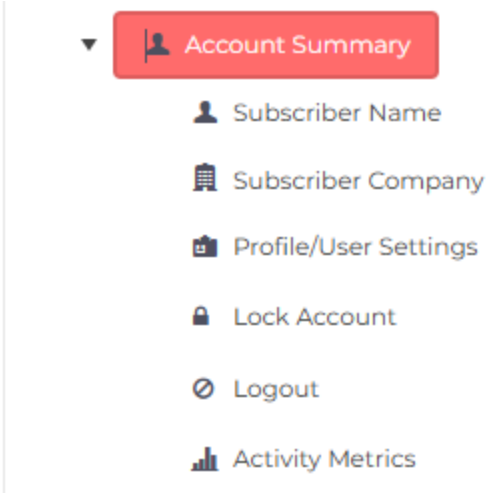
The account summary is a drop down menu which appears on the far right of the toolbar.

It displays useful information about the current logged in user.

Here are the account summary properties:

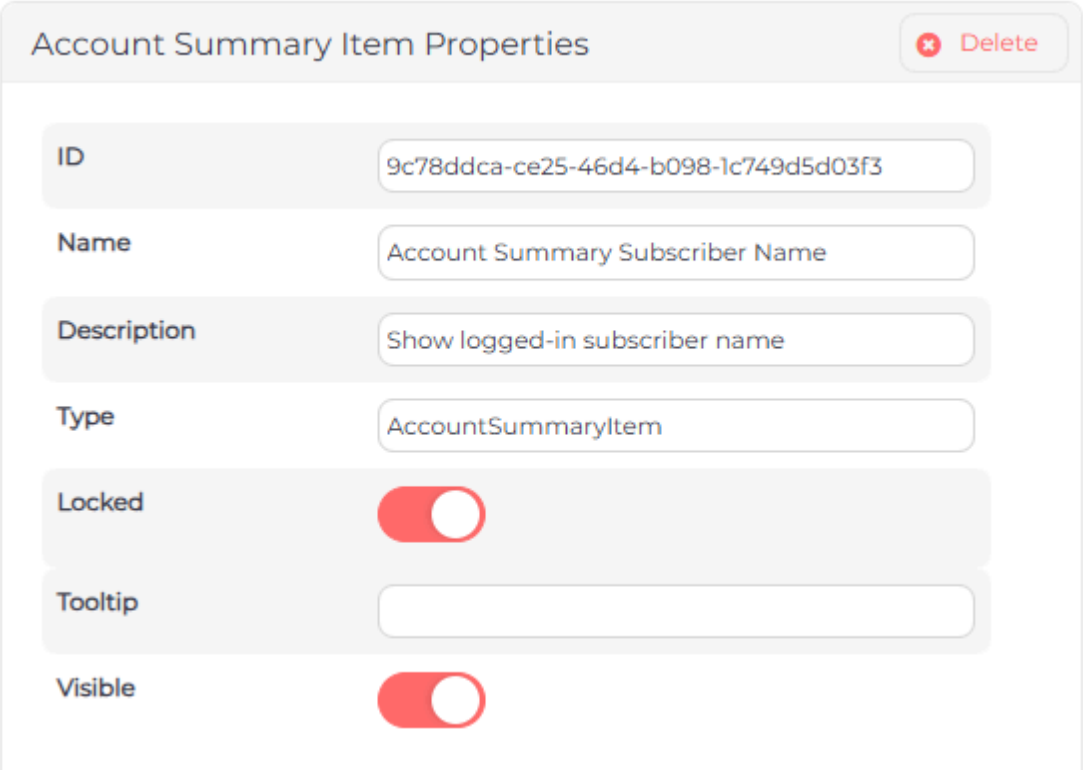
Field	Description
ID	The read-only unique identifier of this element. This is only of importance to developers wishing to customise this on-the-fly.
Name	This is the read-only name of the element.
Description	A description about what this element does.
Type	This is the read-only type of element.
Locked	Whether the item is locked. The Activity Summary is locked i.e. cannot be removed.
Visible	Whether this is visible or not.

There are a number of items comprising the account summary drop down menu.



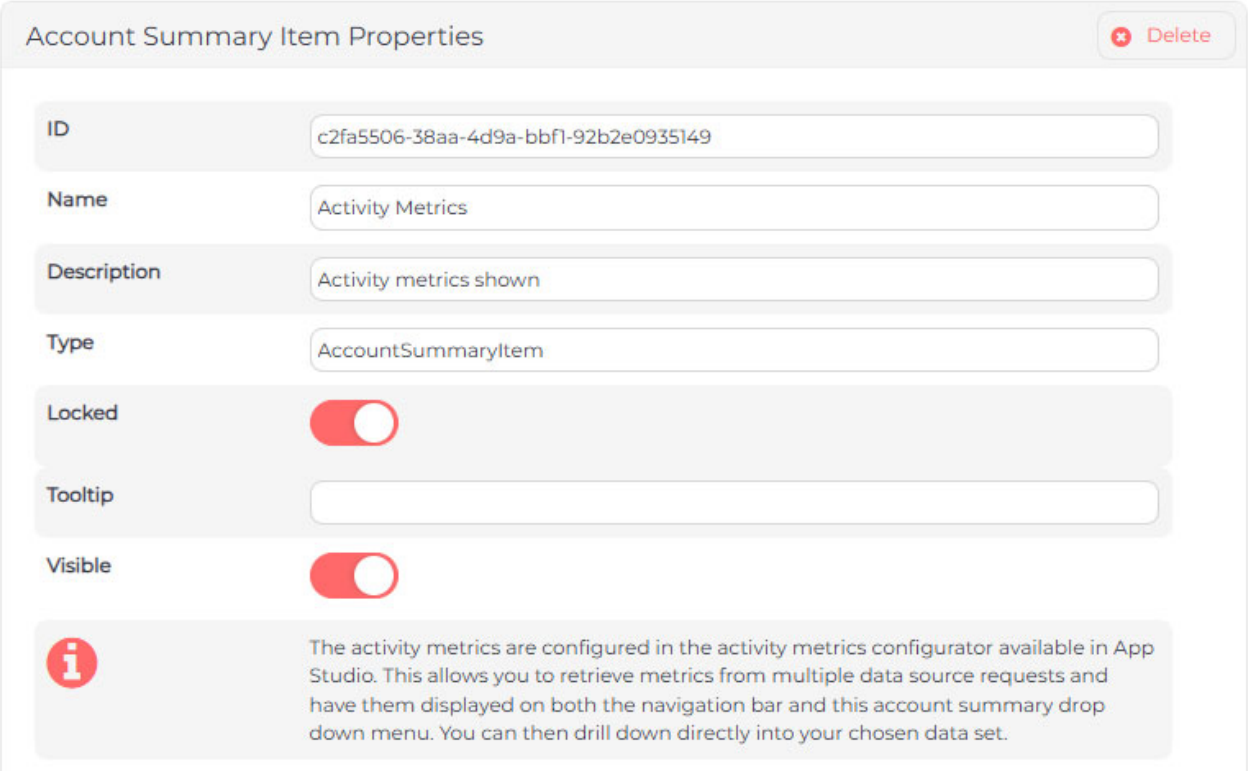
Each menu item can be dragged and dropped into the preferred position.

Both the Subscriber Name and Subscriber Company have similar properties to control how the respective logged-in user name and company are displayed:



The Profile/User Settings, Lock Account and Logout have an additional Icon property.

The Activity Metrics menu item properties can also be configured, however the actual metrics themselves are controlled by the specialised activity metrics configurator.



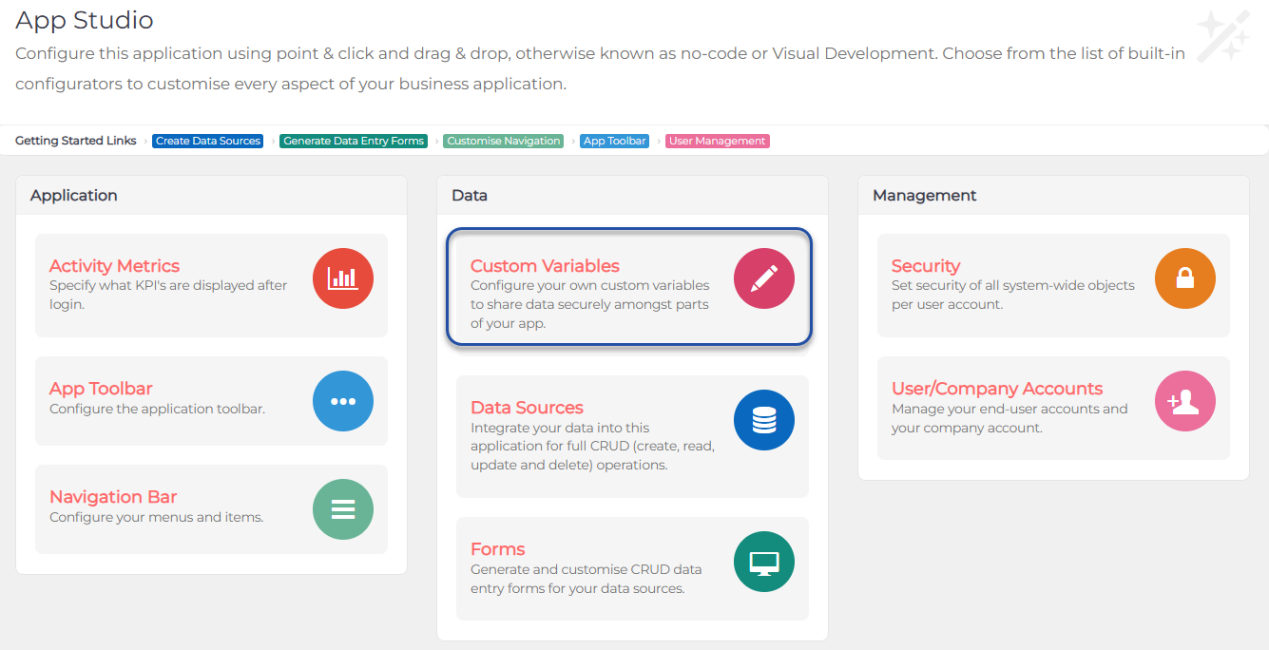
Delete

The Delete button will delete the selected tree view element. Once deleted, the element will no longer be visible in the toolbar.

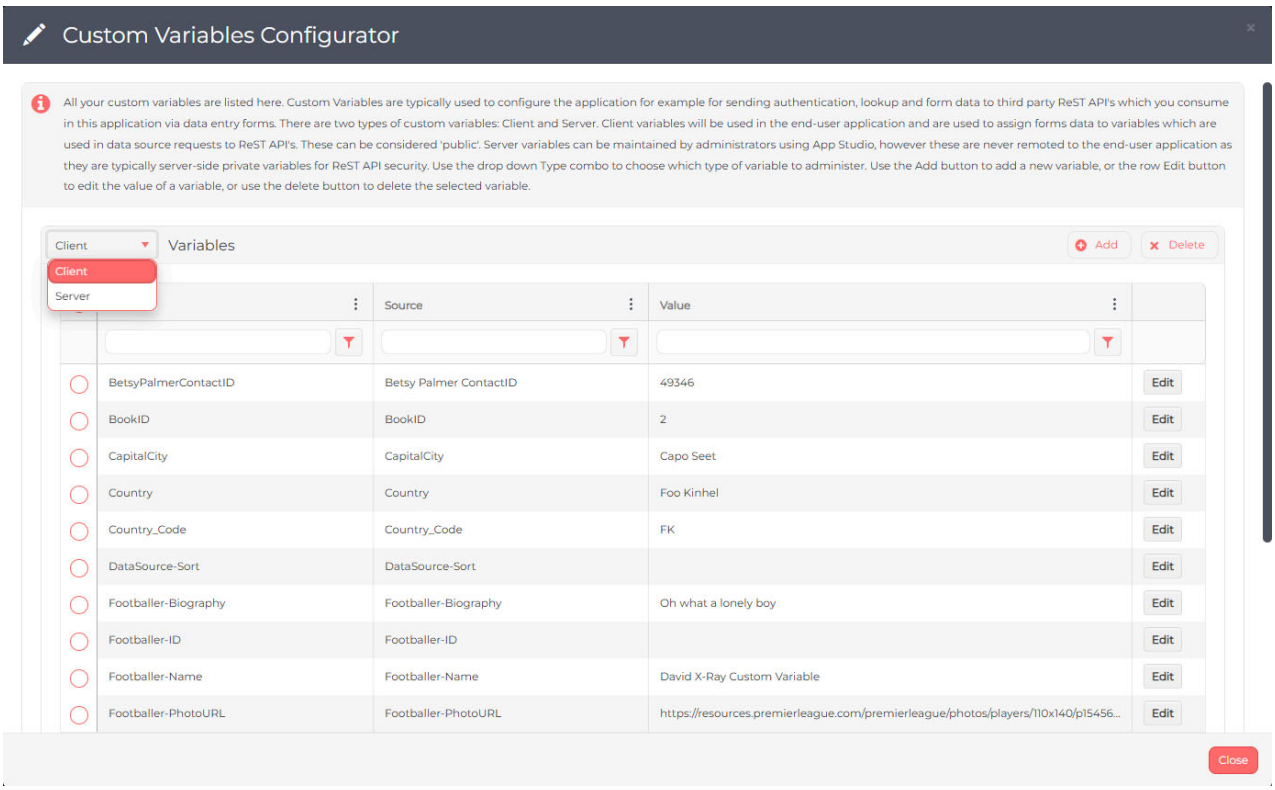
Custom Variables

Configuring the custom variables is important to allow contextual data to flow to and from the back-end ReST API's and the front-end client-side application.

The Custom Variables configurator is available from the App Studio.



The Custom Variables Configurator is a modal dialogue with a grid showing all Client or Server variables.



Custom variables can be thought of as a transient 'data dictionary' where data is automatically passed through the application forms to and from ReST API's to facilitate all CRUD operations.

Client Custom Variables

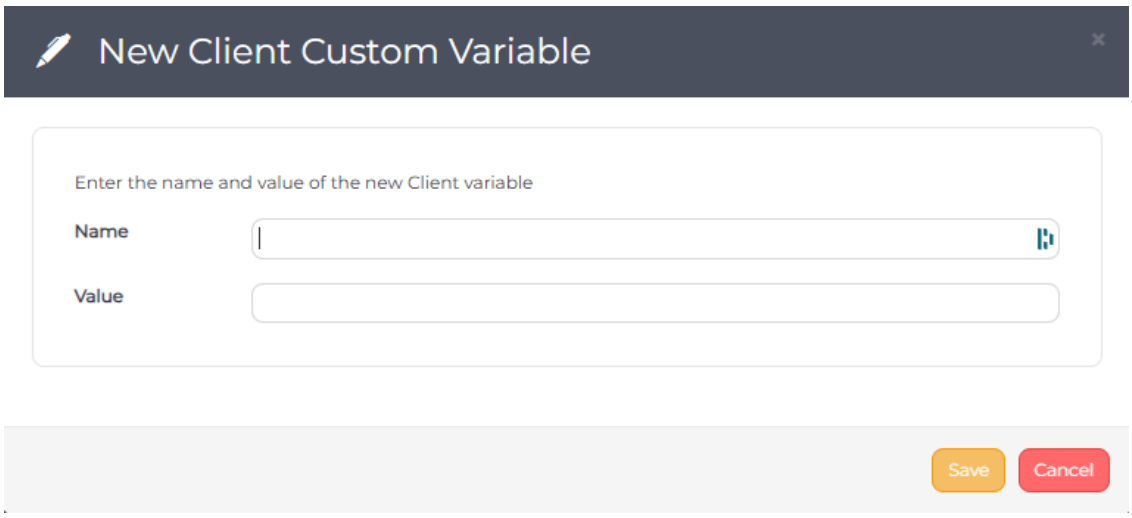
Client variables exist only on the client-side application and are transient i.e. they change constantly when they are linked to forms, allowing data to automatically flow both inside the application and to and from the back-end via the ReST API data source requests.

Server Custom Variables

Server variables are used only on the server-side and are never remoted to the client-side application when end-users are logged in. These are typically ReST API security credentials. Because all ReST API connectivity is conducted by the Flexiva Web API, the client-side never knows how the back-end data is accessed. This adds a high level of security to your application.

Add

The add button is used to create a new custom variable.



Enter the name and initial value of your variable. The name should be as descriptive as possible as it will be referenced in many places. For example "ID", or "Name" is far too ambiguous. It ought to be very specific e.g. "Product-ID" or "Fleet-Car-Registration-Number".

Saving the custom variable adds it to the grid.

Delete

The delete button is used to remove the selected custom variable.

WARNING: If you have linked data source requests or forms to this custom variable, then these may stop working, so this operation can be destructive to your application.

Grid

The grid shows the list of all custom variables for the specific variable scope (Client or Server).

The grid columns are as follows:

Column Name	Description
Name	The name of the custom variable. This must be unique.
Source	The source will often be the same as the variable name, however for some intrinsic custom variables, this may be the name of the internal system variable.
Value	The current value of the custom variable.
Edit	The edit button allows the value of the custom variable to be changed.

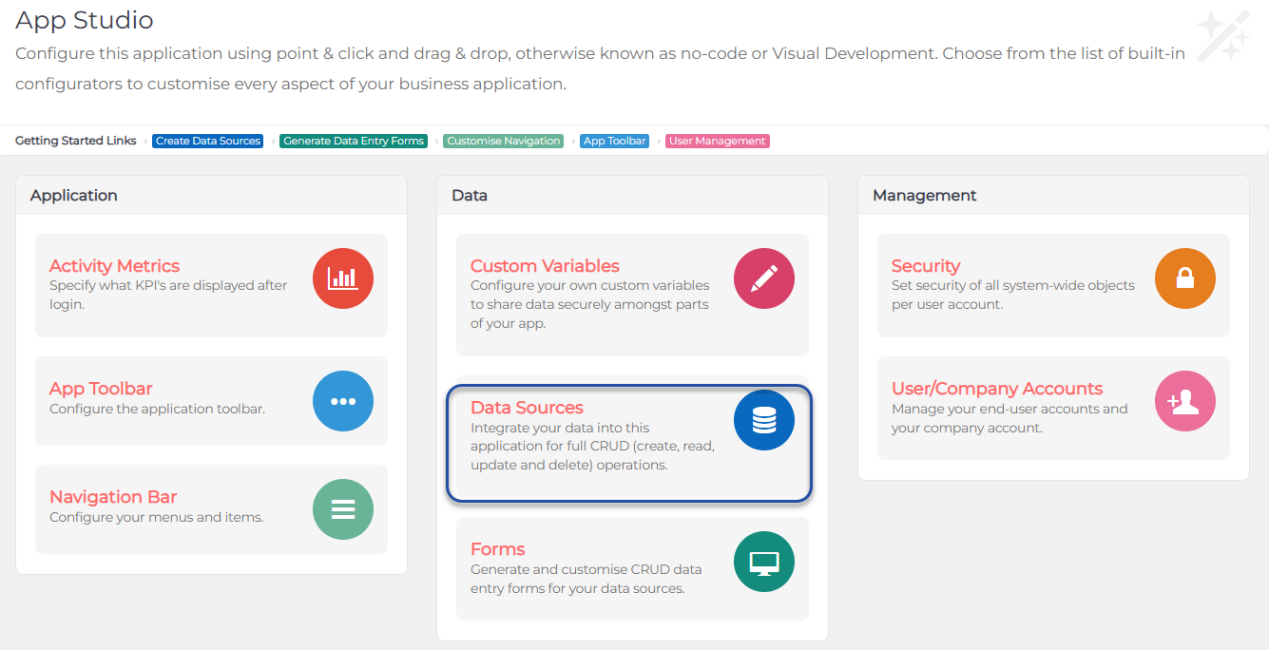
When you are testing your ReST API data source requests, the values of the custom variables will be used to replace the data in your request.

At run-time for end-users, these client-side custom variable values will change as the data is presented and manipulated in the forms.

Data Sources

Configuring the ReST API data sources

The Data Sources configurator is available from the App Studio.



This is a data-driven business application and the data is sourced from ReST API requests which provide Create, Read, Update, Delete (CRUD) end-points to pull and push data between the back-end and the front-end.

HTTP Request Methods

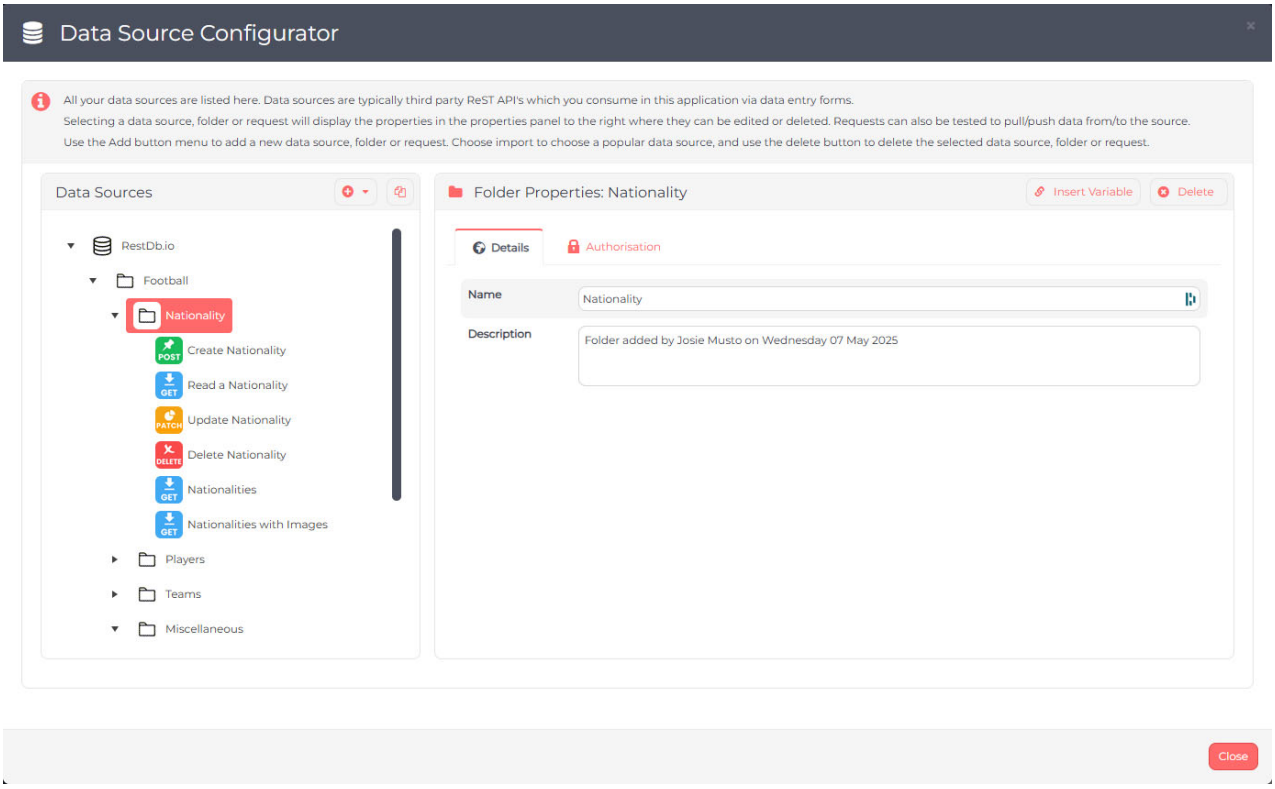
We call a ReST API endpoint, a data source request. Each request will be one of the industry standard HTTP request methods ↗ (also known as verbs). The following are supported:

Method	Description
GET	Retrieve one or more records from a server
POST	Send data to a server to create/update a record
PUT	Send data to a server to create/update a record
DELETE	Deletes the specified record on the server
PATCH	Partial modifications of a record on the server

These coloured icons are used when configuring data source requests.

Configurator Popup Form

Opening the data sources from the app studio opens a modal popup form.



This shows a hierarchical tree view on the left with properties of the selected tree node on the right. Each of these panels has buttons to manage each data source. The tree view shows data sources, folders and requests.

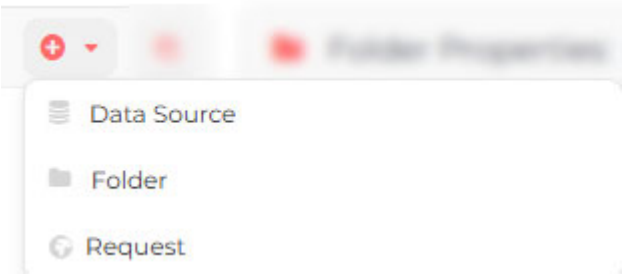
WARNING: This modal dialogue automatically saves all changes i.e. there is no Apply or Cancel button. Follow best-practice which is to make changes slowly at a controlled pace and test your changes constantly.

Data Sources Panel

This panel has an add button menu and a clone button. The tree view shows the data source request hierarchy consisting of data sources, folders, sub-folders and data source requests. The tree view nodes can be dragged and dropped into the most logical order to help you manage potentially dozens of data sources and hundreds of data source requests.

Add Button Menu

This shows a menu allowing the creation of a new data source, folder or request beneath the currently selected tree node.



Data Source

You are permitted to have multiple ReST API's from numerous different servers. It is recommended therefore to give a name to each of these for example "Accounts API" or "Product Library" to easily differentiate where your data is sourced from. This will create a master tree node where you can then create sub-folders.

Folder

Create a folder beneath either a data source, or another folder. This allows you to structure your ReST API request hierarchy into logical groups.

Request

A data source request is a ReST API endpoint which is configured to consume data for CRUD operations using the supported HTTP methods. It is advisable to relate to the CRUD concept when naming your requests e.g. "Read a Product" or "Update a Product".

Each request is available to designers when designing forms, components and fields and these people may not be familiar with ReST API concepts, so naming requests descriptively is important.

Clone Button

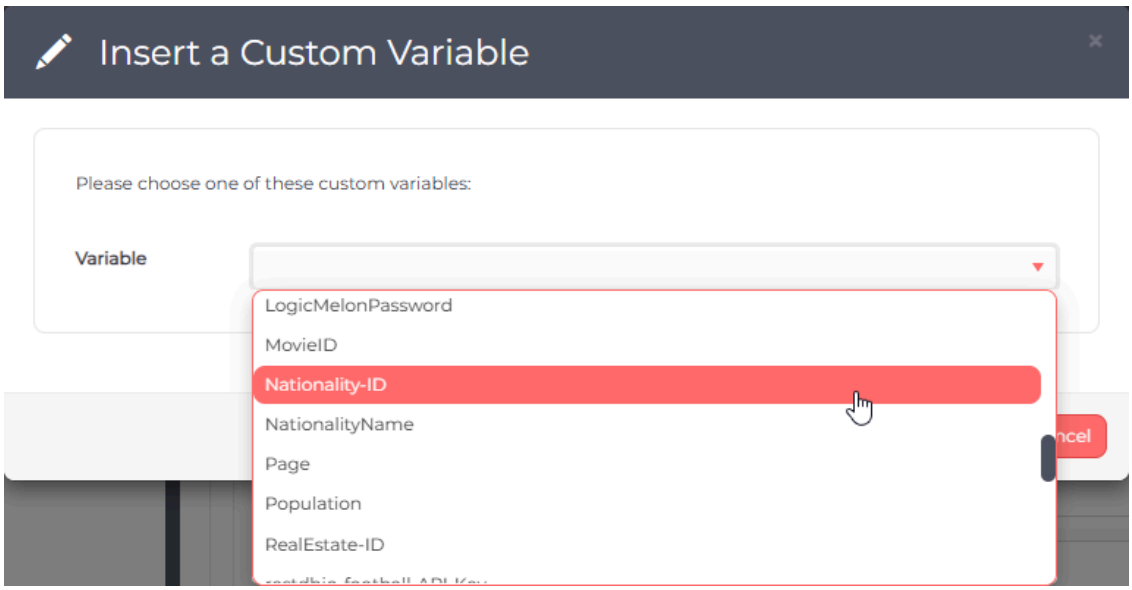
This button is used to make a copy of the selected tree view item. You are prompted to rename the cloned item before it is created beneath the selected item.

Properties Panel

The properties panel shows properties of the selected tree node.

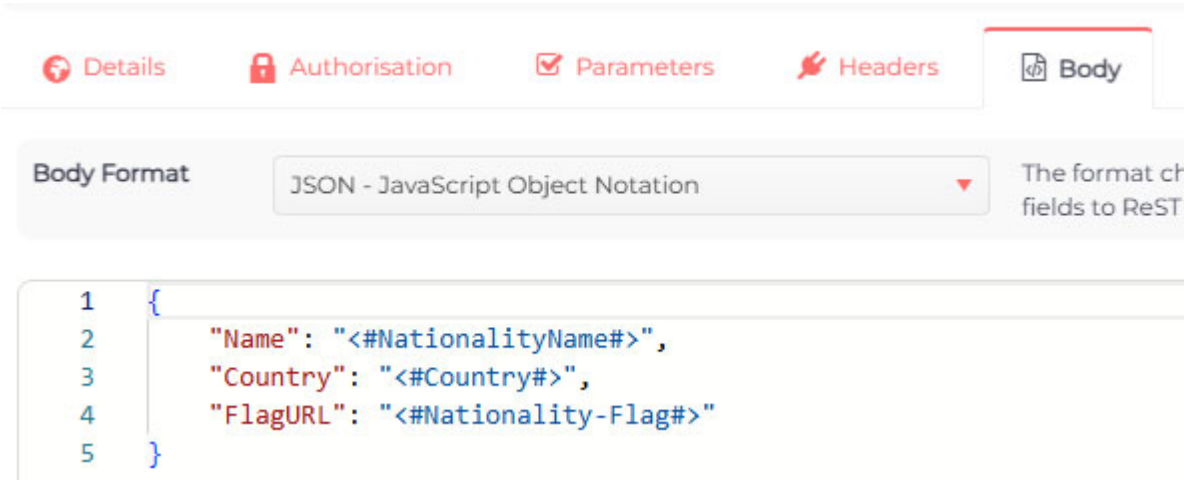
Insert Variable

This button is used to paste a selected custom variable into the data source request.



The popup dialogue has a drop down list of all custom variables. Selecting a variable will paste the variable into the request URL or Body.

The inserted variable will be shown with **<#** and **#>** field delimiters e.g.



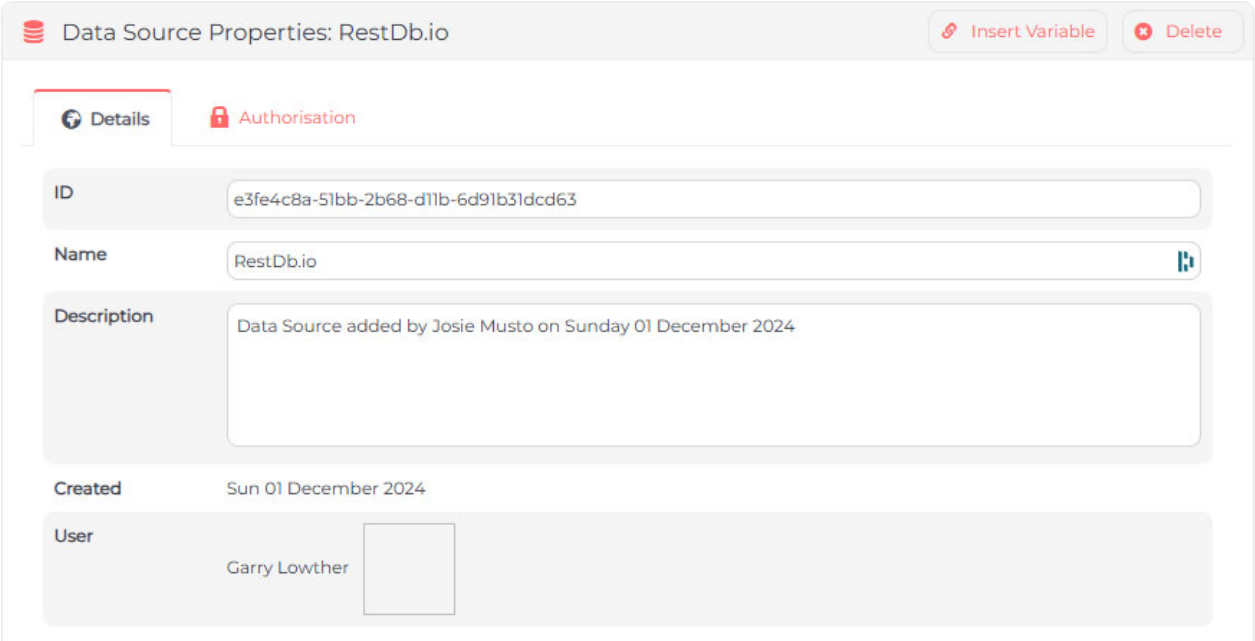
Delete

The delete button is used to delete the currently selected data source, folder or request. It is a recursive operation i.e. removing a root node will also remove all children and children of children.

WARNING: Deleting requests may adversely impact the operations of your application so take great care.

Data Source Properties

When a data source is selected in the tree view, its properties are shown on the right.



There are two tabs displayed in the data source properties.

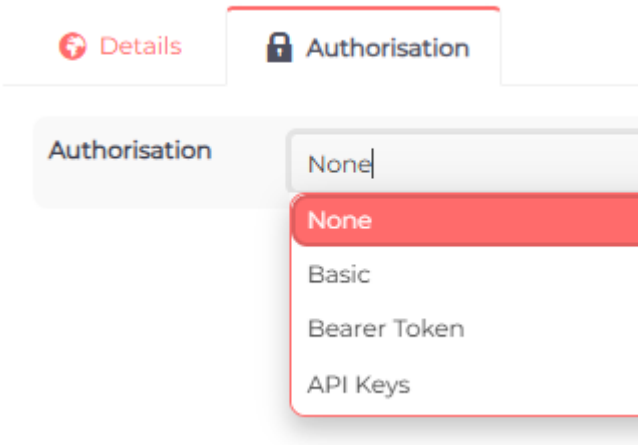
Details

These are the details of the data source.

Field	Description
ID	The read-only unique identifier of this data source. This is generated for each new data source and is only of importance to developers wishing to re-use this in their own applications.
Name	This is the name of the data source.
Description	A description about what this data source does.
Created	The date that this was created.
User	The user who created the data source.

Authorisation

These are the authorisation details associated with the data source.

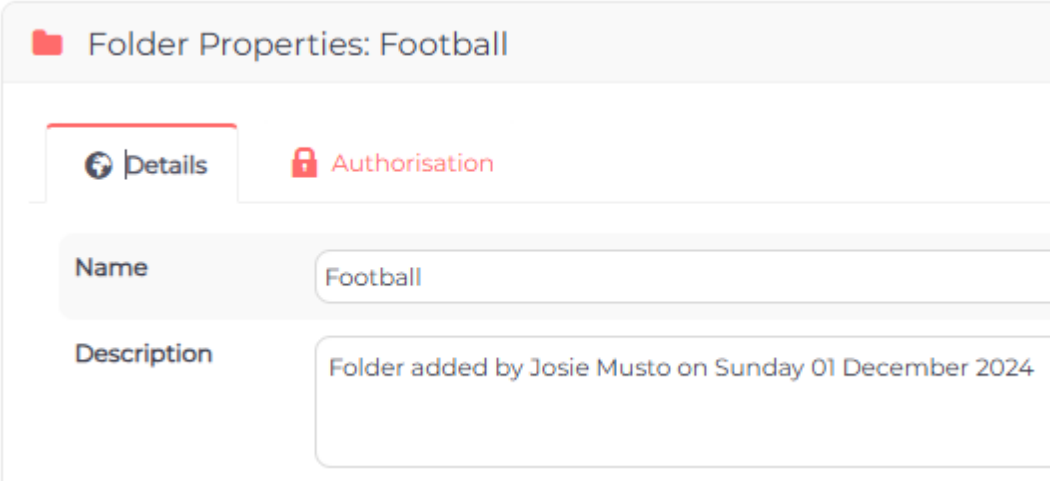


Typically there are four types of authorisation associated with a data source. All children of a data source (folders and requests) can inherit authorisation from their parent. The values of each authorisation credential should be a server-side custom variable to allow centralised control.

Type	Description
None	Either the child data requests do not need authorisation, or the authorisation is configured per folder or per request.
Basic	A username and password is specified.
Bearer Token	A token is supplied.
API Keys	One or more key/value pairs are supplied at HTTP headers.

Folder Properties

When a folder is selected in the tree view, its properties are shown on the right.



Details

These are the details of the folder.

Authorisation

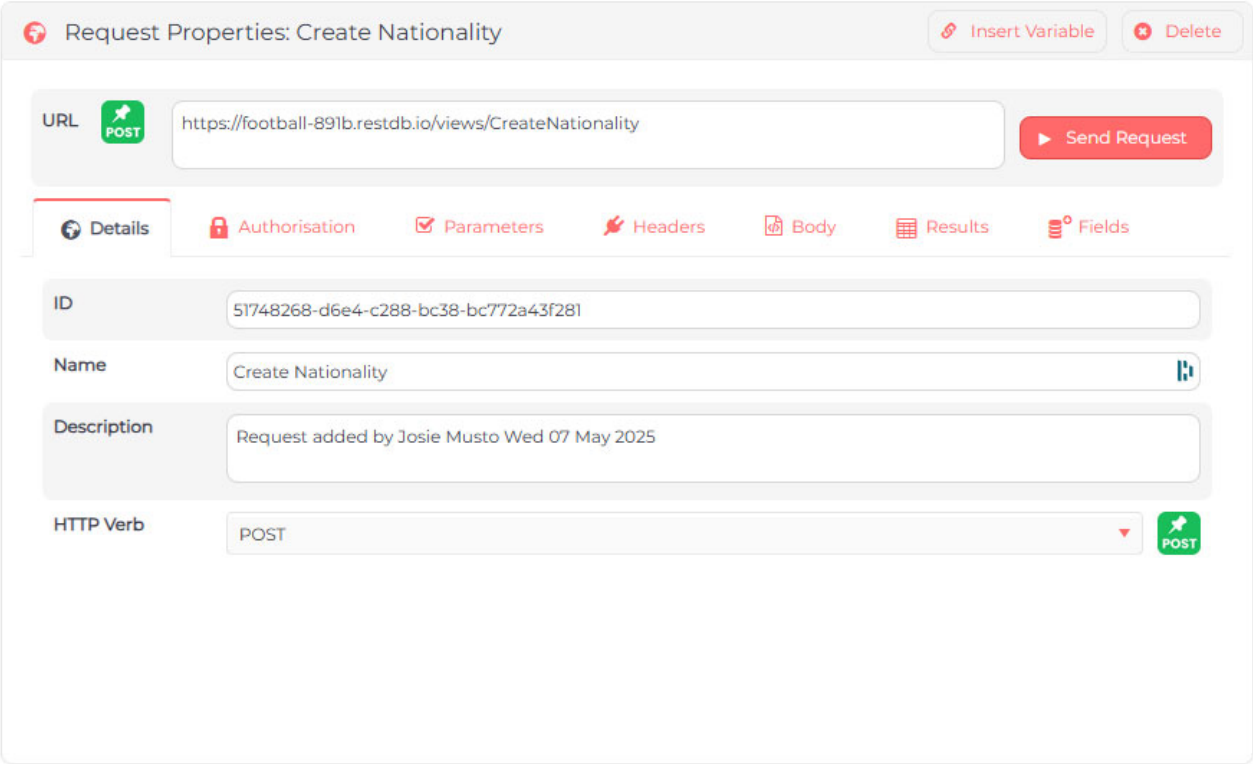
These are the authorisation details associated with the folder. The values of each authorisation credential should be a server-side custom variable to allow centralised control.

These are as follows:

Type	Description
Inherit from parent	Any authorisation credentials are inherited from the folder or data source above. This can be hierarchical so that the parents parent etc.. inherits the authorisation. This allows each set of ReST API requests to have their own centralised credentials.
None	Either the child data requests do not need authorisation, or the authorisation is configured per folder or per request.
Basic	A username and password is specified.
Bearer Token	A token is supplied.
API Keys	One or more key/value pairs are supplied as HTTP headers.

Request Properties

When a request is selected in the tree view, its properties are shown on the right.



This panel consists of the URL text box, a Send Request button, and several tabs.

URL

This uniform resource locator is the end-point of the ReST API. It should always start with https meaning that the end-point is secured with SSL.

This URL can contain parameters adhering to industry standards for example:

```
https://mydomain.com?product=switch&colour=white&material=plastic
```

In the example, the product parameter value is 'switch', the colour is 'white' and the material is 'plastic'.

It is most important when designing data source request URL's to use custom variables to define these parameters either on the URL itself, or using the Parameters tab.

This is an example of a URL used in the sample requests which deletes a nationality using a custom variable:

```
https://football-891b.restdb.io/views/DeleteNationality?ID=
<#Nationality-ID#>
```

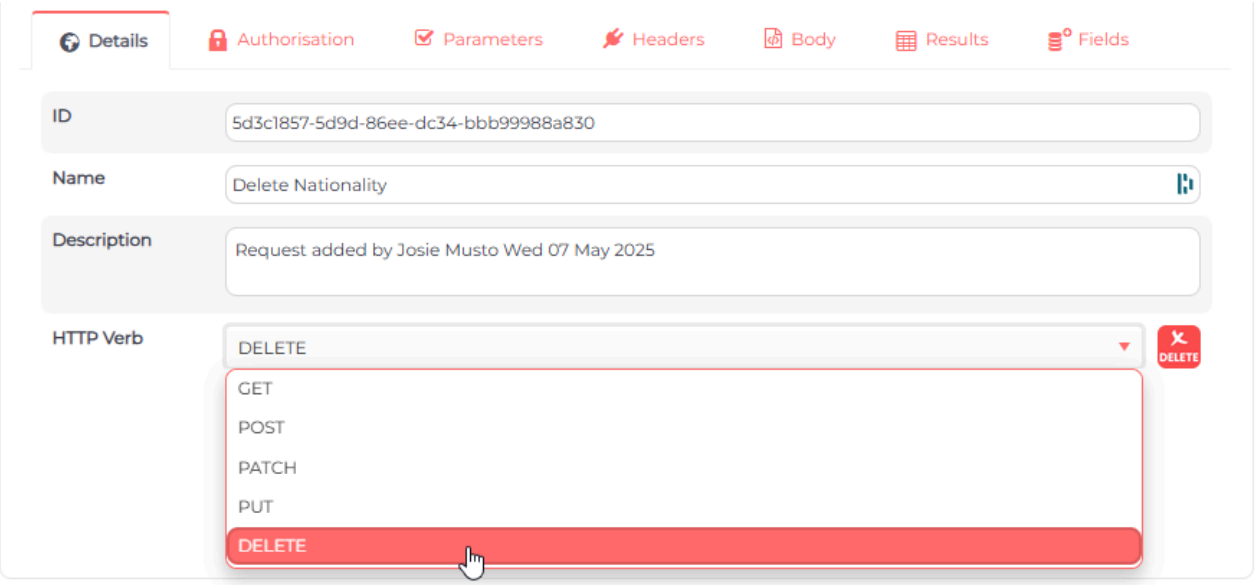
The `<#Nationality-ID#>` custom variable was inserted into the URL using the Insert Variable button. It is strongly recommended to use this mechanism rather than trying to remember and type custom variable names.

Send Request

This button sends the ReST API request to the back-end and presents the data in the Results tab. IT also populates the fields in the Fields tab if they do not already exist.

Details

This tab has the usual properties plus the important HTTP Verb/Method.



The properties are as follows:

Property	Description
ID	The read-only unique identifier of this data source request. This is generated for each new data source request and is only of importance to developers wishing to re-use this in their own applications.
Description	A description about what this data source request does.
HTTP Verb	A drop down combo of all available HTTP methods.

Authorisation

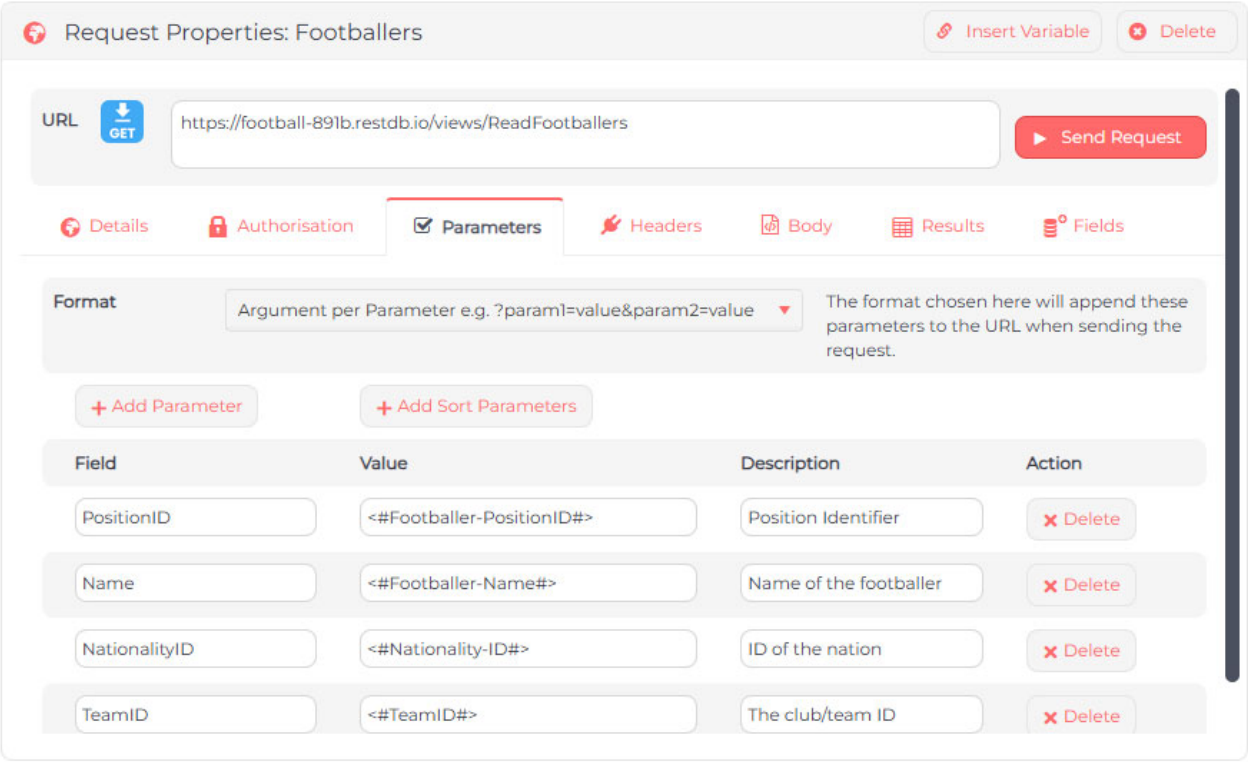
This tab allows the authorisation credentials to be assigned for the ReST API data source request.

Type	Description
Inherit from parent	Any authorisation credentials are inherited from the folder or data source above. This can be hierarchical so that the parents parent etc.. inherits the authorisation. This allows each set of ReST API requests to have their own centralised credentials.
None	Either the child data requests do not need authorisation, or the authorisation is configured per folder or per request.
Basic	A username and password is specified.
Bearer Token	A token is supplied.
API Keys	One or more key/value pairs are supplied as HTTP headers.

Parameters

The parameters tab should be used when the ReST API end-point has multiple discretionary parameters i.e. one or more parameters can be used to filter the data set.

This is an example from the sample data where this end-point is used in a search component in which the end-user can provide zero or more filters:

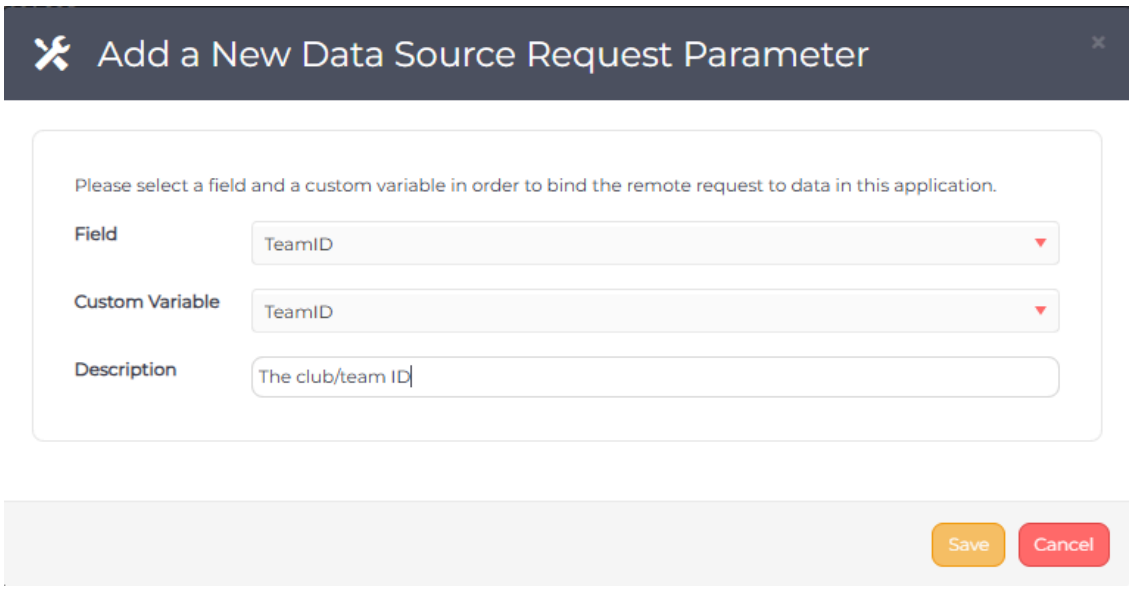


The parameter format can be chosen from the drop-down Format field. Typically this will be of this form:

```
Argument per Parameter e.g. ?param1=value&param2=value
```

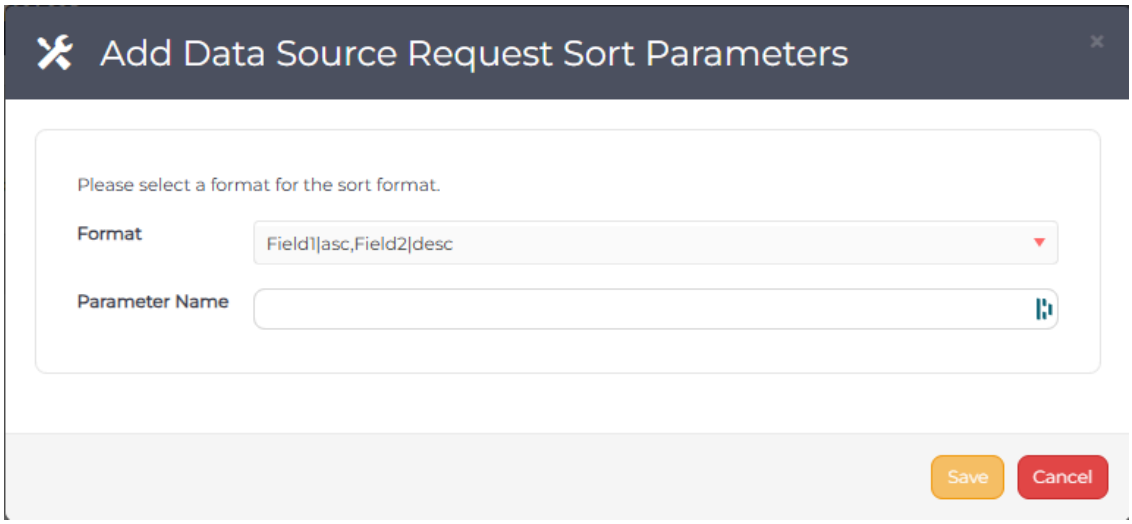
This will add the parameters to the URL in this format.

The **Add Parameter** button opens a modal popup allowing the fields in the ReST API data source request to map to custom variables.



Each parameter added is shown in the list of fields beneath the button.

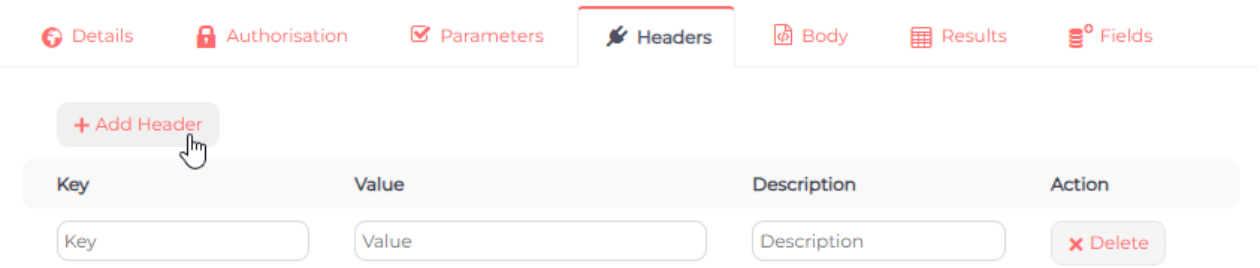
The Add Sort Parameter button opens a modal popup dialogue to select the format of the sort parameters and the name of the parameter.



These sort parameters are also appended to the URL instructing the ReST API to sort the data by a specific field. For example if the endpoint returns products, you may wish to sort these by product name, then category.

Headers

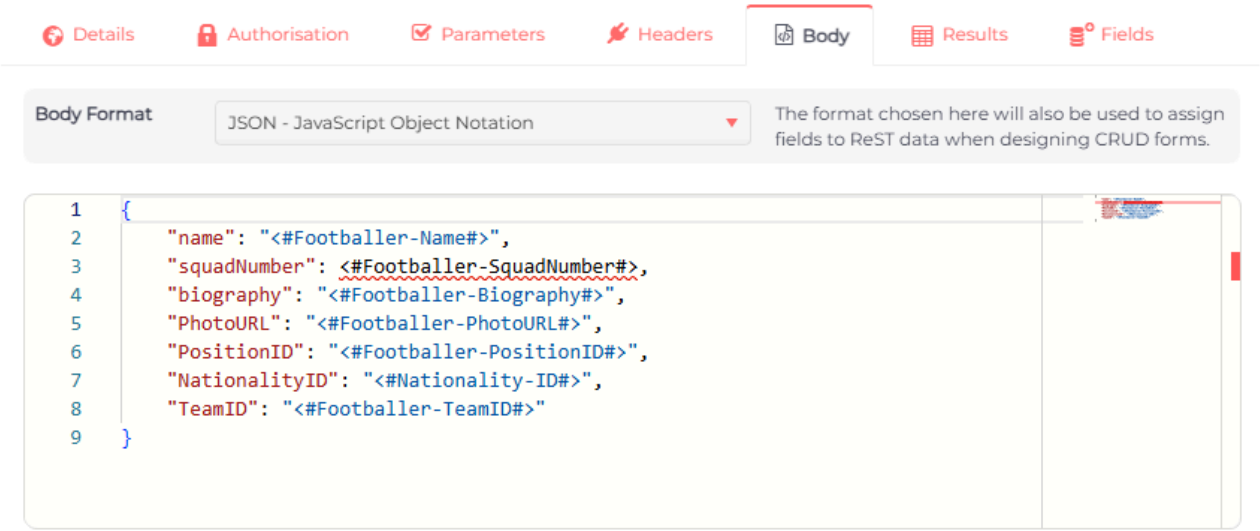
The headers tab is used where this specific data source request needs additional headers other than authorisation to control the data which is returned or updated.



The **Add Header** button adds a new row to the list. The key can be typed in and the value field should be inserted from the Insert Variable button. Header rows can be removed using the **Delete** button.

Body

The body tab is used for sending data from the application to the ReST API usually to create or update records.



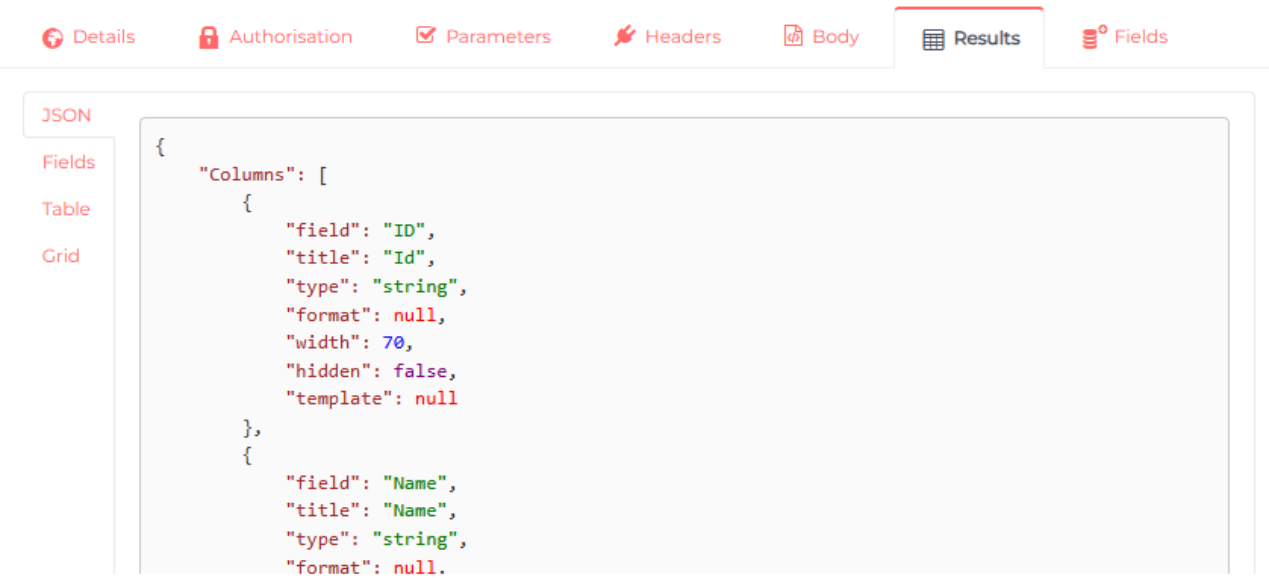
The **Body Format** field provides two options:

Format	Description
JSON - JavaScript Object Notation	This is the recommended option as JSON is easy to read regardless of your technical expertise.
Grid - two dimensional table with columns and rows	Data is represented by rows and columns but without the clarity of JSON.

The values for each field should be inserted using the Insert Variable button as nothing should be hard-coded. The JSON body format shows a thumbnail of the body to the right which is useful when dozens of fields are being specified.

Results

This tab displays the results of the ReST API when the **Send Request** button is pressed.



The Results tab has 4 left docked sub-tabs.

The JSON sub-tab shows the raw JSON data received from the ReST API request. This may show field and data e.g.

```
{
                "ID": "6863f56e78badf6500137ff3",
                "Name": "Ana Crnogorčević",
                "PhotoUrl": "https://xyz.restdb.io/media/12345-xyz-abc",
                "PhotoID": "12345-xyz-abc-image",
                "TeamName": "Cambridge United",
```

The Fields sub-tab will show a list of fields returned from the ReST API request.

JSON
Fields
Table
Grid

Field	Type	Values	Visible	Caption
Biography	String	1 values: https://www.arsenal.com/women/players/alessia-russo	true	Biography
ID	String	1 values: 6863ef8078badf6500137ef6	true	ID
Name	String	1 values: Alessia Russo	true	Name
NationalityFlagID	String	1 values: 67fa336b78badf6500054bea	true	NationalityFlagID
NationalityFlagUrl	String	1 values: https://football-891b.restdb.io/media/67fa336b78badf6500054bea	true	NationalityFlagUrl
NationalityID	String	1 values: 67f7cb8278badf650004fb6b	true	NationalityID
NationalityName	String	1 values: English	true	NationalityName

The Table sub-tab will show a table of data returned from the ReST API request.

JSON
Fields
Table
Grid

ID	Name	PhotoUrl	PhotoID	TeamName	Team
6863ef8078badf6500137ef6	Alessia Russo		6863ef8078badf6500137ef5	Arsenal	674d
681de1ae78badf65000babb4	Alex Morgan		681de1ad78badf65000babb3	Wolverhampton Wanderers	6759

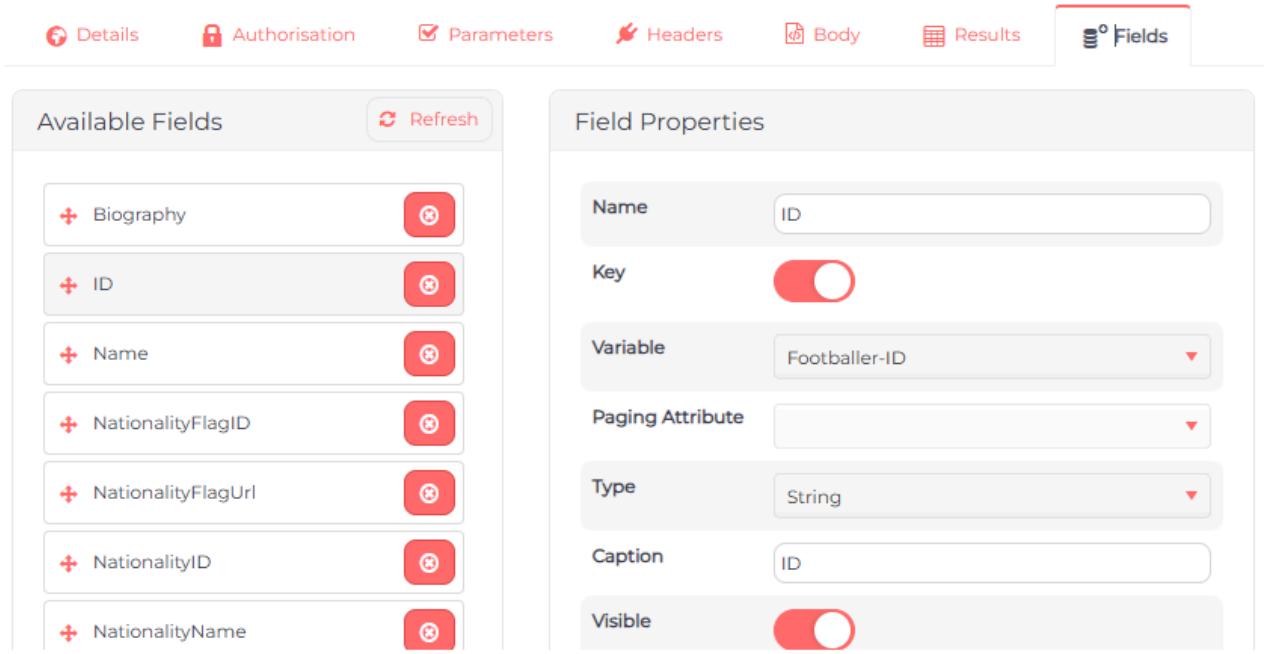
The Grid sub-tab will show a data grid of the data returned from the ReST API request. This closely resembles how the data may look to the end-user when you hook this request up to forms.

JSON
Fields
Table
Grid

ID	Name	PhotoUrl	PhotoID	TeamName	
6863ef8078badf65...	Alessia Russo	https://football-891b.restdb.io/...	6863ef8078ba...	Arsenal	Open
681de1ae78badf65...	Alex Morgan	https://football-891b.restdb.io/...	681de1ad78ba...	Wolverhampton Wanderers	Open
686f8e4778badf65...	Alex Scott	https://football-891b.restdb.io/...	686f8e4778ba...	Bournemouth	Open

Fields

This tab is where the fields received from the ReST API endpoint are listed when the Send Request button is invoked.



There are two panels in a master/detail configuration. Selecting a field in the **Available Fields** panel will show the field in the **Field Properties** panel.

The **Available Fields** panel lists all of the fields in the original order returned by the ReST API. These can be re-arranged by dragging and dropping fields using the left side drag icon. The right side delete button can be used to remove the field from the list. If the ReST API is under development and starts returning different fields, or indeed if you have deleted the wrong field, then the **Refresh** button refreshed the list of fields. Your specific field configuration will be lost in this instance though.

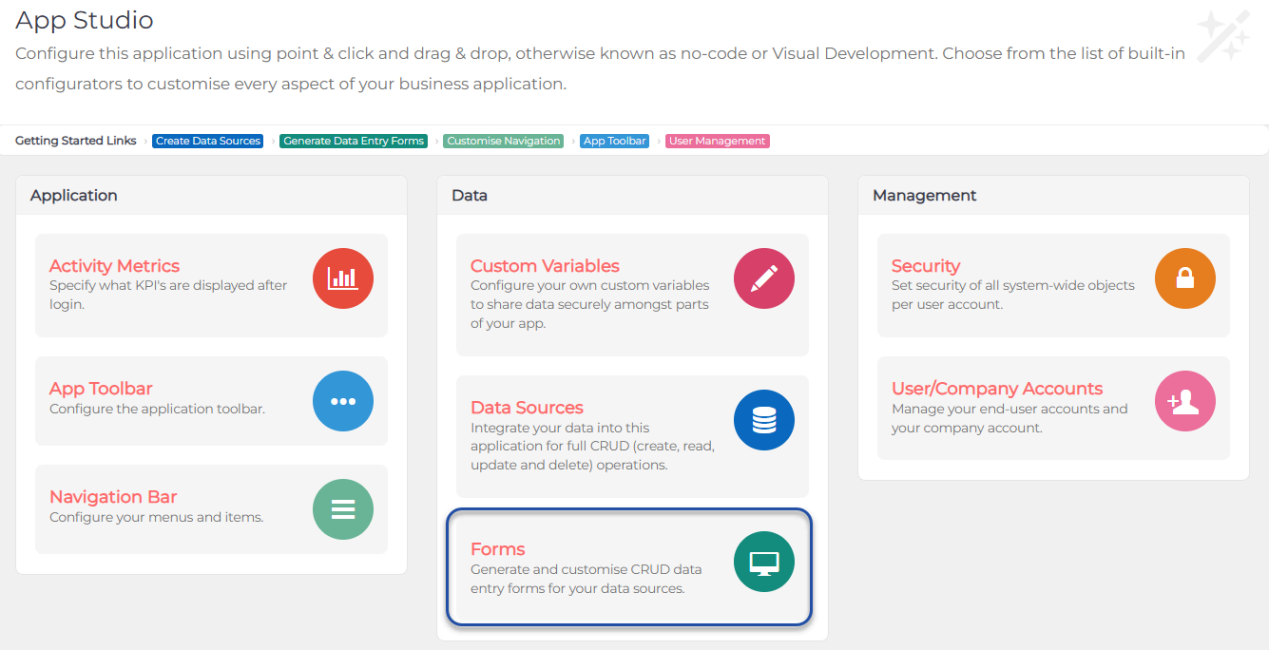
The field properties are as follows:

Property	Description
Name	The name of the column/field in the ReST API
Key	Whether this is a key field i.e. the product ID would uniquely identify a product, however a post code would not uniqiely identify an address.
Variable	This is a drop-down of all custom variables, one of which can be associated with this field.
Paging Attribute	This is a drop-down of four paging attributes used when this ReST API is paginated i.e. it returns a page of data at a time, rather than the entire, potentially very large data set: Page Number, Records per Page, Total Page Count, Total Record Count
Type	The type of field is often a string or a number, however all data types are supported including Image URL and Base64 which can show image data types.
Caption	This is the human readable text of the field when displayed in forms or grids.
Visible	Whether the field is visible in consuming forms or components.
Sample Values	A list of sample data values read from the ReST API

Forms

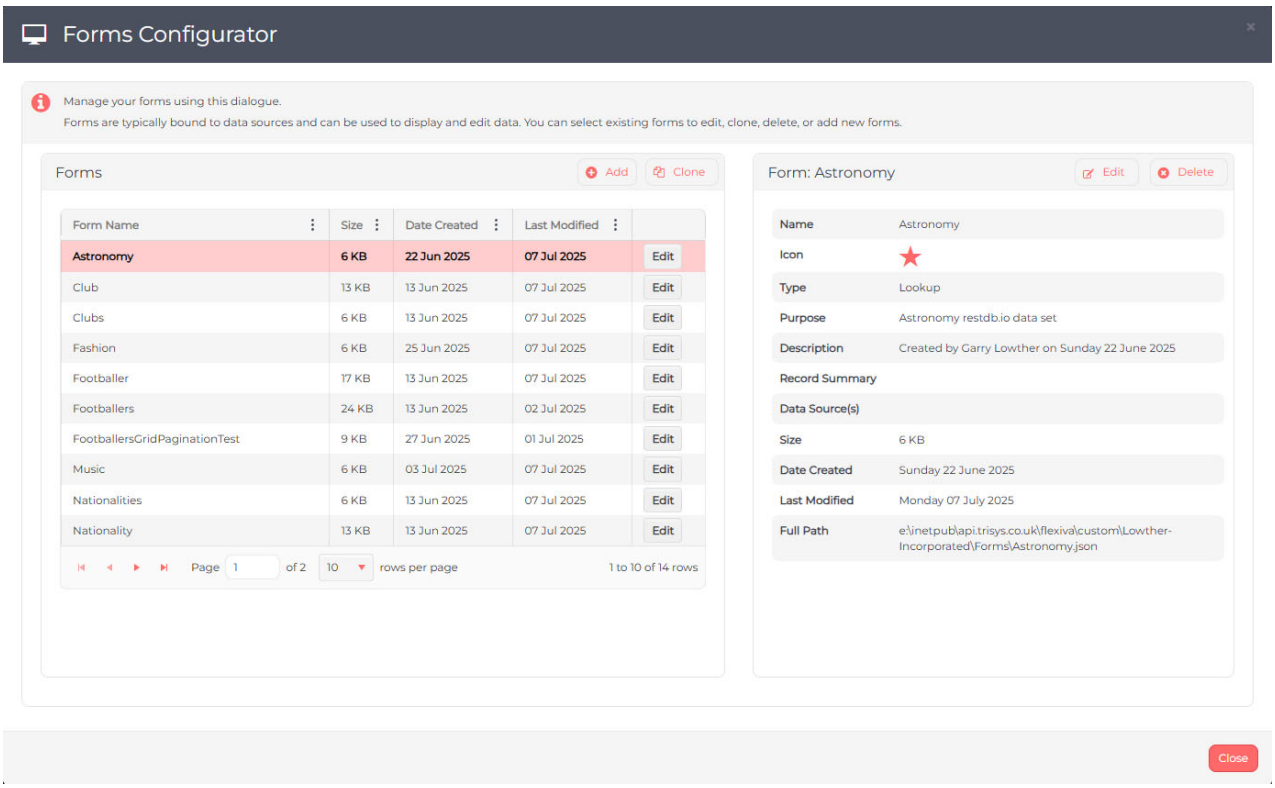
Configuring the lookup and data entry forms.

The Forms configurator is available from the App Studio.



Forms Configurator

The forms configurator is a modal popup dialogue which shows all of the forms in the left panel and the properties of each form in the right panel. It operates as master/detail so that selecting a form in the left panel grid, shows the associated form properties to the right.

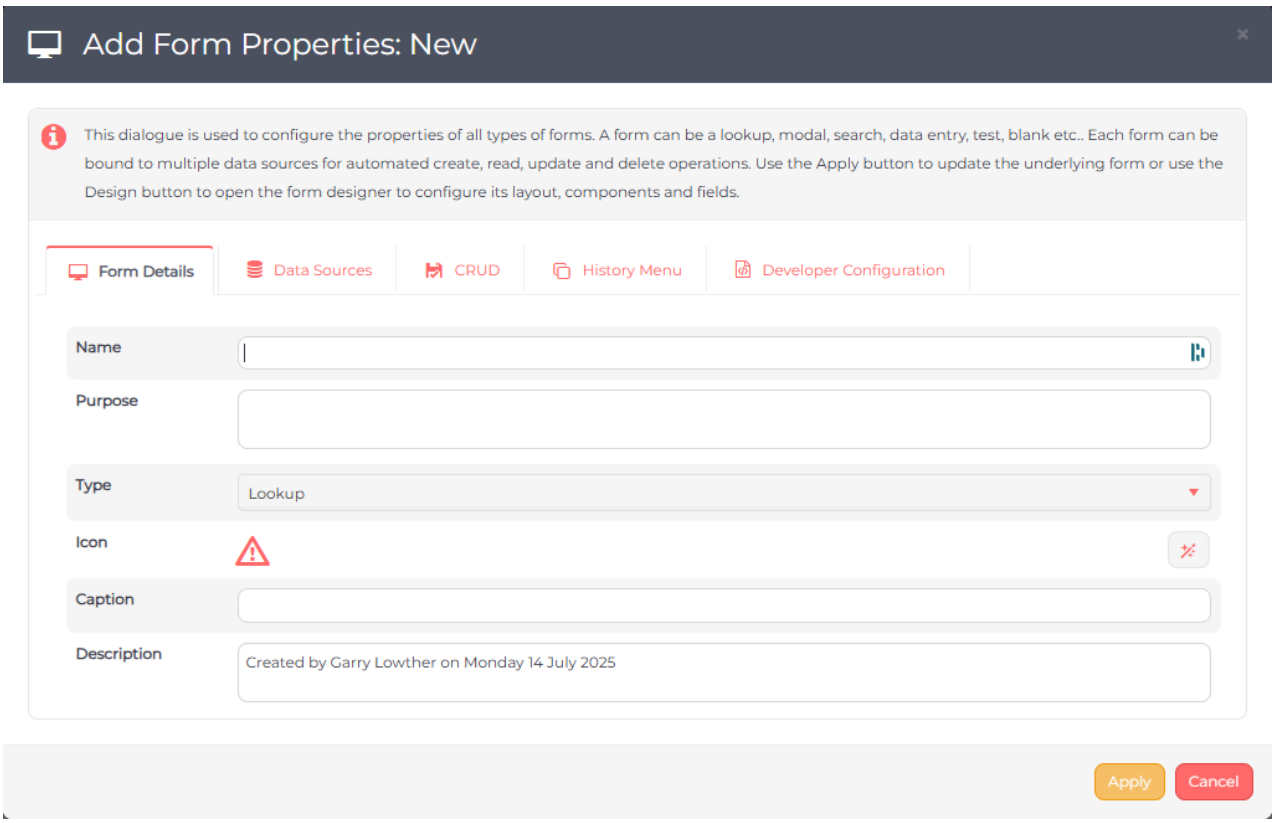


Forms Panel

This panel has two buttons and a data grid.

Add

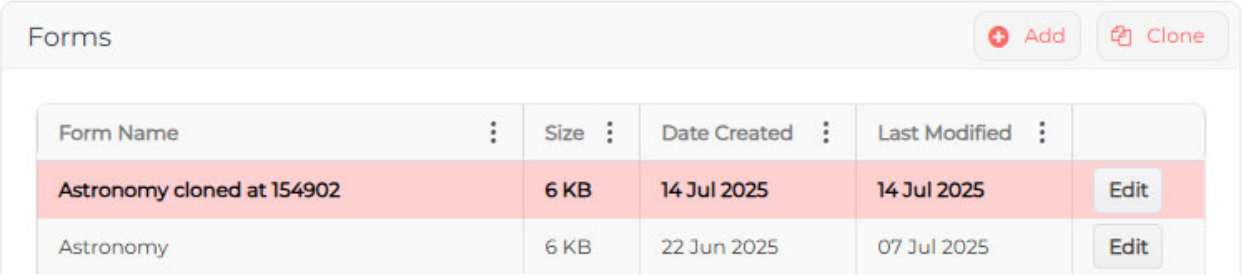
This button is used to add a new form. It opens the following modal popup:



The process of editing new forms is the same as that for existing forms.

Clone

This button is used to take a copy of the selected form, and immediately saves it with a new name:



Use the Edit button in the form panel to change the name and design the new form.

Forms Grid

This is the data grid showing all existing forms showing their name, size, data created and last modified. The Edit button opens the form properties modal popup. The grid operates as a master/detail where each form selected, shows its properties in the form panel.

Form Panel

This panel has two buttons and a list of selected form properties.

Edit

The edit button opens the editor for the form, allowing it's properties to be changed. Note that editing the form is not the same as designing the form. Editing the form properties is focused on accessibility and how it is managed by the application, whereas the form design process focuses on the specific fields, tabs, components, layout and functionality of the form itself.

Delete

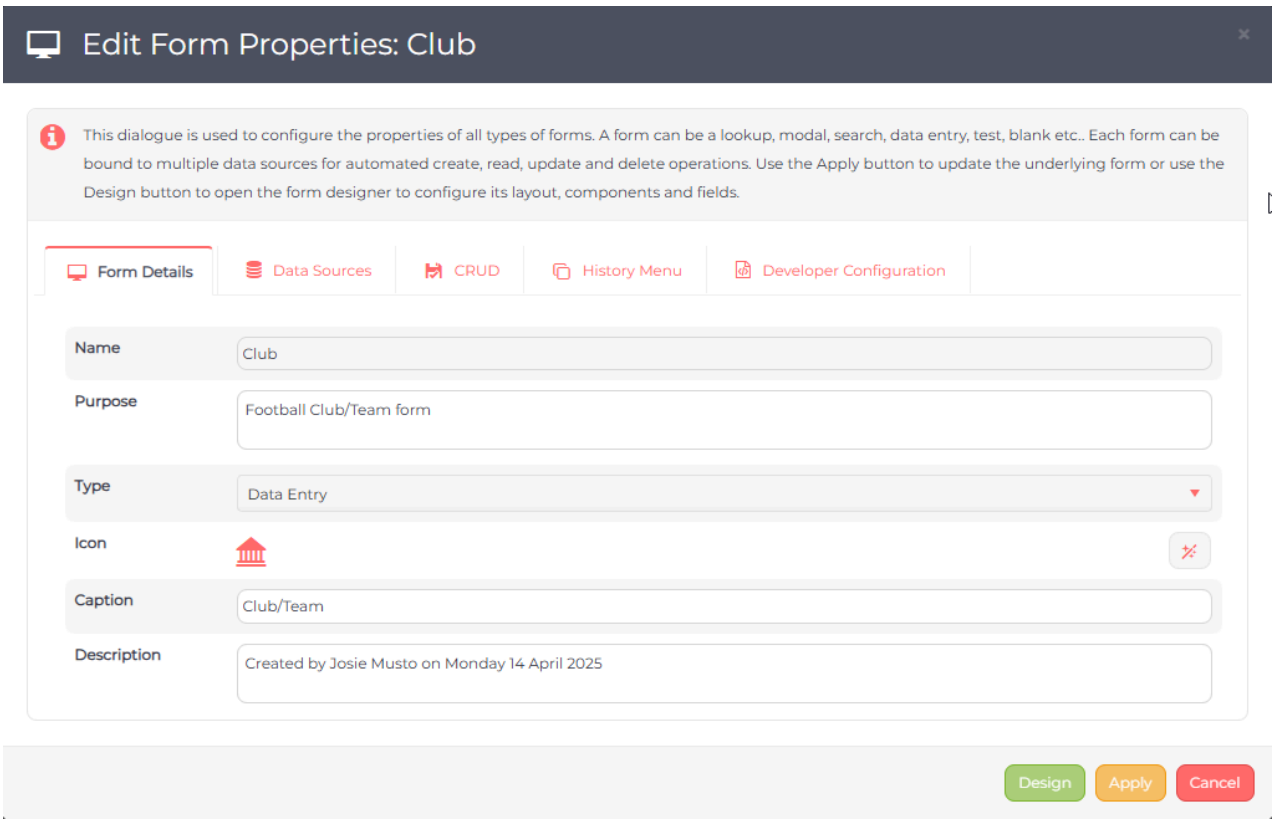
The delete button removes the selected form. Any navigation bar links, activity metrics links or history menu items will cease to function if forms are removed.

The form properties are as follows:

Field	Description
Name	This is the unique name of the form.
Icon	This is the icon used when the form is referenced in menus and drop downs.
Type	The type of form is either a lookup to display data sets, or a data entry form to allow editing of a master record.
Purpose	This describes what the purpose of the form is.
Description	A description about why the form exists and any additional information.
Record Summary	When the record was opened by the end-user, it appears in the History menu. If a record summary is specified, then a string identifying the record is displayed instead of the form name. For example "Guinness" may be displayed if the form name is "Drink" and the record name was used instead.
Data Source(s)	For data entry forms, typically 4 data sources for each of the CRUD methods will be specified.
Date Created	The date that this form was created.
Last Modified	The date that this form was last modified.
Full Path	The internal path to the form on the server. This is only of interest to developers.

Editing Forms

When adding or cloning a new form, or editing one from the grid, or using the Edit button on the form properties panel, this modal popup form is displayed.



There are three buttons at the bottom right of this modal popup dialogue.

Button	Action
Design	Switch into form designer mode after closing the popup.
Apply	Save all changes and close the popup.
Cancel	Cancel all changes and close the popup.

The form has five tabs.

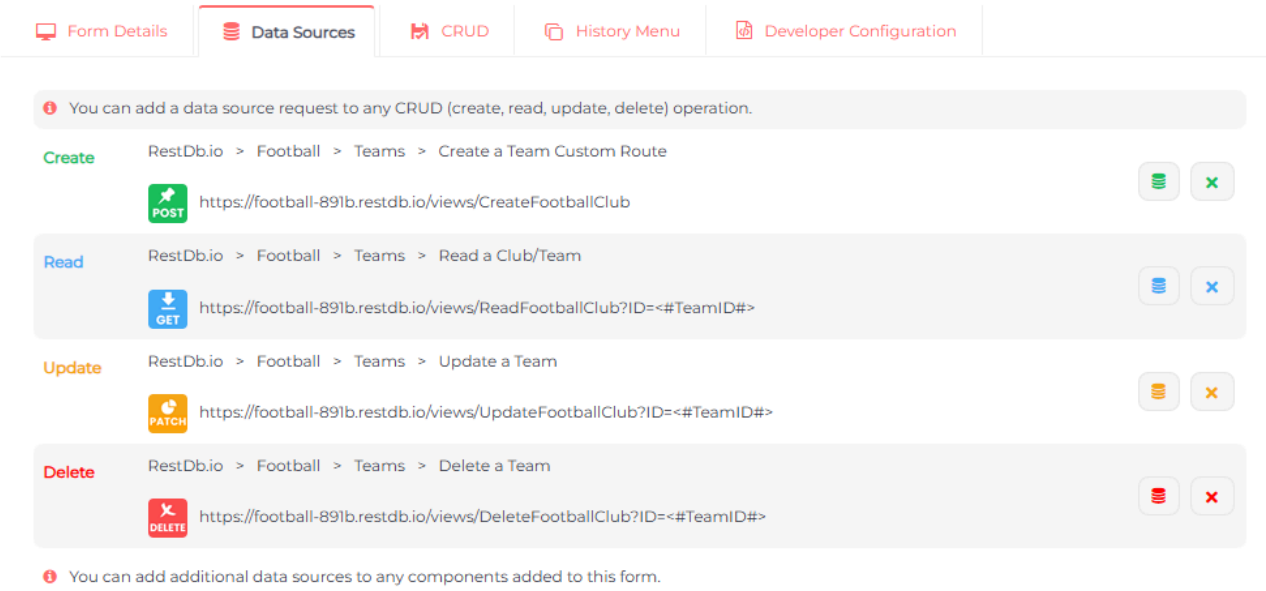
Form Details

This tab shows the basic details of the form.

Field	Description
Name	This is the unique name of the form.
Purpose	This describes what the purpose of the form is.
Type	The type of form is either a lookup to display data sets, or a data entry form to allow editing of a master record.
Icon	This is the icon used when the form is referenced in menus and drop downs.
Caption	The text which will be displayed when the form is referenced.
Description	A description about why the form exists and any additional information.

Data Sources

This tab is where multiple ReST API data source requests can be linked to provide CRUD capabilities to the data entry form.



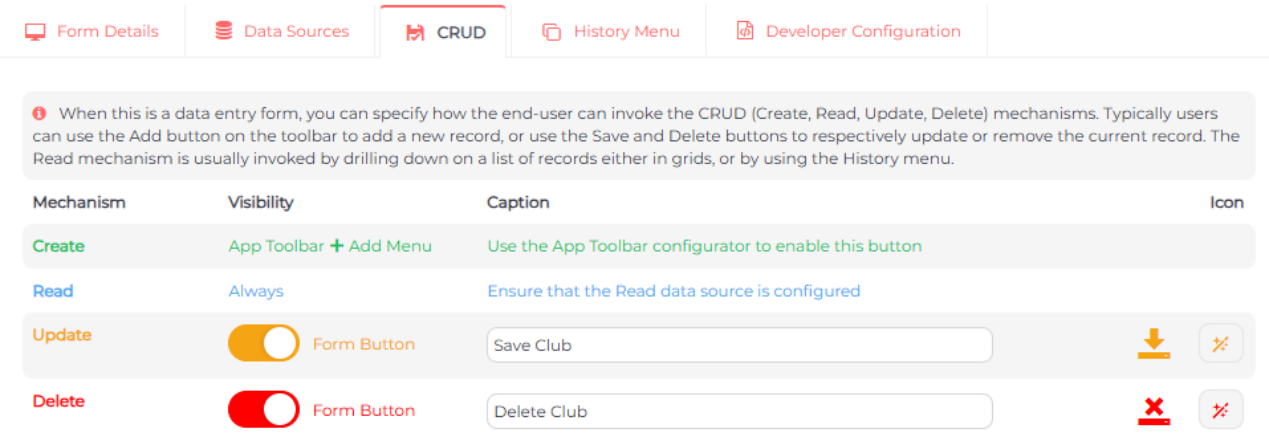
As shown in this example, this form has four linked CRUD data source requests to respectively create, read, update and delete a data record.

The two buttons to the right of each request allows for requests to be linked or unlinked.

For this automated CRUD to work, form fields and components on the respective form design need to be mapped to custom variables and data source requests.

CRUD

This tab specifies how the form data is created, read, updated and deleted.



The mechanism to create a new record is via the Add menu on the app toolbar. Use this configurator to enabled this.

In order to read a data record into the data entry form, a Read data source request must be configured on the Data Sources tab.

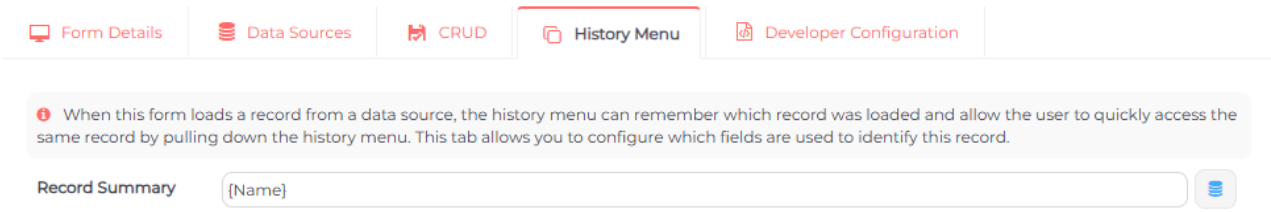
To enable updating, there is a button located top right of the data entry form. You can enable this, set its caption and its icon on this tab. The Update data source request must be configured on the Data Sources tab.

To enable deleting, there is a button top right of the data entry form. You can enable this, set it's caption and its icon on this tab. The Delete data source request must be configured on the Data Sources tab.

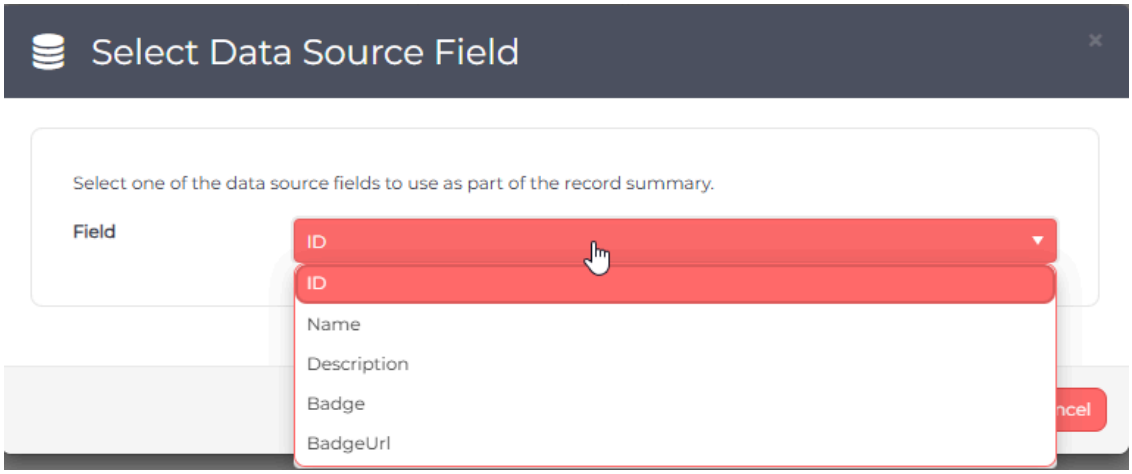
The visibility and enabled status of these form buttons can also be configured on a per user basis on the user security configurator.

History Menu

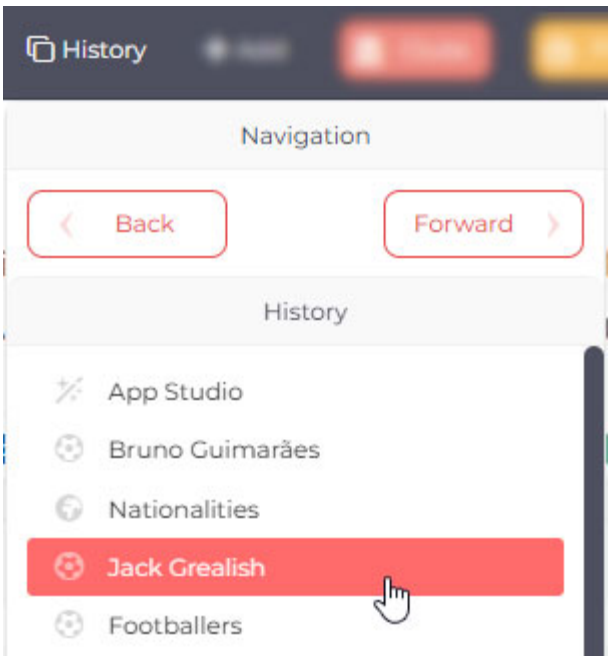
This tab specifies how the history menu displays data associated with this form when it was loaded. Typically end-users want to see the actual data in the history menu, not the name of the form they last opened.



Use the database button to the right of the Record Summary field to choose which of the form fields is used to identify the record in the History Menu.



Multiple fields can be appended together to show the record details in the History menu:



Developer Configuration

This tab is only for use by developers who wish to implement highly bespoke custom modifications to the form. Whilst the Flexiva application is 100% no-code, we make provision for a low-code approach in cases where custom business logic needs to be implemented.

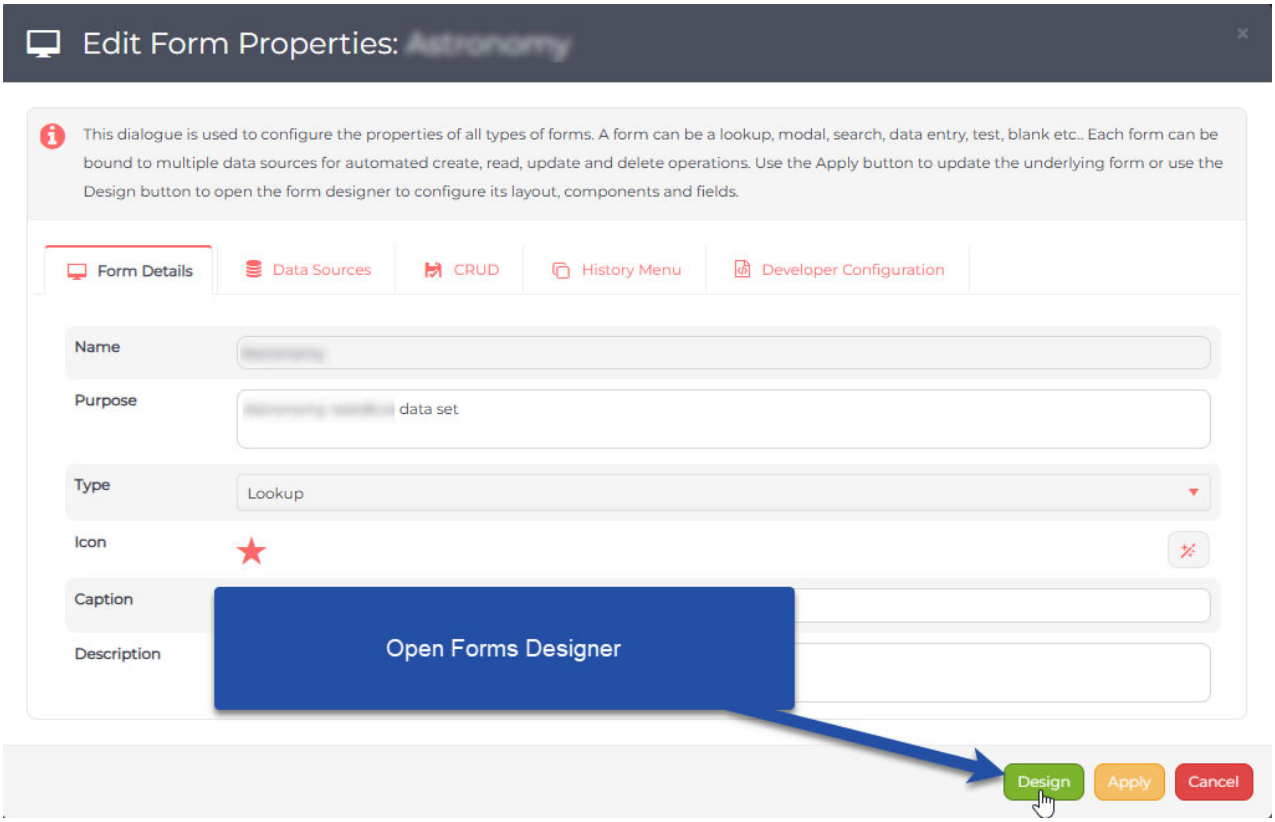
Forms Invocation

Note that in order for forms to be opened at run-time, then it must be added to the navigation bar as this controls the opening of forms. Forms can be opened either from the nav bar, history menu, add menu or by drilling down into grid records.

Form Designer

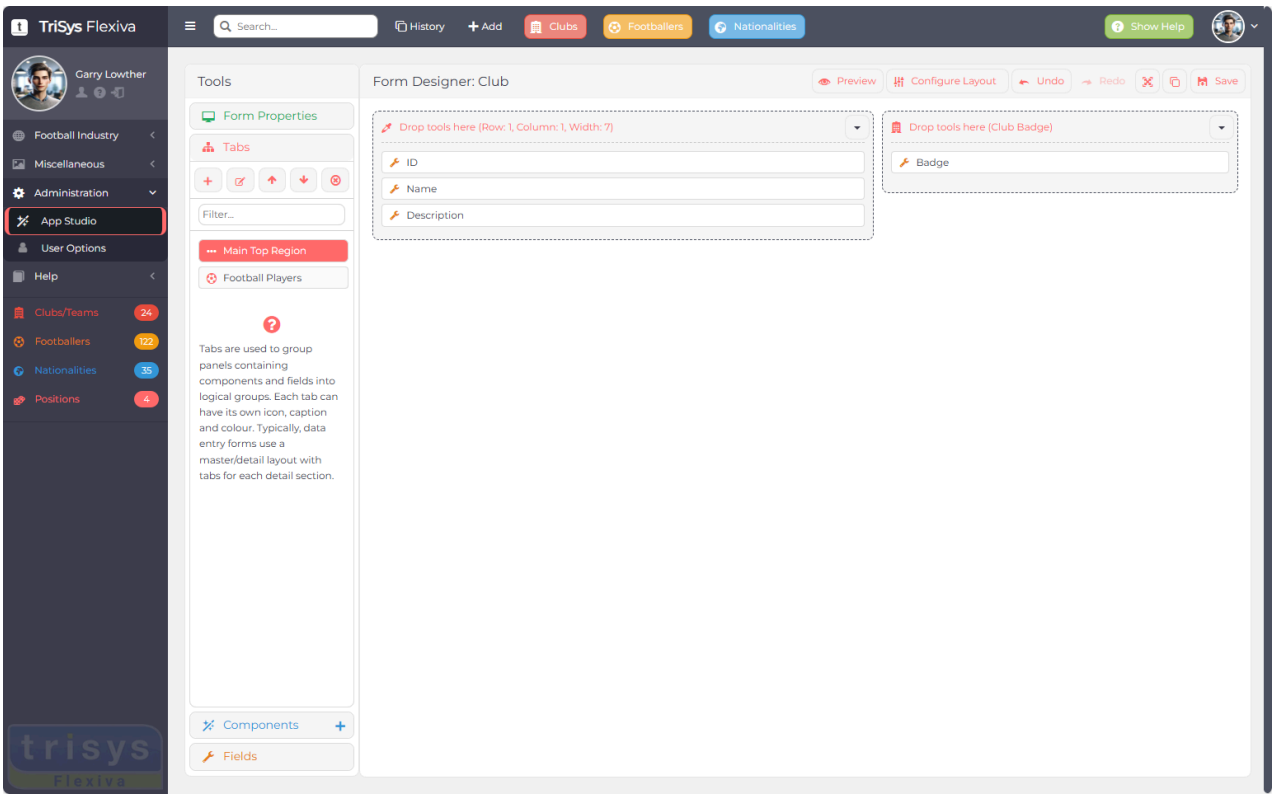
Designing your lookup and data entry forms.

The Forms designer is available when editing a form.



When this button is clicked, the form properties modal popup is closed before the form designer loads. This is to allow the forms designer to utilise the maximum amount of screen space available.

This is an example of how the forms designer looks when opened.



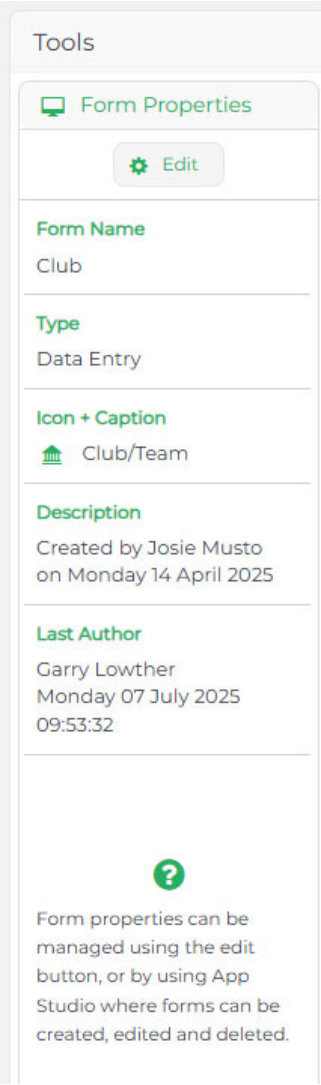
There are two main panels, the Tools panel and the Form Designer panel.

Tools

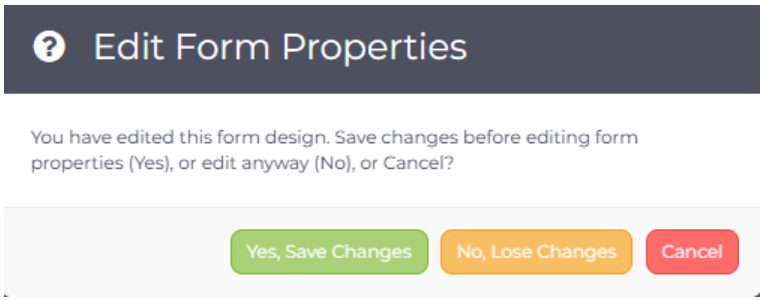
The tools panel contains four sub-panels, Form Properties, Tabs, Components and Fields.

Form Properties

This panel shows the details of the form previously specified.

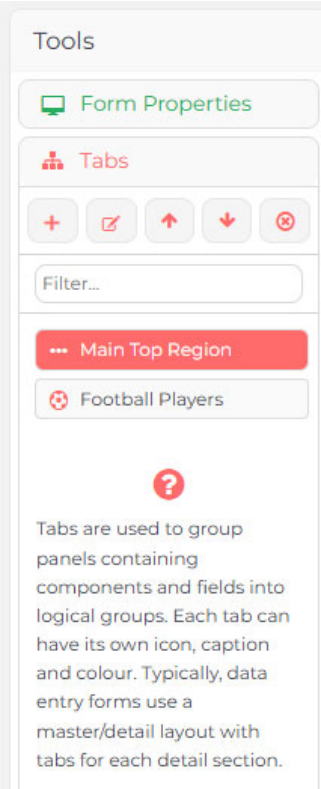


Clicking the Edit button will close the forms designer and re-open the form properties dialogue if you choose Yes or No to this prompt:



Tabs

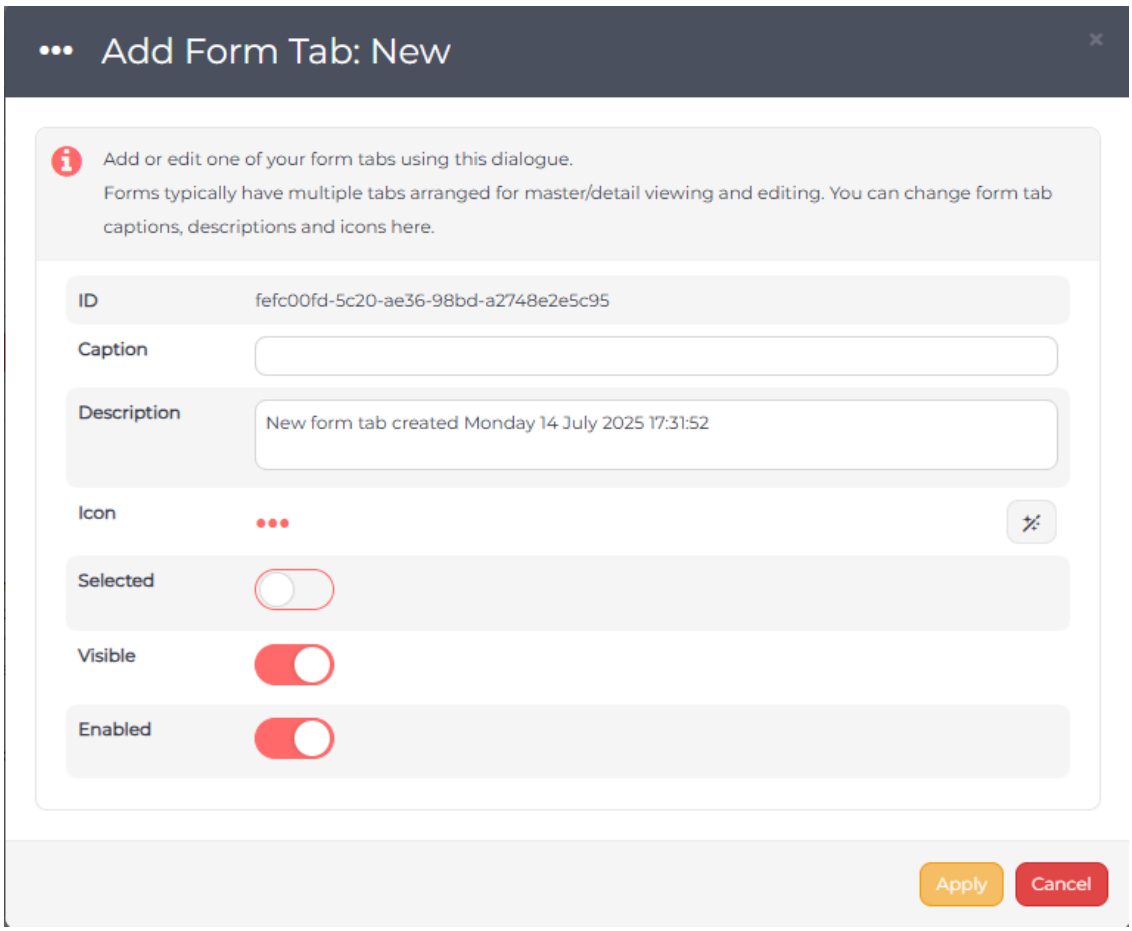
The tabs panel is for designing form tabs. Each data entry form is designed as a master/detail form where the master record details are shown at the top, and other tabs below group data, including related data grids.



The top region is know as the "Main Top Region" and is the first tab shown. All other tabs appear beneath. The Tabs toolbar has the following functions:

Add

Add a tab below that currently selected by opening up this modal popup.



Field	Description
ID	The read-only unique identifier of this form tab. This is generated for each new tab and is only of importance to developers wishing to re-use this in their own applications.
Caption	What text will be displayed in the form tab.
Icon	The icon which will be displayed to the left of the form tab.
Selected	If this form tab is selected on open.
Visible	If this form tab is visible i.e. can be seen.
Enabled	If this form tab is enabled i.e. can be clicked.

Update

Update the selected tab details by showing this same popup.

Move Up

Move the selected tab up the display order.

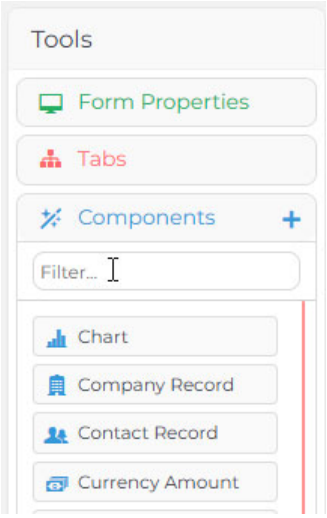
Move Down

Move the selected tab down the display order.

Delete

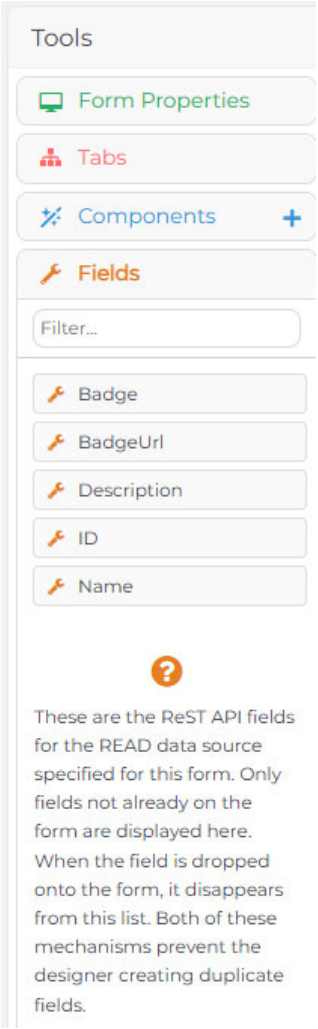
Delete the selected tab, including all panels, fields and components that exist on its design surface.

Components



The components panel shows a list of all available components which can be dragged onto the form. A filter text box allows matching components to be displayed.

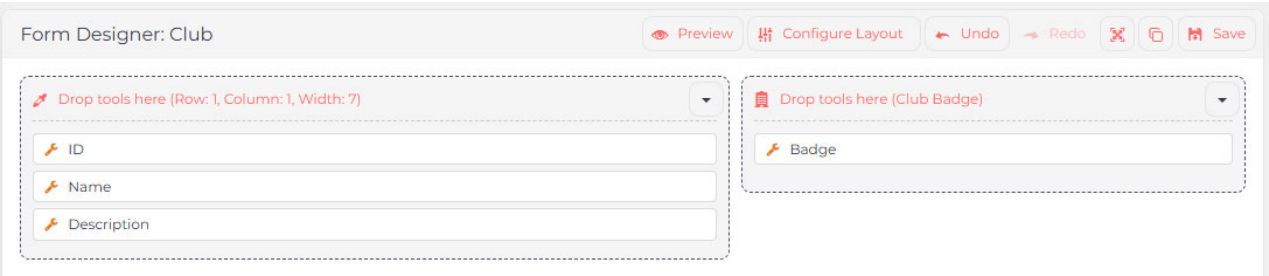
Fields



The fields panel shows a list of all available ReST API data source request fields which can be dragged onto the form. A filter text box allows matching fields to be displayed.

Form Designer

The form designer panel is to the right of the tools panel.

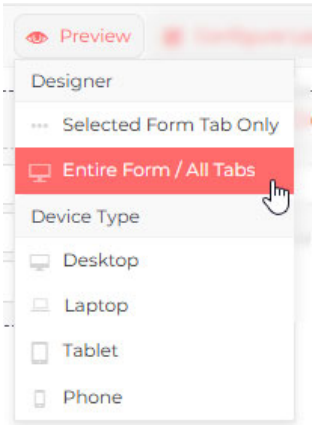


The panel shows the name of the form with a number of toolbar buttons:

Button	Action
Preview	Drop down menu to preview the form as it will look to the end-user.
Configure Layout	Specify the layout of the rows and columns of the form.
Undo	Undo the last form design change.
Redo	Redo the last form design change.
Minimize/Maximize	Hide the tools panel to see more of the form design surface.
Open Tools	Show the tools panel.
Save	Save the design of this form.

Preview

This is a drop down menu:



Selected Form Tab Only

This shows only the currently selected tab in preview mode. This is useful when focusing on specific detail only.

Entire Form / All Tabs

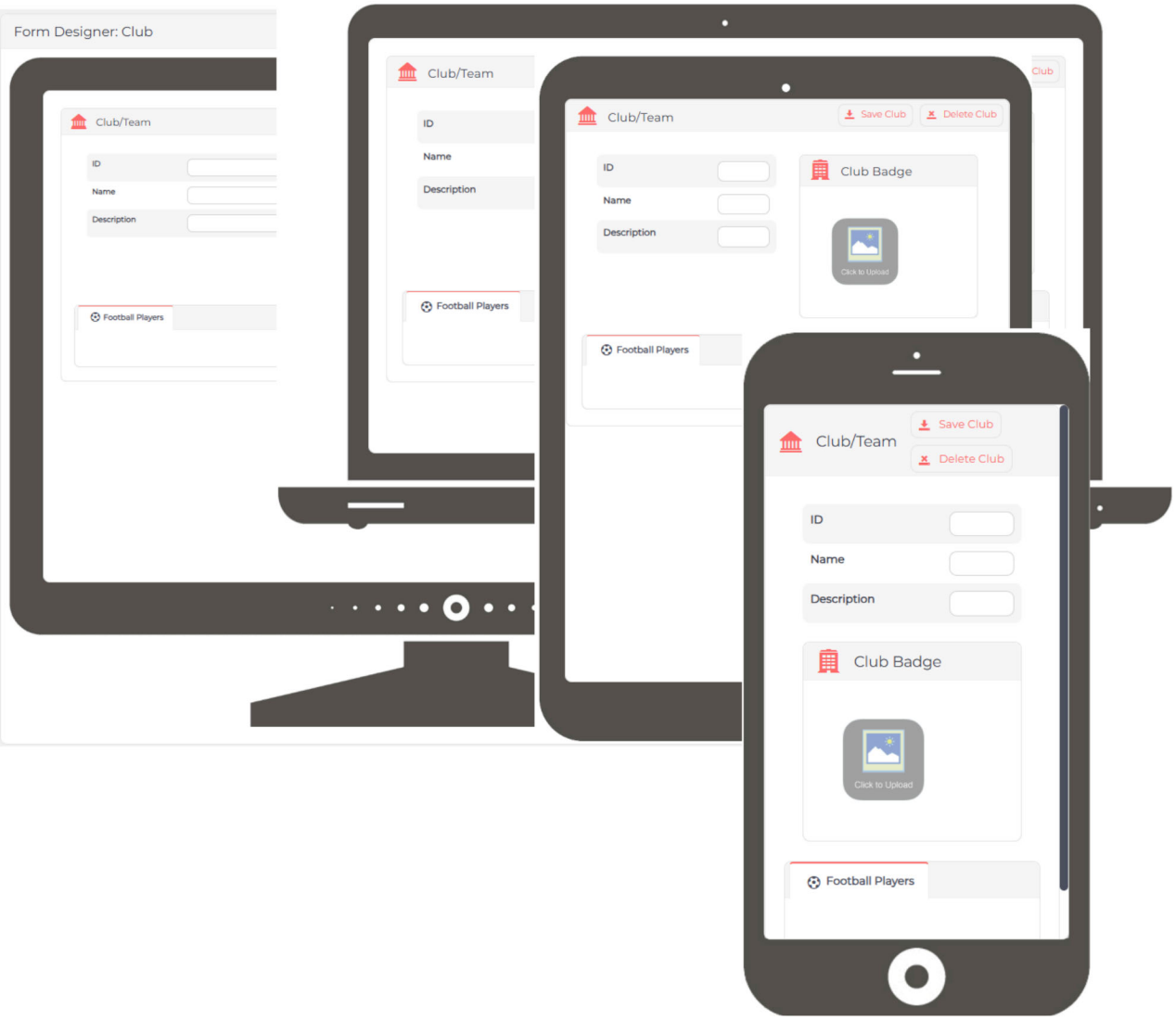
This shows all tabs in preview mode. This is useful when focusing on how the full form looks.

Device Type

Device Type	Description
Desktop	This previews the largest resolution to fit that of a desktop computer which has plenty of height and width.
Laptop	This previews a smaller resolution to fit that of a laptop computer which has less height and width than a desktop.
Tablet	This previews the resolution to fit that of a tablet device which is narrower than the laptop.
Phone	This previews the smallest resolution to fit that of a mobile phone which is very narrow.

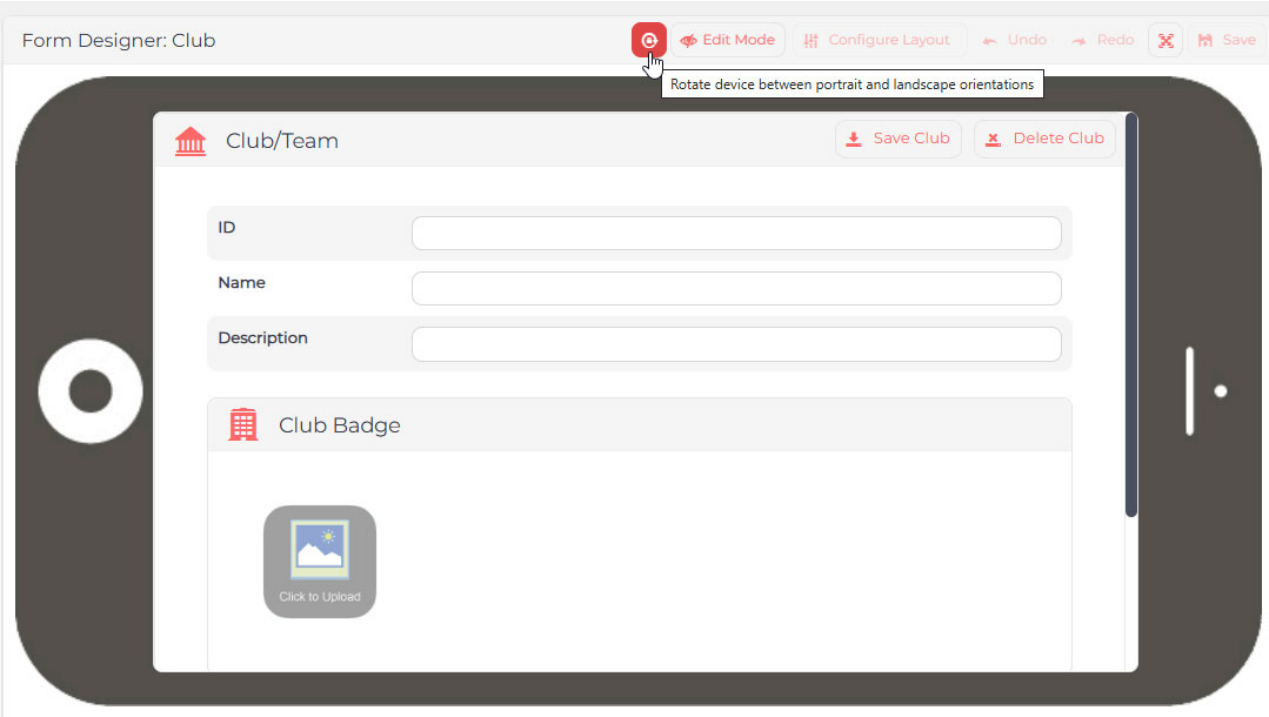
Example Previews

These are the respective previews of desktop, laptop, tablet and phone:



Rotate Preview

When in preview mode, the option to rotate the device appears:

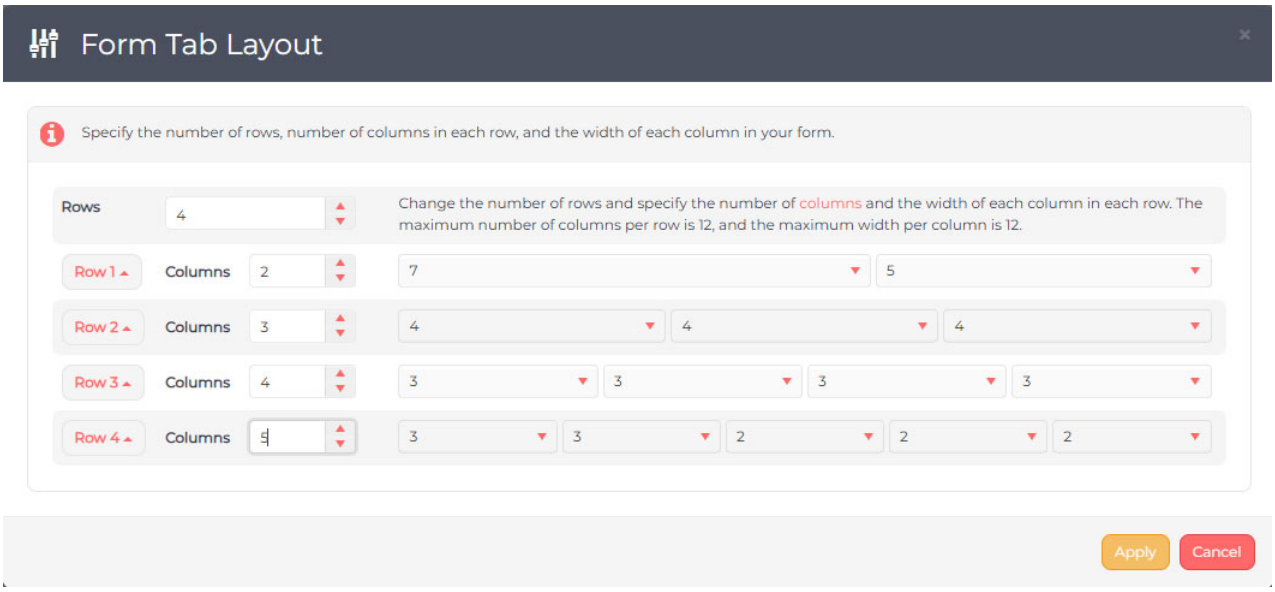


Edit Mode

When in preview mode, the Preview button is replaced with the Edit Mode button to revert back into form design mode.

Configure Layout

This opens a modal popup to configure the column and row layout of the form design surface:

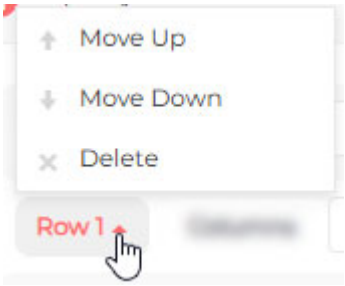


Rows

Incrementing the Rows number increases the number of rows on the form.
Decrementing the Rows number decreases the number of rows on the form.

Row Menu

This drop down menu is available for each row.



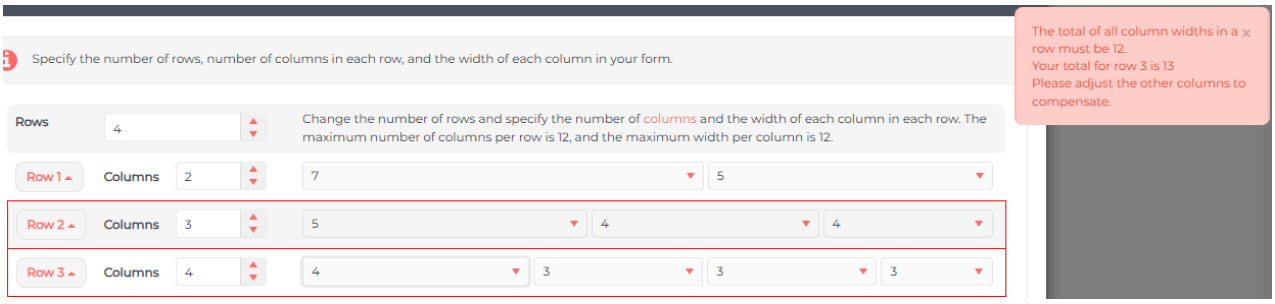
The selected row can be moved up or down or deleted.

Columns

Incrementing the column counter adds an additional column to the row.
Decrementing the column counter removes the right-most column from the row.

Each column that is displayed obeys the 'bootstrap' 12 columns per row model.
Each column in this designer is a drop down combo containing the numbers 1 to 12.
You can therefore set the width of every column for the exact layout you require.

Validation kicks in to warn you if your column widths do not total 12.



Apply

This button saves the row/column layout to the underlying form design which reflects these changes. Note that you will need to save the form design to persist these changes.

Cancel

This button cancels all form layout changes.

Undo

This button maintains a list of changes which can be used to undo the previous change, then the one before that etc.. This is useful if you make a mistake during the design process.

Redo

This button maintains a list of changes which can be used to redo the previous change, then the one after that etc.. This is useful if you make a mistake during the design process.

Maximize

Use this button to maximise the form design surface by hiding the tools panel.

Save

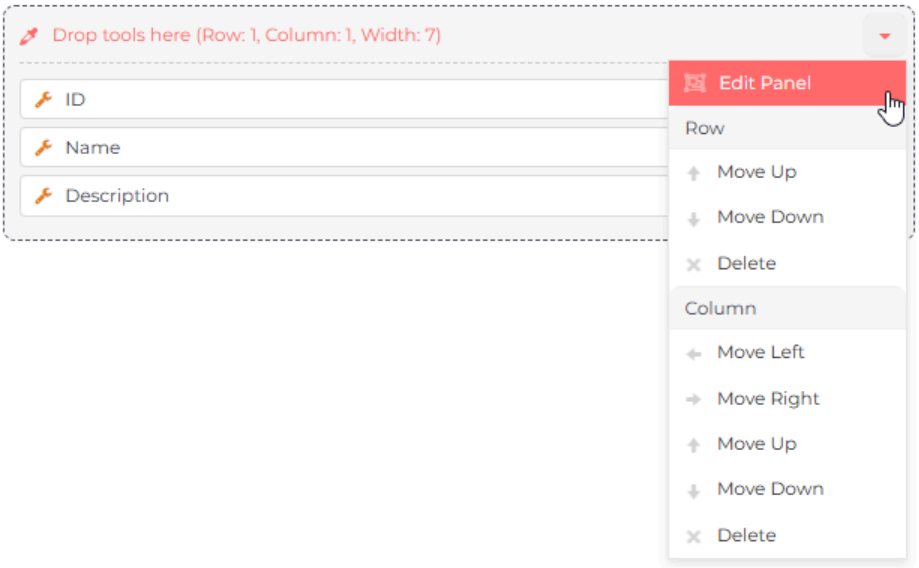
Save the design of the form to persist it. All changes overwrite the previous.

Row/Column Panels

Having defined your row and column layout using the configure layout popup, you can now design each panel which lives inside each column.

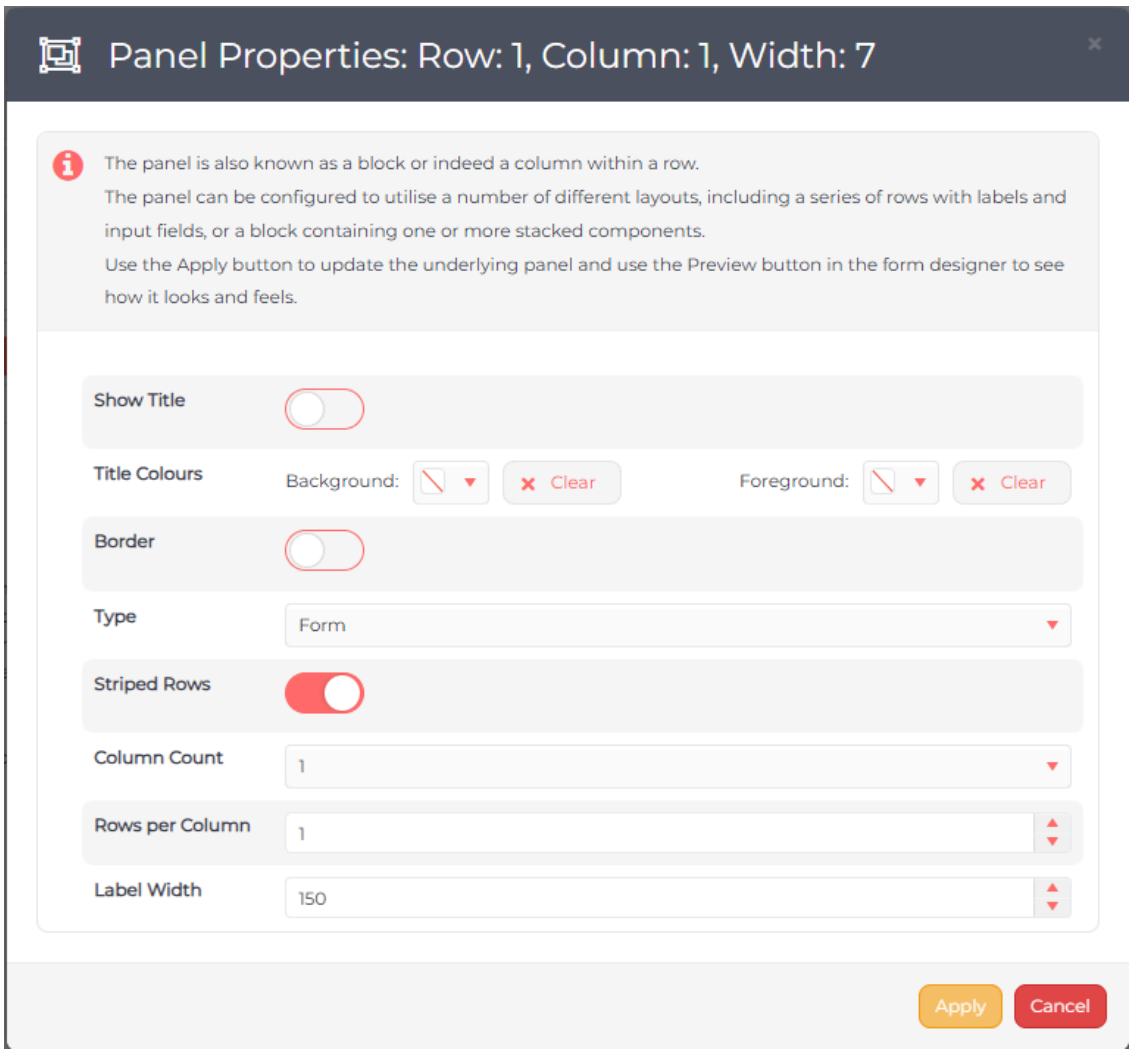
Panel Control Menu

This drop down menu is positioned top right of each column:



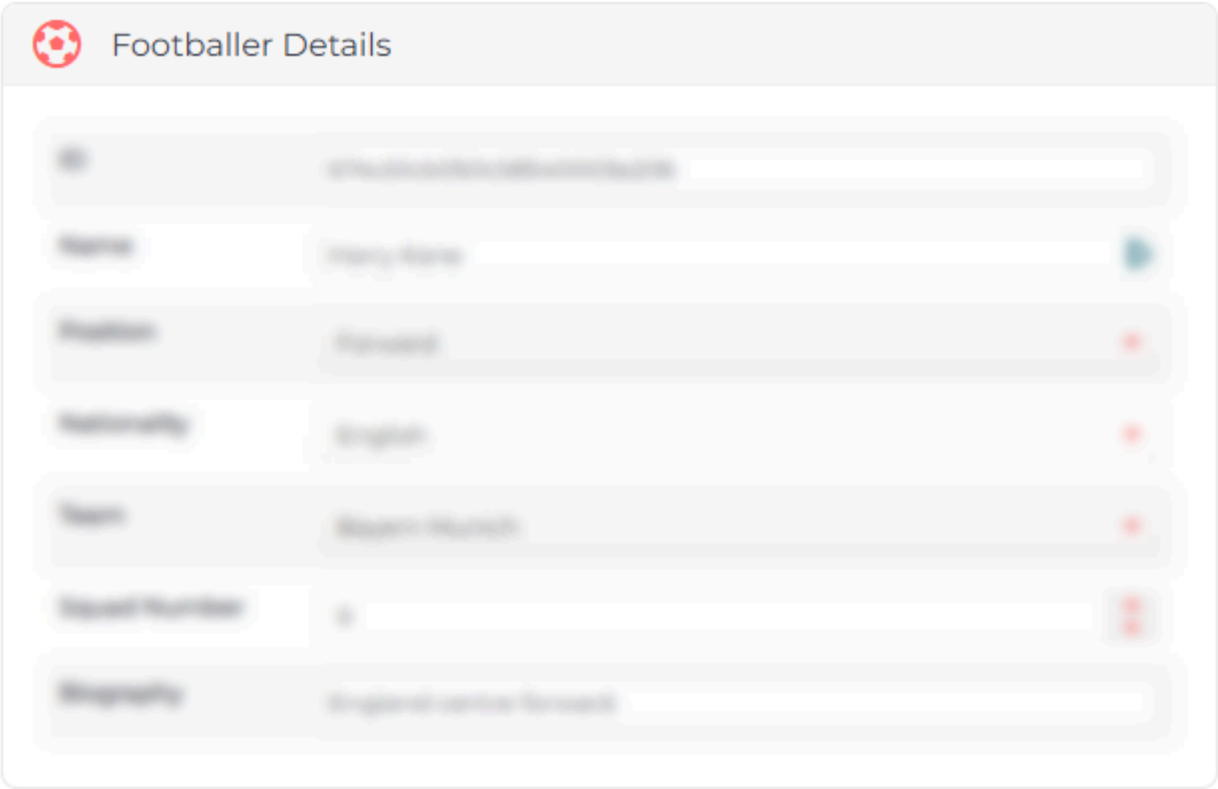
Edit Panel

This option opens up the panel properties modal popup to set the following properties of the panel:



Panels are containers for components and fields when we start the editing process by dragging and dropping from the toolbar onto the form, and setting their respective properties.

The panel at run-time can be shown with an icon and a header and border, or it can be hidden.



The properties of the panel allow for a variety of layouts within the panel.

Property	Description
Show Title	Show or hide the title bar.
Title Icon	Set the icon of the title if shown.
Title Text	Set the text of the title if shown.
Title Colours	Set the background and foreground colours of the title.
Border	Whether the panel has a border or not.
Type	Either Form or Block. Each affects the layout of the panel.
Striped Rows	Alternate rows have different colours, grey and white.
Column Count	Set the number of columns within each panel.
Rows per Column	Set the number of rows for each column.
Label Width	The width of the field label.

The panel properties must be applied and then the form design must be saved for these properties to be persisted.

Row

This section of the panel drop down menu allows all panels in this row to be re-positioned above or below, or deleted.

Column

This section of the panel drop down menu allows this specific column panel to be re-positioned to the left, right, above or below, or deleted.

Having defined the rows, columns and panels of the form, the next sections detail how to start dragging and dropping components and fields to create your form.

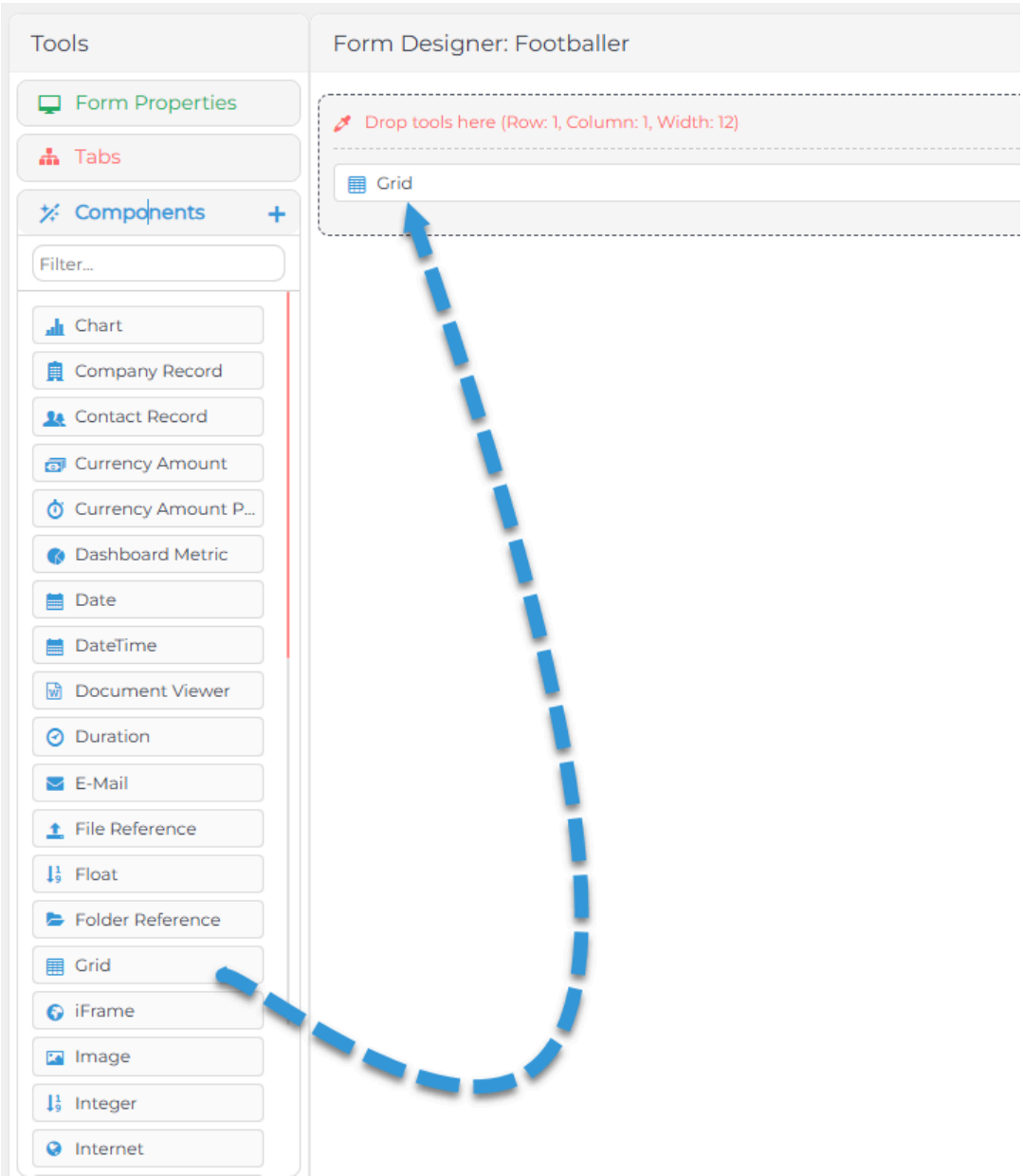
Components

Using components in form designer.

Components are specialised controls, or widgets, to present and manipulate data in a variety of forms, for example data grids, or images, or file references.

Components are often linked to ReST API data sources to consume data and each has their own complex behaviours which need to be configured.

Components appear in the form designer toolbar and can be dragged and dropped into panels as shown:



Once the component has been dragged and dropped into a panel, its properties can be set by simply clicking on it. This will open the component properties configurator.

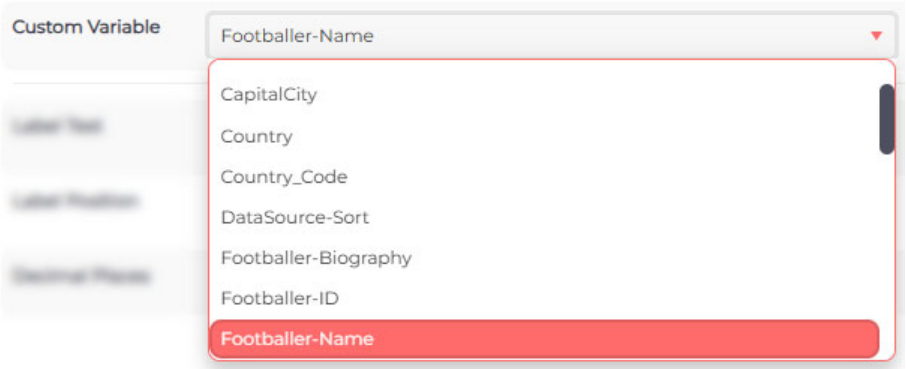
Each of the components has its own section which details how to configure it, however the following tabs are standard when configuring any component.

Details

This is the where the component is to be associated with a custom variable, as well as setting important visual properties.

Custom Variable

The component can be associated with a custom variable in order to facilitate the mapping from the data entry form into the data source request. This property is a drop down combo showing the list of all client-side custom variables.



Label Text

At run-time, the component will have a label displayed to its left or above it, for example:

The text of this label can be specified here.

Label Visible

Set whether the label is visible or not.

Label Position

The label can be positioned to the left of the component, or above it.

Read-Only

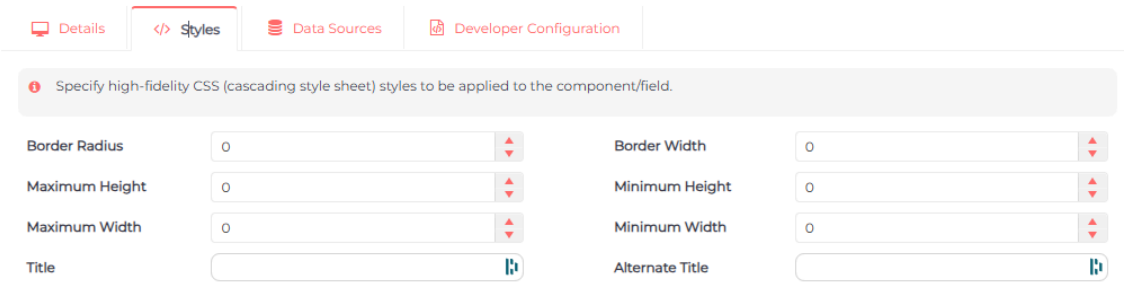
The component can be set to be non-editable or editable.

Decimal Places

For numeric integer or float components, this property specifies how many decimal places are displayed.

Styles

This tab allows the styling properties of the component to be overridden. Web applications use cascading style sheets (CSS) to control styles. These properties are are easy to set. Note that if a setting is 0 or blank, then the default styling remains.



Border Radius

Each of the four corners of the component can be curved if required.

Border Width

The component can be shown with a border of any size.

Maximum Height

The component maximum height can be set. Typically for images, the source image could be very large, so this setting creates a 'thumbnail' i.e. a smaller image for display.

Minimum Height

The component minimum height can be set. Typically for images, the source image could be very small, so this setting creates a potentially enlarged image for display.

Maximum Width

The component maximum width can be set. Typically for images, the source image could be very large, so this setting creates a 'thumbnail' i.e. a smaller image for display.

Minimum Width

The component minimum width can be set. Typically for images, the source image could be very small, so this setting creates a potentially enlarged image for display.

Title

This is the tooltip which hovers over the component when the mouse is over the field. Setting this can add clarity to end-users who need more information about a specific component.

Alternate Title

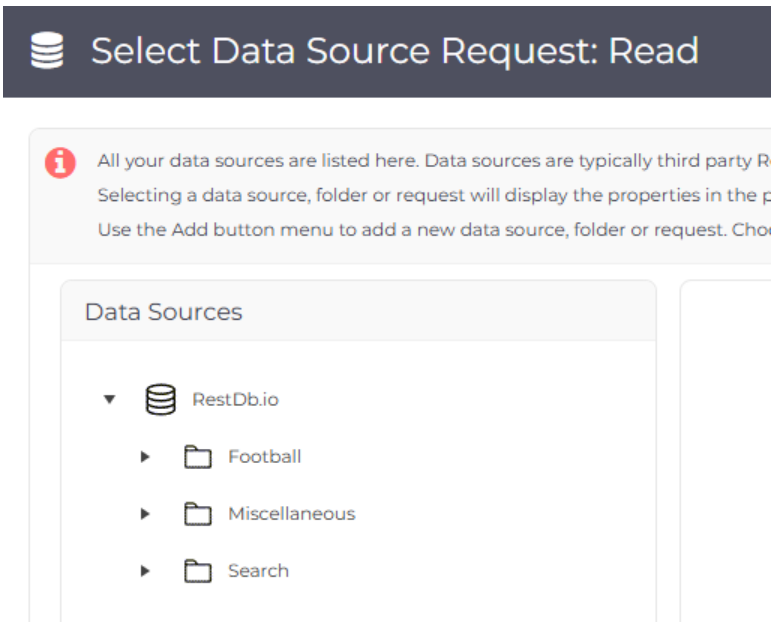
Web applications showing images should have another title which is displayed if the underlying image cannot be rendered e.g. missing. This is displayed instead of the missing image.

Data Sources

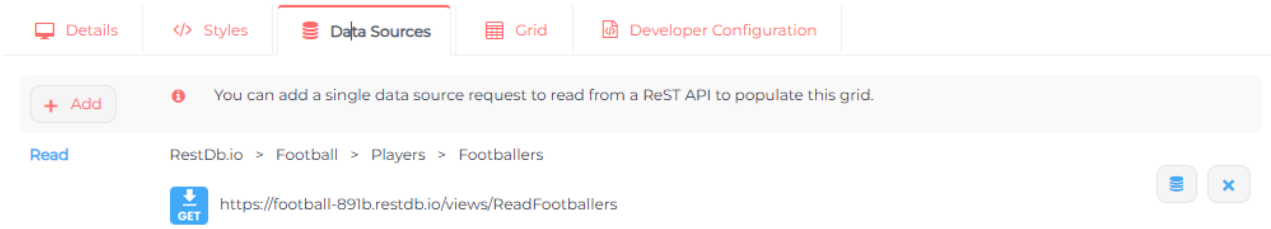
This tab is where a single 'Read' data source request can be added to pull data from a ReST API to populate this component on the form.

Add

This button opens the Select Data Source Request modal popup to choose a ReST API data source request to link to populate the component data.



Once the data source request is selected, it will be displayed in this tab.



Component: Grid

The grid component displays tabular data from ReST API requests which can be filtered and sorted.

The grid is one of the most important components in the application as it displays pages of data received from ReST API data source requests sorted into rows (records) and columns (fields).

This is an example of a grid showing page 1 of data from a ReST API request.

Footballers Pagination Test

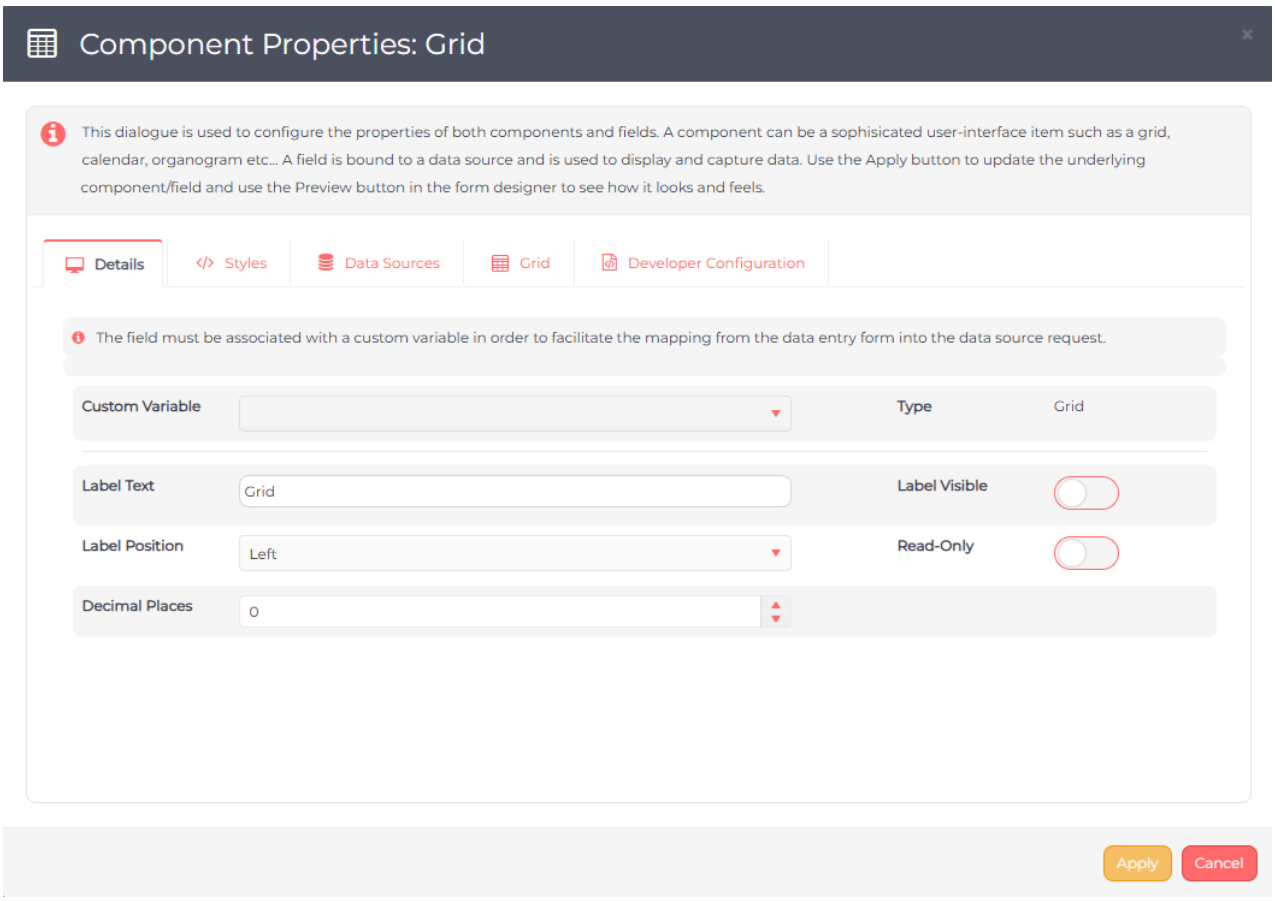
Footballer Name	Position	Club	Nationality	Squad No.
Alessia Russo	Forward	Arsenal	English	23
Alex Morgan	Midfielder	Wolverhampton Wanderers	American	6
Alex Scott	Midfielder	Bournemouth	English	8
Alexander Isak	Forward	Newcastle United	Dutch	9
Alexander Iwobi	Forward	Fulham	Nigerian	17
Alisha Debora Luiz	Forward	Wolverhampton Wanderers	Swiss	9
Alisha Lehmann	Forward	Aston Villa	Swiss	9
Alisson Becker	Goalkeeper	Liverpool	Brazilian	1
Alphonse Areola	Goalkeeper	West Ham United	French	1
Ana Crnogorčević	Midfielder	Wolverhampton Wanderers	Swiss	7

Page 1 of 13 10 rows per page 1 to 10 of 122 rows

The grid has the following features:

Feature	Description
Pagination	Displays only one page of data at a time. This is highly performant and means that only one page of data is read from the ReST API at a time. The end-user is able to page through each page as required.
Filtering	Most columns can be filtered to show only those records which match your filter text. Multiple columns can be filtered at the same time.
Sorting	Most columns can be sorted. Repeated clicking on the column header toggles between ascending, descending and none. Multiple columns can be sorted at the same time.
Column Selection	The column menu allows columns to be shown or hidden.
Column Ordering	Users can drag and drop columns into their preferred position.
Column Resizing	Users can drag column separators to resize each column.
Drill Down	Column data can be hyperlinked to allow the user to 'drill-down' to open a data entry form for the selected data. Multiple drill downs can be configured per grid to drill down into multiple forms.

Grid components can be dragged from the form designer toolbox into the form for both lookup and data entry forms. Once a grid is on the form being designed, click it to open up the component properties modal popup dialogue.

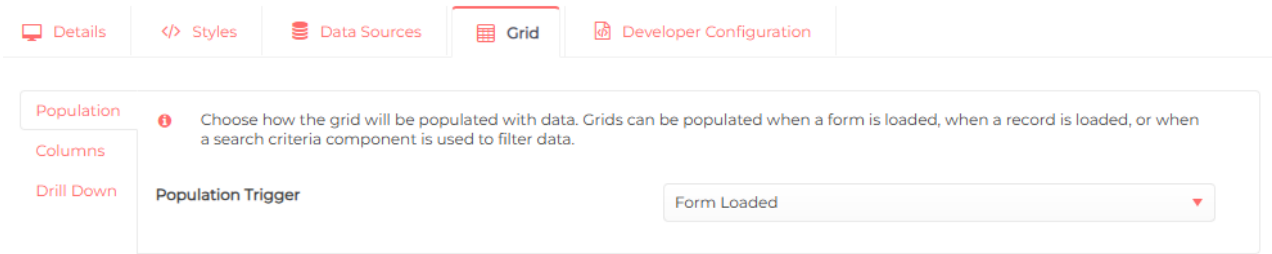


The form has 5 tabs, and an Apply button to save the changes and close the form, and a Cancel button to close the form without saving. Note that saving a component must be followed by saving the form design in order to persist the settings.

The Details, Styles, Data Sources and Developer Configuration tabs are discussed in the Components section. The Grid tab is specific to this component.

Grid

This tab allows the grid specific behaviour to be configured.



There are 3 left docked sub-tabs.

Population

Choose how the grid will be populated with data. Grids can be populated when a form is loaded, when a record is loaded, or when a search criteria component is used to filter data.

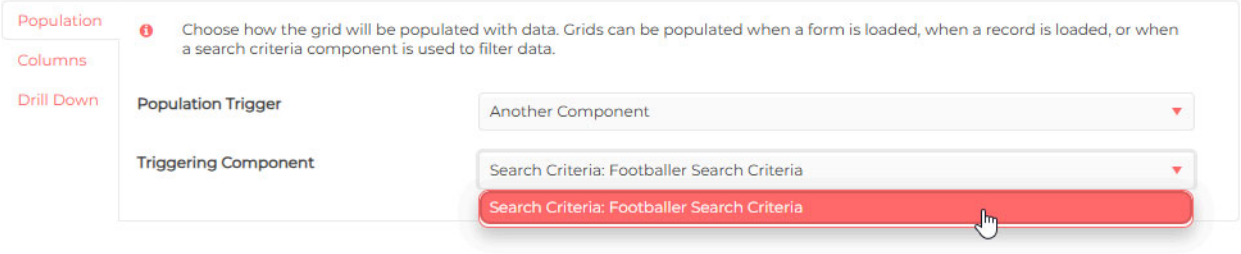
Population Trigger

This drop-down combo has the following actions:

Action	Description
Another Component	The grid is populated when another component triggers it, for example a search criteria filter when the user applies the filter.
Form Loaded	The grid is populated when the form is opened. Typically this is for lookup forms.
None	The grid is not populated automatically. There may be some other custom programming logic which populates this grid.
Record Loaded	The grid is populated when the record is displayed. Typically this is for data entry forms. For example if a product form loads a product, then a grid may then display the historic stock levels of this specific product.

Triggering Component

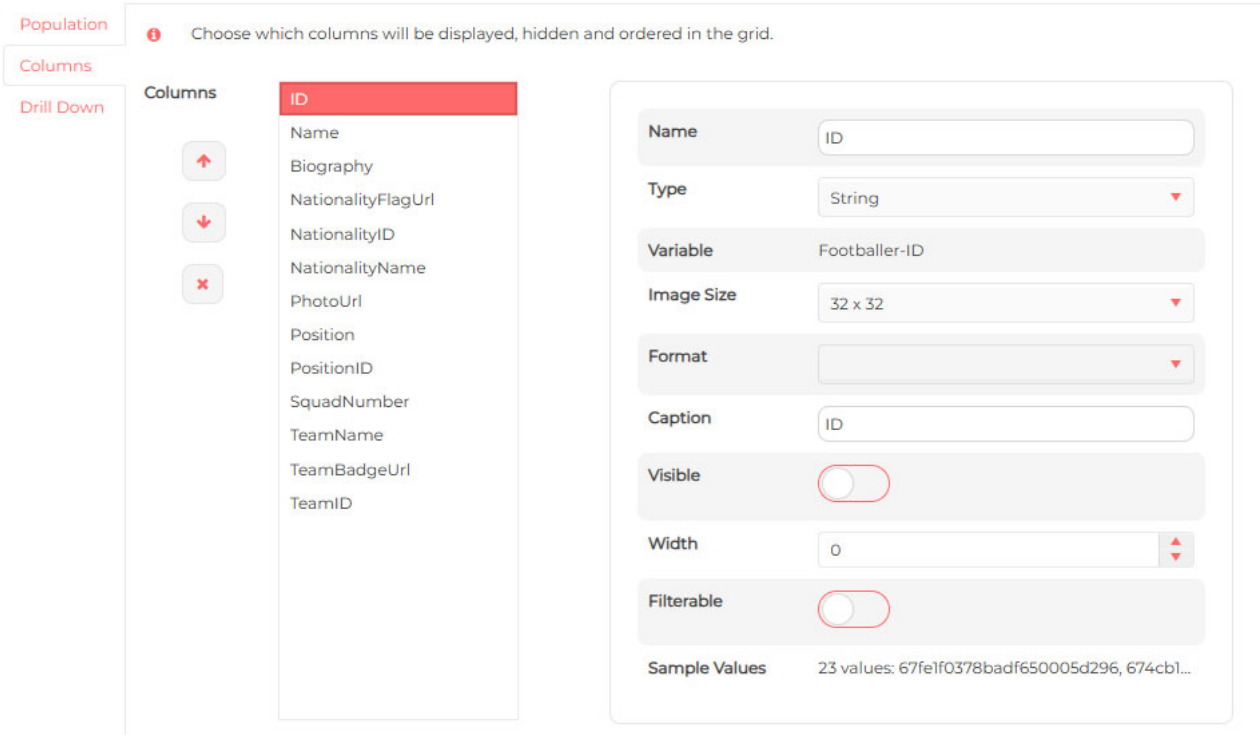
This drop-down is shown only when Population Trigger is set to "Another Component" and shows a list of other components located on this form.



In this example, the component which triggers a data grid population is the Search Criteria component.

Columns

This sub-tab is used to choose which columns will be displayed, hidden and ordered in the grid.



The list of columns is sourced from the fields returned in the linked ReST API data source. When this was configured, those original columns properties are shown here and are read-only.

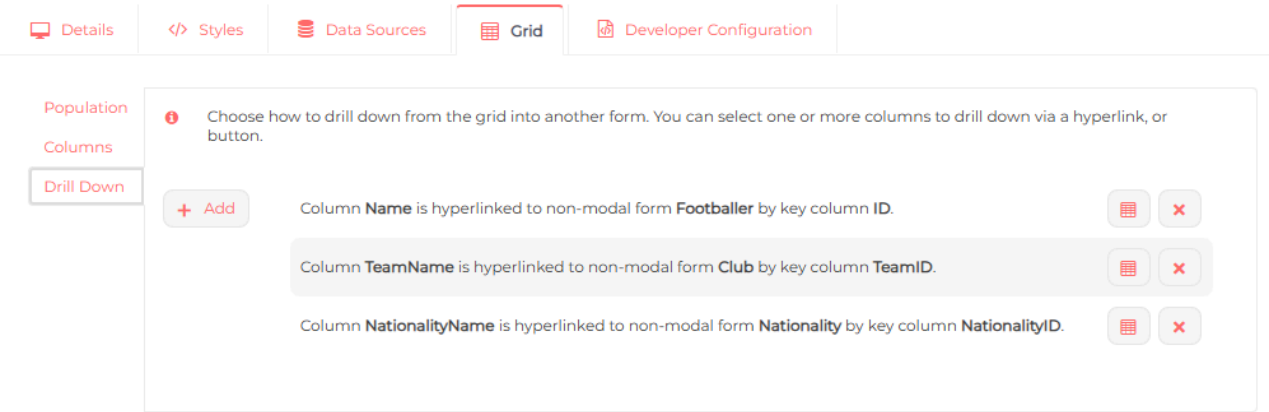
The column order can be manipulated by using the buttons to the left of the list to move up, move down, or indeed delete the column from the list. It is often better to simply hide columns rather than delete them.

There are also new column properties:

Column Property	Description
Name	This is the name of the field from the ReST API request
Type	Usually a string or a number, but sometimes an image URL or Base64 image.
Variable	If the column is linked to a custom variable then that is shown here.
Image Size	When showing images in grid columns, this dictates the size of the image displayed. Images can be between 16 and 1024 pixels in size.
Format	Most columns will display as expected, however some may need to be specially formatted e.g. currency, dates etc..
Caption	This is the column header text. Often field names from ReST API's can be cryptic so setting a human legible caption is recommended.
Visible	Whether the column is visible or not
Width	The width of the column. When 0, the column will be sized to fit available space. For image columns, this should be set to 20 wider than the image size.
Filterable	Whether the column can be filtered. Most columns should be filterable, however filtering on types such as images makes no sense.
Sample Values	This is a sample of the values read from the ReST API request to help configure the column.

Drill Down

This sub-tab is used to configure the drill down from selected columns to open forms. Multiple columns can be configured to drill down into their own form.



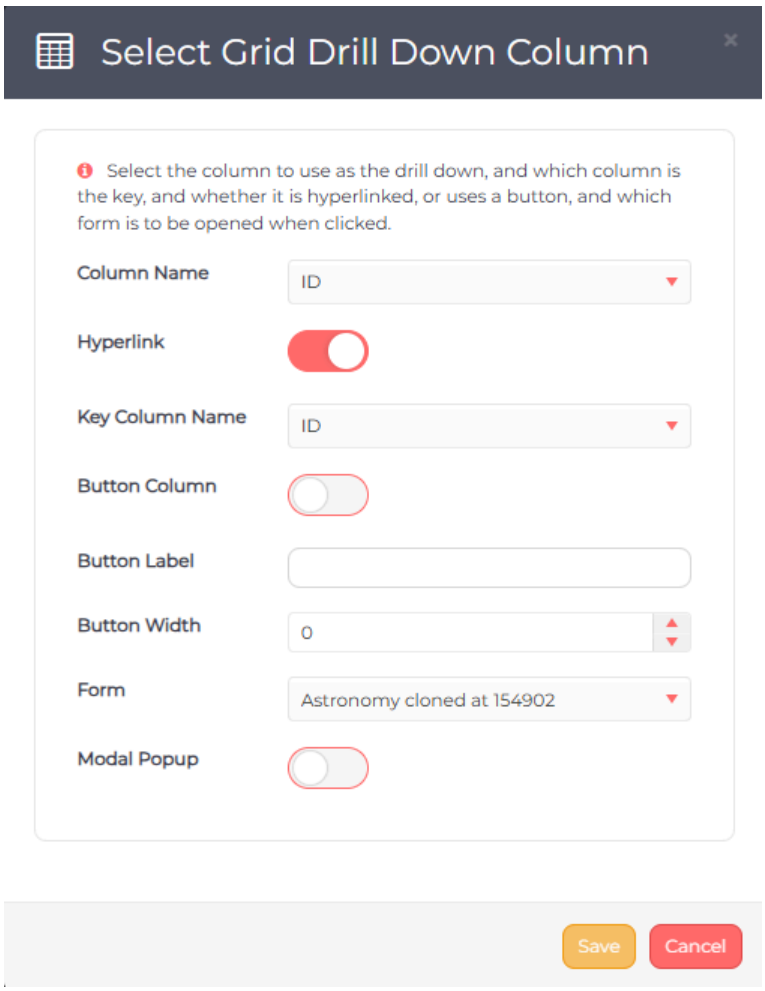
In this example, the grid has 3 columns which the end-user can click to drill down into different forms.

Each drill down column is hyperlinked to the identifier of a specific entity.

Each row in this tab can be edited or deleted using the respective buttons to the right of each row.

Add

This button opens a modal popup dialogue which allows configuration of the drill down column.



The drill down column properties are:

Property	Description
Column Name	This is a drop down combo showing all columns in the ReST API data source. This is the column which will be visible and highlighted when the users mouse hovers over it.
Hyperlink	When checked, this column can be hyperlinked to open a form record.
Key Column Name	This is a drop down combo showing all columns in the ReST API data source. This column will typically be invisible as it will contain a string of characters which represent a server-side record identifier. It will be this value which is passed into the form to identify the record to display.
Button Column	The column can be shown as a button, not a hyperlink. Clicking the button will drill down.
Button Label	If a button column, this is the text shown in the button.
Button Width	If a button column, this is the width of the button.
Form	This is a drop down combo showing all forms in your system. When the hyperlinked column is clicked, this form will open and will show the corresponding record identified by the Key Column Name.
Modal Popup	Hyperlinking can open modal popup forms rather than replace the currently shown form with another. This may be a better experience for some applications.

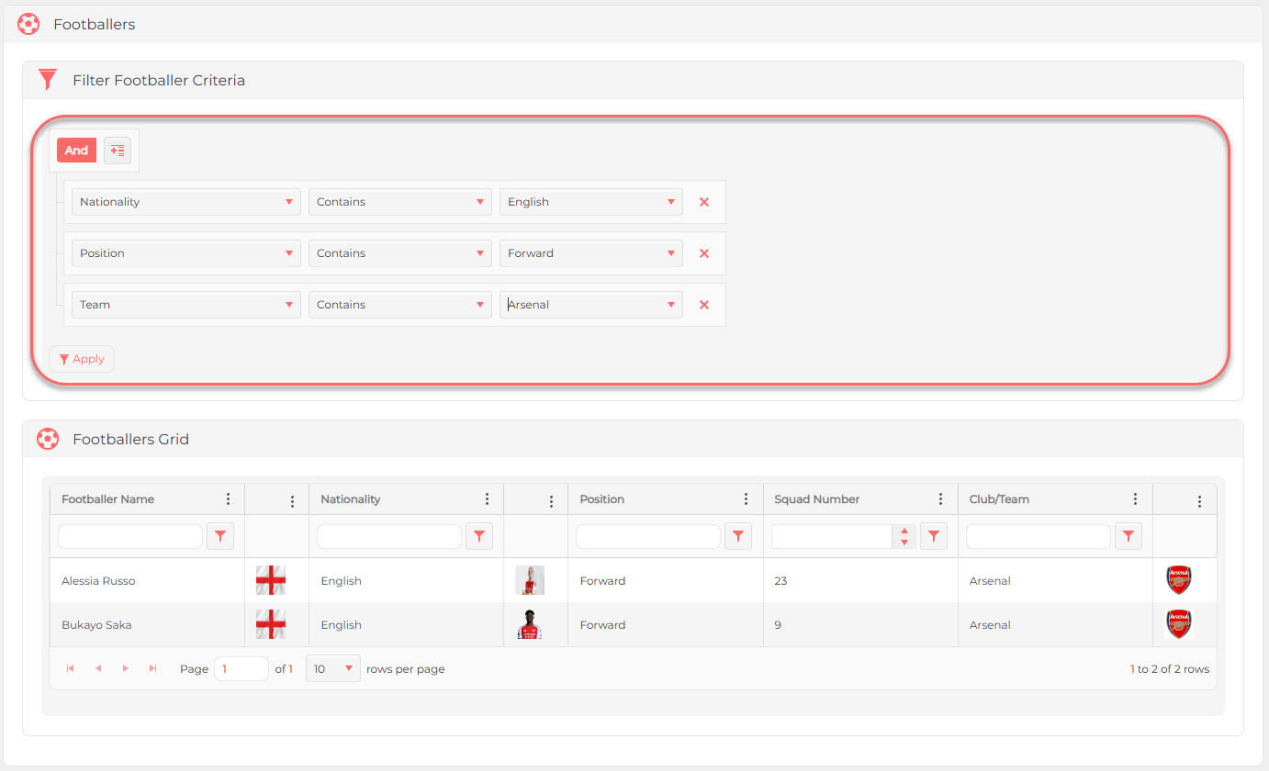
Use the Save button to save the properties to the component properties. Remember that you must save the component properties, and save the form design for any changes to be persisted.

Component: Search Criteria

The search criteria component displays one or more fields from multiple ReST API requests which can be used to search over large data sets.

The search criteria component typically works with another component such as the grid, to allow the end-user to add multiple filter fields which are then applied to return only matching data records.

This is an example of a search criteria component showing 3 lookup fields which when applied, displays matching data in the grid.

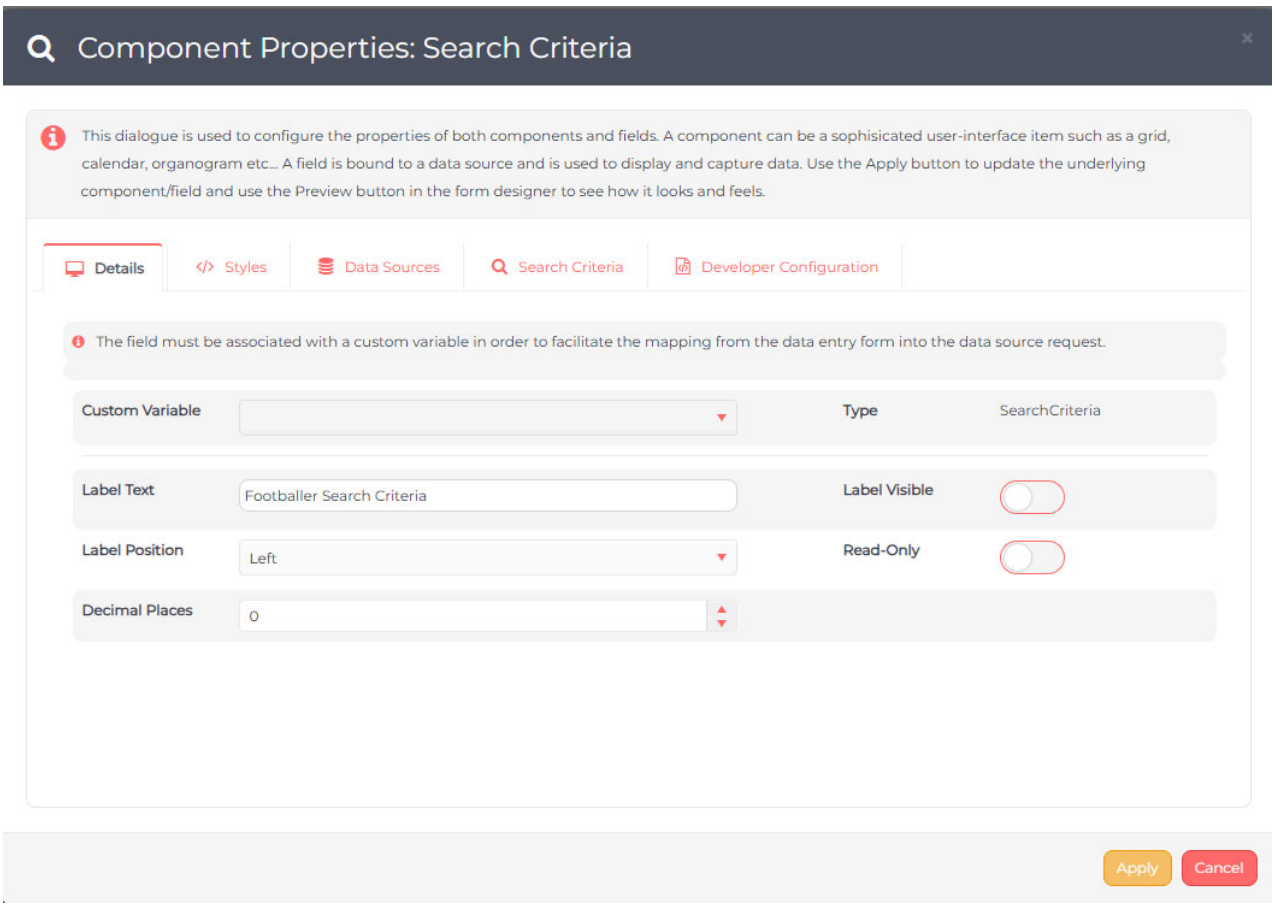


In this example, each of the 3 fields are populated from their own ReST API data source request.

The search criteria component has the following features:

Feature	Description
Multiple Fields	Use the 'plus' button to the right of the And button to add fields to the component. Use the x button to the right to remove each field.
AND	This component does not attempt to be too complicated to use so all fields are 'anded' meaning that only records which match all fields are returned.
Field Types	Fields can be lookups, text, numbers.
Apply	This button is used to send the selected search criteria to the attached component to display the matching data.
Persistence	Every time you re-open this form, the search criteria component remembers your search from last time.

Search Criteria components should be dragged from the form designer toolbox into the form for lookup forms only. Once a search criteria component is on the form being designed, click it to open up the component properties modal popup dialogue.

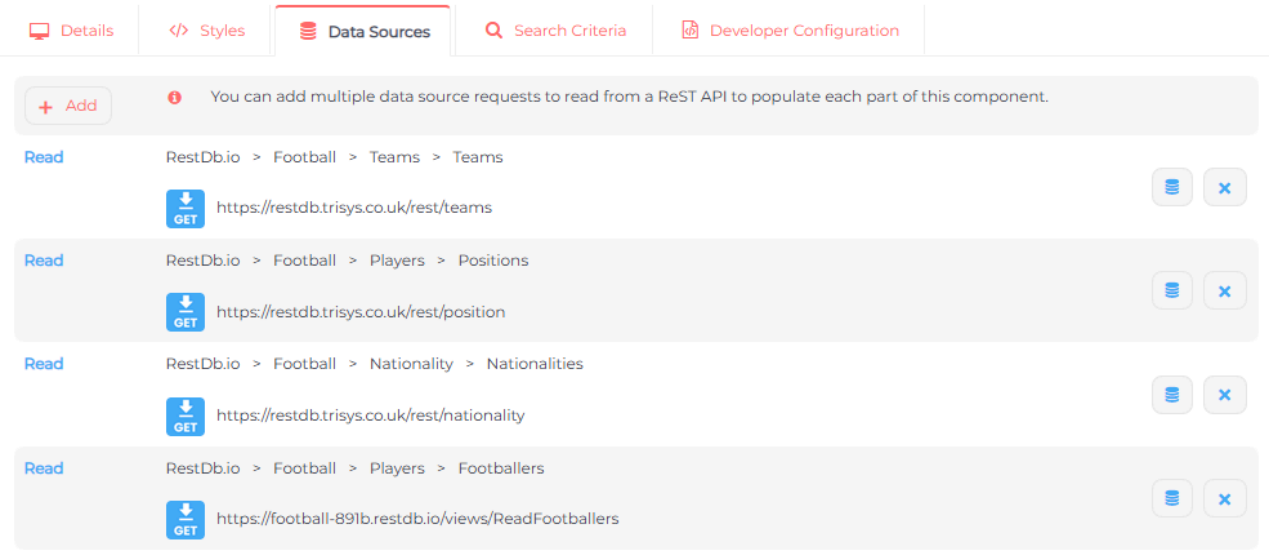


The form has 5 tabs, and an Apply button to save the changes and close the form, and a Cancel button to close the form without saving. Note that saving a component must be followed by saving the form design in order to persist the settings.

The Details, Styles, Data Sources and Developer Configuration tabs are discussed in the Components section. The Search Criteria tab is specific to this component.

Data Sources

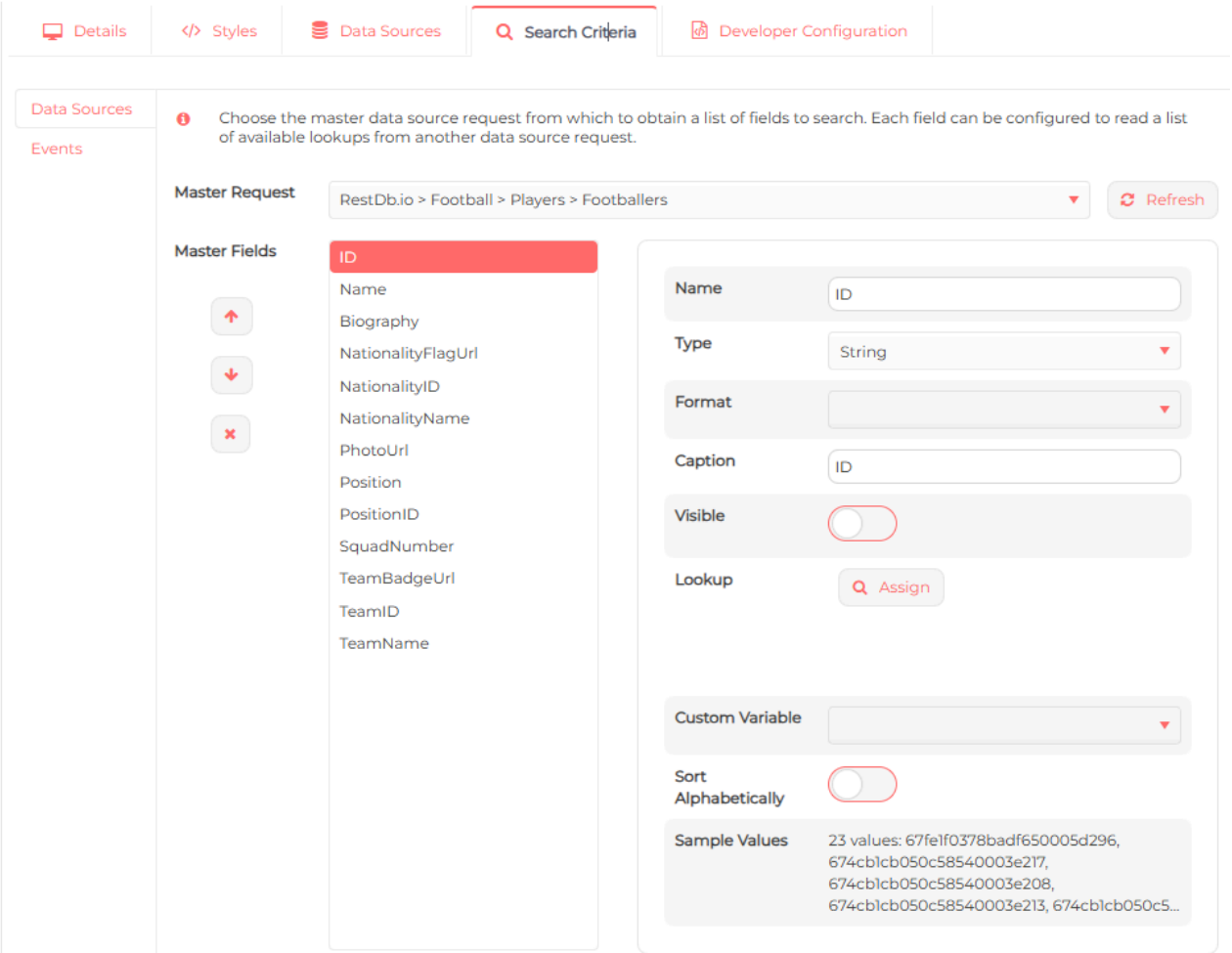
The data sources tab is similar to that discussed in the Components section however it is worth pointing out that this component will have multiple ReST API data source requests, one for each field, but also a master.



In this example, we can see that the first 3 data source requests are for specific fields, whereas the 4th is for the master data set i.e. the one which is searched using the filtered fields.

Search Criteria

This tab allows the search criteria specific behaviour to be configured.



There are 2 left docked sub-tabs.

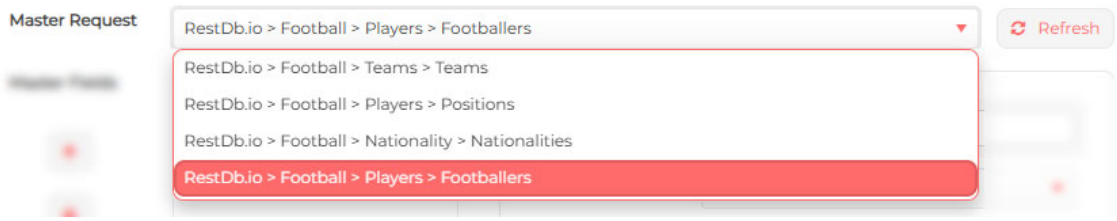
Data Sources

Choose the master data source request from which to obtain a list of fields to search. Each field can be configured to read a list of available lookups from another data source request.

Multiple data sources may have been configured, usually one for each field, but also a master request.

Master Request

This drop-down combo has the list of all the data source requests added in the [Data Sources tab](#).

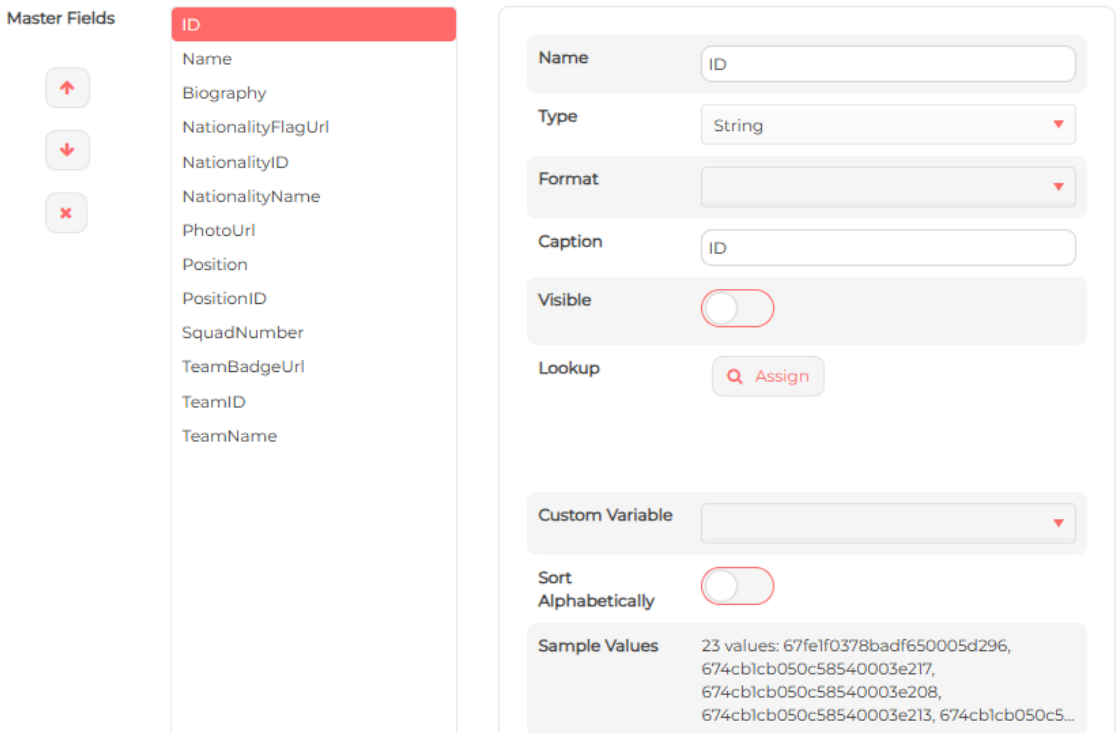


This master request should be selected as this then allows each of the other fields to map to fields in the master data set.

The Refresh button will read the selected master request and replace the existing field list.

Master Fields

This shows the list of all fields from the master ReST API data source request.

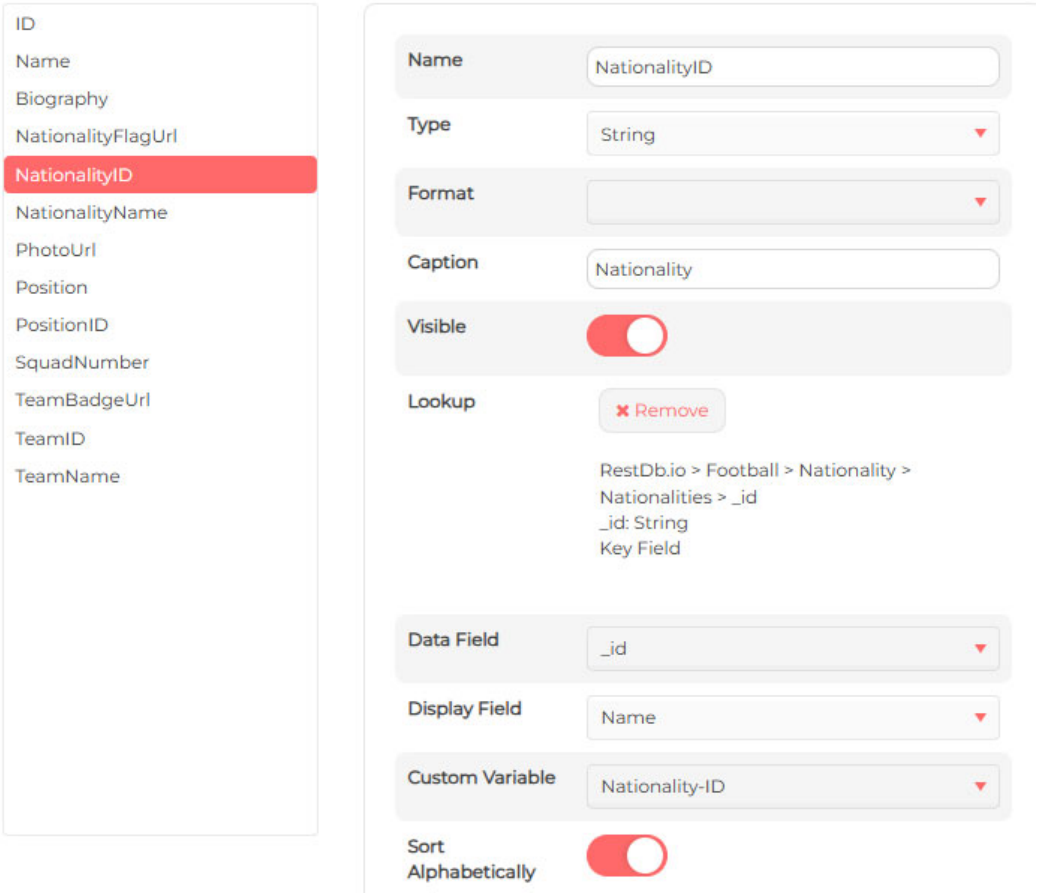


The field order can be manipulated by using the buttons to the left of the list to move up, move down, or indeed delete the field from the list. It is often better to simply hide fields rather than delete them.

The field properties are as follows:

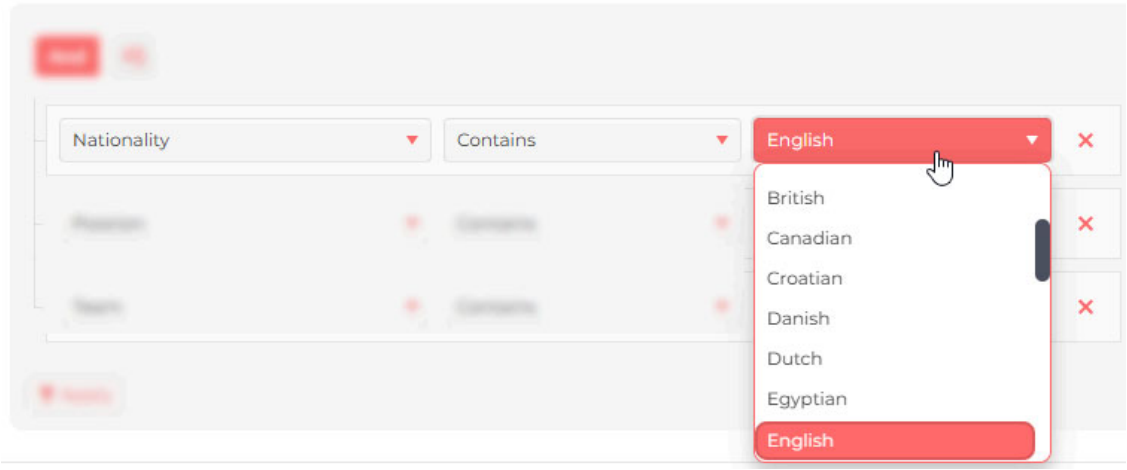
Column Property	Description
Name	This is the name of the field from the ReST API request
Type	Usually a string or a number, but sometimes an image URL or Base64 image.
Format	Most fields will display as expected, however some may need to be specially formatted e.g. currency, dates etc..
Caption	This is the field header text. Often field names from ReST API's can be cryptic so setting a human legible caption is recommended.
Visible	Whether the field is visible or not
Lookup	This shows whether any ReST API data source requests have been assigned to this lookup field. The Assign button is visible when none have been assigned and the Remove button is visible otherwise.
Data Field	Visible when a lookup is assigned. This is the data field from the ReST API request which is linked to the key identifier.
Display Field	Visible when a lookup is assigned. This is the data field from the ReST API request which is human recognisable.
Custom Variable	Visible when a lookup is assigned. This is the custom variable populated with the data field value. It is needed by connected components to filter data.
Sort Alphabetically	Visible when a lookup is assigned. Sorts the display field in alphabetical order.
Sample Values	This is a sample of the values read from the ReST API request to help configure the field.

An example of a Lookup is this:



This shows that the field is a lookup linked to a ReST API data source request, linked to an identifier field, using a different sorted field to display and linked to a custom variable.

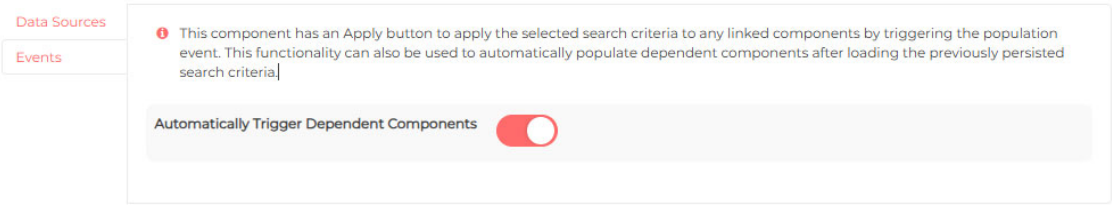
At run-time this specific field in this component shows the field name using the Caption property and shows a drop down list of Display Field values.



When the filter is applied, it is the field Name value which is passed to the connected component.

Events

This sub-tab is used to trigger dependent components. The Apply button applies the selected search criteria to any linked components by triggering the population event. This functionality can also be used to automatically populate dependent components after loading the previously persisted search criteria.



Automatically Trigger Dependent Components

Only the dependent components need to know who we are, not the other way around. This means that the search criteria component is not concerned with which other components are notified when the Apply button is clicked.

For example, a Grid component may be located on the same form, and it will be configured to be populated only when this specific Search Criteria component sends it a filter when the user clicks the Apply button.

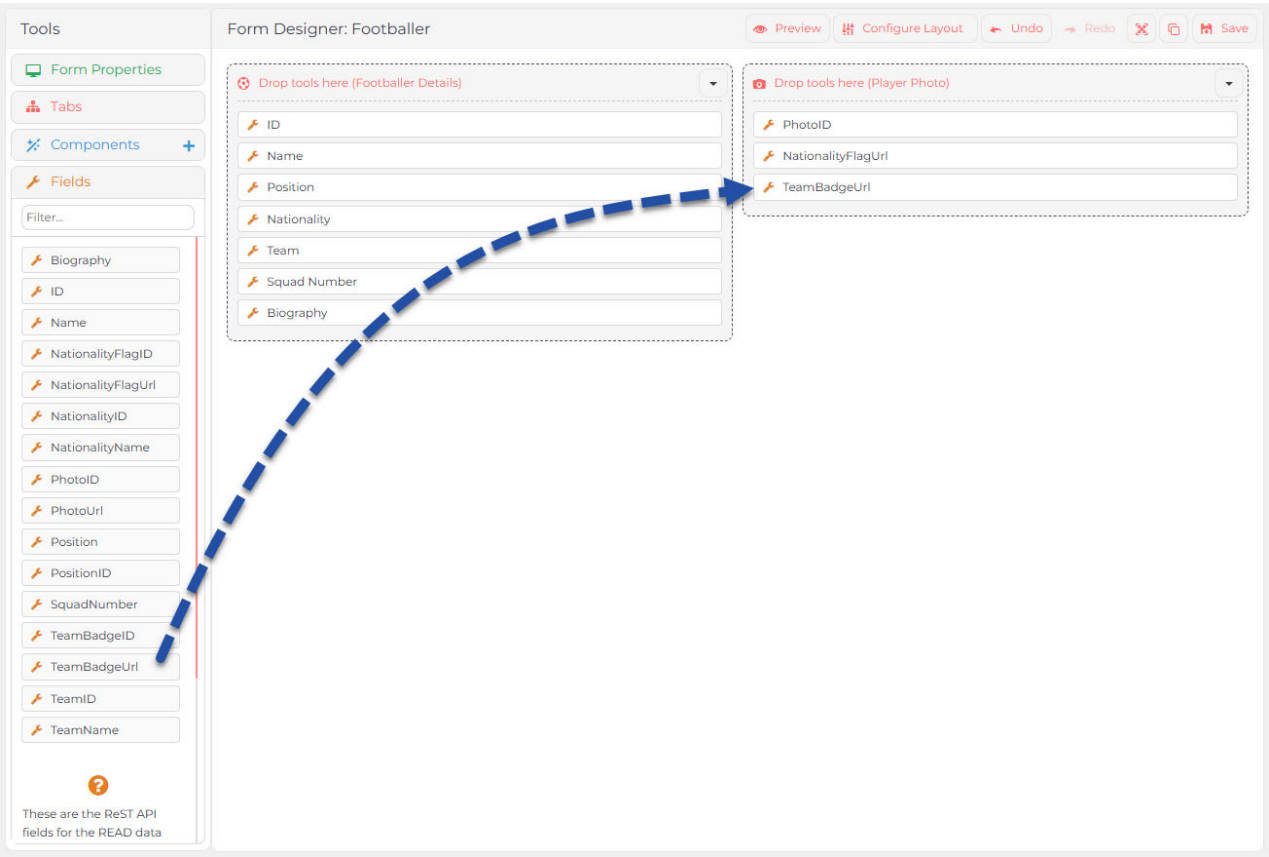
Fields

Using fields in form designer.

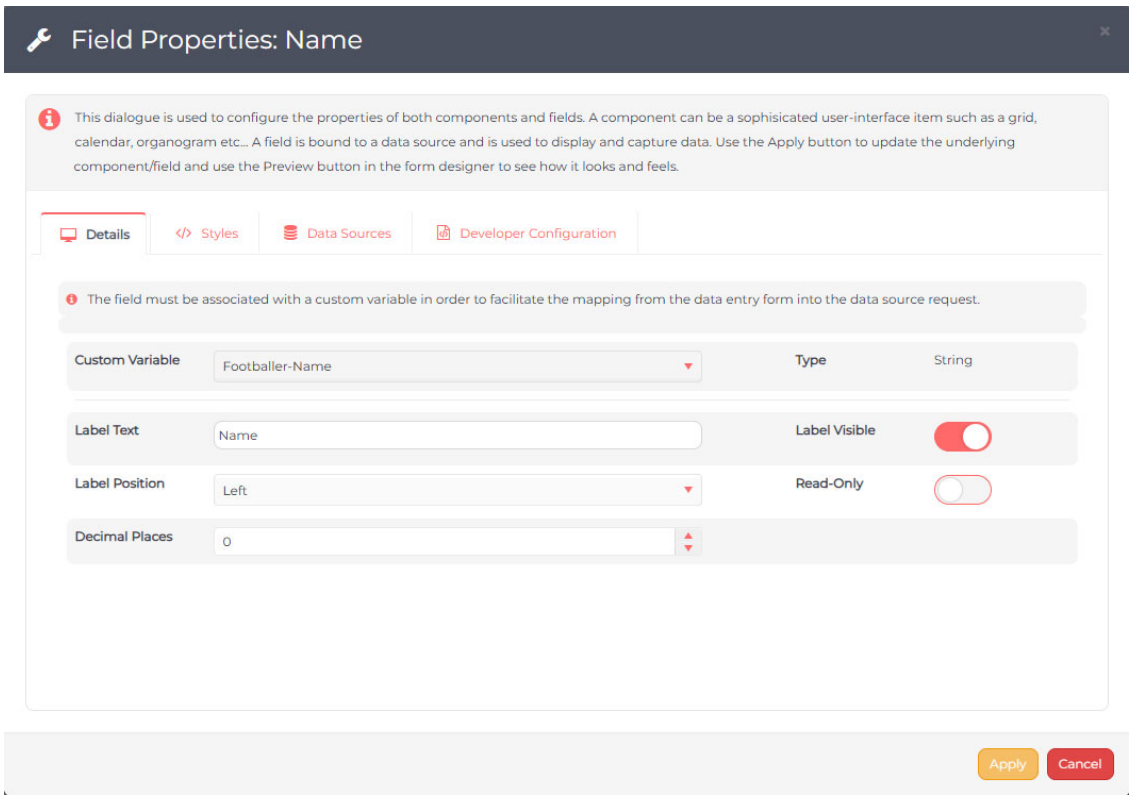
Fields are directly linked to the read ReST API data source configured for the form.

Each field is an essential part of the form CRUD mechanism, both displaying data and allowing the end-user to change it.

Fields appear in the form designer toolbar when the form is linked to a read ReST API data source and can be dragged and dropped into panels as shown:



Once the field has been dragged and dropped into a panel, its properties can be set by simply clicking on it. This will open the field properties configurator.



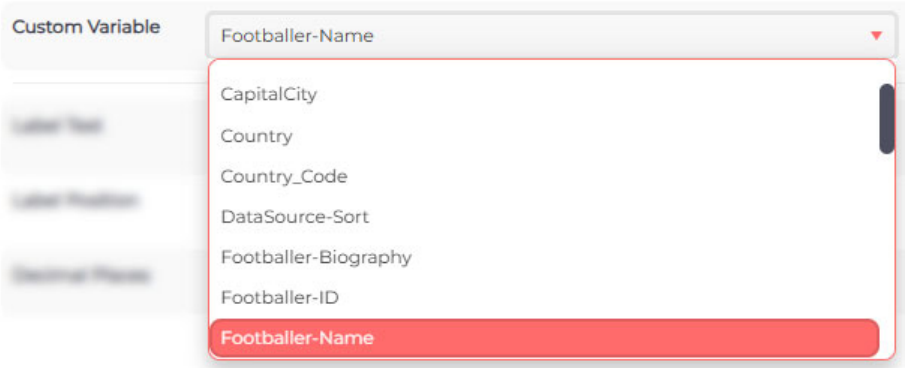
This modal popup dialogue has 4 tabs to control the behaviour of the field.

Details

This is the where the field must be associated with a custom variable, as well as setting important visual properties.

Custom Variable

The field must be associated with a custom variable in order to facilitate the mapping from the data entry form into the data source request. This property is a drop down combo showing the list of all client-side custom variables.



Label Text

At run-time, the field will have a label displayed to its left or above it, for example:

The text of this label can be specified here.

Label Visible

Set whether the label is visible or not.

Label Position

The label can be positioned to the left of the field, or above it.

Read-Only

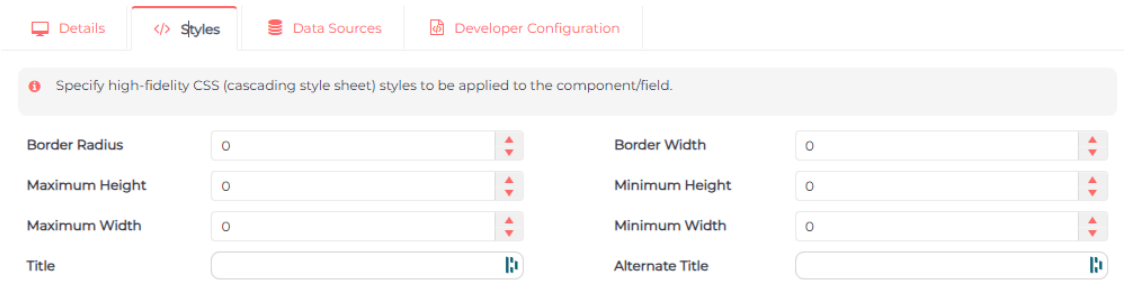
The field can be set to be non-editable or editable.

Decimal Places

For numeric integer or float fields, this property specifies how many decimal places are displayed.

Styles

This tab allows the styling properties of the field to be overridden. Web applications use cascading style sheets (CSS) to control styles. These properties are are easy to set. Note that if a setting is 0 or blank, then the default styling remains.



Border Radius

Each of the four corners of the field can be curved if required.

Border Width

The field can be shown with a border of any size.

Maximum Height

The fields maximum height can be set. Typically for images, the source image could be very large, so this setting creates a 'thumbnail' i.e. a smaller image for display.

Minimum Height

The fields minimum height can be set. Typically for images, the source image could be very small, so this setting creates a potentially enlarged image for display.

Maximum Width

The fields maximum width can be set. Typically for images, the source image could be very large, so this setting creates a 'thumbnail' i.e. a smaller image for display.

Minimum Width

The fields minimum width can be set. Typically for images, the source image could be very small, so this setting creates a potentially enlarged image for display.

Title

This is the tooltip which hovers over the field when the mouse is over the field. Setting this can add clarity to end-users who need more information about a specific field.

Alternate Title

Web applications showing images should have another title which is displayed if the underlying image cannot be rendered e.g. missing. This is displayed instead of the missing image.

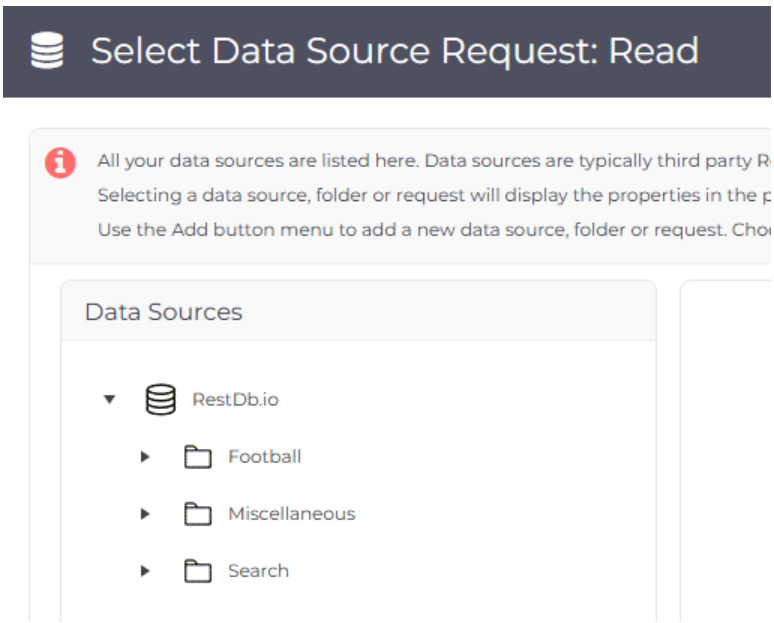
Data Sources

This tab is where a single 'Read' data source request can be added to pull data from a ReST API to populate a lookup combo for this field on the form.

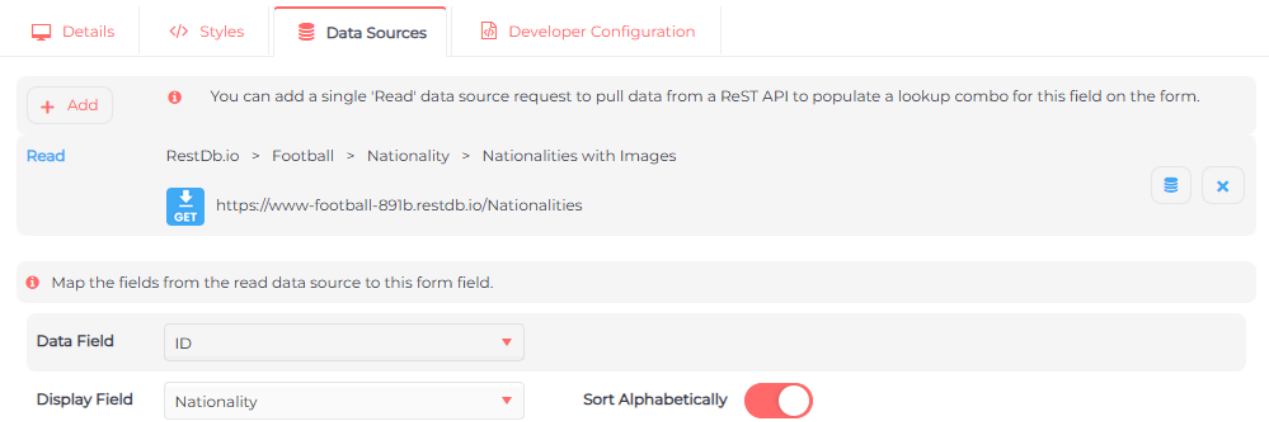
For data entry forms with ReST API data sources already assigned, this is not necessary, however for lookup forms where fields have been added to filter data sets, then assigning a ReST API data source to populate a lookup combo is necessary.

Add

This button opens the Select Data Source Request modal popup to choose a ReST API data source request to link to populate the field data.



Once the data source request is selected, it will be displayed in this tab:



Having selected the data source request, the Data Field and the Display Field drop down combos will be populated with the available fields.

Data Field

This is the data source request field which the form field maps to. Often ReST API's return foreign keys i.e. pointers to numbers of strings which should be displayed in human readable form. Thus the data field could be one of these so it is important to understand the ReST API data before mapping this.

Display Field

This is the data source request field which the form field should display. Often ReST API's return foreign keys i.e. pointers to numbers of strings which should be displayed in human readable form. Thus the display field could be one of these so it is important to understand the ReST API data before mapping this.

An example may be when the field on your form is a lookup combo for a nationality. You display the names of all the nationalities, however your field passes the nationality ID to the ReST API to lookup the data.

Sort Alphabetically

When using a display field, sorting the data alphabetically is recommended, allowing end-users to use their keyboard to quickly locate strings.

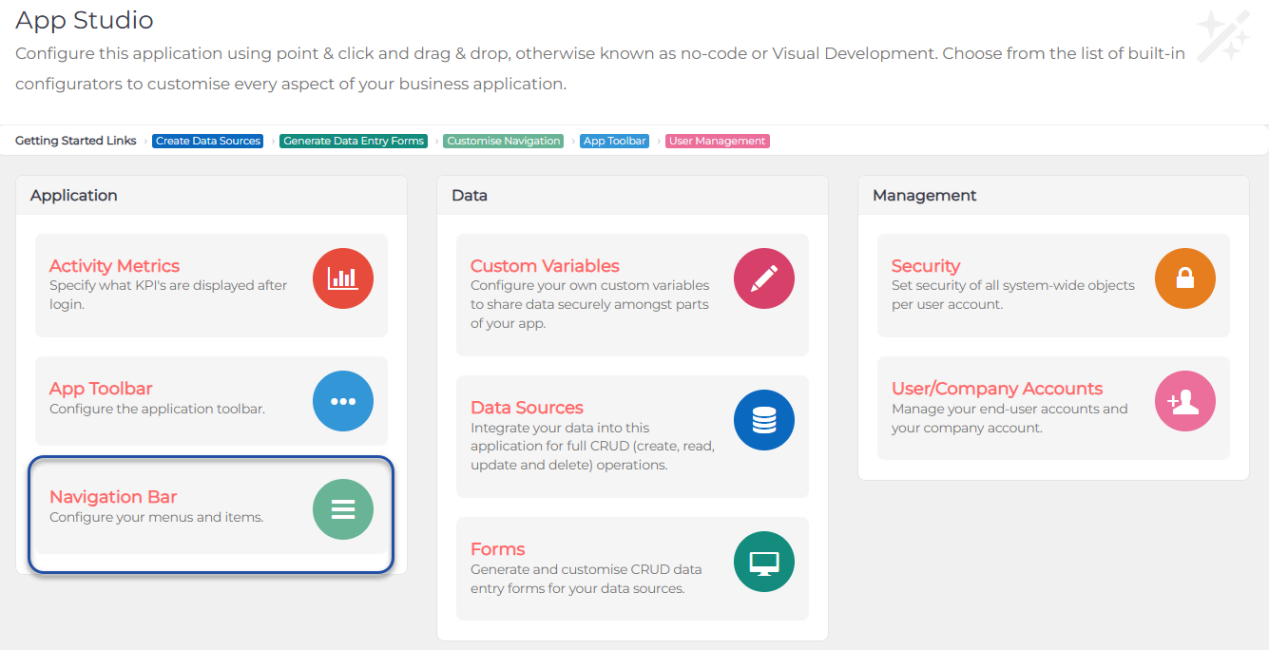
Developer Configuration

This tab is reserved for developers wishing to override custom behaviours.

Navigation Bar

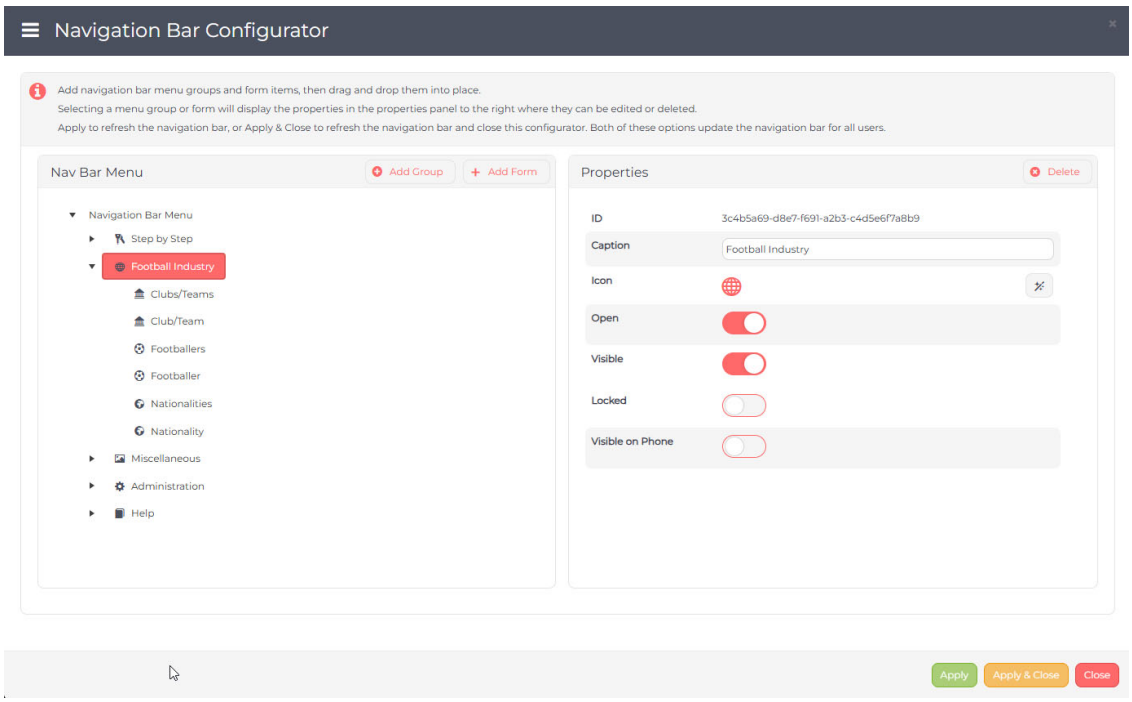
Configuring the navigation bar component

The Navigation Bar configurator is available from the App Studio.



Navigation Bar Configurator

The navigation bar configurator is a modal popup dialogue which shows the navigation bar menu items as a tree view on the left panel and the properties of each selected item in the right panel. It operates as master/detail so that selecting an item in the left panel tree view, shows the associated item properties to the right.



Nomenclature

Generically, a tree view consists of nodes arranged hierarchically. A node can be also described as an item. The top item, or node, in a tree is called the root. Nav Bar is a shortened term for Navigation Bar.

In the nav bar menu tree view the root is the Nav Bar Menu. It's sub nodes are called groups, and those items beneath groups are called forms, as they are used by the end-user to open forms.

Edit Functionality

The nav bar menu tree view is drag and drop enabled, allowing both group and form items to be dragged and dropped into other groups. There are only two levels beneath the root i.e. groups and form items.

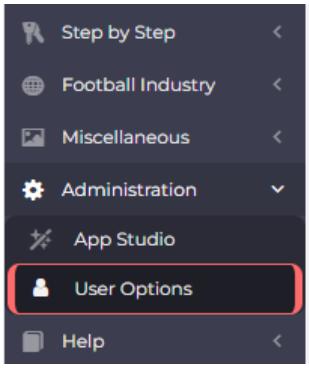
Changing group or form properties will be remembered when this form is open, so that re-selecting an item will remember the last change, however to persist your property changes you need to use the Apply or Apply & Close buttons, which re-draw the visible nav bar in the application for testing.

Forms Concept

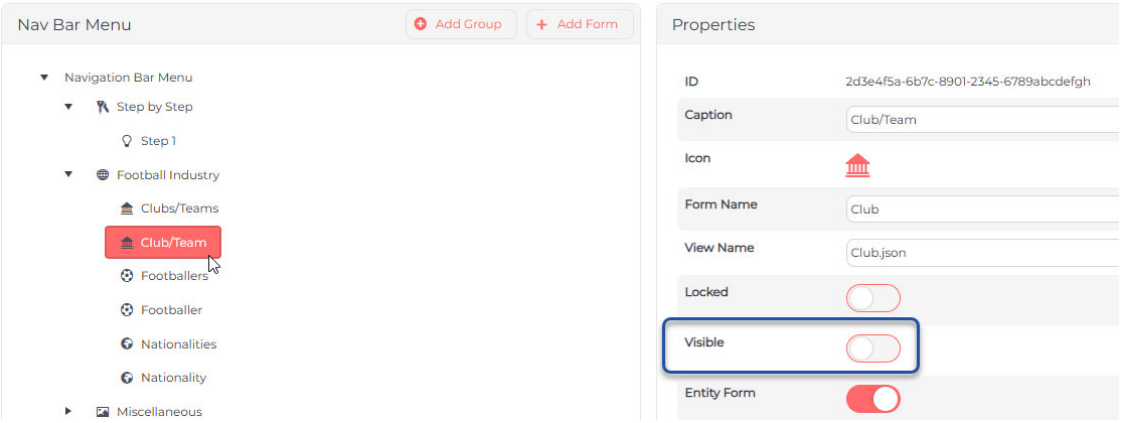
The lowest level, forms or form items, are each linked to a form. The nav bar is one of ways that end-users can open forms. The other ways are from the history menu and the add menu, both on the toolbar, and by drilling down into grids.

Note: The nav bar form properties must be defined in order to open any type of form (lookup or data entry). If a form is not referenced in the nav bar, then it cannot be opened.

When a form is opened, it will be highlighted in the nav bar for example:

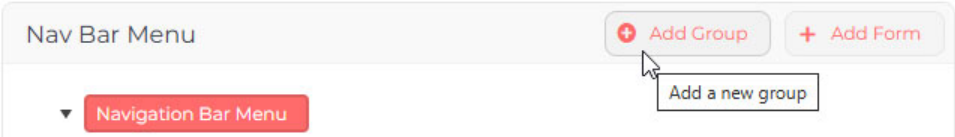


Data entry forms are also shown in the nav bar only when the form is opened. They should be set to invisible in the nav bar so that they only appear once they are opened. For example:

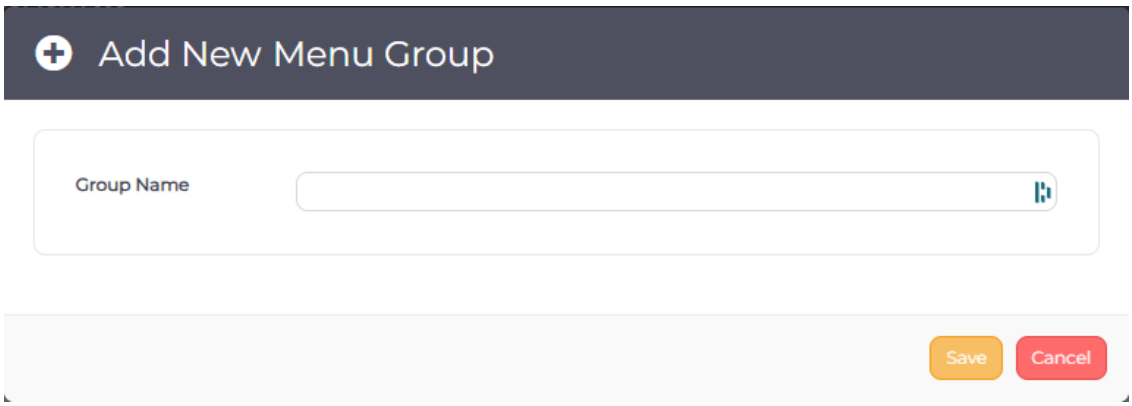


Add Group

The Add Group button on the Nav Bar Menu is where new groups are added to the Navigation Bar Menu root:



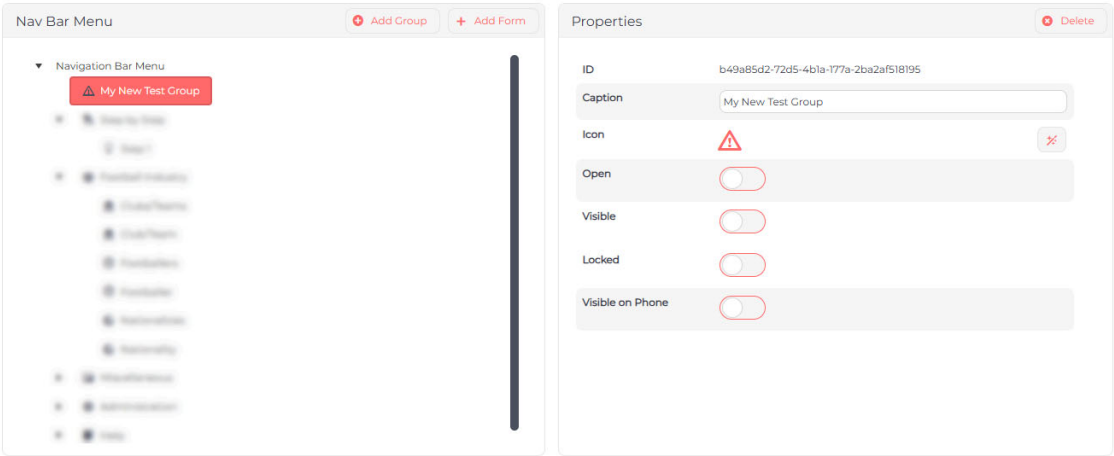
This opens a modal popup form requesting the name of the group:



Typing in the name of the group and using the Save button will add the new group to the nav bar and select it to show its properties.

Group Properties

When a group is selected in the left tree view, its properties are shown to the right:



ID

This is automatically generated and is read-only.

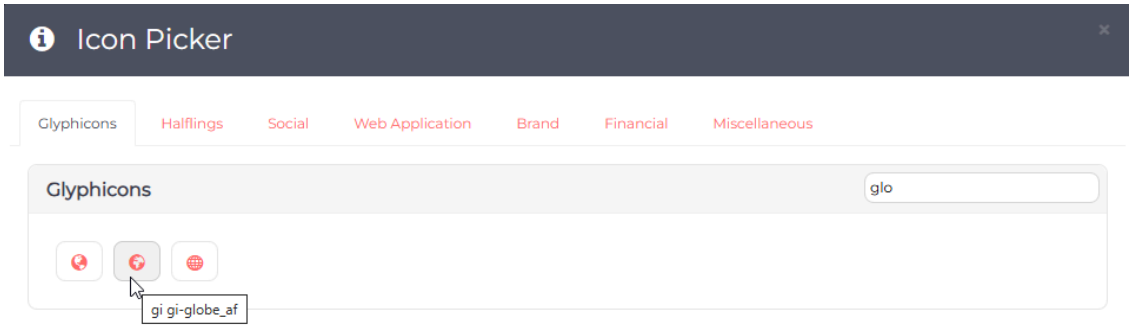
Caption

This is the text which appears in the nav bar group. Editing it will immediately change the text in the selected tree view group.

Icon

This is the icon which is shown in the nav bar to the left of the caption. The icon can be changed using the button to the right:

This opens the icon picker:



There are numerous categories of icons to choose from, and the filter can be used as shown. Clicking on the icon will replace it in the properties. It will also change in the selected tree view group.

Open

Setting this means that the nav bar will be shown open on login, showing its forms, for example:

This nav bar group is open, revealing the sub item forms.

Visible

Whether the group is visible or not when displayed at run-time. If you create a designer-only group, then you may wish to make this invisible, but use the security subsystem to make it visible only to specific users.

Locked

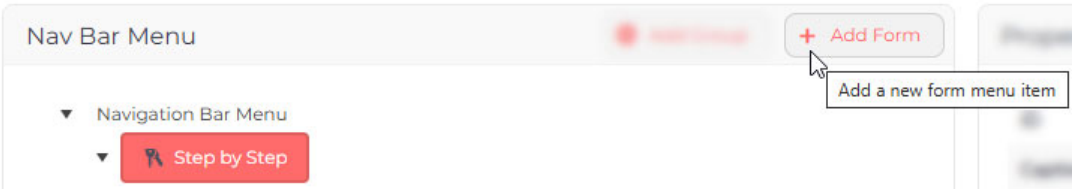
This read-only property means that the properties cannot be changed or deleted.

Visible on Phone

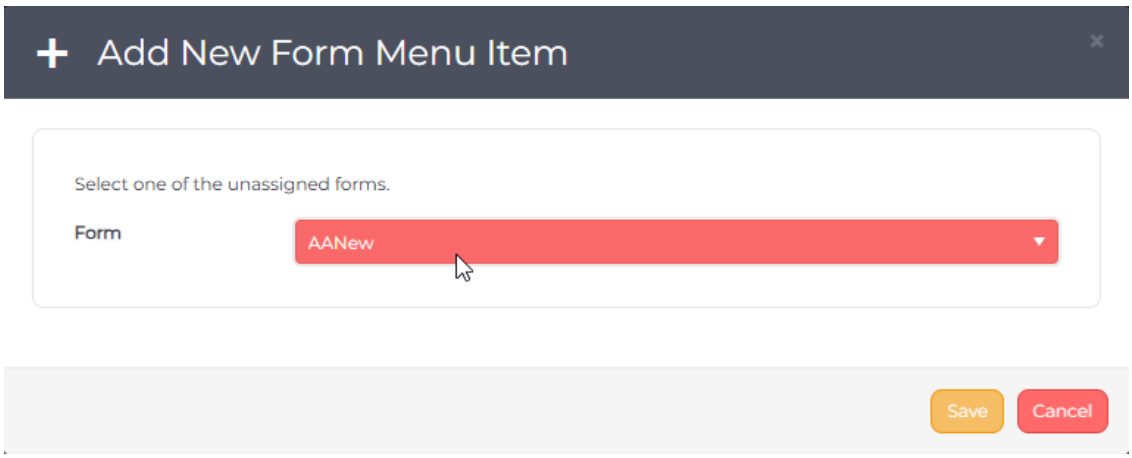
Some groups can be hidden on phones.

Add Form

The Add Form button on the Nav Bar Menu is where new forms are added to the selected form group:

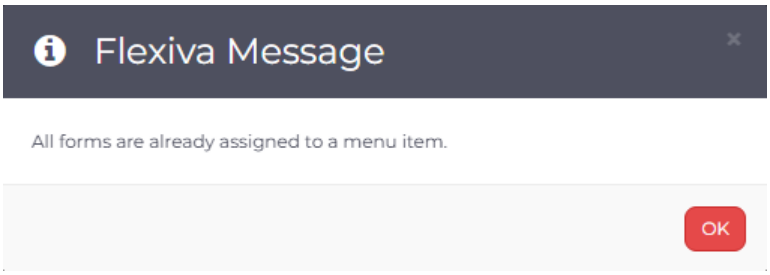


This opens a modal popup form requesting the selection of a form:



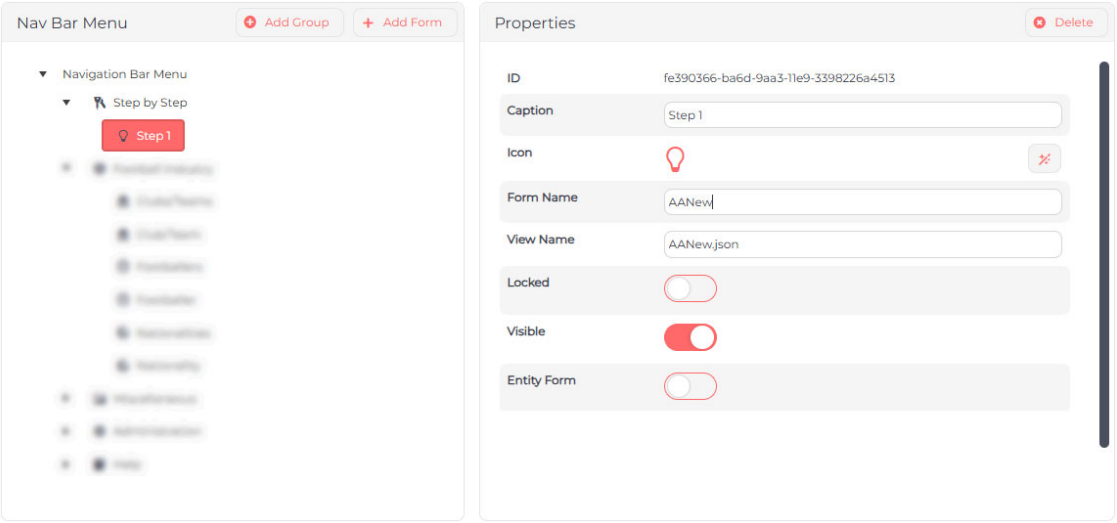
Click Save to add the new form item to the selected group.

WARNING: If you have already add all forms to the navigation bar, this message will be shown:



Form Properties

When a form is selected in the left tree view, its properties are shown to the right:



ID

This is automatically generated and is read-only.

Caption

This is the text which appears in the nav bar form item. Editing it will immediately change the text in the selected tree view form item.

Icon

This is the icon which is shown in the nav bar to the left of the caption. The icon can be changed using the button to the right.

Form Name

This is the read-only name of the form.

View Name

This is the read-only name of the JSON form file.

Locked

This read-only property means that the properties cannot be changed or deleted.

Visible

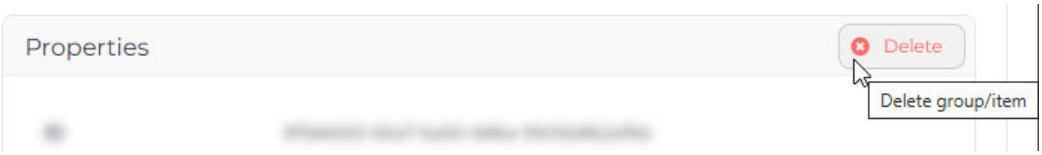
Whether the form item is visible or not when displayed at run-time. You should make data entry forms invisible at run-time. This will not prevent the form being opened, but will prevent the user from opening it from the nav bar.

Entity Form

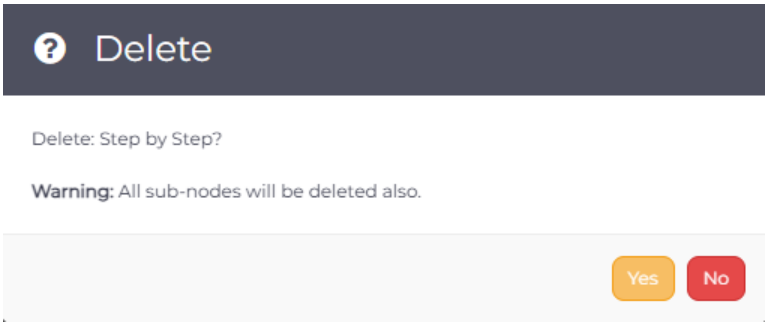
An entity form is a data entry form for CRUD operations. Set this accordingly.

Delete

The delete button appears to the right of both group and form properties:



You will be prompted to confirm the deletion:



WARNING: Deleting a group will delete all form items beneath also.

Apply

Use the Apply button to save your changes and keep the form open.

Apply & Close

Use the Apply & Close button to save your changes and close the form.

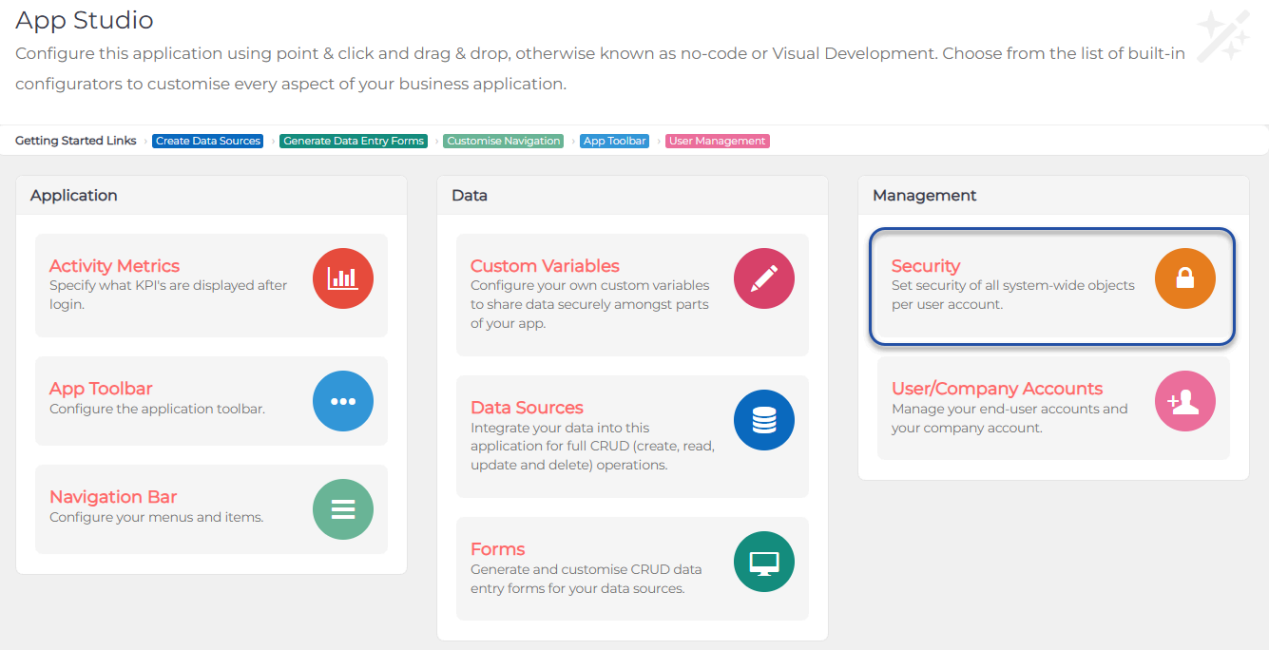
Close

Use the Close button to disregard your changes and close the form.

Security

Configuring the user security.

The Security configurator is available from the App Studio.

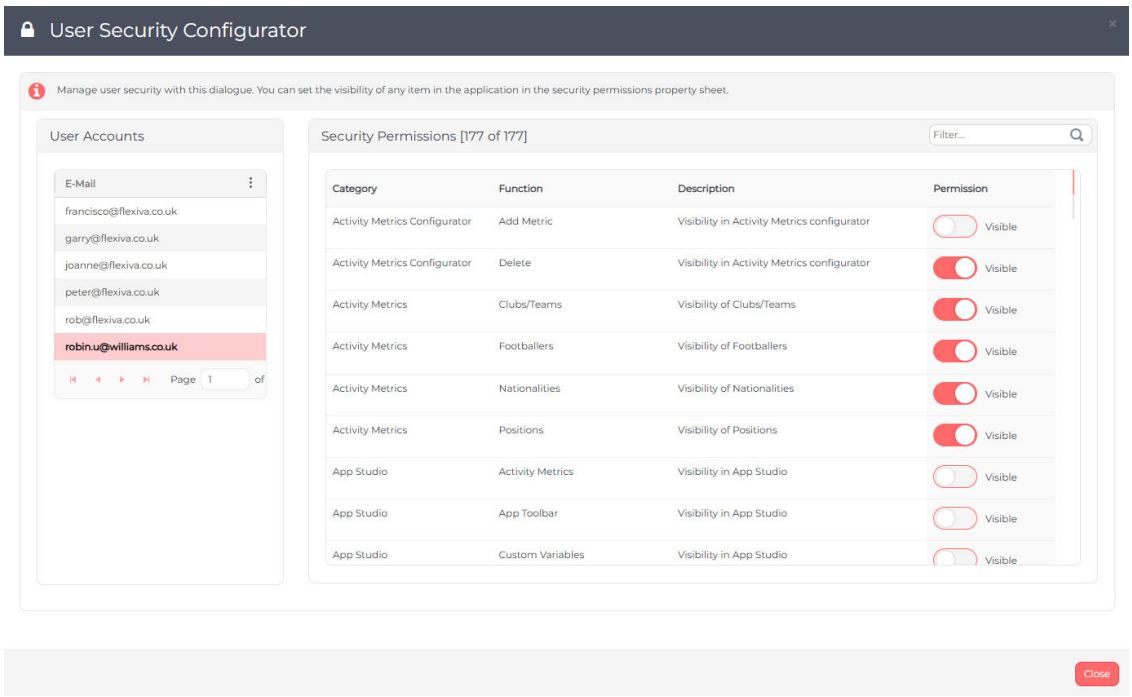


Security Concept

Security applies to 'permissions', or availability of functionality within the application for both designers and end-users. Typically, all UI elements such as nav bar groups, forms, buttons, menus, fields can be either hidden or disabled for specific users, thus securing that functionality to be used only by authorised personnel. Only designers can set security for end-users.

User Security Configurator

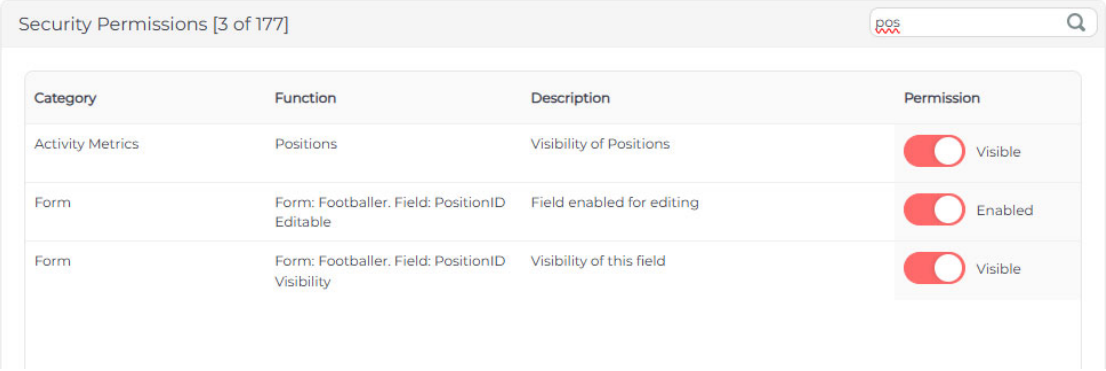
The form consists of a list of user accounts on the left panel, and the selected users security permissions on the right:



Selecting a user will highlight their account and load their respective security permissions.

Filter

There may be hundreds of security permissions, so using the filter text box top right will make setting permissions much easier for example:



The filter applies even when selecting different user accounts on the left.

Permissions

There are 4 columns.

Category

The category is a grouping of permissions. Often the category is the name of the form, GUI elements or its configurator for example "Activity Metrics" and "Activity Metrics Configurator" applies to the side panel KPI's but also the functionality of the respective configurator.

Function

This is the specific function inside the category, for example a menu item, button or field.

Description

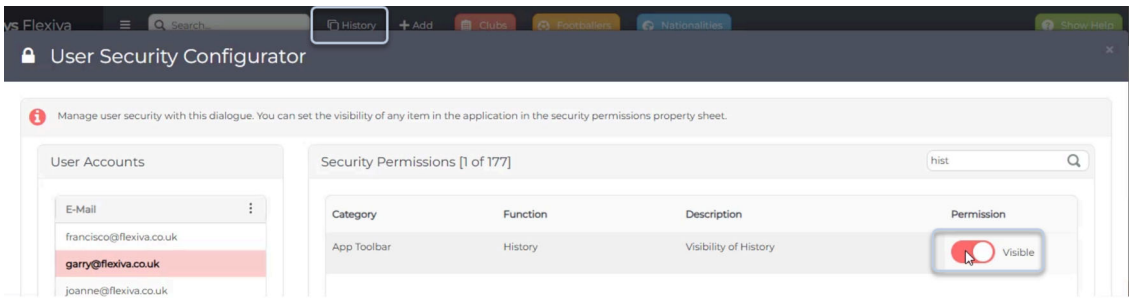
This is the description of the permission. Usually visibility or enabled status.

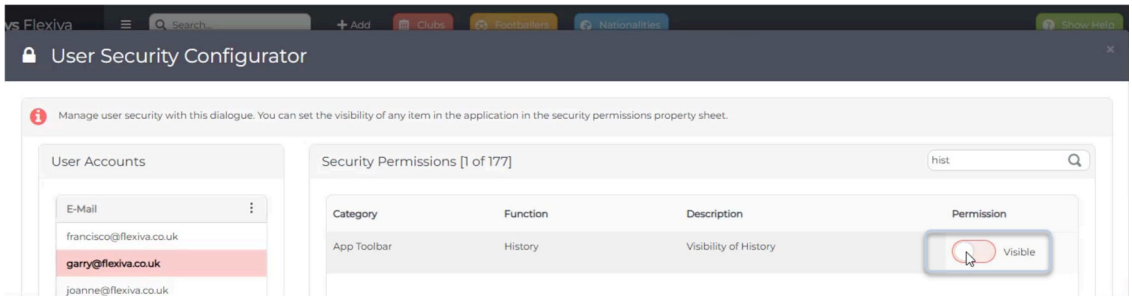
Permission

The permission of the category/function is a check box for either Visible or Enabled. If Visible is unchecked, the element will not be visible. If Enabled is unchecked, the element cannot be clicked or edited.

Persistence

Changing the checkbox immediately changes the permission and if you are changing permissions for yourself, then these changes are reflected in your application immediately, for example:



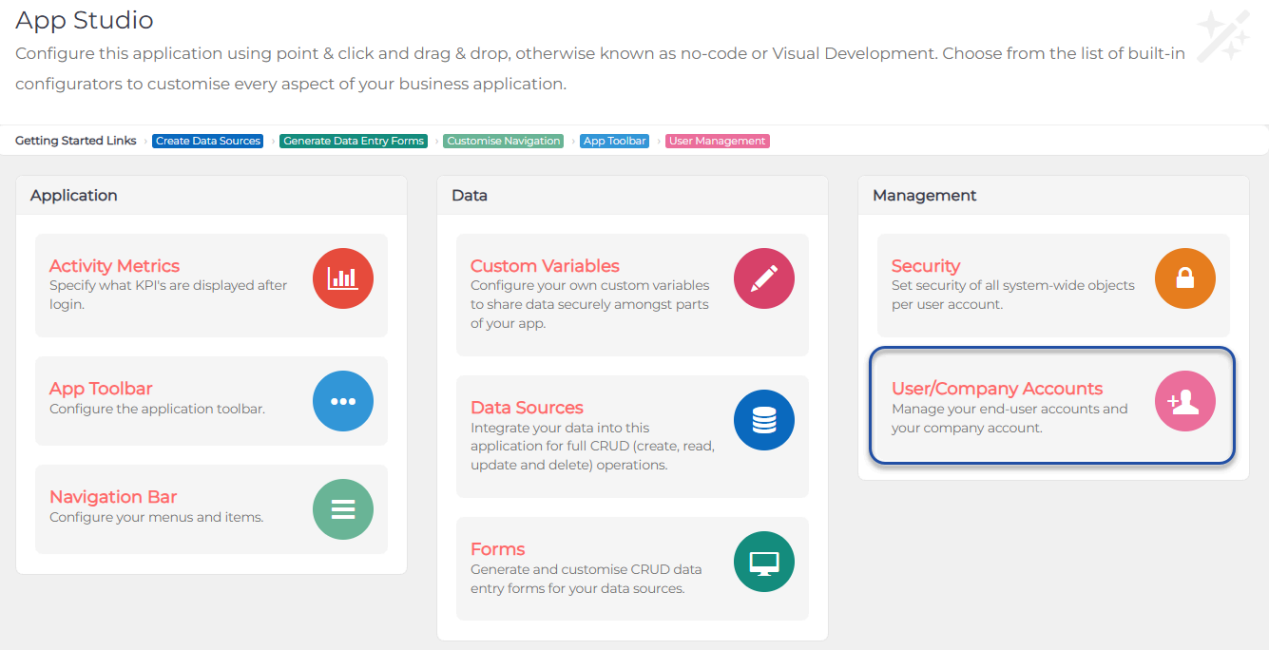


Notice how the History menu disappears when Visible is unchecked?

User/Company Accounts

Configuring the user/company accounts.

The User/Company Accounts configurator is available from the App Studio.



Account Concept

There are two types of user account:

Designer

If you are a designer, then you can use App Studio to design your application with 'administrator' permissions i.e. you can change everything unless restricted by security permissions, using the numerous configurators. When signing up, you were assigned the role of designer.

End-User

An end-user, or user, is restricted from designing the application, but can use all the forms, nav bar, toolbar etc.. functionality for CRUD operations unless restricted by security permissions.

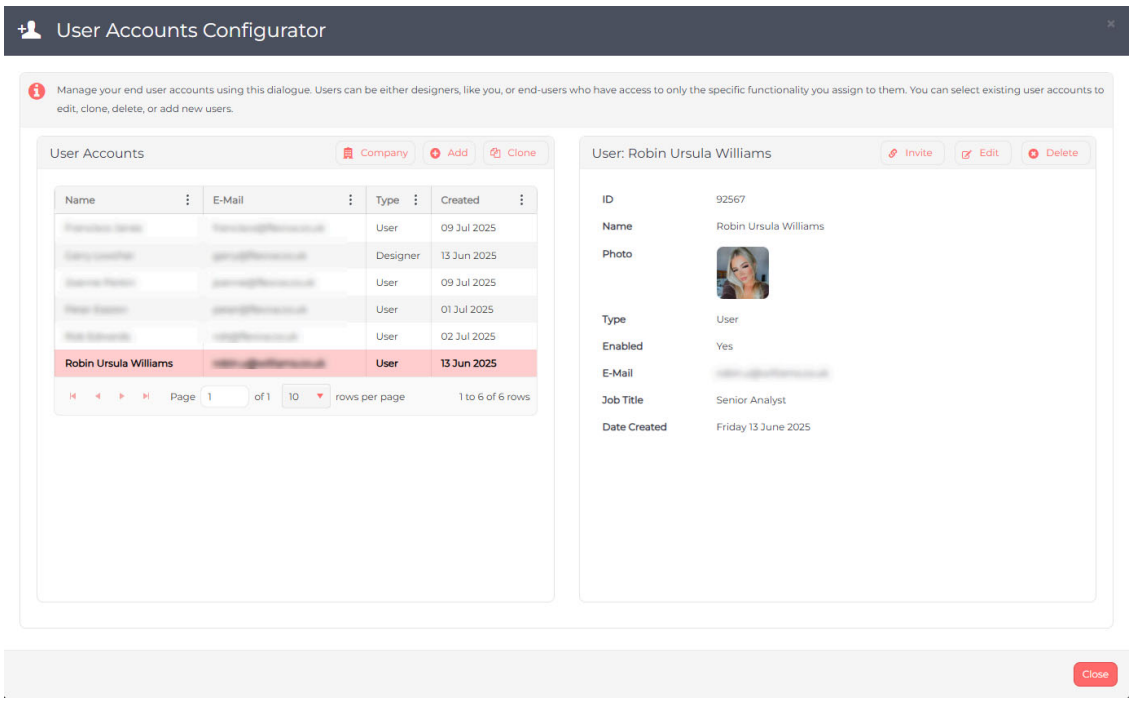
All user accounts are linked to the same company, but only designers can update the company details, and only under certain conditions.

User Custom Variables

When integrating with ReST API's from multiple back-ends, each back-end may have their own user account. By creating a user custom variable for each back-end, we can associate the Flexiva user account with each of these so that each ReST API can pass the back-end user identifier.

User Accounts Configurator

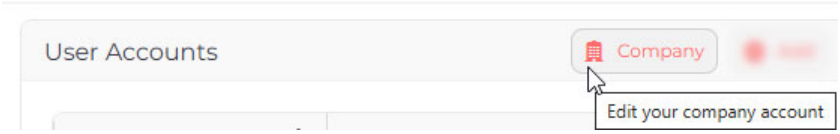
Clicking the User/Company Accounts from App Studio opens this modal form:



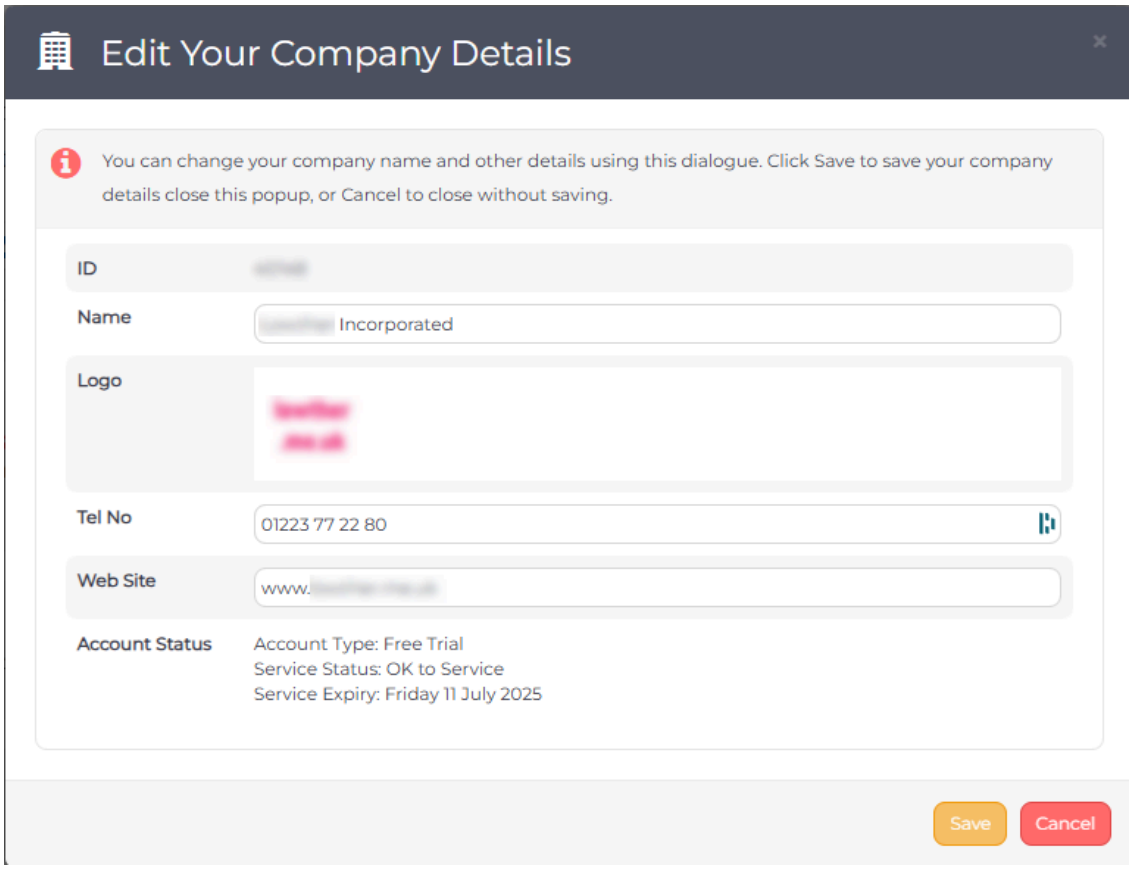
The user accounts are listed on the left panel/grid, and the selected user details are shown on the right panel.

Company

The company button is shown in the left panel:



It is used to edit your company account when clicked:



Fields are as follows:

ID

The internal read-only ID of your company record in our CRM database. This will not change.

Name

Your company name. Companies can change names, so this field is editable, however we will not permit you to change a name which already exists in our CRM database. Please contact us if you can prove that your company name change is valid in this instance.

Logo

This is your company logo image file. Click this to view, upload or delete this image.

Tel No

This is your company telephone number.

Web Site

This is your company web site URL.

Account Status

This is the status of your account.

Status	Description
Account Type	Whether you are on a Free Trial or are a Paid Customer.
Service Status	Whether you are OK to Service or otherwise.
Service Expiry	When your service contract expires.

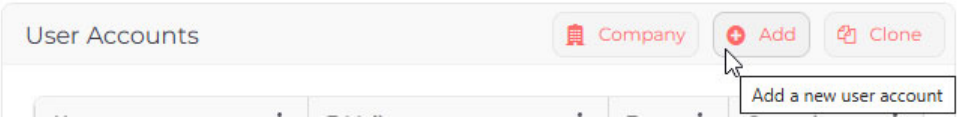
Save

Use this button to save your changes.

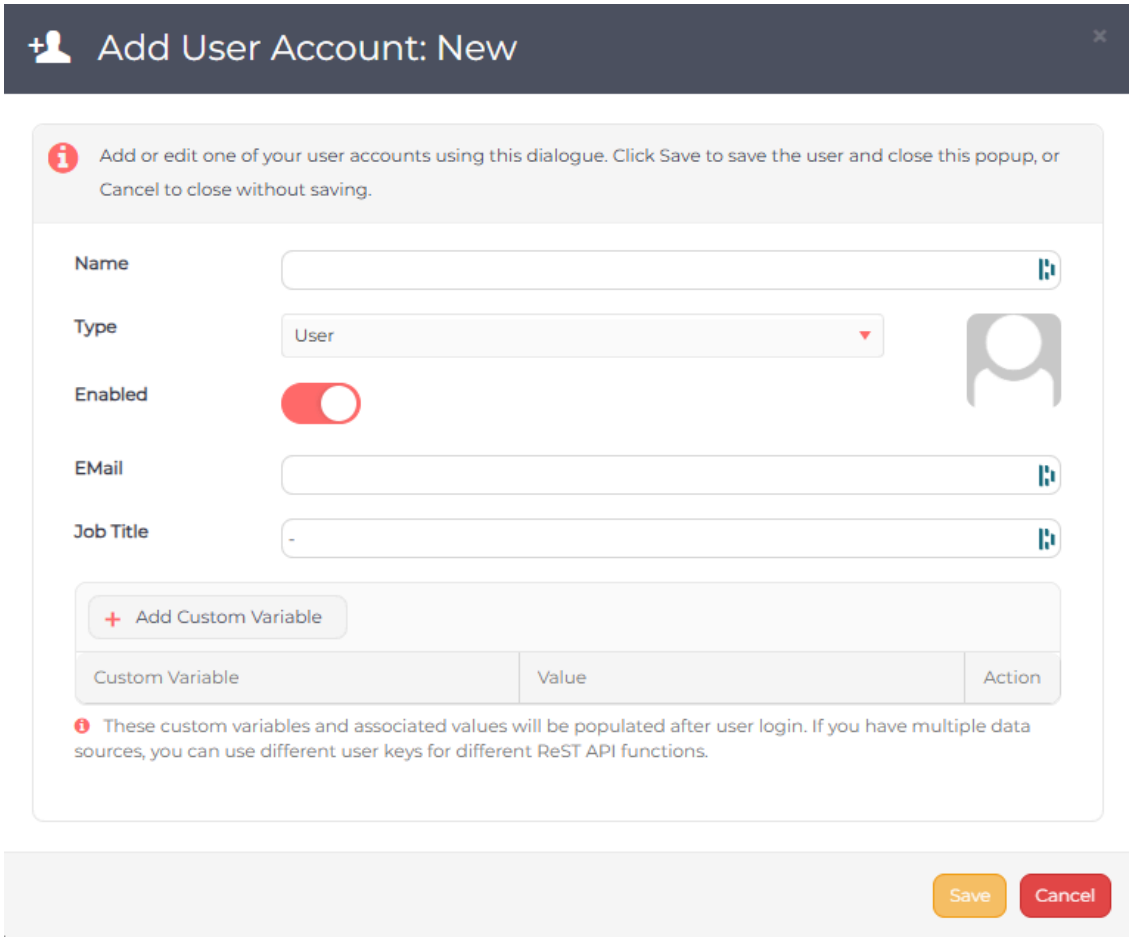
WARNING: If your new company name conflicts with an existing company name in our CRM system, then you will be informed and your change will be rejected. Please contact us to resolve this issue.

Add User Account

The add button on the left panel allows a new user account to be added:



This opens this modal popup:



Name

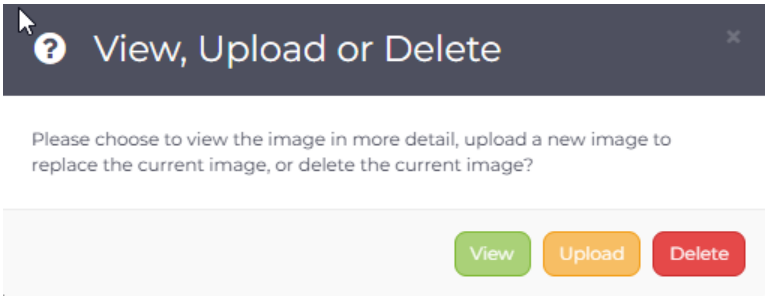
The mandatory name of the new user.

Type

A drop down combo. The only two choices are "User" and "Designer" as explained here.

Photo

This is displayed to the right of the Type field. Click the image to either view it, upload a new image, or delete the image if previously uploaded.



E-Mail

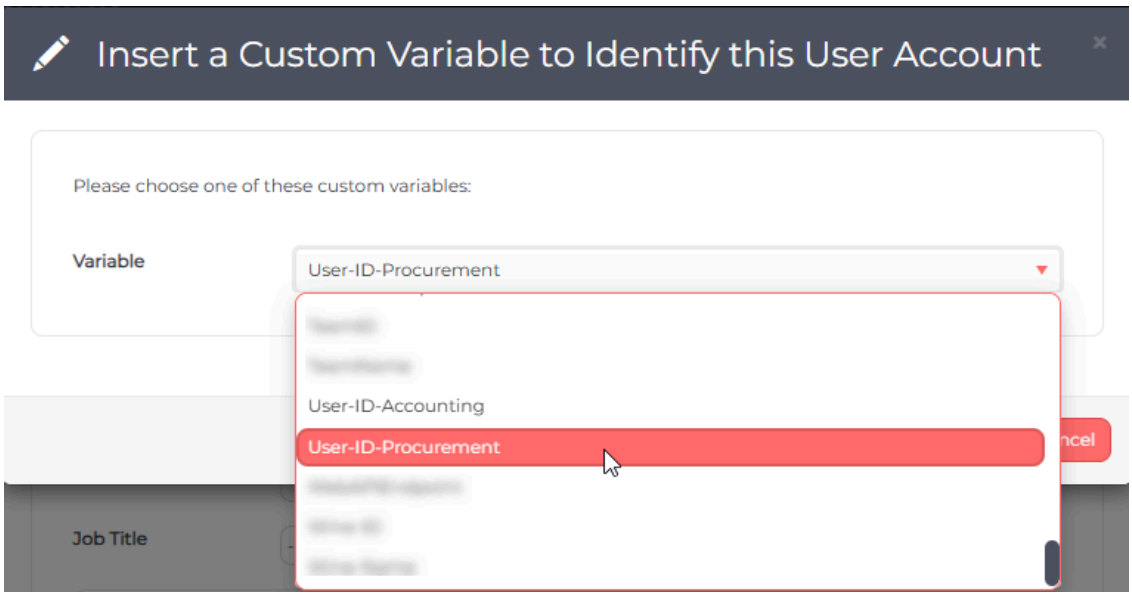
This is the mandatory e-mail address of the new user.

Job Title

The optional job title of the new user.

Add Custom Variable

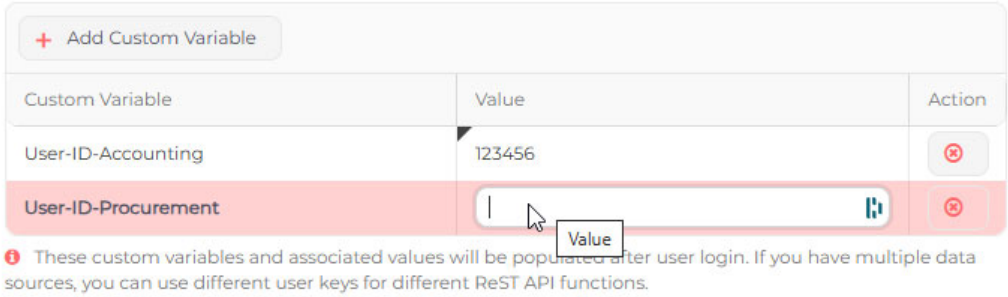
Add one or more user custom variables to this user account using this button. This modal popup form opens:



Choose the custom variable in the drop down combo.

Edit Custom Variable Value

Once the necessary custom variables are added, the grid can be used to set the value of this specific user account variable:



This means that when this user logs in, their respective values will be assigned to their client-side custom variable and are therefore automatically passed to the ReST API data source request so that the back-end can identify which user conducted the CRUD transaction.

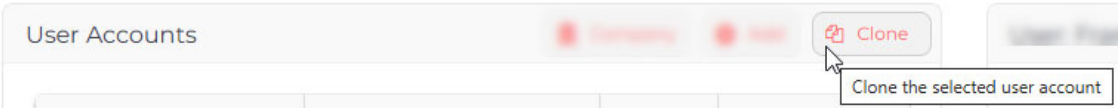
Save

The save button will create the new user account and refresh the grid and highlight the new user.

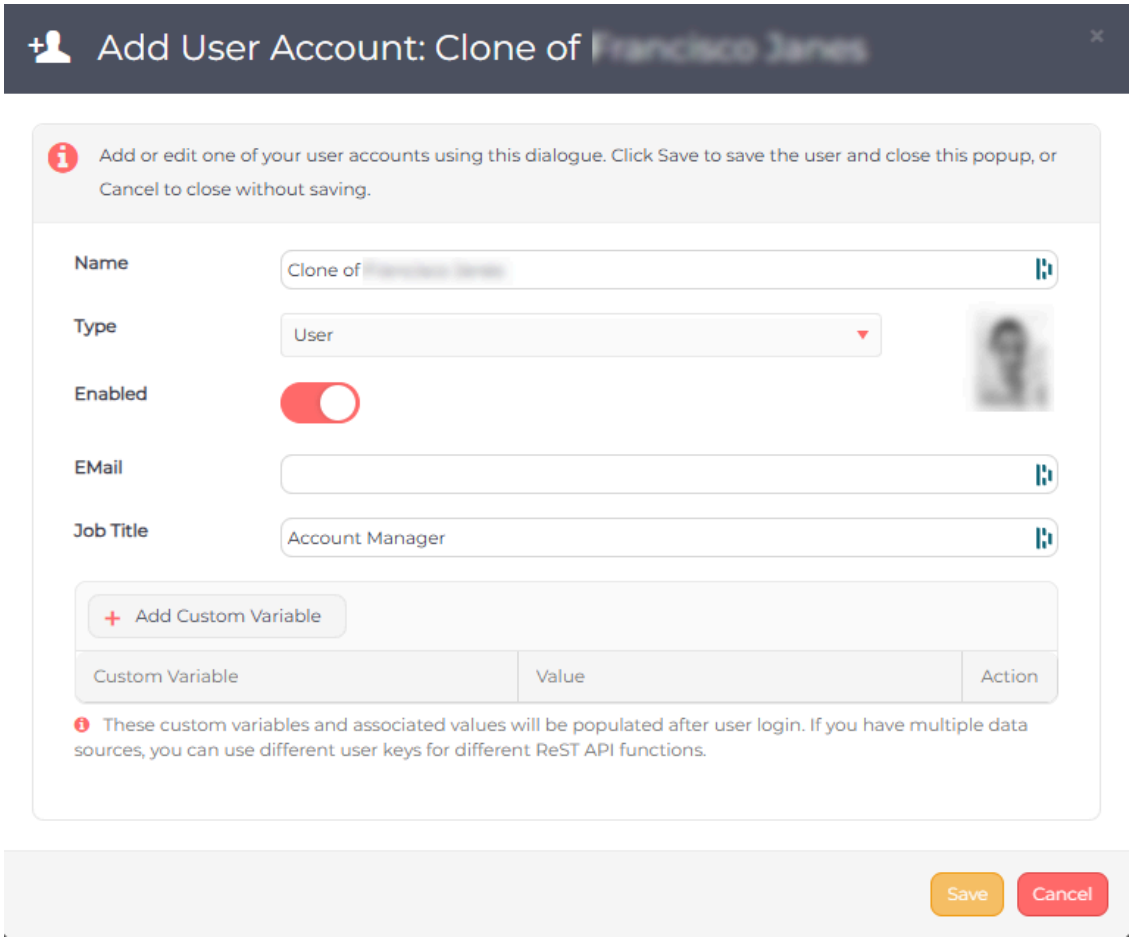
Clone User Account

Cloning an existing user account is the preferred mechanism for creating new user accounts. This is because it also clones the user security permissions.

The clone button on the left panel allows an existing user account to be copied to create a new user with a different name and e-mail address:



The modal popup form is shown:



The name of the contact being cloned is prefixed by "Clone of " and the same photo is displayed to remind you which user account you are cloning.

Change the Name, add the e-mail address, upload a new Photo, set any user custom variables then Save the cloned user account.

Sample App

Introduction

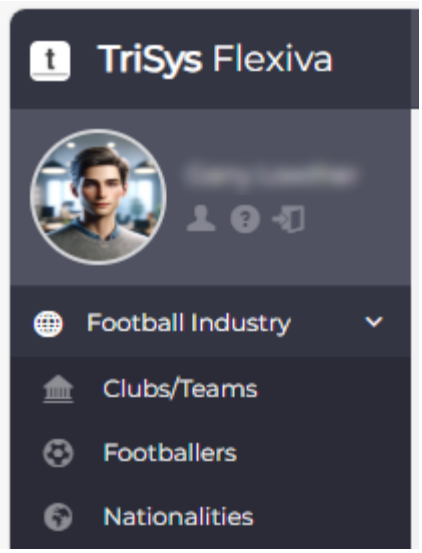
Sample reference app built to demonstrate the process of connecting a third party ReST API to Flexiva and using App Studio to produce a working web and mobile application.

The Flexiva line-of-business (LoB) application is connected to a Representational State Transfer (ReST) Application Programming Interface (API) hosted at restdb.io to allow you to use forms to create, read, update & delete sample footballer data freely available in the public domain.

The forms designed for this sample application all have F1 help and users can click the links to navigate to this series of web pages specifically written to reflect the sample forms.

This approach demonstrates how Flexiva customers can implement their own business-specific F1 help and user-guides for their own custom forms.

The footballer industry forms can be opened from the navigation bar to the left of the application. There is a separate group entitled "Football Industry":

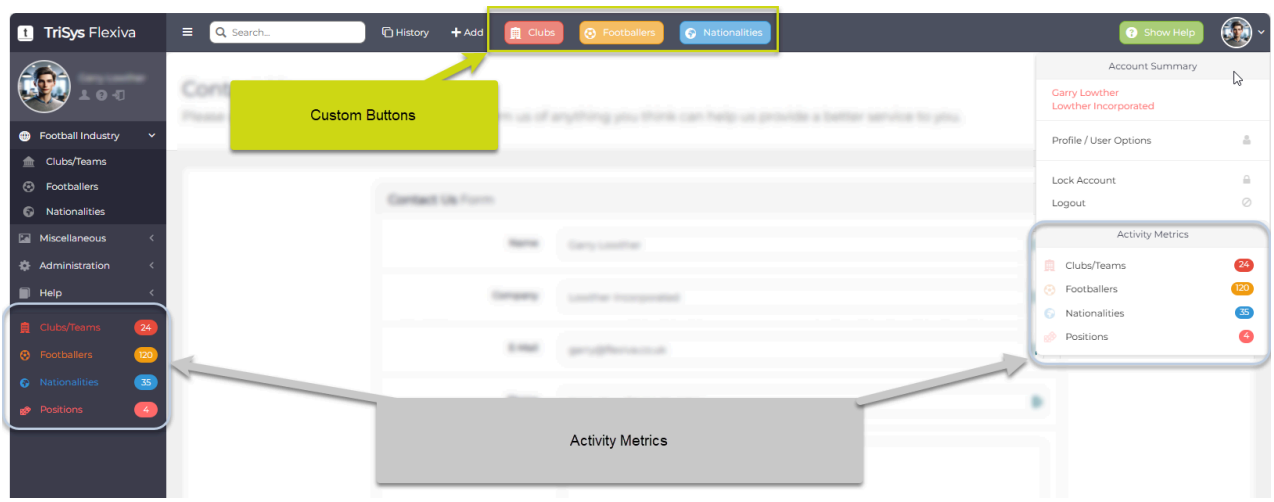


Beneath this group are 3 lookup forms:

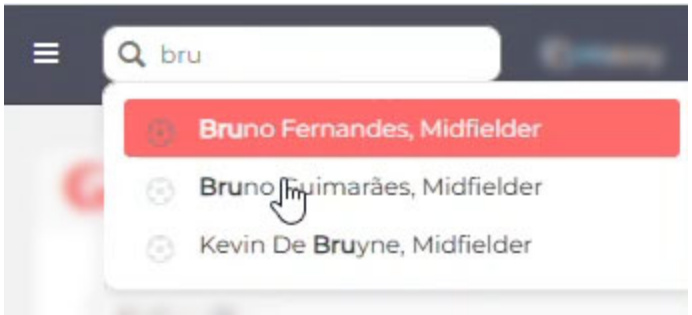
- Clubs/Teams
- Footballers
- Nationalities

All of the custom buttons, activity metrics and navigation bar were designed by a non-developer in the Flexiva App Studio. Each of the custom buttons and activity metrics are linked to ReST API's to display data specific to the business for the end-user community.

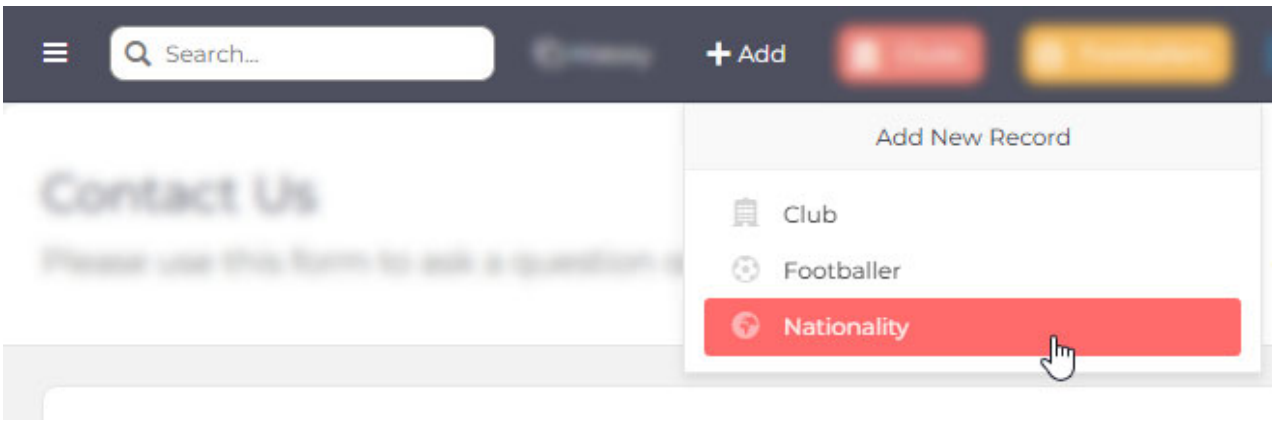
The visibility of all custom functionality can be turned on and off for each user, thus allowing for specific user groups to access or update/delete only information for which they are authorised.



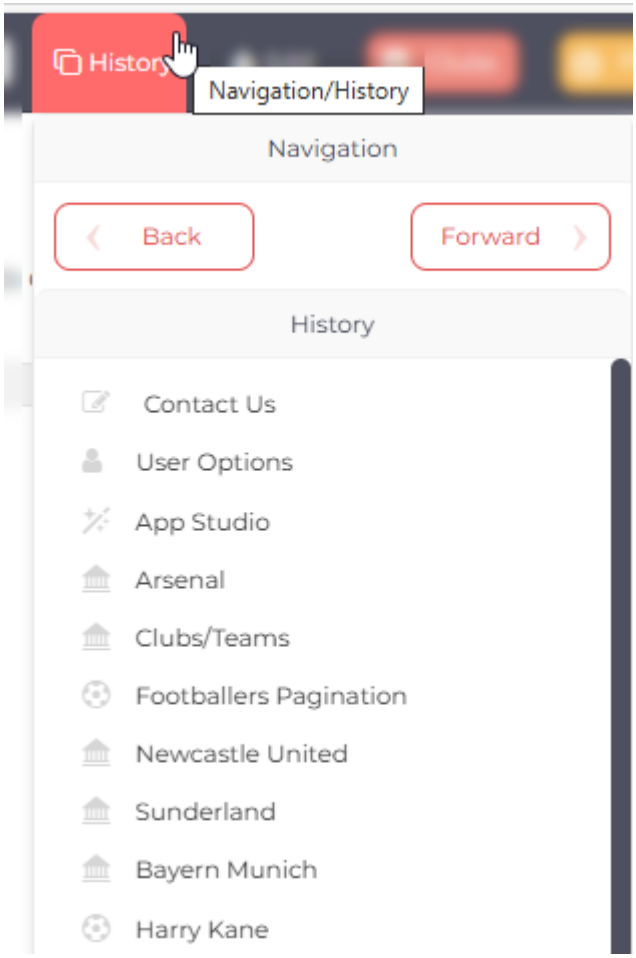
The **Search...** text box is linked to a specific ReST API which permits searching over the entire company specific data set. Users can quickly lookup records which drop down in a list for subsequent drill-down.



The **Add** menu allows end-users to add new records for forms connected to the CRUD ReST API's.



The **History** menu shows the list of all forms which were recently opened. End-users can click on each item to re-open that specific form record.



Clubs/Teams Lookup

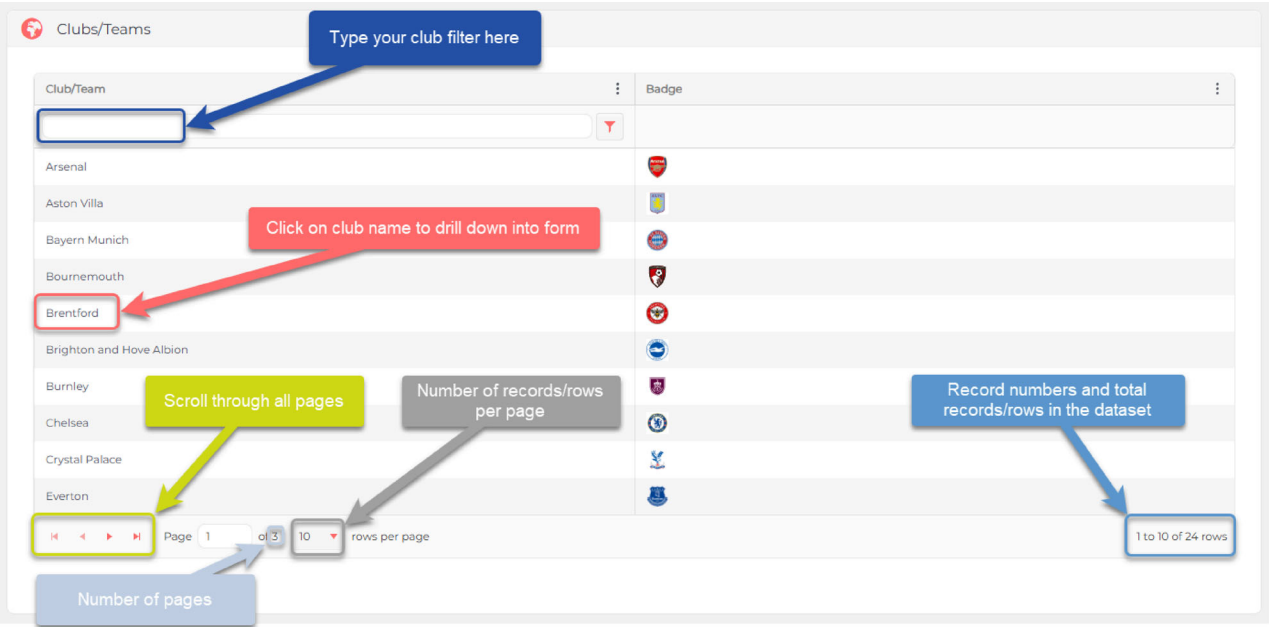
Use this form to lookup football clubs or teams. Each footballer plays for a club.

The grid reads all of the clubs/teams in the associated ReST API dataset and presents them in tabular format. This dataset contains the name and badge image of the club.

You can type text into the filter row for the Club/Team to narrow down your lookup.

The grid footer tells you how many clubs there are in this grid. Each club is a row, or a record in database language. Each group of records is effectively a page, and the number of records/rows per page can be changed to display more data as required.

Clicking on the club name will 'drill-down' and open the full data entry form where you can see further details ↗ about the club.



Club/Team Form

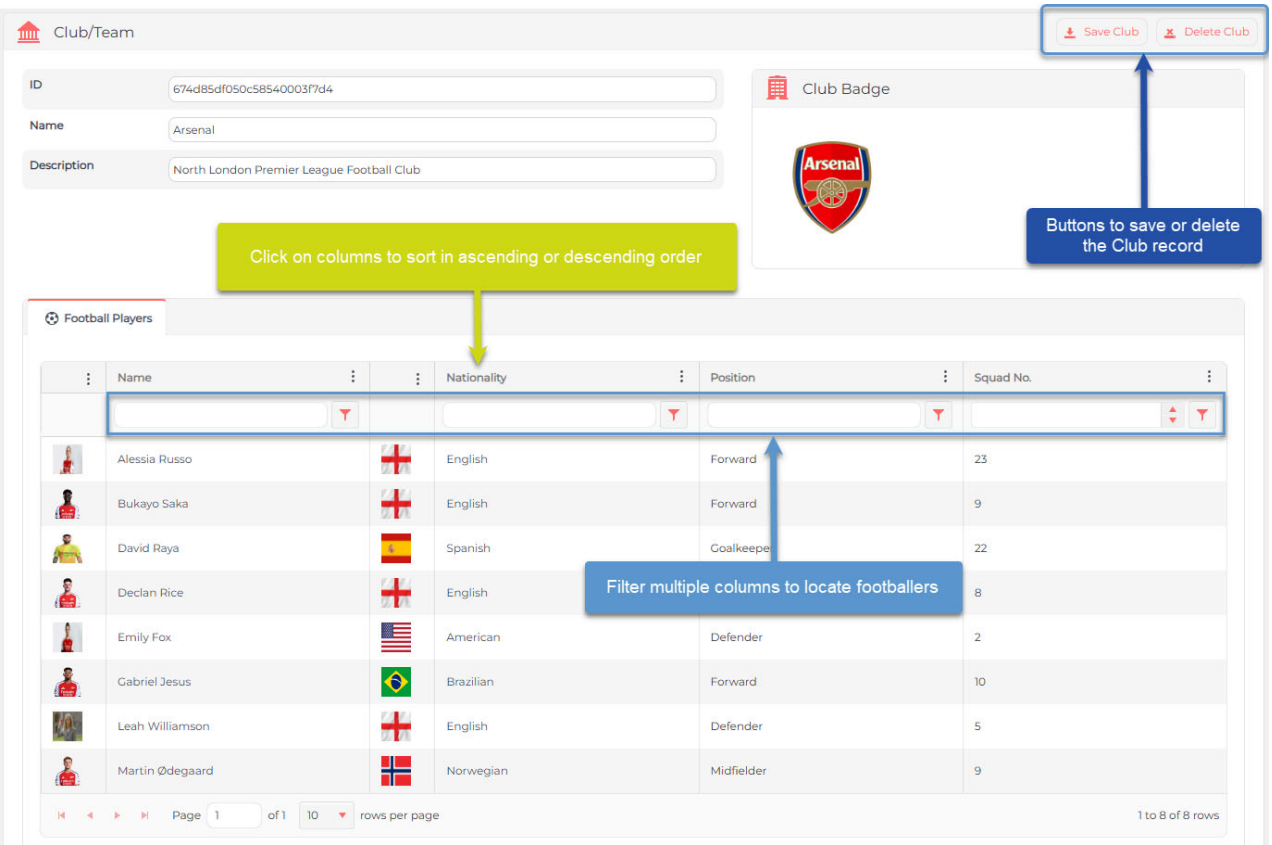
Use this data entry form to view, create, update or delete a club.

The Name and Description are text fields and the club badge is an image which can be clicked to either open the image larger in a popup or replace the image by uploading a new image, or deleting the image.

The top right form buttons respectively allow you to update or delete the record via the ReST API.

The footballer players grid reads all of the footballers in the associated ReST API dataset who play for this club/team and presents them in tabular format. This dataset contains the name, nationality, position and squad number and associated images.

Clicking on the player name or the nationality will 'drill-down' and open the full data entry form for the record type selected.



Footballer Search

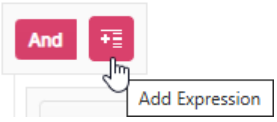
Use this form to search for footballers.

There are two components which have been placed on this form using the App Studio forms designer.

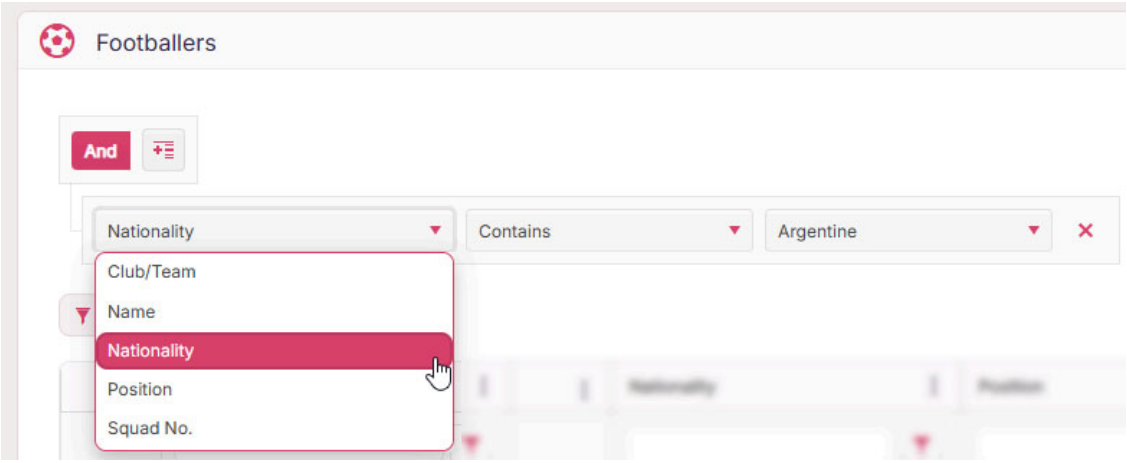
Filter Footballer Criteria

This component is linked to 3 ReST API data sources which are used to populate the drop down combo/list for each of the lookup fields: Club, Nationality and Position.

Expressions, or fields, can be added using this button:



The 4th field is a text field allowing end-users to search for a footballer by name. The 5th field is a numeric field allowing end-users to search for a footballer by squad number. Users can uses as few or as many filter fields are they require. The Apply button populates the grid.



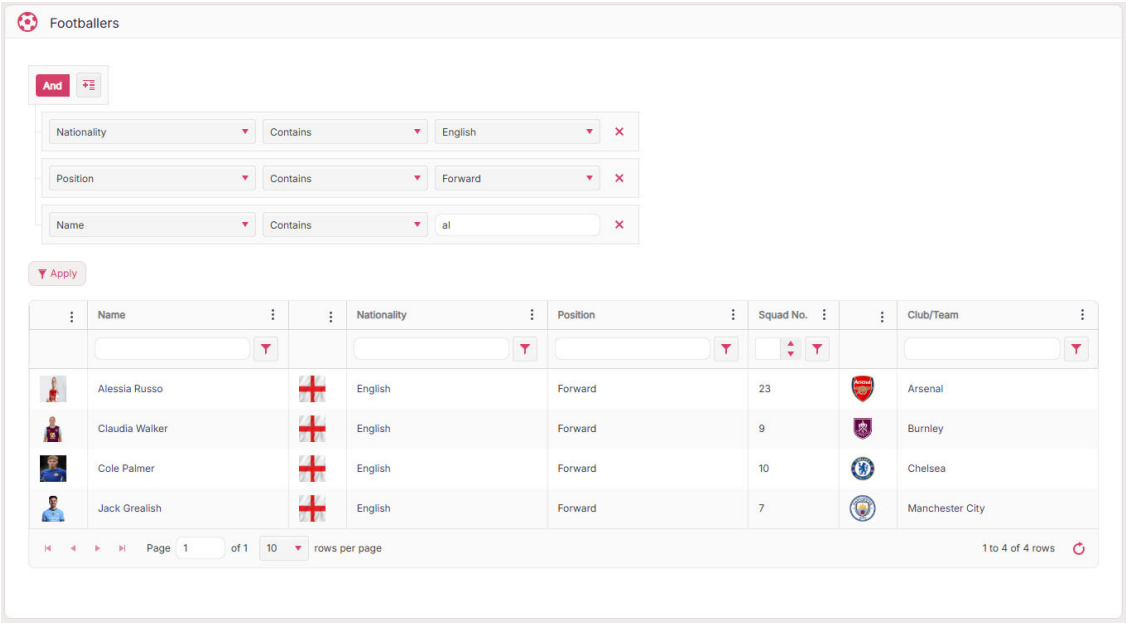
Footballers Grid

The grid reads all of the footballers/players in the associated ReST API dataset and presents them in tabular format. This dataset contains the name and photo of the player, nationality and flag and club and badge, together with position and squad number.

You can type text into the enabled filter rows to narrow down your lookup.

The grid footer tells you how many footballers there are in this grid. Each footballer is a row, or a record in database language. Each group of records is effectively a page, and the number of records/rows per page can be changed to display more data as required.

Clicking on the footballer name will 'drill-down' and open the [footballer form](), likewise clicking on nationality or club will drill-down into their respective forms also:



Footballer Form

Use this data entry form to view, create, update or delete a football player.

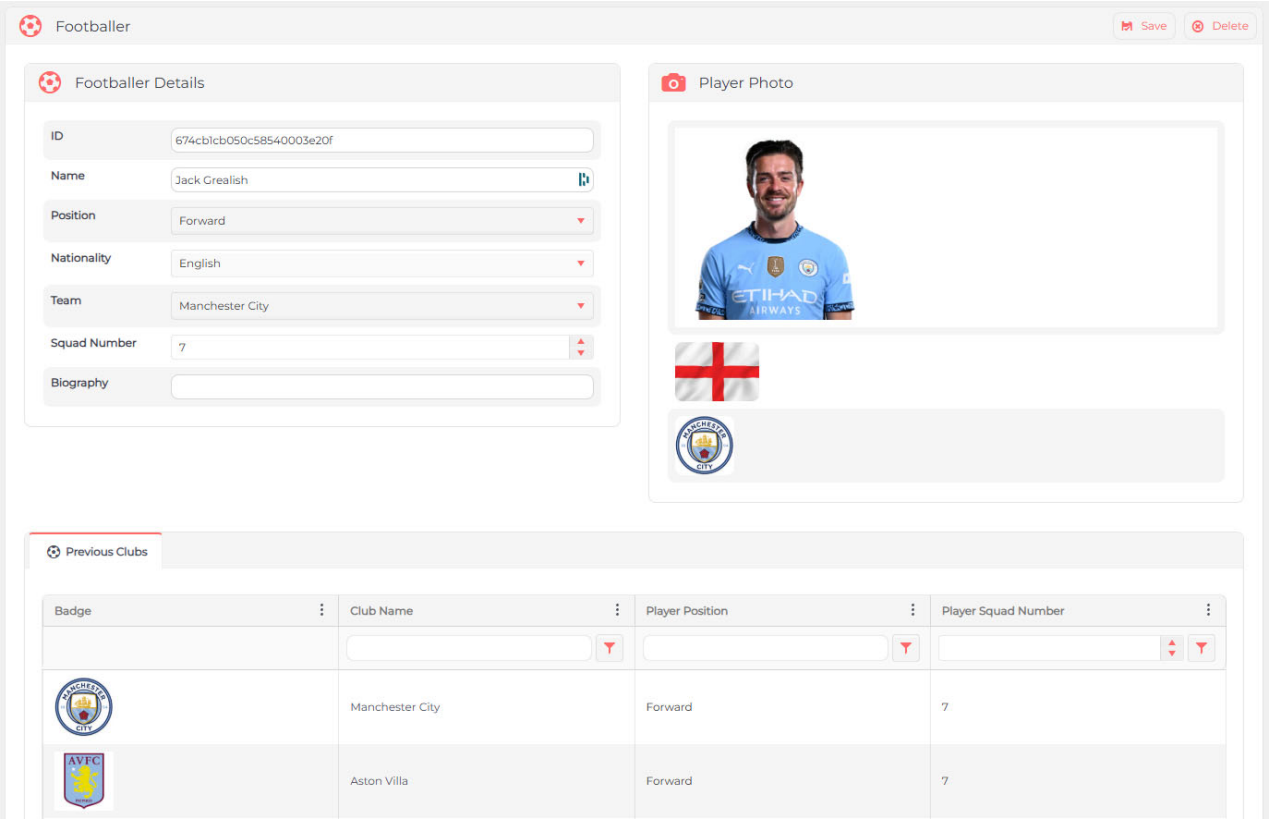
The Name and Biography are text fields, and Position, Nationality and Team are drop down combos linked to respective ReST API's. The Squad Number is an integer field.

The player photo, nationality flag and club badge are images which can be clicked to either open the image larger in a popup or replace the image by uploading a new image, or deleting the image.

The top right form buttons respectively allow you to update or delete the record via the ReST API.

The Previous Clubs grid reads a list of clubs where this footballer played. This dataset contains the badge, name, position and squad number.

Clicking on the Club Name will 'drill-down' and open the full data entry form for the record type selected.



Nationalities Lookup

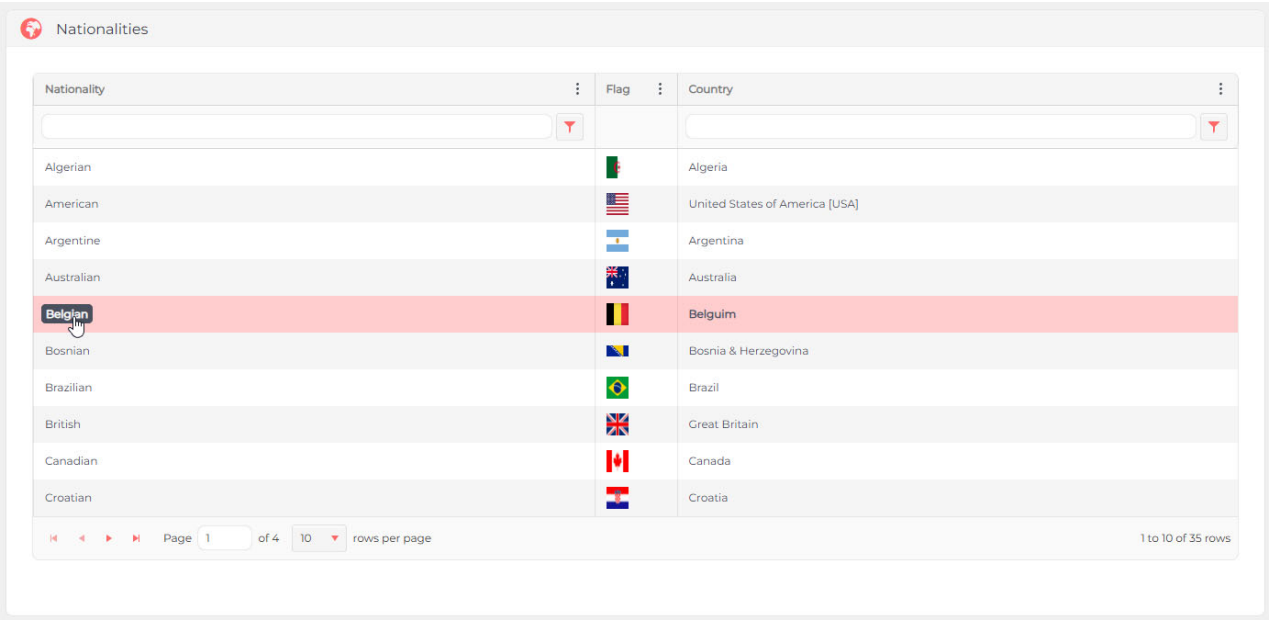
Use this form to lookup nationalities/countries. Each footballer is linked to a nation.

The grid reads all of the nationalities in the associated ReST API dataset and presents them in tabular format. This dataset contains the nationality, country and flag image.

You can type text into the filter row for the Nationality or Country to narrow down your lookup.

The grid footer tells you how many nationalities there are in this grid. Each nationality is a row, or a record in database language. Each group of records is effectively a page, and the number of records/rows per page can be changed to display more data as required.

Clicking on the Nationality or Country will 'drill-down' and open the Nationality form.



Nationality Form

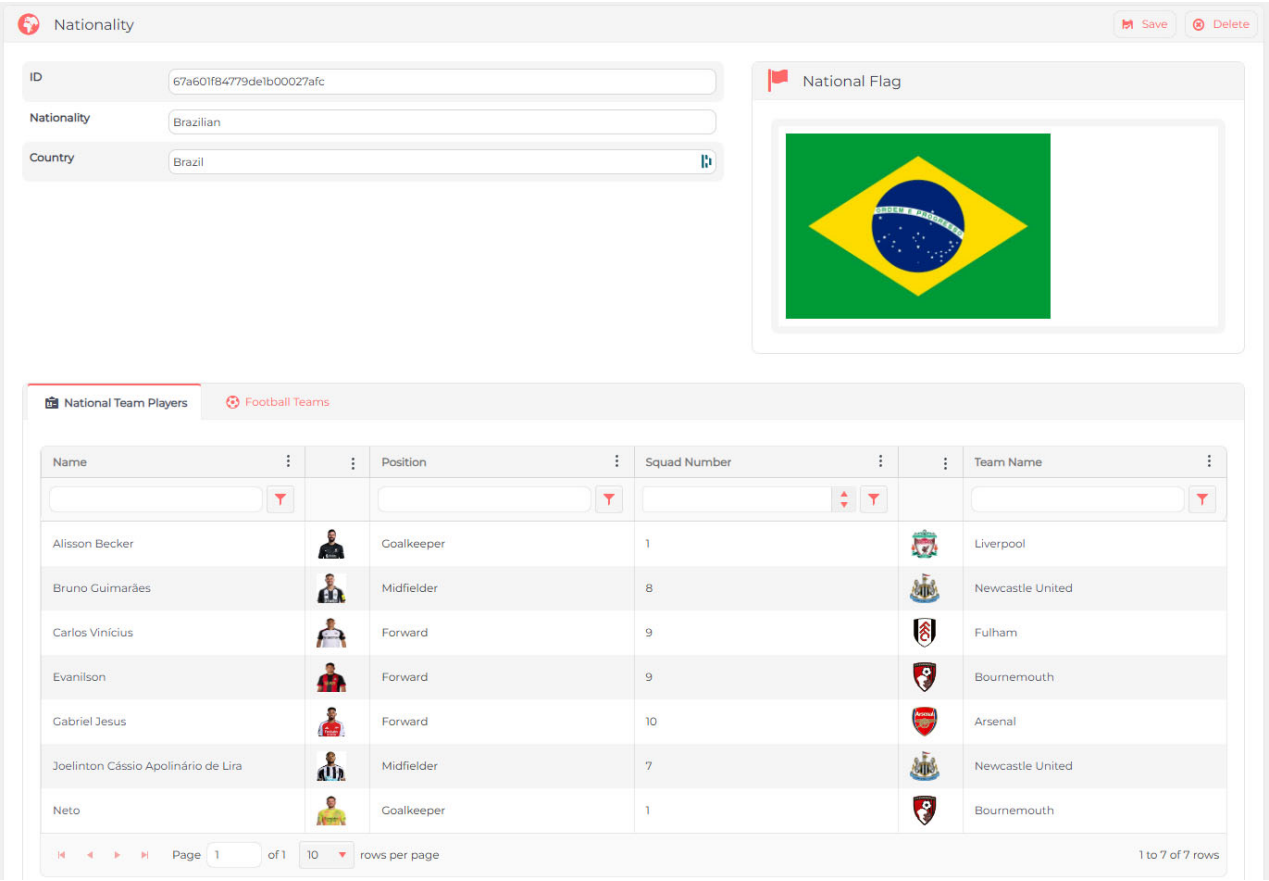
Use this data entry form to view, create, update or delete a nationality/country.

The Nationality and Country are text fields and the national flag is an image which can be clicked to either open the image larger in a popup or replace the image by uploading a new image, or deleting the image.

The top right form buttons respectively allow you to update or delete the record via the ReST API.

The National Team Players grid reads all of the footballers in the associated ReST API dataset who represent this nation and displays them in tabular format. This dataset contains the name, position, squad number and team name and associated images.

Clicking on the player name or the team name will 'drill-down' and open the full data entry form for the record type selected.



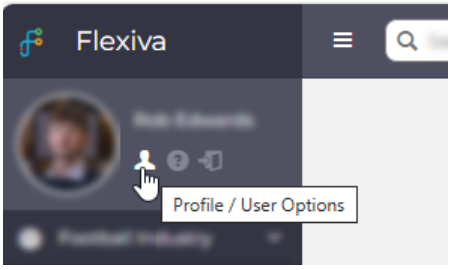
User Options

The end-user can configure the behaviour of the application using this form.

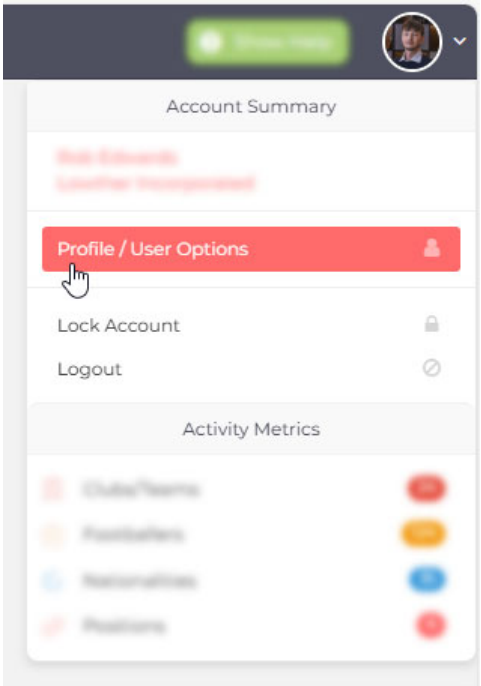
Opening User Options

The user options can be opened from three locations:

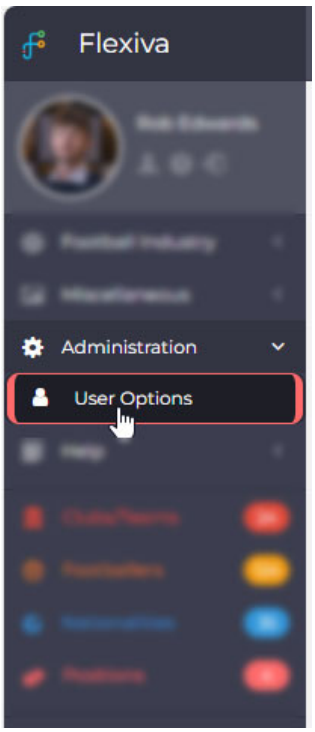
Navigation Bar Account Details



Toolbar Account Summary

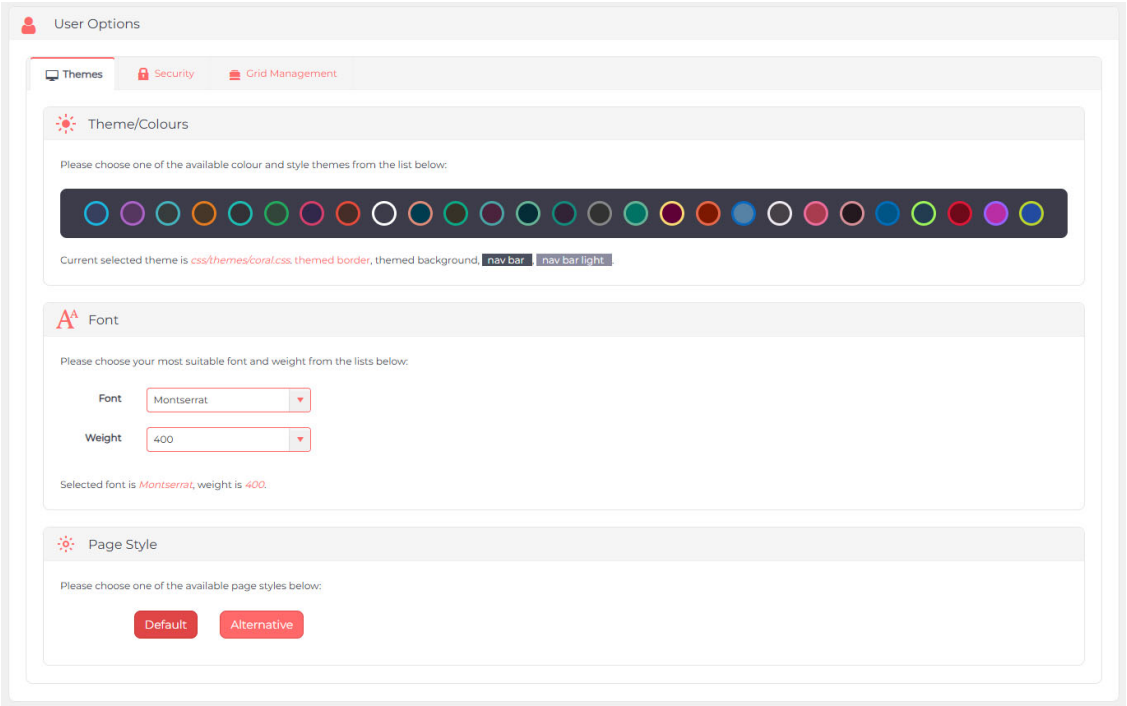


Navigation Bar Administration Group



User Options Form

The form opens up and shows 3 or 4 tabs:



Themes

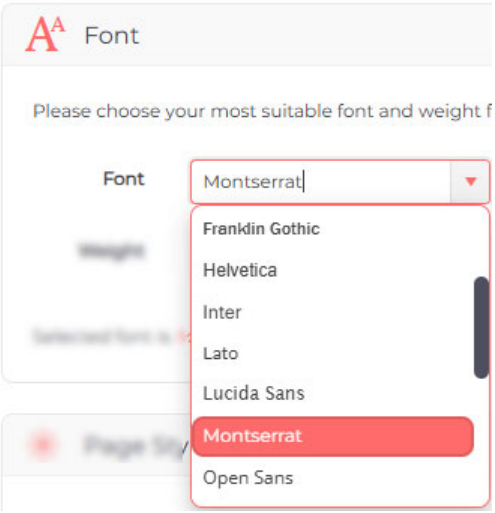
Theme/Colours

You can choose your favourite theme/colours by clicking any of the available themes. Hovering over each theme displays the name as a tooltip:

Clicking each theme immediately refreshes the theme and your selection is persisted automatically.

Font

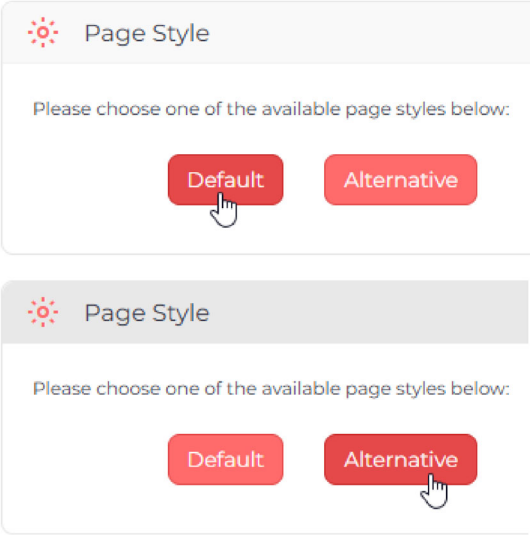
You can choose your favourite font and weight by selecting from both drop down combos:



Selecting each font property immediately refreshes the app, and your selections are persisted automatically.

Page Style

The page style is can be softer (Default) or heavy (Alternative) e.g.



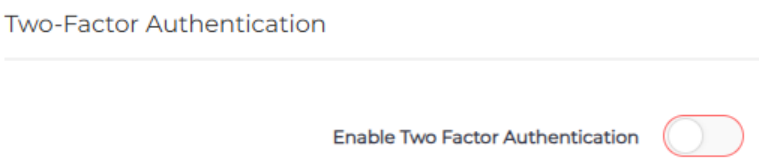
Clicking each style immediately refreshes the app styles and your selection is persisted automatically.

Security

Login Credentials

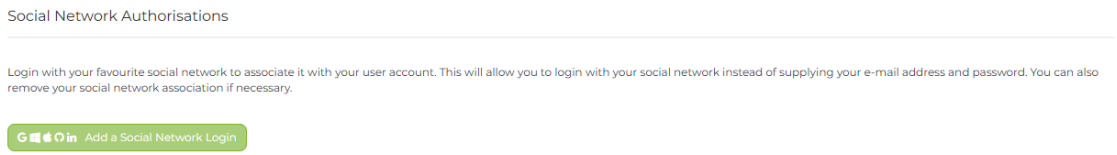
You can enable two-factor authentication to add an additional layer of security to your account. Every time you login, you will need to use an authenticator on your mobile phone to obtain a new generated six digit code and type this to gain access to this system.

Check the **Enable Two Factor Authentication** to open up the workflow to enable the two factor authentication:

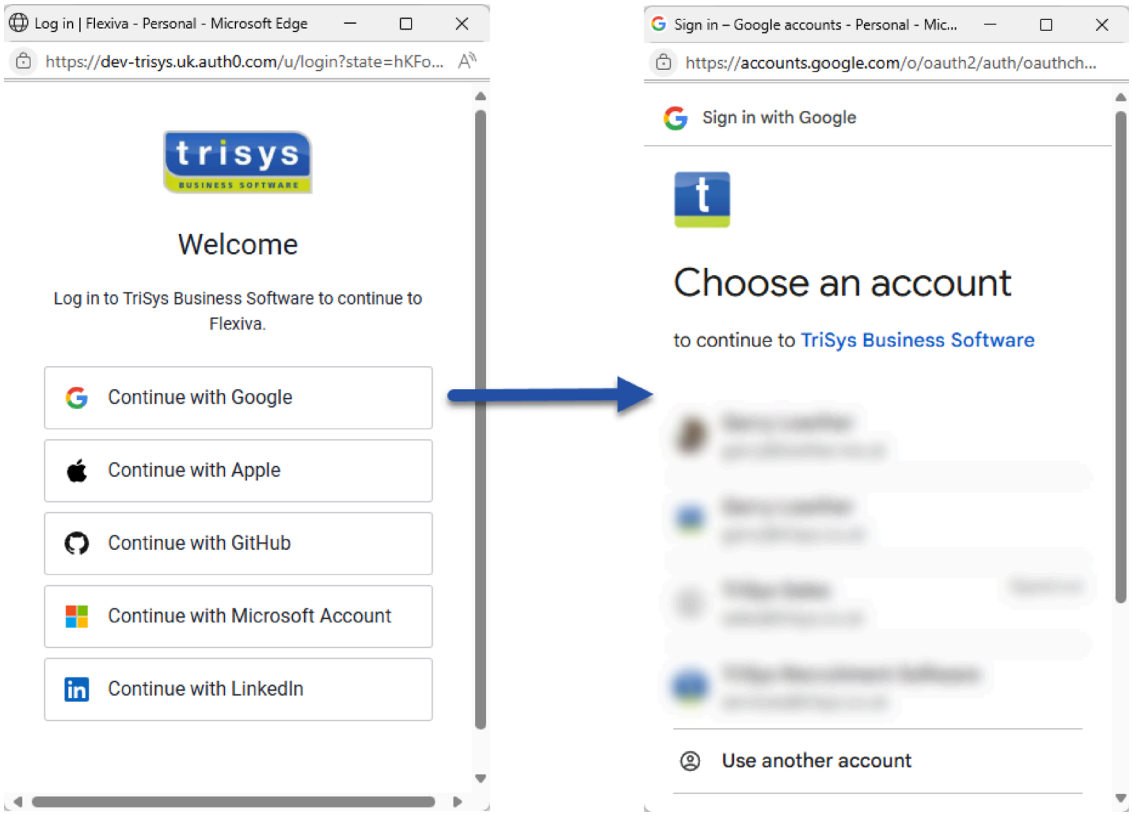


You can use any standard TOTP (Time-based One-Time Password) authenticator app.

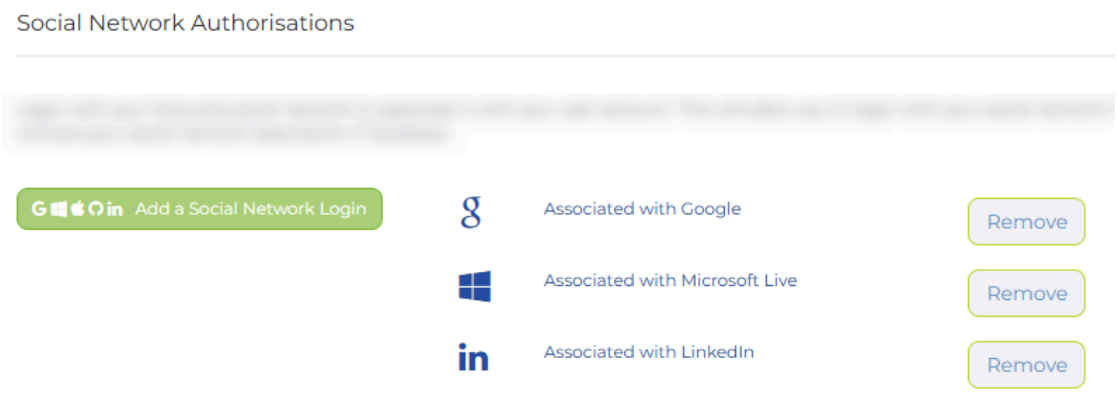
You may also associate your account with your favourite business-oriented social network such as Google, Microsoft, Apple, GitHub or LinkedIn by clicking this button:



Click this button will open these consecutive popup forms which will allow you to choose your network provider and respective login account:



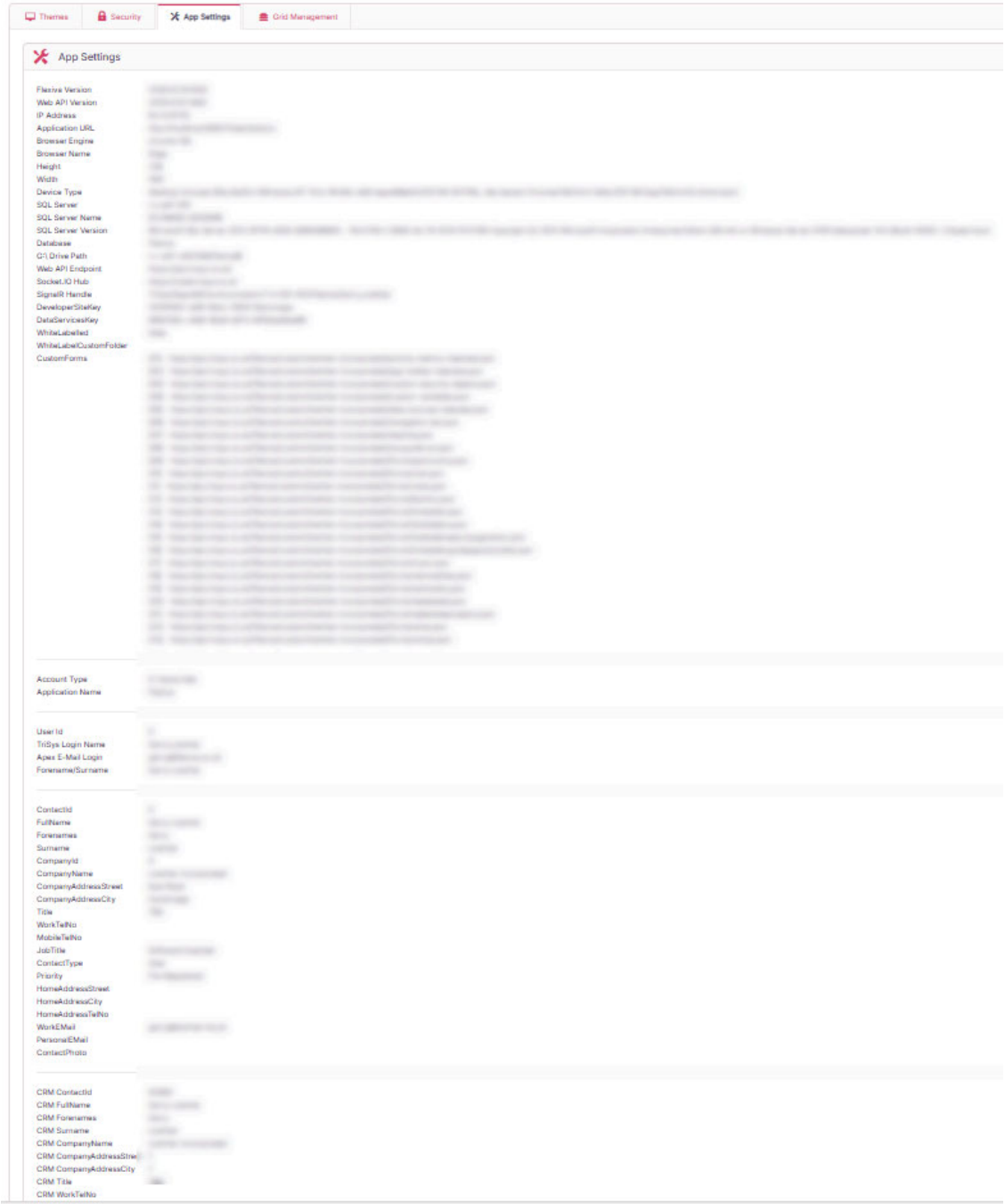
After successful authentication, you will see all of your social network logins associated with your account:



You can now login to the application using any of those social networks you have successfully authenticated against.

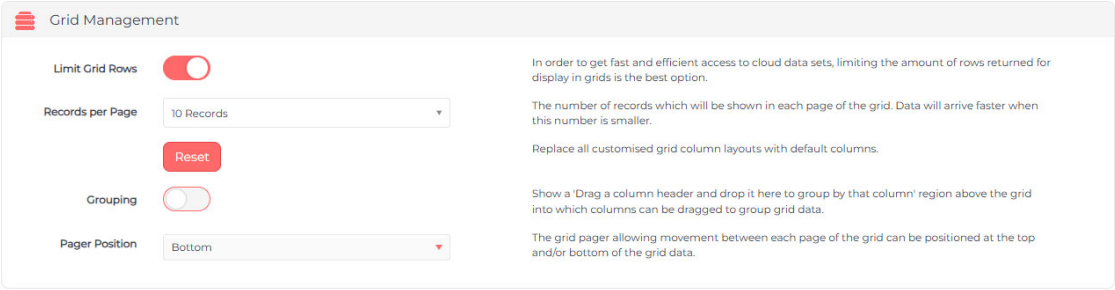
App Settings

This tab is only normally shown to designers or technical support personnel who wish to see technical details about the application configuration:



Grid Management

All of the lookup forms and some data entry forms will have grids to display, filter and sort tabular data sets. The default behaviour of the grids can be specified here:



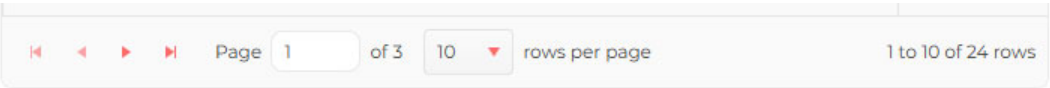
Due to the nature of 'client-server' computing architecture, it is best practice to do all the filtering and sorting of large data sets on the server before shipping small groups of data to the client. **Limit Grid Rows** is this always checked.

Records per Page is the number of records sent from the server to the client each time the user requests data. Keeping this small ensures fast performance, whilst keeping this large will take longer, but each 'page' of data will be larger.

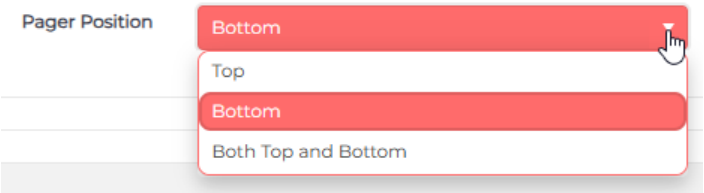
When you resize grid columns, or show/hide columns, or resize columns, this is persisted. Sometimes the grid can end up looking untidy. The **Reset** button resets all of your grids back to how your designer configured them.

Grid columns can be grouped by dragging them into the top of the grid. This is a complex operation and is at the discretion of the designer as to whether this is enabled or not. Use the **Grouping** option to turn on or off.

The grid pager controls look like this:



These can be positioned either at the top, or bottom or both using the **Pager Position** drop down combo:



Beginner Designer Guide

Introduction

Take a one-step-at-a-time approach to design a simple data-driven lookup form.

This section details a step-by-step approach to building a small starter application with a single data-driven lookup form.

The first section will get you up and running with a skeletal app, followed by more comprehensive steps to connect data, design forms and publish your app.

This is a good place to start to understand how to use App Studio and the configurators.

You should allow 30 minutes to work through this three-step beginners guide.

Prerequisites

You should have signed up for the free trial, be logged in as a designer, and have read the first few chapters in the knowledge base.

You should also have tested how the sample app works in order to understand the intrinsic CRUD mechanisms that can be built with the App Studio configurators.

Step 1

The first step is to build a skeletal, or minimal, app comprising a lookup form and a navigation bar group. Click here for step 1.

Step 2

The second step is to connect a ReST API data source and send a request to retrieve a data set. Click here for step 2.

Step 3

The third step is to display a data set in your custom lookup form. Click here for step 3.

Step 1: Skeletal App

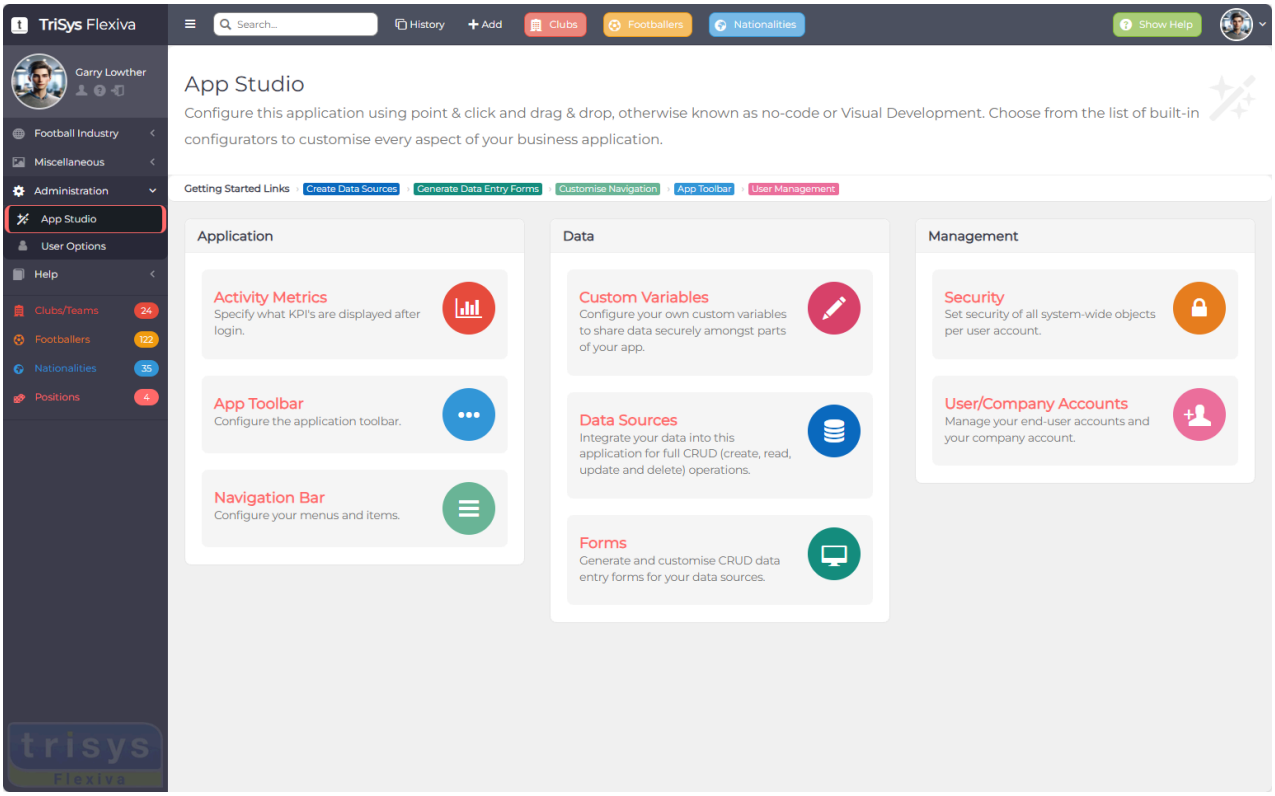
Build the application framework using the app studio configurators.

The first step is to create a custom lookup form, and make it available from the navigation bar so that you understand the fundamental design concepts.

It is expected that this step will take no more than 10 minutes of your time.

Open App Studio

Open the administration group in the navigation bar on the left and click on App Studio.

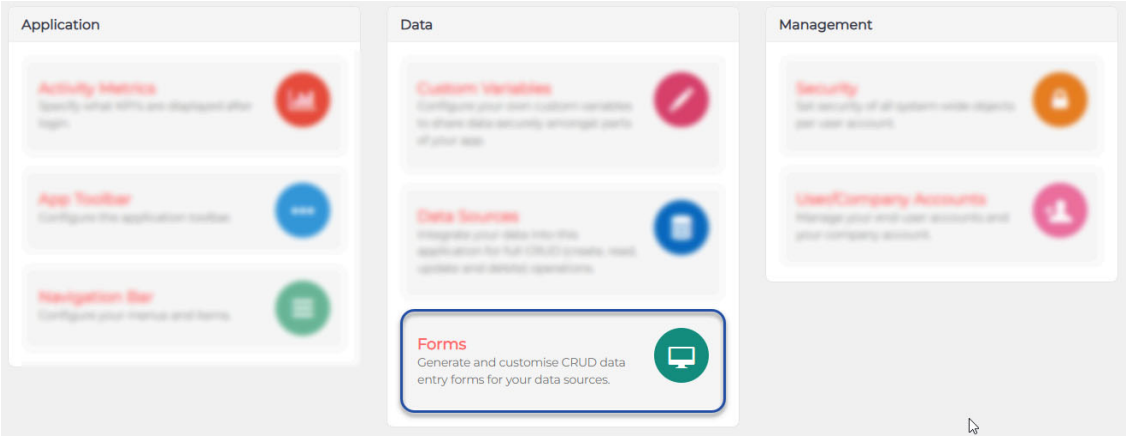


Create a Lookup Form

Create a sparse lookup form, change the panel properties and test how it looks.

Open Forms Configurator

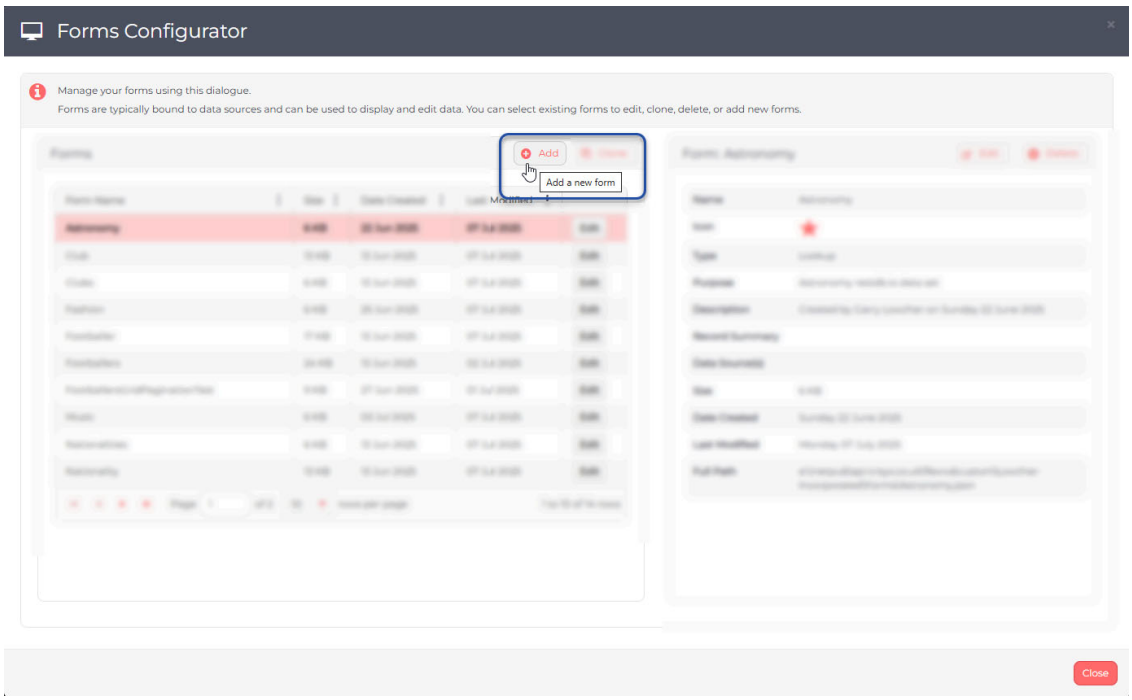
The Forms configurator is located at the bottom of the Data group in App Studio.



When it is clicked, it opens a modal popup dialogue showing a list of all forms in your application.

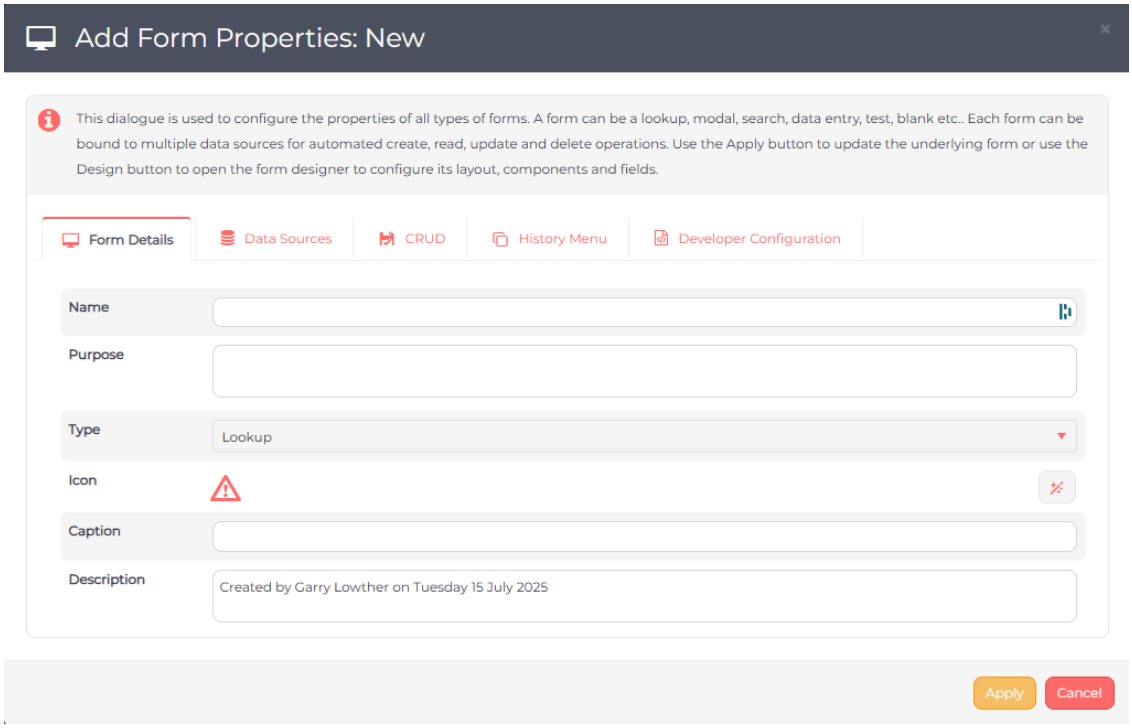
Add

Click the Add button to create your first lookup form:



The Add Form Properties modal popup dialogue will open.

Add Form Popup



Complete the following fields.

Name

Type a name for your new lookup form.

Purpose

Type a reason why you are creating this form.

Icon

Use the button to the far left to open the Icon Picker to choose a suitable icon for your new form.

Caption

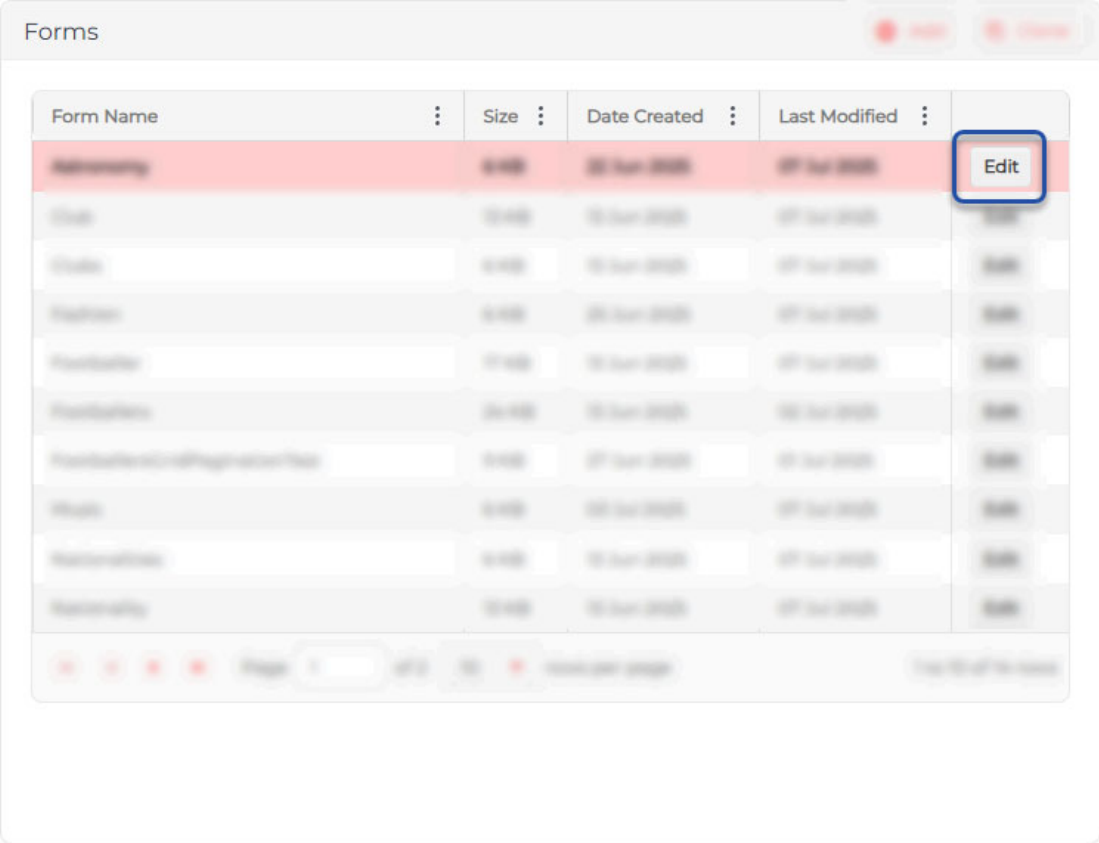
Type the caption you wish to see in your new form.

Apply

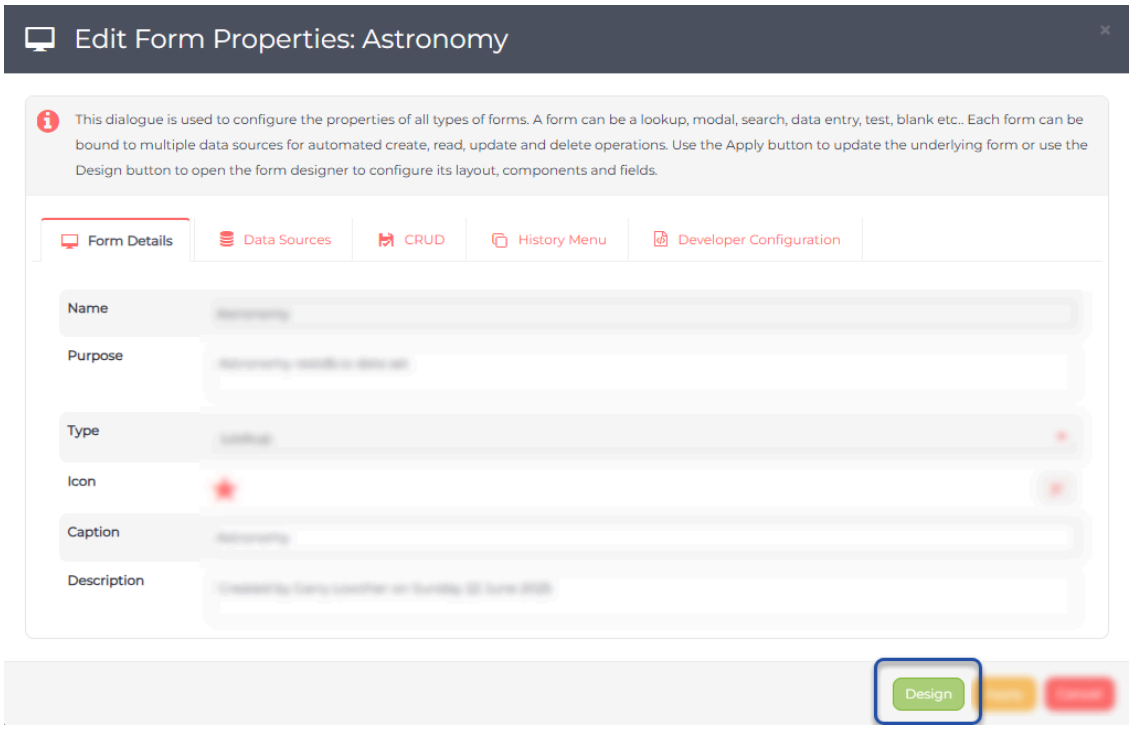
Click the Apply button to create your first lookup form.

Design Form

Your new form will now be visible in your list of available forms.



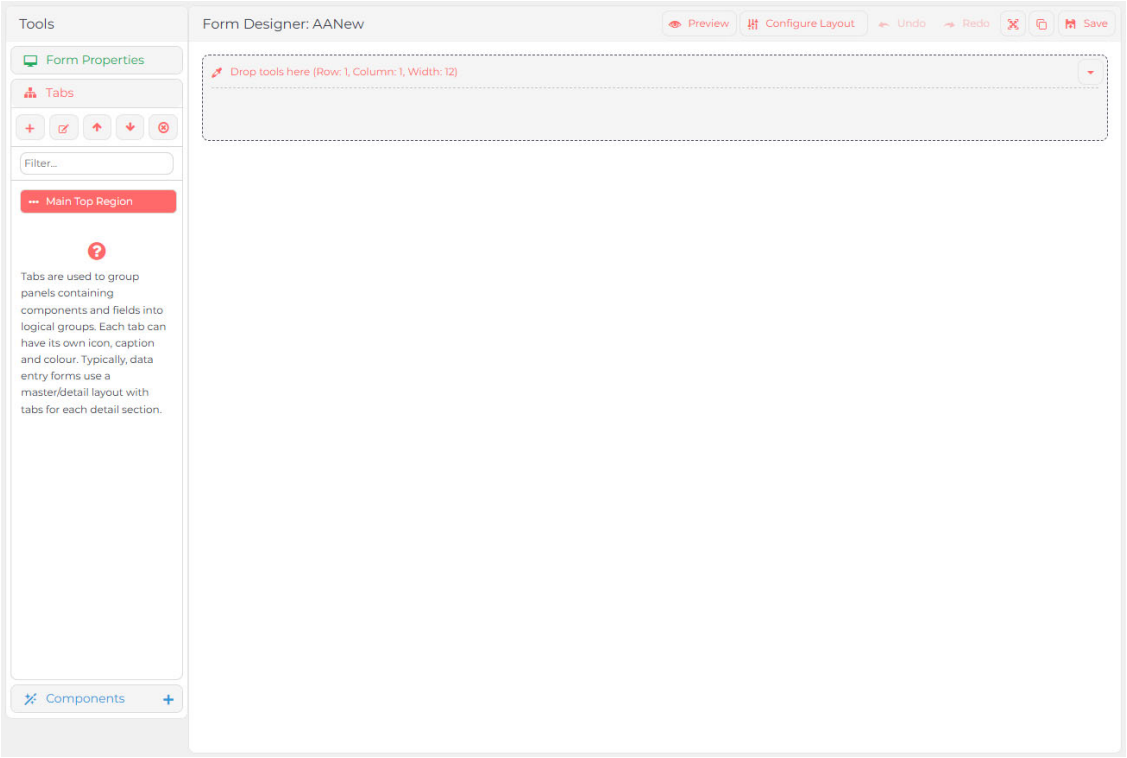
Click the Edit button in the grid row to open up the Edit Form Properties modal popup dialogue again.



Click on the Design button to open the form designer.

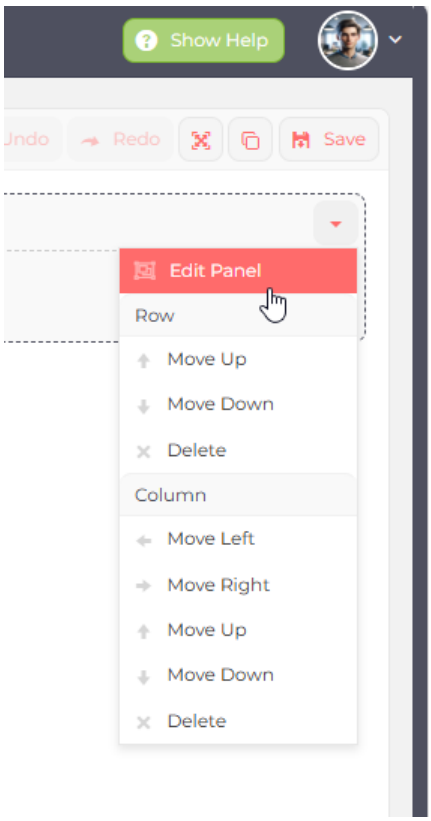
Form Designer

The form designer will open with a single empty panel.

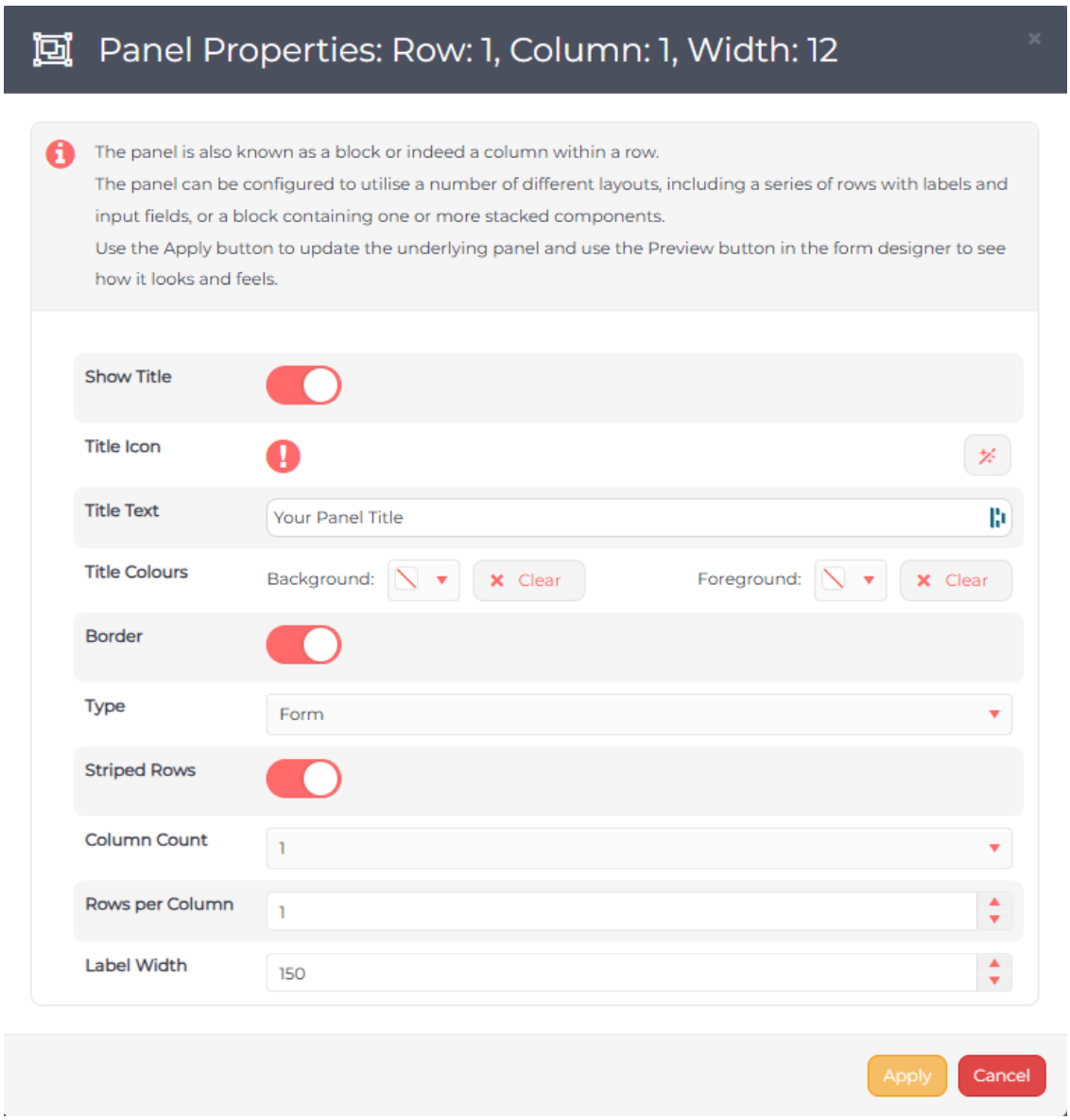


Edit Panel

Edit the panel using the right top drop down menu.



The Panel Properties modal popup dialogue will open.



Assign an Icon, and type your Title Text. This will allow you to identify your first form when it is opened.

Click **Apply**.

Save

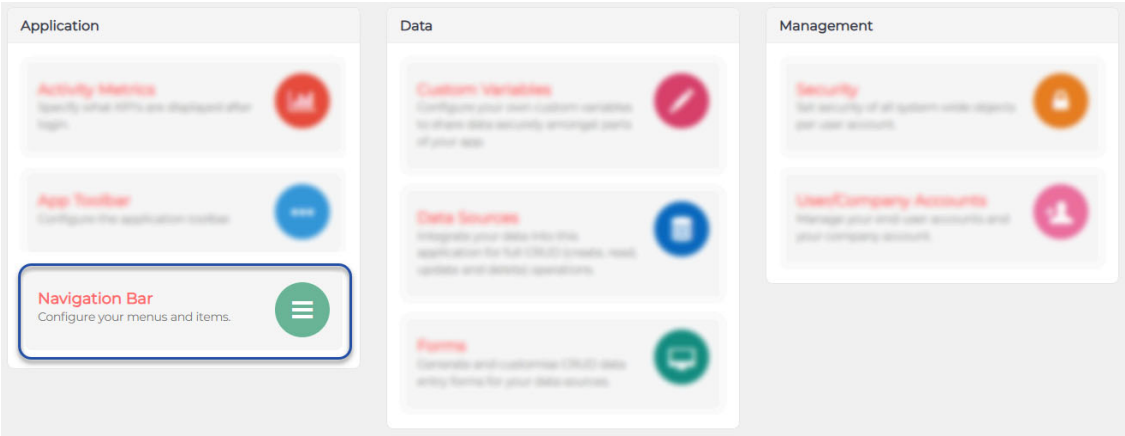
Click the top right **Save** button to persist your form design changes.

Create a Navigation Bar Group

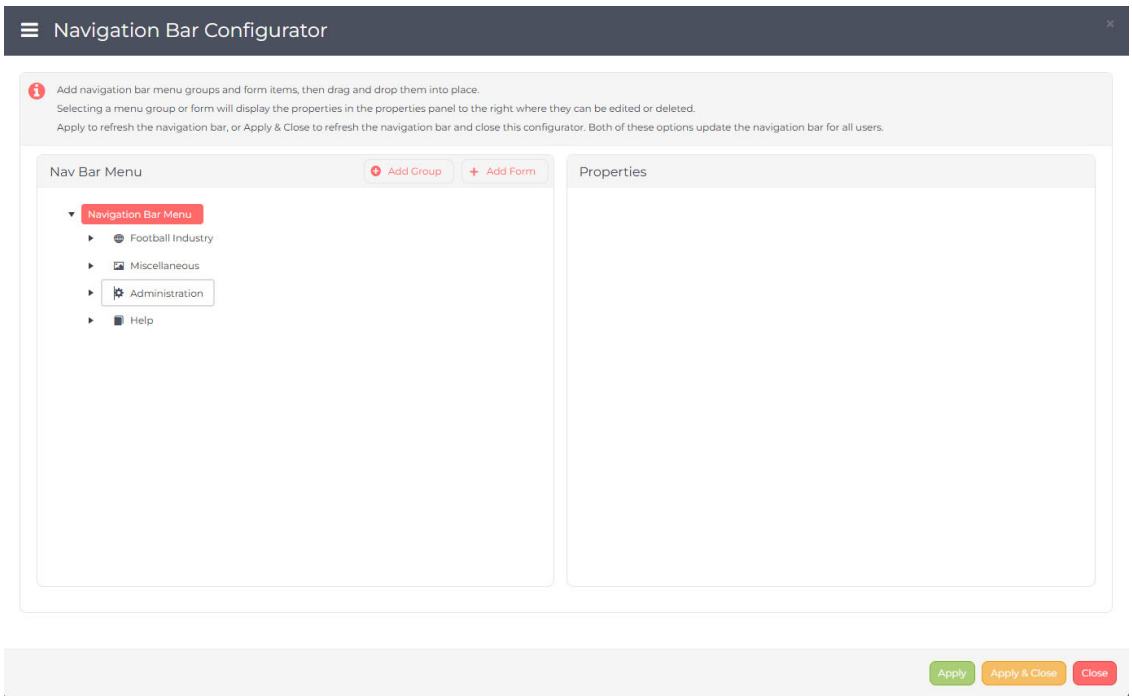
Create your own custom group in the navigation bar.

Open Navigation Bar Configurator

Open App Studio and click the Navigation Bar item:

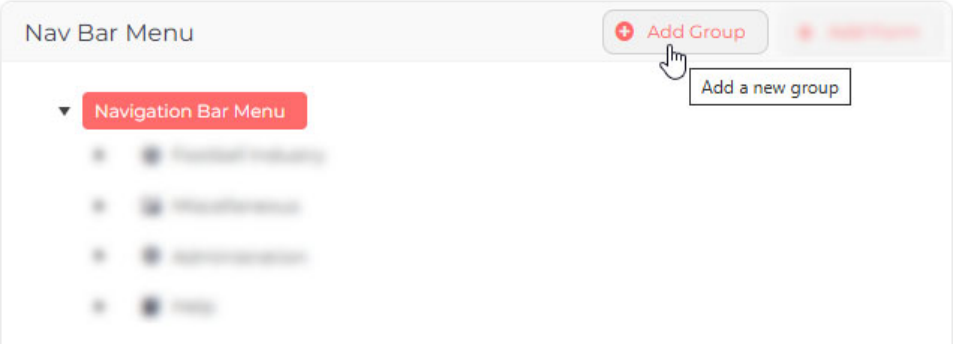


The Navigation Bar configurator will open.



Add Group

Click the Add Group button.



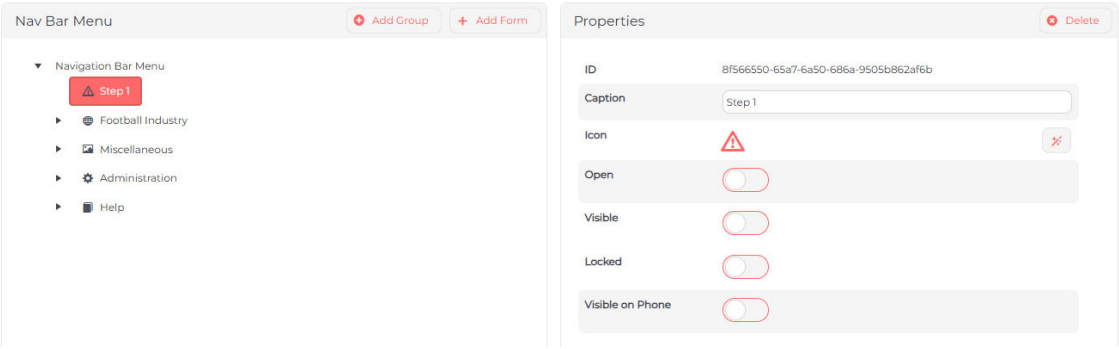
Add New Special Menu Group

This modal popup dialogue will open:

Enter the name of your new navigation bar group, then click the Save button.

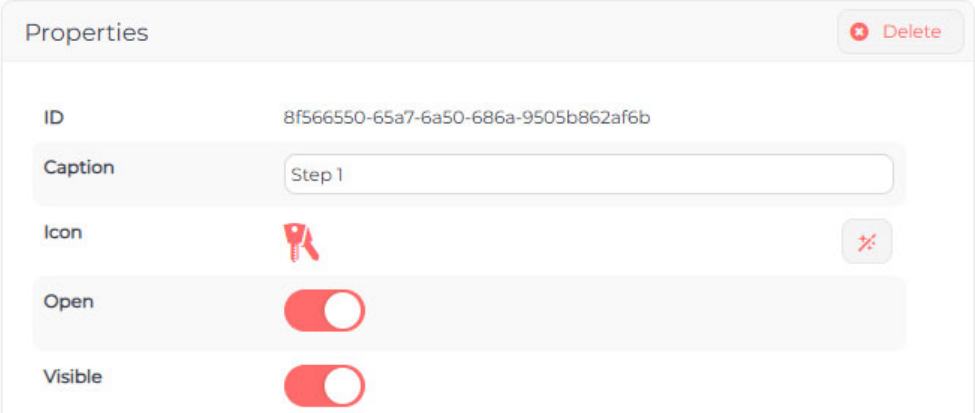
New Navigation Bar Group

The new group is added and selected in the navigation bar tree.



Edit Group Properties

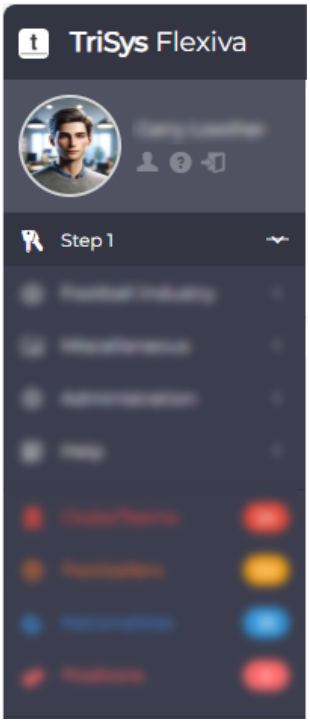
Change the Icon and make sure that this group is Open and Visible.



Apply & Close

Press the Apply & Close button.

The new group should now be visible in the navigation bar



Create a Navigation Bar Group Item

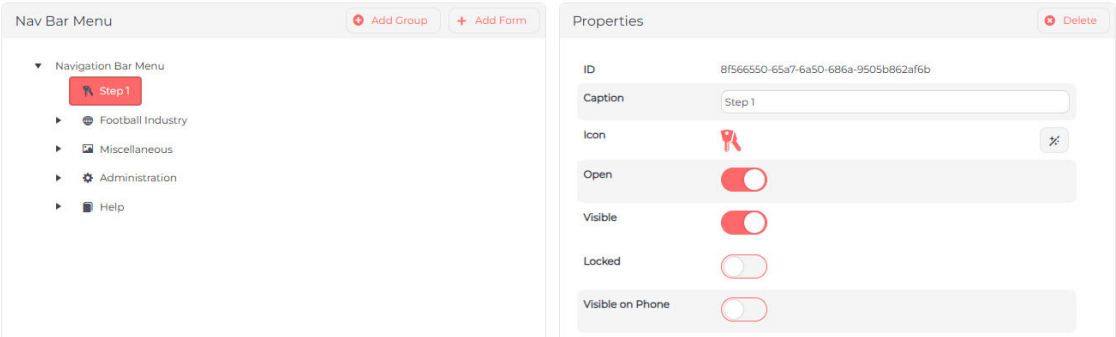
Add a nav bar item linked to your new lookup form.

Open Navigation Bar Configurator

Open like you did here.

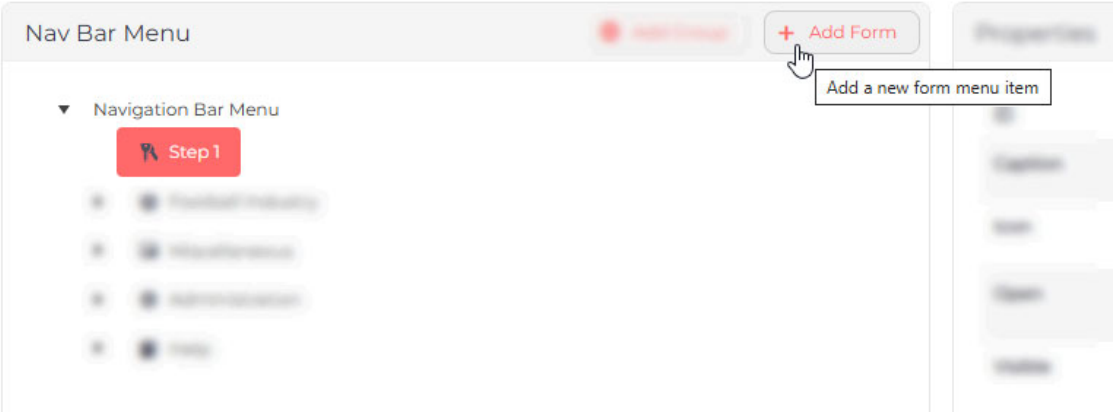
Select the Added Group

Select the new group you added previously:

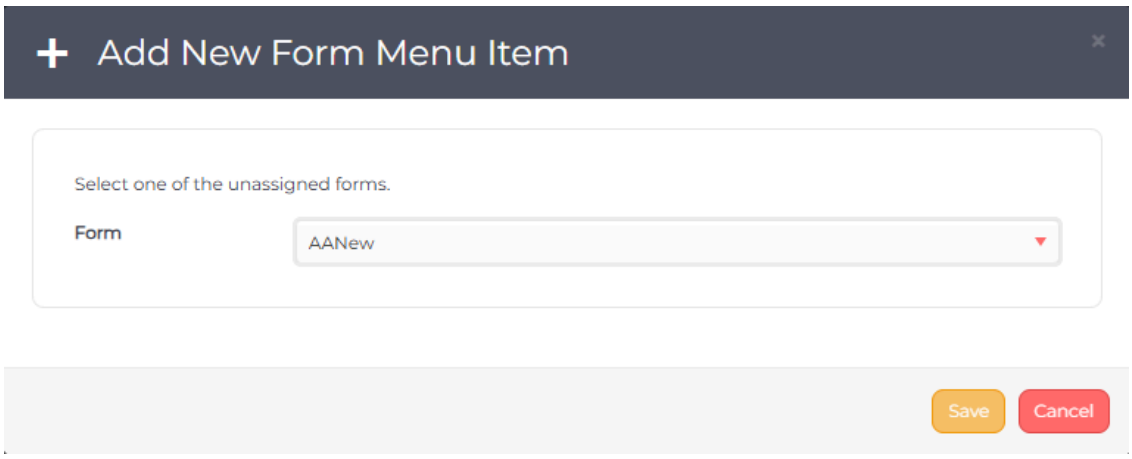


Add Form

Click the Add Form button in the Nav Bar Menu:



The Add New Form Menu Item modal popup dialogue will open:

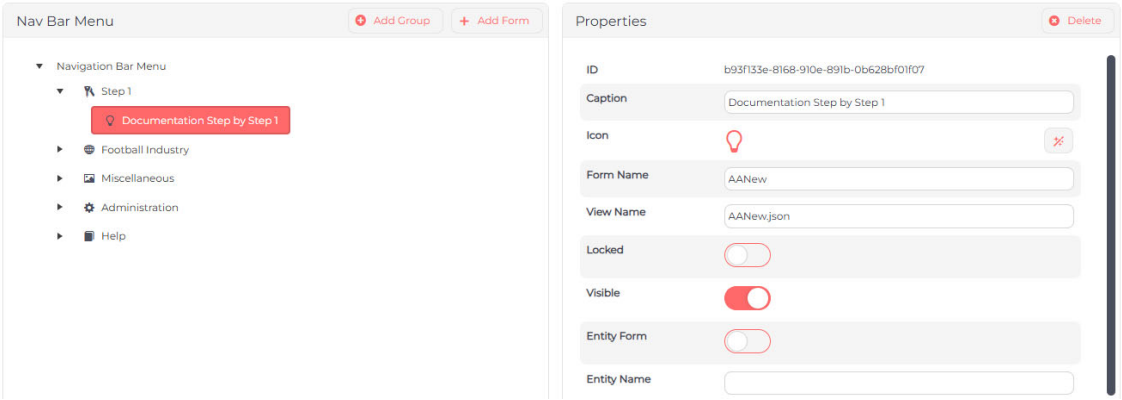


Form

The form field is a drop down combo containing a list of all forms which have not been added to the navigation bar. This should show the form you added above.

Save

Press the Save button to create the new navigation bar group item:



Edit Properties

The properties will have been inherited from the form you created. You can change any properties here.

Apply & Close

After changing any properties, press the Apply & Close button. The new menu group item should appear in the navigation bar.

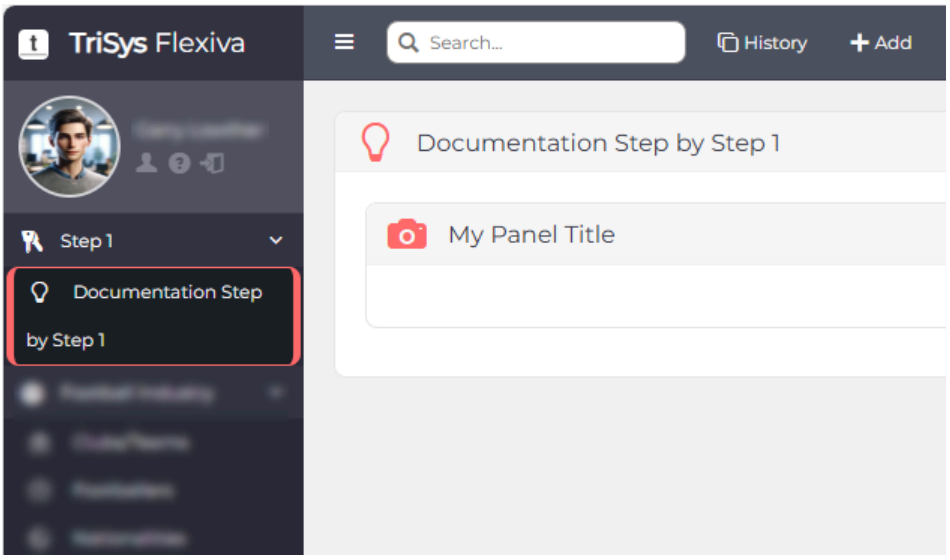
Publish and Test

Publish your changes and test your application.

Because you are designing using App Studio, your changes are already published and ready for you to test.

Open Form

Click on the new navigation bar group menu item.



Your new form should be displayed.

The next step is to configure a ReST API data source connection.

Step 2: Connect to Data

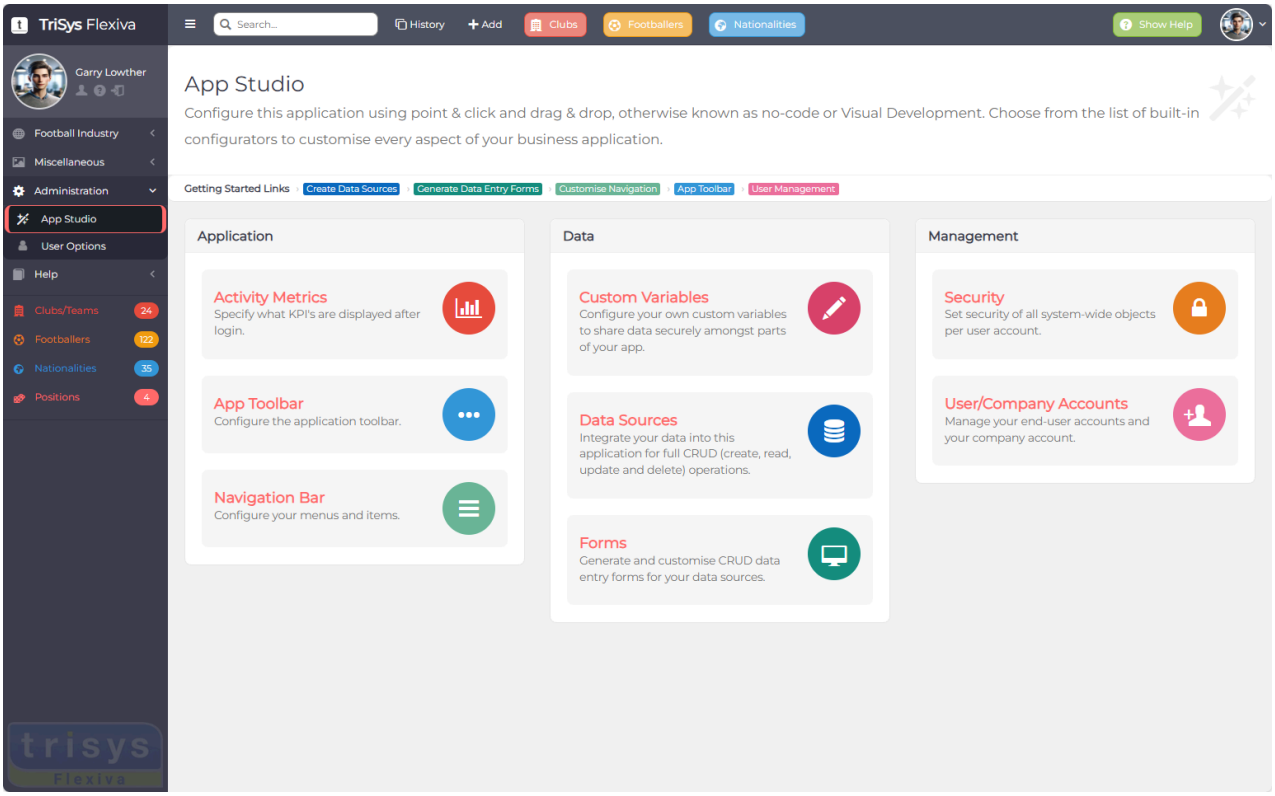
Connect your application to a data source.

The second step is to connect to a ReST API and test it to show a data set so that you understand the fundamental data-oriented concepts.

It is expected that this step will take no more than 10 minutes of your time.

Open App Studio

Open the administration group in the navigation bar on the left and click on App Studio.

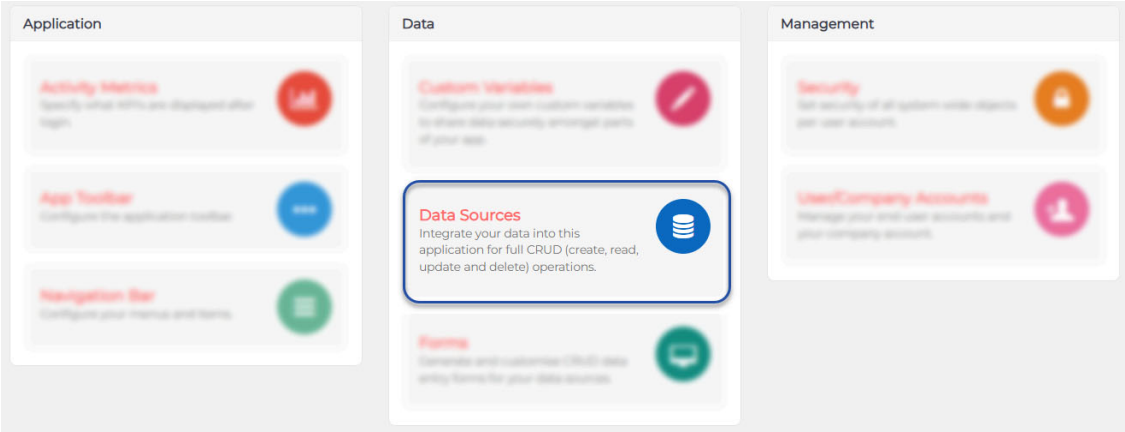


Create a Data Source

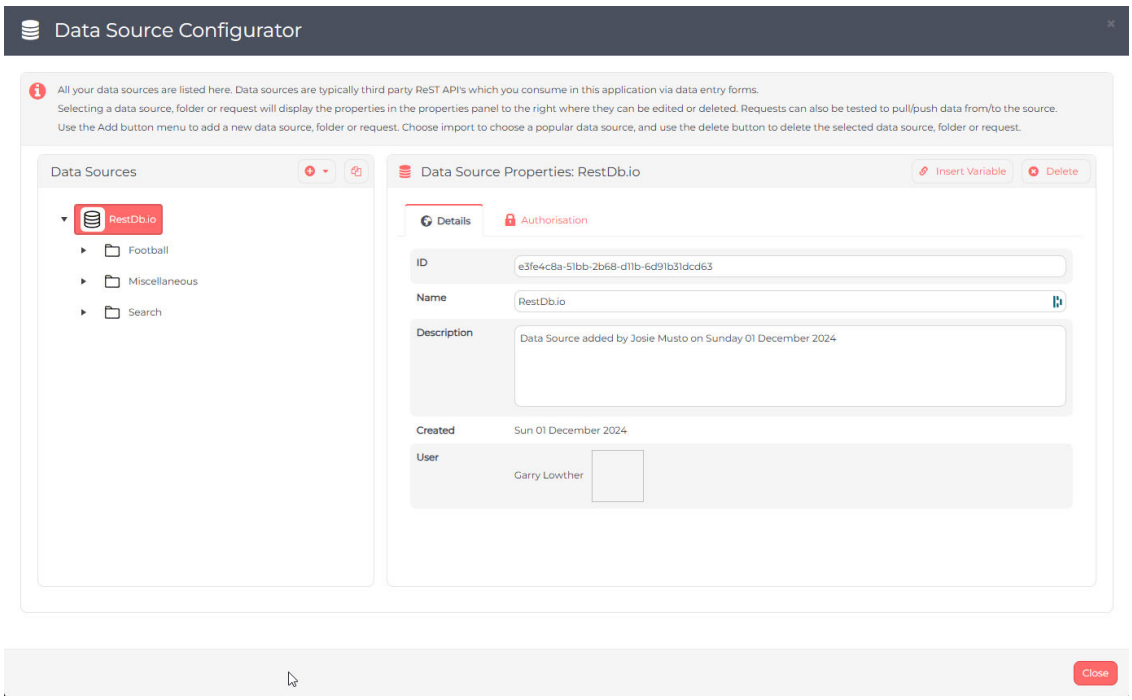
Create a data source request which connects to a ReST API to return a data set.

Open Data Sources Configurator

The Data Sources configurator is located in the middle of the Data group in App Studio:



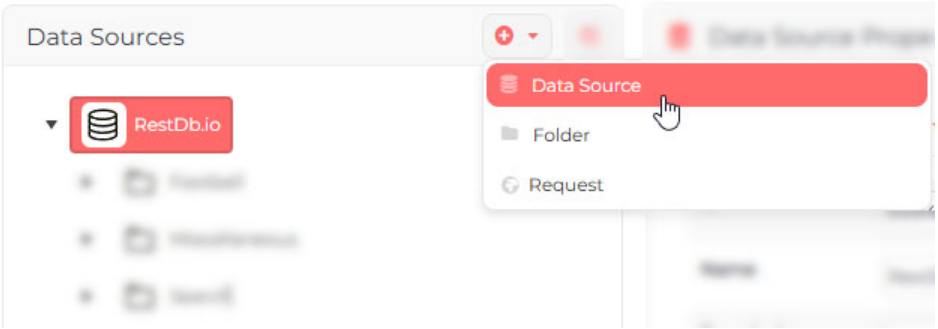
When it is clicked, it opens a modal popup dialogue showing a list of all data sources in your application:



The only data source should be the demo data set provided with the sample app.

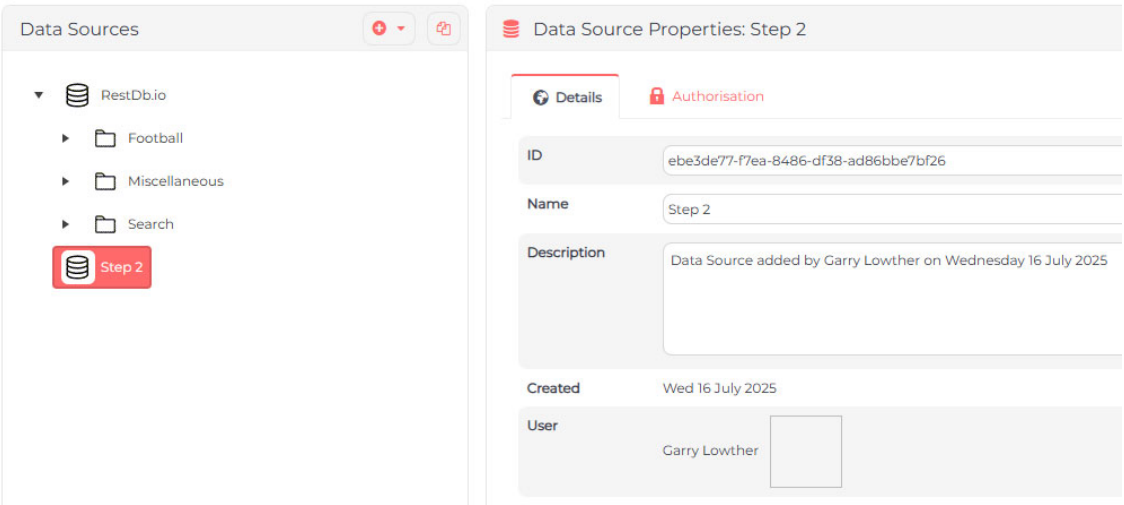
Add New Data Source

Click the plus drop down menu and choose Data Source:



You will be prompted to supply the name of your new data source:

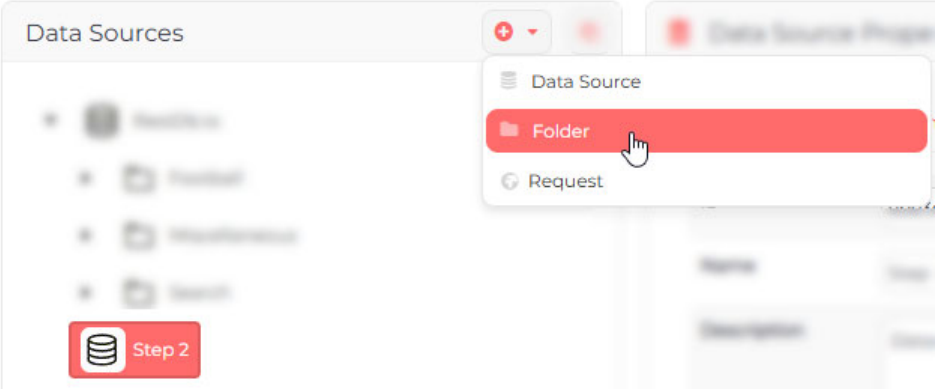
Type the name and use the Save button add the new data source to the tree view list:



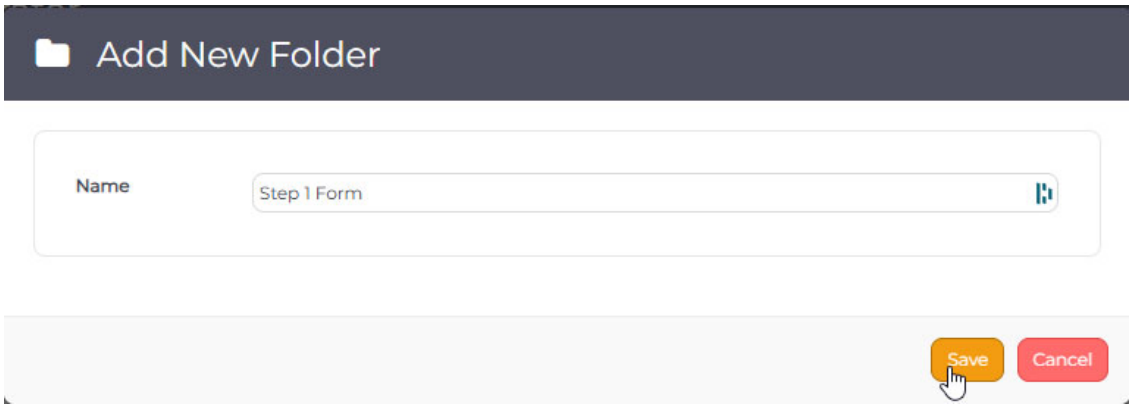
The new data source will be selected on the left and the properties are shown on the right.

Add Folder

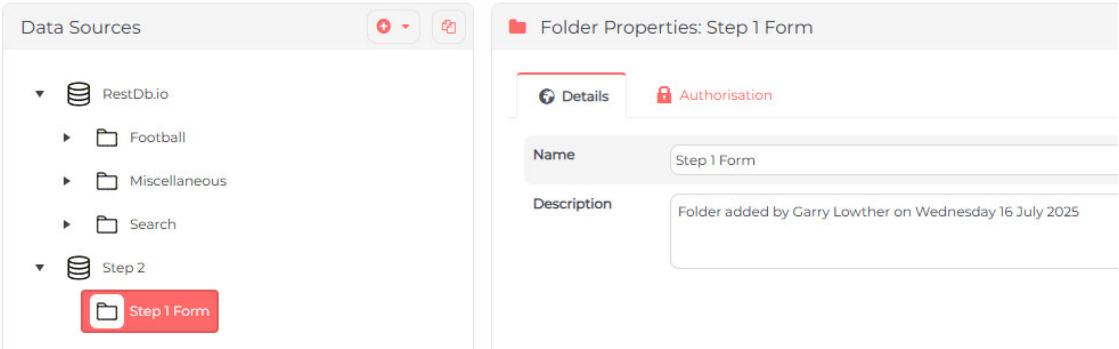
Add a new folder beneath the data source so that all your data source requests are grouped for easier maintenance:



You will be prompted for the name of your new folder. Perhaps use the name of your first lookup form you created in step 1. Press the **Save** button:

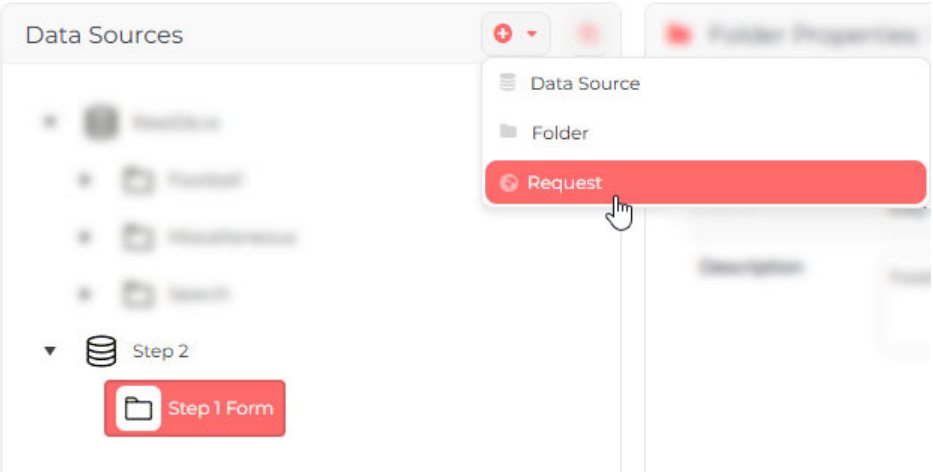


Your new folder will be created beneath your new data source in the left tree view with the folder properties on the right:

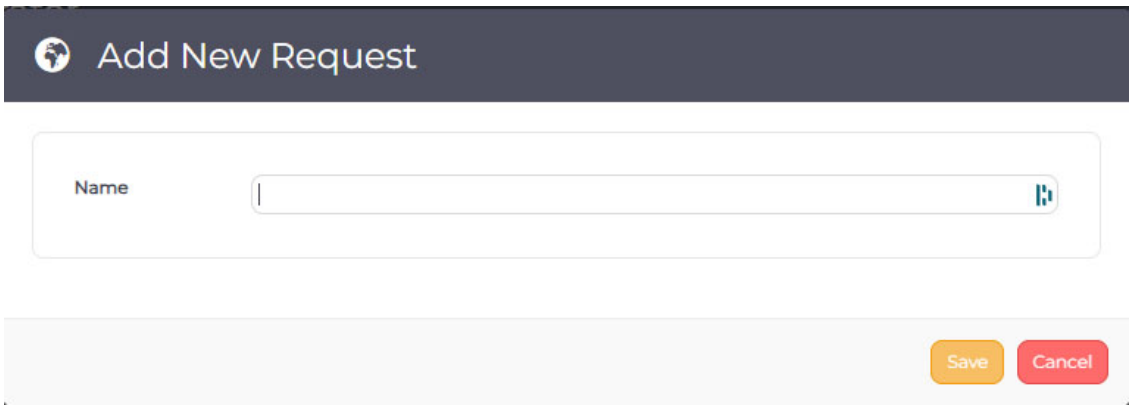


Add a new Request

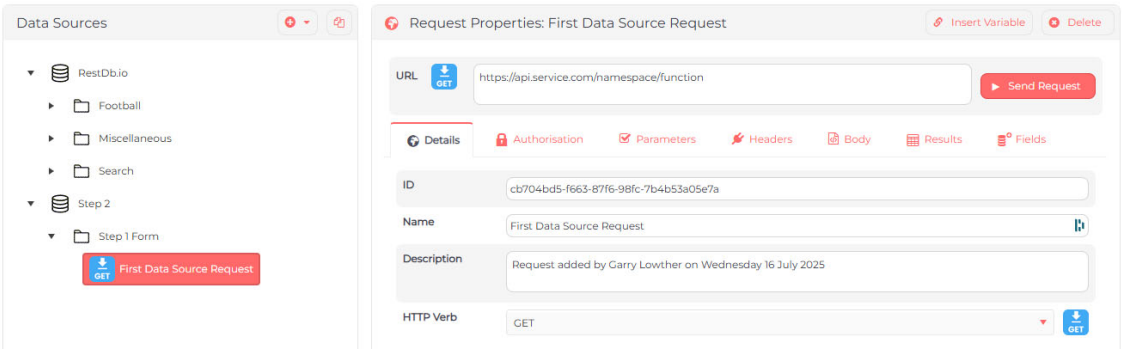
Create a new data source request using the same add button menu:



You will be prompted to enter the name of your new data source request:



Give it a suitable name and press Save:



Your new request will now be selected in the left tree view and its properties are shown on the right.

Specify the URL

We now need to paste in the ReST API endpoint into the URL text box.

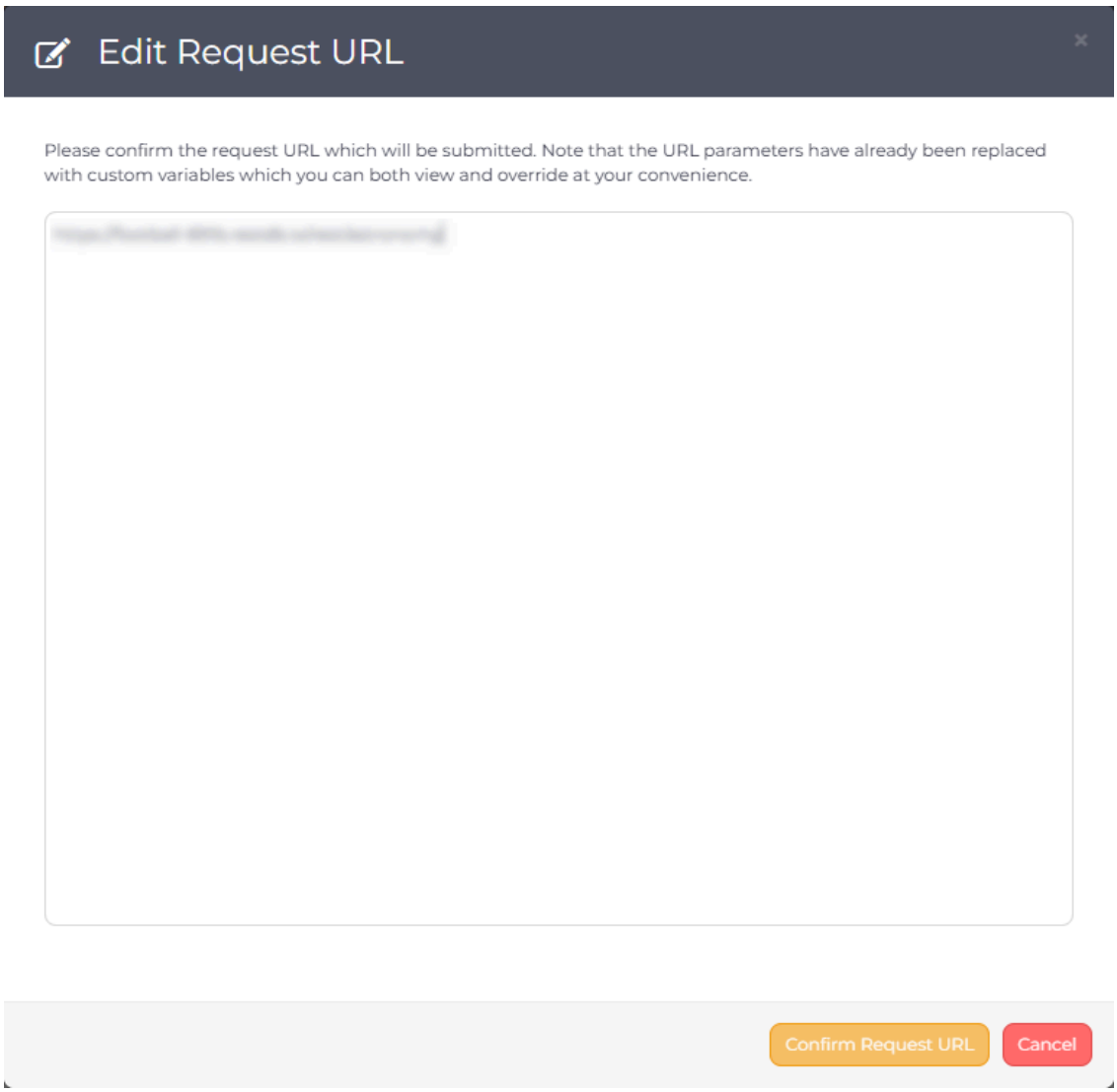
The default for testing is `https://app.flexiva.co.uk/usercontrols/app-studio/data-source/astronomy.json`

This returns an astronomical data set which is enough to demonstrate the principles of data source request consumption. It also requires no authentication credentials which are [documented here](#).


```
[
  {
    "_id": "6857ae6878badf650012ae96",
    "ID": 2,
    "Object_Name": "Andromeda Galaxy",
    "Type": "Galaxy",
    "Constellation": "Andromeda",
    "Distance_LY": 2540000,
    "Magnitude": 3.44,
    "Coordinates_RA": "00h 42m 44s",
    "Coordinates_Dec": "+41° 16' 09\"",
    "Discovery_Year": "964",
    "Image_URL":
"https://cdn.esahubble.org/archives/images/thumb300y/heic1112e.jpg",
    "Image": []
  },
  ......
]
```

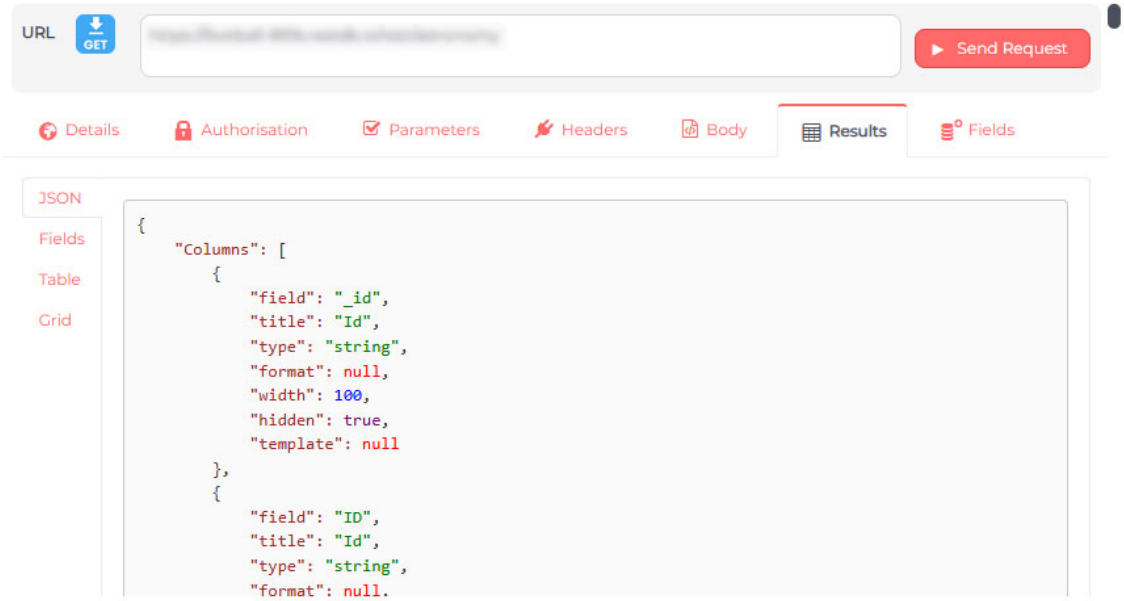

Send Request

We can now test the ReST API endpoint by sending the request to pull the data set. When the Send Request button is pressed, the following prompt is displayed.



This allows designers to override request URL's with injected parameters. In this instance, confirm this by pressing the Confirm Request URL button.

JSON



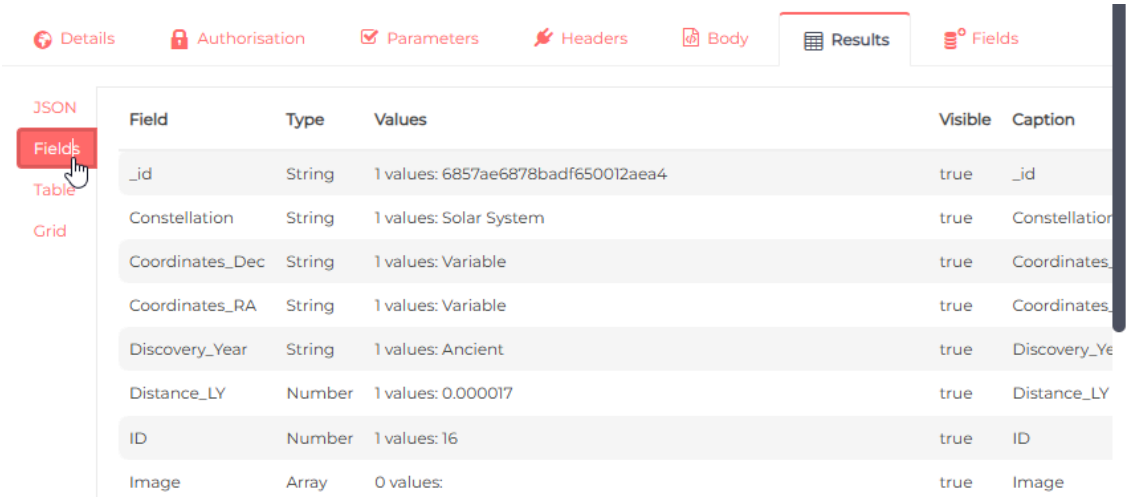
The Results tab should now be selected and the left docked tabs should be visible. The JSON should be displayed showing the raw data set returned from the ReST API.

Analysing Results

The first section of this JSON are the columns/fields which were returned, and scrolling down to the DataTable reveals a list of data items.

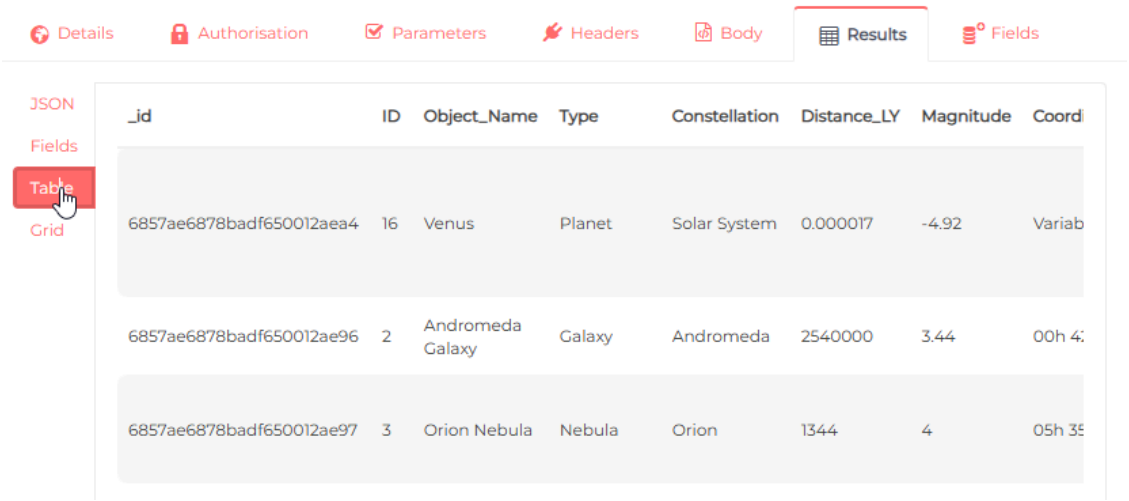
Fields

Clicking on this left docked tab shows all of the fields in the request which were returned.



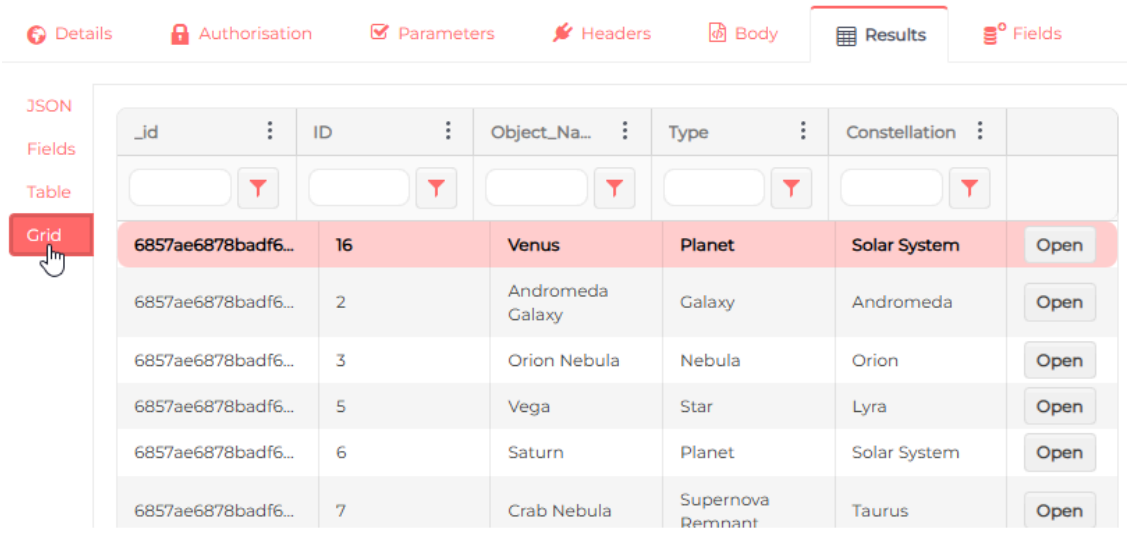
Table

Clicking on this left docked tab shows all of the data in the request represented in table format.



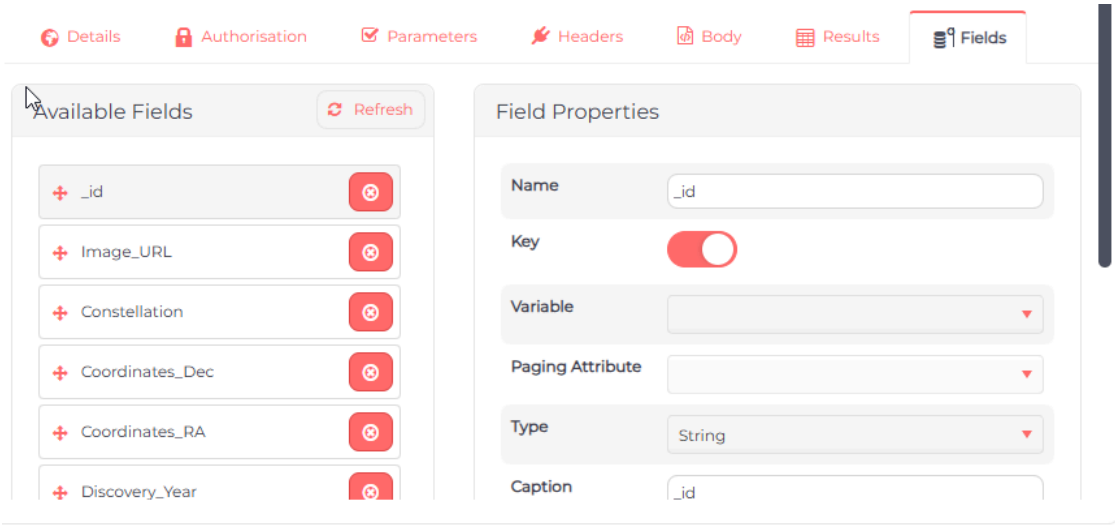
Grid

Clicking on this left docked tab shows all of the fields in the request represented in a data grid. This is a close representation of how your new lookup form grid will show this data when we design it later.



Fields

The field tab is where the extracted list of fields from the request results can be manipulated for presentation in components and forms.

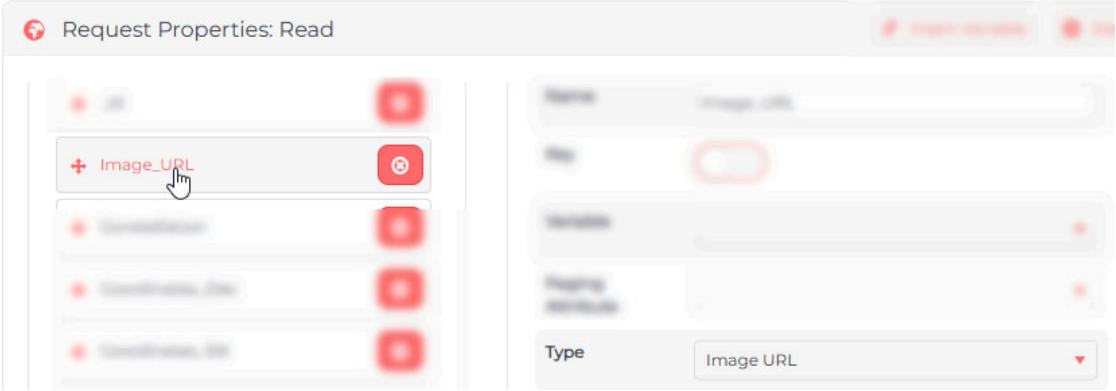


Available Fields

This is a list of all fields returned from the request. They are ordered as they were returned from the ReST API however they can be re-ordered by using the left drag icon of each item. The right delete button removes the field from the list. Selecting any field shows its properties to the right.

Field Properties

These are the properties of the selected field. The Image_URL field should be set to be of Type Image URL as this will allow us to view the image in the grid later. Try dragging this field upwards to reposition it when you design the grid.



Close

Close the configurator as we have now completed adding a new data source request.

The next step will be to consume this data in a lookup form.

Step 3: Lookup Form Data

Design the lookup form to consume ReST API data.

The third step is to edit your custom lookup form to add a data grid to show a data set so that you understand the fundamental form and data relationship concepts.

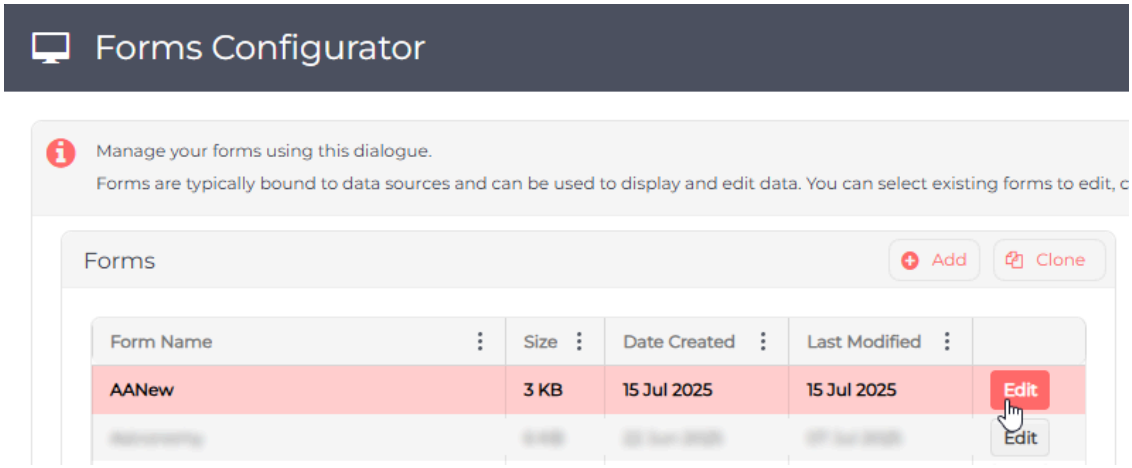
It is expected that this step will take no more than 10 minutes of your time.

Open your Lookup Form

Open App Studio, then open the Forms configurator as you did in step 1.

Edit

Edit your form using the Edit button in the grid row:

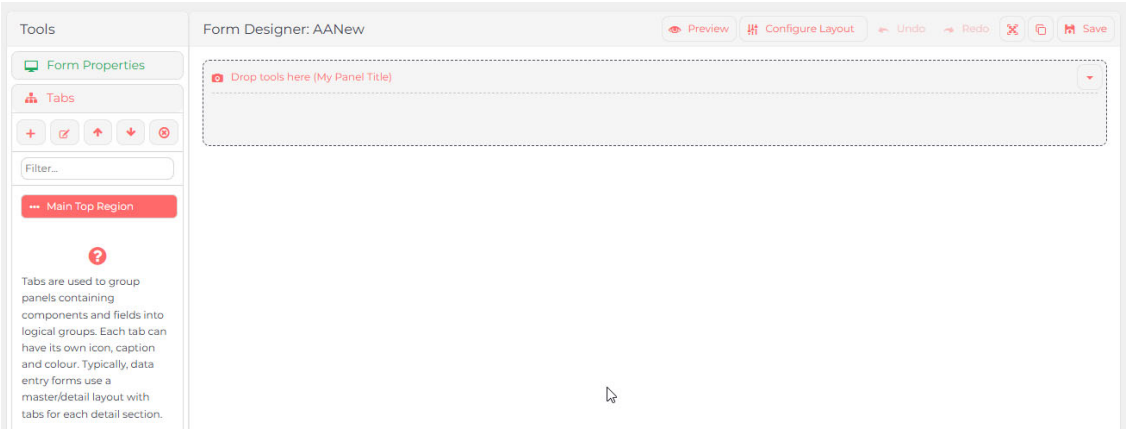


Design

Click the design button to open form designer.

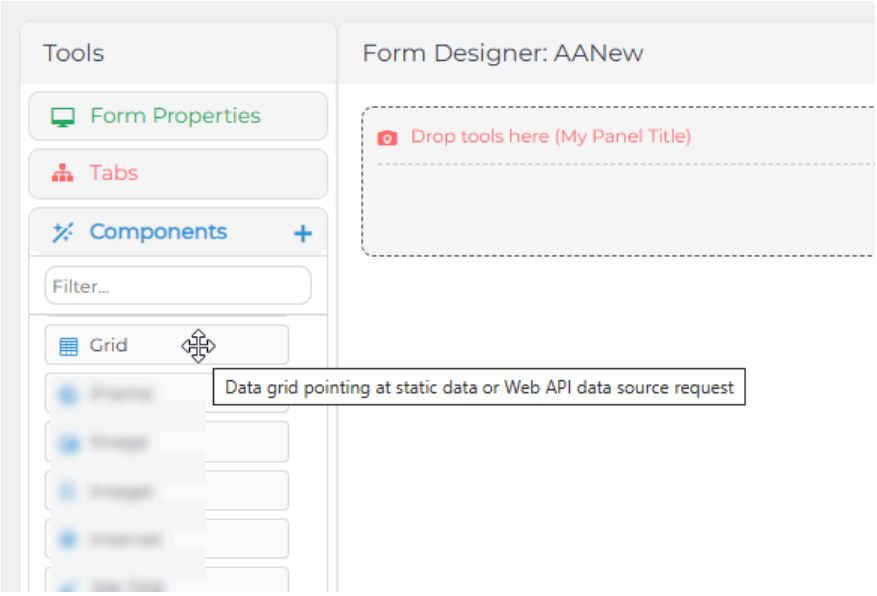
Consume Data in your Lookup Form

Your lookup form, only has one panel. We will now add a grid component and connect it to your data source:



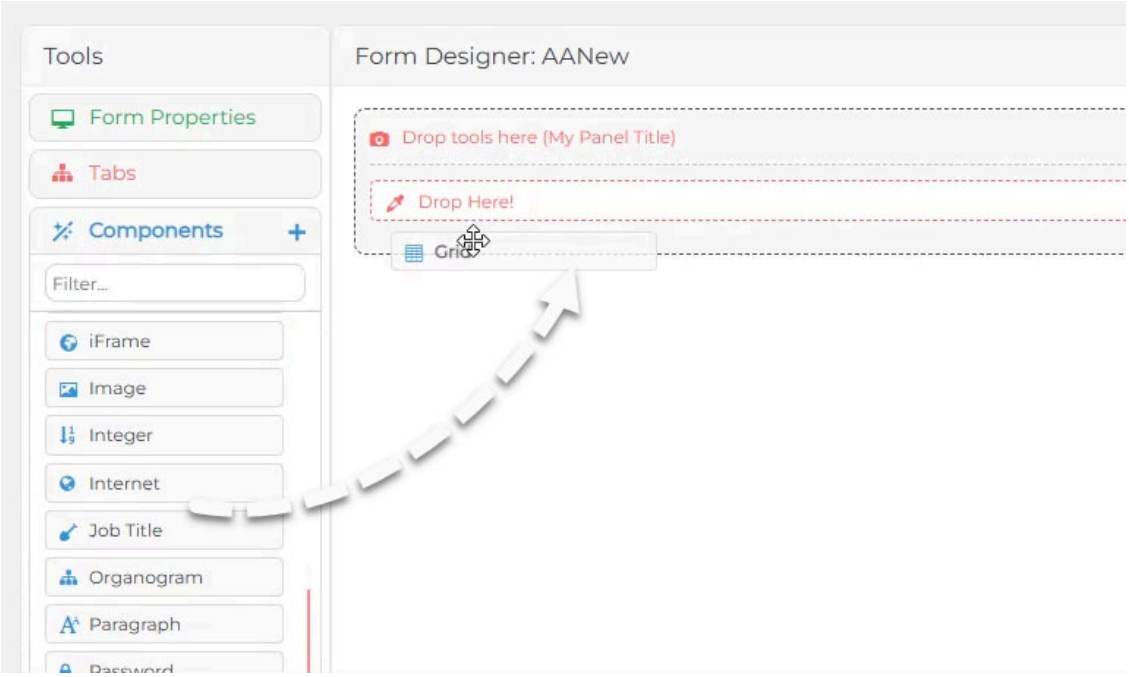
Components

Click on the components section of the tools:

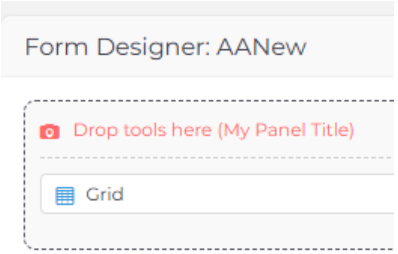


Grid

You should see a Grid component. Drag this into your panel:

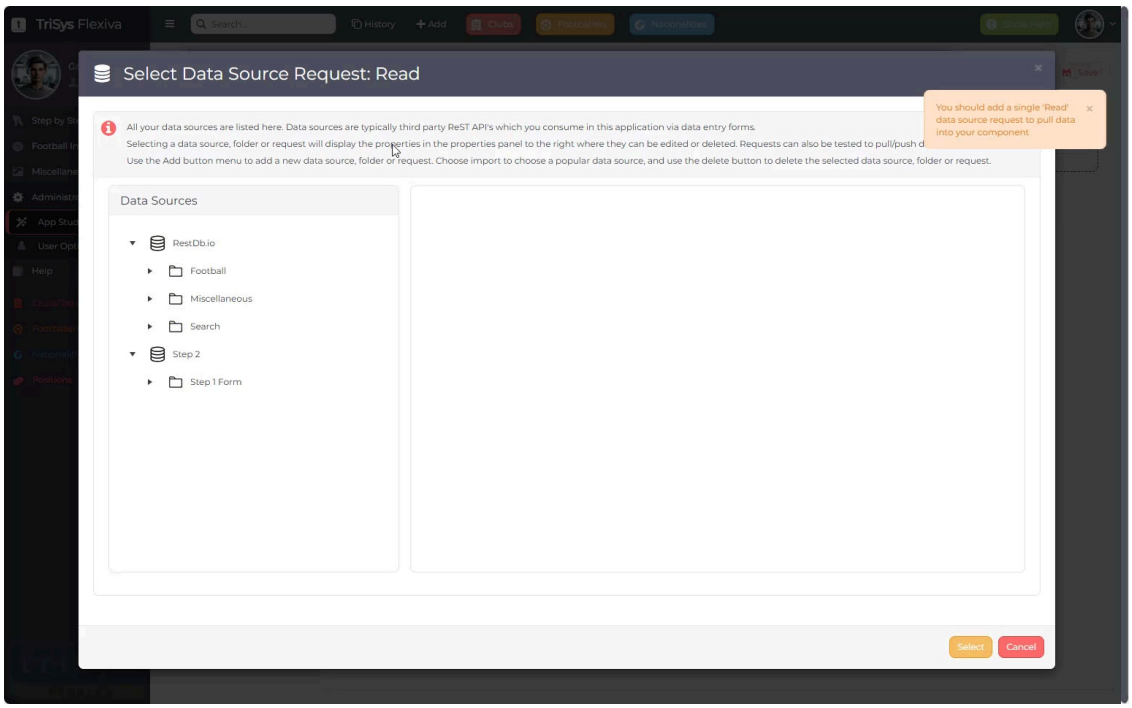


Once the grid has been dropped, the grid will be shown:



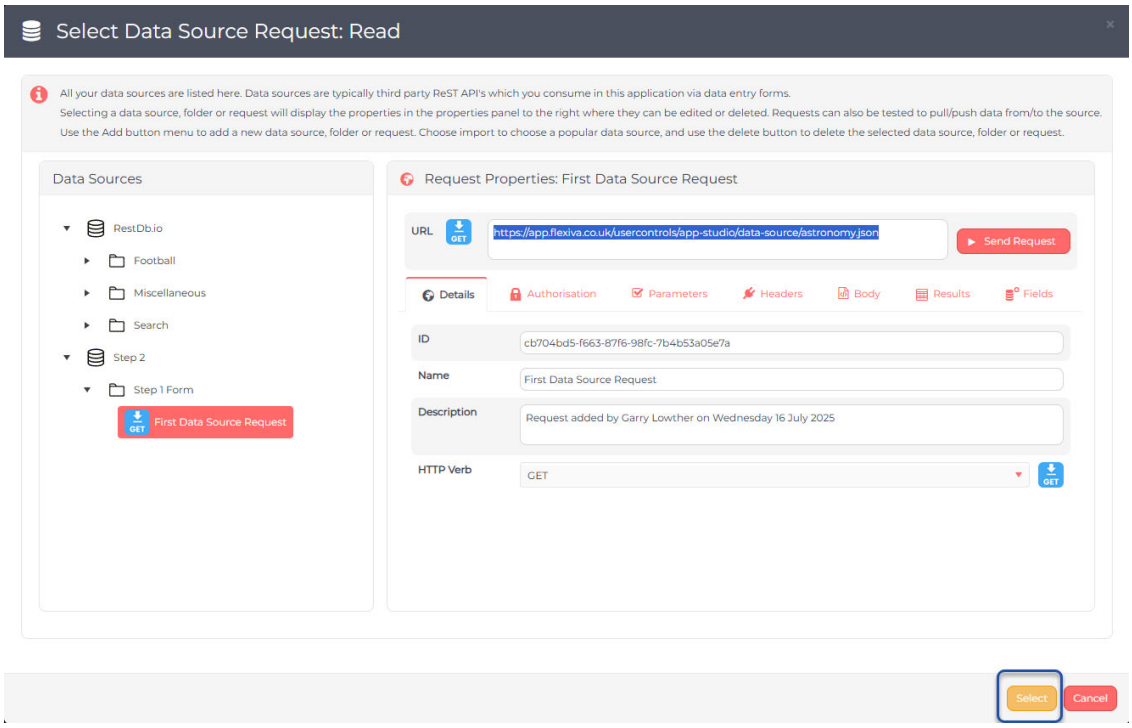
Edit Grid Properties

Click on the Grid component which opens the Component Properties: Grid modal popup, however the system will then prompt you immediately to connect a data source with the prompt: "You should add a single 'Read' data source request to pull data into your component":

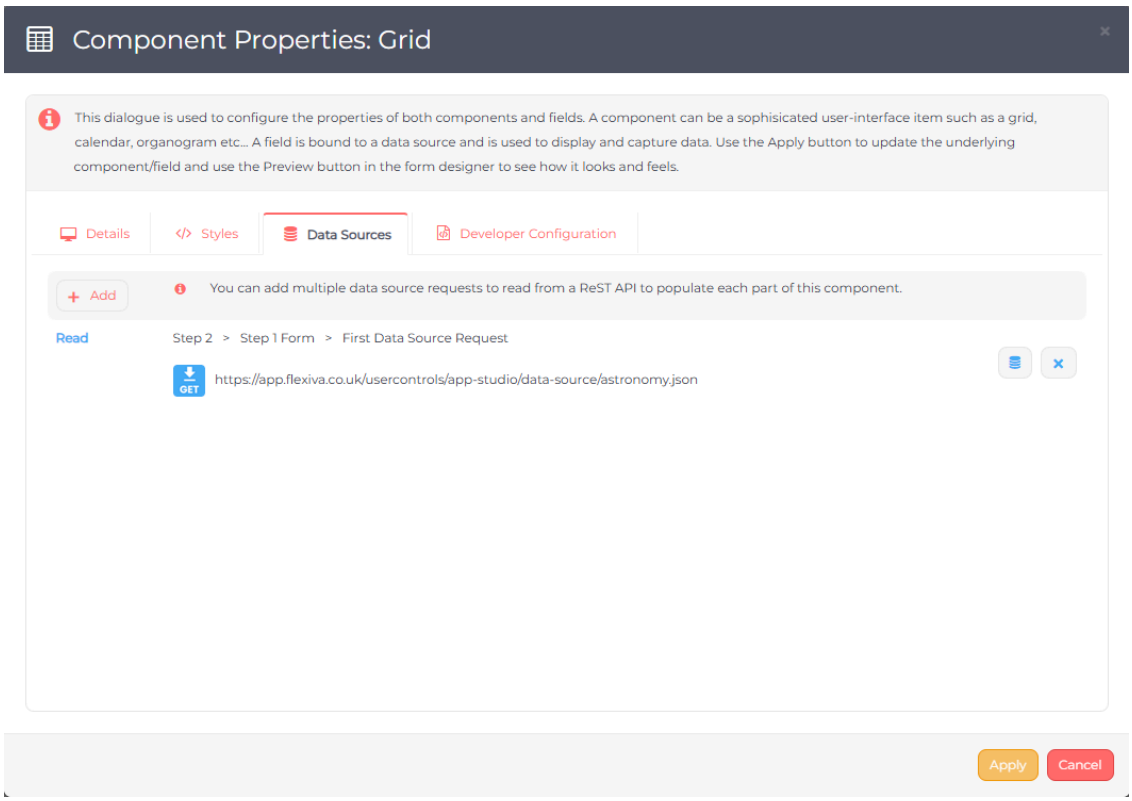


Choose Data Source Request

Locate the data source request you created in [step 2](#) and click the Select button:



The selected data source request will appear in the Data Sources tab:



Apply

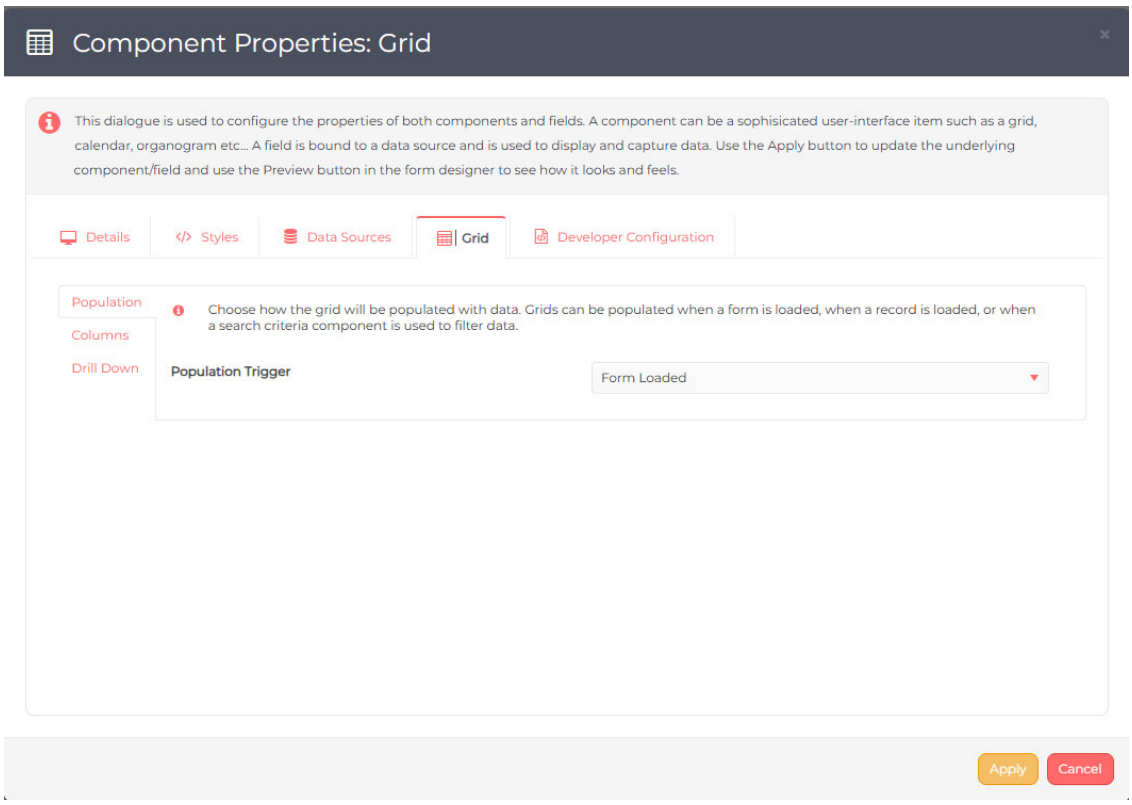
Apply your changes to close the properties popup.

Save

Save your changes using the form designer Save button top right.

Configure Grid Data

Click on the Grid component on your form to re-open the Component Properties: Grid modal popup. This time the Grid tab is visible:

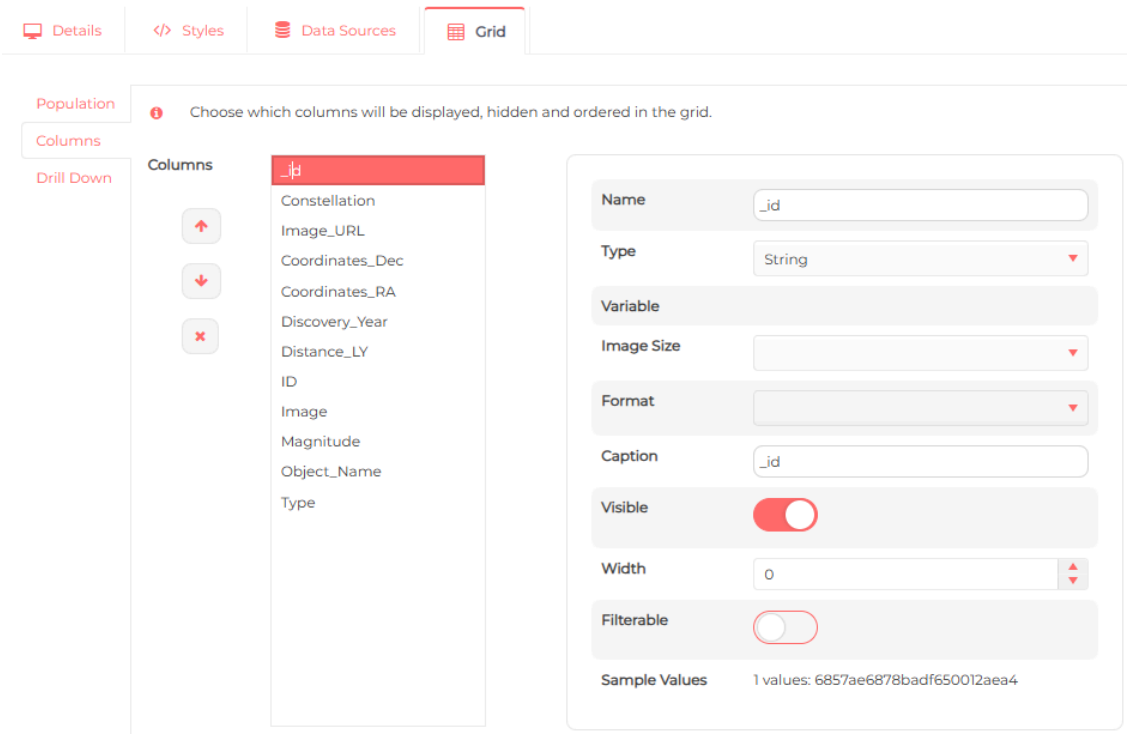


Population

The population tab shows how the grid is populated. Leave it to Form Loaded so that it will populate itself when the form loads.

Columns

The columns tab will show all of the columns with the same name as the fields returned by the ReST API. API's typically talk about fields and records, but data grids talk about columns and rows.

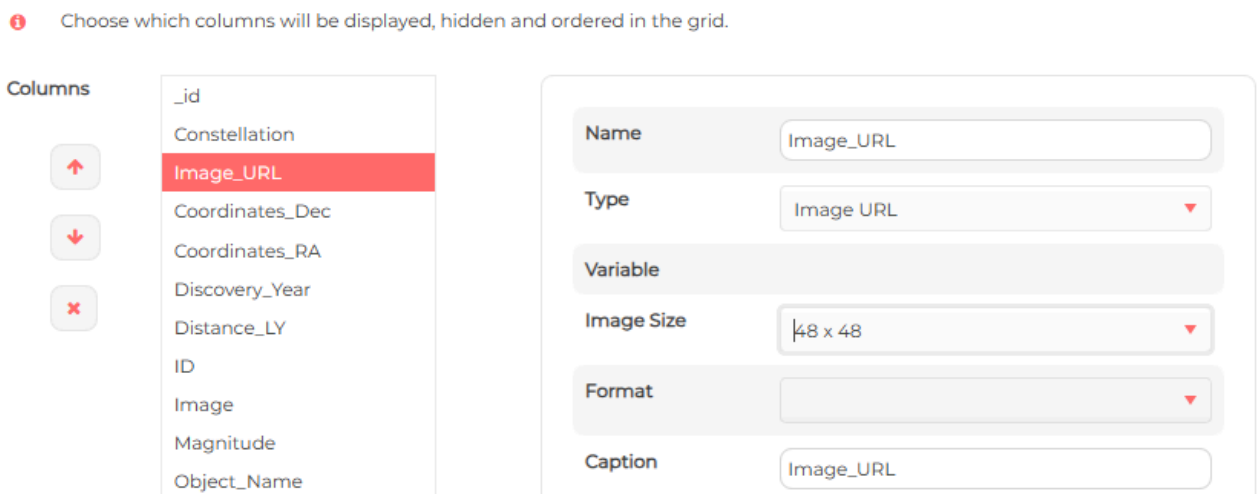


Up, Down and Delete Buttons

These buttons to the left of the column list respectively allow the selected column to be moved up, down, or removed.

Column Properties

The column properties allow you to format each column. The only thing you should change is for the Image_URL column, set the Image Size to be 48 × 48.



Apply

Use the Apply button to apply your changes to the grid.

Save

Save your changes using the form designer **Save** button top right.

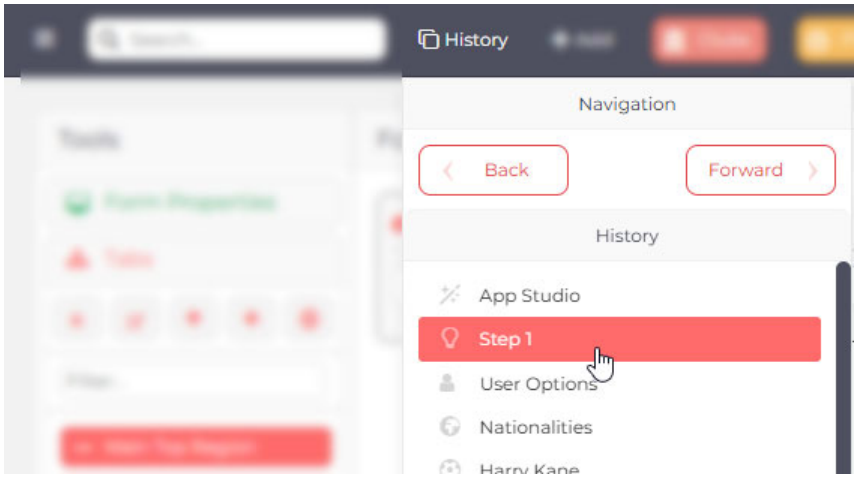
Publish and Test

Publish your changes and test your application.

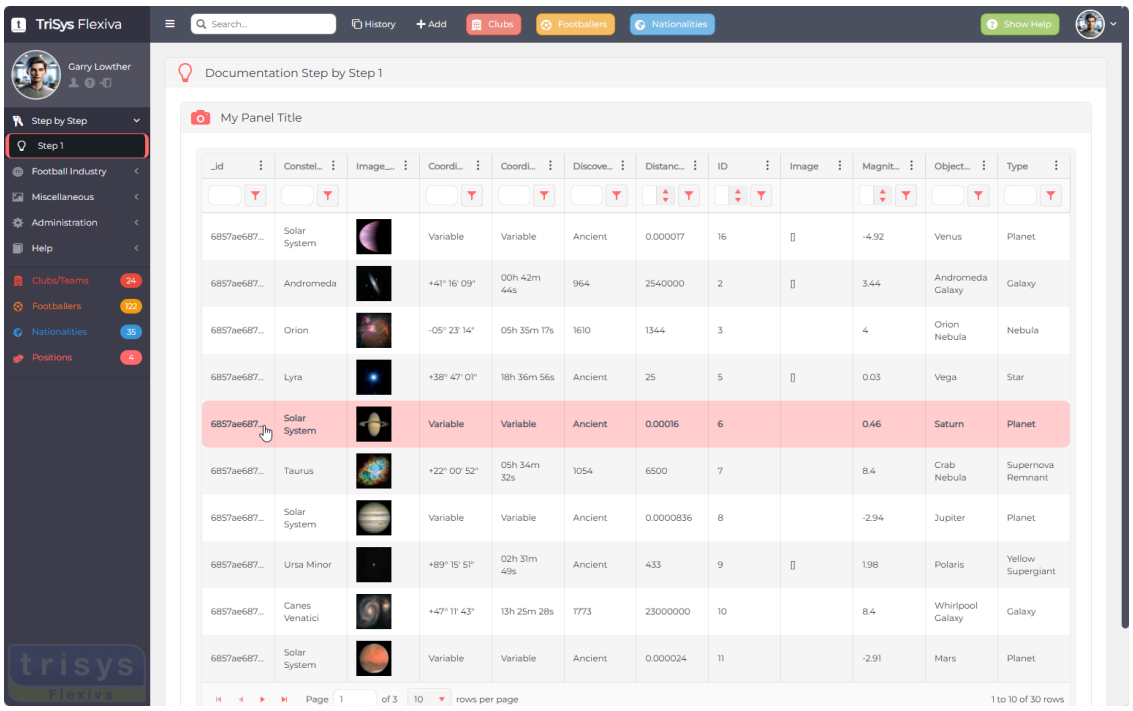
Because you are designing using App Studio, your changes are already published and ready for you to test.

Open Form from History Menu

When you last opened your first lookup form, it should be visible in the **History** menu from where you can click it to re-open your lookup form:



Your lookup form should display showing your grid and all of the data from the ReST API data source you configured in [step 2](#):


TriSys Flexiva
Search...
History
Add
Clubs
Footballers
Nationalities
Show Help
Garry Lowther
Step by Step
Step 1
Football Industry
Miscellaneous
Administration
Help
Clubs/Teams 24
Footballers 122
Nationalities 35
Positions 4
Documentation Step by Step 1
My Panel Title
_id
Constel...
Image_...
Coordi...
Coordi...
Discove...
Distanc...
ID
Image
Magnit...
Object...
Type
6857ae687... Solar System Variable Variable Ancient 0.000017 16 [] -4.92 Venus Planet
6857ae687... Andromeda +41° 16' 09" 00h 42m 44s 964 2540000 2 [] 3.44 Andromeda Galaxy Galaxy
6857ae687... Orion -05° 23' 14" 05h 35m 17s 1610 1344 3 4 Orion Nebula Nebula
6857ae687... Lyra +38° 47' 01" 18h 36m 56s Ancient 25 5 [] 0.03 Vega Star
6857ae687... Solar System Variable Variable Ancient 0.00016 6 0.46 Saturn Planet
6857ae687... Taurus +22° 00' 52" 05h 34m 32s 1054 6500 7 8.4 Crab Nebula Supernova Remnant
6857ae687... Solar System Variable Variable Ancient 0.0000836 8 -2.94 Jupiter Planet
6857ae687... Ursa Minor +89° 15' 51" 02h 31m 49s Ancient 433 9 [] 1.98 Polaris Yellow Supergiant
6857ae687... Canes Venatici +47° 11' 43" 13h 25m 28s 1773 23000000 10 8.4 Whirlpool Galaxy Galaxy
6857ae687... Solar System Variable Variable Ancient 0.000024 11 -2.91 Mars Planet
Page 1 of 3 10 rows per page
1 to 10 of 30 rows
trisys
Flexiva

Summary

A quick summary of the steps followed to construct a simple web application lookup form.

Congratulations on completing the beginner designer guide and building your own lookup form application.

You followed a three step process involving these key mechanisms:

- Created a lookup form
- Added it to the navigation bar
- Created a data source
- Designed your form by dragging on a grid component
- Configured it to connect to your data
- Tested that it worked

You should now be familiar with the basic techniques required to configure a web application.

The next section is aimed at building a production quality client-side application using production quality ReST API's.

Production Designer Guide

Introduction

Design a production quality CRUD business application.

This section details a methodical approach to building a production quality application. These are effectively the same steps taken to build the sample reference app using the same ReST API requests.

This type of documentation could be referred to as '*the bible*', a term used to describe previous product documentation designed for implementors.

Prerequisites

You should read the first few chapters in the knowledge base, as well as followed the beginners guide to create a simple lookup form.

Most importantly, you should know which ReST API's you wish to use and what CRUD forms you require to fulfil your business requirements.

Your Application Concept

You should have a clear idea about what your application should be doing.

You may be using a ReST API from a single source, or you may be consuming multiple ReST API's from a variety of sources.

This should fit our approach and tooling to provide your colleagues with a web and mobile application CRUD front-end to your ReST API's.

Next Steps

The next steps are to start work on building production quality data sources, custom variables and forms to knit together the core of the front-end application.

The ReST API

The back-end API we will be using for building this reference application.

Typically, your back-end data will be accessible through a ReST API which will expose CRUD functionality.

We decided upon a small data model of something which had images, then built an entity relationship model, and then built the ReST API.

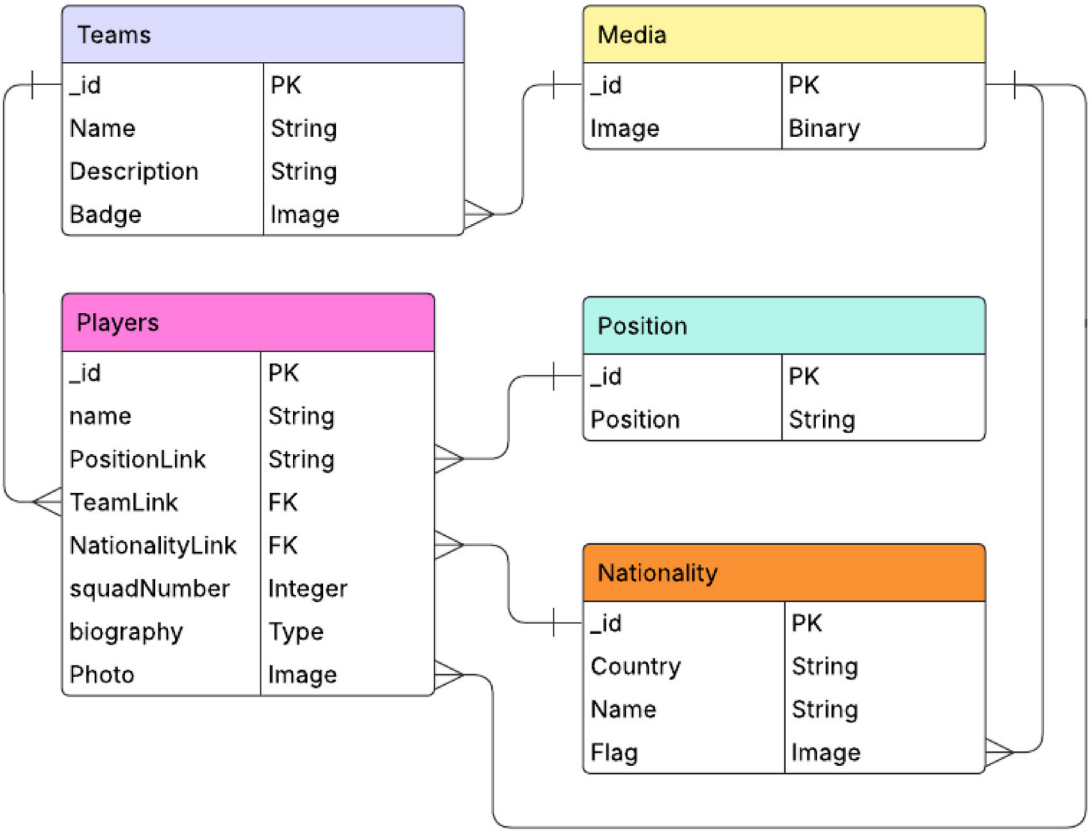
For the purposes of this guide, we chose to utilise one of the many cloud database vendors which provide both storage and a ReST API.

Data Model

We chose to model data which was freely available in the public domain and had no licensing restrictions. We decided upon a small part of the football (soccer) industry which had images. The model will have to support all data CRUD operations.

Entity Relationship Diagram

We chose to model this as a simple ER diagram using Lucidchart ↗:



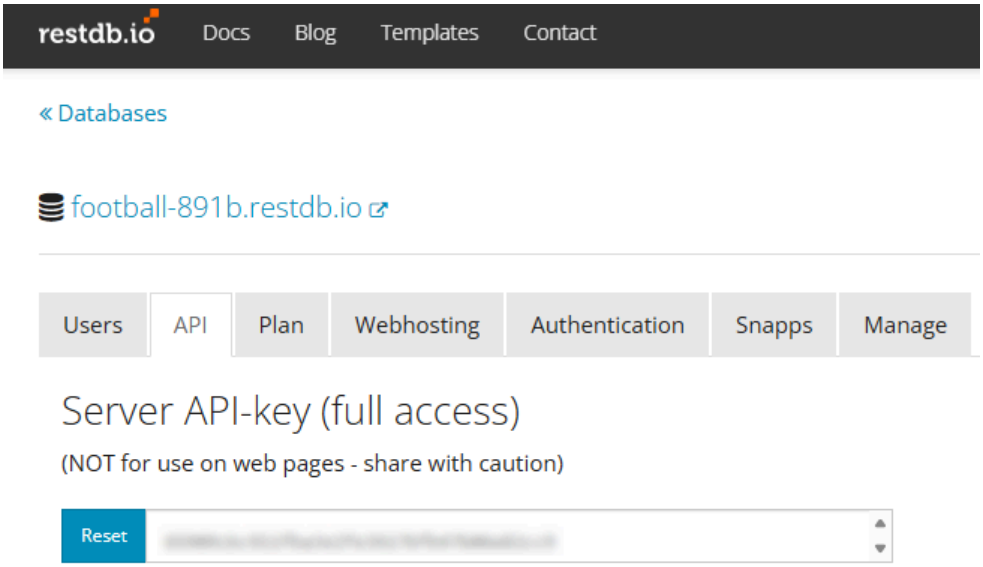
Expression	Meaning
PK	Primary Key which uniquely identifies a record. Typically this is also known as a 'surrogate' which is automatically generated when a record is created.
FK	Foreign Key which links to a primary key in another table. This is the essence of a relational database.
Image	A binary representation of a media file e.g. PNG, GIF, JPG etc.. We have photos, badges and flags in this sample database which are all images.
Crows Feet	The 'crows feet' links between tables implies a one-to-many relationship e.g. a player has one and only one position, however each position can be assigned to zero or many players.

restdb.io

We selected restdb.io ↗ because it was simple to configure, at a reasonable price.

Security

restdb.io prevents public access to data by supplying an API Key which must be used when accessing data via the ReST API. This is where we copied our generated key from when we setup data sources to this ReST API.

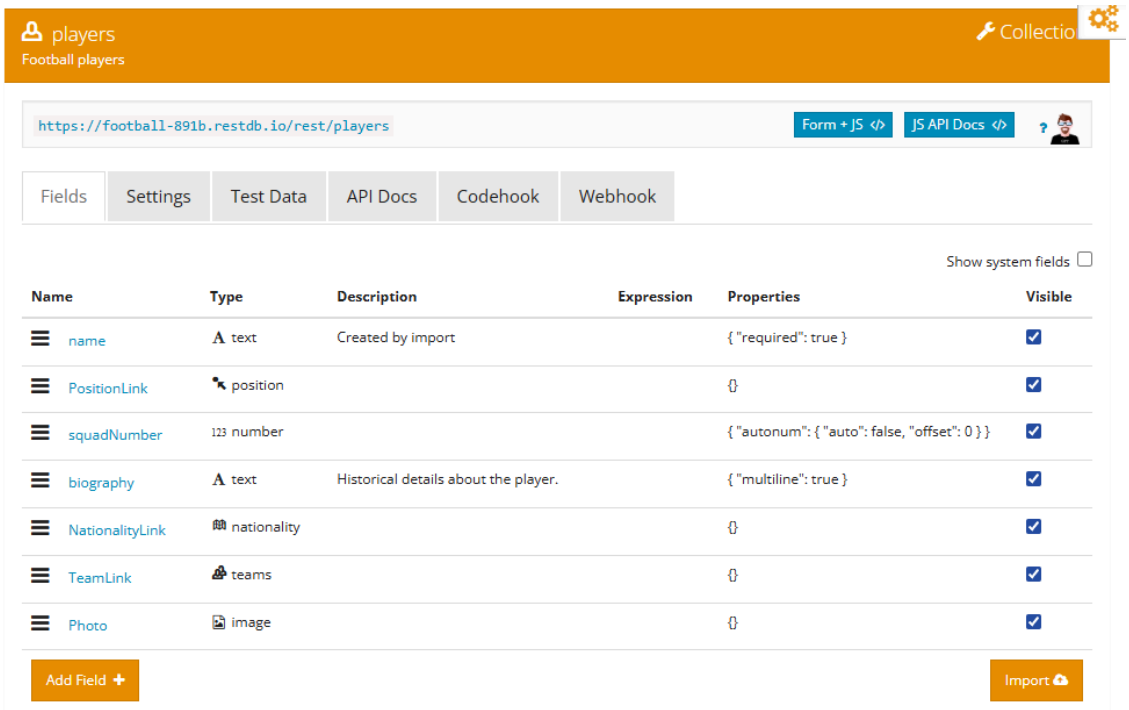


Tables/Entities/Documents/Collections

Typically, relational/SQL database systems store data in tables. No-SQL database systems such as restdb.io use the term 'document' or 'collection' to describe the same thing i.e. a table of data with rows/records and columns/fields. Whilst the former are abstractions, entity is more of a real-world description e.g. a product or a person.

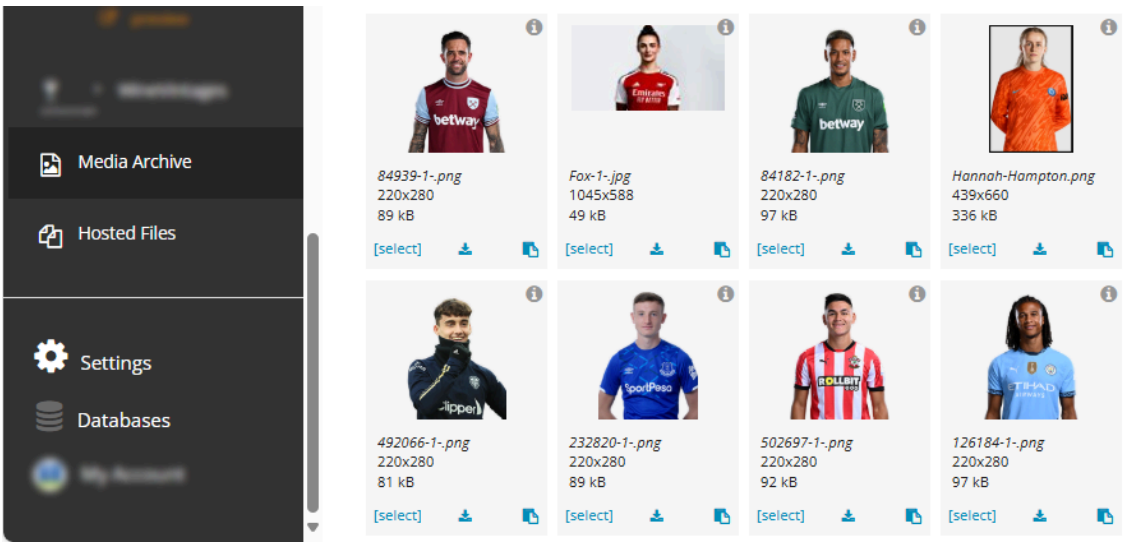
We will refer to this in data storage terminology as a table.

This is how we created the Players table.



Media

restdb.io handles media files such as images in its own media archive:



This is an important consideration when using or creating a ReST API which supports media files.

Importing Data

You can manually create data to get started in restdb.io and this is good for creating the first small set of records to prove that the data model is correct.

If however you already have a large JSON data set, you can import that directly into your restdb.io table so that you can test performance if necessary. Be aware that if your data model has foreign keys, importing data will not create records in linked tables.

Of course the goal is to integrate your ReST API with Flexiva to create CRUD forms where end-users can create and maintain your data, so this is preferrable to importing data.

Viewing Data

When we are logged into restdb.io, we can view raw data using the URL for each table:

https://football-891b.restdb.io/rest/players?max=2

Pretty-print ☑


```
[
  {
    "_id": "674cb1cb050c58540003e208",
    "name": "Bruno Fernandes",
    "PositionLink": [
      {
        "_id": "6756e649050c585400050d90",
        "Position": "Midfielder",
        "_created": "2024-12-09T12:44:57.685Z",
        "_changed": "2024-12-09T12:44:57.685Z"
      }
    ],
    "squadNumber": 8,
    "biography": "International midfielder and club captain.",
    "NationalityLink": [
      {
        "_id": "67f7cb1478badf650004fb4d",
        "Country": "Portugal",
        "Name": "Portuguese",
        "_created": "2025-04-10T13:43:48.701Z",
        "_changed": "2025-04-14T19:08:45.353Z",
        "Flag": [
          "67fd5d3578badf650005ba96"
        ]
```


We can see that this is a hierarchical JSON data set which we can consume in our data source requests, however a ReST API does far more than display raw data.

Views

Typically back-end databases comprise dozens if not hundreds of tables, all connected via foreign keys and primary keys.

Client-side applications do not need, or indeed wish, to know about the internal details of the data model in the back-end.

By adopting 'separation of concerns' or 'abstraction' it means that the server can send two dimensional tables sourced form multiple tables. It does this by creating 'views' of data by joining tables together.

The client-side app only gets this 2D table which is both efficient and easy to understand.

Typically, relational databases have SQL Views which join tables together declaratively. No-SQL databases such as restdb.io require programming to create views.

The benefit of restdb.io views however is that they can be remoted as a ReST API automatically, whereas a relational database SQL view will need an additional API layer in order to remote that data.

The swings and arrows of outrageous fortune.

Building the restdb.io Views

We utilised 'vibe-coding' with an AI LLM to get the basics of the views in place, however this was arguably no more efficient in terms of time than manually coding these views.

A GET/DELETE ReST API will typically have URL arguments which are used to filter the data e.g.

`https://api.domain.com?name=fred&position=goalkeeper&club=arsenal`

These arguments need to be manually coded in the restdb.io view.

A POST/PUT/PATCH ReST API will typically have a body containing data e.g.

```
{
    "ClubID": "abcdefgh123",
    "Club": "Norwich City",
    "PlayerID": "12345678",
    "Photo": "....."
}
```

These body fields need to be manually handled for CRUD operations in the restdb.io view.

The List of ReST API CRUD Views

This is the list of ReST API views which we will be using in the following pages all located at one of these two end-points:

https://restdb.trisys.co.uk ↗

https://www-football-891b.restdb.io/views/

ReST API / View	HTTP Method	Function
`CountTable`	`GET`	Count the records in each table. This is used for activity metrics.
`CreateFootballClub`	`POST`	Create a football club record
`CreateFootballer`	`POST`	Create a footballer record
`CreateNationality`	`POST`	Create a nationality record
`DeleteFootballClub`	`DELETE`	Delete a football club record
`DeleteNationality`	`DELETE`	Delete a nationality record
`FootballClubs`	`GET`	List all football clubs
`FootballerClubs`	`GET`	List all football clubs where a footballer has played
`Nationalities`	`GET`	List all nationalities
`ReadFootballClub`	`GET`	Read a football club record
`ReadFootballers`	`GET`	List all footballers
`ReadFootballersPaged`	`GET`	List a page of footballers using pagination parameters
`ReadNationality`	`GET`	Read a nationality record
`Search`	`GET`	Search over the entire database. This is used in the app toolbar search.
`Team`	`GET`	Read a team/club record
`UpdateFootballClub`	`PATCH`	Update a football club
`UpdateFootballer`	`PATCH`	Update a footballer record
`UpdateNationality`	`PATCH`	Update a nationality record

Postman

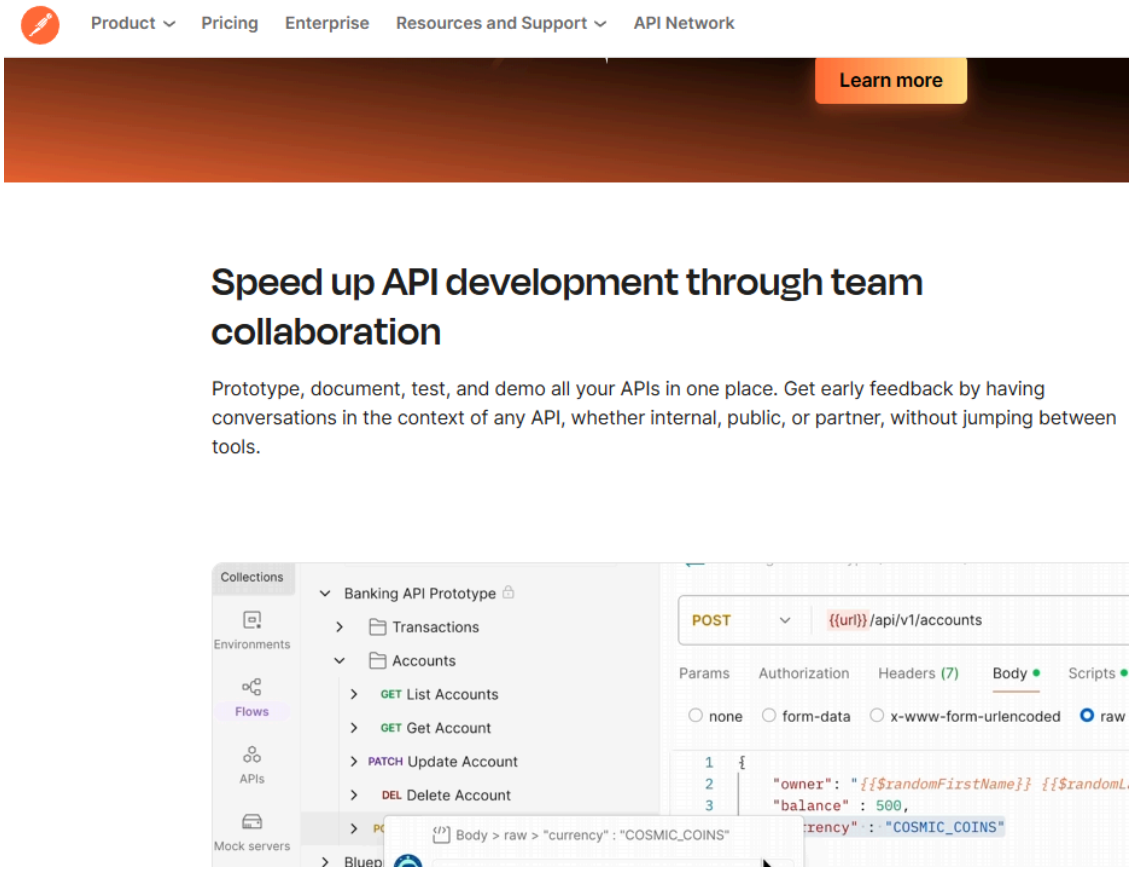
It is recommended that all of these ReST API's are thoroughly tested in a tool like Postman before integration into Flexiva.

Postman

Using the Postman tool to independently test ReST API's.

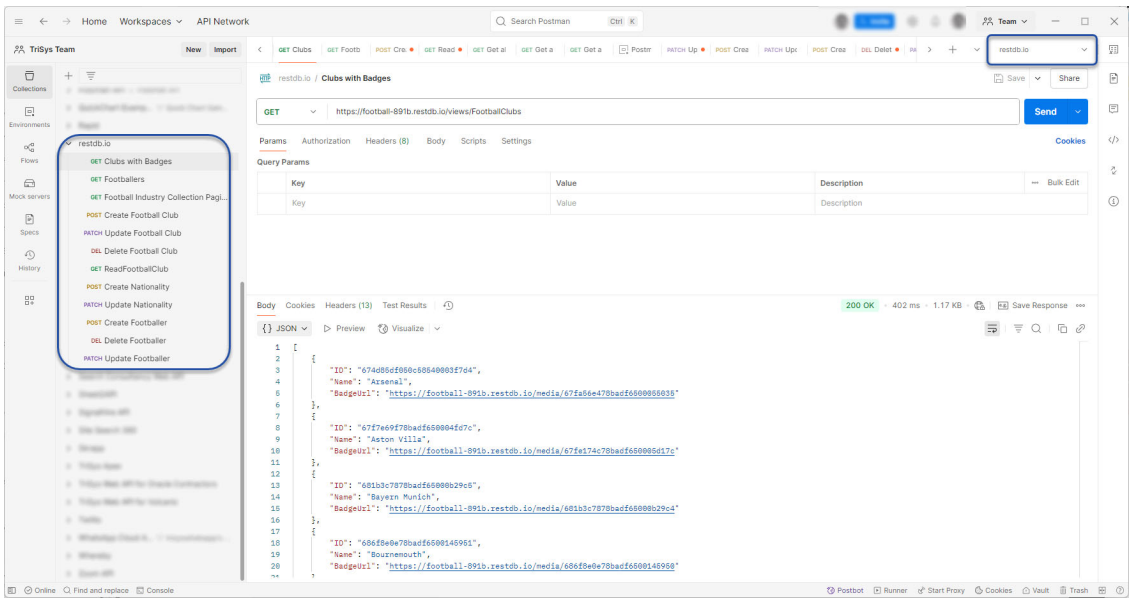
It is generally 'best practice' to test the back-end ReST API before linking this to a client-side application. This is usually done by the developers of the ReST API who supply these tests to the front-end team as part of their integration documentation.

The most well known tool for this is called [Postman ↗](#).



Collections

ReST API developers will normally create a Postman collection, which is a folder containing all of the published ReST API functions:



This is our Postman collection for the restdb.io ReST API from the previous page. We can see the API functions on the left and the 'environment' top right which is where the security credentials are stored for this specific collection. We tested this specific endpoint and it returned the JSON data.

Environments

Each collection is usually associated with an environment which is where the security credentials are stored and used as environment variables in the URL, or headers or body of the request.

Testing

Of course the principal benefit in using Postman is to be able to test the ReST API's independently, another important separation of concerns. This allows a quality assurance (QA) team to sign off that the ReST API is both functional and performant before it is integrated into Flexiva.

Getting Started

Integrating a ReST API into Flexiva has to start somewhere.

One of the most important things when building systems is where to start, however this is not as important as actually starting.

Luckily, adopting Flexiva as your front-end application, means that you have already eliminated a huge amount of design, engineering and learning because Flexiva is built on around 150 man years of software engineering, and around 1.5 million lines of industrial strength back-end and front-end code, used by thousands of customers for decades.

Starting the integration of your ReST API should therefore be the creation of your first data source request.

Keep it Simple

We recommend choosing a simple ReST API with as few foreign key dependencies as possible to start.

Our goal here is to create the data source, then consume that in a lookup form grid to visualise the data sourced from the back-end.

This is a simple process with very few steps:

Create the Data Source

Use this App Studio configurator to integrate the ReST API.

Create Forms

Use this App Studio configurator to create forms, and design them with components linked to data sources.

Add to Navigation Bar

Use this App Studio configurator to link menu items to new forms.

Test

Thoroughly test all CRUD operations for our ReST API.

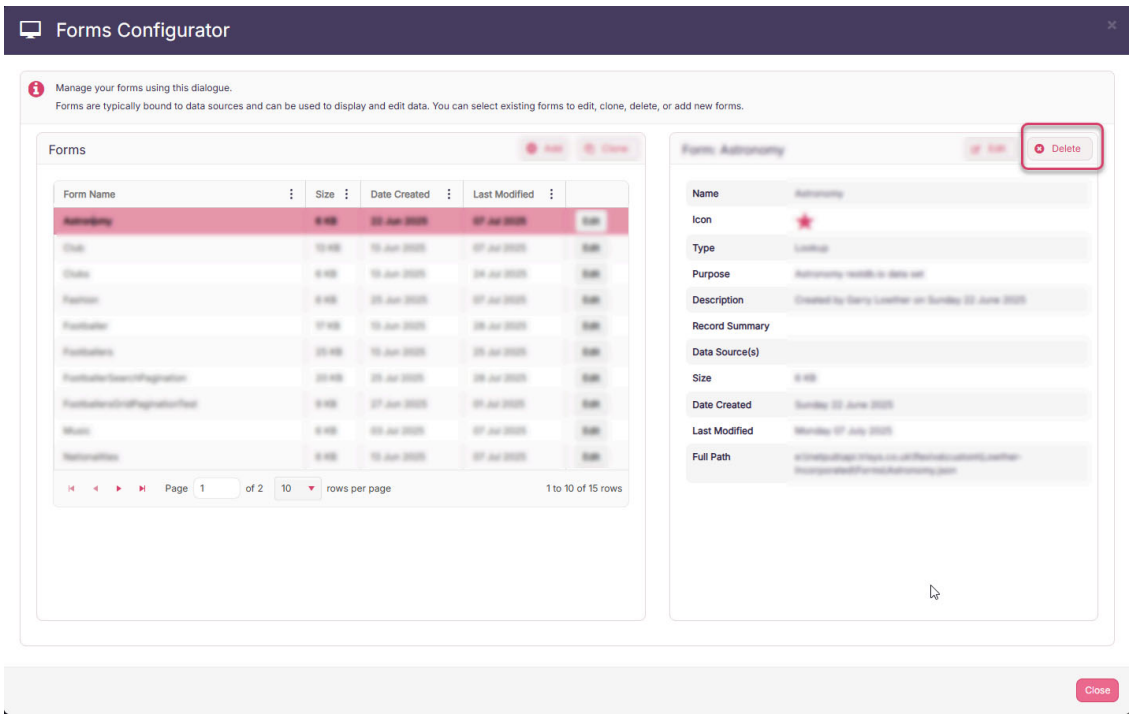
Prerequisites

If you have signed-up and are logged in to the reference app, then you may wish to remove all of the samples from your database before building them all from scratch using the process described next.

WARNING: These are destructive actions and you will be unable to use the sample app forms until after you have re-created them.

Remove Forms

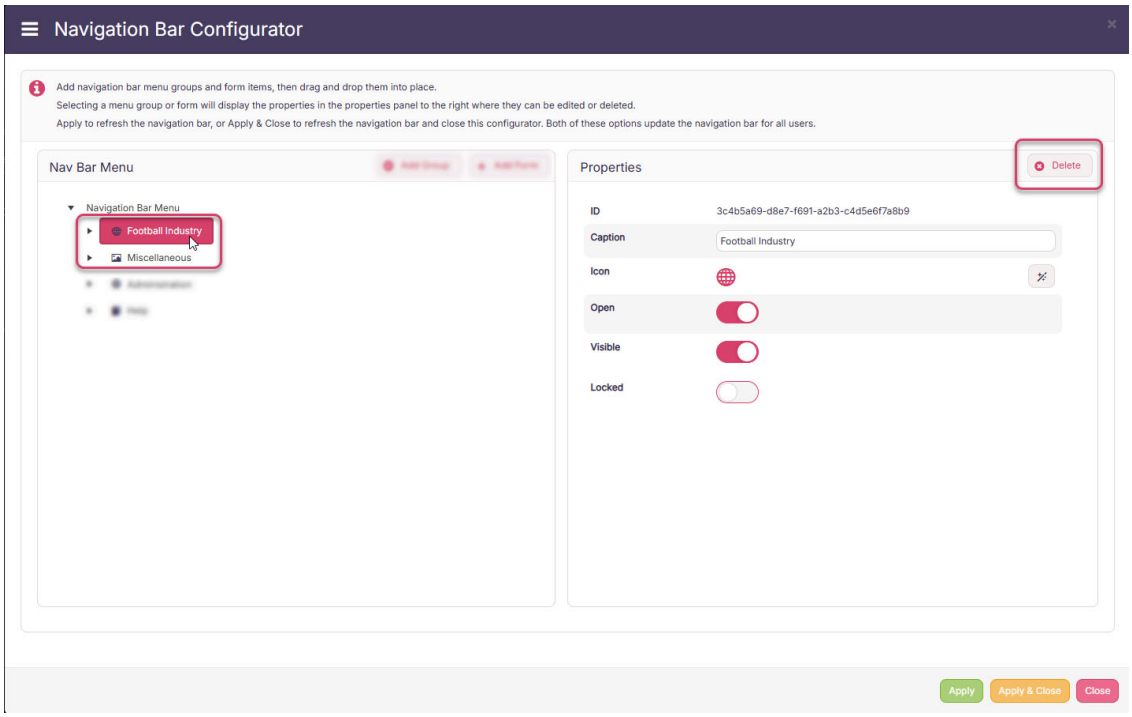
To remove all sample forms, open App Studio and select the Forms configurator:



Select each form in turn, then use the Delete button repeatedly, and confirm until all sample forms are removed. Apply & Close to make them disappear.

Remove Navigation Bar Forms

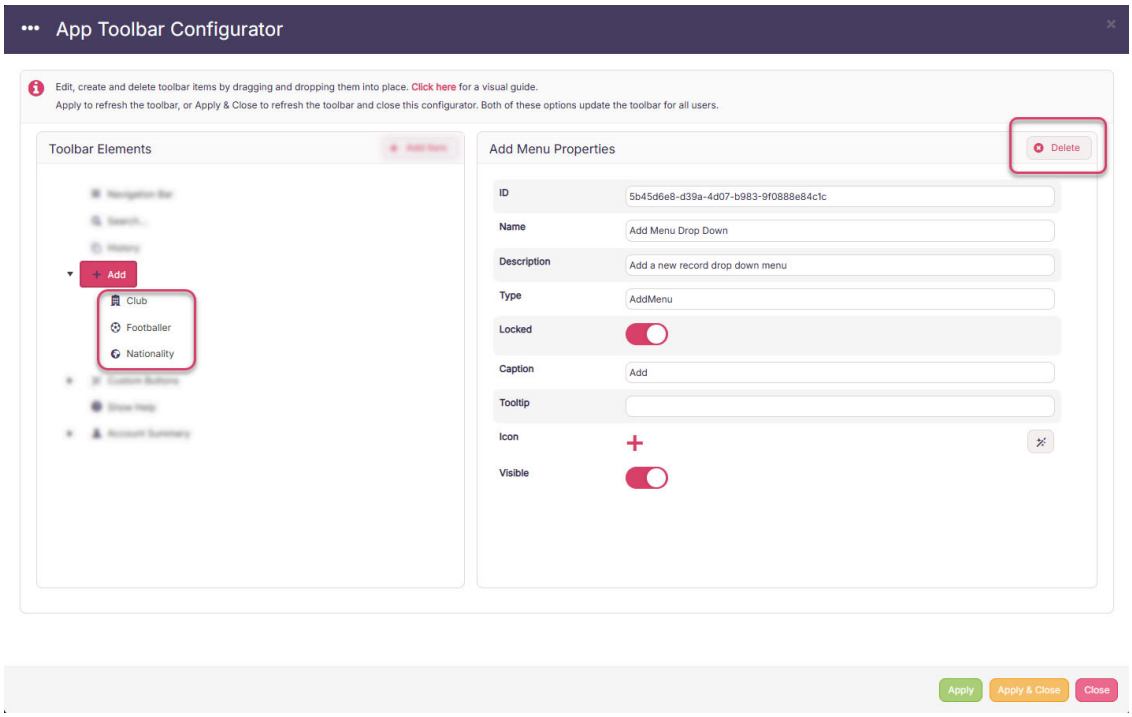
To remove all sample navigation bar groups, open App Studio and select the Navigation Bar configurator:



Select each of these 2 highlighted groups in turn, then use the Delete button repeatedly, and confirm until all sample nav bar groups are removed. Apply & Close to make them disappear.

Remove Add Menu Forms

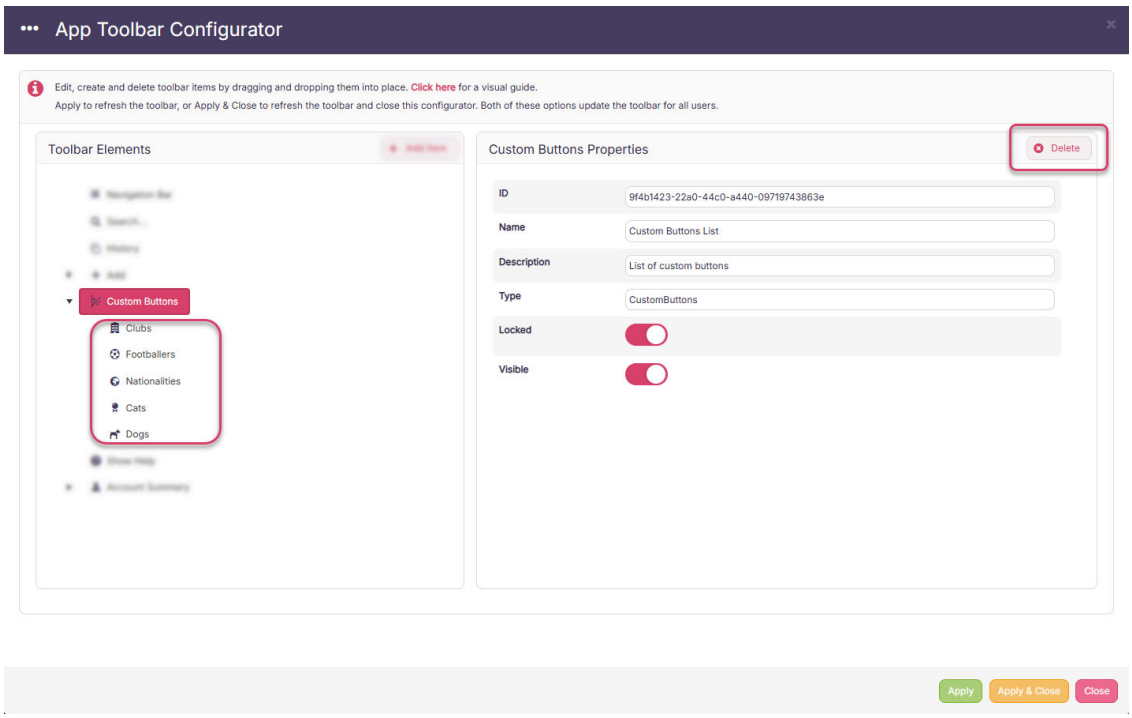
To remove all sample add menu items, open App Studio and select the App Toolbar configurator:



Select each of the forms beneath the Add node in turn, then use the Delete button repeatedly, and confirm until all sample add menu items are removed. Apply & Close to make them disappear.

Remove Custom Buttons

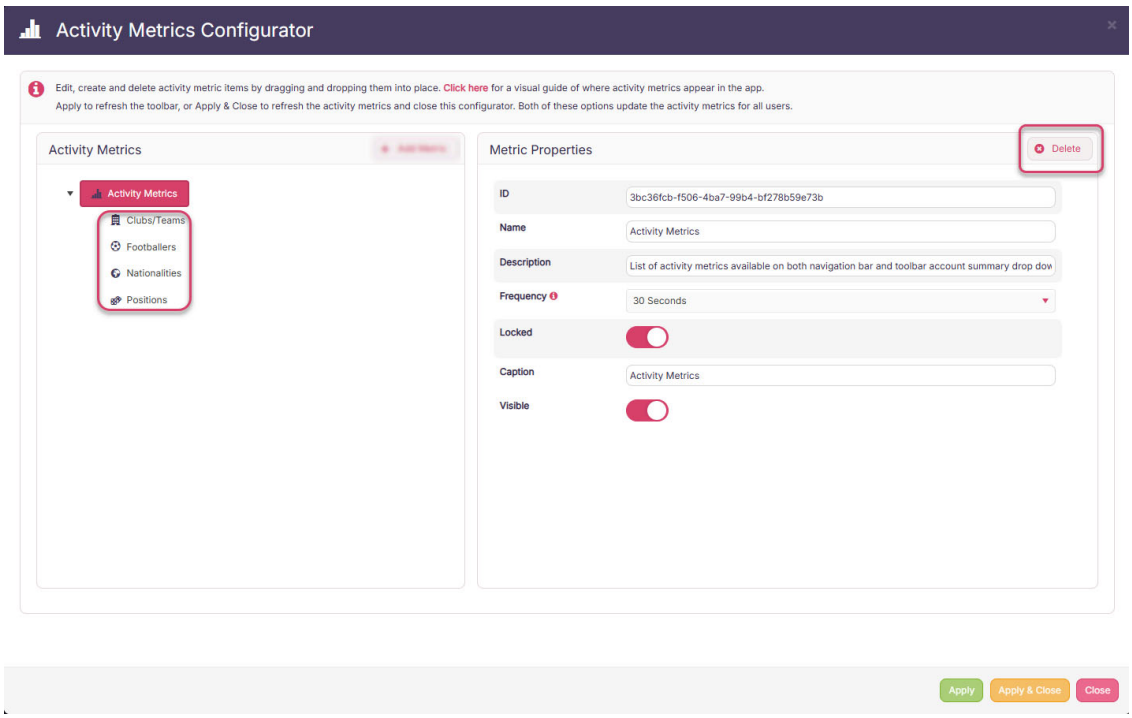
To remove all sample custom buttons, open App Studio and select the App Toolbar configurator:



Select each of the forms beneath the Custom Buttons node in turn, then use the Delete button repeatedly, and confirm until all custom buttons are removed. Apply & Close to make them disappear.

Remove Activity Metrics

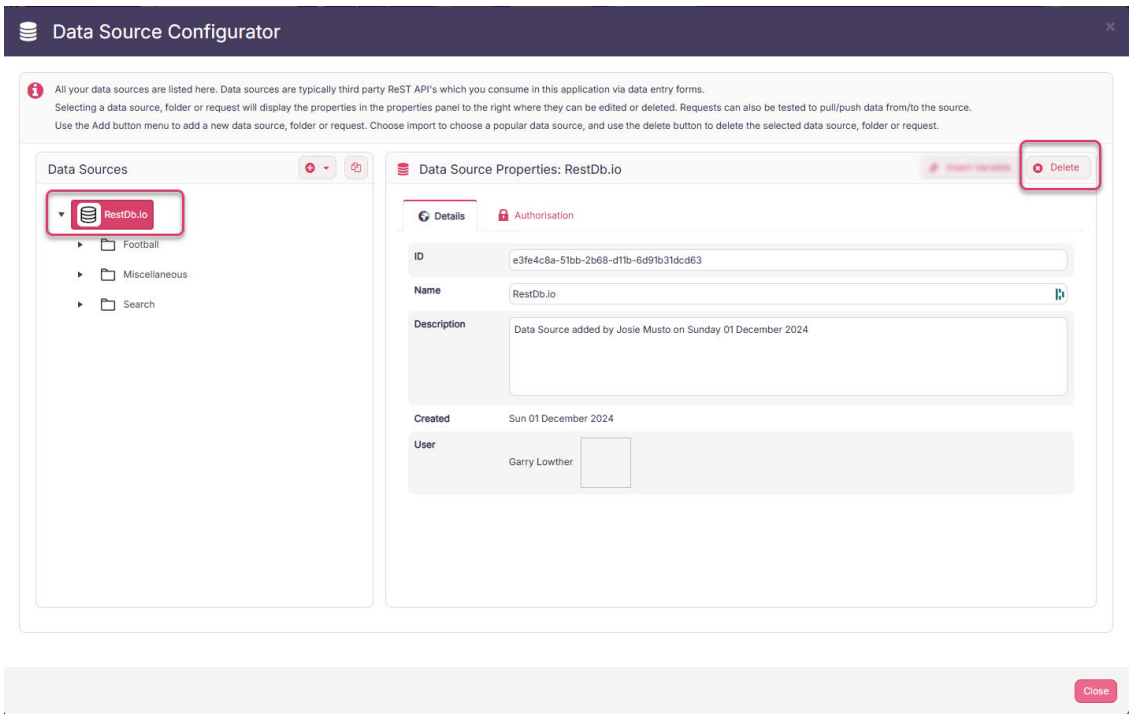
To remove all sample activity metrics, open App Studio and select the Activity Metrics configurator:



Select each of the forms beneath the Activity Metrics node in turn, then use the Delete button repeatedly, and confirm until all activity metrics are removed. Apply & Close to make them disappear.

Remove Data Sources

To remove all sample data sources, open App Studio and select the Data Sources configurator:

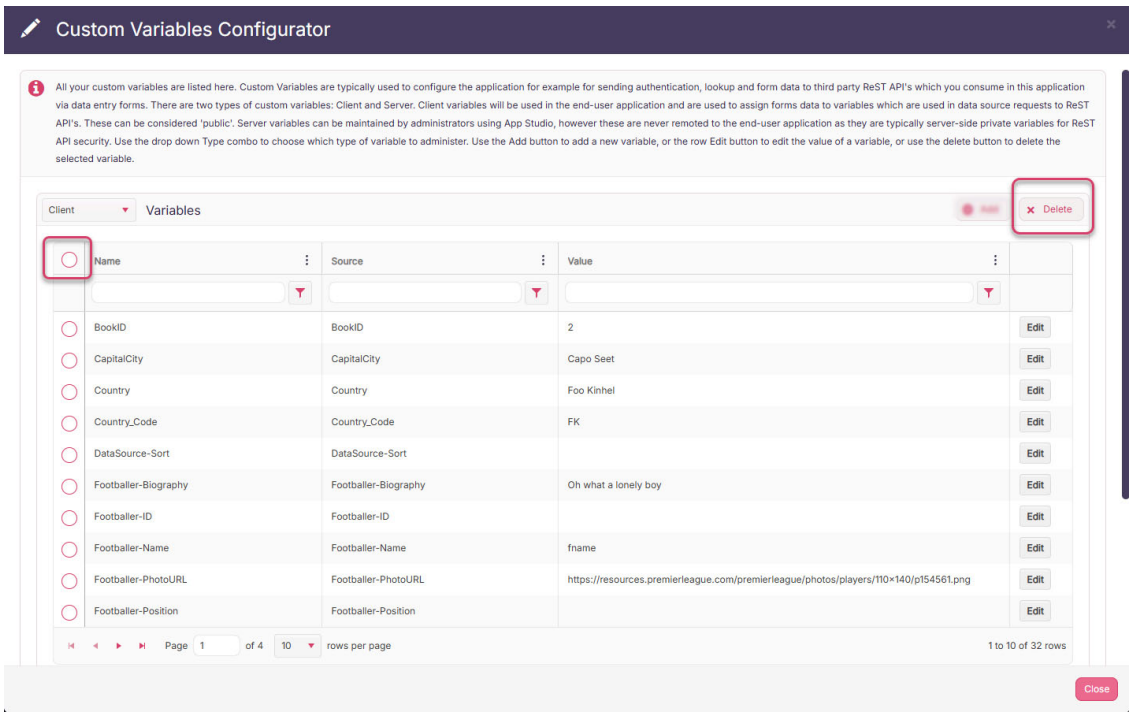


WARNING: This is the most destructive action in the App Studio so take great care.

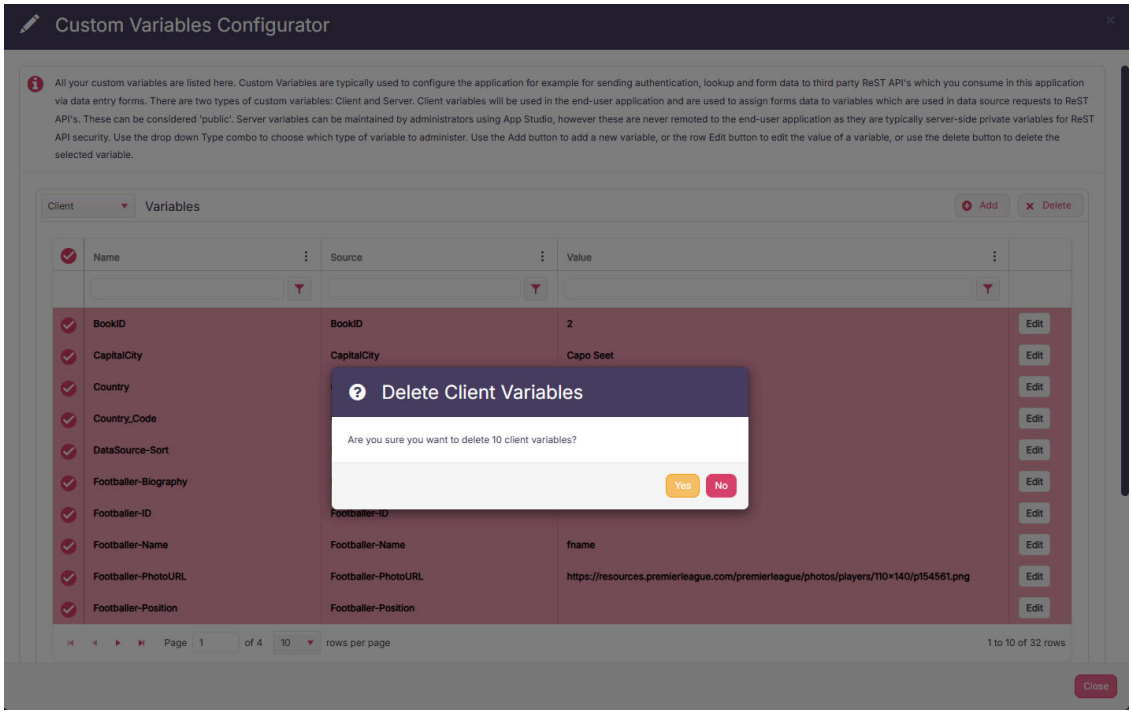
Select the RestDB.io data source, then use the Delete button, and confirm. Every folder, sub-folders and data source request will be removed.

Remove Custom Variables

To remove all sample activity metrics, open App Studio and select the Custom Variables configurator:



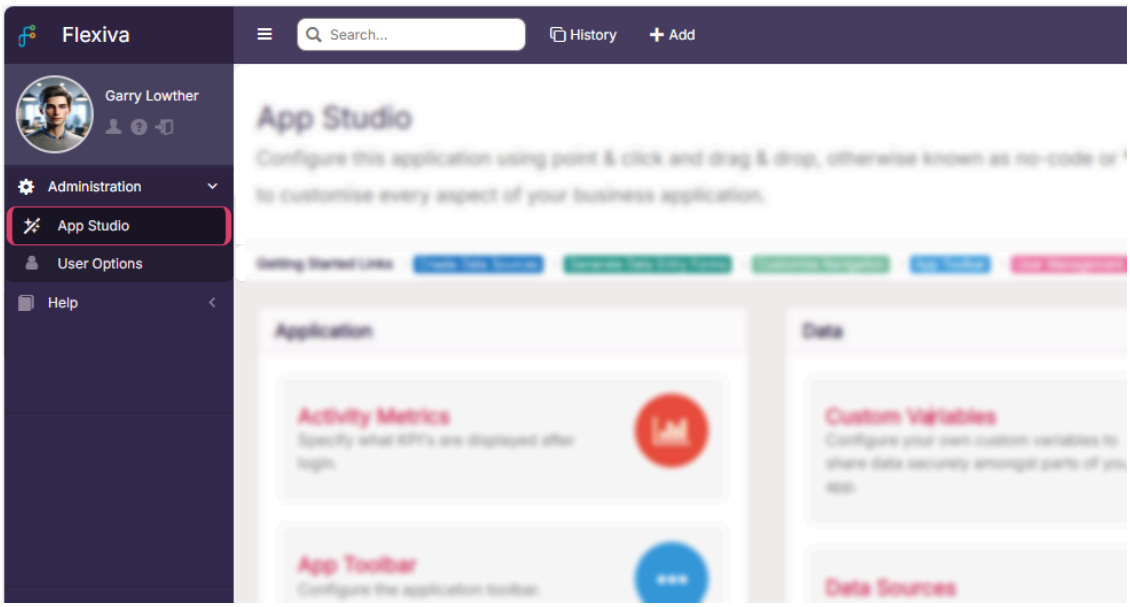
Select the left column header to select all Client variables, then press the **Delete** button:



Click Yes and keep selecting all Client variables and deleting until they have all been removed.

Sample Configuration Removed

Once you have removed all of the sample forms, toolbar items, nav bar items, add items, custom variables and data sources, your application will look like this:



You are now ready to commence re-building the samples from scratch, or indeed integrating your own ReST API's.

Nationalities Lookup Form

The first entity we will lookup is nationalities.

Because the Nationalities table is simple, and of finite size i.e. there are 195 countries in the world, then this is a good choice for the first ReST API to integrate for CRUD operations.

We know from the ReST API what the end points and associated security keys we need to get started. We will always start by reading the data, then displaying it, before moving on to editing, creating and finally deleting data.

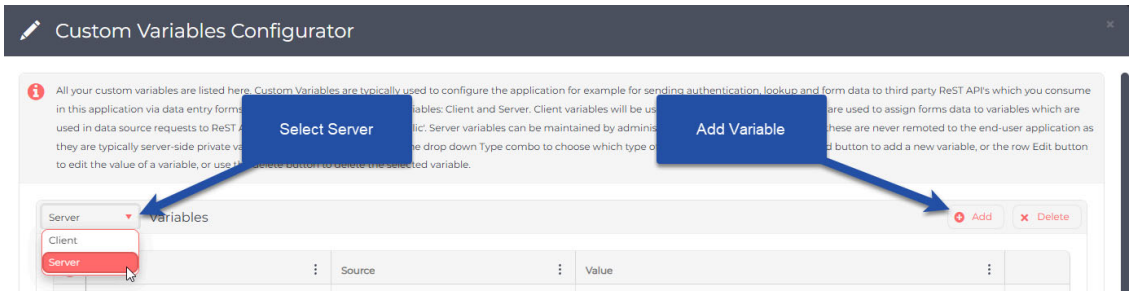
Custom Variables for Security Keys

Interestingly, the first thing to do is NOT to start creating data sources. This is because the data sources will require security credentials, and the best place to store these is in server-side custom variables which will not be remoted to the client when executing requests securely on the server.

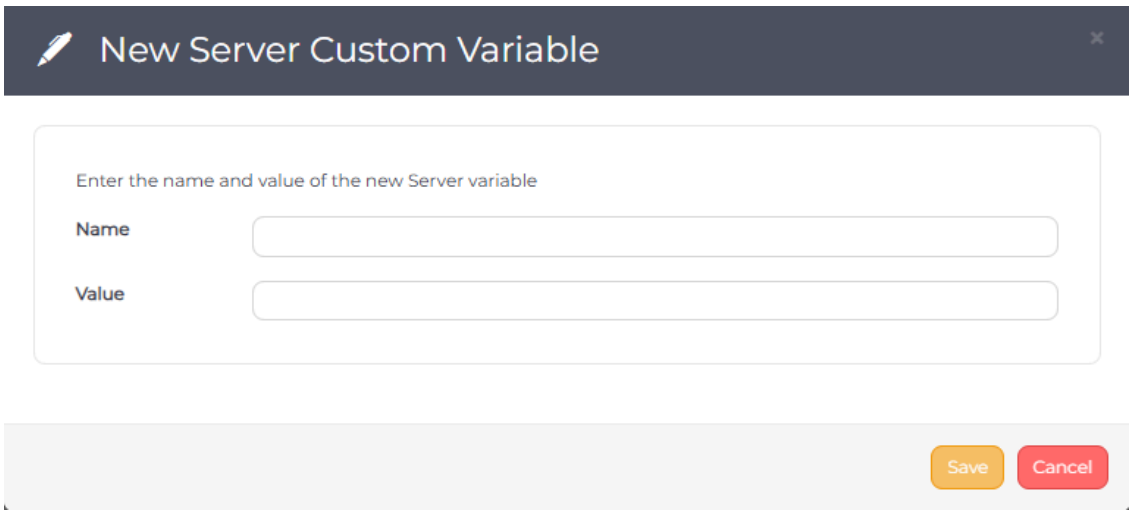
Custom Variables Configurator

Open App Studio, then open the custom variables configurator.

Select Server in the drop down combo, then click the Add button to create a variable:



The New Server Custom Variable popup modal form will open:



Name

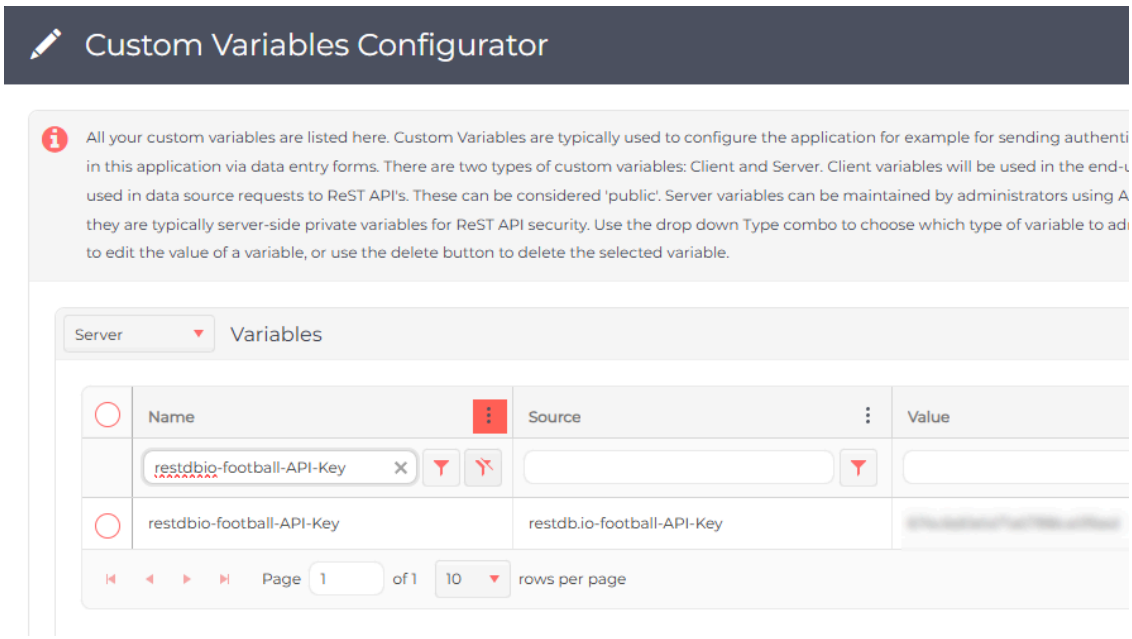
Give this variable a name easily recognisable i.e. in this case we will call it **restdbio-football-API-Key** as this is descriptive and unambiguous. We know it relates to a restdb.io ReST API and the sample models football and the security key is an API key.

Value

The value assigned to the variable is the key generated in the restdb.io database API tab shown on this page.

Save

Save the variable to create it and it will be displayed in the grid:



Custom Variables for Key Fields

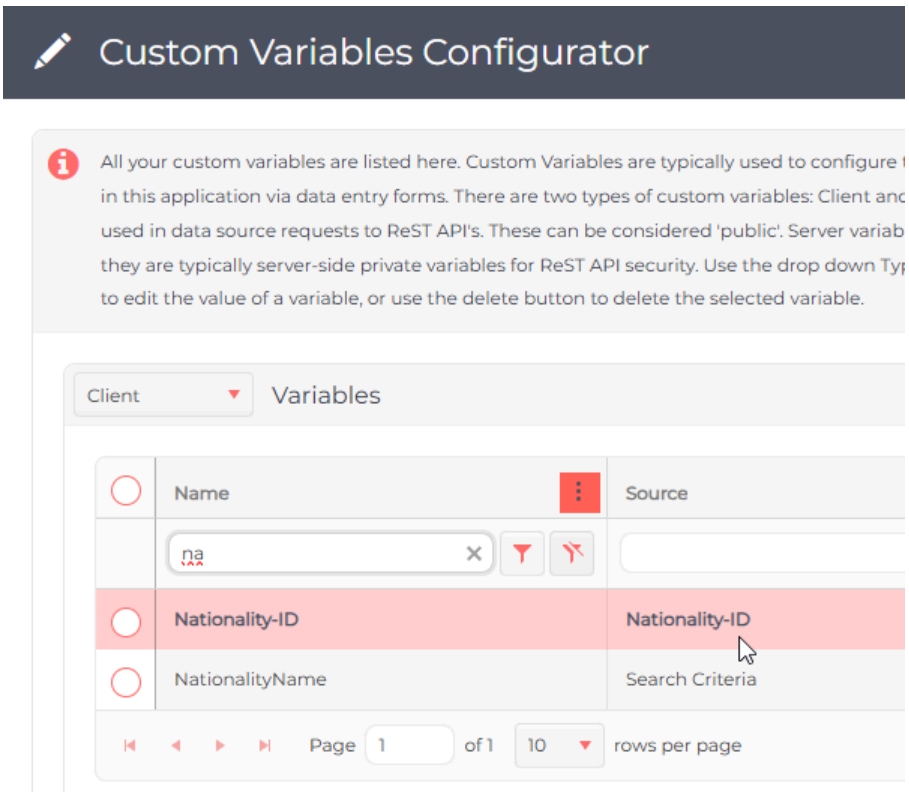
Data source requests will return a data table of rows with fields. Each row will typically have an identifier for example in this sample the Nationality ID. This will be a long random string of characters or numbers generated by the back-end database when records are created.

In order to handle the selection of rows, we need to create a custom variable which will be dynamically set when the end-user selects a specific record in a form or a grid or a field. We will assign this variable to the key field in the data source request.

Because we know that the restdb.io Rest API request will return nationalities, we will create a custom variable called "Nationality-ID" which we will assign to the field later.

Add Nationality-ID

Select the Client in the drop down this time before using the Add button to create the custom variable "Nationality-ID:



We do not need any more custom variables to display and select nationalities in the lookup form, however we will when we need to design a data entry form in order to map fields to the ReST API body.

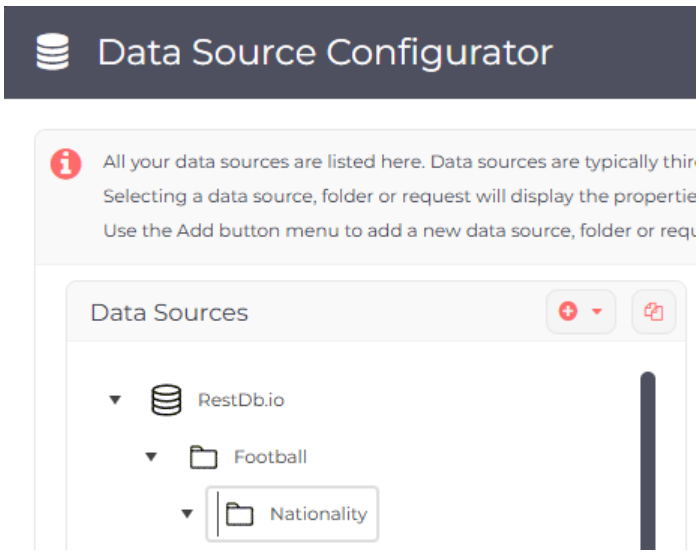
Read List

The next thing is to create the data source request to integrate the list of nationalities pulled from the ReST API.

Data Source Configurator

Open App Studio, then open the data source configurator.

Create a data source, and a sub-folder hierarchy like this:

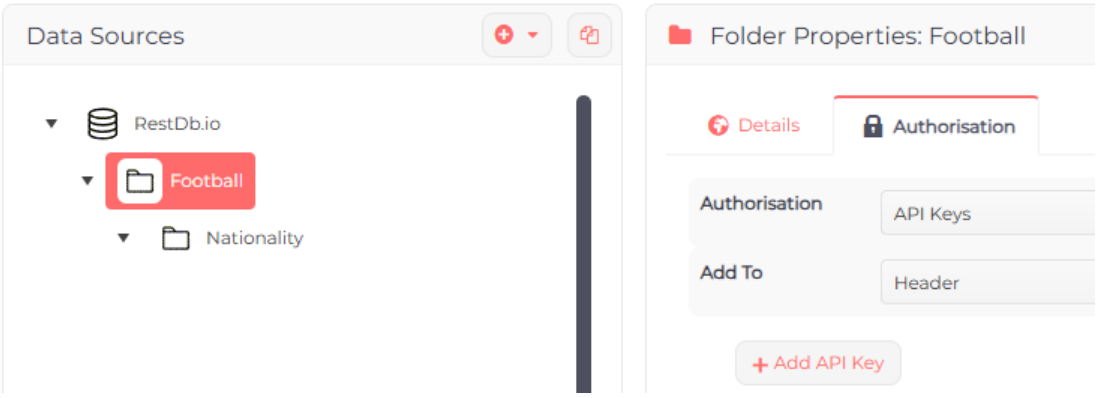


This unambiguously defines that we have a restdb.io data source inside which we have the Football industry depicted as a folder. We then define a sub folder for Nationality which will be where all the CRUD request methods for nationalities will be created.

Security

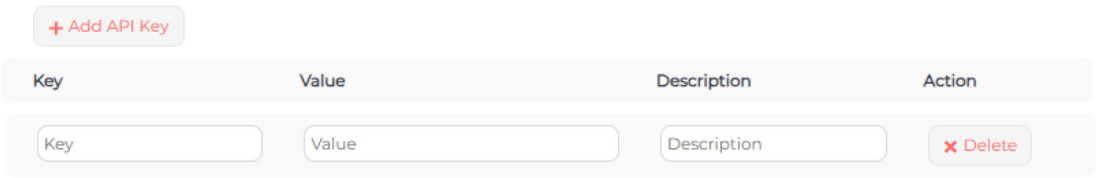
Before creating and testing requests, we need to set the security at the correct level in this folder hierarchy. Specifying this at the restdb.io data source is not correct because we may have other databases with different security keys. Because each restdb.io database has its own security key, then the place to specify security is the Football folder.

Click on the Football folder and click inside the Authorisation tab:



Set the Authorisation to be "API Keys", and Add To to "Header". This is because restdb.io expects an API key as a header.

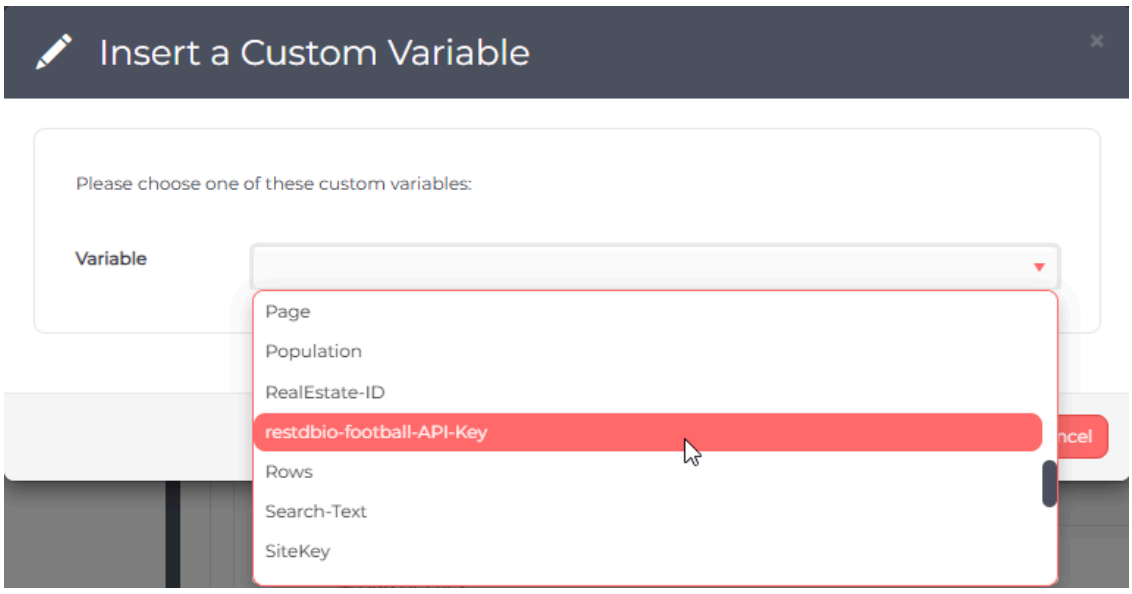
Now click the Add API Key button. This will create a blank row beneath:



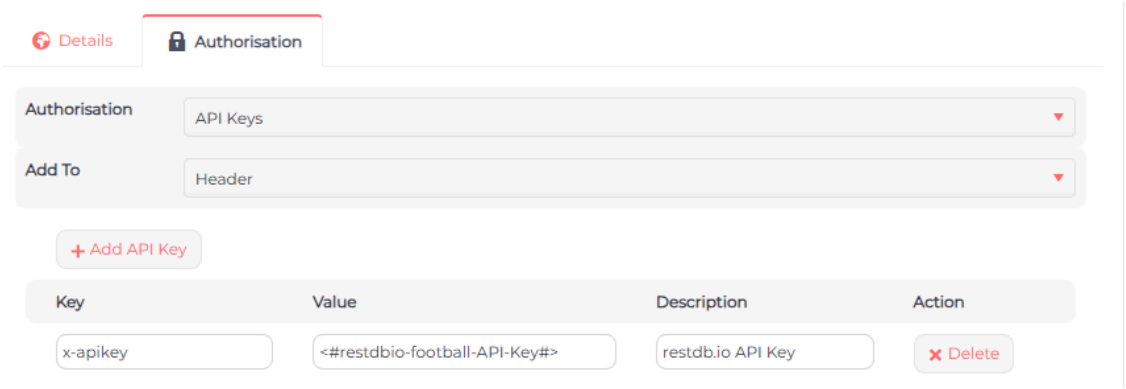
We know from this restdb.io documentation ↗ that the API Key is a header called x-apikey so we type this into the Key text box.

We have already created a centralised custom variable which lives securely on the server, so we will use that as the value. This is done by clicking inside the Value text box then clicking the Insert Variable button:

This will popup the Insert a Custom Variable popup form where we will select the **restdbio-football-API-Key** variable we created earlier:



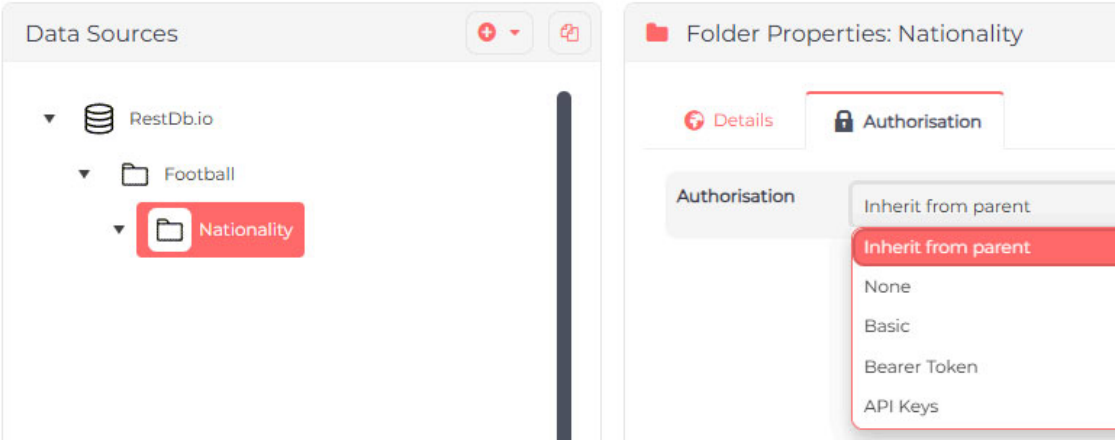
Then we click the **Select** button:



The custom variable is inserted into the Value. We should type something into Description to add clarity if necessary.

Nationality Folder

Now that we have set the API security at the database level, we need to ensure that all requests inherit the same security. This is done by selecting the Nationality folder beneath, clicking into the Authorisation tab and setting the Authorisation drop down combo to "Inherit from parent":



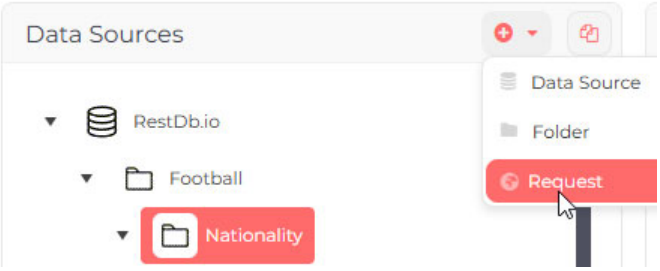
Every data source request we create in this folder will now inherit the security keys specified at the database level.

Create a Data Source Request

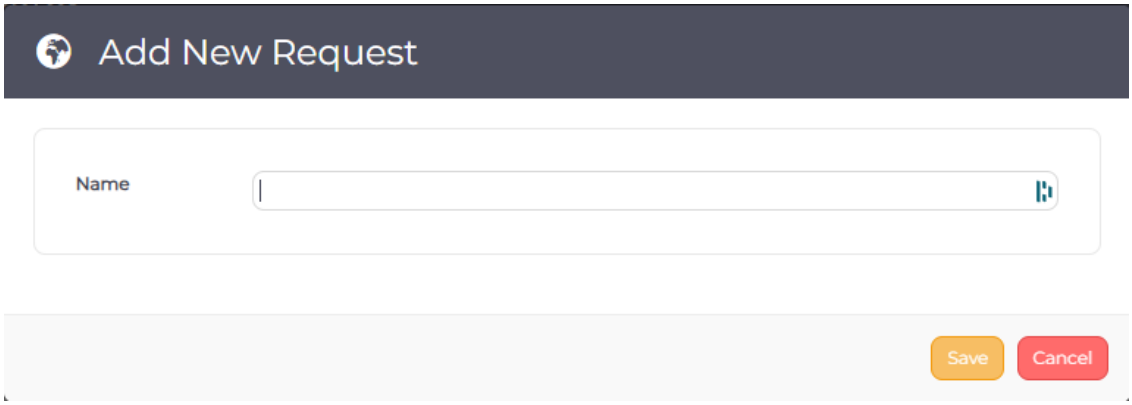
We can now create a data source request to read a list of nationalities from the ReST API.

Add Request

Click the add button menu item Request:



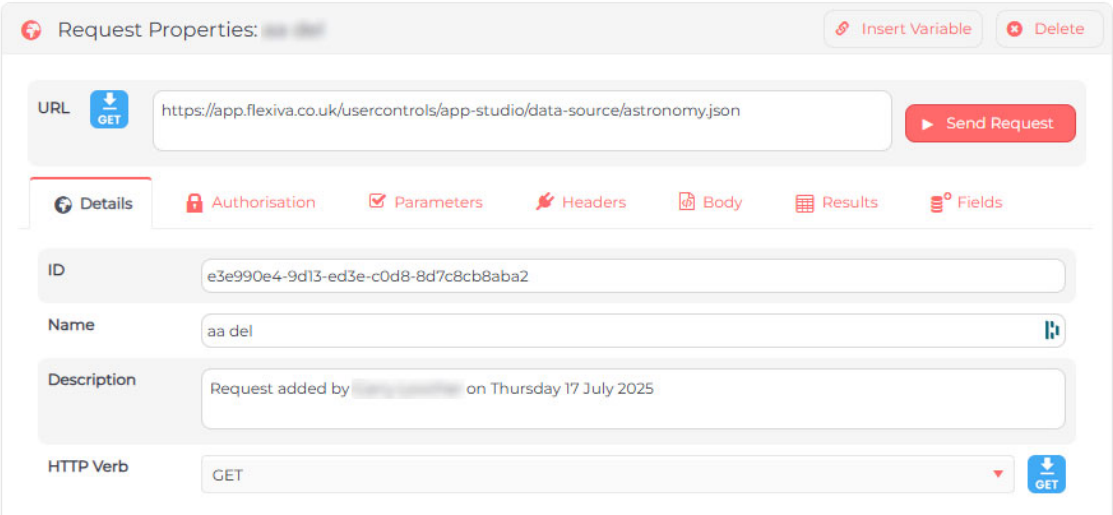
This will open up the Add New Request popup form:



Give this a name that describes exactly what it does. Something like "Read a list of Nationalities" is descriptive and unambiguous in the context of the location of the request in the tree hierarchy. Click the **Save** button to add and select this new request.

Edit Request Properties

The new request properties will look like this:



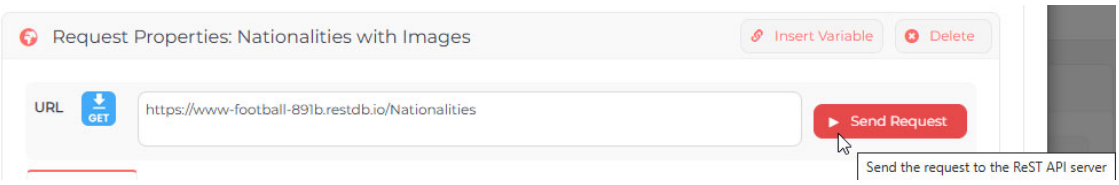
This will provide a default sample URL endpoint, a unique ID, the name you typed, a default description, and a HTTP Verb (method) defaulting to GET.

Paste in the correct URL from the list of restdb.io ReST API views previously tested in your browser, or using Postman

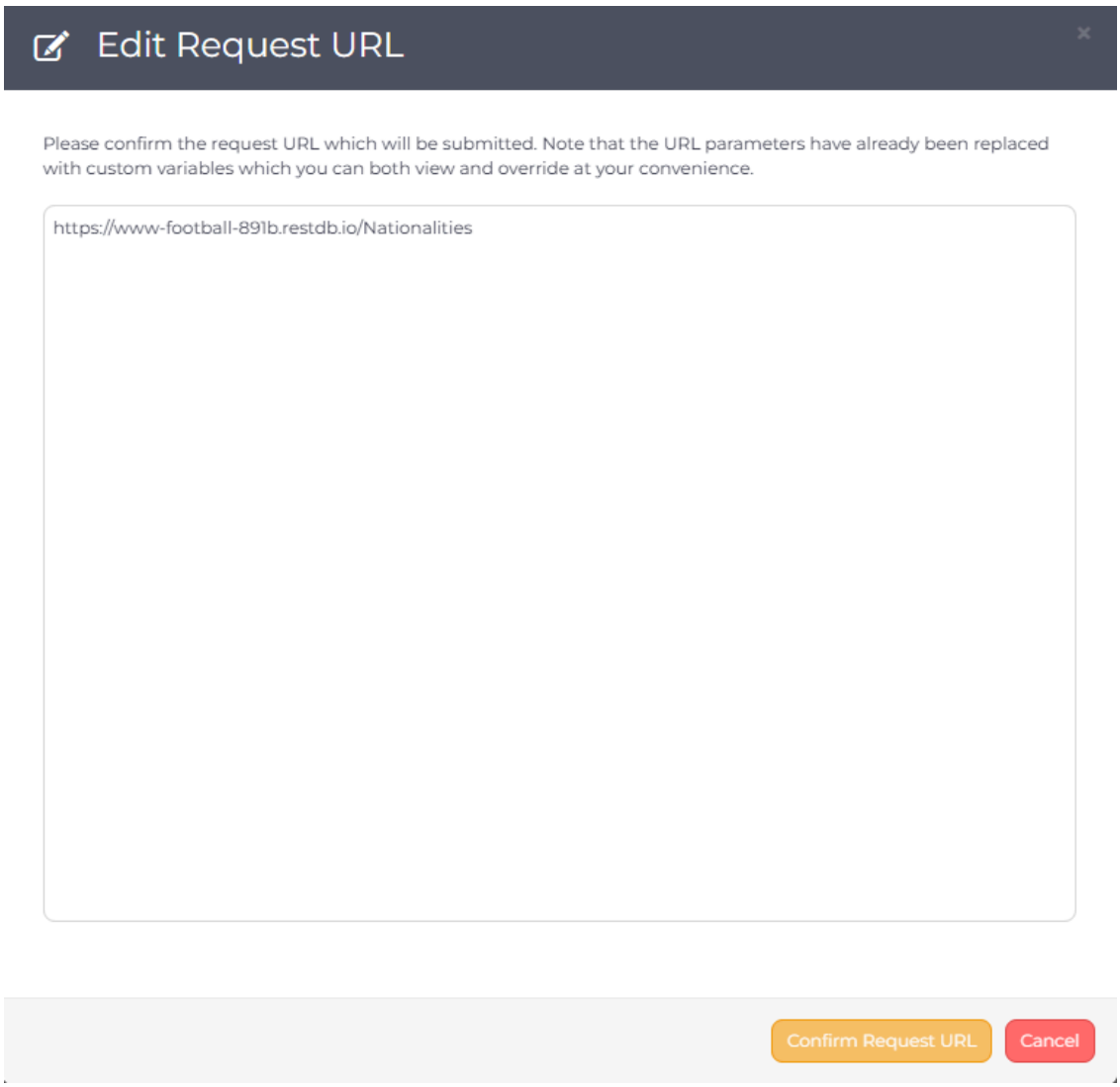
`https://www-football-891b.restdb.io/Nationalities`

Send Request

You should now test the request by clicking the Send Request button:

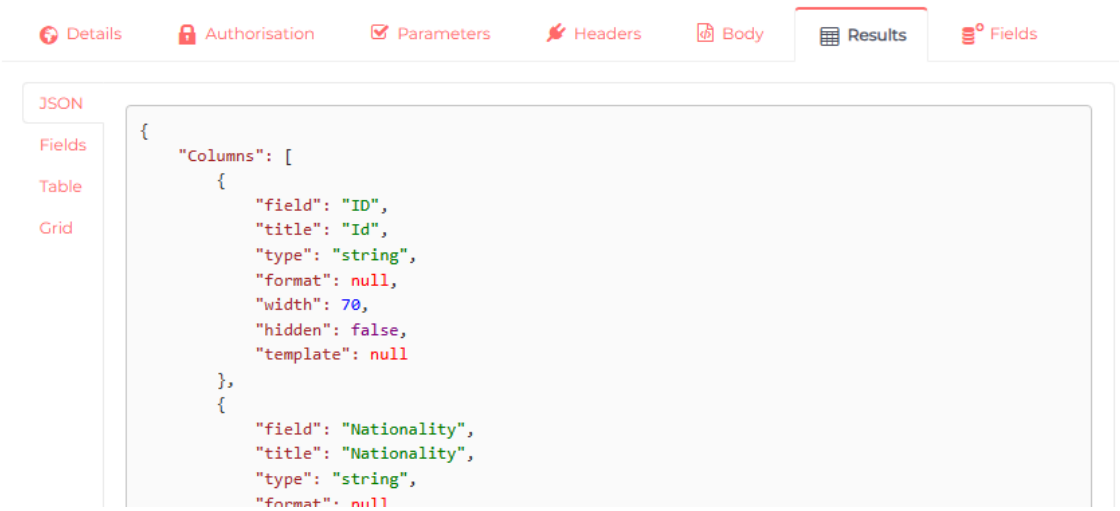


You will be prompted to confirm the URL being requested. Later, when developing filters, you will be able to edit this URL, but for now simply click the **Confirm Request URL** button:



Results

After the ReST API request has been sent, the Results tab will be selected and the JSON tab will show the first columns:



Scroll down until you see the Data Table:

Request Properties: Nationalities with Images

Insert Variable | Delete


```
"DataTable": {
    "List": [
        {
            "ID": "686e910978badf65001445dc",
            "Nationality": "Algerian",
            "Country": "Algeria",
            "Flag": [
                "686e910878badf65001445db"
            ],
            "FlagUrl": "https://football-891b.restdb.io/media/686e910878badf65001445db"
        },
        {
            "ID": "681de29778badf65000babea",
            "Nationality": "American",
            "Country": "United States of America [USA]",
            "Flag": [
                "681de29778badf65000babe9"
            ],
            "FlagUrl": "https://football-891b.restdb.io/media/681de29778badf65000babe9"
        },
        {
            "ID": "6863e1d478badf6500137cc2",
            "Nationality": "Argentine",
            "Country": "Argentina",
```


We can see that the actual data from the ReST API is being returned, including the nation flags as URL images. This is because our restdb.io view is able to map its media files onto a two-dimensional table for easy consumption by client-side applications.

Fields

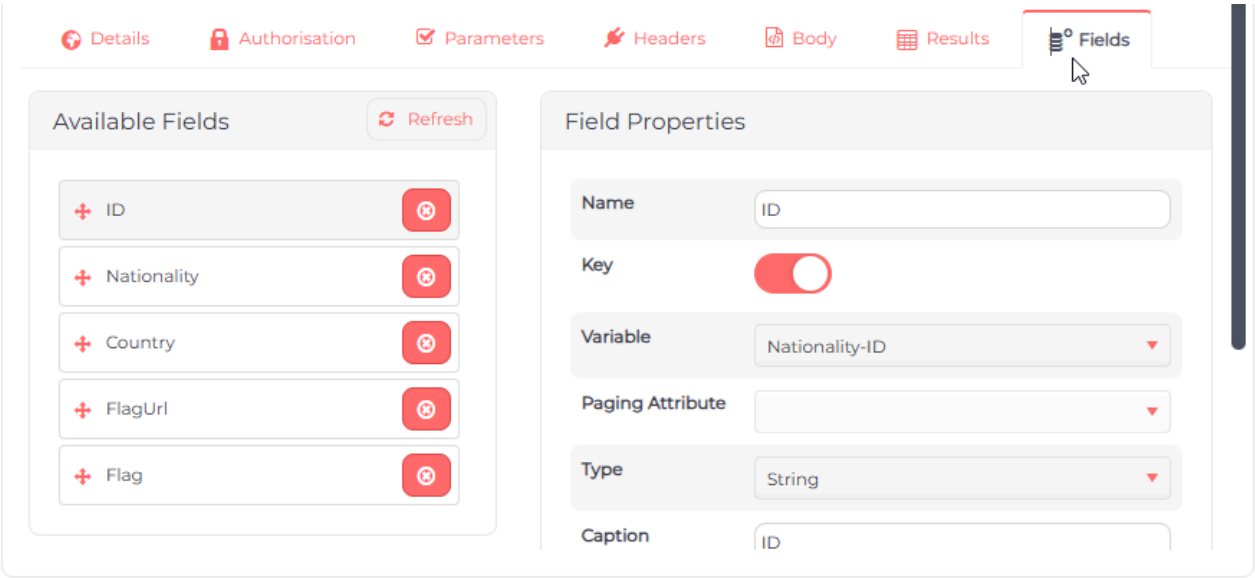
If we scroll back up to the top and click the left Fields tab, we should see a list of the fields accompanying the data:

Details | Authorisation | Parameters | Headers | Body | Results | Fields

JSON
Fields
Table
Grid

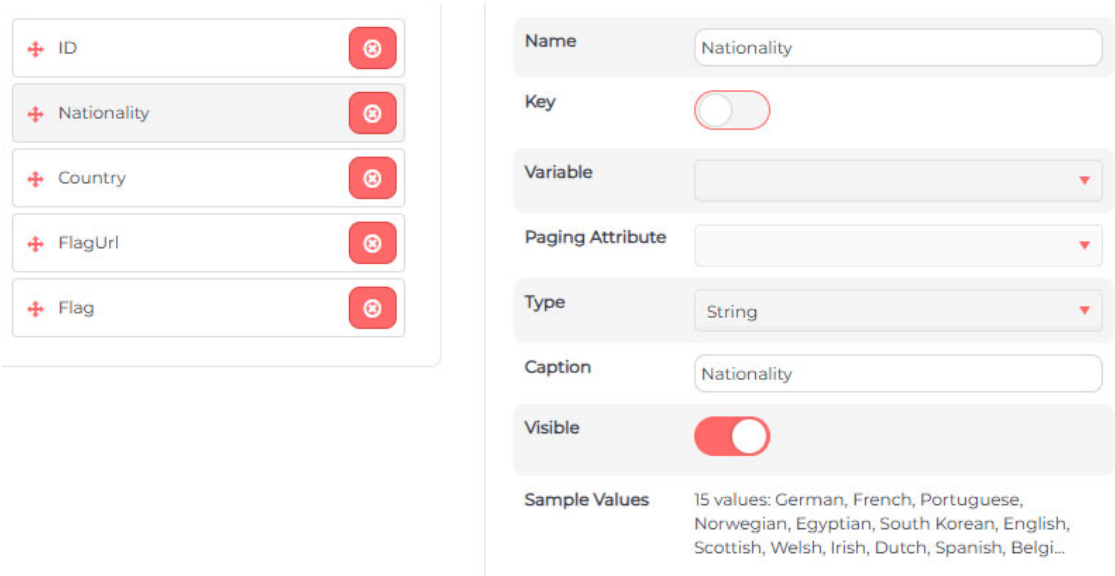
Field	Type	Values	Visible	Caption
Country	String	1 values: Algeria	true	Country
Flag	Array	1 values: 686e910878badf65001445db	true	Flag
FlagUrl	String	1 values: https://football-891b.restdb.io/media/686e910878badf65001445db	true	FlagUrl
ID	String	1 values: 686e910978badf65001445dc	true	ID
Nationality	String	1 values: Algerian	true	Nationality

Clicking on the top right Fields tab will show these Available Fields in a vertical list to the left:



Fields can be re-ordered using the left drag icon, or removed using the right delete button.

Selecting any field will show its corresponding properties to the right. Selecting the Nationality field and scrolling down should reveal the sample values:

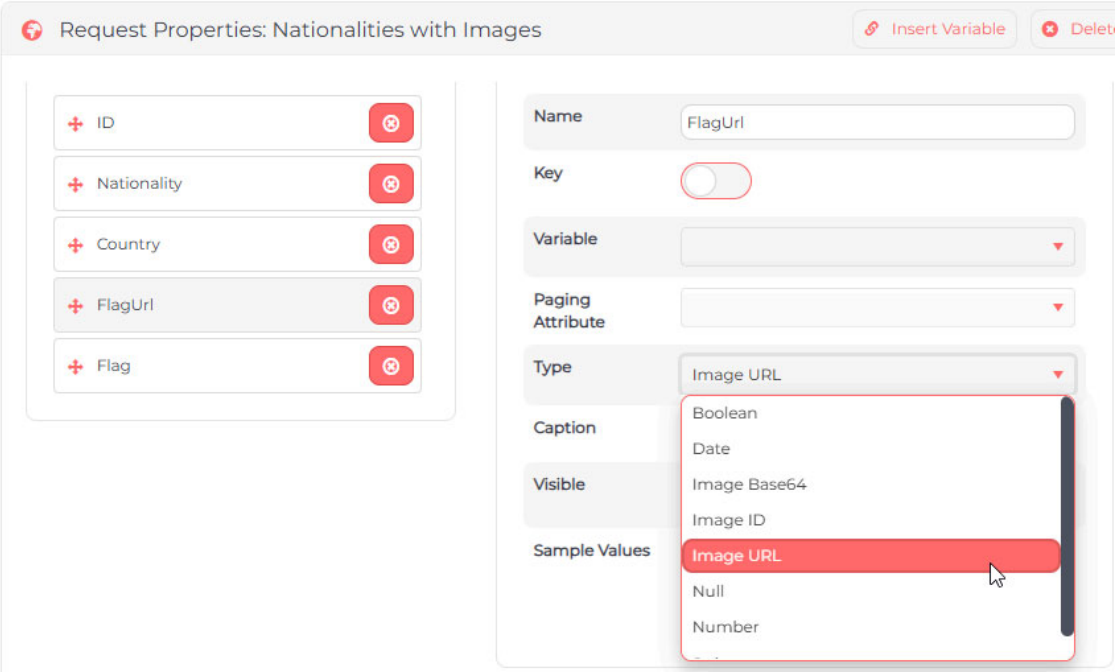


These are useful to confirm that fields are returning the expected data.

Image URL

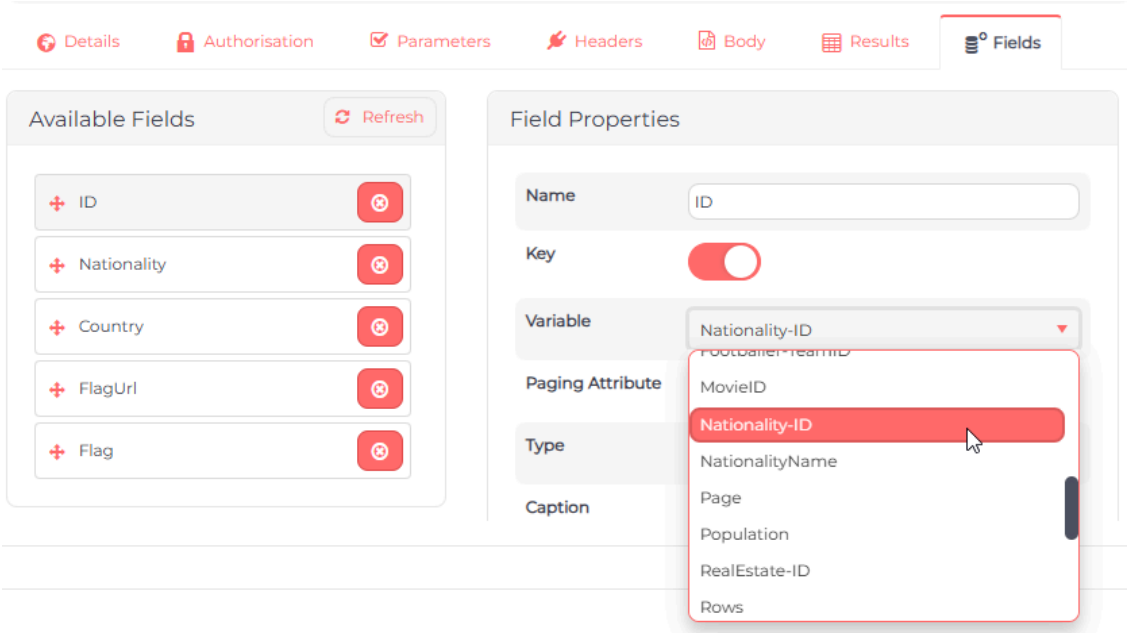
When returning images, you should configure the type of field so that it displays correctly in form components such as grids.

Select the FlagUrl field and set it's Type to "Image URL":



Key Field for Drill Down

Another important field to configure is the unique identifier or "Key" field. This is usually a long random unique string or number. We need to set the ID field to be the Key as well as assign it to the custom variable we created earlier:



This facilitates a process known as 'drill-down' where the end-user selected a hyperlinked column in a row and drills down into it by opening up the data entry form.

Note that we can now simply close the the data source configurator form as all changes are automatically persisted so we can move onto creating the lookup form.

Nationalities Lookup Form

Now that we have the custom variables and data source request to facilitate listing nationalities, we need to create a lookup form upon which to display a grid showing all nationalities.

Create Form

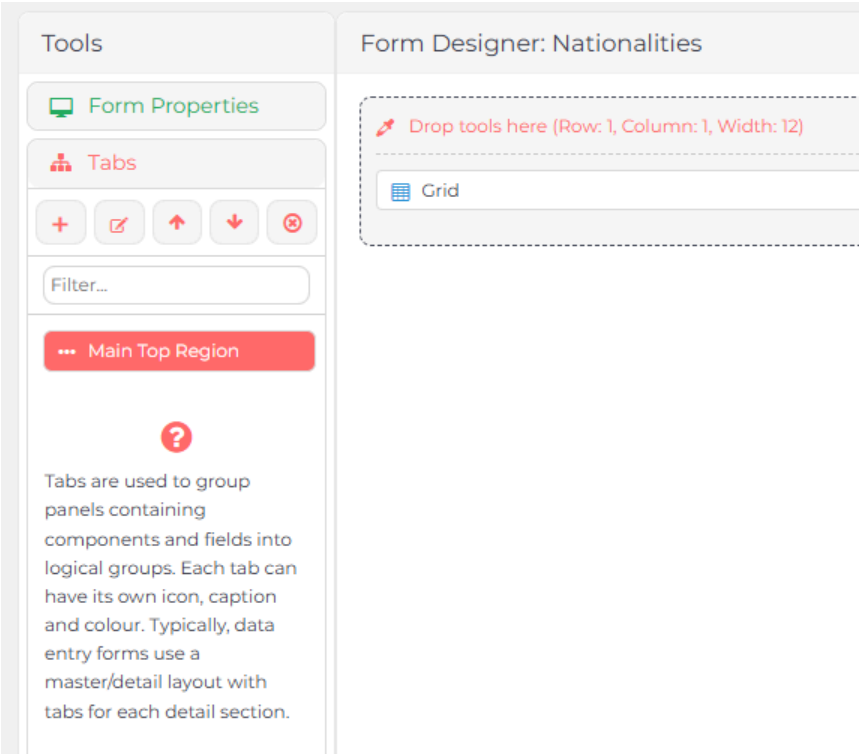
Open App Studio then select the Forms configurator.

Add

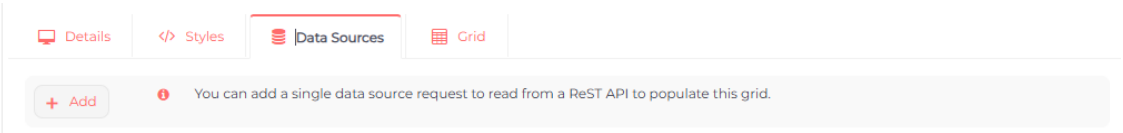
Use the Add button to create a new form called "Nationalities" or "Nationality Lookup". In the form properties, make sure that this form is of type "Lookup". You do not need to assign a data source to the form as we will do this in the form designer when we add a Grid component.

Design

Open form designer from the form properties popup, and drag a grid component into the first panel on the Main Top Region tab so that it looks like this:

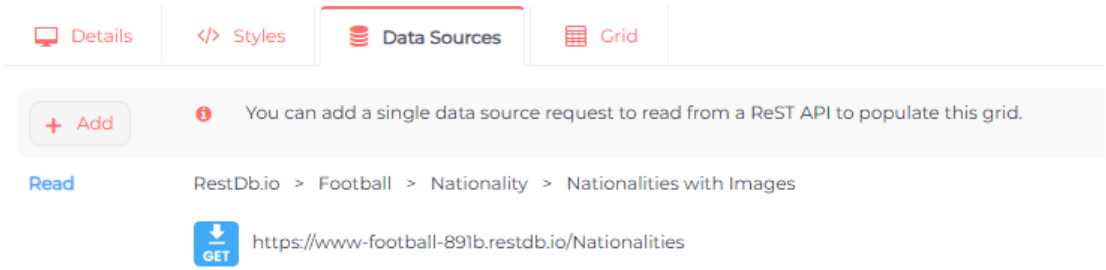


Click on the Grid to open the component properties for the grid. Select the Data Sources tab:



Use the **Add** button to select the data source request you created above.

After selection, the Read data source request should appear like this:



Grid Configuration

Select the Grid tab to configure the grid to display the data.

Population

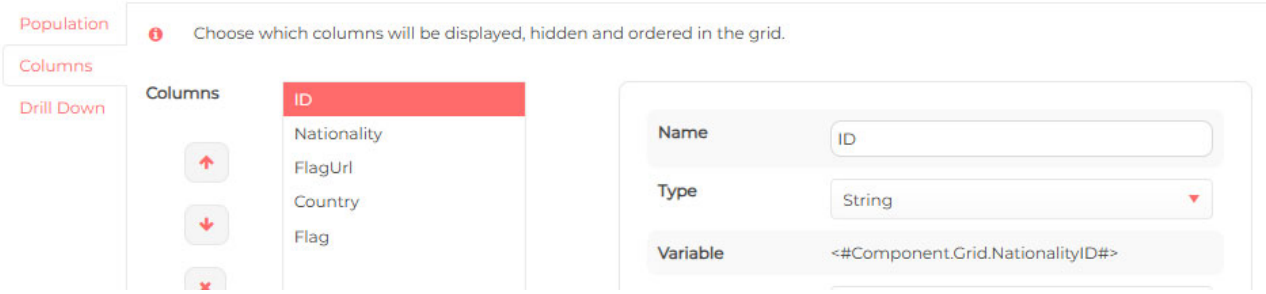
In the Population left tab, set the Population Trigger to Form Loaded. This tells the grid to load the data as soon as the form is displayed.

Columns

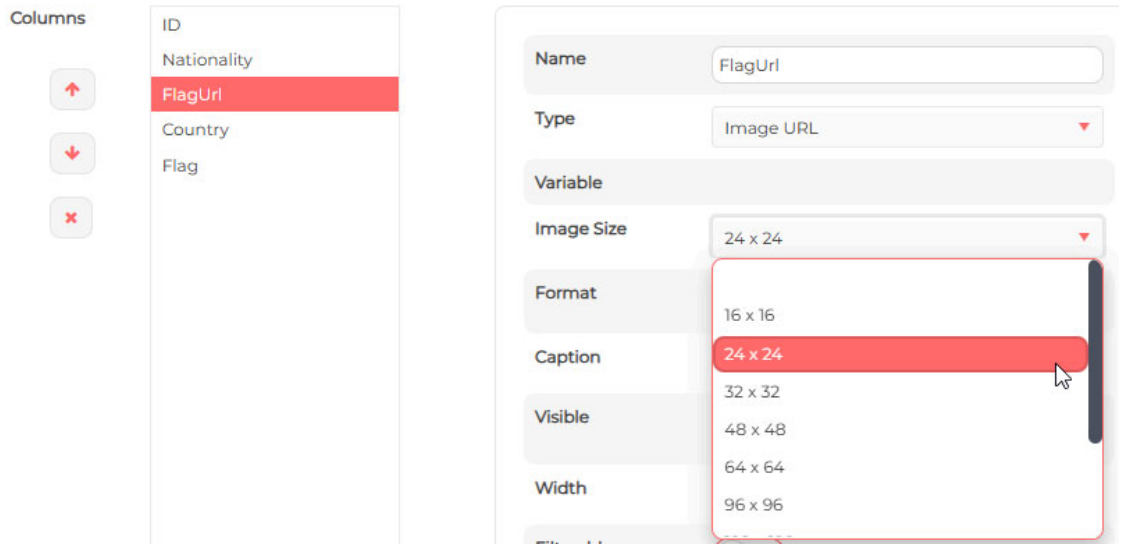
In the Columns left tab, the fields previously retrieved from the ReST API are displayed in the order you designed. You can re-order these by using the up/down arrows or delete a column.

Note that in the world of data, the term field is used, yet in grids, the word column is used. This is why the grid uses the column nomenclature instead of field.

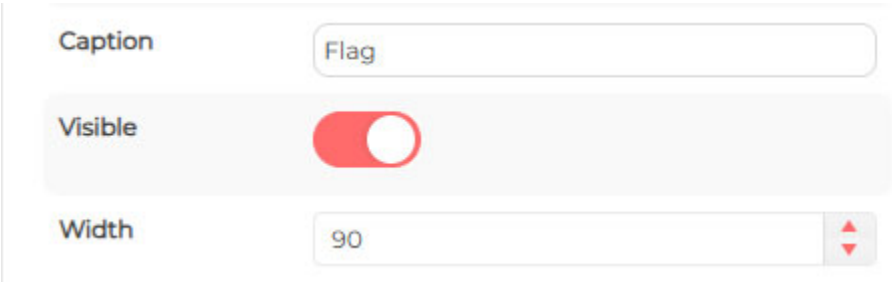
Note that the ID key column is showing the custom variable you assigned to it previously.



For the FlagUrl column, this should remember that it was configured as a Image URL Type. Set the Image Size to be 24 × 24 so that it looks the correct size when displayed in the grid:



Also set the Caption, Visibility and Width of the FlagUrl column:



Hide the Flag and ID columns as these need not be shown to the end-user.

Drill Down

We will not configure the drill down tab until after we have created the data entry form in the next section.

Apply

Apply the grid properties, then Save the form design to persist the form for later use.

Navigation Bar

We can now add this lookup form to the navigation bar from where it can be opened by the end-user.

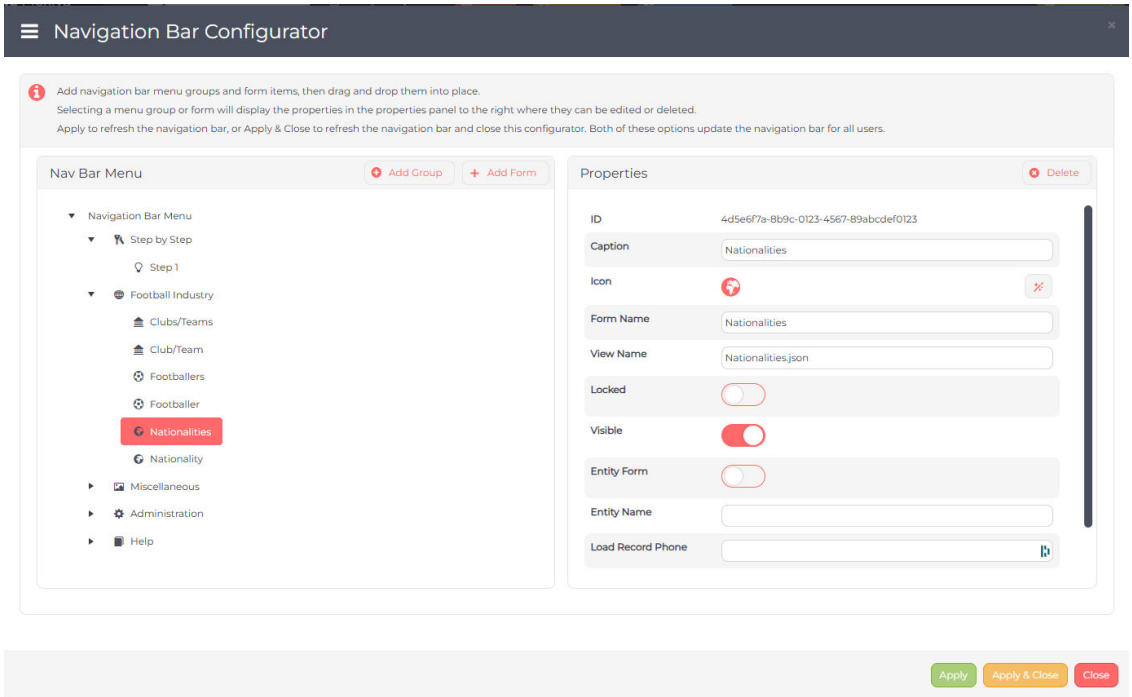
Open App Studio then select the Navigation Bar configurator.

Football Industry

Create a group called "Football Industry" using the Add Group button.

Nationalities

When the Football Industry group is selected, add a form using the **Add Form** button. This will popup a modal dialogue where you can select your newly created Nationalities form. Clicking Save will show the new Nationalities form menu:

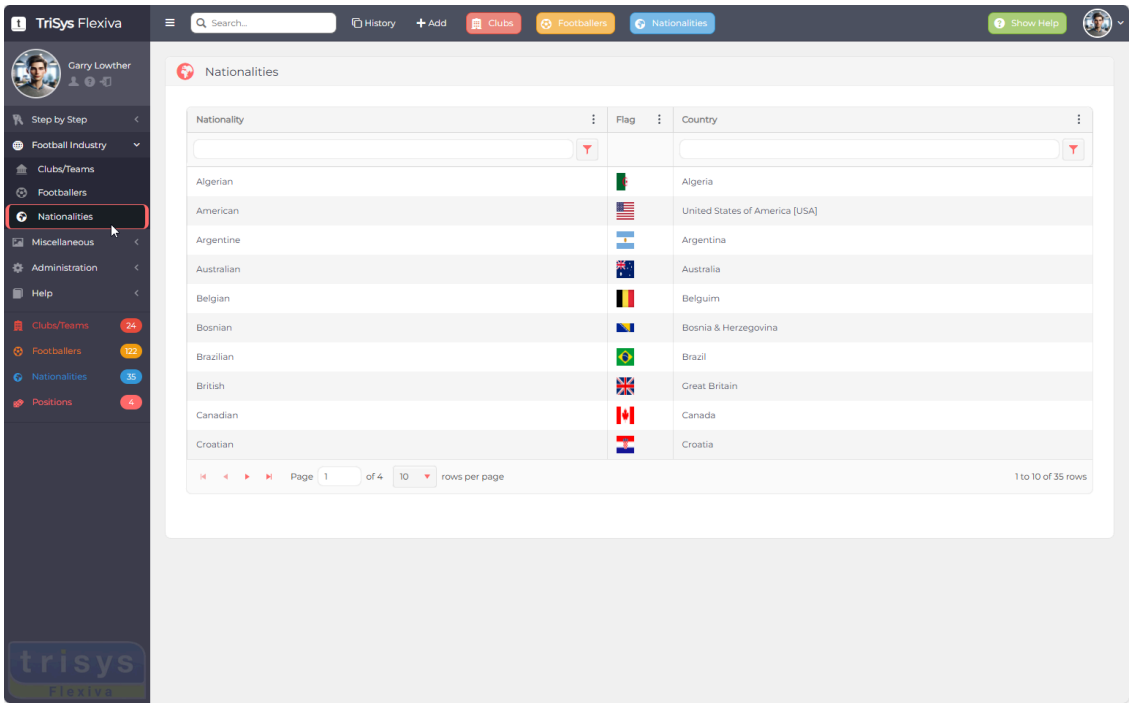


Select Visible, and change the Icon to a globe. Then press the **Apply & Close** button.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Nationalities item.

You should now see the lookup form with the data from the ReST API displayed in a grid.



Test that column filtering and sorting works as expected.

We are now ready to move to the next step to create a data entry form configured for creating, reading, updating and deleting nationality records.

Nationality Data Entry Form

Create a data entry form for the nationality record.

Having previously created custom variables and a data source request to display a lookup form showing, filtering and sorting nationalities, we are now moving on to create a data entry form where a nationality can be created, read, updated and deleted (CRUD).

In our ReST API sample data set, the underlying restdb.io nationality is the `Nationality` table.

The process for all CRUD operations typically starts with READ, as this involves designing the data entry form, and drilling down into it. Here are the four CRUD phases in the order we will configure them:

READ

- Add a 'read' data source for reading a single nationality record
- Link the key nationality record identifier to this ReST API end-point if necessary
- Create a data entry form
- Assign this 'read' data source to the data entry form
- Design this form and drag fields from this data source
- Add this form to the navigation bar
- Configure drill down to the lookup form grid component
- Test that we can lookup nationalities and drill down into a data entry form showing the nationality master record
- Set the History Menu record summary
- Test that we can lookup nationalities and drill down into the nationality form, and that the history menu shows the nation
- Add a master/detail grid to show all footballers who represent this nation
- Test that we can view all footballers who belong to this nation

UPDATE

- Create custom variables for each field
- Add an 'update' data source for updating a single nationality record
- Link the key nationality record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'update' data source to the data entry form
- Configure the update button
- Test that we can update a nationality using the Update button

CREATE

- Add a 'create' data source for creating a single nationality record
- Link the key nationality record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'create' data source to the data entry form
- Configure the app toolbar Add menu to add this data entry form
- Test that we can create a nationality using the Add menu and Update button

DELETE

- Add a 'delete' data source for deleting a single nationality record
- Link the key nationality record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'delete' data source to the data entry form
- Configure the delete button
- Test that we can delete a nationality using the Delete button

Nationality Form: Read

Add a new nationality form to read and display a record.

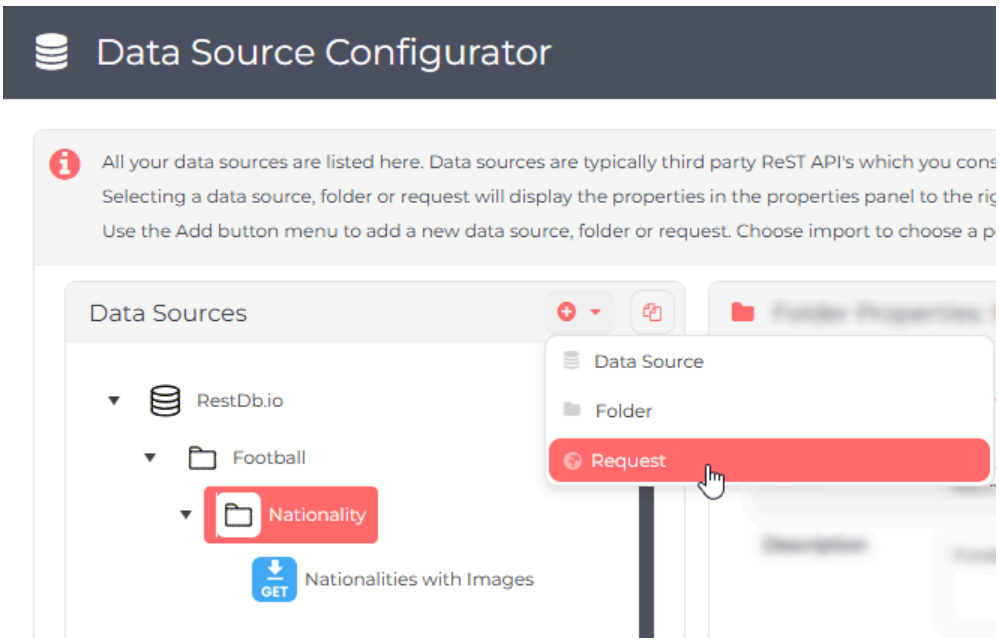
This is the process we will follow.

READ

- Add a 'read' data source for reading a single nationality record
- Link the key nationality record identifier to this ReST API end-point if necessary
- Create a data entry form
- Assign this 'read' data source to the data entry form
- Design this form and drag fields from this data source
- Add this form to the navigation bar
- Configure drill down to the lookup form grid component
- Test that we can lookup nationalities and drill down into a data entry form showing the nationality master record
- Set the History Menu record summary
- Test that we can lookup nationalities and drill down into the nationality form, and that the history menu shows the nation
- Add a master/detail grid to show all footballers who represent this nation
- Test that we can view all footballers who belong to this nation

Add a READ Data Source

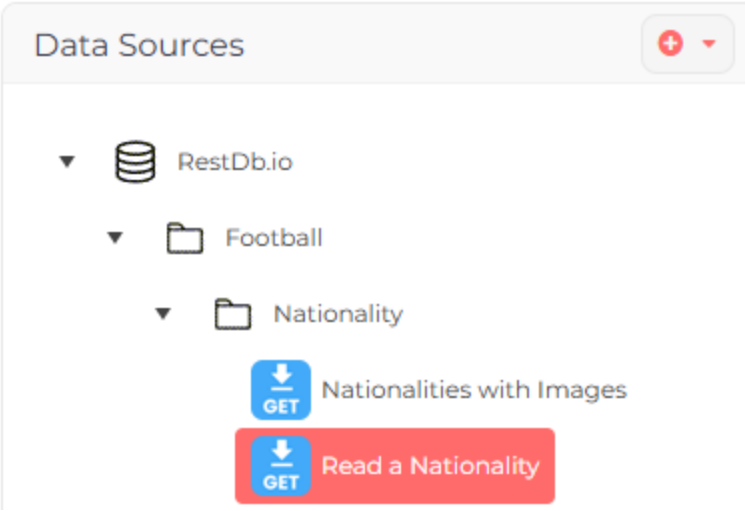
Open App Studio, open the Data Sources configurator, select the Nationality folder you created previously, then use the add button menu to create a new request:



The Add New Request modal popup form shows asking you to name the request:

Type a meaningful name such as "Read a Nationality" and click Save.

The request will be added to your tree view beneath the Nationality folder:



Edit Properties

Edit the properties in the Details tab as following by referencing this list of ReST API end-points.

URL

`https://restdb.trisys.co.uk/ReadNationality`

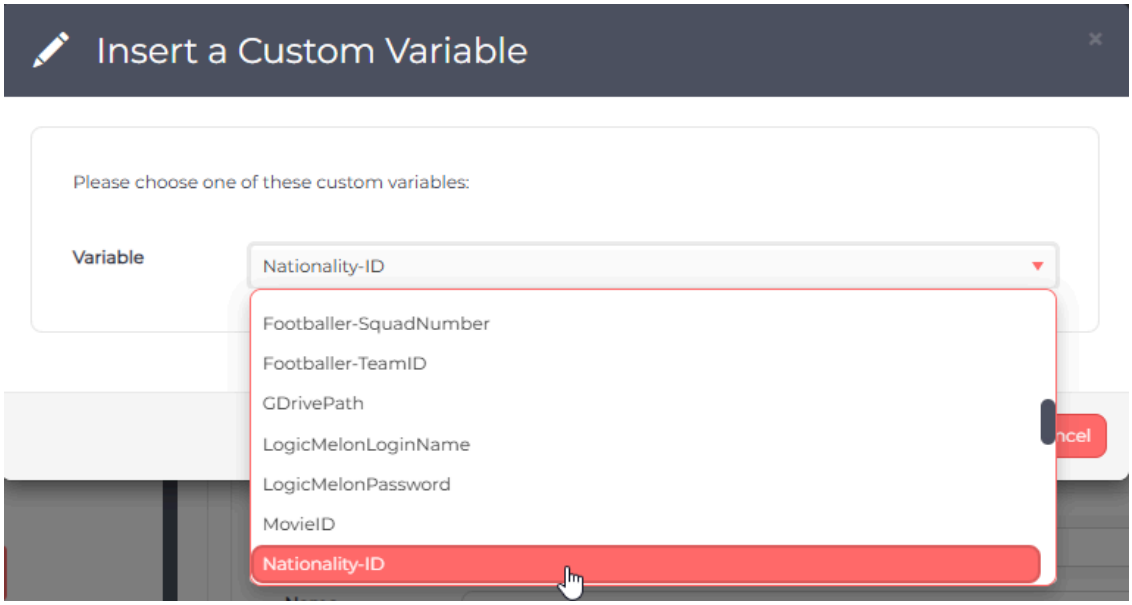
Now it is known from the documentation that this ReST API end-point has a nationality identifier property ?ID=.

We therefore type this parameter list into the URL so that it reads:

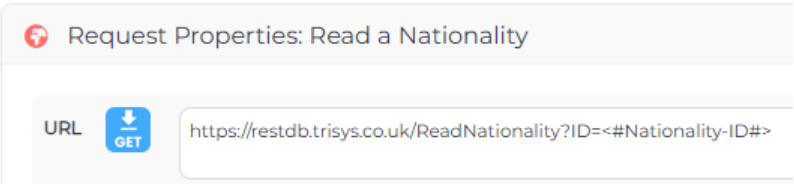
`https://restdb.trisys.co.uk/ReadNationality?ID=`

Whilst the caret is still blinking after the last character typed, click on the Insert Variable button:

This will open the following modal popup for where you should select the Nationality-ID custom variable you created previously in the Variable drop down list . Click the Select button.



The URL should now show the custom variable appended to the end:



HTTP Verb

The default GET verb/method is usually correct for reading a single record from a ReST API and is correct for this specific back-end end-point.

Authorisation

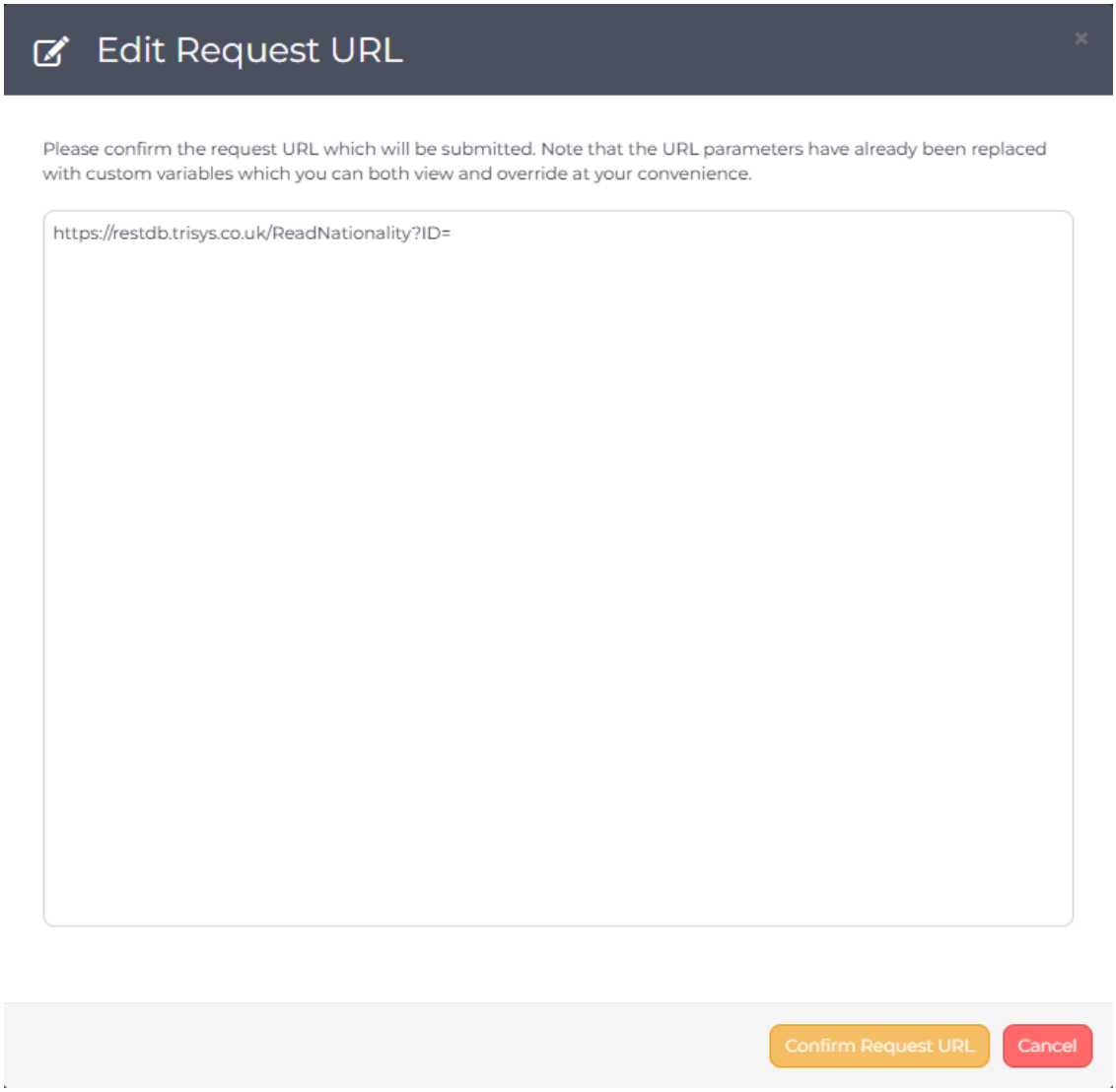
Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Send Request

Click this button on the Details tab to send the request to the ReST API.

You will be prompted to confirm the URL:



The value of the custom variable `Nationality-ID` should show after ?ID= but if it does not, then copy this value in: `67f7cb8278badf650004fb6b` as this is the ID for English.

This URL should now be:

`https://restdb.trisys.co.uk/ReadNationality?ID=67f7cb8278badf650004fb6b` ↗

Press the Confirm Request URL button.

The request should run quickly and select the Results tab and show the JSON top left tab displaying the full JSON returned from the ReST API:

```
{
    "Columns": [
        {
            "field": "ID",
            "title": "Id",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": null
        },
        {
            "field": "Nationality",
            "title": "Nationality",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "Country",
            "title": "Country",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "Flag",
            "title": "Flag",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "FlagUrl",
            "title": "Flagurl",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": "<img src='#: FlagUrl #' style='width: 64px;
height: 64px;' />"
        }
```

```
    ],
    "DataTable": {
        "List": [
            {
                "ID": "67f7cb8278badf650004fb6b",
                "Nationality": "English",
                "Country": "England",
                "Flag": "67fa336b78badf6500054bea",
                "FlagUrl": "https://football-
891b.restdb.io/media/67fa336b78badf6500054bea"
            }
        ],
        "DynamicColumns": null,
        "TotalRecordCount": 0,
        "TotalPageCount": 0,
        "FirstRowNumber": 0,
        "LastRowNumber": 0,
        "PageNumber": 1,
        "RecordsPerPage": 1,
        "SortColumnName": null,
        "SortAscending": true,
        "AICriteria": null,
        "Success": false,
        "ErrorMessage": null
    },
    "URL": "https://restdb.trisys.co.uk/ReadNationality?
ID=67f7cb8278badf650004fb6b",
    "Verb": "GET",
    "Success": true,
    "ErrorMessage": null
}
```


Look at the `DataTable -> List` to see that it contains 1 nationality record.

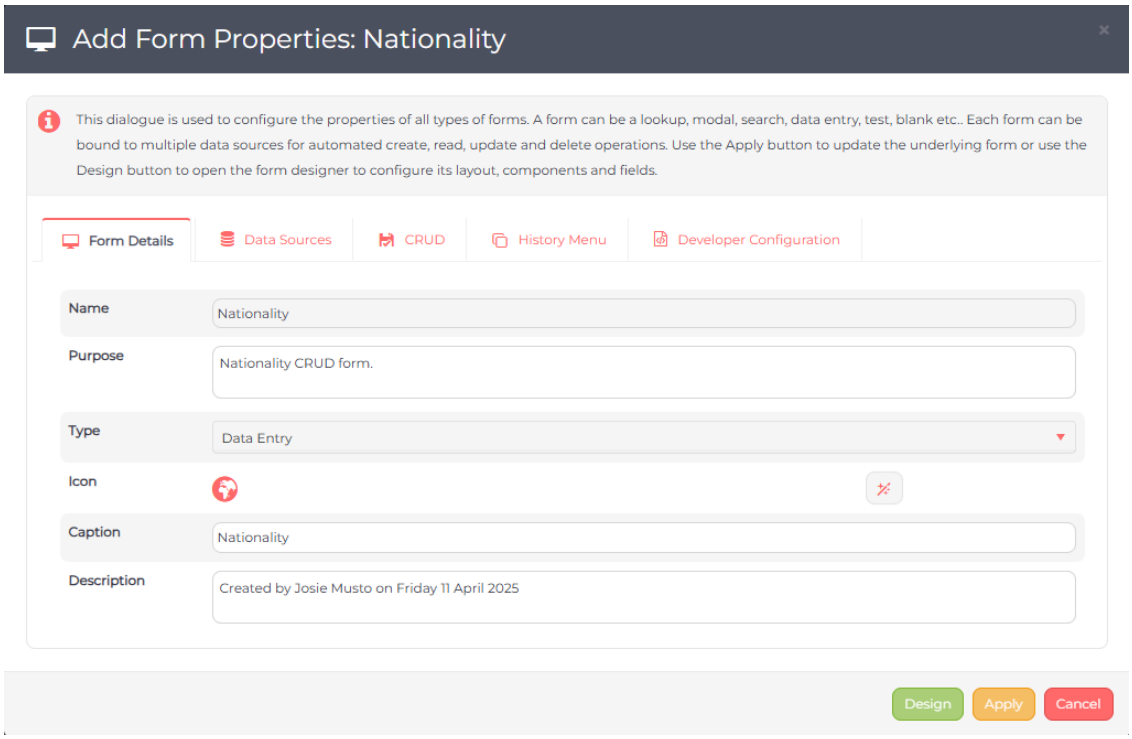
Our data source request has now been created and configured, so we can now close this configurator.

Create a Data Entry Form

Open App Studio, then open the Forms configurator.

Add

Click the Add button to open the modal popup form:



Name

Type the name "Nationality". You do not need the word "Form" to follow it, despite your initial instinct.

Purpose

Type "Nationality CRUD form."

Type

This must be set to "Data Entry".

Icon

Use the far right button to choose a suitable icon for the form. Use the text "glob" in the filter to find mother Earth.

Caption

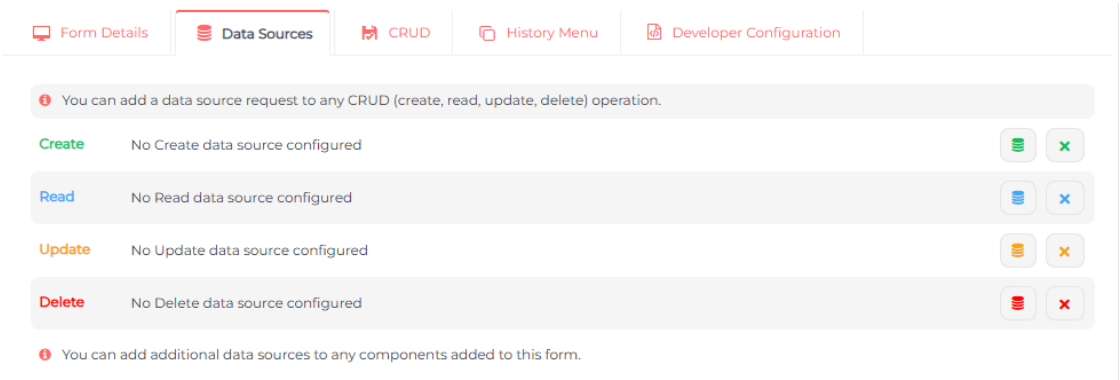
Set the caption to "Nationality". Note that we will configure the history menu to show the nationality, not this caption.

Description

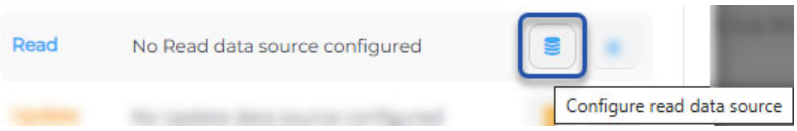
The description will be automatically generated which is fine for now.

Data Sources

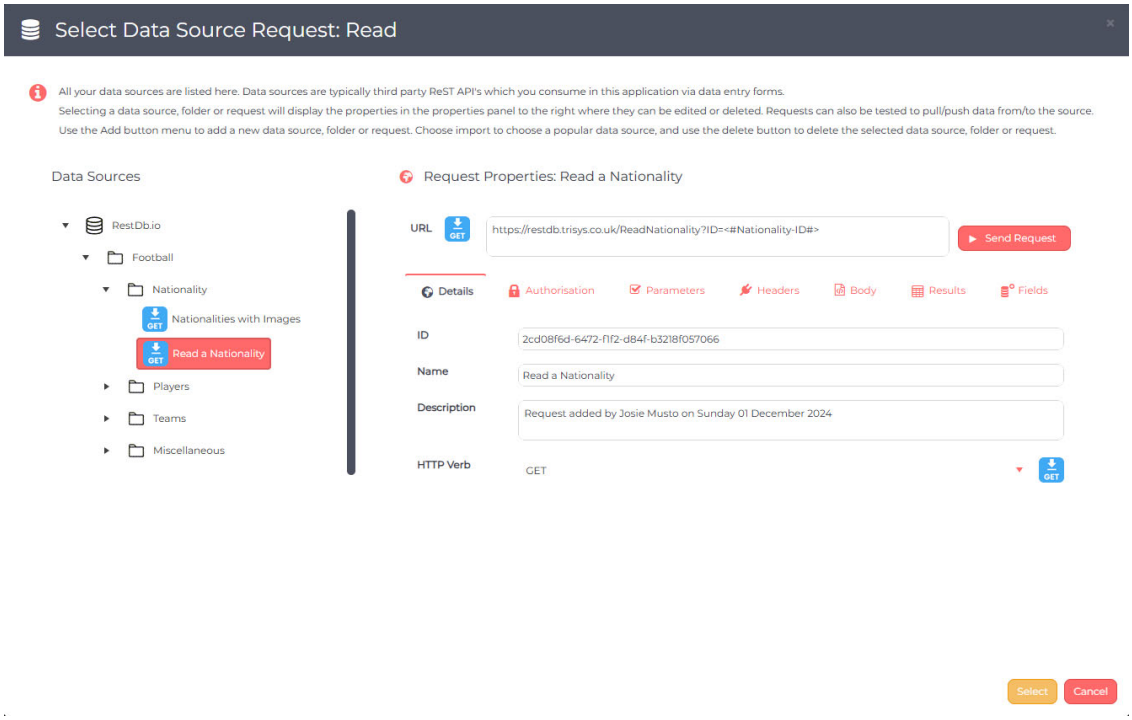
Click the Data Sources tab as this is where we will add the READ data source we created earlier:



There are 4 CRUD data source lines. Click this database icon for the Read data source:

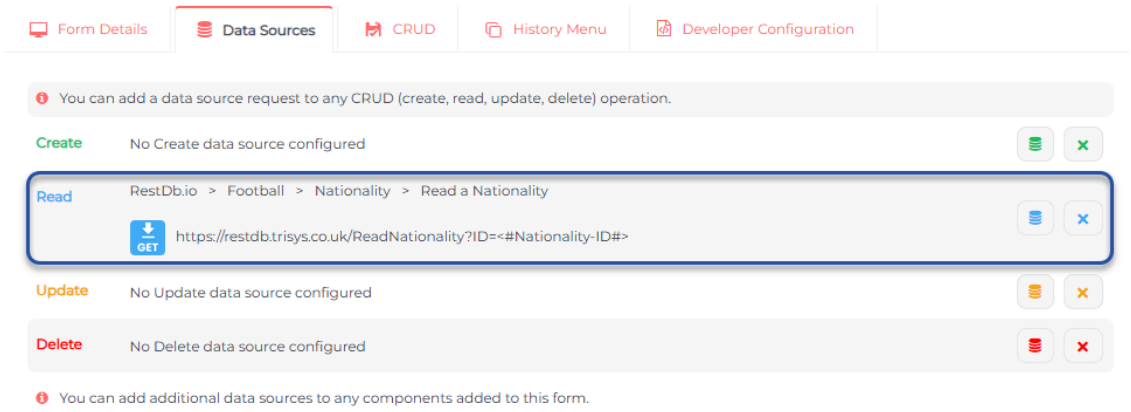


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request:

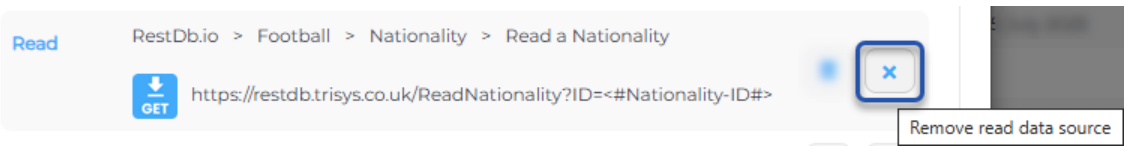


Click the **Select** button to choose this data source.

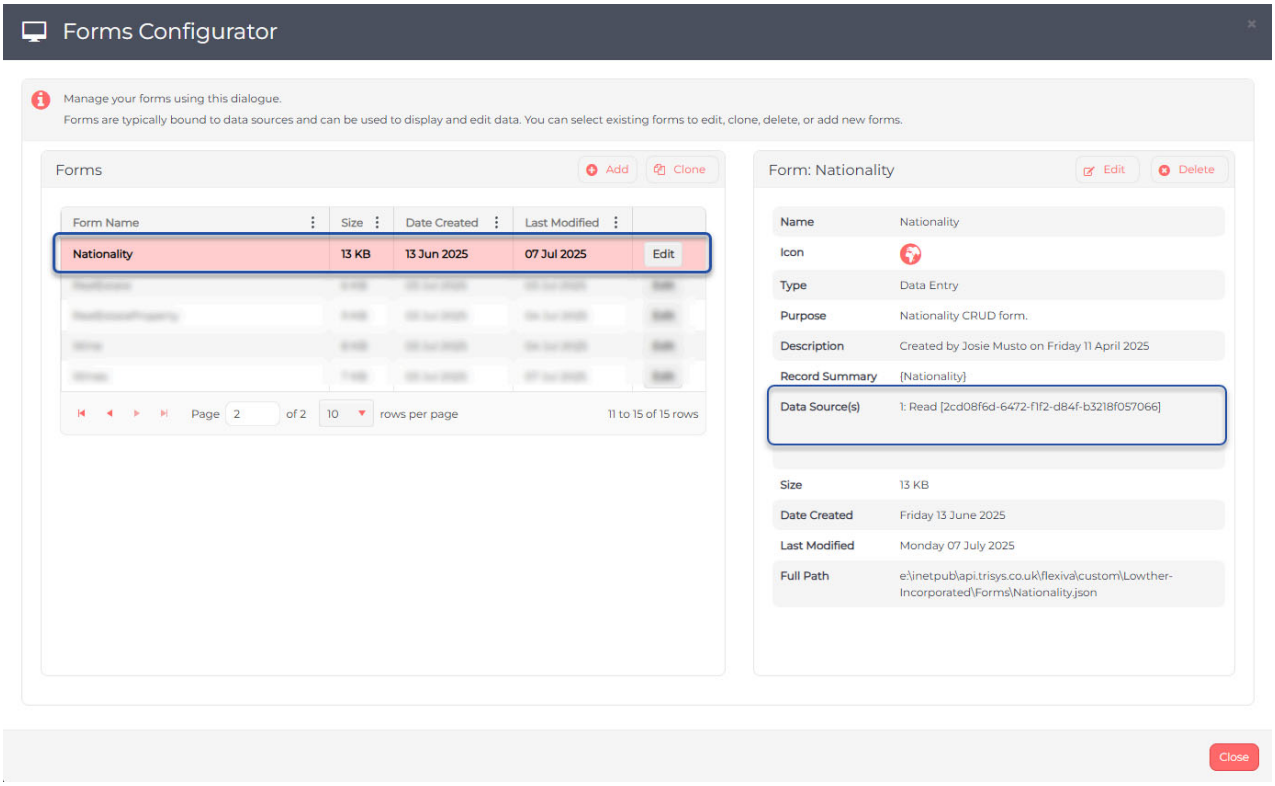
The popup will close and the selected Read data source request is now added to the list of data sources for this data entry form:



If you ever need to remove a data source request from the list, you can use this button:

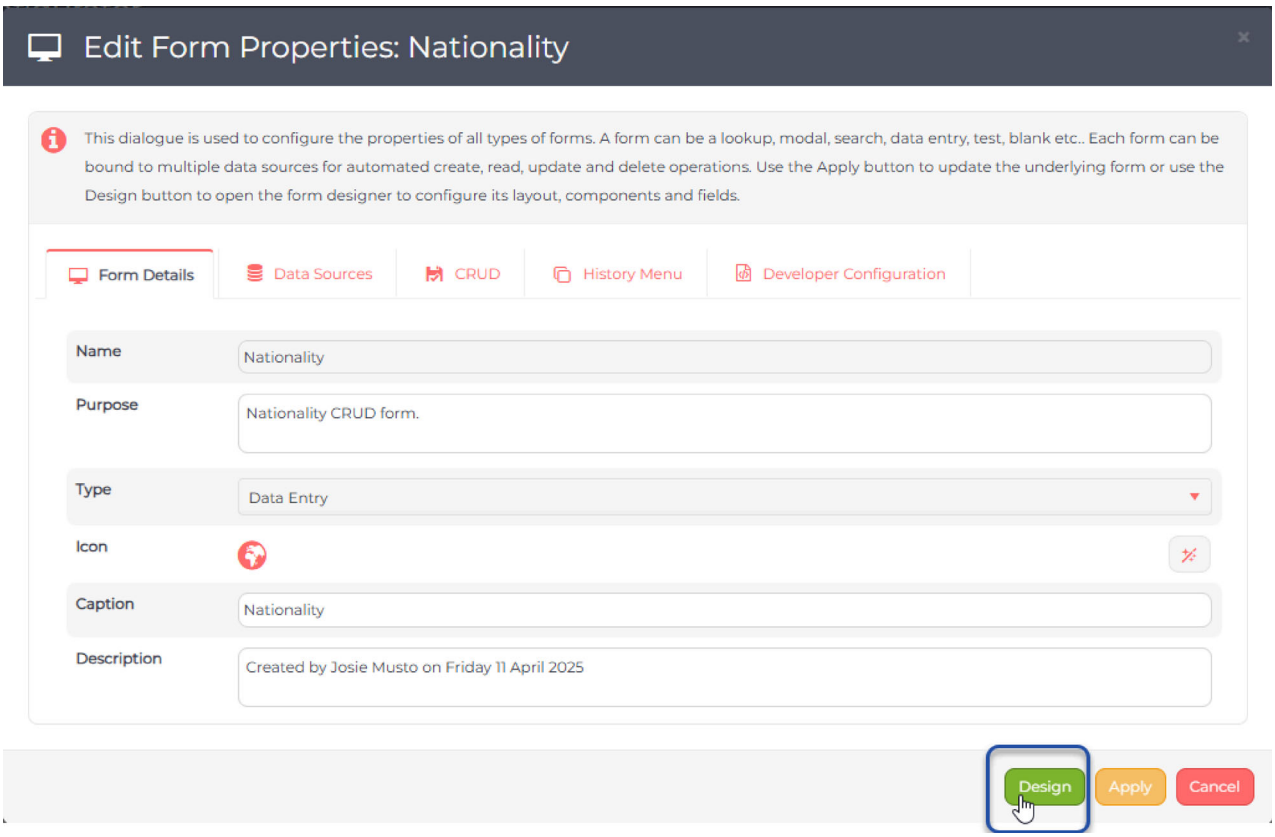


It is recommended that you now click the **Apply** button on this form, in order to persist the form properties before designing your form. Your new form should now appear in the list of forms, and the Read data source request you added should be shown in the properties list:

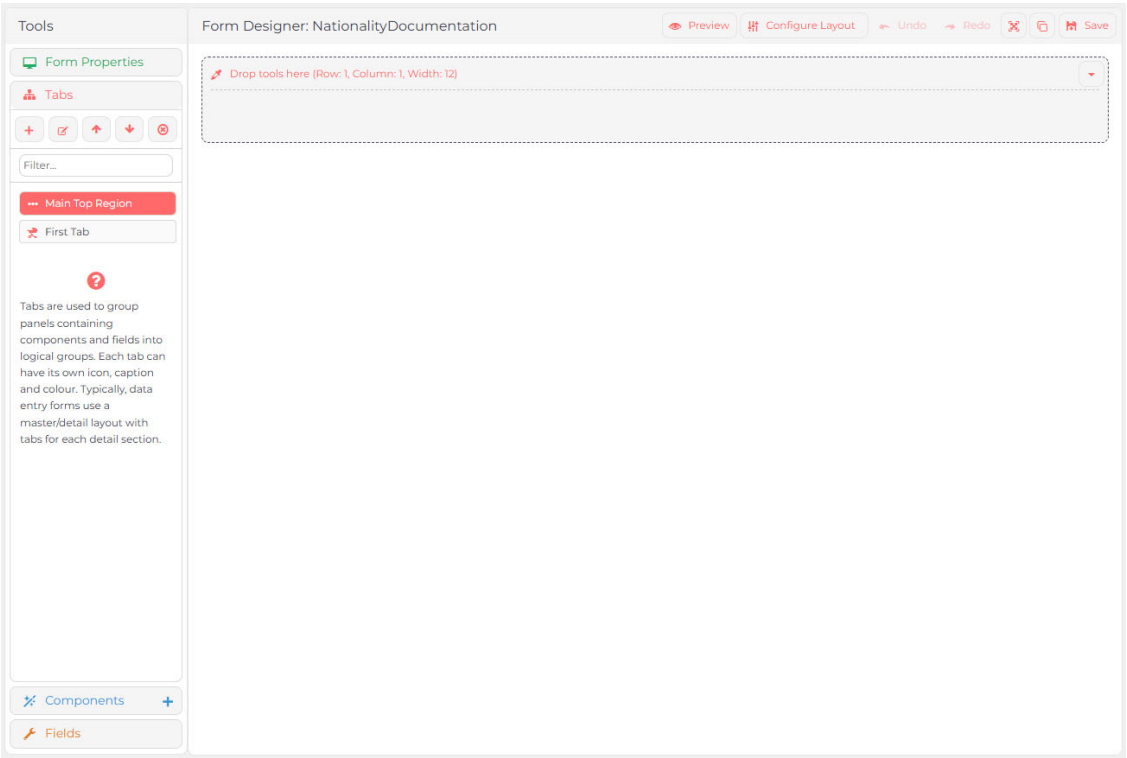


Form Design

Click one of the **Edit** buttons, either the grid row or properties panel. The form modal popup will display. Click the Design button:

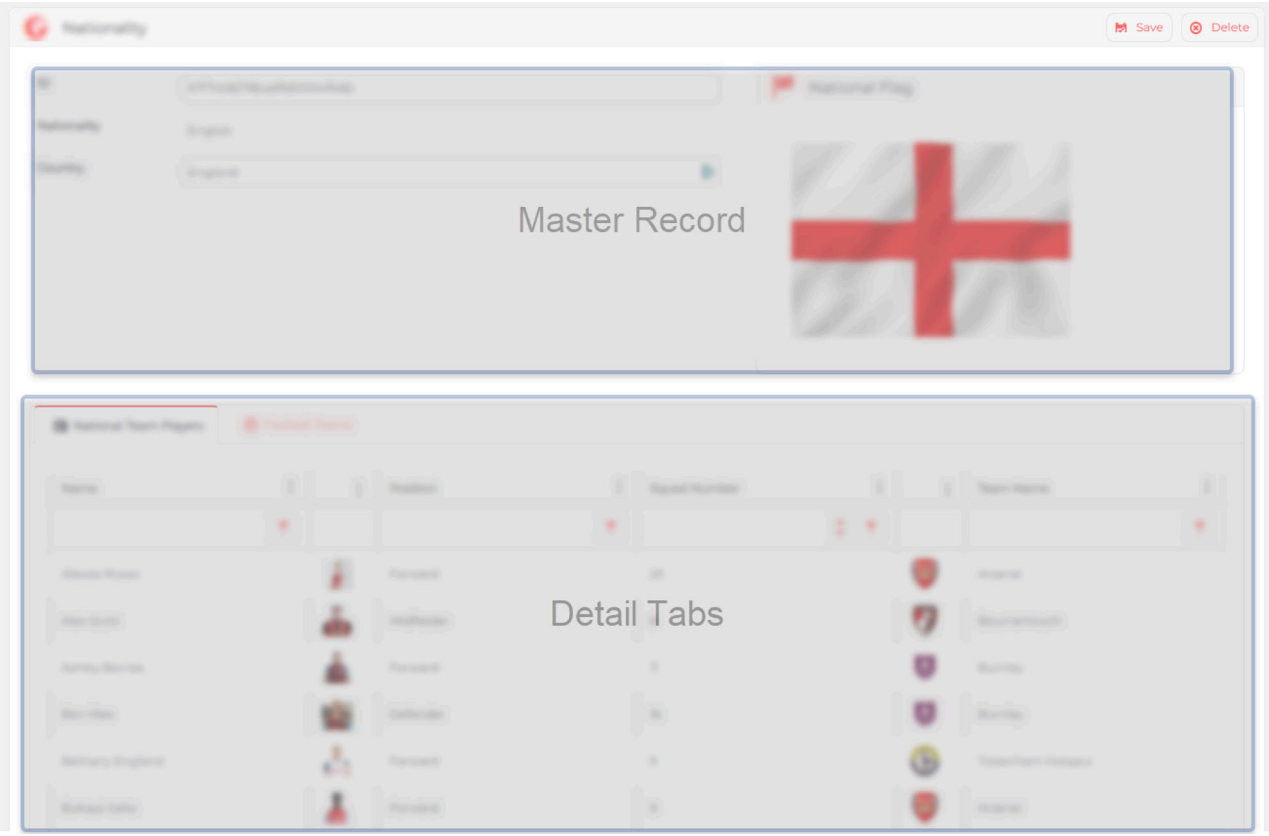


The form designer will open:



Tabs

The tabs toolbar is the selected tool by default. Each data entry form is designed as a master/detail meaning that the master record is shown at the top, and any further details about linked entities is shown below in a series of tabs like this:

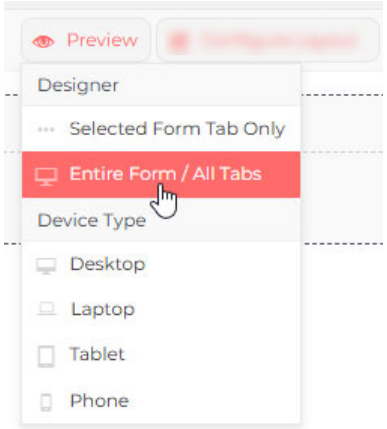


The Main Top Region refers to the master record on the top of the form.

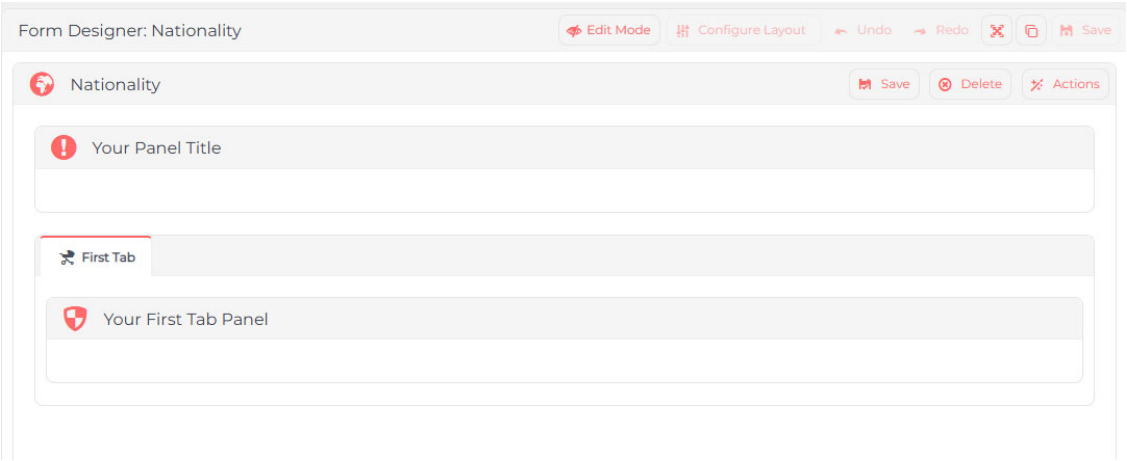
We will design the nationality fields in this region.

Preview

The concept of form rows, columns and panels was introduced here, so we will now take a look at what the default data entry form looks like without any fields by clicking the Preview button to show this drop down menu:



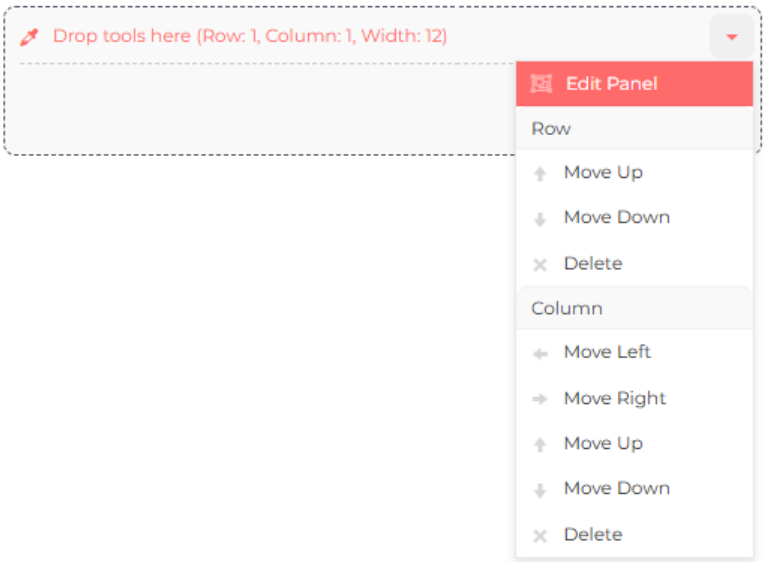
We can see the Nationality form and the master panel at the top of the form, and the First Tab (detail) beneath:



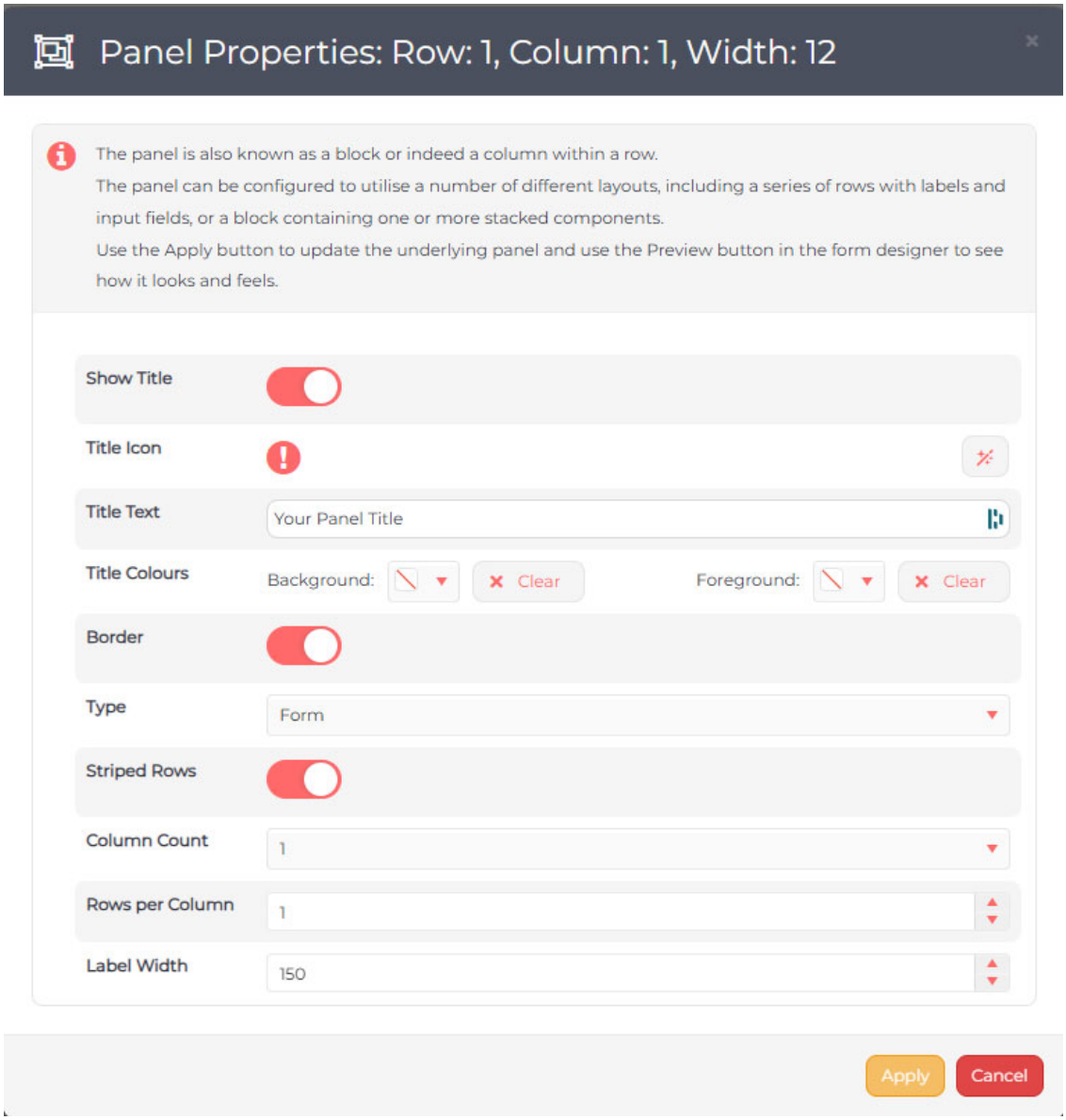
Click the **Edit Mode** button to return to the form designer editor.

Edit Panel

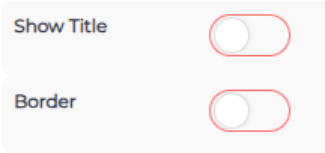
Click the down arrow in the Row 1, Column 1 panel and choose the **Edit Panel** option:



This opens this modal popup:



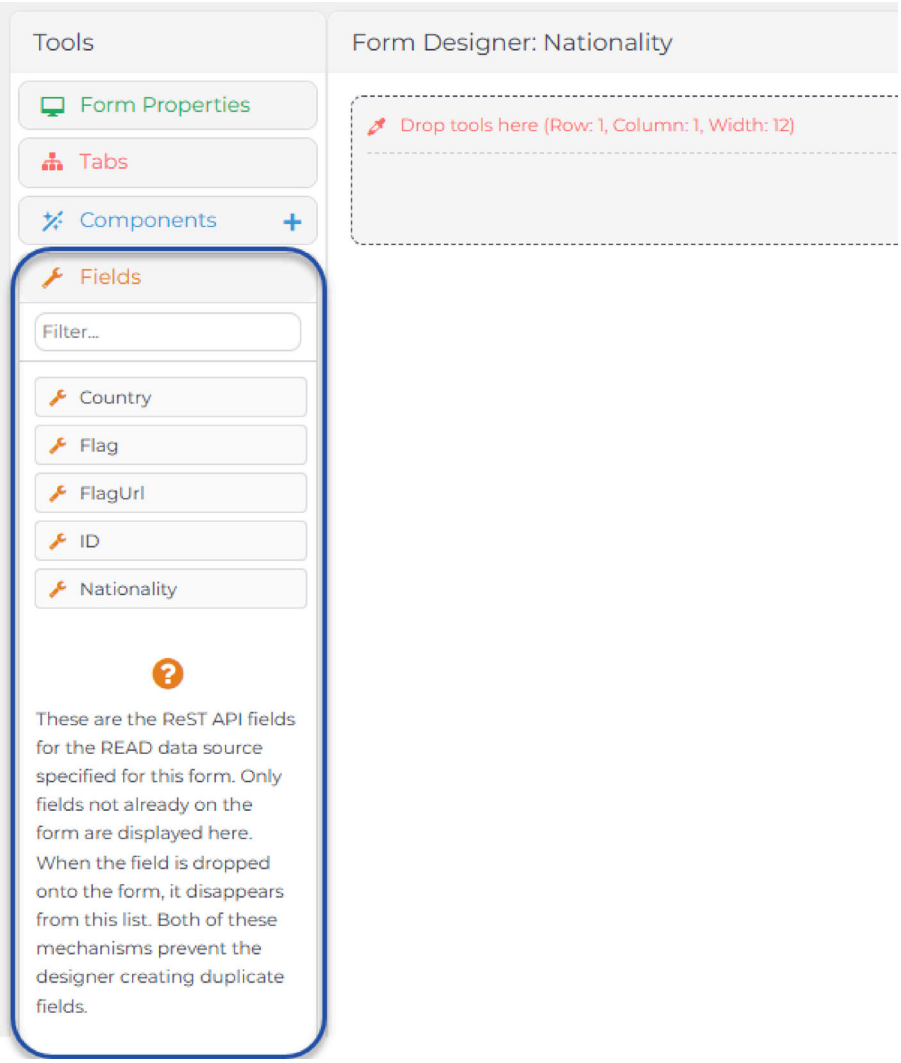
For this simple form, we will hide the title and border by unchecking these properties:



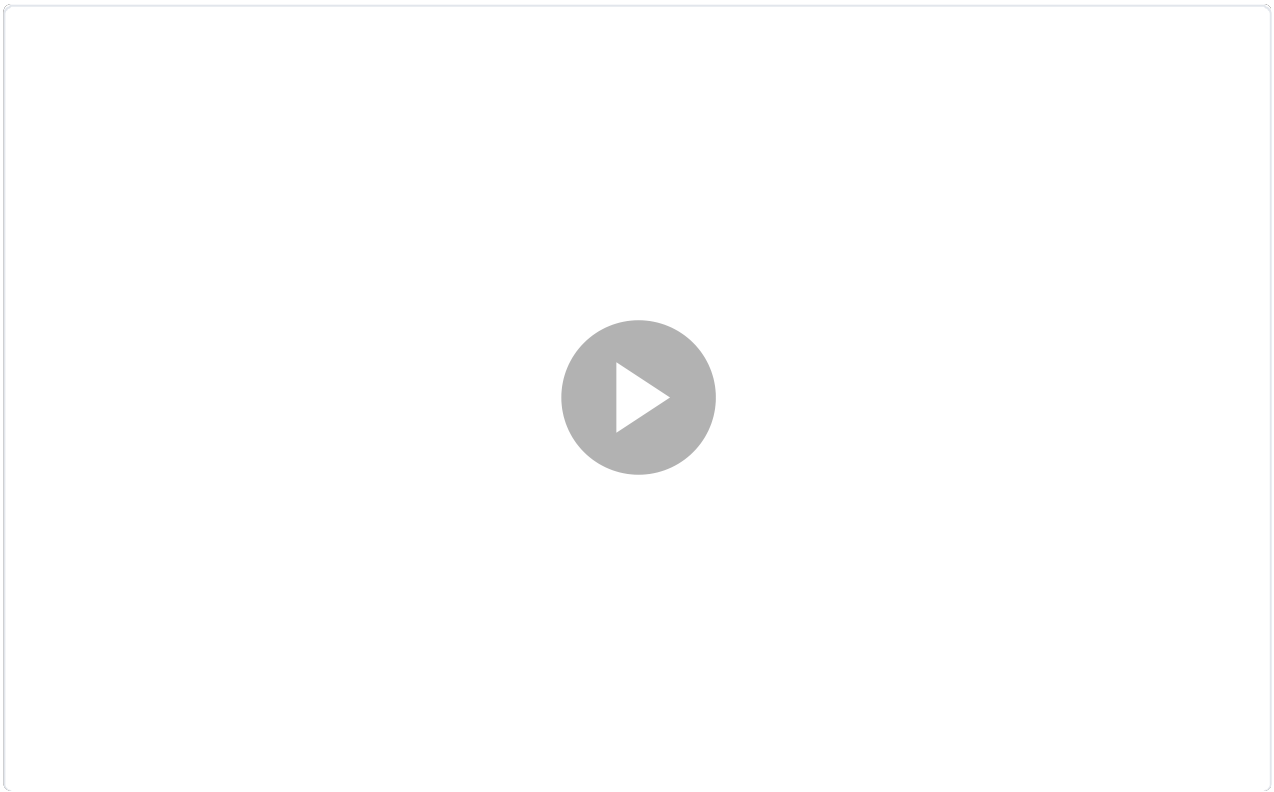
Click the **Apply** button.

Fields

Click on the fields panel in the toolbar to show the available fields. These fields are those connected to the Read data source request we added earlier:

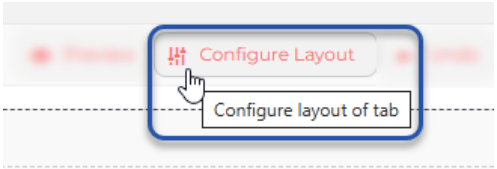


Drag these fields into the form design panel, and preview how it looks:

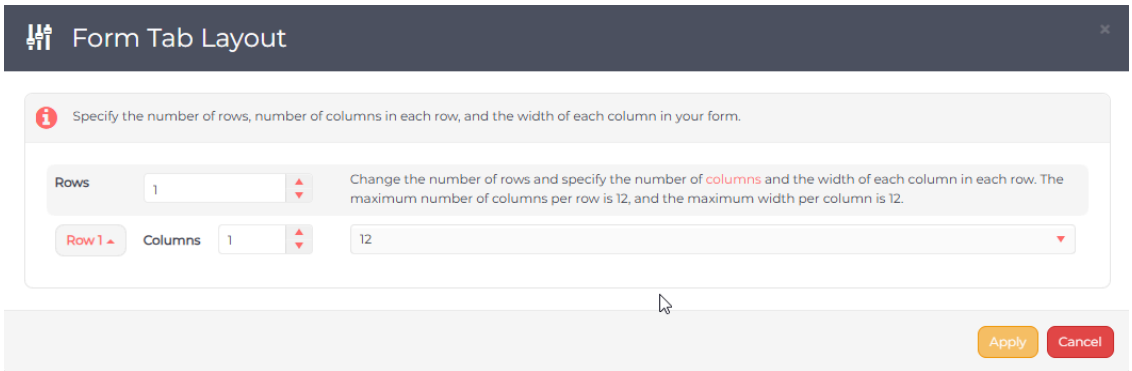


Configure Layout

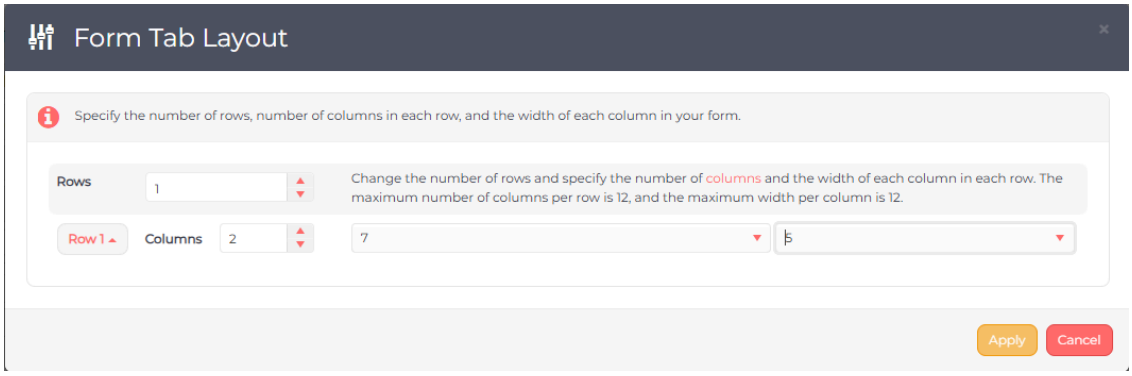
We want to show the nation flag on the form, so we will configure the layout to create another column to the right of these fields to span the 3 rows we created. Click this button:



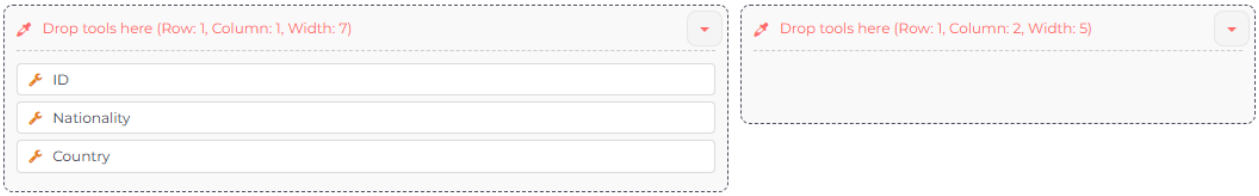
This will open this modal popup to modify the form tab layout:



Use the up arrow on the Columns field to increase the number of columns to 2, then set the first column width to 7 and the second to 5:

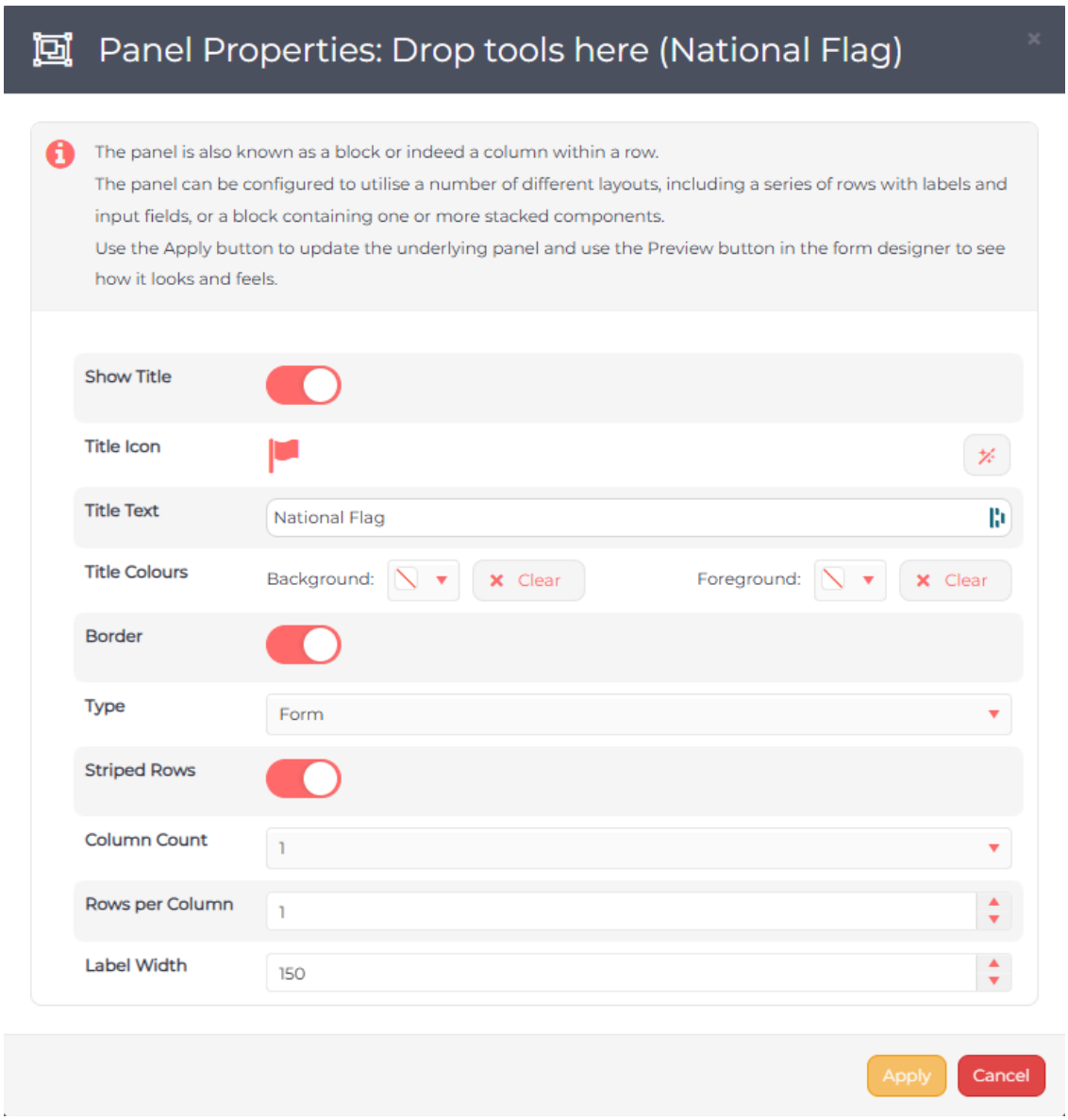


Then click the **Apply** button to apply the layout to the form design. Your form should look like this:



Nation Flag

Click the drop down menu on the right panel to edit it, and make the following changes in the modal popup:



We set the title of the panel to "National Flag", set the icon to be a flag. Click the **Apply** button.

Now we will drag the Flag field into this panel and hide its label:

Save Form

Now save the form design using this button:

We have now completed the design of our data entry form to read a record.

We will now add this to the navigation bar.

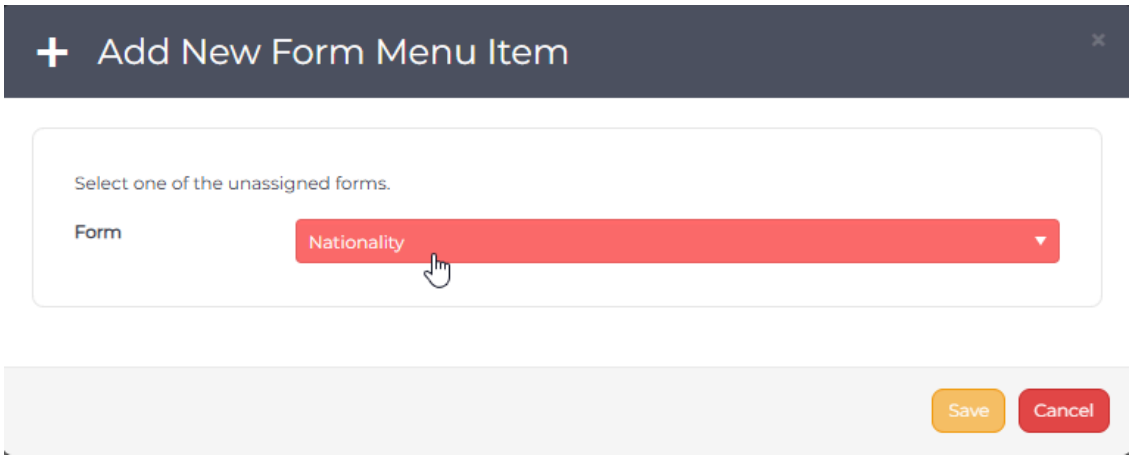
Navigation Bar

When we add this form to the navigation bar, we will be able to test it.

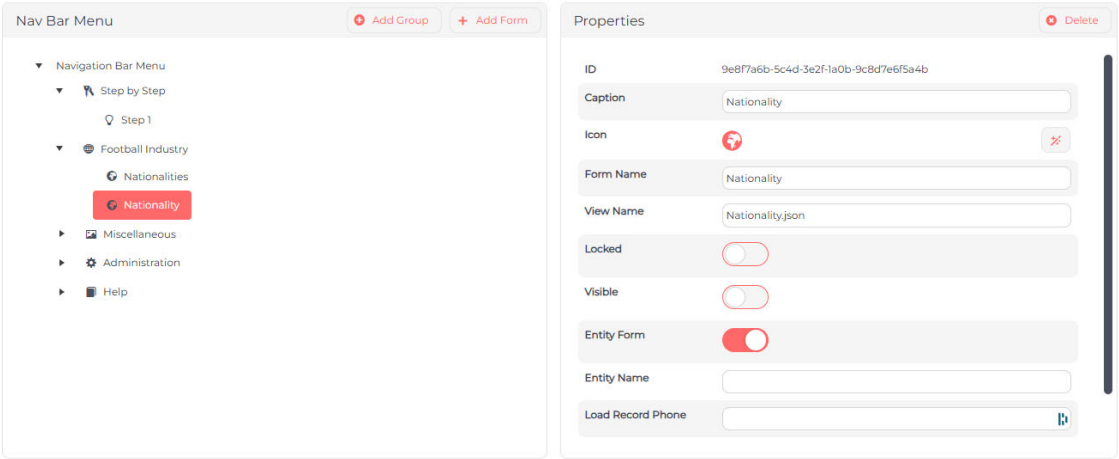
Open App Studio and select the Navigation Bar configurator and select the Football Industry folder you created previously.

Add Form

Click the **Add Form** button to open this modal popup where you should select the Nationality form you created earlier:



Click the **Save** button which will close the popup and show the new form in the nav bar menu:



Properties

Check or set these properties:

Visible

Because this is a data entry form, we only want it to be visible in the nav bar when the form is open and showing a record, so this should be unchecked.

Entity Form

This is a data entry form which models entities, so this should be checked.

Apply & Close

Click the Apply & Close button to persist the navigation bar and refresh the nav bar.

Configure Drill Down

The navigation bar should not show any change from the last time you saw it because the Nationality form will only appear on it when the form is opened.

In order to test this data entry form, we need to enable drill down from the nationalities lookup form we created previously.

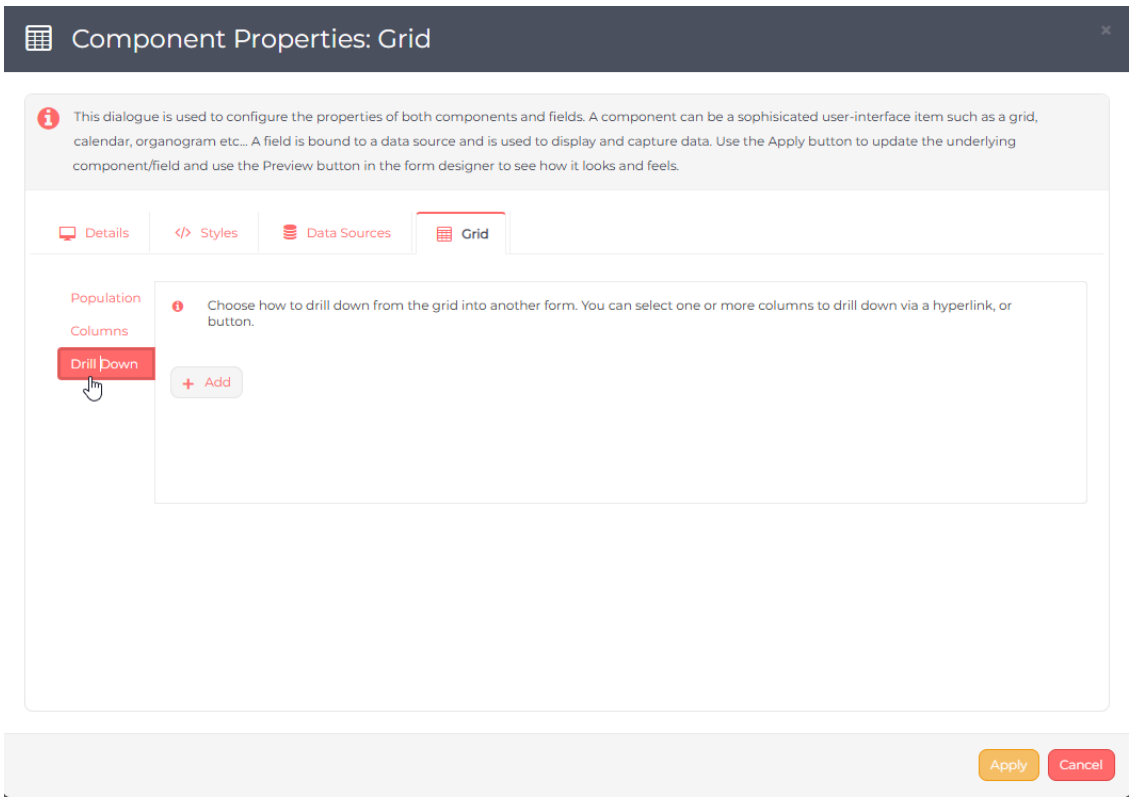
Configure Grid Component

Open App Studio and select the Forms configurator.

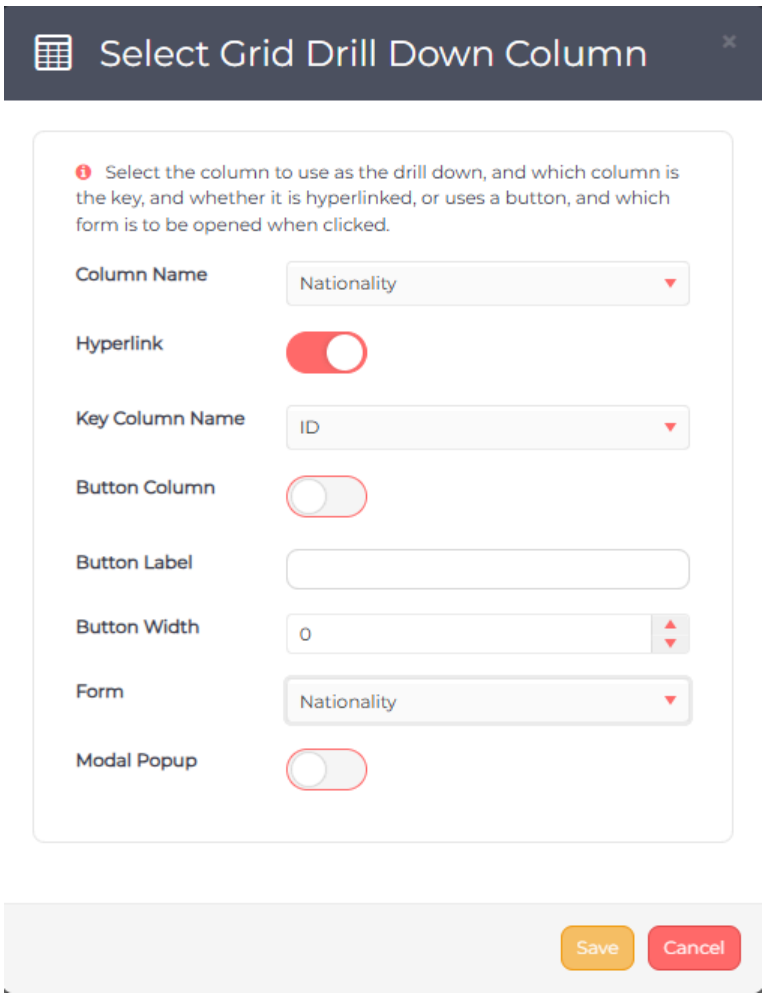
Open Nationalities Form Designer

Open the nationalities form in form designer by choosing **Edit** then **Design**.

Click on the Grid component to open this modal popup form:



Select the Grid tab and click the Drill Down left menu option. Click the **Add** button which will open this modal popup:



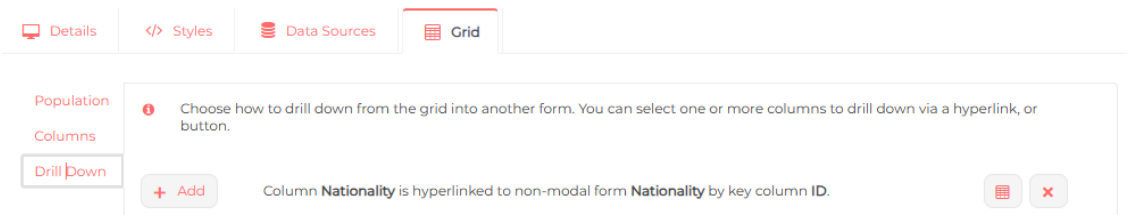
Choose `Nationality` as the Column Name.

Make sure that the Hyperlink is checked.

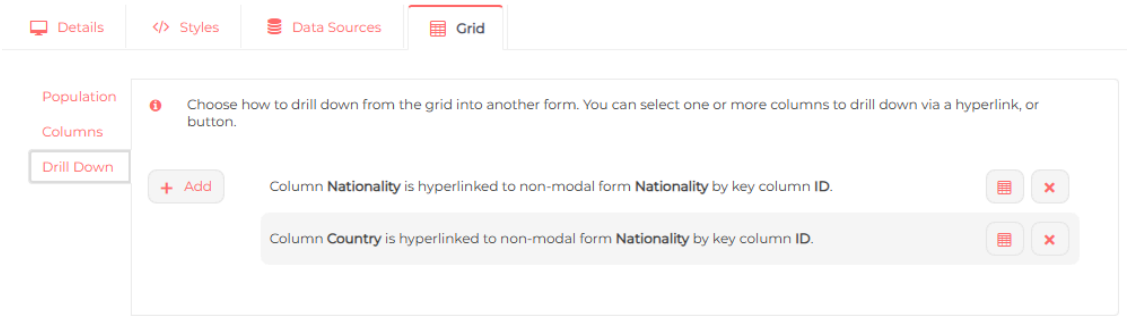
Ensure that Key Column Name is the `ID` i.e. the identifier of the nationality record.

Select the `Nationality` form in the Form field. This is the data entry form we [created earlier](#).

Click the **Save** button which will close the popup and show the drill down hyperlink details:



You should add another hyperlink to the `Country` column so that both of these columns can now be used to drill down into the data entry form:



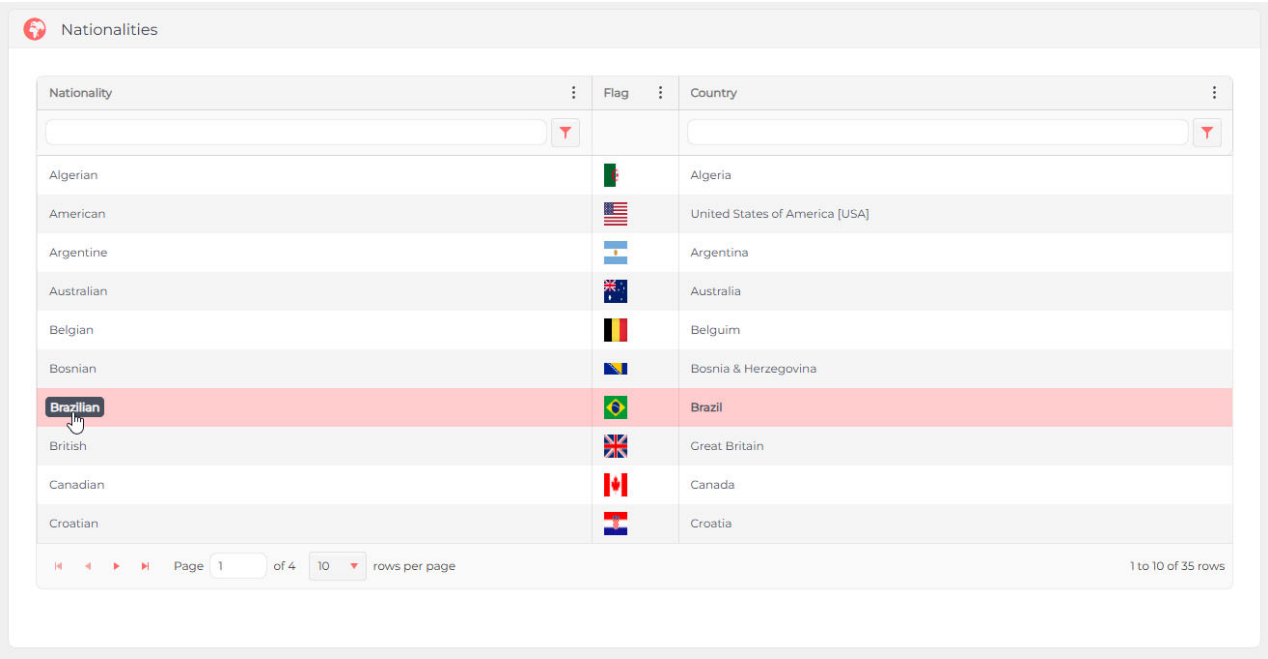
Click the **Apply** button, and then **Save** the form design.

Test Data Entry Form

Open the Football Industry group on the navigation bar.

Nationalities

Click the Nationalities nav bar menu to open the lookup form:

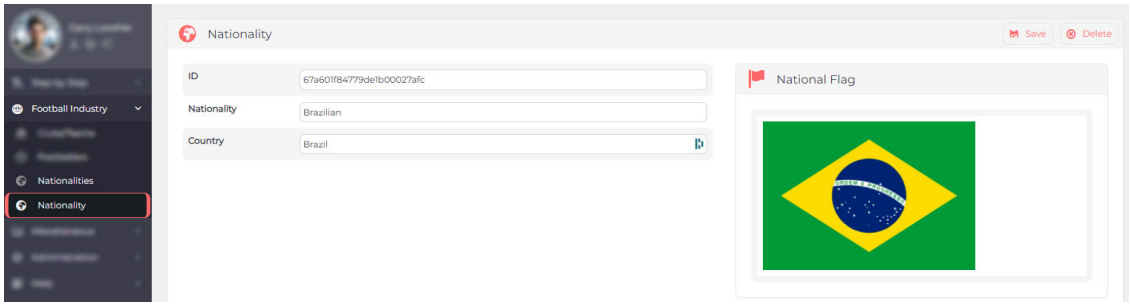


If you hover your mouse over any of the nationalities or countries, you should see that it becomes highlighted. This proves that the drill down configuration has been applied.

Click any nationality or country.

Nationality Form

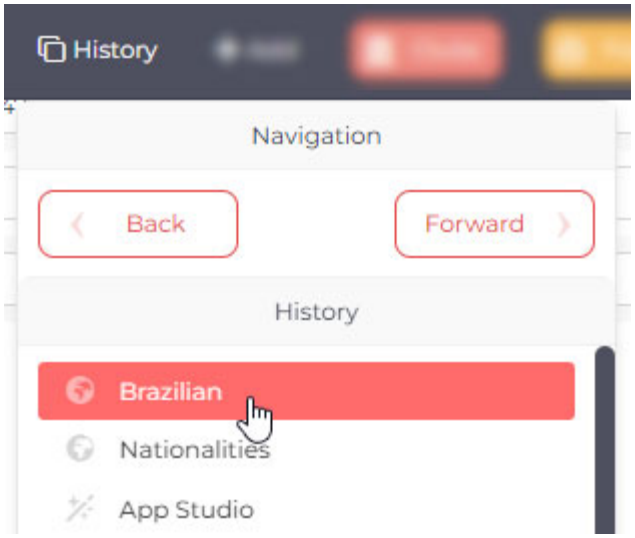
The nationality form should open and show the selected nation details including their flag:



Notice also how the navigation bar now shows the Nationality form as being open?

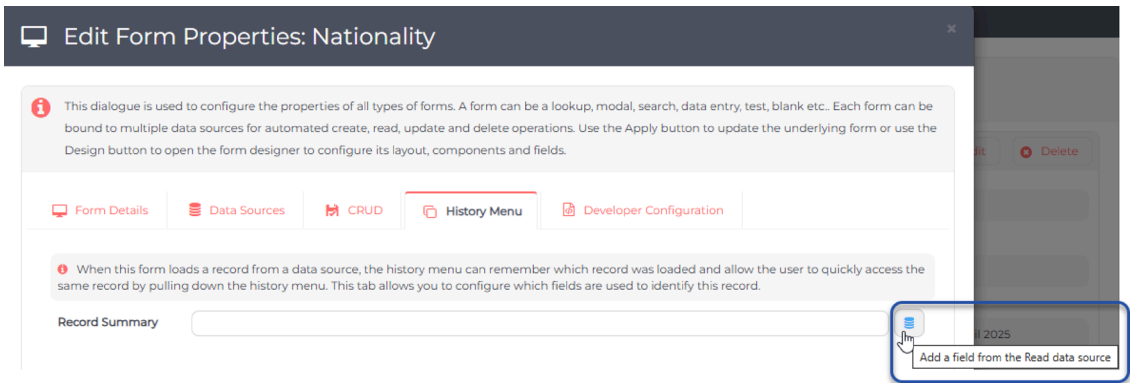
History Menu

When the history menu appears, we want the name of the nationality to appear, not the name of the form for example:

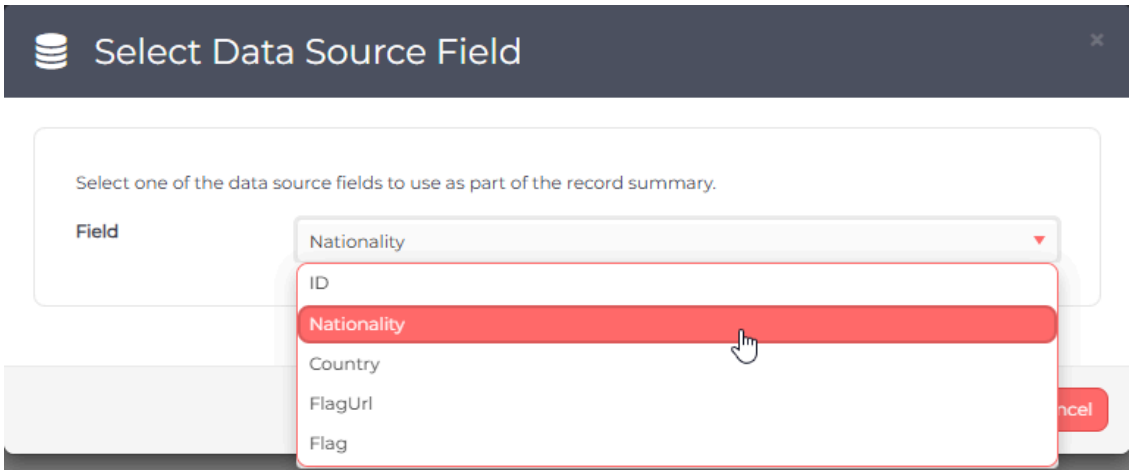


Configuring Record Summary

Open App Studio and select the Forms configurator. Edit the properties of the Nationality form and click the History Menu tab:



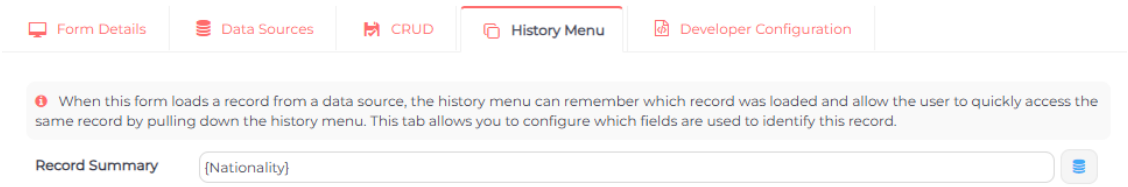
Click this button to open this modal popup:



Choose the `Nationality` field and click the **Save** button to persist this setting and close the popup.

Record Summary

The record summary should now show `{Nationality}` indicating that the `Nationality` field will be displayed in the History Menu:



Apply

Click the Apply button to persist this.

Test Record Summary

Open another nationality from the nationality lookup form, and then click on the **History** drop down menu. You should see the last nationality you opened at the top of the list?

National Team Players

We will now design the nationality form and add a grid component to the first tab.

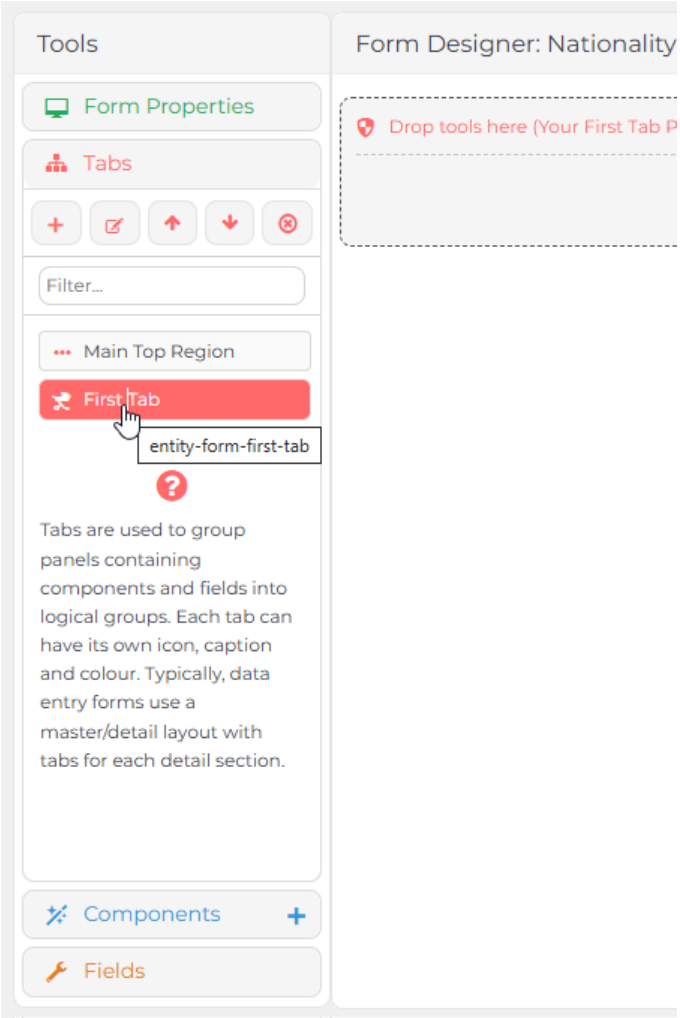
Open App Studio and select the Forms configurator.

Open Nationality Form Designer

Open the Nationality form in form designer by choosing **Edit** then **Design**.

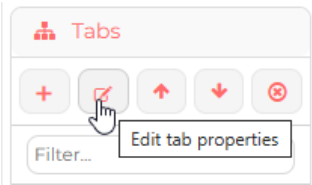
Edit Tabs

Select the Tabs panel in the Tools panel:

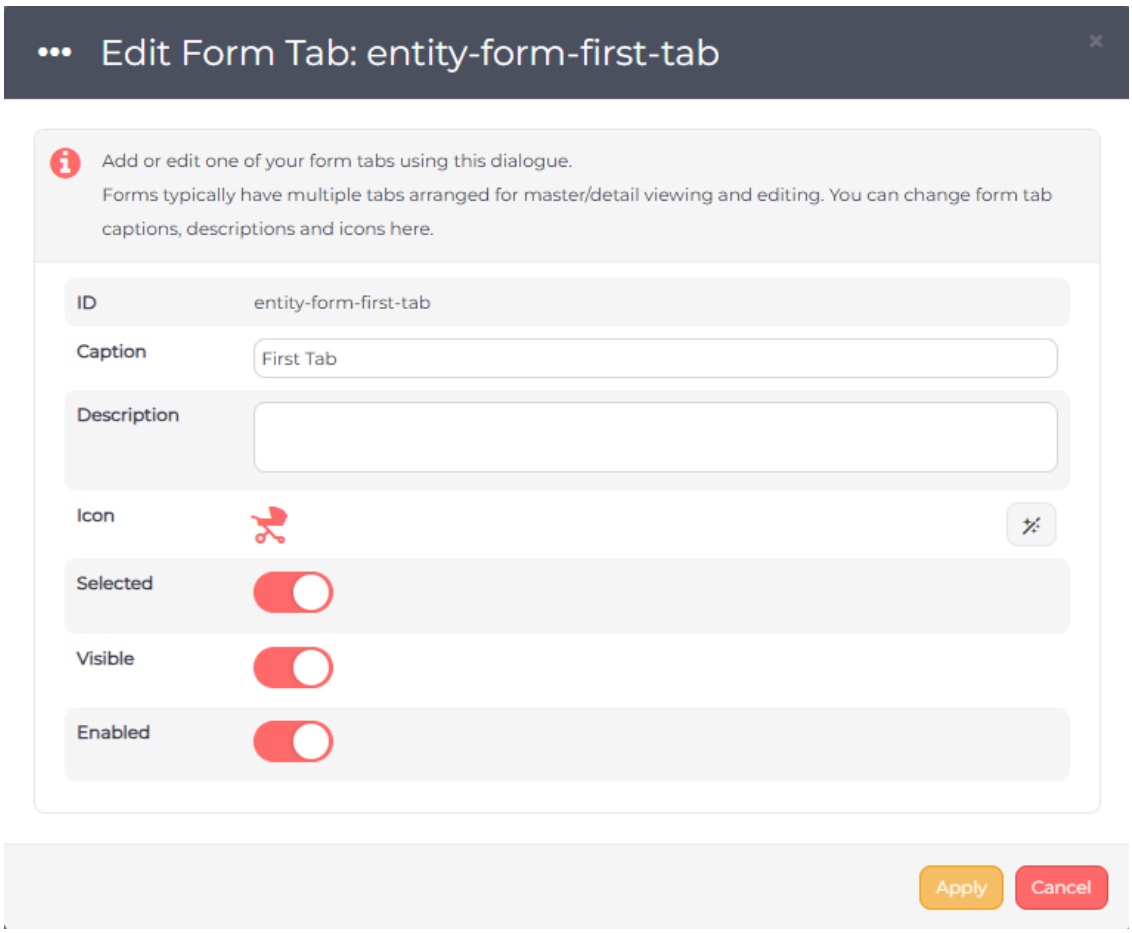


Edit Tab

Click this button:



This will open up this modal popup to edit the tab properties:



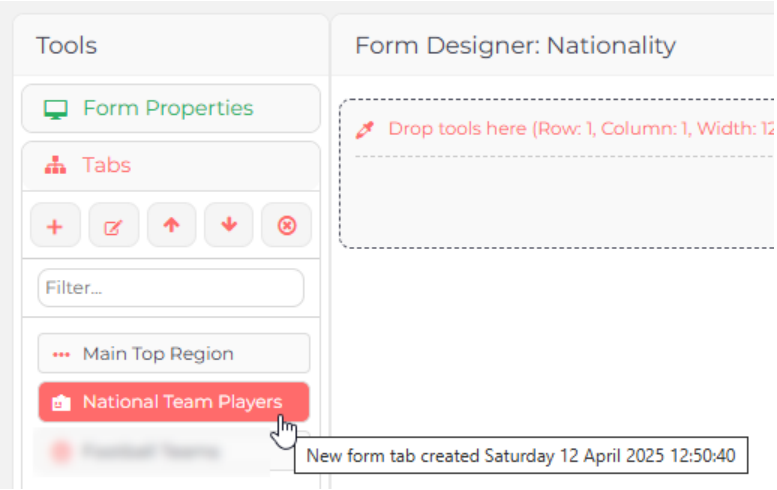
Set the **Caption** to say "National Team Players".

Set the **Icon** to a ball using the filter "ball".

Check **Selected**, **Visible** and **Enabled**.

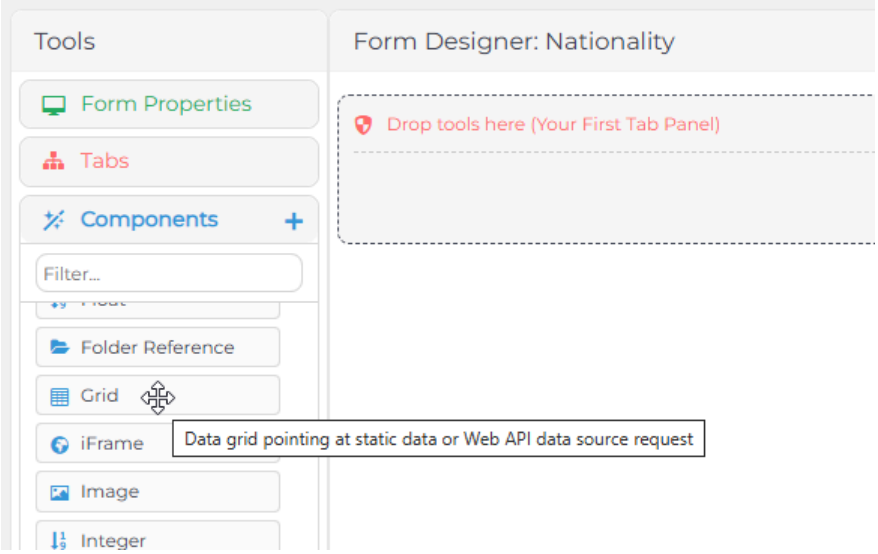
Click the **Apply** button to save the properties and close the popup.

The selected tab should now reflect the changes:

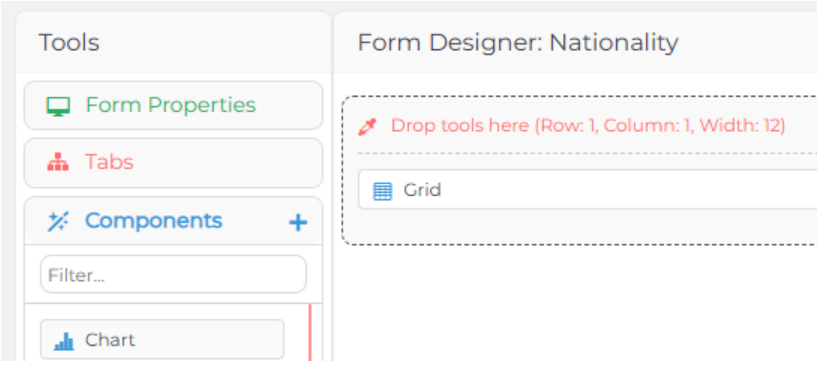


Add a Grid Component

Open the Components panel and drag a Grid into the panel:



After dragging and dropping, the grid should be visible:



We cannot configure the grid yet because we have not yet created a data source to show a list of footballers.

Instead, we need to Save the form design to persist our design.

Add a Footballers Data Source Request

In order to display a list of footballers on our Nationality form, we need to create a data source request connected to the appropriate ReST API.

Thinking ahead about what we'd like to do with this list of footballers, we'd certainly want to drill down into a footballer form when we get that far, so we ought to create a custom variable to be able to bind it to the unique key identifier.

We will do that first.

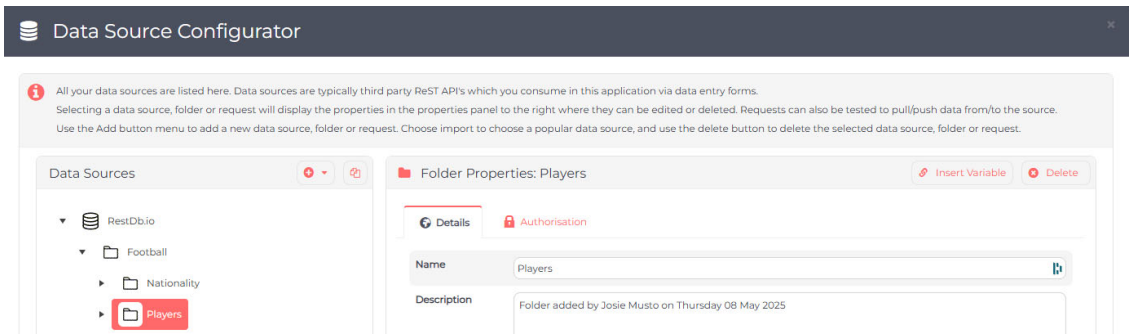
Create Custom Variable: Footballer-ID

Open the App Studio, then the Custom Variables configurator, and add the client-side custom variable called `Footballer-ID`.

We will use this when creating the footballers data source request.

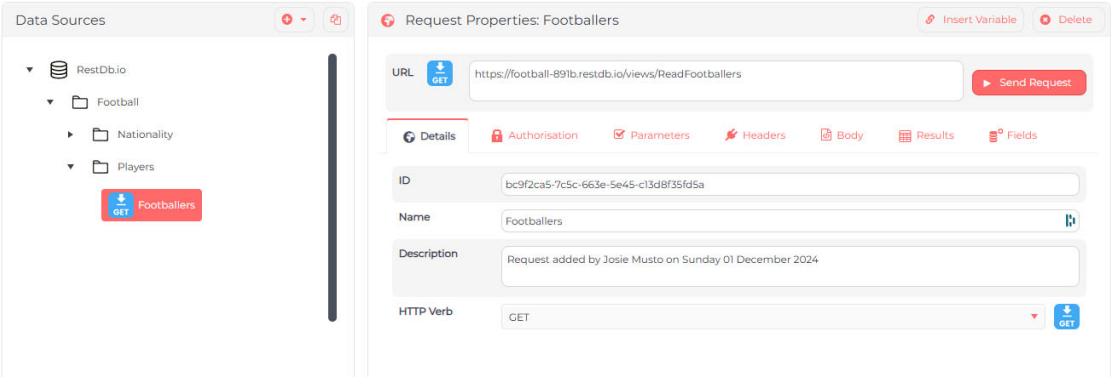
Players Folder

Open App Studio, open the Data Sources configurator, select the Football folder you created previously, then use the add button menu to create a new folder called "Players":



Footballers Request

Use the **Add** button menu to create a new request called "Footballers":



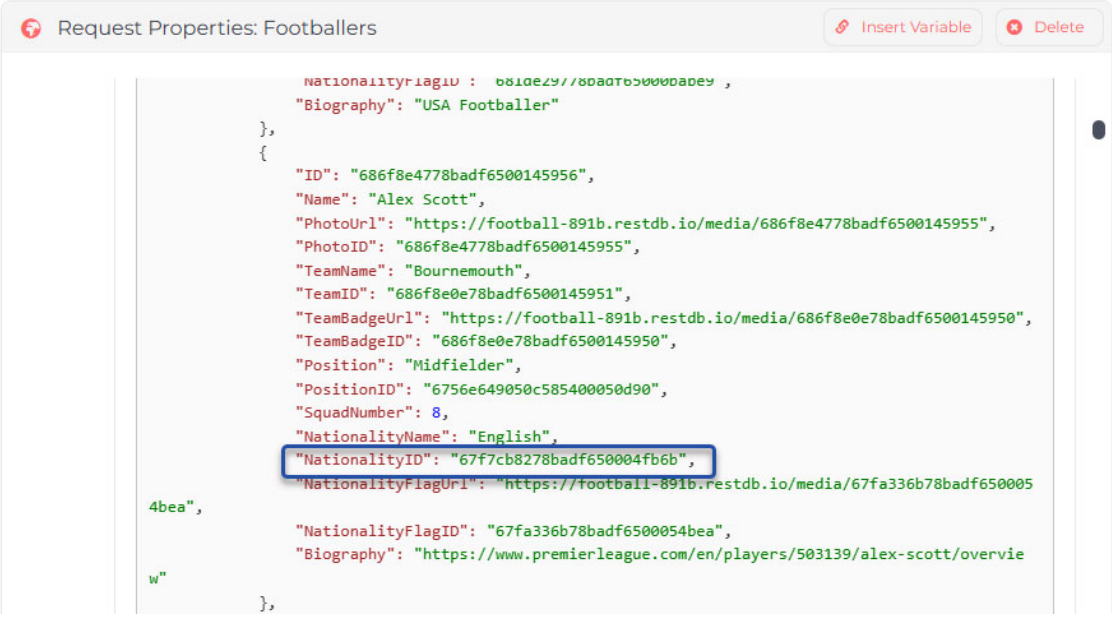
Paste in this URL which is defined in the ReST API we are integrating:

`https://football-891b.restdb.io/views/ReadFootballers` ↗

The Authorisation should always default to "Inherit from parent".

Send Request

To get the list of fields, use this button to send the request and get the list of fields and data. We can see that there is a field we can use to identify the nationality by its unique identifier:

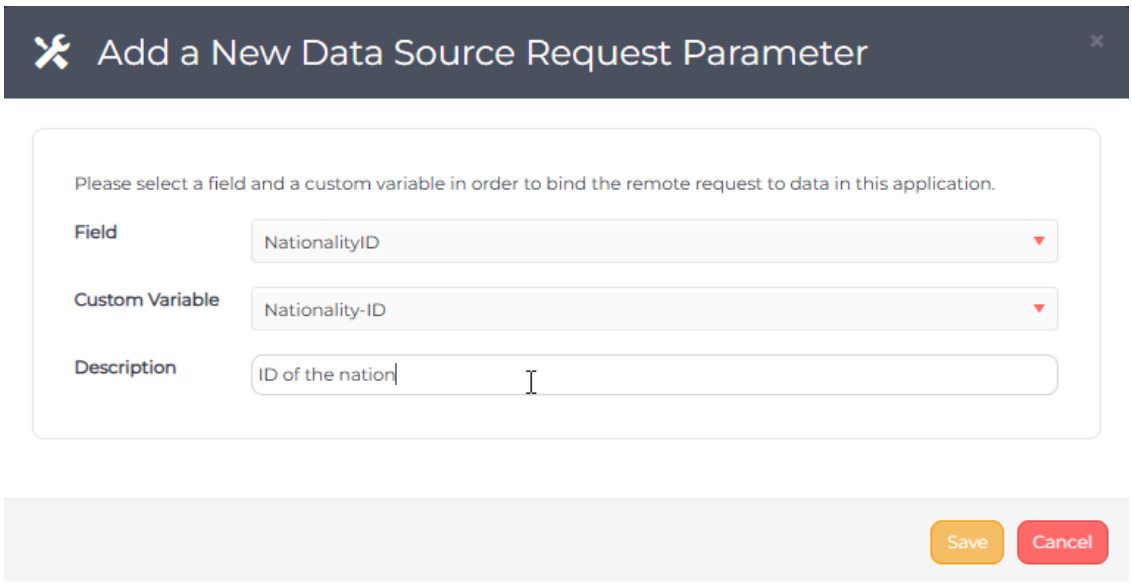


Parameters

Because we want this data source request to be re-used throughout the application, we are going to use optional parameters linked to custom variables. Only if a custom variable value is not empty will it be appended to the URL at run-time.

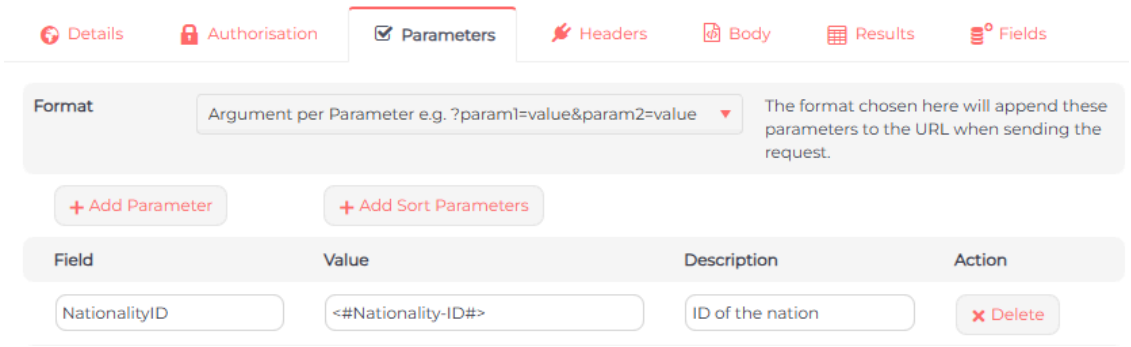
We previously created the `Nationality-ID` custom variable and we know that this ReST API can return footballers belonging to a particular nation. Sending the request identified that the `NationalityID` URL parameter should be the correct URL parameter name.

Click the Add Parameter button to add this parameter:



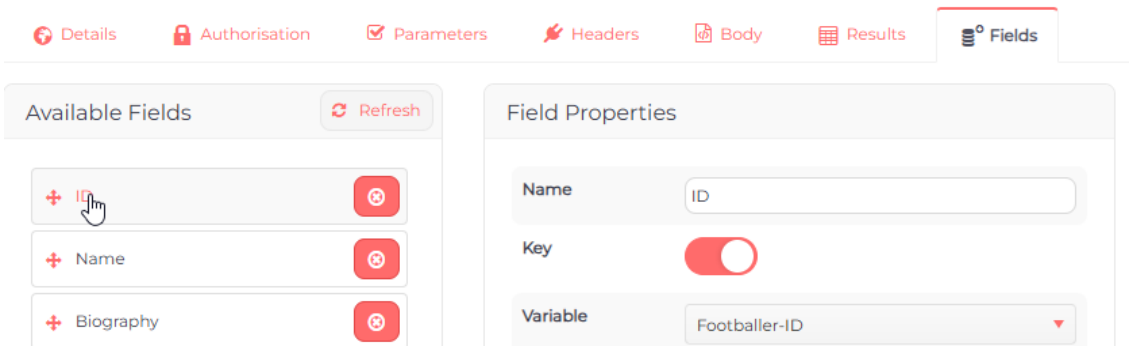
Select the Field `NationalityID`, then assign that to the Custom Variable `Nationality-ID` then click the **Save** button.

The parameter is added to the list:



Fields

In the list of fields populates when sending the request, locate the ID field, then set this as the Key and assign this custom variable `Footballer-ID`:



Because we know by inspecting the data returned, we will also make the fields with https data into images for example:

Do the same for these fields:

- PhotoUrl
- TeamBadgeUrl

That's all we need to do at this stage.

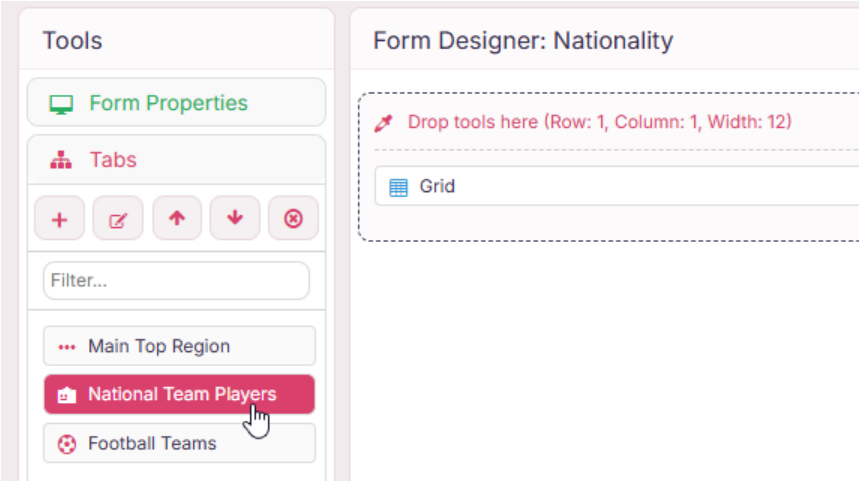
Wire up the Footballers Grid

We can now point the grid on the nationality form at this new data source request.

Open App Studio and select the Forms configurator. Open the Nationality form, edit it, and Design it to open the forms designer.

Add Read Data Source

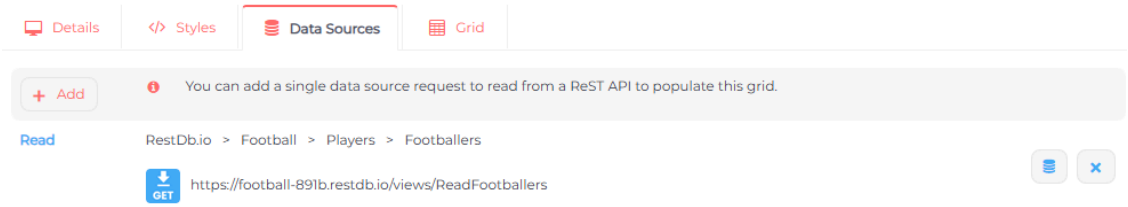
Click on the grid on the National Team Players tab:



This opens the Component Properties: Grid modal popup. Click on the Data Sources tab.

Use the Add button to choose this data source request.

The popup form should now show that a Read data source request is associated with the grid:

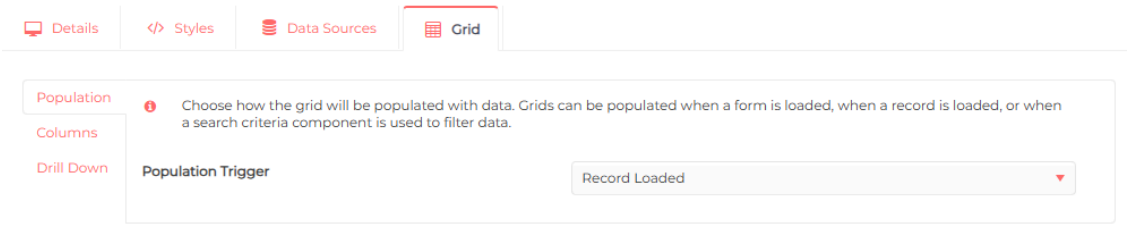


Grid Columns

Click on the Grid tab.

Population Trigger

Select "Record Loaded" in the drop down combo:



Columns

Click on the left tab Columns, and order the columns, hiding some, as you would like to see them.

Because this grid is on a nationality form, we do not need to see the nationality column in this grid. Here are the properties for each column you should set:

Column	Properties
`ID`	Invisible
`Name`	Visible
`Biography`	Invisible
`NationalityFlagUrl`	Invisible
`NationalityID`	Invisible
`NationalityName`	Invisible
`PhotoUrl`	Type: Image URL, Image Size: 32 × 32 Visible, Caption: [space], Width: 70
`Position`	Visible
`PositionID`	Invisible
`squadNumber`	Type: Number, Visible, Caption: Squad Number
`TeamBadgeUrl`	Type: Image URL, Image Size: 32 × 32, Caption: [space], Width: 70
`TeamID`	Invisible
`TeamName`	Visible, Caption: Team Name

Drill Down

Leave this for now as until we create the footballer form, we have nothing to drill into!

Apply

Apply the changes to these properties.

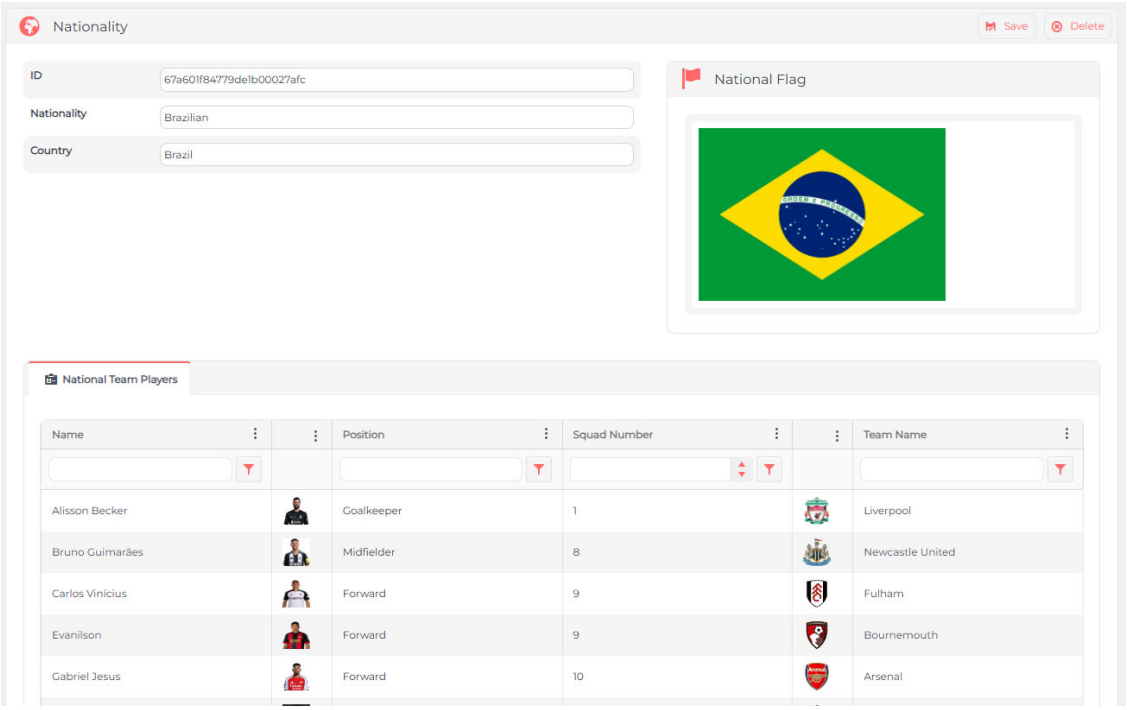
Save

Save the form design to persist the configuration.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Nationalities item. Drill down into any nationality to open the form.

The nationality form now shows the National Team Players in the grid.



We will revisit this form to configure drill-down once we have completed all CRUD configuration and created the footballer forms.

Nationality Form: Update

Configure the nationality form to update a record.

This is the process we will follow.

UPDATE

- Create custom variables for each field
- Add an 'update' data source for updating a single nationality record
- Link the key nationality record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'update' data source to the data entry form
- Configure the update button
- Test that we can update a nationality using the Update button

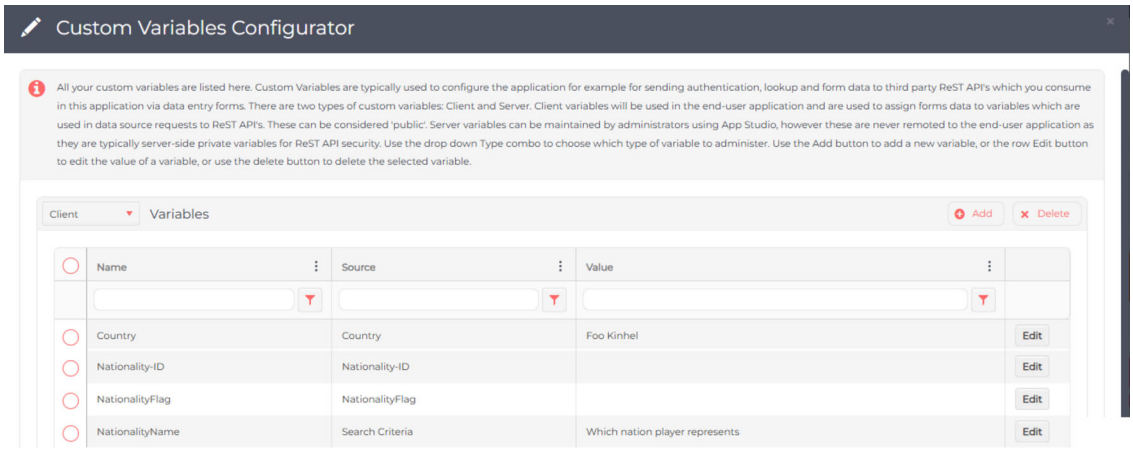
Create Custom Variables

In order to update a nationality record, we need to create custom variables for each form field so that we can send these to the ReST API.

Open the App Studio, then select the Custom Variables configurator.

- Add a client-side custom variable called `NationalityName`.
- Add a client-side custom variable called `Country`.
- Add a client-side custom variable called `NationalityFlag`.

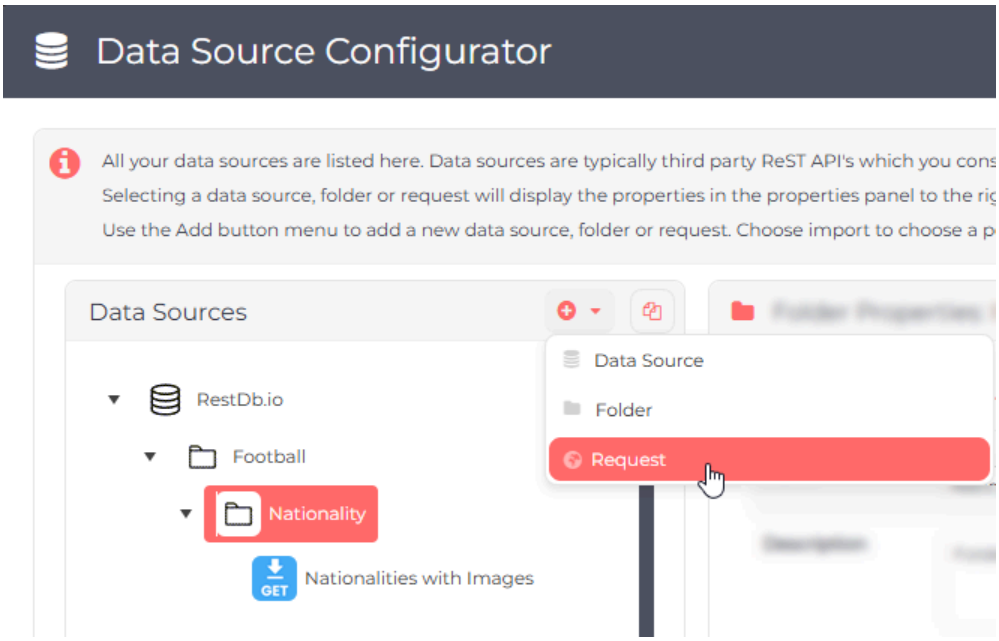
The new custom variables should now be displayed:



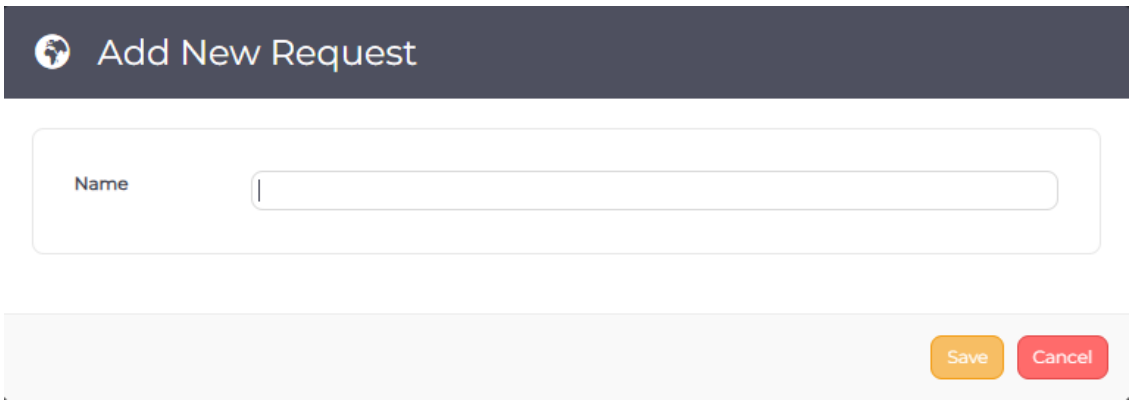
We will need to link these new custom variables to each form field, but first we will use them in a new data source to update the record.

Add an UPDATE Data Source

Open App Studio, open the Data Sources configurator, select the Nationality folder you created previously, then use the add button menu to create a new request:

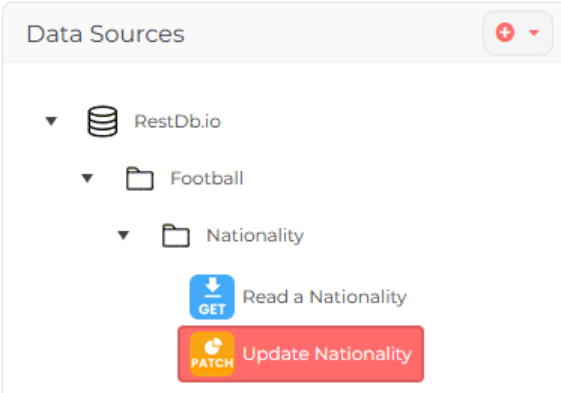


The Add New Request modal popup form shows asking you to name the request:



Type a meaningful name such as "Update Nationality" and click Save.

The request will be added to your tree view beneath the Nationality folder:



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

URL

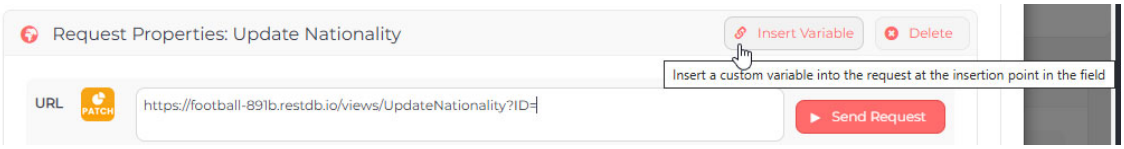
`https://football-891b.restdb.io/views/UpdateNationality` ↗

It is known from the documentation that this ReST API end-point has a nationality identifier property ?ID=.

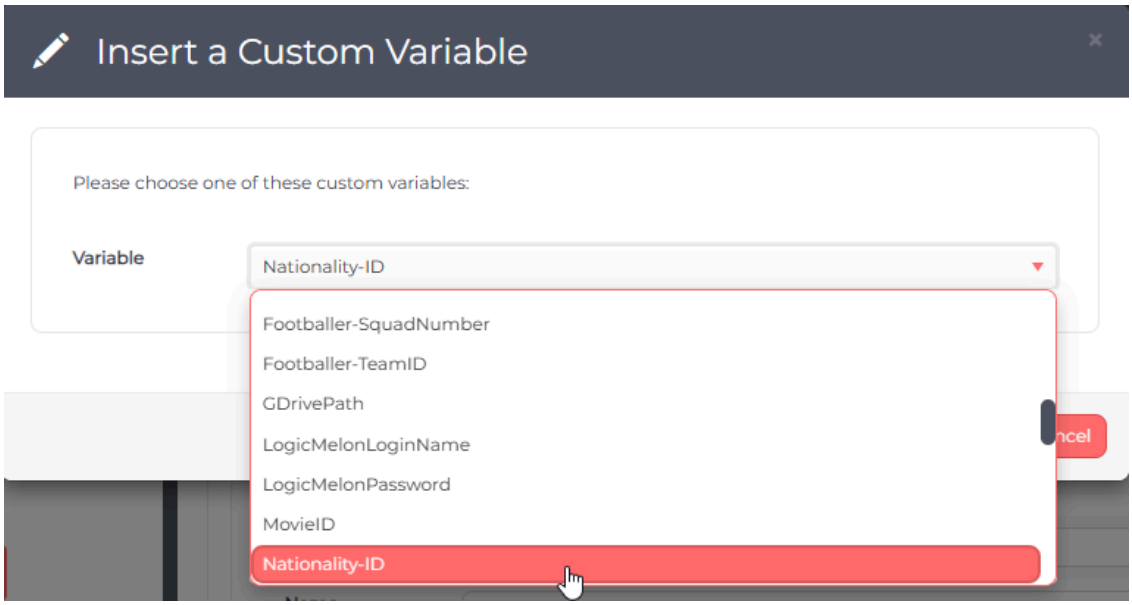
We therefore type this parameter list into the URL so that it reads:

`https://football-891b.restdb.io/views/UpdateNationality?ID=` ↗

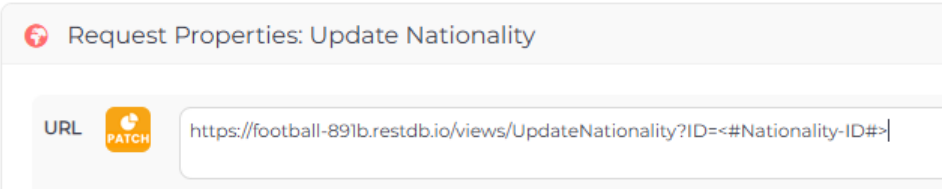
Whilst the caret is still blinking after the last character typed, click on the Insert Variable button:



This will open the following modal popup for where you should select the Nationality-ID custom variable you created previously in the Variable drop down list . Click the Select button.



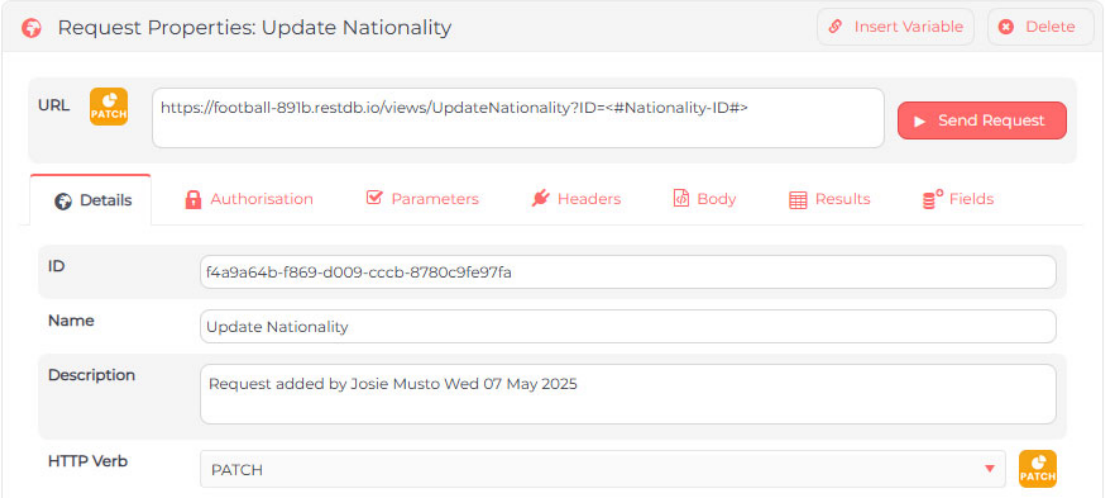
The URL should now show the custom variable appended to the end:



HTTP Verb

The default GET verb/method is not correct for updating a single record using the ReST API and should be set to PATCH for this specific back-end end-point.

The request should now look like this:



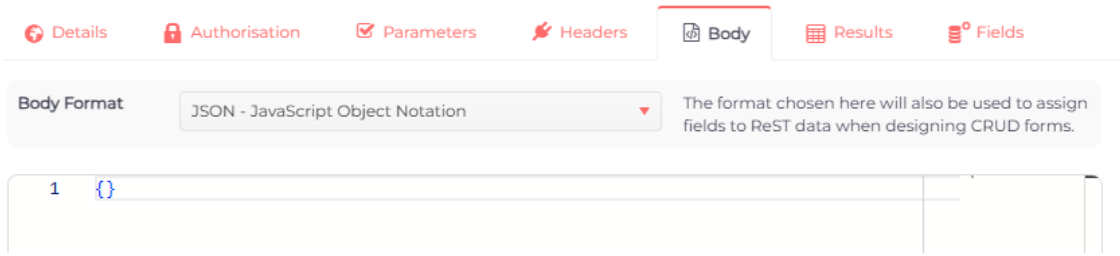
Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Body

When we are updating or creating data, we will be sending the data in the body of the request, so we configure this before sending the request to test it, using the new custom variables. Here is the empty Body tab:



We will always use the JSON body format as this gives us maximum control.

From the ReST API specification, we know that this endpoint expects data in this format:

```
{
    "Name": "",
    "Country": "",
    "FlagURL": ""
}
```

Our job is to now associate each field with the appropriate custom variable.

Put the carat inside each double quotation and use the Insert Variable button to select the respective custom variables setup above.

After all three have been inserted, the body should now look like this:

```
{
    "Name": "<#NationalityName#>",
    "Country": "<#Country#>",
    "FlagURL": "<#NationalityFlag#>"
}
```

Send Request

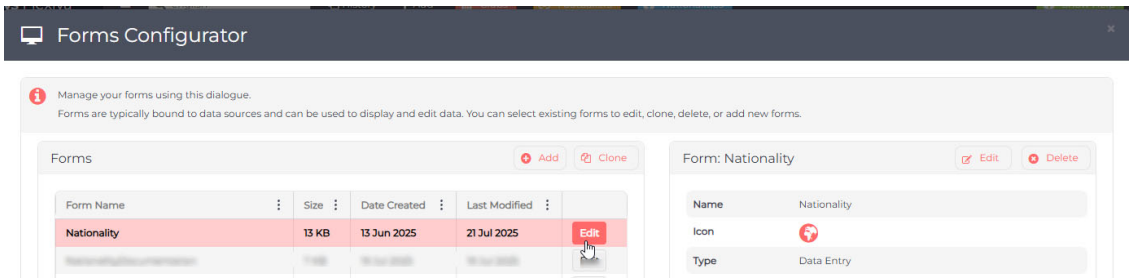
It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to update an existing record in order to test this.

The best way of course to test this is to use the actual nationality form we created here, and link that form to this data source request, then we can test it.

Edit Form

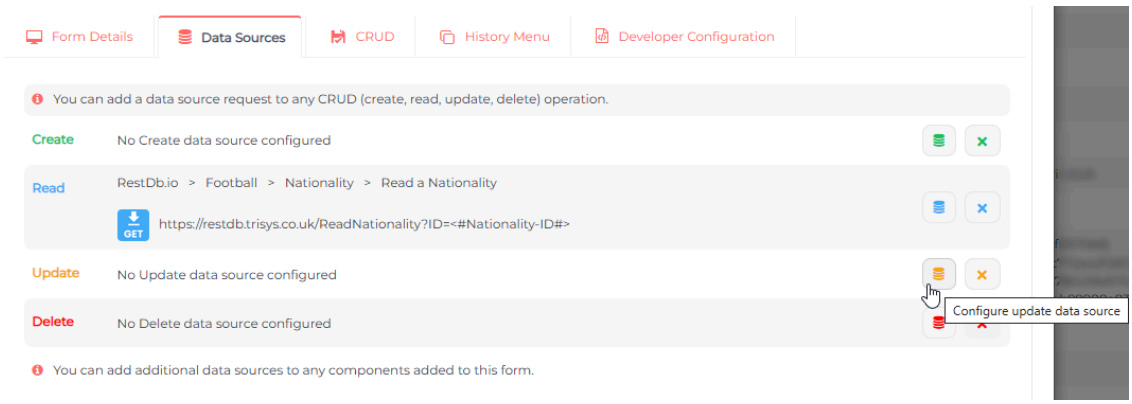
Open App Studio, open the Forms configurator, then select the Nationality form:



Click the Edit button to open the form properties modal popup.

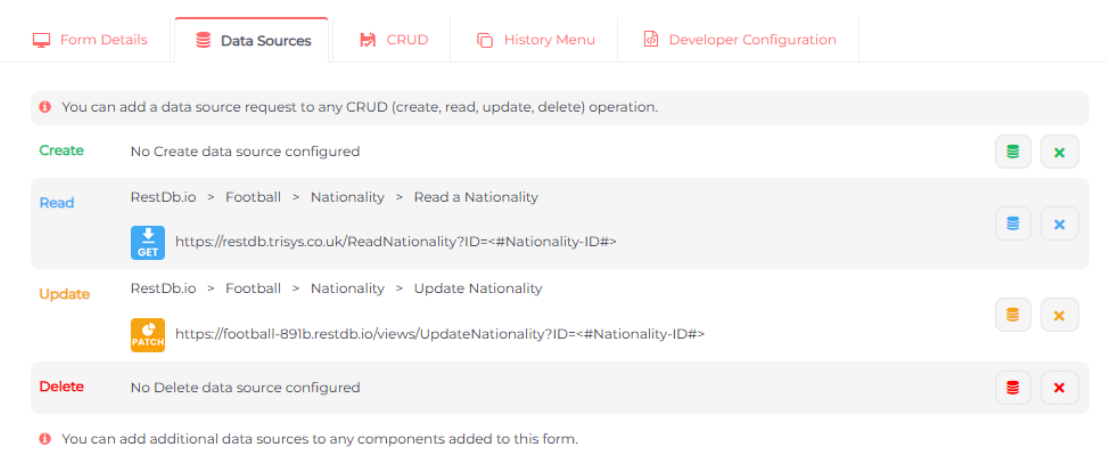
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Update data source:

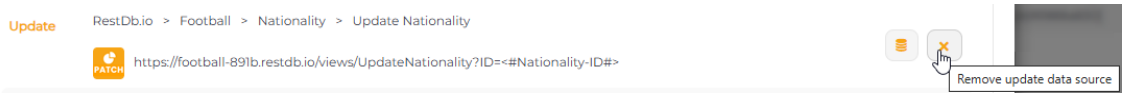


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the Select button.

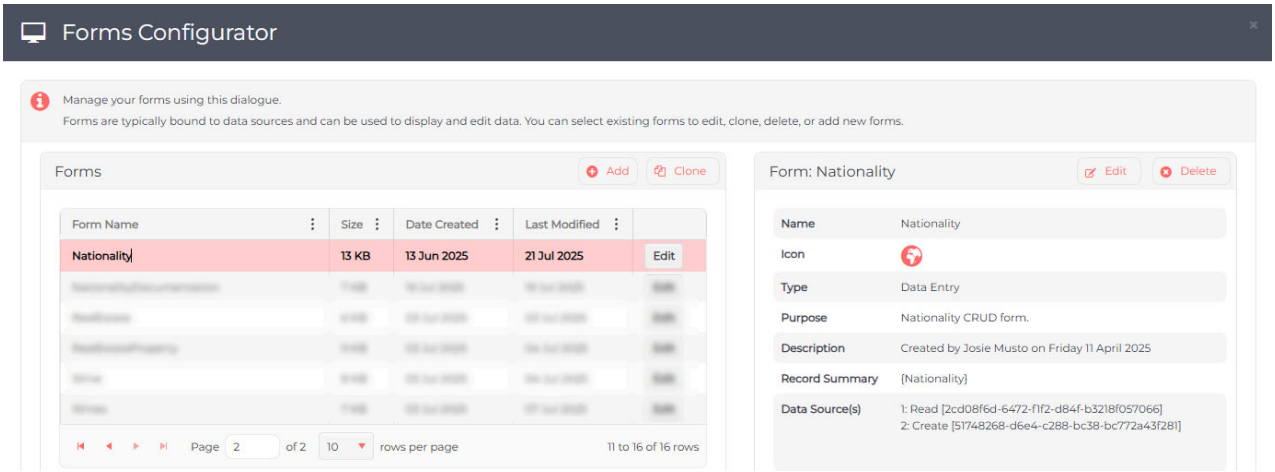
This Update data source request should now appear in the list of assigned data source requests:



If you ever need to remove a data source request from the list, you can use this button:

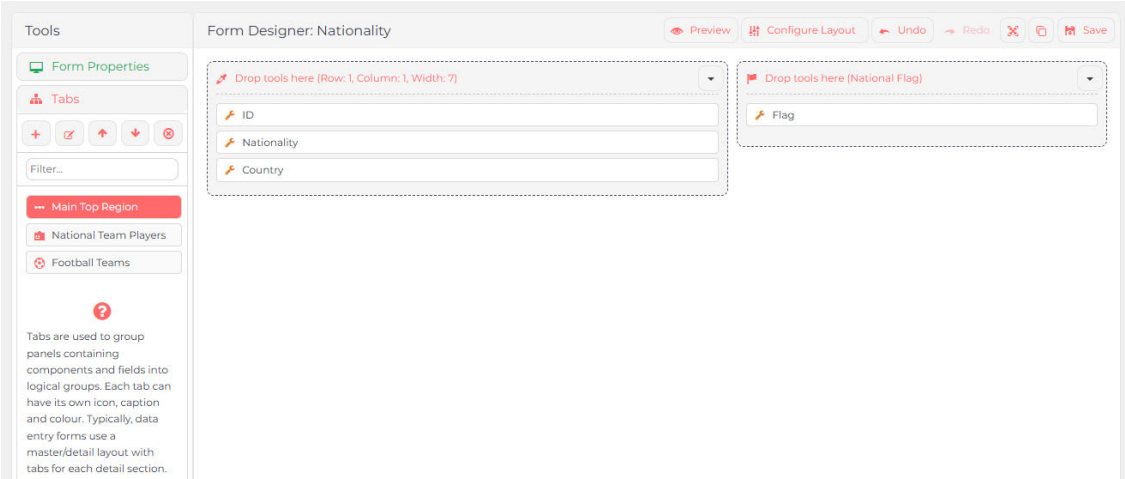


Now click the Apply button on this form, in order to persist the form properties before designing your form. The Update data source request you added should be shown in the properties list:



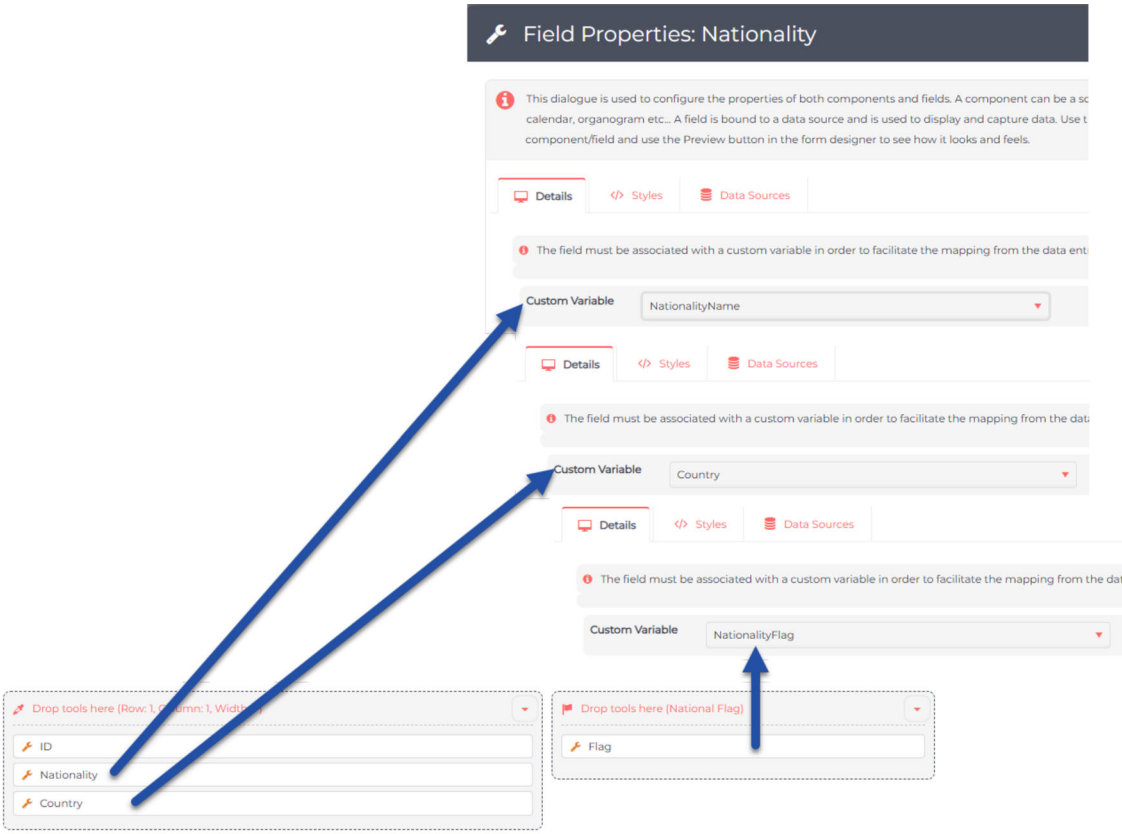
Form Design

Click one of the Edit buttons, either the grid row or properties panel. The form modal popup will display. Click the Design button, and select the Main Top Region tab:



Fields

Click on each of these fields in turn and assign the appropriate custom variable you added previously:



Save Form

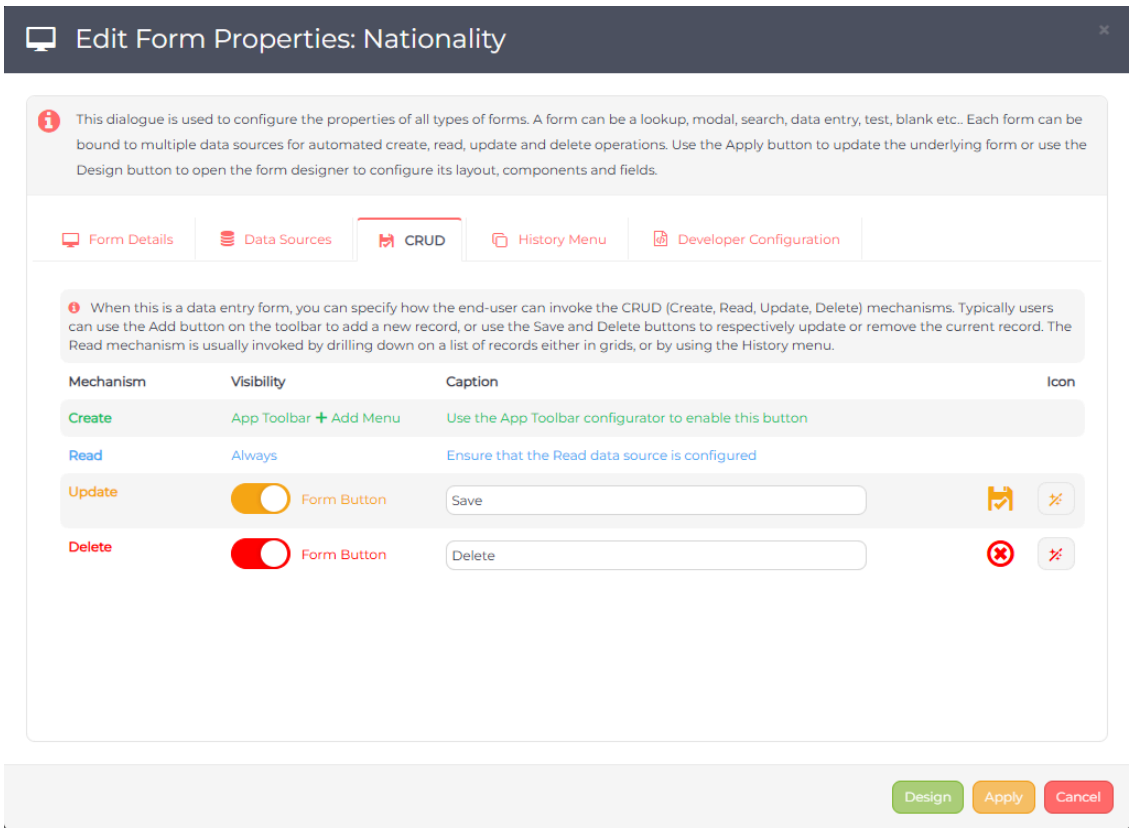
Now save the form design using this button:

The form fields are now assigned to the new custom variables used in the body of the new data source request to update the record.

We will now configure the update button on the form.

Configure Update Button

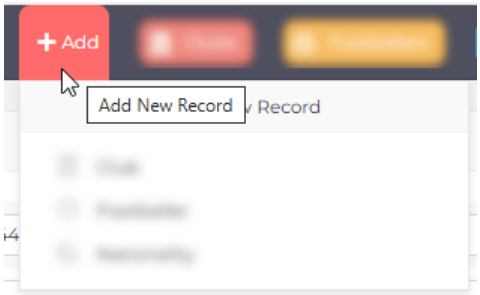
Open App Studio, open the Forms configurator, then select the Nationality form and click the edit button. Then click the CRUD tab:



There are four mechanisms to control: Create, Read, Update and Delete.

Create

A new record can only be created from the app toolbar using the **Add** drop down menu:



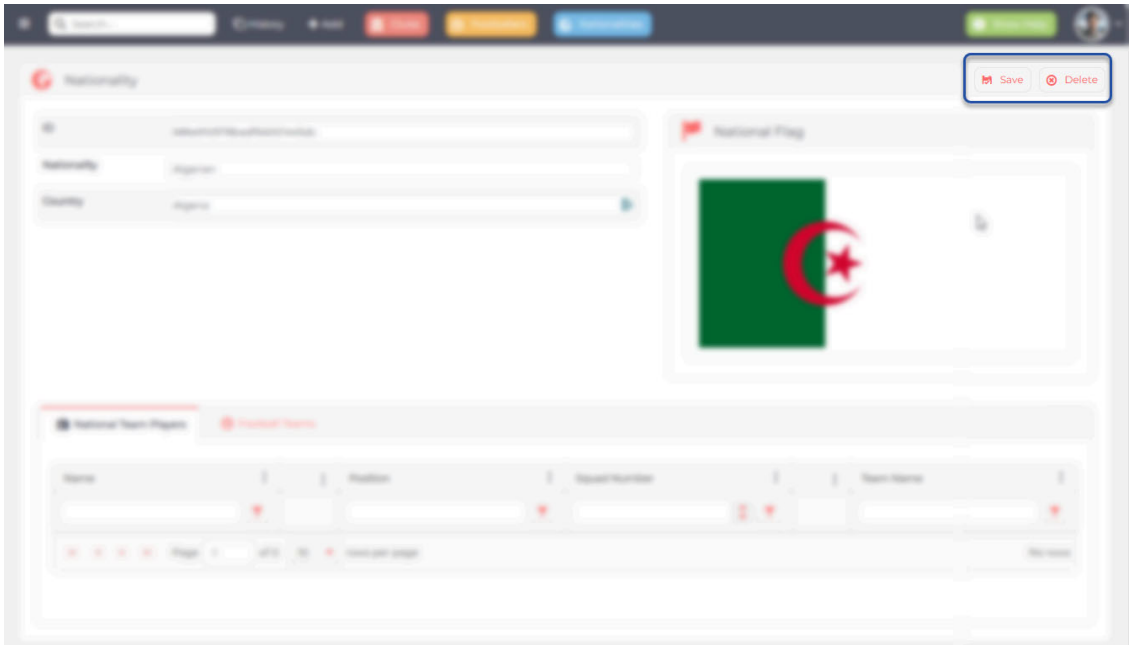
Use [this configurator]() to add any form to this list.

Read

Reading form data is always visible when a form is opened and a record is loaded.

Update

The update button lives together with the delete button top right on the form:



It can be hidden, its caption set and its icon set using these controls:

Delete

The delete button lives together with the update button top right on the form. It can be hidden, its caption set and its icon set using these same controls.

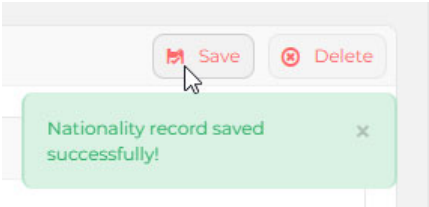
Apply

Apply any changes to persist them before testing.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Nationalities item. Drill down into any nationality to open the form.

Add an X to the end of the Nationality and Country fields, or indeed click the flag and upload a new flag image. Then press the **Save** button. The form record update should be confirmed:



Nationality Form: Create

Configure the nationality form to create a record.

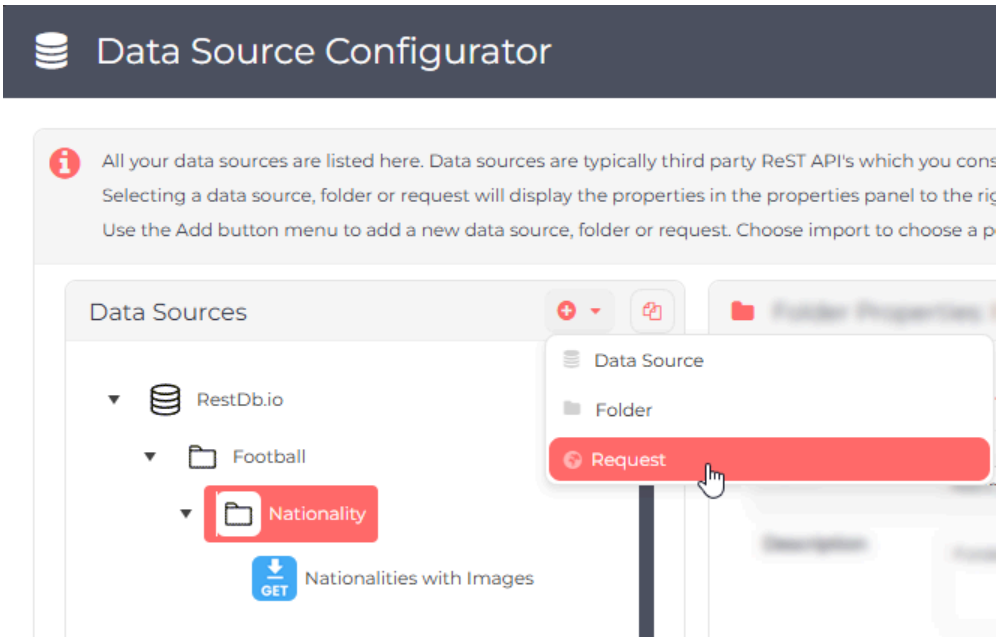
This is the process we will follow.

CREATE

- Add a 'create' data source for creating a single nationality record
- Link the key nationality record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'create' data source to the data entry form
- Configure the app toolbar Add menu to add this data entry form
- Test that we can create a nationality using the Add menu and Update button

Add a CREATE Data Source

Open App Studio, open the Data Sources configurator, select the Nationality folder you created previously, then use the add button menu to create a new request:

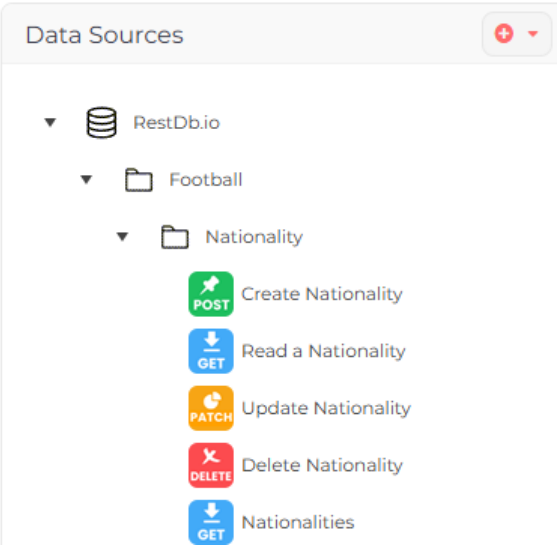


The Add New Request modal popup form shows asking you to name the request:

Type a meaningful name such as "Create Nationality" and click Save.

The request will be added to your tree view beneath the Nationality folder.

Use this opportunity to drag and drop these requests into CRUD order:



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

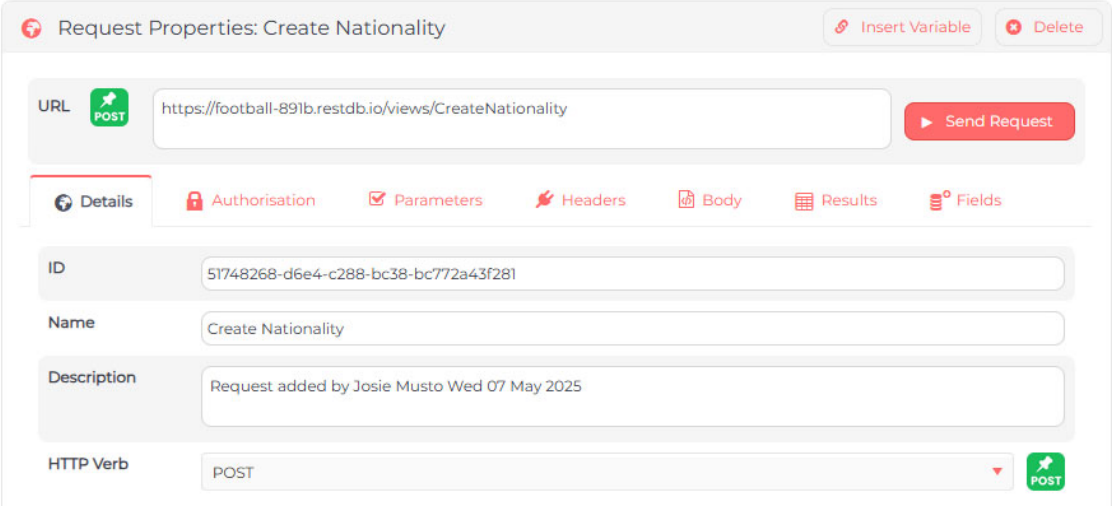
URL

`https://football-891b.restdb.io/views/CreateNationality` ↗

HTTP Verb

The default GET verb/method is not correct for creating a single record using the ReST API and should be set to POST for this specific back-end end-point.

The request should now look like this:



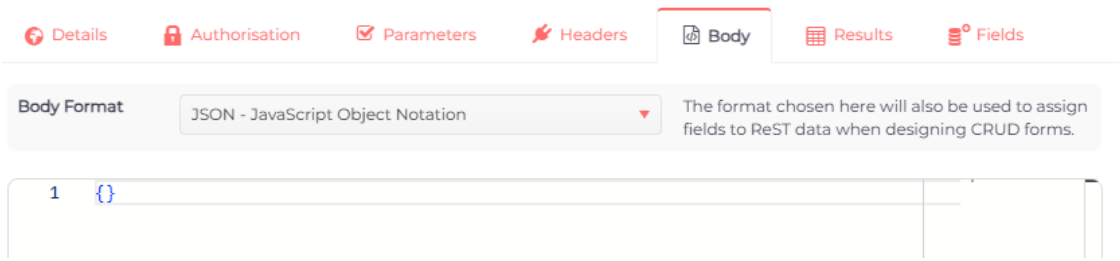
Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Body

We will be sending the data in the body of the request, so we configure using the relevant custom variables. Here is the empty Body tab:



We will always use the JSON body format as this gives us maximum control.

From the ReST API specification, we know that this endpoint expects data in the same format as the update request and indeed the same custom variables too:

```
{
    "Name": "<#NationalityName#>",
    "Country": "<#Country#>",
    "FlagURL": "<#NationalityFlag#>"
}
```

Send Request

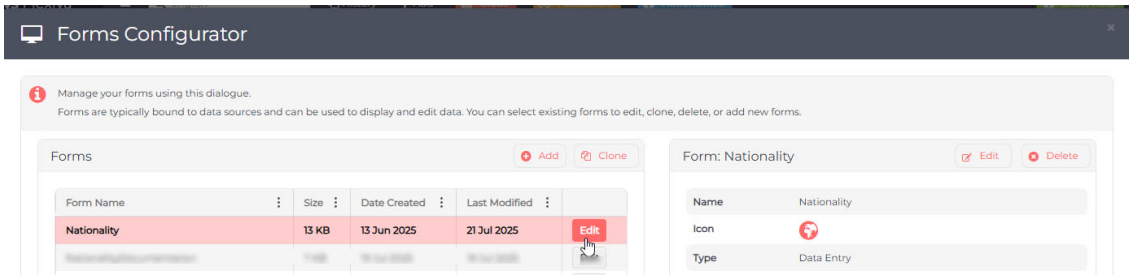
It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to create an existing record in order to test this.

The best way of course to test this is to use the actual nationality form we created here, and link that form to this data source request, then we can test it.

Edit Form Properties

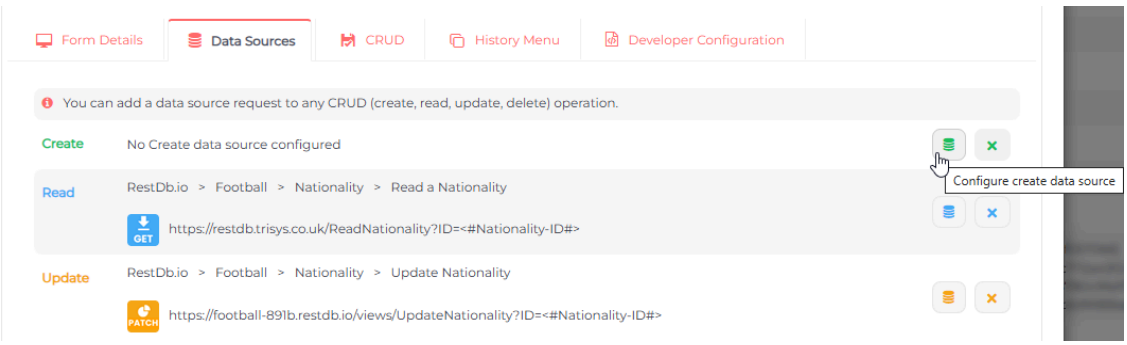
Open App Studio, open the Forms configurator, then select the Nationality form:



Click the Edit button to open the form properties modal popup.

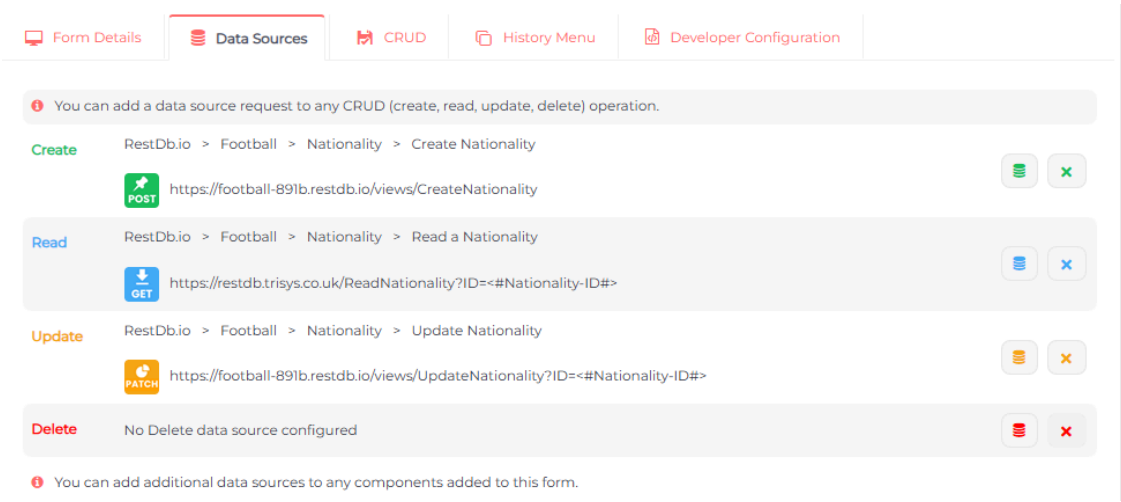
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Create data source:

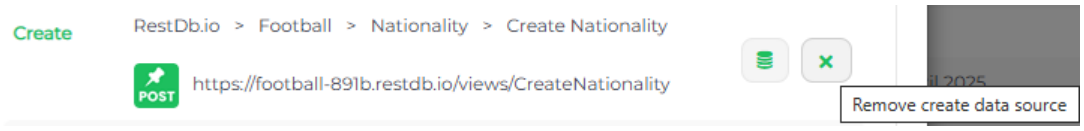


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the Select button.

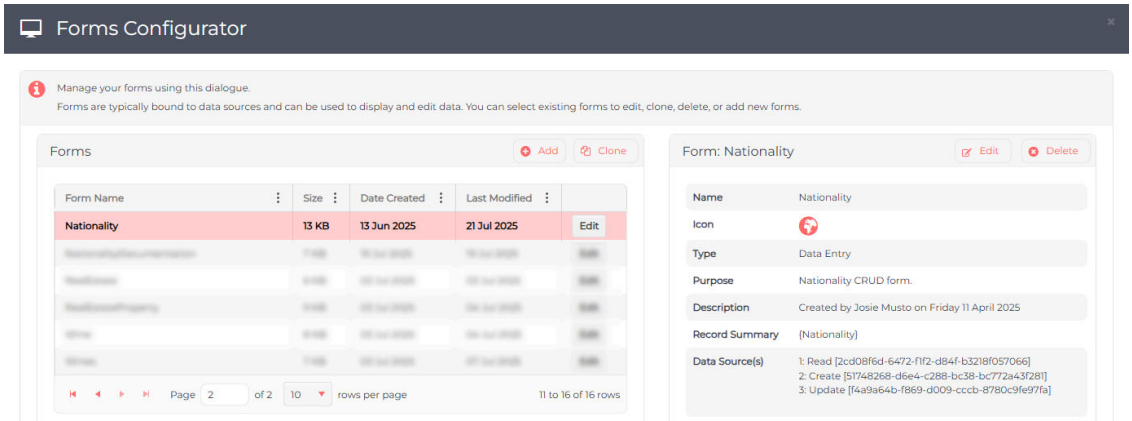
This Create data source request should now appear in the list of assigned data source requests:



If you ever need to remove a data source request from the list, you can use this button:



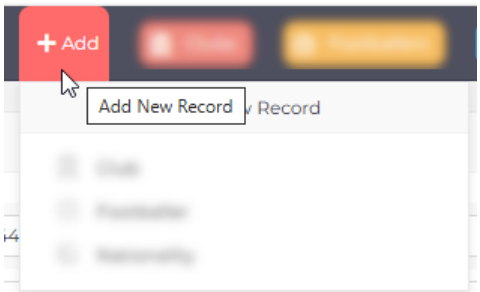
Now click the Apply button on this form, in order to persist the form properties. The Create data source request you added should be shown in the form properties list:



Note that in the previous section we assigned the custom variables to the form fields, so we do not need to do this again, as our new data source request uses the same custom variables. We also configured the update/save button, which will automatically call the new create data source method where necessary.

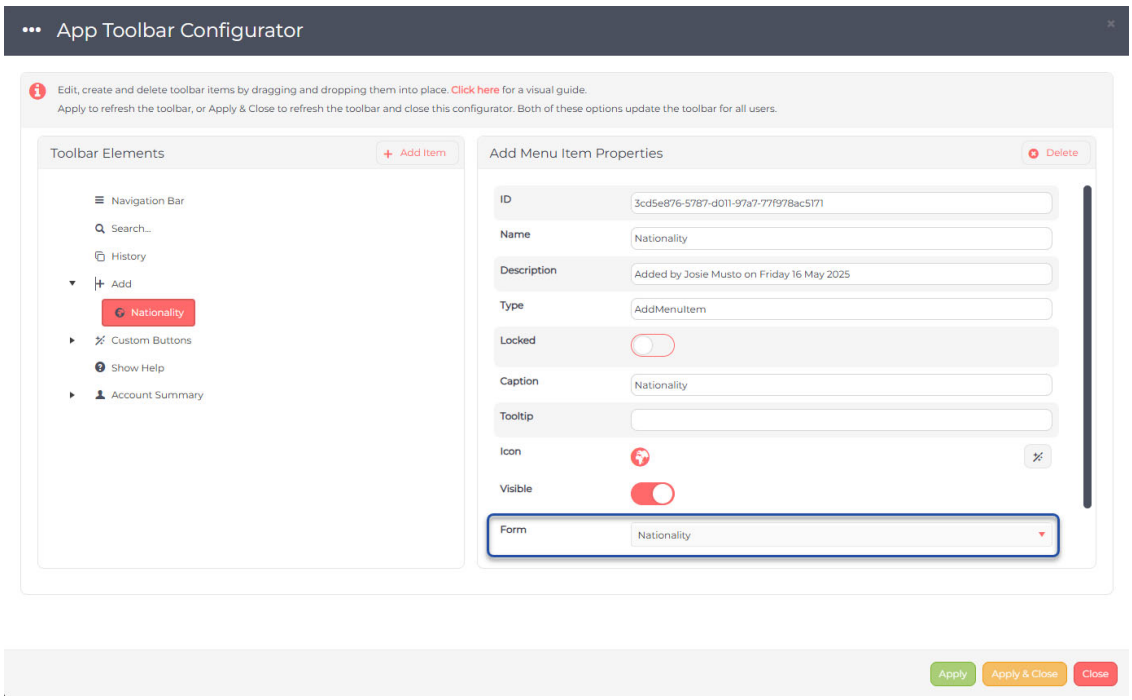
Configure Add Menu

A new record can only be created from the app toolbar using the **Add** drop down menu:



Use this configurator to add this form to this list.

This is what your **Add** menu should look like in the App Toolbar Configurator after you have created the Nationality form. Note how you will have assigned the form as shown:

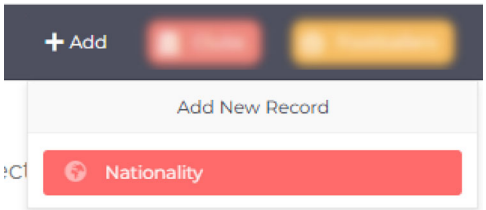


Apply

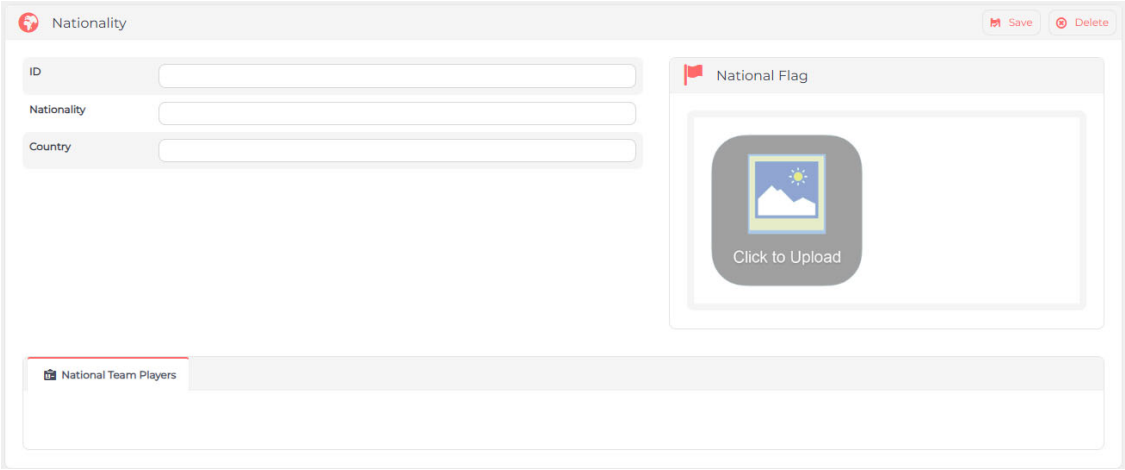
Apply any changes to persist them before testing.

Test

You can now test your configuration by clicking the **Add** menu button on the toolbar and selecting Nationality:



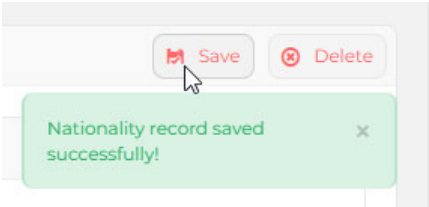
The Nationality form should open:



Type in a nationality and the respective country. Perhaps choose a fictitious country ↗ for testing?

Click the "Click to Upload" image to upload a flag image.

Press the Save button. The form record creation should be confirmed:



Nationality Form: Delete

Configure the nationality form to delete a record.

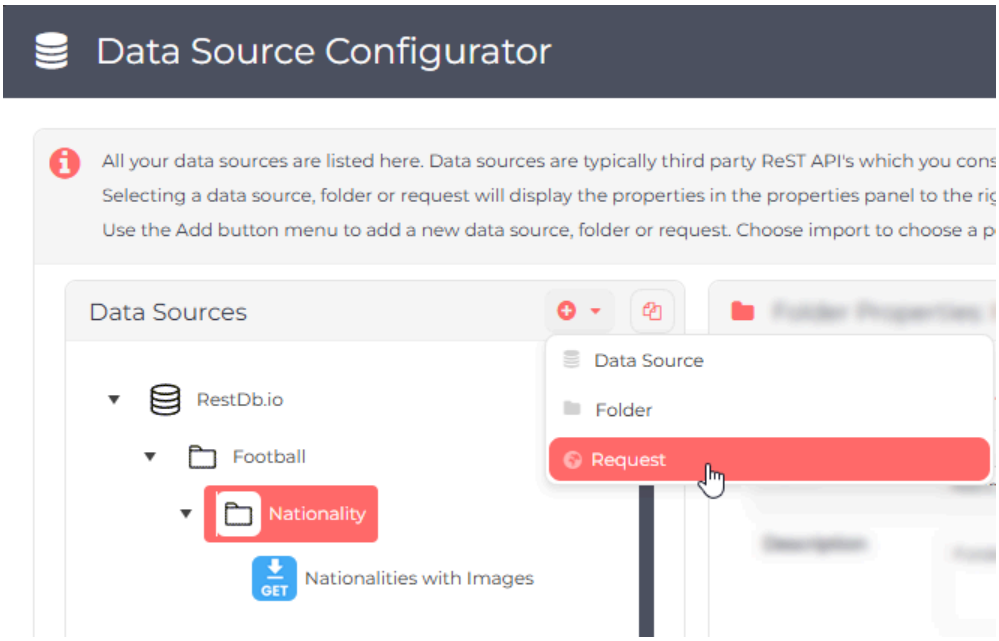
This is the process we will follow.

DELETE

- Add a 'delete' data source for deleting a single nationality record
- Link the key nationality record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'delete' data source to the data entry form
- Configure the delete button
- Test that we can delete a nationality using the Delete button

Add a DELETE Data Source

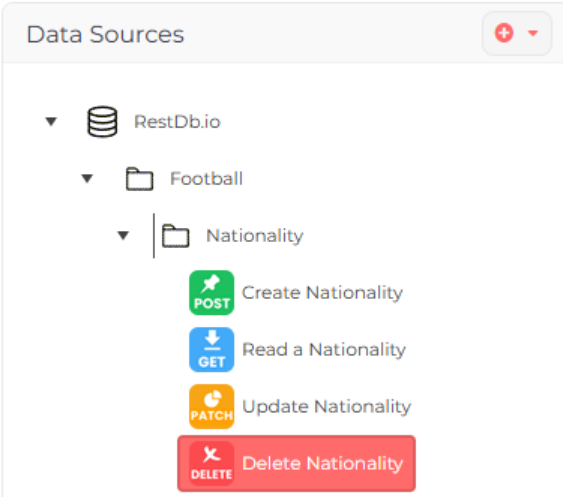
Open App Studio, open the Data Sources configurator, select the Nationality folder you created previously, then use the add button menu to create a new request:



The Add New Request modal popup form shows asking you to name the request:

Type a meaningful name such as "Delete Nationality" and click Save.

The request will be added to your tree view beneath the Nationality folder. Take this opportunity to drag and drop the order of the requests to fit the CRUD acronym:



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

URL

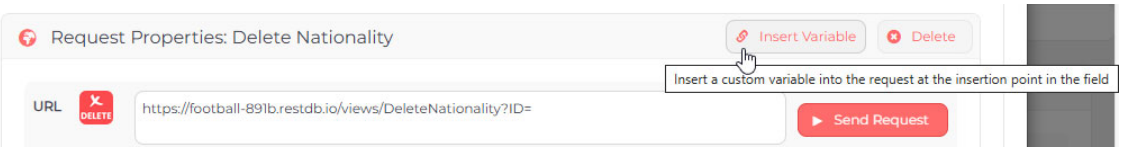
`https://football-891b.restdb.io/views/DeleteNationality` ↗

It is known from the documentation that this ReST API end-point has a nationality identifier property ?ID=.

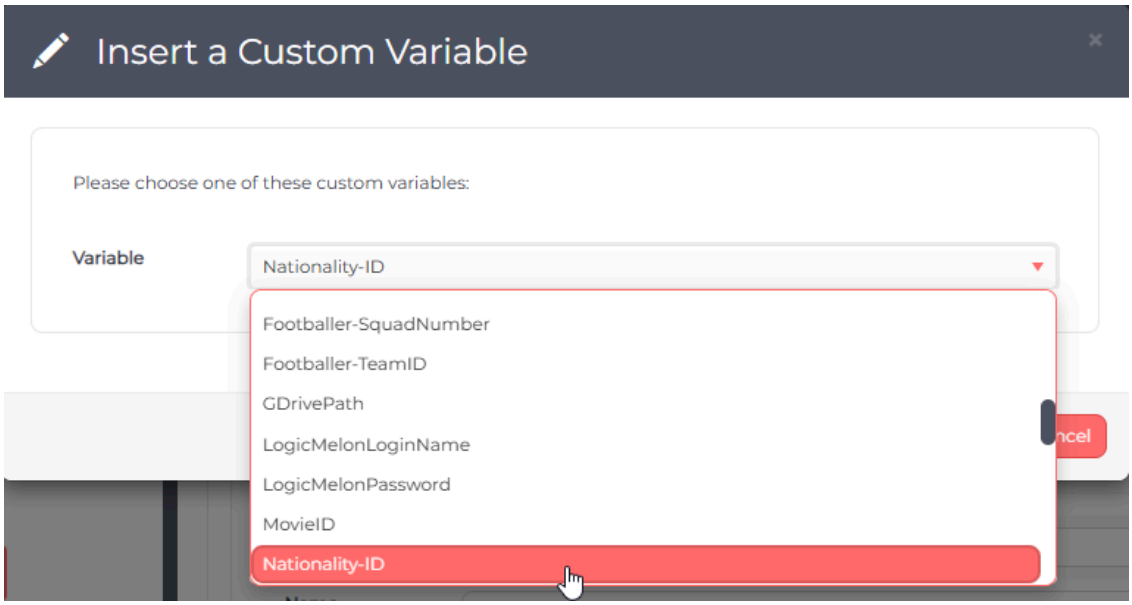
We therefore type this parameter list into the URL so that it reads:

`https://football-891b.restdb.io/views/DeleteNationality?ID=` ↗

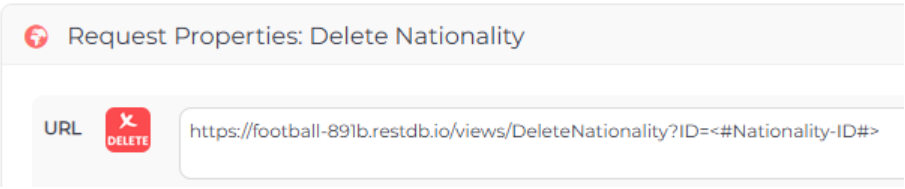
Whilst the caret is still blinking after the last character typed, click on the Insert Variable button:



This will open the following modal popup for where you should select the Nationality-ID custom variable you created previously in the Variable drop down list . Click the Select button.



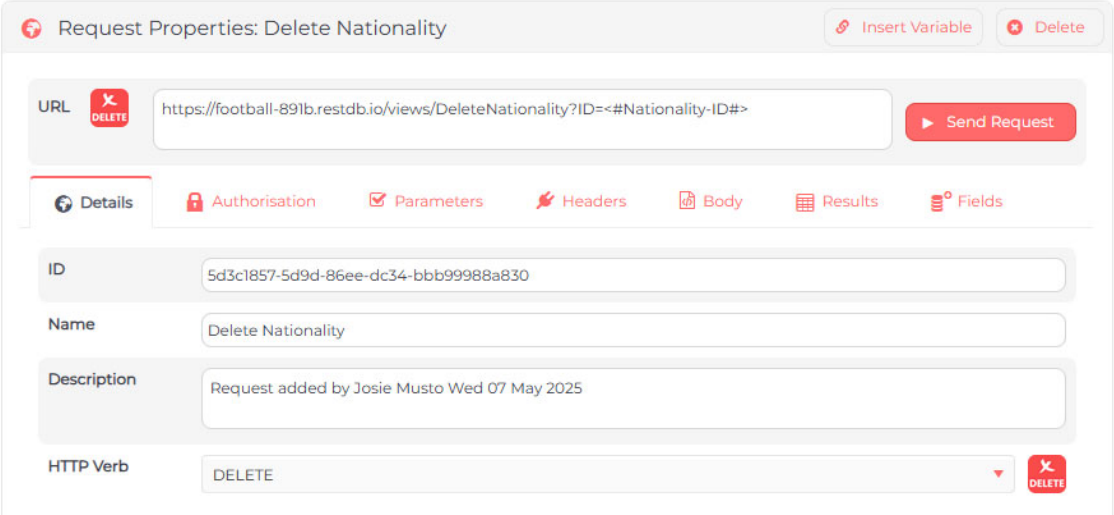
The URL should now show the custom variable appended to the end:



HTTP Verb

The default GET verb/method is not correct for deleting a single record using the ReST API and should be set to DELETE for this specific back-end end-point.

The request should now look like this:



Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Send Request

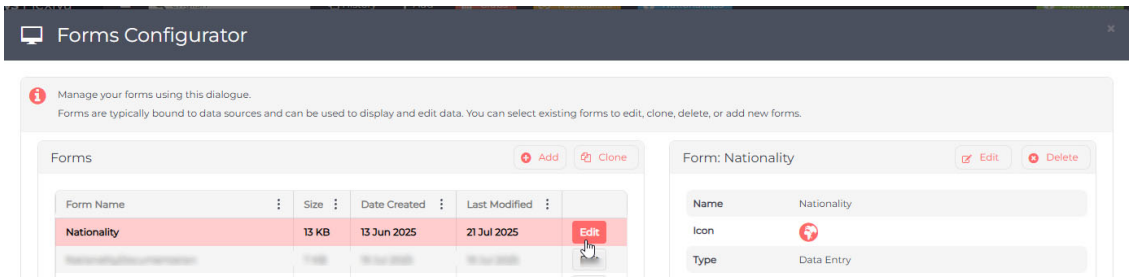
It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to delete an existing record in order to test this.

The best way of course to test this is to use the actual nationality form we created here, and link that form to this data source request, then we can test it.

Edit Form

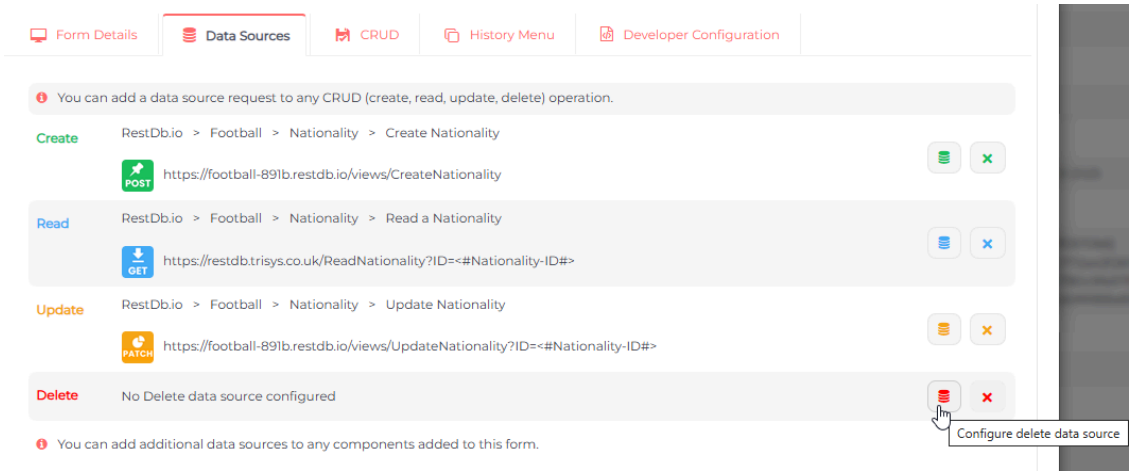
Open App Studio, open the Forms configurator, then select the Nationality form:



Click the Edit button to open the form properties modal popup.

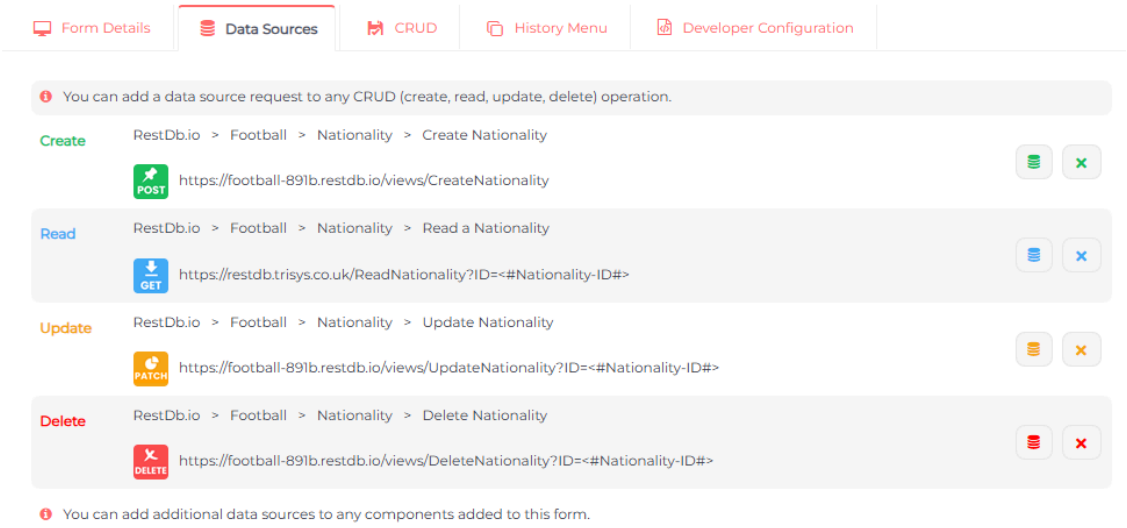
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Delete data source:

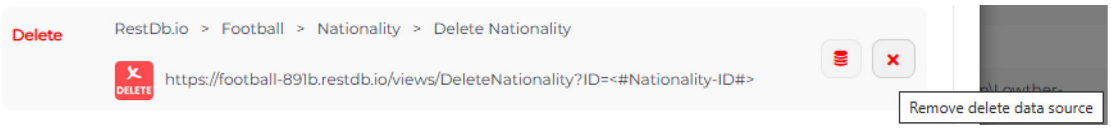


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the Select button.

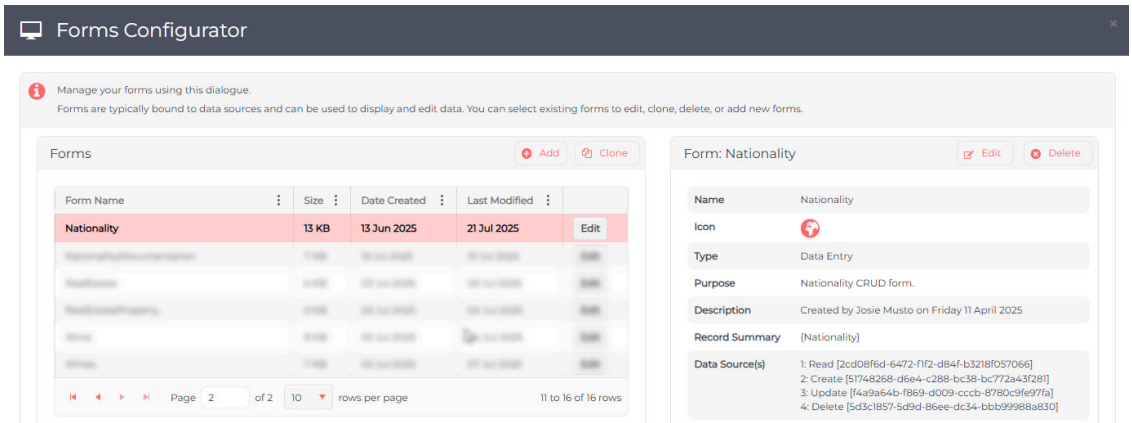
This Delete data source request should now appear in the list of assigned data source requests:



If you ever need to remove a data source request from the list, you can use this button:



Now click the Apply button on this form, in order to persist the form properties. The Delete data source request you added should be shown in the properties list:

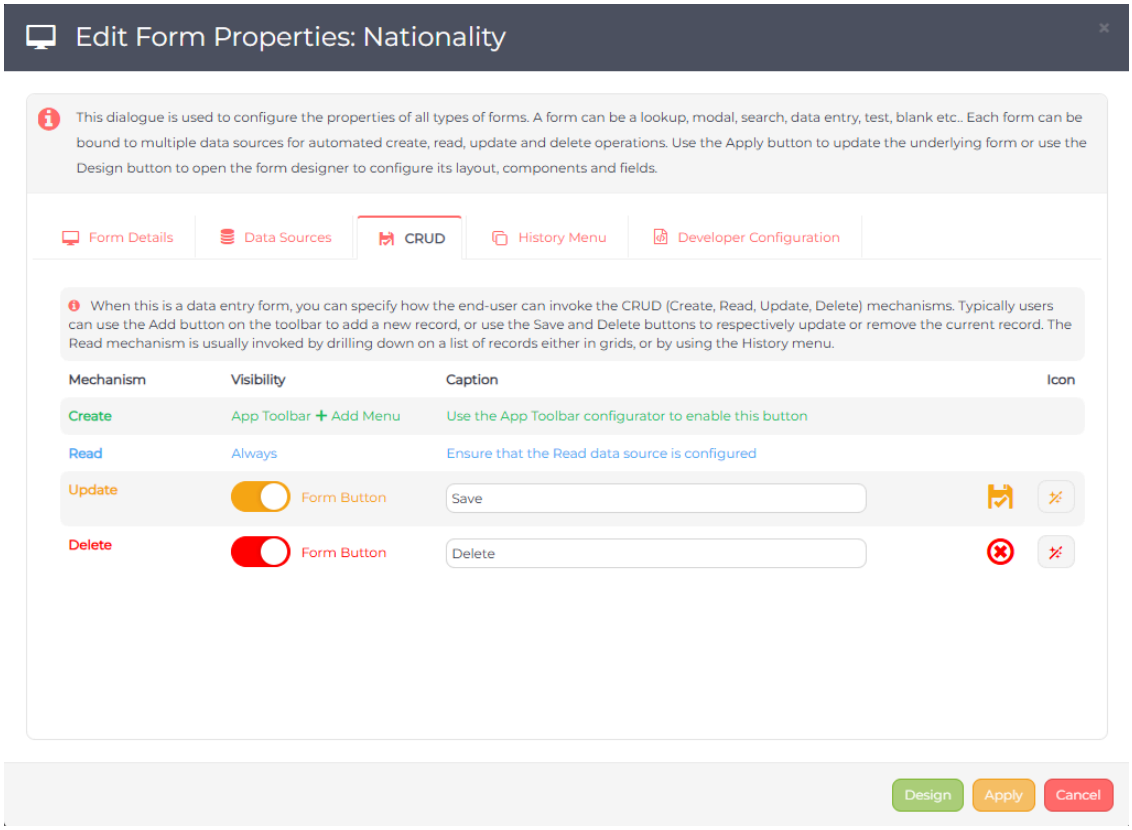


We do not need to design the form.

We will now configure the delete button on the form.

Configure Delete Button

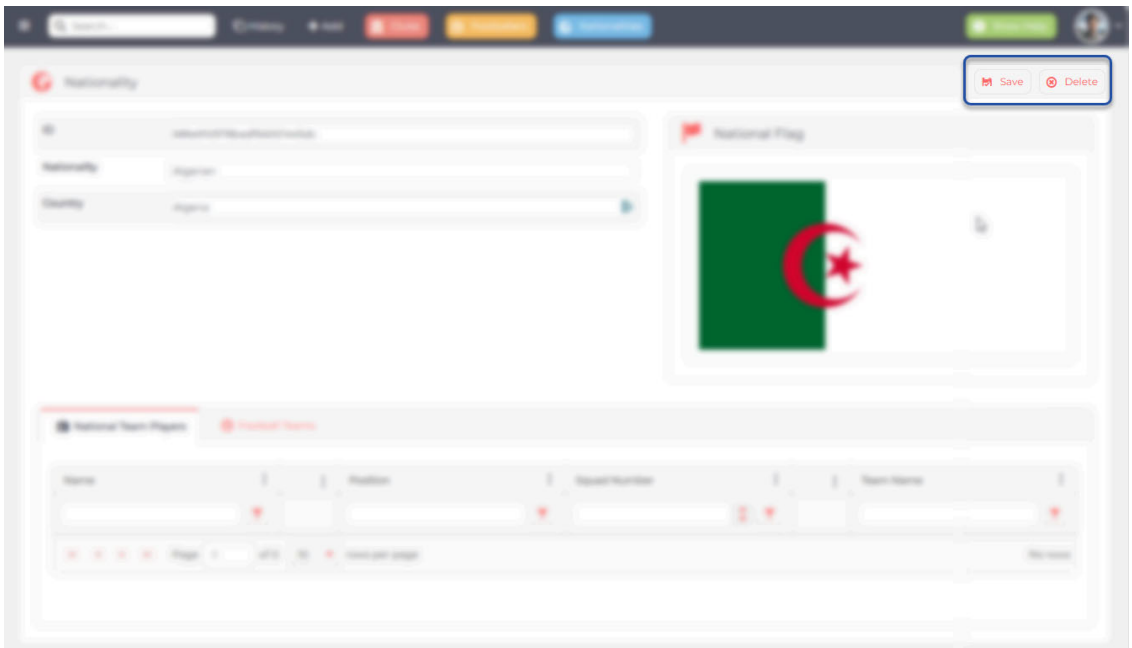
Open App Studio, open the Forms configurator, then select the Nationality form and click the edit button. Then click the CRUD tab:



There are four mechanisms to control: Create, Read, Update and Delete. We are only interested in Delete at this stage.

Delete

The delete button lives together with the update button top right on the form:



It can be hidden, its caption set and its icon set using these controls:

Apply

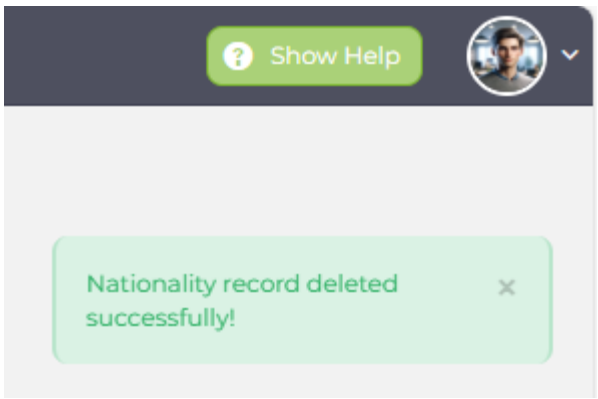
Apply any changes to persist them before testing.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Nationalities item. Drill down into the last test nationality you created to open the form.

Press the Delete button. You will be prompted to confirm the deletion.

The form record deletion should be confirmed when the form is closed:


Show Help
Nationality record deleted successfully!

Clubs/Teams Lookup Form

The second entity we will lookup is football clubs/teams.

A football club may in fact have numerous teams e.g. men and women, reserves, junior, even a pool or darts team 😂. To keep things simple in this sample database, there is a one-to-one relationship between a club and a team i.e. they are synonymous. In fact, the restdb.io sample table is called Teams.

We know from the ReST API what the end points and associated security keys we need to get started. We will always start by reading the data, then displaying it, before moving on to editing, creating and finally deleting data.

Custom Variables for Security Keys

We have already set this up here.

Custom Variables for Key Fields

Data source requests will return a data table of rows with fields. Each row will typically have an identifier for example in this sample the Club/Team ID. This will be a long random string of characters or numbers generated by the back-end database when records are created.

In order to handle the selection of rows, we need to create a custom variable which will be dynamically set when the end-user selects a specific record in a form or a grid or a field. We will assign this variable to the key field in the data source request.

Because we know that the restdb.io Rest API request will return teams, we will create a custom variable called "TeamID" which we will assign to the field later.

Add TeamID

Open App Studio, then open the custom variables configurator.

Select the Client in the drop down this time before using the Add button to create the custom variable `TeamID`.

We do not need any more custom variables to display and select clubs/teams in the lookup form, however we will when we need to design a data entry form in order to map fields to the ReST API body.

Read List

The next thing is to create the data source request to integrate the list of clubs/teams pulled from the ReST API.

Data Source Configurator

Open App Studio, then open the data source configurator.

Create a data source, and a sub-folder hierarchy like this:

`RestDB.io/Football/Teams`

This unambiguously defines that we have a restdb.io data source inside which we have the Football industry depicted as a folder. We then define a sub folder for Teams which will be where all the CRUD request methods for clubs/teams will be created.

Security

We previously set the security keys for the Football folder here so we do not need to do this again.

Teams Folder

Now that we have API security at the database level, we need to ensure that all requests inherit the same security. This is done by selecting the Teams folder beneath, clicking into the Authorisation tab and setting the Authorisation drop down combo to "Inherit from parent".

Every data source request we create in this folder will now inherit the security keys specified at the database level.

Create a Data Source Request

We can now create a data source request to read a list of teams from the ReST API.

Add Request

Click the add button menu item Request to create the following data source request:

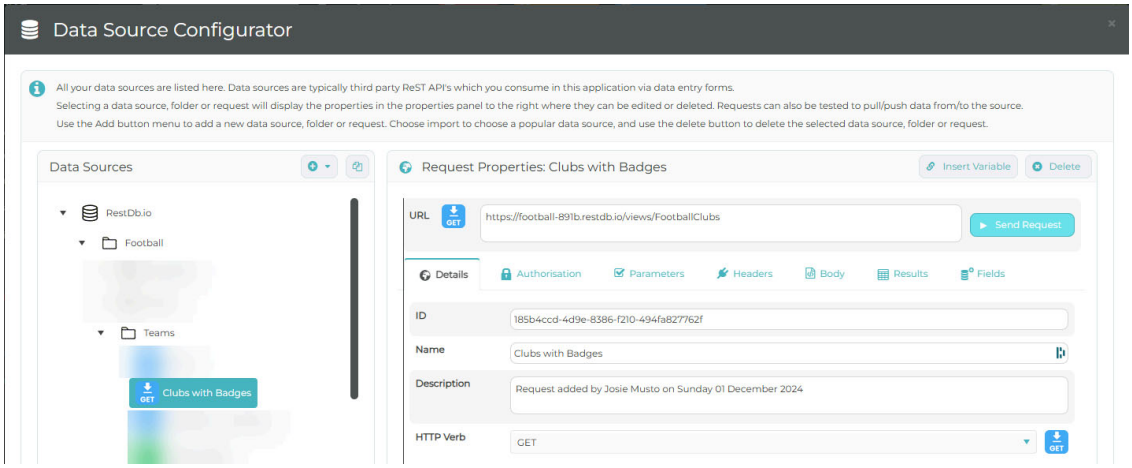
`RestDB.io/Football/Teams/Clubs with Badges`

Edit Request Properties

Edit the new request properties to set the URL to be:

`https://football-891b.restdb.io/views/FootballClubs` ↗

It should look like this:

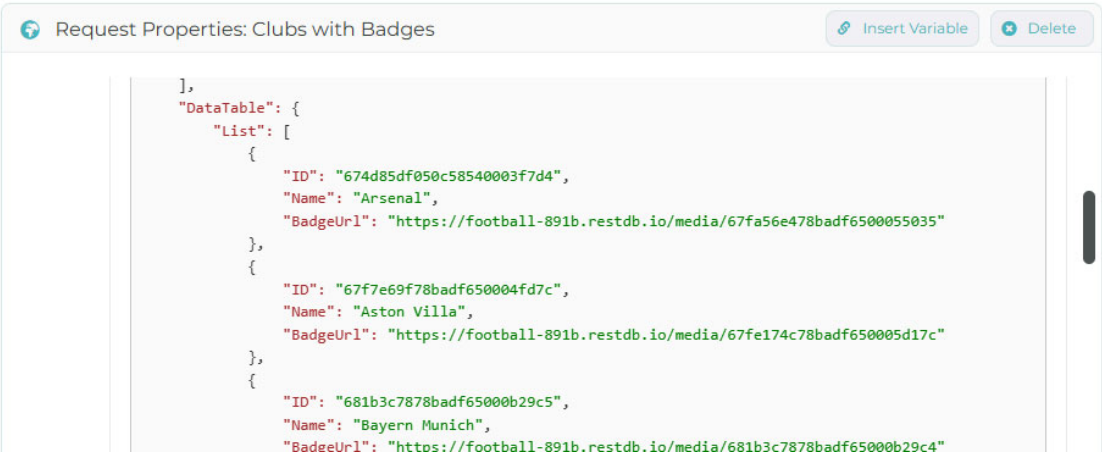


Send Request

You should now test the request by clicking the Send Request button. You will be prompted to confirm the URL being requested.

Results

After the ReST API request has been sent, the Results tab will be selected and the JSON tab will show the columns and this data:

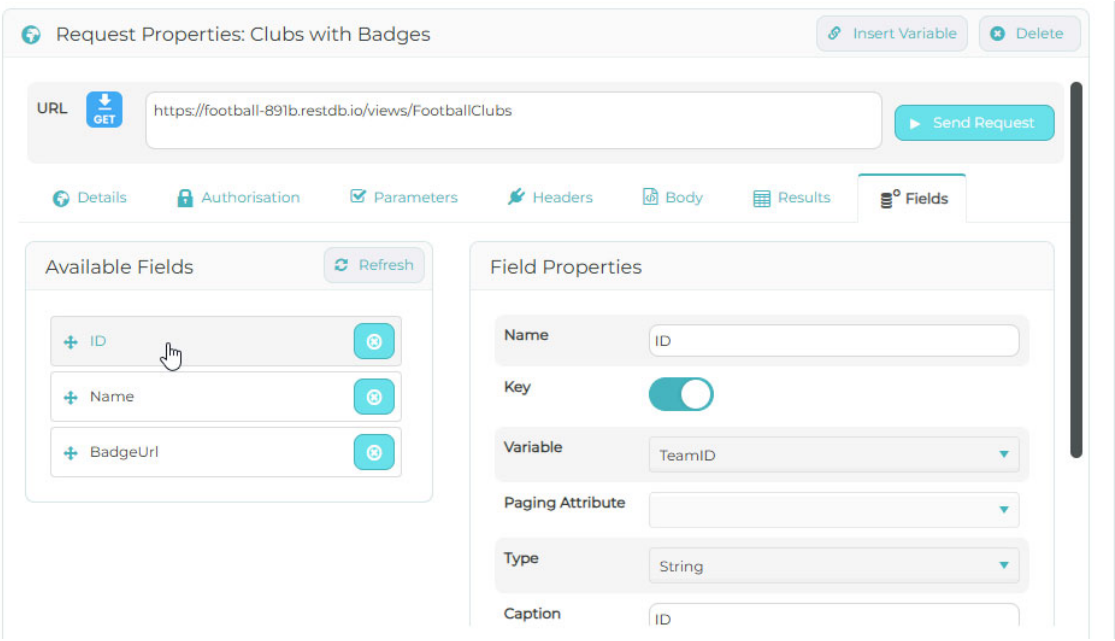


We can see that the clubs/teams data from the ReST API is being returned, including the badges as URL images. This is because our restdb.io view is able to map its media files onto a two-dimensional table for easy consumption by client-side applications.

Fields

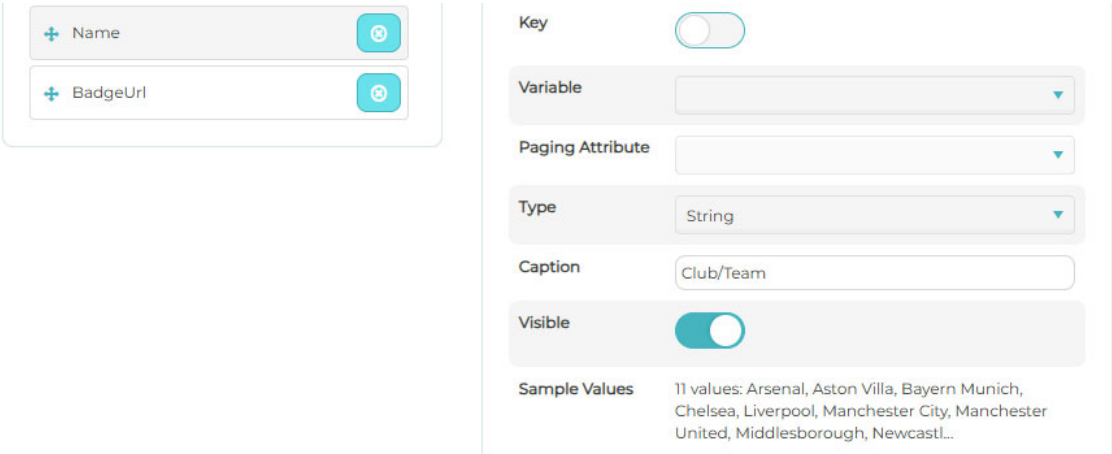
If we scroll back up to the top and click the left Fields tab, we should see a list of the fields accompanying the data.

Clicking on the top right Fields tab will show these Available Fields in a vertical list to the left:



Fields can be re-ordered using the left drag icon, or removed using the right delete button.

Selecting any field will show its corresponding properties to the right. Selecting the Name field and scrolling down should reveal the sample values:



These are useful to confirm that fields are returning the expected data.

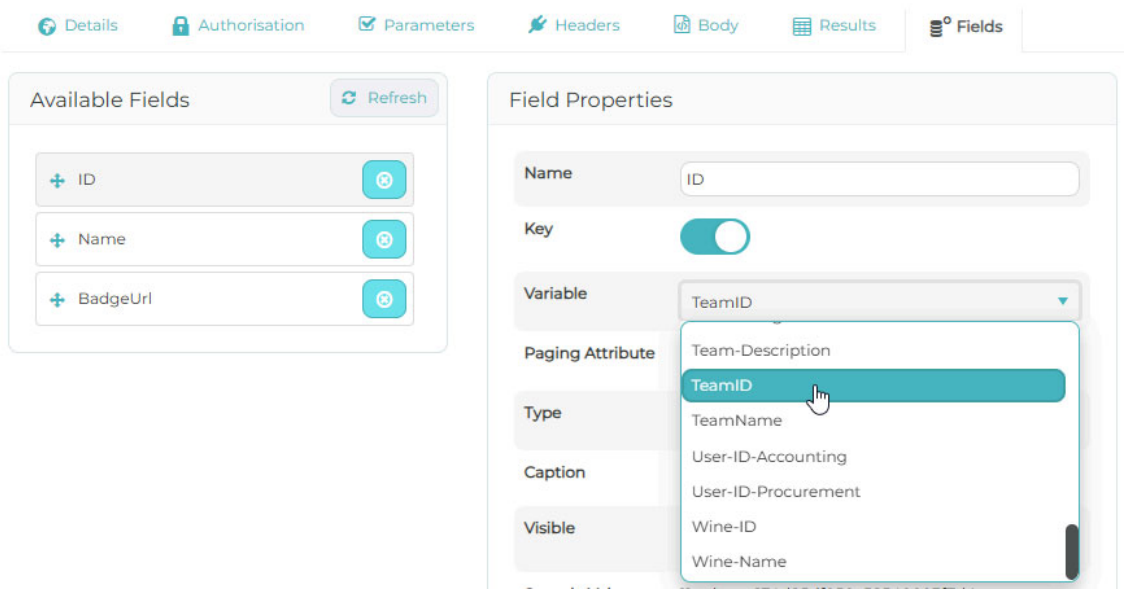
Image URL

When returning images, you should configure the type of field so that it displays correctly in form components such as grids.

Select the BadgeUrl field and set it's Type to "Image URL".

Key Field for Drill Down

Another important field to configure is the unique identifier or "Key" field. This is usually a long random string or number. We need to set the ID field to be the Key as well as assign it to the custom variable we created earlier:



This facilitates a process known as 'drill-down' where the end-user selected a hyperlinked column in a row and drills down into it by opening up the data entry form.

Note that we can now simply close the the data source configurator form as all changes are automatically persisted so we can move onto creating the lookup form.

Clubs/Teams Lookup Form

Now that we have the custom variables and data source request to facilitate listing clubs/teams, we need to create a lookup form upon which to display a grid showing all clubs.

Create Form

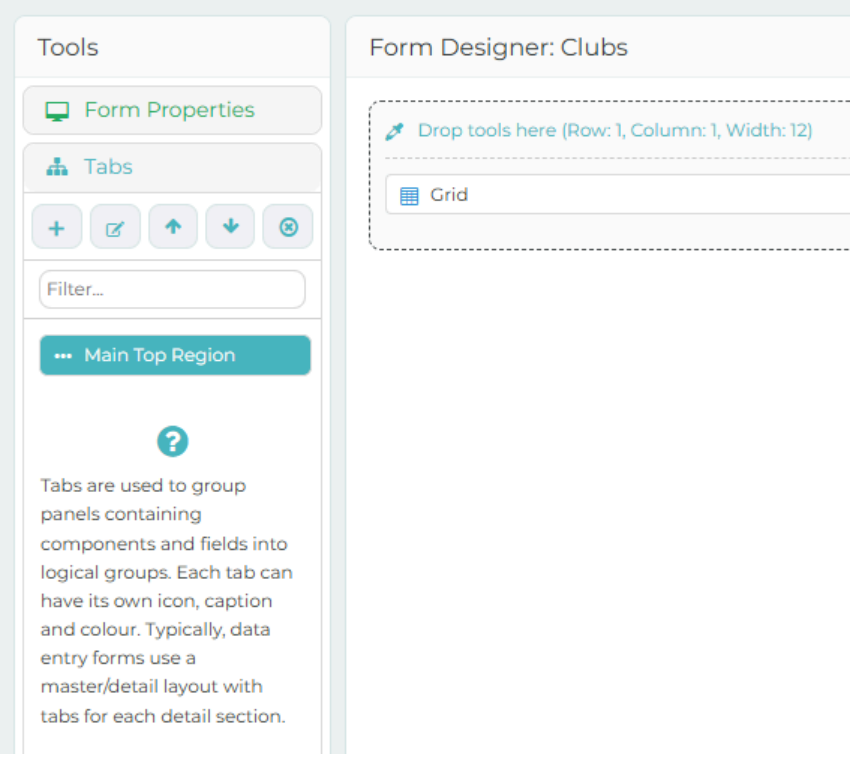
Open App Studio then select the Forms configurator.

Add

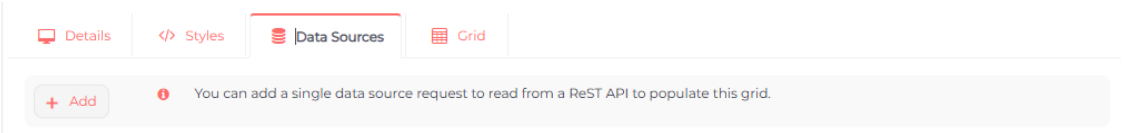
Use the Add button to create a new form called "Clubs" or "Club Lookup". In the form properties make sure that this form is of type "Lookup". You do not need to assign a data source to the form as we will do this in the form designer when we add a Grid component.

Design

Open form designer from the form properties popup, and drag a grid component into the first panel on the Main Top Region tab so that it looks like this:

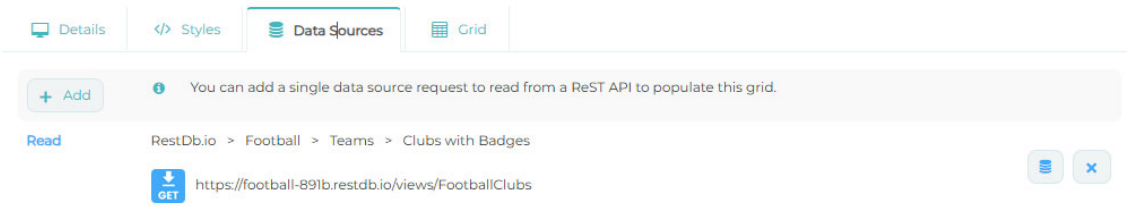


Click on the Grid to open the component properties for the grid. Select the Data Sources tab:



Use the Add button to select the data source request you created above.

After selection, the Read data source request should appear like this:



Grid Configuration

Select the Grid tab to configure the grid to display the data.

Population

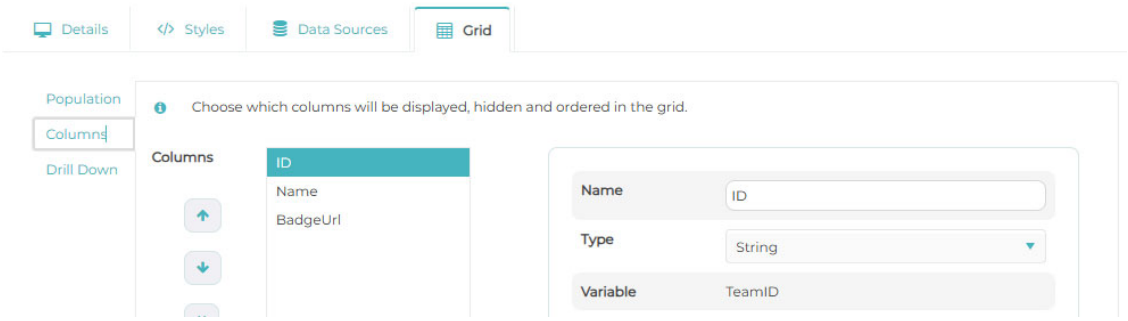
In the Population left tab, set the Population Trigger to Form Loaded. This tells the grid to load the data as soon as the form is displayed.

Columns

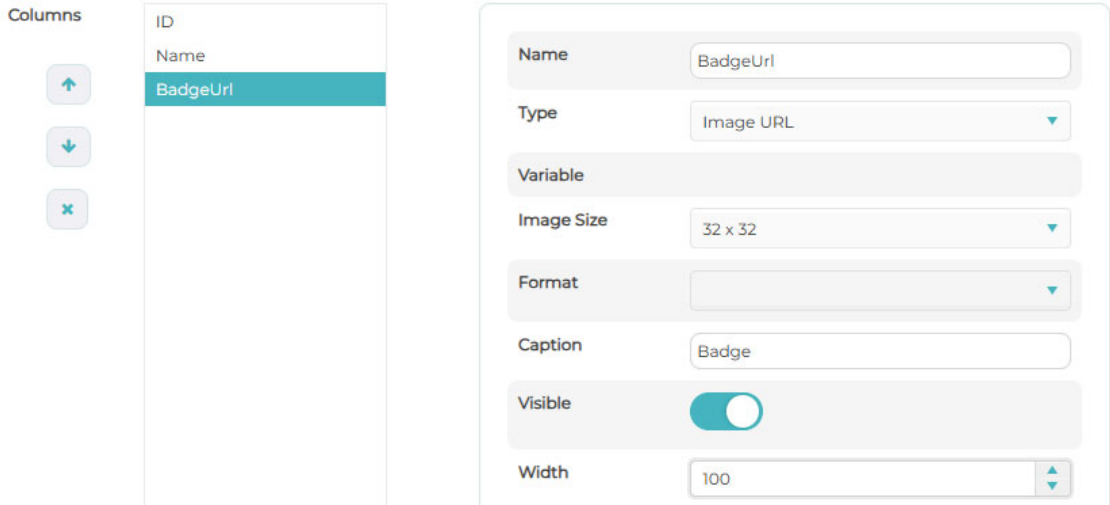
In the Columns left tab, the fields previously retrieved from the ReST API are displayed in the order you designed. You can re-order these by using the up/down arrows or delete a column.

Note that in the world of data, the term field is used, yet in grids, the word column is used. This is why the grid uses the column nomenclature instead of field.

Note that the ID key column is showing the custom variable you assigned to it previously:



For the BadgeUrl column, this should remember that it was configured as a Image URL Type. Set the Image Size to be 32 × 32 so that it looks the correct size when displayed. Also set the Caption, visibility and Width of the BadgeUrl column:



Drill Down

We will not configure the drill down tab until after we have created the data entry form in the next section.

Apply

Apply the grid properties, then Save the form design to persist the form for later use.

Navigation Bar

We can now add this lookup form to the navigation bar from where it can be opened by the end-user.

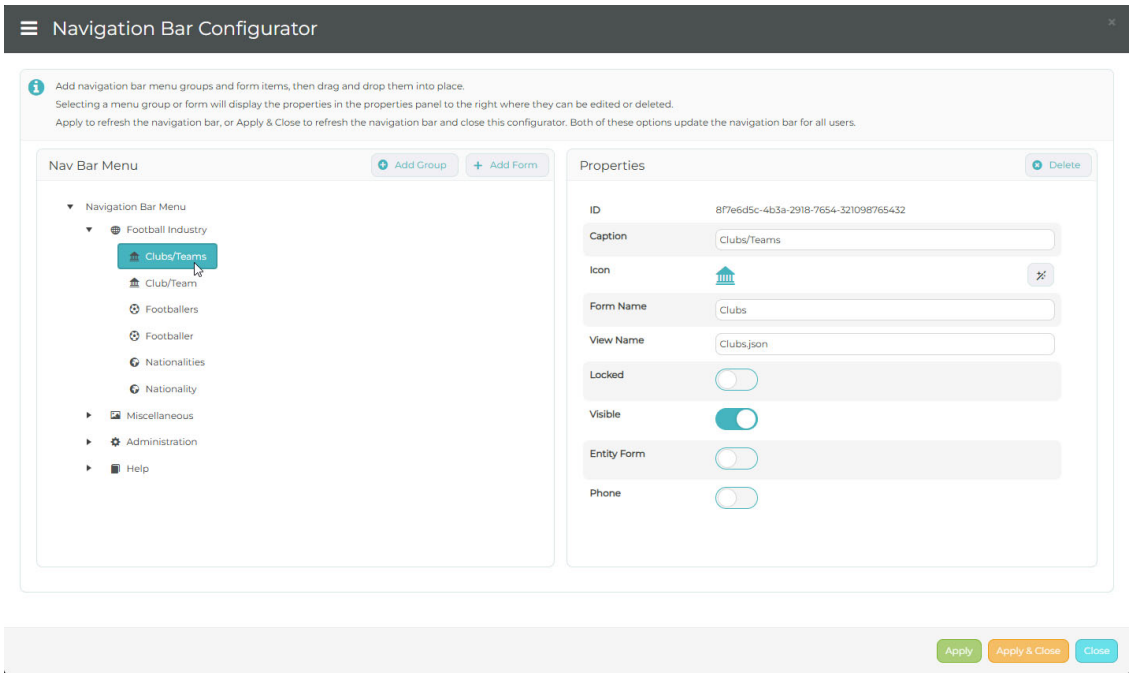
Open App Studio then select the Navigation Bar configurator.

Football Industry

The group called "Football Industry" was already created here.

Clubs/Teams

When the Football Industry group is selected, add a form using the Add Form button. This will popup a modal dialogue where you can select your newly created Clubs form. Clicking Save will show the new Clubs/Teams form menu:

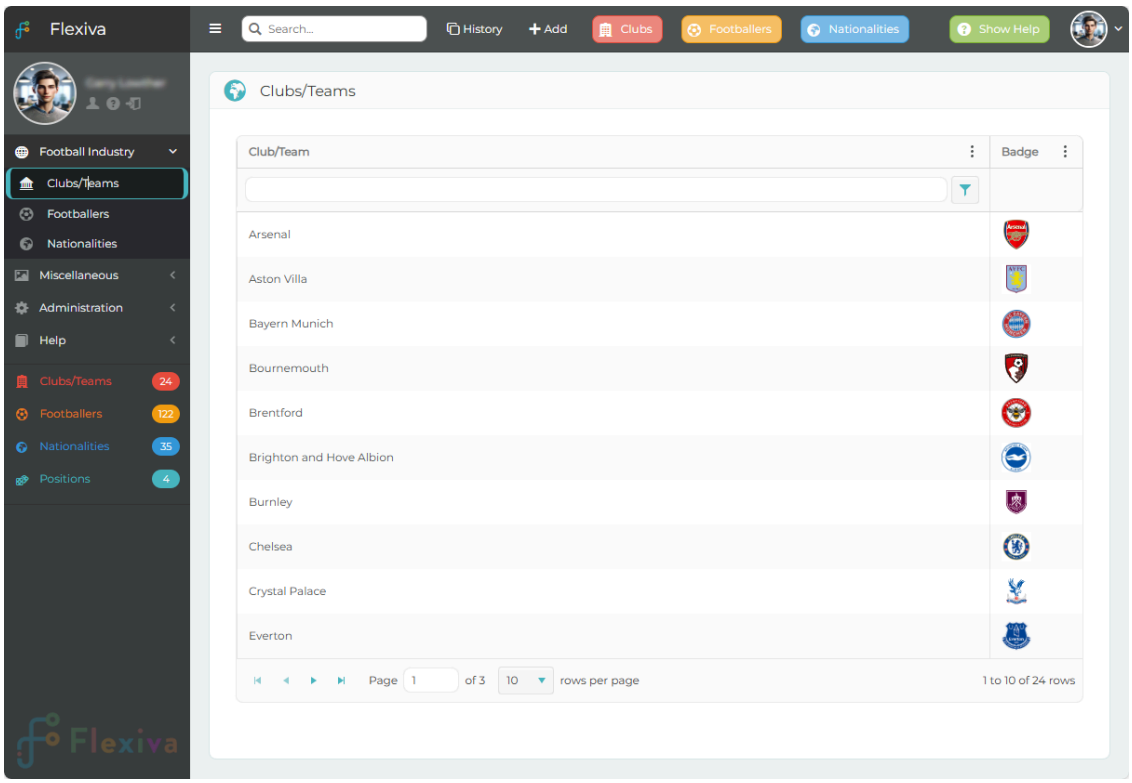


Select Visible, and change the Icon to a building. Then press the Apply & Close button.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Clubs/Teams item.

You should now see the lookup form with the data from the ReST API displayed in a grid:



Test that column filtering and sorting works as expected.

We are now ready to [create a data entry form]() configured for creating, reading, updating and deleting cub/team records.

Club/Team Data Entry Form

The second entity data entry form is a club/team record.

Having previously created custom variables and a data source request to display a lookup form showing, filtering and sorting clubs, we are now moving on to creating a data entry form where a club can be created, read, updated and deleted (CRUD).

In our ReST API sample data set, the underlying restdb.io club is the `Teams` table.

The process for all CRUD operations typically starts with READ, as this involves designing the data entry form, and drilling down into it. Here are the four CRUD phases in the order we will configure them:

READ

- Add a 'read' data source for reading a single club/team record
- Link the key club/team record identifier to this ReST API end-point if necessary
- Create a data entry form
- Assign this 'read' data source to the data entry form
- Design this form and drag fields from this data source
- Add this form to the navigation bar
- Configure drill down to the lookup form grid component
- Test that we can lookup clubs/teams and drill down into a data entry form showing the club master record
- Set the History Menu record summary
- Test that we can lookup clubs and drill down into the club/team form, and that the history menu shows the club name
- Add a master/detail grid to show all footballers who play for this club/team
- Test that we can view all footballers who belong to this club/team

UPDATE

- Create custom variables for each field
- Add an 'update' data source for updating a single club/team record
- Link the key club record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'update' data source to the data entry form
- Configure the update button
- Test that we can update a club/team using the Update button

CREATE

- Add a 'create' data source for creating a single club record
- Link the key club record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'create' data source to the data entry form
- Configure the app toolbar Add menu to add this data entry form
- Test that we can create a club/team using the Add menu and Update button

DELETE

- Add a 'delete' data source for deleting a single club record
- Link the key club record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'delete' data source to the data entry form
- Configure the delete button
- Test that we can delete a club/team using the Delete button

Club/Team Form: Read

Add a new club/team form to read and display a record.

This is the process we will follow.

READ

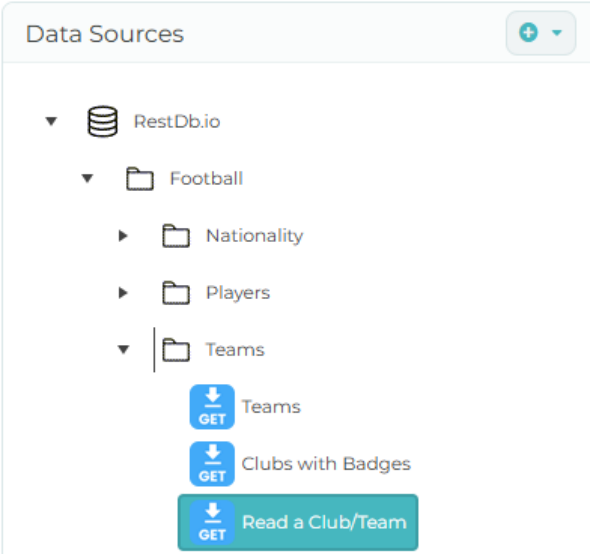
- Add a 'read' data source for reading a single club/team record
- Link the key club/teamrecord identifier to this ReST API end-point if necessary
- Create a data entry form
- Assign this 'read' data source to the data entry form
- Design this form and drag fields from this data source
- Add this form to the navigation bar
- Configure drill down to the lookup form grid component
- Test that we can lookup clubs/teams and drill down into a data entry form showing the club master record
- Set the History Menu record summary
- Test that we can lookup clubs and drill down into the club/team form, and that the history menu shows the club name
- Add a master/detail grid to show all footballers who play for this club/team
- Test that we can view all footballers who belong to this club/team

Add a READ Data Source

Open App Studio, open the Data Sources configurator, select the Teams folder you created previously, then use the add button menu to create a new request.

The Add New Request modal popup form shows asking you to name the request. Type a meaningful name such as "Read a Club/Team" and click Save.

The request will be added to your tree view beneath the Teams folder:



Edit Properties

Edit the properties in the Details tab as following by referencing this list of ReST API end-points.

URL

`https://restdb.trisys.co.uk/ReadFootballClub`

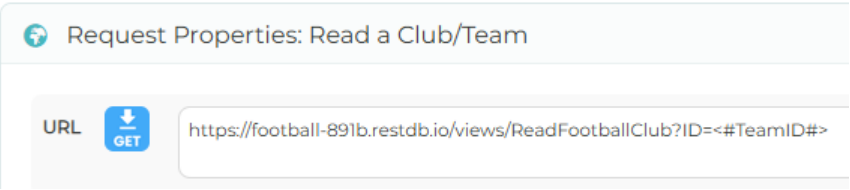
Now it is known from the documentation that this ReST API end-point has a club/team identifier property ?ID=.

We therefore type this parameter list into the URL so that it reads:

`https://restdb.trisys.co.uk/ReadNationality?ID=`

Whilst the caret is still blinking after the last character typed, click on the Insert Variable button and select this variable: `TeamID`

The URL should now show the custom variable appended to the end:



HTTP Verb

The default GET verb/method is usually correct for reading a single record from a ReST API and is correct for this specific back-end end-point.

Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Send Request

Click this button on the Details tab to send the request to the ReST API.

You will be prompted to confirm the URL. The value of the custom variable `TeamID` should show after ?ID= but if it does not, then copy this value in: `686f8e0e78badf6500145951` as this is the ID for Bournemouth.

This URL should now be:

`https://football-891b.restdb.io/views/ReadFootballClub?ID=686f8e0e78badf6500145951` ↗

Press the Confirm Request URL button.

The request should run quickly and select the Results tab and show the JSON top left tab displaying the full JSON returned from the ReST API:

```
{
    "Columns": [
        {
            "field": "ID",
            "title": "Id",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": null
        },
        {
            "field": "Name",
            "title": "Name",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "Description",
            "title": "Description",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": "<img src='#: Description #' style='width: 64px;
height: 64px;' />"
        },
        {
            "field": "Badge",
            "title": "Badge",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "BadgeUrl",
            "title": "Badgeurl",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": "<img src='#: BadgeUrl #' style='width: 64px;
height: 64px;' />"
```

```
        }
    ],
    "DataTable": {
        "List": [
            {
                "ID": "686f8e0e78badf6500145951",
                "Name": "Bournemouth",
                "Description":
"https://www.premierleague.com/en/clubs/91/bournemouth/overview",
                "Badge": "686f8e0e78badf6500145950",
                "BadgeUrl": "https://football-
891b.restdb.io/media/686f8e0e78badf6500145950"
            }
        ],
        "DynamicColumns": null,
        "TotalRecordCount": 0,
        "TotalPageCount": 0,
        "FirstRowNumber": 0,
        "LastRowNumber": 0,
        "PageNumber": 1,
        "RecordsPerPage": 1,
        "SortColumnName": null,
        "SortAscending": true,
        "AICriteria": null,
        "Success": false,
        "ErrorMessage": null
    },
    "URL": "https://football-891b.restdb.io/views/ReadFootballClub?
ID=686f8e0e78badf6500145951",
    "Verb": "GET",
    "Success": true,
    "ErrorMessage": null
}
```


Look at the `DataTable -> List` to see that it contains 1 club/team record.

Our data source request has now been created and configured, so we can now close this configurator.

Create a Data Entry Form

Open App Studio, then open the Forms configurator.

Add

Click the Add button to open the modal popup form.

Name

Type the name "Club". You do not need the word "Form" to follow it.

Purpose

Type "Football Club/Team form"

Type

This must be set to "Data Entry".

Icon

Use the far right button to choose a suitable icon for the form. Use the text "build" in the filter to find a building.

Caption

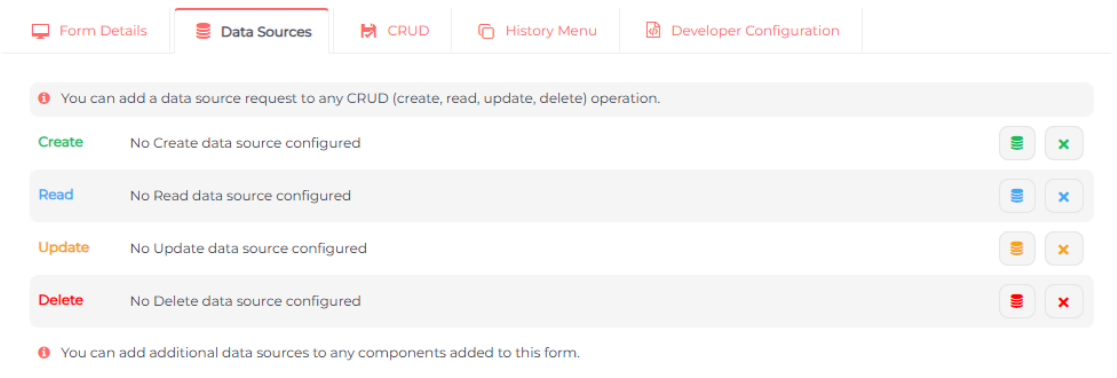
Set the caption to "Club/Team". Note that we will configure the history menu to show the club/team name, not this caption.

Description

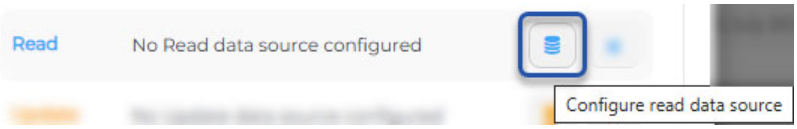
The description will be automatically generated which is fine for now.

Data Sources

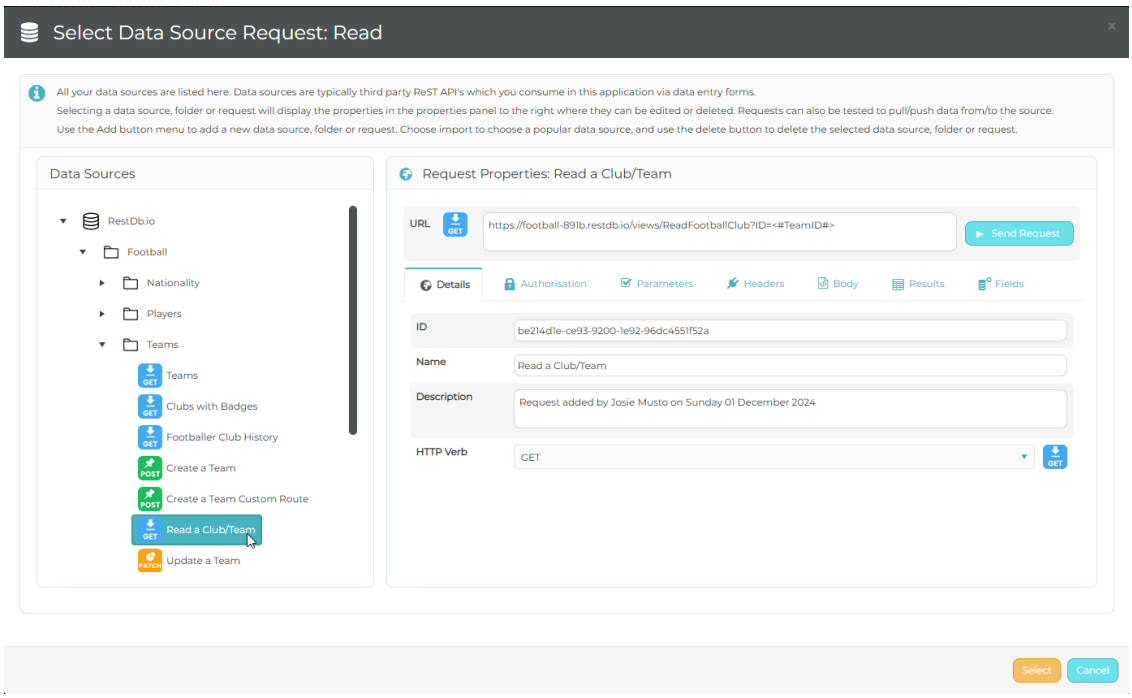
Click the Data Sources tab as this is where we will add the READ data source we created earlier:



There are 4 CRUD data source lines. Click this database icon for the Read data source:

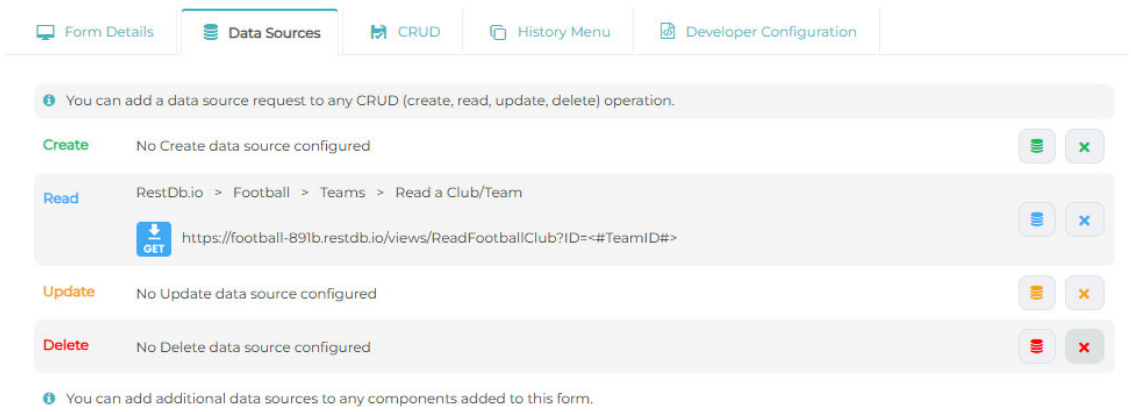


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request:



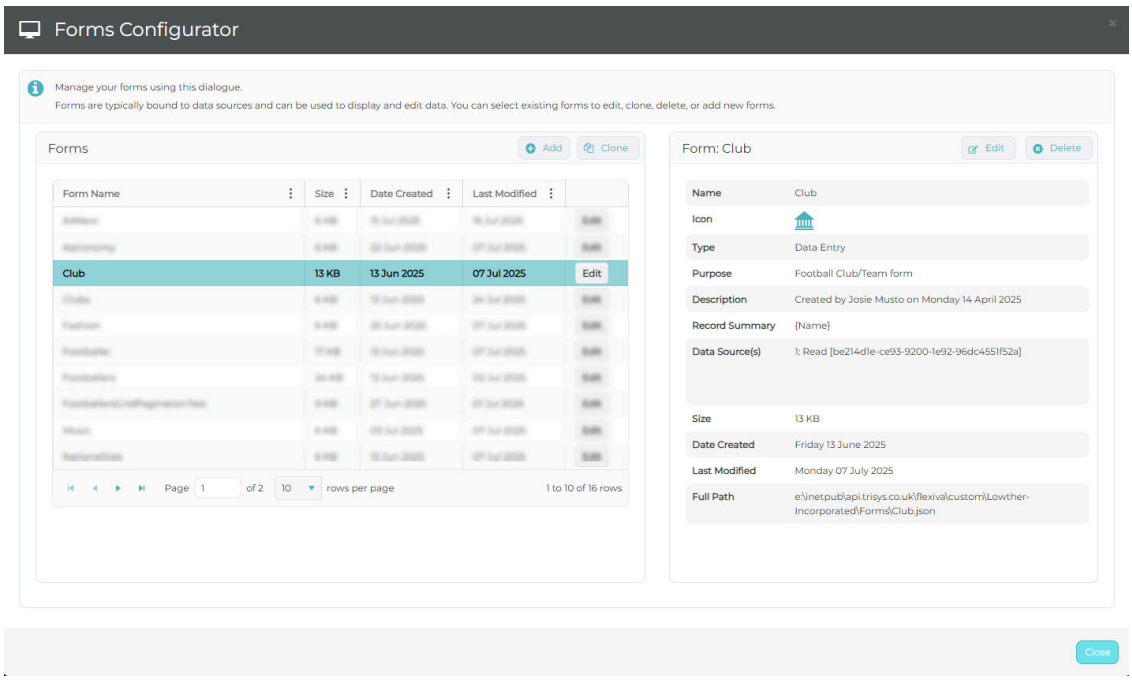
Click the **Select** button to choose this data source.

The popup will close and the selected Read data source request is now added to the list of data sources for this data entry form:



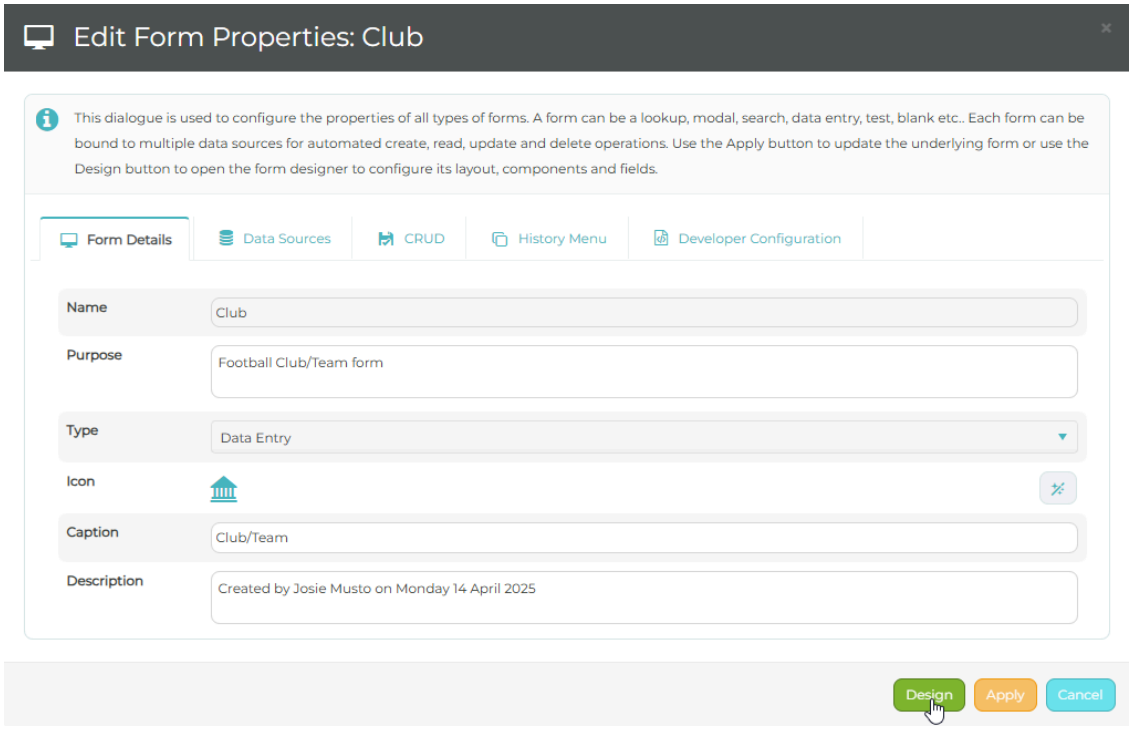
If you ever need to remove a data source request from the list, you can use the X button.

It is recommended that you now click the **Apply** button on this form, in order to persist the form properties before designing your form. Your new form should now appear in the list of forms, and the Read data source request you added should be shown in the properties list:



Form Design

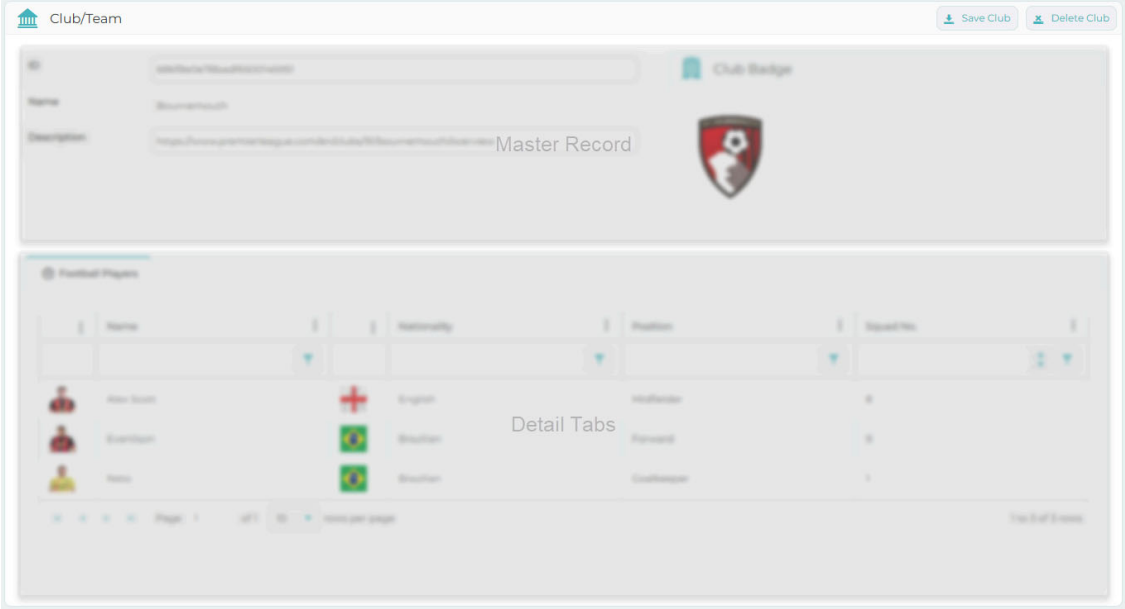
Click one of the Edit buttons, either the grid row or properties panel. The form modal popup will display. Click the **Design** button:



The form designer will open.

Tabs

The tabs toolbar is the selected tool by default. Each data entry form is designed as a master/detail meaning that the master record is shown at the top, and any further details about linked entities is shown below in a series of tabs like this:

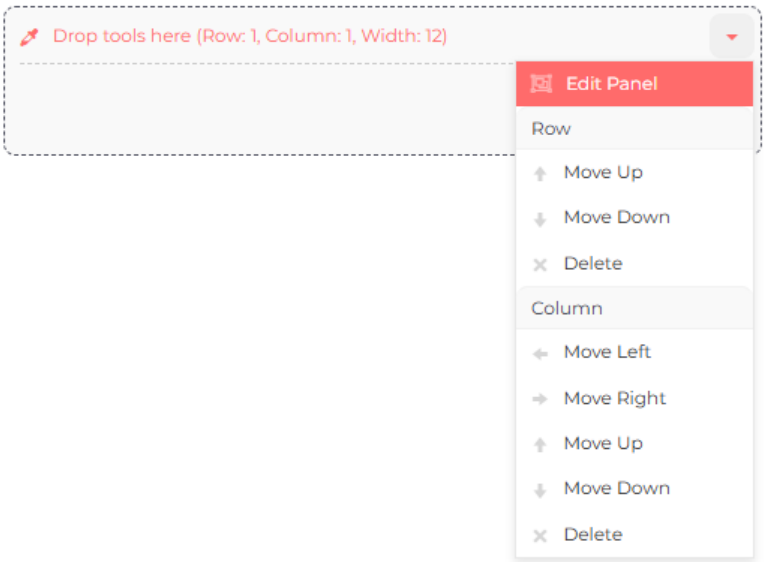


The Main Top Region refers to the master record on the top of the form.

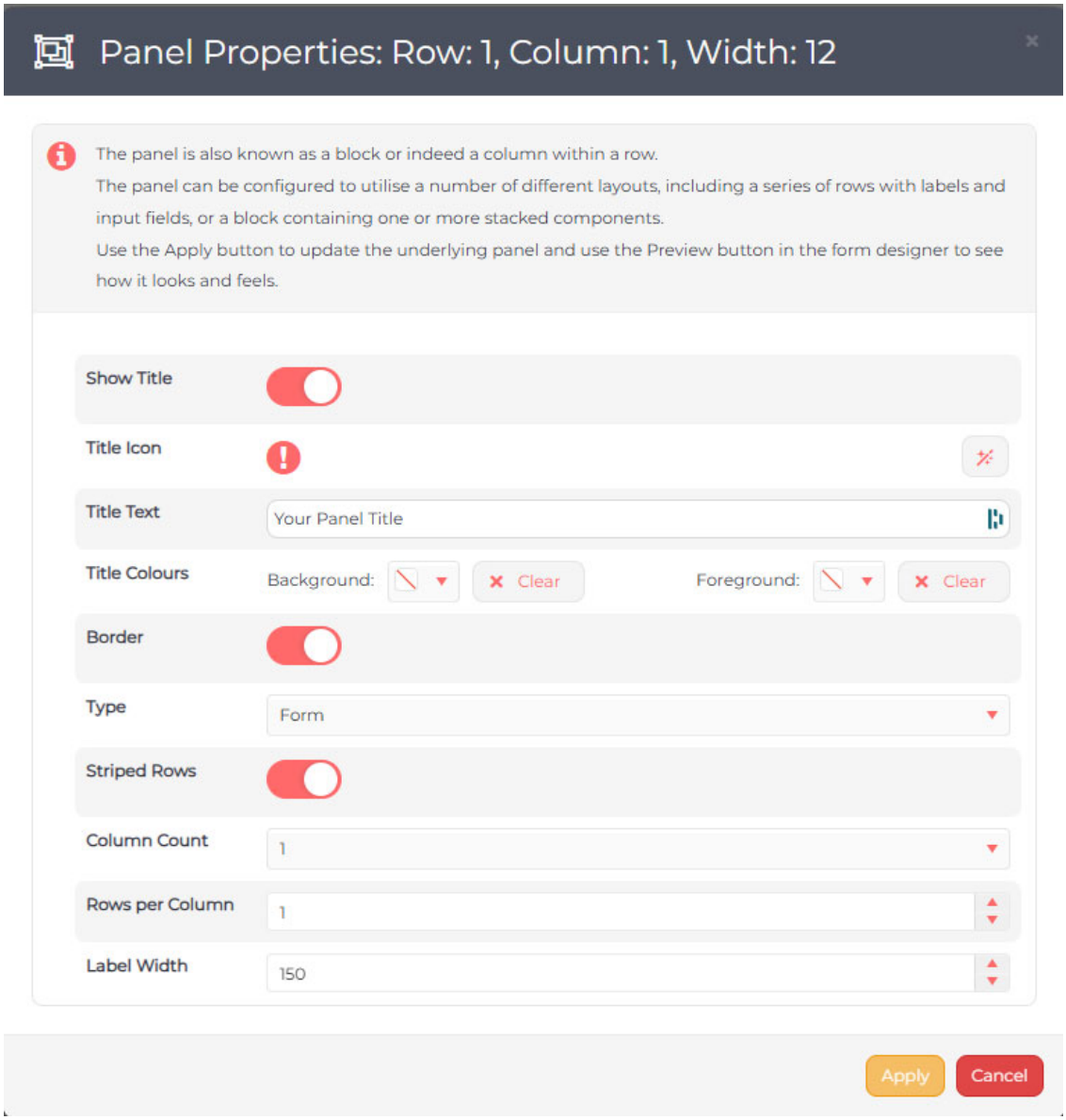
We will design the club/team fields in this region.

Edit Panel

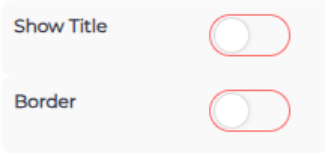
Click the down arrow in the Row 1, Column 1 panel and choose the **Edit Panel** option:



This opens this modal popup:



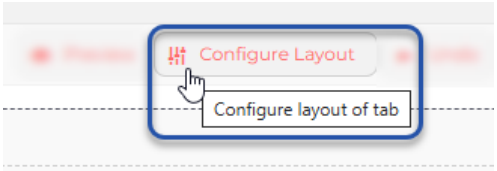
For this simple form, we will hide the title and border by unchecking these properties:



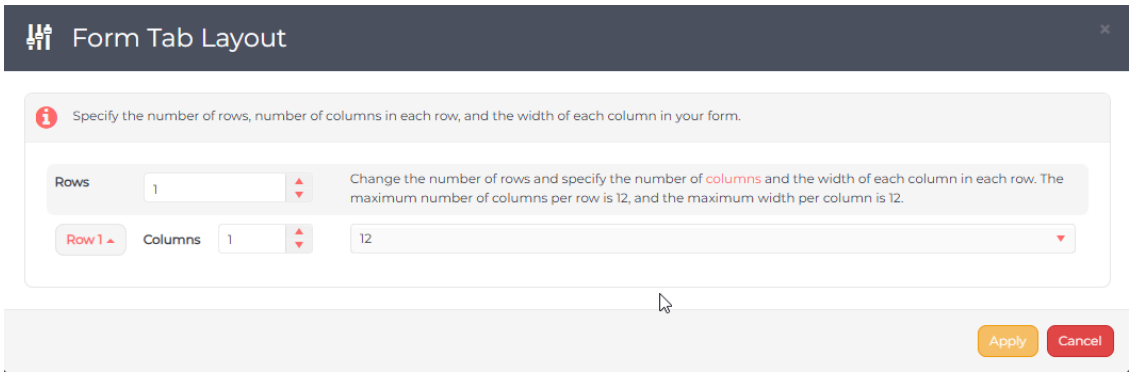
Click the **Apply** button.

Configure Layout

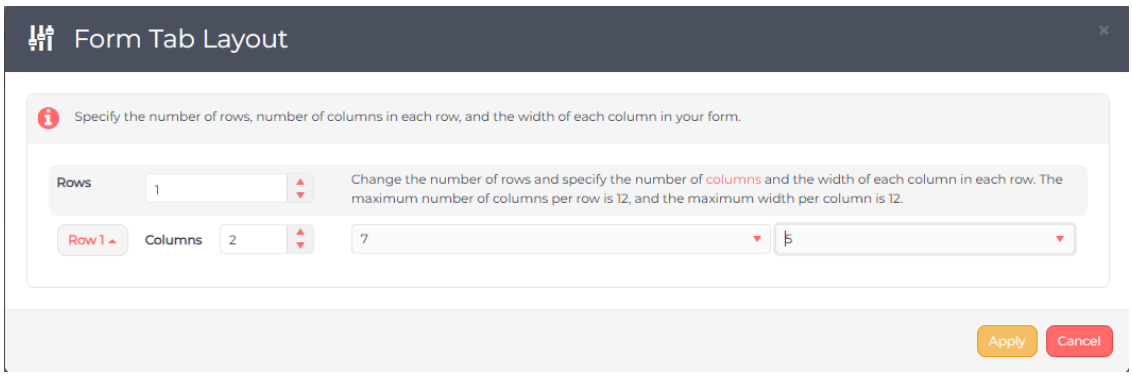
We want to show the club badge on the form, so we will configure the layout to create another column to the right of these fields to span the 3 rows we created. Click this button:



This will open this modal popup to modify the form tab layout:

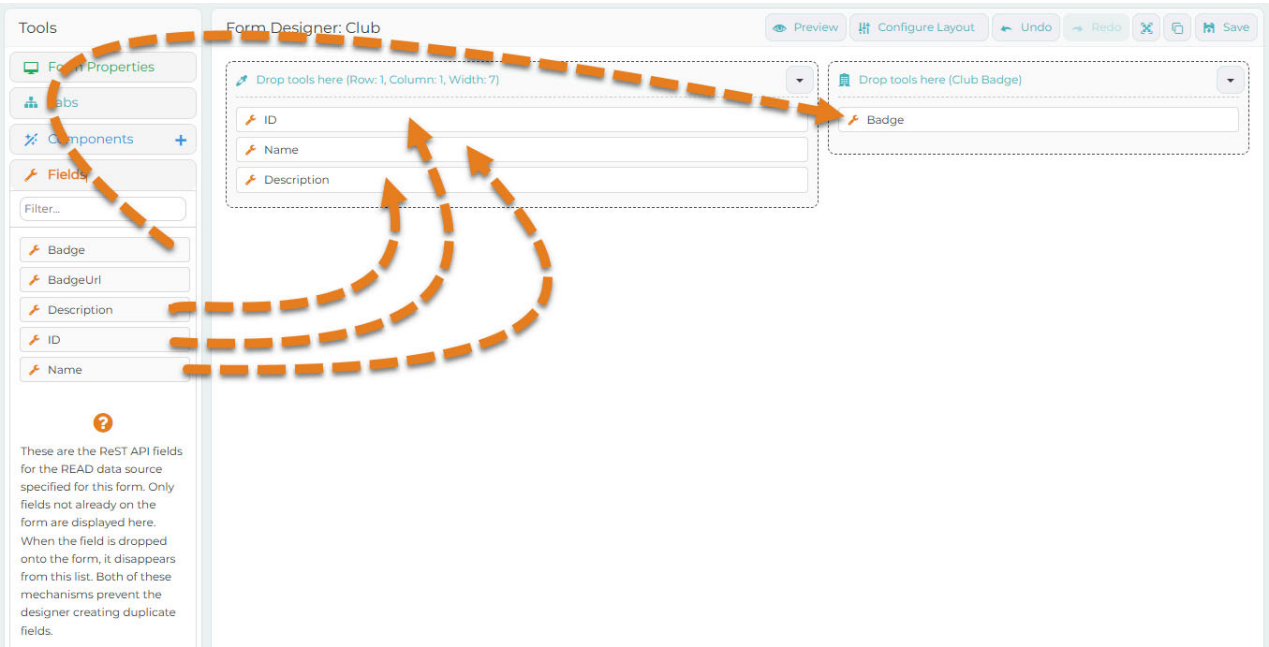


Use the up arrow on the Columns field to increase the number of columns to 2, then set the first column width to 7 and the second to 5:



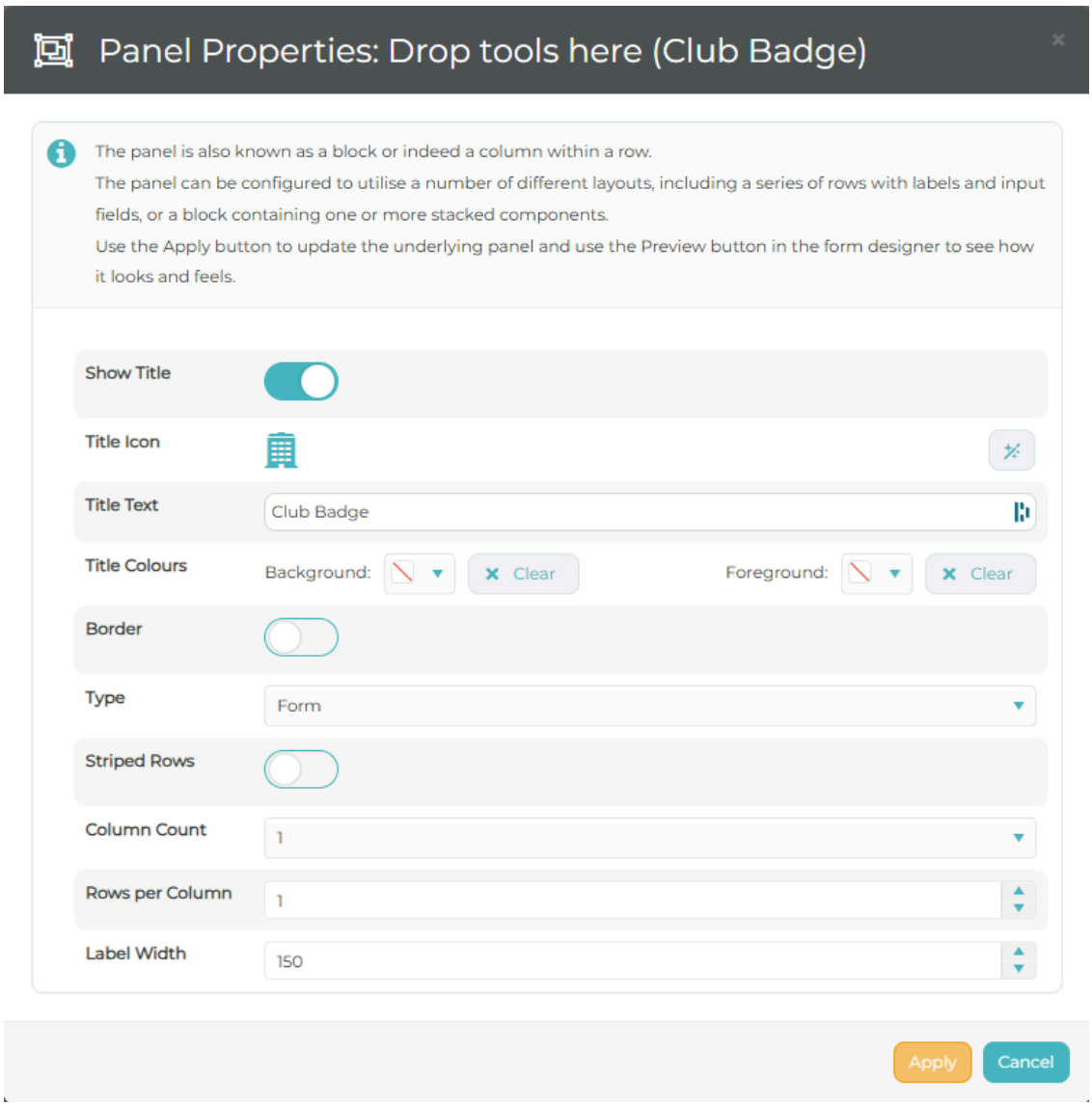
Fields

Click on the fields panel in the toolbar to show the available fields. These fields are those connected to the Read data source request we [added earlier](). Drag these fields as follows:



Club Badge

Click the drop down menu on the right panel to edit it, and make the following changes in the modal popup:



We set the title of the panel to "Club Badge", set the icon to be a building. Click the Apply button.

Save Form

Now save the form design using this button:

We have now completed the design of our data entry form to read a record.

We will now add this to the navigation bar.

Navigation Bar

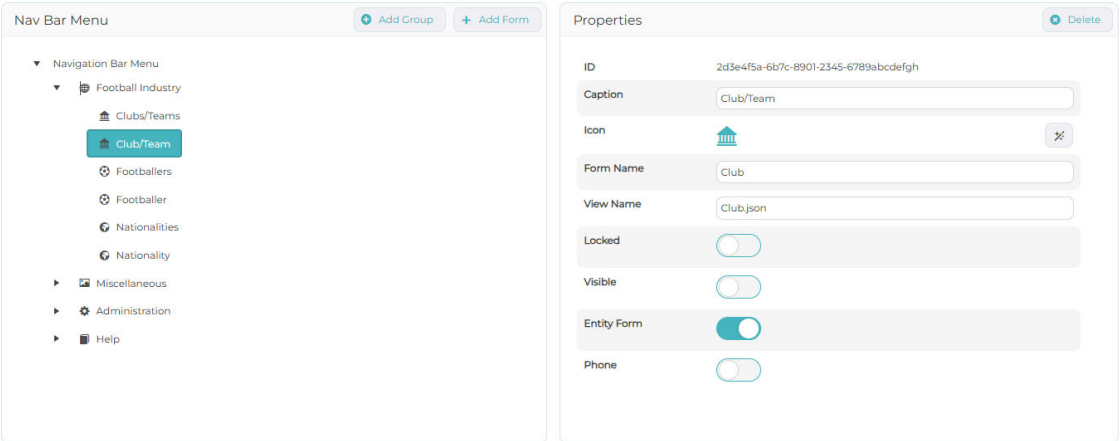
When we add this form to the navigation bar, we will be able to test it.

Open App Studio and select the Navigation Bar configurator and select the Football Industry folder you created previously.

Add Form

Click the Add Form button to open the Add New Form Menu Item modal popup where you should select the Club form you created earlier.

Click the Save button which will close the popup and show the new form in the nav bar menu:



Properties

Check or set these properties.

Visible

Because this is a data entry form, we only want it to be visible in the nav bar when the form is open and showing a record, so this should be unchecked.

Entity Form

This is a data entry form which models entities, so this should be checked.

Apply & Close

Click the **Apply & Close** button to persist the navigation bar and refresh the nav bar.

Configure Drill Down

The navigation bar should not show any change from the last time you saw it because the Club/Team form will only appear on it when the form is opened.

In order to test this data entry form, we need to enable drill down from the Clubs/Teams lookup form we created previously.

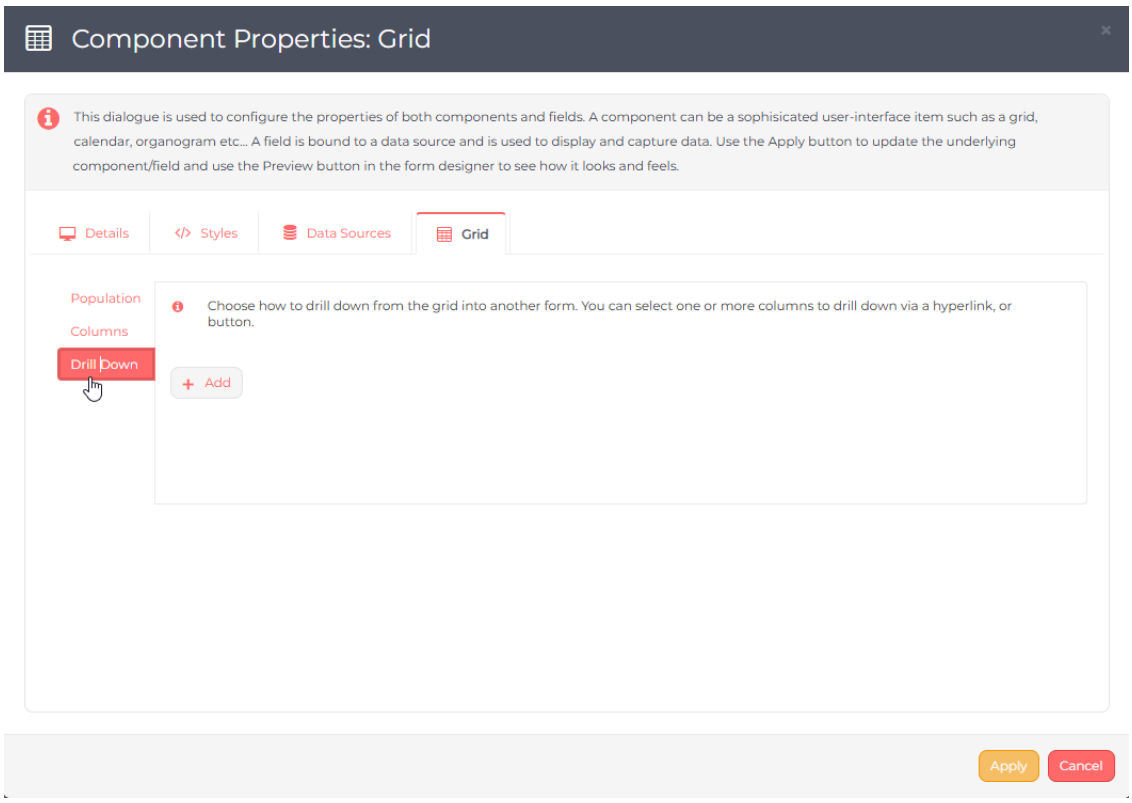
Configure Grid Component

Open App Studio and select the Forms configurator.

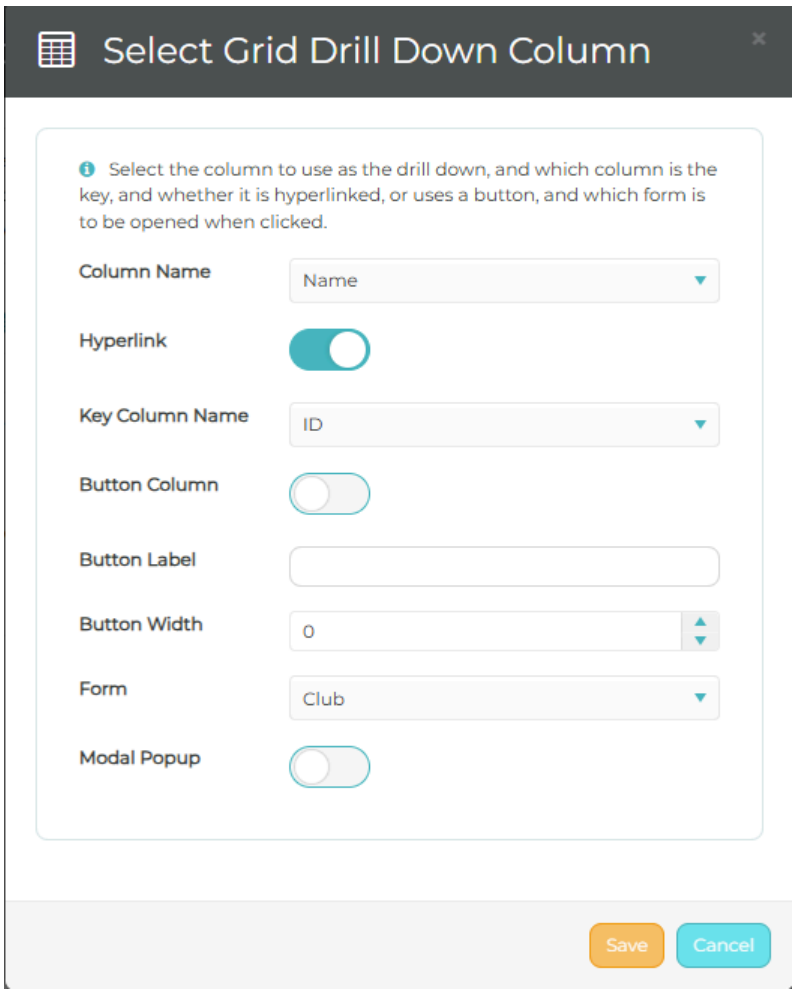
Open Clubs/Teams Form Designer

Open the Clubs form in form designer by choosing Edit then Design.

Click on the Grid component to open this modal popup form:



Select the Grid tab and click the Drill Down left menu option. Click the Add button which will open this modal popup:



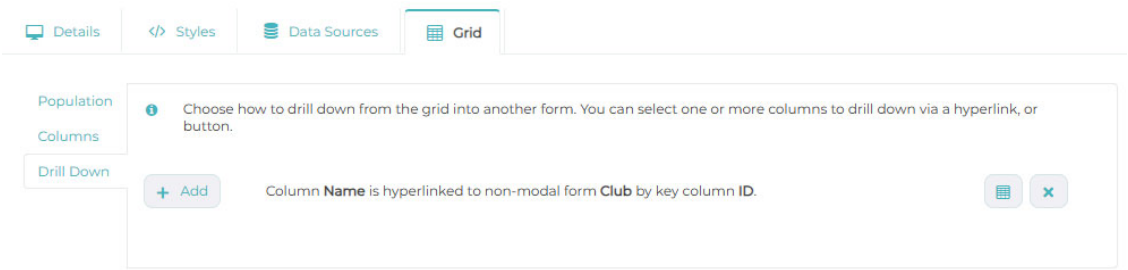
Choose `Name` as the Column Name.

Make sure that the Hyperlink is checked.

Ensure that Key Column Name is the `ID` i.e. the identifier of the club/team record.

Select the `Club` form in the Form field. This is the data entry form we [created earlier](#).

Click the **Save** button which will close the popup and show the drill down hyperlink details:



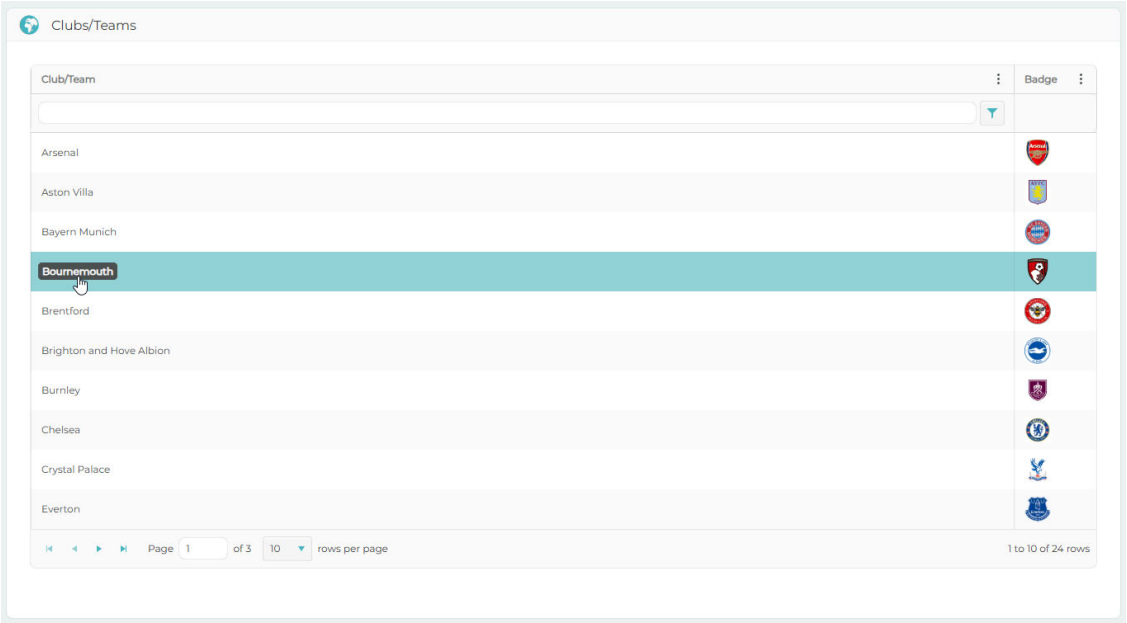
Click the **Apply** button, and then **Save** the form design.

Test Data Entry Form

Open the Football Industry group on the navigation bar.

Clubs/Teams

Click the Clubs/Teams nav bar menu to open the lookup form:

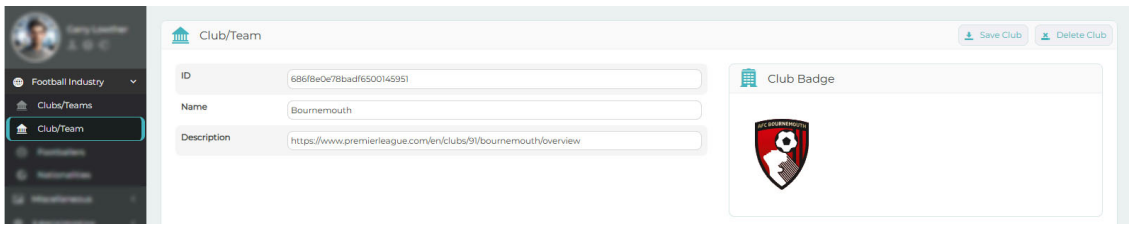


If you hover your mouse over any of the clubs/teams, you should see that it becomes highlighted. This proves that the drill down configuration has been applied.

Click any club.

Club/Team Form

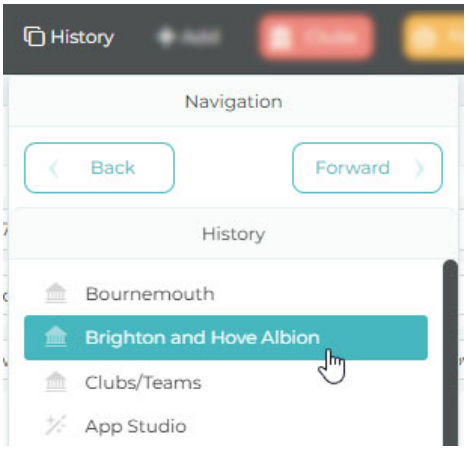
The club/team form should open and show the selected club details including their badge:



Notice also how the navigation bar now shows the Club/Team form as being open?

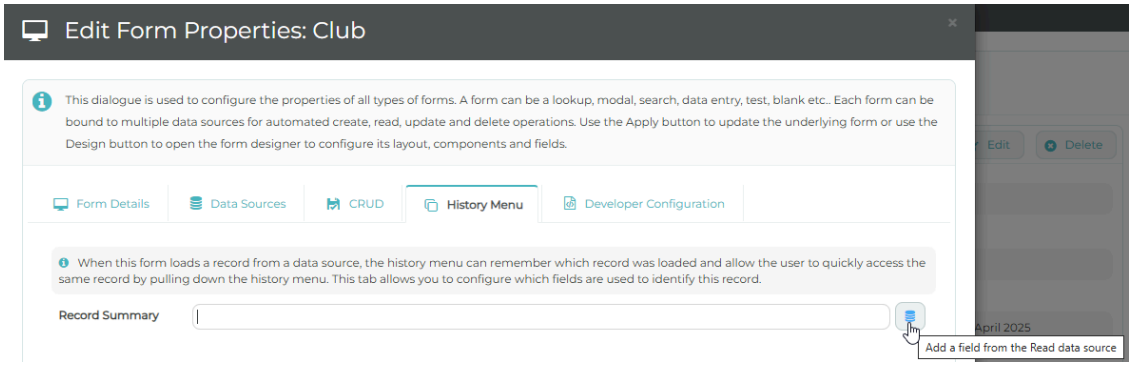
History Menu

When the history menu appears, we want the name of the club/team to appear, not the name of the form for example:

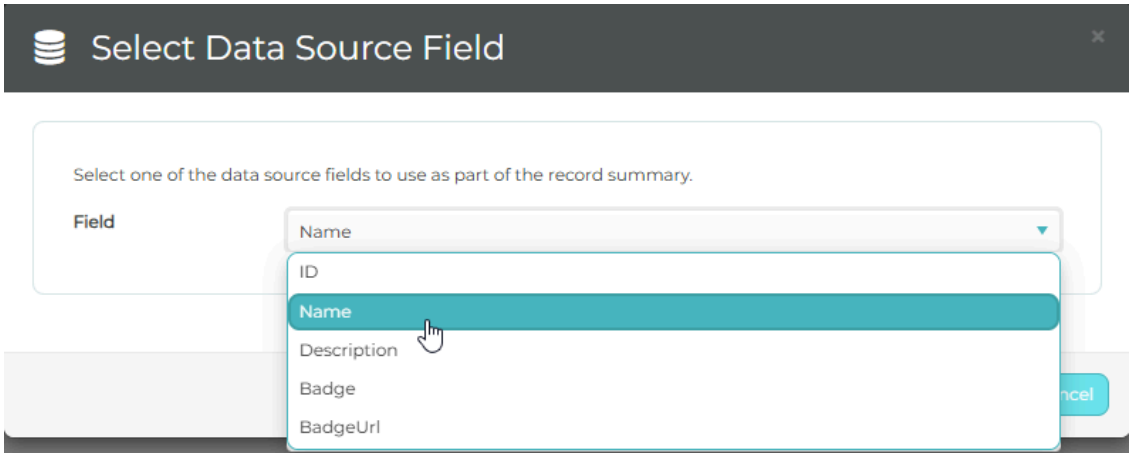


Configuring Record Summary

Open App Studio and select the Forms configurator. Edit the properties of the Club form and click the History Menu tab:



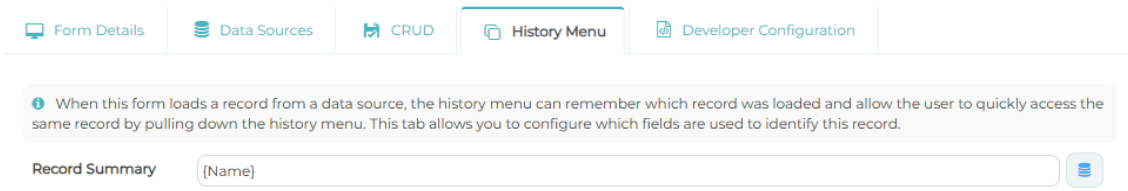
Click this button to open this modal popup:



Choose the `Name` field and click the **Save** button to persist this setting and close the popup.

Record Summary

The record summary should now show `{Name}` indicating that the `Name` field will be displayed in the History Menu:



Apply

Click the **Apply** button to persist this.

Test Record Summary

Open another club/team from the clubs/teams lookup form, and then click on the History drop down menu. You should see the last club you opened at the top of the list?

Football Players

We will now design the Club form and add a grid component to the first tab.

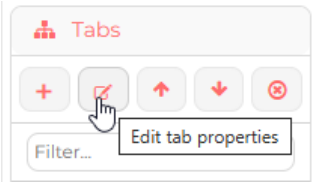
Open App Studio and select the Forms configurator.

Open Club Form Designer

Open the Club form in form designer by choosing Edit then Design.

Edit Tabs

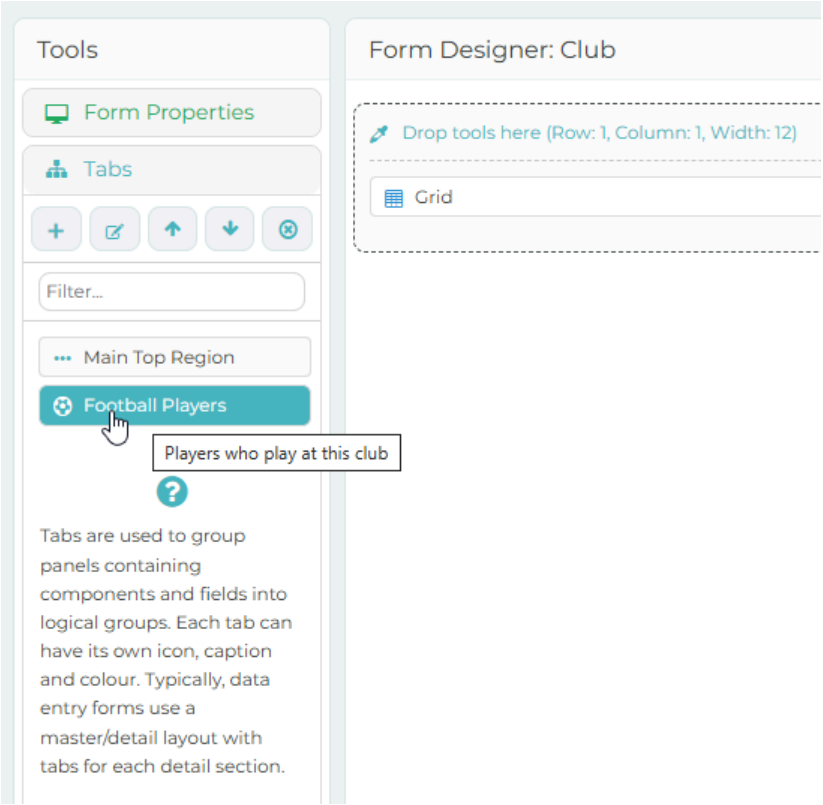
Select the Tabs panel in the Tools panel. Select the second tab and use the tab properties button:



Edit the tab properties then Apply it.

Drag Grid

Drag the grid from the components toolbar onto the first panel:



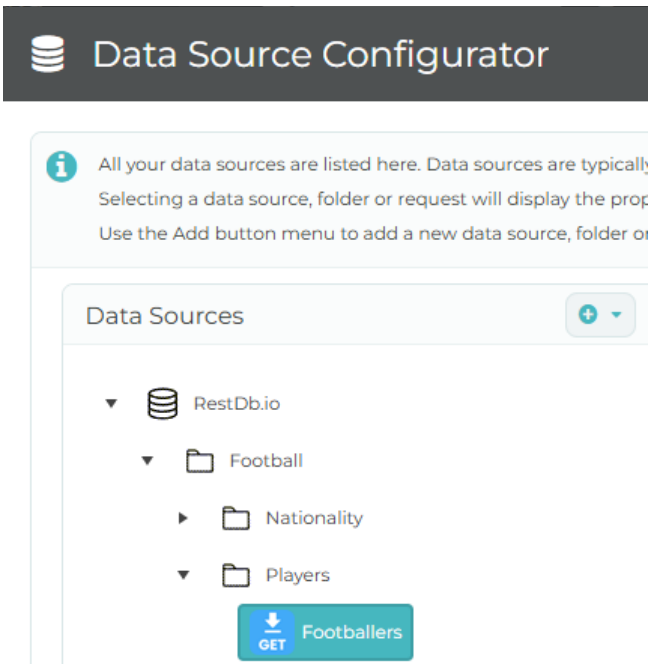
We cannot configure the grid yet because we have not yet configured the data source we created here to show a list of footballers.

Instead, we need to Save the form design to persist our design.

Configure the Footballers Data Source Request

In order to display a list of footballers on our Club form, we need to configure the data source request connected to the appropriate ReST API.

Select this data source in this configurator:

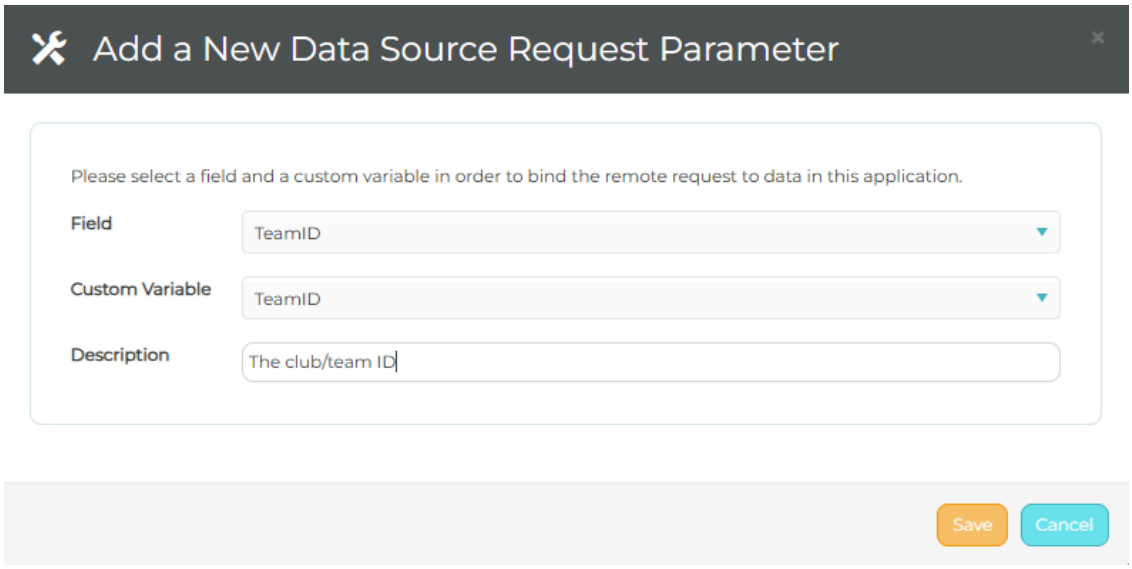


Parameters

Because we want this data source request to be re-used throughout the application, we are going to use optional parameters linked to custom variables. Only if a custom variable value is not empty will it be appended to the URL at run-time.

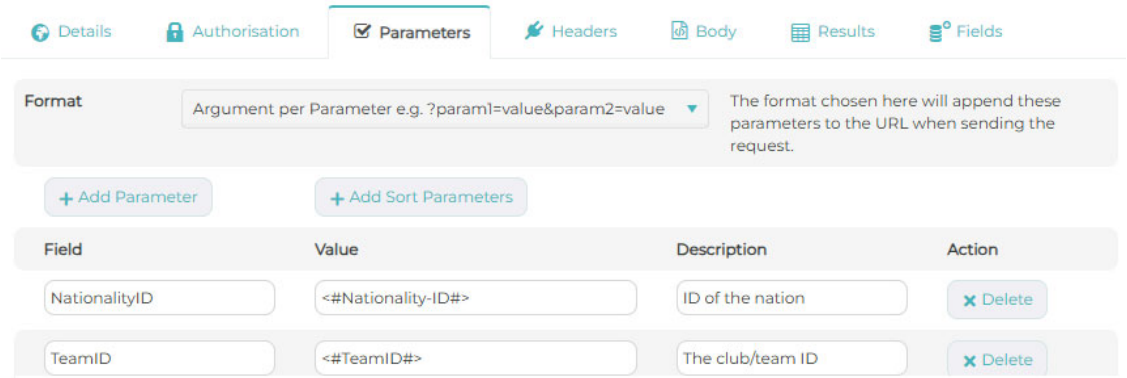
We previously created the `TeamID` custom variable and we know that this ReST API can return footballers belonging to a particular club/team.

Click the Add Parameter button to add this parameter:



Select the Field `TeamID`, then assign that to the Custom Variable `TeamID` then click the Save button.

The parameter is added to the list:



Fields

We previously configured the fields here.

Wire up the Footballers Grid

We can now point the grid on the Club form at this footballers data source request.

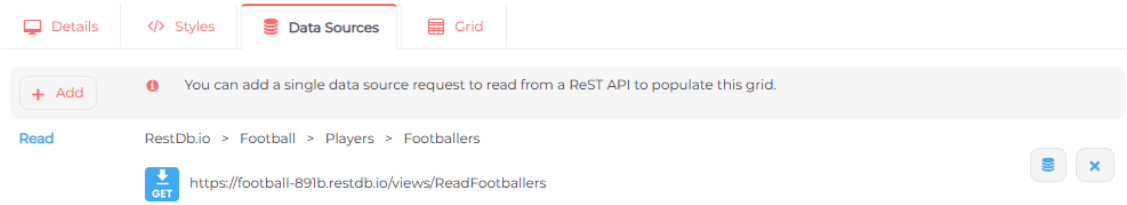
Open App Studio and select the Forms configurator. Open the Club form, edit it, and Design it to open the forms designer.

Add Read Data Source

Click on the grid on the National Team Players tab and open the Component Properties: Grid modal popup and click on the Data Sources tab.

Use the Add button to choose this data source request.

The popup form should now show that a Read data source request is associated with the grid:

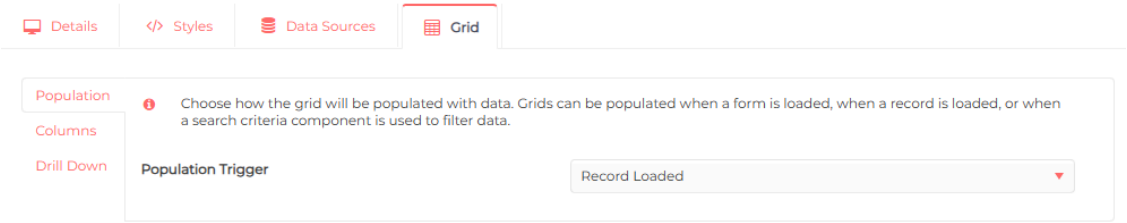


Grid Columns

Click on the Grid tab.

Population Trigger

Select "Record Loaded" in the drop down combo:



Columns

Click on the left tab Columns, and order the columns, hiding some, as you would like to see them.

Because this grid is on a Club form, we do not need to see any of the team columns in this grid. Here are the properties for each column you should set:

Column	Properties
`ID`	Invisible
`PhotoUrl`	Type: Image URL, Image Size: 32 × 32 Visible, Caption: [space], Width: 70
`Name`	Visible
`Biography`	Invisible
`NationalityFlagUrl`	Visible, Type: Image URL, Image Size: 32 × 32, Caption: [space], Width: 70
`NationalityID`	Invisible
`NationalityName`	Visible, Caption: Nationality
`Position`	Visible
`PositionID`	Invisible
`squadNumber`	Type: Number, Visible, Caption: Squad Number
`TeamBadgeUrl`	Invisible
`TeamID`	Invisible
`TeamName`	Invisible

Drill Down

Leave this for now as until we create the footballer form, we have nothing to drill into!

Apply

Apply the changes to these properties.

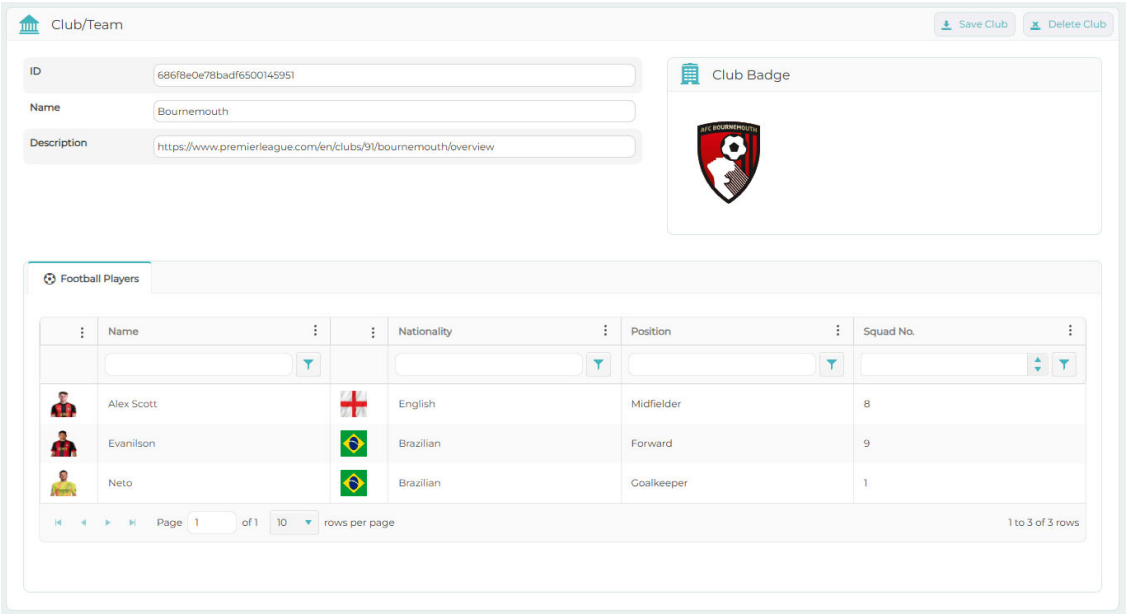
Save

Save the form design to persist the configuration.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Clubs/Teams item. Drill down into any club to open the form.

The Club/Team form now shows the Footballer Players in the grid:



We will revisit this form to configure drill-down once we have completed all CRUD configuration and created the footballer forms.

Club/Team Form: Update

Configure the Club form to update a record.

This is the process we will follow.

UPDATE

- Create custom variables for each field
- Add an 'update' data source for updating a single club/team record
- Link the key club record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'update' data source to the data entry form
- Configure the update button
- Test that we can update a club/team using the Update button

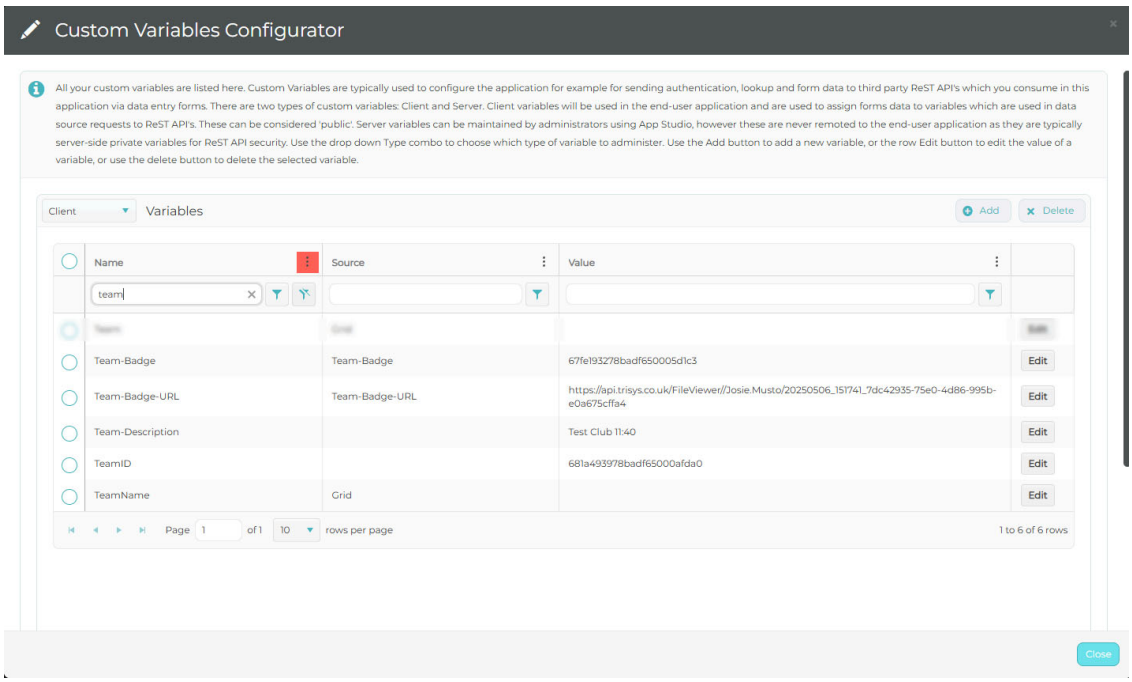
Create Custom Variables

In order to update a club/teamrecord, we need to create custom variables for each form field so that we can send these to the ReST API.

Open the App Studio, then select the Custom Variables configurator.

- Add a client-side custom variable called `TeamName`.
- Add a client-side custom variable called `Team-Description`.
- Add a client-side custom variable called `Team-Badge-URL`.

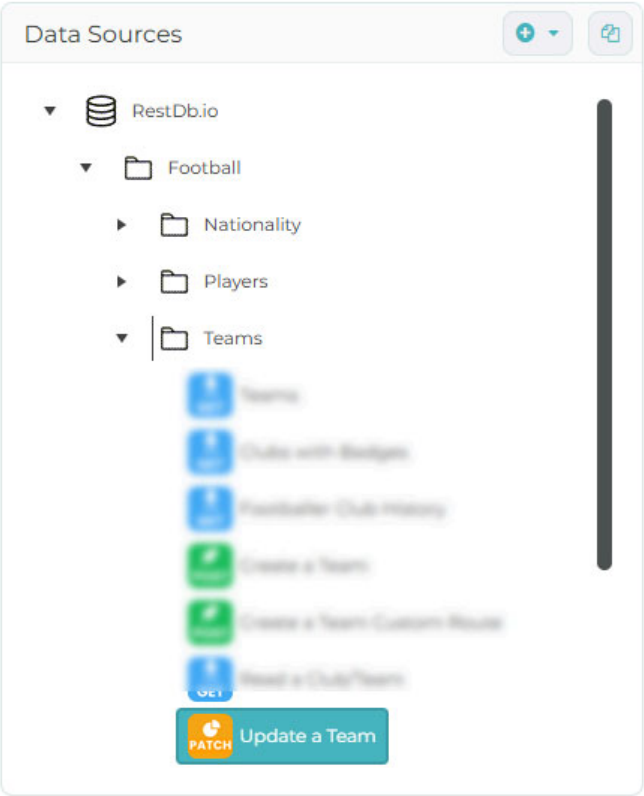
The new custom variables should now be displayed:



We will need to link these new custom variables to each form field, but first we will use them in a new data source to update the record.

Add an UPDATE Data Source

Open App Studio, open the Data Sources configurator, select the `Teams` folder you created previously, then use the add button menu to create a new request called "Update a Team":



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

URL

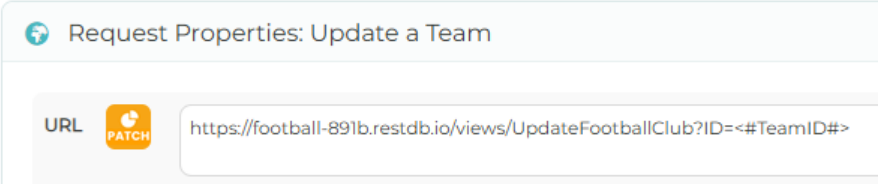
`https://football-891b.restdb.io/views/UpdateFootballClub` ↗

It is known from the documentation that this ReST API end-point has a team identifier property ?ID=.

We therefore type this parameter list into the URL so that it reads:

`https://football-891b.restdb.io/views/UpdateFootballClub?ID=` ↗

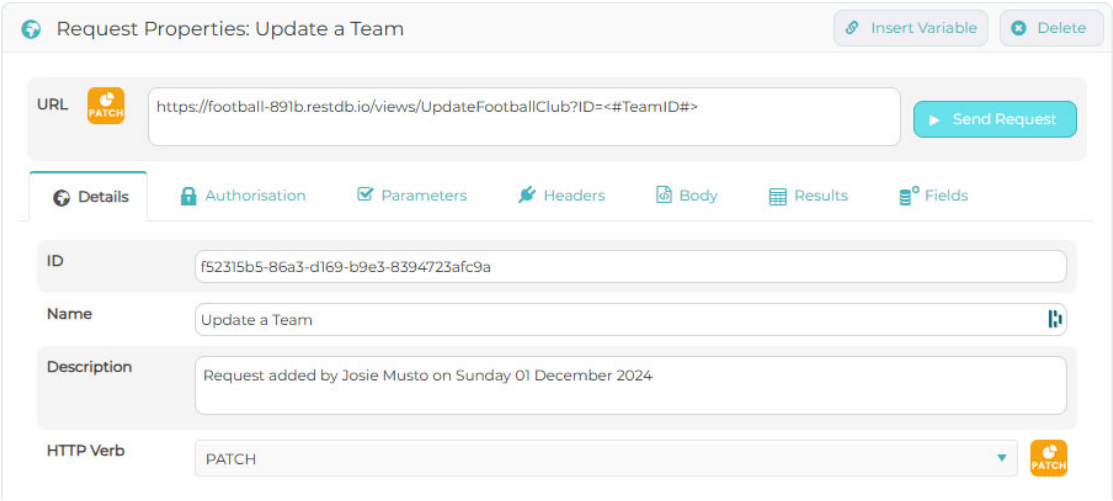
Whilst the caret is still blinking after the last character typed, click on the Insert Variable button and select the `TeamID` variable we created previously. The URL should now show the custom variable appended to the end:



HTTP Verb

The default GET verb/method is not correct for updating a single record using the ReST API and should be set to PATCH for this specific back-end end-point.

The request should now look like this:



Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Body

When we are updating or creating data, we will be sending the data in the body of the request, so we configure this before sending the request to test it, using the new custom variables. Open the Body tab and paste in this JSON from the ReST API specification:

```
{
    "Name": "",
    "Description": "",
    "BadgeURL": ""
}
```

Our job is to now associate each field with the appropriate custom variable.

Put the carat inside each double quotation and use the Insert Variable button to select the respective custom variables setup above.

After all three have been inserted, the body should now look like this:

```
{
    "Name": "<#TeamName#>",
    "Description": "<#Team-Description#>",
    "BadgeURL": "<#Team-Badge-URL#>"
}
```

Send Request

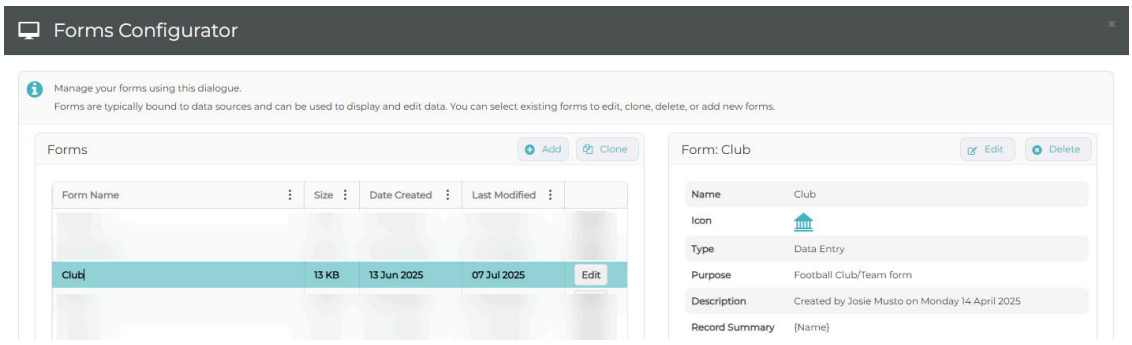
It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to update an existing record in order to test this.

The best way of course to test this is to use the actual club/team form we created here, and link that form to this data source request, then we can test it.

Edit Form

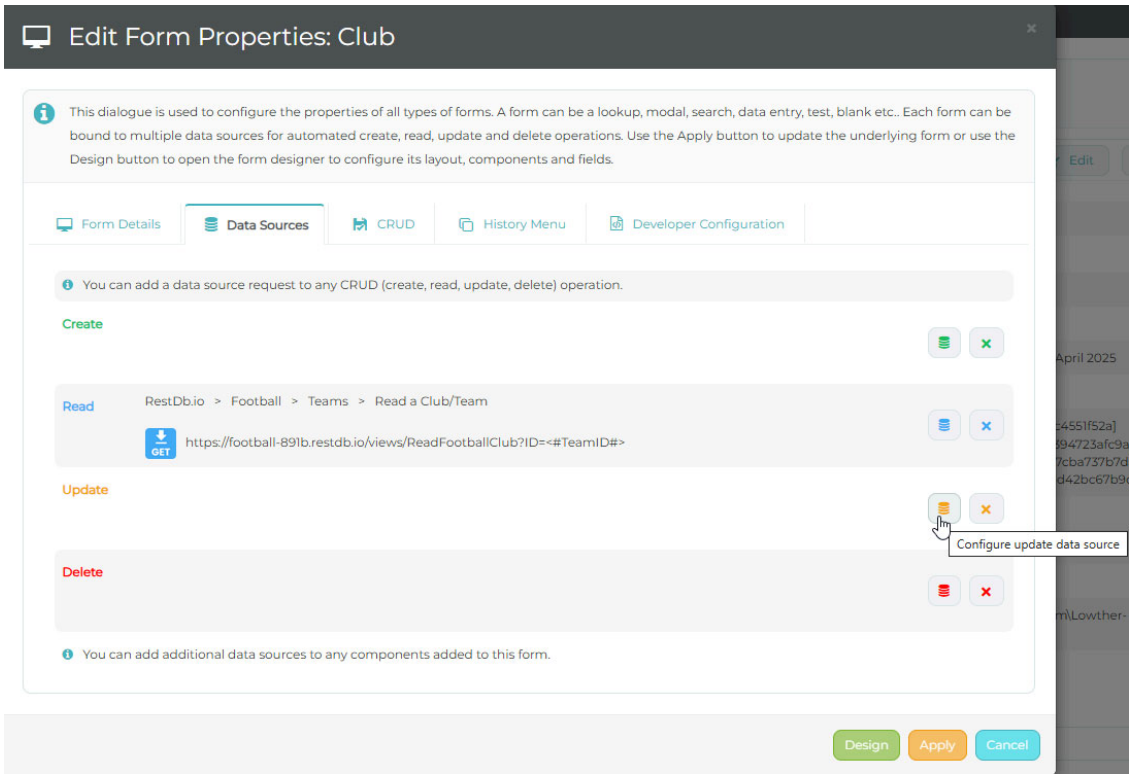
Open App Studio, open the Forms configurator, then select the Club form:



Click the **Edit** button to open the form properties modal popup.

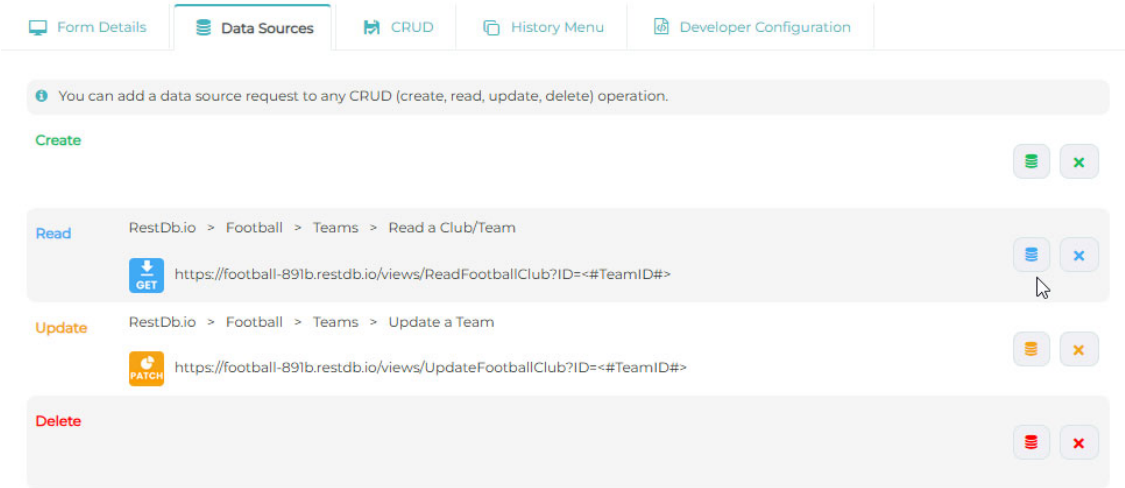
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Update data source:

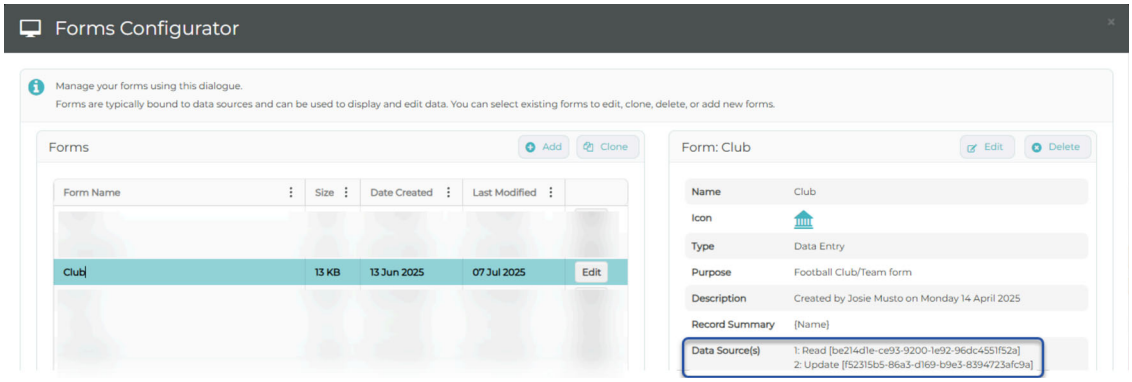


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the Select button.

This Update data source request should now appear in the list of assigned data source requests:

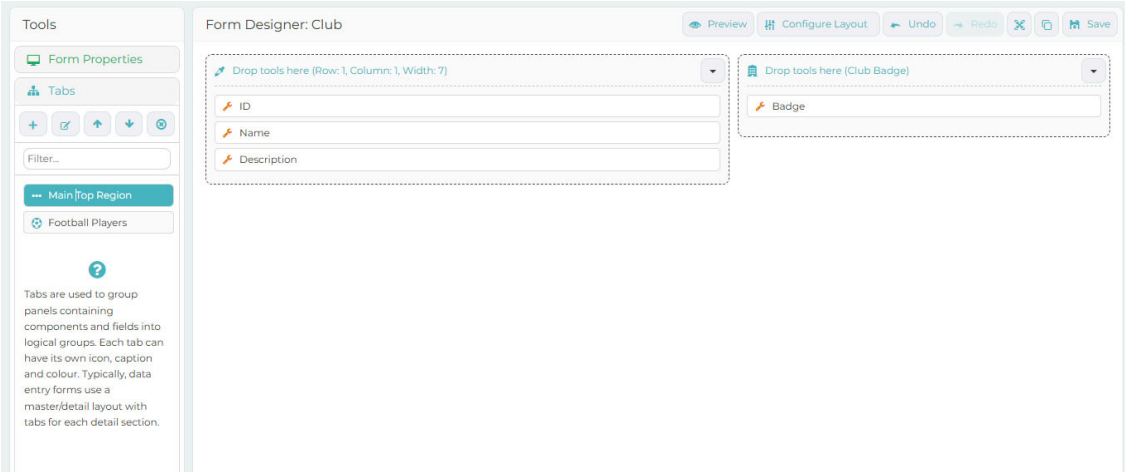


Now click the **Apply** button on this form, in order to persist the form properties before designing your form. The Update data source request you added should be shown in the properties list:



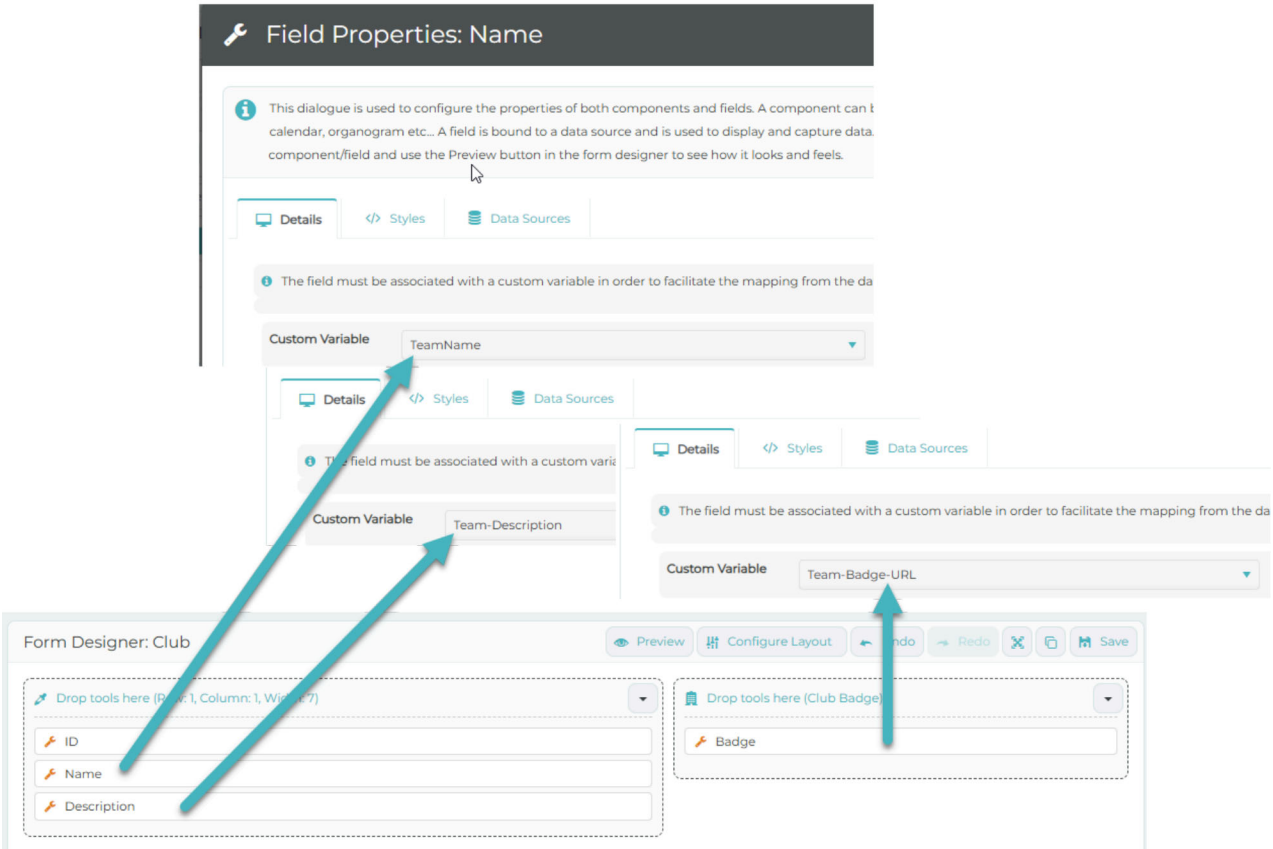
Form Design

Click one of the Edit buttons, either the grid row or properties panel. The form modal popup will display. Click the **Design** button, and select the Main Top Region tab:



Fields

Click on each of these fields in turn and assign the appropriate custom variable you [added previously](#):



Save Form

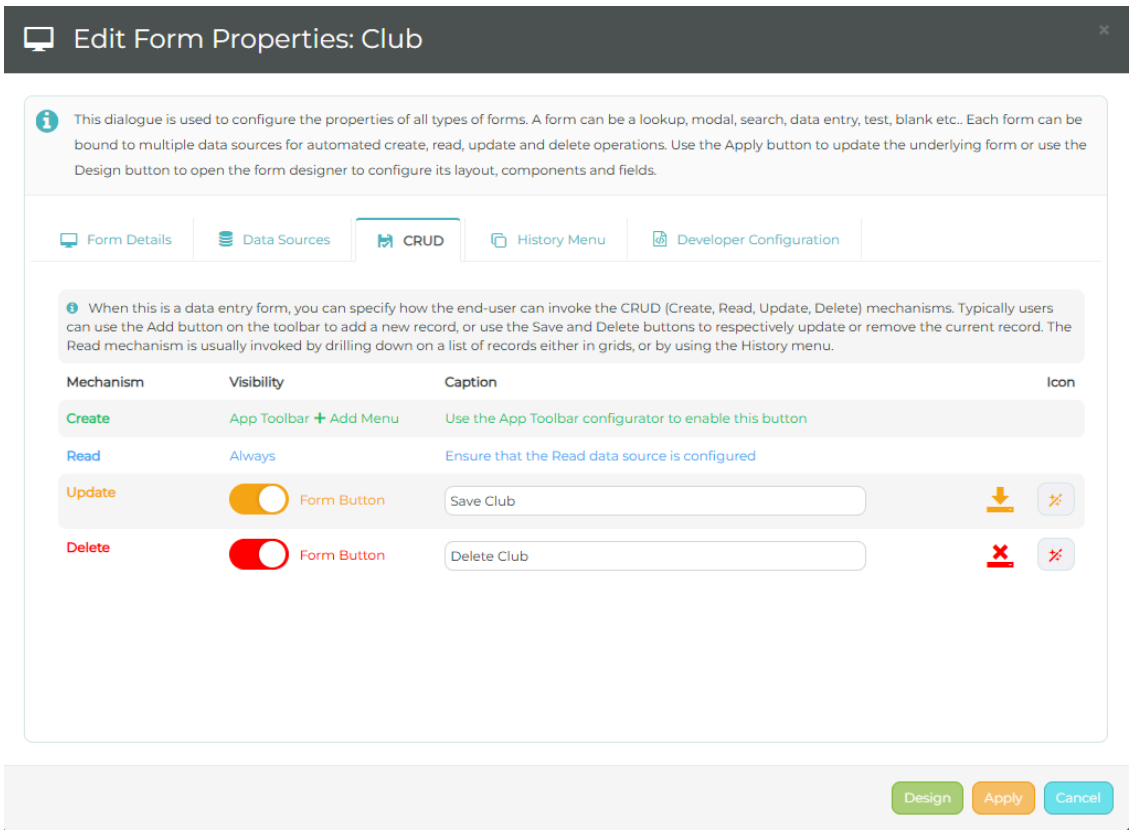
Now save the form design using this button:

The form fields are now assigned to the new custom variables used in the body of the new data source request to update the record.

We will now configure the update button on the form.

Configure Update Button

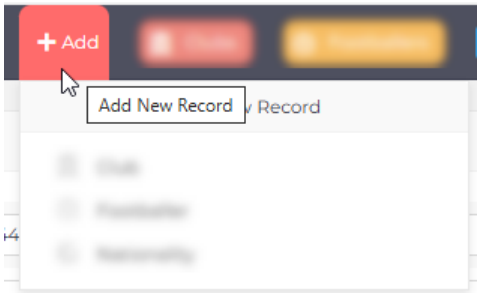
Open App Studio, open the Forms configurator, then select the Club form and click the edit button. Then click the CRUD tab:



There are four mechanisms to control: Create, Read, Update and Delete.

Create

A new record can only be created from the app toolbar using the Add drop down menu:



Use this configurator to add any form to this list.

Read

Reading form data is always visible when a form is opened and a record is loaded.

Update

The update button lives together with the delete button top right on the form:

It can be hidden, its caption set and its icon set using these controls:

Delete

The delete button lives together with the update button top right on the form. It can be hidden, its caption set and its icon set using these same controls.

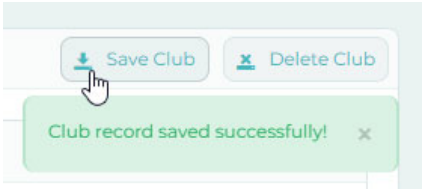
Apply

Apply any changes to persist them before testing.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Club/Team item. Drill down into any club to open the form.

Add an X to the end of the Name and Description fields, or indeed click the Club Badge and upload a new image. Then press the Save button. The form record update should be confirmed:



Club/Team Form: Create

Configure the club/team form to create a record.

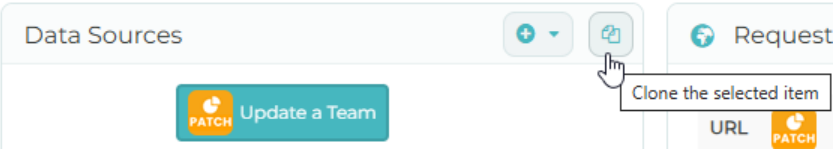
This is the process we will follow.

CREATE

- Add a 'create' data source for creating a single club record
- Link the key club record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'create' data source to the data entry form
- Configure the app toolbar Add menu to add this data entry form
- Test that we can create a club/team using the Add menu and Update button

Add a CREATE Data Source

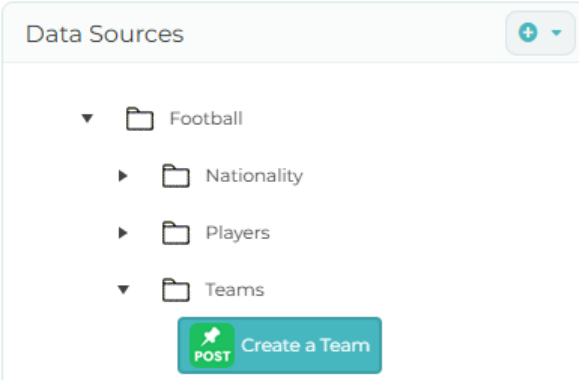
Open App Studio, open the Data Sources configurator, select the Teams folder you created previously, then select the previously created "Update a Team" data source request, and clone it:



The Clone Request modal popup form shows asking you to name the request:

Type a meaningful name such as "Create a Team" and click Save.

The request will be added to your tree view beneath the Teams folder:



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

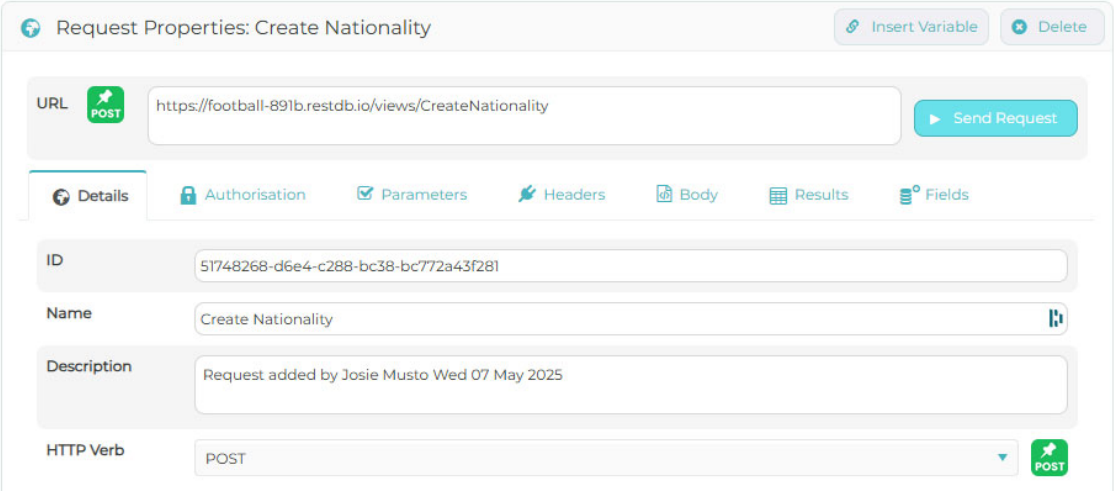
URL

`https://football-891b.restdb.io/views/CreateFootballClub` ↗

HTTP Verb

The default GET verb/method is not correct for creating a single record using the ReST API and should be set to POST for this specific back-end end-point.

The request should now look like this:



Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Body

We will be sending the data in the body of the request, and because we [cloned the Update](#) data source request, then we know that we can use that as-is:

```
{
    "Name": "<#TeamName#>",
    "Description": "<#Team-Description#>",
    "BadgeURL": "<#Team-Badge-URL#>"
}
```

Send Request

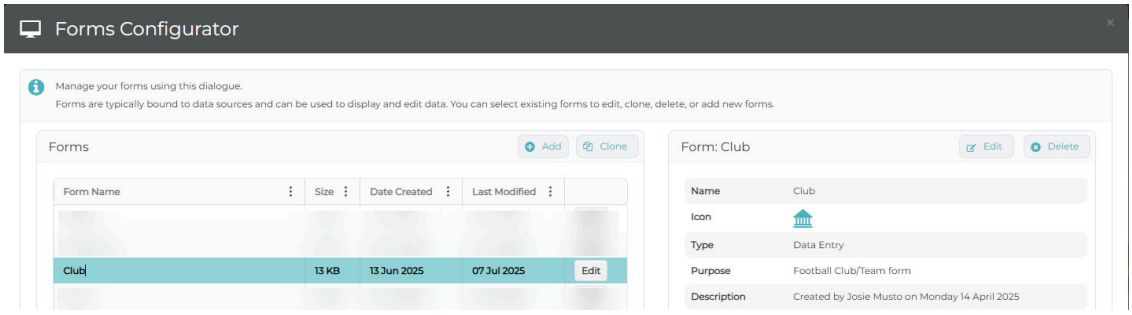
It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to create an existing record in order to test this.

The best way of course to test this is to use the actual nationality form we created here, and link that form to this data source request, then we can test it.

Edit Form Properties

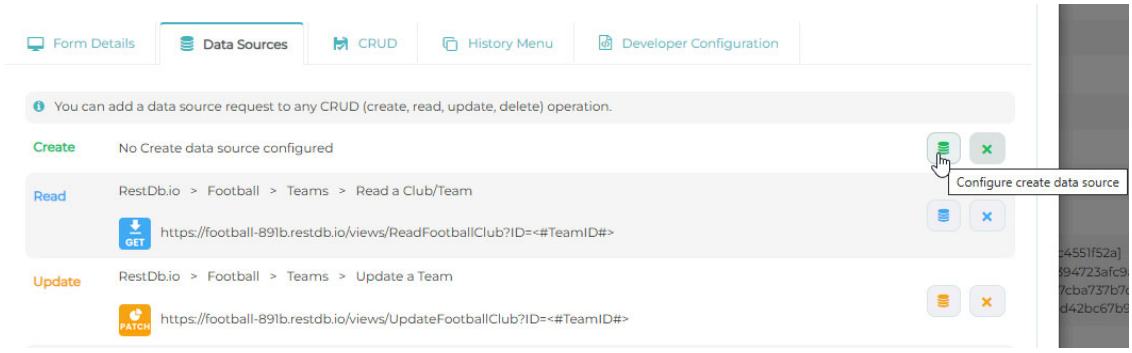
Open App Studio, open the Forms configurator, then select the Club form:



Click the **Edit** button to open the form properties modal popup.

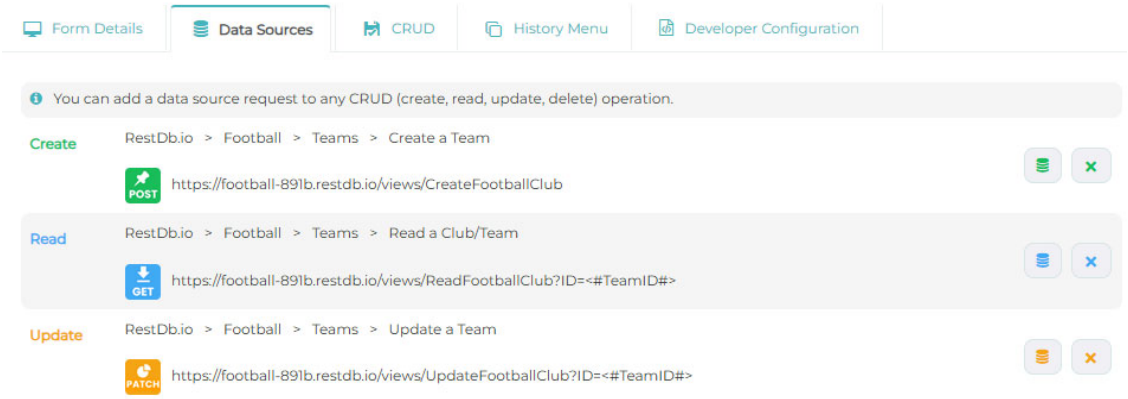
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Create data source:

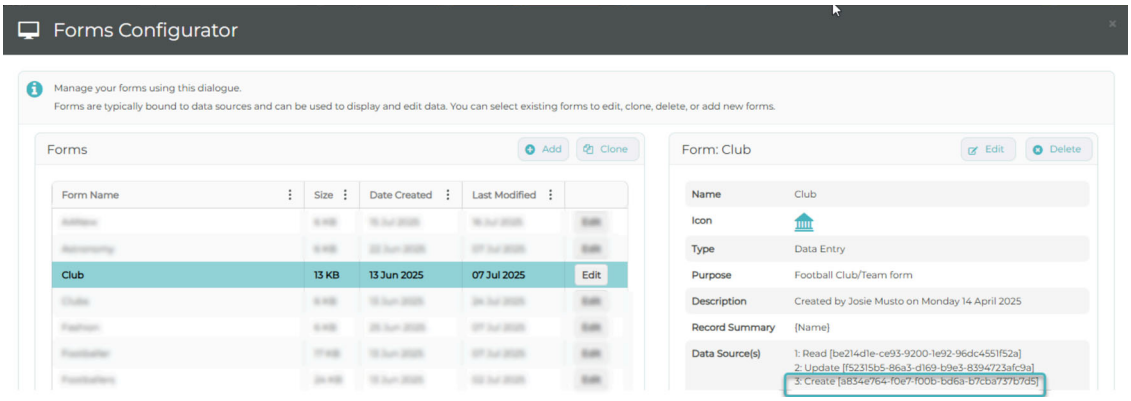


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the **Select** button.

This Create data source request should now appear in the list of assigned data source requests:



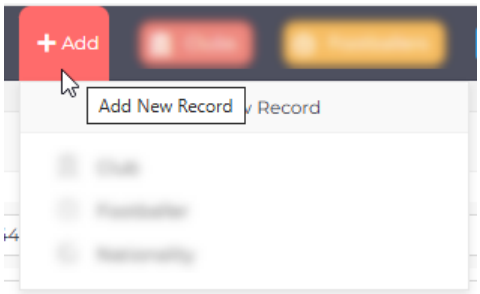
Now click the **Apply** button on this form, in order to persist the form properties. The Create data source request you added should be shown in the form properties list:



Note that we previously assigned the custom variables to the form fields, so we do not need to do this again, as our new data source request uses the same custom variables. We also configured the update/save button, which will automatically call the new create data source method where necessary.

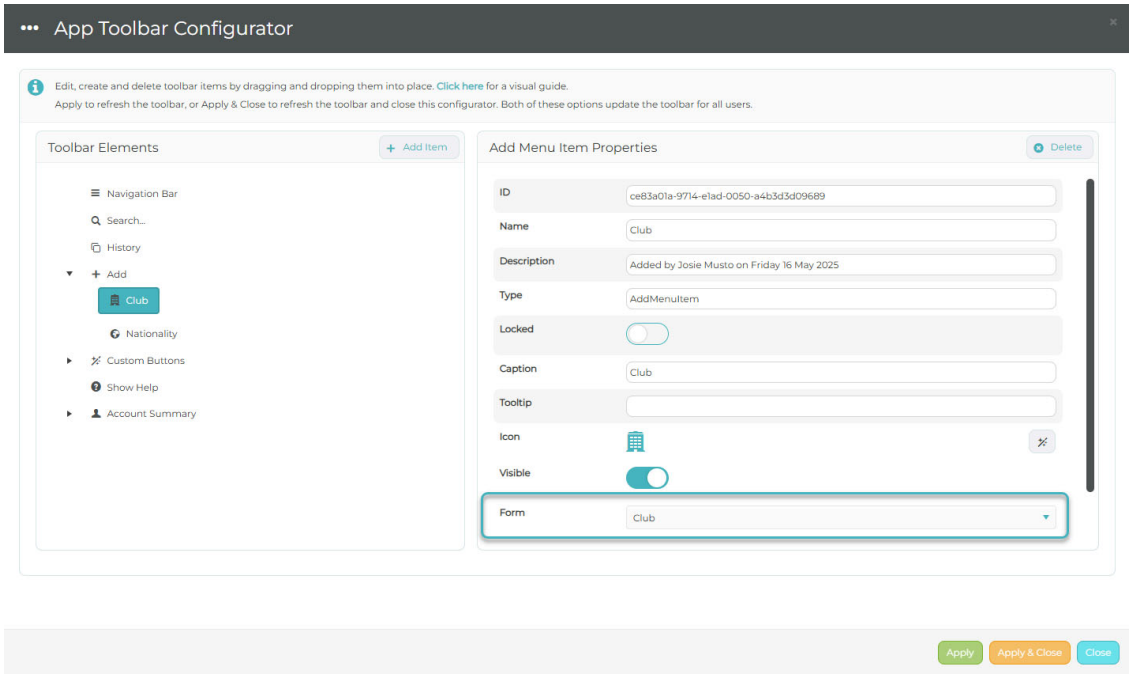
Configure Add Menu

A new record can only be created from the app toolbar using the Add drop down menu:



Use this configurator to add this form to this list.

This is what your Add menu should look like in the App Toolbar Configurator after you have created the Club form. Note how you will have assigned the form as shown:

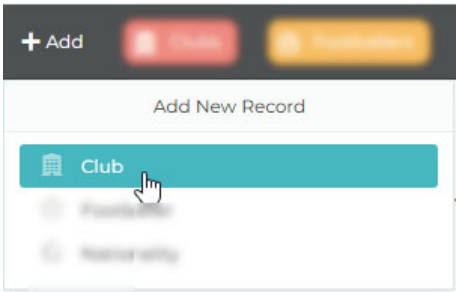


Apply

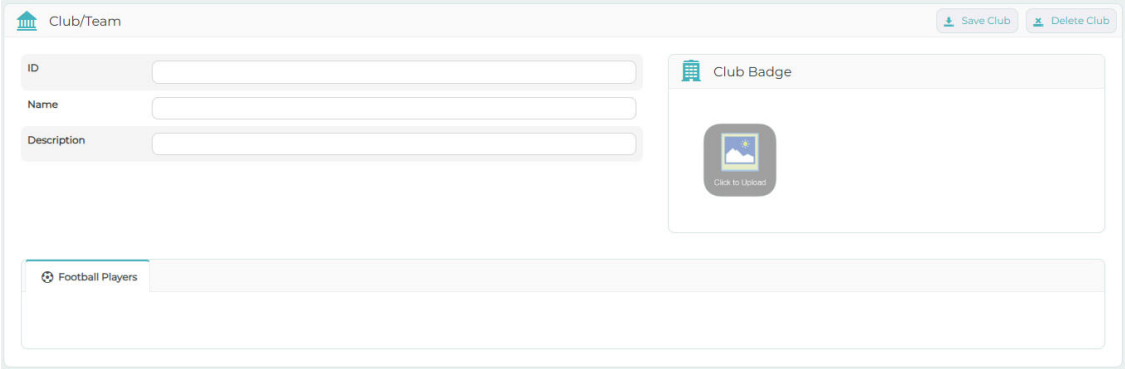
Apply any changes to persist them before testing.

Test

You can now test your configuration by clicking the Add menu button on the toolbar and selecting Club:



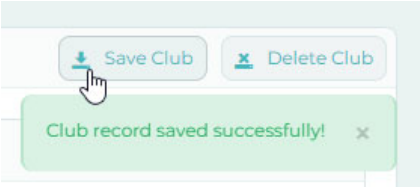
The Club/Team form should open:



Type in a name and the respective description. Perhaps choose a fictitious team ↗ for testing?

Click the "Click to Upload" image to upload a club badge image.

Press the **Save** button. The form record creation should be confirmed:



Club/Team Form: Delete

Configure the club/team form to delete a record.

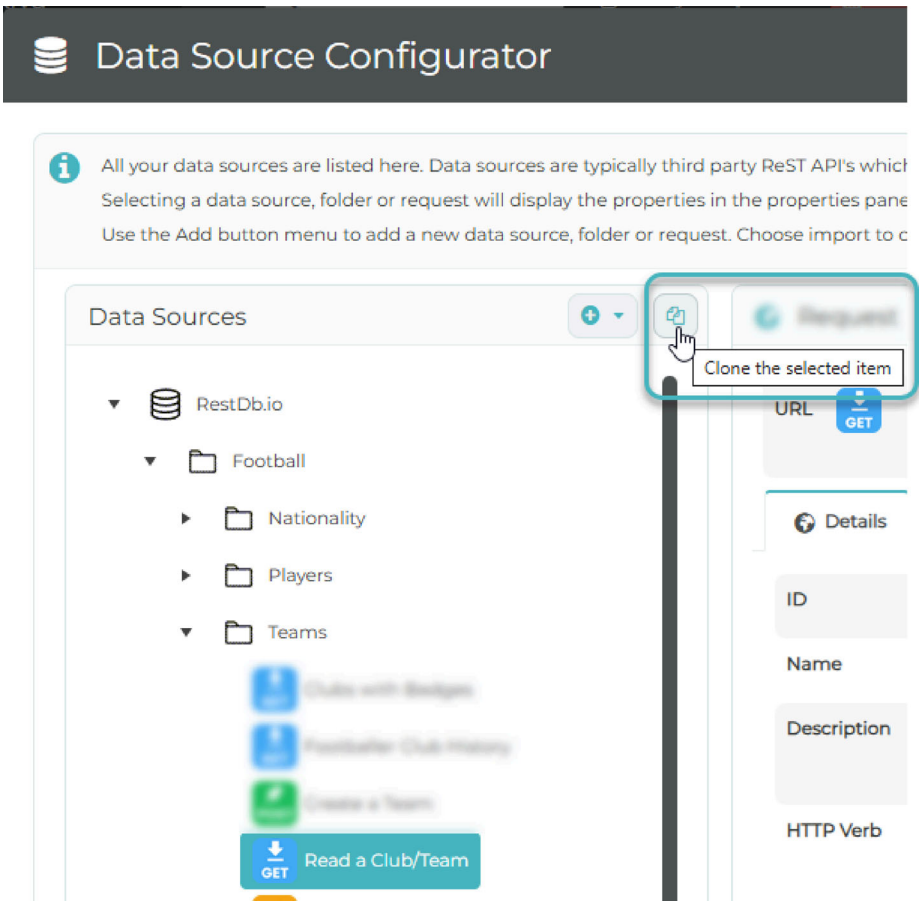
This is the process we will follow.

DELETE

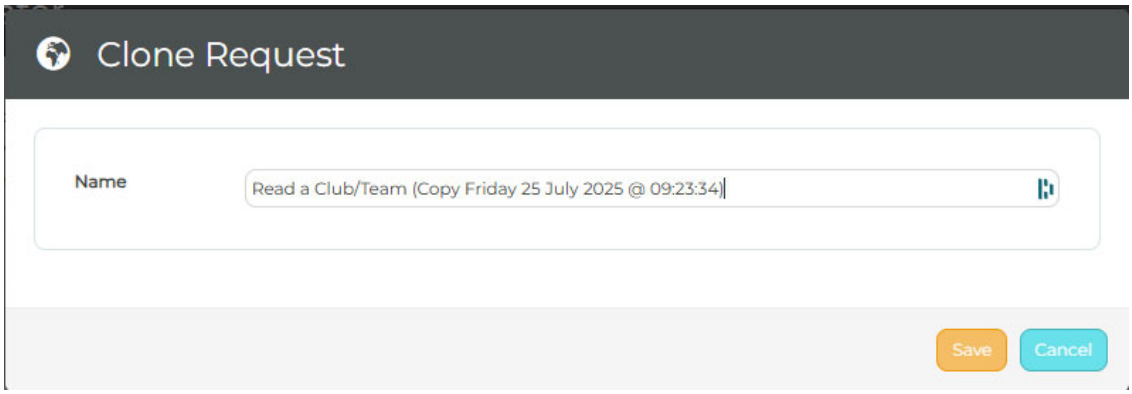
- Add a 'delete' data source for deleting a single club record
- Link the key club record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'delete' data source to the data entry form
- Configure the delete button
- Test that we can delete a club/team using the Delete button

Add a DELETE Data Source

Open App Studio, open the Data Sources configurator, select the Teams folder you created previously, then select the "Read a Club/Team" node and use the clone button to clone this request:

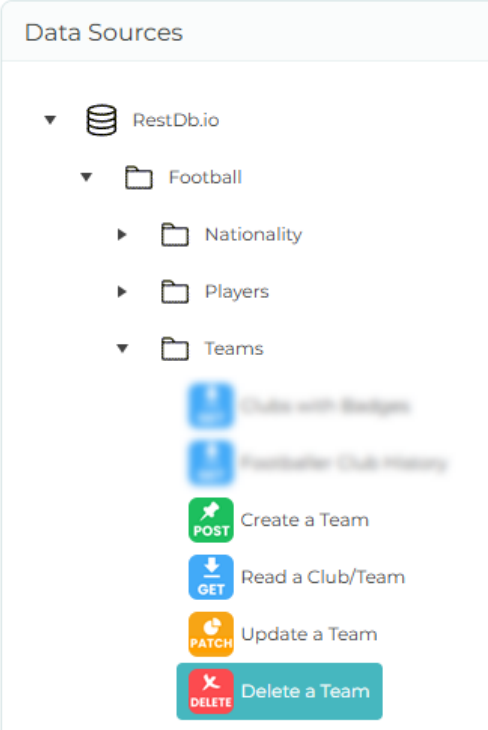


The Clone Request modal popup form shows asking you to name the request:



Type a meaningful name such as "Delete a Team" and click Save.

The request will be added to your tree view beneath the Teams folder. Take this opportunity to drag and drop the order of the requests to fit the CRUD acronym:



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

URL

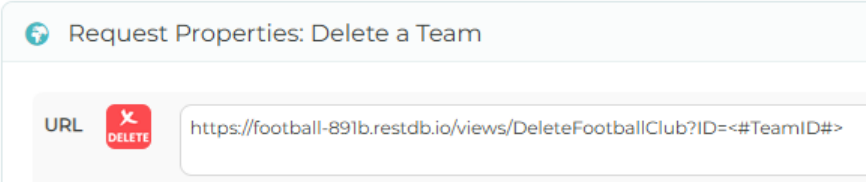
`https://football-891b.restdb.io/views/DeleteFootballClub` ↗

It is known from the documentation that this ReST API end-point has a team identifier property ?ID=.

We therefore type this parameter list into the URL so that it reads:

`https://football-891b.restdb.io/views/DeleteFootballClub?ID=` ↗

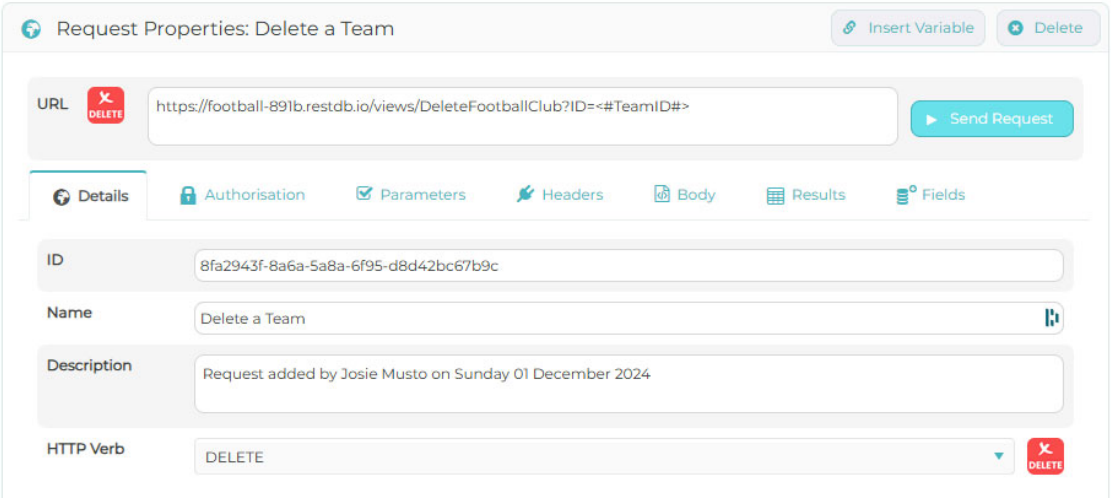
Whilst the caret is still blinking after the last character typed, click on the Insert Variable button to choose the `TeamID` custom variable which when applied should show the custom variable appended to the end:



HTTP Verb

The default GET verb/method is not correct for deleting a single record using the ReST API and should be set to DELETE for this specific back-end end-point.

The request should now look like this:



Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Send Request

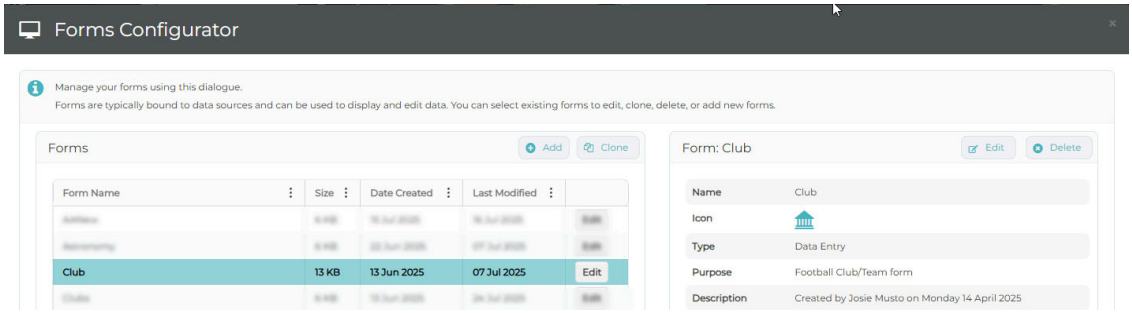
It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to delete an existing record in order to test this.

The best way of course to test this is to use the actual Club form we created here, and link that form to this data source request, then we can test it.

Edit Form

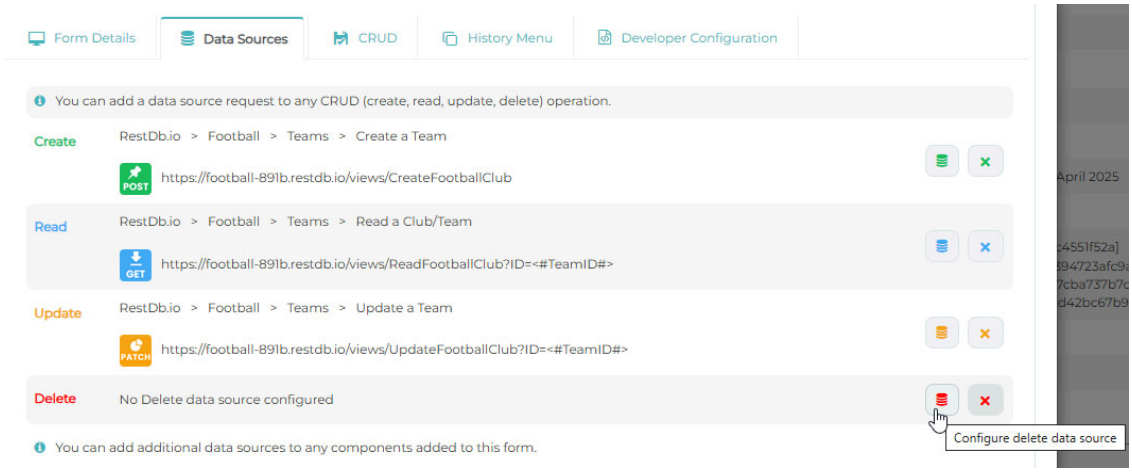
Open App Studio, open the Forms configurator, then select the Club form:



Click the **Edit** button to open the form properties modal popup.

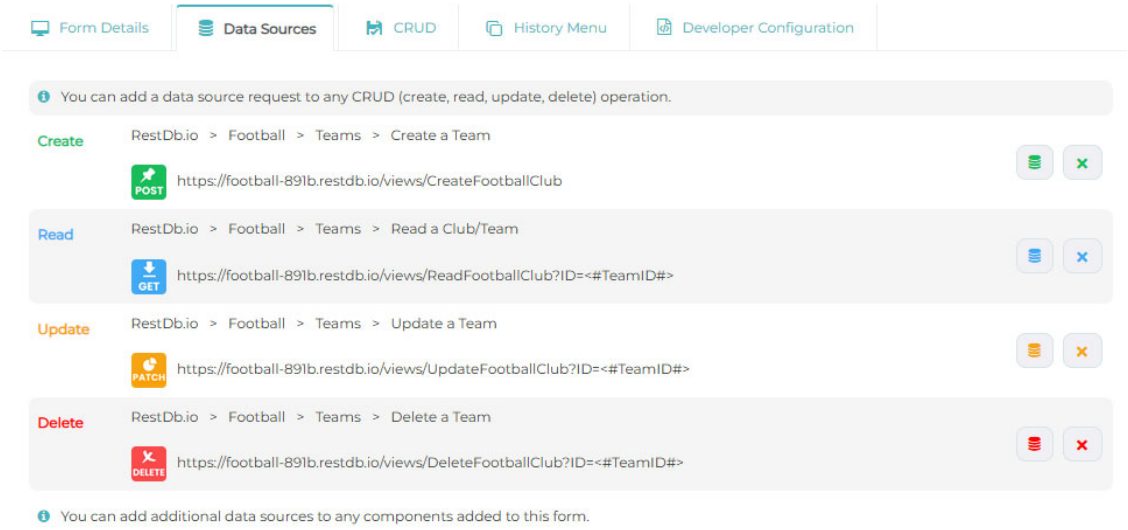
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Delete data source:

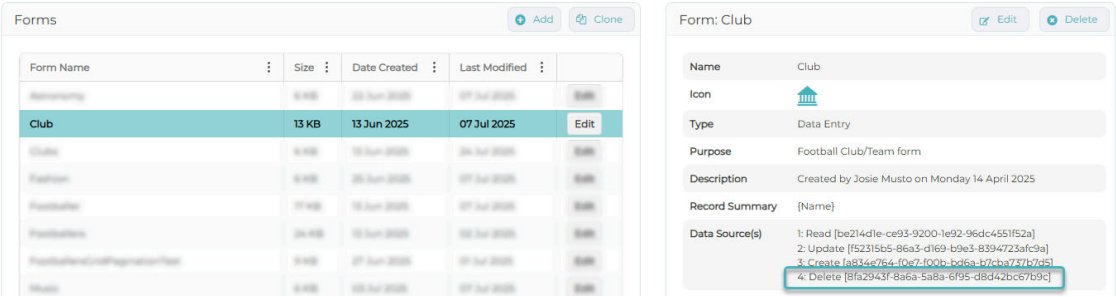


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the **Select** button.

This Delete data source request should now appear in the list of assigned data source requests:



Now click the **Apply** button on this form, in order to persist the form properties. The Delete data source request you added should be shown in the properties list:

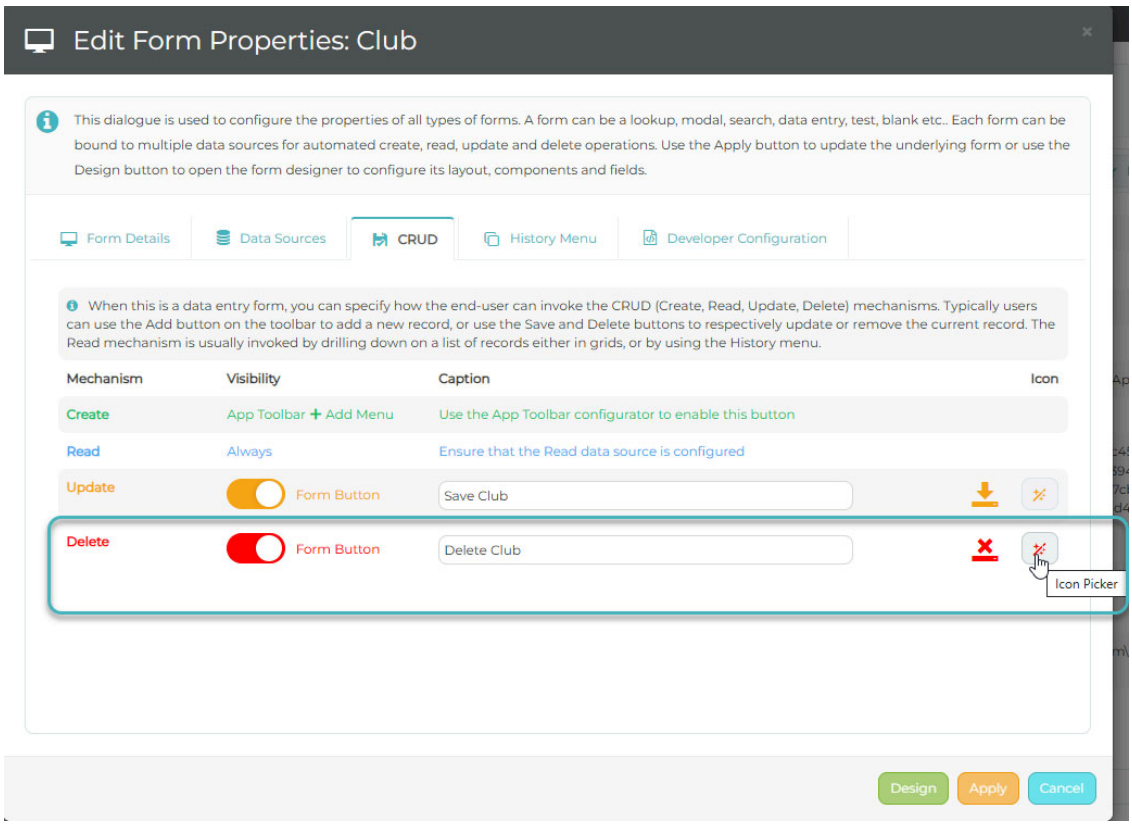


We do not need to design the form.

We will now configure the delete button on the form.

Configure Delete Button

Open App Studio, open the Forms configurator, then select the Club form and click the edit button. Then click the CRUD tab:



There are four mechanisms to control form Create, Read, Update and Delete. We are only interested in Delete at this stage.

Delete

The delete button lives together with the update button top right on the form:

It can be hidden, its caption set and its icon set.

Apply

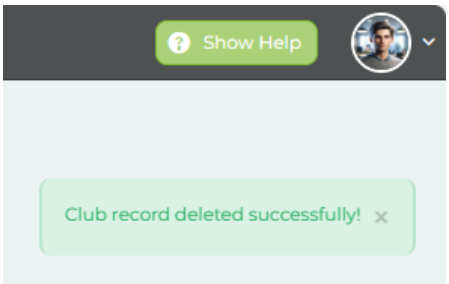
Apply any changes to persist them before testing.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Clubs/Teams item. Drill down into the last test club you created to open the form.

Press the **Delete** button. You will be prompted to confirm the deletion.

The form record deletion should be confirmed when the form is closed:



Footballer Search Form

The third and final entity we will lookup is footballers.

A footballer will play for one club at a time, however may move between clubs. A footballer will also be assigned a single nationality in this sample database, but in the real-world, people can have dual-nationality. The restdb.io sample table is called Players, with foreign keys to Position, Teams and Nationality.

We know from the ReST API what the end points and associated security keys we need to get started. We will always start by reading the data, then displaying it, before moving on to editing, creating and finally deleting data.

Differences from Previous Lookup Forms

This lookup form is different from previous lookup forms for three main reasons:

Search Criteria Component

Instead of just a grid with column filters, we will be using a search criteria component which will allow for multiple fields to be searched before populating the grid. This is particularly useful for very large data sets where pulling the entire dataset into the client-side device is not practical because it would take too long and consume too much computing resource.

Paginated Grid

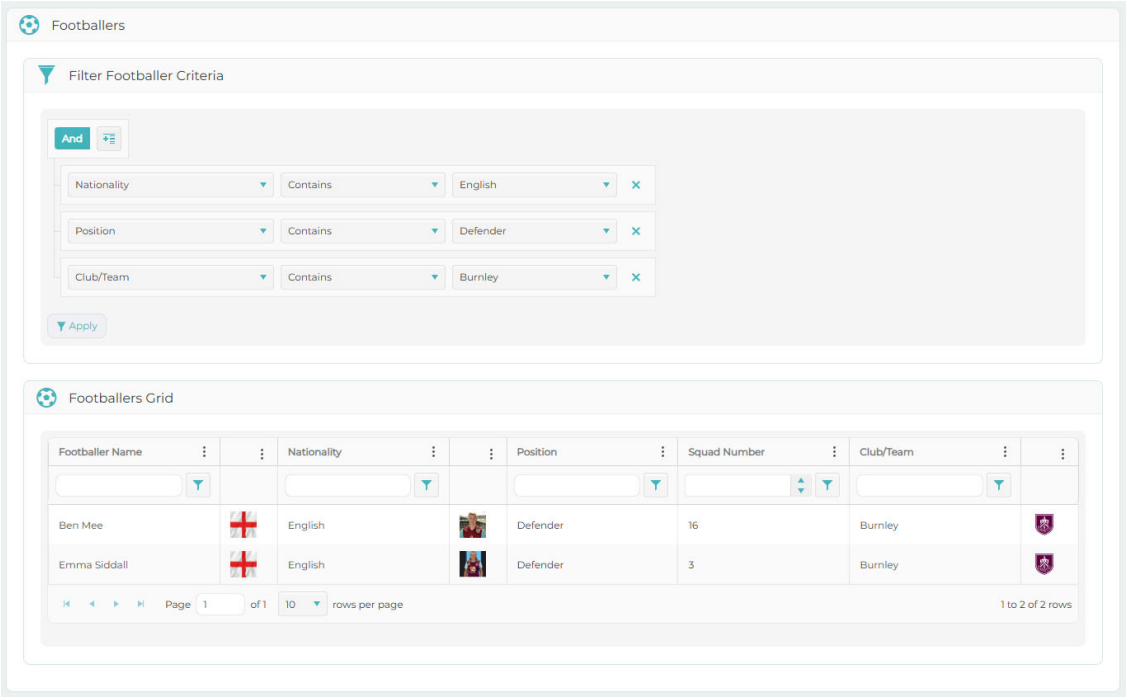
We will be using a technique known as 'pagination' which is where pages of data consisting of a small number of records is pulled from the ReST API 'on-demand'. This drastically increases performance for large data sets reducing the time to load data and keeping client-side computing resource to a minimum. We can also filter and sort data on the server-side so that the grid is responsive at all times.

Paged Data Source Request

Of course the server-side ReST API end-point must also be capable of providing paged data sets. It will typically have URL parameters for specifying the number of records per page, and which page to return. It will know how many records are in the entire data set, as this is how the client-side can calculate and display paged data.

Example

This screenshot of the Footballer Searchform we will build shows the search criteria component above the grid component and how field values are selected before applying the search to the grid:



We will now show how this sophisticated entity lookup/search form is constructed.

Custom Variables for Security Keys

We have already [set this up here](#).

Custom Variables for Key Fields

Data source requests will return a data table of rows with fields. Each row will typically have an identifier for example in this sample the Footballer or Player ID. This will be a long random unique string of characters or numbers generated by the back-end database when records are created.

In order to handle the selection of rows, we need to create a custom variable which will be dynamically set when the end-user selects a specific record in a form or a grid or a field. We will assign this variable to the key field in the data source request.

Because we know that the restdb.io Rest API request will return footballers, we previously created a custom variable called "Footballer-ID" which we will assign to the field later.

Search Criteria Data Source Requests

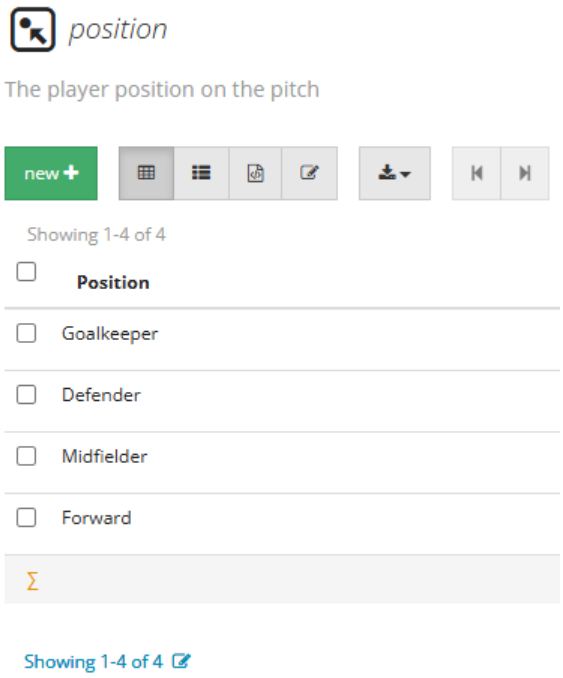
We know from the design of the sample ReST API that we want to provide the end-user with 5 fields with which to search footballers:

Field	Description
Club/Team	The club that the footballer currently plays for. This is a lookup field.
Position	The position on the pitch where the footballer plays. This is a lookup field.
Nationality	The nationality of the player. This is a lookup field.
Name	The name of the player.
Squad No.	The squad number of the player.

So far, we have created data source requests for all lookup fields except positions.

Positions

In the sample ReST API we created a `position` table in restdb.io:



Each player record is linked to a position. In order to therefore allow position to be used in a lookup field, we need to create a data source request.

New Data Source Request

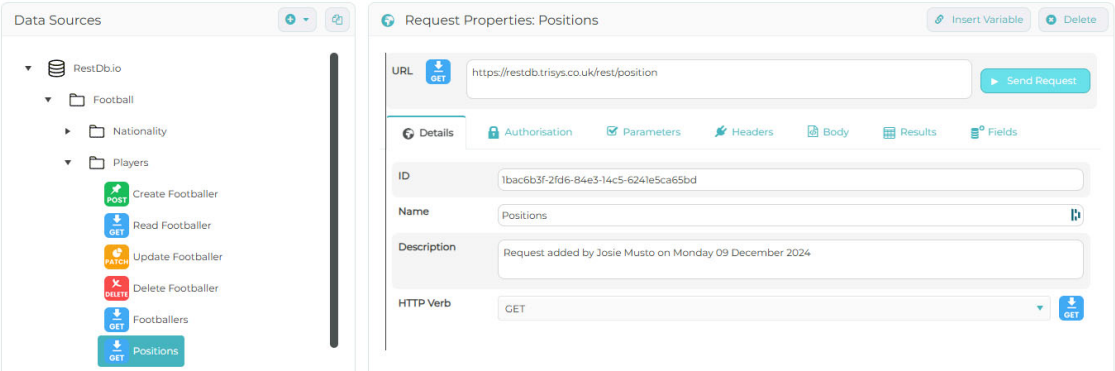
Open App Studio, then open the data source configurator.

Create a data source, and a sub-folder hierarchy like this:

`RestDB.io/Football/Players`

Now create a new data source request called "Positions" using the URL:

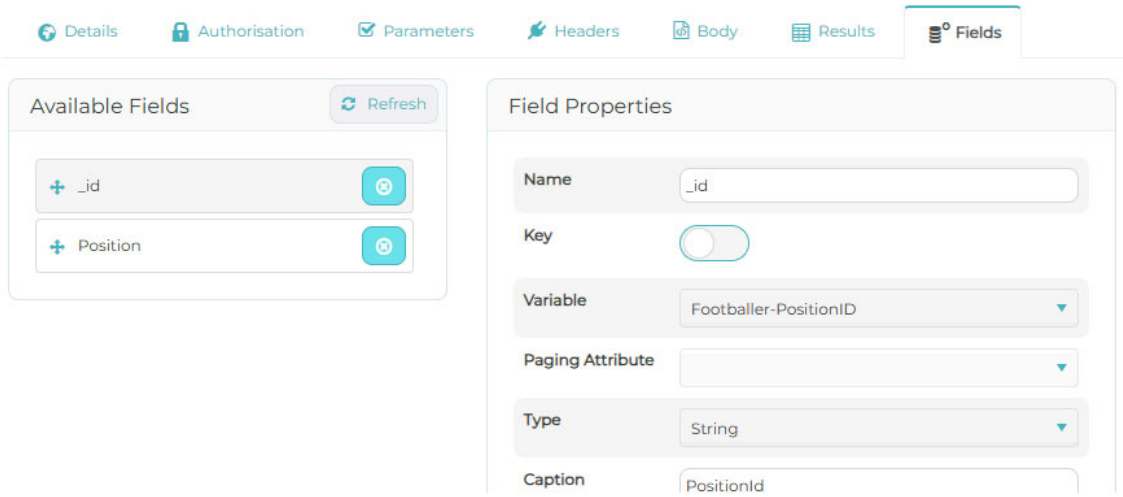
`https://restdb.trisys.co.uk/rest/position` ↗ `:`



Click the Send Request button to return a list of positions and view in the Results-->JSON tab:

We can see that this request is returning all 4 positions.

Click on the Fields tab in order to manage field properties:



Now assign the _id field to the `Footballer-PositionID` variable. If this does not exist, create it using the custom variables configurator.

Paginated Data Source Request

We have previously created a data source request to integrate the list of footballers pulled from the ReST API with the Nationality form and the Club form. We will NOT be using this specific data set because it does not need pagination, whereas we will be needing pagination for the grid we will be using on this footballer lookup form.

Data Source Configurator

Open App Studio, then open the data source configurator.

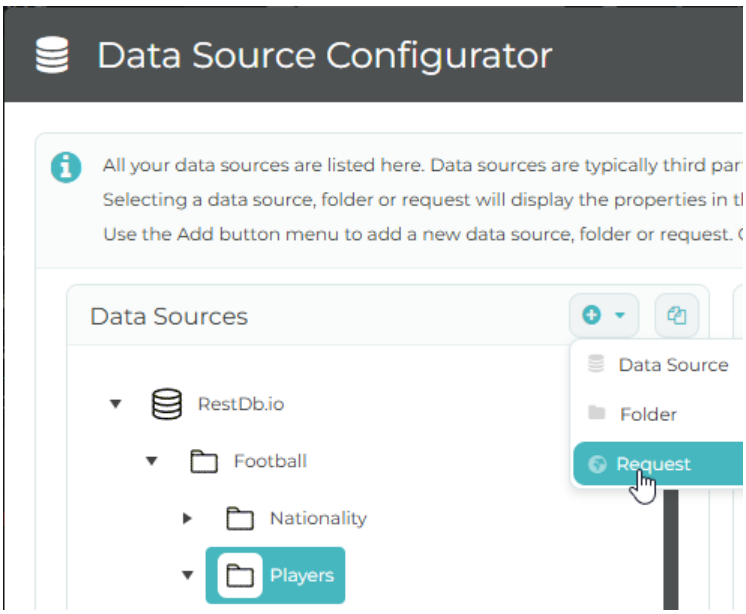
Navigate to this sub-folder hierarchy:

`RestDB.io/Football/Players`

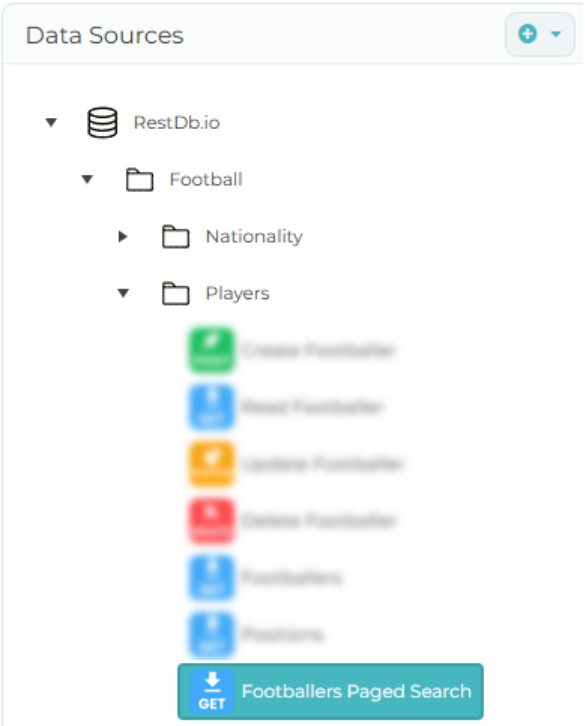
This unambiguously defines that we have a restdb.io data source inside which we have the Football industry depicted as a folder. We will now create a paginated data source request for footballers.

New Request

Create a new request using the drop down menu:



Name this request "Footballers Paged Search":



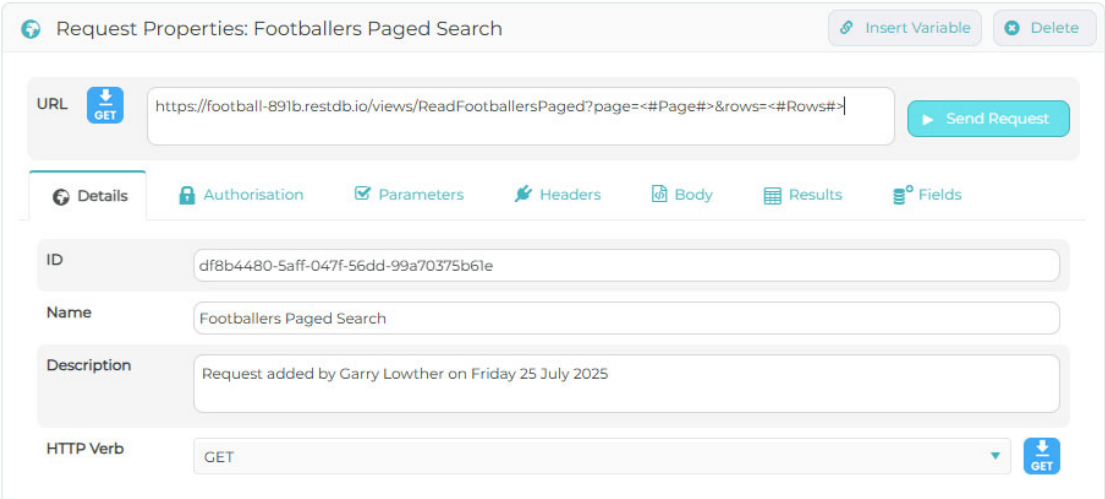
Edit Request Properties

Edit the new request properties to set the URL to be:

`https://football-891b.restdb.io/views/ReadFootballersPaged?page=<#Page#>&rows=<#Rows#>` ↗

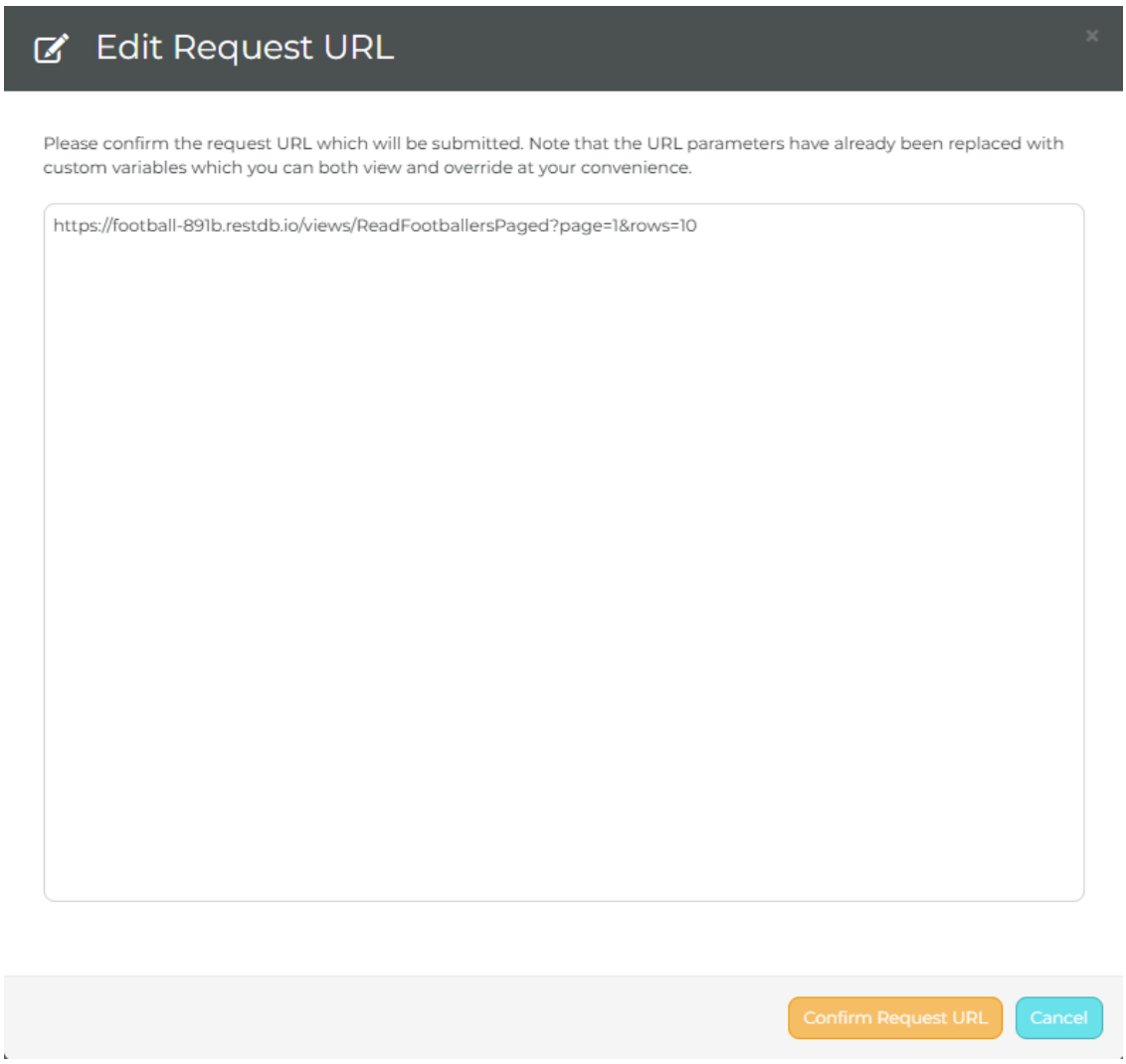
Note how we have a complex URL with arguments and custom variables? These arguments are defined by the sample ReST API.

It should now look like this:



Send Request

You should now test the request by clicking the Send Request button. You will be prompted to confirm the URL being requested:



Notice how the custom variables have been automatically replaced? You can edit these each time you send a request to test that the back-end ReST API is working as expected.

Results

After the ReST API request has been sent, the Results tab will be selected and the JSON tab will show the columns and this data:

Request Properties: Footballers Paged Search Insert Variable Delete


```
    },
    "DataTable": {
        "List": [
            {
                "TotalNumberOfRecords": 122,
                "PageSize": 10,
                "TotalNumberOfPages": 13,
                "PageNumber": 1,
                "HasNextPage": "True",
                "HasPreviousPage": "False",
                "SortField": "name",
                "FilterField": null,
                "FilterValue": null,
                "List": [
                    {
                        "ID": "6863ef8078badf6500137ef6",
                        "Name": "Alessia Russo",
                        "PhotoUrl": "https://football-891b.restdb.io/media/6863ef8078badf6500137ef5",
                        "PhotoID": "6863ef8078badf6500137ef5",
                        "TeamName": "Arsenal",
                        "TeamID": "674d85df050c58540003f7d4",
                        "TeamBadgeUrl": "https://football-891b.restdb.io/media/67fa56e478badf650005503
5",
                        "TeamBadgeID": "67fa56e478badf6500055035",
                        "Position": "Forward",
```


We can see that only a page of footballers data from the ReST API is being returned, including the photo, nation flag and club badges as URL images. This is because our restdb.io view is able to map its media files onto a two-dimensional table for easy consumption by client-side applications.

The next interesting thing about this data set are these pagination fields:

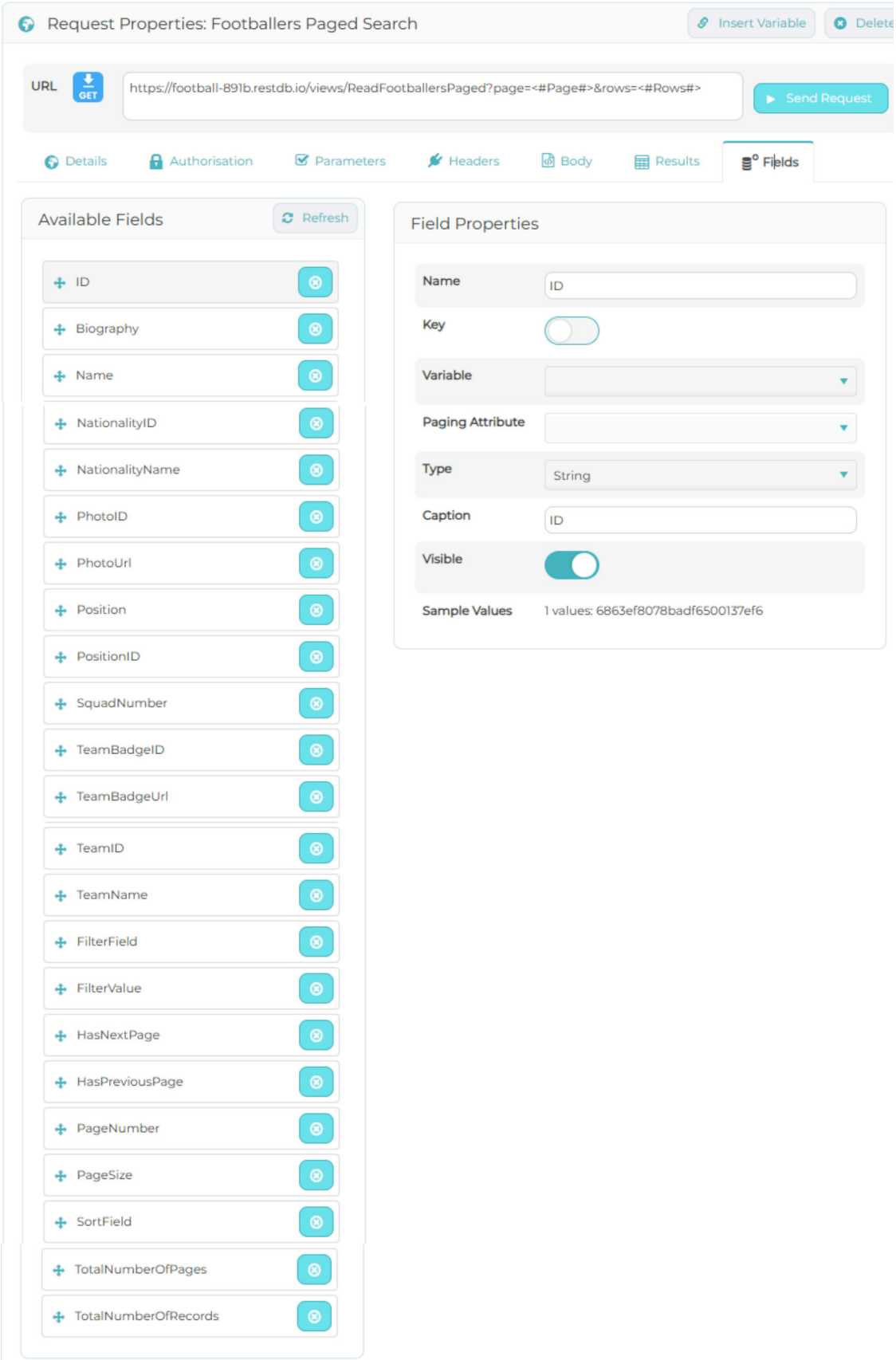
```
{
    "DataTable": {
        "List": [
            {
                "TotalNumberOfRecords": 122,
                "PageSize": 10,
                "TotalNumberOfPages": 13,
                "PageNumber": 1,
                .....
            }
        ]
    }
}
```

This example tell us that there are 122 footballers in the entire data set, and that if the page size (number of records per page) is 10, then there will be 13 pages in total. It also says that this data set is page 1 i.e. the first 10 records of the 122 record data set.

Fields

If we scroll back up to the top and click the left Fields tab, we should see a list of the fields accompanying the data.

Clicking on the top right Fields tab will show these Available Fields in a vertical list to the left:



Fields can be re-ordered using the left drag icon, or removed using the right delete button. Selecting any field will show its corresponding properties to the right.

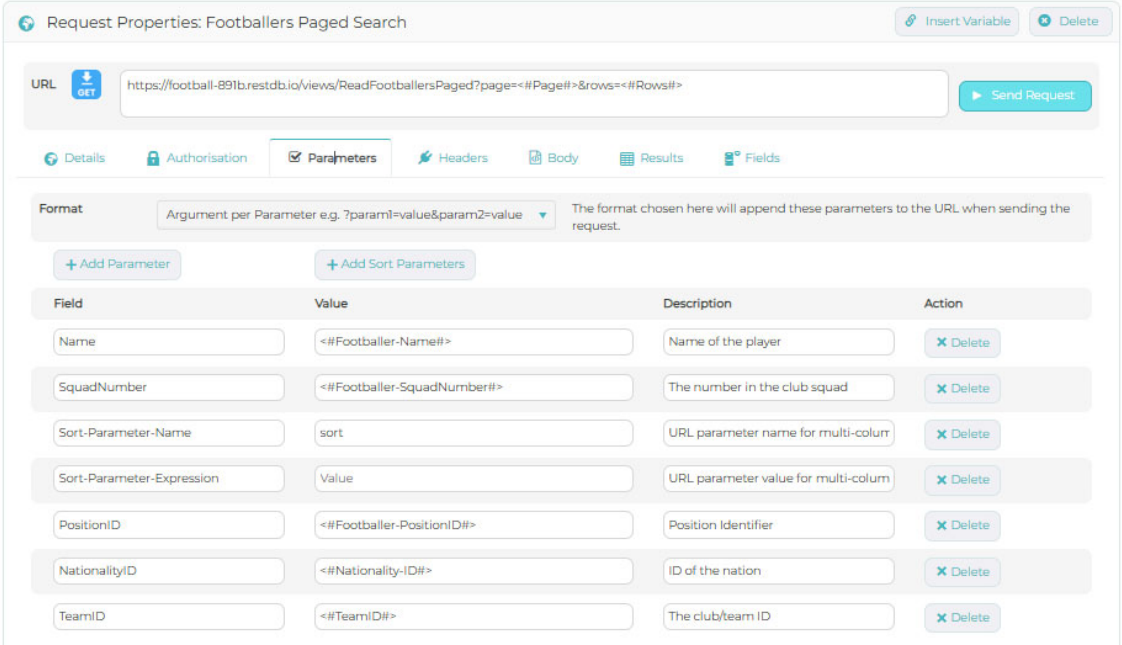
There are may fields in this complex paginated data set, so here is a table specifying how to configure those of interest:

Field	Properties
ID	Key `Checked`, Variable: `Footballer-ID`, Visible: `Unchecked`
PhotoUrl	Type: `Image URL`, Visible: `Checked`
Name	Variable: `Footballer-Name`, Caption: `Name`, Visible: `Checked`
NationalityFlagUrl	Type: `Image URL`, Visible: `Checked`
NationalityName	Caption: `Nationality`, Visible: `Checked`
Position	Variable: `Footballer-Position`, Caption: `Position`, Visible: `Checked`
SquadNumber	Variable: `Footballer-SquadNumber`, Caption: `Squad No.`, Visible: `Checked`
TeamBadgeUrl	Type: `Image URL`, Visible: `Checked`
TeamName	Variable: `TeamName`, Caption: `Club/Team`, Visible: `Checked`
TotalNumberOfRecords	Paging Attribute: `Total Record Count`, Type: `Number`, Visible: `Unchecked`
TotalNumberOfPages	Paging Attribute: `Total Page Count`, Type: `Number`, Visible: `Unchecked`
PageNumber	Paging Attribute: `Page Number`, Type: `Number`, Visible: `Unchecked`
PageSize	Paging Attribute: `Records per Page`, Type: `Number`, Visible: `Unchecked`

Any of the fields not listed above should be set to be invisible.

Parameters

In order to allow the search criteria fields to be used in the data source request URL, we need to create a number of parameters which will be appended to the URL dynamically when the search is applied.



Each parameter field is the name of a URL argument e.g.

```
/ReadFootballersPaged?TeamName=Arsenal&Position=Defender
```

We effectively map these ReST API URL parameters to custom variables and have them dynamically injected into the URL when the search criteria is applied.

Use the **Add Parameter** button to add those below with custom variables, and the **Add Sort Parameters** to add Sort-Parameter-Name and Sort-Parameter-Expression.

These fields are documented as follows:

Field	Value / Custom Variable	Description
Name	`<#Footballer-Name#>`	Name of the player
SquadNumber	`<#Footballer-SquadNumber#>`	The number in the club squad
Sort-Parameter-Name	`sort`	URL parameter name for multi-column sorting
Sort-Parameter-Expression		URL parameter value for multi-column sorting
PositionID	`<#Footballer-PositionID#>`	Position identifier
NationalityID	`<#Nationality-ID#>`	ID of thes nation
TeamID	`<#TeamID#>`	The club/team ID

We will assign this data source request to both the search criteria and grid components below.

Footballers Lookup Form

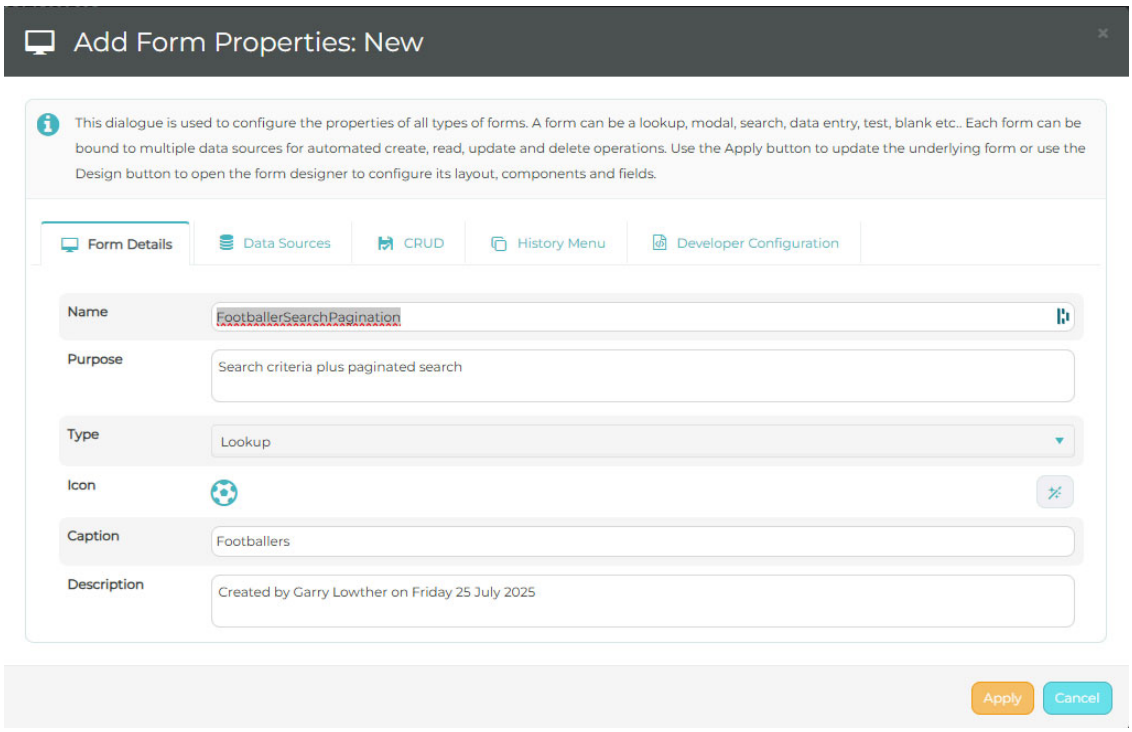
Now that we have the custom variables and data source requests to facilitate searching for footballers, we need to create a lookup form upon which to display the search criteria and a grid showing all matching footballers.

Create Form

Open App Studio then select the Forms configurator.

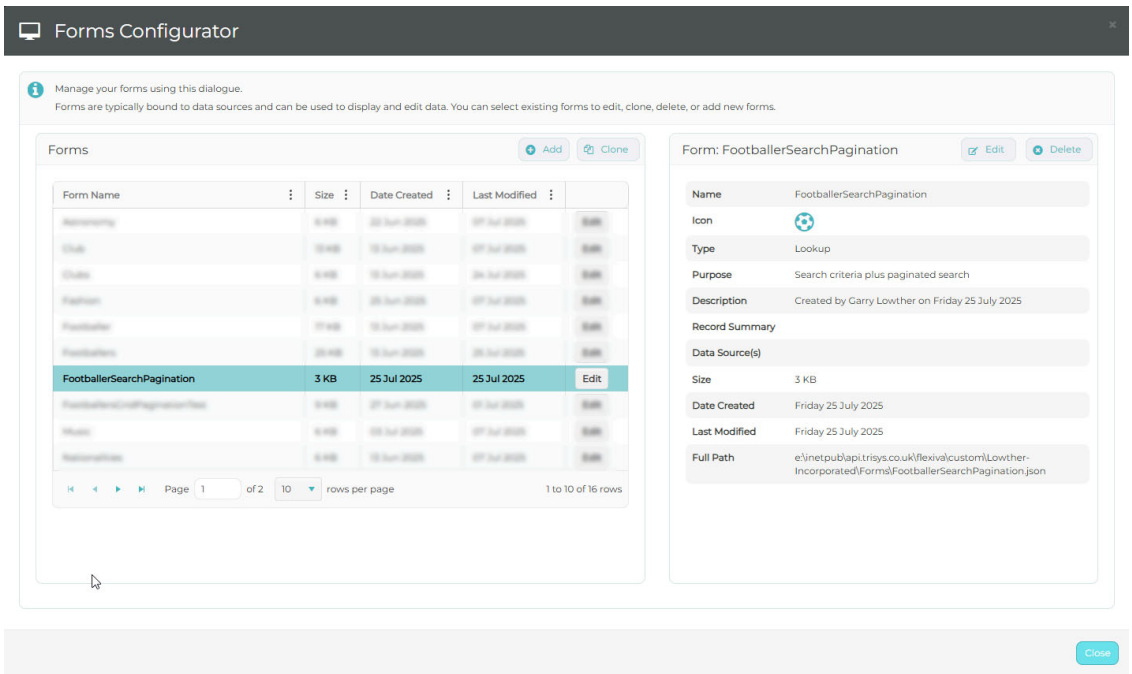
Add

Use the Add button to create a new form called "FootballerSearchPagination". In the form properties make sure that this form is of type "Lookup":



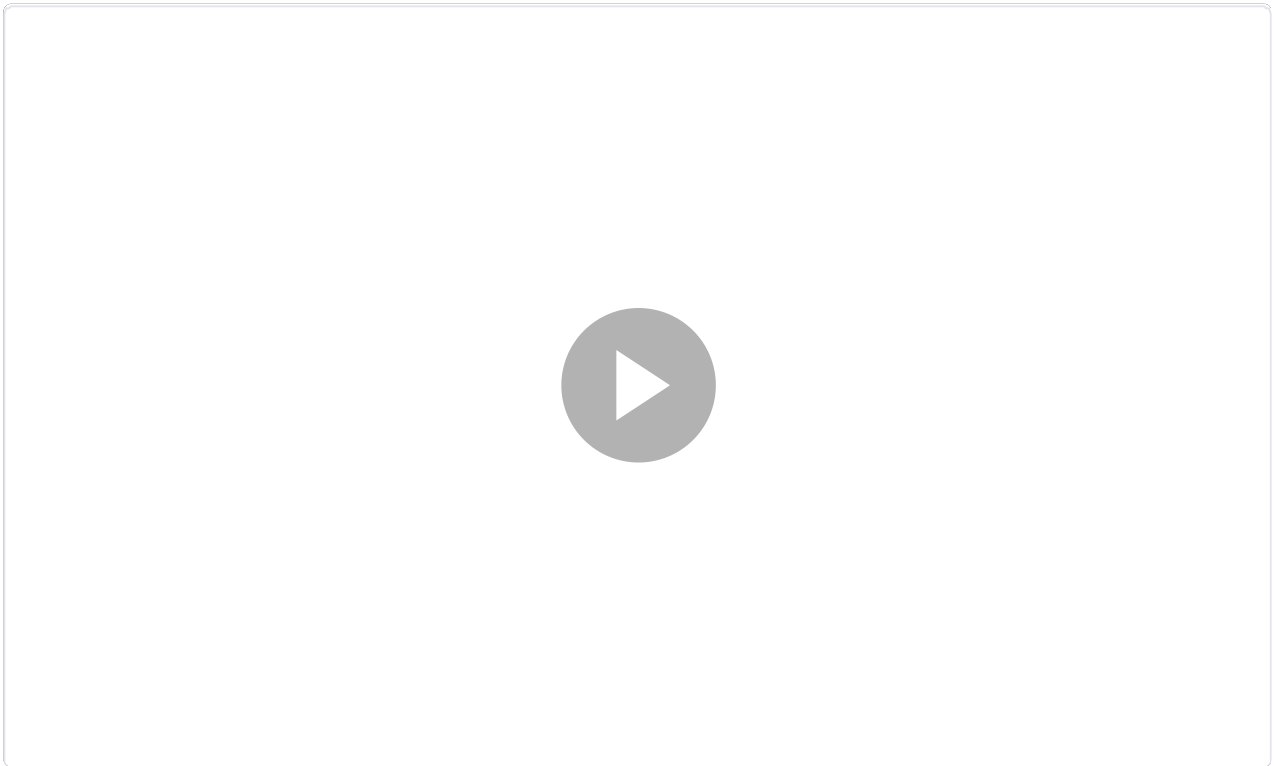
You do not need to assign a data source to the form as we will do this in the form designer when we add a Grid component.

Click **Apply** to save the new form:

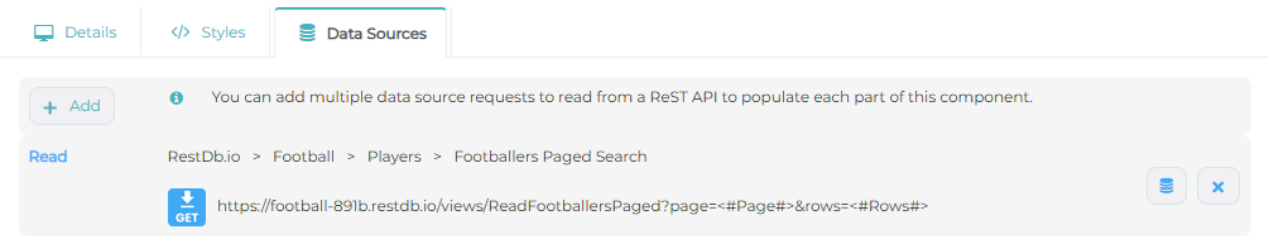


Design

Open form designer from the form properties popup, and drag a search criteria component into the first panel on the Main Top Region tab:



Click on the Search Criteria component to open the component properties. The Data Sources tab will probably popup a data source selection dialogue where you should choose this specific paginated data source request created earlier:



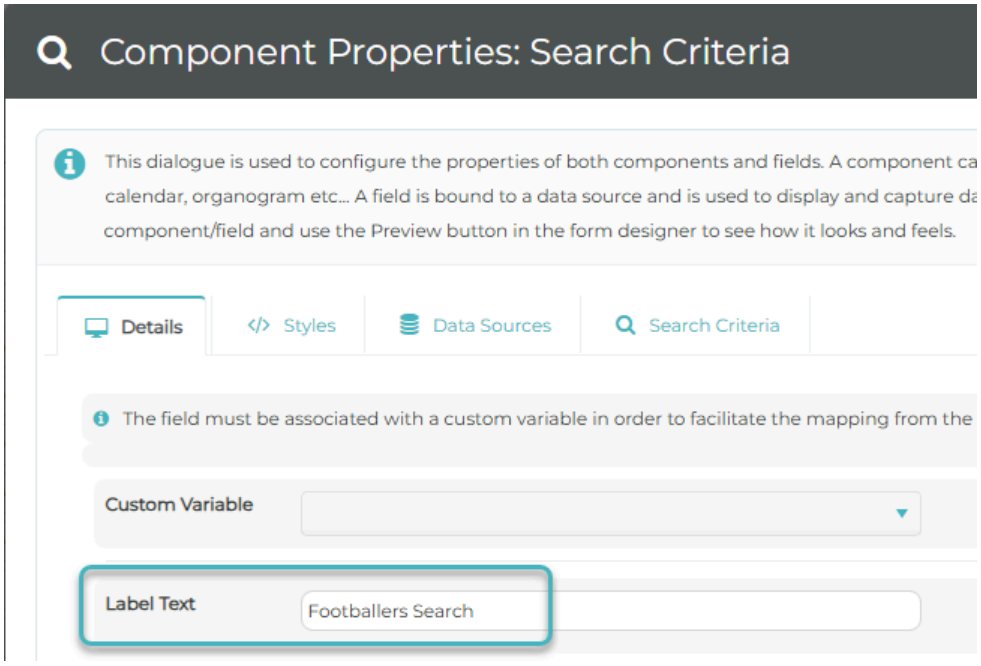
Click the **Apply** button now. Also Save the form design to persist these changes.

Search Criteria Configuration

Select the Search Criteria component on the form designer to configure it.

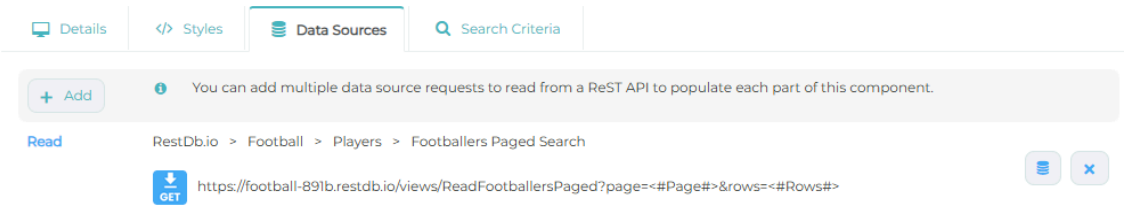
Label Text

Set the label text to allow easier identification of this later:



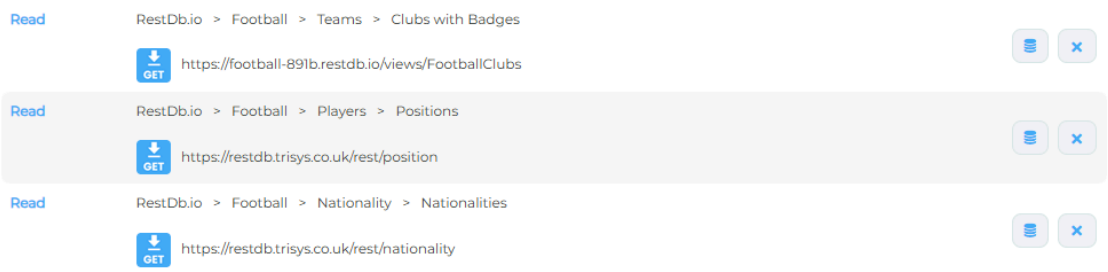
Additional Data Sources

Click on the Data Sources tab:



There is only the master data source configured. We now need to add an additional data source request for each of the lookup fields we want to use to search the footballers.

Use the Add button to add the following 3 additional data source requests:

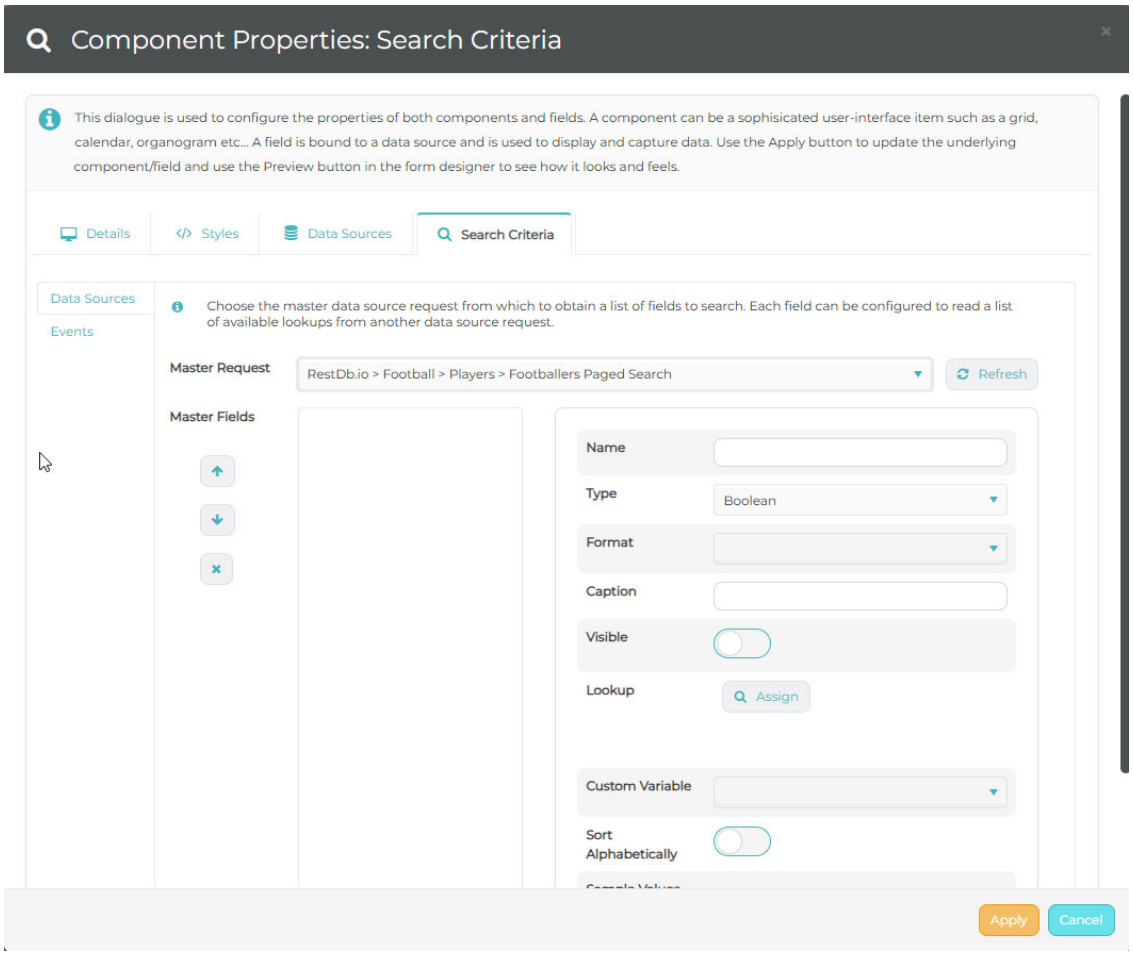


Once these have all been added, we can now start assigning these to fields.

In order to 'play-safe' you should Apply your changes, then save the form design to persist these data source requests, then re-open the search criteria properties modal popup.

Search Criteria Tab

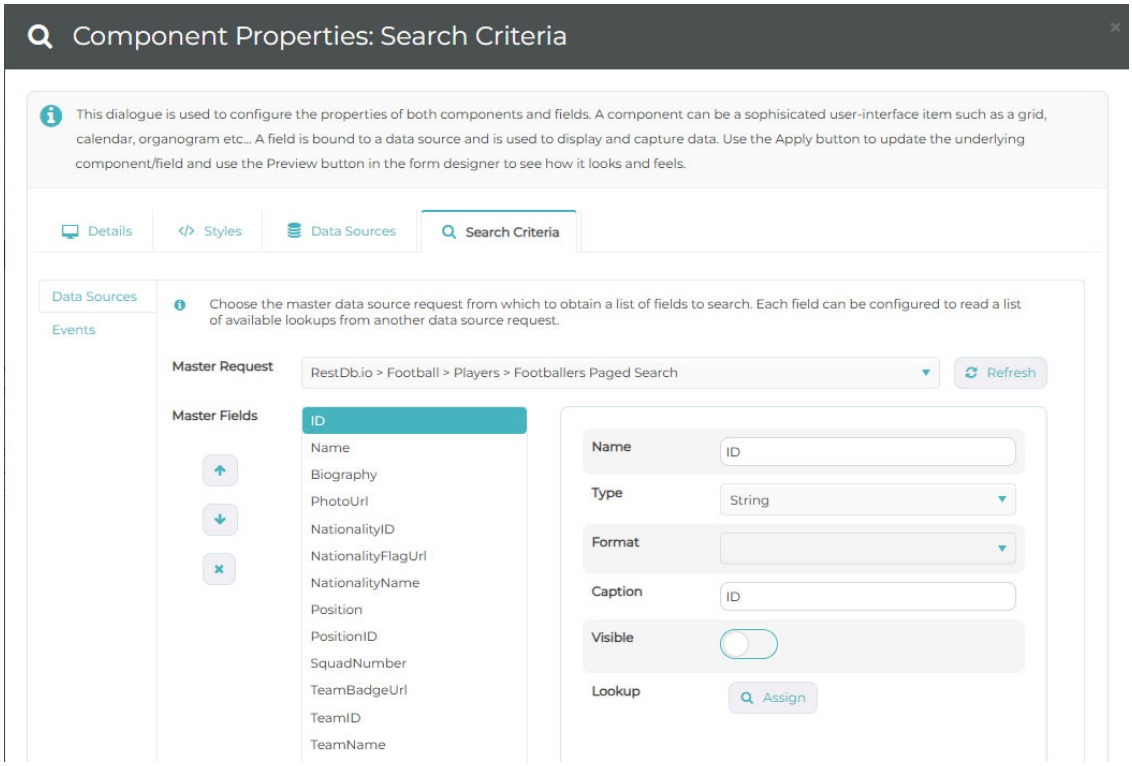
Click on the Search Criteria tab:



The Master Request should be set to the correct paginated data source request "Footballers Paged Search" we created earlier. We have no master fields listed. We must refresh the list using the **Refresh** button:

Master Fields

The fields should now be visible:



Re-Position

In the Master Fields left tab, the fields previously retrieved from the master request ReST API are displayed in the order you designed. Re-order these by using the up/down arrows to be the order you wish the fields to be selected by the end-user at run-time.

Note that the order of the columns displayed in the grid component will be configured later.

Delete Fields

Some of these fields can be safely removed if there are too many fields being displayed.

Text and Number Fields

We can search for any column which is text or number by assigning a custom variable. These are as follows:

Master Field	Properties
Name	Visible: `Checked` Caption: `Name` Custom Variable: `Footballer-Name`
SquadNumber	Visible: `Checked` Caption: `Squad No.` Custom Variable: `Footballer-SquadNumber`

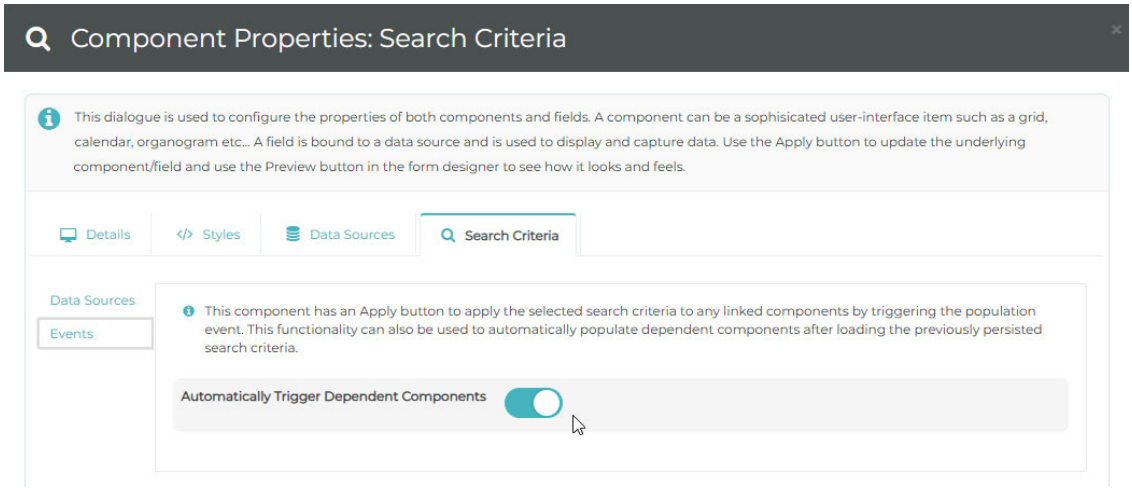
Lookup

We will now configure the fields which we want to be available in the search criteria component at run-time. These should be set to be Visible and all other fields should be hidden. Note that ReST API's will typically use identifiers rather than free text URL parameter arguments. Our sample restdb.io ReST API uses ID's also.

Master Field	Properties
NationalityID	Assign to: `RestDb.io > Football > Nationality > Nationalities > Id` Custom Variable: `Nationality-ID` Data Field: `_Id` Display Field: `Name`
PositionID	Assign to: `RestDb.io > Football > Players > Positions > Position` Custom Variable: `Footballer-PositionID` Data Field: `_Id` Display Field: `Position`
TeamID	Assign to: `RestDb.io > Football > Teams > Clubs with Badges > ID` Custom Variable: `TeamID` Data Field: ID Display Field: `Name`

Events

The Events tab is available on the left beneath the Data Sources tab:



This component has an Apply button displayed at run-time to apply the selected search criteria to any linked components by triggering the population event. This functionality can also be used to automatically populate dependent components after loading the previously persisted search criteria.

Check the **Automatically Trigger Dependent Components** check box to force the run-time Apply button fire automatically when the previously retained last search criteria is re-populated. This has the effect of immediately re-populating the attached grid when the form loads.

Apply

Apply the grid properties, then Save the form design to persist the form for later use.

Navigation Bar

We can now add this lookup form to the navigation bar from where it can be opened by the end-user.

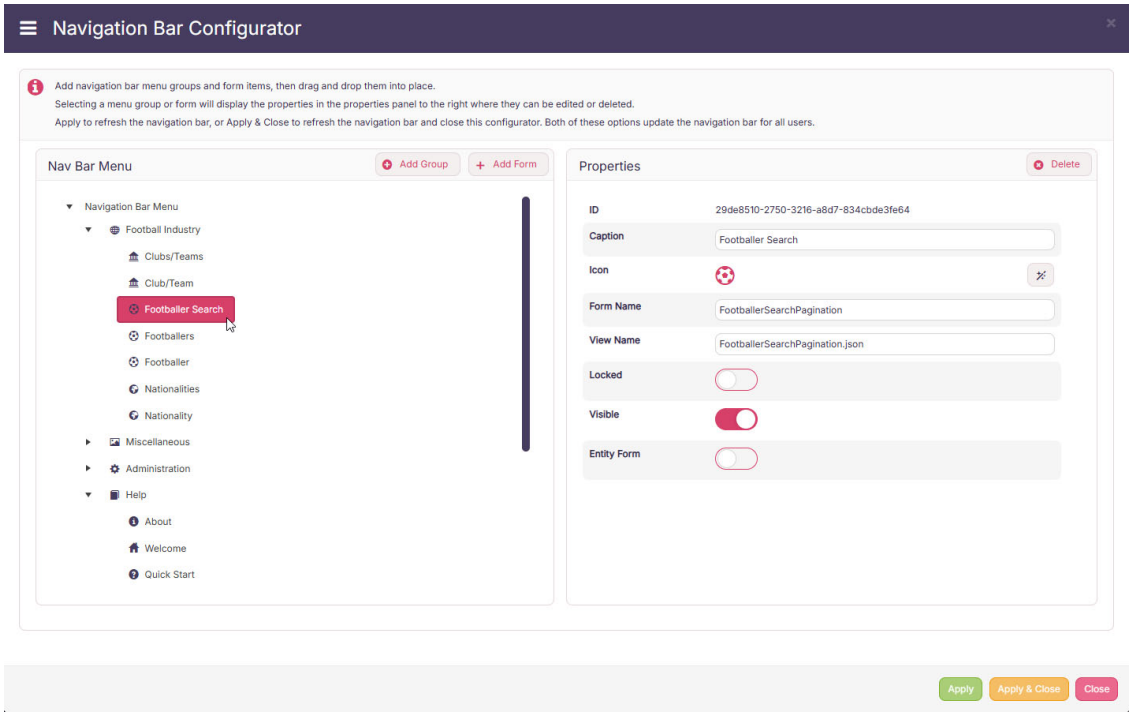
Open App Studio then select the Navigation Bar configurator.

Football Industry

The group called "Football Industry" was already created here.

Footballer Search

When the Football Industry group is selected, add a form using the **Add Form** button. This will popup a modal dialogue where you can select your newly created FootballerSearchPagination form. Clicking **Save** will show the new form menu:

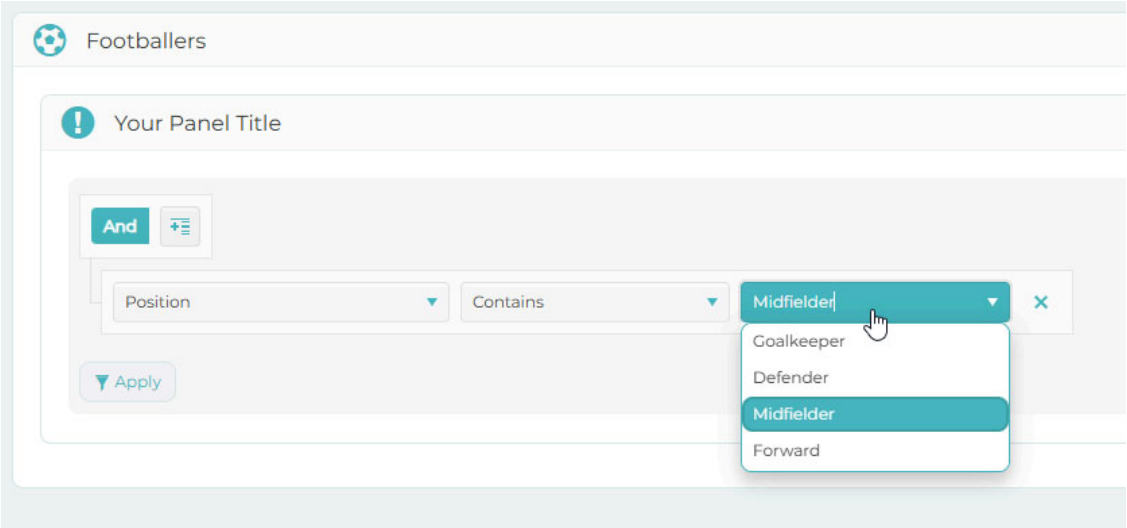


Change the Caption to "Footballer Search". Then press the **Apply & Close** button.

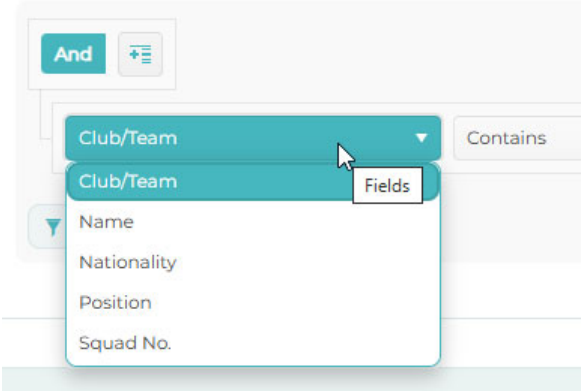
Test Search Criteria

We will now test that the Search Criteria component is working. Click on the Footballer Search in the Football Industry group on the nav bar to load the form.

You should check that the Position, Club/Team and Nationality fields can all be selected and show their respective lookups in the lookup drop down combo for example:



You should also check that all requested fields are available for selection:



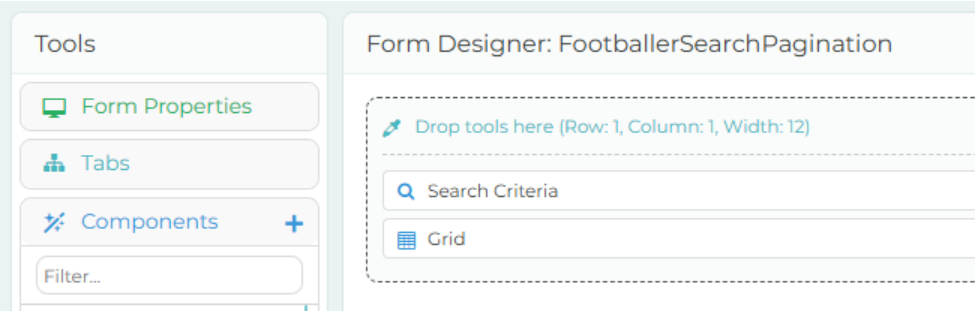
Footballers Lookup Form: Grid

Now we will design a grid to populate all matching footballers when the end-user applies their search criteria.

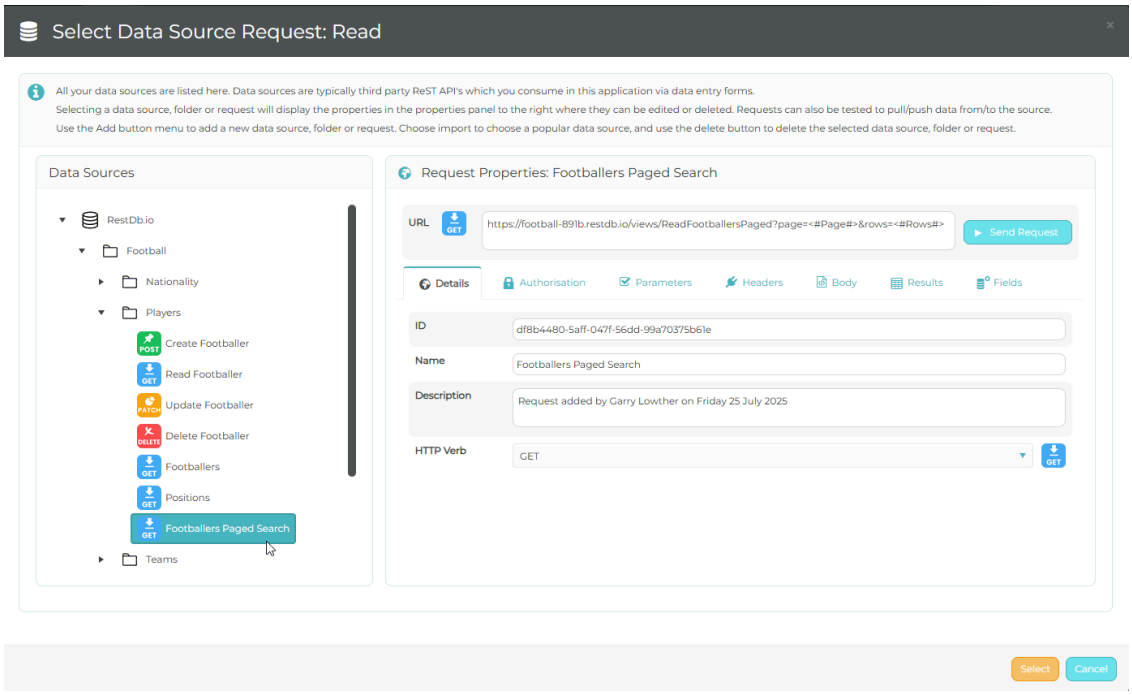
Design Form

Open App Studio then select the Forms configurator, then select the FootballerSearchPagination form and Edit, then Design to open the form designer.

We will keep things simple, so open the Components toolbar and drag the "Grid" component beneath the Search Criteria component:



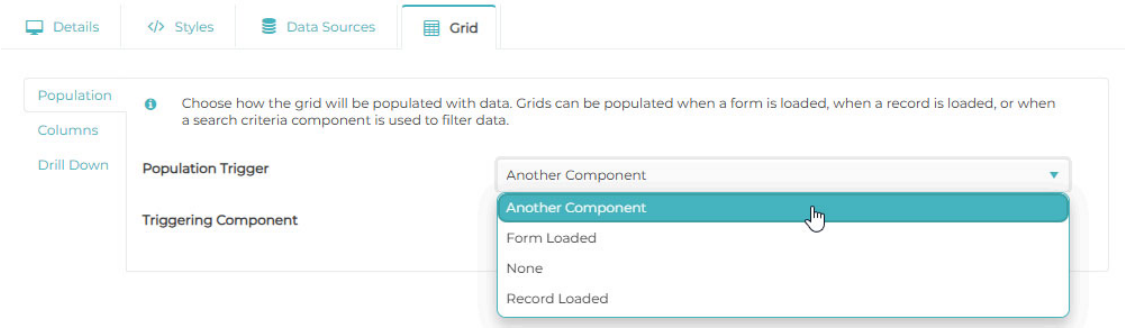
Click on the Grid which will open the "Select Data Source Request: Read" popup form. This is to encourage you to select the data source request which will populate the grid:



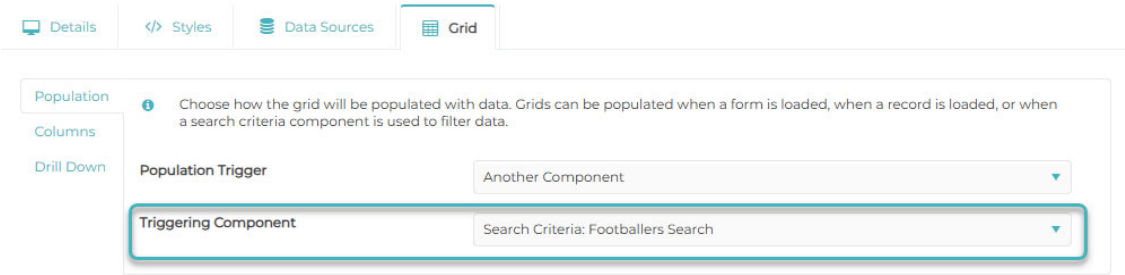
Select the "Footballers Paged Search"

Population

Choose "Another Component" in the **Population Trigger** field:



The Triggering Component field will automatically select the "Footballers Search":

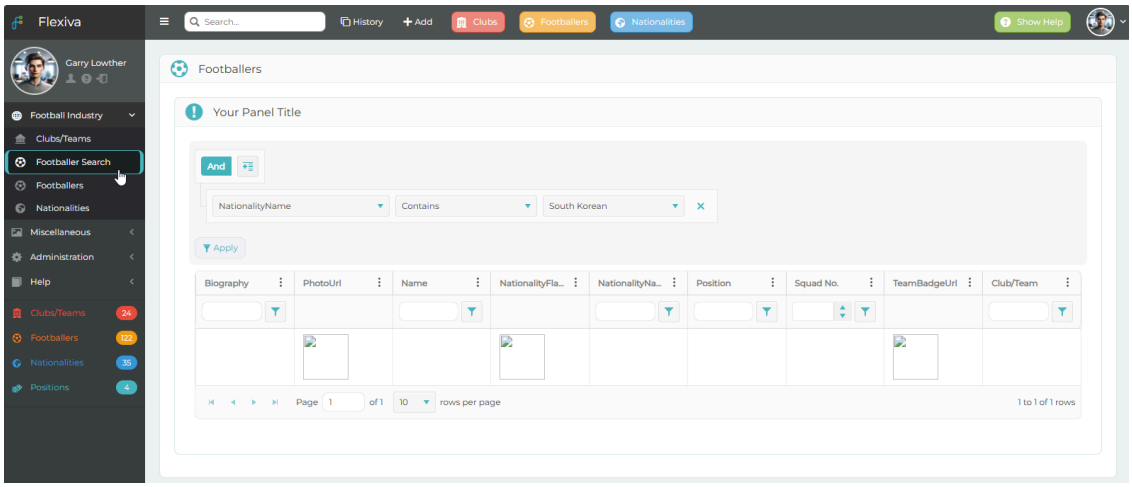


Apply

Apply these grid properties, then Save the form design.

Test Latest Configuration

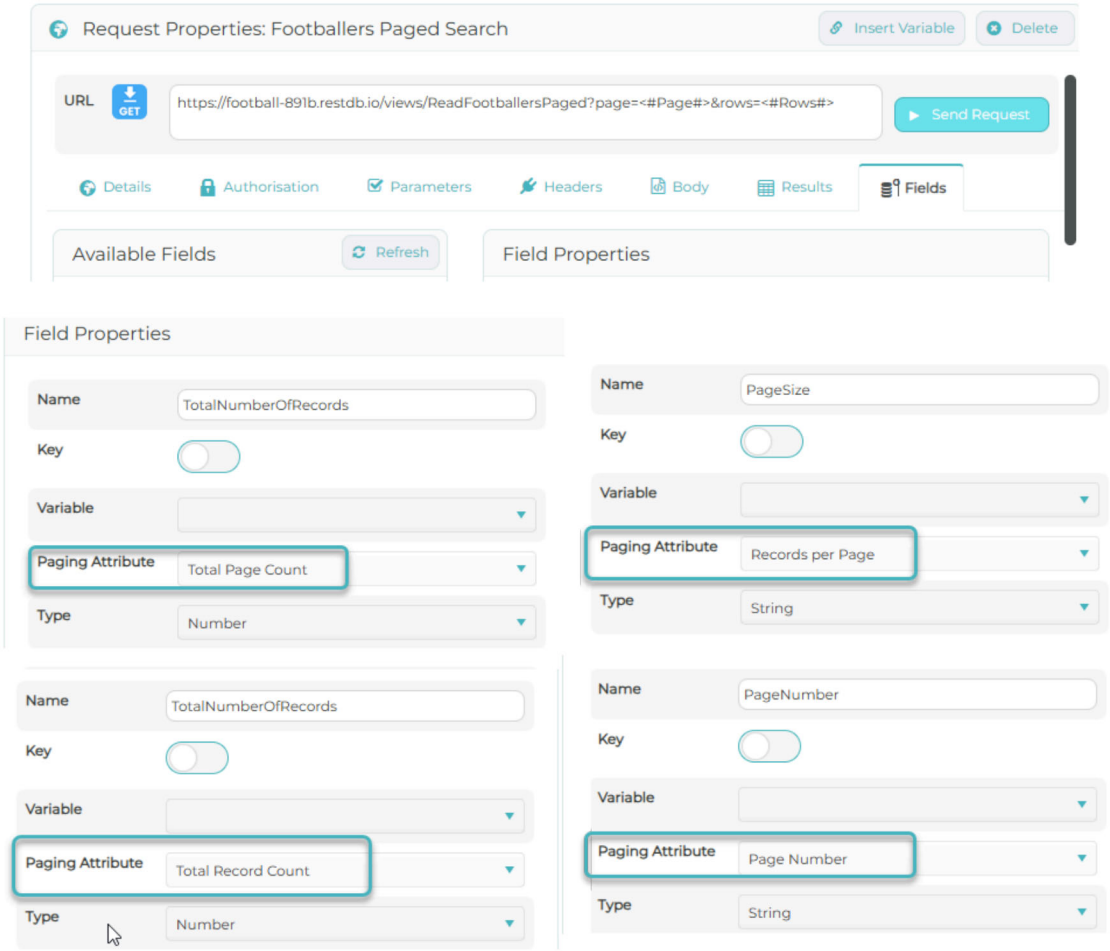
Click the Footballer Search in the nav bar to open the form, then select a nationality name, and click **Apply**. A record may appear:



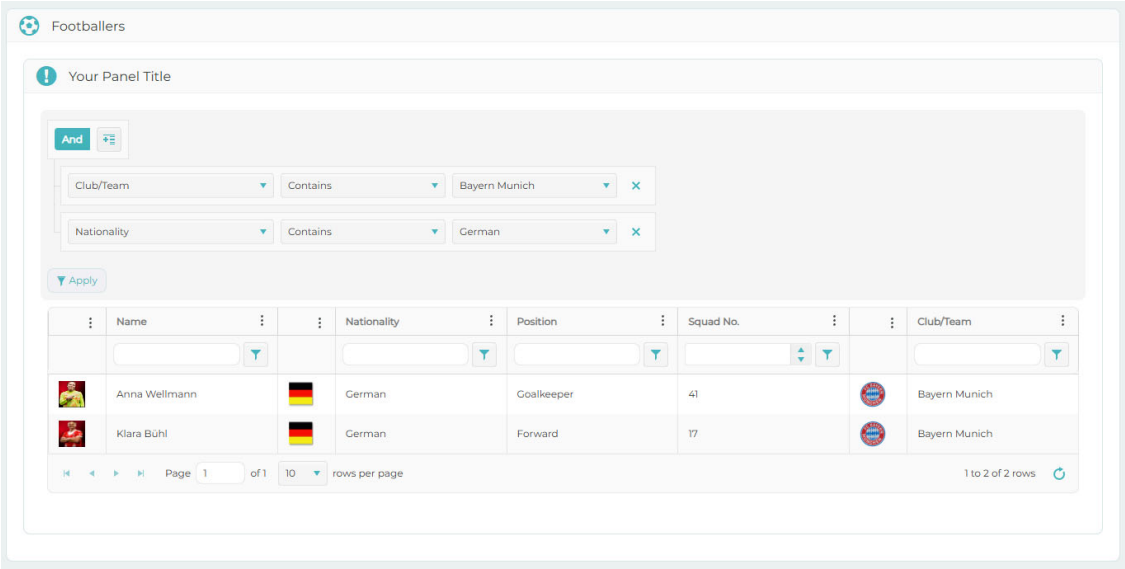
This is indicative that the search criteria component is correctly wired up to the grid, however the grid columns need to be configured.

First thing to double check is whether the pagination columns are correctly setup?

These should be setup as shown:



Once these are re-configured if necessary, and operational, re-opening the form should start displaying data correctly:

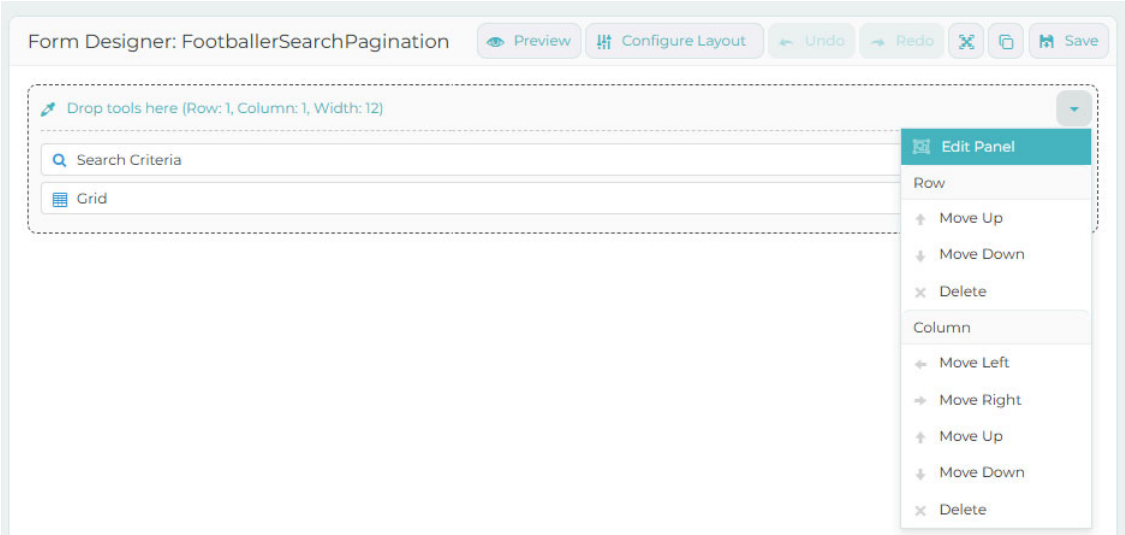


You can edit your grid columns to set sizes of images, column widths and ordering to make the grid look exactly how you want it.

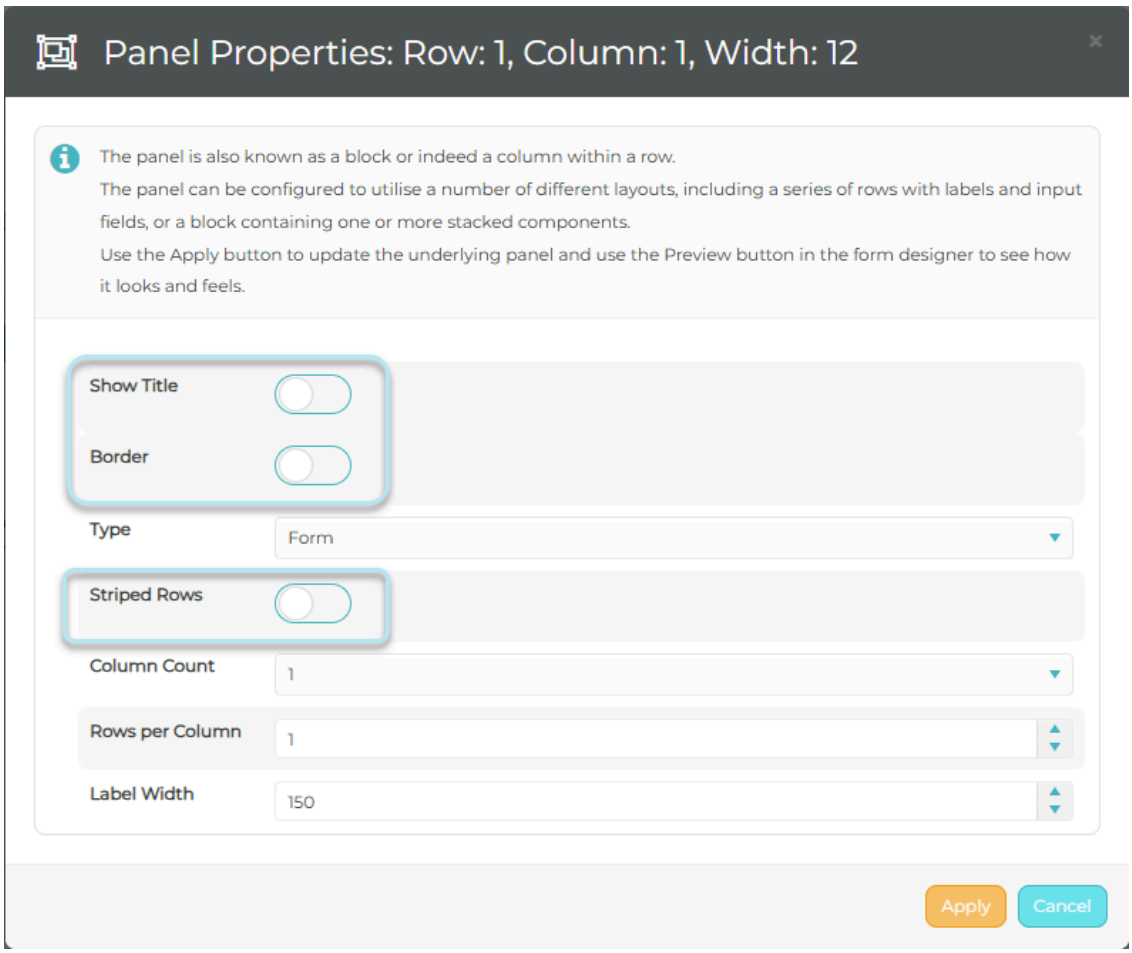
Tidy Up Form

We have the most important features working now, so the final things to do are to tidy the form up.

We do not need the containing panel to be visible, so open App Studio, edit the form design and edit the panel:



Hide the title, turn off the border, and turn off striped rows:

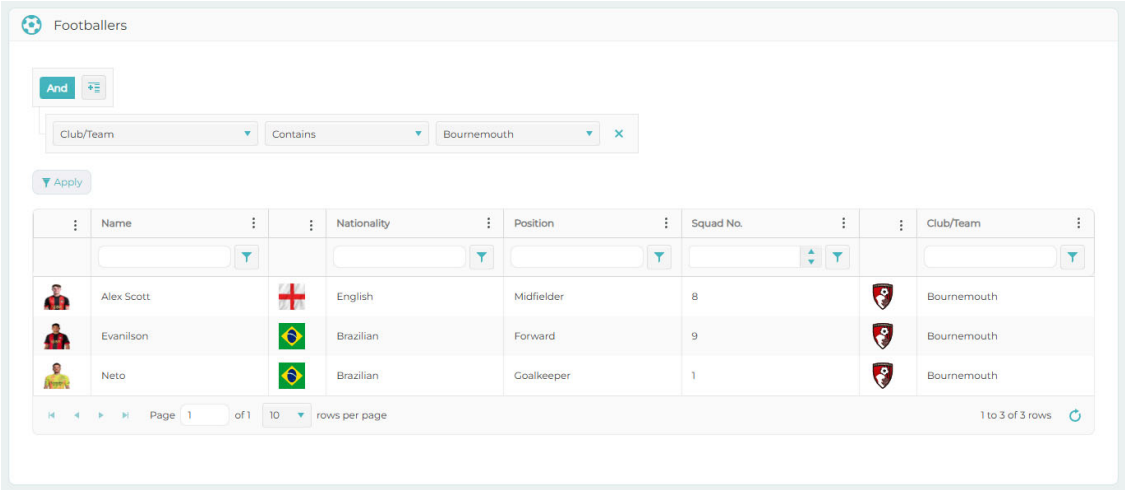


Apply the changes, then save the form design.

Test

You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Footballer Search item.

You should now see the lookup form with the search criteria displayed above the data grid:



Test that column filtering and sorting works as expected.

We will revisit this form later in order to configure drill down.

We are now ready to create a data entry form configured for creating, reading, updating and deleting footballer records.

Footballer Data Entry Form

The third entity data entry form is a footballer record.

Having previously created custom variables and a data source request to display a lookup form showing, filtering and sorting footballers, we are now moving on to creating a data entry form where a footballer can be created, read, updated and deleted (CRUD).

In our ReST API sample data set, the underlying restdb.io footballer is the `Players` table.

The process for all CRUD operations typically starts with READ, as this involves designing the data entry form, and drilling down into it. Here are the four CRUD phases in the order we will configure them:

READ

- Add a 'read' data source for reading a single footballer record
- Link the key footballer record identifier to this ReST API end-point if necessary
- Create a data entry form
- Assign this 'read' data source to the data entry form
- Design this form and drag fields from this data source
- Add this form to the navigation bar
- Configure drill down to the lookup form grid component
- Test that we can lookup footballers and drill down into a data entry form showing the footballer master record
- Set the History Menu record summary
- Test that we can lookup footballers and drill down into the footballer form, and that the history menu shows the footballer name
- Add a master/detail grid to show all previous clubs/teams
- Test that we can view all previous clubs/teams whom this player represented

UPDATE

- Create custom variables for each field
- Add an 'update' data source for updating a single footballer record
- Link the key footballer record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'update' data source to the data entry form
- Configure the update button
- Test that we can update a footballer using the Update button

CREATE

- Add a 'create' data source for creating a single footballer record
- Link the key footballer record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'create' data source to the data entry form
- Configure the app toolbar Add menu to add this data entry form
- Test that we can create a footballer using the Add menu and Update button

DELETE

- Add a 'delete' data source for deleting a single footballer record
- Link the key footballer record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'delete' data source to the data entry form
- Configure the delete button
- Test that we can delete a footballer using the Delete button

Footballer Form: Read

Add a new footballer form to read and display a record.

This is the process we will follow.

READ

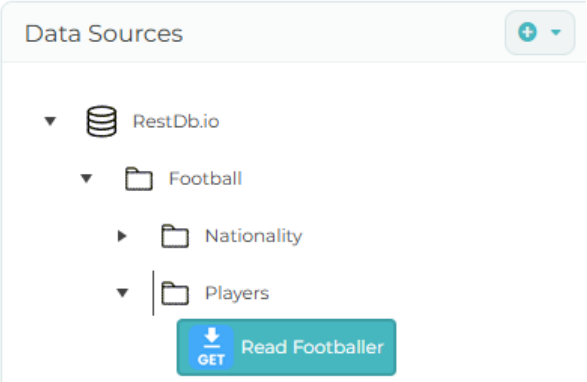
- Add a 'read' data source for reading a single footballer record
- Link the key footballer record identifier to this ReST API end-point if necessary
- Create a data entry form
- Assign this 'read' data source to the data entry form
- Design this form and drag fields from this data source
- Add this form to the navigation bar
- Configure drill down to the lookup form grid component
- Test that we can lookup footballers and drill down into a data entry form showing the footballer master record
- Set the History Menu record summary
- Test that we can lookup footballers and drill down into the footballer form, and that the history menu shows the footballer name
- Add a master/detail grid to show all previous clubs/teams
- Test that we can view all previous clubs/teams whom this player represented

Add a READ Data Source

Open App Studio, open the Data Sources configurator, select the Players folder you created previously, then use the add button menu to create a new request.

The Add New Request modal popup form shows asking you to name the request. Type a meaningful name such as "Read Footballer" and click Save.

The request will be added to your tree view beneath the Players folder:



Edit Properties

Edit the properties in the Details tab as following by referencing this list of ReST API end-points.

URL

`https://football-891b.restdb.io/views/ReadFootballers` ↗

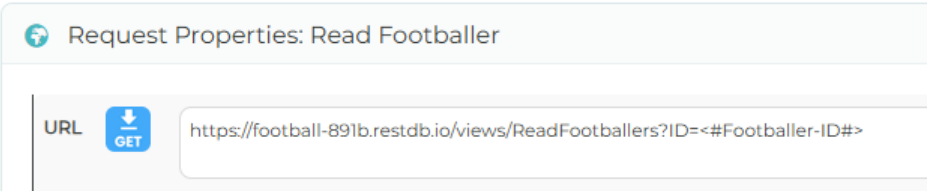
Now it is known from the documentation that this ReST API end-point has a footballer identifier property ?ID=.

We therefore type this parameter list into the URL so that it reads:

`https://football-891b.restdb.io/views/ReadFootballers?ID=` ↗

Whilst the caret is still blinking after the last character typed, click on the Insert Variable button and select this variable: `Footballer-ID`

The URL should now show the custom variable appended to the end:



HTTP Verb

The default GET verb/method is usually correct for reading a single record from a ReST API and is correct for this specific back-end end-point.

Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Send Request

Click this button on the Details tab to send the request to the ReST API.

You will be prompted to confirm the URL. The value of the custom variable `FootballerID` should show after ?ID= but if it does not, then copy this value in: `686400cf78badf6500138270` as this is the ID for Melina Loeck.

This URL should now be:

`https://football-891b.restdb.io/views/ReadFootballers?ID=686400cf78badf6500138270` ↗

Press the Confirm Request URL button.

The request should run quickly and select the Results tab and show the JSON top left tab displaying the full JSON returned from the ReST API:

```
{
    "Columns": [
        {
            "field": "ID",
            "title": "Id",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": null
        },
        {
            "field": "Name",
            "title": "Name",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "PhotoUrl",
            "title": "Photourl",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": "<img src='#: PhotoUrl #' style='width: 64px;
height: 64px;' />"
        },
        {
            "field": "PhotoID",
            "title": "Photoid",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "TeamName",
            "title": "Teamname",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
```

```
        {
            "field": "TeamID",
            "title": "Teamid",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "TeamBadgeUrl",
            "title": "Teambadgeurl",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": "<img src='#: TeamBadgeUrl #' style='width:
64px; height: 64px;' />"
        },
        {
            "field": "TeamBadgeID",
            "title": "Teambadgeid",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "Position",
            "title": "Position",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "PositionID",
            "title": "Positionid",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "SquadNumber",
```

```
            "title": "Squadnumber",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "NationalityName",
            "title": "Nationalityname",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "NationalityID",
            "title": "Nationalityid",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "NationalityFlagUrl",
            "title": "Nationalityflagurl",
            "type": "string",
            "format": null,
            "width": 70,
            "hidden": false,
            "template": "<img src='#: NationalityFlagUrl #'
style='width: 64px; height: 64px;' />"
        },
        {
            "field": "NationalityFlagID",
            "title": "Nationalityflagid",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "Biography",
            "title": "Biography",
            "type": "string",
            "format": null
```

```
                "format": null,
                "width": 70,
                "hidden": false,
                "template": "<img src='#: Biography #' style='width: 64px;
height: 64px;' />"
            }
        ],
        "DataTable": {
            "List": [
                {
                    "ID": "686400cf78badf6500138270",
                    "Name": "Melina Loeck",
                    "PhotoUrl": "https://football-
891b.restdb.io/media/686400ce78badf650013826f",
                    "PhotoID": "686400ce78badf650013826f",
                    "TeamName": "Brighton and Hove Albion",
                    "TeamID": "6863e41f78badf6500137d10",
                    "TeamBadgeUrl": "https://football-
891b.restdb.io/media/6863e41e78badf6500137d0f",
                    "TeamBadgeID": "6863e41e78badf6500137d0f",
                    "Position": "Goalkeeper",
                    "PositionID": "6756e62f050c585400050d8b",
                    "SquadNumber": 28,
                    "NationalityName": "Swedish",
                    "NationalityID": "6863fe6578badf65001381ef",
                    "NationalityFlagUrl": "https://football-
891b.restdb.io/media/6863fe6578badf65001381ee",
                    "NationalityFlagID": "6863fe6578badf65001381ee",
                    "Biography":
"https://www.brightonandhovealbion.com/player-detail-statistics-
goalkeeper/548087"
                }
            ],
            "DynamicColumns": null,
            "TotalRecordCount": 0,
            "TotalPageCount": 0,
            "FirstRowNumber": 0,
            "LastRowNumber": 0,
            "PageNumber": 1,
            "RecordsPerPage": 1,
            "SortColumnName": null,
            "SortAscending": true,
            "AICriteria": null,
            "Success": false,
            "ErrorMessage": null
        },
        "URL": "https://football-891b.restdb.io/views/ReadFootballers?
ID=686400cf78badf6500138270",
        "Verb": "GET",
```


Create a Data Entry Form

Add

Name

Purpose

Type

Icon

```
    "Success": true,
    "ErrorMessage": null
}
```

Caption

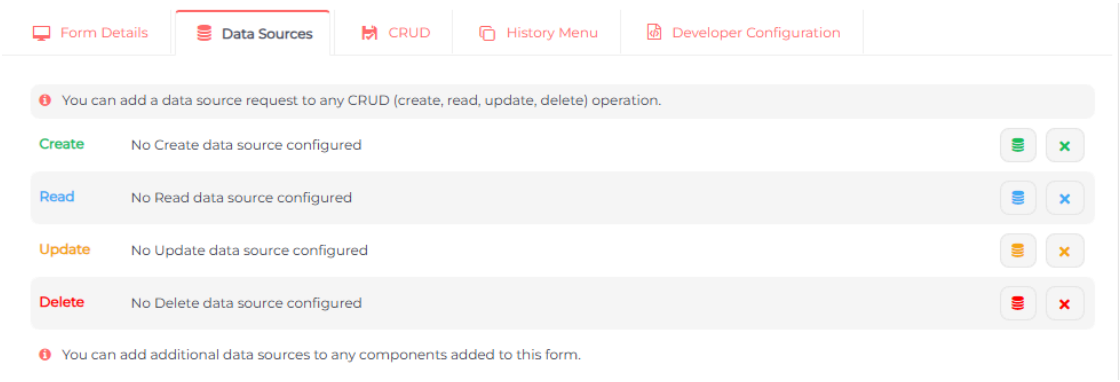
Set the caption to "Footballer". Note that we will configure the history menu to show the footballer name, not this caption.

Description

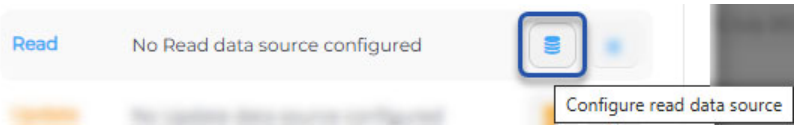
The description will be automatically generated which is fine for now.

Data Sources

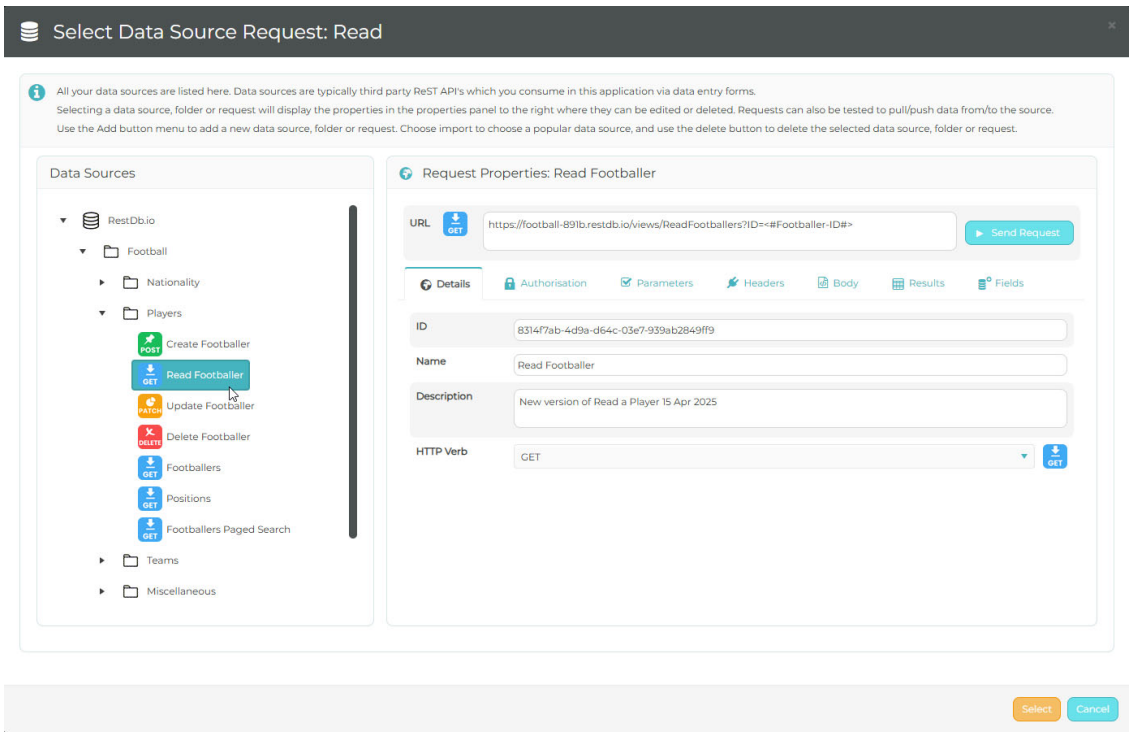
Click the Data Sources tab as this is where we will add the READ data source we created earlier:



There are 4 CRUD data source lines. Click this database icon for the Read data source:

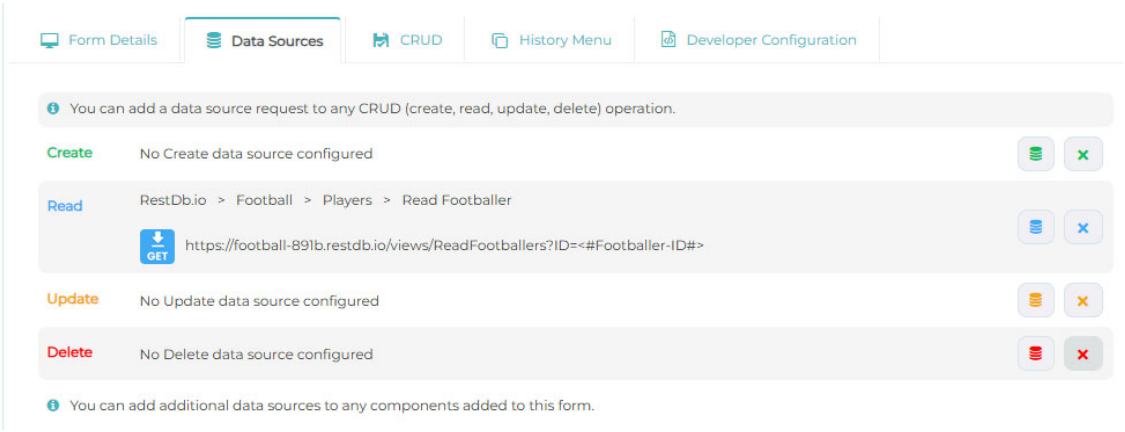


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request:



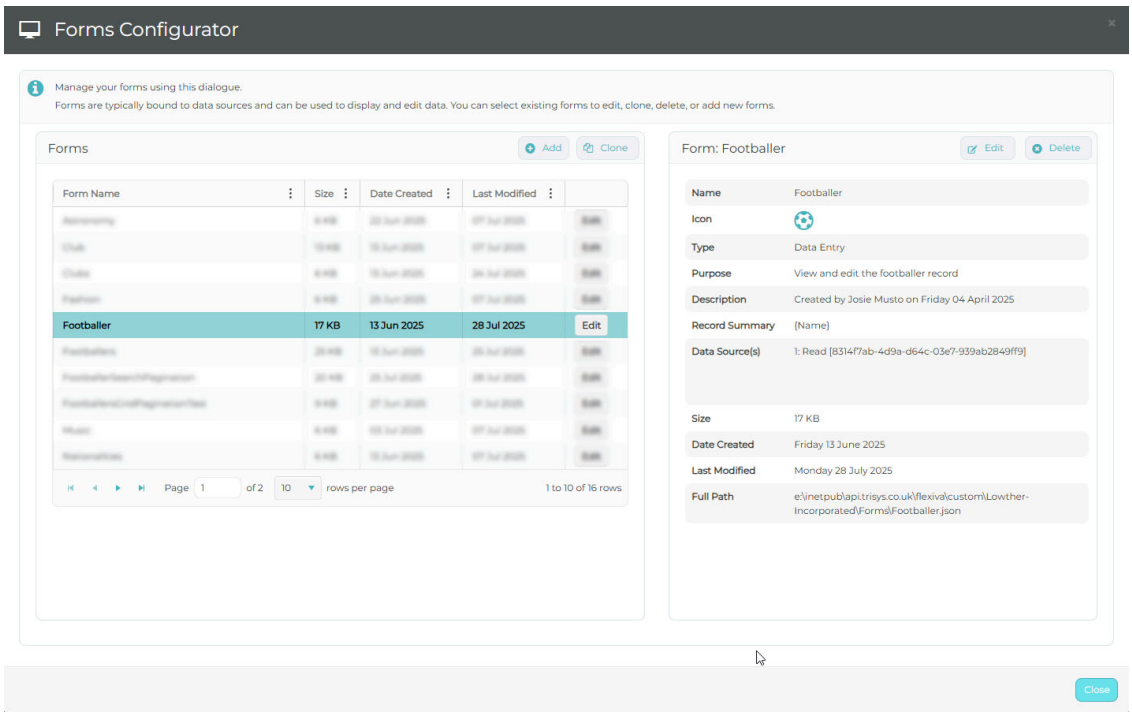
Click the **Select** button to choose this data source.

The popup will close and the selected Read data source request is now added to the list of data sources for this data entry form:



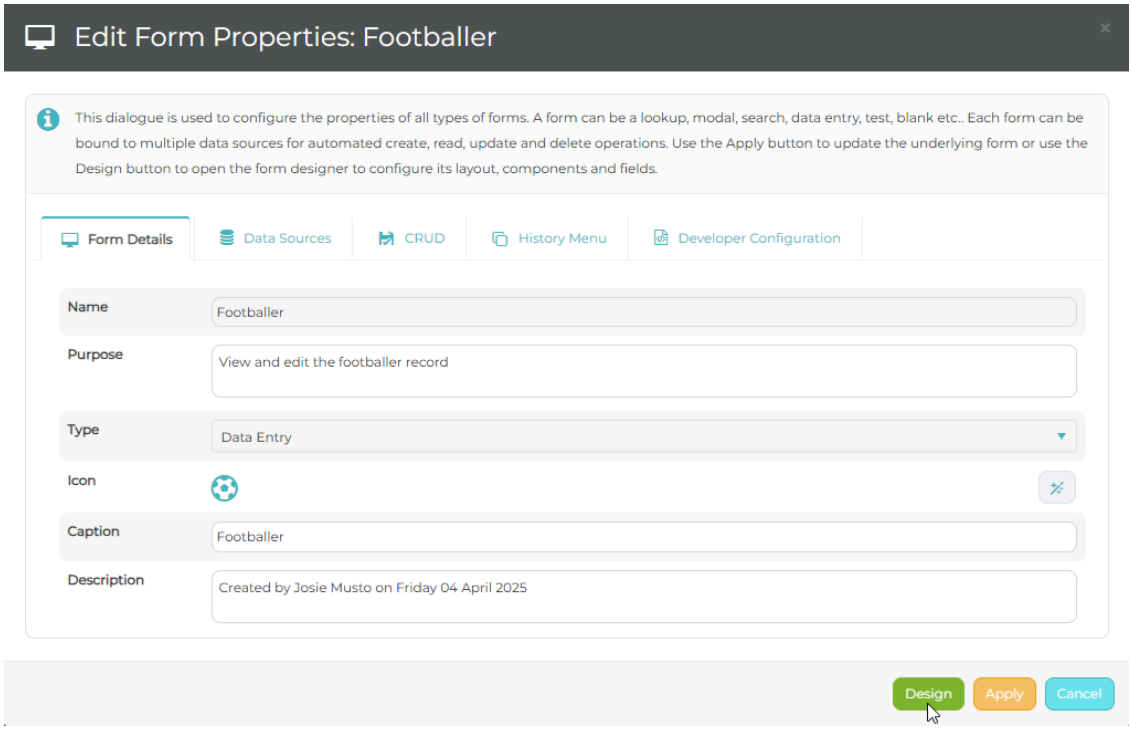
If you ever need to remove a data source request from the list, you can use the X button.

It is recommended that you now click the **Apply** button on this form, in order to persist the form properties before designing your form. Your new form should now appear in the list of forms, and the Read data source request you added should be shown in the properties list:



Form Design

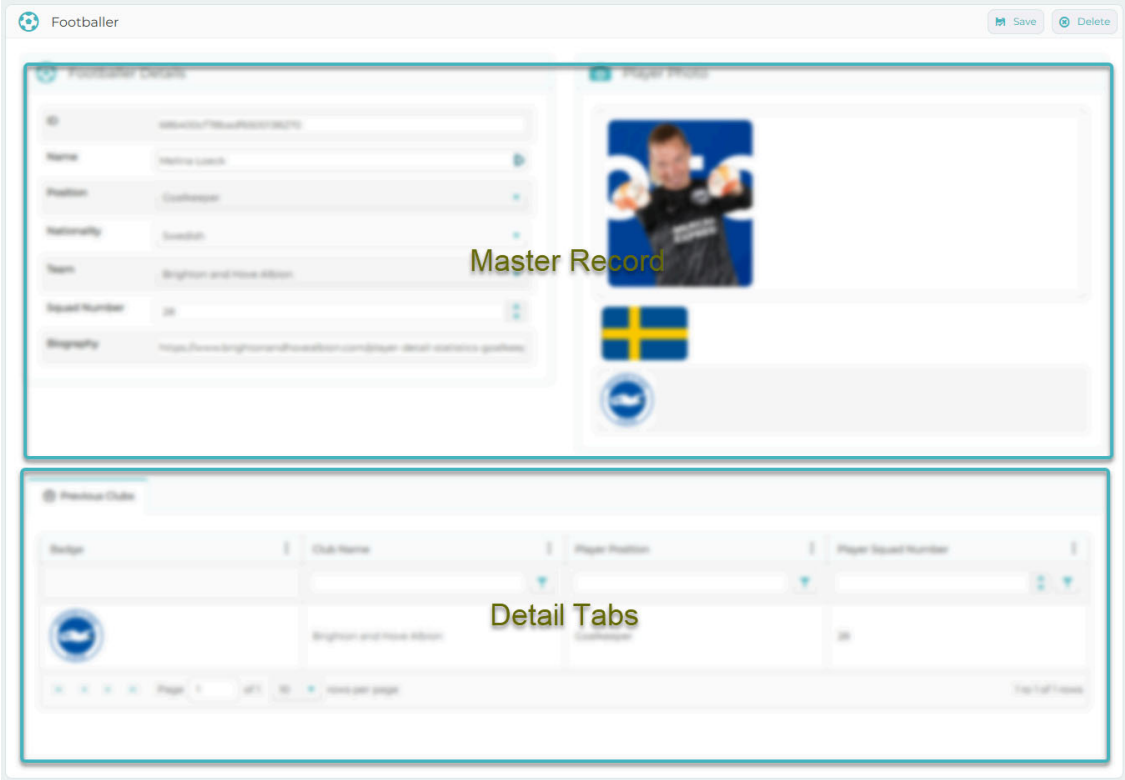
Click one of the **Edit** buttons, either the grid row or properties panel. The form modal popup will display. Click the **Design** button:



The form designer will open.

Tabs

The tabs toolbar is the selected tool by default. Each data entry form is designed as a master/detail meaning that the master record is shown at the top, and any further details about linked entities is shown below in a series of tabs like this:



The Main Top Region refers to the master record on the top of the form.

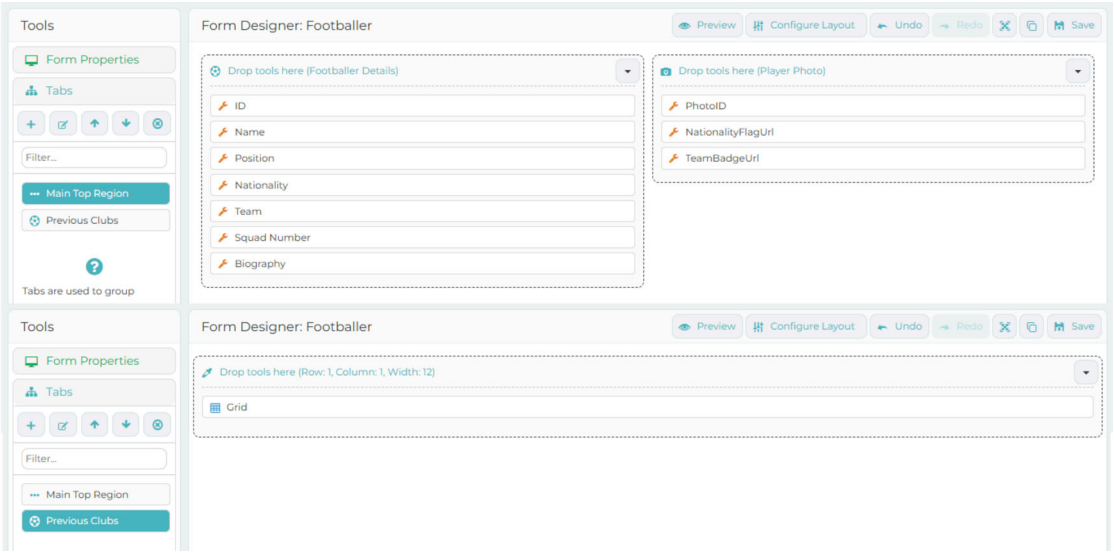
We will design the football fields in this region.

Designer Video

This is the most complex form in the sample application, so this video demonstrates how to design it by creating panels and dragging fields from the toolbox:

We will add another tab to show a grid of previous clubs:

Once both the Main Top Region and the Previous Clubs tabs have been designed, each tab should respectively look like this:



Save Form

Now save the form design using this button:

We have now completed the design of our data entry form to read a record.

We will now add this to the navigation bar.

Navigation Bar

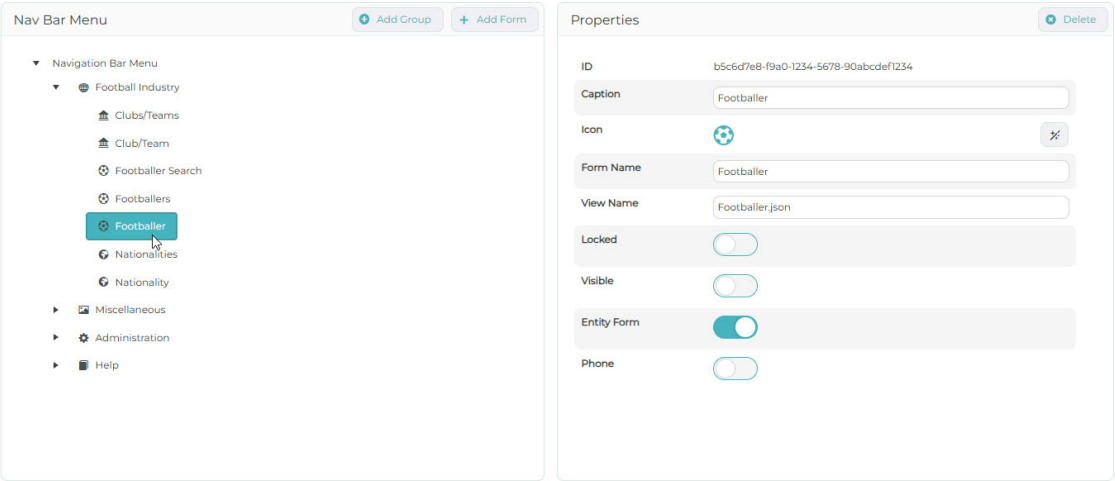
When we add this form to the navigation bar, we will be able to test it.

Open App Studio and select the Navigation Bar configurator and select the Football Industry folder you created previously.

Add Form

Click the Add Form button to open the Add New Form Menu Item modal popup where you should select the Footballer form you created earlier.

Click the Save button which will close the popup and show the new form in the nav bar menu:



Properties

Check or set these properties.

Visible

Because this is a data entry form, we only want it to be visible in the nav bar when the form is open and showing a record, so this should be unchecked.

Entity Form

This is a data entry form which models entities, so this should be checked.

Apply & Close

Click the Apply & Close button to persist the navigation bar and refresh the nav bar.

Configure Drill Down

The navigation bar should not show any change from the last time you saw it because the Footballer form will only appear on it when the form is opened.

In order to test this data entry form, we need to enable drill down from the Footballer Search lookup form we created previously.

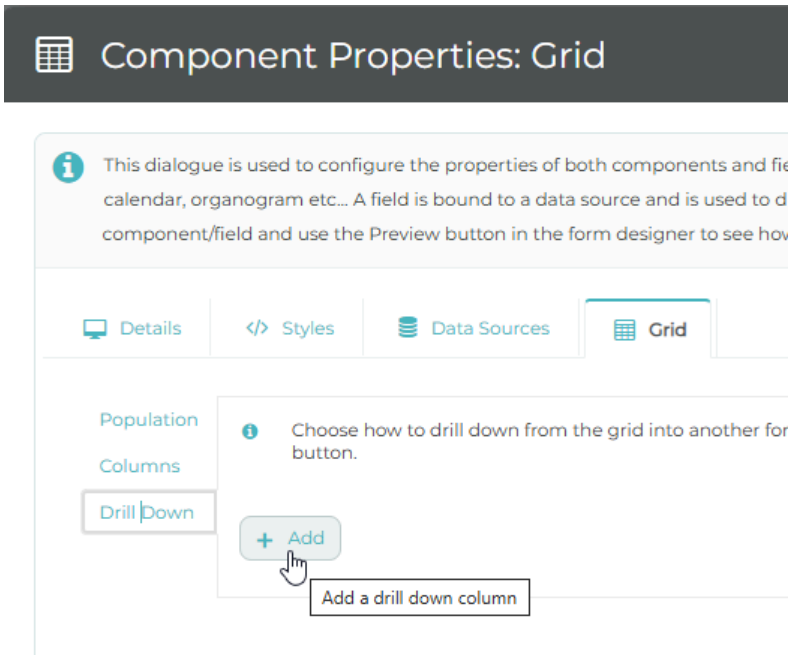
Configure Grid Component

Open App Studio and select the Forms configurator.

Open Footballer Search Form Designer

Open the Footballer Search form in form designer by choosing Edit then Design.

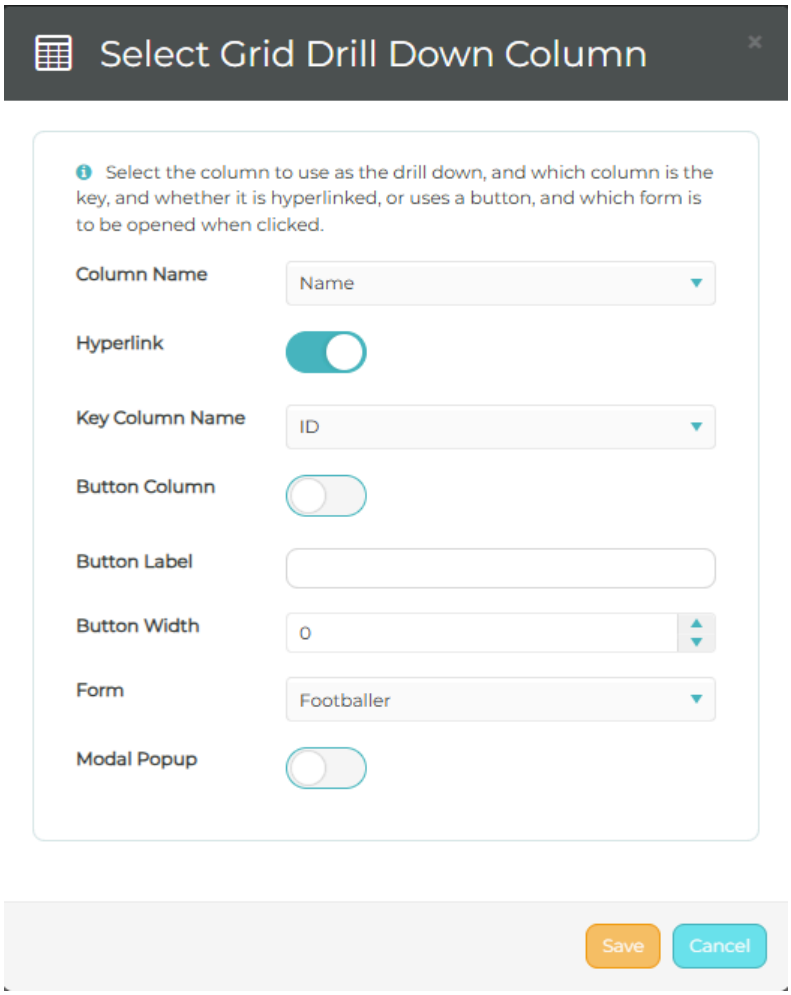
Click on the grid to set the drill down from the footballers grid to the footballer form.



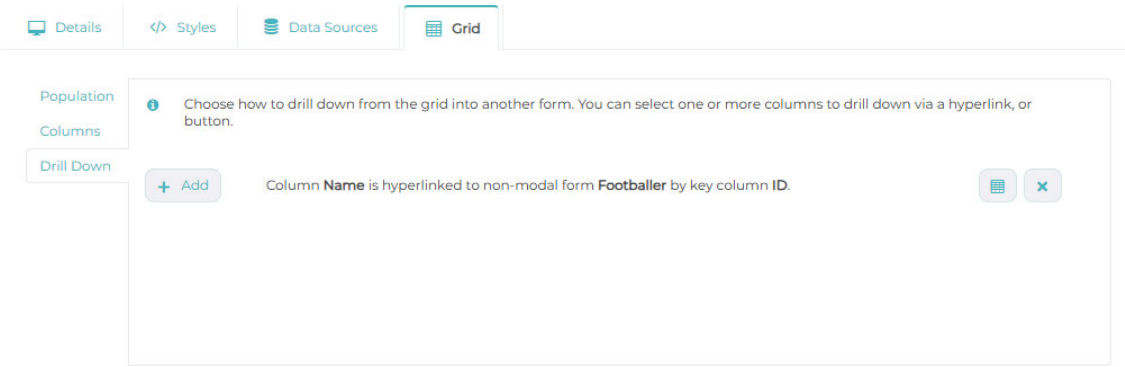
Footballer Name Column

Add a drilldown column.

Choose the `Name` column and hyperlink this via the `ID` key column name to the `Footballer` form:



Click the **Save** button which will close the popup and show the drill down hyperlink details:



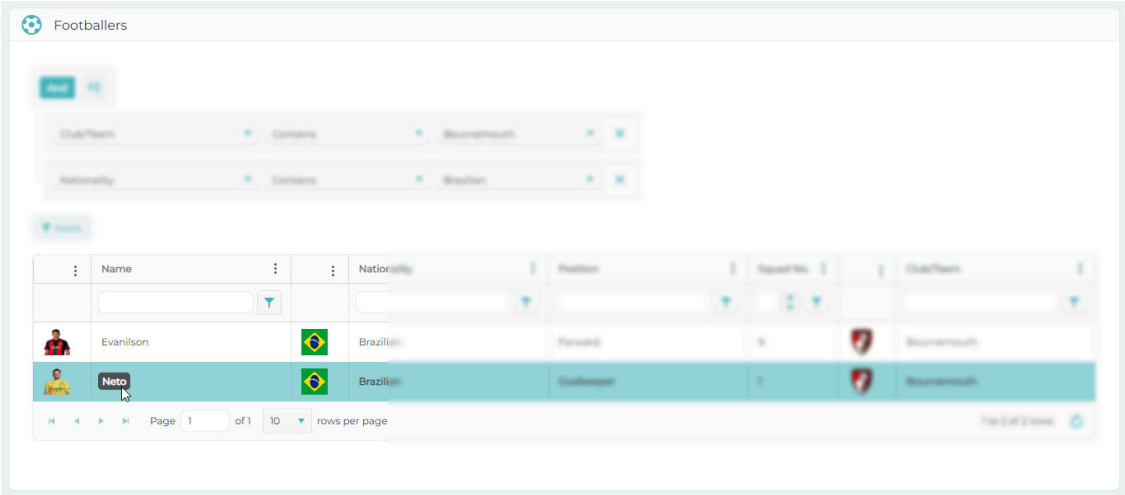
Click the **Apply** button, and then **Save** the form design.

Test Data Entry Form

Open the Football Industry group on the navigation bar.

Footballer Search

Click the Footballer Search nav bar menu to open the lookup/search form:

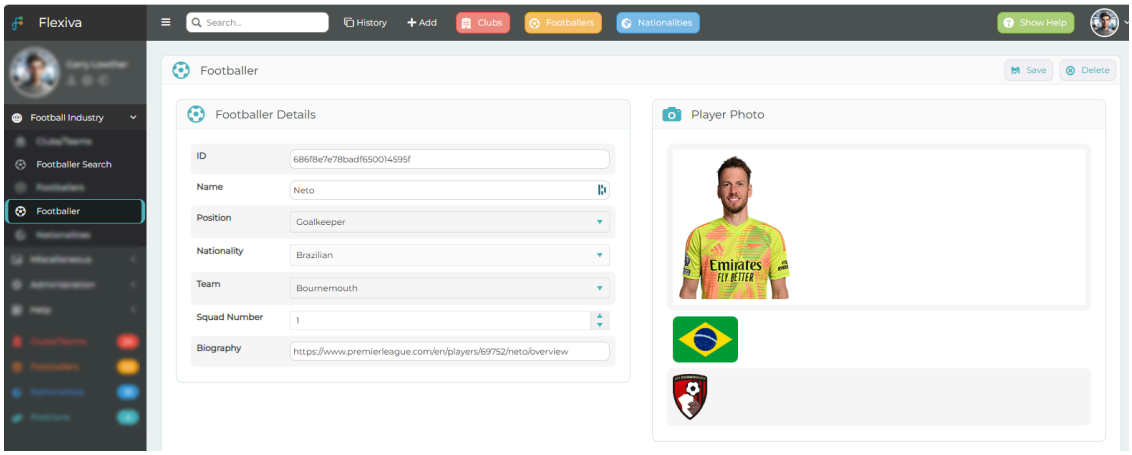


If you hover your mouse over any of the footballers names, you should see that it becomes highlighted. This proves that the drill down configuration has been applied.

Click any footballer.

Footballer Form

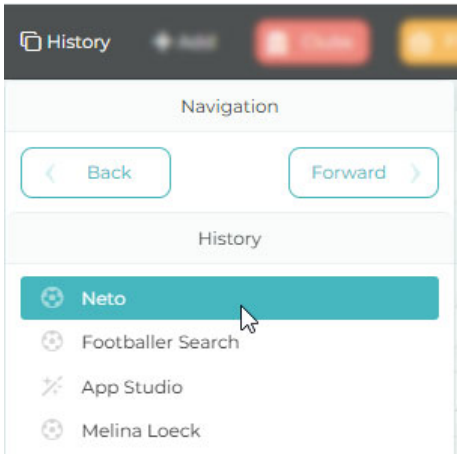
The footballer form should open and show the selected footballer details including their photo, national flag and club badge:



Notice also how the navigation bar now shows the Footballer form as being open?

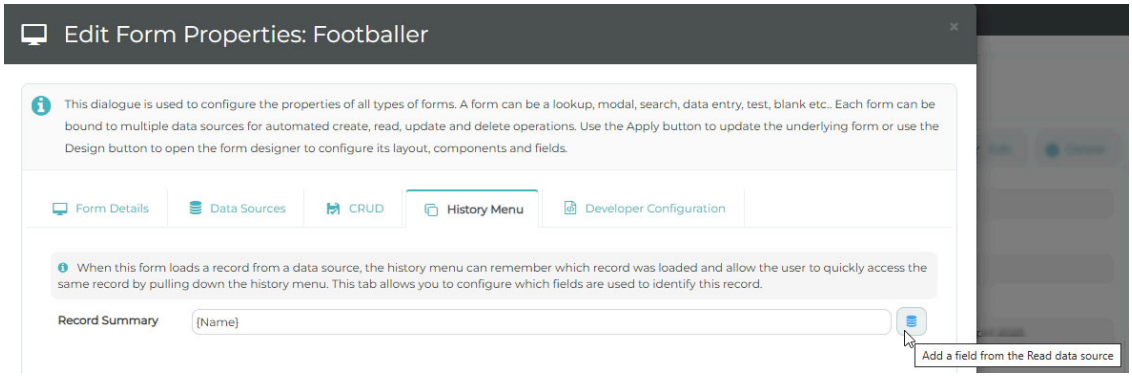
History Menu

When the history menu appears, we want the name of the footballer to appear, not the name of the form for example:



Configuring Record Summary

Open App Studio and select the Forms configurator. Edit the properties of the Footballer form and click the History Menu tab:



Click this button to open the modal popup. Choose the `Name` field and click the **Save** button to persist this setting and close the popup.

Record Summary

The record summary should now show `{Name}` indicating that the `Name` field will be displayed in the History Menu.

Apply

Click the **Apply** button to persist this.

Test Record Summary

Open another footballer from the footballer search lookup form, and then click on the History drop down menu. You should see the last footballer you opened at the top of the list?

Previous Clubs

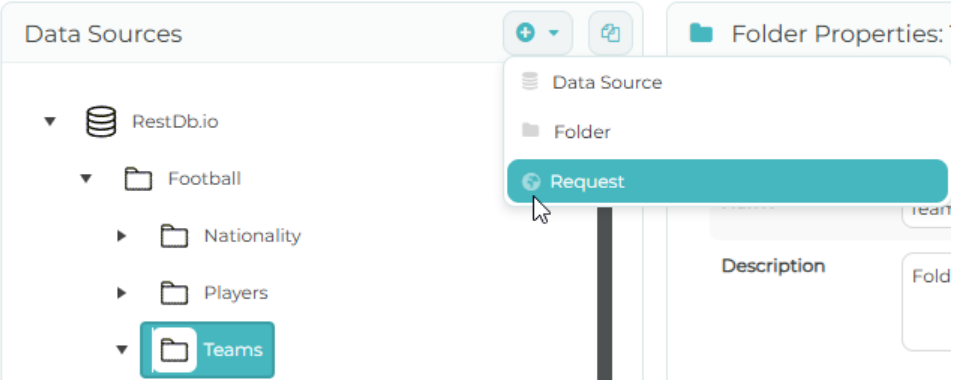
In order to show the previous clubs for the footballer, we will need to create a new data source request, then connect this to our form grid.

Configure the Footballer Club History Data Source Request

In order to display a list of previous clubs on our footballer form, we need to configure the data source request connected to the appropriate ReST API.

Create Footballer Club History

Select the `RestDb.io/Footballer/Teams` folder and add a new data source request:



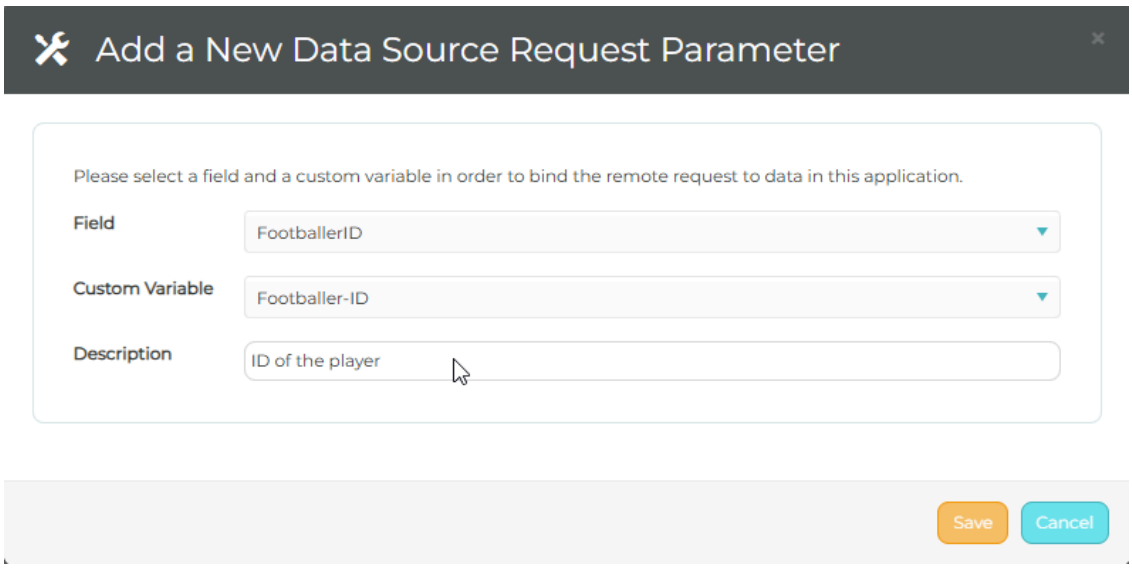
Name it "Footballer Club History" and add this URL:

`https://football-891b.restdb.io/views/FootballerClubs` ↗

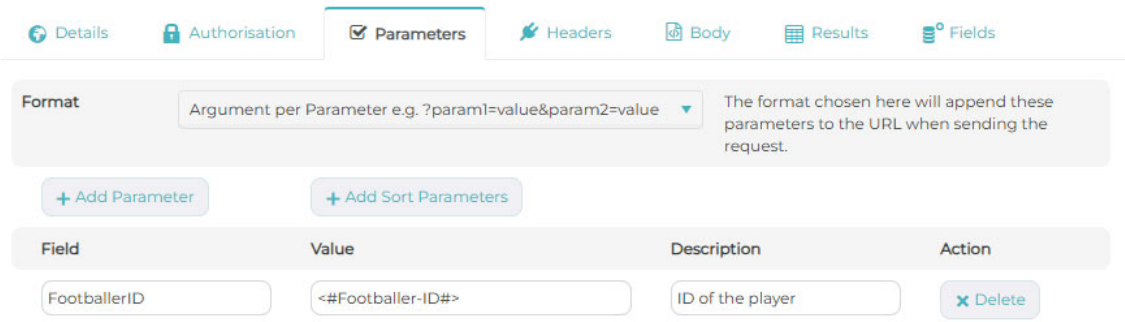
Parameters

We want only the clubs linked to a footballer, so we need to create a parameter which will be appended to the URL at run-time.

Click the Add Parameter button to add this parameter:



The parameter is added to the list:



Send Request

Test the request. Use `674cb1cb050c58540003e206` as the Footballer-ID field when testing as this is the ID of a footballer who was transferred from one club to another:

Request Properties: Footballer Club History

Insert Variable | Delete


```
    "DataTable": {
        "List": [
            {
                "ID": "681b3c7878badf65000b29c5",
                "Name": "Bayern Munich",
                "Description": "German",
                "BadgeUrl": "https://football-891b.restdb.io/media/681b3c7878badf65000b29c4",
                "FootballerID": "674cb1cb050c58540003e206",
                "PlayerPhotoURL": "https://football-891b.restdb.io/media/67fe116e78badf650005d077",
                "PlayerNationality": "English",
                "PlayerNationalityFlagURL": "https://football-891b.restdb.io/media/67fa336b78badf65000
54bea",
                "PlayerName": "Harry Kane",
                "PlayerPosition": "Forward",
                "PlayerSquadNumber": 9
            },
            {
                "ID": "67fe142c78badf650005d0e3",
                "Name": "Tottenham Hotspur",
                "Description": "North London Premier League Football Club. Spurs.",
                "BadgeUrl": "https://football-891b.restdb.io/media/67fe198978badf650005d1cb",
                "FootballerID": "674cb1cb050c58540003e206",
                "PlayerPhotoURL": "https://football-891b.restdb.io/media/67fe116e78badf650005d077",
                "PlayerNationality": "English",
```


The JSON returns 2 footballer clubs.

Wire up the Previous Clubs Grid

We can now point the grid on the Footballer form at this footballer club history data source request.

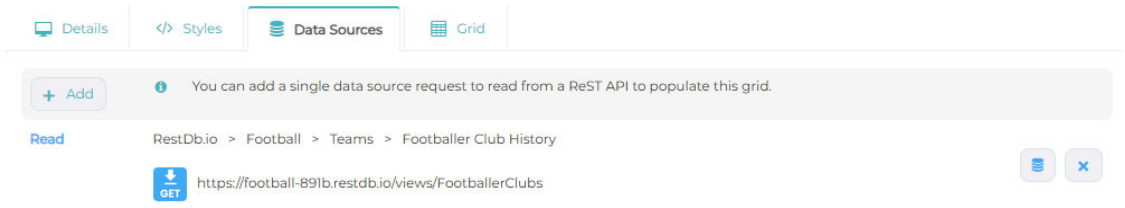
Open App Studio and select the Forms configurator. Open the Footballer form, edit it, and Design it to open the forms designer.

Add Read Data Source

Click on the grid on the Previous Clubs tab and open the Component Properties: Grid modal popup and click on the Data Sources tab.

Use the Add button to choose this data source request.

The popup form should now show that a Read data source request is associated with the grid:

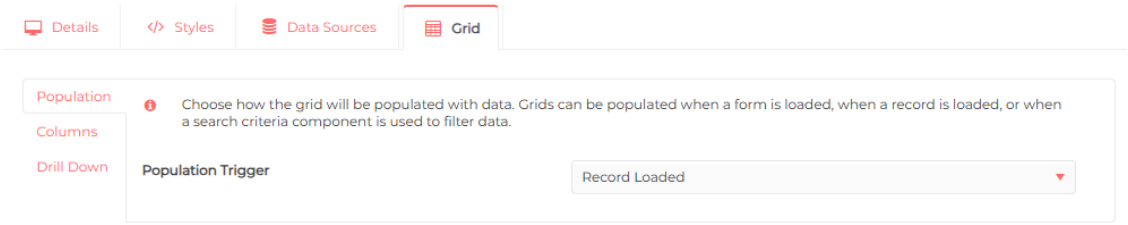


Grid Columns

Click on the Grid tab.

Population Trigger

Select "Record Loaded" in the drop down combo:



Columns

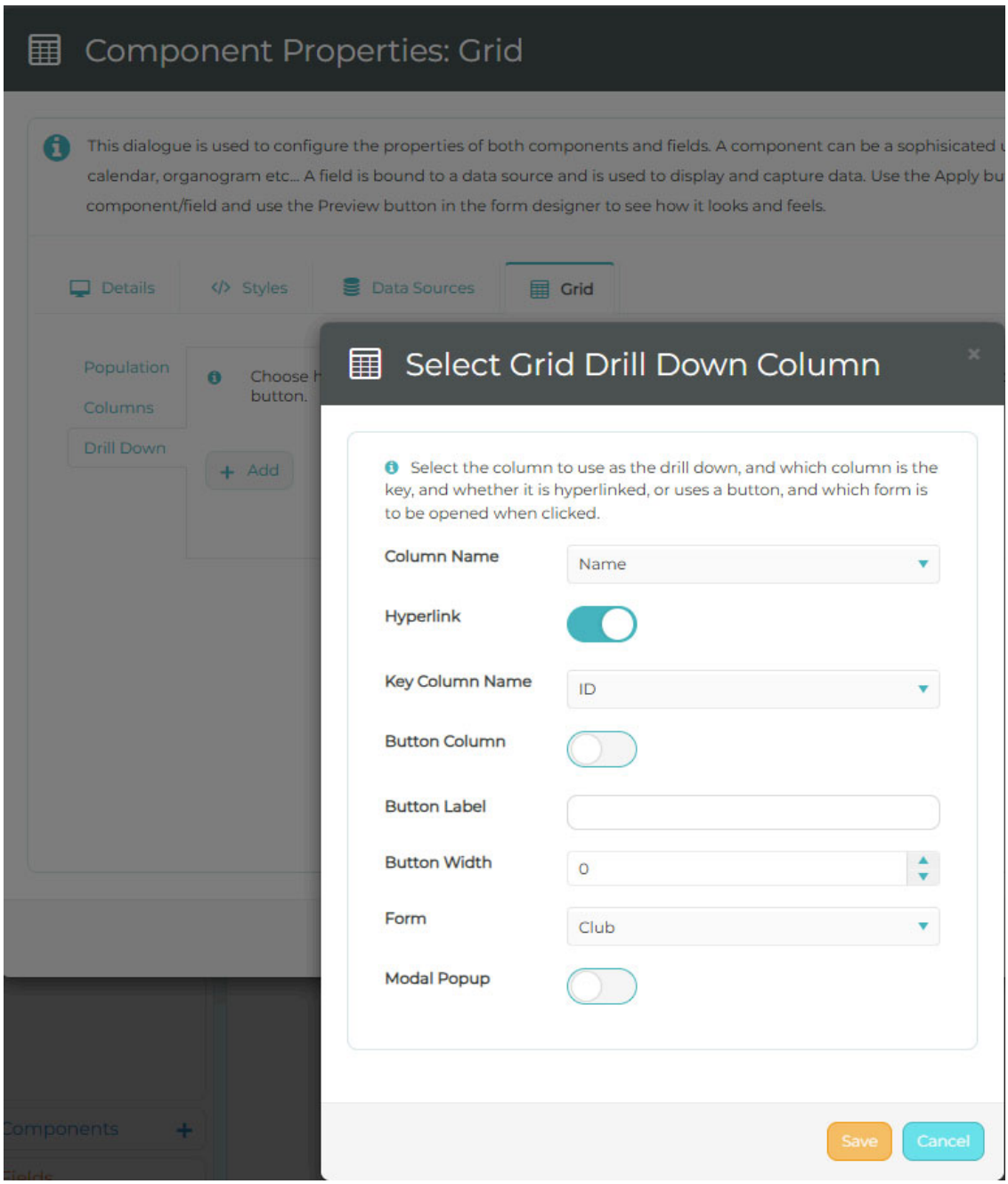
Click on the left tab Columns, and order the columns, hiding some, as you would like to see them.

Because this grid is on a Footballer form, we do not need to see any of the footballer columns in this grid. Here are the properties for each column you should set:

Column	Properties
`ID`	Invisible
`BadgeUrl`	Type: Image URL, Image Size: 32 × 32 Visible, Caption: Club Badge, Width: 120
`Name`	Visible, Caption: Club Name
`Description`	Visible
`FootballerID`	Invisible
`PlayerPhotoURL`	Invisible
`PlayerName`	Invisible
`PlayerNationalityFlagURL`	Invisible, Type: Image URL, Image Size: 32 x 32, Caption: Nation Flag, Width: 80
`PlayerNationality`	Invisible
`NationalityName`	Invisible, Caption: Nationality
`PlayerPosition`	Invisible, Caption: Position, Filterable
`squadNumber`	Invisible, Caption: Squad No., Width: 110, Filterable

Drill Down

We previously created the Club form so we can drill down into it from this grid. Use the **Add** button on the Grid tab, Drill Down left tab and set the column name, key column name and form as follows:



Use the **Save** button to save the drill down.

Apply

Apply the changes to these properties.

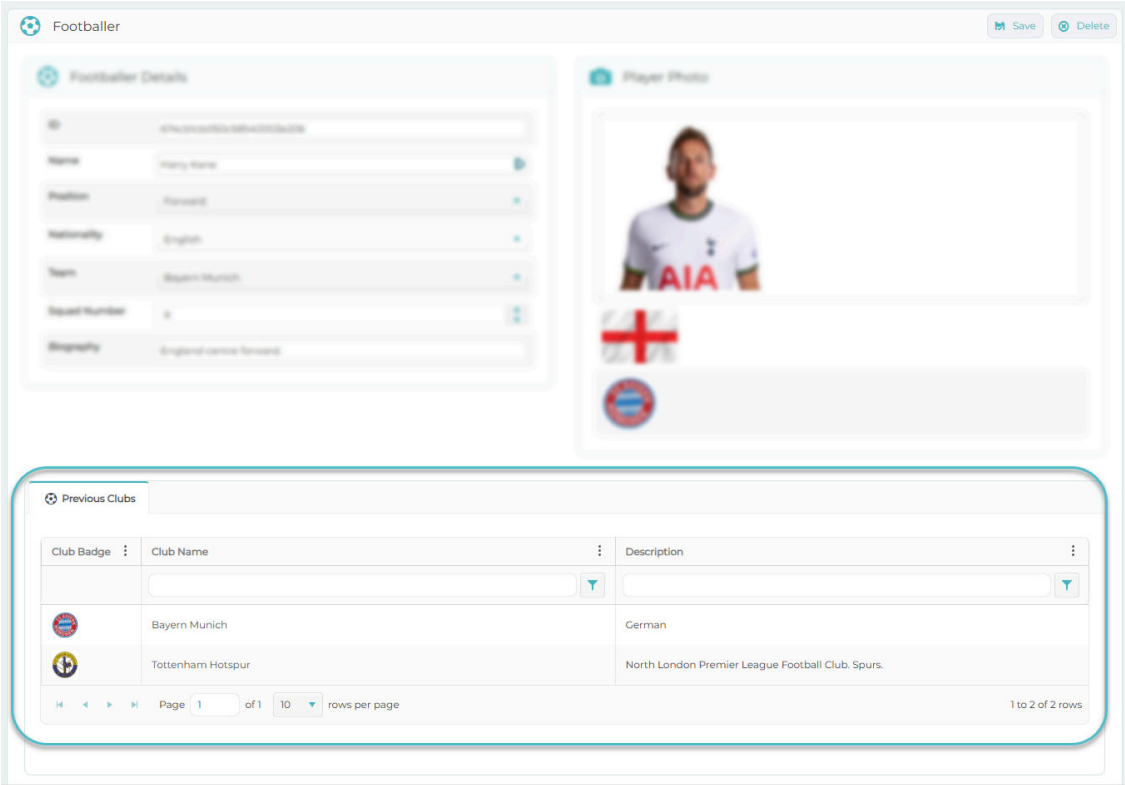
Save

Save the form design to persist the configuration.

Test

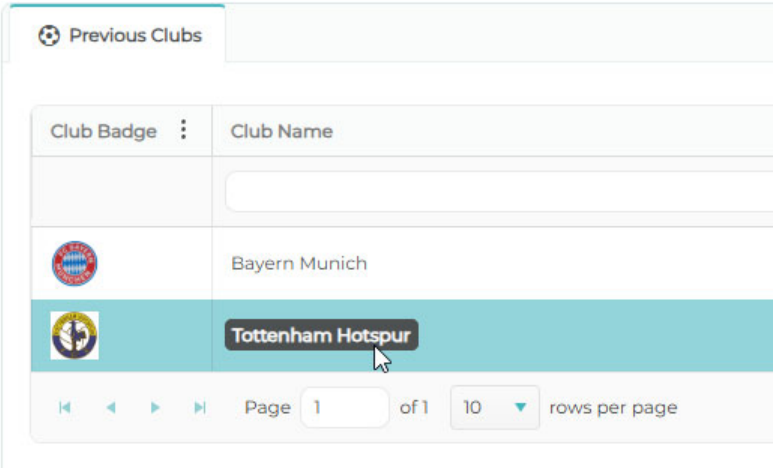
You can now test your configuration by opening the Football Industry group in the navigation bar and selecting the Footballer Search item. Search for Harry Kane, and drill down into his footballer record to open the form.

The Footballer form now shows the previous clubs in the grid:



Drill Down

Test the drill down by hovering over the Club Name:



Click this hyperlink to open the Club form.

Footballer Form: Update

Configure the footballer form to update a record.

This is the process we will follow.

UPDATE

- Create custom variables for each field
- Add an 'update' data source for updating a single footballer record
- Link the key footballer record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'update' data source to the data entry form
- Configure the update button
- Test that we can update a footballer using the Update button

Create Custom Variables

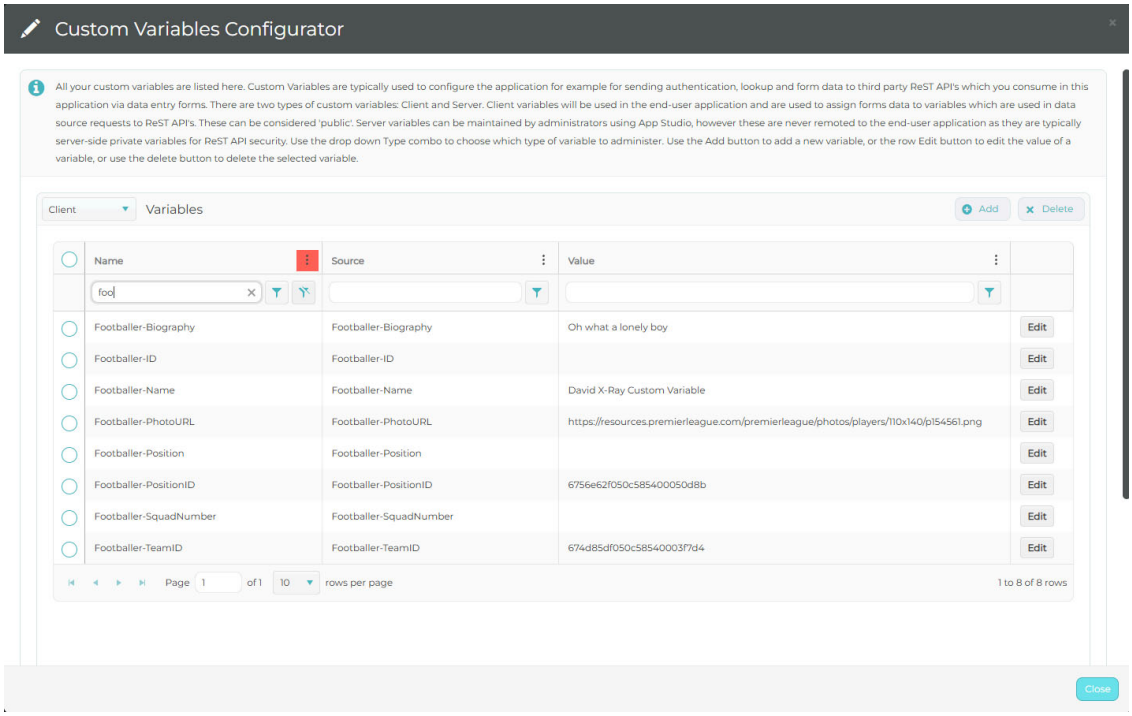
In order to update a footballer record, we need to create custom variables for each form field so that we can send these to the ReST API.

Open the App Studio, then select the Custom Variables configurator.

Add the following client-side custom variables:

- `Footballer-Name`
- `Footballer-PositionID`
- `Footballer-TeamID`
- `Footballer-SquadNumber`
- `Footballer-Biography`
- `Footballer-PhotoURL`

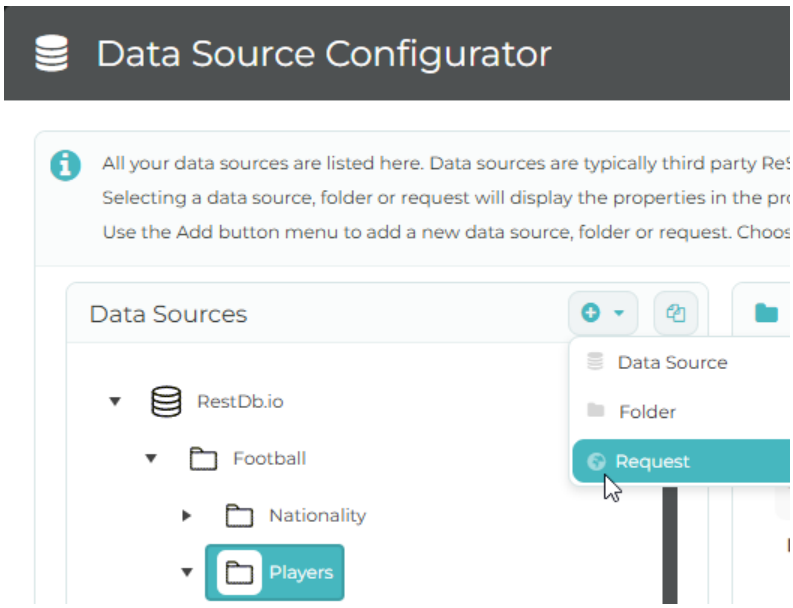
The new custom variables should now be displayed:



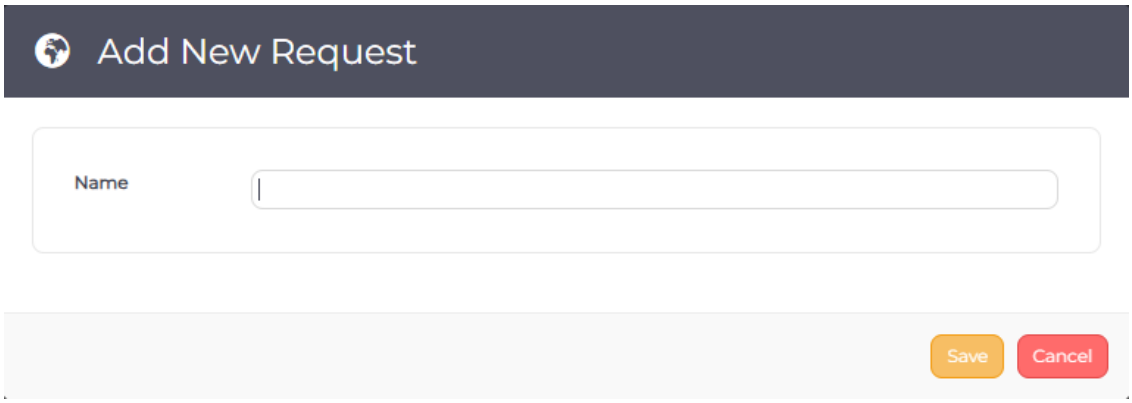
We will need to link these new custom variables to each form field, but first we will use them in a new data source to update the record.

Add an UPDATE Data Source

Open App Studio, open the Data Sources configurator, select the Players folder you created previously, then use the add button menu to create a new request:

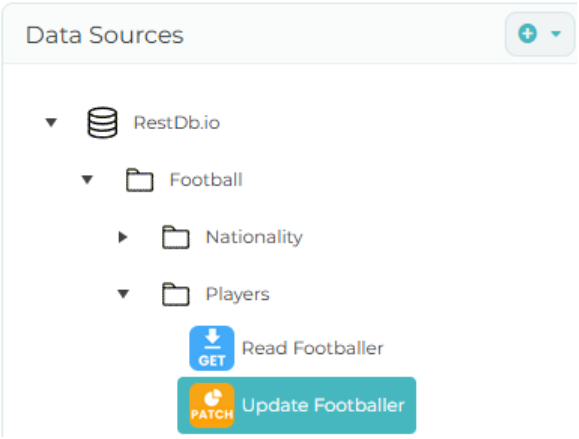


The Add New Request modal popup form shows asking you to name the request:



Type a meaningful name such as "Update Footballer" and click Save.

The request will be added to your tree view beneath the Players folder:



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

URL

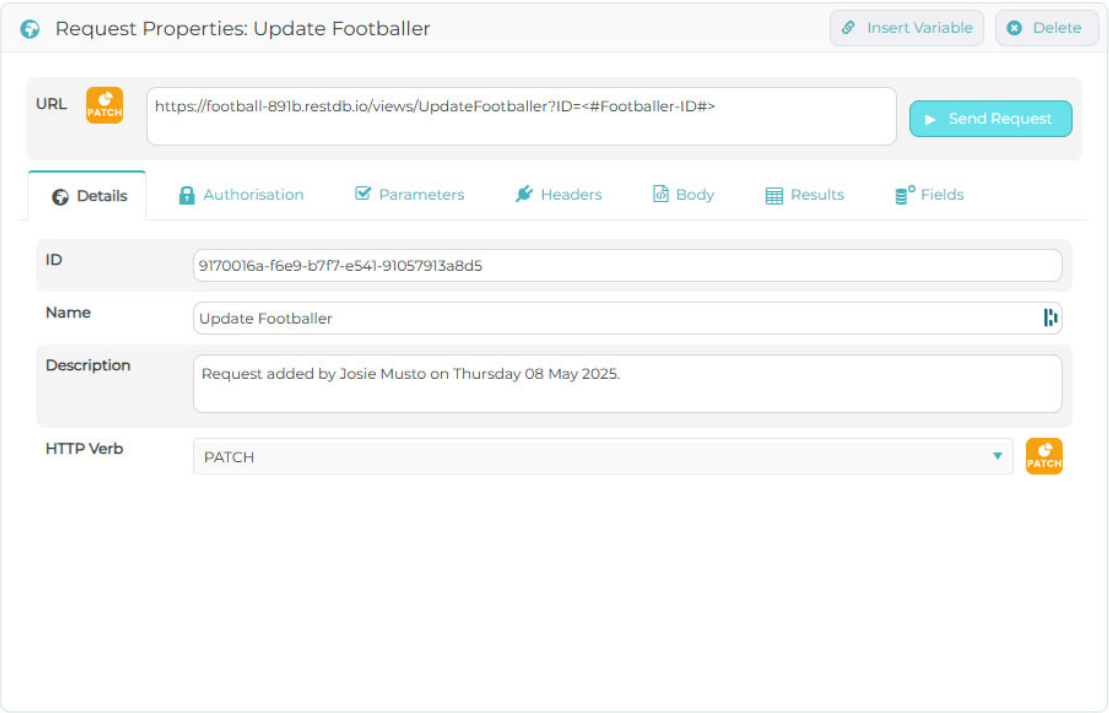
It is known from the documentation that this ReST API end-point has a footballer identifier property ?ID=.

We therefore type this parameter list into the URL so that it reads:

`https://football-891b.restdb.io/views/UpdateFootballer?ID=` ↗

Whilst the caret is still blinking after the last character typed, click on the Insert Variable button and select the Footballer-ID variable.

Then change the HTTP Verb to PATCH. The configurator should now look like this:



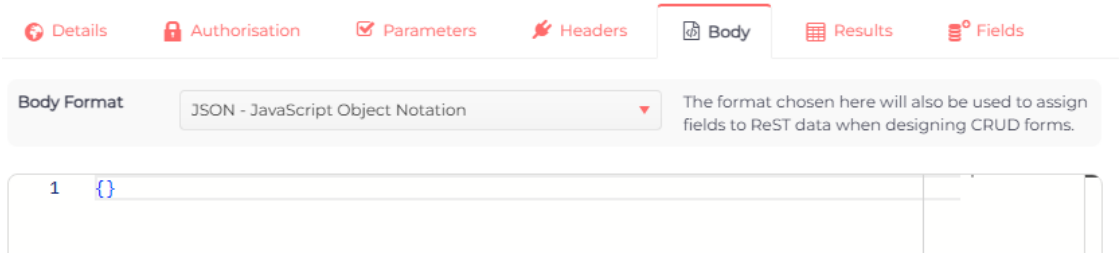
Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Body

When we are updating or creating data, we will be sending the data in the body of the request, so we configure this before sending the request to test it, using the new custom variables. Here is the empty Body tab:



We will always use the JSON body format as this gives us maximum control.

From the ReST API specification, we know that this endpoint expects data in this format:

```
{
    "name": "",
    "squadNumber": 0,
    "biography": "",
    "PhotoURL": "",
    "PositionID": "",
    "NationalityID": "",
    "TeamID": ""
}
```

Our job is to now associate each field with the appropriate custom variable.

Put the carat inside each double quotation and use the Insert Variable button to select the respective custom variables setup above.

After all have been inserted, the body should now look like this:

```
{
    "name": "<#Footballer-Name#>",
    "squadNumber": <#Footballer-SquadNumber#>,
    "biography": "<#Footballer-Biography#>",
    "PhotoURL": "<#Footballer-PhotoURL#>",
    "PositionID": "<#Footballer-PositionID#>",
    "NationalityID": "<#Nationality-ID#>",
    "TeamID": "<#Footballer-TeamID#>"
}
```

Send Request

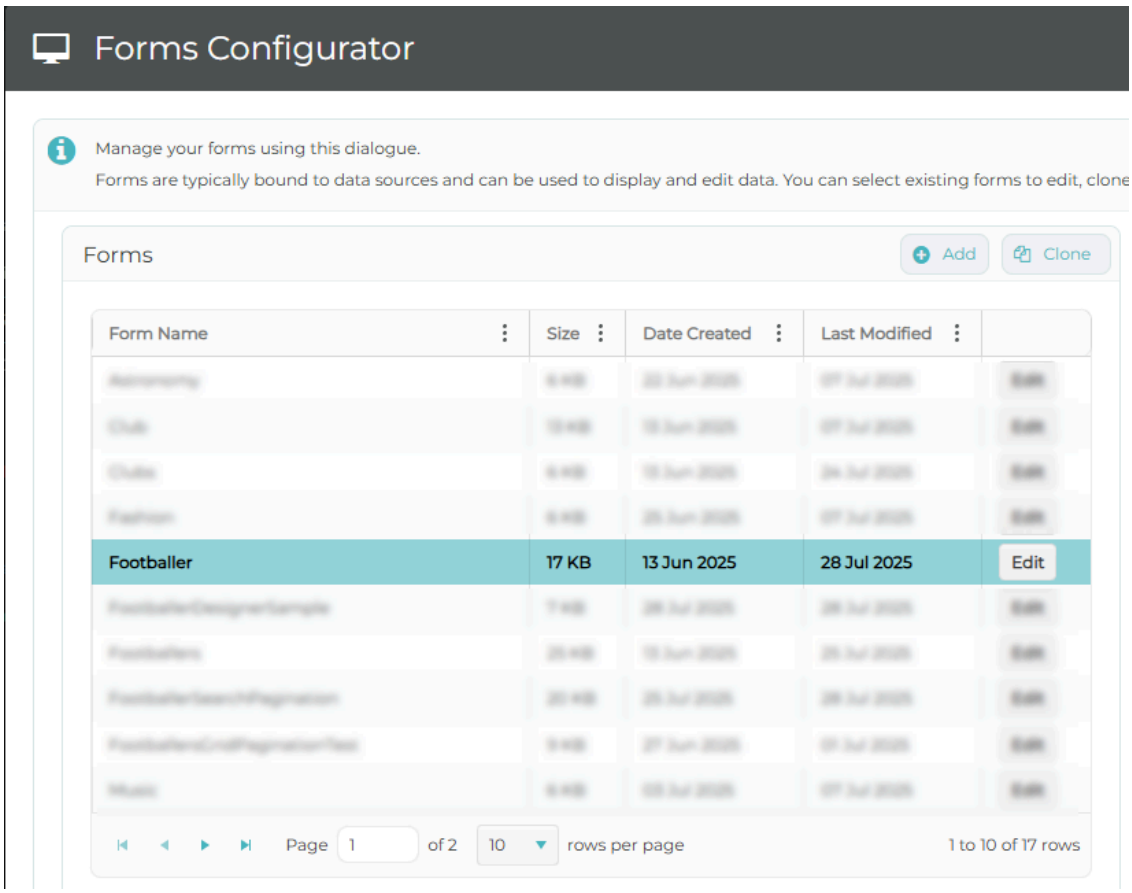
It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to update an existing record in order to test this.

The best way of course to test this is to use the actual footballer form we created here, and link that form to this data source request, then we can test it.

Edit Form

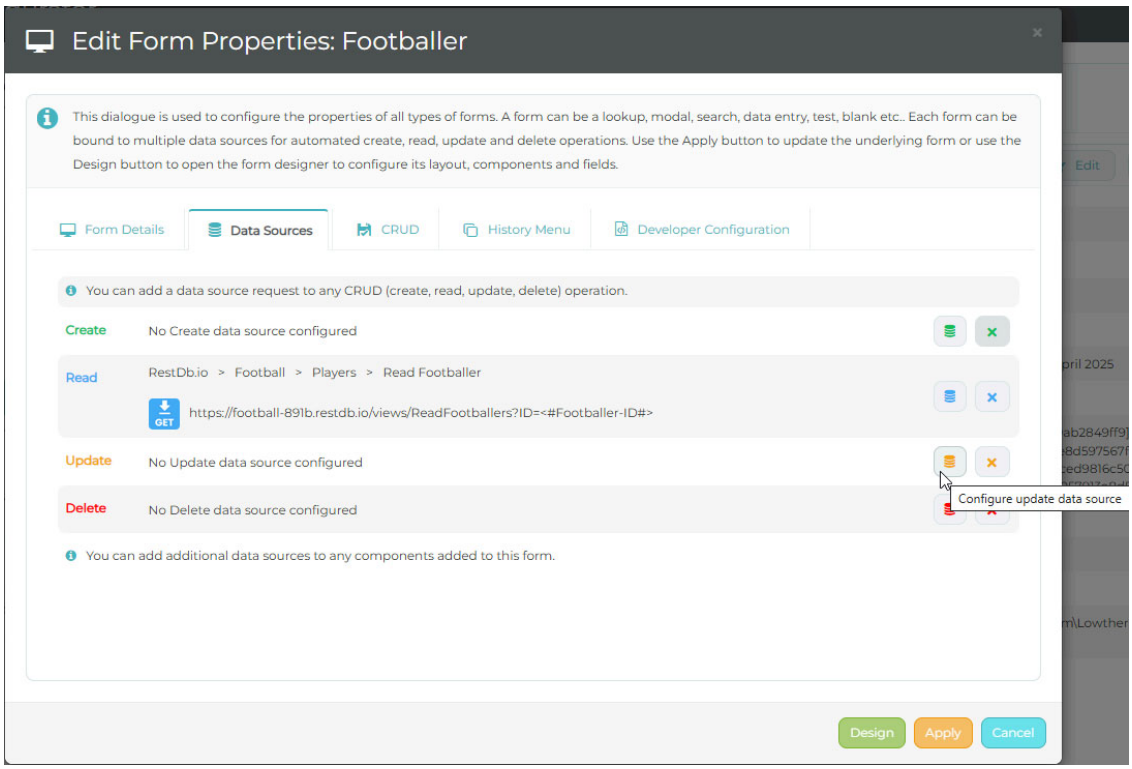
Open App Studio, open the Forms configurator, then select the Footballer form:



Click the Edit button to open the form properties modal popup.

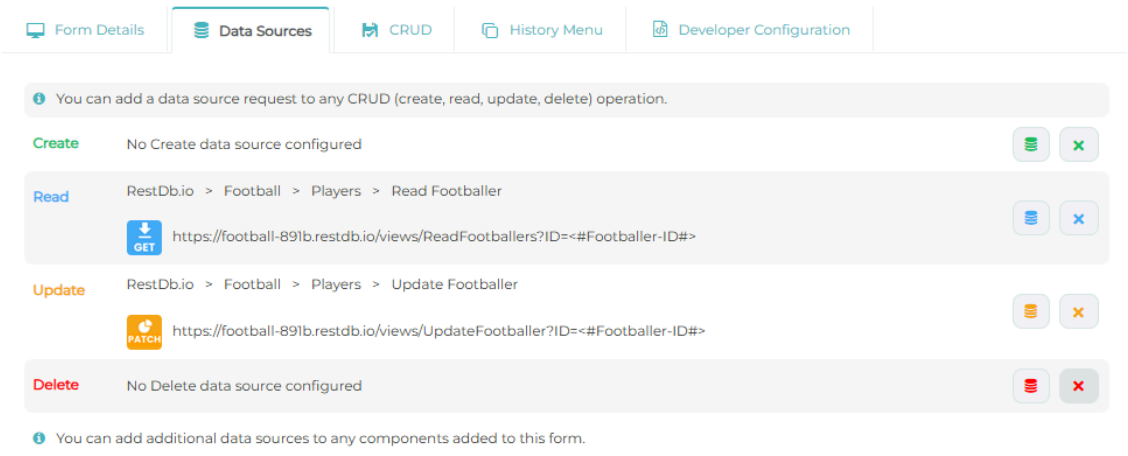
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Update data source:

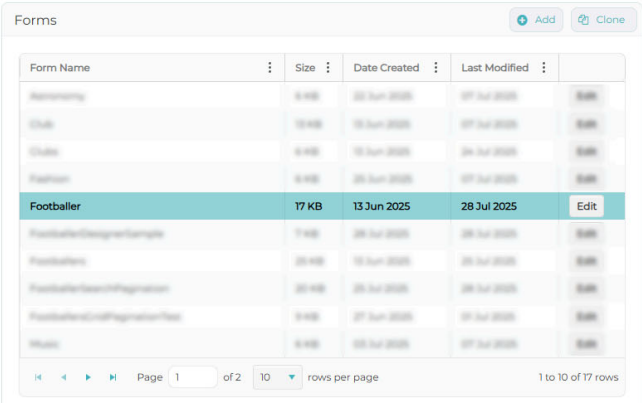


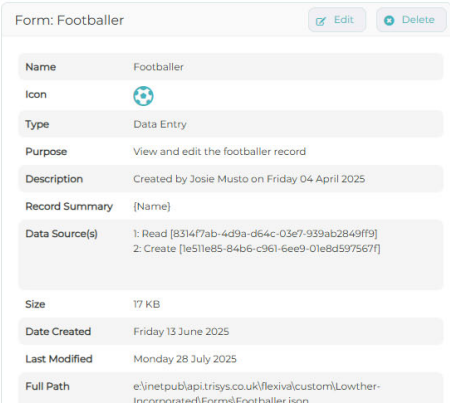
This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the **Select** button.

This Update data source request should now appear in the list of assigned data source requests:



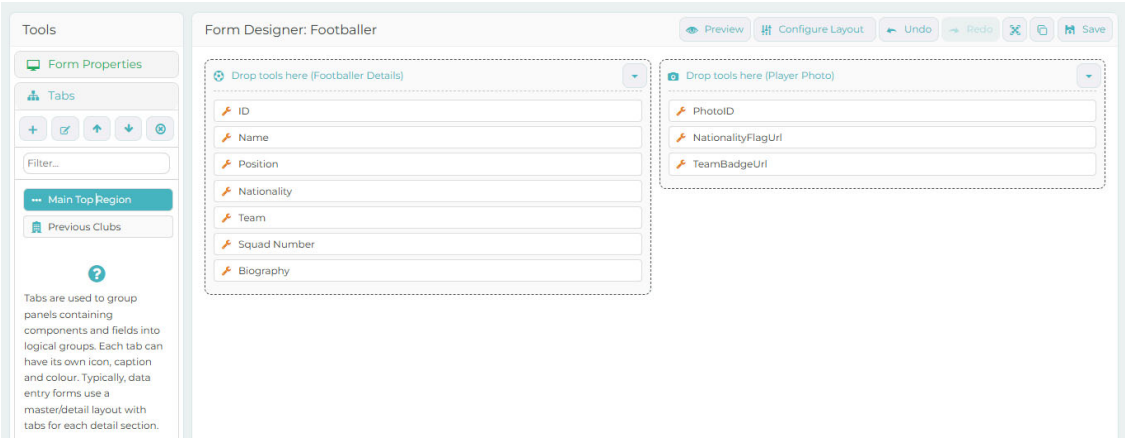
Now click the **Apply** button on this form, in order to persist the form properties before designing your form. The Update data source request you added should be shown in the properties list:





Form Design

Click one of the Edit buttons, either the grid row or properties panel. The form modal popup will display. Click the Design button, and select the Main Top Region tab:



Fields

Click on each of these fields in turn and assign the appropriate custom variable you added previously:

Form Field	Custom Variable
ID	`Footballer-ID`
Name	`Footballer-Name`
Position	`Footballer-PositionID`
Nationality	`Nationality-ID`
Team	`Footballer-TeamID`
Squad Number	`Footballer-SquadNumber`
Biography	`Footballer-Biography`
PhotoID	`Footballer-PhotoURL`

Save Form

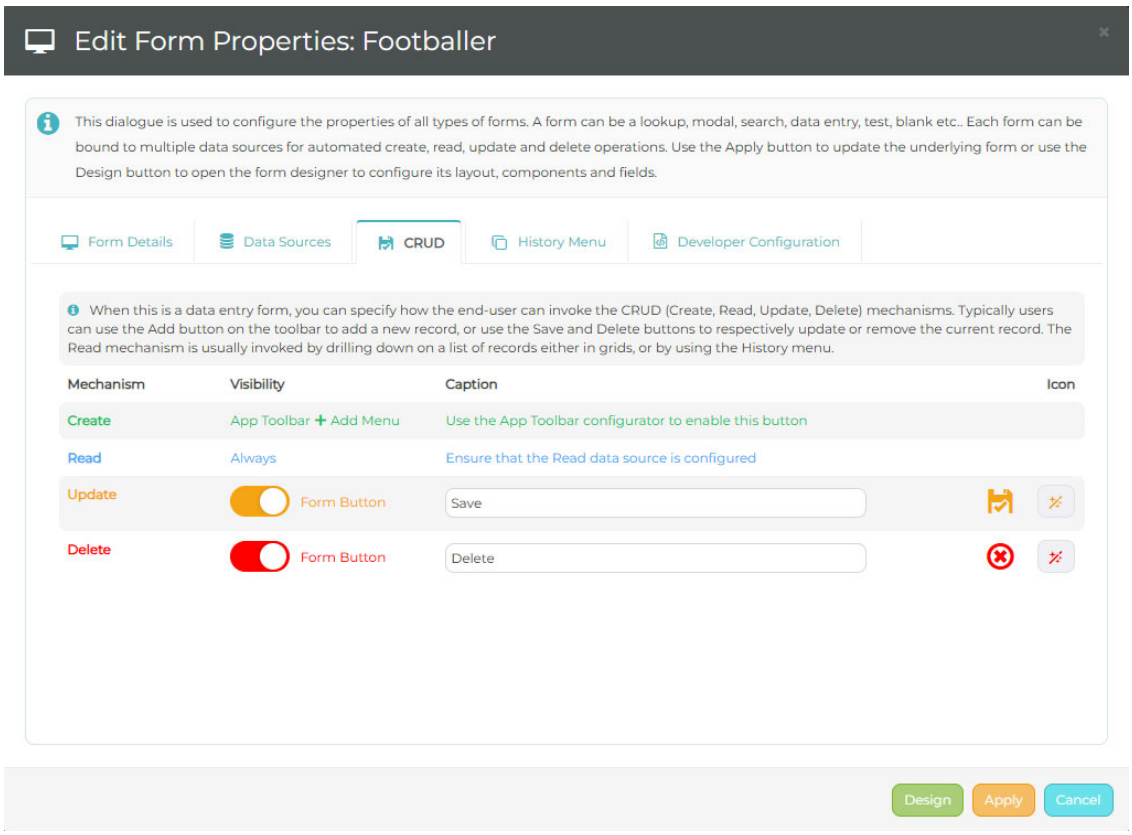
Now save the form design using this button:

The form fields are now assigned to the new custom variables used in the body of the new data source request to update the record.

We will now configure the update button on the form.

Configure Update Button

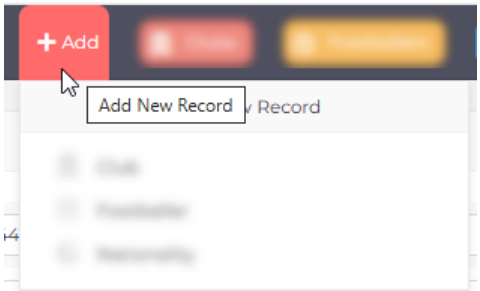
Open App Studio, open the Forms configurator, then select the Footballer form and click the edit button. Then click the CRUD tab:



There are four mechanisms to control: Create, Read, Update and Delete.

Create

A new record can only be created from the app toolbar using the Add drop down menu:



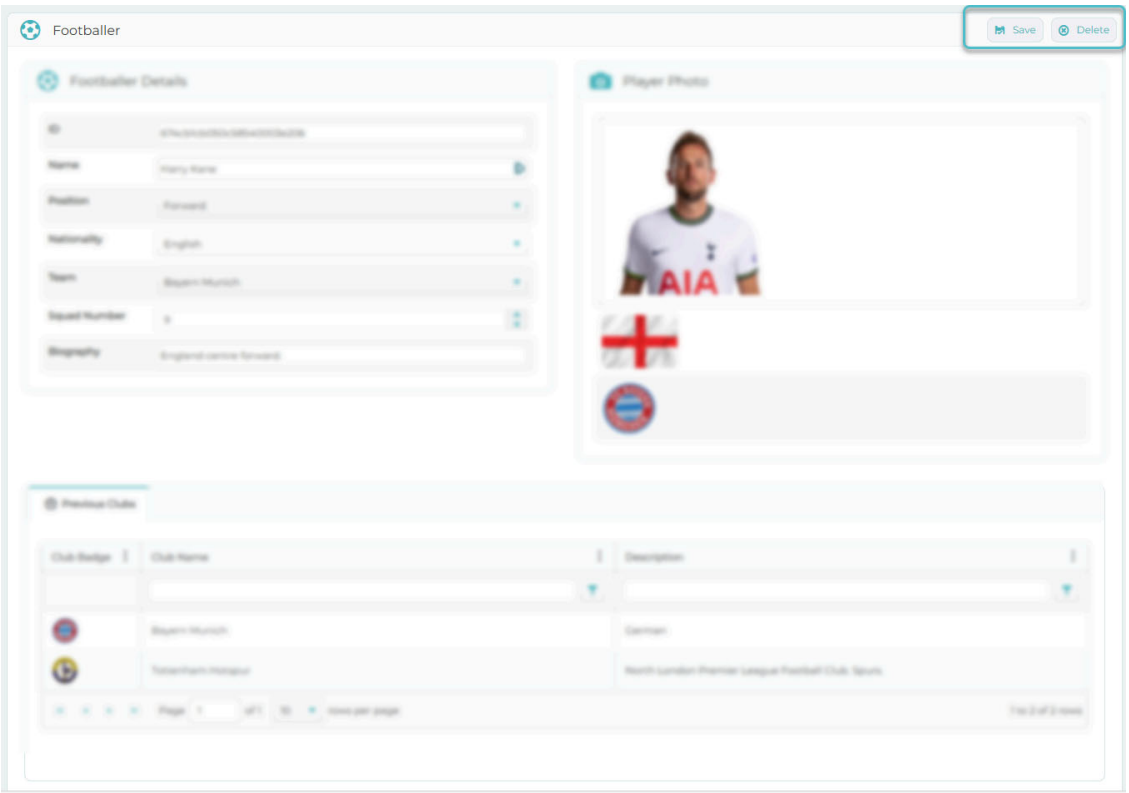
Use [this configurator]() to add any form to this list.

Read

Reading form data is always visible when a form is opened and a record is loaded.

Update

The update button lives together with the delete button top right on the form:



It can be hidden, its caption set and its icon set using these controls:

Delete

The delete button lives together with the update button top right on the form. It can be hidden, its caption set and its icon set using these same controls.

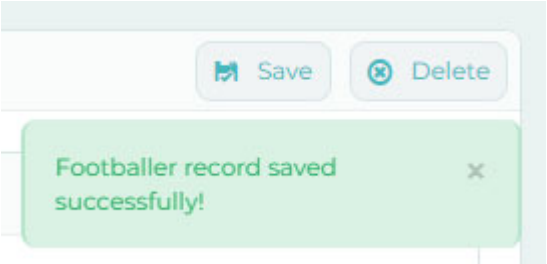
Apply

Apply any changes to persist them before testing.

Test

You can now test your configuration by clicking on the History drop down menu and selecting a footballer to open the form.

Add an X to the end of the Name field. Then press the Save button. The form record update should be confirmed:



Footballer Form: Create

Configure the footballer form to create a record.

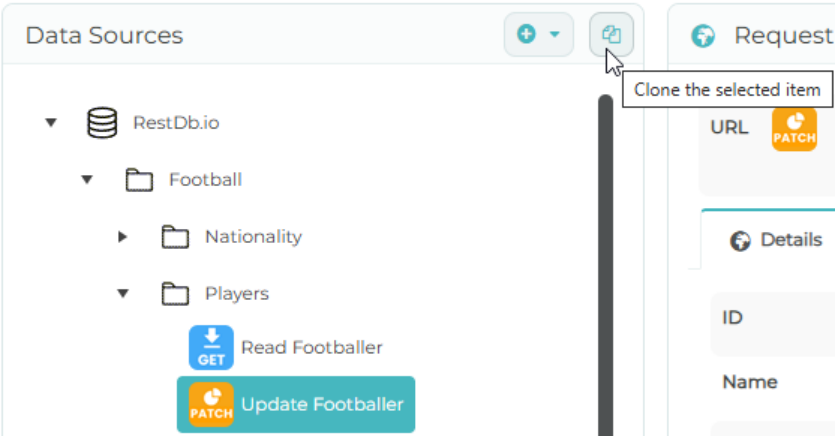
This is the process we will follow.

CREATE

- Add a 'create' data source for creating a single footballer record
- Link the key footballer record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'create' data source to the data entry form
- Configure the app toolbar Add menu to add this data entry form
- Test that we can create a footballer using the Add menu and Update button

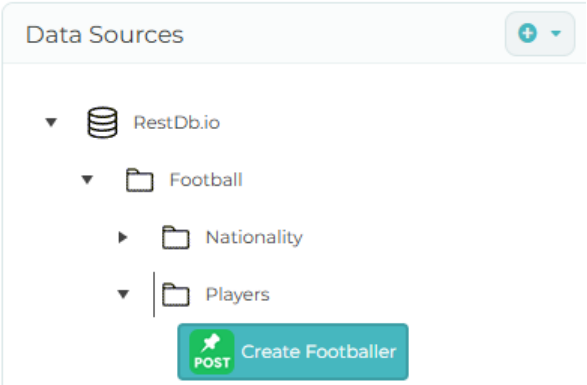
Add a CREATE Data Source

Open App Studio, open the Data Sources configurator, select the Players folder you created previously, then select the previously created "Update Footballer" data source request, and clone it:



The Clone Request modal popup form shows asking you to name the request. Type a meaningful name such as "Create Footballer" and click **Save**.

The request will be added to your tree view beneath the Players folder:



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

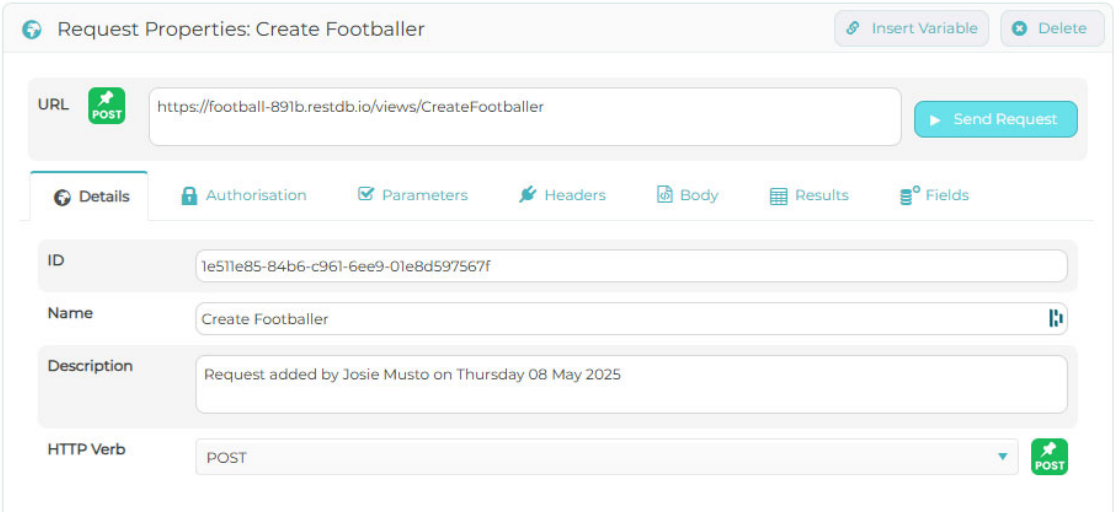
URL

`https://football-891b.restdb.io/views/CreateFootballer` ↗

HTTP Verb

The default GET verb/method is not correct for creating a single record using the ReST API and should be set to POST for this specific back-end end-point.

The request should now look like this:



Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Body

We will be sending the data in the body of the request, and because we cloned the Update data source request, then we know that we can use that as-is:

```
{
    "name": "<#Footballer-Name#>",
    "squadNumber": <#Footballer-SquadNumber#>,
    "biography": "<#Footballer-Biography#>",
    "PhotoURL": "<#Footballer-PhotoURL#>",
    "PositionID": "<#Footballer-PositionID#>",
    "NationalityID": "<#Nationality-ID#>",
    "TeamID": "<#Footballer-TeamID#>"
}
```

Send Request

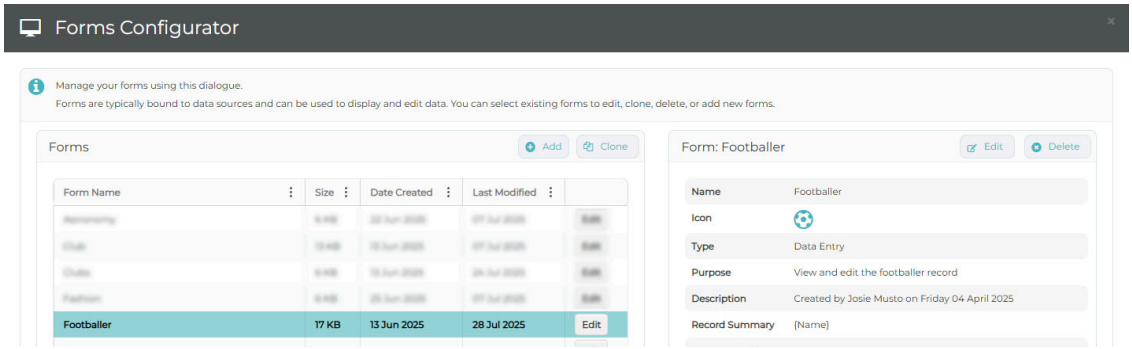
It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to create an existing record in order to test this.

The best way of course to test this is to use the actual Footballer form we created here, and link that form to this data source request, then we can test it.

Edit Form Properties

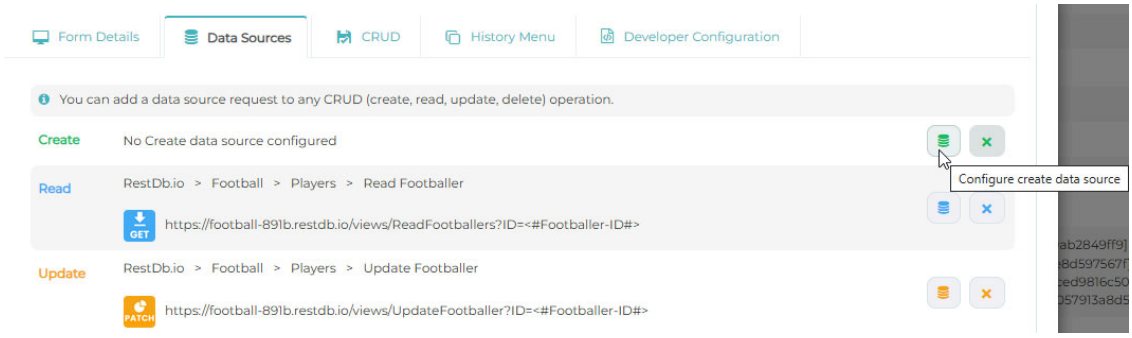
Open App Studio, open the Forms configurator, then select the Footballer form:



Click the **Edit** button to open the form properties modal popup.

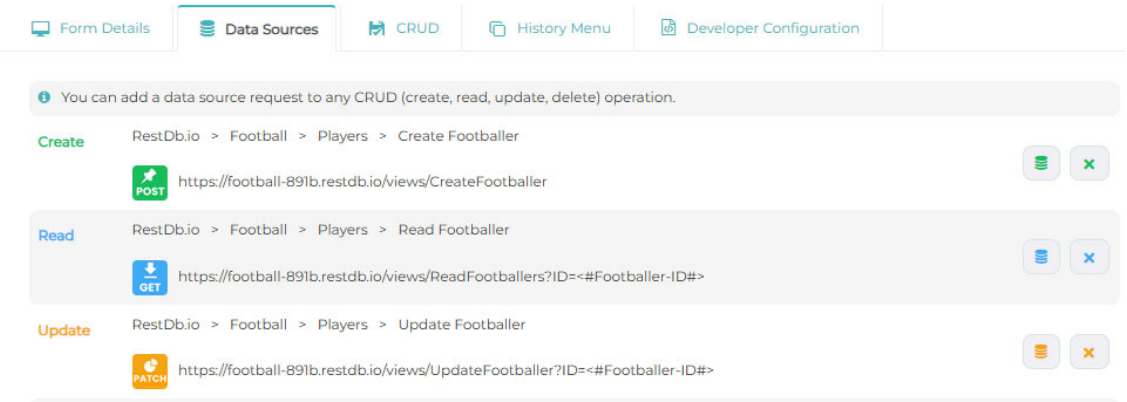
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Create data source:

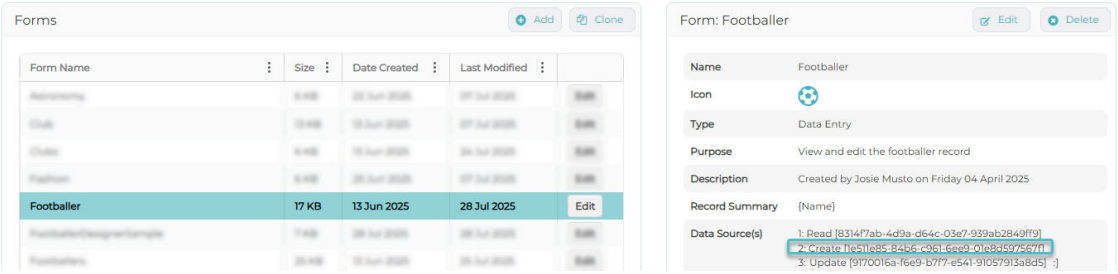


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the **Select** button.

This Create data source request should now appear in the list of assigned data source requests:



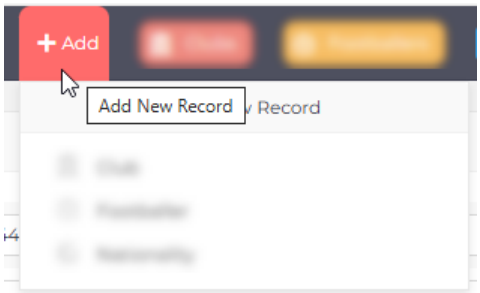
Now click the **Apply** button on this form, in order to persist the form properties. The Create data source request you added should be shown in the form properties list:



Note that we previously assigned the custom variables to the form fields, so we do not need to do this again, as our new data source request uses the same custom variables. We also configured the update/save button, which will automatically call the new create data source method where necessary.

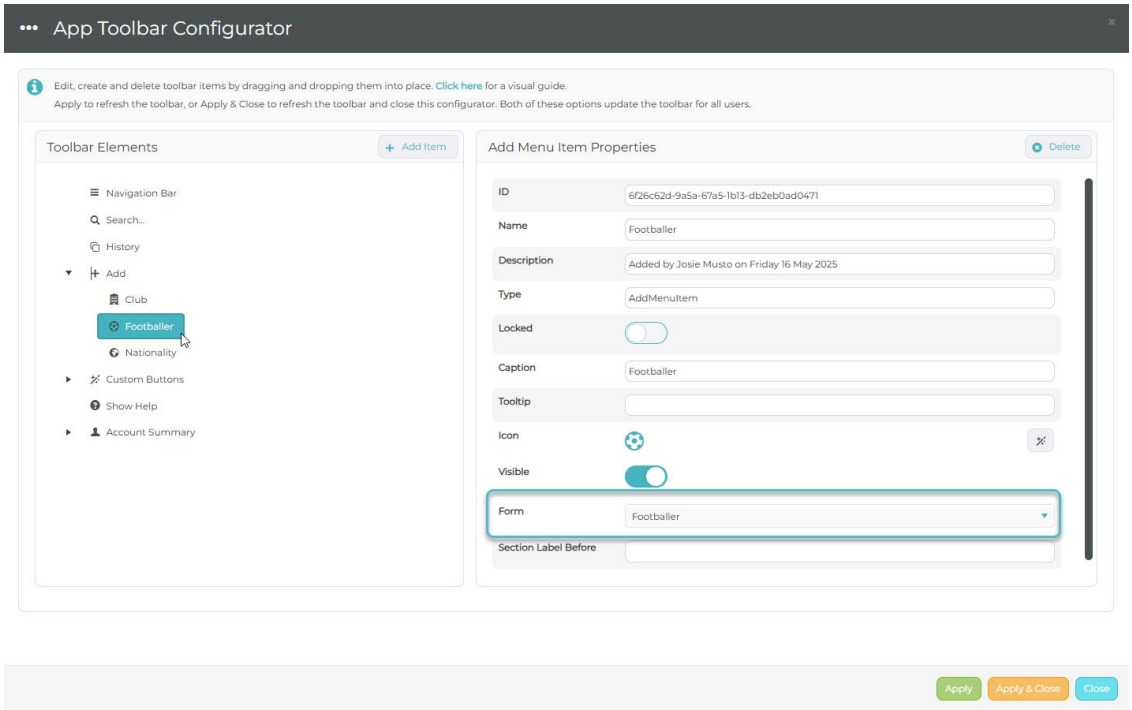
Configure Add Menu

A new record can only be created from the app toolbar using the Add drop down menu:



Use this configurator to add this form to this list.

This is what your **Add** menu should look like in the [App Toolbar Configurator]() after you have created the Footballer form. Note how you will have assigned the form as shown:

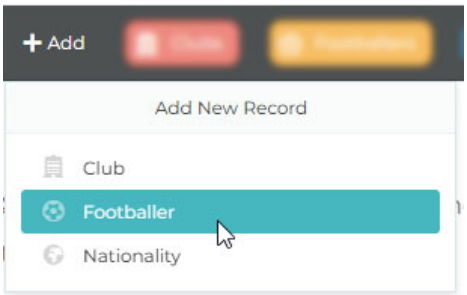


Apply

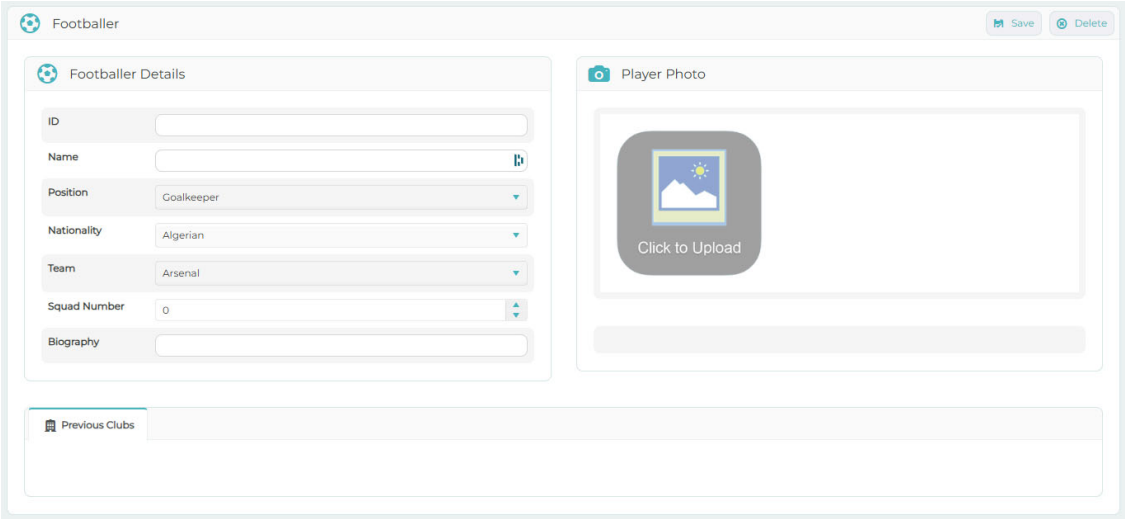
Apply any changes to persist them before testing.

Test

You can now test your configuration by clicking the Add menu button on the toolbar and selecting Footballer:



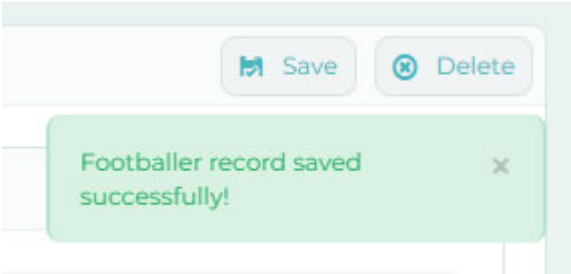
The Footballer form should open:



Note that `Name`, `Photo`, `Squad Number`, `Position`, `Nationality`, and `Team` are all mandatory fields in this sample ReST API so you will need to assign all of these to test the create data source request.

Perhaps choose a player from this winning team ↗ for testing?

Press the **Save** button. The form record creation should be confirmed:



Footballer Form: Delete

Configure the footballer form to delete a record.

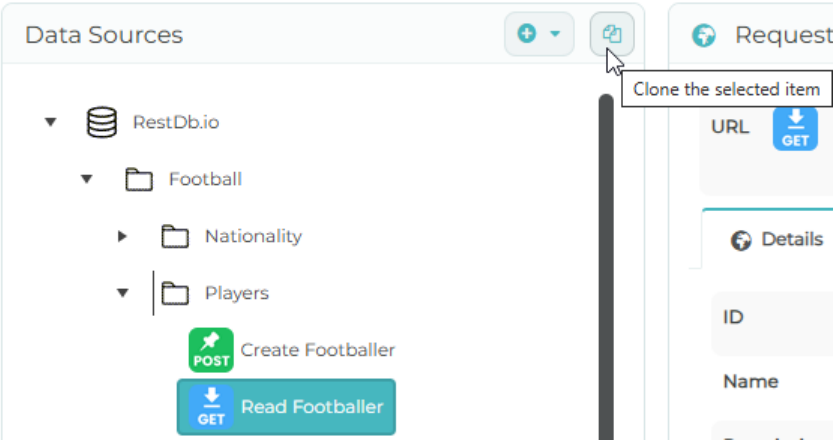
This is the process we will follow.

DELETE

- Add a 'delete' data source for deleting a single footballer record
- Link the key footballer record identifier to this ReST API end-point if necessary
- Map the custom variables mapped to the form fields to the data source body fields
- Assign this 'delete' data source to the data entry form
- Configure the delete button
- Test that we can delete a footballer using the Delete button

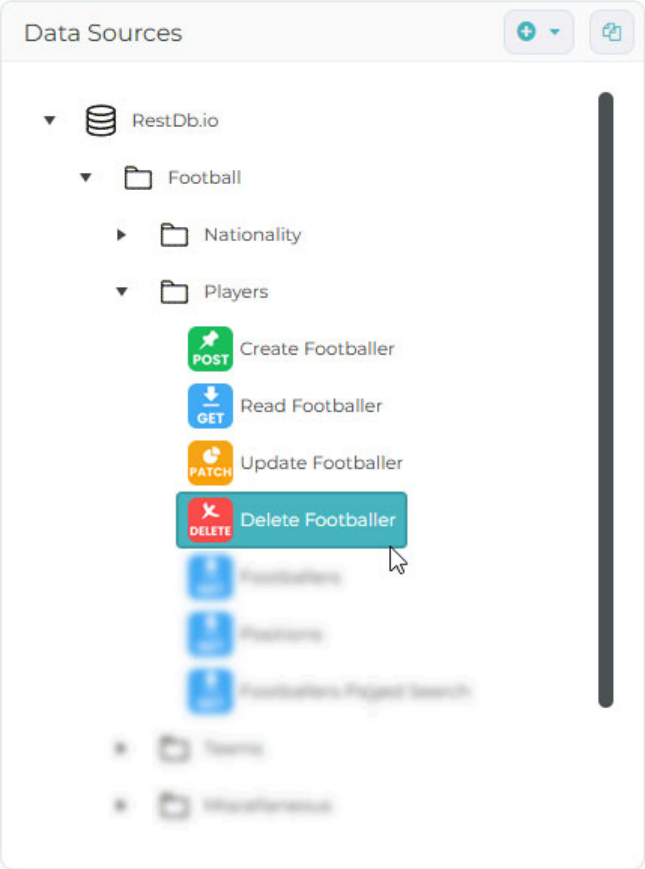
Add a DELETE Data Source

Open App Studio, open the Data Sources configurator, select the Players folder you created previously, then select the "Read Footballer" node and use the clone button to clone this request:



The Clone Request modal popup form shows asking you to name the request. Type a meaningful name such as "Delete Footballer" and click **Save**.

The request will be added to your tree view beneath the Players folder. Take this opportunity to drag and drop the order of the requests to fit the CRUD acronym:



Edit Properties

Edit the properties in the Details tab by referencing this list of ReST API end-points.

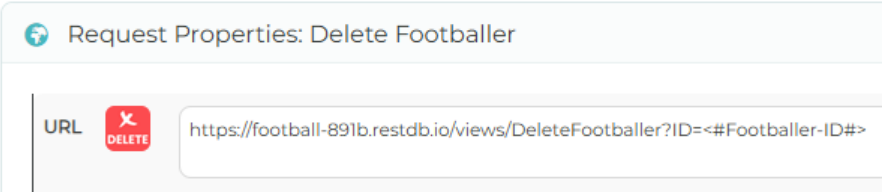
URL

It is known from the documentation that this ReST API end-point has a player identifier property ?ID=.

We therefore type this parameter list into the URL so that it reads:

`https://football-891b.restdb.io/views/DeleteFootballer?ID=` ↗

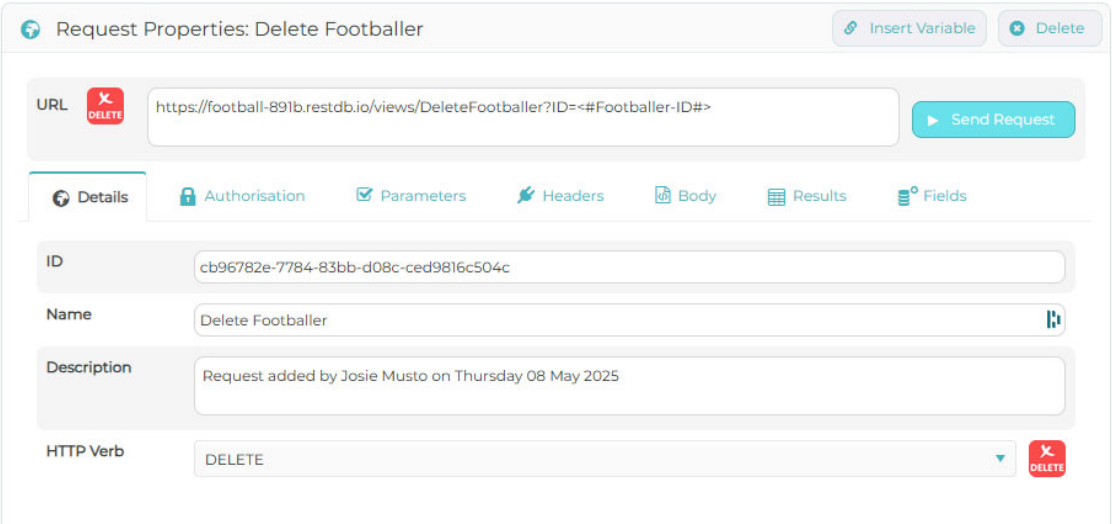
Whilst the caret is still blinking after the last character typed, click on the Insert Variable button to choose the `Footballer-ID` custom variable which when applied should show the custom variable appended to the end:



HTTP Verb

The default GET verb/method is not correct for deleting a single record using the ReST API and should be set to DELETE for this specific back-end end-point.

The request should now look like this:



Authorisation

Previously we set up security credentials at the Football folder level, so we can leave this to be set to "Inherit from parent".

We do not need to change anything else.

Send Request

It is best practice to now click this button on the Details tab to send the request to the ReST API.

The problem however is that we would have to delete an existing record in order to test this.

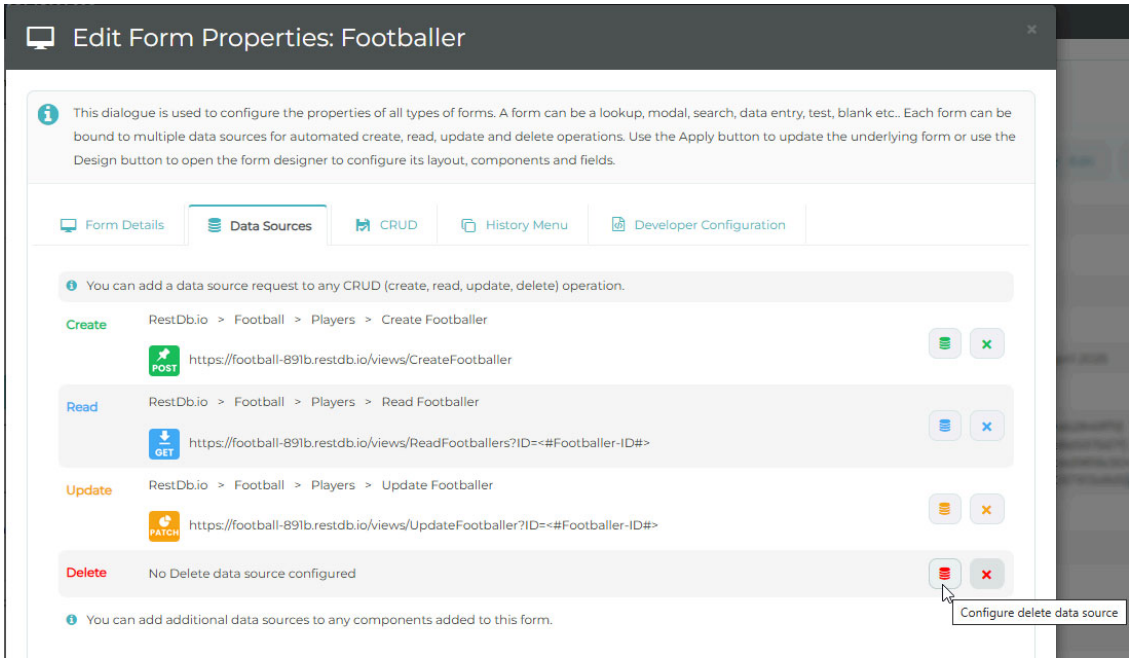
The best way of course to test this is to use the actual Footballer form we created here, and link that form to this data source request, then we can test it.

Edit Form

Open App Studio, open the Forms configurator, then select the Footballer form, then click the Edit button to open the form properties modal popup.

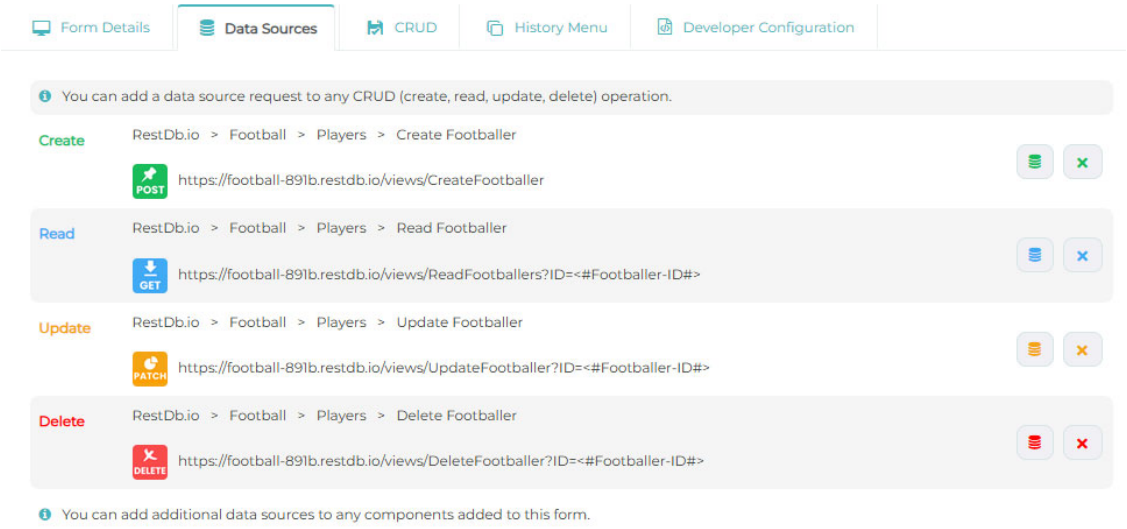
Data Sources

Click the Data Sources tab. There are 4 CRUD data sources lines. Click this database icon to assign the Delete data source:

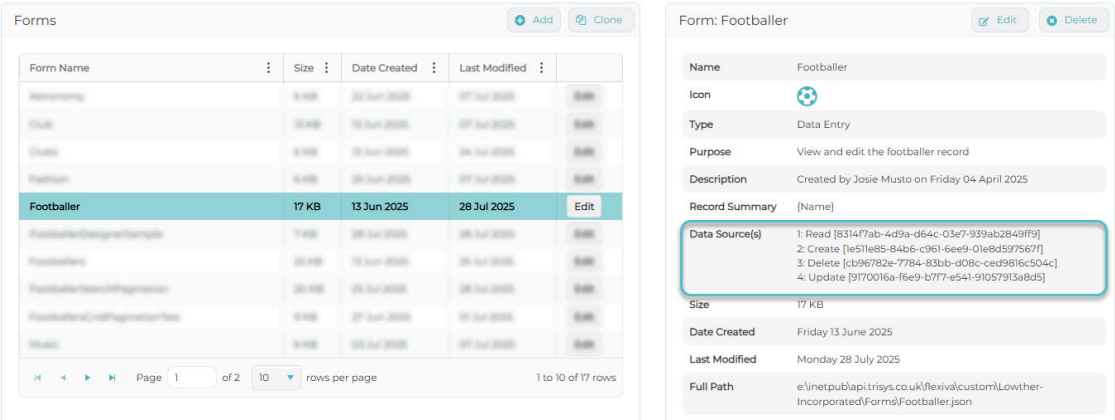


This modal popup form appears to allow you to select the previously created data source request. Navigate through the folder hierarchy until you locate this data source request, then click the **Select** button.

This Delete data source request should now appear in the list of assigned data source requests:



Now click the **Apply** button on this form, in order to persist the form properties. The Delete data source request you added should be shown in the properties list:

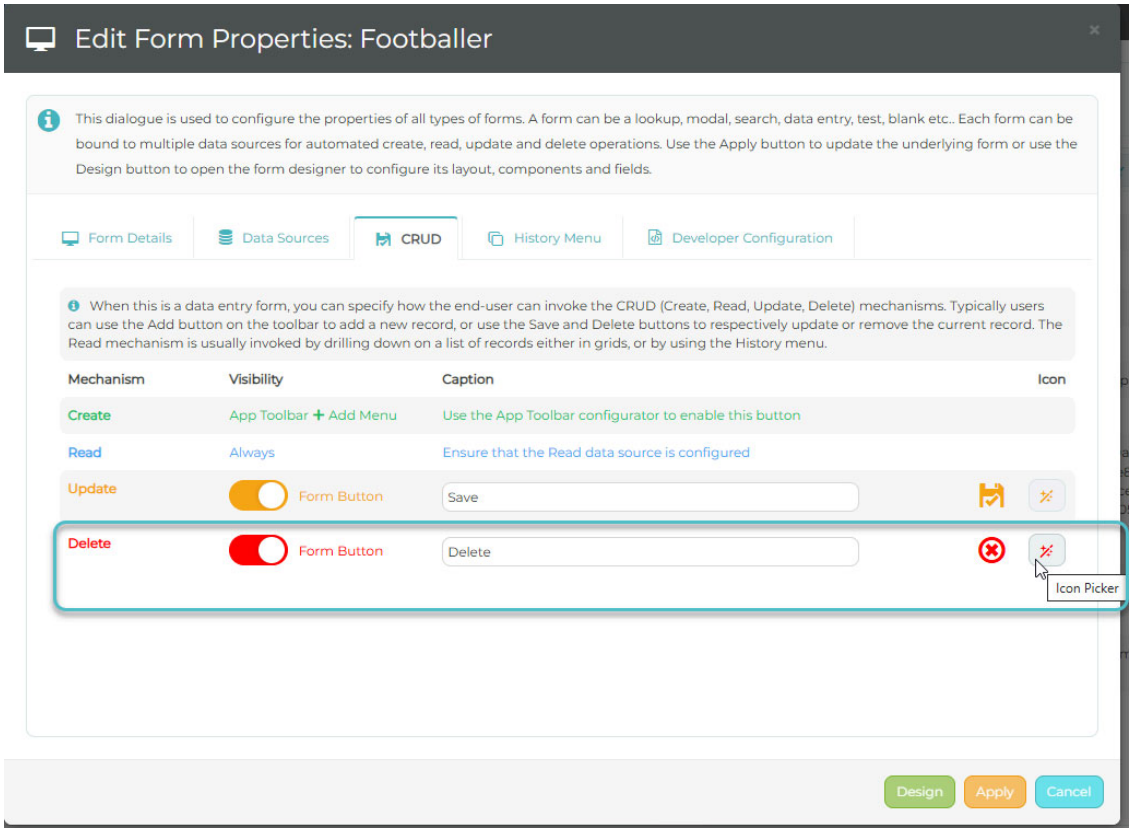


We do not need to design the form.

We will now configure the delete button on the form.

Configure Delete Button

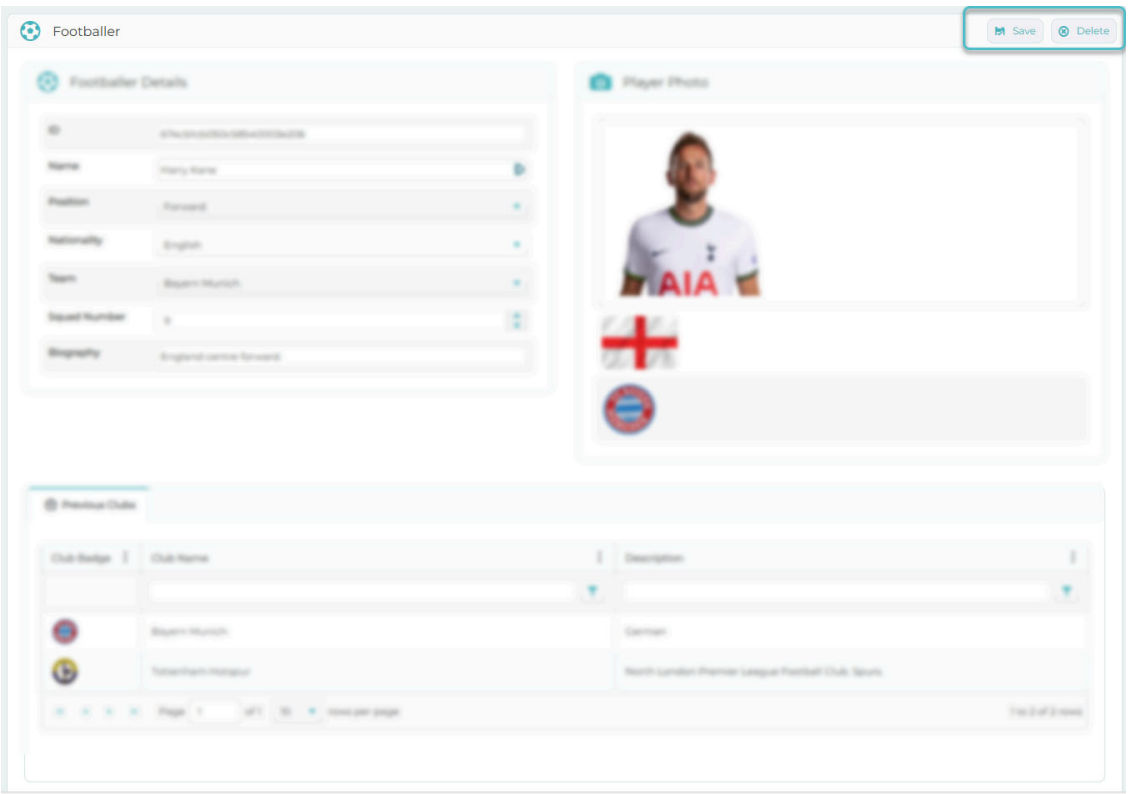
Open App Studio, open the Forms configurator, then select the Footballer form and click the edit button. Then click the CRUD tab:



There are four mechanisms to control form Create, Read, Update and Delete. We are only interested in Delete at this stage.

Delete

The delete button lives together with the update button top right on the form:



It can be hidden, its caption set and its icon set.

Apply

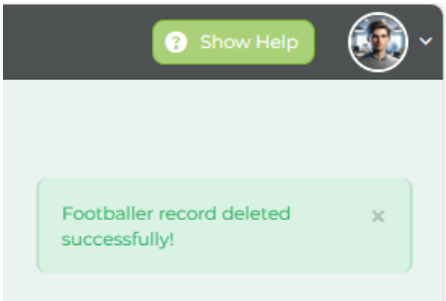
Apply any changes to persist them before testing.

Test

You can now test your configuration by opening the History menu and selecting a previously created footballer.

Press the Delete button. You will be prompted to confirm the deletion.

The form record deletion should be confirmed when the form is closed:


Show Help
Footballer record deleted successfully!

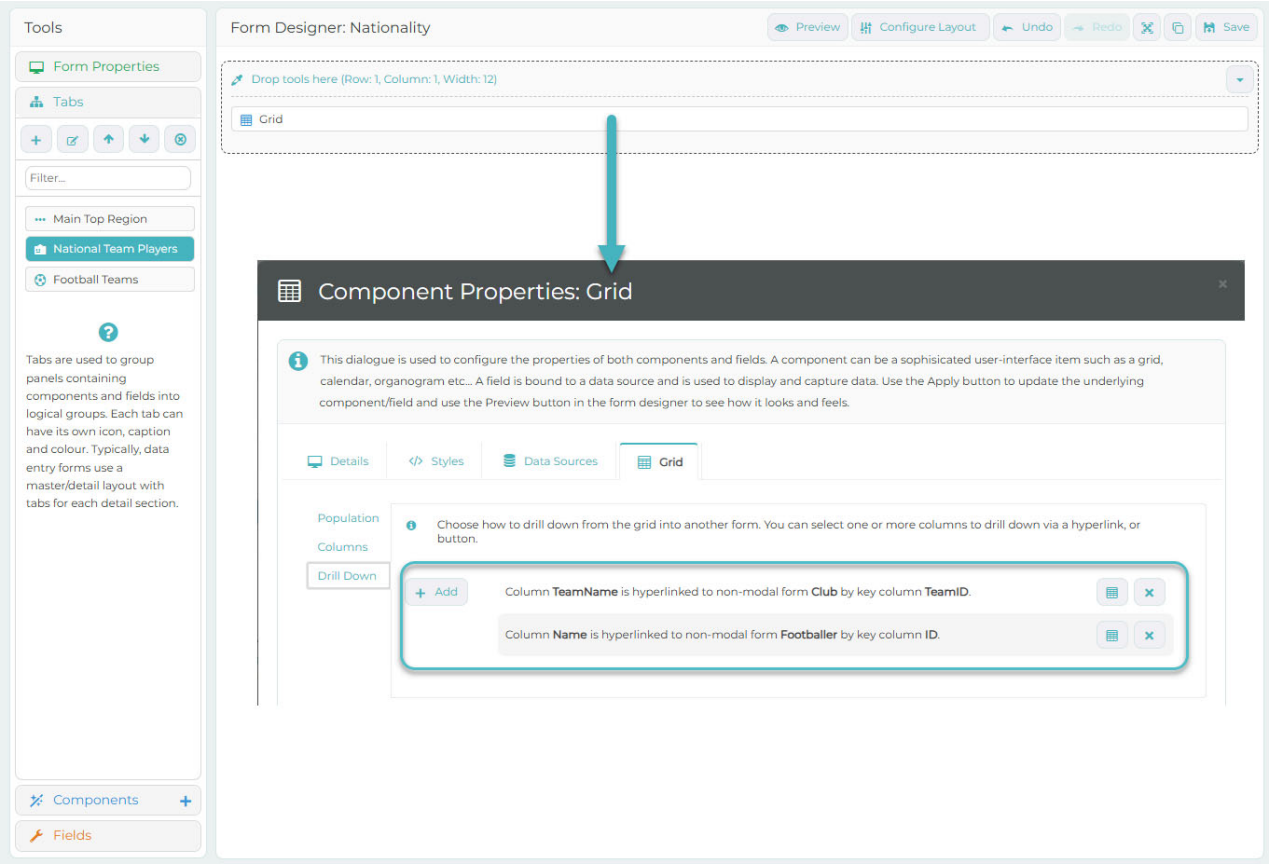
Loose Ends

Tidy up loose ends caused by inter-form dependencies.

We created grids on forms but were unable to drill down until other forms were created.

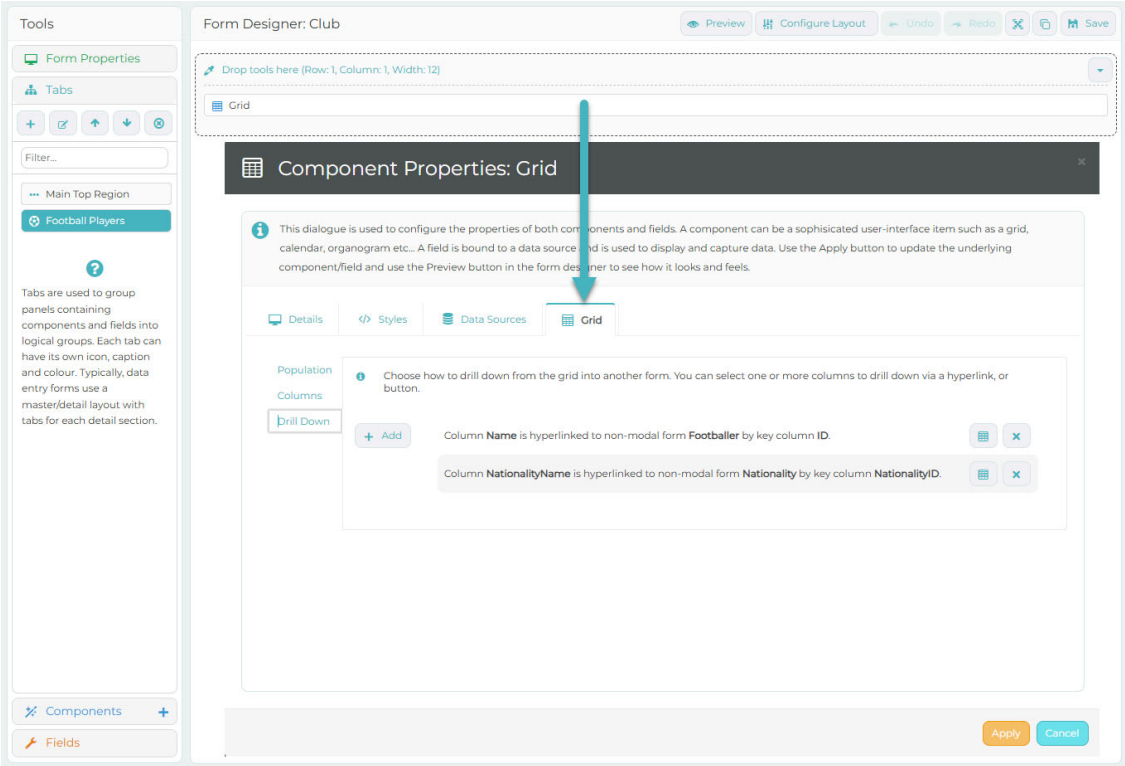
Nationality Form

Open form designer and click on the grid to set the drill down from the National Team Players grid to both the footballer and club forms:



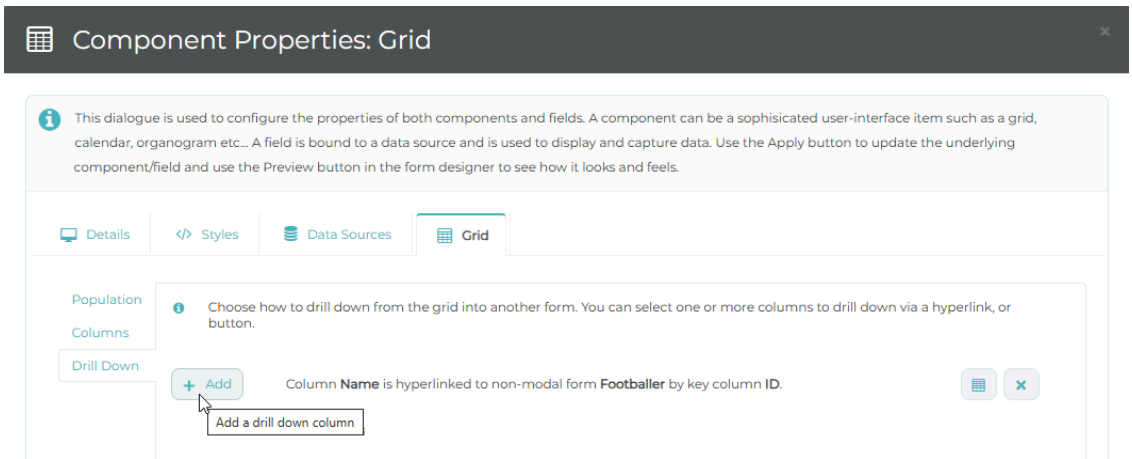
Club Form

Open form designer and click on the grid to set the drill down from the Footballer Players grid to the footballer form.



Footballer Search Form

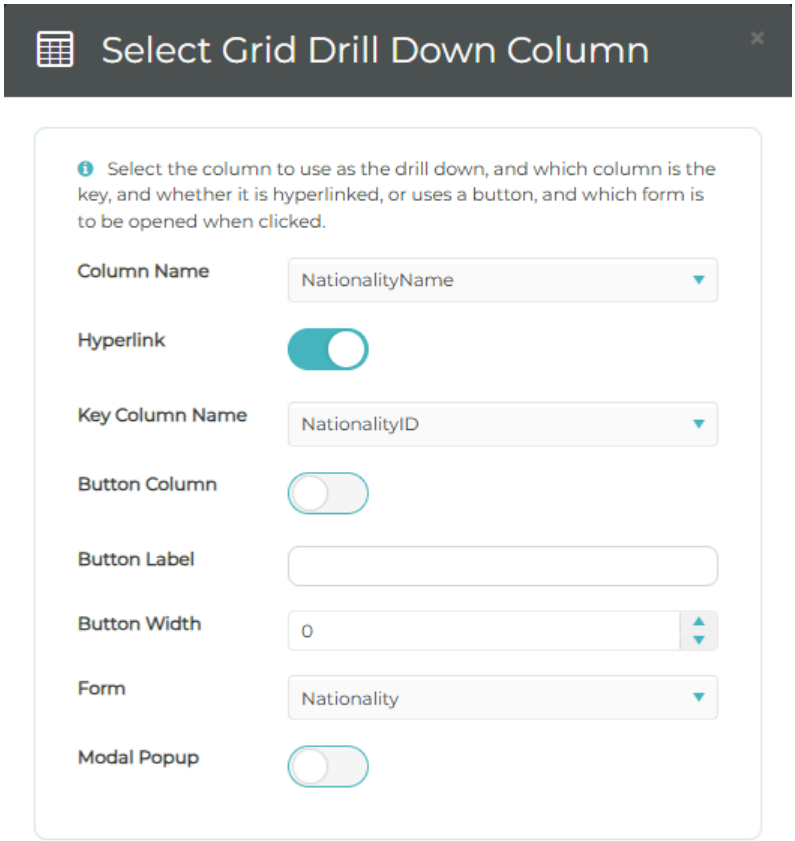
Open form designer and click on the grid to set the drill down from the footballers grid to additional forms.



Nationality Name Column

Add a drilldown column.

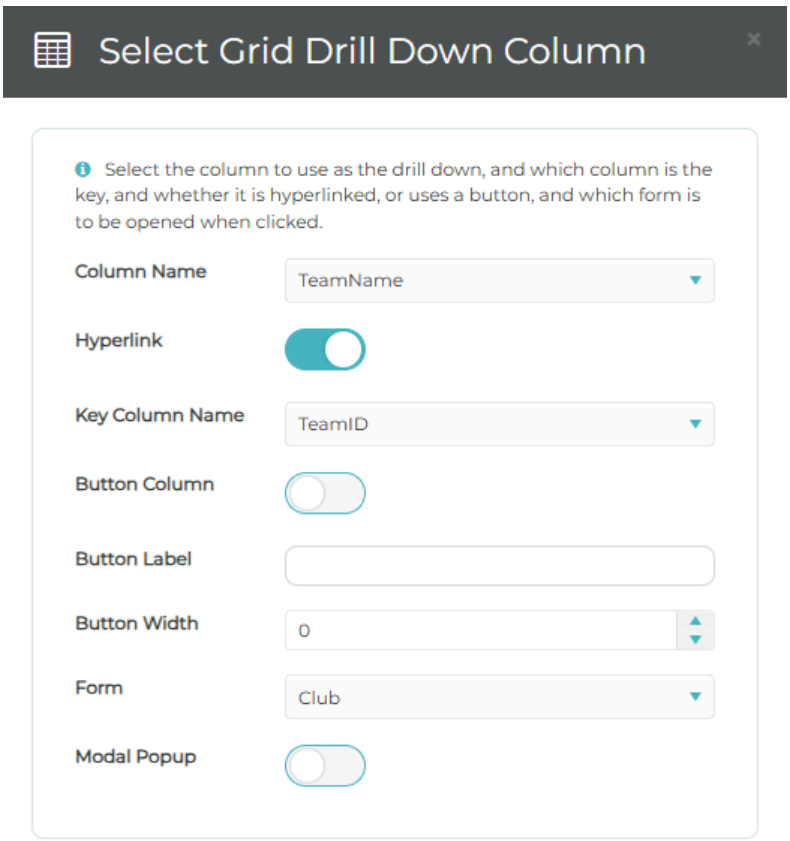
Choose the `NationalityName` column and hyperlink this via the `NationalityID` key column name to the `Nationality` form:



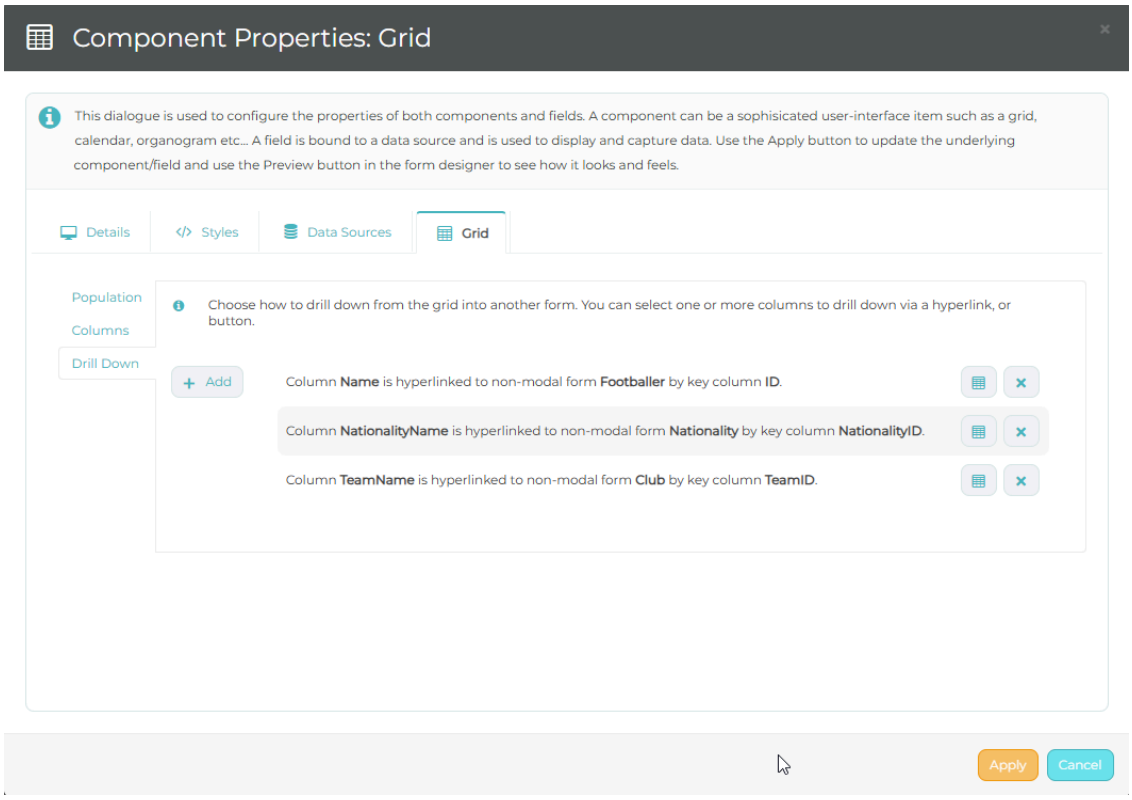
Team Name Column

Add a drilldown column.

Choose the `TeamName` column and hyperlink this via the `TeamID` key column name to the `Club` form:



After adding the additional 2 drill down columns, your Grid component properties should look like this:

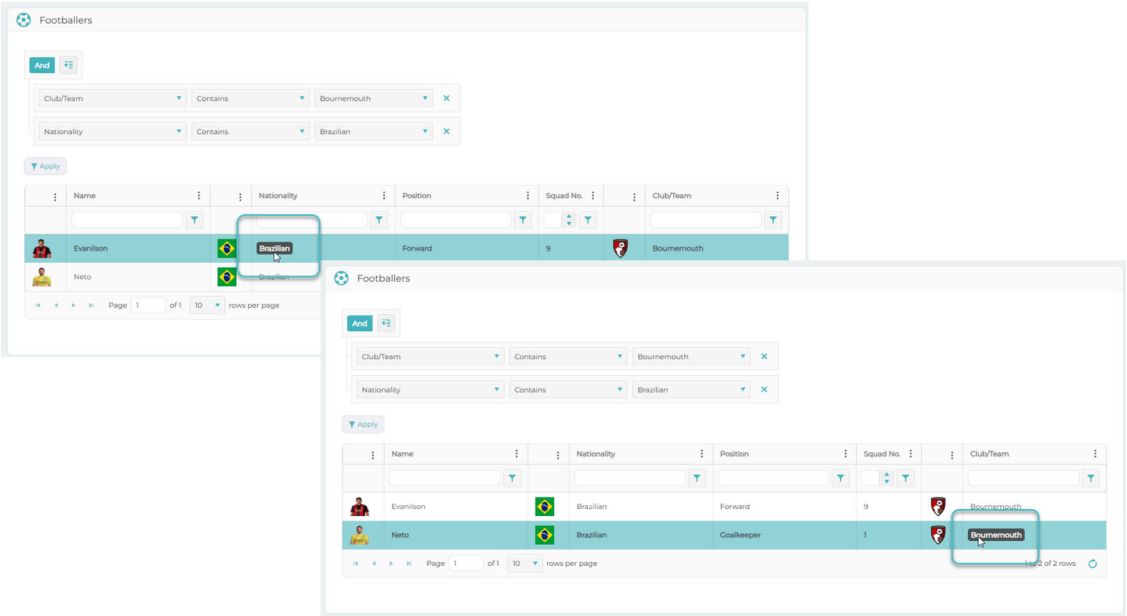


Apply

Apply the changes, then Save the form design to persist.

Test

You should now re-open the Footballer Search form to test that drill down works for both nationality and club/team:



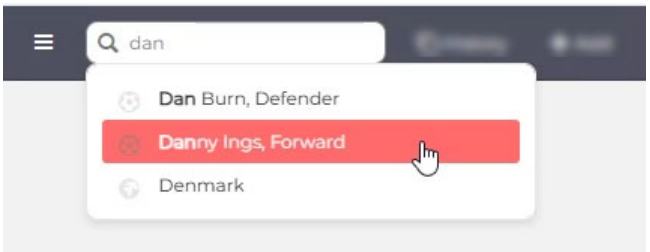
Search

Implement global searching.

End-users can now search for specific entities within your ReST API, so now we want to enable the toolbar Search box to allow them to find any entity with a single search expression.

Typing in text and pressing <Enter> should respond with a drop down list of matching entities and their respective form icon.

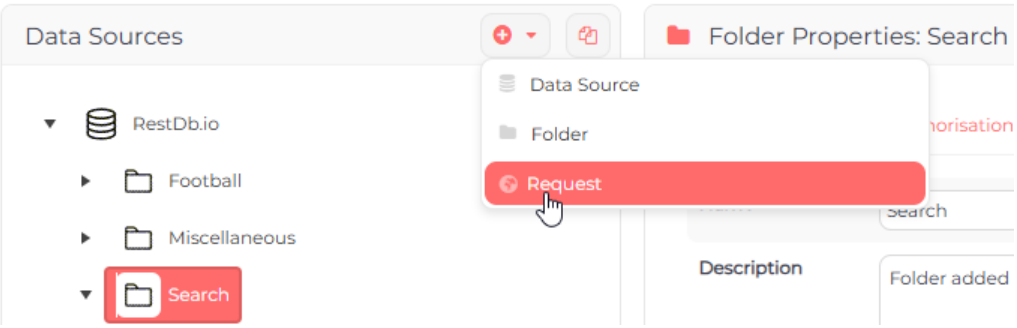
For example this search is finding both footballers and nationalities:



Create a Search Data Source Request

Open App Studio, then open the Data Sources configurator.

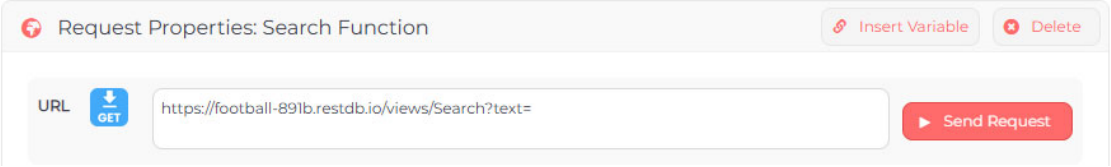
Create a new Search folder beneath your data source folder, then use the add button menu to create a new request:



When the modal popup appears, enter the name of the new request e.g. "Search Function" and Save it.

Copy the ReST API global search function URL and paste this into the URL:

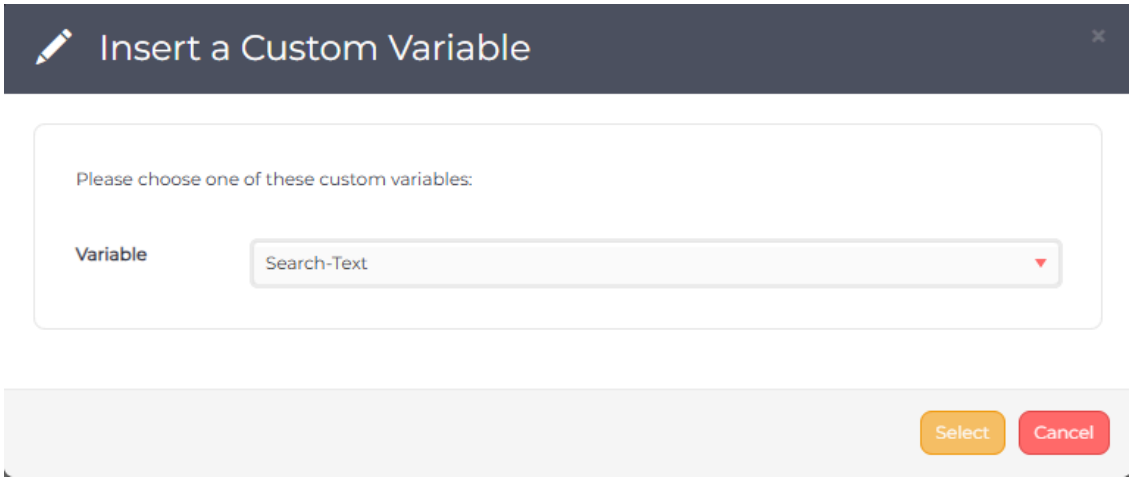
`https://football-891b.restdb.io/views/Search?text=` ↗



Custom Variable

The hard-coded custom variable to use as the text of your URL parameter is called `Search-Text`. This is populated when the end-user types into the app toolbar search text box, so can be sent to your ReST API end-point.

Use the Insert Variable button to insert this into your URL:



The URL should now be:

`https://football-891b.restdb.io/views/Search?text=<#Search-Text#>` ↗

The Authorisation should always default to "Inherit from parent".

Send Request

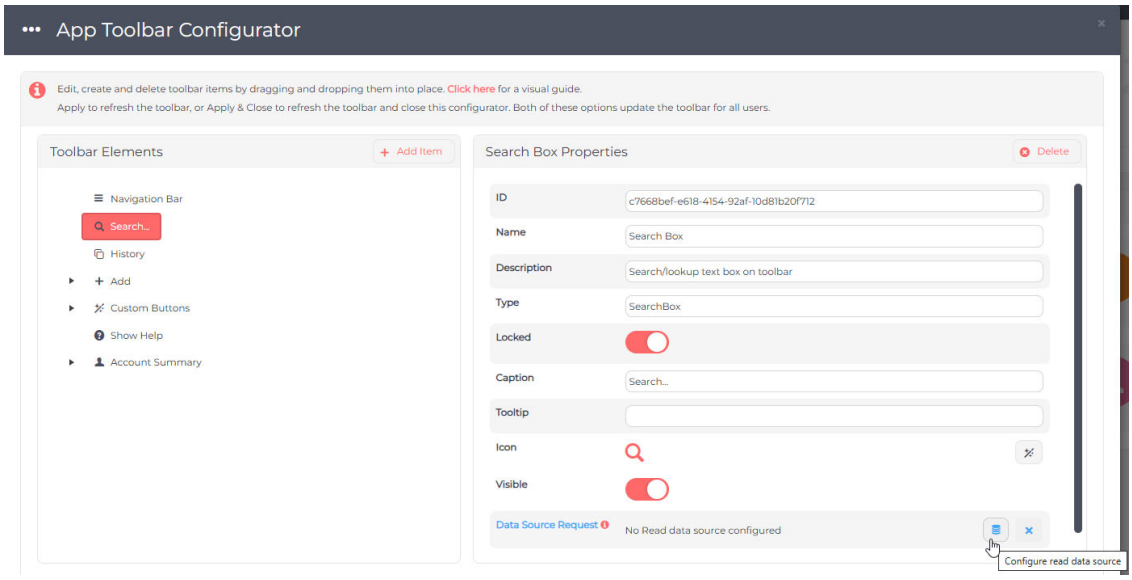
To test this search, use this button to send the request and get the list of fields and data. We can see on the Results tab that the JSON data returned shows the cross-entity search results from the ReST API:

Request Properties: Search Function

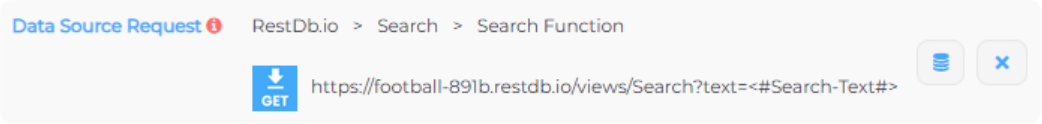

```
        }
    ],
    "DataTable": {
        "List": [
            {
                "EntityName": "Footballer",
                "EntityID": "67fe29f678badf650005d3ee",
                "DisplayName": "Dan Burn, Defender"
            },
            {
                "EntityName": "Footballer",
                "EntityID": "6863ef6c78badf6500137eec",
                "DisplayName": "Danny Ings, Forward"
            },
            {
                "EntityName": "Nationality",
                "EntityID": "6863f5f478badf650013800d",
                "DisplayName": "Denmark"
            }
        ],
```


Assign Data Source Request to Search

Open App Studio, then open the App Toolbar configurator. Select the **Search ...** item in the Toolbar Elements panel:



Click this button to select the above data source request from the modal popup. The assigned data source request should now look like this:



Apply & Close

Apply & Close your changes to persist this configuration.

Test

You can now start typing a range of search expressions into the app toolbar search.

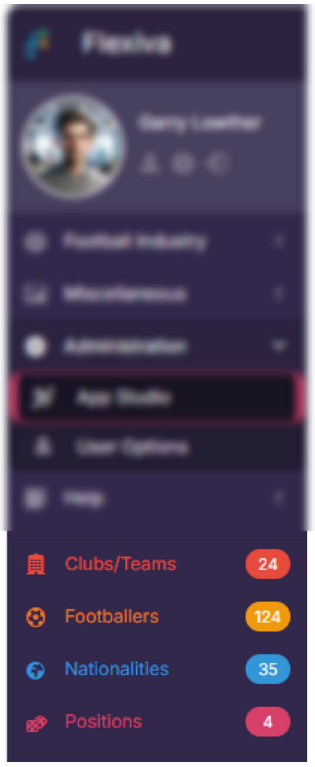
Test that data is returned as expected, and that selecting drop down items loads the correct form and record.

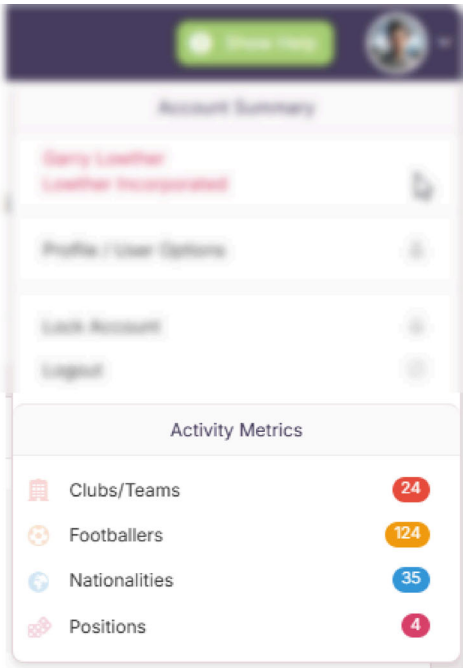
Activity Metrics

Configure visual key performance indicators.

Now that all forms have been created, we can configure the activity metrics to show key performance indicators (KPI's) and allow end-users to drilldown into some forms.

The activity metric KPI's can be shown in both of these places:





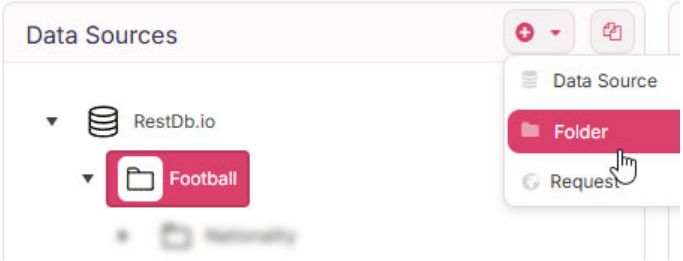
Clicking on a metric can open a form.

We will start by creating the data source requests to pull KPI's from ReST API's.

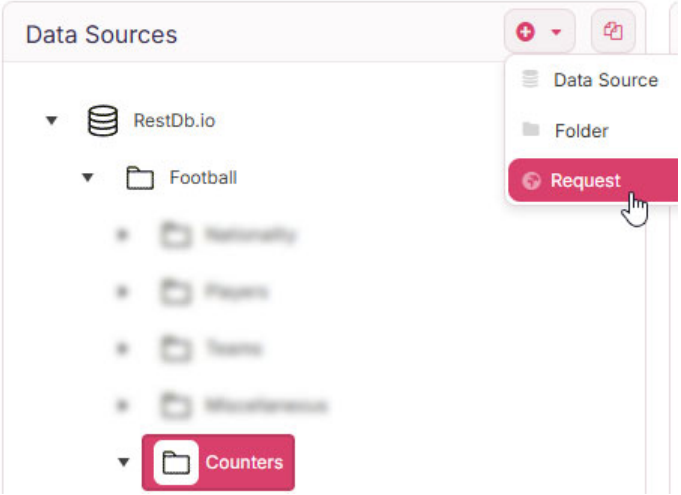
Create Teams Data Source Request

Open App Studio, then open the Data Sources configurator.

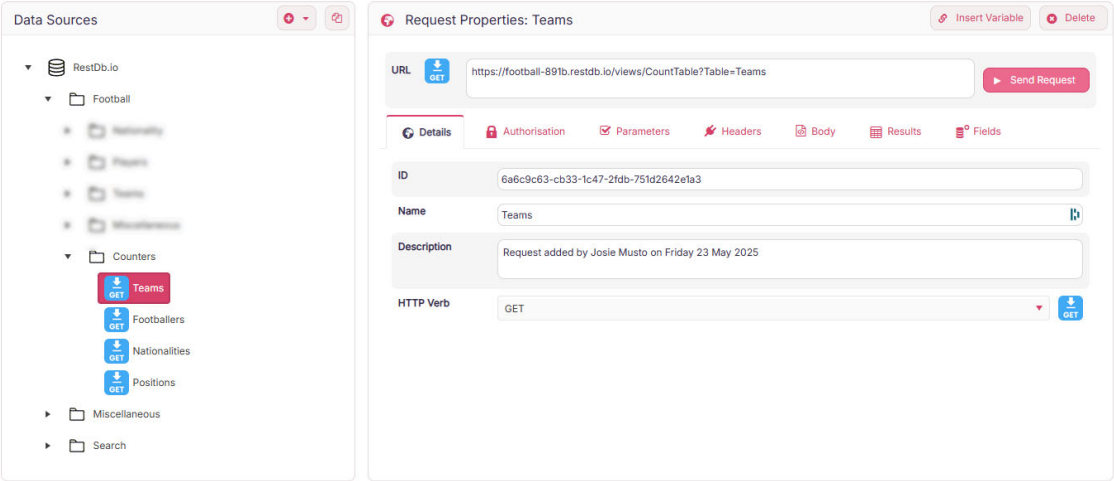
Create a new folder beneath the `Football` folder called `Counters`:



Then create a new request inside this new folder:



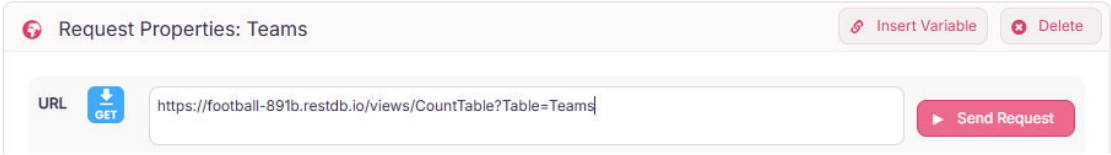
Give this first activity metric the name `Teams`, and save it:



Copy the ReST API Teams KPI function URL:

`https://football-891b.restdb.io/views/CountTable?Table=Teams` ↗

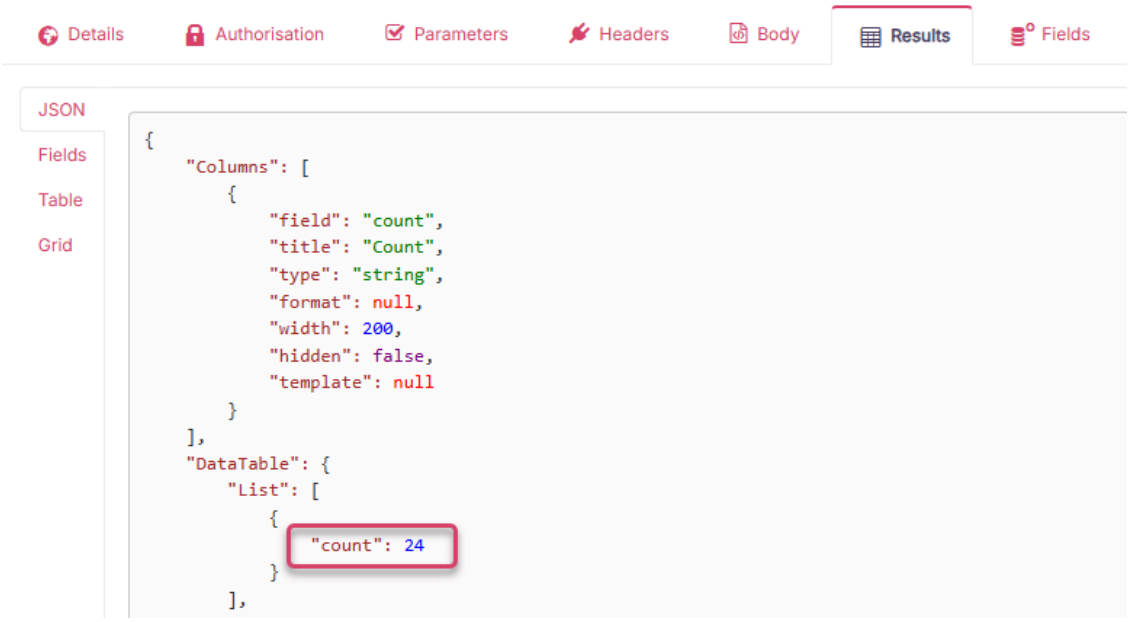
Paste this into the URL:



The Authorisation should always default to "Inherit from parent".

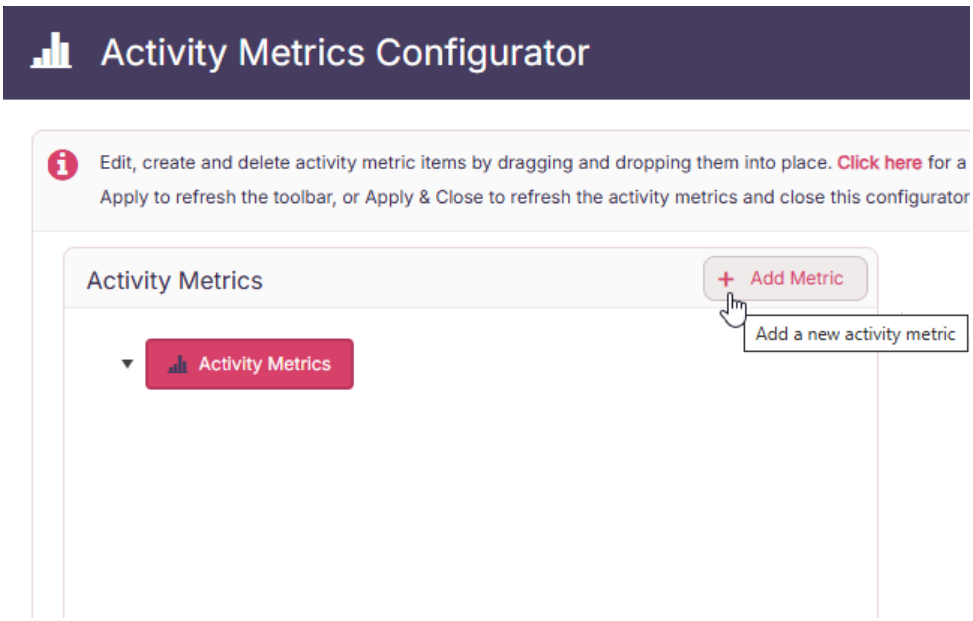
Send Request

To test this, click the Send Request button and get the list of fields and data. We can see on the Results tab that the JSON data returned shows the count from the ReST API:



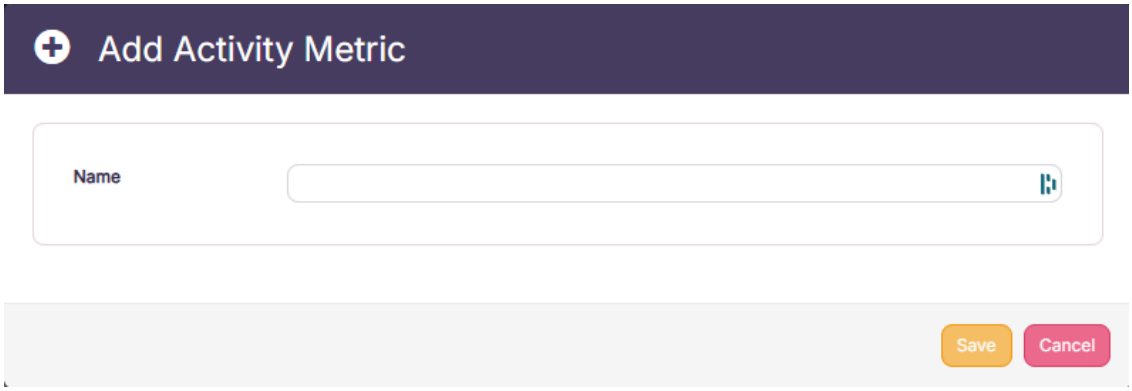
Create Activity Metric

Open App Studio, then open the Activity Metrics configurator:

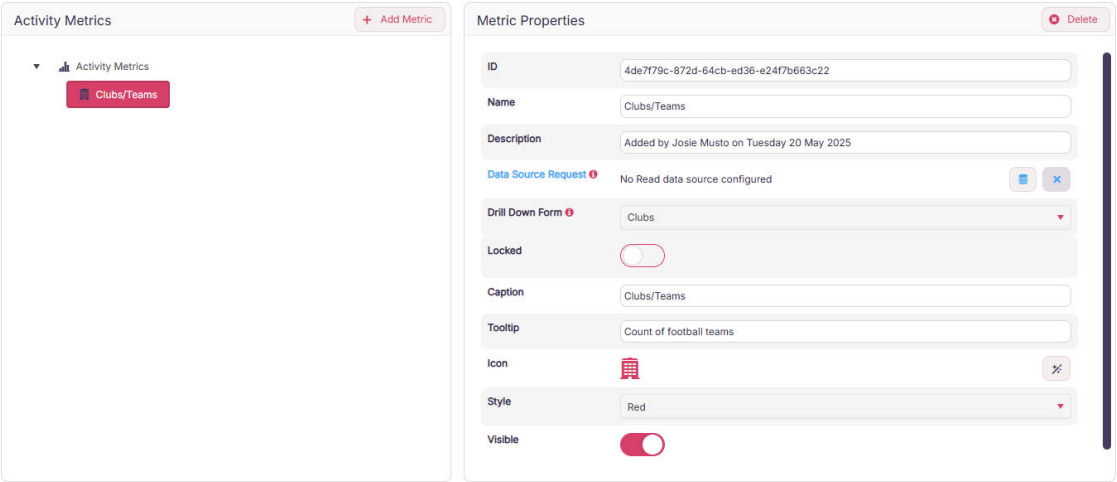


Add Metric

Click this button to open this modal popup:

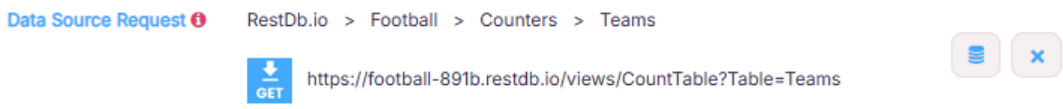


Type the name: `Clubs/Teams` and press the **Save** button. The new activity metric will be created and its properties are displayed:

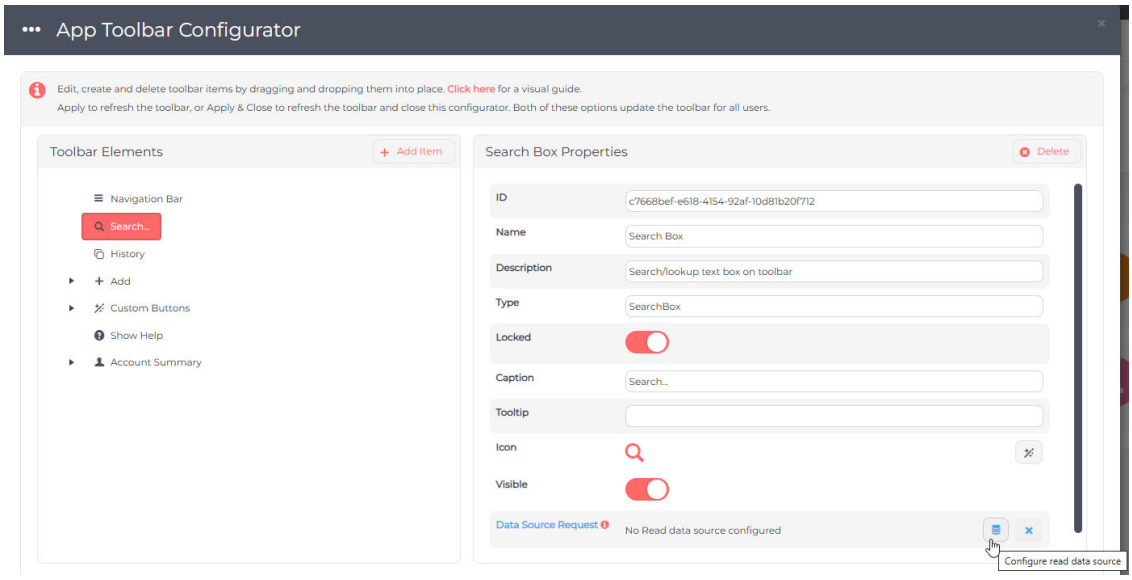


Assign Data Source Request to Metric

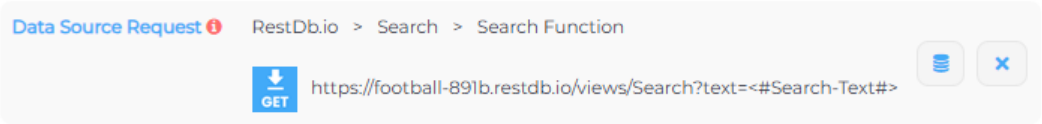
Click the Data Source Request database button to choose the above data source request. This will then show after pressing the Select button:



Open App Studio, then open the App Toolbar configurator. Select the Search ... item in the Toolbar Elements panel:



Click this button to select the above data source request from the modal popup. The assigned data source request should now look like this:



Set Properties

Set these other properties:

Description

Make this as descriptive as possible for future designers and end-users.

Drill Down Form

If the activity metric is to be used as a hyperlink to a form, then choose the form in this drop down list. When the end-user clicks the KPI, the form will open.

Locked

Check this to prevent it being edited by other designers.

Caption

This is the text which will appear to the left of the KPI when displayed.

Tooltip

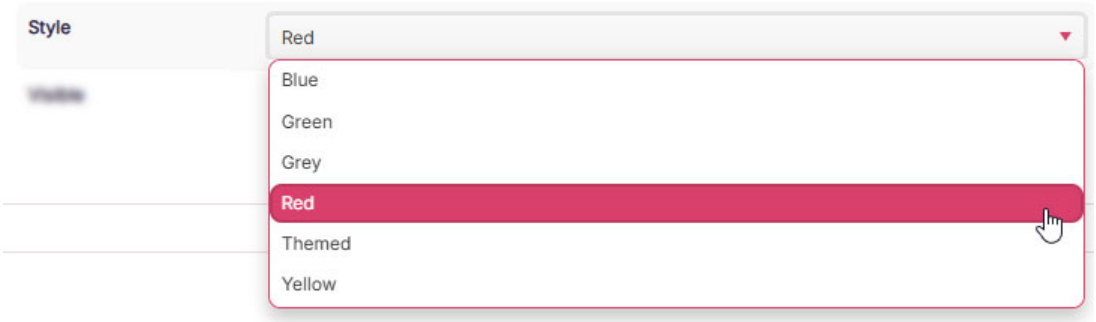
This is the popup text which appears when the end-user hovers their mouse over the KPI. This should speak their language to inform them what this number means.

Icon

The icon displayed to the left of the caption can be selected using the button on the right.

Style

There are limited styles associated with each KPI:



The blue, green, grey, red and yellow are respectively the 'Bootstrap' colours for info, success, default, danger and warning. Themed is the colour chosen in User Options.

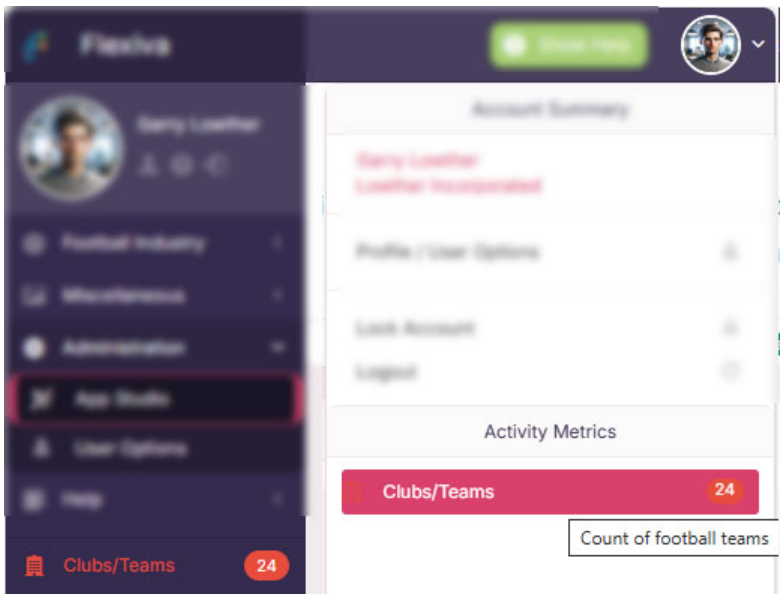
Visible

Check this if the KPI should be visible. Each KPI can also be shown or hidden for specific users using the security configurator.

Apply & Close

Apply & Close your changes to persist this configuration.

The application will reflect your changes as shown in this example:



Here we can see that the new Clubs/Teams metric/KPI is showing the count of 24 obtained from the ReST API.

Clicking the KPI will open the Clubs/Teams lookup form.

Add Additional Metrics

We can now repeat the process for 3 additional metrics:

Footballers

Property	Value
Name	`Footballers`
Description	`Count of footballers`
Data Source Request URL	`https://football-891b.restdb.io/views/CountTable?Table=Players`
Drill Down Form	`FootballerSearchPagination`
Locked	`Unchecked`
Caption	`Footballers`
Tooltip	`Number of football players`
Icon	`Ball`
Style	`Yellow`
Visible	`Checked`

Nationalities

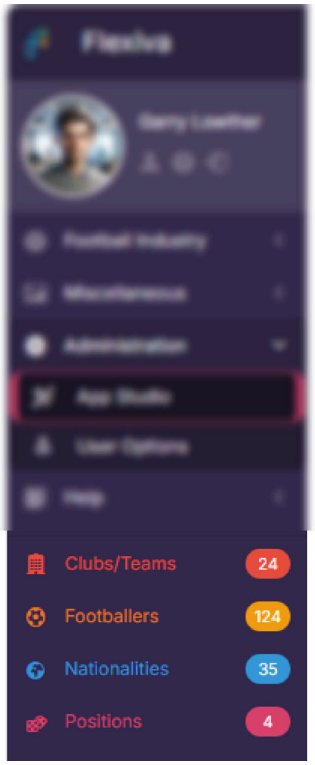
Property	Value
Name	`Nationalities`
Description	`Count of nations`
Data Source Request URL	`https://football-891b.restdb.io/views/CountTable?Table=Nationality`
Drill Down Form	`Nationalities`
Locked	`Unchecked`
Caption	`Nationalities`
Tooltip	`Number of nations`
Icon	`Globe`
Style	`Blue`
Visible	`Checked`

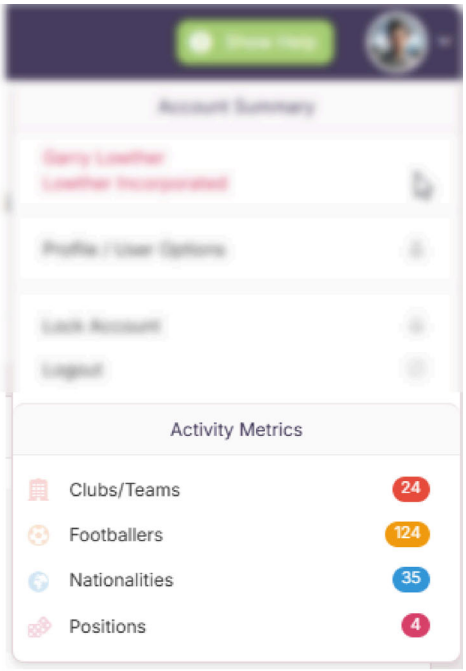
Positions

Property	Value
Name	`Positions`
Description	`Count of positions`
Data Source Request URL	`https://football-891b.restdb.io/views/CountTable?Table=Position`
Drill Down Form	
Locked	`Unchecked`
Caption	`Positions`
Tooltip	`Number of player positions`
Icon	`Dice`
Style	`Themed`
Visible	`Checked`

Test

After all 4 activity metrics have been added, they should look like this:





Test that data is returned as expected, and that clicking each KPI loads the correct form if configured.

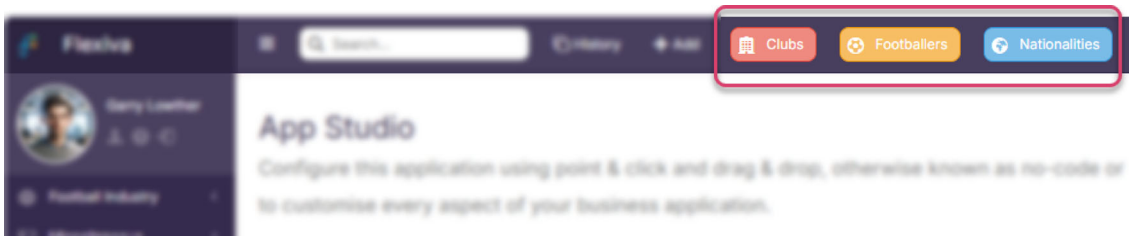
Custom Buttons

Configure custom buttons on the toolbar.

Now that all forms have been created, we can configure custom hyperlink buttons to allow end-users to drilldown into some forms.

In systems with hundreds of functions, allowing end-users to quickly access their favoured forms can be beneficial.

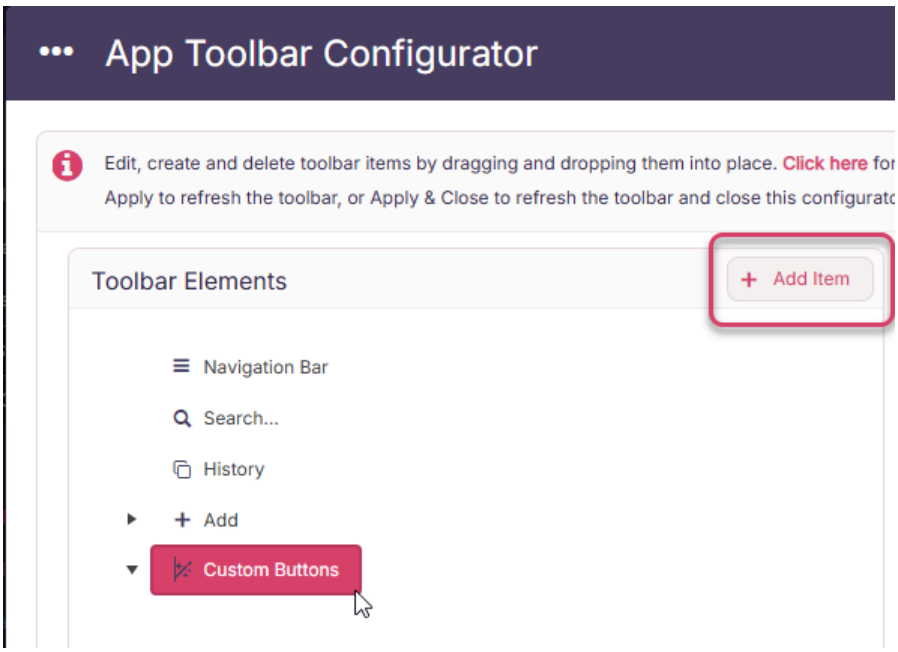
The custom buttons can be shown on the toolbar:



Clicking on a custom button will open a form.

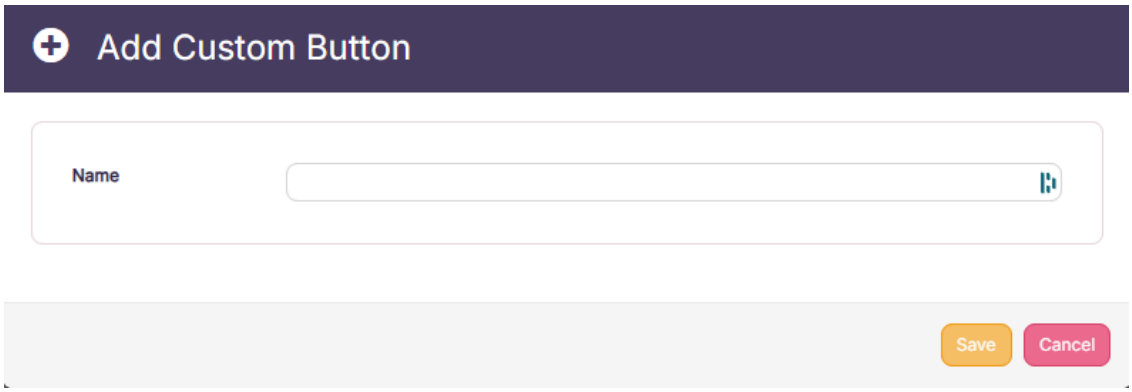
Create a Custom Button

Open App Studio, then open the App Toolbar configurator, then select the Custom Buttons node:

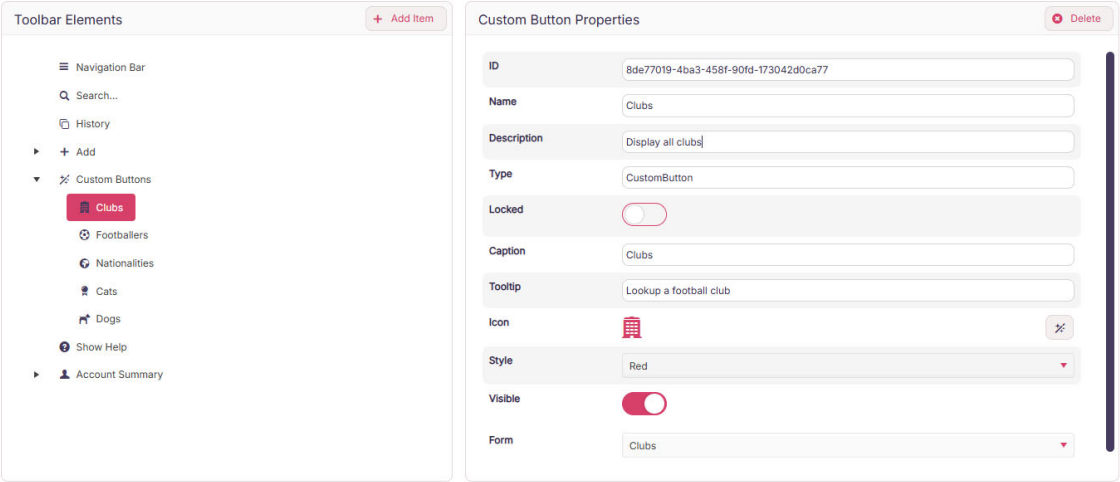


Add Item

Click this button to open this modal popup:



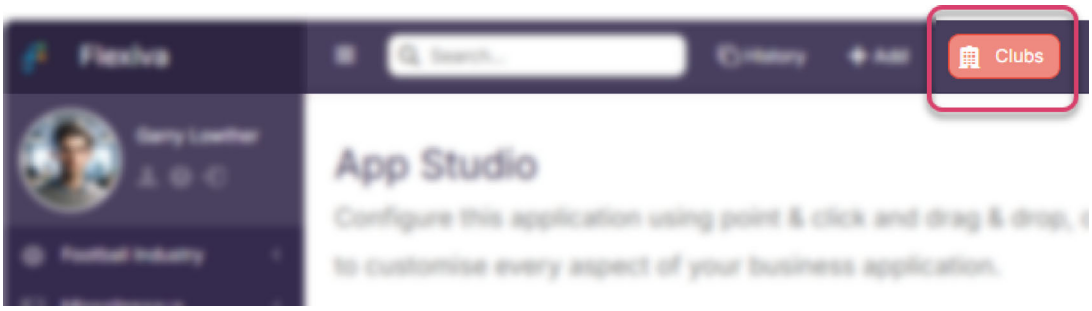
Type the name: `Clubs` and press the **Save** button. The new custom button will be created and its properties are displayed:



Apply & Close

Apply & Close your changes to persist this configuration.

The application will reflect your changes as shown in this example:



Add Additional Custom Buttons

We can now repeat the process for 2 additional metrics:

Footballers

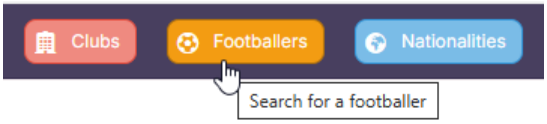
Property	Value
Name	`Footballers`
Description	`Display footballer search`
Type	`CustomButton`
Locked	`Unchecked`
Caption	`Footballers`
Tooltip	`Search for a footballer`
Icon	`Ball`
Style	`Yellow`
Visible	`Checked`
Form	`FootballerSearchPagination`

Nationalities

Property	Value
Name	`Nationalities`
Description	`Display nationalities`
Type	`CustomButton`
Locked	`Unchecked`
Caption	`Nationalities`
Tooltip	`Lookup a nationality`
Icon	`Globe`
Style	`Blue`
Visible	`Checked`
Form	`Nationalities`

Test

After all 3 custom buttons have been added, they should look like this:



Test that clicking each custom button loads the correct form.

Summary

A quick summary of the steps followed to design a production quality CRUD web application connected to live ReST API's.

Congratulations in completing the production designer guide and building your own production quality business application for your colleagues.

You followed a multi step process involving these key mechanisms:

- Created forms
- Added these to the navigation bar
- Created data sources
- Managed custom variables
- Designed your forms by dragging on component and fields
- Configured your forms and components to connect to your data
- Configured activity metrics
- Customised the toolbar Search...
- Customised the toolbar History menu
- Customised the toolbar Add menu
- Managed user accounts
- Applied security
- Tested it comprehensively

You should now be an expert in assembling production quality business applications, and you will therefore have the knowledge to maintain this application as your business and the world changes in the future.

The next section is aimed at software developers building a production quality client-side application using production quality ReST API's.

API

Introduction

Use the application programming interface to consume ReST API app configurations.

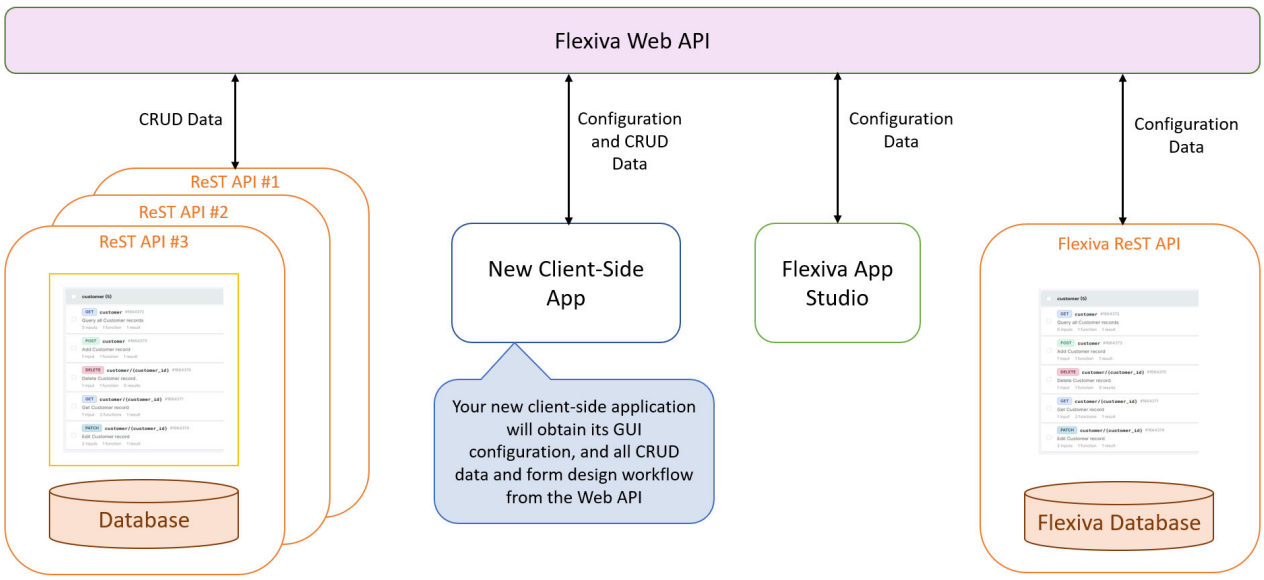
There are numerous client-side applications for Flexiva, all with different styles, themes, colours etc.. and written using a variety of programming languages, frameworks and UI libraries.

This API documentation is aimed primarily at staff and partners who are continually innovating new types of applications, however customers are also invited to build their own custom applications using our API.

This API can be used to build new client-side applications, or server-side automation.

Architecture of a New Flexiva Application

This diagram shows the architecture of how a new client-side application would fit into the Flexiva platform:



- All ReST API’s are connected to Flexiva Web API.
- The Flexiva back-end database is also accessible through the same Web API. This is where the configuration for each customer is stored.
- Flexiva App Studio provides a suite of configurators to customise each customer app. It reads and writes to the Flexiva Web API. It communicates with third party ReST API’s also via the Flexiva Web API.
- New client-side app gets all customer configuration from the Flexiva Web API. It makes CRUD data requests to third party ReST API’s also via the Flexiva Web API.

This architecture frees new client-side apps from worrying about the complex business logic programmed into the back-end ReST API's and each respective custom configuration designed by business specialists. This is the 'separation of concerns' principle, allowing the developer to build a compelling client-side back-end independent front-end application with the worlds current favourite programming language, framework or user interface component library.

API Endpoint

The endpoint for the API is `https://api.trisys.co.uk`

API Key

The API key for developing against the API is `20241002-re90-9e2c-2904-flexiva.app.`

When you move your client-side app into production, we will supply you with a unique key to allow you to monetise your app, or impose tight security rules to limit its use.

Methodology

The API abstracts the complexity of the underlying application configuration, and is used both by the App Studio to configure the client-side CRUD business app, but also used at run-time by the client-side CRUD business app to 'draw' the user interface and present and capture CRUD data via forms.

Login Authentication and Logoff

The end-user must be authenticated either via an e-mail address plus password combination, or via a social network login previously configured by the end-user. If the user has turned on two-factor authentication, then your application must be able to capture and validate this also before allowing users to access their data. The end-user should also be logged off when they request.

Password Reset

Your application will also have to deal with the password reset process to allow end-users who have forgotten their password to reset it.

User Interface Container

Your application must take responsibility for drawing the full UI container including the navigation bar, toolbar, add menu, history menus, custom buttons, activity metrics and F1 help.

The App Studio uses the Font Awesome and Glyph Icons Pro collections for icons assigned to navigation bar, buttons, forms and menus so either these should be available, or a mapping to a similar icon set should be provided.

Forms Subsystem

Your application must also take responsibility for loading custom CRUD forms for both lookup and data entry by understanding the API's relating to the designed forms containing components and fields.

UI Widgets

Your application will probably use its own framework or component library which contains UI widgets such as grids, calendars, date pickers, combo boxes etc.. You will need to be able to map the intrinsic form designed components and fields with your UI widgets.

Custom Variables

Custom variables weld front-end forms with back-end data so your application will need to utilise these to ensure that data is passed between the layers of the application.

Data Source Requests

Your application will need to use the data requests specific API's for all CRUD operations.

Security

You will need the security API's to determine what elements of the designed application are to be made available to which end-user.

Artificial Intelligence

The comprehensive architecture and design of Flexiva plus this extensive API technical documentation will fully support the use of large language model (LLM) generative AI tools to construct client-side applications using the API.

As this LLM ecosystem evolves, and new UI innovations are devised, using AI to generate client-side applications directly from this specification will be possible.

We are very excited to provide this capability to our partners and customers, who understand the importance of future proofing business software.

Login

Authenticate end-users through the application programming interface.

Each end-user must be authenticated during login either via an e-mail address plus password combination, or via a social network login previously configured by the end-user. If the user has turned on two-factor authentication, then your application must be able to capture and validate this also before allowing users to access their data.

Your application will also have to deal with the password reset process to allow end-users who have forgotten their password to reset it.

Authenticate Subscriber

`POST` `Security/Authenticate`

This is a POST method which passes the API Key as a header, together with the credentials in the body. If successful, the session key must be stored and used in all future API requests as this uniquely identifies the subscriber.

Headers

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`

Body

Name	Type	Description
`FullName`	String	e-mail address of the subscriber
`Password`	String	Subscriber password
`Auth0UserObjectBase64`	String	Social network Base64 tokens
`ApplicationName`	String	`Flexiva`
`CustomFolderMandatory`	Boolean	`true`
`URL`	String	URL of your web app

You either supply a FullName + Password, or you supply the `Auth0UserObjectBase64` string which is a Base64 encoded version of this JSON sample received from the auth0 service ↗ which you must implement if you want to enable social network login:

```
{
    "given_name": "Fred",
    "family_name": "Bloggs",
    "nickname": "freddy",
    "name": "Fred Bloggs",
    "picture": "https://lh3.googleusercontent.com/a/ACg8oc...",
    "updated_at": "2025-07-17T07:22:38.346Z",
    "email": "fred.b@domain.com",
    "email_verified": true,
    "sub": "google-oauth2|10024...",
    "authToken": "eyJhbGc.."
}
```

Response

200

```
{
  "Success": true,
  "DataServicesKey": "a GUID cointaining the private session key"
}
```

400

```
{
  "error": "Invalid request"
}
```

Log Off

Authenticated end-users can be logged off using this API.

Each authenticated end-user will be assigned a session key uniquely identifying them for all data requests. Logging the user off invalidates this session key so that it can't be used in future.

Session keys expire after a period of time or if some other event causes the validity of the key to become invalid.

Logoff Subscriber

`POST` `Security/Logoff`

This is a POST method which passes both the API and session keys as headers. If successful, then the session key is invalidated and can't be re-used.

Headers

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Body

No body is required as the API knows who the subscriber is from their session key.

Response

200

```
{
  "Success": true
}
```

400

```
{
  "error": "Invalid request"
}
```

Post Login Subscriber Profile

After login, the authenticated subscriber detail can be requested.

This is usually called when the login authentication API has been successful to return profile information about the logged in subscriber which can be displayed in the UI e.g. name, company, photo etc..

Post Login Subscriber Details

`POST` `Security/PostLoginCRMCredentials`

This is a POST method which passes both the API and session keys as headers. If successful, the subscriber profile details can be stored and displayed. No security credentials are sent in these profile details.

Headers

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Body

No body is required as the API knows who the subscriber is from their session key.

Response

200

```
{
  "Success": true,
  "CDataConnectionKey": {
    "SQLServer": "server-name",
    "Database": "Flexiva",
    "GDrivePath": "UNC-share-name",
    "LoggedInUser": {
        "UserId": 12345,
        "LoginName": "Fred.Bloggs",
        "FullName": "fred.b99@domain.com",
        "ForenameAndSurname": "Fred Bloggs",
        "CompanyName": "Boggy Frogs",
        "ContactId": 67890,
        "Contact": {
            "DateFirstRegistered": "2025-06-13T12:33:39",
            "Comments": "Added by TriSys API on Friday 13 June 2025
12:30:13",
            "ContactId": 54321,
            "FullName": "Fred Bloggs",
            "Forenames": "Fred",
            "Surname": "Blogs",
            "SurnameInitial": "B",
            "CompanyName": "Boggy Frogs",
            "CompanyId": 4567,
            "CompanyAddressStreet": "Hills Road",
            "CompanyAddressCity": "Cambridge",
            "CompanyAddressCounty": "Cambridgeshire",
            "CompanyAddressPostCode": "CB1 1BH",
            "CompanyAddressCountry": "United Kingdom",
            "JobTitle": "Software Engineer",
            "ContactType": "User",
            "Status": "Active",
            "WorkEMail": "fred.b99@domain.com",
            "ContactPhotoURL":
"https://api.trisys.co.uk/Chat/Profiles//dall-e.png",
            "CompanyIndustry": "Service Industry"
        },
        "CRMContact": {
            "ApexDeveloper": true,
            "SendApexScreenshot": false,
            "CustomForms": [
                {
                    "FileName": "Astronomy.json",
                    "FilePath": "custom/boggy-
frogs/forms/astronomy.json",
                    "FullServerPath": null
                }
            ],
```

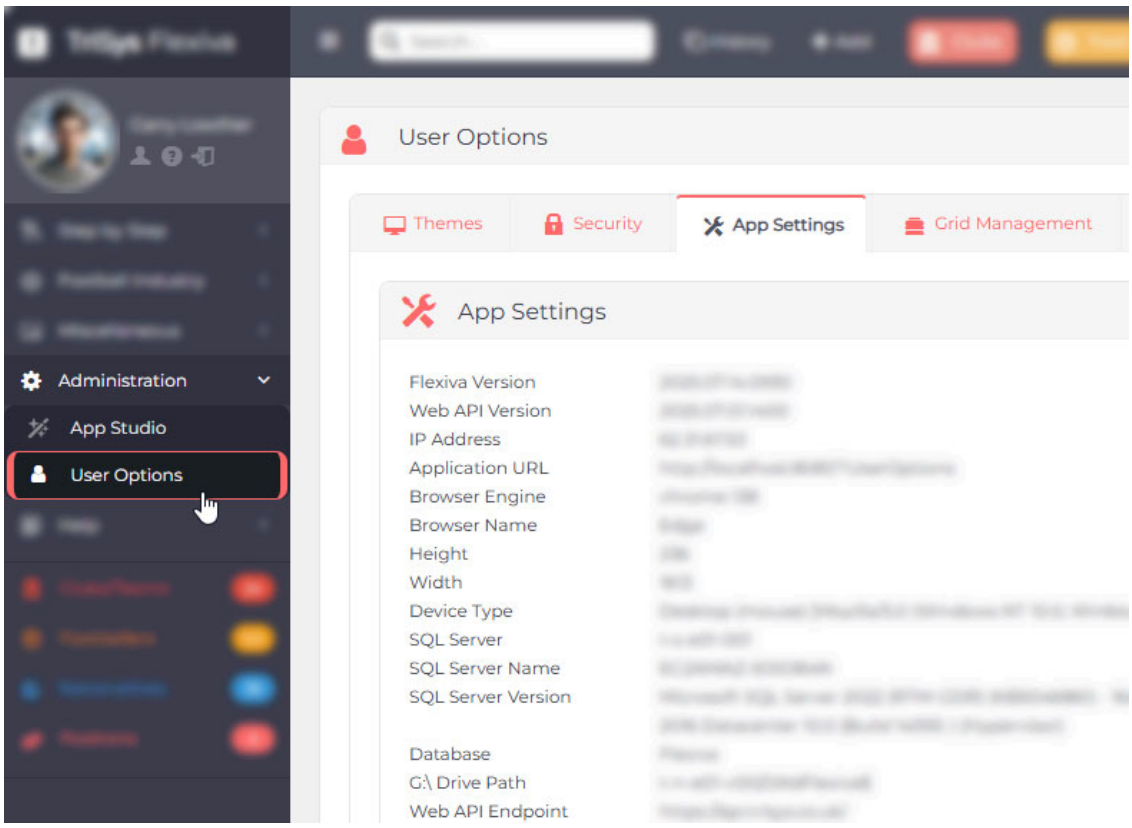
```
            "Subscriber": {
                "AccountMaintainer": true,
                "UserHasRemoteDesktop": false,
                "CompanyHasRemoteDesktops": false,
                "UserHasPlatformixWebSites": false,
                "ASPUserType": "Flexiva",
                "ASPdbName": "Flexiva"
            },
            "ContactId": 123456,
            "FullName": "Fred Bloggs",
            "Forenames": "Fred",
            "Surname": "Bloggs",
            "SurnameInitial": "B",
            "CompanyName": "Boggy Frogs",
            "JobTitle": "Software Engineer",
            "ContactType": "Customer",
            "WorkEMail": "fred.b99@domain.com"
        }
    },
    "TrialAccount": false,
    "AuthenticatedType": 12,
    "AccountType": 0,
    "ApplicationName": "Flexiva",
    "SQLServerVersion": "Microsoft SQL Server 2022...",
    "SQLServerName": "EC2AMAZ-1234INMU"
  }
}
```


400


```
{
  "error": "Invalid request"
}
```


Supplementary Information

Some of this information is available in the Flexiva application in User Options.



The logged in user is the subscriber, who is logged into a back-end customer database on our cloud as a user, which also has a contact record in our back-end CRM database. These details are not connected to a customers own ReST API for which they will have their own back-end databases.

Read Navigation Bar

After login, the navigation bar can be requested for the authenticated subscriber.

This is usually called when the login authentication API has been successful to return the navigation bar configuration.

When an end-user opens a form, it is this navigation bar data which determines whether the form exists, and then the form design can be read and rendered using this API call.

Read App Studio Designed Navigation Bar

`POST` `AppStudio/ReadNavigationBar`

This is a POST method which passes both the API and session keys as headers, and a body. If successful, the navigation bar can be displayed.

Headers

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Body

Name	Type	Description
`TreeFormat`	Boolean	`true`

Response

200

```
{
  "Success": true,
  "JSON": "base64 encoded string"
}
```

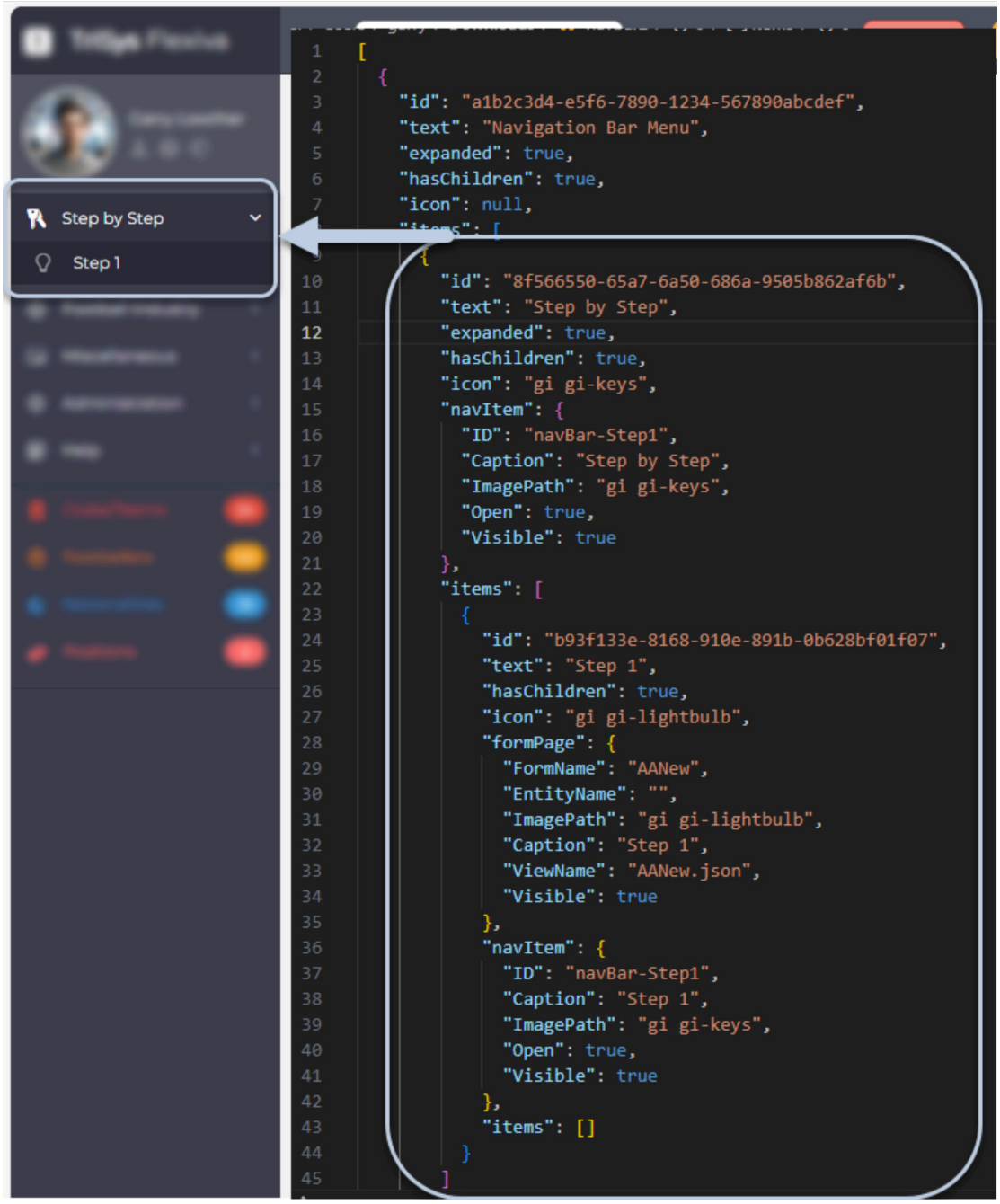
400

```
{
  "error": "Invalid request"
}
```

Base64 JSON Decoding

The JSON data is Base64 encoded. You must decode this into a JSON string and convert that into a multi-hierarchical object.

This example shows the rendered navigation bar on the left and the top portion of the multi-hierarchical object returned from the API.



Here is an example of a full navigation bar.

```
[
  {
    "id": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
    "text": "Navigation Bar Menu",
    "expanded": true,
    "hasChildren": true,
    "icon": null,
    "items": [
      {
        "id": "8f566550-65a7-6a50-686a-9505b862af6b",
        "text": "Step by Step",
        "expanded": true,
        "hasChildren": true,
        "icon": "gi gi-keys",
        "navItem": {
          "ID": "navBar-Step1",
          "Caption": "Step by Step",
          "ImagePath": "gi gi-keys",
          "Open": true,
          "Visible": true
        },
        "items": [
          {
            "id": "b93f133e-8168-910e-891b-0b628bf01f07",
            "text": "Step 1",
            "hasChildren": true,
            "icon": "gi gi-lightbulb",
            "formPage": {
              "FormName": "AANew",
              "EntityName": "",
              "ImagePath": "gi gi-lightbulb",
              "Caption": "Step 1",
              "ViewName": "AANew.json",
              "Visible": true
            },
            "navItem": {
              "ID": "navBar-Step1",
              "Caption": "Step 1",
              "ImagePath": "gi gi-keys",
              "Open": true,
              "Visible": true
            },
            "items": []
          }
        ]
      },
      {
        "id": "3c4b5a69-d8e7-f691-a2b3-c4d5e6f7a8b9",
```

```
        "text": "Football Industry",
        "expanded": false,
        "hasChildren": true,
        "icon": "gi gi-global",
        "navItem": {
          "ID": "navBar-FootballIndustry",
          "Caption": "Football Industry",
          "ImagePath": "gi gi-global",
          "Open": true,
          "Visible": true
        },
        "items": [
          {
            "id": "8f7e6d5c-4b3a-2918-7654-321098765432",
            "text": "Clubs/Teams",
            "hasChildren": false,
            "icon": "gi gi-bank",
            "formPage": {
              "FormName": "Clubs",
              "EntityName": "",
              "ImagePath": "gi gi-bank",
              "Caption": "Clubs/Teams",
              "ViewName": "Clubs.json",
              "Visible": true
            },
            "navItem": {
              "ID": "navBar-FootballIndustry",
              "Caption": "Football Industry",
              "ImagePath": "gi gi-global",
              "Open": true,
              "Visible": true
            },
            "items": []
          },
          {
            "id": "2d3e4f5a-6b7c-8901-2345-6789abcdefgh",
            "text": "Club/Team",
            "hasChildren": false,
            "icon": "gi gi-bank",
            "formPage": {
              "FormName": "Club",
              "EntityName": "",
              "ImagePath": "gi gi-bank",
              "Caption": "Club/Team",
              "ViewName": "Club.json",
              "Visible": false,
              "EntityForm": true,
              "RecordId": 0
```

```
          },
          "navItem": {
            "ID": "navBar-FootballIndustry",
            "Caption": "Football Industry",
            "ImagePath": "gi gi-global",
            "Open": true,
            "Visible": true
          },
          "items": []
        },
        {
          "id": "7a8b9c0d-1e2f-3456-7890-abcdef123456",
          "text": "Footballers",
          "hasChildren": false,
          "icon": "gi gi-soccer_ball",
          "formPage": {
            "FormName": "Footballers",
            "EntityName": "",
            "ImagePath": "gi gi-soccer_ball",
            "Caption": "Footballers",
            "ViewName": "Footballers.json",
            "Visible": true
          },
          "navItem": {
            "ID": "navBar-FootballIndustry",
            "Caption": "Football Industry",
            "ImagePath": "gi gi-global",
            "Open": true,
            "Visible": true
          },
          "items": []
        },
        {
          "id": "b5c6d7e8-f9a0-1234-5678-90abcdef1234",
          "text": "Footballer",
          "hasChildren": false,
          "icon": "gi gi-soccer_ball",
          "formPage": {
            "FormName": "Footballer",
            "EntityName": "",
            "ImagePath": "gi gi-soccer_ball",
            "Caption": "Footballer",
            "ViewName": "Footballer.json",
            "Visible": false,
            "EntityForm": true
          },
          "navItem": {
            "ID": "navBar-FootballIndustry",
            "Caption": "Football Industry",
```

```
      "Caption": "Football Industry",
      "ImagePath": "gi gi-global",
      "Open": true,
      "Visible": true
    },
    "items": []
  },
  {
    "id": "4d5e6f7a-8b9c-0123-4567-89abcdef0123",
    "text": "Nationalities",
    "hasChildren": false,
    "icon": "gi gi-globe_af",
    "formPage": {
      "FormName": "Nationalities",
      "EntityName": "",
      "ImagePath": "gi gi-globe_af",
      "Caption": "Nationalities",
      "ViewName": "Nationalities.json",
      "Visible": true
    },
    "navItem": {
      "ID": "navBar-FootballIndustry",
      "Caption": "Football Industry",
      "ImagePath": "gi gi-global",
      "Open": true,
      "Visible": true
    },
    "items": []
  },
  {
    "id": "9e8f7a6b-5c4d-3e2f-1a0b-9c8d7e6f5a4b",
    "text": "Nationality",
    "hasChildren": false,
    "icon": "gi gi-globe_af",
    "formPage": {
      "FormName": "Nationality",
      "EntityName": "",
      "ImagePath": "gi gi-globe_af",
      "Caption": "Nationality",
      "ViewName": "Nationality.json",
      "Visible": false,
      "EntityForm": true
    },
    "navItem": {
      "ID": "navBar-FootballIndustry",
      "Caption": "Football Industry",
      "ImagePath": "gi gi-global",
      "Open": true,
      "Visible": true
```

```
            },
            "items": []
          }
        ]
      },
      {
        "id": "69479713-c93b-6098-b529-502a7153301a",
        "text": "Miscellaneous",
        "expanded": false,
        "hasChildren": true,
        "icon": "gi gi-picture",
        "navItem": {
          "ID": "navBar-Miscellaneous",
          "Caption": "Miscellaneous",
          "ImagePath": "gi gi-picture",
          "Visible": true
        },
        "items": [
          {
            "id": "d464aee6-8083-4bb1-d8ad-3718a48913f9",
            "text": "Astronomy",
            "hasChildren": true,
            "icon": "gi gi-star",
            "formPage": {
              "FormName": "Astronomy",
              "EntityName": "",
              "ImagePath": "gi gi-star",
              "Caption": "Astronomy",
              "ViewName": "Astronomy.json",
              "Visible": true
            },
            "navItem": {
              "ID": "navBar-Miscellaneous",
              "Caption": "Miscellaneous",
              "ImagePath": "gi gi-warning_sign"
            },
            "items": []
          },
          {
            "id": "ff21b0ef-7dda-e4f8-d2c6-2ca03359d138",
            "text": "Fashion Brands",
            "hasChildren": true,
            "icon": "gi gi-ring",
            "formPage": {
              "FormName": "Fashion",
              "EntityName": "",
              "ImagePath": "gi gi-ring",
              "Caption": "Fashion Brands",
```

```
            "ViewName": "Fashion.json",
            "Visible": true
          },
          "navItem": {
            "ID": "navBar-Miscellaneous",
            "Caption": "Miscellaneous",
            "ImagePath": "gi gi-picture",
            "Visible": true
          },
          "items": []
        },
        {
          "id": "fd28b568-860d-033c-a0a9-59b79abd6378",
          "text": "Footballers Pagination",
          "hasChildren": true,
          "icon": "gi gi-soccer_ball",
          "formPage": {
            "FormName": "FootballersGridPaginationTest",
            "EntityName": "",
            "ImagePath": "gi gi-soccer_ball",
            "Caption": "Footballers Pagination",
            "ViewName": "FootballersGridPaginationTest.json",
            "Visible": true
          },
          "navItem": {
            "ID": "navBar-Miscellaneous",
            "Caption": "Miscellaneous",
            "ImagePath": "gi gi-picture",
            "Visible": true
          },
          "items": []
        },
        {
          "id": "97831c64-ebef-327b-ad0d-bda264885044",
          "text": "Music Tracks",
          "hasChildren": true,
          "icon": "gi gi-music",
          "formPage": {
            "FormName": "Music",
            "EntityName": "",
            "ImagePath": "gi gi-music",
            "Caption": "Music Tracks",
            "ViewName": "Music.json",
            "Visible": true
          },
          "navItem": {
            "ID": "navBar-Miscellaneous",
            "Caption": "Miscellaneous",
```

```
        "ImagePath": "gi gi-picture",
        "Visible": true
      },
      "items": []
    },
    {
      "id": "38a750ce-c446-32fe-eb2e-042f457ad2d4",
      "text": "Real Estate",
      "hasChildren": true,
      "icon": "gi gi-home",
      "formPage": {
        "FormName": "RealEstate",
        "EntityName": "",
        "ImagePath": "gi gi-home",
        "Caption": "Real Estate",
        "ViewName": "RealEstate.json",
        "Visible": true
      },
      "navItem": {
        "ID": "navBar-Miscellaneous",
        "Caption": "Miscellaneous",
        "ImagePath": "gi gi-picture",
        "Visible": true
      },
      "items": []
    },
    {
      "id": "8e27b9b6-f749-ea6d-b853-cf8228e0d4cf",
      "text": "Real Estate Property",
      "hasChildren": true,
      "icon": "gi gi-home",
      "formPage": {
        "FormName": "RealEstateProperty",
        "EntityName": "",
        "ImagePath": "gi gi-home",
        "Caption": "Real Estate Property",
        "ViewName": "RealEstateProperty.json",
        "Visible": true,
        "EntityForm": true
      },
      "navItem": {
        "ID": "navBar-Miscellaneous",
        "Caption": "Miscellaneous",
        "ImagePath": "gi gi-picture",
        "Visible": true
      },
      "items": []
    },
    {
```

```
      {
        "id": "68b42ab3-c837-cafb-e393-dc868548f55d",
        "text": "Wines",
        "hasChildren": true,
        "icon": "gi gi-glass",
        "formPage": {
          "FormName": "Wines",
          "EntityName": "",
          "ImagePath": "gi gi-glass",
          "Caption": "Wines",
          "ViewName": "Wines.json",
          "Visible": true
        },
        "navItem": {
          "ID": "navBar-Miscellaneous",
          "Caption": "Miscellaneous",
          "ImagePath": "gi gi-picture",
          "Visible": true
        },
        "items": []
      },
      {
        "id": "bcc76368-7b73-a59e-cb37-641983a8fe67",
        "text": "Selected Wine",
        "hasChildren": true,
        "icon": "gi gi-glass",
        "formPage": {
          "FormName": "Wine",
          "EntityName": "",
          "ImagePath": "gi gi-glass",
          "Caption": "Selected Wine",
          "ViewName": "Wine.json",
          "Visible": false,
          "EntityForm": true
        },
        "navItem": {
          "ID": "navBar-Miscellaneous",
          "Caption": "Miscellaneous",
          "ImagePath": "gi gi-picture",
          "Visible": true
        },
        "items": []
      }
    ]
  },
  {
    "id": "1a2b3c4d-5e6f-7890-abcd-ef1234567890",
    "text": "Administration",
    "expanded": false,
```

```
      "hasChildren": true,
      "icon": "gi gi-cogwheel",
      "navItem": {
        "ID": "navBar-SystemAdministration",
        "Caption": "Administration",
        "ImagePath": "gi gi-cogwheel",
        "Open": false,
        "Visible": true,
        "Locked": true
      },
      "items": [
        {
          "id": "3b4a5968-7e0f-1d2c-3b4a-596807ef1d2c",
          "text": "App Studio",
          "hasChildren": false,
          "icon": "gi gi-magic",
          "formPage": {
            "FormName": "AppStudioHome",
            "EntityName": "",
            "ImagePath": "gi gi-magic",
            "Caption": "App Studio",
            "ViewName": "AppStudioHome.html",
            "Visible": true,
            "DesignerOnly": true
          },
          "navItem": {
            "ID": "navBar-SystemAdministration",
            "Caption": "Administration",
            "ImagePath": "gi gi-cogwheel",
            "Open": false,
            "Visible": true,
            "Locked": true
          },
          "items": []
        },
        {
          "id": "e3f4a5b6-c7d8-9012-3456-789abcdef012",
          "text": "User Options",
          "hasChildren": false,
          "icon": "fa fa-user",
          "formPage": {
            "FormName": "UserOptions",
            "EntityName": null,
            "ImagePath": "fa fa-user",
            "Caption": "User Options",
            "ViewName": "UserOptions.html",
            "Visible": true,
            "Locked": true
```

```
            },
            "navItem": {
              "ID": "navBar-SystemAdministration",
              "Caption": "Administration",
              "ImagePath": "gi gi-cogwheel",
              "Open": false,
              "Visible": true,
              "Locked": true
            },
            "items": []
          }
        ]
      },
      {
        "id": "8c9d0e1f-2a3b-4c5d-6e7f-8a9b0c1d2e3f",
        "text": "Help",
        "expanded": false,
        "hasChildren": true,
        "icon": "gi gi-book",
        "navItem": {
          "ID": "navBar-Help",
          "Caption": "Help",
          "ImagePath": "gi gi-book",
          "Open": false,
          "Visible": true,
          "Locked": true
        },
        "items": [
          {
            "id": "5b6c7d8e-9f0a-1b2c-3d4e-5f6a7b8c9d0e",
            "text": "About",
            "hasChildren": false,
            "icon": "gi gi-circle_info",
            "formPage": {
              "FormName": "About",
              "EntityName": null,
              "ImagePath": "gi gi-circle_info",
              "Caption": "About",
              "ViewName": null,
              "Function": "TriSysApex.Forms.AboutTriSys",
              "Visible": true,
              "Locked": true
            },
            "navItem": {
              "ID": "navBar-Help",
              "Caption": "Help",
              "ImagePath": "gi gi-book",
              "Open": false,
```

```
            "Visible": true,
            "Locked": true
          },
          "items": []
        },
        {
          "id": "f2a3b4c5-d6e7-f8a9-b0c1-d2e3f4a5b6c7",
          "text": "Welcome",
          "hasChildren": false,
          "icon": "gi gi-home",
          "formPage": {
            "FormName": "Welcome",
            "EntityName": null,
            "ImagePath": "gi gi-home",
            "ViewName": "Welcome.html",
            "Caption": "Welcome",
            "Visible": false,
            "Locked": true,
            "RecordId": 0
          },
          "navItem": {
            "ID": "navBar-Help",
            "Caption": "Help",
            "ImagePath": "gi gi-book",
            "Open": false,
            "Visible": true,
            "Locked": true
          },
          "items": []
        },
        {
          "id": "a7b8c9d0-e1f2-a3b4-c5d6-e7f8a9b0c1d2",
          "text": "Quick Start",
          "hasChildren": false,
          "icon": "gi gi-circle_question_mark",
          "formPage": {
            "FormName": "QuickStart",
            "EntityName": null,
            "ImagePath": "gi gi-circle_question_mark",
            "Caption": "Quick Start",
            "ViewName": "QuickStart.html",
            "Visible": false,
            "Phone": false,
            "Locked": true
          },
          "navItem": {
            "ID": "navBar-Help",
            "Caption": "Help",
            "ImagePath": "gi gi-book"
```

```
      "ImagePath": "gi gi-book",
      "Open": false,
      "Visible": true,
      "Locked": true
    },
    "items": []
  },
  {
    "id": "3e4f5a6b-7c8d-9e0f-1a2b-3c4d5e6f7a8b",
    "text": "User Guide",
    "hasChildren": false,
    "icon": "fa fa-book",
    "formPage": {
      "FormName": "UserGuide",
      "EntityName": null,
      "ImagePath": "fa fa-book",
      "Caption": "User Guide",
      "ViewName": "UserGuide.html",
      "Visible": false,
      "Phone": false,
      "Locked": true
    },
    "navItem": {
      "ID": "navBar-Help",
      "Caption": "Help",
      "ImagePath": "gi gi-book",
      "Open": false,
      "Visible": true,
      "Locked": true
    },
    "items": []
  },
  {
    "id": "d9c8b7a6-5f4e-3d2c-1b0a-9f8e7d6c5b4a",
    "text": "Welcome Wizard",
    "hasChildren": false,
    "icon": "gi gi-magic",
    "formPage": {
      "FormName": "WelcomeWizard",
      "EntityName": null,
      "ImagePath": "gi gi-magic",
      "Caption": "Welcome Wizard",
      "ViewName": "WelcomeWizard.html",
      "Function": "TriSysApex.ModalDialogue.WelcomeWizard.Show",
      "Visible": false,
      "Locked": true
    },
    "navItem": {
      "ID": "navBar-Help",
```

```
              "Caption": "Help",
              "ImagePath": "gi gi-book",
              "Open": false,
              "Visible": true,
              "Locked": true
            },
            "items": []
          },
          {
            "id": "7f8a9b0c-1d2e-3f4a-5b6c-7d8e9f0a1b2c",
            "text": "Application",
            "hasChildren": false,
            "icon": "gi gi-circle_question_mark",
            "formPage": {
              "FormName": "Application",
              "EntityName": null,
              "ImagePath": "gi gi-circle_question_mark",
              "Caption": "Application",
              "ViewName": "ApplicationHelp.html",
              "Visible": false,
              "Locked": true
            },
            "navItem": {
              "ID": "navBar-Help",
              "Caption": "Help",
              "ImagePath": "gi gi-book",
              "Open": false,
              "Visible": true,
              "Locked": true
            },
            "items": []
          },
          {
            "id": "c2d3e4f5-a6b7-c8d9-e0f1-a2b3c4d5e6f7",
            "text": "Terms &amp; Conditions",
            "hasChildren": false,
            "icon": "gi gi-more_items",
            "formPage": {
              "FormName": "TermsAndConditions",
              "EntityName": null,
              "ImagePath": "gi gi-more_items",
              "Caption": "Terms &amp; Conditions",
              "ViewName": null,
              "Function":
"TriSysApex.ModalDialogue.TermsAndConditions.Show",
              "Visible": true,
              "Locked": true
            },
```

```
            "navItem": {
              "ID": "navBar-Help",
              "Caption": "Help",
              "ImagePath": "gi gi-book",
              "Open": false,
              "Visible": true,
              "Locked": true
            },
            "items": []
          },
          {
            "id": "8a9b0c1d-2e3f-4a5b-6c7d-8e9f0a1b2c3d",
            "text": "Lock",
            "hasChildren": false,
            "icon": "gi gi-lock",
            "formPage": {
              "FormName": "Lock",
              "EntityName": null,
              "ImagePath": "gi gi-lock",
              "Caption": "Lock",
              "ViewName": "Lock.html",
              "ShowWhenLoggedIn": true,
              "LoadRecord": "TriSysApex.Forms.Lock.Initialise",
              "Visible": false,
              "Locked": true
            },
            "navItem": {
              "ID": "navBar-Help",
              "Caption": "Help",
              "ImagePath": "gi gi-book",
              "Open": false,
              "Visible": true,
              "Locked": true
            },
            "items": []
          },
          {
            "id": "4c5d6e7f-8a9b-0c1d-2e3f-4a5b6c7d8e9f",
            "text": "Logoff",
            "hasChildren": false,
            "icon": "gi gi-exit",
            "formPage": {
              "FormName": "Logoff",
              "EntityName": null,
              "ImagePath": "gi gi-exit",
              "Caption": "Logoff",
              "ViewName": "Logoff.html",
              "Function":
```

"TriSysApex.LoginCredentials.LogoffWithPrompt",
              "Visible": false,
              "Locked": true
            },
            "navItem": {
              "ID": "navBar-Help",
              "Caption": "Help",
              "ImagePath": "gi gi-book",
              "Open": false,
              "Visible": true,
              "Locked": true
            },
            "items": []
          },
          {
            "id": "9f0e1d2c-3b4a-5968-7e0f-1d2c3b4a5968",
            "text": "Contact Us",
            "hasChildren": false,
            "icon": "fa fa-pencil-square-o",
            "formPage": {
              "FormName": "ContactUs",
              "EntityName": null,
              "ImagePath": "fa fa-pencil-square-o",
              "Caption": "Contact Us",
              "ViewName": "ContactUs.html",
              "ShowWhenLoggedIn": true,
              "Visible": true,
              "Locked": true
            },
            "navItem": {
              "ID": "navBar-Help",
              "Caption": "Help",
              "ImagePath": "gi gi-book",
              "Open": false,
              "Visible": true,
              "Locked": true
            },
            "items": []
          },
          {
            "id": "2a1b0c9d-8e7f-6a5b-4c3d-2e1f0a9b8c7d",
            "text": "Support Portal",
            "hasChildren": false,
            "icon": "fa fa-support",
            "formPage": {
              "FormName": "SupportPortal",
              "EntityName": null,
              "ImagePath": "fa fa-support",
              "Caption": "Support Portal"

Read App Studio Designed App Toolbar


```
                "Caption": "Support Portal",
                "ShowWhenLoggedIn": true,
                "Function": "TriSysApex.Forms.SupportPortal",
                "Visible": true,
                "Locked": true
              },
              "navItem": {
                "ID": "navBar-Help",
                "Caption": "Help",
                "ImagePath": "gi gi-book",
                "Open": false,
                "Visible": true,
                "Locked": true
              },
              "items": []
            }
          ]
        }
      ]
    }
  ]
```


Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Body

Name	Type	Description
`TreeFormat`	Boolean	`true`

Response

200

```
{
  "Success": true,
  "JSON": "base64 encoded string"
}
```

400

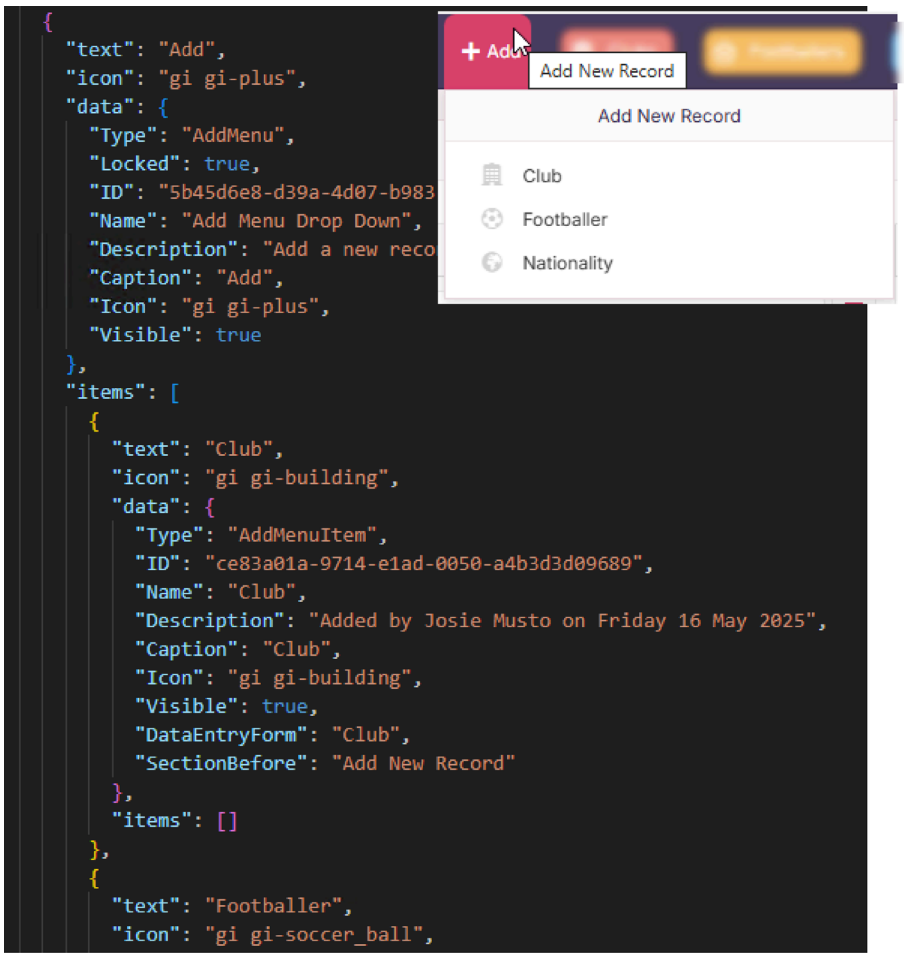
```
{
  "error": "Invalid request"
}
```

Base64 JSON Decoding

The JSON data is Base64 encoded. You must decode this into a JSON string and convert that into a multi-hierarchical object.

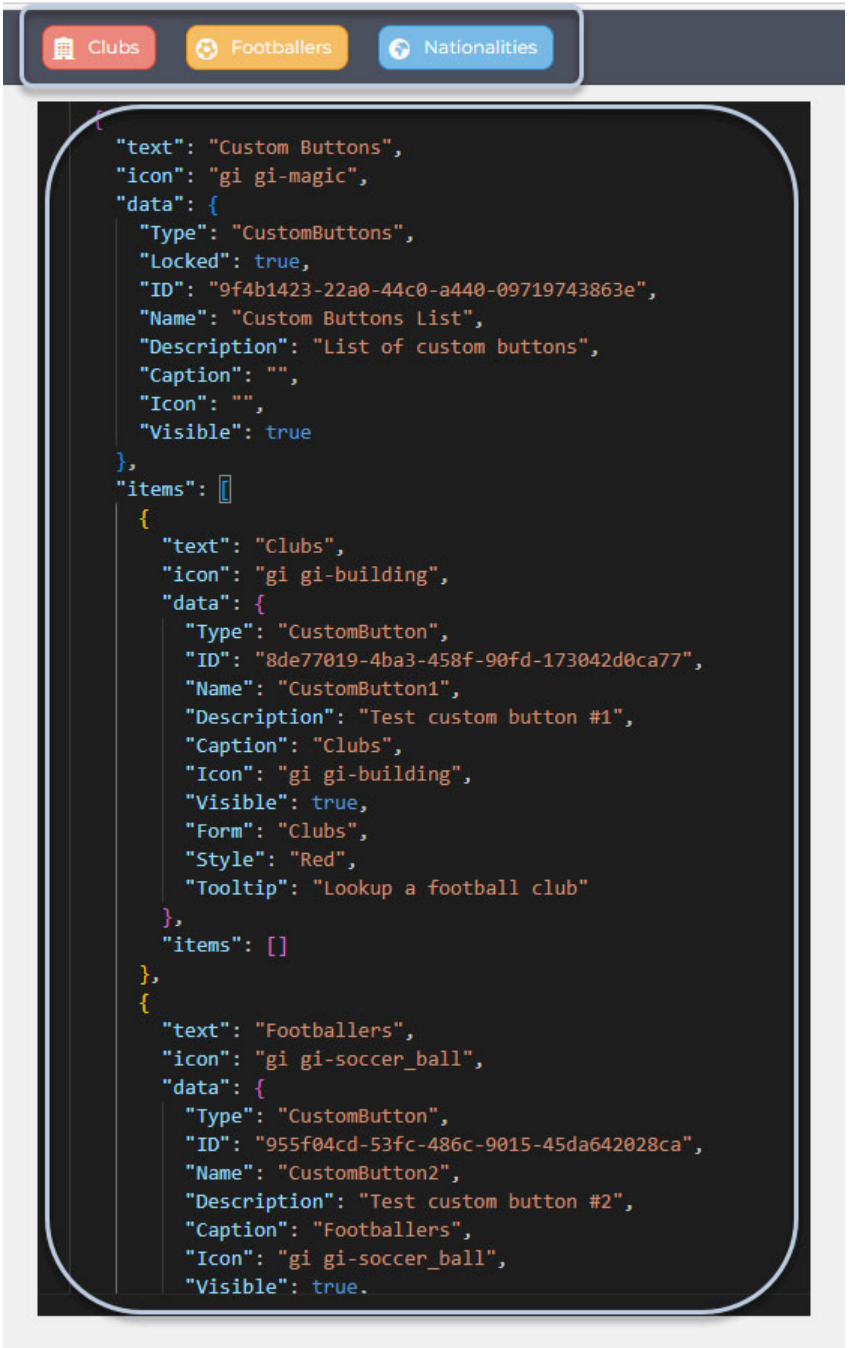
Add Menu

This example shows the add menu on the rendered app toolbar on the top and a matching portion of the multi-hierarchical object returned from the API:



Custom Buttons

This example shows the custom buttons on the rendered app toolbar on the top and a matching portion of the multi-hierarchical object returned from the API:




```
{
  "text": "Custom Buttons",
  "icon": "gi gi-magic",
  "data": {
    "Type": "CustomButtons",
    "Locked": true,
    "ID": "9f4b1423-22a0-44c0-a440-09719743863e",
    "Name": "Custom Buttons List",
    "Description": "List of custom buttons",
    "Caption": "",
    "Icon": "",
    "Visible": true
  },
  "items": [
    {
      "text": "Clubs",
      "icon": "gi gi-building",
      "data": {
        "Type": "CustomButton",
        "ID": "8de77019-4ba3-458f-90fd-173042d0ca77",
        "Name": "CustomButton1",
        "Description": "Test custom button #1",
        "Caption": "Clubs",
        "Icon": "gi gi-building",
        "Visible": true,
        "Form": "Clubs",
        "Style": "Red",
        "Tooltip": "Lookup a football club"
      },
      "items": []
    },
    {
      "text": "Footballers",
      "icon": "gi gi-soccer_ball",
      "data": {
        "Type": "CustomButton",
        "ID": "955f04cd-53fc-486c-9015-45da642028ca",
        "Name": "CustomButton2",
        "Description": "Test custom button #2",
        "Caption": "Footballers",
        "Icon": "gi gi-soccer_ball",
        "Visible": true,
```

Here is an example of a full app toolbar.

```
[
  {
    "text": "Navigation Bar",
    "icon": "fa fa-bars",
    "data": {
      "Type": "NavBar",
      "Locked": true,
      "ID": "727fcb55-1677-4947-81b5-d926278fa5d2",
      "Name": "Navigation Bar",
      "Description": "Toolbar header for the navigation bar",
      "Caption": "<strong>TriSys</strong> Flexiva",
      "Icon": "fa fa-bars",
      "Visible": true,
      "ProductLogoImage": "",
      "BrandName": "TriSys",
      "ProductName": "Flexiva",
      "NavBarBottomLogoImage": "images/nav-bar/trisys-flexiva-80hx190w-
25percent.png",
      "NavBarBottomLogoHeight": 80,
      "NavBarBottomLogoWidth": 190,
      "NavBarBottomLogoMargin": 5
    },
    "items": []
  },
  {
    "text": "Search...",
    "icon": "gi gi-search",
    "data": {
      "Type": "SearchBox",
      "Locked": true,
      "ID": "c7668bef-e618-4154-92af-10d81b20f712",
      "Name": "Search Box",
      "Description": "Search/lookup text box on toolbar",
      "Caption": "Search...",
      "Icon": "gi gi-search",
      "Visible": true,
      "DataSourceRequest": {
        "RequestId": "cffda37d-430a-b739-d5c1-80b2ffb276d2",
        "Type": "Read"
      }
    },
    "items": []
  },
  {
    "text": "History",
    "icon": "gi gi-more_windows",
    "data": {
      "Type": "HistoryMenu",
```

```
        "Locked": true,
        "ID": "40c12a13-1cdf-4066-a4f8-555ac045ef21",
        "Name": "History Menu",
        "Description": "Drop down menu of previously opened forms",
        "Caption": "History",
        "Icon": "gi gi-more_windows",
        "Visible": true
      },
      "items": []
    },
    {
      "text": "Add",
      "icon": "gi gi-plus",
      "data": {
        "Type": "AddMenu",
        "Locked": true,
        "ID": "5b45d6e8-d39a-4d07-b983-9f0888e84c1c",
        "Name": "Add Menu Drop Down",
        "Description": "Add a new record drop down menu",
        "Caption": "Add",
        "Icon": "gi gi-plus",
        "Visible": true
      },
      "items": [
        {
          "text": "Club",
          "icon": "gi gi-building",
          "data": {
            "Type": "AddMenuItem",
            "ID": "ce83a01a-9714-e1ad-0050-a4b3d3d09689",
            "Name": "Club",
            "Description": "Added by Josie Musto on Friday 16 May 2025",
            "Caption": "Club",
            "Icon": "gi gi-building",
            "Visible": true,
            "DataEntryForm": "Club",
            "SectionBefore": "Add New Record"
          },
          "items": []
        },
        {
          "text": "Footballer",
          "icon": "gi gi-soccer_ball",
          "data": {
            "Type": "AddMenuItem",
            "ID": "6f26c62d-9a5a-67a5-1b13-db2eb0ad0471",
            "Name": "Footballer",
            "Description": "Added by Josie Musto on Friday 16 May 2025",
```

```
          "Caption": "Footballer",
          "Icon": "gi gi-soccer_ball",
          "Visible": true,
          "DataEntryForm": "Footballer"
        },
        "items": []
      },
      {
        "text": "Nationality",
        "icon": "gi gi-globe_af",
        "data": {
          "Type": "AddMenuItem",
          "ID": "3cd5e876-5787-d011-97a7-77f978ac5171",
          "Name": "Nationality",
          "Description": "Added by Josie Musto on Friday 16 May 2025",
          "Caption": "Nationality",
          "Icon": "gi gi-globe_af",
          "Visible": true,
          "DataEntryForm": "Nationality"
        },
        "items": []
      }
    ]
  },
  {
    "text": "Custom Buttons",
    "icon": "gi gi-magic",
    "data": {
      "Type": "CustomButtons",
      "Locked": true,
      "ID": "9f4b1423-22a0-44c0-a440-09719743863e",
      "Name": "Custom Buttons List",
      "Description": "List of custom buttons",
      "Caption": "",
      "Icon": "",
      "Visible": true
    },
    "items": [
      {
        "text": "Clubs",
        "icon": "gi gi-building",
        "data": {
          "Type": "CustomButton",
          "ID": "8de77019-4ba3-458f-90fd-173042d0ca77",
          "Name": "CustomButton1",
          "Description": "Test custom button #1",
          "Caption": "Clubs",
          "Icon": "gi gi-building",
          "Visible": true
```

```
        "Visible": true,
        "Form": "Clubs",
        "Style": "Red",
        "Tooltip": "Lookup a football club"
      },
      "items": []
    },
    {
      "text": "Footballers",
      "icon": "gi gi-soccer_ball",
      "data": {
        "Type": "CustomButton",
        "ID": "955f04cd-53fc-486c-9015-45da642028ca",
        "Name": "CustomButton2",
        "Description": "Test custom button #2",
        "Caption": "Footballers",
        "Icon": "gi gi-soccer_ball",
        "Visible": true,
        "Form": "Footballers",
        "Style": "Yellow",
        "Tooltip": "Lookup a footballer"
      },
      "items": []
    },
    {
      "text": "Nationalities",
      "icon": "gi gi-globe_af",
      "data": {
        "Type": "CustomButton",
        "ID": "83cbb1d8-5538-b38a-d97f-e41f689f2fc9",
        "Name": "Nationalities",
        "Description": "Added by Josie Musto on Monday 19 May 2025",
        "Caption": "Nationalities",
        "Icon": "gi gi-globe_af",
        "Visible": true,
        "Form": "Nationalities",
        "Style": "Blue",
        "Tooltip": "Lookup a nationality"
      },
      "items": []
    },
    {
      "text": "Cats",
      "icon": "gi gi-certificate",
      "data": {
        "Type": "CustomButton",
        "ID": "888cc48a-6054-c3ad-7733-a13fefd6c3f1",
        "Name": "Yaay haway",
        "Description": "Added by Josie Musto on Friday 16 May 2025",
```

```
          "Caption": "Cats",
          "Icon": "gi gi-certificate",
          "Visible": false,
          "Form": "Cats",
          "Style": "Green",
          "Tooltip": "List all types of cat"
        },
        "items": []
      },
      {
        "text": "Dogs",
        "icon": "gi gi-dog",
        "data": {
          "Type": "CustomButton",
          "ID": "e3805263-adc5-cb24-aad0-e3c9be331572",
          "Name": "Dogs",
          "Description": "Added by Josie Musto on Monday 19 May 2025",
          "Caption": "Dogs",
          "Icon": "gi gi-dog",
          "Visible": false,
          "Form": "Dogs",
          "Style": "Grey",
          "Tooltip": "List all dog breeds"
        },
        "items": []
      }
    ]
  },
  {
    "text": "Show Help",
    "icon": "gi gi-circle_question_mark",
    "data": {
      "Type": "ShowHelp",
      "Locked": true,
      "ID": "b15c0622-0cdc-4e7c-9d22-4554f927fb98",
      "Name": "Show Help Button",
      "Description": "Show Help Button in Primary theme colour",
      "Caption": "Show Help",
      "Icon": "gi gi-circle_question_mark",
      "Visible": true,
      "Style": "Green",
      "Tooltip": "Show popup help about the currently displayed form"
    },
    "items": []
  },
  {
    "text": "Account Summary",
    "icon": "gi gi-user",
```

```
    "data": {
      "Type": "AccountSummary",
      "Locked": true,
      "ID": "043c7522-e6a5-4de6-a957-37bf8fe2ecf8",
      "Name": "Account Summary drop down menu",
      "Description": "Items in drop down menu for logged in subscriber",
      "Caption": "",
      "Icon": "gi gi-play_button",
      "Visible": true
    },
    "items": [
      {
        "text": "Subscriber Name",
        "icon": "gi gi-user",
        "data": {
          "Type": "AccountSummaryItem",
          "Locked": true,
          "ID": "9c78ddca-ce25-46d4-b098-1c749d5d03f3",
          "Name": "Account Summary Subscriber Name",
          "Description": "Show logged-in subscriber name",
          "Caption": "",
          "Icon": "gi gi-play_button",
          "Visible": true
        },
        "items": []
      },
      {
        "text": "Subscriber Company",
        "icon": "gi gi-building",
        "data": {
          "Type": "AccountSummaryItem",
          "Locked": true,
          "ID": "d27434e0-2a5d-4b2d-a3f5-79a63e4e9182",
          "Name": "Account Summary Subscriber Company",
          "Description": "Show logged-in subscriber company",
          "Caption": "",
          "Icon": "gi gi-building",
          "Visible": true
        },
        "items": []
      },
      {
        "text": "Profile/User Settings",
        "icon": "gi gi-nameplate",
        "data": {
          "Type": "AccountSummaryItem",
          "Locked": true,
          "ID": "6aca51f3-c2cb-41f3-ae35-005550b901ed",
```

```
            "Name": "Profile/User Settings",
            "Description": "Show subscriber settings",
            "Caption": "Profile/User Settings",
            "Icon": "gi gi-nameplate",
            "Visible": true
          },
          "items": []
        },
        {
          "text": "Lock Account",
          "icon": "fa fa-lock",
          "data": {
            "Type": "AccountSummaryItem",
            "Locked": true,
            "ID": "4df40642-ed9b-4743-9fef-d89091fd036f",
            "Name": "Lock Account",
            "Description": "Lock application",
            "Caption": "Lock Account",
            "Icon": "fa fa-lock",
            "Visible": true
          },
          "items": []
        },
        {
          "text": "Logout",
          "icon": "fa fa-ban",
          "data": {
            "Type": "AccountSummaryItem",
            "Locked": true,
            "ID": "1728cf85-48eb-473c-80e5-ef7c7bb7338a",
            "Name": "Logout",
            "Caption": "Logout",
            "Icon": "fa fa-ban",
            "Visible": true
          },
          "items": []
        },
        {
          "text": "Activity Metrics",
          "icon": "gi gi-charts",
          "data": {
            "ID": "c2fa5506-38aa-4d9a-bbf1-92b2e0935149",
            "Name": "Activity Metrics",
            "Description": "Activity metrics shown",
            "Caption": "",
            "Icon": "gi gi-charts"
```

Read App Studio Designed Activity Metrics

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Name	Type	Description
`TreeFormat`	Boolean	`true`

```
            "icon": "gi gi-charts",
            "Visible": true
          },
          "items": []
        }
      ]
```

200

```
{
  "Success": true,
  "JSON": "base64 encoded string"
}
```

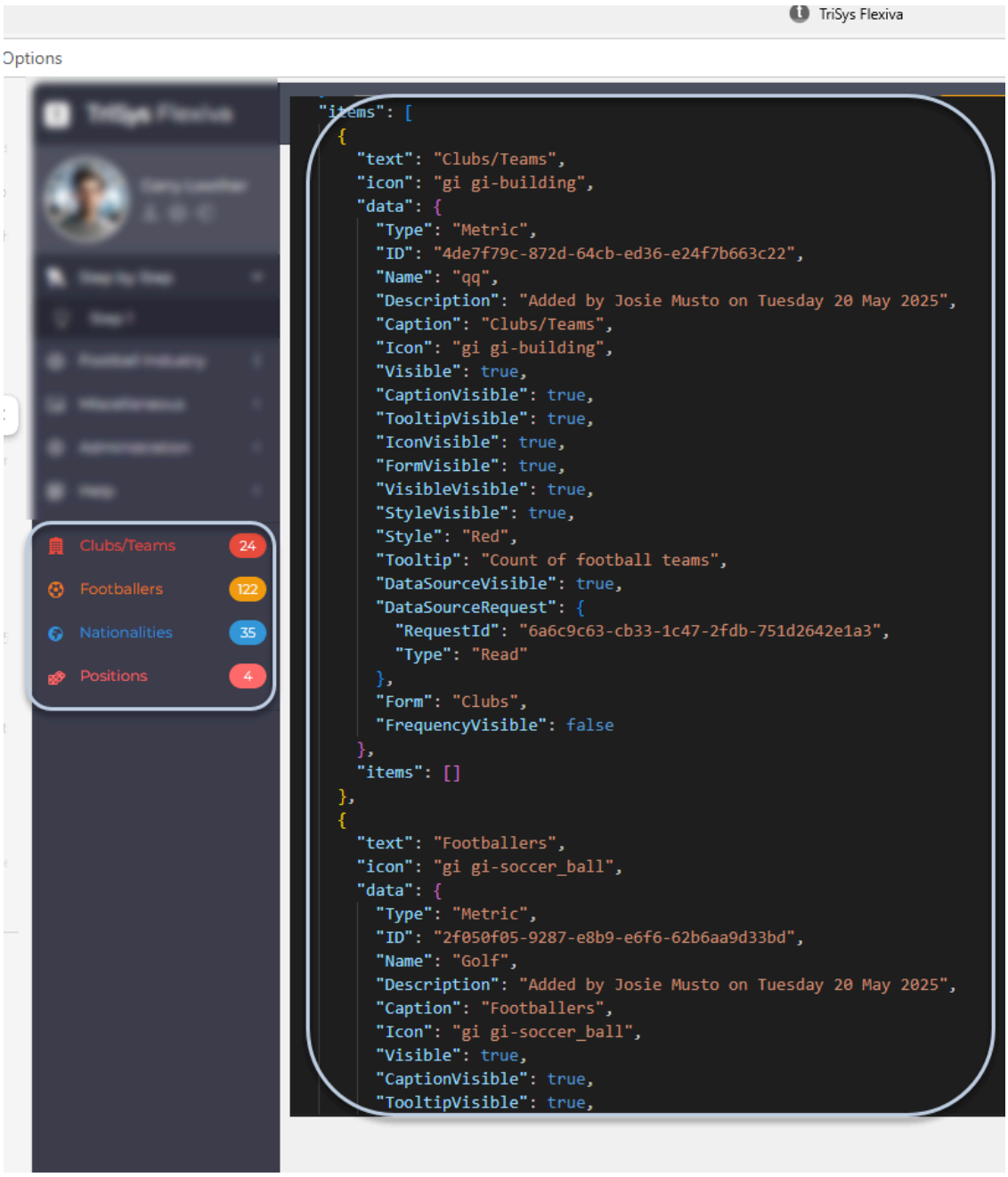
400

```
{
  "error": "Invalid request"
}
```

Base64 JSON Decoding

The JSON data is Base64 encoded. You must decode this into a JSON string and convert that into a multi-hierarchical object.

This example shows the activity metric KPI's on the rendered app on the left and a matching portion of the multi-hierarchical object returned from the API.



Scheduling Frequency

How often the ReST API requests should be called to refresh the display of each KPI is determined by these properties.

```
"FrequencyVisible": true,
"Frequency": "30 Seconds
```

In this example, we call this API every 30 seconds for each visible metric using the RequestId of the ReST API data source request:

```
"DataSourceRequest": {
            "RequestId": "6a6c9c63-cb33-1c47-2fdb-751d2642e1a3",
            "Type": "Read"
}
```

The data returned for metrics is usually simply a number, for example:

```
{
  "DataTable": {
    "List": [
      {
        "count": 24
      }
    ]
  }
}
```

.. and this specific metric would be displayed like this:

Example Response JSON

Here is an example of the full activity metrics.

```
[
  {
    "text": "Activity Metrics",
    "icon": "gi gi-charts",
    "data": {
      "Type": "Root",
      "Locked": true,
      "ID": "3bc36fcb-f506-4ba7-99b4-bf278b59e73b",
      "Name": "Activity Metrics",
      "Description": "List of activity metrics available on both
navigation bar and toolbar account summary drop down menu",
      "Caption": "Activity Metrics",
      "Icon": "gi gi-charts",
      "Visible": true,
      "CaptionVisible": true,
      "TooltipVisible": false,
      "IconVisible": false,
      "FormVisible": false,
      "VisibleVisible": true,
      "StyleVisible": false,
      "DataSourceVisible": false,
      "FrequencyVisible": true,
      "Frequency": "30 Seconds"
    },
    "items": [
      {
        "text": "Clubs/Teams",
        "icon": "gi gi-building",
        "data": {
          "Type": "Metric",
          "ID": "4de7f79c-872d-64cb-ed36-e24f7b663c22",
          "Name": "qq",
          "Description": "Added by Josie Musto on Tuesday 20 May 2025",
          "Caption": "Clubs/Teams",
          "Icon": "gi gi-building",
          "Visible": true,
          "CaptionVisible": true,
          "TooltipVisible": true,
          "IconVisible": true,
          "FormVisible": true,
          "VisibleVisible": true,
          "StyleVisible": true,
          "Style": "Red",
          "Tooltip": "Count of football teams",
          "DataSourceVisible": true,
          "DataSourceRequest": {
            "RequestId": "6a6c9c63-cb33-1c47-2fdb-751d2642e1a3",
            "Type": "Read"
```

```
            },
            "Form": "Clubs",
            "FrequencyVisible": false
          },
          "items": []
        },
        {
          "text": "Footballers",
          "icon": "gi gi-soccer_ball",
          "data": {
            "Type": "Metric",
            "ID": "2f050f05-9287-e8b9-e6f6-62b6aa9d33bd",
            "Name": "Golf",
            "Description": "Added by Josie Musto on Tuesday 20 May 2025",
            "Caption": "Footballers",
            "Icon": "gi gi-soccer_ball",
            "Visible": true,
            "CaptionVisible": true,
            "TooltipVisible": true,
            "IconVisible": true,
            "FormVisible": true,
            "VisibleVisible": true,
            "StyleVisible": true,
            "Style": "Yellow",
            "Tooltip": "Number of football players",
            "DataSourceVisible": true,
            "DataSourceRequest": {
              "RequestId": "3fc781bc-0035-3716-4db3-48e415596cb1",
              "Type": "Read"
            },
            "Form": "Footballers",
            "FrequencyVisible": false
          },
          "items": []
        },
        {
          "text": "Nationalities",
          "icon": "gi gi-globe_af",
          "data": {
            "Type": "Metric",
            "ID": "24e8c094-1253-ff02-0684-2260dfb82196",
            "Name": "goo goo",
            "Description": "Added by Josie Musto on Tuesday 20 May 2025",
            "Caption": "Nationalities",
            "Icon": "gi gi-globe_af",
            "Visible": true,
            "CaptionVisible": true,
            "TooltipVisible": true,
```

Read App Studio Form History

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Name	Type	Description
`UserId`	Integer	The logged in user ID in their database. See this API.


```
            "IconVisible": true,
            "FormVisible": true,
            "VisibleVisible": true,
            "StyleVisible": true,
            "Style": "Blue",
            "Tooltip": "Number of nationalities",
            "Form": "Nationalities",
            "DataSourceVisible": true,
            "FrequencyVisible": false,
            "DataSourceRequest": {
              "RequestId": "28120411-73f2-b0a4-2117-1cf5f1c28546",
              "Type": "Read"
            }
          },
          "items": []
        },
        {
          "text": "Positions",
          "icon": "gi gi-playing_dices",
          "data": {
            "Type": "Metric",
            "ID": "2d25ff20-efec-4d48-053c-b58ffe5a7429",
            "Name": "Loopy",
            "Caption": "Positions",
            "Icon": "gi gi-playing_dice
            "Visible": true,
            "CaptionVisible": true,
            "TooltipVisible": true,
            "IconVisible": true,
            "FormVisible": true,
            "VisibleVisible": true,
            "StyleVisible": true,
            "Style": "Themed",
            "DataSourceVisible": true,
            "FrequencyVisible": false,
            "DataSourceRequest": {
              "RequestId": "7dab991d-e5b2-1972-e33e-62f632f2f668",
              "Type": "Read"
            }
          },
          "items": []
        }
      ]
    }
  ]
```

200

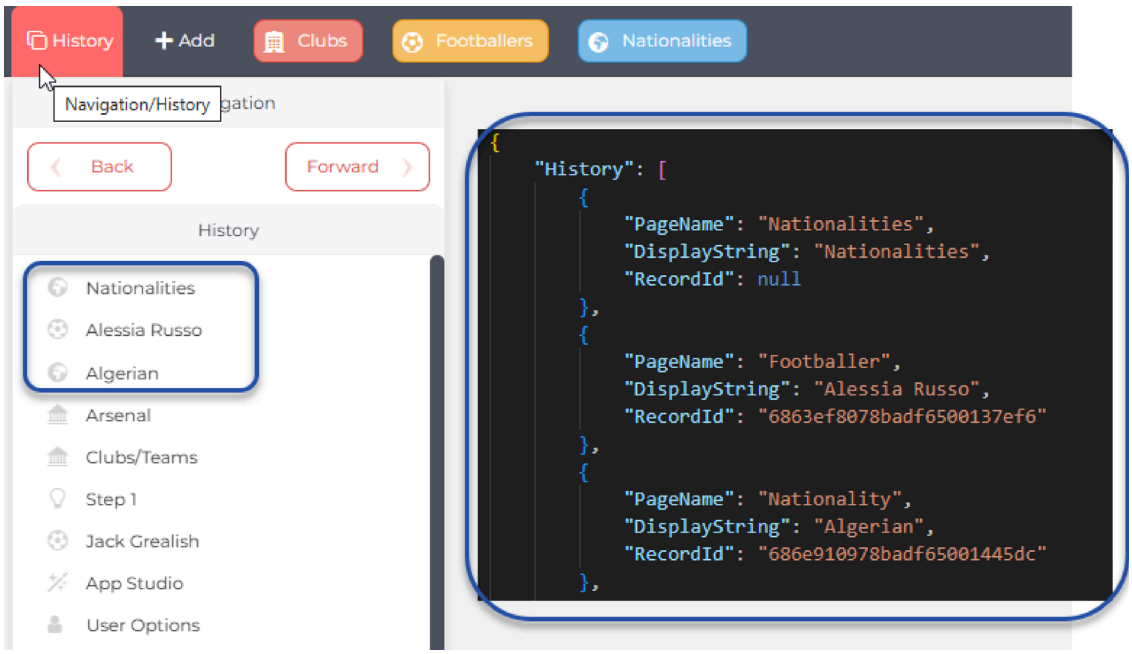
```
{
  "Success": true,
  "History": [
    {
      "PageName": "Footballer",
      "DisplayString": "Alessia Russo",
      "RecordId": "6863ef8078badf6500137ef6"
    }
  ]
}
```

400

```
{
  "error": "Invalid request"
}
```

UI Display

This example shows the rendered history menu using the data returned from the API.



Menu Display Algorithm

Combine the history menu list with the list of pages in the navigation bar. This allows you to determine which icon to display in the history menu, and indeed which form and record to open when the user clicks on each history menu item.

Example Response JSON

Here is an example of a full form history:

```
{
    "History": [
        {
            "PageName": "Nationalities",
            "DisplayString": "Nationalities",
            "RecordId": null
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Alessia Russo",
            "RecordId": "6863ef8078badf6500137ef6"
        },
        {
            "PageName": "Nationality",
            "DisplayString": "Algerian",
            "RecordId": "686e910978badf65001445dc"
        },
        {
            "PageName": "Club",
            "DisplayString": "Arsenal",
            "RecordId": "674d85df050c58540003f7d4"
        },
        {
            "PageName": "Clubs",
            "DisplayString": "Clubs/Teams",
            "RecordId": null
        },
        {
            "PageName": "AANew",
            "DisplayString": "Step 1",
            "RecordId": null
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Jack Grealish",
            "RecordId": "674cb1cb050c58540003e20f"
        },
        {
            "PageName": "AppStudioHome",
            "DisplayString": "App Studio",
            "RecordId": null
        },
        {
            "PageName": "UserOptions",
            "DisplayString": "User Options",
            "RecordId": null
        },
        {
```

```
            "PageName": "Footballers",
            "DisplayString": "Footballers",
            "RecordId": null
        },
        {
            "PageName": "FootballersGridPaginationTest",
            "DisplayString": "Footballers Pagination",
            "RecordId": null
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Harry Kane",
            "RecordId": "674cb1cb050c58540003e206"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Son Heung-min",
            "RecordId": "674cb1cb050c58540003e20e"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Alisson Becker",
            "RecordId": "674cb1cb050c58540003e217"
        },
        {
            "PageName": "Nationality",
            "DisplayString": "Brazilian",
            "RecordId": "67a601f84779de1b00027afc"
        },
        {
            "PageName": "Club",
            "DisplayString": "Newcastle United",
            "RecordId": "674cb652050c58540003e244"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Bruno Guimarães",
            "RecordId": "674cb1cb050c58540003e213"
        },
        {
            "PageName": "ContactUs",
            "DisplayString": "Contact Us",
            "RecordId": null
        },
        {
            "PageName": "Club",
            "DisplayString": "Sunderland",
            "RecordId": "67f8e47778badf65000521b7"
```

```
        },
        {
            "PageName": "Club",
            "DisplayString": "Bayern Munich",
            "RecordId": "681b3c7878badf65000b29c5"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Rico Henry",
            "RecordId": "686f8f5f78badf6500145993"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Joachim Christian Andersen",
            "RecordId": "6863f63578badf6500138016"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Mathias Jensen",
            "RecordId": "686f8f2478badf6500145986"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Alexander Iwobi",
            "RecordId": "6863f7fc78badf650013805e"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Leo Hjelde",
            "RecordId": "686f907d78badf65001459aa"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Dennis Cirkin",
            "RecordId": "6863ecbf78badf6500137e73"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Jayden Bogle",
            "RecordId": "686f919878badf65001459cd"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Pascal Struijk",
            "RecordId": "686f9e5778badf6500145a7f"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Ethan Ampadu"
```


Write App Studio Form History

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

```
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Karl Da
            "RecordId": "686f915878badf65001459c2"
        },
        {
            "PageName": "Footballer",
            "DisplayString": "Patrick Roberts",
            "RecordId": "686f902278badf65001459a0"
```

Name	Type	Description
`UserId`	Integer	The logged in user ID in their database. See this API.

```
            "PageName": "Footballer",
            "DisplayString": "Footballer",
            "RecordId": null
        }
    ],
    "Success": true,
    "ErrorMessage": null
}
```

Item	JSON Object	

```
{
"DisplayString": "Port Rush Golf Club",
    "PageName": "GolfClub",
    "RecordId": "abc123-port-rush"
}
```

Response Format/Sample

200

```
{
  "Success": true
}
```

400

```
{
  "error": "Invalid request"
}
```

Read Logged In User

After login, the user account of the authenticated subscriber can be requested.

This is usually called when the login authentication API has been successful to return the user account.

Read App Studio Logged in User

`POST` `AppStudio/ReadUserAccountList`

This is a POST method which passes both the API and session keys as headers, and a body. If successful, the app toolbar can be displayed.

Headers

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Body

Name	Type	Description
`ContactId`	Integer	The logged in subscriber contact id returned in this API call.
`Enabled`	Boolean	`true`

Response

200


```
{
  "Success": true,
  "UserAcccounts": [
        {
            "ContactId": 123456,
            "Name": "Fred Bloggs",
            "EMail": "fred.b99@domain.com",
            "Password": "************",
            "Photo":
"https://api.trisys.co.uk/FileViewer/20250716_174331_5a2f8734-78e0-
4b26-bfbc-7d0c261f4dd6/dall-e.png",
            "JobTitle": "Software Engineer",
            "Type": "Designer",
            "Enabled": true,
            "CreatedDate": "2025-06-13T12:30:13",
            "CustomVariables": null,
            "ASPLogin": "fred.bloggs_98765abc",
            "UserId": 987654
        }
    ]
}
```


400


```
{
  "error": "Invalid request"
}
```

Read Custom Variables

The list of all client-side custom variables can be requested.

This is required to assign client-side variable values when engaging with server-side ReST API requests.

Read App Studio Custom Variables

`POST` `AppStudio/ReadCustomVariables`

This is a POST method which passes both the API and session keys as headers, and a body. If successful, the custom variables can be stored.

Headers

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Body

Name	Type	Description
`TreeFormat`	Boolean	`true`

Response

200

```
{
  "Success": true,
  "JSON": "base64 encoded string"
}
```

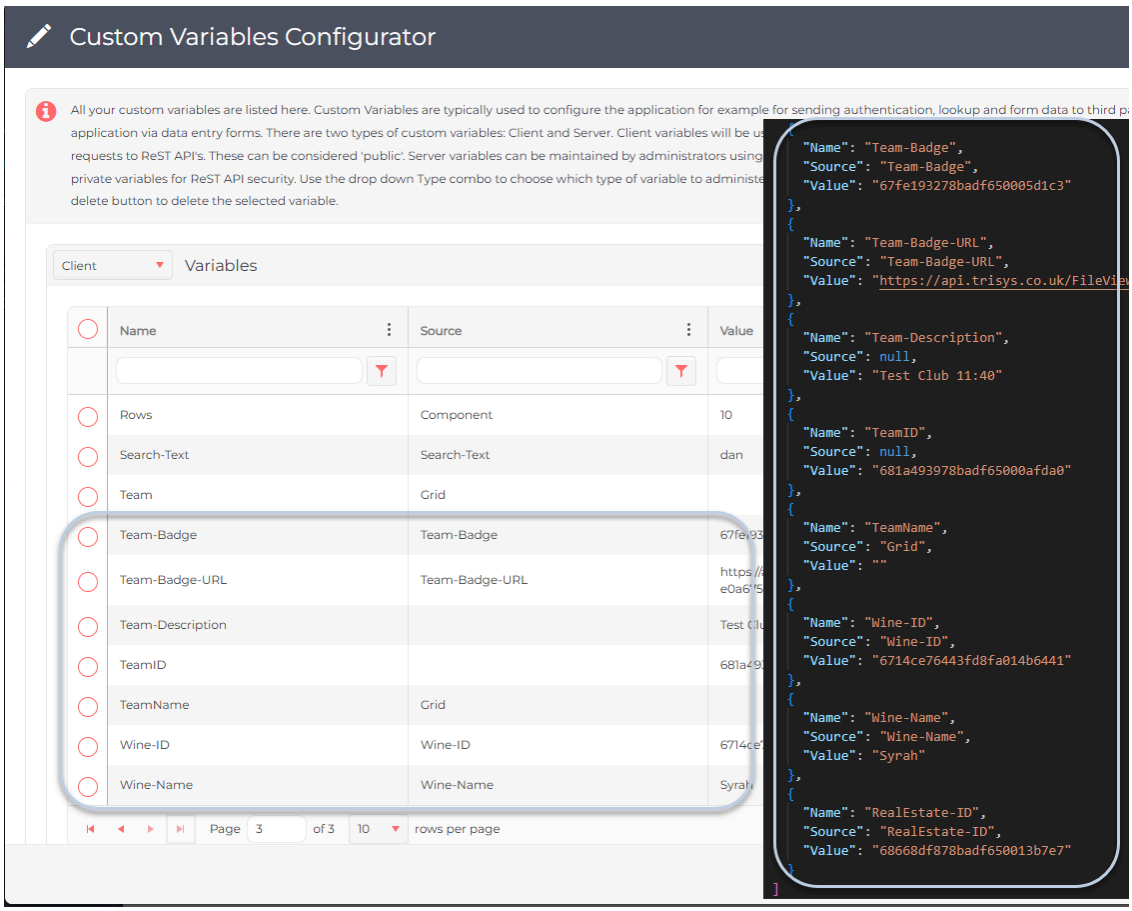
400

```
{
  "error": "Invalid request"
}
```

Base64 JSON Decoding

The JSON data is Base64 encoded. You must decode this into a JSON string and convert that into a multi-hierarchical object.

This example shows the custom variables in the App Studio configurator on the left and a matching portion of the multi-hierarchical object returned from the API.



Here is an example of a client-side list of custom variables.

```
{
  "Client": [
    {
      "Name": "BetsyPalmerContactID",
      "Source": "Betsy Palmer ContactID",
      "Value": "49346"
    },
    {
      "Name": "BookID",
      "Source": "BookID",
      "Value": "2"
    },
    {
      "Name": "CapitalCity",
      "Source": "CapitalCity",
      "Value": "Capo Seet"
    },
    {
      "Name": "Country",
      "Source": "Country",
      "Value": "Foo Kinhel"
    },
    {
      "Name": "Country_Code",
      "Source": "Country_Code",
      "Value": "FK"
    },
    {
      "Name": "DataSource-Sort",
      "Source": "DataSource-Sort",
      "Value": ""
    },
    {
      "Name": "Footballer-Biography",
      "Source": "Footballer-Biography",
      "Value": "Oh what a lonely boy"
    },
    {
      "Name": "Footballer-ID",
      "Source": "Footballer-ID",
      "Value": ""
    },
    {
      "Name": "Footballer-Name",
      "Source": "Footballer-Name",
      "Value": "David X-Ray Custom Variable"
    },
    {
```

```
      "Name": "Footballer-PhotoURL",
      "Source": "Footballer-PhotoURL",
      "Value":
"https://resources.premierleague.com/premierleague/photos/players/110x14
0/p154561.png"
    },
    {
      "Name": "Footballer-Position",
      "Source": "Footballer-Position",
      "Value": ""
    },
    {
      "Name": "Footballer-PositionID",
      "Source": "Footballer-PositionID",
      "Value": "6756e62f050c585400050d8b"
    },
    {
      "Name": "Footballer-SquadNumber",
      "Source": "Footballer-SquadNumber",
      "Value": ""
    },
    {
      "Name": "Footballer-TeamID",
      "Source": "Footballer-TeamID",
      "Value": "674d85df050c58540003f7d4"
    },
    {
      "Name": "MovieID",
      "Source": "MovieID",
      "Value": "2"
    },
    {
      "Name": "Nationality-ID",
      "Source": "Nationality-ID",
      "Value": ""
    },
    {
      "Name": "NationalityName",
      "Source": "Search Criteria",
      "Value": "Which nation player represents"
    },
    {
      "Name": "Page",
      "Source": "Component",
      "Value": "1"
    },
    {
      "Name": "Population",
```

```
        "Source": "Population",
        "Value": "747"
      },
      {
        "Name": "Rows",
        "Source": "Component",
        "Value": "10"
      },
      {
        "Name": "Search-Text",
        "Source": "Search-Text",
        "Value": "dan"
      },
      {
        "Name": "Team",
        "Source": "Grid",
        "Value": ""
      },
      {
        "Name": "Team-Badge",
        "Source": "Team-Badge",
        "Value": "67fe193278badf650005d1c3"
      },
      {
        "Name": "Team-Badge-URL",
        "Source": "Team-Badge-URL",
        "Value":
 "https://api.trisys.co.uk/FileViewer//Josie.Musto/20250506_151741_7dc429
 35-75e0-4d86-995b-e0a675cffa4"
      },
      {
        "Name": "Team-Description",
        "Source": null,
        "Value": "Test Club 11:40"
      },
      {
        "Source": null,
        "Value": "681a493978badf65000af
      },
      {
        "Name": "TeamName",
        "Source": "Grid",
        "Value": ""
      },
      {
        "Name": "Wine-ID",
        "Source": "Wine-ID",
        "Value": "6714ce76443fd8fa014b6441"
```


Read App Studio Custom Data Sources

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Name	Type	Description
`TreeFormat`	Boolean	`true`

200

```
{
  me": "Wine-Name",
  "Source": "Wine-Name",
  "Value": "Syrah"
},
{
  "Name": "RealEstate-ID",
  "Source": "RealEstate-ID",
  "Value": "68668df878badf650013b7e7"
```

```
{
  "Success": true,
  "JSON": "base64 encoded string"
}
```

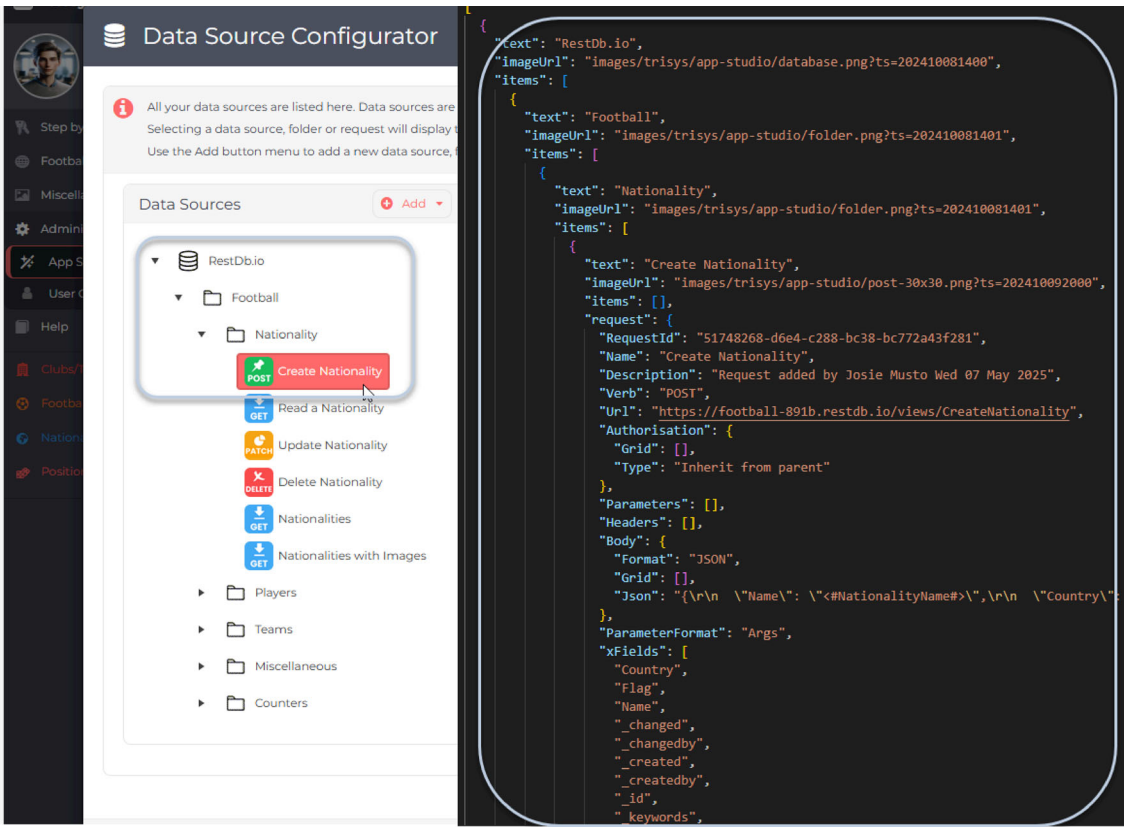
400

```
{
  "error": "Invalid request"
}
```

Base64 JSON Decoding

The JSON data is Base64 encoded. You must decode this into a JSON string and convert that into a multi-hierarchical object.

This example shows the custom variables in the App Studio configurator on the left and a matching portion of the multi-hierarchical object returned from the API.



Here is an example of a partial list of all data sources. The full list can be very large so it is not supplied here.

```
[
  {
    "text": "RestDb.io",
    "imageUrl": "images/trisys/app-studio/database.png?ts=202410081400",
    "items": [
      {
        "text": "Football",
        "imageUrl": "images/trisys/app-studio/folder.png?
ts=202410081401",
        "items": [
          {
            "text": "Nationality",
            "imageUrl": "images/trisys/app-studio/folder.png?
ts=202410081401",
            "items": [
              {
                "text": "Create Nationality",
                "imageUrl": "images/trisys/app-studio/post-30x30.png?
ts=202410092000",
                "items": [],
                "request": {
                  "RequestId": "51748268-d6e4-c288-bc38-bc772a43f281",
                  "Name": "Create Nationality",
                  "Description": "Request added by Josie Musto Wed 07
May 2025",
                  "Verb": "POST",
                  "Url": "https://football-
891b.restdb.io/views/CreateNationality",
                  "Authorisation": {
                    "Grid": [],
                    "Type": "Inherit from parent"
                  },
                  "Parameters": [],
                  "Headers": [],
                  "Body": {
                    "Format": "JSON",
                    "Grid": [],
                    "Json": "{\r\n  \"Name\": \"
<#NationalityName#>\",\r\n  \"Country\": \"<#Country#>\",\r\n
\"FlagURL\": \"<#Nationality-Flag#>\"\r\n}"
                  },
                  "ParameterFormat": "Args",
                  "Fields": [
                    {
                      "Field": "_id",
                      "Type": "String",
                      "Values": "1 values: 681b90a278badf65000b3941",
                      "Visible": true,
```

```
                        "Caption": "_id",
                        "Key": true,
                        "KeyVariable": "Nationality-ID"
                      },
                      {
                        "Field": "Country",
                        "Type": "String",
                        "Values": "1 values: Foo Kinhel",
                        "Visible": true,
                        "Caption": "Country"
                      },
                      {
                        "Field": "Flag",
                        "Type": "Image ID",
                        "Values": "1 values: 681b90a178badf65000b3940",
                        "Visible": true,
                        "Caption": "Flag"
                      },
                      {
                        "Field": "message",
                        "Type": "String",
                        "Values": "1 values: Nationality created
successfully",
                        "Visible": true,
                        "Caption": "message"
                      },
                      {
                        "Field": "Name",
                        "Type": "String",
                        "Values": "1 values: AA Which nation player
represents",
                        "Visible": true,
                        "Caption": "Name"
                      },
                      {
                        "Field": "nationality",
                        "Type": "String",
                        "Values": "1 values:
{\"_id\":\"681b90a278badf65000b3941\",\"Country\":\"Foo
Kinhel\",\"Flag\":[\"681b90a178badf65000b3940\"],\"Name\":\"AA Which
nation p...",
                        "Visible": true,
                        "Caption": "Nationality"
                      },
                      {
                        "Field": "success",
                        "Type": "Boolean",
                        "Values": "1 values: true",
```

```
                "Visible": true,
                "Caption": "success"
              }
            ]
          },
          "attr": {
            "RequestId": "51748268-d6e4-c288-bc38-bc772a43f281"
          }
        },
        {
          "text": "Read a Nationality",
          "imageUrl": "images/trisys/app-studio/get-30x30.png?
ts=202410092000",
          "items": [],
          "request": {
            "RequestId": "2cd08f6d-6472-f1f2-d84f-b3218f057066",
            "Name": "Read a Nationality",
            "Description": "Request added by Josie Musto on Sunday
01 December 2024",
            "Verb": "GET",
            "Url": "https://restdb.trisys.co.uk/ReadNationality?
ID=<#Nationality-ID#>",
            "Authorisation": {
              "Grid": [],
              "Type": "Inherit from parent"
            },
            "Parameters": [],
            "Headers": [],
            "Body": {
              "Format": "JSON",
              "Grid": [],
              "Json": "{}"
            },
            "ParameterFormat": "Args",
            "Fields": [
              {
                "Field": "ID",
                "Type": "String",
                "Values": "1 values: 67f8e87578badf6500052231",
                "Visible": true,
                "Caption": "ID",
                "Key": true,
                "KeyVariable": "Nationality-ID"
              }
              {
                "Field": "Nationality",
                "Type": "String",
                "Values": "1 values: German",
                "Visible": true
```


Read Custom Form

```
                    "Visible": true,
                    "Caption": "Nationality"
                  },
                  {
                    "Field": "Country",
                    "Type": "String",
                    "Values": "1 values: Germany",
                    "Visible": true,
```

Name	Value
Content-Type	application/json
SiteKey	API Key
DataServicesKey	Session Key

```
                  {
                    "Field": "FlagUrl",
                    "Type": "Image URL",
                    "Values": "1 values: https://football-
891b.restdb.io/media/67f8e87478badf6500052230",
                    "Visible": true,
                    "Caption": "Flag Url"
                  },
                  {
                    "Field": "Flag",
                    "Type": "Image ID",
                    "Values": "1 values:
[\"67f8e87478badf6500052230\"]",
                    "Visible": true,
                    "Caption": "Flag"
                  }
                ]
              },
              "attr": {
                "RequestId": "2cd08f6d-6472-f1f2-d84f-b3218f057066"
              }
            },
            ....
```

200

```
{
    "Name": "Nationalities",
    "Namev2": "e.g. footballers or football-teams i.e. no .json
extension",
    "Created": "06 Nov 2024",
    "Purpose": "Display a list of all nationalities represented in
the footballing world.",
    "WebAPI": "The web api does not care what the format of this
file is. It simply reads it and writes it as a JSON string.",
    "Type": "Search form or CRUD form etc..",
    "Version": "1.0",
    "AuthorDetails": {
        "Copyright": "TriSys Business Software",
        "LastUpdated": "Monday 07 July 2025 10:58:08",
        "UpdatingUser": "Garry Lowther"
    },
    "Form": {
        "Icon": "gi gi-globe_af",
        "Caption": "Nationalities",
        "Caption_Notes": "Displayed in the title bar of the form,
but may be overriden by additional rules yet to be defined.",
        "Description": "Created by Josie Musto on Friday 11 April
2025",
        "EntityName": "Timesheet",
        "Buttons": [
            {
                "ID": "save",
                "Icon": "gi gi-floppy_saved",
                "Caption": "Save",
                "Function":
"TriSysFlexiva.Forms.Events.CRUD.Update()"
            },
            {
                "ID": "delete",
                "Icon": "gi gi-remove",
                "Caption": "Delete",
                "Function":
"TriSysFlexiva.Forms.Events.CRUD.Delete()"
            },
            {
                "ID": "actions",
                "Icon": "gi gi-magic",
                "Caption": "Actons",
                "Type": "dropdown",
                "Items": [
                    {
                        "ID": "action1",
                        "Caption": "Action 1",
```

```
                        "Icon": "gi gi-brightness_increase",
                        "Function":
"TriSysFlexiva.Forms.Events.CallCustomFormFunctionAnonymously('Butto
n_Click', { ID: 'action1' })"
                    },
                    {
                        "ID": "action2",
                        "Caption": "Action 2",
                        "Icon": "gi gi-screenshot",
                        "Function":
"TriSysFlexiva.Forms.Events.CallCustomFormFunctionAnonymously('Butto
n_Click', { ID: 'action2' })"
                    }
                ]
            }
        ],
        "Type": "Lookup",
        "RecordSummary": ""
    },
    "Tabs": [
        {
            "ID": "top",
            "Caption": "Main Top Region",
            "HTML": [
                "<div class=\\"block full form-control-
borderless\\">",
                "    <div class=\\"row\\" id=\\"cb5f7cfd-b694-6260-
17fd-08d410fe7963\\">",
                "        <div id=\\"233a1a48-4c67-62ff-b694-
51b3b49b449c\\" flexiva-data-
panel=\\"eyJpZCI6IjIzM2ExYTQ4LTRjNjctNjJmZi1iNjk0LTUxYjNiNDliNDQ5YyI
sInRpdGxlVGV4dCI6IllvdXIgUGFuZWwgVGl0bGUiLCJpY29uIjoiZ2kgZ2ktY2lyY2x
lX2V4Y2xhbWF0aW9uX21hcmsiLCJ0eXBlIjoiRm9ybSIsInNob3dUaXRsZSI6ZmFsc2U
sImJvcmRlciI6ZmFsc2UsInN0cmlwZWRSb3dzIjpmYWxzZSwiY29sdW1uQ291bnQiOjE
sInJvd3NQZXJDb2x1bW4iOjEsImxhYmVsV2lkdGgiOjE1MCwidGl0bGVCYWNrZ3JvdW5
kQ29sb3VyIjpudWxsLCJ0aXRsZUZvcmVncm91bmRDb2xvdXIiOm51bGx9\\"
class=\\"col-md-12\\">",
                "            <div class=\\"toolbox-component\\"
id=\\"48686a08-01a0-6113-a3b6-c1d45a04b7f0\\" flexiva-data-
component=\\"eyJJRCI6IjA3NGE2NDRhLTIyOTQtODc5NS0zODNhLTc2M2JiMTU4NzE
1NyIsIk5hbWUiOiJHcmlkIiwiRGVzY3JpcHRpb24iOiJEYXRhIGdyaWQgcG9pbnRpbmc
gYXQgc3RhdGljIGRhdGEgb3IgV2ViIEFQSSBkYXRhIHNvdXJjZSByZXF1ZXN0IiwiSWN
vbiI6ImdpIGdpLXRhYmxlIiwiVHlwZSI6IkdyaWQiLCJEYXRhU291cmNlcyI6W3siUmV
xdWVzdElkIjoiNDhiMTQ4YzEtNTRmZi1jYTUyLTdlNTAtNjhjNDEzMThkZTczIiwiRGl
zcGxheURlc2NyaXB0aW9uIjoiUmVzdERiLmlvICZuYnNwOyA+ICZuYnNwOyBGb290YmF
sbCAmbmJzcDsgPiAmbmJzcDsgTmF0aW9uYWxpdGllcyB3aXRoIEltYWdlcyIsIlR5cGU
iOiJSZWFkIn1dLCJHcmlkIjp7IkNvbHVtbnMiOlt7IkZpZWxkIjoiSUQiLCJUeXBlIjo
iU3RyaW5nIiwiVmFsdWVzIjoiMTUgdmFsdWVzOiA2N2Y4ZTg3NTc4YmFkZjY1MDAwNTI
```

```
yMzEsIDY3YTYwMjA4NDc3OWRlMWIwMDAyN2FmZSwgNjdmN2NiMTQ3OGJhZGY2NTAwMDR
mYjRkLCA2N2Y3Y2IzMzc4YmFkZjY1MDAwNGZiNTcsIDY3ZjdjYjUyNzhiYWQuLi4iLCJ
WaXNpYmxlIjpmYWxzZSwiQ2FwdGlvbiI6IklEIiwiS2V5Ijp0cnVlLCJLZXlWYXJpYWJ
sZSI6IjwjQ29tcG9uZW50LkdyaWQuTmF0aW9uYWxpdHlJRCM+IiwiV2lkdGgiOjB9LHs
iRmllbGQiOiJOYXRpb25hbGl0eSIsIlR5cGUiOiJTdHJpbmciLCJWYWx1ZXMiOiIxNSB
2YWx1ZXM6IEdlcm1hbiwgRnJlbmNoLCBQb3J0dWd1ZXNlLCBOb3J3ZWdpYW4sIEVneXB
0aWFuLCBTb3V0aCBLb3JlYW4sIEVuZ2xpc2gsIFNjb3R0aXNoLCBXZWxzaCwgSXJpc2g
sIER1dGNoLCBTcGFuaXNoLCBCZWxnaS4uLiIsIlZpc2libGUiOnRydWUsIkNhcHRpb24
iOiJOYXRpb25hbGl0eSIsIldpZHRoIjowLCJGaWx0ZXJhYmxlIjp0cnVlfSx7IkZpZWx
kIjoiRmxhZ1VybCIsIlR5cGUiOiJJbWFnZSBVUkwiLCJWYWx1ZXMiOiI0IHZhbHVlczo
gaHR0cHM6Ly9mb290YmFsbC04OTFiLnJlc3RkYi5pby9tZWRpYS82N2Y4ZTg3NDc4YmF
kZjY1MDAwNTIyMzAsIGh0dHBzOi8vZm9vdGJhbGwtODkxYi5yZXN0ZGIuaW8vbWVkaWE
vNjdmOTBmNzQ3OGJhZGY2NS4uLiIsIlZpc2libGUiOnRydWUsIkNhcHRpb24iOiJGbGF
nIiwiV2lkdGgiOiI5MCIsIkltYWdlU2l6ZSI6IjI0IHggMjQifSx7IkZpZWxkIjoiQ29
1bnRyeSIsIlR5cGUiOiJTdHJpbmciLCJWYWx1ZXMiOiIxNSB2YWx1ZXM6IEdlcm1hbnk
sIEZyYW5jZSwgUG9ydHVnYWwsIE5vcndheSwgRWd5cHQsIFNvdXRoIEtvcmVhLCBFbmd
sYW5kLCBTY290bGFuZCwgV2FsZXMsIElyZWxhbmQsIE5ldGhlcmxhbmRzL0hvbGxhbmQ
sIFNwYWluLy4uLiIsIlZpc2libGUiOnRydWUsIkNhcHRpb24iOiJDb3VudHJ5IiwiV2l
kdGgiOjAsIkZpbHRlcmFibGUiOnRydWV9LHsiRmllbGQiOiJGbGFnIiwiVHlwZSI6IlN
0cmluZyIsIlZhbHVlcyI6IjMgdmFsdWVzOiBbXCI2N2Y4ZTg3NDc4YmFkZjY1MDAwNTI
yMzBcIl0sIFtcIjY3ZjhlNWM0NzhiYWRmNjUwMDA1MjFlOFwiXSwgW1wiNjdmOTExOWQ
3OGJhZGY2NTAwMDUyNzM3XCJdIiwiVmlzaWJsZSI6ZmFsc2UsIkNhcHRpb24iOiJGbGF
nIiwiV2lkdGgiOjB9XSwiUG9wdWxhdGlvblRyaWdnZXIiOiJGb3JtIExvYWRlZCIsIlR
yaWdnZXJpbmdDb21wb25lbnQiOiIiLCJEcmlsbERvd24iOlt7IkNvbHVtbk5hbWUiOiJ
OYXRpb25hbGl0eSIsIkh5cGVybGluayI6dHJ1ZSwiS2V5Q29sdW1uTmFtZSI6IklEIiw
iQnV0dG9uQ29sdW1uIjpmYWxzZSwiQnV0dG9uTGFiZWwiOiIiLCJCdXR0b25XaWR0aCI
6MCwiRm9ybU5hbWUiOiJOYXRpb25hbGl0eSIsIkZvcm1Nb2RhbCI6ZmFsc2V9LHsiQ29
sdW1uTmFtZSI6IkNvdW50cnkiLCJIeXBlcmxpbmsiOnRydWUsIktleUNvbHVtbk5hbWU
iOiJJRCIsIkJ1dHRvbkNvbHVtbiI6ZmFsc2UsIkJ1dHRvbkxhYmVsIjoiIiwiQnV0dG9
uV2lkdGgiOjAsIkZvcm1OYW1lIjoiTmF0aW9uYWxpdHkiLCJGb3JtTW9kYWwiOmZhbHN
lfV19LCJMYWJlbFRleHQiOiJHcmlkIiwiTGFiZWxQb3NpdGlvbiI6IkxlZnQiLCJMYWJ
lbFZpc2libGUiOmZhbHNlLCJSZWFkT25seSI6ZmFsc2UsIkFsdFRpdGxlIjoiIiwiVGl
0bGUiOiIiLCJCb3JkZXJSYWRpdXMiOjAsIkJvcmRlcldpZHRoIjowLCJNYXhlZWlnaHQ
iOjAsIk1pbkhlaWdodCI6MCwiTWF4V2lkdGgiOjAsIk1pbldpZHRoIjowLCJDdXN0b21
WYXJpYWJsZSI6IiIsIkRlY2ltYWxQbGFjZXMiOjB9\\"
onclick=\\"ctrlFormDesigner2024.Components.Editor(this)\\"
style=\\"\\">",
                "            </div>",
                "        </div>",
```


400

```
{
  "error": "Invalid request"
}
```

Form URL

The URL of the form has to be determined by identifying the custom URL of the customers forms.

This is because the navigation bar only stores the name of the form, not it's full URL path.

Because each customer company has their own custom folder where their custom forms live, we need to read the custom folder from this API call made after login. This is the relevant data which was returned:

```
{
    "LoggedInUser": {
        "CRMContact": {
            "CustomForms": [
                {
                    "FileName": "Nationalities.json",
                    "FilePath": "custom/boggy-
frogs/forms/nationalities.json",
                    "FullServerPath": nulll
                }
            ],
            ...
        }
    }
}
```

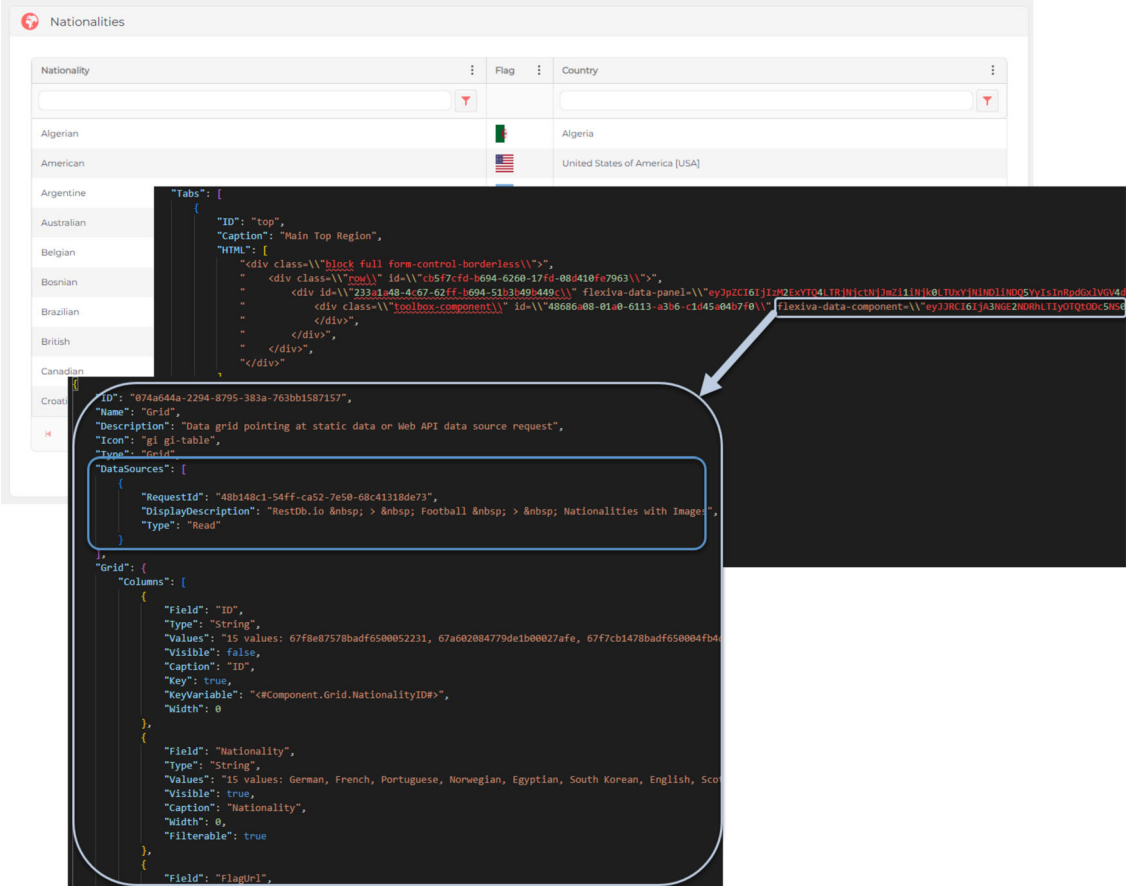
If you know the name of the form when the navigation bar item is clicked e.g. `nationalities.json`, then you can find this in the list of custom forms available to the logged in user.

The full URL path is thus the API Endpoint + FilePath e.g.

`https://api.trisys.co.uk/custom/boggy-frogs/forms/nationalities.json`

Rendered Form

This is how this example form looks at run-time after being rendered. The middle layer shows the portion of JSON which is the form design, showing a `flexiva-data-component` Base64 string. The top layer shows the decoded JSON for this component which is a grid with specified columns and a data source request ID.



The application must render the form, then generate the data grid, then send this specific request to the Web API to read the data from the ReST API to populate the grid.

Send ReST API Request

Send the data source request via the Web API proxy server for CRUD operations.

This function is how client-side applications securely access server-side ReST API requests without having to know about security credentials.

All ReST API CRUD operation for custom forms, and activity metrics calls this function which is designed to handle any type of ReST API for all HTTP methods ↗.

Send App Studio Data Source Request

`POST` `AppStudio/InvokeCRUDoperationUsingThirdPartyDataSource`

This is a POST method which passes both the API and session keys as headers, and a body. If successful, the custom data sources can be stored.

Headers

Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Body

Name	Type	Description
`BodyJson`	String	JSON body specified when data source request was connected in App Studio.
`CustomVariables`	List	A list of custom variables assigned to fields on the form, or components on the form. This passes the context of the client-side data to the server for processing.
`RequestGUID`	String	The data source request ID associated with the form, component or field. Identifying this is described here.
`RequestURL`	String	This is optionally supplied only when the designer is creating and testing data source requests and wishes to manually override parameters.

Response Snippet

200

```
{
  "Success": true,
  "URL": "https://.....",
  "Verb": "GET",
  "Columns": [
      {
         "field": "_id",....
     }, ....
  ],
  "DataTable": {
    "FirstRowNumber": 0,
    "LastRowNumber": 0,
    "List": [
      {
        "Country": "Mexico",
        "Flag": "[\r\n  \"681c5e4f78badf65000b663e\"\r\n]",
        "Name": "Mexican",
        "_id": "681c5e4f78badf65000b663f"
      }
    ],
    "PageNumber": 1,
    "RecordsPerPage": 10,
    "Success": false,
    "TotalPageCount": 1,
    "TotalRecordCount": 1
  }
}
```

400

```
{
  "error": "Invalid request"
}
```

Pagination

The response snippet above shows how this Web API call always returns data in this format with columns and a data table containing a list. This is because the list may only be a small part of the entire data table i.e. only a page of data is returned. These are useful when the ReST API is enabled for paging to optimise performance for both the client and server by sending only small pages of data, rather than the entire data set which could be millions of records.

This is a complex subject discussed for production designers here.

Full Response Sample

This shows the data set displayed in a data grid.

Nationalities

Nationality	Flag	Country
Algerian		Algeria
American		United States of America [USA]
Argentine		Argentina
Australian		Australia
Belgian		Belguim
Bosnian		Bosnia & Herzegovina
Brazilian		Brazil
British		Great Britain
Canadian		Canada
Croatian		Croatia

Page 1 of 4 10 rows per page 1 to 10 of 35 rows

Here is the response from the Web API after it has executed the ReST API request:

```
{
    "Columns": [
        {
            "field": "_id",
            "title": "Id",
            "type": "string",
            "format": null,
            "width": 100,
            "hidden": true,
            "template": null
        },
        {
            "field": "Country",
            "title": "Country",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "Name",
            "title": "Name",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        },
        {
            "field": "Flag",
            "title": "Flag",
            "type": "string",
            "format": null,
            "width": 200,
            "hidden": false,
            "template": null
        }
    ],
    "DataTable": {
        "List": [
            {
                "_id": "67f8e87578badf6500052231",
                "Country": "Germany",
                "Name": "German",
                "Flag": "[\\r\\n  \\"67f8e87478badf6500052230\\"\\r\\n]"
            },
            {
```

```
            "_id": "681c5e4f78badf65000b663f",
            "Country": "Mexico",
            "Name": "Mexican",
            "Flag": "[\\r\\n  \\"681c5e4f78badf65000b663e\\"\\r\\n]"
        },
        {
            "_id": "6863e59178badf6500137d4f",
            "Country": "Nigeria",
            "Name": "Nigerian",
            "Flag": "[\\r\\n  \\"6863e59178badf6500137d4e\\"\\r\\n]"
        },
        {
            "_id": "67f7cb3378badf650004fb57",
            "Country": "Norway",
            "Name": "Norwegian",
            "Flag": "[\\r\\n  \\"67fe0bf178badf650005cfb5\\"\\r\\n]"
        },
        {
            "_id": "67f7cb6f78badf650004fb67",
            "Country": "South Korea",
            "Name": "South Korean",
            "Flag": "[\\r\\n  \\"67fe0c2c78badf650005cfbf\\"\\r\\n]"
        },
        {
            "_id": "6863f5f478badf650013800d",
            "Country": "Denmark",
            "Name": "Danish",
            "Flag": "[\\r\\n  \\"6863f5f478badf650013800c\\"\\r\\n]"
        },
        {
            "_id": "6863e4f378badf6500137d33",
            "Country": "Morocco",
            "Name": "Morocco",
            "Flag": "[\\r\\n  \\"6863e4f378badf6500137d32\\"\\r\\n]"
        },
        {
            "_id": "67f7cba378badf650004fb74",
            "Country": "Ireland",
            "Name": "Irish",
            "Flag": "[\\r\\n  \\"67fe0d3478badf650005cfdc\\"\\r\\n]"
        },
        {
            "_id": "67f7cbea78badf650004fb8d",
            "Country": "Spain/Espana",
            "Name": "Spanish",
            "Flag": "[\\r\\n  \\"67fe0d6d78badf650005cff1\\"\\r\\n]"
        },
        {
```

```
        "_id": "67a601e44779de1b00027af9",
        "Country": "Great Britain",
        "Name": "British",
        "Flag": "[\\r\\n  \\"67fa327278badf6500054bc4\\"\\r\\n]"
    },
    {
        "_id": "67a601f84779de1b00027afc",
        "Country": "Brazil",
        "Name": "Brazilian",
        "Flag": "[\\r\\n  \\"67f9119d78badf6500052737\\"\\r\\n]"
    },
    {
        "_id": "67a602084779de1b00027afe",
        "Country": "France",
        "Name": "French",
        "Flag": "[\\r\\n  \\"67fa30a978badf6500054b87\\"\\r\\n]"
    },
    {
        "_id": "67f7cbff78badf650004fb92",
        "Country": "Belguim",
        "Name": "Belgian",
        "Flag": "[\\r\\n  \\"67fa312678badf6500054ba0\\"\\r\\n]"
    },
    {
        "_id": "67f7cbce78badf650004fb81",
        "Country": "Netherlands/Holland",
        "Name": "Dutch",
        "Flag": "[\\r\\n  \\"67fa31a478badf6500054bb4\\"\\r\\n]"
    },
    {
        "_id": "67f7cb5278badf650004fb5d",
        "Country": "Egypt",
        "Name": "Egyptian",
        "Flag": "[\\r\\n  \\"67fa32d678badf6500054bcc\\"\\r\\n]"
    },
    {
        "_id": "67f7cb8278badf650004fb6b",
        "Country": "England",
        "Name": "English",
        "Flag": "[\\r\\n  \\"67fa336b78badf6500054bea\\"\\r\\n]"
    },
    {
        "_id": "67f7cb1478badf650004fb4d",
        "Country": "Portugal",
        "Name": "Portuguese",
        "Flag": "[\\r\\n  \\"67fd5d3578badf650005ba96\\"\\r\\n]"
    },
    {
        "_id": "67f7cb8e78badf650004fb6d"
```

```
        "_id": "67f7cb8e78badf650004fb6d",
        "Country": "Scotland",
        "Name": "Scottish",
        "Flag": "[\\r\\n  \\"67fe0c6078badf650005cfc4\\"\\r\\n]"
    },
    {
        "_id": "67f7cb9778badf650004fb6f",
        "Country": "Wales",
        "Name": "Welsh",
        "Flag": "[\\r\\n  \\"67fe101b78badf650005d03d\\"\\r\\n]"
    },
    {
        "_id": "681b7aaa78badf65000b3561",
        "Country": "Falkland Islands",
        "Name": "Falkland Islander",
        "Flag": "[\\r\\n  \\"681b7aa978badf65000b3560\\"\\r\\n]"
    },
    {
        "_id": "681b79b578badf65000b353a",
        "Country": "Croatia",
        "Name": "Croatian",
        "Flag": "[\\r\\n  \\"681b79b578badf65000b3539\\"\\r\\n]"
    },
    {
        "_id": "681b924878badf65000b3979",
        "Country": "Mauritius",
        "Name": "Mauritian",
        "Flag": "[\\r\\n  \\"681b924878badf65000b3978\\"\\r\\n]"
    },
    {
        "_id": "681b84b078badf65000b371b",
        "Country": "Kazakhstan",
        "Name": "Kazakh",
        "Flag": "[\\r\\n  \\"681b84b078badf65000b371a\\"\\r\\n]"
    },
    {
        "_id": "681de29778badf65000babea",
        "Country": "United States of America [USA]",
        "Name": "American",
        "Flag": "[\\r\\n  \\"681de29778badf65000babe9\\"\\r\\n]"
    },
    {
        "_id": "686e910978badf65001445dc",
        "Country": "Algeria",
        "Name": "Algerian",
        "Flag": "[\\r\\n  \\"686e910878badf65001445db\\"\\r\\n]"
    },
    {
        "_id": "6863e1d478badf6500137cc2",
```

```
        "Country": "Argentina",
        "Name": "Argentine",
        "Flag": "[\\r\\n  \\"6863e1d478badf6500137cc1\\"\\r\\n]"
    },
    {
        "_id": "6863f46478badf6500137fc8",
        "Country": "Canada",
        "Name": "Canadian",
        "Flag": "[\\r\\n  \\"6863f46478badf6500137fc7\\"\\r\\n]"
    },
    {
        "_id": "6863e45878badf6500137d1a",
        "Country": "Australia",
        "Name": "Australian",
        "Flag": "[\\r\\n  \\"6863e45778badf6500137d19\\"\\r\\n]"
    },
    {
        "_id": "6863e78278badf6500137da9",
        "Country": "Switzerland",
        "Name": "Swiss",
        "Flag": "[\\r\\n  \\"6863e78278badf6500137da7\\"\\r\\n]"
    },
    {
        "_id": "6863e37978badf6500137cfa",
        "Country": "Ivory Coast",
        "Name": "Ivorian",
        "Flag": "[\\r\\n  \\"6863e37978badf6500137cf9\\"\\r\\n]"
    },
    {
        "_id": "6863ee4f78badf6500137eb7",
        "Country": "Bosnia & Herzegovina",
        "Name": "Bosnian",
        "Flag": "[\\r\\n  \\"6863ee4f78badf6500137eb6\\"\\r\\n]"
    },
    {
        "_id": "6863fe6578badf65001381ef",
        "Country": "Sweden",
        "Name": "Swedish",
        "Flag": "[\\r\\n  \\"6863fe6578badf65001381ee\\"\\r\\n]"
    },
    {
        "_id": "686f8b6478badf650014590f",
        "Country": "Italy",
        "Name": "Italian",
        "Flag": "[\\r\\n  \\"686f8b6378badf650014590e\\"\\r\\n]"
    },
    {
        "_id": "686f88dc78badf65001458ce",
```


Send App Studio Data Source Request

```
                "Country": "Slovakia",
                "Name": "Slovakian",
                "Flag": "[\\r\\n  \\"686f88dc78badf65001458cd\\"\\r\\n]"
            },
            {
                "_id": "686f8d8b78badf6500145941",
                "Flag": "[\\r\\n  \\"686f8d8b78badf6500145940\\"\\r\\n]"
            }
        ],
        "DynamicColumns": null,
        "TotalRecordCount": 0,
        "TotalPageCount": 0,
        "FirstRowNumber": 0,
        "LastRowNumber": 0,
        "PageNumber": 1,
        "RecordsPerPage": 35,
        "AICriteria": null,
        "Success": false,
        "ErrorMessage": null
    },
    "Verb": "GET",
    "Success": true,
    "ErrorMessage": null
}
```


Name	Value
Content-Type	`application/json`
SiteKey	`API Key`
DataServicesKey	`Session Key`

Name	Type	Description
`Expression`	String	The text used to search over all entities e.g. "Dan".
`MaximumRecordsPerEntity`	Number	The maximum number of records to return.
`RequestId`	String	The data source request ID of the specific cross-entity search ReST API.

Response Snippet

200


```
{
  "Success": true,
  "ErrorMessage": null,
  "Items": [
        {
            "EntityName": "Footballer",
            "Display": "Dan Burn, Defender",
            "EntityID": "67fe29f678badf650005d3ee"
        },
        {
            "EntityName": "Footballer",
            "Display": "Danny Ings, Forward",
            "EntityID": "6863ef6c78badf6500137eec"
        },
        {
            "EntityName": "Nationality",
            "Display": "Denmark",
            "EntityID": "6863f5f478badf650013800d"
        }
    ]
}
```


400


```
{
  "error": "Invalid request"
}
```


Entity Name

This should be the name of the form in the application configuration so that the client-side app can obtain the form icon to display.

Display

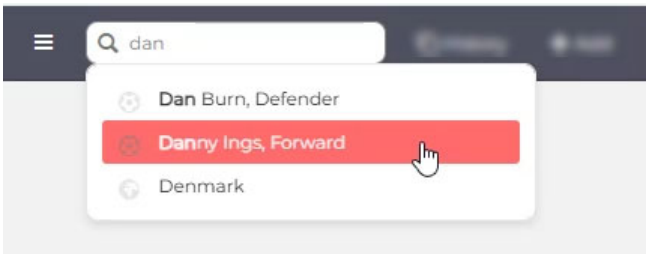
This is the text to display in the client-side app for each result.

EntityID

This is the unique identifier of the record returned. If the end-user clicks this item, the client-side app should then be able to open the matching form record.

Response Example

This is how this example response could like in a client-side app:



Summary

Summary of building client-side apps which utilise the API and the application configuration.

The API and accompanying documentation has allowed you to build new and fresh client-side applications which utilise the platform independent flexible integrated visual application designed for your business.

By adhering to the Flexiva framework, customers are in control of which ReST API's they wish to connect to, and by using the App Studio, they can control accessibility. Your app is free from having to worry about all of these things.

It is likely that your application has utilised artificial intelligence somewhere along the line, to help you build this new Flexiva client much faster than previously possible.