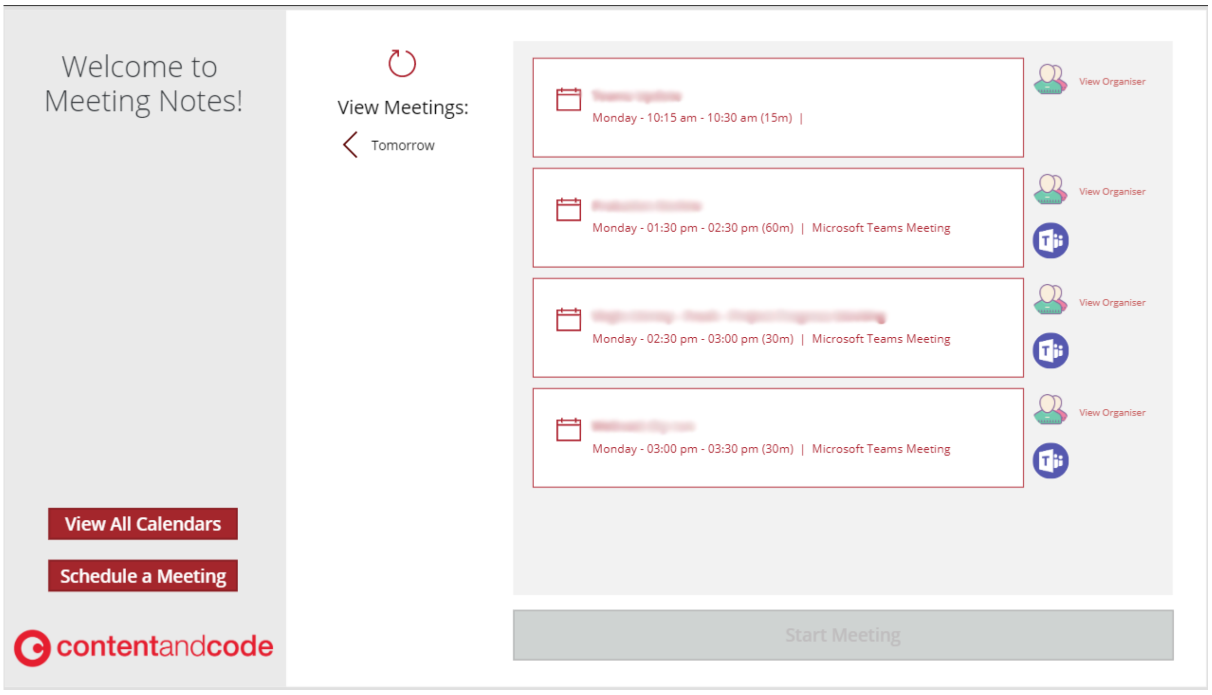
Meeting Capture V2.1 – Adding functionality to your Microsoft Teams meetings with the use of the Power Platform!

Initially Out-of-the-Box (OOTB), Meeting Capture is developed by Microsoft, expanded by [MVP Reza Dorrani]() and further tweaked by me to adapt and adjust to our ways of working.

In the guide below, I break down some of the features Reza implemented in Meeting Capture, the process I use to tailor the app (including codes used) and a few useful lessons I've learned when developing on top!

The App:

Meeting Capture is an all-in-one tool for capturing information during meetings as they happen. Users can view meeting details, take notes and pictures of whiteboards, sketch, assign tasks, and easily schedule follow up sessions.



[redacted]
Monday - 10:15 am - 10:30 am (15m) |

[redacted]
Monday - 01:30 pm - 02:30 pm (60m) | Microsoft Teams Meeting

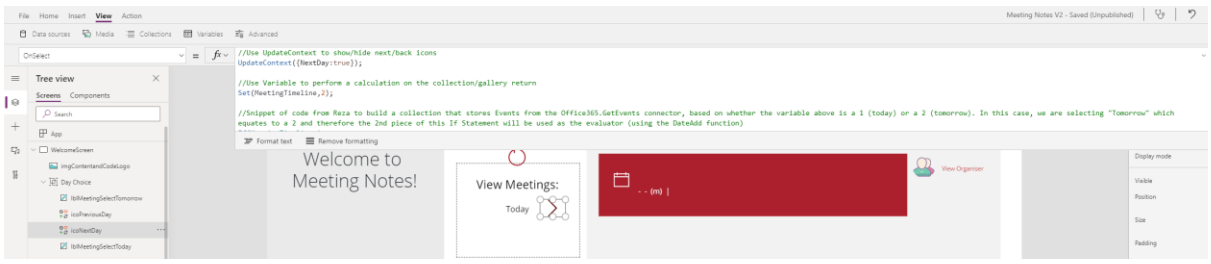
[redacted]
Monday - 02:30 pm - 03:00 pm (30m) | Microsoft Teams Meeting

[redacted]
Monday - 03:00 pm - 03:30 pm (30m) | Microsoft Teams Meeting

Choosing “Today” or “Tomorrow” meetings:

Moving the OOTB Meeting Capture app forward, and tweaking Reza’s V2, you can see from the front screen that a few extra features have been added. Firstly, and quite impressively, the ability to view meetings with a “Today” or “Tomorrow” scope – all at the click of a button!

I achieved this by tweaking Reza’s initial drop-down choice field, and inserting some OnSelect code to an icon:



```
//Use UpdateContext to show/hide next/back icons
UpdateContext({NextDay:true});

//Use Variable to perform a calculation on the collection/gallery return
Set(MeetingTimeline,2);

//Snippet of code from Reza to build a collection that stores Events from the Office365.GetEvents connector, based on whether the variable above is a 1 (today) or a 2 (tomorrow). In this case, we are selecting "Tomorrow" which equates to a 2 and therefore the 2nd piece of this If Statement will be used as the evaluator (using the DateAdd function)
```

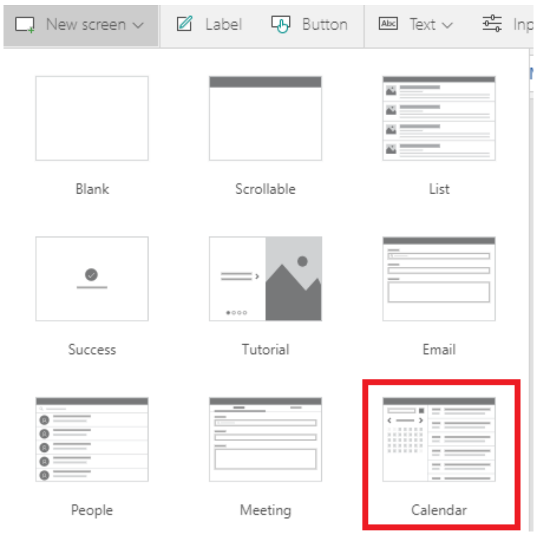
```
//Here I use UpdateContext to show/hide next/back icons
UpdateContext({NextDay:true});

//And based on this being a set icon/button, here I set a variable to perform a
calculation on the collection/gallery return (setting MeetingTimeline to 2
[Tomorrow])
Set(MeetingTimeline,2);

//Snippet of code from Reza to build a collection that stores Events from the
Office365.GetEvents connector, based on whether the variable above is a 1 (today)
or a 2 (tomorrow). In this case, we are selecting "Tomorrow" which equates to a 2
and therefore the 2nd piece of this If Statement will be used as the evaluator
(using the DateAdd function)
If(MeetingTimeline=1,
ClearCollect(
    AllFutureEvents,
    Office365.GetEventsCalendarView(
        MyCalendarID,
        Text(Today(), UTC),
        Text(DateAdd(Today(), MeetingTimeline, Days), UTC)).Values
        ),
ClearCollect(
    AllFutureEvents,
    Office365.GetEventsCalendarView(
        MyCalendarID,
        Text(DateAdd(Today(), 1, Days), UTC),
        Text(DateAdd(Today(), MeetingTimeline, Days), UTC)).Values
    )
);
```

```
//Using the results of the above collection, Reza now builds his final collection
that is used on the gallery. He first adds a column to his new collection that he
then performs a calculation on whether the meeting is current or not - which he
later uses within the gallery to display a visual indicator of "Current Meeting".
The filter he is applying here from above is based on how long the meetings last,
filtered to reduce the potential of a holiday or other event being pulled in to the
app
ClearCollect(
    MeetingsOnly,
    Filter(
        AddColumns(
            AllFutureEvents,
            "isCurrent",
            DateDiff(
                Start,
                Now(),
                Seconds
            ) > 0 && DateDiff(
                Now(),
                End,
                Seconds
            ) > 0
        ),
        DateDiff(
            Start,
            End,
            Hours
        ) < 6
    )
);

//Using a CountRows formula, Reza then sets a variable to allow him to interrogate
and automate the process (using the column he inserted in his above ClearCollect),
followed in IF statement formula below
Set(NumberOfCurrentMeetings, CountRows(Filter(MeetingsOnly, isCurrent)));

//If the user has a current meeting, and it is the only one, Reza then uses a
variable to automatically select that meeting (automating the process slightly,
presuming the user wants to write notes relating to a meeting that is currently
active
If(
    NumberOfCurrentMeetings = 1, Set(AutoSelectMeeting, false);
Set(SelectedMeeting, LookUp(MeetingsOnly, isCurrent))
)
```

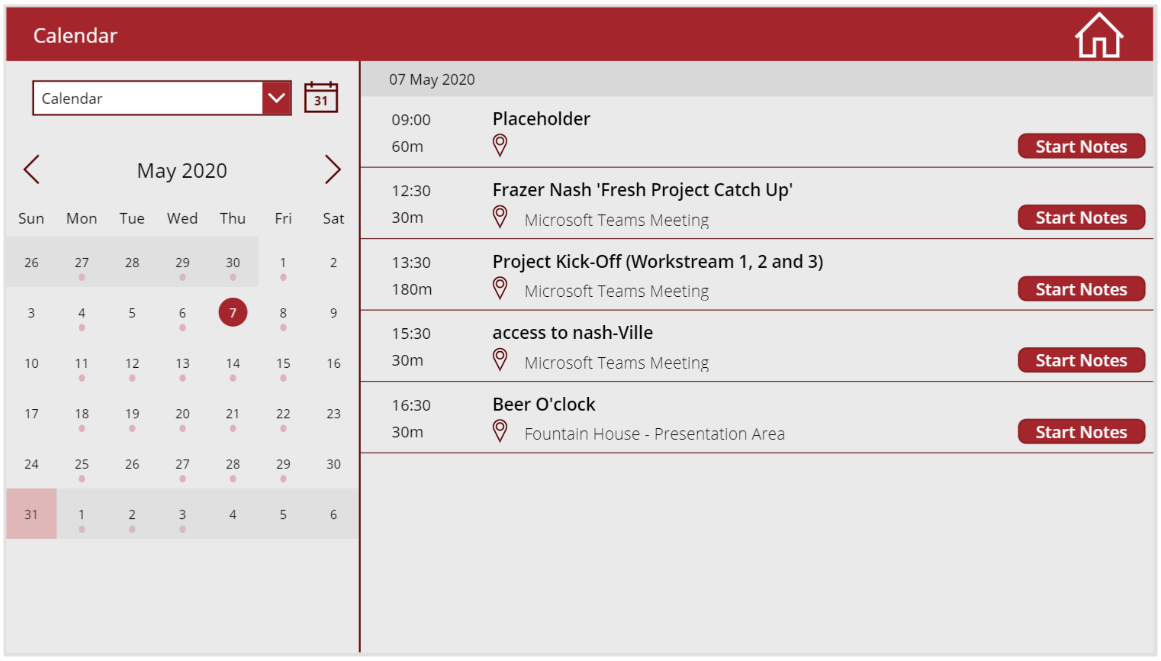
This icon then has its Visible property set to !NextDay which will automatically hide it once pressed (to stop users from being able to continue moving through the days) – drawing your attention here to the use of the NOT operator, as we're setting the Context variable to true OnSelect.

Using a Calendar View to choose any meeting:

First things first, we need to create a new screen. Using the built-in template, we can create a Calendar screen:



Using this built-in screen, we are provided with all the complex functionality completely OOTB. The only feature we then need to add is a small amendment within the gallery view which will allow us to “Select” a meeting:



09:00 60m Placeholder

12:30 30m Frazer Nash 'Fresh Project Catch Up' Microsoft Teams Meeting

13:30 180m Project Kick-Off (Workstream 1, 2 and 3) Microsoft Teams Meeting

15:30 30m access to nash-Ville Microsoft Teams Meeting

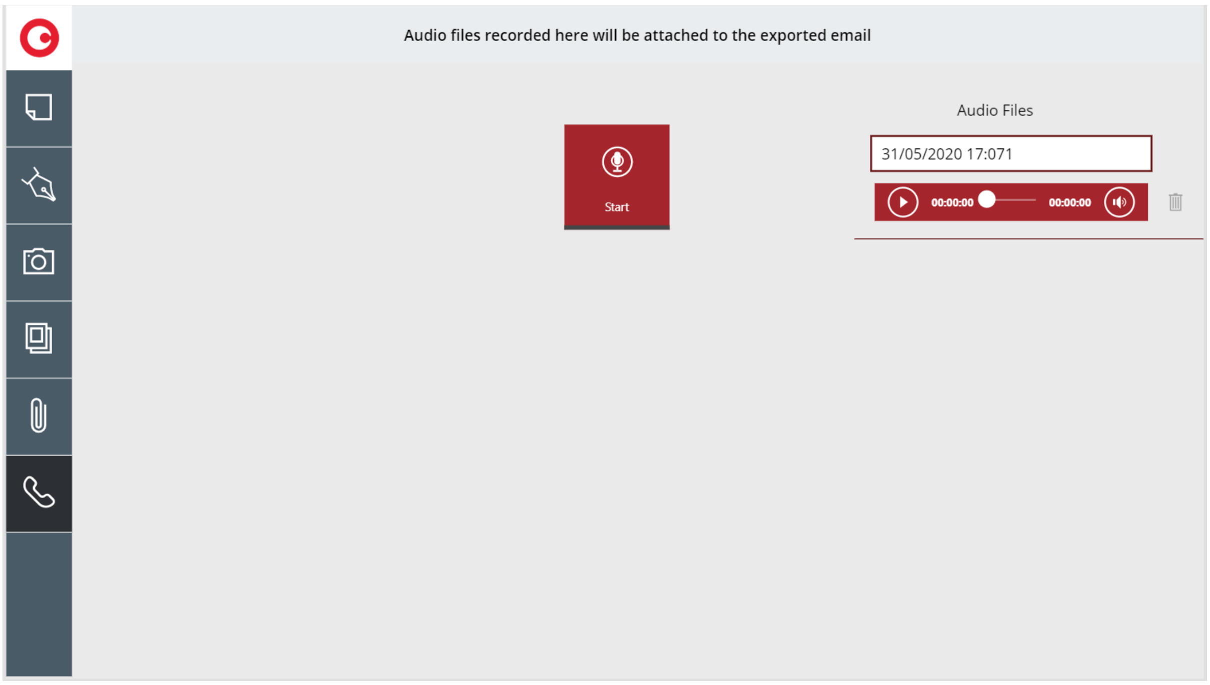
16:30 30m Beer O'clock Fountain House - Presentation Area

Placing a button into the gallery, Reza inserts the following code to allow this to provide an “Any meeting” picker:

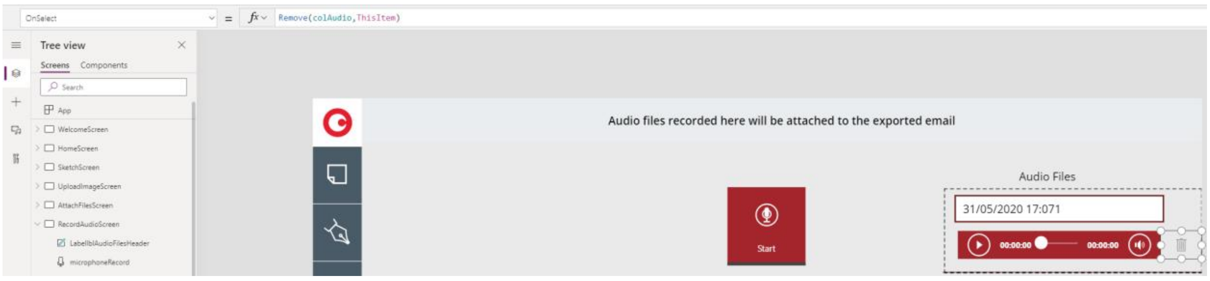
```
//Set the variable that depicts which meeting notes you'll be taking
Set(SelectedMeeting, ThisItem);

//Navigate to the meeting notes screen
Navigate(HomeScreen, None);

//Set the variable that will show the pop-up presented to user upon loading
home screen
Set(ShowDataLossWarning, true);

//Clears a feature not yet finished (task tracking with time/cost
estimates)
Clear(Calculations);

//Builds the new attendee list that's relevant for the meeting selected
from this calendar
ClearCollect(CalendarItemInfo, Office365.CalendarGetItem(MyCalendarID,
galCalendarEvents.Selected.Id).Attendees)
```

Extra functions (Attachments, Audio, etc.):

As with most meetings, having text notes on its own is not ideal. Because of this, Reza built in the ability to capture audio and attachments alongside sketches, images, and photo uploads. Each one of these functionality pieces operate on its screen, connected using a menu collection that can be created manually if you prefer it (which I do personally)!

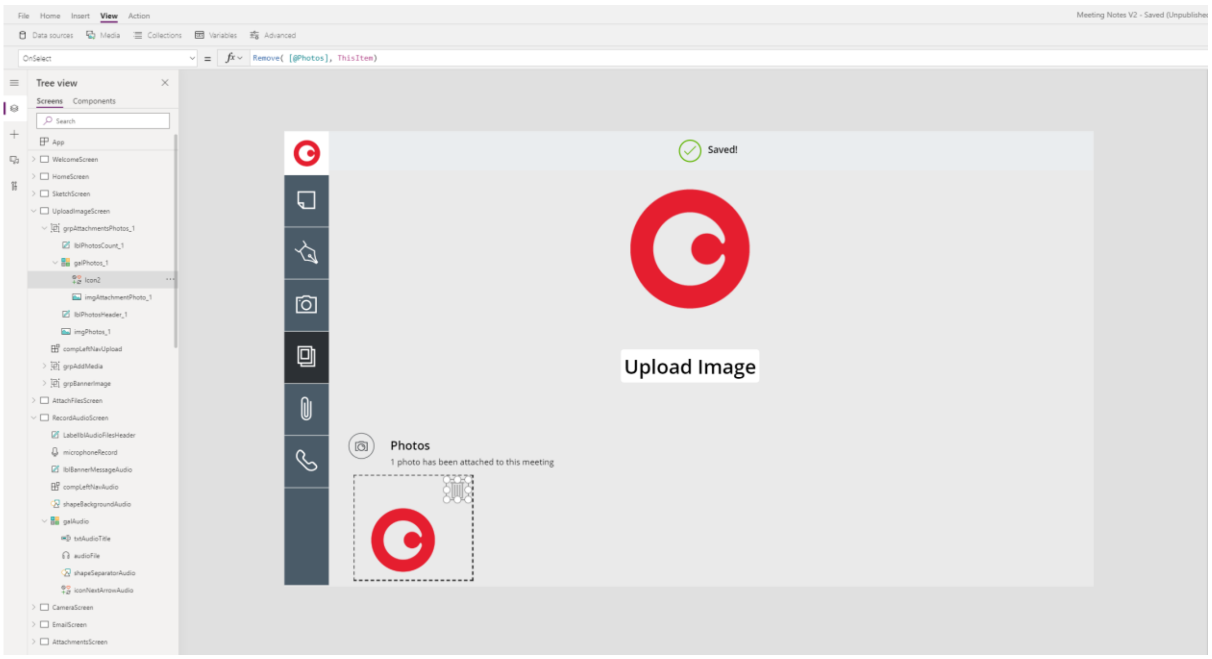

Audio files recorded here will be attached to the exported email
Audio Files
31/05/2020 17:071
Start
00:00:00
00:00:00

A useful feature on each of these screens, which I continued to add for all of them, is to then display the outputs in a gallery – mainly because I know how common it is to attach, upload and record something and need to delete it because it's wrong! Adding a gallery and linking it to the output allows me to place a small trash icon, with a basic OnSelect using the Remove function:



OnSelect = Remove(colAudio,ThisItem)

I then ensured all of these galleries were added to all the added functionality pieces, for example on the image upload:

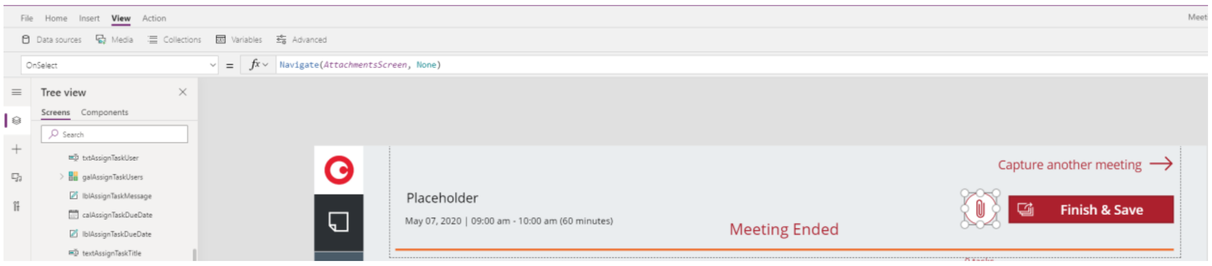


OnSelect

```
Remove( [@Photos], ThisItem)
```

Full visibility of all added attachments:

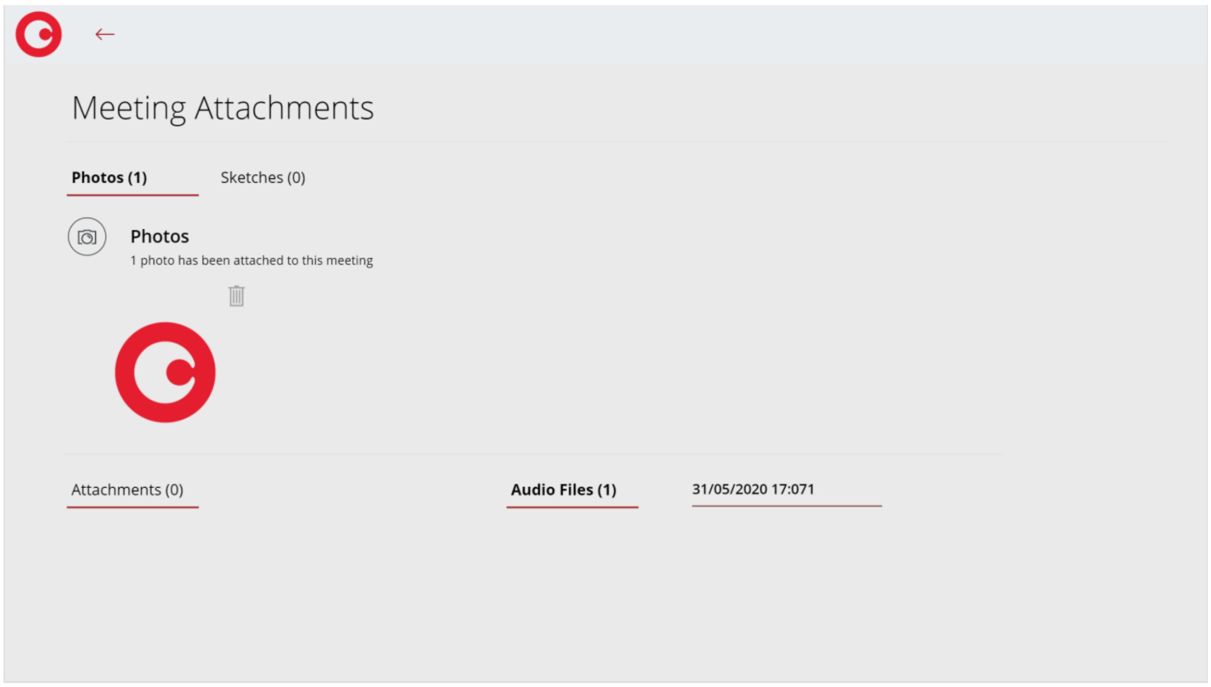
Based on the idea that each meeting might end up being extremely large (in respect to added audio, attachments, pictures and sketches), it made sense to ensure there was a single place to manage these attachments. Using the attachment icon on the OOTB meeting capture screen, we move to the Attachments Screen:



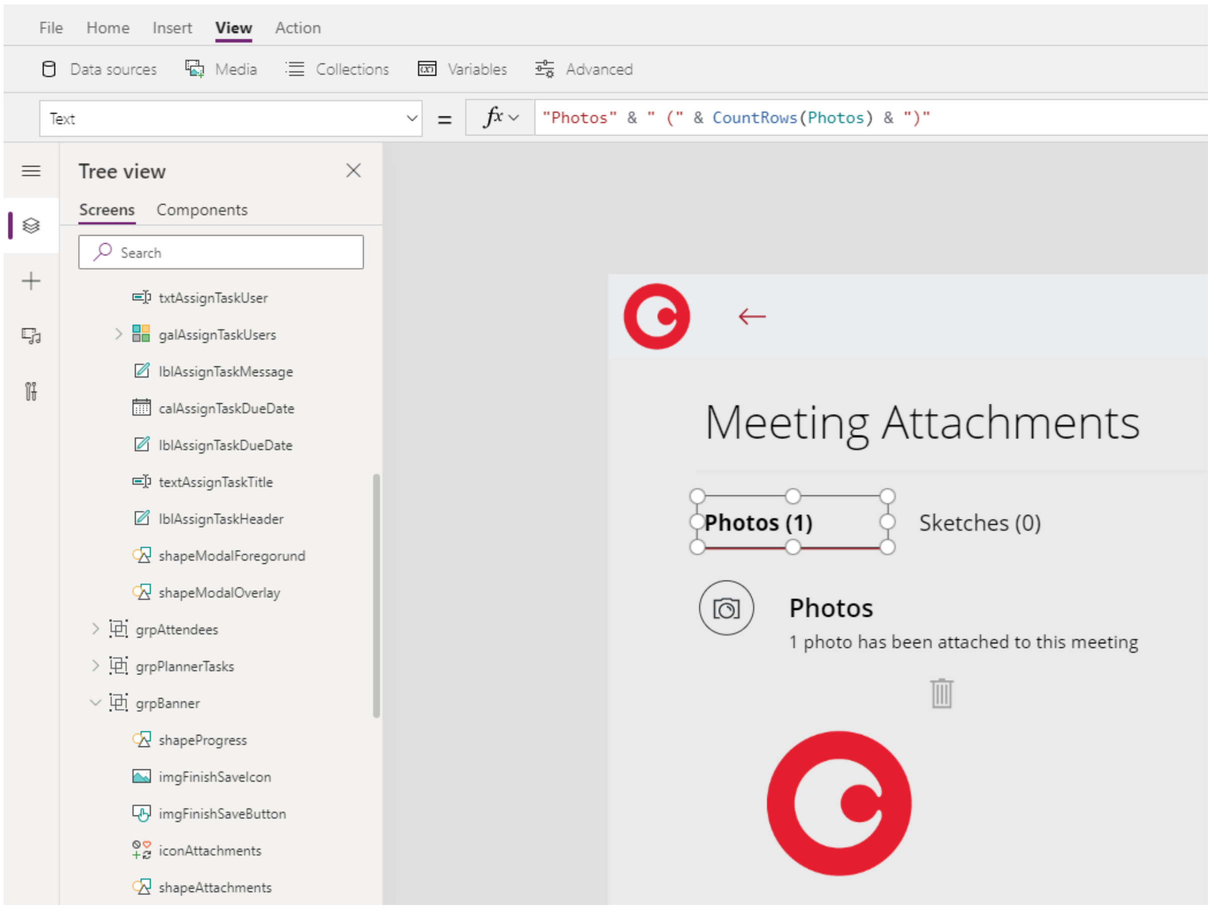
OnSelect

```
Navigate(AttachmentsScreen, None)
```

In here, Reza had added the required galleries, however, I felt a different visual display would benefit the users more and so with a combination of some FontWeight snippets and some CountRow functions, I created the following attachment screen:

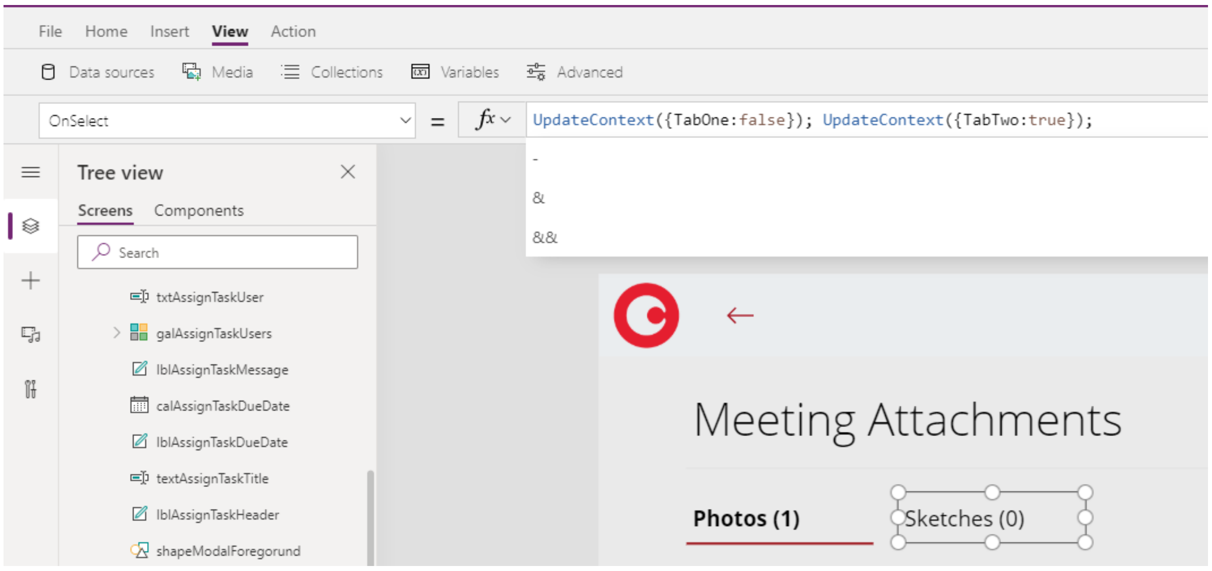


Each piece of this runs on a similar principle, for example:



```
"Photos" & " (" & CountRows(Photos) & ")"
```

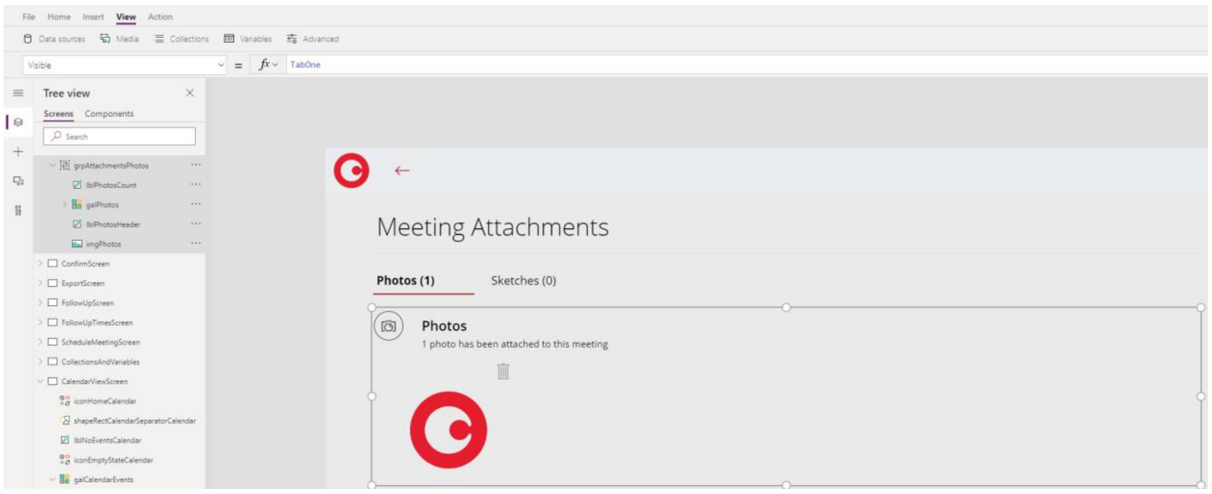
What you can see here is that I am using a combination of plain text “Photos” and a CountRows function on the Photos collection to display “Photos (1)”. Moving this a step forward, I have then placed the Photos and Sketches in a tab format, using the UpdateContext variable to show or hide the elements for each option:



OnSelect =

```
UpdateContext({TabOne:false}); UpdateContext({TabTwo:true});
```

The way this is broken down will allow my users to select either “Photos” or “Sketches”, and the result is that each of the relevant elements is using a Visible set to the related Context variable, for example:



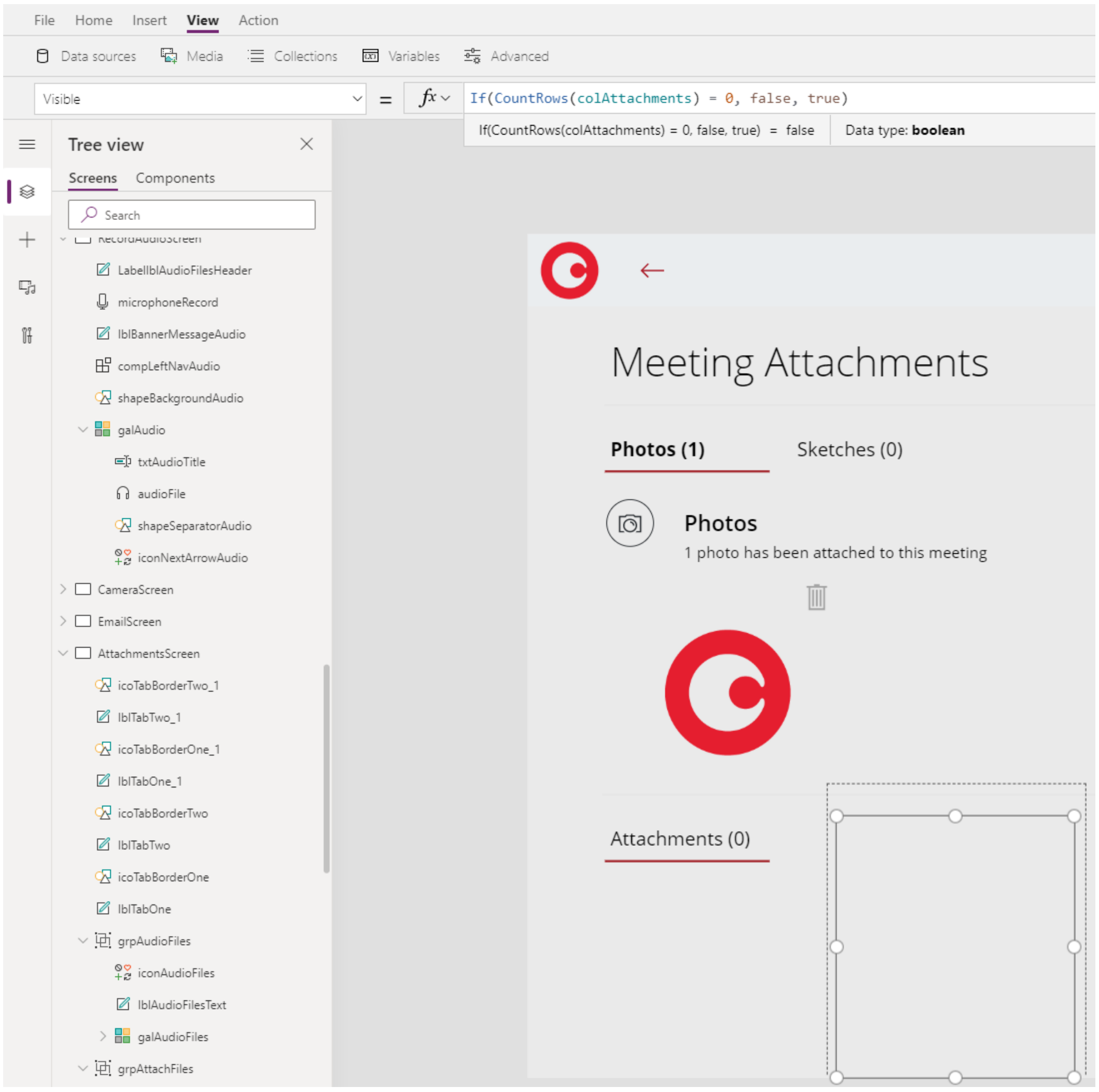
Visible =

```
TabOne
```

Each element (lblPhotosCount, galPhotos, lblPhotoHeader and imgPhotos) has its Visible property set to TabOne, meaning that they will only be visible when TabOne = true, and likewise for TabTwo and each of the sketch elements. This is then operated based on the OnSelect of the labels, shown in the previous image.

What this allows me to do is utilise the space on the screen and offer the user the ability to look and browse the attachment categories they feel is most important. As you can also see, I have ensured there is also a trash icon placed inside the galleries to allow the users to tidy and clear out anything not needed from this screen too.

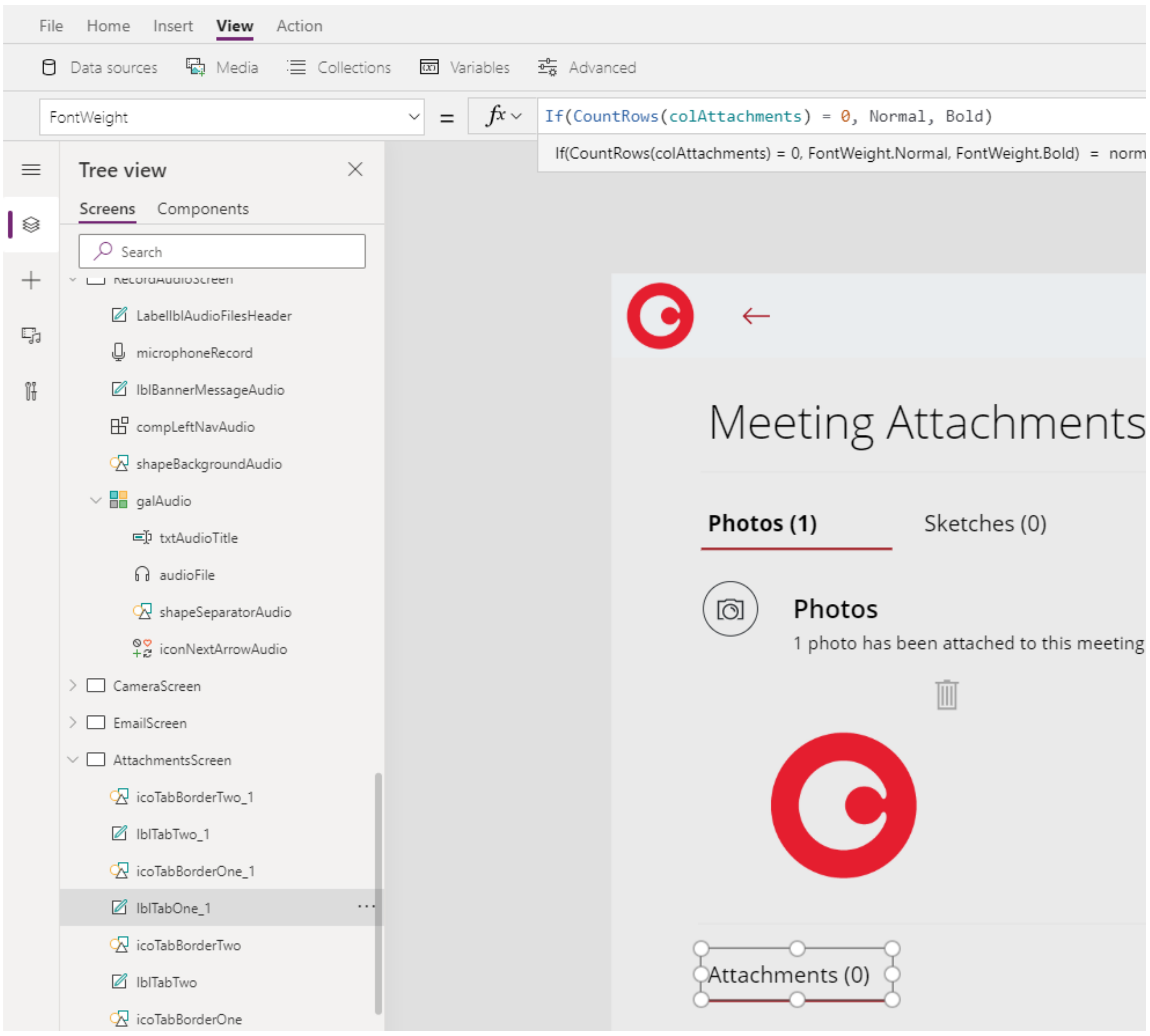
The Attachments and Audio Files tabs are slightly different as the galleries Reza added are marginally smaller. That said, I found they would display better on a permanent display, with a Visible property used to hide the galleries if there is nothing present in their respective collections, for example:



```
If(CountRows(colAttachments) = 0, false, true)
```

If(CountRows(colAttachments) = 0, false, true) = false   Data type: **boolean**

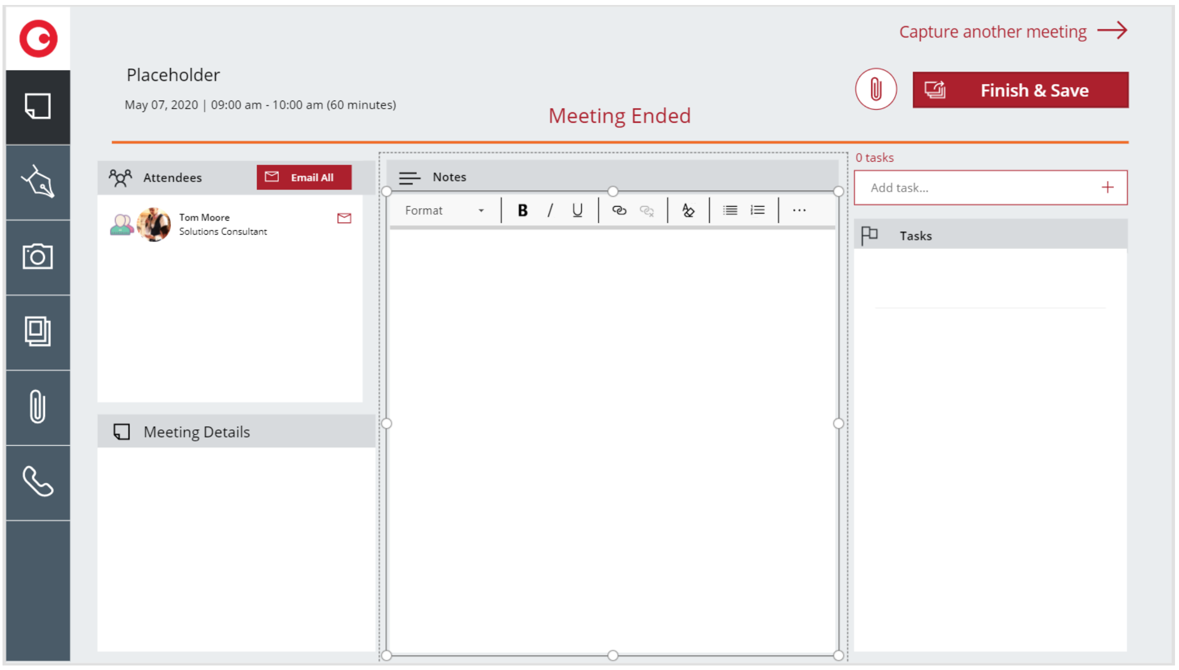
To offer the user slightly easier functionality too, I then amended the FontWeight property of Attachments and Audio Files only to highlight bold if, yes – you guessed it – there are items within the collections too! All based off a simple IF statement format:



```
If(CountRows(colAttachments) = 0, Normal, Bold)
```

Rich Text Notes:

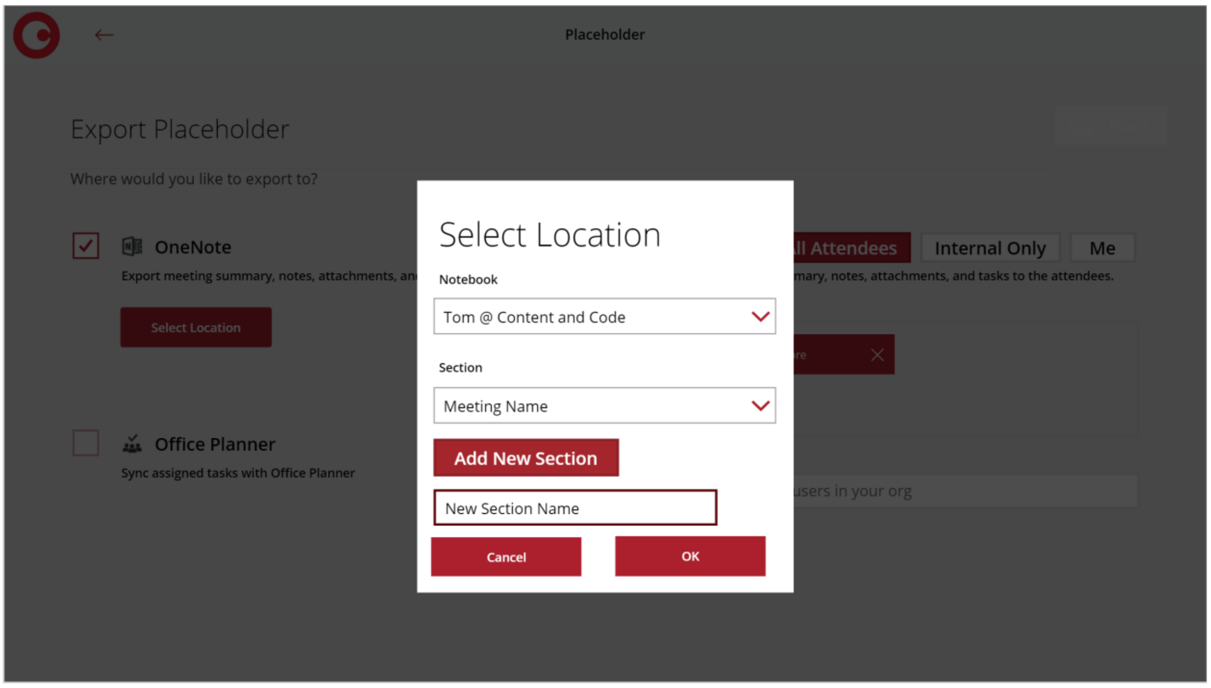
It's also worth noting that while the OOTB Meeting Capture app allows you to capture basic notes, Reza made a swap of the standard text box over to a rich text box to allow you to achieve rich text formatting:



Exporting:

Using the Finish & Save button from the Home Screen, we can move to the most crucial piece of this app – the exporting!

This button moves us through to the Export Screen, where we can now select the places to which we would like to export these captured items. One functionality piece worth noting here that Reza included is the ability to add a new OneNote section directly from within the app itself:



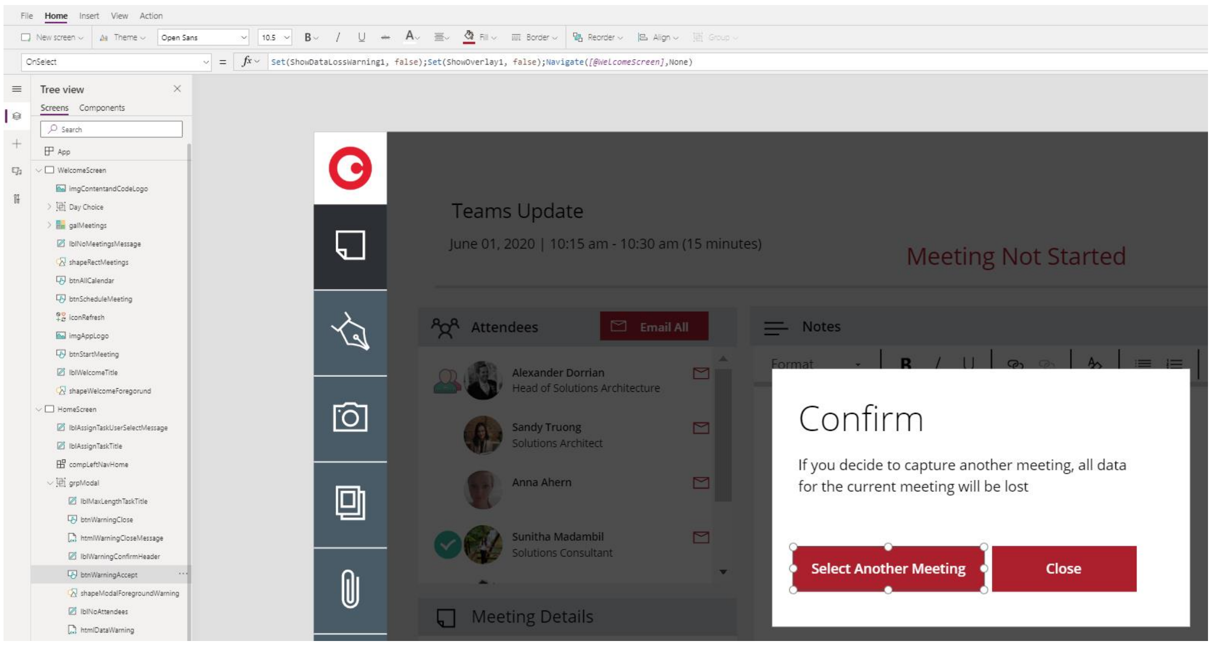
This Add New Section button then has the following code on the OnSelect property:

```
//Using the OneNote connector, we are able to take advantage of the
CreateSectionInNotebook option - to which we are using the dropdown choice
to select the relevant notebook and the freetype text box to specify the
name of the new section
'OneNote(Business)'.CreateSectionInNotebook(drpOneNoteBookSelect.SelectedTex
t.Key,{name:txtNewOneNote.Text});

//Refresh the ClearCollect and Reset the text box, updating the drop down
with the new addition
ClearCollect(OneNoteSections,'OneNote(Business)'.GetSectionsInNotebook(drpOn
eNoteBookSelect.SelectedText.Key).value);Reset(txtNewOneNote);UpdateContext(
{SectionAdded:true})
```

Once you've chosen each of the areas you wish to export, we are then allowed to add extra attendees if you need to FYI someone, or we can continue straight through to the "Export" option.

Other useful features:

Moving this app away from being a quick-use feature for a meeting and positioning it as more of a tool and platform, the ability to both "Capture another meeting" and also "Schedule a meeting" are two features added by Reza that are extremely useful. Focusing on the first, which on the premise is relatively simple, is the use of a navigate function to take us back to the initial start-up screen:


OnSelect

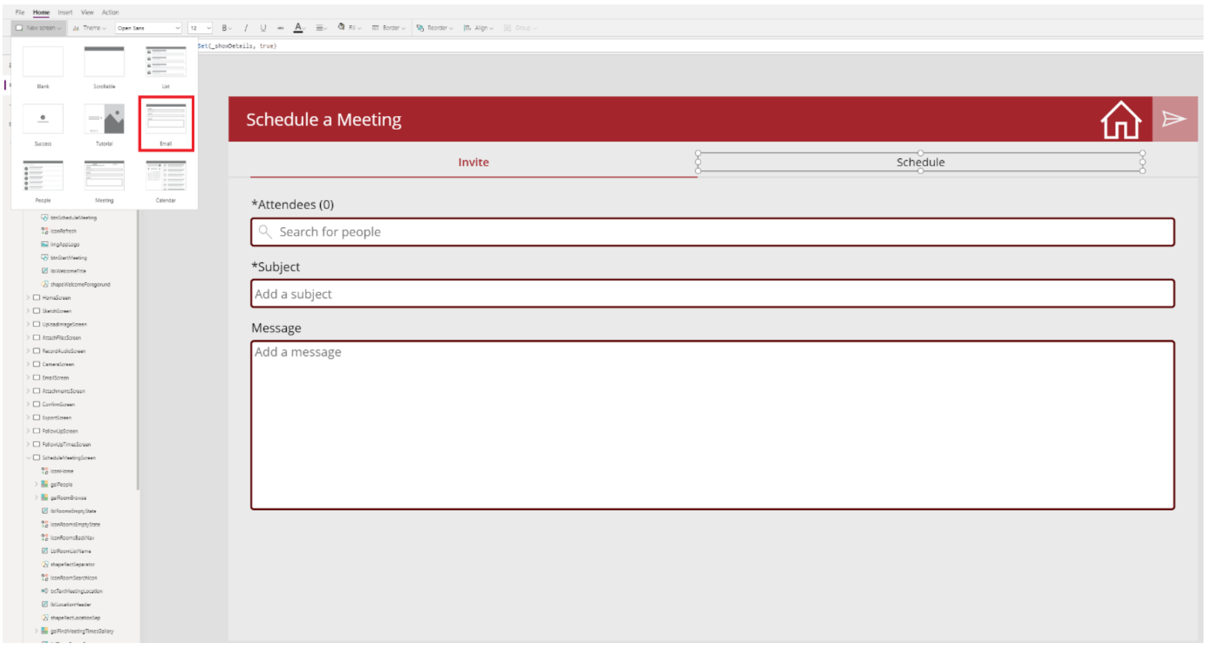
```
Set(ShowDataLossWarning1, false);Set(ShowOverlay1, false);Navigate([@WelcomeScreen],None)
```

Confirm

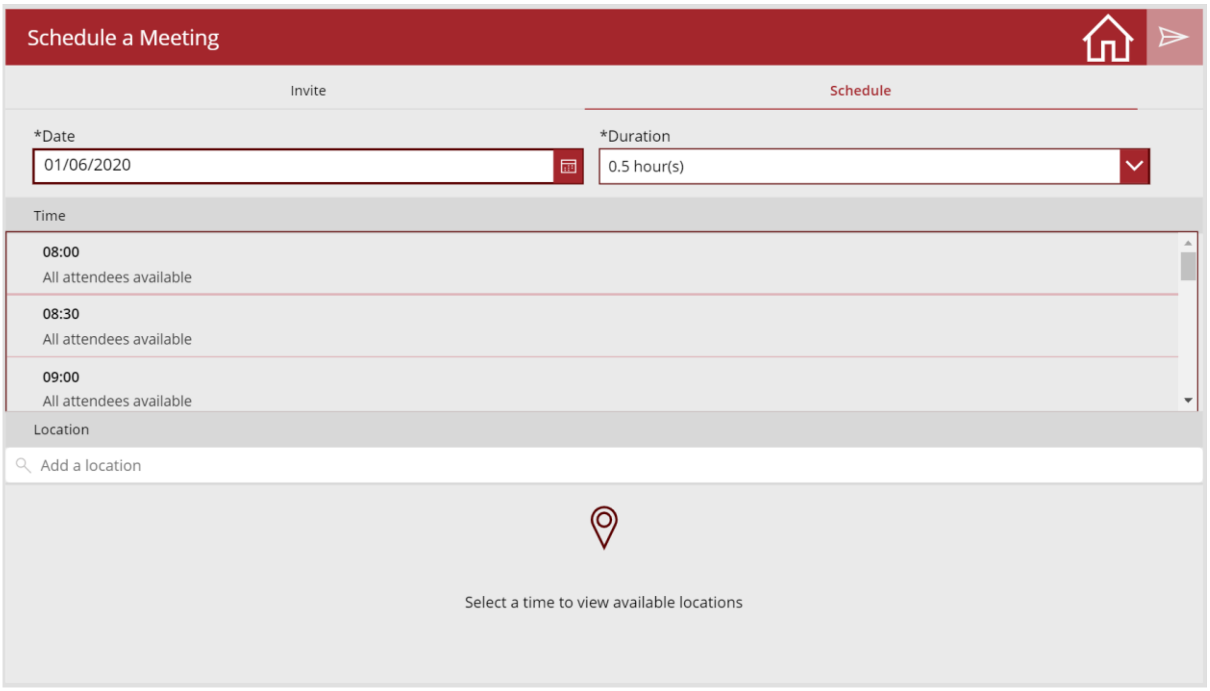
If you decide to capture another meeting, all data for the current meeting will be lost

The Schedule Meeting, however, has some extra complexity to it.

For instance, a button placed on the front screen allows easy navigation to this feature, on which Reza has added an OOTB Email screen:



However, Reza has also taken this one step further and added in the ability to allow meeting schedules to show – aiding the entire process!



The key feature on this schedule tab is, of course, the ability to surface and show when each of the attendees is available at the same time – cleverly done so using the OnSelect of the date picker, with the following code snippet:

```
Concurrent(
Reset(txtSearchBoxScheduledMeeting),
Set(_showMeetingTimes, false),
UpdateContext({_loadingMeetingTimes: true}),
Set(_selectedMeetingTime, Blank()),
Set(_selectedRoom, Blank()),
Set(_roomListSelected, false),
/*
Uses Office365 Outlook FindMeetingTimes operation to find available meeting
times given various parameters including a semicolon separated list of
attendee email addresses, a meeting duration (in minutes), an acceptable
start and end range to find the meeting. Attendee emails are retrieved from
the MyPeople collection. Duration and Start/End are retrieved from their
respective dropdowns, where Start is set to 8:00 AM on the date selected,
and End is set to 5:00 PM on the date selected.
*/
ClearCollect(MeetingTimes, AddColumns(Office365.FindMeetingTimes(
    {RequiredAttendees:Concat(MyPeople, UserPrincipalName & ";"),
MeetingDuration:drpMeetingDurationSelect.SelectedText.Minutes,
    Start:Text(DateAdd(dateMeetingDateSelect.SelectedDate, 8, Hours), UTC),
End:Text(DateAdd(dateMeetingDateSelect.SelectedDate, 17, Hours), UTC),
    MaxCandidates:15, MinimumAttendeePercentage:1, IsOrganizerOptional:
false, ActivityDomain: "Work"}).MeetingTimeSuggestions,
"StartTime", MeetingTimeSlot.Start.DateTime, "EndTime",
MeetingTimeSlot.End.DateTime))
);
UpdateContext({_loadingMeetingTimes: false});
Set(_showMeetingTimes, true)
```

All in all, a very, very useful feature from Reza!

Summary

The way we currently work is starting to feel slightly more normal every day. That said, we should always be exploring new ways to make our everyday lives more manageable and, without a doubt, the Meeting Capture app within Microsoft Teams offers an increased amount of efficiency!

There are plenty of really excellent OOTB Power Apps from Microsoft that can be created at the click of a button from the make.powerapps.com website – from apps that help us with Onboarding, Praise, Inventory Control, Ticketing and much more. And best of all, as demonstrated here, these only form a base for us to continue building on. Reza has done a fantastic job with his Meeting Notes V2, and I've found it extremely useful to have such a great foundation to continue the build to make it even more relevant to how I like to operate!

If you are anything like me, I can only recommend getting stuck into the available templates to kick off your creative sparks. Before long, you’ll be extending your Teams capabilities beyond your wildest dreams – good luck!