Contents

About the Author

Carlos Aguilar Mares is the Partner Group Engineering Manager leading the PowerApps engineering team part of the Cloud and Enterprise Team at Microsoft in charge of creating the best platform for creating business mobile applications easily without code. Previously he led the Azure team in charge of the end-to-end experience of discovering (http://www.windowsazure.com/) **signup**, **billing** (https://account.windowsazure.com/), **management portal** (https://manage.windowsazure.com/), **monitoring**, **new management portal** (https://portal.azure.com/), Azure Resource Manager, Windows Azure Pack, Azure Logic Apps, and more. Previously he was the Development Manager for **Internet Information Services (IIS)** product group at Microsoft that developed the IIS service in Windows, offering a high performance, scalable, and extensible platform for hosting Web applications and services and other products like Web Matrix, Web Platform Installer, Web Deploy, URL Rewrite, and more. Before that, he worked on the ASP.NET 2.0 Page and Controls team and before that, he was a Development Consultant working in Microsoft Consulting Services Mexico where he is originally from and where he joined Microsoft over sixteen years ago. When not at work, Carlos enjoys being with his family as well as playing tennis, guitar, Xbox, coding games, and pretty much anything related to computers. Co-Author of IIS 7.0 Resource Kit book.

Chapter 1 - Getting Started

What is PowerApps

Microsoft PowerApps is a new platform to create line of business mobile applications that can be built in minutes without the need of writing code (also known as No-code/Low-Code platform) or worrying about all the complexity involved in writing cross platform applications that support iOS, Android, Windows Phone, Windows and the web. PowerApps allows you to easily build applications that connect to several data sources, including SQL databases, SharePoint lists, Excel Spreadsheets, Office 365, Dynamics CRM, OneDrive, Dropbox, Google Drive, Trello, Facebook, Twitter, Wunderlist, and more. But more importantly it includes a new secure business database called the Microsoft Common Data Model.

PowerApps provides a very powerful and yet simple experience that allows you to build applications with rich mobile capabilities such as access to Camera, GPS, etc. while providing a modeling experience as simple as **Microsoft PowerPoint** and providing a familiar way to express logic like **Microsoft Excel**.

Line of business mobile applications

As mobile technology continues to evolve and improve, mobile devices have taken over every interaction in the consumer/personal space where mobile applications exist to help us in every aspect of our lives - from social networks, photo sharing, interactive media, news, smart home tools and many more - almost everything we need can be found in your favorite mobile device application store. However, on the enterprise/professional space this has not yet been the case and the modern workforce struggles daily to gain access to most of their internal LOB applications running on legacy systems that are hard to expose to mobile devices, from expense reports, invoice management, absence reporting, sales tools, time tracking and many

more are examples of key data that historically has only been accessible in the internal network through intranet applications. Numerous research studies have shown that there is a huge interest by companies to empower their employees and mobilize these applications and their data for them, however, the cost and complexity of building mobile applications (not to mention supporting multiple devices and platforms) has been getting on the way of internal IT teams on scaling and keeping up with the large number of applications.

Dataflow programming

One of the first concepts to get familiar when building applications with PowerApps is the way to think when interacting with data and performing transformations in it. PowerApps uses a model called ***dataflow programming*** where the emphasis is in the movement of data and the relationship and transformations of it. In this programming model the entire application is modeled as a set of logical components that receive some data as inputs, apply a transformation or business logic to it and produce an output. Such transformations are only triggered when the values of their inputs are modified, and once the resulting output is evaluated it triggers updates on other components that use it as input. This means that you don't have to deal with large complex imperative instructions that provide the logic to push arbitrary changes to other components, but instead the individual components get updated as soon as their inputs and state changes and their calculations and business logic are applied.

This all sounds very complicated, but luckily you do not need to worry about all the complexity behind it, in fact if you have used tools such as **Microsoft Excel** or other similar tools, it will feel very familiar since they follow the same model and you use formulas and expressions to display and manipulate data.

Let's look at an example. If you are using Microsoft Excel and you have a cell that displays the price of a product and a cell that displays the discount for it, and you want

to display the final price in another cell, you select the “Total” cell where you want the output of the calculation to be displayed and enter the appropriate calculation formula in it using the other two cells values as inputs. Microsoft Excel will figure out automatically when and how to refresh the contents on the output cell and whenever any of the input values or cells used in that formula calculation are modified. You don't have to worry about the order of calculations, update logic in multiple places to update the same cell value, invalidations or anything else. Excel will even detect if you are causing circular references, data type inconsistency, and many more.

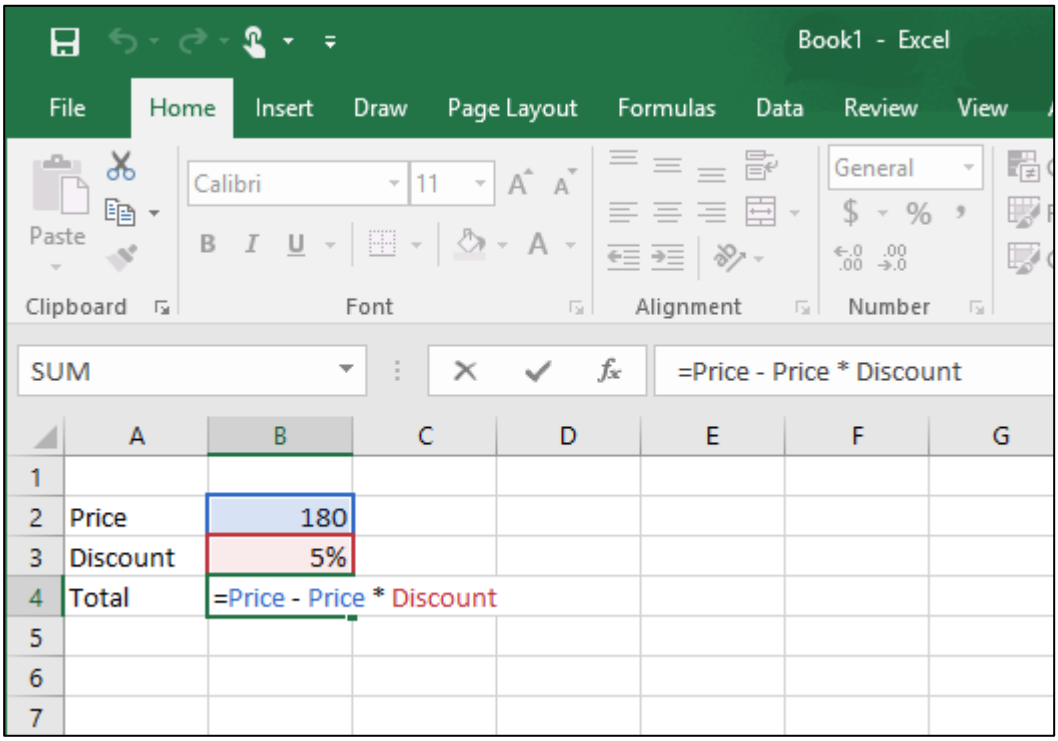

Figure 1. Microsoft Excel dataflow model

PowerApps applies the same model for building rich mobile cross platform applications providing a rich set of functions or formulas that can be used to perform calculations just like Excel, but it also extends it forward with a set of imperative constructs that can be used to update data, communicate with external data sources, navigate across multiple screens, and even provides the ability to delegate complex calculations for server side processing. In addition to these formulas, PowerApps provides a set of rich

controls to display data from simple input text boxes, barcode scanning, rich media and video controls as well as mobile input capabilities such as Camera, Location and more.

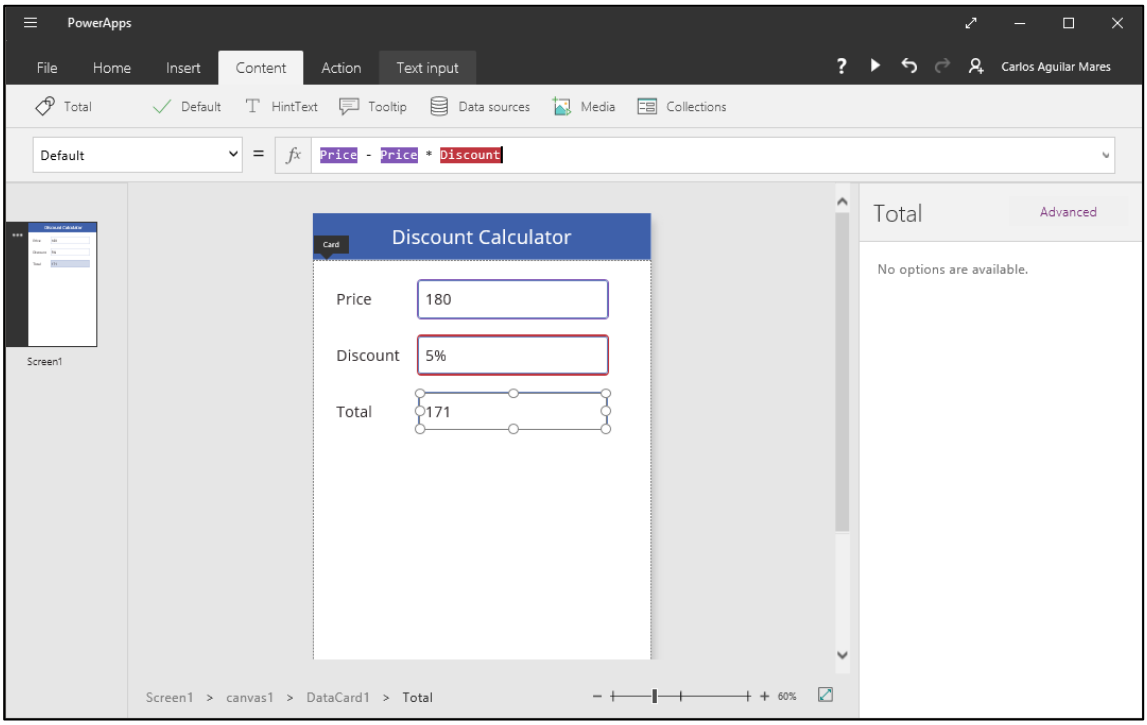

Figure 2. PowerApps dataflow model

Main components of PowerApps

PowerApps platform is made up of many components you will interact with, including:

- https://web.PowerApps.com/ – Also sometimes referred as the "*Maker Portal*", it is the portal where application creators go to manage their applications, manage the Common Data Service (entities, and more), Connections, Gateways and more.
- https://home.dynamics.com/ – Also referred as the "*Consumer Portal*", it is the place where the end users of the applications can go to find and run the applications they have access as well as discover new applications that have been shared with them by someone on their organization in Microsoft AppSource.
- https://admin.PowerApps.com/ – Also referred as the "*Admin Center*" is the portal where environment administrators can go and manage environments and policies for them.
- PowerApps Studio – This is the rich application that provides the designer environment for building PowerApps apps for Windows 8 and above that can be downloaded from the Microsoft Store.
- PowerApps Web Studio – This provides the same capabilities as PowerApps Studio, however it runs on any of the main Web browsers (such as Microsoft Edge, Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, and more) giving application creators access from any platform and any device.
- PowerApps Players – These are the Mobile players that allows users to run their applications on their mobile devices and the Web, including support for iOS, Android, Windows Phone, Windows and Web.
- Connectors and Gateways – Connectors provide connectivity to hundreds of SaaS services such as Microsoft SharePoint, Microsoft Office 365, Google Drive, Facebook, Twitter, Microsoft OneDrive, Salesforce, Slack, GitHub, Dynamics AX, Dynamics CRM, and more, but also platforms and protocols such as Microsoft

SQL Database, FTP, SMTP, RSS, and many more. Gateways allow you to connect to on premises resources such as Microsoft SQL Server or Microsoft SharePoint.

- Microsoft Flow – Provides a powerful business orchestration tool to automate long running processes in a simple point and click way, leveraging the same Connectors that PowerApps provide.
- Common Data Service – This is the rich PowerApps business data platform that provides an implementation of the Microsoft Common Data Model giving you services to build applications such as a set of standard entities, ability to extend and create custom entities, picklists, and more, as well as features to protect, interact, import and export data, including an Excel plugin, and more.

Summary

In this chapter, we talked briefly about PowerApps and the motivation for Low code-No-code platforms to help organizations provide access to the information their employees require on the go without spending months and expensive resources on building cross platform applications and allow them to deliver on the large backlog of applications demanded.

We also talked about how PowerApps provides a familiar environment that offers a combination of Microsoft PowerPoint-like designer environment while providing a Microsoft Excel-like formula and expression execution experience, giving you a simple and yet powerful platform to build line of business applications in a very rapid manner. Unlike most other rapid application development environments PowerApps provides all the power that you need, giving you not only the quick productivity, but also making sure you don't run into a cliff that blocks you from achieving more complex experiences or that locks you into only specific platforms, backend or data access. PowerApps gives you an easy way to connect to an ever growing number of SaaS services, access to on premises environments, and the ability to extend to any API with Custom APIs that developers can build and host in any platform and language, and build rich experiences that are not limited to simple forms but instead that provide rich mobile capabilities and experiences.
We finally briefly discussed some of the main components that make the PowerApps platform.

Chapter 2 - Building Applications

Creating your first application

Like any other book that explains building software, we will start with the "Hello World" of PowerApps. In the PowerApps case it would be too simple to just add a Text Box and enter it, so to make it do something more interesting we will do an application that greets you by your full name and also displays your photo (if you have configured one at (https://portal.office.com/account/#personalinfo).

To do this we will use PowerApps Studio. If you don't have it installed, you can get it by visiting https://aka.ms/PowerAppsWin. Once installed:

Run PowerApps

- In Windows go to the start menu and search for PowerApps. If you rather use PowerApps Web you can navigate to https://create.PowerApps.com/

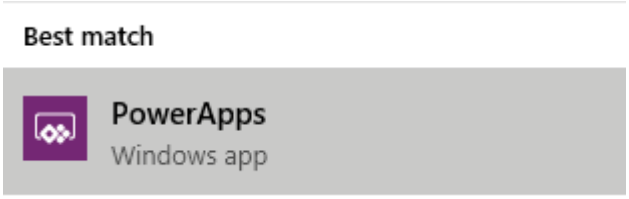

Figure 3. PowerApps icon in Windows

Start creating the new Application

There are many kinds of templates to get you started that we will cover later, but for this simple example we will use the option Blank App->Phone Layout.

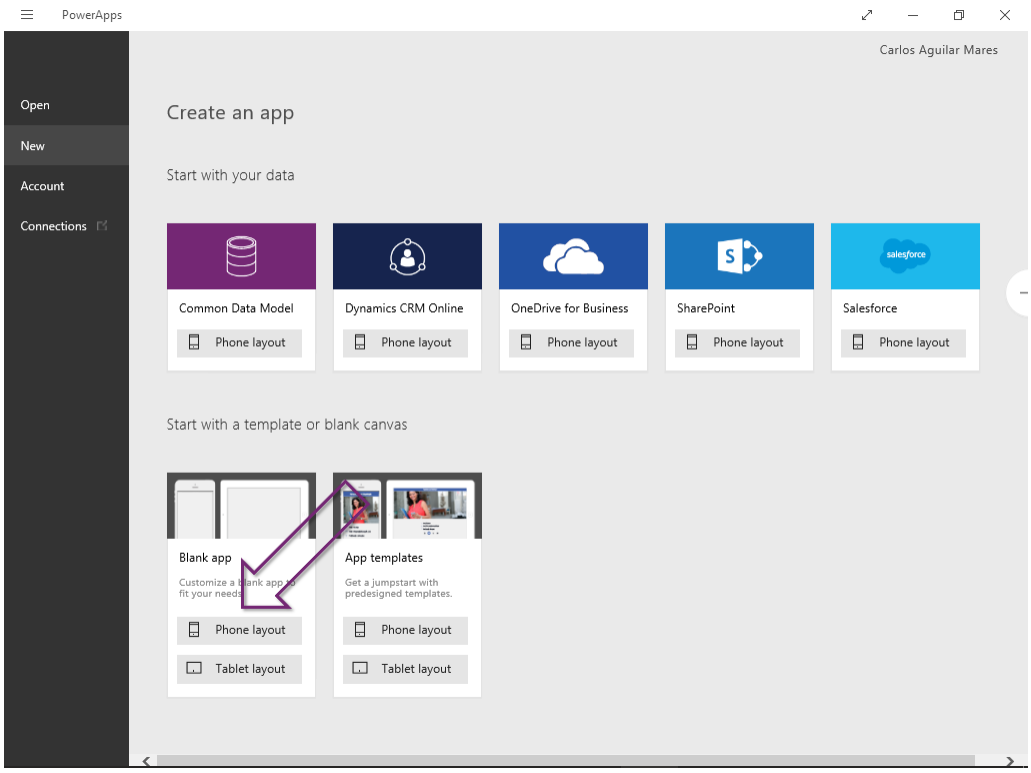

Figure 4. Create an app screen – Balnk app - Phone layout

Implement the User Interface

For this application, we will be adding a Text Box and an Image. To do that just use the Insert menu at the top, and choose Text box.

Once the Text box control is added to the Canvas modify the formula for its Text property to leverage the “User()” formula and read the FullName property of the currently logged on user by using the following expression:

```
"Hello " & User().FullName
```

This expression is concatenating the full name of the currently logged in user with the static string “Hello”.

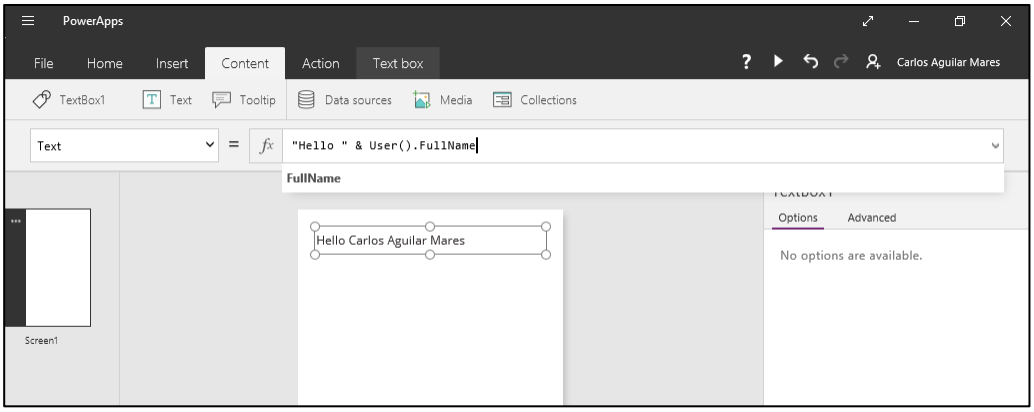

Figure 5. User() formula in action

Now add an Image control by selecting in the Insert→Media→Image control. Once the image control is added to the Canvas modify the formula for its Image property to use the "User()" formula to get the photo configured for the currently logged on user by using the following expression:

```
User().Image
```

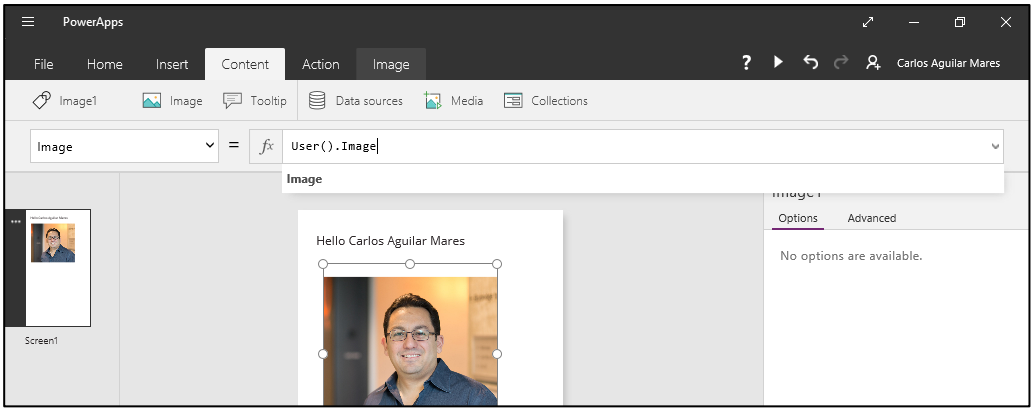

Figure 6. Reading the user's photo

Live Canvas

One thing that you can immediately see is that PowerApps provides a true live canvas experience where any changes you do are reflected in a running instance of the application. This is not an emulation of the application, in fact it is the application running embedded inside, using real data and real behavior.

Running the Application

To run the application in full screen you can press “F5”, or the “play” icon in the top right corner.

> Note: Clicking the Run icon only switches to full screen the canvas and nothing more than that, in other words the application will not restart or trigger any additional logic than the one already in the live canvas.

Save the Application to the cloud

In order for you and your users to be able to run the application on the Web or their mobile devices you will need to save the application to the cloud. You can do that by using the File→Save as option, or you can also press Ctrl+S.

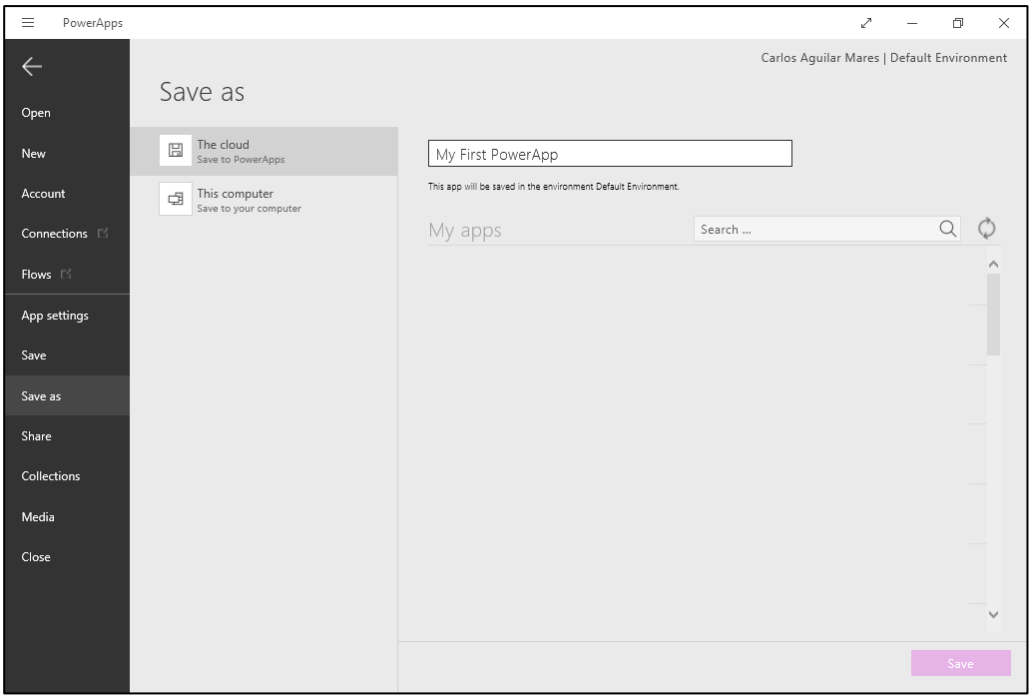

Figure 7. Save as

Running the application in your mobile device

Once the application has been saved in the cloud you can download the PowerApps player for your device in their application store. Currently PowerApps is available in iOS, Android, Windows Phone and Windows Store.

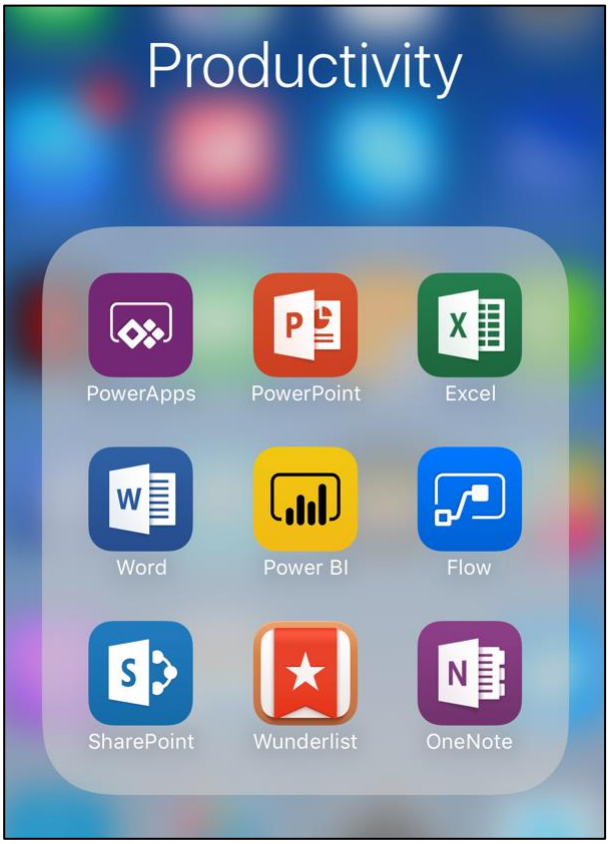

Figure 8. PowerApps in iOS

After running PowerApps you will be asked to authenticate with the service. PowerApps uses **Azure Active Directory** which provides an enterprise grade directory service with great capabilities, including the ability to federate to your on premises Active Directory so your users can utilize their company's credentials and not require additional user and password. In addition, this also means that if an employee leaves the company they will automatically lose access to the corporate applications providing a very robust solution for line of business applications. Also provides a flexible sharing model that can rely on security groups and more. The good news is that if you don't have one we can create one for you if you use your work or school account in only a few seconds and for free.

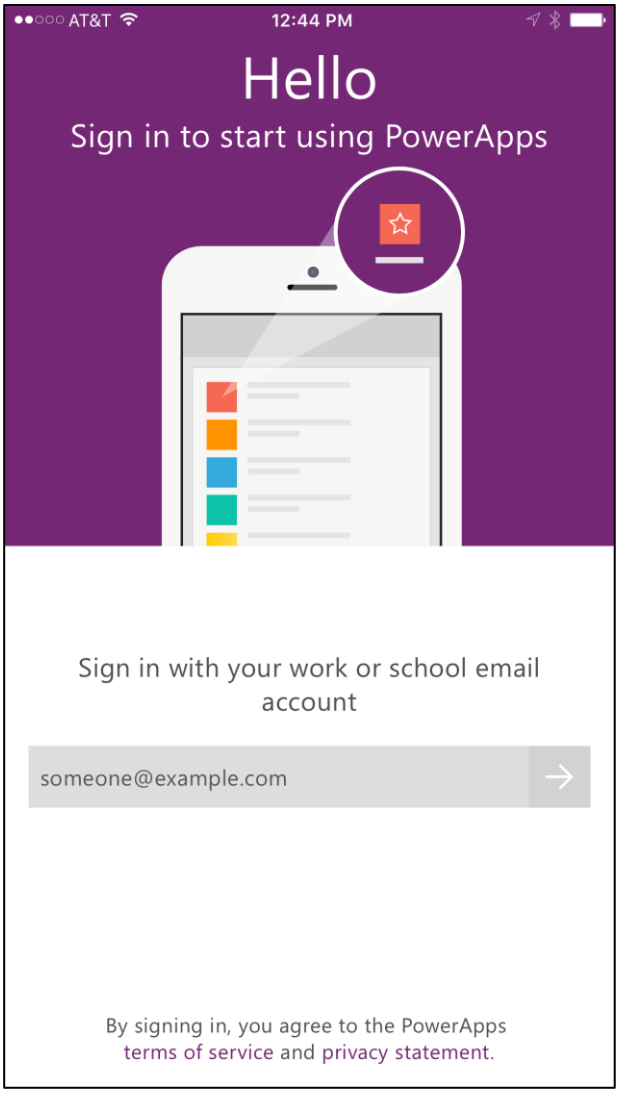

Figure 9. Azure Active Directory support in PowerApps

Once authenticated to the service, PowerApps will display a list of the applications you have access. You can use the different filters offered such as:

- Favorite – Displays the applications that you have explicitly marked as favorite
- My Apps – Apps that you have created or you have used
- All Apps – Displays all the applications that anyone on your organization has shared with you
- Featured – Displays applications that have been tagged by your organization to be featured

- Samples – Access to a few samples that are provided for you to try

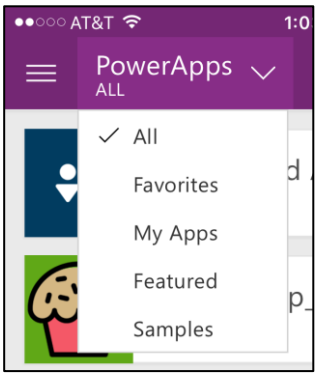

Figure 10. PowerApps application listing filters

To run any application just tap the icon and the application will run.

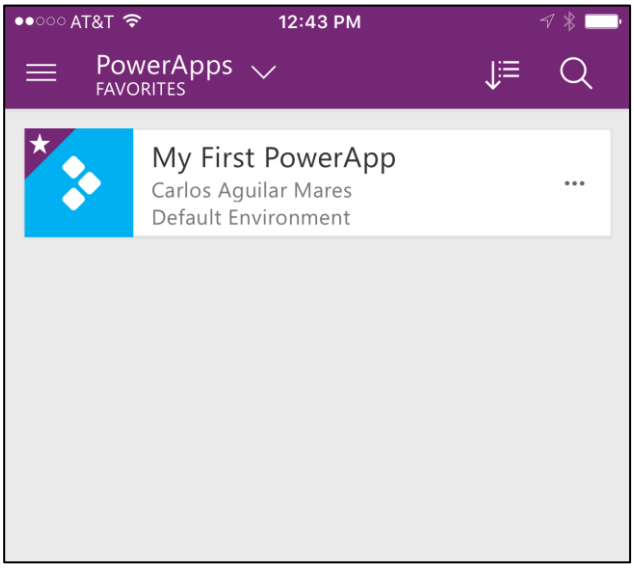

Figure 11. PowerApps application listing

Note: The first time you run an application in your current device PowerApps will create a package for it and download it to your device and show you a "Downloading" message while this happens. This can take several seconds. We'll talk more about this later.

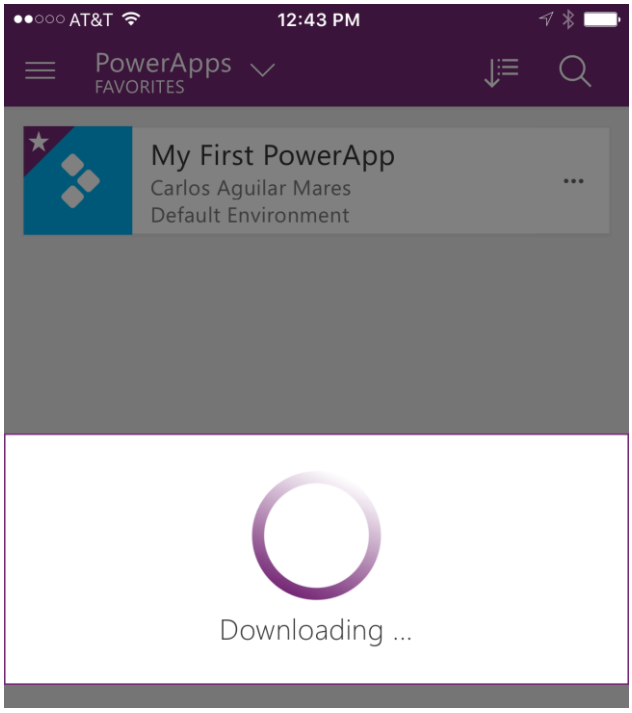

Figure 12. Downloading an application in the PowerApps Player

Once the application is available in the device, it will run it, displaying the Splash screen settings for the application.

Note. You can customize the color and icon of the splash screen of your application in the App Settings page inside PowerApps Studio.

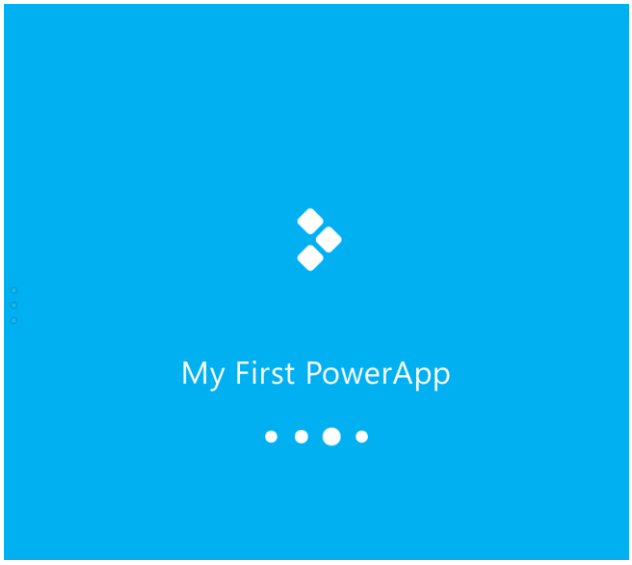

Figure 13. Application Splash Screen

Summary

In this chapter, we briefly explored PowerApps Studio and created a simple application that utilizes the User() formula. You can find all the formulas available and their documentation in the PowerApps Formula Reference at https://powerapps.microsoft.com/tutorials/formula-reference/.

We finally used the PowerApps Player on a mobile device to run the application.

In the next chapters we will look at how to build more interesting applications, including how to access data, navigate, and create Custom APIs that can take the capabilities of PowerApps anywhere you need them to be.

Appendix B: Creating a Calendar App using PowerApps

What we will be building

For this simple example, we are going to be building an application that allows you to display a 7-day-calendar-like navigation for users to select a day. As you select the dates, it highlights the date by using a different color for the font and draws a circle around the selected date.

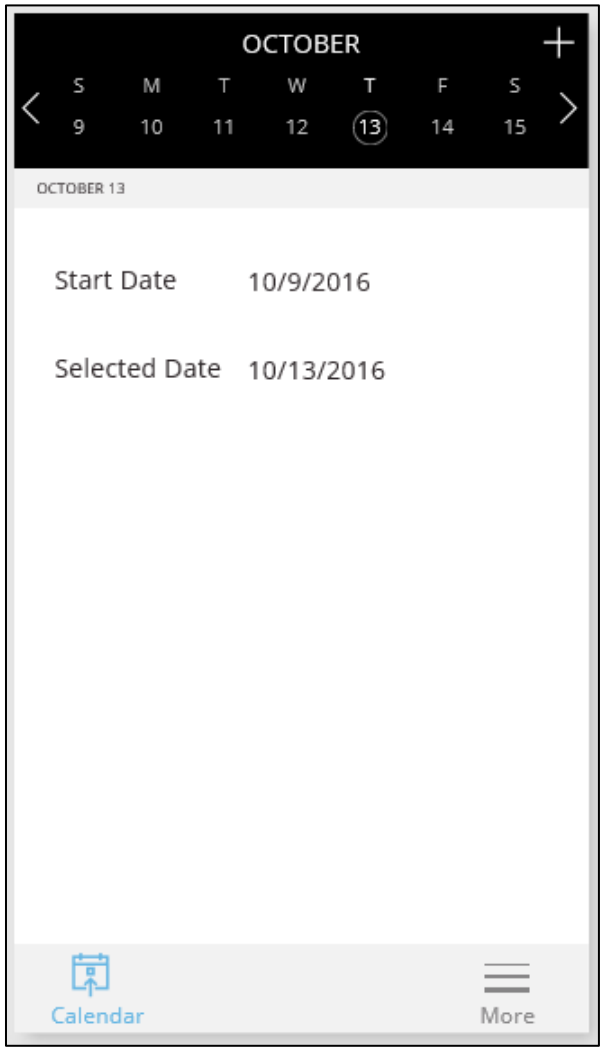

Figure 14. Calendar application

This is not meant to be a complete application but this navigation can easily be extended to build a richer application that displays appointments, tasks or other items, by adding a Vertical Gallery and setting its Items to include a Filter by the selected date

and leverage connections to services like Common Data Model, Office 365 or Google Calendar very easily.

Storing the date

To build this application we will use two context variables (using UpdateContext) in the main screen:

1) **SelectedDate**: which will be the one indicating the date that the user selected.
2) **StartDate**: which we will use to determine the first day of the week to be displayed on the Date selector.

Initializing variables

The best place to initialize both of these variables is on the Screen::OnVisible event of the first screen which will be triggered when the user launches the application and navigates to that screen. One thing to keep in mind is that navigating back from other screens will trigger this event as well, so it is a best practice to make sure you only initialize the state when it has not been set to avoid resetting the data that the user might have set. We'll look on how to do this later.

Start a new Blank - Phone Layout - Application.

Launch PowerApps.

Select the option New→Blank app→Phone layout.

Select the first screen and switch to the Content tab in the ribbon. Click the Screen1 label name and rename it to MainScreen.

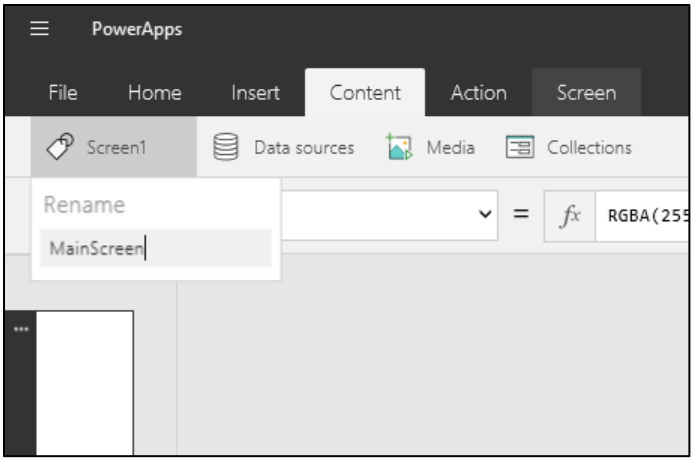

Figure 15. Renaming a screen or control

Now, we are going to add the code to initialize the variables, for this select the OnVisible on the properties dropdown and set the following rule:

```
UpdateContext({SelectedDate: Today(), StartDate: Today() - Weekday(Today()) + 1})
```

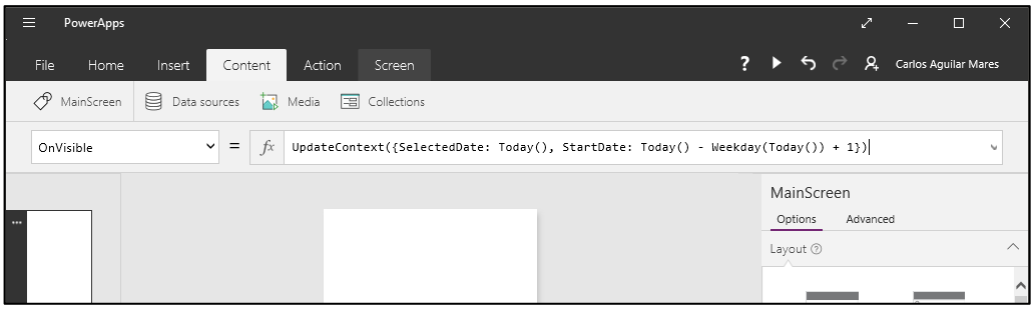

Figure 16. Screen.OnVisible event

This rule is initializing the SelectedDate context variable to Today(), and updating the StartDate context variable to be the Sunday for the current week by subtracting the day of the week.

> **Tip:** You can add two Text boxes and set their Text property to SelectedDate and StartDate respectively to allow you to easily inspect their content. This is

a common practice while you are building applications to quickly inspect easily values as they mutate.

Important Note: You might need to add a new screen, select it in the screen selector and come back to the main screen to ensure that the OnVisible event of the screen is triggered and the variable is initialized properly. This is a current limitation of PowerApps that will be addressed in the future.

Building the User Interface

Now that we have the two variables initialized and available to us, it is time to build the user interface that will allow us to read the values as well as change the date as needed.

1) Adding the Background for the date selector

We will start adding the rectangle that gives the black background for the date selector. So, add a Rectangle by clicking the Insert→Icon→Rectangle which will be the background for the top section of our calendar app where we will display the date selector. Set the following properties:

```
Name: BackgroundRectangle
Fill: Black
X: 0, Y:0, Width: 640, Height: 174
```

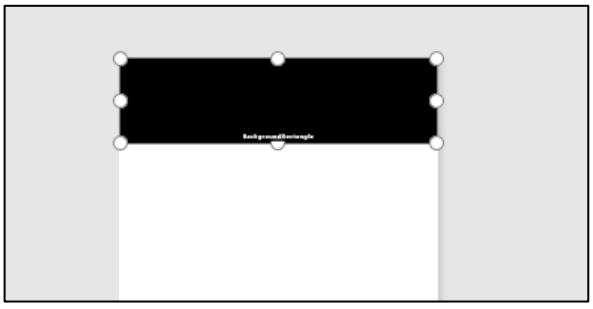

Figure 17. Background for the date selector

2) Adding a Label to Display the Month

This will be the label that displays the name of the month of the currently visible dates so users can get the right context.

```
Name: SelectedMonthLabel
Align: Center
Color: White
Size: 21
X: 0, Y:0, Width: 640
Text: Upper(Text(StartDate, "mmmm"))
```

The Text rule above uses the formatting formula “Text” to display the full name of the month (i.e. October), and then uses the “Upper” formula to convert it to upper case. (i.e. OCTOBER)

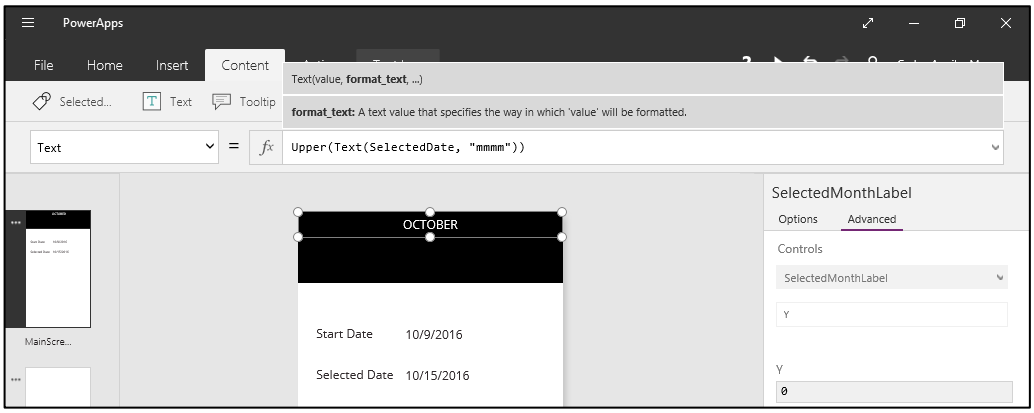

Figure 18 - Formatting the month

> **Note**: If you don’t see the Label displaying the Month, make sure to add a new Screen, navigate to it, and navigate back. This will ensure that the OnVisible event is triggered. This will not be a problem for your users, and is only a current limitation of the authoring environment.

3) Adding the date selector.

To allow users to select different days, we want to display a full week (7 days) starting Sunday so they can easily click one of them. For this we will use a Horizontal Gallery, and we will bind it to an array of 7 items (0-6) that we will use to add one day for each item to the StartDate so that we can easily calculate the date that each item will represent.

a. **Add a new Horizontal Gallery.**

Click Insert→Gallery→Horizontal (text gallery)

By default it will include three text box controls, remove all of them from the default template that is included. You can use the Advanced view to facilitate the selection of those controls.

b. **Display the day of the number**

For this we will use a textbox to display the day (number). So, add a **textbox** to the Gallery by clicking Insert→Text. Make sure that the gallery is selected so that the textbox is added inside the template. Set the following properties:

```
Name: DateNumberLabel
X: 0, Y: 44, Width: 76, Height: 40
Align: Center
Size: 16
Color: If(ThisItem.IsSelected,  RGBA(255,255,255,1), RGBA(200,
200, 200, 1))
Text: Day(StartDate + Value)
```

The Color property above uses the "If" statement to check if the current date is the selected one in the gallery, and if so it will use a different color to differentiate the users selection. In a gallery you can use the **ThisItem** to get a reference to the current item and you can use the **IsSelected** property to determine if it is the currently selected one.

The Text property is being set by using the "Day" formula that returns the day of the month for the specified date.

c. Add the highlight for the currently selected date

For this, we will use a Circle to surround the day of the month that is currently selected. Add a Circle by using the Insert→Icon→Circle, and set the following properties:

```
Name: DateCircle
X: 20, Y: 46, Width: 38, Height: 38
BorderColor: White
BorderStyle: Solid
Fill: RGBA(0,0,0,0)
Visible: If(SelectedDate = DateAdd(StartDate, ThisItem.Value),
true, false)
```

The Visible property is being set to the true if the selected date is the same as the date that they current item represents.

d. Display the First letter of the month

We want to also help the user by displaying the first letter of the month so they can easily know if it is Wednesday or Thursday without the need to count days, so to do that add another textbox where we will display the day of the Week. Note that this will also be used as the “hit point” for our selection, so it will take over the entire template of the gallery to capture all clicks.

```
Name: DateFirstLetter
X: 0, Y: 0, Width: 76, Height:106
Size: 16
Align: Center
VerticalAlign: Top
Text: Left(Text(StartDate + Value, "ddd"), 1)
Color: If(ThisItem.IsSelected,  RGBA(255,255,255,1), RGBA(200,
200, 200, 1))
```

```
OnSelect: UpdateContext({SelectedDate: DateAdd(StartDate,
ThisItem.Value, Days)})
```

Here the Text property is using the "Left" formula to get only the first character of the three letter day (Mon, Tue, Wed, etc).

We are also establishing the "OnSelect" event to Update the SelectedDate with the value that this item represents.

e. Finally, lets format the gallery/date selector itself

For that select the gallery in the Canvas and set the following properties to it.

```
Name: DateGallery
X: 29, Y: 58, Width: 580, Height: 106
TemplateSize: 76
Items: [0, 1, 2, 3, 4, 5, 6]
```

By now you should be able to select any dates displayed and the application will react to the selected date and reflect the value on any control that depends on it.

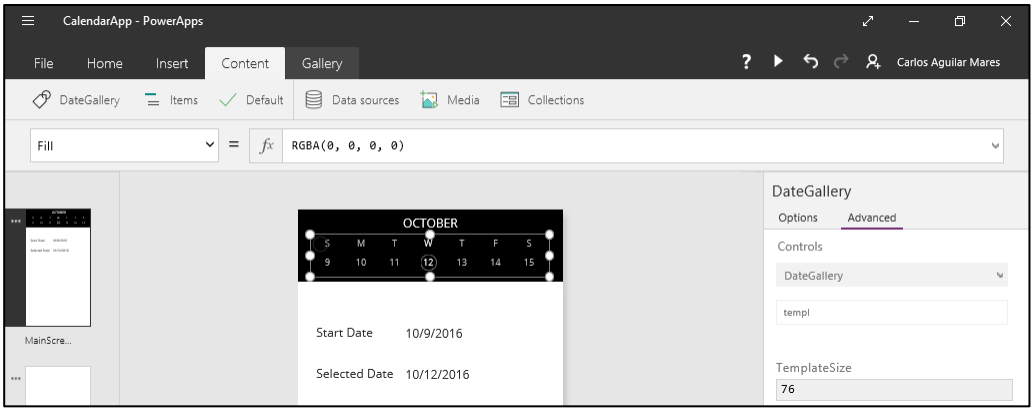

Figure 19. Days gallery complete

4) Adding the "Previous Week" navigation.

For this we will use an Icon, so click Insert→Icon→ Chevron Left, and set the following properties:

```
Name: PrevWeekArrow
X: 0, Y: 70, Width: 36, Height: 72
Color: White
OnSelect: UpdateContext({StartDate: DateAdd(StartDate, -7,
Days)})
```

The OnSelect event is specifying to update the StartDate variable by decreasing one week from the current StartDate.

5) Adding the "Next Week" navigation.

For this add a Chevron Right with the following properties:

```
Name: NextWeekArrow
X: 600, Y: 70, Width: 36, Height: 72
Color: White
OnSelect: UpdateContext({StartDate: DateAdd(StartDate,  7)})
```

Similarly, here the OnSelect event is specifying to update the StartDate variable by adding one week from the current StartDate.

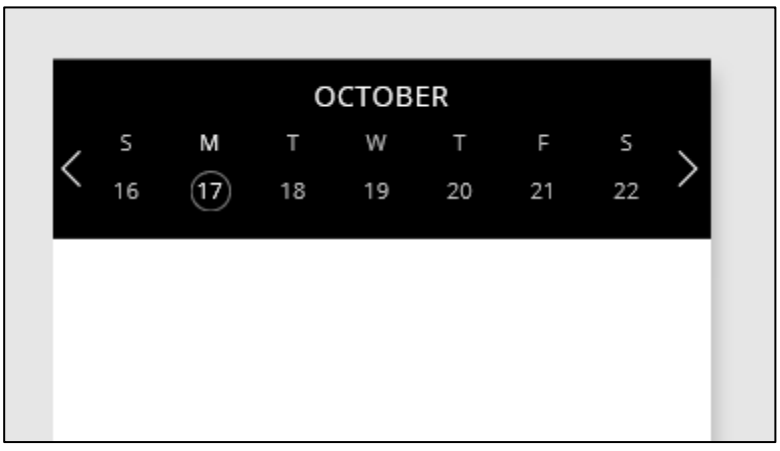

Figure 20. Previous and Next week navigation included

6) Display the selected date

We want to make sure that it is obvious to the user which date is selected so for that add a text box with the following properties:

```
Name: SelectedDateTextBox
Size: 12
X: 0, Y: 174, Width: 640, Height: 43
Fill: RGBA(242, 242, 242, 1)
PaddingLeft: 25
Text: Upper(Text(SelectedDate, "mmmm dd"))
```

At this point our date selector is completed and you can now navigate using the arrow icons, and select any date.

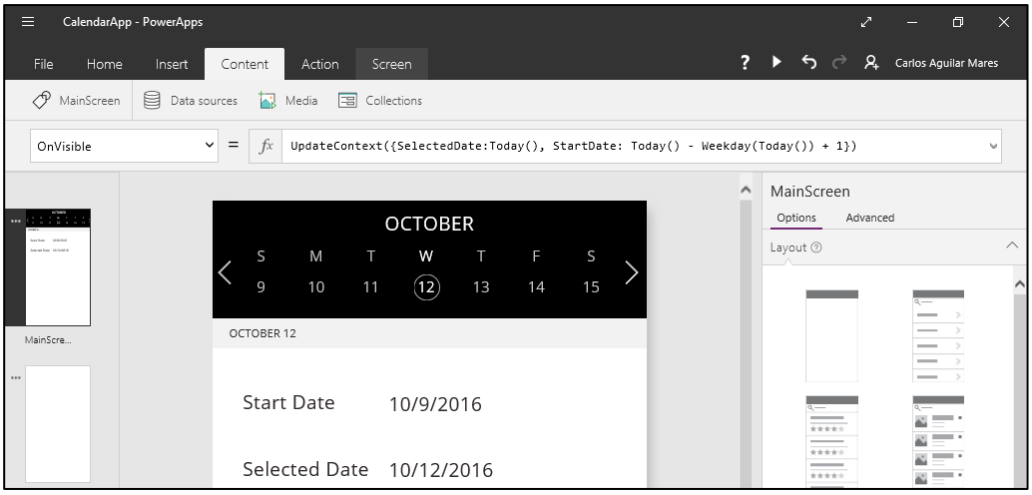

Figure 21. Date selector completed

7) Adding a create option

To complete the app, we can use a "Create" icon by clicking Insert→Icon→Add with the following properties:

```
Name: AddIcon
Color: White
X: 586, Y: 12, Width: 42, Height: 42
OnSelect: Navigate(CreateTask, ScreenTransition.Fade,
{SelectedDate: SelectedDate})
```

Here the OnSelect event is navigating to the CreateTask screen and it is passing the local (to MainScreen) variable SelectedDate to it, so that we can reference the screen easily from there.

Add a new Screen and call it “CreateTask” to allow the “+” navigation to work.

Final thoughts

You can easily add a navigation bar, and use formulas like Navigate and Back, to complete the application.

To make it part of a real application you can use now an expression such as “Filter('Sales Order', OrderDate = SelectedDate)” and bind that to a Gallery where you display all the sales orders coming from your data.

I mentioned briefly that it in our application it is important to only update the context on the OnVisible event when navigating to it for the first time so that when the user navigates to another screen and goes back we do not reset the selection that they previously had. In this case we can use an If statement to check for IsBlank to only trigger the UpdateContext in that case, for example:

```
If (IsBlank(SelectedDate),  UpdateContext({SelectedDate:Today(),
StartDate: Today() - Weekday(Today()) + 1}))
```