0.0.1 Komponent 3: Administrations hjemmeside

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

Specifikationer

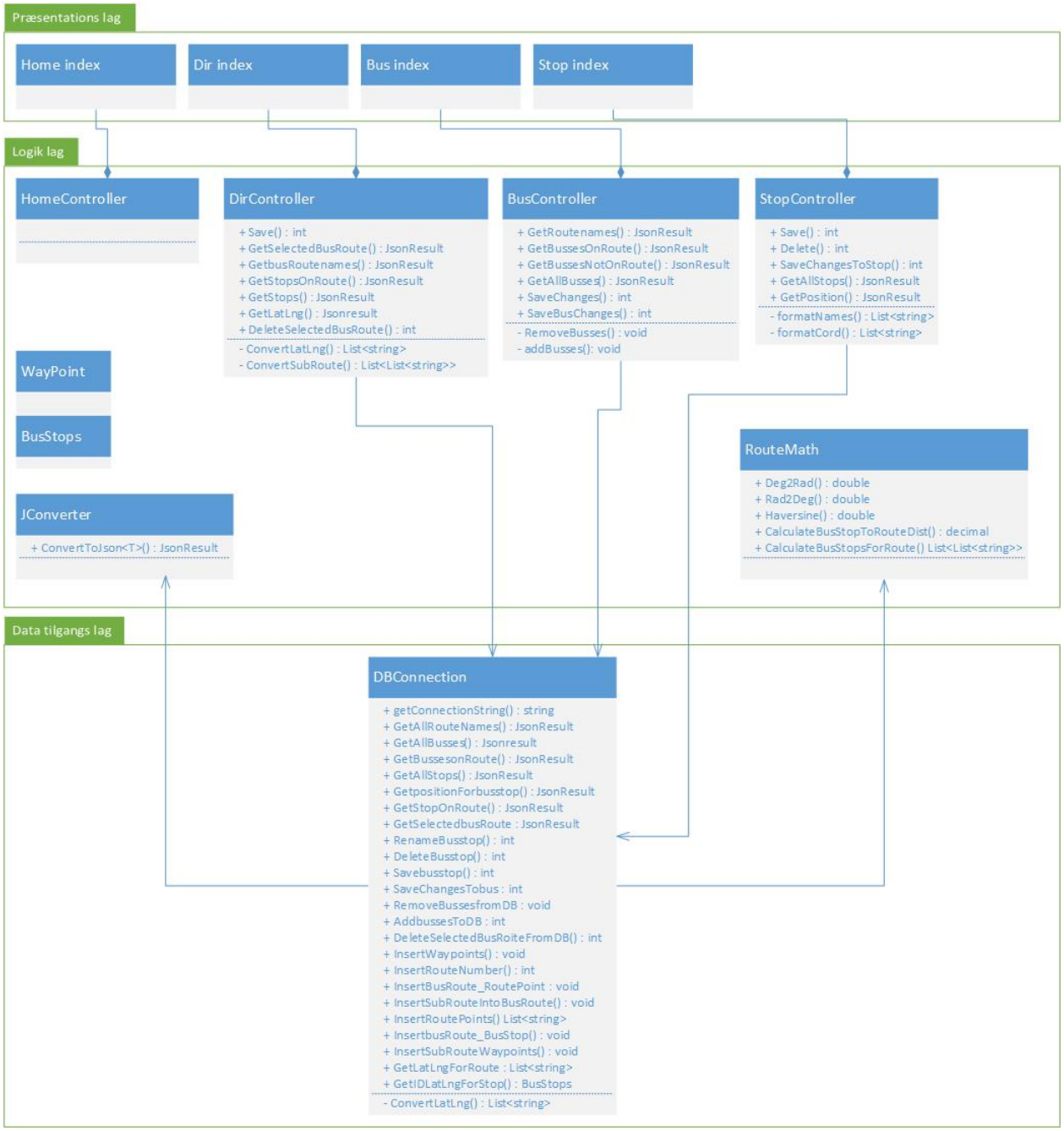
Denne komponent består fire delkomponenter:

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

På figur 1 kan administrator systemets lagdelte struktur følges, i form af et klassediagram. Herpå kan det ses, hvordan klasserne er bygget op i en tre-lags model, samt hvordan de interagerer. Når der laves en association, betyder det at klasse tilgås statisk.

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

- Home index
  - Denne klasse består af viewet for home skærmen. Denne laver ikke andet arbejde, end at lade administratoren vælge, en delkomponent.
- Dir index
  - Denne klasse består af viewet for rute redigerings skærmen. Heri ligger alle funktionaliter administratoren kan tilgå, for at oprette, fjerne eller ændre en rute i systemet.
- Bus index



Figur 1: Klassediagram for simulator. Opbygget som trelags-model

  - Denne klasse består af viewet for bus redigerings skærmen. Heri ligger alle funktionaliter administratoren kan tilgå, for at oprette, fjerne eller ændre en bus i systemet. Denne tilbyder yderligere muligheden for, at tilføje eller fjerne en bus på en rute.

- Stop index
  - Denne klasse består af viewet for stoppesteds redigerings skærmen. Heri ligger alle funktionaliter administratoren kan tilgå, for at oprette, fjerne eller ændre et stoppested i systemet.

- HomeController
  - Denne klasse står for at modtage kaldende fra Home viewet. Denne klasse gør dog intet i denne sammenhæng, da ingen kald skal modtages.

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

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

- StopController
  - Denne klasse står for at modtage kaldende fra stoppesteds viewet. Den sørger for, at kalde til data tilgangs laget, for tilgå databasen, vedrørende stoppesteds relvante oplysninger. Desuden manipulerer den det data, der kan hentes fra

viewet, således at data bliver gemt på korrekt form, samtidig med, at data hentet fra databasen bliver manipuleret, så det vises korrekt på viewet.

- Waypoint og BusStops
  - Disse er model klasser. De anses derfor som datatyper, og har ingen relationer til eller fra sig.
- RouteMath
  - Denne klasse indeholder alt relevant matematik, som skal bruges under manipulering af data, før det gemmes på databasen. Alle funktioner er statiske, så ingen klasser opretter denne.
- JConverter
  - Denne klasse sørger udelukkende for at konvertere en vilkårlig liste, til et JSON objekt. Dette bruges i sammenhæng med, at listerne af data returneret fra databasen, skal ændres til et JSON objekt, før dette kan sættes på viewet.
- DBConnection
  - Denne klasses eneste ansvar er, at tilgå databasen. Derfor er det også den eneste der ligger under data tilgangs laget. Den implementerer specifikke metoder, til brug i de forskellige Controllers. Samtlige funktioner er statiske, så de kan tilgås uden at et objekt af klassen skal oprettes.

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

Design:
Hjemmesiden er blevet implementeret ved brug af Microsoft ASP.NET MVC 4 frameworket. Dette gør det nemt og hurtigt at implementere en sofistikeret og moderne hjemmeside, der følger gode design principper. MVC står for Model-View-Controller og følger de samme principper som MVVM angående 'separation of concerns'.
For at kunne indtegne busruter og stoppesteder skal der bruges et kort, til dette er der

blevet brugt Google Maps JavaScript API[1], samt Google Directions API.[2]
Hjemmesiden består af fire views, hvor hver view har en controller knyttet til sig. Det første view administratoren vil se er startsiden, der linker til de tre andre views. Disse tre sørger for at håndtere alt vedrørende busser, stoppesteder samt busruter.

Viewet der håndtere alt om busserne, består af to dele.
Første del gør det muligt at tilføje en ny bus til systemet, fjerne en bus fra systemet og rediger IDet for en bus. Dette er blevet implementeret ved, at når view'et bliver loaded, bliver en JavaScript function kaldt. Denne kalder funktionen "GetAllBusses"i BusControlleren, der henter alle busser der er i MySQL databasen. Se *afsnit 9.2.3 Implementering af persistens i online værktøjet* for nærmere beskrivelse af hvordan databasen bliver tilgået. Til at lave dette kald fra JavaScript til controlleren, bliver der brugt AJAX. AJAX gør det muligt at udveksle data med controlleren og opdatere viewet uden det er nøvendiger at reloade hele hjemmesidensiden. AJAX kaldet kan ses på kodeudsnit 1.

Kodeudsnit 1: Ajax kald til controller funkionen "GetAllBusses"

```
1          $.ajax({
2              type: "POST",
3              url: "Bus/GetAllBusses",
4              dataType: "json",
5              success: function (result) {
6                  var select = document.getElementById("busses");
7                  select.options.length = 0;
8                  for (var i = 0; i < result.length; i++) {
9                      select.options.add(new Option(result[i]));
10                     ListOfAllBusses.push(result[i]);
11                 }
12             }
13         });
```

Når BusControlleren er færdig med at hente busserne, retuneres der et JSON object, og callback funkionen, der er defineret i success parameteren i AJAX, vil blive kaldt. "Result"parameteren på callback funktionen er returværdien fra GetAllBusses funktionen i

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

BusControlleren, der i dette tilfælde er et JSON object. Denne vil indeholde en liste af alle bussernes IDer, hentet fra MySQL databasen. Callback funktionen tilføjer alle IDer i listen, til et HTML select element. Dette gør det muligt for administratoren at se, hvilke busser der er gemt i databasen. Administratoren har nu mulighed for at enten tilføje en ny bus, slette en bus, eller ændre IDet på en bus.

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

Anden del af dette view gør det muligt at tilføje busser til- og fjerne busser fra en busrute. Denne del består af tre lister, hvor der er en, der indeholder alle busruter hentet fra databasen, en der indeholder samtlige busser, der ikke har en busrute knyttet til sig, samt en der viser, hvilke busser der køre på en valgt busrute. I dette views "onload"funktion bliver der, ud over den overnævte "GetAllBusses"funktion, også kaldt to andre funktioner. Dette forgår igen gennem to AJAX kald til BusControlleren. Den første henter navnene på alle busruter fra databasen og den anden henter en liste af IDer for alle de busser, der ikke er tilknyttet en rute. Disse to AJAX kald er magen til AJAX kaldet vist i kodeudsnit: 1, hvor den eneste forskel er, hvilken BusController funktion der bliver kaldt,

samt hvilet HTML select element der bliver tilføjet til. Det er nu muligt for administratoren at vælge en af busruterne. Dette vil starte et 'onchange' event, der laver endnu et AJAX kald til BusControlleren for at hente alle de busser, der køre på den valgte rute. De hentede busser, vil vises i listen "Busses on route". Der kan nu tilføjes busser fra listen "Avaliable busses"til listen "Busses on route". Ved tryk på knappen "Save"vil de busser, der er blevet flyttet til listen "Busses on route"bliver opdateret i databasen, således at de nu er knyttet til den valgte rute. Dette håndteres ved en AJAX kald til BusControllerens "SaveChanges"funktion, som tilgår databasen og ændrer værdierne for bussen. Dette kan der læses mere om i afsnittet *9.2.3: Implementering af persistens i online værktøjet*

Busrute viewet gør det muligt at oprette en ny busrute, ændrer i en, der alleredes findes, samt slette en givet busrute fra systemet. For at indtegne en busrute, kræver det et kort. Hertil er der blevet brugt Google Maps API[3] og Google Directions API.[4] For at få vist kortet på hjemmesiden, er det nødvendigt at det bliver initialiseret. Først og fremmest skal man have lavet plads til det på siden. Dette kan ses på kodeudsnit 2

Kodeudsnit 2: Div til google maps

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

Når HTML body elementet er loaded bliver dens onload"event kaldt. Dette kalder en JavaScript funktion, der initializere kortet samt dennes Google directions service.
Først bliver der defineret en style, som kortet skal bruge. Heri sørges der for at "Points of interest"bliver fjernet, da der bare skal bruges et tomt kort. Dernæst bliver der oprettet et mapOptions object definerer, at kortets start position sættes til at vise Aarhus. Desuden sættes der også, at korttypen er ROADMAP, da dette vil vise kortet som et simpelt kort, hvor der kun er veje. StreetViewControl bliver sat til false, da det er en feature der ikke er relevant for systemem. Denne opsætning kan ses på figur 3.

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

Kodeudsnit 3: Map opsætning

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

Efter at have defineret mapOptions bliver der oprettet et map object. Dette object skal have det HTML map-canvas element som ses på kodeudsnit 2, samt den mapOptions som laver i kodeudsnit 3. Denne oprettelse kan ses på kodeudsnit 4.

Kodeudsnit 4: Map object init

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

Kortet er nu blevet initialiseret og vil blive vist på siden. Det næste der bliver initialiseret er Google direction rendereren. Denne bliver brugt til at vise en rute på kortet mellem to givne punkter. Først bliver der defineret de options som ruten skal bruge. Dette indebærer at det skal være muligt at trække i ruten, for at ændre på den vej, den skal følge. Herudover muliggøres det også at klikke på de markører, der repræsenterer start og slut punktet for ruten. Dette rendererOptions object bliver derefter brugt i constructoren for DirectionsRendereren, som senere bliver brugt til at tegne ruten på kortet. Dette kan ses på kodeudnsit 5.

Kodeudsnit 5: DirectionsRenderer opsætning

```
1 rendererOptions = {
2 map: map,
3   draggable: true,
4   markerOptions: {
```

```
5      clickable: true
6    },
7    suppressInfoWindows: true
8  };
9  directionsDisplay = new google.maps.DirectionsRenderer(↩
       rendererOptions);
```

Efter kortet og DirectionRenderen er blevet initialiseret, bliver der sat en listener på kortet. Denne lytter efter om der bliver trykket et vilkårligt sted på kortet. Dette kan ses på kodeudsnit 6.

Kodeudsnit 6: map klik listener

```
1  google.maps.event.addListener(map, 'click', function (event) {
2    if (startPoint == null && endPoint == null) //No markers, set ↩
         first
3      startPoint = new google.maps.Marker({
4        map: map,
5        draggable: true,
6        position: event.latLng
7      });
8    else if (startPoint != null && endPoint == null) {//if 1 ↩
         markers, set last markers
9      endPoint = new google.maps.Marker({
10       map: map,
11       draggable: true,
12       position: event.latLng
13     });
14     calcRoute(startPoint, endPoint);
15     ClearMarkers();
16 }
```

Når der bliver trykket på kortet vil listeneren undersøg, om der er blevet trykket på kortet tidligere. Hvis der ikke er, vil der blive placeret en markør på kortet, der hvor der blev trykket. Denne markør symboliserer busrutens første endestation. Hvis der allerede er en markør på kortet vil der bliver placeret endnu en markør, som symboliserer busrutens sidst endestation. Efter begge markører er blevet placeret, vil functionen "calcRoute"blive kaldt, med de to satte markører. Denne function bruger Google direction service til at udregne en rute mellem de to punkter. Dette bliver gjort ved at lave et request object der definerer start og slut GPS koordinaterne for ruten. Disse koordinater bliver taget fra

de to markøre, medsendt i funktionen. Desuden sættes travelMode til at være DRIVING, så ruten vil blive vist den umiddelbart hurtigste rute i bil. Request objektet bruges i den oprettede Direction Service, til at lave ruten. Dette ses på kodeudsnit lst:calcRoute.

Kodeudsnit 7: calcRoute function

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

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

Der er også blevet sat en listener på de to DirectionRenderers, der lytter ændringer i ruten. I tilfælde af at ruten bliver ændret, vil callback funktionen blive kaldt på disse to. Heri vil der, om det første, blive der sat et kort delay. Dette gøres da ruten skal færdigudregnes før de forskellige brugte properties, bliver sat. Efter delayet bliver der itereret igennem alle properties, for at finde den af typen "Markers". Denne indeholder de markører, der symboliser start og slut punkterne, samt alle de waypoints der må være blevet lavet på ruten når den ændres. GPS koordinaterne for disse markører vil blive brugt når hele ruten skal gemmes i MySQL databasen. Til sidst vil informationen om ruten, og herunder de GPS koordinater, der bliver brugt til at tegne ruten, gemt i en variabel, der senere bliver brugt når ruten skal. Hele denne situation kan ses på kodeudnsit 8.

Kodeudsnit 8: DirectionsRenderer listener

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

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

På kodeudsnit 9 kan "SaveRouteAndStops"funktionen ses. Denne kaldes når "Save route"knappen bliver trykket. Funktionen står for at finde det rutedata der skal gemmes, samt kalde til DirControllerens "Save"funktion igennem et AJAX kald. Først findes alle de GPS koordinater, der bruges til at optegne rute. Dette gøres igennem et kald til "getRoutePath". Denne undersøger hver path, step og leg af ruten.[5]. Heri bruges variab-

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

len "temp"fra kodeudsnit 8, og vil ved fuldendelse returnere en række GPS koordinater, som ruten er bygget omkring. I denne sammenhæng ses stoppesteder og waypoints ikke som punkter. Desuden sendes tidligere fundet stoppesteder og waypoints, samt eventuelle subruter hvis ruten er kompleks, også.

Kodeudsnit 9: SaveRouteAndStops

```
function SaveRouteAndStops() {
$.ajax({
  type: "POST",
  url: '@Url.Action("Save", "Dir")',
  dataType: "json",
  traditional: true,
  data: {
    route: getRoutePath(),
    routeWayPoints: waypoints,
    stops: stopsToSave,
    SubRoutes: SplitRoute(SubRouteArray),
    SubrouteWaypoint: SubRouteWaypoints,
    RouteNumber: document.getElementById("RouteNumber").value,
    contentType: "application/json; charset=utf-8"
}
```

I "Save"funktionen kaldes der til funktionen "CalculateBusStopsForRoute", der udregner mellem hvilke rute punkter stoppestederne skal ligge. På kodeudsnit 10 kan denne udregning ses. Den første del af foreach-løkken, som kigger alle stoppesteder på ruten igennem, hvorefter stoppestedets position hentes. Dette vil ikke vises i kodeudsnittet. "CalculateBusStopToRouteDist"funktionen kan der læses mere om i afsnittet *8.2.5: Komponent: Matematik*, "Tætteste punkt på en linje".

Kodeudsnit 10: Udregninger af placering af stoppesteder på rute

```
for (int k = 0; k < chosenRouteLatLng.Count - 2; k = k + 2)
{
  currentDist =
    RouteMath.CalculateBusStopToRouteDist(
    stop.Lat, stop.Lng,
    decimal.Parse(chosenRouteLatLng[k]),
    decimal.Parse(chosenRouteLatLng[k + 1]),
    decimal.Parse(chosenRouteLatLng[k + 2]),
```

```
 9      decimal.Parse(chosenRouteLatLng[k + 3]));
10    if ((currentDist < leastDist || leastDist == -1)
11       && currentDist <= 15 && currentDist != -1)
12    {
13      leastDist = currentDist;
14      pointBeforeStopIndex = k / 2;
15    }
16  }
17  if (stops.IndexOf(s) == 0 )
18  {
19    RouteWithStopsID.Insert(0, stop.ID.ToString());
20    StopOnRoute.Add(s);
21    stopCounter++;
22    continue;
23  }
24  else if (stops.IndexOf(s) == stops.Count - 1 )
25  {
26    RouteWithStopsID.Add(stop.ID.ToString());
27    StopOnRoute.Add(s);
28    stopCounter++;
29    continue;
30    }
31  else if (leastDist != -1)
32  {
33    RouteWithStopsID.Insert(
34      pointBeforeStopIndex + stopCounter + 1,
35       stop.ID.ToString());
36    StopOnRoute.Add(s);
37    stopCounter++;
38  }
```

På dette kodeudsnit, kan processen der udføres for hvert stoppested følges. For-løkken itererer igennem samtlige rutepunkter, og udregner distancen til den linje der er spændt mellem det nuværende rutepunkt og det næste. Hvis den udregnede længde er mindre end den tidligere korteste længde, sættes det nuværende rutepunkt, som værende det bus-stoppestedet skal indsættes før. Desuden undersøges der også om distancen er mindre en 25 meter, da det er muligt at stoppestedet ligger tilpas nok forskudt til ruten, så ingen rutelængder vil opfange den. Listen af punkterne der udgør ruten, består af både bredde og længdegrader, i den rækkefølge. Derfor er listen dobbelt så lang som den egentlige punkt-liste, der består af ID'er. Indexet for det fundne tætteste punkt, skal derfor halveres. Når samtlige punkter på ruten er undersøget, findes der ud af, om stoppestedet er det første eller det sidste på ruten. Dette gøres da det første og sidste stoppested altid skal

være endestationer, og derfor ligges først eller sidst på ruten. Dette gøres uden hensyn til, hvilket punkt der egentligt er tættest på. Hvis stoppestedet ikke er det første eller sidste, indsættes ID'et for dette i listen af ID'erne for punkternerne. Hver gang et stoppested tilføjes, forøges listen med en, så derfor skal stoppestedet sættes i listen der svarende til det fundne ID plus antal tilføjede stoppesteder plus en, da det skal være punktet efter, det fundne ID. Stoppestedet tilføjes desuden til listen af stoppestedsnavne, og antallet af stoppesteder inkrementeres. Herefter returneres både listen af punkt ID'er og listen af stoppestedsnavne.
Efter alle udregnigner er fortaget, bliver alt data indsat i MySQL databasen. *Se afsnit 9.2.3 Implementering af persistens i online værktøjet* for en beskrivelse af, hvordan databasen bliver tilgået.

Det sidste view omhandler funktionerne for at tilføje nye stoppesteder, ændre position og navn for eksiterende stoppesteder, samt slette stoppesteder fra databasen. For at kunne oprette nye stoppesteder, bliver der igen brugt Google Maps API[6]. Dette bliver initialiseret på samme måde som beskrevet tidligere i afsnittet, dog uden Google Direction Services, da det kun er enkelte punkter på kortet der skal gemmes. For at oprette et nyt stoppested trykkes der på kortet, hvorefter kortets listener event bliver kaldt. Dette event vil sætte en marker på kortet, der hvor der blev trykket, som symboliserer stoppestedets position. Dette kan ses på kodeudsnit 11.

Kodeudsnit 11: Stoppested map listener

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

Når et stoppested sat kan der trykkes på "Save"knappen, som kalder JavaScript funktionen "SaveStopsToDB". Denne vil tilgå StopControllerens "Save"igennem et AJAX kald,

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

hvorefter stoppestedet vil gemmes med dennes position og indskrevne navn.

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

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