TrackABus

BACHELORPROJEKT

Systemarkitektur for TrackABus

Author:

Gruppe 13038

Supervisor:

Michael Alrøe

5. 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:	35
Kunde:	Michael Alrøe (MA).

Sted og dato: _____

10832 Christoffer Lousdahl Werge

MA Michael Alrøe

09421 Lasse Lindsted Sørensen

Indhold

1 DATA VIEW

1.1 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

1.1.1 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.

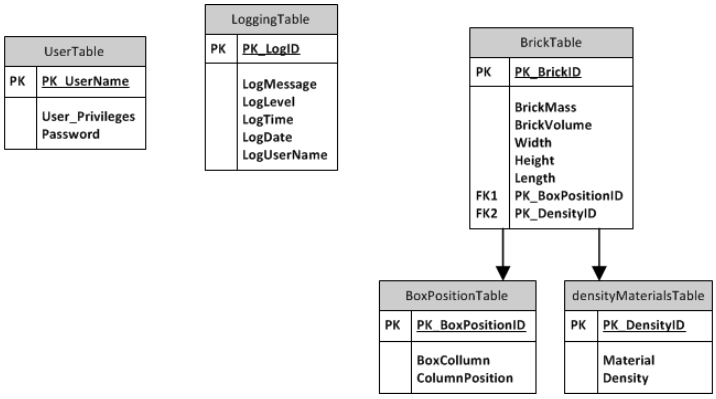
- Busposition er defineret som den fysiske placering af en given bus. I dette projekt var der dog ikke tilgang til nogen fysiske busser, så denne katagori af positions data blev simuleret. Simulatoren kunne dog skiftes ud med en virkelig bus, hvis position for denne kunne stilles til rådighed.

- 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".

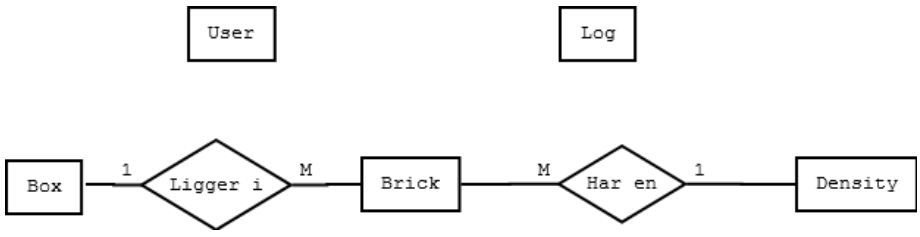
- Waypoints bruges som "genskabelses-punkter"til en given rute. Disse punkter bliver udelukkende brugt af administrationsværktøjet, til at genskabe den rute de beskriver.

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 1 vises opbygningen af tabellerne som et UML OO diagram, og på figur 2 kan relationerne i databasen ses som et ER diagram.



Figur 1: UML OO diagram over den distribuerede MySQL database



Figur 2: 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.

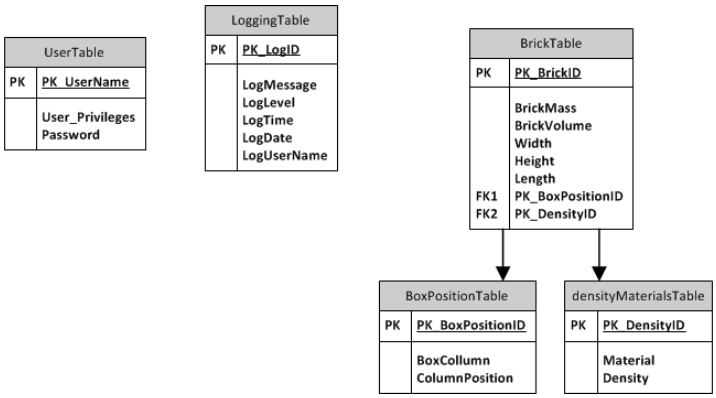
Normalform

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:

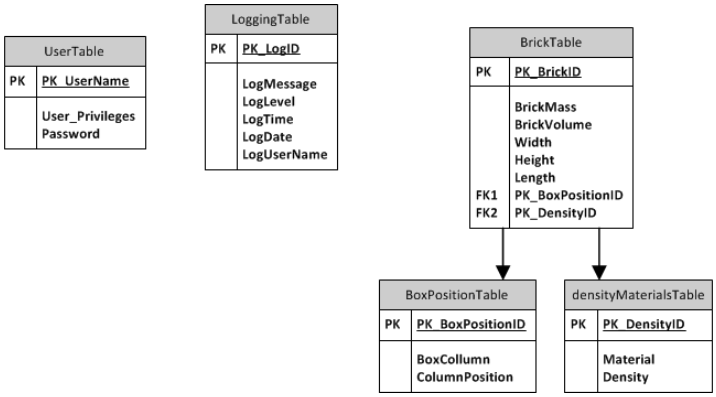
- Ingen elementer er i sig selv elementer. Dvs. ingen kolonner gentager sig selv.
- Ingen primary keys er composite keys, og derfor er ingen ikke keys afhængig af kun en del af nøglen
- Ingen elementer er afhængigt af et ikke-nøgle element. Dvs. ingen kolonner i én tabel, definerer andre kolonner i samme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

1.1.2 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 3 kan man se et UML OO diagram over den lokale SQLite database og på figur 4 kan man se et ER diagram over samme database.



Figur 3: UML OO diagram over den lokale SQLite database



Figur 4: 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.

1.1.3 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 1: 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 forskellene mellem GetClosestBusAscProc og GetClosestBusDescProc, følger efter kodeudsnittet.

Kodeudsnit 2: 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 forskelle.

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 3: 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 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.

Kodeudsnit 4: GetClosestBusDescProc forskel 2

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

1.1.4 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 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 7 kan *GetClosestEndpointAsc*-funktionen ses. Den er givet uden kommentarer eller initialisering af variabler.

Kodeudsnit 5: 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 forskelle, 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 6: 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 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 7: 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 8: 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 9: 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 10: 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. CalcRoute-LineDist 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**.

1.2 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.

1.2.1 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 13 kan en generisk oprettelse og transmitering af en SoapEnvelope ses.

Kodeudsnit 11: 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 14. 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 12: 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 15 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 13: 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".

1.2.2 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.

1.2.3 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.