TrackABus

BACHELORPROJEKT

Systemarkitektur for TrackABus

Author:

Gruppe 13038

Supervisor:

Michael Alrøe

17. december 2013

Versionshistorie:

Ver.	Dato	Initialer	Beskrivelse
0.1	18-11-2013	13038	Arbejde påbegyndt
0.2	03-12-2013	13038	Use Case View færdiggjort
1.0	11-12-2013	13038	Første version udfærdiggjort

Godkendelsesformular:

Forfatter(e):	Christoffer Lousdahl Werge (CW) Lasse Sørensen (LS)
Godkendes af:	Michael Alrøe.
Projektnr.:	bachelorprojekt 13038.
Filnavn:	Systemdesign.pdf
Antal sider:	144
Kunde:	Michael Alrøe (MA).

Sted og dato: _____

10832 Christoffer Lousdahl Werge

MA Michael Alrøe

09421 Lasse Lindsted Sørensen

Indhold

1 INTRODUKTION

Dette dokument beskriver designet og arkitekturen af softwaren for bachelorgruppe 13038 - TrackABus. Softwaredesignet og arkitekturen er opbygget ud fra Use Cases fundet i *TrackABus_Kravspecifikation*.
I dette dokument findes blandt andet:

- En systemoversigt samt dets grænseflader
- Use Case view
- Logisk view
- Proces og task view
- Deployment view
- Implementerings view
- Data view

Dokumentet er obygget på 4+1 arkiktetur modellen, hvori der er tilføjet et ekstra view til data. I disse forskellige afsnit kan der findes domænemodel, klassediagrammer, sekvensdiagrammer og lignende. Derudover findes diverse illustrationer, formler og kodeeksempler.
Alle disse elementer samt øvrige dele, er med til at beskrive systemet i detaljer.
Figurer og illustrationer kan findes i fuld størrelse på bilags CDen. Her kan fuld implementering også findes.

1.1 Formål

Formålet med dokumentet er:

- At fastlægge systemets overordnede design, arkitektur og virkemåde. Dette er gjort jævnfør kravspecifikationen.
- At definerer og beskrive de nødvendige klasser og vise samspillet mellem disse.
- At give individer med tilpas faglig viden indblik i systemets opbygning.

Dokumentets målgruppe er en person med grundlæggende viden om det overordnede faglige emne, dog uden viden inden for det specifikke emne.

1.2 Referencer

I dette dokument vil der optræde referencer til følgende dokumenter:

- TrackABus_Kravsspecifikation

Ovenstående dokument kan findes på medfølgende bilags CD.

1.3 Definitioner og forkortelser

- GUI - Graphical User Interface.
- UI - User Interface.
- WPF - Windows Presentation Foundation.
- MVC - Model - View - Controller.
- API - Application programming interface.
- XML - Extensible Markup Language.
- SOAP - Simple Object Access Protocol.
- JSON - JavaScript Object Notation.
- AJAX - Asynchronous JavaScript and XML.
- GPS - Global Positioning System.
- UML - Unified Modeling Language.
- OO - Objektorienteret.
- ER - Entity relationship.
- CRUD - Create, read, update and delete.
- URI - Uniform resource identifier.

- Simpel rute - En busrute som spænder to endestationer.
- Kompleks rute - En busrute som spænder mere end to endestationer.

1.4 Dokumentstruktur og læsevejledning

Afsnit 1 – INTRODUKTION: Introducerer læseren til projektet og dokumentet.

Afsnit 2 - SYSTEM OVERSIGT: Dette afsnit giver en kort oversigt over systemet og dets omgivelser. Derudover vises der diagrammer over systemets aktører, samtidig med at systemet kort beskrives.

Afsnit 3 – SYSTEMETS GRÆNSEFLADER: Her beskrives grænsefladerne til de forskellige aktører til systemet.

Afsnit 4 – USE CASE VIEW: Afsnittet beskriver alle Use Cases og Use Case scenarier fra Use Case modellen. De forskellige Use Cases beskrives i prosa form, samtidig med at et Use Case diagram præsenteres.

Afsnit 5 – LOGISK VIEW: Beskriver systemets opdeling i delsystemer og pakker og deres organisering i en lagdelt struktur. Viser hvordan de forskellige Use Cases er realiseret, blandt andet ved brug af simple- og udvidede systemsekvensdiagrammer.

Afsnit 6 – PROCES/TASK VIEW: Dette afsnit beskriver systemets opdeling i processer og tråde, og hvorledes disse processer kommunikerer og synkroniserer. Giver et overblik over de forskellige tråde anvendt i systemet.

Afsnit 7 – DEPLOYMENT VIEW: Viser den fysiske struktur af systemet, hvilket er computere og andre hardware enheder.

Afsnit 8 – IMPLEMENTERINGS VIEW: Dette afsnit beskriver den endelige implementering. Her er de forskellige komponenter beskrevet i detaljer.

Afsnit 9 – DATA VIEW: Er en beskrivelse af persistent data lagring i systemet.

Afsnit 10 – GENERELLE DESIGNBESLUTNINGER: Dette afsnit fastholder de generelle designbeslutninger, der tages under arktiekturdesignet. Her beskri-

ves systemets begrænsninger, anvendte designmønstre, samt anvendte værktøjer og (kode)biblioteker.

Afsnit 11 – STØRRELSE OG YDELSE: I dette afsnit er angivet de kritiske størrelser og ydelsesparametre for systemet.

Afsnit 12 – KVALITET: Her opremses de kvalitetskrav for systemet der er med til at udforme arkitekturen. Ligeså beskrives hvorledes arkitekturen opfylder andre af de ikke-funktionelle krav vedrørende for eksempel udvidbarhed.

afsnit 13 - OVERSÆTTELSE: Her beskrives der hvordan der kommes fra kildeteskt til objektkode, og hvordan systemet installeres

Afsnit 14 – KØRSEL: Her beskrives hvordan de forskellige dele af systemet køres, samt en forklaring på de forskellige fejlbeskeder der kan forkomme.

Afsnit 15 – BILAG: Liste over anvendte bilag.

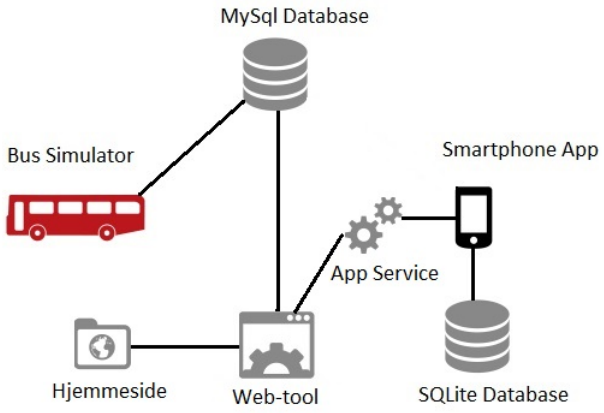
1.5 Dokumentets rolle i en iterativ udviklingsproces

Da der er blevet fulgt en iterativ udviklingsprocess er systemet blevet udviklet over en række iterationer. I hver iteration er selve TrackABus mobil applikationen, administrations hjemmesiden og simulatoren blevet videreudviklet, og har fået flere funktionaliteter samtidig med at systemet er blevet effektiviseret. At processen er en iterativ udviklingsproces kommer til udtryk ved, at dokumentation er opdateret løbende i takt med, at softwaren har udviklet sig, og større erfaring blev opnået.

2 SYSTEM OVERSIGT

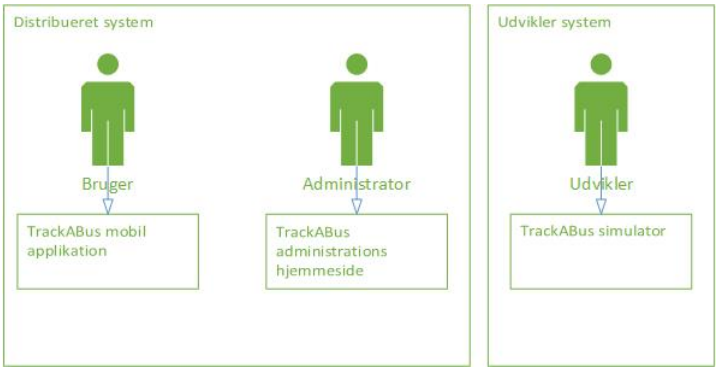
2.1 System kontekst

Systemet består af en android mobil applikation, en administrations hjemmeside og en bus simulator. Systemet kan skitseres som på figur 1.



Figur 1: Simpel systemoversigt

Systemets påvirkninger fra omverden, kan overskueliggøres via et aktørkontekst-diagram vist på figur 2.



Figur 2: Aktørkontekst-diagram

2.2 System introduktion

Systemet består af to hoveddele, en mobil applikation og en administrations hjemmeside. Mobil applikationens hovedfunktion er at gøre det muligt for brugeren at se en valgt busrute, rutens stoppesteder og alle busser der kører på ruten, indtegnet på et kort. Desuden

er det muligt at få vist, hvor lang tid der er, til den næste bus ankommer til et valgt stoppested. Det er også muligt at favorisere de forskellig busruter. Dette vil gemme busruten lokalt på mobil telefonen, hvorefter der ikke længere bruges mobil data til at downloade ruten, hver gang den skal bruges.
Hoved funktionaliteten for administrations hjemmesiden er at gøre det nemt at tilføje en ny busruter med stoppesteder til systemet, samt gøre det muligt at ændre allerede eksisterende busruter. Alt dette sker igennem en GUI på TrackABus hjemmsiden.[1]

Der er også blevet udviklet en simulator der har til formål at simulere en til flere busser, der kører på de forskellig busruter. Simulatoren er ikke en del af hovedsystemet da den kun er lavet til udviklingsformål, og derfor ikke beskrevet i Use Cases. Det er muligt at tilgå simulatoren igennem en GUI, hvorfra en udvikler kan starte forskellige simulationer. Det er nødvendigt med en simulator, da der skal være mulighed for at teste forskellige situationer og algoritmer, hvor man ikke kan være afhængig af en rigtig bus. *Se afsnit 8.2.4 Komponent 4: Simulator* for en dybere beskrivelse af simulatoren.

[1] Administrationsside: www.TrackABus.dk

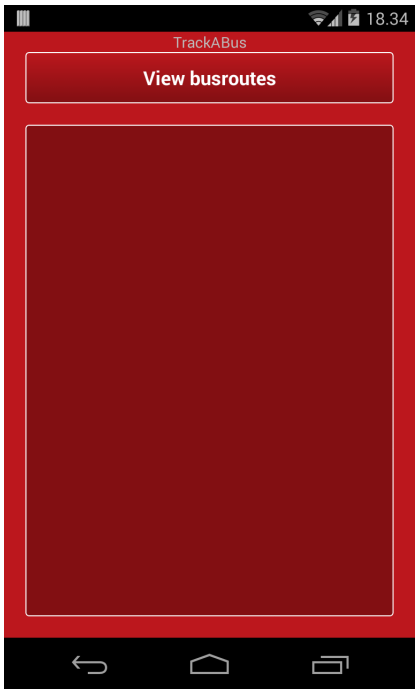
3 SYSTEMETS GRÆNSEFLADER

3.1 Grænseflader til person aktører

Systemet har to forskellige personaktører, der begge virker som primæraktører.[2] Disse to tilgår forskellige dele af systemet, og derfor benytter de hver især forskellige brugergrænseflader. Bruger aktørern tilgår mobil applikationen, og administrator aktøren tilgår administrations hjemmesiden.

3.1.1 Bruger

Når mobil applikationen startes bliver brugeren præsenteret for brugerfladen, der ses på figur 3. Herfra kan brugeren trykke på knappen 'View busroutes' hvis han ønsker at få vist en liste af busruter der findes i systemet.

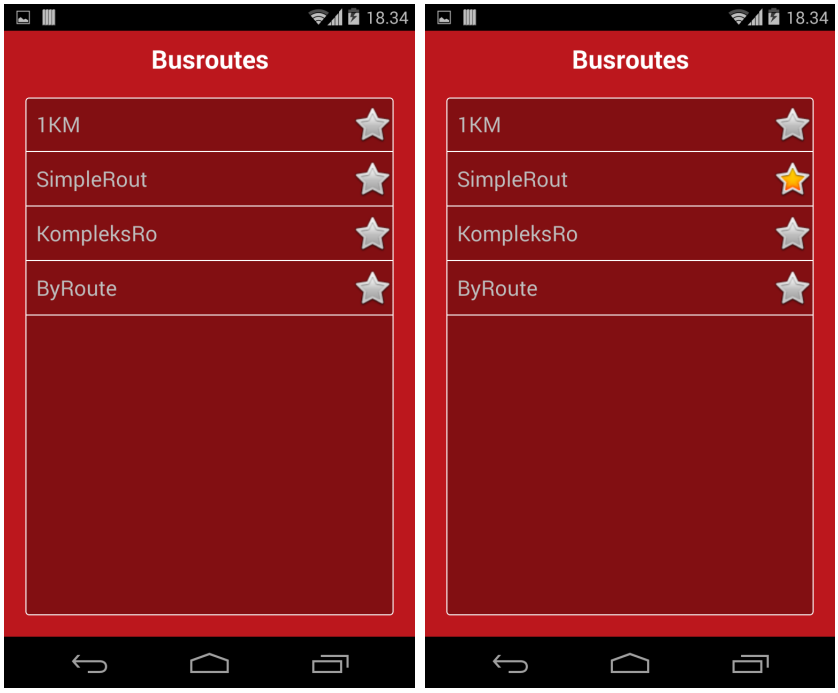


Figur 3: Mobil applikationens start skærm

Trykker brugeren på knappen 'View busroutes' vil der blive præsenteret en liste af de forskellige busruter der er gemt i den distribuerede database. Denne skærm kan ses på figur 4 til venstre. Herfra er der enten mulighed for at vælge en af de viste busruter, eller trykke på en grå stjerne for at favorisere en busrute. Bliver der trykket på stjernen, vil

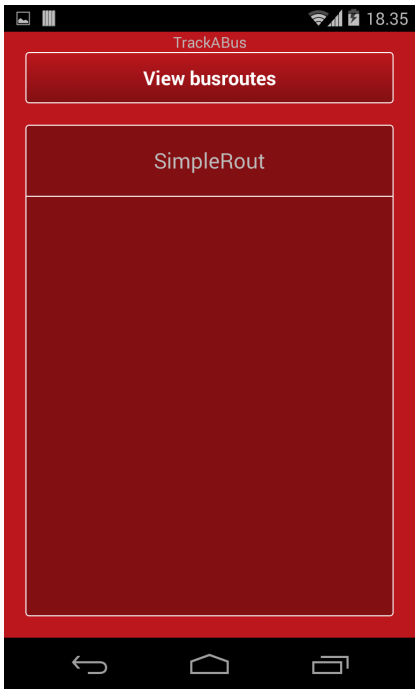
[2] Aktører kan findes i afsnit *4: Use Case View*, på figur **??**

systemet favorisere den valgte rute, gemme den i den lokale database og stjernen vil blive gul, som vist på figur 4 til højre. Trykkes der på en gul stjerne, ufavoriseres den valgte rute. Den bliver slettet fra den lokale database og stjernen bliver grå.



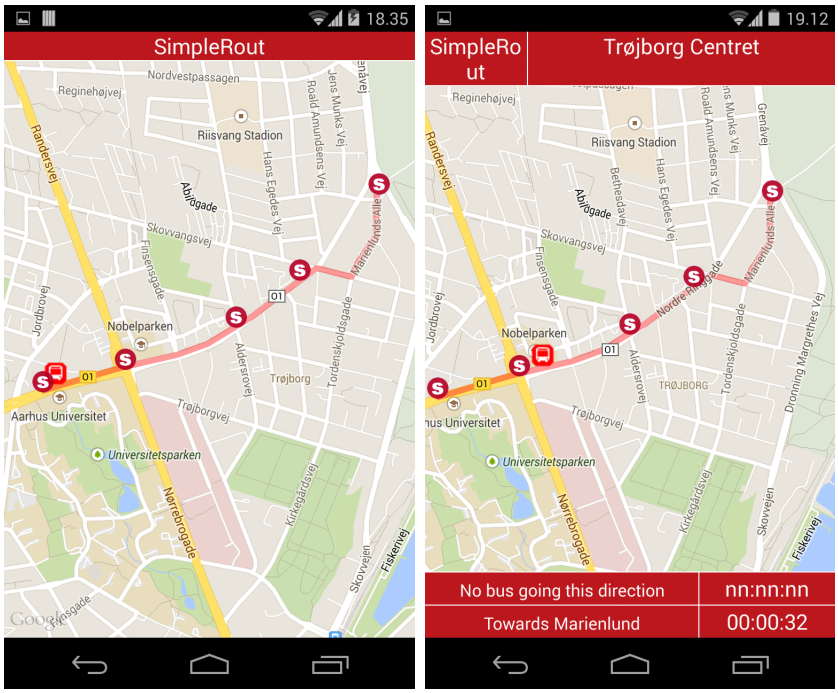
Figur 4: Listen af alle busruter, der findes i den distribuerede databasen, hvor der på det billedet til højre er en busrute som er blevet favoriseret

Hvis en busrute er favoriseret er den tilgængelig fra startskærmen. Dette medfører, at den er hurtigere at tilg. På figur 5 kan det ses hvordan startskærmen ser ud, efter en busrute er blevet favoriseret



Figur 5: Startskærmen, der nu viser den busrute, der er blevet favoriseret

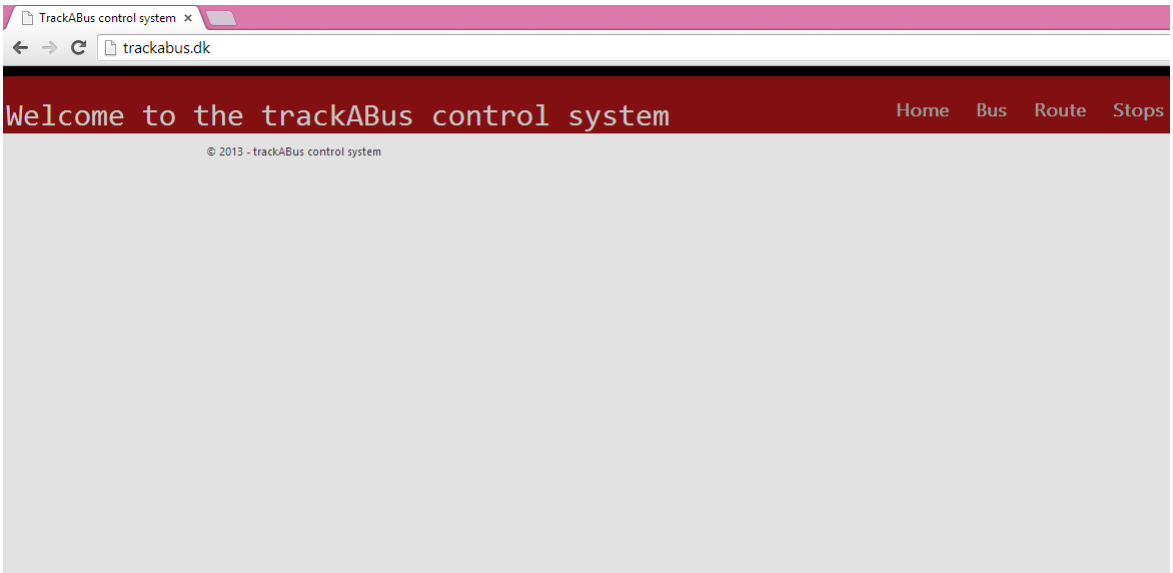
Hvis brugeren vælger en busrute, fra listen af alle busruter på skærmen vist på figur 4, eller fra favorit listen vist på figur 5, vil brugeren nu få vist et kort. Dette se på figur 6 til venstre. Herpå er der indtegnet den valgte busrute, dens stoppesteder og de busser der kører på ruten. Stoppestederne er indtegnet med et tydeligt 'S' og busserne er indtegnet med et busikon. Brugeren kan nu vælge et af stoppestederne. Ved valgt stoppested, vil kærmen blive opdateret til at vise information om det valgte stoppested, samt hvor lang tid der er til den næste bus ankommer ved det valgte stoppested. Hvis der ikke er nogen busser på vej imod det valgte stoppested vil teksten: 'No bus going this direction' blive vist, ellers vil der blive vist navnet på den endestation bussen køre imod. Dette kan ses på figur 6 til højre.



Figur 6: Kort med indtegnet busrute, stoppesteder og en bus. Til højre er stoppestedet "Trøjborg Centret"valgt

3.1.2 Administrator

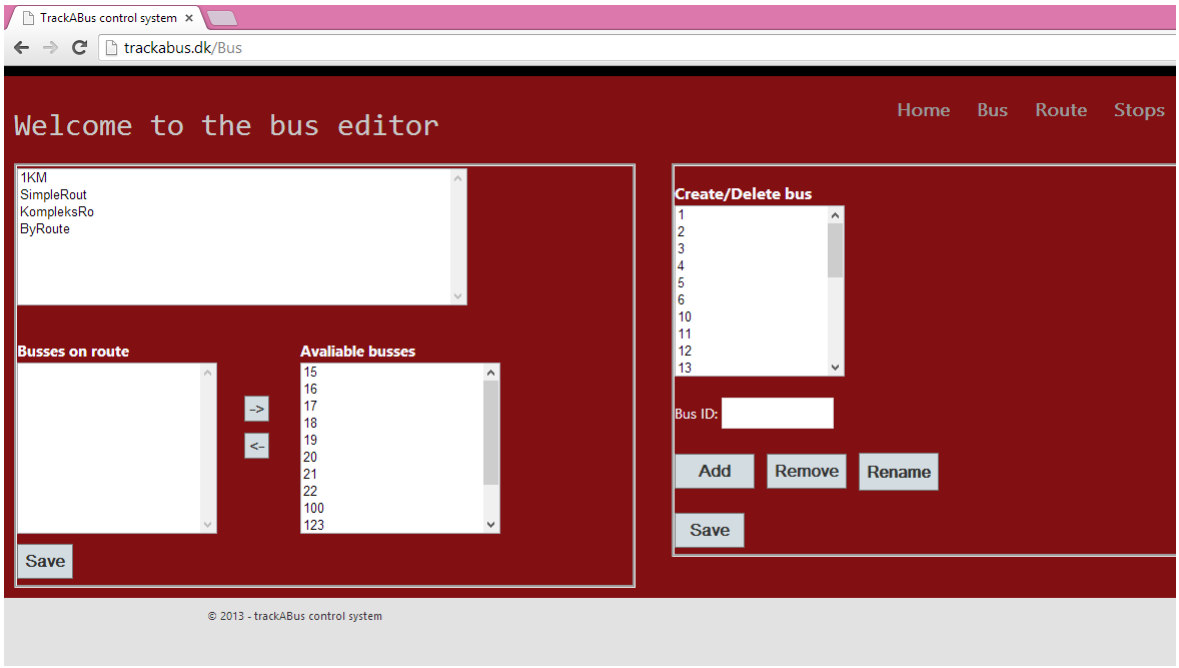
Når administratoren er gået ind på administrations hjemmesiden[3] vil han få vist siden, som ses på figur 7. Dette er startskærmen for administrations hjemmesiden. Herfra er det muligt at navigere til de forskellige dele af systemet.



Figur 7: Startskærmen for administrations hjemmesiden

[3]Hjemmeside : www.trackabus.dk

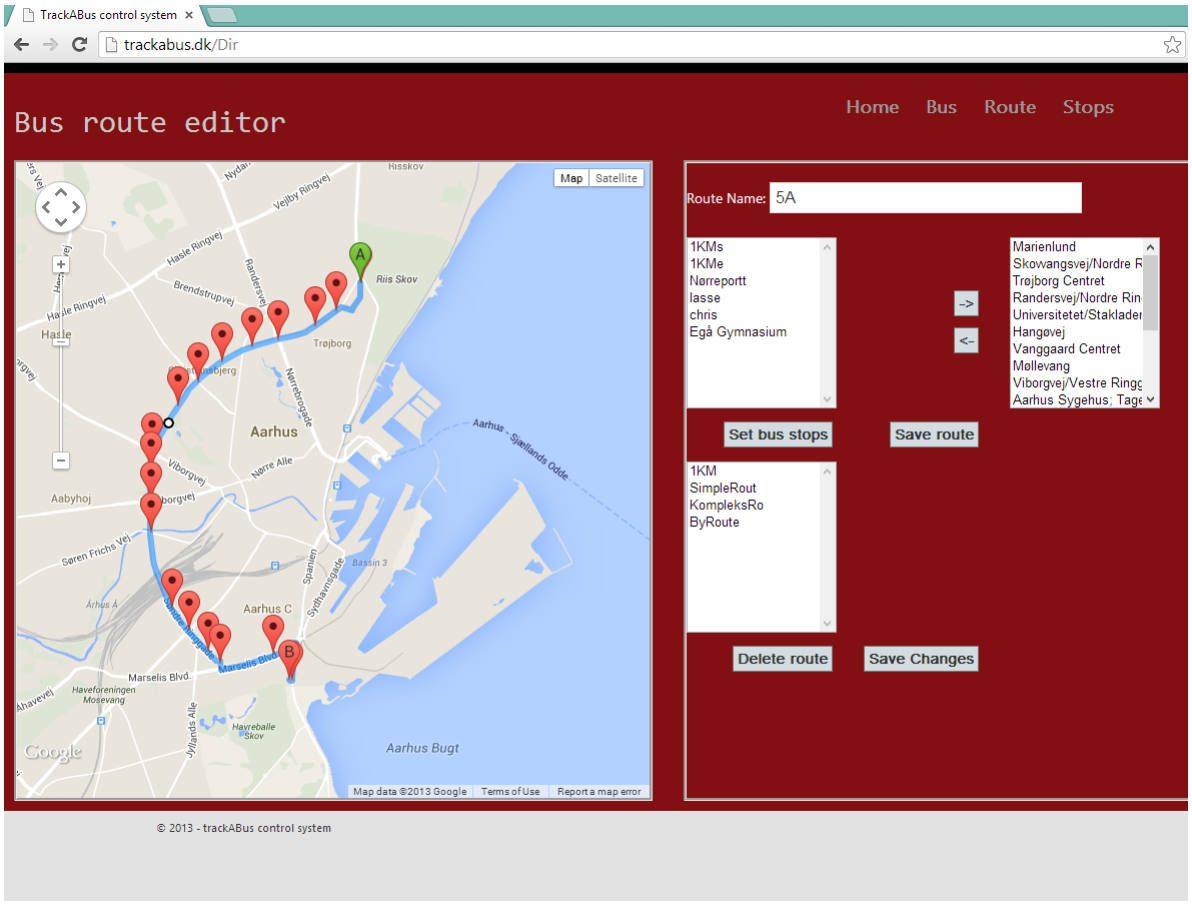
Vælger administratoren at trykke på knappen 'Bus' vil der blive navigeret til bus redigerings siden som ses på figur 8. Herfra er det muligt at vælge en busrute i den øverste liste på venstre side af skærmen. Dette vil gøre det muligt at tilføje busser til, eller fjerne busser fra, den valgte rute. For at tilføje en bus til ruten, vælges en bus, fra listen "Available busses"og der trykkes på knappen «-". Dette vil tilføje en bus til den valgte rute.
For at fjerne en bus på den valgte rute, vælges der en fra listen "Busses on route", hvorefter der trykkes på knappen ->"Dette vil fjerne bussen fra den valgte rute. For at gemme ændringerne trykkes der på knappen save.
Den højre side af skærmen omhandler redigering af bus information. Herfra vil det være muligt for administratoren at tilføje, fjerne og ændre information om busser. For at tilføje en bus skrives dennes ID i feltet "Bus ID". Herefter trykkes der på knappen "Add", hvorefter bussen tilføjes til listen. For at fjerne en bus, vælges der en fra listen, og der trykkes remove, hvorefter dette ID vil fjernes fra listen. For at redigere ID'et for en bus, vælges der igen en fra listen, hvorefter der skrives et nyt ID og "Rename"knappen trykkes. Når ændringer er foretaget, trykkes "Save knappen"for at gemme disse på den distribuerede database.



Figur 8: Skærmen for bus redigerings siden

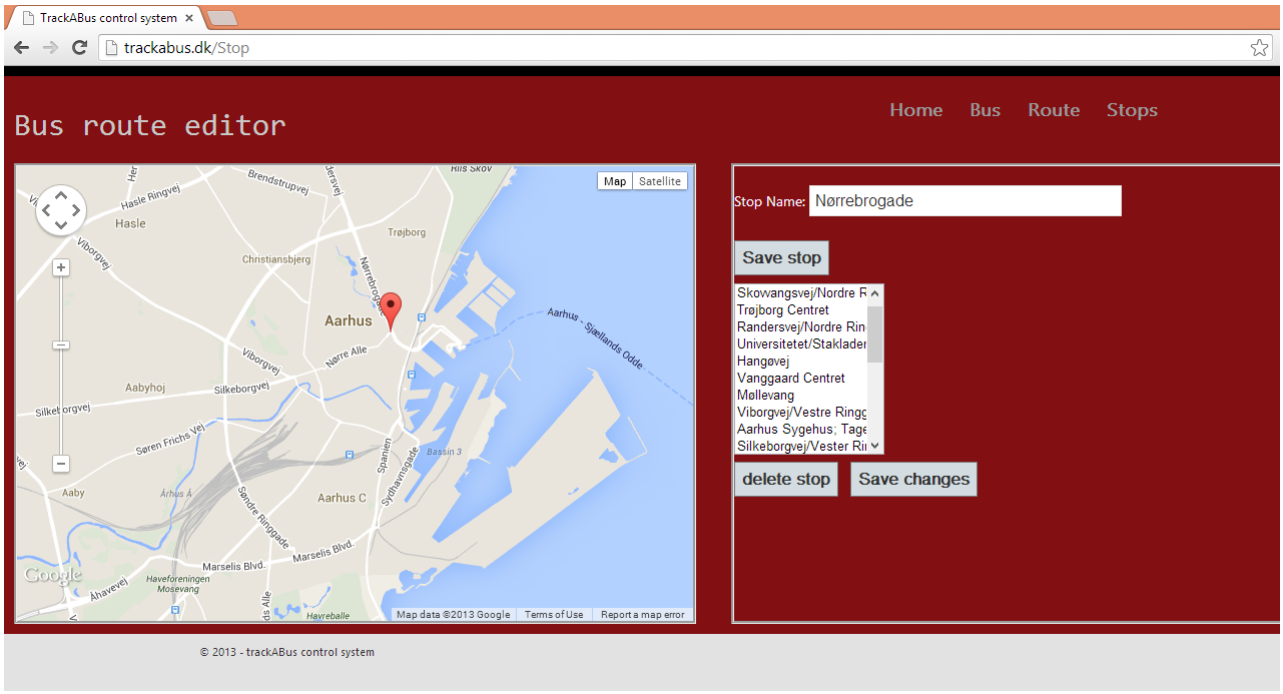
Vælger administratoren at trykke på knappen 'Route' vil der blive navigeret til rute redigerings siden, som ses på figur 9. Her er der mulighed for at tilføje, fjerne eller redigere en

busrute, samt tilføje og fjerne stoppesteder til denne. Dette bliver gjort ved, at der vælges et start og slut punkt for ruten på kortet. På figur 9 kan en rute ses indtegnet på kortet, hvor start punktet er punkt 'A' og slut punktet er punkt 'B'. For at sætte stoppesteder på ruten, vælges de fra den øverste liste til venstre, hvorefter der trykkes på knappen ->". Dette vil tilføje stoppestederne til ruten. Hvis stoppesteder vælges fra den øverste liste til højre, hvorefter der trykkes på knappen «-", vil de valgte stoppesteder fjernes fra ruten. Når et antal stoppesteder er tilføjet til ruten, kan der trykkes på knappen "Set bus stops", hvorefter stoppestederne vises på kortet. Når ruten er færdig kan der trykkes på knappen "Save route", hvorefter ruten bliver gemt i den distribuerede database. Administratoren har også mulighed for at redigere i en rute. Dette gøres ved at vælge en fra den nederste liste, som når den bliver valgt, vil blive indtegnet på kortet. Administratoren kan nu redigere rutens forløb, tilføje og fjerne bustoppesteder og til sidste gemme ændringerne på databasen, ved at trykke på "Save Changes"knappen. Hvis en valgt rute ønskes slette, trykkes der på "Delete route"knappen.



Figur 9: Skærmen for rute redigerings siden, her med indtegnet busrute 5A, med stoppesteder

Vælger administratoren at trykke på knappen 'Stops' vil der blive navigeret til busstop redigerings siden som ses på figur 10. Herfra er det muligt at oprette nye stoppested i systemet, fjerne stoppesteder, eller ændre navn og position for allerede eksisterende stoppesteder. For at tilføje et nyt stoppested, trykkes der et sted på kortet hvor et stoppested ønskes tilføjet, hvorefter navnet blivet indtastet i feltet "Stop Name". For at gemme stoppestedet på den distribuerede database, trykkes der på knappen "Save stop". Når et stoppested vælges fra listen, indtegnes dennes position på kortet. Herefter er det muligt at ændre positionen og navn på stoppestedet. Når ændringer er foretaget, trykkes der på knappen "Save changes", hvorefter disse gemmes i den distribuerede database. Hvis der efter stoppestedsvalg, trykkes på knappen "Delete stop", vil stoppestedet slettes.



Figur 10: Skærmen for busstoppested redigerings siden, her med indtegnet stoppested "Nørrebrogade"

3.2 Grænseflader til eksternt software

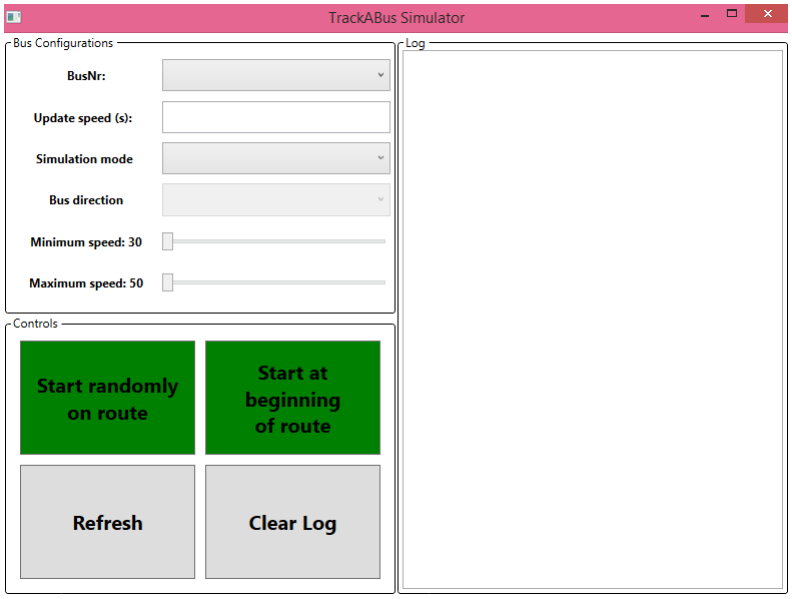
Her beskrives grænsefladerne for de software delsystemer, der er blevet udviklet som separate projekter.

Der er blevet udviklet en simulator sideløbende med projektet, da det blev nødvendigt at have mulighed for at simulere en bus der kører på en rute. Et andet formål med simulatoren er at gøre det muligt at teste forskellige situationer, der kan forekomme i systemet.

3.2.1 Simulator

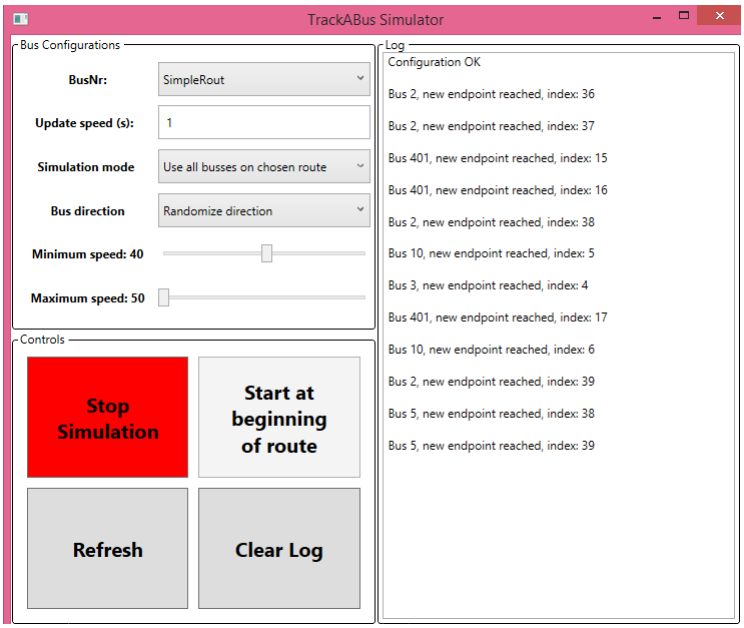
Når simulatoren bliver åbnet vil grænsefladen se ud, som vist på figur 11. Herfra kan der ved "BusNr:"vælges hvilken busrute, busserne skal køre på. Kun ruter med tilknyttede busser vises. "Update speed"bestemmer hvor ofte positionen for busserne skal opdateres i sekunder, hvor '1' er minimumsværdien. I "Simulation mode", vælges der hvilken form for simulation der skal startes, dette kan være "Use single bus on route", "Use all busses on chosen route"eller "Use all busses in system". "Use single bus on route"vil simulere en enkelt bus der kører på den valgte rute, 'Use all busses on chosen route' vil simulere alle de busser der er knyttet til den valgte rute og 'Use all busses in system' vil simulere

alle busser i hele systemet på deres tilknyttede rute. Ved valg af sidste simulerings måde, er ligemeget hvilken busrute der blev valgt i "BusNr". Under "Bus direction"kan der vælges om busserne skal køre i en bestemt eller tilfældig retning. Til sidst kan der vælges hvor hurtig busserne skal køre. Her er der mulighed for at sætte hastigheden på busserne imellem 30 og 1000 km/t. Simulatoren kan nu startes på to forskellige måder; Enten hvor busserne alle starter tilfældige steder på deres rute, eller om alle busserne skal starte fra rutens startpunkt.



Figur 11: Grænseflade for simulator

På figur 12 kan det ses, hvordan grænseflade kan se ud efter simulerings start. I højre side ses et logvinde, hvor der forekommer opdateringer fra busserne, samt eventuelle fejlbeskeder. De to sidste knapper på simulatoren, "Refresh"og "Clear Log"bruges henholdsvist til at hente de nyeste ruter fra den distribuerede database, og til at slette alt hvad der står i log vinduet.



Figur 12: Grænseflade for simulator efter den er blevet configureret og startet

4 USE CASE VIEW

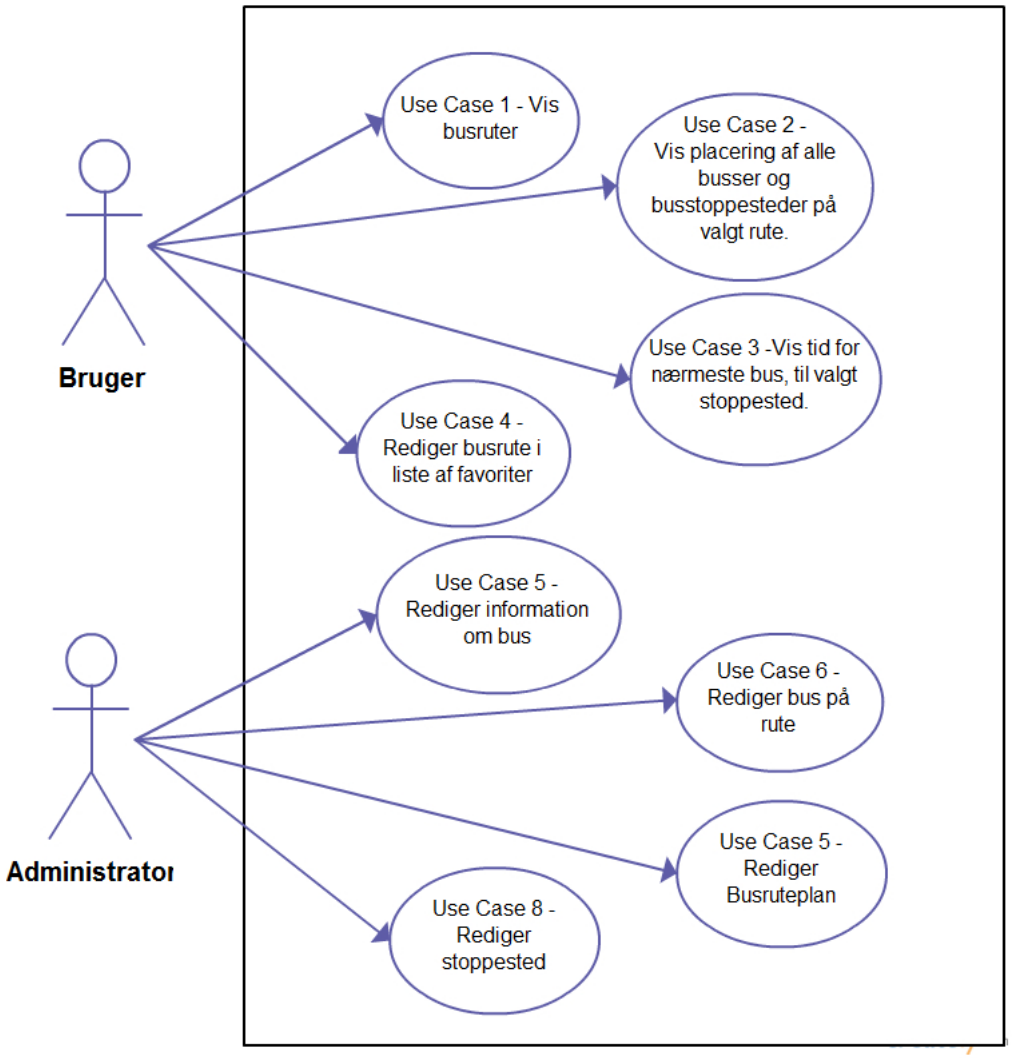
I dette afsnit forklares, hvordan Use Case view'et er sat op, samt hvad de forskellige Use Cases gør.

4.1 Oversigt over arkitektursignifikante Use Cases

I dette afsnit er de enkelte Use Cases præsenteret. Use casene beskriver udelukkende mobil applikationen samt det online adminstrations værktøj. Den distribuerede database, den lokale database, samt simuleringsværktøjet anses som interresanter for systemet, men indgår ikke som aktører. Tilgangen til databaserne beskrives i *afnist 9.2: Implementering af persistens*, og simulatoren beskrives i *8.2.4: Komponent 4: Simulator*.
Use Casene i systemet er som følgende:

- Use Case 1: Vis Busruter
- Use Case 2: Vis Placering af alle busser og stoppesteder på valgt rute
- Use Case 3: Vis tid for nærmeste bus, til valgt stoppested
- Use Case 4: Rediger busrute i liste af favoriter
- Use Case 5: Rediger information om bus
- Use Case 6: Rediger bus på rute
- Use Case 7: Rediger busruteplan
- Use Case 8: Rediger stoppested

Som det fremstår af Use Case diagrammet, figur 13, er der udelukkende to aktører; brugeren og administratoren. Selvom samtlige Use Cases kommunikerer med den distribuerede database, er det blevet vedtaget, at databasen blot er en interresant og ikke en sekundær aktør. Use case diagram kan findes på bilags CDen under Diagrammer/Use Case Diagram



Figur 13: Use Case diagram

Use Case 1 til 4 er relateret til mobil applikationen og er derfor initieret af brugeren. Ligeledes er Use case 5 til 8 relateret til server-side operationer og således initieret administratoren. Brugeren kan kun tilgå databasen i læsnings-øjemed, mens administratoren både kan skrive og læse. Brugerne kan dog, hvis han ønsker, gemme dele af læst data lokalt. Use Cases for brugeren er derfor mere visuelle, end de er redigerende, hvor Use Cases for administratoren er meget mere redigerende. Der er intet overlap mellem administratoren og brugeren, således at den grafiske brugergrænseflade for administratoren udelukkende skal bruges af administratoren, og mobil applikation kun af brugeren.
Brugeren og administratorens Use Cases er tæt koblet til deres respektive grafiske brugergrænseflade, da alle Use Cases initieres igennem disse. Eksempler på dette kan ses i *afsnit 3.1: Grænseflader til person aktører*

4.2 Use Case 1 scenarier - Vis busruter

4.2.1 Use Case mål

Målet med denne Use Case er at få vist, på mobil-applikationen, en liste over alle busruter der er gemt i den distribuerede database.

4.2.2 Use Case scenarier

Denne Use Case viser en liste over busruter, der er gemt i databasen, til brugeren. Det kræver at brugeren står ved startskærmen, hvorefter brugeren tilkendegiver overfor systemet, at der ønskes at ses listen over gemte busruter. Herefter hentes busruterne fra databasen og brugeren bliver præsenteret for disse.

4.2.3 Use Case undtagelser

Da busruterne bliver hentet fra en database, er der risiko for, at forbindelsen til databasen mistes. Hvis dette sker, vil brugeren blive præsenteres for en besked om, at det ikke er muligt at etablere forbindelse til databasen, hvorpå han kan vende tilbage til startskærmen og prøve igen. Der er mulighed for at annullere indlæsningen. Hvis dette sker vil systemet stoppe indlæsningen fra databasen, samt returnere til startskærmen. systemet kan gå i dvale, imens der bliver indlæst fra databasen. Hvis dette sker vil systemet hente busruterne færdig i baggrunden.

4.3 Use Case 2 scenarier - Vis placering af alle busser og busstoppesteder på valgt rute

4.3.1 Use Case mål

Målet med denne Use Case er at få vist et kort, med indtegnet busrute, stoppesteder samt busser der kører på ruten.

4.3.2 Use Case scenarier

Denne Use Case viser et kort til brugeren, med indtegnet busrute, stoppesteder samt busser der kører på valgt rute. Det kræver at brugeren står ved listen over busruter, eller

startskærmen, hvsi en rute er favoriseret. Dernæst tilkendegives der overfor systemet, hvilken busrute der ønskes vist. Systemet henter busruten, samt stoppestederne fra databasen. Herefter bliver brugeren præsenteret for et kort, med indteget busrute og stoppesteder. Systemet henter nu gps-koordinaterne for busserne på ruten, samt indtegner dem på kortet. Efter to sekunder vil systemet igen hente gps-koordinaterne, og opdatere bussernes position. Systemet vil forsætte med at opdatere positionerne indtil brugeren tilkendegiver overfor systemet, at dette ikke længere ønskes.

4.3.3 Use Case Undtagelser

Da busruten, stoppestederne samt bussernes gps-koordinater bliver hentet fra en database, er der risiko for, at forbindelsen til databasen mistes. Hvis dette sker, når systemet henter busruten og stoppestederne, vil brugeren præsenteres for en besked om, at det ikke er muligt at etablere forbindelse til databasen. Hvis dette sker når systemet henter gps-koordinaterne vil brugeren præsenteres for en besked om, at det ikke er muligt at opdatere bussernes position. Det vil dog stadigvæk være muligt at se kortet, med indtegnet rute og stoppesteder, men bussernes position vil ikke længere opdateres. Der er mulighed for, at systemet genetabler forbindelse til databasen. Hvis dette sker, vil systemet igen opdatere bussernes position.

4.4 Use Case 3 scenarier - Vis tid for nærmeste bus, til valgt stoppested

4.4.1 Use Case mål

Målet med denne Use Case er at få vist tid til ankomst, for den bus der er tættest på det valgte busstoppested i begge retninger.

4.4.2 Use Case scenarier

Før denne Use Case kan startes, skal *Use Case 2: Vis placering af alle busser og stoppesteder på valgte rute* være gennemført. Brugeren vælger et stoppesteder, der er indtegnet på kortet. Systemet udregner nu den tid det vil tage, før den nærmeste bus ankommer til det valgte stoppested i begge retninger. Der hentes desuden information om det valg-

te stoppested fra databasen. Herefter bliver brugeren præsenteret for ankomstiden, samt information om valgt stoppested.

4.4.3 Use Case Undtagelser

Da gps-koordinaterne, information om stoppesteder samt udregningen for ankomsttiden til valgt stoppested, bliver hentet fra en database, er der risiko for, at forbindelsen til databasen mistes. Hvis dette sker, vil brugeren præsenteres for en besked om, at det ikke er muligt at etablere forbindelse til databasen. Herefter vil tiden til ankomst ved valgt stoppested ikke længere holdes opdaret, men vil genstartes ved reetablering af forbindelsen.

4.5 Use Case 4 scenarier - Rediger busrute i liste af favoriter

4.5.1 Use Case mål

Målet med denne Use Case er at tilføje en bus til listen over favoriserede busruter, eller fjerne en bus fra denne liste.

4.5.2 Use Case scenarier

Før denne Use Case kan startes skal *Use Case 1: Vis busruter* være gennemført. Fra listen af alle busruter, tilkendegiver brugeren overfor systemet, at en busrute ønskes favorisert, eller fjernes fra favoriseringen. Ved favorisering, henter systemet den valgte busrute, samt stoppestederne, fra databasen. Dernæst persisteres dette på en lokal database på telefonen. Busruten bliver markeret som favorit på listen af busruter. Brugeren vil nu kunne vælge den favoriserede busrute fra startskærmen, frem for listen over alle busruter.
Ved fjernelse af favorisering vil systemet slette ruten, samt dens stoppesteder fra den lokale database. Markeringen vil fjernes fra ruten i listen af busruter, og det vil ikke længere være muligt at vælge ruten fra startskærmen.

4.5.3 Use Case Undtagelser

Da ruten og stoppestederne hentes fra en database, er der risiko for, at forbindelsen til databasen mistes. Hvis dette sker, vil brugeren blive præsenteret for en besked om, at det

ikke er muligt at etablere forbindelse til databasen.

4.6 Use Case 5 scenarier - Rediger information om bus

4.6.1 Use Case mål

Målet med denne Use Case er at redigere information om en bus i systemet. Dette indebærer at kunne tilføje eller fjerne en bus, samt blot at ændre i informationen om en bus der allerede eksisterer.

4.6.2 Use Case scenarier

Denne Use Case har tre normalforløb, idet at man både kan tilføje en bus, fjerne en bus eller ændre i en eksisterende bus. Det er kun en administrator der kan initialisere denne Use Case. Normalforløb 1 beskriver, hvordan en bus tilføjes. Dette foregår ved at administratoren tilkendegiver overfor systemet, at der ønskes at tilføjes en bus. Herunder gøres det muligt at indtaste information om bussen. Når administratoren har indtastet den ønskede information, tilkendegives det at informationen ønskes gemt. Systemet gemmer nu informationen på den distribuerede database.

Normalforløb 2 beskriver hvordan der redigeres i informationen om en bus der allerede eksisterer. Administratoren vælger en bus fra listen af alle busser i systemet, hvorefter det tilkendegives at der ønskes at ændre information om den valgte bus. Systemet gør det muligt for administratoren at ændre informationen om bussen og dette er gjort, tilkendegives der at ændringerne ønskes gemt. Systemet gemmer nu ændringerne i den distribuerede database.

Normalforløb 3 beskrives hvordan en bus fjernes. Denne foregår ved, at administratoren vælger en bus fra en liste af alle busser i systemet, hvorefter der tilkendegives at denne ønskes fjernet. Systemet fjerner nu den valgte bus fra den distribuerede database.

4.6.3 Use Case Undtagelser

Da informationen om busserne både skal hentes fra og gemmes på en den distribuerede database, er der risiko for, at forbindelsen til databasen mistes. Hvis dette sker, vil administratoren blive præsenteret for en besked om, at det ikke er muligt at etablere forbindelse til databasen.

4.7 Use Case 6 scenarier - Rediger bus på rute

4.7.1 Use Case mål

Målet med denne Use Case er at kunne redigere listen af busser, som kører på en rute.

4.7.2 Use Case scenarier

Denne Use Case har to normalforløb, idet at det både er muligt at tilføje en bus til en rute, samt fjerne en bus fra en rute. Det er kun en administratorenm der kan initialisere denne Use Case. Normalforløb 1 beskriver hvordan en bus tilføjes til en busrute. Dette foregår ved, at administratoren først vælger en busrute fra en liste af alle busrute i systemet. Herefter vælger administratoren en bus, fra en liste af alle busser i systemet, som ikke allerede er på en busrute, hvorefter der tilkendegives at den valgt bus ønskes tilføjet til valgt busrute. Administratoren kan nu tilkendegive overfor systemet at ændringerne ønskes gemt, hvorefter de gemmes på den distribuerede database.

Normalforløb 2 beskriver hvorda en bus fjernes fra en busrute. Dette forgår ved at administratoren først vælger en busrute, fra listen af alle busruter i systemet. Herefter vælger administratoren en bus, fra listen af busser, der er i øjeblikket er på ruten, hvorefter der tilkendegives at valgt bus ønskes fjernet fra valgt busrute. Administratoren kan nu tilkendegive overfor systemet at ændringerne ønskes gemt, hvorefter de gemmes på den distribuerede database.

4.7.3 Use Case undtagelser

Da informationen om rute og busser både skal hentes fra og gemmes på en den distribuerede database, er der risiko for, at forbindelsen til databasen mistes. Hvis dette sker,

vil administratoren blive præsenteret for en besked om, at det ikke er muligt at etablere forbindelse til databasen.

4.8 Use Case 7 scenarier - Rediger busruteplan

4.8.1 Use Case mål

Målet med denne Use Case er at kunne ændre en busrute. Dette indebærer at kunne oprette en ny busrute, fjerne en busrute, samt ændre i en eksisterende busrute.

4.8.2 Use Case scenarier

Denne Use Case har tre normalforløb, idet det både er muligt at tilføje en busrute, fjerne en busrute, samt ændre i en busrute. Det er kun en administrator der kan initialisere denne Use Case. Normalforløb 1 beskriver hvordan en ny busrute kan tilføjes. Dette foregår ved, at administratoren tilkendegiver overfor systemet at en ny busrute ønskes oprettet. Administratoren præsenteres for et kort hvorpå det er muligt at indtegne en busrute. Når den ønskede busrute er indtegnet på kortet, kan den gemmes på den distribuerede database ved, at administratoren tilkendegiver overfor systemet, at ruten ønskes gemt.

Normalforløb 2 beskriver hvordan en allerede eksiterende busrute kan ændres. Dette foregår ved, at administratoren vælger en busrute, fra listen af busruter der findes i systemet. Administratoren præsenteres nu for et kort, hvorpå den valgte busrute er indtegnet. Nu kan den valgte busrute ændres som det ønskes, hvorefter det kan tilkendegives, at ændringerne ønskes gemmet. Systemet vil herefter gemme ændringerne i den distribuerede database.

Normalforløb 3 beskriver hvordan en allerede eksisterende busrute kan fjernes. Dette forgår ved at administratoren vælger en busrute, fra listen af busruter der findes i systemet. Der tilkendegives nu overfor systemet, at den valgte rute ønskes slettet, hvorefter busruten slettes fra den distribuerede database.

4.8.3 Use Case undtagelser

Da informationen om ruten både skal hentes fra og gemmes på en den distribuerede database, er der risiko for, at forbindelsen til databasen mistes. Hvis dette sker, vil administratoren blive præsenteret for en besked om, at det ikke er muligt at etablere forbindelse til databasen.

4.9 Use Case 8 scenarier - Rediger stoppested

4.9.1 Use Case mål

Målet med denne Use Case er at kunne ændre information om stoppesteder. Dette indebærer at kunne oprette et nyt stoppested, fjerne et stoppested, samt ændre i et eksisterende stoppested.

4.9.2 Use Case scenarier

Denne Use Case har tre normalforløb, idet det både er muligt at tilføje, fjerne, samt ændre et stoppested. Det er kun en administrator der kan initialisere denne Use Case. Normalforløb 1 beskriver hvordan der kan tilføjes et nyt stoppested til systemet. Dette forgår ved at administratoren tilkendegiver at et nyt stoppested ønskes oprettet. Systemet præsenterer nu administratoren for et kort, hvorpå postionen for stoppested kan indtegnes. Herefter navngives det indtegnede stoppested, og det tilkendegives at dette ønskes gemt. Systemet gemmmer nu det oprettede stoppested på den distribuerde database.

Normalforløb 2 beskriver hvordan administratoren kan ændre et allerede eksiterende stoppested. Dette forgår ved at et allerede eksisterende stoppested vælges fra listen af stoppesteder, hvorefter postionen og navn kan ændres. Hvis ændringerne ønskes gemt, kan administratoren tilkendegive dette, hvorpå systemet vil gemme ændringerne i den distribuerede database.

Normalforløb 3 beskriver hvordan administratoren kan fjerne et allerede eksiterende stoppested. Dette forgår ved at et allerede eksisterende stoppested vælges fra listen af stoppesteder, hvorefter der tilkendegives at dette ønskes slettet. Systemet sletter så det valgte

stoppested fra den distribuerde database.

4.9.3 Use Case undtagelser

Da informationen om stoppestederne både skal hentes fra og gemmes på en den distribuerede database, er der risiko for, at forbindelsen til databasen mistes. Hvis dette sker, vil administratoren blive præsenteret for en besked om, at det ikke er muligt at etablere forbindelse til databasen.

5 LOGISK VIEW

5.1 Oversigt

Hele systemet består reelt set, af tre komponenter der interagerer over den distribuerede database. Hver af disse komponenter er derfor beskrevet detaljeret i afsnit *8: Implementerings View*, i deres respektive underafsnit. Da disse tre komponenter er stående systemer i sig selv, vil et klasse diagram for komponenten vises i starten af hvert afsnit.
Alle tre komponener er opbygget efter tre-lags modellen, og dette reflekteres i den måde klasse diagrammet er opbygget, hvor præsentations-, logik- og data tilgangs laget er pakker for dem selv. Hver data tilgangs klasse, kalder til databasen, ud af systemet.

5.2 Arkitektursignifikante designpakker

Som nævnt i forrige afsnit, er alle tre komponenter delt op i tre-lags modellen, derfor vil der eksistere tre designpakker i hver komponent. Hver pakke har samme formål, på kryds af komponenter, så hver pakke vil beskrives generelt, frem for komponent-specifikt.

5.2.1 Pakke 1 - Præsentations Lag

Denne pakke indeholder alt view relateret. Det vil sige at events vil blive håndteret og funktionaliteter i logik laget vil startes. Ligeledes vil hentet og ændret data præsenteres her. Data tilgangs laget tilgås aldrig direkte herfra, da alt funktionalitet sker i gennem logik laget. Alle Use Cases startes igennem denne pakke.

5.2.2 Pakke 2 - Logik laget

Alle funktionaliteter og data manipulationer sker i dette lag. Dette indebærer at sørge for, at Use Cases bliver udviklet korrekt. Denne pakke sørger for at modtage og håndtere kald fra præsentations laget og, hvis nødvendigt, sørge for at tilgå data tilgangs laget for enten at skrive eller læse. Læst data fra data tilgangs laget vil blive manipuleret og sendt til præsentations laget.

5.2.3 Pakke 3 - Data tilgangs laget

Denne pakkes eneste formål er, at tilgå databasen når det bliver nødvendigt i logik laget. Dette lag kan enten læse eller skrive til databasen, alt efter behov. Hentet data bliver ikke manipuleret, men returneret direkte til logik laget. Dette lag kan aldrig tilgås fra, eller returnere til, præsentations laget.

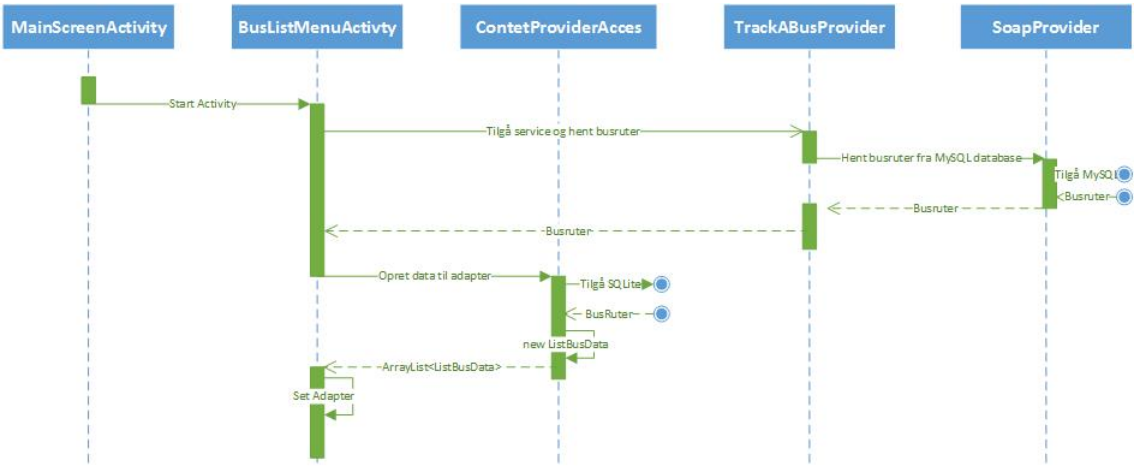
5.3 Use Case realiseringer

Dette afsnit beskriver, hvordan de fastsatte Use Cases i kravspecifikationen er realiseret. Dette indeholder initialisering, process og afsluttelse. Samtlige Use Cases startes i præsentations laget, men interagerer på forskellige måder med systemet. Ikke Use Case relevante processer (funktionalitet der ikke er strengt use case relateret) beskrives i *afsnit 8: Implementerings View*, for den relevante komponent, Use Casen udføres på.

5.3.1 Use Case 1: Vis busruter

"Vis busruter"Use Casen, er den eneste Use Case, brugeren kan starte, i TrackABus mobil applikationen, hvis der er installeret en hel ny kopi af applikationen. Den initialiseres når man trykker på "View busroutes"knappen fra hovedskærmen. Ved dette tryk startes BusListMenuActivity. I "onCreate"sættes der en progressbar, som kører indtil processen er færdig, hvorefter "onStart"kaldes, som undersøger om internettet kan tilgås. Hvis ikke, færdiggøres progressbaren og brugeren notificeres om, at der ikke kunne skabes forbindelse til nettet. Hvis der derimod er net, tilgås TrackABusProvideren som en BoundService og rutenumrene bliver hentet. Alle funktioner i servicen sker i en seperat tråd, og svarer tilbage til viewet via en MessageHandler. Når rutenumrene er hentet, sender TrackABusProvideren en besked til den medsendte MessageHandler, som er oprettet i BusListMenuActivity. Hvis ingen ruter blev fundet, notificeres brugeren om dette og progresbaren færdiggøres. Ellers oprettes der en ListBusData for hver rute, og hvis nummeret eksisterer i SQLite databasen sættes denne rute som favoriseret. En BusListAdapter oprettes, og samtlige oprettede ListBusData sættes i den. Herefter sættes denne adapter til listen i BusListMenuActivity.
På det simple sekvens diagram på figur 14, kan hovedforløbet for denne Use Case følges.

Efter denne Use Case er fuldendt, kan Use Case 2 og 4 tilgås. Hvis Use Case 4 i forvejen er gennemgået, kan Use Case 2 startes fra hovedskærmen.



Figur 14: Sekvensdiagram over Use Case 1

5.3.2 Use Case 2: Vis placering af alle busser og stoppesteder på valgt rute

Denne Use Case kan først initialiseres efter Use Case 1 eller 4 er gennemført. Den initialiseres ved et tryk på en rute, enten fra hovedskærmen eller fra listen af alle busruter. Hovedforløbet splitter sig altså op, alt efter hvordan Use Case 2 startes. I "onCreate"i BusMapActivity, hvis det medsendte intent ikke har et favoriserings flag med, sættes der ikke en progressbar. Dette gøres, fordi busruten læses så hurtigt ind, at det ikke vil have nogen effekt. Herudover opsættes viewet, og der vælges om der skal læses fra SQLite- eller MySQL databasen. Rutenummeret, medsendt i det intent activityet er startet med, sættes i informations baren, over kortet. Desuden opsættes kortet med det samme, hvis det er en favoriset rute. I "onStart"startes servicen, og når forbindelsen er oprettet, startes tegningen af ruten baseret dennes favoriserings status.

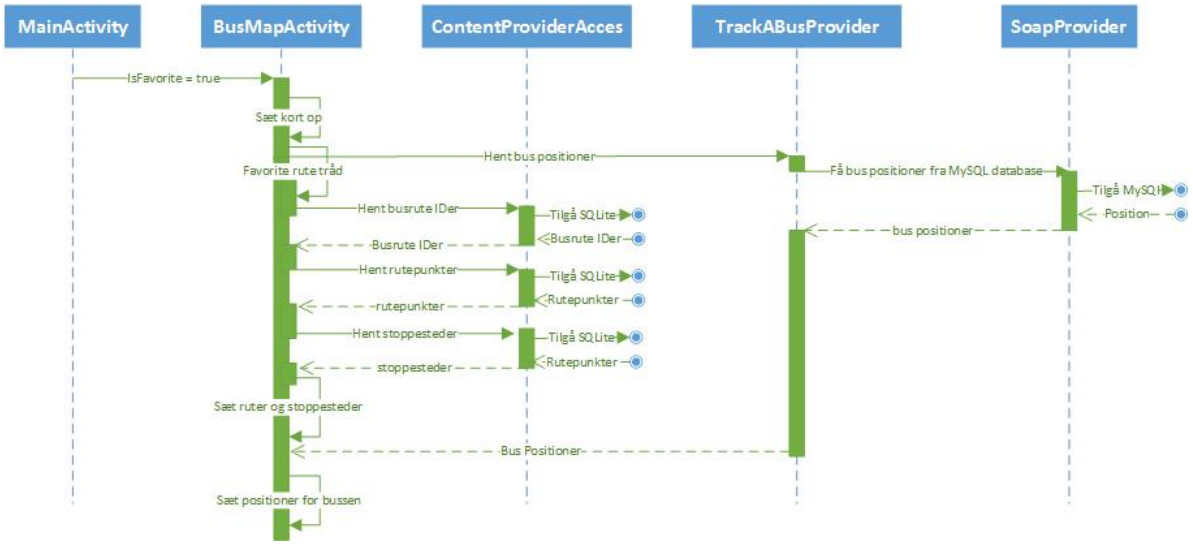
- **Favorit rute**
  - Samtlige rute IDer hentes fra SQLite databasen. Dette gøres da det er muligt at ruten er kompleks, og er blevet gemt som seperate subruter. Herefter bliver hver subrutes stoppesteder og rutepunkter hentet, igen fra SQLite databasen, og disse tegnes på kortet. Hentningen sker i en seperat tråd, men rutepunkter og stoppesteder skal sættes i main-tråden, da viewet skal opdateres, og det

kun kan gøres fra den tråd der oprettede viewet. Dette gøres med en Handler der konfigureres til at køre i main tråden. Til den kan der postes tråde, som afvikles i maintråden, selv hvis de postes fra en seperat tråd.

- **Ikke favoriset rute**
  - Servicen kaldes og den sørger for at hente busruten og stoppesteder fra MySQL databasen. Ruten kan være kompleks, og derfor hentes samtlige ruter og stoppesteder, med det valgte rutenummer. Til hvert stoppested og busrute er der oprettet en speciel designet model klasse, som implementerer parcelable. Denne funktionalitet beskrives i *afsnit 8.2.1: Komponent 1: Mobil applikation*. Servicen henter det hele i en seperat tråd.
    Servicen sætter hentede busruter og stoppesteder i en message, og sender den til den medsendte MessageHandler. I BusMapActivity håndteres denne besked i main-tråden. Progressbaren færdiggøres, kortet opsættes, og hentede ruter og stoppesteder indtegnes på kortet.

Ruten består kun af de punkter, som ikke er stoppesteder. Stoppestederne kan ligge forskudt for ruten, hvilket ville gøre, at ruten ikke vil følge vejene. Ruten bliver indtegnet som en rød polyline, og stoppestederne bliver indtegnet som et punkt med custom markers.

Samtidig med ruten hentes og tegnes, startes processen, hvor positionen for busserne hentes. Dette gøres ved et kald til servicen, hvor en MessageHandler sendes med. Hentningen sker i seperat tråd, som kører så længe servicen er bundet til activiyet. Samtlige hentede positioner sendes tilbage over den medsendte MessageHandler, og markerne for busserne fjernes, hvis de i forvejen er sat, og sættes igen med deres nye position. Når viewet lukkes, unbindes servicen og positionen for busserne holdes ikke længere opdateret. På figur 15, kan et simpelt sekvensdiagram ses, for processen omhandlende tegning af favorit rute, stoppesteder og bus positioner. Samme sekvensdiagram kan bruges for tegning af en ikke favoriseret rute med stoppesteder, hvor activityet starter fra BusListMenuActivity, og TrackABusProvideren tilgås i stedet for ContentProviderAcces, for at hente ruten.



Figur 15: Sekvensdiagram for tegning af en favoriseret rute, og hentning af bus positioner

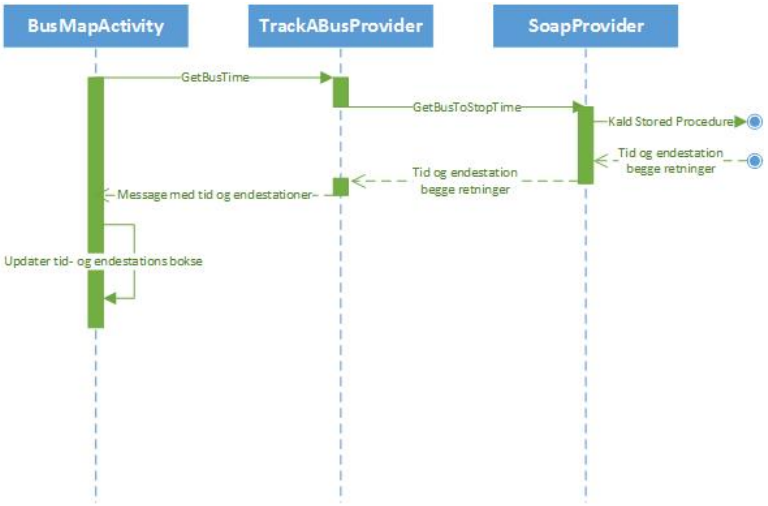
5.3.3 Use Case 3: Vis tid for nærmeste bus for valgt stoppested

Use casen kan initialiseres efter Use Case 2 er fuldført. Når et stoppested trykkes, sættes stoppestedsnavnet i den øverste informations bar på viewet. Desuden sættes teksten i begge endestationsbokse til "Loading", og tiden sættes til "nn:nn:nn". Herefter undersøges der først om der er net. Hvis der er, undersøges der om et stoppested allerede er valgt, og dermed om Use Case 3 allerede er igang, og hvis der er, interruptes tråden. Når tråden er helt lukket, kaldes "GetBusTime"i TrackABusProvideren med rutenavnet, det valgte stoppested samt en ny MessageHandler som parametrer

"GetBusTime"står for at hente tiden og endestationen for den nærmeste bus, i begge retninger på ruten. Udregningen af dette ligger som en stored procedure på MySQL databasen. Dette kan læses om i afsnit *9.1.3: Stored Procedures*. Når udregningerne er færdige, returneres resultaterne til TrackABusProvideren, hvor de bliver pakket ned i en Message, og sendt til MessageHandleren. MessageHandleren ligger i BusMapActivity, og sørger for at opdatere viewet. Hvis der ikke er nogen bus, som kører mod stoppestedet i en eller begge retninger, retuneres der "anyType"i den relevante retnings endesstations værdi. Hvis en eller begge endestations resultater er denne værdi, opdateres endestations teksten til "No bus going in this direction"og tiden til "nn:nn:nn", for den relevante retning. Hvis endestations teksten i forvejen er "No bus going in this direction", opdateres der intet for den givne retning. Grunden til endestations værdien opdateres er, at en rute kan være

kompleks med mere end to endestation, og en given bus, kan køre på samme ruten som en anden bus, men ikke have samme endestation.

Hvis tilgangen til internettet forsvinder efter et stoppested er valgt, vil opdateringstråden stadig køre, men der vil blive vist en besked om, at tiden ikke længere holdes opdateret. Herved, når forbindelsen genetableres, fortsættes tidsopdateringen på ny. Når BusMapActivity går i dvale sættes opdateringstråden på pause, og ved genåbning startes tråden igen. På figur 16 kan et simpelt sekvensdiagram for forløbet ses. Diagrammet viser en hentning af tiden og endestationer i begge retninger, hvor hentningen antages at være succesfuld.



Figur 16: Sekvensdiagram for hentning af endestationer og tid for nærmeste bus.

5.3.4 Use Case 4: Rediger busrute i listen af favoriter

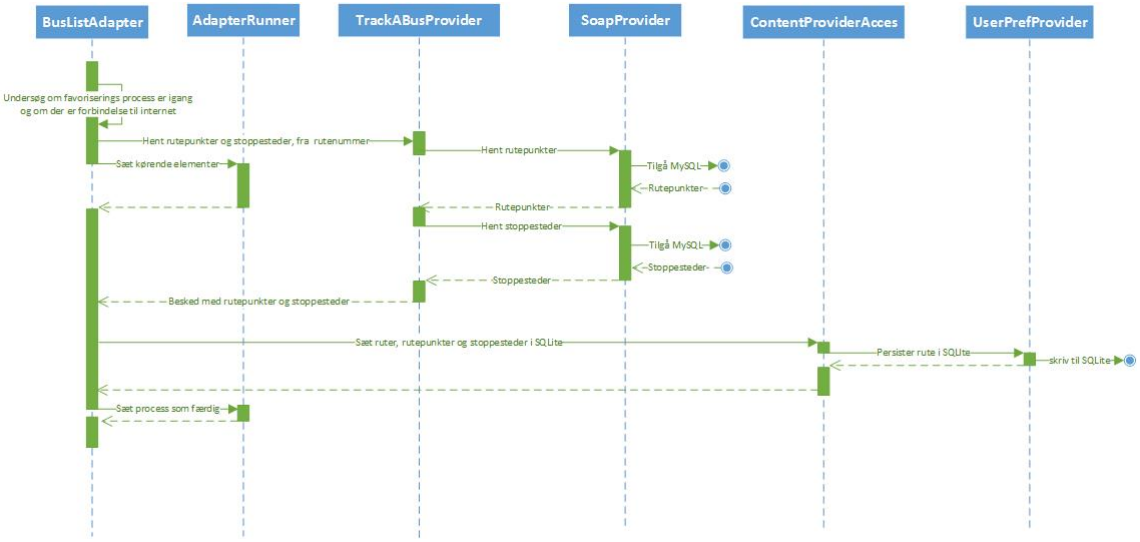
Denne Use Case kan startes efter Use Case 1 er gennemført. Der eksisterer to primære mål; At persistere en busrute lokalt, og fjerne lokal persistering af busrute. Begge mål startes på samme måde, ved at trykke på stjernen ud for en given busrute. Hvis stjernen er gul, betyder det at ruten er favoriseret, og hvis den er grå betyder det den endnu ikke er favoriseret. Processen er ens, op til det punkt, hvor der undersøges om ruten skal favoriseres eller ikke være favoriseret længere. Da listen af all busruter er baseret på en custom ListAdapter, sker view opdateringerne i BusListAdapter, frem for den egentlige BusListMenuActivity. Dette betyder også, at event handleren for trykket på favoriserings knappen, også sættes i BusListAdapter. Knappen har egentlige to states; En toggled og

en untoggled. Hertil er der dog lavet en ekstra state, hvor favoriseringen er igang. Dette er symboliseret med, at knappen fjernes, og en progressbar vises. Kun én favoriserings process kan køre af gangen, så til dette formål er AdapterRunner klassen lavet. Denne klasse indeholder de view-elementer, som ligger på det list element, favoriseringen er startet ud fra. De er statiske, da der kun kan være én kørende favorisering process. Når BusListMenuActivityen startes, sættes dens adapter, og det rutenummer på listen, der svarer overens med det rutenummeret i AdapterRunner klassen, får sat de view elementer der er gemt her. Herved bibeholdes elementernes state, også selvom activityetn lukkes og startes igen.
Når knappen trykkes undersøges der først, om en process er igang. Dette er en simpel bool, som er sat i AdapterRunneren. Hvis den er sat, notificeres brugeren at processen er igang, og click-eventet bliver ignoreret. Herfra splittes Use Casen op i de to normalforløb.

- **Favoriser**
  - Der undersøges først om der er internet forbindelse og hvis der ikke er, annuleres processen og brugeren notificeres om manglen på forbindelse. Ellers tilgås TrackABusProvideren, og hele busruten samt dens stoppesteder hentes fra MySQL databasen. Dette sker asynkront, så mens ruten hentes fjernes knappen, progressbaren sættes, og hele elementet gemmes i AdapterRunner klassen. Når busruten er hentet, sendes den som en besked, til MessageHandleren i adapteren. Herefter startes der en tråd, til at lave selve persisteringen. Heri tilgås ContentProviderAcces med rutepunkterne og stoppestederne, hvori det hele persisteres i SQLite databasen. Når persisteringen er fuldent, sættes elementerne i AdapterRunner til at være færdige, således at progressbaren fjernes og favoriserings-knappen vises igen, som nu er gul.
- **Fjern favorisering**
  - Ligesom i favoriseringen sættes elementeterne ved den trykkede knap i AdapterRunner klassen, samt progressbaren vises og favoriserings knappen fjernes. ContentProviderAcces tilgås i en ny tråd, hvori ruten med det givne busnummer fjernes fra persisteringen. Når fjerningen af persisteringen er fuldent, sættes elementerne i AdapterRunner til at være færdige, således at progressbaren

fjernes og favoriserings knappen vises igen. Desuden vil stoppestederne ikke slettes, da disse kan være brugt i andre favoriserede ruter.

På figur 17 kan favoriseringen af en busrute ses som et simpelt sekvens diagram. Tilgangen til ContentProvideren ses på figuren som et kald, men er faktisk flere kald, et for hver tabel. Fjerning af favorisering vises ikke, da dette kun er et kald til ContentProvideren, hvor alting fjernes, på nær stoppesteder.



Figur 17: Sekvensdiagram for lokal persistering af busrute.

5.3.5 Use Case 5: Rediger information om bus

Denne Use Case består af tre normalforløb, der alle bliver initialiseret fra administrations hjemmesiden under busredigering.[4]. Viewet vil, når siden er blevet indlæst, kalde funktionen "ExecuteOnLoad"der laver tre asynkrone kald til BusControlleren, der henter navne på alle busruterne, alle busserne, og alle busser der ikke er knyttet til nogle rute. Når disse er færdige vil viewet opdatere de relevante lister på skærmen.

Normalforløb 1, hvor en ny bus skal tilføjes til systemet bliver initialiseret, ved at administratoren indskriver ID'et for den nye bus i feltet: "Bus ID", hvorefter der trykkes på knappen "Add". Dette vil tilføje bussen til listen, men vil endnu ikke gemme det på databasen. Dette vil først ske når administratoren trykker på "Save"knappen. Et tryk på save knappen, vil kalde BusController funktionen "SaveBusChanges", der vil finde de

[4]Administrations hjemmeside, bus redigering: www.TrackABus.dk/bus

busser, der er blevet tilføjet eller fjernet i listen. "SaveBusChanges"vil nu kalde funktionerne "removeBusses"for at fjerne de busser fra databasen, der blev fjernet fra listen, og "addBusses"for at tilføje til databasen, de busser administratoren tilføjede til listen. Disse to funktioner kalder ned i DBConnection klassen, der ligger i data tilgangs laget, hvilket er en statisk klasse. Denne bliver udelukkende brugt som tilgang klasse til MySQL databasen. Til sidst vil der blive retuneret til viewet, der nu vil informere administratoren om at der er blevet gemt.

Normalforløb 2, hvor der skal slettes en bus fra MySQL databasen, bliver initialiseret ved at administratoren vælger en bus, fra listen af alle busser gemt i databasen. Herefter bliver der trykket på knappen "Remove"og bussen vil nu blive fjernet fra listen. Først ved tryk på "Save"knappen vil funktionen "SaveBusChanges"blive kaldt som i normalforløb 1.

Normalforløb 3, hvor det skal være muligt at ændre informationen om en bus, bliver initialiseret ved, at administratoren vælger en af busserne fra listen. Herefter indskrives nyt ID i feltet "Bus ID", hvorefter der bliver trykket på knappen "Rename". Bussen vil nu skifte navn i listen, men igen først ved tryk på knappen "Save", vil funktionen "SaveBusChanges"blive kaldt, som i normalforløb 1.

5.3.6 Use Case 6: Rediger bus på rute

Rediger bus på rute Use Casen består af to normalforløb, hvor det første omhandler tilføjelse af bus til en busrute, og normalforløb 2 omhandler fjernelse af bus fra en busrute. Begge normalforløb bliver startet fra administrations hjemmesiden under busredigering.[5]. Viewet vil, når siden er blevet indlæst, kalde funktionen "ExecuteOnLoad"der laver tre asynkrone kald til BusControlleren, der henter navne på alle busruterne, alle busserne, og alle busser der ikke er knyttet til nogle rute. Når disse er færdige vil viewet opdatere de relevante lister på skærmen.

Administratoren kan initialisere normalforløb 1, ved at vælge en busrute, fra listen af alle busruter. Ved tryk på en af disse, vil et "onchange"event blive kaldt. Dette starter et asynkront kald til BusController funktionen "GetBussesOnRoute", som tilgår data tilgangs laget for at hente alle de busser, der er knyttet til den valgte rute, i MySQL databasen. Busserne vil blive retuneret til BusControlleren, som igen returner tilbage til

[5]Administrations hjemmeside, bus redigering: www.TrackABus.dk/bus

viewet, sim opdaterer listen "Busses on route"med de hentede busser.
Adminsitratoren kan nu tilføje busser fra listen "Avaliable busses"til listen 'Busses on route', og trykke på knappen 'Save'. Dette vil starte et asynkront kald til BusController funktionen "SaveChanges", der tilgår funktionen i data tilgangs laget "SaveChangesToBus"bliver kaldt. Denne funktion knytter bussen til den valgte rute.

Normalforløb 2, forgår på samme måde som i normalforløb 1, med den undtagelse, at der bliver taget busser fra listen "Busses on route"og tilføjet til listen "Avaliable busses". Efter der bliver trykket på knappen "Save' vil der ske det samme som i normalforløb 1, nu bare hvor busserne får fjernet deres knyttede ruter.

5.3.7 Use Case 7: Rediger busruteplan

Denne Use Case består af tre normalforløb, der alle bliver startet fra siden administrations hjemmesiden, under ruteredigerings siden. [6].
De tre normalforløb består i at kunne tilføje en ny busrute til systemet, fjerne en busrute fra systemet, og ændre en rute der allerede er oprettet.
I normalforløb 1 vil det være muligt at oprette en hel ny busrute, og gemme den i MySQL databasen. Normalforløbet bliver initialiseret ved, at administratoren trykker et sted på kortet, der symbolisere start punktet for en busrute, dette vil starte et click event på kortet, som placerer en markør, hvor der blev trykket. Der vil samtidigt blive sat et flag, der registrerer at start punktet for ruten er blevet sat. Administratoren kan nu trykke et nyt sted på kortet, der symboliserer slut punktet for ruten. Dette laver igen et click event som placerer endnu en marker, og kalder funktionen "calcRoute", i det, det tidligere nævnte flag er sat. Ved hjælp af Googles Direction API[7], udregnes en rute mellem start og slut punkterne, og denne indtegnes denne på kortet. Efter ruten er blevet udregnet og indtegnet, har administratoren mulighed for at trække i ruten, for at placere den, så den følger de ønskede veje. Administratoren har også mulighed for, efter rutene er blevet indtegnet, at trykke på enten start eller slut markøren. Dette vil starte et click event på den givne markør, som muligører oprettelsen af en kompleks rute. Dette gøres ved at trykke et nyt sted på kortet, hvorefter der vil blive oprettet en rute imellem den trykkede markør, og

[6] Administrations hjemmeside, rute redigering: www.TrackABus.dk/dir
[7] For mere information, se https://developers.google.com/maps/documentation/directions/

det nye punkt. Til sidst kan administratoren også tilføje busstoppesteder til ruten. Dette gøres ved, at der bliver flyttet stoppesteder fra listen af alle stoppesteder til listen af stoppesteder på ruten. Ved tryk på knappen 'Set bus stops', vil der gps-koordinaterne for stoppestederne i listen af stoppesteder på ruten, blive hentet fra databasen, hvorefter de vil blive indtegnet som markører på kortet. Når administratoren er færdig med at oprette ruten, og denne ønskes gemt på databasen, trykkes der på knappen "Save route". Samtlige gps-koordinater, der er blevet brugt til at tegne rute, hentes. Disse bliver fundet ved brug af JavaScript funktionen "getRoutePath". Herefter vil der blive lavet et asynkront kald til DirController funktionen "Save". Gps-koordinaterne fundet med "getRoutePath", de punkter på ruten som symbolisere start og slut punkterne, de punkter der er blevet lavet ved, at administratoren har trukket i ruten, alle stoppesteder på ruten, de gps-koordinater der udgør en eventuel subrute, samt dennes waypoints(givet der blev lavet en kompleks busrute), samt rutens navn, sendes som parametre til "Save"funktionen. Denne funktion vil nu starte med at kalde ned til data tilgangs laget, for at indsætte navnet for den nye ruten i databasen. Herefter vil waypoints blive indsat i databasen og alle gps-koordinaterne for hele ruten vil ligeså blive indsat. Hvis der er blevet lavet en kompleks rute, vil de waypoints og rutepunkter der udgøre denne del, også blive indsat i databasen. Save funktionen vil nu kalde funktionen "CalculateBusStopsForRoute"i RouteMath klassen. Denne funktion udregner mellem hvilke rutepunkter, stoppestederne skal ligge i mellem, og retunere en liste med den rigtige rækkefølge tilbage til "Save"funktionen. Denne vil nu, til sidst, kalde til data access laget, for at indskrive rækkefølgen på gps-koordinaterne og stoppestederne.

I normalforløb 2, skal det være muligt at slette en busrute fra databasen. Denne blive initialiseret ved, at administratoren vælger en busrute fra listen af alle busruter, hvorefter der bliver trykket på knappen "Delete". Der bliver nu lavet et asynkront kald til DirController funktionen "DeleteSelectedBusRoute", som kalder ned i data tilgangs laget, som sørger for, at hele ruten og alt releateret til denne, bliver slettet fra databasen.

Normalforløb 3 ændringer i en allerede eksisterende busrute. Denne bliver initialiseret ved, at administratoren vælger en busrute, fra listen af alle busruter. Dette vil starte et

"onchange"event, der laver et asynkront kald til DirController funktionen "GetSelectedBusRoute", som der kalder ned i data tilgangs laget hvor den valgte rutes waypoints bliver hentet fra databasen. De hentede waypoints bliver sendt tilbage til viewet, der nu ved brug af Googles Directions API[8], udregner og indtegner på kortet den valgte rute ud fra rutens waypoints. Administratoren har nu mulighed for at trække i ruten, tilføje eller fjerne stoppesteder som i normalforløb 1. Når ruten er blevet ændret, og dette ønskes gemt, trykkes der på knappen "Save Changes". Den gamle rute vil blive slettet fra databasen, som beskrevet i normalforløb 2, hvorefter den nye rute bliver gemt på databasen, som beskrevet i normalforløb 1.

5.3.8 Use Case 8: Rediger stoppested

Use Case 8 består af tre normalforløb der omhandler at kunne tilføje, fjerne eller ændre stoppesteder i databasen. Alle tre normalforløb bliver startet fra administrations hjemmesiden under stoppesteds redigering. [9]
Normalforløb 1 initialiseres ved, at administratoren trykker et sted på det viste kort. Dette vil starte et click event på kortet Der placerer en markør, hvor der blev trykket. Det er muligt at fjerne denne markør ved at trykker på den igen, da dette vil trigger et click event på markøren, som sletter den, og derved gør det muligt at placere en ny markør. Efter at have placeret markøren, vil det være muligt at trække den rundt på kortet, så dens position kan angives præcist. Når den endelige ønskede plasering for markøren er fundet, indtastes navnet på stoppestedet i feltet "Stop name". Ved tryk på knappen "Save", vil gps-koordinaterne for markøren blive fundet, og sendt sammen med stoppestedsnavnet, med et asynkron kald til StopController funktionen "Save". Denne kalder til data tilgangs laget, hvor det nye busstoppested gemmes i databasen.

Normalforløb 2 bliver initialiseret ved at administratoren vælger et stoppested fra listen af alle stoppesteder, og trykker på knappen "Delete". Dette vil lave et asynkront kald til StopController funktionen "Delete", der kalder ned til data tilgangs laget, hvor sørges for, at det valgte stoppested fjernes fra databasen.

[8] For mere information, se https://developers.google.com/maps/documentation/directions/
[9] Administrations hjemmeside, stoppesteds redigering: www.trackabus.dk/Stop

Normalforløb 3 omhandler at kunne ændre position og navn for et givet stoppested. Denne bliver initialiseret ved, at administratoren vælger et stoppested fra listen af alle stoppesteder. Dette vil starte et onchange event, der laver et asynkront kald til StopController funktionen "GetPosistion". Denne kalder videre ned i data tilgangs laget, for at hente gps-koordinaterne for det valgte stoppested fra databasen. Koordinaterne vil blive sendt tilbage til viewet, der nu vil oprette en ny marker på kortet, med position tilsvarende det modtagede koordinat-sæt. Administratoren har nu mulighed for at trække i markeren for at ændre stoppestedets position, samt ændre dennes navn i feltet "Stop Name". Når de ønskede ændringer er foretaget, trykkes der på knappen "Save changes". Dette starter et asynkront kald til StopController funktionen "SaveChangeToStop, som kalder videre ned i data tilgangs laget. Her sørges der for at stoppestedets navn og position bliver opdateret i databasen.

6 PROCES/TASK VIEW

I dette afsnit bliver der beskrevet systemets opdeling af tråde, samt hvorledes de kommunikere og bliver synkroniseret.

6.1 Oversigt over processer/task

Processen på mobil applikationen består af følgende tråde:
Main/UI Thread - Denne tråd er systemets hovedtråd og står for at modtage input fra brugeren, vise ting på skærmen samt initialisere andre komponenter og tråde.
TrackABusProvider - Klasse der står for at lave trådene der bruges til at tilgå Soap-Provideren.
SetFavoriteBusRoute - Står for at gemme busrute som favorit i SQLite databasen.
RemoveFavorite - Står for at fjerne en favoriseret busrute fra SQLite databasen.

Administrations hjemmesiden består af to tråde:
Main/Client thread- Dette er hoved tråden, der køre på clienten, for hjemmesiden. Den står blandt andet for at modtage input fra administratioren og opdatere skærmen.
Background/Server thread- Denne tråd, der bliver kørt på serveren, står for at skrive til- og hente fra MySQL databasen.

Simulatoren består af main tråden, samt en tråd for hver bus der bliver simuleret.
Main/UI thread - Denne tråd står for at modtage input fra brugeren, starte de forskellige tasks, samt opdatere skærmen.
Bus threads - Disse tråde bruges til at simulere en bus der kører på en busrute.

6.2 Proces/task kommunikation og synkronisering

Kommunikation

TrackABusprovideren er en klasse, som bliver kaldt fra Main tråden når MySQL databasen skal tilgås. Dette sker ved at TrackABusProvideren laver en ny tråd, der kalder ned i data tilgangs laget til SoapProvideren der tilgår databasen, og returnerer til TrackABusProvider med det relevante data. For at tråden i TrackABusProvideren kan sende data tilbage til main tråden, bliver der brugt en message handler, der lytter på, om der kommer en besked fra TrackABusProvideren.

Både SetFavoriteBusRoute og RemoveFavorite bliver begge startet af main tråden, men da de blot skal indsætte og slette fra SQLite databasen, har de ikke behov for at kommunikerere med andre tråde.

Administrations hjemmesidens server tråd og client tråd snakker sammen ved brug af AJAX kald, *Se afsnit 8.2.3 Komponent 3: Administrations hjemmeside* for en beskrivelse af, hvordan dette virker og er implementeret. Dette kald starter en asynkron task der bliver kørt på serveren. Når serveren er færdig med at udføre tasken vil der blive sendt et response tilbage til clienten, der lavede AJAX kaldet.

Simulatorens bus tråde kommuikerer med UI tråden ved hjælp af to custom events. Det ene sørger for at ændre viewet når det er nødvendigt, og det andet sørger for at sætte en log-besked. Dette sker igennem BeginInvoke funktionen, som sørger for, at selv hvis en view opdatering sker direkte fra en tråd, håndteres det i UI-tråden. **Synkronisering**
Trådene SetFavoriteBusRoute og RemoveFavorite er de eneste tråde på mobil applikationen, der har behov for at blive synkroniseret. Grunden til at TrackABusProvideren ikke har brug for Synkronisering er, at den ikke kan skrive til MySQL databasen, men kun hente. SetFavoriteBusRoute skal skrive til SQLite databasen, hvilket skal synkroniseres så der ikke bliver skrevet flere gange til SQLite databasen på samme tid. Dette er blevet opnået ved at knappen, der bliver brugt til at favorisere en bus, bliver deaktiveret indtil

tråden er færdig med at skrive til databasen, dette gør, at det ikke er muligt at starte to tråde der skriver til SQLite databasen på samme tid.

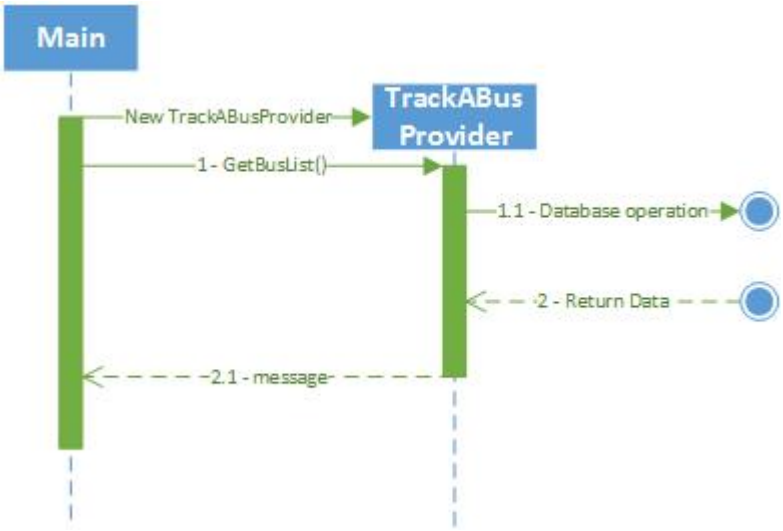
Administrations hjemmesidens client og server skal synkroniseres, så der ikke bliver skrevet til databasen flere gange på samme tid. Dette sker på samme måde som i mobil applikationen ved, at det ikke er muligt at skrive til databasen, hvis en anden task allerede er igang med at skrive.

I simulatoren bliver tråd synkroniseringen håndteret, ved hjælp af binære semaforer. Dette er kun relevant, når MySQL databasen skal tilgås. Skrivnings- og læsnings funktionerne er statiske, og det samme er deres relevante semaforer, således at når tråd starter skrivningen, tager denne semaforen, og frigiver den når den er færdig. I tilfælde af, at simuleringen stoppes, og trådene lukkes, før semaforen er frigivet, frigives semaforerne ved hver afslutning af simulering. Dette gøres for at undgå deadlocks.

6.3 Procesgrupper

Hver del af systemet består af én procesgruppe. Dette vil sige at mobil applikationen, administrations hjemmesiden og simulatoren er en procesgruppe for sig. I de følgende afsnit vil hver procesgruppe blive beskrevet. Sekvens diagrammerne der bliver brugt er forsimplede, for at gøre det mere overskueligt, dvs. der ikke vil blive brugt navne på alle klasserne trådne går igennem, men fokusere mere på hvordan trådne bliver starten og kommunikere med hinanden.

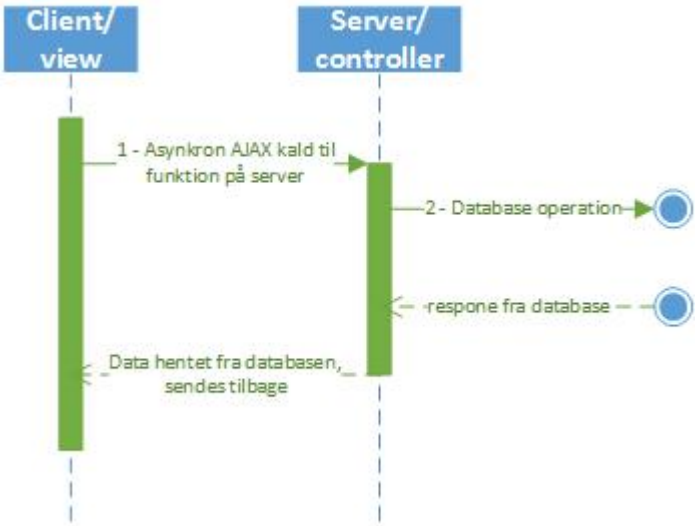
6.3.1 Proceskommunikation i mobil applikation



Figur 18: Sekvensdiagram over - Main og TrackABusProvider på mobil applikationen

På figur 18 vises et eksempel på, hvordan main tråden i mobil applikationen kommuniker med TrackABusProvideren. TrackABusProvideren agerer som en BoundService, hvilket betyder, at det er muligt at kalde til den, i det øjeblik den er bundet. Den skal alstå ikke instantieres. En ny tråd skabes, der kalder ned i data tilgangs laget for at hente data fra MySQL databasen, alt efter hvilken TrackABusProvider funktion, der er blevet kaldet. Det kan ses at main tråden ikke bliver blokeret imens det hentes fra databasen. Når TrackABusProvider er færdig med at hente data fra databasen, bliver der sendt en message til main tråden, som indeholder det hentede data. Ved hjælp af en message handler vil beskeden blive modtaget i main tråden, og brugeren bliver præsenteret for det hentede data.
SetFavoriteBusRoute og RemoveFavorite bliver blot startet af main tråden, og sender en besked tilbage når de er færdige med at gemme eller slette fra SQLite databasen. Det vil ikke være muligt at starte en ny SetFavoriteBusRoute eller RemoveFavorite imens en af dem kører.

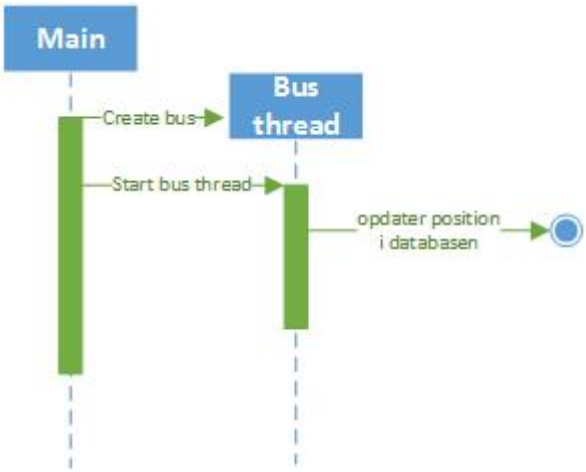
6.3.2 Proceskommunikation i administrations hjemmesiden



Figur 19: Sekvensdiagram over - Client/server kommunikation

På figur 19 ses det, hvordan der bliver lavet et asynkron kald fra client til serveren, som vil tilgå databasen i skrivnings- eller læsnings øjemed. Efter serveren er færdig, vil den retunerer et resultat tilbage til main tråden.

6.3.3 Proceskommunikation i simulatoren

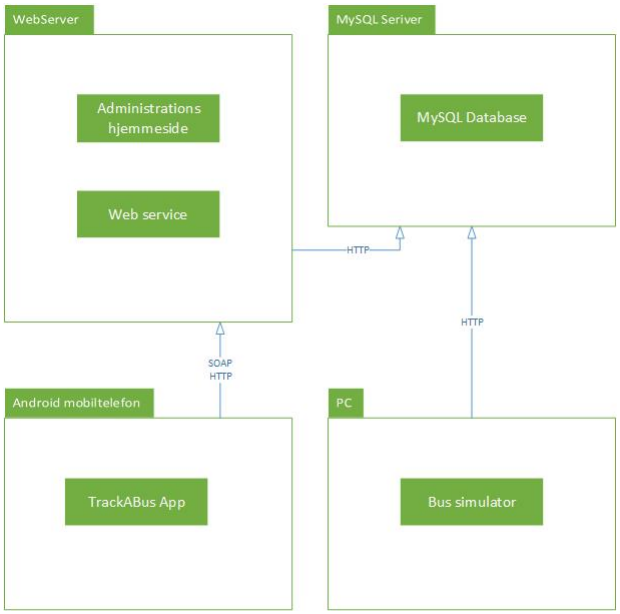


Figur 20: Sekvensdiagram over - simulator tråd kommunikation

På figur 20 ses det, hvordan main tråden opretter en ny bus, hvorefter den starter bus tråden. Main tråden opretter en tråd for hver bus, da disse skal køre asynkront.

7 DEPLOYMENT VIEW

Systemet indeholder 4 processer; Serveren der hoster hjemmesiden og servicen, serveren der hoster MySQL databasen, android mobiltelefonen og den PC hvor simulationsprogrammet afvikles på.



Figur 21: Deployment Diagram

7.1 Oversigt over systemkonfigureringer

Det er muligt at udskifte både webserveren samt MySQL serveren, så længe de overholder minimumskravene. Android mobiltelefonen kan være en hvilken som helst android mobiltelefon, givet den kører med Android 4.3 Jelly Bean- eller Android 4.4 KitKat styresystem. Det er desuden også muligt at udskifte PCen, hvorpå bus simulatoren kører, med en anden PC der overholder minimumskravene. 1

7.2 Node-beskrivelser

7.2.1 Node 1. beskrivelse - Android mobil applikation

På denne enhed kørers TrackABus mobilapplikationen. Dette skal være en android mobiltelefon med enten android 4.3 Jelly Bean, eller android 4.4 KitKat styresystem. For at få mest muligt ud af applikationen skal der være mulighed for, at have adgang til internettet.

Dette kan være wifi såvel som mobilt datanetværk. Applikatkionen kræver desuden ca. 4.1 MB ledig plads.

7.2.2 Node 2 beskrivelse - Webserver

Denne enhed hoster administrations hjemmesiden[10] samt web servicen, som ligger på serveren nt21.unoeuro.com. Denne er hostet hos UnoEuro[11]. Webserveren er en ASP/ASP.NET server. Hjemmesiden er implementeret ved brug af ASP.NET og Web servicen er en ASP.NET webservice.

7.2.3 Node 3 beskrivelse - MySQL Server

Denne enhed hoster MySQL databasen, som ligger på serveren mysql23.unoeuro.com, der er hostede af UnoEuro[12]. For at kunne logge ind på databasen kræver det følgende oplysninger:

- **Server type:** MySQL
- **Server:** mysql23.unoeuro.com
- **Port:** 3306
- **Login:** trackabus_dk
- **Password:** 1083209421

Databasen er blevet implementeret ved brug af MySQL workbench.

7.2.4 Node 4 beskrivelse - PC

På denne enhed køre GPSsimu.exe som indeholder bus simulatoren. Denne bruges til at simulere busser, som kører på en busrute. Der skal som minimum være tale om et 64-bit styresystem, baseret på Windows 8.

[10] www.trackabus.dk
[11] www.unoeuro.com
[12] www.unoeuro.dk

8 IMPLEMENTERINGS VIEW

8.1 Oversigt

Dette afsnit beskriver den endelige implementeringsopdeling af softwaren i lagdelte delsystemer. Dette view specificerer opdelingen i det logiske. Figurerne i afsnittet kan findes på bilags CDen, under Diagrammer. Fuld implementering af kode findes ligeså på bilags CDen under kode, i komponentens respektive undermappe.

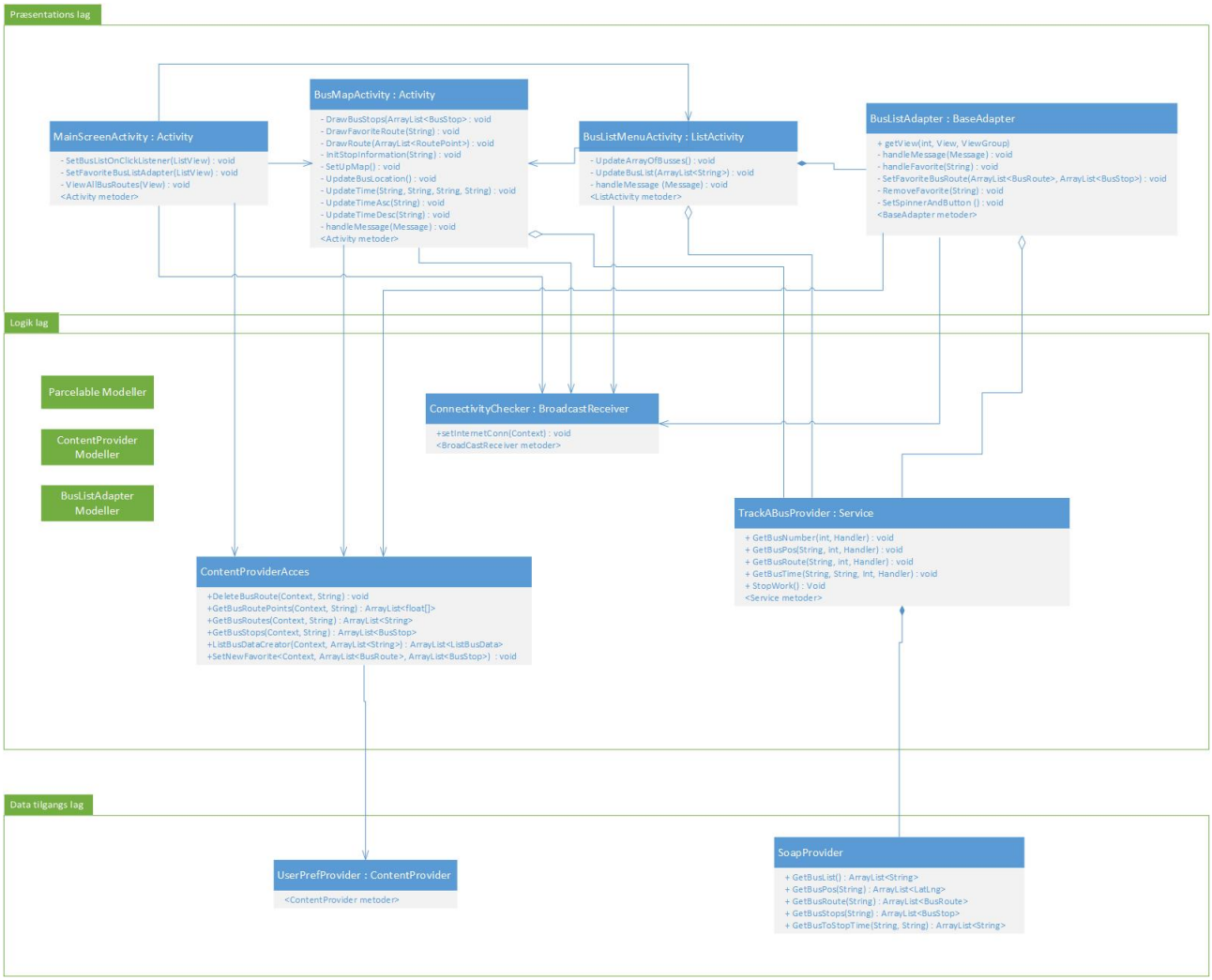
8.2 Komponentbeskrivelser

8.2.1 Komponent 1: Mobil applikation

Denne komponent har til formål at formidle alt bus information til brugeren. Den gør det muligt for brugeren at se busruter med stoppesteder indtegnet på et kort samt positionen for de busser der køre på ruten. Herudover vil det være muligt at se hvor lang tid der er til, at den næsten bus ankommer ved et valgt busstoppested.

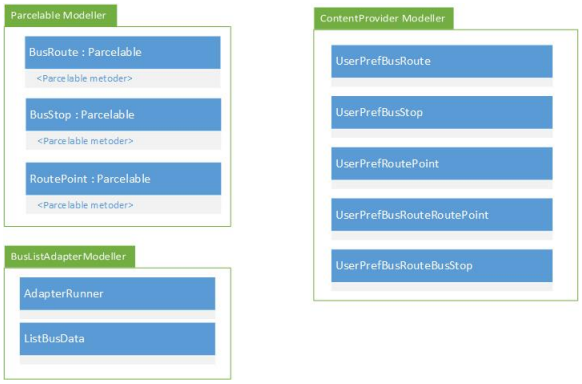
Specifikationer

På figur 22 kan de forskellige klasser ses, samt hvilke lag de ligger i og hvordan de interagerer. Model klasser vises ikke som interageringer.



Figur 22: Klassediagram for mobil applikationen

På figur 23 ses de forskellige model klasser, der hovedsageligt består af custom datatyper.



Figur 23: Klassediagram for model klasser i mobil applikationen

Her følger en kort beskrivelse af hver klasse, samt den funktionalitet klassen tilføjer til systemet.

- MainScreenActivity
  - Denne klasse implementerer Activity interfacet, hvilket betyder, at den bruges som et view. Denne agerer som startskærm for applikationen, og lader brugeren vælge en favorit bus. Herfra kan brugeren desuden også vælge at åbne BusListMenuActivity.
- BusListMenuActivity
  - Denne klasse implementerer ListActivity interfacet, hvilket betyder, at den bruges som et view, men en list adapter kan sættes hertil. Den har til formål at præsentere brugeren for listen af ruter der eksisterer i MySQL databasen, samt favorisere en given rute. Selve viewet er bygget op med en adapter, hvori favoriserings funktionaliteten ligger, samt click eventet for favoriserings knappen. Selve klikket på et element fra listen, håndteres i BusListMenuActivity klassen. Herigennem har brugeren mulighed for at vælge en busrute, favoriseret såvel som ikke.
- BusMapActivity
  - Denne klasse implementerer Activity interfacet, hvilket betyder, at den bruges som et view. Den har til formål at præsentere brugeren for et kort med

indtegnet valgt busrute og stoppesteder. Her vil kørende busser på ruten også præsenteres. Den har desuden til formål at starte tidsopdaterings funktionalitet, ved et tryk på et stoppested.

- BusListAdapter
  - Denne klasse implementerer Adapter interfacet, hvilket betyder, at den kan bruges til at repræsentere de layout-elementer der skal vises på viewet. I denne sammenhæng er den koblet til BusListMenuActivityet. Den har til formål at håndtere alle favoriserings funktionerne, og click eventet for favoriserings knappen.
- ConnectivityChecker
  - Denne klasse implementerer BroadcastReceiver, hvilket betyder, at den modtager registrede system events. Disse events bliver sat i manifestet. I denne sammenhæng er den koblet til CONNECTIVITY_CHANGE og WIFI_STATE_CHANGED. Disse events omhandler ændringer i netværksforbindelse. Når der sker en ændring, undersøger den hvorvidt nettet kan tilgås, hvorefter den sætter en statisk bool til true hvis det kan tilgås, og false hvis ikke.
- ContentProviderAcces
  - Denne klasse er et abstraktions lag mellem præsentations laget og ContentProvideren, som ligger i data tilgangs laget. Dette betyder at det er den eneste klasse, som tilgår ContentProvideren. Funktionerne i denne klasse er statiske, da kun én indskrivnings- eller sletnings process kan udføres af gangen.
- TrackABusProvider
  - Denne klasse implementer Service interfacet, hvilket betyder at klassen kan tilgås som en Bound Service. Klassen agerer som et abstraktions lag mellem præsentations laget og SoapProvideren, som ligger i data tilgangs laget. Dette betyder, at den er den eneste klasse, som tilgår SoapProvideren. Funktionerne i denne klasse returnerer aldrig direte til det view som kaldte den, men i

stedet over en MessageHandler. Dette gør at service funktionerne kan afvikles asynkront.

- UserPrefProvider
  - Denne klasse implementer ContentProvider interfacet, hvilket betyder at klassen kan tilgås igennem "getContentResolver"funktionen, hvis den er koblet til applikationen i manifestet. Denne klasse sørger for at tilgå den lokale SQLite database, for at persistere eller fjerne en favoriseret rute. Denne er således den ene klasse i data tilgangs laget. Der kommunikeres udelukkende med denne klasse igennem ContentProviderAcces klassen.
- SoapProvider
  - Denne klasse har til formål at tilgå de forskellige funktioner i mobil servicen på serveren. Disse kald tilgår MySQL databasen, og er derfor den anden klasse i data tilgangs laget. Der kommunikeres udelukkende med denne klasse igennem TrackABusProvideren.
- Parcelable modeller
  - Disse klasser bruges som datatyper TrackABusProvideren. Disse har alle til fælles at de implementerer Parcelable. Dette gør det muligt for disse modeller at pakkes ned i en message, og sendes til en nessage handler.
- ContentProvider modeller
  - Disse klasser er modeller for de tabeller, SQLite databasen indeholder. De indeholder herunder navnene på de forskellige kolonner i tabellen, samt hvilken URI, der skal kaldes med, for at tilgå den givne tabel.
- BusListAdapter modeller
  - Disse klasser bruges i sammenhæng med at gemme de UI elementer, som bruges i BusListAdapteren, samt det data denne indeholder.

Design:

Applikationen er blevet udviklet til android mobiltelefoner som kører med OS version 4.3

Jelly Bean og op til 4.4 KitKat. Dertil er der blevet udviklet imod android API level 18 og 19.

Mobil applikationen er udviklet med fokus på, at der skulle ske så lidt arbejde på telefonen som muligt. Alle tunge udregninger og processeringer sker på en webservice, *Se afsnit 8.2.2 Komponent 2: Mobile service*

For at kunne vise busruter, stoppesteder, busser og tid til ankomst, skal dette hentes fra en databasen. Til dette formål er der blevet lavet to klasser, TrackABusProvider og SoapProvider. TrackABusProvideren er udviklet som en BoundService, som både BuslistMenuActivity og BusMapActivity bruger. For at binde til TrackABusProvideren, bliver "startService"kaldt, som starter servicen, hvis den ikke allerede kører. Herefter vil funktionen "bindService"blive kaldt, som i dette tilfælde binder BuslistMenuActivity til TrackABusProvideren.

Kodeudsnit 1: Binding til TrackABusProvider

```
1 if(ConnectivityChecker.hasInternet){
2   Intent intent = new Intent(BuslistMenuActivity.this, ↩
       TrackABusProvider.class);
3   startService(intent);
4   bindService(intent, Connection, Context.BIND_AUTO_CREATE);
5   }
```

Denne binding vil ske asynkront, og det er derfor ikke muligt at vide, hvornår den er færdig. Til dette formål bruges en ServiceConnection. Denne har en callback function, "onServiceConnected", der vil blive kaldt så snart der er blevet bundet til servicen. For at sikre at de funktioner, som kræver en bundet service, først bliver kaldt når dette er opfyldt, bliver kaldt i onServiceConnected. "getService"vil initialisere en instans af den service der er bliver bundet til, og herfra vil det være muligt at kalde de forskellige service funktioner, der er implementeret i TrackABusProvider klassen.

Kodeudsnit 2: ServiceConnection i BuslistMenuActivity

```
1 private ServiceConnection Connection = new ServiceConnection(){
2   @Override
3   public void onServiceConnected(ComponentName name, IBinder ↩
```

```
        service) {
4       LocalBinder  binder = (LocalBinder ) service;
5       BusProvider = binder.getService();
6       mBound = true;
7       UpdateArrayOfBusses();
8     }
9
10    @Override
11    public void onServiceDisconnected(ComponentName name) {
12      mBound = false;
13    }
14  };
```

Alle funktionerne i TrackABusProvider klassen bliver afviklet i deres egen tråd for ikke at blokere main/UI tråden. For at TrackABusProvider funktionerne kan sende data rigtigt tilbage til den klasse der kaldte funktionen, bliver der brugt en Message Handler.
På TrackABusProvider siden er dette blevet implementeret ved, at hver funktion tager imod en ReplyMessage og Handler som parameter. Når funktionen er færdig med at hente data fra SoapProvideren, bliver der oprettet en ny Message. Se *afsnit 9.2.1 Implementering af persistens i mobilapplikationen* for information om, hvordan MySQL databasen bliver tilgået. Messagen sendes over den medfølgende Handler, som er implementeret i den klasse, der kaldte servicen. ReplyMessage beskriver overfor Handleren, hvad den er blevet færdig med, så Handleren kan udføre det korrekte arbejde med dataen. På kodeudsnit 3 kan der ses et eksempel på en funktion i TrackABusProvider klassen. "GetbusRoute"tilgår SoapProvider klassen, for at hente en bestemt busrute, og alle de stoppesteder der hører til denne fra MySQL databasen. Data hentet herfra, bliver lagt i en ParcelableArrayList. Dette gør det muligt at sende custom datatyper med i de messages, der bliver sendt. Disse custom datatyper skal dog implementere Parcelable interfacet. Disse klasser kan ses på 23, under "Parcelable Modeller". Til sidst bliver der oprettet en ny Message, med den ReplyMessage der fulgte med som parameter, så Handleren ved hvilket arbejde den skal udføre. Denne message sendes til Handler parameteren.

Kodeudsnit 3: GetBusRoute() i TrackABusProvdider

```
1 public void GetBusRoute(final String busNumber, final int ↩
    ReplyMessage, final Handler replyTo){
2 try{
3   new Thread(new Runnable() {
4     public void run() {
5       mMessenger = new Messenger(replyTo);
6       Bundle b = new Bundle();
7       ArrayList<BusRoute> arg0 = soapProvider.GetBusRoute(↩
          busNumber);
8       ArrayList<BusStop> arg1 = soapProvider.GetBusStops(↩
          busNumber);
9       b.putParcelableArrayList("BusRoute", arg0);
10      b.putParcelableArrayList("BusStop", arg1);
11
12      Message bMsg = Message.obtain(null, ReplyMessage, 0, 0);
13      bMsg.setData(b);
14      try {
15      mMessenger.send(bMsg);
16      } catch (RemoteException e) {
17      e.printStackTrace();
18      }
19  }}).start();
20 }
```

De klasser der er bundet til TrackABusProvideren skal implementere en Message Handler, der skal bruges i sammenhæng med kald til funktionerne i TrackABusProvideren. Denne Handler står for at modtage beskederne, når den kaldte TrackABusProvider funktion er færdig. For at kalde en af TrackABusProvider funktionerne skal der, som beskrevet ovenstående, altid medsendes en ReplyMessage og en Message Handler. Handler parameteren er den Message Handler der er oprettet i den klasse som kalder servicen. På kodeudsnit 4 vises Message Handleren i BusMapActivity. Når den modtager en besked, vil den undersøge hvilken ReplyMessage der ligger i beskeden. Dette gøres for at være sikker på, at data håndteret korrekt. For at få det data der bliver sendt med i beskeden, bliver "getData"kaldt på beskeden.

Kodeudsnit 4: msgHandler i BusMapActivity

```
1 final static public int BUS_ROUTE_DONE = 1;
2 final static public int BUS_POS_DONE = 2;
3 ...
4 class msgHandler extends Handler{
5 @Override
6     public void handleMessage(Message msg) {
7       if(msg != null){
8         switch(msg.what){
9         case BUS_ROUTE_DONE:
10          ...
11          ArrayList<BusRoute> BusRoutes = msg.getData().↩
              getParcelableArrayList("BusRoute");
12          ArrayList<BusStop> BusStops = msg.getData().↩
              getParcelableArrayList("BusStop");
13          ...
14        break;
15        case BUS_POS_DONE:
16          ...
17        break;
```

For at kunne indtegne ruterne, stoppestederne og busserne skal det være muligt at have et kort at tegne på. Hertil er der blevet brugt Google Maps.[13] For at kunne bruge dette kort, kræves det at en API nøgle fra Google bliver tilføjet til manifestet. Denne kan anskaffes fra Googles API konsol[14].

Kodeudsnit 5: API nøgle i manifest

```
1 <meta-data
2   android:name="com.google.android.maps.v2.API_KEY"
3   android:value="AIzaSyC9qLxvm9yVIBJ5Dp0VqMapFvc4VLUlqu8"/>
```

For at vise selve kortet, skal der laves en layout fil, som indeholder et mapFragment, som vist i kodeudsnit 6. Det vil nu være muligt at se et kort i applikationen.

[13] Dette kan der læses mere om på https://developers.google.com/maps/documentation/android/
[14] Denne tilgås igennem https://code.google.com/apis/console

Kodeudsnit 6: mapFragment

```
1 <fragment
2   android:id="@+id/map"
3   android:layout_width="match_parent"
4   android:layout_height="wrap_content"
5   android:name="com.google.android.gms.maps.MapFragment"/>
```

Det er nu muligt at indtegne ruter, stoppesteder og busser på ruten, på dette fragment. Rute og bus relvant information hentes igennen TrackBusProvideren, og på kodeudsnit 7 kan det ses, hvordan en rute tegnes. Ruten bliver tegnet ved brug af en Polyline, som opbygges af de koordinater der hentes fra MySQL databasen.

Kodeudsnit 7: hvordan en Polyline bliver indtegnet på kortet

```
1 PolylineOptions pOption = new PolylineOptions().width(10).color(0↩
    x66ff0000);
2 for(int i = 0; i < points.size(); i++){
3   pOption.add(points.get(i).Position);
4 }
5 map.addPolyline(pOption);
```

Busstoppestederne bliver ligeså tegnet ind ved at bruge de GPS-koordinater, der er blevet hentet fra MySQL databasen. Disse koordinater bruges til at tegne markører på kortet. Der vil blive knyttet en ClickListener på alle markerne, som kaldes når et stoppested trykkes. Denne sørger for at starte tidsopdaterings funktionerne. På kodeudsnit 8 kan det ses, hvordan et stoppested tilføjes til kortet.

Kodeudsnit 8: Hvordan stoppesteder bliver indtegnet på kortet

```
1 for(int i = 0; i<stops.size(); i++){
2   map.addMarker(new MarkerOptions()
3   .position(stops.get(i).Position.Position).title(stops.get(i).↩
      Name)
4   .icon(BitmapDescriptorFactory.fromResource(R.drawable.teststop)↩
      ));
5 }
```

BusListAdapteren er lavet til det formål, således at hvert list element i BusListMenu-Activity, kan indeholde en togglebutton, en progressbar, foruden tekst. Dette gøres da

favoriserings metoder også startes fra en af disse elementer.
Mange steder i applikationen skal der bruges adgang til internettet, og der kan derfor forkomme fejl, hvis der ikke er tilgang hertil. Dette er blevet håndteret ved at implementere en BroadcastReceiver i klassen ConnectivityChecker, der abonnerer på, ændringer i netværksforbindelsen for både wifi og mobilt data netværk. På kodeudsnit 9, kan manifest filen ses, hvori det vises hvordan BroadcastReceiveren kobles til de nødvendige events. Når der sker et event undersøges der, om der er internet, og en statisk bool sættes til true hvis der er forbindelse, og false hvis der ikke er. Hertil er det muligt for applikationen til hver en tid at vide, om der er adgang til internettet.

Kodeudsnit 9: ConnectivityChecker i manifest filen

```
1        <receiver android:name="ConnectivityChecker">
2            <intent-filter>
3                <action android:name="android.net.conn.↩
                     CONNECTIVITY_CHANGE"/>
4                <action android:name="android.net.conn.↩
                     WIFI_STATE_CHANGED"/>
5            </intent-filter>
6        </receiver>
```

8.2.2 Komponent 2: Mobile service

Denne komponent har til formål at være mellemled mellem mobil applikationen og MySQL databasen. komponenten er blevet lavet, da mobil applikationen ikke må have direkte tilgang til MySQL databasen, da dette har store sikkerhedsmæssige implikationer. Et andet formål med denne web service er at gøre det gøre det nemt at udvikle til en anden mobil platform, da der ikke skal tænkes på database tilgang, men blot kaldes til servicen. Herudover abstraheres samtlige processeringstunge udregninger væk fra applikationen.

Design:

Mobil servicen bliver brugt til at hente data fra MySQL databasen som mobil applikationen skal bruge. Dette indebære at hente en liste af busruter, hente en bestemt rute, hente stoppestederne for en rute, hente positionen for alle busser på ruten og kalde den proceduren på databasen som udregner tiden til ankomst for den nærmeste bus ved et valgt

stoppested. Tidsudregningen kan der læses mere om i afsnit **??**. Web servicen er tilgængelig for alle, da de eneste funktionaliteter den udbyder er at hente data fra databasen. Med en åben web service er det muligt for alle at bruge data til at udvikle nye applikationer. [15] På kodeudsnit 10 kan der følges et eksempel på, hvad det kræver at kalde funkionen GetBusPos fra web servicen, ved brug af SOAP version 1.1.[16] Det kan ses at funktionen kræver en "busNumber"input parameter i form af en string. Implementeringen af SOAP på mobil applikations siden kan findes under afsnit *9.2.1: Implementering af persistens i mobil applikationen.*

Kodeudsnit 10: request til service function GetBusPos

```
1 POST /AndroidToMySQLWebService.asmx HTTP/1.1
2 Host: trackabus.dk
3 Content-Type: application/soap+xml; charset=utf-8
4 Content-Length: length
5 <?xml version="1.0" encoding="utf-8"?>
6 <soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-↩
    instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" ↩
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
7   <soap12:Body>
8     <GetbusPos xmlns="http://TrackABus.dk/Webservice/">
9       <busNumber>string</busNumber>
10    </GetbusPos>
11  </soap12:Body>
12 </soap12:Envelope>}
```

På kodeudsnit 11, kan det SOAP response, servicen returner ved fuldent kald til GetBusPos. Svaret kan ses som et XML. SOAP implementeringen på applikations siden, står for at håndtere læsningen at dette træ. Det kan ses at der retuneres en liste af Points, hvor hver Point indeholder en Lat, en Lng og et ID som alle er af datatypen string.

Kodeudsnit 11: response fra service function GetBusPos

```
1 HTTP/1.1 200 OK
2 Content-Type: application/soap+xml; charset=utf-8
3 Content-Length: length
```

[15] For en liste af funktioner i servicen, se http://trackabus.dk/AndroidToMySQLWebService.asmx
[16] For mere information om SOAP, se http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetbusPosResponse xmlns="http://TrackABus.dk/Webservice/">
      <GetbusPosResult>
        <Point>
          <Lat>string</Lat>
          <Lng>string</Lng>
          <ID>string</ID>
        </Point>
        <Point>
          <Lat>string</Lat>
          <Lng>string</Lng>
          <ID>string</ID>
        </Point>
      </GetbusPosResult>
    </GetbusPosResponse>
  </soap12:Body>
</soap12:Envelope>
```

8.2.3 Komponent 3: Administrations hjemmeside

Denne komponent har til formål at håndtere alle de adminitrative opgaver i systemet.

Specifikationer

Denne komponent består af fire delkomponenter:

- Den første delkomponent gør det muligt at tilføje en bus til systemet, fjerne den, eller rediger i en bus der allerede findes i systemet.
- Den anden delkomponent gør det muligt at tilføje eller fjerne en bus fra en rute der findes i systemet.
- Den tredje delkomponent gør det muligt at kunne oprette en ny busrute i systemet, ændrer i en allerede eksiterende busrute, eller slette en fra systemet.
- Den fjerde delkomponent består af muligheden for at kunne tilføje, ændre samt fjerne busstoppesteder fra systemet.

På figur 24 kan administrator systemets lagdelte struktur følges, i form af et klassediagram. Herpå kan det ses, hvordan klasserne er bygget op i en tre-lags model, samt hvordan de interagerer. Når der laves en association, betyder det at klasse tilgås statisk.

Figur 24: Klassediagram for simulator. Opbygget som trelags-model

Her følger en kort beskrivelse af hver klasse, samt den funktionalitet klassen tilføjer til systemet.

- Home index
  - Denne klasse består af viewet for home skærmen. Denne laver ikke andet arbejde, end at lade administratoren vælge, en delkomponent.
- Dir index
  - Denne klasse består af viewet for rute redigerings skærmen. Heri ligger alle funktionaliter administratoren kan tilgå, for at oprette, fjerne eller ændre en rute i systemet.
- Bus index
  - Denne klasse består af viewet for bus redigerings skærmen. Heri ligger alle funktionaliter administratoren kan tilgå, for at oprette, fjerne eller ændre en bus i systemet. Denne tilbyder yderligere muligheden for, at tilføje eller fjerne en bus på en rute.
- Stop index
  - Denne klasse består af viewet for stoppesteds redigerings skærmen. Heri ligger alle funktionaliter administratoren kan tilgå, for at oprette, fjerne eller ændre et stoppested i systemet.
- HomeController
  - Denne klasse står for at modtage kaldende fra Home viewet. Denne klasse gør dog intet i denne sammenhæng, da ingen kald skal modtages.
- DirController
  - Denne klasse står for at modtage kaldende fra rute viewet. Den sørger for, at kalde til data tilgangs laget, for at tilgå databasen, vedrørende rute relvante oplysninger. Desuden manipulerer den det data, der kan hentes fra viewet, således at data bliver gemt på korrekt form, samtidig med, at data hentet fra databasen bliver manipuleret, så det vises korrekt viewet.
- BusContoller

  - Denne klasse står for at modtage kaldende fra bus viewet. Den sørger for, at kalde til data tilgangs laget, for tilgå databasen, vedrørende bus relvante oplysninger. Desuden manipulerer den det data, der kan hentes fra viewet, således at data bliver gemt på korrekt form, samtidig med, at data hentet fra databasen bliver manipuleret, så det vises korrekt på viewet.

- StopController
  - Denne klasse står for at modtage kaldende fra stoppesteds viewet. Den sørger for, at kalde til data tilgangs laget, for tilgå databasen, vedrørende stoppesteds relvante oplysninger. Desuden manipulerer den det data, der kan hentes fra viewet, således at data bliver gemt på korrekt form, samtidig med, at data hentet fra databasen bliver manipuleret, så det vises korrekt på viewet.

- Waypoint og BusStops
  - Disse er model klasser. De anses derfor som datatyper, og har ingen relationer til eller fra sig.

- RouteMath
  - Denne klasse indeholder alt relevant matematik, som skal bruges under manipulering af data, før det gemmes på databasen. Alle funktioner er statiske, så ingen klasser opretter denne.

- JConverter
  - Denne klasse sørger udelukkende for at konvertere en vilkårlig liste, til et JSON objekt. Dette bruges i sammenhæng med, at listerne af data returneret fra databasen, skal ændres til et JSON objekt, før dette kan sættes på viewet.

- DBConnection
  - Denne klasses eneste ansvar er, at tilgå databasen. Derfor er det også den eneste der ligger under data tilgangs laget. Den implementerer specifikke metoder, til brug i de forskellige Controllers. Samtlige funktioner er statiske, så de kan tilgås uden at et objekt af klassen skal oprettes.

Alle disse delkomponenter udgør tilsammen en vigtig del af systemet, da det ikke vil være muligt at oprette de dataelementer, systemet består af, uden dem.

Design:

Hjemmesiden er blevet implementeret ved brug af Microsoft ASP.NET MVC 4 frameworket. Dette gør det nemt og hurtigt at implementere en sofistikeret og moderne hjemmeside, der følger gode design principper. MVC står for Model-View-Controller og følger de samme principper som MVVM angående 'separation of concerns'.

For at kunne indtegne busruter og stoppesteder skal der bruges et kort, til dette er der blevet brugt Google Maps JavaScript API[17], samt Google Directions API.[18]

Hjemmesiden består af fire views, hvor hver view har en controller knyttet til sig. Det første view administratoren vil se er startsiden, der linker til de tre andre views. Disse tre sørger for at håndtere alt vedrørende busser, stoppesteder samt busruter.

Viewet der håndtere alt om busserne, består af to dele.

Første del gør det muligt at tilføje en ny bus til systemet, fjerne en bus fra systemet og rediger IDet for en bus. Dette er blevet implementeret ved, at når view'et bliver loaded, bliver en JavaScript function kaldt. Denne kalder funktionen "GetAllBusses"i BusControlleren, der henter alle busser der er i MySQL databasen. Se *afsnit 9.2.3 Implementering af persistens i online værktøjet* for nærmere beskrivelse af hvordan databasen bliver tilgået. Til at lave dette kald fra JavaScript til controlleren, bliver der brugt AJAX. AJAX gør det muligt at udveksle data med controlleren og opdatere viewet uden det er nøvendiger at reloade hele hjemmesidensiden. AJAX kaldet kan ses på kodeudsnit 12.

Kodeudsnit 12: Ajax kald til controller funkionen "GetAllBusses"

```
1          $.ajax({
2              type: "POST",
3              url: "Bus/GetAllBusses",
4              dataType: "json",
```

[17] For mere information, se https://developers.google.com/maps/documentation/javascript/
[18] For mere information, se https://developers.google.com/maps/documentation/directions/

```
5             success: function (result) {
6                 var select = document.getElementById("busses");
7                 select.options.length = 0;
8                 for (var i = 0; i < result.length; i++) {
9                     select.options.add(new Option(result[i]));
10                    ListOfAllBusses.push(result[i]);
11                }
12            }
13        });
```

Når BusControlleren er færdig med at hente busserne, retuneres der et JSON object, og callback funkionen, der er defineret i success parameteren i AJAX, vil blive kaldt. "Result"parameteren på callback funktionen er returværdien fra GetAllBusses funktionen i BusControlleren, der i dette tilfælde er et JSON object. Denne vil indeholde en liste af alle bussernes IDer, hentet fra MySQL databasen. Callback funktionen tilføjer alle IDer i listen, til et HTML select element. Dette gør det muligt for administratoren at se, hvilke busser der er gemt i databasen. Administratoren har nu mulighed for at enten tilføje en ny bus, slette en bus, eller ændre IDet på en bus.

For at tilføje en bus, skriver administratoren bussens ID ind i feltet: "Bus ID", hvorefter der trykkes på knappen "Add". Dette vil tilføje bussen til listen af busser, direkte igennem JavaScript. Administratoren kan tilføje så mange busser til listen, som der ønskes. Det kan også fjernes en bus fra listen. Dette gøres ved at vælge en bus og trykke på knappen "Remove", hvilket fjerner elementet fra listen igennem JavaSCript. Til sidst er også mulighed for at ændre IDet på en bus. Dette gøres ved at vælge en bus fra listen, ændre dennes ID i "Bus ID"feltet, og trykke på "Rename"knappen, som giver bussen det nye ID, igennem JavaScript. Først ved tryk på "Save"knappen vil ændringerne blive tilføjet til MySQL databasen. Dette sker igennem et AJAX kald til BusControlleren, der kalder "SaveBusChanges"funktionen. Denne funktion modtager listen af busser, hvori ændringerne administratoren har foretaget også indeholder, samt en liste af alle busserne på databasen. Funktionen sammenligner de to lister, finder de busser der er blevet tilføjet, de som er blevet fjernet og dem som har fået nyt ID, hvorefter denne vil slette, tilføje eller ændre de relevante busser.

Anden del af dette view gør det muligt at tilføje busser til- og fjerne busser fra en busrute. Denne del består af tre lister, hvor der er en, der indeholder alle busruter hentet fra databasen, en der indeholder samtlige busser, der ikke har en busrute knyttet til sig, samt en der viser, hvilke busser der køre på en valgt busrute. I dette views "onload"funktion bliver der, ud over den overnævte "GetAllBusses"funktion, også kaldt to andre funktioner. Dette forgår igen gennem to AJAX kald til BusControlleren. Den første henter navnene på alle busruter fra databasen og den anden henter en liste af IDer for alle de busser, der ikke er tilknyttet en rute. Disse to AJAX kald er magen til AJAX kaldet vist i kodeudsnit: 12, hvor den eneste forskel er, hvilken BusController funktion der bliver kaldt, samt hvilet HTML select element der bliver tilføjet til. Det er nu muligt for administratoren at vælge en af busruterne. Dette vil starte et 'onchange' event, der laver endnu et AJAX kald til BusControlleren for at hente alle de busser, der køre på den valgte rute. De hentede busser, vil vises i listen "Busses on route". Der kan nu tilføjes busser fra listen "Avaliable busses"til listen "Busses on route". Ved tryk på knappen "Save"vil de busser, der er blevet flyttet til listen "Busses on route"bliver opdateret i databasen, således at de nu er knyttet til den valgte rute. Dette håndteres ved en AJAX kald til BusControllerens "SaveChanges"funktion, som tilgår databasen og ændrer værdierne for bussen. Dette kan der læses mere om i afsnittet *9.2.3: Implementering af persistens i online værktøjet*

Busrute viewet gør det muligt at oprette en ny busrute, ændrer i en, der alleredes findes, samt slette en givet busrute fra systemet. For at indtegne en busrute, kræver det et kort. Hertil er der blevet brugt Google Maps API[19] og Google Directions API.[20] For at få vist kortet på hjemmesiden, er det nødvendigt at det bliver initialiseret. Først og fremmest skal man have lavet plads til det på siden. Dette kan ses på kodeudsnit 13

Kodeudsnit 13: Div til google maps

```
1 <section id="Map">
2     <div id="map-canvas"></div>
3 </section>
```

[19] For mere information, se https://developers.google.com/maps/documentation/javascript/
[20] For mere information, se https://developers.google.com/maps/documentation/directions/

Når HTML body elementet er loaded bliver dens onload"event kaldt. Dette kalder en JavaScript funktion, der initializere kortet samt dennes Google directions service.
Først bliver der defineret en style, som kortet skal bruge. Heri sørges der for at "Points of interest"bliver fjernet, da der bare skal bruges et tomt kort. Dernæst bliver der oprettet et mapOptions object definerer, at kortets start position sættes til at vise Aarhus. Desuden sættes der også, at korttypen er ROADMAP, da dette vil vise kortet som et simpelt kort, hvor der kun er veje. StreetViewControl bliver sat til false, da det er en feature der ikke er relevant for systemem. Denne opsætning kan ses på figur 14.

Kodeudsnit 14: Map opsætning

```
1 var featureOpts = [{
2   featureType: 'poi',
3   stylers: [
4     { visibility: 'off' }]
5 }];
6 var Aarhus = new google.maps.LatLng(56.155955, 10.205011);
7 var mapOptions = {
8   zoom: 13,
9   mapTypeId: google.maps.MapTypeId.ROADMAP,
10  center: Aarhus,
11  streetViewControl: false,
12  styles: featureOpts
13 };
```

Efter at have defineret mapOptions bliver der oprettet et map object. Dette object skal have det HTML map-canvas element som ses på kodeudsnit 13, samt den mapOptions som laver i kodeudsnit 14. Denne oprettelse kan ses på kodeudsnit 15.

Kodeudsnit 15: Map object init

```
1 map = new google.maps.Map(document.getElementById('map-canvas'), ↩
    mapOptions);
```

Kortet er nu blevet initialiseret og vil blive vist på siden. Det næste der bliver initialiseret er Google direction rendereren. Denne bliver brugt til at vise en rute på kortet mellem to givne punkter. Først bliver der defineret de options som ruten skal bruge. Dette indebærer at det skal være muligt at trække i ruten, for at ændre på den vej, den skal følge.

Herudover muliggøres det også at klikke på de markører, der repræsenterer start og slut punktet for ruten. Dette rendererOptions object bliver derefter brugt i constructoren for DirectionsRendereren, som senere bliver brugt til at tegne ruten på kortet. Dette kan ses på kodeudnsit 16.

Kodeudsnit 16: DirectionsRenderer opsætning

```
1 rendererOptions = {
2 map: map,
3   draggable: true,
4   markerOptions: {
5     clickable: true
6   },
7   suppressInfoWindows: true
8 };
9 directionsDisplay = new google.maps.DirectionsRenderer(↩
    rendererOptions);
```

Efter kortet og DirectionRenderen er blevet initialiseret, bliver der sat en listener på kortet. Denne lytter efter om der bliver trykket et vilkårligt sted på kortet. Dette kan ses på kodeudsnit 17.

Kodeudsnit 17: map klik listener

```
1 google.maps.event.addListener(map, 'click', function (event) {
2   if (startPoint == null && endPoint == null) //No markers, set ↩
      first
3     startPoint = new google.maps.Marker({
4       map: map,
5       draggable: true,
6       position: event.latLng
7     });
8   else if (startPoint != null && endPoint == null) {//if 1 ↩
      markers, set last markers
9     endPoint = new google.maps.Marker({
10      map: map,
11      draggable: true,
12      position: event.latLng
13    });
14    calcRoute(startPoint, endPoint);
15    ClearMarkers();
```

```
16 }
```

Når der bliver trykket på kortet vil listeneren undersøg, om der er blevet trykket på kortet tidligere. Hvis der ikke er, vil der blive placeret en markør på kortet, der hvor der blev trykket. Denne markør symboliserer busrutens første endestation. Hvis der allerede er en markør på kortet vil der bliver placeret endnu en markør, som symboliserer busrutens sidst endestation. Efter begge markører er blevet placeret, vil functionen "calcRoute"blive kaldt, med de to satte markører. Denne function bruger Google direction service til at udregne en rute mellem de to punkter. Dette bliver gjort ved at lave et request object der definerer start og slut GPS koordinaterne for ruten. Disse koordinater bliver taget fra de to markøre, medsendt i funktionen. Desuden sættes travelMode til at være DRIVING, så ruten vil blive vist den umiddelbart hurtigste rute i bil. Request objektet bruges i den oprettede Direction Service, til at lave ruten. Dette ses på kodeudsnit lst:calcRoute.

Kodeudsnit 18: calcRoute function

```
1 function calcRoute(start, end) {
2   request = {
3     origin: start.position,
4     destination: end.position,
5     travelMode: google.maps.TravelMode.DRIVING
6   };
7   directionsService.route(request, function (response, status) {
8     if (status == google.maps.DirectionsStatus.OK) {
9       route = response.routes[0];
10      directionsDisplayArray[0].setDirections(response);
11    }
12  });
13 }
```

Start og slut markørene har også en click listener på sig. Disse tages i brug når en kompleks rute skal oprettes. Dette gøres ved, at der bliver lavet endnu en DirectionsRenderer, som tegner en rute mellem den trykkede markør, og endnu et sted på kortet, hvor der trykkes.

Der er også blevet sat en listener på de to DirectionRenderers, der lytter ændringer i ruten. I tilfælde af at ruten bliver ændret, vil callback funktionen blive kaldt på disse

to. Heri vil der, om det første, blive der sat et kort delay. Dette gøres da ruten skal færdigudregnes før de forskellige brugte properties, bliver sat. Efter delayet bliver der itereret igennem alle properties, for at finde den af typen "Markers". Denne indeholder de markører, der symboliser start og slut punkterne, samt alle de waypoints der må være blevet lavet på ruten når den ændres. GPS koordinaterne for disse markører vil blive brugt når hele ruten skal gemmes i MySQL databasen. Til sidst vil informationen om ruten, og herunder de GPS koordinater, der bliver brugt til at tegne ruten, gemt i en variabel, der senere bliver brugt når ruten skal. Hele denne situation kan ses på kodeudnsit 19.

Kodeudsnit 19: DirectionsRenderer listener

```
1 google.maps.event.addListener(directionsDisplay, '↩
    directions_changed', function () {
2   var that = this;
3   setTimeout(function () {//et kort delay, så ruten kan nå at ↩
      blive udregnet helt
4     for (var k in that) {//kigger alle properties igennem efter ↩
        den der skal bruges.
5       if (typeof that[k].markers != 'undefined') {//Hvis man ↩
          finder den man skal bruge
6         var markers = that[k].markers;
7         waypoints = [];
8         for (var i = 0; i < markers.length; ++i) {
9           waypoints.push(markers[i].position);
10          markers[i].setZIndex(1);
11          StartEndMarkers.push(markers[i]);
12        };
13      }
14    }
15    temp = that.directions.routes;
16  }, 100);
17 });
```

For at kunne sætte stoppestederne på en rute, kaldes funktionen "SetBusStopsOnMap", der henter navnene på de stoppesteder, som er defineret på viewet i listen af stoppesteder, der skal være på ruten. Efter alle navnene er hentet, bliver der lavet et AJAX kald til DirController funktionen "GetLatLng", der bruger navnene på stoppestederne til at hente deres GPS koordinater. Disse koordinater bliver sendt tilbage til AJAX callback funktionen, som en succes parameter i form af et JSON objekt. Dette objekt bruges til at

indtegne stoppestederne på kortet.

På kodeudsnit 20 kan "SaveRouteAndStops"funktionen ses. Denne kaldes når "Save route"knappen bliver trykket. Funktionen står for at finde det rutedata der skal gemmes, samt kalde til DirControllerens "Save"funktion igennem et AJAX kald. Først findes alle de GPS koordinater, der bruges til at optegne rute. Dette gøres igennem et kald til "getRoutePath". Denne undersøger hver path, step og leg af ruten.[21]. Heri bruges variablen "temp"fra kodeudsnit 19, og vil ved fuldendelse returnere en række GPS koordinater, som ruten er bygget omkring. I denne sammenhæng ses stoppesteder og waypoints ikke som punkter. Desuden sendes tidligere fundet stoppesteder og waypoints, samt eventuelle subruter hvis ruten er kompleks, også.

Kodeudsnit 20: SaveRouteAndStops

```
1 function SaveRouteAndStops() {
2 $.ajax({
3   type: "POST",
4   url: '@Url.Action("Save", "Dir")',
5   dataType: "json",
6   traditional: true,
7   data: {
8     route: getRoutePath(),
9     routeWayPoints: waypoints,
10    stops: stopsToSave,
11    SubRoutes: SplitRoute(SubRouteArray),
12    SubrouteWaypoint: SubRouteWaypoints,
13    RouteNumber: document.getElementById("RouteNumber").value,
14    contentType: "application/json; charset=utf-8"
15 }
```

I "Save"funktionen kaldes der til funktionen "CalculateBusStopsForRoute", der udregner mellem hvilke rute punkter stoppestederne skal ligge. På kodeudsnit 21 kan denne udregning ses. Den første del af foreach-løkken, som kigger alle stoppesteder på ruten igennem, hvorefter stoppestedets position hentes. Dette vil ikke vises i kodeudsnittet. "Calcu-

[21]For mere information om disse, se https://developers.google.com/maps/documentation/directions/

lateBusStopToRouteDist"funktionen kan der læses mere om i afsnittet *8.2.5: Komponent: Matematik*, "Tætteste punkt på en linje".

Kodeudsnit 21: Udregninger af placering af stoppesteder på rute

```
for (int k = 0; k < chosenRouteLatLng.Count - 2; k = k + 2)
{
  currentDist =
    RouteMath.CalculateBusStopToRouteDist(
    stop.Lat, stop.Lng,
    decimal.Parse(chosenRouteLatLng[k]),
    decimal.Parse(chosenRouteLatLng[k + 1]),
    decimal.Parse(chosenRouteLatLng[k + 2]),
    decimal.Parse(chosenRouteLatLng[k + 3]));
  if ((currentDist < leastDist || leastDist == -1)
     && currentDist <= 15 && currentDist != -1)
  {
    leastDist = currentDist;
    pointBeforeStopIndex = k / 2;
  }
}
if (stops.IndexOf(s) == 0 )
{
  RouteWithStopsID.Insert(0, stop.ID.ToString());
  StopOnRoute.Add(s);
  stopCounter++;
  continue;
}
else if (stops.IndexOf(s) == stops.Count - 1 )
{
  RouteWithStopsID.Add(stop.ID.ToString());
  StopOnRoute.Add(s);
  stopCounter++;
  continue;
  }
else if (leastDist != -1)
{
  RouteWithStopsID.Insert(
    pointBeforeStopIndex + stopCounter + 1,
     stop.ID.ToString());
  StopOnRoute.Add(s);
  stopCounter++;
}
```

På dette kodeudsnit, kan processen der udføres for hvert stoppested følges. For-løkken

itererer igennem samtlige rutepunkter, og udregner distancen til den linje der er spændt mellem det nuværende rutepunkt og det næste. Hvis den udregnede længde er mindre end den tidligere korteste længde, sættes det nuværende rutepunkt, som værende det busstoppestedet skal indsættes før. Desuden undersøges der også om distancen er mindre en 25 meter, da det er muligt at stoppestedet ligger tilpas nok forskudt til ruten, så ingen rutelængder vil opfange den. Listen af punkterne der udgør ruten, består af både bredde og længdegrader, i den rækkefølge. Derfor er listen dobbelt så lang som den egentlige punkt-liste, der består af ID'er. Indexet for det fundne tætteste punkt, skal derfor halveres. Når samtlige punkter på ruten er undersøget, findes der ud af, om stoppestedet er det første eller det sidste på ruten. Dette gøres da det første og sidste stoppested altid skal være endestationer, og derfor ligges først eller sidst på ruten. Dette gøres uden hensyn til, hvilket punkt der egentligt er tættest på. Hvis stoppestedet ikke er det første eller sidste, indsættes ID'et for dette i listen af ID'erne for punkternerne. Hver gang et stoppested tilføjes, forøges listen med en, så derfor skal stoppestedet sættes i listen der svarende til det fundne ID plus antal tilføjede stoppesteder plus en, da det skal være punktet efter, det fundne ID. Stoppestedet tilføjes desuden til listen af stoppestedsnavne, og antallet af stoppesteder inkrementeres. Herefter returneres både listen af punkt ID'er og listen af stoppestedsnavne.
Efter alle udregnigner er fortaget, bliver alt data indsat i MySQL databasen. *Se afsnit 9.2.3 Implementering af persistens i online værktøjet* for en beskrivelse af, hvordan databasen bliver tilgået.

Det sidste view omhandler funktionerne for at tilføje nye stoppesteder, ændre position og navn for eksiterende stoppesteder, samt slette stoppesteder fra databasen. For at kunne oprette nye stoppesteder, bliver der igen brugt Google Maps API[22]. Dette bliver initialiseret på samme måde som beskrevet tidligere i afsnittet, dog uden Google Direction Services, da det kun er enkelte punkter på kortet der skal gemmes. For at oprette et nyt stoppested trykkes der på kortet, hvorefter kortets listener event bliver kaldt. Dette event vil sætte en marker på kortet, der hvor der blev trykket, som symboliserer stoppestedets position. Dette kan ses på kodeudsnit 22.

[22]For mere information, se https://developers.google.com/maps/documentation/javascript/

Kodeudsnit 22: Stoppested map listener

```
1 google.maps.event.addListener(map, 'click', function (event) {
2   if (markers.length <= 0) {
3     var mark = new google.maps.Marker({
4       map: map,
5       draggable: true,
6       position: event.latLng,
7       title: markers.length.toString()
8 });
```

Når et stoppested sat kan der trykkes på "Save"knappen, som kalder JavaScript funktionen "SaveStopsToDB". Denne vil tilgå StopControllerens "Save"igennem et AJAX kald, hvorefter stoppestedet vil gemmes med dennes position og indskrevne navn.

For at ændre positionen og navnet på et allerede eksiterende stoppested, vælges der et stoppested fra listen. Når dette vælges vil listens "onchange"event blive kaldt. Dette event vil kalde "SetSelectedOnMap"funktionen, hvor et AJAX element bliver oprettet, og StopControllerens "GetPosition"funktion vil kaldes, med det valgte stoppestedsnavn. Denne vil returnere det valgte stoppesteds GPS-koordinater, og returnere disse som et JSON objekt. Dette vil bruges til at sætte en markør på kortet, der repræsenterer stoppestedets position. Når denne markørs sættes vil den click listener, som i ved et click event sletter markøren. Det valgte navn bliver desuden indskrevet i "Stop name"feltet. Nu kan navnet på stoppestedet ændres, samt markøren kan trækkes til en ny position. Når ændringerne ønskes gemt trykkes der på "Save changes"knappen, som kalder JavaScript funktionen "SaveChangesToStop"som kalder til funktionen "SaveChangeToStop"i StopControlleren. Denne opdaterer stoppestedets navn og position i MySQL databasen, hvorefter stoppestedets navn, vil blive ændret i listen.

For at slette et stoppested, vælges der et fra listen som før, hvorefter der trykkes på knappen "Delete stop". Dette kalder "DeleteStopsFromDB"funktionen, som kalder "Delete"funktionen i StopControlleren. Til denne funktionen sendes stoppesteds navnet, som stoppestedet slettes ud fra i MySQL databasen. Herefter vil listen blive opdateret, så det fjernede stoppested ikke længere vises i listen.

8.2.4 Komponent 4: Simulator

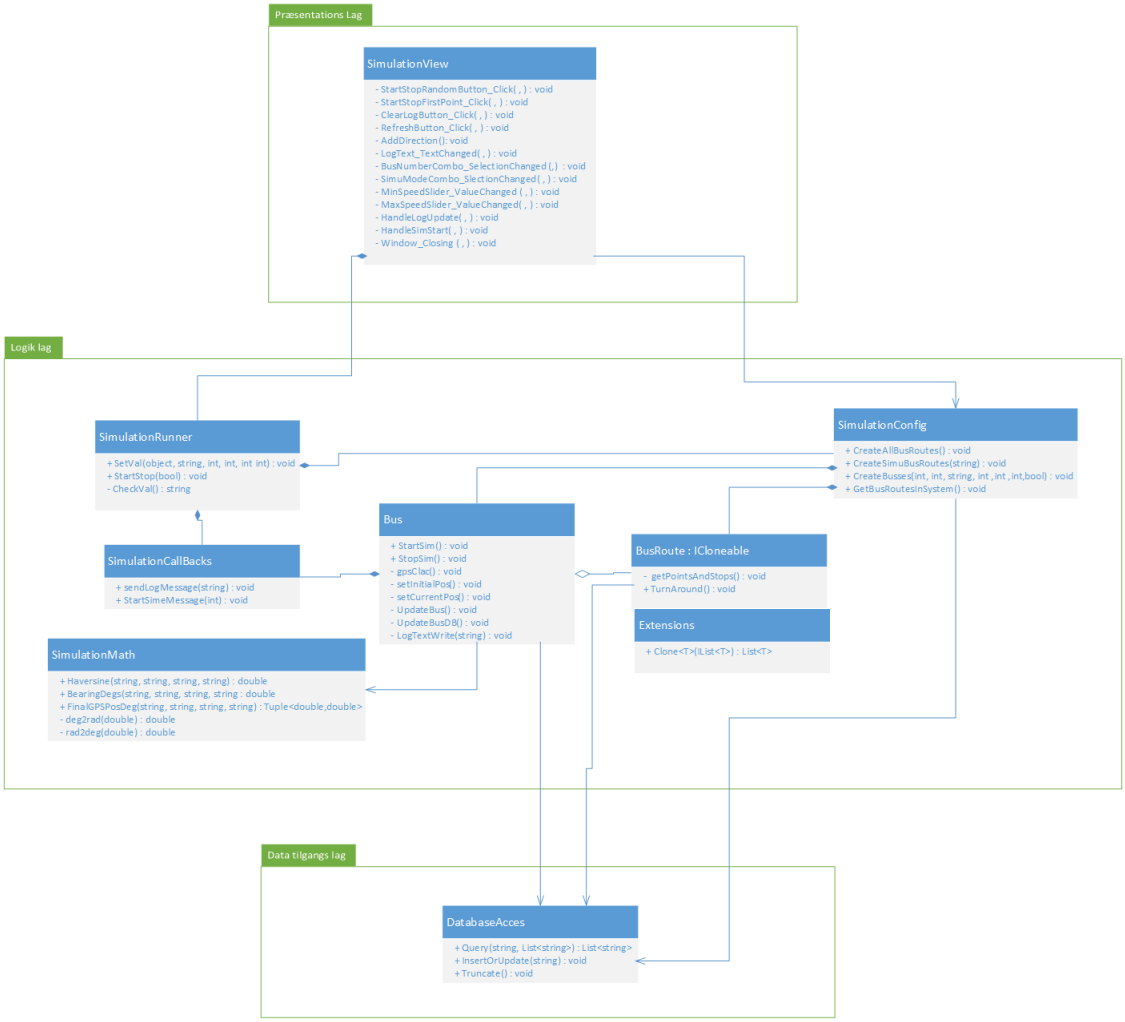
Denne komponent har til formål at beskrive bus simulatoren og dennes funktioner.

Specifikationer

Simulatoren er bygget som en WPF applikation, hvor .NET framework version 4.5 er blevet brugt. Den er opbygget efter tre-lags modellen. Hertil er der designet en række klasser til at håndtere logik-laget, samt en til at håndtere data-tilgangs laget. Præsententationslaget består udelukkende af ét view. Heri sørges der for, at events bliver håndteret samt påhægtet og afhægtet, når det er nødvendigt. På figur 25 ses et klasse diagram, hvorpå det vises, hvordan simulatoren er bygget op. Heri kan klassernes funktioner, samt deres relationer til andre klasser, ses. De steder hvor der er en association relation, betyder det, at klassen enten tilgår en statisk funktion eller statisk attribut i den associerede klasse.

Her følger en kort beskrivelse af hver klasse:

- SimulationView
  - Denne klasse består udelukkende af event-handlers, eller hjælpefunktioner dertil. Den opretter en SimulatorRunner, som igangsættes ved et tryk på en af de to start-knapper. Udover de to start knapper, er der lavet to event handlers, som bruges når der fra logik-laget skal ske ændringer i viewet. Dette sker i sammenhæng med, at en en log-besked skal tilføjes, eller simulatoren igangsættes og viewet skal ændres til en startet/stoppet tilstand.
- SimulationRunner
  - Denne klasse sørger for at starte og stoppe simulatoren. Når simulatoren startes sørger klassen for at simulatoren opsættes. Der undersøges om opsætningen er valid, og opstarts processen annuleres, hvis den ikke er. Denne klasse er den eneste i systemet der instantiere SimulationConfig, da det er muligt at hente ruter og busser statisk, men ikke oprette dem. SimulationRunner sørger derfor også for ruter og busser bliver oprettet.
- SimulationConfig



Figur 25: Klassediagram for simulator. Opbygget som trelags-model

  - Denne klasse er bindeledet mellem simulations enhederne, altså alt information om busser og ruter. Hvis klassen er instantieret kan relevanter busruter og busser hentes, og sættes i der respektive lister. Listen af kørende busser, og listen af samtlige busruter, som har en bus koblet på sig, kan tilgås statisk.

- SimulationMath
  - Klassen bruges i sammenhæng med udregning af ny position for en simuleret bus. Hver Bus klasse instantierer SimulationMath ved oprettelse.

- SimulationCallbacks
  - Klassen indeholder events til at håndtere viewændringer fra logik laget. Dette gøres igennem custom events som hægtes på viewet ved oprettelse. Der er kun to klasser der bruger SimulationCallbacks; SimulationRunner og Bus.

- BusRoute
  - Denne klasse er repræsentationen af en busrute, i simulatoren. Ved oprettelse sørges der for at rutens punkter hentes fra databasen. I denne sammenhæng hentes punkterne for stoppestederne ikke, da disse ikke relevante for simulatoren. Navnene på rutens stoppesteder hentes dog ud, da de skal tages i brug når ruten skal vende. Dette er kun relevant når ruten er kompleks, og det skal undersøges hvilken subrute bussen nu skal køres på. Når bussen vender, vendes hele ruten. Dette betyder også at hver bus skal have sin egen instans af ruten den kører på, og til det formål implementerer BusRoute IClonable, og indeholder derfor funktionen "Clone". Denne returner en ny instans, men en kopi af den givne rute. Hvis dette ikke gøres, vil det være den samme busrute reference, samtlige busser kører på, hvilket vil betyder, at når én bus vender, vil alle andre busser på samme rute også vende. "Clone"funktionen bruges i sammenhæng med Extensions klassen.

- Extensions
  - Når et BusRoute objekt skal kopieres, bruges metoden "Clone", men hvis det er en liste af BusRoute objekter der skal kopieres, er det nødvendigt at oprette

en extension metode til en liste. Denne metode har til formål at returne en ny liste, som er en klonet kopi af den gamle. Denne funktion kan dog kun bruges, hvis elementerne i listen implementerer IClonable, som i dette tilfælde er BusRoute objekter.

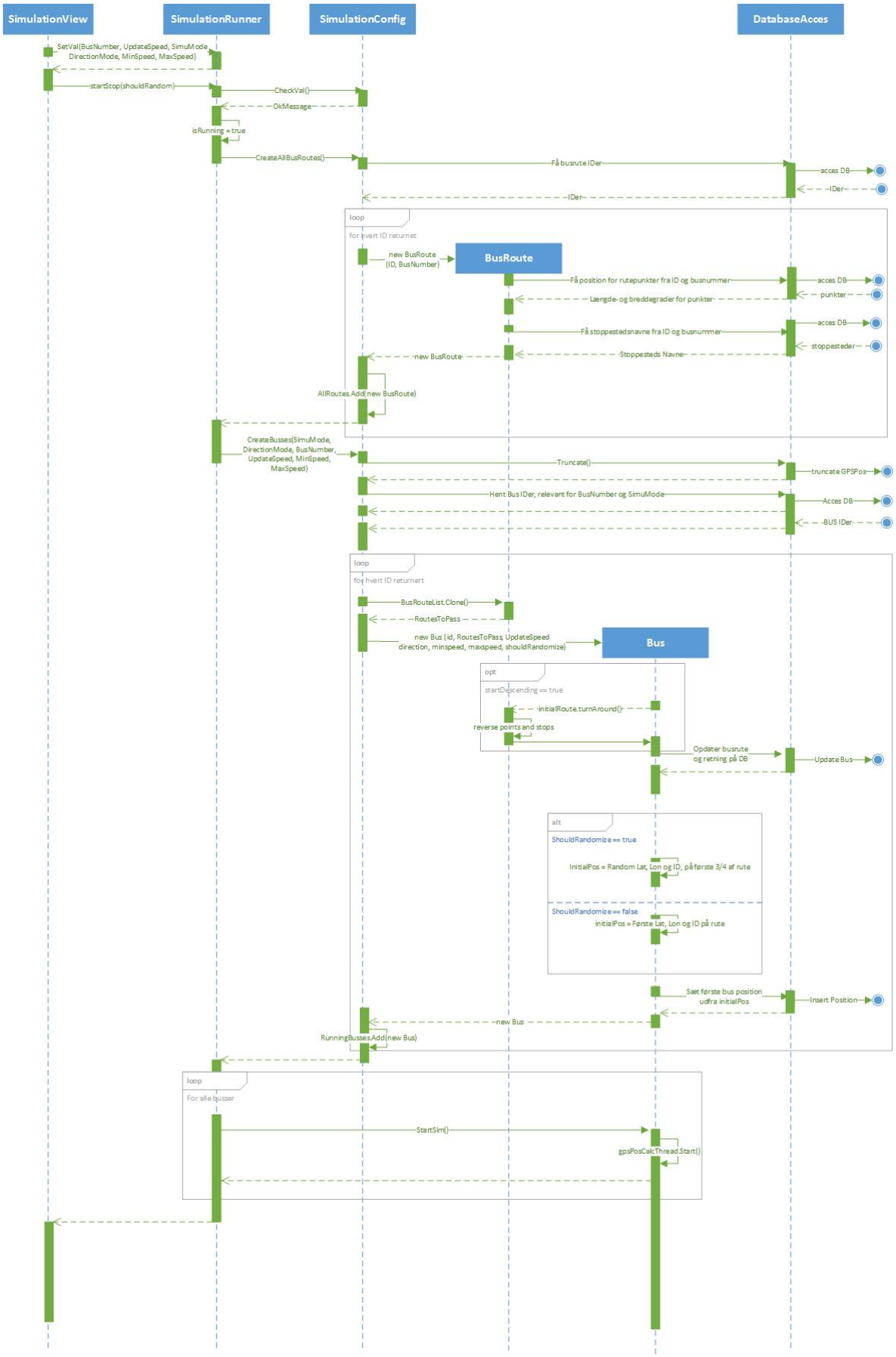
- Bus
  - Klassen er den simulerede version af en fysisk bus, og indeholder derfor alle funktionaliteter en bus har, for at kunne køre på en rute. Det er også i denne klasse den væsentligste simulering finder sted, idet denne klasse indeholder funktionen til at bestemme nye koordinater for bussen. Når et Bus objekt bliver instantieret, oprettes der samtidig en tråd til denne. Det vil sige, at hver bus har sin egen tråd, og kører derfor uafhængigt af resten af systemet, med den undtagelser af, at den kaster et logging event igennem SimulationCallbacks klassen. Selvom flere busser kan kører på samme rute, har bussen kun en kopi af ruten, hvilket betyder, at nå én bus vender, vendes hele kopi-ruten. Den tager desuden også beslutningen, om hvilken rute der skal køres på, hvis ruten er kompleks.
- DatabaseAcces
  - Indeholder funktioner til at tilgå datbasen. Mere information om denne klasse kan findes under afsnit *9.2.2: Implementering af persistens i simulator*.

Design

Der eksisterer to vigtige implementeringer i simulatoren, som udgør hovedfunktionaliteten; Start af simulering, samt udregning af ny position til en bus.

På figur 26 ses der et sekvensdiagram, der beskrives handlingsforløbet ved start af simulering. Handlings forløbet starter når en af de to start-knapper trykkes, og fuldføres når hver oprettet bustråd er startet. Sekvensdiagrammet repræsenterer solskinsscenariet, hvori samtlige konfigurationer er korrekte og internettet kan tilgås. Først sættes simulationsværdierne i SimulatorRunner. Disse består af de brugerdefinerede værdier, der kan sættes på GUIen. Herefter igangsættes startningsproceduren, ved et kald til "StartStop".

De brugerdefinerede værdier undersøges for validitet (eg. om der er nogen der ikke er sat, valgt eller skrevet korrekt), og simulatoren sættes til at være kørende. Samtlige busruter, som har minimum én bus knyttet til sig oprettes. Dette gøres igennem SimulatorConfig, hvor rutenumre og ID'er hentes, og BusRoute objekterne oprettes. Constructoren for BusRoute, sørger for at hente stoppesteder og punkter relevant for den givne busrute og sættes i objektet. Samtlige BusRoute objekter gemmes statisk i SimulationConfig. Herefter oprettes alle busser, på baggrund af hvilken måde der simuleres samt, hvilken retning busserne skal køres. Mere information omkring simulerings muligheder kan findes under afsnit *3.2: Grænseflader til eksterne system aktører, under afsnittest "Simulator"*. Ved oprettelse bestemmes der, hvilken subrute af den givne busrute, bussen skal køre på. Dette gøres i tilfælde af, at busruten er kompleks. Dette sker tilfældigt, uden hensyn til, hvilken simulerings mulighed er valgt. Herefter sættes ruten og den retning bussen kører, i databasen. Til sidst findes den bussens startposition. Hvis bussen skal starte først på ruten, vælges det første punkt fra listen af punkter i bussens tilknytede BusRoute objekt. Hvis ikke, vælges der et tilfældigt punkt på den første tre fjerdedel af ruten. Dette gøres, så der ikke ved et tilfælde vælges et punkt, der er helt i slutningen af ruten. Bussen gemmes herefter i listen af alle busser, og udregningstråden for hver bus startes.



Figur 26: Sekvensdiagram over start af simuelering

Formålet med simulatoren er, at en bus kan køre på en rute, uden behov en fysisk bus. På kodeudnit 23, 24 og 25 kan processen, der står for udregning af ny position ses. Selve udregnings processen er delt op i to kodeudsnit, hvor det første kodeudsnit omhandler initialisering af udregning samt udregninger på rutepunkter, og andet kodeudsnit omhandler bestemmelse af nyt rutepunkt. Kodeudsnit 25 omhandler busvending samt eventuelt ruteskift.

Kodeudsnit 23: Udregning af ny position del 1.

```
1 while (true)
2 {
3   double nextSpeed =SimulationConfig.rand.Next(minSpeed, maxSpeed+1) ;
4   double travelLengthMeters = speed * (1000d / 3600d) * updateSpeed;
5   double currentLength = 0;
6   double nextLength = 0;
7   double brng;
8   if(indexCounter == -1)
9     indexCounter = initialPosIndex + 1;
10
11  while (currentLength < travelLengthMeters)
12  {
13
14    if(indexCounter == initialRoute.points.Count - 1)
15    {
16      currentPos = new Tuple<double,double>(double.Parse(initialRoute.points[indexCounter].Item1),
          double.Parse(initialRoute.points[indexCounter].Item2));
17      UpdateBus();
18      break;
19    }
20
21    if (currentPos.Item1 != 0 && currentPos.Item2 != 0 && nextLength == 0)
22    {
23      nextLength = sMath.Haversine(currentPos.Item1.ToString(),
24        currentPos.Item2.ToString(),
25        initialRoute.points[indexCounter].Item1,
26        initialRoute.points[indexCounter].Item2);
27
28      brng = sMath.BearingDegs(currentPos.Item1.ToString(),
```

```
29          currentPos.Item2.ToString(),
30          initialRoute.points[indexCounter].Item1,
31          initialRoute.points[indexCounter].Item2);
32      }
33      else
34      {
35        nextLength = sMath.Haversine(
36          initialRoute.points[indexCounter - 1].Item1,
37          initialRoute.points[indexCounter - 1].Item2,
38          initialRoute.points[indexCounter].Item1,
39          initialRoute.points[indexCounter].Item2);
40        brng = sMath.BearingDegs(
41          initialRoute.points[indexCounter - 1].Item1,
42          initialRoute.points[indexCounter - 1].Item2,
43          initialRoute.points[indexCounter].Item1,
44          initialRoute.points[indexCounter].Item2);
45      }
46  ...
```

Det første der sker under udregningen er, at det bestemmes, hvor hurtigt bussen skal køre ved denne opdatering. Dette er en tilfældig værdi mellem den satte minimums og maksimums hastighed. Denne hastighed bruges til at udregne hvor langt bussen skal køre. Da hastigheden er udtrykt ved kilometer i timen, konverteres den til meter i sekundet, da meter og sekunder er de enheder der arbejdes med. Tiden mellem hver opdatering ganges på, da dette er et udtryk for, hvor lang tid bussen har kørt med den givne hastighed. Hvis dette er første gang opdateringen foregår, sættes indexCounter til det valgte start index plus 1, da dette symboliserer det næste rutepunkt som bussen ikke er kørt forbi. Herefter startes udregninger for nyt rutepunkt. While-løkken symboliserer et inkrementerende rutestyke, hvor currentLength er det stykke, der er blevet udregnet og travelLengthMeters er det stykke der skal rejses. Der undersøges om sidste rutepunkt er nået, og hvis det er, sættes bussens position til dette punkt, og ruten vendes. Dette beskrives efter kodeudsnit 25. Herefter udregnes det næste rutestykke, og hvilken kurs dette rutestykke følger. Udregningen af længden af rutestykket og kursen kan findes i afsnittet *8.2.5: Komponent 5: Anvendt matematik* under "Haversine"og "Bearing". Hvis der endnu ikke er udregnet et linjestykke, og hvis bussens nuværende position er sat, udregnes næste linjestykke og kurs udfra bussens position, og næste rutepunkt. Hvis ikke, udregnes der mellem det nuværende og næste rutepunkt. Udregningen fortsætter på kodeudsnit 24.

Kodeudsnit 24: Udregning af ny position del 2.

```
1  ...
2
3        if (nextLength + currentLength > travelLengthMeters)
4        {
5          double missingLength = travelLengthMeters - currentLength;
6
7          if (currentPos.Item1 != 0 && currentPos.Item2 != 0 && currentLength == 0)
8          {
9            currentPos = sMath.finalGPSPosDeg(
10             currentPos.Item1.ToString(),
11             currentPos.Item2.ToString(),
12             brng, missingLength);
13         }
14         else
15         {
16           currentPos = sMath.finalGPSPosDeg(
17             initialRoute.points[indexCounter - 1].Item1,
18             initialRoute.points[indexCounter - 1].Item2,
19             brng, missingLength);
20         }
21         SetCurrentPos();
22         break ;
23       }
24       else
25       {
26         currentLength += nextLength;
27       }
28       string currPosMsg = "Bus " + bID.ToString() +
29         ", new endpoint reached, index: "
30         + (indexCounter + 1).ToString();
31       LogTextWrite(currPosMsg);
32       indexCounter++;
33     }
34     Thread.Sleep(updateSpeed * 1000);
35   }
36 }
```

Næste del af udregningen starter med, at der undersøges om den nyudregnede længde sammenlagt den totale udregnede længde, er længere end det stykke, bussen skal køre.

Hvis det ikke er, inkrementeres den total udregnede længde med den nyudregnede. Hvis det derimod er, skal bussens nye position være mellem det nuværende rutepunkt og det næste. Variablen missingLength indeholder afstanden fra det nuværende punkt til bussens nye position og er findes som længden bussen skal køre, minus det stykke der allerede er udregnet. Herefter undersøges der, hvilket punkt der skal bruges som initial punktet for udregningen af ny position. Denne udregning kan der læses mere om i afsnittet *8.2.5 - Anvendt matematik*) under "Ny position mellem to punkter"Hvis bussens position tidligere har været sat, og hvis bussens nye position er efter det næste rutepunkt, udregnes bussens nye position udfra bussens forrige. Hvis ikke udregnes den nye position udfra det forrige rutepunkt, relativt til bussens position. Når et nyt rutepunkt nås, startes et logging-event, som sender en ny loggingbesked til GUI'en . Desuden inkrementeres indexCounteren, så det næste rutepunkt er det gældende. Når en opdatering er fuldført, sover tråden i det antal sekunder der er sat i opdaterings hastigheden.

Kodeudsnit 25: Valg af ny rute ved endestation.

```
1 if (routes.Count == 1)
2 {
3   initialRoute.TurnAround();
4   indexCounter = 0;
5 }
6 else
7 {
8   List<BusRoute> possibleRoutes;
9   string atStop = initialRoute.stops[initialRoute.stops.Count - ↩
        1];
10  possibleRoutes = routes.FindAll(
11    R => (R.stops[R.stops.Count - 1] == atStop) ||
12    R.stops[0] == atStop);
13  if (possibleRoutes.Count == 1)
14  {
15    initialRoute.TurnAround();
16  }
17
18  else
19  {
20    possibleRoutes.Remove(initialRoute);
21    if (possibleRoutes.Count == 1)
22      initialRoute = possibleRoutes[0];
```

```
23      else
24        initialRoute = possibleRoutes[SimulationConfig.rand.Next(
25          0, possibleRoutes.Count)];
26      if (initialRoute.stops[0] != oldRouteStop)
27      {
28        initialRoute.TurnAround();
29      }
30
31    }
32    indexCounter = 0;
33 }
34 UpdateBusDB();
```

Når en bus vender, vil der ske en ruteændring. Hvis bussen kører på en simpel rute, vil der ikke ske andet, end at listen af punkter i ruten vil blive vendt, således at det punkt bussen holder ved, vil være det første punkt på ruten.
Hvis ruten derimod er kompleks, skal det undersøges, hvilken subrute, bussen nu skal køre på. Dette gøres ved først at udtage alle subruter på ruten, hvis første eller sidste stoppested, er svarende til den endestation bussen er ved. Hvis der kun kan findes én rute, må denne være den nuværende, og ruten vendes bare. Hvis ikke, fjernes den nuværende rute fra listen af mulige ruter og der undsøges om der nu kun er en rute listen, eller om der stadig er flere. Hvis der stadig er flere, vælges en rute tilfældigt, men hvis ikke, vælges den rute der er tilbage. Herefter undersøges der, om den valgte rutes første stoppested er det samme som den endestation som bussen er ved. Hvis ikke vendes den nye rute. Til sidst sætter indexCounteren til nul, og bussen bliver opdateret på databasen, med en ny retning og rute. Dette sker igennem DatabaseAcces klassen.

8.2.5 Komponent 5: Anvendt matematik

Denne komponent beskriver ikke opbygningen af systemet, men derimod de matematiske formler, der er blevet anvendt for at nå frem til resultaterne. Hver formel defineres og beskrives, samt i hvilken sammenhæng de bruges

Haversine

Denne formel bruges som en hjælpefunktion i forbindelse med udregninger på ruter. Formlen bruger to punkters geografiske position, og udregner fugleflugts afstanden mellem dem i meter, selv hvis punkterne er langt nok fra hinanden til, at jordens krumning spiller en rolle. Jordens krumning spiller ikke en rolle i dette system, da punkterne vil være relativt tæt på hinanden. Følgende variabler og konstanter tages i brug i denne formel:

d:	Fugleflugts afstand mellem to punkter i meter.
R:	Jordens gennemsnits radius. Konstant sat til 6371 kilometer.
θ_1, θ_2:	Længdegrad for punkt 1 og punkt 2
λ_1, λ_2:	Breddegrad for punkt 1 og punkt 2
a, c	Subresultater

Igennem formel (1) til (3) kan det ses, hvordan Haversine udregningerne foretages.

$$a = sin^2(\frac{\Delta\theta}{2}) + cos(\theta_1) * cos(\theta_2) * sin^2(\frac{\Delta\lambda}{2}) \quad (1)$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{1-a}) \quad (2)$$

$$d = R * c * 1000 \quad (3)$$

atan2 returner et grad værdi mellem -180°til 180°. Det er dog ikke nødvendigt at tage højde for dette i denne sammenhæng, da graderne ikke skal konvertes til helcirkel fra 0°til 360°.

Hjemmesiden, simulatoren og MySQL databasen tager alle brug af Haversine formlen.

- På hjemmesiden bruges formlen når det skal udregnes, hvilke to rutepunkter et stoppested skal ligge mellem. I denne situation skal det undersøges om stoppestedet ligger en vis afstand væk fra et punkt. Her til bruges Haversine til at udregne

distancen.

- I simulatoren bruges formlen når bussens næste position skal udregnes. I denne situation skal afstanden mellem to rutepunkter udregnes.
- I databasen bruges den i udregningen for ankomsttiden for en bus ved et stoppested. Et eksempl på dette kunne være, når afstanden fra bussen til det valgte stoppested skal udregnes. Her vil afstanden mellem hvert rutepunkt udregnes ved hjælp af denne formel, og ligges sammen. Resultatet vil være den søgte afstand.

Formlen og implementeringen i koden er ikke lavet selv, men derimod hentet fra http://www.movable-type.co.uk/scripts/latlong.html. Udregninger og implementeringer fra denne side er Open-Source og lavet af Chris Veness. Den eneste modifikation der er blevet implementeret er, at det endelige resultat er konverteret fra kilometer til meter

Kurs

Hvis et objekt bevæger sig mod et punkt, bruges denne formel til at udregne, hvilken retning objektet bevæger sig, på en 360°-skala. I denne udregning er nord sat til 0°/360°. Der blevet gjort brug af følgende variable:

b:	Kursen objektet følger på en 360°-skala.
θ_o, θ_p:	Længdegrad for objektet og punktet
λ_o, λ_p:	Breddegrad for objektet og punktet
x, y	Subresultater

Igennem formel (4) til (6) kan det ses, hvordan kursen udregnes.

$$x = cos(\theta_o) * sin(\theta_p) - sin(\theta_o) * cos(\theta_p) * cos(\Delta\lambda) \tag{4}$$

$$y = sin(\Delta\lambda) * cos(\theta_p) \tag{5}$$

$$b = atan2(y, x) \tag{6}$$

Da atan2 returner en værdi mellem -180°til 180°, er det nødvendigt at konvertere denne værdi til en 360°. Dette gøres ved hjælp af formel (7), hvor "%"er modulo.

$$b_{360} = ((b + 360)\%360) \tag{7}$$

Kun simulatoren gør brug af kursen. Den tages i brug, når der skal udregnes, hvor bussens nye position skal være. Det er tidligere udregnet, mellem hvilke to punkter bussen skal ligge, og der skal derfor udregnes et ny position mellem disse to punkter. Til denne udregning bruges kursen.

Formlen og implementeringen i koden er ikke lavet selv, men derimod hentet fra `http://www.movable-type.co.uk/scripts/latlong.html`. Udregninger og implementeringer fra denne side er Open-Source og lavet af Chris Veness.

Ny position mellem to punkter

Hvis et objekt skal placeres mellem to punkter, en vis distance udfra det første punkt, med en vis kurs mod det andet punkt, tages denne formel i brug. Kursen mellem de to punkter er givet på en 360°skala, hvor nord er sat til 0°/360°. Igennem udregnningen er det blevet gjort brug af følgende variabler og konstanter:

R:	Jordens gennemsnits radius. Konstant sat til 6371 kilometer.
b:	Kursen objektet følger fra initial punktet på en 360°-skala.
d:	Distancen objektet skal bevæge sig ud fra initial punktet.
θ_o, θ_p:	Længdegrad for objektet og punktet
λ_o, λ_p:	Breddegrad for objektet og punktet
x, y	Subresultater

Igennem formel (8) til (11) kan det ses, hvordan punktet udregnes.

$$\theta_o = asin(sin(\theta_p) * cos(\frac{d}{R * 1000}) + cos(\theta_p) * sin(\frac{d}{R * 1000}) * cos(b) \quad (8)$$

$$x = sin(b) * sin(\frac{d}{R * 1000}) * cos(\theta_p) \quad (9)$$

$$y = cos(\frac{d}{R * 1000}) - sin(\theta_p) * sin(\theta_o) \quad (10)$$

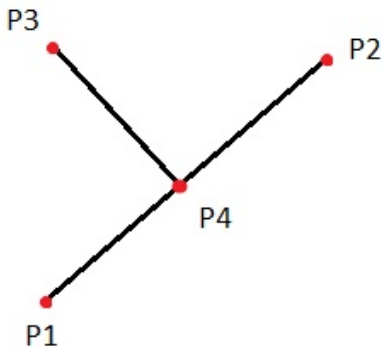
$$\lambda_o = \lambda_p + atan2(x, y) \quad (11)$$

Kun simulatoren tager brug af denne funktion. Den skal bruges i sammenhæng med at udregne, hvor en bus skal placeres på ruten ved en ny opdatering. Distancen bussen skal bevæge sig er tidligere blevet udregnet, samt det punkt på ruten, bussen skal være før. Kursen findes ved hjælp af bussens nuværende position samt det udregnede rutepunkt. Ved hjælp af startpunktet, distancen og kursen, udregnes bussens nye position.
Ved udregninger af breddegraden af objektet bruges atan2, og da der skal returneres et antal grader, skal denne værdi konvertes til en 360°skala. Dette gøres ved hjælp af formel (7)

Formlen og implementeringen i koden er ikke lavet selv, men derimod hentet fra http://www.movable-type.co.uk/scripts/latlong.html. Udregninger og implementeringer fra denne side er Open-Source og lavet af Chris Veness.

Tætteste punkt på en linje

Et linjestykke er spændt op mellem to punkter. Et tredje punkt kan ligge et vilkårligt stykke ud fra denne linje. Det tredje punkts tætteste punkt på linjen, vil være det punkt, hvis linjestykke skabt med det tredje punkt, er ortogonal med linjestykket mellem det første og andet punkt. Situationen kan ses på figur 27, hvor P1 og P2 er de punkter der spænder det originale linjestykket, P3 er det vilkårlige punkt, og P4 er det vilkårlige punkts tætteste punkt på linjen mellem P1 og P2. Udregningen er kun relevant, hvis linjestykket mellem P1 og P2 ikke er horizontal eller vertikal. Disse situationer forklares senere.
Igennem udregningerne gøres der brug af disse variabler:



Figur 27: Situationen 1: Tætteste punkt på en linje

A:	hældningskoefficient for Linjestykket mellem P1 og P2 .
B:	Skæring med y-aksen for injestykket mellem P1 og P2
$\theta_1, \theta_2, \theta_4, \theta_4$:	Længdegrad for de fire punkter.
$\lambda_1, \lambda_2, \lambda_3, \lambda_4$:	Breddegrad for de fire punkter.

Denne formel virker kun for relative tætte punkter, hvor der ikke skal tages hensyn til jordens hældning, og jorden derfor kan ses som et plan. Igennem formel (12) til (15), kan det ses, hvordan punktet findes.

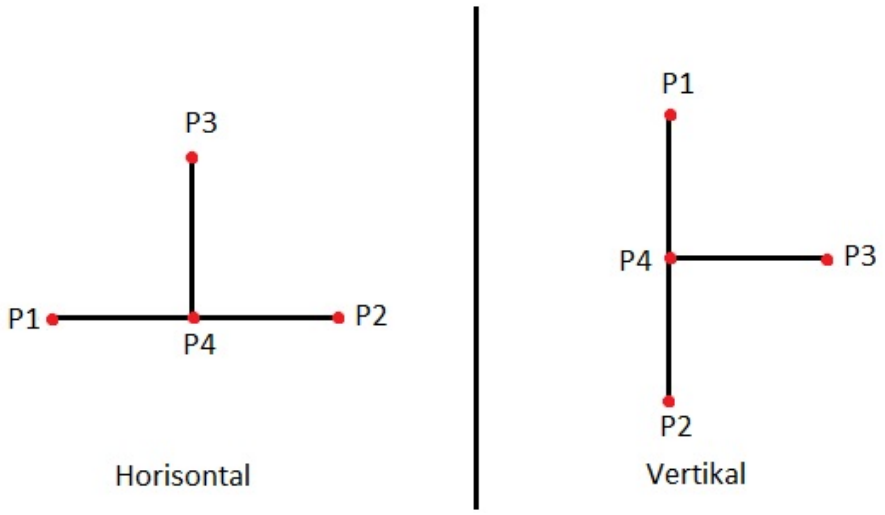
$$A = \frac{\theta_2 - \theta_1}{\lambda_2 - \lambda_1} \tag{12}$$

$$B = \theta_1 + A * (-\lambda_1) \tag{13}$$

$$\theta_4 = \frac{A * \theta_3 + \lambda_3 - A * B}{A^2 + 1} \tag{14}$$

$$\lambda_4 = \frac{A^2 * \theta_3 + A * \lambda_3 + B}{A^2 + 1} \tag{15}$$

I tilfælde af at linjen mellem punkt 1 og punkt 2 er horisontal eller vertikal, er der under implementering skabt special tilfælde. Hvis linjen er horisontal sættes $\theta_4 = \theta_1$ og $\lambda_4 = \lambda_3$. Hvis linjen er vertikal sættes $\theta_4 = \theta_3$ og $\lambda_4 = \lambda_1$. Situation ses på figur 28



Figur 28: Situationen 2 og 3: Horisontalt eller vertikalt linjestykke

Denne formel tages i brug i databasen og på hjemmesiden.

- I databasen tages denne udregning i brug når ankomsttiden for bus ved et stoppested skal udregnes. I denne process skal der på et tidspunkt udregnes, hvilket rutepunkt, bussen er tættest på. Dette gøres ved hjælp af en kombination af denne formel, samt Haversine formlen. Ved hvert linjestykke skabt af punkterne på ruten, udregnes det tætteste punkt for bussen på dette linje stykke. Mellem bussen og dette punkt gøres der brug af Haversine, for at udregne afstanden fra bussen til linjestykket. Endepunktet for det linjestykke, hvor bus til linje afstanden er kortest, må være det rutepunkt bussen er tættest på.

- På hjemmesiden skal der udregnes, mellem hvilke to punkter et stoppested skal ligge. Udregningen fortages på sammen måde, som i databasen, hvor bussen blot er erstattet med et stoppested. Resultatet er det rutepunkt, stoppested skal ligge før.

Den linære funktion (Ax + B) der skabes til linjestykket vil ikke kun strække sig mellem de to længde- og breddegrader der gives. Der kan derfor opstå en situation, hvor objektet egentlig ligger tættest på ét linjestykke, men den ortogonale linje der bliver skabt fra objetet til et andet linjestykke vil have en mindre distance. Derfor er det vigtigt, at der ved implementering tages højde for, at det kun er linjestykket der undersøges og ikke hele linjen. Dette sikres ved, at en eller flere af de følgende fire regler ikke må være gældende for $\theta_4 og \lambda_4$:

- $\theta_4 > \theta_1 \quad \& \quad \theta_4 > \theta_2$
- $\theta_4 < \theta_1 \quad \& \quad \theta_4 < \theta_2$
- $\lambda_4 > \lambda_1 \quad \& \quad \lambda_4 > \lambda_2$
- $\lambda_4 < \lambda_1 \quad \& \quad \lambda_4 < \lambda_2$

Hvis blot en af disse regler passer, er punktet ugyldigt, da det ikke ligger på linjestykket. Under udregningen af stoppestedernes position på ruten, er der dog tilføjet en ekstra undersøgelse. Hvis distancen mellem stoppestedet er mindre en 25 meter, gyldigøres udregninger selvom en eller flere af de overstående punkter er gældende. Dette gøres, da et stoppested kan ligge tilpas forskudt fra ruten til, at den ikke vil passe på noget linjestykke. Denne formel er lavet på baggrund af information fundet på to hjemmesider.
http://demo.activemath.org/ActiveMath2/search/show.cmd?id=mbase://AC_UK_calculus/functions/ex_linear_equation_two_points beskriver hvordan den lineære funktion findes givet ved to punkter.
http://math.ucsd.edu/~wgarner/math4c/derivations/distance/distptline.htm beskriver hvordan det ortogonale punkt findes ved hjælp af en lineær funktion.

Grader og radianer konvertering

Når de forrige fire formler skal implementeres er det ofte nødvendigt at konvertere mellem radianer og grader. På formel (16) kan konverteringen fra grader til radianer ses, og på formel (17) kan konverteringen fra radianer til grader ses.

$$Radianer = \frac{Grader * \pi}{180} \tag{16}$$

$$Grader = \frac{Radianer * 180}{\pi} \tag{17}$$

9 DATA VIEW

9.1 Data model

En kritisk del af dette system er data storage og data retrieval. Dette er blevet implementeret i form af to relationelle databaser; en distribueret og en lokal.
Til den distribuerede database er server blevet lejet hos UnoEuro[23]. Herunder er databasen oprettet som en MySQL database på serveren http://mysql23.unoeuro.com
Den lokale database eksisterer, fordi brugeren skal kunne gemme busruter lokalt på sin telefon. Dette er blevet implementeret i form af en SQLite database.
Diagrammer kan findes i fuld størrelse i bilag under Diagrammer/Database Diagrammer, og fuld implementering kan findes under Kode/Database.

9.1.1 Design af MySQL database

Den distribuerede database gemmer alt information vedrørende busserne og deres ruter. Opbygningen af databasen kan ses som tre komponenter der interagerer; Busser, busruter og stoppesteder.
Samtlige komponenter er defineret ved positions data i form af punkter. Disse punkter er længde- og breddegrader og kan ses som den fysiske position af den komponent, de relaterer til. Disse falder derfor i tre katagorier; Busposition, rutepunkter med stoppesteder og waypoints.

- Busposition er defineret som den fysiske placering af en given bus. Da der under systemets udvikling ikke var adgang til nogen fysiske busser, blev denne katagori af positions simuleret. Simulatoren kan dog skiftes ud med en virkelig bus, hvis positioner for denne kan stilles til rådighed.

- Rutepunter og stoppesteder indeholder positionsdata, som bruges til at tegne- eller laver udregninger på ruten. Disse udregninger er defineret senere under afsnittene *9.1.3: Stored Procedures og 9.1.4: Functions*

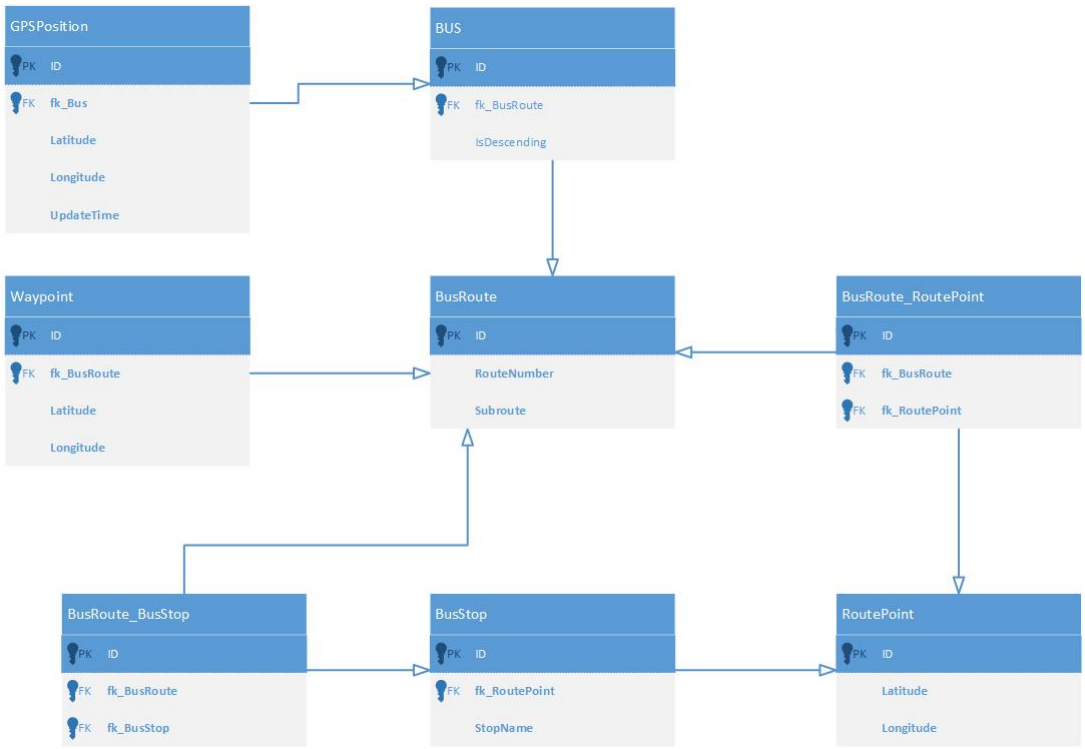
- Waypoints bruges som genskabelses-punkter til en given rute. Disse punkter bliver udelukkende brugt af administrationsværktøjet, til at genskabe den rute de beskri-

[23] www.unoeuro.com

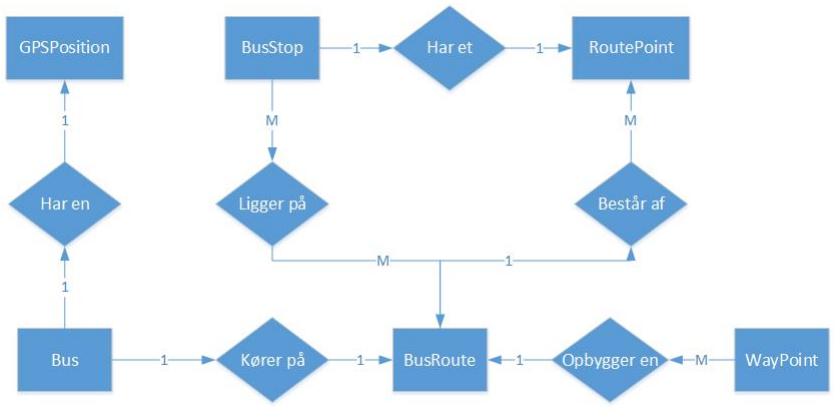
ver.

Hele systemet er opbygget omkring oprettelse, fjernelse og manipulation af positions data. Dette er klart afspejlet i databasen i form af, hvor meget dette data bliver brugt.
Tidligt i udviklingsprocessen blev det fastsat at positions data skulle have en præcision på seks decimaler, da dette ville resultere i en positions afvigelse på under en meter. Systemet virker stadig med en lavere præcision, men dette vil resultere i en større positionsafvigelse.

Databasen er bygget op af følgende tabeller: Bus, BusRoute, BusRoute_RoutePoint, BusRoute_BusStop, BusStop, GPSPosition, RoutePoint og Waypoint.
På figur 29 vises opbygningen af tabellerne som et UML OO diagram, og på figur 30 kan relationerne i databasen ses som et ER diagram.



Figur 29: UML OO diagram over den distribuerede MySQL database



Figur 30: ER Diagram over den distribuerede MySql database

Herunder følger en forklaring af tabellerne og deres rolle i systemet.

- **Bus**
  - Indeholder alt relevant data vedrørende kørende busser. fk_BusRoute er en foreign key til BusRoute tabellen og definerer hvilken rute bussen kører på. IsDescending er et simpelt flag, som bestemmer i hvilken retning bussen kører. Hvis IsDescending er true, betyder det at bussen kører fra sidste til første punkt defineret ved ID i BusRoute_RoutePoint, og omvendt hvis den er false.
    Som den eneste tabel er der mulighed for, at nulls kan fremkomme. Dette vil ske i situationer hvor bussen eksisterer i systemet, men endnu ikke er sat på en rute. Tabellens primary key er sat til at være det ID som defineres ved busses oprettelse. Dette nummer vil også stå på den fysiske bus.

- **BusRoute**
  - Indeholder detaljer omkring busruten foruden dens rutepunkter. BusNumber er ikke nødvendigvis en unik værdi, da en kompleks rute er bygget op af to eller flere underruter. Derfor bliver tabelens primary key sat til et autogeneret ID, som bliver inkrementeret ved nyt indlæg i BusRoute. BusNumber er rutenummeret, og også det nummer som vil kunne ses på bussens front. Nummeret er givet ved en varchar på ti karakterer, da ruter også kan have bogstaver i deres nummer.
    Hvis SubRoute er sat til nul, vil ruten kun bestå af det enkelte ID, men hvis ruten er kompleks vil SubRoute starte fra et, og inkrementere med en for hver

delrute på den givne rute. Ruter er i denne sammenhæng defineret som vejen mellem to endestationer, og hvis en rute har mere end to endestationer, vil den have minimum to hele ruter sat på det givne rutenummer.

- **BusRoute_RoutePoint**
  - Indeholder den egentlige rute for det givne rutenummer. Primary keyen er ID'et i denne tabel og autogenereret, men bruges til at definere rækkefølgen på punkterne, som ruten bliver opbygget af. fk_BusRoute er foreign key til ID'et for busruten, og fk_RoutePoint er foreign key til ID'et for rutepunktet på et givet sted på ruten. Det første og sidste punkt for den givne rute vil altid være de to endestationer på ruten.
    Rutepunkterne for stoppestedet bliver tilføjet til ruten ved hjælp af en funktion beskrevet i afsnittet *8.2.3: Komponent 3: Administrations hjemmeside.*
- **BusRoute_BusStop**
  - Indeholder stoppestedsplanen for det givne rutenummer. ID'et i denne tabel er autogeneret, men bruges til at definere rækkefølgen af stoppestederne på den givne rute. fk_BusRoute refererer til den busrute stoppestedet er på, og fk_BusStop refererer til selve stoppestedet. Det første og sidste ID for den givne busrute, vil være de to endestationer på den givne rute.
- **BusStop**
  - Indeholder alle stoppesteder i systemet. Primary keyen er ID'et i denne tabel og er autogeneret. StopName er navnet på det givne stoppested, og er en varchar på 100 karakterer.
    fk_RoutePoint er en foreign key til ID'et i RoutePoint tabellen, og vil være det fysiske punkt for stoppestedet givet ved en længde- og breddegrad.
- **RoutePoint**
  - Indeholder alle punkter for alle ruter og stoppesteder. Primary keyen er sat til at være et autogeneret ID. Hvert indlæg i denne tabel vil definere en geografisk

position. Longitude og latitude er i denne sammenhæng længde- og breddegraden, og de er defineret som et decimal med 15 decimaler. Alle 15 decimaler er ikke nødvendigs i brug og ved en indsættelse af et tal på f.eks. seks decimaler, vil de sidste ni være sat til nul.

- **GPSPosition**
  - Indeholder alle kørende bussers nuværende og tidligere positioner. Primary keyen er sat til et ID, som bruges til at definere rækkefølgen på indlægene, således det højeste ID for en given bus vil være den nyeste position. Longitude og Latitude er længde- og breddegraden for den givne bus. Både Longitude og Latitude er givet ved 15 decimaler, dog hvor alle 15 ikke nødvendigvis er i brug. Ved en indsættelse af et tal på f.eks. seks decimaler, vil de sidste ni være sat til nul. UpdateTime er et timestamp for positionen og bruges til, at udregne hvor lang tid bussen har kørt. Dette er beskrevet nærmere i afsnittene *9.1.3: Stored Procedures og 9.1.4: Functions*. fk_Bus er en foreign key til tabellen Bus og bruges til at definere hvilken bus der har lavet opdateringen.

- **Waypoint**
  - Indeholder alle punkter der er nødvendige for genskabelse af en rute på administrations siden. Primary keyen er ID'et og autogeneret. Denne bruges ikke til andet end at unikt markere punktet.
    Longitude og Latitude er længde- og breddegraden for det givne punkt. Både Longitude og Latitude er givet ved 15 decimaler, dog hvor alle 15 ikke nødvendigvis er i brug. Ved en indsættelse af et tal på f.eks. seks decimaler, vil de sidste ni være sat til nul. fk_BusRoute er en foreign key til BusRoute tabellen, og definerer således, hvilken busrute det givne waypoint er relateret til.

Normalform

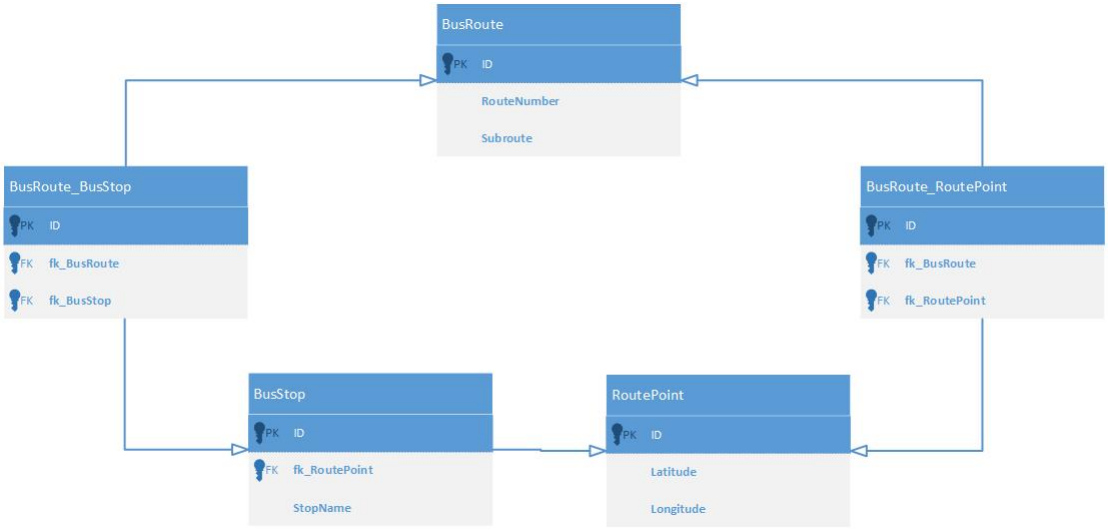
Databasen er normaliseret til tredje normalform, hvor nulls er tilladt i enkelte tilfælde da det sås som gavnligt. Tabellen Bus indeholder alle oprettede busser, men det er ikke et krav, at en bus er på en rute. I tilfælde af en bus uden rute, vil fk_BusRoute og IsDescending være null.

Det antages at tredje normalform er tilstrækkeligt for systemet.
Begrundelsen for, at databasen er på tredjenormalform er:

- Ingen elementer er i sig selv elementer. Dvs. ingen kolonner gentager sig selv.
- Ingen primary keys er composite keys, og derfor er intet ikke-nøgle element afhængig af kun en del af nøglen
- Ingen elementer er afhængige af et ikke-nøgle element. Dvs. ingen kolonner i én tabel, definerer andre kolonner i samme tabel.

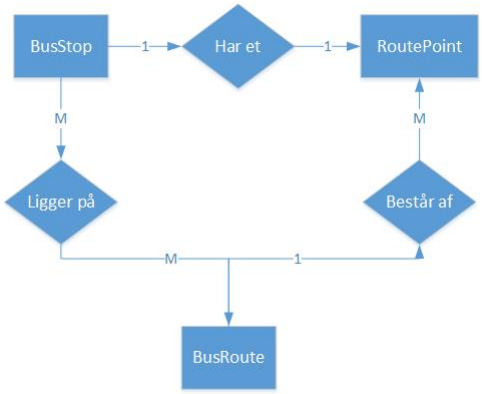
9.1.2 Design af SQLite databasen

Mobil applikationen har en favoriserings funktion der bruges til at persistere brugervalgte ruter lokalt. Dette er gjort så brugeren hurtigt kan indlæse de ruter som bruges mest. Ruterne persisteres lokalt som et udsnit af den distribuerede database.
På figur 31 kan man se et UML OO diagram over den lokale SQLite database og på figur 32 kan man se et ER diagram over samme database. Da den lokale database blot er



Figur 31: UML OO diagram over den lokale SQLite database

et udsnit af den distribuerede MySQL database, henvises der til tabel beskrivelserne for MySQL tabellerne i forrige afsnit. Databasen er derfor også på tredje normalform, som MySQL databasen.
Den eneste forskel fra MySQL databasen er, at denne tabel gør brug af Delete Cascades. Dette vil sige, at sletningen af data fra SQLite databasen kun kræver at man sletter fra



Figur 32: ER diagram over den lokale SQLite database

BusRoute og RoutePoint tabellerne, da disse har foreign keys i de andre tabeller. Da flere ruter med de samme stoppesteder godt kan indskrives er det blevet vedtaget, at stoppestederne ikke slettes, når en rute ufavoriseres. Dette betyder at stoppestederne kan genbruges ved nye favoriseringer.

9.1.3 Stored Procedures

Der eksisterer kun Stored Procedures på MySQL database siden, og derfor vil dette afsnit kun omhandle disse.
Der er blevet lavet tre Stored Procedures i sammenhæng med tidsudregning for tætteste bus til valgt stoppested. Disse tre vil blive beskrevet herunder, udfra kodeudsnit. I kodeudsnittet vil ingen kommentarer være tilstede.
I kodeudsnittene fremkommer forkortelserne "Asc"og "Desc. Dette står for Ascending og Descending og er en beskrivelse af, hvordan ruten indlæses. Ascending betyder at busruten indlæses fra første til sidste punkt i BusRoute_RoutePoint og BusRoute_BusStop tabellerne og Descending betyder at den indlæses fra sidste til første punkt.
Temporary tabeller bliver brugt meget i funktionerne og procedurene. De beskriver en fuldt funktionel tabel, med den forskel, at de kun er synlige fra den givne forbindelse. Når der i proceduren kun laves indskrivninger i temporary tabeller, gør det tilgangen trådsikker. Dette betyder at proceduren godt kan tilgås fra flere enheder på samme tid. Når forbindelsen lukkes, vil de oprettede temporary tabeller slettes.

CalcBusToStopTime

Denne Stored Procedure er kernen i tidsudregningen. Den samler alle værdierne og sender dem videre i de forskellige funktioner. På kodeudsnit 26 ses et udsnit af proceduren. I den fulde procedure, vil udregningerne for begge retninger mod stoppestedet foregå, men da dette blot er en duplikering af samme kode, med forskellige variabler og funktionsnavne, vises dette ikke. Alle deklareringer af variabler er også fjernet.

Kodeudsnit 26: CalcBusToStopTime. Finder nærmeste bus og udregner tiden begge veje

```
1 create procedure CalcBusToStopTime(
2 IN stopName varchar(100), IN routeNumber varchar(10),
3 OUT TimeToStopSecAsc int, OUT TimeToStopSecDesc int,
4 OUT busIDAsc int, out busIDDesc int,
5 OUT EndBusStopAsc varchar(100), OUT EndBusStopDesc varchar(100))
6
7 BEGIN
8 drop temporary table if exists possibleRoutes;
9 create temporary table possibleRoutes(
10   possRouteID int,
11   possRouteStopID int
12 );
13
14 insert into possibleRoutes
15 select distinct BusRoute.ID, BusRoute_RoutePoint.ID from BusRoute
16 inner join BusRoute_BusStop on BusRoute.ID = BusRoute_BusStop.↩
     fk_BusRoute
17 inner join BusStop on BusRoute_BusStop.fk_BusStop = BusStop.ID
18 inner join BusRoute_RoutePoint on BusRoute.ID = ↩
     BusRoute_RoutePoint.fk_BusRoute
19    and BusStop.fk_RoutePoint = BusRoute_RoutePoint.fk_RoutePoint
20 where BusRoute.RouteNumber = routeNumber and BusStop.StopName = ↩
     stopName ;
21
22 call GetClosestBusAscProc(@ClosestEndEPIdAsc, @ClosestBDIstAsc, ↩
     @ClosestBIDAsc );
23 select @ClosestsEndPointIDAsc, @ClosestBDIstAsc, @ClosestBIDAsc
24 into ClosestEndPointIdAsc,ClosestBusDistanceAsc,ClosestBusIdAsc;
25
26 select CalcBusAvgSpeedAsc(ClosestBusIdAsc) into ↩
     ClosestBusSpeedAsc;
27
28 set TimeToStopSecAsc = ClosestBusDistanceAsc/ClosestBusSpeedAsc;
29 set busIDAsc = ClosestBusIdAsc;
30
```

```
31 select BusStop.StopName from BusStop
32 inner join BusRoute_BusStop on BusRoute_BusStop.fk_BusStop = ↩
     BusStop.ID
33 inner join Bus on BusRoute_BusStop.fk_BusRoute = Bus.fk_BusRoute
34 where Bus.ID = ClosestBusIdAsc Order by BusRoute_BusStop.ID desc ↩
     limit 1 into EndBusStopAsc;
35
36 drop temporary table possibleRoutes;
37
38 END$$
```

Proceduren modtager navnet på det valgte stoppested, samt det valgte rutenummer. Ved fuldent forløb vil den returnere tiden for den nærmeste bus til det valgte stop, den nærmeste bus samt endestationen for den nærmeste bus. Alt returneres parvist i form af begge retninger mod stoppestedet.

Først findes samtlige mulige ruter udfra det medsendte stoppestedsnavn og rutenummer. Disse ruter indsættes i en temporary tabel. Dette er nødvendigt i tilfælde af komplekse ruter, hvor mere end en rute kan have samme stoppested og rutenummer. "GetClosestBusAscProc", som beskrives senere i dette afsnit. Denne procedure returnerer tætteste rutepunkt, ID'et for den tætteste bus, samt afstanden fra den nærmeste bus til stoppestedet. Herefter udregnes bussens gennemsnitshastighed ved kaldet til "CalcBusAvgSpeedAsc", som bruger det fundne bus ID. Denne funktion beskrives dybere under afsnittet "Functions".

Tiden fra bussen til stoppestedet findes ved at dividere distancen med gennemsnitshastigheden (Meter / Meter/Sekund = Sekund).

Til sidst findes endestationen, og returneres sammen med tiden og bus ID'et.

GetClosestBusAscProc og GetClosestBusDescProc

Da proceduren for begge retninger er meget ens, vil der kun vises et kodeudsnit for "GetClosestBusAscProc". Denne kan ses på kodeudsnit 27.

Alle deklareringer er fjernet for at give et bedre overblik over funktionalitetet af proceduren. En detaljeret forklaring, samt forskellene mellem "GetClosestBusAscProc"og "GetClosestBusDescProc", følger efter kodeudsnittet.

Kodeudsnit 27: GetClosestBusAscProc. Udregner nærmeste bus- samt distance til stop og nærmeste rutepunkt

```
1 create procedure GetClosestBusAscProc(OUT busClosestEndPointAsc ↩
    int, Out routeLengthAsc float, OUT closestBusId int)
2 begin
3
4 drop temporary table if exists BussesOnRouteAsc;
5 create temporary table BussesOnRouteAsc(
6   autoId int auto_increment primary key,
7   busId int,
8   stopID int
9 );
10
11 insert into BussesOnRouteAsc (busId, stopID) select distinct Bus.↩
    ID, possibleRoutes.possRouteStopID from Bus
12 inner join possibleRoutes on Bus.fk_BusRoute = possibleRoutes.↩
    possRouteID
13 where Bus.IsDescending=false;
14
15 select count(busId) from BussesOnRouteAsc into NumberOfBusses;
16
17 while BusCounter <= NumberOfBusses do
18   select busId,stopID from BussesOnRouteAsc where autoId = ↩
       BusCounter into currentBusId,currentStopId;
19
20   select GetClosestEndpointAsc(currentBusId)
21     into closestEndPoint;
22
23   if(closestEndPoint <= currentStopId) then
24     select GPSPosition.Latitude, GPSPosition.Longitude from ↩
         GPSPosition where GPSPosition.fk_Bus = currentBusId
25     order by GPSPosition.ID desc limit 1 into busPos_lat, ↩
         busPos_lon;
26
27     select CalcRouteLengthAsc(busPos_lon, busPos_lat, ↩
         closestEndPoint, currentStopId) into currentBusDist;
28   else
29     set currentBusDist = 10000000;
30   end if;
31   if (currentBusDist < leastBusDist) then
32     set leastBusDist = currentBusDist;
33     set closestbID = currentBusId;
34     set closestEP = closestEndPoint;
35   end if;
36   set BusCounter = BusCounter + 1;
```

```
37 end while;
38 set busClosestEndPointAsc = closestEP;
39 set routeLengthAsc = leastBusDist;
40 set closestBusId = closestbID;
41
42 drop temporary table BussesOnRouteAsc;
43 END $$
```

Denne procedure modtager ingen parametre, da den kun bruger data sat i "possibleRoutes"tabellen fundet i "CalcBusToStopTime"proceduren. Hovedfunktionaliteten i denne procedure er, at at udregne hvilken bus, der er tættest på det valgte stoppested. Dette repræsenteres ved bussens ID. Igennem denne udregning findes der også to underresultater der skal bruges i senere udregninger; Distancen fra bussen hen til stoppestedet, samt det tætteste rutepunkt bussen endnu ikke har nået.

Alle buser, som kører på en af de ruter i "possibleRoutes"og hvor "IsDescending"er sat til false udtages. Disse busser bliver parret med det ID stoppestedet har, i "BusRoute_RoutePoint"tabellen og et auto-inkrementeret ID startende fra 1. Disse værdier bliver lagt ind i "BussesOnRouteAsc"temporary tabellen. Herefter findes det antal af busser, der er blevet udtaget, og dette tal bruges til den øvre grænse for while-løkkens. Den nedre grænse er blot en counter som sættes til 1 ved initiering.

While-løkkens rolle er at iterere igennem samtlige busser, og udregne distancen fra hver bus til dens parrede stoppested, hvorefter der skal vælges den bus, der har den korteste distance til sit stoppested.

Først udregnes Det nærmeste rutepunkt ved et kald til funktionen "GetClosestEndpointAsc". Hvis dette rutepunkt har et større ID end busstoppets, vil distancen fra bussen til stoppestedet sættes til 10000000, altså meget højt. I en fysisk forstand vil dette ske, hvis bussen er kørt forbi det givne stoppested, og derfor ikke længere kan være den nærmeste bus til stoppestedet. Hvis rutepunktet derimod har et mindre ID end stoppestedet, vil de nyeste koordinater for bussen findes, og distancen fra bussen hen til stoppestedet vil udregnes ved et kald til funktionen "CalcRouteLengthAsc".

Herefter undersøges der, om den givne bus har en mindre distance hen til stoppestedet end den bus med den nuværende korteste distance. "leastBusDist"er sat til 100000, altså højt, men ikke lige så højt som det tal den nuværende distance sættes til, hvis bussen er

kørt forbi stoppestedet. Dette vil betyde at ingen sådan bus, ved en fejl, kan vælges som den tætteste bus. Hvis denne bus derimod har en mindre distance end den nuværende korteste distance, vil den mindste distance sættes til denne. ID'et for bussen samt det tætteste rutepunkt, vil også sættes i denne situation. Til sidst vil den korteste distance, det tætteste rutepunkt samt ID'et for den tætteste bus blive returneret.
I "GetClosestBusDescProc"(samme udregning, blot for rute der køre fra sidste til første stoppested), er der to definerende forskelle.
På kodeudsnit 28, kan den første forskel ses. I dette tilfælde hentes der kun busser ud hvor *IsDescending* er true, altså hvor den givne bus kører fra første til sidste stoppested.

Kodeudsnit 28: GetClosestBusDescProc forskel 1

```
1 ...
2 insert into BussesOnRouteDesc (busId,stopId) select distinct Bus.↩
    ID,      possibleRoutes.possRouteStopID from Bus
3 inner join possibleRoutes on Bus.fk_BusRoute = possibleRoutes.↩
    possRouteID
4 where Bus.IsDescending=true;
5 ...
```

På kodeudsnit 29, kan den anden forskel ses. Hvis en bus kører fra første til sidste stoppested, vil det nærmeste rutepunkt til bussen, have et større ID end stoppestedets, hvis bussen endnu ikke er kørt forbi. Derfor undersøges der her om rutepunktets ID er større eller ligmed stoppestedets ID, hvor der i "GetClosestBusAscProc"undersøges om det er mindre eller ligmed.

Kodeudsnit 29: GetClosestBusDescProc forskel 2

```
1 ...
2 if(closestEndPoint >= currentStopId) then
3 ...
```

9.1.4 Functions:

Igennem forløbet af "CalcBusToStopTime"proceduren, tages en del funktioner i brug. Disse bruges når kun en enkelt værdi behøves returneres. Funktionerne er delt om i to typer; Funktioner til udregning af relevant information til procedurene, samt matematik-funktioner. Der vil ikke vises kodeeksempler for matematik funktionerne i dette afsnit, men der henvises til afsnit *8.2.5: Komponent 5: Anvendt Matematik*, for beskrivelser af disse.

Som i afsnittet *9.1.3 Stored Procedures*, er funktionerne bygget op parvist; En funktion til busser der kører fra første til sidste stop (ascending), samt en anden til busser, der kører fra sidste til første stop (descending). Der vil kun vises et kodeudsnit af ascending-funktionerne, hvorefter forskellene i descending-funktionerne beskrives. Kodeudsnittene vil ikke indeholde kommentarer eller initialiseringer af variable, så et bedre overblik af funktionalitet kan gives.

GetClosestEndpointAsc og GetClosestEndpointDesc

Disse funktioner tages i brug i "GetClosestBusAscProc"og "GetClosestBusDescProc"procedurene, og bruges til at finde ID'et for det rutepunkt, en given bus er tættest på. Dette ID er dog ikke rutepunktets egentlige ID i "RoutePoint"tabellen, men derimod dens ID i "BusRoute_RoutePoint"tabellen. Denne bus er defineret ved dens ID, givet til funktionen som dens eneste parameter. På kodeudsnit 30 kan "GetClosestEndpointAsc"funktionen ses.

Kodeudsnit 30: GetClosestEndpointAsc finder det tætteste punkt på ruten fra bussen

```
1 create function GetClosestEndpointAsc(busID int)
2 returns int
3 begin
4 drop temporary table if exists ChosenRouteAsc;
5 create TEMPORARY table if not exists ChosenRouteAsc(
6   id int primary key,
7   bus_lat decimal(20,15),
8   bus_lon decimal(20,15)
9 );
10
11 insert into ChosenRouteAsc (id,bus_lat,bus_lon)
```

```
12 select BusRoute_RoutePoint.ID, RoutePoint.Latitude,  RoutePoint.↩
     Longitude from RoutePoint
13 inner join BusRoute_RoutePoint on BusRoute_RoutePoint.↩
     fk_RoutePoint = RoutePoint.ID
14 inner join Bus on Bus.fk_BusRoute = BusRoute_RoutePoint.↩
     fk_BusRoute
15 where Bus.ID = busID
16 order by(BusRoute_RoutePoint.ID) asc;
17
18 select ChosenRouteAsc.ID from ChosenRouteAsc order by id asc ↩
     limit 1 into RouteCounter;
19 select ChosenRouteAsc.ID from ChosenRouteAsc order by id desc ↩
     limit 1 into LastChosenID;
20
21 select GPSPosition.Latitude,  GPSPosition.Longitude from ↩
     GPSPosition where GPSPosition.fk_Bus = busID
22 order by GPSPosition.ID desc limit 1 into BusLastPosLat, ↩
     BusLastPosLon;
23
24 while RouteCounter < LastChosenID do
25   select bus_lon from ChosenRouteAsc where id = RouteCounter into↩
       R1x;
26   select bus_lat from ChosenRouteAsc where id = RouteCounter into↩
       R1y;
27   select bus_lon from ChosenRouteAsc where id = RouteCounter+1 ↩
      into R2x;
28   select bus_lat from ChosenRouteAsc where id = RouteCounter+1 ↩
      into R2y;
29   set BusDist = CalcRouteLineDist(BusLastPosLon, BusLastPosLat, ↩
      R1x, R1y, R2x, R2y);
30
31   if BusDist < PrevBusDist then
32     set PrevBusDist = BusDist;
33     set ClosestEndPointId = RouteCounter+1;
34   end if;
35   Set RouteCounter = RouteCounter + 1;
36 end while;
37 return ClosestEndPointId;
38 END$$
```

Ruten som den givne bus kører på hentes ud og gemmes i en temporary tabellen "ChosenRouteAsc". I denne tabel gemmes længde- og breddegrader, sammen det ID punktet har, i *BusRoute_ RoutePoint*-tabellen. Da samtlige punkter ligges ind i databasen samtidig, efter en rute er skabt på hjemmesiden, garanteres det, at punterne ligger sekvensielt. Punkterne gemmes altså i rækkefølge i *BusRoute_ RoutePoint*, uden spring i IDerne. Det-

te gør at punkterne kan itereres igennem, uden at der skal tages højde for spring, og kan sorteres efter ID i den rækkefølge der skal bruges (ascending for første til sidste stoppested, descending for sidste til første stoppested). Det er meget sandsynligt at det første ID hentet ikke er 1, Så det første og sidste punkt på ruten findes også, og bruges som den nedre og øvre grænse for while-løkken. På den måde vil der itereres igennem samtlige punkter på ruten, hvor ID'et for første og sidste stop ikke har nogen betydning for funktionen. Inden while-løkken startes, hentes bussens sidste position ud, så det ikke har nogen betydning, hvis bussens position ændrer sig under itereringen af ruten.
Så længe "routeCounter"(det nuværende ID der undersøges) er mindre end "LastChosenID"(Det sidste ID på ruten), udtages punkterne for det nuværende ID og det næste. Således laves der et linjestykke spændt ud mellem to punkter. Afstanden fra bussen til dens tætteste punkt på dette linjestyke, udregnes i "CalcRouteLineDist". Denne funktion er udelukkende matematisk og vil beskrives i *8.2.5: Komponent 5: Anvendt Matematik*. Hvis bussens position på et givent linjestykke ikke er gyldigt, vil 1000000, et stort tal, returneres. Dette tal vil være større end prevBusDist som har en initial værdi sat til 100000. Dette sørger for, at det givne endpoint ikke, ved en fejl, tælles med. Hvis den udregnede værdi af distancen til punktet på linjen, er mindre end den forrige distance, vil det næste punkt på ruten, i forhold til det punkt man undersøger, søttes til bussens tætteste. Ved en fuldent gennemiterering af ruten, vil bussens tætteste rutepunkt være fundet, og ID'et for dette punkt returneres.
I "GetClosestEndpointDesc"er der nogle enkelte forskelle, som her vil beskrives. På kodeudsnit 31, kan det ses hvordan ruten nu hentes ud i en tabel, hvor der sorteres efter ID'et i "BusRoute_RoutePoint"tabellen, i faldende rækkefølge.

Kodeudsnit 31: GetClosestEndpointDesc forskel 1

```
1 ...
2   insert into ChosenRouteDesc  (id,bus_lat,bus_lon)
3   select BusRoute_RoutePoint.ID, RoutePoint.Latitude,  RoutePoint↩
       .Longitude from RoutePoint
4   inner join BusRoute_RoutePoint on BusRoute_RoutePoint.↩
       fk_RoutePoint = RoutePoint.ID
5   inner join Bus on Bus.fk_BusRoute = BusRoute_RoutePoint.↩
       fk_BusRoute
```

```
6   where Bus.ID = busID
7   order by(BusRoute_RoutePoint.ID) desc;
8 ...
```

På kodeudsnit 32 ses det hvordan, der nu læses i modsat rækkefølge fra "ChosenRouteDesc"tabellen. "RouteCounter"er nu den øverste grænse, og "LastChosenID"er nu den nedre. Der læses nu også i omvendt rækkefølge fra tabellen, da ID'erne nu er faldende. Det vil også sige, at "RouteCounter"dekrementeres i stedet for inkrementeres i slutningen af hver iteration.

Kodeudsnit 32: GetClosestEndpointDesc forskel 2

```
1 ...
2 select ChosenRouteDesc.ID from ChosenRouteDesc order by id asc ↩
     limit 1 into LastChosenID;
3 select ChosenRouteDesc.ID from ChosenRouteDesc order by id desc ↩
     limit 1 into RouteCounter;
4 ...
5 while RouteCounter >  LastChosenID  do
6   select bus_lon from ChosenRouteDesc where id = RouteCounter ↩
       into R1x;
7   select bus_lat from ChosenRouteDesc where id = RouteCounter ↩
       into R1y;
8   select bus_lon from ChosenRouteDesc where id = RouteCounter-1 ↩
       into R2x;
9   select bus_lat from ChosenRouteDesc where id = RouteCounter-1 ↩
       into R2y;
10 ...
11 Set RouteCounter = RouteCounter - 1;
12 ...
```

CalcRouteLengthAsc og CalcRouteLengthDesc

Disse funktioner tages i brug i "GetClosestBusAscProc og "GetClosestBusDescProc"procedurene. De bruges til at udregne afstanden fra en bus til det valgte stoppested. Funktionerne modtager et koordinat-sæt for bussen, ID'et for bussens tætteste rutepunkt, samt ID'et på stoppestedet. Bemærk at disse ID'er er hentet fra "BusRoute_RoutePoint"tabellerne og symboliserer derfor rutepunktet og stoppestedets placering på ruten, og ikke deres egentlige ID'er i henholdsvis "RoutePoint"og "BusStop"tabellerne. Funktionerne er ikke meget forskellige, ud over hvilken ChosenRoute tabel fra forrige funktioner, der tages i brug.

Herudover itereres der også i omvendt rækkefølge. Der vises kodeudsnit et for "CalcRouteLengthAsc", hvorefter funktionen forklares i detajler. Til sidst forklares forskellene mellem "CalcRouteLengthAsc og "CalcRouteLengthDesc"mere detaljeret. På kodeudsnit 33 kan funktionen ses uden kommentarer eller initialiseringer uden værdi. Dette gøres for at bevare det funktionelle overblik.

Kodeudsnit 33: CalcRouteLengthAsc. Udregner afstanden fra bus til stoppested

```
1 drop function if exists CalcRouteLengthAsc $$
2 create function CalcRouteLengthAsc(bus_pos_lon decimal(20,15), ↩
    bus_pos_lat decimal(20,15), BusClosestEndPointID int, ↩
    busStopId int)
3 returns float
4 BEGIN
5 declare RouteCounter int default BusClosestEndPointID;
6
7 select bus_lon from ChosenRouteAsc where id = RouteCounter into ↩
    R2x;
8 select bus_lat from ChosenRouteAsc where id = RouteCounter into ↩
    R2y;
9 set BusToStop = Haversine(R2y, bus_pos_lat, R2x, bus_pos_lon);
10
11 while RouteCounter < busStopId do
12   select bus_lon from ChosenRouteAsc where id = RouteCounter into↩
       R1x;
13   select bus_lat from ChosenRouteAsc where id = RouteCounter into↩
       R1y;
14   select bus_lon from ChosenRouteAsc where id = RouteCounter+1 ↩
      into R2x;
15   select bus_lat from ChosenRouteAsc where id = RouteCounter+1 ↩
      into R2y;
16   set BusToStop = BusToStop + Haversine(R2y, R1y, R1x, R2x);
17   set RouteCounter = RouteCounter+1;
18 end while;
19 drop temporary table ChosenRouteAsc;
20 return BusToStop;
21 END$$
```

"RouteCounter"initialiseres til det tætteste rutepunkt, hvorefter ID'et for dette punkt bruges til at hente det første koordinatsæt ud fra "ChosenRouteAsc". Dette koordinat sæt bruges sammen med bussens koordinater til at udregne afstanden fra bussen til rutepunktet. Denne udregning sker i den anden matematik funktion, "Haversine". Funktionen

vil ikke beskrives videre i dette afsnit, men for mere information henvises der til afsnittet *8.2.5: Komponent 5: Anvendt Matematik*. Herefter itereres der igennem ruten, hvor ID'et for det tætteste rutepunkt på bussen er den nedre grænse, og ID'et for stoppestedet er den øvre. Ved hver iteration findes afstanden mellem det nuværende punkt og det næste, og den totale afstand inkrementeres med denne værdi. Til sidst returneres den totale afstand. "CalcRouteLengthDesc"gøres der brug af "ChosenRouteDesc"tabellen i stedet for "ChosenRouteAsc". Desuden bruges ID'et for stoppestedet nu som den nedre grænse og det tætteste rutepunkt som den øvre, og R̈outeCounter"dekrementeres i stedet for inkrementeres. Dette kan ses på kodeudsnit 34

Kodeudsnit 34: CalcRouteLengthDesc forskel

```
1 ...
2 while RouteCounter > busStopId do
3   select bus_lon from ChosenRouteDesc where id = RouteCounter ↩
       into R1x;
4   select bus_lat from ChosenRouteDesc where id = RouteCounter ↩
       into R1y;
5   select bus_lon from ChosenRouteDesc where id = RouteCounter-1 ↩
       into R2x;
6   select bus_lat from ChosenRouteDesc where id = RouteCounter-1 ↩
       into R2y;
7 ...
8   set RouteCounter = RouteCounter-1;
9 ...
```

CalcBusAvgSpeed

Denne funktion tages i brug i slutningen af "CalcBusToStopTime"proceduren, og bruges til at udregne bussens gennemsnitshastighed. Dette bruges sammen med bussens afstand til stoppestedet, til at udregne, bussens tid til ankomst ved stoppestedet.

Da det i denne funktion er ligegyldigt, hvilken rute bussen kører på, er det også ligegyldigt hvilken vej den kører. Derfor er det kun nødvendigt at have en funktion til at udregne gennemsnitshastigheden.På kodeudsnit 35 kan funktionen ses uden kommentarer eller initialiseringer uden værdi. Dette gøres for at bevare det funktionelle overblik. Efter udsnittet forklares funktion detaljeret.

Kodeudsnit 35: CalcBusAvgSpeed. Udregner gennemsnitshastigheden for en bus.

```
1   create function CalcBusAvgSpeed(BusId int)
2 returns float
3 begin
4   drop temporary table if exists BusGPS;
5   create TEMPORARY table if not exists BusGPS(
6   id int auto_increment primary key,
7   pos_lat decimal(20,15),
8   pos_lon decimal(20,15),
9   busUpdateTime time
10 );
11
12 insert into BusGPS (pos_lat, pos_lon, busUpdateTime)
13 select GPSPosition.Latitude,  GPSPosition.Longitude, GPSPosition.↩
     Updatetime from GPSPosition
14 where GPSPosition.fk_Bus=BusId order by GPSPosition.ID asc;;
15
16 select count(id) from BusGPS into MaxPosCounter;
17 while PosCounter < MaxPosCounter do
18
19   select pos_lon from BusGPS where id= PosCounter into R1x;
20   select pos_lat from BusGPS where id= PosCounter into R1y;
21   select pos_lon from BusGPS where id = PosCounter+1 into R2x;
22   select pos_lat from BusGPS where id = PosCounter+1 into R2y;
23
24   set Distance = Distance + Haversine(R2y, R1y, R1x, R2x);
25   select busUpdateTime from BusGPS where id= PosCounter  into ↩
       ThisTime;
26   select busUpdateTime from BusGPS where id = PosCounter+1  into ↩
       NextTime;
27   set secondsDriven = secondsDriven + (Time_To_Sec(NextTime) - ↩
       Time_To_Sec(ThisTime));
28   set PosCounter = PosCounter + 1;
29 end while;
30 set speed = Distance/secondsDriven;
31 drop temporary table BusGPS;
32 return speed;
33 end $$
```

Først udhentes alle GPS positioner og opdateringstiderne for disse, for det relevante bus ID. Dette data indskrives i "BusGPS", en temporary tabel, med et ID, som autoinkrementeres fra 1. Antallet af GPS opdateringer fundet for den givne bus, bruges i en while-løkke som den øvre grænse. En counter instantieres til 1, og bruges som den nedre grænse. Ved hver iteration hentes den opdatering af bussens position, vis ID i "BusGPS"svarer

til counteren. Den næste opdatering i rækken hentes også ud, og afstanden mellem de to punkter udregnes ved hjælp af "Haversine"funktionen. For mere information om "Haversine", se afnsit *8.2.5: Komponent 5: Anvendt Matematik*. Den totale afstand inkrementeres med den udregnede afstand. Herefter findes opdateringstiden for det første punkt, samt opdateringstiden for det næste. De to tidspunkter omregnes til sekunder, og tiden for det først punkt trækkes fra tiden for det næste. Således findes den tid, det har taget bussen at køre det linjestykke, som spændes over de to punkter. Den totale tid inkrementeres med den fundende tid.
Efter fuldent gennemiterering af bussens positioner, divideres den total afstand med tiden det har taget at køre afstanden. Således findes gennemsnitshastigheden, og denne værdi returneres.

Haversine og CalcRouteLineDist

Disse to funktioner vil ikke vises som kodeudsnit, da de blot er MySQL implementeringer af matematiske funktioner.
Haversine bruges til at udregne afstanden mellem to punkter, i en fugleflugt. CalcRouteLineDist bruges til at udregne afstanden fra et punkt, til det nærmeste punkt på en linje, udspændt af to andre punkter. Udregninger vil blive vist og forklaret nærmere under afsnittet *8.2.5: Komponent 5: Anvendt Matematik* under henholdsvist "Haversine"og "Tætteste punkt på en linje".

9.2 Implementering af persistens

Datapersistering og datahentning er vigtig komponent i dette system. Implementering af persistens vil derfor blive beskrevet meget nøje, og herunder delt op i tre dele; Implementering af persistens i mobilapplikation, Implementering af persistens i simulator og Implementering af persistens i online værktøjet. Hver del vil ikke have en beskrivelse af den fulde implementering, men blot repræsenteret af væsentlige dele. For fuld implementering af persistens henvises der til bilags CDen, i den respektive komponent under mappen Kode.

9.2.1 Implementering af persistens i mobilapplikationen

Persistering i denne komponent falder i to underpunkter. Dette er fordi, denne komponent er den eneste, som har kontakt til to databaser; Den distribuerede MySQL database samt den lokale SQLite database. Disse to vil blive beskrevet i seperate afsnit.

Tilgang til MySQL databasen

Mobilapplikationen har aldrig direkte tilgang til den distribuerede database. Tilgangen sker i afsnittet *8.2.2: Komponent 2: Mobil service*.

Applikation kommunikerer med databasen igennem en service, og altid kun som en læsning. Dette gør det muligt at tilgå databasen fra flere enheder, da en database læsning er trådsikker. Grunden til at der bliver gjort brug af en service er, at database tilgangen skal kunne gemmes væk fra brugeren, således en person ikke kan få fuld tilgang til databasen igennem sin mobil. Desuden vil mobil applikationen nemt kunne skiftes ud, uden at skulle tænke på tilgangen til databasen.

Selve kommunikationen med servicen sker igennem en SoapProvider. SOAP bruges som en transportmetode til XML beskeder. Når mobilen tilgår servicen opretter den en SOAP-envelope, der indeholder information om, hvilken metoden der skal kaldes, under hvilket namespace metoden ligger, samt eventuelle parametre metoden. På kodeudsnit 36 kan en generisk oprettelse og transmitering af en SoapEnvelope ses.

Kodeudsnit 36: Generisk SoapEnvelope.

```
1 SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
2 request.addProperty(PARAMETER_NAME, PARAMETER_VALUE);
3 SoapSerializationEnvelope envelope = new ↩
    SoapSerializationEnvelope(SoapEnvelope.VER11);
4 envelope.dotNet = true;
5 envelope.setOutputSoapObject(request);
6 HttpTransportSE androidHttpTransport = new HttpTransportSE(↩
    URL_OF_SERVICE);
7 androidHttpTransport.call(NAMESPACE+METHOD_NAME, envelope);
8 SoapObject response = (SoapObject)envelope.getResponse();
```

Requestet oprettes som et SoapObject, hvor metodenavnet, samt det namespace metoden ligger i, gives med. Disse to parametre er strings. Til metodekaldet kan der tilføjes pa-

rametre ved "addProperty"metode, som tager imod et parameter navn og en parameter værdi, begge to strings. Envelopen bliver oprettet og et versionsnummer bliver givet med, der definerer hvilken version af protokollen der skal tages i brug. I dette system bliver der udelukkende gjort brug af version 1.1. dotNet flaget er sat til true, da servicen er skabt i ASP.NET. Request-objektet sættes i envelopen, og kommunikerer med servicen over HTTP. Efter den relevante metode er færdigjort på servicen bliver returværdien sat i envelopen, og et SoapObject indeholdende de returnerede værdier kan hentes ved et kald til "getResponse"metoden på envelopen.

Et SoapObject er reelt set et XML-træ, som kan itereres igennem. Et eksempel på et sådan XML-struktur kan ses i afsnitet *8.2.2: Komponent 2: Mobil service*

Et fuldt eksempel på et kald til servicen kan ses på kodeudsnit 37. Denne funktion bruges til at hente samtlige busser med et givent busnummer, og returnere dem som en ArrayList.

Kodeudsnit 37: GetBusPos. Returnerer alle bussers position på en given rute.

```
1
2 final String NAMESPACE = "http://TrackABus.dk/Webservice/";
3 final String URL = "http://trackabus.dk/AndroidToMySQLWebService.↩
    asmx";
4 ...
5 public ArrayList<LatLng> GetBusPos(String BusNumber)
6 {
7     ArrayList<LatLng> BusPoint = new ArrayList<LatLng>();
8     try
9     {
10        SoapObject request = new SoapObject(NAMESPACE, "GetbusPos↩
              ");
11        request.addProperty("busNumber", BusNumber);
12        SoapSerializationEnvelope envelope = new ↩
              SoapSerializationEnvelope(SoapEnvelope.VER11);
13        envelope.dotNet = true;
14        envelope.setOutputSoapObject(request);
15      HttpTransportSE androidHttpTransport = new HttpTransportSE(↩
            URL);
16      androidHttpTransport.call(NAMESPACE+"GetbusPos", envelope);
17      SoapObject response = (SoapObject)envelope.getResponse();
18
19      for(int i = 0; i<response.getPropertyCount(); i++)
20      {
```

```
21          double a = Double.parseDouble(((SoapObject)response.↩
                getProperty(i)).getProperty(0).toString().replace(",",↩
                 "."));
22          double b = Double.parseDouble(((SoapObject)response.↩
                getProperty(i)).getProperty(1).toString().replace(",",↩
                 "."));
23          BusPoint.add(new LatLng(a, b));
24        }
25      }
26      catch(Exception e)
27      {
28        return null;
29      }
30      return BusPoint;
31  }
```

Metoden på servicen returnerer en liste, indeholdende typen "Point", som er en custom datatype lavet i servicen. Denne har to attributer, Latitude og Longitude, som begge er strings. "getPropertyCount"funktionen returner længden af denne liste, og bruges til at iterere igennem den.

Det første kald af "getProperty"på responset, returnerer "Point"datatypen. Denne property castes til et nyt SoapObjekt, hvor "getPropety"kaldes igen. Rækkefølgen af propeties i et SoapObject, defineres af rækkefølgen de bliver oprettet i, i datatypen. I "Point"kommer latitude først og longitude kommer bagefter. "GetProperty(0)"på et "Point"SoapObjekt vil derfor returnere Latitude og "GetProperty(1)"vil returnere Longitude. Begge bliver castet til en string, og floatingpoint sættes til et dot frem for et komma. Dette gøres da ASP.NET tager et floatingpoint som værende komma.

Da applikationen er lavet til Android bruges biblioteket ksoap2[24], som er specifik for Android. I dette bibliotek ligger alle funktioner, der er nødvendige for at bruge af SOAP.

Tilgang til SQLite databasen

Når en busrute favoriseres gemmes alt data om denne i en lokal SQLite database. Dette gøres for at spare dataforbrug, hvis en rute tages i brug ofte. Samtidig muligøres det også, at brugeren kan indlæse en rute med stoppestedder, uden at have forbindelse til internettet. Hvis kortet samtidig er cachet (Google Maps cacher indlæste kort), kan kortet også indlæses og indtegnes.

Der er gjort brug af en ContentProvider i denne sammenhæng, som abstraherer data-

[24]For mere information, se http://ksoap2.sourceforge.net/

access laget, så flere applikationer kan tilgå databasen, med den samme protokol, hvis det skulle være nødvendigt.

En ContentProvider tilgås igennem et kald til "getContentResolver", hvorefter der kan kaldes til de implementerede CRUD-operationer. En ContentProvider skal defineres i projektets AndroidManifest, før den kan tilgås. Dette gøres ved at give den et navn, samt en autoritet, som dette tilfælde, er den samme værdi som navnet. Hvis ContentProvideren bruges i en anden applikationen, skal den have en autoritet tilsvarende den, den er blevet oprettet med.

Da ContentProvideren blot er et transportlag mellem brugeren og databasen, er det nødvendigt for den, at kende den egentlige database. Dette er gjort ved at lave en inner class til provideren, som extender SQLiteOpenHelper. Denne klasse indeholder create proceduren, samt muligheden for at kunne tilgå både en læsbar og skrivbar version af databasen. Create proceduren bliver kørt, hvis databasen med det valgte navn ikke eksisterer i forvejen, og bruges til at oprette databasen og tabellerne deri. En SQLite database gør, som default, ikke brug af foreign key constraints. Det er derfor blevet implementeret sådan, at foreign key contraints aktiveres hver gang databasen åbnes.

Hver CRUD-operation modtager et URI, der skal være en kombination af en identifieren "content://", en autoritet (ContentProviderens placering) samt evt. en tabel og en underoperation. Hvis en given CRUD-operation på ContentProvider siden er lavet, sådan at der altid gøre det samme (f.eks. en query der altid returner alt data i den samme tabel), vil identifieren og autoriteten være nok, til at kunne tilgå denne operation. Hvis tilgangen derimod skal være specifik for en given tabel, og evt. underoperation kan en UriMatcher tages i brug. Denne kobler et URI med en given værdi, hvorefter der i operationen kan laves en switch/case der matcher det medsendte URI, og vælger en operation ud fra dette. På 38 ses et eksempel på, hvordan dette er implementeret i systemet. Det skal noteres at dette ikke er komplet implementering, men blot et udsnit. Kommentarer vises ved "!!"

Kodeudsnit 38: GetBusPos. ContentProvider implementering.

```
1 !!ContentProvider class!!
2 public static final String AUTHORITY = "dk.TrackABus.↩
    DataProviders.UserPrefProvider";
3 public static String BUSSTOP_TABLE = "BusStop";
```

```
4 private static final int BUSSTOP_CONTEXT = 1;
5 private static final int BUSSTOP_NUM_CONTEXT = 2;
6 ...
7 static {
8   uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
9   uriMatcher.addURI(AUTHORITY, BUSSTOP_TABLE, BUSSTOP_CONTEXT);
10  uriMatcher.addURI(AUTHORITY, BUSSTOP_TABLE+"/#", ↩
       BUSSTOP_NUM_CONTEXT);
11 }
12 ...
13 public Cursor query(Uri uri, String[] projection, String ↩
     selection,
14     String[] selectionArgs, String sortOrder)
15 String Query;
16 SQLiteDatabase db = dbHelper.getReadableDatabase()
17 switch(uriMatcher.match(uri))
18 {
19   case BUSSTOP_CONTEXT:
20     routeID = selection;
21     query = !!Query to get all busstops and their position on a ↩
          route with id being RouteID!!
22     returningCursor =  db.rawQuery(query,null);
23     break;
24   case BUSSTOP_NUM_CONTEXT:
25     stopID = uri.getLastPathSegment();
26     query = !!Query to get a single bustop with id being stopID!!
27     returningCursor =  db.rawQuery(query,null);
28   default
29     return null;
30 }
31 return returningCursor;
32
33 ...
34 !!BusStop model class!!
35 public static final Uri CONTENT_URI = Uri.parse("content://"
36       + UserPrefProvider.AUTHORITY + "/BusStop");
37
38 ...
39 !!Hentningen af stoppesteder!!
40 getContentResolver().query(UserPrefBusStop.CONTENT_URI, null, ↩
     RouteID, null, null);
41 String StopID = !!Some ID!!
42 String specifikStop = UserPrefBusStop.CONTENT_URI.toString() + "/↩
     "+ StopID;
43 getContentResolver().query(Uri.parse(specifikStop),null,null,null↩
     ,null);
```

Den første del af kodeeksemplet viser oprettelsen af UriMatcheren. Hvis UriMatcheren kender det ID den bliver givet ved i dennes "match"funktion, vil den returne en værdi, der svarer til den, den er blevet givet ved oprettelse. Herefter ses et udsnit af "query"metoden. Hvis URI'et kun indeholder BusStop udover autoriteten, vælges BUSSTOP_CONTEXT, og der hentes alle stoppesteder, som er relevant for den rute der sættes i selection parameteren. Hvis tabellen efterfølges af et nummer i URI'et, vælges BUSSTOP_NUM_CONTEXT, og der hentes kun det stoppested som har det ID sat i URI'et.
Til samtlige tabeller i SQLite databasen er der lavet en model klasse. Disse klasser indeholder kun statiske variabler som definerer den givne tabels kolonner samt den URI, ContentProvideren skal have med, for at kunne tilgå den tabel, modellen definerer.
I sidste del af kodeafsnittet kan det ses, hvordan ContentProvideren tilgås. Det første kald tilgår query funktionen under BUSSTOP_CONTEXT, og henter alle stoppesteder ud for ruten hvor ID'et er "RouteID". Det andet kald tilgår også query funktionen men under BUSSTOP_NUM_CONTEXT, og henter stoppestedet ud hvor ID'et er "StopID".

9.2.2 Implementering af persistens i simulator

Simulatoren implementerer persistens i form af at hente ruter, opdatere hvilken vej en bus kører, samt udregne og persistere ny GPS position for en bus. Samtlige busser kører i deres egen tråd i simulatoren, derfor er det vigtigt at håndtere trådsikkerhed når databasen skal tilgås. DatabaseAccess klassen tager sig af selve databasen tilgangen, og indeholder to funktioner; En til at skrive til databasen, samt en til at læse. Begge funktioner er statiske, og indeholder en binær semafor, således kun en tråd af gangen kan tilgå databasen. Hvis en tråd allerede er igang med en datahentning eller -skrivning, vil den anden tråd tvinges til at vente, til processen er færdig. Begge funktioner modtager en string, som er den kommando der skal udføres på database. Funktionen der læser fra databasen tager yderligere en liste af strings, som indeholder de kolonner der skal læses fra. Efter fuldent tilgang returneres en liste af strings, med de værdier der er blevet hentet.
Databasen tilgangen bliver håndteret med i librariet MySQL.Data.[25] Da simulatoren er lavet i Visual Studio 2012, og er en WPF-applikation, er der blot gjort brug af NuGet[26]

[25] For mere information, se http://dev.mysql.com/doc/refman/5.6/en/connector-net.html
[26] Et integret værktøj i Visual Studio 2012, til at hente og tilføje biblioteker

til at hente og tilføje dette bibliotek til programmet. Forbindelsesopsætningen ligger i App.config filen, og hentes ud når der skal bruges en ny forbindelse. På kodeudsnit 39 ses funktionen der læser fra databasen, samt hvordan den tilgås. Kun denne vil vises, da det er den mest interresante. Fuld kode kan findes på bilags CDen under Kode/Simulator.

Kodeudsnit 39: Simulator. Select statement.

```
1 public static bool SelectWait = false;
2 public static List<string> Query(string rawQueryText,List<string>↩
      columns)
3 {
4   while(SelectWait)
5   {
6     Thread.Sleep(10);
7   }
8   SelectWait = true;
9     using(MySqlConnection conn = new MySqlConnection(↩
         ConfigurationManager.ConnectionStrings["TrackABusConn"].↩
         ToString()))
10  {
11    using(MySqlCommand cmd = conn.CreateCommand())
12    {
13      try
14      {
15        List<string> returnList = new List<string>();
16        cmd.CommandText = rawQueryText;
17        conn.Open();
18        MySqlDataReader reader = cmd.ExecuteReader();
19        while (reader.Read())
20        {
21          foreach (string c in columns)
22          {
23            returnList.Add(reader[c].ToString());
24          }
25        }
26        reader.Close();
27        conn.Close();
28        SelectWait = false;
29        return returnList;
30      }
31      catch(Exception e)
32      {
33        SelectWait = false;
```

```
34          return null;
35        }
36      }
37    }
38 }
39 ...
40 String query = "Select BusRoute.ID from BusRoute";
41 List<string> queryColumns = new List<string>(){"ID"};
42 List<string> returnVal= DatabaseAcces.Query(query, queryColumns);
```

Der ventes i starten af funktionen på, at semaforen frigives. Hvis tråden skal tilgå databasen og en anden tråd allerede er igang, ventes der på, at den låsende tråd gør processen færdig og sætter SelectWait til false.

Når forbindelsen oprettes, gives den en configurations string. Denne string indeholder database navn, server, brugernavn og password, som er alt hvad forbindelsen skal bruge, for at tilgå databasen. Af denne forbindelse laves der en kommando, som indeholder alt den information som skal eksekveres på forbindelsen. Ved kaldet til "ExecuteReader", udføres kommandoen og en reader returneres med de rækker der kunne hentes udfra den givne query. I skrivnings funktionen ville "ExecuteNonQuery", blive kaldt i stedet, da der, i dette tilfælde, ikke skal returneres noget data. Readeren repræsenterer en række i databasen, og når "Read"funktionen kaldes på den, tilgås den næste række. Hvis "Read"returner false, er der ikke flere rækker at læse. Når data skal hentes ud fra readeren, kan der enten vælges at bruge index (kolonne nummeret i rækken), eller kolonnenavn. I dette tilfælde gives samtlige kolonner med som en parameter, og derfor læses der på navn. Til sidst frigøres semaforen og læst data returneres.

I slutningen af kodeudsnittet kan det ses, hvordan denne funktion tilgås. Først laves der en query, som i dette tilfælde henter samtlige busrute ID'er. Herefter oprettes der en liste af de kolonner der skal hentes, hvorefter Query funktionen kaldes med begge værdier.

9.2.3 Implementering af persistens i online værktøjet

Online værktøjet består af to dele; Mobil servicen og hjemmesiden. Begge dele er lavet i ASP.NET, og derfor vil database tilgangs proceduren være ens med simulatoren. Servicen står får at lade mobil applikationen tilgå data på MySQL databasen, hvilket også betyder, at funktionerne kun læser data. Ved et kald til servicen vil læst data pakkes ved hjælp af SOAP, som er beskrevet i *9.2.1: Implementering af persistens i mobil applikationen.*

Servicen står i midlertid også for at kalde tidsudregnings proceduren på databasen, hvilket er et anderledes kald, end en læsning. På kodeudsnit 40 ses det, hvordan servicen tilgår denne procedure. Det skal noteres at det ikke er den fulde funktion der vises, men blot et udsnit, og derfor kun viser de vigtigste dele. Herved vises der ikke hvordan forbindelsen og kommandoen laves, da oprettelsen er ens med simulatoren.

Kodeudsnit 40: Service. Udsnit af kald til tidsudregnings procedure

```
1 ...
2 cmd.CommandText = "CalcBusToStopTime";
3 cmd.CommandType = System.Data.CommandType.StoredProcedure;
4 ...
5 cmd.Parameters.Add("?stopName", MySqlDbType.VarChar);
6 cmd.Parameters["?stopName"].Value = StopName;
7 cmd.Parameters["?stopName"].Direction = System.Data.↩
    ParameterDirection.Input
8 ...
9 cmd.Parameters.Add(new MySqlParameter("?TimeToStopSecAsc", ↩
    MySqlDbType.Int32));
10 cmd.Parameters["?TimeToStopSecAsc"].Direction = System.Data.↩
    ParameterDirection.Output;
11 ...
12 cmd.ExecuteNonQuery();
13 ...
14 string TimeToStopAsc = cmd.Parameters["?TimeToStopSecAsc"].Value.↩
    ToString();
15 string EndStopAsc = cmd.Parameters["?EndBusStopAsc"].Value.↩
    ToString();
```

I kodeudsnittet kan det ses, at der bliver tilføjet flere værdier til kommandoen i forhold til kodeudsnit 39, hvor der kun blev tilføjet en enkelt. Først og fremmest bliver kommandotypen sat som værende en Stored Procedure. Herefter kan det ses hvordan både en input og en output parameter bliver sat i kommandoen. Parametrene bliver givet et navn, samt en datatype, hvorefter de gives en værdi hvis de er input parametre. Herefter gives parametrene en retning; Input hvis de er værdier der skal læses i proceduren og output hvis de skal skrives til. Efter proceduren er kørt, vil output parameterne nu kunne læses, med de værdier der er blevet udregnet. I dette tilfæde er der kun vist to parametre, men antallet og deres navne og retning, skal passe overens med den procedure der er lavet på database siden. De oprettede Stored Procedures er trådsikre, og dette beskrives i afsnittet

9.1.2: Stored Procedures.

Da databasetilgangen på hjemmesiden kun er flertrådet når der læses, er systemet trådsikkert. Når der læses vil det altid ske fra den med server tråden. Databasen tilgås ligesom servicen og simulatoren ved hjælp af MySql.Data bibilioteket[27], og tilgås kun i form af simple CRUD-operationer. Der vil derfor ikke vises et kodeeksempel, da dette anses som værende beskrevet i tidligere afsnit. Samtlige funktioner er samlet i DBConnection klassen, som agerer som data tilgangs laget. Det vil sige, at alle database aktioner tilgås igennem denne klasse.

[27] For mere information, se http://dev.mysql.com/doc/refman/5.6/en/connector-net.html

10 GENERELLE DESIGNBESLUTNINGER

10.1 Arkitektur mål og begrænsninger

Der var ikke fastsat nogen funktionelle krav til udviklingsværtøjer eller arkitektur fra et produktoplæg, og derfor blev produktet udhviklet med de værktøjer, der var mest erfaring i. Da projektet dog bygger ovenpå en eksamensopgave udført i ITSMAP, som var Android baseret, blev det fastsat, at mobilapplikationen skulle udføres på samme Android platform.
Simulatoren er udbygget i WPF med .NET 4.5 frameworket, hvilket betyder at simulatoren skal bruges på en Windows platform.
En domænenavn blev købt hos UnoEuro[28], til at holde administrator delen af produktet, samt den distribuuerede database. Dette domæne satte visse begrænsninger for systemet, da kun en MySQL database var supporteret, og der skulle vælges imellem PHP og ASP.NET. Da ASP.NET er C# baseret, blev denne mulighed valgt, da projektgruppen havde mere erfaring med dette, end med PHP.

10.2 Arkitektur mønstre

Nedenstående liste er en opremsning af de arkitekturmønstre, der er anvendt i systemet. Under listen er de forskellige mønstre dokumenteret.

- Tre-lags model
- MVC (Model-View-Controller)
- ContentProvider
- Bound Service

Tre-Lags model

Tre-lags modellen bruges i sammenhæng med seperation-of-concerns, til at splitte et projekt op i præsentation, logik og data tilgang. Præsentations laget sørger for at håndtere

[28] www.unoeuro.com

alt visuelt, logik laget laver manipulationer på data, og data tilgangen sørger for at tilgå de eksterne data kilder og hente relevant data.
Se http://en.wikipedia.org/wiki/Multitier_architecture for mere dokumentation.

MVC

En afart af tre-lags modellen og bruges i sammenhæng med hjemmeside design i ASP.NET. Viewet præsenterer data for brugeren, hentet fra modellen. Brugeren har også mulighed for at tilgå controlleren igennem viewet og ændre data sat i model-laget. Efter kaldet, vil viewet blive opdateret med de nye ændringer.
Se http://en.wikipedia.org/wiki/Model-view-controller for mere dokumentation.

ContentProvider

En Android specifik metode til at tilgå data. Bruges i sammenhæng med data-deling, mellem applikationer. Ved at bruge forskellige URI'er, kan et kald til samme funktion, tilgå forskellige funktionaliteter.
Se http://developer.android.com/reference/android/content/ContentProvider.html for mere dokumentation.

BoundService

Et Android specifikt interface til en client/server implementering. Når en service er bound, kan den kun tilgås i den givne applikation, fra de klasser der binder til den. Efter service kaldet er færdiggjort, returneres der til den kaldende klasse, over en message handler.
Se http://developer.android.com/guide/components/bound-services.html for mere information.

10.3 Generelle brugergrænsefladeregler

Kravene til brugergrænsefladen kan deles op i to kategorier:

- Arkitekturspecifikke
- Udseendesspecifikke

Begge er beskrevet herunder

10.3.1 Arkitekturspecifikke

Samtlige brugergrænseflader er bygget op på tre-lags modellen, som beskrevet i afsnit *10.2: Arkitekturmønstre*. Dette er gjort på baggrund af, at brugergrænsefladen nemt kan skiftes ud, hvis det skulle blive nødvendigt, hvilket betyder en lavere kobling mellem data og præsentation. Hjemmesiden er bygget op på baggrund af MVC, som er specifikt for ASP.NET. Dette gør, ligesom tre-lags modellen, at hver komponent er udskiftelig, uden at der er behov for, at ændringer foretages i viewet.

10.3.2 Udseendesspecifikke

Under applikations udvikling, blev det besluttet, at bruge to nuancer af rød, til opbygning af grænsefladen. Disse to er baseret på "midttrafik rød", da det ville være dette firma, applikation skulle udvikles til, hvis et firma skulle kobles på projektet.

10.4 Exception og fejlhåndtering

10.4.1 Fejlhåndtering ved tab af internet forbindelse

Da systemet er meget afhængigt af kommunikation med den distribuerede database, håndteres der i samtlige komponenter at forbindelsen til internettet kan gå tabt.

- Mobil applikationen sørger for, at hvis internettet går tabt, bliver brugeren notificeret om, at der ikke er internet længere, og derfor kan den givne funktion ikke længere udføres. Applikation sørger for, at hvis forbindelsen genetableres fortsættes den givne funktion. Dette er mest relevant under opdatering af bussernes position og/eller tid til valgt stop. En BroadcastReceiver bruges i sammehæng med at undersøge forbindelsen til internettet. Når der sker en ændring i wifi, eller i det mobile data netværk, vil denne undersøge, om der efter den nye ændring er tilgang til internettet. Et statisk flag sættes i klassen som symboliserer telefones forbindelse til netværk. Hvis denne er sat til false, vil data ikke hentes, og en fejlbesked vil vises.

- I tilfælde af, at internettet forsvinder imens en hjemmeside er indlæst, og en database tilgangs funktion tilgås, er der sat et timeout på tilgangen på to sekunder. Hvis dette timeout nås, vil administratoren notificeres om, at internettet ikke kan tilgås.

10.5 Implementeringssprog og værktøjer

Herunder beskrives udviklingsværktøjer. Afsnittet er ikke ment som et udtømmende afsnit, og derfor vil alle værktøjer der har været brugt under projektet, ikke nødvendigvis beskrives.

10.5.1 C#

Hjemmesidens logik lag, samt data tilgangs lag og simulatoren er skrevet i C#. Dette er gjort, da begge gør brug af .NET frameworket og dens biblioteker.

10.5.2 Java

Mobil applikation er skrevet i Java, som er standard for Android applikationer

10.5.3 HTMl, CSS og JavaScript

Selve viewet på adminstrator værktøjet, er bygget op som en standard hjemmeside. Det vil sige med HTML, CSS og JavaScipt.

10.5.4 MySQL

Den distruberede database er baseret på MySQL, og derfor er oprettelse, funktioner og procedurer skrevet i dette sprog.

10.5.5 Microsoft Visual Studio 2012

Administrator delen og simulatoren er bygget på .NET frameworket, og derfor skrevet i Visual Studio.

10.5.6 Eclipse

Mobil-applikationen er skrevet til Android, og det værktøj som projektgruppen havde mest erfaring med, var Eclipse.

10.5.7 MySQL Workbench

Dette program er brugt til at opsætte og ændre MySQL databasen.

10.5.8 Microsoft Visio 2013

Samtlige diagrammer i dette dokument er udbygget ved hjælp af Visio. Der er nogle enkelte diagrammer i kravspecifikationen der er blevet lavet ved hjælp af online-værktøjet creately.com

10.5.9 Git og GitHub

Til versions styring er der blevet gjort brug af GIT, som er implementeret i GitHub. Til denne tilgang er der brugt commandline værktøjet Git shell.

10.5.10 TexMaker

Alt dokumentation er skrevet i LaTeXog som editerings-værktøj er der brugt TexMaker.

10.6 Implementeringsbiblioteker

Både de systemer i projekter der er baseret på .NET og på Android, gør brug af eksterne biblioteker. Disse vil her beskrives

- MySQL.Data - Bibliotek som indeholder alle nødvendige værktøjer til at tilgå en MySQL database. Bruges i simulatoren og i online-værktøjet.
- ksoap2 - Bibliotek til at håndtere SOAP-protokollen på android applikationen. Indeholder alle funktionaliteter til dette formål.
- Google Play Services v12 - Biblotek til at håndtere oprettelse og vedligeholdelse af kort på android applikationen
- Google Maps JavaScript API v3 - Biblotek til at håndtere oprettelse og vedligeholdelse af kortet på administrator hjemmesiden.
- Google Maps Direction API - Bibliotek til at håndtere ruteopretning og vedligeholdelse på administrator hjemmesiden.

11 STØRRELSE OG YDELSE

I dette afsnit er der angivet de kritiske størrelse og ydelsesparametre for systemet.

11.1 Dataforbrug for mobil applikation

Det er ikke altid der er adgang til wifi netværksforbindelse og det vil derfor være nødvendigt at bruge telefonenes mobile data netværk, til at hente ruter, stoppesteder og bussernes position fra databasen. Dataforbruget på en time, hvor kortet bliver vist og bussernes position bliver opdateret ligger på ca. 4 megabytes. Er et stoppested valgt, og tidsopdateringer foregår, vil dataforbruget ligge på ca. 7 megabytes på en time.

11.2 Performance for mobil applikation

Lagerforbruget på mobiltelefonen er omkring 4 megabytes.
Der benyttes i gennemsnit ca. 7% CPU kraft.

11.3 Server

Lagerforbruget på serveren, til hjemmesiden, MySQL databasen og servicen ligger på omkring 6 megabytes.
Serveren står for at lave alle udregnigner og dataprocessering som mobil applikationen skal bruge. Da det er tænkt som et distribueret system, kan der forkomme mange forbindelser til serveren på samme tid. Der har dog ikke været mulighed for at udføre en stres-test af serveren, i sammenhæng med en undersøgelse af, hvor mange forbindelser den kan håndteres på en gang, da der ikke har været ressourcer til mere end to mobil telefoner.

11.4 Performance for simulator

Lagerforbruget på disken ligger omkring 2 megabytes.
Der benyttes ca. 33.4 megabytes memory og gennemsnit mellem ca. 0.16GHz og 0.4Ghz CPU kraft ved normalt brug.

12 KVALITET

12.1 Brugervenlighed

Der er sat stort fokus på brugervenlighed, da det er tænkt som et distribueret system, der skal bruges af et mange forskellige personer. Dette er blevet opnået ved, at der er tydelige knapper med sigende tekst og at brugeren skal gøre så lidt som muligt for at opnå de resultat der ønskes.

12.2 Pålidelighed

Det er vigtigt at systemet er pålideligt, da hovedfunktionaliteten i mobil applikationen er, at præcist kunne vise bussens position samt tiden til ankomst ved et valgt stoppested. Dette er blevet opnået ved at bussens position bliver opdateret hvert sekund, og tiden bliver udregnet hver andet sekund.

12.3 Effektivitet

Det er vigtigt at brugeren hurtigt kan få vist en busrute samt tiden til ankomst for en ved et stoppested. Dette er blevet opnået ved at gøre alle tunge udregninger og data processering på en server, så mobil telefonen ikke bliver belastet med det.

12.4 Udvidbarhed

Der er blevet sat et stort fokus på, at hele systemet skal være nemt at videreudvikle på, og vedligeholde. For at opnå dette der systemet bla. blevet opbygget på tre-lags modellen, samt mængden af udregninger der sker på serveren. Dette gør det er nemt at udskifte de forskellige udregninger, og hvilken database der bliver brugt, uden at skulle opdatere selve mobil applikationen. Samtidig gøres det nemt at udvikle applikationen til en ny platform, da der ikke skal tages højde for database tilgang og udregninger.

13 OVERSÆTTELSE

I dette afsnit beskrives, hvordan der kommes fra kildeteksten til objektkoden, altså til et program, der er klart til at blive kørt. mobil applikationen er pre-compiled og kræver derfor kun en installering. Administrations hjemmesiden er published til www.trackabus.dk og det er derfor ikke nødvendig at publish den igen, medmindre der er fortaget ændringer i den. Simulatorenen er designet til udvikleren, og der er derfor ikke lavet en installer til den.

13.1 Oversættelses-software

Den software, der er nødvendig for at oversætte mobil applikationen er compileren indbygget i Eclipse. Ved kompilering vil det resultere i TrackABus.apk filen, der bruges til at installere programmet på mobil telefonen.
Til administration hjemmesiden og simulatoren skal compileren i Visual Studio 2012 bruges.

13.2 Installation

Den arbejdsgang, der skal følges, for at de oversatte programmer kan bringes til at køre i de rette omgivelser er at følgende skal udføres:
Mobil applikationen skal være tilsluttet til en computer, hvorpå TrackABus.apk filen ligger. TrackABus.apk overføres nu til mobil telefonen, hvorefter der, via et filhåndterings program på mobil telefonen, navigeres hen til .apk filen. Ved et tryk på denne vil TrackABus applikationen nu blive installeret på mobil telefonen.
Hjemmesiden skal publishes til serveren, som gøres fra Visual Studio 2012. Først højreklikkes der på projektet "MapDrawRouteTool"hvorfra der nu vælges "Publish...". Der oprettes en ny publish profil med følgende connection indstillinger:

- **publish method:** FTP
- **Server:** trackabus.dk
- **Site path:** public_html
- **User name:** trackabus.dk

- **Password:** 1083209421
- **Destination URL:** trackabus.dk

  Dette vil publishe hjemmesiden til www.TrackABus.dk hvor den nu kan tilgåes og bruges.

Simulatorenen bruges i udviklings formål, og skal derfor ikke installeres. Denne køres blot fra .exe filen.

14 KØRSEL

Når programmet skal køres, er der visse forudsætninger, der er gældende. Dette er beskrevet i følgende afsnit.

14.1 Kørsels-software

Før de forskellige dele af programmet kan køres, kræves henholdsvist, at mobil applikationen er installeret på en mobil telefon, simulatereren er kompileret og hjemmesiden er published til en server. Se *afsnit 13.2: Installation* for en beskrivelse af, hvordan dette gøres.

14.2 Start, genstart og stop

Mobil applikationen startes ved at klikke på TrackABus programmet på mobil telefonen, hvorefter TrackABus programmet åbnes og nu kan bruges af brugeren.
Applikationen sættes i dvale, som alle andre mobil applikationer, ved at enten trykke på telefonens "Home"knap, og ved genstart, vil starte applikationen fra samme sted hvor den blev lukket. For at stoppe applikationen helt, kan der trykkes på telefonens "Tilbage"knap, indtil applikationen ikke længere er åben.

Administrations hjemmesiden vil, så længe den er hosted på serveren, altid være startet. Administratoren kan "starte"hjemmesiden ved at gå ind på www.trackabus.dk
Det vil være muligt at lukke hjemmesiden, som alle andre hjemmesider, ved enten at trykke på det røde kryds i højre hjørne, eller navigere til en anden hjemmeside. Hjemmesiden kan selvfølgelig også stoppes ved, at enten slette den fra serveren eller slukke helt for den server den er hostede på, men dette frarådes

Simulatoren startes ved double klik på at GPSSimu.exe. Programmet lukkes som ethvert andet program, ved at trykke på det røde kryds i højre hjørne.

14.3 Fejludskrifter

Fejlbeskeder for mobil applikationen består af små beskeder der vises på skærmen i et kort stykke tid, uden at forstyre brugeren. For administrations hjemmesiden, vil administratoren blive præsenteret for en fejlbesked på skærmen, der beskriver kort, hvad der er gået galt.
De følgende fejlbeskeder, beskriver de fejl der kan forkomme på mobil applikationen.

- **"Please check internet connection, and try again"**
  - Fejlen kan forekomme flere steder i systemet, hver gang der skal skabes forbindelse til internettet. Beskeden forkommer når der ikke kan skabes forbindelse til internetet.
- **"No connection to internet. Can only show route for favorite bus "**
  - Denne besked forekommer, hvis brugeren trykker på en busrute der ikke er favoriseret, i listen af alle busruter, og der ikke er forbindelse til internettet. Beskeden beskriver, at det kun er muligt at vise de ruter, der er blevet favoriseret, når der ikke er forbindelse til internettet.
- **"Please wait until other route is favorited/unfavorited"**
  - Denne besked forekommer, hvis brugeren prøver at favorisere eller fjerne en rute fra favoriseringen, imens en anden rute er i gang med at blive favoriseret/ufavoriseret.
- **"No connection to internet. Cannot set route to favorite."**
  - Denne besked forekommer, hvis brugeren prøver at favorisere en busrute, imens der ikke er forbindelse til internettet.
- **"Cannot favorite this route"**
  - Denne besked forekommer, hvis den rute brugeren ønsker at favorisere enten ikke har nogle rute punkter, eller ikke har nogle stoppesteder på sig. Den kan også forkomme, hvis forbindelse til internettet tabes imens en rute er igang med at blive favoriseret.

- **"Not connected to internet, can only show route"**
  - Denne besked forekommer når en favoriseret rute bliver valgt, og der ikke er forbindelse til internettet. Beskeden beskriver, at kun ruten og stoppesteder kan indtegnes på kortet. Der vil ikke blive vist nogle busser, da de ikke kan hentes fra MySQL databasen.
- **"No bus found on route"**
  - Denne besked forekommer når der bliver valgt en rute, og der ikke findes nogle busser i databasen, der køre på den valgte busrute.
- **"There are no bus stops on chosen route"**
  - Denne besked forekommer når der bliver valgt en rute, og der ikke findes nogle busstoppesteder på den valgte busrute.
- **"No connection to internet. Cannot update position"**
  - Denne besked forekommer, hvis forbindelsen til internettet mistes, imens kortet, med indtegnet rute, stoppesteder og bus, bliver vist. Kortet med rute og stoppesteder vil stadigvæk blive vist, dog vil bussens position ikke længere opdateres, så længe der ikke er forbindelse til internettet.
- **"No connection to internet. Cannot update time"**
  - Denne besked forekommer, hvis et stoppested bliver valgt og der ikke er forbindelse til internettet, og tiden til et stoppestedet derfor ikke kan udregnes.

De følgende fejlbeskeder, beskriver de fejl der kan forkomme på administrations hjemmesiden.

- **"No Internet connection"**
  - Fejlen kan forekomme flere steder i systemet, hver gang der skal skabes forbindelse til internettet. Beskeden forkommer når der ikke kan skabes forbindelse til internetet.
- **"Can't remove bus. Remove it from its route first"**
  - Denne besked forekommer hvis en bus bliver forsøgt slettet fra systemet, og den stadigvæk er knyttet til en busrute. administratoren kan fjerne bussen fra busruten og prøve igen.
- **"Bus already exist"**
  - Dette besked forekommer, hvis administratoren prøver at tilføje en bus til systemet med et ID, imens der allerede findes en bus i systemet med samme ID.
- **"Failed to save, please try again"**
  - Denne besked forekommer, hvis der sker en fejl imens der bliver gemt på databasen. Dette kan forkomme hvis nettet forsvinder kort, eller der sker en anden serverfejl. Administratoren bør prøve igen. Hvis dette ikke virker foreslåes det at hjemmesiden bliver genindlæst.
- **"Please name the stop"**
  - Dette besked forekommer, hvis administratoren prøver at gemme et stoppested, uden at have navngivet det først.
- **"There are busses on the chosen route, please delete them first"**
  - Denne besked forekommer, hvis administratoren prøver at slette en busrute fra systemet, men der stadigvæk er busser der er knyttet til den valgte rute. Administratoren kan fjerne busserne fra ruten og prøve igen.

- **"The route does not have any busstops"**
  - Denne besked forekommer, hvis administratoren prøver at gemme en busrute, uden at have tilføjet nogle stoppesteder til ruten først. Administratoren skal tilføje stoppesteder og prøve igen.
- **"Please Type in a route number"**
  - Denne besked forekommer, hvis administratoren prøver at gemme en busrute, uden at have navngivet det først.

15 Litteraturliste

Google. *The Google Map JavaScript API v3.* Lokaliseret d. 23. september 2013: https://developers.google.com/maps/documentation/javascript/.

Oracle, *22.2. MySQL Connector/Net.* Lokaliseret d. 23. september 2013: http://dev.mysql.com/doc/refman/5.6/en/connector-net.html.

Veness, Chris. *Calculate distance, bearing and more between Latitude/Longitude points.* Lokaliseret d. 23. september 2013: http://www.movable-type.co.uk/scripts/latlong.html/

DFKI & Saarland University. *Equation of a linear function through two points.* Lokaliseret d. 23. september 2013: http://demo.activemath.org/ActiveMath2/search/show.cmd?id=mbase://AC_UK_calculus/functions/ex_linear_equation_two_points

Garner, Will. *Shortest Distance from a Point to a Line.* Lokaliseret d. 23. september 2013: http://math.ucsd.edu/~wgarner/math4c/derivations/distance/distptline.htm

Google. *The Google Directions API.* Lokaliseret d. 28. oktober 2013: https://developers.google.com/maps/documentation/directions/.

kSOAP2. Lokaliseret d. 28. oktober 2013: http://ksoap2.sourceforge.net/.

Bilag

Nedenfor er mappestrukturen i bilagsmappen listet.

Biblioteker

Diagrammer

1. Database diagrammer
2. Domænemodeller
3. Klassediagrammer
4. Sekvensdiagrammer
    (a) UseCase Realiseringer
    (b) Simulator
    (c) Applikation
    (d) Hjemmeside
    (e) Tasks
5. UseCase Diagrammer

Billeder

1. Brugergrænsefladen
2. Tætteste punkt på en linje
3. Processrapport

Kode

1. Mobil applikation
2. Simulator
3. Online værktøj
4. Database

Installationsfiler