Hands-on lab

Lab: Xamarin Intro

September 2020

In this lab, you will build two apps. First you create a simple Xamarin.Android app called Phoneword. This app serves as a simple test to make sure your development environment is correctly set up. Second, you create a simple app built with .NET Standard libraries.

Exercise 1: Phoneword

Follow this tutorial:

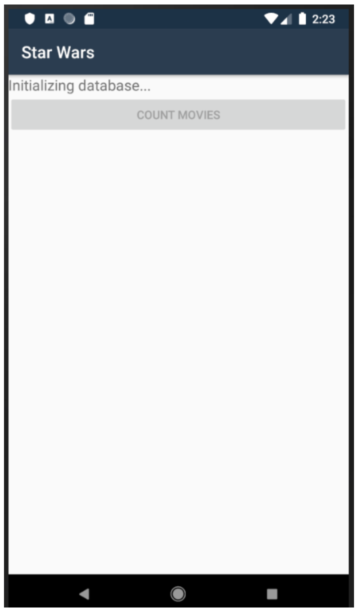
https://docs.microsoft.com/xamarin/android/get-started/hello-android/hello-android-quickstart

Note: when you perform certain actions for the first time in your project, like building, deploying, starting an emulator etc., things are slow. Subsequent actions will be faster.

Exercise 2: StarWarsAndroidApp

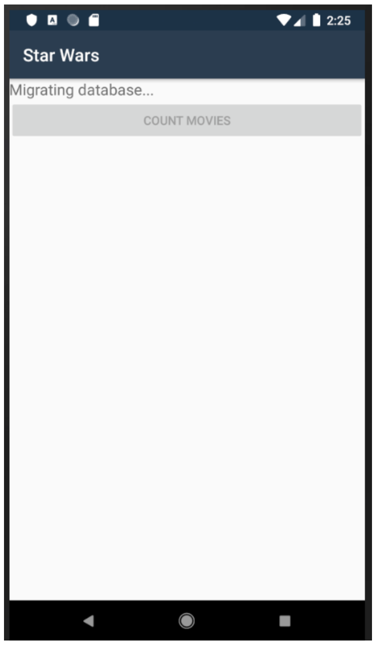
Overview

You will build the following Xamarin.Android application. The application presents the following start screen:

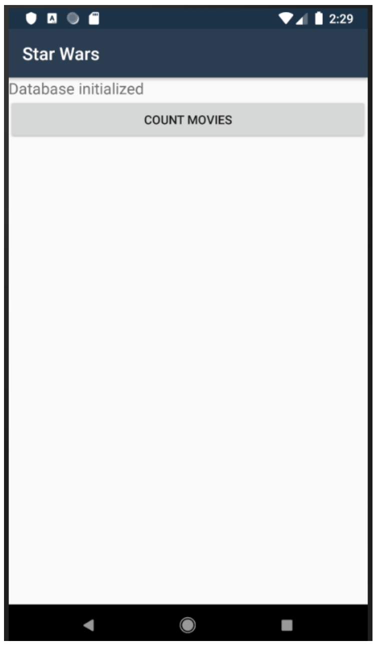


When the activity is created a “StarwarsContext” is created and if needed the database will be migrated.

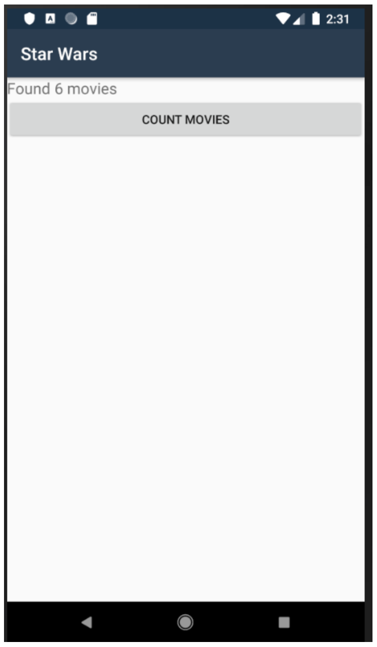
When this is happening (it could take a while) the “Count Movies” button is disabled.



When the “StarWarsContext” is instantiated (and the migration is executed), the button is enabled.



When the user taps the “Count Movies”-button, the app displays the number of movies in the database.



When you complete this exercise you will have learned the following:

- Create a simple native Android application using Xamarin.Android
- Use EF Core with SQLite in a mobile application
- Use asynchronous programming to do long running operations (initialize database) without freezing the UI

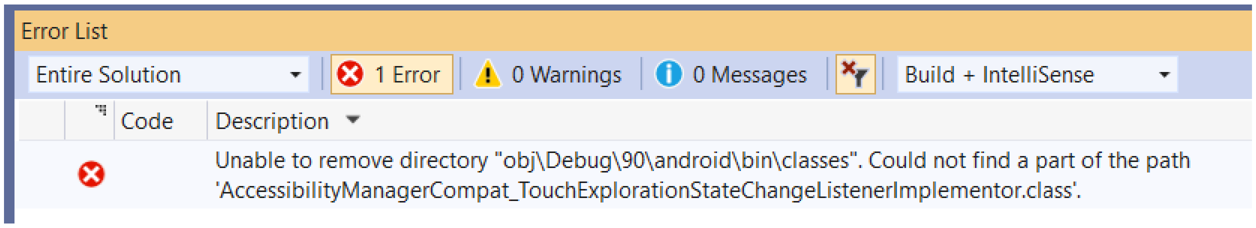
Step 1 – Create a solution with .NET Standard libraries

Start a new Blank Solution and call it “StarWarsAndroidApp”.

Tip: watch out for long path names!

Xamarin is very unforgiving about long path names. So choose your paths wise and don’t store your solution in a folder that is too long, like: C:\users\john\EnterpriseAndMobile\exercises

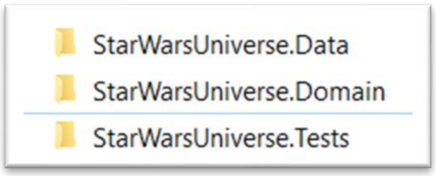
Chances are that you run into the following error:



```
Error List
Entire Solution | 1 Error | 0 Warnings | 0 Messages | Build + IntelliSense
Code | Description
Unable to remove directory "obj\Debug\90\android\bin\classes". Could not find a part of the path 'AccessibilityManagerCompat_TouchExplorationStateChangeListenerImplementor.class'.
```

Your safest bet is always to store your project into a simple folder, not too deeply nested, e. g. : C:\Workspace.

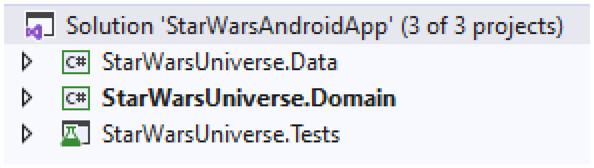
Copy the following folders from the EF Core lab to the “StarWarsAndroidApp” folder:



```
StarWarsUniverse.Data
StarWarsUniverse.Domain
StarWarsUniverse.Tests
```

For each folder:

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



```
Solution 'StarWarsAndroidApp' (3 of 3 projects)
  StarWarsUniverse.Data
  StarWarsUniverse.Domain
  StarWarsUniverse.Tests
```

Next:

- Remove the “Migrations” folder from the data project (as we will have to recreate them from scratch)
- Remove the “Microsoft.EntityFrameworkCore.SqlServer” NuGet package. On our mobile device we will use a SQLLite database, so the provider to use is in the package “Microsoft.EntityFrameworkCore.Sqlite”. Add this package to the Data project (it should be already in the Test project).

- Change the OnConfiguring-method in StarWarsContext to point to a SQLite database. Note: this is not the database that will be used on the actual device, but it is used to create the migration code.

  C#

  ```
  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
  {
      if (!optionsBuilder.IsConfigured) // this one is used to create migrations
      {
          optionsBuilder.UseSqlite("Data Source = starwars.db");
      }
  }
  ```

Now all projects should build without errors.

Step 2 – Create a migration

Add a new Console App (.NET Core) to the solution and call it "DummyConsole". You will only need this project to create the migration (just as in the previous lab).

- Add the "Microsoft.EntityFrameworkCore.Tools" NuGet package to the console project
- Make the console project the startup project
- Create the necessary references between projects
- Create a migration called "Initial"

> Tip: use add-migration Initial -verbose
>
> This will output useful information to correct any mistakes.

If everything went well, you will see a Migrations folder in the StarWars.Data-project. We will execute this migration from within the Android app to create a SQLite database from scratch.

Step 3 – Build the Android app

Add a new Android App (Xamarin) project to the solution. Call it StarWarsAndroidApp. Select a "Blank App" template and choose Android 7.1 (Nougat) as the minimal Android version. Make this project the Startup project.

> Tip: now is a good time to build and run your app, before you make any references to other projects. The first time, this can take quite long (several minutes).

Now create the user interface (look at the Phoneword exercise) and try to complete the application.

Create a “_context” field of type “StarWarsContext” in the activity. In the “OnCreate” method you can instantiate this field. Do this in an asynchronous method (returns “Task”) and await the result. After that you can enable the button.

How to instantiate (and migrate) the StarWarsContext?

To run something in a separate thread, you can use the “Task.Factory.StartNew” method which takes a delegate (Action) and returns a Task.

Now you need to create a connection string that points to the database. On a laptop or server, there is a filesystem that the user/developer can access to create and store files. On a mobile device, access is very limited. However, Xamarin (by means of .NET) provides a nice way to find the location where the user can access files. With the statement *System.Environment.GetFolderPath(System.Environment.SpecialFolder.LocalApplicationData)* you get the path to the application data folder (which the app can access).

With that path you can create a connection string that you can use to create the “DbContextOptions” that the “StarWarsContext” needs.

To migrate the database you can use the “_context.Database.Migrate” method.

C#

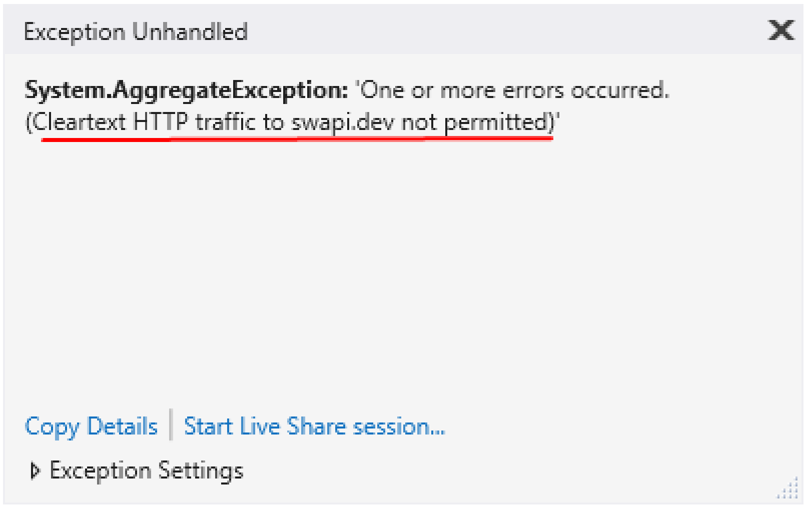
```
        private Task InitializeDbContextAsync()
        {
            _infoTextView.Text = "Initializing database...";
            return Task.Factory.StartNew(() =>
            {
                string dbName =
Path.Combine(System.Environment.GetFolderPath(System.Environment.SpecialFolder
.LocalApplicationData), "starwars.db");

                string connectionString = $"Data Source = {dbName}";

                var dbContextOptions = //TODO: use DbContextOptionsBuilder

                _context = new StarWarsContext(dbContextOptions);

                _infoTextView.Text = "Migrating database...";
                _context.Database.Migrate();
                _infoTextView.Text = "Database initialized";
            });
        }
```

Using the StarWarsContext will be quite slow. The reason for this is that the StarWars Api is called to retrieve the movies and planets each time the database model is created in memory (OnModelCreating).

So be patient after you start the application. (Luckily this is just an academic example.)

Tip: fix errors regarding “cleartext http”

If you run your app on Android 9 (or higher), you may get the following error:


Exception Unhandled

System.AggregateException: 'One or more errors occurred. (Cleartext HTTP traffic to swapi.dev not permitted)'

When you fetch all your planets, subsequent ResultPages will direct you to http-addresses instead of https. While the back-end will redirect you, Android 9 (and up) does not allow an application to make a http call (because the data is transmitted in clear text).

You as an application developer can decide to circumvent this by following the steps in this article:

https://devblogs.microsoft.com/xamarin/cleartext-http-android-network-security/

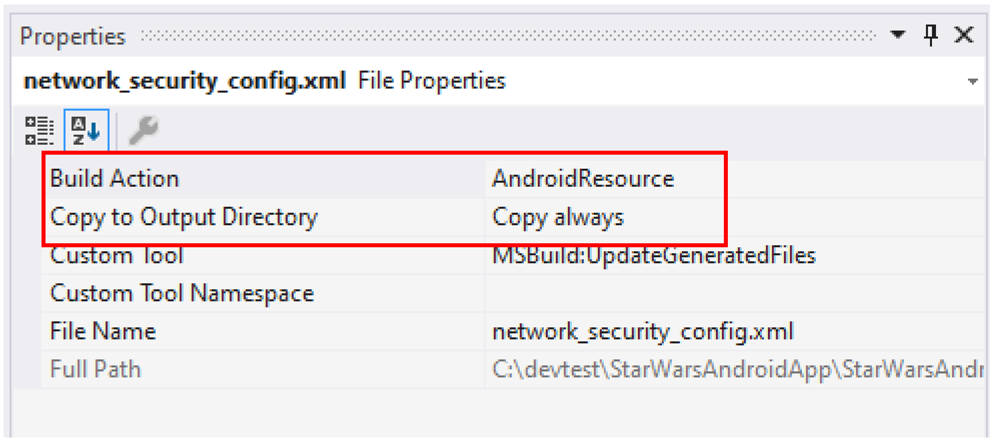
It boils down to the following steps:

- Inside the Resources folder of the StarWarsAndroidApp project, create a folder “xml”
- Create a file inside this folder, called network_security_config.xml:

network_security_config.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="true">
    <domain includeSubdomains="true">swapi.dev</domain>
  </domain-config>
</network-security-config>
```

- Modify the properties of this file:



Properties

network_security_config.xml File Properties

Build Action	AndroidResource
Copy to Output Directory	Copy always
Custom Tool	MSBuild:UpdateGeneratedFiles
Custom Tool Namespace	
File Name	network_security_config.xml
Full Path	C:\devtest\StarWarsAndroidApp\StarWarsAndr

- Now open the file AndroidManifest.xml (in the Properties of the Android project), and add a reference to the file above:

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:versionCode="1" android:versionName="1.0"
package="com.companyname.starwarsandroidapp" android:installLocation="auto">
   <uses-sdk android:minSdkVersion="25" android:targetSdkVersion="28" />
   <application android:networkSecurityConfig="@xml/network_security_config"
                android:allowBackup="true"
                android:icon="@mipmap/ic_launcher"
                android:label="@string/app_name"
                android:roundIcon="@mipmap/ic_launcher_round"
                android:supportsRtl="true"
                android:theme="@style/AppTheme">
  </application>
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
</manifest>
```

Now rebuild and run your app!