0.0.1 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, hvor hver view har en controller knyttet til sig, først et view til startsiden der linker til de 3 andre views, der består af et der håndtere alt vedrørende busser, et til stoppesteder samt et til busruter.
Det første view der håndtere alt om busserne består af 2 dele. Første del gør det muligt at tilføje en ny bus til systemet, fjerne en bus fra systemet og rediger ID'et for en bus. Dette er blevet implementeret ved at når view'et bliver loaded, bliver en JavaScript function kalde, der kalder funktionen GetAllBusses() i controlleren, der henter alle busser der er i

MySQL databasen. Til at lave dette kald fra JavaScript til controlleren, bliver der brugt ajax. Ajax gør det muligt at udveksle data med controlleren og updatere view'et uden at skulle reloade hele websiden.

Kodeudsnit 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         });
```

Dette eksemple på et ajax kald, kalder GetAllBusses(), dette er en funktion der ligger i Bus controlleren, som henter en liste af alle bussernes ID'er fra MySQL databasen. Se *afsnit 9.2.3 Implementering af persistens i online værktøjet* for nærmere beskrivelse af hvordan databasen bliver tilgået. Når controlleren er færdig retunere den et json object, og callback funkionen der er defineret i success parameteren af ajax bliver kaldt. Result parameteren på callback funktionen er returværdien fra controller funktionen, der i dette tilfælde er et json object, der indeholder en liste af alle bussernes ID'er, hentet fra MySQL databasen. Callback funktionen løber igennem listen af ID'er og tilføjer dem til et HTML select element. Dette gør det muligt for administratoren at se hvilke busser der er gemt i databasen. Administratoren har nu mulighed for at enten tilføje en ny bus, slette en bus, eller ændre ID'et på en bus.

For at tilføje en bus, skriver administratoren bussens ID ind i feltet: 'Bus ID' hvorefter han trykker på knappen 'Add'. Dette vil tilføje busses til listen, administratoren kan blive ved med at tilføje busser til listen. Administratoren kan også fjerne en bus fra listen, ved at vælge en bus i listen og trykke på knappen 'Remove', der er også mulighed for at ændre navnet for en bus, ved at vælge en bus, og trykker på 'Rename' knappen.

Først ved tryk på 'Save' knappen vil ændringerne blive tilføjet til databasen. Dette sker igen gennem et ajax kald til controller, der kalder SaveBusChanges() funktionen. Denne funktion modtager listen af busser, med de nye busser administratoren har tilføjet, samt en liste af alle busserne på databasen. Funktionen sammenligner de 2 lister, finder de busser der er blevet tilføjet, de som er blevet fjernet og dem som har fået nyt ID. Efter alt er fundet, vil den slette de relevante busser fra databasen og tilføje de nye busser.

Anden del af dette view gør det muligt at tilføje busser til en busrute og fjerne busser fra en busrute. Denne del består af 3 lister, hvor den ene indeholder alle busruter, hentet fra databasen, en der indeholder alle busser, der ikke er på nogle busruter samt en der viser hvilke busser der køre på en valgt busrute. I dette views Onload funktion bliver der, ud over den overnævte GetAllBusses() funktion, også kaldt 2 andre funktioner, dette forgår igen gennem 2 ajax kald til controlleren, den første henter navnene på alle busruter fra databasen, den anden henter en liste af ID'er for alle de busser der ikke er tilknyttet en rute endnu. Disse 2 ajax kald er magen til ajax kaldet vist i kodeudsnit: 1, den eneste forskel er hvilken controller funktion der bliver kaldt, samt hvilen HTML select element der bliver tilføjet til. Det er nu muligt for administratoren at vælge en af busruterne, fra listen. Dette vil trigger et 'onchange' event, der laver endnu et ajax kald til controller for at hente alle de busser der køre på den valgte rute, og vise dem i listen 'Busses on route'. Der kan nu tilføjes busser fra listen 'Avaliable busses' over til listen 'Busses on route' og ved tryk på knappen 'Save' vil de busser der er blevet flyttet til listen 'Busses on route' bliver opdateret i databasen, således at de nu er knyttet til den valgte rute.

Det næste view gør det muligt at oprette en ny busrute, ændrer i en der alleredes findes, samt slette en givet busrute fra systemet. For at indtegne en busrute, kræver det et kort, hertil er der blevet brugt Google maps API og Google Directions API. For at få vist kortet på hjemmesiden, kræves det at kortet bliver initializeret. Først og fremmest skal man have lavet plads til det på siden.

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

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

Kodeudsnit 4: Map object init

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

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

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

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 listenern tjekke på om der er blevet trykket på kortet tidligere, hvis der ikke er, vil der blive plaseret en marker på kortet, der hvor der blev trykket, denne marker symbolisere der hvor busruten starter. Hvis der allerede er 1 marker på kortet vil der bliver placeret endnu en marker, denne marker symbolisere der hvor busruten slutter. Efter begge markers er blevet placeret vil functionen calcRoute(startPoint, endPoint) blive kaldt. Denne function bruger Google direction service til at udregne en rute der går mellem 2 punkter. Dette bliver gjort ved først at lave et request object der definere start og slut GPS koordinaterne for ruten, disse koordinater bliver taget fra de 2 marker der er blevet placeret på kortet, samt hvilken Travelmode der skal bruges. Med TravelMode sat til DRIVING, vil der blive udregnet den rute der er hurtigts at tage med bil.

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

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

Kodeudsnit 8: directions renderer listener

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

For at kunne sætte stoppesteder på en rute, kaldes funktionen SetBusStopsOnMap(), der henter navnene på de stoppesteder der er i listen 'ToLB'. Efter alle navnene er hentet, bliver der lavet et ajax kald til controllerens GetLatLng() funktion der bruger stopnavnene til at hente GPS koordinaterne for hver stop i MySQL databasen. Disse koordinater

bliver sendt tilbage til ajax callback funktionen som et Json object, der bliver brugt til at tegne stoppestederne ind på kortet ved brug af markers.

Kodeudsnit 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"
}
```

SaveRouteAndStops() funktionen, står for at samle data der skal gemmes i databasen, og sende det til Save funktionen i Dir controlleren gennem et ajax kald. Først finder den alle de GPS koordinater der bruges til at tegne ruten, hertil bliver funktionen getRoutePath() brugt. her bliver den tidligere defineret temp variable, der holder information om ruten, brugt. For at få GPS-koordinaterne for de punkter der bliver brugt til at tegne ruten, skal hver path, for hver step, for hver leg findes. Når disse er fundet, retuneres en liste af GPS koordinater. For mere information om path, step og legs refereres til Google maps api documentation Dernæst bliver der fundet de waypoints og stoppesteder der ligger på ruten. Waypoints skal bruges til at genskabe ruten på hjemmesiden. Til sidste bliver der fundet de ruter der udgøre en kompleks rute, hvis dette er blevet tilføjet. controlleren kalder CalculateBusStopsForRoute() funktionen der laver udregninger på mellem hvilke rute punkter stoppestedet skal ligge. Denne funktion undersøger hvert individuelt stop, og udregner, hvorpå ruten dettes stop skal ligge. På kodeudsnit 10 kan første del af denne udregning ses. Udregningen er den første del af foreach-løkken, som kigger alle stoppesteder på ruten igennem, og efter stoppestedets position hentes. Haversine og CalculateBusStopToRoute-

Dist funktionerne kan der læses mere om i afsnittet *8.2.5: Komponent: Matematik*, under "Haversine"og "Tætteste punkt på en linje".

Kodeudsnit 10: Udregninger af placering af stoppesteder på rute del 1

```
1 if (k == 0)
2 {
3   currDistToStartOfRoute =
4     (decimal)RouteMath.Haversine(
5       stop.Lat,
6       decimal.Parse(chosenRouteLatLng[0]),
7       stop.Lng,
8       decimal.Parse(chosenRouteLatLng[1]));
9   currDistToEndOfRoute =
10    (decimal)RouteMath.Haversine(
11    stop.Lat,
12    decimal.Parse(chosenRouteLatLng[chosenRouteLatLng.Count-2]),
13    stop.Lng,
14    decimal.Parse(chosenRouteLatLng[chosenRouteLatLng.Count-1]));
15  if (currDistToStartOfRoute < leastDistToStartOfRoute
16      || leastDistToStartOfRoute == -1)
17  {
18    leastStartStopID = stop.ID;
19    leastStartStopName = s;
20    leastDistToStartOfRoute = currDistToStartOfRoute;
21  }
22  if (currDistToEndOfRoute < leastDistToEndOfRoute
23      || leastDistToEndOfRoute == -1)
24  {
25    leastEndStopID = stop.ID;
26    leastEndStopName = s;
27    leastDistToEndOfRoute = currDistToEndOfRoute;
28  }
29 currentDist = RouteMath.CalculateBusStopToRouteDist(
30    stop.Lat, stop.Lng
31    decimal.Parse(chosenRouteLatLng[k]),
32    decimal.Parse(chosenRouteLatLng[k + 1]),
33    decimal.Parse(chosenRouteLatLng[k + 2]),
34    decimal.Parse(chosenRouteLatLng[k + 3]));
35
36 if ((currentDist < leastDist || leastDist == -1)
37  && currentDist <= 15 && currentDist != -1)
38 {
39   leastDist = currentDist;
```

```
40    pointBeforeStopIndex = k / 2;
41 }
42 }
```

SaveRouteAndStops

se afsnit 8.2.5 Anvendt matematik Efter alle udregnigner er fortaget, bliver alt data indsat i MySQL databasen *Se afsnit 9.2.3 Implementering af persistens i online værktøjet* for hvordan databasen bliver tilgået.

Det sidste view omhandler funktionerne for at tilføje, nye stoppesteder, ændre position og navn for eksiterende stoppesteder, samt slette dem fra databasen. For at kunne oprette nye stoppesteder, bliver der igen brugt Google maps API, dette bliver initialiseret på samme måde som beskrevet tidligere i afsnittet, dog uden Google Direction Services, da det kun er enkelte punkter på kortet der skal gemmes. For at oprette et nyt stoppested, bliver kortets listener event kaldt, ved tryk på kortet, dette event vil sætte en marker på kortet, der hvor der blev trykket, som symbolisere stoppestedets position.

Kodeudsnit 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 });
```

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