CONTENTS

Pre-course setup of Office 365 Accounts

Before we begin, we will make sure that we can sign into Office 365 with the account that we are going to use during this course.

On most course deliveries your instructor will provide the files we use in the exercises. These files are also available on GitHub at https://github.com/QA365-co-uk/QAPAESS

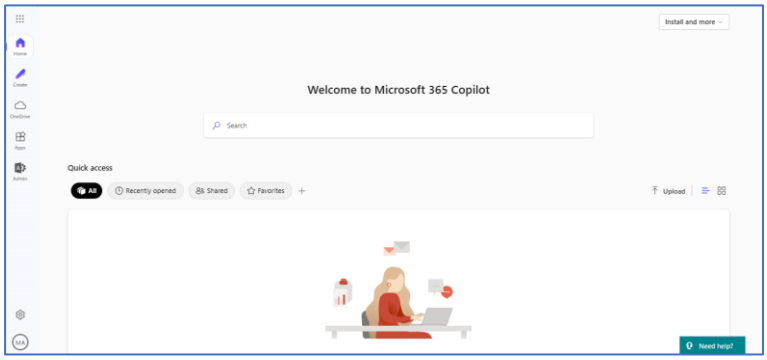
Exercise 1: Sign into Office 365

1. In the lab environment, open the supported browser of your choice (we recommend Chrome or Edge).
2. Navigate to https://m365.cloud.microsoft.

3. Select **Sign in**, then sign in with the username and password provided by your instructor.

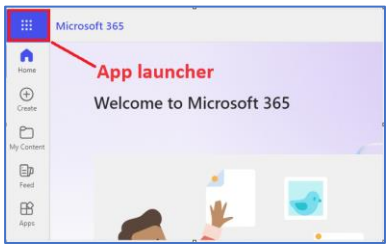
Note: For all further labs – these credentials are referred to as your **Maker Account.**

4. Close any first run experiences / welcome panes.
5. You will then be taken to the Microsoft 365 home page.



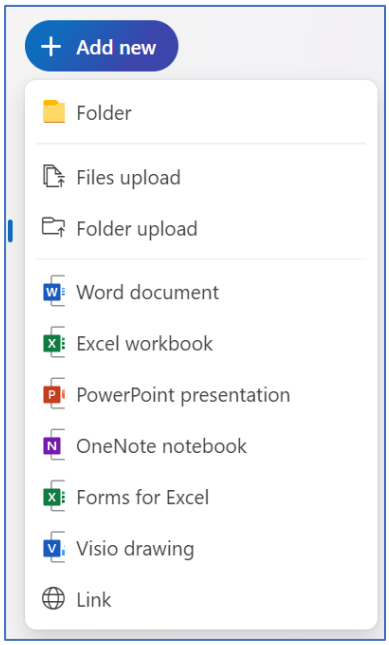
6. Using the App launcher, open **OneDrive**.

Note: Although it says “OneDrive” it is in fact “OneDrive for Business” when we access it through Power Apps, we need to ensure that we choose the correct connector – i.e. “OneDrive for Business”.



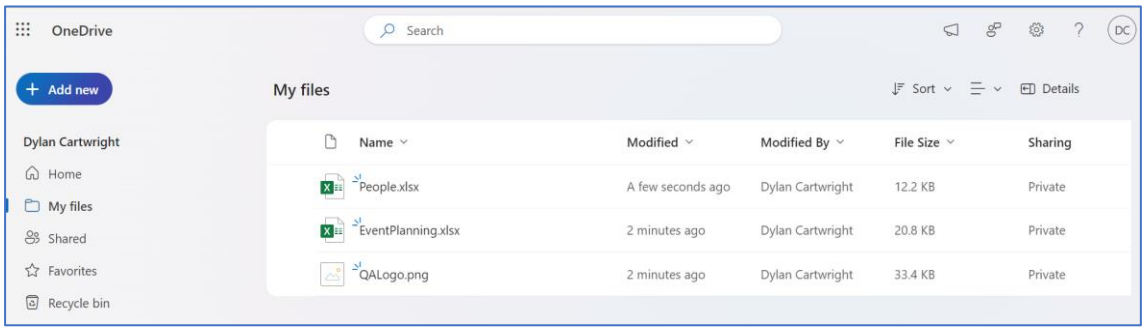
7. Once OneDrive opens, select **My files** in the menu pane.

8. Select the **Add new** blue button above the menu bar and then select **Files upload**.



9. Select all the student files and then select Open.

10. The student files will be uploaded to your OneDrive for Business library.



End of Exercise

Module 1: Power Apps introduction

Power Apps is a suite of tools that provide a rapid application development environment to build custom apps for your business needs.

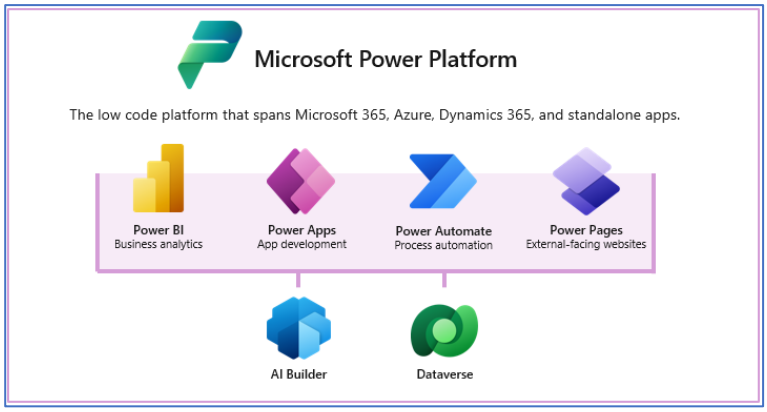
Using Power Apps, you can quickly build custom apps that connect to your business data, stored either in the underlying data platform (Dataverse) or in various online and on-premises data sources. This includes SharePoint, Excel, Microsoft 365, SQL Server, and many more.

Apps built using Power Apps provide rich business logic and workflow capabilities, transforming your manual steps into digital, automated processes. Additionally, these apps have a responsive design and can run seamlessly in browser or on mobile devices (phone or tablet).

Power Apps 'democratises' the custom business app building experience, by enabling users to build feature-rich, custom business apps without writing code.

Power Apps also provides an extensible platform that lets pro developers programmatically interact with data and metadata, apply business logic, create custom connectors, and integrate with external data.

The Power Platform



Power Apps is part of the suite of software Microsoft call the Power Platform. The other components are Power Automate, Power BI, and Power Pages. They all provide a rapid development environment aimed at non-programmers and often solutions will require elements from more than one part of the Power Platform.

The Power Platform enables you to develop applications that allow users to:

- **Act** – Power Apps
- **Automate** – Power Automate
- **Analyse** – Power BI
- **Respond** – Copilots (Previously Power Virtual Agents)

Types of users

Power Apps will typically be deployed in environments where there are different categories of users:

- **Power Apps end-users**: can run apps that have been created for and shared with them. The apps will typically run via a web browser or on a mobile device, such as a smart phone or tablet.

- **Citizen Developers**. These are users that create custom business apps without writing code. They are also referred to as 'App Makers' or 'Power Users'.

- **Power Platform administrators**. These administrators use the Power Platform admin center to secure data, manage environments, and monitor usage of Power Apps as well as the rest of the Power Platform.

- **Pro developers**. Full time developers who will use a range of tools to produce applications and solutions for the business. Power Apps is one of the tools they may choose to use.

In this course we will be focusing on how the 'Citizen Developers' in your organisation use Power Apps.

Environments

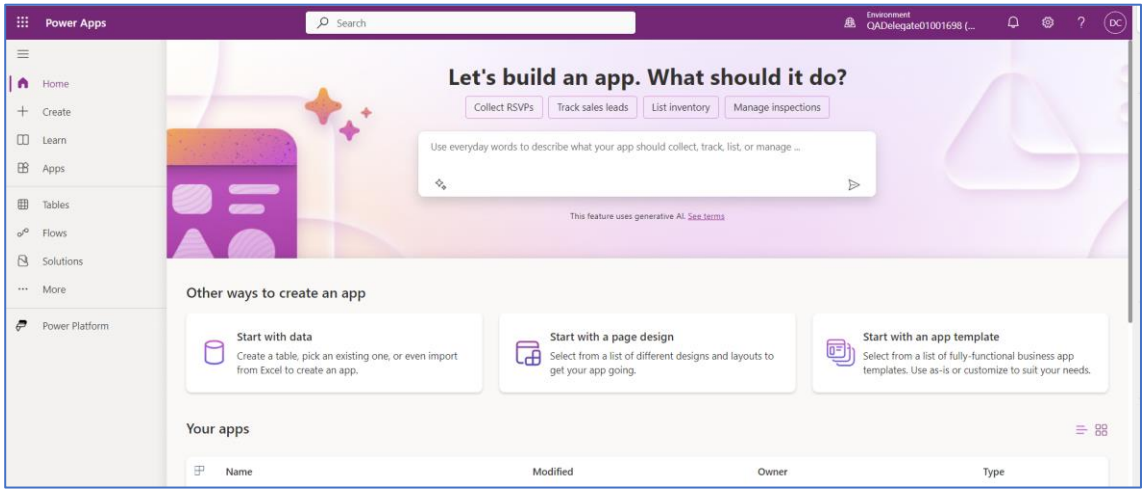
An Environment is a container used by the Power Platform to contain its various components, including apps, flows, connectors and, if required, a Dataverse database. Components in one environment are isolated from components in other environments.

Organisations may deploy multiple environments when they need to provide separation, for example between development, testing and production or between different business units that require separation for other reasons. To run multiple environments, you need to have a licensing plan that supports this.

Your organisation will control which environment you will use to create your apps. On this course, you will use a Developer Environment which will be created for you automatically when you create your first app.

Module 2: The Power Apps Studio

The Power Apps Studio is a web-based environment that can be used to create, manage, run, and develop your Power Apps. It can be accessed either by using the Power Apps icon on the Office 365 home page or by navigating to https://make.powerapps.com.



Creating apps

When using the Power Apps studio, there are several ways to create a new canvas Power App.

Using Copilot

With the Copilot prompt option, you can describe the purpose of your app using everyday words. Copilot will suggest a data structure which you can fine tune (again using everyday words). From this data structure an app will be built.

Start with data

The 'Start with data' experience gives the user three options. All these options will build an app using the modern design style.

- **Create new tables** – This option gives you four additional choices. You can use Copilot to describe the table to be built, you can import a list from SharePoint, you can import a list from an Excel or .CSV file and finally you can start with a blank table that you can then edit.
- **Select existing tables –** An app will then be generated from the table you select.
- **Connect external data –** This option will allow you to select existing data either stored in SharePoint, in an Excel file stored in One Drive for Business or in SQL server. An app will then be generated from the table. Start from data (Classic experience)

Start with a page design

This option allows you access to several choices.

- **Various page templates** – Common blank page layouts are listed and can be selected.
- **Blank canvas** – You can choose between Tablet and Phone layout.
- **An image or Figma file** – see below
- **Dashboard** – This option will create a model-driven application.

From an Image

An image of the required layout of an app can be used to create the basic app. This can be hand-drawn if required. Once the app is created it will still need to be connected to the data sources.

From a Figma file

Figma is a web-based graphics editing and user interface design application. It allows teams to design, prototype, develop, and collect feedback on products. A UI designed in Figma can be uploaded and used to create a Power App.

From Blank

Choosing Canvas App from Blank will literally give you a blank screen onto which you can place whatever elements you wish. When picking this option, you can choose between a responsive, tablet and a phone format (layout), but then have complete control over the app. This option gives the app-maker the most flexibility, but also requires a higher level of skill and knowledge than the other options.

From a template

There are several templates provided to create with Power Apps. Apps can be made very easily and can provide some reasonably complex solutions with minimal effort. Like all Canvas Apps, once created, they can be customised and edited but, due to their complexity, a reasonable level of knowledge may be required to edit one of these apps without breaking it.

Navigating Power Apps Studio

Once you create or edit an existing app, studio interface will change to allow you to work on it. This area has several sections that you will work with: Navigation Pane, Edit Pane, Properties Pane, and the Menu / Ribbon area.

Left Navigation Pane

Contains several elements for use within your apps. The default element will be the Tree View, which allows you to see and navigate the various screens and controls that make up your app. The other elements here are Insert, Data, Media, Power Automate, Advanced Tools, and Search.

Canvas Area

The visual workspace wherein the app is built; you can see and manipulate controls on any screen within your app, chosen with the Tree View.

Properties Pane

Can be used to set the most common properties on the various controls and screens used within your app. It will often have controls to make setting the property easier, such as a colour picker for fill and text colours. The control or screen that you are editing can be selected either in the Tree View or directly in the Edit Pane.

Menu / Ribbon area

The Menu / Ribbon area is where you will find various other ways to edit and configure your app and the elements that it comprises of. It also has a properties editor that will allow you to configure any edible property of a screen or control, although without the benefit of the controls found in the properties pane (e.g., colour pickers).

Copilot Pane

The Copilot pane can be used to make changes to your app using natural language prompts. Use of Copilot in Power Apps is covered in more details in Appendix B.

Running apps

Once created and published, apps created with Power Apps can be launched in various ways. Any user that navigates to the Power Apps studio will see a list of published apps that have been shared with them (as well as any apps that they created) and they can run them from there.

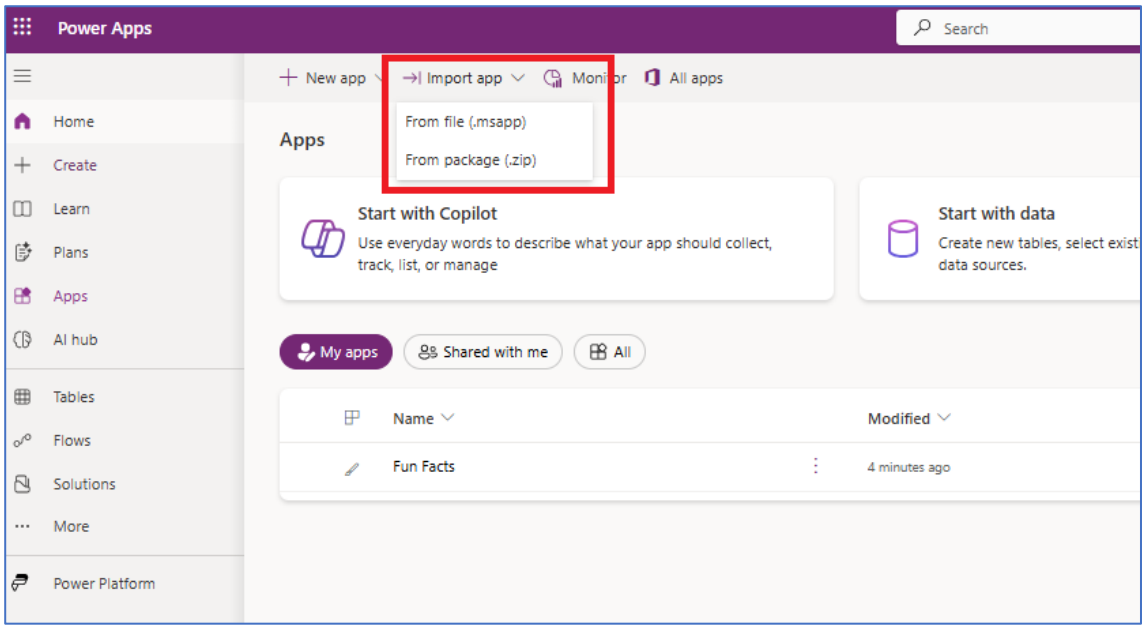
If they install the Power Apps mobile app on their phone or tablet, they will also see a list of apps they can run. There is also a Power Apps app for Windows available from the Microsoft Store.

Finally, each app has a unique URL, and the app can be run by navigating to that URL, either directly, or by embedding the URL into another entity such as adding it as a tab in a Microsoft Team channel. The app URL can be found on the app Details page.

Importing apps

Whilst Power Apps are usually stored in the cloud, it is possible to download an off-line copy of your app. This can either be in the format of a package (.zip file) or a Power Apps file (.msapp file).

The process for importing these is the same, navigate to the Apps section of Power Apps studio, and then select Import App from the navigation bar. You will then be presented with a choice between importing an .msapp file or a package.

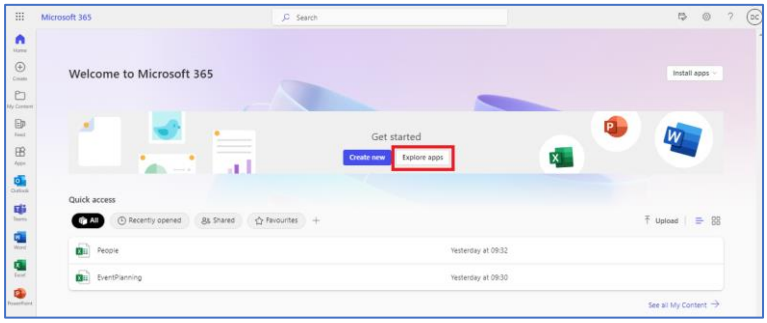


Once the app is imported, you should save it.

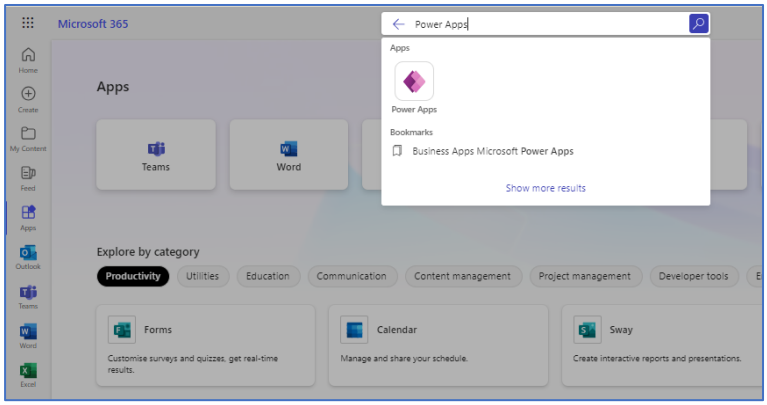
Exercise 2: Using the Power Apps Studio and running apps.

Task 1 – Launching the Power Apps Studio

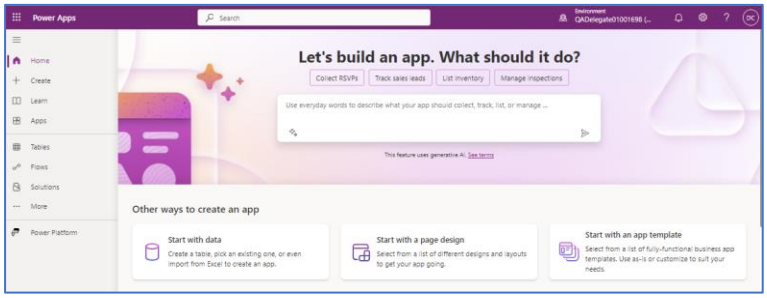
1. While signed in to Office 365 as your Maker account, select Explore apps.



2. Search for Power Apps.



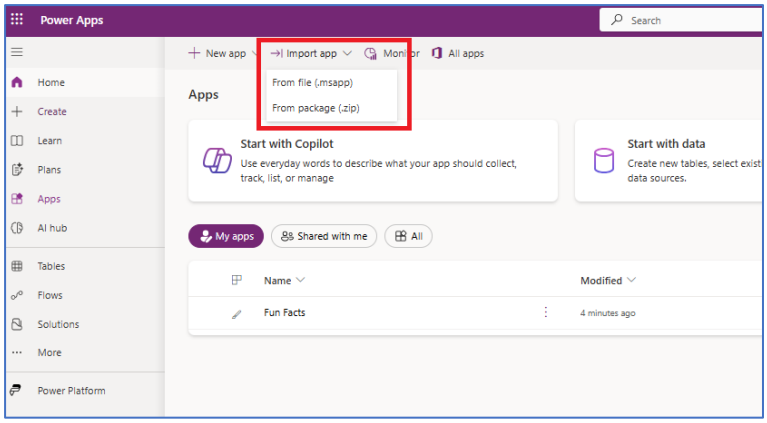
3. Launch Power Apps – a new tab will be created. The Power Apps studio will launch.



4. Close any pop-up messages that may appear.

Task 2 – Import an app

1. In the Power Apps studio, ensure that you are on the “Apps” page and then select the **Import app** drop down. Then select **From File (.msapp)**.

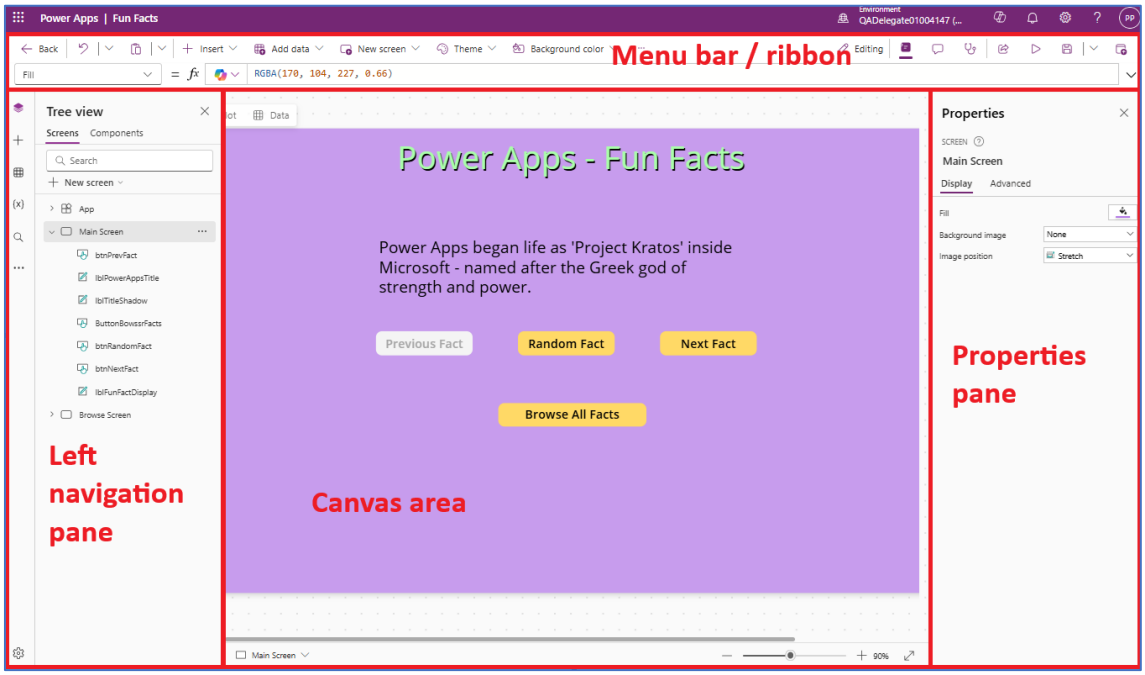


2. Browse for, and then select the "**Fun Facts.msapp**" file from the Lab Files folder. Select **Open**.

3. If a Welcome to Power Apps Studio pop-up appears then select Skip.

Task 3 – Exploring the Power Apps design environment

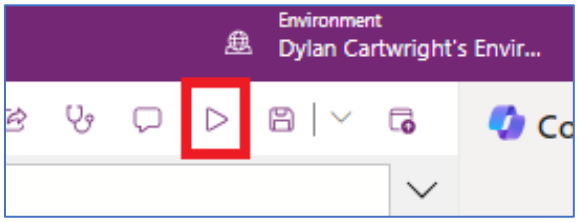
1. In the Power Apps design environment, make sure that you can identify the main areas that you will be working in: The **Left Navigation Pane** (**Tree View**), the **Canvas Area** (a.k.a. the **Edit Pane**), the **Properties Pane,** the **CoPilot Pane,** and the **Menu** Bar / **Ribbon** area. Note – the Copilot pane will probably be closed.



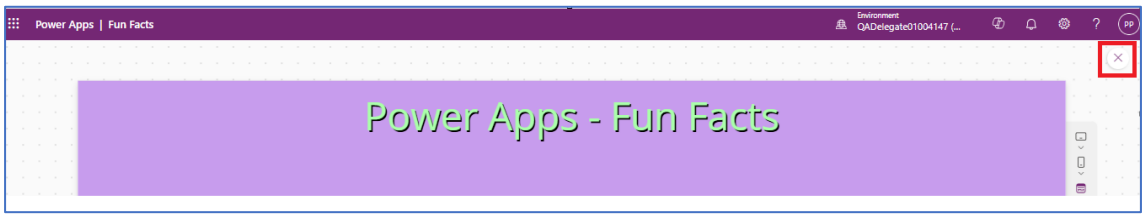
2. Notice that this app just two screens: Main Screen and Browse Screen.

Task 4 – Running an app

1. Click the Preview App button in the menu ribbon.



2. Use the buttons and other controls to interact with the app.
3. When you are done, exit Preview Mode by selecting the X in the top right corner.

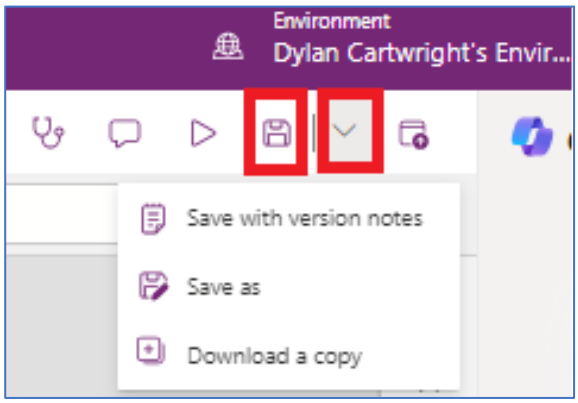


End of Exercise

Saving apps

Your app can be saved using the Save icon (and drop down) on the right of the ribbon bar. When clicked directly, the icon will save the app. When used as a drop-down menu, the following options are available:

- Save with version notes.
- Save as.
- Download a copy.



Version notes can be seen when you are looking at an app's version history.

The Save as option will allow you to save the app with a different name.

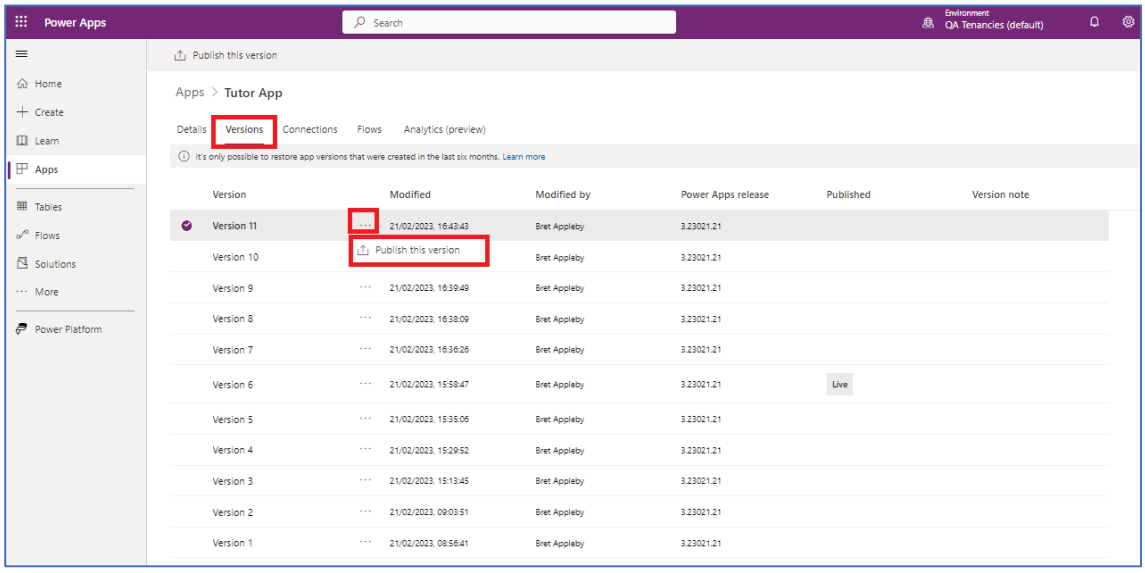
Download a copy will download a file with the extension: '.msapp'. This file can then be opened with the Power Apps Studio and is one way to transfer apps from one Environment or Tenant to another, but exporting is generally better.

Once the app has been saved once, auto save will be in effect (unless disabled in the settings). The save button is disabled when there are no changes since the last save (manual or automatic).

Publishing apps

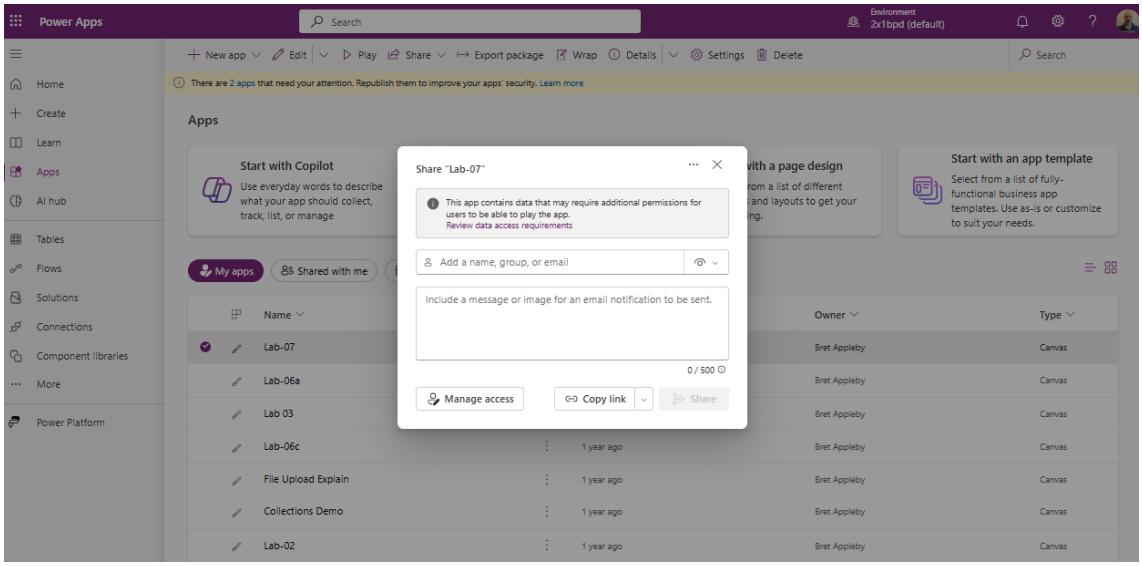
When you save an app for the first time, it will automatically be published. Once you make and save changes to your app, you must publish those changes before they are visible to the users you share the app with.

After you have saved your app, you may be presented with an option to publish. If you choose not to publish at that time, you can publish the latest version of the app from the versions tab of the app settings page or by using the Publish icon.



Sharing apps

Before an app can be used by anyone else, the maker must share it with them. Sharing can be performed by selecting an app in the Power Apps Studio and then clicking the Share icon that appears in the ribbon.



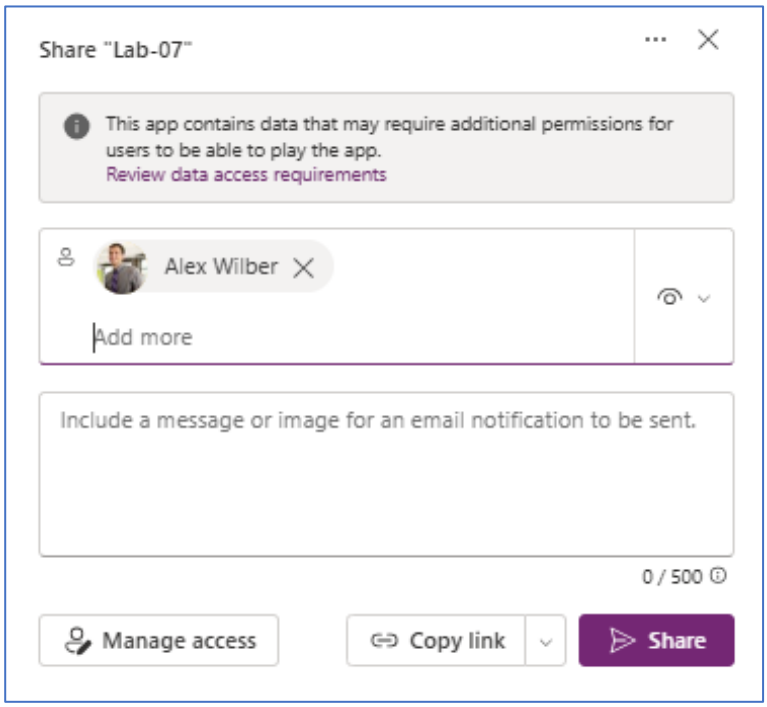
An app can also be shared when you are in the development area by selecting the Share icon – this opens a new tab back to the Apps section shown above. The first time you publish an app, you will also be presented with a button to share it and once again it will open a new tab.

To share an app, type the name or partial name of a user or group that you wish to share the app with, then select their name card from the options presented to you.

Once you have selected the user, you then get an option to make them a Co-owner, in addition to just being able to run the app. A Co-owner can edit, publish, and work with versions, just like the initial maker can.

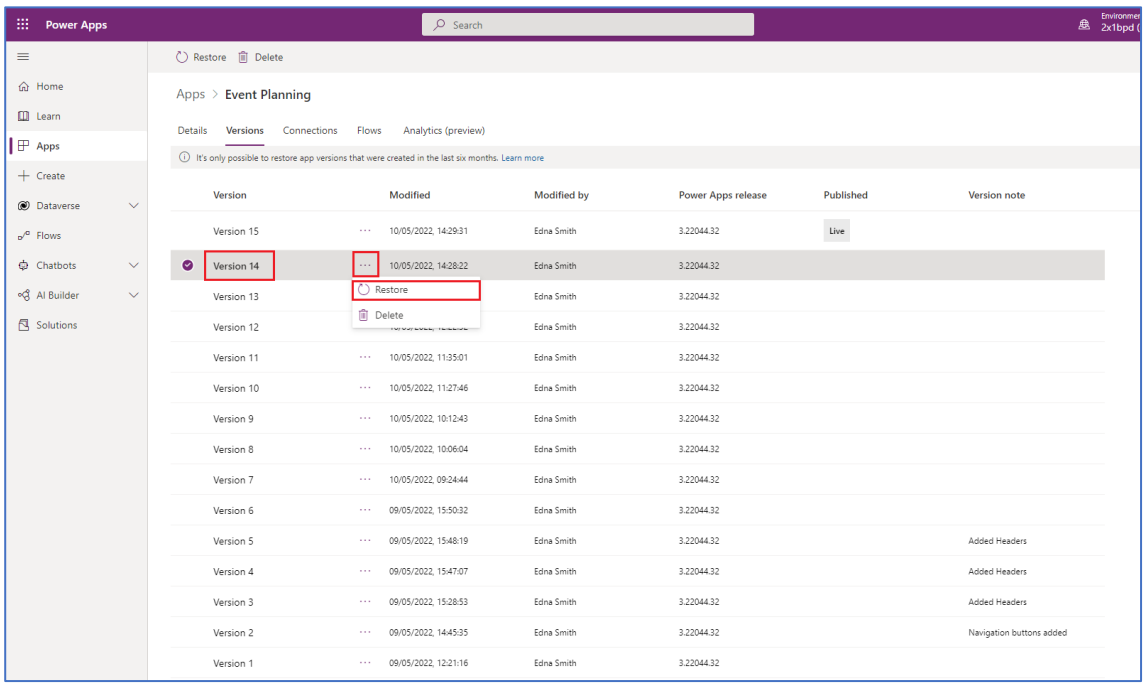
This sharing will only share the app. Any user running the app will also need access to the data that the app uses. Depending on how the data connections are configured,

this may require additional steps to be completed before the app can be used correctly by others.



Versions

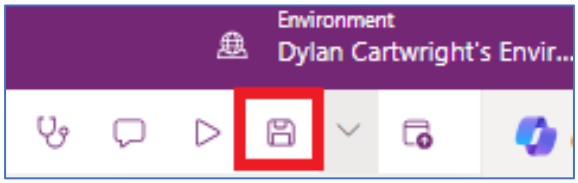
Each time you save an app, a new version is created. These versions can be seen from the versions tab when you look at the details of an app. As well as publishing the latest version from here, you also have the option to restore a previous version of the app. Restoring a previous version makes a new copy of the version you restored. That way, the highest version number will always be the one to work with.



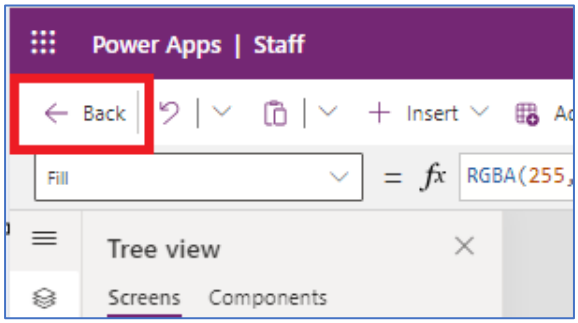
Exercise 3 – Saving and running an app

Task 1 – Save the app

1. Select the Save icon.



2. Once the app is saved, select the Back control to exit the design environment.

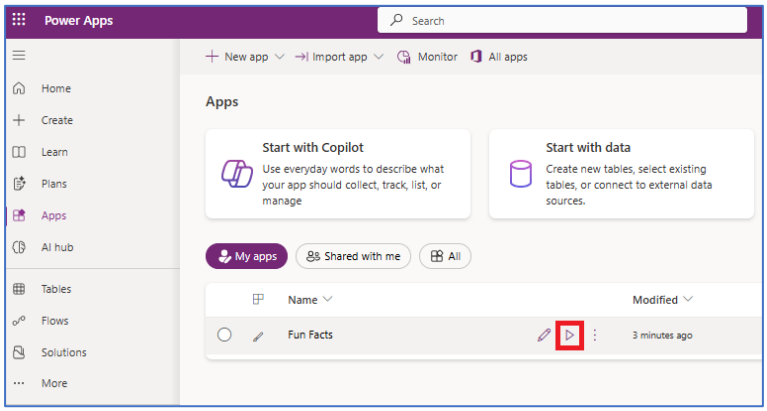


3. When prompted, select **Leave**.

4. You should be returned to the Home page, and you should see your app "Fun Facts" listed under My apps.

Task 2 – Run the app

1. Select the run icon beside the app you just saved.



2. The app should now run (with a user experience) in a new browser tab. When you are happy close this tab.

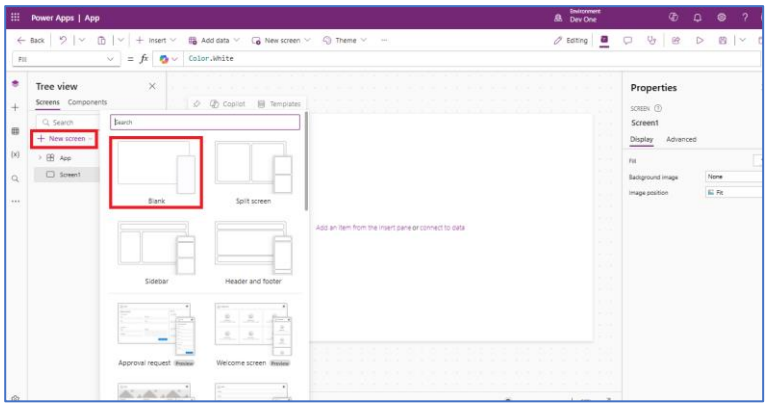
End of Exercise

Module 3: Screens, controls, and navigation

Screens

A screen is a canvas onto which you can place controls to provide the functionality that you desire in your app. Most apps will consist of multiple screens, with only one being visible to the user at a time. The app will need to have navigation commands added to it to allow a user to move from one screen to another.

When you add a screen, you can choose a blank screen or a template with pre-populated controls. You can add a screen with the 'New screen' drop-down in the Tree view.



The first screen is the one that will be initially shown when your users run the app. The 'more' icon can be used to move a screen up or down the list in the Tree View.

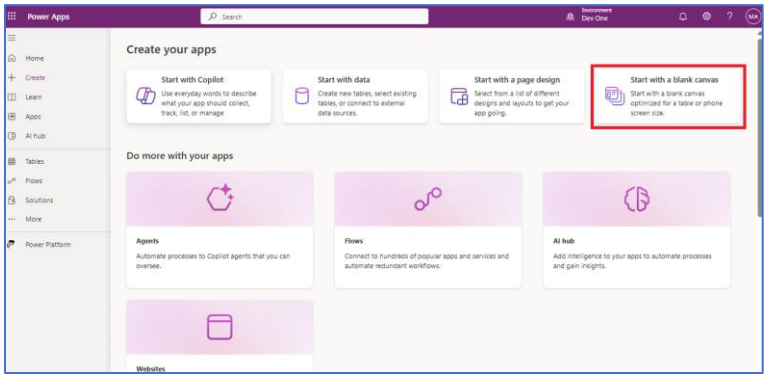
In the Tree View, a screen is considered a container for the controls placed on that screen, and as such, screens can be minimised and maximised to help navigate the tree and find the controls you need.

Exercise 4: Creating and sharing a simple app

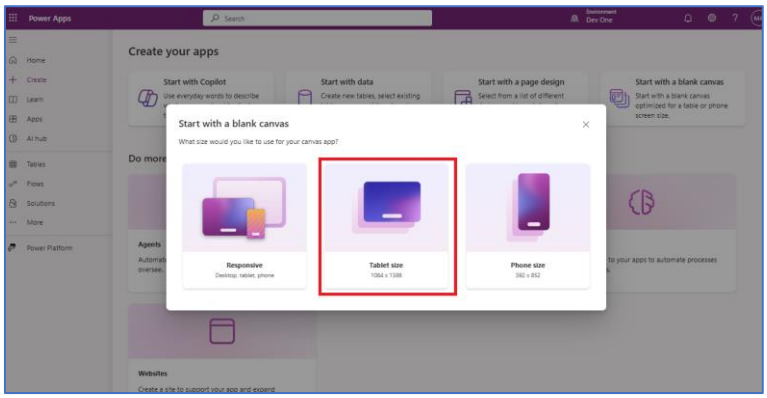
Note: In this exercise you will share your app with another user. Your instructor will give details for you to use for this. In these notes, that user will be referred to as your **Test User Account**.

Task 1 – Create a Canvas App from blank with multiple screens

1. In the Power Apps studio, click **Create** in the left navigation pane, then then click **Start with a Blank canvas**.

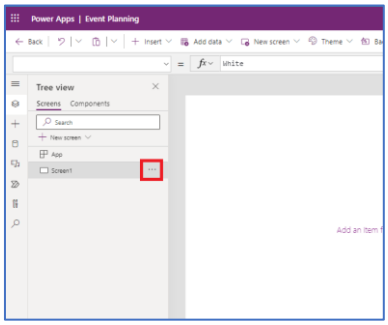


2. In the "Start with a blank canvas" dialog that appears, select **Tablet size**.



3. If it appears, **Close** (or Skip) the Welcome to Power Apps Studio dialog.

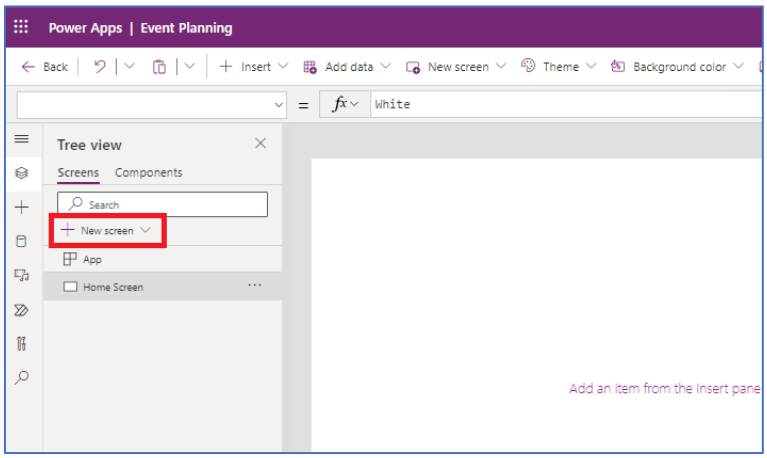
4. Click on the more icon beside Screen1 in the left navigation pane (you may need to move you mouse over the Screen1 entry before the icon appears).



5. In the menu that appears, select **Rename**.

6. Rename the screen as **Home Screen**.

7. In the Tree view pane click **New Screen** (if you cannot see this, make sure that you have the Tree view selected).

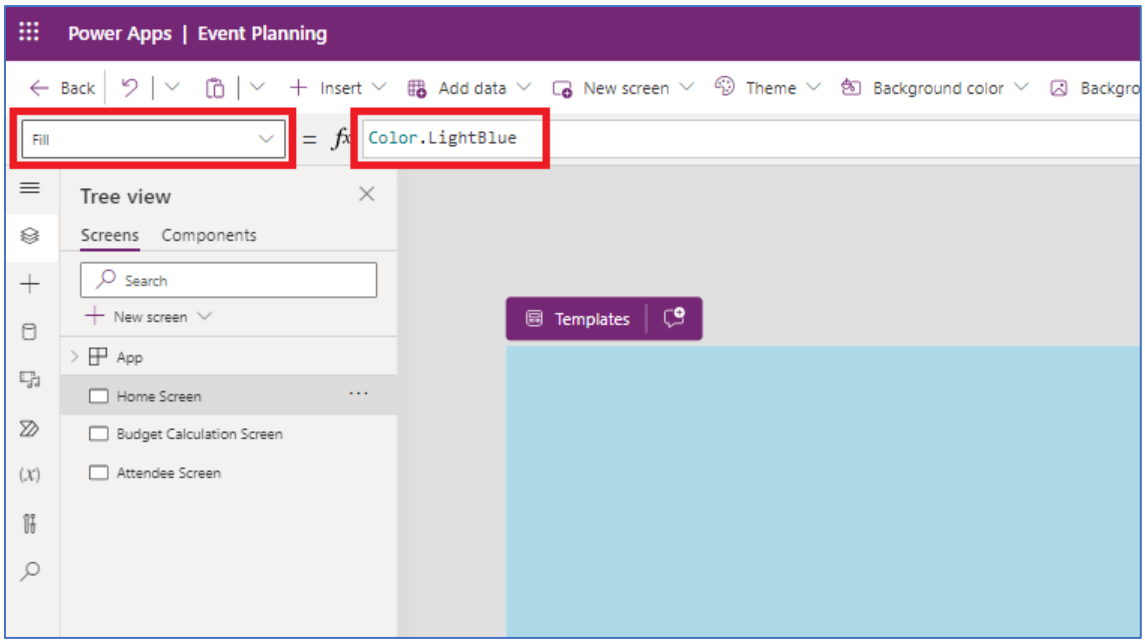


8. Click **Blank**.

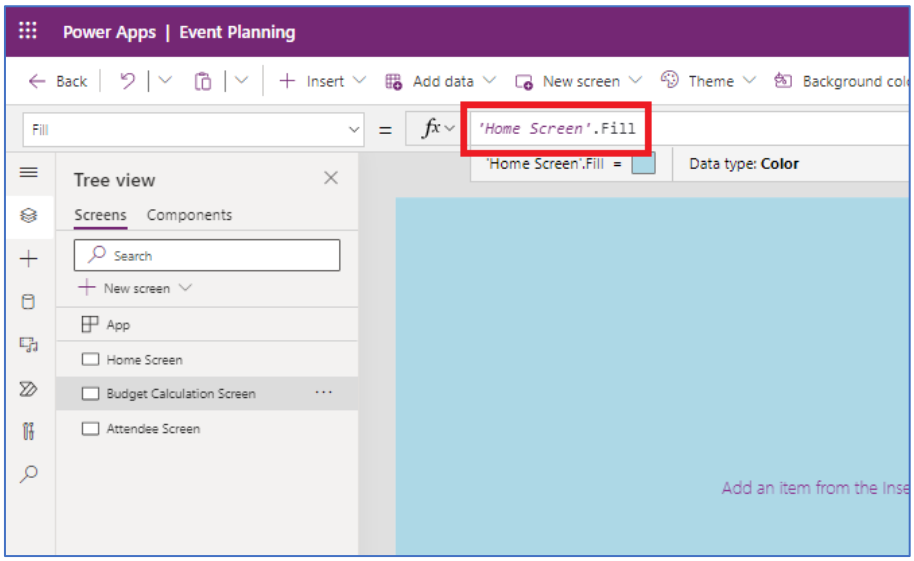
9. A new screen will appear in the left navigation with the name of Screen2 – use the method from above to rename it **Budget Calculation Screen**.

10. Add a third (blank) screen and give it the name **Attendee Screen** (note you can double click on a screen name in the Tree View to rename screens).

11. Using the left navigation, select the **Home Screen** and then using the ribbon area set the **Fill** property to **Color.LightBlue**.



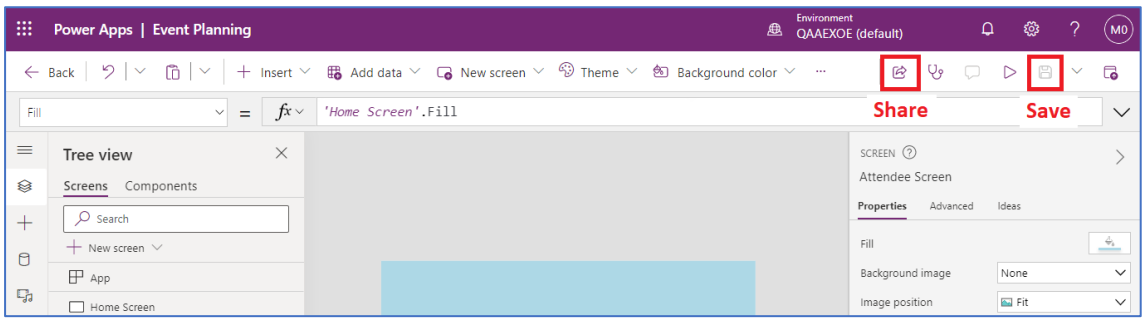
12. To maintain continuity in our app, we will use a relative fill colour for the other screens. Select the Budget Calculation screen and set its Fill property to **'Home Screen'.Fill**.



13. **Repeat** the previous step for the **Attendee Screen**.

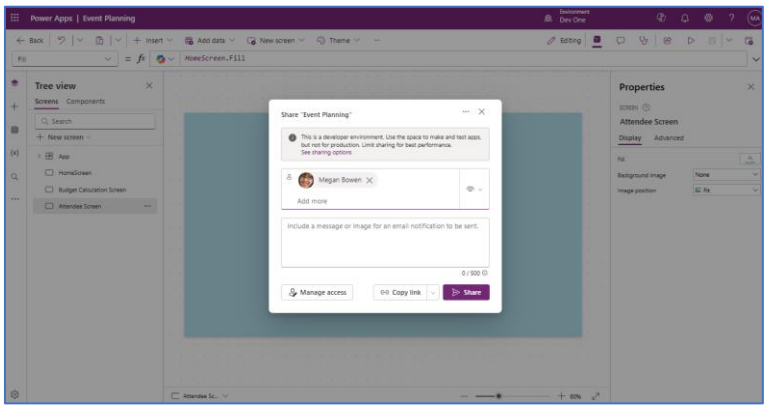
Task 2 – Save, publish, and share your app

1. In the Power Apps Studio, click the **Save** icon.

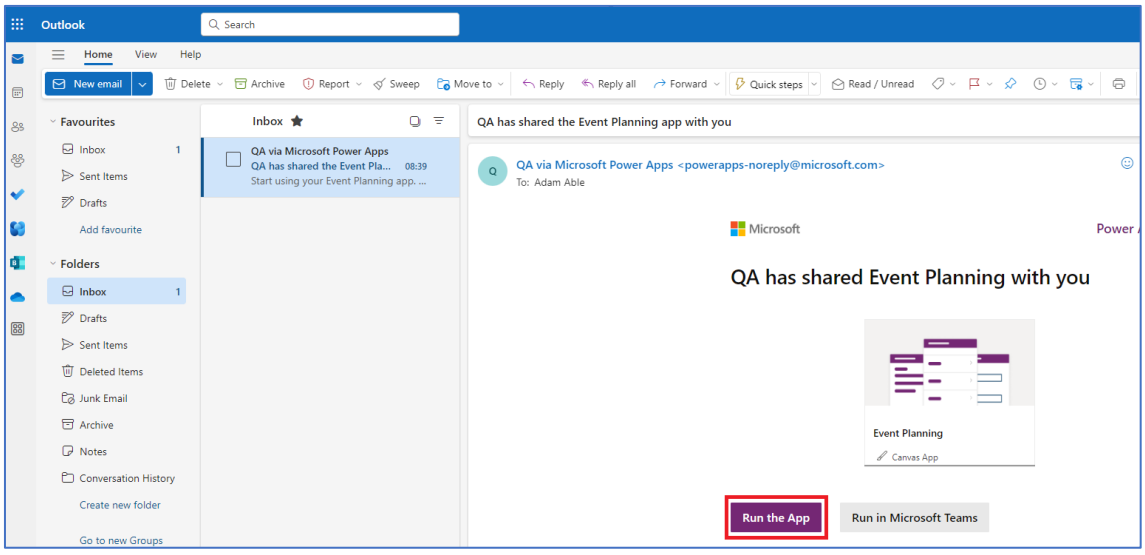


2. Enter the name **Event Planning** and then select **Save**.
3. When the app is saved, click the **Share** button.

4. In the Enter a name... field type User and then select your Test User account.



5. Note the option to also make a co-owner if you wish, but do not select this.
6. Click **Share**.
7. Close the sharing summary pop-up.
8. Open a new browser profile for your user account. (Incognito, InPrivate or additional profile that you have set up).
9. Navigate to https://office.com and then sign-in as your **test user.**
10. Click the Outlook icon.
11. Locate the email from your maker account and then click the **Run the app** button.

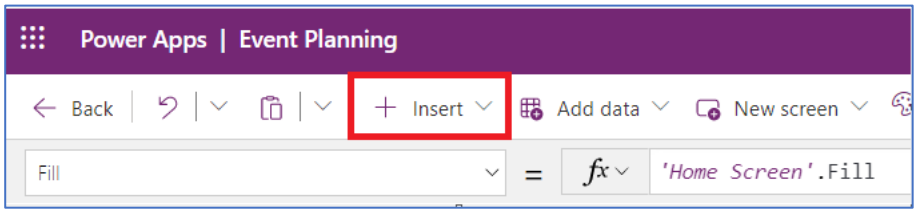


12. The **Power App** will now launch.
13. As we have not added any navigation controls you will only be able to view the Main Menu screen.
14. Close the whole Chrome Profile instance and return to the Maker account.

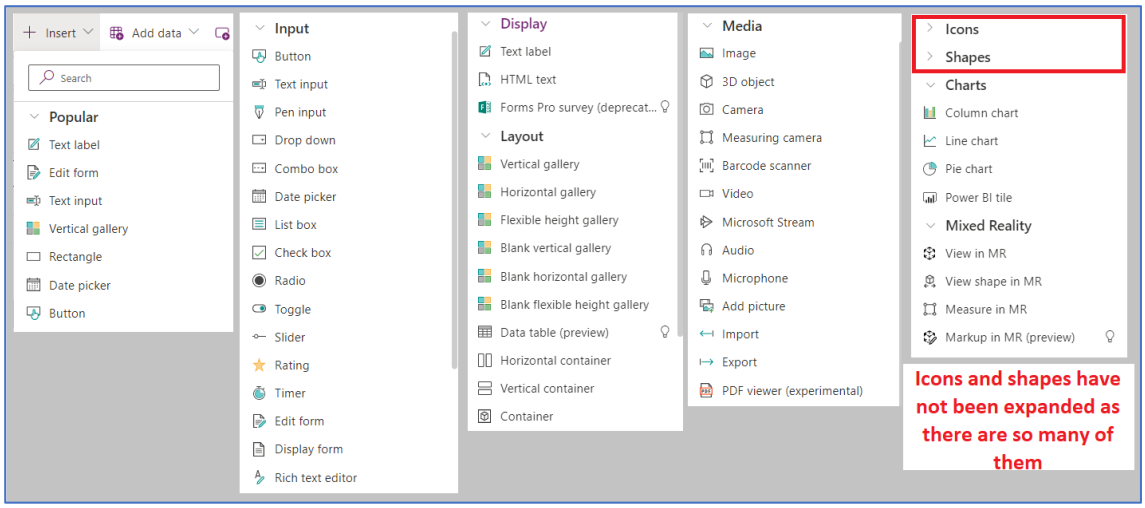
End of Exercise

Controls

Controls are the basic building blocks you use to make your app. Controls are placed onto screens and then configurated by setting their properties.



Each type of control has a different set of properties. Some properties, such as Height and Width, are common to almost every type of control, but other properties, such as CheckboxSize, are specific to one type of control.



Full details about the various controls available in Power Apps can be found at https://docs.microsoft.com/en-us/powerapps/maker/canvas-apps/reference-properties.

Naming conventions

Having a sensible naming convention for your screens, controls, and other components makes the design and ongoing maintenance of your apps much easier. Controls and screens are placed with a default name but can be renamed by using the more icon beside the screen or control name.

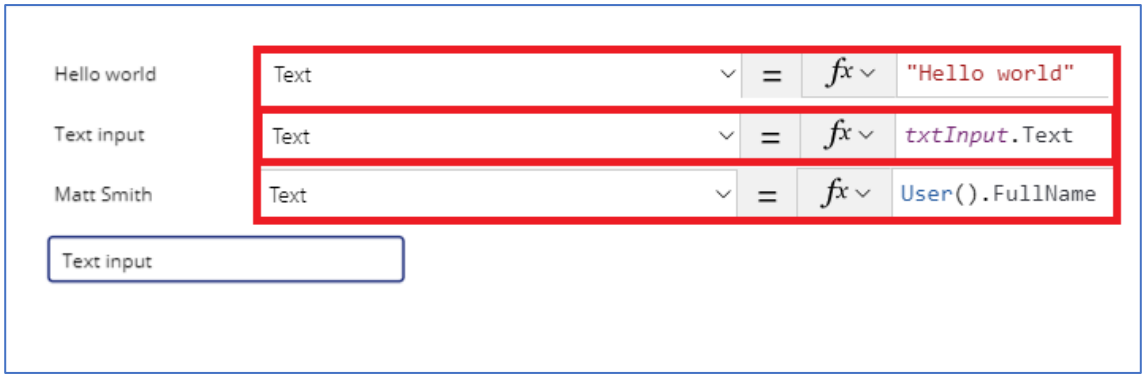
Screens should be configured with descriptive names that will make sense if read aloud by screen readers. The generally accepted convention for controls is to name them starting with a short code indicating the control type and then a descriptive label.

A naming convention that you can implement along with other guidelines can be found in the Power Apps Canvas App Coding Standards and Guidelines which can be found at https://powerapps.microsoft.com/en-us/blog/powerapps-canvas-app-coding-standards-and-guidelines/.

Properties

Configuring properties is the way that we configure controls, thereby providing functionality for our app. Each type of control has a different set of properties. Some properties, such as Height and Width, are common to almost every type of control, but other properties, such as CheckboxSize, are specific to one type of control.

Properties can be literal values, they can refer to properties of other controls, or they can be the result of a function, for example the text property of a label could be "Hello World", txtInput.Text, or User().FullName.



Configuring properties

Properties of screens and controls can be configured in several ways:

Edit Pane

The Edit Pane can be used for some basic configuring of controls, notably size and position, although Text / Default Value can often be set in the Edit Pane.

Properties Pane

The Properties Pane provides access to the most used properties of each control type. This Pane has the added advantage that the most frequently used properties are displayed at the top of the Pane and many settings have interactive control and pickers, such as Colour Pickers, Drop-down lists, and On/Off switches.

Advanced Properties Pane

This view will list all the properties available for the control or screen. It will display the raw data without the benefits of the helpers on the properties pane. The search box can help locate the current property. Where a control can have an expression added, IntelliSense for the code will be displayed.

Properties drop-down in ribbon area

In the ribbon area, there is a drop-down box that will list (in alphabetical order) all the configurable properties for the selected control. This method has the advantage that you can use the built-in intelligence to select property set values (e.g., DisplayMode.Edit) as well as to specify functions.

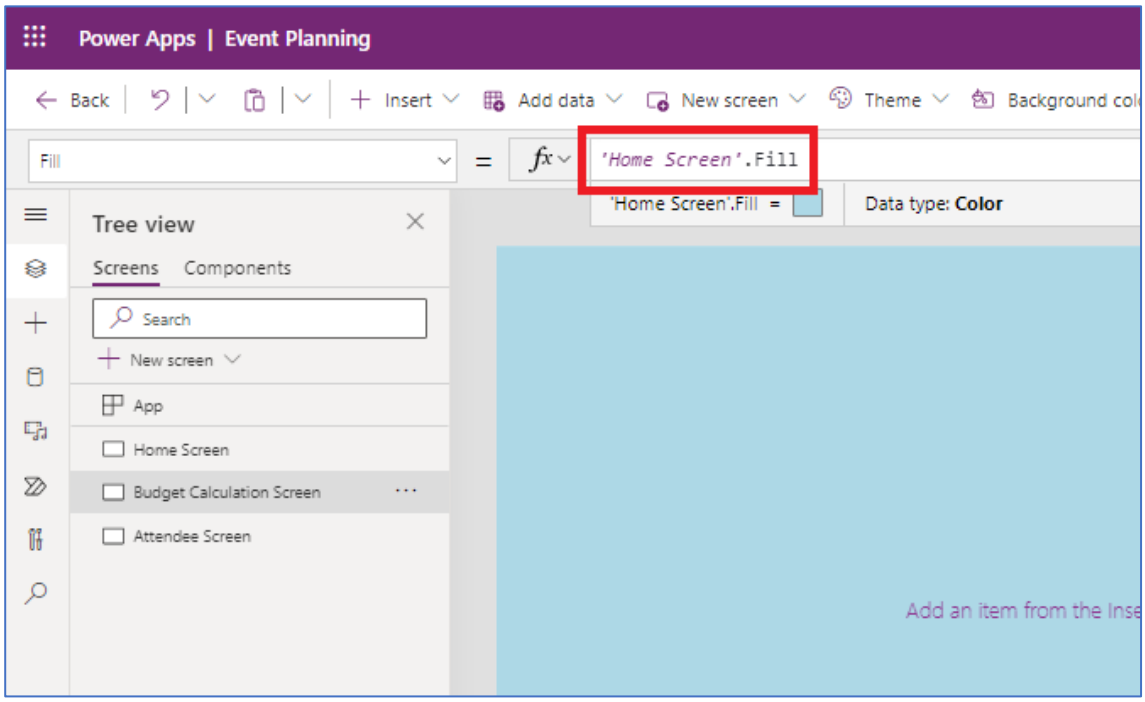
Text Properties and formulas

Care must be taken when entering formulas into the Text properties. If you use the Properties pane to enter this, Power Apps will encapsulate what you type with double quotes. This will change the code into a literal string which will then be displayed. Formulas need to be entered in either the properties drop-down, or by using the advanced properties pane.

Relative / linked properties

To maintain continuity with your app, you may decide to use relative or linked properties. This is where a property of a screen or control uses or derives its value from another screen or control. This ensures that if you alter one object, the others are updated in a sensible way.

For example, buttons performing related tasks may use the same X property and derived Y properties so if the first button is moved all the others will follow, or screens could use a fill property derived from the first screen so if it is changed, the colour of all screens will change.



On... properties

Some properties work like events do in other programming languages; in that they respond to an action occurring when the app runs. For example, a button or icon is pressed, or a screen becomes visible. Against these properties, we can configure declarative functions – these can do something to an external entity, such as setting a variable or navigating to a different screen.

The properties that work in this way start with ‘On’, for example OnSelect or OnVisible. The predefined App object in the Tree view also has an OnStart property.

Note: The App.OnStart property is being deprecated and other options should be used, for example using the OnVisible property of screens.

Module 4: Functions

Introducing functions

Functions are pieces of code within Power Apps that can take parameters, perform an operation, and return a value. They are used to perform tasks to help provide the functionality we want in our app and are set as part of control properties.

Functions in Power Apps use the "Power Fx" language which is modelled after Microsoft Excel functions.

Some functions will be introduced in this course. A complete list and reference for functions can be found at https://learn.microsoft.com/en-us/power-platform/power-fx/formula-reference-canvas-apps

Navigate function

To move between screens, we can use the Navigate option from the Action menu. This provides us with drop-down boxes that can be used to select the parameters for the Navigate function that is set as the control's property.

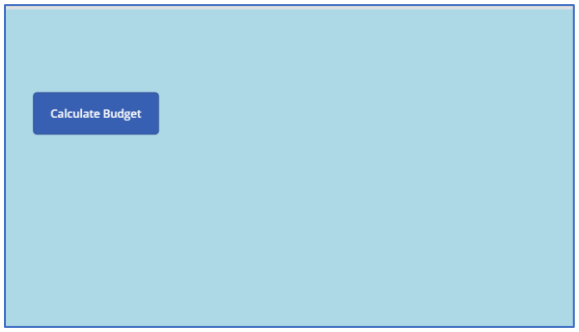
```
Syntax


Navigate(Screen [, Transition [, UpdateContextRecord ] ])


Screen - Required. The screen to display.

Transition - Optional. The visual transition to use between the current screen and the next screen. The default value is None.

UpdateContextRecord - Optional. A record that contains the name of at least one column and a value for each column. This record updates the context variables of the new screen as if passed to the UpdateContext function.


Example

Navigate(Details, ScreenTransition.Fade)
```

Exercise 5: Adding navigation to your app

1. Switch back to your browser tab where you are editing the Power App you created.
2. The Power Apps studio should still be open, with the Event Planning app being edited.
3. Make sure you are on the **Home Screen** and using the **Insert** icon select **Button** to add a button.

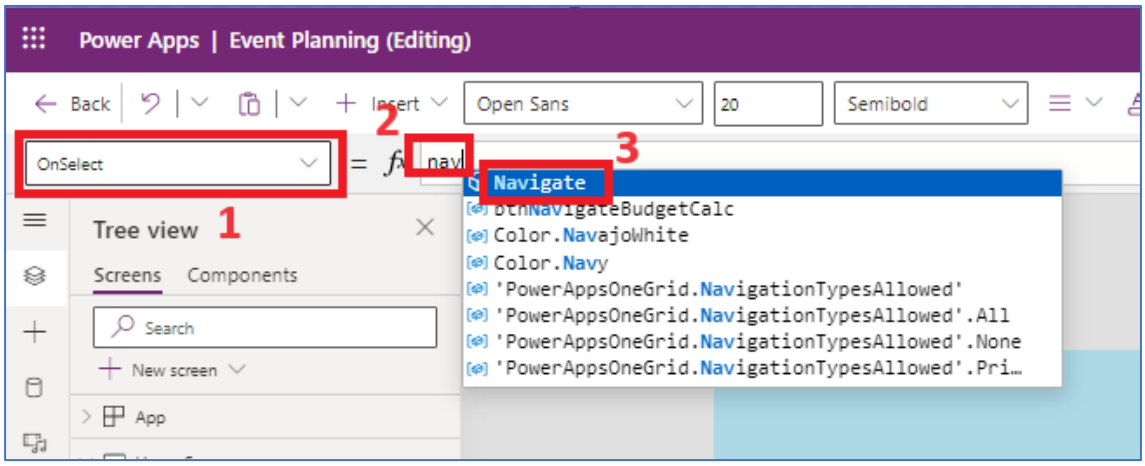
4. Set the following properties:

Property	Value
Name	btnNavigateBudgetCalc
Height	100
Size	20
Text	"Calculate Budget"
Width	300
X	70
Y	200

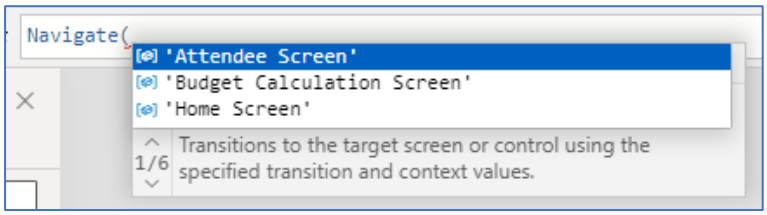


5. Select the **button** and ensure that the properties drop-down shows the OnSelect property. In the value field (beside the fx), remove the existing value of 'false' and type **nav.**

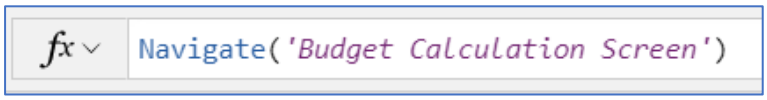
6. The IntelliSense dialog should appear below the input box, and from that select **Navigate**.



7. **Navigate** will now be shown in the expression box. Type an opening parenthesis (Round bracket) and IntelliSense will be indicating that are target is needed. Select **'Budget Calculation Screen'** from the suggestions.



8. Close the parenthesis so the code reads as below.



9. Select the button and then **Copy** it.

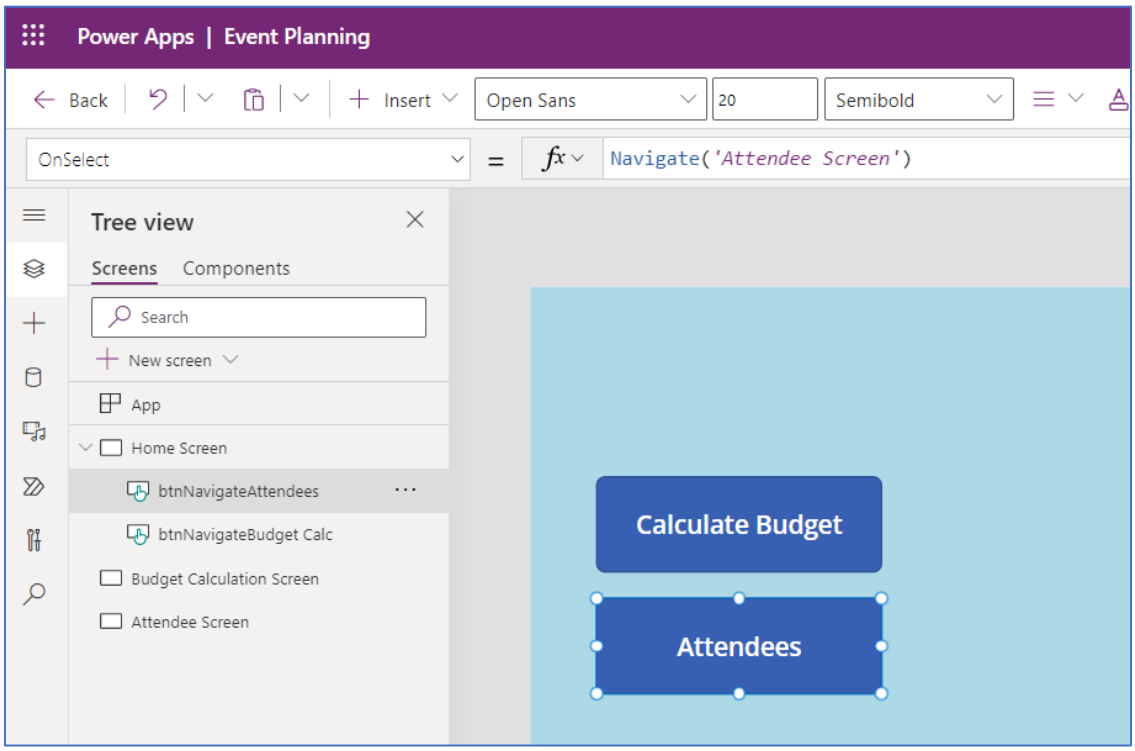
10. **Paste** onto the **Home screen**.

11. Position the copy below the first button (note how PowerApps will help you align – like in PowerPoint),

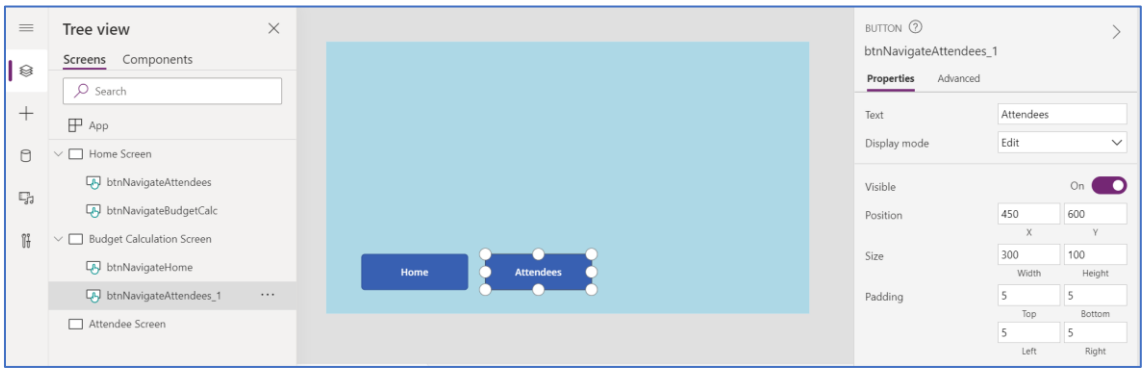
12. Set the following Properties (try double clicking on the screen name and the button):

Property	**Value**
Name	btnNavigateAttendees
Text	"Attendees"

13. Change the OnSelect property to navigate to the **Attendee screen**.



14. On the **Budget Calculation screen**, add two buttons to navigate to the **Home Screen** and the **Attendees screen**. Name and set their Text property accordingly, making sure the Y value is 600 to help position them at the bottom of the screen.



15. On the Attendee screen, add a button to navigate to the Home Screen. Note if you copy and paste it, Power Apps will place it in the same location.

16. Using icons **Save** the app.

17. Select **Publish** the app. Then select **Publish this version**.

18. When you return to the App design and either Run the App and test the buttons, or holding ALT click on each button to test whilst in edit mode.

End of Exercise

User function

Returns information about the current user. It returns the following properties:

- Email
- Full name
- Image
- EntraObjectID

```
Syntax

User()


Example

User().FullName
```

Text function

The Text function converts a number or date/time value into text and then optionally formats the text based on a predefined format or a custom format. This function is typically used to format date/time information or currency. Details about the predefined formats and how to create custom formats can be found at https://learn.microsoft.com/en-us/power-platform/power-fx/reference/function-text

```
Syntax

Text(NumberOrDateTime, Format [,ResultLanguageTag ])


NumberOrDateTime - Required. The number or the date/time value to
format.

Format - Required. A member of the DateTimeFormat enumeration or one or
more placeholders enclosed in double quotation marks.

ResultLanguageTag - Optional. The language tag to use for the result
text. By default, the language of the current user is used.


Example

Text(1234.5,"£#,##0.00")
```

Value function

The Value function converts a string of text that contains number characters to a number value. Use this function when you need to perform calculations on numbers that were entered as text by a user.

```
Syntax

Value(String [, LanguageTag ])


String - Required. String to convert to a numeric value.

LanguageTag - Optional. The language tag in which to parse the string. If not specified, the language of the current user is used.


Example

Value("123.456")
```

If function

Tests one or more conditions for a true result. If this result is found, the function returns one value, if not it can return a different value.

```
Syntax

If( Condition, ThenResult [, DefaultResult ] )

If( Condition1, ThenResult1 [, Condition2, ThenResult2, ... [ , DefaultResult ] ] )


Condition(s) - Required. Formula(s) to test for true. Such formulas commonly contain comparison operators (such as <, >, and =) and test functions such as IsBlank and IsEmpty.

ThenResult(s) - Required. The corresponding value to return for a condition that evaluates to true.

DefaultResult - Optional. The value to return if no condition evaluates to true. If you don't specify this argument, blank is returned.
```

Now function

Returns the current date and time.

```
Syntax

Now()
```

Nesting functions

Just like other applications (e.g., Excel), you can nest functions within each other; for instance, the Value and the Text function will often be nested within each other, one to ready a Value, the other to format the Text into a recognised format for currencies or dates for instance.

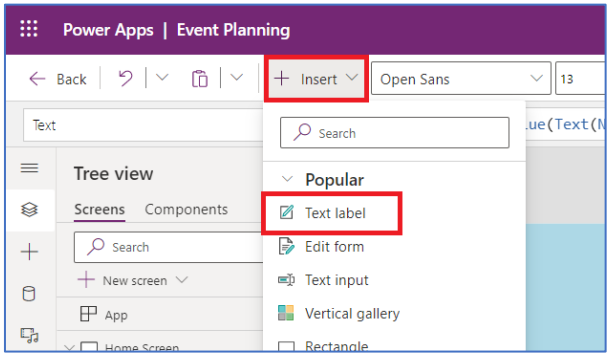
An example of a more advanced function based on the User function will be demonstrated later in this guide.

```
If(Value(Text(Now(),"hh")) <12,"Good Morning ","Good Afternoon ") &
User().FullName
```

Exercise 6: Adding Controls and functions to your app

Task 1 – Adding Controls to our screens

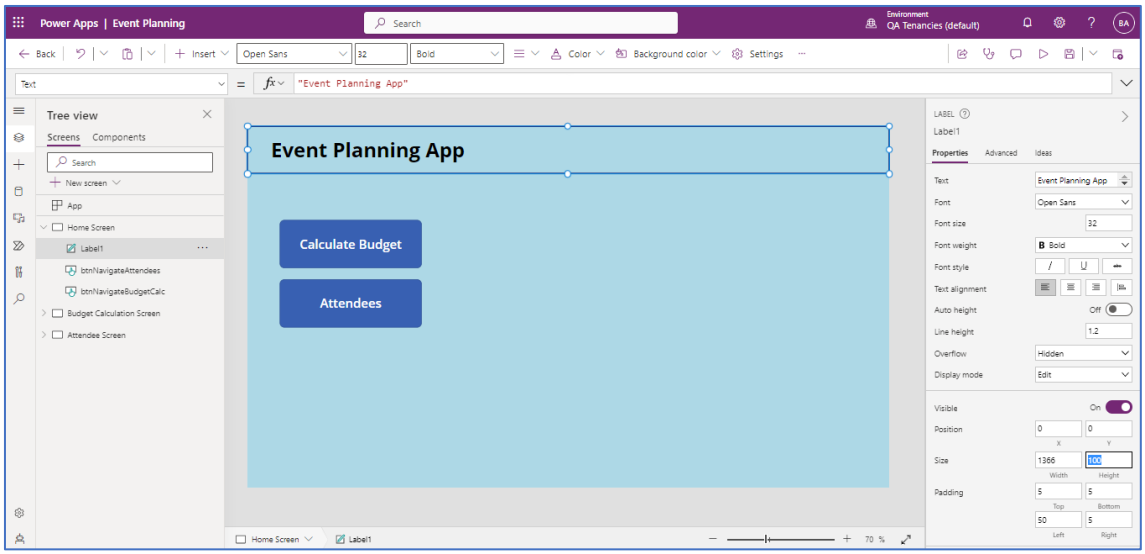
1. Select the Home Screen in the left navigation.
2. In the menu bar click **Insert.**
3. Click **Text Label** to add a label to the Home Screen.



4. Rename the label: **lblHeaderAppName** and then set the following properties:

Property	**Value**
BorderThickness	1
FontWeight	FontWeight.Bold
Height	100
PaddingLeft	120
Size	32
Text	"Event Planning App"
X	0
Y	0
Width	1366

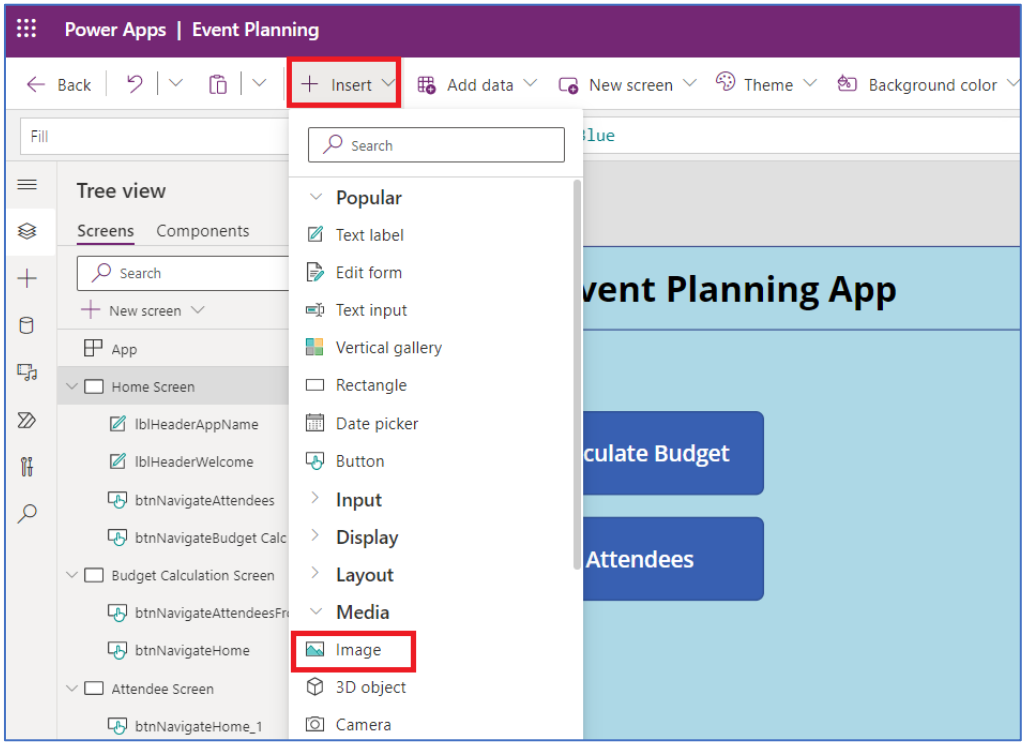
Note: these properties can be set using the menu ribbon, by editing the object in the canvas work area or by using the properties pane.



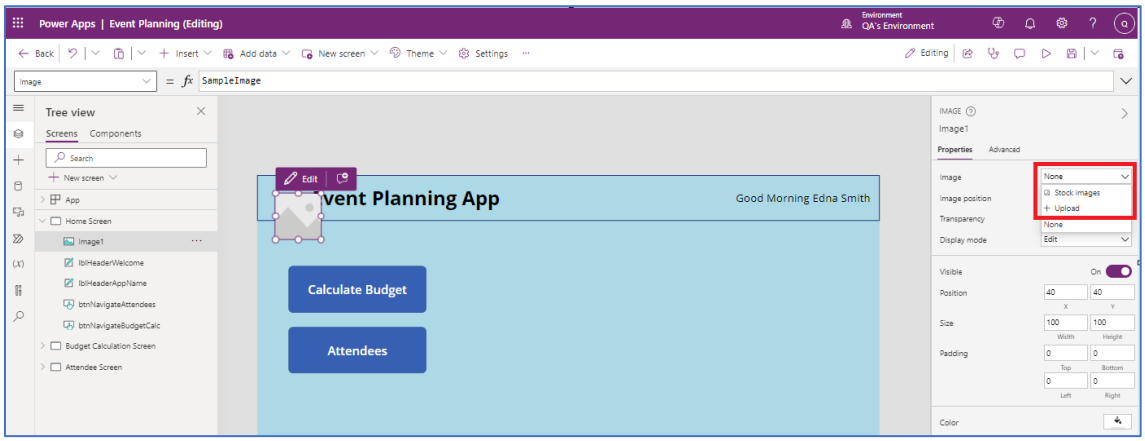
5. Add another label with the following properties:

Property	Value
Name	lblHeaderWelcome
Align	Align.Right
Height	100
PaddingRight	20
Size	18
Text	If(Value(Text(Now(),"hh")) <12,"Good Morning ","Good Afternoon ") & User().FullName
Width	450
X	916
Y	0

6. Using the Insert menu, add an Image control from the Media section.



7. With the image selected in the left navigation pane, use the properties pane to **Upload** an image file from the local computer. Select the **QALogo.png** file from the student files.



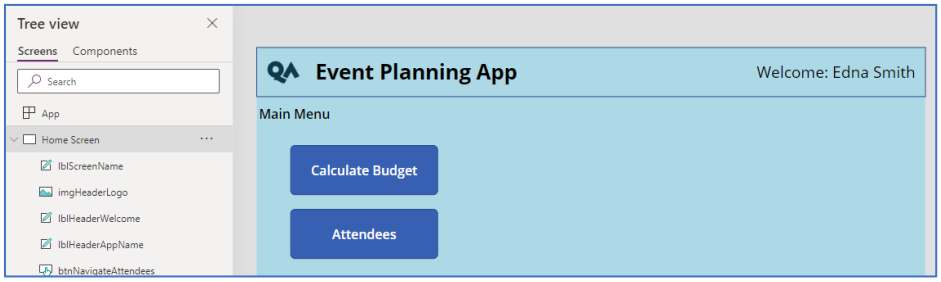
8. Configure the image with the following properties:

Property	Value
Name	imgHeaderLogo
Height	100
PaddingLeft	10
Width	100
X	0
Y	0

Note: If the image does not appear, use the Image drop down in the properties pane to select None and then reselect QALogo (This is a bug).

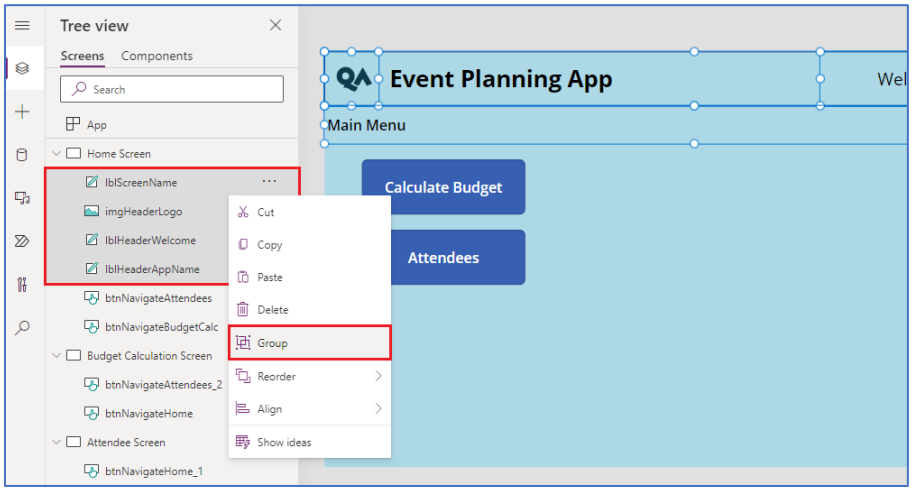
9. Add a final Text label with the following properties:

Property	Value
Name	lblScreenName
FontWeight	FontWeight.Semibold
Height	70
PaddingLeft	20
Size	20
Text	"Main Menu"
X	0
Width	1366
Y	100



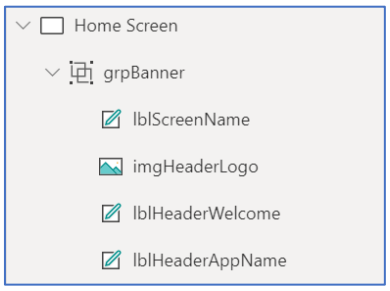
10. Using the **CTRL** or **SHIFT** key, **select** the all the controls that you have just placed (excluding the buttons).

11. **Right-click** the selected controls and then select **Group**.

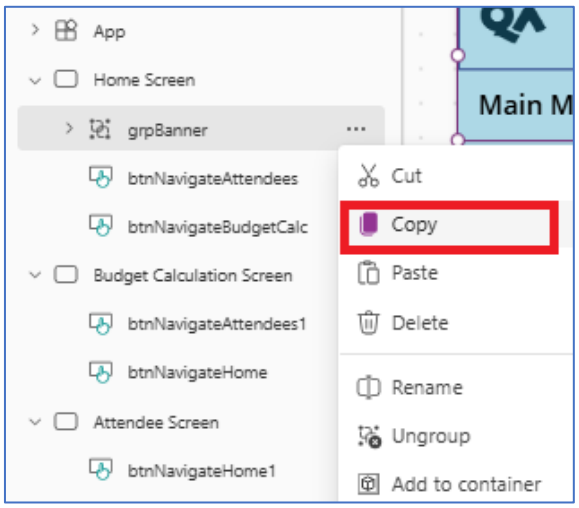


12. The controls will be added to a new group. Rename the group: **grpBanner**.

13. If you expand the group, you will be able to see the individual controls.

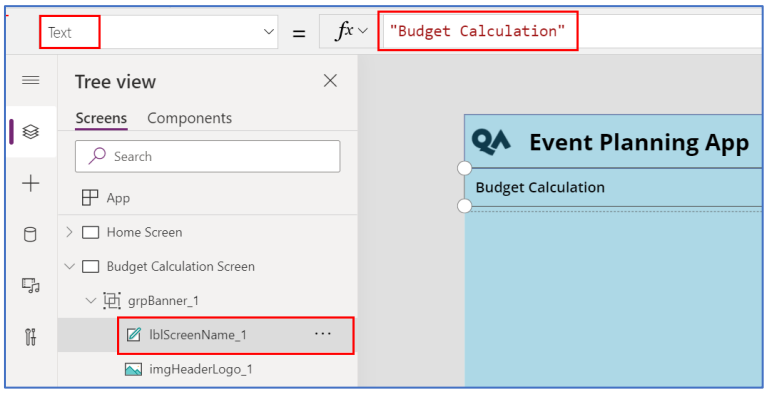


14. Click the **more** icon beside the group and then select **Copy**.

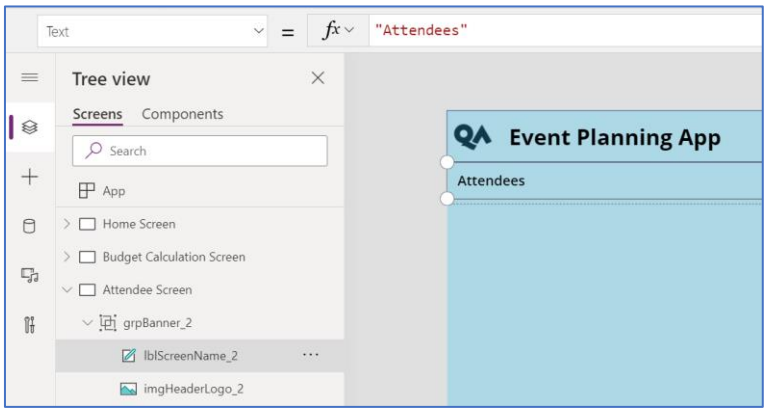


15. Select the **Budget Calculation Screen**, click the **more** icon and then select **Paste**.

16. Expand the grpBanner_1 group and locate the lblScreenName_1 label. Change its text to **Budget Calculation**.



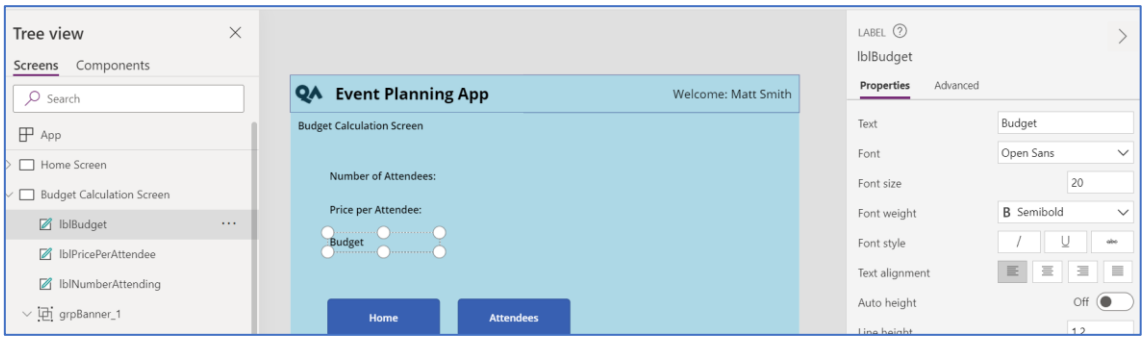
17. Repeat steps to copy the banner to the Attendee Screen. Alter the text ‘Attendees’ for the screen name label.



18. **Save** and **Publish** this version.

Task 2 – Adding functions

1. Navigate to the **Budget Calculation screen** of your app.
2. **Add three text labels**, position them, and align as shown below.
3. Set each relevant property for them.



- Label #1

Property	Value
Name	lblNumberAttending
Text	"Number of Attendees:"

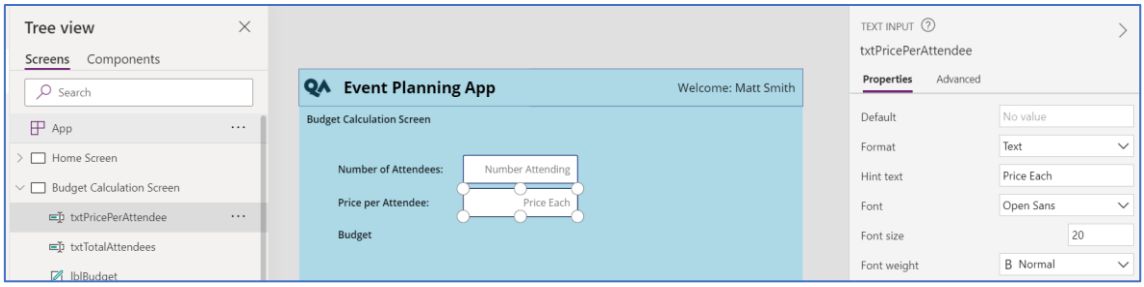
- Label #2

Property	Value
Name	lblPricePerAttendee
Text	"Price per Attendee:"

- Label #3

Property	Value
Name	lblBudget
Text	"Budget:"

4. **Insert two text input controls**, position as shown below, and enter the following properties:



- Text Input #1

Property	Value
Name	txtTotalAttendees
Align	Align.Right
Default	""
Hint Text:	"Number Attending"
Format	TextFormat.Number

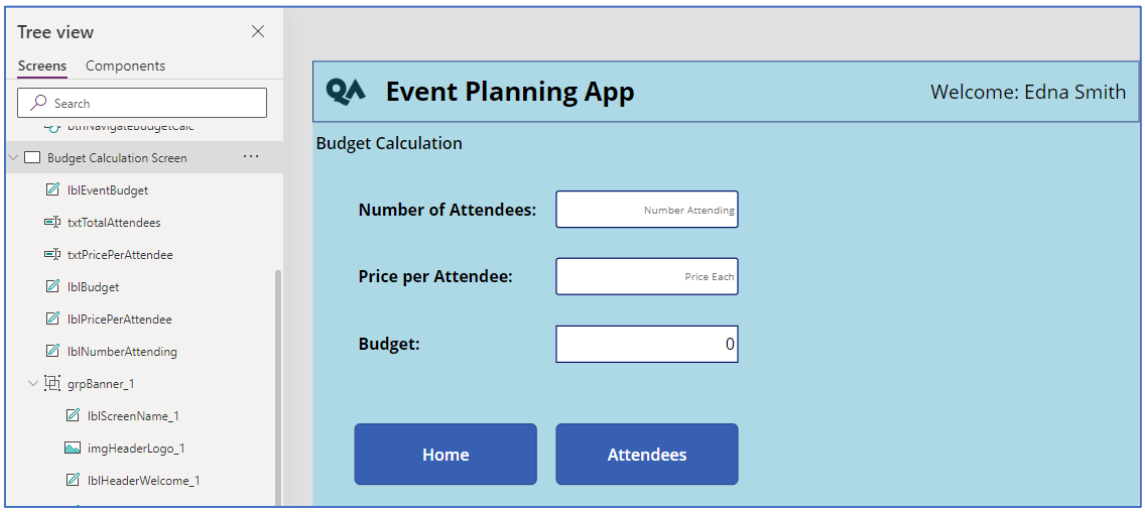
- Text Input #2

Property	Value
Name	txtPricePerAttendee
Align	Align.Right
Default	""
Hint Text:	"Price Each"
Format	TextFormat.Number

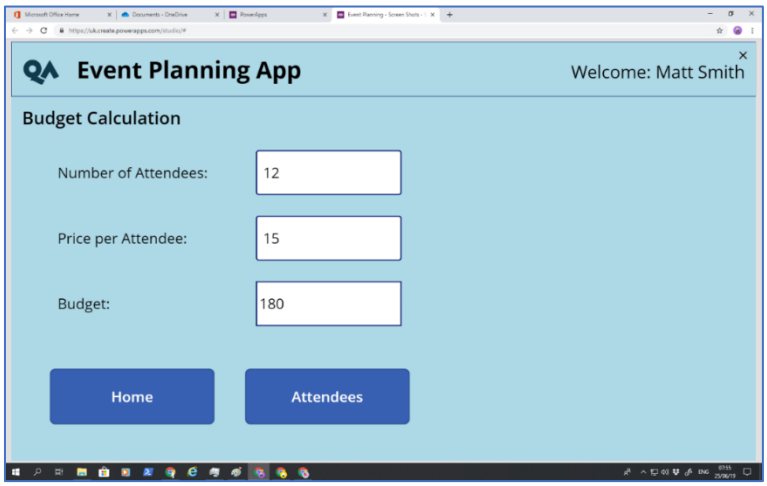
5. Insert a final **Text Label** to display the total budget. Set the properties as follows:

Property	Value
Name	lblEventBudget
Align	Align.Right
BorderThickness	2
Fill	White

Size	20
Text[1]	txtTotalAttendees.Text * txtPricePerAttendee.Text
Width	265 (or the same size as the text input boxes above it)



6. Alter and fine tune any other properties that you think enhance the look (e.g., Size, Padding Right).
7. **Save** and **Publish** the app, then return to the edit screen.
8. Preview (Run) the app.
9. Enter a figure for Attendees and Price, then confirm that the correct calculation is made.



10. Hit the X to return to editing the app.

[1] We can use the expression as it is because both text input controls were configured to accept numbers only (Format). If this had been left to the default value of text, they the values would have needed converting from text, and the code would have been: Value(txtTotalAttendees.Text) * Value(txtPricePerAttendee.Text)

11. The budget is showing as a number, not as currency, so we will need to change that.

12. Select the **lblEventBudget** label, highlight the **Text** value.

13. In the Text property type: **Text(** and before the existing formula, complete the command by typing,**"£#,##0.00")**. The text property should now read:

14. Preview the app again and confirm it displays the budget as a currency value.

End of Exercise

Module 5: Modern Styling

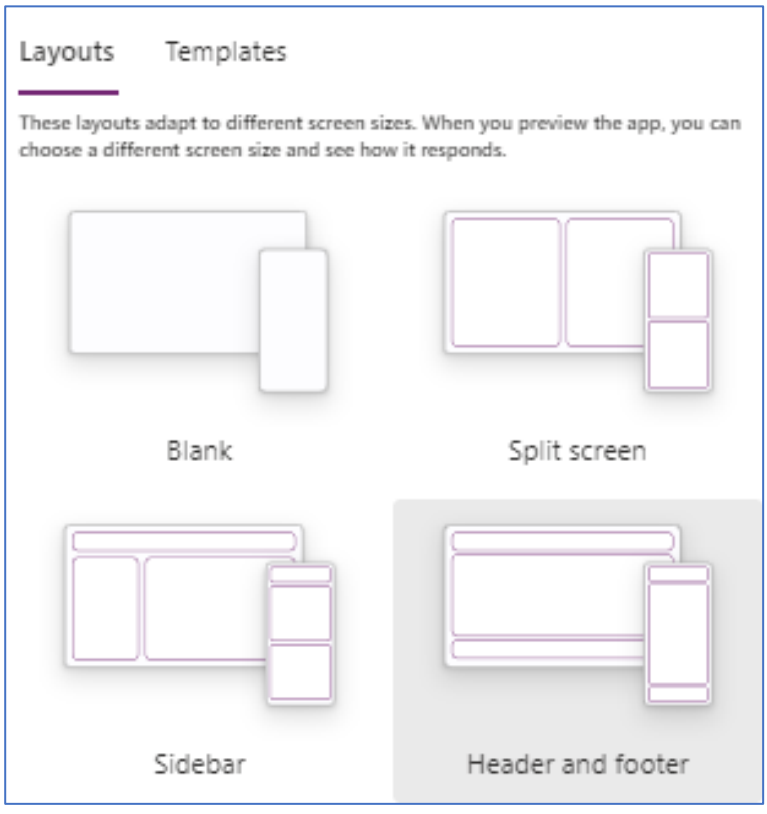
Since the initial release of Power Apps, Microsoft have given us some new ways to compose the screens we will be using. The original method (that we can still use) gave us complete freedom to place controls anywhere on the screen. While this provides great flexibility, it can also take a lot of time to fine tune and get the controls correctly positioned and aligned with each other.

Using relative properties was one way to guarantee alignment, but this took even longer to set up.

More recently, Microsoft have given us containers (Vertical Containers and Horizontal containers). When using these containers, you first divide the screen (using the containers) into the different "zones" that you wish to place controls into. Once the zones have been defined, just place the controls and they will automatically be positioned and aligned within the control, you do not need to set X and Y properties.

In a vertical container, controls added to it will be placed underneath each other. In a horizontal container, the controls will be placed side by side.

The containers can be added to an existing screen, there are also screen layouts available when you select "New Screen" that have pre-configured containers.



When you use these containers, there are properties on the containers that can help with layout. In addition, objects placed inside these containers have their properties changed to also aid in this process.

The properties available on the containers include:

- **Direction** – You can switch between vertical and horizontal containers by changing the direction property.
- **Justify and Align** – How the space is used by controls added to the container.
- **Gap** – The space left between objects.

On the controls themselves, the following properties become available.

- **Align in container** – To override the setting on the container if required.
- **Flexible height or width** – With this setting enabled, controls will expand to use all the available space (height or width). Fixed width/height controls use the space allocated, flexible width/height controls will share all the remaining space. If flexible height/width is turned off, the control will use the standard height / width.
- **Fill portions** – Used with flexible height/width. Used when multiple controls have a flexible setting and controls the ratio used to share the available space.
- **Minimum height/width** – Used with flexible height/width. Specifies the minimum amount of space that must be allocated to the control.

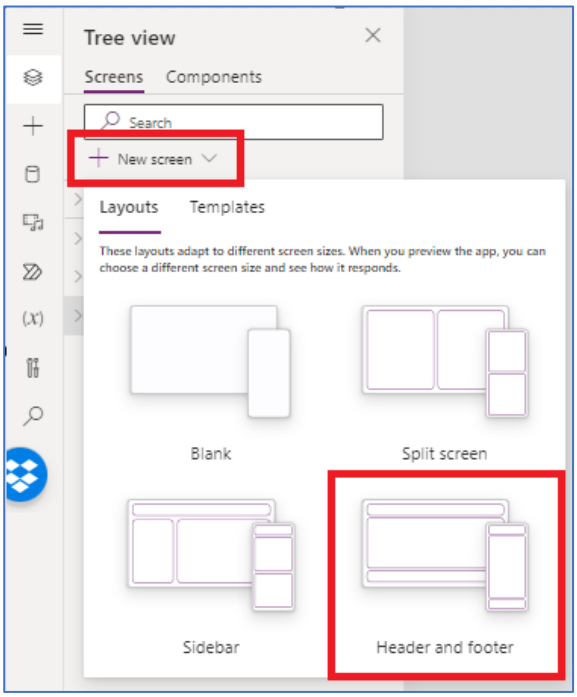
Vertical and Horizontal containers can be nested to provide more flexibility to layout design.

Exercise 7 – Add a page with Modern Styling

Modern styling is an excellent way to set up your screens, however it can take a bit of getting used to. In this next exercise we will set up a simple page so you have an opportunity to see vertical and horizontal containers in action, however we will not be changing the design of the rest of the app.

1. Switch to the browser where you are editing your Event planning app.

2. In the tree view, select **New screen** and then choose the **Header and Footer layout**.



3. **Rename** the screen giving it the name of **Variable Test Screen**.

This pre-defined layout looks very different to the other pages we have added to the app. We will change a few properties to minimise these differences.

4. Select the **Variable Test Screen** and then configure its **Fill** property to **be 'Home Screen'.Fill** – you will not be able to see the effect of this initially.
5. Select the **ScreenContainer**, and change its **Fill** property to **RGBA(0,0,0,0)** – This is transparent so we can see the screen fill.
6. Repeat the above step for the other three containers:
    a. HeaderContainer
    b. MainContainer
    c. FooterContainer
7. Select the **HeaderContainer** and then **insert** an **image control**.
8. As the header smaller than on the other screens we need to resize the image. Set the **height** and **width** properties of the image control to **75**.
9. Set the image property to QALogo.
10. With the **HeaderContainer** selected, **insert** a **Text label control**.
11. This label will be placed to the right of the image control. Set the following properties for this label.

Property	**Value**
Name	lblVTSName
Height	75
Flexible width	On

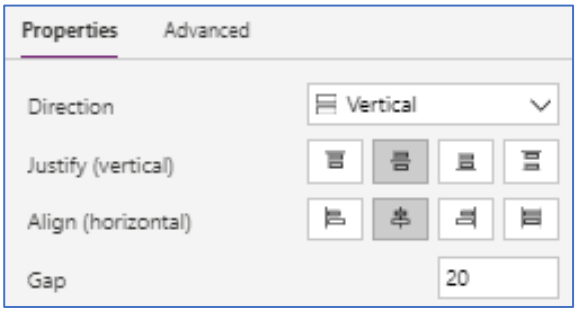
Font Weight	FontWeight.Bold
Size	32
Text	“Variable Test”

12. With the **HeaderContainer** selected, **insert** a **Text label control.**
13. This label will be placed to the right of the previous label. Set the following properties:

Property	**Value**
Name	lblVTSGreeting
Align	Align.Right
Height	75
PaddingRight	20
Size	18
Text	If(Value(Text(Now(),"hh")) <12,"Good Morning ","Good Afternoon ") & User().FullName
Width	450

14. Select the **MainContainer**, and then **add two buttons** and a **Text label**.
15. Select the MainContainer and configure the following properties:

Property	**Value**
Justify (vertical)	Center
Align (horizontal)	Center
Gap	20



16. Set the properties of the first button to:

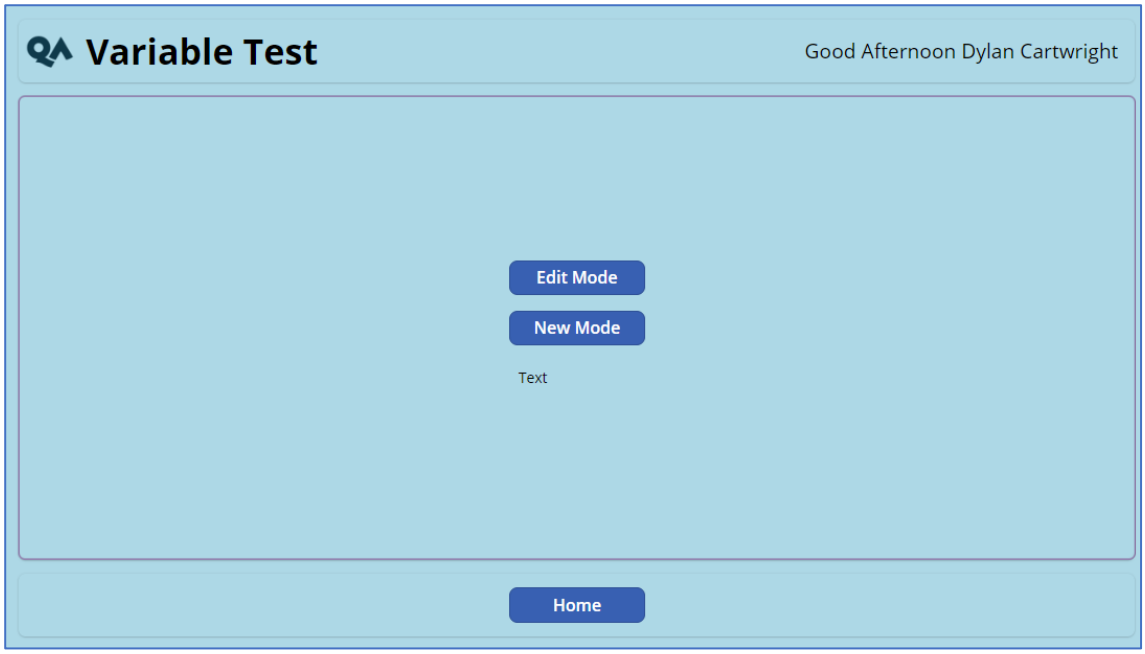
Property	Value
Name	btnEditMode
Text	“Edit Mode”

17. Set the properties of the second button to:

Property	Value
Name	btnNewMode
Text	“New Mode”

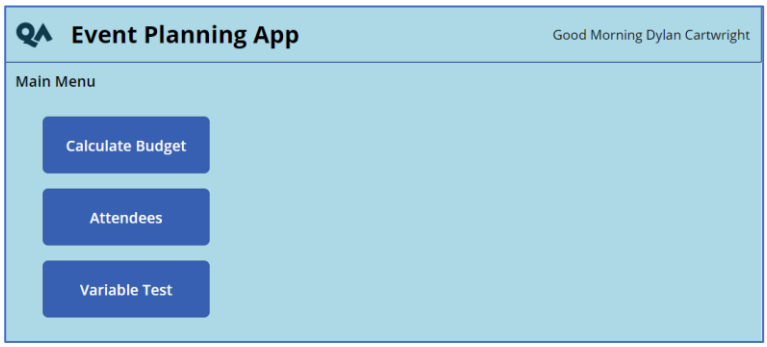
18. **Rename** the **label** to **lblMode**.
19. **Select** the container **FooterContainer**.
20. Add a button.
21. Use the Justify and Align properties of the container to position the button in the middle of the container.
22. Set the properties of the button to:

Property	Value
Name	btnVTS_Nav_Home
Text	Home
OnSelect	Navigate(‘Home Screen’)



23. In the tree view, select the Home Screen

24. Add and configure a button that you can use to navigate to the Variable Test Screen.



We will not be configuring the Variable Test Screen further at this time, but it will be used in a later exercise.

25. **Save** the app.

End of Exercise

Module 6: Galleries

Data Sources and connectors

Data Sources

In Power Apps, most canvas apps use external information stored in cloud services known as Data Sources. Common examples of data sources include a table in an Excel file stored in OneDrive for Business, and a SharePoint list.

Data Sources can be managed by using the Power Apps Studio View menu, by using the Data Source property of Forms, or the Items property of Galleries and Data Tables.

Connectors

A connector allows you to connect to an external location and access that data there. Power Apps comes with over 1100 connectors. Connectors can be:

- Standard
- Premium
- Custom

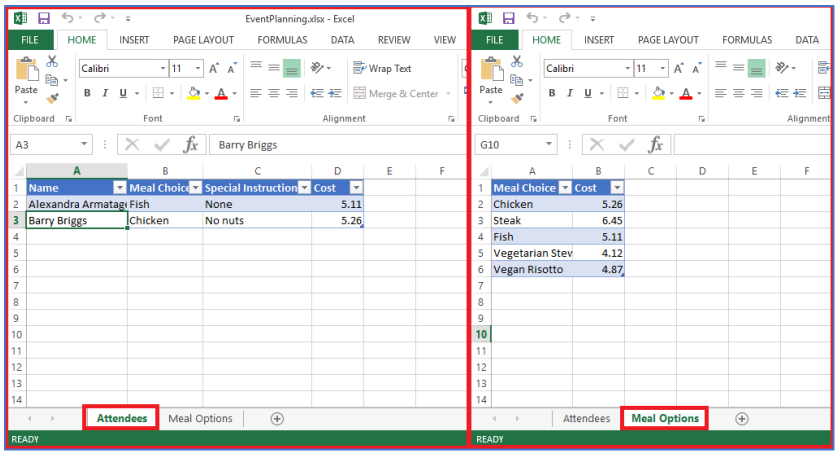
Premium and Custom connectors require higher licensing levels than the standard connectors.

Connections

A connection is a specific instance of a connector used to access a data source. It will be targeted at a table or similar structure within the data source.

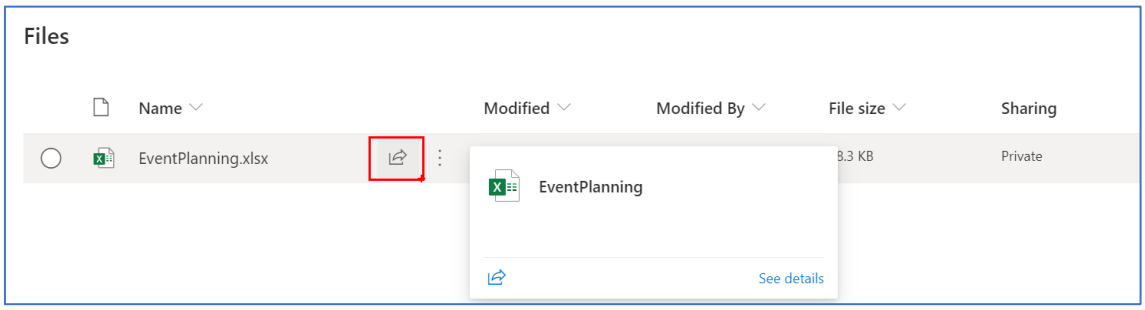
Exercise 8: Preparing data for connectors

1. In your maker account, switch to **OneDrive** and open the file **EventPlanning.xlsx**.
2. Notice that there are two tabs, **Attendees** and **Meal Options**. Each tab contains a defined **Table** with sample data.

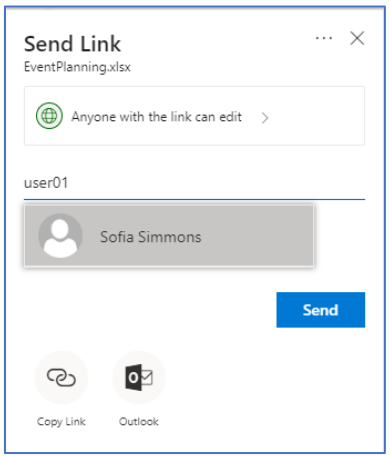


3. **Close** Excel online. *** This step is important as if the file is open Power Apps cannot access it – this is one of the reasons that Excel is not a good data source to use ***

4. Within **OneDrive**, move you mouse over the file and then select **Share**.



5. In the Send Link dialog, enter User and then select user account allocated to you. If you are in your own tenancy, select a colleague's name.



6. Confirm that the permissions are **Can edit**, and then click **Send**. (You don't have to complete the final step – read the note below however.)

Note: Although we have previously shared the app, the data also needs to be shared. As we are using OneDrive, this must be done explicitly.

Depending on the Shared settings the data might be Read Only, so always check permissions.

End of Exercise

Galleries

A gallery is one of the ways to show information from a data source. Galleries allow you to display multiple records at the same time. Each property can be displayed in a separate control that is nested inside the gallery. The gallery will create as many instances of these controls as is necessary to display the required data.

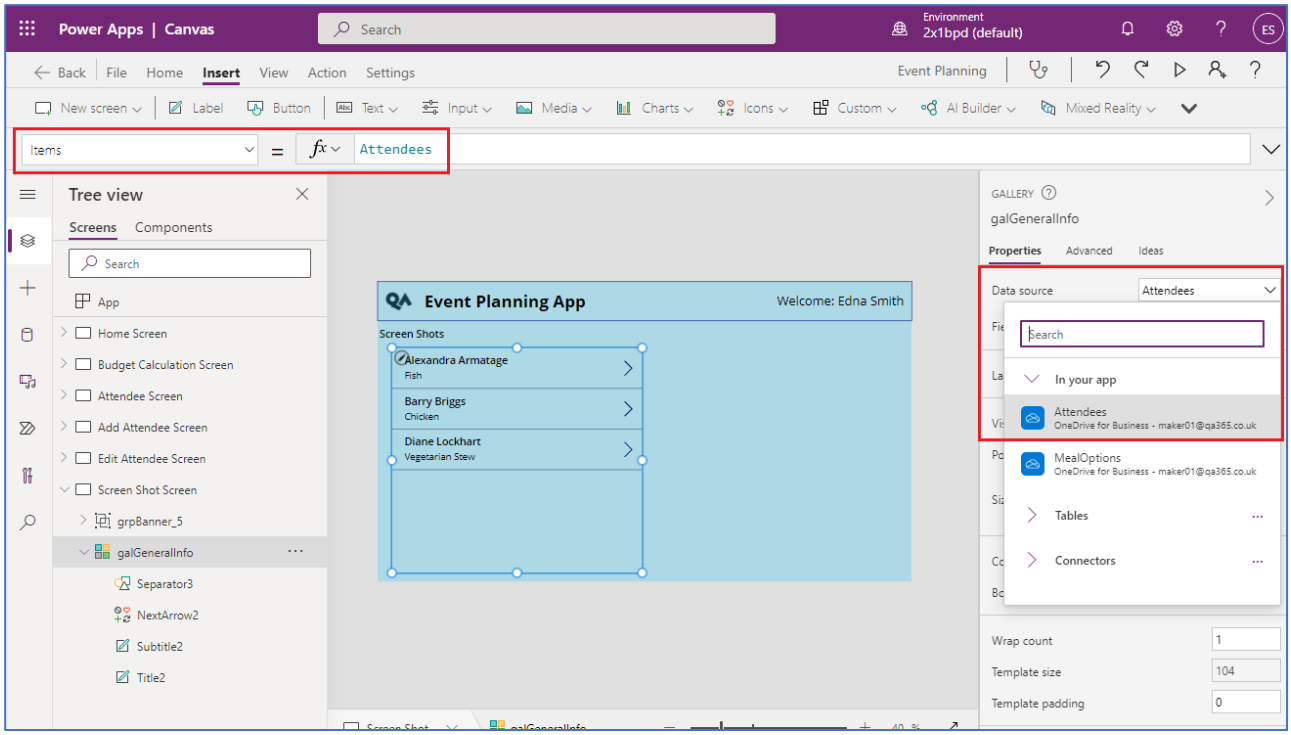
Gallery template

The template appears as the first instance of the data inside the gallery. Any changes made to the template will affect all the data instances, for example, adding a new control or moving a control. The template can be selected by clicking the edit icon that appears in the top-left of the gallery when it is selected.

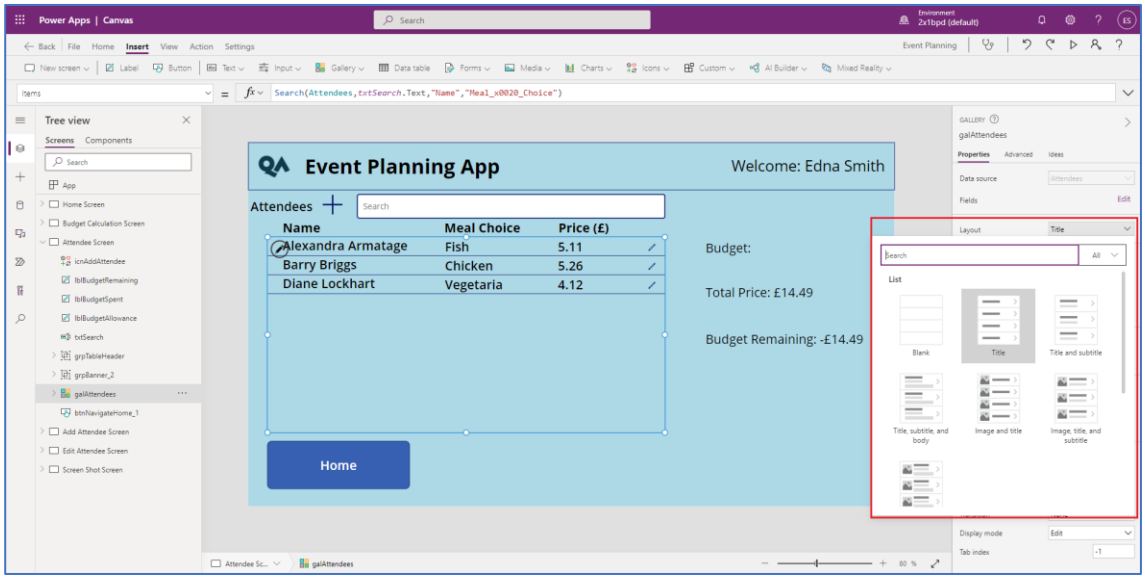
Gallery Data Source

Before a gallery can display any data, it must be connected to a data source. This is done by using the Items property of the gallery. In the properties pane, this shows as “Data Source”.

Using the drop-down box in the properties pane, you can select an existing connection, or you will be given the option to add a data source.

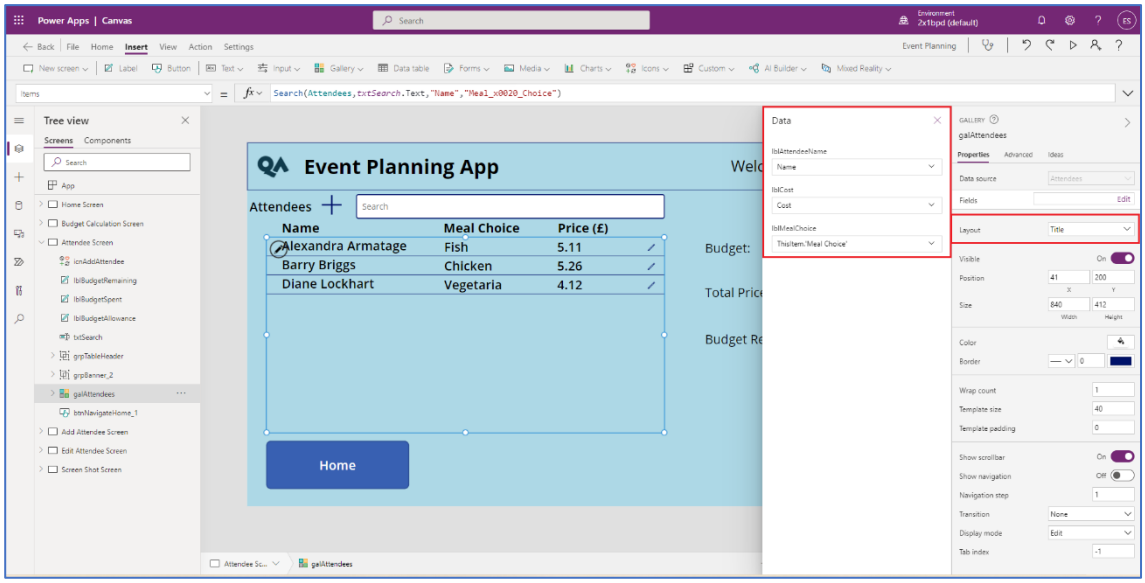


Gallery layout



Galleries come with several pre-defined layouts that you can select using the Layout option in the gallery properties pane. Once you have selected the best layout for your gallery it can be modified using the Gallery Template.

Gallery fields

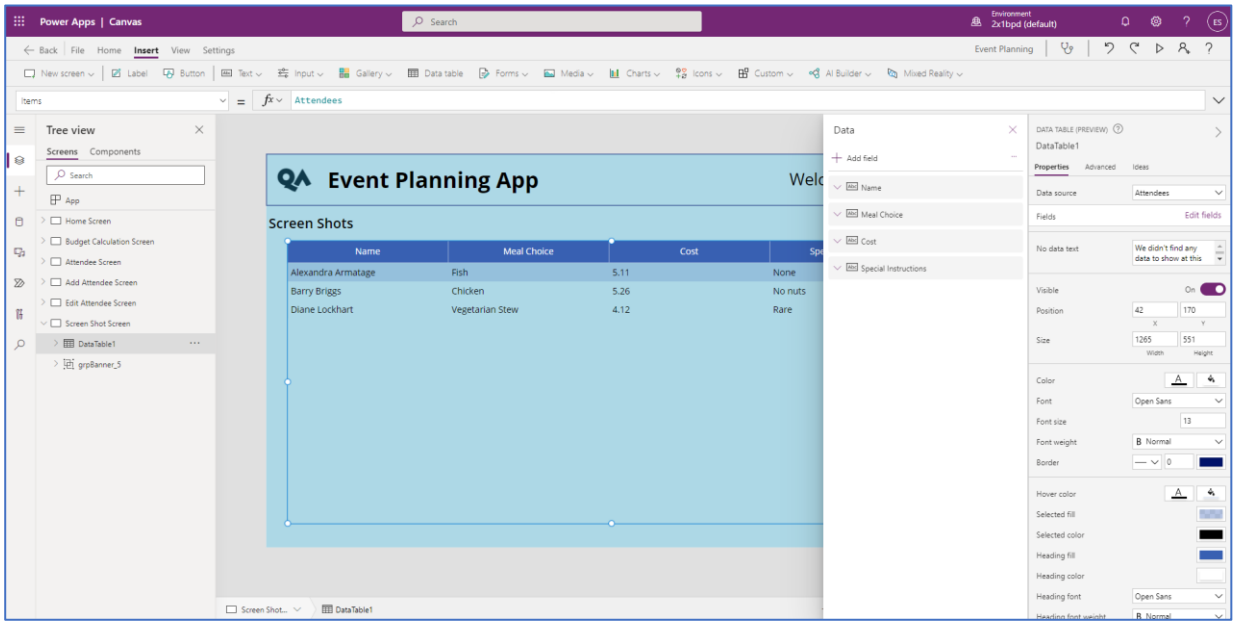


Once the gallery has a data source and layout selected, you can then specify which data from the source will be displayed. This is done by editing the Fields property of the gallery, which can be found in the Properties Pane. The Data Pane will show you the controls (both from the layout and any you added to the template) and then you select which column from the data source you want to appear in each field.

Data tables

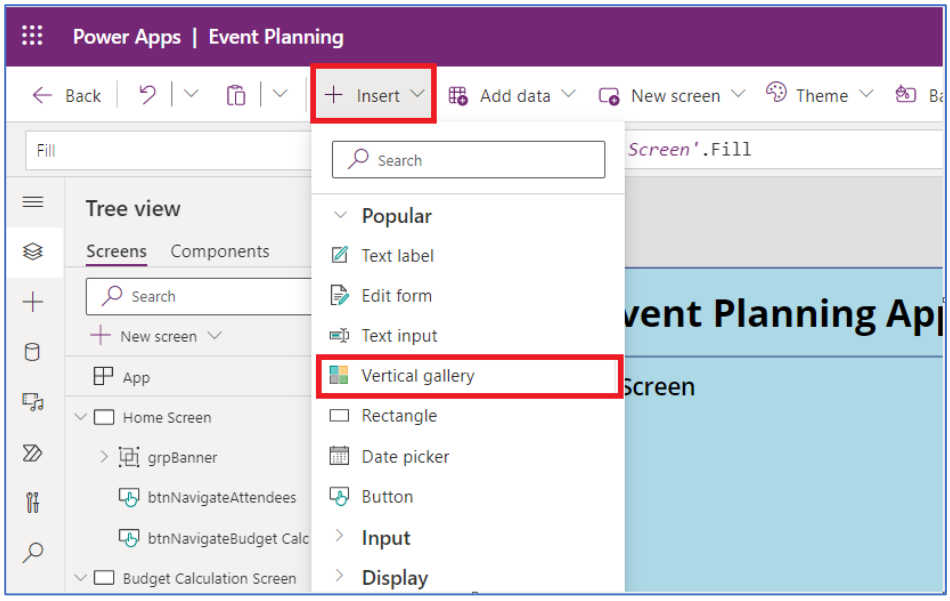
Another way to view multiple records from a data source is by using a data table control.

The data table control shows a dataset in a format that includes column headers for each field that the control shows. As an app maker, you have full control over which fields appear and in what order.



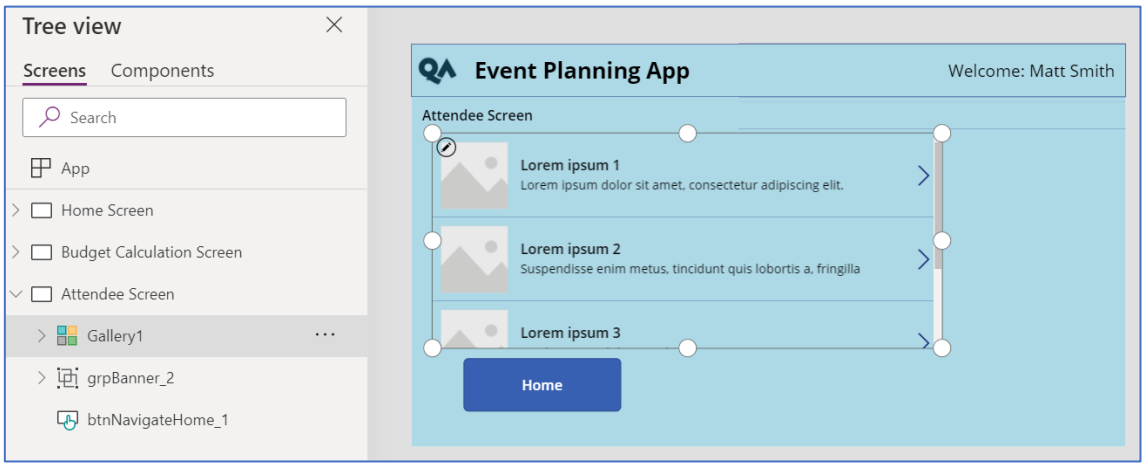
Exercise 9: Adding and configuring the gallery

1. As your maker account, switch to the tab where you are editing your app and navigate to the **Attendee screen**.
2. Using the **Insert** ribbon button, insert a **Vertical Gallery**.



3. The gallery will be placed onto the screen, and it will contain sample data. Resize and move the gallery so that it fits between the Attendees screen title and the Home button. Make its width about 2/3 of the screen size.

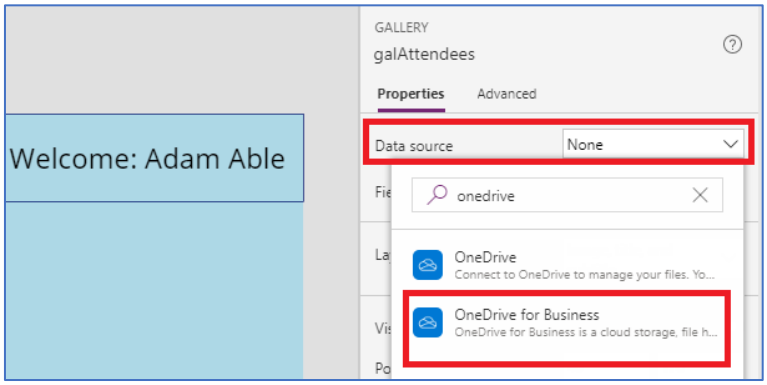
4. You may want to close the Select a data source pane for now.



5. Rename the gallery to **galAttendees**.

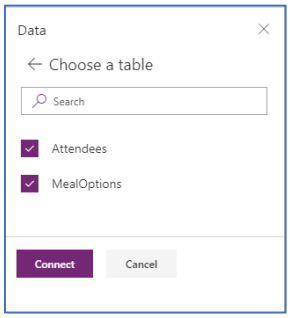
6. With **galAttendees** selected, select the **Data source drop-down** in the **properties** pane.

7. Type **OneDrive** in the **search** box and then select **OneDrive for Business**.



8. Select **Connect**.

9. Next you will be asked to choose an Excel file in your OneDrive library, pick **EventPlanning.xlsx**.

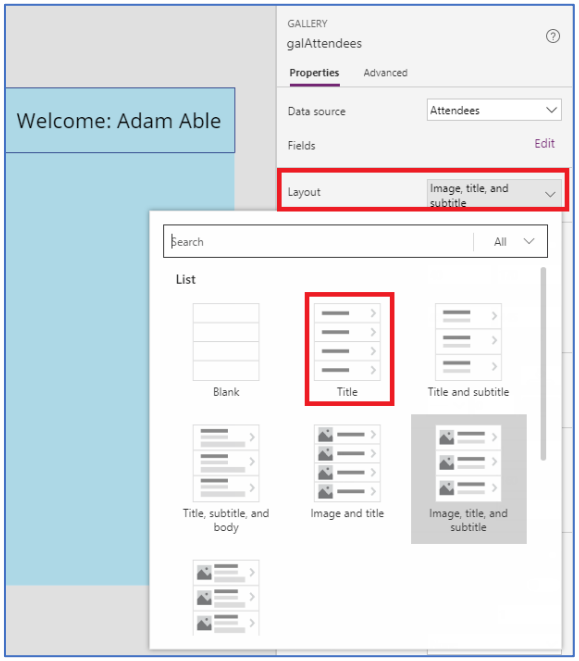


10. **Select both tables** and then click **Connect**.

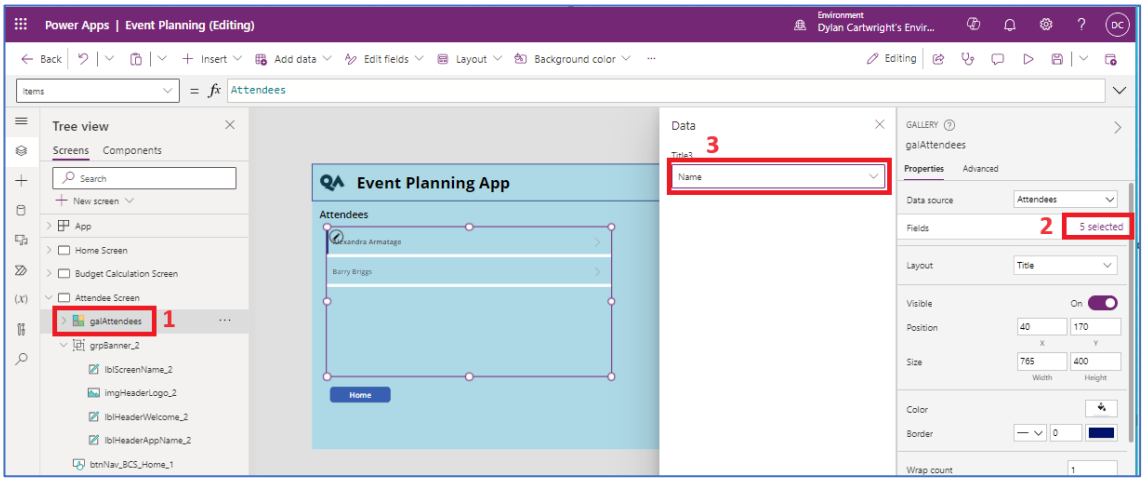
Note: Although we will only be using one of the tables in the gallery, we will require both in the app. Adding both tables now will save us having to revisit the data connections and adding a table later.

11. In the properties pane, confirm that the **Data source** property now has **Attendees** selected. If not, use the drop down to select Attendees.

12. In the properties pane, use the **Layout** drop-down and select **Title**.

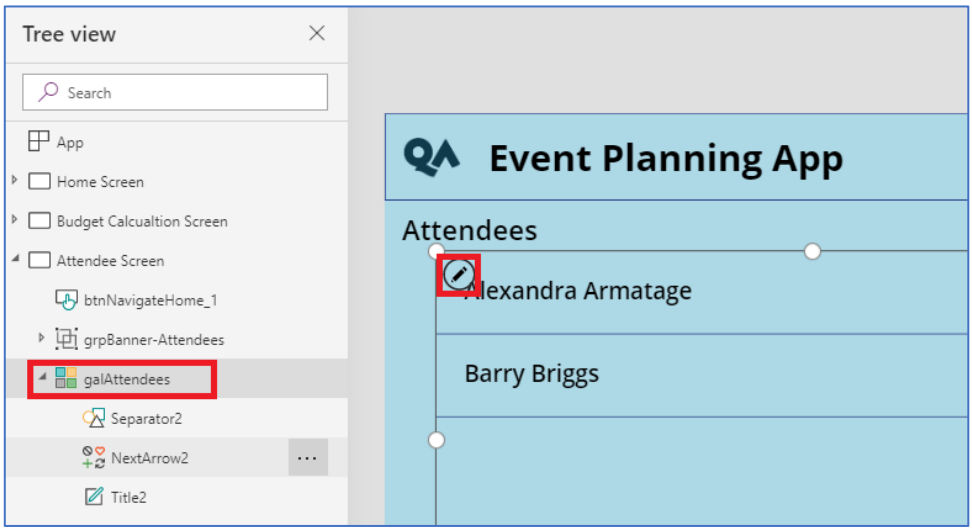


13. The gallery will change to display just one datum per entry, which will probably be a number. Beside the **Fields** entry in the properties pane, click **5 selected** and then select **Name** as the Title card.

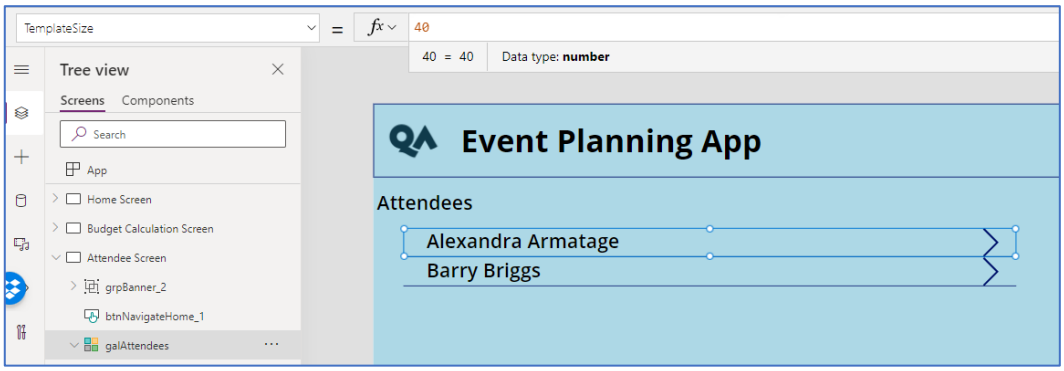


14. At this point you can close the data pane.

15. Select the gallery in the left navigation pane, then click the gallery edit icon in the top-left corner. This will select the first row (Template) in the gallery.



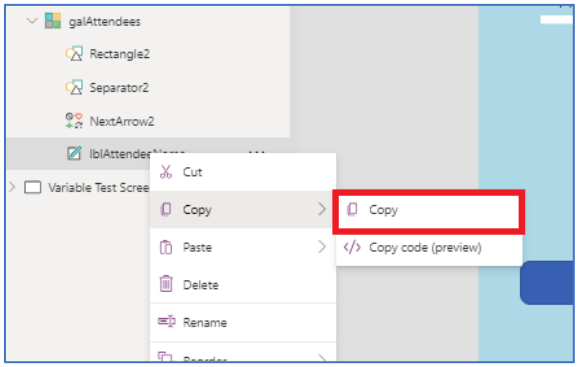
16. The top row will now be selected, but any changes you make will now affect all current (and future entries). Set the **TemplateSize** (in effect Row Height) of the entries to **40**.



17. **Rename** the title label inside the gallery to **lblAttendeeName**.

18. Set the **Width** of lblAttendeeName to **300**.

19. With **lblAttendeeName** selected, use the more icon, and select **Copy** and then select **Copy** again.



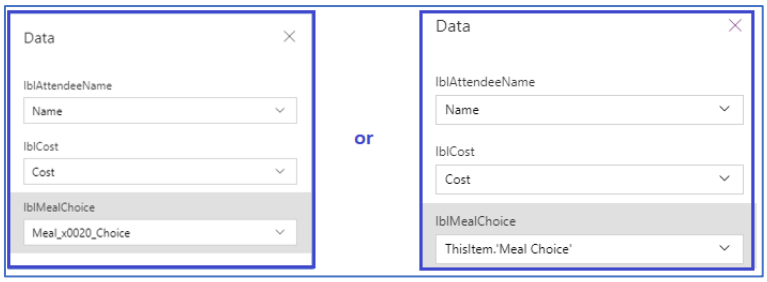
20. Press **Ctrl-V twice** and two new labels will appear in each entry (making a real mess).

21. Rename these two new labels to be called **lblMealChoice** and **lblCost**.

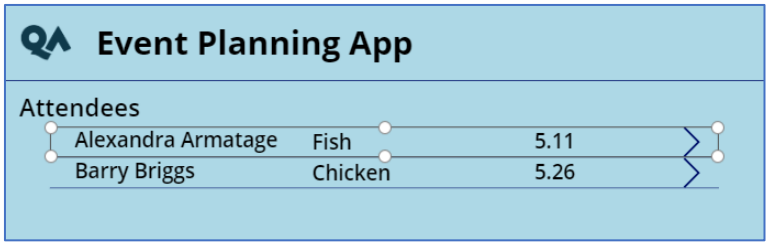
22. In the left navigation pane, **select the gallery**, and then in the property pane select **7 Selected** beside **Fields**. This will reopen the data pane.

23. Using the drop-down menus, set **lblCost** to show **Cost** and **lblMealChoice** to show **Meal_x0020_Choice**. (_x0020_ represents the space in Excel cell).

Note: Meal_x0020_Choice may appear as ThisItem.'Meal Choice' depending on the release version of Power Apps you are using. If this is the case, then there is nothing to worry about – either value displayed is correct for what we are doing.



24. **Move** and **resize** the labels so that all the information we want to see can be displayed.



25. Inside the gallery, select the nextArrow **icon** and **rename** it to **icnEditAttendees**.

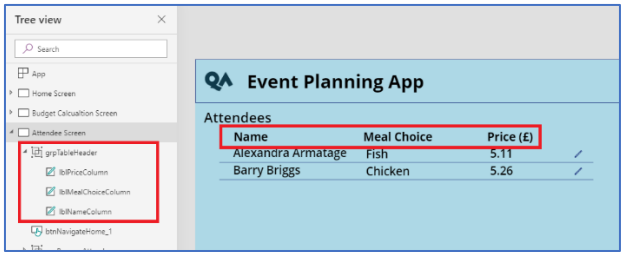
26. Edit the **Icon property** of icnEditAttendees to **Icon.Edit**.

27. Resize the icon to ensure it fits within the table.

28. Edit the **Y** property of the **gallery** to **200** to make room for the gallery headings.

29. **Add three new Text labels** to the Attendee Screen as shown below:

30. Set their properties such as Bold size 20 and position accordingly.



31. **Save** the App.

Search function

In some situations, you may want to display all the items from the data source in a gallery or data table, but in many cases, you may want to display some of the items only. For a gallery, this can be done by changing the items property to include a search function rather than just the name of the data source.

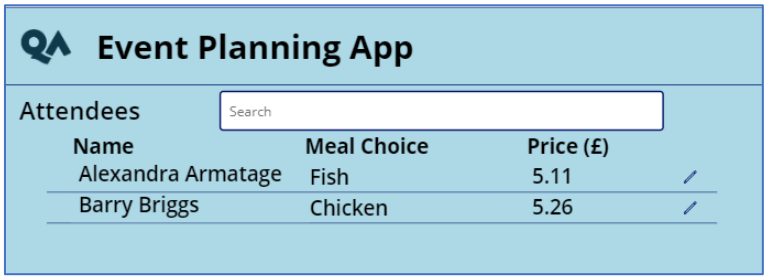
The Search function finds records in a table that contains a string in one of their columns. The string may occur anywhere within the column; for example, searching for "rob" or "bert" would find a match in a column that contains "Robert".

```
Syntax

Search(Table, SearchString, Column1 [, Column2, ... ])


Table - Required. Table to search.

SearchString - Required. The string to search for. If blank or an empty string, all records are returned.

Column(s) - Required. The names of columns within Table to search. Columns to search must contain text. Column names must be strings and enclosed in double quotes. However, the column names must be static and cannot be calculated with a formula. If SearchString is found within the data of any of these columns as a partial match, the full record will be returned.


Example

Search(Attendees,txtSearch.Text,"Name","MealChoice")
```

Exercise 10: Adding Search to a gallery

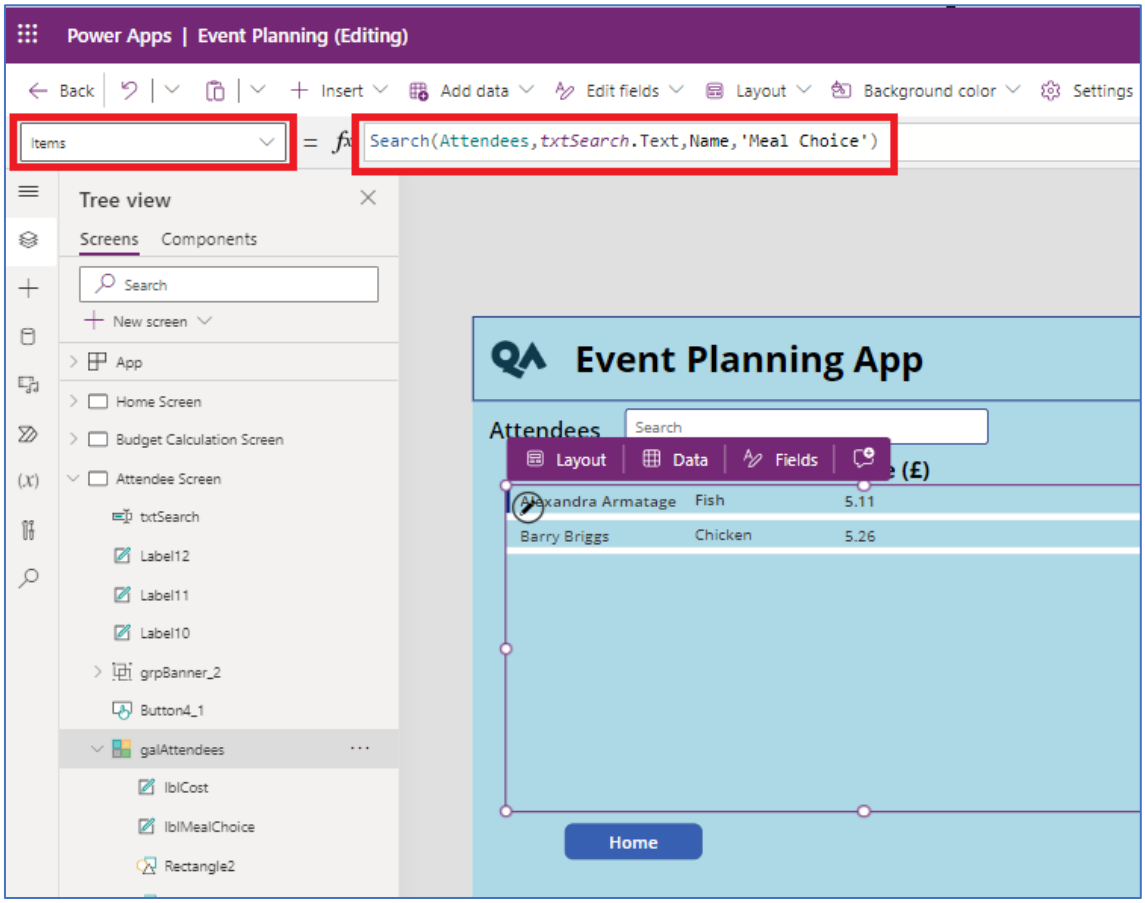
1. Add a new **Text Input** box to the **Attendee** screen. Position it as shown in the screen shot and set its properties as below:

Property	**Value**
Name	txtSearch
Clear	true
Default	""
Height	40
HintText	"Search"



2. Select the **gallery**, and edit its **Items** property to be:

```
Search(Attendees,txtSearch.Text,Name,'Meal Choice')
```



3. **Preview** the app and confirm that if you type a partial name or meal choice, the display is filtered to only show matching entries.

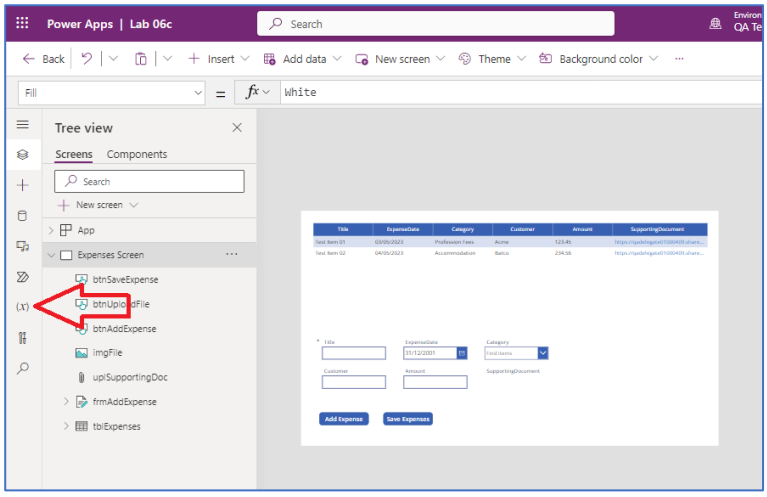
4. **Clear the search** box and **close** the app preview mode.
5. **Save** the app.

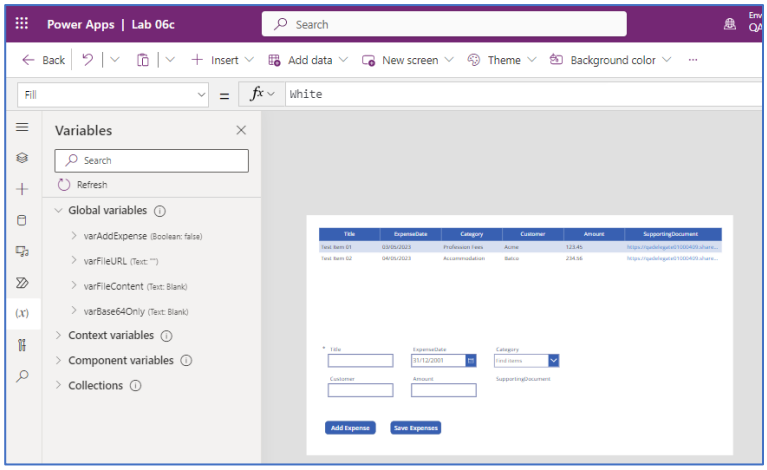
End of Exercise

Module 7: Variables and Named Formulas

As your apps become more complex, it may be necessary to store information in memory so that it can be used later. Power Apps supports several types of variables, but the two simple types are Global Variables and Context Variables.

In the Power Apps studio, your variables can be inspected using the Variables pane.





Context variables

Context variables are scoped to a single screen within your app. This means that they can only be accessed from that screen and are invalid on other screens. This is useful if you need to provide isolation of information between different screens. However, if necessary, Context variables can be set with information passed from elsewhere as part of the Navigate function.

Context variables are set with the UpdateContext function.

Update Context function

```
Syntax

UpdateContext(UpdateRecord)

UpdateRecord – Required. A record that contains the name of at least one column and a value for that column. A context variable is created or updated for each column and value that you specify.

Example

UpdateContext({ Counter: 1 })
```

Global variables

Global variables are accessible from anywhere within your app. Global variables are set with the Set function.

Set function

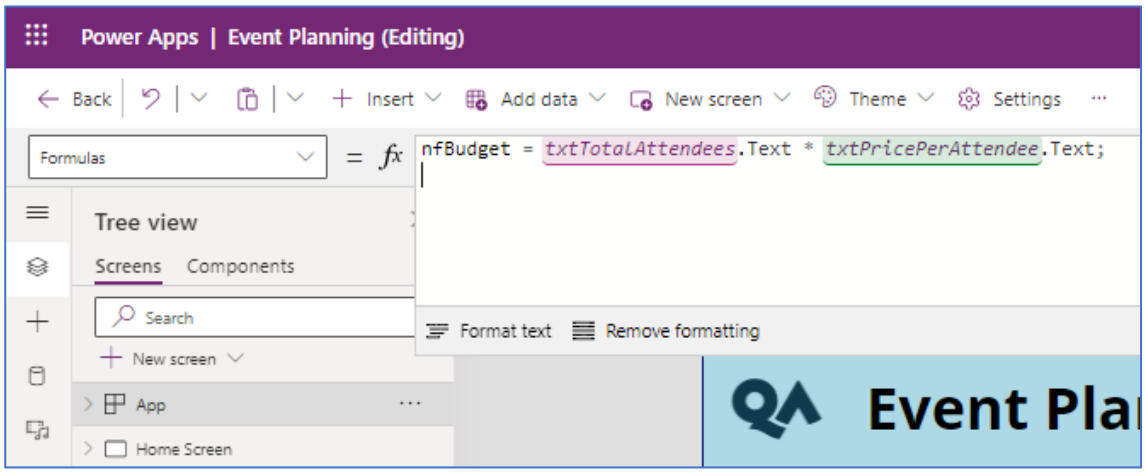
```
Syntax

Set(VariableName, Value)

VariableName - Required. The name of a global variable to create or update.

Value - Required. The value to assign to the context variable.

Example

Set(Counter, 1)
```

Named Formulas

Named Formulas, much like variables, can be used to store information for use elsewhere in the app. The scope of a Named Formular is "Global", meaning that they can be used anywhere in the app.

Unlike a variable, Named Formulas are calculated and recalculated as needed, this can allow apps that will load quicker as the alternative often had a lot of code execution occur before the user interface was shown.

To configure a Named Formula, select the App object in the Tree view and then configure its Formulas property.



Named Formulas are immutable, meaning that once create they cannot be altered in the app, this is because they can only be defined in one place. To contrast this, a variable could be configured on several different controls, and could end up storing very different information.

Finally, the syntax to create a Named Formula is different to either of the syntaxes that we use for variables. The syntax is:

```
Name = [Code];
```

It is important to note that each Named Formula definition should end with a semi-colon, even if there is only one definition.

The next exercise will show how to add a column within a gallery, to do this the Sum function will be used.

Sum function

The Sum function calculates the sum of its arguments:

```
Syntax

Sum(Table, NumericalFormula)


Table - Required. Table to operate on.

NumericalFormula - Required. Formula to evaluate for each record. The result of this formula is used for the aggregation. You can use columns of the table in the formula.


Example

Sum(Attendees, Cost)
```

Exercise 11: Using variables, named formulas, and performing calculations on a gallery

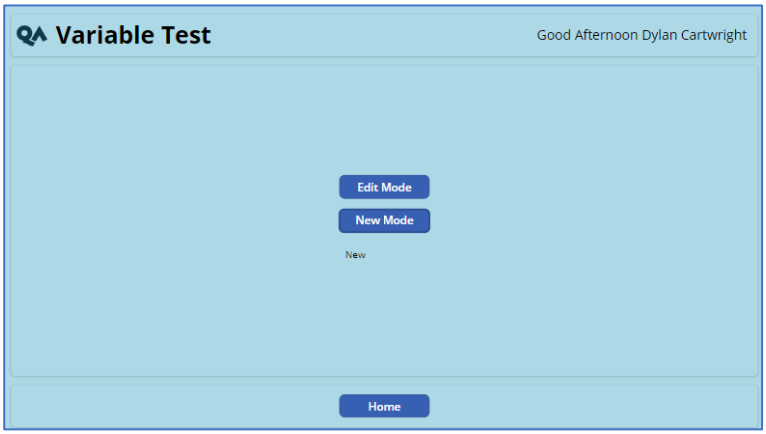
Task 1 – Using Variables

Later in the design of the app we will want to use a variable to track what task the user is performing. At this point, we will use the Variable Test Screen to try the formulas that we will be using later to set variables.

Variables will be used in this case as we want the user to change the value of the variable depending on the task they are performing, selecting different buttons (or icons) will set a different value to a variable.

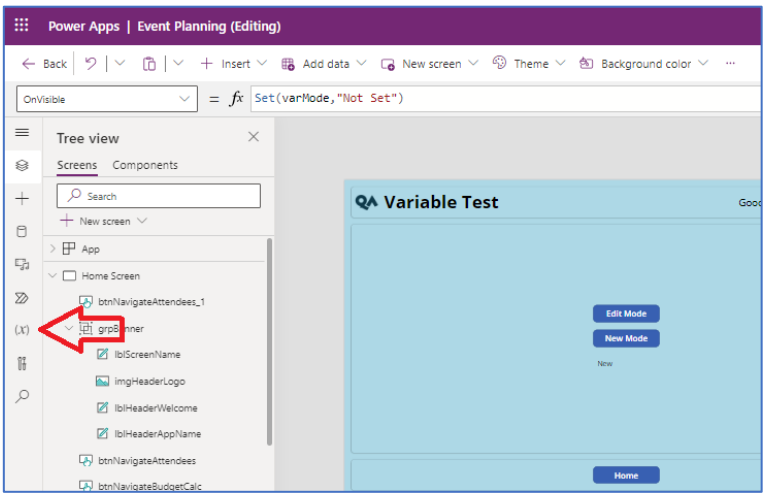
In this first task, we will be using a global variable named varMode.

1. In the browser where you are editing the Event Planning, app, select the **Variable Test Screen** in the tree view.
2. Using either the properties drop down or the advanced properties tab, configure the **OnVisible** property of the screen to the following expression: **Set(varMode,"Not Set").**
3. Configure the **OnSelect** property of **btnEditMode** to: **Set(varMode,"Edit")**.
4. Configure the **OnSelect** property of **btnNewMode** to: **Set(varMode,"New").**
5. Set the **Text** property of **lblMode** to **varMode**.
6. In the tree view, select the **Home Screen** and then preview the app.
7. Navigate to the Variable Test screen using the button. The label should display "Not set".
8. Select the Edit Mode button. The label should display "Edit".
9. Select the New Mode button. The label should display "New".

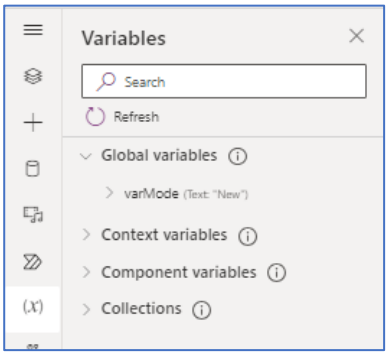


10. Exit preview mode.

11. Using the menu to the left of the tree view, select Variables.



12. Expand Global variables and confirm that you can see varMode listed and its current value.



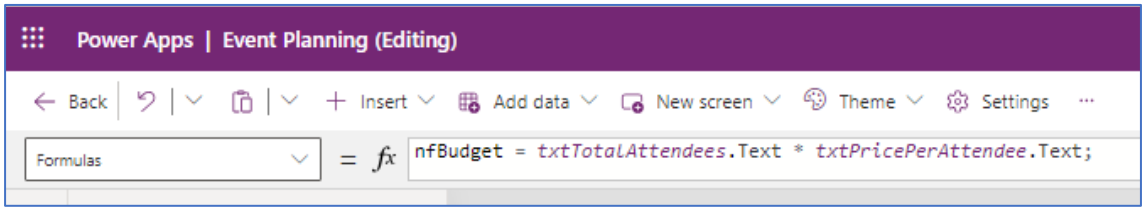
13. Return to the Tree view.

Task 2 – Using Named Formulas

On the Budget Calculation Screen, we calculated a budget. We will want to use this value in several places so we will save it as a named formula. We are using a named formula instead of a variable as to use a variable would require two OnChange properties to set with the same code, using a named formula means we only need to configure the code once.

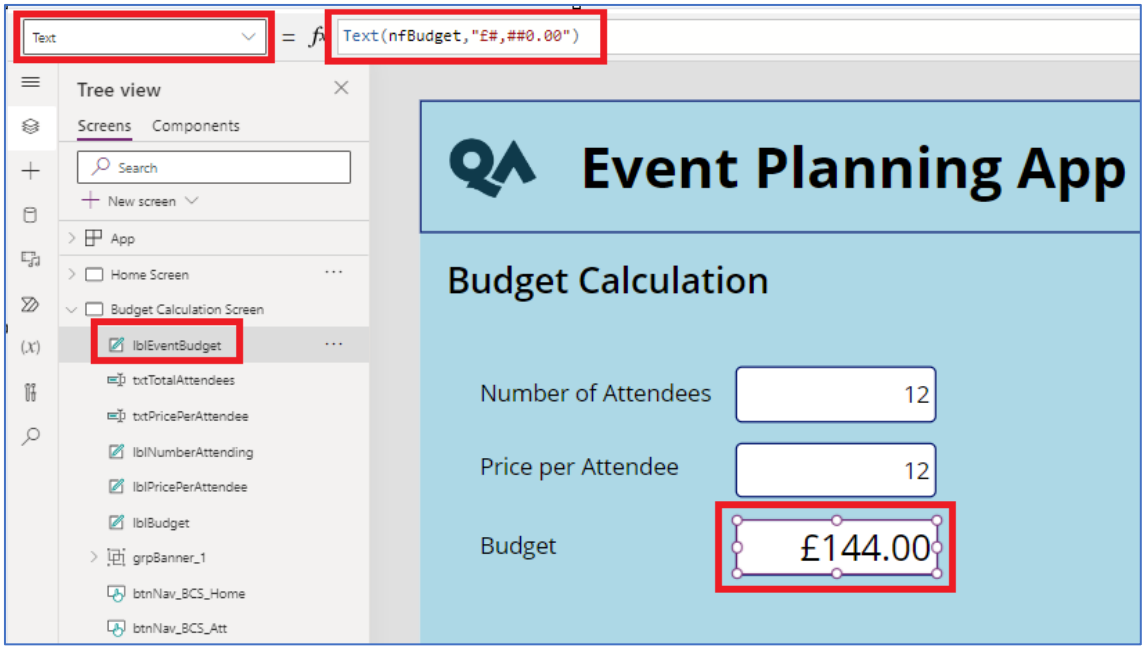
1. Select the **App** object and then select the **Formulas** property.
2. Add the following expression into the Formulas property:

```
nfBudget = txtTotalAttendees.Text * txtPricePerAttendee.Text;
```



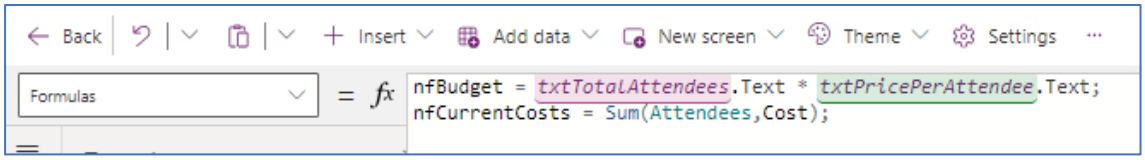
3. Navigate to the **Budget Calculation Screen**.
4. Edit the **Text property** of **lblEventBudget** to read:

```
Text(nfBudget,"£#,##0.00")
```



5. **Save** the app.
6. **Preview** the app and confirm that the budget calculation still works.
7. Select the App object and configure a second formula using the following code to calculate the current cost of meals.

```
nfCurrentCosts = Sum(Attendees, Cost);
```



8. Navigate to the **Attendee screen.** We will be adding some more Text labels, look at the next screen shot for an idea of size and position.
9. Add a new **Text label**. Set the properties as shown below:

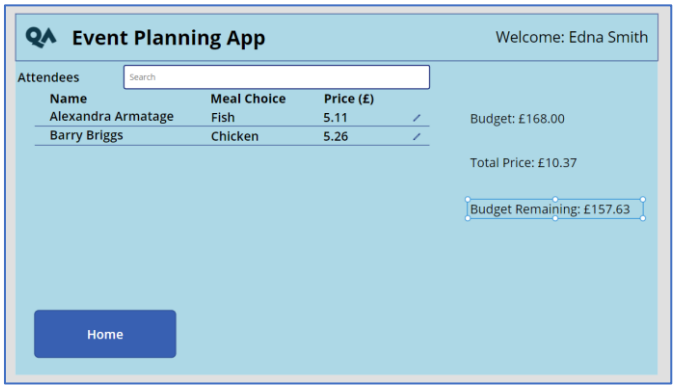
Property	Value
Name	lblBudgetAllowance
Size	20
Text	"Budget: " & Text(nfBudget,"£#,##0.00")

10. Add another new Text label with the following properties:

Property	Value
Name	lblBudgetSpent
Size	20
Text	“Total Price: “ & Text(nfCurrentCosts,”£#,##0.00”)

11. Add another new Text label with the following properties:

Property	Value
Name	lblBudgetRemaining
Size	20
Text	"Budget Remaining: " & Text(nfBudget-nfCurrentCosts,"£#,##0.00")



12. **Save** and **Publish** the app.

13. **Preview** the app from the Home Screen, set a budget and test it follows to the Attendees.

End of Exercise

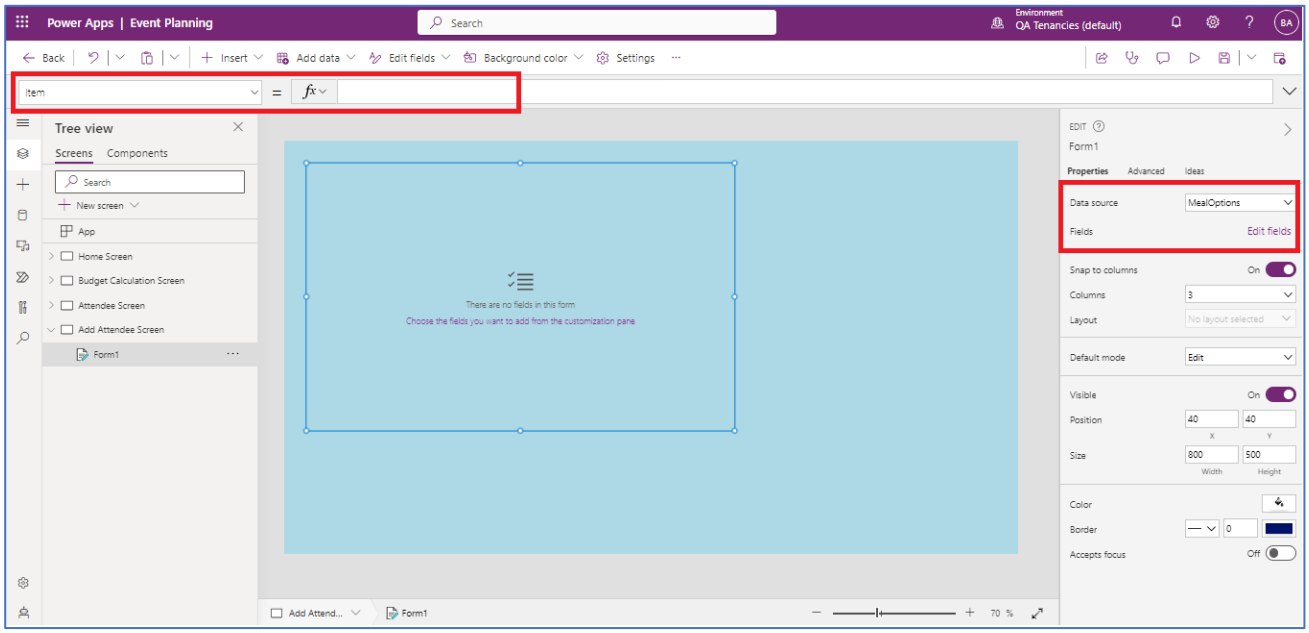
Module 8: Forms

Forms

Forms can be used to interact with a single record from a data source. There are two form controls available: edit form and display form. Data displayed in a display form cannot be altered.

When you add a form to your app, you need to configure the data source property to indicate which data source the records are from. Unless you are using the form to create a new record in the data source, you should also configure the Item property to control which record to interact with. The item property is frequently linked to the Selected property of a gallery or data table control.

Once you have a Data Source connected, you then use the fields property to control which properties of the record will be displayed. For each property selected, a data card is added. The fields pane can be used to configure and re-order the data cards.



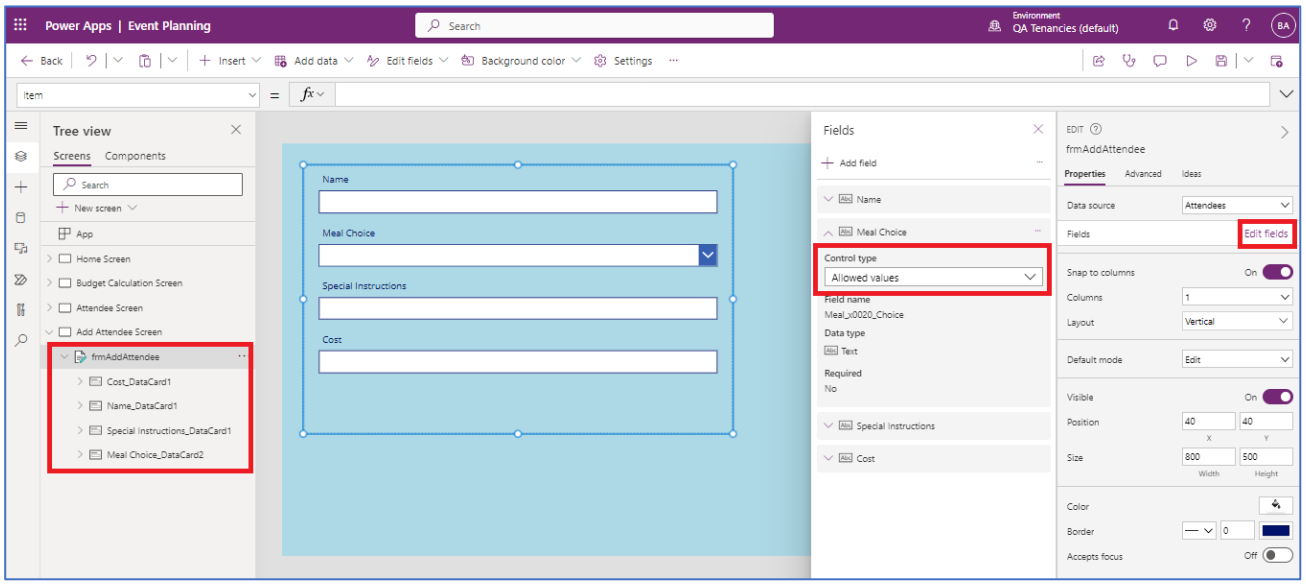
Form data cards.

Each field added to a form will be implemented as a data card. A data card is a collection of other controls, used to display the information within the form. A typical data card (on an edit form) contains four controls:

- **StarVisible**: Often hidden by default. Used to indicate when a value is mandatory.
- **ErrorMessage**: Hidden by default, can be used for error checking on a card.
- **DataCardValue**: The data from the data source.
- **DataCardKey**: The label displayed on the card.

The default type of controls in a data card can be selected using the control type property of the field in the fields pane.

By default, cards are locked, which prevents editing of the card controls. If you need to do this, you can unlock the card using the advanced tab of the properties pane.



Hiding Cards

When a card is present in an edit form, its value will be used to update the data source. There may be times where you want this update to be a calculated value instead of one that the user can enter. In this situation, you should use the default property of the DataCardValue to calculate the value. You should then hide the card by setting its visible property to false. As the card is hidden the user will not be able to interact with it, but it is still present so its value will be used.

Distinct function

The Distinct function looks up a column in a table and removes duplicate values. This is a useful function for populating drop down controls.

```
Syntax
Distinct( Table, Formula )

Table - Required. Table to evaluate across.
Formula - Required. Formula to evaluate for each record.

Example
Distinct(MealOptions,'Meal Choice')
```

Lookup function

The LookUp function finds the first record in a table that satisfies a formula. Use LookUp to find a single record that matches one or more criteria.

```
Syntax

LookUp(Table, Formula [, ReductionFormula ])


Table - Required. Table to search.

Formula - Required. The formula by which each record of the table is evaluated. The function returns the first record that results in true. You can reference columns within the table.

ReductionFormula - Optional. This formula is evaluated over the record that was found, and then reduces the record to a single value. You can reference columns within the table. If you don't use this parameter, the function returns the full record from the table.

Examples

LookUp(IceCream, Flavour = "Chocolate", Quantity)

LookUp(MealOptions,'Meal Choice'=drpMealChoice.SelectedText.Result).Cost
```

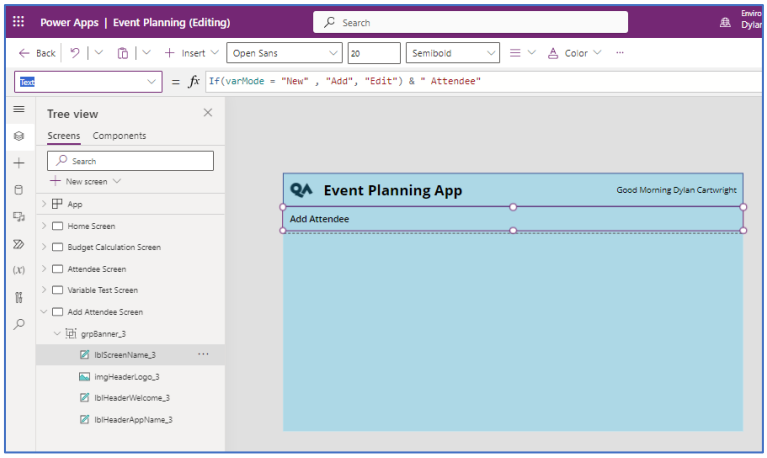
Exercise 12: Adding an edit form

Task 1 – Adding a new form

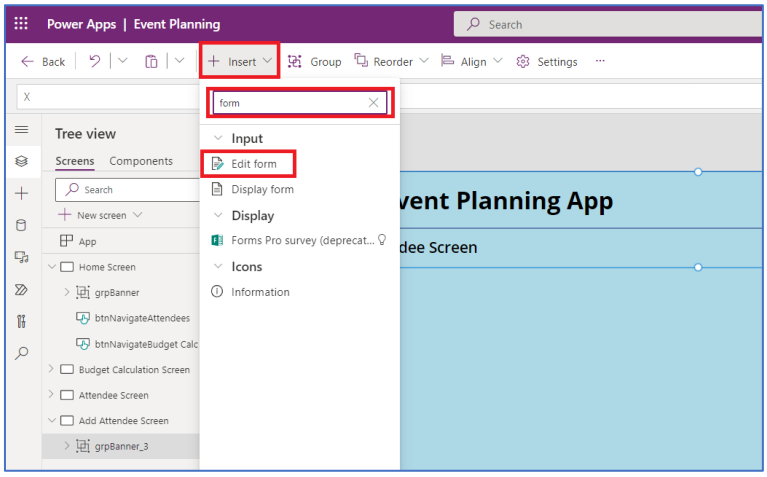
1. In the Tree view, select the **Variable Test Screen**.
2. **Preview** the app and **select** the **New Mode button** and then **exit** the **preview** mode. This step is to make sure that varMode is set correctly for this exercise.
3. Add a **new blank screen** to your App and rename it **Add Attendee Screen**.
4. Set the **Fill property** of the Add Attendee Screen to **'Home Screen'.Fill**.
5. **Copy grpBanner** from the Home Screen to the **Add Attendee Screen**.
6. Expand the group, then edit the **text** property of the **Screen Name label** to the following code:

```
If(varMode = "New" , "Add", "Edit") & " Attendee"
```

7. Rename this label to **lblScreenName_AddScreen**.
8. If necessary, reposition the Banner and any element inside it.

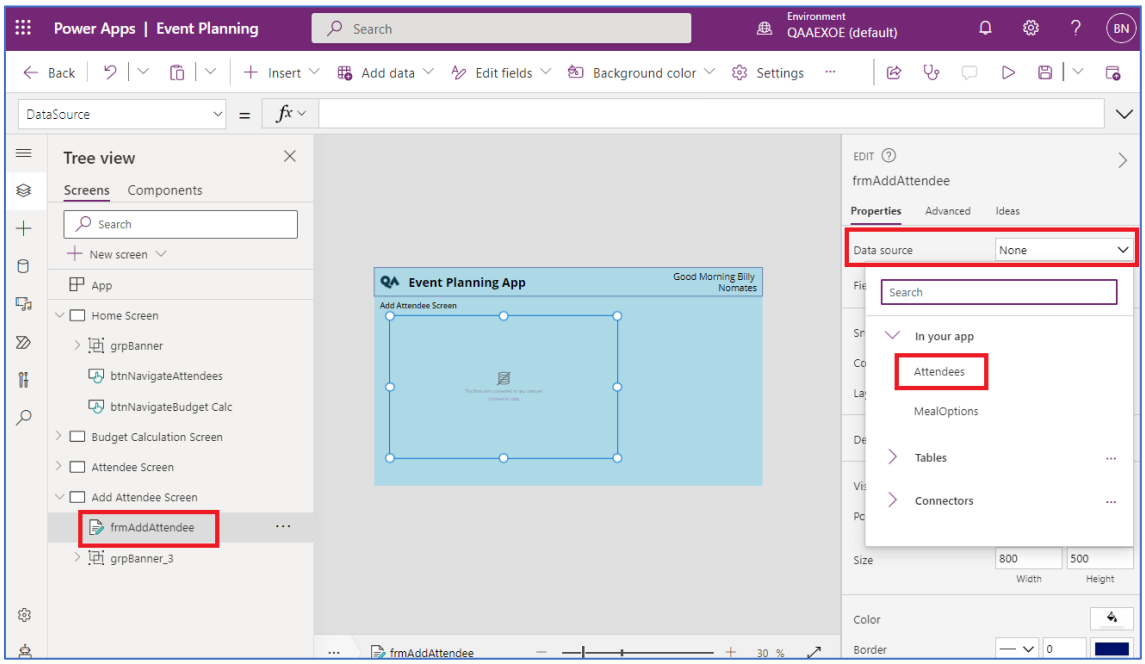


9. Using the Insert button, add an Edit form.



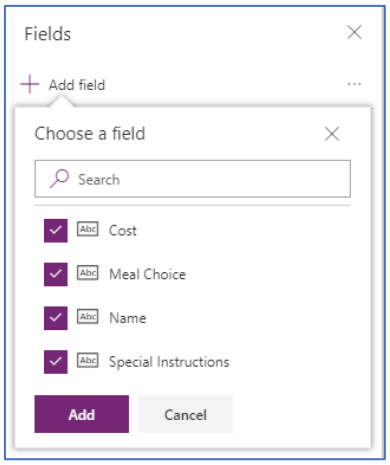
10. Move the edit form below the screen title and then **rename** it **frmAddAttendee**.

11. With the form still selected, use the **Data source** drop-down in the properties pane to select **Attendees**.



12. Click **Edit fields** (just below Data source) and then click **Add field** in the Fields pane that appears.

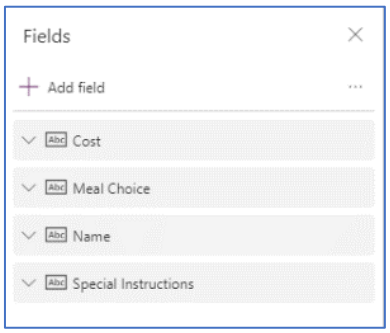
13. **Select all** four fields and then click **Add**.



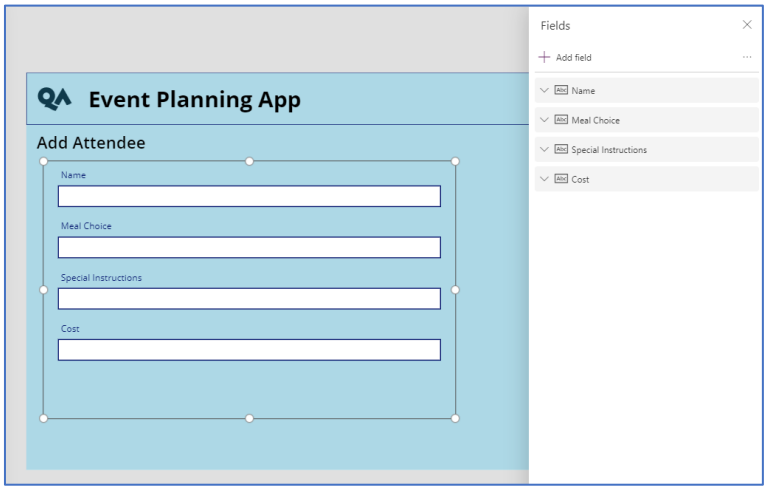
14. The fields will now appear in the form.

15. With the form still selected, use the properties pane to set the **columns property** to **1**.

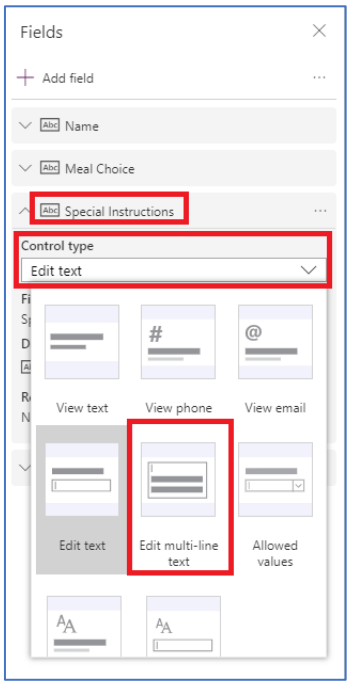
16. To make re-ordering the fields easier, minimise all four fields in the fields pane.



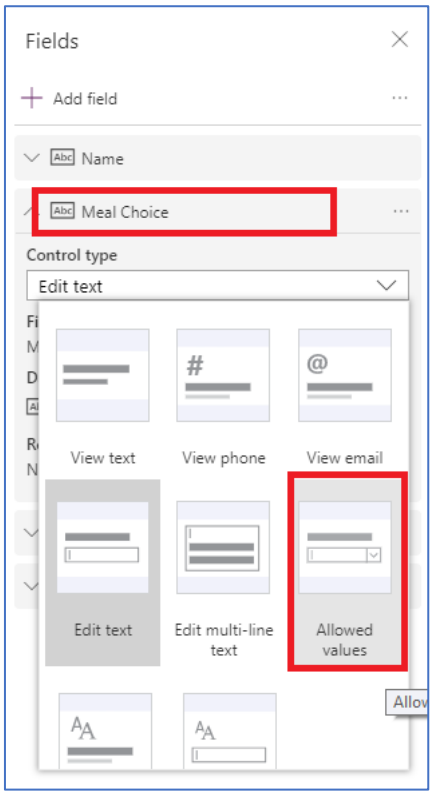
17. Drag the fields in the fields pane so they are in the order: Name, Meal Choice, Special Instructions and Cost. Note that the order also changes on the form.



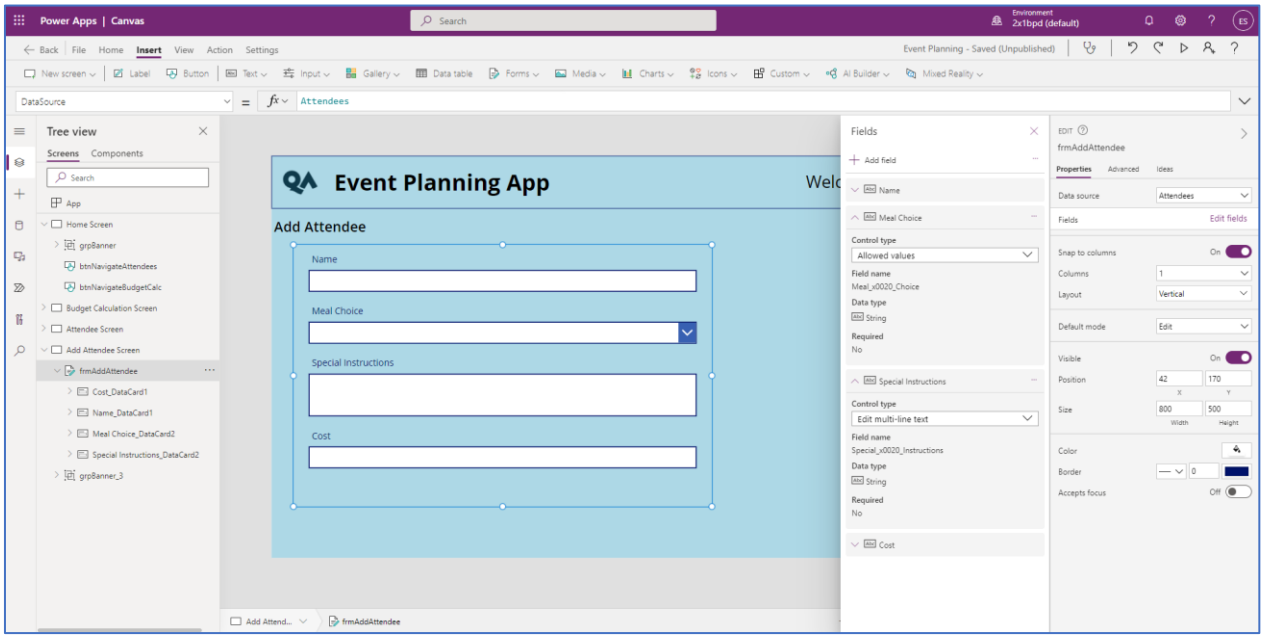
18. The Special Instructions field may need to contain several lines of text. To allow for this, **expand** the **Special Instructions** field in the Fields pane and then under **control type** select **Edit multi-line text**.



19. We want the Meal Choice to be a drop-down option, so with the form still selected, display the Fields pane if it is closed and then expand **Meal Choice**. For control type, choose **Allowed Values**.



20. Your app should now look like the picture below:

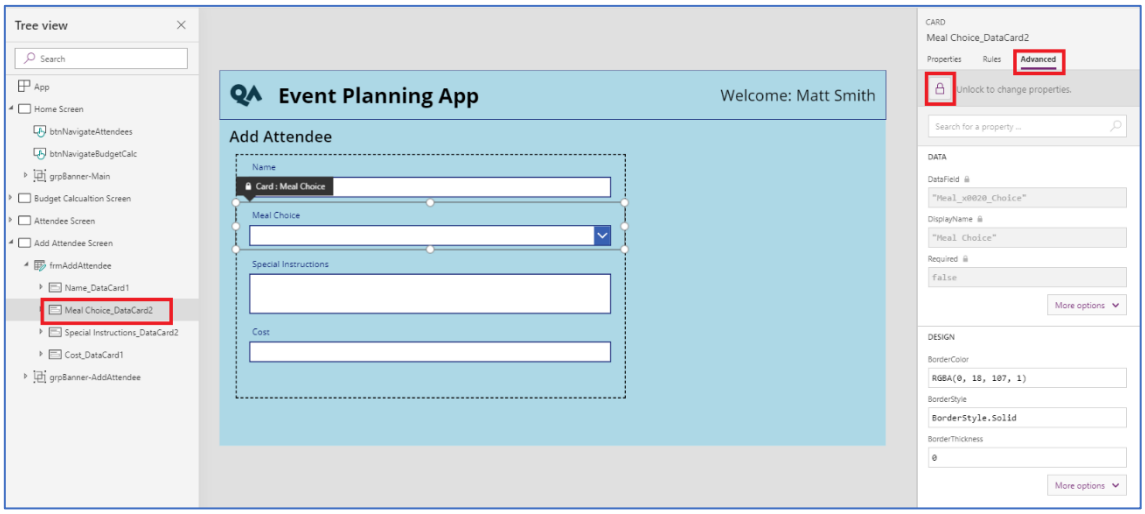


21. **Save** the app.

22. You can close the Fields pane if it is still open.

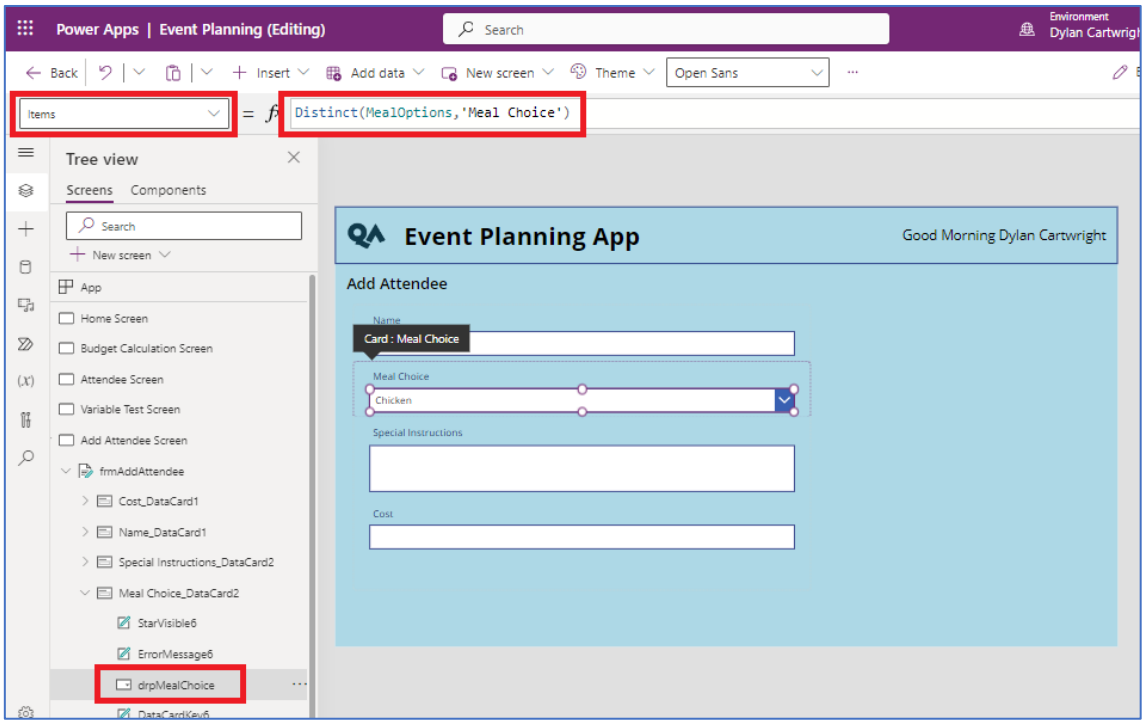
Task 2 – Populating controls

1. In the left navigation pane, **expand** the **form** and then select the **Meal Choice Data card**. In the **properties** pane, click **Advanced**, and then click the **padlock** icon to **unlock** the card.



2. Select the **DataCardValue** control inside the **Meal Choice card** and then **rename** it **drpMealChoice**.

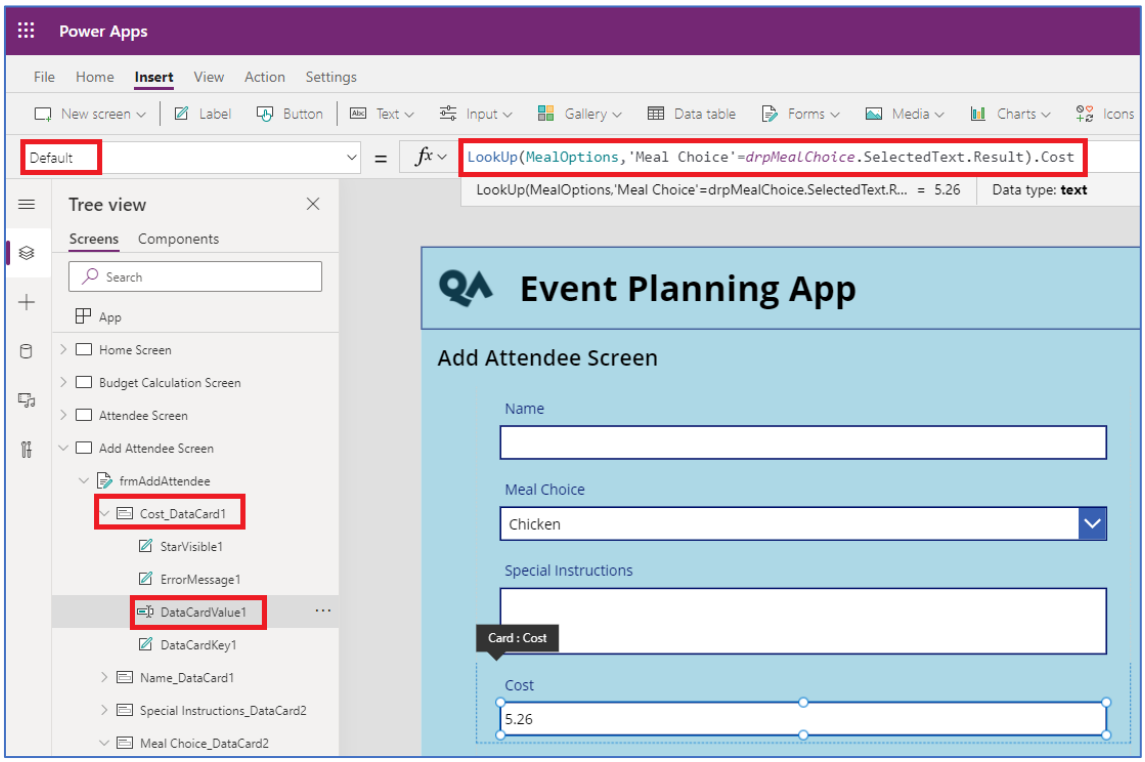
3. Edit **drpMealChoice** and set its **Items** property to read: **Distinct(MealOptions,'Meal Choice')**.



The cost will also come from saved information, but in this case, we will have to perform a lookup.

4. **Select** and **Unlock** the **Cost Data Card**.

5. Set the **Default property** of the **DataCardValue** of the **Cost Data Card** to: **LookUp(MealOptions,'Meal Choice'=drpMealChoice.SelectedText.Value).Cost**



6. **Save** and **Publish** the Form if no error makers show. You cannot test the form until some more configuration changes are made.

End of Exercise

Form Modes

Edit form have three modes of operation available to them:

- FormMode.Edit – Used to edit records.
- FormMode.New – Used to create new records.
- FormMode.View – Used to display a record.

The initial state of the form can be set by using the **DefaultMode** property of the form. The form's mode can be changed by running one of the following functions:

- EditForm()
- NewForm()
- ViewForm()

These three functions take the name of the form as a parameter to the function.

Although Power Apps has had a View Form since its release, the current design patterns we see from Microsoft use Edit forms in View mode to provide this functionality. The current release of "Modern Controls" only has a single type of form (known as a 'Form') and it supports the same modes as the classic Edit Form.

NewForm function

This function is required when you want to use a form to create a new data entry in the data source.

The NewForm function changes the Form control's mode to FormMode.New. In this mode, the contents of the Form control's Item property are ignored, and the default values of the Form's DataSource property populate the form.

If the SubmitForm function runs when the form is in this mode, a record is created, not changed. This function is typically tied to the OnSelect property of a button or icon used to start the process of entering a new record in your app.

```
Syntax
NewForm(FormName)

FormName - Required. Form control to switch to FormMode.New mode.

Example
NewForm(frmAddAttendee)
```

SubmitForm function

When you are making changes to data (including adding a new record) using a form, you call the SubmitForm function to invoke that change. Use the SubmitForm function in the OnSelect property of a Button control to save any changes in a Form control to the data source.

Before submitting any changes, this function checks for validation issues with any field that's marked as required or that has one or more constraints on its value.

```
Syntax
SubmitForm(FormName)

FormName - Required. Form control to submit to the data source.

Example
SubmitForm(frmEditAttendee)
```

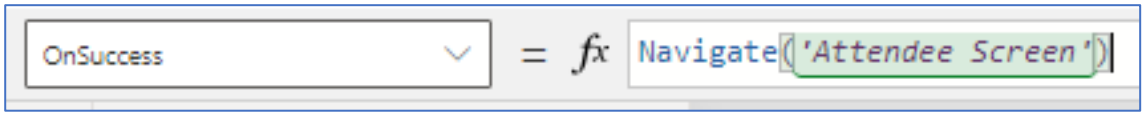
OnSuccess and OnFailure

Forms have properties that will be evaluated after the data in a form is submitted to the data source. If the submission is successful, then the OnSuccess property is evaluated, if the submission is not successful, then the OnFailure property is used.

These properties can be used to check for errors and then, if necessary, take appropriate actions.

Typical actions after a successful form submission include:

- Navigating to another screen
- Changing the form mode



ResetForm function

Should you need to undo any changes a user has made to a form, it can be done with the Reset Form function. This function resets the contents of a form to their initial values, prior to the user making any changes.

```
Syntax
ResetForm( FormName )


FormName - Required. Form control to reset to initial values. Also
switches the form from FormMode.New mode to FormMode.Edit mode.


Example
ResetForm(frmAddAttendee)
```

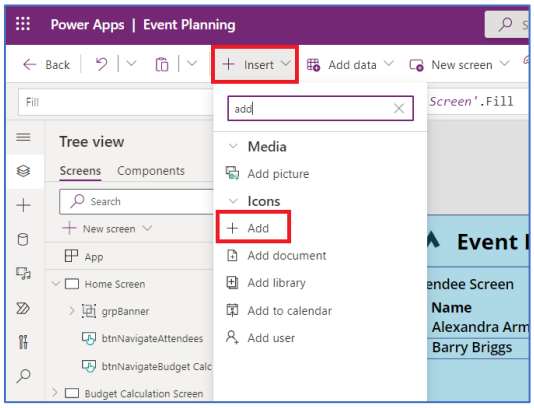
Chaining functions

There may be times when you want to run multiple functions within the same control property, for example, you may want to set a variable, switch the mode of a form, and navigate to a new page when you press a button. To do this you can chain several commands or functions with the **;** separator.

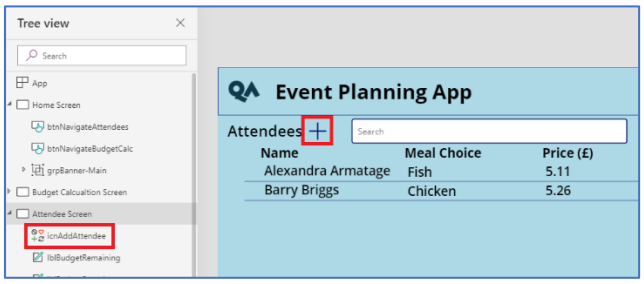
Exercise 13: Working with forms

Task 1 – Adding a record with a form

1. Switch to the **Attendee Screen** and insert an **Add icon**.

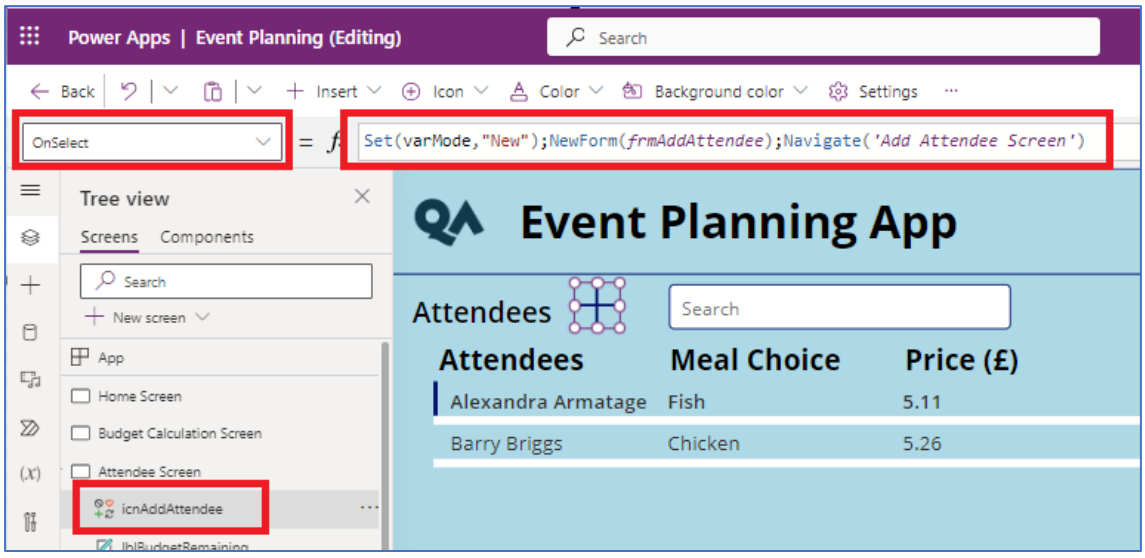


2. **Rename** the icon **icnAddAttendee** and then **move** and **resize** it positioning it in-between the Screen name and the search box.



3. Set the **OnSelect** property of **icnAddAttendee** to the following code:

```
Set(varMode,"New");NewForm(frmAddAttendee);Navigate('Add Attendee Screen')
```

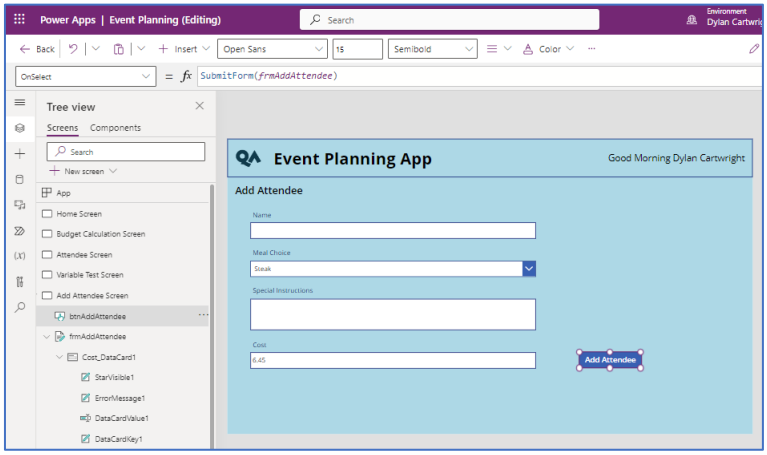


4. Return to the **Add Attendee Screen** and add a button with the following properties:

Property	Value
Name	btnAddAttendee
Text	Add Attendee

5. Set the **OnSelect** property of this button to the following code:

```
SubmitForm(frmAddAttendee)
```



6. Select **frmAddAttendee** and configure its **OnSuccess** property to the following code:

```
Navigate('Attendee Screen')
```

7. Select the **Home screen** and then **Preview** the app. When it is running, **Calculate a Budget** and then visit the **Attendees** page. Click on the **Add** icon and add an **attendee**.

8. **Close** the preview and return to the **Add Attendee screen**.

9. **Save** your app.

Now we have proved that the functionality of the form works, we will hide the cost card so that the value cannot be edited manually.

10. Navigate to the **Add Attendee Screen** the select and expand the form in the Tree view.

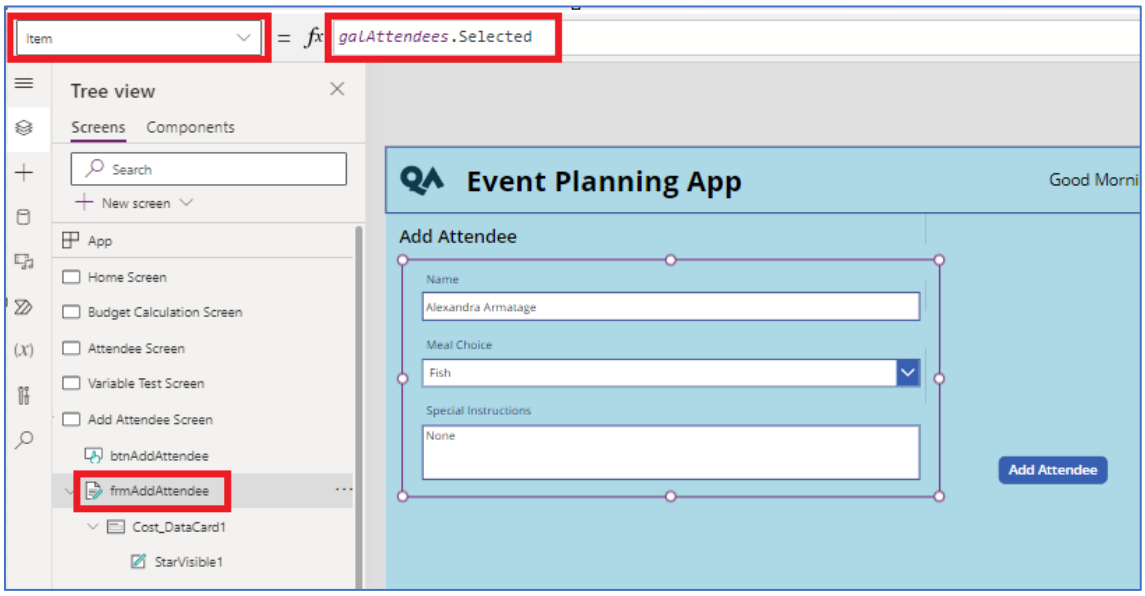
11. Select the **Cost Data card** in the form and set the **visible** property to be **false**.

12. **Resize** the form if needed, as the cost is no longer shown.

13. **Save** the app.

Task 2 – Editing a record with a form

1. Navigate to the **Add Attendee Screen**.
2. Select the form **frmAddAttendee** and set the **Item property** to **galAttendees.Selected**.

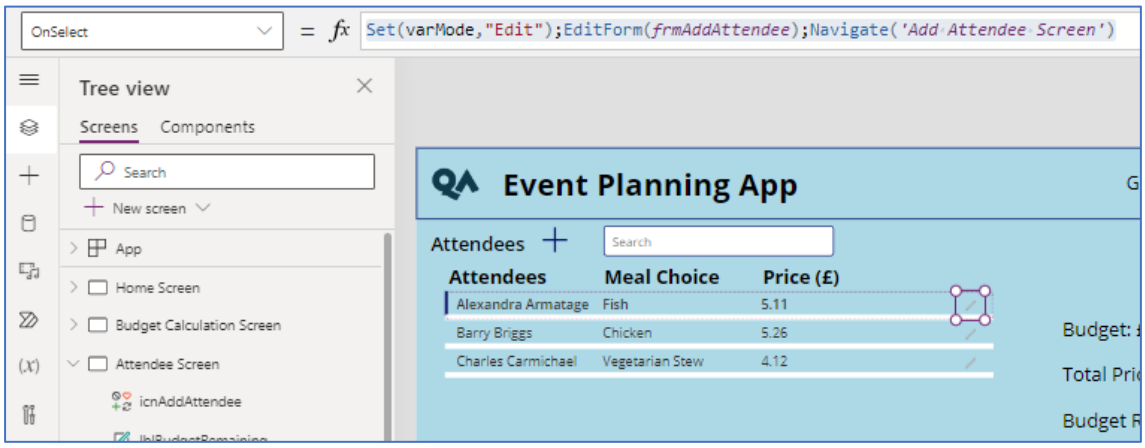


3. Change the **Text** property of **btnAddAttendee** to the following code:

```
lblScreenName_AddScreen.Text
```

4. Switch to the **Attendee screen** and select the gallery there.
5. Click the **Edit** gallery icon in the top left of the gallery.
6. Select the edit icon (**icnEditAttendees**) and change the **OnSelect Property** to:

```
Set(varMode,"Edit");EditForm(frmAddAttendee);Navigate('Add Attendee Screen')
```



7. **Save** the app and test it by previewing it.

Deleting records

The **Remove** function can be used to delete a record from a data source. This function requires you to specify the data source and which record (or records) should be deleted.

When specifying the record that is to be deleted, several methods are available which include:

- Using a Lookup function.
- Referring to the selected property of a gallery – generally used from a control outside of the gallery.
- Using ThisItem – used from a control inside the gallery where the record exists.

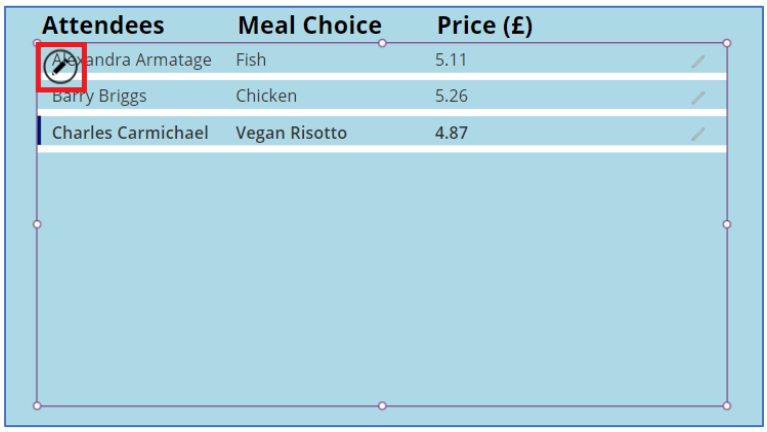
Remove Function

Deletes a record from a data source.

```
Syntax


Remove( DataSource, Record(s))


DataSource – Required. The data source that contains the record or
records that you want to remove.

Record(s) – Required. The record or records to remove.
```

Exercise 14: Adding removal functionality to the app

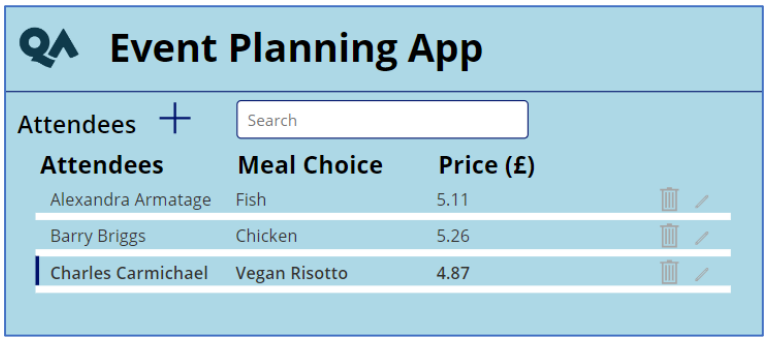
1. Navigate to the **Attendee Screen**.
2. **Select** the gallery, **galAttendees**, and then **select** the **Edit Gallery icon** that appears in the top-left of the gallery to select the gallery template.



3. Ensuring that the template is still selected, **insert a Trash icon**. Multiple instances of this icon should appear inside the gallery. If only one icon appears outside the gallery, delete it, and try again.

4. Resize the trash icon and position it to the left of the edit icon in the gallery.

Property	Value
Name	icnDeleteAttendee
Height	30
Width	30
Color	RGBA(166,166,166,1)



5. Set the OnSelect property of icnDeleteAttendee to the following code:

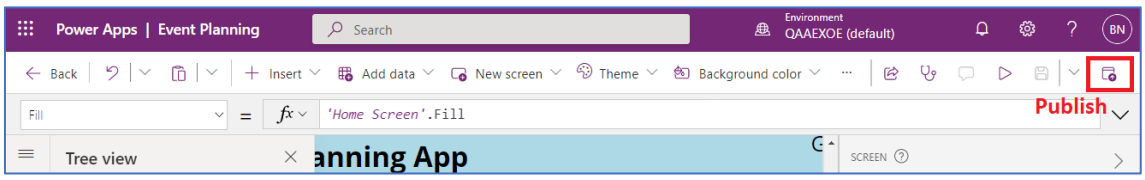
```
Remove(Attendees,ThisItem)
```

6. Save the app and then test the delete functionality.

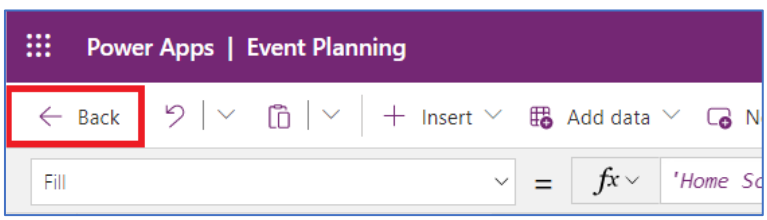
End of Exercise

Exercise 15: Working with versions

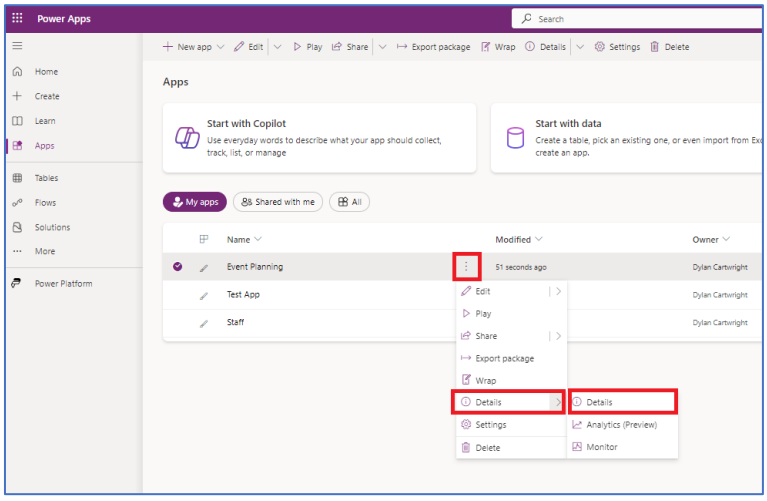
1. Using the **Publish** button, publish your app.



2. Select the **Back** button inside Power Apps.

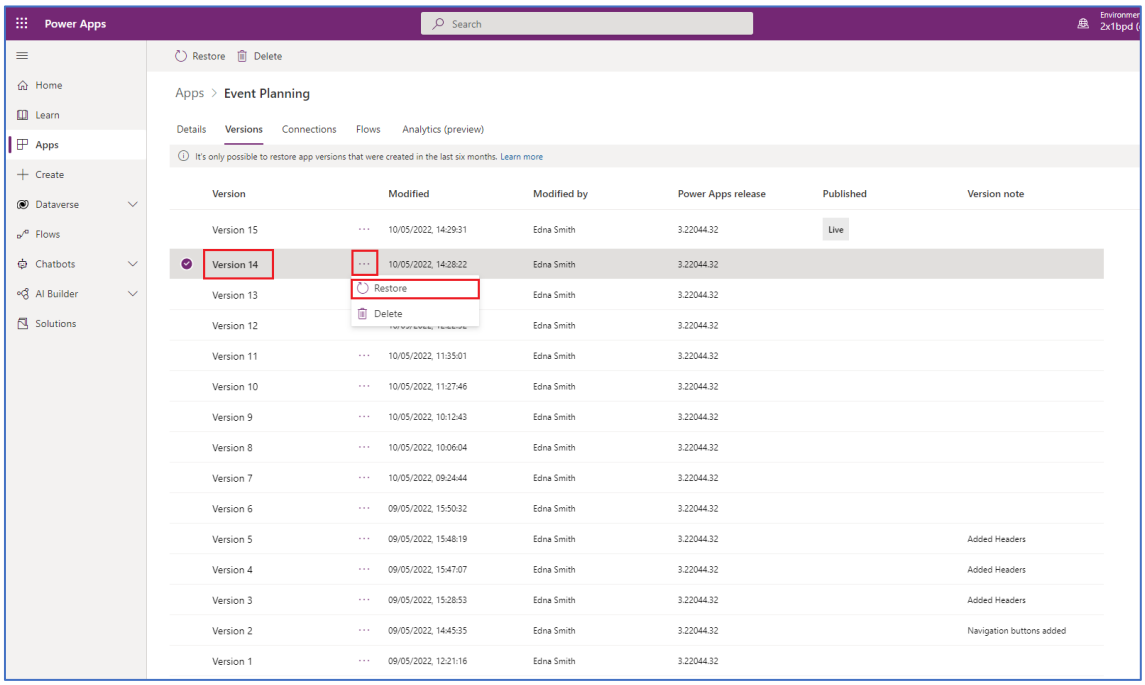


3. If prompted, select **Leave**.
4. Switch to the browser instance for your user account and run the app to test it. Using the link in the Outlook email is usually the easiest way to run this app.
5. When you are happy, close the app.
6. Switch back to your Maker browser instance and edit the app.
7. Change the fill property of the Home Screen, then confirm that this affects all the screens.
8. Save and Publish the app. Close the tab.
9. Run the app as your Test User and confirm you see the change.
10. Close the app.
11. As your maker account return to the Power Apps studio and select the Apps item in the left navigation.
12. Click the **more** icon beside your app and then select **Details** and then select **Details**.

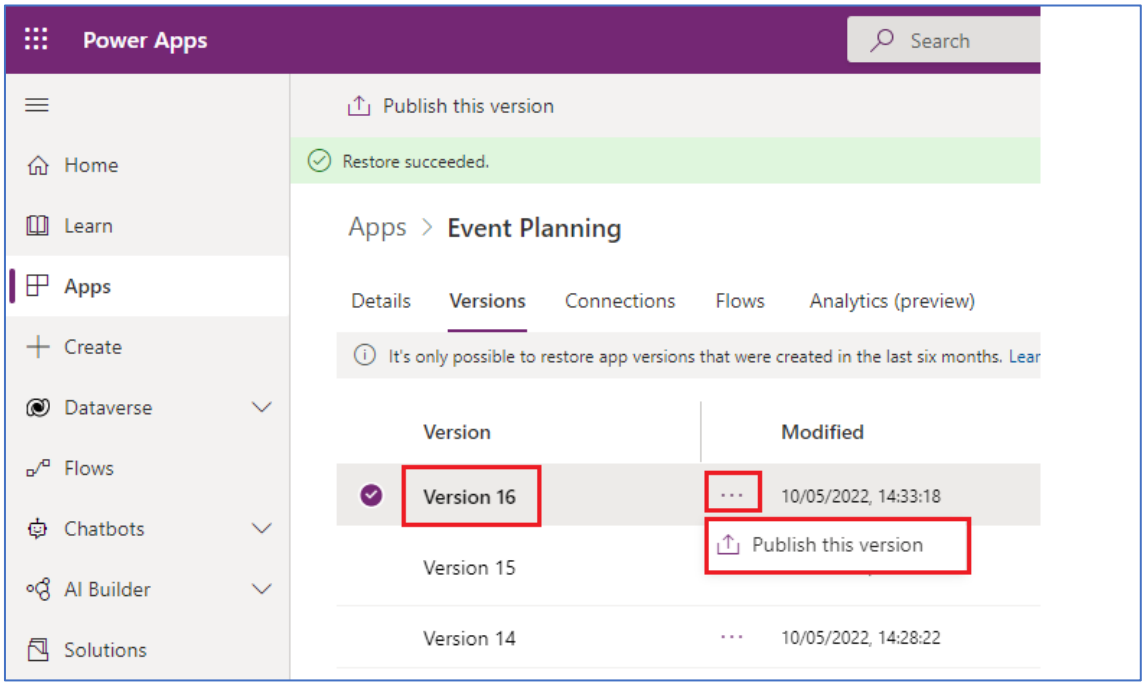


13. Select **Versions**.

14. You should see several versions, with the latest one showing as Live. Select the more icon beside the previous version and then choose restore.



15. Select the more icon beside the new version that has just been created and select Publish this version.



16. Click **Publish this version**.

17. Re-run the app as your user account and confirm that the recent changes have been undone.

End of Exercise

Appendix A – Licensing and data sources

Licensing

For full information about licensing, please refer to a Microsoft Licensing professional, alternatively the Power Platform licensing guide can be found at https://go.microsoft.com/fwlink/?LinkId=2085130&clcid=0x409.

For a user to create a Power App, or to run a Power App, that user must have a Power Apps license assigned to their account (Entra ID account) used to log into Office 365. There are several types of Power Apps licenses available, we will mention some of the more commonly used ones here.

- **Power Apps for Microsoft 365** – When available, this license comes as part of the users' Office 365 or Microsoft 365 subscription. It is available with the following subscriptions: Microsoft 365 (E3, E5 & F3), Office 365 (E1, E3, E5 & F3). It is designed to allow Power Apps to be used with Office 365 services and provides access to the standard connectors only.
- **Power Apps Premium** – This license is assigned to users and will grant them access to Premium and custom connectors, the on-premises data gateway and access to Dataverse. The user can use these additional features in an unlimited number of apps.
- **Power Apps per app** – This license grants a user the same functionality as the Power Apps Premium license, but only for a single Power App.
- **Power Platform Developer plan** – This is a free license which allows unlimited apps using Dataverse and premium connectors. However, this plan is restricted to building and testing apps to validate prior to production. Once the apps are distributed for production purposes, licensing from the other plans will be required.

Dataverse Overview

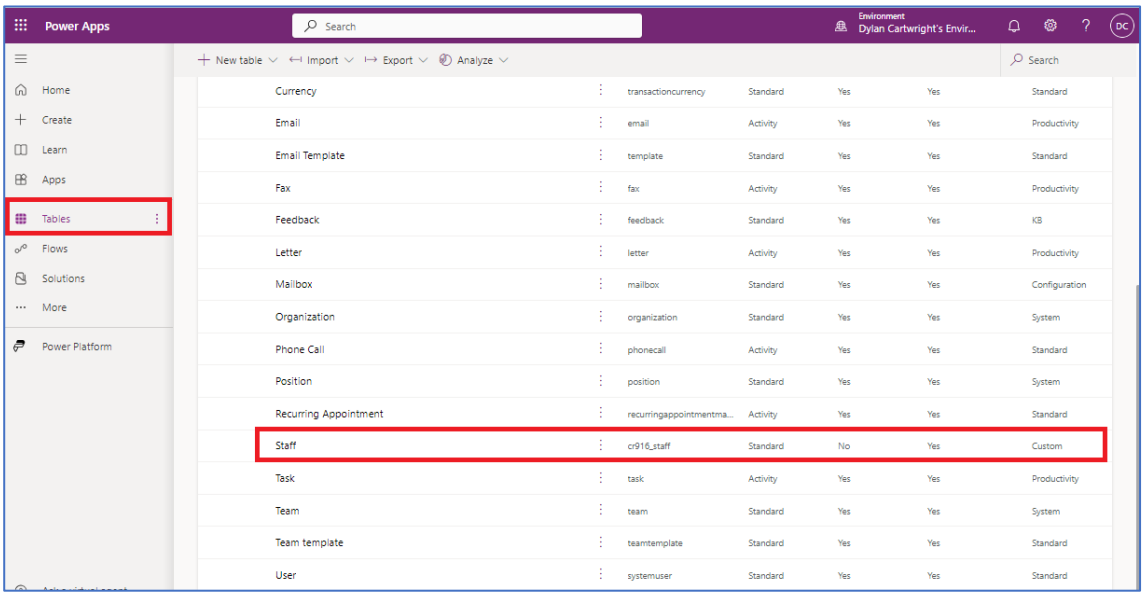
Dataverse (formerly known as the Common Data Service or CDS) is a cloud-based storage system available with some licensing levels.

It provides a centralised location not only to store your data but also entity metadata (including business logic, workflows, views, and forms) that can then be used to develop apps based on this information, including Power Apps Model Apps. Dataverse is also used as the foundation of the data storage of Microsoft Dynamics 365.

If you have access to Dataverse, then you can examine and edit the tables in this database from the Tables option in the Power Apps navigation pane.

Dataverse tables come in several types: there are many tables created automatically but we can create our own custom tables to store data in if we require. When we did

the first exercise on this course (imported data from an Excel spreadsheet) it created a Dataverse table (called Staff) for us as part of this process.



As we move on to look at CoPilot with Power Apps, much of what it can do for us will involve using Dataverse, this will then require a premium level of licensing.

Choosing Data sources

In many situations, you will have no control where the data you are accessing is stored and you will create your app to access that data where it is. However, there may be situations where as well as designing the app, you can choose where the data the app is accessing is being stored. In this section we will examine some of the more commonly used data sources and highlights their strengths and weaknesses.

Excel

Excel is a simple data source to use with Power Apps, however using it can be limiting. To use Excel, the data must be stored in a table created in the file. The file can then be stored in a SharePoint document library or on OneDrive.

Licensing – Standard

Location – Cloud: SharePoint or OneDrive.

Pros

- Simple to use.
- All data in a single file.
- Easy to back up.

Cons

- If file is opened it will be 'locked' and Power Apps cannot access it.
- Does not support server-side processing (will have performance issues on larger data sets).
- Not a relational database

Performance

- Limited to 2000 rows of data.
- File maximum size 2MB.
- 100 API calls per minute

Notes – Data must be stored in a table created in the file.

SharePoint

SharePoint is the best storage option when using standard connectors. A list needs to be created in a SharePoint site to store the data.

Licensing – Standard

Location – Cloud or on-premises (on-premises will require data gateway which is a premium feature)

Pros

- Supports data types.
- Supports mandatory fields.
- Includes metadata fields.
- Items and documents can be accessed with unique URLs.
- Support server-side processing.

Cons

- Not a relational database (but lookup columns supported).
- Not all functions support server-side processing (e.g. Search).

Performance

- 30 million rows of data (per list)
- 600 API calls per minute

SQL Server

If your organisation is already using SQL server and the data is present there already, then this will be the best option to use. However, where the data is not present it will generally be easier and cheaper to use another option.

Licensing – Premium

Location – Cloud or on-premises

Pros

- Rich relational database.
- Feature rich (including data validation and auto-incrementing fields).
- Good delegation functionality.
- Can be optimised with joins and views.

Cons

- More complex to integrate with Power Apps.

- Working with attachments not straight forward.
- SQL licensing required (will not be part of Office 365 license).

Performance

- Supports 2 billion objects.
- 600 API calls per minute.

Dataverse

Unless the data is already present elsewhere, the Dataverse is the best option to store your data for Power Apps when you have access to premium licensing.

Licensing – Premium

Location – Cloud

Pros

- Integrates well with Power Apps
- Rich relational database.
- Feature rich (including data validation and auto-incrementing fields).
- Good delegation functionality.
- Easier to setup than SQL
- Support environments for ALM.

Cons

- Does not scale as well as SQL server.

Performance

- Unlimited data size (constrained by licensing)
- 1200 API calls

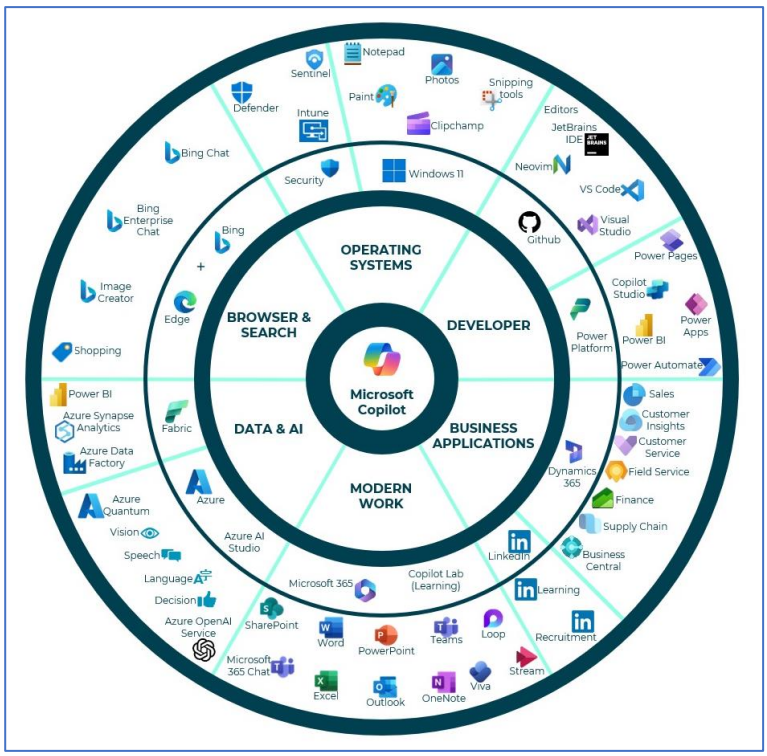
Appendix B – Power Apps Copilot

What is Copilot?

Copilot is Microsoft's AI powered tools that use GPT. It leverages the power of AI to provide a range of services, from answering questions and providing information to creating content.

What Copilots are there?

Copilot is not a single thing, there are many Copilots providing AI functionality ...



Licensing Power Apps Copilot

There is no additional licensing cost to use Copilot with Power Apps, it comes with the Power Apps license that you are using. However, if you use this Copilot to create tables to store your data in, it will do that in Dataverse.

To use Dataverse, you will need a level of Power Apps licence that includes these use rights, for example a Power Apps Premium licence.

What are prompts?

A prompt is an instruction or question you use to tell Copilot what you want. It's like having a conversation, using plain but clear language and providing context like you would with an assistant.

Prompts can include four parts:

- **Goal**: What response do you want from Copilot?
- **Context**: Why do you need it and who is involved?
- **Source**: Which information sources or samples should Copilot use?
- **Expectations**: How should Copilot respond to best meet your expectations?

For example, here's a prompt that includes a goal and source:

"Write a summary based on all emails from Sam in the past two weeks." And here's an example that includes a goal, context, and the expectations: "Draft an outline of a training manual about time management. Our audience is professionals who work in a hybrid environment and constantly need to attend virtual meetings and meet deadlines. The tone of the document will be friendly and suggestive."

It's important to always review and verify the responses you get from Copilot. Using the exact same prompt multiple times can result in different responses. You can use Copilot to create or edit content, ask questions, summarize information, and catch up on things.

Writing effective prompts for Power Apps

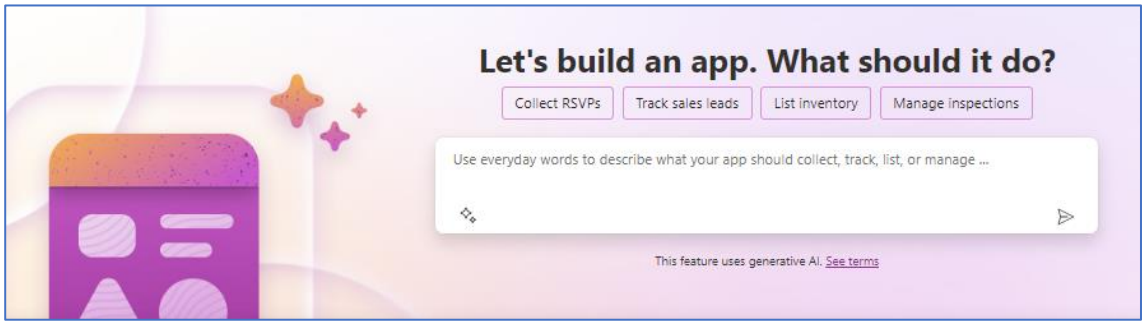
- Describe your app's functionality using simple terms.
    - **Example**: "Collect team event feedback"
    - **Example**: "Track new employee onboarding"
- Modify the data table using prompts such as:
    - **Example**: "Add a new column for..."
    - **Example**: "Delete the ... column"
    - **Example**: "Change the data type for the ... column to..."
- If assistance is needed, request help from the AI assistant by typing "Give me suggestions".

Using the Power Apps Copilot

Copilot for Power Apps can work in two ways. It can create Dataverse tables for you and it can create (and modify the canvas apps).

Creating tables for your app to use

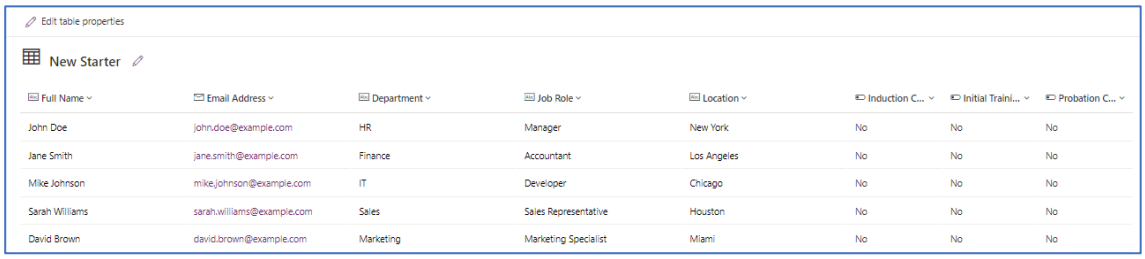
On the home page of Power Apps is a prompt area with the message "Let's build an app. What should it do?" Using this prompt will start the process by defining a new Dataverse table in your current environment.



The Copilot will suggest columns for the table based on the prompt you give it, but after the initial suggestion you can make changes to the table structure including adding columns, removing columns, and changing the data type of columns. The Copilot will also provide some sample data which will help you visualise the data, and how it will look in the app when that is generated.

Create an app to track new starters in the business. The app should record: Full Name, Email Address, Department, Job Role, and Location. There should also be yes/no fields for Induction Complete, Initial Training Complete and Probation Complete.

A prompt such as the one above can provide a suggest table such as the one shown below.



Full Name	Email Address	Department	Job Role	Location	Induction C...	Initial Traini...	Probation C...
John Doe	john.doe@example.com	HR	Manager	New York	No	No	No
Jane Smith	jane.smith@example.com	Finance	Accountant	Los Angeles	No	No	No
Mike Johnson	mike.johnson@example.com	IT	Developer	Chicago	No	No	No
Sarah Williams	sarah.williams@example.com	Sales	Sales Representative	Houston	No	No	No
David Brown	david.brown@example.com	Marketing	Marketing Specialist	Miami	No	No	No

There will be a Copilot pane on the right-hand side of the browser, prompts can be used here to make changes to the structure of the table.

Change the Department to be a choice.

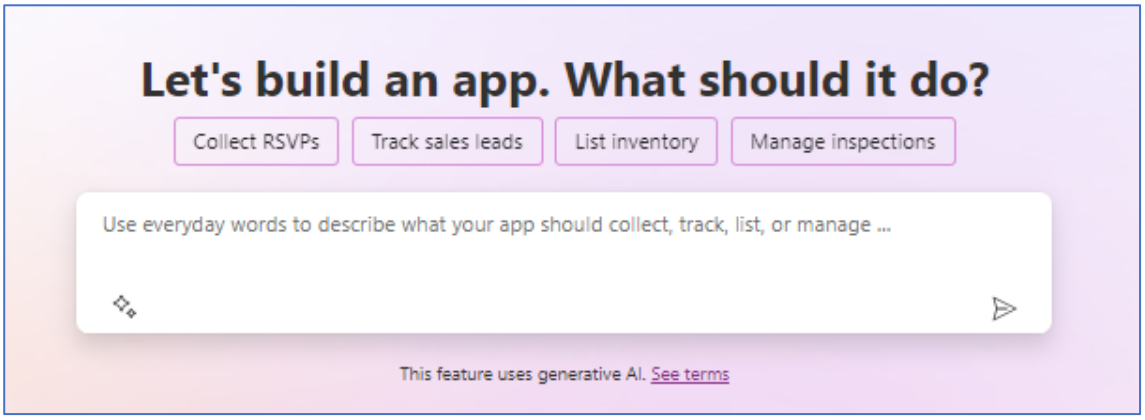
If the prompt is understood, you should see a change to the table structure.

Once you are happy with the structure of the table, you can select the **Create App** button. This will save your Dataverse table and then build an app based on that data structure. The app can then be customised with CoPilot which is covered in the next section.

Exercise 17: Creating an app with Copilot

Note – As Copilot uses generative AI it is constantly learning. This means that when you try these tasks, the responses you receive from Copilot may be different to those shown in the manual.

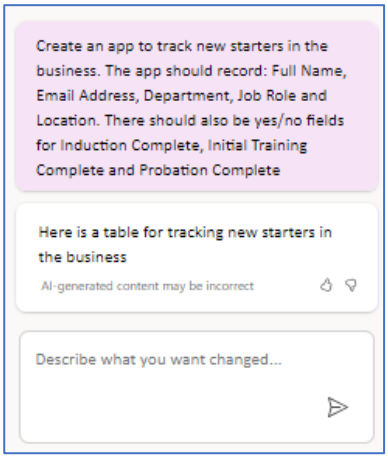
1. Navigate to the Power Apps home page (https://make.powerapps.com) and ensure that the Home icon is selected in the navigation pane.
2. Ensure that you can see the Copilot input panel at the top of the page.



3. In the prompt box type the following prompt and then select the 'Go' icon.

> Create an app to track new starters in the business. The app should record: Full Name, Email Address, Department, Job Role, and Location. There should also be yes/no fields for Induction Complete, Initial Training Complete and Probation Complete

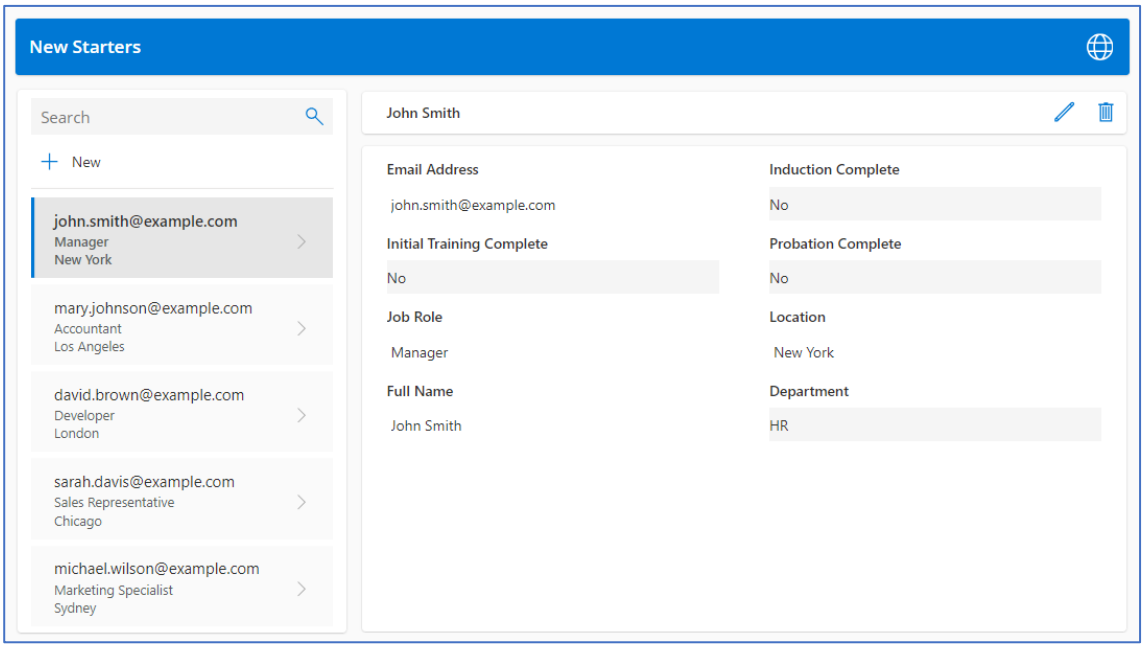
4. A suggested table will be shown complete with some data.
5. Locate the Copilot pane (on the right-side of Power Apps) and confirm that your prompt and a response is present.



6. If you look at the Department field in the table, you will notice that it is text (Abc icon). Change it to be a choice by typing the following prompt.

> Change the Department to be a choice.

7. Confirm that this change is made.
8. If you are happy with the changes, select the **Create App** button at the bottom of the Copilot pane.
9. After a few moments the app will be created.



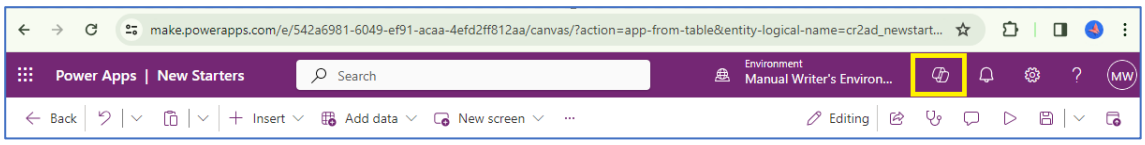
10. Save the app.

End of Exercise

Editing your app using Copilot

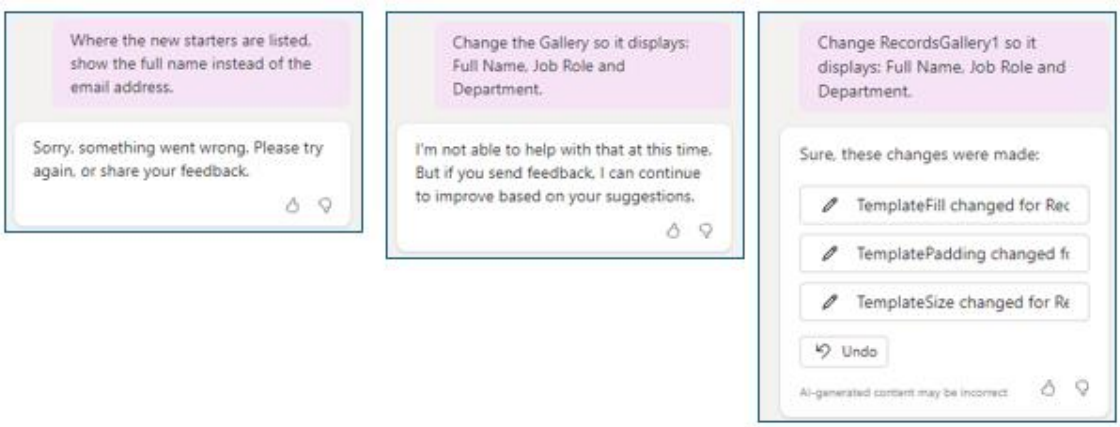
Editing your app with Copilot will not create new tables in Dataverse, so can be used with any licensing level of Power Apps. To access this functionality, use the copilot pane on the right-hand side of the app.

If the Copilot pane does not open automatically, it can be toggled with the Copilot icon on the purple Power Apps bar.



Unfortunately, at the time of writing, editing an app with Copilot is less than ideal. There are some sample tasks that appear in the Copilot pane as buttons such as "Add a text label" and "Add a button". Using commands as simple as these are generally successful, however attempts to modify properties of controls are often not successful.

From current experience it has been found that the only way to successfully configure controls using Copilot is to write a prompt that is so detailed it would have just been quicker and easier to edit the control directly.



In the next exercise you will have an opportunity to try making some modifications to your app by using Copilot. Some suggested prompts are given, feel free to try some other variations. Do not save changes to the app when you have finished.

Appendix C – Practical Application Exercise

If you have completed the other exercise on the course, and there is additional time available to you, please have a go at the following exercise.

This exercise does not have step by step instructions, rather just a summary of the desired result. It is up to you to apply the lessons learnt on the course to produce a working solution.

Overview

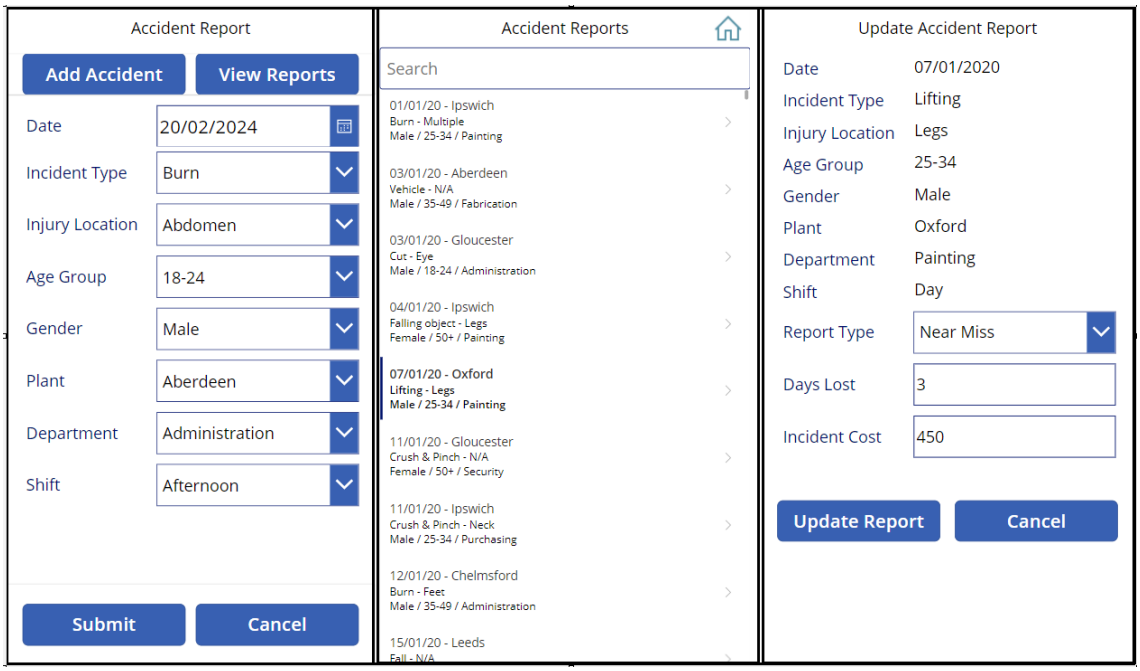
Your organisation has asked you to produce an app that can be used to add and edit safety data recording accidents in the organisation. The current data is stored in an Excel file (SafetyData.xlsx) a copy of which should be in your student files.

Data location

For this exercise, it does not matter where you choose to store this data. If you wish to carry on using Excel as the data source, then you will need to upload the file to OneDrive. If you wish to use either SharePoint or Dataverse as the data source, then you will need create the necessary Lists / Tables in those locations.

App functionality

The app should have at least two screens. The first screen should be used to allow the addition of new Accident reports. The other screens should be used to browse, search, and edit existing accident reports. You can use a single screen for the browse, search and edit or you can split this over multiple screens if you prefer.



Appendix D - Function index