

Hands-on lab

Lab: Xamarin Forms

September 2020

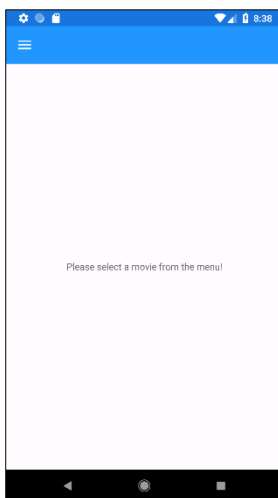
In this lab, you will build a Xamarin.Forms application that presents some of the Star Wars data from the previous lab. This data is stored on the device itself in a SQLite database. EF Core is used to access the database.

Note: SQLite is often used as a solution for storing offline data. EF Core facilitates this, but for the time being still suffers from some performance issues, e.g.:

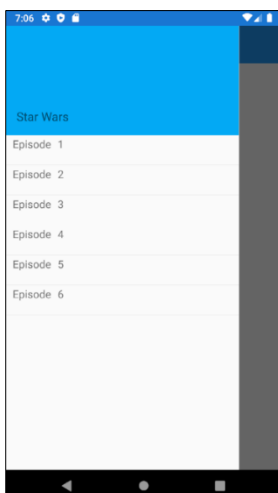
<https://github.com/aspnet/EntityFrameworkCore/issues/12087>

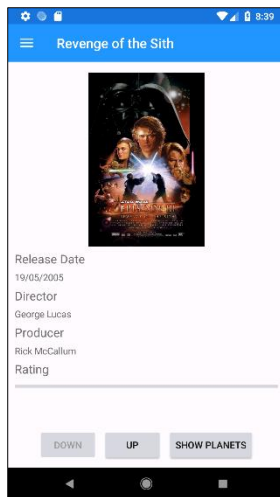
General application flow

When you start the application, there is an empty start page with a hamburger menu (three lines in the corner):



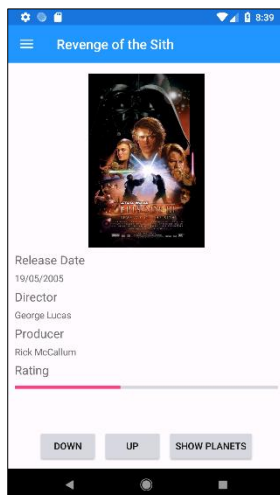
When the user taps the menu, a list of all movies (with episode number) is shown:



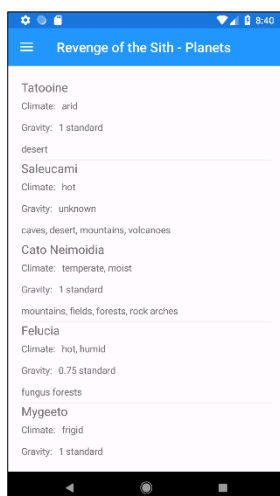


Selecting a movie shows some info, including a movie poster, director, etc.

Note the Up/Down buttons. With these buttons, the user can rate the movie. Buttons are disabled when pressing them makes no sense. E.g. you can't rate below 0 or above 10.



After tapping the “Show Planets” button, the next page fetches all planets appearing in that movie, with some details.

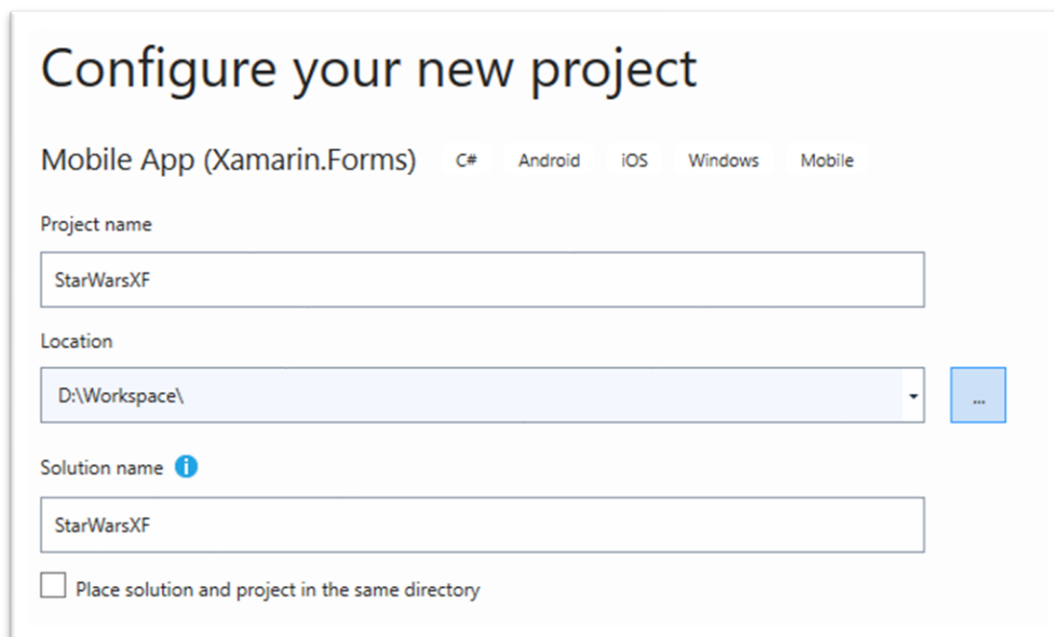
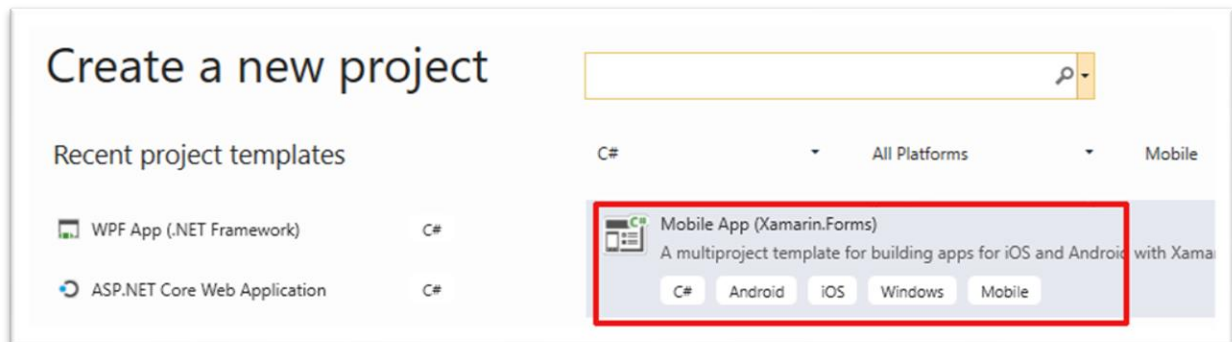


The application uses a MasterDetailPage to list all the movies and select one to go into detail. Detail pages are wrapped in a NavigationPage. Lists are presented in ListViews.

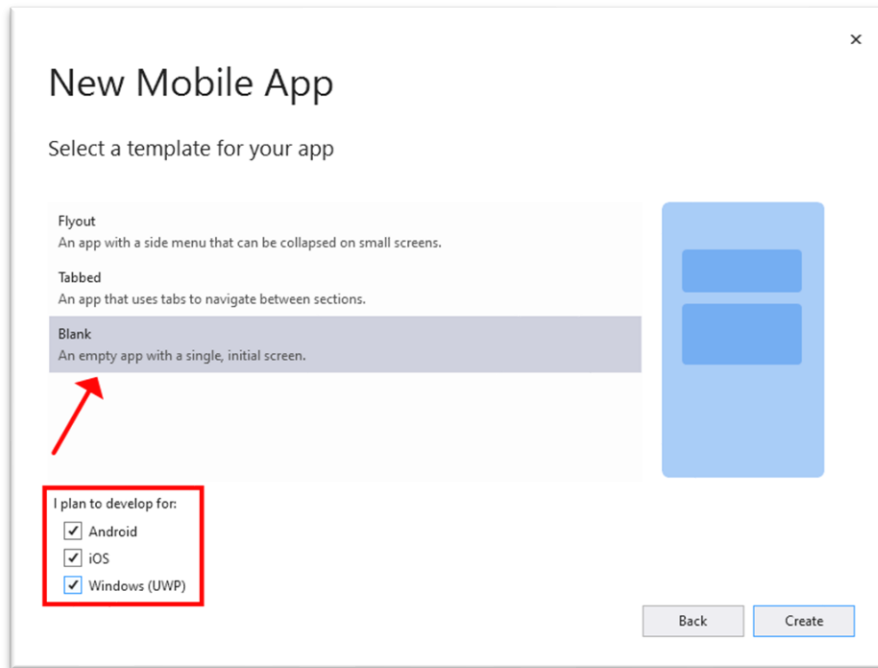
We will assume you have completed the previous labs, so you have some code to copy over to this lab.

Exercise 1: Create the project

Create a **new project** called “**StarWarsXF**”. The project type is “Mobile App (Xamarin.Forms)”:



Select the **Blank** application template and select the Android/iOS (optional)/Windows (UWP) platforms.



Build and run your app, it should present you a “Welcome to Xamarin.Forms” screen.

Exercise 2: Reuse Domain, Data and test projects

Copy the following folders from the StarWars Android lab to the “StarWarsXF” folder:



For each folder:

- Right-click on the solution in VS and choose “Add -> Existing project...”.
- Select the .csproj file in the folder

Make sure the solution compiles.

Exercise 3: Create a migration to support rating a movie

Add a migration called “RatingToMovie” which adds the following property to the Movie class:

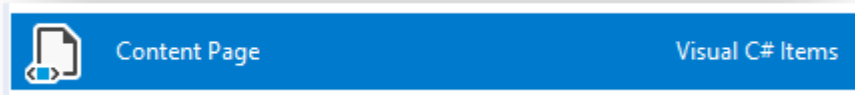
```
C#  
[JsonIgnore]  
public float Rating { get; set; }
```

Create a new “DummyConsole”-project to actually create the migration. You will need to add the necessary dependencies yourself.

This new property will allow you to store the rating for each movie.

Exercise 4: Build the main UI

First, delete MainPage.xaml and MainPage.xaml.cs from StarWarsXF and create a new ‘ContentPage’ called **MainView**.



Change App.xaml.cs to instantiate this page.

Now manually change the ContentPage to a MasterDetailPage.

More info:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/master-detail-page>

Replace the XAML code from MainView to:

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:StarWarsXF"
    x:Class="StarWarsXF.MainView">
    <MasterDetailPage.Master>
        <local:MovieListView x:Name="MyMasterPage" />
    </MasterDetailPage.Master>
    <MasterDetailPage.Detail>
        <local:EmptyMovieDetailsView />
    </MasterDetailPage.Detail>
</MasterDetailPage>
```

The code behind has to be changed as follows:

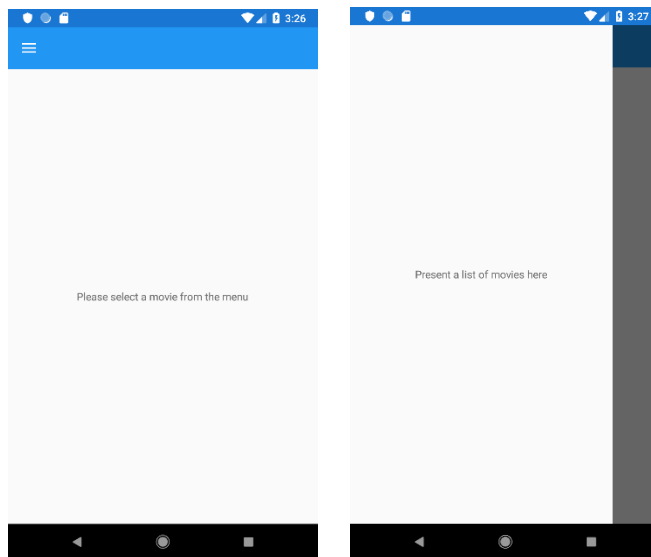
C#

```
public partial class MainView : MasterDetailPage
```

The Master page from this code will be in a new page called MovieListView and uses a NavigationPage as Detail. The first Page on the stack of the NavigationPage will be a new ContentPage called EmptyMovieDetailsView.

Now create these two new pages: `MovieListView` and `EmptyMovieDetailsView`. Just use some text into these two `ContentPages`. Provide a Title for the `MovieListView` page ("Star Wars").

Compile and run your app. This will now look something like this:



Exercise 5: Migrate the database

First you have to migrate your database inside your app. Insert the code below inside the constructor after the `InitializeComponent` method of your application (`App.xaml.cs`):

```
C#
string dbName =
Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder.LocalApplicationData), "starwars.db");
StarWarsContextFactory.ConnectionString = $"Data Source = {dbName}";

using (StarWarsContext context = StarWarsContextFactory.Create())
{
    context.Database.Migrate();
};
```

Class `StarWarsContextFactory` is a helper class to easily obtain the `StarWarsContext` from several places in the application. Create this class in `StarWarsUniverse.Data`:

```
C#
public class StarWarsContextFactory
{
    public static string ConnectionString;

    public static StarWarsContext Create()
    {
        {
```

```

        var options = new DbContextOptionsBuilder<StarWarsContext>()
            .UseSqlite(ConnectionString)
            .Options;
        return new StarWarsContext(options);
    }
}

```

Note

Because we don't use a proper layered architecture, we instantiated this `StarWarsContextFactory` directly in the UI code. In the next lab, we use proper dependency injection.

You will also have to make the necessary references into the `StarWarsXF` project and install the `Microsoft.EntityFrameworkCore.Sqlite` package into this project.

Exercise 6: Movie list

Now we will load all movies and present them in the `MovieListView` page. First add a `ListView` component with the name `SWMovieListView` (see code below).

XAML

```

<StackLayout VerticalOptions="FillAndExpand">
    <ListView x:Name="SWMoviesListView">
        <ListView.Header>
            <Grid BackgroundColor="#03A9F4">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="10"/>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="10"/>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="30"/>
                    <RowDefinition Height="80"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="10"/>
                </Grid.RowDefinitions>
                <Label
                    Grid.Column="1"
                    Grid.Row="2"
                    Text="Star Wars"
                    Style="{DynamicResource SubtitleStyle}"/>
            </Grid>
        </ListView.Header>
        <ListView.ItemTemplate>
            <DataTemplate>
                <!-- Add your code here -->
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>

```



```
</StackLayout>
```

Inside the DataTemplate tag you will have to add code to show the episode numbers. Use a custom ViewCell that has a StackPanel with 2 labels and some margin (10 left and right, 2 above and below).

In MovieListView.xaml.cs, add the code to create a context, load movies from it and set the ItemSource of the ListView. Expose this ListView as a public property so that you can add an event handler to it (see further).

C#

```
public partial class MovieListView : ContentPage
{
    public ListView MyListView { get; private set; }

    public MovieListView ()
    {
        InitializeComponent ();

        // Add your code here

        // Use StarWarsContextFactory to create a context
        // Create a MovieDBRepository
        // Get all movies and set the ItemSource property

        ...

        MyListView = SWMoviesListView;
    }
}
```

Compile and run your app. An episode list should appear on the master page on the left hand side.

Exercise 7: Movie details

When the user selects an episode from the list, a page with movie details is presented. Therefore we need to handle the ItemSelected event from the ListView. We handle this event in MainView.xaml.cs because we need to access the Detail property of the MasterPage. This is possible because we exposed the ListView as a public property.

Add the following code to MainView.xaml.cs:

C#

```
public MainView()
{
    InitializeComponent();
}
```

```

        MyMasterPage.MyListView.ItemSelected += MyListView_ItemSelected;
    }

    private void MyListView_ItemSelected(object sender,
        SelectedItemChangedEventArgs e)
    {
        if (movie != null)
        var movie = (Movie) e.SelectedItem;
        {
            var page = new MovieDetailsView {Title = movie.Title};
            page.FillMovieDetails(movie);

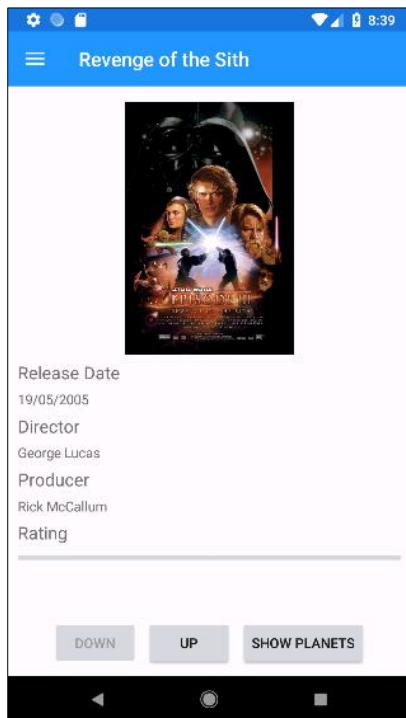
            Detail = new NavigationPage(page);

            IsPresented = false;
        }
    }
}

```

IsPresented collapses the master page once you have selected the item.

This code does not compile (yet) because there is no page to present movie details. Create this class now and add a method FillMovieDetails to pass in a movie. Try to create a page that looks like this:



In order to show images, you use an Image control and you set the Source property with the image file name. This name can be calculated from the movie title. That actual images should be copied to your StarWarsXF.Android project in folder Resources/drawable.

Note

More info on working with images can be found here:

<https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/images>

Try to run your application. You will notice a delay when images are loaded. Therefore you could use the FFLoading plugin so you could use CachedImage, a direct replacement for Image. This increases performance significantly.

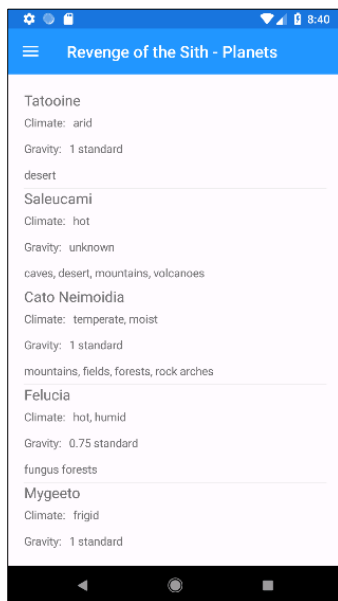
FFImageLoading info

<https://github.com/luberda-molinet/FFImageLoading/wiki/Xamarin.Forms-API>

Also note the Up/Down buttons. These buttons set a rating between 0 and 10 to the movie. Each tap alters the rating with 0.5 unit. The progressbar reflects this rating (you will need to convert the rating to a value suitable for displaying into the progressbar). The buttons get enabled/disabled according to the current rating.

Exercise 8: Planet details

From the movie details, you get to the planet details.



Try to implement this without extra tips.

Exercise 9: Run on another platform

Change the startup project (e.g. StarWarsXF.UWP) and run the application. Without any extra code, your app should work!

Note


You will have to copy the images into each target platform according to the documentation referenced above.

StarWarsXF.UWP

Revenge of the Sith

Star Wars

[Episode 1](#)
[Episode 2](#)
[Episode 3](#)
[Episode 4](#)
[Episode 5](#)
[Episode 6](#)
[Episode 7](#)



Release Date

19/05/2005

Director

George Lucas

Producer

Rick McCallum

Rating

[Down](#) [Up](#) [Show Planets](#)