

Avstand og bompengeberegningsmodul til HRessurs

BO15-G02

Vedlegg B - Notater fra utviklingsprosessen

Robin Furu – Ingvild Karlsen Bjørlo – Christian Jacobsen – Glenn Bjørlo

1. Utvikling av applikasjonen - steg for steg

1.1 Kommentarer til vedlegg

Dette vedlegget inneholder våre notater fra utviklingsprosessen. Her finnes en del tekniske og grundige beskrivelser av prosessen som er utelatt i hoveddokumentet. Vi går gjennom alle steg vi har vært gjennom i utviklingsprosessen. Vi går inn på hva som fungerte, hva som ikke fungerte og hvordan vi løste problemene som oppsto. Alle de ulike forsøkene vi har gjort for å finne beste måte å løse et problem på er beskrevet.

Det er i all hovedsak lagdelingen av applikasjonen, det å komme frem til den beste løsningen for viapunkter og hvordan vi skal fremstille kjøreruta som er beskrevet. Disse har vi måtte teste flest alternative løsninger for, før vi kom frem til et godt fungerende resultat

Notatet er skrevet etterhvert som vi har utviklet applikasjonen, slik at tekstene av og til bærer preg av at vi ikke hadde full forståelse av alle begreper og sammenhenger fra begynnelsen. Av denne grunn er også strukturen preget av at teksten er skrevet uten fokus på struktur. Dette vedlegget har vært brukt som hjelpemiddel når hoveddokumentet har blitt skrevet.

1.2 Alt i ett løsning

Vår første versjon av applikasjonen baserer seg kun på HTML og Javascript. Den bruker også 3 forskjellige API-systemer. For den geografiske delen og for å hente steder og koordinater bruker vi Google Maps API og Google Autocomplete API. All informasjon i forbindelse med kjøreruta håndteres av Ruteplantjenesten.

Når man taster inn start- og stopp-adressene, bruker vi Google sitt API til å autofullføre søkene. Når autofullføring av søk er ferdig og man trykker «Beregn», vil det kjøre en funksjon som henter koordinatene fra adressesøket. Deretter konverteres koordinatene om til presentasjonsformatet UTM som Ruteplantjenesten benytter.

Når koordinatene er ferdig konvertert, blir de lagt inn i en URL-string som vi sender til Ruteplantjenesten sitt REST API. Fra Ruteplantjenesten får vi returnert et JSON-objekt med forskjellig informasjon slik som avstand, kjørebekrivelse, bomstasjoner, rasteplasser, bompengekostnader og mye annet. Fra JSON-objektet henter vi ut total kjørelengde og bompengekostnader på ruta. Vi presenterer de beregnede verdiene, og viser kjøreruta på et kart. Her angir vi bare start og stopp, slik at Google automatisk finner beste kjørerute, som ikke nødvendigvis stemmer helt med kjøreruten Ruteplantjenesten har beregnet.

1.3 HTML - Web Service Versjon 1

I vår applikasjon skal brukergrensesnittet være knyttet mot en Web Service, som senere skal ligge på Infotjenesters server. Dette er for å ha så lite logikk som mulig på klientsiden.

Denne første løsningen baserer seg i utgangspunkt på testdata. Vi skal sende inn start, stopp, viapunkter og kjøretøy til Web Service, slik den ferdige løsningen også skal kunne. Web Service skal så returnere testdata for bomstasjoner, priser og avstand. Web Service skal senere kobles mot Ruteplantjenesten, slik at testdata kan endres til virkelige verdier beregnet på grunnlag av angitte steder og valgt fremkomstmiddel.

Flere søk på Google viser at det som ser ut som den enkleste og mest utbredte måten å gjøre dette på, er ved bruk av Microsoft sin `useService`. Eksemplene på nett er mange, og de er veldig relevant med tanke på applikasjonen vi utvikler. Vi prøver mange eksempler, både ved å klippe ut og lime inn kode, og ved å laste ned hele løsninger. Ingen av løsningene klarer derimot å opprette forbindelse til Web Service, og ingen gir feilmelding på hvorfor. Dette testes i både Mozilla Firefox og Google Chrome. Ved å studere Microsoft sin dokumentasjon av `useService` for å finne ut av hvorfor dette ikke fungerer, viser det seg at denne funksjonen var faset ut siden Internet Explorer 10, som ble lansert høsten 2012. Da dette er nokså nylig, med tanke på at en del fremdeles kjører eldre versjoner av programmer en stund før de tvinges til å oppgradere, finnes det også en del nyere eksempler der `useService` brukes. Det kan nevnes at ved å senere teste løsningen i Internet Explorer 11, får vi ut feilmelding. Ved nærmere undersøkelse, viser det seg også at hovedtyngden av eksempler er datert til og med 2013. Publiseringsdato, samt å teste kode i alle nettlesere, er noe vi vil ha fokus på dersom vi møter tilsvarende problemstillinger underveis.

Da vi ikke skal bruke ASP.NET, som beskrevet i kapittel 2.2.2 i hoveddokument, blir løsningen å bruke `xmlHttpRequest`. Her sender vi inn parameterne som «key value pair», verdipar. Viapunktene samler vi i en tekststreng, adskilt med komma. På denne måten kan vi sende x-antall viapunkter inn som et parameter, slik at vi ikke må ha ulike metoder for hvert antall viapunkter. I Web Service splitter vi denne tekststrengen på komma, slik at vi får viapunktene fordelt i et array. Da kan viapunktene behandles hver for seg under kalkuleringen av ruten.

Vi har laget en klasse, `CalculatedRoute`, der et objekt av klassen, inneholder all informasjon om en beregnet kjørerute. Dette objektet inneholder informasjon om start, stopp, valgt kjøretøy, viapunkter, totalpris for liten og stor bil og bomstasjoner (navn og priser på alle bomstasjonene). Strukturen kan sees i figur 1.1 på neste side. Når brukeren trykker knappen «Send rute», sendes verdiparet med inntastet data til Web Service. I Web service initialiseres et nytt objekt av `CalculatedRoute` basert på input-dataene og resultatet av den beregnede ruta. Dette sendes så tilbake til klientsiden, der vi «parser» (pakker ut) objektet, og fremstiller relevant informasjon for brukeren. Den første løsningen inneholder bare testdata, slik at bomstasjonene får navn fra Bom0 til Bom1, og prisene for hver av bomstasjonene generes tilfeldig med en «randomfunksjon». Prisen for liten bil, `TotalCostSmall`, generes tilfeldig mellom 0 og 20, mens prisen for stor bil, `TotalCostLarge`, er `TotalCostSmall + 100`. Denne beregningen skal senere utvides til å foregå med data fra Ruteplantjenesten.

```
{
  "Start": "Halden",
  "Stopp": "Oslo",
  "Via": [
    "Sarpsborg"
  ],
  "Vehicle": "mc",
  "Distance": 93,
  "TotalCostSmall": 45,
  "TotalCostLarge": 545,
  "Barriers": [
    [
      "Bomnr0",
      "12",
      "112"
    ],
    [
      "Bomnr1",
      "2",
      "102"
    ]
  ]
}
```

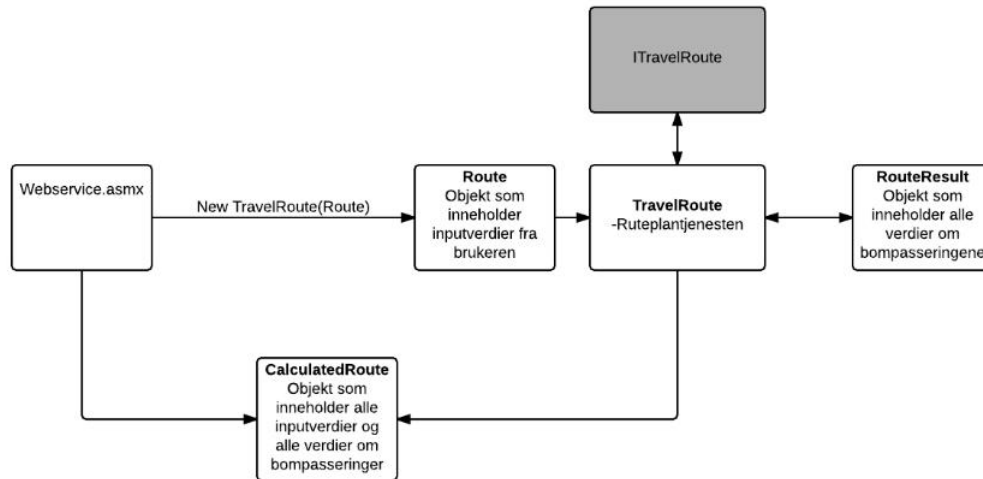
Figur 1.1: Strukturen på CalculatedRoute

Da Web Service ikke returnerer flerdimensjonale arrays, har vi valgt å returnere JSON. JSON har den fordelen at selv om selve objektet er flerdimensjonalt, kan det returneres som en endimensjonal tekststreng, som vi kan parse i Javascript for å hente ut objektet. Web Service har XML som standardformat for data som returneres. Dette vil ikke la seg endre til JSON så lenge vi bruker `xmlHttpRequest` for å opprette forbindelse med Web Service. Meningene på utallige diskusjonsforum er delte på om det i det hele tatt er mulig å få Web Service til å returnere JSON via `xmlHttpRequest` eller ei. Løsningen blir å returnere JSON pakket inn i XML-tagger. Disse fjerner vi i Javascript, før vi parser JSON-objektet.

1.4 HTML - Web Service Versjon 2

I et sprintmøte med vår kontaktperson Petter Ekrann, får vi bekreftet at det skal la seg gjøre å få Web Service til å returnere et JSON-objekt. Alt tilsier at `xmlHttpRequest` skal kunne returnere JSON, men dette lar seg ikke gjøre. Etter mye testing og søking på nett kan det se ut som dette er en funksjonalitet som er fjernet i nyere versjoner av .NET. Et nytt forsøk er å endre `xmlHttpRequest` til AJAX-request, i håp om at dette vil fungere. For å redusere kode og antall parametere, og blant annet forbedre innsendingen av viapunkter, vil det være hensiktsmessig å sende inn et JSON-objekt som parameter, istedenfor verdipar.

All kode som nå ligger direkte i Web Service, skal flyttes ut i en egen klasse, bygd opp etter struktur vi fikk introdusert i et kort intensivkurs med Petter Ekrann i midten av januar, se figur 1.2. Dette vil gjøre et eventuelt bytte av ruteberegningstjeneste enklere.



Figur 1.2: Strukturen fra Web Service til Ruteplantjenesten

Vi begynner først med å implementere strukturen i vår eksisterende løsning. All kode flytter vi deretter inn i klassen TravelRoute, som er bygd opp basert på interfacet ITravelRoute. I Web Service opprettes et objekt av klassen Route, som etter hvert skal inneholde all informasjon om ruten brukeren har tastet inn. Foreløpig inneholder dette objektet kun stedsnavn for start, stopp og via, i tillegg til valgt kjøretøy. I denne versjonen har vi fokuset på å utbedre applikasjonen på serversiden. For å kunne teste applikasjonen raskere, definerer vi et ferdig inputobjekt i Web Service, slik at vi slipper å angi verdier hver gang vi tester. Ved å teste applikasjonen ved å trykke på en knapp på klientsiden, får vi samtidig testet at vi ikke foretar endringer som ødelegger for kommunikasjonene mellom klientsiden og Web Service.

I Web Service initialiserer vi et nytt objekt av klassen TravelRoute med Route-objektet som inputparameter. All data er foreløpig fortsatt dummydata. Koden som skal beregne bompenger, flyttes over i CalculateResult, all kode som bygger opp det endelige objektet som returneres til klientsiden ligger i CalcReturn. Strukturen i interfacet kan sees i figur 1.3

```

public interface ITravelRoute
{
    2 references
    String Search(travelroute.Route input);
    2 references
    Result CalculateResult(String jsonInput, travelroute.Route input);
    2 references
    CalculatedRoute Calculate(travelroute.Route input);
    2 references
    CalculatedRoute CalcReturn(travelroute.Route input, travelroute.Result result);
}
  
```

Figur 1.3: Strukturen på interfacet

Vi har nå flyttet det meste av koden ut av Web Service, og nå skal dummydata erstattes med reelle data. Vi begynner her med å implementere metoden Search. Her benytter vi en `HttpRequest` mot Ruteplantjenesten. For å gjøre testingen lettere, kjører vi til å begynne med en forespørsel mot en ferdigdefinert URL fra Ruteplantjenesten.

Ruteplantjenesten returnerer JSON. Vi sender JSON-tekststrengen videre som input til metoden `CalculateResult`. Her bygger vi opp et objekt av typen `Result`, som inneholder alle relevante verdier vi henter fra JSON-fila. Til slutt kjøres metoden `CalcResult`, som bygger objektet med all data om reisen, som returneres til klientsiden. Dette endelige objektet inneholder start, stopp og viapunkter i tekstformat, samt all informasjon om bompasseringer og avstander. Det tekstlige innholdet med start, stopp og viapunkter skal senere byttes ut med en mer detaljert rutebeskrivelse som skal erstatte det nåværende tekstfeltet i `HRessurs` der kjøreruta skal spesifiseres detaljert.

1.5 Utforming av viapunktliste

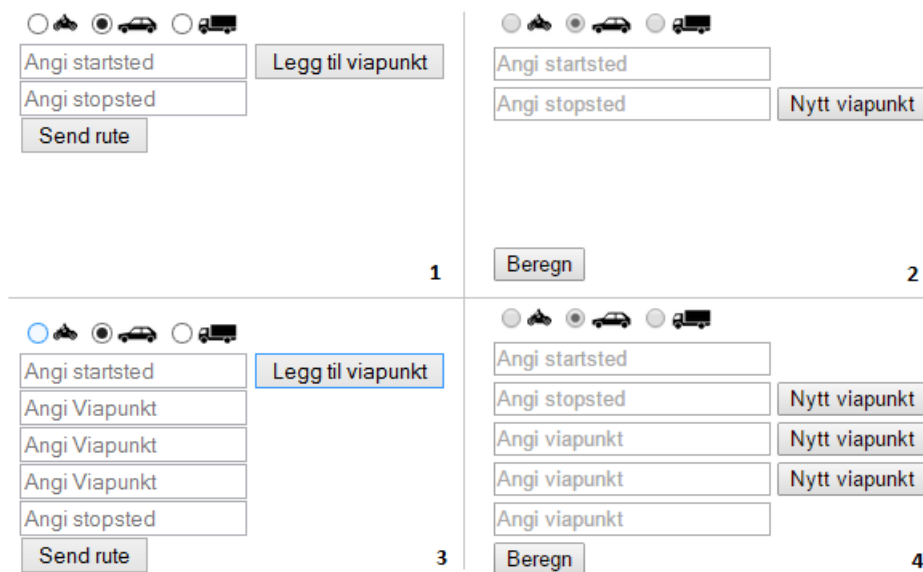
For at brukeren skal kunne legge inn en så nøyaktig reiserute som mulig, har vi valgt å gjøre det mulig å legge til viapunkter. Dersom en destinasjon ikke har en opplagt rute, men kan nås ved å velge flere forskjellige ruter, eller hvis brukeren har blitt tvunget til å kjøre en omvei, kan det være nødvendig å legge til flere punkter for å få en mer nøyaktig rute enn bare start og stopp. En omkjøring kan føre til at ruten blir adskillig lengre og prisene for bompasseringer på ruta kan endre seg. Vi vil at registrering av rute skal foregå så dynamisk som mulig, men også så nøyaktig som mulig. Brukeren skal derfor kunne legge til viapunkter mellom start og stopp-feltene. Det vil si at brukeren kan velge å trykke «Legg til viapunkt», og nye tekstfelt vil bli lagt til. Brukeren fyller da inn disse feltene med de stedene han har måttet kjøre via for å nå destinasjonen. Det bør selvsagt være en begrensning på antall felter som kan legges til, slik at ikke systemet overbelastes med unaturlige mange ekstra lokasjoner. Vi har valgt å legge maks antall viapunkter på fem, slik at det totale antall steder som kan registreres er sju.

For å angi viapunkter starter vi med skjulte tekstfelt, tekstfelt som allerede eksisterer, men er satt til å være usynlige (`hidden`) i CSS. Disse endrer vi til synlig, når brukeren trykker knappen «Legg til viapunkt». Et alternativ er å gjøre alle tekstfeltene synlig (`visible`) når knappen trykkes. Problemet da er at man får mange felter på én gang som brukeren kanskje ikke har behov for. Vi benytter først en løsning med én knapp til hvert tekstfelt. Når knappen «Legg til viapunkt» trykkes, gjøres både det første tekstfeltet, og den neste «Legg til viapunkt»-knappen synlig. Et stort problem her er at brukeren må vite at det kun er den nyeste knappen som vil fungere til å legge til enda et tekstfelt, eventuelt må den første knappen skjules når den neste gjøres synlig. En annen løsning er å benytte en knapp og knytte denne mot en teller.

Vi finner ut at å bruke Javascript til å lage nye tekstfelt som settes inn i en ordnet liste kanskje er en bedre løsning. Listen er i utgangspunktet tom, og hver gang knappen «Legg til nytt viapunkt» trykkes, legges det til et nytt listeelement med et tekstfelt. Dette gjør at antall viapunkter ikke er forutbestemt av antall forhåndsdefinerte tekstfelt, samtidig som viapunktene ikke opptar plass i brukergrensesnittet før de legges til. Her kan man senere endre maks antall felter ved å endre verdien på en variabel. Innenfor CSS finnes det riktignok to måter å skjule/vise HTML-elementer. Det ene er å bruke «`visibility`» og det andre er «`display`». Fordelen med sistnevnte er at elementet ikke tar opp plass

fordi det i utgangspunktet ikke vises. Førstnevnte gjemmer bare elementet slik at det alltid vil ta opp plass selv om det ikke er synlig for brukeren. Skjermbildene i figur 1.4 på neste side, illustrerer forskjellen på usynlige tekstfelt, og tekstfelt i liste.

1. viapunktene ligger i liste, ingen viapunkter er lagt til
2. viapunktene ligger som usynlige tekstfelt, legg merke til avstanden ned til knappen «Beregn»
3. Her er det lagt til viapunkter i lista, og det er fremdeles plass til flere
4. Her er det også lagt til viapunkter, og de utfyller den «ledige plassen» i bilde 2. Her er det ikke plass til flere viapunkter.



Figur 1.4: Viapunkter, CSS og Javascript

Brukeren skal også kunne sortere viapunktene dersom det legges inn to eller flere felter. For eksempel hvis det oppstår en situasjon der brukeren oppdager at han har lagt inn steder han har kjørt via, i feil rekkefølge. Da skal han slippe å fjerne teksten i feltene og fylle de inn på nytt. I stedet skal brukeren kunne ta tak i ett av feltene å dra det over eller under det andre feltet. Den beste måten for å gjøre dette teknisk er å bruke Javascript-biblioteket jQuery. Å bruke dette biblioteket gjør det enklere å manipulere HTML-elementer. Dessverre kan ikke tekstfelt flyttes på direkte, vi må gå en omvei. Dersom feltene legges inn i listeelementer og knyttes sammen med et bilde (i vårt tilfelle en hånd som viser at man kan manipulere feltene, se figur 1.5), kan man derimot flytte dem rundt. Brukeren klikker på bildet som ligger ved siden av tekstfeltet og flytter på dette. Tekstfeltet følger med, siden bildet ligger i samme listeelement som det tilhørende tekstfeltet.

Vi har oppdaget at det oppstår store problemer med vår valgte metode for å lage tekstfelt for viapunkter. Vår foretrukne løsning så langt er å bruke Javascript til å lage nye elementer ettersom det er behov for dem, men løsningen kommer ikke uten utfordringer. Det første problemet som oppstår ved vår løsning skjer i nettleseren Mozilla Firefox. Dersom en bruker lager nye tekstfelt ved å trykke en legg til- knapp, får han ikke muligheten til å trykke seg inn i selve feltet med musepekeren slik at

han kan fylle det ut. Det fungerer i start og stopp-feltene som eksisterer hele tiden, men ikke i nye input-felt som først eksisterer etter at brukeren har valgt å legge de til. Den eneste måten brukeren har for å kunne skrive noe som helt i disse feltene er ved å bruke tabulator-knappen på tastaturet. Da trykker man seg gjennom alle elementer på siden inntil man når input-feltet man vil fylle ut. Men dette blir en uholdbar løsning. Har man trykket inn noe feil i foregående element, i dette tilfellet et av tekstfeltene for viapunkter, må man trykke seg gjennom hele runden av diverse elementer før man kommer tilbake til det elementet som skal redigeres. Funksjonaliteten er også testet i Google Chrome og Internet Explorer uten problemer, men siden Firefox er en såpass utbredt nettleser kan vi ikke overse denne feilen.






Figur 1.5: Viapunkter til å manipulere

En feil som oppstår i forhold til Google Autocomplete skjer dersom brukeren vil slette et felt, og så legge det til igjen. La oss si brukeren vil legge til to felter for viapunkter og trykker legg til-knappen to ganger slik at feltene lages og blir tilgjengelige. Brukeren fyller ut først kun ett felt, og velger å slette det andre som ble lagt til. Men kanskje han ombestemmer seg, og vil ha ett felt til allikevel. Trykkes knappen for å lage nye felter flere ganger etter at brukeren først har slettet feltene vil ikke Google Autocomplete fungere. Den fungerer kun dersom man legger til ett eller flere felter en gang, men ikke dersom disse slettes for så å legges til igjen. Dette ser vi på som et mindre problem, da det er lite sannsynlig at brukeren vil gjøre akkurat dette. Man blir i så fall nødt til å laste inn siden på nytt. Men feilen er en av flere som altså oppstår ved vårt valg av funksjonalitet for å lage felter for viapunkter. Vi innser at det antakeligvis må til en annen løsning for viapunkter.

Den alvorligste feilen har ikke direkte å gjøre med hvordan felter for viapunkter blir tilgjengelige for brukeren, men kan løses ved nettopp å forandre måten dette gjøres på. La oss si en bruker vil legge til maks antall viapunkter. Hun trykker derfor legg til -knappen fem ganger slik at fem viapunkter kan registreres. Men til sammen blir dette sju koordinater (inkludert start/stopp) som først skal behandles av Google Autocomplete, konverteres til UTM-formatet av vår Javascript-konverter, behandles av Vegvesenets servere, og så sende korrekt informasjon tilbake til klienten. Det finnes ingen måte å lagre noe informasjon på forhånd, det tas i liten grad hensyn til at dersom en av tjenestene bruker tid på å svare vil applikasjonen kunne stoppe opp. Og denne feilen har skjedd flere ganger, rett og slett fordi for mye informasjon behandles på én gang. Dette er det alvorligste problemet vi har støtt på i klientsiden. Ved å lagre noe informasjon underveis mens brukeren fyller inn det hun har bruk for, vil vi kunne unngå denne feilen. Samtidig vil vi løse problematikken angående viapunkt-feltene.

For å løse problemene vi har støtt på, har vi valgt å gå litt tilbake, og kombinere CSS med Javascript igjen. Vi behøver ikke å ha mer enn ett felt for viapunkter, se figur 1.6. Velger bruker å legge til viapunkter trykker han knappen for å legge til viapunkt én gang og feltet blir synlig (dette eksisterer allerede). På høyre side av tekstfeltet for viapunkter blir to nye knapper tilgjengelig. En for å lagre og

en for å slette. Hver gang brukeren lagrer et viapunkt, blir koordinatene for stedet hentet av Google, sendt til konverteringsfunksjonen og lagret i en egen sortèrbar liste (en HTML ul-liste) under selve tekstfeltet. Samtidig tømmer innholdet i tekstfeltet slik at nye steder kan legges til. Ett og ett viapunkt konverteres og lagres før selve beregningen av avstander og bompenger kjøres. På denne måten unngår vi at applikasjonen blir overbelastet. Samtidig kan brukeren sortere ul-listen ved å dra rundt på hvert element som inneholder et lagret viapunkt. Her unngår vi også å legge til et ekstra element for å gjøre sortering mulig siden viapunktene ikke lagres i tekstfelter.

Kjøretøy:
☐  ☒  ☐ 

Start:

Stopp:

Figur 1.6: Viapunkter med ett input-felt.

Som beskrevet tidligere har vi valgt å legge til en ekstra funksjonalitet som gjør eventuelle behov for å redigere listen av viapunkter enklere. Nemlig at brukeren kan sortere listen av viapunkter etter at de er lagt til. En ting er hvordan det skal løses i selve brukergrensesnittet, altså hvordan skal brukeren kunne sortere listen. En annen utfordring er hvordan applikasjonen skal registrere en ny rekkefølge på viapunktene og dermed gjøre beregningene ut i fra den nye rekkefølgen.

Vi har beholdt den samme jQuery-funksjonen som tidligere beskrevet for å muliggjøre sortering av elementer på nettsiden. Forskjellen ligger i hva slags HTML-elementer vi nå sorterer. Før sorterte vi tekstfelter som vi knyttet sammen med et bildeelement for å muliggjøre sortering, mens nå sorterer vi liste-elementer (ul-liste).

Vi utvider bruken av jQuery-biblioteket til også å gjelde funksjonaliteten som skal fange opp sorterte verdier fra viapunkt-listen. Siden vi har bestemt oss for å forhåndslagre viapunkter før selve beregningen utføres, utnytter vi dette når vi skal lage funksjonalitet for sortering. For hver gang brukeren lagrer et viapunkt blir disse konvertert og lagret som et array som igjen lagres inni et annet array, et multidimensjonalt array. Vi har et «hovedarray» som lagrer opptil fem andre array. Hvert array som lagres inne i hovedarrayet inneholder tre verdier: stedsnavn, x-koordinater og y-koordinater.

Stedsnavnet lagres som en string (ren tekst) direkte fra tekstfeltet.

```
[[Drammen,Norge", 230607.516, 6632656.55],[Son,Norge", 256162.997, 6606055.00]]
```

Listen som viser viapunktene for brukeren henter ut stringen som inneholder via-stedsnavnet og viser denne i listen. Det er ikke nødvendig å skrive ut koordinatene til brukeren, da dette bare vil være overflødig informasjon som kan skape forvirring. Vi har kommet frem til at den beste måten å sortere et array, er å sammenligne det med et annet.

Listen som inneholder viapunktene (altså stedsnavnene) er som nevnt en HTML uordnet-liste (ul) som igjen inneholder en eller flere liste-elementer (li). Vi har en Javascript/jQuery funksjon som henter all informasjon som finnes i ul-listen. Denne funksjonen samler disse verdiene i den rekkefølgen de befinner seg i. Det betyr at dersom brukeren har lagret for eksempel Stavanger, Drammen og Kristiansand som viapunkter i den rekkefølgen, men velger å bytte om på Drammen og Kristiansand vil funksjonen lagre byene i den nye rekkefølgen i stedet.

Ved å bruke jQuery kan man telle seg gjennom alle listeelementer for å hente verdiene som er lagret. Hvert listeelement funksjonen finner lagres i en variabel som så pushes inn i et array. Denne arrayen vil tilslutt inneholde alle stedene i ul-listen. Denne funksjonen kaller på en ny funksjon som skal sammenligne innholdet i det aller første arrayet som inneholder både koordinatene og stedsnavnene i den rekkefølgen de først ble lagret av brukeren. Denne funksjonen vil sortere elementene basert på et «key»-element, altså en nøkkel som skal gjenkjennes. Husk at det opprinnelige arrayet består av flere arrays som hver og en inneholder tre verdier: to sett koordinater pluss stedsnavnet. Og det er stedsnavnet som brukes som nøkkel-elementet. Dersom nøkkelen befinner seg på en annen plass enn tidligere, så vil funksjonen ta nøkkelen og dens tilhørende verdier (altså koordinatene) å lagre dem i den nye posisjonen. Figur 1.7 viser vår sorterbare liste.

Panel	Start:	Viapunkter	Stopp:
1	Moss, Norge	Son, Norge Drammen, Norge	Kristiansand, Norge
2	Moss, Norge	Drammen, Norge Son, Norge	Kristiansand, Norge
3	Moss, Norge	Son, Norge Drammen, Norge	Kristiansand, Norge
4	Moss, Norge	Son, Norge Drammen, Norge	Kristiansand, Norge

Figur 1.7: Sorterbare

Et nytt problem som dukket opp med tanke på funksjonen som behandler sortering er at man ikke kan legge til et nytt viapunkt etter at brukeren har trykket beregn. Altså etter at alle verdier er lagret og eventuelt sortert, og brukeren vil vite avstander og bompenger. Dersom han vil legge til et ekstra viapunkt hvis ruten var feil, vil funksjonen som skal samle inn stedsnavnene i ul-listen ikke ta med seg de gamle, men kun den/de nye. Hva som forårsaker dette er uvisst, men problemet kan løses ved å tømme viapunktlisten etter hver beregning. Det betyr at dersom brukeren vil gjøre en ny kalkulering av bompenger og avstander så må hun legge til viapunktene på nytt. Start og stopp trengs derimot ikke å legges til på nytt. Så lenge man ikke laster siden på nytt vil disse verdiene fortsatt være tilgjengelig i tekstfeltene. Det er heller ikke nødvendig å legge til funksjonalitet som tømmer disse feltene da de ikke har noe med feilen i forbindelse med sortering å gjøre.

For at brukeren ikke skal kunne legge til et uendelig antall viapunkter, har vi laget en teller i Javascript som lytter på knappen brukeren må trykke på for å lagre et viapunkt. Dersom denne telleren står på fem, vil brukeren få beskjed om at det ikke kan legges til flere viapunkter samt at både feltet for å lagre viapunkter og de tilhørende knappene forsvinner fra siden. Men dersom slett-knappen trykkes slik at viapunktlisten tømmes, vil knappen nullstilles. Da kan brukeren fortsatt legge til nye viapunkter. Når en beregning har blitt gjennomført vil ikke bare viapunktlisten tømmes, men også telleren vil bli satt tilbake til null.

1.6 Veien mot endelig brukergrensesnitt

Brukergrensesnittet vårt baserer seg mye på HRESSURS sitt brukergrensesnitt. Dette er fordi et av hovedmålene med denne oppgaven er å få applikasjonen vår integrert i HRESSURS. Brukergrensesnittet skal være enkelt og det skal være like lett for brukere som er innom daglig som det er for de som er innom å registrerer reisen en gang i måneden.

Etter første møte med oppdragsgiver bestemte vi oss for å lage en skisse over hvordan hvordan vi så for oss at vi kunne løse oppgaven, se figur 1.8.



The image shows a first draft of a user interface for a travel application. On the left, there is a form with two input fields labeled 'Start' and 'Stopp', each with a location pin icon to its right. Below these are two radio button options: 'Direkte' (selected) and 'Via-punkt'. At the bottom of the form is a large text area with the labels 'Avstand:' and 'Bompenger:'. To the right of the form is a map of Norway, labeled 'Norge' and 'Norway'. The map shows major cities like Bergen, Oslo, and Trondheim, and is overlaid with a network of yellow lines representing roads or routes.

Figur 1.8: Første utkast brukergrensesnitt

Når vi fikk tilgang til HRESSURS, laget vi en skisse på hvordan vi kunne tenke oss at applikasjonen vår kunne bli implementert i HRESSURS. Skissen er vist i figur 1.9

The mockup shows a web browser window with a navigation bar at the top containing links: Info, Dokumenter, Ferie, Fri & permisjon, Sykefravær, Kompetanse, Utlegg, and Reiser. Below the navigation bar is a button labeled 'Gå tilbake til mine reiser'. The main content area has a horizontal progress bar with steps: Start, Overnatting, Måltidsfradrag, Kjøring (highlighted), Utlegg, Tillegg, and Fullfør. The 'Kjøring' section contains two columns of input fields. The left column includes: 'Rute' (a large empty box), 'Total kilometer' (a text input), 'Fremkomstmiddel' (a dropdown menu with 'Privat bil' selected), 'Kilometer utland' (a text input), 'Årsak til eventuell omkjøring' (a text input), and two checkboxes: 'Passasjerer' and 'Spesielle tillegg'. The right column includes: 'Fremkomstmiddel' (radio buttons for different vehicle types), 'Start' (a text input), 'Stopp' (a text input), 'Via' (a text input with a plus icon), and a 'Beregn' button. Below these inputs are two buttons: 'Legg til' and 'Kalkulator'. On the far right, there is a 'tes' label. The 'Beregnet rute:' section on the right contains a text box with the text: 'Resultat av beregning: start stopp og via, avstand, bomstasjoner m/ priser Veibeskrivelse'. Below this is a 'Legg til i reiseregning' button.

Figur 1.9: Ideutkast implementert versjon

Når vi startet og utvikle applikasjonen vår så startet vi med den samme enkle tankegangen som de tidligere versjonene, og vår første applikasjon er vist i figur 1.10.

The mockup shows a web browser window with a title 'Kart informasjon'. The main content area has a left sidebar with input fields: 'Start:' (a text input with 'Angi startsted' below it), 'Stopp:' (a text input with 'Angi stopsted' below it), 'Biltype:' (radio buttons for different vehicle types), 'Via:' (a text input with 'Via punkt' below it), and a 'Beregn' button. Below these are labels for 'Avstand:', 'Bompenger:', 'Du kjørte fra:', 'via:', and 'til:'. The right side of the interface features a Google Map of Norway and Sweden, with labels for 'Norge', 'Norway', 'Sver', 'Swe', 'Bergen', 'Oslo', and 'Göteborg'. The map is titled 'Kart' and 'Satellitt'. The Google logo is visible at the bottom left of the map area.

Figur 1.10: Tidlig fremstilling av Brukergrensesnitt

Figur 1.11 viser hvordan det nåværende systemet til Infotjenester ser ut, og det er det vi har basert vårt endelige brukergrensesnitt på. Vi venter nå på skisser som en designer hos Infotjenester skal sette sammen for oss, slik at vi kan utbedre GUI'et slik at det blir helt identisk til HRESSURS.

The screenshot shows a web application interface for travel planning. At the top, there is a navigation bar with tabs: Info, Dokumenter, Ferie, Fri & permisjon, Sykefravær, Kompetanse, Utlegg, and Reiser. Below the navigation bar is a button labeled "Gå tilbake til mine reiser" and a "test" link. The main content area is divided into two columns. The left column contains a form with the following fields: "Rute" (a text input), "Total kilometer" (a text input), "Fremkomstmiddel" (a dropdown menu with "Privat bil" selected), "Kilometer utland" (a text input), and "Årsak til eventuell omkjøring" (a text input). Below these fields are two checkboxes: "Passasjerer" and "Spesielle tillegg". At the bottom of the left column is a button labeled "Legg til". The right column contains an "Info" section with the following text: "Kjørerruten skal beskrives så nøyaktig som mulig, med adresser for hvert delstopp. Kjørerruten skal oppgis så nøyaktig at den kan etterprøves i antall kilometer." and "Kilometer med passasjer oppgis ved at man summerer antall kilometer pr passasjer, og legger inn totalsummen." Below this text is an example: "F.eks. Per sitter på 50 km, mens Kari sitter på 30 km. Da legger du inn 80 km med passasjer."

Figur 1.11: HRESSURS design

Figur viser vårt endelige resultat før tilpasning etter skisser fra Infotjenester.

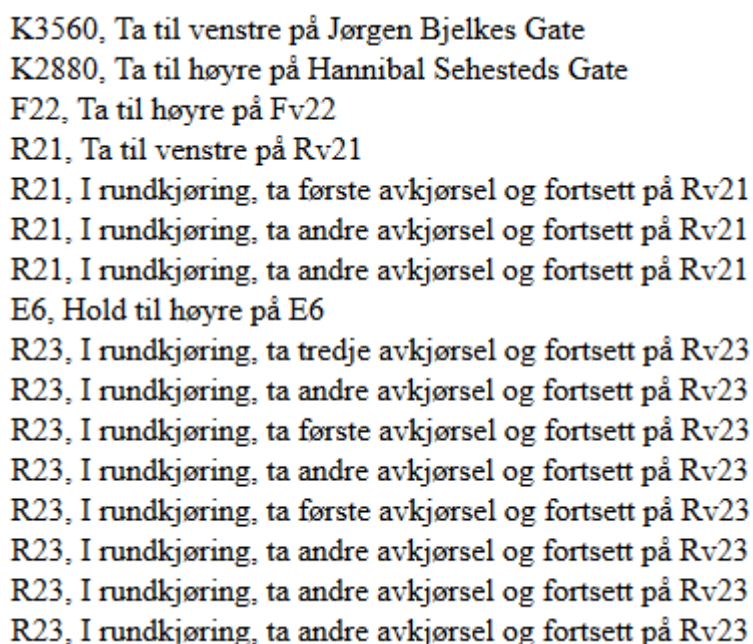
The screenshot shows a web application interface for travel planning. At the top, there is a navigation bar with tabs: Kart, Resultat, Kjørerrute, and Bomstasjoner. Below the navigation bar is a section labeled "Kjørertøy:" with four radio buttons: a car, a truck, a motorcycle, and a bus. Below this section are three text input fields: "Start:" (labeled "Angi startsted"), "Stopp:" (labeled "Angi stoppested"), and "Eventuell kommentar". Below these fields are two buttons: "Legg til via punkt" and "Beregn". To the right of the input fields is a large text area labeled "Kart". Below the "Kart" area is an "Info" section with the following text: "Dobbeltsjekk at all informasjon stemmer i forhold til reiseruten. Hvis reiseruten er korrekt, trykk bekreft. Hvis ikke, prøv å legge til et eller flere via punkter og beregn rute på nytt." Below this text is a button labeled "Bekreft rute".

Figur 1.12: Tidlig brukergrensesnitt

1.7 Fremstilling av beregnet kjørerute

1.7.1 Tekstlig fremstilling

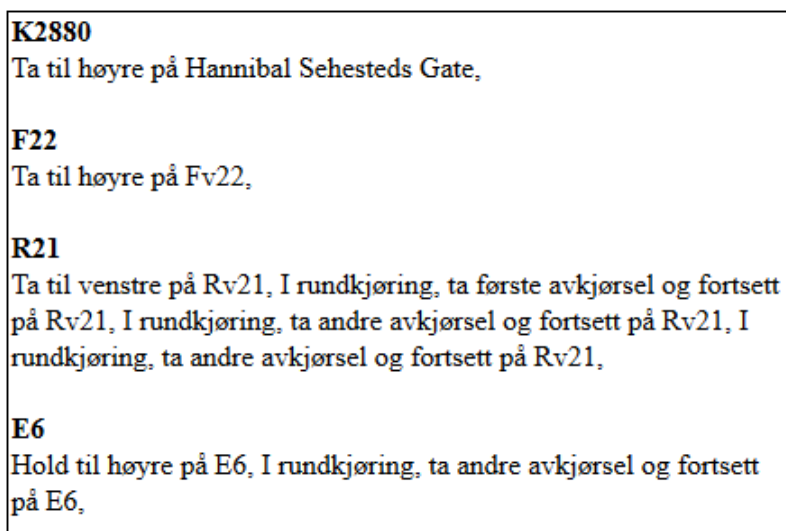
Ruteplantjenesten er i hovedsak, som navnet tilsier, en ruteplanleggingstjeneste. Dette innebærer at innholdet tjenesten returnerer, bærer preg av å være tiltenkt reiseplanlegging og ikke dokumentering i ettertid slik som vår applikasjon skal brukes til figur 1.13 viser et utklipp av en veibeskrivelse fra Halden til Drammen. Med mindre man har et kart man kan følge med på, er ikke dette en god nok presentasjon av ruta dersom man i ettertid skal kontrollere at ruta stemmer med hvor man har kjørt. Det brukes veldig få stedsnavn i beskrivelsen, og der det gjøres er det kun gatenavn på småveier. Dette vil si at gatenavnene kun er til hjelp om man er lokalkjent, eller har mulighet til å slå opp gatenavnene i et kart. I det store og hele kan man få en oversikt over hvor den beregnede ruta går, dersom man kjenner veinummeret på de store veiene man har kjørt, men antall tekstlinjer per veinummer kan skape forvirring. På kjøreturen fra Halden til Drammen ligger man på E6 i ca 1 time, og E6 har kun en tekstlinje da det kun er å holde rett fram fra man kjører på, og til man skal av. Riksvei 21, veien ut av Halden, har hele fire tekstlinjer, men der kjører man til sammenligning bare i ca 10 minutter.



K3560, Ta til venstre på Jørgen Bjelkes Gate
K2880, Ta til høyre på Hannibal Sehesteds Gate
F22, Ta til høyre på Fv22
R21, Ta til venstre på Rv21
R21, I rundkjøring, ta første avkjørsel og fortsett på Rv21
R21, I rundkjøring, ta andre avkjørsel og fortsett på Rv21
R21, I rundkjøring, ta andre avkjørsel og fortsett på Rv21
E6, Hold til høyre på E6
R23, I rundkjøring, ta tredje avkjørsel og fortsett på Rv23
R23, I rundkjøring, ta andre avkjørsel og fortsett på Rv23
R23, I rundkjøring, ta første avkjørsel og fortsett på Rv23
R23, I rundkjøring, ta andre avkjørsel og fortsett på Rv23
R23, I rundkjøring, ta første avkjørsel og fortsett på Rv23
R23, I rundkjøring, ta andre avkjørsel og fortsett på Rv23
R23, I rundkjøring, ta andre avkjørsel og fortsett på Rv23
R23, I rundkjøring, ta andre avkjørsel og fortsett på Rv23

Figur 1.13: Tekstlig beskrivelse av kjøreruta.

For å kunne oppfylle reiseregningskravet om at kjøreruta skal være spesifisert tekstlig, er vi nødt til å komprimere veibeskrivelsen slik at vi kan få de viktigste punktene. Det er ikke relevant hverken for brukeren selv eller den som skal kontrollere reiseregningene å vite alle småveier brukeren har kjørt på. Det er mer relevant å vite de store veiene, og det gir en overordnet og god nok oversikt over hvor brukeren har kjørt. Første steg er å gruppere veibeskrivelsen kronologisk etter veinummer. For å få til denne sorteringen benytter vi oss av multidimensjonale array. Vi bruker en løkke for å gå gjennom alle veibeskrivelsene, og kjører en test på om veinummeret er det samme som det foregående. Dersom gjeldene veinummer er likt det foregående, legges den tekstlige veibeskrivelsen inn i underarrayet som tilhører veinummeret. Dersom det er avvik i veinumrene, opprettes et nytt element i hovedarrayet med det nye veinummeret. Slik fortsetter det til hele veibeskrivelsen er gruppert. Utskriften av veibeskrivelsen blir da seende ut slik som i figur 1.14.



K2880
Ta til høyre på Hannibal Sehesteds Gate,

F22
Ta til høyre på Fv22,

R21
Ta til venstre på Rv21, I rundkjøring, ta første avkjørsel og fortsett på Rv21, I rundkjøring, ta andre avkjørsel og fortsett på Rv21, I rundkjøring, ta andre avkjørsel og fortsett på Rv21,

E6
Hold til høyre på E6, I rundkjøring, ta andre avkjørsel og fortsett på E6,

Figur 1.14: Gruppert, tekstlig beskrivelse av kjøreruta.

Å gruppere etter veinummer gir en bedre oversikt over hvor man har kjørt. For brukeren som skal kontrollere reiseruta vil dette være et mye bedre alternativ enn beskrivelsen i figur 1.13, men det er fremdeles ikke en god nok beskrivelse til at vi ikke trenger å ha med en kartvisning i applikasjonen. For den kortfattede rutebeskrivelsen som skal lagres i reiseregningene beholder vi kun veinumrene på riksveier, fylkesveier og europaveier, og utelater private og kommunale veier, som er veier med et veinummer som starter med K eller P. Den kortfattede veibeskrivelsen for ruta i figur 1.14 blir da: «Kjørerute via F22, R21 og E6». Denne må så settes i sammenheng med angitt start og stopp, slik at den endelige beskrivelsen blir: «Kjørt fra Halden til Drammen, kjørerute via F22, R21 og E6».

Å gruppere arrayet etter veinummer viser seg å være tidkrevende. Det tar tid å finne ut nøyaktig hvilke variabler vi skal teste mot hverandre, om vi skal teste mot det foregående veinummeret, eller det neste veinummeret. Her går det med en del timer til prøving og feiling, og til å finne ut hvordan den nye arraystrukturen skal bygges opp mest fornuftig. Vi får problemer med å nulle ut underarrayene når vi oppretter en ny veibeskrivelse, og vi får istedenfor en haug med tomme arrayer slik som dette «[[[[[[]]]]]]», istedenfor den tekstlige beskrivelsen: «[tekstbeskrivelse1, tekstbeskrivelse 2, tekstbeskrivelse 3]». Det viser seg til slutt at feilen kun ligger i hvor vi deklarerer variablene, og at vi slipper å nulle de ut manuelt hver gang, slik vi prøvde først.

1.7.2 Grafisk fremstilling

Konvertering av koordinater

Som nevnt foretar Ruteplantjenesten kun beregninger av kjøreruter innenfor Norges grenser, mens Google Maps ikke tar hensyn til landegrenser når kjøreruter plottes på kartet. Dette skaper problemer med tanke på å plotte kjøreruta direkte, kun basert på angitt start, stopp og eventuelle viapunkter. Både Ruteplantjenesten og Google Maps beregner korteste veien, slik at det i utgangspunktet ikke skal være noe problem å plotte kun basert på angitte viapunkter i tilfeller der det ikke finnes tvil om at korteste rute kun er innenfor Norge. Dette kan for eksempel være en kjøretur mellom Haugesund og Stavanger. Det er derimot enda et problem, som tilsier at vi ikke kan ta sjansen på at begge beregner samme rute. Ruteplantjenesten inneholder oppdatert informasjon fra Vegvesenets trafikkmeldinger om stengte veier, omkjøringer osv. Det kan av denne grunn føre til at Google Maps viser korteste veien, mens avstanden og bompengekostnadene som beregnes, skjer på grunnlag av en omkjøring Google ikke tar hensyn til. Vi er derfor nødt til å tvinge Google Maps til å legge kjøreruta innom punkter underveis, for å få ruta som tegnes på kartet til å bli så lik som mulig den ruta som er beregnet. Google støtter det å legge til viapunkter i API-et når en kjørerute skal plottes, og dette ønsker vi å benytte oss av når ruta skal tegnes inn på kartet. Da Ruteplantjenesten kun gir tekstlig veibeskrivelse med få stedsnavn, kan vi ikke benytte veibeskrivelsen til å hente ut stedsnavn til viapunkter.

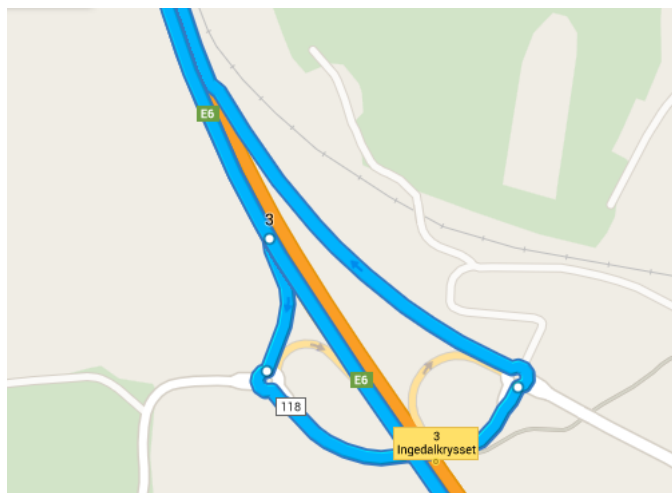
Vi er nødt til å hente koordinater underveis i den beregnede ruta, i form av compressed geometry (komprimerte koordinater). Et eksempel på compressed geometry er: `”+34+pcuhk+jirn14+ls+34+fk+io+0+fk-68+cg-cg+cg-fk+34-fk-68-1”`. Dette er ikke koordinater som kan overføres direkte til Google Maps. For det første må disse koordinatene dekomprimeres, slik at vi får de som vanlige koordinater. Vi prøver først et rent kodeeksempel fra Esri, men her blir det masse feilmeldinger. Vi velger å gå videre til å teste et annet eksempel de tilbyr, et ferdig prosjekt for å dekomprimere compressed geometry. Det kan spare oss mye tid, dersom dette fungerer uten å måtte gjøre alt for store endringer. Esri tilbyr en plattform for geografiske applikasjoner, Arcgis, og her tilbyr de DecodeGC, et ferdig prosjekt som dekomprimerer koordinatene. Dette er en konsollapplikasjon, som vi har tilpasset og gjort om til klassen DecompressGeometry. Det er ikke mulig å legge til kun denne klassen i vårt eksisterende prosjekt, da den inneholder en web-referanse til Arcgis. Ved å prøve å legge til denne web-referansen i vårt prosjekt Ruteplantjenesten, får vi bare feilmeldinger om at referansen ikke finnes. For å få tatt vare på web-referansen som er nødvendig for at dekomprimeringen skal fungere, modifiserer vi derfor heller prosjektet DecodeGC slik at dette får samme struktur som resten av prosjektet vårt, og legger til en referanse fra Ruteplantjenesten til DecodeGC. DecodeGC skriver bare ut koordinatene i Debug-vinduet, og har støtte for koordinater bestående av X, Y, Z, M. De komprimerte koordinatene Ruteplantjenesten returnerer består kun av X og Y, slik at vi er nødt til å bygge opp et multidimensjonalt array som for hvert koordinat inneholder X og Y-verdier.

Koordinatene vi får returnert fra DecompressGeometry, er i UTM. Google krever koordinatene i desimalgrader. Dette er en omstendelig omregning slik at vi tar utgangspunkt i et ferdig eksempel. Vi gjør om eksempelkode til å ta koordinatene fra DecompressGeometry som inputparametere, og legger resultatet inn i et nytt multidimensjonalt array. Vi legger til et nytt attributt på objektet som returneres til klientsiden, slik at vi får returnert koordinatene. Konverteringa kjøres en gang per koordinatpar, slik at alle koordinatene konverteres.

Dette fører til at det vil ta lenger tid å beregne kjøreruta fra Halden til Bodø, enn det vil ta å beregne ruta fra Halden til Oslo. En løsning er å konvertere kun de koordinatene vi har bruk for.

Google Viapunkter

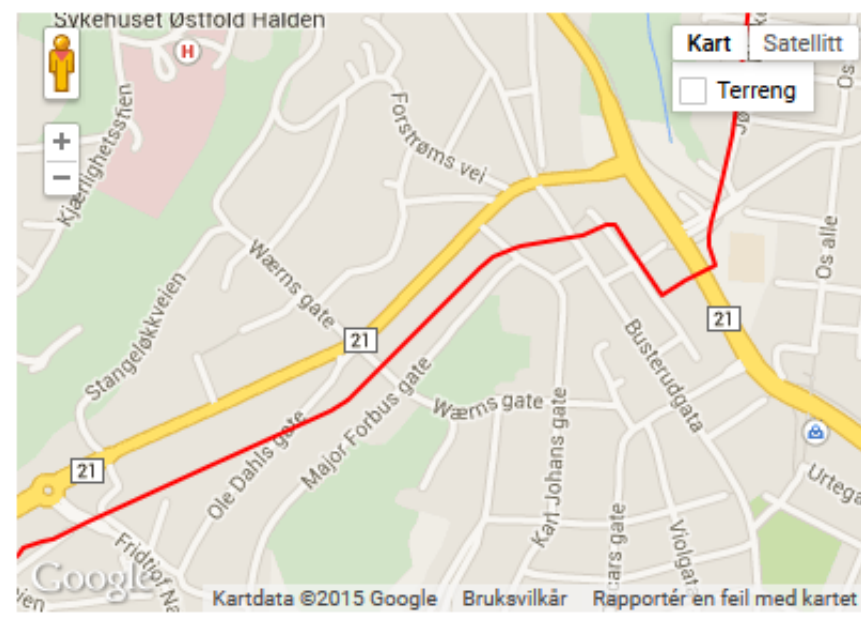
I Google kan man tvinge en kjørerute innom viapunkter. I dette kapittelet vil vi bruke ordet viapunker og Google viapunkter, og ikke viapunkter brukeren angir. Google tillater kun åtte viapunkter, med mindre man kjøper tilgang for å kunne angi noen flere viapunkter. Vi velger derfor å plukke åtte koordinater, jevnt fordelt på ruta. Vi dividerer antall koordinater på åtte, og får da spranget mellom hvert av koordinatene vi skal plote på kartet. Når vi plotter disse, virker det ved første øyekast som at dette er en holdbar løsning. Kjøreruta fra Halden til Kirkenes legges via Norge hele veien opp, og ikke gjennom Sverige slik som tidligere. Når vi begynner å teste flere ulike kjøreruter, dukker problemene opp med at kjøreruta ikke alltid vises i kartet. For eksempel er det ikke noe problem å vise kjøreruta fra Halden til Kirkenes, mens kjøreruta fra Oslo til Kirkenes ikke vil la seg vise i kartet. Det viser seg også at koordinatene ikke er helt nøyaktige, slik at man på E6 der det er to separate kjøreretninger, ofte opplever at kjøreruta vises i begge feltene. Dersom et punkt ligger i motsatt kjøreretning, er kjøreruta inntegnet som vanlig frem til neste avkjøring. Der tar man av fra E6, og fortsetter tilbake i motsatt kjørefelt i retning opprinnelsesstedet. Ved neste avkjørsel, tar man av E6 igjen, og fortsetter igjen mot målet på E6. Med et zoom-nivå som dekker hele kjøreruta, ser alt tilsynelatende greit ut, og det er først når man zoomer nærmere inn på ruta, at man legger merke til at kjøringen på E6 går fram og tilbake mellom avkjørslene slik som figur 1.15 viser. Dette er ikke en holdbar løsning, og vi er nødt til å prøve å finne en annen måte å tvinge ruta innom punkter Ruteplantjenesten returnerer.



Figur 1.15: Kjørerute fram og tilbake på E6

Google Polyline

Valget faller på å prøve Google Polyline. Denne plotter en mengde koordinater på kartet, og trekker en linje mellom disse. Jo flere koordinater, jo mer nøyaktig blir den inntegnede linja i forhold til den virkelige kjøreruta. Vi velger derfor å plote alle koordinatene, og vi får en jevn linje mellom start og stopp. Polyline er ikke avhengig av at koordinatene ligger ved en vei, slik at problemet vi hadde med viapunkter der ruta ikke alltid ble plottet, forsvinner. Her vil ruta plottes hver eneste gang. Et problem som blir tydelig da vi plotter alle koordinatene som en polyline, er at det er en nokså konsekvent forskyvning i forhold til veien ruta er beregnet langs, slik som figur 1.16 på neste side viser.



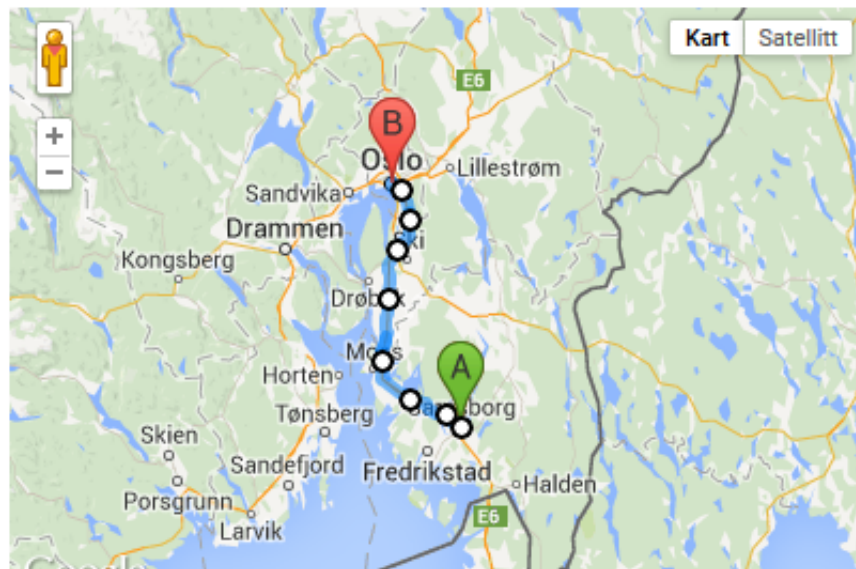
Figur 1.16: Polyline er jevnt forskjøvet iforhold til riktig kjørerute

For å prøve å fikse dette, sender vi koordinatene gjennom Googles Snap-to-Road-funksjon. Denne tvinger en polyline til de veiene det er mest sannsynlig at kjøreruta går. Problemet med Snap-to-Road er at den tvinger polyline til nærmeste vei, slik at den ikke vil håndtere forskyvningen. Dersom en annen vei ligger nærmere enn den man faktisk har kjørt, vil ruta tvinges innom denne. Dersom Snap-to-road ikke finner et godt alternativ til en vei, vil resultatet bli som i figur 1.17 på neste side. Her følger kjøreruta delvis en vei, før den ved punkt A tar en snarvei tvers gjennom skogen og kommer inn på veien ved punkt B. I utgangspunktet er ruta beregnet langs Tistedalsveien, illustrert med den røde pila.



Figur 1.17: Googles snap-to-road på polyline

Utseende på den plottede kjøreruta har også betydning for at vi velger polyline framfor viapunkter. Figur 1.19 viser hvordan kjøreruta ser ut etter å ha blitt plottet med viapunkter. På en kort strekning slik som Sarpsborg – Oslo, der de åtte punktene har en kort avstand mellom hverandre, fremstår viapunktene som forstyrrende elementer. På en lengre strekning vil punktene fordeles utover et større område, slik at de ikke fremstår like forstyrrende.



Figur 1.19: Sarpsborg - Oslo, fremstilt med viapunkter

Figur 1.20 viser strekningen Halden – Oslo, plottet ved hjelp av polyline. Her ser kjøreruta mer oversiktlig ut, og den er på denne korte strekningen plottet gjennom ca. 1300 koordinater.



Figur 1.20: Halden - Oslo, fremstilt som polyline

Optimalisering av koden

Etter et møte hos Infotjenester der vi demonstrerte vår foreløpige applikasjon for Petter Ekrann, kommer vi i samarbeid fram til at beregningen av koordinatene tar for lang tid. For vår del, som skal teste applikasjonen hver gang vi gjør små endringer i koden, blir hastigheten på beregningene, et frustrasjonselement. Vi går derfor også ut fra at dette vil kunne oppfattes veldig negativt for fremtidige brukere som ofte kjører langt i forbindelse med jobb.

Det første vi gjør i fellesskap, er å prøve å finne ut hvorfor beregningene tar så lang tid. Ekrann foreslår at det kan være dekomprimeringsmetoden som tar tid, da denne som nevnt tidligere i kapitlet, foregår med en web-referanse mot ArcGIS sin NAserver. Han ønsker at vi prøver å bytte ut denne, slik at alle beregningene i applikasjonen, med unntak av Google-elementene som må skje på Googles server, skal foregå på Infotjenesters server. Fra testing og debugging, har vi derimot slått fast at dekomprimeringen går veldig raskt, og at det først er når konverteringen starter, at beregningene begynner å gå sakte. Det kan være mange grunner til at konverteringen går sakte. Blant annet beregnes det en hel mengde verdier hver gang konverteringsmetoden kjøres. Disse kan med fordel flyttes ut av metoden, slik at de bare beregnes én gang. En reiserute fra Halden til Kirkenes inneholder ca 30 000 koordinater, og dersom vi kan beregne de fleste verdiene som trengs til konverteringen bare én gang istedenfor 30 000 ganger, kan dette føre til en raskere beregning.

For å kontrollere koordinatene som ble beregnet, la vi i implementeringsfasen av konverteringsmetoden på en debug-utskrift. Når vi så går i gang med å optimalisere konverteringen, viser det seg at utskriften fremdeles er aktiv, og i en beregning på 30 000 koordinater, vil den skrive ut alle de 30 000 beregnede koordinatene. Dette kan også ta unødvendig tid. Vi fjerner først denne utskriften, og ved å fjerne denne ene kodelinja, går applikasjonen med ett mye raskere. Fremdeles tar det lenger tid å beregne en lang rute, men det går vesentlig raskere og hastigheten er akseptabel. Det viser seg at koordinatkonverteringen er avhengig av alle beregningene for hver konvertering slik at vi ikke får gjort noe med disse.

Samtidig som vi jobber videre på å utbedre kode, begynner vi å se på mulighetene for å publisere applikasjonen, slik at vi kan få testet den. Det dukker raskt opp et problem med delprosjektet DecodeGC. På grunn av DecodeGC lar ikke prosjektet seg publisere, og vi har dermed enda en viktig grunn til at vi må gå bort fra DecodeGC og NAserveren. Vi endrer da dekomprimeringen, slik at vi ikke lenger bruker NAserveren. Etter litt undersøkelser, kommer vi bare fram til to ferdige C#-konverteringsfunksjoner, den fra ArcGIS med NAserveren, og den fra Esri som vi prøvde helt i begynnelsen uten hell. Vi er derfor nødt til å gjøre et nytt forsøk på skriptet fra Esri. Nå har vi allerede grunnstrukturen i metoden, med tanke på input- og outputformatene. Og med kun små justeringer i koden, fungerer dekomprimeringen som den skal. Løsningen på hvorfor denne ikke fungerte til å begynne med, er mest sannsynlig at vi kopierte koden inn på feil sted i vår kode, og at uriktig kodestruktur gjorde at koden ikke fungerte.

1.8 Tilpassninger til HRessurs - Veien mot implementering

1.8.1 Optimalisering av koden - Javascript

Etter de siste endringene vedrørende viapunkter, har vi gjennomgått all Javascript kode klientside og foretatt en opprydning. En del logikk ligger igjen fra tidligere i prosjektperioden, og kan enten fjernes eller forenkles. I tillegg har vi begynt å se på om vi kan avkorte noen av funksjonene, unngå gjentakelser, samt flytte noe mer kode over i de samme filene slik at vi har minst mulig Javascript foran i index.html. Vi har derfor samlet det meste av Javascript koden i to egne filer som hentes inn i HTML-filen. Noe kode er det begrenset hva vi kan forandre på siden vi benytter oss av Google API enkelte steder. Koden i forbindelse med Google API-et er ikke veldig mottakelig for redigering, og vi er nødt til å beholde både globale variabler og noen gjentakelser for at disse skal fungere optimalt.

Noe kode har vi kunnet utbedre, slik at vi ikke lenger har gjentakelser. For eksempel har vi hatt én kalkulatorfunksjon for hver av koordinatene vi har trengt konvertert. Det vil si en for start, en for stopp og en for viapunkter. Den viktigste årsaken til dette var å unngå overbelastning. Siden vi nå konverterer ett og ett viapunkt for hver gang det lagres, har vi sett på muligheten for å konvertere start og stopp gjennom den samme kalkulatoren uten å forandre for mye på Google API-et. Det lot seg gjøre ved å sende med start- og stoppverdiene som parametere til selve funksjonen. I stedet for å opprette globale variabler for lengde- og breddegrader, altså fire variabler som inneholder to og to koordinatsett for henholdsvis start og stopp, sender vi i stedet inn to lokale variabler som holder på koordinatene hver gang funksjonen som skal konvertere verdier kalles. Det vil si den kalles to ganger, først for start og så for stopp.

Kalkulatoren er bygget opp slik at den bare kan regne ut to koordinater av gangen. Det betyr at man ikke kan sende flere enn ett koordinatsett på én gang. Kalkulatoren forventer å få inn lengde- og breddegrader som den skal konvertere til x- og y-koordinater. Kalkulatoren kan få disse verdiene på to måter: enten hente variabler fra ett sted, eller få verdiene som parametere.

Den først nevnte metoden må da hente inn to globale variabler som settes til lik to lokale variabler inne i konverteringsfunksjonen. Problemet da blir at de to lokale variablene er fastsatt til alltid å være lik kun disse to globale variablene, for eksempel variablene for start-koordinatene. Variablene for stopp-koordinatene vil ha andre navn som ikke kan settes til lik de to lokale variablene i konverteringsfunksjonen siden de allerede er opptatt. Løsningen blir da å lage to kalkulatorer. En som får de globale variablene for start og en annen kalkulator for stopp.

Dersom variablene i stedet sendes med som parametere når konverteringsfunksjonen kalles, kan man bruke den samme funksjonen om igjen så mange ganger man vil. I stedet for å sette variablene i konverteringsfunksjonen lik de globale variablene, setter man dem til likinput parameterne i stedet. Med tanke på viapunktene har ikke dette latt seg gjøre, da viapunktene foretar en god del flere beregninger for hvert viapunkt som lagres. Da konverteres koordinatene, før de legges til i den sorterbare lista.

1.8.2 Optimalisering av koden - C#

I vår applikasjon har vi benyttet mange lister, blant annet nestede lister (en liste, med flere underlister i) og lister med array. For oss som har utviklet koden, er ikke dette noe problem, da vi har full oversikt over hvordan listene er bygd opp, og hvor vi finner alle verdier vi er ute etter. For de som ikke har skrevet koden selv, kan dette være tidkrevende å finne ut av. Og siden applikasjonen vi utvikler skal tas i bruk i HRESSURS, er det viktig at den er vedlikeholdbar. Det første vi begynner med er å redusere antall lister, og gjøre koden mer objektorientert. Et eksempel på kode vi forbedrer, er koordinatlistene i dekomprimeringen og konverteringen. Her er koordinatene lagret som en liste med array av double (desimaltall). For å hente ut første koordinat må vi gjøre det i form av «Listenavn[0][0]» og «Listenavn[0][1]». For en som ikke har sett koden før, vil det være veldig vanskelig å vite hva disse listeelementene inneholder. Vi gjør listen med array om til en liste med tupler, som vil si en liste med to elementer. Det første koordinatet i lista hentes da ut som «Listenavn[0].Item1» og «Listenavn[0].Item2». Dette er en litt bedre måte, men enda bedre vil det være å erstatte tuplene med et objekt, slik at det vil være en liste av koordinatobjekter. Det er ikke mulig å endre navnene på Item1 og Item2, men dersom vi erstatter tuplene med koordinatobjekter, vil vi kunne navngi disse som Xcoordinate og Ycoordinate, og de er med ett mer forståelig. I første omgang lar vi de stå som tupler.

Veibeskrivelsen har vi også lagret i lister med array, her av typen string. Den usorterte lista med tekstbeskrivelser består av mange array som inneholder to tekststrenger: veinummer og den tekstlige beskrivelsen. Disse må også refereres til slik som med lista over koordinater, «Listenavn[0][0]» og «Listenavn[0][1]». Vi erstatter arrayene med objekter av en ny klasse vi lager, Directions, med egenskapene RoadNumber og TextDescription. Da kan vi referere til listeelementene på en måte som er forståelig også uten å ha fullstendig oversikt over koden: «Listenavn[0].RoadNumber» og «Listenavn[0].TextDescription». Tilsvarende endringer gjøres alle steder i koden der vi har nestede lister og lister med array.

Vi tilpasser også koden slik at objektet som returneres til klientsiden, inneholder et eget objekt som kan sendes direkte inn i HRESSURS. Til nå har det kun blitt returnert et objekt med all informasjon, men for at informasjonen enklere skal kunne legges til i reiseregningene, lagrer vi nå informasjonen som skal til HRESSURS i et eget objekt. Dette objektet består av to objekter, et med innhold i forbindelse med kjørestrekningen (start, stopp, via, avstand og komprimert kjørebekrivelse) og et med innhold i forbindelse med bomutgifter og kostnader (navn på bomstasjoner, priser per bomstasjon og totalpris). Grunnen til dette er at kjøring og utgifter håndteres to helt forskjellige steder i HRESSURS.

Alle for-løkker i koden, endres til foreach, da dette optimaliserer koden. Ved for-løkker der man sjekker lengden på en liste/et array, kjøres denne beregningen for hvert trinn i løkka. Ved bruk av foreach beregnes lista kun én gang før programmet går i gjennom hele lista/arrayet.

