YRKESHÖGSKOLAN
NOVIA

Alarmapplikation för frivilliga brandkåren

Utveckling av en mobilapplikation i Xamarin.Forms

Rasmus Granberg

Examensarbete för ingenjörsexamen (YH) i teknik

Utbildningsprogrammet för informationsteknik

Vasa 2018

EXAMENSARBETE

Författare: Rasmus Granberg
Utbildning och ort: Informationsteknik, Vasa
Inriktningsalternativ: Informationsteknik
Handledare: Susanne Österholm

Titel: Alarmapplikation för frivilliga brandkåren

Datum 25.04.2018 Sidantal 30

Abstrakt

Detta examensarbete behandlar utvecklingen av en mobil alarmapplikation för frivilliga brandkårens medlemmar. Applikationen utvecklas på uppdrag av Black Label Bytes AB. Examensarbetet behandlar allt från skapandet av projektet och val av tekniker till utvecklingen av användargränssnittet samt funktioner.

Behovet av ett system för anmälning av tillgänglighet att delta vid utryckningar har länge konstaterats av frivilliga brandkårens medlemmar. Med nuvarande system har de inte på något smidigt sätt kunnat veta vem som har möjlighet att delta i utryckningen och när de eventuellt anländer till stationen. Det har i sin tur orsakat onödigt väntande för brandbilar.

Målet var att skapa en testversion av applikationen på plattformen Android med Azure som back-end. Tack vare att Xamarin.Forms användes är stora delar av applikationen även körbar på iOS och UWP.

Resultatet blev en körbar Android applikation som är redo att testas av frivilliga brandkårens medlemmar.

Språk: svenska Nyckelord: Xamarin, Azure, C#, XAML, Android

BACHELOR'S THESIS

Author: Rasmus Granberg
Degree Programme: Information Technology, Vaasa
Specialization: Information Technology
Supervisor: Susanne Österholm

Title: Alarm Application for the Volunteer Fire Department

Date 25.04.2018 Number of pages 30

Abstract

This thesis is based on the development of a mobile alarm application for the volunteer fire department. The application was made on the request of Black Label Bytes AB. The thesis contains every step of the development process, from creating the project and choices of tools to the development of the user interface and functions.

The need for an application to report availability to attend an emergency have been long-overdue. Using existing systems, members of the fire department have not been able to know who is going to participate in the emergency and when they might turn up to the station. As a result, emergency vehicles might spend unnecessary time waiting.

The goal was to create a test version of the application on the Android platform with Azure as a back-end. The fact that Xamarin.Forms was used meant that a large part of the application is also able to run on iOS and UWP.

The result was an executable Android application, which is ready to be tested by the members of the volunteer fire department.

Language: Swedish Key words: Xamarin, Azure, C#, XAML, Android

---

Innehållsförteckning

Ordförklaringar

Front-end – En applikation eller en webbsida som användaren kan interagera mot. (IT-mästaren, *Back-end & Front-end).*

Back-end – Till exempel en server eller databas som inte användaren kan se men som kommunicerar med front-enden. (IT-mästaren, *Back-end & Front-end).*

Active Directory – Katalogtjänst från Microsoft. Man skapar en skalbar, hanterbar och säker infrastruktur för användare och resurshantering. (Microsoft, *Översikt över Active Directory Domain Services)*.

OData v3 – Open Data Protocol är ett applikationsnivåprotokoll för hantering av data med RESTful webbtjänster. (OData, *Overview*).

SQL – Structured Query Language är ett standardiserat språk för lagring, hantering och hämtning av data från en databas. (W3schools, *SQL Tutorial*).

SDK – Software Development Kit. Utvecklingsverktyg för mjukvaruutvecklare. (Wikipedia, *Software Development Kit*).

Apache Cordova – Program för utveckling av plattformsoberoende mobilapplikationer i HTML, CSS & JS. (Cordova, *Home Page*).

Node.js – Kan skapa skalbara internetapplikationer, till exempel webbservrar. Skrivs i JavaScript och använder sig av händelsedrivet, asynkront I/O. (Node.js, *About*).

1 Inledning

I detta examensarbete behandlas utvecklingen av en alarmapplikation för frivilliga brandkåren. Syftet med applikationen var att frivilliga brandkårens medlemmar på ett effektivt sätt ska kunna kommunicera med varandra och få information om vem som deltar i uppdragen.

1.1 Uppdragsgivare

Applikationen utvecklades på uppdrag av företaget Black Label Bytes AB. Företaget grundades år 2016 av Magnus Sundell och erbjuder olika former av konsulteringstjänster samt designar, utvecklar och underhåller mjukvara för datorer och mobila lösningar.

1.2 Problembeskrivning

Behovet av denna applikation har konstaterats vid flera tillfällen av frivilliga brandkårens medlemmar. Försök med att skicka sms eller ha WhatsApp-grupper har inte fungerat speciellt smidigt. Exempelvis har det varit svårt att veta när deltagare kommer att anlända till stationen och om det är nödvändigt för brandbilarna att vänta på dem. Detta har i sin tur lett till ineffektiv tidsanvändning och onödigt väntande.

1.3 Syfte

Syftet med examensarbetet var att påbörja utvecklingen av en alarmapplikation för Android-mobiltelefoner, och om möjligt få ut en tidig testversion för att få feedback från testpersonerna. Via applikationen skall medlemmarna kunna anmäla om de är tillgängliga för ett uppdrag eller inte, samt även kunna meddela om de är försenade. Applikationen utvecklas så att det i framtiden enkelt kan implementeras mera funktionalitet, samt även så att den enkelt kan utvecklas för att användas på mobiltelefoner med andra operativsystem än Android.

2 Tekniker

I detta kapitel presenteras de olika teknikerna, verktygen samt ramverken som använts för att utveckla applikationen.

2.1 C#

I största delen av examensarbetet har programmeringsspråket C# (uttalas ”C sharp”) använts. C# har utvecklats av Microsoft och är ett objektorienterat språk som är en del av plattformen .NET Framework (uttalas ”dot net”). Språket är baserat på C++ och eftersom det hör till C-familjen finns likheter även med C men även med Java. (Microsoft, Intro to the C# Language and the .NET Framework).

2.2 XAML

I examensarbetet gjordes största delen av användargränssnittet i XAML. XAML (uttalas ”zammel”) står för Extensible Application Markup Language och är utvecklat av Microsoft. XAML är ett deklarativt programmeringsspråk baserat på XML och används för den visuella presentationen av en applikation. (Microsoft, What is XAML?).

2.3 Xamarin

Redan från början var det bestämt att applikationen skulle utvecklas med Xamarin, så inga andra alternativ behövde tas i beaktande.

Xamarin är ett företag som grundades 2011 av Miguel de Icaza och Nat Friedman. Företaget blev 2016 uppköpt av Microsoft. Xamarin erbjuder en egen plattform för mobilapplikations utveckling, Xamarin Platform, som består av bland annat Xamarin.Forms, Xamarin.iOS och Xamarin.Android. Xamarin har mer än 350 anställda, 15 000 kunder samt 1,4 miljoner användare i 120 länder. Det som gör Xamarin speciellt unik är att applikationerna utvecklas nästan enbart i programmeringsspråket C#, vilket gör att användare inte behöver lära sig flera olika språk för att kunna utveckla en komplett mobilapplikation. (Xamarin, About).

2.3.1 Xamarin.Forms

Xamarin.Forms erbjuder utvecklaren ett enkelt sätt att utveckla mobilapplikationer som fungerar på Android, iOS samt Windows, det vill säga, som är plattformsoberoende. Xamarin.Forms lämpar sig bäst för applikationer som endast kräver lite plattformsspecifik funktionalitet och där fokus ligger på att använda en så stor del av koden som möjligt till alla plattformar. Med Xamarin.Forms utvecklar man applikationer som efter kompilering ger kod som är direkt körbar (native) på alla system med rätt sorts processor och operativsystem, och behöver därför ingen särskild exekveringsmiljö. Xamarin.Forms erbjuder även ett plattformsspecifikt användargränssnitt (native UI), vilket gör att applikationen fungerar smidigare samt att den kan utnyttja operativsystemets egna funktionaliteter. (Xamarin, Forms).

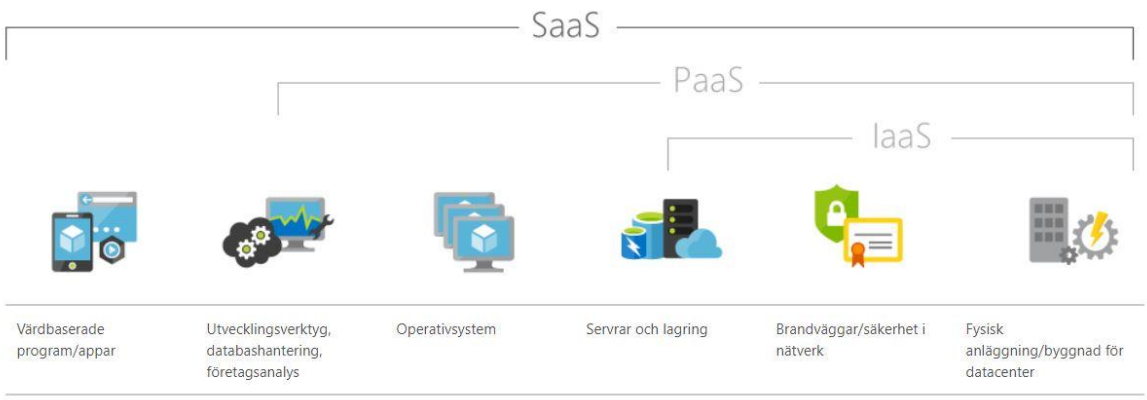
2.3.2 Xamarin.Android & Xamarin.iOS

Xamarin.Android och Xamarin.iOS är till skillnad från Xamarin.Forms plattformsspecifika. Dessa lämpar sig bäst när det behövs mycket plattformsspecifik funktionalitet och där det viktiga inte är att dela koden utan att få utveckla skräddarsydda användargränssnitt. (Xamarin, Forms).

2.4 Microsoft Azure

Microsoft Azure valdes som molntjänst eftersom det redan använts av uppdragsgivaren. Azure erbjuder även flera användbara tjänster för mobilapplikationer som passar bra till detta examensarbete.

Azure, som ägs av Microsoft, är en plattform i molnet som erbjuder mer än 100 tjänster och verktyg för allt från utveckling och testning av applikationer till distribution samt underhåll. Azure har stöd för många olika operativsystem, databaser, ramverk, programmeringsspråk samt enheter och erbjuder SaaS (Software as a service), PaaS (Platform as a service), och IaaS (Infrastructure as a service). Enligt Microsoft själv är Azure det enda enhetliga hybridmolnet på marknaden. (Microsoft, What is Azure).



Figur 1. SaaS, PaaS och Iaas exempel (Microsoft, Vad är SaaS).

2.4.1 Azure Mobile Apps

Azure Mobile Apps är en del av Azure App Service, och är en plattform som ger hög skalbarhet samt global tillgänglighet vid utveckling av mobilapplikationer. Mobile Apps har många viktiga funktionaliteter för molnkompatibel mobilutveckling:

- Autentisering och auktorisering:

  Ger stöd för identitetsleverantörer, till exempel via Azure Active Directory för företagsautentisering men även autentisering via sociala nätverk så som Twitter, Google och Facebook.

- Dataåtkomst:

  Mobile Apps erbjuder OData v3-datakälla som antingen kopplas till en lokal SQL Server eller till Azure SQL Database. Tjänsten byggs på Entity Framework och kan därför lätt integreras med andra NoSQL och SQL dataleverantörer, till exempel Azure Table Storage, MongoDB med flera.

- Klient-SDK:

  En komplett uppsättning klient-SDK:er finns för plattformsspecifik utveckling, det vill säga iOS, Android, samt Windows. För utveckling av plattformsoberoende applikationer finns Xamarin.iOS, Xamarin.Android, Xamarin.Forms samt hybridprogramutveckling med Apache Cordova. Klient-SDK:erna har öppen källkod samt fås med MIT-licens.

- Offlinesynkronisering:

  Med klient-SDK:erna byggs mobilapplikationer som kan spara data lokalt för att sedan ladda upp den till molnlagringen. Datamängden kan synkroniseras automatiskt med data på serverdelen.

- Push-meddelanden:

  Klient-SDK:erna integreras med registreringsfunktionerna som finns i Azure Notification Hubs, vilket innebär att push-meddelanden kan skickas till miljontals mottagare samtidigt.

(Microsoft, Om Mobile Apps i Azure App Service).

2.4.2 Easy Tables

Azure erbjuder en funktion som heter Easy Tables som i praktiken skapar en back-end för datalagring utan att utvecklaren behöver skriva en enda rad kod. Många mobilapplikationer behöver ha en back-end att kommunicera med för att till exempel hämta samt ladda upp data till. Azure Mobile Apps funktion Easy Tables ger möjligheten att skapa en back-end med Node.js som är perfekt för att lagra samt hämta mindre mängder av data, till exempel för en inköpslista eller som i detta examensarbete, en enkel lista över medlemmars status. Listan diskuteras närmare i kapitel 5.3. Back-enden kan även konfigureras med C# som programmeringsspråk.

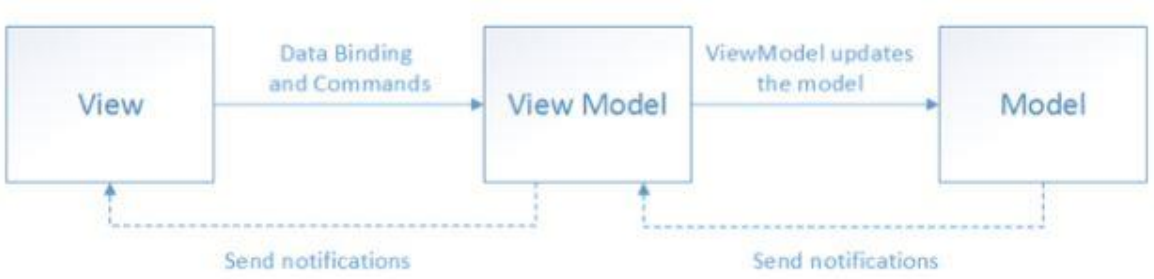
Azure erbjuder en guide på Easy Tables vilket gör att man kan ansluta en back-end till projektet på några minuter.

2.5 Visual Studio 2017

Visual Studio är en avancerad programutvecklingsmiljö som är mycket populär speciellt vid utveckling av C# applikationer. I Visual Studio finns Xamarin som ett tillägg vilket gör att utvecklingen av mobilapplikationer går mycket smidig. Därför användes Visual Studio 2017 som utvecklingsmiljö i detta examensarbete men också för att den var bekant från tidigare. (Microsoft, What is New in Visual Studio 2017).

2.6 MVVM

I projektet implementerades arkitekturmönstret Model-View-ViewModel (MVVM) där det var logiskt och behövligt. När Xamarin.Forms används utvecklas oftast användargränssnittet i XAML och koden som sköter händelser i C#. Vartefter applikationerna växer i storlek blir det svårare att underhålla koden och därför används MVVM för att kunna separera affärs- och presentationslogiken från dess användargränssnitt. Detta innebär att koden blir lättare att förstå, underhålla, testa samt utveckla. Utvecklare kan även på ett enklare sätt återanvända koden och det underlättar även samarbetet mellan applikations- och användargränssnittsutvecklarna.



Figur 2. Delarna i MVVM och deras sammankoppling (Microsoft, MVVM).

Figur 2 visar de olika delarna i MVVM och hur de är sammankopplade. Förenklat kan sägas att View känner till View Model, View Model känner i sin tur till Model, men Model känner inte till View Model som i sin tur inte känner till View.

I fortsättningen används vy som hänvisning till View. Vyn är den del där användargränssnittet finns, vilket ofta innebär en sida när det gäller Xamarin.Forms projekt.

ViewModel kommer i fortsättningen att benämnas vymodell. Vymodellen implementerar egenskaper och kommandon som vyn kan bindas ihop med, samt meddelar om eventuella förändringar genom till exempel en PropertyChanged händelse. Vyn får alltså information om vilken data som ska användas men bestämmer själv hur det skall presenteras. Vymodellen har även ansvaret för interaktionen mellan vyn och de modellklasser som behövs. Till exempel kan vymodellen tillåta vyn att direkt ha tillgång till modellklasserna och på så sätt kan kontrollerna i vyn binda data till klasserna direkt. (Microsoft, MVVM).

Model kommer i fortsättningen att benämnas modell. Modellen innehåller applikationens data och kan även innehålla validering av data för att säkerställa att den är av rätt typ samt form (Microsoft, *The Model Class)*.

3 Planering

I detta kapitel beskrivs kravspecifikationer, planering, hur olika verktyg använts samt hur applikationen är uppbyggd.

3.1 Specifikationer

Eftersom det inte fanns någon befintlig kod utan allt skulle utvecklas från grunden, gjordes specifikationerna utgående från hur applikationen skulle fungera färdigt utvecklad. Kraven delades in i tre olika prioriteringsnivåer där första nivån var sånt som behövdes för att applikationen skulle gå att använda, andra nivån var sånt som underlättade användningen och den tredje var extra funktionalitet som kunde utvecklas i ett senare skede. Eftersom det var svårt att kunna bedöma hur lång tid som utvecklingen av alla olika delar skulle kräva sattes inget specifikt mål upp. I stora drag var målet att få en applikation med de mest grundläggande funktionerna som några testanvändare kunde få använda och se hur det fungerar.

Till de mest grundläggande funktionerna hör framför allt anmälningsfunktionen. Som nämndes i kapitel 2.2 har största problemet varit att vid ett larm veta ifall medlemmar har möjlighet att delta i uppdraget samt när de är på kommande till stationen.

Ett krav för att kunna använda denna applikation är att frivilliga brandkårens medlemmar har en smarttelefon. För att säkerställa att de flesta skall kunna använda applikationen utvecklades den i Xamarin.Forms för att största möjliga del av koden skall kunna användas på både Android och iOS. Även om man genom att använda Xamarin.Forms får mycket kod som direkt går att använda på både Android och iOS skulle det kräva mera tid samt gärna flera utvecklare för att få applikationen att fungera tillfredsställande på båda plattformarna. Därför gjordes examensarbetet huvudsakligen för Android, men tack vare Xamarin.Forms fungerar stora delar av koden även på iOS vilket underlättar i framtiden.

3.2 Användargränssnitt

Innan arbetet inleddes diskuterades idéer och funktionalitet med uppdragsgivaren. Brandstationen i Korsholm besöktes också för att få en bättre bild av hur befintliga systemet fungerar och vilka funktioner som behövs i applikationen.

Eftersom applikationen utvecklas från grunden måste även ett användargränssnitt utvecklas. I figur 3 visas en idé på hur anmälningsfunktionen kunde se ut. Användaren är först oanmäld och anmäler sig genom att välja en av tre ikoner som föreställer ”deltar”, ”deltar inte” samt ”försenad”.

Figur 3. Skiss för sidorna ”Status” och ”Uppdrag”.

3.3 Förberedelse

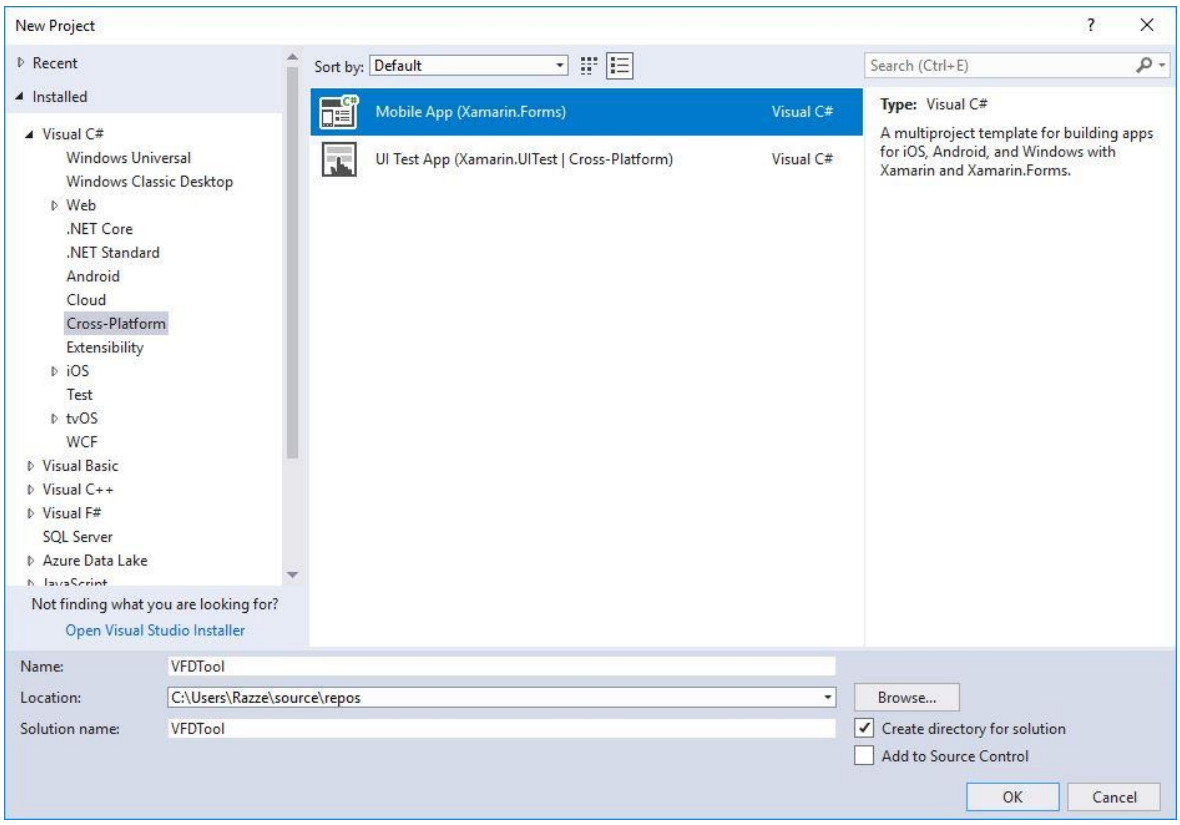
När projektet var planerat så att arbetet kunde påbörjas var första steget att undersöka hur Xamarin fungerar. Testapplikationer och exempelprojekt laddades ner, testkördes och koden studerades. Xamarin har en egen sida för mobilapplikationsutvecklare på adressen www.university.xamarin.com, där det fanns mycket material med allt från hur man skapar ett nytt Xamarin projekt till de mest avancerade funktionerna. På grund av att Xamarin är en relativt ny plattform utvecklas den snabbt, vilket gör att dokumentationen i många fall inte hålls uppdaterad. Detta gjorde det i sin tur svårt att i vissa fall hitta kod och funktioner som fungerar, vilket följaktligen resulterade i mycket extra sökande och problem genom hela arbetet.

4 Utveckling

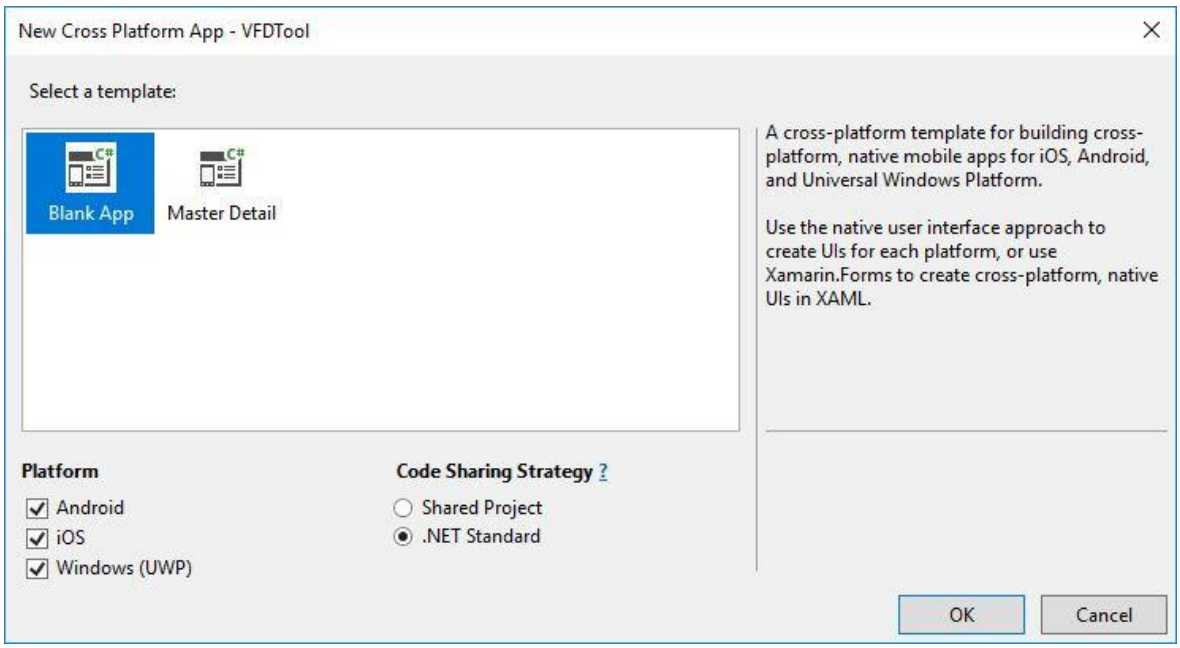
I detta kapitel beskrivs utvecklingen av applikationen, från gränssnittet till de olika funktionerna som implementerades. Användningen och val av funktioner och tillvägagångssätt motiveras, förklaras och visas.

4.1 Skapande av projektet

Projektet skapades i Visual Studio 2017 där många olika valmöjligheter finns. Naturligtvis skapades ett nytt plattformsospecifikt Xamarin-mobilapplikationsprojekt (Cross-Platform) som kan ses i Figur 4. Vidare fanns det möjlighet att välja mellan en Blank Applikation (Blank App) och en applikation med en del färdig kod. Till sist fanns möjligheterna att välja vilka plattformar som skulle stödas och vilken kod-delningsstrategi man ville ha (Figur 5).



Figur 4. Valmöjligheter vid skapande av nytt projekt i Visual Studio 2017.



Figur 5. Valmöjligheter gällande plattformar och kod-delning.

En blank applikation valdes för att själv få bestämma hur man bygger upp användargränssnittet och både Android samt iOS valdes som plattformar men även UWP (Universal Windows Platform) ifall behovet av en Windows applikation skulle visa sig i framtiden. För kod-delningen fanns tre alternativ:

- Shared Projects:

  Shared Projects (Delade projekt) är den enklaste formen när det gäller att dela kod, men blir snabbt ineffektiv att använda vid större projekt bland annat på grund av svårigheter att se vilken del av koden som verkligen körs på vilken enhet. Man är även tvungen att inkludera kompileringsdirektiv i koden beroende på vilka plattformar man vill stöda (figur 6).
  (Pedley, 2016, *Using Shared Projects*).

```
#if __ANDROID__
path = "android";
#else

#if __IOS__
path = "iOS";
#else
```

Figur 6. Exempel på kompileringsdirektiv.

- Portable Class Library (PCL)

  Portable Class Library (Portabelt klassbibliotek) är vanligare att använda än Shared Projects när det gäller delbar kod, speciellt vid projekt där man vill dela koden med andra utvecklare. Vid användning av PCL behövs inga kompileringsdirektiv för att

stöda olika plattformar. Istället väljs plattformar genom att välja vilken standardversion som skall stödas. (Pedley, 2016, *Portable Class Libraries*).

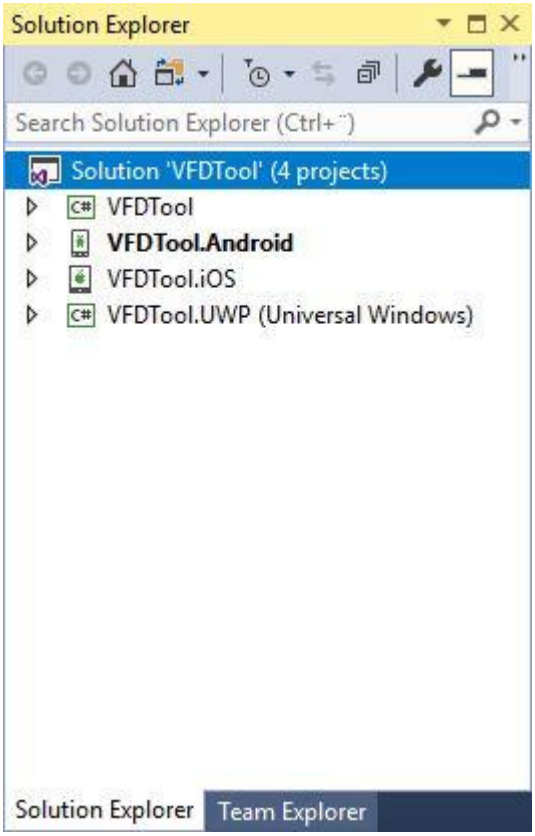
- Standard Class Libraries (SCL)

  Standard Class Libraries (.NET Standard) är liknande som PCL. Den största skillnaden är ett större utbud av funktioner gentemot PCL. SCL ersätter PCL sedan slutet av 2017. (Pedly, 2016, *Using .NET Standard Class Libraries)*.

I examensarbetet användes PCL vilket inte finns som ett alternativ i figur 5 på grund av att SCL har börjat ersätta PCL under tiden som applikationen har utvecklats. När projektet ursprungligen skapades hade redan SCL lanserats men var inte inkluderad i den Xamarin.Forms-version som vid tillfället var den senaste. Som nämndes i kapitel 4.3 har det funnits en del ouppdaterad info vilket innebar att SCL inte rekommenderades ens i Microsofts eller Xamarins egna dokument.

När man valt projekttyp skapas ett huvudprojekt samt underprojekt enligt vilka plattformar som valts, vilka går att kompilera och köra på Android, iOS respektive UWP.

I figur 7 kan man se de olika projekt som skapats. VFDTool, PCL-projektet, innehåller den delbara koden som körs på alla plattformar. VFDTool.Android innehåller kod som är plattformsspecifikt för Android och som inte går att köra på andra plattformar. VFDTool.iOS är i sin tur för plattformsspecifikt innehåll för iOS och VFDTool.UWP för UWP. Figur 8 illustrerar hur det ser ut om man bygger och kör projektet på en Android emulator.



Figur 7. Projektets delar.

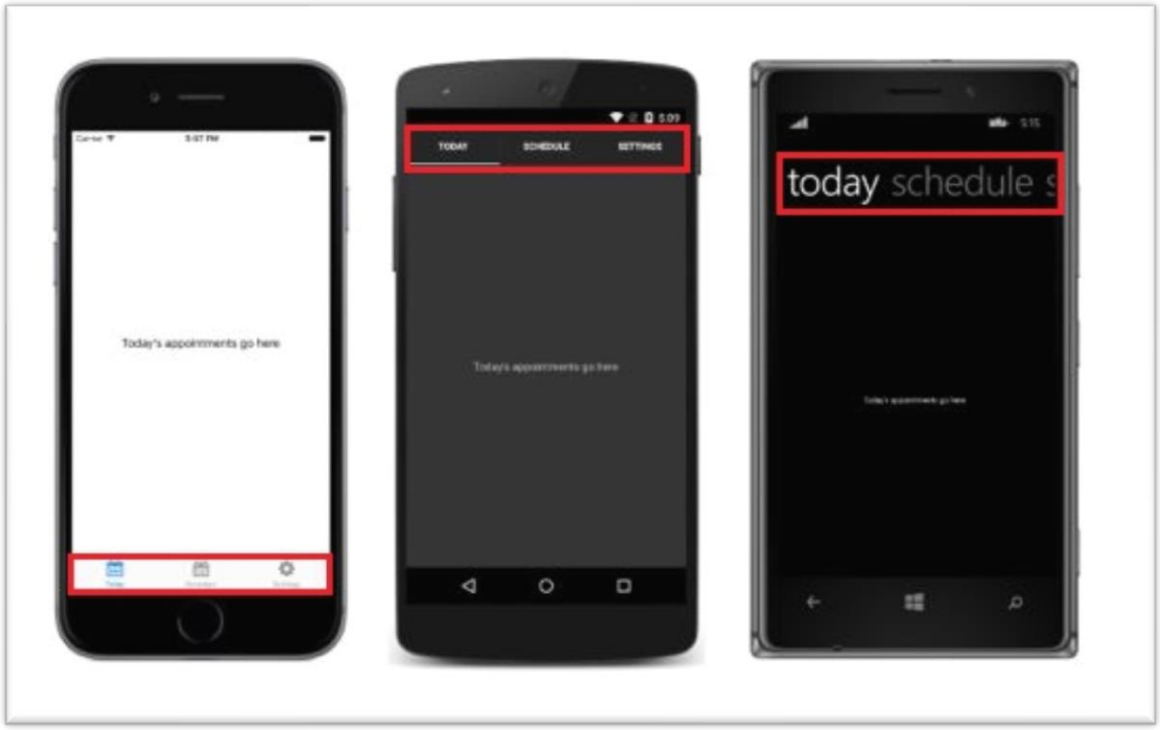
Figur 8. Projektet körs på en Android emulator.

4.2 Användargränssnitt

Fördelen med att använda Xamarin.Forms är att man får plattformsspecifikt utseende på applikationen oberoende om man kör Android eller iOS. Som nämndes i kapitel 4.1 blev det beslutat att utveckla applikationen för Android först men vissa skillnader i utseendet beroende på plattform kommer ändå att presenteras.

4.2.1 Navigation

Gällande navigation i Xamarin finns många olika valmöjligheter. Valet föll på något som kallas TabbedPage, det vill säga navigationen sker med flikar. Som tidigare nämnts kommer samma kod att se olika ut beroende på plattform, vilket visas i figur 9.



Figur 9. TabbedPage utseende på iOS, Android samt Windows Phone (Microsoft, Tabbed Page).

Skillnaderna på de olika plattformarna är att iOS har sina flikar i nedre kanten medan Android samt Windows Phone har flikarna i övre kanten. Flikarna beter sig även lite olika beroende hur många flikar som finns. Till exempel i iOS skapas en extra knapp som beter sig som en hamburgermeny dit resterande flikar flyttas medan i Android läggs alla flikar bredvid varandra och användaren kan dra flikarna till sidorna.

För att skapa ett användargränssnitt kan XAML och/eller C# användas. I detta arbete har XAML använts för användargränssnittet samt C# för koden som sköter funktionalitet och dylikt. Kodexempel 1 visar XAML-koden för att skapa en knapp. Kodexempel 2 visar den bakomliggande C#-koden som bestämmer knappens funktionalitet. I detta fall får användaren en ruta som bekräftar att knapptryckningen har registrerats.

Kodexempel 1. Uppdragssidans XAML-kod.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="VFDTool.MissionPage"
             x:Name="root"
             xmlns:ViewModels="clr-namespace:VFDTool.ViewModel"
             Title="Uppdrag"
             Icon="ic_warning.png">
    <ContentPage.Content>
        <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
            <Button Text="Submit" Clicked="SubmitButton_Clicked"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Kodexempel 2. Uppdragssidans C#-kod.

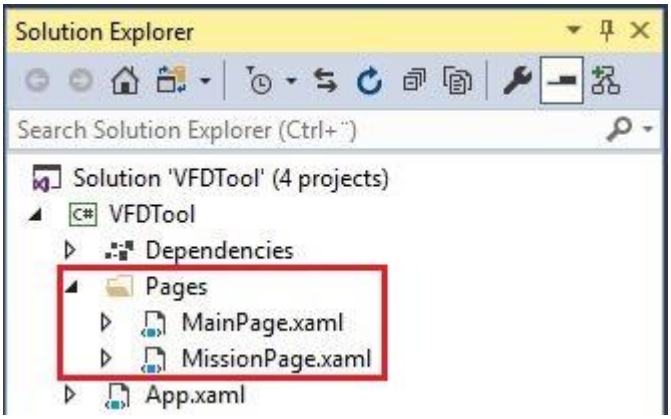
```
namespace VFDTool.Pages
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    - references
    public partial class MissionPage : ContentPage
    {
        - references
        public MissionPage ()
        {
            InitializeComponent ();
        }

        - references
        public void SubmitButton_Clicked(object sender, EventArgs e)
        {
            DisplayAlert("Godkänn", "Knapptryckning registrerad", "OK");
        }
    }
}
```

4.2.2 Sidor

I Xamarin.Forms finns en klass Page (sida) som fungerar som basklass till flera olika underklasser för olika typer av Pages som används enligt behov. I denna applikation används både ContentPage och även TabbedPage, som nämndes i kapitel 5.2.1. Varje sida är en ContentPage men presenteras sedan i en TabbedPage för att på ett smidigt sätt kunna navigera mellan sidorna. ContentPage är den vanligaste samt enklaste typen av sidor som endast presenterar ett innehåll medan TabbedPage använder sig av flikar för att navigera mellan flera olika undersidor. Figur 10 visar olika sid-typer.

Figur 10. Olika typer av sidor (Microsoft, Pages).

Sidor är plattformsospecifik kod vilket innebär att sidorna implementeras i PCL-projektet och därför skrivs koden endast en gång för att kunna köra på alla plattformar som projektet stöder. På så sätt får man utvecklat ett användargränssnitt till flera plattformar med samma kod.



Figur 11. Sidorna implementeras i PCL-projektet och kan köras på alla plattformar.

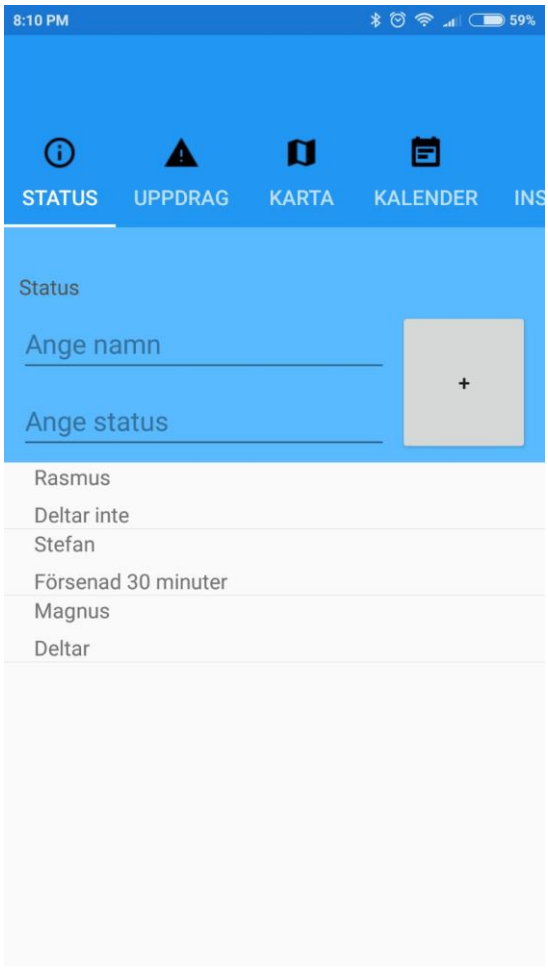
En TabbedPage implementeras som en vanlig sida men deklareras som en TabbedPage istället för ContentPage som kan ses i kodexempel 3

Kodexempel 3. TabbedPage deklaration i XAML-kod.

```
2   <TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
3               xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4               xmlns:local="clr-namespace:VFDTool"
5               xmlns:localAppointment="clr-namespace:VFDTool.Pages.AppointmentPage.Views"
6               x:Class="VFDTool.MainPage">
7
8   ...
9
10  </TabbedPage>
```

4.3 Anmälningsfunktionen

Anmälningsfunktionen är en av de största orsakerna till att denna applikation behövs. Ett smidigt sätt att få en medlem i frivilliga brandkåren att anmäla att han deltar i ett uppdrag, gör första delen av utryckningen mycket smidigare. I detta arbete gjordes en testversion som är en lista (figur 12). Användaren fyller manuellt i sitt namn och sin status, det vill säga ”deltar”, ”deltar inte” eller ”försenad”. Listan är kopplad till Azures Easy Tables, vilket gör att listan synkroniseras automatiskt med molnet och användaren behöver bara dra ner listan för att den skall uppdatera och hämta den senaste datan från databasen.

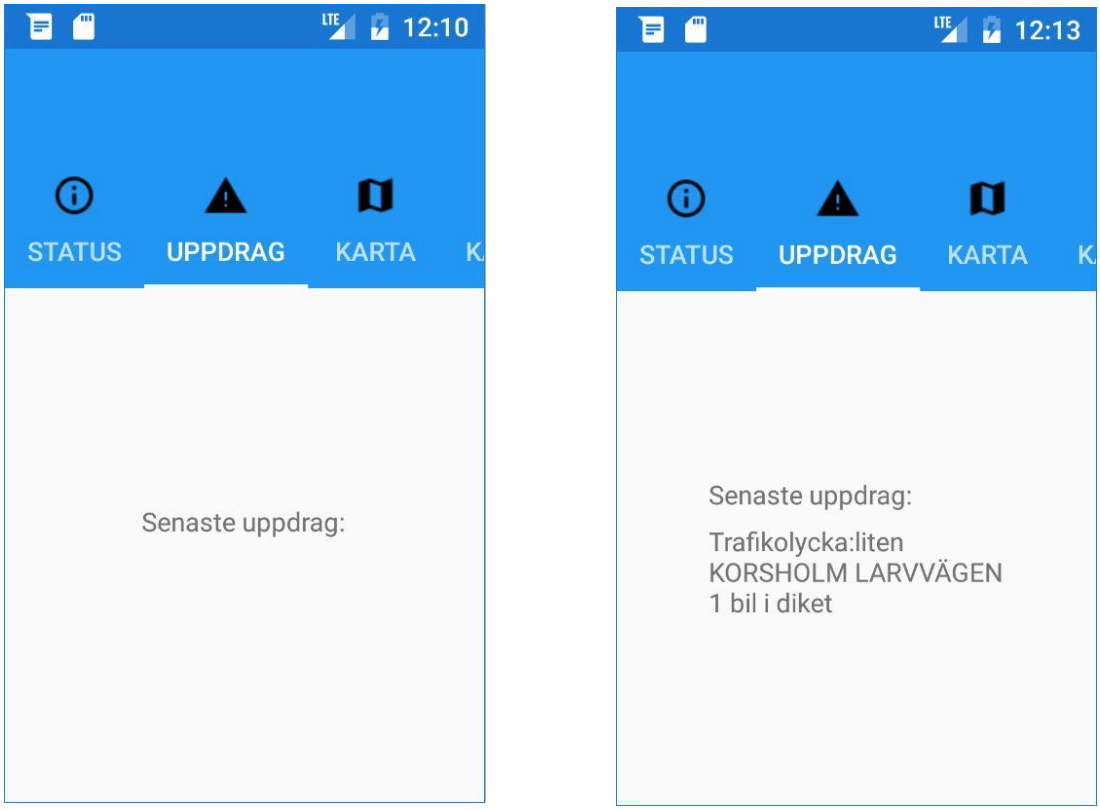


Figur 12. Anmälningslista som synkroniserar med Azure.

4.4 Uppdragssidan

Alla alarm från nödcentralen kommer med sms med bland annat beskrivning av uppdraget och platsen för uppdraget. På uppdragssidan i applikationen visas inkommande sms från alarmcentralen för att inte användarna skall behöva byta mellan applikationen och sms inkorgen.

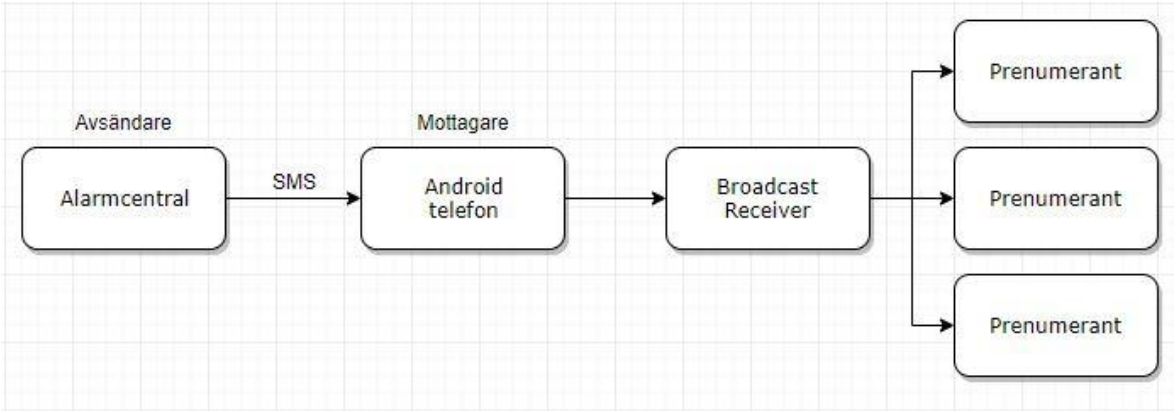
I detta kapitel kommer funktionen för mottagande av sms samt funktionen för att skicka det mottagna meddelandet som en textsträng från Android-projektet till PCL-projektet presenteras. Funktionerna kommer att förklaras och nerskalad kod kommer att visas. I figur 13 visas hur det ser ut före och efter ett sms från nödcentralen tagits emot.



Figur 13. Före och efter mottagandet av SMS från alarmcentralen.

4.4.1 BroadcastReceiver

I Xamarin.Android finns en komponent BroadcastReceiver, fritt översatt till svenska blir det sändningsmottagare, som tillåter en applikation att reagera på ett meddelande som skickats antingen av operativsystemet, alltså Android, eller av en applikation. Broadcasten använder systemet att publicera – prenumerera. En händelse (event) orsakar en broadcast att publiceras som i sin tur tas emot av de komponenter som är prenumeranter på händelsen. Själva meddelandet som Android sänder ut kallas ”Intent”. I denna applikation är meddelandet som skickas ett sms från nödcentralen. I figur 14 visas ett förenklat exempel på flödet. (Microsoft, Broadcast Receivers in Xamarin.Android).



Figur 14. Förenklat exempel över BroadcastReceiver och prenumeranter.

På grund av att komponenten BroadcastReceiver är Android-specifik, är det inte möjligt att använda samma komponent för iOS. Istället måste man använda sig av komponenter med liknande funktionalitet som BroadcastReceiver, men som är iOS-specifikt. Det innebär att när man i vanliga fall skriver den plattformsospecifika koden i PCL-projektet måste koden för BroadcastReceivern skrivas i Android-projektet.

BroadcastReceiverns uppgift i detta examensarbete är som bekant att ta emot sms och därför skapades en ny klass SmsReceiver.cs i Android-projektet som i sin tur ärver från BroadcastReceiver.

I Android finns det olika Intents, i det här fallet har en Intent för att läsa inkommande sms använts och är deklarerad som ”IntentAction” i kodexempel 4. Där ser man också att det finns en funktion OnReceive som i sin tur granskar om meddelandet som kommit från Android eller en annan applikation, är ett sms. Ifall meddelandet är ett sms skapas en array som sparar meddelandet samt avsändaren. Eftersom klassen SmsReceiver kommer att köras varje gång ett sms tas emot blir följande steg att kontrollera från vilket nummer som sms:et kommer. Alla andra nummer än från nödcentralen ignoreras. Meddelandet omvandlas till en textsträng och skickas vidare till PCL-projektet med MessagingCenter, som kommer att behandlas i kapitel 5.4.2.

Kodexempel 4. Liten del av klassen SmsReceiver.

```
29      private const string IntentAction = "android.provider.Telephony.SMS_RECEIVED";
        0 references
30      public override void OnReceive(Context context, Intent intent)
31      {
32
33          // read incoming SMS
34          if (intent.Action == IntentAction)
35          {
36              SmsMessage[] messages = Telephony.Sms.Intents.GetMessagesFromIntent(intent);
37
38              var sb = new StringBuilder();
39              var sbBody = new StringBuilder();
51              if (sender == "112")
52              {
53                  this.ParsedSms = SmsParser.ParseSms(sbBody.ToString());
54                  string SmsBody = sbBody.ToString();
55                  MessagingCenter.Send<Object, string>(this, "SmsReceived", SmsBody);
56              }
```

Applikationer som används på nyare versioner av Android måste få tillstånd av användaren att använda sig av funktioner så som att använda kameran, spela in ljud och så vidare. I denna applikation behöver användarens tillstånd att läsa inkommande sms. Förut räckte det med att utvecklare lade till tillståndet i filen AndroidManifest, men på grund av utökad säkerhet behövs nu extra tillstånd. Denna funktion läggs till i Androids MainActivity-fil och kontrollerar varje gång applikationen startas ifall användaren har gett tillstånd eller inte. Ifall tillstånd inte ges startar fortfarande applikationen men utan funktionen att läsa sms. För iOS behövs tillstånd från användaren även på äldre versioner av operativsystemet.

4.4.2 MessagingCenter

Xamarin.Forms fungerar så att all kod som skrivs i PCL-projektet kan ärvas av de plattformsspecifika projekten, men i allmänhet fungerar det inte andra vägen. Eftersom funktionen BroadcastReceiver är Android-specifik deklarerades klassen SmsReceiver i Xamarin.Android-projektet. För att kunna visa det inkomna sms:et i PCL-projektet behövdes ett sätt att skicka textsträngen med sms:et från Android-projektet till PCL-projektet. Detta löstes med en funktion som heter MessagingCenter.

MessagingCenter är en funktion som tillåter till exempel vymodeller och andra komponenter att kommunicera med varandra även om de inte har tillgång till varandras data i vanliga fall. MessagingCenter är uppbyggd av två delar, en prenumerationsdel och en sändningsdel. Prenumerationsdelen väntar på ett meddelande från en specifik signatur och gör något när de fått meddelandet. Sändningsdelen sänder meddelandet till alla som prenumererar på just

den delen. Ifall ingen prenumerant finns ignoreras meddelandet. (Microsoft, MessagingCenter).

Eftersom MessagingCenter inte är plattformsspecifik kan flera plattformar använda sig av samma system för att skicka meddelanden till PCL-projektet även om en annan funktion än BroadcastReceiver måste användas för att ta emot sms:en.

Kodexempel 4 i föregående underkapitel visar hur MessagingCenters sändningsdel blir implementerad i klassen SmsReceiver. Delen som prenumererar finns i PCL-projektets vymodell för sidan som kommer att visa sms:et och kan ses i kodexempel 5. Sms:et tas emot och skickas vidare till en funktion SmsDisplayer som i sin tur initialiserar en property som är ihopbunden med XAML-koden för MissionPage-sidan.

Kodexempel 5. MessagingCenter tar emot sms:et.

```
30      public MissionPageViewModel()
31      {
32          // Receive the SMS
33          MessagingCenter.Subscribe<Object, string>(this, "SmsReceived", (sender, arg) =>
34          {
35              SmsDisplayer(arg);
36          });
37      }
38
        1 reference
39      public void SmsDisplayer(string message)
40      {
41          this.ShowMessage = message;
42      }
```

4.5 Kalender

En kalender är något som behövs vid många olika tillfällen. Att ha en kalender i applikationen betyder att medlemmarna har möjlighet att använd en skild kalender för brandkårsrelaterade uppgifter. Tanken bakom kalendern är att kunna ha en gemensam kalender för alla medlemmar för att på ett smidigt sätt kunna se till exempel vem som har dejour, när gemensamma övningar och möten är inplanerade med mera. I detta examensarbete implementeras en kalender som endast körs mot en lokal databas.

Att utveckla en kalender som ensam utvecklare skulle vara en mycket tidskrävande uppgift och därför söktes andra lösningar. Valet föll på en kalender från Syncfusion.

Syncfusion är ett företag med mer än 13 000 kunder och mer än en miljon användare. De erbjuder över 800 olika kontroller och ramverk för webb-, mobil- samt desktoputveckling. Företaget erbjuder sina produkter kostnadsfritt för företag med en omsättning på mindre än

en miljon dollar och färre än 5 anställda utvecklare. Black Label Bytes finns inom dessa gränser och därför valde vi att pröva på deras produkter. (Syncfusion, About Us).

Deras kalender är mycket enkel att implementera. Ett NuGet-paket läggs till i alla projekt som skall använda komponenten och sedan behövs endast en deklaration i till exempel XAML-koden, se kodexempel 6.

Kodexempel 6. Del av kalenderns deklaration i XAML.

```
18      <schedule:SfSchedule WidthRequest="600"
19                           x:Name="schedule"
20                           FirstDayOfWeek="2"
21                           ScheduleView="MonthView"
22                           DataSource="{Binding ListOfMeeting}"
23                           HorizontalOptions="FillAndExpand"
24                           ShowAppointmentsInline="True"
25                           VerticalOptions="FillAndExpand">
```

4.6 Inloggning

Kravet på inloggning har för tillfället inte en väldigt hög prioritet men kommer att spela en viktigare roll i framtiden när mera avancerade funktioner implementeras. Exempelvis kan man via inloggning ge användare vissa rättigheter som behövs när en mera omfattande delad kalenderfunktion implementeras. Detta för att förhindra att alla har möjlighet att lägga till i kalendern och ta bort ur kalendern. Istället har bara en utsedd planerare rättigheten att göra ändringar i kalendern. Via inloggning kan man också autentisera användare för att förhindra utomstående att få tillgång till data som inte är ämnad för allmänheten. Eftersom det inte var möjligt att utveckla så avancerade funktioner i detta examensarbete, gjordes endast en inloggningsmöjlighet via Facebook.

Att ha inloggning via Facebook är ett smidigt alternativ eftersom de flesta har ett Facebook-konto och man kan implementera mera avancerad autentisering i framtiden vid behov. Facebook stöder inloggning direkt i applikationen utan att behöva gå via en webbläsare och ifall användaren har Facebooks applikation installerad på sin telefon används den för att logga in vilket gör processen ännu smidigare.

För att få tillåtelse att använda Facebooks inloggningsfunktion måste applikationen registreras på https://developers.facebook.com. Efter det får man genom att lägga till ett NuGet-paket tillgång till Facebooks officiella SDK, det vill säga deras utvecklingsverktyg för tredjepartsutvecklare.

Som första steg implementerades en knapp i XAML-koden samt en klass för rendering av knappen. Renderingsklassen har hand om att rendera Facebook-knappen enligt om en användare är in- eller utloggad. I denna applikation kontrollerar koden först om användaren är inloggad från förr. Om inte skickas användaren till inloggningssidan där Facebooks inloggningsknapp är synlig. När inloggningen är klar skickas användaren till huvudsidan och knappen som finns på inställningssidan renderas automatiskt som en utloggningsknapp istället. Exempel på detta kan ses i figur 15.

Figur 15. Facebooks automatiskt renderande in- och utloggningsknappar.

5 Framtidsplaner

I detta kapitel diskuteras slutligen framtidsplaner. Vissa av planerna har redan börjat förberedas för och andra är bara på planeringsstadiet ännu.

5.1 Plattformar

För att så många som möjligt skall kunna använda applikationen måste även stöd för iOS implementeras. Det som behöver göras är att skriva koden för alla plattformsspecifika funktioner, så som att ta emot sms. Användargränssnittet är som diskuterat redo att användas på alla plattformar.

5.2 Karta och vägbeskrivning

En karta skall implementeras för att kunna förse användare med vägbeskrivning till olycksplatsen. Detta är speciellt praktiskt ifall förstärkning tillkallas och den behöver ta sig till olycksplatsen på egen hand.

Google Maps går att inkludera i ett Xamarin.Forms-projekt och ger en karttjänst som fungerar och som många är bekanta med. Däremot är det en hel del arbete att få tjänsten

inkluderad i projektet. Därför har även en karttjänst kallad Leaflet undersökts som alternativ karttjänst. Black Label Bytes har en egen server för Leaflet där några funktioner har testats och konstaterats att tjänsten skulle kunna passa i detta projekt ifall implementeringen lyckas.

För att kartan automatiskt skall visa vägbeskrivningen behövs en funktion som läser av adressen som inkommit i sms:et från nödcentralen. Ett problem är att sms:en inte alltid ser identiska ut vilket gör att en automatiserad avläsning behöver kodas för alla olika sms modeller som kan användas. En testklass skapades i detta arbete för att läsa in sms:et samt dela upp det enligt kommatecken, snedstreck och liknande. Klassen presenteras inte i detta examensarbete på grund av att den inte kommer att användas i första versionen av applikationen utan endast förbereddes för kommande utveckling.

5.3 Inloggningsalternativ

För att inte behöva förlita sig på eller tvinga användare att ha ett Facebook-konto behövs flera inloggningsalternativ. Exempel på identitetsleverantörer som kunde användas är Twitter, Google, Microsoft med flera. Även en helt skild användarregistrering kunde skapas för att eliminera risken att någon användare inte har ett konto på något av de sociala medierna. Speciellt nu i början av 2018 är det mera aktuellt än någonsin att fundera på användningen av sociala medier och vad företagen gör med den personliga informationen som insamlas. Därför kan det finnas en möjlighet att användare inte känner sig bekväma att använda någon av de sociala mediernas tjänster för inloggning.

Även inloggning med fingeravtryck eller iris-scanning kunde implementeras.

5.4 Språk

Olika språk är också ett krav i framtiden eftersom det talas både svenska och finska på brandstationen, även engelska är ett vanligt språk och även det borde implementeras.

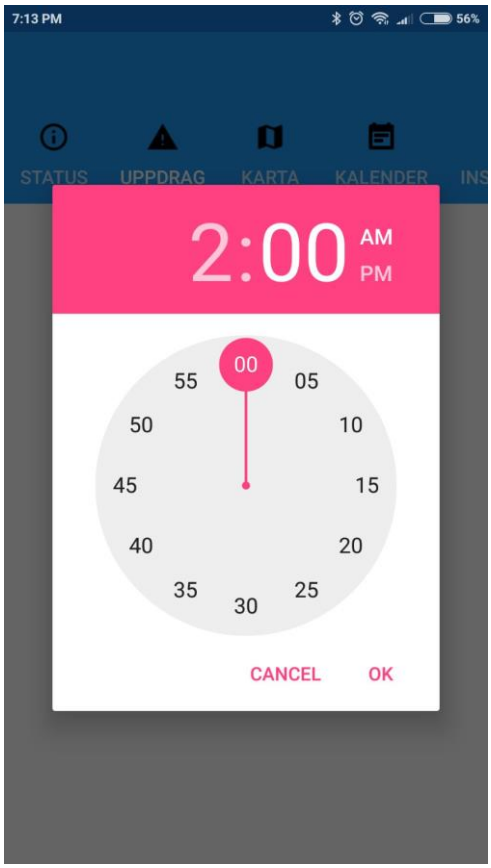
5.5 Avancerad kalender

Som redan nämndes i kapitel 5.5 kunde en mera avancerad kalender utvecklas. Som första steg skall kalendern kopplas till Azure för att kunna synkronisera med en databas och på så sätt få en kalender som en administratör har möjlighet att underhålla medan vanliga

medlemmar endast kan se informationen i kalendern. Även en skild kalender eller kalenderdel kunde implementeras för till exempel styrelsen, kassören eller liknande. På detta sätt skulle alla kalenderfunktioner, förutom användares privata händelser, finnas samlade på samma ställe.

5.6 Avancerad anmälningsfunktion

Efter att feedback från testanvändare inkommit hur anmälningsfunktionen fungerar kunde en mera avancerad och lättanvänd funktion implementeras. Till exempel kunde man antingen från identitetsleverantören hämta namnet på användaren för att undvika att användaren måste skriva in sitt namn manuellt. Sedan kunde en lista med alternativ såsom ”deltar”, ”deltar inte” och ”försenad” skapas. Ifall alternativet ”försenad” väljs kunde en så kallad tidsväljare (figur 16) öppnas där det kunde anges vilket klockslag användaren uppskattar att hen är på plats alternativt hur länge i minuter, till exempel ”30 minuter”.



Figur 16. Tidsväljare i Android.

5.7 Övrigt

Övriga funktioner är till exempel en inbyggd chattfunktion för snabb kommunikation mellan medlemmarna där även en möjlighet att skicka röstmeddelande kunde finnas. Detta för att underlätta kommunikationen medan användaren till exempel sitter i bilen på väg till stationen eller olycksplatsen.

En användarprofil kunde det även finnas användning av där användaren har möjlighet att fylla i olika specialkompetenser såsom båtförare eller rökdykare och på så sätt kan planerare se vilka specialkompetenser som kommer att finnas tillgängliga för uppdraget och vilka som saknas ifall förstärkning behövs.

Användargränssnittet kan komma att behöva uppdateras vart efter feedback inkommer från användare för att få allt att fungera så smidigt som möjligt. Målet med användargränssnittet är att användaren skall kunna använda applikationen utan att behöva fästa särskilt stor uppmärksamhet vid den. Detta eftersom det kommer att uppstå situationer där användning av applikationen kommer att ske vid olämpliga tillfällen, till exempel bilkörning. På grund av att applikationen utvecklades från grunden och fokus låg på att få en fungerande testversion, lades inte extra tid på att finjustera användargränssnittet. Därför bör man fundera vidare på bland annat hur menyer och navigering fungerar utgående från användarnas feedback.

Intresse för applikationen har redan anmälts från andra frivilliga brandkårsföreningar, vilket i sin tur leder till att ett system för att särskilja olika stationer från varandra kommer att behövas för att undvika att kalenderdata och anmälningar inte blandas ihop.

Denna applikation har stor potential att vidareutvecklas vartefter behov dyker upp. Xamarins utveckling går snabbt framåt och kommer förhoppningsvis att kunna erbjuda flera och ännu smidigare funktioner och lösningar i framtiden.

Intresset från andra stationer visar att behovet av en applikation av denna typ är stort för att underlätta verksamheten.

6 Resultat

Resultatet av examensarbetet är en mobilapplikation som uppfyller flera av de krav som den slutgiltiga applikationen kommer att ha och den kommer att kunna fungera som en första testversion.

Applikationen består i dagsläget av en inloggningsfunktion med Facebook som identitetsleverantör, ett anmälningssystem som synkroniserar data med en SQL-databas i Azure, en funktion som läser inkommande meddelanden samt en kalender med lokal lagring. Därtill finns flera klasser och funktioner som förbereder för vidareutveckling.

Genom användning av denna applikation kommer förhoppningsvis första skedet av utryckningarna att gå mycket smidigare och mindre tid går åt till onödigt väntande. Applikationen borde också ge en bra grund för fortsatt utveckling av flera funktioner för att slutligen ha en applikation som samlat alla de funktioner som frivilliga brandkårens medlemmar kan behöva på ett och samma ställe.

Nästa steg är att samla in användarnas feedback om hur användargränssnitt samt funktionaliteten upplevs.

7 Diskussion

Själva applikationen var väldigt intressant och Xamarin var en användbar plattform även om den hade en hel del buggar och dokumentation som i flera fall inte hängde med den snabba utvecklingen. Tyvärr var projektet alldeles för omfattande och avancerat för att få lanserat en färdig version, speciellt med en enda utvecklare.

Examensarbetet var väldigt lärorikt, både att utveckla en mobilapplikation från grunden samt hela Xamarin var nästan helt obekant. Endast Visual Studio, programmeringsspråket C# samt delvis XAML var bekant från förr. Vid framtida utveckling av nya mobilapplikationer och program överlag kommer denna erfarenhet att vara väldigt nyttig. I nya projekt skulle

nog arbetet utföras annorlunda på några punkter men i det stora tycker jag att utvecklingen har lyckats bra.

Xamarin som plattform har väldigt stor potential. I dagsläget fungerar den skapligt men just på grund av buggarna och den bristande dokumentationen har väldigt mycket tid gått åt till efterforskning ifall funktionen över huvud taget stöds längre eller om det finns någon ny funktion som inte finns med i den officiella dokumentationen ännu. I många fall har buggar och problem funnits i åratal och trots löften om snabba åtgärder från Xamarins utvecklare har i många fall ännu inget hänt. Men sådant är ändå helt förståeligt med en relativt ny plattform som utvecklas i den takt som Xamarin gör, där funktioner överges till förmån för bättre alternativ. En bättre kommunikation med användare önskas dock.

Plattformsoberoende utveckling är något som säkert kommer att växa sig ännu större i framtiden, fast detta examensarbete fokuserade på Android fick man ändå en körbar iOS version på köpet.

Introduktionen av det nya .NET standard klassbiblioteket (SCL) borde göra utvecklingen ännu smidigare samt ge tillgång till flera funktioner och kommer att bli väldigt intressant att få använda sig av i kommande projekt.

Förhoppningsvis kommer applikationen att vidareutvecklas för att i slutändan användas av många olika frivilliga brandkårer över hela landet.

Källförteckning

IT-mästaren, *Back-end & Front-end*. (u.å.) [Online]
https://www.itmastaren.se/om-oss/ordlista/back-end-front-end [hämtat 8.4.2018]

Microsoft. *Översikt över Active Directory Domain Services.* (u.å.) [Online]
https://msdn.microsoft.com/sv-se/library/hh831484(v=ws.11).aspx [hämtat 8.4.2018]

OData. *Overview* (u.å.) [Online]
http://www.odata.org/documentation/odata-version-3-0/odata-version-3-0-core-protocol/
[hämtat 8.4.2018]

W3schools. *SQL Tutorial* (u.å.) [Online]
https://www.w3schools.com/sql/ [hämtat 8.4.2018]

Wikipedia, 2014. *Software Development Kit* [Online]
https://sv.wikipedia.org/wiki/Software_Development_Kit [hämtat 8.4.2018]

Cordova. *Home Page* (u.å) [Online]
https://cordova.apache.org/ [hämtat 8.4.2018]

Node.js. *About* (u.å.) [Online]
https://nodejs.org/en/about/ [hämtat 8.4.2018]

Microsoft, 2015. *Introduction to the C# Language and the .NET Framework.* [Online]
https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework [hämtat 4.4.2018]

Microsoft, 2017. *What is XAML?* [Online]
https://docs.microsoft.com/en-us/previous-versions/visualstudio/design-tools/expression-studio-2/cc295302(v=expression.10) [hämtat 8.4.2018]

Xamarin. *About.* (u.å.) [Online]
https://www.xamarin.com/about [hämtat: 28.3.2018]

Xamarin. *Forms.* (u.å.) [Online]
https://www.xamarin.com/forms [hämtat: 28.3.2018]

Microsoft. *What is Azure.* (u.å.) [Online]
https://azure.microsoft.com/en-in/overview/what-is-azure/ [hämtat: 28.3.2018]

Microsoft. *Vad är SaaS.* (u.å.) [Online]
https://azure.microsoft.com/sv-se/overview/what-is-saas/ [hämtat: 28.3.2018]

Microsoft, 2016. *Om Mobile Apps i Azure App Service.* [Online]
https://docs.microsoft.com/sv-se/azure/app-service-mobile/app-service-mobile-value-prop
[hämtat 29.3.2018]

Microsoft. *What is New in Visual Studio 2017.* (u.å.) [Online]
https://www.visualstudio.com/vs/whatsnew/ [hämtat 29.3.2018]

Microsoft, 2017. *MVVM.* [Online]
https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm [hämtat 28.3.2018]

Microsoft. *The Model Class*. [Online]
https://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx [hämtat 8.4.2018]

Pedly, A., 2016. *Portable Class Library (PCL) vs Shared Projects.* [Online]
https://xamarinhelp.com/portable-class-library-pcl-vs-shared-projects/ [hämtat 28.3.2018]

Microsoft, 2017. *Tabbed Page*. [Online]
https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/tabbed-page [hämtat 28.3.2018]

Microsoft, 2016. *Pages.* [Online]
https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/pages
[hämtat: 29.3.2018]

Microsoft, 2018. *Broadcast Receivers in Xamarin.Android* [Online]
https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/broadcast-receivers
[hämtat 2.4.2018]

Microsoft, 2016. *MessagingCenter* [Online]
https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/messaging-center [hämtat 2.4.2018]

Syncfusion. *About us*. (u.å.) [Online]

https://www.syncfusion.com/company/about-us [hämtat: 31.3.2018]