HBO-ICT & Ad ITSM

Programmeren 2

Opdrachten

Foppele, J. & Mross, S.
11-18-2020

NHL STENDEN hogeschool

Programmeren 2 Opdrachten

Inhoud

Week 1

Beschrijving

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

Doel

Het doel van deze opdracht is om te leren:

- Omgaan met classes
- Gebruikmaken van lijsten
- Een simpele Xamarin applicatie op te zetten

Opdracht A – GUI in Xamarin Forms

Loop de volgende stappen door:

1. Maak een nieuw Xamarin Forms project aan. Selecteer een blank layout als template.
2. Vervang de gehele gegeven StackLayout inclusief content voor onderstaand Grid inclusief content.

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <StackLayout>
        <Label Text="Persoon" FontAttributes="Bold" FontSize="Large"></Label>
        <Label Text="Voornaam"></Label>
        <Entry x:Name="Voornaam"></Entry>
        <Label Text="Achternaam"></Label>
        <Entry x:Name="Achternaam"></Entry>
        <Label Text="Leeftijd"></Label>
        <Entry x:Name="Leeftijd"></Entry>
        <!-- Entry kan vervangen worden door een Spinner of Slider -->
        <Button Text="Toevoegen" Clicked="AddPersonClicked"></Button>
    </StackLayout>
    <ListView x:Name="ListViewPersonen" Grid.Column="1"
              ItemSelected="PersonSelected">
    </ListView>
    <StackLayout Grid.Row="1">
        <Label Text="Huisdier" FontAttributes="Bold" FontSize="Large"></Label>
        <Label Text="Naam"></Label>
        <Entry x:Name="HuisdierNaam"></Entry>
        <Label Text="Soort"></Label>
        <Entry x:Name="HuisdierSoort"></Entry>
        <Button Text="Toevoegen" Clicked="AddAnimalClicked"></Button>
    </StackLayout>
    <ListView x:Name="ListViewHuisdieren" Grid.Row="1" Grid.Column="1">
    </ListView>
</Grid>
```

Test of de interface zichtbaar is in de editor.

3. Je hebt nu de interface gemaakt, maar hierbij is nog code nodig die deze interface gebruikt. Neem hiervoor de onderstaande code over in de code van de MainPage.

```
public void AddPersonClicked(Object sender, EventArgs e)
{
    //TODO Add a person to the list persons
}

public async void PersonSelected(Object sender, SelectedItemChangedEventArgs e)
{
    //TODO Respond on person click event
}

public void AddAnimalClicked(Object sender, EventArgs e)
{
    //TODO Add an animal to the person currently selected
}
```

Check of je de app nu ook kunt runnen of simuleren.

4. Gebruik de code die je tot nu toe hebt voor Opdracht B.

Opdracht B – Persoon

Breid de code uit Opdracht A uit. We gaan nu een class toevoegen en de werking hiervan

1. Maak een nieuwe class en noem deze Person (in een nieuw bestand). Geef een persoon de private attributen firstName, lastName en age (respectievelijk van het type string, string en int).
2. Maak een constructor waarmee je de Person aan kunt maken. Zorg ervoor dat je de 3 attributen kunt instellen met parameters van de constructor.
3. Voeg de onderstaande methoden toe.

```
public string GetName() { return firstName + " " + lastName; }

public int GetAge() { return age; }

public override string ToString() { return GetName(); }
```

Voeg XML Doc toe aan de 4 methoden die nu in de class zitten.

We gaan nu de Person class, die tot Opdracht A in week 2 niet meer hoeft te veranderen, koppelen aan onze eerder gemaakte interface. Om het simpel te houden gaan we nog geen ingewikkelde databindings gebruiken (deze volgen volgende week).

4. Voeg in de MainPage een private property toe van het type List<Person>. Geef deze lijst de naam persons en zorg dat de list ook aangemaakt is.
5. Zorg ervoor dat in de AddPersonClicked methode een persoon wordt aangemaakt met de ingevoerde gegevens (Voornaam, Achternaam en Leeftijd). Voeg daarna deze persoon toe aan de lijst persons.
6. Voeg onderstaande regels toe aan de AddPersonClicked methode.

```
ListViewPersonen.ItemsSource = null;
ListViewPersonen.ItemsSource = persons;
```

Dit lijkt omslachtig, maar het op null zetten van de ItemSource is nodig om het refreshen van de interface af te dwingen.

7. OPTIONEEL: Voeg in de MainPage constructor jezelf toe aan de lijst zodat je niet bij iedere test van je systeem een nieuw persoon hoeft in te voeren. Voeg daarnaast nog een random persoon toe aan de lijst (je zou bijvoorbeeld een docent kunnen nemen, maar ook een fictief persoon mogen toevoegen). Krijg je het voor elkaar om dit ook al zichtbaar tee maken in de ListView zonder een derde persoon toe te voegen? HINT: denk aan de ItemSource.

8. Test de werking van de GUI, kun je personen toevoegen aan de lijst? Test ook of je een persoon uit de lijst visueel kunt selecteren.
9. Voeg aan de attributen van de MainPage een selectedPerson van het type Person toe. Maak ook dit attribuut private en vul het met niets (null).
10. Voeg nu de volgende code toe aan de methode PersonSelected:
```
selectedPerson = (Person)e.SelectedItem;
if (selectedPerson != null)
    await DisplayAlert("Info", selectedPerson.GetName(), "OK");
```
Verklaar ook wat deze code doet.
11. Zet aan het einde van de AddPersonClicked methode de selectedPerson op de nieuw aangemaakte persoon.

Bewaar je uitwerkingen zodat je er volgende week mee verder kunt! Heb je tijd over, dan kun je al opdracht A van week 2 ook doorwerken. Dit scheelt je volgende week tijd.

Week 2

Beschrijving

Deze week gaan we verder met onze huisdieren app waar we vorige week mee zijn begonnen.

Doel

Het doel van deze opdracht is om te leren:

- Werken met Bindings
- Toepassen van overerving

Opdracht A – Huisdieren

Nu gaan we zorgen dat ieder persoon ook een aantal huisdieren kan hebben. Hiervoor gaan we de volgende stappen ondernemen:

1. Open je app van vorige week (of een kopie hiervan) zodat je deze code kunt verbeteren en uitbreiden.
2. Voeg een nieuwe class genaamd Animal toe, ook weer als nieuw bestand. Geef deze class de private attributen name en kind, beide van het type string. Maak ook een constructor.
3. Voeg de functie ToString toe. Deze functie geeft de naam en het soort van het dier terug als string, gescheiden door een spatie. Let op: je hebt net als bij de Person ook bij deze ToString een override nodig.
4. Voeg aan de Person class een private attribuut toe genaamd pets. Dit is een lijst gevuld met objecten van het type Animal. Zorg er ook voor dat de lijst aangemaakt is.
5. Maak de functie AddPet in Person. Deze functie moet een animal, meegegeven als parameter, toevoegen aan de lijst pets. Deze functie heeft geen returnType.
6. Maak de functie GetPets in Person. Deze functie geeft de lijst pets terug.
7. In de AddAnimalClicked methode moet een animal aangemaakt worden op basis van de entries HuisdierNaam en HuisdierSoort. Zorg ervoor dat het aangemaakte huisdier aan de geselecteerde persoon wordt toegevoegd. Houd er rekening mee dat het kan voorkomen dat er nog geen persoon is geselecteerd, dan kan er ook niets worden toegevoegd! Maak de lijst van huisdieren van de geselecteerde persoon daarna ook zichtbaar in de ListViewHuisdieren.
8. Vervang de alert in de PersonSelected methode voor het updaten van de ListViewHuisdieren. Valt je op dat de async access modifier nu ook niet meer nodig is? Kun je verklaren hoe dit komt?
9. OPTIONEEL: Voeg in de MainPage constructor enkele huisdieren toe aan de voorgedefineerde personen. Dit maakt het testen van je app makkelijker.
10. Zorg ervoor dat je na invoeren van gegevens de juiste invoervelden weer leeg zijn. Dit verhoogt het gebruiksgemak.
11. OPTIONEEL: Krijg jij het voor elkaar om wanneer een huisdier wordt geselecteerd ook een pop-up te tonen met daarin de info van het huisdier én het baasje?
12. OPTIONEEL: Voeg nog een extra invoerveld toe aan het huisdier en verwerk dit op de juiste manier in de class.
13. OPTIONEEL: Vervang de entry van leeftijd door een Spinner of Slider.

Opdracht B – Xamarin Bindings

Voer de volgende opdrachten uit:

1. We gaan een aantal dingen veranderen zodat we een goede databinding kunnen maken. Ten eerste gaan we de classes Person en Animal public maken. Dit doen we door voor het keyword class public te zetten.
2. Nu gaan we de Person class verder aanpassen zodat deze geschikt wordt voor DataBinding. We halen de private variabele age weg (firstName en lastName laten we staan) en voegen de volgende code bij als attributen:
   ```
   public string Name { get { return firstName + " " + lastName; } }
   public int Age { get; set; }
   ```
   We zien hier een typische manier van declareren die alleen werkt binnen C#, met get en set. Wen hier niet te veel aan want andere programmeertalen hebben weer andere gespecialiseerde constructies. De voorheen geleerde constructies zijn wel universeel te gebruiken.
3. Haal nu ook de GetName, GetAge en ToString methoden weg uit Person. Verander ook de constructor zo dat de meegegeven parameter voor age in de vernieuwde Age terecht komt.
4. Nu halen we in de MainPage alle veranderingen aan de ListViewPersonen.ItemSource weg. Deze zitten waarschijnlijk in de constructor en de AddPersonClicked methode.
5. Verander de List persons in onderstaande code.
   ```
   public ObservableCollection<Person> persons { get; private set; }
   ```
   Zorg er ook voor dat persons wordt aangemaakt in de constructor. Een ObservableCollection werkt ongeveer hetzelfde als een List, je ziet dit ook omdat je alle andere code waar persons wordt gebruikt niet hoeft te veranderen.
6. Voeg nu de volgende regel nog toe aan de constructor van de MainPage:
   ```
   BindingContext = this;
   ```
   Hiermee zorgen we ervoor dat de binding daadwerkelijk werkt.
7. Nu staat alles klaar om een binding op te zetten. We gaan nu de ListView uitbreiden met onderstaande code, waarin de databinding naar de lijst persons staat, maar ook per item een databinding naar de aangemaakte Name en Age van een Person.
   ```
   <ListView x:Name="ListViewPersonen" Grid.Column="1"
                    ItemSelected="PersonSelected"
                    ItemsSource="{Binding persons}">
       <ListView.ItemTemplate>
           <DataTemplate>
               <ViewCell>
                   <StackLayout>
                       <Label Text="{Binding Name}"></Label>
                       <Label Text="{Binding Age}"></Label>
                   </StackLayout>
               </ViewCell>
           </DataTemplate>
       </ListView.ItemTemplate>
   </ListView>
   ```
   Valt je op dat ieder item is voorzien van een StackLayout? Uiteraard hadden we het visueel ook anders kunnen aanpakken door andere layouts te kiezen.
8. Test je app. Is de werking nog hetzelfde als voorheen? Zie je meer info in de lijst met personen? OPTIONEEL: experimenteer met de layout van de lijst met personen totdat die aan jouw smaak voldoet.

Opdracht C – Animals en overerving

We gaan de Animal class veranderen in een abstracte class, en de verschillende huisdiersoorten toevoegen aan onze app als van Animal ervende classes. Voer de volgende opdrachten uit:

1. Verander als eerste de Animal class in een abstracte class. Verwijder de constructor en het attribuut kind. Verander Name in onderstaande code
   ```
   public string Name { get; protected set; }
   ```
   Het is belangrijk om de set protected te hebben en niet private. Kun jij dit verschil verklaren?
2. Voeg nu de abstracte methode GetAnimalType toe aan Animal. Deze methode moet een string teruggeven en heeft geen parameters.
3. Voeg nu minstens 2 subclasses toe die erven van Animal. Bijvoorbeeld Cat en Dog. Zorg ervoor dat de GetAnimalType methode de naam van de diersoort teruggeeft. Ook moet met de constructor van ieder class de naam van het dier als parameter meegegeven worden en in Name worden gezet.
4. Voeg in Animal ook de volgende methode toe
   ```
   public override string ToString()
   {
       return Name + " " + GetAnimalType();
   }
   ```
   Valt je op dat deze methode niet overschreven hoeft te worden in de ervende classes? Je zou daarom deze methode ook sealed mogen maken. Wat zijn hiervan de voor- en nadelen?
5. Verander in de MainPage code de methode AddAnimalClicked zodat er afhankelijk van de ingevoerde huisdiersoort het juiste object wordt aangemaakt.
6. Verander pets in de class Person in een ObservableCollection.
7. Test je app. Werk eventuele errors in je code weg (die kunnen voorkomen in het gedeelte waar je testgegevens inlaadt).
8. OPTIONEEL: verander de XAML zodat je meer invloed hebt op de layout van de lijst met huisdieren.

Week 3

Beschrijving

In tegenstelling tot mensen en hun huisdieren, gaan we ons nu bezig houden met helden en schurken (heroes and villains). Hiervoor gaan we een nieuwe app maken. Ook gaan we ervoor zorgen dat we al onze data kunnen opslaan in een database.

Doel

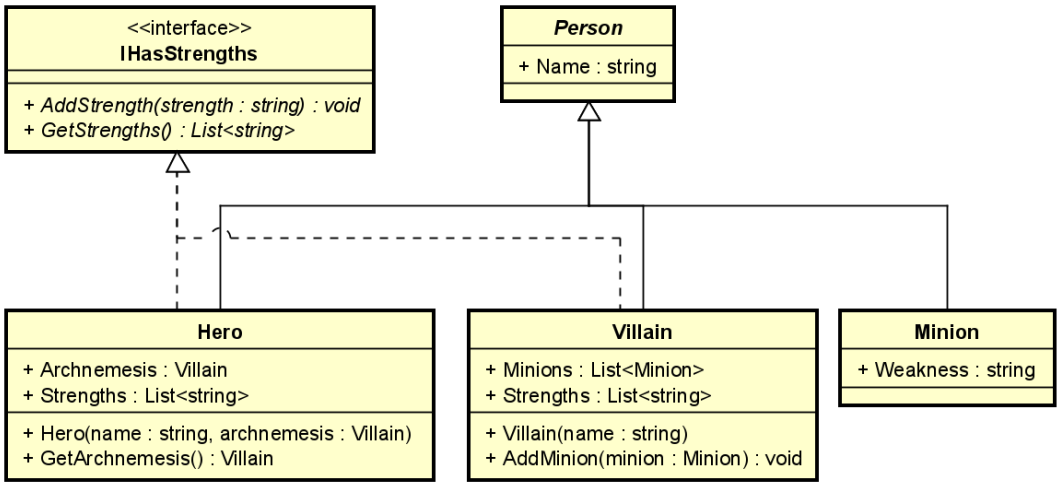
Het doel van deze opdracht is om te leren:

- Omgaan met overerving
- Gebruikmaken van een user interface

Opdracht A – Heroes and Villains

Voer de volgende opdrachten uit:

1. Maak een nieuw Xamarin Project aan. Aanbevolen wordt om de blank layout template te gebruiken, maar je kunt ook experimenteren met andere templates als je dat wilt. Let er dan wel op dat je allerlei Design Patterns voor je kiezen krijgt wat tweede jaars stof is.
2. Gegeven is onderstaand klassediagram.



   Werk dit klassediagram uit in code. Bedenk hierbij zelf de constructor voor Minion. Alle attributen zijn public en hebben de C# { get; set; } constructie. Let op de klasse Person, die abstract is.
3. Breid de klasse Hero uit met een sidekick. Een hero kan maximaal 1 sidekick hebben. Een sidekick kan een andere hero zijn, maar ook een villain of een minion. Voeg een tweede contructor toe zodat een hero met naam, villain en sidekick geïnstantieerd kan worden. Zorg er voor dat een Sidekick aanpasbaar is met de methode UpdateSidekick(sidekick) en opvraagbaar is met de methode GetSidekick().
4. Een persoon is vrijwel nooit geheel goed of slecht. Voeg hiertoe aan Person een evilnessIndicator toe die kan variëren tussen 0 en 100. Als een evilness buiten deze range wordt opgegeven, moet de evilness op 50 gezet worden. De evilness moet alleen gezet kunnen worden in de constructor. Zorg voor nieuwe constructors zonder de oude constructors weg te halen of te veranderen. Evilness moet opvraagbaar zijn met behulp van de methode GetEvilness().
   P.S. evilness loopt van 0 (totale goedheid) tot 100 (most evil ever a.k.a. docent)
5. Maak XML Doc bij al de classes en methodes.
6. OPTIONEEL: Maak voor strengths ook een aparte class aan en gebruik deze in plaats van de strings in Hero, Villain en IHasStrengths.

7. Zorg ervoor dat er in de MainPage 3 ObservableCollections zijn, 1 met Heroes (heroCollection), 1 met Villains (villainCollection) en 1 met Minions (minionCollection).
8. Test of alle classes en methodes werken. Doe dit bijvoorbeeld door onderstaande methode toe te voegen aan de MainPage en aan te roepen vanuit de constructor, of schrijf zelf hier code of tests voor. Zet aan het eind van de methode een breakpoint zodat je de tekst in de output (meestal rechtsonder) kan bekijken.

```
public void testMyCode()
{
  // voor de test: toevoegen personen en eigenschappen
  Villain ozymandias = new Villain("Adrian Veidt");
  Villain joker = new Villain("Mr. J");
  Hero robin = new Hero("Jason Todd", joker);
  Hero batman = new Hero("Bruce Wayne", joker);
  batman.AddStrength("Intelligence");
  batman.AddStrength("Wealth");
  batman.UpdateSidekick(robin);
  Hero rorschach = new Hero("Walter Joseph Kovacs", ozymandias);

  Villain gru = new Villain("Gru");
  gru.AddStrength("Inspiration");
  Minion bob = new Minion("Bob");
  Minion kevin = new Minion("Kevin");
  Minion stuart = new Minion("Stuart");
  gru.AddMinion(bob);
  gru.AddMinion(kevin);
  gru.AddMinion(stuart);

  heroCollection.Add(batman);
  heroCollection.Add(robin);
  heroCollection.Add(rorschach);

  // Output om de functies te testen

  Debug.WriteLine("Aantal toegevoegde heroes: " + heroCollection.Count);
  int count = 0;
  foreach (Hero h in heroCollection)
  {
     Debug.WriteLine(count + ". " + h.Name);
     Debug.WriteLine("Strength: " + String.Join("; ", h.GetStrength()));
     Debug.WriteLine("Evilness: " + h.GetEvilness());
     Debug.WriteLine("Archnemesis " + h.GetArchnemesis() == null ? "" :
                                                    h.GetArchnemesis().Name);
     count++;
  }
  // TODO: Test villain: voeg de villains toe aan de collectie en test deze net als de heroes
  Debug.WriteLine("Aantal villains: " + villainCollection.Count);

  // TODO: Test minions: voeg de minions toe aan de collectie en test deze net als de heroes
  Debug.WriteLine("Aantal minions: " + minionCollection.Count);

 } //set breakpoint
```

Werkt dit allemaal in 1 keer? En zo nee, wat gaat er mis? Los eventuele fouten op.

Opdracht B – Tabbed Layout

We gaan nu de interface voor de app toevoegen. We gaan 3 verschillende tabs gebruiken in onze interface, een verschillende tab voor Hero, Villain en Minion. Voer hiervoor de volgende opdrachten uit:

1. Verander de MainPage van ContentPage naar TabbedPage. Doe dit door in de code MainPage te laten erven van TabbedPage in plaats van ContentPage. Neem in de XAML van de MainPage de volgende code over:

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="XamarinWeek3example.MainPage">
    <TabbedPage.Children>
        <ContentPage Title="Heroes">
            <StackLayout>
                <Frame BackgroundColor="#CE8ED5" Padding="24" CornerRadius="0">
                    <Label Text="Heroes" HorizontalTextAlignment="Center"
                           TextColor="White" FontSize="36"/>
                </Frame>
                <Label Text="Hier komen de Heroes" FontSize="Title"
                       Padding="30,10,30,10"/>
            </StackLayout>
        </ContentPage>
        <ContentPage Title="Villains">
            <StackLayout>
                <Frame BackgroundColor="#E128F5" Padding="24" CornerRadius="0">
                    <Label Text="Villains" HorizontalTextAlignment="Center"
                           TextColor="White" FontSize="36"/>
                </Frame>
                <Label Text="Hier komen de Villains" FontSize="Title"
                       Padding="30,10,30,10"/>
            </StackLayout>
        </ContentPage>
        <ContentPage Title="Minions">
            <StackLayout>
                <Frame BackgroundColor="#E4E693" Padding="24" CornerRadius="0">
                    <Label Text="Minions" HorizontalTextAlignment="Center"
                           TextColor="White" FontSize="36"/>
                </Frame>
                <Label Text="Hier komen de Minions" FontSize="Title"
                       Padding="30,10,30,10"/>
            </StackLayout>
        </ContentPage>
    </TabbedPage.Children>
</TabbedPage>
```

   Vervang bij x:Class XamarinWeek3example voor je eigen namespace (zie ook de code). Test of je nu in je app 3 tabs hebt met placeholder tekst.
2. Maak nu in iedere tab een interface waarbij je een ListView gebruikt om de juiste ObservableCollection te laten zien. Zorg er ook voor dat er (vergelijkbaar met de app van vorige week) een gedeelte is waar je nieuwe Heroes/Villains/Minions aan de lijst kunt toevoegen. Gebruik hierbij databindings.
3. OPTIONEEL: als je voor strength een aparte class hebt gemaakt kun je hiervoor ook een aparte tab maken in je interface om de strengths inzichtelijk te maken. Vergeet niet om hiervoor ook een ObservableCollection aan te maken.

Week 4

Beschrijving

Deze week gaan we een app maken die kan communiceren met een ander systeem. Dit andere systeem kan op je computer draaien als aparte applicatie, maar kan ook bijvoorbeeld een Arduino zijn. We gaan er bij de opdrachten vanuit dat je geen Arduino tot je beschikking hebt, je kunt daarom de MobileServer gebruiken om je app met Socket Client te testen. Deze MobileServer kun je downloaden van blackboard.

Doel

Het doel van deze opdracht is om te leren:

- Communiceren met andere systemen
- Wat de werking is van Sockets

Opdracht A – Client Applicatie

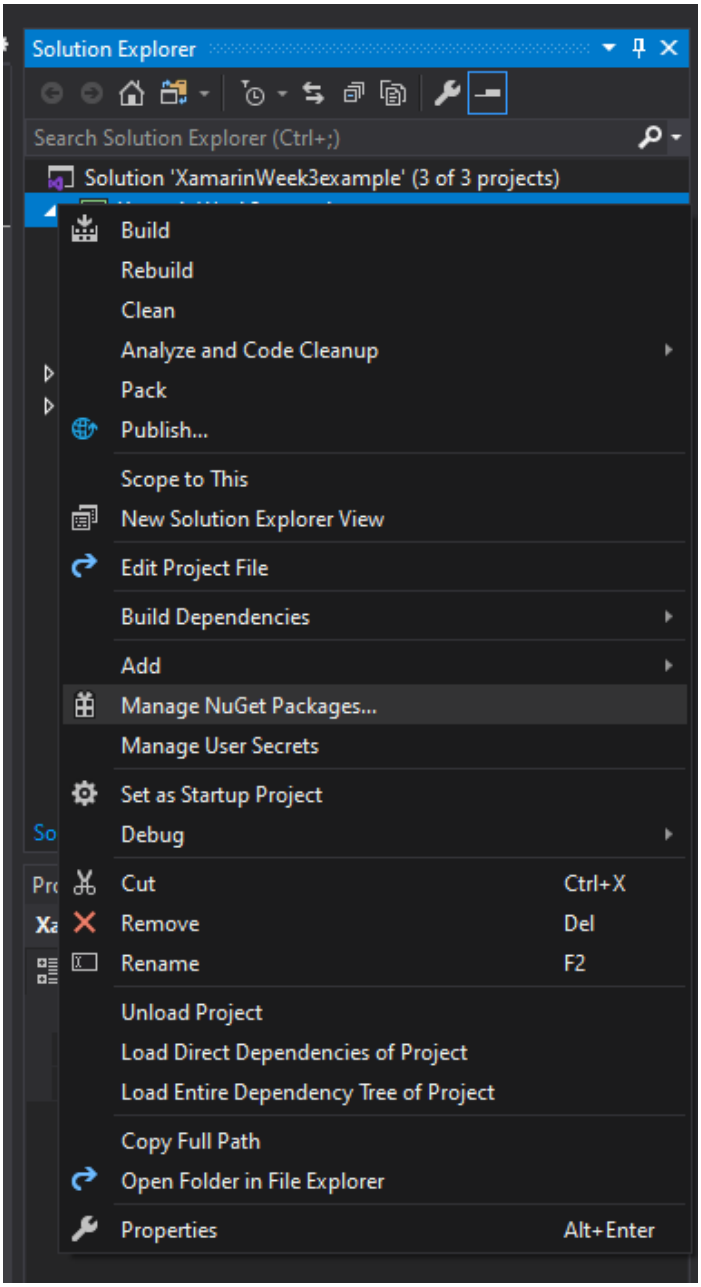
1. Download de MobileServer applicatie en zorg ervoor dat je deze kunt draaien op je systeem. Dit is een Console Applicatie. Zorg er ook voor dat je de app de nodige permissies geeft, mocht daarom gevraagd worden (door bijvoorbeeld de firewall). Pas het IPAddress aan zodat deze het adres is van je eigen Computer.
2. Maak een nieuwe Xamarin Project met een blank layout. Dit gaat onze client worden die met de server kan verbinden en een (random) temperatuur kan ophalen.
3. Zorg ervoor dat je in de XAML een tag hebt waarin je de temperatuur gaat laten zien. Dit kan bijvoorbeeld een label zijn. Maak ook een knop die updateButton heet.
4. Voeg een nieuwe class genaamd Client toe aan je App, hierin gaan we alle communicatie regelen.
5. 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 overeenkomt met die van de server, evenals de poort.
6. Vervolgens moet de Client dus een bericht kunnen versturen naar de server in dezelfde methode, waarop de server een bericht terugstuurt (de temperatuur). Maak dit.
7. Zorg ervoor dat de Client daarna een bericht kan ontvangen en geef dit bericht terug als resultaat van de methode.
8. In de Client moeten foutmeldingen kunnen worden afgevangen. Doe dit met een try/catch.
9. Het ontvangen bericht moet op de pagina, de MainPage, getoond kunnen worden in de tag die is aangemaakt in de XAML. Doe dit door vanuit de MainPage de statische methode van de Client class aan te roepen en het resultaat in de GUI te tonen. Doe dit op zo'n manier dat je met een druk op de knop nieuwe info kunt ophalen.
10. Maak het IP-adres en poort invulbaar in de interface van je app, zodat je meer controle hebt over waar je app mee connect. Zorg ook voor een label waarin je de status van de connectie kunt zetten (een andere visuele weergave mag ook).
11. Voeg een extra commando toe naast <TEMP>. Dit commando naar de server moet zijn <PING> en de server moet reageren met <OK>. Je moet dus zowel de server als de client code hierbij uitbreiden. Zorg ervoor dat zodra in de app het IP-adres en poortnummer zijn ingevuld, dat een ping naar de server wordt verstuurd en de statusweergave naar "connected" gaat als er <OK> terugkomt.
12. Maak een nieuwe class DataValue aan. Deze heeft de publieke property 'Value'. 'Value' is een integer. Let op, je maakt een property, dus vergeet { set; get; } niet. Maak in de class ook nog een constructor met een parameter. Hier kan je de ontvangen temperatuur meegeven. Deze constructor zet de property 'Value'. Dit heb je nodig voor Opdracht B.

13. OPTIONEEL: Als je een Arduino, een ethernet shield en een temperatuur sensor hebt kun je hiermee ook een andere server maken waar je telefoon mee connect. Bedenk hiervoor de code (kijk ook op https://www.arduino.cc/en/Reference/EthernetServer) en zorg dat je in je app het juiste IP-adres en poort gebruikt.

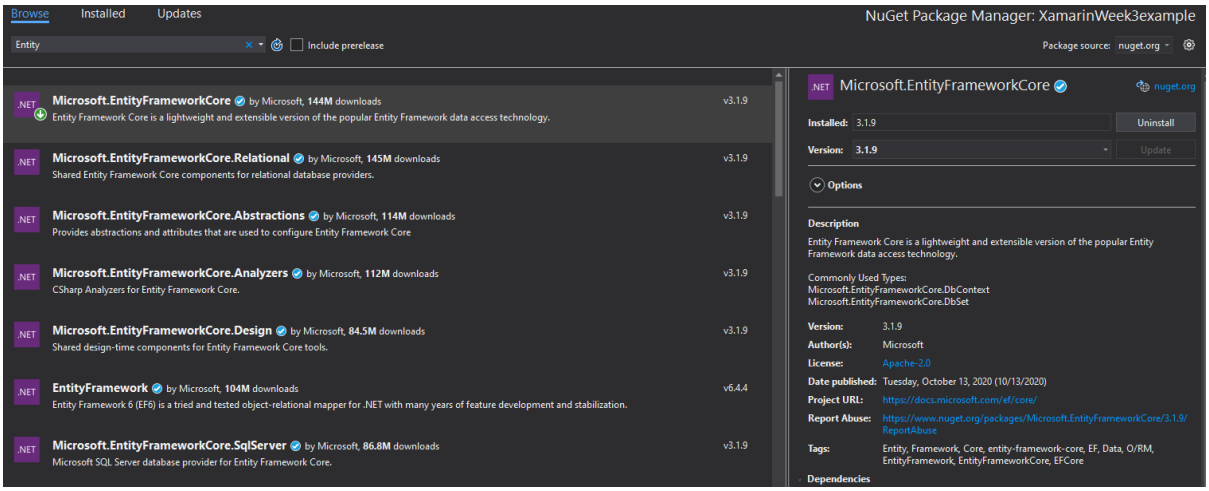
Opdracht B – Database connectie

Nu gaan we ervoor zorgen dat we de data ook kunnen opslaan, zodat we niet alles wat we invoeren kwijtraken als we de app afsluiten. Tip: simuleer hiervoor de server door in de client random temperatuurwaardes te laten genereren. Dit maakt het testen makkelijker.
Voer de volgende opdrachten uit:

1. Installeer EntityFrameworkCore. Doe dit door met de rechtermuisknop op je project te klikken in de solution explorer en dan “Manage NuGet packages” aan te klikken (zie afbeelding).



In het scherm dat je nu krijgt zoek je op EntityFrameworkCore, selecteer je EntityFrameworkCore en klik je op Install (zie afbeelding). Let op: Als je .NetStandard 2.0 gebruikt, moet je EntityFrameworkCore 3.1.9 of 3.1.10 gebruiken. Je kan de versie aan de rechterkant kiezen. Links van de button 'Install', kan je de versie aanpassen.



Je kunt na de installatie checken of EntityFrameworkCore geïnstalleerd is door "Installed" te kijken in de NuGet package manager.

2. Installeer nu op dezelfde manier EntityFrameworkCore.Sqlite. Let ook op dat je dezelfde versie van Sqlite kiest als je voor EntityFrameworkCore hebt gekozen
3. Nu gaan we een zogenaamde Database Context opzetten, waarin we gaan regelen welke informatie in de database moet worden opgeslagen. Doe dit door een nieuwe class aan je project toe te voegen, noem deze class DataContext (we willen straks de temperaturen van de server opslaan). Laat deze class erven van DbContext en zorg ervoor dat de using goed staat (naar Microsoft.EntityFrameworkCore).
4. Voeg aan de DataContext een DbSet<DataValue> genaamd DataValues toe, maak deze publiek en zorg voor een get en set.
5. Maak nu een constructor zonder parameters aan. Hierin wordt gekeken of de database ook echt bestaat. Voor iOS moet je ook nog een init uitvoeren. Voeg hiervoor de volgende code toe aan de contructor:

```
SQLitePCL.Batteries_V2.Init(); // initiate SQLite on iOS
this.Database.EnsureCreated();
```

6. Voeg nu de onderstaande methodes toe aan de net aangemaakte class DataContext. Hiermee Maak je het locale databasebestand aan en zeg je welk datatype verwacht kan worden <DataValue>.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<DataValue>();
    base.OnModelCreating(modelBuilder);
}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    string dbPath = Path.Combine(FileSystem.AppDataDirectory, "data.db3");
    optionsBuilder.UseSqlite($"Filename={dbPath}");
}
```

7. Voeg nu aan de class DataValue een property Timestamp toe. Deze is public en van de type DateTime. Zet in de regel boven de declaratie van deze property dit sleutelwoord: [Key] Let op dat je ook de haakjes gebruikt. Dit hebben we nodig voor het opslaan. De timestamp wordt zo de key waaronder we de temperatuur opslaan. Hierdoor zijn alle inputs uniek.
8. Set in de constructor de property Timestamp op de actuele tijd. Dit kan met DateTime.Now. Hierdoor wordt nu altijd als een nieuwe waarde wordt aangemaakt, de tijd van aanmaken opgeslagen.
9. Pas nu het user interface (XAML) aan zodat we de data vanuit de app kunnen opslaan. We willen de opgeslagen data ook kunnen verwijderen. Voeg hiervoor onder de bestaande button nog twee buttons toe en voorzie deze van een Clicked event.
10. Voeg ook een ListView aan de user interface (XAML) toe en noem deze dataCollectionView. Hierin gaan we de binnenkomende data laten zien (een timestamp en een temperatuur).
11. Ga nu naar de MainPage.cs. Voeg onderstaande methodes toe aan MainPage.cs en roep initDbView aan in de constructor van MainPage.

```
public void initDbView()
{
    using (var context = new DataContext())
    {
        dataCollectionView.ItemsSource = context.DataValues.ToList();
    }
}

async Task UpdateData(DataContext context, int value)
{
    await context.DataValues.AddAsync(new DataValue(value));
    await context.SaveChangesAsync();
}
```

12. Schrijf nu de code die je nodig hebt om de temperatuur op te slaan in de database. Deze regels moeten dus uitgevoerd worden als je op de opslabutton klikt. Gebruik hiervoor de volgende code.

```
using (var context = new DataContext())
{
    await UpdateData(context, [je temperatuur uit de XAML]);
    dataCollectionView.ItemsSource = context.DataValues.ToList();
}
```

Let op, je moet onClick methode nu wel async maken. Kan je uitleggen wat de code doet?

13. Je wilt de data ook nog kunnen verwijderen. Voeg volgende code toe aan het onclick event van de verwijder button. Denk er aan om ook deze methode weer async te maken.

```
using (var context = new DataContext ())
{
    context.RemoveRange(context.DataValues);
    await context.SaveChangesAsync();
    dataCollectionView.ItemsSource = context.DataValues.ToList();
}
```

Wat doet de code? En waarom gebruiken we ook hier ‘using'?

14. Test het opslaan nu met waardes die je van de server krijgt. Werkt alles zoals je hebt verwacht?
15. Optioneel: Kan je de UI mooier maken? Zet bijv. de buttons naast elkaar en geef de actuele waarde meer ruimte. Maak deze ook opvallender door ze groter te maken en bijv. een kleur of achtergrond te geven.