

DX Integration with HCL Volt MX for Developers

HCLSoftware U

Creating a new generation of experts

Edition: March 2024

Herbert Hilhorst – herbert.hilhorst@hcl-software.com

Dai Nguyen – dai.nguyen@hcl-software.com

Mario D'Anna – mario.danna@hcl-software.com

Table of Contents

Author(s)..... 3

Introduction..... 4

Prerequisites..... 5

Lab Overview 6

Part 1: Use DX Digital Assets in a new HCL Volt MX Mobile Application..... 9

Part 2: Use DX Content in a new HCL Volt MX Web Application 33

Part 3: Use Volt MX Foundry to Integrate External Data Sources in DX 63

Part 4: Integrate Volt MX Foundry Web Applications in DX 64

Part 5: Use Volt MX Iris to turn a DX Site into a Native Mobile Application..... 67

Conclusion 73

Resources 74

Legal statements 75

Disclaimers..... 76

Author(s)

This document was created by the following Subject Matter Experts:



Herbert Hilhorst
Company:
HCLSoftware

Bio

Herbert Hilhorst is an HCL Digital Experience (DX) Technical Advisor at HCLSoftware.

Contact: herbert.hilhorst@hcl-software.com



Dai Nguyen
Company:
HCLSoftware

Bio

Dai Nguyen is an HCL Volt MX Technical Advisor at HCLSoftware.

Contact: dai.nguyen@hcl-software.com



Mario D'Anna
Company:
HCL Software

Bio

Mario Danna is an HCL Volt MX Senior Manager at HCLSoftware.

Contact: mario.danna@hcl-software.com

Introduction

HCL Volt MX Foundry is a back-end service provider that helps developers build omni-channel digital applications. Volt MX Foundry allows you to define the back-end to build native mobile apps for iOS, Android, and Windows and SPAs and Responsive web apps for browsers. Volt MX Foundry ensures that developers build mobile applications quickly and obtain secured back-end services instantly. Volt MX Foundry has multiple features, such as - Identity, Integration, Objects, Orchestration, and Engagement Services. These features can be accessed through a common, centralized Volt MX Foundry Console.

Many applications require imagery, videos, and content to bring them to life and create an engaging user experience. Such content and assets often come from a central content & asset management system, which is used by multiple mobile and non-mobile applications, to ensure consistent communication and branding across all customer interactions. This also allows business users to update the content & assets without needing to republish the application. HCL Digital Experience (DX) provides content and asset services, as well as services to personalize them, accessible via a set of GraphQL and REST APIs via HCL Volt MX Foundry.

Example: include news, help guides, tutorials, or other corporate content in your application; retrieve imagery or support videos from the asset management service for consistent branding & quick update.

This hands-on lab gets you started on the HCL Digital Experience (DX) platform and its integration with HCL Volt MX. You will learn how to manage digital assets and content centrally and use it in HCL Volt MX to manage your applications.

You will also learn how to use Foundry to integrate external data sources and Volt MX Web applications into DX. And you will learn how you may HCL Volt MX Iris to turn your existing DX sites into native mobile applications.

In this DX developer lab, you play the role of Gene, a developer for the fictitious Woodburn Studio company.



Gene Hayes, Developer, based in Chicago (USA)

As Gene, you will learn how to manage the REST APIs for DX assets in Foundry and build a mobile application using Iris. Then you learn how to manage the REST API for DX content in Foundry and build a web application with Iris. You learn how to use Foundry to integrate external data sources into DX and how to integrate Volt MX web applications into DX. And you will create a native mobile application of the Woodburn Stores DX site.

Prerequisites

1. Completion of the [HDX-INTRO](#) course as this gives you access to your own DX Solution Modules instance on HCL SoFy
2. Completion of Digital Assets and Web Content lessons of [HDX-BU-100](#), as this gives you some assets and content to be used in this lab
3. Completion of Experience API of [HDX-DEV-100](#) as this helps you setting up the security to use the REST APIs and gets you started using them
4. Completion of Introduction of [HDX-INT-DEV](#) as this helps you understanding how to access the assets and content externally easily
5. A HCL Volt MX trial or live environment to develop HCL Volt MX applications that integrate your DX assets and content and Volt MX application installed on your favorite device(s). You may use HCL Volt MX Go on HCL SoFy (<https://hclsofy.com/catalog/hcl-voltmxgo>) or the Volt MX Hackathon Environment Setup guide to create your trial environment: https://drive.google.com/file/d/17kP86SH7NCQarMXwtiag5OMitSrmG_hR/view

You will be using the following user IDs and passwords:

Purpose	User	Password
SoFy Login	Your official email id	Your password
SoFy Solution Console Login	sol-admin	<from solution console>
DX Developer Login	ghayes	HCL-Dem0

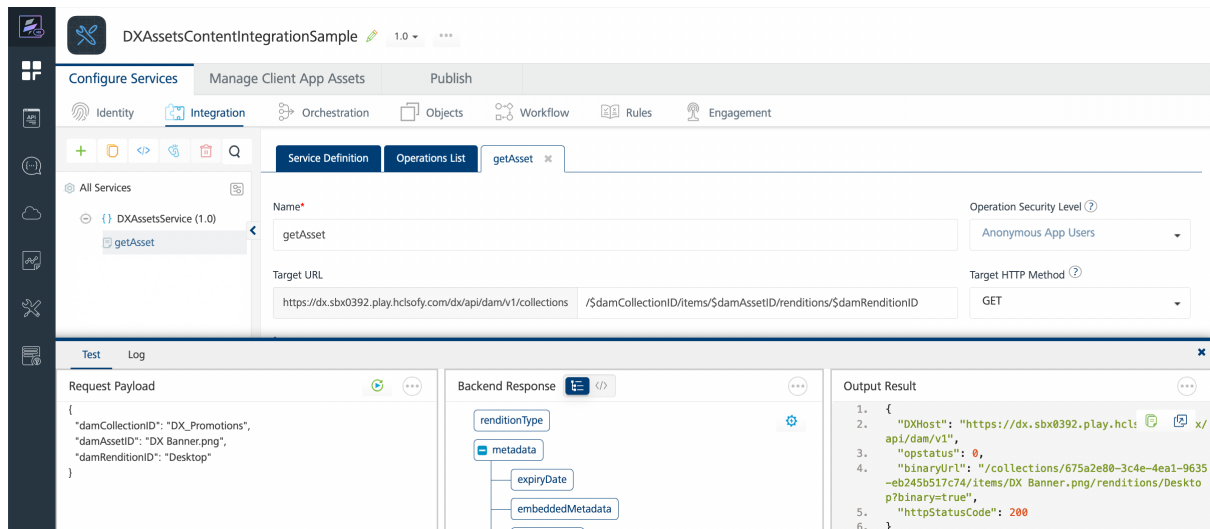
Lab Overview

In this lab, you will explore the integration of HCL Digital Experience assets and content in HCL Volt MX.

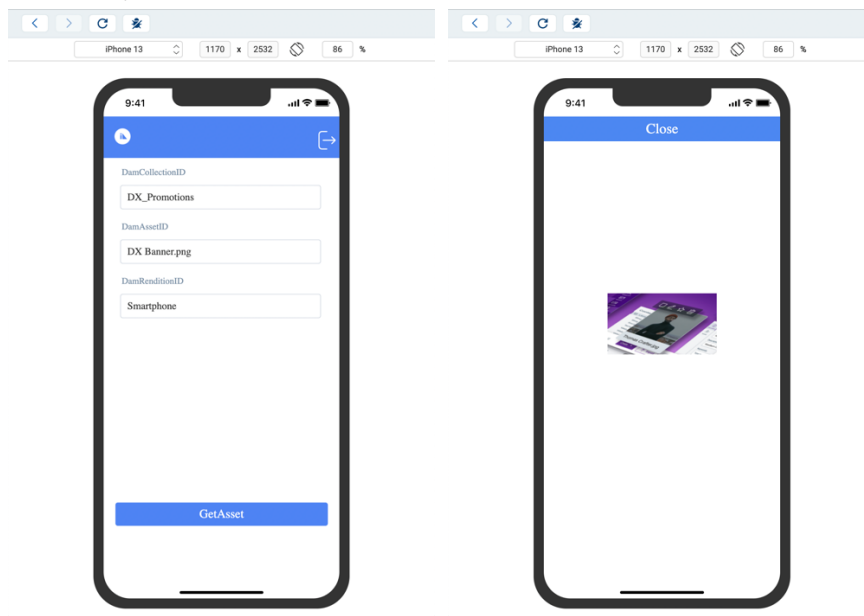
There are several parts in this lab, shortly introduced now.

Part 1: Use DX Assets in a new HCL Volt MX Mobile Application

In this part, you will develop, build, and run your Volt MX mobile application that uses your HCL Digital Experience assets. You will build a Volt Foundry Integration service to get the right details on an asset.

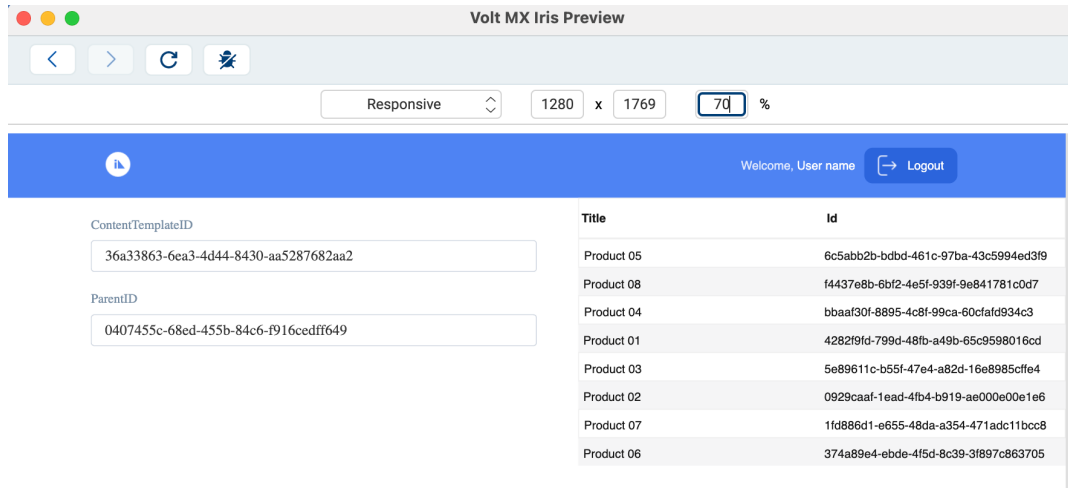


And a Volt Iris project to access DX assets. This is how your mobile application will look like, using the collection, asset and rendition IDs.

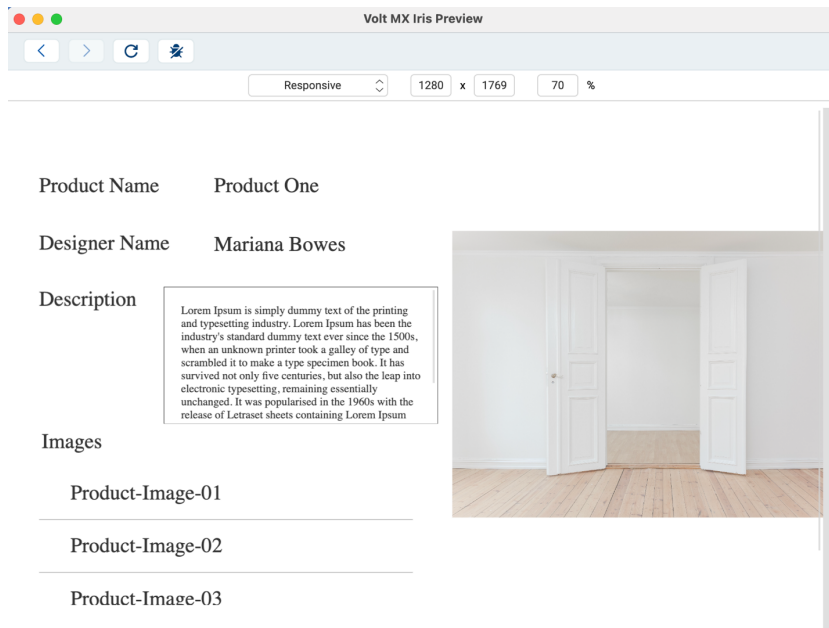


Part 2: Use DX Content in a new HCL Volt MX Web Application

In this part, you will learn how to develop, build, and run a new Volt MX web application that uses your HCL Digital Experience content. To facilitate communication between the DX backend content and Foundry, you will create integration services in Foundry that will search DX content, based on a Content Template ID and Parent ID (Site Area) as input to the service, and a service to get details on a single content. Then you will build your web application with Iris that uses these services. It allows you to select a ContentTemplateID and ParentID and show the titles and IDs of the content that correspond.

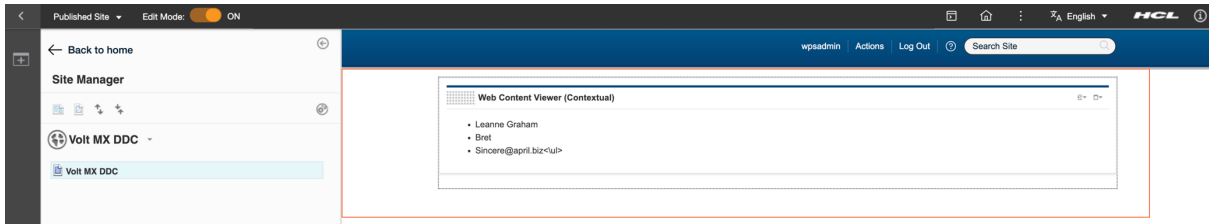


And when selecting a product, you will see the details.



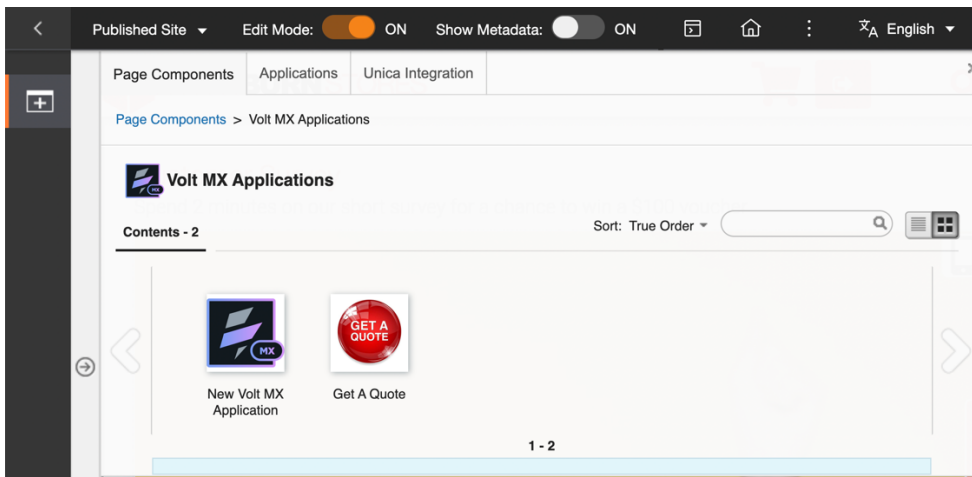
Part 3: Use Volt MX Foundry to Integrate External Data Sources in DX

In this part, you will learn how use Digital Data Connector to integrate data sources from HCL Volt MX Foundry into HCL Digital Experience.



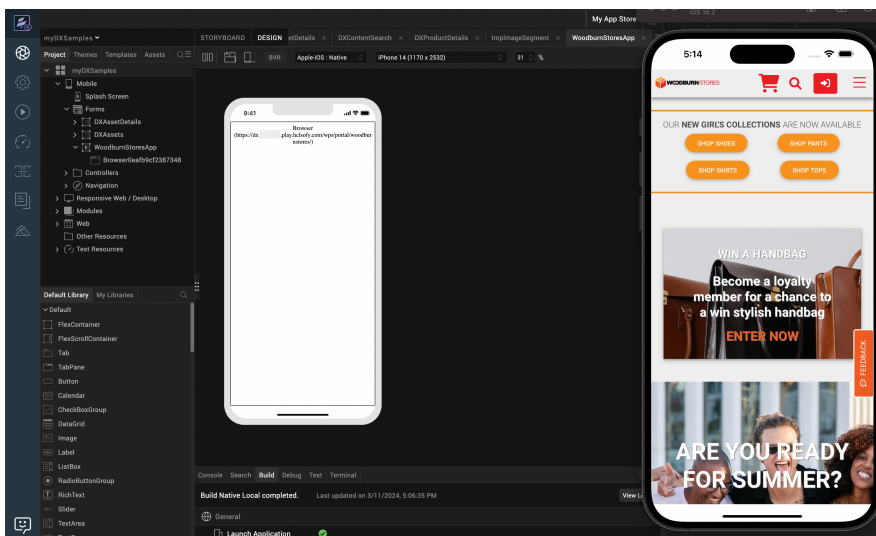
Part 4: Integrate Volt MX Foundry Web Application in DX

And you learn how to integrate Foundry web applications into HCL Digital Experience.



Part 5: Use Volt MX Iris to turn a DX Site into a Native Mobile Application

And you learn how you may use Volt MX Iris to build a native mobile application from any DX site and deploy this to any store, like Apple AppStore and Google Play.



Part 1: Use DX Digital Assets in a new HCL Volt MX Mobile Application

In this part, you will develop, build, and run your Volt MX mobile application that uses your HCL Digital Experience assets. You will build a Volt Foundry Integration service to get the right details on an asset and a Volt Iris project to access DX assets.

As you have configured the DX DAM to provide anonymous access, you can easily integrate them in Volt MX. You can directly use the DAM asset URL in your application. In this part you will build a Volt Foundry Integration service and a Volt Iris project to access DX assets using the REST API. It fetches a DX DAM asset, based on three input parameters. In the DX Integration Introduction, you learned how to access the asset with this REST API (see <https://opensource.hcltechsw.com/experience-api-documentation/dam-api/#operation/RenditionController.getRenditionById>):

API URL:

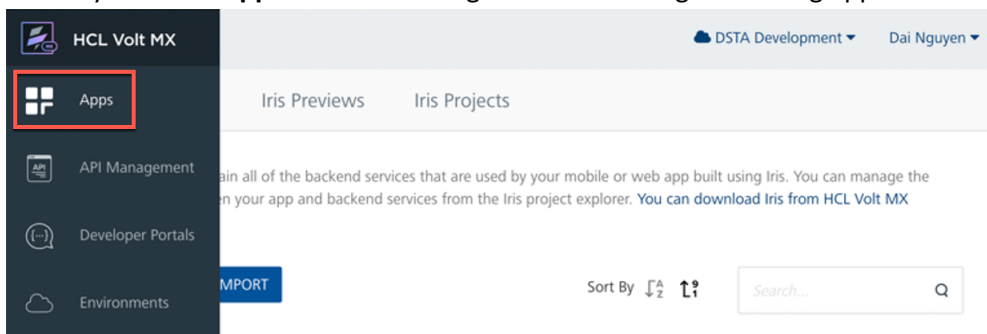
```
<host>/dx/api/dam/v1/collections/<collection_id>/items/<content_id>/renditions/<rendition_id>
```

The first parameter is the “DAM Collection ID”, and the second is the “DAM Asset ID”. The third parameter “DAM Rendition ID” may be determined automatically depending on the device that is using the application. The output parameters, that your Volt Iris app will display, are the DAM binaryUrl and the DX Host. Your application will generate a valid URL to the DAM image.

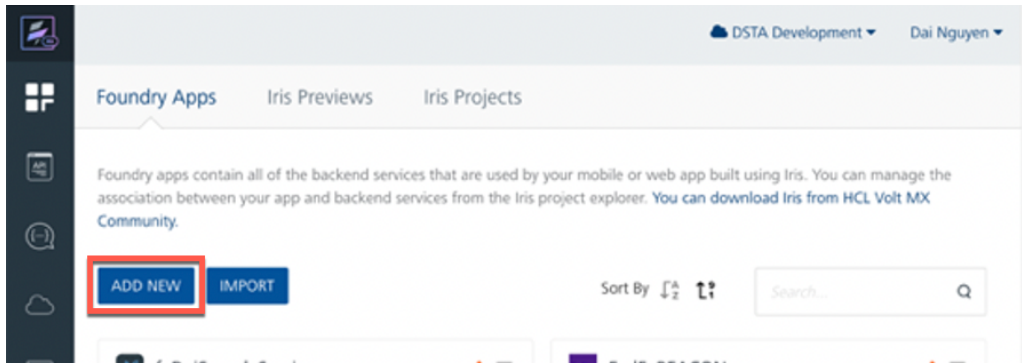
Ensure you have the console URL (for example, <https://manage.demo-hclvoltmx.com/>) to your cloud-Foundry and the required Volt MX credentials (email and password) to login to the Foundry.

If you are working with Volt Iris, in any of the steps below, ensure you save your updates regularly, at least once per step, using the menu Project – Save All or pressing CTRL S.

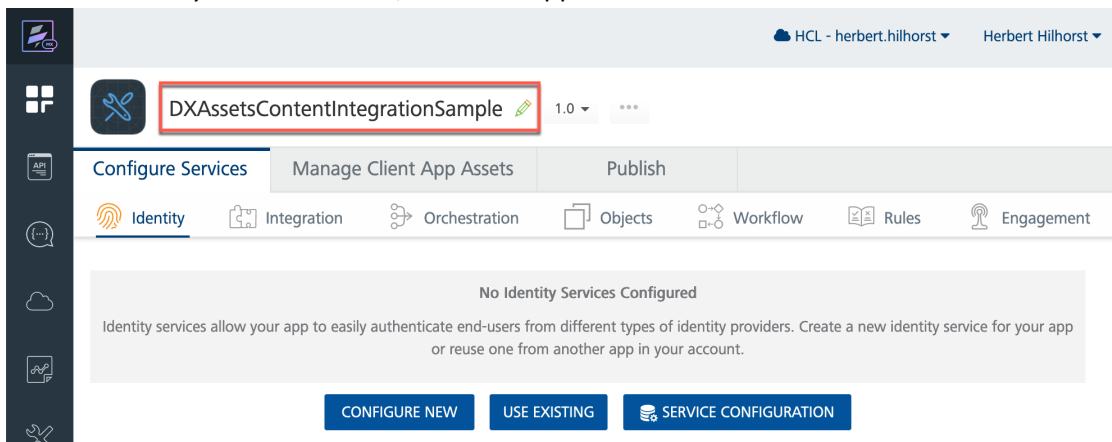
1. You first need to create a Foundry application that contains the service to access the DAM to invoke that you will link to in your Iris application. While you could use the REST APIs directly, here you will use the Foundry API Application Management capability to create more consumable APIs to access resources in DX. Expand the left-hand-navigation-bar of Foundry and click **Apps** to start creating a new or viewing an existing application.



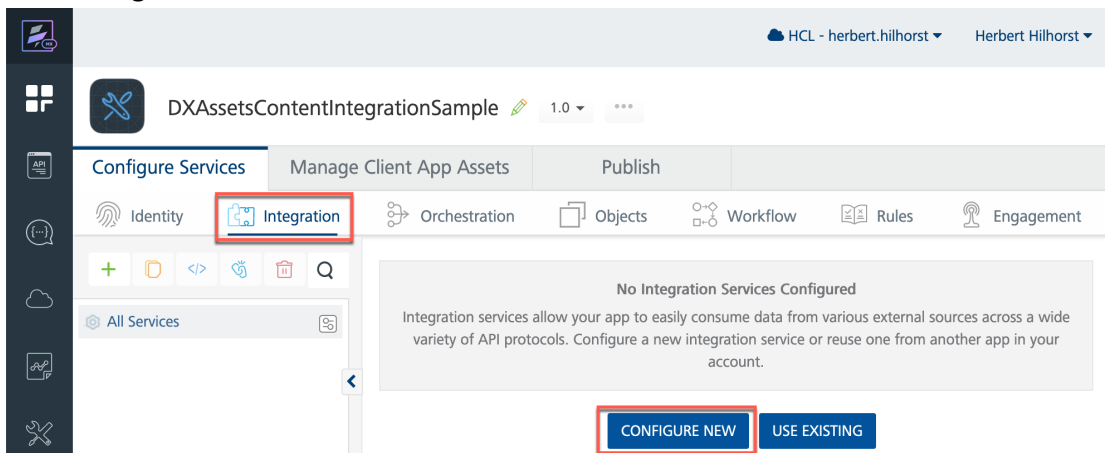
- Then click **Add New** to create your Foundry application.



- And update the application name with a good name, e.g. **DXAssetsContentIntegrationSample**. Note that the Foundry application name does not need to be the same as your Iris web and/or mobile application.



- Then add the integration services to access the assets in DX that will contain your new REST API calls. They are based on JSON. Under the **Configure Services** tab, click on **Integration** and then **Configure New**.



- Name your new Integration Service, such as **DXAssetsService**. Note that service names need to be unique to a Foundry instance. You cannot have another service in your Foundry with the same name, even in another application. Select **JSON** as **Service Type**.

Now you will configure this Integration service to be perform the GET DAM REST API call with your SoFy instance, which has this format (see

<https://opensource.hcltechsw.com/experience-api-documentation/dam-api/#operation/RenditionController.getRenditionById>):

```
<host>/dx/api/dam/v1/collections/<collection_id>/items/<asset_id>/renditions/<rendition_id>
```

First set the **Base URL** to **https://<host>/dx/api/dam/v1/collections** from this DAM REST API URL (e.g. **https://dx.sbx0000.play.hclsofy.com/dx/api/dam/v1/collections** with your SoFy host). Click **SAVE & ADD OPERATION**.

The screenshot shows the Foundry configuration interface for a new Integration Service. The service is named **DXAssetsService**, has a **Service Type** of **JSON**, and a **Base URL** of **https://dx.sbx0000.play.hclsofy.com/dx/api/dam/v1/collections**. The **Path Expression** is set to **JSON Path** and the **Version** is **1.0**. The **Web Service Authentication** is set to **None** and the **Identity Service for Backend Token** is also set to **None**. The **Advanced** section is expanded, showing a **Description** field. The **SAVE & ADD OPERATION** button is highlighted.

- Services are bundles of operations, each connecting to a full-qualified endpoint relative to the BASE URL of the service. You will need to add an operation to your service definition above to make the service usable. Name your new Operation **getAsset**. Set the **Operation Security Level** to **Anonymous App Users**, as it does not require a specific Identity to authenticate against before connecting to this operation. Ensure your **Target HTTP Method** is set to **GET**. Using **\$** parameters to compose the **Target URL** means those parameter values will come from your **Request Input** parameter list (path params). To replace a param **\$SomeName**, you will need to pass a Request Input SomeName from the application invoking this Service/Operation.

Note that the Request Input parameter name must match those in the Target URL field.

Set the **Target URL** to match the GET DAM REST API call using these path parameters for collection_id, asset_id and rendition_id:

/\$damCollectionID/items/\$damAssetID/renditions/\$damRenditionID

The screenshot shows the configuration interface for a service named "DXAssetsContentIntegrationSample". The "Operations List" tab is active, displaying a single operation named "getAsset". The configuration details for this operation are as follows:

- Name:** getAsset
- Operation Security Level:** Anonymous App Users
- Target URL:** https://dx.play.hclsofy.com/dx/api/dam/v1/collections/\$damCollectionID/items/\$damAssetID/renditions/\$damRenditionID
- Target HTTP Method:** GET

The interface also shows a sidebar with "All Services" and "DXAssetsService (1.0)" containing the "getAsset" operation. At the bottom, there are tabs for "Request Input" and "Response Output", and a "Body" section with "Header" and "Body" sub-sections. There are also buttons for "Add Parameter", "Copy", "Paste", "Delete", and "Request Template" (Show/Hide).

- Then add the path and query parameters and set their default values. Scroll down and under **Request Input**, click **Add Parameter** to add each parameter: **damCollectionID** with the **Default Value** to your DX DAM Collection ID, such as **ccfd0442-36ba-44e5-9c27-05299727635b** or **DX_Promotions** (Unique name), and **damAssetID** with the **Default Value** to your DX DAM Asset ID, such as **ccfd0442-1bdf5cdd-3f5d-45be-b595-d5126832b597** or asset file name, such as **DX Banner.png** and the **damRenditionID** with the **Default Value** for mobile (as you are creating a mobile application this should be **Smartphone**). Select your preferred Foundry runtime environment in the bottom to test this service against (ideally the same runtime environment you'll eventually be publishing this service to) and click **SAVE AND FETCH RESPONSE** to save your Integration service operation and make a test call immediately.

Request Template ? Show Hide

Search Q

<input type="checkbox"/>	NAME	VALUE ?	TEST VALUE	DEFAULT VALUE	DATA TYPE	ENCODE	DESCRIPTION
<input type="checkbox"/>	damCollectionID	request		DX_Promotions	string	<input checked="" type="checkbox"/>	
<input type="checkbox"/>	damAssetID	request		DX Banner.png	string	<input checked="" type="checkbox"/>	
<input type="checkbox"/>	damRenditionID	request		Smartphone	string	<input checked="" type="checkbox"/>	

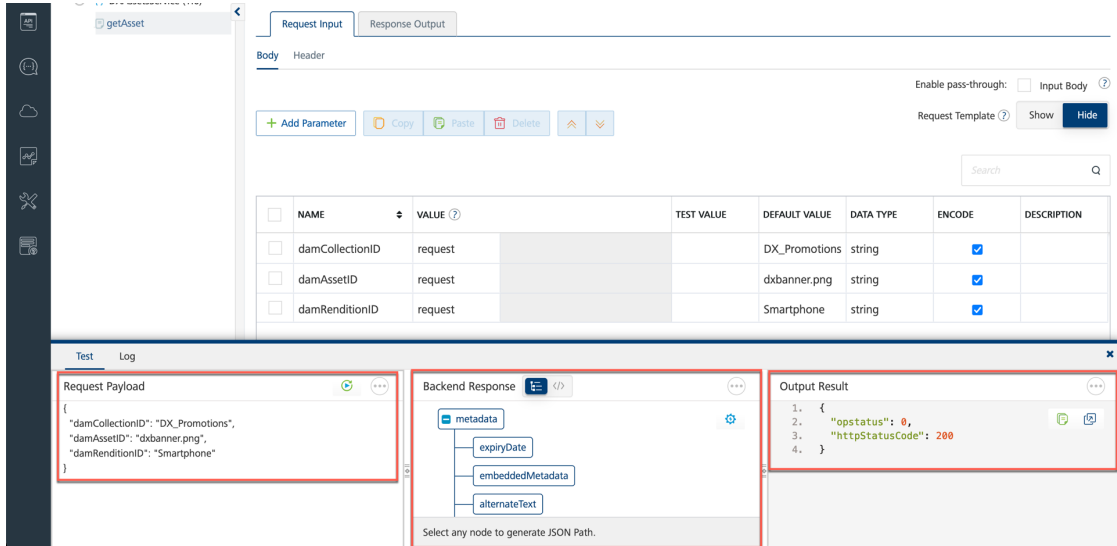
Default value will be used if Test value is empty.

m10000

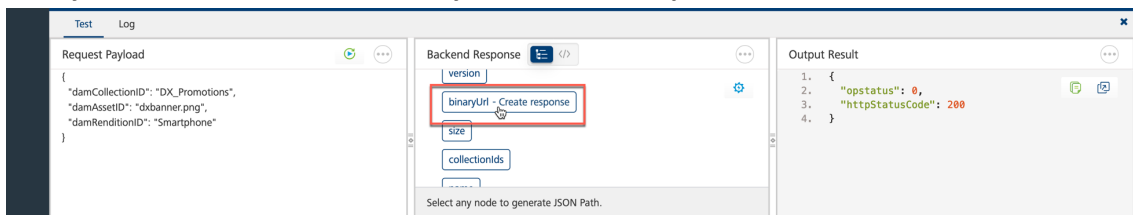
SAVE AND FETCH RESPONSE

CANCEL SAVE OPERATION

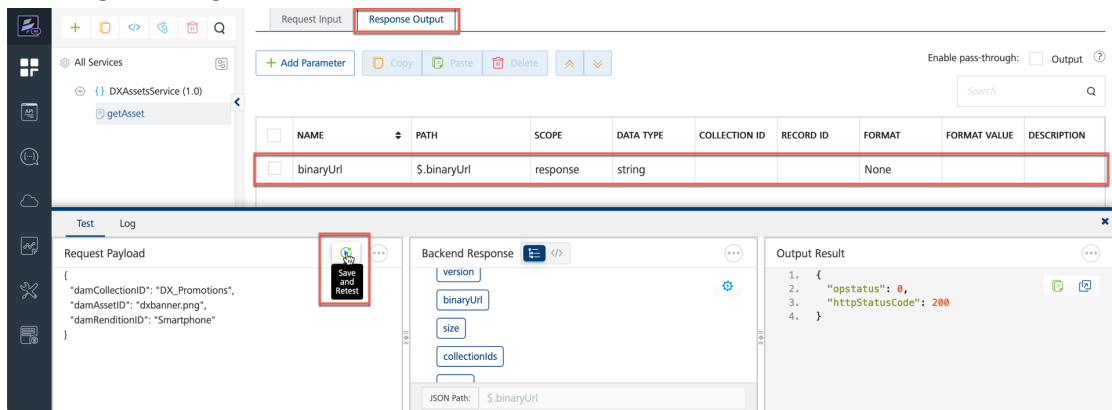
- The Test panel is appearing now and shows the **Test** and **Log** sections. The Test sections shows the **Request Payload**, the **Backend Response** and the **Output Result**. **Backend Response** displays what DX communicated back to Foundry, and **Output Result** is what Foundry will relay to the Iris application invoking this service. Since we have not specified what data-points to pick and choose for our application or enabled the entire output to *pass-through* to the application, **Output Result** only includes **opstatus** and **httpStatusCode** by default. You want also the **binaryURL** in the output result.



- You want to add the **binaryURL** to this result. In this lab, you are focusing on images only, so you know the format of the asset to display. If you want to manage different asset types, you may want to add a mime-type here, as this may help you identify what asset you are finding and how to display it correctly. You will focus on image assets for now, but you may use the DAM for managing videos and other files too. Scroll down in the **Backend Response** to locate the **binaryURL** of your asset in the REST API response structure. Hover over the **binaryURL** node, so it will show **binaryURL – Create Response** and then **click** it.

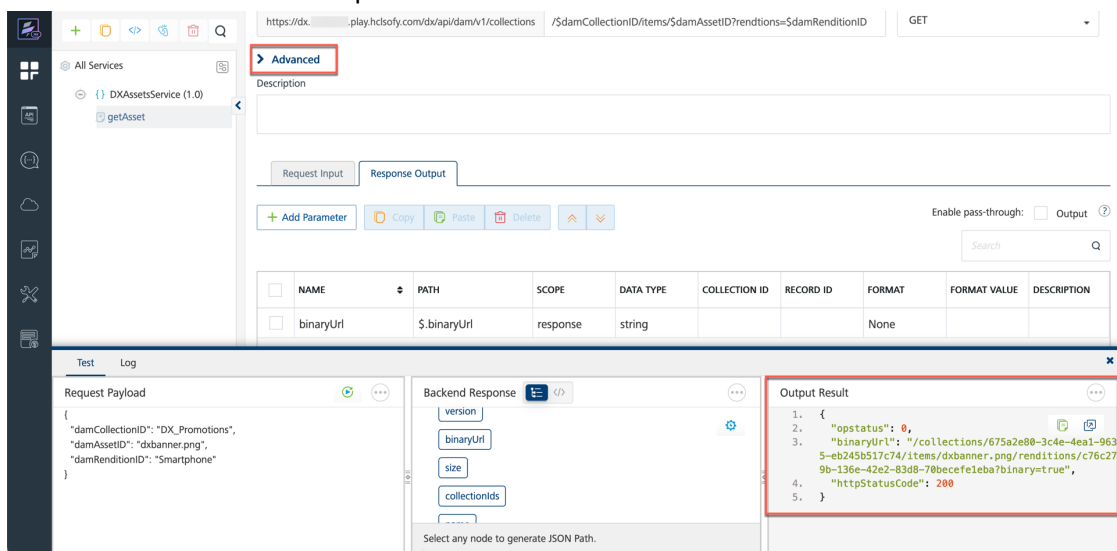


10. This switches you to the **Response Output** and you see the added **binaryUrl** parameter. Now test it again, using the **Save and retest** icon.



NAME	PATH	SCOPE	DATA TYPE	COLLECTION ID	RECORD ID	FORMAT	FORMAT VALUE	DESCRIPTION
binaryUrl	\$.binaryUrl	response	string			None		

11. Now you see the **binaryUrl** added in the **Output Result**. The DAM REST API response does not contain a full URL to the asset, but a relative part that starts after the <host>/dx/api. To be able to directly point to the image, you need to complete this URL. You want to add a post process to this result that adds the host to this output. You do this under the advanced section. Click **Advanced** to expand.



```

1. {
2.   "opstatus": 0,
3.   "binaryUrl": "/collections/675a2e88-3c4e-4ea1-9635-eb245b517c74/items/dxbanner.png/renditions/c76c279b-136e-42e2-83d8-70becefe1eba?binary=true",
4.   "statusCode": 200
5. }

```

12. You can create any JSON structure in the output response by manually entering paths/names/data-types, or even generate completely custom response JSON structures (including fields not present in original backend response, but generated based on custom-business-logic) using [Data Processors](#) (Pre and Post processors). Post processors are used to modify the backend-response using custom business logic. In this case, the custom logic is written in JavaScript and injects a new field into the response. Select **JavaScript** under **Postprocessor** and add the two JavaScript lines below, with your own DX host (change **sbx0000** to your instance):

```
var tmpDXHost = "https://dx.sbx0000.play.hclsofy.com/dx/api/dam/v1";
result.addParam("DXHost", tmpDXHost);
```

Then select **Response Output** and click **Add Parameter** and add **DXHost** as a parameter. Click **Save And Fetch Response** to save your service and to make a test call.

The screenshot shows the HCL Volt MX interface for configuring a service. The 'Postprocessor' section is expanded, and the 'JavaScript' tab is selected. The custom code is pasted into the editor. The 'Response Output' tab is active, and a new parameter 'DXHost' is added to the list. The 'Save And Fetch Response' button is highlighted.

NAME	PATH	SCOPE	DATA TYPE	COLLECTION ID	RECORD ID	FORMAT	FORMAT VALUE	DESCRIPTION
binaryUrl	S.binaryUrl	response	string			None		
DXHost		response	string			None		

13. The **Output Result** now has **DXHost** and **binaryUrl** output parameters. Your Volt Iris project will use both to compose a valid URL to your DAM asset.

The screenshot shows the 'Response Output' tab in Volt Iris. It features a table with the following columns: NAME, PATH, SCOPE, DATA TYPE, COLLECTION ID, RECORD ID, FORMAT, FORMAT VAL..., and DESCRIPTION. Two parameters are listed:

NAME	PATH	SCOPE	DATA TYPE	COLLECTION ID	RECORD ID	FORMAT	FORMAT VAL...	DESCRIPTION
binaryUrl	\$.binaryUrl	response	string			None		
DXHost		response	string			None		

Below the table, the 'Output Result' window is open, showing a JSON response:

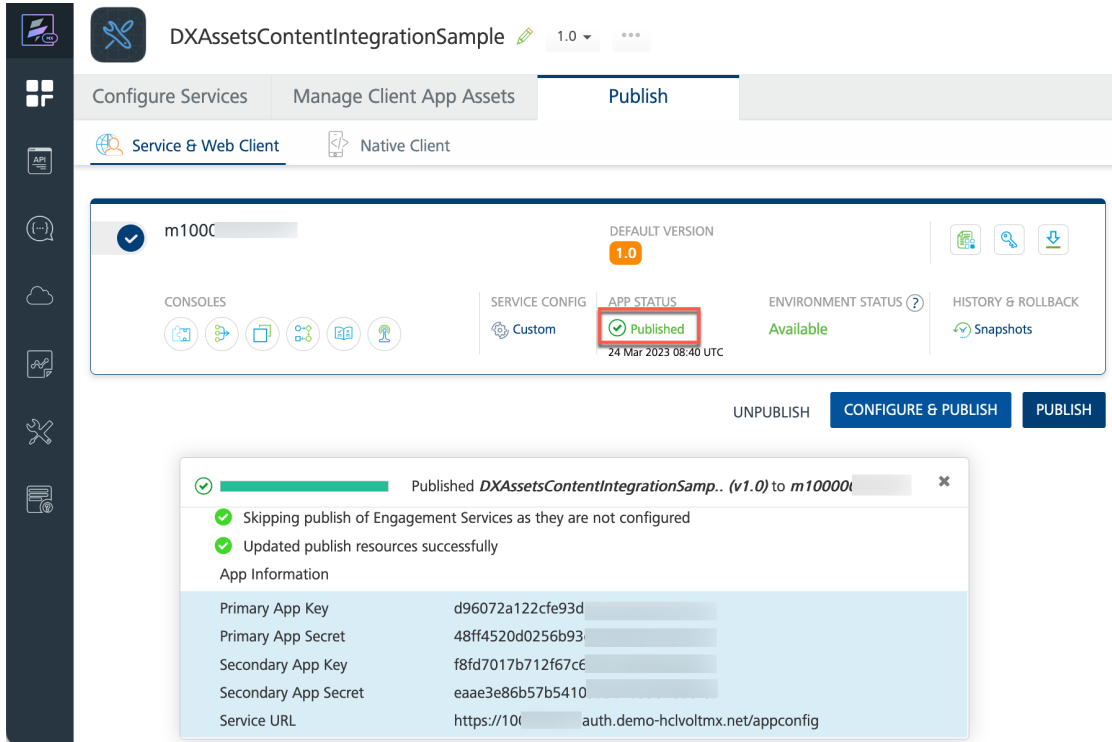
```

1. {
2.   "DXHost": "https://dx. .play .com",
3.   "opstatus": 0,
4.   "binaryUrl": "/collections/675a2e80-3c4e-4ea1-9635-eb245b517c74/items/dxbanner.png/renditions/c76c279b-136e-42e2-83d8-70becefe1eba?binary=true",
5.   "statusCode": 200
6. }
    
```

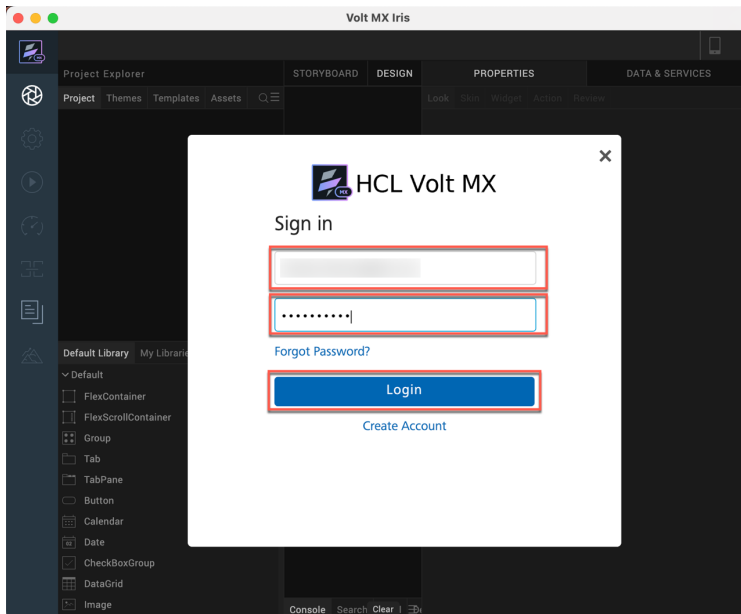
14. You are ready to publish your new Foundry application **DXAssetsContentIntegrationSample** for consumption in Volt Iris. Click **Publish**, select your server and click **Publish**.

The screenshot shows the Foundry application management interface for 'DXAssetsContentIntegrationSample' (version 1.0). The 'Publish' button is highlighted with a red box. Below, the application details for 'm1000' are shown, including a checkmark icon, console logs, service configuration (Custom), app status (Not Published), environment status (Available), and history/rollback options (Snapshots). At the bottom right, the 'CONFIGURE & PUBLISH' and 'PUBLISH' buttons are visible, with the 'PUBLISH' button highlighted by a red box.

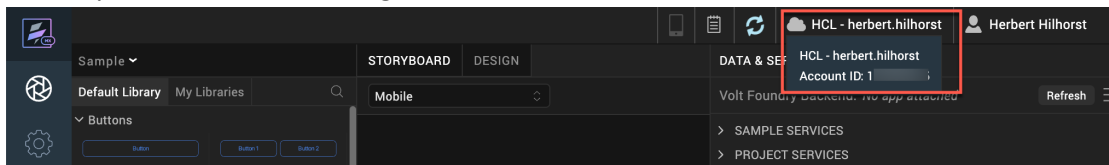
15. And then it gets published and finally shows the application status as published.



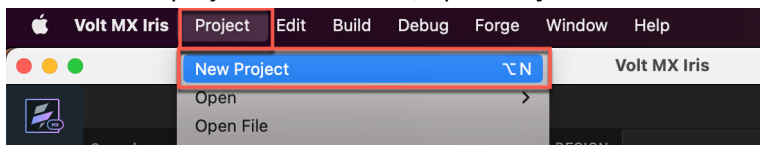
16. Now you will use the Volt Iris IDE to create a web application that displays the DX image resource links, using low-code. First, you will create a linkage to the Foundry service you just created: **DXAssetsService**. Ensure you have set up Volt Iris IDE and have the right credentials. See the prerequisites for instructions on how to set this up. Then launch Volt Iris IDE in your workstation and log in using your Volt MX credentials.



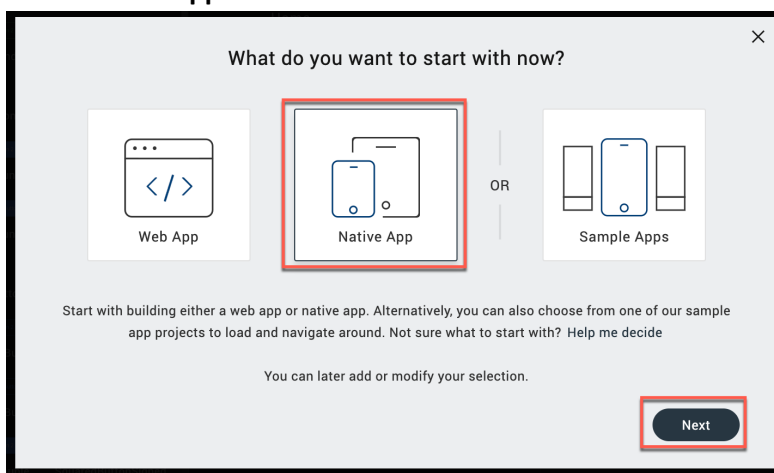
17. Ensure you have selected the right cloud account.



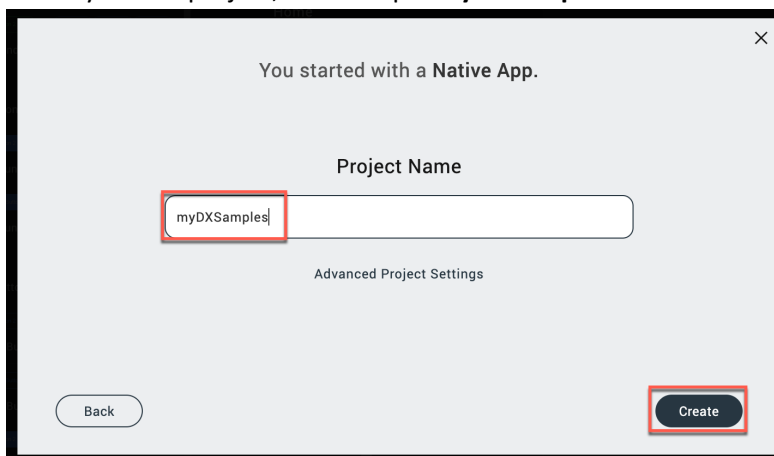
18. Create a new project. In the menu, open **Project** and click **New Project**.



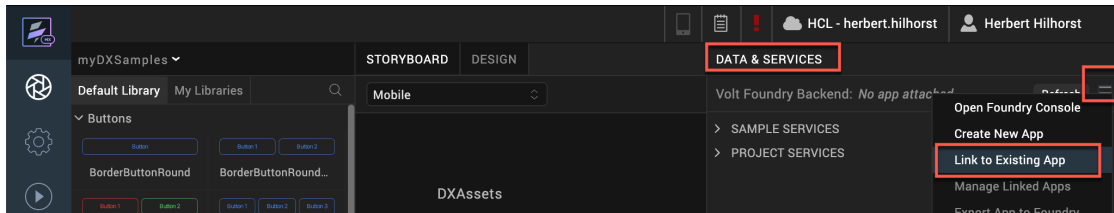
19. Select **Native App** and click **Next**.



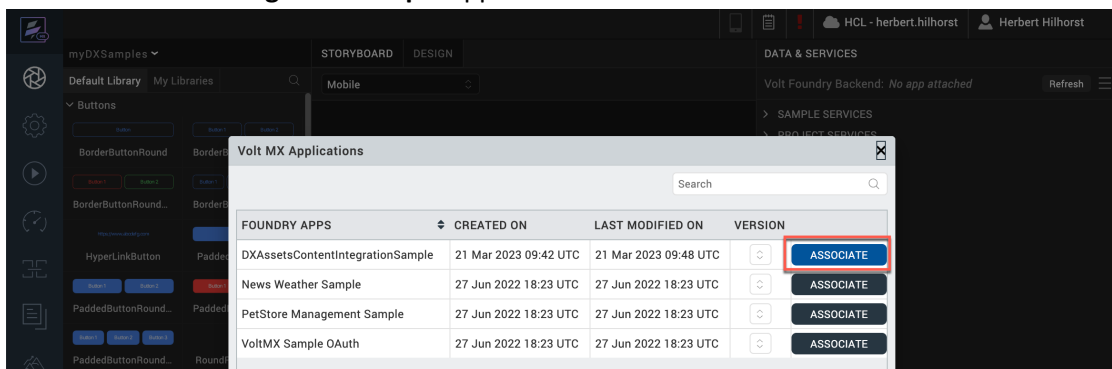
20. Name your Iris project, for example **myDXSamples** and click **Create**.



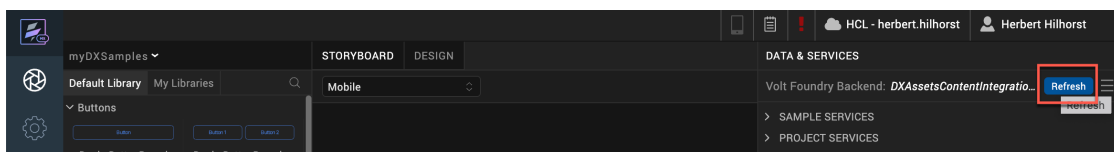
- Link your new Iris application to your Foundry application first. Under **DATA & SERVICES**, to the right of your Volt Foundry Backend application, open the menu and select **Link to Existing App**.



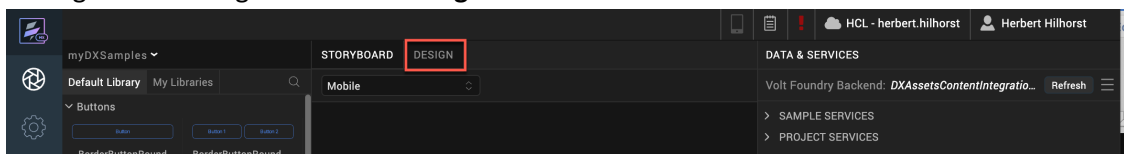
- This shows all the published Foundry applications. Click **ASSOCIATE** for your **DXAssetsContentIntegrationSample** application.



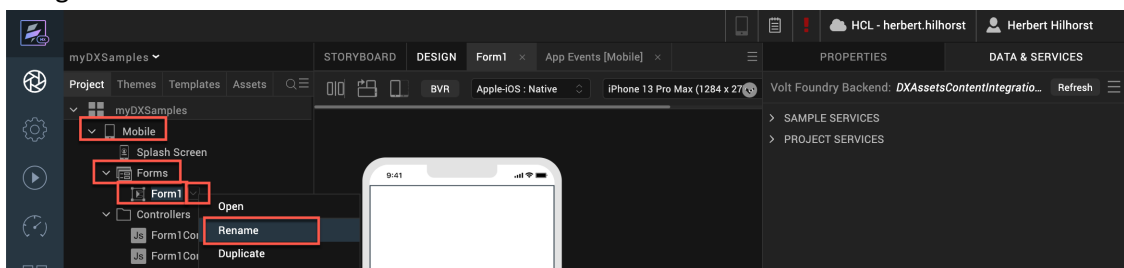
- Then refresh the **DATA & SERVICES** first. Click **Refresh**.



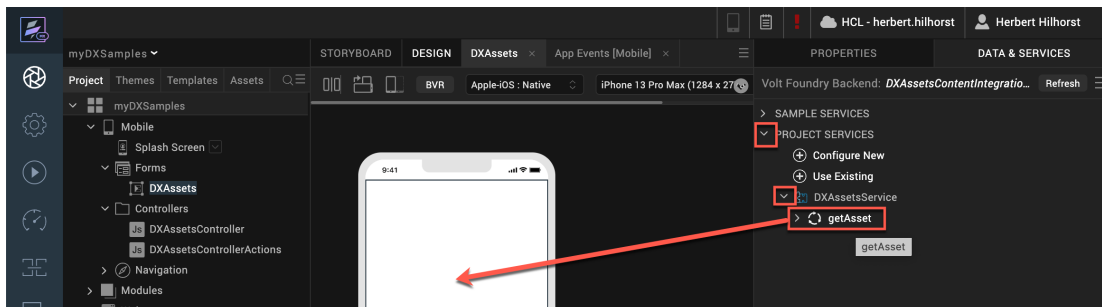
- Then go to the design view. Click **Design**.



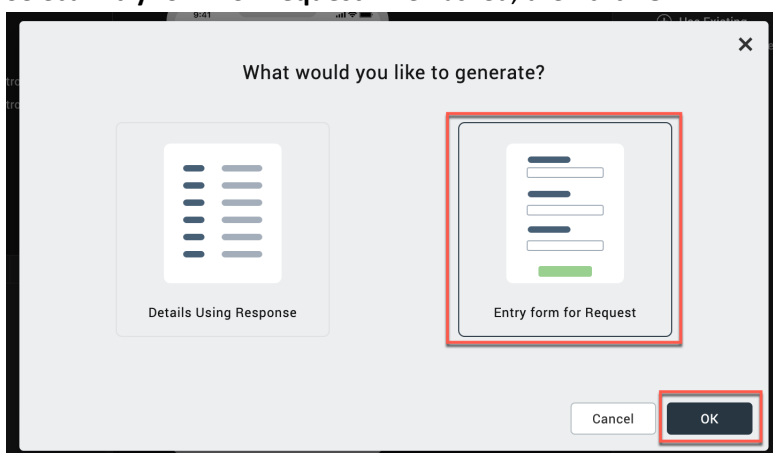
- Expand **Mobile** and **Forms** and you will see a default first form, named **Form1**. Click it and rename it to a better name, e.g. **DXAssets**, using the down-control to the right of **Form1** using **Rename**.



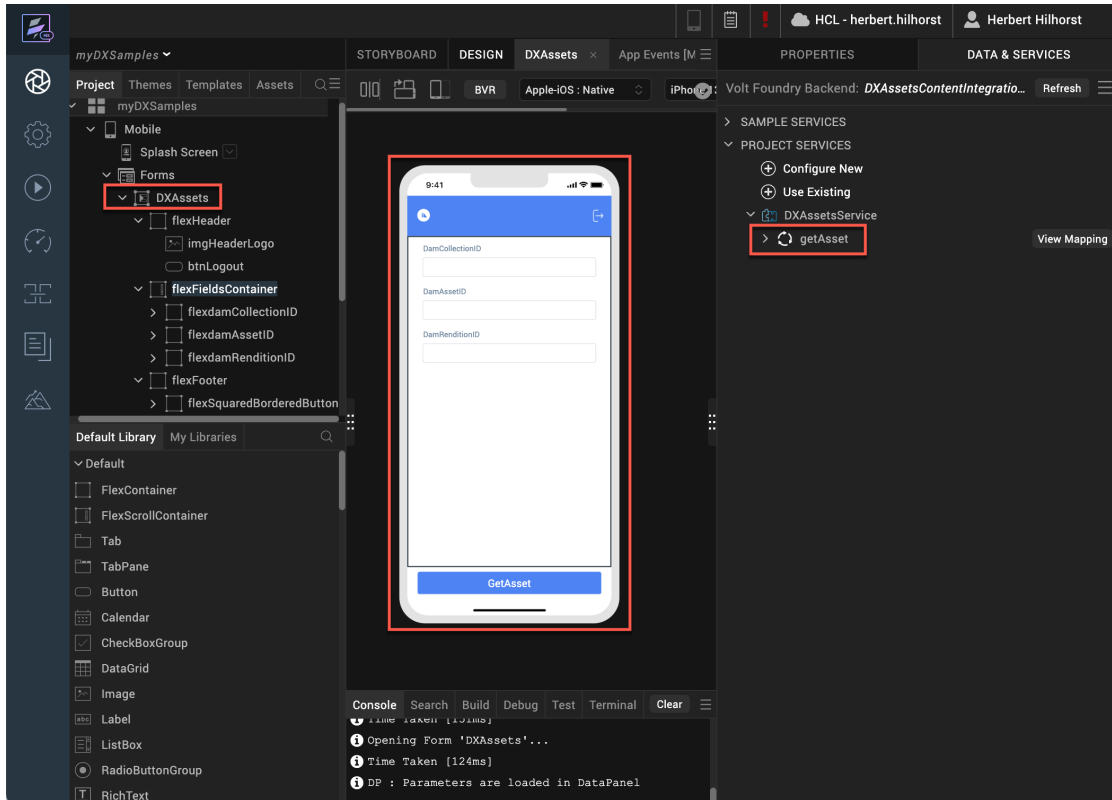
- Under **DATA & SERVICES**, expand the **PROJECT SERVICES** and then expand your **DXAssetService** service and then drag the **getAsset** operation to your newly created form **DXAssets**, as shown.



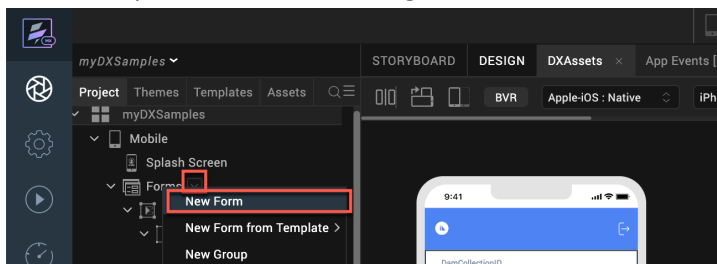
- Select **Entry form for Request** when asked, then click **OK**.



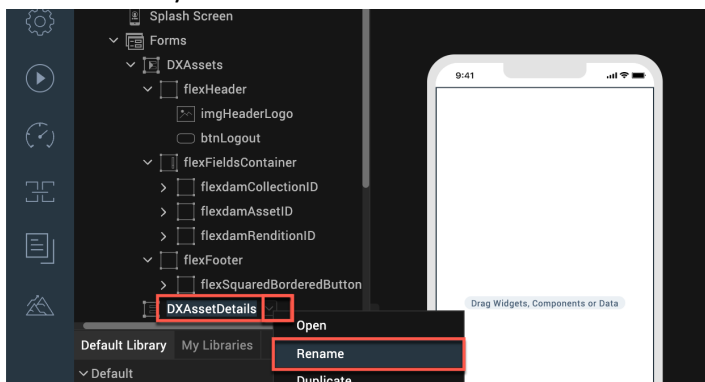
28. You should see **DXAssets** updated, generated via Iris low-code capability, with a UI to input the DAM Collection ID, DAM Asset ID and Rendition ID, along with a button **GetAsset** at the bottom. You may change the labels and button texts by selecting the items and modifying the properties. You could also hide the DamRenditionID and generate this automatically for the device using the asset. To keep this lab simple, leave it as it is.



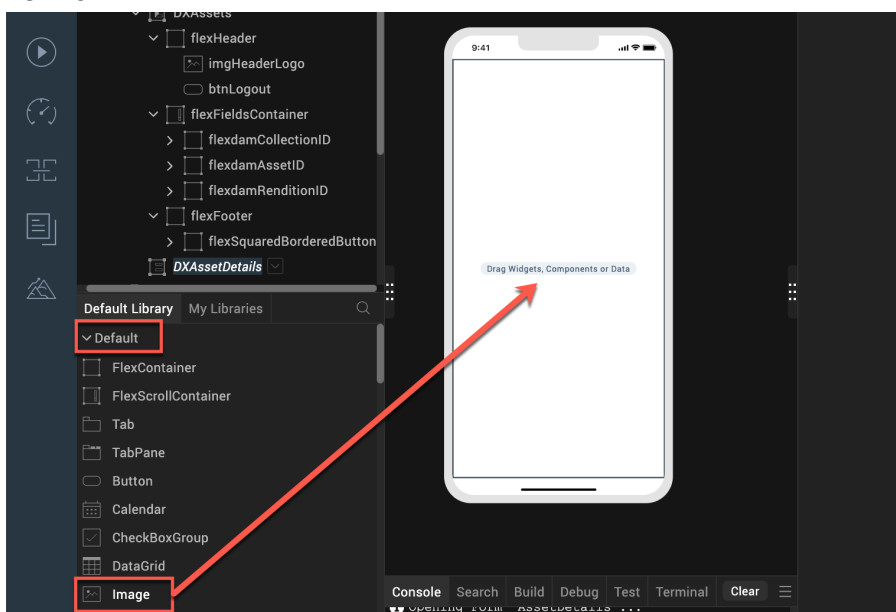
29. Now you want to add a new view to render your asset. Add a new form to your Iris mobile channel. Open the menu to the right of Forms and click **New Form**.



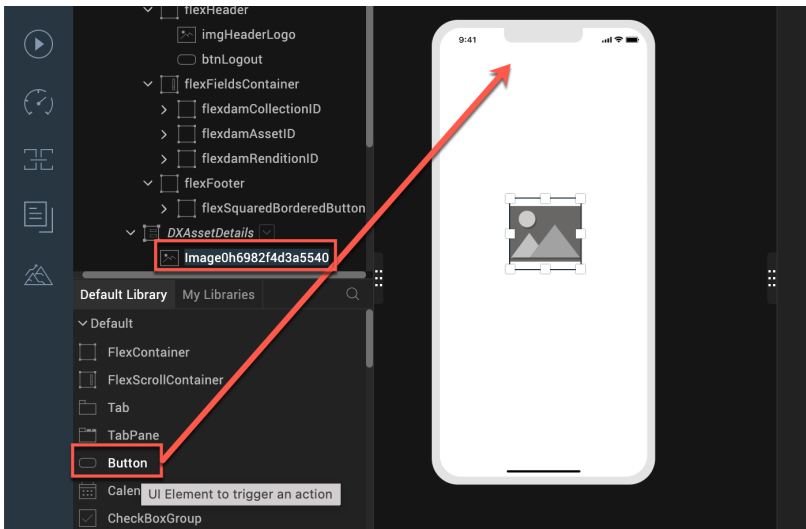
30. Then rename your new form to **DXAssetDetails**.



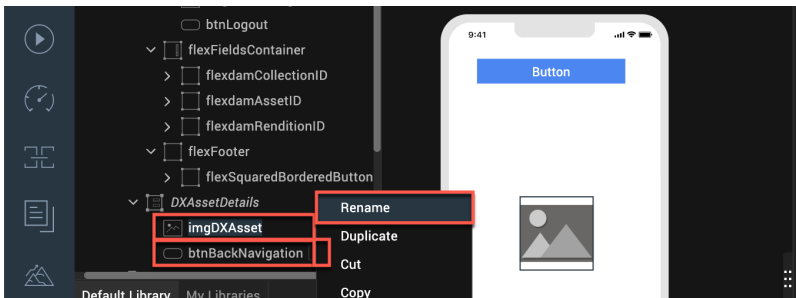
31. You are focusing on image assets for now. To render the asset, you will add an image widget to your new form. From the **Default Library, Default** widgets, drag the **Image** widget to your new form.



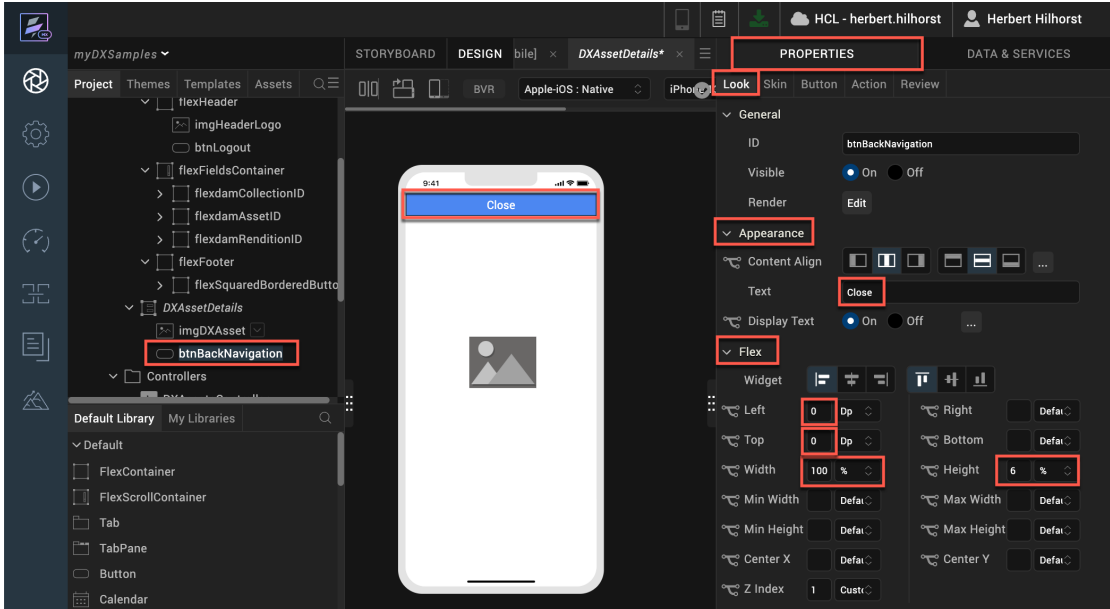
32. Your image widget now shows under your new form. Now add a button that allows you to go back to the previous DXAssets form. Drag the **Button** widget to the top of your new form.



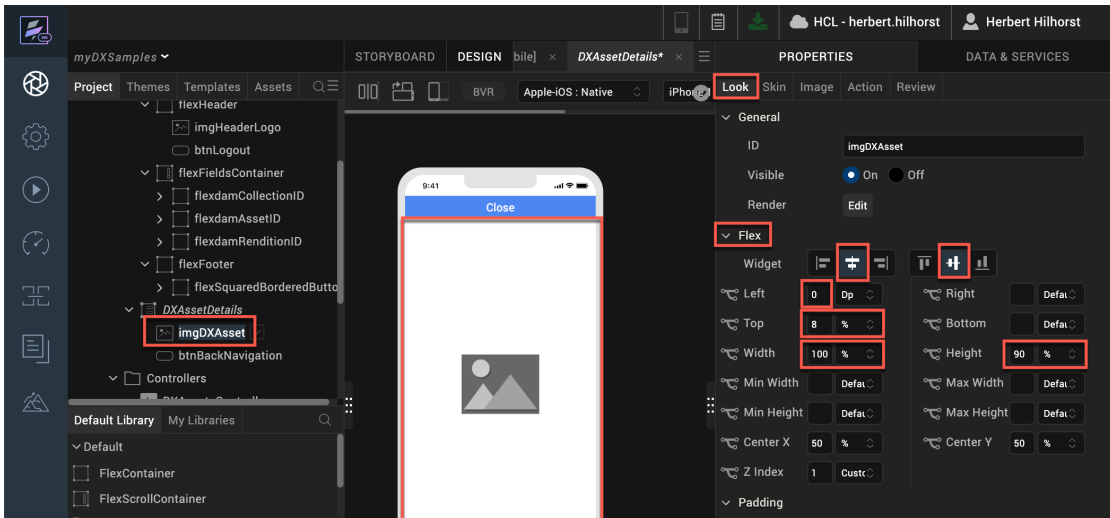
33. You now see your new button. Similar as with your forms, rename these widgets to a more easily understandable name, like **imgDXAsset** for your image and **btnBackNavigation** for your button.



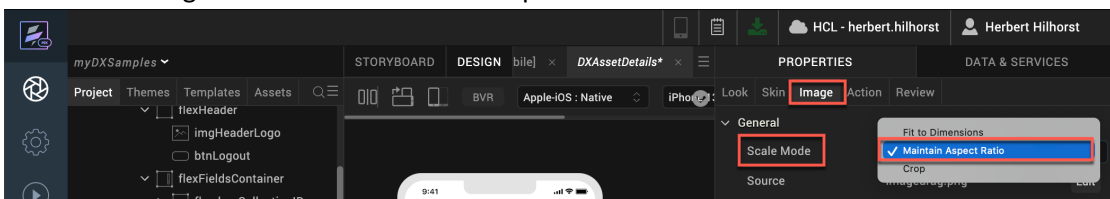
34. Set the properties of your button to appear at the top of the screen and occupying the entire width of the viewport. To do this, select the button from the project explorer or click on the button in the canvas, then select **PROPERTIES**. Now change, under **Look** and **Appearance**, the button **Text** to **Close**, and under **Flex**, the **Left** and **Top** margins to **0 Dp** (Density independent pixels), the **Width** to **100 %** and set your Height to **6 %**. It would look like this.



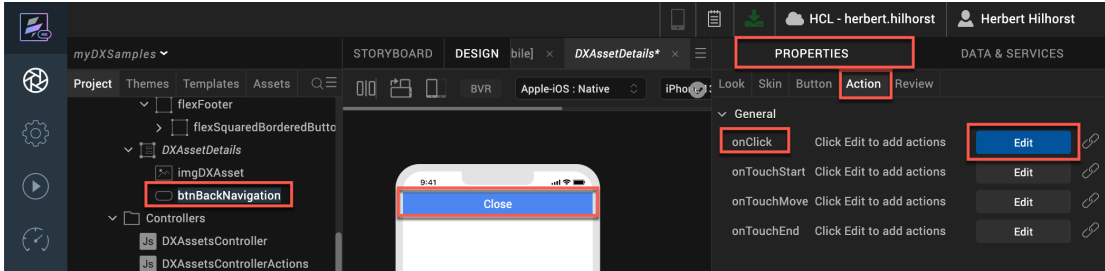
35. Now change the image. Select it and then under **Look**, change the **Flex** settings to pin the **Widget** to the **Center** and **Middle**, set **Left** to **0 Dp**, **Top** to **8 %**, **Width** to **100 %** and **Height** to **90 %**.



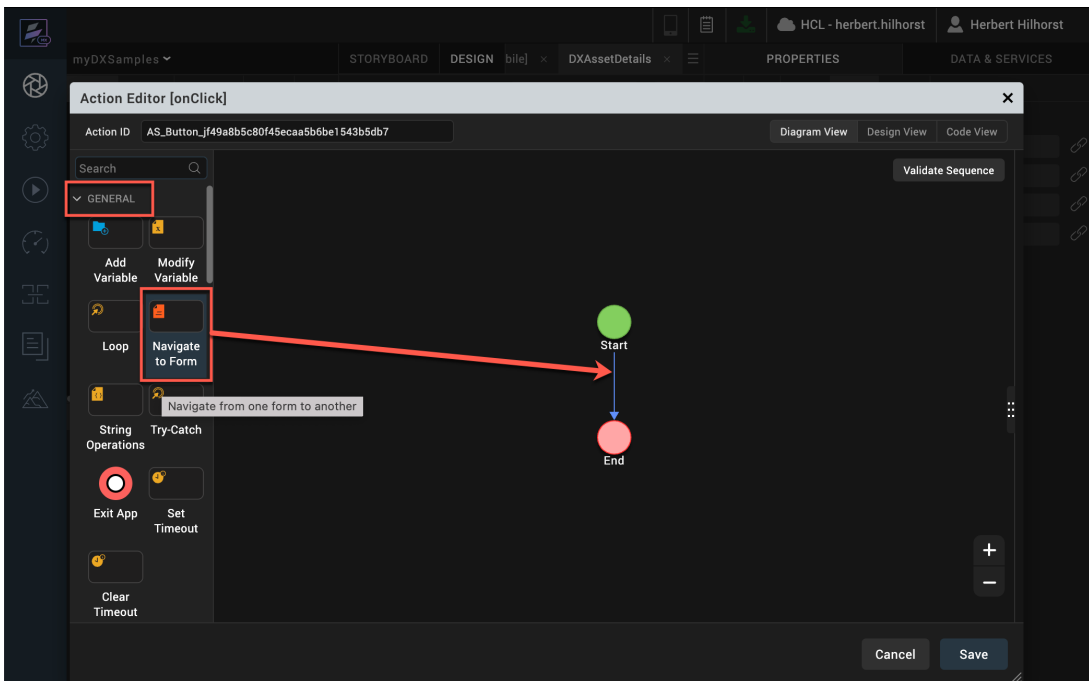
36. Then go to the **Image** settings and ensure the **Scale Mode** is set to **Maintain Aspect Ratio** to ensure the image is rendered in its initial aspect ratio.



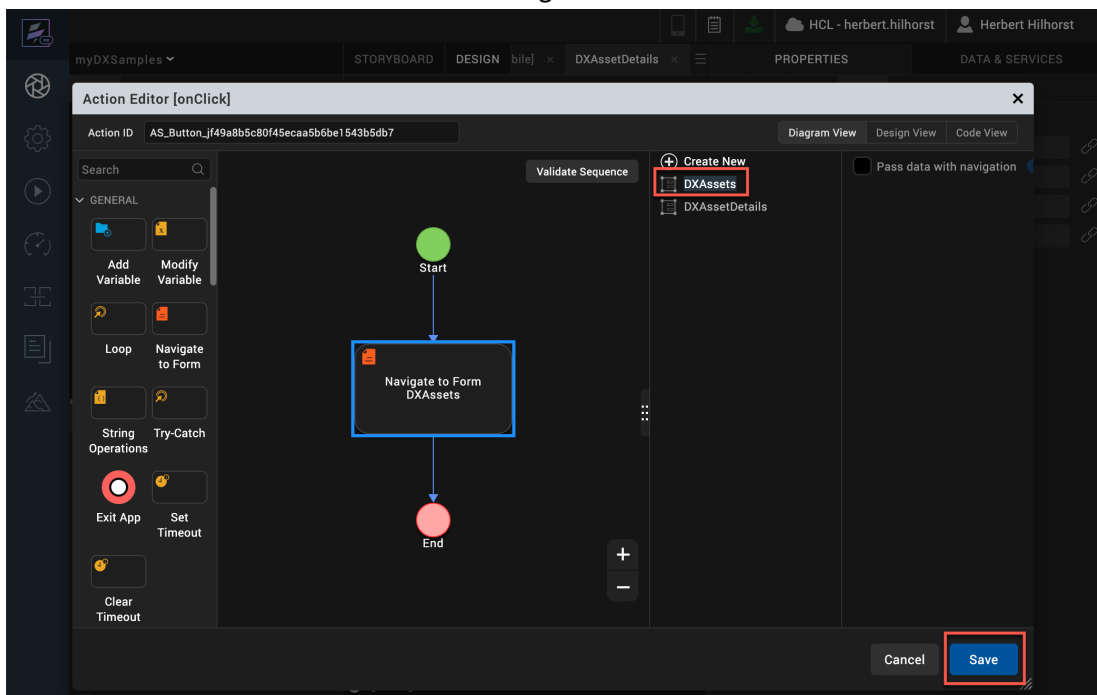
37. You want your users to use the Close button to go back to the DXAsset form. Therefore, you will use an Iris generated low-code action script for the onClick event of your btnBackNavigation button. Ensure the focus in on the **btnBackNavigation** button. Select **PROPERTIES - Action** and click Edit button next to the **onClick** event.



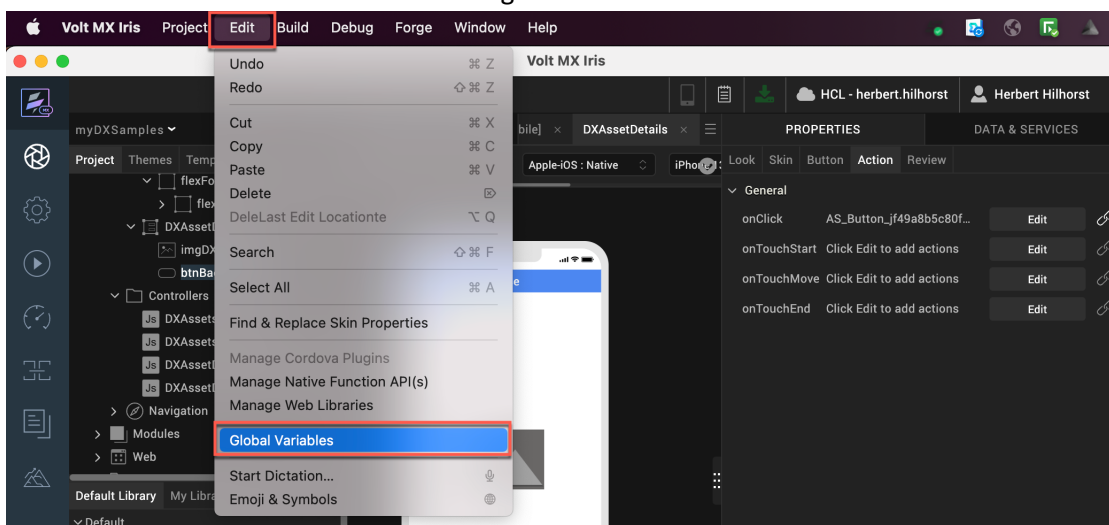
38. The Action Editor shows and from the action palette, drag the **General - Navigate to Form** action to the flow-line between **Start** and **End** in the action tree.



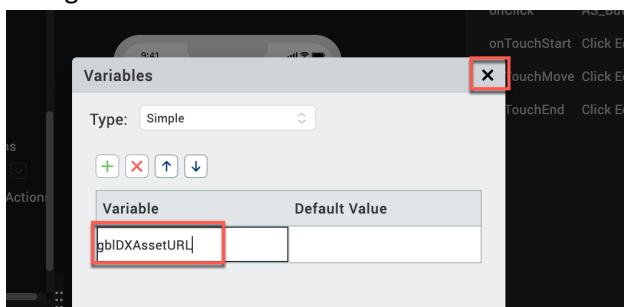
39. Then select the **DXAssets** form it should navigate to and click **Save**.



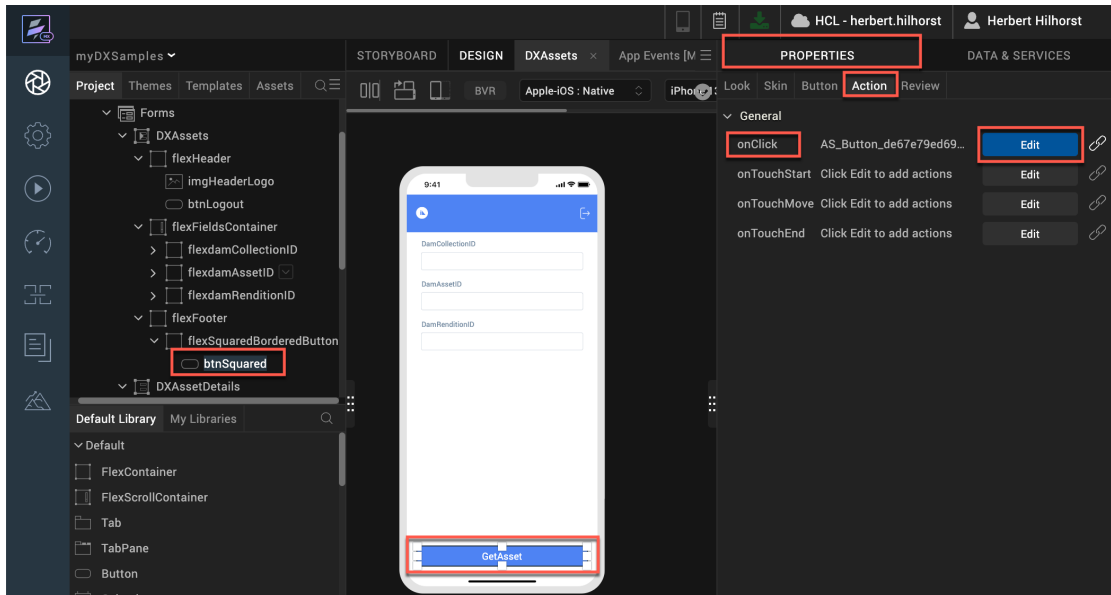
40. To display a DX image, you will need the fully qualified DX asset URL. You will use a global variable to hold the DX URL. Add this using the menu **Edit – Global Variables**.



41. Keep it as a **Simple Type**, name it **gbIDXAssetURL** to hold the DX asset URL and close the dialog box.



42. You will set this variable when your user clicks the GetAsset button on your DXAssets form and then open your DXAssetsDetails form. Select your **DXAssets** form and **GetAsset** button (it may have a different default ID), then **PROPERTIES**, **Action** and click **Edit** for the **onClick** event.

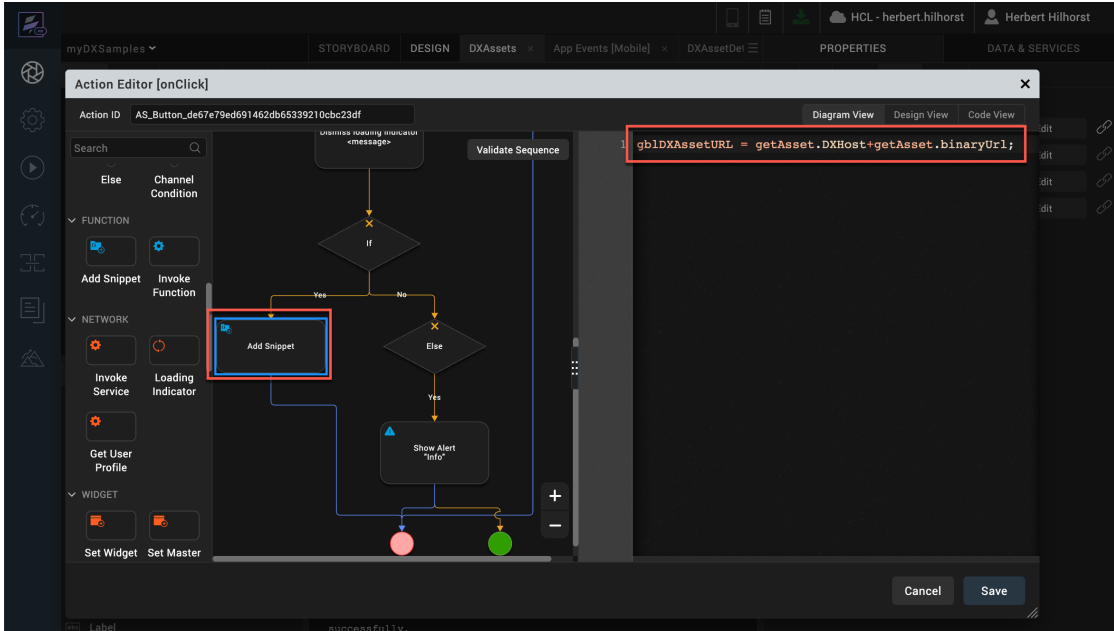


43. In the Action Editor you will see the auto-generated action flow for the click event of the button. You will notice that this covers the failure scenario for the service call Show Alert "Info" stating that the service has failed. You will need to create the action flow for the success condition. Here, if the service succeeds, you will initialize the global variable you just created and then navigate to the next form DXAssetDetails. Drag the **Add Snippet** action under **FUNCTION** to the **If** node attached to the service Callback in the script tree. The **If** node currently only has the **No** and **Else** branch. Drop the **Snippet** next to the **If** node when you see the "blue" guiding arrow.

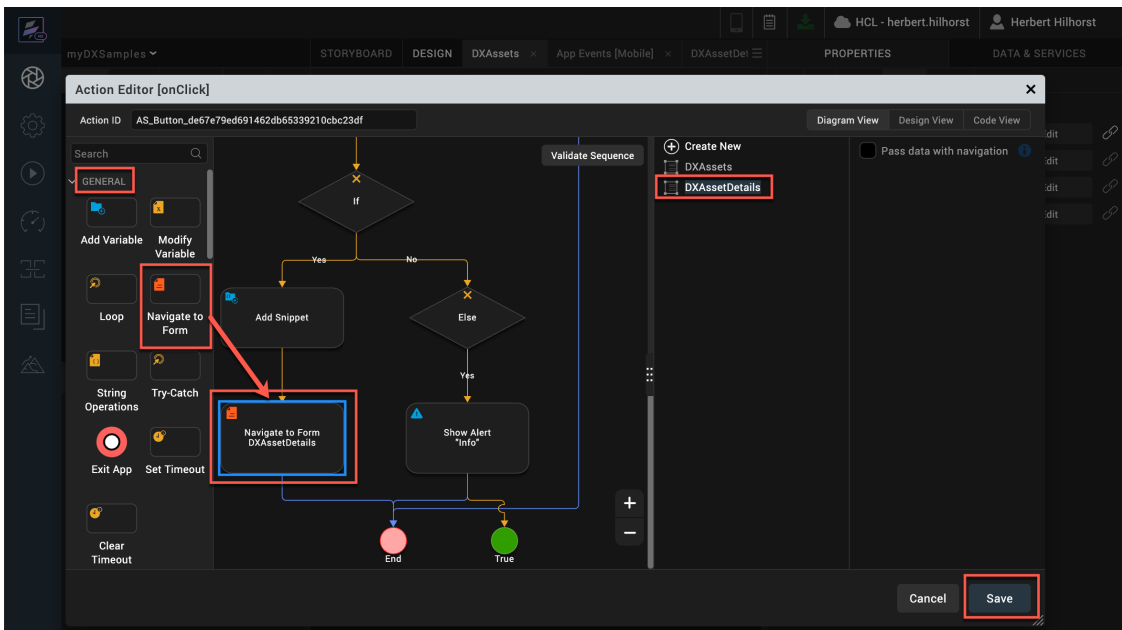


44. Then add the JavaScript line below that generates the global variable to the fully qualified URL.

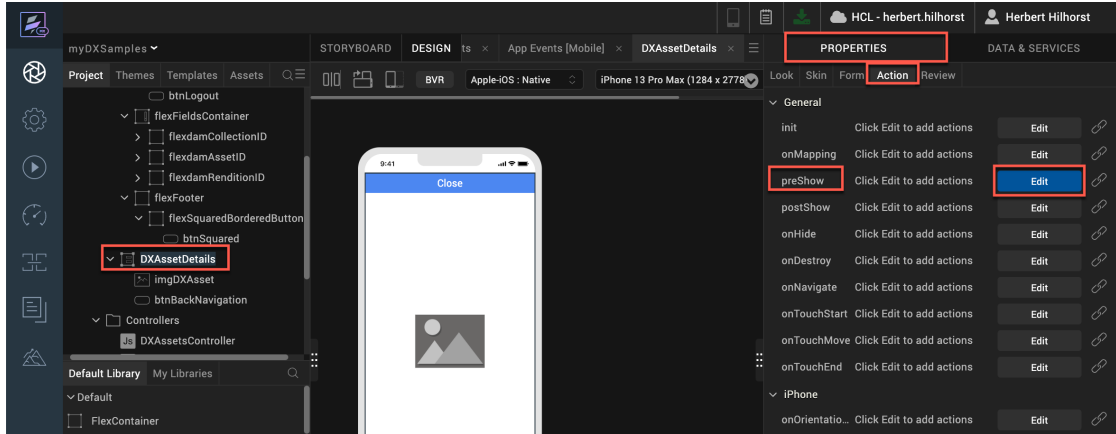
```
gblDXAssetURL = getAsset.DXHost+getAsset.binaryUrl;
```



45. Then add the navigation to the DXAssetDetails form. Add the **GENERAL – Navigate to Form** between the **Add Snippet** and **End** using drag & drop, select the **DXAssetDetails** form and click **Save**.

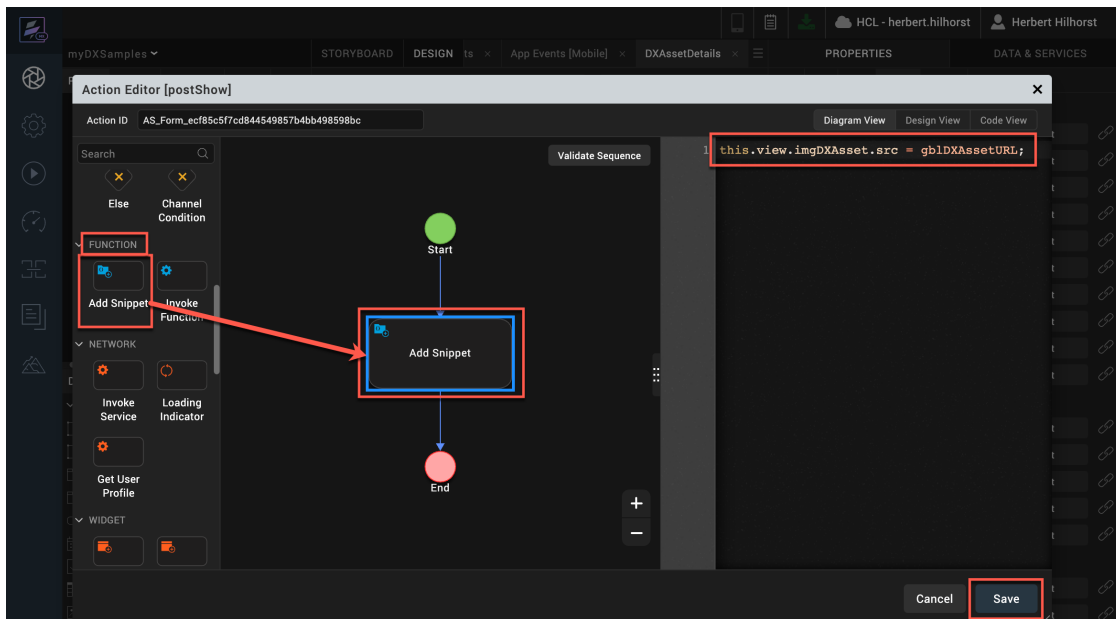


46. Then you need to use this URL in the global variable in your DXAssetDetails form. Select your **DXAssetDetails** form. Then under **PROPERTIES – Action** click the **Edit** button for the **preShow** event.

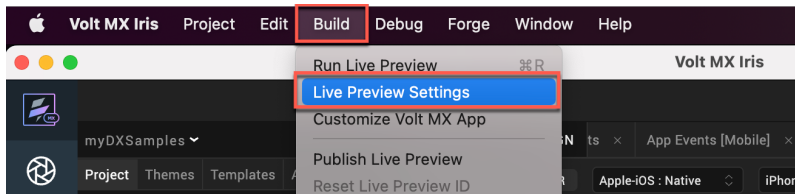


47. Then add the **Add Snippet** between **Start** and **End** and add the code that sets the source of your **imgDXAsset** image widget to your global URL **gblDXAssetURL**. You do that with the following JavaScript. Then click **Save**.

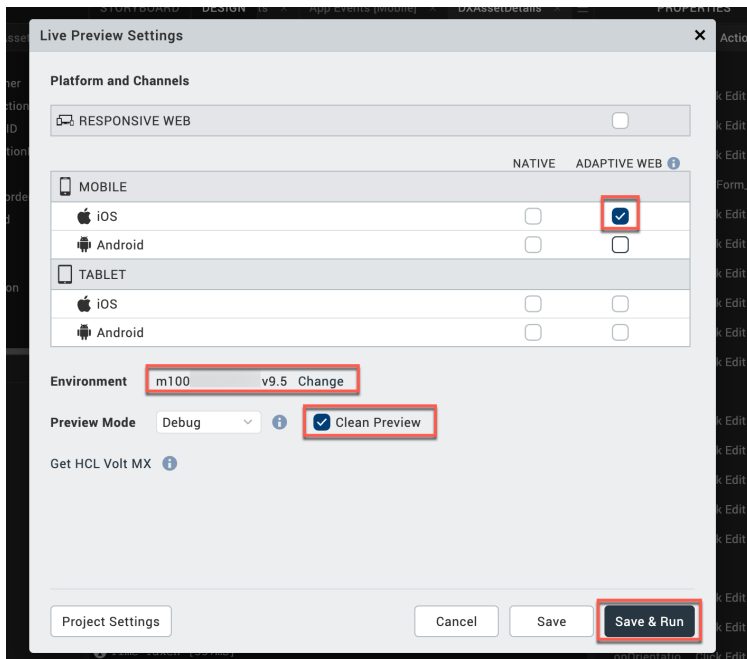
```
this.view.imgDXAsset.src = gblDXAssetURL;
```



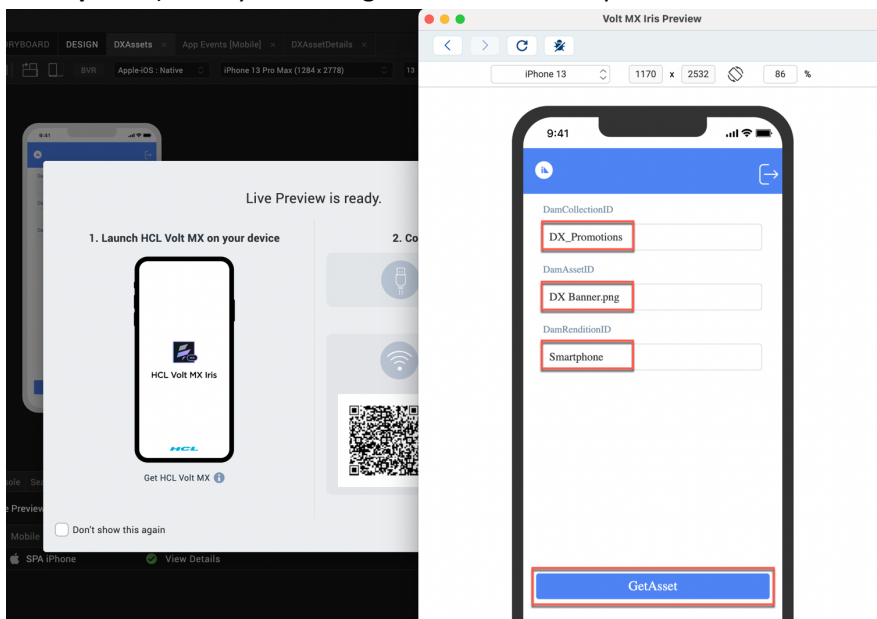
48. You are now ready to build and test your mobile channel application. Click **Build** -> **Live Preview Settings**.



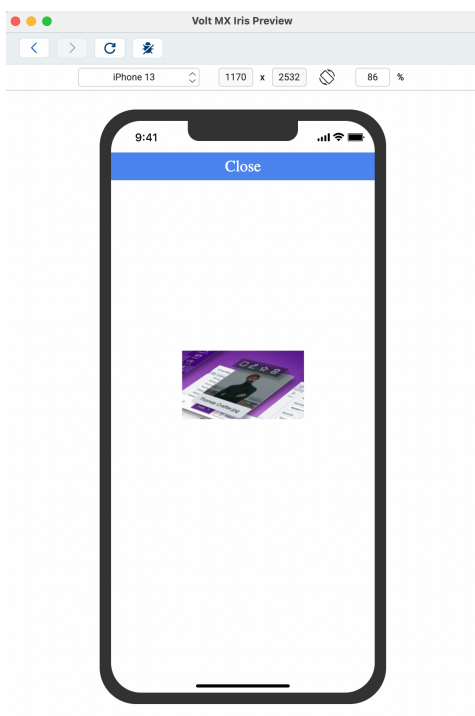
49. Then check any **ADAPTIVE WEB** option that matches your favorite mobile device, for example **iOS**. Then ensure you have the same environment selected as the one where you published your Foundry application. Select **Clean Preview** to clear build caches and click **Save & Run**.



50. You should see your application running as an iPhone SPA. Provide a valid **DAM Collection ID** (e.g. **DX_Promotions**), **DAM Asset ID** (e.g. **DX Banner.png**) and **DAM Rendition ID** (e.g. **Smartphone**) from your DX Digital Assets to the input text boxes. Click the button **GetAsset**.



51. You should see your DAM asset. Your business user can easily update these assets with a user-friendly interface. Feel free to close and try with a different collection, asset ID and rendition.



Congratulations! You have successfully developed, built, and ran your Volt MX mobile application using the Volt MX low-code capability that renders a DX asset, managed by business users.

Part 2: Use DX Content in a new HCL Volt MX Web Application

In this part, you will learn how to develop, build, and run a new Volt MX web application that uses your HCL Digital Experience content. To facilitate communication between the DX backend content and Foundry, you will create integration services in Foundry that will search DX content, based on a Content Template ID and Parent ID (Site Area) as input to the service, and a service to get details on a single content. You will also learn to use the HCL Digital Experience Content Adaptor that is available on the Volt MX Marketplace. Then you will build your web application with Iris that uses these services. It allows you to select a ContentTemplateID and ParentID and show the titles and IDs of the content that correspond.

And when selecting a product, you will see the details.

1. First select **Apps** in Foundry and click your **DXAssetsContentIntegrationSample** Foundry app you built in **Part 1**.

The screenshot shows the Foundry Apps page. The 'Foundry Apps' tab is active. A sidebar on the left contains navigation icons, with the 'Apps' icon highlighted. The main content area shows a list of apps. The 'DXAssetsContentIntegrationSam...' app is highlighted with a red box. Below it, a table shows the app's details:

Version	Modified By	Modified On
1.0	Herbert Hilhorst	21 Mar 2023 09:48 UTC

Other apps visible include 'VoltMX Sample OAuth'.

2. Then create a new integration service. Select **Integration, CONFIGURE NEW**.

The screenshot shows the configuration page for the 'DXAssetsContentIntegrationSample' app. The 'Integration' tab is selected and highlighted with a red box. Below the tab, there are two buttons: 'CONFIGURE NEW' (highlighted with a red box) and 'USE EXISTING'. A table below shows the list of services:

NAME	SERVICE TYPE	VERSION
<input type="checkbox"/> DXAssetsService	JSON	1.0

3. Then **name it DXContentService** and select **JSON Service Type**. Now configure this integration service with your DX SoFy instance. In the DX Integration Introduction, you have seen the structure of the REST API calls to get DX content. Assign the **Base URL** to the web content REST API URL for your DX SoFy instance:
<host>/dx/api/core/v1/<access_type>/webcontent, for example,
<https://dx.sbx0000.play.hclsofy.com/dx/api/core/v1/dxrest/webcontent> (replace 0000 by the one of your instance). You have anonymous access to your content with the REST API, so you can use **Web Service Authentication - None**. If you would use protected content, you need use **dxmyrest** and **Basic** Web Service Authentication using the **User ID** and **Password** of a user that has access to your content provided in your SoFy instance. Then click **SAVE & ADD OPERATION**.

The screenshot shows the configuration interface for a service definition in HCL Volt MX. The interface is titled "DXAssetsContentIntegrationSample" and includes tabs for "Configure Services", "Manage Client App Assets", and "Publish". The "Integrations" tab is active, showing a list of services on the left: "NewService (1.0)" and "DXAssetsService (1.0)". The main configuration area is titled "Service Definition" and contains the following fields:

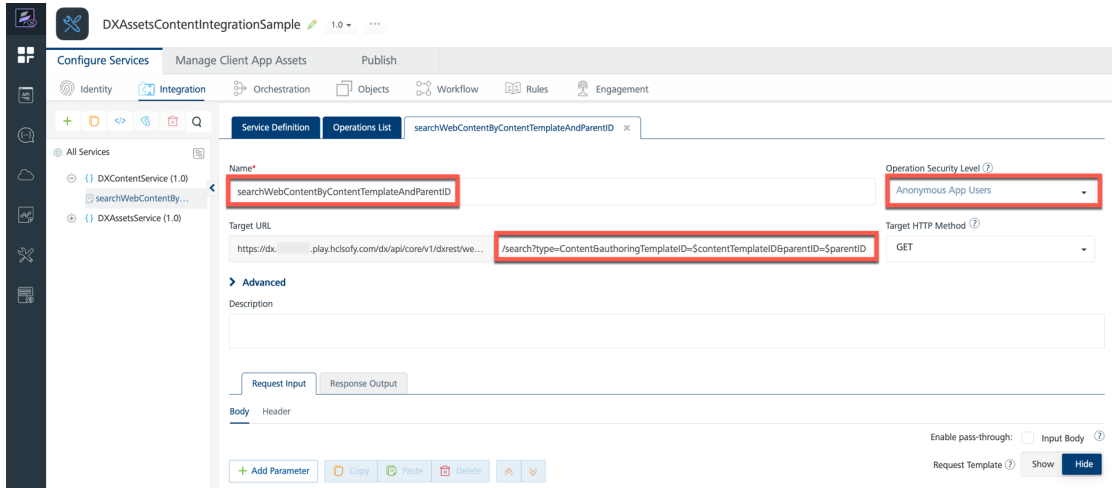
- Name***: DXContentService
- Service Type**: JSON
- Path Expression**: JSON Path
- Version**: 1.0
- Base URL***: https://dx.sbx0000.play.hclsofy.com/dx/api/core/v1/dxrest/webcontent
- Web Service Authentication**: None (selected), Basic, NTLM
- Identity Service for Backend Token**: None
- Advanced**: Description field (empty)

At the bottom right, there are three buttons: "CANCEL", "SAVE", and "SAVE & ADD OPERATION".

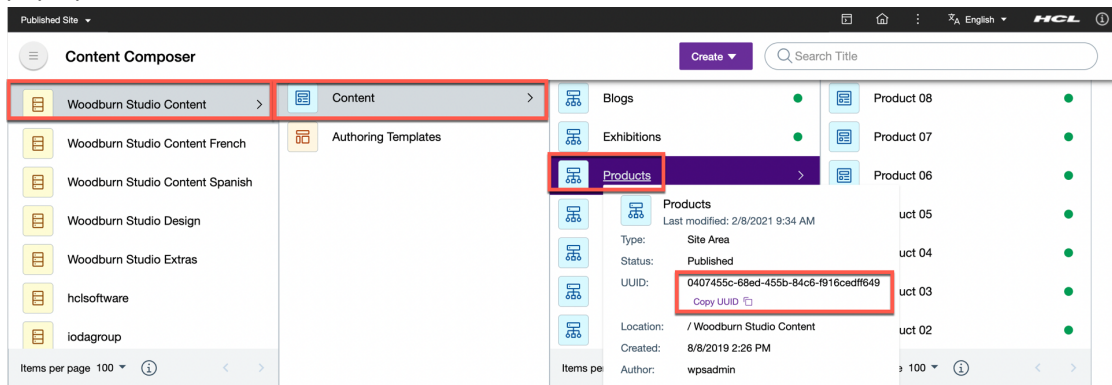
- You first want to add an operation that gives you the list of product contents based on a specific Content Template and in a specific Site Area (using its parent ID) in DX. Give your new operation a name that reflects this, e.g.

searchWebContentByContentTemplateAndParentID. Set the **Operation Security Level** to **Anonymous App Users** and set the target URL to the search API:

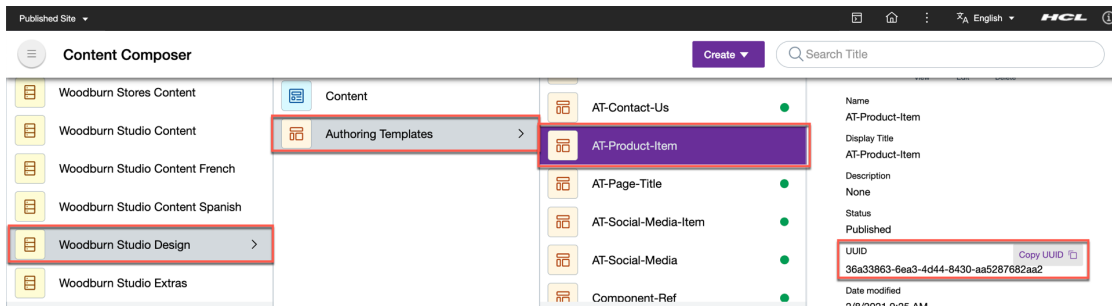
/search?type=Content&authoringTemplateID=\$contentTemplateID&parentID=\$parentID



- As done in the previous part for Assets, we should define the two parameters of the REST API URL, and we should provide some default value to them. In your DX instance, look up the UUID of the Site Area (parent ID), e.g. for Woodburn Studio Content – Products for all the product content items: **0407455c-68ed-455b-84c6-f916cedff649**. Hover over it to have the pop up show.



- Also look up the UUID of a Content Template, e.g. for Woodburn Studio Design - AT-Product-Item: **36a33863-6ea3-4d44-8430-aa5287682aa2**.

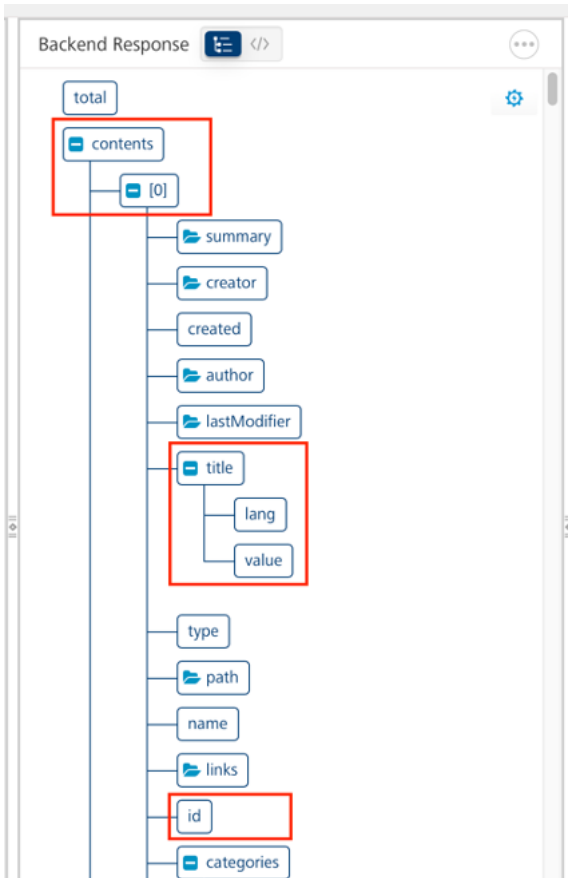


- And set these as the Default Value for a new request parameters **contentTemplateID** and **parentID** using **Add Parameter**. Then click **SAVE AND FETCH RESPONSE**.

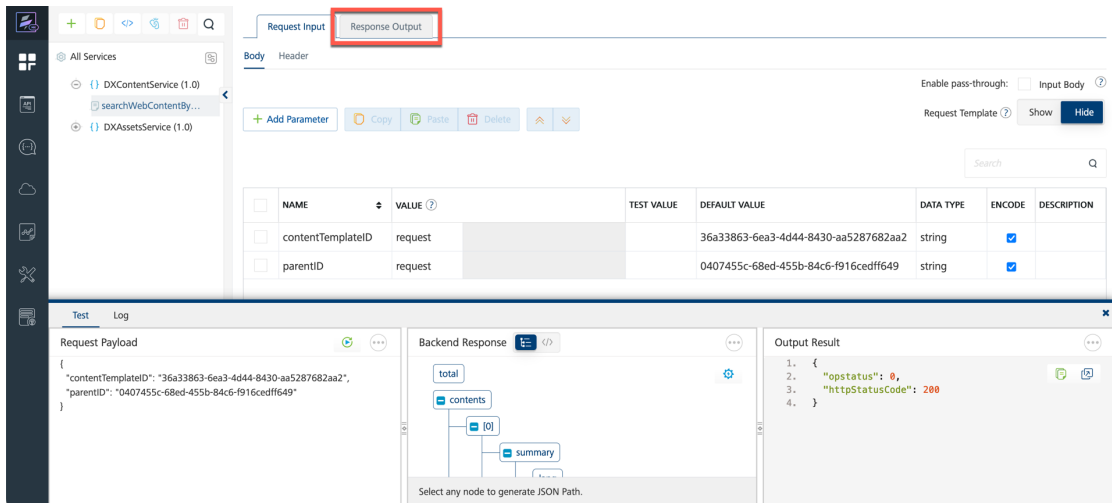
NAME	VALUE	TEST VALUE	DEFAULT VALUE	DATA TYPE	ENCODE	DESCRIPTION
contentTemplateID	request		36a33863-6ea3-4d44-8430-aa5287682aa2	string	<input checked="" type="checkbox"/>	
parentID	request		0407455c-68ed-455b-84c6-f916cedff649	string	<input checked="" type="checkbox"/>	

Default value will be used if Test value is empty. m100000625002 **SAVE AND FETCH RESPONSE** CANCEL SAVE OPERATION

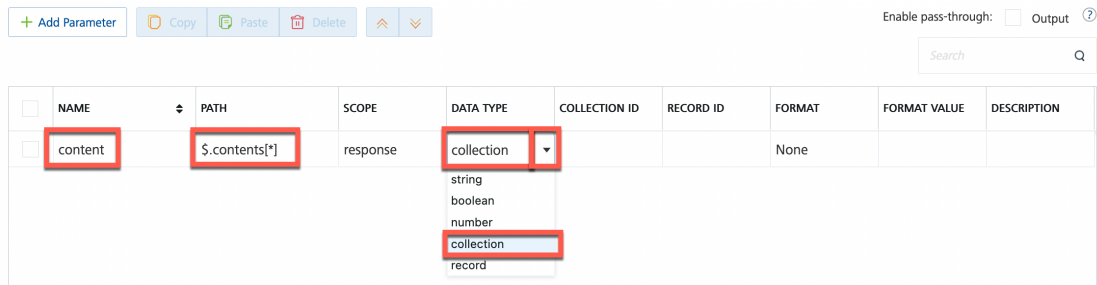
- Check the Output Result and Backend Response. If you have a 401, you may not have correctly set up the security of the virtual resource WCM REST SERVICE, under the Resource Permissions where you should have edit access to. You may refer to HDX-DEV-100 Experience API lab to configure this properly. Then you want to add the **title** and **id** of each product to your output result. Have a look at the **Backend Response** and you will see that the title is addressed with **contents[*].title.value** and the id with **contents[*].id**.



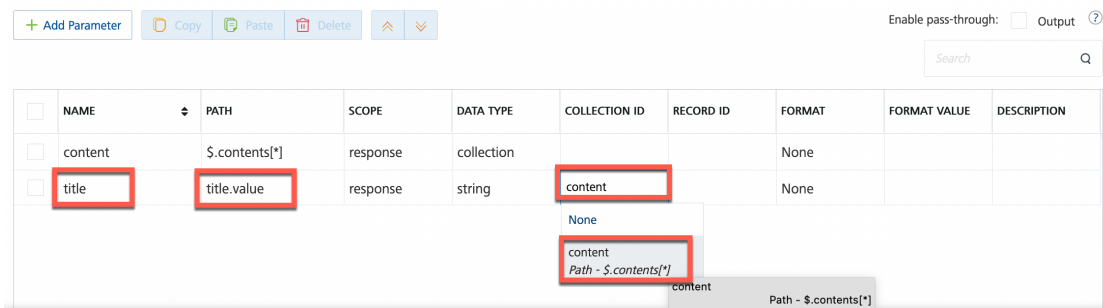
- In this case, you will manage these parameters under the Response Output directly. Click **Response Output**.



- First add the **content** parameter and set it to **\$.contents[*]** and change this to **Collection**. This will make it become a list in the output.



- And then add a **title** parameter, **name** it **title**, set the **PATH** to **title.value** and set the **COLLECTION ID** to the collection you just created: **content**.



12. And add the **id** parameter, **name** it **id**, set the **PATH** to **id** and again select **content** as the **COLLECTION ID**. Then click **SAVE AND FETCH RESPONSE**.

NAME	PATH	SCOPE	DATA TYPE	COLLECTION ID	RECORD ID	FORMAT	FORMAT VALUE	DESCRIPTION
content	\$.contents[*]	response	collection			None		
title	title.value	response	string	content		None		
id	id	response	string	content		None		

Default value will be used if Test value is empty. m10000 **SAVE AND FETCH RESPONSE**

13. Your **Output Result** contains the list of content title and id. Now add an operation to get an individual content. Click **Operations List**.

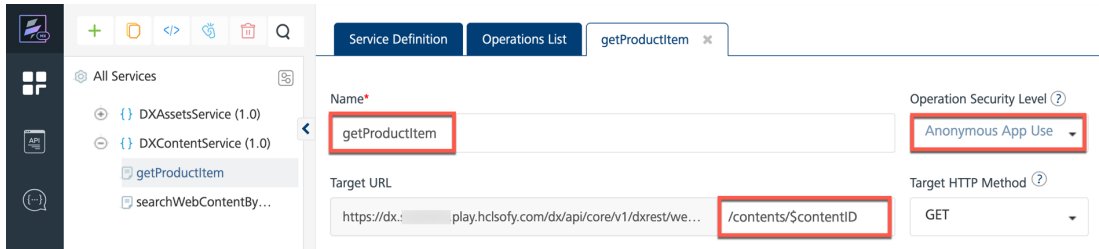
```

1. {
2.   "opstatus": 0,
3.   "content": [
4.     {
5.       "id": "6c5abb2b-bdbd-461c-97ba-43c5994ed3f9",
6.       "title": "Product 05"
7.     },
8.     {
9.       "id": "f4437e0b-6bf2-4e5f-939f-9e841781c0d7",
10.      "title": "Product 08"
11.     }
12.   ]
13. }
    
```

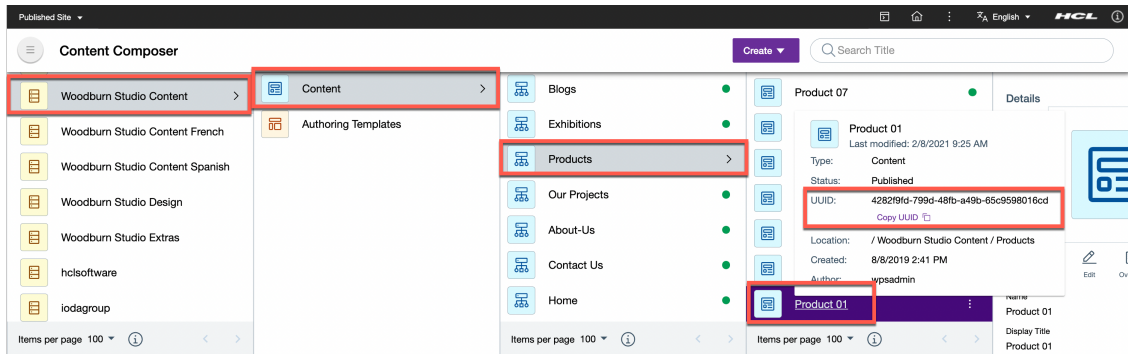
14. Click **ADD OPERATION**.

NAME	URL	MODIFIED BY	MODIFIED ON
searchWebContentByC...	https://dx.pla...	Herbert Hilhorst	06 Apr 2023 09:07 UTC

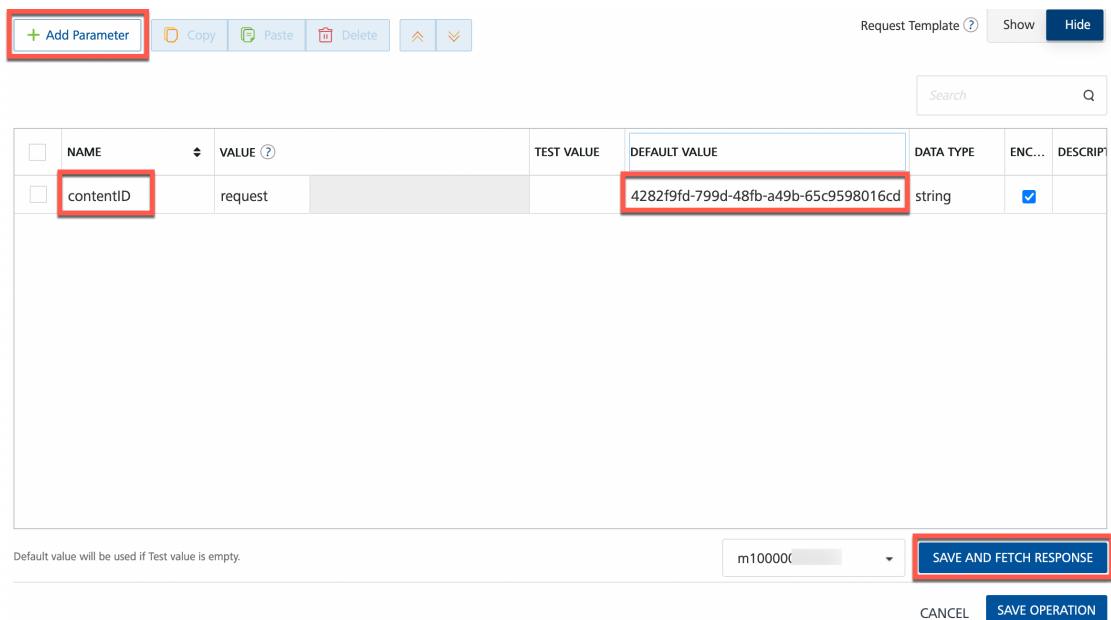
15. Name it **getProductItem** as it is dedicated to your Content Template, set the **Operation Security Level** to **Anonymous App Users** and set the Target URL to **/contents/\$contentID**.



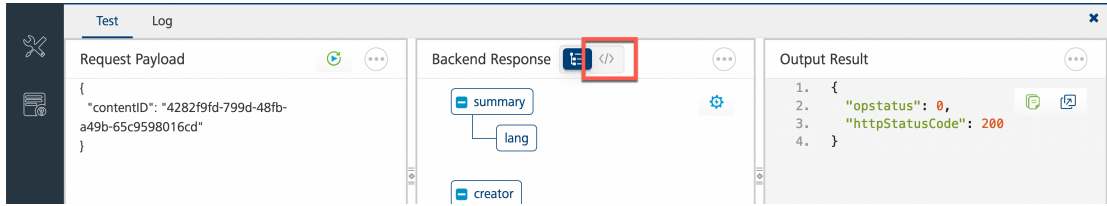
16. Then find the UUID for your default content, e.g. Product 01 under Woodburn Studio Content – Products: **4282f9fd-799d-48fb-a49b-65c9598016cd**. Hover over Product 01 to get the pop up to copy the UUID.



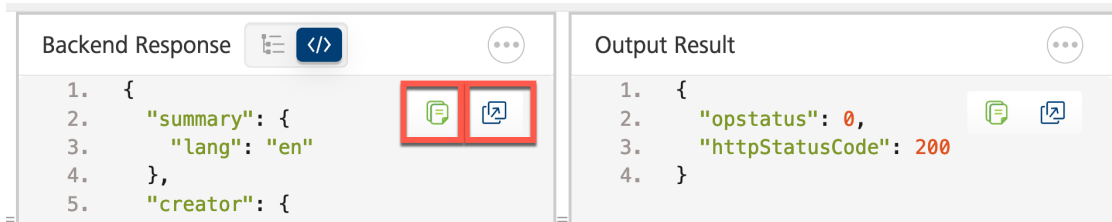
17. Then add the request parameter **contentID** using **Add Parameter** and set the **Default Value** to the UUID of your content item, in this case **4282f9fd-799d-48fb-a49b-65c9598016cd**. Click **SAVE AND FETCH RESPONSE**.



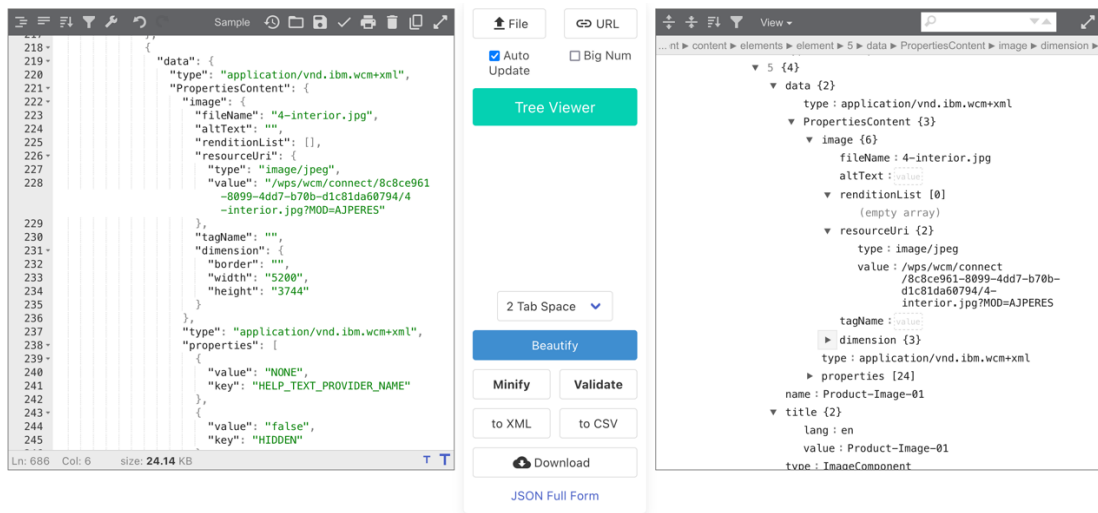
18. Now have a look at the JSON payload. Open the source view.



19. Then you may want to expand it or copy it to paste and see it in your preferred JSON beautifier/viewer, like <https://codebeautify.org/jsonviewer>.



20. To enable your operation to return all the **Product Title, Designer Name, Description** and **images** belonging to this Woodburn Studio product content, you need to understand the JSON format. For example, you see that the Product-Image-01 element is the element number 5 and has the name and data nodes.



21. To understand the structure of this JSON, use the right panel of <https://opensource.hcltechsw.com/experience-api-documentation/ring-api/#operation/webContentReadContent>, where you can expand the path to see that the elements are under a **content – content - elements node** and each **element** has a **name**, **title**, **displayTitle**, **type** and **data** node. Depending on the element type, the data will look different. Here you see the sample structure of an image with its renditions expanded, using `renditionList`.

```

{
  "id": "ad0490ea-27cb-4392-801b-5b559170e69e",
  + "title": { _ },
  + "summary": { _ },
  "name": "Example Content Name",
  "type": "Content",
  "updated": "Fri, 11 Aug 2020 00:00:00.789Z",
  "created": "Fri, 11 Aug 2020 00:00:00.789Z",
  + "author": { _ },
  + "owner": { _ },
  + "lastModifier": { _ },
  + "creator": { _ },
  + "profile": { _ },
  + "workflow": { _ },
  - "content": {
    "type": "application/vnd.ibm.wcm+xml",
    - "content": {
      - "elements": {
        - "element": [
          - {
            "name": "Example Component Reference",
            + "title": { _ },
            + "displayTitle": { _ },
            "type": "ReferenceComponent",
            - "data": {
              "type": "application/vnd.ibm.wcm+xml",
              - "PropertiesContent": {
                "type": "application/vnd.ibm.wcm+xml",
                + "properties": { _ },
                "reference": "/wps/mycontenthandler/!ut/p/digest!uX_-12H17yLg1C25cVR9g/wcmrest/L1",
                + "date": { _ },
                + "resourceUri": { _ },
                "value": "<h1>HCL Digital Experience</h1>",
                - "image": {
                  + "dimension": { _ },
                  "altText": "",
                  "tagName": "",
                  "fileName": "5kB-test-image.png",
                  - "resourceUri": {
                    "damId": "ebc24b4b-7e2c-42a6-bab7-b1b8488486a0",
                    "type": "image/png",
                    "value": "/wps/wcm/myconnect/ebc24b4b-7e2c-42a6-bab7-b1b8488486a0/5kB-test-";
                  },
                  - "renditionList": [
                    - {
                      "type": "desktop",
                      "fileName": "filename.png",
                      - "resourceUri": {
                        "type": "image/png",
                        "value": "/wps/wcm/myconnect/ebc24b4b-7e2c-42a6-bab7-b1b8488486a0/5k";
                      },
                      - "binaryresource": {
                        "fileName": "filename.png",
                        "type": "image/png",
                        "value": "base64";
                      }
                    }
                  ],
                }
              },
            },
            + "jsp": { _ },
            + "linkElement": { _ },
            "double": "1,000",
            + "optionSelection": { _ },
            + "userSelection": { _ },
            + "file": { _ }
          }
        ]
      }
    }
  },
  - "links": {
    + "publish": { _ },
  }
}

```

22. You may use a JSON Path Tester, like <https://codebeautify.org/jsonpath-tester>, to find the paths for each of your elements. You may do a lookup for each element using its name in the following path `$.content.content.elements.element[?(@.name=='<element name>')]`, e.g. for Product-Name this would be `$.content.content.elements.element[?(@.name=='Product-Name')]`. In this way, you may find the paths to product title, designer name, description, and images.

The screenshot displays the Code Beautify JSON Path Tester interface. It is divided into three main sections:

- JSON Input:** Contains a text area with a sample JSON object. A red box highlights the `"value": "Product One"` field within the `elements` array.
- JSON Path Expression:** A text input field containing the path `$.content.content.elements.element[?(@.name=='Product-Name')]`. A red box highlights this entire expression.
- Result : Generated JsonPath:** A code editor showing the resulting JSON path `$.name` for the selected element. A red box highlights this path.

23. The images are a bit special, as they have a source image, accessible with `$.content.content.elements.element[?(@.name=='<Image Element Name>')].data.PropertiesContent.image.resourceUri.value` and different renditions that may be retrieved easily as well using `$.content.content.elements.element[?(<Image Element Name>)].data.PropertiesContent.image.renditionList[?(@.name=='<rendition>')].resourceUri.value`. See the table with element names and corresponding paths.

Element Name	Path
Product-Name	<code>\$.content.content.elements.element[?(@.name=='Product-Name')].data.PropertiesContent.value</code>
Designer-Name	<code>\$.content.content.elements.element[?(@.name=='Designer-Name')].data.PropertiesContent.value</code>
Description	<code>\$.content.content.elements.element[?(@.name=='Description')].data.PropertiesContent.value</code>
Product-Image-01	Source: <code>\$.content.content.elements.element[?(@.name=='Product-Image-01')].data.PropertiesContent.image.resourceUri.value</code> Desktop: <code>\$.content.content.elements.element[?(@.name=='Product-Image-01')].data.PropertiesContent.image.renditionList[?(@.name=='desktop')].resourceUri.value</code> Tablet: <code>\$.content.content.elements.element[?(@.name=='Product-Image-01')].data.PropertiesContent.image.renditionList[?(@.name=='tablet')].resourceUri.value</code> Smartphone: <code>\$.content.content.elements.element[?(@.name=='Product-Image-01')].data.PropertiesContent.image.renditionList[?(@.name=='smartphone')].resourceUri.value</code>
Product-Image-02	Source: <code>\$.content.content.elements.element[?(@.name=='Product-Image-02')].data.PropertiesContent.image.resourceUri.value</code> Desktop: <code>\$.content.content.elements.element[?(@.name=='Product-Image-02')].data.PropertiesContent.image.renditionList[?(@.name=='desktop')].resourceUri.value</code> Tablet: <code>\$.content.content.elements.element[?(@.name=='Product-Image-02')].data.PropertiesContent.image.renditionList[?(@.name=='tablet')].resourceUri.value</code> Smartphone: <code>\$.content.content.elements.element[?(@.name=='Product-Image-02')].data.PropertiesContent.image.renditionList[?(@.name=='smartphone')].resourceUri.value</code>
Product-Image-03	Source: <code>\$.content.content.elements.element[?(@.name=='Product-Image-03')].data.PropertiesContent.image.resourceUri.value</code> Desktop: <code>\$.content.content.elements.element[?(@.name=='Product-Image-03')].data.PropertiesContent.image.renditionList[?(@.name=='desktop')].resourceUri.value</code> Tablet: <code>\$.content.content.elements.element[?(@.name=='Product-Image-03')].data.PropertiesContent.image.renditionList[?(@.name=='tablet')].resourceUri.value</code> Smartphone: <code>\$.content.content.elements.element[?(@.name=='Product-Image-03')].data.PropertiesContent.image.renditionList[?(@.name=='smartphone')].resourceUri.value</code>

24. Go to your **Response Output** and start creating the **productName**, **designerName** and **description** parameters by using the path defined above for the images, you may manage them into a collection of elements (images). Create a collection of elements (images) that has a `data.PropertiesContent.image` defined, using **`$.content.content.elements.element[?(!@.data.PropertiesContent.image)]`**. And then, for each image, add the name with `name`, entries of the `sourceUri` values for source using **`data.PropertiesContent.image.resourceUri.value`**, and the desktop, tablet and smartphone renditions (if they exist, which is not the case by default for the Woodburn Studio Products) for each image, using **`data.PropertiesContent.image.renditionList[?(@.name=='<rendition>')].resourceUri.value`**. The next time you test the operation you will see the list of image and other elements with values in the Output Response. To re-test, click the green **play** icon located left Test Log panel.

The screenshot displays the HCL Volt MX interface. The top section shows the 'Response Output' configuration table, which is highlighted with a red box. The table lists parameters for productName, designerName, description, images, name, source, desktop, tablet, and smartphone. The 'images' parameter is a collection, while the others are strings. The 'name', 'desktop', 'tablet', and 'smartphone' parameters are also marked as 'images' in the 'COLLECTION...' column.

NAME	PATH	SCOPE	DATA TYPE	COLLECTION...
productName	<code>\$.content.content.elements.element[?(@.name=='Product-Name')].data.PropertiesContent.value</code>	response	string	
designerName	<code>\$.content.content.elements.element[?(@.name=='Designer-Name')].data.PropertiesContent.value</code>	response	string	
description	<code>\$.content.content.elements.element[?(@.name=='Description')].data.PropertiesContent.value</code>	response	string	
images	<code>\$.content.content.elements.element[?(!@.data.PropertiesContent.image)]</code>	response	collection	
name	<code>name</code>	response	string	images
source	<code>data.PropertiesContent.image.resourceUri.value</code>	response	string	images
desktop	<code>data.PropertiesContent.image.renditionList[?(@.name=='desktop')].resourceUri.value</code>	response	string	images
tablet	<code>data.PropertiesContent.image.renditionList[?(@.name=='tablet')].resourceUri.value</code>	response	string	images
smartphone	<code>data.PropertiesContent.image.renditionList[?(@.name=='smartphone')].resourceUri.value</code>	response	string	images

Below the table, the 'Test' panel shows the 'Request Payload' and the 'Output Result'. The 'Request Payload' is a JSON object with a 'contentID' field. The 'Output Result' is a JSON object with an 'images' array containing an object with 'name', 'source', 'desktop', 'tablet', and 'smartphone' fields.

```

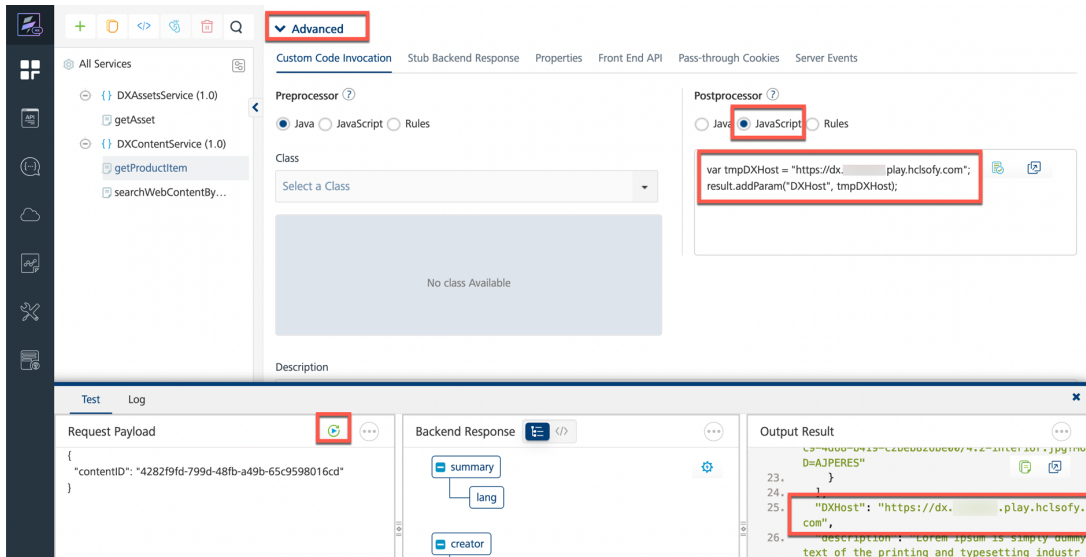
Request Payload:
{
  "contentID": "4282f9fd-799d-48fb-a49b-65c9598016cd"
}

Output Result:
1. {
2.   "images": [
3.     {
4.       "tablet": "",
5.       "desktop": "",
6.       "smartphone": "",
7.       "name": "Product-Image-01",
8.       "source": "/Apps/ACM/Connect/8c8ce961-8899-4d67-b70b-d1c81da60794/4-interior.jpg?MOD=AJPERES"
9.     }
10.  ]
}

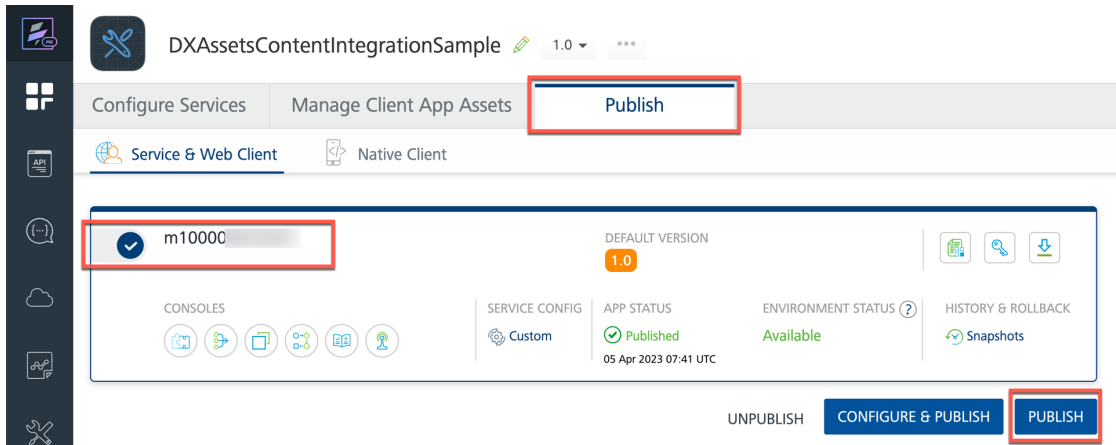
```

25. Then, like with the DXAssets, you need to add the host to allow you create a full URL to your images. Click **Advanced**, select **JavaScript** and paste this code, updated with your own host URL. Click **SAVE AND FETCH RESPONSE** and check it works.

```
var tmpDXHost = "https://dx.sbx0000.play.hclsofy.com";
result.addParam("DXHost", tmpDXHost);
```

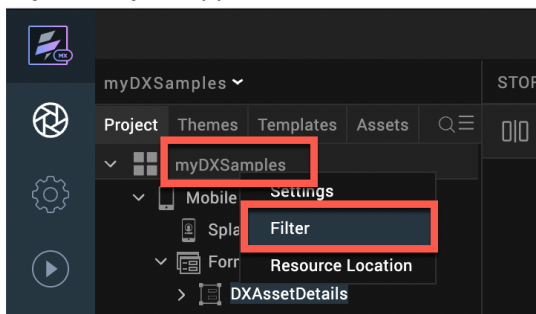


26. Then republish your Foundry application **DXAssetsContentIntegrationSample**. Click the **Publish** tab, select your runtime and click the **Publish** button.

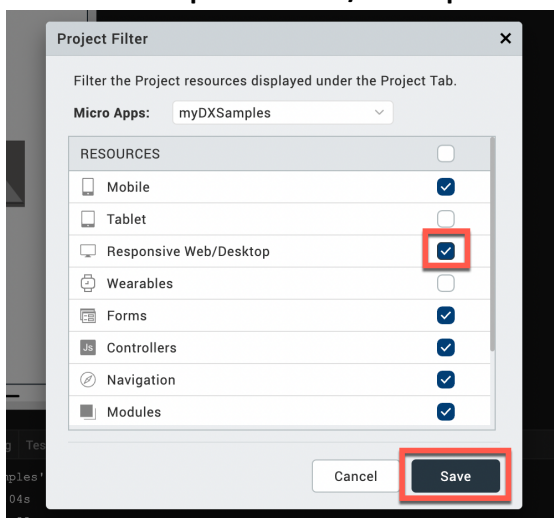


27. You may also use the HCL Digital Experience Content Adaptor. This includes all endpoints of the WCM REST V2 APIs. You can download this from the Volt MX Marketplace <https://marketplace.hclvoltx.com/items/hcl-dx-content-adapter>. Details on how to download, import and start using are described here https://opensource.hcltechsw.com/digital-experience/latest/extend_dx/integration/mx/example/dx_apis_in_foundry/.

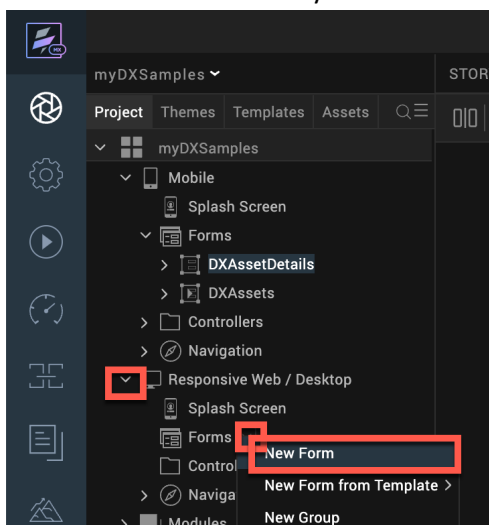
28. Now switch to your Volt Iris IDE to create your web application. In your Iris project **myDXSamples**, you need to unhide the Responsive Web /Desktop first, as this help you creating the right resources more easily for your web application. For that, right click on your **myDXSamples** application and then **Filter**.



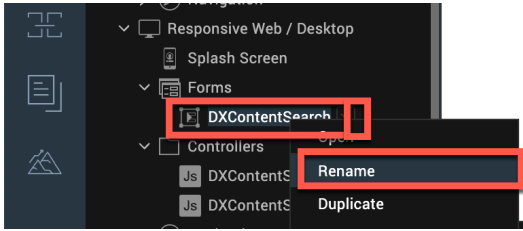
29. Then check **Responsive Web/Desktop** and click **Save**.



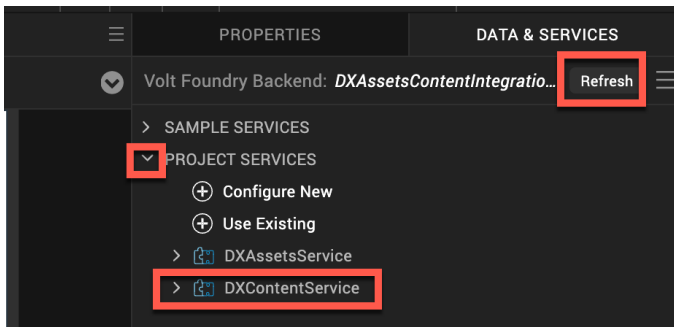
30. Then expand the **Responsive Web / Desktop**, right click on **Forms** and click **New Form** to create a form that allows you to search the right content first.



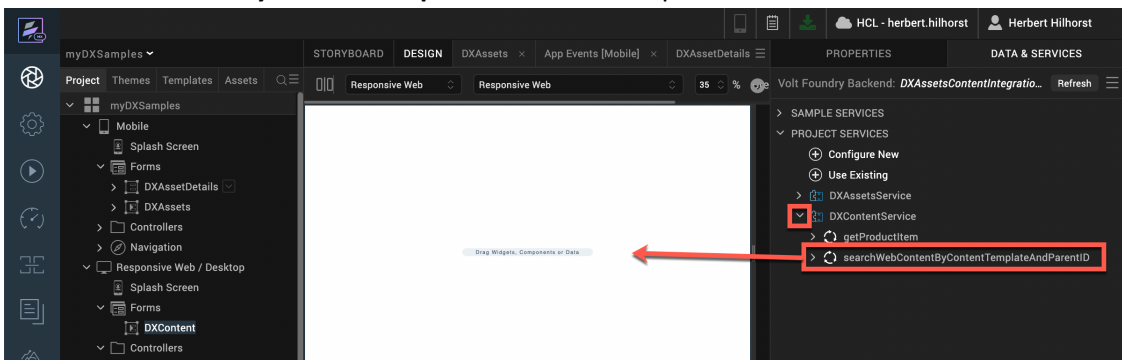
31. Then rename it, e.g. to **DXContentSearch**.



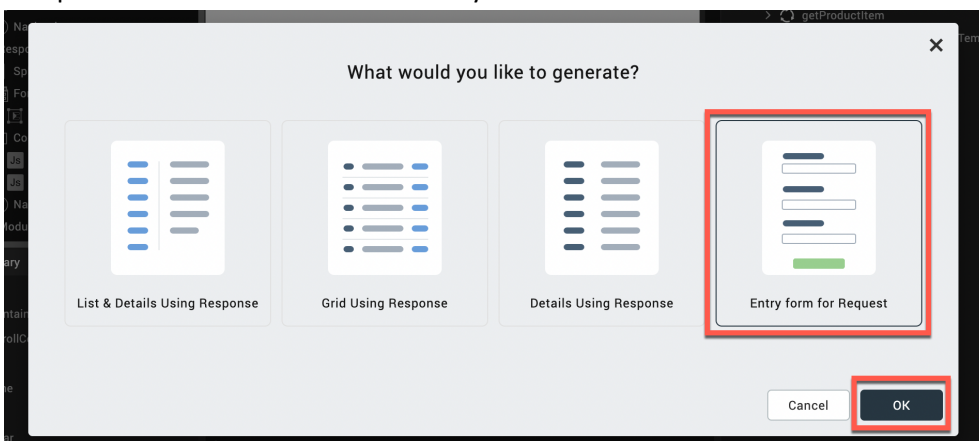
32. This application is already associated with the Foundry services you have created before. As you have updated the Foundry services, you need to refresh the Foundry association. Under **DATA & SERVICES**, click **Refresh** and expand **PROJECT SERVICES** to see your **DXContentService**.



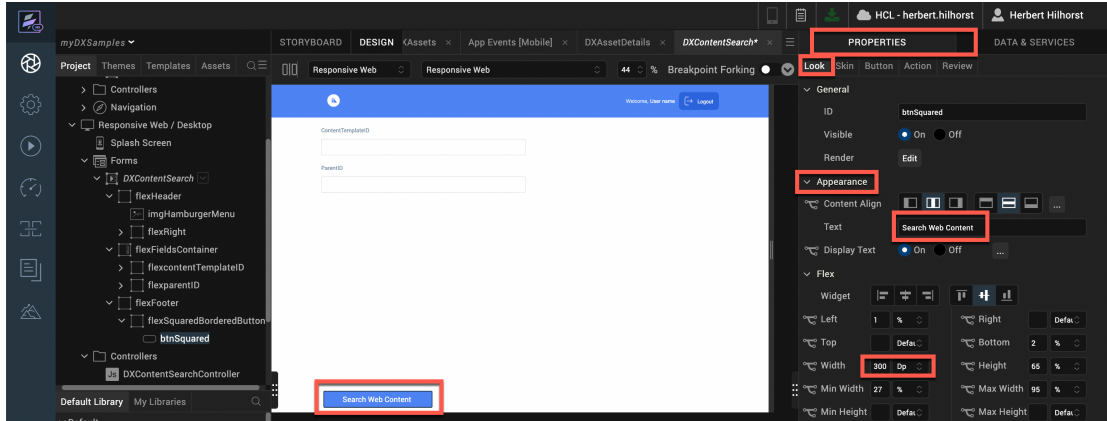
33. Expand **DXContentService** to see your operations and drag & drop **searchWebContentByContentTemplateAndParentID** operation into the form.



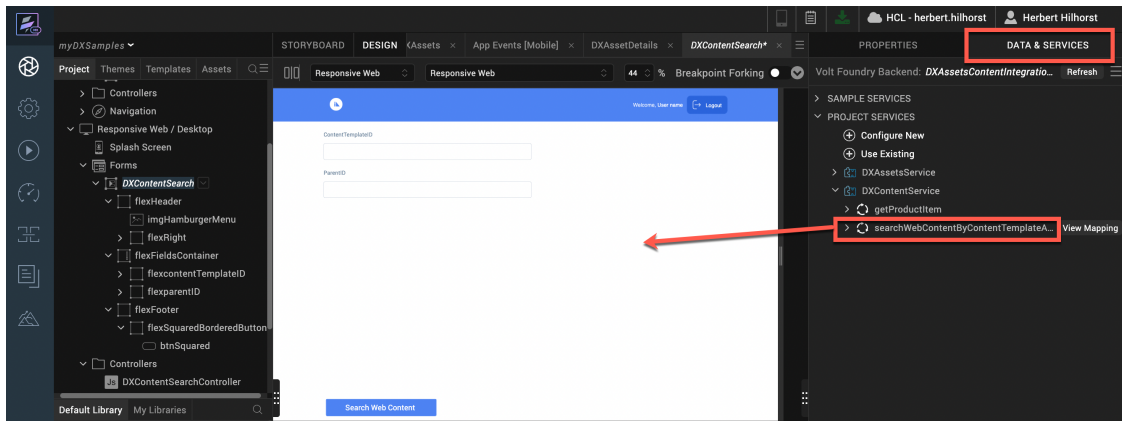
34. Select **Entry form for Request** when asked and **OK**, as this allows you to select the Content Template and Parent ID of the content you want to show.



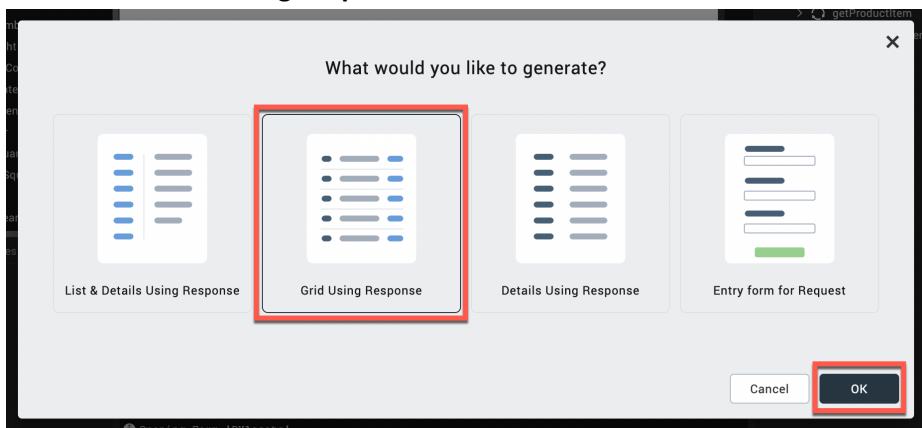
35. This adds a flex container with your input parameters and a button. Then update this button. Click your button, then **PROPERTIES**, and update under **Look – Appearance** the **text**, e.g. to **Search Web Content** and the **width**, e.g. to **300 Dp**. Feel free to make other changes as well.



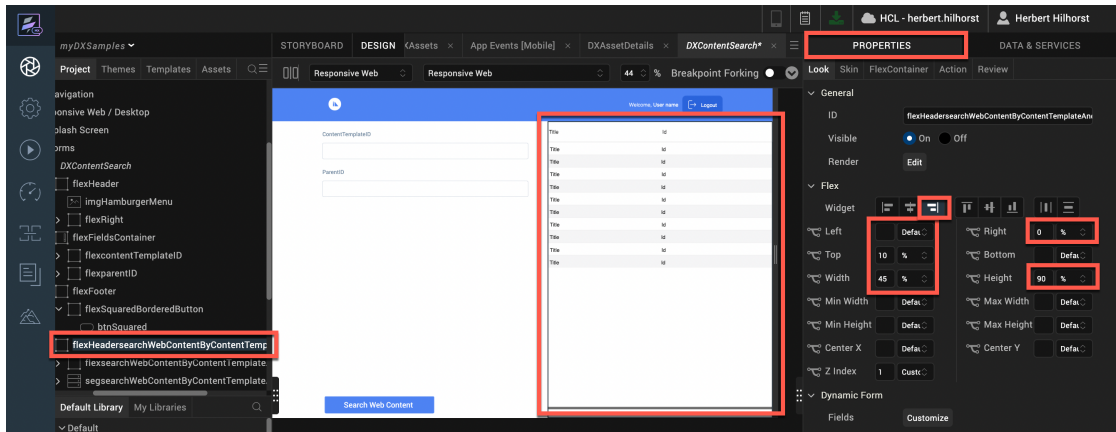
36. Then you want to show the results on the same form. Select the form, click **DATA & SERVICES** and drag your **searchWebContentTemplateAndParentID** operation to any place in the form, except for your existing flex container.



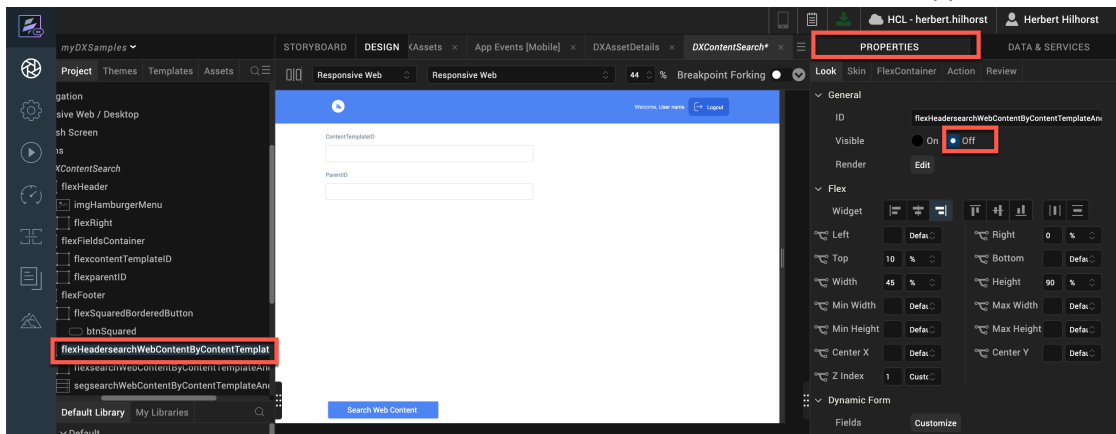
37. Now select a **Grid Using Response** and click **OK**.



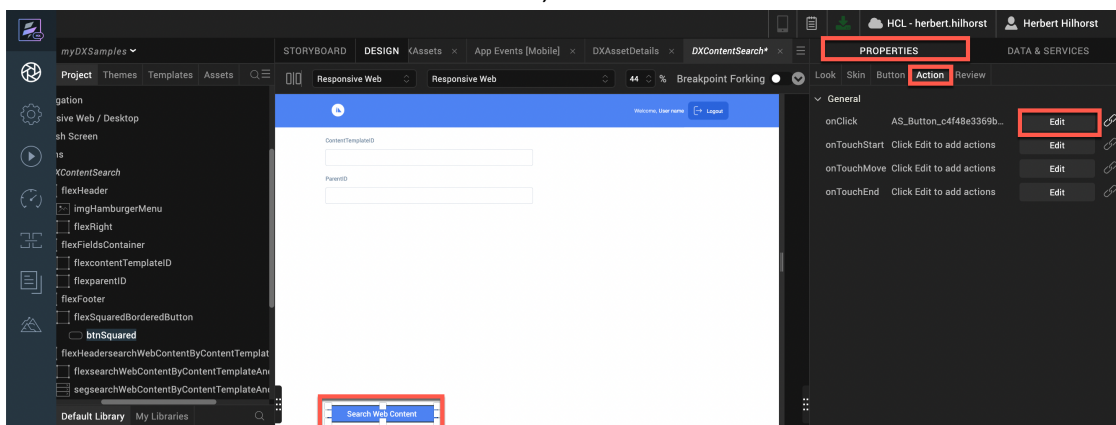
38. Configure this flex container to show correctly on the right of the form. Click **PROPERTIES** and update the Flex parameters, **Pin to right** for the **Widget** setting, **Left to Default, Right to 0%, Top to 10%, Width to 45% and Height to 90%**.



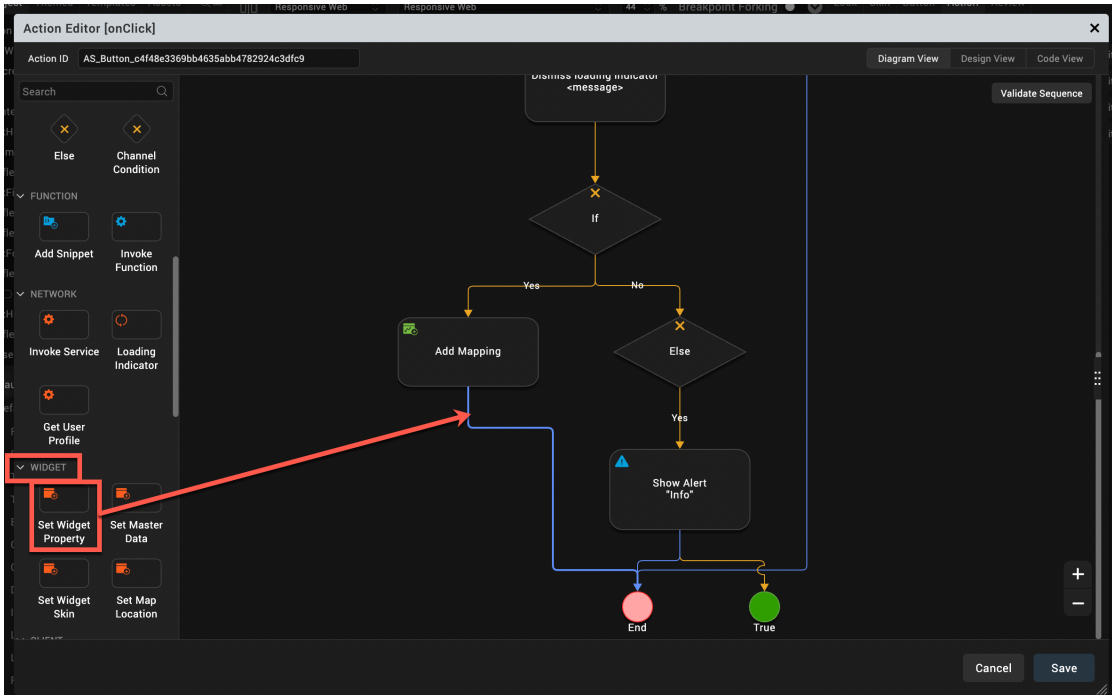
39. Then hide this flex container, until you have done a search. Ensure you have selected the container and then under **PROPERTIES**, switch **Visible** to **Off** and notice it disappears.



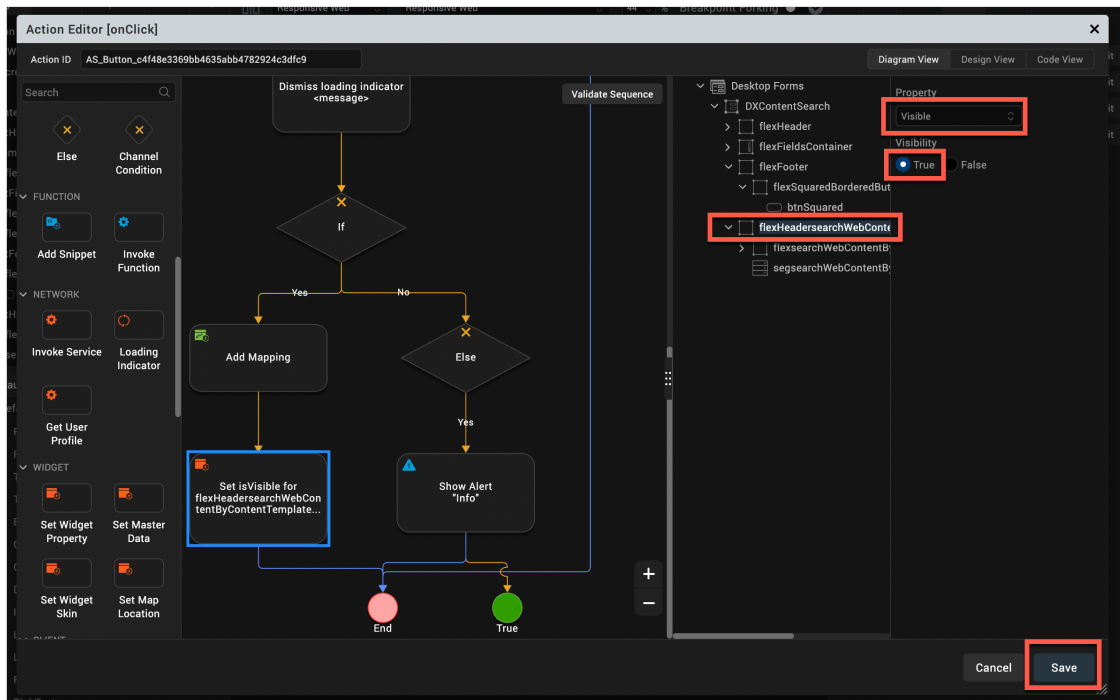
40. Then change the button to show the flex container when you start the search. Select the **button** and then under **PROPERTIES – Action**, click **Edit** for the **onClick** event.



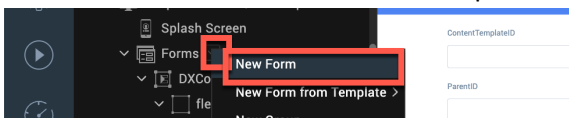
41. Scroll down to see the Add Mapping node and then drag the **Set Widget Property** node under **Widget** category to the **blue connector** between **Add Mapping** and **End** nodes.



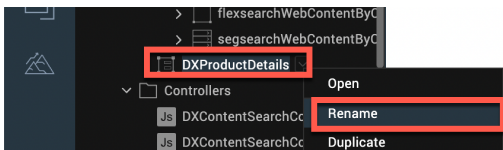
42. Then, select **flexHeadersearchWebContentByContentTemplateAndParentID** flex container and ensure the **Property** is set to **Visible** and **Visibility** to **True**. Click **Save**.



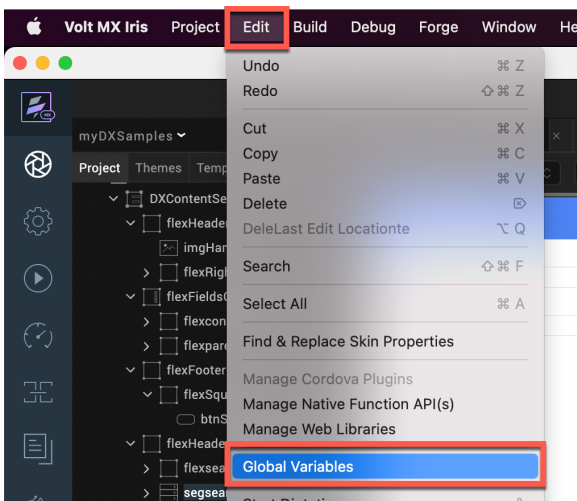
43. Then add a form for the details of each product. Right click **Forms** and **New Form**.



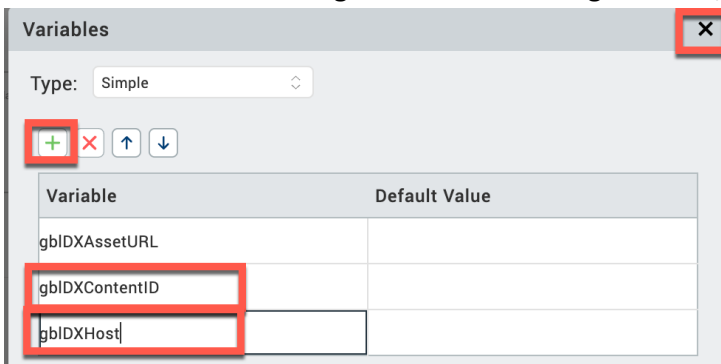
44. And rename it to **DXProductDetails**.



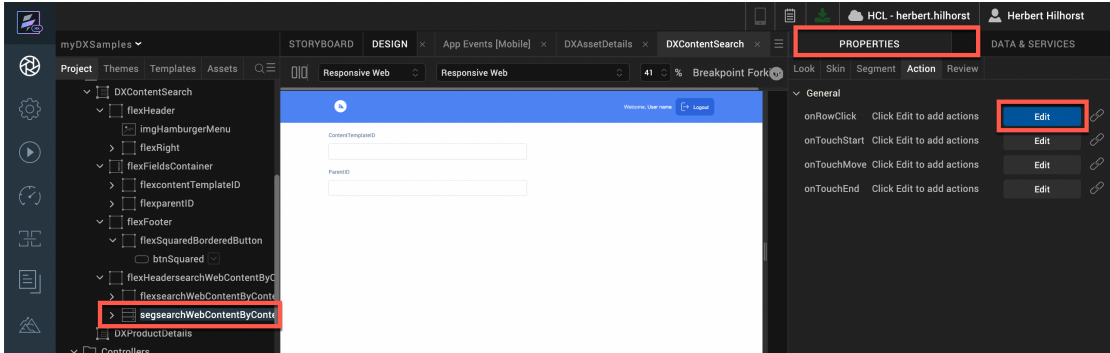
45. Then create a global variable to pass the Content UUID and have easy access to its details in the DXProductDetails form and one to access the DXHost. Click **Edit** and then **Global Variables**.



46. Then add two the variables **gbIDXContentID** and **gbIDXHost**, and close it.

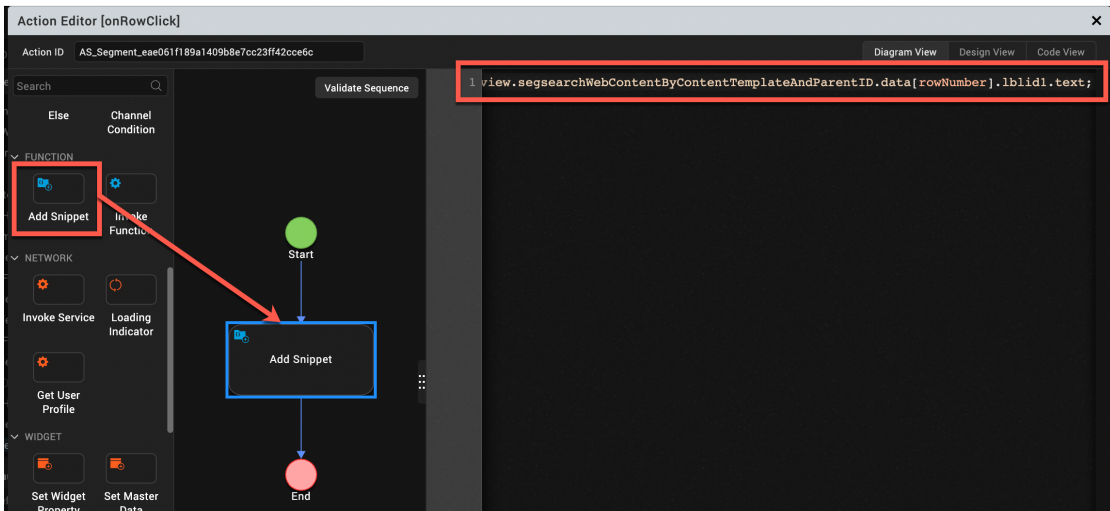


47. Then set this variable when you click a row in your search list of content result. Under the search result form, select the segment **segsearchWebContentByContentTemplateAndParentID**, click **PROPERTIES** and **Edit** for the **onRowClick** action.

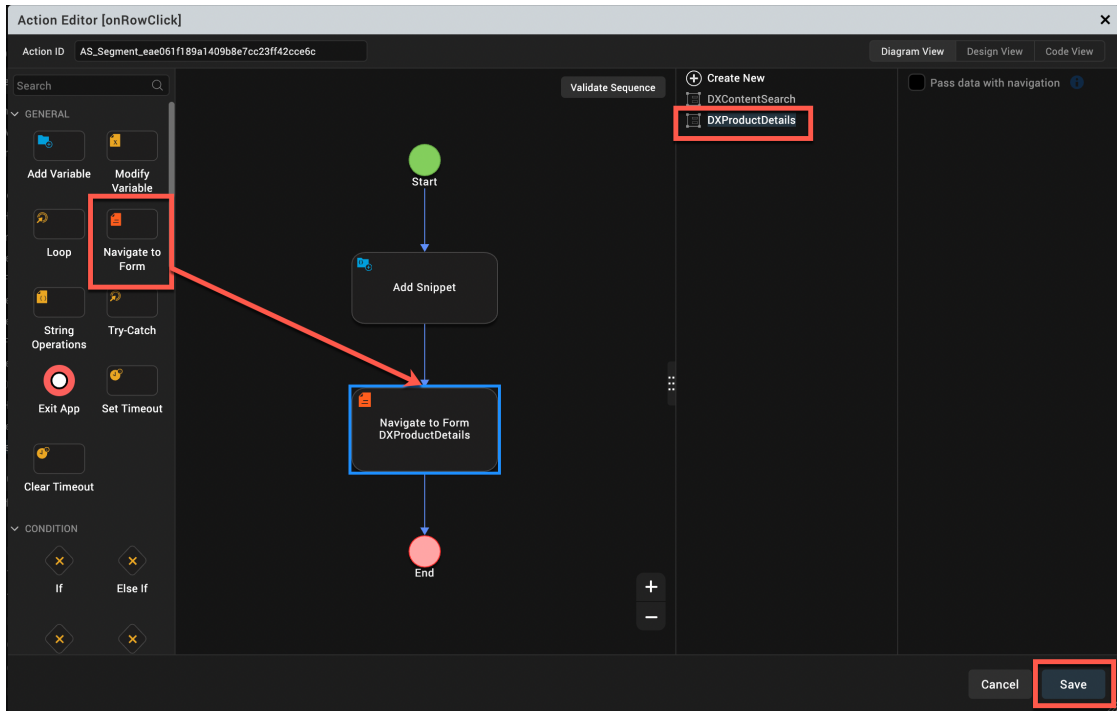


48. Use a snippet to set the new global variable to the content ID in the row. Drag the **FUNCTION – Add Snippet** between **Start** and **End** and add the follow code:

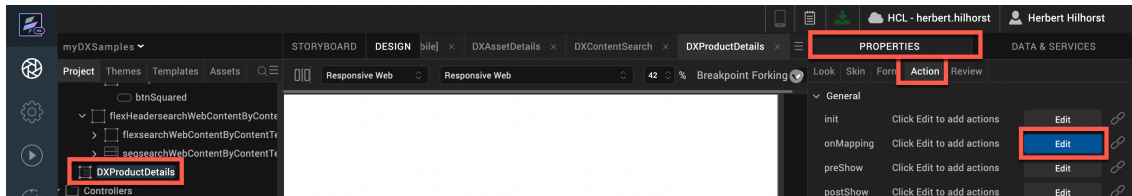
```
gblDXContentID=self.view.segsearchWebContentByContentTemplateAndParentID.data[rowNumber].lblid1.text;
```



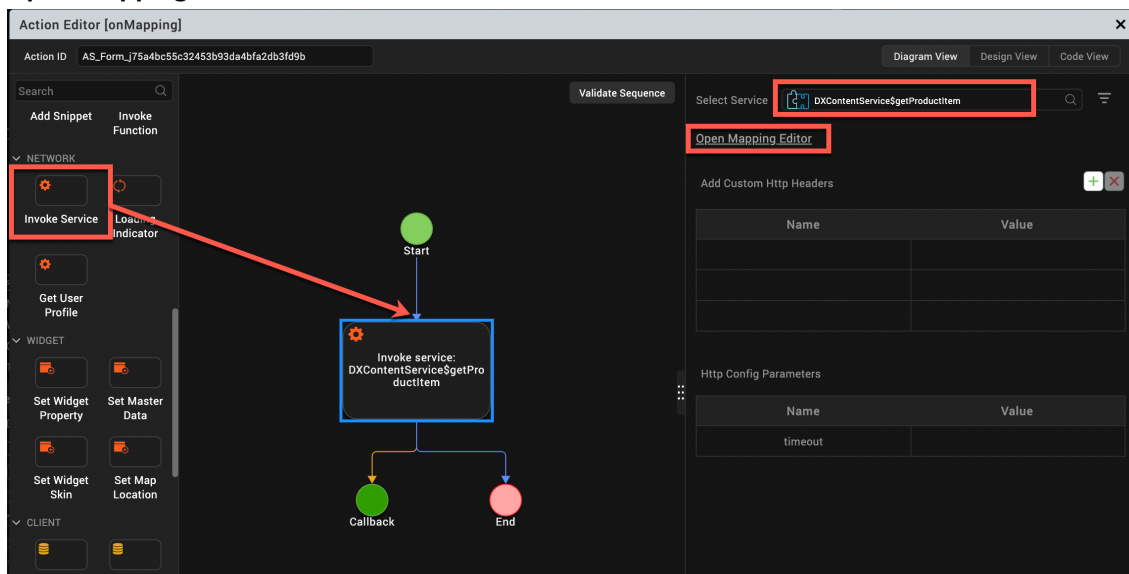
49. Then add a navigation to your new form. Add a **GENERAL – Navigate to From** widget between your **Add Snippet** and **End** and set it to your new **DXProductDetails** form. Then click **Save**.



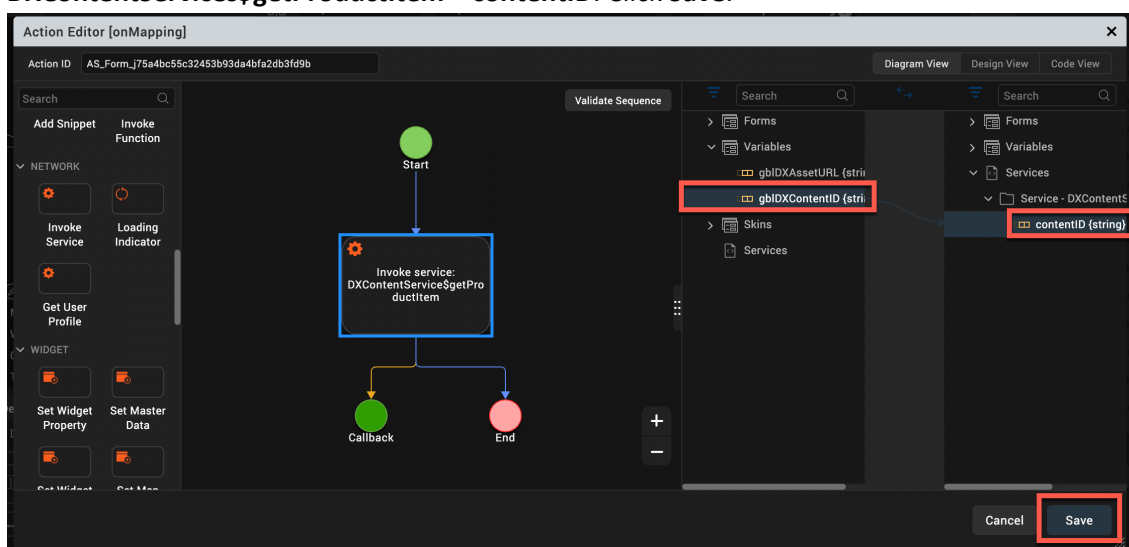
50. Now create the mapping for the form. Select your new form **DXProductDetails** and click **Edit** under **PROPERTIES – Action – onMapping**.



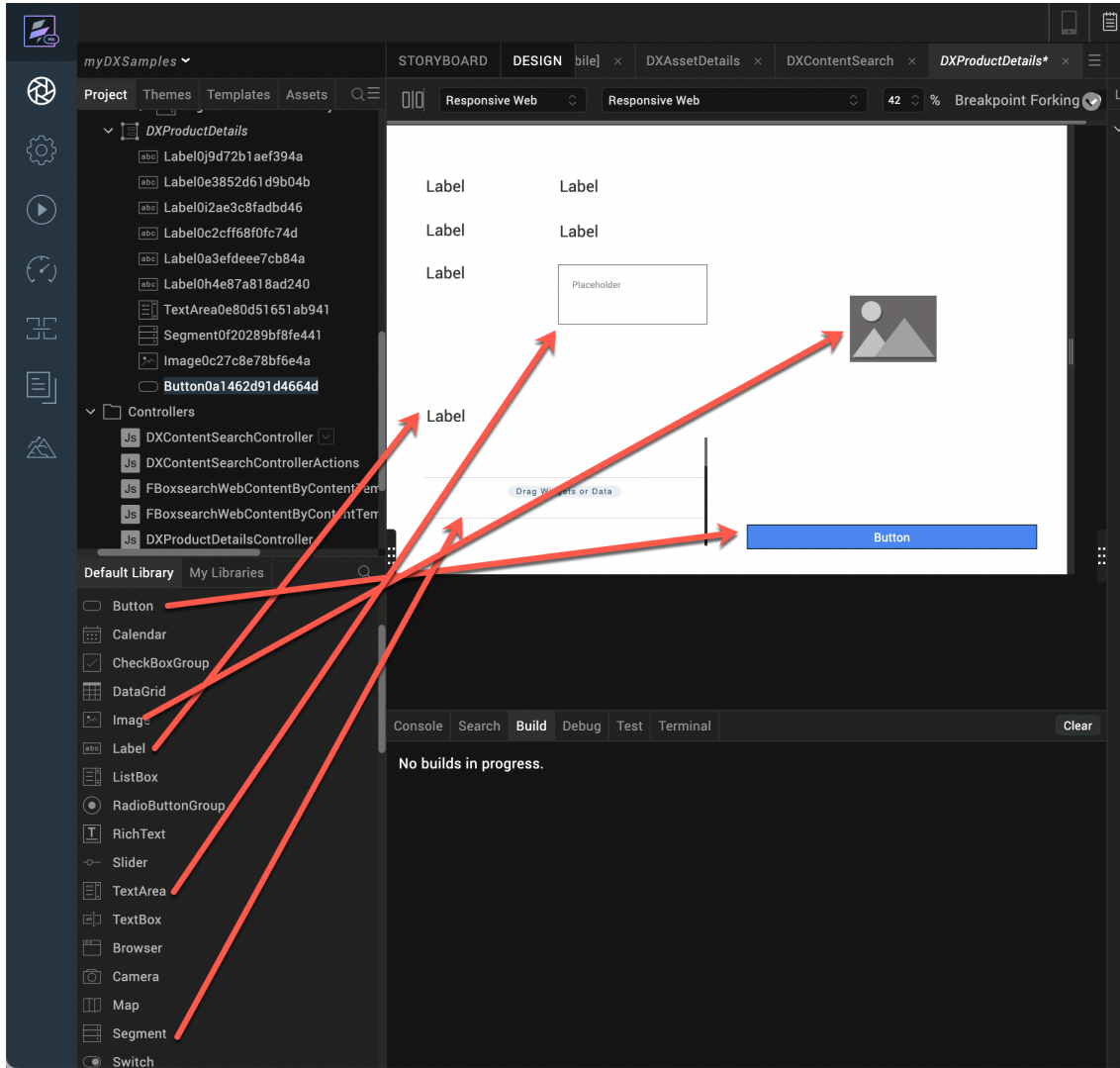
51. Drag the **NETWORK – Invoke Service** between **Start** and **End** and set it to **DXContentService\$getProductItem** operation to get the details of your content. Then click **Open Mapping Editor**.



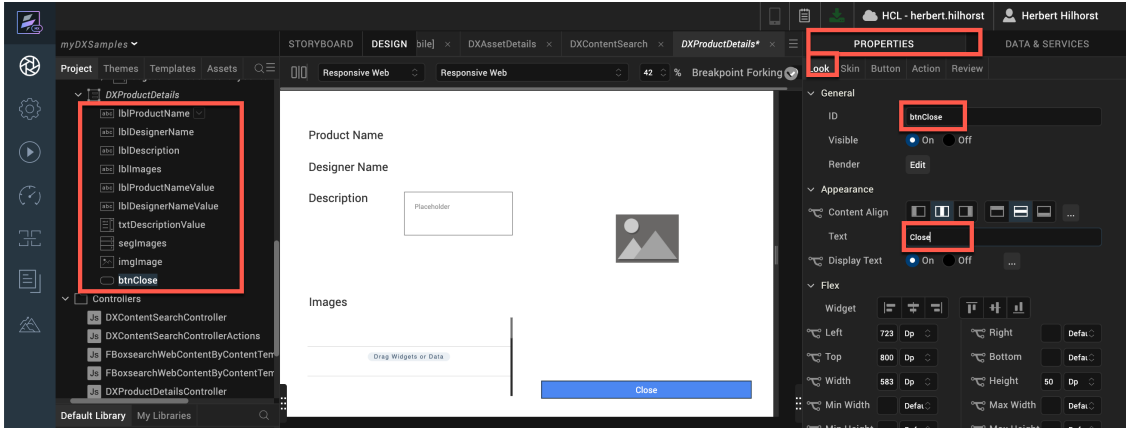
52. Then select your **Variable - gbIDXContentID** and link it to the **Services – Service – DXContentServices\$getProductItem – contentID**. Click **Save**.



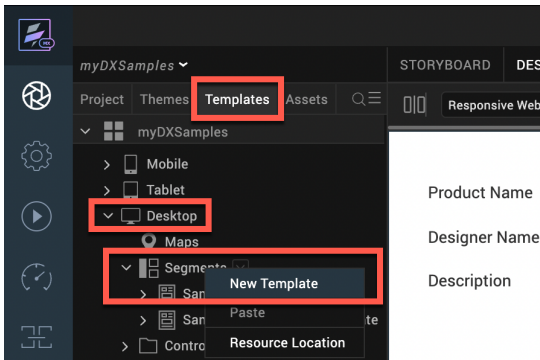
53. Then build your form using the widgets from the palette. To render a collection, you created to manage your image, you may use a segment. You may create something like using **Labels**, a **TextArea**, a **Segment**, an **Image** and a **Button**. You will use the labels to display the text elements, a text area for the description (as this may be multiple lines), a segment for the list of images (source for now, but you may also add renditions later), an image to render the selected image and a button to navigate back to the search page. Drag them to your form from the Widgets Palette.



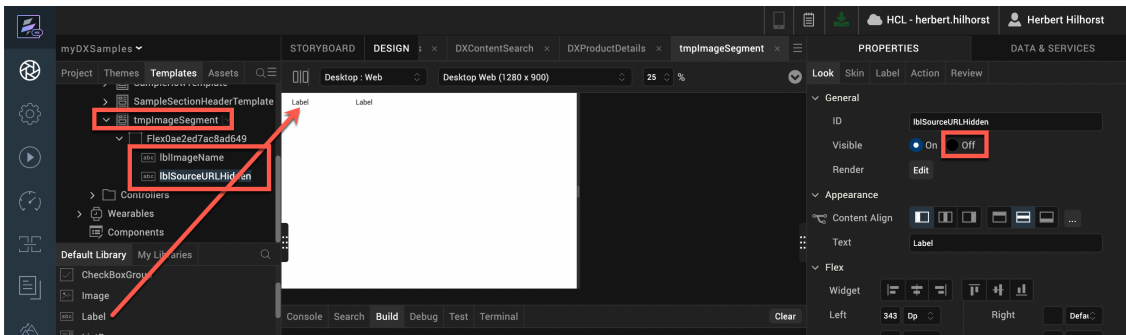
54. Then update the IDs from the autogenerated ones to make them more context specific as well as the text value of the labels, when they have a constant value. Use the prefixes “lbl” for labels, “txt” for the text area, “seg” for the segment and “btn” for your button. For the values, clear the text. It should look like this:



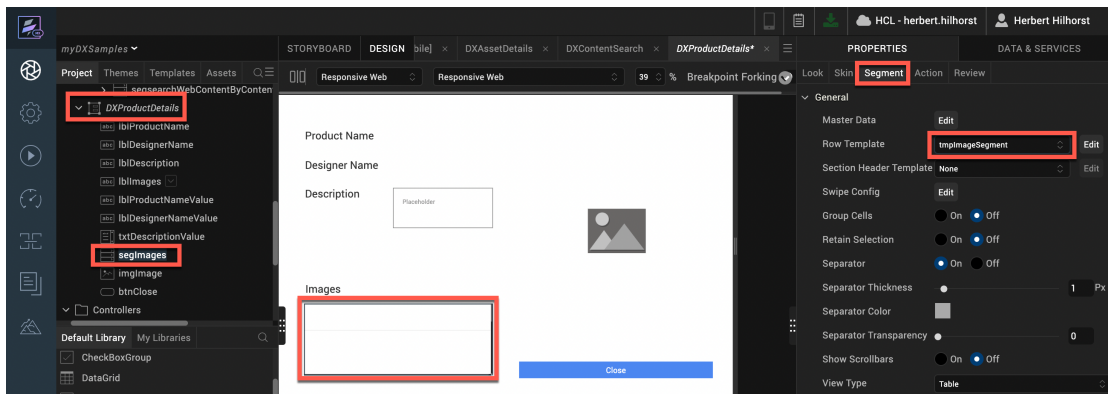
55. Then define a template for the segment for the images that may render the title of the image and set the right URL. Do this for the desktop now. Click **Templates**, open **Desktop**, right click **Segments** and click **New Template**.



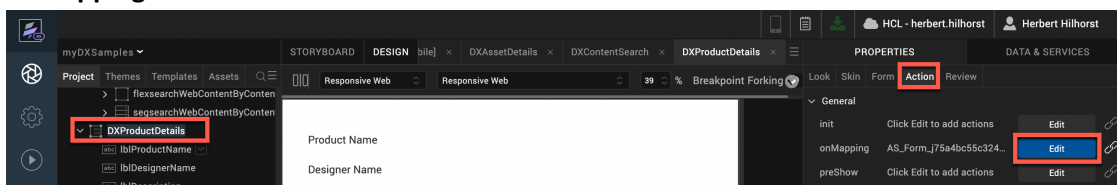
56. Rename the Template to something more easily readable, like **tmpImageSegment**. This template will store the image name (visible) and the image URL (hidden). Then, add a label for each of them, **lblImageName** and **lblSourceURLHidden**. Clear the text of **lblImageName** and hide **lblSourceURLHidden**, by clicking **Visible Off**. When this is set, your template looks empty again. In this case, you will only show the source image. You are making the change now for the desktop. You may repeat these steps for the Tablet and Smartphone renditions as well.



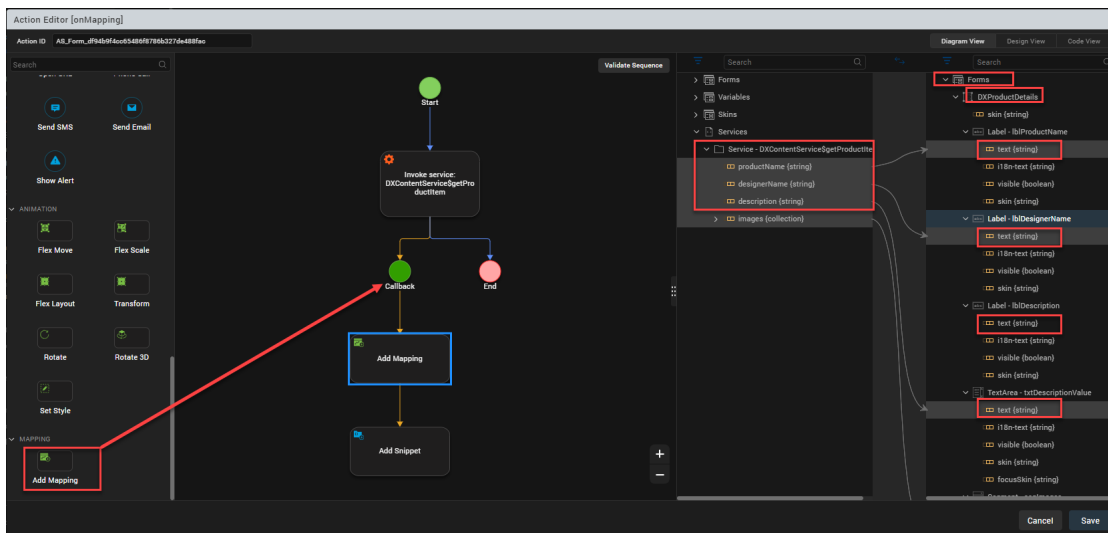
57. Then update your form with this new template. Select your **DXProductDetails** form and then your **seglImages** segment and under **Segment** select your new **tmplImageSegment Row Template**.



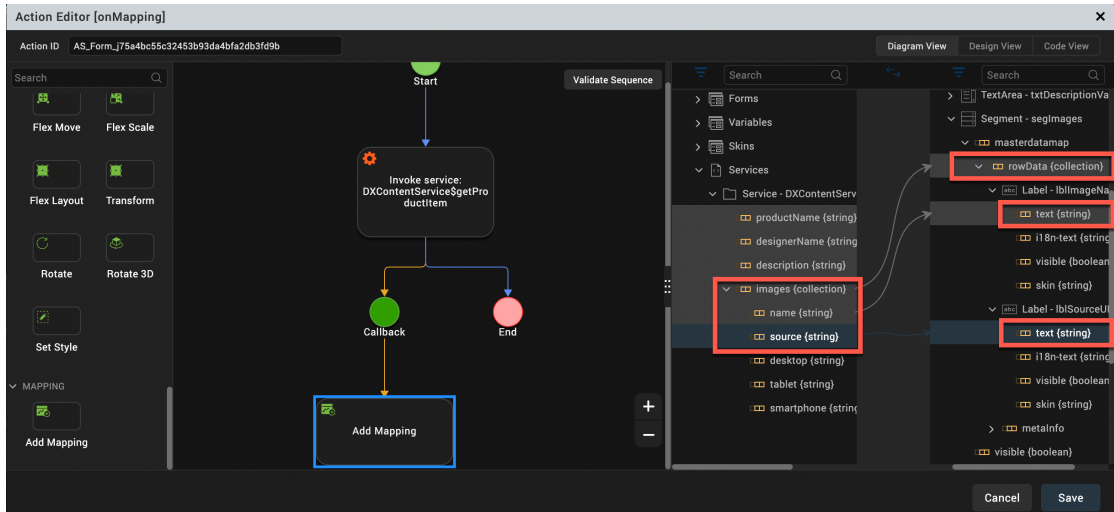
58. Now update your form mapping. Click **DXProductDetails** form, **PROPERTIES – Action** and the **onMapping** Edit.



59. Then add a **MAPPING – Add Mapping** node on top of the **Callback**. It will show below. And then map the first text fields under **Forms – DXProductDetails - productName, designerName** and **description** under **Services – Service – DXContentService\$getProductItem** to the corresponding **text** of the value labels under **Forms – DXProductDetails**.



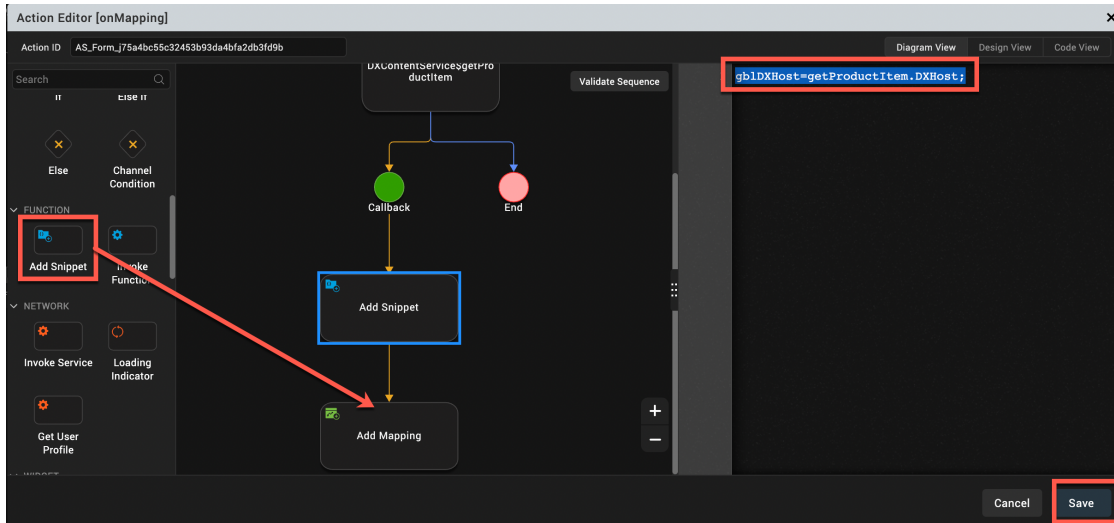
60. Map the **images** collection to the **Segment – masterdatamap – rowData** collection first and then map the **images name** and **source** to the corresponding text of the labels.



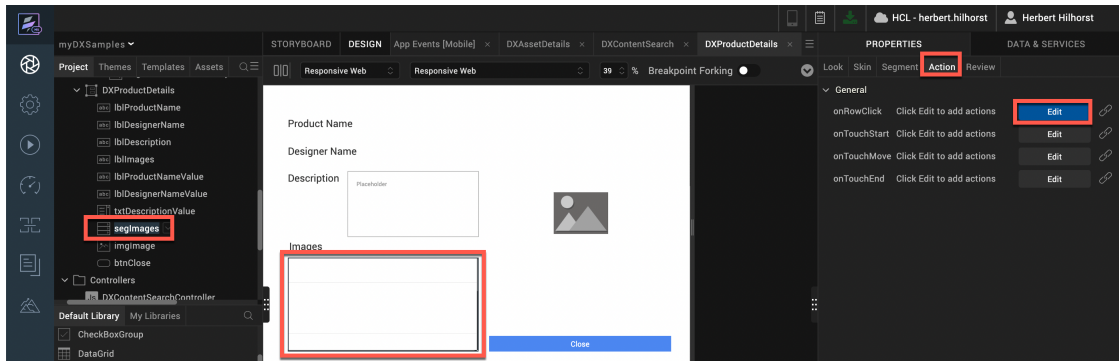
61. Then add an **Add Snippet** widget to set the global variable **DXHost**. Drag it to **Add Mapping** and enter:

```
gblDXHost=getProductItem.DXHost;
```

Then click **Save**.

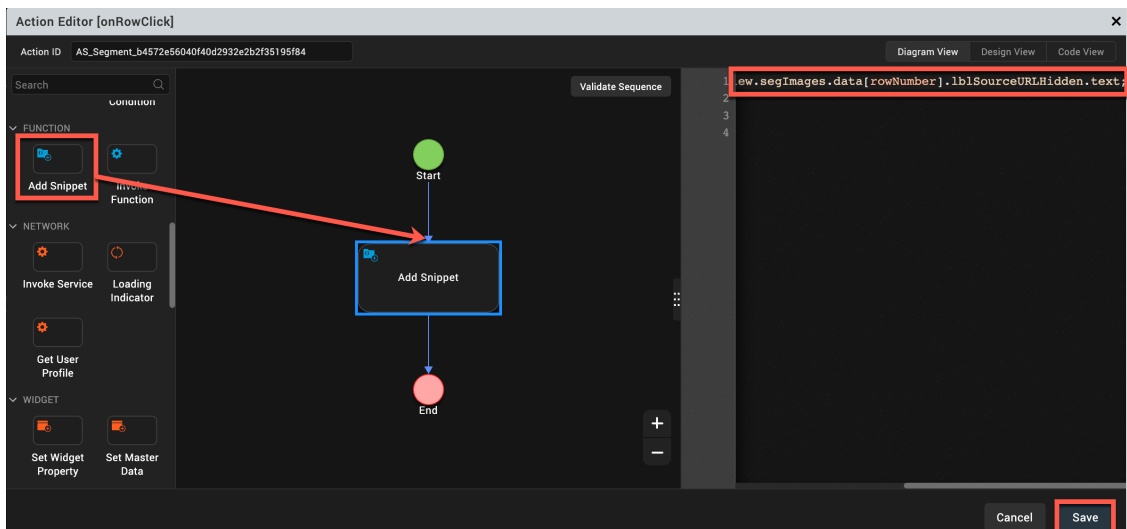


62. Then update the segment to render the image in the form, once one of the images are clicked. Select your **images** segment, click **PROPERTIES – Action** and then **Edit** on the **onRowClick** event.

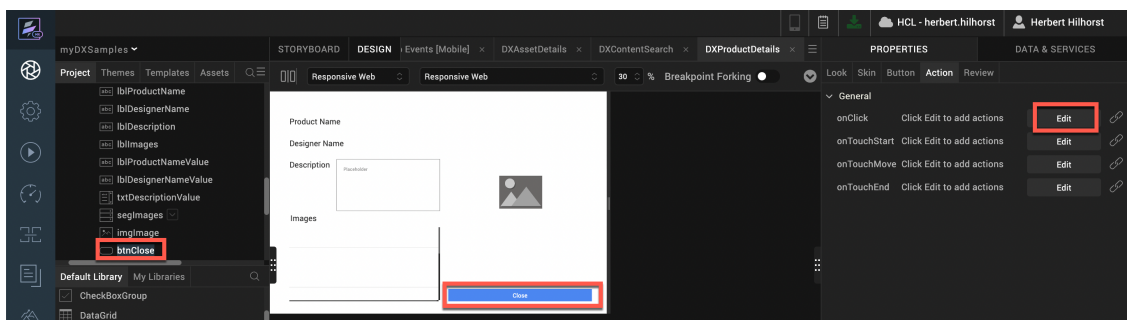


63. Then set the image URL to the one selected in your segment. Add a **FUNCTION – Add Snippet**, set the value using the code below and click **Save**.

```
self.view.imgImage.src=globalDXHost+self.view.segImages.data[rowNumber].lblSourceURLHidden.text;
```



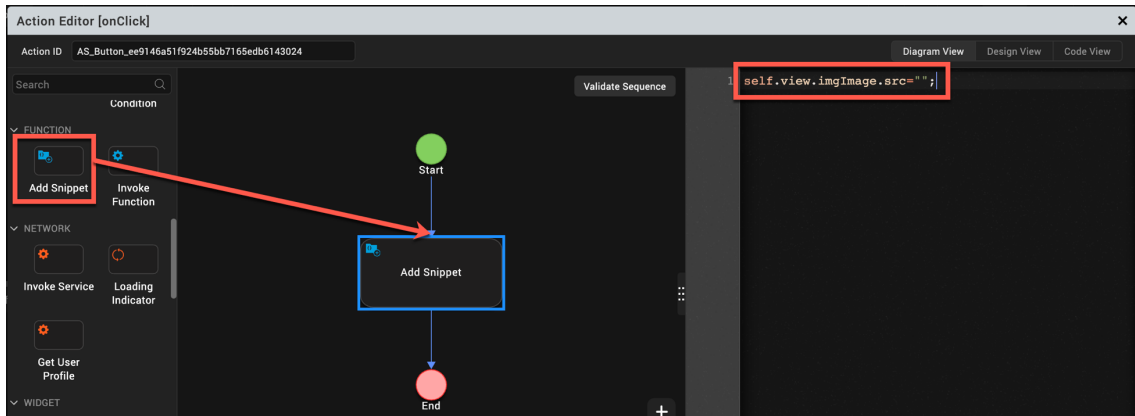
64. Finally, update your button to clear the image and go back to the search form. Select your button and click **onClick Edit**.



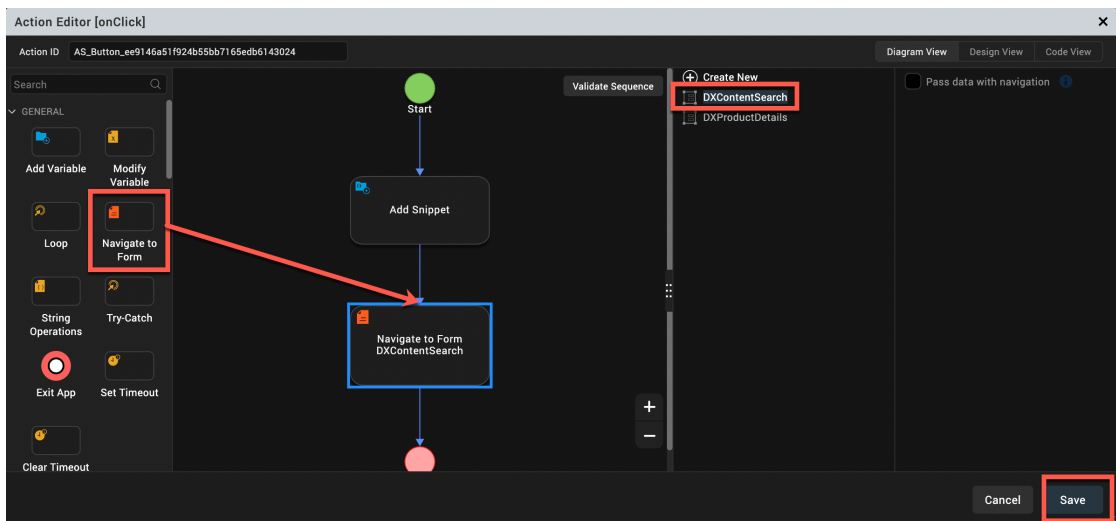
65. First clear the image URL. Drag a **FUNCTION – Add Snippet** and enter:

```
self.view.imgImage.src="";
```

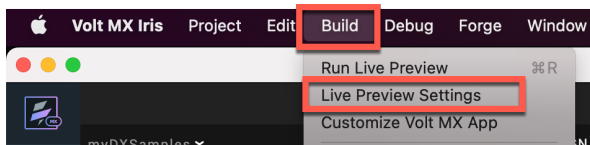
DX INTEGRATION WITH HCL VOLT MX FOR DEVELOPERS



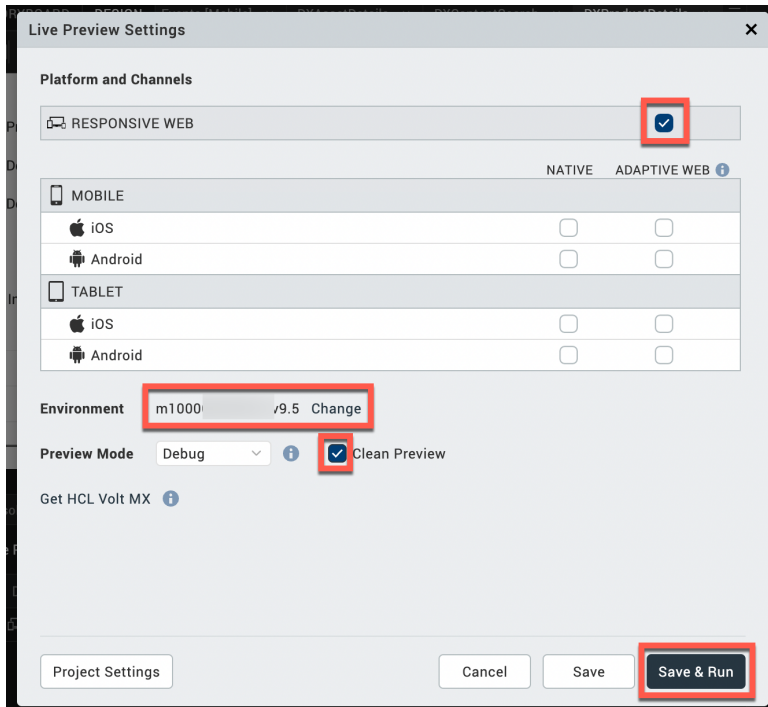
66. Then add the navigation. Drag a **GENERAL – Navigate to Form** node, set it to **DXContentSearch** and click **Save**.



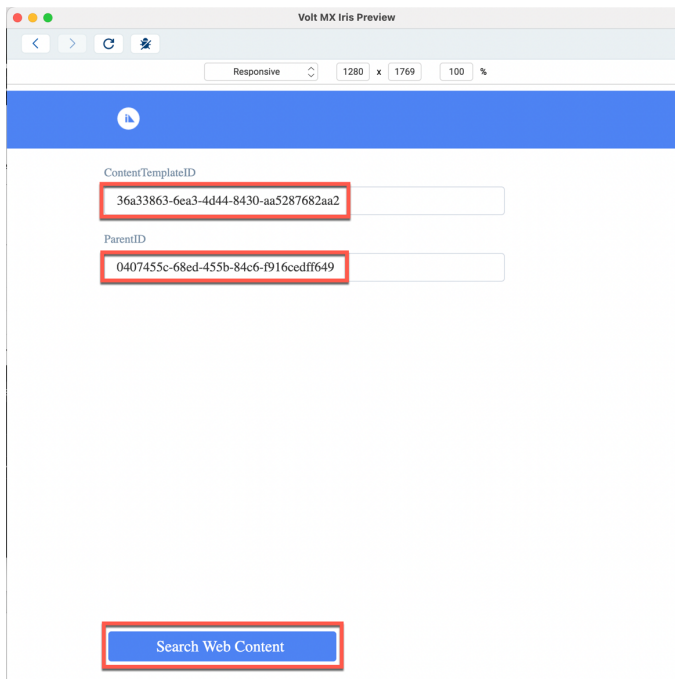
67. Now test it. In the menu click **Build – Live Preview Settings**.



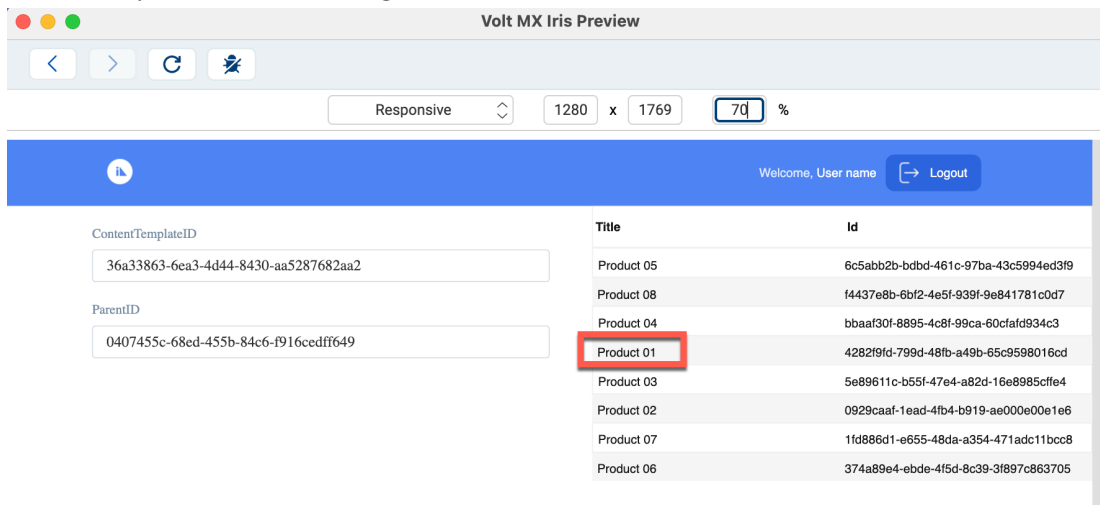
68. Ensure the **Responsive Web** is the only channel selected and select the **Clean Preview** checkbox. Ensure the Foundry runtime matches the Foundry runtime environment, which you published Foundry application. Then click **Save & Run**.



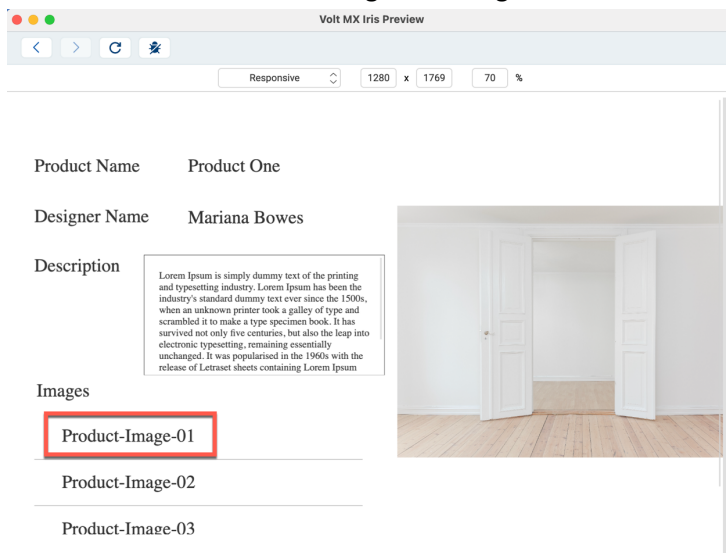
69. Enter your Content Template ID **36a33863-6ea3-4d44-8430-aa5287682aa2** and Parent ID **0407455c-68ed-455b-84c6-f916cedff649** and click **Search Web Content**.



70. It shows the list of content items of that Content Template and ParentID (Site Area). Click one of the products to show, e.g. **Product 01**.



71. It shows the details on that content item. Then select one of the images, e.g. **Product-Image-01** and it should show the image to the right.



72. Feel free to make the UI look better, add the other image renditions, add a loading icon when you load the image, etc.

You have successfully learned how to integrate the DX content into your HCL Volt MX application.

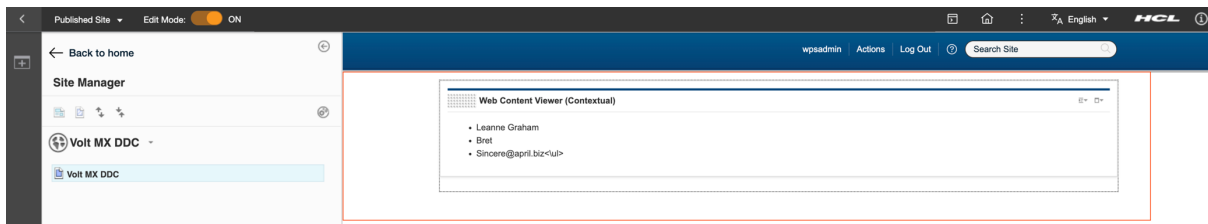
Part 3: Use Volt MX Foundry to Integrate External Data Sources in DX

In this part, you will learn how use Digital Data Connector to integrate data sources from HCL Volt MX Foundry (integrated external data sources) into HCL Digital Experience.

To learn how more on the Digital Data Connector, use the DX for Developers (Beginners) course <https://hclsw.co/hdx-dev-100> with the Digital Data Connector lesson: <https://hclsoftwareu.hcltechsw.com/courses/lesson/?id=1451>.

You may use Volt MX Foundry for API management, as documented in https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Foundry/voltmx_foundry_user_guide/Content/API_Management.html. Foundry helps to integrate easily with many different data sources using endpoint adapters. See details on https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Foundry/voltmx_foundry_user_guide/Content/Services.html.

Then, the Help Center has the following document *Connecting to HCL Volt MX Foundry through Digital Data Connector (DDC)* : https://opensource.hcltechsw.com/digital-experience/latest/extend_dx/ddc/integrating_voltmx_foundry/. Follow these steps and you will get this result.



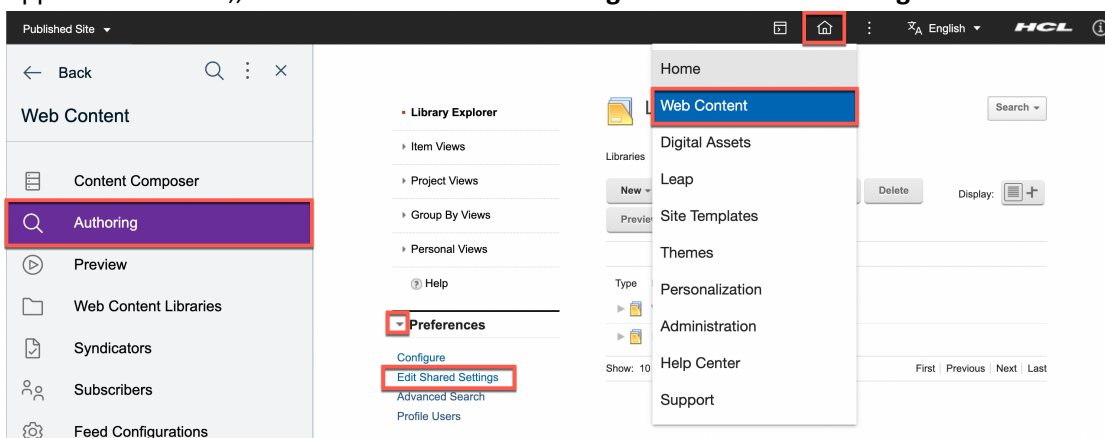
Part 4: Integrate Volt MX Foundry Web Applications in DX

In this part, you learn how to integrate Foundry web applications into HCL Digital Experience.

Today, an easy way to integrate an existing Foundry web application, is using an iFrame. This may change in the future. This works only for Foundry web applications that are suitable to be displayed in an iFrame.

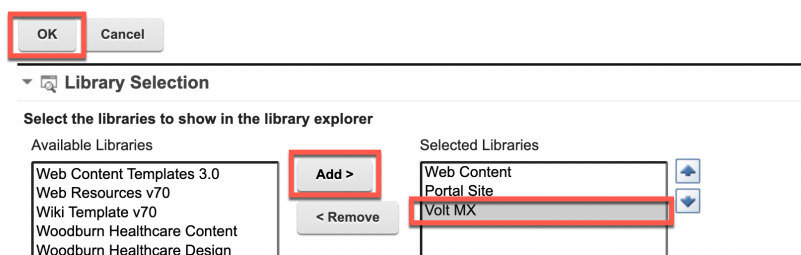
This has been used for the integration you have seen in the DX integration with HCL Volt MX for Business Users.

1. Go to the Web Content Authoring and add the Volt MX web content library. Open the Applications Menu, click **Web Content – Authoring and Edit Shared Settings**.

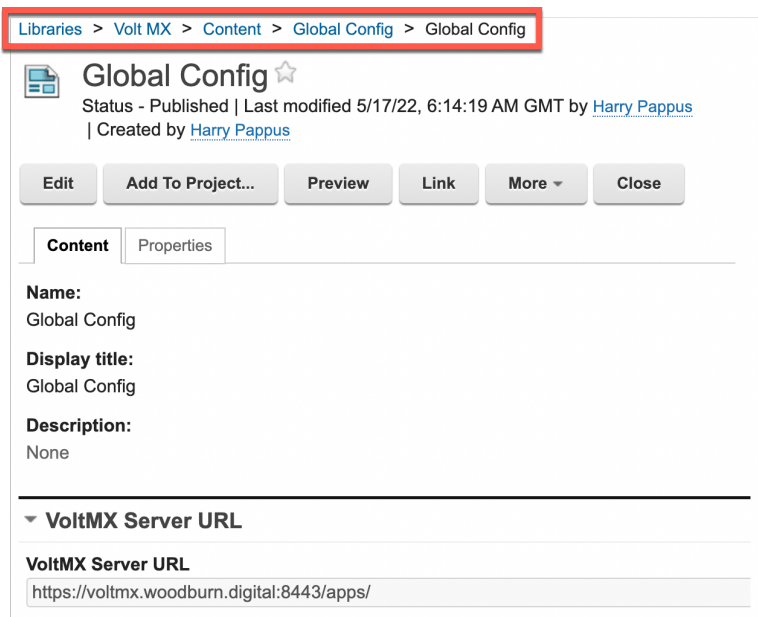


2. Then add the Volt MX web content library to the right and click **OK**.

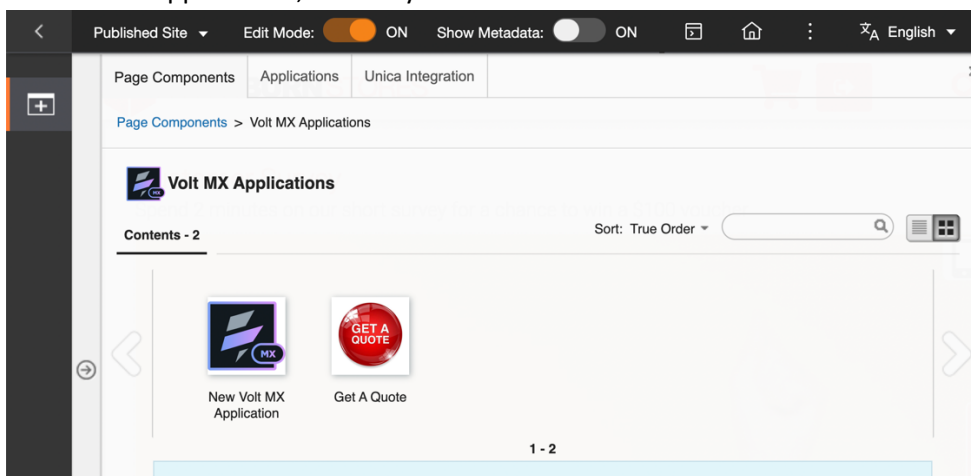
 Edit Shared Settings



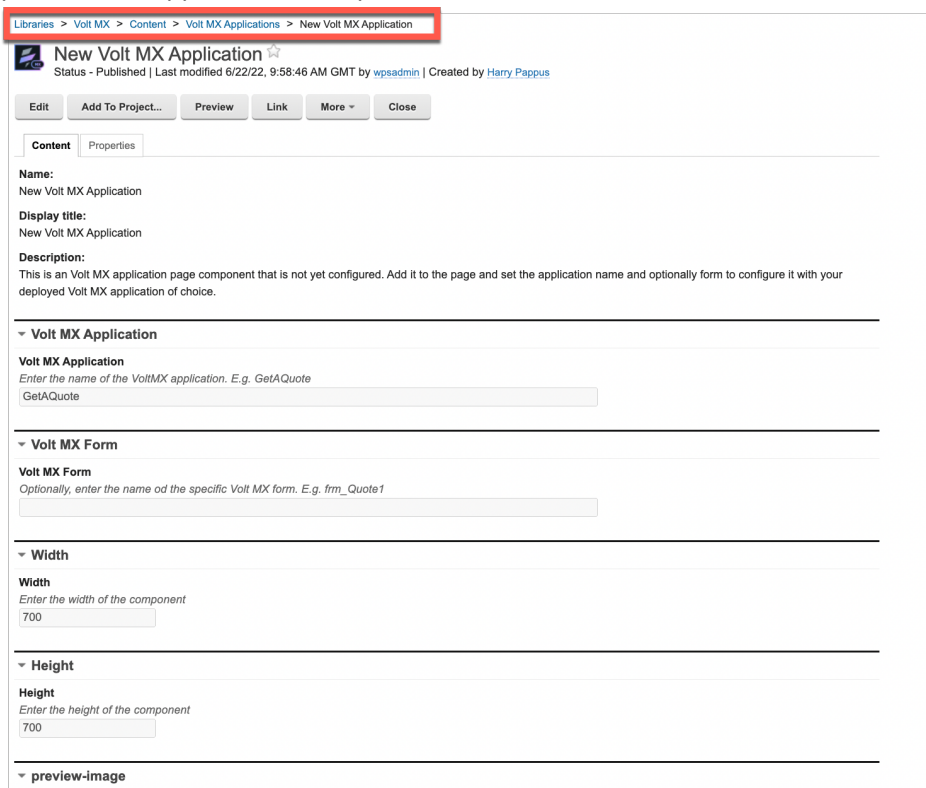
- You may want to set this to work with your own Volt MX server. In this case, edit the **Volt MX** (web content library) – **Content** – **Global Config** (site area) – **Global Config** content and update the **VoltMX Server URL** with your own server.



- Then have a look at the page components that are exposed in the toolbar for your business users, to allow them to easily add any Volt MX application into a DX page. There is a folder for Volt MX Applications, that may look like this.



- You may look at how this **New Volt MX Application** that is preconfigured to show a default application (here GetAQuote) and optional form with a dedicated width and height. You may also update this to your needs. Feel free to reach out to our HCL DX services team to get production support on this implementation.



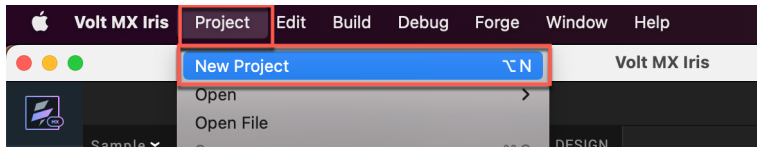
- And have a look at the use Content and Presentation Template for these content items. Feel free to copy and change them. You may involve HCL services to make them production ready.

You have successfully learned how to integrate HCL Volt MX Foundry web applications into DX.

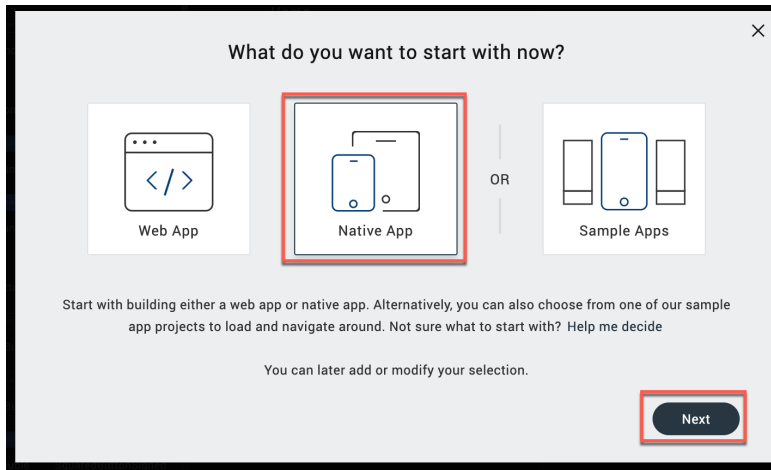
Part 5: Use Volt MX Iris to turn a DX Site into a Native Mobile Application

In this part, you learn how you may use Volt MX Iris to build a native mobile application from any DX site and deploy this to any store, like Apple AppStore and Google Play.

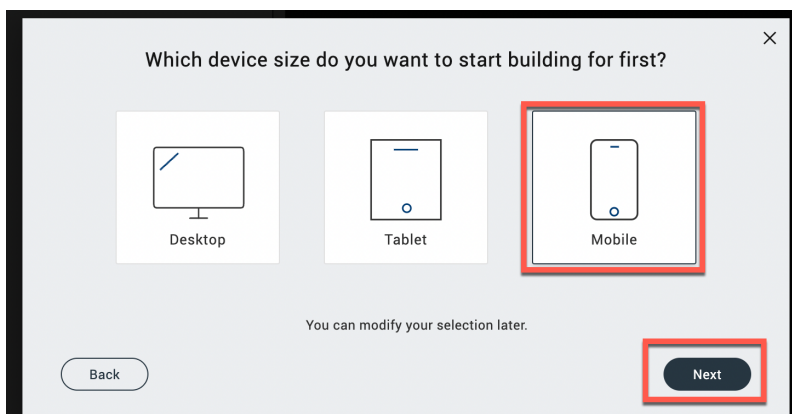
1. Create a new application in Iris. Create a new project. In the menu, open **Project** and click **New Project**.



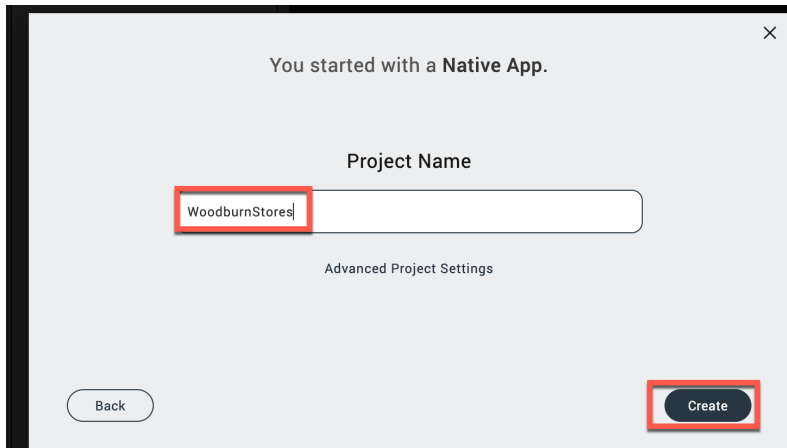
2. Select **Native App** and click **Next**.



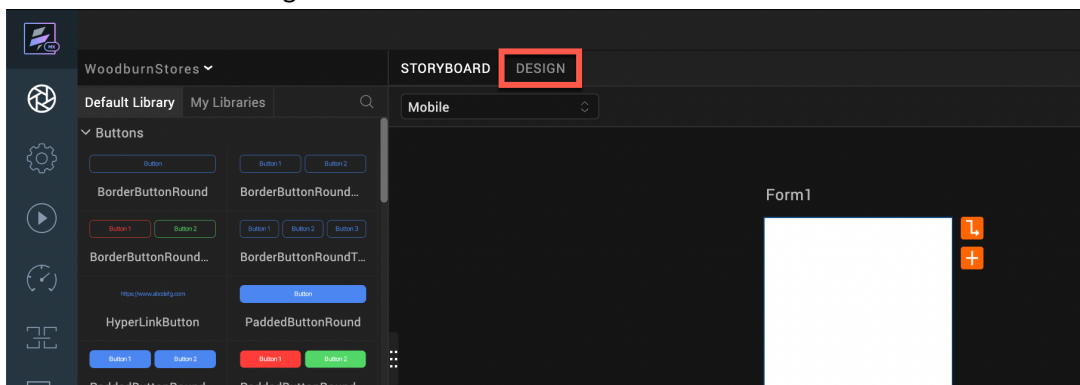
3. Then select the Mobile devices size, as it will directly create you the form for Mobile. Click **Next**.



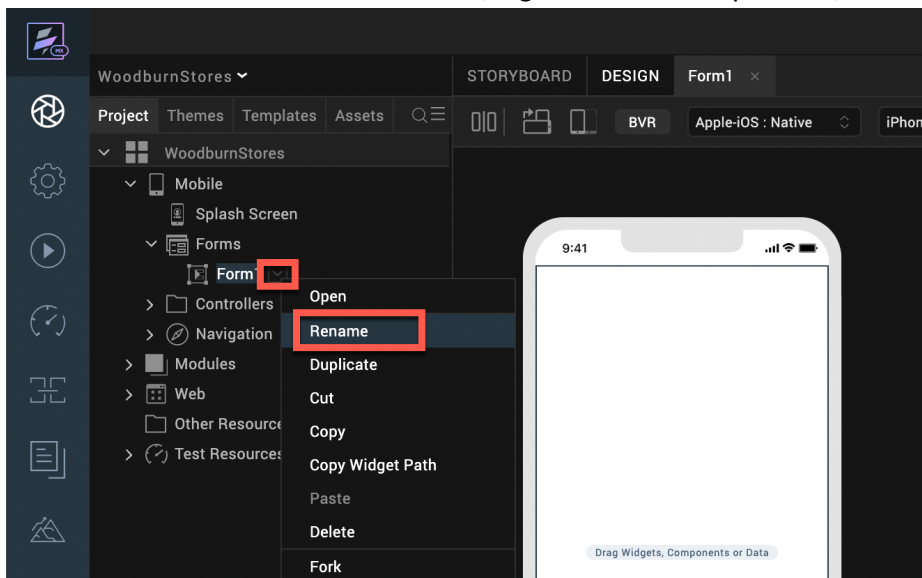
- Name your Iris project, for example **WoodburnStores** and click **Create**.



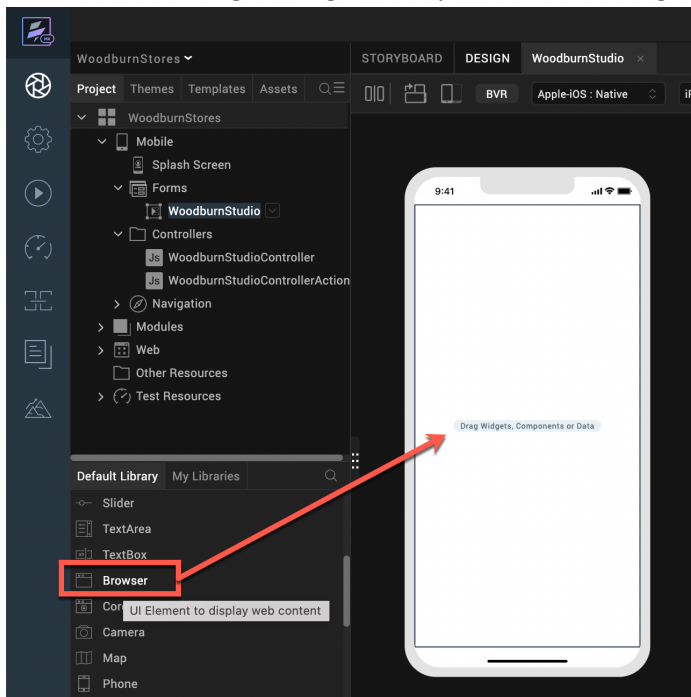
- Then switch to the design mode. Click **DESIGN**.



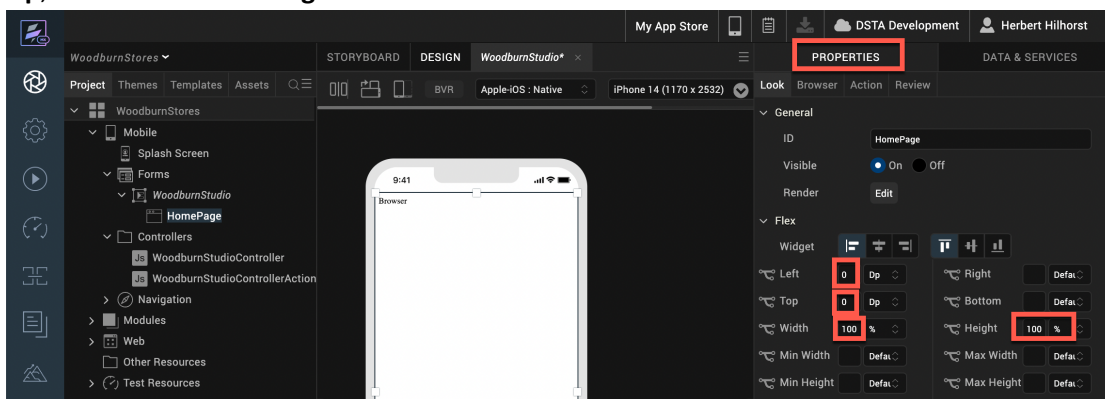
- Then rename the form that was created, e.g. to the name of your site, like Woodburn Stores.



7. Add a browser widget. Drag and drop the Browser widget to your form.

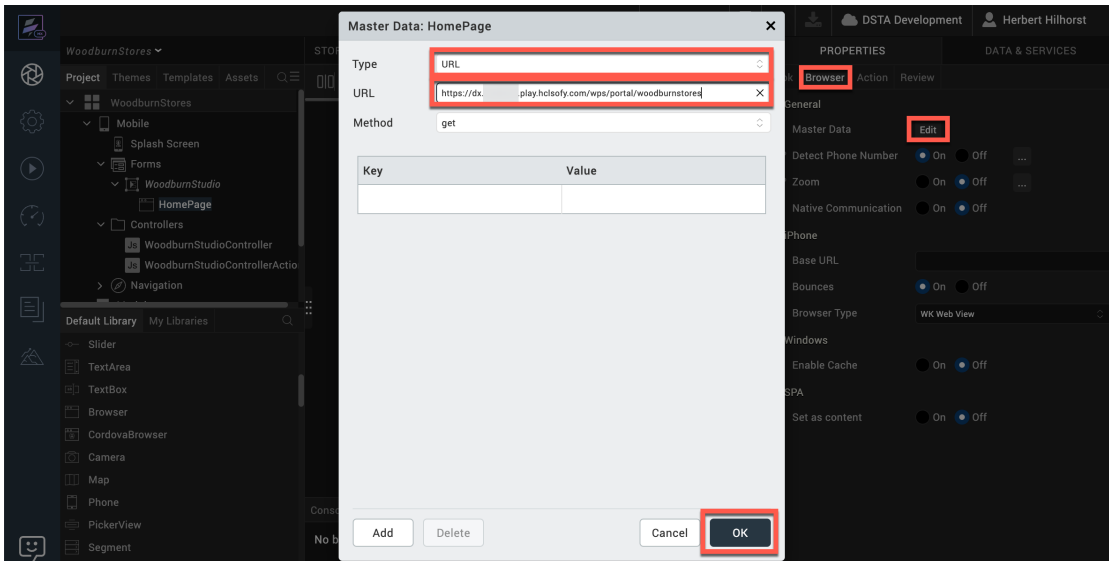


8. Ensure it takes all space of the form. Click **PROPERTIES**, under Flex make **Left 0 Dp**, **Right 0 Dp**, **Width 100%** and **Height 100%**.

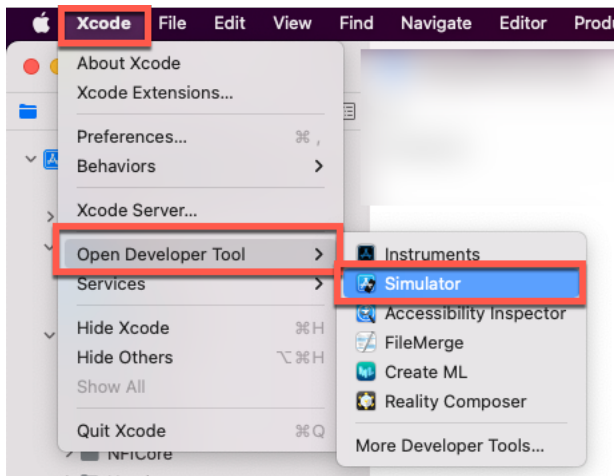


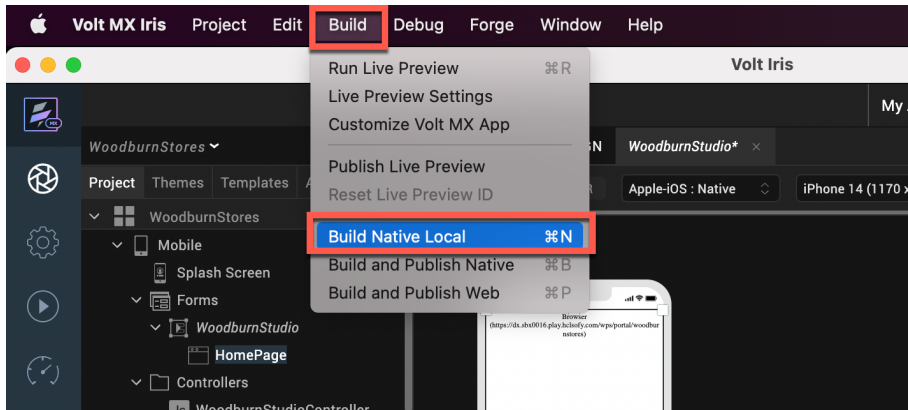
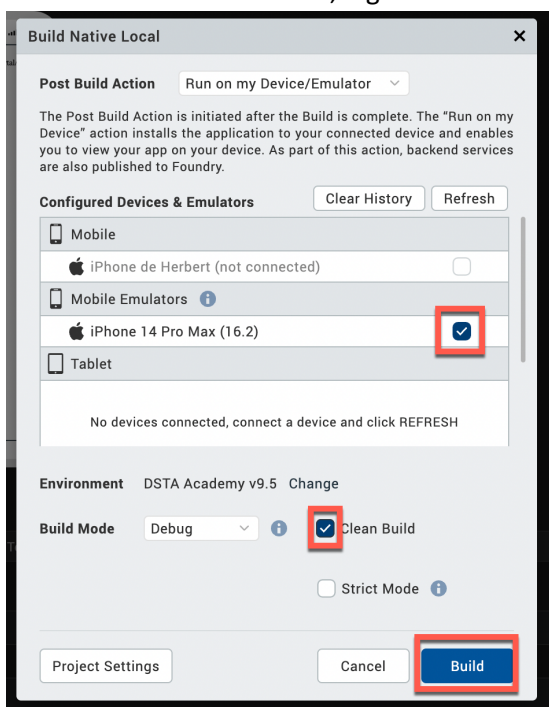
9. Set it to the URL of your site. Under **PROPERTIES**, switch to **Browser** and click **Edit** of **Master Data**. Then switch the **Type** to **URL** and enter the **URL** of your site, e.g. the one of Woodburn Stores (update the URL to your instance) :

<https://dx.sbx0000.play.hclsofy.com/wps/myportal/woodburnstores>.

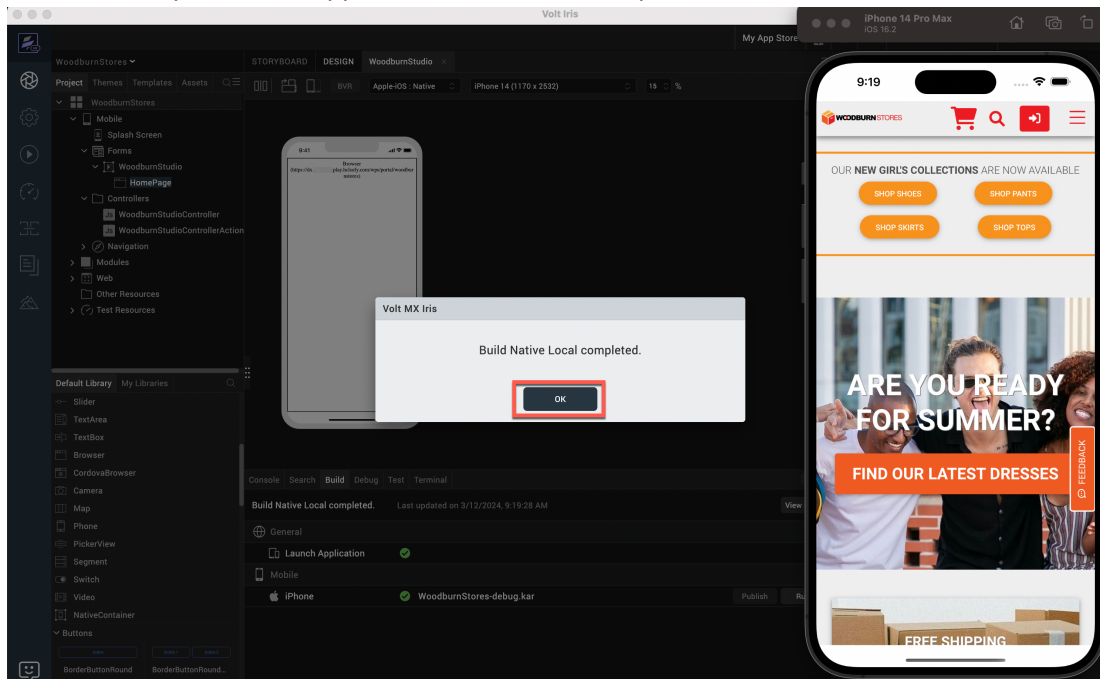


10. Then build your new native application and test it locally. You need to have set up a local emulator. This could be with Xcode <https://developer.apple.com/xcode/> if you have a Mac or Android Studio <https://developer.android.com/studio>. For example, on Xcode, start the Simulator. In the Xcode menu, under **Open Developer Tool**, click **Simulator**.



11. Then run **Build – Build Native Local**.12. Select the mobile emulator, e.g. iPhone 14 Pro Max (16.2). Select Clean Build and click **Build**.

13. It then builds your native application and shows it in your mobile emulator. Click **OK**.



Congratulations. You have successfully turned your DX site into a native mobile application and may get it published on the different stores.

Conclusion

Using this lab tutorial, you have learned how easy it is to use HCL Digital Experience assets like images, videos and files and structured content in your HCL Volt MX applications, and how to use Volt MX to easily integrate its data into DX, using Digital Data Connector and to integrate Foundry web applications into DX sites. And you learned how you may HCL Volt MX Iris to turn your existing DX sites into native mobile applications.

Resources

Refer to the following resources to learn more:

HCL Digital Experience Home - <https://hclsw.co/dx>

HCL Volt MX Home - <https://www.hcltechsw.com/volt-mx>

HCL Digital Experience on HCL SoFy - <https://hclsofy.com/>

HCL Software - <https://hclsw.co/software>

HCL Product Support - <https://hclsw.co/product-support>

HCL DX Product Documentation - <https://hclsw.co/dx-product-documentation>

HCL Volt MX Product Documentation - <https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/index.html>

HCL DX Support Q&A Forum - <https://hclsw.co/dx-support-forum>

HCL DX Video Playlist on YouTube - <https://hclsw.co/dx-video-playlist>

HCL DX Product Ideas - <https://hclsw.co/dx-ideas>

HCL DX Product Demos - <https://hclsw.co/dx-product-demo>

HCL DX Did You Know? Videos - <https://hclsw.co/dx-dyk-videos>

HCL DX GitHub - <https://hclsw.co/dx-github>

Legal statements

This edition applies to version 9.5, release 216 of HCL Digital Experience and HCL Volt MX V9 and to all subsequent releases and modifications until otherwise indicated in new editions.

When you send information to HCL Technologies Ltd., you grant HCL Technologies Ltd. a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

©2024 Copyright HCL Technologies Ltd and others. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with HCL Technologies Ltd.

Disclaimers

This report is subject to the HCL Terms of Use (<https://www.hcl.com/terms-of-use>) and the following disclaimers:

The information contained in this report is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied, including but not limited to the implied warranties of merchantability, non-infringement, and fitness for a particular purpose. In addition, this information is based on HCL's current product plans and strategy, which are subject to change by HCL without notice. HCL shall not be responsible for any direct, indirect, incidental, consequential, special or other damages arising out of the use of, or otherwise related to, this report or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from HCL or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of HCL software.

References in this report to HCL products, programs, or services do not imply that they will be available in all countries in which HCL operates. Product release dates and/or capabilities referenced in this presentation may change at any time at HCL's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. The underlying database used to support these reports is refreshed on a weekly basis. Discrepancies found between reports generated using this web tool and other HCL documentation sources may or may not be attributed to different publish and refresh cycles for this tool and other sources. Nothing contained in this report is intended to, nor shall have the effect of, stating,

or implying that any activities undertaken by you will result in any specific sales, revenue growth, savings or other results. You assume sole responsibility for any results you obtain or decisions you make as a result of this report. Notwithstanding the HCL Terms of Use (<https://www.hcl.com/terms-of-use>), users of this site are permitted to copy and save the reports generated from this tool for such users own internal business purpose. No other use shall be permitted.