IHA – ITSMAP:

ITSMAP Temaprojekt

TrackABus

Lasse Lindsted Sørensen (09421) & Christoffer Lousdahl Werge (10842)

6/6/2013

Indholdsfortegnelse

Kravspecification

Introduktion

Hovedopgaven for applikationen var, at kunne følge en vilkårlig bus rundt i Århus. Applikationen skulle opbygges så det var nemt og intuitivt at tilgå en bus, og finde ud af hvor den var henne. Brugeren skulle kunne vælge en bus fra en liste over alle busser i Århus området, og ved et simpelt tryk, komme hen til et kort hvor han kunne se bussens rute samt, bussens nuværende placering. Bussen skulle vælges på en liste af busnavne. Det skulle desuden også være nemt for brugeren at vælge en ny bus hvis det skulle være nødvendigt.

Hovedfokusset for opgaven var ruten for bussen. Da vi ikke har mulighed for at anskaffe os egentlige GPS data for bussen, skulle dennes placering blot omtrentligøres. Bussens hastighed og hvilken vej den kørte var ikke vigtig, så længe bussen kunne ses på ruten. Ruten skulle vises ved hjælp af en linje som forbinder et antal punkter. Optimalt ville disse punkter består af en omtrentlig placering af busrutens stoppesteder. Da der var en varierende afstand mellem hvert stoppested, ville ruten komme til at passe mere eller mindre på kortets veje. Dette er dog ikke så essentielt for bybusser, da afstanden mellem hvert stoppested er relativ kort.

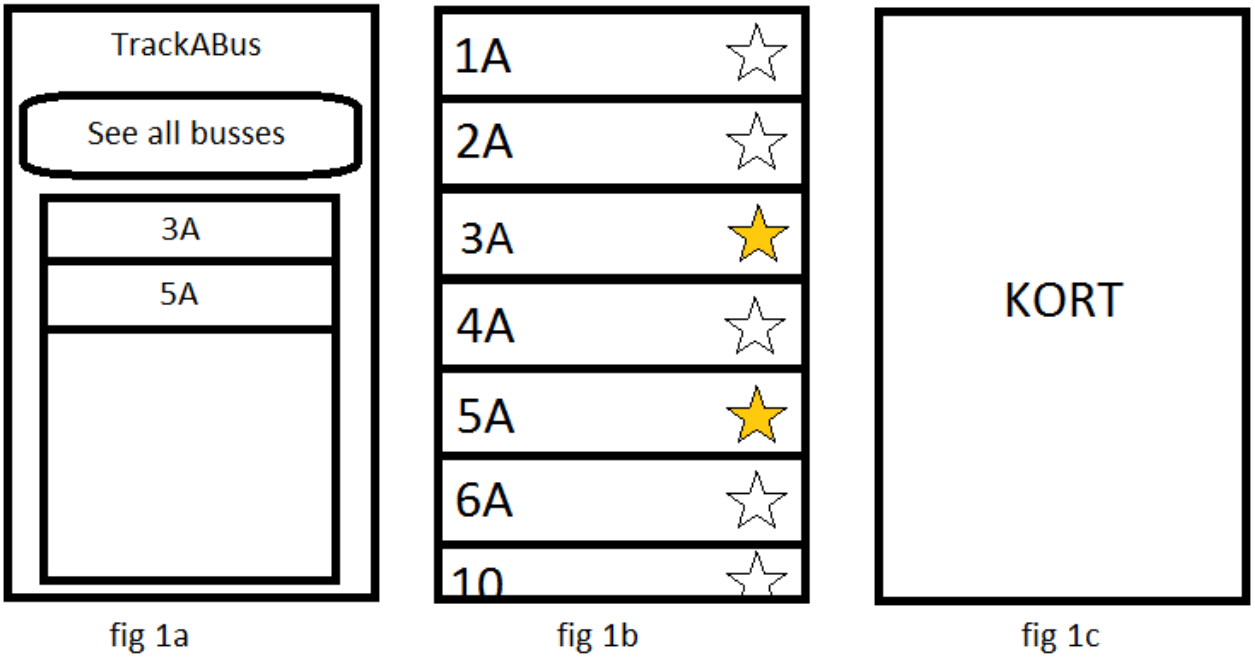
Ideer og koncepter

Der var også et antal komponenter vi krævede skulle tages i brug. Datastorage var en vigtig del af systemet, da alle busser, deres position samt deres rute skulle gemmes. Data skulle gemmes eksternt, da systemet skulle kunne køres på mange mobiler på samme tid. Vi tog derfor Windows Azure i brug, for at gemme data. Denne service har også en MobileServiceClient man kan tilgå, så man nemt kalde funktioner op mod databasen i Java. Da det dog ville kræve meget datatrafik at hente busnavne og ruter hele tiden, sås det også som en god i det at lave et favoriserings system, hvor man kunne vælge op til fem busser, som var dem man oftest kørte med. Når en favorit bus vælges, skulle den gemmes i en SQLitedatabase med en tilknyttet ContentProvider og det skulle kunne ses på listen over busser hvilken var sat til favorite. Sammen med bussen bliver dens rute også lagt på databasen. Dette betyder at man kan tilgå favorit bussernes kort

uden egentlig at have internet tilgang. På menuskærmen kan man også se en liste over de busser man har sat til favorite, så man hurtigt kan tilgå dem. Når man ikke vil have sat den til favorite mere, vil bussen blive fjernet fra SQLite databasen og fra listen på menuen. Så længe en bus er favorited vil ruten aldrig blive hentet på databasen.

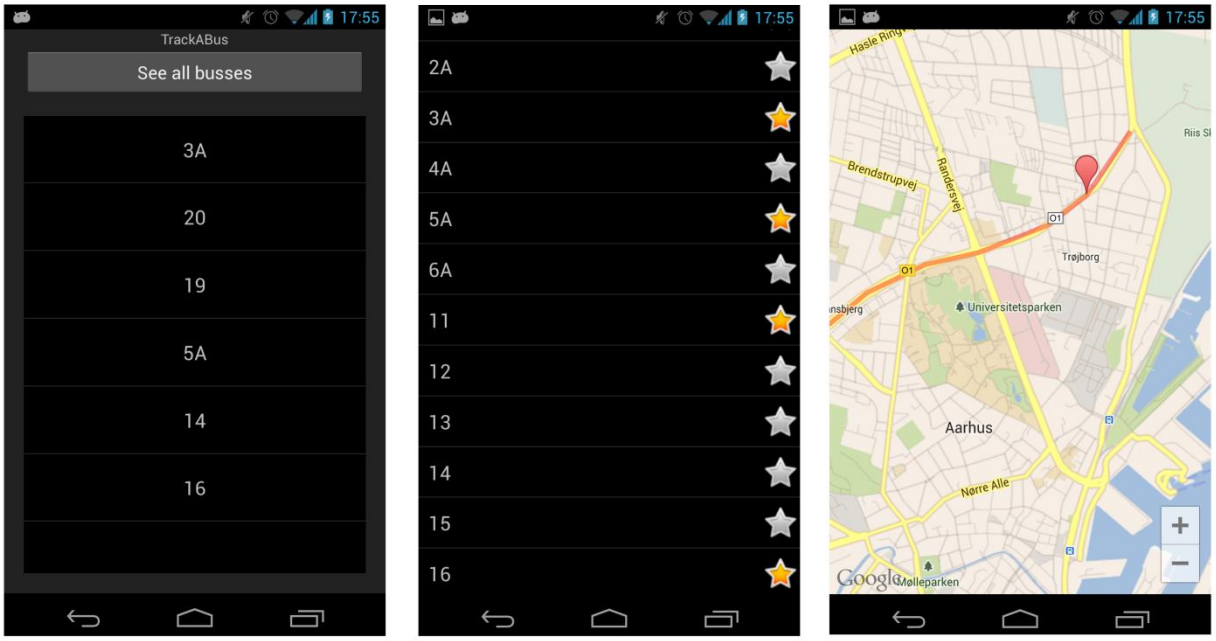
Kortet skulle vises ved hjælp af androids indbyggede Google Maps API, og skulle køre i fullscreen, så brugeren havde et klart overblik.

På figur 1 ses en skitse af de tre activities som smartphone applikationen vil have.



Figur 1: De tre aktivities; HovedMenu (1a), Listen af busser (1b) og kortet (1c)

På figur 2 kan man se hvordan vores tre aktiviter kom til at se ud i brug.

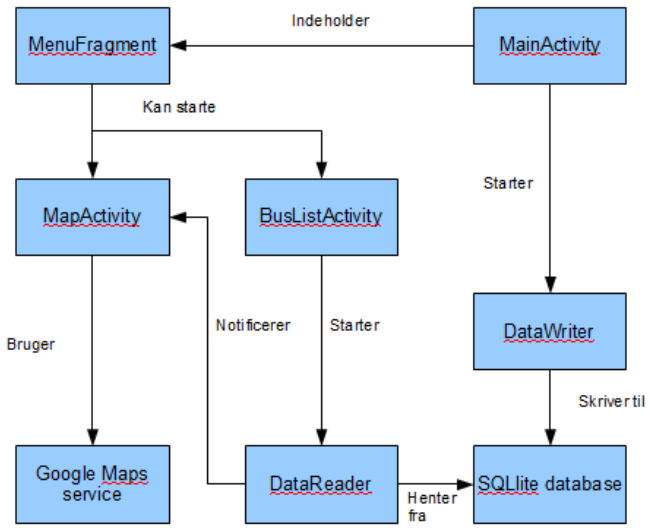


Figur 2: Implementering af HovedMenu, Listen af busser og kortet

Designvalg

Udkast

Under skabelsen af synopsen havde vi en idé om hvordan systemet skulle sættes sammen. På figur 2 kan det ses, hvordan vores indledende system skulle sættes op.



Figur 3: Indledende komponentdiagram

Her følger en beskrivelse af komponenterne.

- MainActivity + MenuFragment
    - MainActivity skulle indeholde MenuFragment som det eneste. Dette fragment skulle bestå af en simple Menu, hvor fra man kunne tilgå applikationens primære funktioner.
- MapActivity + Google Maps Service
    - MapActivity skulle indeholde et fullscreen kort, skabt ved hjælp af Google Maps Servicen. Dette kort skulle vise en valgt busrute samt følge den valgte bus via en markør
- BusListActivity
    - BusListActivitens hoved formål var at vise en liste over alle busser, som var mulige at tilgå i systemet. Når en bus var valgt, skulle DataReaderen begynde for den valgte bus.
- DataWriter
    - DataWriteren eneste funktion var at skabe tilfældigt data for en given bus' position. Dette skulle skabe illusionen at en bus bevægede, så man kunne se det på kortet. Den skulle holde skulle skrive et nyt koordinatsæt til alle buser i databasen.
- SQLite database
    - Databasen skulle holde længdegradder og breddegrader for en given bus. Disse var udelukkende tilfældiggjort af DataWriteren, men det var tænkt, at når man også kunne lægge GPS-data ind i den, når bachelor projektet gik i gang.
- DataReader
    - For hvert tidsinterval skulle DataReaderen hente koordinater for den valgte bus i databasen. Disse koordinater skulle sendes videre til kortet, hvorefter bussens position skulle opdateres.

Implementeringen

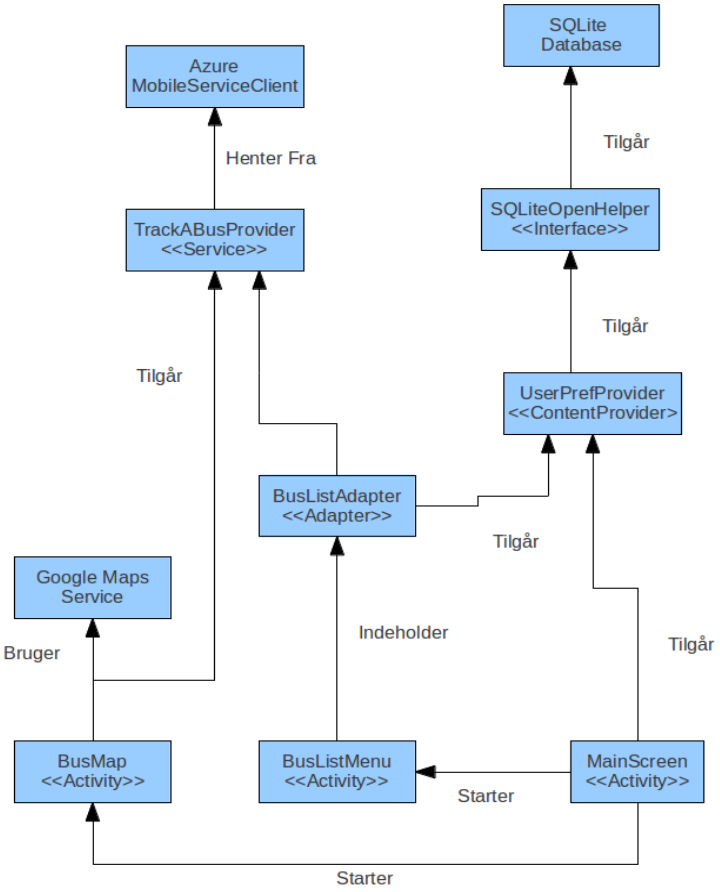
Mellem vores første udkast og vores færdige system var skete der dog en hel del ting, og kravende til systemet blev ændret. Det var f.eks. ikke tanken fra starten at der skulle være en favoritliste, og det krævede en del arbejde at få sat det op rigtigt. På figur 3 kan det ses hvordan applikation er sat op efter projektet var lavet.

På det færdige komponentdiagram er der kun taget de vigtigste komponenter med, altså er fragments og dataklasser ikke taget med. Her er en beskrivelse af komponenterne:

- MainScreen, BusListMenu og BusMap

    - MainScreen er hovedskærmen for applikationen. På den er der en knap der leder brugeren over til listen af alle busser, og en liste over busser der er sat til favorite.
    - BusListMenu listen af alle busser samt en indikator som fortæller, hvilke busser brugeren har favoritet
    - BusMap indeholder kortet hvor den valgte rute og bus bliver vist.
- Google Maps servicen er beskrevet tidligere.
- BusListAdapter
    - En custom adapter til at et ListView. Dette er skabt da vi havde behov for at lave en speciel ListView som indeholder både de normale ListView funktioner samt en ToggleButton funktion. Desuden løb vi ind i et problem hvor ToggleButtonen ville resette til false hvis man scrollede
- UserPrefProvider + SQLiteOpenHelper
    - UserPrefProvideren er en ContentProvider til SQLiteDatabasen. Den har en indre klasse som implementerer de nødvendige metoder. Man tilgår den med to URIer der eksisterer i to dataklasser

- TrackABusProvider + MobileServiceClient
    - TrackABusProvideren er en service som asynkront tilgår MobileServiceClienten. Den bruges til at tilgå data på den eksterne database så programmet ikke bare fryser hvis forbindelsen er langsom. Når data er hentet fra databasen, sender TrackABusProvideren en Message til den handler som Provideren får fra start.
    - MobileServiceClienten ligger i et bibliotek man kan downloade fra Windows Azure. Den indeholder værktøj så man nemt kan tilgå databasen fra sin mobil.



Figur 4: Komponentmodel ved færdiggjort projekt

Vi kunne godt have brugt en SQLite database til at håndtere det hele, og så bare bruge test data på telefonen, men da vi gerne ville have dette projekt til at være et startsted for vores bachelor, ville vi gerne gøre den så ”Rigtig” som muligt. Vi har derfor brugt en del tid på at få den eksterne database op at køre da det lå lidt uden for pensum, og ikke var særligt godt dokumenteret.

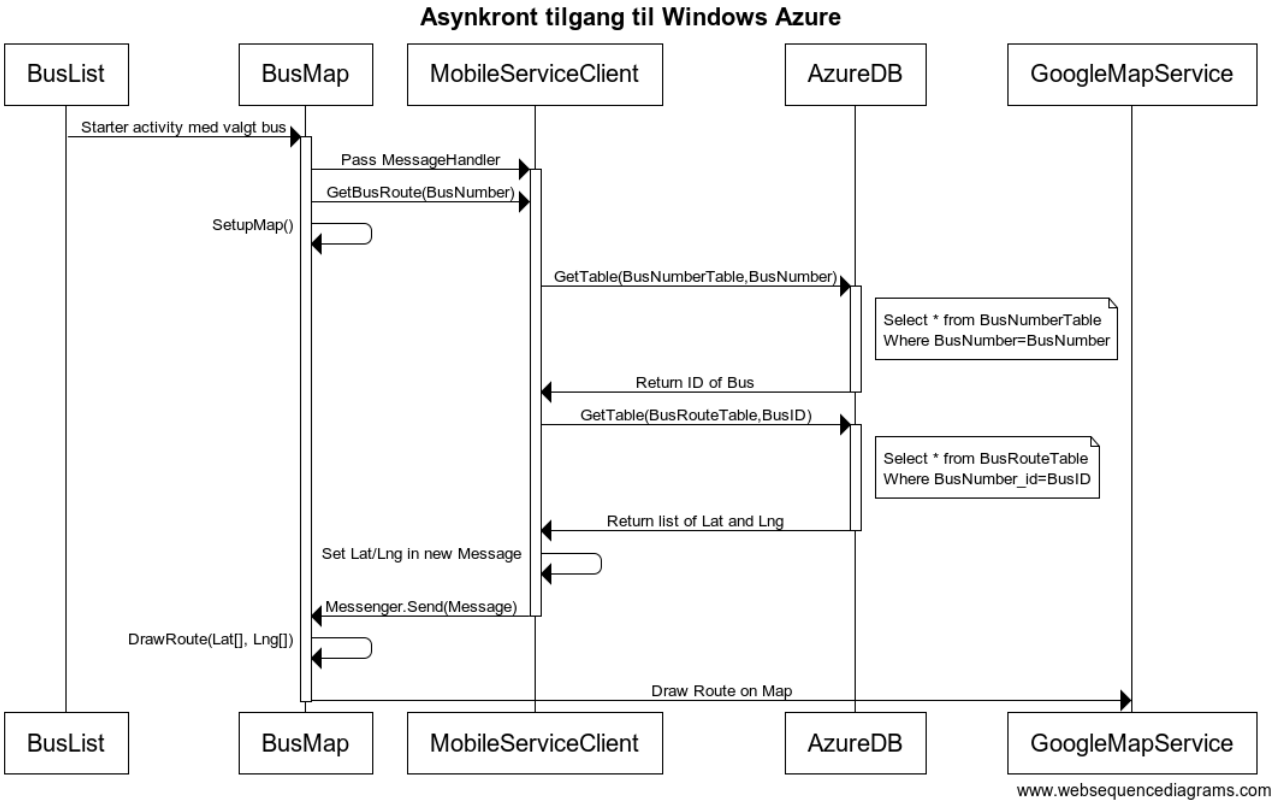
ContentProvideren er rigtig smart at bruge for en større og mere kompleks SQLite database, men i vores tilfælde giver ligger giver det bare projektet mere kompleksitet. Reelt set kunne vi godt have undgået og

bruge en den da mange af dens funktioner ikke bliver brugt i vores tilfælde. Hvis SQLite databasen skulle tilgås fra flere forskellige applikationer ville det være en omlagt mulighed at bruge ContentProvideren, da det er den egentlige funktion. Grunden til at vi gerne ville have den med i vores projekt var dog, at vi gerne ville bruge så mange af de ting vi havde lært igennem forløbet, som muligt. Det er strengt taget heller ikke nødvendigt for applikationen at køre på en tablet, men det har vi understøttet alligevel.

Under skabelsen af ListViewet, hvor der både skulle være en onClick funktion på hver item på listen, og en onToggle funktion på knappen i hver item, opstod der nogle problemer. Da det nye ListView bestod af et TekstView og en ToggleButton, men onToggle eventet på ToggleButtonen blev ikke fundet hvis man klikkede. Det fiksede vi ved at lave en custom ListView adapter hvor den blev tvunget til at kaste. Vi løb også ind i problem hvor man ikke kunne trykke på en item på listen. Dette var så fordi at knappen skulle være non focusable. Meningen med ToggleButtonen var, at man skull kunne vælge sin favorit bus. Når man slog den til insertede den bussen og ruten i databasen og når man slog den fra deletede den. Hvis man havde slået en til, og den forsvandt ud view, altså hvis man scrollede på listen, blev den sat til sin default værdi og så det som et onToggle. Default værdien var false og derfor kaldte den en delete hver gang den forsvandt ud af view. Dette fiksede vi ved at gemme staten på favorite knappen i adapteren.

Sekvensdiagrammer for vigtige dele

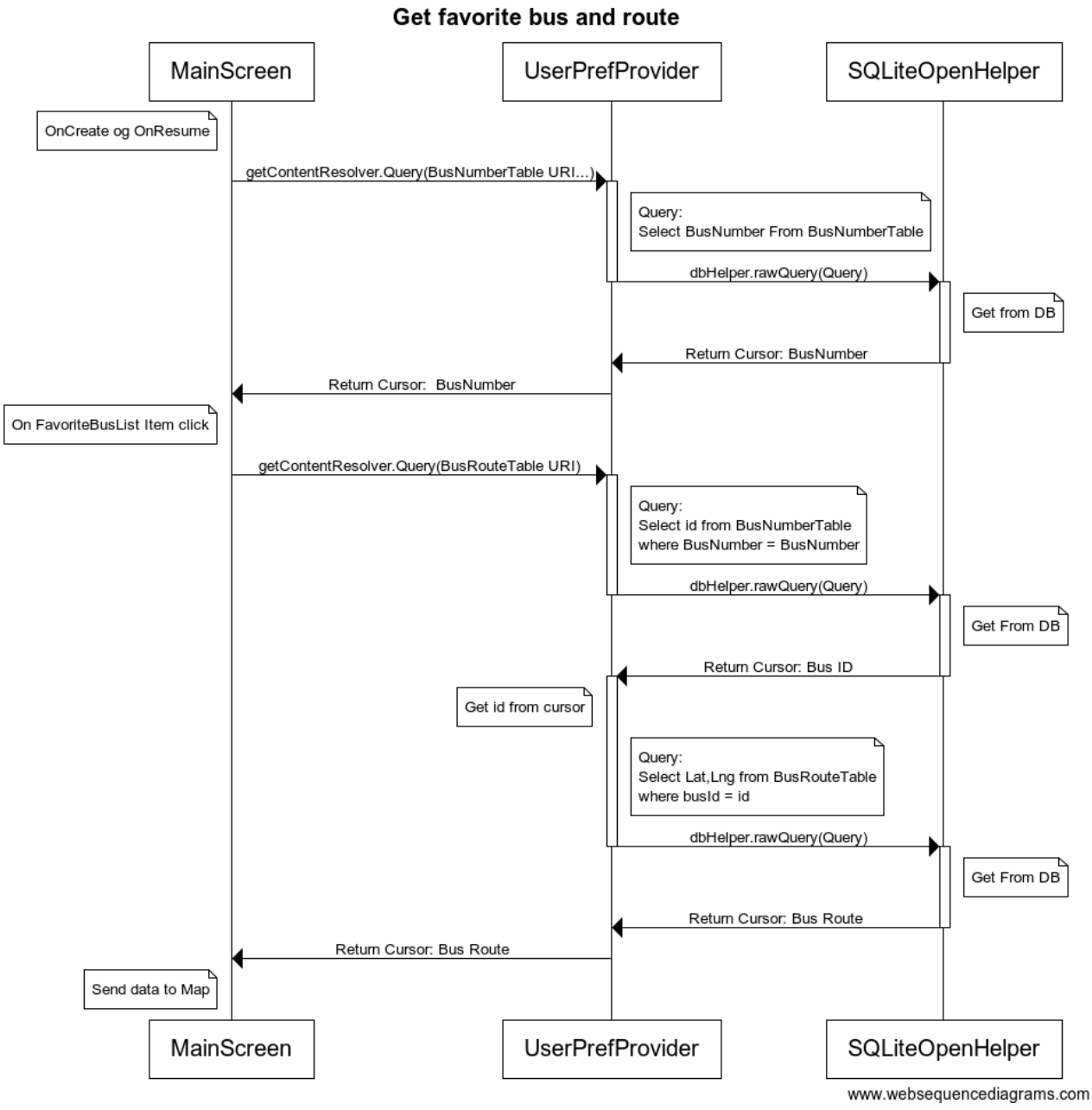
På de næste figurer kan man se de vigtigste nogle af de vigtigste dele af vores system. Dette drejer sig mere nøjagtigt om at hente fra- og skrive til databaser. Der er mange andre vigtige punkter i projektet, men de er ikke sekvensdiagrammer værdige, da de ofte bare snakker kort med sig selv. Det er datahentning og skrivning der er spændende her, da det spænder sig over mange dele af systemet.



Figur 5: Hentning af ruter fra Azure

På figur 4 kan det ses hvordan man tilgår vores eksterne database og får fat i en bus rute. Når man trykker på en rute på listen over busser, starter man Map activityet med det samme. Hentningen af data sker fuldstændigt asynkront, sådan at man samtidig kan sætte kortet op. Når map activityet starter, initialiserer den også en service som tager sig af tilgangen til databasen. Dette er dog ikke vist her, men bare vist som en MobileServiceClient. Når servicen initialiseres får den også en MessageHandler med sig. Denne er implementeret i BusMap aktivitet, og det vil være den der i sidste ende bliver svaret tilbage til. Servicen

starter denne ved en simpel connection string, samt en randomiseret kode, man skal hente fra sin Azure Managment side. Når BusMap kalder GetBusRoute sker der to ting. Først Henter servicen id'et på den givne bus fra dens busnummer, da id'et er foreign key i rute tabellen. Herefter henter den alle længdegrader og breddegrader som svarer til dette id. Herefter bliver disse længdegrader og breddegrader gemt i en Message, som bliver sendt til den passede MessageHandler. Herved kommer positionsdata over til BusMap asynkront. Når MessageHandleren i BusMappet modtager en message, vil den tegne en rute på den samtidige opsatte kort, med de punkter som blev hentet fra databasen.



Figur 6: Hente favorit busser og ruter fra SQLite

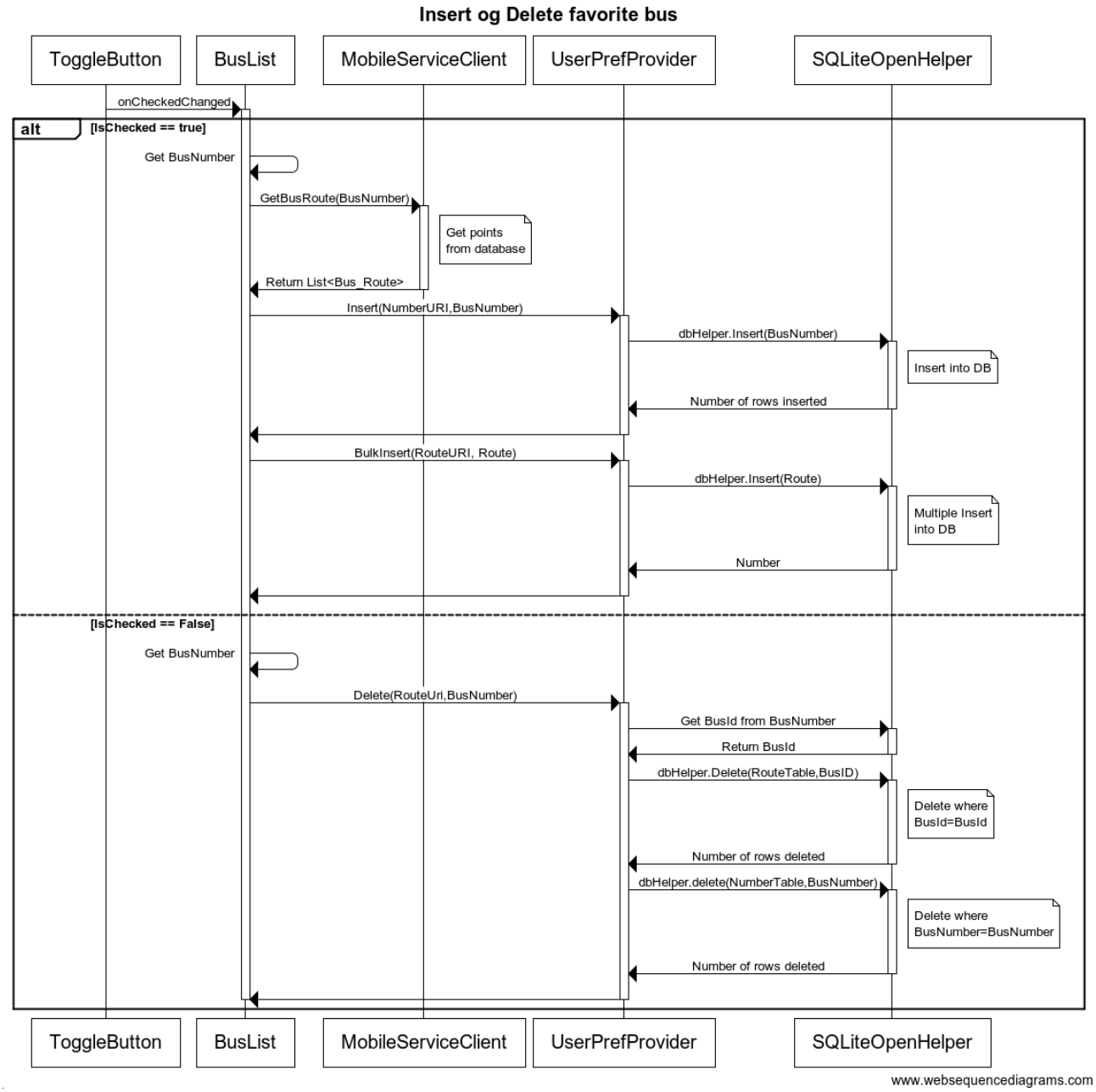
På figur 5 ses det, hvordan en valgt favoritbus og rute vil blive hentet fra SQLite Databasen. Dette er gjort på grund af, at hastigheden på at hente fra en lokal SQLite database er hurtigere end det er at hente fra en ekstern SQL database. Når MainMenu (Hovedskærmen) bliver startet, laver den et kald til getContentResolver.Query. Baseret på det URI man sender med, vil den finde den ContentProvider man skal bruge. I vores projekt er der to URIer man kan bruge, som begge ligger i to hjælpeklasser som statiske variabler. Alt efter hvilken URI man sender med, vil query funktionen gøre noget forskelligt. Til at starte med sender vi BusNumberURIet med. Dette fortæller ContentProvideren at den skal hente fra Bus numrene fra databasen. På ContentProvider siden bliver en en rawQuery sat op til at hente dette, og returne alle busserne som en Cursor. Da den hentede ContentProvider hentes som et interface må man følge de regler der er gældende, og på client siden selv omformatere fra en Cursor ved et kald til query. Cursoren vil indeholde alle busnumre, og vil blive sat som en liste på forsiden, så man nemt kan tilgå sine favoritter fra hovedskærmen. Grunden til at dette også sker på OnResume er, at brugeren kan få sin liste af favoritbusser opdateret, hver gang han returnerer til hovedskærmen. Hvis han netop har sat en bus til sin favorit og returnerer til hovedskæren vil onResume blive kaldt og listen opdateret.

Hvis der trykkes på en af de busser der er valgt som favorit, fra listen på hovedskærmen vil ruten blive hentet. Dette sker på samme måde med getContentResolver, men man sender nu BusRouteURIet med sammen med bus nummeret. ContentProvideren vil først lave et query, hvor den får fat i id'et på bagrund af busnummeret. Dette bruger den i den næste query, hvor den henter alle punkter der svarer til dette id. Den Cursor der returneres fra denne query, bliver returnet tilbage hovedskærmen og denne sætter så ruten på kortet.

På listen over alle busser er der mulighed for at vælge en bus til sin favorit. Dette gøres ved en ToggleButton på hvert item. På denne knaps onCheckedChanged event sker der to ting, afhængig af hvilken state knappen er i. Dette kan ses på figur 6.

Hvis IsChecked er true betyder det man har valgt at denne bus skal være en af ens favorit busser. Det første der sker i denne sammenhæng er, at hele ruten som svarer til den klikkede bus. Herefter vil der bliver lavet et kald til getContentResolver.insert(), hvor BusNumber URIet og busnummeret bliver givet med. Dette busnummer vil herefter blive sat ind i databasen, og antal linjer sat ind i databasen vil blive returnet. Herefter vil der blive kaldt et bulk insert med den downloadede rute. Bulk insert kalder bare insert flere gange. Når dette er færdigt vil antal linjer igen blive returnet og det totale antal linjer vil blive returnet til buslisten aktivitet. Når hele processen er fuldført vil et nyt busnummer og den tilhørende rute ligge i SQLiteDatabase.

Hvis IsChecked er false vil favoritbussen blive fjernet fra databasen. Dette sker ved et simpelt kald til getContentResolver.delete. Dette ene kald tager sig af at slette både ruten og busnummeret. Da ruten har en foreign key til bus nummer tabelen skal denne rute slettes først. Dette kræver dog at man først laver et query, på busnummer tabelen for at finde ud af hvilket id dette nummer har. Herefter vil ruten og busnummeret blive slettet og antal linjer slettet vil blive returnet.



Figur 7: Insert og Delete af favorit bus

Konklusion

Der er under dette forløb blevet arbejdede med at udvikle en smartphone applikation til android telefoner, denne opgave blev først beskrevet i vores synopsis, der beskrev i grove træk hvad vi havde tænkt os at lave, samt en skitse af et Use Case diagram. Der er herefter blevet arbejde ud fra denne synopsis for at udvikle vores applikation. Vi valgte at lave en applikation der kunne følge en valgt bus i Aarhus, vise dens nuværende position samt indtegne hele dens rute, Der blev brugt Googles maps API til tegne kortet. Valget af dette projekt blev gjort ud fra vores bachelor opgave, hvor der også skal udvikles en lignende smartphone applikation, Vi har derfor brugt dette tema projekt til at udvikle en grov skitse af vores bachelor applikation, som vi nemt kan videreudvikle på.

Der blev hurtigt valgt at bruge en SQL database, til at gemme information på de forskellige busser samt deres ruter, dette gør det nemt at tilføje flere busser og ændre i ruter fra et potentielt administrations program senere. Der blev også lavet en SQLite database, så det ville være muligt at gemme nogle bruger specifikke oplysninger, dette blev brugt til at gøre det muligt for brugeren at sætte nogle bus ruter som favorit ruter, dette vil cache hele busruten lokalt, så der ikke skal bruges mobil-data for at downloade ruten hver gang den skal vises, samtidigt med at bussen vil blive vist på start skærmen som den er nem at vælge. Der blev sat et maksimum på 6 bus ruter der kan caches, da det passede med en rimelig størrelse at gemme lokalt, samt det passede med antal linjer der var nemt at tilgå på startskærmen.

For at vise en bus der bevæger sig på sin rute, gemmer SQL databasen en række GPS koordinater, til forskellige nøglepunkter på ruten, disse bliver tegnet ind på kortet og forbundet, hvorefter bussens bevægelse blive simuleret, som alternativ til at have en GPS på bussen og modtage koordinator derfra, dette bliver gjort simpelt ved at flytte en marker mellem alle de gemte nøgle punkter på ruten, for at få det til at ligne en bus i bevægelse.

Sat sammen giver dette vores applikation: 'TrackABus', den længe ventede applikation til alle busgænger i Aarhus, nu ved man altid hvor lang tid der er til bussen er ved stoppestedet, eller om man lige har misset den, og kan nå ind og få en hotdog.