TrackABus

BACHELORPROJEKT

Systemarkitektur for TrackABus

Author:

Gruppe 13038

Supervisor:

Michael Alrøe

7. december 2013

Versionshistorie:

Ver.	Dato	Initialer	Beskrivelse
0.1	18-11-2013	??	Arbejde påbegyndt
0.2	03-12-2013	??	Use Case View færdiggjort

Godkendelsesformular:

Forfatter(e):	Christoffer Lousdahl Werge (CW) Lasse Sørensen (LS)
Godkendes af:	Michael Alrøe.
Projektnr.:	bachelorprojekt.
Filnavn:	Systemdesign.pdf
Antal sider:	66
Kunde:	Michael Alrøe (MA).

Sted og dato: _____

10832 Christoffer Lousdahl Werge

MA Michael Alrøe

09421 Lasse Lindsted Sørensen

Indhold

1 USE CASE VIEW

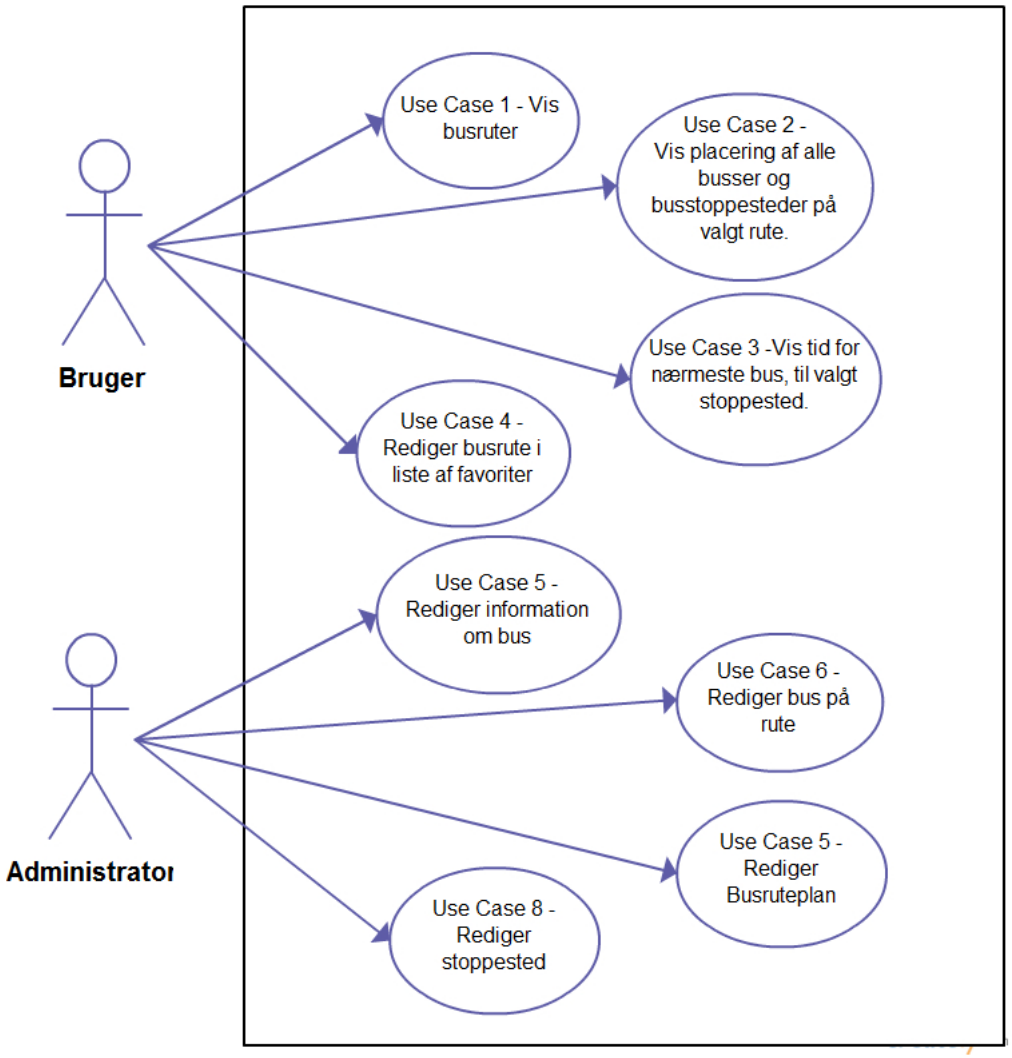
I dette afsnit forklares, hvordan Use Case view'et er sat op, samt hvad de forskellige Use Cases gør.

1.1 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. De beskrives senere i *afsnit 9.2 Implementering af persistens*
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

Use case diagram kan findes i bilag under Diagrammer/Use Case Diagram



Figur 1: Use Case diagram

Som det fremstår af Use Case diagrammet, figur 2, er der udelukkende to aktører; brugeren og administratoren. Selvom samtlige Use Cases kommunikerer med den distribuerede databasen, er det blevet vedtaget, at databasen blot er en interresant og ikke en sekundær aktør.

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 brugern, således at den grafiske brugergrænseflade for administratoren skal

udelukkende bruges af administratoren.

Brugeren skal kun tilgå mobil applikationen. Brugeren og administratorens Use Cases er derfor tæt koblet til deres respektive grafiske brugergrænseflade, da alle Use Cases initieres igennem disse. Eksempler på dette kan ses i *afsnit 3.4 Grænseflader til person aktører*

1.2 Use Case 1 scenarier - Vis busruter

1.2.1 Use Case mål

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

1.2.2 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, dernæst tilkendegiver brugeren overfor systemet at han ønsker at se listen over gemte busruter. Herefter hentes busruterne fra databasen, hvorefter brugeren bliver præsenteret for en liste af busruter.

1.2.3 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 brugeren kan annullere indlæsningen fra databasen. Hvis dette sker vil systemet stoppe indlæsningen fra databasen, samt returnerer til startskærmen. Der er mulighed for at systemet går i dvale, imens der indlæses fra databasen. Hvis dette sker vil systemet hente busruterne færdig i baggrunden.

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

1.3.1 Use Case mål

Målet med denne Use Case er at få vist et kort, med indtegnet busrute, busser der køre på valgt rute, samt busstoppestederne på ruten.

1.3.2 Use Case scenarier

Denne Use Case viser et kort til burgeren, med indtegnet busrute, alle busser der køre på ruten, samt alle stoppesteder på valgt rute. Det kræver at brugeren står ved listen over busruter, dernæst tilkendegiver brugeren overfor systemet hvilken busrute han ønsker vist. Derefter henter systemet busruten, samt stoppestederne på ruten fra databasen. Herefter bliver brugeren præsenteret for et kort, med indteget busrute samt stoppesteder. Systemet henter nu gps-koordinaterne for busserne på ruten samt indtegner dem på kortet. Efter 2 sekunder vil systemet igen hente gps-koordinaterne, og opdatere bussernes position på kortet. Systemet vil forsætte med at opdatere bussernes position indtil brugeren tilkendegiver overfor systemet at dette ikke længere ønskes.

1.3.3 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 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. Hvis det sker når systemet henter gps-koordinaterne vil brugeren blive præsenteret for en besked om at det ikke er muligt at opdatere bussernes position. Der vil stadigvæk være muligt at se kortet, med indtegnet rute, samt stoppesteder, bussernes position vil blot ikke opdateres. Det er mulighed for at systemet genetabler forbindelse til databasen, Hvis dette sker vil brugeren blive præsenteret for en besked om at det igen er muligt at opdatere bussernes position. Systemet vil forsætte med at opdatere bussernes position.

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

1.4.1 Use Case mål

Målet med denne Use Case er at få vist tid til ankomst, for den bus der er tættest på et valgt busstoppested.

1.4.2 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 en af busstoppestederne der er indtegnet på kortet. Systemet udregner nu den tid det vil tage, før den nærmestebus ankommer til det valgte stoppested, samt henter information om det valgte busstoppested fra databasen. Herefter bliver brugeren præsenteret for ankomstiden, samt information om valgt busstoppested.

1.4.3 Use Case Undtagelser

Da gps-koordinaterne, information om busstoppestedet samt udregningen for ankomsttiden til valgt busstoppested 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. Hvis gps-koordinaterne ikke kan hentes, vil bussens position blot ikke længere opdateres.

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

1.5.1 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.

1.5.2 Use Case scenarier

Før denne Use Case kan startes skal *Use Case 1: Vis busruter* være gennemført. Fra listen over alle busruter, tilkendegiver brugeren overfor systemet at han ønsker at favorisere et

busrute, eller fjerne en busrute fra favoriter. Ved favorisering af busrute, henter systemet den valgte busrute, samt stoppestederne for valgte busrute fra databasen, dernæst persisterer systemet dette på en sqlite database på telefonen. Busruten bliver markeret som favorit på listen over busruter. Brugeren vil nu kunne vælge den favoriseret busrute på startskærmen, i stedet for fra listen over alle busruter. ved fjernelse fra favorisering vil systemet slette ruten, samt dens busstoppesteder fra sqlite databasen, fjerne markeringen fra listen over busruter, samt det ikke længere vil være muligt at vælge ruten fra startskærmen.

1.5.3 Use Case Undtagelser

Da busruten samt busstoppestederne hentes 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.

1.6 Use Case 5 scenarier - Rediger information om bus

1.6.1 Use Case mål

Målet med denne Use Case er at rediger information om et bus i systemet. Dette indebære at kunne tilføje eller fjerne en bus fra systemet, samt blot at ændre i information om en bus der allerede eksistere i systemet.

1.6.2 Use Case scenarier

Denne Use Case har tre normalforløb, idet at man både kan tilføje en bus, fjerne en bus eller ændre i en eksistere bus. Det er kun en administrator der kan initialisere denne Use Case. Normalforløb 1 beskriver, hvordan en bus tilføjes til systemet. Dette forgår ved at administratoren tilkendegiver overfor systemet at han vil tilføje en bus til systemet. Herefter gør systemet det muligt at indtaste information om bussen. Når administratoren har indtastet det ønskede information, tilkendegiver administratoren at han ønsker at gemme informationen. Systemet gemmer nu informationen på databasen.

Normalforløb 2 beskriver hvordan administratoren ænder information om en bus der eksistere i systemet. Administratoren vælger en bus fra listen over alle busser i systemet.

Herefter tilkendegiver administratoren overfor systemet at han ønsker at ændre information om den valgte bus. Systemet gør det muligt for administratoren at ændre information om bussen. Når administratoren har indtastet det ønskede information, tilkendegiver administratoren at han ønsker at gemme informationen. Systemet gemmer nu informationen på databasen

I normalforløb 3 fjernes en bus. Denne initieres ved, at administratoren vælger en bus fra en liste over alle busser i systemet. herefter tilkendegiver administratoren overfor systemet at han ønsker at fjerne den valgte bus fra systemet. Systemet fjerner nu den valgte bus fra databasen.

1.6.3 Use Case Undtagelser

Da informationen om busserne skal både hentes og gemmes på 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 forbindelsetil databasen.

1.7 Use Case 6 scenarier - Rediger bus på rute

1.7.1 Use Case mål

Målet med denne Use Case er at kunne tilføje en bus til en valgt rute, eller fjerne en bus fra valgt rute.

1.7.2 Use Case scenarier

Denne Use Case har 2 normalforløb, idet at man både kan tilføje en bus til en rute, samt fjerne en bus fra en rute. Det er kun en administrator der kan initialisere denne Use Case. Normalforløb 1 beskriver hvordan en bus tilføjes til en busrute. Dette forgår ved at administratoren først vælger en busrute fra en liste over alle busrute i systemet. Herefter vælger administratoren en bus, fra en liste over alle busser i systemet, som ikke allerede er på en busrute. Administratoren tilkendegiver nu overfor systemet at han ønsker at tilføje valgt bus, til valgt busrute. Administratoren kan nu gemme ændringerne, hvis dette vælges, gemmes ændringerne på databasen. Normalforløb 2 beskriver hvorledes en bus fjernes fra en valgt busrute. Dette forgår ved at administratoren vælger en busrute, fra listen over

alle busrute i systemet. Herefter vælger administratoren en bus, fra listen over busser, der er på den valgte busrute. Administratoren tilkendegiver nu overfor systemet at den valgte bus ønskes fjernet fra valgt busrute. Administratoren kan nu gemme ændringerne, hvis dette vælges, gemmes ændringerne på databasen.

1.7.3 Use Case undtagelser

Da informationen om busser og ruter skal både hentes og gemmes på 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.

1.8 Use Case 7 scenarier - Rediger busruteplan

1.8.1 Use Case mål

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

1.8.2 Use Case scenarier

Denne Use Case har 3 normalforløb, idet det både er muligt at tilføje ny busrute til systemet, fjerne en busrute fra systemet, samt ændre i en busrute der findes i systemet. Det er kun en administrator der kan initialisere denne Use Case. Normalforløb 1 beskriver hvorledes der kan tilføjes en ny busrute til systemet. Dette forgår ved at administratoren tilkendegiver overfor systemet at han ønsker at oprette en ny busrute. Systemet præsentere nu administratoren for et kort. Administratoren kan nu indtegne en busrute på dette kort. Når den ønskede busrute er indtegnet på kortet, kan busruten gemmes på databasen, ved at brugeren tilkendegiver overfor systemet at busruten ønskes gemmes.

Normalforløb 2 beskriver hvordan administratoren kan ændre i en allerede eksiterende busrute. Dette forgår ved at administratoren vælger en busrute, fra listen over busruter der findes i systemet. Administratoren tilkendegiver nu overfor systemet at han ønsker at ændre i den valgte busrute. Systemet præsentere nu brugeren for et kort, med indtegnet busrute. Administratoren kan nu ændre busrute som ønskes. Ønskes ændringerne at gemmes, kan administratoren tilkendegive overfor systemetat dette ønskes, hvorpå systemet

vil gemme ændringerne i databasen.

Normalforløb 3 beskriver hvordan administratoren kan fjerne en allerede eksiterende busrute fra systemet. Dette forgår ved at administratoren vælger en busrute, fra listen over busruter der findes i systemet. Administratoren tilkendegiver nu overfor systemet at den valgte busrute ønskes slettes fra systemet. Systemet sletter busruten fra databasen.

1.8.3 Use Case undtagelser

Da ruten både skal gemmes på en database, samt hentes 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 forbindelset til databasen. Hvis administratoren ønsker at annullere processen efter at have fortaget ændringer vil administratoren blive præsenteret for en besked, der spørger om der ønskes at stoppe uden at gemme. hvis administratoren vælger at stoppe uden at gemme, retuneres til startskærmen.

1.9 Use Case 8 scenarier - Rediger stoppested

1.9.1 Use Case mål

Målet med denne Use Case er at kunne ændre i et busstoppested. Dette indebære at kunne lave et nyt stoppested, fjerne et stoppested, samt ændre i et eksisterende stoppested.

1.9.2 Use Case scenarier

Denne Use Case har 3 normalforløb, idet det både er muligt at tilføje nyt stoppested til systemet, fjerne et stoppested fra systemet, samt ændre i et stoppested der findes i systemet. Det er kun en administrator der kan initialisere denne Use Case. Normalforløb 1 beskriver hvorledes der kan tilføjes et nt stopepsted til systemet. Dette forgår ved at administratoren tilkendegiver overfor systemet at han ønsker at oprette et nyt stoppested. Systemet præsentere nu administratoren for et kort. Administratoren kan nu vælge placering af stoppestedet på kortet. Når placering af stoppested er valgt, kan stoppestedet gemmes på databasen, ved at brugeren tilkendegiver overfor systemet at stoppestedet ønskes gemt.

Normalforløb 2 beskriver hvordan administratoren kan ændre et allerede eksiterende

stoppested. Dette forgår ved at administratoren vælger et stoppested, fra listen over stoppesteder der findes i systemet. Administratoren kan nu ændre placering samt navn for det valgte stoppested. Ønskes ændringerne at gemmes, kan administratoren tilkendegive overfor systemetat dette ønskes, hvorpå systemet vil gemme ændringerne i databasen.

Normalforløb 3 beskriver hvordan administratoren kan fjerne et allerede eksiterende stoppested fra systemet. Dette forgår ved at administratoren vælger et stoppested, fra listen over stoppesteder der findes i systemet. Administratoren tilkendegiver nu overfor systemet at det valgte stoppested ønskes slettes fra systemet. Systemet sletter stoppestedet fra databasen.

1.9.3 Use Case undtagelser

Da stoppestedet både skal gemmes på en database, samt hentes 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 forbindelset til databasen.

2 PROCES/TASK VIEW

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

2.1 Oversigt over processer/task

Processen på android mobiltelefonen består af en række forskellige 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 tråede der bruges til at tilgå SoapProvideren.
updateTimeThread - Tråd der står for at opdatere tiden, til bus ankommer, på skærmen.
SetFavoriteBusRoute - Står for at gemme busrute som favorit i SQLite databasen.
RemoveFavorite - Står for at fjerne en favoriseret busrute fra SQLite databasen.

2.2 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 access layer til SoapProvideren der tilgår databasen, og retuner 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.

updateTimeThread snakker sammen med main tråden for at opdatere tiden på skærmen, bliver der brugt en handler, som er konfigureret til at sende beskeder til main tråden. Når tiden er udregnet, bliver der oprettet en ny runnable klasse der vil blive sendt som besked til main tråden, gennem handleren. Denne besked vil blive sat i kø, og eksekveret ved lejlighed.

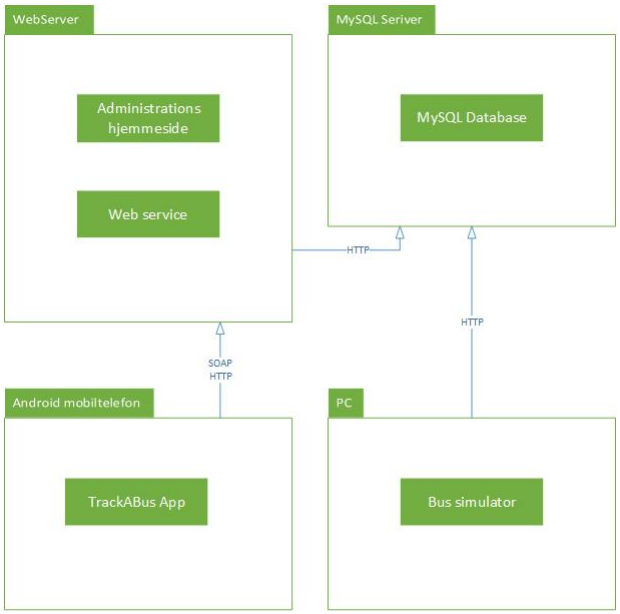
Både SetFavoriteBusRoute og RemoveFavorite bliver begge startet af main tråden, men da de blot skal indsætte og slette fra SQLite databasen, har de ikke behov for at kommunikerer med andre tråde.

Synkronisering

Trådene SetFavoriteBusRoute og RemoveFavorite er de eneste tråde 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, kun hente. SetFavoriteBusRoute skal skrive til SQLite databasen, dette 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.

3 DEPLOYMENT VIEW

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



Figur 2: Deployment Diagram

3.1 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 køre det enten Android 4.3 Jelly Bean eller Android 4.4 KitKat styresystem, ligesom det er muligt at udskrifte den PC hvorpå bus simulatoren køre, med en anden PC der overholder minimumskravene.

3.2 Node-beskrivelser

3.2.1 Konfigurering 1

3.2.2 Node 1. beskrivelse - Android mobil applikation

På denne enhed kørers TrackABus applikationen. Dette skal være en android mobiltelefon med enten android 4.3 Jelly Bean, eller android 4.4 KitKat styresystem. for at få mest ud

af applikationen skal der være mulighed for adgang til internet. Desuden kræver det ca. 4.1 MB ledig plads.

3.2.3 Node 2 beskrivelse - Webserver

Denne enhed hoster administrations hjemmesiden TrackABus.dk samt web servicen, som ligger på serveren nt21.unoeuro.com der er hostede af UnoEuro. Webserveren er en ASP/-ASP.NET server. Hjemmesiden er implementeret ved brug af ASP.NET MVC og Web servicen er en ASP.NET webservice.

3.2.4 Node 3 beskrivelse - MySQL Server

Denne enhed hoster MySQL serveren, som ligger på serveren mysql23.unoeuro.com der er hostede af UnoEuro. 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.

3.2.5 Node 4 beskrivelse - PC

På denne enhed køre GPSsimu.exe som indeholder bus simulatoren, der bruges til at simulere busser der køre på en busrute. Som minimum skal der være tale om en 64-bit windowsmaskine med windows 8 styresystem.

4 IMPLEMENTERINGS VIEW

4.1 Oversigt

Dette afsnit beskriver den endelige implementeringsopdeling af softwaren i lagdelte delsy-

stemer. Dette view specificerer opdelingen i det logiske. Alle bilag findes under Diagrammer og Billeder i fuld størrelse.

4.2 Komponentbeskrivelser

4.2.1 Komponent 4: 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 adgang til en databasen, da dette har store sikkerhedsmessige implikationer. Uden denne service vil det også være muligt for ondsindet brugere at tilgå databasen og manipulere med data på en ikke ønsket måde. Et andet formål med denne web service er at gøre det gøre det nemt at udvikle ny mobil applikation, til et hvilkensomhelst styresystem, uden at skulle tænkte på database tilgang.

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 Stored procedure der udregner tiden før der er en bus ved et valgt stoppested. 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. på http://trackabus.dk/AndroidToMySQLWebService.asmx er det muligt at se tilgængelige funktioner.

Her kan ses et eksempel på hvad det kræver at kalde funkionen GetBusPos fra web servicen, ved brug af SOAP. Det kan ses at den kræver et 'busNumber' i form af en string, som input parameter.

Kodeudsnit 1: 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>}
```

Herunder ses det SOAP response man får tilbage efter at have lavet det overstående kald til servicen. Det kan ses at man får en liste af points tilbage, hvor hver point indeholder en Lat, en Lng og et ID som alle er af datatypen string.

Kodeudsnit 2: response fra service function GetBusPos

```
1 HTTP/1.1 200 OK
2 Content-Type: application/soap+xml; charset=utf-8
3 Content-Length: length
4 <?xml version="1.0" encoding="utf-8"?>
5 <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">
6   <soap12:Body>
7     <GetbusPosResponse xmlns="http://TrackABus.dk/Webservice/">
8       <GetbusPosResult>
9         <Point>
10          <Lat>string</Lat>
11          <Lng>string</Lng>
12          <ID>string</ID>
13        </Point>
14        <Point>
15          <Lat>string</Lat>
16          <Lng>string</Lng>
17          <ID>string</ID>
18        </Point>
19      </GetbusPosResult>
20    </GetbusPosResponse>
21  </soap12:Body>
22 </soap12:Envelope>
```

4.2.2 Komponent 3: Administrations hjemmeside

Denne komponent har til formål at håndtere alle de adminitrative opgaver i system. Dette består af 4 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.

- Derefter skal det være muligt at tilføje eller fjerne en bus fra en rute der findes i systemet.
- Den tredje delkomponent gør det muligt at kunne oprette en hel ny busrute i systemet, ændrer i en allerede eksiterende busrute, eller slette en fra systemet.
- Den sisdte delkomponent består af muligheden for at kunne tilføje, ændre samt fjerne busstoppesteder fra systemet.

Alle disse delkomponenter udgøre tilsammen en vigtig del af systemet, da uden nogle af dem vil det ikke være muligt at kunne få vist nogle af overstående ting på mobil applikationen.

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 samt Google Directions API.

Hjemmesiden består af 4 view, hvor hver view har en controller knyttet til sig, først et view til startsiden der linker til de 3 andre views, der består af et der håndtere alt vedrørende busser, et til stoppesteder samt et til busruter.
Det første view der håndtere alt om busserne består af 2 dele. Første del gør det muligt at tilføje en ny bus til systemet, fjerne en bus fra systemet og rediger ID'et for en bus. Dette er blevet implementeret ved at når view'et bliver loaded, bliver en JavaScript function kalde, der kalder funktionen GetAllBusses() i controlleren, der henter alle busser der er i MySQL databasen. Til at lave dette kald fra JavaScript til controlleren, bliver der brugt ajax. Ajax gør det muligt at udveksle data med controlleren og updatere view'et uden at skulle reloade hele websiden.

Kodeudsnit 3: Ajax kald til controller funkionen 'GetAllBusses'

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

Dette eksemple på et ajax kald, kalder GetAllBusses(), dette er en funktion der ligger i Bus controlleren, som henter en liste af alle bussernes ID'er fra MySQL databasen. Se *afsnit 9.2.3 Implementering af persistens i online værktøjet* for nærmere beskrivelse af hvordan databasen bliver tilgået. Når controlleren er færdig retunere den et json object, og callback funkionen der er defineret i success parameteren af ajax bliver kaldt. Result parameteren på callback funktionen er returværdien fra controller funktionen, der i dette tilfælde er et json object, der indeholder en liste af alle bussernes ID'er, hentet fra MySQL databasen. Callback funktionen løber igennem listen af ID'er og tilføjer dem 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 ID'et på en bus.

For at tilføje en bus, skriver administratoren bussens ID ind i feltet: 'Bus ID' hvorefter han trykker på knappen 'Add'. Dette vil tilføje busses til listen, administratoren kan blive ved med at tilføje busser til listen. Administratoren kan også fjerne en bus fra listen, ved at vælge en bus i listen og trykke på knappen 'Remove', der er også mulighed for at ændre navnet for en bus, ved at vælge en bus, og trykker på 'Rename' knappen. Først ved tryk på 'Save' knappen vil ændringerne blive tilføjet til databasen. Dette sker igen gennem et ajax kald til controller, der kalder SaveBusChanges() funktionen. Denne funktion modtager listen af busser, med de nye busser administratoren har tilføjet, samt en liste af alle busserne på databasen. Funktionen sammenligner de 2 lister, finder de busser der er blevet tilføjet, de som er blevet fjernet og dem som har fået nyt ID. Efter

alt er fundet, vil den slette de relevante busser fra databasen og tilføje de nye busser.

Anden del af dette view gør det muligt at tilføje busser til en busrute og fjerne busser fra en busrute. Denne del består af 3 lister, hvor den ene indeholder alle busruter, hentet fra databasen, en der indeholder alle busser, der ikke er på nogle busruter 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 2 andre funktioner, dette forgår igen gennem 2 ajax kald til controlleren, den første henter navnene på alle busruter fra databasen, den anden henter en liste af ID'er for alle de busser der ikke er tilknyttet en rute endnu. Disse 2 ajax kald er magen til ajax kaldet vist i kodeudsnit: 3, den eneste forskel er hvilken controller funktion der bliver kaldt, samt hvilen HTML select element der bliver tilføjet til. Det er nu muligt for administratoren at vælge en af busruterne, fra listen. Dette vil trigger et 'onchange' event, der laver endnu et ajax kald til controller for at hente alle de busser der køre på den valgte rute, og vise dem i listen 'Busses on route'. Der kan nu tilføjes busser fra listen 'Avaliable busses' over til listen 'Busses on route' og 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.

Det næste view 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 og Google Directions API. For at få vist kortet på hjemmesiden, kræves det at kortet bliver initializeret. Først og fremmest skal man have lavet plads til det på siden.

Kodeudsnit 4: Div til google maps

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

Når HTML body elementet er loaded bliver dens OnLoad() event kaldt, dette kalder en JavaScript function, der initializere kortet samt Google directions service. Først bliver der

defineret en style, som kortet skal bruge, denne fjerner 'Points of interest'. Dernæst bliver der oprettet et mapOptions object, der definere forskellige options for kortet, her bliver kortets start position defineret til at vise Aarhus, kort typen bliver sat til ROADMAP, da dette vil vise kortet som et simplet vej kort. StreetViewControl bliver sat til false, da det er en feature der ikke er relevant for systemet.

Kodeudsnit 5: Map opsætning

```
var featureOpts = [{
  featureType: 'poi',
  stylers: [
    { visibility: 'off' }]
}];
var Aarhus = new google.maps.LatLng(56.155955, 10.205011);
var mapOptions = {
  zoom: 13,
  mapTypeId: google.maps.MapTypeId.ROADMAP,
  center: Aarhus,
  streetViewControl: false,
  styles: featureOpts
};
```

Efter at have defineret mapOptions bliver oprettet et map object. Dette object skal have det overnævte HTML map-canvas div element, samt mapOptions som constructor parameter.

Kodeudsnit 6: Map object init

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

kortet er nu blevet initialiseret og bliver vist på siden. Det næste der bliver initialiseret er Google direction renderer, dette bliver brugt til at vise en rute på kortet mellem 2 givet punkter. Først bliver der defineret de options som ruten skal bruge. Dette indebære om det skal være muligt at trække i ruten, for at ændre på den vej den skal tage, og om det skal være muligt at klikke på de markers der repræsentere start og slut punktet for ruten. Dette rendererOptions object bliver derefter brugt i constructoren for DirectionsRenderer, der laver et nyt DirectionsRenderer object der senere bliver brugt til at tegne ruten på

kortet.

Kodeudsnit 7: 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 direction renderen er blevet initialiseret, bliver der sat en listener på kortet der lytter efter om der bliver trykket på kortet.

Kodeudsnit 8: 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 listenern tjekke på om der er blevet trykket på kortet tidligere, hvis der ikke er, vil der blive plaseret en marker på kortet, der hvor der blev trykket, denne marker symbolisere der hvor busruten starter. Hvis der allerede er 1 marker

på kortet vil der bliver placeret endnu en marker, denne marker symbolisere der hvor busruten slutter. Efter begge markers er blevet placeret vil functionen calcRoute(startPoint, endPoint) blive kaldt. Denne function bruger Google direction service til at udregne en rute der går mellem 2 punkter. Dette bliver gjort ved først at lave et request object der definere start og slut GPS koordinaterne for ruten, disse koordinater bliver taget fra de 2 marker der er blevet placeret på kortet, samt hvilken Travelmode der skal bruges. Med TravelMode sat til DRIVING, vil der blive udregnet den rute der er hurtigts at tage med bil.

Kodeudsnit 9: 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 markerne har også en click listener på sig, dette skal bruges når en kompleks rute skal lave. dette bliver gjort ved at der bliver lavet endnu en DirectionsRenderer der laver en rute mellem enten start eller slut markeren, og et andet punkt på kortet hvor der er blevet trykket.

der er også blevet sat en listener på direction renderer'ne, der lytter efter om ruten ændre sig. I tilfælde af at ruten bliver ændret, vil listenerns callback function blive kaldt. Som det første, bliver der sat en kort delay, dette sker da ruten skal udregnes færdig før de forskellige properties der skal bruges, bliver sat. Efter dette delay bliver der itereret igennem alle properties, for at finde den af typen: 'Markers', der indeholder de markers de symboliser start og slut punkterne, samt alle de waypoints der må være blevet lavet på ruten, ved at administratoren har ændret på ruten. GPS koordinaterne for disse markers

vil blive brugt når hele ruten skal gemmes på databasen. Som det sidste i listenern vil information om ruten, der i blandt de GPS koordinater der bliver brugt til at tegne ruten, gemt i en variable der senere bliver brugt når ruten skal gemmes på MySQL databasen.

Kodeudsnit 10: directions renderer 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 stoppesteder på en rute, kaldes funktionen SetBusStopsOnMap(), der henter navnene på de stoppesteder der er i listen 'ToLB'. Efter alle navnene er hentet, bliver der lavet et ajax kald til controllerens GetLatLng() funktion der bruger stopnavnene til at hente GPS koordinaterne for hver stop i MySQL databasen. Disse koordinater bliver sendt tilbage til ajax callback funktionen som et Json object, der bliver brugt til at tegne stoppestederne ind på kortet ved brug af markers.

Kodeudsnit 11: 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 }
```

SaveRouteAndStops() funktionen, står for at samle data der skal gemmes i databasen, og sende det til Save funktionen i Dir controlleren gennem et ajax kald. Først finder den alle de GPS koordinater der bruges til at tegne ruten, hertil bliver funktionen getRoutePath() brugt. her bliver den tidligere defineret temp variable, der holder information om ruten, brugt. For at få GPS-koordinaterne for de punkter der bliver brugt til at tegne ruten, skal hver path, for hver step, for hver leg findes. Når disse er fundet, retuneres en liste af GPS koordinater. For mere information om path, step og legs refereres til Google maps api doc Dernæst bliver der fundet de waypoints og stoppesteder der ligger på ruten. Waypoints skal bruges til at genskabe ruten på hjemmesiden. Til sidste bliver der fundet de ruter der udgøre en kompleks rute, hvis dette er blevet tilføjet. controlleren kalder CalculateBusStopsForRoute() funktionen der laver udregninger på mellem hvilke rute punkter stoppestedet skal ligge, *se afsnit 8.2.5 Anvendt matematik* for dybere beskrivelse af udregningerne der bliver fortaget. Efter alle udregnigner er fortaget, bliver alt data indsat i MySQL databasen *Se afsnit 9.2.3 Implementering af persistens i online værktøjet* for hvordan databasen bliver tilgået.

Det sidste view omhandler funktionerne for at tilføje, nye stoppesteder, ændre position og navn for eksiterende stoppesteder, samt slette dem fra databasen. For at kunne oprette nye stoppesteder, bliver der igen brugt Google maps API, 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, bliver kortets listener event kaldt, ved tryk på kortet, dette event vil sætte en marker på

kortet, der hvor der blev trykket, som symbolisere stoppestedets position.

Kodeudsnit 12: 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 });
```

SaveStopsToDB() funktionen kan nu kaldes, denne vil lave et ajax kald til stop constrolleren, med GPS koordinaterne og navnet på stoppestedet, der vil gemme det på MySQL databasen.

4.2.3 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 vil beskrives og der vil forklares, hvorefter det forklares 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.

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

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

$$d = R * c * 1000 \tag{3}$$

atan2 returner et grad værdi mellem -180°til 180°. Det er dog ikke nødvendigt at tage højde for dette i denne sammenhæng, da det ikke er nødvendigt at se på graderne som en helcirkel fra 0°til 360°.
Hjemmesiden, simulatoren og den distribuerede database tager alle brug af Haversine formlen.

- I databasen bruges den i udregningen for en bus tid til ankomst 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 Haversine formlen, og ligges sammen. Resultatet vil være den søgte afstand.

- I simulatoren bruges den når bussens næste position skal udregnes. Der er fundet, hvor langt bussens skal køre, og der undersøges så mellem hvilke to rutepunkter, bussens skal være. Dette gøres ved at udregne afstanden mellem rutepunkter ved hjælp af Haversine formlen, finde ud af ved hvilket punkt afstanden er større end den bussens skal køre, hvorefter bussens nye position før dette punkt findes.
- På hjemmesiden bruges den når det skal udregnes, hvilke to rutepunkter et stoppested skal ligge mellem. I denne situation skal det undersøges om stoppestedet ligger en hvis afstand væk fra et punkt. Her til bruges Haversine til at udregne distancen.

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 objekter 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.

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

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 hvis distance udfra det første punkt, med 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.

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

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

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

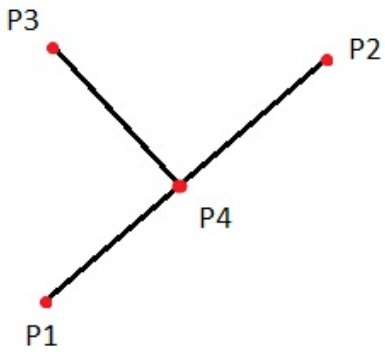
$$\lambda_o = \lambda_p + atan2(x, y) \quad (11)$$

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 antalgrader, skal denne værdi konvertes til en 360°version. 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 3, 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 3: Situationen 1: Tætteste punkt på en linje

A:	Linjestykket mellem P1 og P2 hældningskoefficient.
B:	Linjestykket mellem P1 og P2 skæring med y-aksen.
$\theta_1, \theta_2, \theta_4, \theta_4$:	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 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.

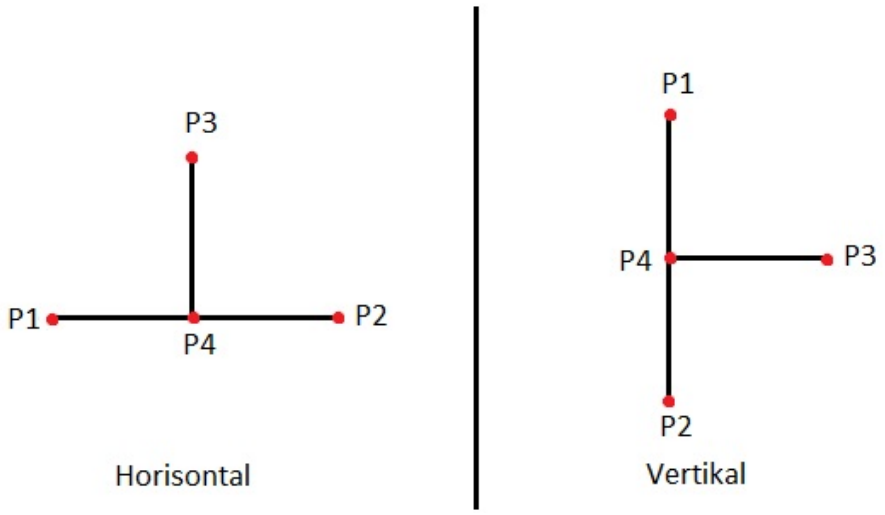
$$A = \frac{\theta_2 - \theta_1}{\lambda_2 - \lambda_1} \tag{12}$$

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

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

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

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 4



Figur 4: Situationen 2 og 3: Horisontalt eller vertikalt linjestykke

Denne formel tages i brug i databasen og på hjemmesiden.

- I database tages denne udregning i brug når en bus tid til ankomst 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 linje der bliver skabt af et andet linjestykke vil have en mindre ortogonal distance hen til objektet. Derfor er det vigtigt, at der ved implementering tages højde for, at det kun er 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.

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 en lineær funktion findes 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

Nå 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.

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

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

5 DATA VIEW

5.1 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 og til administrationshjemmesiden er et domænenavn blevet købt hos `www.unoeuro.com`, ved navn `www.trackabus.dk`. Herunde 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

5.1.1 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.

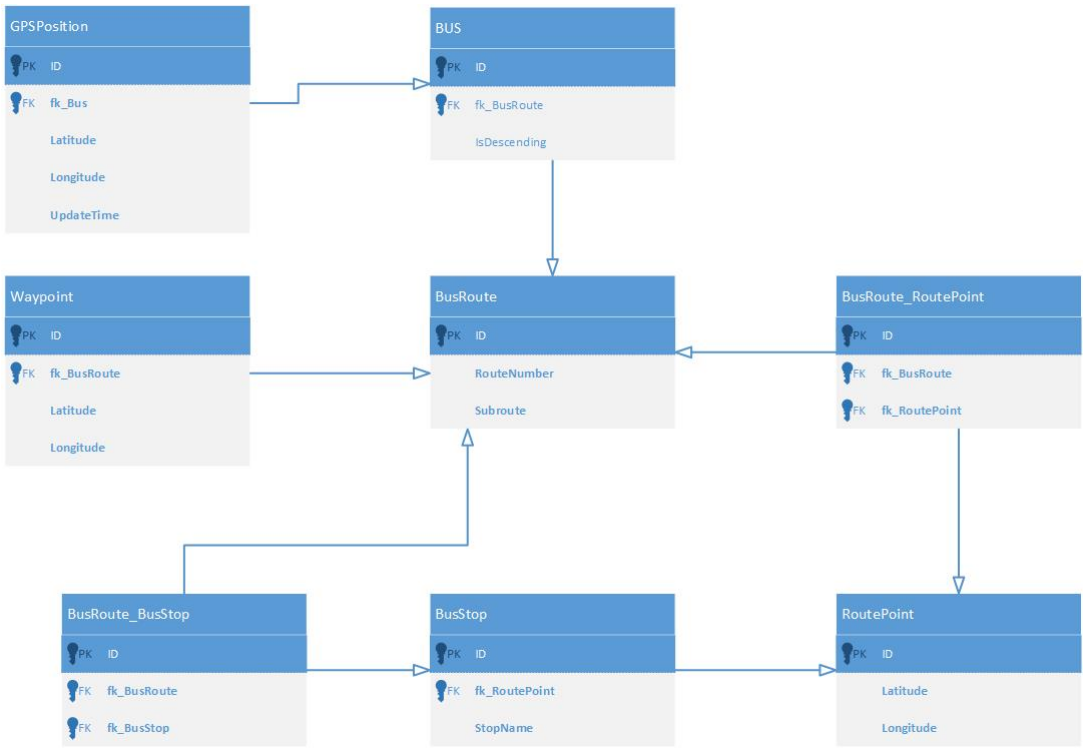
- Busposition er defineret som den fysiske placering af en given bus. I dette projekt var der dog ikke tilgang til nogen fysiske busser, så denne katagori af positions data blev simuleret. Simulatoren kunne dog skiftes ud med en virkelig bus, hvis position for denne kunne stilles til rådighed.

- Rutepunter og stoppesteder indeholder positionsdata, som bruges til at tegne ruten eller lave udregning på. Disse udregninger er defineret senere under "Stored procedures"og "Functions".

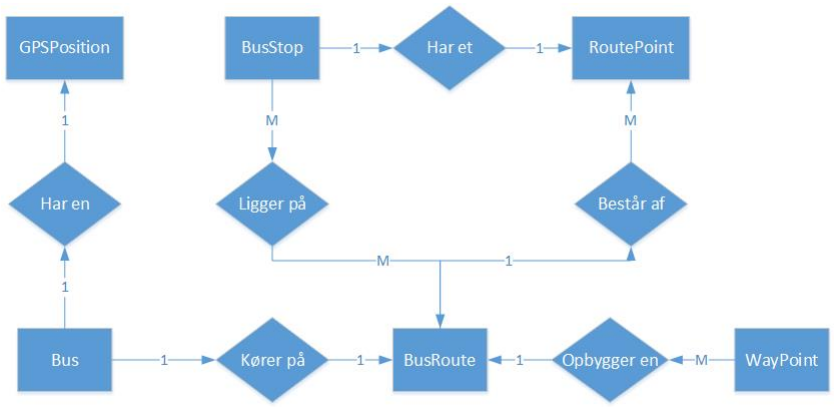
- Waypoints bruges som "genskabelses-punkter"til en given rute. Disse punkter bliver udelukkende brugt af administrationsværktøjet, til at genskabe den rute de beskriver.

Hele systemet er opbygget omkring oprettelse, fjernelse og manipulation af positions data. Dette er klart afspejlet i database i form hvor meget dette data bliver brugt.
Tidligt i udviklingsprocessen blev det fastsat at positions data 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, Waypoint.
På figur 5 vises opbygningen af tabellerne som et UML OO diagram, og på figur 6 kan relationerne i databasen ses som et ER diagram.



Figur 5: UML OO diagram over den distribuerede MySQL database



Figur 6: 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 fysisk bus.

- **BusRoute**
  - Indeholder detaljer omkring Busruten foruden dens rutepunkter. BusNumber er ikke nødvendigvis unikt, 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 10 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 delrute på den givne rute. Ruter er i denne sammenhæng defineret som turen

mellem to endestationer, og hvis en rute har mere end to endestation, 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 IDet i denne tabel og autogenereret, men bruges til at definere rækkefølgen på punkterne, som ruten bliver opbygget af. fk_BusRoute er foreign key til IDet for busruten, og fk_RoutePoint er foreign key til IDet 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 lagt ind i listen ved hjælp af en forklaret i afsnittet "IMPLEMENTERING: ADMINISTRATOR SIDE".
- **BusRoute_BusStop**
  - Indeholder stoppestedsplanen for det givne rutenummer. IDet i denne tabel er autogeneret, men bruges til at definere rækkefølgen på 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 IDet i denne tabel og er autogeneret. StopName er navenet på det givne stoppested, og er en varchar på 100 karakterer.
    fk_RoutePoint er en foreign key til IDet 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 position på verdenskortet. Longitude og latitude er i denne sammenhæng længde- og breddegraden, og de er defineret ved en number med 15 decimaler. Alle 15

decimaler er ikke nødvendig i brug og ved en indsættelse af et tal på f.eks. 6 decimaler, vil de sidste 9 være sat til 0.

- **GPSPosition**
  - Indeholder alle kørende bussers position. 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. 6 decimaler, vil de sidste 9 være sat til 0. UpdateTime er et timestamp for positionen og bruges til, at udregne hvor lang tid bussen har kørt. Dette er beskrevet nærmere i afsnittene "Stored procedures"og "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 IDet og autogeneret. Den Bruges ikke ti andet en 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. 6 decimaler, vil de sidste 9 være sat til 0. fk_BusRoute er en foreign key til BusRoute tabellen, og definerer således hvilken Busrute det givne waypoint er relateret til.

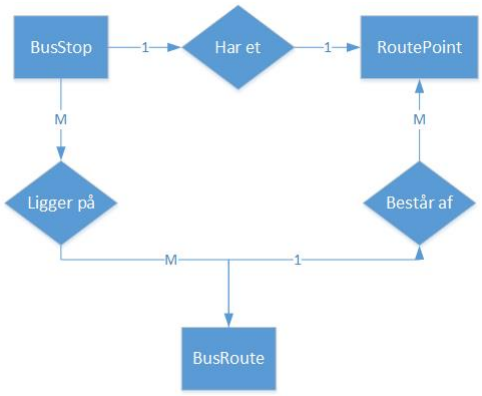
Normalform

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 ingen ikke keys afhængig af kun en del af nøglen
- Ingen elementer er afhængigt af et ikke-nøgle element. Dvs. ingen kolonner i én tabel, definerer andre kolonner i samme tabel.

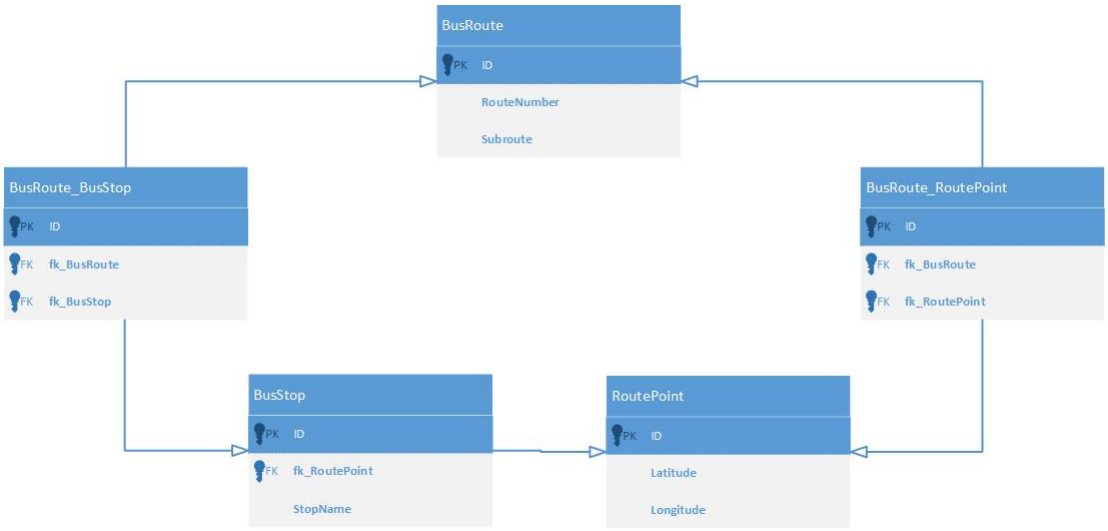
5.1.2 Design af SQLiteDatabase database

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 7 kan man se et UML OO diagram over den lokale SQLite database og på figur 8 kan man se et ER diagram over samme database. Da den lokale database blot er et udsnit



Figur 7: UML OO diagram over den lokale SQLite database

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 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.



Figur 8: UML OO diagram over den lokale SQLite database

5.1.3 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, givet sammen med et kodeudsnit. I kodeudsnittet vil ingen kommentarer være tilstede. For fuld kode henvises der til bilags CDen, i filen Stored Procedures under Kode/Database.

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 tabellen 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 tables, gør det tilgangen trådsikker. Dette betyder at proceduren godt kan tilgås fra flere enheder på samme tid.

CalcBusToStopTime

Denne Stored procedure er kernen i tidsudregningen. Den samler alle værdierne sender dem videre i de forskellige funktioner. På kodeudsnit 13 ses et udsnit af proceduren. I den fulde procedure, vil udregningerne for begge retninger hen til 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 13: 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 valgt stop, 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.

Først findes mulige ruter fra givet stoppesteds navn og rutenummer og indlægges i en Dette er nødvendigt i tilfælde af komplekse ruter, hvor mere end en rute kan have samme stoppested og rutenummer. Herefter kaldes den anden stored procedure, som beskrives senere i dette afsnit. Denne procedure returnerer tætteste rutepunkt, IDet 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 senere 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 IDet.

GetClosestBusAscProc og GetClosestBusDescProc Da proceduren for begge retninger er meget ens, vil der kun vises et kodeudsnit for GetClosestBusAscProc. Dette kan ses på kodeudnsit 14.

Alle kommentarer og 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 14: 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 fra fundet i forrige procedure. 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 (bussen kører fra første til sidste stoppested) udtages. Disse busser bliver parret med det ID stoppestedet har, i *BusRoute_RoutePoint* tabellen og et auto-inkrementeret ID startende fra 1, og lagt ind i *BussesOnRouteAsc* tabellen. Herefter findes det antal af busser, der er blevet udtaget, og dette tal bruges til den øvre grænse for while-loopet. Den nedre grænse er blot en counter som sættes til 1 ved initiering.

While-loopets rolle er, at iterere igennem samtlige busser, og udregne distancen fra hver bus til dens parrede stoppested, hvorefter at vælge 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 tallet 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 sættes 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. IDet, samt det tætteste rute-

punkt, for denne bus vil også sættes i denne situation. Til sidst vil den korteste distance, det tætteste rutepunkt samt IDet 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 15, kan den første ændring 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 15: 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 16, kan den anden ændring ses. Hvis en bus kører fra første til sidste stoppested, vil det nærmeste rutepunkt til bussen, have et større ID end stoppestedet, 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 ligemed.

Kodeudsnit 16: GetClosestBusDescProc forskel 2

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

5.1.4 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 **IMPLEMENTATION-MATEMATIK**, for beskrivelser af disse. Som i *Stored Procedures*-afsnittet, 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. For fulde kodeudsnit henvises der til bilags CDen i filen Functions under Kode/Database.

GetClosestEndpointAsc og GetClosestEndpointDesc

Disse funktioner tages i brug i *GetClosestBusAscProc*- og *GetClosestBusDescProc* procedurene, og bruges til at finde IDet for det rutepunkt, en given bus er tættest på. Dette ID er dog ikke rutepunktet 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 17 kan *GetClosestEndpointAsc*-funktionen ses. Den er givet uden kommentarer eller initialisering af variabler.

Kodeudsnit 17: 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 tabel. I denne tabel gemmes længde- og breddegrader, sammen med 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. Dette gør at punkterne kan itereres igennem, uden at der skal tages højde for spring, og kan sorteres efter ID i den rækkefølge man skal bruge (ascending for første til sidste stoppested, des-

cending for sidste til første stoppested). Det er meget sandsynligt at det første ID hentet ikke er et, Så det første og sidste punkt på ruten findes også, og bruges som den nedre og øvre grænse for while-lykken. På den måde vil der itereres igennem samtlige punkter på ruten, hvor IDet 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 *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, og afstanden fra bussen til dens tætteste punkt på dette linjestyke, udregnes i *CalcRouteLineDist*. Denne funktion er udelukkende matematisk og vil beskrives i **IMPLEMENTATION-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 IDet for dette punkt returneres.

I *GetClosestEndpointDesc* er der nogle enkelte forskelle, som her vil beskrives. På kodeudsnit 18, kan det ses hvordan ruten nu hentes ud i en tabel, hvor der sorteres efter IDet i *BusRoute_ RoutePoint* tabellen, i faldende rækkefølge.

Kodeudsnit 18: 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 19 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 IDerne nu er faldende. Det vil også sige, at *RouteCounter* dekrementeres i stedet for inkrementeres i slutningen af hver iteration.

Kodeudsnit 19: 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, IDet for bussens tætteste rutepunkt fra forrige funktioner, samt ID et på stoppestedet. Bemærk at disse IDer er hentet fra *BusRoute_RoutePoint* tabellerne og symboliserer derfor rutepunktet og stoppestedets placering på ruten, og ikke deres egentlige IDer i *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 for *CalcRouteLengthAsc*, hvorefter funktionen forklares i detajler. Til sidst forklares forskellene mellem *CalcRouteLengthAsc* og *CalcRouteLengthDesc* mere detaljeret. På kodeudsnit

?? kan funktionen ses uden kommentarer eller initialiseringer uden værdi. Dette gøres for at bevare det funktionelle overblik.

Kodeudsnit 20: 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 dette ID 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. Denne funktion finder afstanden mellem to koordinater i fugleflugt. Funktionen vil ikke beskrives videre i dette afsnit, for mere information henvises der til afsnittet **IMPLEMENTERING-MATEMATIK**. Herefter itereres der igennem ruten, hvor IDet for det tætteste rutepunkt på bussen er den nedre

grænse, og IDet 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.

I *CalcRouteLengthAsc* gøres der brug af *ChosenRouteDesc* tabellen i stedet for *ChosenRouteAsc*. Desuden bruges IDet for stoppestedet nu som den nedre grænse og det tætteste rutepunkt som den øvre, og *RouteCounter* dekrementeres i stedet. Dette kan ses på kodeudsnit **??**

Kodeudsnit 21: 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 hvor lang tid der er tilbage, før bussen når stoppestedet.Funktionen modtager IDet på et bus som den eneste parameter.

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. På kodeudsnit **??** kan funktionen ses uden kommentarer eller initialiseringer uden værdi. Dette gøres for at bevare det funktionelle overblik. Efter udsnittet forklares funktion detaljeret.

Kodeudsnit 22: CalcBusAvgSpeed. Udregner gennemsnitshastigheden for en bus. label

```
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 $$
```

Første udhentes alle GPS position og opdaterings tiderne 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 et, og bruges som den nedre grænse. Ved hver iteration hentes den opdatering af bussens position, hvis 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. Den totale afstand inkrementeres med den udregne 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 **IMPLEMENETERING-MATEMATIK**.

5.2 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 i mobilapplikation, Implementering i simulator og Implementering i online værktøjer. 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.

5.2.1 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 distrubuerede database. Tilgang sker i afsnittet *Implementering af online værktøjet* i underafsnitet *Mobilservice*.

Applikation kommunikerer med den 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 databasen tilgangen skal kunne gemmes væk fra brugeren, således en en person ikke kan få fuld tilgang til databasen igennem sin mobil.

Selve kommunikationen med servicen sker igennem en SoapProvider. SOAP står for Simple Object Access Protocol, og bruges som et 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 modtager og parametre navnene. På kodeudsnit 23 kan en generisk oprettelse og transmitering af en SoapEnvelope ses.

Kodeudsnit 23: 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 simple strings. Til metodekaldet kan der tilføjes parametre ved addProperty metode, som tager imod et parameter navn og en parameter værdi, begge to strings. Envelopen bliver oprettet og en versionsnummer bliver givet med, der definerer hvilken version af protokollen der skal tages i brug. I vores projekt har vi udelukkende gjort brug af version 1.1. dotNet flaget er sat til true, da vores service 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 fås ved et kald til getResponse 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 *8.2.4 Komponent: Webservice*

Et fuldt eksempel på et kald til servicen kan ses på kodeudsnit 24. Denne funktion bruges til at hente samtlige busser med et givent busnummer, og returnere dem som en ArrayList. Vil alt efter SoapObject responset forklares.

Kodeudsnit 24: 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() 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 returnerer latitude og GetProperty(1) returnerer longitude. Begge bliver castet til en string, og decimalpoint sættes til et dot frem for et komma. Dette gøres da ASP.NET tager et decimalpoint som værende komma.

Da applikationen er lavet til Android bruges biblioteket ksoap2, som er specifik for android. I dette bibliotek ligger alle funktioner, der er nødvendige for brug af SOAP. For mere information om protokolen henvises der til afsnittet: "REFERENCE"under SOAP.

Tilgang til SQLite databasen

Når en busrute favoriseres gemmes alt data om denne i en lokal SQLite database. Dette gøres for at spare datatrafik for ruter, som brugeren ville tage i brug ofte. Samtidig muligøres det også, at brugeren kan indlæse en rute med stoppestedder, uden at have internet. 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-access laget, så flere applikationer kan tilgå databasen, med den samme protokol, hvis det skulle være nødvendigt.

En ContentProvidere 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 er den samme værdi som navnet.
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 authority (ContentProviderens placering i projektet) 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 25 ses et eksempl på, hvordan dette er implementeret i systemet. Det skal noteres at dette ikke er komplet implementering, men blot et udsnit. Kommentarer vises ved "!!"

Kodeudsnit 25: 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 URImatcher.match, vil den returner en værdi, der svarer til den, den er blevet givet ved oprettelse. Herefter ses et eksempel på query-metoden.

Hvis URIet 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 URIet, vælges BUSSTOP_NUM _CONTEXT, og der hentes kun det stoppested som har det ID sat i URIet.
Til samtlige tabeller i SQLite databasen er der lavet en model klasse. Disse klasser indeholder kun statiske variabler. Disse definerer den givne tabels kolonner samt den URI ContentProvideren skal have med for at 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 IDet er "RouteID". Det andet kald tilgår også query funktionen men under BUSSTOP_NUM_CONTEXT, og henter stoppestedet ud hvor IDet er "StopID".

5.2.2 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. Da simulatoren er lavet i Visual Studio 2012, og er en WPF-applikation, er der blot gjort brug af NuGet til at hente og tilføje dette library til programmet. Forbindelsesopsætningen ligger i App.config filen, og hentes ud når der skal bruges en ny forbindelse. På kodeudsnit **??** 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.

```
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);
```

Som det ses, ventes der 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 færdigm 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 skulle returneres noget data. Readeren repræsenterer en række, og når Read() bliver kaldt på den, læser den næste række. Hvis Read() returner false, er der ikke flere rækker at læse. Når data skal hentes ud fra reader, kan man enten vælge 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 IDer. Herefter oprettes der en liste af de kolonner der skal hentes hvorefter Query funktionen kaldes med begge værdier.

5.2.3 Implementering af persistens i online værktøjet

Online værktøjet er todelt i mobil service og hjemmeside. 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 tidligere i dette afsnit.
Servicen står i midlertid også for at kalde tidsudregnings proceduren på databasen, hvilket er et anderledes kald, end en læsning. På kodeudsnit **??** 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.

```
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));                    cmd.Parameters["?↩
    TimeToStopSecAsc"].Direction = System.Data.ParameterDirection.↩
    Output;
10 ...
11 cmd.ExecuteNonQuery();
12 ...
13 string TimeToStopAsc = cmd.Parameters["?TimeToStopSecAsc"].Value.↩
    ToString();
14 string EndStopAsc = cmd.Parameters["?EndBusStopAsc"].Value.↩
    ToString();
```

I kodeudsnittet kan det ses, hvordan der i kodeudsnittet, i forrige afsnit, blev tilføjet en kommandotext bestående af en MySQL string, nu bliver tilføjet flere værdier til kommandoen. 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. Parameterne bliver givet et navn, samt en datatype, hvorefter de gives en værdi hvis de er input parametre. Herefter gives parameteren 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. I afsnittet *9.1.2: Stored Procedures* kan der læses om trådsikkerheden for proceduren.

Da databasetilgangen på hjemmesiden kun er flertrådet når der læses, er systemet trådsikkert. Når der læses vil det altid ske i hovedtråden. Databasen tilgås ligesom servicen og simulatoren ved hjælp af MySql.Data bibilioteket, og tilgås kun i form af simple CRUD-operationer. Der vil derfor ikke vises et kodeeksempel, det dette anses som værende beskrevet i tidligere afsnit.