Practicumopgaven

Inhoudsopgave

Week 1

Beschrijving

De eerste week aan opdrachten. Om deze compleet te hebben dienen opdrachten A en B te worden afgewerkt.

Doel

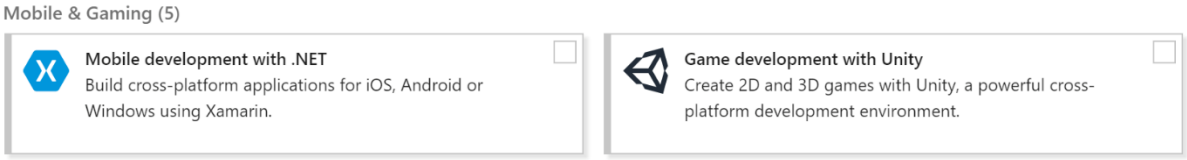
Het doel van deze opdracht is om te leren:

- Hoe Xamarin Forms werkt
- Hoe je een Xamarin Forms applicatie maakt
- Hoe je een basic lay-out in XAML kunt schrijven
- Hoe je basic backend code in C# voor Xamarin Forms kan schrijven
- Hoe je lay-out elementen kunt koppelen aan de backend

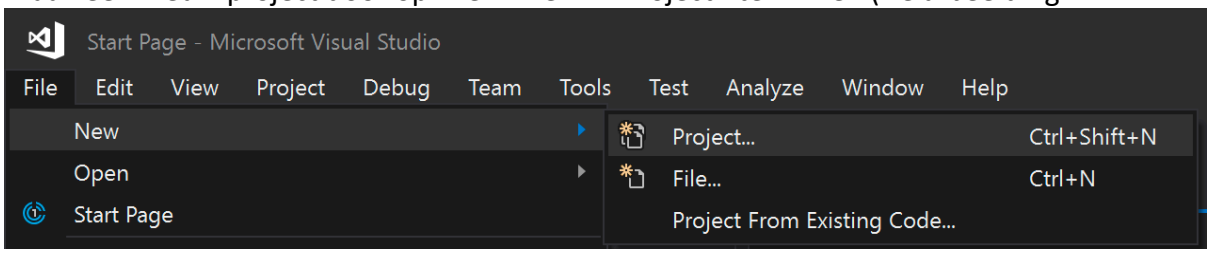
Opdracht A – Installatie

Loop de volgende stappen door:

1. Open de Visual Studio Installer om bepaalde componenten te installeren die nodig zijn voor Xamarin (als je dit nog niet gedaan hebt). Selecteer de “Mobile development with .NET” module (zie afbeelding).



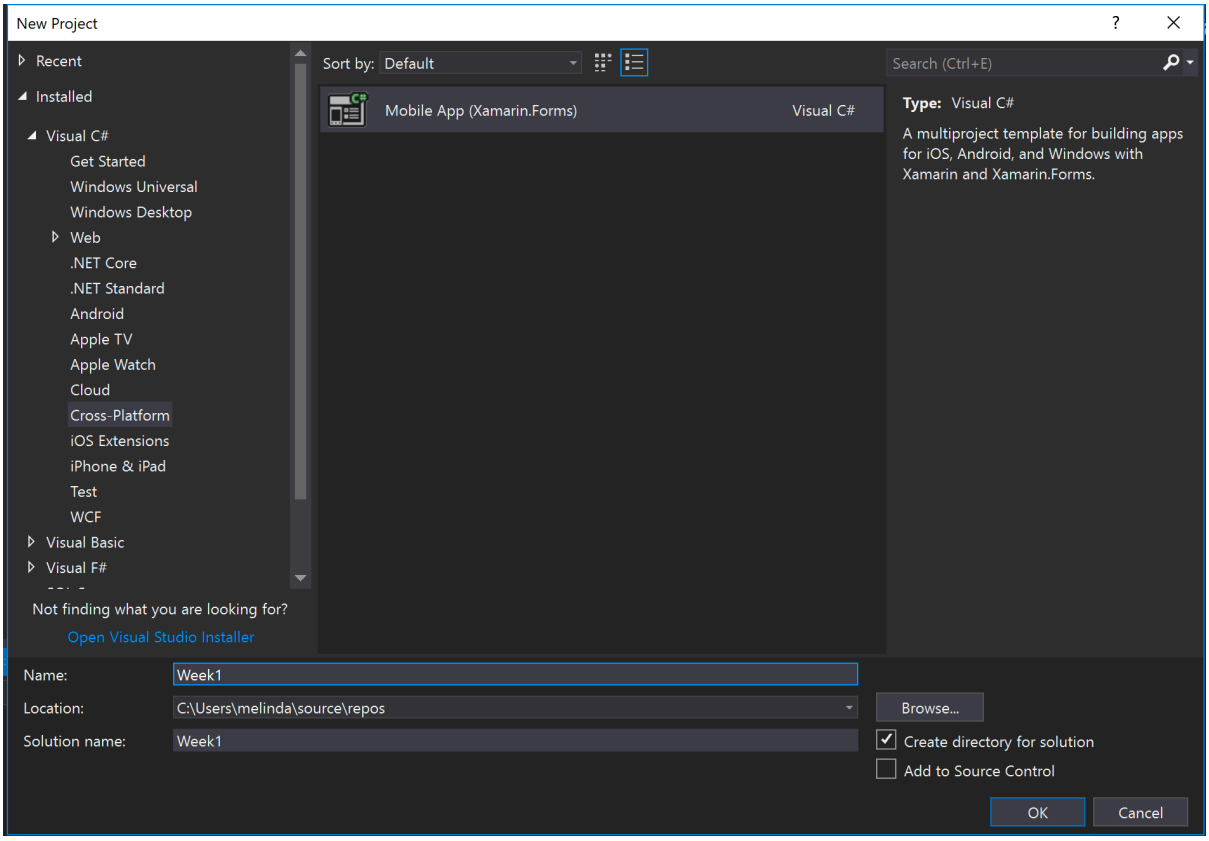
2. Maak in je file systeem een map waarin je je huiswerk wilt opslaan (voor overzicht). In deze handleiding gaan we er vanuit dat deze map “Apps Programmeren” heet, maar dit mag je geheel zelf bepalen.
3. Maak een nieuw project door op File -> New -> Project.. te klikken (zie afbeelding.



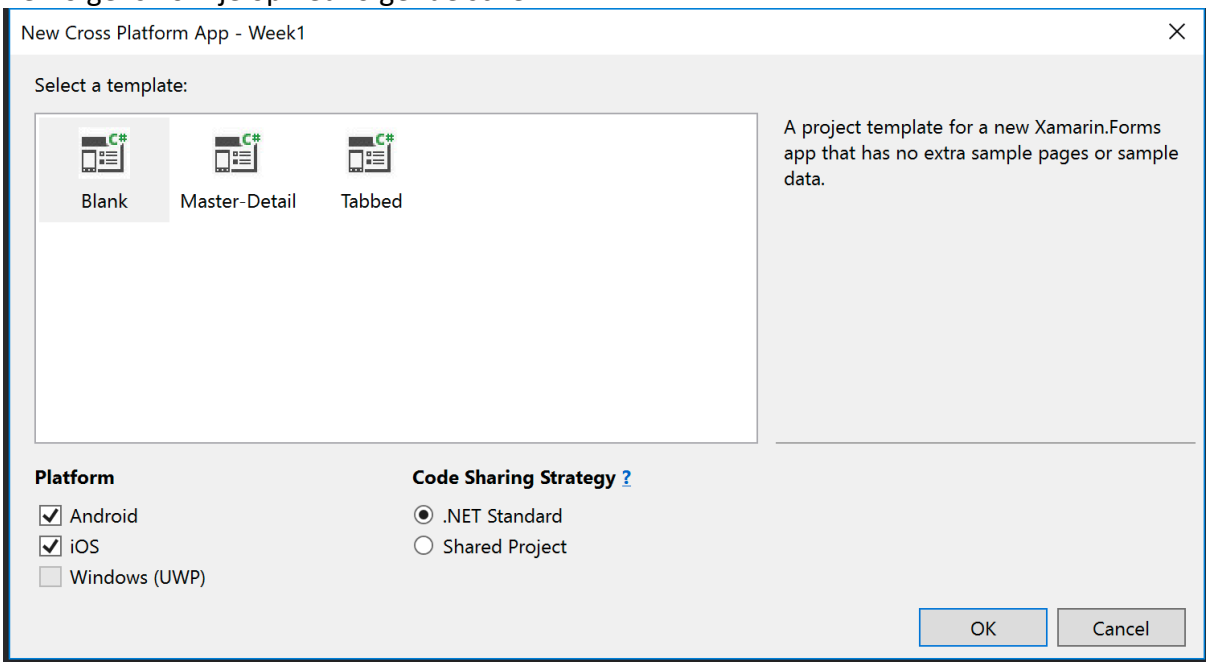
Je krijgt nu het New Project scherm te zien. Zorg dat je volgende dingen doet:

   a. Selecteer “Cross Platform”
   b. Selecteer “Mobile App (Xamarin Forms) Visual C#”
   c. Noem het project Week1
   d. Selecteer bij location de map aangemaakt bij stap 2
   e. Zet een vinkje bij “Create directory for solution”

   Controleer voordat je op OK klikt of het New Project scherm nu overeen komt met de onderstaande afbeelding.



4. Vervolgens kom je op het volgende scherm

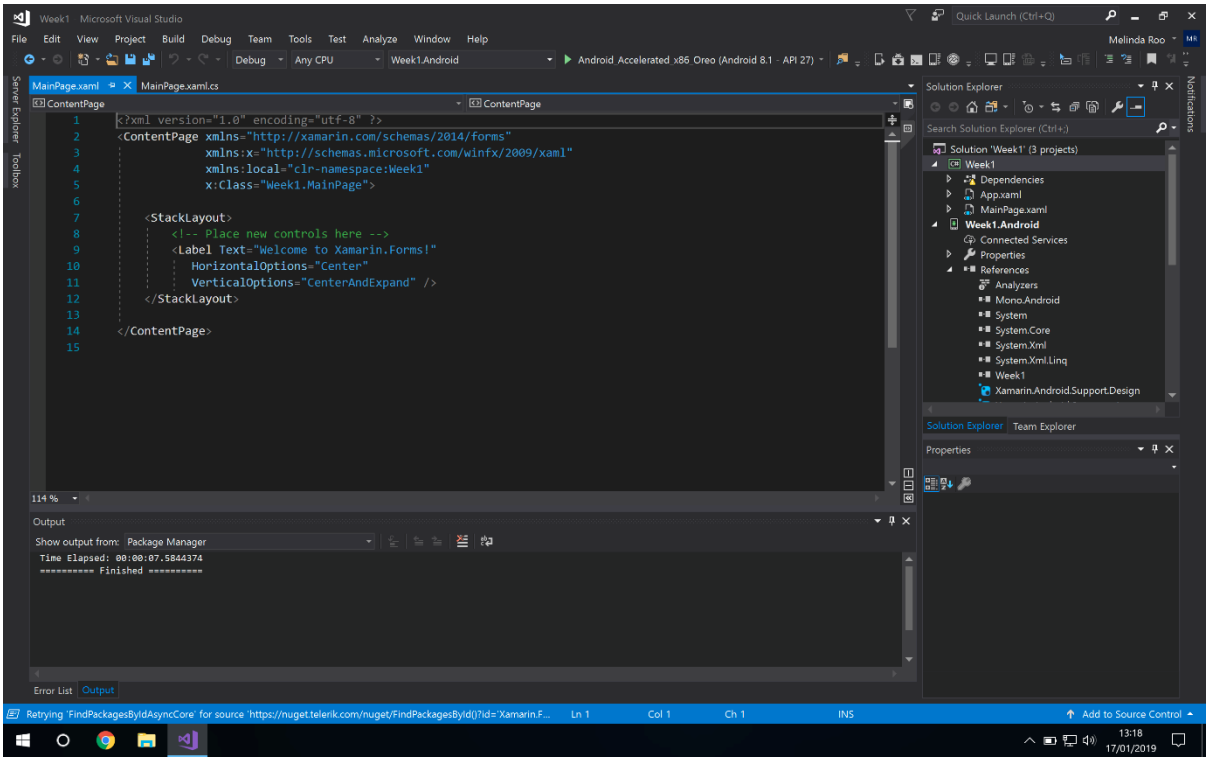


Neem de volgende stappen

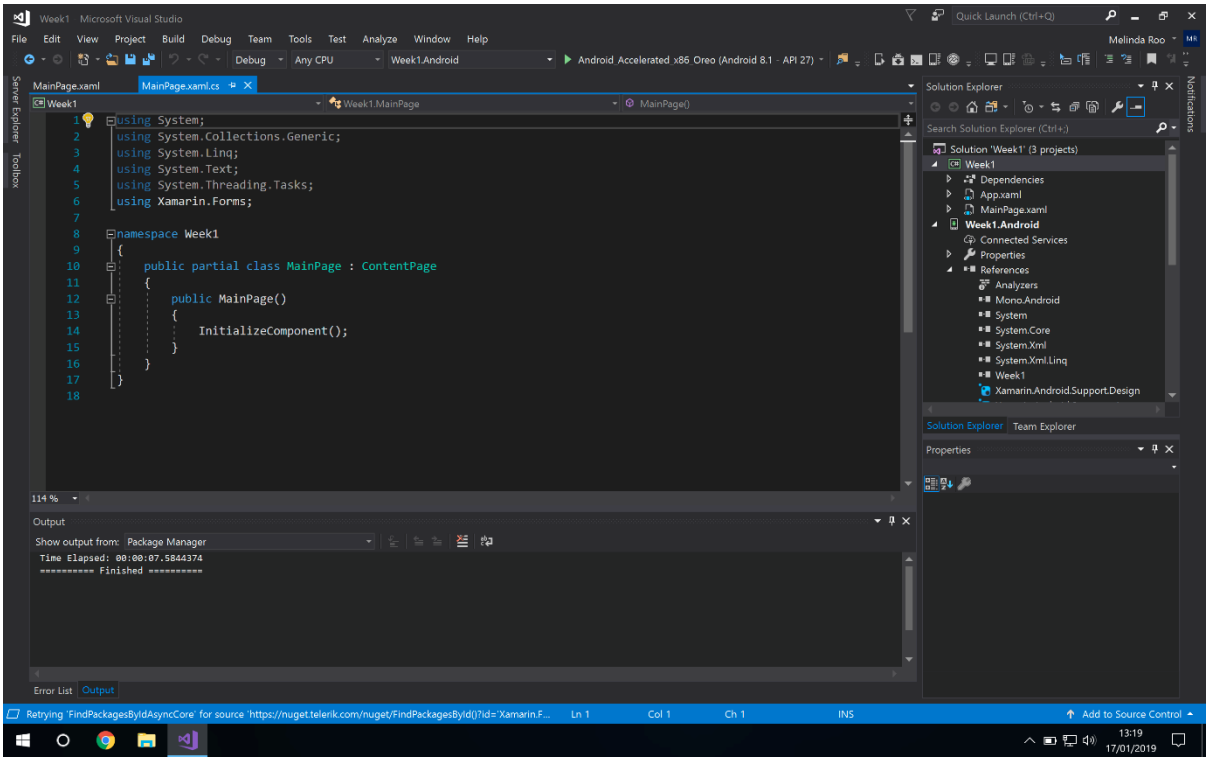
a. Selecteer: “Blank”

Blank = Lege app, Master-Detail = Navigatie, Tabbed = Tabjes navigatie

Klik op OK en je krijgt het onderstaande scherm.

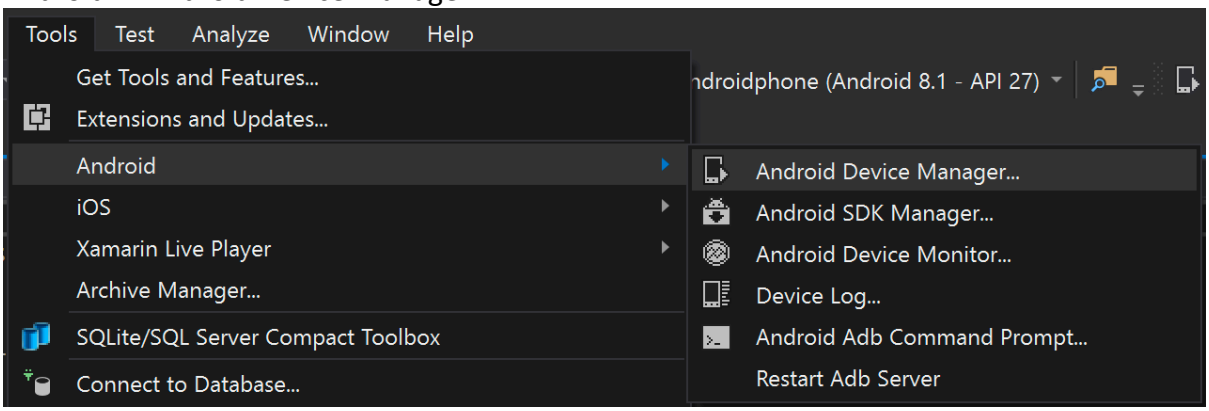


De pagina die je nu voor je hebt, is de lay-out pagina. Hierin wordt de lay-out van jouw eerste pagina van de app geschreven, dit is de “**MainPage.xaml**”.



De pagina die erbij geopend is, is de pagina waarin de backend code van jouw eerste pagina van de app geschreven wordt, dit is de “**MainPage.xaml.cs**”.

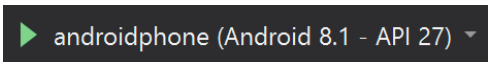
5. De volgende stap is het aanmaken van een virtueel device om de app op te debuggen (dit mag je ook op je eigen Android of iOS device doen). Klik op Tools -> Android -> Android Device Manager.



6. Neem de volgende stappen:
    a. Klik op “+New”
    b. Selecter bij “Base Device”: **Pixel**

Klik op “Create” om het device aan te maken. De reden voor Pixel is omdat deze snel is.

Nu er een device is aangemaakt, kan de standaard code worden uitgevoerd op het test device. Voer de code uit door op AndroidPhone te klikken.



Hieronder staat het resultaat van het starten van de lege applicatie op een Android emulator.

Opdracht B – Hello World!

Breid de “Week1” applicatie uit Opdracht A uit. Alle gemaakte code komt in deze applicatie. Hiervoor moet er soms een variabele en/of methode worden toegevoegd.

XAML

1. Verwijder alle code uit de <StackLayout></StackLayout>
2. Geef de StackLayout een margin van 16.
3. Centreer de StackLayout verticaal.
4. Maak, in de nu lege “StackLayout”, een nieuwe <Label></Label> aan. De Label moet de tekst “Naam” bevatten.
5. Maak daaronder een <Entry></Entry> aan, waarin een stukje tekst kan worden ingevoerd. Geef deze Entry de naam “NameEntry”.
6. Maak daar weer onder een <Button></Button> aan waarop de tekst “Toevoegen” staat. Geef deze Button de naam “AddButton”.
7. Maak tot slot nog een <Button></Button> aan waarop de tekst “Toon lijst” staat. Geef deze Button de naam “ShowButton”.
8. Beide Buttons moeten horizontaal gecentreerd worden.
9. Daarnaast moeten de Buttons een vaste grootte hebben van 128.
10. Er moet ook op beide Buttons geklikt kunnen worden. Dit gebeurd aan de hand van het “Clicked” attribuut. Geef de klik methodes een toepasselijke naam (bijv. AddClicked en ShowClicked).
11. **BONUS**: Pas de lay-out naar eigen smaak aan (bijv. verander de achtergrondkleur, stijl van de entry).
12. Er moet een tweede pagina komen in de applicatie die de lijst met namen gaan tonen. Maak deze tweede pagina en vul de StackLayout met een <ListView></ListView>. Noem deze pagina “ListPage”.

Als je alles hebt gedaan zoals in het stappenplan staat, moet de applicatie er uit zien als onderstaande afbeelding:

Backend

1. Allereerst hebben we een lijst nodig waarin de namen, die worden ingevoerd, kunnen worden opgeslagen. Maak deze lijst bovenaan de klasse aan. Het initialiseren van deze lijst gebeurd in de constructor. Geef hem een toepasselijke naam.
2. Maak de twee methoden voor de klik functionaliteit aan. Voorbeeld:
   public void AddClicked(object sender, ClickedEventArgs e)
3. In de methode voor het toevoegen van de ingevulde naam moet de naam aan de lijst worden toegevoegd.
4. Als we de applicatie nu debuggen en we vullen geen naam in, wordt er bij een klik op de Button een lege string toegevoegd. Dit willen we niet. Zorg ervoor dat er geen lege string kan worden toegevoegd aan de lijst.
5. Nu willen we de applicatie uitbreiden met een tweede pagina om de lijst te tonen. Voordat we dit doen, openen we “App.xaml.cs”. Hierin staat “MainPage = new MainPage();”. Verander dit in “MainPage = new NavigationPage(new MainPage());”.
6. Nu krijgen we, als we gaan debuggen, een menu balk bovenaan in onze applicatie. Deze willen we niet hebben. Ga terug naar “MainPage.xaml.cs”. Om de menubalk weg te halen, voegen we “NavigationPage.SetHasNavigationBar(this, false);” toe aan de constructor.
7. De ListPage mag de menubalk wel houden, omdat we terug moeten kunnen navigeren naar de vorige pagina, waar we de lijst vullen.
8. In de code van de ListPage moet de constructor worden aangepast, zodat er een lijst kan worden meegegeven.
9. Vervolgens moet de ItemsSource van de ListView nog worden gevuld met de namen.
10. Tot slot moet de ShowClicked nog worden gevuld. Hierbij navigeren we naar de “ListPage” en geven we de _nameList mee.
11. **BONUS**: Breid deze code uit zodat er een bericht wordt teruggegeven aan de gebruiker dat je iets in de “NameEntry” moet hebben ingevuld.

Week 2

Beschrijving

De tweede week aan opdrachten. Inmiddels zijn jullie alweer twee weken bezig geweest met het programmeren voor de telefoon. Tijd om er een lokale database aan te koppelen.

Doel

Het doel van deze opdracht is om te leren:

- Hoe bindings werken in XAML
- Hoe een koppeling kan worden gemaakt met een lokale database
- Hoe er gecommuniceerd kan worden met een database in Xamarin Forms

Opdracht A – Bindings

In deze opdracht gaan jullie werken met bindings om elementen te koppelen aan de ListView. De volgende informatie is nodig bij deze opdracht:

> **The Avengers (2012)**
> Earth's mightiest heroes must come together and learn to fight as a team if they are going to stop the mischievous Loki and his alien army from enslaving humanity.
>
> **The Avengers: Age of Ultron (2015)**
> When Tony Stark and Bruce Banner try to jump-start a dormant peacekeeping program called Ultron, things go horribly wrong and it's up to Earth's mightiest heroes to stop the villainous Ultron from enacting his terrible plan.
>
> **The Avengers: Infinity War (2018)**
> The Avengers and their allies must be willing to sacrifice all in an attempt to defeat the powerful Thanos before his blitz of devastation and ruin puts an end to the universe.
>
> **The Avengers: End Game (2019)**
> After the devastating events of Avengers: Infinity War (2018), the universe is in ruins. With the help of remaining allies, the Avengers assemble once more in order to undo Thanos' actions and restore order to the universe.

1. Maak een nieuwe Xamarin Forms applicatie aan. Als je niet meer weet hoe dit moet, kun je terugkijken naar Opdracht A van vorige week. Noem dit project “Week2” en plaats deze in de folder die jij hebt aangemaakt voor al jouw Apps Programmeren huiswerkopgaven.

XAML

1. Verwijder alle XAML code die in de ContentPage staat.
2. Voeg een <ListView></ListView> toe en geef deze een toepasselijke naam (bijvoorbeeld: MovieList).
3. Omdat we de items willen koppelen aan een model hebben een ItemTemplate nodig. Dit doen we door binnen de ListView tag <ListView.ItemTemplate></ListView.ItemTemplate> toe te voegen.

4. Binnen de ItemTemplate tag voegen we <DataTemplate></DataTemplate> toe.
5. Om meerdere Labels te kunnen gebruiken voor een titel en omschrijving, moeten we binnen de DataTemplate de tag <ViewCell></ViewCell>.
6. Nu gaan we de lay-out bedenken voor onze ListView items. Deze items zullen dus een titel en een omschrijving hebben. Omdat de omschrijvingen best wel uitgebreid zijn, willen we dat de tekst daarvan wat kleiner is, zodat het allemaal in het ListView item past. Zorg er eerst voor dat de RowHeight van de ListView op 72 wordt gezet.
7. Voeg aan de ViewCell een <StackLayout></StackLayout> toe. Zet hierbij de Margin op 4, zodat de titel en omschrijving iets meer naar binnen staan straks.
8. In de StackLayout komen twee Labels. Voeg eerst een <Label></Label> voor de title toe. Zet hierbij de Binding op Title. Hiervan moet de grootte van het font 16 en de horizontale Margin 8 zijn.
9. Voeg een tweede <Label></Label> toe. Zet hierbij de Binding op Description. Hiervan moet de font grootte 8 en de horizontale Margin 8 zijn.
10. **BONUS**: Alleen maar tekst in de ListView item is natuurlijk saai. Dus als bonusopdracht gaan we een afbeelding toevoegen (probeer hiervoor vierkante afbeeldingen te vinden). Zet de afbeeldingen in een nieuwe folder genaamd 'Images'.
11. **BONUS**: De afbeeldingen moeten ook aan het Android project en iOS project worden toegevoegd. Voeg ze bij het Android project toe aan de folder drawable, die te vinden is in de resources folder. Voeg ze bij het iOS project toe aan de resources folder.
12. **BONUS**: Om de StackLayout in de ViewCell heen komt een tweede StackLayout. Zet hiervan de oriëntatie op horizontaal. Verplaats de Margin van de binnenste StackLayout naar deze.
13. **BONUS**: Zet de oriëntatie van de binnenste StackLayout op verticaal.
14. **BONUS**: Voeg een <Image></Image> toe aan de horizontale StackLayout. Zet de Source naar een Binding op ImageSource. Daarbij voegen we een vaste hoogte en breedte toe van 64.

Backend

1. Nu de lay-out geschreven is, gaan we beginnen aan de backend. Allereerst zullen we dan een model aanmaken om de data in onze ListView te krijgen (om deze te binden). Maak hiervoor een nieuwe klasse aan in je solution. Geef de klasse de naam 'Movie'
2. Verwijder de constructor uit de klasse, omdat het een model is.
3. Voeg een publieke string toe die Title heet.
4. Voeg een publieke string toe die Description heet.
5. Het model is nu compleet. De rest van de code die geschreven moet worden, komt in de MainPage.xaml.cs. Open deze file.
6. Voeg de volgende regel boven de constructor toe:
   ObservableCollection<Movie> movies = new ObservableCollection<Movie>();
7. Zet vervolgens de ItemSource van de ListView naar de net aangemaakte collectie van films.
8. Voeg de data toe aan de collectie van films (deze data staat bovenaan de opdracht, film titels en omschrijvingen).

9. **BONUS**: Voeg een publieke string boven de Title toe aan het model en noem deze ImageSource.
10. **BONUS**: Pas de collectie items aan door de ImageSource aan de films toe te voegen met de afbeeldingen die aan je project hebt gevoegd (“Images/<imageNaam>.<extensie>”).

Opdracht B – Database

Deze opdracht is een uitbreiding op opdracht A. Hierbij gaan we een koppeling maken naar een lokale database. Installeer hiervoor de ‘DB Browser voor SQLite’. In deze opdracht maken we gebruik van 1 enkele tabel. In het echt zal hier dan geen database voor worden gebruikt. Dit is alleen een oefening om te begrijpen hoe een connectie met een lokale database werkt!

1. Maak een database aan met de naam ‘MovieDatabase’ en voeg daar 1 tabel aan toe met de naam ‘Movie’. Deze tabel heeft 4 kolommen.
    a. MovieID, Integer, PrimaryKey, AutoIncrement, Unique, NotNull
    b. Title, Text, Unique, NotNull
    c. Description, Text, NotNull
    d. ImageSource, Text, NotNull
2. Voeg de data van opdracht A toe aan de net aangemaakte tabel ‘Movie’.
3. Zet de ‘MovieDatabase’ in de volgende mappen
    a. Week2
    b. De Assets folder van de Android applicatie
    c. De Resources folder van de iOS applicatie

XAML

1. Onder de ListView willen we opties hebben om het laatste element uit de lijst & database te verwijderen & toe te voegen of te updaten (1 Button). Begin eerst met een <StackLayout></StackLayout>. Deze StackLayout moet een horizontale oriëntatie hebben. Ook moet het horizontaal gecentreerd worden en een verticale Margin hebben van 16.
2. In de StackLayout moet tweemaal een <Button></Button> komen. De ene Button is voor het toevoegen & updaten van een item. De tweede Button is voor het verwijderen van een item. Geef ze toepasselijke tekst. Zorg daarnaast dat beide een klikfunctie krijgen.

Backend

1. Voeg aan de bestaande backend code twee methoden toe voor de klikfuncties van de zojuist aangemaakte Buttons.
2. Nu gaan we de model aanpassen. Voeg bovenaan de int toe voor het ‘MovieID’.
3. Om de koppeling te leggen met de database moet er een NuGet package aan alle projecten binnen de solution worden toegevoegd. Deze heet ‘sqlite-net-pcl’. **Let op dat je deze exact zo overneemt als je naar de NuGet package zoekt!**
4. We gaan weer terug naar de model. Boven het ‘MovieID’ moeten we nog zeggen dat het hier gaat om [PrimaryKey, AutoIncrement, Unique, NotNull]
5. Boven ‘Title’ komt [Unique, NotNull].
6. Boven ‘Description’ komt [NotNull].
7. Boven ‘ImageSource’ komt [NotNull].

8. Het model is nu klaar om gebruikt te worden in combinatie met de database. Omdat Android en iOS op verschillende manieren met de database communiceren qua code, moet er apart voor Android en iOS code worden geschreven. Dit gaan we doen met een DependencyService. Hiervoor maken we eerst een Interface aan die 'IDBInterface' heet.
9. In de Interface komt 1 methode SQLiteConnection CreateConnection();
10. Maak een nieuwe klasse in de Android folder aan die 'DatabaseService' heet. De onderstaande code moet hierin komen:

```
[assembly: Dependency(typeof(DatabaseService))]
namespace Week3.Droid
{
    public class DatabaseService : IDBInterface
    {
        public SQLiteConnection CreateConnection()
        {
            var sqliteFilename = "MovieDatabase.db";
            string documentsDirectoryPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            var path = Path.Combine(documentsDirectoryPath, sqliteFilename);

            if (!File.Exists(path))
            {
                using (var binaryReader = new BinaryReader(Android.App.Application.Context.Assets.Open(sqliteFilename)))
                {
                    using (var binaryWriter = new BinaryWriter(new FileStream(path, FileMode.Create)))
                    {
                        byte[] buffer = new byte[2048];
                        int length = 0;
                        while ((length = binaryReader.Read(buffer, 0, buffer.Length)) > 0)
                        {
                            binaryWriter.Write(buffer, 0, length);
                        }
                    }
                }
            }
            var conn = new SQLiteConnection(path, false);

            return conn;
        }
    }
}
```

11. Maak een nieuwe klasse in de iOS folder aan die 'DatabaseService' heet. De onderstaande code moet hierin komen:

```
[assembly: Dependency(typeof(DatabaseService))]
namespace Week3.iOS
{
    public class DatabaseService : IDBInterface
    {

        public SQLiteConnection CreateConnection()
        {
            var sqliteFilename = "MovieDatabase.db";

            string docFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            string libFolder = Path.Combine(docFolder, "..", "Library", "Databases");

            if (!Directory.Exists(libFolder))
            {
                Directory.CreateDirectory(libFolder);
            }
            string path = Path.Combine(libFolder, sqliteFilename);

            if (!File.Exists(path))
            {
                var existingDb = NSBundle.MainBundle.PathForResource("MovieDatabase", "db");
                File.Copy(existingDb, path);
            }

            var connection = new SQLiteConnection(path, false);
            return connection;
        }
    }
}
```

12. Nu kunnen we in de ‘Week2’ folder aan de slag. Maak hierin een klasse aan die ‘DatabaseManager’ heet. Deze klasse gaat SQL query’s loslaten op de database.
13. Maak de volgende methodes in de DatabaseManager:
    a. Constructor
    b. List<Movie> GetAllMovies()
    c. void AddOrUpdateMovie(string title, string description, string imagesource)
    d. void DeleteMovie(string title)
    e. bool DoesMovieExist(string title)
    f. int GetMovieID(string title)
14. Pas de code in de MainPage zo aan, dat in plaats van een hardcoded gevulde lijst de data uit de database wordt gehaald. Gebruik hiervoor de DatabaseManager.

Week 3

Beschrijving

De derde week aan opdrachten. Nu de basis vaardigheden er in zitten & een simpele database kan worden gebruikt in jullie applicatie, is het tijd voor de communicatie met de Arduino.

Doel

Het doel van deze opdracht is om te leren:

- Hoe classes werken in Xamarin Forms
- Hoe overerving werkt in Xamarin Forms
- Hoe je een gecompliceerde lay-out in XAML kunt schrijven
- Hoe er vanuit de applicatie een verbinding wordt gemaakt met een Arduino met Sockets

Opdracht A – Server & Client

In deze opdracht gaan jullie vanuit de mobiele applicatie verbinding maken met een server via sockets.

1. Maak een nieuwe Xamarin Forms applicatie aan. Als je niet meer weet hoe dit moet, kun je terugkijken naar Opdracht A van vorige week. Noem dit project "Week3Client" en plaats deze in de folder die jij hebt aangemaakt voor al jouw Apps Programmeren huiswerkopgaven.

XAML

1. Verwijder alle code binnen de <ContentPage></ContentPage> tag.
2. Maak een <StackLayout></StackLayout> aan.
3. Vanaf de server zal straks, na jouw aanroep, een temperatuur worden overgestuurd. Deze moet in de app visueel worden gemaakt. Maak een tag aan in de XAML code die ervoor zorgt dat straks de temperatuur wordt getoond.

Backend

1. De server is jullie gegeven. Het enige dat in de applicatie nog moet worden gemaakt, is de client code. Maak hiervoor een nieuwe klasse aan en noem deze 'Client'.
2. In de Client klasse moet een Socket worden aangemaakt. Dit kan in een statische methode, die ook het bericht verstuurd en ontvangt. Zorg ervoor dat IPAddress overeen komt met die van de server, evenals de poort.
3. Vervolgens moet de Client dus een bericht kunnen versturen naar de server, waarop die een bericht terug stuurt (de temperatuur).
4. Zorg ervoor dat de Client daarna een bericht kan ontvangen.
5. In de Client moeten foutmeldingen kunnen worden afgevangen. Doe dit met een try/catch.
6. Het ontvangen bericht moet op de pagina, de MainPage, getoond kunnen worden in de tag die is aangemaakt in de XAML.

Opdracht B – Read & Write

In deze opdracht moeten jullie naast het schrijven naar de Arduino ook gegevens gaan uitlezen van de Arduino. Hierbij gebruiken we opdracht 10a van Embedded Systems.

1. Maak een nieuwe Xamarin Forms applicatie aan. Als je niet meer weet hoe dit moet, kun je terugkijken naar Opdracht A van vorige week. Noem dit project “Week3” en plaats deze in de folder die jij hebt aangemaakt voor al jouw Apps Programmeren huiswerkopgaven.

XAML

1. Verwijder alle code binnen de <ContentPage></ContentPage> tag.
2. Maak een <StackLayout></StackLayout> aan.
3. Zet daarvoor op dezelfde regel een <Label></Label> met een titel.
4. Zorg ervoor dat deze nieuwe elementen dezelfde stijl krijgen als uit opdracht A.
5. Centreer beide Labels.

Backend

1. Het kan zijn dat er geen verbinding kan worden gemaakt met de Arduino, omdat deze nog niet is aangesloten bijvoorbeeld. Jouw applicatie zal dan een error teruggeven. Deze is te verhelpen met een:

   ```
   try { } catch () { }
   ```

   Voeg deze toe zodat de er geen foutmelding wordt gegeven als de applicatie eruit vliegt.

2. Zorg er tot slot voor dat de data die door de Arduino wordt verstuurd, gelezen kan worden in de applicatie en wordt getoond in de daarvoor aangemaakte Label.

Week 4

Beschrijving

De vierde week aan opdrachten. Het is de bedoeling om iets dieper op de database koppeling in te gaan en hoe fragments werken.

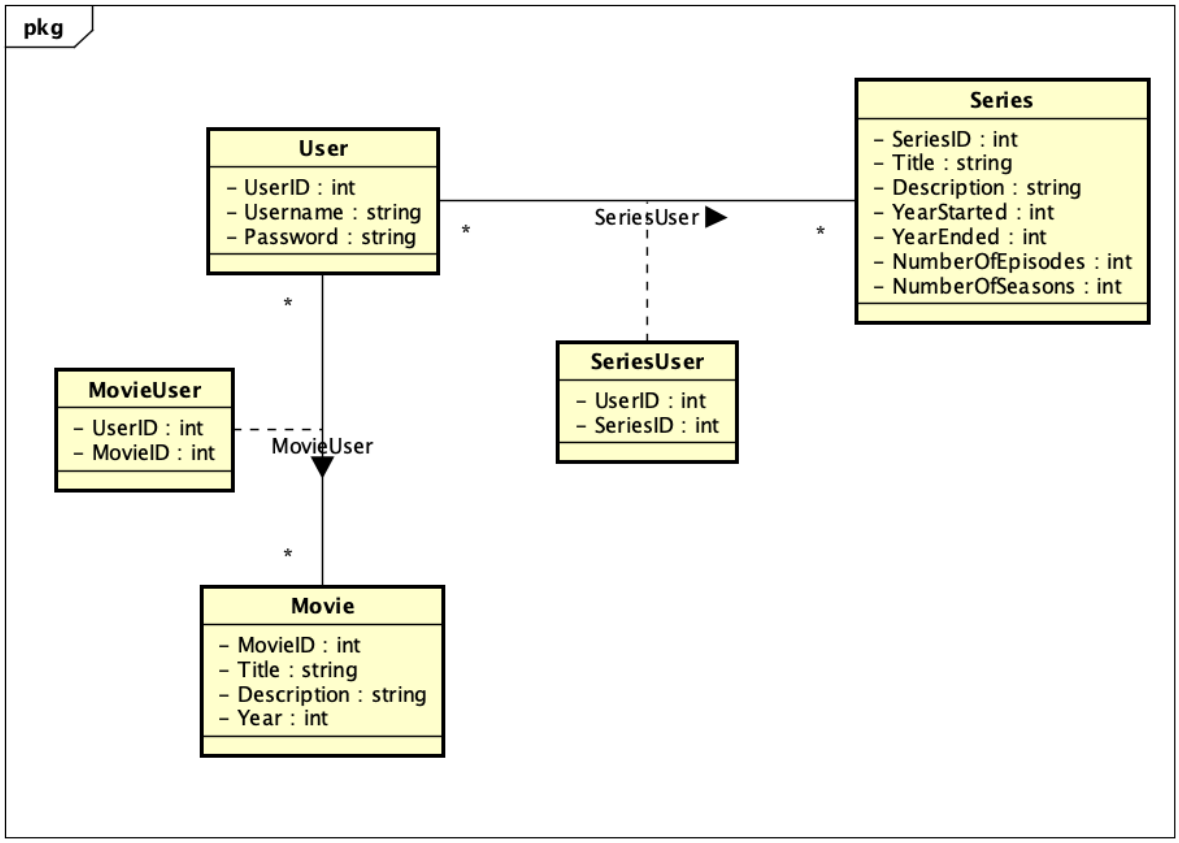
Doel

Het doel van deze opdracht is om te leren:

- Hoe alle aangeleerde technieken gecombineerd kunnen worden met elkaar
- Hoe fragments werken in Xamarin Forms
- Hoe een hamburger menu werkt aan de hand van fragments (Master Detail Page)
- Hoe de GPS werkt

Opdracht A – Gecompliceerde applicatie

Alle technieken die zijn aangeleerd in de afgelopen 3 weken gecombineerd in een opdracht.



De bovenstaande afbeelding is de structuur van de database die gebruikt gaat worden voor deze applicatie.

1. Maak een nieuwe Xamarin Forms applicatie aan. Als je niet meer weet hoe dit moet, kun je terugkijken naar Opdracht A van week 1. Noem dit project “Week4” en plaats deze in de folder die jij hebt aangemaakt voor al jouw Apps Programmeren huiswerkopgaven.
2. Maak hierna twee folders aan. Noem de eerste folder ‘Views’ en de tweede ‘Models’. Deze zijn nodig voor de verschillende views die de applicatie zal hebben en de models die gekoppeld zullen zijn met de database.

XAML

1. Er gaan drie views worden gebruikt in deze applicatie. De MainPage bestaat al. Verplaats deze naar de Views folder. Pas ook de Namespace van de MainPage aan naar Week4.Views. Dit moet op verschillende plekken: In de MainPage.xaml.cs, xmlns:local=”clr-namespace:Week4”, x:Class=”Week4.MainPage”.
2. Maak in de Views folder nog twee views aan. Noem deze views LoginPage en RegisterPage.

LoginPage

1. Zorg ervoor dat de ContentPage van de LoginPage een <StackLayout></StackLayout> bevat. Centreer de StackLayout verticaal en geeft het een Margin van 32.
2. In deze StackLayout moeten de volgende elementen komen:
    a. Een <Label></Label> met de tekst ‘Username’
    b. Een <Entry></Entry> met de naam ‘Username’
    c. Een <Label></Label> met de tekst ‘Password’
    d. Een <Entry></Entry> met de naam ‘Password’
    e. Een <Button></Button> met de naam ‘LoginButton’. Stijl deze Button naar eigen smaak en zorg ervoor dat deze horizontaal gecentreerd is. De functie van de Button als erop geklikt wordt moet ‘LoginClicked’ heten.
    f. Een <Label></Label> met de tekst ‘If you don't have an account yet, register here’. Centreer deze Label horizontaal en zorg ervoor dat de tekst is verkleind en op 1 regel past. De Label moet ook een TapGestureRecognizer hebben, zodat er op geklikt kan worden. De functie die wordt aangeroepen als erop wordt geklikt moet ‘RegisterClicked’ heten.

RegisterPage

1. Maak in de RegisterPage.xaml ook een <StackLayout></StackLayout> aan. Zorg ervoor dat deze ook verticaal gecentreerd wordt en een Margin krijgt van 32.
2. In deze StackLayout moeten de volgende elementen komen:
    a. Een <Label></Label> met de tekst ‘Username’
    b. Een <Entry></Entry> met de naam ‘Username’
    c. Een <Label></Label> met de tekst ‘Password’
    d. Een <Entry></Entry> met de naam ‘Password’
    e. Een <Label></Label> met de tekst ‘Repeat Password’
    f. Een <Entry></Entry> met de naam ‘RepeatPassword’
    g. Een <Button></Button> met de naam ‘RegisterButton’. Stijl deze Button naar eigen smaak, een beetje zoals de LoginButton op de LoginPage. Zorg ervoor dat de Button horizontaal gecentreerd is. Als erop wordt geklikt wordt de functie ‘RegisterClicked’ aangeroepen.

MainPage

1. Maak in de MainPage ook een <StackLayout></StackLayout> aan. Geef deze StackLayout een Margin van 8.
2. Bovenaan de pagina moet een <Label></Label> komen met de tekst ‘All movies and series’. De tekstgrootte moet 36 zijn.

3. Daaronder komt een tweede <Label></Label> met de tekst ‘Movies’. De tekstgrootte van deze Label moet Large zijn.
4. Voeg een <ListView></ListView> toe en geef deze een toepasselijke naam (bijvoorbeeld: MovieList).
5. Omdat we de items willen koppelen aan een model hebben een ItemTemplate nodig. Dit doen we door binnen de ListView tag <ListView.ItemTemplate></ListView.ItemTemplate> toe te voegen.
6. Binnen de ItemTemplate tag voegen we <DataTemplate></DataTemplate> toe.
7. Om meerdere Labels te kunnen gebruiken voor een titel en omschrijving, moeten we binnen de DataTemplate de tag <ViewCell></ViewCell>.
8. Nu gaan we de lay-out bedenken voor onze ListView items. Deze items zullen dus een titel en een omschrijving hebben. Omdat de omschrijvingen best wel uitgebreid zijn, willen we dat de tekst daarvan wat kleiner is, zodat het allemaal in het ListView item past. Zorg er eerst voor dat de RowHeight van de ListView op 64 wordt gezet.
9. Voeg aan de ViewCell een <StackLayout></StackLayout> toe. Zet hierbij de Margin op 4, zodat de titel en omschrijving iets meer naar binnen staan straks.
10. In de StackLayout komen twee Labels. Voeg eerst een <Label></Label> voor de title toe. Zet hierbij de Binding op Title. Hiervan moet de grootte van het font 16 en de horizontale Margin 8 zijn.
11. Voeg een tweede <Label></Label> toe. Zet hierbij de Binding op Description. Hiervan moet de font grootte 8 en de horizontale Margin 8 zijn.
12. We gaan dezelfde stappen 3 t/m 11 herhalen voor de lijst met series.
13. Zet van beide ListViews de VerticalOptions op ‘Start’.

Backend

1. Maak in de models folder een nieuw model aan met de naam ‘User’. In het User model moet het volgende komen:
    a. ‘UserID’ [PrimaryKey, AutoIncrement, Unique, NotNull]
    b. ‘Username’ [Unique, NotNull]
    c. ‘Password’ [NotNull]
2. Maak in de models folder een nieuw model aan met de naam ‘Movie’. In het Movie model moet het volgende komen:
    a. ‘MovieID’ [PrimaryKey, AutoIncrement, Unique, NotNull]
    b. ‘Title’ [Unique, NotNull]
    c. ‘Description’
    d. ‘Year’ [NotNull]
3. Maak in de models folder een nieuw model aan met de naam ‘Series’. In het Series model moet het volgende komen:
    a. ‘SeriesID’ [PrimaryKey, AutoIncrement, Unique, NotNull]
    b. ‘Title’ [Unique, NotNull]
    c. ‘Description’
    d. ‘YearStarted’ [NotNull]
    e. ‘YearEnded’
    f. ‘NumberOfEpisodes’ [NotNull]
    g. ‘NumberOfSeasons’ [NotNull]

4. Om de koppeling te leggen met de database moet er een NuGet package aan alle projecten binnen de solution worden toegevoegd. Deze heet ‘sqlite-net-pcl’. **Let op dat je deze exact zo overneemt als je naar de NuGet package zoekt!**
5. Het model is nu klaar om gebruikt te worden in combinatie met de database. Omdat Android en iOS op verschillende manieren met de database communiceren qua code, moet er apart voor Android en iOS code worden geschreven. Dit gaan we doen met een DependencyService. Hiervoor maken we eerst een Interface aan die ‘IDBInterface’ heet.
6. In de Interface komt 1 methode SQLiteConnection CreateConnection();
7. Maak een nieuwe klasse in de Android folder aan die ‘DatabaseService’ heet. De onderstaande code moet hierin komen:

```
[assembly: Dependency(typeof(DatabaseService))]
namespace Week3.Droid
{
    public class DatabaseService : IDBInterface
    {
        public SQLiteConnection CreateConnection()
        {
            var sqliteFilename = "MovieDatabase.db";
            string documentsDirectoryPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            var path = Path.Combine(documentsDirectoryPath, sqliteFilename);

            if (!File.Exists(path))
            {
                using (var binaryReader = new BinaryReader(Android.App.Application.Context.Assets.Open(sqliteFilename)))
                {
                    using (var binaryWriter = new BinaryWriter(new FileStream(path, FileMode.Create)))
                    {
                        byte[] buffer = new byte[2048];
                        int length = 0;
                        while ((length = binaryReader.Read(buffer, 0, buffer.Length)) > 0)
                        {
                            binaryWriter.Write(buffer, 0, length);
                        }
                    }
                }
            }
            var conn = new SQLiteConnection(path, false);

            return conn;
        }

        void ReadWriteStream(Stream readStream, Stream writeStream)
        {
            int Length = 256;
            byte[] buffer = new byte[Length];
            int bytesRead = readStream.Read(buffer, 0, Length);
            while (bytesRead >= 0)
            {
                writeStream.Write(buffer, 0, bytesRead);
                bytesRead = readStream.Read(buffer, 0, Length);
            }
            readStream.Close();
            writeStream.Close();
        }
    }
}
```

8. Maak een nieuwe klasse in de iOS folder aan die ‘DatabaseService’ heet. De code op de volgende pagina moet hierin komen:

```
[assembly: Dependency(typeof(DatabaseService))]
namespace Week3.iOS
{
    public class DatabaseService : IDBInterface
    {

        public SQLiteConnection CreateConnection()
        {
            var sqliteFilename = "MovieDatabase.db";

            string docFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            string libFolder = Path.Combine(docFolder, "..", "Library", "Databases");

            if (!Directory.Exists(libFolder))
            {
                Directory.CreateDirectory(libFolder);
            }
            string path = Path.Combine(libFolder, sqliteFilename);

            if (!File.Exists(path))
            {
                var existingDb = NSBundle.MainBundle.PathForResource("MovieDatabase", "db");
                File.Copy(existingDb, path);
            }

            var connection = new SQLiteConnection(path, false);
            return connection;
        }
    }
}
```

9. Nu kunnen we in de 'Week4' folder aan de slag. Maak hierin een klasse aan die 'DatabaseManager' heet. Deze klasse gaat SQL query's loslaten op de database.
10. Maak de volgende methodes in de DatabaseManager:
    a. Constructor
    b. List<Movie> GetAllMovies()
    c. List<Series> GetAllSeries()
    d. List<Movie> GetAllMoviesByUser()
    e. List<Series> GetAllSeriesByUser()
    f. bool DoesAccountExist(string username, string password)
    g. bool AddUser(string username, string password)
11. Nu we de database hebben en alle code die daarbij hoort, kunnen we de rest van de applicatie gaan schrijven. Pas in de App.xaml.cs de regel, waarin de MainPage wordt geset, aan naar LoginPage.

LoginPage

1. De LoginPage heeft een instantie nodig van de DatabaseManager. Maak deze aan.
2. Daarnaast moet er op de Login button geklikt kunnen worden, maak hiervoor de 'Clicked' methode aan. Er moet binnen deze methode gecontroleerd worden of er een gebruikersnaam en wachtwoord is ingevuld, maar ook of het account bestaat. Als dit allemaal klopt, wordt de gebruiker naar de MainPage geleidt.
3. Tot slot moet de gebruiker naar de registratiepagina gaan als er op de registratie Label wordt geklikt.

RegisterPage

1. De RegisterPage heeft ook een instantie nodig van de DatabaseManager. Maak deze aan.

2. Als er op de Register button wordt geklikt, moet er gecontroleerd worden of de gebruikersnaam al bestaat in de database. Als dit niet het geval is, moet de gebruiker worden toegevoegd en gaat de nieuwe gebruiker naar de MainPage.

MainPage

1. Ook de MainPage heeft een instantie nodig van de DatabaseManager. Maak deze aan.
2. Op deze pagina willen we de films en series uit de database halen en tonen in de twee ListViews die op de pagina staan. Maak daarom twee lijsten aan, de ene voor de films en de andere voor de series.
3. Vul de twee lijsten in de constructor vanuit de database.
4. Zet de ItemsSource van de ListViews ook in de constructor.
5. Ook willen we in de console straks terug geven of een film of serie tot de favoriet van de gebruiker behoort. Maak hiervoor opnieuw twee lijsten aan voor de favoriete films en favoriete series.
6. Vul deze twee lijsten in de constructor vanuit de database.
7. Wanneer een item in de ListView geselecteerd wordt, moet er True of False terug worden gegeven in de console als een film of serie tot de favoriet van de gebruiker hoort of niet.

Opdracht B – Fragments

Bij deze opdracht wordt opdracht A uitgebreid met een hamburger menu. Hierdoor kunnen we nu door middel van fragments andere informatie laten tonen op meerdere pagina's zonder dat we elke keer weer een pagina op de stack hoeven te plaatsen. Zo zal het geheugen van de telefoon niet zomaar vol raken.

XAML

1. De applicatie wordt verder uitgebreid met drie nieuwe xaml pagina's. Deze heten 'MasterPage', 'FavoritePage' en 'AddPage'.
2. Geef ze allemaal de juiste titels: 'Movies and series', 'Favorites', 'Add'.

MasterPage

1. Verander de ContentPage tag naar <MasterDetailPage></MasterDetailPage>. Dit zal de lay-out van het hamburger menu worden.
2. Een MasterDetailPage heeft een Master en een Detail. Maak de tags <MasterDetailPage.Master></MasterDetailPage.Master> en <MasterDetailPage.Detail></MasterDetailPage.Detail> aan.
3. In de Detail moet een <NavigationPage></NavigationPage> komen.
4. Nu gaan we bezig met de Master. Stop hierin een <ContentPage></ContentPage>. Hierin gaat het menu komen.
5. In de oude Gmail app was de lijst met menu items gescheiden van een profielitem. Daarom maken we een <Grid></Grid> aan in de ContentPage.
6. In dit Grid willen we 2 rijen. Maak daarom een <Grid.RowDefinitions></Grid.RowDefinitions> aan. Hierin gaan we het profielitem en de menuitems van elkaar scheiden.
7. In de RowDefinitions komen twee <RowDefinition></RowDefinition> tags. De eerste met een Height van 200 en de tweede met een Height van *.

8. In de eerste rij van het Grid komt een tweede <Grid></Grid>. Hierin hoeven geen rijen gedefinieerd te worden, want alles moet over elkaar heen komen staan.
    a. Hierin komt een <Image></Image> waarvan de Source gelijk moet komen te staan aan “Images/bg.jpg” (extensie kan ook iets anders zijn). Zoek zelf een leuk achtergrond plaatje. Zorg er ook voor dat de Aspect op AspectFill komt te staan.
    b. Daaronder komt een <StackLayout></StackLayout>. Deze StackLayout heeft een padding van 0, 20, 0, 0. Horizontaal en verticaal moeten zij gecentreerd en expanded zijn.
    c. In de StackLayout komt een <Label></Label> waarin de account naam komt te staan (dit wordt in de code gedaan). Geef de tekst een toepasselijke tekstkleur en zet de font grootte op Large.
9. Onder de Grid komt een <StackLayout></StackLayout>. Deze moet op de tweede rij komen. Zorg ervoor dat de spacing op 8 staat. Hierdoor zullen de menu items mooi worden uitgelijnd.
    a. In de StackLayout komt een <ListView></ListView>. Deze krijgt de naam ‘NavigationDrawerList’. Daarnaast moet de rijhoogte op 48 en moet er geen separator zichtbaar zijn. Er moet ook een functie aan de ListView worden gekoppeld die iets doet als er een item geselecteerd is.
    b. De ListView zal custom zijn. Hierbinnen gaan we dus de <ListView.ItemTemplate></ListView.ItemTemplate> aanmaken.
    c. Maak daarbinnen de <DataTemplate></DataTemplate> tag aan.
    d. In de code maken we straks een CustomViewCell aan, deze voegen we ook alvast aan de lay-out toe. Hierdoor krijgen de geselecteerde items geen vreemde achtergrondkleur. Voeg de tag <local:CustomViewCell></local:CustomViewCell> aan. Dit zal op dit moment nog een error geven.
    e. Geef de CustomViewCell de achtergrondkleur die een geselecteerd item moet hebben (kies zelf).
    f. In de CustomViewCell moet de tag <ViewCell.View></ViewCell.View> komen. Hierin komen de custom elementen voor de ListView.
    g. Nu moeten we de elementen nog aanmaken. Maak hiervoor een <StackLayout></StackLayout> aan. Verticaal moet deze gevuld en expanded worden. De orientatie moet horizontaal zijn. De padding vanaf de linkerkant moet op 20 staan en de spacing op 12.
    h. Hierbinnen komt een <Image></Image> (icon). De source wordt verbonden aan het nog aan te maken model op de variabele ‘Icon’. De breedte en hoogte van deze image moet op 24 komen. Daarnaast moet de image verticaal gecentreerd worden.
    i. Onder de Image komt een <Label></Label>. De tekst wordt verbonden aan de variabele ‘Title’ van hetzelfde model. De font grootte moet default zijn, tekstkleur zwart en verticaal moet de de Label gecentreerd worden.

FavoritePage

1. De FavoritePage is hetzelfde als de MainPage. Gebruik de vorige opdracht om deze pagina te vullen.

AddPage

1. De ContentPage van de AddPage wordt gevuld met een <StackLayout></StackLayout>.
2. We willen kunnen selecteren of we een film of een serie gaan toevoegen. Maak daarom in de StackLayout nog een <StackLayout></StackLayout> aan.
3. Hierin komen twee <Image></Image> tags (MovieButton en SeriesButton). De afbeeldingen moeten 100x100 zijn, een source hebben en een linker en rechter margin van 2.
4. De tweede afbeelding (SeriesButton) moet een Opacity van 0.5 hebben.
5. Beide Images hebben een <TapGetureRecognizer></TapGestureRecognizer> in een <Image.GestureRecognizers></Image.GestureRecognizers> met een tapped functie die 'SelectImageClicked' heet.
6. Onder de StackLayout komt een <ScrollView></ScrollView>. Dit omdat niet alles op 1 pagina past en je er beter doorheen kan scrollen.
7. In de ScrollView komt een <StackLayout></StackLayout> met de orientatie op verticaal.
8. In de StackLayout komen 3 <Label></Label> en <Entry></Entry> tags voor 'Title', 'Description' en 'Year'.
9. Daaronder komen 3 <Label></Label> en <Entry></Entry> tags voor 'YearEnded', 'NumberOfEpisodes' en 'NumberOfSeasons' die niet zichtbaar zijn.
10. En tot slot komt er nog een <Button><Button> tag. De Button heeft de naam 'AddButton' en bevat de tekst 'Add'. Geef de Button ook een toepasselijke achtergrondkleur en tekstkleur. Ook moet er een klik functie komen, die 'AddClicked' heet. Centreer de Button horizontaal en geef de grootte 32x96. De padding moet op 2 en de margin op 0, 16, 0, 0.

Backend

1. Maak een nieuwe folder aan die 'MenuItems' heet. Voeg aan deze map een nieuwe klasse toe die 'MasterPageItem' heet. Dit wordt het model voor de menu items.
2. Het MasterPageItem model heeft een 'Title' van het type string, een 'Icon' van het type string en een 'TargetType' van het type Type.
3. In de lay-out hebben we gebruik gemaakt van een CustomViewCell. We gaan nu deze klasse aanmaken in het project.
4. Voeg de onderstaande regel toe aan de CustomViewCell:
   public static readonly BindableProperty SelectedItemBackgroundColorProperty = BindableProperty.Create("SelectedItemBackgroundColor", typeof(Color), typeof(CustomViewCell), null);
5. Vervolgens moet er nog een variabele komen die ervoor zorgt dat de kleur ook in de code kan worden veranderd:

```
public Color SelectedItemBackgroundColor
{
    get
    {
        return (Color)GetValue(SelectedItemBackgroundColorProperty);
    }
    set
    {
        SetValue(SelectedItemBackgroundColorProperty, value);
    }
}
```

6. In de Android en iOS folders moeten twee CustomViewCellRenderder klassen worden toegevoegd. Maak deze aan. Stop de juiste code, die hieronder staat, in de twee klassen.

```
using System.ComponentModel;
using Week4;
using Week4.Droid.CustomControls;
using Android.Content;
using Android.Graphics.Drawables;
using Android.Views;
using Xamarin.Forms;
using Xamarin.Forms.Platform.Android;
[assembly: ExportRenderer(typeof(CustomViewCell), typeof(CustomViewCellRenderer))]
namespace Week4.Droid.CustomControls
{
    public class CustomViewCellRenderer : ViewCellRenderer
    {
        private Android.Views.View _cellCore;
        private Drawable _unselectedBackground;
        private bool _selected;
        protected override Android.Views.View GetCellCore(Cell item, Android.Views.View convertView, ViewGroup parent, Context context)
        {
            _cellCore = base.GetCellCore(item, convertView, parent, context);
            _selected = false;
            _unselectedBackground = _cellCore.Background;
            return _cellCore;
        }
        protected override void OnCellPropertyChanged(object sender, PropertyChangedEventArgs e)
        {
            base.OnCellPropertyChanged(sender, e);
            if (e.PropertyName == "IsSelected")
            {
                _selected = !_selected;
                if (_selected)
                {
                    var extendedViewCell = sender as CustomViewCell;
                    _cellCore.SetBackgroundColor(extendedViewCell.SelectedItemBackgroundColor.ToAndroid());
                }
                else
                {
                    _cellCore.SetBackground(_unselectedBackground);
                }
            }
        }
    }
}
```

```
using Week4;
using UIKit;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;
using Week4.iOS.CustomControls;
[assembly: ExportRenderer(typeof(CustomViewCell), typeof(CustomViewCellRenderer))]
namespace Week4.iOS.CustomControls
{
    public class CustomViewCellRenderer : ViewCellRenderer
    {
        public override UITableViewCell GetCell(Cell item, UITableViewCell reusableCell, UITableView tv)
        {
            var cell = base.GetCell(item, reusableCell, tv);
            var view = item as CustomViewCell;
            cell.SelectedBackgroundView = new UIView
            {
                BackgroundColor = view.SelectedItemBackgroundColor.ToUIColor(),
            };
            return cell;
        }
    }
}
```

DatabaseManager

1. Maak de volgende methodes in de DatabaseManager:
    a. User GetUser()
    b. void AddMovie(string title, string description, int year)
    c. void AddSeries(string title, string description, int yearStarted, int yearEnded, int numberOfEpisodes, int numberOfSeasons)

MainPage

1. Code weghalen die ervoor zorgt dat de favorieten worden getoond.

MasterPage

1. Op dit moment implementeert de MasterPage nog een ContentPage. Verander dit in een MasterDetailPage.
2. De MasterPage is voor de menu items, dus hiervoor moet een lijst komen. Maak een lijst aan voor de menu items.
3. In de constructor moeten de volgende dingen gedaan worden:
    a. Zet de AccountName naar de gebruikersnaam van de ingelogde gebruiker.
    b. Vul de lijst met 3 menu items, namelijk: 'Movies & series', 'Favorites' en 'Add'.
    c. Zet de ItemsSource van de NavigationDrawerList naar de lijst met menu items.
    d. De onderstaande regel zorgt ervoor dat er een Detail is:

    Detail = new NavigationPage((Page)Activator.CreateInstance(typeof(MainPage)));
4. Maak de methode voor OnMenuItemSelected:
    a. Zorg ervoor dat je het huidige geselecteerde item kan opvragen
    b. Zorg ervoor dat je daarna het Type kan opvragen.
    c. Zorg ervoor dat je daarna de nieuwe Detail zet naar het geklikte Type.
    d. Zorg ervoor dat de IsPresented daarna op false gaat.

FavoritePage

1. De FavoritePage werkt hetzelfde als de MainPage, echter willen we hier alleen de favoriete films en series van de huidige gebruiker tonen.

AddPage

1. De AddPage heeft als attributen een bool nodig die controleert of er een film of serie gaat worden toegevoegd & een instantie van de DatabaseManager klasse.
2. Er moeten 2 methodes worden aangemaakt in deze klasse:
    a. **AddClicked**: Controleer eerst of de 3 zichtbare velden gevuld zijn. Als er een film wordt toegevoegd, voeg deze dan toe aan de database. Als er een serie wordt toegevoegd, controleer dan of het aantal episodes en seizoenen ook is ingevuld. Zorg ervoor dat er -1 wordt meegegeven voor yearEnded als er geen eindjaar is ingevuld.
    b. **SelectImageClicked**: Als film is geselecteerd en film was daarvoor niet geselecteerd, zorg ervoor dat de bool op true komt te staan, de opacity van MovieButton op 1.0 en die van SeriesButton op 0.5, de invisible labels op invisible.
    Als serie is geselecteerd en serie was daarvoor niet geselecteerd, zorg ervoor dat de bool op false komt te staan, de opacity van MovieButton op 0.5 en die van SeriesButton op 1.0, de invisible labels op visible.

Week 5

Beschrijving

De vijfde week aan opdrachten. Koppeling tussen de arduino en een mobiele applicatie. Dit is een verdieping op de derde week.

Doel

Het doel van deze opdracht is om te leren:

- Hoe het embedded systeem gekoppeld kan worden aan de mobiele applicatie

Opdracht A – Blinking LED

In deze opdracht wordt er vanaf de telefoon verbinding gemaakt met een Arduino. In opdracht 4 van Embedded Systems staat een opdracht waarbij je een LEDje moet laten knipperen. Gebruik deze code bij het maken van deze opdracht.

1. Maak een nieuwe Xamarin Forms applicatie aan. Als je niet meer weet hoe dit moet, kun je terugkijken naar Opdracht A van week 1. Noem dit project “Week5” en plaats deze in de folder die jij hebt aangemaakt voor al jouw Apps Programmeren huiswerkopgaven.

XAML

1. Verwijder alle code uit de <StackLayout></StackLayout>.
2. Geef de StackLayout een margin van 16 en zorg ervoor dat deze verticaal wordt gecentreerd.
3. Maak een <Slider></Slider> aan.
4. Zorg ervoor dat deze Slider een toepasselijke naam krijgt. Verander het maximum naar 100 en definieer een functie die wordt aangeroepen wanneer de waarde wordt geüpdatet.
5. Onder de Slider komt een <Label></Label>.
6. Deze Label moet een toepasselijke naam krijgen, de tekst moet de grootte 64 krijgen, horizontaal moet deze Label gecentreerd worden en de tekst in het begin moet 0 zijn.
7. Tot slot moet er nog een <Button></Button> in de lay-out komen. Deze Button moet een toepasselijke naam krijgen, horizontaal gecentreerd worden, de tekst “Send” moet op de Button staan en als erop wordt geklikt moet er een functie aan worden geroepen.
8. Pas je lay-out nu aan dat de achtergrond een andere kleur krijgt.
9. Zorg er in jouw Android applicatie voor dat de **ColorPrimary**, **ColorPrimaryDark** en **AccentColor** zijn verandert.
10. Geef de Button en kleur van het bolletje op de slider dezelfde kleur als **ColorPrimary**.

Backend

1. In de backend code voegen we eerst de twee methoden toe, die zijn gedefinieerd in de lay-out.
2. Zorg er in de Update functie voor dat de tekst van Label zich aanpast naar de waarde van de Slider.
3. Nu de front-end en backend grotendeels in elkaar zit, moeten we de verbinding met de Arduino opzetten. Maak een nieuwe klasse aan en noem deze “Connection”.

Deze klasse gaat ervoor zorgen dat er een verbinding met de Arduino komt, dat er naartoe geschreven kan worden, vanaf gelezen kan worden en dat de verbinding gesloten kan worden.

4. Maak een variabele aan "_connectionSocket" van het type Socket.
5. Schrijf de methode om verbinding met de Arduino te openen.
6. Schrijf de methode om gegevens over te sturen naar de Arduino.
7. Schrijf de methode om gegevens te ontvangen van de Arduino.
8. Schrijf de methode om de verbinding te sluiten.

Pas de functie van de Button aan zodat de waarde van de slider wordt overgestuurd naar de Arduino (maak gebruik van de "Connection" klasse).

Opdracht B – Updating data (BONUS)

Soms wil je ook dat data automatisch wordt bijgewerkt in jouw applicatie. Neem bijvoorbeeld de temperatuur. Hiervoor is een DHT 11 luchtvochtigheidssensor nodig.

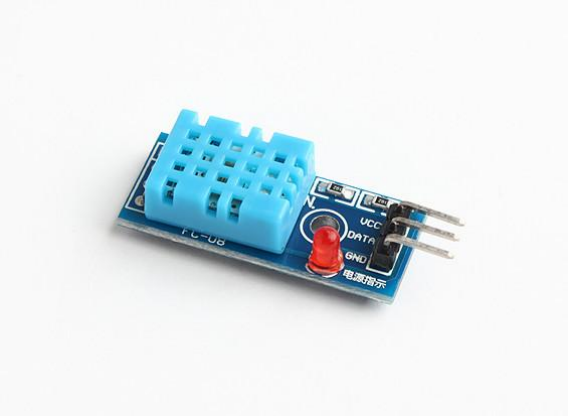


Figure 1: DHT 11 humidity sensor

De temperatuur in huis daalt naar een graag of 16, terwijl jij het gemiddeld 20 graden wilt hebben. In jouw applicatie, zoals je nu kan programmeren, heb je geprogrammeerd dat de temperatuur wordt bijgewerkt wanneer de applicatie opstart of dat er op een Button wordt geklikt. Dit is natuurlijk niet handig, want je wilt het direct zien als de temperatuur daalt. De data moet dus automatisch worden bijgewerkt!

Maak voor deze bonusopdracht een nieuw project aan en noem deze "Week2Bonus".

XAML

Maak een lay-out in XAML die de temperatuur laat zien. En zorg ervoor dat dit element kan worden aangepast in de backend code.

Backend

Maak in de backend code verbinding met de Arduino. Zorg ervoor dat de luchtvochtigheidssensor kan worden uitgelezen in de app. De temperatuur en de luchtvochtigheid moet vervolgens in de lay-out getoond worden.