Byggesag Borger App til Smartphones og Tablets

Afgangsprojekt ved DTU, IMM

Af Emil Sylvest Jensen (s100025)

Vejleder Stig Høgh

Afleveret 31. Januar 2014

Kongens Lyngby

Emil Sylvest Jensen_____

Technical University of Denmark

Informatics and Mathematical Modeling

Building 303B, DK-2800 Kongens Lyngby, Denmark

Phone + 45 4525 3031

compute@compute.dtu.dk

www.imm.dtu.dk

Summary

This report describes the development of an application for presenting building case data to case workers and the general public. The application (app) is implemented for iOS and Android smartphones and tablets.

The app is developed in C# using Xamarin Studio and the Xamarin plugin for Visual Studio, making it possible to develop a cross platform application in a single programming language, thus being able to share a lot of the code between the implementations for the different platforms. Some code, especially concerning the Graphical User Interface, has to be implemented in different ways for iOS and Android, and some focus will be put on these differences in implementation in this report.

The project will end with a prototype of the application where the most important functions are implemented.

Resumé

Denne rapport beskriver udviklingen af en applikation til at præsentere byggesagsdata til sagsbehandlere og borgere. Applikationen (app) er implementeret til iOS and Android smartphones og tablets.

Appen er udviklet i C# med brug af Xamarin Studio og Xamarin plugin til Visual Studio, som gør det muligt at udvikle en applikation på tværs af platforme i et enkelt programmeringssprog, således at man kan dele store dele af koden mellem implementeringerne på de forskellige platforme. Noget kode, især det der har med den grafiske brugerflade at gøre, skal dog implementeres på forskellige måder for iOS og Android, og der vil i rapporten blive lagt en del fokus på disse forskelle i implementering.

Projektet ender med en fungerende prototype af appen hvor de vigtigste funktioner er blevet implementeret.

Forord

Dette projekt er lavet som afgangsprojekt for Diplomingeniør (Diplom IT) på Danmarks Tekniske Universitet (DTU) af Emil Sylvest Jensen. Projektet giver de sidste 20 ECTS point der kræves af uddannelsen.

Tak til Stig Høgh, lektor ved Institut for Matematik og Computer Science på DTU, for vejledning i løbet af dette projekt.

Projektet bliver udviklet i samarbejde med Geograf A/S, som efterfølgende får rettighederne til den udviklede applikation, der skal færdigudvikles af undertegnede efter afslutning på dette projekt.

Indhold

Figurer

Indledning

Om Geograf A/S og Byggesag

Geograf A/S er et lille firma beliggende i Allerød, som primært har specialiseret sig inden for GIS (Grafiske Informationssystemer) og med programmet Byggesag, der kan bruges af kommuner til at styre hele sagsgangen omkring en byggesag.

Systemoversigt

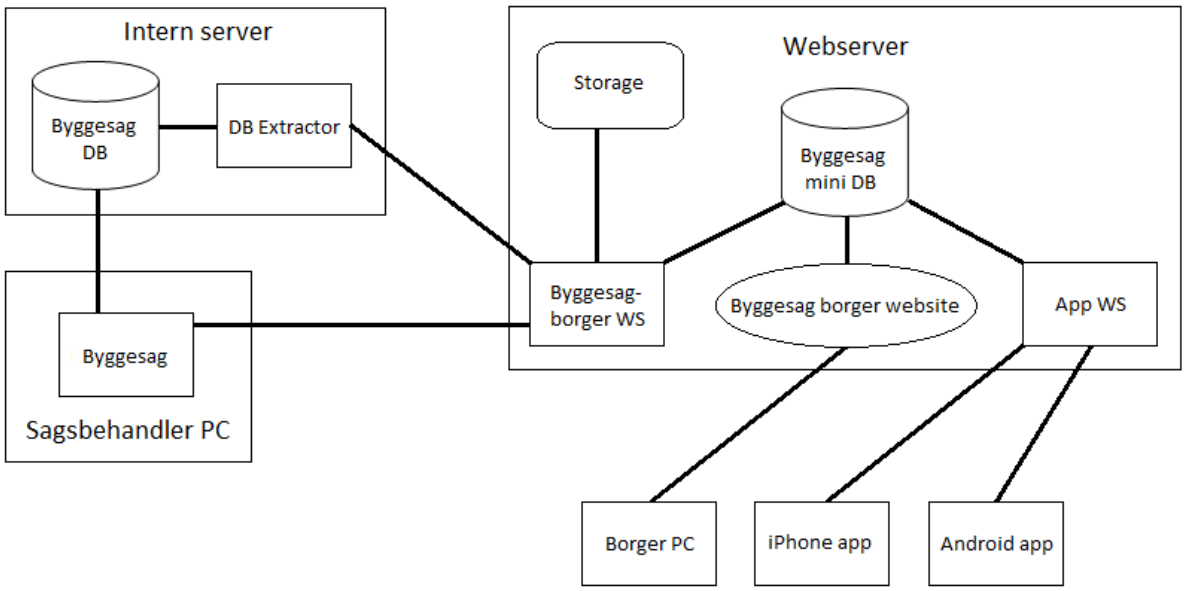
De to apps og webservicen dette projekt omhandler, er del af et større system omkring programmet Byggesag[1]. Figur 1 herunder viser fordelingen af de forskellige programmer og databaser for Byggesag Borger. Udover Byggesag programmet og den tilhørende database er resten af systemet udviklet af undertegnede i praktikperioden og i løbet af dette projekt. Kun iPhone appen, Android appen og App webservicen er blevet skabt til dette projekt. DB Extractor programmet, Byggesag borger webservicen og byggesag mini databasen er der blevet lavet nogle ændringer i for at tilpasse det til appen.

Byggesag bruges af sagsbehandlere i kommuner til at styre byggesager. Dataen for disse byggesager gemmes i Byggesag databasen.

DB Extractor programmet er et program der køres på en kommunes interne server med et fast interval, fx en gang om dagen. Programmet tager en del af databasen og kopierer det til Byggesag mini databasen på web serveren. Overførslen bliver udført via Byggesag borger webservicen.

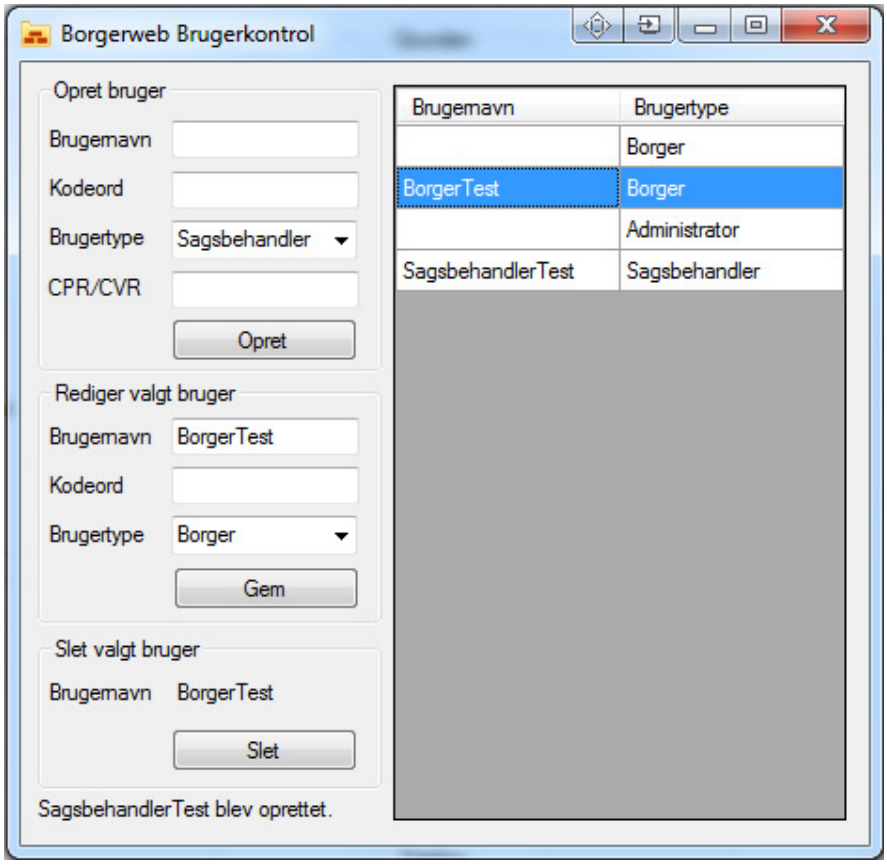
I Byggesag kan man offentliggøre et dokument der hører til byggesagen, dette overfører dokumentet til en mappe på web serveren (Storage ikonet i figur 1) via Byggesagborger webservicen.

[1] http://geograf.dk/byggesag (14-01-2014)



Figur 1 : Systemfordeling for Byggesag Borger

Brugerstyring for Byggesag borger udføres i Byggesag. Figur 2 herunder viser et billede af brugerstyringen. Som det kan ses kan kodeordet og CPR/CVR nummeret ikke ses i listen da disse bliver hashet før de bliver uploadet.



Figur 2 : Brugeropret

Som det kan ses i figur 1 kan brugere tilgå systemet enten via PC og hjemmesiden, eller via Android eller iOS appen. Hjemmesiden tilgår databasen direkte da den ligger på serveren, men de to apps er nødt til at hente data der skal vises fra App-webservicen.

Problemformulering

Projektet går ud på at udvikle en smartphone/tablet applikation der kan præsentere data og dokumenter fra Geograf A/S's program Byggesag. Applikationen skal også kunne vise byggesager på en kortbaggrund (fx Google Maps, Geodatastyrelsens gratis korttjenester og WMS-baserede tjenester generelt).

Ydermere skal byggesagsbehandlere kunne få rapporter/opsummeringer af data fra byggesagsdatabasen, fx antal sager med en specifik sagsbehandler eller med et areal over 100m^2.

Programmet skal udvikles for/i samarbejde med Geograf A/S, hvor jeg også udførte mit praktikforløb.

Der skal laves en version af applikationen til Android, iOS og evt. Windows 8. Alle vil blive programmeret i C# og så tilpasset til det aktuelle styresystem.

Analyse

Kravspecifikation

Dette kapitel vil finde og opstille krav ved først at beskrive de Use Cases (UC's) der er relevante for programmet og derfra opstille en række funktionelle krav (FK) til programmet. Derudover vil der også blive opsat en række nonfunktionelle krav (NFK).

Use Cases

Da appens primære funktion er at vise data for en byggesag betyder det at der kun er 3 overordnede Use Cases:

1. Vis sagsdata.
2. Rediger data.
3. Hent rapport.

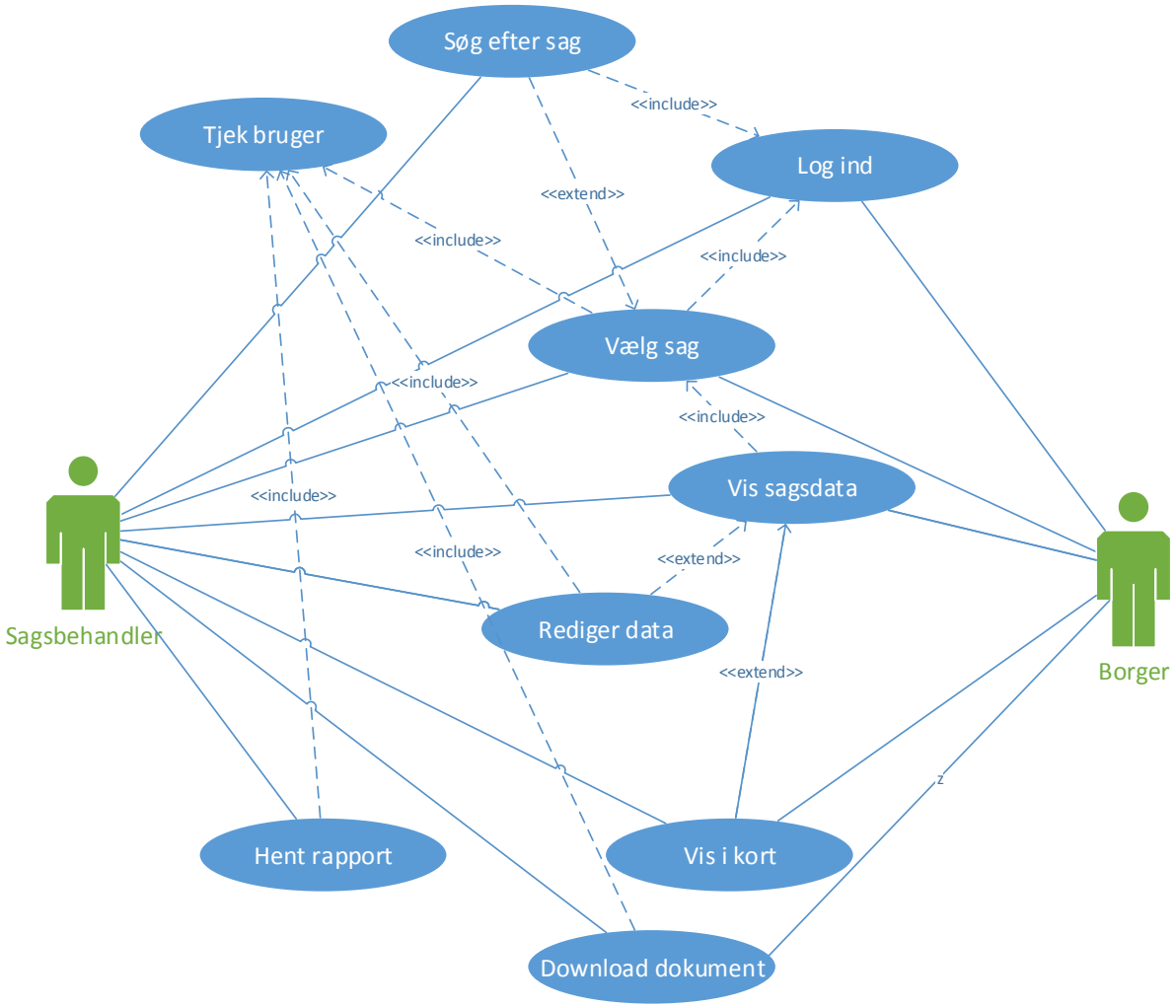
Den første af disse (vis sagsdata) er også inkluderet i den anden (rediger data), da man først skal vise en sag før man kan redigere den.

Til de overordnede UC's hører der nogle mindre UC's:

4. Log ind.
5. Søg efter sag.
6. Vælg sag.
7. Vis i kort.
8. Download dokument
9. Tjek bruger. (ser om bruger har adgang til det data han vil hente)

Forholdet mellem de forskellige UC's kan ses i figur 3 herunder.

Grundet tidsbegrænsning og at den vigtigste del af programmet ligger i UC 1 og de underordnede UC's der bliver brugt i UC 1, er UC 2 og UC 3 ikke blevet implementeret i projektperioden.



Figur 3 : Use case diagram

Use case 1 - Vis sagsdata (som borger)

Successcenariet er beskrevet i trinene uden bogstav på, trinene med bogstav er alternative forløb.

Sub UC	Trin	Handling
4	1	Bruger indtaster brugernavn, password, vælger kommune og trykker på ”Log ind” knap.
	2	Appen ser at bruger har indtastet et brugernavn, password og valgt en kommune. Der sendes en login-forespørgsel til kommunens webservice.
	2a	Appen ser at bruger mangler at indtaste et brugernavn, password eller vælge en kommune, og fortæller brugeren at det manglende felt skal indtastes.
	3	Webservice ser i databasen at brugeren findes og returnerer brugeren til appen.
	3a	Webservice ser at brugeren ikke findes i databasen og Null returneres, brugeren får at vide at brugerinfo ikke er korrekt.
	3b	Webservice er ikke tilgængelig, bruger får at vide at der var forbindelsesproblemer.
6	4	Appen ser at det er en borger og spørger webservicen om beskrivelser af borgerens byggesager.
	4a	Appen ser at det er en sagsbehandler og brugeren præsenteres for søgeskærm.
	5	Webservicen trækker beskrivelser af byggesagerne ud af databasen.
	6	Beskrivelserne returneres til appen.
	7	Bruger præsenteres for beskrivelserne og bedes vælge en sag.
	8	Bruger klikker på en sag.
	9	Appen sender en forespørgsel til webservicen om sagens data.
	10	Webservicen henter sagsdata fra databasen.
	11	Webservicen returnerer sagsdata til appen.
1	12	Bruger præsenteres for sagsdata i 4 faneblade.
	13a	Bruger kan trykke på "Back", dette fører tilbage til trin 7.
7	13b	Bruger klikker på ”Vis i kort” knap.
8	13c	Bruger trykker på et dokument i dokument fanebladet, hvorefter dokumentet hentes fra webservicen.
	13d	Bruger lukker programmet.

Use case 1 - Vis sagsdata (som sagsbehandler)

Sub UC	Trin	Handling
4	1	Bruger indtaster brugernavn, password, vælger kommune og trykker på ”Log ind” knap.
	2	App ser at bruger har indtastet et brugernavn, password og valgt en kommune. Der sendes en login-forespørgsel til kommunens webservice.
	2a	App ser at bruger mangler at indtaste et brugernavn, password eller vælge en kommune, og fortæller brugeren at det manglende felt skal indtastes.
	3	Webservice ser i databasen at brugeren findes og returnerer brugeren til appen.
	3a	Webservice ser at brugeren ikke findes i databasen og Null returneres, brugeren får at vide at brugerinfo ikke er korrekt.
	3b	Webservice er ikke tilgængelig, bruger får at vide at der var forbindelsesproblemer.
5	4	App ser at det er en sagsbehandler og brugeren præsenteres for søgeskærm.
	4a	App ser at det er en borger og spørger webservicen om beskrivelser af borgerens byggesager.
	5	Bruger indtaster andet data end byggesagsnummer og trykker "Søg".
	5a	Bruger indtaster byggesagsnummer og trykker "Søg". Hvis sagen findes springes der til trin 7.
	5b	Bruger trykker "Søg" uden at indtaste data. Og får at vide at der skal indtastes noget før der kan søges.
	6	Liste med beskrivelser af sager der passer på de søgte data hentes fra webservice og bruger præsenteres for liste med beskrivelserne.
	6a	Der fandtes ingen sager der matchede data. Bruger får dette at vide og kan søge på nyt data.
6	7	Bruger præsenteres for beskrivelserne og bedes vælge en sag.
	8	Bruger klikker på en sag.
	9	App’en sender en forespørgsel til webservicen om sagens data.
	10	Webservicen henter sagsdata fra databasen.
	11	Webservicen returnerer sagsdata til app’en.
1	12	Bruger præsenteres for sagsdata i 4 faneblade.
	13a	Bruger kan trykke på "Back", dette fører tilbage til trin 4.
7	13b	Bruger klikker på ”Vis i kort” knap.

	13c	Bruger trykker "Rediger data".
8	13d	Bruger trykker på et dokument i dokument fanebladet, hvorefter dokumentet hentes fra webservice.
	13e	Bruger lukker programmet.

Funktionelle krav

Krav 2 og 3 er ligesom med de use cases de er fundet ved ikke blevet implementeret i projektperioden.

1. Programmet skal kunne vise en mængde data for hver byggesag.
2. Sagsbehandlere skal kunne bruge appen til at ændre i noget af dataen.
3. Sagsbehandlere skal kunne hente en rapport over en mængde data, fx hvor mange byggesager der blev godkendt inden for et givent tidsinterval.
4. Borgere og sagsbehandlere skal kunne logge ind på appen.
5. Sagsbehandlere skal kunne søge på og vælge forskellige sager via nogle parametre.
6. Borgere skal kunne vælge mellem de sager hvor de er ejer eller ansøger.
7. Brugere skal kunne se hvor bygningen er på et kort, hvis sagen er stedfæstet (den indeholder koordinater).
8. Brugere skal kunne hente dokumenter der er offentliggjort i sagen.
9. Kun borgere der har adgang til sagen og sagsbehandlere skal kunne se data for den individuelle sag.

Nonfunktionelle krav

1. Flest mulige borgere med adgang til smartphones og tablets skal kunne bruge programmet, dog uden at tilgodese teknologier der bliver brugt af fåtal.
2. Koden skal være genanvendelig da der skal udvikles til forskellige platforme.
3. Appen skal være brugervenlig. Det skal være nemt at overskue hvad man kan og hvordan man gør det. Der skal altså ikke være skjulte funktioner som man udfører med fingerbevægelser som fx swiping og pinching.
4. Ved fejl fra brugerens side skal brugeren informeres om hvordan disse fejl kan håndteres.
5. Ved softwarefejl og serverfejl skal fejlen indberettes til serveren som gemmer dem i en Event Log.
6. Appen skal ikke være for langsom, så kun data der er relevant for det brugeren er i gang med skal hentes fra webservicen.

Valg af platforme og teknologier

De tre største platforme på smartphone/tablet markedet i Danmark er uden tvivl Android, iOS og Windows Mobile. Windows Mobile har dog stadig så lille en del af det danske marked[2], at det blev valgt at vi i første omgang kun udvikler en app til Android og iOS.

App udvikling bliver normalt udført i Objective C til iOS og Java til Android. Men da Geograf A/S normalt udvikler i C#/.Net vil det være mere passende at kunne udvikle i .Net. Java ligner meget C# så det ville nok være muligt at udvikle Android delen i Java hvis det kunne genbruges i C# til andre platforme. Men da Objective C er så forskelligt fra C# og Java er det meget svært at kunne genbruge kode mellem platformene i deres normale sprog.

En hurtig søgning viste dog at der er blevet lavet udviklingsmiljøet Xamarin[3] (tidligere kendt som Mono) der gør det muligt at programmere Android og iOS apps i C#, hvorefter programmet recompiler appen til Java og Objective C kode. Dette gør at det meste af koden kan genbruges, dog ikke det hele, da GUI delen til de to platforme stadig skal laves i forskellige filer(xml til android og xib til iOS).

Med Xamarin kan man udvikle til både Android og iOS enten i Visual Studio med et plugin, eller i Xamarins eget Xamarin Studio[4]. Da der i forvejen bliver udviklet i Visual Studio hos Geograf A/S blev der valgt at udvikle i dette. For at få iOS apps til at virke kræver det dog at man udvikler på eller kan forbinde til en Mac, da apples xcode compiler kun findes til Mac. Da udviklingen til iOS appen startede fandt jeg så ud af at GUI delen ikke kan laves i Visual Studio med mindre man laver det hele programmatisk (tilføjer alle elementer i koden), i stedet for at gøre det i en dedikeret GUI fil med grafisk editor.

Android delen bliver altså implementeret i Visual Studio (2012) og iOS delen bliver implementeret i Xamarin Studio på en Mac. Begge apps og webservicen bliver udviklet i C#.

[2] http://www.dr.dk/Nyheder/Viden/Tech/2013/08/15/104843.htm (20-01-2014)
[3] http://xamarin.com/ (20-01-2014)
[4] https://xamarin.com/studio (20-01-2014)

Design

Webservice

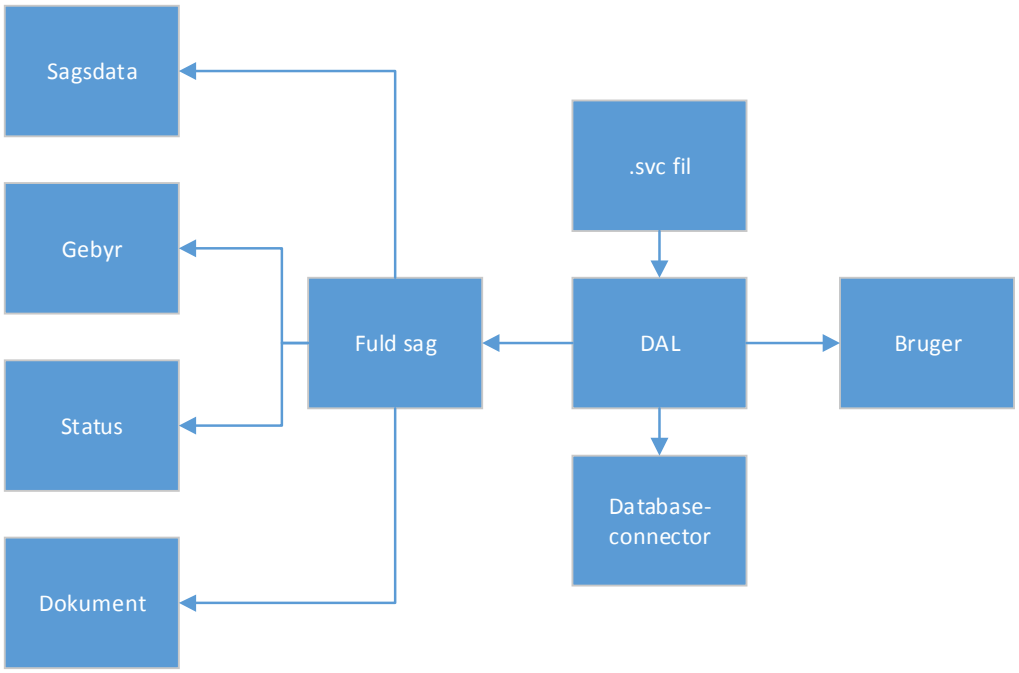
Da Byggesag Borger hjemmesiden allerede var lavet (i praktikperioden) af undertegnede, kunne en del af designet til den nye webservice tages derfra.

Figur 4 herunder viser domænemodellen for webservicen, som er magen til hjemmesidens domænemodel for data-delen, bortset fra "Fuld sag" og ".svc fil".

Fuld sag var nødvendig da det med en asynkron webservice er upraktisk at hente de 4 dele (Sagsdata, Gebyr, Status og Dokument) hver for sig. De 4 dele indeholder hver især data der vises på de 4 faneblade med data og dokumenter der er tilknyttet en sag.

Appen kalder metoder på svc filen som via DAL klassen henter data fra databasen som metoderne i svc filen så behandler, fx at hente sagsdata og godkende brugeren.

Et fuldt klassediagram over webservicen kan ses i det fortrolige appendix.

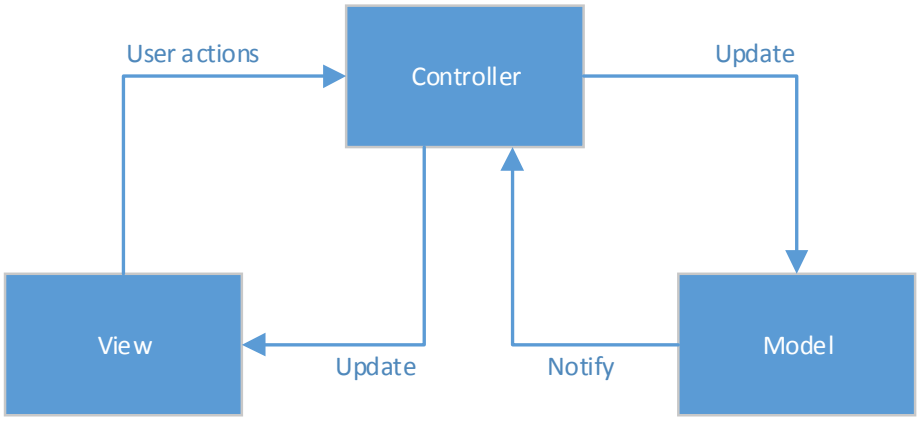


Figur 4 : Domænemodel

Appen

MVC

I iOS udvikling bruges der ofte Model-View-Controller pattern, da dette er nemt og intuitivt at bruge med Apples Xcode designer. Det er dog ikke et helt normalt MVC pattern da view og model interagerer lidt i det normale MVC pattern, hvor de i iOS ikke interagerer. Figur 5 herunder viser hvordan MVC patternet[5] kan se ud for iOS apps. Det samme pattern bliver brugt i Android appen, dog er det ikke implementeret på samme måde.



Figur 5 : MVC

View

View delen er XIB filerne og XML filerne for iOS henholdsvis Android.

XIB filer[6] er Apples eget filformat der indeholder XML med et specifikt schema. XIB filer består dels af handles (outlets) som man bruger til at tilgå elementerne fra en Viewcontroller, dels af XML der beskriver opsætningen af designet for de grafiske elementer. I figur 6 herunder kan man se et udsnit af det XML kode der står i en XIB fil. Man kan i toppen se at filen indeholder to outlets og længere nede kan man se view-elementet som er rod for de grafiske elementer, bl.a. de to textfields man kan se i koden.

[5] https://developer.apple.com/library/ios/documentation/general/conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html (21-01-2014)
[6] http://www.speirs.org/blog/2007/12/5/what-are-xib-files.html (21-01-2014)

```
        <outlet property="txtPassword" destination="EvN-aP-e34" id="zSO
        <outlet property="view" destination="6" id="7"/>
    </connections>
</placeholder>
<placeholder placeholderIdentifier="IBFirstResponder" id="-2" customCla
<view clearsContextBeforeDrawing="NO" contentMode="scaleToFill" id="6">
    <rect key="frame" x="0.0" y="0.0" width="320" height="480"/>
    <autoresizingMask key="autoresizingMask" widthSizable="YES" heightS
    <subviews>
        <textField opaque="NO" clipsSubviews="YES" contentMode="scaleTo
            <rect key="frame" x="59" y="69" width="202" height="30"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES"
            <fontDescription key="fontDescription" type="system" pointS
            <textInputTraits key="textInputTraits"/>
        </textField>
        <textField opaque="NO" clipsSubviews="YES" contentMode="scaleTo
            <rect key="frame" x="59" y="107" width="202" height="30"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES"
            <fontDescription key="fontDescription" type="system" pointS
            <textInputTraits key="textInputTraits" secureTextEntry="YES
        </textField>
```

Figur 6 : Eksempel på XML fra XIB fil

Hvor XIB filer indeholder andet XML end det rent grafiske, indeholder XML filerne der bruges i android appen kun information om de grafiske elementer. Rod-elementet i XML'en er således et Layout, fx LinearLayout, som indeholder sub-views der enten kan være views (fx TextView), controls (som knapper) og også andre layouts.

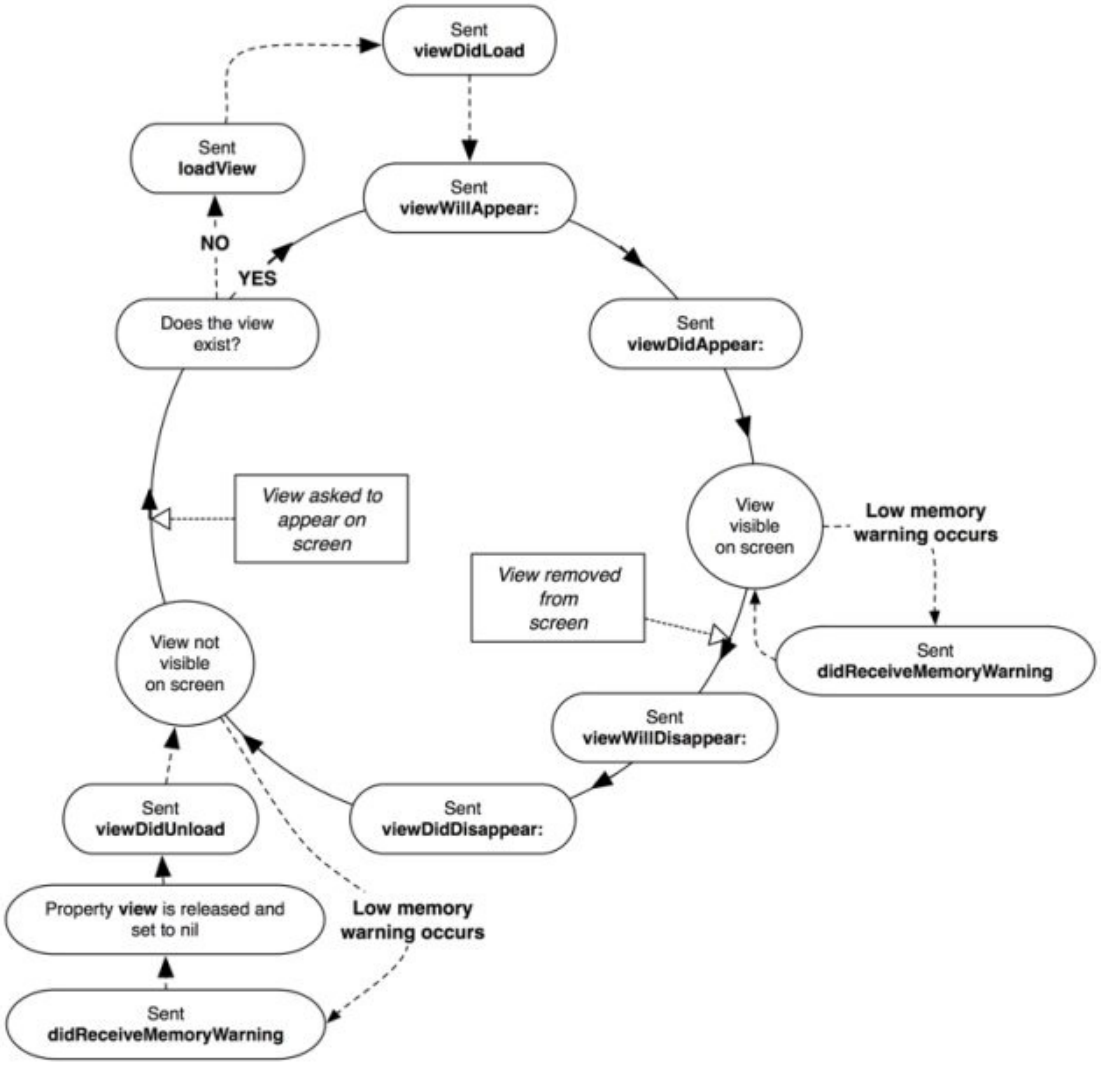
Controller

Controller-rollen udføres i iOS appen af en ViewController (VC) og i Android appen af en Activity (AC).

Både ViewControllere og Activities håndterer hver et enkelt skærmbillede i appen. Et skærmbillede med login tekstbokse og knapper kan fx håndteres af et en VC kaldet "LoginScreenViewController" eller en AC kaldet LoginActivity. Når man så trykker "Log ind" og føres til et nyt skærmbillede, vil det være en ny VC eller AC der står for at håndtere dette skærmbillede. I iOS skiftes der skærmbillede ved at pushe en ny VC på en VC-stack i en klasse kaldet AppDelegate. I Android gøres det anderledes, da man kan starte i en vilkårlig AC og man kan så oprette en ny AC og sætte denne til at være den aktive AC. Det fungerer ligesom en stack i den hentydning at hvis man lukker en AC ved at trykke "Back" så kommer man til den forrige AC, men det er ikke en rigtig stack da man til hver en tid kan sætte hvilken som helst AC som den aktive og derved ændre dens position i stacken til øverst.

En VC og en AC håndterer begge livscyklus events for det view der er tilknyttet. I figur 7 herunder kan man fx se livscyklussen for et iOS view. ViewDidLoad er den vigtigste metode da det er i denne metode at dataen skal loades ind i viewet fra modellen.

I Android er der en lignende livscyklus der starter med OnCreate, som svarer til ViewDidLoad.



Figur 7[7] : Livscyklus for iOS view

[7] http://i.stack.imgur.com/eYCHy.jpg (23-01-2014)

Model

Model delen af appen er implementeret med en static klasse der indeholder det data der bliver brugt af appen, fx sagsdata og brugerinfo. Klassen er lavet static da alle Activities skal have adgang til informationen i den og da der kun skal være en enkelt sag åben ad gangen var det hensigtsmæssigt at bruge en static klasse med static variabler.

Brugergrænseflade

Ud fra analysen med Use Cases var det muligt at opsætte en række skærmbilleder som skal vises til brugeren i appen. Disse skærmbilleder er:

- Login
- Søg efter sag
- Vælg sag
- Tabs

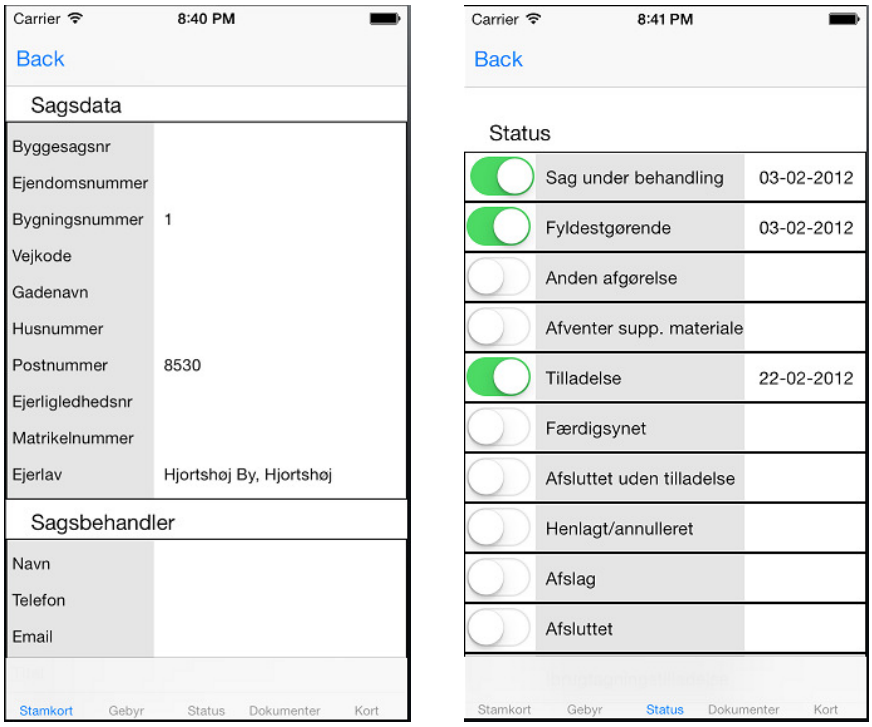
Tabs skærmbilledet består af en TabBar hvor man kan skifte mellem 4/5 skærmbilleder der vises sammen med fanebladene. De skærmbilleder der vises med TabBaren er:

- Stamkort
- Gebyr
- Status
- Dokumenter
- Kort (Er sit eget skærmbillede uden TabBar i Android appen)

I figur 8 herunder kan man se Stamkort og Status skærmbillederne for iOS appen og i figur 9 kan man se Stamkort og Status skærmbillederne for Android appen.

Som det kan ses i figurerne er der meget forskel på designet i de to apps. Dels er det fordi der endnu ikke er blevet lagt vægt på at gøre designet pænt og dels er det fordi man på de to platforme bruger helt forskellige designs. Apple har udgivet iOS 7 som kommer med en masse ændringer og anbefalinger til hvordan man designer sin app, så den passer med apples egne værktøjer, så brugeren har nemmere ved at finde ud af hvad de arbejder med da det er noget de har set før. Eksempelvis kan man se at "Back" knappen ikke har nogen border, men bare ligner et hyperlink, dette er noget som er kommet i iOS 7. En anden ting er at iOS ikke har et checkbox view, men i stedet bruger den type knap man kan se til højre i figur 8, hvor Android bruger checkbokse som man kan se i figur 9.

Grunden til at Kort/Vis i kort er lavet som en tab i iOS appen og et selvstændigt skærmbillede i Android appen, er at den action bar der bruges i android appen, ikke kan vise mere end 4 tabs, uden at man skal scrolle til siden, for at se de resterende tabs.

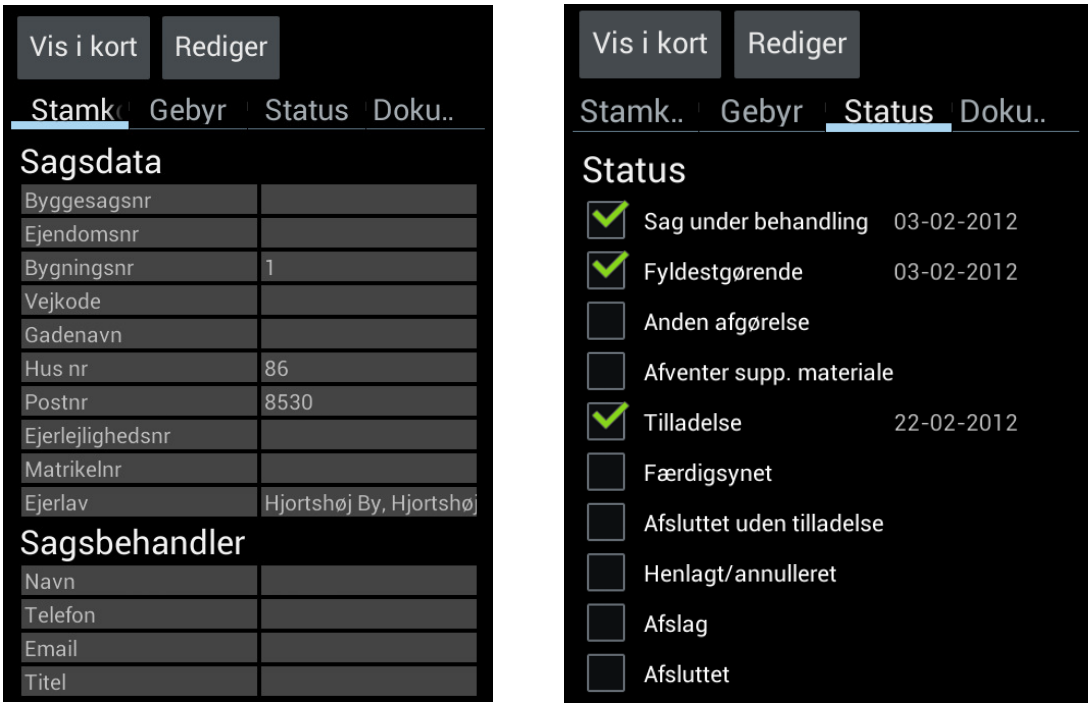


Figur 8 : iOS skærmbilleder (Stamkort og Status)

En anden stor forskel der er på at designe GUI til iOS og Android, kommer af de faktum at Android bliver brugt på så mange forskellige enheder. Dette betyder at der er mange forskellige skærmstørrelser, hvilket man skal tage højde for hvis man vil have appen til at se ordentlig ud på alle enheder. En del af dette løses ved at have flere mapper med layout filer der kan bruges til forskellige skærmstørrelser. Det tydeligste eksempel på dette er at have en mappe med layout filer der bruges når appen bliver brugt på en tablet. Når appen starter en ny Activity med et tilknyttet layout ser den om enheden er en telefon eller tablet, hvis enheden er en telefon bruges standard layoutet, men hvis enheden er en tablet bruges layout filen med samme navn fra "layout-large" mappen. Hvis man så har nogle layouts der kun skal bruges på tablet kan man lave et tjek for enhedens type før man starter sin Activity, så man ikke prøver at bruge en layout fil der ikke findes. Ydermere kan man lave layout filer der bliver brugt når enheden drejes, hvilket især er brugbart når man skal lave layouts til tablet, hvor pladsen på skærmen afhænger meget at hvordan telefonen

vender. Dette kunne også være et problem på telefoner, men der har jeg slået rotation fra, da denne type app som skal vise en liste af data ikke fungerer særligt godt med et horisontalt layout på telefoner, da højden så vil være meget lille.

På iOS har man også brug for seperate view filer for iPhone og iPad, men i stedet for Androids løsning med forskellige mapper, kan man når man opretter en ViewController i Xamarin Studio vælge at oprette en Universal ViewController, hvorefter der oprettes to XIB filer, fx LoginViewController_iPad og LoginViewController_iPhone. Der behøves ikke tages ret meget hensyn til forskellen på skærmstørrelsen på forskellige enheder for iOS apps, da der er meget få forskellige skærmstørrelser og iOS er god til at få det til at se ordentligt ud på de ældre telefoner, der har en lidt mindre skærm.



Figur 9 : Android skærmbilleder (Stamkort og Status)

Implementering

I dette afsnit vil implementeringen af dele af systemet blive forklaret og der vil især blive forklaret om forskellene mellem implementeringsmetoder i iOS og Android appen.

Webservice

Valg af teknologi

Der er mange forskellige måder at implementere webservices på, men tre af de mest almindelige er REST, SOAP og WCF.

REST[8] er oplagt til services der skal implementeres i forskellige programmer og forskellige platforme da det bliver udført via HTTP og derfor er meget lidt afhængig af platform, da næsten alle sprog kan håndtere HTTP fordi det bare kræver en normal netforbindelse.

SOAP[9] kan ligesom REST være mere eller mindre uafhængigt af platform, hvis man designer sin webservice fra bunden med WSDL filen først, men dette er besværligt og der vil oftest blive brugt et framework med annotation til at lave servicen og så kan du hurtigt miste uafhængigheden af platform hvis man bruger parametre med sprogspecifikke typer som fx Nullable types.

WCF[10] er et framework der bruges til at oprette og forbinde til webservices lavet i .Net. Implementering af webservices i WCF er meget simpelt, da man bare sætter attributten [OperationContract] på de metoder i ens ".svc" fil der skal være tilgængelige på webservicen og sætter attributterne [DataContract] og [DataMember] på henholdsvis klasserne og variablerne i klasserne som skal med i webservicen. Et eksempel på dette kan ses i figur 10 herunder. WCF er simpelt at implementere i et Xamarin projekt både på Android og iOS, da et Xamarin projekt kan implementere en WCF service ved at oprette en proxy klasse som projektet kan kalde metoder på med Silverlight Service Model Proxy Generation Tool (SLsvcUtil). Dette gør at metoderne i webservicen vil være asynkrone, hvilket er vigtigt i en smartphone/tablet app da man ikke vil låse GUI'en mens programmet venter på et svar.

[8] http://en.wikipedia.org/wiki/Representational_state_transfer (25-01-2014)
[9] http://en.wikipedia.org/wiki/SOAP (25-01-2014)
[10] http://en.wikipedia.org/wiki/Windows_Communication_Foundation (25-01-2014)

```
[DataContract]
public class Case
{
    [DataMember]
    int caseid;
    [DataMember]
    string propertynr;
```

Figur 10 : Eksempel på WCF attributter

Sikkerhed

Som vist i designafsnittet tilgår webservicen databasen via en DAL klasse der kalder metoder på en DatabaseConnector klasse. DatabaseConnector klassen (DBC) indeholder nogle metoder der udfører SQL queries på databasen via SqlCommand klassen fra .Net biblioteket. DBC indeholder bl.a. nogle select metoder, hvoraf nogle af dem tager både SQL querien og en eller flere SqlParameter med for at parameterize queries. Metoderne der også tager parametre med til SQL kommandoen bliver brugt i de web service metoder hvor brugeren har sendt input med, for at sikre mod SQL injections[11].

Udover beskyttelsen mod SQL injections med parameterization bliver der brugt hashing af passwords og CPR/CVR numre, for at sikre at uvedkommende ikke kan komme til at få fat i denne information. En løsning der blev overvejet med CPR/CVR numre var at kryptere dataen og så afkryptere den igen når det skulle bruges, men feedback fra Byggesag Borger hjemmesiden fra Århus kommune gjorde at CPR/CVR nummeret ikke skulle vises. De ville have at vi skulle fjerne det helt, men da en bruger bliver identificeret med nummeret og brugerens sager findes ved at finde de sager hvor brugerens CPR/CVR svarer til ansøgerens eller ejerens CPR/CVR, kunne det ikke fjernes helt uden at skulle lave meget om på systemet og gøre brugeroprettelse mere besværligt. Løsningen var så at hashe nummeret så man stadig kan sammenligne hashen i databasen med hashen som kommer fra brugerens login.

Til hashingen anvendes der Salt, som er en streng af karakterer der tilføjet til strengen der skal hashes før den bliver hashet. Dette gør det langt sværere at "gætte" passwordet, enten manuelt eller via brute force hvor de prøver kendte ord og navne.

[11] http://en.wikipedia.org/wiki/Sql_injection (25-01-2014)

App

Nyt skærmbillede

Når et nyt skærmbillede oprettes og sættes som det synlige skærmbillede, gøres dette ved at oprette en ny Activity(AC) på Android og ViewController (VC) på iOS.

Android har i en AC en overrideable metode (da alle AC'er nedarver fra Activity eller en anden klasse der nedarver fra Activity) kaldet OnCreate, hvor der udføres den kode der skal ske før brugeren ser skærmbilledet. I denne metode kaldes SetContentView som tager et layout som parameter, enten en layout fil der er oprettet med designeren eller et layout man selv programmerer direkte i koden.

I iOS kaldes en VC's constructor først og her kan der detekteres om enheden er en smartphone eller tablet, hvorefter den XIB fil der passer til VC'en og enheden bliver valgt, fx LoginViewController_iPhone.

Efter constructoren er udført køres metoden ViewDidLoad, hvor man kan udføre det man skal før brugeren bliver præsenteret for viewet.

Tilgå elementer i views

Tilgang til elementer i views gribes meget forskelligt an i Android og iOS.

I iOS kan man i designeren bruge drag-and-drop for at oprette handles (Outlets) eller events (Actions) til elementer. Dette er praktisk da det holder code overhead nede da man kun behøver have kode til håndtering af de elementer man vil ændre i, men det gør det også besværligt at skulle lave drag-and-drop når man har mange elementer i et view. Ved at oprette disse handles og events er det nemt at tilgå elementerne i sin kode, da et en XIB fil er direkte tilknyttet til en VC.

Man kan tilknytte eventhandlers til elementer i iOS enten ved at tilføje dem til et outlets events eller ved at oprette en action direkte. Hvis man skal gøre flere ting ved fx en Button ville det være hensigtsmæssigt bare at lave et Outlet og så bruge dette til at tilknytte handlers, men da jeg prøvede at tilføje en eventhandler til en tekstboks' TextChanged event viste det sig at man med et Outlet ikke kan tilgå det rigtige TextChanged event, men kun TextWillChange, som bliver udført før teksten ændrer sig og derfor kan man ikke finde ud af hvad den nye indtastning er. Derfor er det nogle gange nødvendigt at bruge Actions istedet for Outlets, så man kan tilgå nogle metoder der ikke er tilgængelige i koden med Outlets men kun i designeren.

I Android kan man i layout XML filer tilføje et ID til et element, dette ID bliver så tilføjet til Resource filen i projektet. Når man så skal tilgå et element fra en AC skal man så hente et handle til elementet i resource

filen med følgende metode: FindViewById<Button>(Resource.Id.btnLogInd);. Dette er nødvendigt da man i forhold til iOS har en løs forbindelse mellem controlleren (AC) og viewet. Men det skaber også en del problemer, da man ikke kan have elementer i forskellige layouts med samme ID (navn) og intellisense foreslår elementer fra andre layouts da den ikke kan se hvilket layout elementet hører til.

Kald til webservice

Som nævnt i designafsnittet tilgår man webservicen i en Xamarin App ved at oprette en proxy class som man kalder metoder på. Når man opretter en proxy tager den adressen på webservicen med som en parameter, så ved at skifte adressen kan man kalde forskellige instanser af webservicen, hvilket er meget praktisk da forskellige kommuner skal have webservicen liggende på deres server.

Kald til webservicen sker asynkront, da synkrone kald ville låse den tråd de kører i, hvilket er uhensigtsmæssigt i en app da GUI så ville låse hvis den kører i main tråden. Istedet for at kalde en metode på web service proxien og vente på metodens resultat kalder man en metode, fx CheckCase, og så har man tilknyttet et eventhandler til proxiens CheckCaseCompleted event. Når webservicen så returnerer et resultat starter en ny tråd der kalder eventets "Completed" metode (se eksempel i figur 11 herunder)..

```
private void OnCheckCaseIdCompleted(object sender, checkCaseIdCompletedEventArgs args)
{
    string msg = null;

    if (args.Error != null)
    {
        msg = args.Error.Message;
    }
    else if (args.Cancelled)
    {
        msg = "Request was cancelled.";
    }
    else
    {
        if (args.Result == true)
            client.getFullCaseAsync(soegtSagsnummer,SessionVars.bruger);
        else
            msg = "Ingen sag fundet med dette byggesagsnummer";
    }
    if (msg != null)
        InvokeOnMainThread ( () => {
            lblErrorText.Text = msg;
        });
}
```

Figur 11 : Eventhandler til webservice metode completed (iOS)

Threading

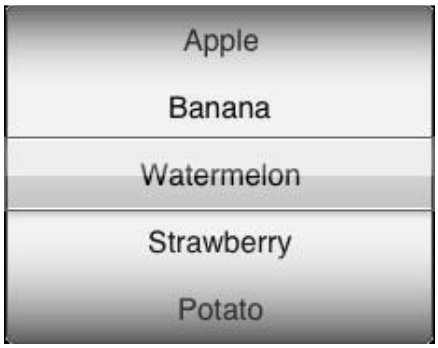
Som det kan ses nederst i figur 11 kan man ikke bare ændre i GUI elementer direkte hvis man ikke arbejder i Main tråden. Man er nødt til at lave metodekaldet InvokeOnMainThread (på iOS) eller RunOnUIThread (på Android). Hvis man ikke ændrer GUI elementer på denne måde kan man risikere at kollidere med andre tråde der arbejder på GUI'et, hvorefter der kan forekomme fejl, som fx race conditions hvor en tråd prøver at læse hvad der står i en tekstboks mens en anden prøver at skrive i den, hvilket kan give forskellige resultater alt efter hvilken rækkefølge de to tråde fik adgang til resursen i.

InvokeOnMainThread og RunOnUIThread venter på at få adgang til main/UI tråden og udfører så deres arbejde når de har adgang til denne.

Autosuggest

Når en sagsbehandler skal søge efter sager, er et af felterne man søger på et adressefelt. Dette felt skal implementeres med autocomplete, altså at den foreslår gadenavne når du har skrevet mindst 2 bogstaver. I Android var dette simpelt at implementere da Android har et AutoCompleteTextView, så man bare skal give listen af adresser til en adapter og tilknytte denne til textviewet.

I iOS er dette en del sværere at implementere, da iOS hverken har autocomplete support eller dropdown lister. Istedet for dropdown lister har iOS "spinners" (se figur 12), der fungerer som et hjul hvor man skal dreje sig ned til det man vil vælge. En sådan spinner bliver brugt i Login skærmbilledet, dog ikke på selve layoutet, men som en popup istedet for tastaturet når man trykker på Kommune tekstfeltet. En spinner fungerer dog ikke til autocomplete, så i stedet indsættes der ved køretid et view under tekstfeltet, med et listview der viser listen over adresser der passer til det man har indtastet.



Figur 12 : iOS spinner[12]

[12] http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/flex/articles/introducing-flex46sdk/fig4.jpg (26-01-2014)

Tab Bar

Tab baren (kan ses i figur 8 og 9) skulle implementeres på forskellige måder i iOS og Android og der opstod problemstillinger ved begge implementeringer.

I Android var der valget mellem to forskellige teknologier til implementering af tab baren; TabHost og ActionBar. ActionBar har et mere moderne design og passer bedre med action bar designmønsteret der blev introduceret i Android 3.0. Ulempen ved dette nye design er dog at ActionBar ikke er understøttet i tidligere versioner end 3.0. En mulig løsning på dette var at implementere ActionBarSherlock, som er et API der gør det muligt at anvende ActionBar i ældre versioner af Android. Da dette blev forsøgt implementeret opstod der dog så mange problemer at det blev droppet til fordel for TabHost, som fungerer i alle versioner af Android men er et lettere forældet design.

TabHost bliver implementeret ved at den AC der indeholder baren nedarver fra TabActivity klassen og rod-elementet i layout filen skal være et TabHost element. Man kan så oprette nye tabs med metoden AddTab på TabHost objektet (se Figur 13 herunder).

```
private void CreateTab(Type activityType, string tag, string label)
{

    try
    {
        var intent = new Intent(this, activityType);
        intent.AddFlags(ActivityFlags.NewTask);

        var spec = TabHost.NewTabSpec(tag);
        //var drawableIcon = Resources.GetDrawable(drawableId);
        //spec.SetIndicator(label, drawableIcon);
        spec.SetIndicator(label);
        spec.SetContent(intent);

        TabHost.AddTab(spec);
    }
    catch (Exception ex)
    {

    }
}
```

Figur 13 : Tilføjelse af tabs til TabHost

I iOS er det noget mere besværligt at implementere en tab bar hvis den ikke RootView (det første view der startes). I iOS er en UITabBarController nemlig nødt til at være RootView. Så hvis man skal bruge en tab bar andre steder end som det originale skærmbillede skal man i AppDelegate klassen have en metode der skifter fra den UINavigationController man indtil nu havde brugt, til en UITabBarController.

En UINavigationController holder styr på stacken af VC'er man har været på og kan navigere tilbage til med Back knappen i UINavigationController baren (øverst i figur 8). Så når man skifter til UITabBarControlleren mister man navigationsbaren med Back knappen. Løsningen på dette viste sig at være at man i hver VC der bliver tilføjet til TabControlleren tilføjer NavigationControlleren. Og hvis man så trykker Back i en tab skifter den NavigationControlleren tilbage til at være RootView.

Maps integration

I sager hvor der er tilknyttet koordinater skal man kunne se sagens location på et kort. Dette gøres med Google Maps henholdsvis Apples Maps.

I iOS var det meget simpelt at implementere, da man bare skal oprette et MKMapView objekt, vælge typen af kort, i dette tilfælde Hybrid (satelitbillede med optegnede veje), tilføje en annotation (en markør på kortet) og zoome ind på området. Alt dette kan ses i figur 14 herunder.

```
map = new MKMapView(UIScreen.MainScreen.Bounds);
map.MapType = MKMapType.Hybrid;
map.AddAnnotation (new MKPointAnnotation (){
    Title = theTitle,
    Coordinate = new CLLocationCoordinate2D (xCoord, yCoord)
});
CLLocationCoordinate2D center = new CLLocationCoordinate2D (xCoord, yCoord);
map.Region = new MKCoordinateRegion(center,new MKCoordinateSpan(0.003,0.003));

View = map;
```

Figur 14 : Apples Maps implementering

I Android er det en del sværere at implementere Google Maps da Google Maps nu er en del af Google Play Services[13]. Appen skal have tilføjet en Google Play Services komponent som man først skal installere i Android SDK hvorefter man kan tilføje den til ens projekt. Da appen skal kunne bruges på ældre versioner af Android skulle den bagudkompatible version "Google Play Services (Froyo) vælges. Derudover skal appen have tilladelse til at bruge funktioner som Maps skal bruge, bl.a. til at bruge telefonens GPS. Dette gøres i appens AndroidManifest.XML fil.

Når dette er gjort kan man oprette et MapFragment (eller hvis det skal være bagudkompatibelt; et SupportMapFragment) (se figur 15 herunder), hvor man skifter kamerapositionen til det sted man vil vise. For at indsætte en markør på kortet er man nødt til at vente til at kortet har loadet, så jeg opretter en handler som kører en metode der indsætter metoden efter et sekund. Metoden tjekker så om kortet er loadet og hvis det ikke er delayer den metoden et sekund mere, hvilket kan ske flere gange, dog tjekkes der for at det ikke sker for mange gange da programmet ellers kunne gå ind i en uendelig løkke.

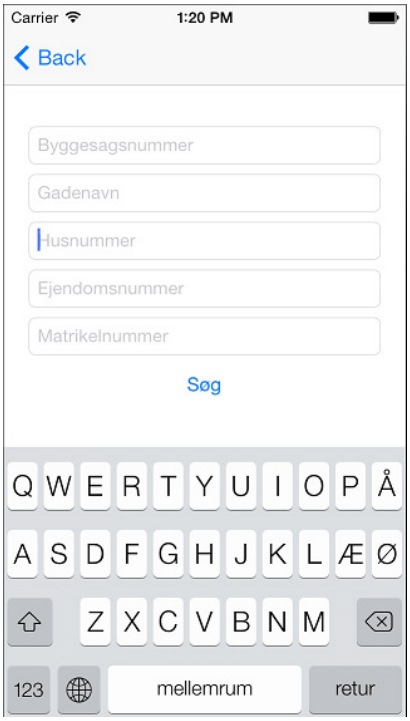
```
LatLng location = new LatLng(loc.xCoord, loc.yCoord);
CameraPosition.Builder builder = CameraPosition.InvokeBuilder();
builder.Target(location);
builder.Zoom(18);
CameraPosition cameraPosition = builder.Build();
GoogleMapOptions mapOptions = new GoogleMapOptions()
    .InvokeCamera(cameraPosition)
    .InvokeMapType(GoogleMap.MapTypeHybrid);

myMapFragment = SupportMapFragment.NewInstance(mapOptions);

Android.Support.V4.App.FragmentTransaction tx = SupportFragmentManager.BeginTransaction();
tx.Replace(Resource.Id.aNameFrameLayout, myMapFragment);
tx.Commit();

delayRunnable = () =>
{
    addMarkerStartup();
};

Handler handler = new Handler();
handler.PostDelayed(delayRunnable, 1000);
```

Figur 15 : Google Maps implementering

[13] http://docs.xamarin.com/guides/android/platform_features/maps_and_location/part_2_-_maps_api/ (28-01-2014)

Manglende funktionalitet

På iPhone lukker man on-screen tastaturet ved at trykke på knappen nederst til højre, som kan have forskellige navn alt efter hvad der startede tastaturet. Hvis man fx trykker i et TextField, hedder knappen return/retur(se figur 16 herunder) og ved andre elementer kan den hedde Done eller Next. Denne knap viste sig ikke at fungere ad sig selv, man er nødt til at assigne en delegate til hver tekst boks som får tastaturet til at lukke når der bliver trykket retur.



Figur 16 : iOS skærmbillede med tastatur fremme

Test

Test af webservicen og appen er sket løbende, da de enkelte funktioner er blevet testet white box testet da funktionerne blev oprettet og senere black box testet ved at afprøve appen.

White Box testing, hvor man tester at programmet at funktioner gør det de skal internt, er hovedsageligt blevet brugt til test af webservicen, hvor der blev oprettet en projekt der blev forbundet til webservicen og de forskellige metoder i webservicen blev testet både med Unit Tests, hvor man ser om et metodekald giver det forventede resultat, og mere dybdegående tests hvor man debugger webservicen trin efter trin for at se om den reagerer korrekt på forskellige forhold.

Appen er primært blevet gennemtestet med Black Box testing, hvor man ignorerer "the inner workings" og bare fokuserer på om appen kan udføre de metoder den skal korrekt. De steder hvor der så er fundet fejl er der så blevet brugt White Box testing med debugging for at finde ud af hvor fejlen opstod.

Selvom der er blevet testet grundigt på webservicen og appen er det dog muligt at der stadig er fejl i programmet, da det er umuligt at forvente alle fejl der kan opstå i et program, enten pga. systemfejl eller brugerfejl.

Konklusion

Den udviklede prototype har fået implementeret og gennemtestet de funktioner som borgere skal kunne tilgå, altså visning af data og visning i kort, samt søgning på forskellige sager for sagsbehandlere. Der mangler dog at blive implementeret nogle af de forskellige skærmbilleder til tablet, samt at få tilpasset og finpudset designet i alle skærmbilleder så det ser pænere ud.

Implementeringen viste at meget af koden til Android appen og iOS appen kunne deles. Det var primært GUI delen der var forskellig for de to apps, og der opstod forskellige problemstillinger for de to apps, der dog for det meste viste sig at have relativt simple løsninger.

Efter projektperioden skal appen laves færdig da undertegnede har fået job i virksomheden. Det der mangler at blive implementeret er som sagt layouts til forskellige skærmtyper, redigering af data, da dette kræver en ændring af hvordan der arbejdes på serveren, da der indtil nu i Byggesag Borger kun er blevet kopieret data fra den rigtige byggesagsdatabase over til Byggesag Borger databasen, og der med redigering også skal ændres i den rigtige database som ligger på en intern server der ikke kan tilgås fra webserveren. En mulig løsning på dette er at gemme ændringerne i en ny database på webserveren, som et program på den interne server så tjekker med et fast tidsinterval for at lave de passende ændringer i byggesagsdatabasen.
Derudover mangler der også at blive implementeret rapportering, altså hvor en sagsbehandler fx kan få at vide hvilke sager der findes med et areal på over 100 kvadratmeter. Dette er allerede implementeret i Byggesag, men det kræver Microsoft Word for at virke, hvilket jo som regel ikke er implementeret på en webserver. Så det kan enten løses med den gamle løsning og så kræve at de har installeret Word på deres webserver, ellers må der laves en anden løsning hvor der ikke laves .doc filer.