

Microsoft Power Apps and Azure

Hands-on Lab

November 2019

Table of Contents

Lab Overview and Objectives	1
Overview	1
Objectives	1
Main components of the lab	2
Exercise 1 – Build the API App	3
Task 1: Download from GitHub	3
Task 2: Test the API.....	4
Exercise 2 – Deploy to Azure.....	6
Task 1: Configure the app service.....	6
Exercise 3 – Create the custom connector	10
Task 1: Save the Open API definition	10
Task 2: Create the connector	11
Exercise 4 – Create the canvas app	17
Task 1: Create a Canvas App with Phone Layout	17
Task 2: Add your custom connector	19
Task 3: Bind Data to the Gallery	20
Task 4: Create the Detail Screen to Edit and Delete Products.....	26
Task 5: Add functionality to Insert new Items.....	29
Task 6: [Optional] Run the same app on your phone (iOS or Android)	31
Lab Disclaimer	32

Lab Overview and Objectives

Overview

Most enterprises have most of their core business data trapped in several silos (legacy databases and apps - Systems of Records). It is essential for digital transformation to break those data silos and make the information trapped in those systems available to employees and business teams as needed.

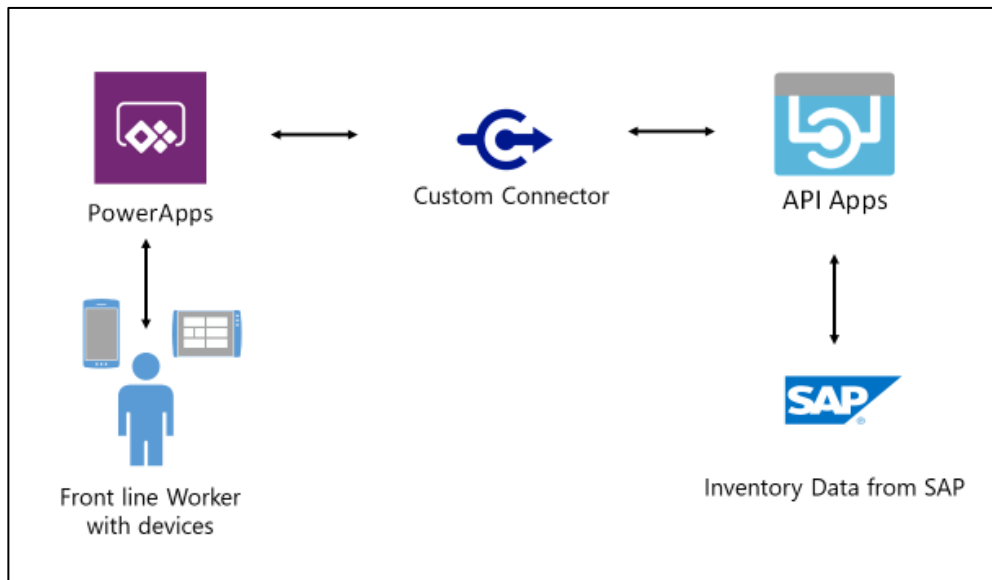
It is also important to modernize the user experience of the enterprise apps to drive productivity. [Microsoft Power Apps](#) along with [Azure](#) could be an effective way to bridge this gap, by enabling an RESTful API layer powered by Azure which could encapsulate and standardize the enterprise data for use with apps + Power Apps as the rapid application development user interface where business apps can be built with agility and in collaboration with the business.

Objectives

This lab has two key objectives:

- To demonstrate how the Microsoft Power Apps platform can help unlock the potential of untapped assets within an Enterprise (legacy APIs, data sources, processes) with a low code / no code approach.
- Learn to create a cross-platform application user experience using Microsoft Power Apps to consume these RESTful APIs through custom connectors

Main components of the lab



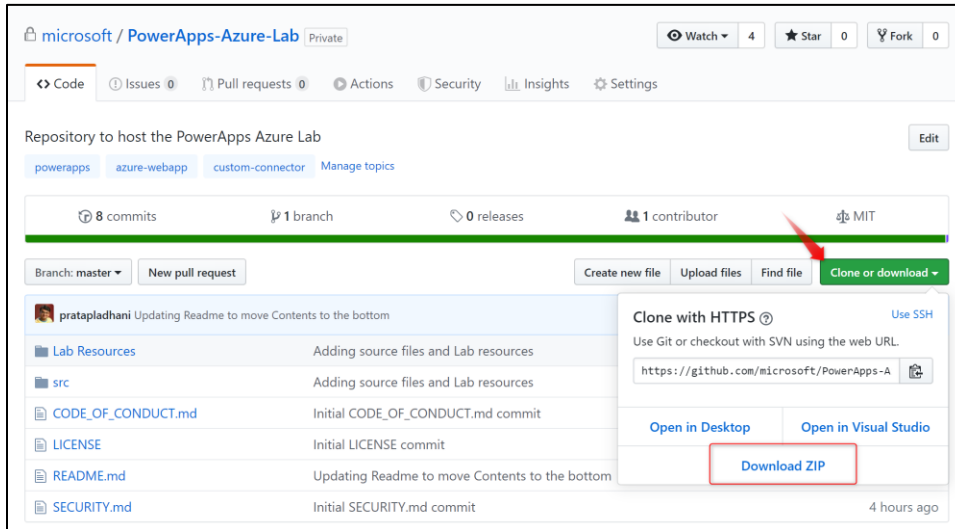
- **Microsoft Power Apps:** The end users consume the apps created from either a web-browser or through the Power Apps mobile apps from iOS or Android
- **Custom Connectors:** Allow the use of RESTful API operations within Microsoft Power Apps
- **API Apps:** An Azure service which allows you to build and host web apps, mobile back ends, and RESTful APIs

Exercise 1 – Build the API App

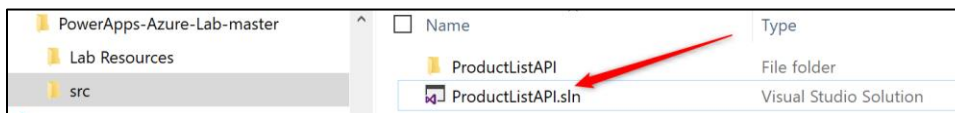
In this exercise, you will download a pre-built API app from a GitHub repo and test it locally before deploying it to Microsoft Azure

Task 1: Download from GitHub

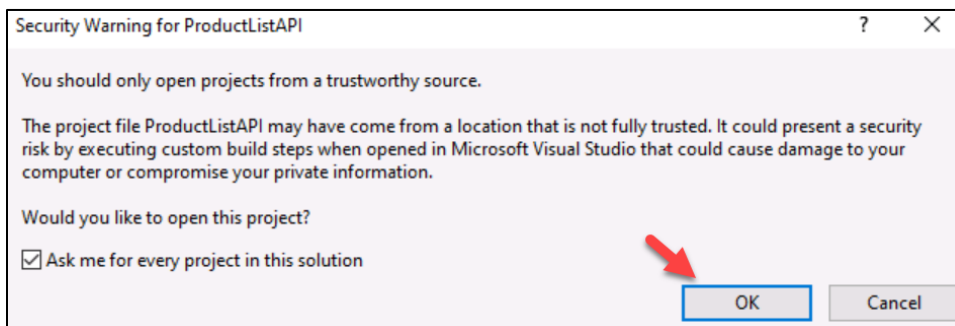
- 1) Navigate to <https://github.com/microsoft/PowerApps-Azure-Lab>
- 2) Click on the **Clone or download** and then **Download Zip**



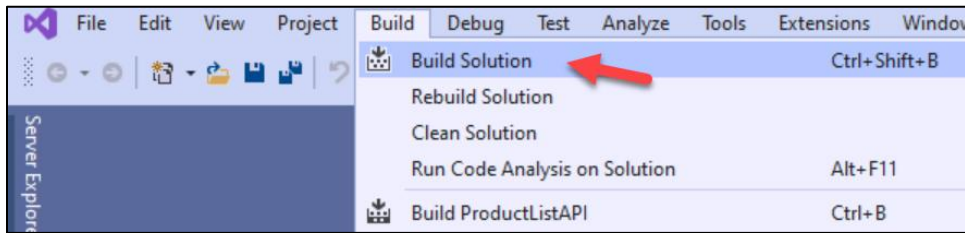
- 3) Extract the contents of the zip file and remember the location
- 4) In the extracted folder, open the “src/ProductListAPI.sln” Solution file in Visual Studio.



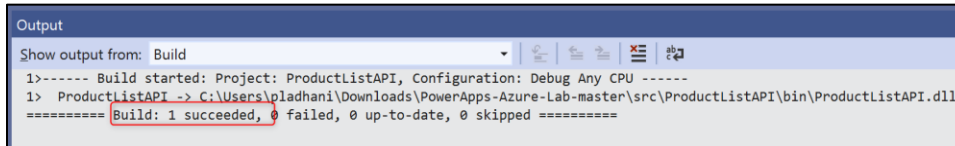
- 5) Select **Visual Studio 2019** and click OK
- 6) If you receive a security warning, click OK for the project as it loads



7) Build the Solution.

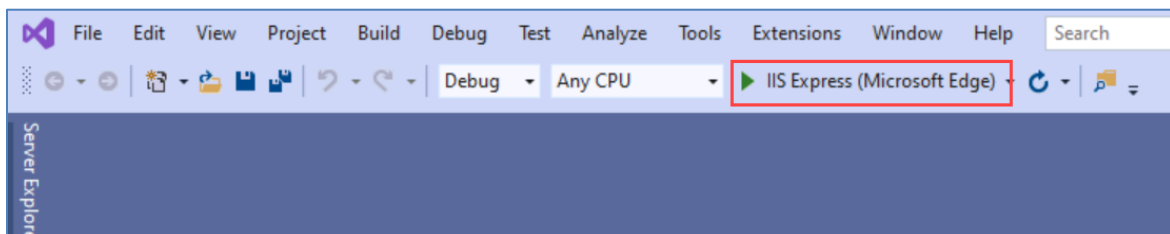


8) Confirm the success of the build and resolve any problems

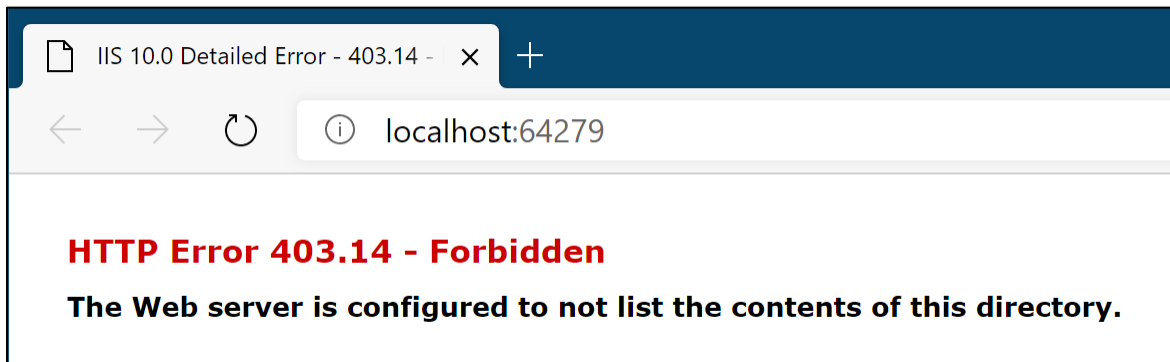


Task 2: Test the API

1) Click the IIS Express (Microsoft Edge) to launch the API app locally for testing



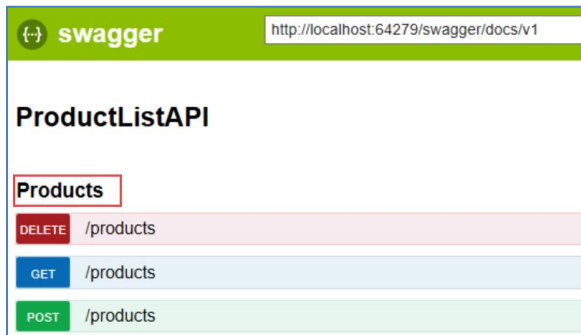
2) A new browser tab will open, and will initially show the following error, that is expected



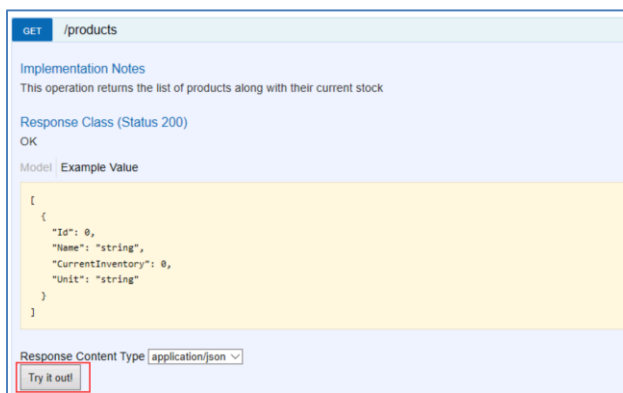
3) Add **/swagger** to the end of the URL and then you should see the following which is the Swagger API Explorer.



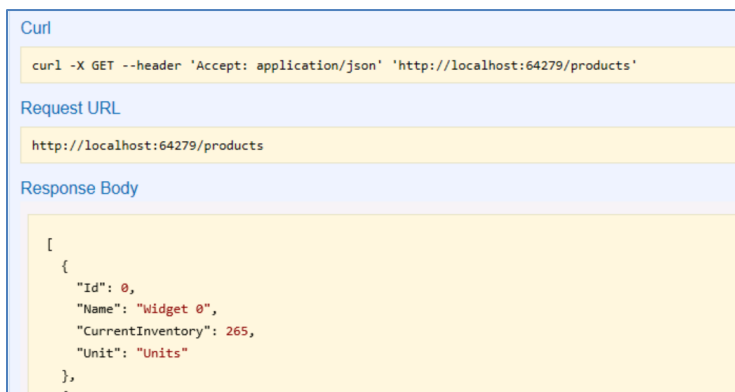
- 4) Click on **Products** to expand the list of actions available to test



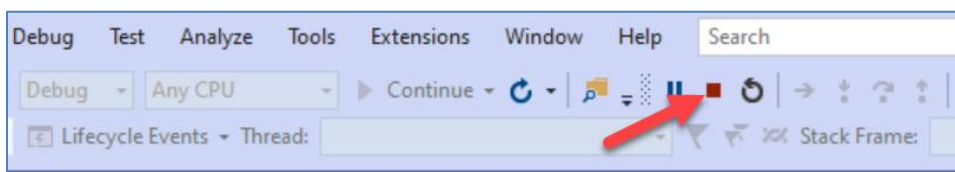
- 5) Click on **GET** and it will expand and show you an example of what the API will return as a response. Click the **Try It out** to run the test



- 6) You should now see the response below the Try it out button and it will contain a list of all the products



- 7) Your API app is working locally, next we will deploy it to Microsoft Azure – Click the Stop button to end the debug session.



Exercise 2 – Deploy to Azure

In this exercise, you will deploy the API app to Microsoft Azure.


Note: You need an Azure Subscription for performing this step.

If you don't have an Azure Subscription, you have the following options:

1. Use your monthly Azure credits included in various Visual Studio Subscriptions. Visit this link for more details: <https://azure.microsoft.com/en-us/pricing/member-offers/credit-for-visual-studio-subscribers/>.
 2. Create a free Azure account from this URL: <https://azure.microsoft.com/en-us/free/>.
 3. If your event team provided you a pre-provisioned resource-group or an Azure Pass – use that for the lab.
-

Task 1: Configure the app service

- 1) Create a new Web App in Microsoft Azure to host your API app, by visiting this URL: <https://portal.azure.com/#create/Microsoft.WebSite>
- 2) Fill in the required information using the following
 1. **Subscription:** If you have multiple azure subscriptions, make sure you are using the right Azure Subscription.

[If you are not seeing your subscription in the Subscription dropdown, chances are that you may be logged in on a different directory. You can switch the directory by selecting the Book and Filter icon on top of the Portal]

 2. **Resource Group:** Select an existing Resource Group or create new one
 3. **Name:** Provide a unique name for your WebApp. This will be part of the URL for your API
 4. **Publish:** Code
 5. **Runtime Stack:** ASP.NET V4.7
 6. **Operating System:** Windows
 7. **Region:** Select the Azure region near to you.
 8. **App Service Plan:** Choose an existing or create a new App Service Plan

Note: You can choose a Free shared infrastructure plan as well for this lab

See the screenshot below for the options for creating the Web App

Home > New > Web App > Web App

Web App

Project Details
Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ 1

Resource Group * ⓘ 2
[Create new](#)

Instance Details

Name * 3 ✓
azurewebsites.net

Publish * 4

Runtime stack * 5

Operating System * 6

Region * 7
ⓘ Can't find your App Service Plan, try a different region.

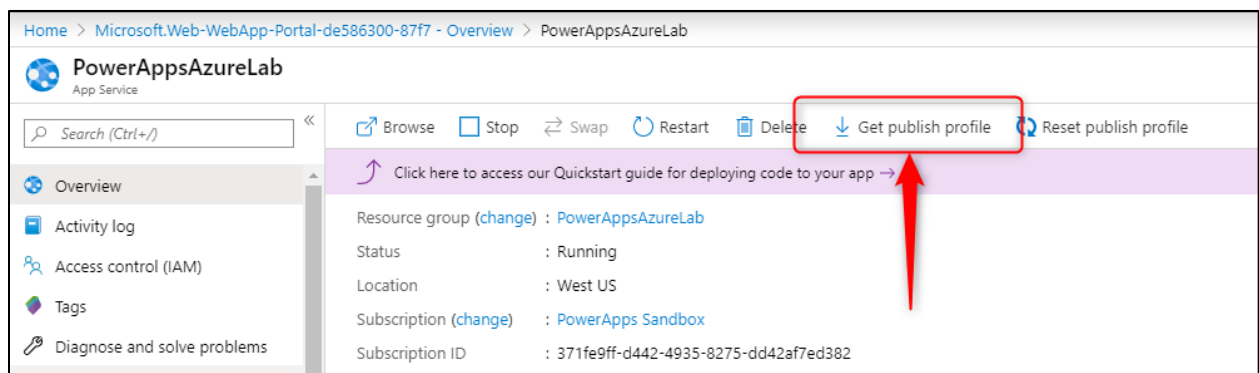
App Service Plan
App Service plan pricing tier determines the location, features, cost and compute resources associated with your app.
[Learn more](#)

Windows Plan (West US) * ⓘ 8
[Create new](#)

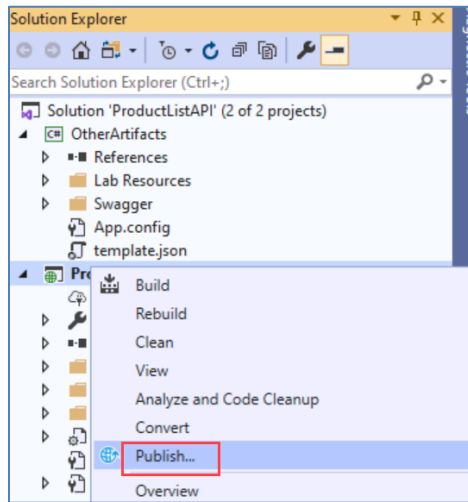
Sku and size *
Free F1
Shared infrastructure, 1 GB memory
[Change size](#)

[Review + create](#) [< Previous](#) [Next : Monitoring >](#)

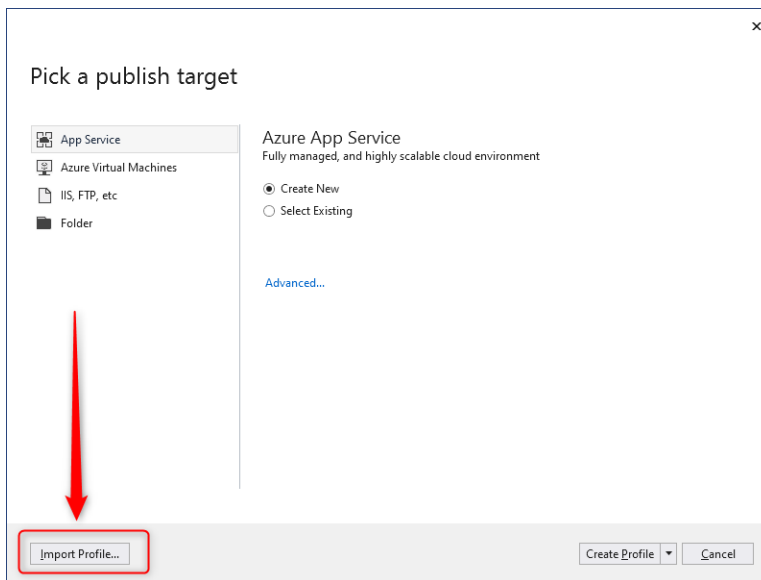
- 3) Wait for the deployment to complete and click the “Go to Resource” to browse to the newly created Web App. Click on the “**Get publish profile**” icon on the top of the service and save it locally on your machine.



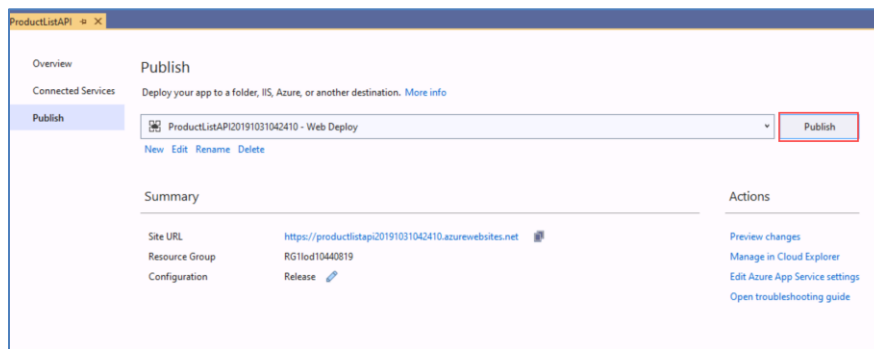
- 4) In Solution Explorer, right click on the **ProductListAPI** project and select **Publish**.



- 5) Select the **Import Profile** button and select the publish profile file that you had downloaded in the previous step



- 6) The App Service should now be ready to publish. Click the **Publish** button



7) Confirm that the publish completed successfully

```

Output
Show output from: Build
2>Adding ACLs for path (ProductListAPI20191031042410)
2>Adding ACLs for path (ProductListAPI20191031042410)
2>Publish Succeeded.
2>Web App was published successfully https://productlistapi20191031042410.azurewebsites.net/
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Publish: 1 succeeded, 0 failed, 0 skipped =====
    
```

8) Click on the URL to your deployed site to launch it in the browser

Summary

Site URL	https://productlistapi20191031042410.azurewebsites.net
Resource Group	RG1lod10440819
Configuration	Release

9) Add `/swagger/ui/index` to the URL just like we did previously

<https://productlistapi20191031042410.azurewebsites.net/swagger/ui/index>

<https://productlistapi20191031042410.azurewebsites.net/swagger>

ProductListAPI

10) You now have the same API deployed to Azure. You can repeat the test we did previously on [Exercise 1: Task 2](#) running the GET on product list if you would like to double check it is working.

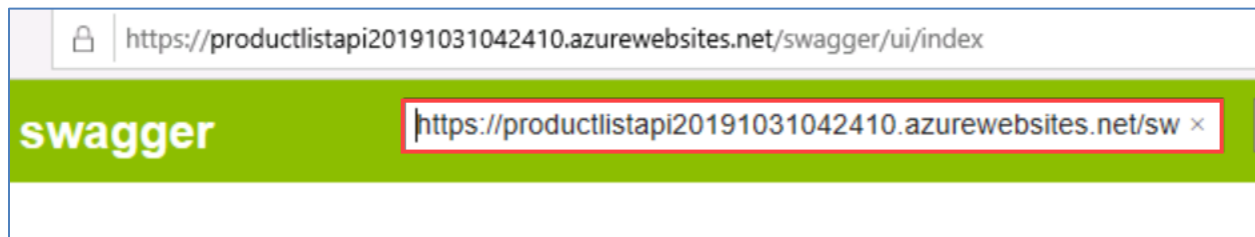
11) Leave the browser tab open as you will use it in the next exercise

Exercise 3 – Create the custom connector

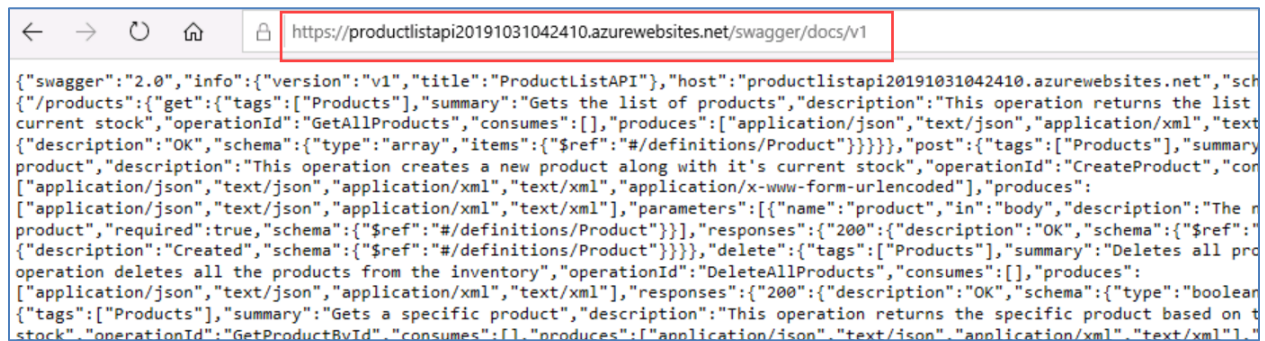
In this exercise, you will create a Power Apps custom connector in PowerApps to use the APIs that we exposed in the previous exercise inside of a Canvas App. Read this article to learn more about Custom Connectors: <https://docs.microsoft.com/en-us/connectors/custom-connectors/>.

Task 1: Save the Open API definition

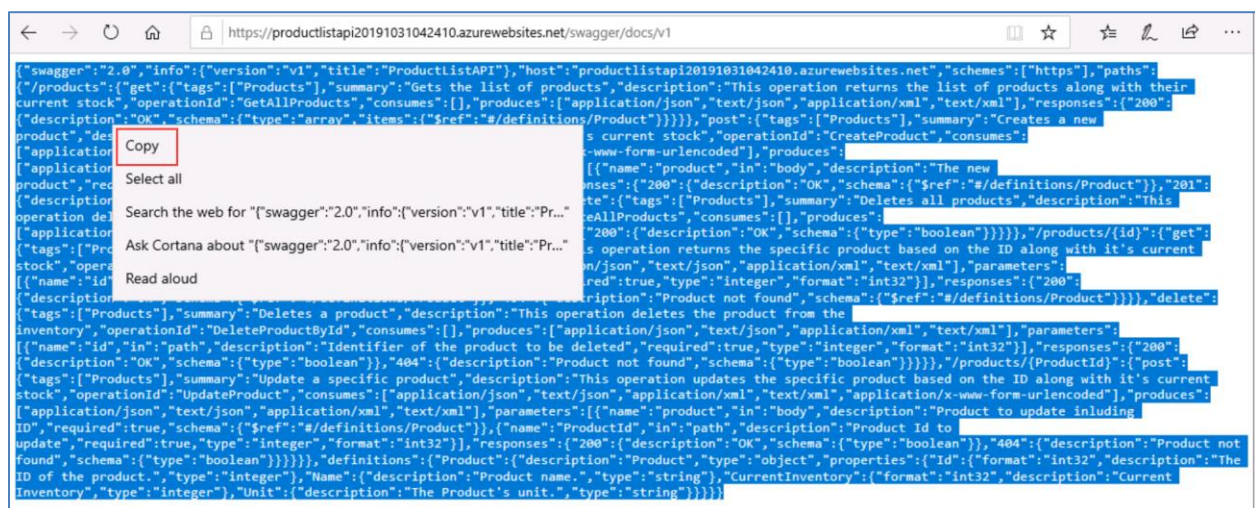
- 1) In the Swagger Explorer on your deployed site, copy the URL for the Open API definition. Make sure you copy the whole URL, some of it is hidden.



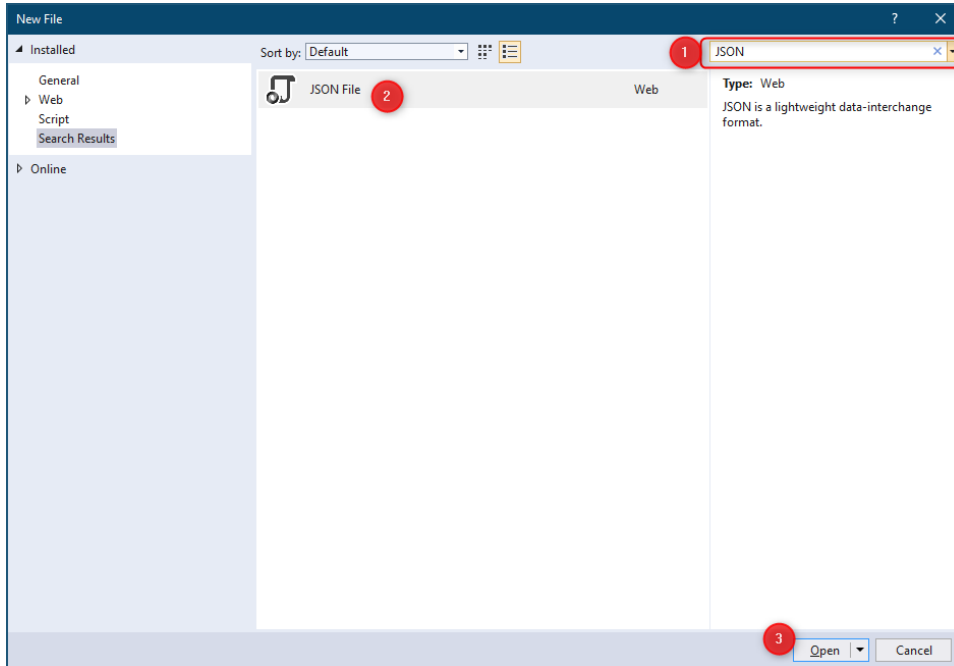
- 2) Paste that URL into the browser address bar and press enter. Your browser should load with the JSON which is the Open API/Swagger definition. We will need this saved to a file to import later.



- 3) Select all of the JSON and then right click and select Copy

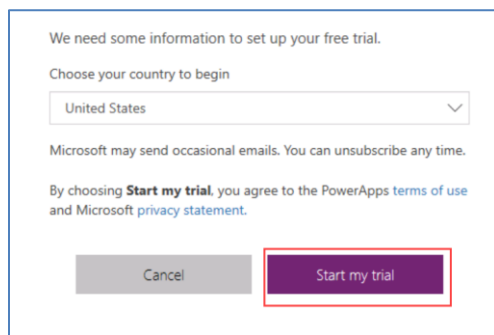


- 4) Create a new JSON file in Visual Studio and press CTRL + A & CTRL+V to replace all the content with the copied JSON definition from the previous step. **Save** the file in the local Documents folder as **ProductsOpenAPI.json**. Make sure to remember where you saved the file as you will need it in the next task..

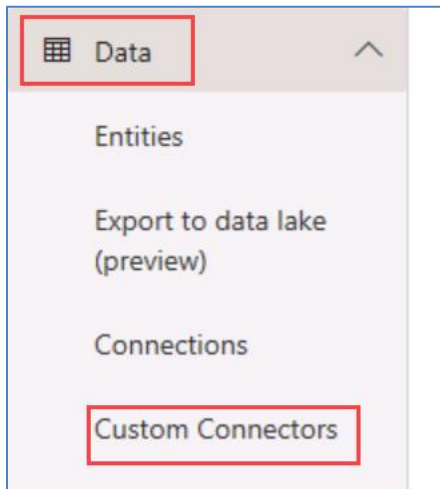


Task 2: Create the connector

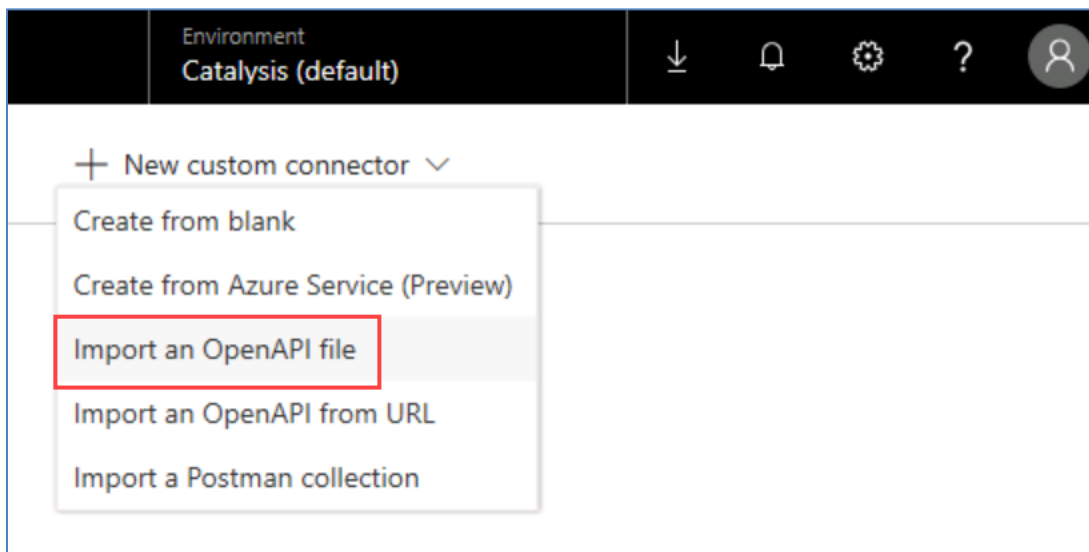
- 1) Navigate to <https://make.powerapps.com>
- 2) Log-in using the user credentials provided as part of your lab environment. Alternatively, you can also choose to use your own Organization credentials (Azure AD / Office 365 login).
- 3) If you don't have a PowerApps License, you will be prompted to start a trial – Click Start my trial



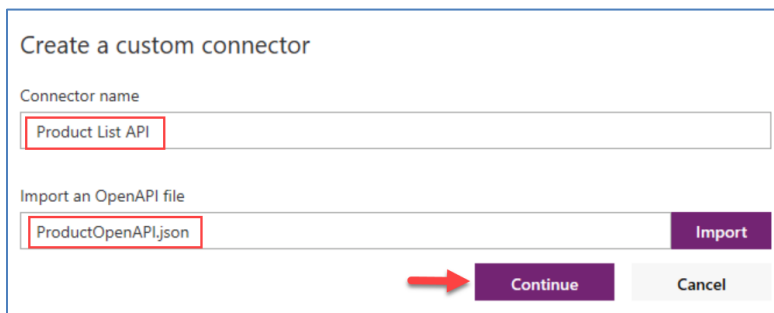
- 4) In the left navigation click **Data** to expand the menu and then click **Custom Connectors**



- 5) In the upper right corner click the **+ New custom connector** and select the **Import an OpenAPI file** option



- 6) Input Product List API in the Connector Name field
7) In the Import an Open API File, click Import and browse to documents and select the ProductOpenAPI.json file you saved in the prior task
8) The information should look like the following and then click Continue

A screenshot of the 'Create a custom connector' form. The 'Connector name' field contains 'Product List API' and the 'Import an OpenAPI file' field contains 'ProductOpenAPI.json'. The 'Continue' button is highlighted with a red arrow. Other buttons visible include 'Import' and 'Cancel'.

- 9) After the import completes the custom connector editor will load and should look like the following

The screenshot shows the 'General information' tab of the Custom Connector Editor for 'Product List API'. The interface includes a breadcrumb trail: 1. General > 2. Security > 3. Definition > 4. Test. At the top right, there are buttons for 'Swagger Editor' (disabled), 'Create connector' (checked), and 'Cancel'. The main content area is divided into two columns. The left column contains instructions: 'Add an icon and short description to your custom connector. Your host and base URL will be automatically generated from the swagger file.' The right column contains the following fields:

- General information** header.
- Icon**: A green square with a white globe icon. Below it is an 'Upload' button. Text: 'Upload connector icon. Supported file formats are PNG and JPG. (< 1MB)'.
- Icon background color**: A text input field with placeholder text 'A color to show behind the icon (e.g., '#007ee5')'.
- Description**: A text input field with placeholder text 'Give your custom connector a short description'.
- Connect via on-premises data gateway**: A checkbox that is unchecked, followed by a link 'Learn more'.
- Scheme ***: Two radio buttons, 'HTTPS' (selected) and 'HTTP'.
- Host ***: A text input field containing 'productlistapi20191031042410.azurewebsites.net'.
- Base URL**: A text input field containing '/'.

 At the bottom right of the panel is a 'Security' button with a right-pointing arrow.

- 10) Input #00753C in the Icon Background Color

- 11) Input Custom connector for Product List

This screenshot shows the same 'General information' tab as the previous one, but with the following updates:

- The **Icon background color** field now contains the value '#00753c'.
- The **Description** field now contains the text 'Custom connector for Product List'.

 The other fields and the overall layout remain the same.

- 12) In the lower left corner of the General Information click Security to advance.

General information

Upload connector icon
Supported file formats are PNG and JPG. (< 1MB)

Icon background color
#00753c

Base URL
/

Security →

- 13) Review the list of authentication options, but leave it set at **No Authentication** and click **Definition** to advance.

***Note:** This is not advisable to use “No authentication” in Production APIs – we are skipping this only for saving time in this lab.*

Refer to this article : <https://docs.microsoft.com/en-us/connectors/custom-connectors/azure-active-directory-authentication> for the guidance for protecting the API endpoint with Azure AD.

Authentication type

Choose what authentication is implemented by your API *

No authentication

Edit

← General Definition →

- 14) Review the connector definition, notice from the OpenAPI file imported it brought in 6 actions. After reviewing click **Create connector**

Connector Name Product List API

1. General > 2. Security > 3. Definition > 4. Test

Swagger Editor Create connector Cancel

Actions (6)
Actions determine the operations that users can perform. Actions can be used to read, create, update or delete resources in the underlying connector.

- 1 GetAllProducts
- 2 DeleteAllProducts
- 3 CreateProduct
- 4 GetProductById
- 5 DeleteProduct
- 6 UpdateProduct

New action

General

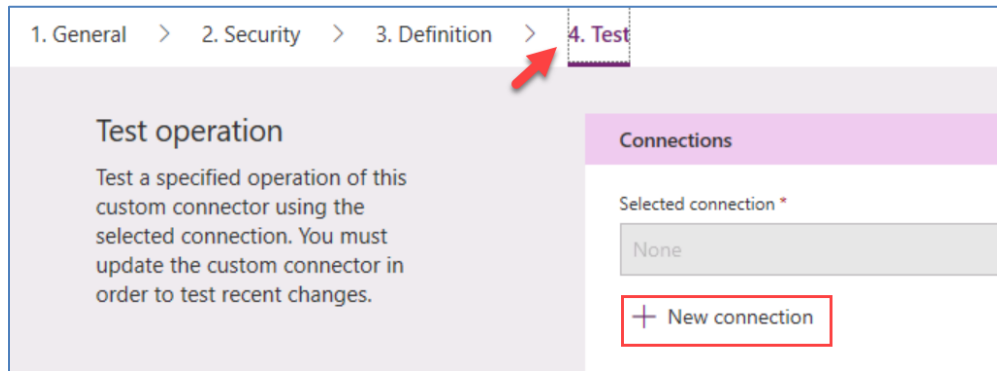
Summary Learn more
Gets the list of products

Description Learn more
This operation returns the list of products along with their current stock

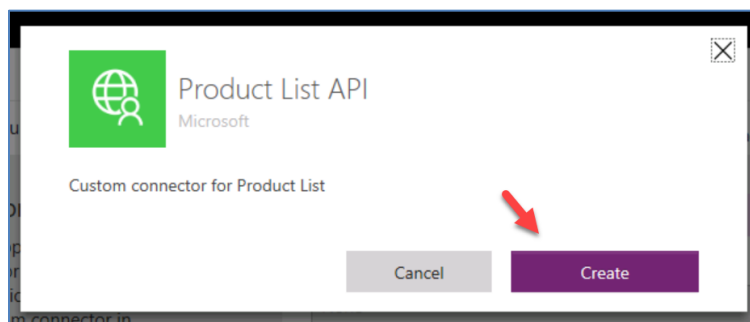
Operation ID *
This is the unique string used to identify the operation.
GetAllProducts

Visibility Learn more
☒ none ☐ advanced ☐ internal ☐ important

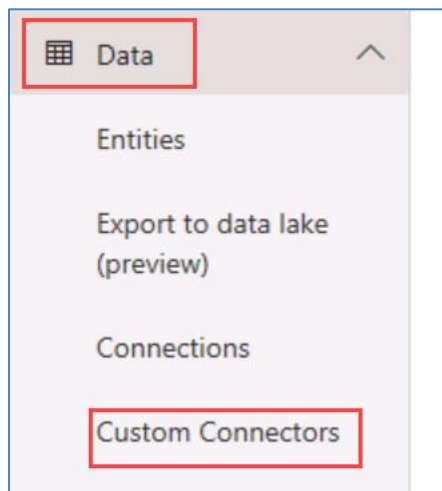
15) Click on the Test tab and then click +New Connection



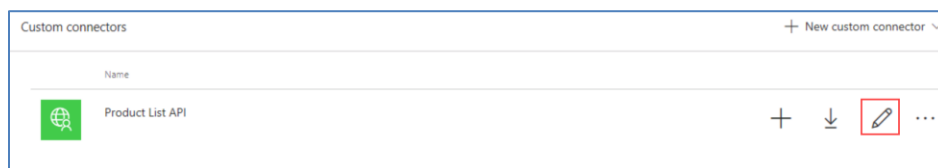
16) Click Create



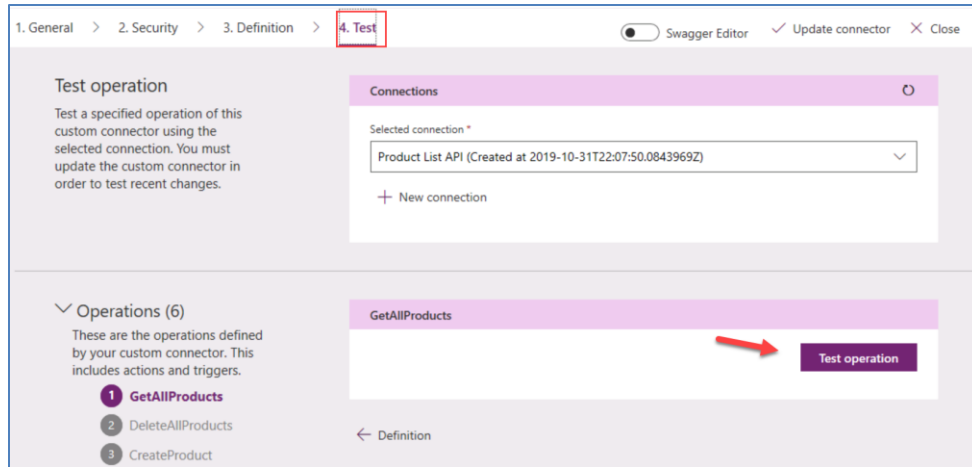
17) After clicking create, you will be navigated to your list of connections. Using the left navigation, navigate back to **Custom Connectors**



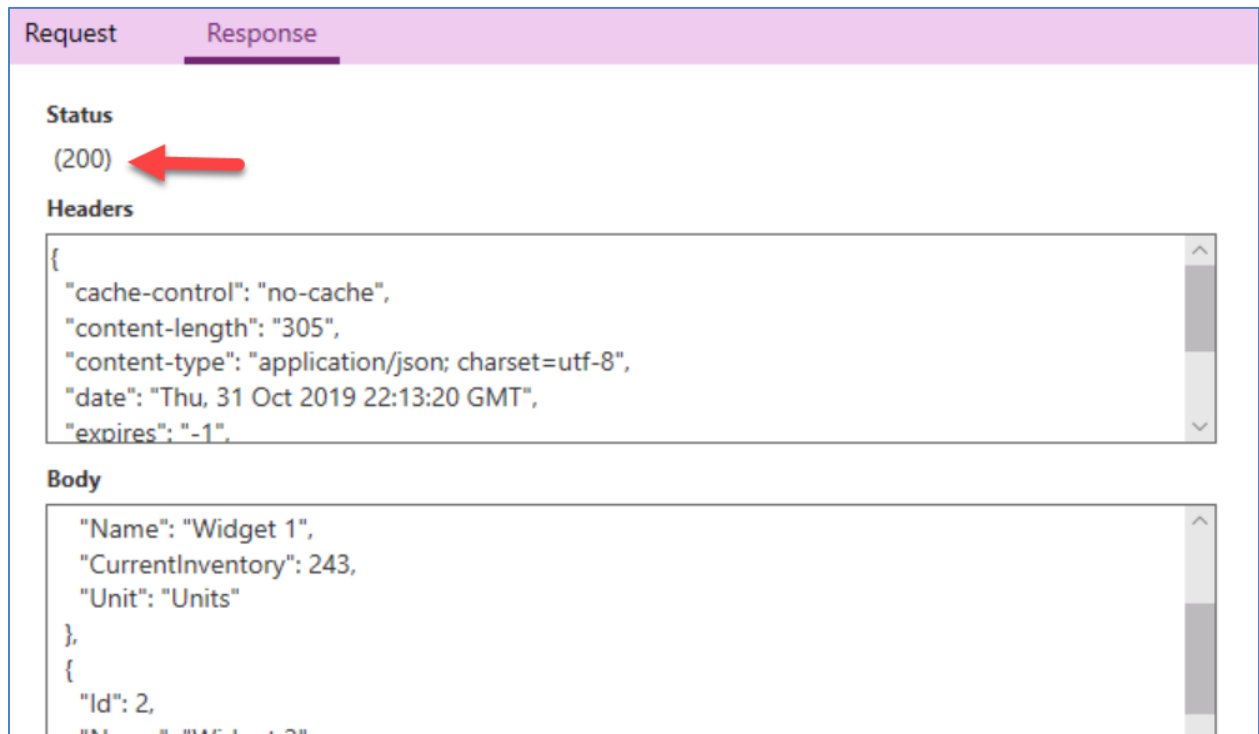
18) Select the custom connector you just created and click the edit icon



19) Change to the Test tab, and then click on the Test Operation for the **GetAllProducts** action



20) Confirm the Status is 200, and you should be able to review the Request and Response



21) You have successfully configured the custom connector.


Exercise 4 – Create the canvas app

Task 1: Create a Canvas App with Phone Layout

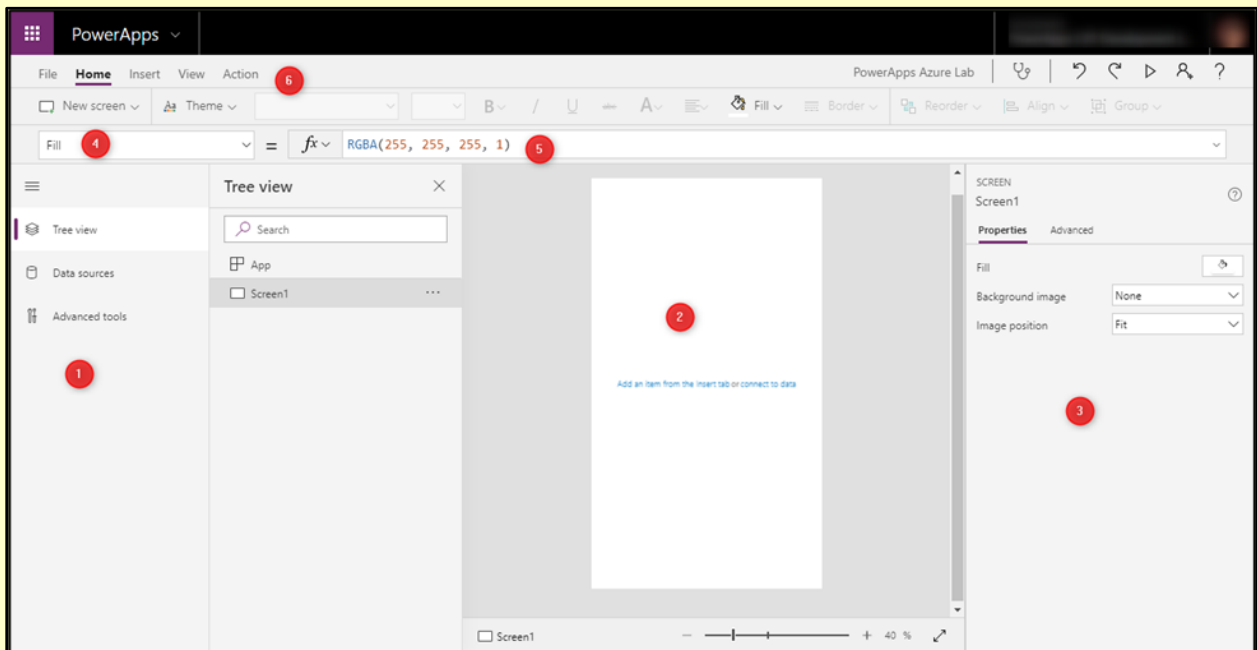
1. Navigate to <http://make.powerapps.com>

Power Apps Canvas Studio Layout (information only – no action required)

Power Apps Canvas Studio is available as a web application (<http://make.powerapps.com>) that you can use in any modern browser.

Power Apps Studio is designed to have a user interface familiar to users of the Office suite. It has three panes and a ribbon that make app creation feel like **building a slide deck in PowerPoint**. Formulas are entered within a function  bar that is like Excel. Studio components:

1. **Left navigation bar**, which shows the Tree view, Data sources & Advanced tools
2. **Middle pane**, which contains the app screen you are working on
3. **Right-hand pane**, where you configure properties for controls, bind to data, create rules, and set additional advanced settings
4. **Property** drop-down list, where you select the property for the selected control that you want to configure
5. **Formula bar**, where you add formulas (like in Excel) that define the behavior of a selected control
6. **Ribbon**, where you perform common actions including customizing design elements

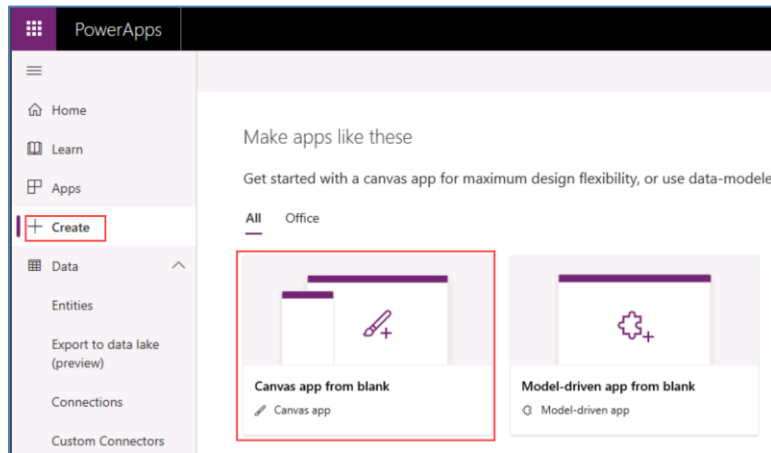


Locale-specific difference in formulas

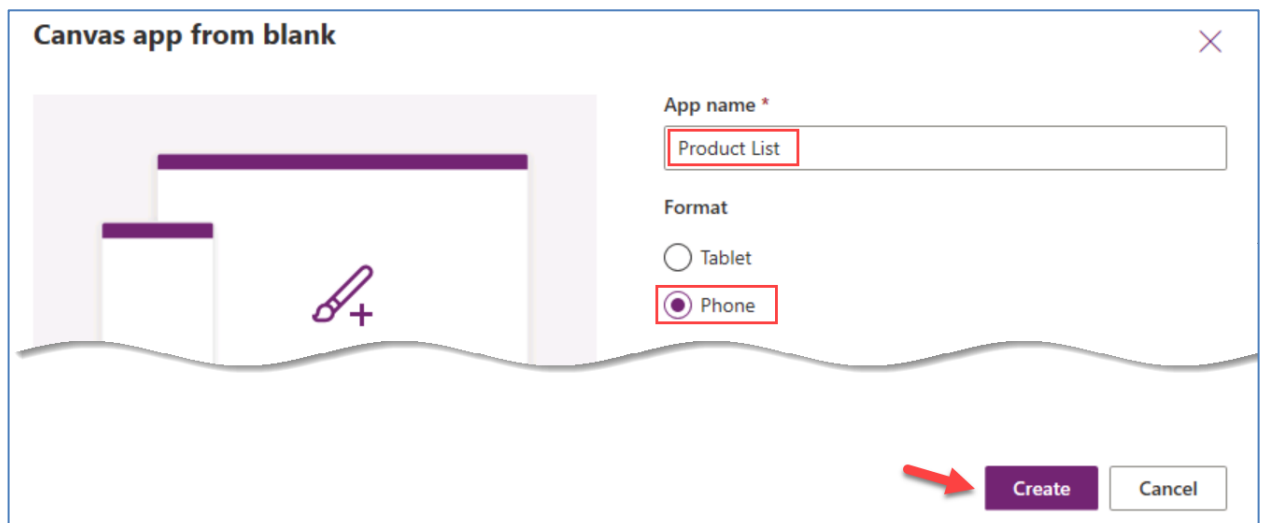
Before you begin, please note that if the primary language set on your browser has its regional settings set to use the comma ‘,’ for its decimal separator (like in much of Europe) your formulas will need to use a semicolon ‘;’ instead of a comma ‘,’ and 2 semicolons ‘;;’ instead of a single semicolon ‘;’ in your formulas. For example:

en-US	<code>ClearCollect(colInventory, ProductListAPI.GetAllProducts())</code>
de-DE	<code>ClearCollect(colInventory; ProductListAPI.GetAllProducts())</code>

2. Select + Create in the left navigation and choose the Canvas app from blank template

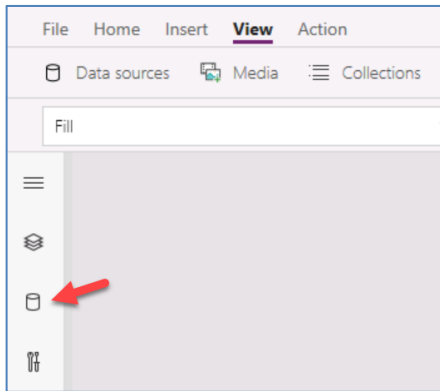


3. Input Product List as the App Name
4. Select Phone for the Format
5. Click Create to create the app

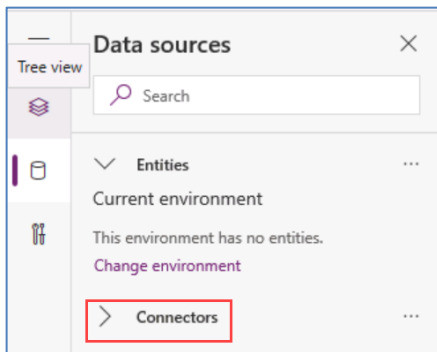


Task 2: Add your custom connector

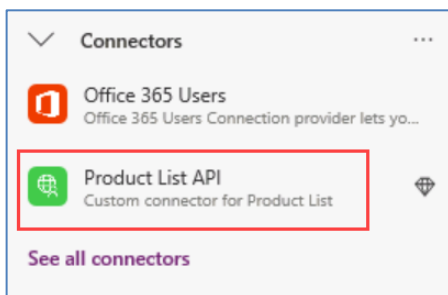
1. In the left tool bar click on the Data Source icon



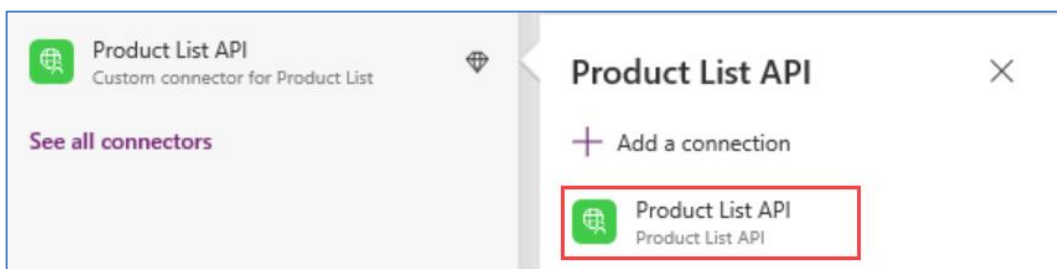
2. Click on > Connectors to expand the section



3. Click on the Product List API connector

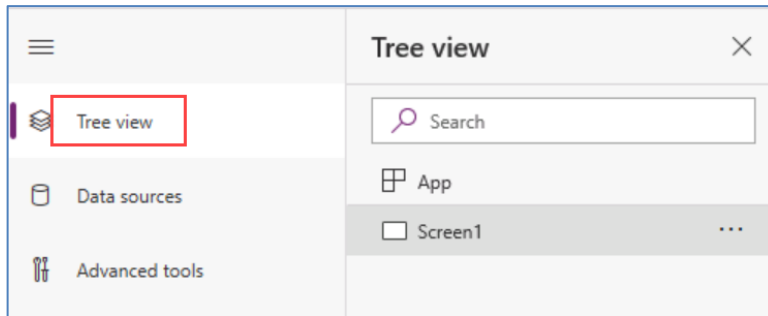


4. Then in the fly out click on Product List API again to associate the existing connection with your app

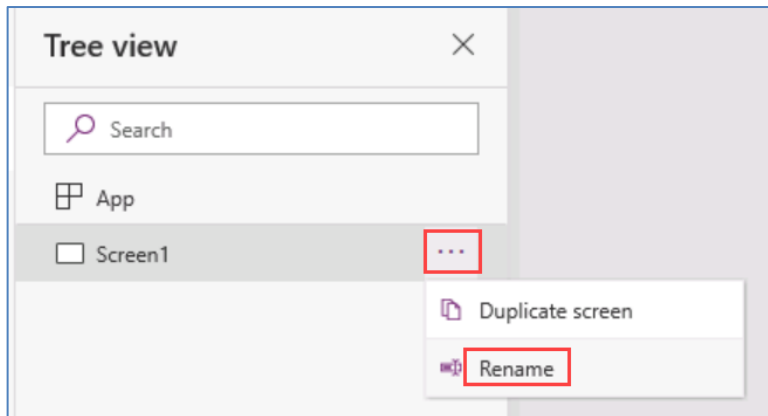


Task 3: Bind Data to the Gallery

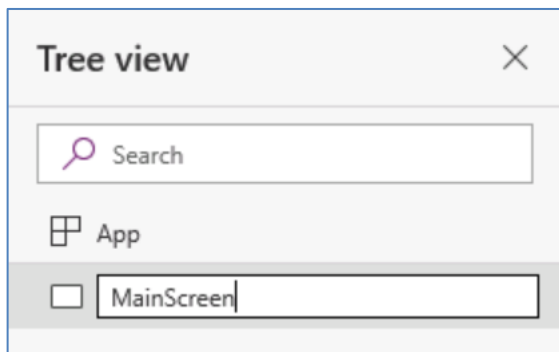
1. Change to Tree view to see the screens in the app



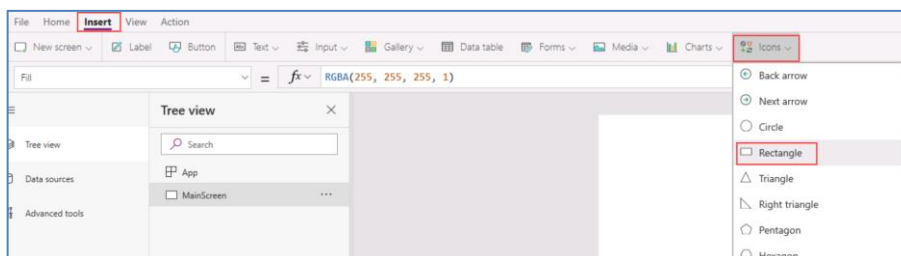
2. In the Tree view click ... next to Screen1 and Select Rename



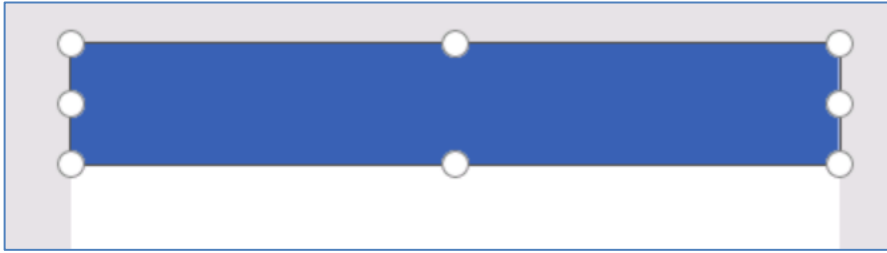
3. Rename Screen1 to MainScreen



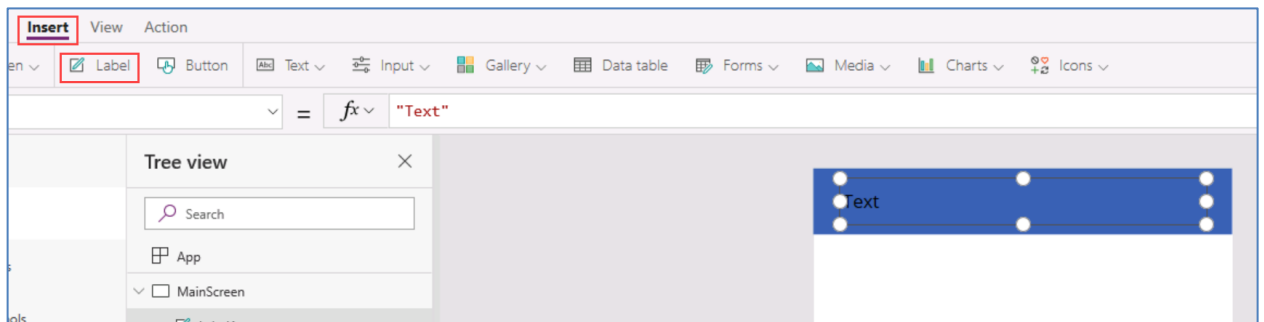
4. Next, we are going to add a header, select Insert -> Icons and choose Rectangle



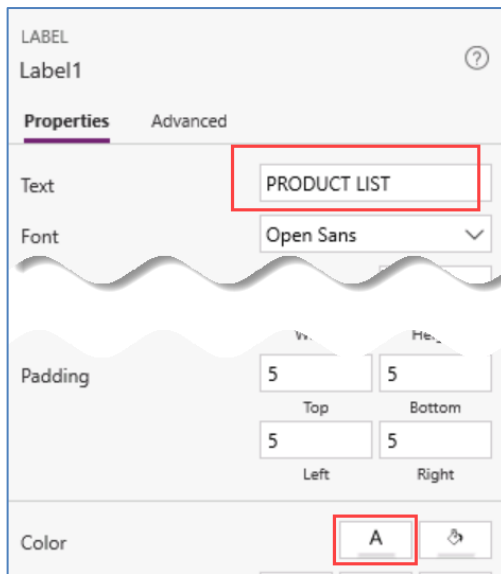
5. Drag the rectangle to the top of the screen and change the size to cover the whole top of the screen



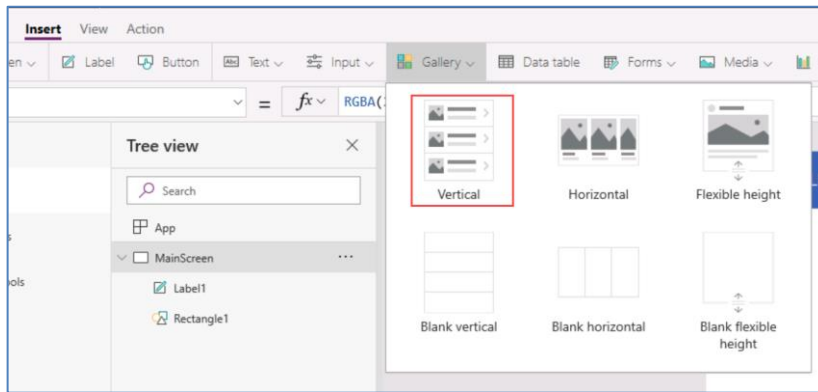
6. Select Insert -> Label and drag it up to be located in the header



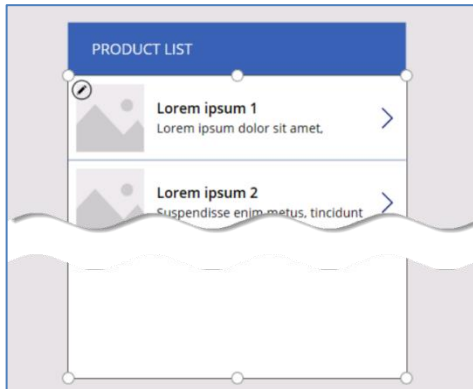
7. With the label selected in the Properties panel on the right change Text to "PRODUCT LIST" and change Color to White



8. Insert a vertical Gallery Control



9. Move and adjust the size of the gallery you inserted to fit in the white space below the header.

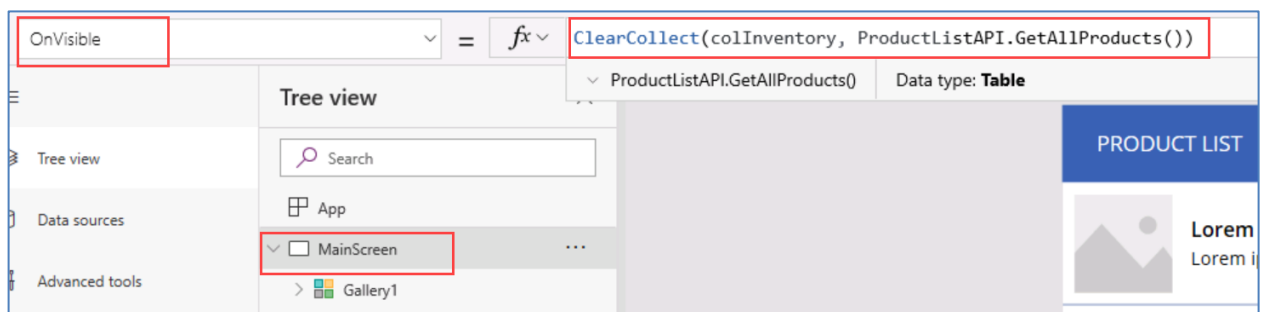


10. Select **MainScreen** in the Tree View, In the property drop down select **OnVisible** and input the following formula. This action will call the GetAllProducts API on the OnVisible event of the Screen and save the response in a local collection called **colInventory**

```
ClearCollect(colInventory, ProductListAPI.GetAllProducts())
```

Use the following formula if you use comma “,” as a decimal separator

```
ClearCollect(colInventory; ProductListAPI.GetAllProducts())
```

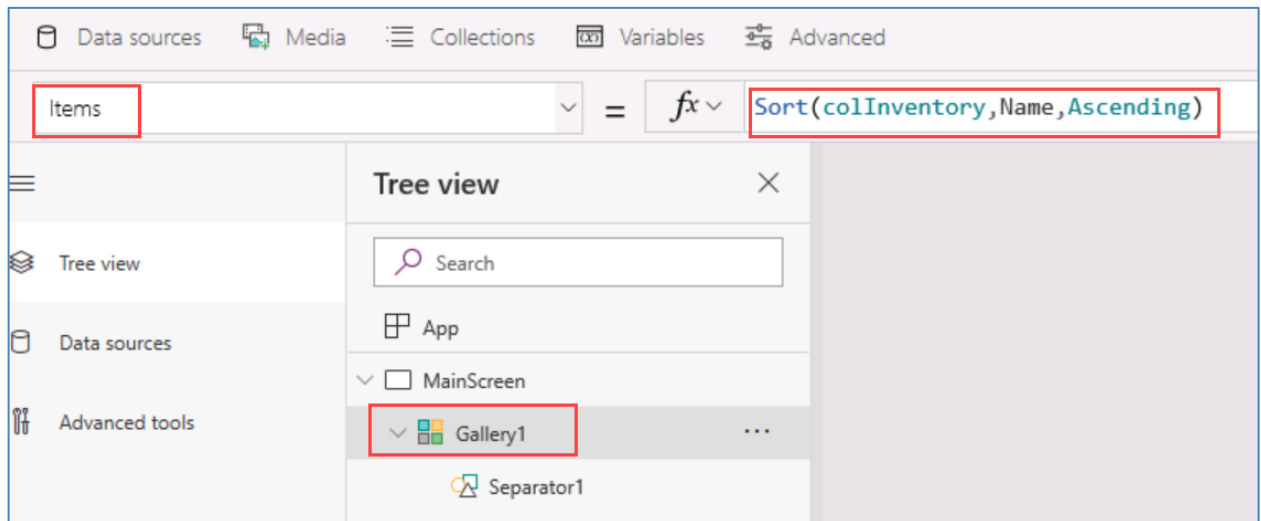


11. Select Gallery1=> Items property and type the following formula:

```
Sort(colInventory, Name, Ascending)
```

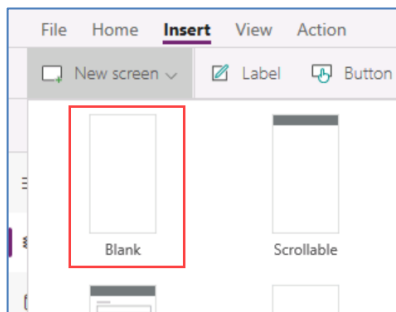
Use the following formula if you use comma “,” as a decimal separator

```
Sort(colInventory; Name, Ascending)
```



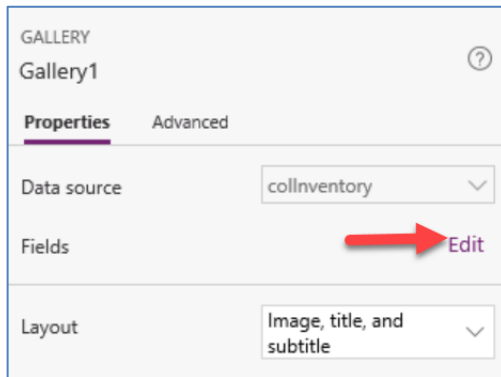
Note: The Gallery won't show data immediately, because the OnVisible event for the MainScreen would not have fired since we are already on this screen. Perform the next 2 steps where Screen2 is Visible and then when we open MainScreen again, it's OnVisible event shall fire and then you should be able to see the gallery populated with data.

12. Create Screen2 that we will use as the detail screen later by selecting **Insert > New Screen** and then the **Blank** screen template

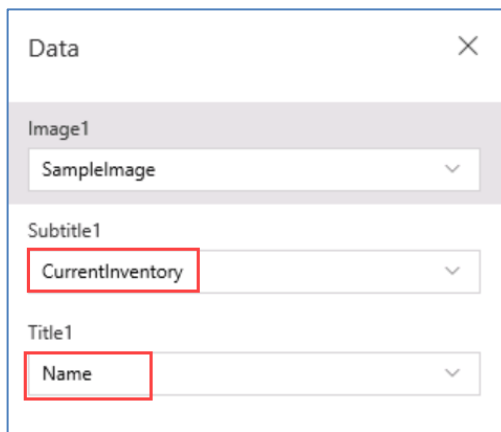


13. In the Tree View select MainScreen again. This will ensure the OnVisible function will be triggered for the MainScreen and the data is loaded to the collection. You should now see data from your API bound to your Gallery.

14. With Gallery1 still selected click on Edit Fields in the right Property panel



15. In the Data panel, Select Name for Title1, CurrentInventory for Subtitle1 and leave SampleImage for the Image1

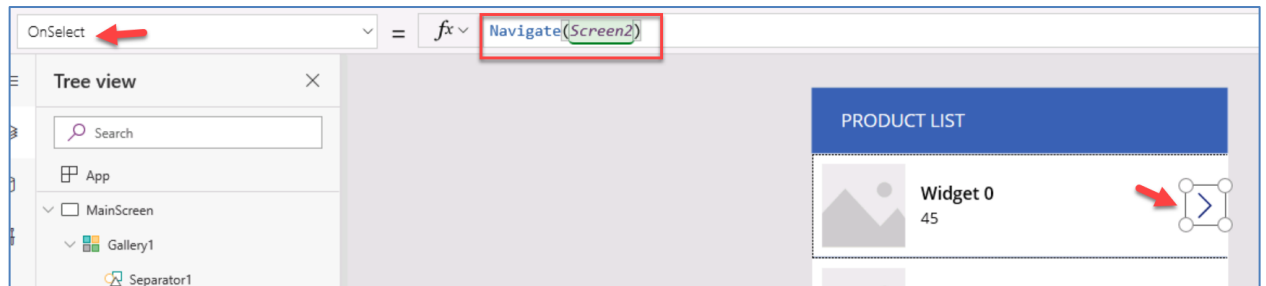


16. You should already start seeing data in the gallery and it should look something like the following image



17. Select the Detail icon on the first item in the gallery and input the following formula in the **OnSelect** property

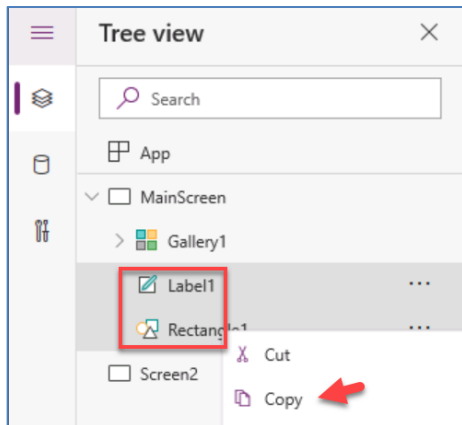
Navigate(Screen2)



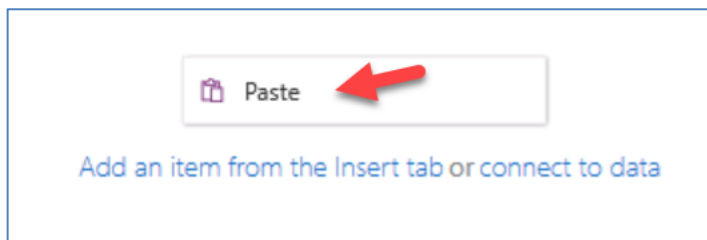
18. Save your app by clicking File -> Save and then clicking Save again in the lower right corner of the screen
19. Click the Back button to return to the app

Task 4: Create the Detail Screen to Edit and Delete Products

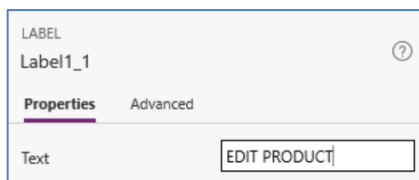
1. While holding the shift key, Select both Label1 and Rectangle1 from the Tree View inside the **MainScreen** group of controls and right-click Copy



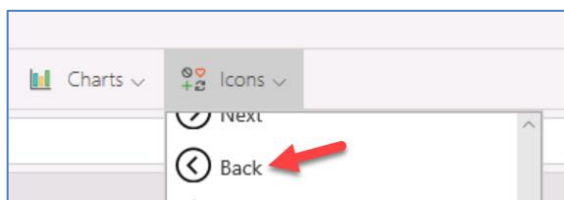
2. Click on Screen2 in the Tree View to make Screen2 the active screen
3. Right click on the screen surface and select Paste



4. While the controls are still selected drag them both to the top of the screen
5. Click on the Title label in the header and change the text in the right property panel to Edit Product



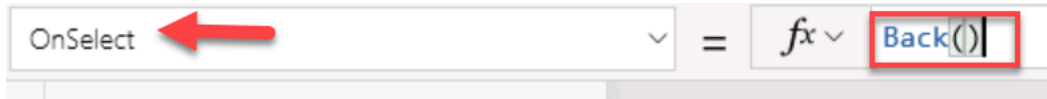
6. Insert a Back icon in the left side of the header



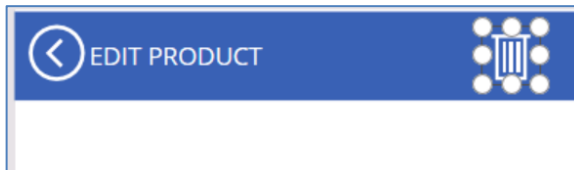
7. If necessary, drag the back icon into position on the left side of the header

8. With the back icon selected, in the right property panel change the Color to White
9. With the back icon still selected, in the property list choose OnSelect and in the formula type

Back()



10. Add Trash icon to the right side of the header to be a delete icon, change the color to white just like you did for the back icon

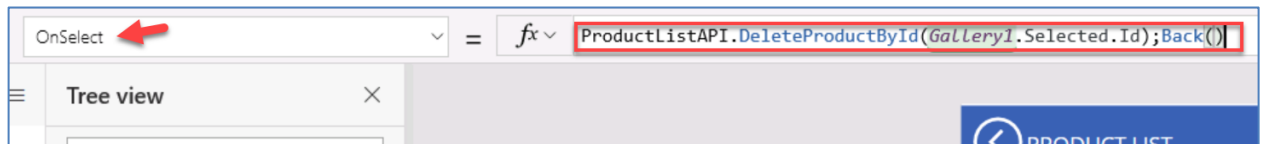


11. In the property selector above the Tree View choose **OnSelect** for icon2 and paste the following formula in the formula bar

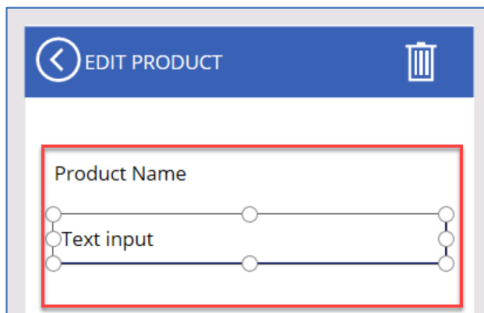
ProductListAPI.DeleteProductById(Gallery1.Selected.Id);Back()

Use the following formula if you use comma “,” as a decimal separator

ProductListAPI.DeleteProductById(Gallery1.Selected.Id);;Back()



12. Insert a label control and change the text to be “**Product Name**”
13. Insert a **TextInput** control right below Product Name label



14. With the **TextInput** control still selected, set the behavior formula for the Default property as the following in the Formula Bar

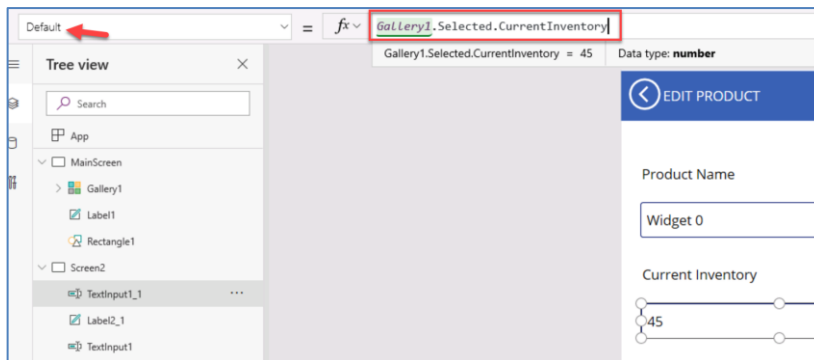
Gallery1.Selected.Name



15. Below the last **TextInput** add a **Label** for “Current Inventory”

16. Add another **TextInput** control below the **label** and set the behavior formula for the Default property to:

`Gallery1.Selected.CurrentInventory`



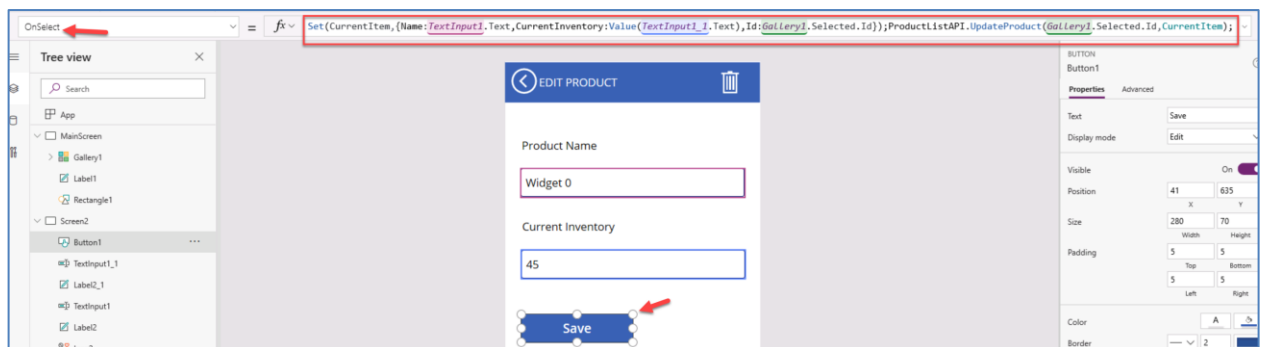
17. Insert a Button Control at the bottom of the screen, change the label to Save

18. Set the OnSelect property formula for the button to be the following formula

```
Set(CurrentItem, {Name:TextInput1.Text,
CurrentInventory:Value(TextInput1_1.Text), Id:Gallery1.Selected.Id});
ProductListAPI.UpdateProduct(Gallery1.Selected.Id, CurrentItem);
```

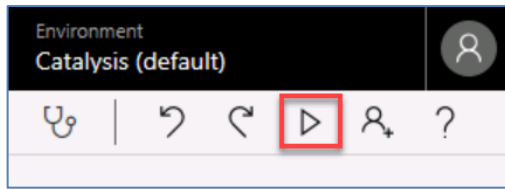
Use the following formula if you use comma “,” as a decimal separator

```
Set(CurrentItem; {Name:TextInput1.Text;
CurrentInventory:Value(TextInput1_1.Text); Id:Gallery1.Selected.Id});;
ProductListAPI.UpdateProduct(Gallery1.Selected.Id; CurrentItem);;
```



19. Save the app, using File -> Save

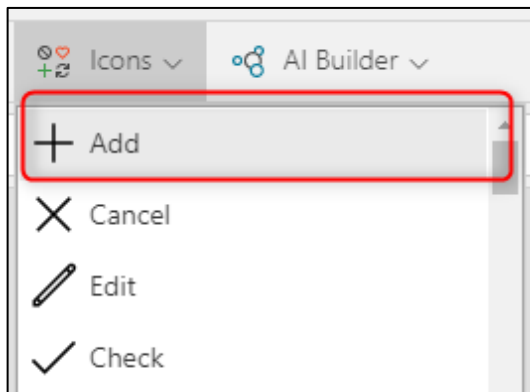
20. Click on the MainScreen in the Tree View to make it the active screen
21. Click the Play button to run the app in the upper right corner



22. Click on the Detail icon on one of the items to navigate to the detail page
23. Click the Delete button, you should return to the list with the item removed
24. Click on the Detail icon on another item
25. Change the values and click Save, then click the Back icon and review the revised value in the list

Task 5: Add functionality to Insert new Items

1. Select MainScreen and insert a Add “+” icon



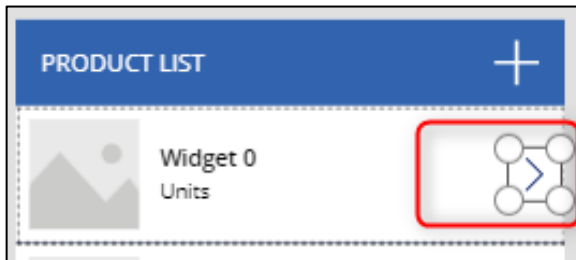
2. If necessary, drag the Add icon into position on the right side of the header
3. With the Add icon selected, in the right property panel change the Color to White
4. With the Add icon still selected, in the property list choose OnSelect and in the formula type. We are setting the CurrentMode variable to “Add” and then navigating to the next screen.

```
Set(CurrentMode, "Add");Navigate(Screen2)
```

Use the following formula if you use comma “,” as a decimal separator

```
Set(CurrentMode; "Add");;Navigate(Screen2)
```

5. Select the greater than icon in the first cell of the Gallery



6. Update the formula for OnSelect as the following which will set the CurrentMode variable to "Edit" and then Navigate to Screen2, when we select that icon

```
Set(CurrentMode,"Edit");Navigate(Screen2)
```

Use the following formula if you use comma "," as a decimal separator

```
Set(CurrentMode;"Edit");;Navigate(Screen2)
```

7. On Screen2, we don't to show the delete icon, if the CurrentMode is "Add". Select Screen2 and select the Trash icon. Select the Visible property and set it to this formula

```
CurrentMode<>"Add"
```

8. We also want to modify the title of the page to "ADD PRODUCT" and the button text to "Add" if the screen is in Add mode. We also don't want the textboxes to show any values from the gallery, if in Add mode.

9. Select the label of the title of the Screen2 and write this formula in the Text Property

```
If(CurrentMode="Add", "ADD", "EDIT") & " PRODUCT"
```

Use the following formula if you use comma "," as a decimal separator

```
If(CurrentMode="Add"; "ADD"; "EDIT") & " PRODUCT"
```

10. Select the Button on Screen2 and write this formula on the Text property

```
If(CurrentMode="Add", "Add", "Update")
```

Use the following formula if you use comma "," as a decimal separator

```
If(CurrentMode="Add"; "Add"; "Update")
```

11. Select the first InputText for the Product Name, change the formula on the Default property to the following

```
If(CurrentMode="Add", "", Gallery1.Selected.Name)
```

Use the following formula if you use comma "," as a decimal separator

```
If(CurrentMode="Add"; ""; Gallery1.Selected.Name)
```


12. Same way, select the second InputText for the Current Inventory, change the formula on the Default property to the following

```
If(CurrentMode="Add", "", Gallery1.Selected.CurrentInventory)
```

Use the following formula if you use comma “,” as a decimal separator

```
If(CurrentMode="Add"; ""; Gallery1.Selected.CurrentInventory)
```

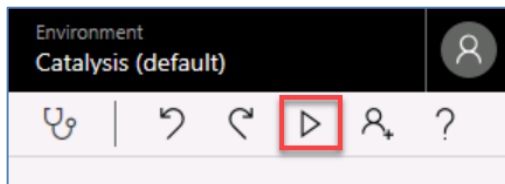
13. Also, we want to call the CreateProduct API instead of the UpdateProduct API if we are in Add mode. While the Button is selected, update the formula to the following for the OnSelect property

```
Set(CurrentItem, {Name:TextInput1.Text,  
CurrentInventory:Value(TextInput1_1.Text), Id:Gallery1.Selected.Id});  
If(CurrentMode="Add", ProductListAPI.CreateProduct(CurrentItem),  
ProductListAPI.UpdateProduct(Gallery1.Selected.Id, CurrentItem));
```

Use the following formula if you use comma “,” as a decimal separator

```
Set(CurrentItem, {Name:TextInput1.Text;  
CurrentInventory:Value(TextInput1_1.Text); Id:Gallery1.Selected.Id});;  
If(CurrentMode="Add"; ProductListAPI.CreateProduct(CurrentItem);  
ProductListAPI.UpdateProduct(Gallery1.Selected.Id; CurrentItem));;
```

14. Click on the MainScreen in the Tree View to make it the active screen
15. Click the Play button to run the app in the upper right corner



16. Click on the Add icon on the top to navigate to the add page
17. Click the Delete button, you should return to the list with the item removed
18. Click on the Detail icon on another item
19. Change the values and click Save, then click the Back icon and review the revised value in the list

Task 6: [Optional] Run the same app on your phone (iOS or Android)

1. Download the PowerApps client app from the iOS or Android store or from the links below
 - iOS: <https://aka.ms/powerappsios>
 - Android: <https://aka.ms/powerappsandroid>
2. Login on the app using the same credentials used for this lab and run the app

Lab Disclaimer

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/Usage/General.aspx> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.