Microsoft Power Platform   Microsoft Teams

Building solutions with Dataverse for Teams

Lab 2 – Advanced data models and apps

Workshop Version: 1.3, Published: 08-2022

Table of contents

Exercise 1:
Advance data model

Objectives:

- Create additional tables in Dataverse for Teams
- Define relationships between tables
- Set up table permissions

Estimated time:

20 minutes

Task 1: Navigate to Dataverse for Teams content

> ❗ **Important**
>
> This routine will be referenced as **"Navigate to Dataverse for Teams content"** and will be repeated many times during this course.

1. In **InPrivate/Incognito** browser mode, while logged in into your **M365xXXXXXX lab tenant**, navigate to https://teams.microsoft.com

2. Open **Power Apps** app using the left bar
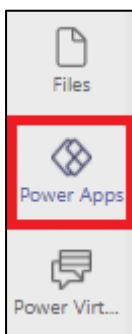
3. Select **Build** tab, then find and select your Team
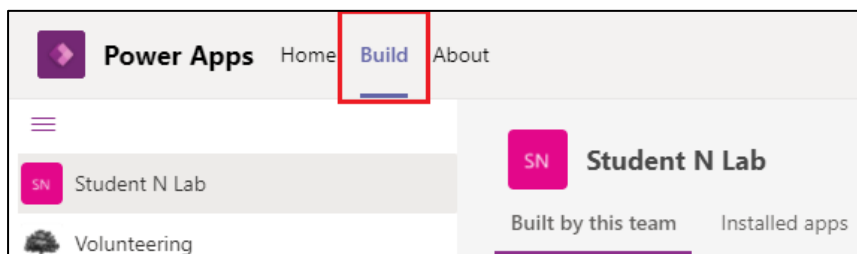
4. In **Items created for ...** , click **See all** link
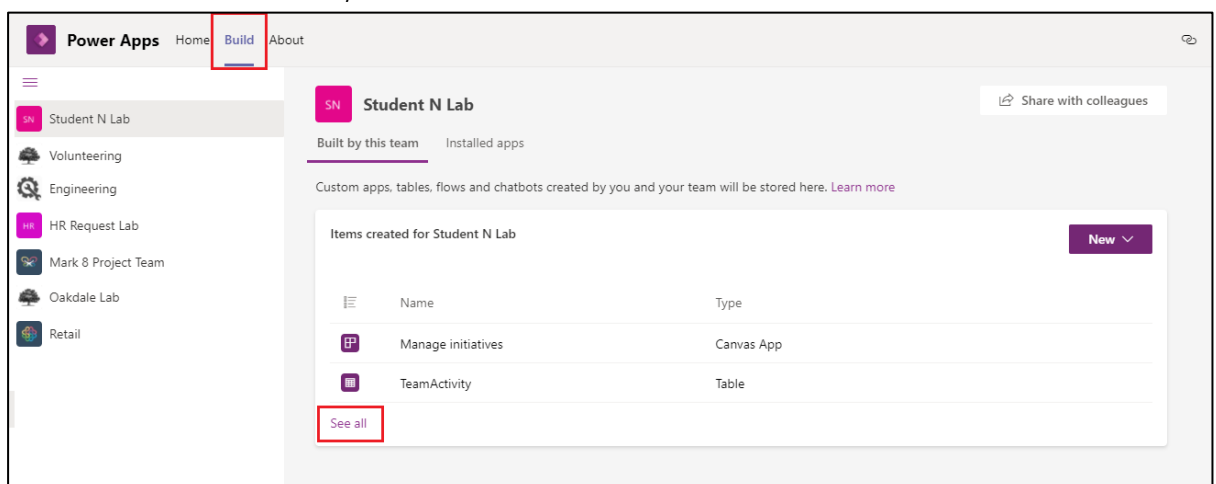
5. In this interface, you can see all the content created in your Team's **Dataverse for Teams** environment:
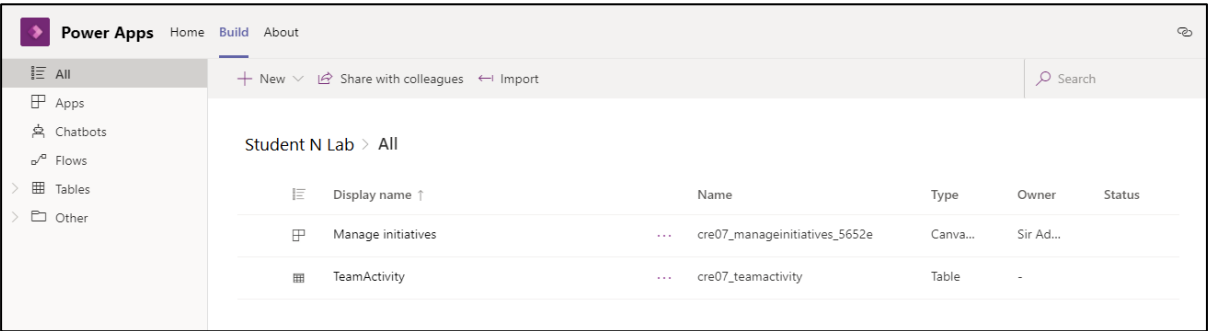
Task 2: Create Registration table and relationship

1. **Navigate to Dataverse for Teams content** of your Team

2. Click **+New** and select **Table** in a dropdown menu:
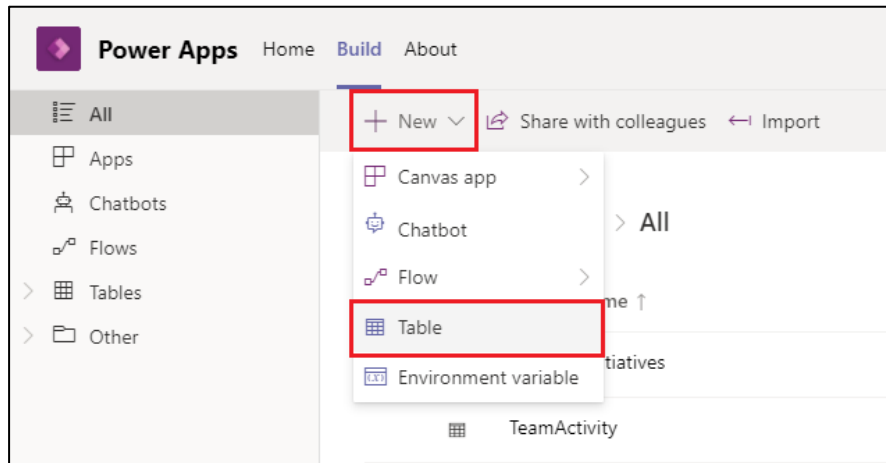
3. Set **Display Name** – **Registration** and click **Save** button

4. Select newly created **Registration** table details will get displayed. In a toolbar, click **Edit** to open table editor

5. Now you can modify table in the same way, you did in the previous lab.
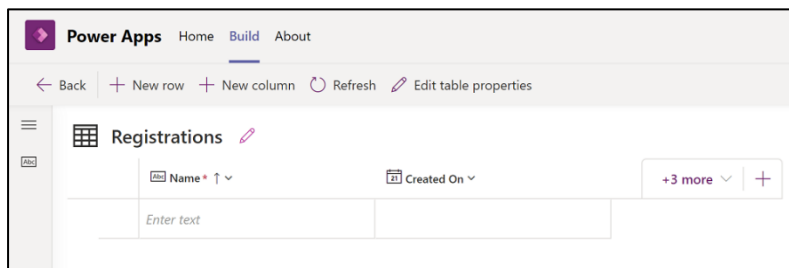
6. Let's add a few columns:

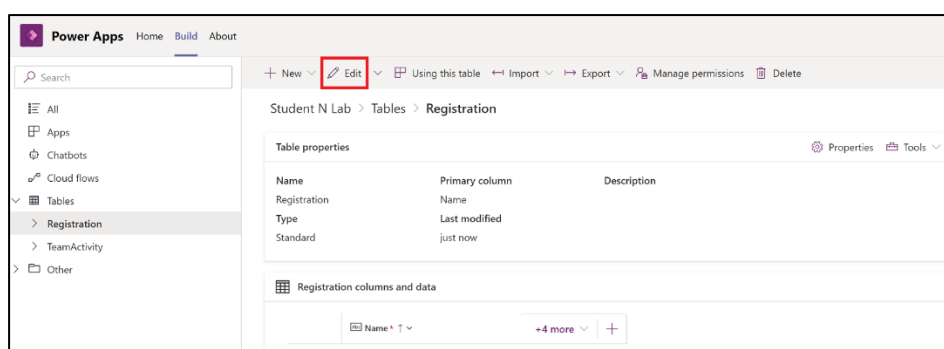| Name	Type	Advanced Settings
UserEmail	Email	Max length: 100
ApprovalStatus	Choice	Choices: - In waitlist - Approved - Rejected

Click **+ New choice** to add a new choice.

Click 🎨 **Palette icon** if you want to set a colour for the choice.

7. Now we will add a many-to-one relationship between **TeamActivity** and **Registration** by creating a **Lookup** column:

This table will be used later to track user registrations for Team initiatives.

8. Add two sample entries *(don't forget to replace xXXXXXX)*:

    a. **Name:** Adele Vance
    UserEmail: adelev@m365xXXXXXX.onmicrosoft.com
    ApprovalStatus: In waitlist
    relTeamActivity: Team Building CY2023

    b. **Name:** Megan Bowen
    UserEmail: meganb@m365xXXXXXX.onmicrosoft.com
    ApprovalStatus: Approved
    relTeamActivity: Team Building CY2023

9. Task is completed.

Task 3: Set table permissions

1. If you just finished previous task and are still in the table editor, click **Back**. Otherwise, **Navigate to Dataverse for Teams content** (see Task 1)
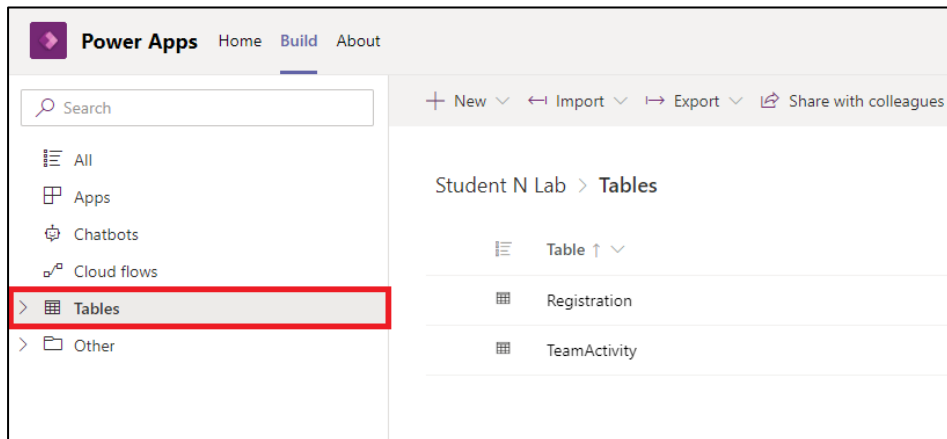
2. Select **Tables** in left navigation menu:

3. To comply with solution security requirements, we need to define table permissions:

   Team Owners can perform any operations with table in Dataverse for Teams environment of their team.

   Team Members should have limited permissions for certain tables:

   - *TeamActivity* – only team owners can create, update, delete, but other team members should be able to read-only

   - *Registration* – team members can only create requests, but not modify them

4. Click **ellipsis (...)** to open context menu for **Registration** table and select **Manage permissions option**:
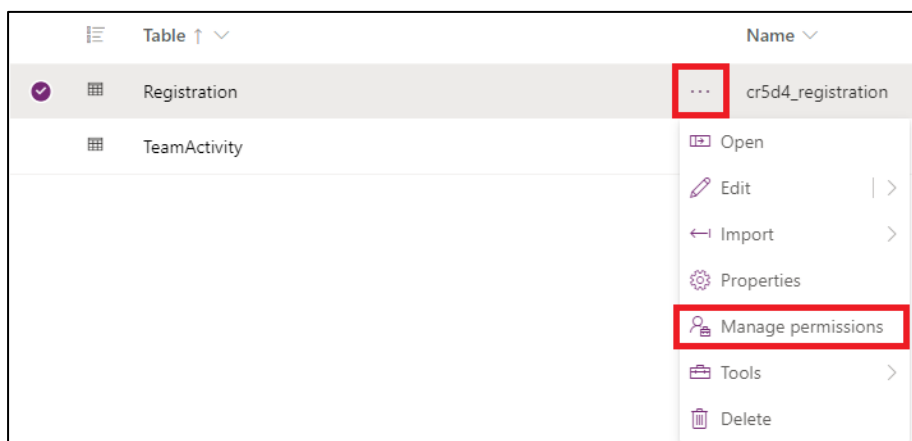
5. Set **Registration** table permissions for **Members** as **Collaborate**
   Click **Save** button

6. **Repeat steps 2-5** to set up permissions for **TeamActivity** table:

   TeamActivity –> Members – Reference

7. Task is completed.

Optional task: Create Points table to store award points balance

1. **Navigate to Dataverse for Teams content** of your Team

2. Click **+New** and select **Table** in a dropdown menu:
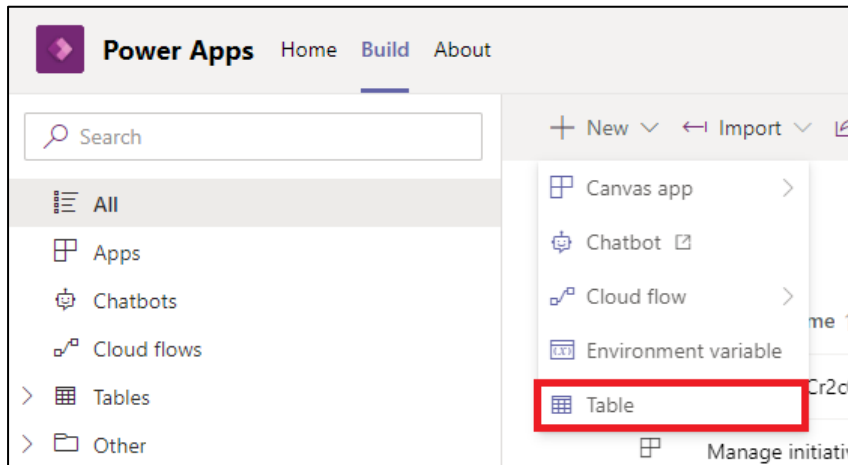
3. Set **Display Name** – **Points** and click **Save** button

4. Click **Edit** in toolbar to start the table editor

5. Let's add a few columns:

Name	Type	Advanced Settings
UserEmail	Email	Max length: 100
Balance	Number	Minimum value: 0

6. Add a few sample entries *(don't forget to replace xXXXXXX)*:

   a. **Name:** Adele Vance
      UserEmail: adelev@m365xXXXXXX.onmicrosoft.com
      Balance: 1000

   b. **Name:** System Administrator
      UserEmail: admin@m365xXXXXXX.onmicrosoft.com
      Balance: 750

7. Task is completed.

Exercise 2:
Build Power App from scratch

Objectives:

- Learn how to use galleries to display and select data entries
- Use the advantages of Dataverse for Teams relational database
- Utilize Power Apps formula language to implement business logic

Estimated time:

40 minutes

Task 1: Create a new Power App and connect to data

1. **Navigate to Dataverse for Teams content** of your Team

2. Create a new Power App, click **+New** -> **Canvas App** -> **Tablet form factor**
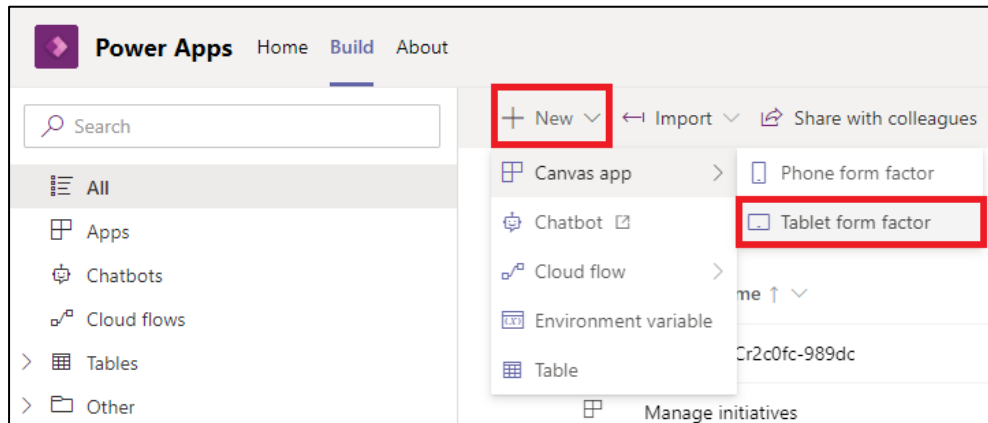
3. Name a new app as **Team Initiatives** and click **Save**
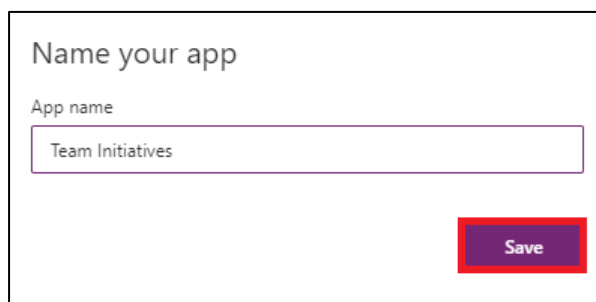
4. In Data ⬡ panel, click **+Add data** and add **TeamActivities** and **Registrations** tables to the app:
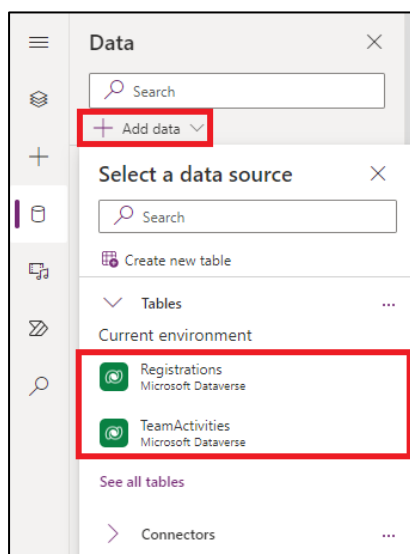
5. In Data ▯ panel, click **+Add data** and in **Search** type *Office 365 Users*.
    Add **Office 365 Users** connector to the app by clicking on it.
    We will use this connector to query user profile (e.g., extract user photo from directory)

    Click **Connect** in the right panel.
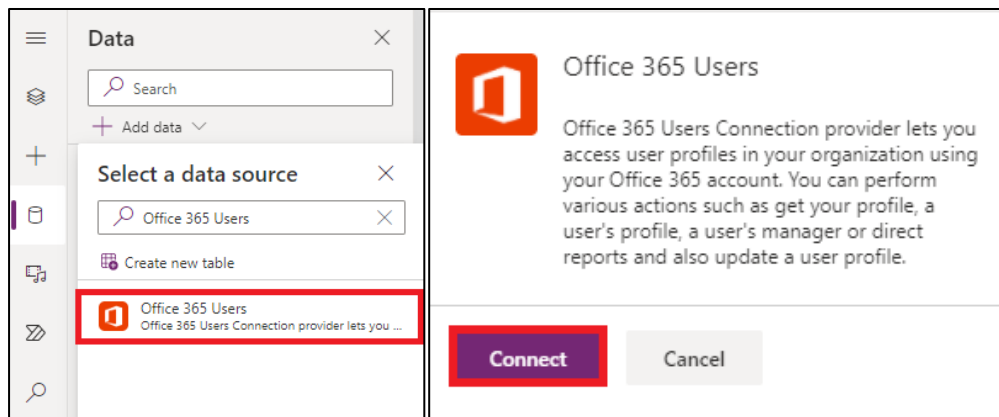
> ❗ **Localized connector names**
>
> Office 365 Users connector may have different name in your language if you have different locale settings on your computer.

6. The list of connected data sources should look like this:
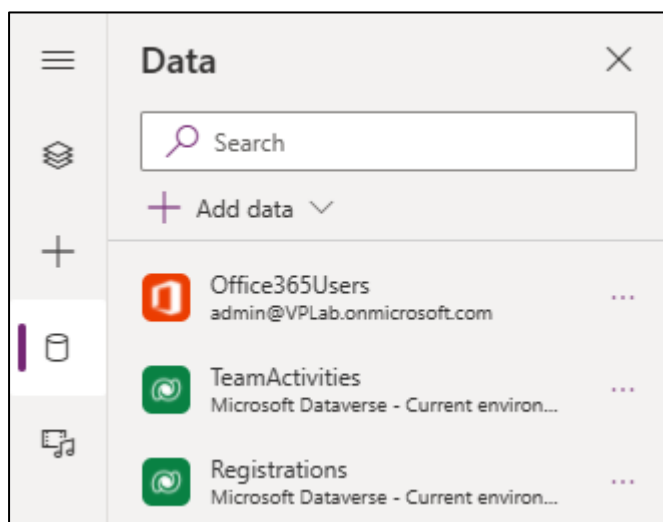
Order of data sources is NOT important.

Task is completed. Proceed to the next task.

Task 2: Create layout and learn naming conventions

> ## ⚙ Canvas Apps Coding Standards and Guidelines
>
> In this lab, we will rely on the best practices and naming conventions defined in Canvas apps coding standards and guidelines.
>
> Full document link: https://aka.ms/powerappscanvasguidelines

1. Switch left panel to **Tree View** :
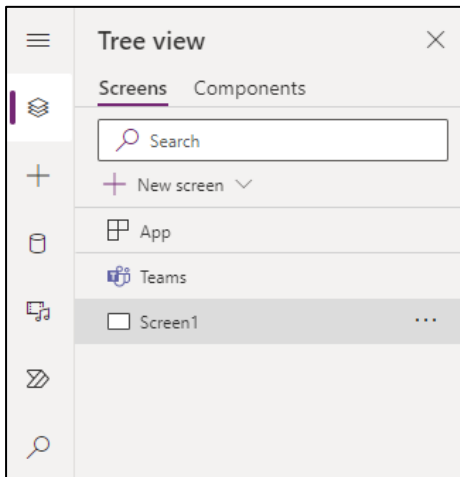
2. Notice **App** and **Teams** objects:

   a. **App** is a top-level object, that represents our application. **OnStart** property, is especially interesting for us – it allows us to define, what commands should be executed when the app starts.

   b. **Teams** is an integration object unique for apps created in Dataverse for Teams. It contains the details of the current context in Microsoft Teams: current team, channel, colour theme information etc

3. Right-click on **Screen1** in **Tree View** and select **Rename** it as **Main Screen**
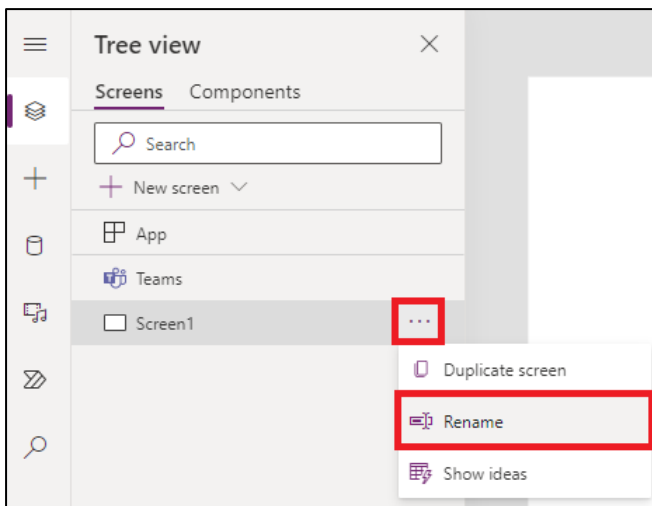
4. Open **Insert** ⊥ pane and add **2 Vertical Galleries** and **1 Display Form** to **Main Screen**
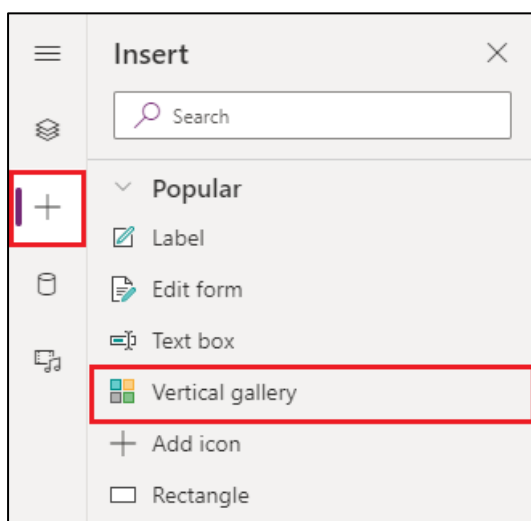
We will utilize this method to add controls to the screen during this lab. Use **Search** to find control that you need, then select it.

5. Move and resize galleries and display form to achieve the following layout on **Main Screen**:
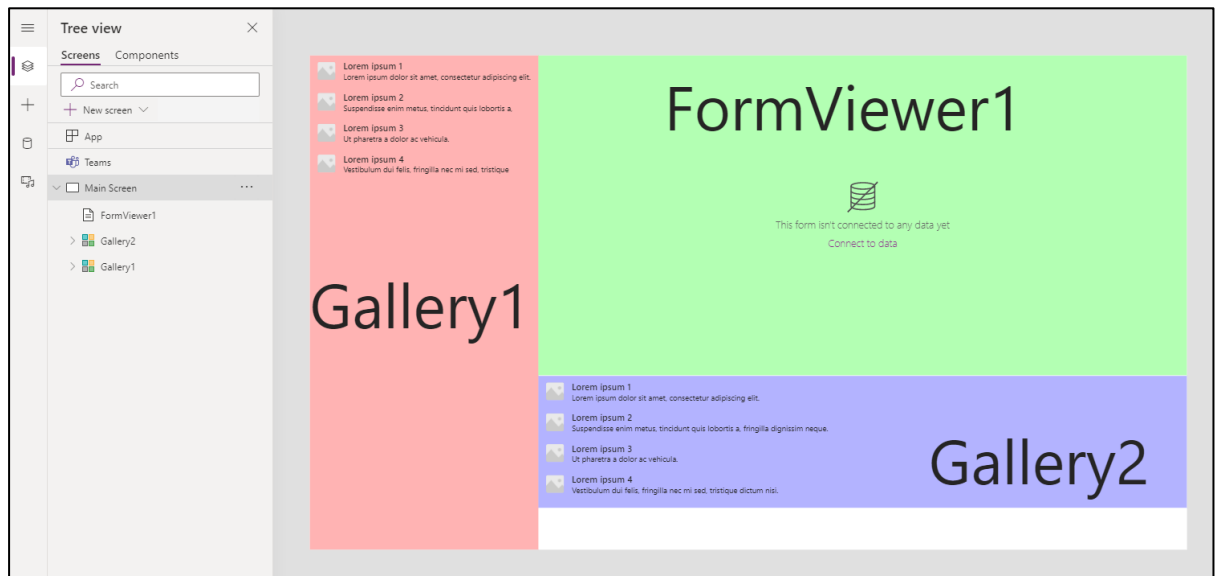
Note: Not actual colours – just a layout example.

6. In **Tree View** pane, **Rename** controls according to naming conventions:

- `Gallery 1 -> galInitiatives`

- `Gallery 2 -> galRegistrations`

- `FormViewer1 -> frmInitiativeDetails`

Tree View should look like this as a result:
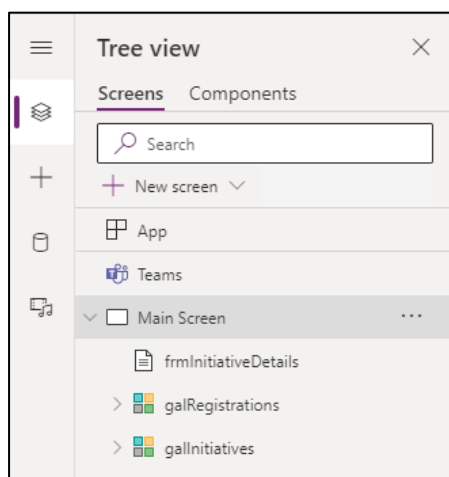
⚙️ Controls naming

All control names on the canvas should use camel case. They should begin with a three-character type descriptor, followed by the purpose of the control. This approach helps identify the type of control and makes it easier to build formulas and search.

Here are good examples: `lblUserName, btnSend, imgLogo`

3-character type descriptors for the controls in this lab:

- `lbl` - label
- `btn` – button
- `frm` – form
- `gal` – gallery
- `img` – image
- `tim` - timer

Don't forget to save your app.
Task is completed.

Task 3: Setting up Initiatives gallery

⚙ Gallery control

Gallery is one of the most useful Power Apps controls, when we are working with data or collections of objects.

Gallery displays the repetitive data, based on pre-defined template.
Template can contain not only labels but any controls that you want.

1. Select **galInitiatives** and connect it to **TeamActivities** data source
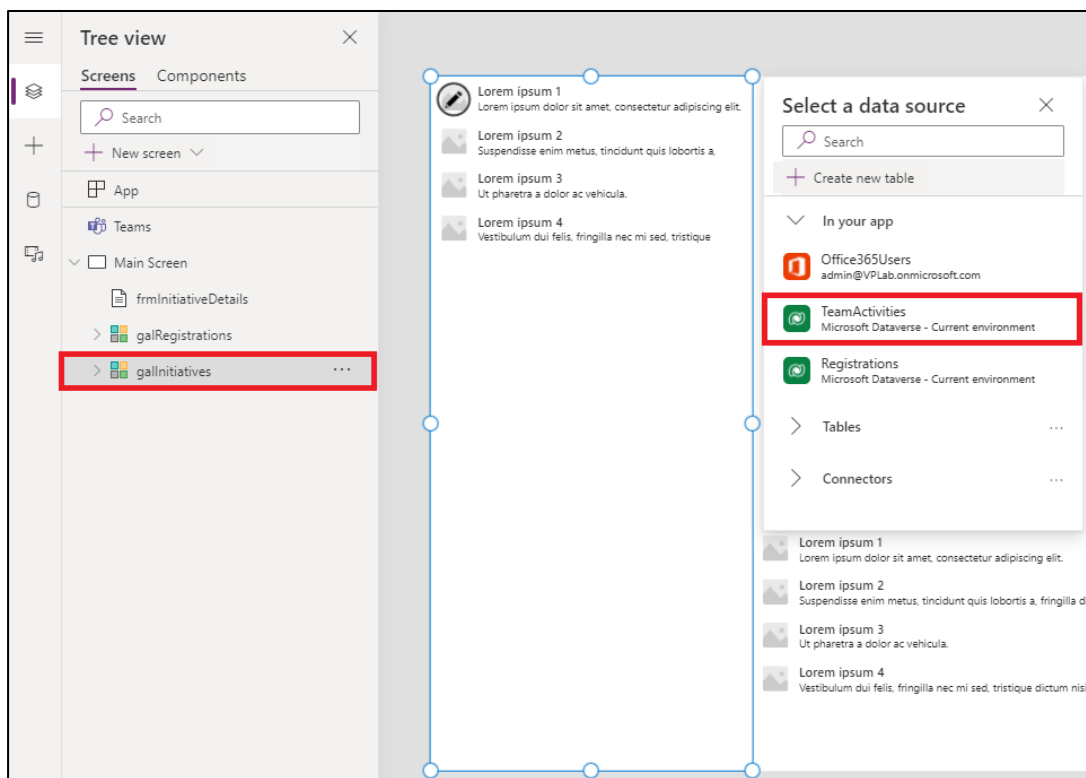
2. Try to use some pre-defined gallery templates.

    In **Properties** pane, change **Layout** property, set it as *"Title and subtitle on overlay"*
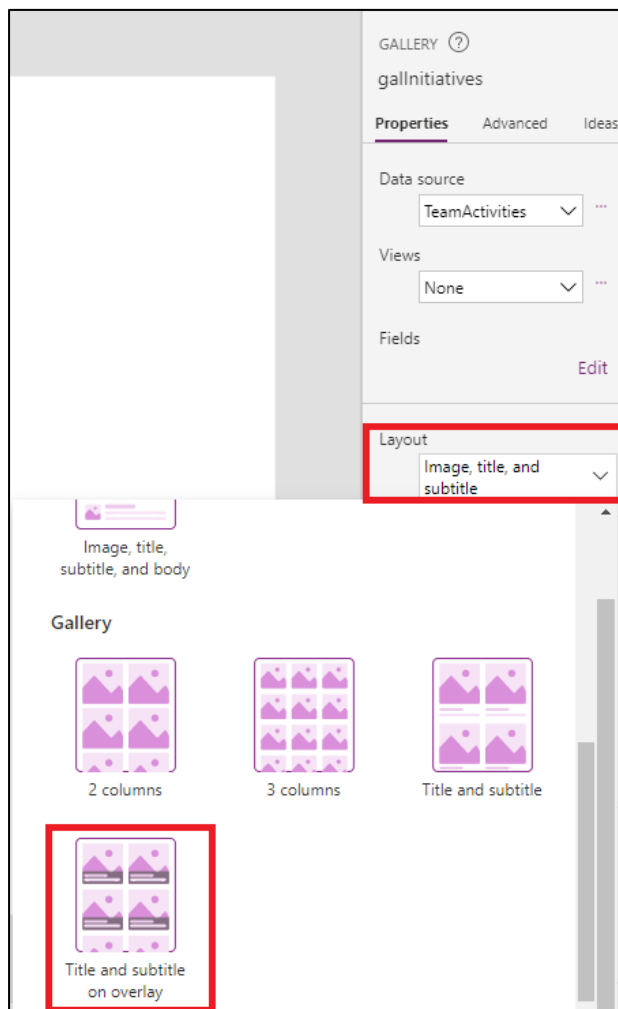
3. Change one more property for **gallInitiatives**, set **Wrap count** property as **1**

    You can also modify template size (height) and template padding if you want.
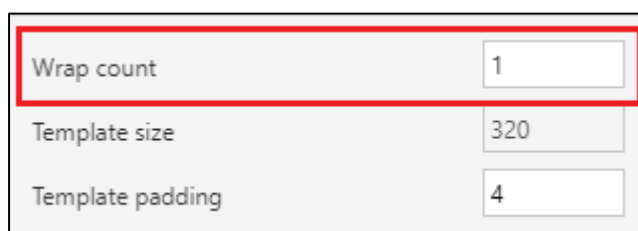
4. In **Tree View** ⊜ expand **gallnitiative** and select an image within it, rename it as **imgInitiative**
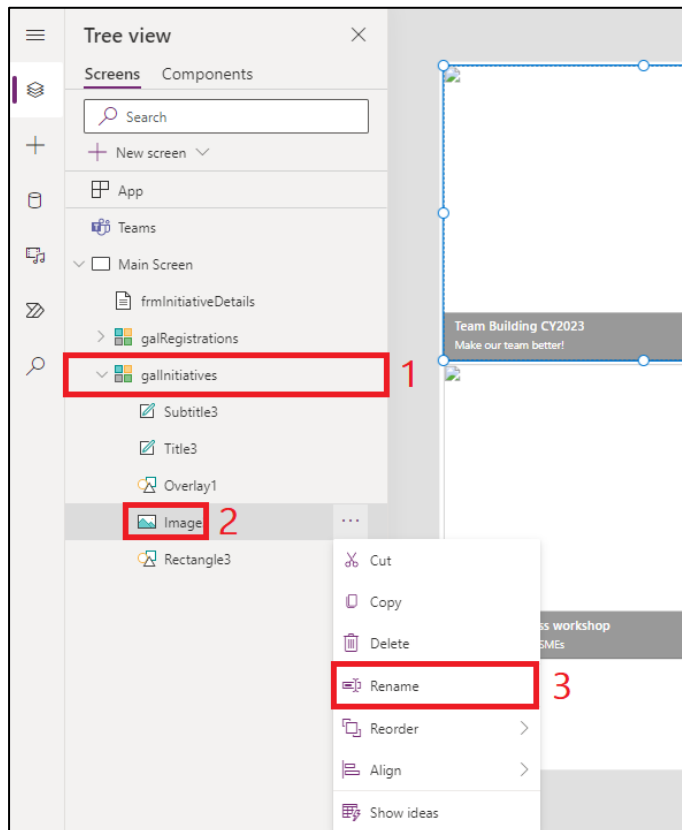
Controls within gallery are parts of its template.

5. For **imgInitiative**, set **Image** property as `Text(ThisItem.ImageURL)`

   ImageURL type is hyperlink, use `Text()` function to convert it to text
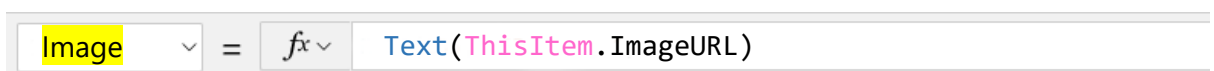
   If the images are still not displayed in the gallery – double-check ImageURL links in your TeamActivities table.

6. In **Tree View** ⊜ expand **gallnitiative** and select a Rectangle within it.

   Rename it as **shpHighlight**
   Use **Reorder -> Bring to front** to put this invisible rectangle on top of gallery template.
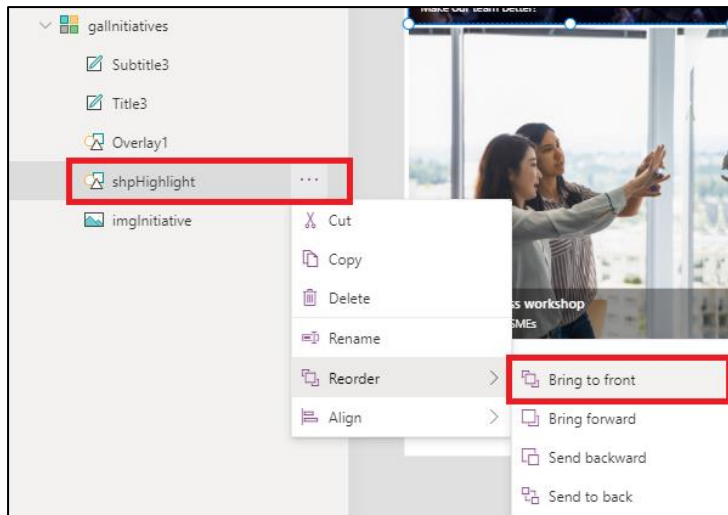
7. For **shpHighlight**, set **Fill** property as *(go to localized formula)*

```
Fill           ∨  =  ƒx ∨   If(ThisItem.IsSelected,RGBA(0,0,100,0.2),RGBA(0,0,0,0))
```
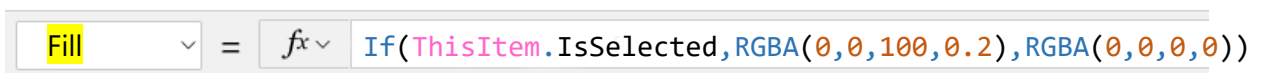
⚙️ **Conditions and conditional formatting**

`If(condition, then, else)` – works in the same way as in Excel.

In this case, we are using it to implement conditional formatting.
If the initiative is selected, then it should be highlighted (fill of rectangle changed).
Technically, you can apply this method to any property of the control.

`RGBA(red,green,blue,alpha)` - function defines colour.

`RGBA(0,0,100,0.2)` – transparent light blue colour
`RGBA(0,0,0,0)` – absolutely invisible black colour

Now, try to Alt-click on initiatives in gallery and colour should slightly change if the item is selected.

Don't forget to save your app. Task is completed.

Task 4: Setting up Initiative display form

1. Select **frmInitiativeDetails** and set **Data source** property as **TeamActivities**
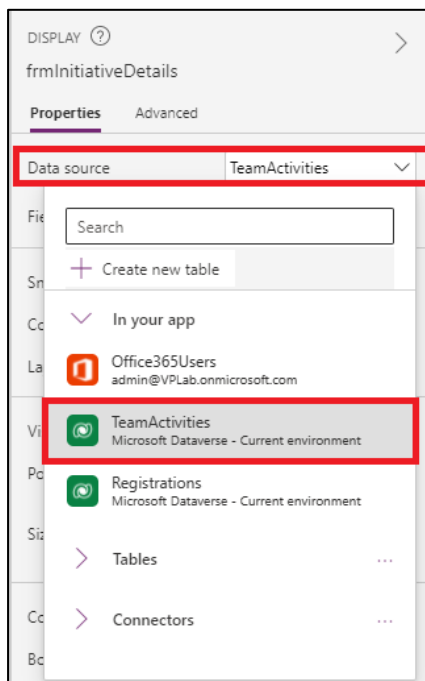
2. Let's hide some fields, that we don't want to display to users.

   Click **Edit fields** in **Property** pane.

   This interface can be used to add, reorder or remove form fields.

   Remove **Short description, ImageURL** and **Organizer** from the form, if they are present.

   If any fields are missing, then add them using **+Add field** button on top.
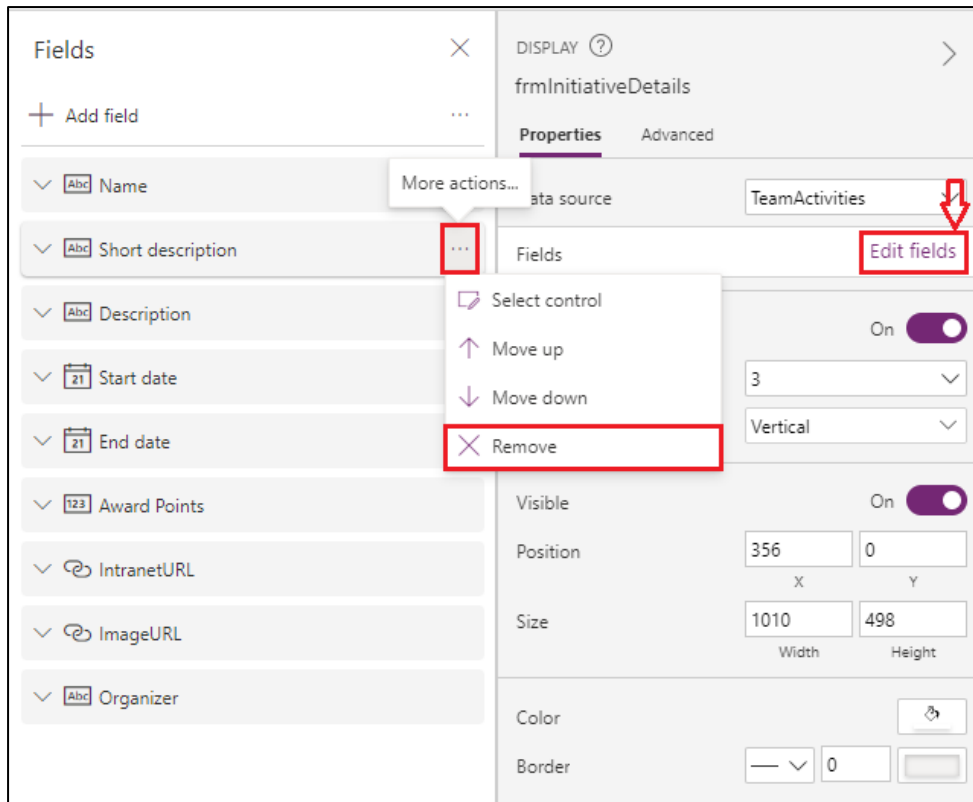
3. Change **Columns** property to **2** and resize form elements to make it look nicer.

4. For **frmInitiativeDetails**, set **Item** property as

Form will display Initiative details of currently selected item in **galInitiatives**.

5. Your app may look like this:
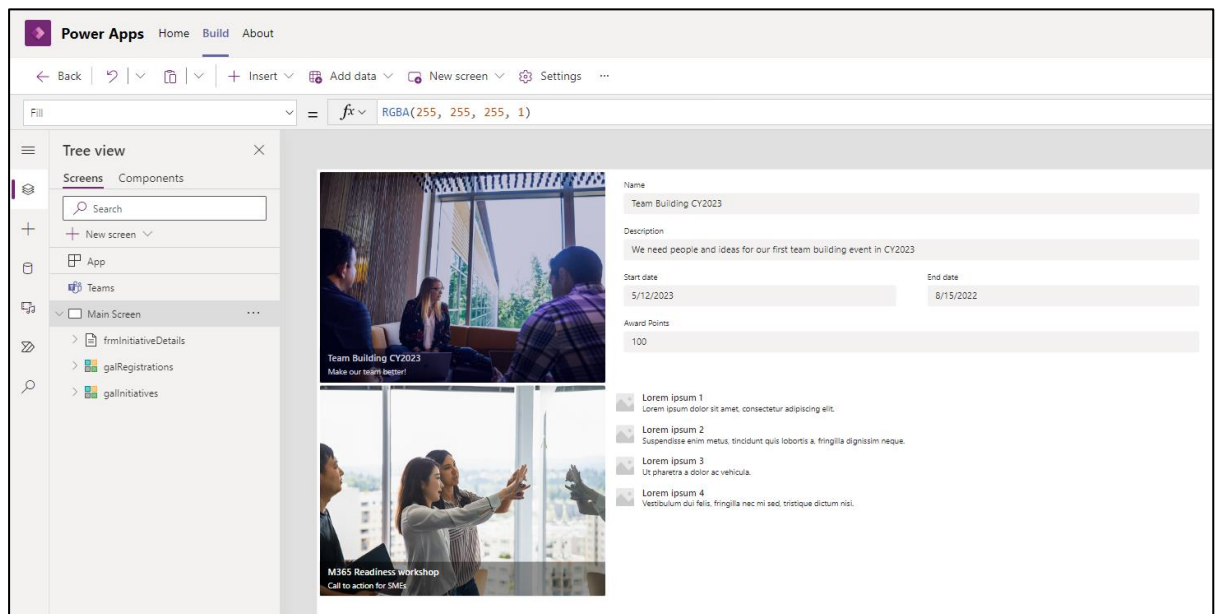
Don't forget to save your app. Task is completed.

Task 5: Setting up Registrations gallery

1. Select **galRegistrations** and set **Registrations** as a data source for it
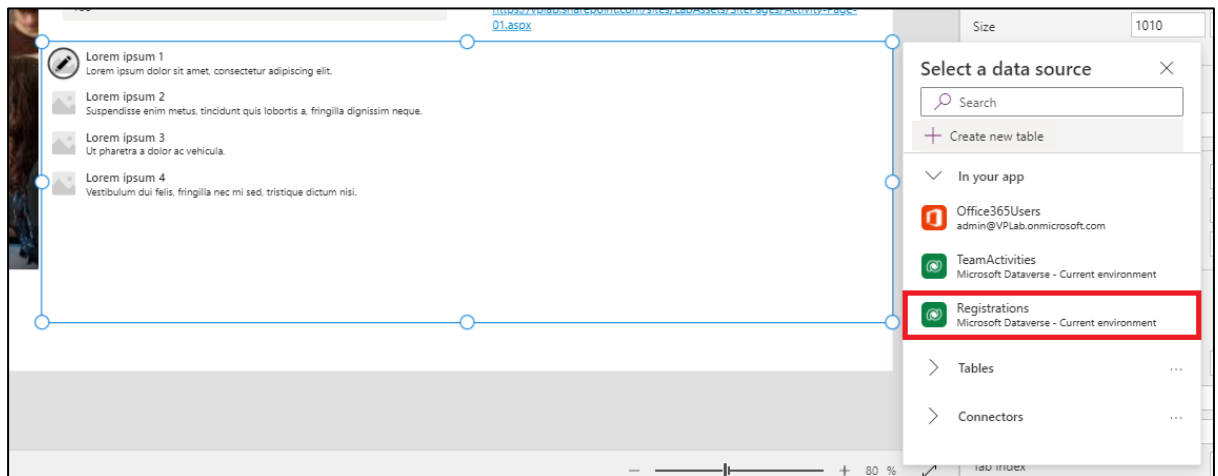
2. Resulting gallery doesn't look useful.
Let's modify look and feel first and then we will filter data properly.

    Expand **galRegistrations** in **Tree View** ⬢ and select title label (e.g., Title3).

    Rename it as **lblUserName**

3. For **lblUserName**, check that **Text** property is

Text	∨	=	_fx_ ∨	ThisItem.Name

4. Select subtitle label in **galRegistrations** (e.g., Subtitle3).

    Rename it as **lblRegisteredOn**.

5. For **lblRegisteredOn**, set **Text** property as

Text	∨	=	_fx_ ∨	"Registered on: " & ThisItem.'Created On'

    > ⚙ **Text concatenation**
    >
    > Ampersand **&** is a concatenation operator (glues two strings together), when used between text values in Power Apps formula language.
    >
    > In this case we start with static text **"Registered on: "** in double-quotes and then we add a dynamic value `ThisItem.'Created On'`
    >
    > (single quotes are used when column name contains spaces)

6. Select image in **galRegistrations** (e.g., Image3).

   Rename it as **imgUserPhoto**.

7. For **imgUserPhoto**, set **Image** property as *(go to localized formula)*

```
Image  =  fx∨   If(

              !IsBlank(ThisItem.UserEmail),

              Office365Users.UserPhotoV2(ThisItem.UserEmail)

          )
```
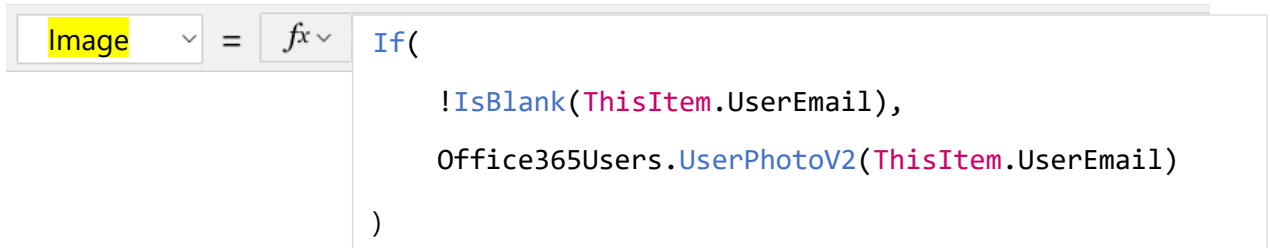
⚙️ **Using connector actions in Power Apps**

Office365Users represents data connector, that we added to this app in the beginning.

It allows to execute actions in the source system, just like we do in Power Automate flows.

Office365Users.UserPhotoV2(<User Id or UPN>) returns user photo.

We also need to check if UserEmail is blank before getting photo to avoid an error.

You may find many other uses to this connector, when you need to search for people, get profile details or traverse organizational structure.

8. **galRegistrations** should look like this now:
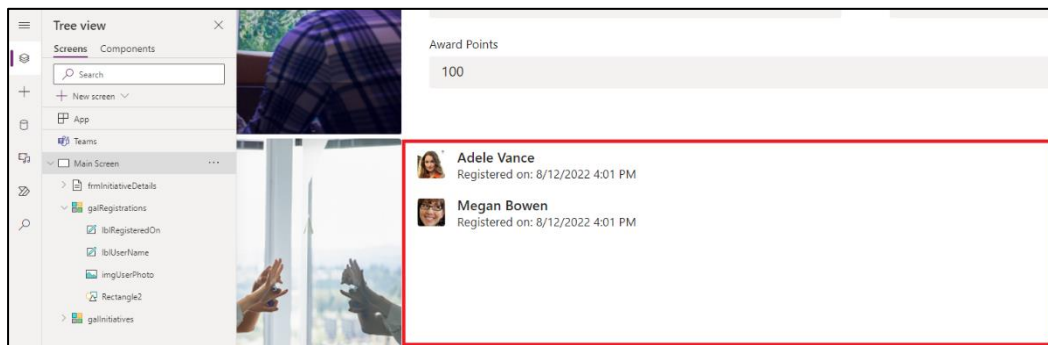
9. Select **galRegistrations** and then click **Edit gallery** icon ✏️.
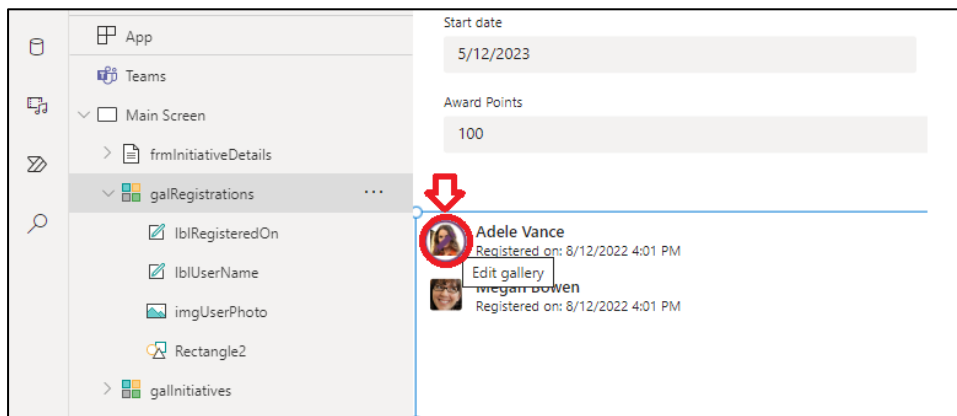
Switch to **Insert** ➕ pane and add a **Label** and a **Button** to the template.
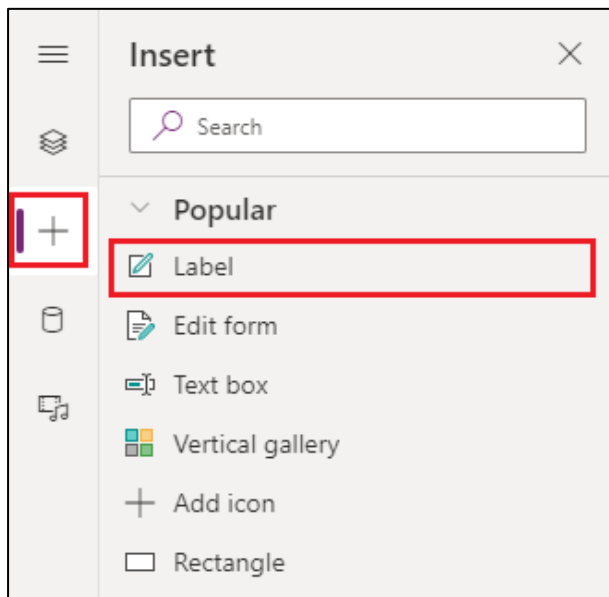
Add the button in the same way.

10. Open **Tree View** ≋ and make sure that label and button are added **WITHIN galRegistrations**, not outside of it.
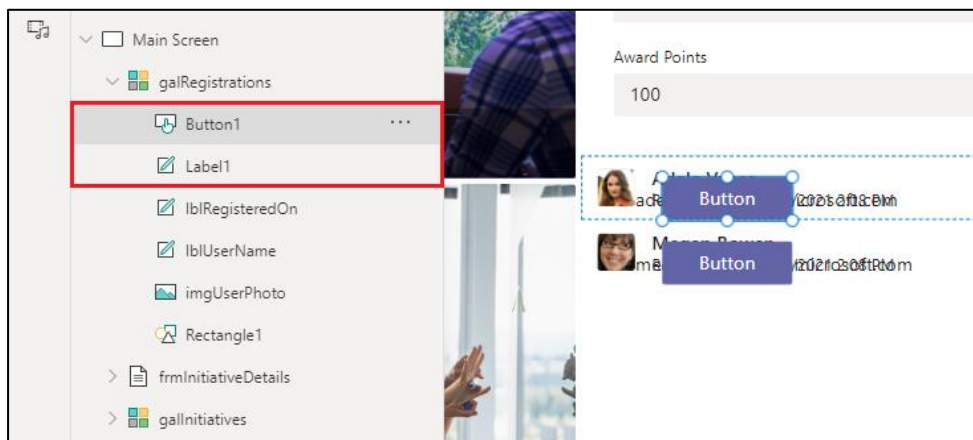
If the controls were created outside of gallery, you won't see them replicated for each gallery item – just recreate them **WITHIN** the gallery.

11. Rename controls and move them within the template

    a. **Button1 -> btnRevoke**

    b. **Label1 -> lblStatus**

Result may look like this:
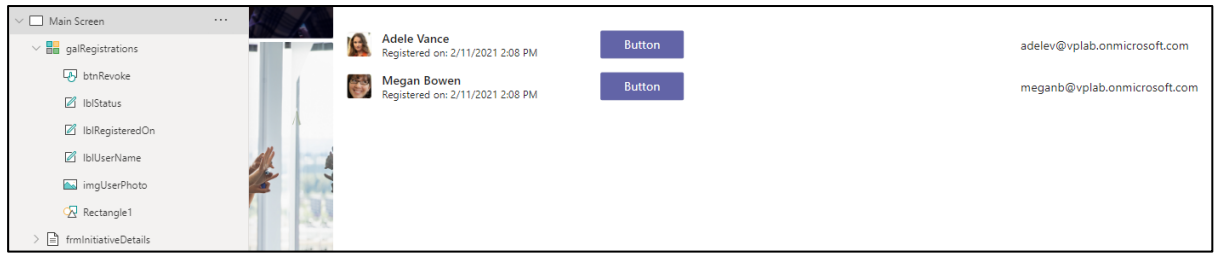
12. For **btnRevoke**, set **Text** property as

| Text | ∨ | = | *fx* ∨ | "Revoke" |

13. For **lblStatus**, set **Text** property as

| Text | ∨ | = | *fx* ∨ | ThisItem.ApprovalStatus |
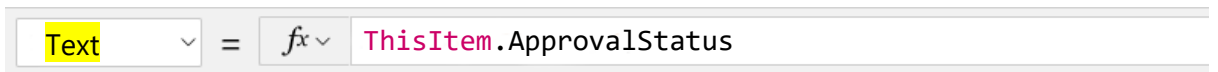
14. For **lblStatus**, set **Color** property as *(go to localized formula)*

| Color | ∨ | = | *fx* ∨ |

```
Switch(
    ThisItem.ApprovalStatus,
    'ApprovalStatus (Registrations)'.'In waitlist',
    Color.Gray,
    'ApprovalStatus (Registrations)'.'Approved',
    Color.Green,
    'ApprovalStatus (Registrations)'.'Rejected',
    Color.Red
)
```
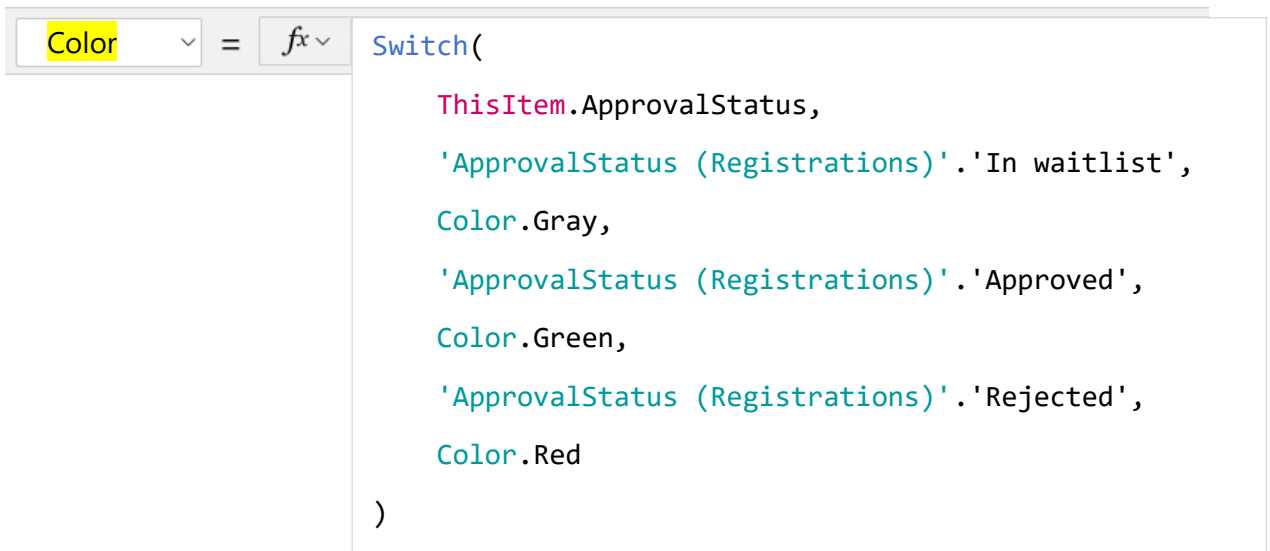
Let's use **Switch**, instead of, **If** this time.
This formula will enable conditional formatting for **ApprovalStatus** choices.

> ### ⚙️ Enumerations and Dataverse Choices data type
>
> You can notice, that instead of `RGBA()` function, this time colours defined differently.
>
> `Color.Green` – we use built-in `Color` enumeration, that can be easier sometimes, then defining `RGBA()` parameters or HEX-code in `ColorValue()` function.
>
> Power Apps has many built-in enumerations, here are some examples:
>
> `Color` – pre-defined colours (e.g., `Color.Red`);
>
> `DisplayMode` – control modes (e.g., `DisplayMode.Disabled` or `DisplayMode.Edit`)
>
> `ScreenTransition` – screen navigation animations (e.g., `ScreenTransition.Cover`)
>
> ---
>
> Dataverse **Choices** data types are represented as enumerations that contain all possible options, that you defined, when you created a column of this data type.
>
> Condition `ThisItem.ApprovalStatus = "In waitlist"` **will not work**, because of type mismatch.
>
> Use `'ApprovalStatus (Registrations)'` enumeration instead. It became automatically available, when you added **Registrations** table as a data source:
>
> | fx ∨ | `'ApprovalStatus (Registrations)'` |
> | --- | --- |
> | | Approved |
> | | 'In waitlist' |
> | | Rejected |
>
> Condition
> `ThisItem.ApprovalStatus = 'ApprovalStatus (Registrations)'.'In waitlist'`
> will work correctly.
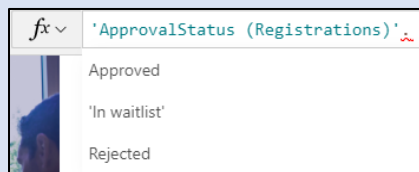
15. There is a **relationship** between **Registrations** and **TeamActivities** tables, that was created in the previous exercise.
    We can utilize it to easily **filter** registrations for the selected Initiative only.

    For **galRegistrations**, set **Items** property as *(go to localized formula)*

Items	∨	=	`Filter(`
			`Registrations,`
			`relTeamActivity.TeamActivity=galInitiatives.Selected.TeamActivity`
			`)`
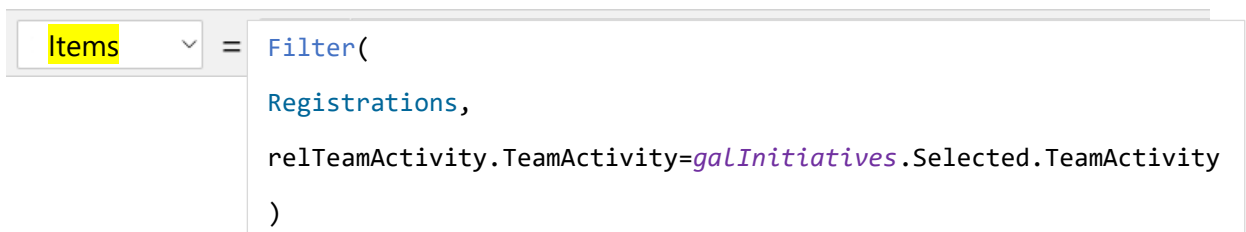
    Filtering query will be **delegated** to Dataverse for Teams database – it allows to achieve good performance, even on large data sets.

16. Try to navigate through Initiatives by Alt+Clicking on **galInitiatives**.

    You will see error messages and some controls displayed, even if the gallery is empty.

    > ⚙️ **Handling gallery display errors**
    >
    > Generally, there are two ways to resolve gallery display errors, that we're facing here:
    >
    > **1) Perform error handling and value validation for each control in gallery template, hide if empty or display a placeholder (more robust way)**
    >
    > It will help to handle any data in gallery no matter of data quality.
    >
    > **2) Hide gallery when it's empty (quick)**
    >
    > Doesn't resolve any data display issues – just hides empty gallery.

    Let's imagine that we trust our data source quality and will just hide an empty gallery.
    For **galRegistrations**, set **Visible** property as *(go to localized formula)*

    ```
    Visible   ∨  =  fx ∨   If(!IsEmpty(galRegistrations.AllItems), true)
    ```

    Once again, try to navigate through Initiatives by ALT+Clicking on
    galInitiatives – nothing will be displayed for Initiatives without Registrations.

    Don't forget to save your app.
    Task is completed.

Task 6: Add and delete Registrations

1. Go to **Insert** + pane and add a new **button** to **Main Screen**

2. Rename this button as **btnParticipate** and change button text to *"Participate"*
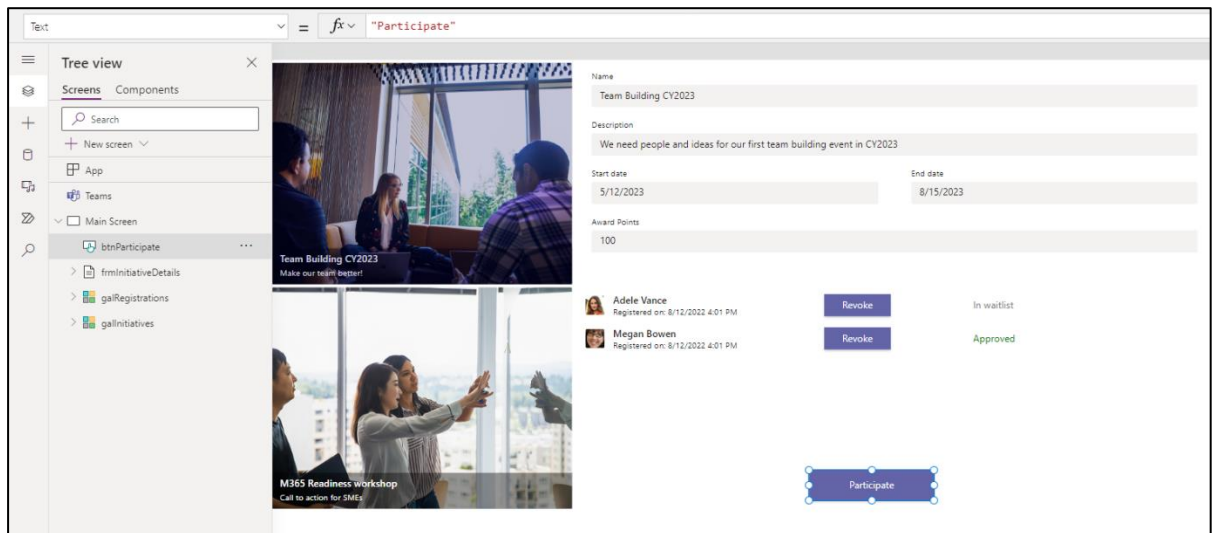
3. To add a new registration for the currently selected event, set **OnSelect** property of **btnParticipate** as *(go to localized formula)*

```
OnSelect = fx  Patch(

    Registrations,

    Defaults(Registrations),

    {

        Name: User().FullName,

        UserEmail: Lower(User().Email),

        relTeamActivity: galInitiatives.Selected,

        ApprovalStatus:
'ApprovalStatus (Registrations)'.'In waitlist'

    }

);
Refresh(Registrations);
Notify("Your registration has been added.");
```

4. **Save** your app. Start **Preview** mode and try to click **btnParticipate**.

   Give it a moment. Your user should appear in Registrations list for the currently selected event. Close **Preview** mode.

5. **btnRevoke** must be shown only for current user's registration, let's add a condition to its visibility. *Invisible buttons are not clickable.*
   For **btnRevoke**, set **Visible** property as


```
Visible   =   fx   ThisItem.UserEmail=Lower(User().Email)
```

Button should be visible only for current user now:

6. To remove a registration for the currently selected event, set **OnSelect** property of **btnRevoke** as *(go to localized formula)*

```
OnSelect  =  fx v    Select(Parent);

                    Remove(Registrations, ThisItem);

                    Refresh(Registrations);

                    Notify("Your registration has been revoked");
```

7. **Save** your app. Start **Preview** mode and try to click **btnRevoke** near your registration entry.

   Give it a moment. Your user should disappear from Registrations gallery for the currently selected event. Close **Preview** mode.

8. We need to avoid adding the same user more than once.
   For **btnParticipate**, set **DisplayMode** property as *(go to localized formula)*

```
DisplayMode  =  fx v    If(

                            Lower(User().Email) in
                        galRegistrations.AllItems.UserEmail,

                            DisplayMode.Disabled,

                            DisplayMode.Edit

                        )
```

btnParticipate should be disabled now, if the current user is present in Registrations for the selected Initiative:

Adele Vance Registered on: 2/11/2021 2:08 PM		In waitlist
Megan Bowen Registered on: 2/11/2021 2:08 PM		Approved
Administrator Registered on: 2/12/2021 2:51 PM	Revoke	In waitlist
	Participate	
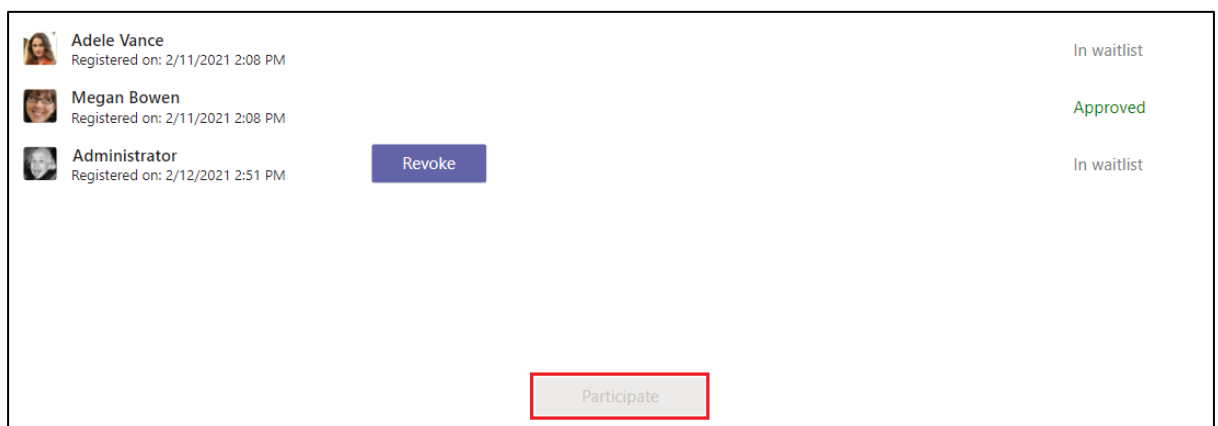
Save and **Publish** your app. **Task is completed.**

Appendix: Localized formulas

1. Exercise 2 – Task 3. Overlay colouring formula *(go back)*

```
Fill  = fx∨  If(ThisItem.IsSelected;RGBA(0;0;100;0,2);RGBA(0;0;0;0))
```

2. Exercise 2 – Task 5. Getting user's photo *(go back)*

```
Image  = fx∨  If(
                  !IsBlank(ThisItem.UserEmail);
                  Office365Users.UserPhotoV2(ThisItem.UserEmail)
              )
```

3. Exercise 2 – Task 5. Approval status conditional colouring *(go back)*

```
Color  = fx∨  Switch(
                  ThisItem.ApprovalStatus;
                  'ApprovalStatus (Registrations)'.'In waitlist';
                  Color.Gray;
                  'ApprovalStatus (Registrations)'.'Approved';
                  Color.Green;
                  'ApprovalStatus (Registrations)'.'Rejected';
                  Color.Red
              )
```

4. Exercise 2 – Task 5. Filtering Registrations *(go back)*

```
Items  =  Filter(
              Registrations;
              relTeamActivity.TeamActivity=galInitiatives.Selected.TeamActivity
          )
```

5. Exercise 2 – Task 5. Hiding empty gallery *(go back)*

```
Visible  = fx∨  If(!IsEmpty(galRegistrations.AllItems); true)
```

6. Exercise 2 – Task 6. Patching data to Dataverse table *(go back)*

```
OnSelect ˅ = fx ˅   Patch(

                        Registrations;

                        Defaults(Registrations);

                        {

                            Name: User().FullName;

                            UserEmail: Lower(User().Email);

                            relTeamActivity: galInitiatives.Selected;

                            ApprovalStatus:
                    'ApprovalStatus (Registrations)'.'In waitlist'

                        }

                    );;
                    Refresh(Registrations);;
                    Notify("Your registration has been added.");;
```

7. Exercise 2 – Task 6. Removing data from Dataverse table *(go back)*

```
OnSelect ˅ = fx ˅   Select(Parent);;

                    Remove(Registrations; ThisItem);;

                    Refresh(Registrations);;

                    Notify("Your registration has been revoked");;
```

8. Exercise 2 – Task 6. Conditional DisplayMode for button *(go back)*

```
DisplayMode = fx ˅   If(

                        Lower(User().Email) in
                    galRegistrations.AllItems.UserEmail;

                        DisplayMode.Disabled;

                        DisplayMode.Edit

                    )
```