Power Apps Foundations

Power Apps – Foundations

All rights reserved. Printed in Canada. No part of this material protected by this copyright noticed may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, scanning or by information storage and retrieval systems without written permission from the copyright owner.

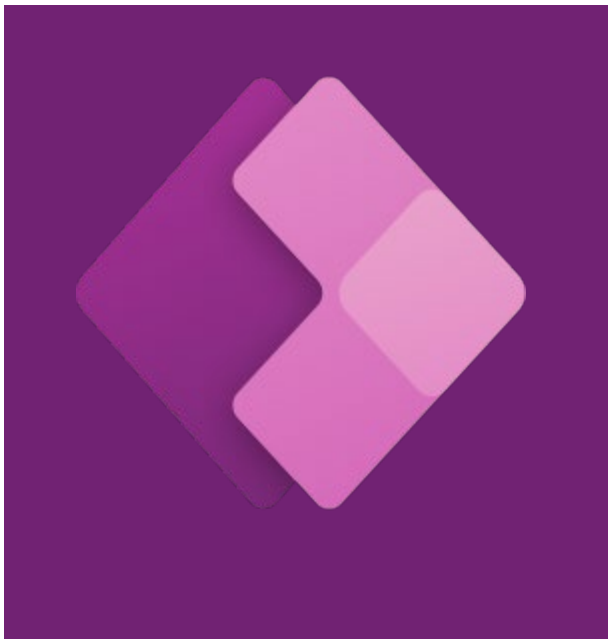
Microsoft is a registered trademark of Microsoft Corporation in Canada and/or other countries. This title is an independent publication not affiliated with Microsoft Corporation. Other product and company names mentioned herein may be the trademarks of their respective owners.

Any and all example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company organization, product, person or event is intended or inferred.

All screenshots used with permission from Microsoft.

Manufactured and printed in Canada

Copyright 2023 Alpha One Media Corp. Ver 2.0



Power Apps Foundations

Robin Solonenko
ALPHA ONE MEDIA CORP AIRDRIE, ALBERTA

Table of Contents

Lesson 1: Power Apps Concepts 5
What is Microsoft's Power Apps? 5
Features of Power Apps 7
Environment Setup 8
Connecting to Data Sources 9
Connecting to OneDrive 9
Delegation in a Canvas App 14
Connecting to SharePoint List 14
Connecting to SQL Cloud 18
The Panes of Power Apps 23
The Tree View Pane 23
The Properties / Advanced Panes 24
Type of Applications 25
The Cost of Power Apps 27
Saving & Publishing Your Apps 27
Lesson 2: Canvas App Concepts 31
Understanding the Gallery 31
Forms & Data Cards 36
Modifications for Detail and Edit Screens 41
Learning Triggers 45
Navigating Your Screens 46
Running Your App on Smart Phone 47
Lesson 2 Exercise 49
Lesson 3: Building a Canvas App 71
App From Scratch 71
Master Screens 72
Building an Object Library 74
Creating Buttons 76
Attaching Galleries to Screens 78
Creating Form Galleries 81
Adding Navigation Features 84

Search & Filter Functions 89
Data Validation 93
Multiple Data Sources 95
Using Data Tables 98
Lookup Function 100
Adding Call Functionality 101
Lesson 3 Exercise 103
Lesson 4: Advancing Your Canvas App 119
Useful Functions 119
Grouping 122
Cascading Drop-Down Lists 123
Deleting Records 126
Creating Popups 127
Input Elements 129
Media Elements 136
Charts & Graphs 141
Lesson 4: Exercise 145
INDEX 157

POWER APPS FOUNDATIONS

LESSON 1

Power Apps Concepts

LEARNING OBJECTIVES

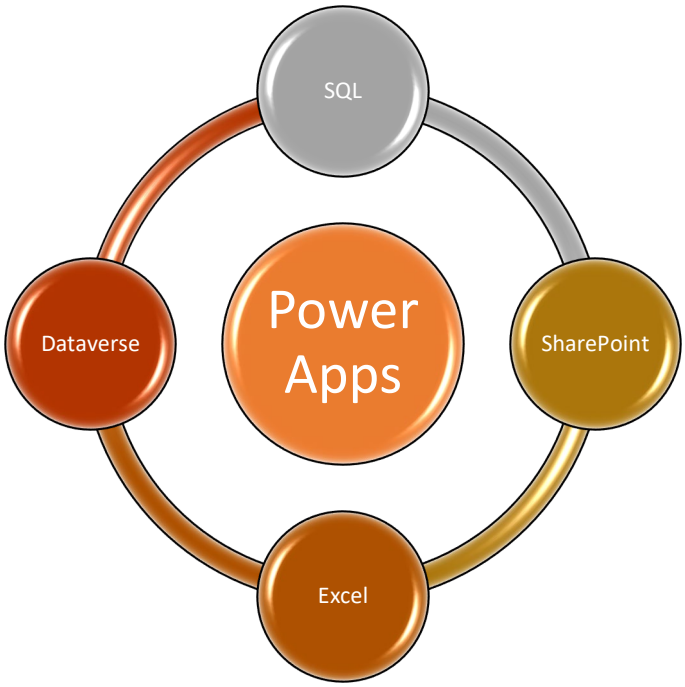
- Setting up Environment
- Connecting to Data Sources
- The Panes of Power Apps
- Types of Applications
- Licensing Costs
- Saving and Publishing Your Apps

Lesson 1: Power Apps Concepts

What is Microsoft's Power Apps?

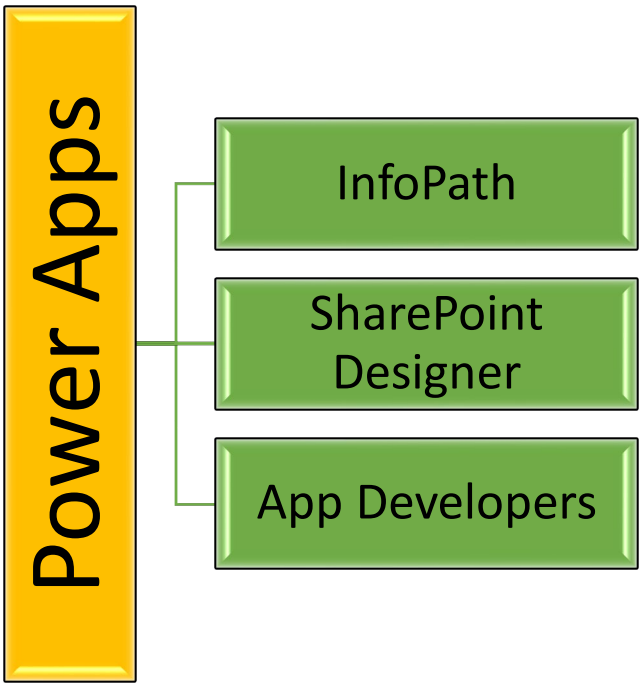
Microsoft's definition of their Power Apps is a suite of apps, services, connectors, and data platform that provides an application development environment to build custom apps for all your business needs.

It is a development environment that connects to data sources like SQL, Excel or Web Services to build your apps. It can also store your data in the Dataverse from multiple sources so you can build your app. These apps are intended to be accessed via a digital device such as phone or pad or can be embedded within your SharePoint site.

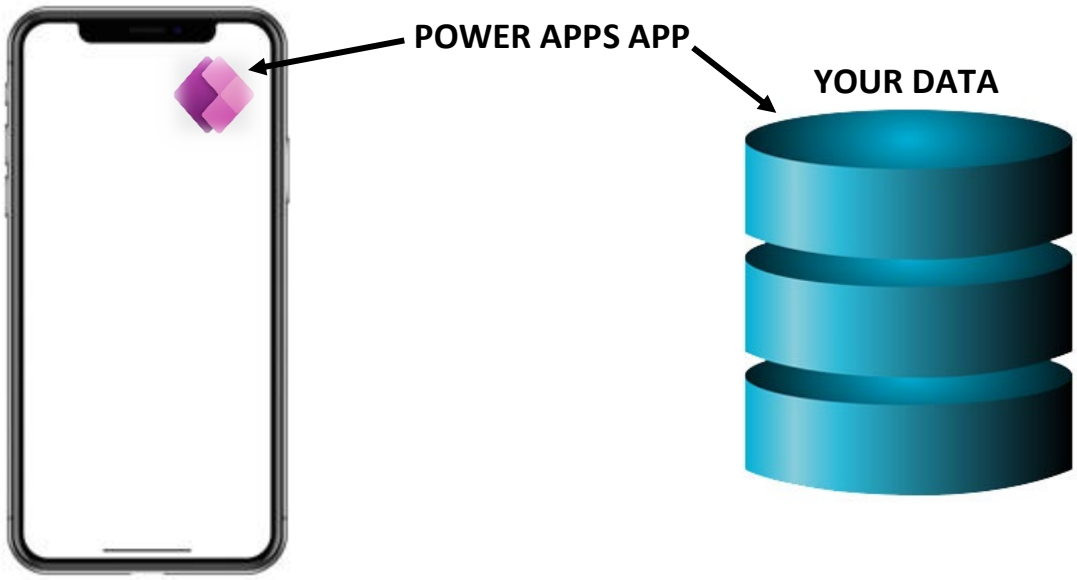


Power Apps essentially replaces Microsoft's InfoPath product and SharePoint Designer. If you have an O365 account, you can access Power Apps from the waffle menu. Keep in mind it is free to develop but there is a per month, per user, per app charge if you intend to share your apps. There is also now a charge to connect to SQL Cloud database.

This is also used by App Developers but what I mean by that is anyone who wants to access information via a Smart Phone app. Now the Smart Phone "App" is already created – Power Apps App by Microsoft. You are using their platform (App) to access to your "Apps" or "Forms" to have access to your data.



So, who should use Power Apps? Anyone who needs to have access to their data from a digital device or if you require a team of employees to have access as well then access from a SharePoint site would do the trick. You will create dynamic forms to either access your data or static forms to view dashboards that have been created. It is a one stop shopping for anyone who needs to access to their data on the go!



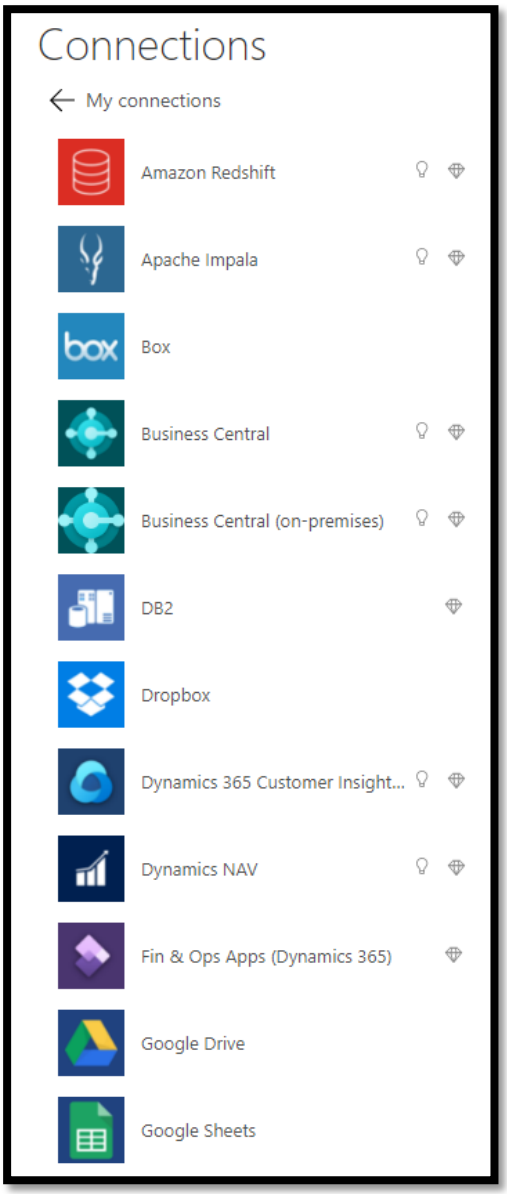
For more info you can go directly to Microsoft Power Apps site - Power Apps.microsoft.com.

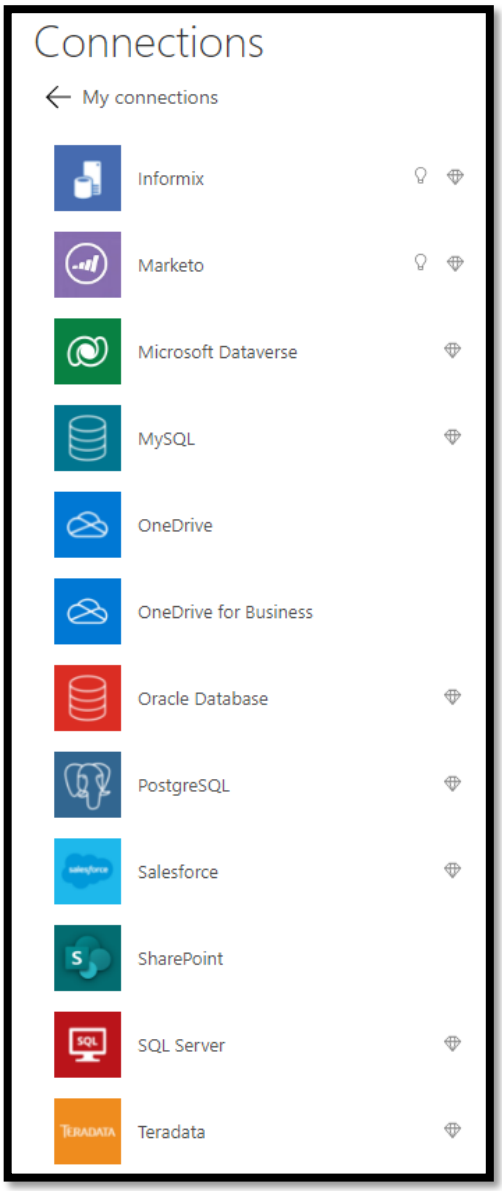
Features of Power Apps

The Power Apps environment is easy to use, and you do not need to know coding. You do however need to have a strong knowledge of Excel function syntax as you will be creating formulas. You cannot get away from doing so to make your app more interactive.

Another feature is that you can use your App that you created anywhere. By using the Power Apps app installed on your Phone or Pad you can connect to any of the apps you created and hence have access to your data anywhere you go.

You can also connect to any source of data. If you are familiar with Power BI which has over 300 connections, Power Apps has similar data connectors allowing you to connect to any source you may have including documents in Google Drive, OneDrive, or Drop Box.





Power Apps is also fully customizable, and you can design to meet your corporate needs and wants. You can also take advantage of your Phone features like your Camera, Microphone, and the Cell phone itself to make calls right from the app!

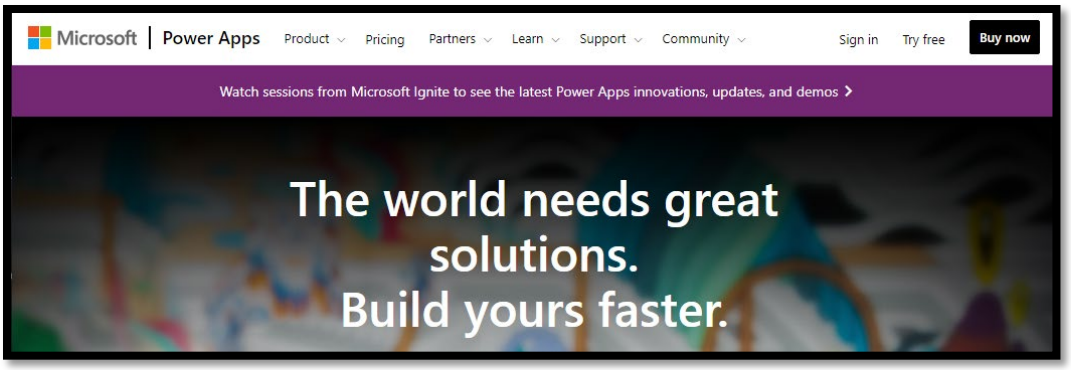
Note the Diamond icon in the previous screen shot is beside some of your connectors. That is because those are premium, and Microsoft will charge a fee to use those connections. It is usually $5 per user per app per month and that is at the time of the writing of this manual and subject to change.

Environment Setup

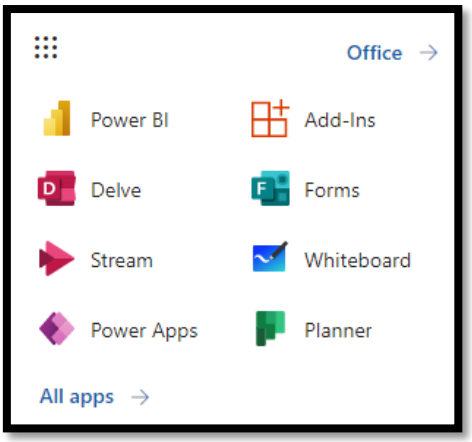
First you need to have an account in the Power Apps platform. You can either go to Power Apps.microsoft.com and signup or if you already have an Office 365 account just click on the waffle in the upper left corner to view that apps you have access to. Either way it is free to design to get you started.

To Set up Your Power Apps Environment

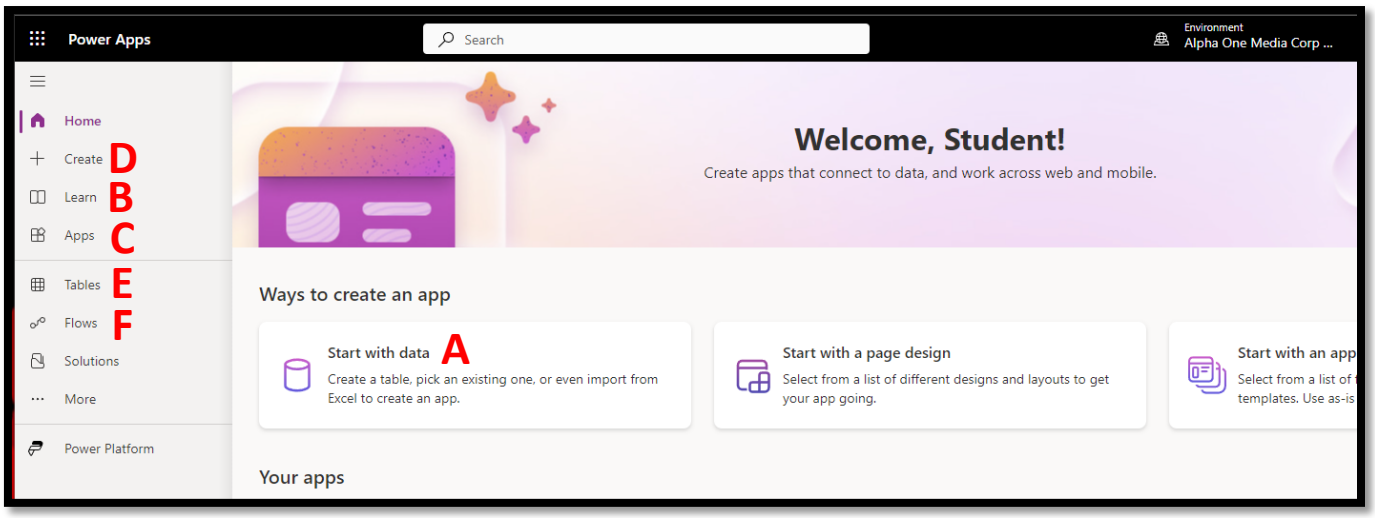
1. If you have **NO** O365 account, go to **PowerApps.microsoft.com** and click **Start Free** to create an account or **Sign-in** if you already have one. When signing up it must be a work or school account. They will not allow you to create an account with a personal email service (Gmail)



2. If you **DO** have an O365 account, then you can just login to your account click the **waffle** in the upper left corner to access all your apps and choose Power Apps from the list. You still may need to sign up for it like it the previous step for it to appear in this list.



3. Once logged into your Power Apps Environment your landing page will be displayed as seen below.



a. **Start with Data** – choose from Microsoft's most popular data sources.
b. **Learning for Every Level** – Microsoft provide help videos.
c. **Apps** – will list all your apps you have developed.
d. **Create** – create a new app.
e. **Dataverse** – this is Microsoft's new data housing feature (next course)
f. **Flows** – ability to create workflows for your apps.

Connecting to Data Sources

As mentioned before, Power Apps has many connectors to connect to various kinds of data sources. We will review three popular ones in this chapter – OneDrive, SharePoint List and SQL – however the course will revolve around using Excel documents in OneDrive.

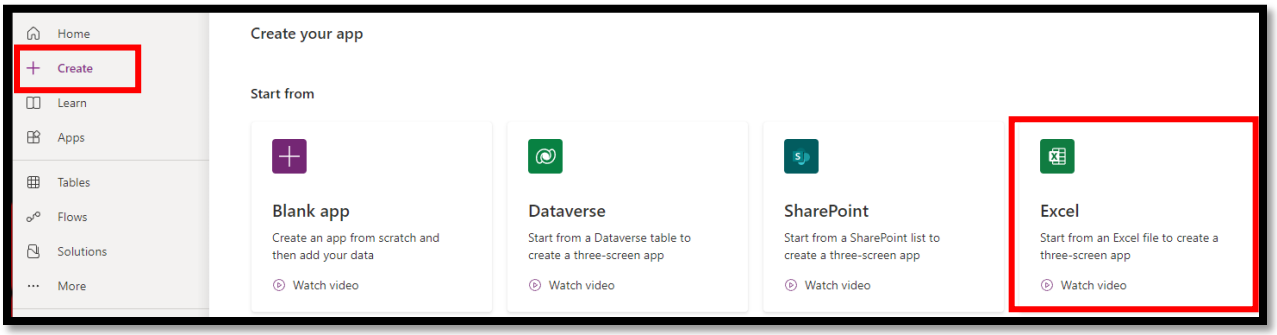
One thing to note about connecting to any data source. Power Apps will always create a **Power Apps ID Column** as the last column of your data. **DO NOT** remove this column. This column is used by Power Apps to communicate with your data. It will be created automatically and maintained by Power Apps, just let it be!

Connecting to OneDrive

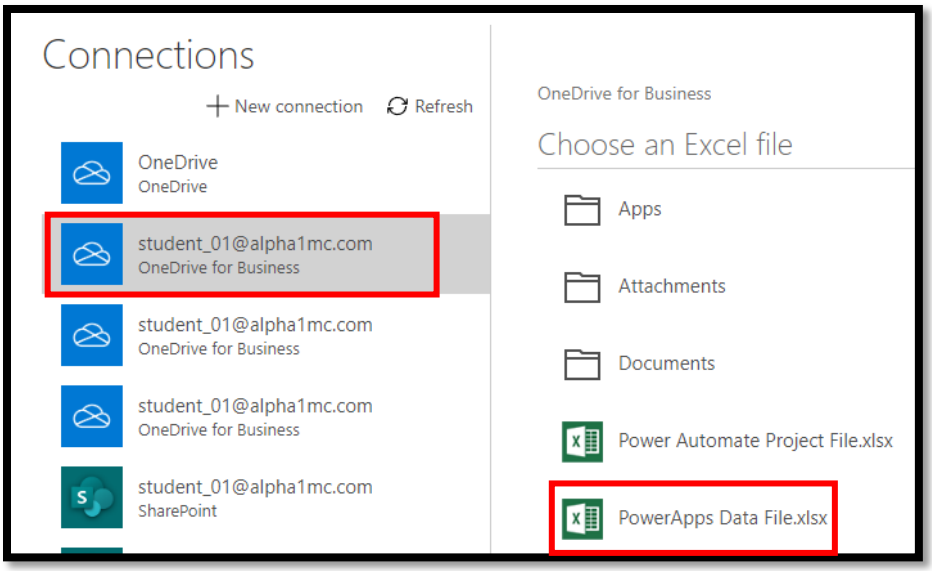
Most of us have access to our data in an Excel file. Whether we manually created that file, or it was dumped from a third-party software or database like SQL, we all know how to use Excel. You just have to store that file on some platform like OneDrive or Google Drive in order for Power Apps to access it.

One other requirement is that you must have turn each dataset into an Excel table and give it a proper name. The reason for this is when you connect Power Apps to an Excel file it will only read table names not sheet tab names.

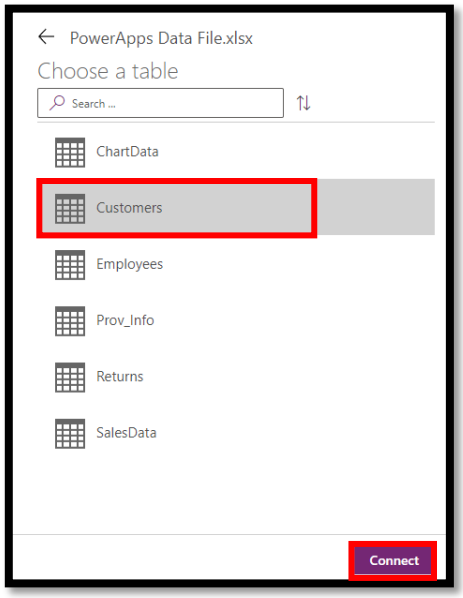
1. From your Power Apps landing screen menu options on the left, choose **Create** and then choose the **Excel** tile from the list of options.



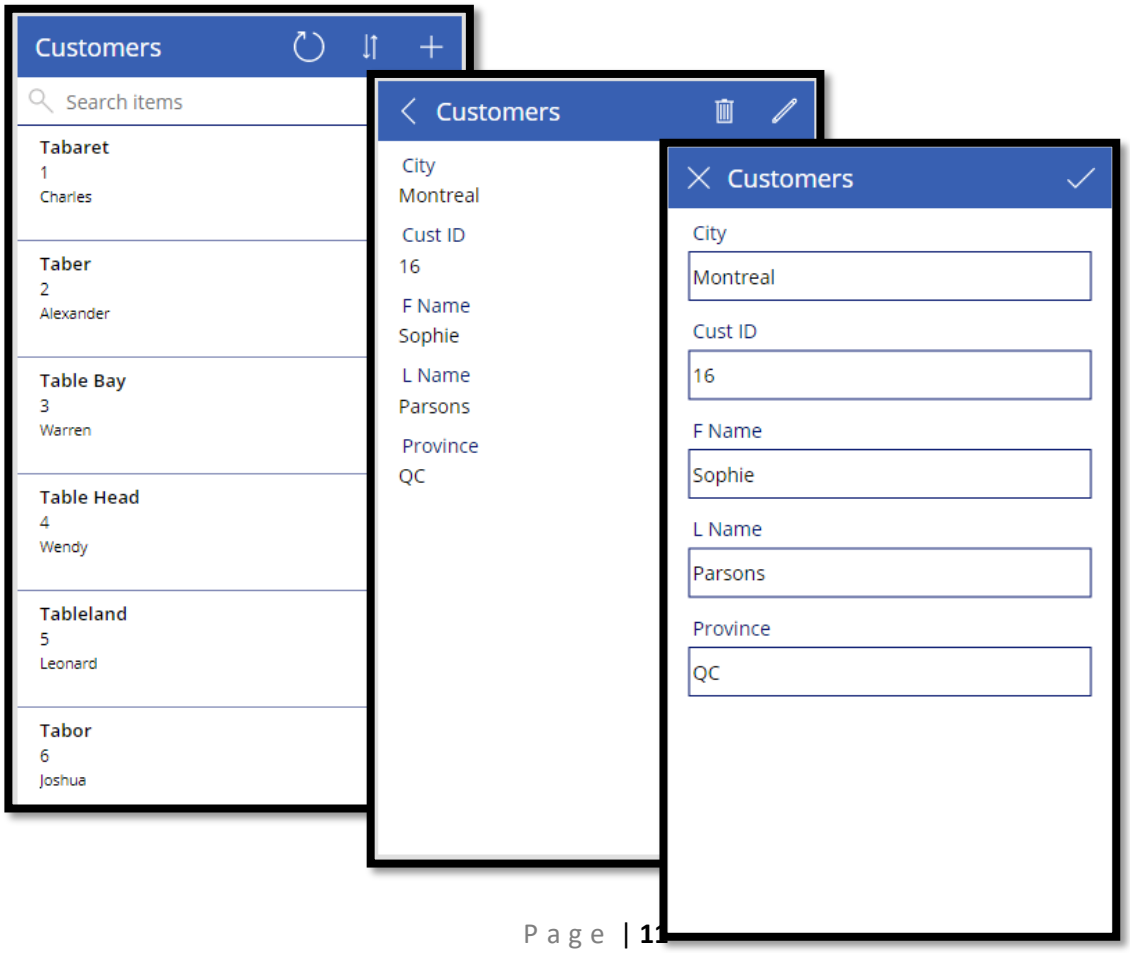
2. Locate the **OneDrive for Business** connection attached to your email and then select the **Excel** file you want to connect to.



3. Choose a table from the list and click **Connect**.

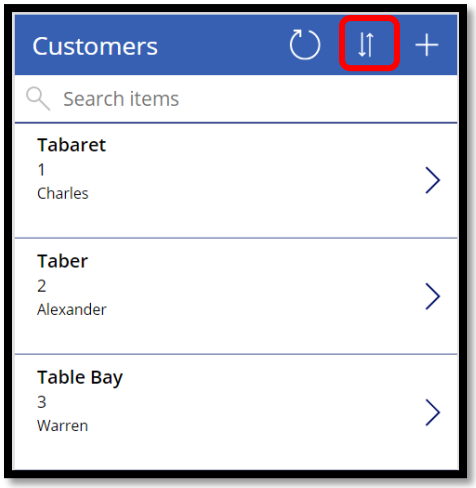


4. Once you click **Connect**, Power Apps creates the standard 3 screen app – **Browse** Screen, **Detail** Screen, and the **Edit** Screen. It may take a minute or two, but once complete it will display the Browse screen by default.

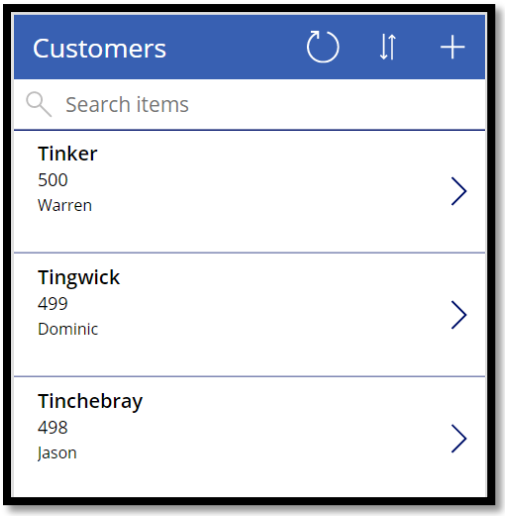


5. Ensuring that your **BrowseScreen1** is active, click the **Play** button in the upper right corner of your environment.

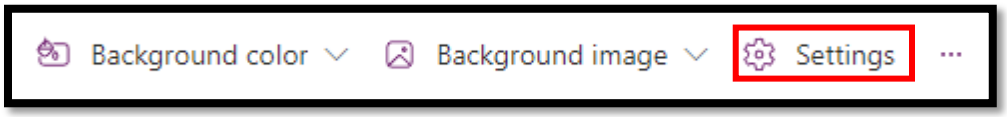
6. You can see you are viewing customers with IDs 1, 2, 3, etc. Now from within your **Smart Phone** view, click the **double arrows** in the upper right corner to sort your data.

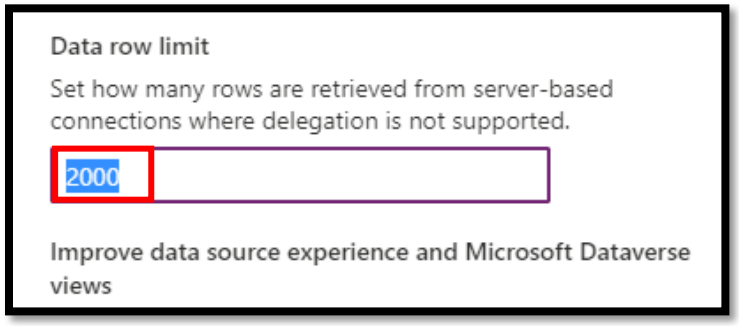


7. It will now sort by the first field in the table which happens to be the **Customer ID** number. You will see you have Customer IDs 500, 499, 498 etc. What happened to the other records? Read the note at the end of this section for a deeper explanation. In the meantime, click the "X" in the upper right corner to return to your canvas.

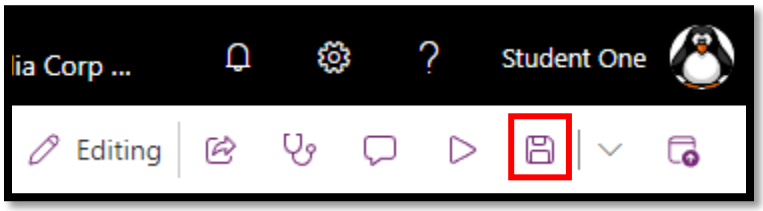


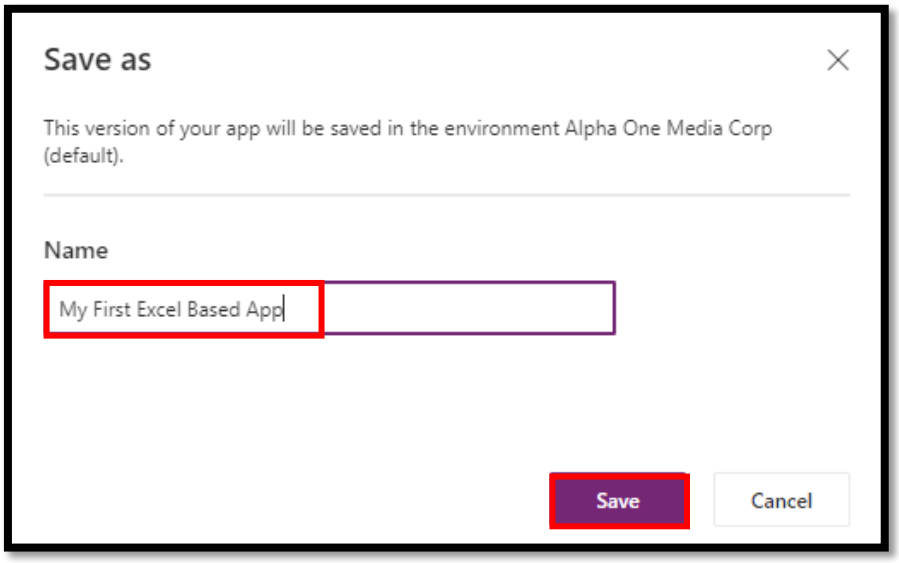
8. If you wish to see up to 2000 records at a time, go to **Settings** option on the menu bar and in the **General** section change the **Data Row Limit** field to **2000**. Exit out of the screen once complete.



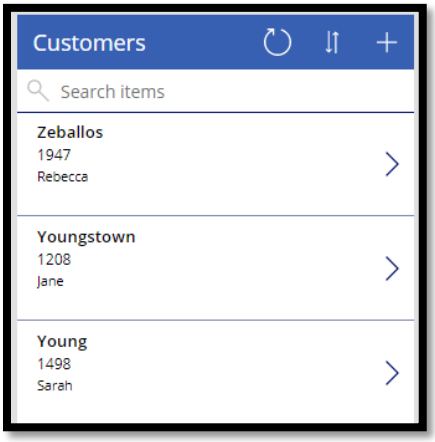


9. Remember you are within a tab in the browser, so if you hit the back button before you save you may lose your work. To save, click the **Save** button in the upper right, give it a file name and click the Save button. You can now hit the **Back** button on the menu bar ***(not the back button in the browser!).***





10. If you return to your canvas and view your Customer's Browse Screen and click the **Play** button, you will notice you now have a maximum of 2000 records.



NOTE: Using Excel as a data source for Power Apps has limitations. It cannot be larger than 2MB!! And Power Apps will only read the first 500 rows in a table (by default but can be adjusted to 2000 rows). Power Apps will not even connect to a file over 2MB so do not even try. It doesn't mean you can't have more than 2000 rows of data, but it will only show the first 2000. This 2000 rule applies for ALL connections.

Delegation in a Canvas App

The term delegation means to entrust a task or responsibility onto someone or in this case something else (software). Power Apps "delegates the processing of data to the data sources rather than moving all the data to Power Apps for processing locally. What does this mean for you? Excel does not support delegation; therefore, it will only retrieve and send the first 2000 records at max and then let Power Apps apply the queries. As a result, Excel is not a good data source for Power Apps if you have more than 2000 records.

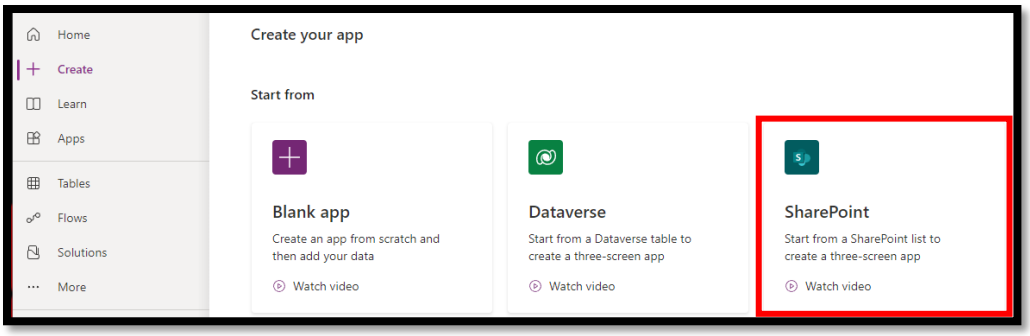
Microsoft's Dataverse, SharePoint and SQL Server can be delegated to along with the sales software Salesforce. There may be more in the over 300 connections to data sources that Microsoft allows, however those four are the main delegation software's you can use.

Connecting to SharePoint List

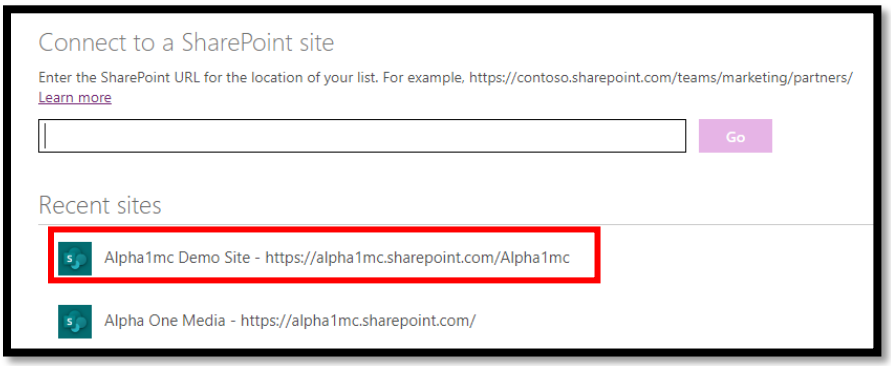
Before you're ready to connect your new Power Apps app to a SharePoint list, first you have to create those lists in your SharePoint. It goes without saying you need to have access to the List objects in SharePoint as well as any and all users that you plan to share this app with, otherwise the app will not work for them.

These SharePoint lists do not need to be located in any place specifically in SharePoint but as you know all objects created are always stored in Site Contents.

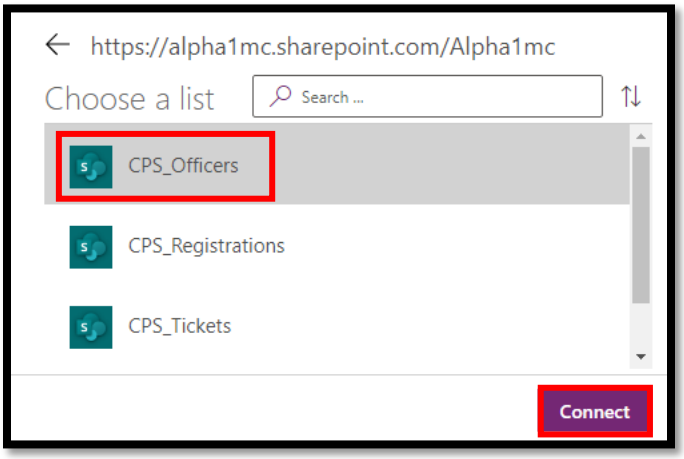
1. From your Power Apps landing screen menu options on the left, choose **Create** and then choose the **SharePoint** tile from the list of tiles.



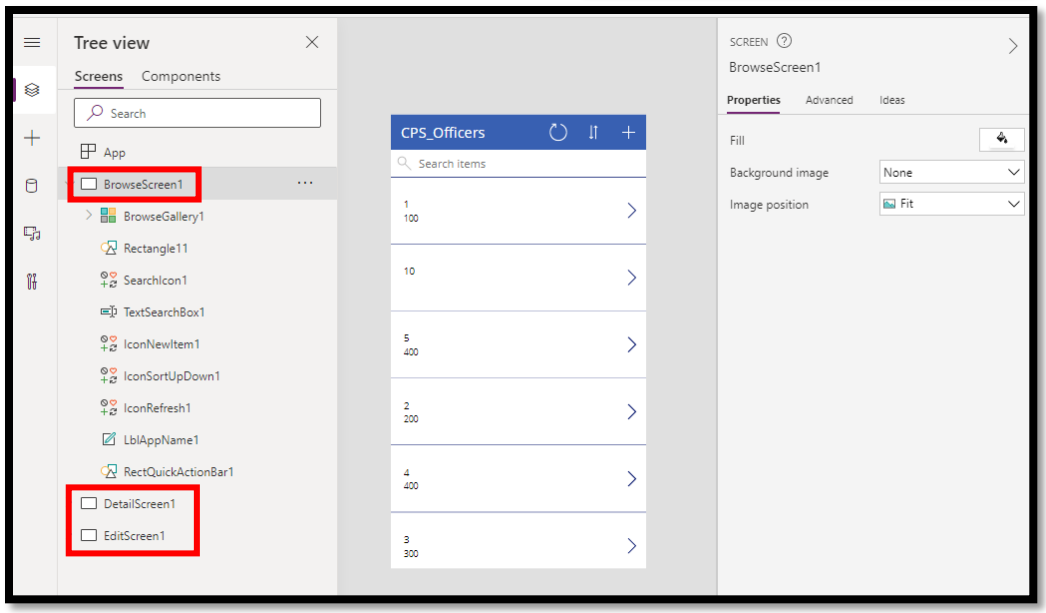
2. If you never connected to your SharePoint site, you would need to click the **New Connection** button to locate the SharePoint connector and follow the prompts to connect *(you will need to know the URL to your site or subsite at this point)*. Otherwise, just click an existing connection and it should display a list of sites that your current login (your email address) has access to. Just choose a site from the list of sites listed.



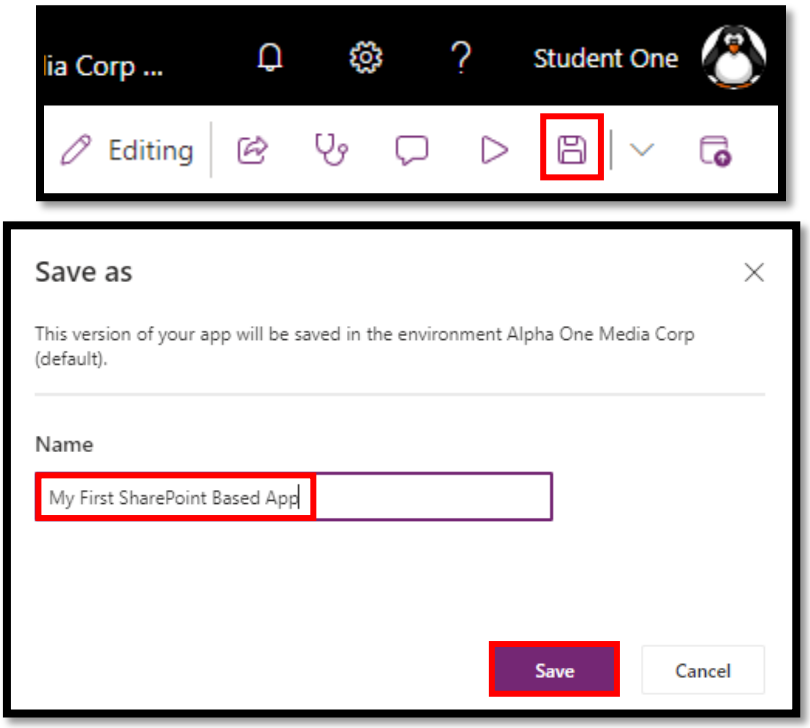
3. Power Apps will display a page of all the lists from the site you choose above that you have access to. Select one of your lists then click **Connect**.



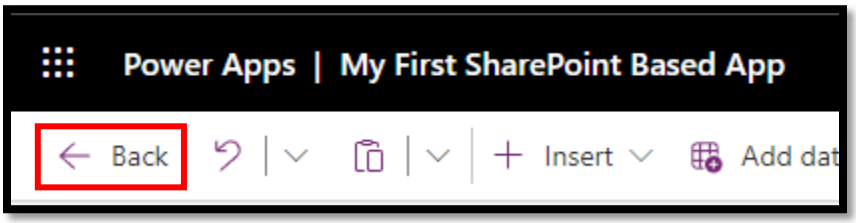
4. Once you click **Connect**, Power Apps creates the standard 3 screen app – **Browse** Screen, **Detail** Screen, and the **Edit** Screen. It may take a minute or two, but once complete it will display the Browse screen by default.



5. Remember you are within a tab in the browser, so if you hit the back button before you save you may lose your work. To save, click the **Save** button in the upper right, give it a file name and click the Save button. You can now hit the **Back** button on the menu bar ***(not the back button in the browser!).***



5. Once saved you will get a confirmation screen. Click the **Back** button on the menu bar to return to the Power Apps landing page.

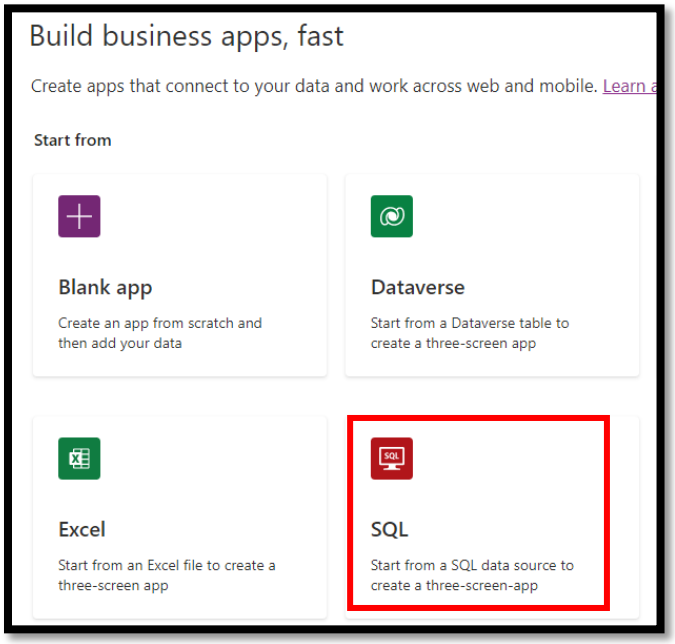


Connecting to SQL Cloud

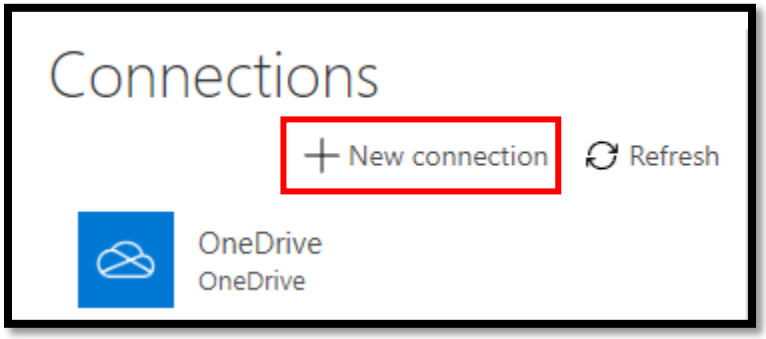
Connecting directly to a data source like SQL is often the best scenario. Mainly because you are working with live data results. As data is updated or added in SQL, anyone using your apps will have current info. And vice versa, if you are updating data from your app, then it is current within the SQL Database.

Before making the connection to your SQL database, you will need to know the URL to your server along with the name of the database and your credentials to login. So, your SQL Database is created, data is entered and now you are ready to connect your app to that database.

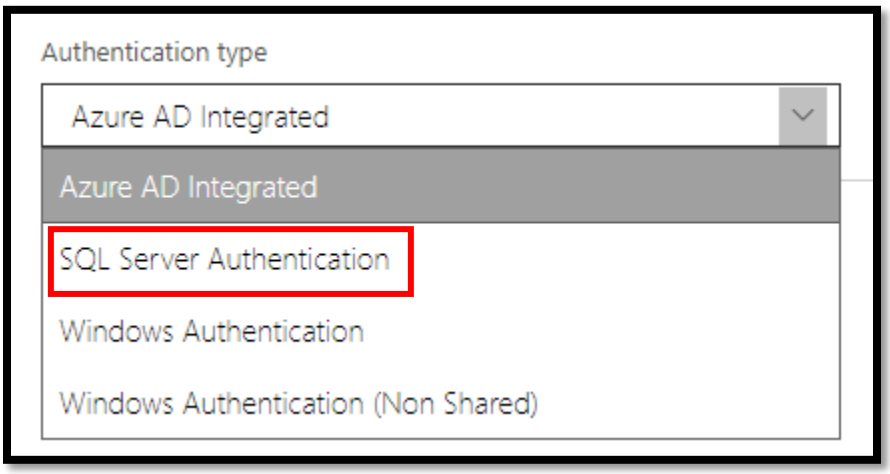
1. From your Power Apps **Create** screen choose the **SQL** tile from the list of tiles.



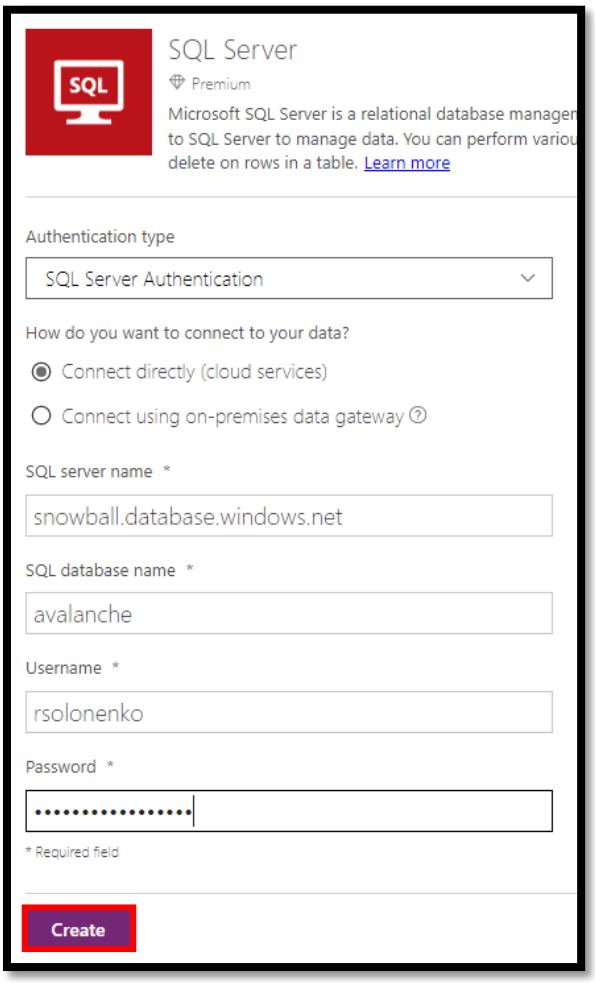
2. Choose **New connection** from the **Connections** pane.



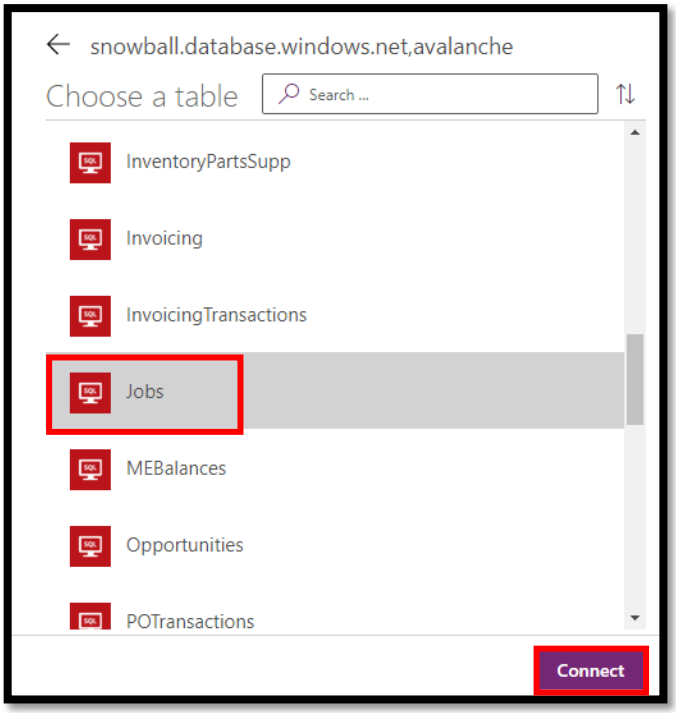
3. Scroll thru the list until you find SQL Server connector then choose the SQL Server Authentication type.



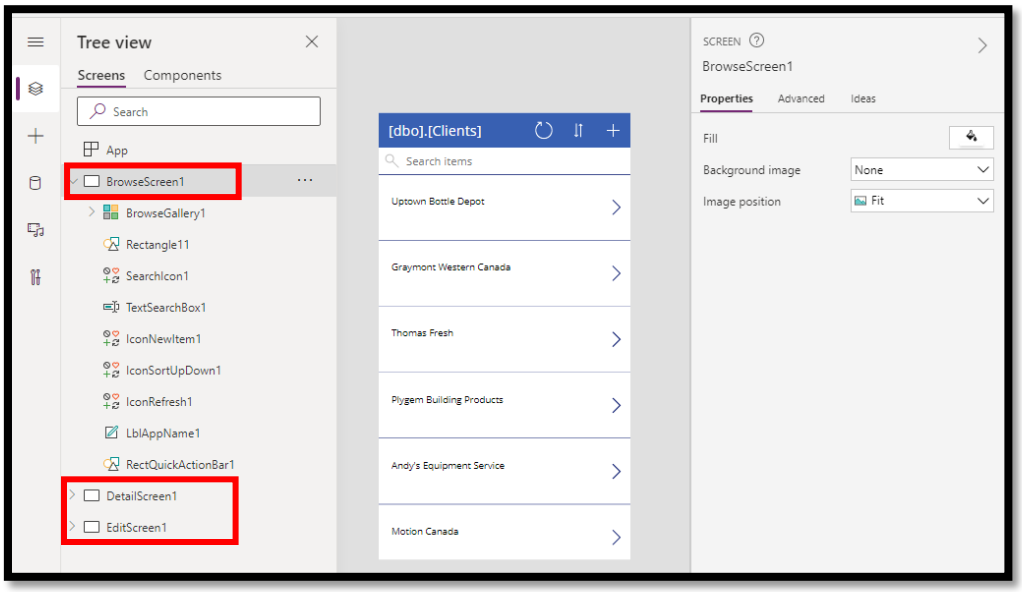
4. Based on previous selection enter in all fields and credentials. Click **Create** when done.

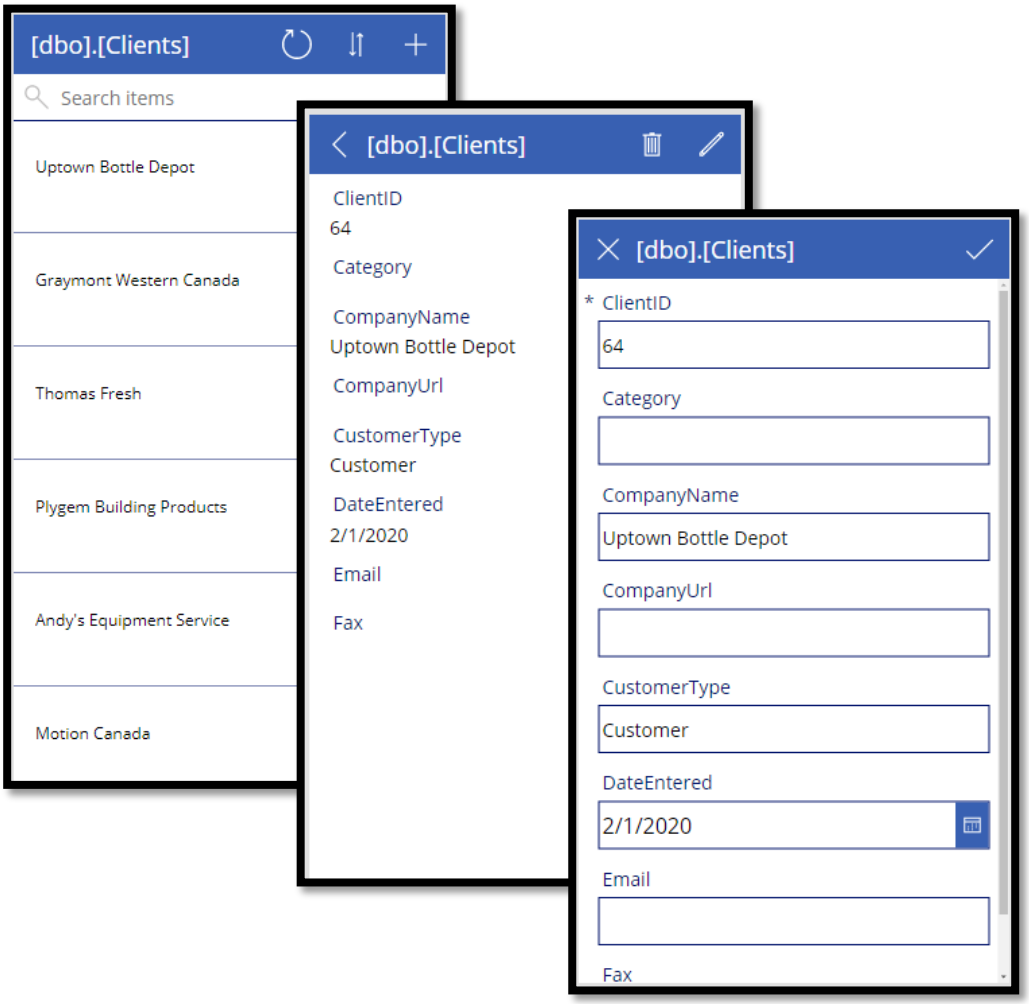


5. Once connected you will have access to the database and see a list of all tables. Select the table you want to access and then click the **Connect** button.

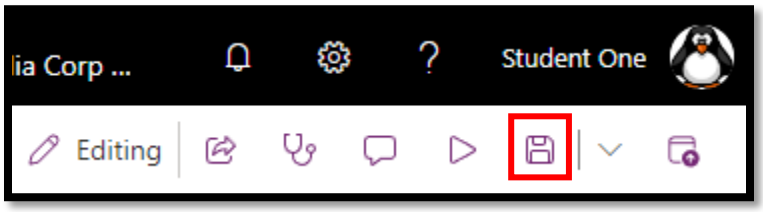


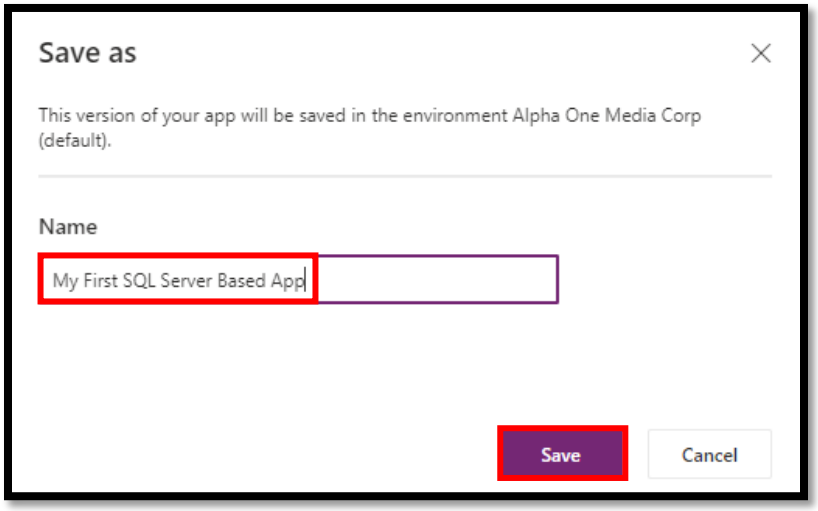
6. Once you click **Connect**, Power Apps creates the standard 3 screen app – **Browse** Screen, **Detail** Screen, and the **Edit** Screen. It may take a minute or two, but once complete it will display the Browse screen by default.





7. Remember you are within a tab in the browser, so if you hit the back button before you save you may lose your work. To save, click the **Save** button in the upper right, give it a file name, and click the **Save** button. You can now hit the **Back** button on the menu bar ***(not the back button in the browser!).***



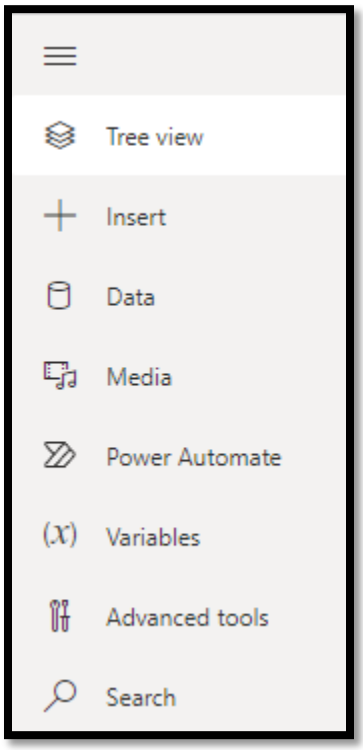


The Panes of Power Apps

Now that you are connected, regardless of what data source you are connected to, your Power Apps environment will look the same with various panes that now need to be discussed.

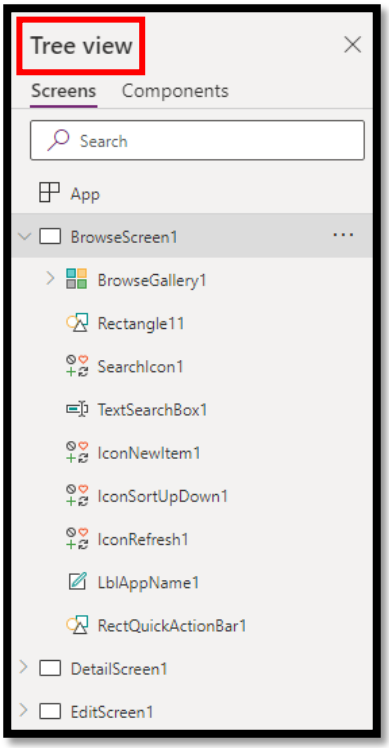
Starting on the left you have your main menu options where you can:

- Expand / Collapse this menu
- View your objects in Tree View
- Insert Objects
- Connect to other Data Sources
- Upload / View Media within your App
- Access Power Automate Workflows
- Access your App Variables
- Advanced Tools for your App
- Search your App for Content



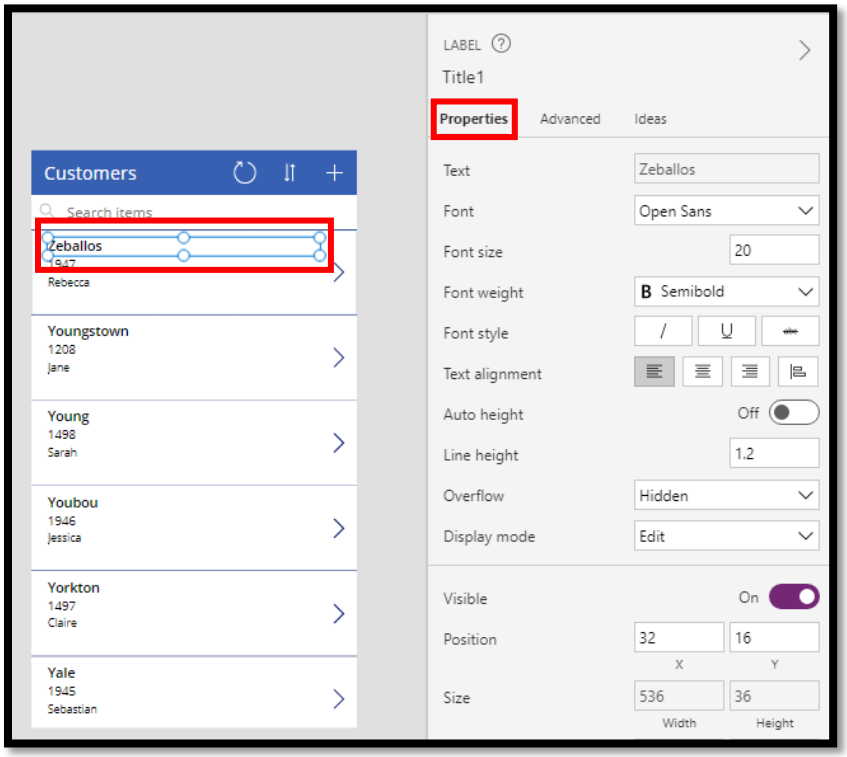
The Tree View Pane

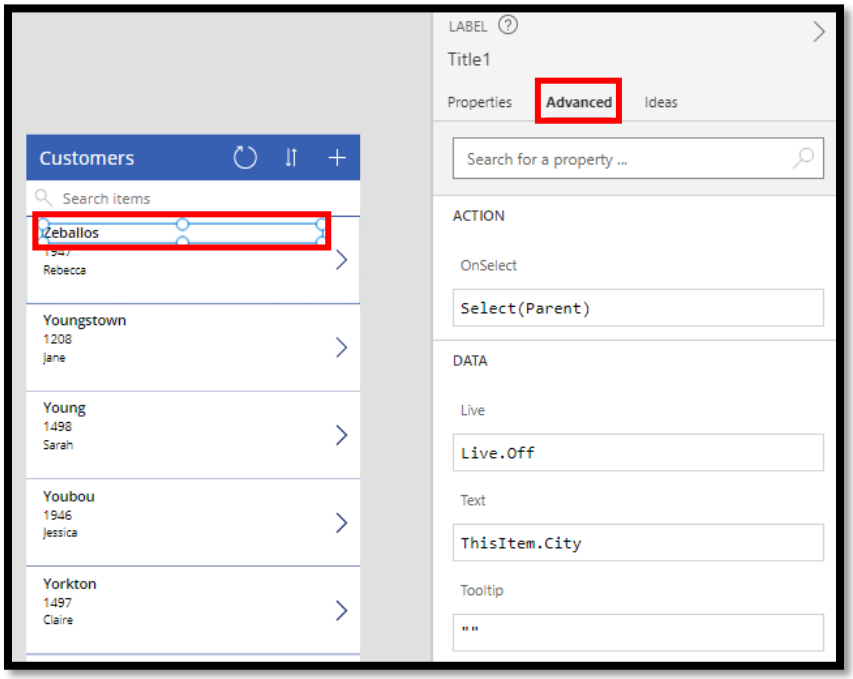
You will traditionally be working with this pane expanded so you can see all your screens and each object on each screen. This is also the pane you will use to select your objects because since the Smart Phone screen can be cluttered it may be hard to select your object on the screen.



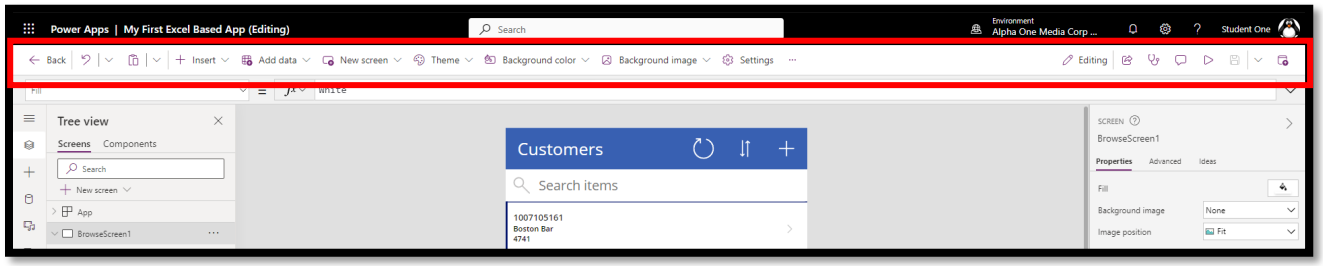
The Properties / Advanced Panes

These panes will adjust depending on what object you have selected in the Tree view or on the canvas itself. After selecting your object, you selected either Properties or Advanced to have access to all the features of that object.



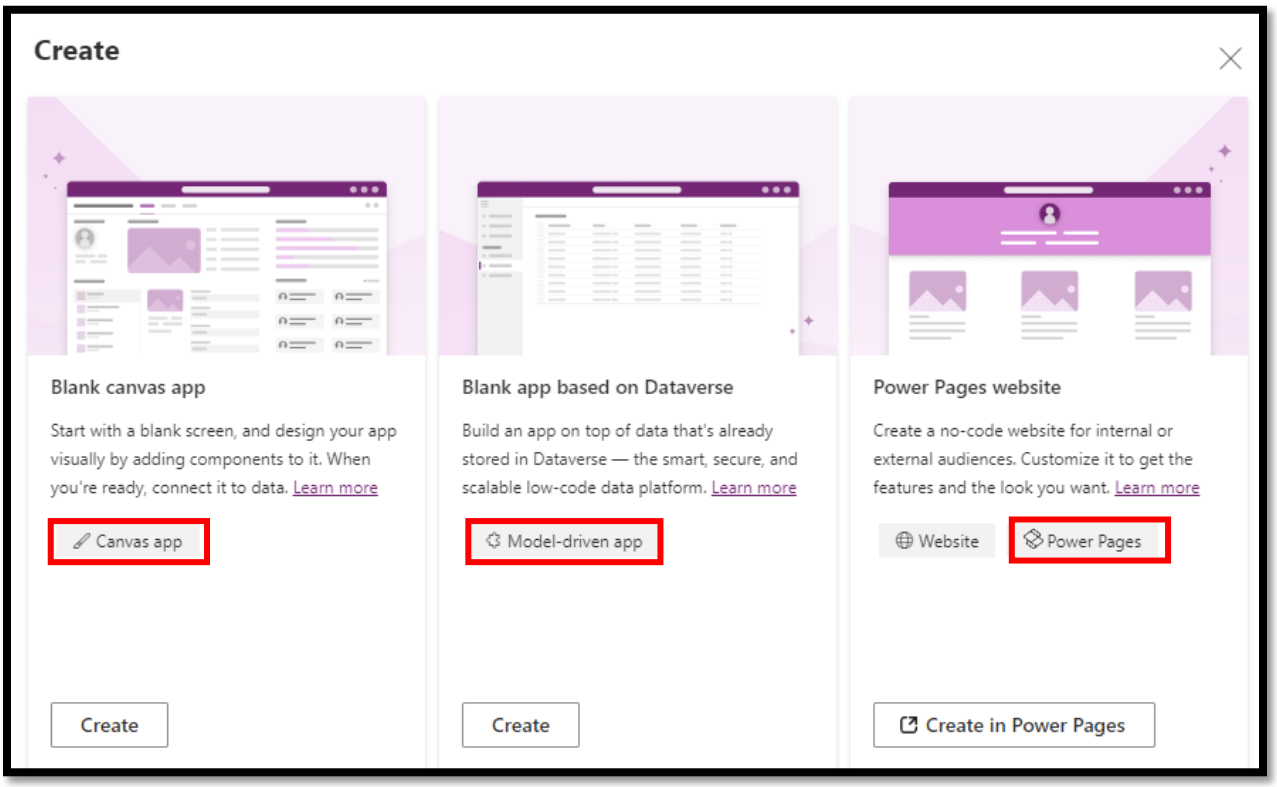


So, there you have the main panes of your Power Apps canvas. The rest of the menu options on the top act like traditional Microsoft menus buttons.



Type of Applications

When you create a **blank** app from the Create App landing page, you have three options to choose from: **Canvas App**, **Model Driven App** or **Power Pages App**. Each have their own purpose and place in an organization but knowing which is best for yours is the first decision you need to make.



Canvas App – Is the default app that gets created when you connect to your data source. It will also default to the size of a Smart Phone. You can however make it the size of a pad as well. No coding is required but knowing your Excel functions will help enhance your app.

You can connect your app to multiple data sources (Tables in Excel) but an understanding of how to relate them will be required. You can custom design your screens by simply drag and dropping the elements onto the canvas. It is easily shared with fellow employees on a fee basis. The screens are exactly like you have seen in the previous section where we showed you how to connect to Excel, SharePoint and SQL.

Model-driven App – This focuses on adding other components such as forms, views, charts, and dashboards using the app designer tool. In addition, your tables will be connected via relationships just like a typical database. Model-driven apps use Power Apps new Dataverse which contains all your data that you would have imported for faster retrieval speeds. They are also referred to as data model driven apps because without a data model housed within the Dataverse you cannot create a model-driven app.

These are especially well suited for process driven apps that have a lot of data and users need to move between related records. Unlike the canvas app where the designer has complete control over the app layout, the Model-driven app has much of the user interface determined for you.

Below is a table that outlines the differences between the two types of apps:

Category	Model-driven apps	Canvas apps
Data platform	Dataverse only	Dataverse + many others using connectors
Design experience	No-code component focused design	Manipulation of control properties using Power Fx expressions
UI control	Limited, predominantly customization	Full control
App consistency	High – differs predominantly based on the tables and views chosen	Often low, given the significant control the designers have of the user experience
Migration between environments	Simple	Potentially complex given that the data sources may need to be updated
Speed of creation	Rapid	Relative to the complexity of the design
Responsive	Automatically responsive	Only responsive if designed in this way
Navigation through relationships	Automatic, provided relationships exist	Only where designed and applied using Power Fx formulas
Accessibility features	Built in	Designed into the app: Create accessible canvas apps

Power Pages App – Launched in October 2022, Power Pages Apps replaced the Portals option they used to have which in return replaced Dynamics 365 Portals. These are external facing websites that allow users outside of the organization to sign in with a variety of identities, create and view data in the Dataverse or browse the content.

Like other features in Power Apps, there is a per user license to have access to your Portal Pages app. So, make sure you give access to the right people in order to keep your costs in line.

The Cost of Power Apps

We just wanted to touch a little on the topic of the cost of having Power Apps. As already mentioned, the cost of having the Power Apps environment can be free (pending on what type of O365 license you have) however, there will be the standard cost per user, per app, per connection.

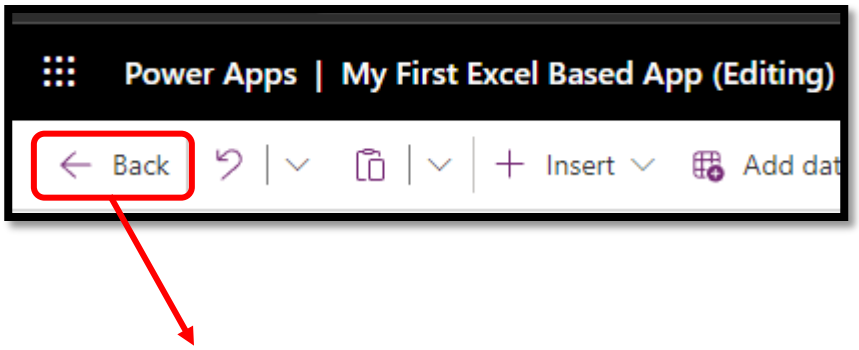
So, pending if it is a Canvas App, Model-driven app, is your data in SQL or the Dataverse, are you using the Portal App or just the Canvas app – and how many users will be requiring access to these apps – there will be a cost.

Most organizations already have some sort of O365 subscription package, and the cost will just be added on to that invoice per user, per app etc. Now you may get a little overwhelmed with the potential cost as this can soon lead to $1000 monthly bill.

You must remember, if you are a smaller organization that is entirely in the cloud (no local servers for anything), then you already are saving cost of not having hardware, or people to maintain them. So, a $1000/month bill to have immediate access to all your data anywhere you go is not a bad price to pay. Especially when you do not have to worry about security, backups or server uptime as Microsoft takes care of all of that and you can just focus on designing your apps and getting your users to use them!

Saving & Publishing Your Apps

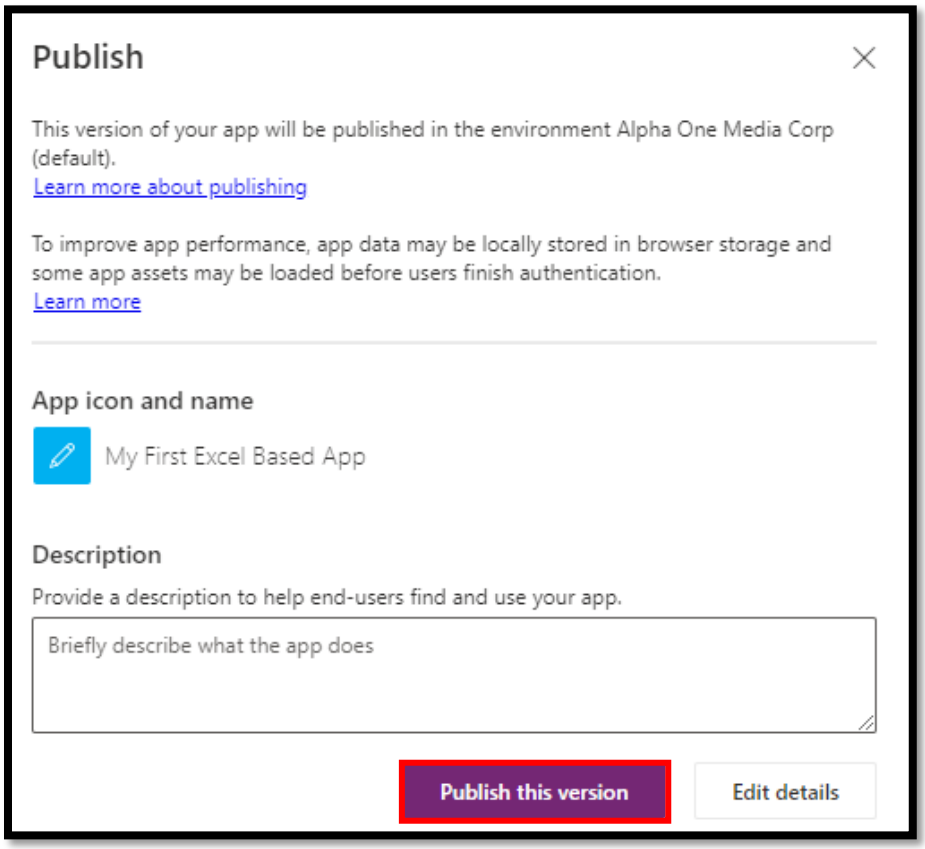
Seems simple enough yet some people have lost all their changes if they do not save properly in the Power Apps environment. You must remember your environment is within a browser setting and hitting the wrong back button can have dire consequences.



This will take you to the Power Apps main landing page.

Always click the **Save** button in the upper right corner of your top menu bar system. Only use the **Back Arrow** button when you want to back to the landing page to open a different app or create a brand new one.

Also, to remember, your app needs to be **Published** from the **File > Save** screen so you and others can run it from your Smart Phone app.



POWER APPS FOUNDATIONS

LESSON 2

Creating a Basic Canvas App

LEARNING OBJECTIVES

- Understanding the Gallery
- Forms & Data Cards
- Learning Triggers
- Navigating Your Screens
- Running Your Apps

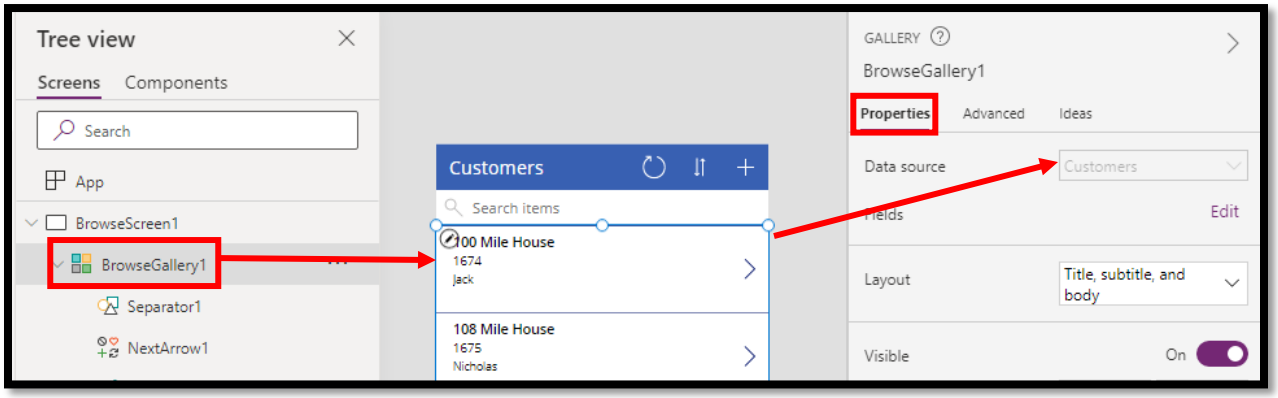
Lesson 2: Canvas App Concepts

In this section will create a basic canvas app connecting to an Excel file stored in OneDrive for Business. Remember you will have to ensure all your tables in the Excel file have been turned into Excel Tables and each given a proper name. Also, to note is that your Excel file should not be open or in edit mode when you are connecting your Power Apps otherwise it will return zero records.

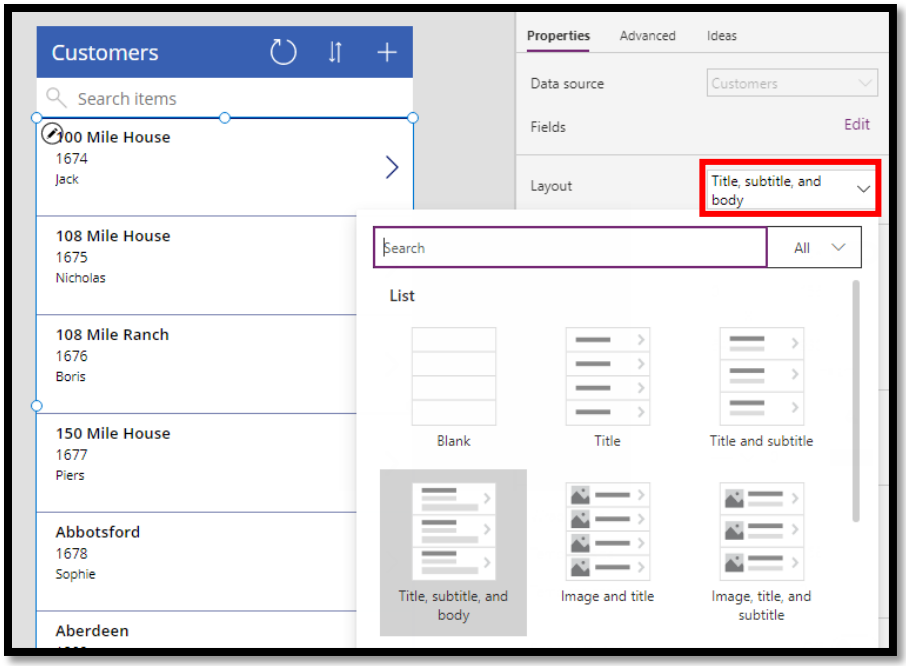
Understanding the Gallery

When you created your first Canvas App in the last chapter, on your **Browse Screen** you will see an object called **Browse Gallery 1** in your **Tree View**. This is a simple list that shows your records from the table in Excel that you connected to.

It is only a preview of the records and not all the fields in the record. It doesn't mean you can't show different fields than the ones that are originally displayed as you can choose others if you not like the ones that Power Apps chooses for you. Power Apps will usually pick the first two fields of your table plus one other like City.



Just like PowerPoint, your gallery can have various predefined layouts to choose from that will display your data in various views. You can select from the presets of course or you can customize as well.

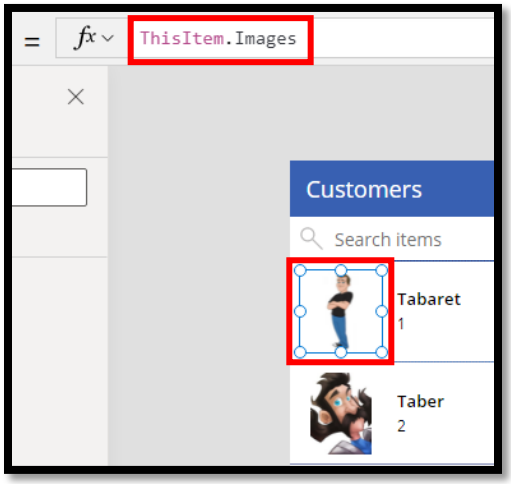


Depending which layout, you choose will depend on what and how many objects appear under the **Browse Gallery 1** element in your **Tree View** and what fields in your data source file are displayed by default.

Choosing the layout that has an Image to display requires you have a field in your data source named **Image** with URLs for its data. You must also change the syntax for this object to point to that Image field.

1. First select the **BrowseGallery1** from your **Tree View** and from the **Property Pane > Layout** dropdown select one of the layouts with an Image placeholder.

2. Now from within the **BrowseScreen1** screen, select the **Image1** placeholder and in the formula bar change it to read **ThisItem.Images.**

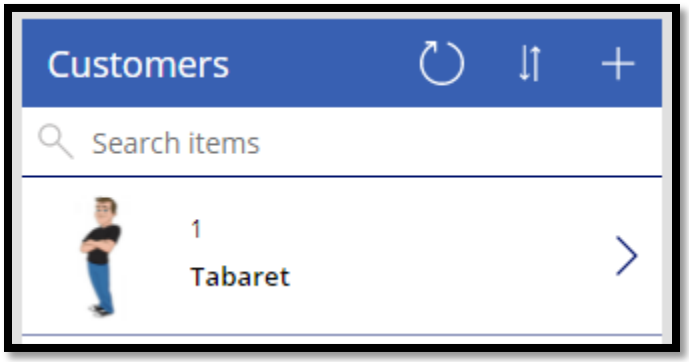


3. If it keeps giving you a red underline under the word Images above, that means it cannot see your Image field in your data source.

As you would have noticed already, changing the layout will change what fields are displayed as a result. In our example above it now shows the Image, Customer ID and the City fields. These fields can be dragged and dropped around to appear in the order you wish, and they can also be modified to include static text or combining other fields.

Let us introduce your first function in Power Apps – **Concatenate** – just like it is in Excel, it means to put together.

1. First rearrange your fields within the canvas to your liking.

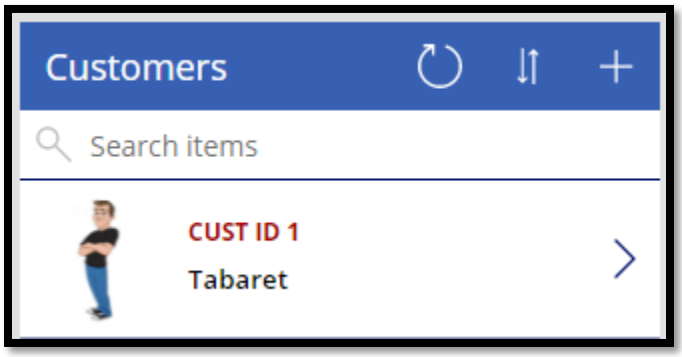


2. Now, in this example, we are going to select the **Customer ID** text box (field) and modify its Text property formula within the formula bar to read: **Concatenate("Cust ID: ",ThisItem.'Cust ID')**.

```
ThisItem.'Cust ID'
```

```
Concatenate("CUST ID ", ThisItem.'Cust ID')
```

3. Now using the **Properties** pane for this object, we changed the font colour and made it bold.

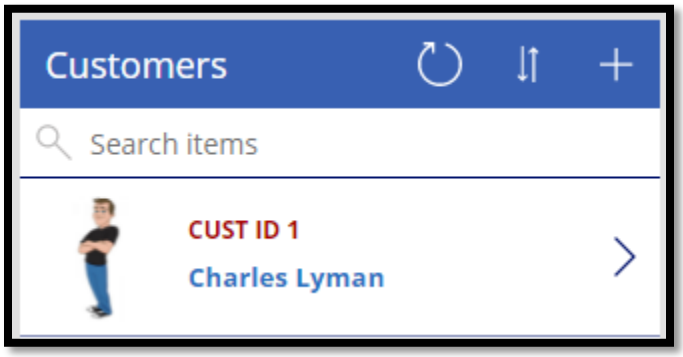


4. Select the **City** field in the canvas and change its Text property formula to the following: **Concatenate(ThisItem.'F Name'," ",ThisItem.'L Name')**.

```
ThisItem.City
```

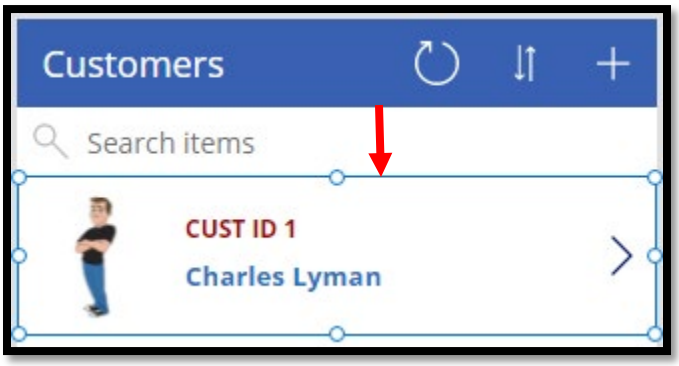
```
Concatenate(ThisItem.'F Name'," ",ThisItem.'L Name')
```

5. As we did to the previous field, feel free to adjust the font properties.

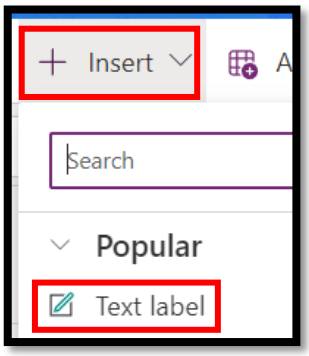


Lastly, if you want to add in another field into this Customer browse screen, you must make sure you select the **BrowseGallery1** placeholder box within the canvas, then from the **Insert** menu choose **Label**. Follow the instructions below.

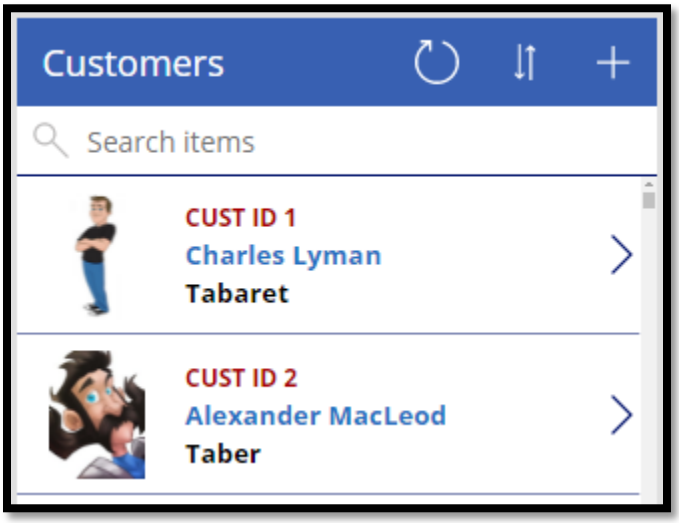
1. Select the border of the first record within the **Browse Screen** canvas.



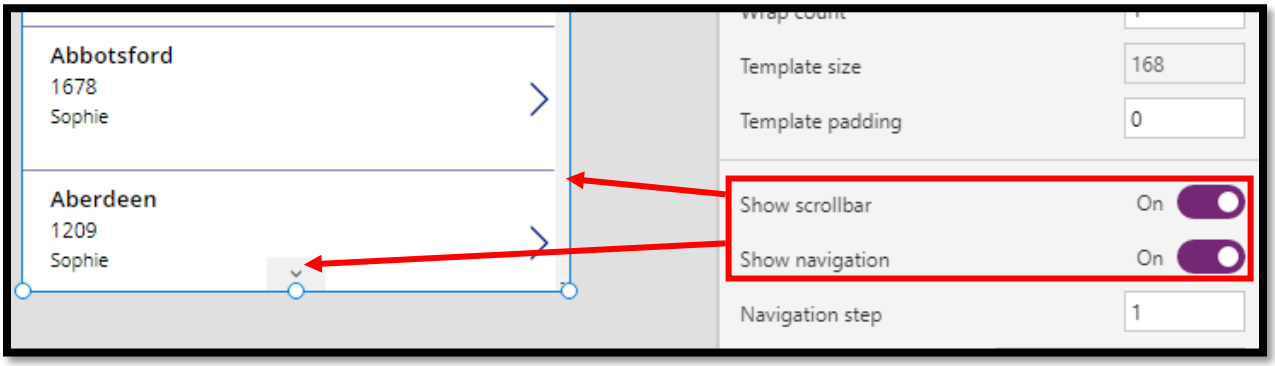
2. From the **Insert** menu choose **Text Label** and it will insert a label, then change the Text property to read **ThisItem.City** in the formula bar.



3. Position the object within the placeholder.



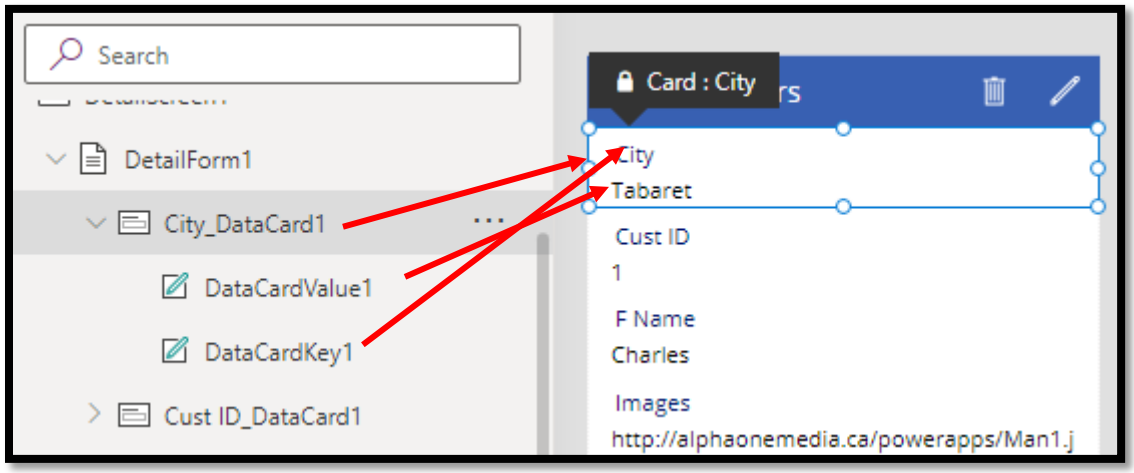
Now off by default, you can turn on **Show Scrollbar** or **Show Navigation**. We all know how to scroll with our thumbs on a Smart Phone, but you may want these features to actually be visible to your users. There are also other properties you can see that you may want adjusted like colours, fonts, lines etc.



Forms & Data Cards

Now that we have reviewed the Gallery object which sits on the Browse screen, we will now look at Forms which typically appear on the **Detail** and **Edit** Screen. You have two types of forms – Edit and Display – and each form will have a multiple Data Cards and each card will have data card value and data card key. Here is a breakdown of what each is:

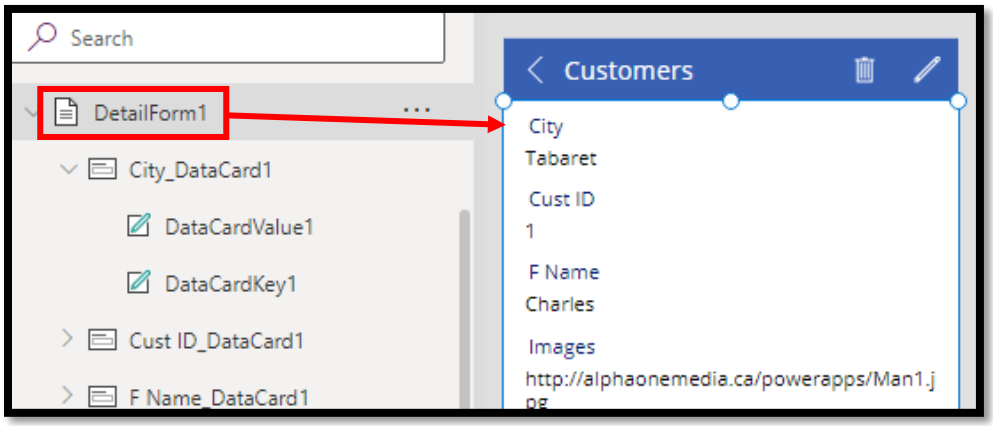
Forms will contain…	
Data Card	Placeholder w/ same name as field
Data Card Key	Name of field from table
Data Card Value	Value of that field from table



You will have one data card for every field that is to be displayed on this form. Remember you do not have to show every field on every screen as it will depend on what you want your users to see and have access to, but by default it will show you one data card for every field in your data table.

To adjust or remove cards:

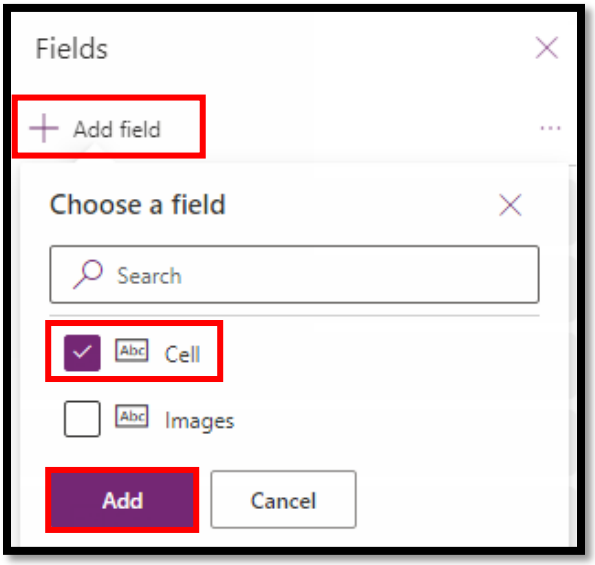
1. Select the **DetailForm1** either from the **Tree View** or from the canvas.



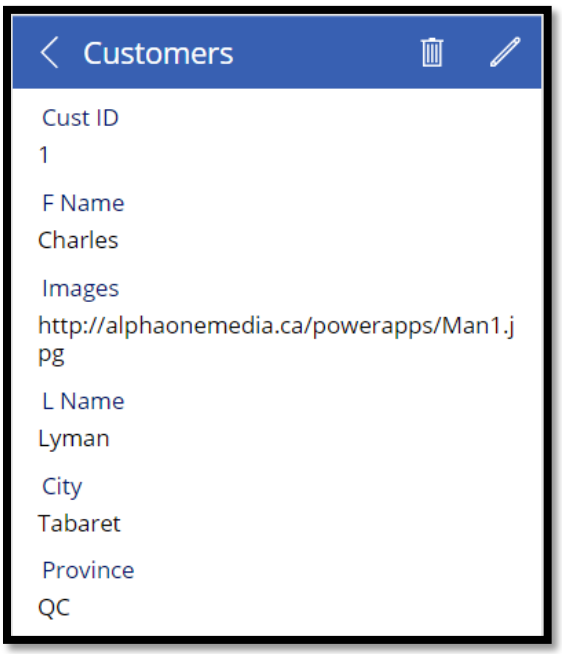
2. From the **Properties** pane click the **Edit Fields** link.

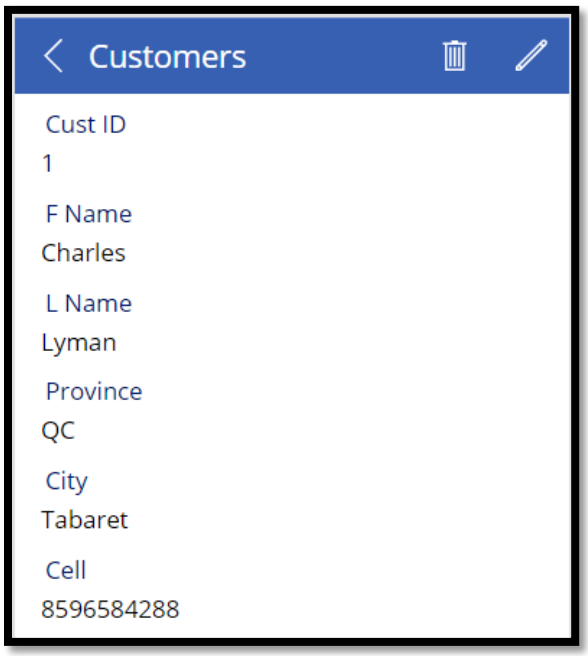
3. Within the **Fields** pane you can drag and drop the order in which you want the Cards to appear, or you can delete an individual card by hovering over the Cards name and clicking on the ellipsis **(…)** and choosing **Remove**.

4. If you want to add a field back into the card, choose the **Add Field** button in the **Fields** pane and either search for your field or find it in the list and click add.

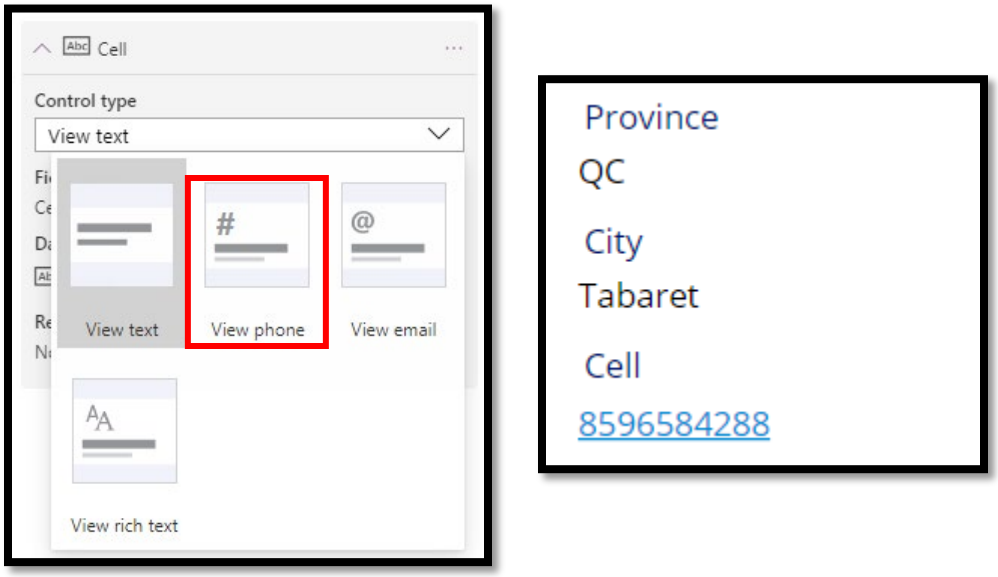


5. Once complete, simply close out the **Fields** pane.



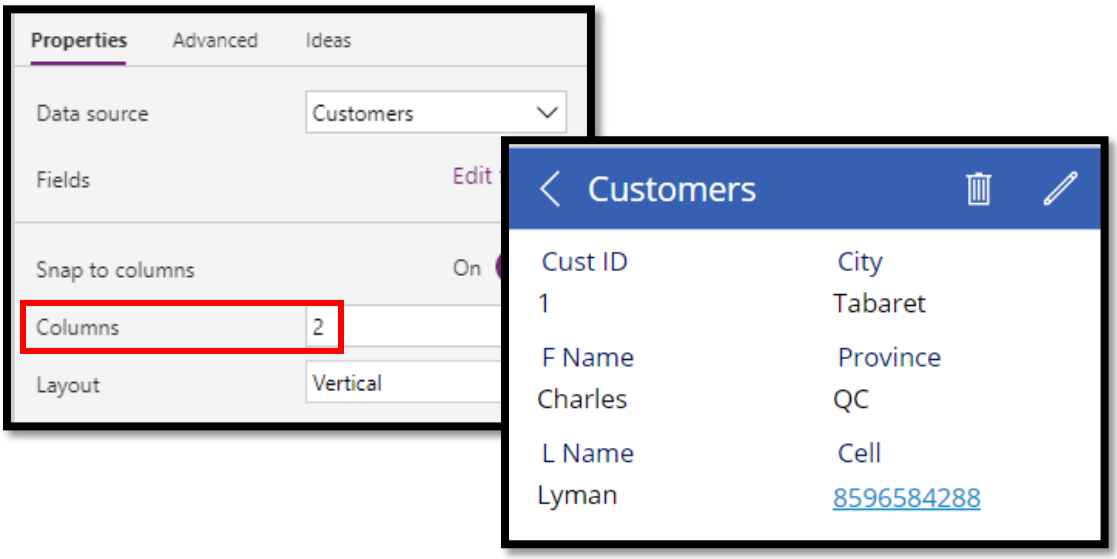


6. Now you can also adjust the **Control Type** and how its displayed between four options – Text, Rich Text, Email or Phone. Assign it a Phone or Email type will allow the app to use the phone or email features on your Smart Phone when clicked by the user.

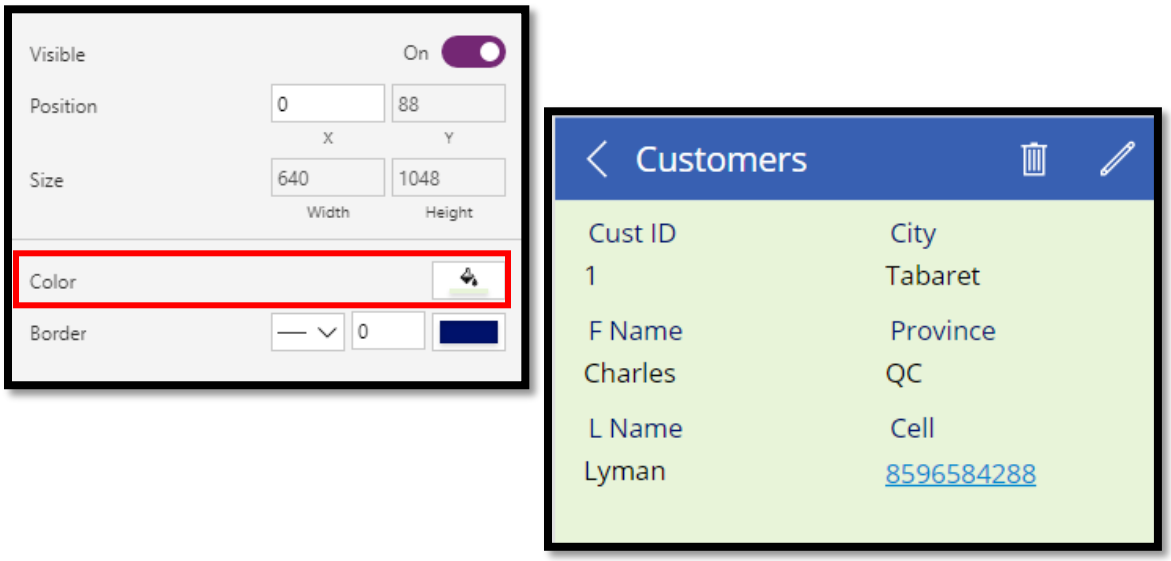


Some other properties of the Form and Data Cards that should be looked at are Columns, Colours and changing Label Names. One thing to note about the **Data Cards**, is they are usually locked automatically so you must unlock them before you can adjust their properties.

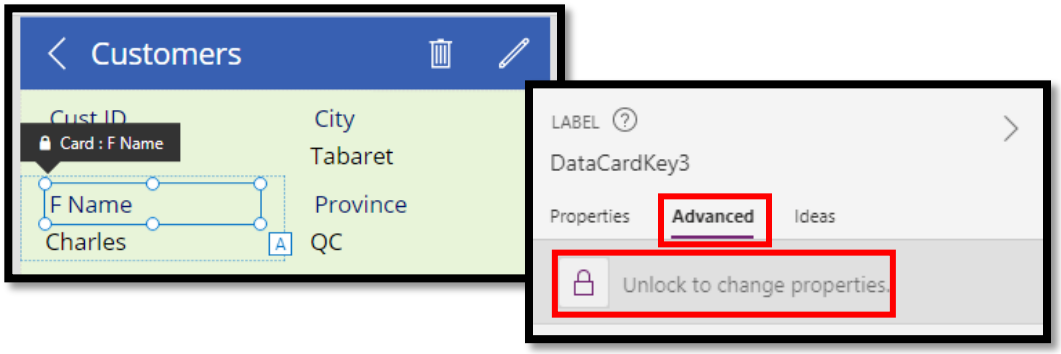
1. To adjust the layout using columns, select the **Detail** Form object and in the **Properties** pane adjust the **columns** to **2**.



2. Now let's adjust the colour of the form by using the Paint Can on the properties sheet.



3. Lastly let us change the name of the F Name and L name field objects. First select the **F Name Data Card Key**, then in the **Advanced** pane click the **Unlock** button so you can change properties. Then in the change the Text property formula to read: **"First Name"**.

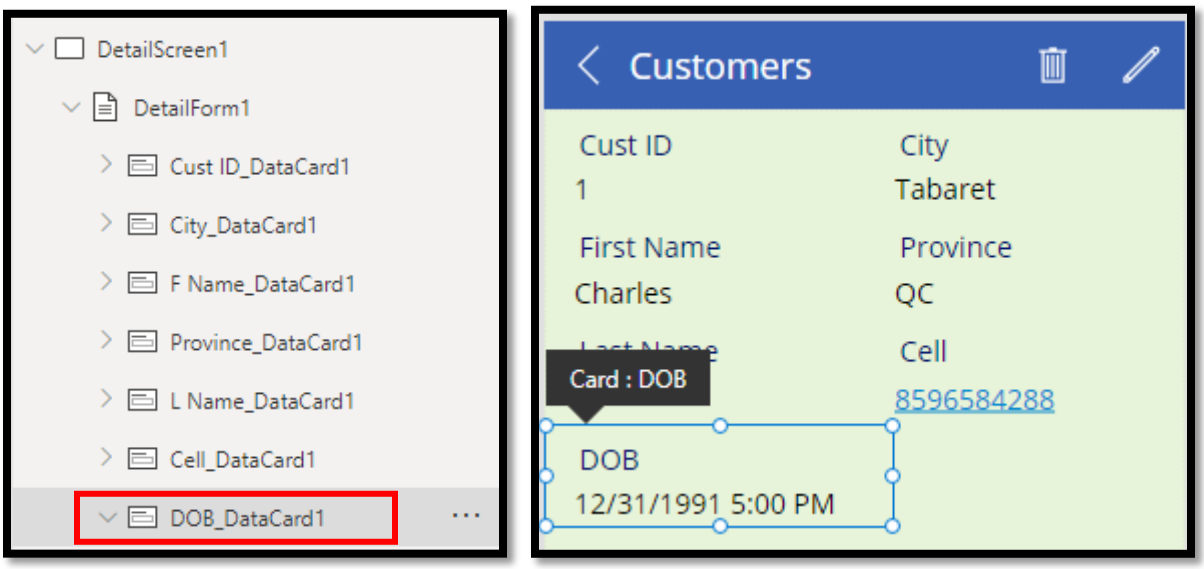


Modifications for Detail and Edit Screens

We started in the last section adjusting properties of the **Form** and **Data Card**. Now let's look at some of the other objects that should have their data properties adjusted. Like the previous one you will have to select the object via the **Tree View** or **Canvas** and then use the properties pane to make the adjustments. We will start using the **Formula Bar** to make more interesting changes.

Let us look at formatting a date field as dates can always be a little tricky. In our example the date in your Excel file is written as "01/01/1990", however as you can see by the screen shot below, it will display it as date and time. Here is how we change that:

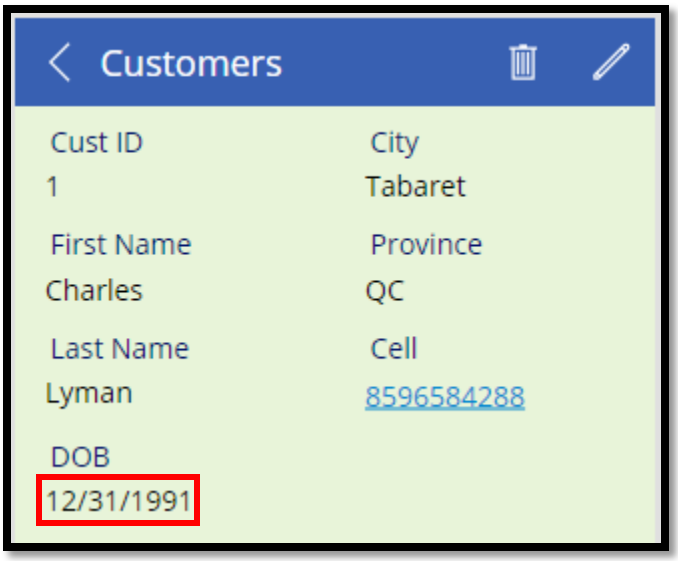
1. From your **Tree View** pane select the **Card** within your form that has **Date** data.



2. From the **Property List** drop down, select **Default** if necessary.

3. Now adjust the formula to read the following using the **Text** function.

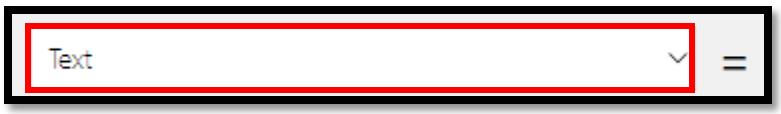
4. Your date in your field on the **Detail Screen** will now have adjusted.

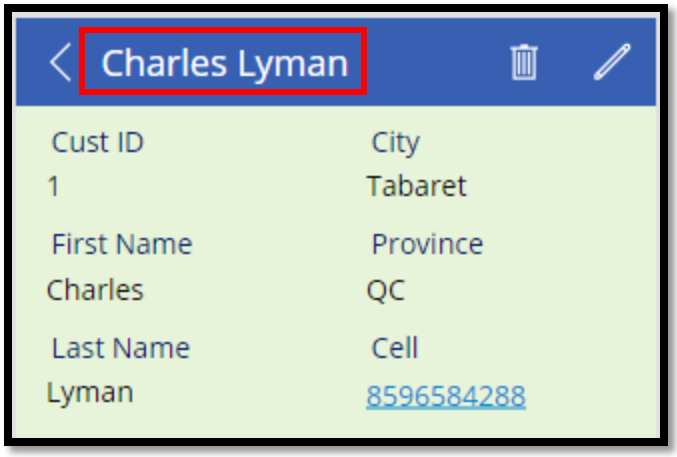


Now let us change the name of the page. You will notice that the name of the page comes from the name of the table in Excel. In our example it is Customers, however yours may say Table1 if you never named your Excel table. We will change it to give us the first and last name of each customer.

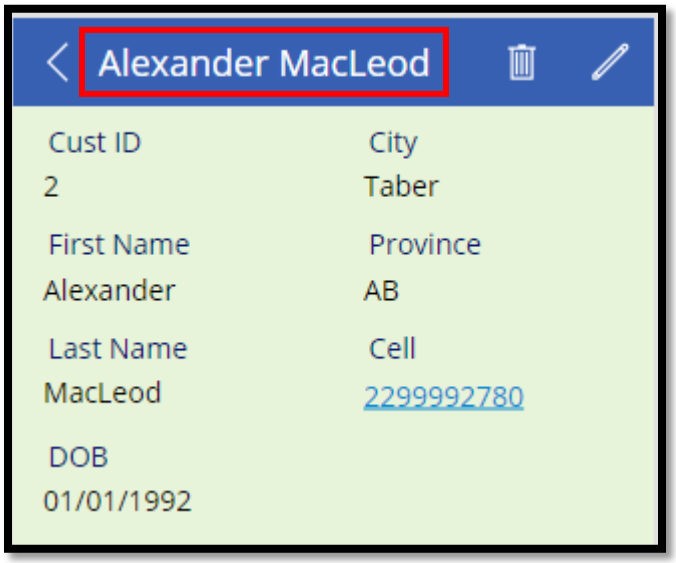
1. Select the **Title** object directly on your canvas.

2. Ensuring the **Text** property is selected, adjust the formula to what you see below. We are using the **Concatenate** function to combine the values of both the first and last name fields. We also need to add a space between the two.



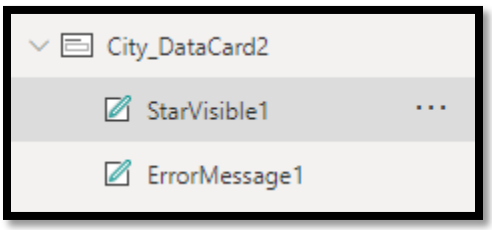


3. If you go and **Play** your app, and then switch customers the title should adjust for each name now.

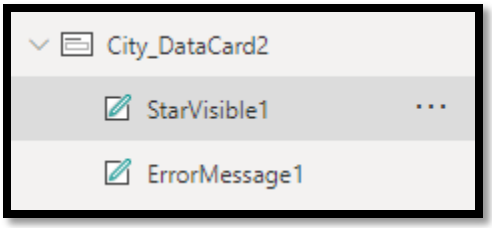


In the previous section we worked within the **Detail** Screen and its objects and now we will look at the **Edit** Screen and its objects. You would think they should be identical, but they are not. The Edit Screen's Data Cards have more objects, and each card value is editable by the user. How you handle each object and modify its properties will be the same as in the previous section.

Star Visible Card – this will display the asterisk (*) which we all know to be mandatory field.

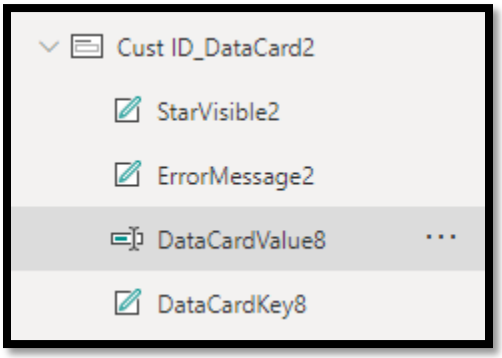


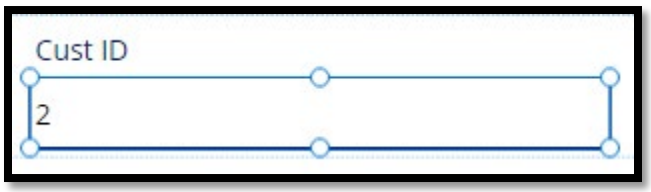
Error Message Card – this will display an error based on a function that will validate what the user enters. By default, users can enter anything into the fields, and it will accept it.



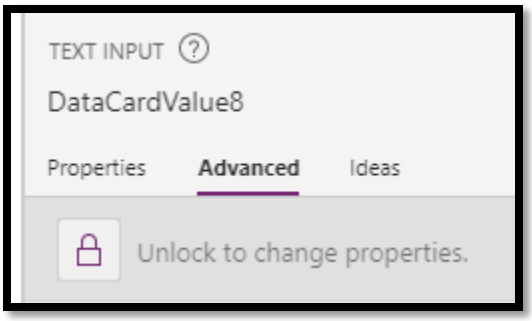
By default, users can change every field that is displayed on this screen. However, sometimes there are some fields that should not be allowed to change like ID numbers. Power Apps can turn the editing feature off of fields that require it off.

1. Select the field that you want to disable the editing ability for (***on the edit screen***).

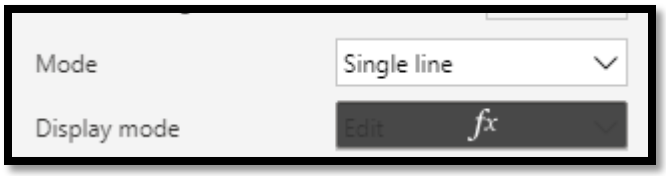




2. Click the **Unlock** button in the **Advanced** pane so you can make the changes.



3. Now from the **Properties** pane locate the **Display Mode** field and select it once.



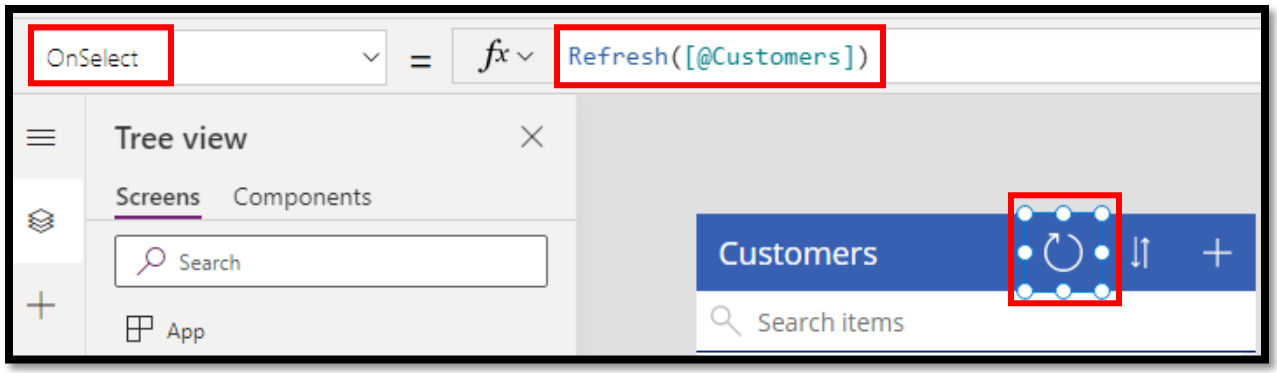
4. In the formula bar locate the **DisplayMode** property and delete the current text and replace it with **DisplayMode.View.**

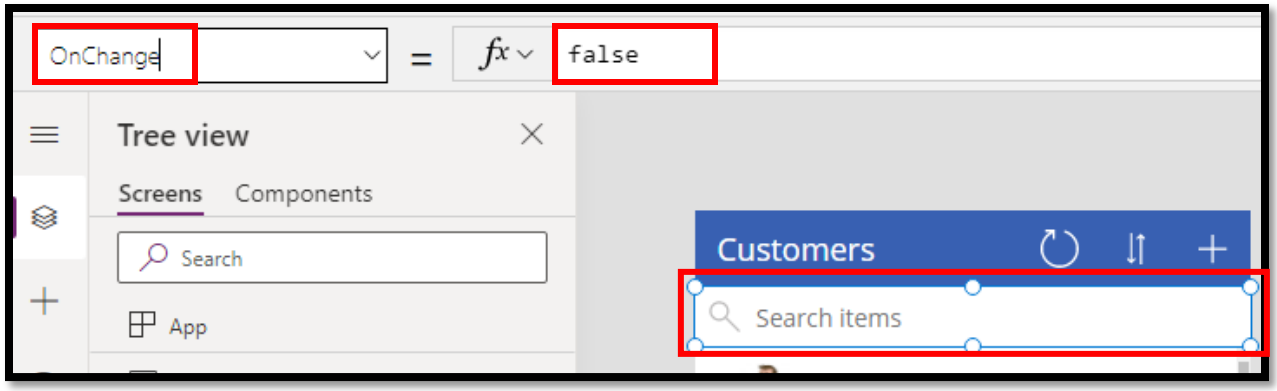
5. You will now see in your Canvas the look of the Cust ID number field has changed and if you play this app, you will see you cannot change the Customer ID at all.

Learning Triggers

While we have just learned that the elements on the screens, or objects, have various properties that can be adjusted, they also have triggers that can perform an action. A trigger is when a user clicks on the object or hits the enter key for example. When that trigger happens what do you want the object on the form to do is basically what this section is about.

All triggers in Power Apps start with the word "On" – OnSelect, OnHidden etc. So, once you select the object, you then will select the trigger from the drop down to determine which trigger will start the action, then you will of course provide the formula that will actually perform the action. Not all objects have the same triggers.

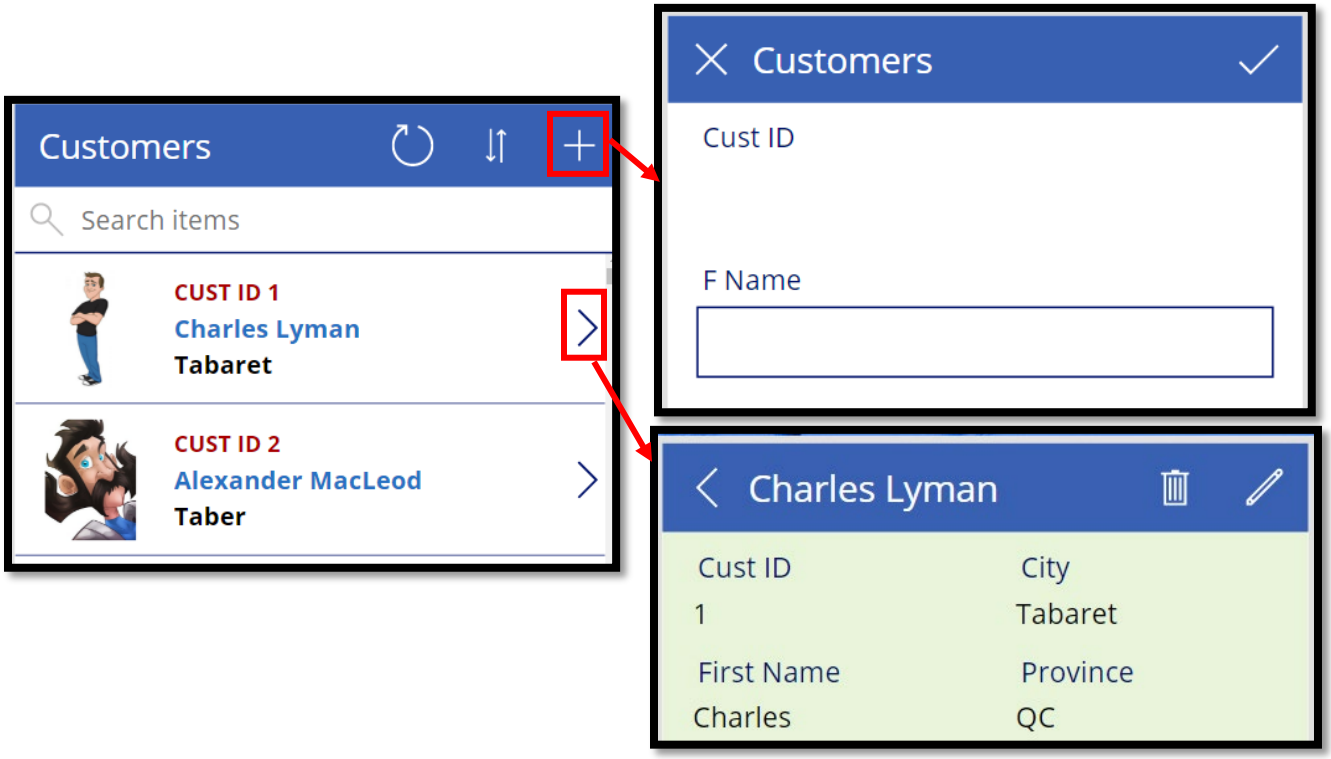




Navigating Your Screens

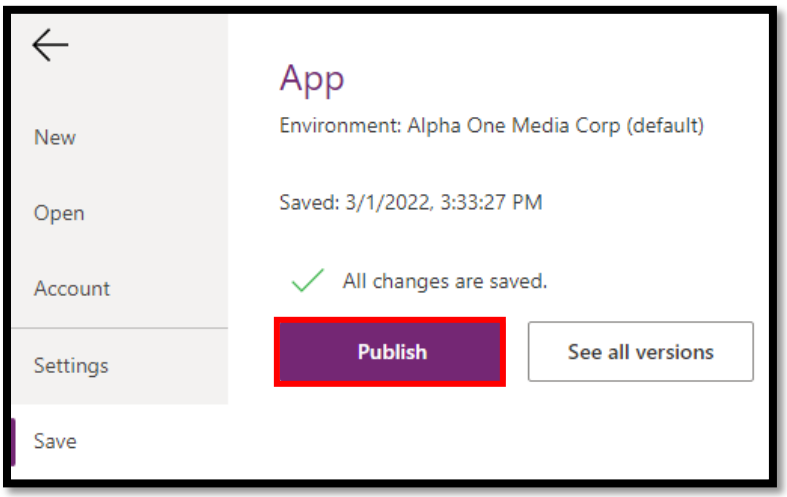
While you are in edit mode, you can simply use the Tree view pane to navigate to the various screens. In the case of the standard 3 screen app that is automatically generated when you create your first app there are only the Browse Screen, Detail Screen and the Edit Screen.

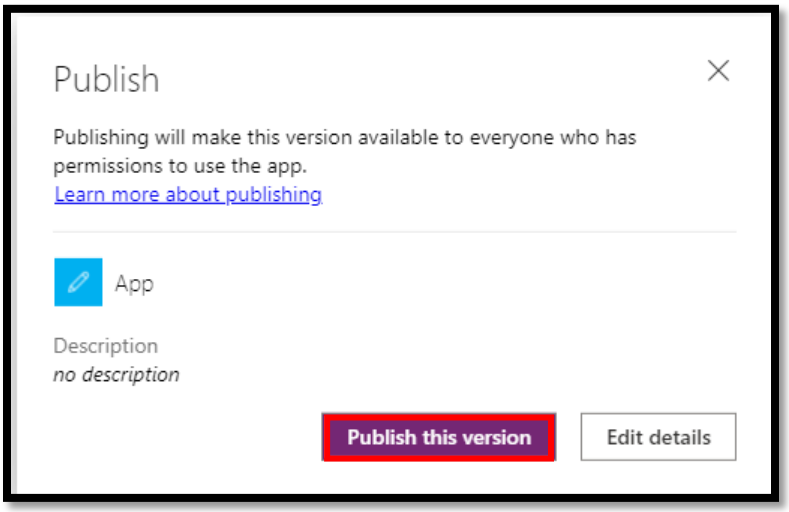
Once you play your app, you still have the same three screens to check out, but you will do so by using the icons within your screens to navigate between them.



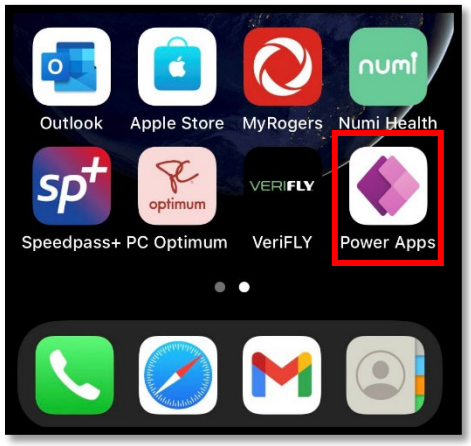
Running Your App on Smart Phone

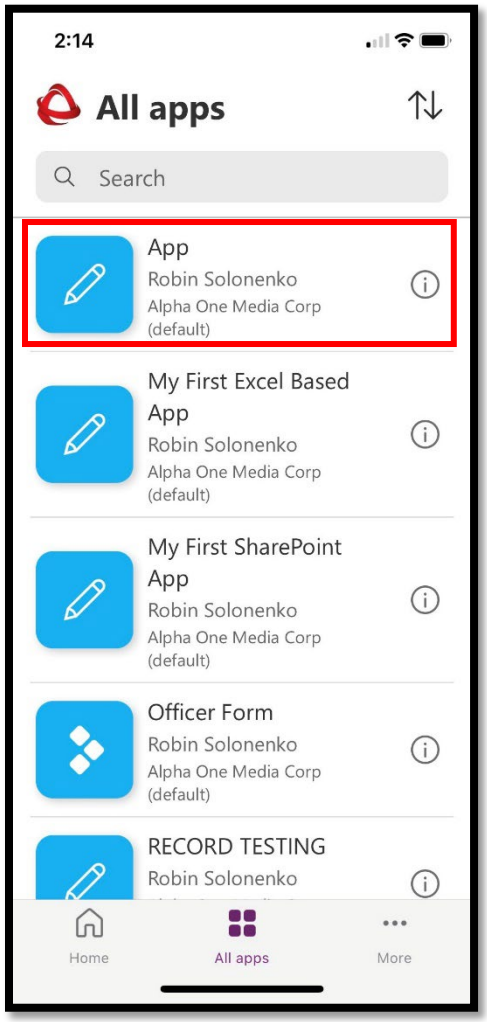
Now once you are finished you should save your app and then actually Publish to the Power Apps platform. You will not be able to use it on your smartphone until done.

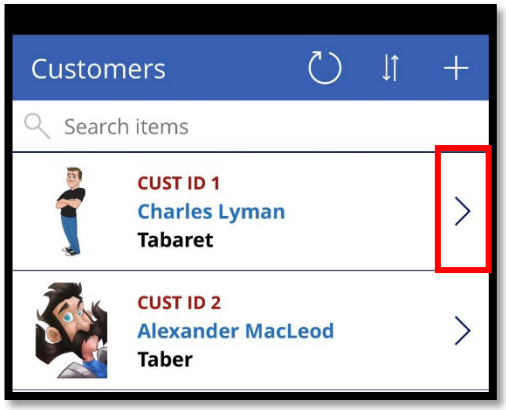




Now switch over to your phone and run the Power App application. If you do not have it yet, download it from the store and run it. You will have to login using the same credentials as your O365 account. Once in the app click the **All Apps** icon on the bottom of the screen to view all your apps that you have published, then select the app you wish to run. Once running you can now use it like it was intended.

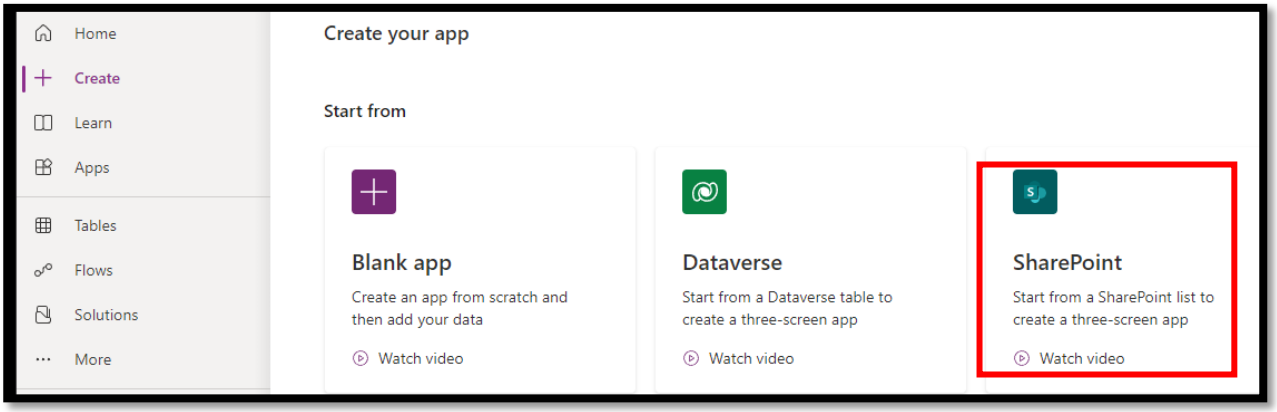




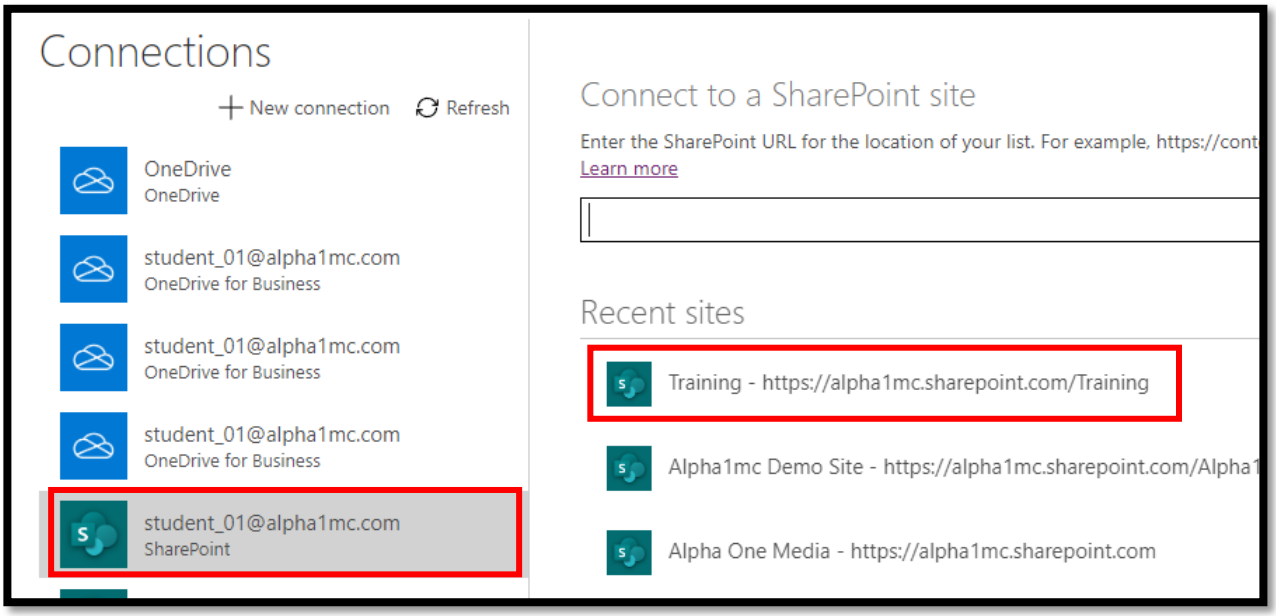


Lesson 2 Exercise

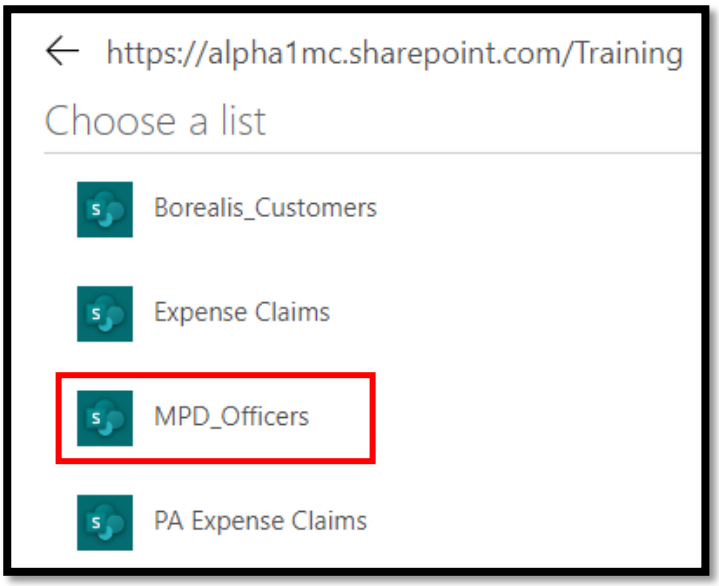
1. If you are doing this on your own and don't have access to **SharePoint**, then ensure your **Student Exercises Data.xlsx** file is already uploaded to your **OneDrive** and then just choose Excel options for this and the next step. Otherwise choose Create > SharePoint.



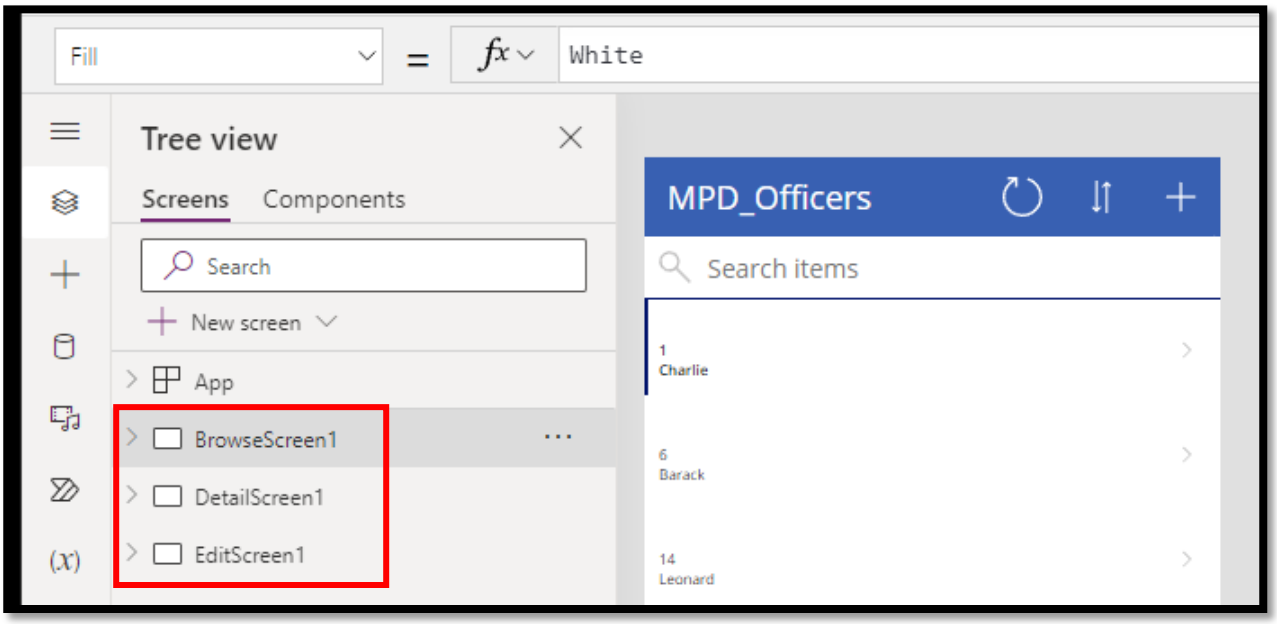
2. Select your SharePoint connection from the left and from the **Recent Sites** list choose the **Training** one. If it is not listed, you will need to copy and paste this URL: **https://alpha1mc.sharepoint.com/Training/**



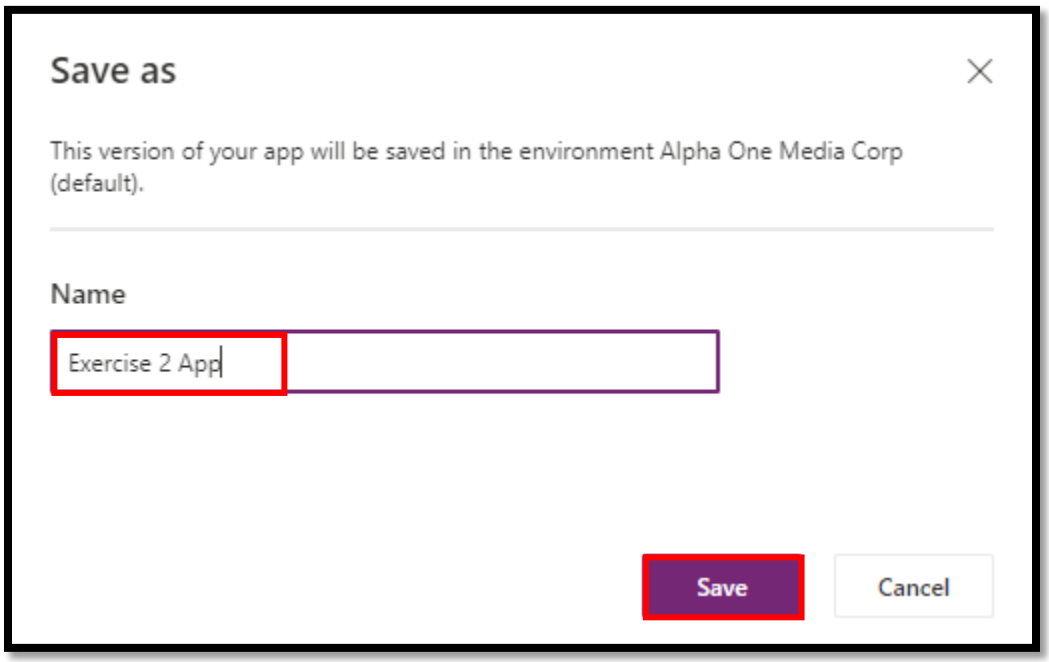
3. Now choose the **MPD_Officers** list and choose **Connect** on the bottom right of your screen.



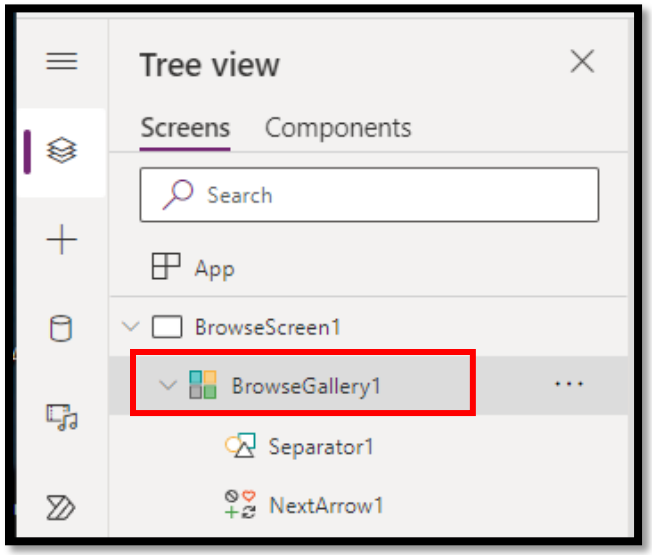
4. You are now looking at your 3-screen app – **Browse** Screen, **Detail** Screen and **Edit** Screen.



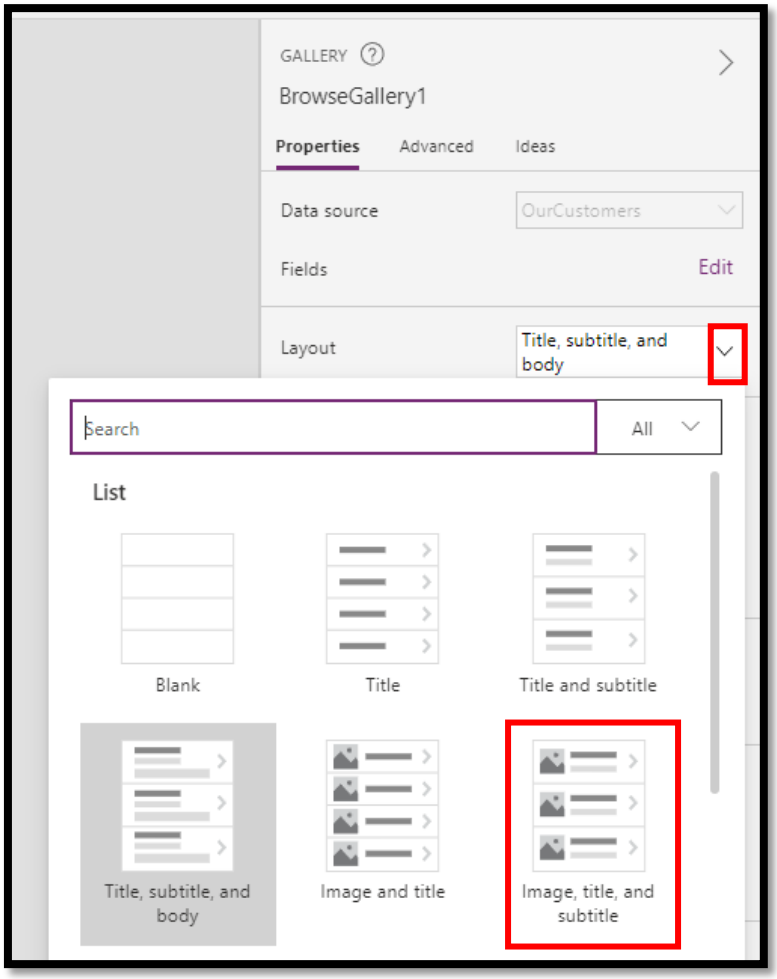
5. Save your app by clicking the **Save** button and giving it a name – **Exercise 2 App**.



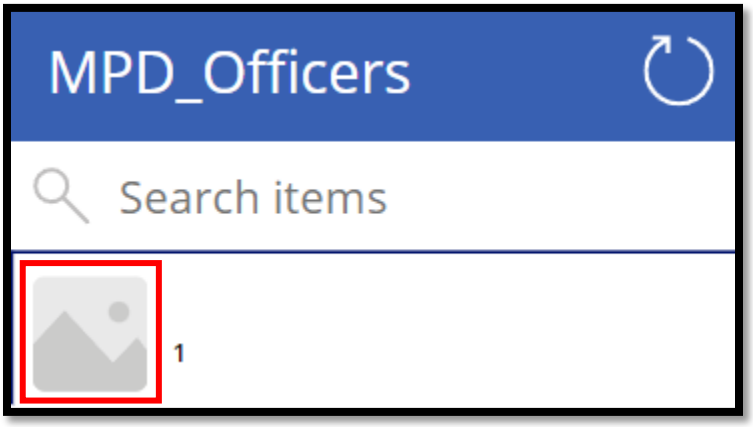
6. Now since we have an image field in our data, we will need to change the gallery layout to accommodate and display our image. First step is to change the layout. From the **Tree view** select the **Browsegallery1** object.



7. Now from the **Properties** pane from the **Layout** drop-down, **choose Image, Title, Subtitle**.

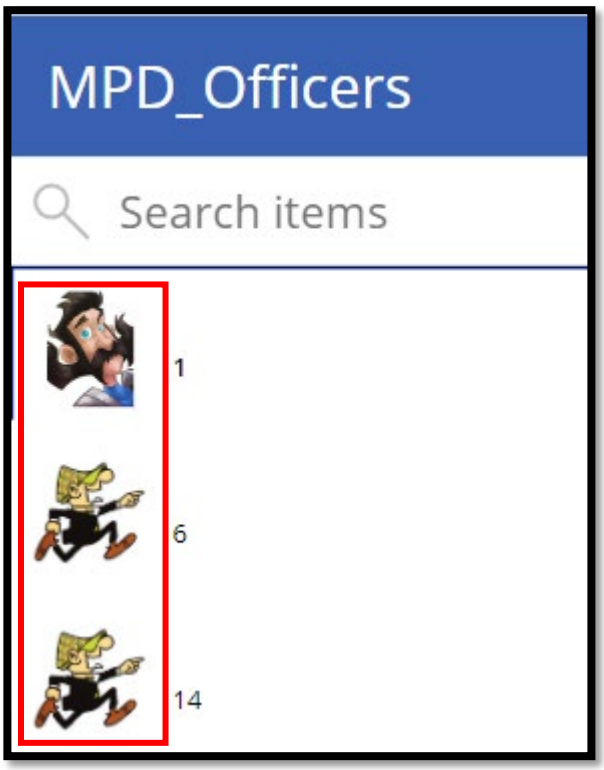


8. It should now change the layout to include the image place holder.



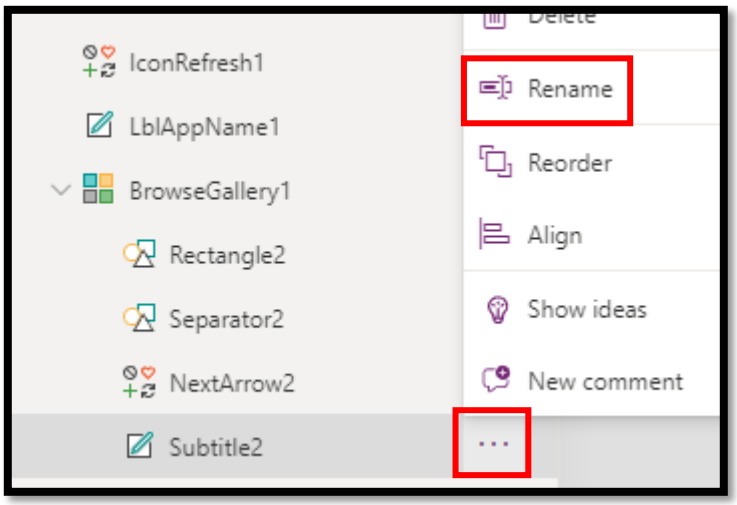
9. Now from within the **BrowseScreen1** canvas, select the **Image** placeholder and change the **Image** property formula to read: **ThisItem.Photo.** *(Photo is the name of the field in the SharePoint List)*

10. All you image place holders should change to a cartoon image of the officers. *(Note: If you are using Excel as a Data Source - you cannot have your Excel Data source open during these processes as it will show no data here).*

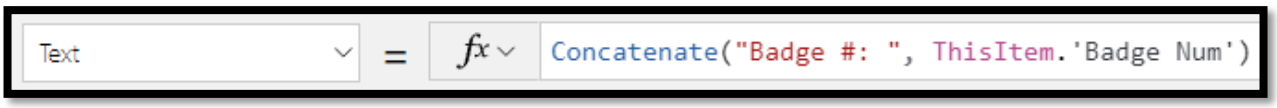


11. Currently, beside the image of the customer, the gallery displays the **Badge Num** field from your data source. Our objective is to have it read the **Badge ID**, **First** and **Last** name of the officer and their **Rank**. To do this, we will adjust the one that is already there and add in new ones with formulas.

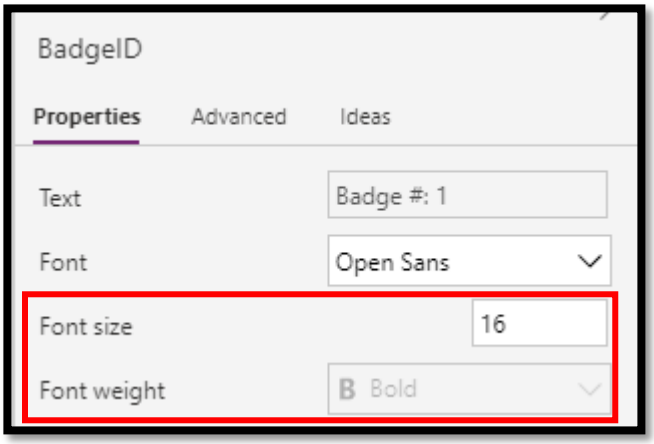
a. First, from the **Tree View**, select **Subtitle#** under the **BrowseGallery1**. It should be the one that displays the Badge Num and rename it to **BadgeID**.

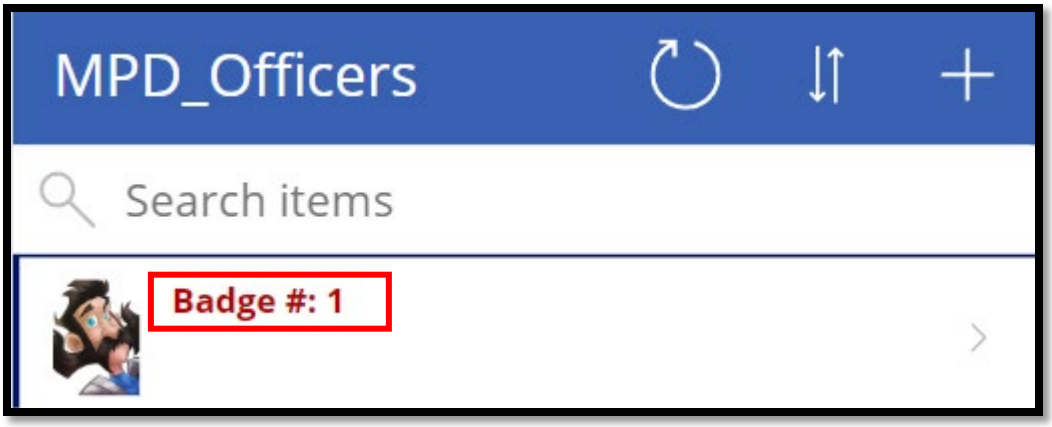


b. With the **BadgeID** object active use the **Concatentate** function and change the formula for the **Text** property as follows: **Concatenate("Cust ID: ", ThisItem.'Cust ID').**

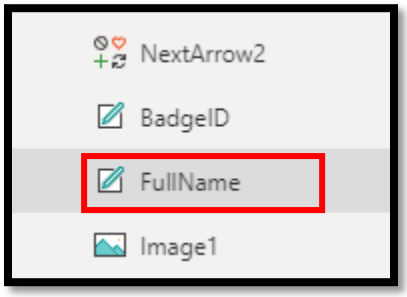


c. With the **BadgeID** still active, you can use the **Property** pane to adjust its font size, colour etc. Also align the field to the top of the gallery.



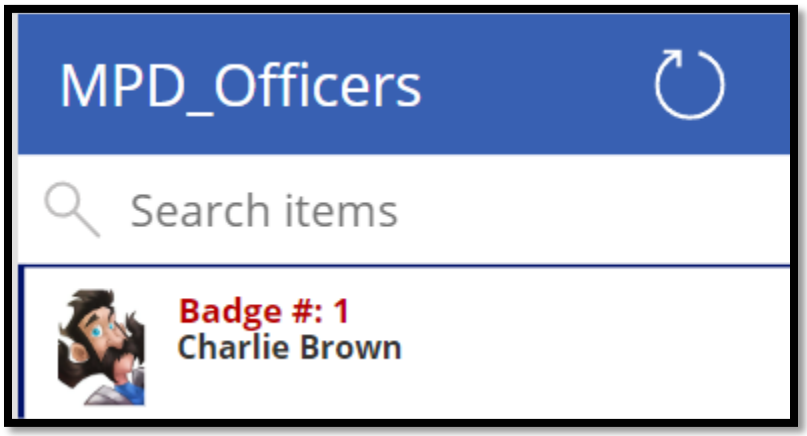


d. Now select the **Title#** object in the **Tree View** under the **BrowseGallery1** and rename it to **FullName**.

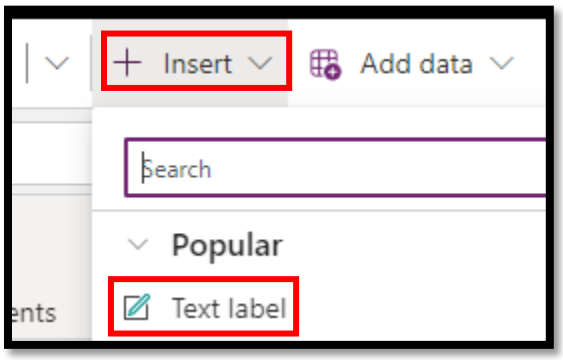


e. With the **FullName** object active, change the formula for its **Text** property to read the following: **Concatenate(ThisItem.City, ", ", ThisItem.Prov)**

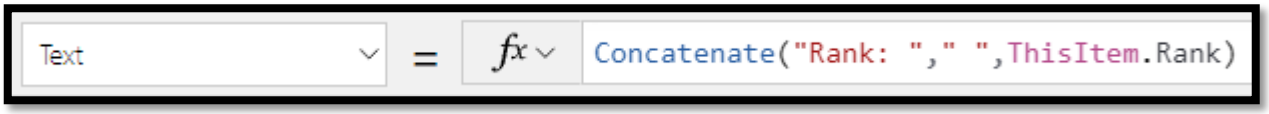
f. Adjust its font size, colour as necessary and place the field under the Badge #.



g. Now we must **insert** a new object between the **Badge #** and the **FullName** fields and have it point to the **Rank** field in the SharePoint list. Ensuring that the border of the first record is active, from the **Insert** menu choose **Text Label**.

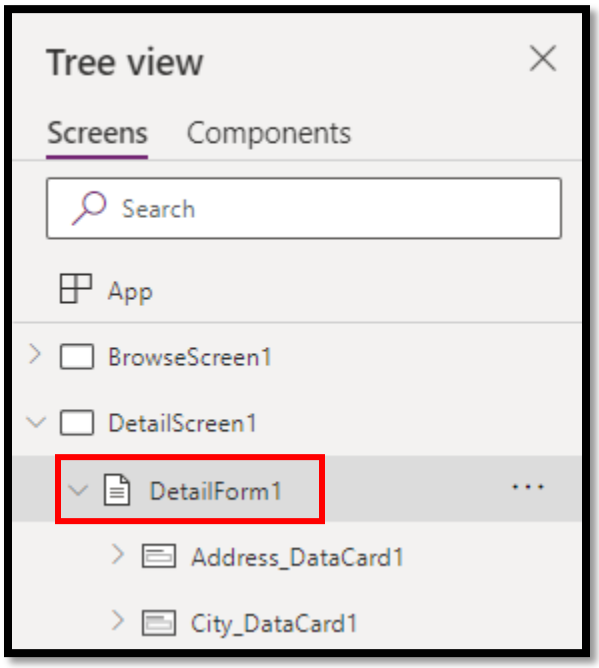


h. Now size and position this new label under the Full Name field. Then change its text formula to the following: **Concatenate("Rank: "," ",ThisItem.Rank).**

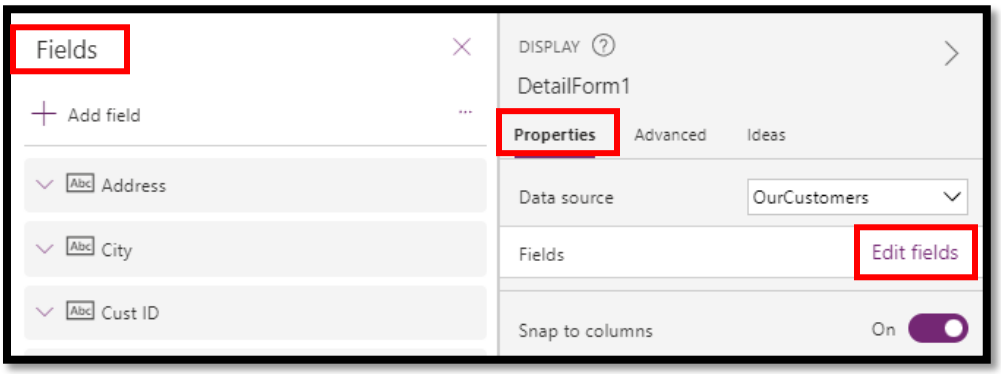


i. Using the **Properties** pane adjust the text size, colour etc and line up the fields within the gallery placeholder.

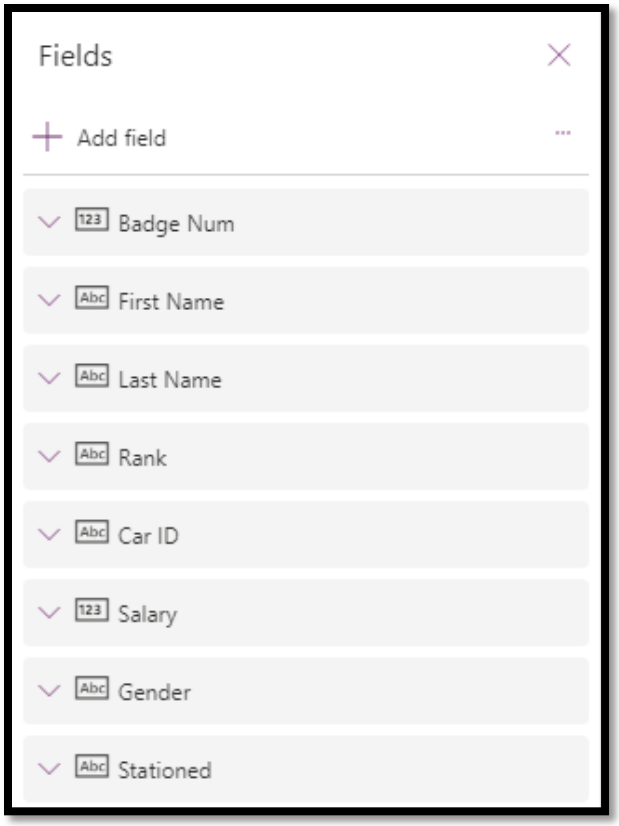
12. Now we need to clean up the **DetailScreen1** object. So, from the **Tree view**, select the **DetailForm1** object.



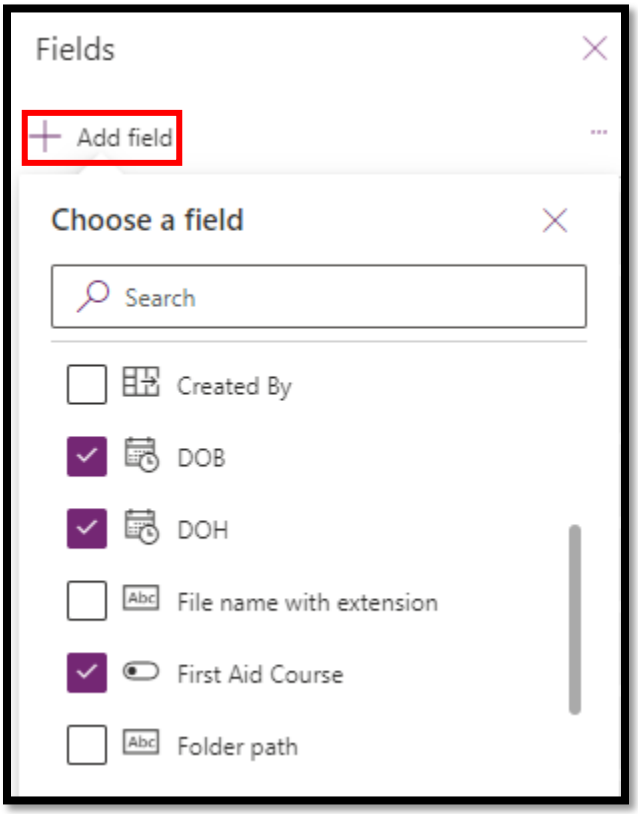
13. From the **Properties Pane**, click the **Edit Fields** link to open the **Fields Pane** to the left of the current pane.



14. Adjust the field list order from the **Fields** pane by clicking the **…** and choosing **move up**. Or by dragging and dropping the fields to match the order below.

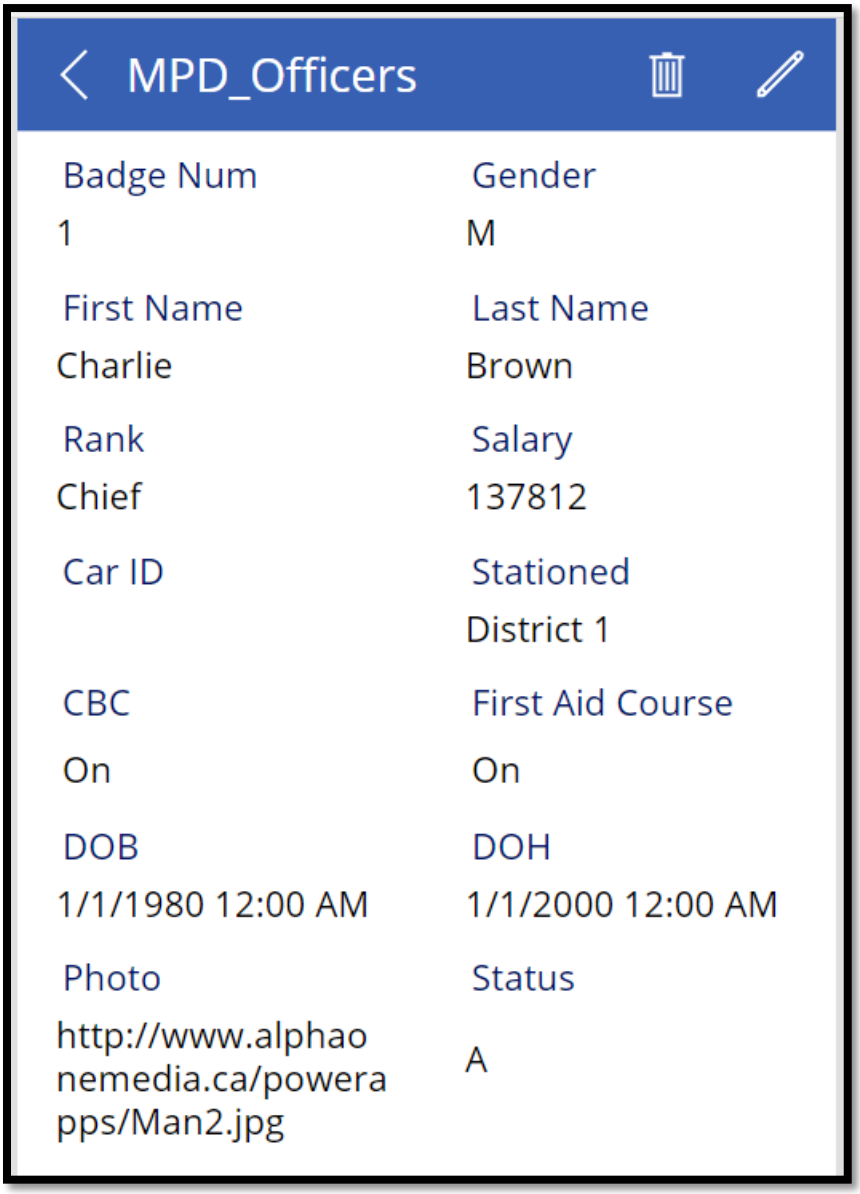


15. Still within the Fields pane, click the Add Field and add the following fields: CBC, DOB, DOH, First Aid Course, Photo and Status.

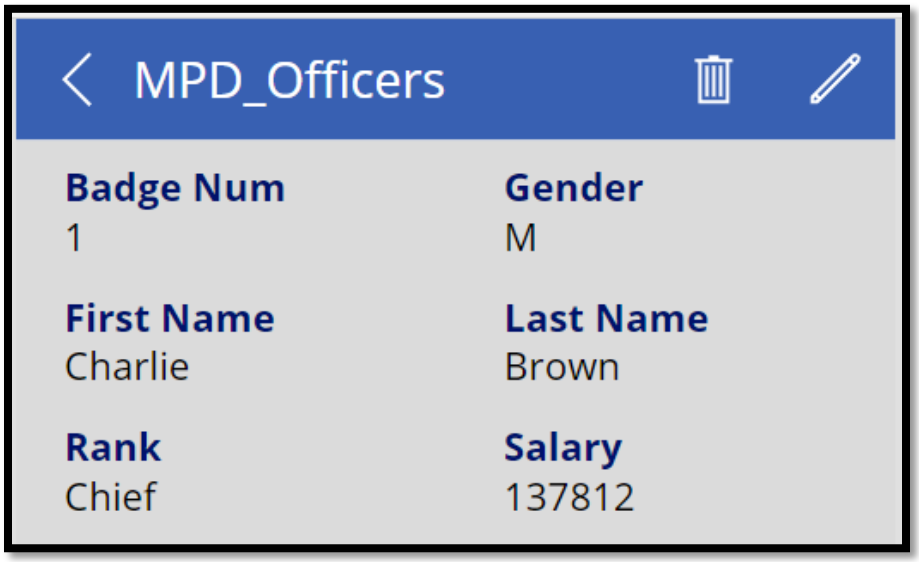


16. In the Properties pane, adjust the **Columns** setting to **2**.

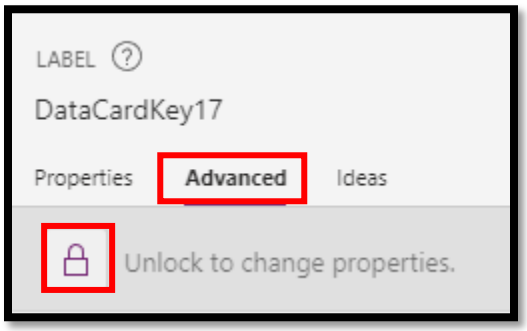
17. Adjust the fields in the **Fields** pane so that your Detail Screen looks like the following.

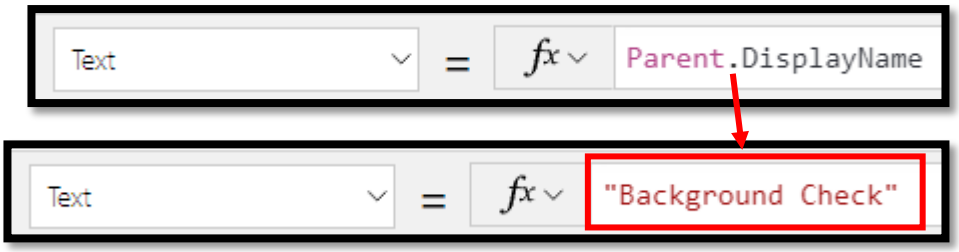


18. Apply bold to all the labels, align the values to the labels (adjust the padding for each) and color the background colour to your choice.

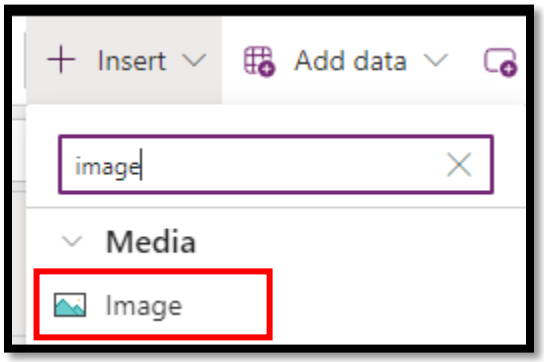


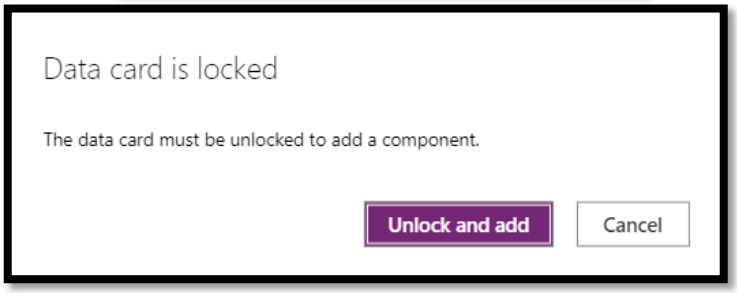
19. Rename the CBC label to read **Background Check.** When you are renaming the labels, you must choose the **unlock** button on the **Advanced** properties pane to change the **Text** property.

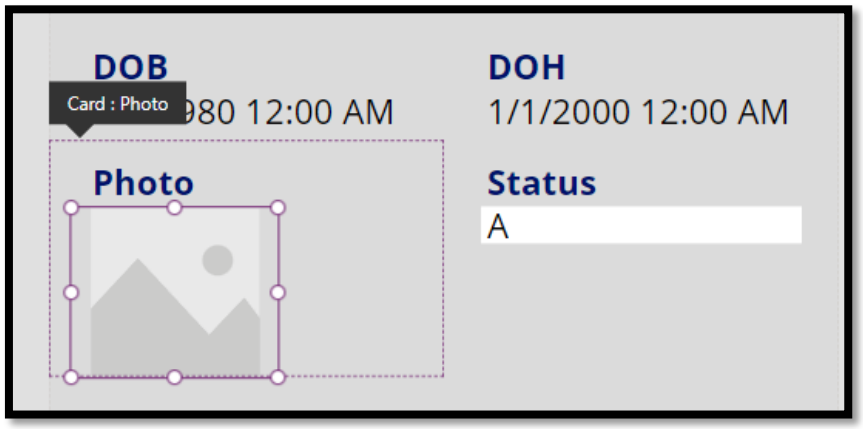




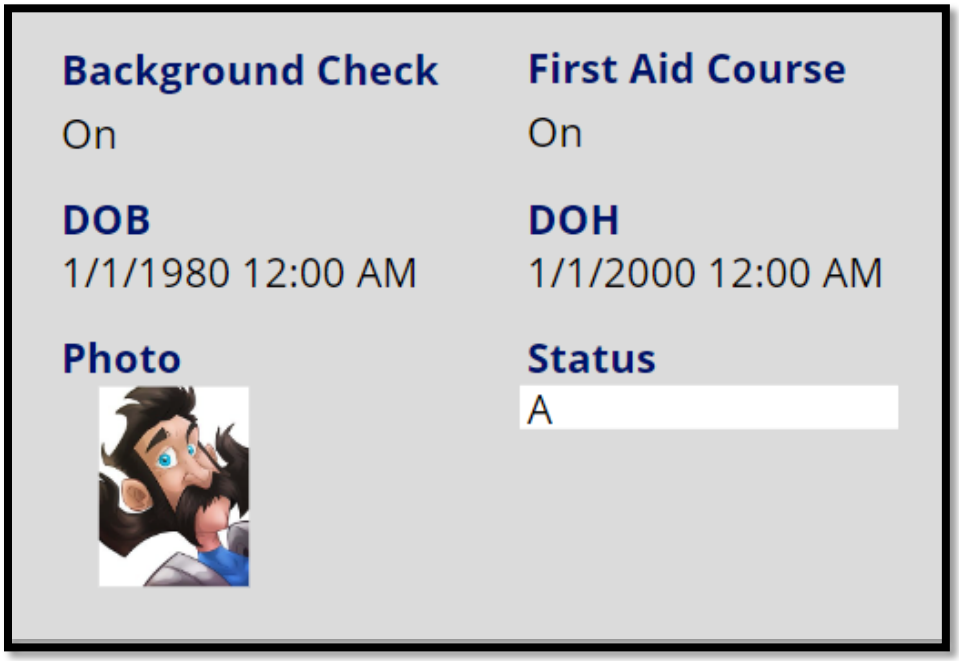
20. Now with the **Image** field, we would like the photo to display instead of the URL. To do this, first delete the object from the tree *(DataCardValue# under the object Photo_DataCard#)*, then with the photo card still active from the **Insert** menu insert the **Image** object. You will be prompted to **Unlock and Add** that object.





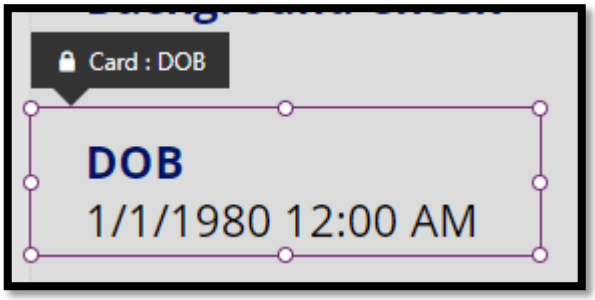


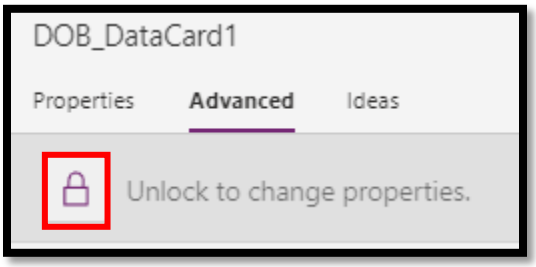
21. With the **Image#** placeholder still selected, change the Image property in the formula bar to read: **ThisItem.Photo**. This will connect to the field called Image and because it is an image object, it will display the image.

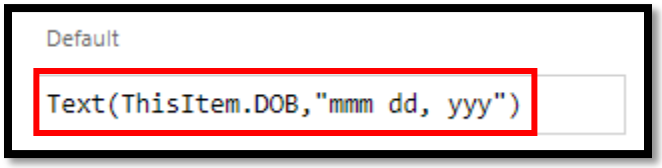


22. Now let us change the name of the screen on top. Select the object in the dark blue that is labelled **MPD_Officers** and using the formula bar**,** change the **Text** property to: **Concatenate(First_DataCard1.Default," ",Last_DataCard1.Default)**.

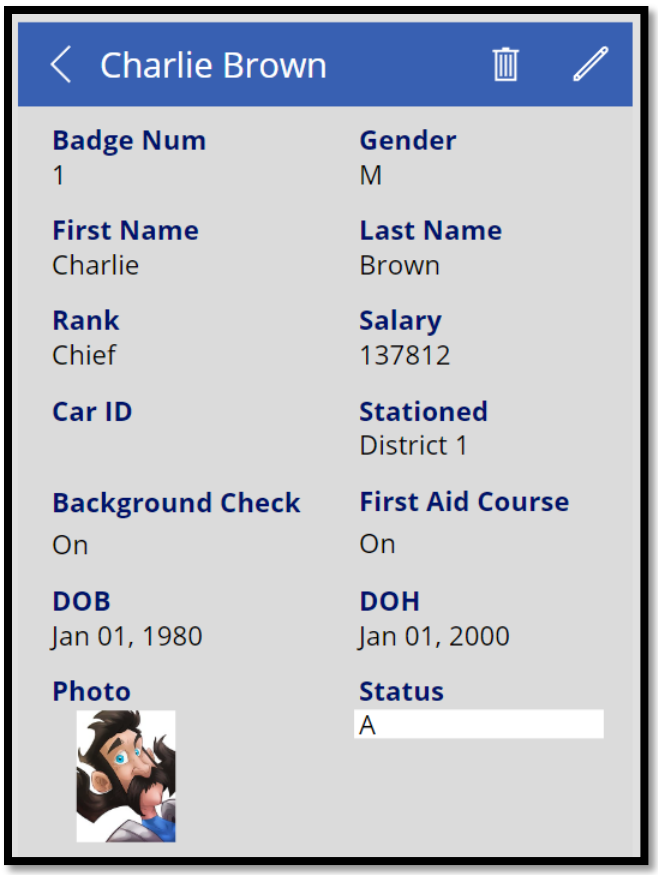
23. Currently the **DOB** & **DOH** fields show both date and time. We need to remove the time portion and to do that, select the **DOB_DataCard1** object (the one that holds both the Label and Data fields) then click the unlock button from the **Advanced** tab so you can edit the **Default** property to read: **Text(ThisItem.DOB,"mmm dd, yyy"). Repeat the process for the DOH data card.**



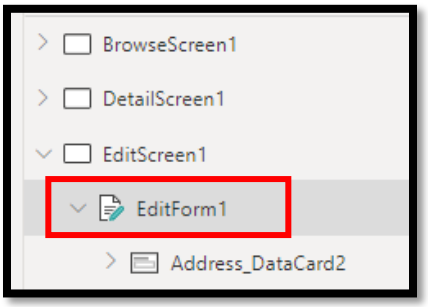


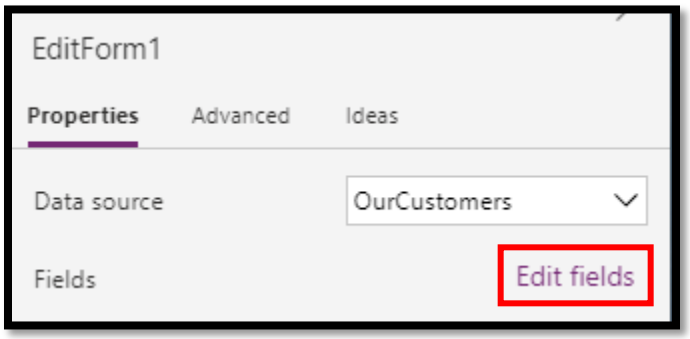


22. Your completed **Customer Info** (detail) screen should look like below. Save your work.

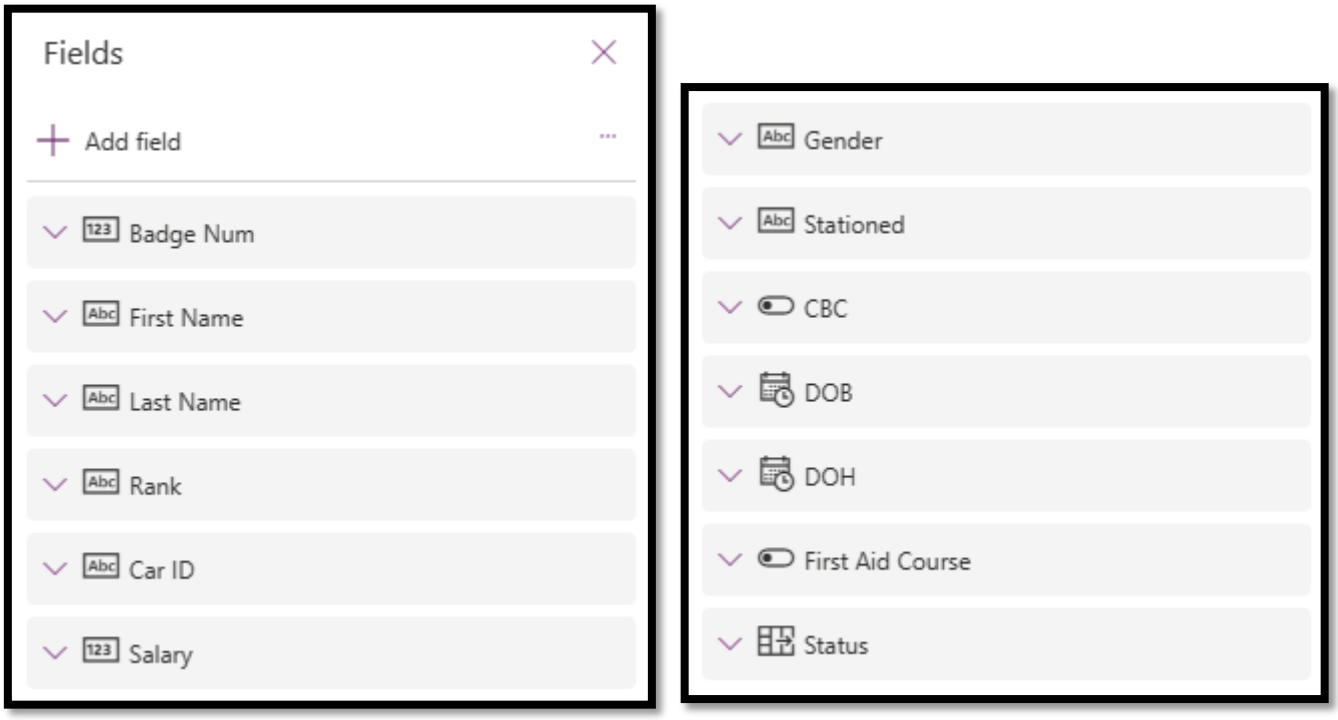


23. Let us now switch over to the **EditScreen#** object and clean this screen up. First thing is to arrange the order of fields. It will be done the same way we did the detail screen. Select the **EditForm1** object within **Tree view**, then click the **Edit Fields** link in the **Properties** pane.

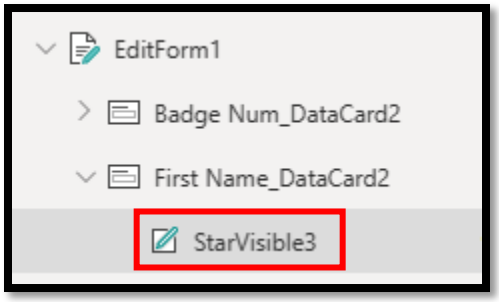




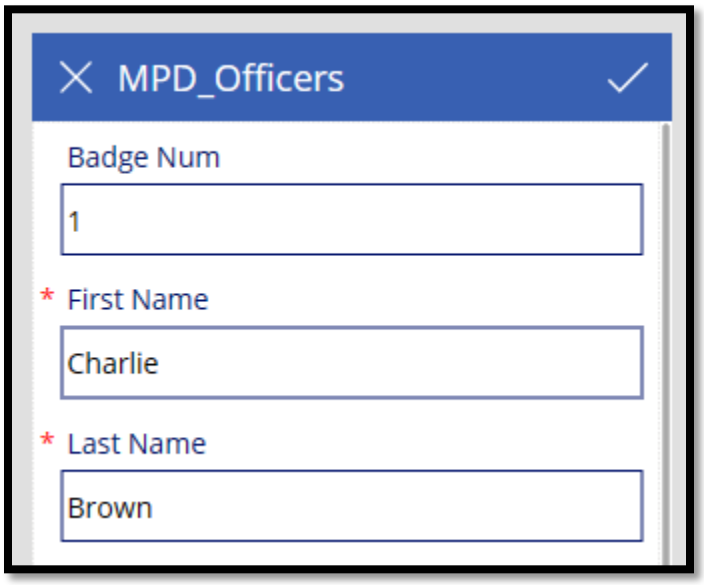
24. Using the **Fields** pane, align your fields based on the screen shot below and add any missing ones.



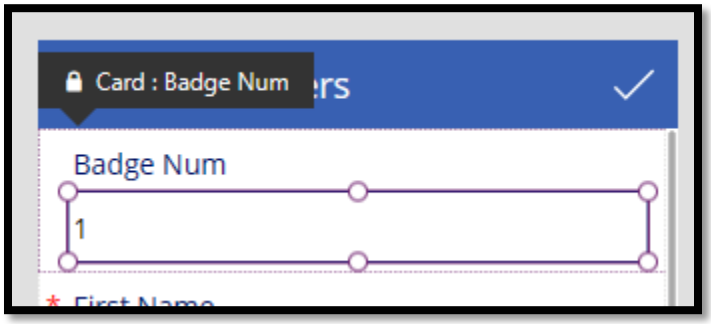
25. Now we want to turn on the required fields symbol (*) for the **First**, **Last** and **DOB** fields. To do this, from the **Tree view**, select the **StarVisible** Object for the **First_NameDataCard** object and then from the **Advanced** property pane click the **unlock** button. Now from the **Properties** page you can click the **Visible** slider and then in the **Formula** bar change the **Visible** property to read: **true**. Feel free to change the colour of the asterisk.



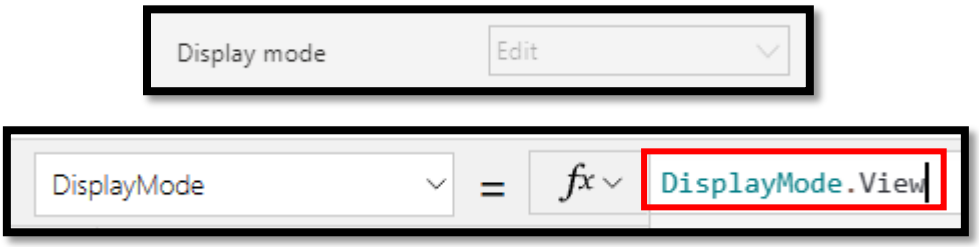
26. Repeat the last step for the **Last Name** and **DOB** fields.



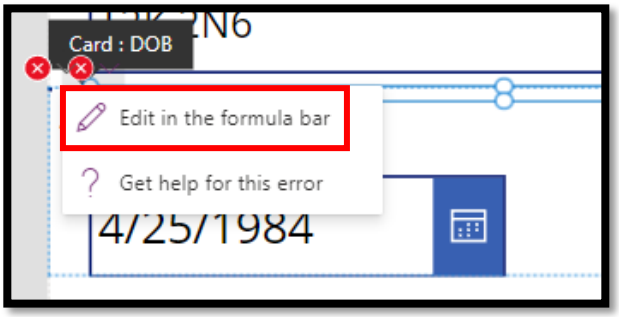
27. From within the canvas, select the **Badge Num** value.

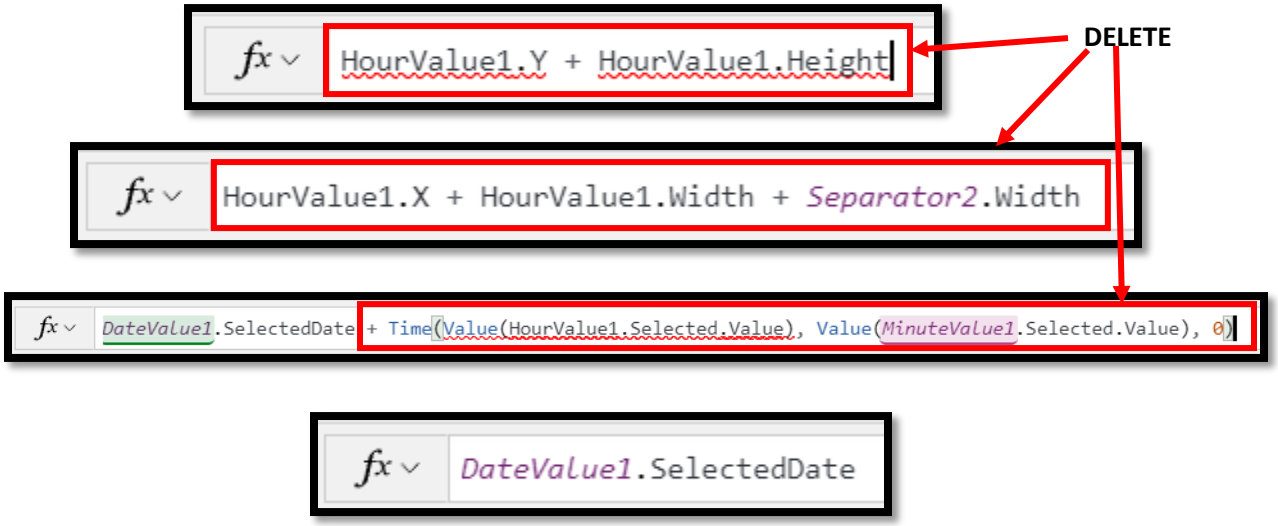


28. Click the **Unlock** button from the **Advanced** pane and then click the **Display** mode button from the **Properties** pane. Then from the **Formula** bar change the formula to **DisplayMode.View**. This will ensure that no one can edit the Badge Num field.



29. We also do not want to be entering the "Time" someone was born or hired, so delete the **Hour** and **Seconds** placeholder in the **DOB** and **DOH** placeholder. However, when you do that, you will notice some errors appear within the DOB placeholder. To fix these click the **red x** and choose **Edit in the formula bar** and delete any references to the hour or minutes objects. *(Note: there will be another 6 errors not listed here that need to be fixed)*





32. Finish this canvas off by **bolding** the labels, **colouring** the text input boxes and change the heading of the screen to read "**Editing Name of Officer**".

33. Do a **File > Save** to make sure you saved all this work. Play your application via the browser to ensure it works correctly. Then **Publish** this to the cloud, go to your phone and connect to it via the **Power Apps** app to make sure it works there as well.

POWER APPS FOUNDATIONS

LESSON 3

Building a Canvas App

LEARNING OBJECTIVES

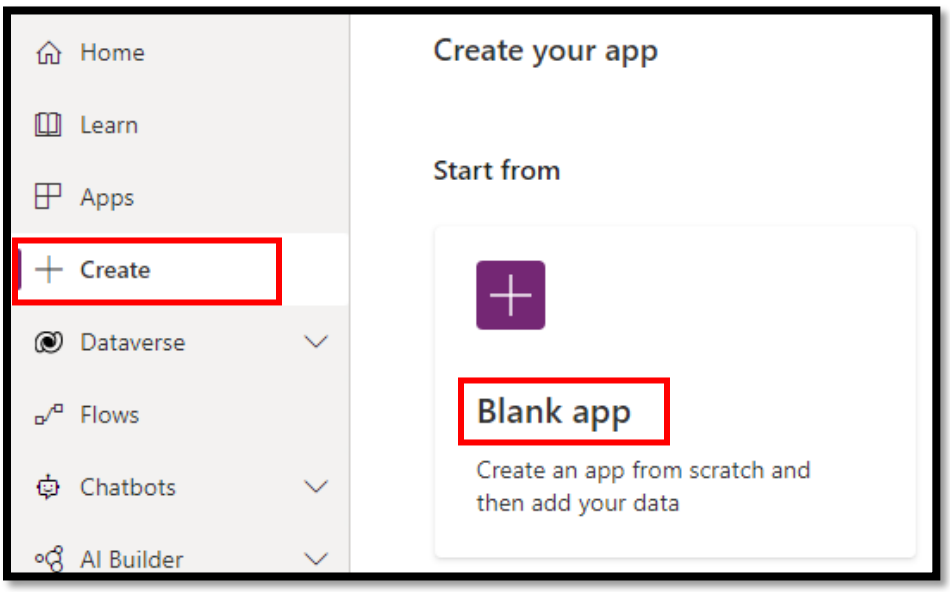
- Master Screens
- Building Object Library
- Creating Buttons
- Creating Form Galleries
- Search and Filter Functions
- Data Validation

Lesson 3: Building a Canvas App

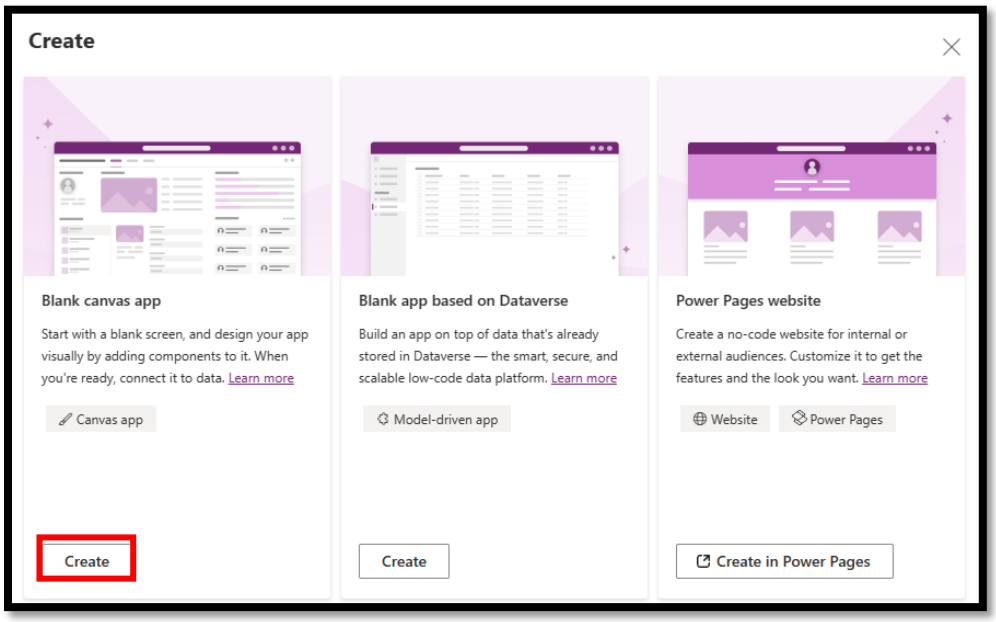
App From Scratch

In the previous section you created your app based on an existing data source. When doing it that way, Power Apps will auto create the three basic screens to view your data, edit your data and add new data. In this section will create your first canvas app with no linkage to data to start and one basic screen. You will now learn how to create all the screens you need and link to one or more data sources (tables) as you require them.

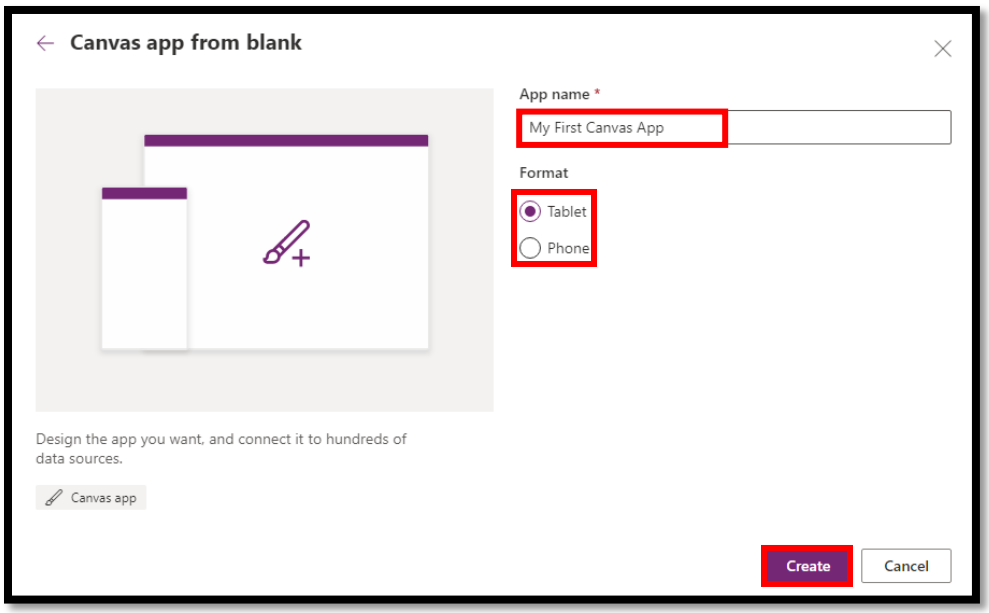
1. From within the Power Apps environment (but not from within an existing app), click the **Create** button from the left menu then **Blank App** from the **Start From** section.



2. Click the **Create** button in the **Blank canvas app** panel.



3. Give your app a name and choose the format type then click **Create**.

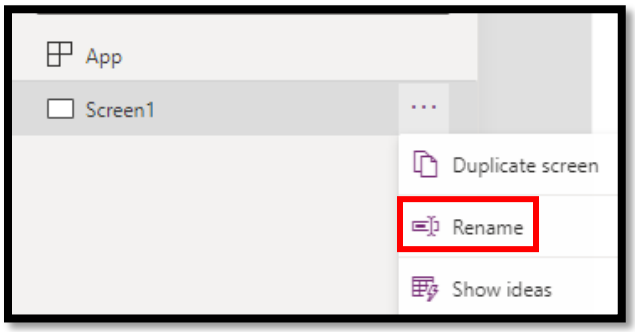


4. You will now have a blank canvas app with one **Screen1** created underneath the **Tree View**. You are now ready to create your own objects.

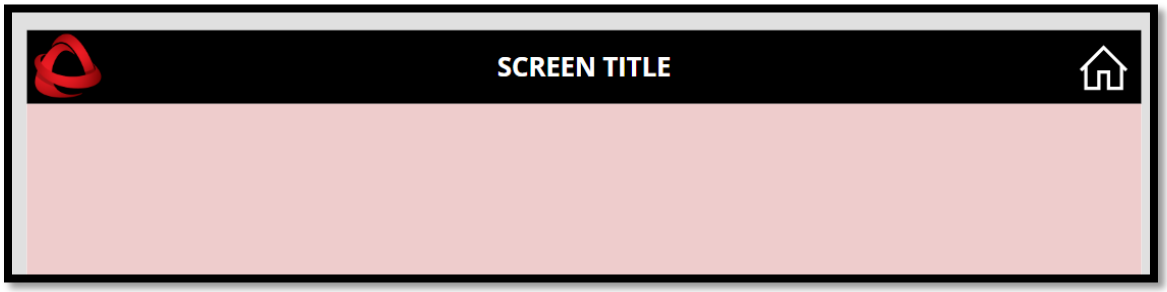
Master Screens

The more screens your app will have the greater the need to have a Master screen to keep consistency within your app. Basically a Master screen is a template in which you may have such things as titles, standard icons like refresh, search and even a company or app logo. You simply create a Master screen and then all subsequent screens are based off the Master. One important note about creating new screens off the Master screen – there is no linkage between the two. Meaning if you want to change the layout of the subsequent screens you will have to change each one manually.

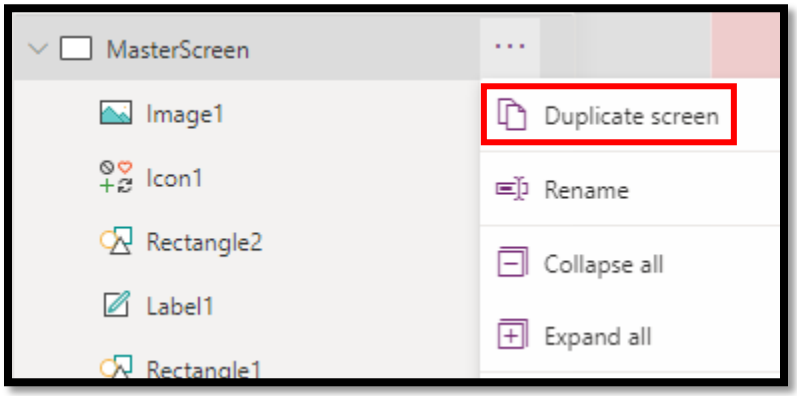
1. To create a **Master Screen**, you can start by renaming the default **Screen1** to **Master Screen** by clicking the 3 ellipses and choosing **Rename**.



2. Now you can add titles, shapes, logos to your **Master Screen** of your choosing. In the screenshot below we added a black rectangle with white centered text, our corporate logo, the **Home** icon and a large reddish rectangle where our content will be displayed.



3. Every time you need to build a new screen, you simply click the **ellipses** to the right of the Master Screen and choose **Duplicate**, rename it and now you have another screen with the same basic elements.

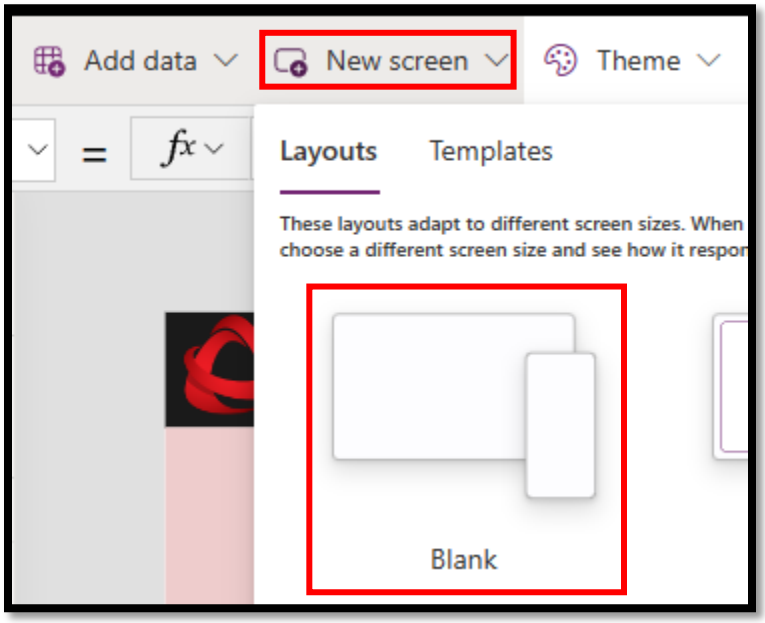


Building an Object Library

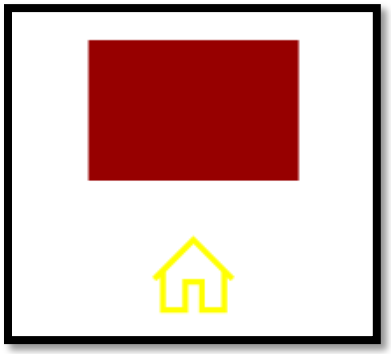
As indicated in the previous section, when you duplicate the Master Screen and create new screens there is no linkage between the two. Therefore, if you decide to change the back colour of the title rectangle, you will have to manually change it on all duplicated screens.

To counter that, we will have you build an **Object Library** (another screen) where you will house all the objects you will use within your app. Then the Master Screen objects will reference what it needs from the Object Library. Moving forward, you will change the property of the object tin the Object Library and all screens with that object will adjust accordingly.

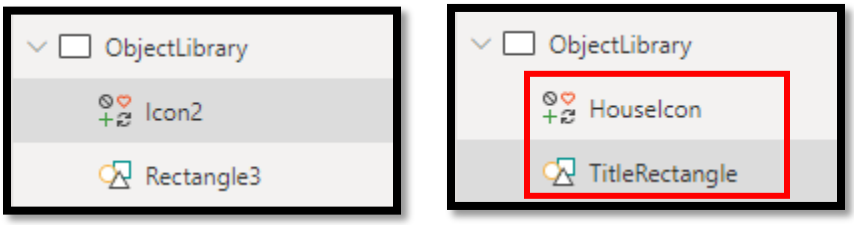
1. Create another new screen from the menu bar by clicking **New Screen > Blank** layout and rename it **Object Library**.



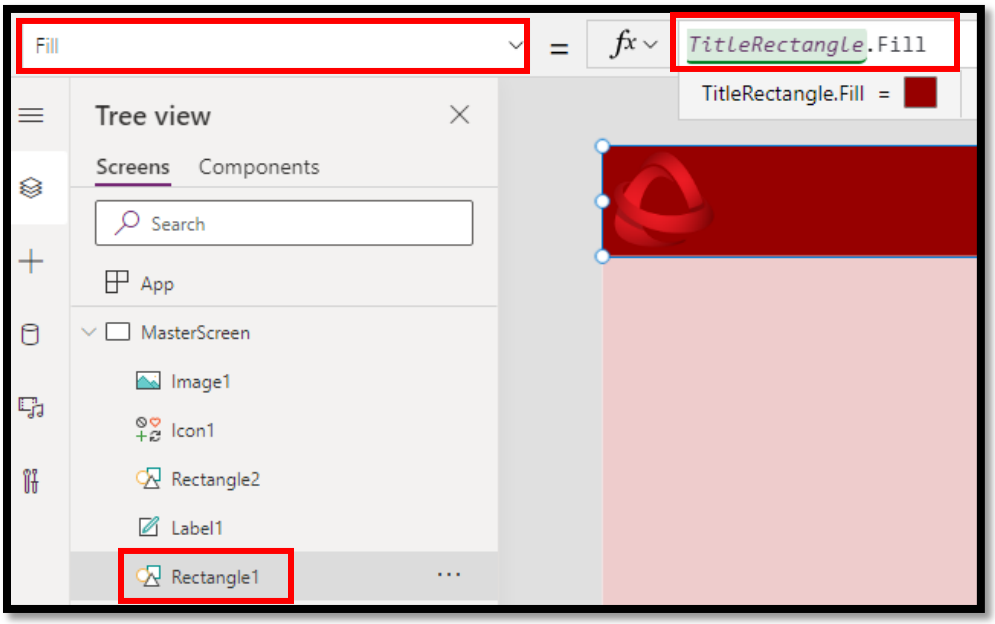
2. Recreate some or all the objects from your **Master** Screen – in our example we recreated black header rectangle and home icon. Then change the colours of each (they do not need to be exact same size or placement).



3. From within the **Tree View**, locate those same objects and rename them something more meaningful.

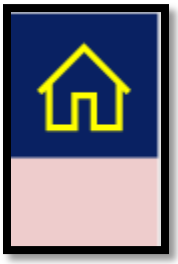


4. Still within the **Tree View** pane, select the object in the **Master Screen** list that you want to link a property to from the **Object Library**. In our case it will be **rectangle1**. From the property drop down list select the property that needs to be linked (in our case **FILL**) then in the formular bar adjust the formula to point to the property of the object in the Object Library. In our example, the rectangle on the title background will now take the reddish colour of the rectangle on the **Object Library** page.



5. To test the linkage out, return to the object in the **Object Library** screen, adjust the fill colour and return to the **Master Screen** to see if it updated.

6. Repeat steps 2 to 4 for every object that you want to be linked.

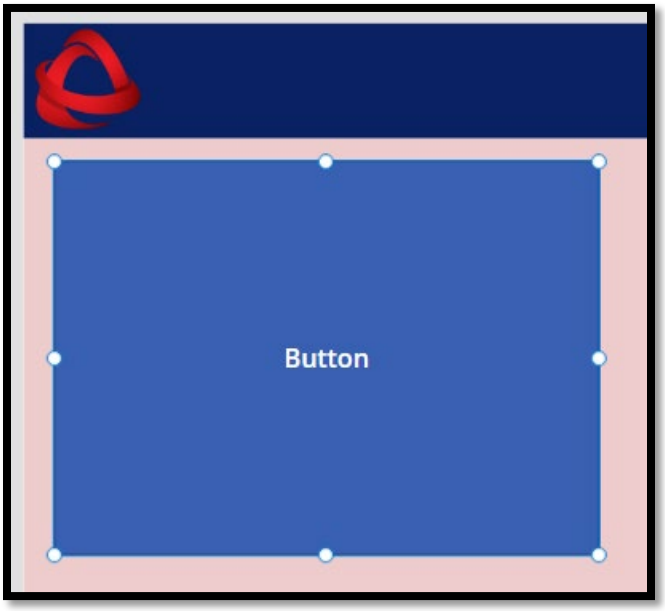


Creating Buttons

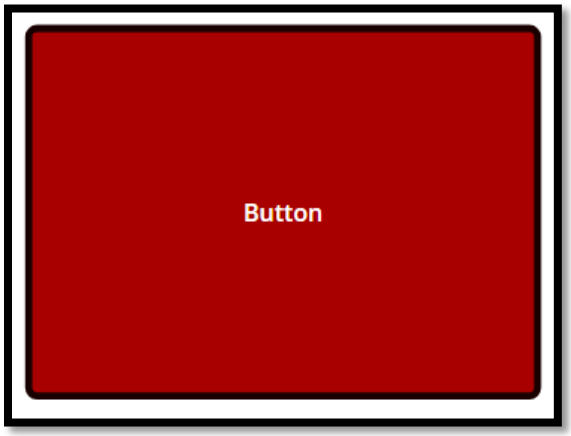
At some point you will have to create buttons to help create navigation within your app. Do not mistake these for triggers (On actions) that can also act as navigation features. Buttons come built in with a hover and click status which can change the appearance of such button. Where assigning a trigger to an object (icon, text box) can be a form of navigation but the appearance of them does not change when you hover over them with the mouse.

1. From your screen that you want to create the buttons on, choose **Insert** from the menu bar and select **Button** from the **Popular** group.

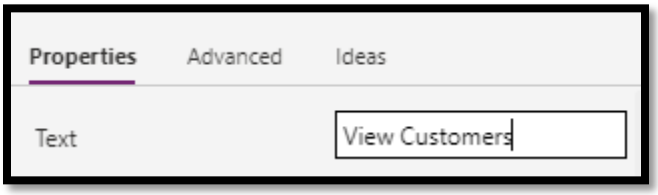
2. Size your button to your screen as you see fit.



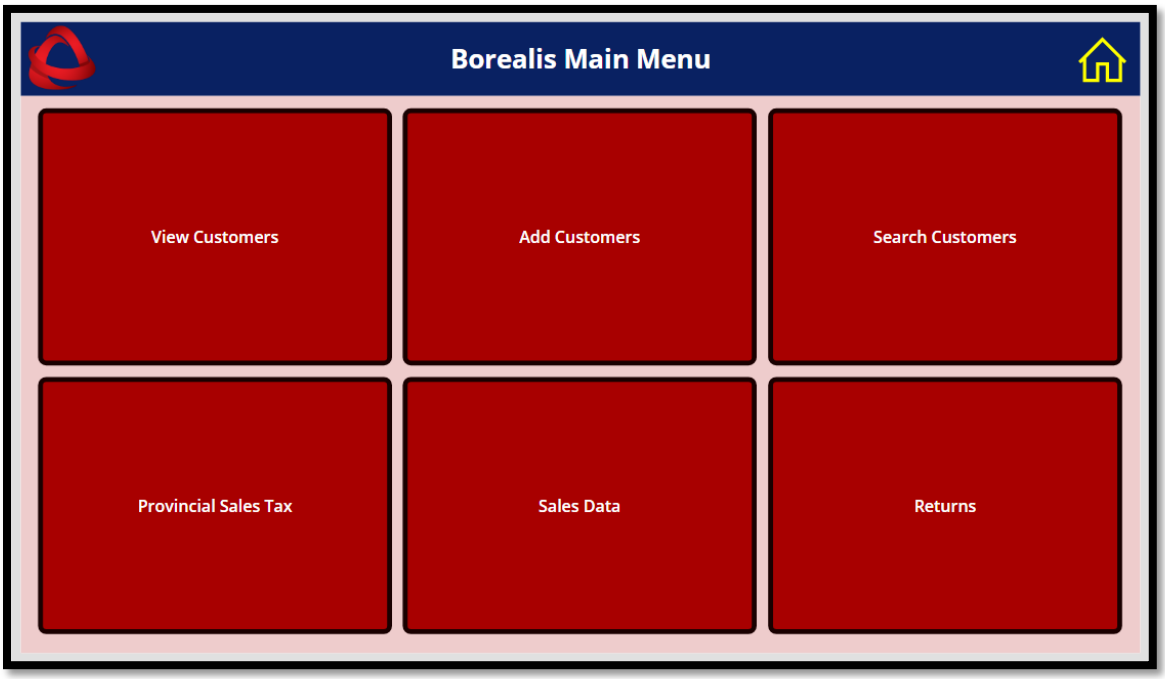
3. Now here is a good example of needing one of these buttons on the **Object Library** screen as we will have a few of these. So, copy and paste this button into the **Object Library**. Then adjust some basic properties like box or text colours.



4. Return to the **Main Screen** with the original button, select it and adjust its properties to point to the button in the **Object Library** and its same properties that you changed. It should now be identical as the one in the Object Library. Now is a good time to update the text on the button as well.



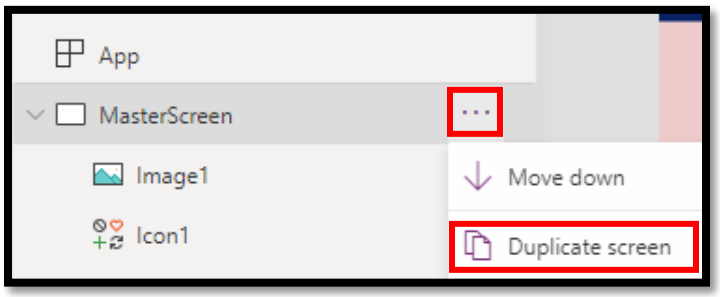
5. You can now copy and paste your button as you require and changing the text as you need. Each button now maintains a linkage via the function to the properties on the one in the Object Library. You can test that by changing one of those properties to see if they all change.

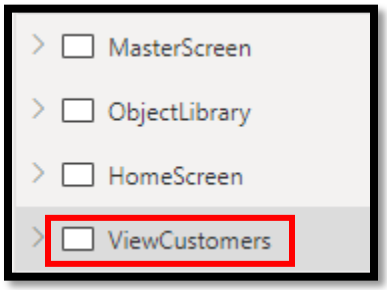


Attaching Galleries to Screens

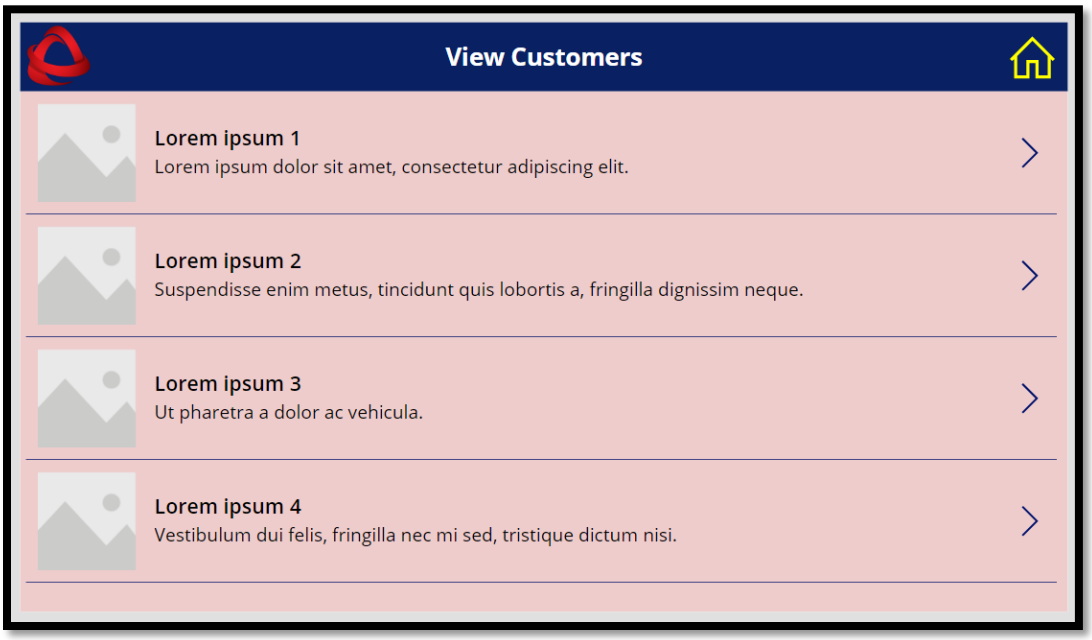
If you remember from the previous chapter, a gallery is simply a list that displays your records that is pulled from your data source. Well since you did not connect to a data source, Power Apps did not create you a gallery. You can create one manually and then connect to the data you require to be displayed.

1. Duplicate the **Master Screen** if you have not already done so and give the new screen an appropriate name.

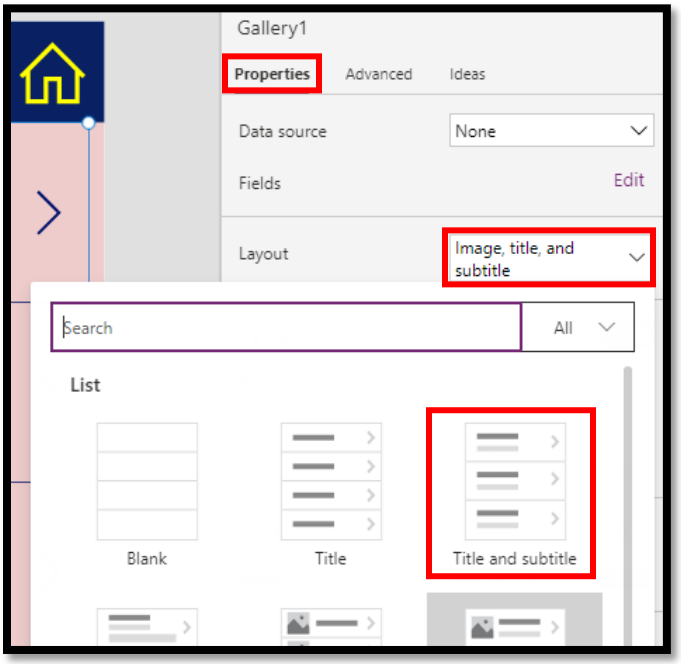




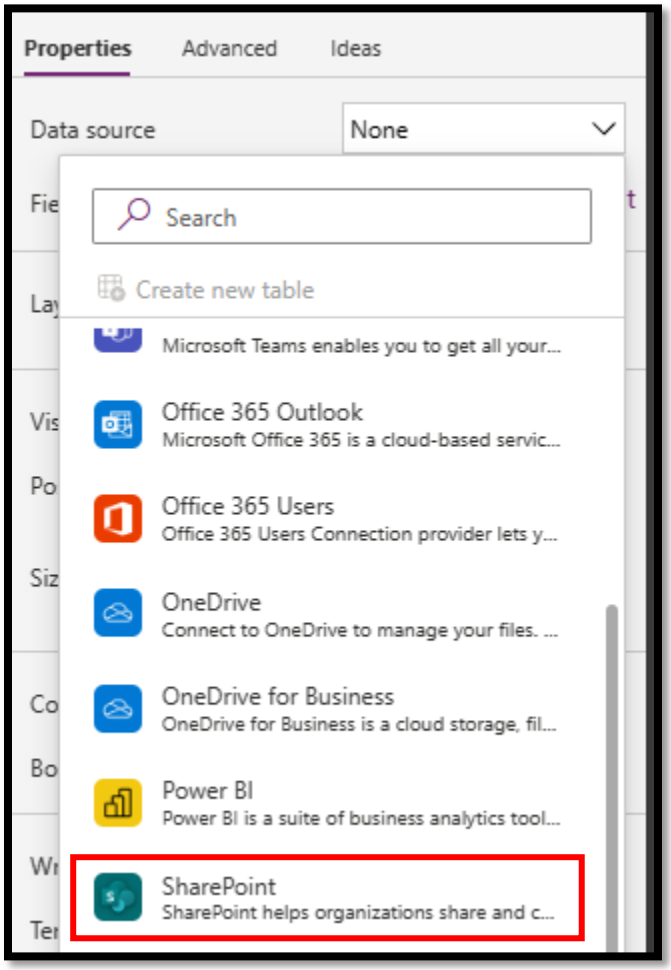
2. While on your new screen (**ViewCustomers** in our example), from the **Insert** menu > **Vertical Gallery**. It will add a pop-up form to link to a data source, you can close it at this time. Adjust the gallery to fit your screen.

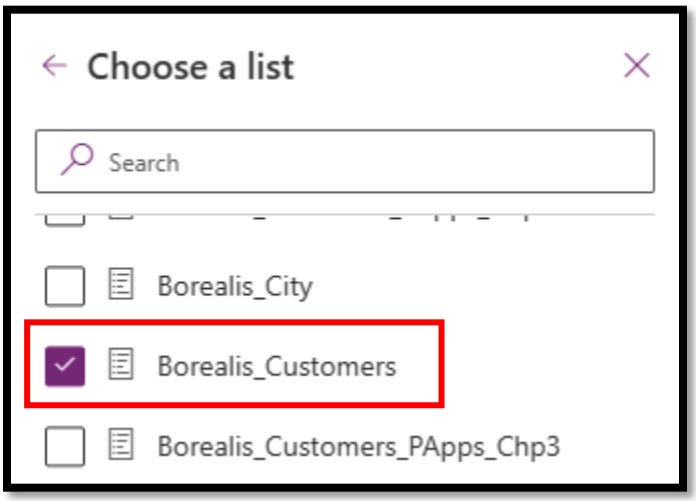


3. With your gallery selected, let us change the **List Layout** as you have more options now that you have created this gallery. From the **Properties** pane in the **Layout** field dropdown, select a type of layout that you desire.

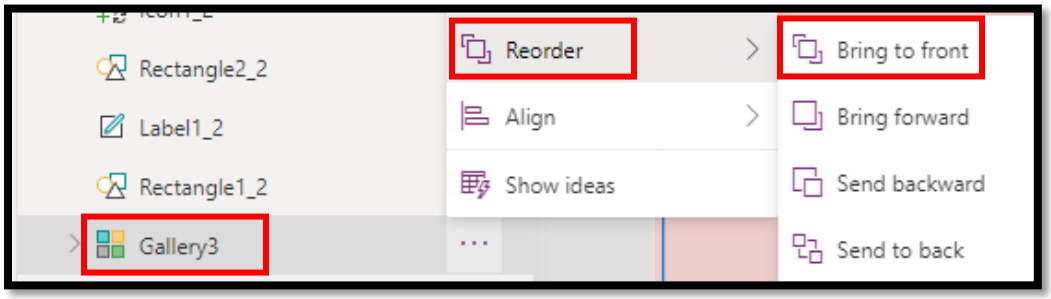


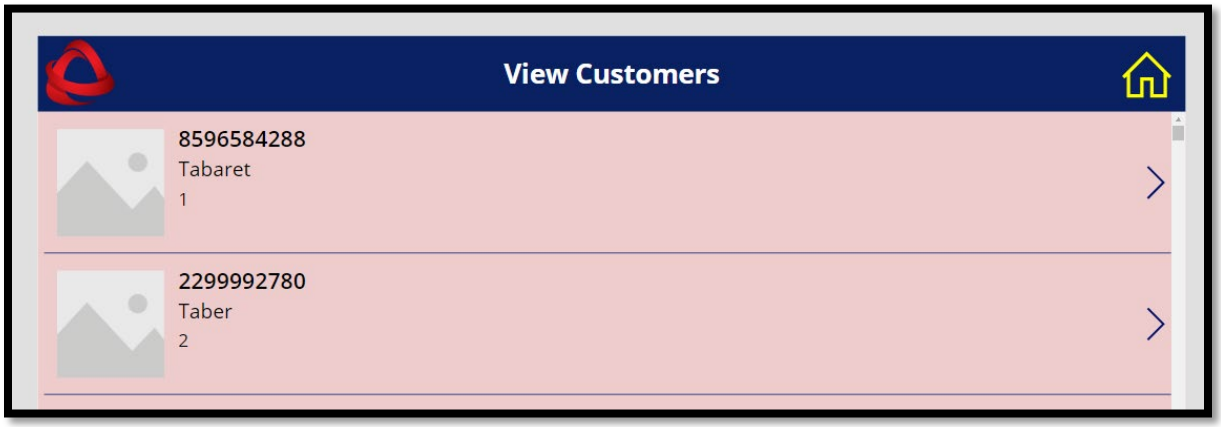
4. Now we need to attach it to a data source. We are going to use the SharePoint list called Borealis Customers (*You can use the Excel data provided if you do not have access to SharePoint*), from the **Data Source** drop down select **SharePoint** under **connectors**, then select your **Borealis_Customers** list.





5. If you do **NOT** see your fields (based on the Layout you selected in step #3, the issue may be the order placement of your objects. So, from the **Tree View**, select your gallery, then click the **ellipses** and choose **Reorder > Bring to Front**.



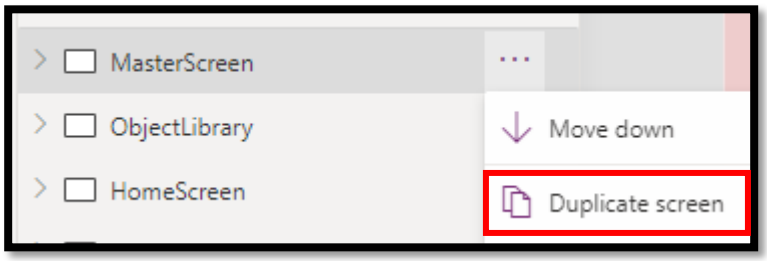


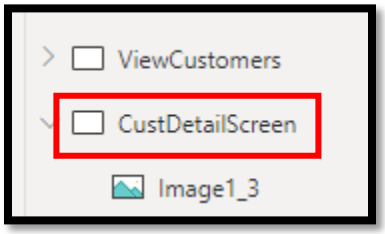
Creating Form Galleries

Unlike simple galleries that only display limited number of fields from your data sources, a Form Gallery will display all fields from that data source. We require **Form Galleries** to either display all the data to view or to edit the data. We also need them to add new records.

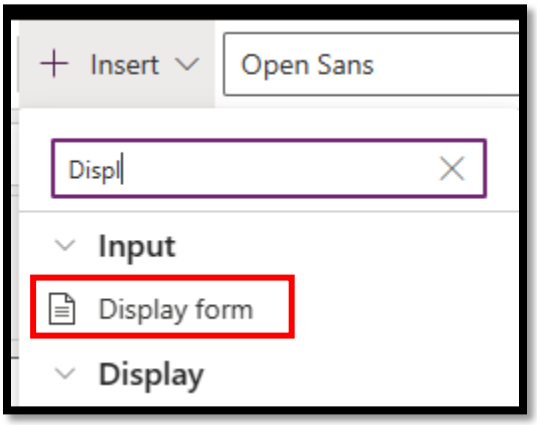
To start this process, you would typically create a brand-new screen (or create from Master Screen) then add either the Display or Edit form. Then you can make any cosmetic changes you require followed by connecting it to display the correct data.

1. Create a new screen off the **Master Screen** and give it a proper name.

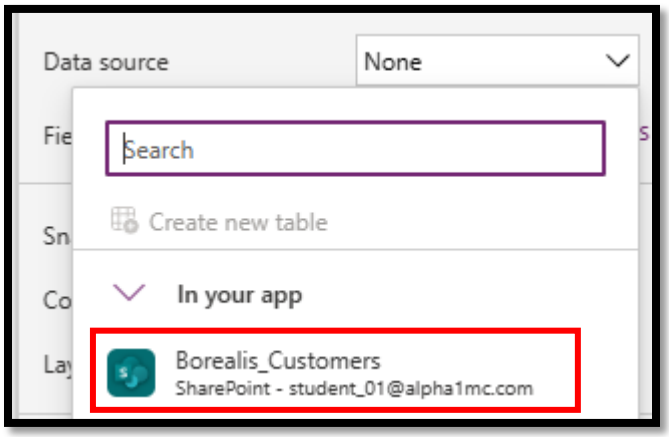




2. From the **Insert** menu > **Input** group, choose **Display Form** (we don't want users to edit our customers). Then Adjust the object to fit nicely on your screen.

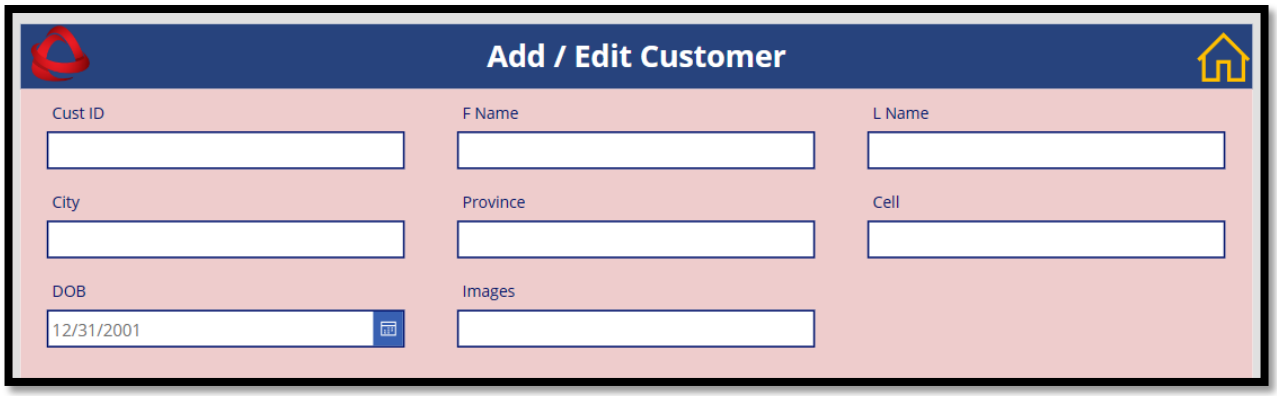


3. We now need to connect to a data source so from the **Properties Pane > Data Source**, select the **Borealis_Customers** list from the drop down. The reason why it appears here is because we already connected to the same source in the last screen.

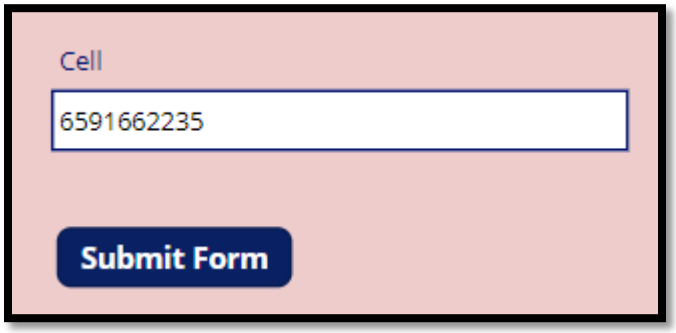


4. It will now display your fields in the form you see below.

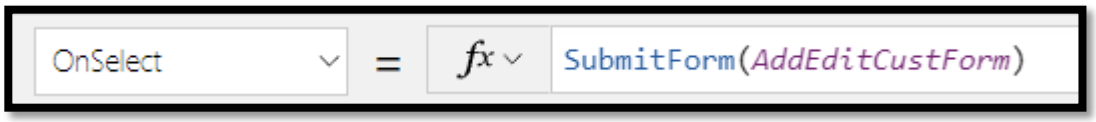
5. Repeat the last 5 steps (1 thru 5), but this time call the screen **Add/Edit Customer** and use the **Insert** Menu **> Input** group **> Edit Form** instead. It should look something like the screen shot below.



6. From the **Insert** menu > **Popular** group choose **Button** and rename it to something like **Submit Form**.



7. Now once the user is ready to submit the changes or the new record, they need to click the **Submit Form** button to push the data to the data source it came from. Add the following formula to the **OnSelect** property of **Submit Form** button you just created (you will have to change the name of the from in Tree View to match the name in the brackets).



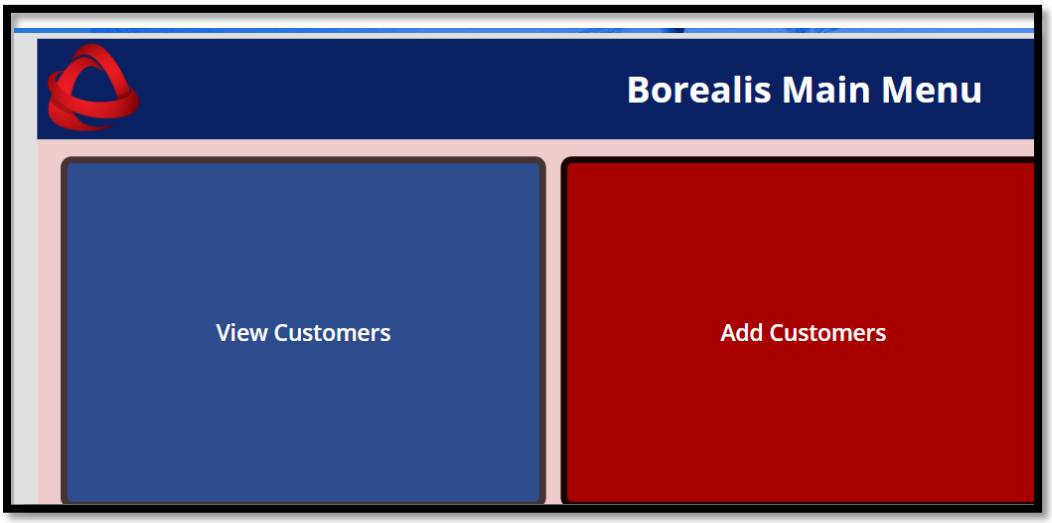
Adding Navigation Features

When developing the first app, most of the navigation was done for you because it built a simple 3 screen app with navigation buttons already linked to everything. Now because you created a blank canvas app you will now have to link everything manually. They are all simple, you will just need to decide what icons you want to use for your navigation. Also, you should start naming your screens or other objects with proper names (No spaces) to help facilitate easier function writing.

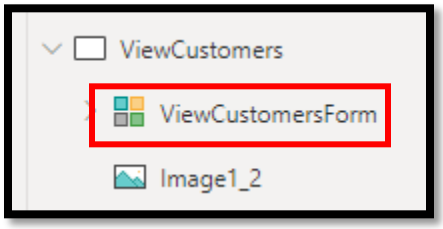
1. Now that we have the following screen: Home Screen, ViewCustomers, CustDetailScreen and CustAddEditScreen, we can now link them up with various navigation buttons for the user. From your **Tree View** select your **Home Screen**, then select the button you want to link to the **View Customers** screen. From the formula bar in the **OnSelect** property change it from **False** to read **Navigate(Nameofyourscreen).**

2. Now switch to the View Customers screen. This one already has a house icon on it, but it does not link anyplace. So, select the **House** Icon and adjust the formula bar to read – **Navigate(thenameofyourhomescreen)**.

3. If you now play your app, you should be able to return to the Home Screen and from the Home Screen click the button to view your customers.

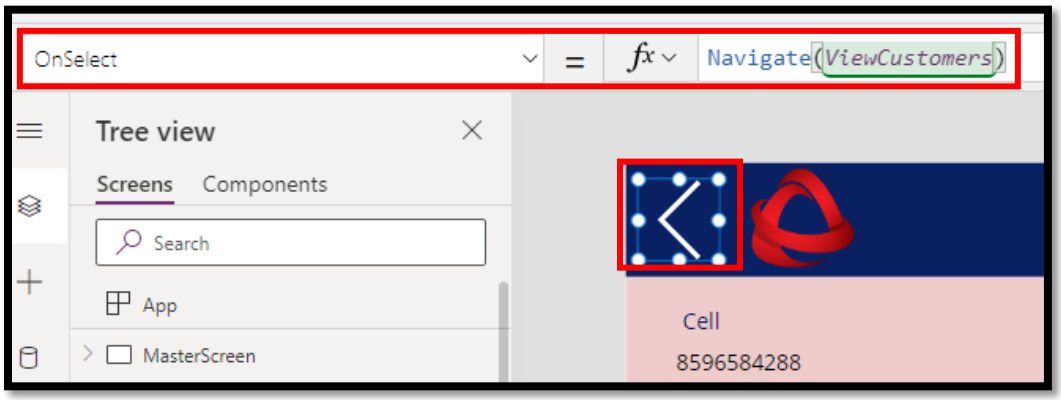


4. Now we need to connect the **View Customers** screen with the **Cust Detail Screen**. Remember the View Customer is just a list of all customers and a couple of their fields but when the user clicks the “>” button it should display all the fields about that customer. First, we will rename the **Gallery** from the **View Customer** screen, then on the **details screen** change the **Item** property of the **form** to read **NameOfCustomersForm.Selected**. If entered in correctly you should now see the data from the Customers Screen that was last selected.

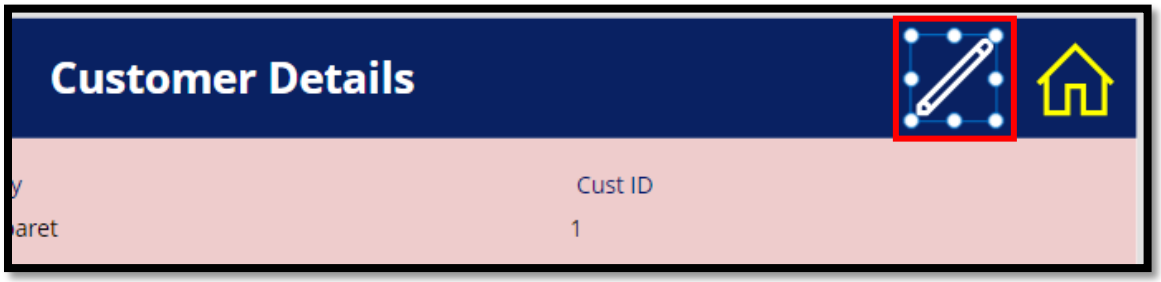


5. Now return to the **View Customers** screen and select the right arrow **“>”** and change the formula for the **OnSelect** function to read **Navigate(nameofcustdetailscreen)**. Then test your app.

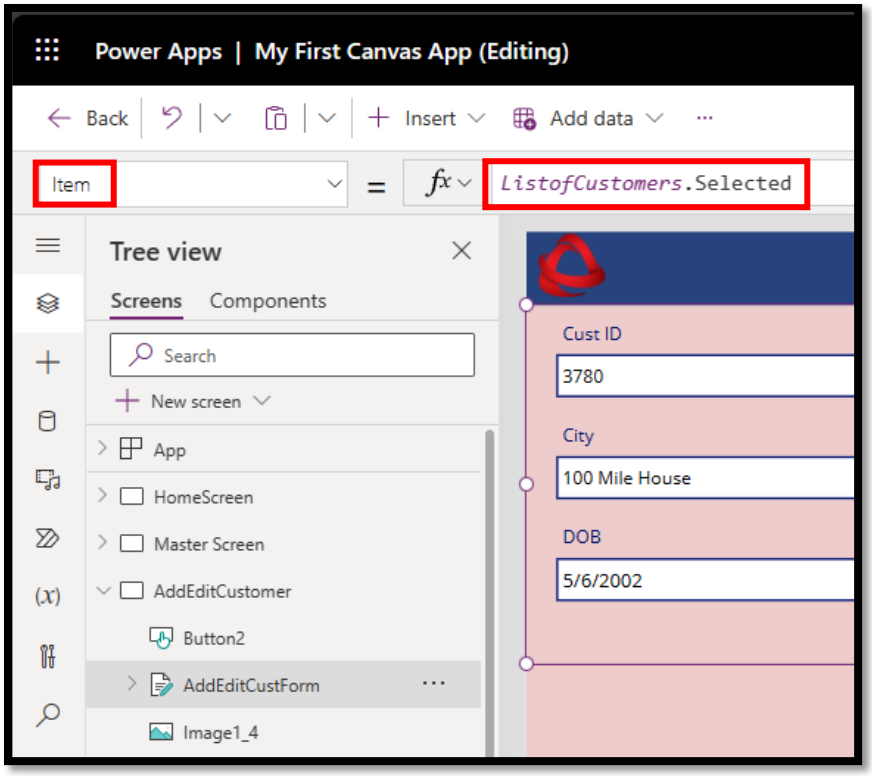
6. One last thing to do, on your **Customer Details** screen, you have no way of returning to the screen you just came from. So, choose your **Left** Arrow Icon from the **Insert** Menu > **Icon** group that navigates back to the **View Customer Screen**.



7. Now, from the Customer Detail Screen, add the Edit (Pencil) icon from the **Insert** Menu > **Icon** group so users can click it and get over to the Customer Edit Screen. Change the formula for the **OnSelect** property to **Navigate(nameofcustomtereditScreen).**



8. On the **Customer Add/Edit Screen**, select the **Form** object in the **Tree View** and change its Item property to read **NameofViewCustomersForm.Selected** *(The Form from the View Customers Screen)*. If it is linked properly, you will automatically see data populate in your canvas.

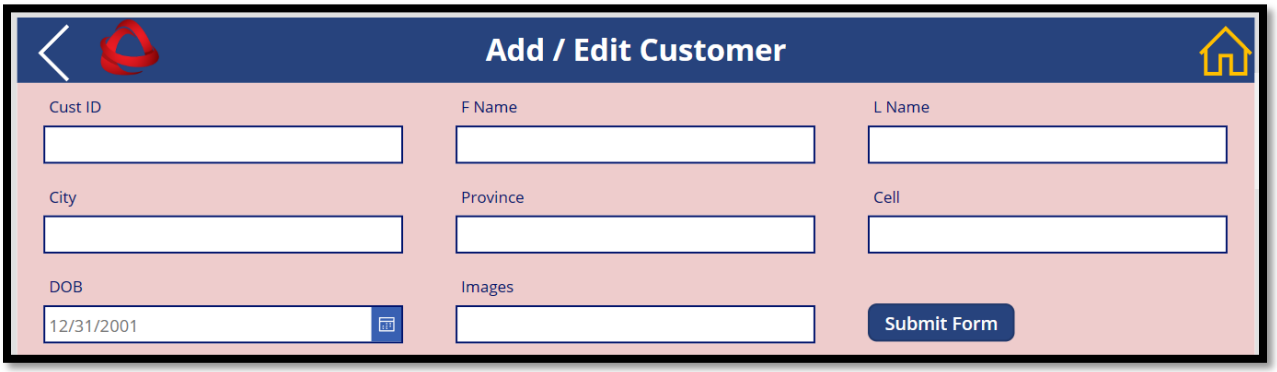


9. Last thing we need to do is repeat step #6 and create another left arrow icon "<" on this **Add/Edit** screen so the user can go back a screen.

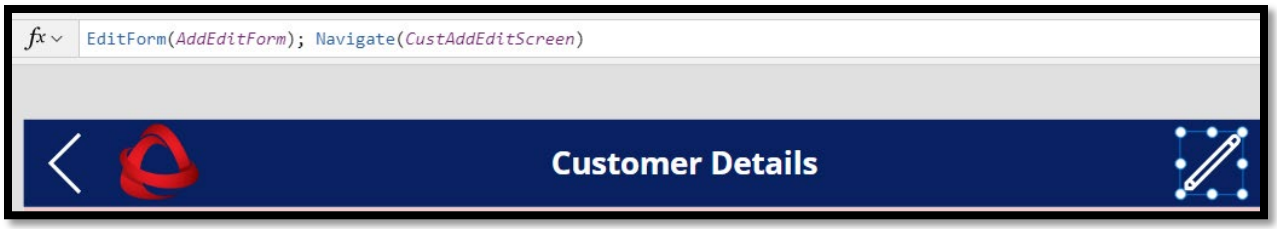
If you look at our home screen button options, we have a button to link to Add Customer. Do we need to create another Add/Edit screen to link to? Or can we somehow use the same one we just did to save us some work? We can just use the same one and fine tune the functions to do it for us.

1. Start with your Home Screen and select the button that says **Add Customers**. In the formula bar for the **OnSelect** property change it to read **NewForm.(nameofform); Navigate(NameofAddEditCustomerform).**

2. When you play your app from the **Home Screen** and click **Add Customer**, it should take you to the same **Add/Edit Customer** form but this time there should be no data in there as you are about to add a new record.



3. You will notice if you go back to the Home Screen, click View Customer, Customer Details then Edit Customer (Pencil Icon) that the screen is still in Add mode. This is because we are sharing the same screen and retains its last action unless told otherwise. So from the **Customer Details** screen, change the **OnSelect** function for the Pencil to **EditForm(NameofForm); Navigate(Nameofeditscreen)**.



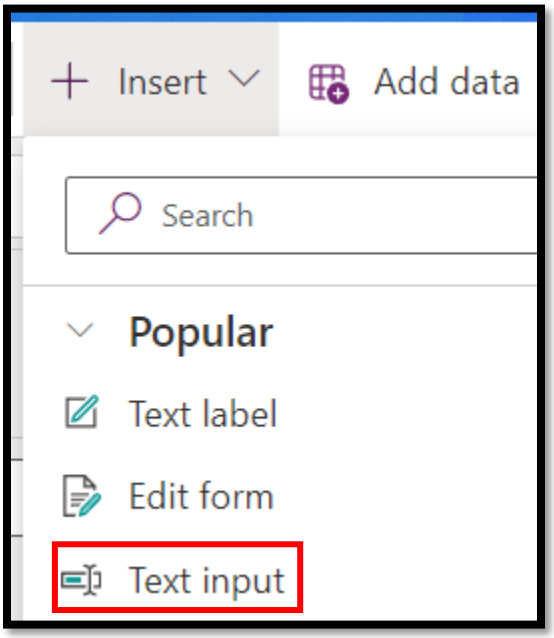
4. There is one more thing to fix. We set the back icon "<" on the Add/Edit customer screen to go back to the Detail screen. However, if you come to this screen via the Home Screen it also will take you back to the Detail screen. To fix this change the **OnSelect** formula on the Add/Edit screen to the following **Back().**

You should now be able to navigate around all your screens we have made so far.

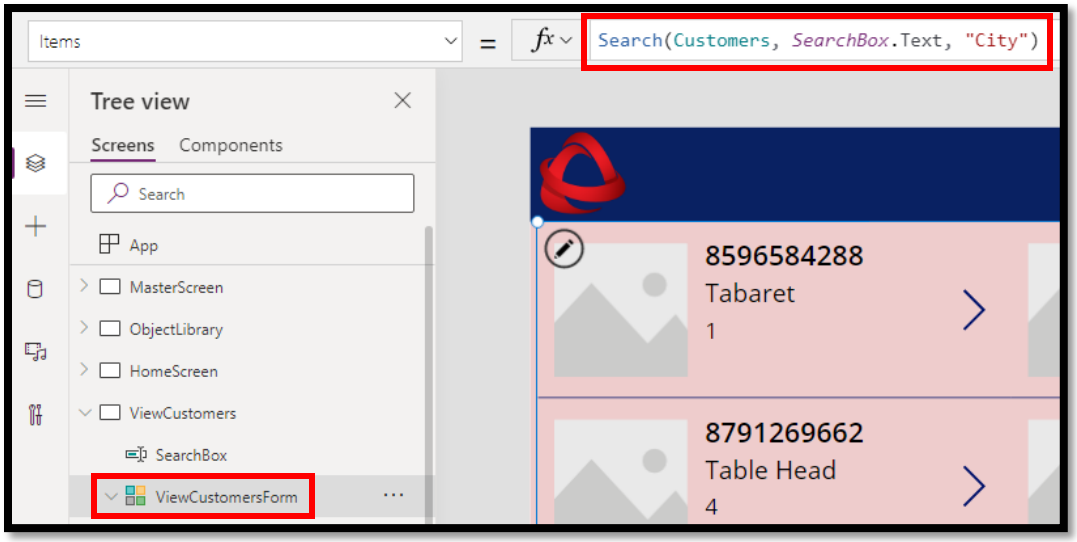
Search & Filter Functions

At some point you are going to need the ability to Search or Filter your customers. In the previous chapter when you made the automatic three screen app, the search field was already there. Well now we must manually create that and the formula behind that for it to actually search. It goes without saying, you will be putting the search box on the screen that lists your entire table of records (customers) so your users can narrow the results. You just need to figure out what columns you want the search to look in.

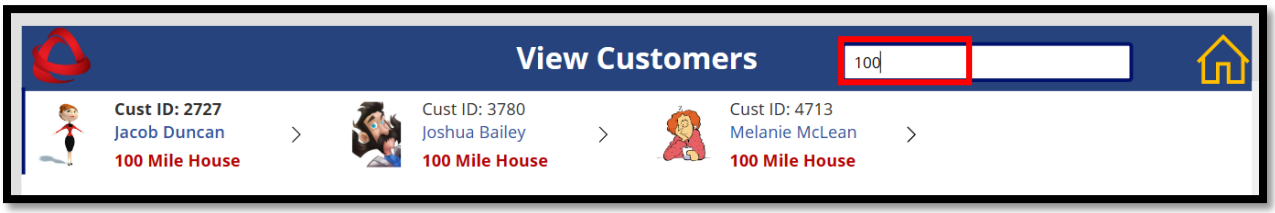
1. Return to the screen that displays all your customers. In our case it's the **View Customers** screen. From the **Insert** menu > **Popular** group choose **Text Input** and place it somewhere in the header.



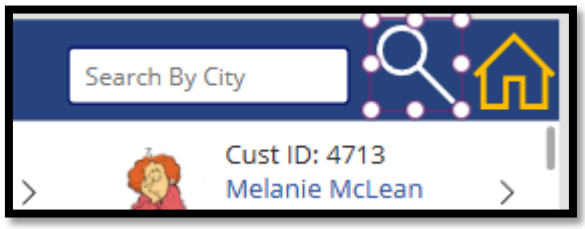
2. Now from the **Tree View**, select the **Form** object and change its **Items** property to **Search(NameofTable, NameofSearchBox.Text, "Nameoffieldtosearch").** Note – if you wanted the search to look in multiple fields, just continue to list the fields in quotes after the first field we listed above.



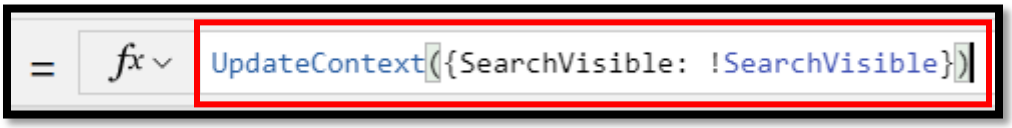
3. If you now play your screen and search your customer list, you should see it narrow down to all those fields with that search word.



4. Now you may not want the search box to appear unless the user clicks the search icon, so let's figure out how to get this done. Return to your canvas and from the **Insert** menu > **Icon** group select the **Search** icon and insert on the top title bar.



5. Although beyond the scope of this course, we must introduce inline variables (more in depth on course 2 for variables). We need to use an inline variable to hold the "state" of an object (true or false) and make it the opposite state when the search button is clicked. Select the **Search icon** and change its **OnSelect** property to read **UpdateContext({SearchVisible: !SearchVisible})**



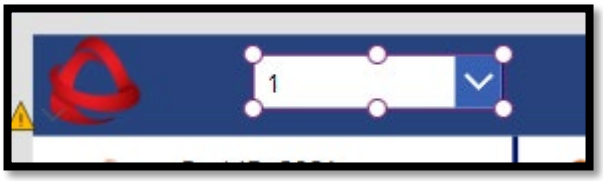
6. Now select the **Search Box** itself and change its **Visible** property to read **SearchVisible**. It should automatically hide the Search Box.

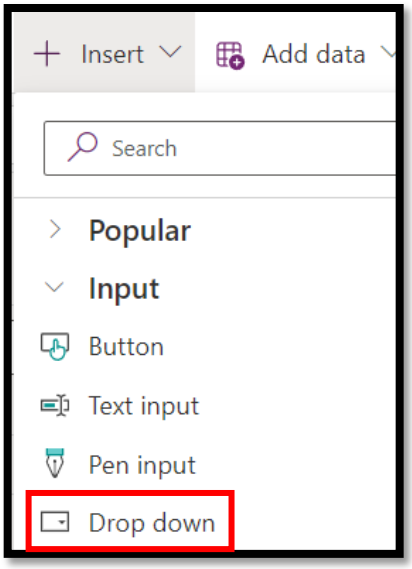
7. If you play your screen the **Search Icon** now becomes a toggle for turning on or off the **Search Box** you created.

Sometimes however using search just does not cut it. You are expecting your users to know the words to search for and know how to spell! Your next best option then is to create filters. With a filter you can eliminate the guess work for your user, and they can just pick from a dropdown list what they want. This will require a new object (drop down list) and a new formula, but the results will be worth it.

One thing to remember when using filters. Remember in the beginning of this course we told you that Power Apps will limit your records to 500 (2000 if you change it in the settings)? Well, the same issue will be applied here. The filter will only show up to a default of 500 items (or 2000) so if you have more than that it will not list them all!

1. Return to the canvas (screen) that has your search feature installed and create yourself a **Filter** object from **Insert** menu **>** **Input** group **> Drop Down** and position it somewhere on the header. Give the Filter object a name – **FilterBox**.





2. With the Filter drop down still selected, change the **Items** property formula to read the following: **Sort(Distinct(Customers,City),Value, Ascending).**

```
Sort(Distinct(Borealis_Customers_1,City),Value,SortOrder.Ascending)
```

This will result in distinct values from the **City** column in the **Borealis_Customers** list. You can change City to any field from the Borealis_Customers list that you want the user to use as a filter.

Also if you do not want to pull the list from your data source you can just list them in the Items formula like this: **["Option1","Option2","Option3", etc]**

3. Now that we have the list, we need the selection from the list to Filter the form below. If you select the form from the Tree View, you will notice you already have the Search formula in there and that is ok. Power Apps lets you combine your functions to get the job done. Also, we need to be able to "un" filter our options as well. Follow the steps below to add in and adjust the functions for the objects. Your instructor will explain in detail how each function works.

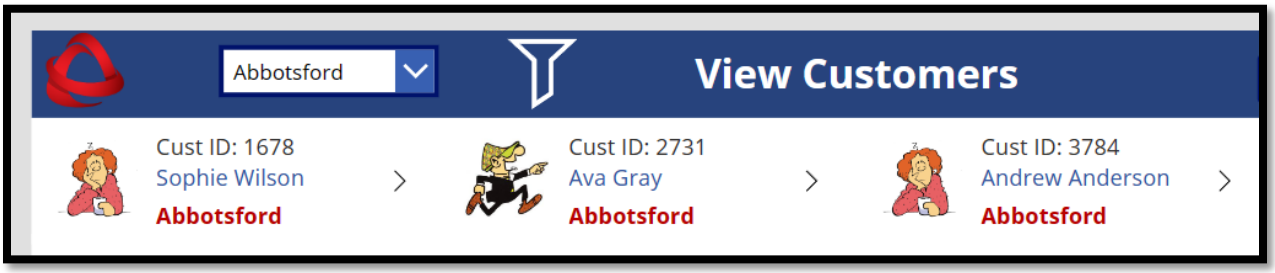
   a. Select the **Form** object and adjust the current Search formula as follows: **Filter(Search(Customers, SearchBox.Text, "City"),If(BtnStatus = true, true, City = FilterBox.Selected.Value))**

```
Filter(Search(Borealis_Customers_1,SearchBox.Text,"field_3"),If(BtnStatus = true, true, City = FilterBox.Selected.Value))
```

b. Select the Filter Box drop down and change the **OnSelect** formula to read as follows: **UpdateContext({BtnStatus: false}).**

c. Insert a new **Filter** Icon object and place it someplace on the title bar. Then adjust its **OnSelect** formula to as follows: **UpdateContext({BtnStatus: true})**.

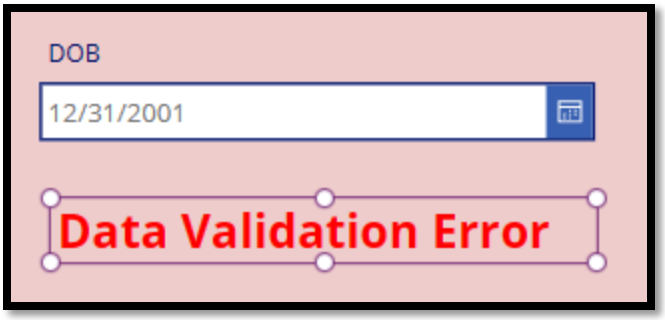
4. Now when you play your screen, you can select your city from the drop down, then click the Filter button to un-filter and put the list back to full.



Data Validation

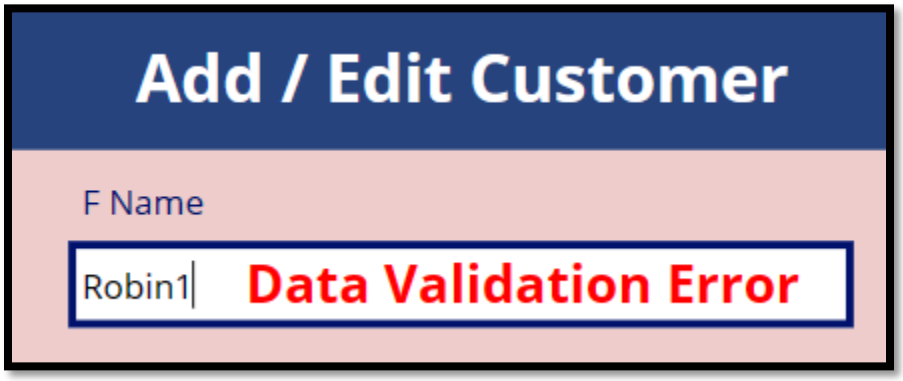
Currently the default behavior allows users to enter any data in any fields and it will be accepted. With Data Validation we will verify the data first to make sure it is allowed. Now if your data source is SQL or some database that currently has validation set then it will not allow it to be saved anyways. However, you still need a way to communicate to your users that they did something wrong, as you will not get a message back from the database.

1. Return to the **Add/Edit** screen and from the **Insert** menu > **Popular** group select **Text Label**. Format the label accordingly.



2. Now we need to change the **Text** property formula, of this new label, to look at the F Name field to ensure the user does not enter any digits in. We would accomplish this by using an IF statement combined with the IsMatch and Match functions. Adjust the formula for the Text property as follows: **IF(IsMatch(DataVardValue4.Text, Match.Digit, MatchOptions.Contains),"Data Validation Error",""**).

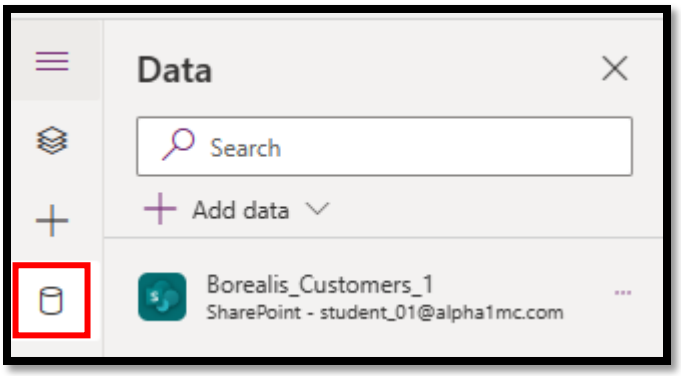
3. When you go play your screen and try to enter digits in the F Name field you should get the label error message.



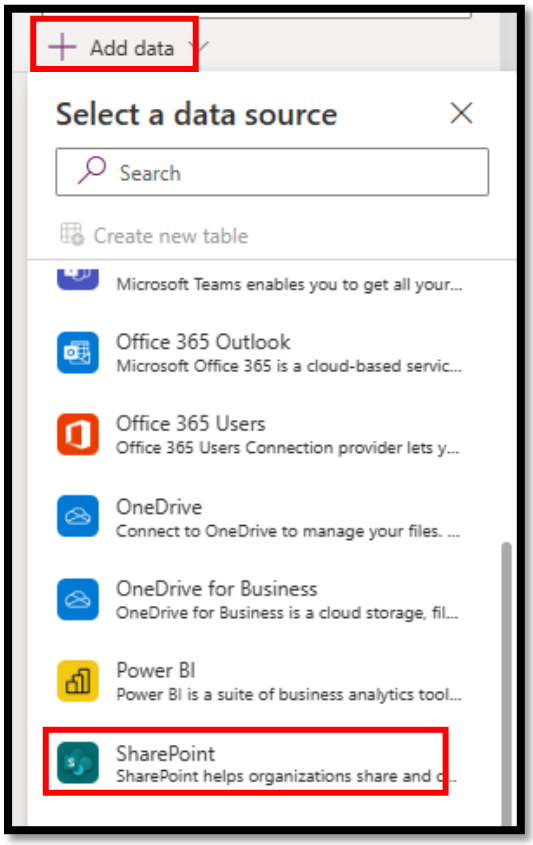
Multiple Data Sources

In all our examples so far, we have connected to one file, one table. Reality is that will not be the case most of the time. You will have one file (or one database), but you will have to connect to multiple tables within that file. If you are using an Excel document, the tables must be designated a table and given a unique name.

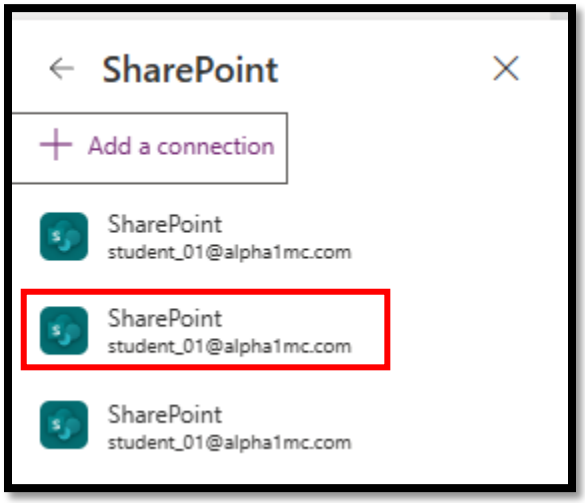
1. From the **Tools** menu on the left side of your screen click the **Data** button.

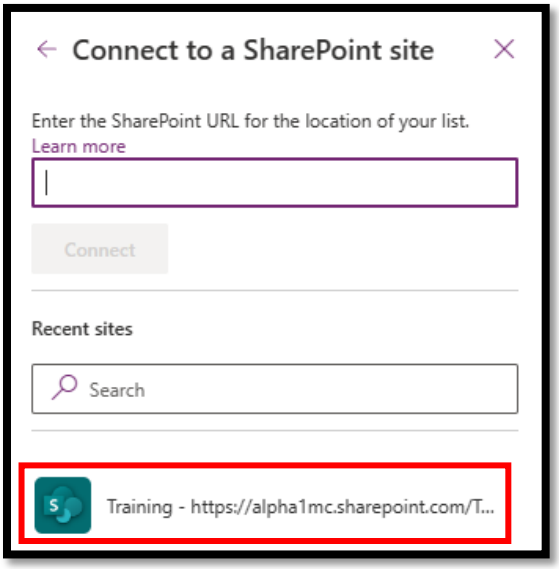


2. Click the **Add Data button** > **Connectors** > **SharePoint**.

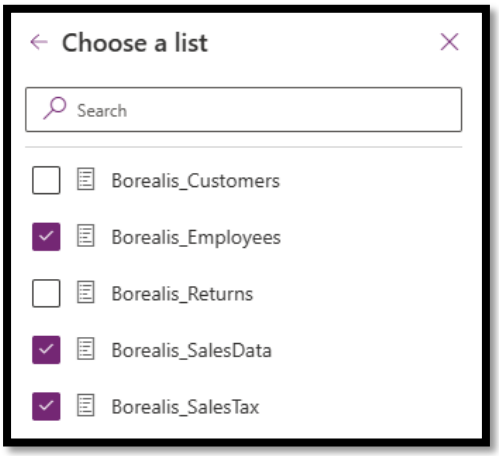


3. Choose your SharePoint connection followed by your SharePoint site.

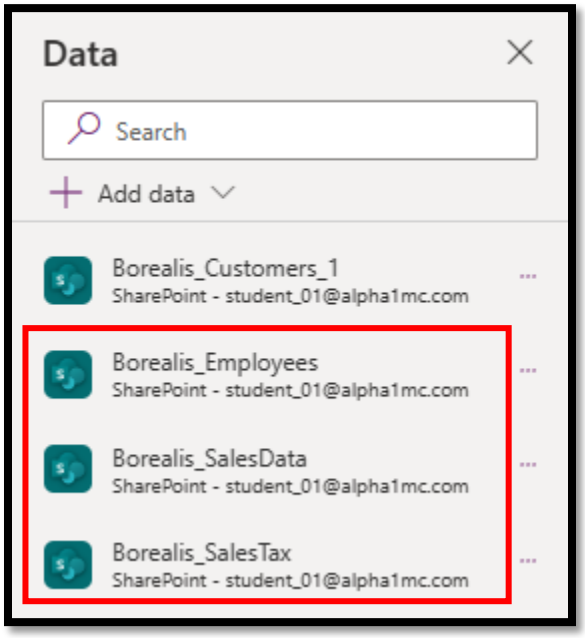




4. Select one or more SharePoint lists you want to be able to connect to. Click **Connect** at the bottom once complete.



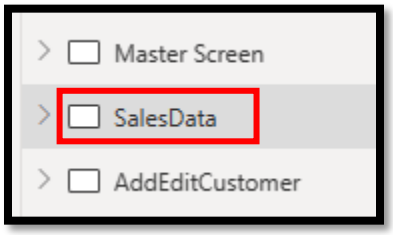
5. You should now have a list of all your SharePoint Lists that you are connected to. They are ready to be connected to future data forms that you insert on your screens.



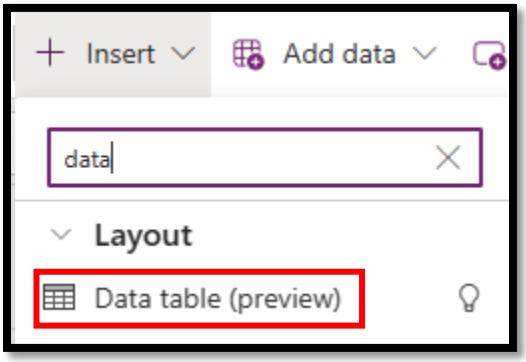
Using Data Tables

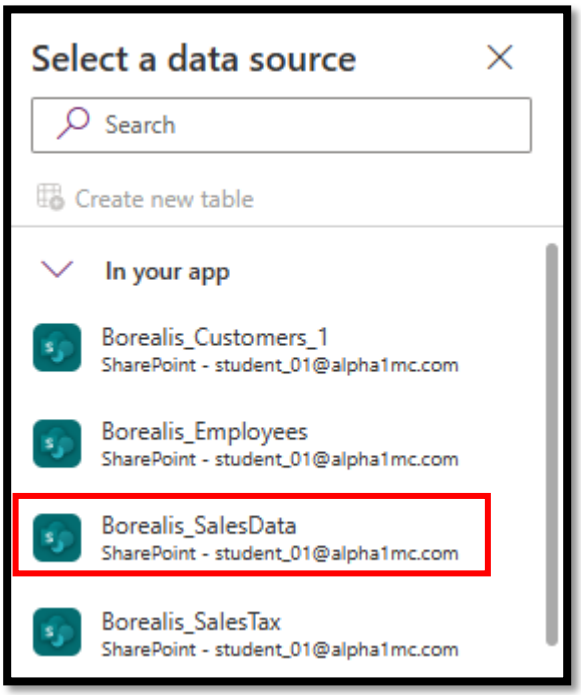
Data Tables are exactly what they imply – simply a table of data. It will display the source data you tell it to connect to in a table. This is not meant to be edited, but just viewed. Keeping in mind your canvas is either as wide as a phone or pad you will not necessarily want to view 50 columns of data but rather enough that it fills the screen.

1. Return to your **Tree View** and create another screen based on your Master Screen and call it **Sales Data**.

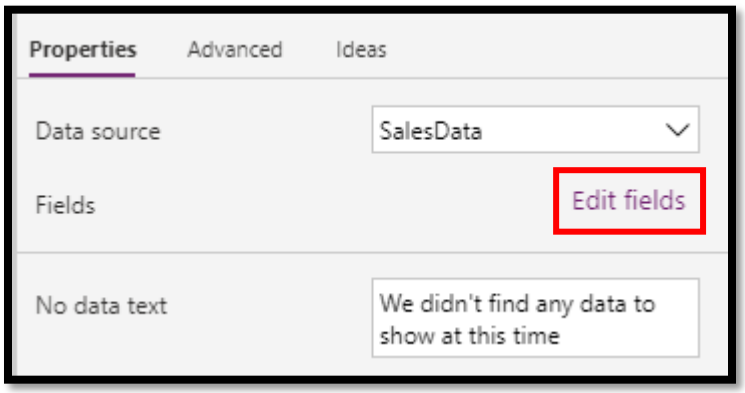


2. From the **Insert** menu > **Layout** group, choose **Data Table** followed by assigning the data from the SharePoint list you want to display. Adjust the size of the data table as well.





3. If you need to adjust the fields that should appear in the table, click the **Edit Fields** link just below the **Data Source** drop down in the properties pane. Delete or add whatever fields you require.

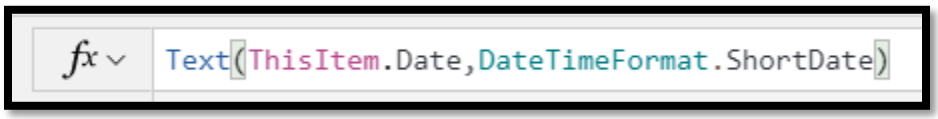


4. You should now see your data listed in a table. *Note: if it says no data to display, you will have to go to your Data Source on the left menu and click the … beside your data source and choose refresh. Also remember your data source cannot be open for Power Apps to read the data*.

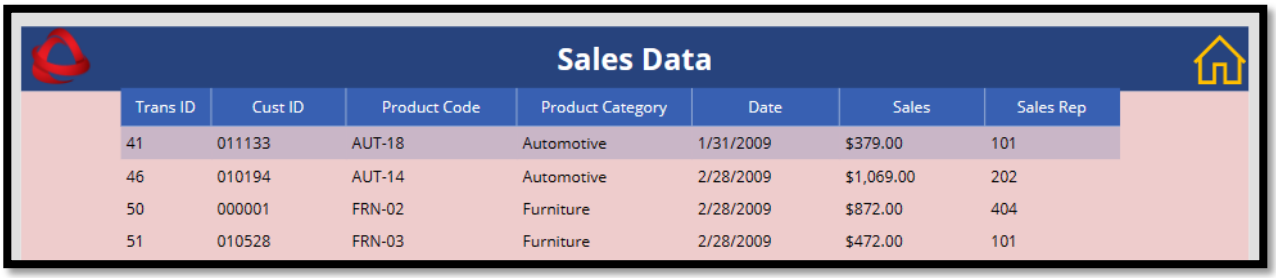
Trans ID	Cust ID	Product Code	Product Category	Date	Sales	Sales Rep
41	011133	AUT-18	Automotive	Saturday, January 31, 2009 12:00:00 AM	379	101
46	010194	AUT-14	Automotive	Saturday, February 28, 2009 12:00:00 AM	1069	202
50	000001	FRN-02	Furniture	Saturday, February 28, 2009 12:00:00 AM	872	404
51	010528	FRN-03	Furniture	Saturday, February 28, 2009 12:00:00 AM	472	101

5. Now we have some formatting issues regarding our data. Let us fix the Sales column which should be currency, but Power Apps displays as text. Select the **Sales** column and change the formula for the **Text** property to: **Text(Value(ThisItem.Sales),"[$-en-US]$#,##0.00")**.

6. If you look at the **Date** column, it includes long date and time. So let us change that to just short date by adjusting the **Text** property for the **Date** column to be: **Text(ThisItem.Date,DateTimeFormat.ShortDate)**.



7. Your Data Table should now have the two columns formatted properly.

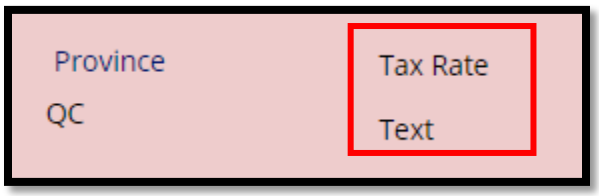


Trans ID	Cust ID	Product Code	Product Category	Date	Sales	Sales Rep
41	011133	AUT-18	Automotive	1/31/2009	$379.00	101
46	010194	AUT-14	Automotive	2/28/2009	$1,069.00	202
50	000001	FRN-02	Furniture	2/28/2009	$872.00	404
51	010528	FRN-03	Furniture	2/28/2009	$472.00	101

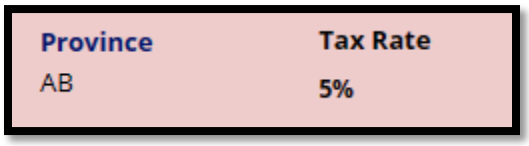
Lookup Function

The Lookup function does the exact same job as it does in Excel. It will look up a value in a different table and bring back a corresponding value in the column you specify. Where the syntax in Excel is Lookup – What, Where, Bring back what?, in Power Apps it is Lookup – Where, What, Bring back what column?.

1. Return to the screen where you want the lookup function to be performed from. In our case it is the **Customer Details** screen. We are going to look up the province for this customer in the data source SharePoint List Sales Tax and return the provincial tax rate. Create two **Text Labels** as one will be the name of the field (Tax Rate) and the other will contain the lookup function shortly.



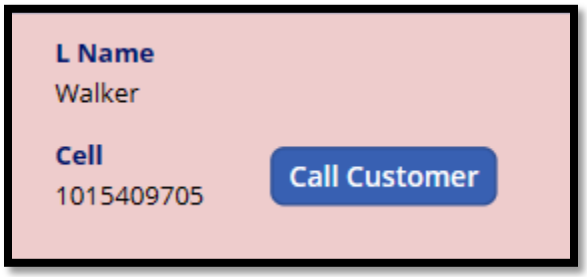
2. Select the 2nd label you created and adjust its Text property formula to ready the following: **Text(LookUp(Borealis_SalesTax,Prov = DataCardValue14.Text,'SalesTax')*100,"#0%")**



Adding Call Functionality

Since Power Apps is intended to run on your phone or pad, it makes since to have the ability to use the Call feature so your users can Call a telephone number that is displayed from your data. It is quite simple to insert and add functionality to your app. Just a note, if you click the button we are about to create while using the app on your computer, it will try and use Microsoft Skype.

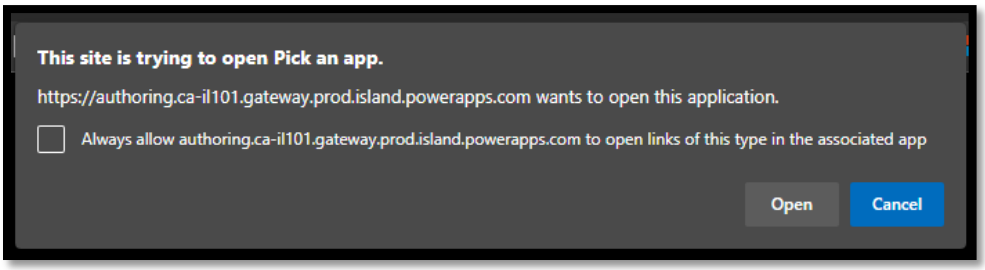
1. Return to a screen of yours that has a telephone number for one of your records. Insert a **Button** someplace on this screen like the one below.



2. With the Button selected, change the formula for the **OnSelect** property to the following: **Launch(Concatenate("tel: ",DataCardValue1_1.Text))**.

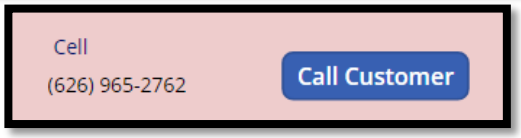
```
Launch(Concatenate("tel: ",DataCardValue1_1.Text))
```

3. If you go play and test this screen, it will try and call, but you will be getting a warning message like below.



4. Return to the Customer Details screen and let us do one last thing and actually change the format of the Cell field so it actually looks like a phone number. Select the Cell field and change the formula for the Text property to: **Text(Value(Parent.Default),"[$-de-DE](000) 000-0000").**

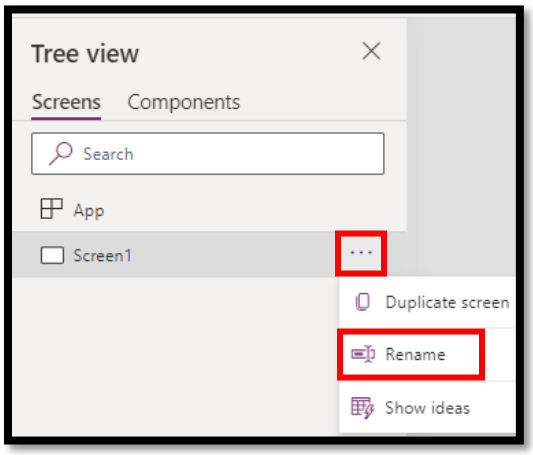
```
Text(Value(Parent.Default),"[$-de-DE](000) 000-0000")
```



Lesson 3 Exercise

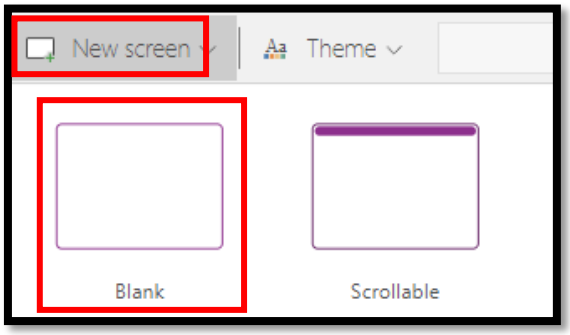
1. Starting from your Power Apps main environment screen create a new with a new **Blank Canvas App** with a Tablet format and call it Exercise 3 App.

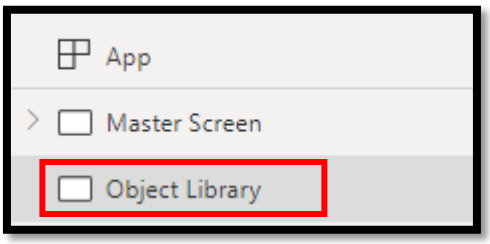
2. From **Tree View**, rename the **Screen1** object to **Master Screen**.



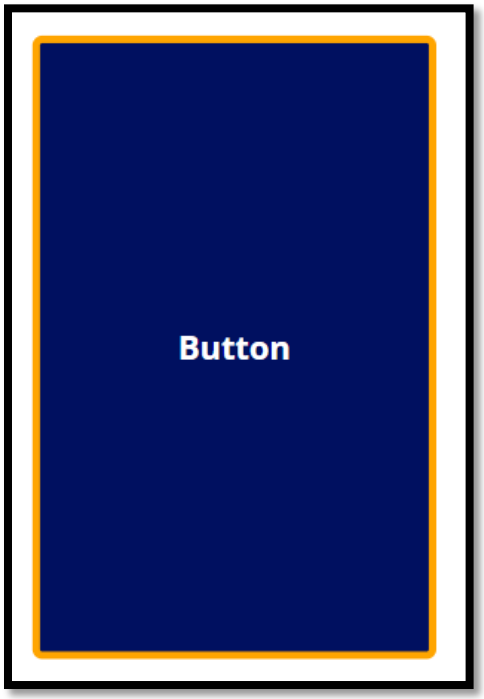
3. Create a header using a rectangle (**Insert > Shapes**) a house icon (**Insert > Icons**), and a Screen Title (**Insert > Text Label**) and colour both the rectangle and background as seen in the final screen shot. Then insert the Police Car image (**Insert > Media**) as well from the data files you were given.

4. Create another new screen with a **Blank** Layout and rename it **Object Library**.

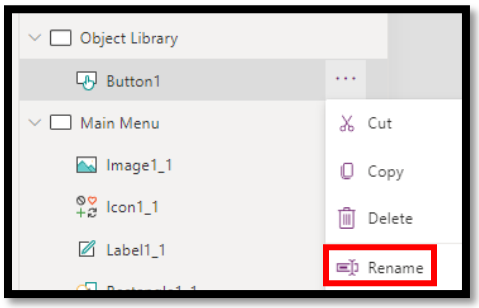


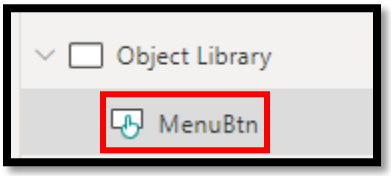


5. In the **Object Library**, create a button that is longer than it is wide, colour it, and put a border on it (Insert > Popular).

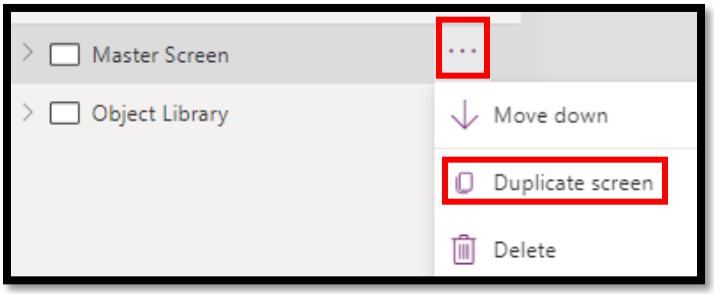


6. From within the **Tree view**, rename this button to **MenuBtn** (no spaces).

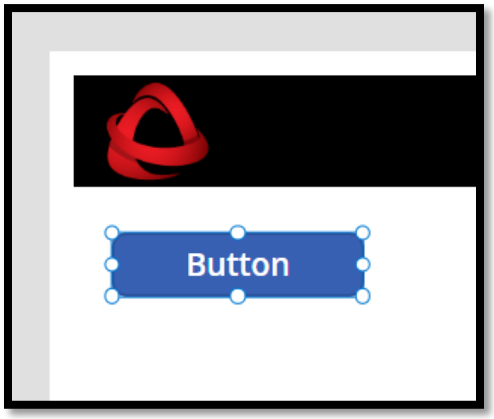




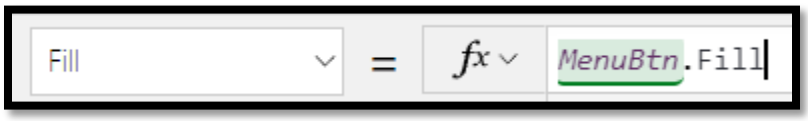
7. Duplicate the **Master Screen** and name the new screen **Main Menu**.



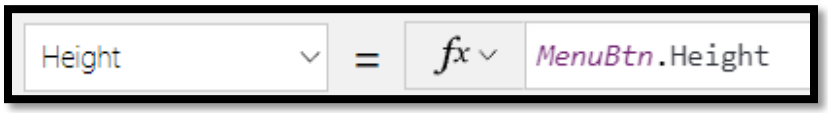
8. Insert a **Button** into this screen.

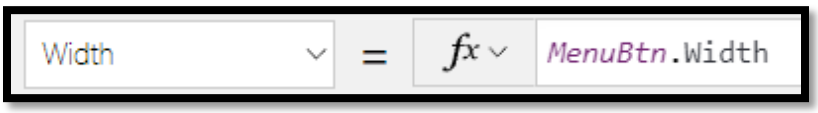


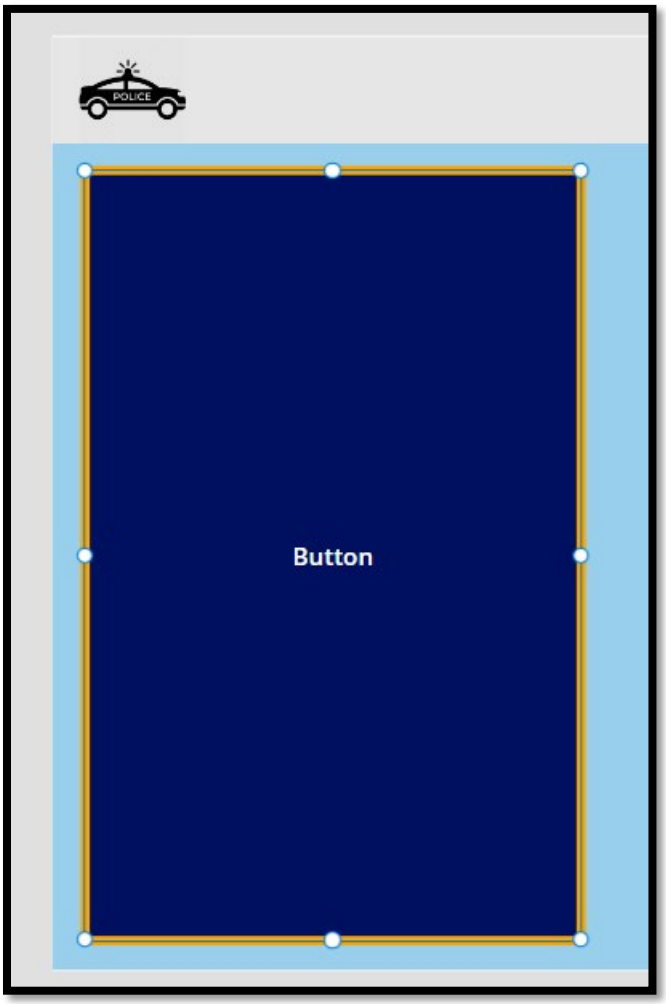
9. With this button active, using the drop-down properties list, choose the **Fill** property and use the formula: **MenuBtn.Fill**. For the **BorderColor** formula use: **MenuBtn.BorderColor** and for the **BorderThickness** property use the formula: **MenuBtn.BorderThickness**.



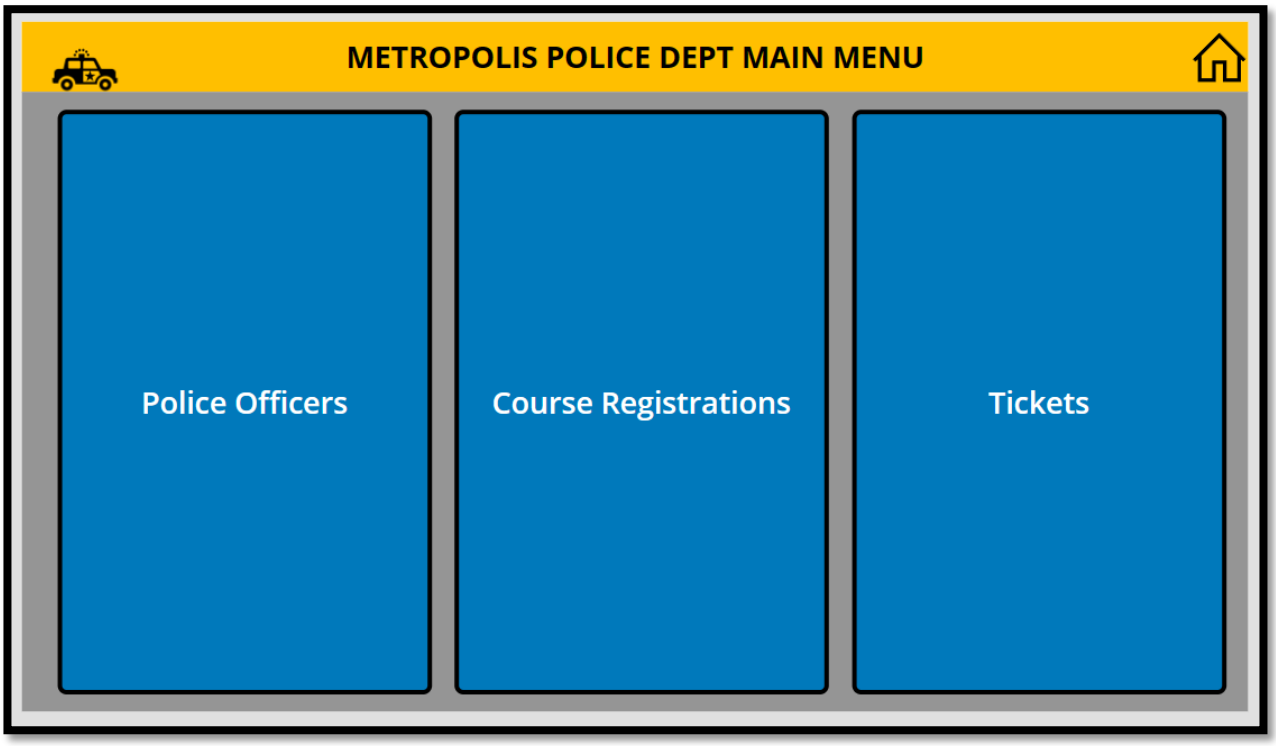
10. Now those properties where just colours and borders. Now let's link up the physical size changing the **Height** and **Width** properties like below. Also see if you can figure out the property to adjust the text properties.



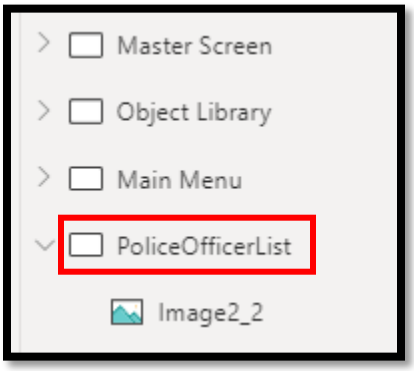




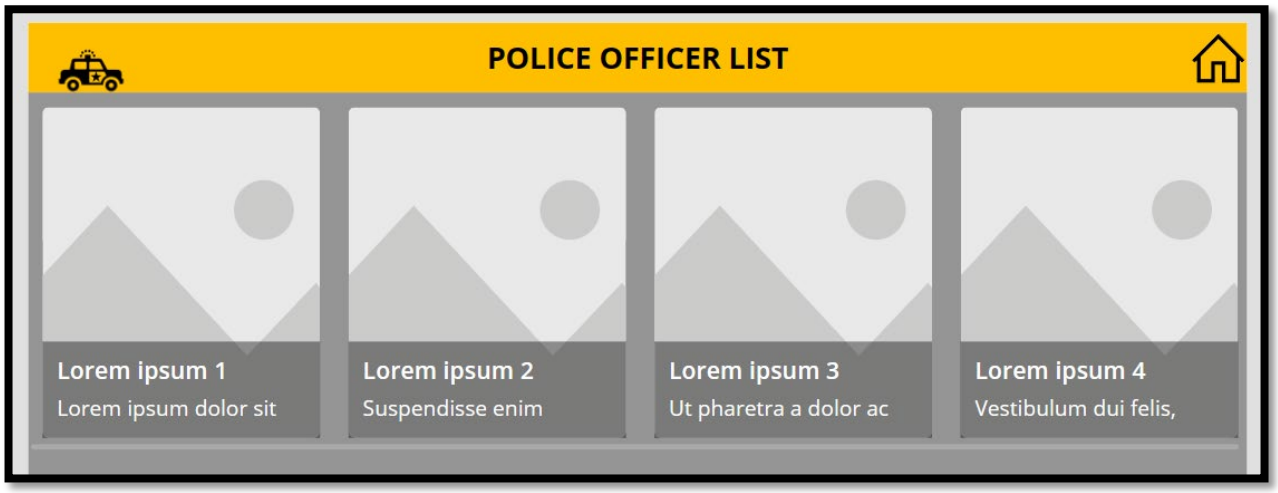
11. Duplicate this button 2 more times and adjust to fit. Change the title accordingly and change the buttons to read: **Police Officers**; **Course Registrations**; **Tickets**.

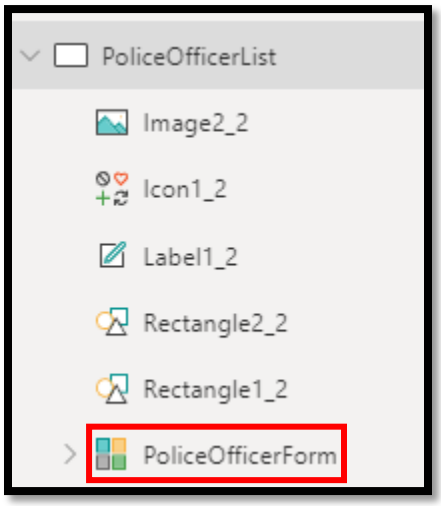


12. Duplicate the Master Screen again and call the new one **Police Officer List**. Then change the title of the page to **Police Officer List** as well.

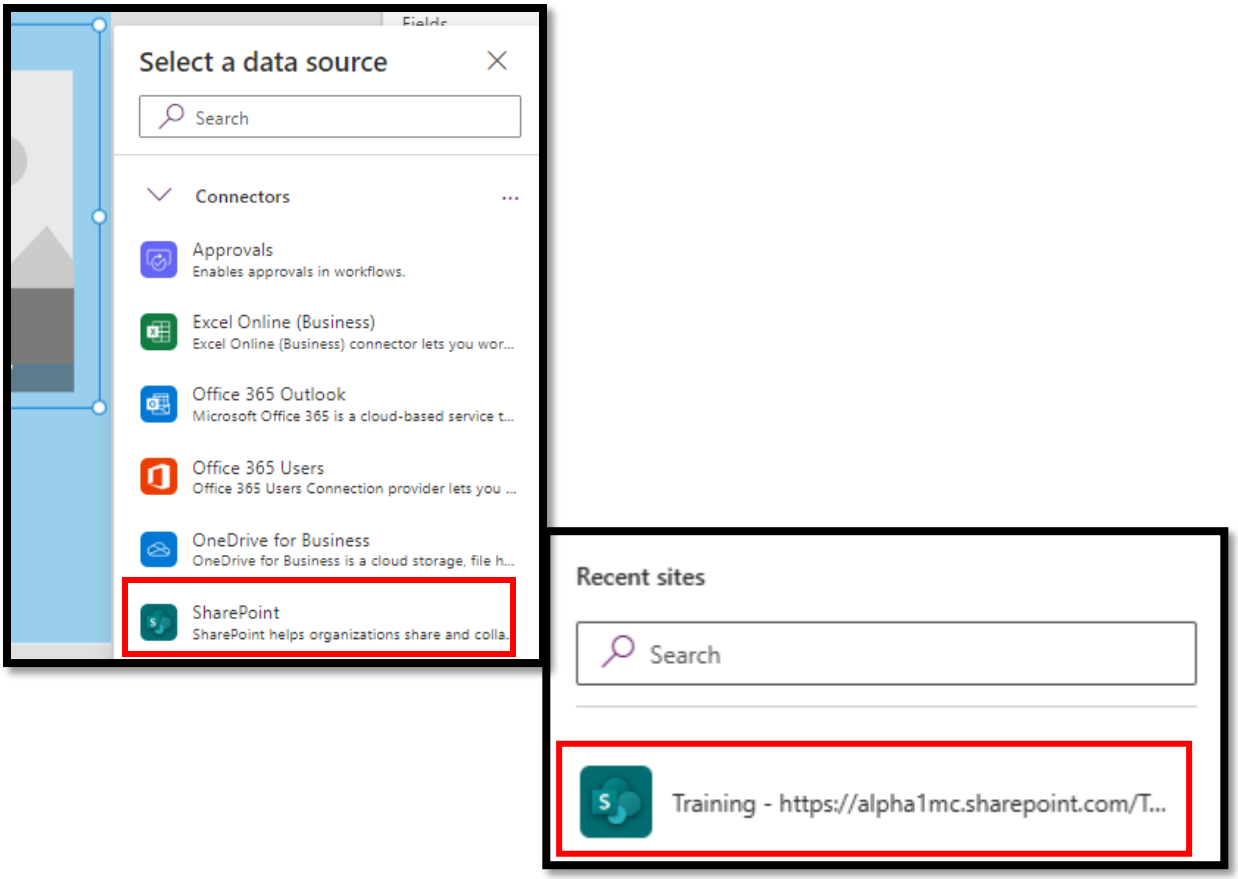


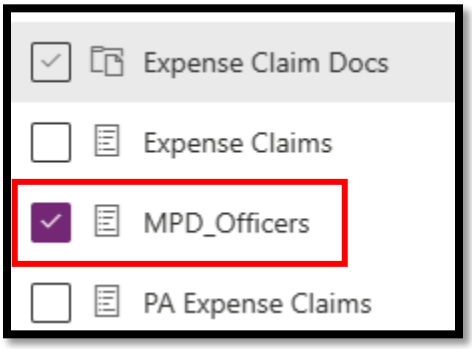
13. From this new screen, **Insert > Display** and choose **Horizontal Gallery**. Then from the **Properties** pane, **Layout** property choose **Title and subtitle on overlay**. Now adjust the gallery so you see 4 image placeholders as seen in the screenshot below. Also select and rename the form **PoliceOfficerForm**.



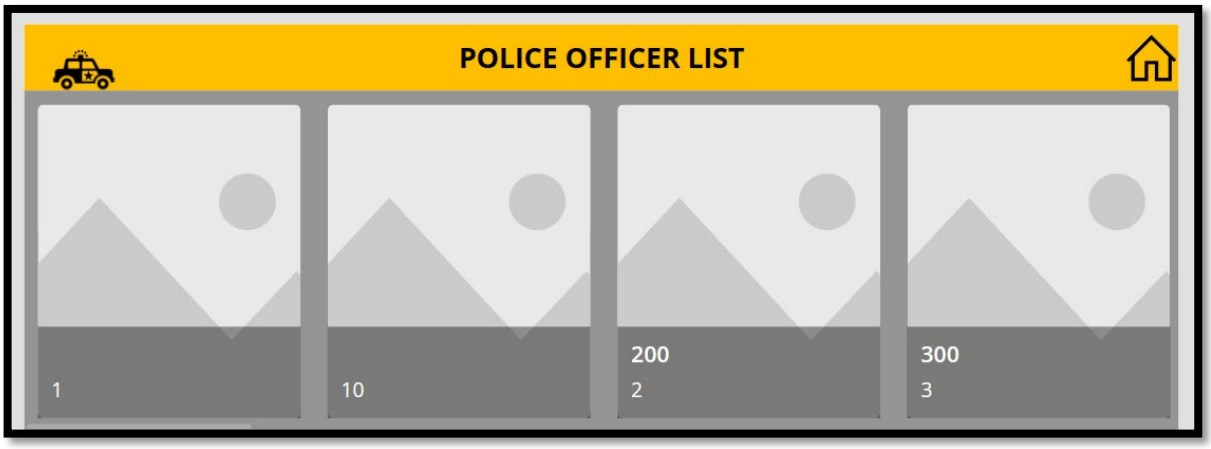


14. We now need to connect this form to our data. When you select the gallery within the screen you will notice the pane **Select a data source** appears *(if it does not you have to select the data sources from the Properties Pane)*. From this pane choose **Connectors** and locate the **SharePoint**. Locate the **Alpha1MC Training Site** and select it followed by selecting the **MPD_Officers** list and clicking the **Connect** button on the bottom.





15. You can see it is pointing to the correct data; however, we need to modify the fields that are displayed and change the background rectangle color so the images blend in better.



First select the grey background rectangle and either change the color to white or delete background. Select the **Image** placeholder in the form and change the **Image** property to read **ThisItem.Photo**.

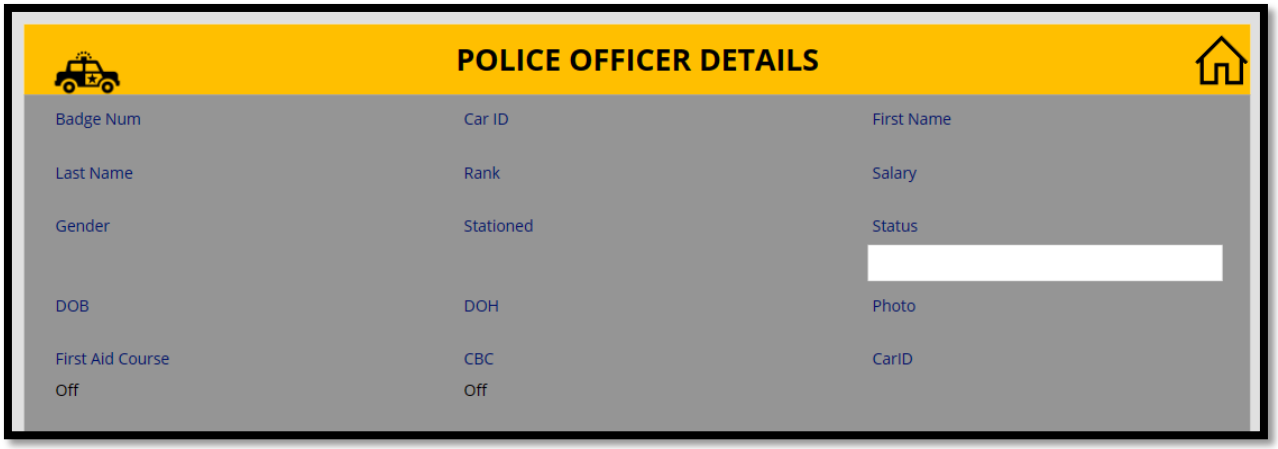
16. Currently the layout displays the badge number of each officer. We want it to read their first and last name along with their rank. To do this, select the current badge field in the first placeholder and in the **Text** property formula bar change it to read: **Concatenate("Badge #",ThisItem.'Badge Num'**. Adjust the current Car ID label and add a new label and modify their Text field properties to the screen shots below.

```
fx ⌄ Concatenate(ThisItem.'First Name', " ",ThisItem.'Last Name')
```

```
fx ⌄ ThisItem.Rank
```

17. Now we need to create the **Police Officer Detail Screen**. Duplicate the **Master Screen** again name it **Police Officer Details**.

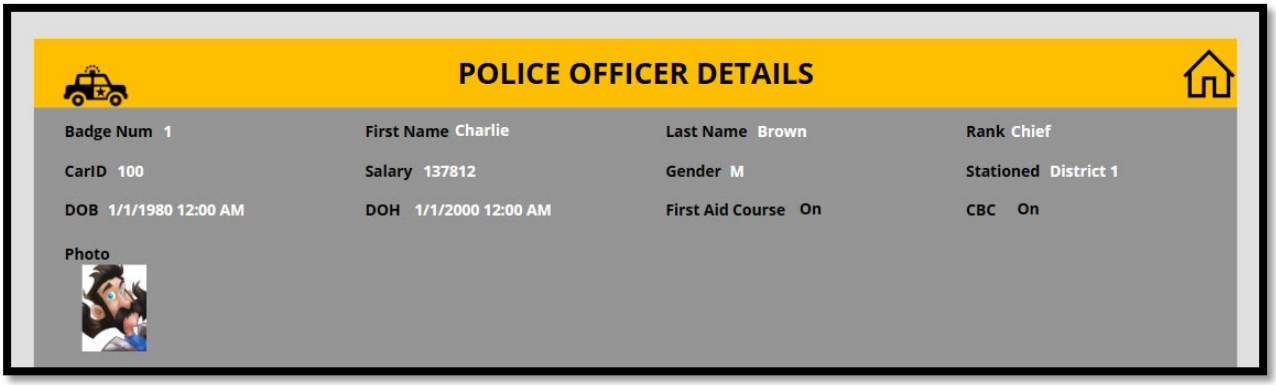
18. Since this is just the Officer Details screen, we just want to be able to view the details not edit them. Choose **Insert > Input > Display Form a**nd connect to the **MPD_Officers** data source via the **Data Source** drop down in the properties pane**.**



19. We now need to link this detailed screen to the record that the user selected from the **Police Officer List** screen. Select the form and in the **Item** property formula bar type: **PoliceOfficerForm.Selected**.

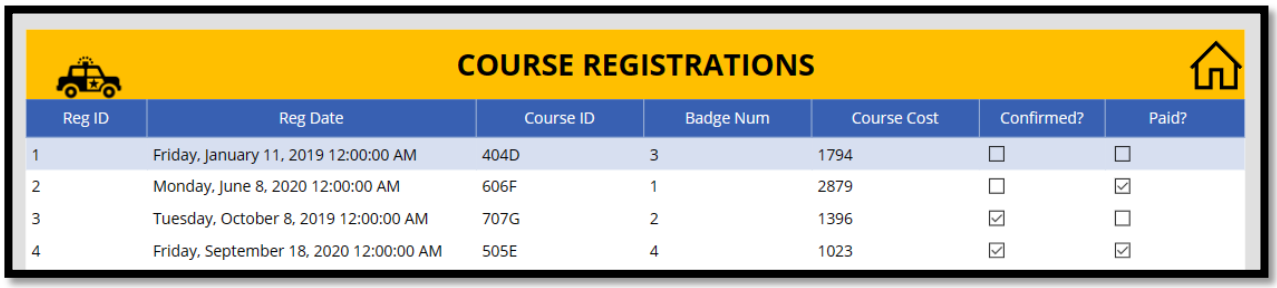
20. Now return to the Police Officer List screen, select the **Overlay#** object and change its **OnSelect** property formula to read: **Navigate(PoliceOfficerDetails)**.

21. Using the Properties pane and adjust each **Label** and **Input** field so it looks something like below.

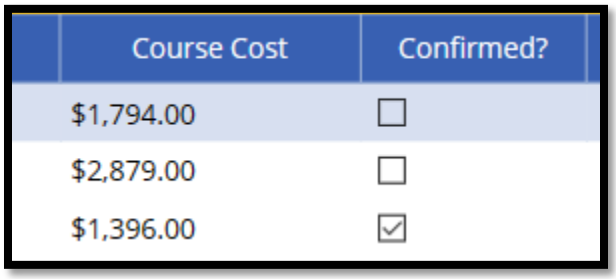


22. Create a new screen based on the Master Screen and call it **Course Registrations**.

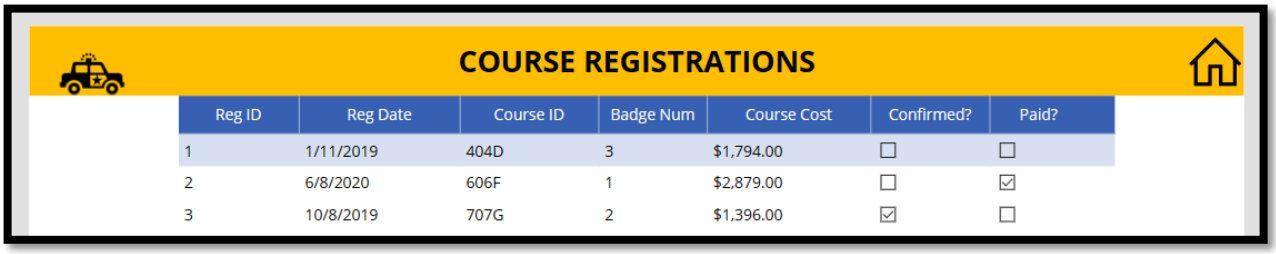
23. Insert a **Data Table** onto this screen connecting SharePoint List **MPD_CourseReg**. Remove the Attachment columns and adjust columns to closely resemble the screen shot below.



24. Select the **Course Cost** column an adjust the Text property formula to read: **Text(Value(ThisItem.'Course Cost'),"[$-en-US]$#,##0.00")**.

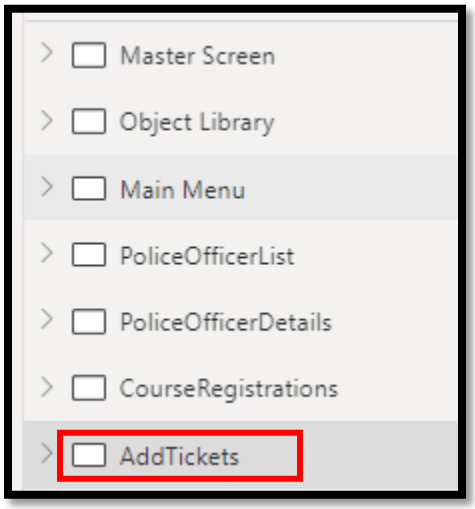


25. Select the **Reg Date** column and change the formula for the **Text** property to read **ShortDate** instead of **LongDate**.

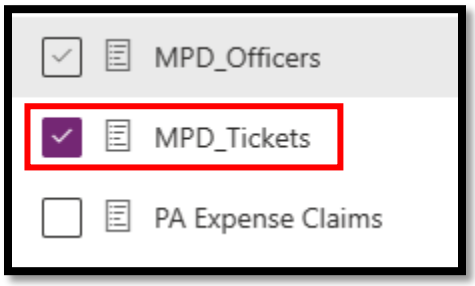


Reg ID	Reg Date	Course ID	Badge Num	Course Cost	Confirmed?	Paid?
1	1/11/2019	404D	3	$1,794.00	☐	☐
2	6/8/2020	606F	1	$2,879.00	☐	☑
3	10/8/2019	707G	2	$1,396.00	☑	☐

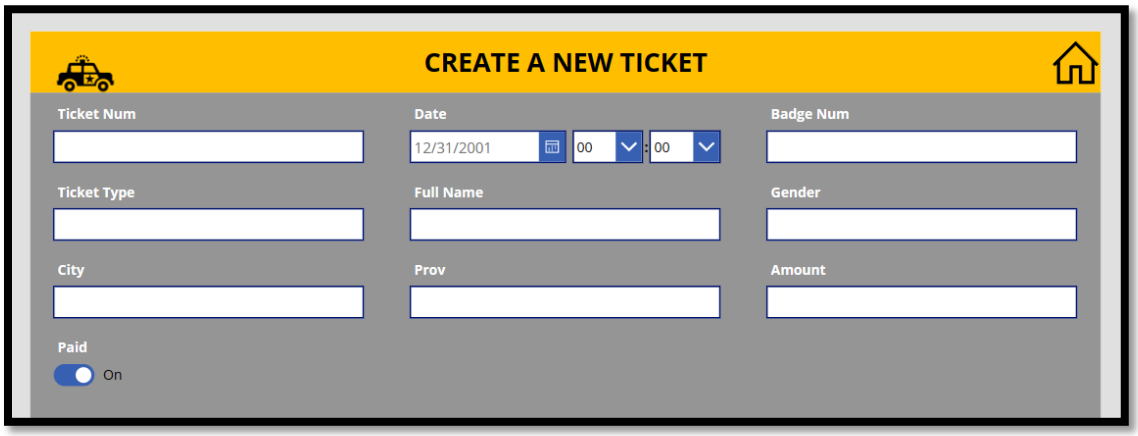
26. Create another new screen off the Master Screen and call it **Add Tickets**.



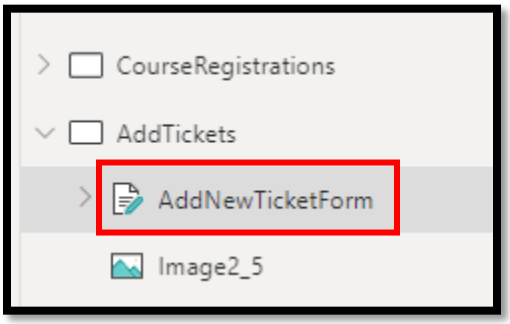
27. Insert a new **Form** in **Edit** mode that is connected to the **MPD_Tickets** SharePoint list. Choose **Insert > Input > Edit Form.**



28. Add a title to this screen, delete the **Attachments** field, adjust the label colours and change the **Display Mode** to **New**.



29. Via the **Tree View**, rename this form **Add New Ticket Form**.

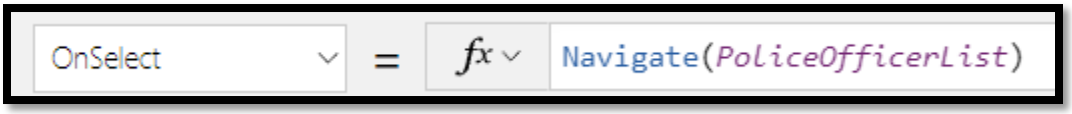


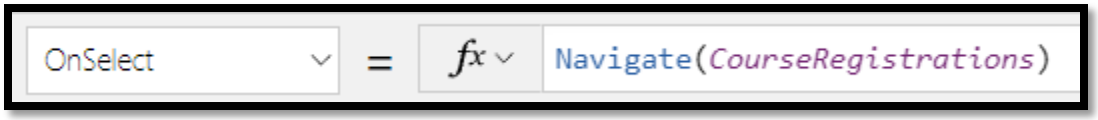
30. Create a new button on this form that submits the data to the SharePoint list it is attached to. You will need to adjust the **OnSelect** property formula to read: **SubmitForm(AddNewTicketForm)**.

31. Return to the **Police Officer Details** screen and add a left arrow to the header **(Insert > Icons)** and adjust its **OnSelect** property formula to read: **Back().**

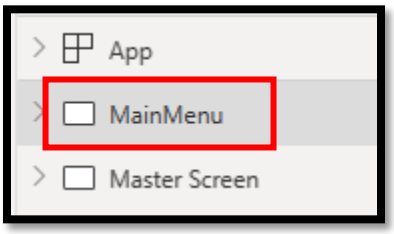
32. For all the screens with the **House** Icon, adjust the **OnSelect** property formula to navigate back to the **Main Menu** screen.

33. Now return to the **Main Menu** screen and adjust the **OnSelect** property formula for each button to navigate to their perspective screens.





34. Via the **Tree View**, move up to the top the **Main Menu** screen. Save all your work, and test your app. Publish it if you want to view it via your pad.



POWER APPS FOUNDATIONS

LESSON 4

Advancing Your Canvas App

LEARNING OBJECTIVES

- Learning Useful Functions
- Grouping Your Objects
- Cascading Drop Down Lists
- Deleting Records
- Creating Popups
- Inserting Other Elements

Lesson 4: Advancing Your Canvas App

For the most part your canvas app is done and ready to deploy. However, you will need some more advanced features like functions, the ability to delete records or create dependent drop-down lists. Adding these advanced features will greatly enhance your app for your users and can make it more interactive. We will also demonstrate how to work with image elements including charts and graphs.

Useful Functions

We have already used some functions to get some of our tasks done and you can always refer to the link given to you at the beginning of the course for all available functions for Power Apps. Here are a couple more that we thought would be useful.

User() – allows you to pull in user information like name, email or photo of the person using your app.

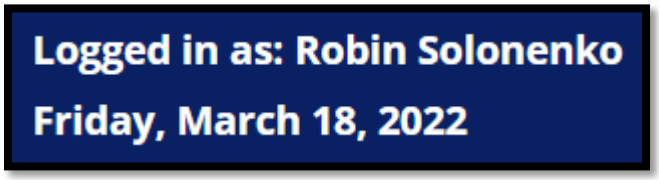
1. Return to your current home screen canvas and create a **Text** label to insert someplace on to the header. Change the **Text** property formula to read: **Concatenate("Logged in as: ", User().FullName)**.

2. When you play your app, you will get the users login full name (this is the name attached to their O365 account).

Today() / Now () – These two are exactly like they are in Excel and they will provide the date (Today) or date and time (Now). In either case, you can use the Text function to adjust the format of each.

1. Still on your current home screen, create another label to insert on to the title bar and change it's Text property formula to read: **Text(Today(),DateTimeFormat.LongDate)**.

2. When you play your app, you will the current date in the format you selected.



Launch() – We already played with this one in the previous section about launching the app to place a phone call. You can also Launch the web browser, email or other apps.

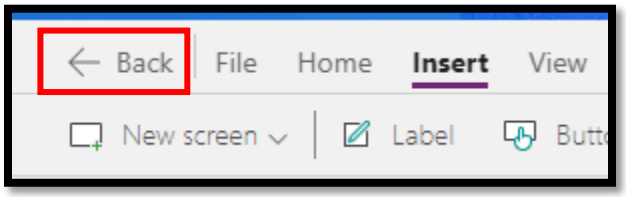
Launch()	
Telephone	Launch("Tel: 555-555-5555")
Email	Launch("Mailto: robins@gmail.com")
URL	Launch("Http://www.google.com")
Other Apps	Launch("/providers/Microsoft.Power Apps/apps/APPIDCODE")

1. From your main home screen canvas insert 3 icons on your title bar and adjust their colours if necessary: Email, Weather, Items.

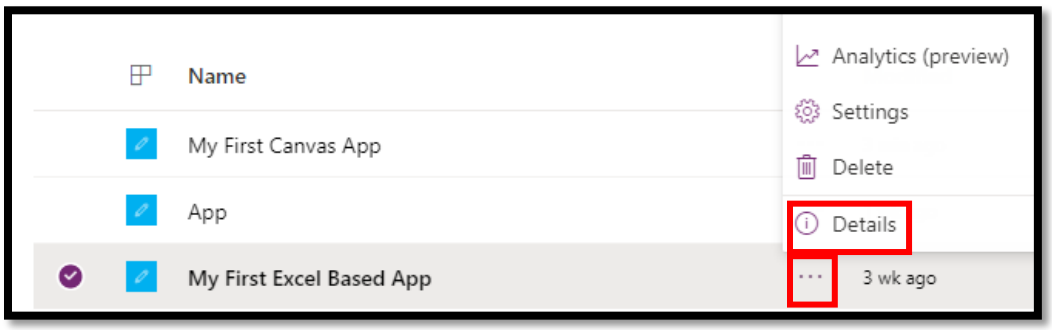
2. Starting with the Email icon, select it and change its **OnSelect** property formula to read: **Launch("Mailto: robins@gmail.com").**

3. Open a web browser and go to theweathernetwork.com and search for your city. Copy the URL to your cities weather forecast then return to the Home Screen canvas. Select the Weather icon and change its **OnSelect** formula to read: **Launch(https://www.theweathernetwork.com/ca/weather/alberta/calgary").**

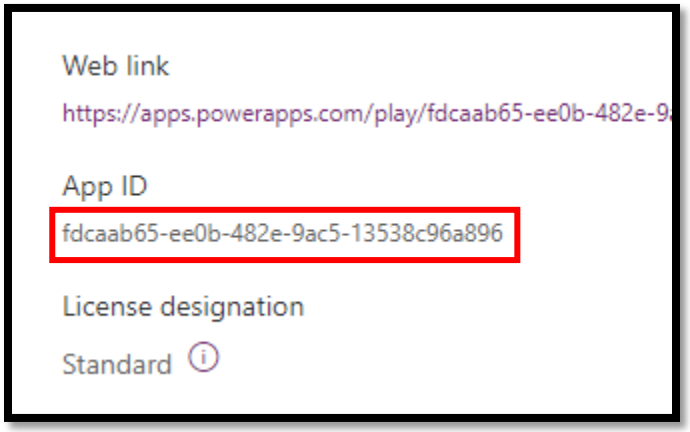
4. Before you do the next step, make sure you **File > Save** your current work. We need to locate the **App ID** to any of our other apps that we have in our profile. To do that click the Back button from the menu bar.



5. Locate the app you want to run, click the **…** and choose **Details**.



6. From the **Details** screen select and copy the **App ID** for this app.



7. Return to your list of apps you have created and go back into edit mode of the one we have been working in previously. Select the Item icon from the Home Screen and change its **OnSelect** property formula to read the following: **Launch("/providors/Microsoft.Power Apps/apps/fdcaab65-ee0b-482e-9ac5-13538c96a896").**

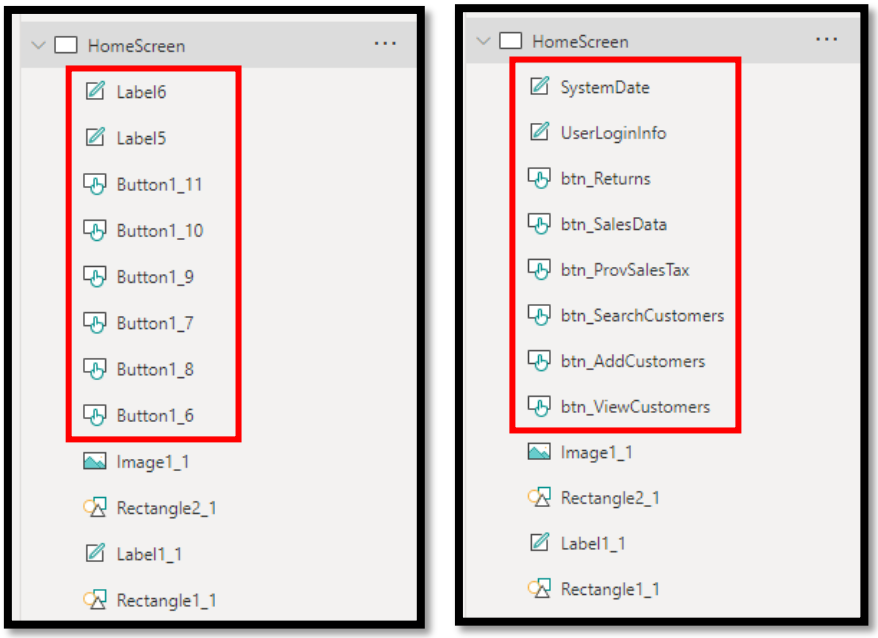
8. Make sure you save your work, then play your app. If you click the **Email** icon it should open your email app. The **Weather** icon will load a new tab and take you to the weather website for your city and the **Items** icon will load the other app you created. Keep in mind if you are on a web browser it will simply open that app in the Power Apps studio. If you are on your phone/pad, it will open and run the app.

Grouping

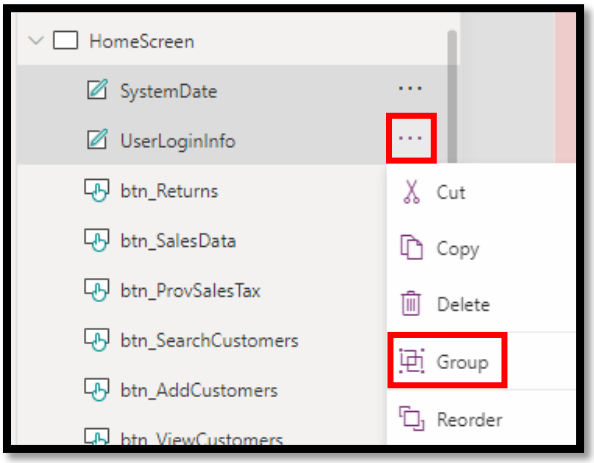
Power Apps usually will auto group its elements that it needs. For example, when you create a form all the data cards it creates are in the same group. Then each data card also has a group of its own elements. Other than the obvious advantage that is allows for smaller lists of objects it also saves time when you want to apply the same format to all the objects in one group.

However, when you manually create objects such as labels, input boxes etc, they are all individually listed in your Tree View. You can group them as you see fit to help organize your screens.

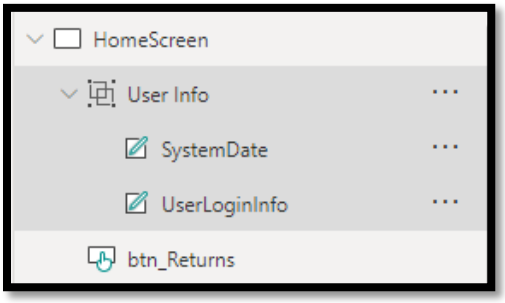
1. Return to the current home screen canvas. We currently have 6 buttons and two labels on this screen. First, rename the buttons and the two labels to something more appropriate.



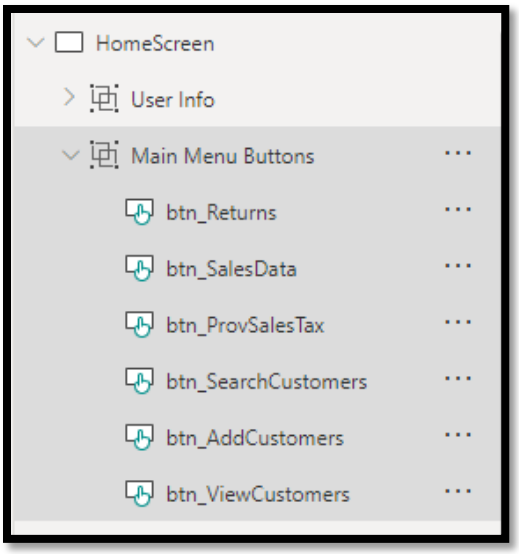
2. From within the **Tree View** and using the CTRL key, select both **SystemDate** and **UserLoginInfo** objects then click the **…** and select **Group**.



3. Rename the group to something appropriate using the **…** to the right of the current group name.



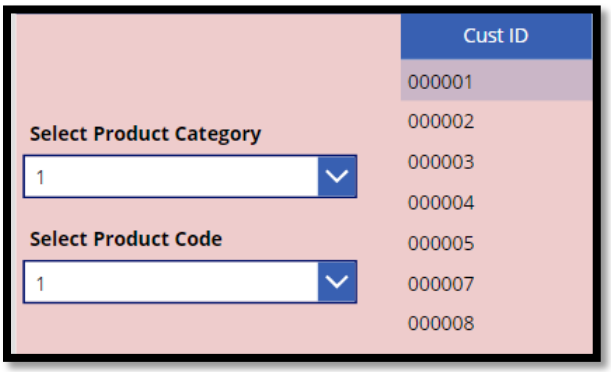
4. Repeat the last step for the 6 buttons we had on the main menu.



Cascading Drop-Down Lists

Earlier we had you create simple drop-down lists to filter a form now we will use multiple drop-down lists to filter a data table. Keep in mind, Power Apps will only allow a max of 2000 records. Which means if you use these drop-down filters, it will only filter the 2000 it sees, not the other potential records it did not pull in, in the first place.

1. Return to the screen where we created a data table for our customer sales data and adjust the current table so you can create two drop-down lists as seen below (**Insert > Input > Drop Down**). Name each drop down via the Tree View.



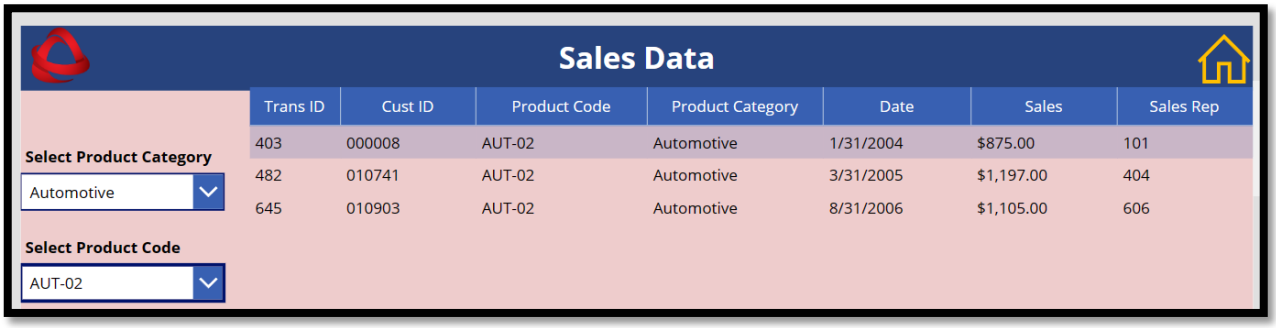
2. Change the **Items** property formula for each drop down object as follows:
   a. Product Category: **SortByColumns(Distinct(SalesData, 'Product Category'),"Value")**
   b. Product Code: **SortByColumns(Distinct(SalesData, 'Product Code'),"Value")**

```
SortByColumns(Distinct(Borealis_SalesData, 'Product Category'),"Value")
```

```
SortByColumns(Distinct(Borealis_SalesData, 'Product Code'),"Value")
```

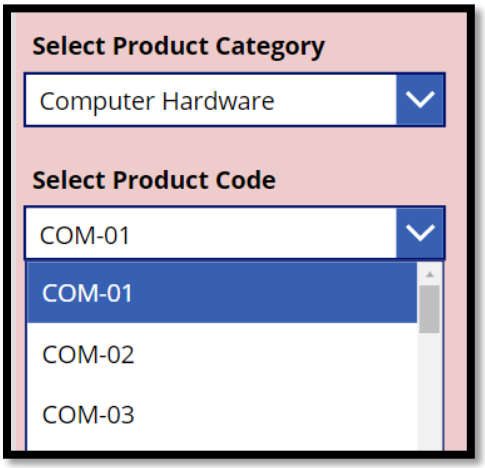
3. Now select the data table itself (best from the **Tree View**) and change its **Items** property to the following: **Filter(Borealis_SalesData, 'Product Category' = ProdCatDropDown.Selected.Value, 'Product Code' = ProdCodeDropDown.Selected.Value).**

4. If you go play your app, you should be able to choose a category and a product code, and your data table will filter to your choices.

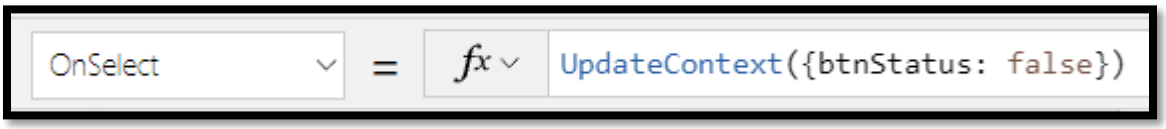


5. Now that you have tried it, you will have noticed that you have way too many **Product Codes** to scroll through and you are correct. What we need to happen is that when a user selects the **Product Category** above, then the **Product Codes** should adjust to only show those unique codes for that category. Adjust the formula for the **Items** property on the Product Code dropdown to read as follows: **SortByColumns(Distinct(Filter(Borealis_SalesData, 'Product Category' = ProdCatDropDown.Selected.Value), 'Product Code'),"Value").**

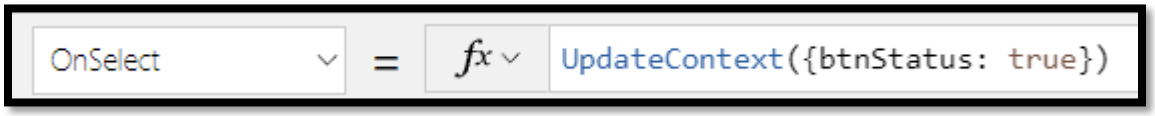
6. Now when you play your app, choose your **Product Category**, your **Product Codes** will only show codes available for that category.



7. Now we need to have the ability to un-filter and return the list back to the full data. To do this, select the **ProdCatDropDown** and change its **OnSelect** property formula to read: **UpdateContext({btnStatus: false}).**

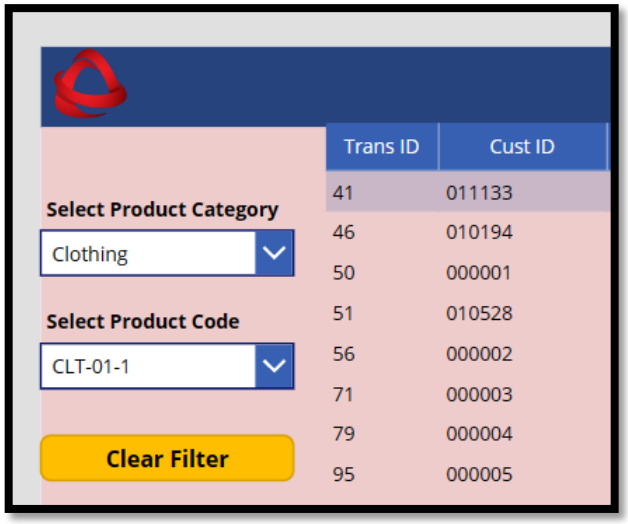


8. Add a new button underneath both drop downs and adjust its **OnSelect** property formula to read: **UpdateContext({btnStatus: true})**.



9. Finally, via the Tree View, select your **Data Table** object and update the current Items property formula to read: **If(btnStatus=true, Borealis_SalesData, Filter(Borealis_SalesData, 'Product Category' = ProdCatDropDown.Selected.Value, 'Product Code' = ProdCodeDropDown.Selected.Value)).**

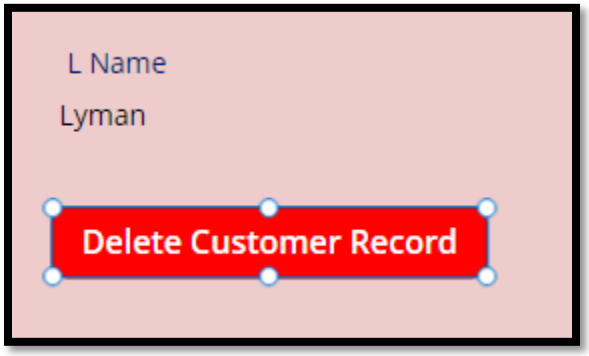
10. Now when you play your app, as soon as you choose a Product Category it will start the filtering process, but once you click the **Clear Filter** button, it will return to all data.



Deleting Records

The reason this is put towards the end of the course is not everyone wants their users to be able to delete records. Maybe the administrator but not just every user. So, the action is self-explanatory, it will delete a record from your data source. You just need to create the button and the action to do it. Be careful as there is no undo!

1. Switch to the **Customer Details** screen and create a new button with proper labeling.



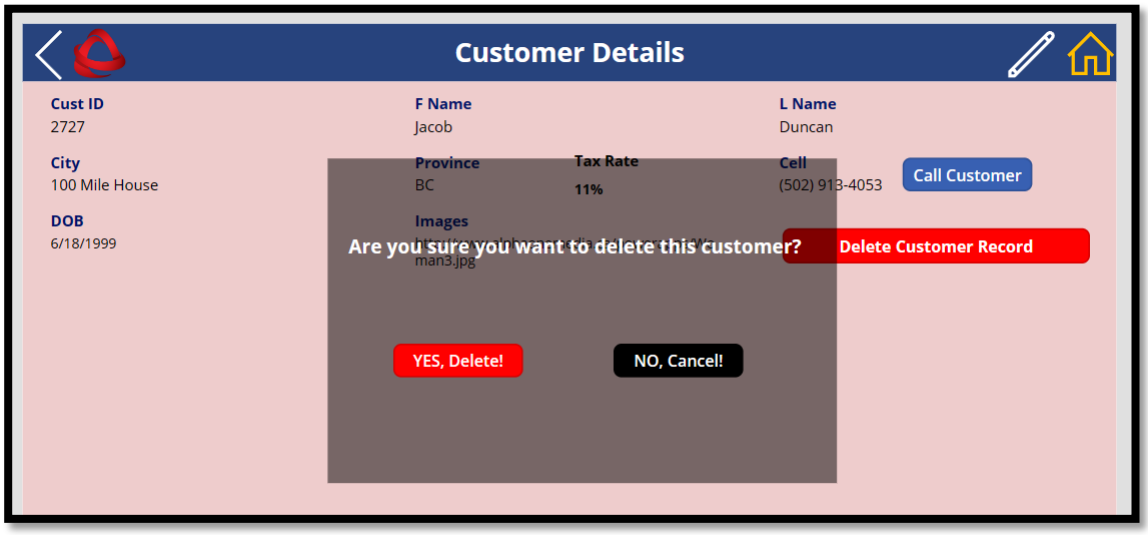
2. Change the OnSelect property formula to read the following: **Remove(Borealis_Customers_1, ListofCustomers.Selected);Navigate(ViewCustomers).** Play your app and delete a customer, it may take a minute to delete but it should turn to the View Customers Screen once complete.

```
fx  Remove(Borealis_Customers_1, ListofCustomers.Selected);Navigate(ViewCustomers)
```

Creating Popups

Popups are a great way to communicate with your users, especially when we perform the last action of deleting a record. You may want to confirm with the users first before you perform the action. But they can be used for any reason to help the users understand what actions are happening when they do certain things like click buttons etc.

1. On the screen that has your current **Delete Record** button, create a rectangle, text label and two buttons as you see in the screen shot below. Also, via Tree View, make sure you group those 4 elements and give the group a name. If you wish to change the opacity of the rectangle, adjust the 4th input on the RGBA color of the Fill property to .5.




```
fx  RGBA(0, 0, 0, 0.5)
```

2. Now we need to change and add some functions to some of these objects we created. First, you need to cut and paste the formula from the red **Delete Customers Record** button and insert it into the **OnSelect** formula for the **Yes, Delete** red button, while adding an **UpdateContext** function to update a new variable called 'confirmation'.

```
fx  Remove(Customers, ViewCustomersForm.Selected);UpdateContext({confirmation: false})
```

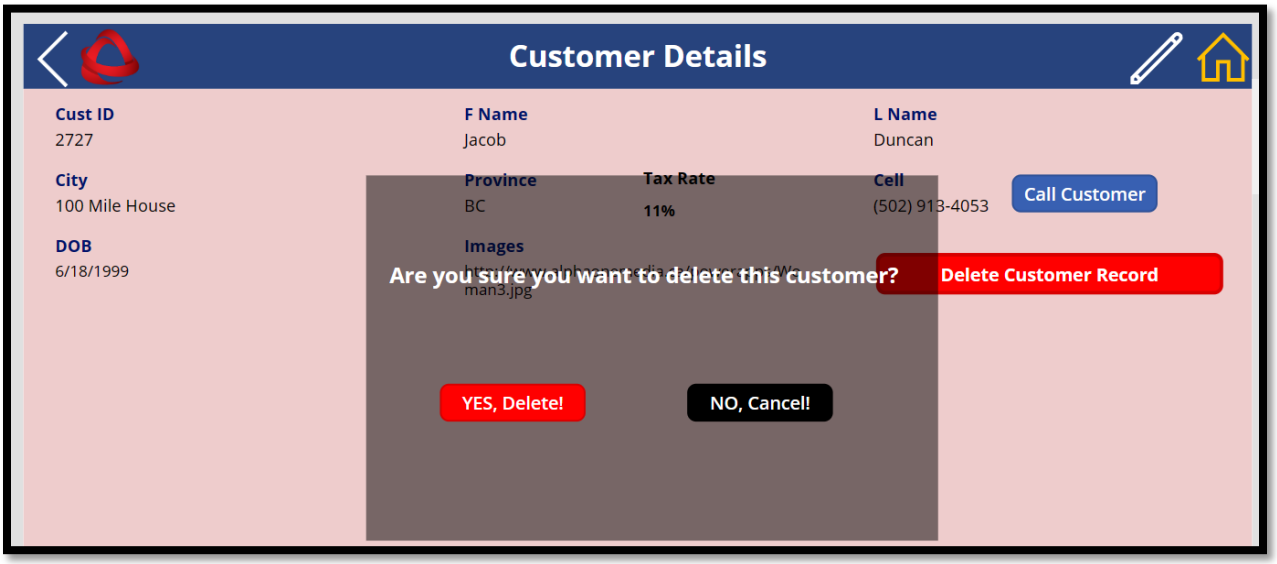
3. Change the **OnSelect** formula for the blue **No, Cancel** button to as follows: **UpdateContext({confirmation: false})**.

```
fx  UpdateContext({confirmation: false})
```

4. Change the **OnSelect** formula for the red **Delete Customer Record** button to as follows: **UpdateContext({confirmation: true})**.

5. Lastly, change the **Visible** property formula for the group that you created to as follows: **confirmation**.

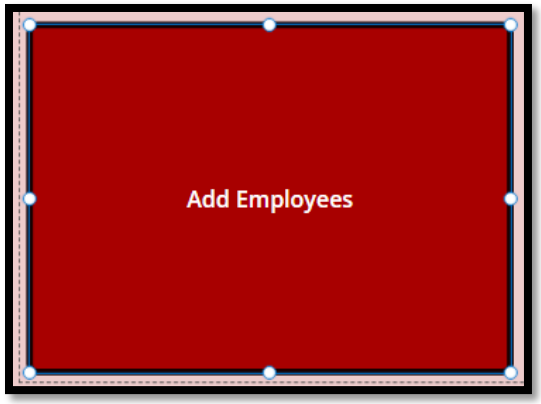
6. Now when you play your screen, the grouped items will appear when you click the red **Delete Customer Record** button and will close out when you cancel.



Input Elements

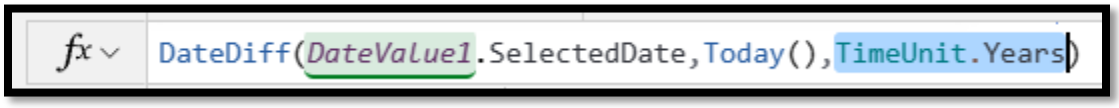
Input elements allow users to input values that of course we later store in our data source. We have already covered the basic popular ones such as Text boxes, Buttons and Drop Downs but there are so much more to discuss. A good form layout is essential at this point as well, so your input form has a natural flow with your data for your users.

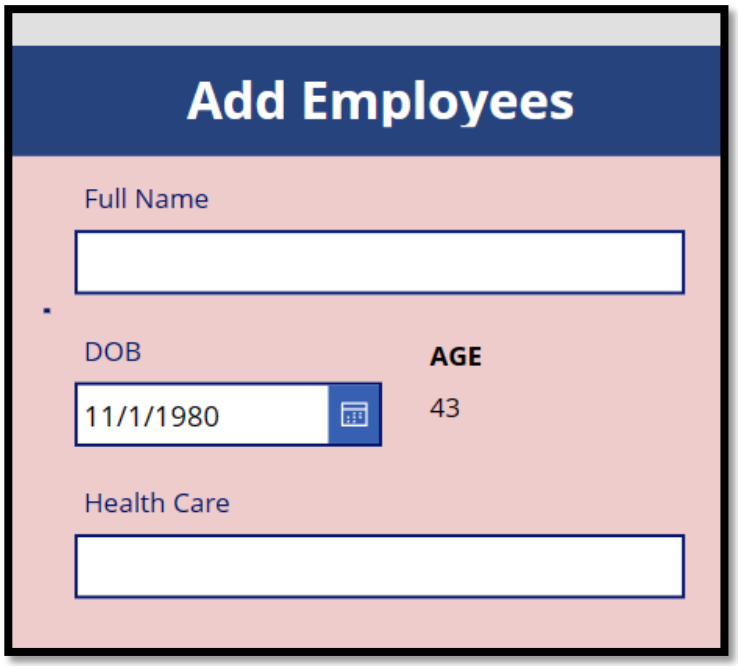
1. If you don't already have one, create a new screen linked to a data source. We have a table source called **Borealis_Employees** and we linked it from the **Home Screen** button **Add Employees** with the formula on the **OnSelect** Property: **NewForm(AddEmpForm);Navigate(AddEmployees)**.



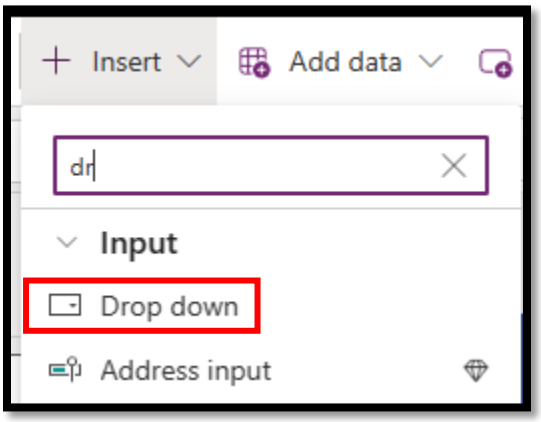
2. Using the form fields (data cards) above let us show you how to manipulate or change them to be different input elements. Starting with the DOB form field card, we want to calculate the age of the person once they select their **DOB**.
    a. First delete the **hours** and **seconds** dropdown and fix the related errors.
    b. Add a text label that says **Age** and a second text label where you change the **Text** property to ready: **DateDiff(DateValue3.SelectedDate,Today(),TimeUnit.Years)**. Play

your screen to see your results after you assign your DOB. (You will have to play from the main menu first.





3. Activate the **City Data Card** from **Tree View** then from the Insert menu select In**put > Drop down**. Power Apps will ask you to unlock and add before it allows the insert. We are doing this, so our new input element takes on all the characteristics of the City Data Card.



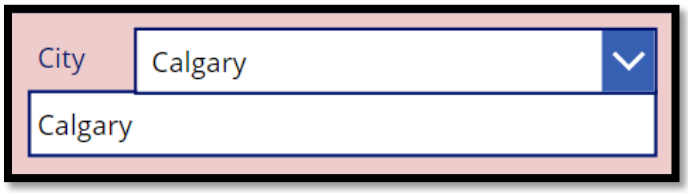
4. With the new **Drop Down** selected, change its **Items** formula to read the following: **["","Vancouver","Edmonton","Calgary","Saskatoon","Regina","Winnipeg","Ottawa","Toronto"," Montreal","Quebec City"]**.

```
fx ["","Vancouver","Edmonton","Calgary","Saskatoon","Regina","Winnipeg","Ottawa","Toronto","Montreal","Quebec City"]
```

5. Now selected the "Current" city field (**DataCardValue#**) and change its default property formula to read: **Dropdown2.Selected.Value**. The reason why we are doing this is because it is this **City** field that is connected to the data source and when we update the fields it will take the value assigned to this field and push it to the field in the data source.

```
fx Dropdown2.Selected.Value
```

6. You can now play and test this feature. When you select a city from the drop down it should also populate the original city field.



7. Once you're done with your testing you can change the **Visible** property formula of the **City** field to false and adjust the drop down to fit better. Although it may be hidden from view, it will still be assigned the value of your drop-down selection.

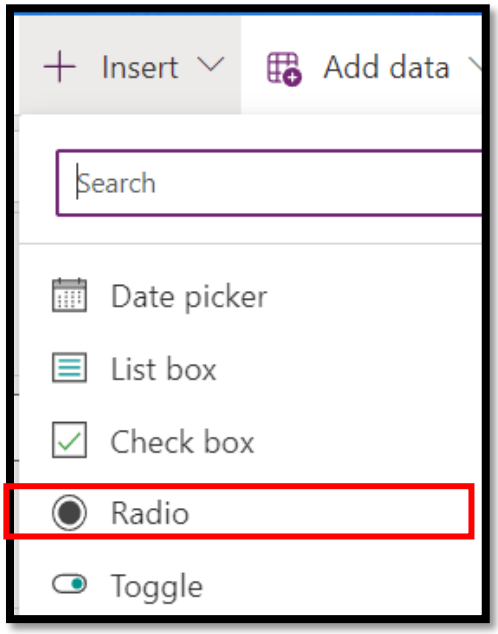
8. Now select the **Prov** field and change its **Default** property formula to read: **LookUp(Borealis_ProvLookUp,City = Dropdown2.Selected.Value,Prov).** This will look up the city that is selected in the previous step into a SharePoint list called **Borealis_ProvLookUp** (Must connect to it first) and return the corresponding value in the **Prov** column of that table.

```
fx  LookUp(Borealis_ProvLookUp,City = Dropdown2.Selected.Value,Prov)
```

9. Without having any data card selected from the **Insert** menu > **Input** options choose **Slider** and place it someplace beside the **Weight** field. Change the properties of the **Slider** to a Max of 300 and then change the **Default** property formula for the **Weight** data card to read as follows: **Slider#.Value**. When you play your app, you should be able to adjust your slider for your weight and it will insert it in the Weight field.

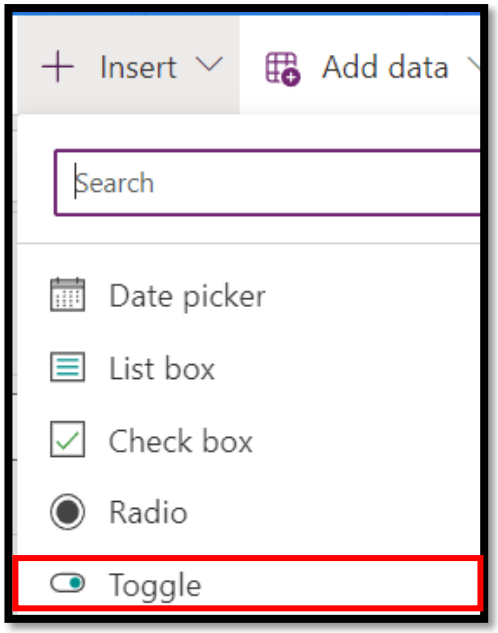
```
fx  Slider1.Value
```

10. With the **Health Card** data card active, insert the **Radio** buttons (**Insert > Input**). Then change the **Items** property formula to read the following: **["Yes","No"]**.



11. Now you can change the **Default** property formula of the Health Care text box to read: **Radio2.Selected.Value**. Test to make sure it takes the value of the radio buttons, then return and change the **Visible** property formula of the health care text box to false.

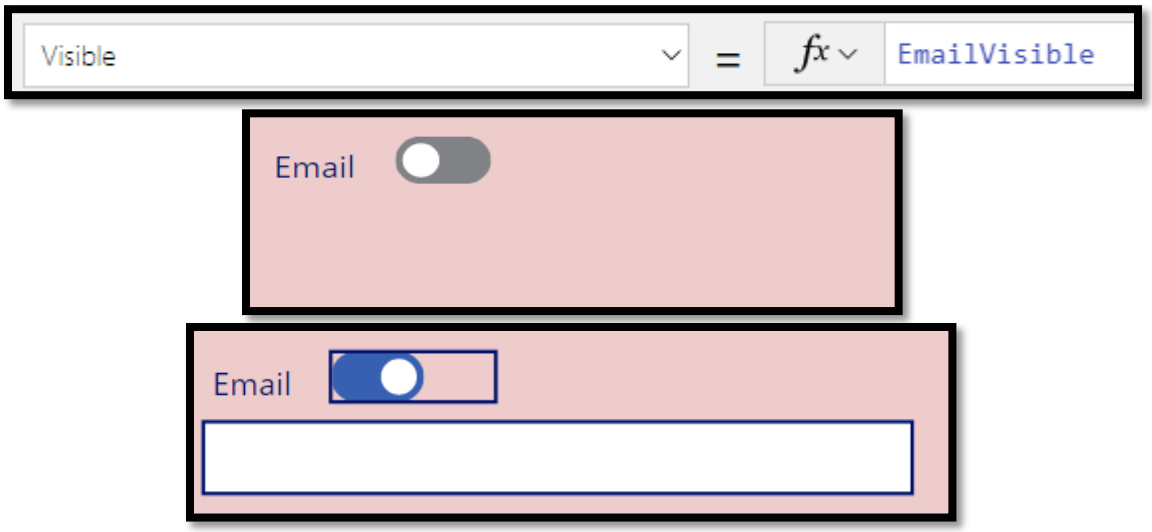
12. Let us do one more input element that is fun to use, the **Toggle** element. In our case we will select the **Email** data card then go to **Insert > Input > Toggle**.



13. Now we need to update the **OnChange** property formula to the following: **If(Toggle2.Value = true,UpdateContext({EmailVisible: true}),UpdateContext({EmailVisible: false}))**, followed by changing the **TrueText** and **FalseText** properties formula to **“”**.

```
fx  If(Toggle2.Value = true,UpdateContext({EmailVisible: true}),UpdateContext({EmailVisible: false}))
```

14. Lastly change the **Visible** property formula for the **Email Data Card** field to : **EmailVisible**. When you go play your screen, the toggle button will turn on or off the field for the email.



15. Lastly, you can now create a **Submit Form** button to submit the new record to the Excel data source. Change the **OnSelect** property formula for the button to **SubmitForm(AddEmpForm)**.

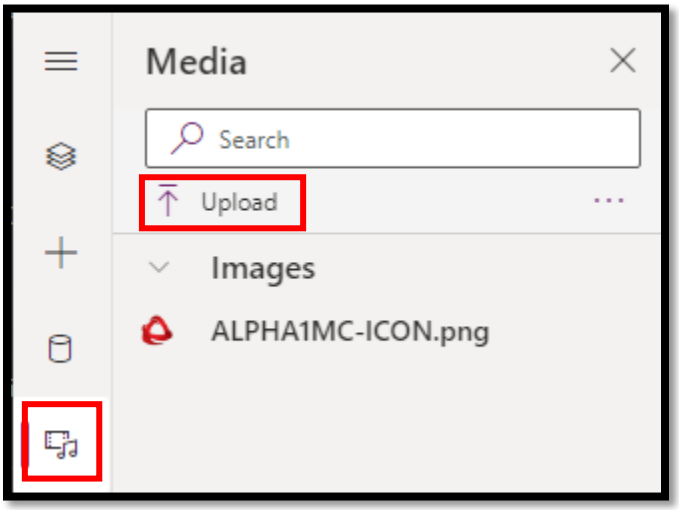
16. Finally let us organize the screen a little better by activating the form from the **Tree View** then selecting **Edit Fields** in the properties pane. You can now drag and drop your fields that you have been working with in the order you want them to appear left to right.

Media Elements

If you're creating your apps to be used on your phone or pads, you may as well take advantage of some of the media elements features. Creating images or having your users upload their own images, using a signature box or using your phone's camera or microphone are just some of the media elements you can use for your app.

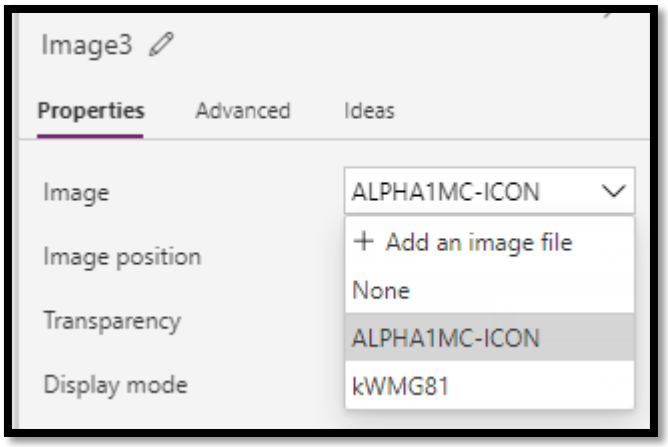
For the rest of this section, we will demonstrate the purpose of some of the Media elements and how to use them, but you will need to use your knowledge you just learnt how to connect them to data sources if you so desire.

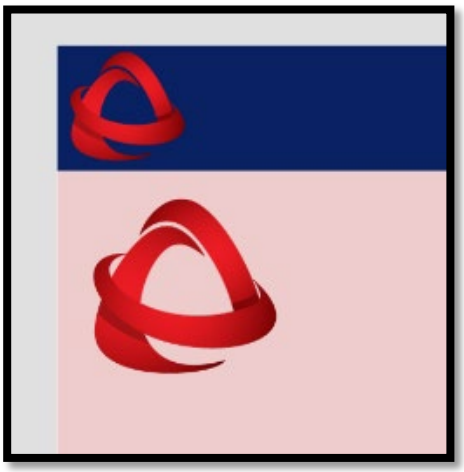
1. Start with a blank screen (duplicate the master) and click on the **Media** icon from your left-hand menu and you should already see the image that we inserted earlier. Click on the **Upload** option and upload any other image you desire.



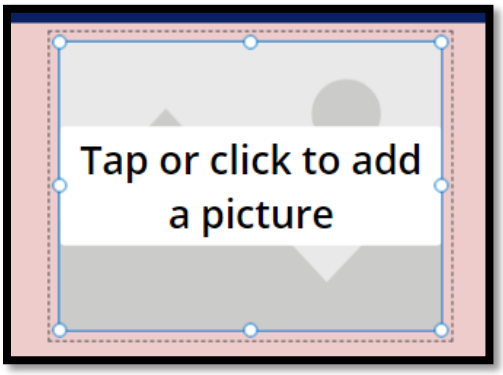
2. Now from your newly created screen, choose **Insert** > **Media > Image**.

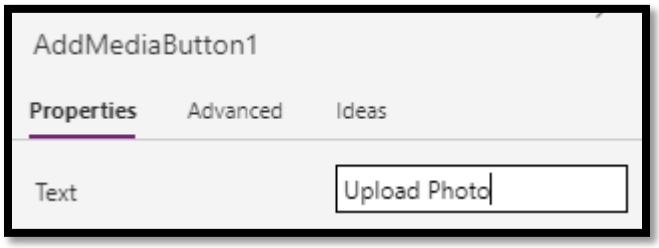
3. With the image placeholder active, from your **Properties** pane, **Image** field choose your image from the drop down.



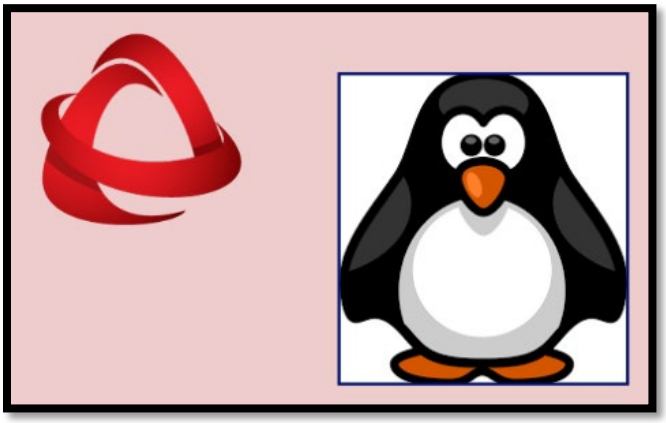


4. From the same **Insert > Media** options choose **Add Picture**. Change the text to something more appropriate. Also delete the field contents for **Change Picture Text**.

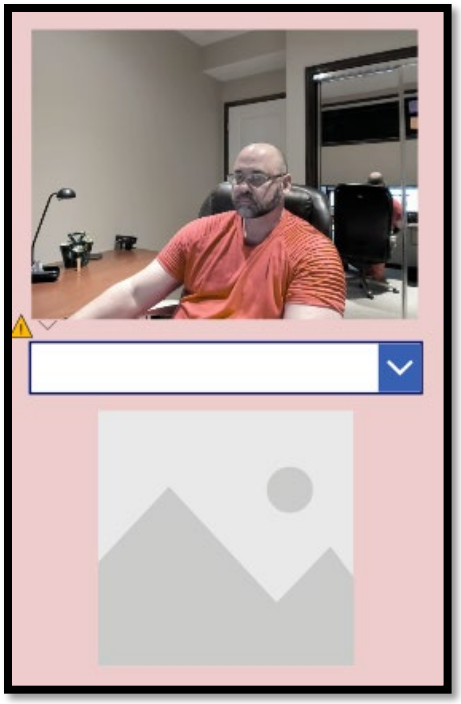




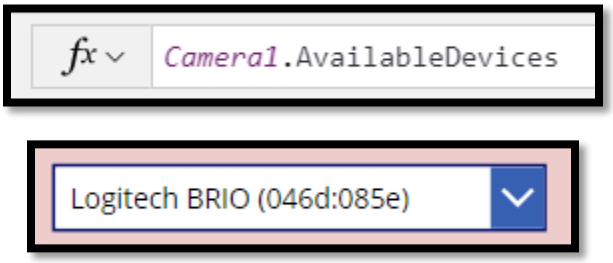
5. When you test it, the user can click it and upload their own photos.



6. Return to your canvas and create the following: **Insert > Media > Camera, Insert > Media > Image** and **Insert > Input > Drop Down** and arrange them as you see. Below.

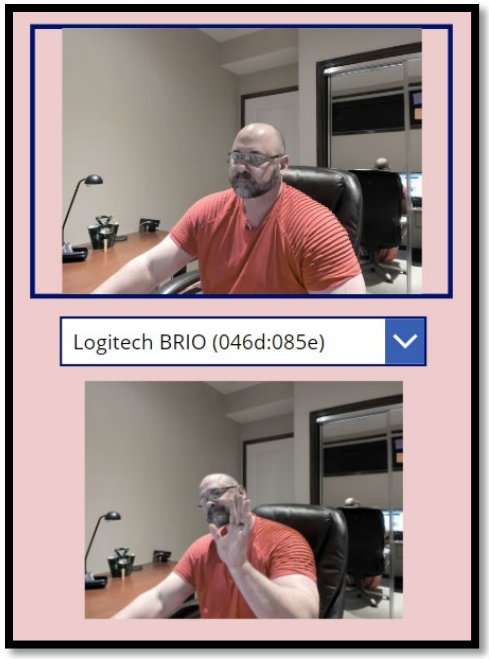


7. Now change the Drop Down's **Item** property formula to read: **Camera1.AvailableDevices**. This will show the users all the cameras they have available on the system or the front / back of your phones.



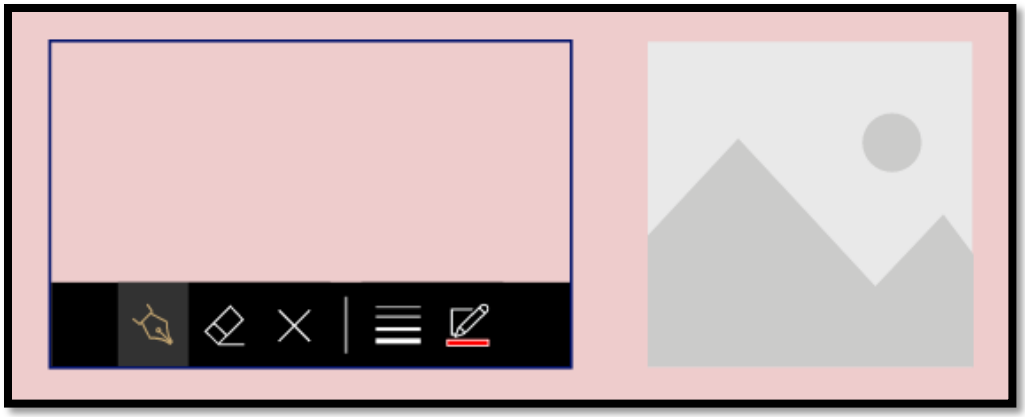
8. Now change the **Image** property formula for the image to read: **Camera1.Photo**. This will store the photo that the users save when they are running the app.

9. Now when you play the screen, if you have multiple cameras, you can choose which one you want, and if you are ready to take your photo, just click once on the video stream and it will snap an image and put it to the image place holder.

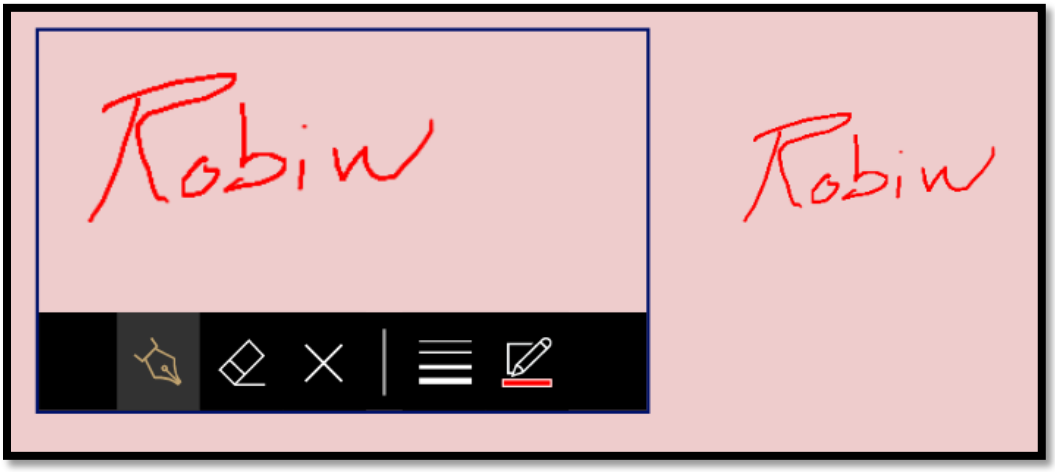


10. On the same screen, insert the **Video** element. Just like the image element, you can link it to / upload a video into your app. However, instead of trying to store large files, you can link to them directly to YouTube. With the Video Element active, change its **Media** property formula to read the following: **"https://www.youtube.com/watch?v=B83nmCSwRuw"** (or copy and paste any link to any video on YouTube.

11. The last element to discuss is actually an **Input** element however if you want to save it as a photo then it becomes a **Media** element. From your current screen choose **Insert > Input > Pen Input**. Then from the **Insert** > **Media > Image**.



12. With the **Image** placeholder active, change its **Image** property formula to read: **PenInput1.Image**. When you play this screen and write in the Pen input square, the image will then be pushed over to the Image placeholder which now can be saved.



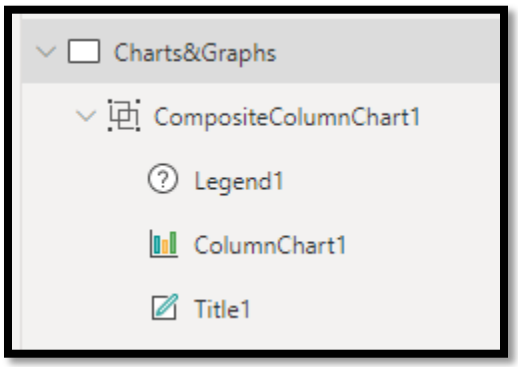
Charts & Graphs

Power Apps comes with the ability to insert three basic charts: Column, Line, and Pie. However not to complex of charts. Like everything else, the data table would have to be created in your source file and then connected here. The best use for graphs is to connect your chart element to a Power BI Tile from your Power BI Dashboard!

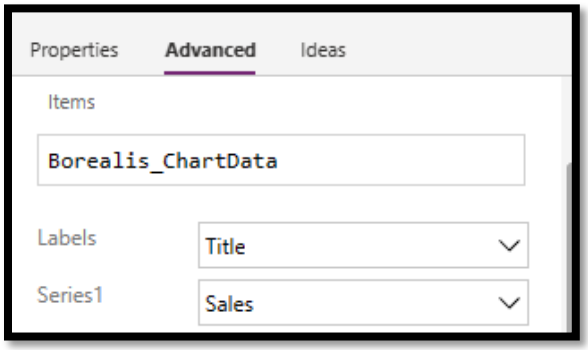
We will demonstrate how to connect to your Power BI Dashboard but before you do, you would have had to create your Power BI file first, publish it to your app.powerbi.com service, then create your dashboard with various tiles. Then you will come back here and follow the instructions below. Also, a must: the email account you used/assigned to use your current Power Apps environment MUST be the same email account that is used / attached to your Power BI web service in order for you to see the dashboard items.

1. Duplicate your **Master Screen** template, and call it Charts and Graphs - then choose **Insert** > **Charts** > **Column Chart**.

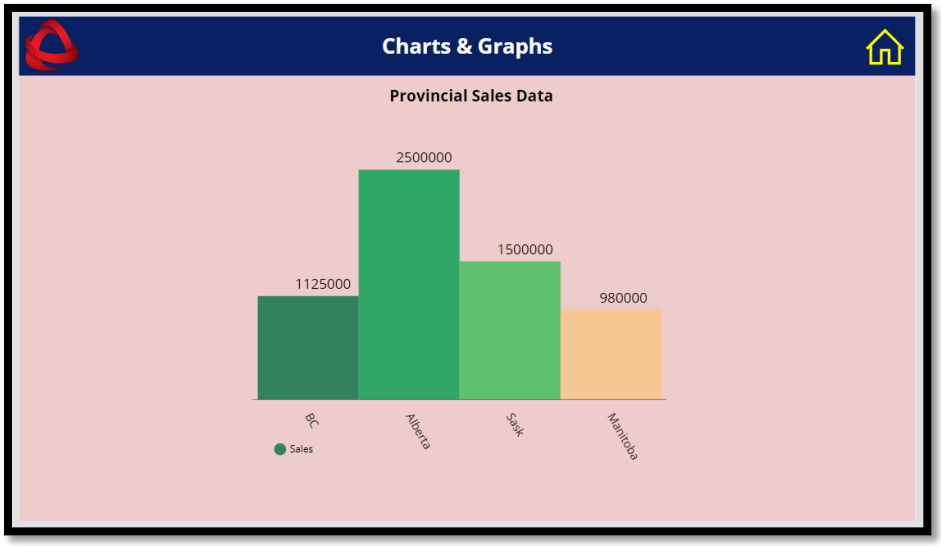
2. Activate the **ColumnChart1** object in your **Tree View**, then change the **Items** property formula to read the following: **Borealis_ChartData** *(You will have to connect to the SharePoint List called Borealis_ChartData*).



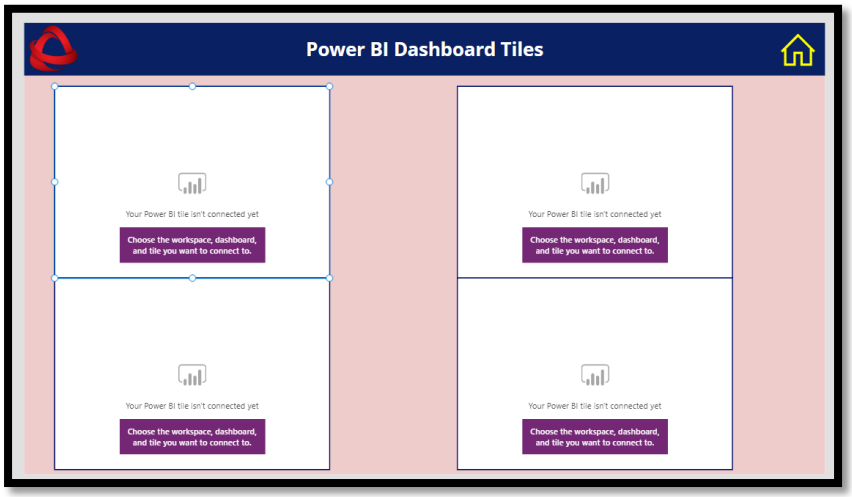
3. The last step would suffice if your data source is Excel, however if your data source is a SharePoint list, you have to map the fields from SharePoint via the Advanced tab.



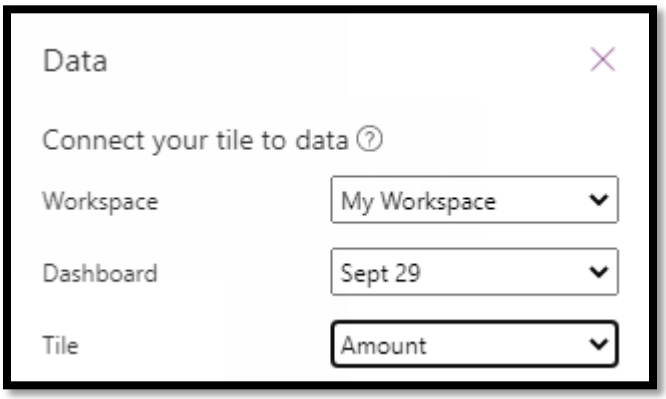
4. You can now adjust other things like your chart title, font size, colour etc.



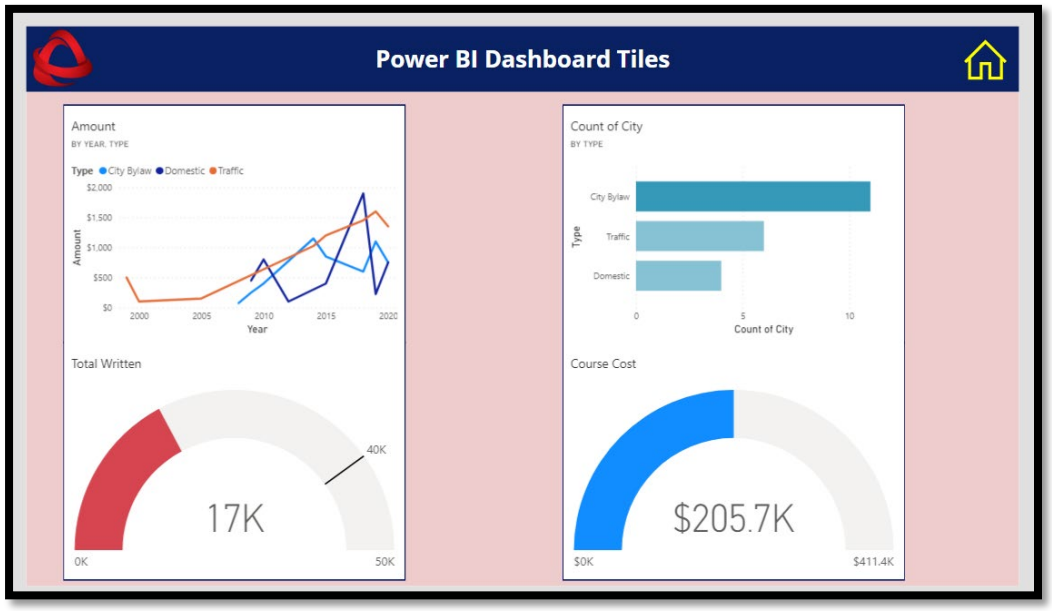
5. If you have an existing Power BI file published on the web service, you can do the following steps.
   a. Create another screen of the Master screen template and then choose **Insert** > **Charts > Power BI Tile** and insert 4 tiles.



   b. Select any one of the 4 Power BI Tiles (click the purple button if necessary) and a Data pane will appear. Choose from each drop down your Workspace, Dashboard and Tile.



c. Repeat the last step for each of the 4 Power BI Tiles. You can see that you can reference different dashboards if necessary.



NOTE: For your users of your app to be able to see the Power BI tiles, you must have "Shared" it within the Power BI Web Service. But before you can do that, you must have the Pro license of Power BI!

Lesson 4: Exercise

1. Return to the **Exercise 3 App** file, go into edit mode and do a **Save As** and call it **Exercise 4 App**.

2. From with the Exercise 4 App, on the **Main Menu** screen add two **text labels** each with the following formulas for their Text property. Place the labels on the top menu bar area.

```
Text = Concatenate("You Are ",User().FullName)
```

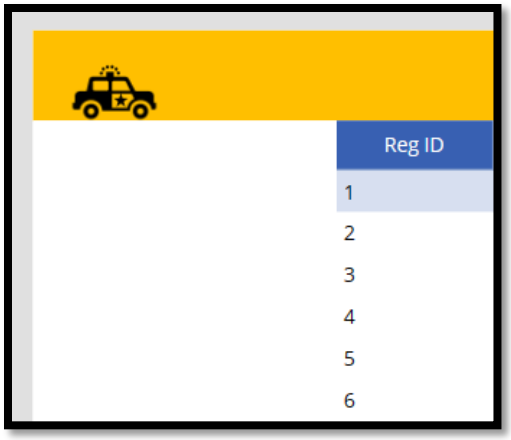
```
Text = Text(Today(),LongDate)
```

3. Insert two icons – **Email** and **Globe** – and place them on the opposite side of the top title bar. Change the **OnSelect** property formula for each of the following (the email should be your email and the company URL should be the URL of the company you work for).

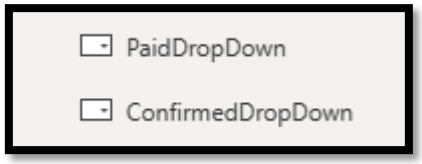
```
Launch("Mailto: Student_01@alpha1mc.com")
```

```
OnSelect = Launch("Http://www.alphaonemedia.ca")
```

4. Switch your screen to the **Course Registrations** screen (one with the data table) and move the table over to fit two drop downs and a button.



5. Now we are ready to add the filters. Add 2 text labels and 2 dropdowns as you see in the screenshot below. Also rename the drop downs in the **Tree View** as well.



6. Select the drop down for the **Confirmed** object and change its **Items** property formula to read: **["","Yes","No"]**, then change its **OnChange** property formula to read: **UpdateContext({btnStatus: false});If(ConfirmedDropDown.Selected.Value = "Yes", Set(ConfirmedChoice,true),Set(ConfirmedChoice,false)).**

7. Now you can change the **Items** property formula for the **Paid** dropdown to: **["","Yes","No"]**, and the **OnChange** property formula to read: **If(PaidDropDown.Selected.Value = "Yes", Set(PaidChoice,true), Set(PaidChoice, false));UpdateContext({btnStatus: false})**.

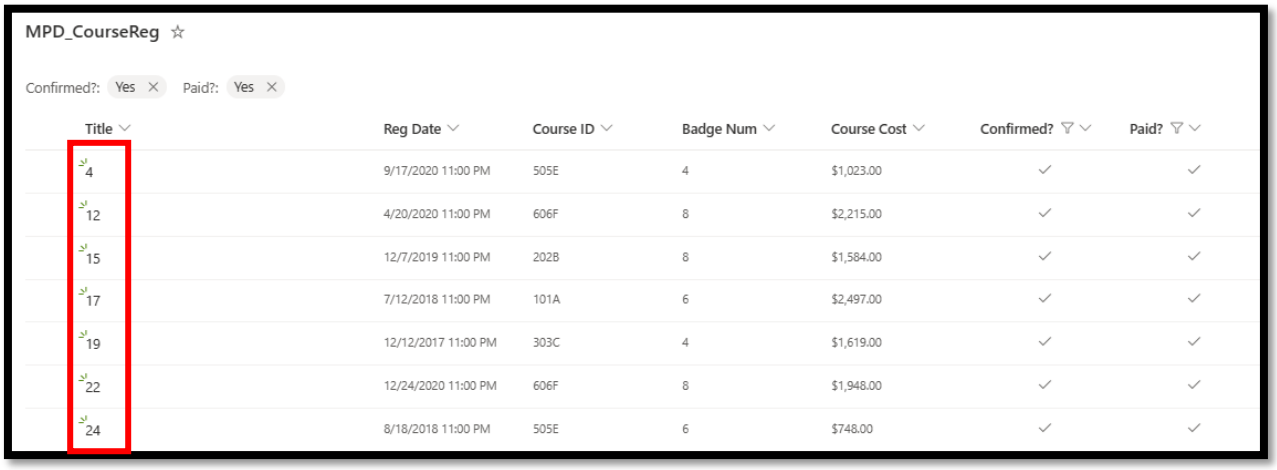
8. Now you can add a new button and label it **Reset**, then change the **OnSelect** property formula to read the following: **UpdateContext({btnStatus: true})**.

```
OnSelect = fx UpdateContext({btnStatus: true})
```

9. Finally, via the Tree View, select the **Data Table** and change its Items property formula to read the following: **If(btnStatus = true,CPS_Registrations, Filter(CPS_Registrations, 'Paid?' = PaidChoice, 'Confirmed?' = ConfirmedChoice)).**

```
fx If(btnStatus = true,MPD_CourseReg, Filter(MPD_CourseReg, 'Paid?' = PaidChoice, 'Confirmed?' = ConfirmedChoice))
```

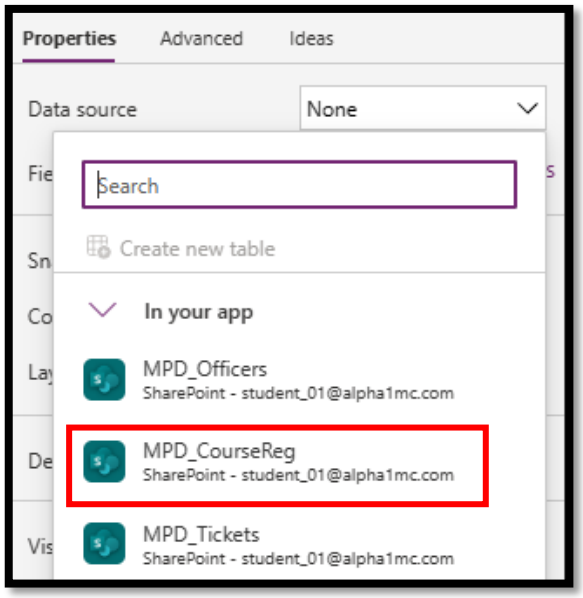
10. Now go test your screen. If you change both the confirmed and paid drop downs to Yes you will see the result below. Then go to your SharePoint CPS Registration list and filter for the same to ensure you get the same results.

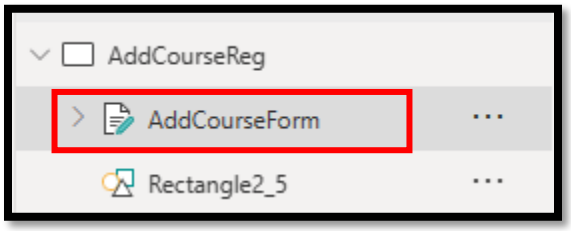


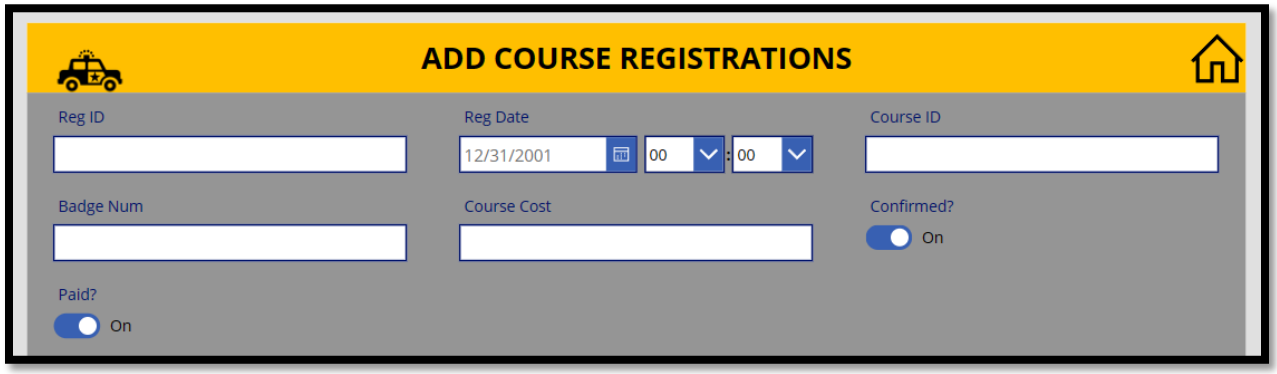
***NOTE**: Step #10 cannot handle Boolean fields that are null. Hence it would best if that field was just a simple text field with true / false inputs instead of actually Boolean fields.*

11. Create a new screen from your **Master Screen** and call it **Add Course Registrations.**

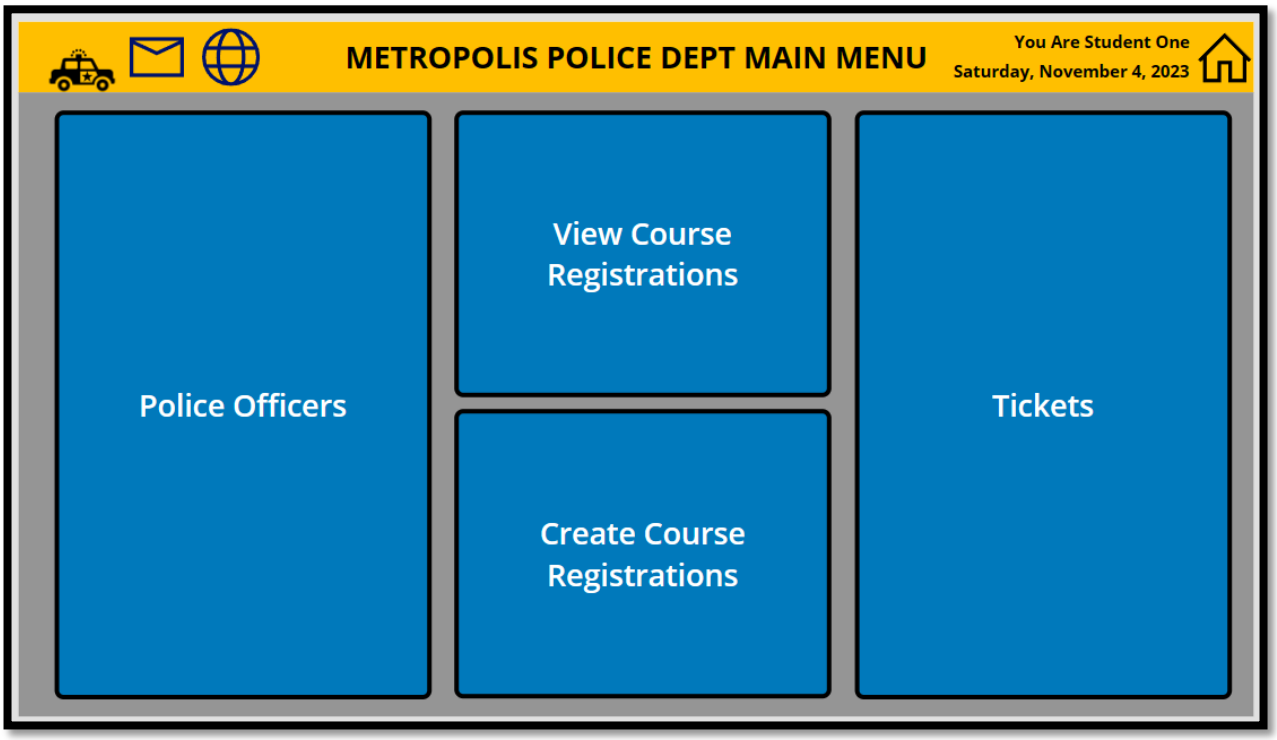
12. From the menu bar **Insert > Input > Edit form** and attach the form to the **MPD_CourseReg** SharePoint list and then delete the **Attachments** field from the form. Via the **Tree View**, rename the form something appropriate.





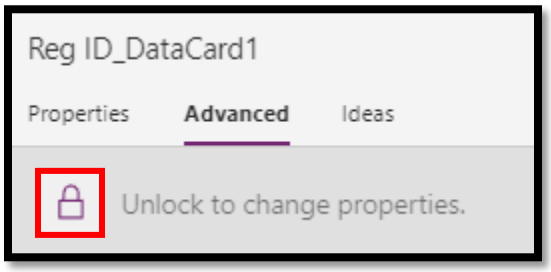


13. Return to the **Main Menu** screen and adjust the current **Course Registrations** button and change the text as you see in the screen shot. Then add another button as you see below that will link to the new **Add Course Registration** screen you just created.



14. Select the newly created button and link it to the **Add Course Registration** form by changing the **OnSelect** property formula to read: **NewForm(AddCourseForm);Navigate(AddCourseRegs)**. When you play the menu and click the **Create Course Registrations** button, it will take you to the new form and put it in add/edit mode.

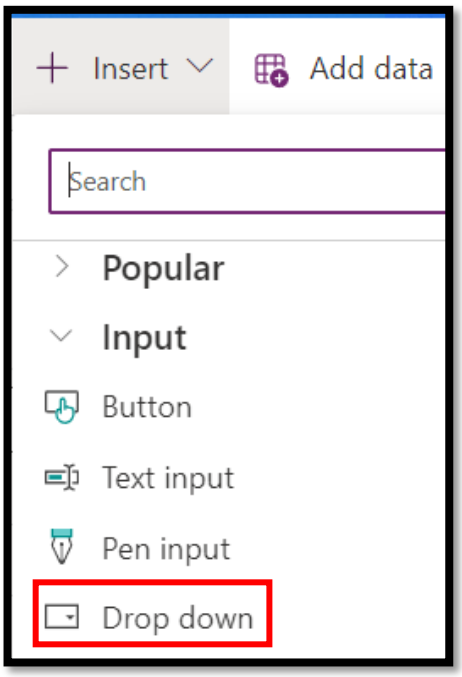
15. Return to the **Add Course Registration** form and select the **Reg ID** field and unlock it from the **Advanced** properties pane.

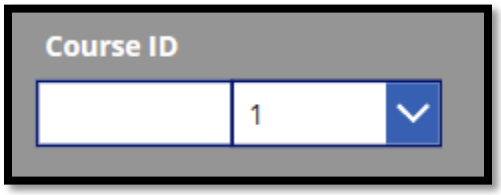


16. Activate the **Reg ID** input box and change the **Default** property formula to read the following: **First(SortByColumns(MPD_CourseReg,"Reg_x0020_ID",SortOrder.Descending)).'Reg_x0020_ID'+1**. This will search for the highest number in the Reg ID field and add 1, therefore the number in the screen shot below will change as more students do this exercise.

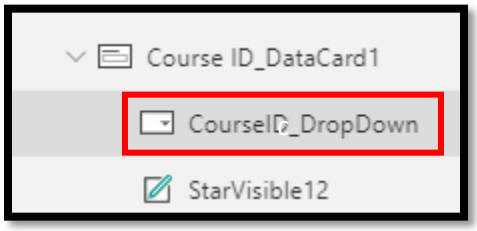
```
First(SortByColumns(MPD_CourseReg,"Reg_x0020_ID",SortOrder.Descending)).Reg_x0020_ID+1
```

17. Select the **Course ID** data card group and from the menu bar choose **Insert > Input > Drop down**. It will ask you to unlock and add. For now, align the current input field and the drop down as you see below.

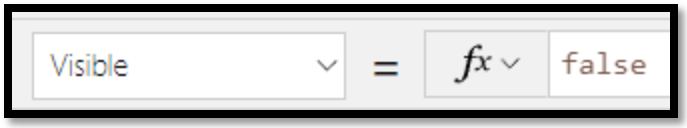




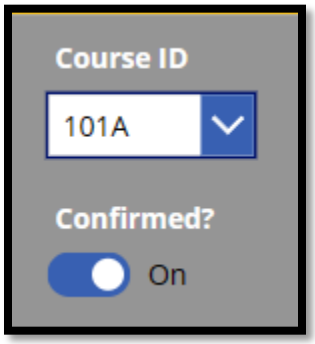
18. Select the drop down you just created and change its **Items** property formula to read the following: **["101A","202B","303C","404D","505E","606F","707G","808H"]**. Now is a good time to change the name of this drop down in **Tree View** to something more appropriate.



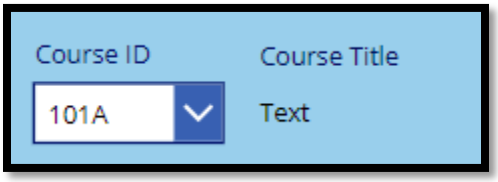
19. Now activate the original **Course ID** input box (data card) and change its **Default** property formula to read the following: **CourseID_DropDown.Selected.Value**. This will pull in the value from the drop-down box that is selected by the user. Then change the **Visible** property formula of this input box to false.



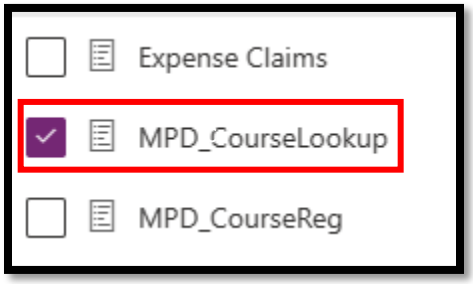
20. Reposition the **Course ID** dropdown box to lay over the original **Course ID** input box.



21. From **Tree View**, select the **Course ID** data card and then insert a **Text Label**. Change the text to read **Course Title**. Repeat the process and insert another label, leaving the title as is.

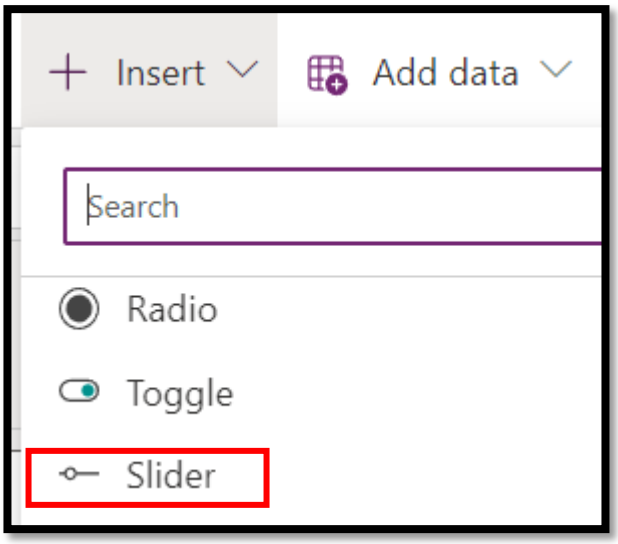


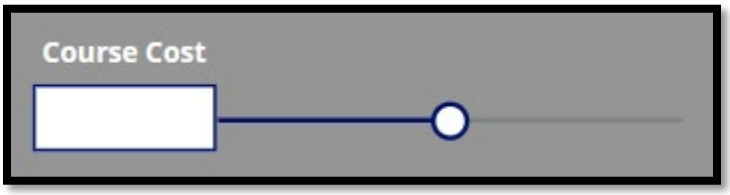
22. From the **Menu** bar on the left side of your screen, click the **Data** button and add the **MPD_CourseLookup** SharePoint list to your app.



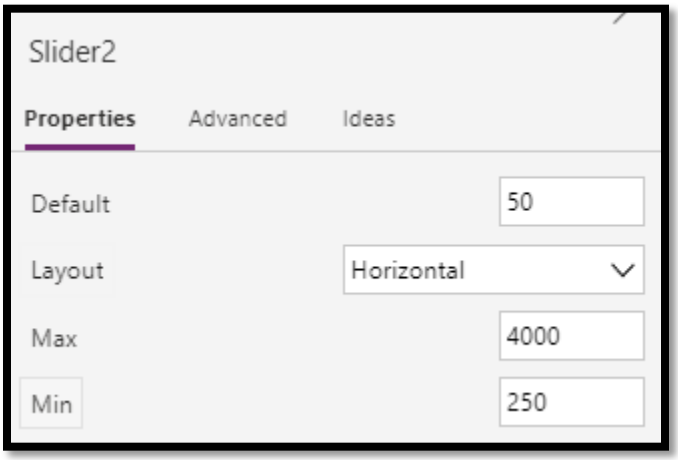
23. Return to the **Course ID Data Card** and select the second label that you created and left as Text. Change the Text property formula to read the following: **LookUp(MPD_CourseLookup,'Title' = CourseID_DropDown.Selected.Value,'Course Title').** Now when you select the **Course ID** from the drop-down box it should lookup and return the **Course Title**.

24. Without having any data card selected from the insert menu choose **Insert > Input > Slider** and position it someplace beside the **Course Cost** input box.

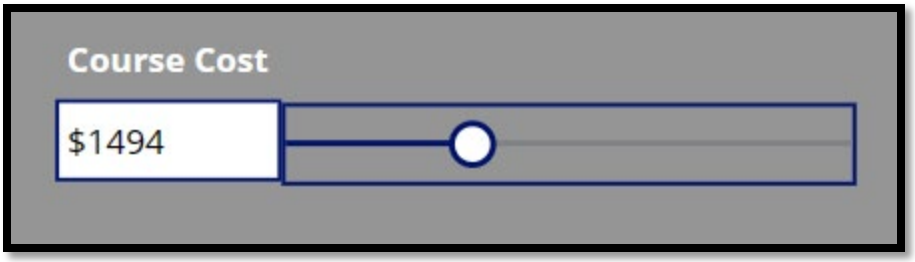




25. Select the **Slider** and in its properties, pane adjust the max to **4000** and the min to **250.**



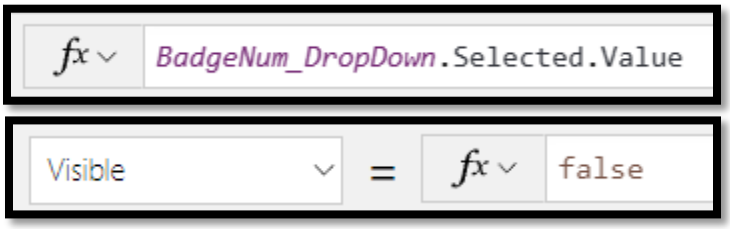
26. Activate and unlock the **Course Cost** data card input box and then change the **Default** property formula to read the following: **Concatenate("$",Slider1.Value)**.



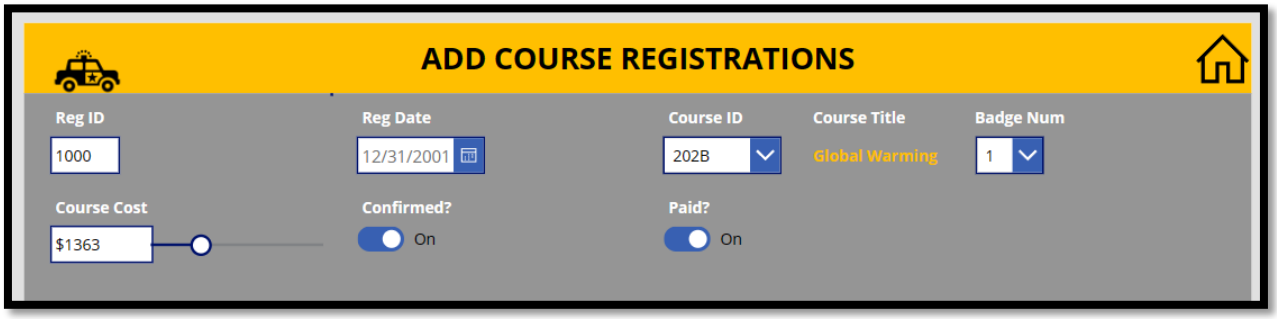
27. Select the **Badge Num** data card group and insert another drop down into it and adjust the input boxes accordingly. Then via Tree View, rename the drop down as well to **BadgeNum_DropDown**.

28. Select the newly created drop down and change its **Items** property formula to read the following: **SortByColumns(Distinct(CPS_Course_Registrations,'Badge Num'),"Value")**.

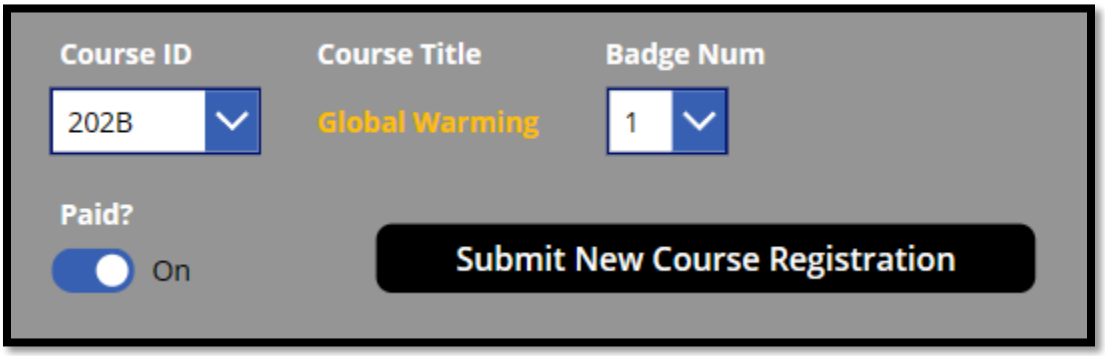
29. Now select the original **Badge Num** input box and change the **Default** property formula to read: **BadgeNum_DropDown.Selected.Value**. Then change the **Visible** property formula to **False** and move the drop down over top of the invisible badge num text box.



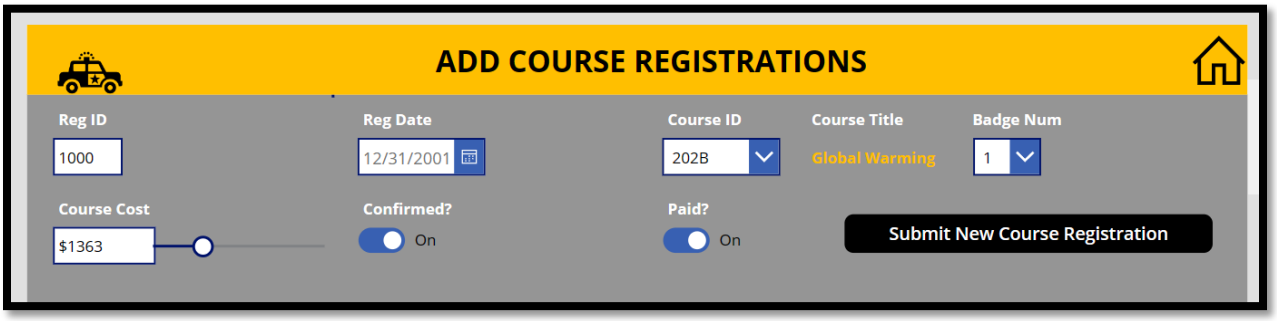
30. Change the columns of the form from 3 to 4 via the properties pane and then adjust the size of the input boxes as you see below and delete the hours/minutes from reg date.



31. Finally create a new button that says **Submit New Course Registration** as seen below and change its **OnSelect** property formula to read the following: **SubmitForm(AddCourseForm);Navigate(MainMenu).**



32. Save your work and play your app from the main menu and create and submit a new course registration.



INDEX

A

Advanced Panes · 24

B

Buttons · 76

C

Call Functionality · 101
Canvas App · 26
Charts & Graphs · 141

D

Data Cards · 36
Data Sources · 9
Data Tables · 98
Data Validation · 93
Dataverse · 5, 27
Deleting Records · 126

E

Environment · 8

F

Filter · 89
Form Galleries · 81
Forms · 36

G

Galleries · 78
Gallery · 31
Grouping · 122

I

Input Elements · 129

L

Launch() · 120
Lookup Function · 100

M

Master Screens · 72
Media Elements · 136
Model-driven App · 26
Multiple Data Sources · 95

N

Navigating · 46
Navigation Features · 84

O

Object Library · 74
OneDrive · 9

P

Popups · 127
Portal App · 27
PowerApps · 5
Publishing · 27

S

Saving · 27
Search · 89
Search & Filter · 89
SharePoint List · 14
Smart Phone · 47
SQL Cloud · 18

T

Today() / Now () · 119
Tree View · 23
Triggers · 45
Type of Applications · 25

U

User() · 119