TrackABus

BACHELORPROJEKT

Systemarkitektur for TrackABus

Author:

Gruppe 13038

Supervisor:

Michael Alrøe

5. december 2013

Versionshistorie:

Ver.	Dato	Initialer	Beskrivelse
1.0	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:	36
Kunde:	Michael Alrøe (MA).

Sted og dato: _____

10832 Christoffer Lousdahl Werge

MA Michael Alrøe

09421 Lasse Lindsted Sørensen

Indhold

1 USE CASE VIEW

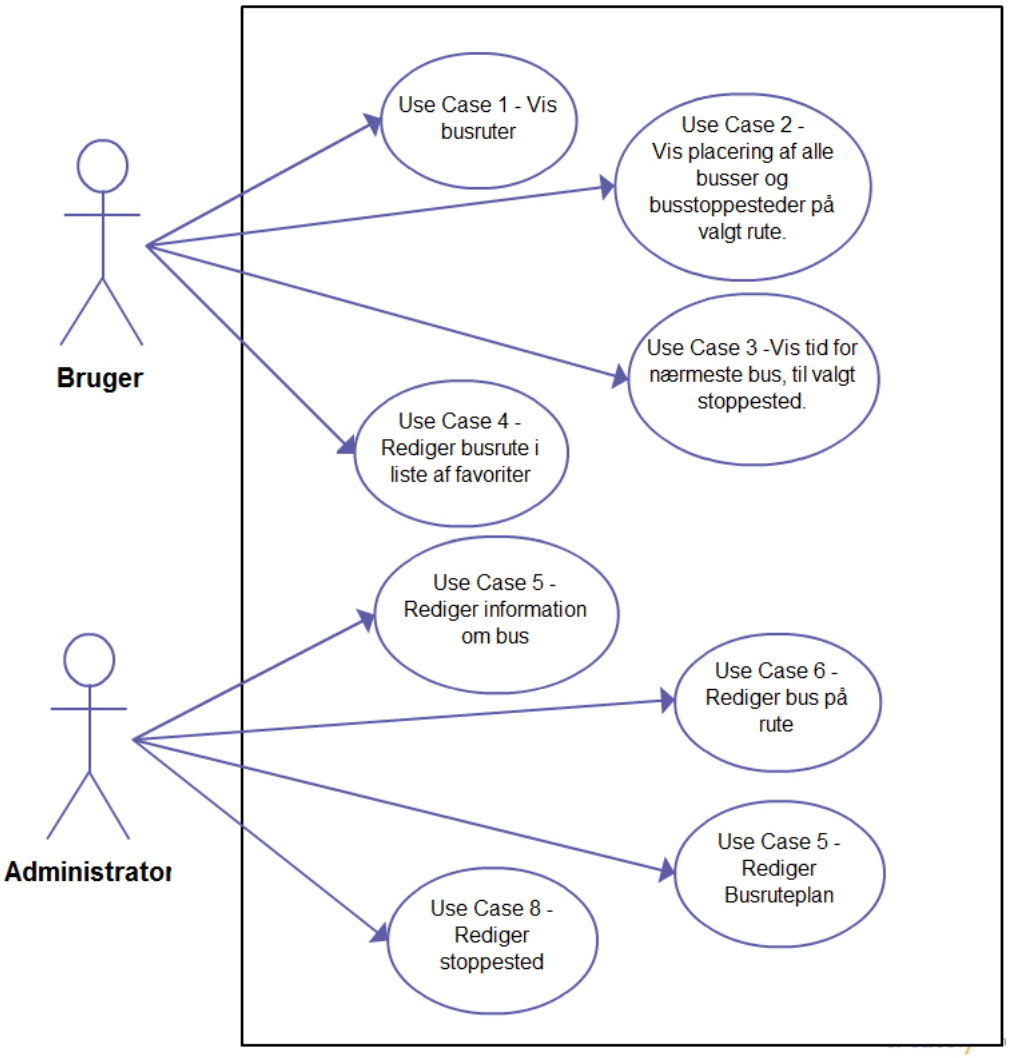
I dette afsnit forklares, hvordan Use Case view'et er sat op, samt hvad de forskellige Use Cases gør.

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.

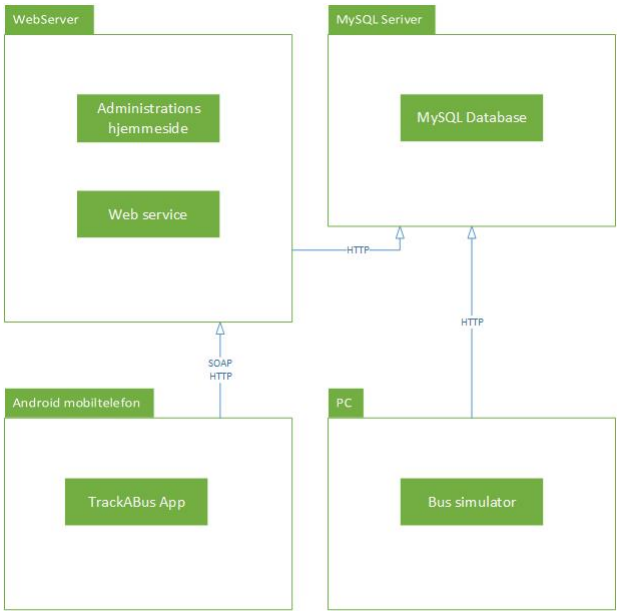
Normalforløb 3 beskriver hvordan administratoren kan fjerne et allerede eksiterende stoppested fra systemet. Dette forgår ved at administratoren vælger et stoppested, fra listen over stoppesteder der findes i systemet. Administratoren tilkendegiver nu overfor systemet at det valgte stoppested ønskes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

2.1 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.

2.2 Node-beskrivelser

2.2.1 Konfigurering 1

2.2.2 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.

2.2.3 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.

2.2.4 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.

2.2.5 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.

3 IMPLEMENTERINGS VIEW

3.1 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.

3.2 Komponentbeskrivelser

3.2.1 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,først og fremmest 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

3.2.2 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 3 strings, Lat, Lng og ID.

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 DATA VIEW

4.1 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

4.1.1 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.

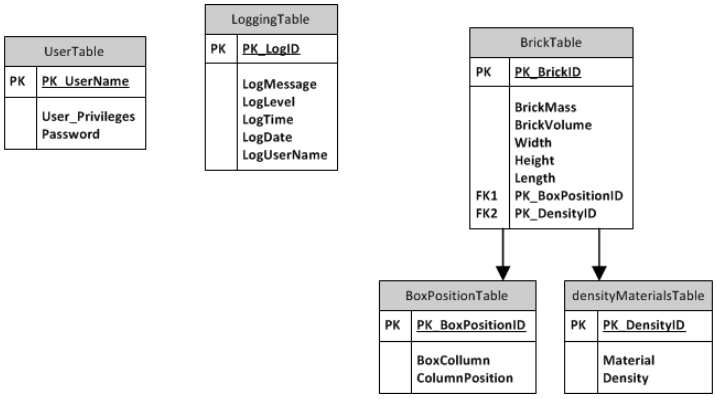
- Busposition er defineret som den fysiske placering af en given bus. I dette projekt var der dog ikke tilgang til nogen fysiske busser, så denne katagori af positions data blev simuleret. Simulatoren kunne dog skiftes ud med en virkelig bus, hvis position for denne kunne stilles til rådighed.

- 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".

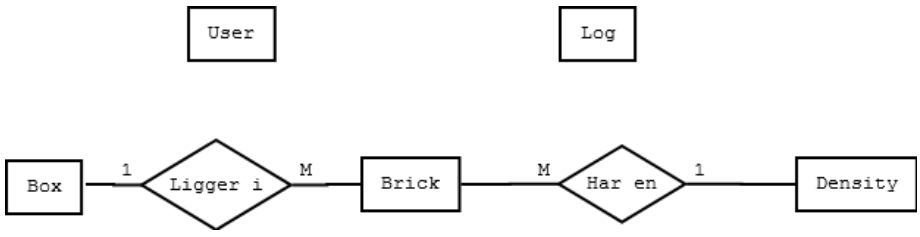
- Waypoints bruges som "genskabelses-punkter"til en given rute. Disse punkter bliver udelukkende brugt af administrationsværktøjet, til at genskabe den rute de beskriver.

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 3 vises opbygningen af tabellerne som et UML OO diagram, og på figur 4 kan relationerne i databasen ses som et ER diagram.



Figur 3: UML OO diagram over den distribuerede MySQL database



Figur 4: 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.

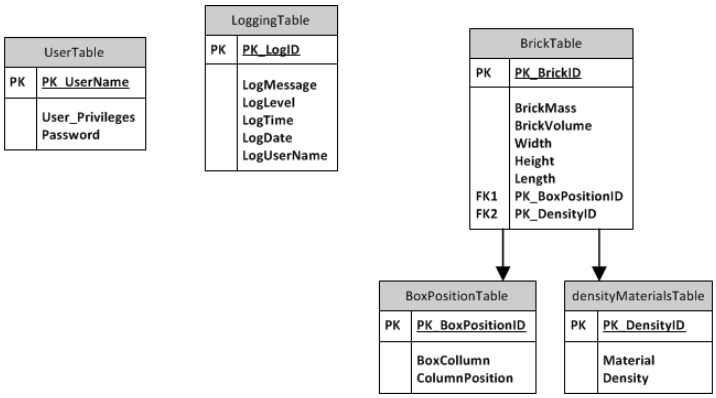
Normalform

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:

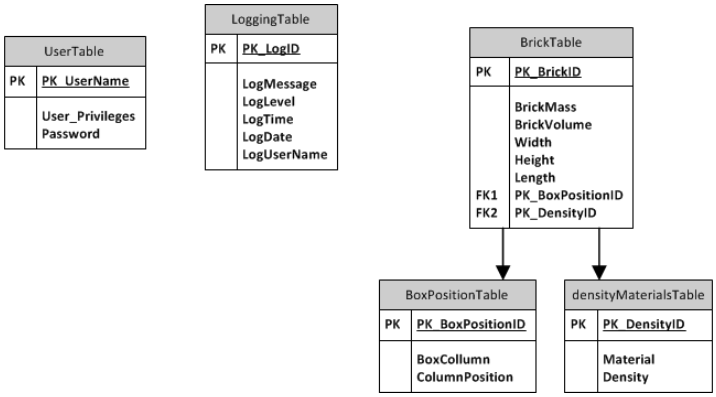
- Ingen elementer er i sig selv elementer. Dvs. ingen kolonner gentager sig selv.
- Ingen primary keys er composite keys, og derfor er ingen ikke keys afhængig af kun en del af nøglen
- Ingen elementer er afhængigt af et ikke-nøgle element. Dvs. ingen kolonner i én tabel, definerer andre kolonner i samme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

4.1.2 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 5 kan man se et UML OO diagram over den lokale SQLite database og på figur 6 kan man se et ER diagram over samme database.



Figur 5: UML OO diagram over den lokale SQLite database



Figur 6: UML OO diagram over den lokale SQLite database

Da den lokale database blot er et udsnit af den distribuerede MySQL database, henvises der til tabel beskrivelserne for MySQL tabellerne i forrige afsnit. Databasen er derfor også på tredje normalform, som MySQL databasen.

Den eneste forskel fra MySQL databasen er, at denne tabel gør brug af Delete Cascades. Dette vil sige, at sletningen af data fra SQLite databasen kun kræver at man sletter fra 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.

4.1.3 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 3 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 3: 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 4.
Alle kommentarer og deklareringer er fjernet for at give et bedre overblik over funktionalitetet af proceduren. En detaljeret forklaring, samt foreskellene mellem GetClosestBusAscProc og GetClosestBusDescProc, følger efter kodeudsnittet.

Kodeudsnit 4: 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 rutepunkt, 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 foreskelle.

Kodeudsnit 5: GetClosestBusDescProc foreskel 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 5, 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 6: GetClosestBusDescProc foreskel 2

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

På kodeudsnit 6, 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.

4.1.4 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 matematikfunktioner. 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 foreskellene 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 Denne funktion tages i brug i *GetClosestBusAscProc*-procedure, 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 7 kan *GetClosestEndpointAsc*-funktionen ses. Den er givet uden kommentarer eller initialiseringer af variable.

Kodeudsnit 7: 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, descending 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 foreskelle, som her vil beskrives. På kodeudsnit 8, kan det ses hvordan ruten nu hentes ud i en tabel, hvor der sorteres efter IDet

i *BusRoute_RoutePoint* tabellen, i faldende rækkefølge.

Kodeudsnit 8: GetClosestEndpointDesc foreskel 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 9 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 9: GetClosestEndpointDesc foreskel 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 ...
```

4.2 Implementering af persistens

Persisteringen af data består af to hoveddele. En facade til den direkte tilgang og en besked kø der lavet efter observer design mønsteret som bliver beskrevet i implementeringsviewet, afsnit 8.2.7-Observer Pattern

For at tilgå databasen på C# siden er der blevet brugt facade designmønsteret. Tilgangen til databasen sker altså igennem én klasse, så vi bevarer lav kobling. Til selve forbindelsen blev der brugt SQLCommands. Dette er en del af .Net frameworket. Forbindelsen ligger hardcodet i facaden, dvs. at det ikke er op til brugeren af systemet, at definere hvilken database man tilgår. Denne forbindelse består af en streng med bruger id, password, server, database og diverse optionelle valgmuligheder som f.eks. Connection Timeout som i dette tilfælde bestemmer, hvor lang tid systemet maksimalt må bruge på at skabe forbindelse. På kodeudsnit 10 kan man se hvordan forbindelsen bliver defineret og oprettet.

Kodeudsnit 10: Definition and opening of connection to database

```
1 private const string UserID      = "user id=F12I4PRJ4Gr5;";
2 private const string Password    = "password=F12I4PRJ4Gr5;";
3 private const string Server      = "server=webhotel10.iha.dk;";
4 private const string Database    = "database=F12I4PRJ4Gr5; ";
5 private const string ConnectionTimeOut = "Connection TimeOut=2";
6
7 private readonly SqlConnection _myConnection = new SqlConnection(
8                                                 UserID +
9                                                 Password +
10                                                Server +
11                                                Database +
12                                                ConnectionTimeOut
13                                                );
14 ..............
15
16 if (_myConnection.State != ConnectionState.Open)
17         _myConnection.Open();
```

Forbindelsen bliver brugt ved at eksekvere en SQL streng fra C# siden, altså en SQL statement skrevet i en string. Det er blevet forsøgt at gøre disse commands generelle, så alt efter hvad man vil på en vilkårlig tabel, kan man tilgå en generel funktion, som specificeres via parametre. Når en funktion, der henter noget fra databasen køres, vil data

returneres via en SQLDatareader som funktionen, ExecuteReader(), returnerer. Nedunder, på kodeudsnit 11 kan man se, hvordan en simpel SELECT statement bliver brugt, kørt og hvordan den returnerer data.

Kodeudsnit 11: Definition of a Select statement with a where clause

```
1 public List<string> SelectFrom(string columnName,
2                                 string tableName,
3                                 string whereColumnName,
4                                 string target)
5 {
6       if (_myConnection.State != ConnectionState.Open)
7                 _myConnection.Open();
8
9       var back = new List<string>();
10      var command = new SqlCommand(string.Format(
11                      "SELECT {0} FROM {1} WHERE {2}='{3}'",
12                        columnName, tableName,
13                        whereColumnName,
14                        target),
15                        _myConnection);
16
17      var read = command.ExecuteReader();
18      while (read.Read())
19      {
20            back.Add(read[columnName].ToString().TrimEnd(' '));
21
22      }
23      read.Close();
24      _myConnection.Close();
25      return back;
26 }
```

Når man kalder Execute reader bliver den specificerede streng kørt på databasen. Hvis operationen på databasen returnerer noget vil dette blive hentet via read[columnName] hvor columnName er den kolonne man vil hente fra. Hvis der hentes flere kolonne fra databasen vil disse blive gemt i den kolonne man specificerer. Database tilgangen sker på samme måde ved samtlige funktioner, med den undtagelse, at der ikke returneres noget på de operationer på databasen, der ikke skal returnere noget f.eks. insert- eller update statements. Disse operationer er de eneste der forekommer på databasen, udover de triggers og functions der også kan køres.