

# Avstand

og bompengeberegningsmodul til HRessurs

## Bacheloroppgave:

Avdeling for informasjonsteknologi

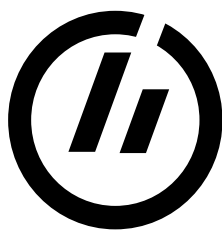
Høgskolen i Østfold

21.05.2015



• Gruppe BO-G02 • Robin Furu • Ingvild Karlsen Bjørlo • Christian Jacobsen • Glenn Bjørlo





# HØGSKOLEN I ØSTFOLD

Avdeling for Informasjonsteknologi  
Remmen  
1757 Halden  
Telefon: 69 21 50 00  
URL: [www.hiof.no](http://www.hiof.no)

## BACHELOROPPGAVE

Prosjektkategori: <b>Bachelorprosjekt</b>	<input checked="" type="checkbox"/>	Fritt tilgjengelig
Omfang i studiepoeng: <b>20</b>	<input type="checkbox"/>	Fritt tilgjengelig etter
Fagområde: <b>Informasjonsteknologi</b>	<input type="checkbox"/>	Tilgjengelig etter avtale med oppdragsgiver

Tittel: <b>Utvikling av reiseregningsmodul</b>	Dato: <b>24. april 2015</b>
Forfatterere: <b>Robin Furu, Ingvild Bjørlo Karlsen, Christian Jacobsen, Glenn Bjørlo</b>	Veileder: <b>Terje Samuelsen</b>
Avdeling / Program: <b>Avdeling for Informasjonsteknologi</b>	Gruppenummer: <b>BO15-G02</b>
Oppdragsgiver: <b>Infotjenester AS</b>	Kontaktperson hos oppdragsgiver: <b>Petter Ekran</b>

### Ekstrakt:

HRESSURS skal utvides med en mulighet for å beregne avstand og bompengekostnader under føring av reiseregninger, slik at brukeren ikke skal måtte ta i bruk eksterne verktøy for å gjøre dette. Denne må ta hensyn til statens krav for føring av reiseruter, som innebærer at kjørerutene skal beskrives detaljert, for senere kontroll/etterdokumentasjon av reiseregningene. Brukeren skal kunne angi type fremkomstmiddel, og en detaljert beskrivelse av kjørt rute, i form av start og endepunkt for ruta, og punkter underveis på strekningen. Basert på angitt informasjon, skal programmet beregne avstand og bompengekostnader for ruta. For at brukeren skal kunne kontrollere at programmet har beregnet riktig kjørerute, skal programmet skrive ut en kort veibeskrivelse av ruta.

3 emneord:

HRESSURS
Reiseregning
Webutvikling



# Innhold

<b>Figurliste</b>	<b>5</b>
<b>1 Introduksjon</b>	<b>1</b>
1.1 Prosjektgruppen . . . . .	1
1.2 Oppdragsgiver . . . . .	2
1.3 Oppdraget . . . . .	4
1.4 Formål, arbeidsmetode og Leveranser . . . . .	5
1.5 Rapportstruktur . . . . .	8
<b>2 Bakgrunnsanalyse</b>	<b>10</b>
2.1 Eksisterende løsninger samt regler og hensyn. . . . .	10
2.2 Teknologi . . . . .	18
<b>3 Planlegging - Struktur og design</b>	<b>21</b>
3.1 Kodemessig struktur . . . . .	21
3.2 Brukergrensesnitt . . . . .	23
<b>4 Utvikling av applikasjonen - steg for steg</b>	<b>25</b>
4.1 Google og konvertering . . . . .	25
4.2 Alt i ett løsning . . . . .	26
4.3 HTML - Web Service Versjon 1 . . . . .	26
4.4 HTML - Web Service Versjon 2 . . . . .	28
4.5 Utforming av viapunktliste . . . . .	29
4.6 Veien mot endelig brukergrensesnitt . . . . .	37
4.7 Fremstilling av beregnet kjørerute . . . . .	40
4.8 Tilpassninger til HRessurs - Veien mot implementering . . . . .	48
4.9 Publisering på skolens server . . . . .	51
4.10 Implementering . . . . .	51
4.11 Hvordan kan applikasjonen videreutvikles? . . . . .	52
4.12 Programmer, verktøy og hjelpemidler . . . . .	52
4.13 SCRUM - utførelse . . . . .	54
<b>5 Test av Applikasjon</b>	<b>55</b>
5.1 Brukerevaluering . . . . .	55
5.2 Resultater av brukerevaluering: . . . . .	57
<b>6 Diskusjon</b>	<b>58</b>
<b>7 Konklusjon</b>	<b>59</b>
<b>Register</b>	<b>63</b>



# Figurer

1.1	Oversikt over Infotjenesters produkter . . . . .	3
1.2	Eksisterende løsning . . . . .	4
1.3	Vår tenkte løsning . . . . .	5
1.4	Grafisk fremvisning av tenkt metoder. . . . .	7
1.5	Ganttdiagram - Planlagt fremdrift . . . . .	8
2.1	Skjermdump fra HRESSURS . . . . .	10
2.2	Eksisterende ruteplanleggingstjenester . . . . .	14
3.1	Vedlikeholdbar struktur . . . . .	22
3.2	Skal utbedres. Skisse som illustrerer utskiftbarhet . . . . .	22
3.3	Tidlig eksempel på GUI . . . . .	23
3.4	Tidlig eksempel på GUI . . . . .	24
4.1	Viser eksempel på hvordan man henter Google API Autocomplete basert på steder. . . . .	25
4.2	Strukturen på CalculatedRoute . . . . .	27
4.3	Strukturen fra Web Service til Ruteplantjenesten . . . . .	28
4.4	Strukturen på interfacet . . . . .	29
4.5	Viapunkter, CSS og Javascript . . . . .	31
4.6	Viapunkter til å manipulere . . . . .	31
4.7	Viapunkter med ett input-felt. . . . .	34
4.8	Viapunkter med ett input-felt og en egen ul-liste. . . . .	35
4.9	Tidlig fremstilling av Brukergrensesnitt . . . . .	37
4.10	HRESSURS design . . . . .	38
4.11	Tidlig brukergrensesnitt . . . . .	38
4.12	Innhold Tabvisning . . . . .	39
4.13	Tekslig beskrivelse av kjøreruta. . . . .	40
4.14	Gruppert, tekslig beskrivelse av kjøreruta. . . . .	41
4.15	Kjørerute fram og tilbake på E6 . . . . .	44
4.16	Polyline er jevnt forskjøvet iforhold til riktig kjørerute . . . . .	44
4.17	Googles snap-to-road på polyline . . . . .	45
4.18	Polyline følger kjørerute, uten forskyvning . . . . .	46
4.19	Sarpsborg - Oslo, fremstilt med viapunkter . . . . .	47
4.20	Halden - Oslo, fremstilt som polyline . . . . .	47
5.1	Eksempel fra Skjema og fremvisning . . . . .	56





# 1. Introduksjon

I dette kapittelet vil vi gi en introduksjon til oss, arbeidsgiver samt vårt oppdrag og arbeidsmetode.

## 1.1 Prosjektgruppen

Glenn og Christian jobbet for Infotjenester høsten 2014 i forbindelse med faget Bedriftspraksis. De fikk en forespørsel om å fortsette ved bedriften i forbindelse med bacheloroppgaven. Dette takket de ja til, og tok så med seg Ingvild og Robin på gruppa. Christian, Ingvild og Robin har tidligere jobbet sammen i labgrupper i faget Industriell IT, og på et eksamensprosjekt i faget Integrerte IT-Systemer, høsten 2014. Her jobbet de med oppkoblingen av passivhuset «Villa Mjølerød», i regi av Bolig Enøk. Dette gikk ut på å programmere smarthuskomponenter som skulle brukes i huset

### **Ingvild Karlsen Bjørlo**

Invgilkb@hotmail.com - tlf: 90204584 , Ingvild bor i Halden, hvor hun også er oppvokst. Hun gikk studiespesialisering på Halden videregående, med et utvekslingsår i Tyskland, og jobbet i et år før hun søkte seg til dataingeniørstudiet. Hun har tidligere jobbet 2,5 år på Svinesund Infosenter, som i tillegg til turistinformasjon, fungerer som servicestasjon for bompengeselskapet Svinesundsforbindelsen. De faglige interessene retter seg mot programmering og dokumentasjon

### **Robin Furu**

robin.furu@hotmail.com - tlf: 41524376, er oppvokst i Fredrikstad, Gikk Idrettslinjen på videregående, hvor det så bar videre til et års tjeneste i Hans Majestet Kongens Garde. Rett ut fra førstegangstjenesten bar det videre til Tre-semester Dataingeniørkurs hvor det først var en sommer med matte og fysikk. Han har gjennom studiene blitt veldig interessert i HMI/automatisering – Industriell IT, en mer praktisk tilnærming til IT.

### **Christian Jacobsen**

christian.jacobsen@hotmail.no - tlf: 41758575, er født og oppvokst i Fredrikstad og bor på Gressvik. Han har tidligere erfaring som tekniker i telekomfaget hvor han jobbet i YIT Building systems i 2 år og har fagbrev i telekommunikasjon. Ved siden av studiene jobber han på Expert, og har foto som en stor interesse.

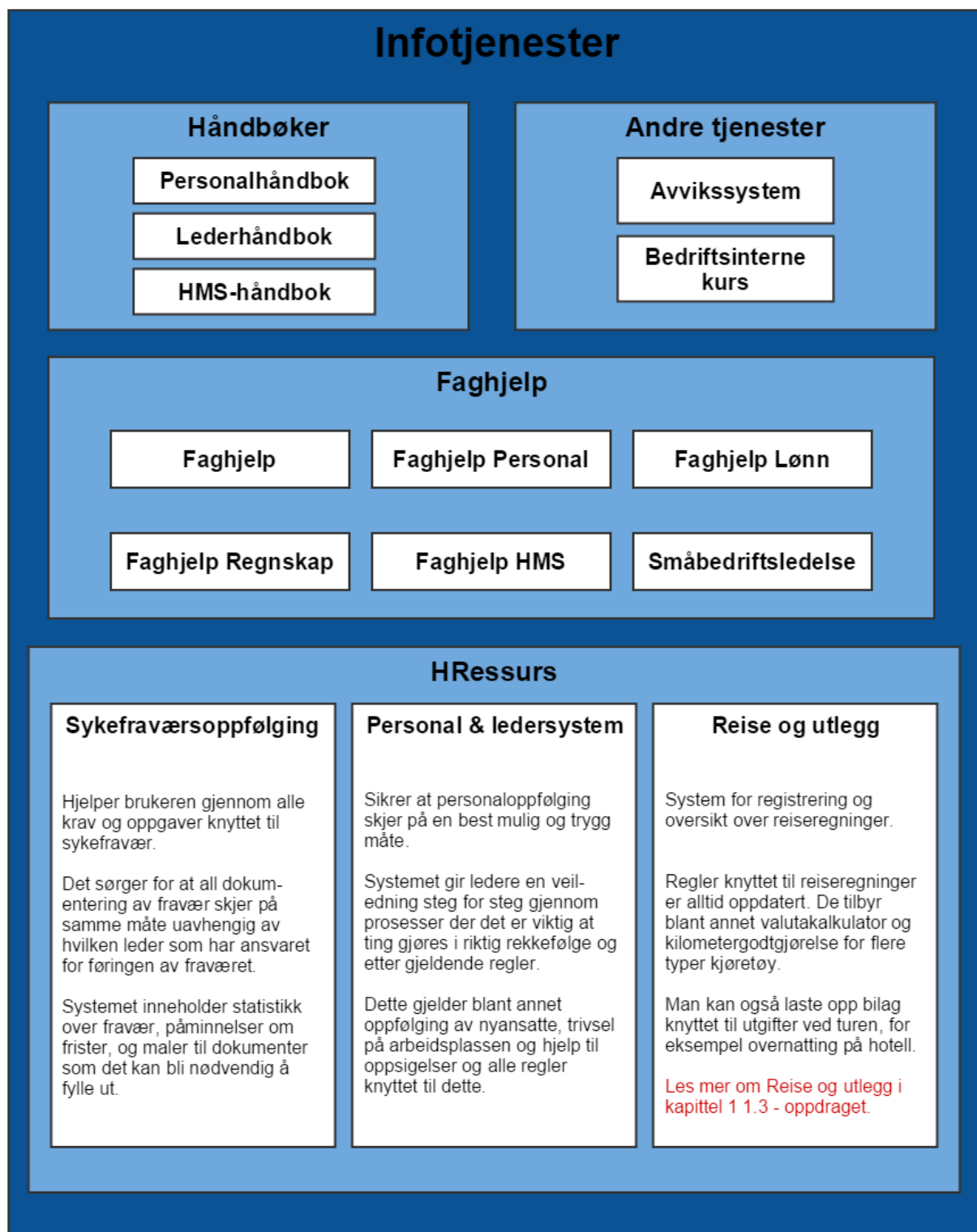
### **Glenn Bjørlo**

g-amb@hotmail.com - 47384852, er oppvokst i Askim og har bodd i Halden siden høsten 2014. Han ble uteksaminert fra Askim Videregående skole i 2005 og flyttet deretter til Lillehammer hvor han studerte film- og fjernsynsvitenskap. Senere flyttet han til Sarpsborg og begynte studiet DMpro. Han har nylig kommet hjem fra ett utvekslingsår i Australia.

## 1.2 Oppdragsgiver

Infotjenester i Sarpsborg er et IT/konsulent firma som siden 1985 har levert faglig og juridisk kompetanse til norsk arbeidsliv. Deres hovedfagområder er innen personal, ledelse, HMS, lønn og Regnskap. De har i dag ca 150 ansatte og et datterselskap i Sverige, Stage Competence. De har en ledende posisjon når det gjelder kurs og nettbaserte fag- og kompetanseverktøy som skal gjøre det enklest mulig for bedrifters ledelse og arbeidere. Deres kompetanseverktøy er egenutviklet, de stiller derfor også opp med kundesupport og konsulenttjenester i forbindelse med dette. Sammensetningen av kompetanse hos de ansatte innen jus, HR og systemutvikling gjør at de kan utvikle gode og praktiske systemer i henhold til regelverk og brukervennlighet. De har et stort nettbasert oppslagsverk med fagsupport, med mer enn 40.000 brukere. Flere hundre tusen ledere og medarbeidere har tilgang til deres nettbaserte håndbøker og systemer. De tilbyr i tillegg rådgiving utover oppslagsverkene, og deres rådgivere er alle enten jurister eller fagspesialister med lang erfaring innen både offentlig og privat sektor. De svarer på mer enn 50 000 spørsmål hvert år.

Et av deres mest populære produkter er det egenutviklede personalsystemet HRessurs, der man kan behandle alt fra personaladministrering, sykefravær, ferie og reiseregninger. Dette er et nettbasert system, slik at kundene abonnerer på tilgang til nettløsningen og ikke trenger å kjøpe og installere programvare. Dette fører til at systemet alltid er oppdatert, uten at kundene må tenke på å laste ned oppdateringer. Kundene kan velge mellom tre ulike pakker ettersom hva bedriften trenger: HRessurs Sykefraværsoppfølging, HRessurs personal- og ledersystem og HRessurs Reise og Utlegg. I denne oppgaven er det HRessurs Reise og Utlegg, som behandler reiser og alle utgifter knyttet til dette, vi jobber med. Se oversikt over alle Infotjesters produkter og systemer på neste side, med vekt på HRessurs. ref. Figur 1.1



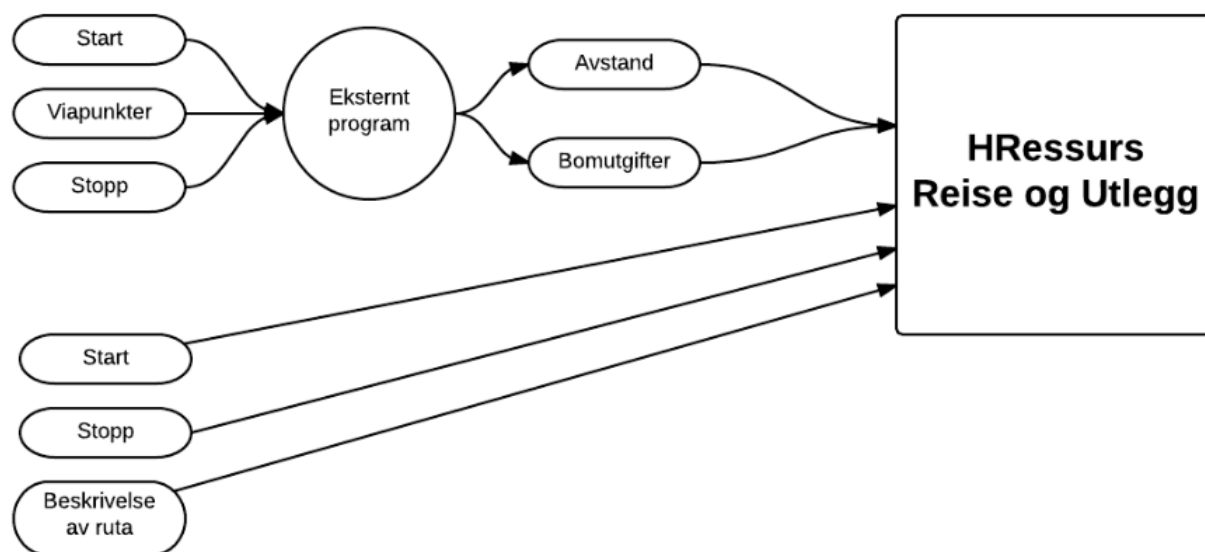
Figur 1.1: Oversikt over Infotjenesters produkter

## 1.3 Oppdraget

Infotjenester lover på sine nettsider et «brukervennlig og intuitivt grensesnitt som gjør den ansatte i stand til å registrere/behandle reiser og utlegg uten opplæring. Brukeren trenger ingen kunnskaper om regler og satser». Infotjenester har derimot fått tilbakemeldinger på at modulen ikke er brukervennlig nok, blant annet med hensyn til bompenger. Vår utfordring er å utbedre en av manglene som er årsaken til misnøyen blant enkelt kunder

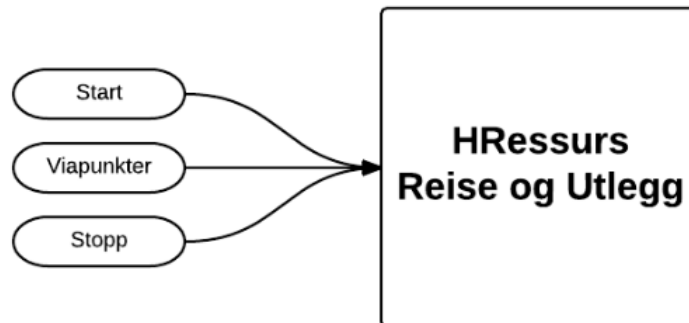
HRESSURS skal utvides med en mulighet for å beregne avstand og bompengestnader under føring av reiseregninger, slik at brukeren ikke skal måtte ta i bruk eksterne verktøy for å gjøre dette. Denne må ta hensyn til statens krav for føring av reiseruter, som innebærer at kjørerutene skal beskrives detaljert, for senere kontroll/etterdokumentasjon av reiseregningene. Brukeren skal kunne angi type fremkomstmiddel, og en detaljert beskrivelse av kjørt rute, i form av start og endepunkt for ruta, og punkter underveis på strekningen. Basert på angitt informasjon, skal programmet beregne avstand og bompengestnader for ruta. For at brukeren skal kunne kontrollere at programmet har beregnet riktig kjørerute, skal programmet skrive ut en kort veibeskrivelse av ruta.

Dette er dagens løsning ved bruk av ekstern teknologi:



Figur 1.2: Eksisterende løsning

Merk at et annet alternativ til skissen over er at man selv noterer bomstasjoner på ruta, og leser avstanden av kilometerstanden. Slik er målet at beregninger av reiseregninger skal foregå:



Figur 1.3: Vår tenkte løsning

Løsningen vil være en bedre løsning både for de ansatte som kjører og firmaet, da beregningen foretas automatisk i det de registrerer reiseregningen. Løsningen er bedre for de ansatte som kjører, da de slipper unna masse beregninger, og de får tilbake det de legger ut, og firmaet betaler kun ut det beløpet de skal, fremfor et beløp beregnet på hukommelsen til den ansatte. Det viktigste målet er at registreringen av bompengeutgifter på reiser i forbindelse med arbeidet skal bli enklere, og mer korrekt.

Ettersom dette er en tilleggsmodul til en eksisterende løsning, er det med tanke på fremtidig drift og vedlikehold, en stor fordel at løsningen er bygd opp med samme struktur som det eksisterende. Det at HRESSURS er en løsning firmaene betaler for tilgang til, setter også en del ekstra krav til standarden det må følge. HRESSURS forutsetter ikke at brukeren har oversikt over til en hver tid gjeldene regler for reiseregninger, så dette må også tas hensyn til. Hvordan dette skal løses, og hvilke krav løsningen må følge, vil vi komme nærmere til i analysedelen i kapittel 2.1.

## 1.4 Formål, arbeidsmetode og Leveranser

### 1.4.1 Formål

Med denne oppgaven ønsker vi å utvikle en applikasjon til HRESSURS som skal gjøre det enkelt for brukere å føre reiseregninger selv uten å være avhengig av å skrive ned hver eneste bomstasjon man passerer. Vi skal derfor få til funksjonalitet som lister opp ruten for bruker utifra Fra, til og viapunkter som han har, samtidig skal vi prøve å få frem hvor mange bomstasjoner man har vært igjennom med en total kostnad.

**Hovedmål**

Gjøre reiseregning/beregning lettere

**Delmål 1**

Kunne beregne kostnadene og avstander

**Delmål 2**

Kunne beskrive ruta detaljert nok til å etterkomme reglene for føring av reiseregninger

## 1.4.2 Arbeidsmetode

Arbeidsmetodene vi planlegger å benytte er en kombinasjon av individuelt arbeid, og samarbeid i gruppe. Vi planlegger å jobbe sammen to til tre dager i uka fra skolen, og resten av tiden individuelt. Vi vil kommunisere i en gruppechat på facebook, og prosjektfiler, både dokumenter og kode, vil vi laste opp i en fellesmappe på dropbox, slik at alle hele tiden har tilgang på de nyeste versjonene. Vi planlegger også å bruke github for ekstra back up av data, og for at oppdragsgiver kan se filene. Da dette er et prosjekt som både innebærer en del undersøkelser og programmering, vil vi prøve å få til en oppdeling av prosjektet slik at alle best mulig kan arbeide parallelt, og ingen må sitte å vente på å kunne gjøre noe. For at flere av oss skal kunne jobbe samtidig med programmeringen, vil vi prøve å dele opp slik at vi jobber på forskjellige deler av applikasjonen, før vi samarbeider om å sette disse sammen. I felleskap med arbeidsgiver har vi kommet frem til at bruk av Scrum metoden er det som passer oss best når det kommer til selve utviklingen av applikasjonen.

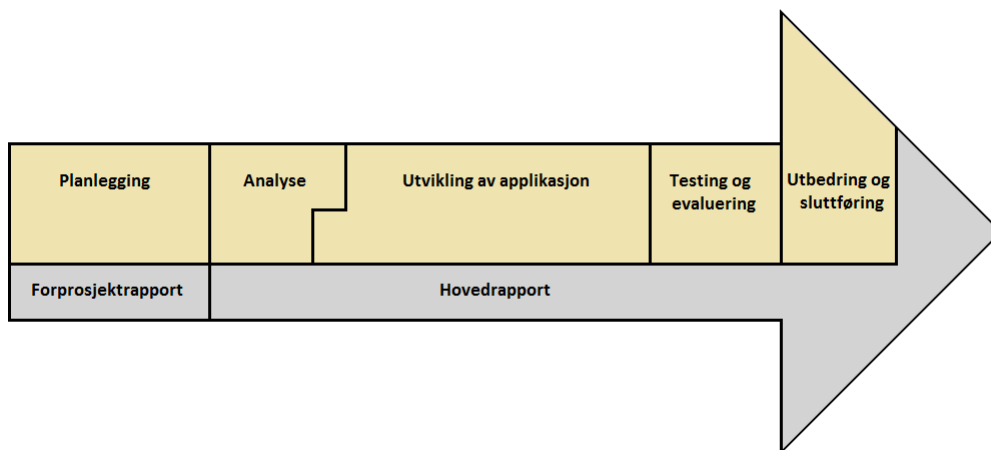
Vi vil derfor bruke en forenklet versjon av SCRUM, da denne metoden er mye brukt i forbindelse med programmering og systemutvikling. I hovedsak går SCRUM ut på at det ikke settes konkrete kravspesifikasjoner fra begynnelsen, men at prosjektideen hele tiden videreutvikles gjennom prosjektperioden. Det utvikles hele tiden ny teknologi, og det som ved prosjektoppstart kan ha vært en revolusjonerende ide, kan underveis i prosjektperioden bli erstattet av nye og bedre løsninger dersom utviklingsfasen strekker seg over lang tid. Vi har en kortere utviklingsfase og et klart mål for hva resultatet skal gjøre, men vi vil ta i bruk hovedelementene i scrum da det ikke er fastsatt hvordan applikasjonen skal utvikles. Vi vil definere «user stories» som er små delmål som beskriver hva som skal kunne gjøres fra brukerens perspektiv, samt for å definere hvilken funksjonalitet vi bør fokusere på. Vi vil sette opp mål for hva som skal gjøres for to og to uker av gangen, kalt sprinter. Ved hver sprintstart/-slutt, vil vi ha et møte med produkteier(oppdragsgiver) hvor vi i felleskap blir enig om hvilke user stories vi skal fokusere på i neste sprint. Alle definerte user stories som ikke er tildelt sprinter er lagret i en back log.

Elementer fra scrum er integrert i Gant diagrammet vårt ref.kap. 1.4.3

Vi bruker metoden for å ha et mer dynamisk arbeidsperspektiv på løsningen, da det hele tiden kan komme endringer eller synspunkter i forhold hvordan vi kan løse det på en bedre måte, eller at arbeidsgiver ønsker ekstra funksjoner i applikasjonen. Til å holde styr på back log, sprinter og user stories benytter vi en felles Github, hvor vi lagrer alt

under Issues. Ved å annenhver uke bestemme hva som skal gjøres videre, vil vi ha større forutsetninger til å få en løsning både vi og oppdragsgiver er fornøyd med. Etter å ha utviklet applikasjonen skal denne testes, slik at vi får utbedret eventuelle feil og mangler vi selv ikke oppdager. Vi vil foreta testing på bekjente med varierende dataerfaring og -kompetanse, og på bakgrunn av tilbakemeldingene vil vi utbedre og ferdigstille applikasjonen.

I hovedsak kan prosjektperioden deles inn i fem klare faser, se ref. *Figur 1.4* Hver av fasene blir nærmere beskrevet de påfølgende avsnittene. Den første er oppstarten med planlegging og undersøkelse av gjennomførbarhet, før analysefasen der vi vil foreta analyser av tilsvarende, eksisterende løsninger og finne ut hvordan vi best mulig bør gå fram for å løse oppgaven. Parallelt med at noen av oss fullfører siste del av analysedelen som er beskrevet ovenfor, vil de resterende på gruppa starte med utviklingen av applikasjonen. Når oppdragsgiver er fornøyd med resultatet, skal vi gjennomføre testing av applikasjonen, før vi ferdigstiller den ved å utbedre feil og mangler som kommer frem under testingen. For å slippe tekniske problemer på slutten, og risikere at vi ikke kommer i mål med applikasjonen, planlegger vi å ferdigstille applikasjonen i løpet av april, slik at vi kan ha fullt fokus på rapporten frem til levering



Figur 1.4: Grafisk fremvisning av tenkt metoder.

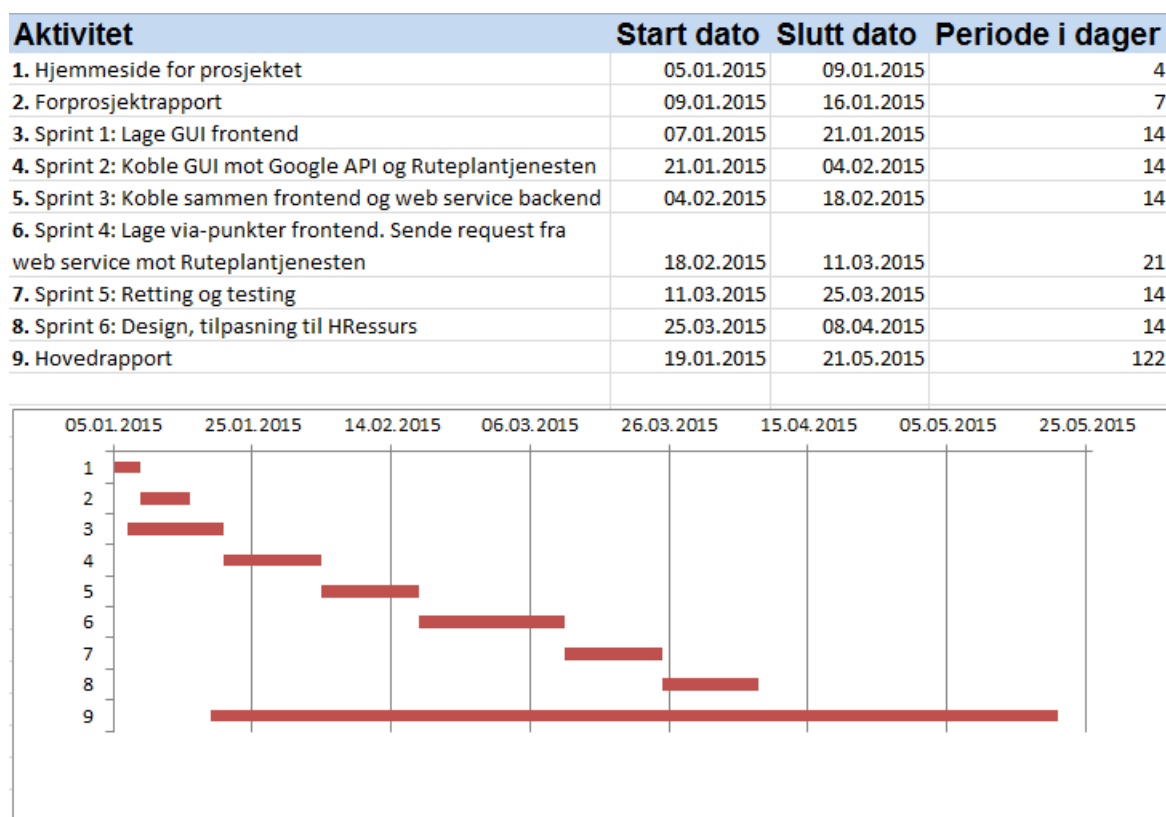
Rapporten vil vi skrive jevnt gjennom hele prosjektet, fra forprosjektrapporten er levert og til rapporten skal leveres. Siden rapporten spiller en veldig stor del av prosjektet, vil vi fordele slik at vi hele tiden har minst en på gruppa som har hovedansvar for å skrive rapporten, mens resten programmerer. Alle har hele prosjektperioden ansvaret for å skrive bidrag til «kapittel 4 – gjennomføringen» i rapporten om hva han/hun har gjort, etterhvert som det gjøres.

De første ukene vil være en typisk oppstartsfase, der vi vil bruke en del tid på planlegging og forprosjektrapporten, i tillegg til at alle må sette seg inn i programmeringsspråkene oppgaven baserer seg på. I denne perioden skal vi også finne ut om prosjektet er gjennomførbart, blant annet ved å finne ut hvordan vegvesenets API fungerer, om vi finner all informasjon vi trenger der, eller om det finnes andre bedre alternativer. Dersom prosjektet viser seg å bare delvis la seg gjennomføre, eller ikke lar seg gjennomføre i det hele tatt, må vi bruke

denne perioden til å finne andre alternativer for å innhente og beregne data. For å se at det lar seg gjøre vil vi lage en veldig enkelt testversjon. Vi skal så begynne med analyse for å se på hvilke hensyn vi må ta i forhold til den eksisterende løsningen i HRessurs, hvilke regler som gjelder for reiseregninger, hvilke krav til universell utforming som må dekkes og hvordan vi best mulig kan løse oppgaven programmeringsmessig.

Når vi har funnet ut at prosjektet lar seg gjennomføre, og bestemt hvordan vi skal løse det, skal vi begynne å utvikle applikasjonen.

### 1.4.3 Fremdriftsplan



Figur 1.5: Ganttdiagram - Planlagt fremdrift

## 1.5 Rapportstruktur

### Kapittel 2 - Bakgrunnsanalyse

Bakgrunnsanalyse kapittelet inneholder generell beskrivelse av type teknologi som er brukt i prosjektet, samt hvorfor vi har valgt akkurat den type teknologi og utførelse, og fordeler og ulemper ved de forskjellige metodene gjennom utvikling



### **Kapittel 3 - Design utforming og Planlegging**

Under dette kapitlet finner du utdypende informasjon om prosjektetsoppbygging og design. Her er alt listet fra de diverse komponentene som inngår i prosjektet ( Backend / frontend ) samt hva som forventes og levere og de forventede leveransene vi skal gjennomføre i løpet av prosjektet. Det beskriver også hvilken type utførelse som er valgt og hva vi har gjort av research og gjennomført av informasjonsinnhenting.

### **Kapittel 4 - Implementasjon/produksjon/gjennomføring**

dette er ikke ferdig beskrevet enda.

### **Kapittel 5 - Test av applikasjon**

I dette kapitlet beskriver vi resultatene rundt testing av systemet mot et knipe tilfeldige brukere.

### **Kapittel 6 - Diskusjon**

I dette kapitlet diskuterer vi hva vi har fått ut av oppgaven, hva vi har lært og utfordringer vi har støtt på.

### **Kapittel 7 - konklusjon**

Sammendrag av prosjektet, hva som ble oppnådd og evt hva som ikke ble gjennomførtbart.

## 2. Bakgrunnsanalyse

I dette kapittelet tar vi for oss hvilke hensyn som må gjøres og hvilke eksisterende løsninger som allerede finnes, samt hvordan vi tenker å bygge opp programvaren vår.

### 2.1 Eksisterende løsninger samt regler og hensyn.

#### 2.1.1 HRessurs i dag, brukergrensesnitt og sideoppsett

Vi vil nå gå nærmere inn på hvordan HRessurs er designet i dag, slik at vi vet mer om hvilke hensyn vi må ta når vi tilpasser brukergrensesnittet vårt. *Figur 2.1* viser et skjermbilde av dagens HRessurs.

Figur 2.1: Skjermdump fra HRessurs

Brukeren får tilgang til reise-modulen ved å logge seg inn på sin konto og deretter velge «mine reiser». Her får brukeren valget mellom å se på allerede registrerte reiser, eller legge til en ny. Registreringen foregår trinn for trinn, og det er enkelt for brukeren å holde oversikten over hvor han er. Det er både en meny som forteller hvor brukeren befinner seg i registreringsprosessen, samt en frem- og tilbakeknapp nederst på skjermen som viser antall trinn. Knappene i menyen øverst på skjermen er formet som piler som tydelig viser hvor man er samt at det gir inntrykk av at man befinner seg i en prosess som foregår stegvis. Det er ikke mulig å navigere seg frem før man har fylt ut nødvendige felt. Antall trinn avhenger av om man har benyttet eget kjøretøy, om man skal ha diettgodtgjørelse eller om man har vært i utlandet. Hukes det av for noen av disse dukker det opp et nytt trinn hvor man må fylle inn informasjon for dette valget. Hver side i applikasjonen er

minimalistisk designmessig. Det er hvit bakgrunn og samme teksttype både i hoveddelen, menyen og knappene. Bakgrunnsfargen i menyen er grå, og blå i knappene på siden. Dette er det eneste som skiller ulike komponenter og funksjoner fra hverandre.

På første side må brukeren navngi reisen, fylle inn dato og klokkeslett for avreise og hjemkomst, hva som er formålet med reisen og reisested. Når man navngir hver enkelt reise er det også enklere og holde oversikten for brukeren når antall reiser registrert begynner å bli mange. Har brukeren huket av for diettgodtgjørelse dukker det opp to nye trinn hvor man skal legge inn overnattingssted, ankomst og avreise. På neste trinn kan brukeren legge til måltidsfradrag. Andre utlegg i forbindelse med overnatting og mat på reiser, slik som diettgodtgjørelse og måltidsfradrag osv er ikke relevant i forhold til vår applikasjon.

Hvis det har vært huket av for eget kjøretøy blir neste trinn registrering av reiserute. Ruten må oppgis så nøyaktig som mulig i et eget tekstfelt. For eksempel: Sarpsborg – Oslo via E6 til Tusenfryd, E18 til Oslo. I neste tekstfelt må brukeren oppgi nøyaktige antall kilometer. Brukeren kan ikke velge reiserute dynamisk, for eksempel ved å fylle ut fra- og til-felt som autofullføres og hvor avstanden regnes ut automatisk. Han kan heller ikke velge eventuelle via-punkter som legges inn automatisk. Utfra et brukerperspektiv vil dette kanskje virke altfor gammeldags og tungvint, samt gi inntrykk av lite automatisering. Neste steg blir å velge type kjøretøy fra en nedtrekksmeny. Deretter kan man eventuelt fylle ut informasjon om kjøring utland, årsak til omkjøring, medpassasjerer og andre spesielle tillegg. På tredje trinn kan brukeren fylle ut utlegg. Man må beskrive type utlegg, dato for utlegget og beløpet. Brukeren kan også laste opp eventuelle bilag. På nest siste trinn kan brukeren fylle inn ekstra informasjon, som om han har mottatt forskudd, og kostnadsbærere. På siste side får brukeren opp en detaljert oversikt over all informasjon som har vært fylt ut. Øverst på siden ligger en kort oversikt over registrert informasjon som startdato og varighet, eventuelle utlegg i kroner, og hvor mye som skal utbetales. Brukeren kan også se nåværende status på registreringen. Altså om den venter på innsendelse eller om den er godkjent/ ikke godkjent. Lenger ned på siden vises en digital versjon av selve reiseregningen med all nødvendig informasjon. Brukeren kan nå velge å signere og sende inn, eller gå tilbake og gjøre oppdateringer.

### 2.1.2 Bomstasjoner - rabattordninger og regler?

I Norge øker antall bomstasjoner stadig. Blant annet langs E6 mellom Gardermoen og Hamar har det på få år kommet fem bomstasjoner, og en sjette åpnes i juni 2015<sup>1</sup>. Senest 3.februar 2015 var det nevnt i Halden Arbeiderblad 2 muligheter for bompenger i Halden i forbindelse med bygging av to tuneller i fjellet under Fredriksten Festning. «– En bompengefinansiering gjør at man kan komme fortere i gang med den type prosjekter. Halden bør ta en runde rundt bompengefinansiering, mener han. Ordfører Thor Edquist tror ikke vi kommer utenom bompenger». Fra og med 1.januar er det obligatorisk for biler med tillat totalvekt over 3,5t å ha autopass i Norge. Regelverket gjelder alle biler som tilhører «foretak, stat, fylkeskommune eller kommune, eller som på annen måte hovedsakelig benyttes i næringsvirksomhet»<sup>3</sup> Straffen for å ikke følge disse reglene er bot på 8000kr, og denne satsen doubles dersom pålegget ikke etterkommes, og kjøretøyet stoppes uten brikke igjen innen 2år fra første straff. Til tross for påbudet om autopassbrikke i kjøretøy over 3,5t, vil vi når bompengekostnadene beregnes, ikke ta hensyn til eventuelle autopassavtaler de ansatte måtte ha. Dette er med tanke på at priser, rabatter og betalingspraksis varierer veldig over hele Norge, slik vi i de påfølgende avsnitt vil gå nærmere inn på

4

Det er i Norge slik at alle bomstasjoner er tilknyttet bompengeselskap. Det er ikke et selskap som drifter alle bomstasjoner i Norge, men 44 forskjellige, lokale selskaper som drifter hver sitt/sine bomanlegg<sup>5</sup> Det finnes ingen standardtakster og rabattavtaler som er gjeldene for alle. Hvert selskap bestemmer selv om de skal ha rabatt til alle med autopass, rabatt kun til egne abonnenter osv. Om bompengeselskapene tilbyr forskuddsbetaling, etterskuddsbetaling eller begge deler er også opp til hvert enkelt selskap, og rabattene varierer gjerne ut ifra betalingsalternativene de tilbyr.

Østfold Bompengeselskap,<sup>6,7</sup> som drifter seks bomstasjoner i Østfold, tilbyr forskuddsbetaling. Avhengig av beløpet som forskuddsbetales, vil størrelsen på rabatten for passeringer ved bomstasjonene de drifter, variere. Ved påfyll av 805kr vil rabatten være 30%, 3450kr vil gi 40% rabatt og 5750kr vil gi 50% rabatt. De tilbyr kun rabatt til sine egne abonnenter, og alle andre passerende med autopass fra andre selskaper, vil måtte betale full pris. Et annet selskap her i Østfold, som har en helt annen praksis, er Svinesundsforbindelsen<sup>8</sup> De tilbyr kun etterskuddsbetaling, og gir 13% rabatt per passering for alle som har autopass. Tilsvarende rabatter for alle med autopass finnes blant annet ved Bomringen i Oslo, Kråkerøyforbindelsen, Bomringen i Kristiansand og flere andre selskaper. Nord-Jæren Bompengeselskap<sup>9</sup> er et selskap som tilbyr både forskudd og etterskuddsbetaling for sine abonnenter, men her fordeler rabatten seg på 10% for privatkunder med etterskuddsbetaling, og 20% for alle med forskuddsbetaling, samt firmakunder med etterskuddsbetaling.

Det er ikke bare ulik praksis med autopassavtale som skiller bompengeselskapene. I Trondheim praktiseres rushtidspriser, som vil si at i tidsrommet 07-09 og 15-17 doubles taksten for å passere bomstasjonene i Trondheim sentrum. Disse tilhører prosjektet «Miljøpakke Trondheim»<sup>10</sup> Resterende bomstasjoner i Trondheimområdet følger ikke rushtidpraksisen. Miljøpakken er et prosjekt som ble startet i 2009 for å begrense miljøproblemene i forbindelse med at Trondheimsområdet er et av områdene i Norge med størst befolkningsvekst<sup>11</sup>

Det gjøres en rekke tiltak for å blant annet redusere klimautslipp ved hjelp av kortere bilkøer og mindre trafikkstøy, derav å prøve å redusere de lange køene i forbindelse med rushtidstrafikken. I Trondheim finnes det også tidsgrenser for om man skal betale for passeringene eller ikke. Bomstasjonene er grupperte i seks grupper, det Miljøpakken omtaler som «snitt». Hvert av snittene inneholder mellom en og ni bomstasjoner, og innenfor et tidsrom på en time, betaler man kun for en passering i hvert snitt <sup>12</sup> Denne praksisen finner man blant annet igjen i Haugalandspakken <sup>13</sup> som dekker bomstasjoner i Haugesund og på Karmøy, der man også kun betaler en passering innenfor en time.

Med så mange forskjellige rabattordninger og varierende priser innenfor et tidsrom, vil det være tilnærmet umulig å ta hensyn til hver enkelt bruker av HResurs og en eventuell autopassavtale han eller hun måtte ha. Etterskuddsfakturerings av passeringer og utsendelse av oversikt over passeringer for de med forskuddsbetaling, skjer gjerne en tid etter passeringen fant sted. Hyppigheten for utsendelse varierer fra bompengeselskap til bompengeselskap, samt hvilke avtaler man har. Noen selskaper praktiserer månedlig etterskuddsfakturerings, og andre sender en oversikt over passeringer når forhåndsbetalt beløp er brukt opp. Passeringene kan også være vanskelig å holde oversikt over, da disse ofte har lokale navn, som ikke sier den reisende så mye uten å være lokalkjent. Bedrifter har også ofte en månedlig frist for å registrere reiseregninger for måneden som har vært. Dette tilsier at man i de fleste tilfeller ikke har tid til å vente på faktura/oversikt over passeringer, og et annet viktig argument er statens regler i forhold til refusjon av bompenger. Dette er utgifter det ikke kreves kvittering på, og på altinn.no står følgende: «For enkelte småutgifter som bompenger, parkometeravgift og lignende er det ikke nødvendig med originalbilag. Det holder at utgiften spesifiseres på reiseregningen eller et eget oppsett.» <sup>14</sup> Dette er bakgrunn for at det istedenfor å dokumentere hver enkelt passering, med hver enkelte brukes rabatter, kan foretas en generell prisberegning ut ifra vanlige takster.

### 2.1.3 Eksisterende ruteplanleggingstjenester

Når vi var i første møtet med arbeidsgiver fikk vi presentert et API fra vegvesenet som gjør det mulig å beregne bomutgifter og kjørelengde. Den webaserte tjenesten deres <sup>15</sup> virker ikke helt optimal, så vi bestemmer oss for å undersøke mulighetene våre ved å gjøre research rundt tilsvarende, eksisterende løsninger. Vi tar for oss andre nettbaserte tjenester for å beregne kjøreruter, for å se hvilken tilleggsinformasjon de kan tilby utover å beregne kjøreruter. Vi tester NAF Ruteplanlegger <sup>16</sup> og veibeskrivelsestjenesten hos 1881 <sup>17</sup>, Google Maps <sup>18</sup>, Gule sider <sup>19</sup> og Kvasir <sup>20</sup>. *Figur.2.2* viser en grafisk oversikt over tjenestene vi har testet og hvilke funksjoner hver av de tilbyr.

Felles for alle er at de oppgir kjørelengde for strekningen mellom to steder, samt at de har en mulighet til å legge til via-punkter. Google Maps og 1881 tilbyr i tillegg en funksjon slik at man grafisk kan endre kjøreruta, kun ved å flytte kjøreruta og legge til viapunkter i kartet. Bompengekostnader er det færre som viser. Kvasir og Gule Sider benytter seg av samme tjeneste, og inkluderer ingen informasjon om bomstasjoner langs ruta. NAF og Google oppgir at det er bomvei på deler av strekningen, uten å spesifisere nøyaktig

	Kvasir	Gule Sider	Naf	Ruteplantjenesten	1881.no	Google Maps
Avstand	✓	✓	✓	✓	✓	✓
Kjørebeskrivelse	✓	✓	✓	✓	✓	✓
Bompengekostnader totalt på reisen	✗	✗	✗	✓	✓	✗
Bomstasjoner på strekningen	✗	✗	✓	✓	✓	✓
Bomstasjoner med pris	✗	✗	✗	✓	✓	✗
Spesialpriser, rushtid osv	✗	✗	✗	✗	✓	✗
Autofullført søk	✓	✓	✓	✗	✓	✓
Kart	✓	✓	✓	✓	✓	✓

Figur 2.2: Eksisterende ruteplanleggingstjenester

hvor på strekningen bomstasjonene befinner seg og ei heller prisen. 1881 oppgir prisen for vanlig personbil og lastebil, i tillegg navn på bomstasjon og hvor den ligger (grafisk på kart og med koordinater). Siden bakgrunnen for applikasjonen vi skal utvikle, er å beregne både avstand og bompengekostnader, er det kun Ruteplantjenesten og 1881 som tilbyr informasjonen vi trenger.

### 1881 eller Ruteplantjenesten?

For å finne ut om 1881 eller Ruteplantjenesten egner seg best, tar vi en nærmere sjekk på hvilken informasjon hver av tjenestene inneholder. Vi vet som beskrevet i kapittel 2.1.2, at det finnes både rabattordninger og tilleggspriser rundt omkring i Norge. For å finne ut om Ruteplantjenesten eller 1881 inneholder informasjon om disse spesialprisene, har vi testet ruteberegning gjennom de bomstasjonene vi på forhånd vet har slike ordninger. Mengderabatter med tidsbegrensing, slik som i Trondheim og Haugesund er ikke tatt hensyn til på hverken 1881 eller ruteplantjenesten, men 1881 viser informasjon om rushtidsprisene i Trondheim. Ved beregning av kostnader for reiser gjennom bomanlegg med rushtidstillegg, benyttes vanlig takst uten tillegg. Det spesifiseres så i veibeskrivelsen at det mellom 07:00 og 09:00, samt 15:00 og 17:00 er dobbel takst.

1881 tilbyr en bedre kartløsning som er mer brukervennlig. Kartløsningen hos ruteplantjenesten er uoversiktlig og lite brukervennlig, og kan, sammenlignet med tjenester som 1881 og Google Maps, virke gammeldags. I ruteplantjenesten må adresser angis korrekt. En tastefeil midt i adressen vil ikke gi resultater. Ved å utelate de siste bokstavene i et stedsnavn, vil den i noen tilfeller gjette riktig, men dersom flere steder begynner med samme navn, vil den velge tilfeldig. Et søk på Storgatablir Storgata, Moss". Hvilken Storgata" man får opp er helt tilfeldig, og er hverken basert på avstand fra hvor vi befinner oss (da skulle Halden ha blitt valgt), og det er heller ikke alfabetisk. Brukeren har ingen mulighet til å påvirke dette, utenom å rette opp i ettertid når han eller hun ser at byen er feil. 1881 tilbyr autofullførfunksjon, så et søk på Storgatagir ut en liste over de forskjellige Storgataer som finnes. Man vil da i tillegg raskt oppdage en eventuell skrivefeil i adressen, siden adressen man ønsker å søke etter, ikke lenger vil dukke opp blant forslagene dersom man har en

skrivefeil i adressen.

1881 sitt system for veibeskrivelser innehar alle nødvendige data vi trenger. Vi konkluderer derfor innad i gruppa med at 1881 egner seg best, siden 1881 tilbyr litt mer detaljinformasjon om noen av bomstasjonene, søkefunksjonen er mer brukervennlig med tanke på autofullføringen, kartløsningen er bedre, og det er mulighet for å grafisk kunne endre kjøreruta, dersom man ser at ruta som beregnes ikke er nøyaktig den man har kjørt. Det eneste som skiller 1881 negativt fra ruteplantjenesten er at kjøreruter kan bli beregnet gjennom Sverige dersom dette er korteste vei. Dette er uhensiktsmessig da dette vil komplisere beregning av priser, da det etter Statens Reiseregulativ er andre takster for utenlandskjøring. Det er mulig det finnes en mulighet å tvinge kjørerutene til å holde seg innenfor Norges grenser. 1881 opplyser at de har et API, som etter en gratis testperiode, ville koste penger. Petter er enig med oss i at 1881 høres ut som en god løsning, selv om det på sikt vil koste penger. Dersom tjenesten er stabil og gir all relevant informasjon vi trenger, vil ikke kostnadene være et hinder for fremtidig bruk. Ved nærmere undersøkelse av API-et viser det seg derimot at dette kun er et API for person- og bedriftssøk, ikke for veibeskrivelser. Det eneste geografiske innhold i API-et er privatpersoner og bedrifters adresser. Dette til tross for at vi forklarte hva vi skulle bruke API-et til, da vi tok kontakt med 1881 for å starte en testperiode. 1881 vil derfor ikke dekke de kravene til informasjon prosjektet vårt har. Ved nærmere undersøkelser av Ruteplantjenesten, viser det seg at JSON-objektet som returneres, har tilleggsinformasjon om navn og priser på alle bomstasjonene på ruta, selv om dette ikke vises i webtjenesten de tilbyr. Hvis det er fem bomstasjoner på den angitte ruten vil JSON-filen som returneres inneholde fem bomstasjonobjekter, der hvert enkelt objekt inneholder navn, takst litenbil, takst storbil. Disse objektene ligger under `nvdb:Bomstasjon` i JSON-fila.

### Andre muligheter

Ruteplantjenesten er, som vist i *figur.2.2*, eneste reiseplanleggingsverktøy vi har testet, uten autofullført søk. Da autofullføring bidrar til å gjøre de andre resterende ruteberegningstjenestene mer brukervennlig enn Ruteplantjenesten i med tanke på adressesøk, er dette en tilleggsfunksjon vi gjerne ønsker å inkludere i vår applikasjon. Vi er da nødt til å finne en autofullføringsfunksjon for adressesøk som er kompatibel med vår planlagte applikasjon som baserer seg på Ruteplantjenesten. Google sin funksjon for å autofullføre søk både på google og google maps, er en løsning vi alle er kjent med og ser på som god. Med Google Autocomplete får vi også tilgang til koordinatene til adressen, og dette er positivt da beregningene i Ruteplantjenesten er basert på koordinater. Google og Ruteplantjenesten benytter ulike koordinatsystemer som gjør at disse ikke kan benyttes samtidig uten å konvertere koordinatene først. Mer om koordinatsystemene og utfordringene forbundet med dette, finnes i kapittel 4.1. I hovedsak vil resten av applikasjonen kun basere seg på APIet til ruteplantjenesten. Det ville selvsagt vært enklere og mer oversiktlig å kun benytte seg av et api. Dessverre finnes det ingen system tilgjengelig som både kan tilby autofullført søk, gi god informasjon om avstander, detaljinformasjon bomstasjoner (med priser) og kjøreretning.

Vi har også et ønske om å kunne vise ruten på kart, slik at brukeren kan kontrollere at den beregnede ruta stemmer. Vi forsøker å sammenligne kjøreruter som beregnes hos Ruteplantjenesten med tilvarende kjørerute på Google Maps, for å se om vi kan benytte

Google Maps direkte. På korte strekninger ser kjørerutene tilsynelatende like ut, men et problem som dukker opp er at på lengre kjøreruter, og spesielt kjøreruter der korteste vei går via Sverige, vil Ruteplantjenesten og Google vise forskjellige ruter. Google beregner korteste vei uavhengig av landegrenser, mens Ruteplantjenesten beregner korteste vei i Norge. For eksempel kan en bruker ha kjørt fra Oslo til Kirkenes gjennom Norge, mens Google vil vise korteste distanse, altså gjennom Sverige og Finland. Google tilbyr ingen mulighet for restriksjoner i forhold til landegrenser. På den andre siden støtter ikke Ruteplantjenesten kjøreretninger utenom Norge. Google opplyser også at de ikke støtter andre karttjenester kombinert med deres autocomplete-funksjon. Det store spørsmålet er om vi virkelig har behov for en karttjeneste i vår applikasjon, siden vi ikke skal levere et system for veibeskrivelser, men et system som skal behandle reiser etter at de har funnet sted. Kartet er ment som et hjelpemiddel for å kontrollere at beregnet reiserute stemmer med ruten som er kjørt. Kan denne informasjonen vises bedre på andre måter? Istedenfor å presentere ruta på kart, kan vi heller vise den i tekstformat. For eksempel: Bruker A kjørte fra Oslo til Trondheim, E6 til Hamar, om Østerdalen, og videre på E6 frem til destinasjonen. Vi vil argumentere for at denne informasjonen best vises punktvis i tekstformat fremfor som en linje på et kart. Denne tekstlige beskrivelsen skal kunne hentes ut fra veibeskrivelsen fra Ruteplantjenesten. Denne tekstlige beskrivelsen vil erstatte det nåværende feltet der kjøreruta beskrives detaljert, slik som beskrevet i kapittel 2.1.1.

## 2.1.4 Universiell utforming

### **Dette kapittelet skal skrives om og også illustreres i form av figurer**

1.juli 2014 ble nye regler for universell utforming av IKT-løsninger, «Retningslinjer for tilgjengelig webinnhald (WCAG) 2.0» *ref.21* Regelen innebærer at alle nye IKT-løsninger i Norge, både innen privat og offentlig sektor, organisasjoner og lag, skal være universelt utformet *ref.22*. Kort sagt er dette en lovpålagt standard for utforming av IKT-løsninger. Eksisterende IKT-løsninger har frist til 21.januar 2021 på å rette seg etter kravene. Kravet er at nettløsninger må oppfylle 35 av de 61 kriteriene definert i WCAG 2.0 for å være godkjent. Kriteriene er delt inn i tre nivåer A, AA og AAA, der alle kravene i nivå A og AA (unntatt 1.2.3, 1.2.4 og 1.2.5) må være oppfylt *ref.23*.

Kravene innebærer at IKT-løsninger skal være tilgjengelige for alle, og det tas hensyn med tanke på brukervennlighet slik at alle skal ha forutsetninger for å kunne bruke løsningen, også de som har nedsatt oppfattelsesevne i form av dårlig syn og hørsel, fargeblindhet osv. Brukere av vår applikasjon skal registrere reiseregninger på reiser de selv har kjørt, og det at de kan kjøre bil forsetter at de oppfyller en rekke krav med tanke på syn osv. Kravene gjelder alle nettsider, uavhengig av innhold, men siden dette ikke er en side for svaksynte/blinde osv, holder det å oppfylle minimumskravene. Kriteriene er delt inn etter 4 prinsipper.

### **Prinsipp 1**

mulig å oppfatte, setter krav til bruken av kontraster, fargevalg, forstørrelsesmuligheter og bruk av tekst som alternativ til bilder, lyd og video.



**Prinsipp 2**

Mulig å betjene, setter krav til brukervennlighet i form av at brukeren fullt ut skal kunne bruke løsningen uavhengig av teknisk utstyr, blant annet slik at en løsning tenkt brukt med mus og tastatur, også skal være mulig å betjene kun via tastatur.

**Prinsipp 3**

Forståelig, innebærer at løsningene skal være forutsigbare, det skal være lett å forstå hvordan de skal brukes og informasjonen man finner skal være enkel å forstå. Krav til bruk av både enkelt språk og hjelpetekster bidrar til dette.

**Prinsipp 4**

Robusthet, er rettet mer mot det tekniske. Her settes det noen retningslinjer i form av koding og tilgjengelighet. Innholdet må kunne tolkes på en god måte av forskjellig teknologi, nettsider må valideres, og koden må være riktig. I HTML, som vil bruke, blir som regel dette ivarettatt, men det er spesielt viktig å sikre god tilgjengelighet dersom man utvikler egne elementer (widgets).

Vi har studert retningslinjene, for å vite hvilke hensyn vi må ta under utformingen av applikasjonen. Vi har utelukket kriteriene som går på lyd og video, da disse ikke vil være aktuelle for vår del. Det er i tillegg en del av kriteriene som ikke vil treffe vår applikasjon direkte da denne kun utgjør en liten del av HRESSURS. Vi tar utgangspunkt i at HRESSURS allerede oppfyller disse kravene, eventuelt at disse oppfylles når HRESSURS relanseres i ny drakt. Vi vil rette oss inn etter eksisterende HRESSURS med tanke på fargevalg, tekstfarger osv.

Ettersom dette er en løsning der brukeren skal angi informasjon, er alle underpunktene «Retningslinje 3.3 – Inndatahjelp» spesielt viktige. Denne innebærer blandt annet identifikasjon av feil, forslag ved feil og ledetekster som angir hva som skal testes inn. Vi må her sørge for at dersom brukeren taster inn for eksempel en ugyldig adresse, må det bli gitt tydelig tilbakemelding om dette. For å gjøre inntastingen av adressene enklere, benytter vi Google Autocomplete. Kravet til ledetekster innebærer at alle inndatafelt skal ha ledetekst eller instruksjoner som forteller hvilke data som skal angis hvor. Vi vil her tydeliggjøre hvor start, stopp og via punkter skal angis, at kjøretøy må velges osv. I forbindelse med via punktene vil vi også nevne viktigheten av meningsfylt rekkefølge, som spesifiseres i retningslinje 1.3. Rekkefølgen via punktene angis i, har betydning for hvilken kjørerute som beregnes. Dette skal vi gjøre tydelig for bruker ved å presisere viktigheten av rekkefølgen ved hjelp av instruksjonstekster og ved at brukerne skal ha mulighet til å sortere listen med via punkter. Denne sorteringsmuligheten vil vi også tekstlig spesifisere. Det spesifiseres i tillegg i retningslinje 2.1 at hele applikasjonen skal være tilgjengelig fra tastatur. Dette vil for eksempel bety at det skal være mulig å navigere mellom tekstfelt med tabulator-tasten. Dette vil vi sørge for at fungerer.

3.3.4 Forhindring av feil (juridiske feil, økonomiske feil, datafeil) er også et kapittel som er veldig viktig i forhold til vår applikasjon. Dette prinsippet går i all hovedsak ut på at dersom brukeren taster inn data som endrer brukerdata, slik som informasjon på brukerens reiseregning, skal det være en kontrollmulighet for å hindre at uriktige data overskriver eksisterende, riktige data. En av følgende tre mulige løsninger kan håndtere dette. Enten

ved at prosessen med innsending av data kan reverseres, at brukeren manuelt har mulighet til å gå tilbake å rette opp i feil eller at brukeren må bekrefte innsending av data. Selv om kravet bare er at minst et av disse er oppfylt, er det i vår planlagte applikasjon, fornuftig både at brukeren kan gå tilbake å endre på destinasjoner på ruta dersom beregningen blir feil, før ruta beregnes på nytt, og at det skal være en kontrollfunksjon der brukeren må bekrefte at han eller hun faktisk ønsker å legge de beregnede verdiene til i sin reiseregning.

Vi velger å se bort i fra følgende retningslinjer:

1.1 Teskstalternativer

1.2 – tidsbaserte medier

2.3 – Anfall

Dette er alle kapitler som er mer aktuelle i en mulitmedieapplikasjon, eller en applikasjon der grafikken spiller en stor del av opplevelsen.

Følgende retningslinjer er mer relevant for hele HRESSURS, og ikke bare vår tilleggsapplikasjon:

1.4 – Mulig å skille fra hverandre

2.2 – nok tid, 2.4 – navigerbar

3.2 – Forutsigbar,

Disse går på blant annet ut på navigering på websidene, fargevalg og kontraster, at brukerne skal varsles på forhånd ved automatisk utlogging osv. Alt dette er retningslinjer som bør være oppfylt overordnet i HRESSURS. Vi vil tilpasse farger og skriftstørrelser etter slik HRESSURS er i dag. Om kravene til universell utforming er oppfylt eller ikke, vil vi ikke studere nærmere. Nåværende HRESSURS er ikke en ny webløsning etter 1.juli 2014, noe som innebærer at den bare omfattes av regelverket for eksisterende løsninger, slik at den først må følge reglene innen 2021. HRESSURS skal, som vi vil gå nærmere inn på i kapittel 2.2.2, relanseres i ny drakt. Når den relanseres trer regelverket i kraft, og da vil HRESSURS omfattes av reglene. Vi vil legge alt til rette for at det ikke skal bli problemer med vår applikasjon i forhold til kravene om universell utforming, og vi vil ha retningslinjene i bakhodet mens vi utformer applikasjonen.

## **2.2 Teknologi**

### **2.2.1 .NET**

Vi har valgt å utvikle vår applikasjon innenfor Microsoft sitt .Net rammeverk. Vi har undersøkt mulighetene for å benytte oss av andre teknologier, språk og rammeverk som kanskje ville ha ført til at vi satt igjen med en mer åpen og frittstående applikasjon, men denne ville ikke kunne implementeres direkte inn i HRESSURS uten tilpassninger. Vårt ønske er at oppdragsgiver skal kunne fase inn vår applikasjon inn i personalsystemet uten for mange tilpasninger. Infotjenester benytter i dag .Net i hele sin virksomhet. Personalsystemet HRESSURS er som nevnt i kapittel 2.1.1 delt inn i flere kategorier, men felles for hele systemet er teknologien som ligger bak. HRESSURS som webapplikasjon er laget i ASP

.Net. Siden vårt system på sikt skal erstatte deler av dagens ordning for registrering av reise og utlegg er det naturlig at vi velger samme plattform som HRessurs allerede er basert på. Til hvilken grad applikasjonen vil erstatte eksisterende løsning, eller komme i tillegg til eksisterende løsning, avhenger av hvordan vår endelige applikasjoner utformet.

Innenfor .Net rammeverket er det ulike måter å strukturere en webapplikasjon slik som den vi skal lage. Innenfor .Net finnes det flere ulike typer klassebibliotek som er spesialisert mot ulike typer programvare. Den vanligste for web, er ASP.Net biblioteket. Innenfor ASP.Net finnes det igjen ulike plattformer for webutvikling, slik som MVC, web-api osv. Microsoft har planer om å knytte alle disse sammen i et nytt rammeverk som kalles ASP.Net MVC6.

### 2.2.2 ASP.NET vs ren HTML

Vi har fra uke 7 begynt å flytte noe av logikken over på server. Det er for at systemet skal være mer vedlikeholdbart. For eksempel i forhold til gjenbruk slik at man slipper å lage samme komponent flere ganger. Men også for å unngå «cross-domain», altså at brukers nettleser som ligger på vår server oppretter kontakt med en annen server for å innhente informasjon. I stedet skal vår server innhente informasjonen (feks fra Ruteplantjenesten), og så vise dataene til brukeren i nettleseren. Brukeren skal kun være på vår server hele tiden. Andre fordeler er at man løser «interoperability problems», altså at vi muliggjør at forskjellige systemer på forskjellige plattformer kan kommunisere med hverandre. En mulighet vi nå utforsker for å løse dette er å sette opp en web-service som oppretter kontakt med vegvesenets server. Det vil si at nettleseren sender en ajax-request til vår web-service som så sender en forespørsel til Ruteplantjenesten. Når denne svarer får web-servicen et svar tilbake (i vårt tilfelle JSON), som så sender denne tilbake til frontend Javascript som viser dataene til brukeren.

Vi har både sett på hvordan man kan gjøre dette i Microsoft sitt ASP .Net rammeverk som gjør jobben med å snakke med en web-service mindre krevende, men også hvordan dette kan la seg gjøre fra en ren htmlside. Microsoft sitt ASP .Net er ment å gjøre jobben med å lage en dynamisk webside enklere. Det vil si at det å vedlikeholde websiden (innholdet) skal være mer overkommelig. For eksempel hvis man vil inkludere en ny side (web forms/aspx) eller redigere innholdet i en meny. Men samtidig vil arbeidet (i teorien) med å «snakke» med en web-service vært enklere via web forms enn i ren html. Microsoft introduserte muligheten for at backendkoden ligger i aspx.cs-filer (.cs hvis man bruker C#), mens statisk kode ligger i aspx-filene. Det vil si at all kode som skal behandle funksjoner basert på brukervalg på siden behandles av filene med aspx.cs-endelse. Alle behandlede og kompilerte filer i ASP .Net lagres i ulike kataloger som er tilgjengelig for alle sider på nettstedet. Det inkluderer også web-services. Alle filer (inkludert WSDL-filer som bestemmer hvordan kommunikasjonen skal foregå) som refererer til en web service ligger i en egen katalog slik at denne kan nå enkelt i koden fra alle sider, altså web forms som Microsoft kaller det.

Men Infotjenester er i ferd med å lansere HRessurs i ny drakt. Det innebærer ikke bare

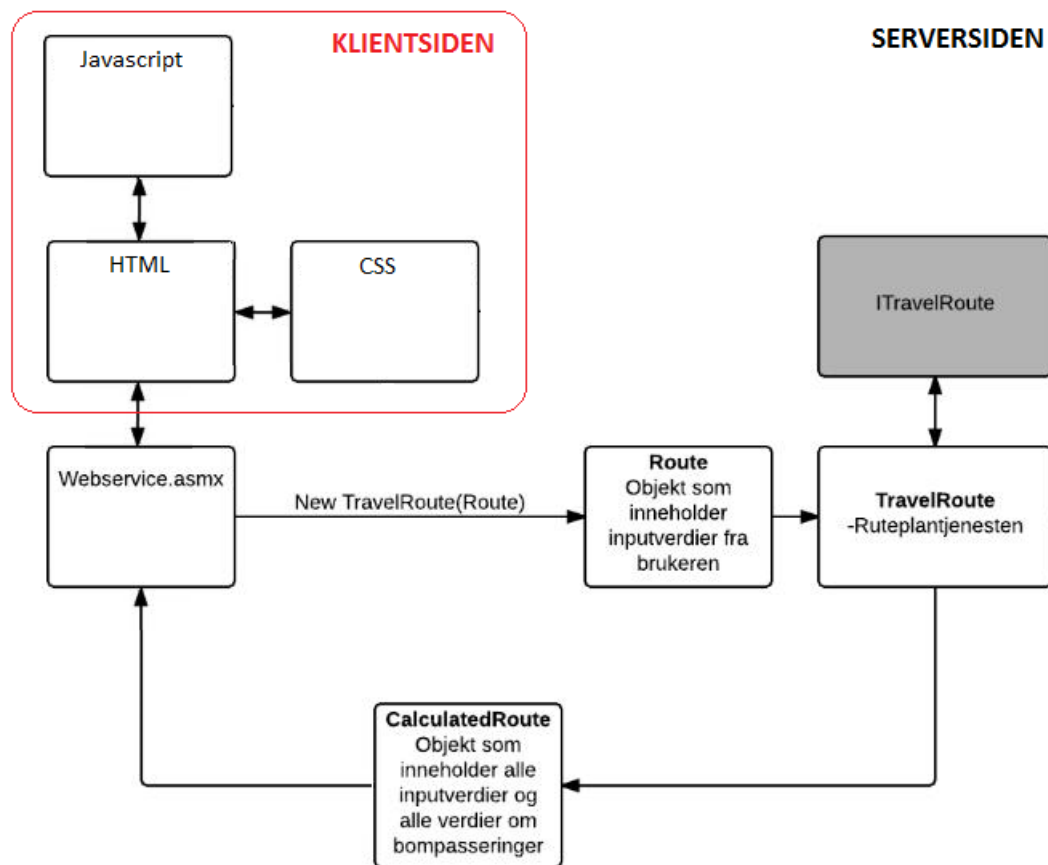
rent utseendemessig, men også bak sløret. Tilbakemeldinger fra flere av utviklerne ved bedriften er at aspx-systemet som ligger bak dagens HRessurs begynner å bli uoversiktlig og vanskelig å vedlikeholde. Mest sannsynlig vil neste generasjon HRessurs bestå av rene HTML-filer frontend, mens selve funksjonaliteten fortsatt befinne seg i .Net baserte web-services. Vårt dilemma blir da om vi skal fortsette å utvikle vår reiseapp i det dynamiske aspx-biblioteket som teknisk sett skal gjøre jobben med denne typen webapp mer overkommelig, samt at dagens ordning er laget i aspx, eller om vi skal utvikle for fremtidens HRessurs og dermed tilpasse oss Infotjenesters behov i større grad. Vi ønsker å utvikle en applikasjon som faktisk vil bli tatt i bruk, og som vil være i bruk en god stund framover. Dette er hovedgrunnen for at vi velger å rette oss inn mot fremtidens HRessurs ved å bruke ren HTML og asmx-webservice istedenfor aspx.

## 3. Planlegging - Struktur og design

### 3.1 Kodemessig struktur

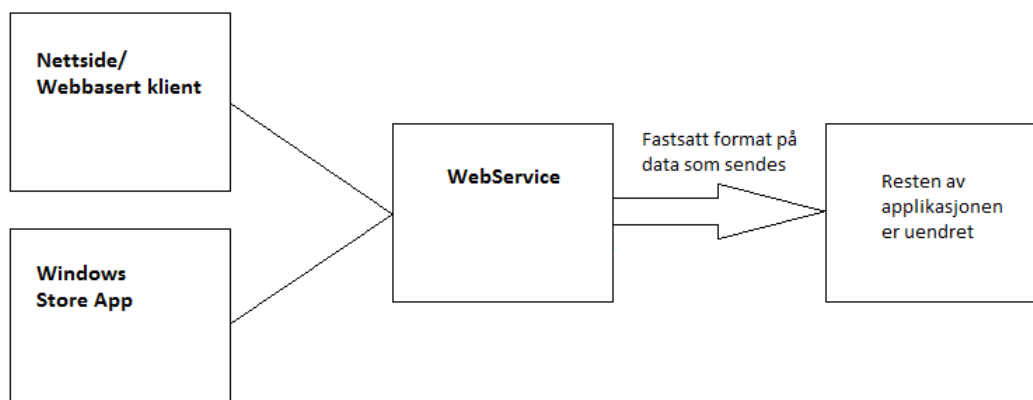
Til vår applikasjon er det en rekke ønsker/krav fra oppdragsgiver. Det er et ønske om at applikasjonen skal være bygd etter en lignende struktur som HRessurs har i dag, med klientside der brukergrensesnittet ligger, og en serverside der alle beregninger foregår. Vi står nokså fritt til å bestemme hvordan vi ønsker å gjøre dette, men for at vi ikke ender opp med å utvikle noe som er helt forskjellig fra HRessurs, tar vi imot forslag og veiledning fra Petter før vi går i gang med denne delen av prosjektet. Vi får ett krav, og det er at alle beregninger som kan foregå på en server, skal foregå på Infotjenesters server. På serveren skal alle beregninger foregå i C#, mens på klientsiden (brukergrensesnittet) skal brukeren angi all informasjon som er relevant for beregningene. Klientsiden vil inneholde en HTML-side, css-fil der vi tilpasser utseendet på HTML-siden, i tillegg til flere javascriptfiler. I javascript verifiseres data inntastet av brukes, disse gjøres om disse til riktig format før det til slutt sendes til serversiden. Når responsen fra serveren kommer, vil vi også behandle responsdata i javascript, slik at vi kan presentere de beregnede dataene for brukeren. Da det vil bli en del forskjellige beregninger i javascript, ser vi det som hensiktsmessig å fordele javascriptkoden i ulike filer med tanke på om koden utfører verifisering av inntastet data eller behandler responsen fra server osv. På serveren vil vi ha en asmx-webservice som håndterer kommunikasjonen mellom klient og server. Her vil inputdata fra klientsiden, formateres om til et objekt av klassen Route, slik at vi har kontroll på at all data som behandles på serveren vil være på riktig format før beregningene startes.

Noe av hovedtanken bak en slik lagdeling av applikasjonen er at alle deler skal være utskiftbare. Dette innebærer at alle delene prosjektet består av, skal kunne byttes ut og erstattes av en tilsvarende del, uten at dette innebærer endring i de resterende delene a prosjektet. For at dette skal være mulig, må det ligge en grunnleggende struktur i bunn. Dette vil i praksis si at det skal være mulig å bytte ut ruteplantjenesten med en annen veiberegningsleverandør. For eksempel kan det hende at 1881 begynner å tilby et API for veiberegningstjenester, mens ruteplantjenesten kutter ut sin tjeneste. Dette byttet av tjenestetilbyder kan da gjennomføres uten at andre deler av koden må endres. Ved å fastsette hvilke parametere som mottas fra klientsiden, og hvilke parametere som returneres tilbake til klientsiden, spiller det ingen rolle om beregningene foregår via Ruteplantjenesten eller 1881. Web Service kontrollerer at inputdata fra klientsiden er på riktig format, og ved at klassen TravelRoute (der beregninger foregår) implementerer interfacet ITravelRoute, vil beregningsklassen alltid ha samme struktur uavhengig av tjenesteleverandør som benyttes til beregningene. Det samme gjelder også dersom man ønsker å lage en annen type applikasjon enn en nettside for å utføre beregningene. Hvis det er fastsatt at serveren skal ta i mot kjøretøy og stedsnavn og koordinater på formen UTM33, kan websiden byttes ut med en Windows Store App som sender inn de samme parameterne. Det er ikke nødvendig å gjøre andre endringer på serversiden, annet enn evt småendringer i Web Service som tar



Figur 3.1: Vedlikeholdbar struktur

i mot parameterne, dersom parametere fra en webapplikasjon og Store App formateres ulikt. Figur xxxx illustrerer dette.



Figur 3.2: Skal utbedres. Skisse som illustrerer utskiftbarhet

## 3.2 Brukergrensesnitt

Etter første møte med arbeidsgiver hvor vi diskuterte hva som bør være med i vår applikasjon bestemte vi oss for å lage en tidlig skisse på funksjonalitet og brukergrensesnitt se *figur 3.3* Vi gjorde dette for å få i gang tankegangen på hvordan vi kunne gjøre ting, samt hvilke elementer vi følte var nødvendig å ha med applikasjonen før vi begynte på selve utviklingsprosessen.



Figur 3.3: Tidlig eksempel på GUI

I en slik applikasjon som designes med hovedfokus på at den skal være brukervennlig og lett forståelig, vet vi at vi må ha minst mulig knapper, og at funksjonaliteten bør være selvforklarende. Planen er å få frem all funksjonalitet med så få steg som mulig. Det er nødvendig å kunne angi start, stopp og viapunkter og kjøretøy, og å ha en knapp for å beregne kjøreruta. Ved behov for eventuelle andre elementer utover dette, må vi veie opp funksjonaliteten disse elementene tilføyer, samt hvor viktige de er for applikasjonen. Vi må også forholde oss til hvordan HRessurs er bygd opp i dag, slik at det blir en helhetlig applikasjon når applikasjonen vår blir integrert mot slutten av bachelorperioden. Vi har en testbruker i HRessurs, slik at vi hele tiden har mulighet til å logge oss inn og holde oss oppdatert på hvordan HRessurs er bygd opp, og eventuelle endringer det måtte komme iløpet av bachelorperioden.

Figur 3.4 er også en av våre første eksempler på design. Her tar vi hensyn til HResurs, og utformer skissen i Paint. Disse eksemplene ble derfor tatt i bruk for å komme i gang og for å få frem ideer på hvordan ting egentlig bør være. Vi velger også å ha hovedfokus på funksjonalitet igjennom hele prosjektet, designet blir ikke prioritert før slutfasen.

The screenshot displays a web application interface for travel planning, specifically the 'Kjøring' (Driving) tab. The interface is organized into two main columns. The left column contains input fields for 'Rute' (Route), 'Total kilometer', 'Fremkomstmiddel' (Vehicle type, currently set to 'Privat bil'), 'Kilometer utland', and 'Årsak til eventuell omkjøring'. Below these are expandable sections for 'Passasjerer' and 'Spesielle tillegg'. At the bottom of this column are two buttons: 'Legg til' and 'Kalkulator'. The right column features a 'Fremkomstmiddel' section with radio button icons for different vehicle types. Below this are input fields for 'Start', 'Stopp', and 'Via', followed by a 'Beregn' button. A 'Beregnet rute' section displays the result of the calculation: 'Resultat av beregning: start stopp og via, avstand, bomstasjoner m/ priser Veibeskrivelse'. At the bottom of this column is a 'Legg til i reiseregning' button. The top of the interface has a navigation bar with tabs: 'Start', 'Overnatting', 'Måltidsfradrag', 'Kjøring' (active), 'Utlegg', 'Tillegg', and 'Fullfør'.

Figur 3.4: Tidlig eksempel på GUI



## 4. Utvikling av applikasjonen - steg for steg

### 4.1 Google og konvertering

Google sin Autocomplete er lagt til i tekstfeltene der adresser skal angis, og fungerer slik at brukeren får opp forslag til start og stopp og via-lokasjoner når man fyller inn tekstfeltene. Deretter vil Google hente koordinatene til disse stedene. Google benytter koordinater etter den amerikanske standarden WGS84, og Ruteplantjenesten benytter den europeiske standarden ETRS89. For å kunne bruke informasjonen i Ruteplantjenesten, må koordinatene fra Google konverteres slik at vi kan bruke stedene Google finner til ruteberegning fra Rutenplantjenesten.

For å oppfylle alle ønsker og krav til prosjektet har vi har vært avhengig av å bruke Google APIet til flere deler av vår webapplikasjon. Enn så lenge vil vi benytte Googles autocomplete-funksjon, for å gjøre adressesøk mer brukervennlig. For å kunne benytte Google Autocomplete API er man avhengig av å laste «Google Places Library» før funksjonene som skal vise stedsforslag kjører. Dette gjøres ved å legge en link øverst på siden som henviser til adressen der Google API ligger, samtidig som man legger ved et «libraries» parameter på slutten av linken. Men i hovedsak vil applikasjonen basere seg på APIet til ruteplantjenesten. Det er på vegvesenets servere at det meste av informasjonen vi trenger ligger lagret.

```
<script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?libraries=places"></script>
```

Figur 4.1: Viser eksempel på hvordan man henter Google API Autocomplete basert på steder.

Konverteren fungerer slik at den får tilsendt koordinatene fra Google som ligger lagret i globale variabler, konverter dem til europeisk standard, og sender de nye koordinatene til Ruteplantjenesten i en JSON-string. Funksjonene som utfører konverteringen består av komplekse matematiske utregninger som gjøres i flere steg. Siden koordinatene i WGS84 kontra ETRS89 er så forskjellige fra hverandre trengs det flere formler som gjør en liten bit konvertering hver for seg. Man må blant annet ta hensyn til ekvator radius, polar utflating og polar akse. Deretter må breddegrader og lengdegrader fra WGS84 omregnes til radianer som kan brukes av ETRS89-systemet, og deretter regnes om til x- og y-koordinater i meter. I tillegg til dette regnes forskjellige land inn i ulike soner. Norsk sone regnes som nummer 33, selv om Norge egentlig strekker seg fra sone 32 til 36. Men avvikene er såpass små at det holder med å definere sone 33 som standard for hele landet.

## 4.2 Alt i ett løsning

Vår første versjon baserer seg kun på HTML og Javascript. Den bruker også 3 forskjellige API systemer. For den geografiske delen samt koordinater bruker vi Google Map API og Google Autocomplete API, og for all nyttig kjøredata håndteres det av Ruteplantjenesten som er eid av Statens Vegvesen. Vi er i kontakt med Vegvesenet for å få opprettet fri tilgang til det lukkede API'et gjennom hele bachelorperioden. Om løsningen er varig i etterkant med tanke på om Vegvesenet lar private bedrifter bruke løsningen deres er ikke undersøkt, men vi jobber med saken.

Når man taster in start og stopp addressene, bruker vi Google sitt API til å autofullføre de forskjellige søkene man sender inn. Når autofullføring av søk er ferdig og man trykker beregn vil en funksjon kjøre som konverterer adressesøket om til koordinater. Deretter konverteres koordinatene om til et format som Ruteplantjenesten forstår. I det koordinatene er ferdig konvertert blir de bygd inn i en URL-string som vi sender til Ruteplantjenesten sitt REST API, hvor vi får returnert et JSON objekt med forskjellig informasjon som for eksempel avstander, kjørebekrivelse og bomavgifter.

I JSON objektet henter vi så ut total kjørelengde, beregnet tid samt navn på bomstasjoner og takst for passeringer. Dette blir så igjen returnert til HTML-siden som skiserer ruten grafisk på et Google kart og returnerer de spesifikke dataene.

## 4.3 HTML - Web Service Versjon 1

I vår løsning, skal brukergrensesnittet være knyttet mot en web service, som senere skal ligge på Infotjenesters server for å ha så lite logikk som mulig på klientsiden. Vi skal ha klientsiden, med html, javascript og css, og web servicen med c#. Denne første løsningen baserer seg i utgangspunkt på testdata. Vi skal sende inn start, stopp, via og kjøretøy til web service, slik den ferdige løsningen også skal kunne. Web servicen skal så returnere dummydata for bomstasjoner, priser og avstand. Web service skal senere kobles mot ruteplantjenesten, slik at dummydata 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 løsningen vi tenker. Vi prøver mange eksempler, både ved å klippe ut og lime inn kode, og ved å laste ned hele løsninger som skal fungere. Ingen av løsningene klarer derimot å opprette forbindelse til web servicen, 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 en stund før de tvinges til å oppgradere, finnes det også en del nyere

eksempler der useService brukes. Det skal også 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 t.o.m 2013. Publiseringsdato, og å teste i alle nettlesere, er etter dette noe vi vil ha fokus på å sjekke opp i, dersom vi møter tilsvarende problemstillinger underveis.

Da vi ikke skal bruke aspx, som beskrevet i kap xxxxx.. 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 kun et parameter, slik at vi ikke må ha ulike metoder for hvert antall viapunkter. I webservicen splitter vi denne tekststrengen på komma, slik at vi får viapunktene fordelt i et array, slik at viapunktene kan behandles hver for seg under kalkuleringen av ruta. Denne løsningen for viapunkter er ikke ideell, og vil bli byttet ut senere.

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

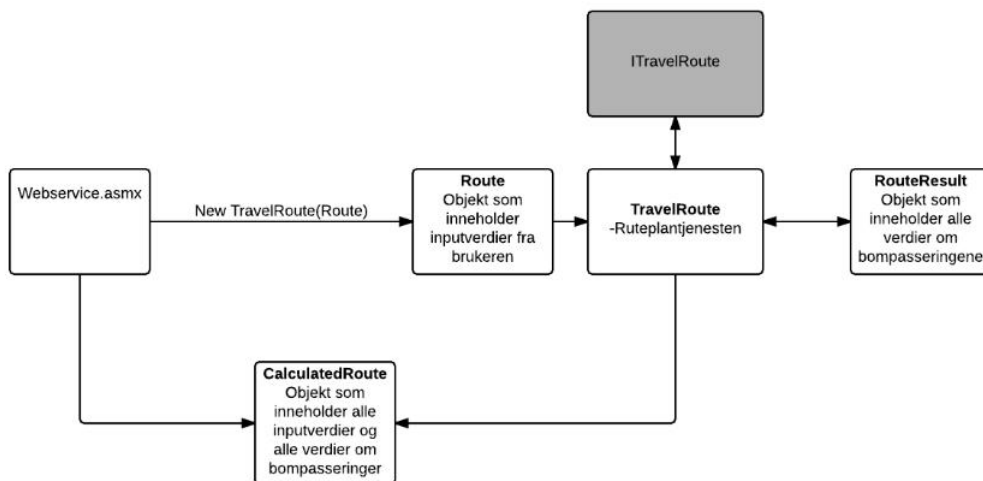
Figur 4.2: Strukturen på CalculatedRoute

Vi har en klasse, CalculatedRoute, der et objekt av klassen, inneholder all informasjon om en kjørerute. Dette objektet inneholder informasjon om start, stopp, valgt kjøretøy, viapunkter, totalpris for liten og stor bil og bomstasjoner, derav navn på alle bomstasjonene, samt priser for stor og liten bil. Strukturen kan sees i *Figur.4.2*. Når brukeren trykker knappen «Send rute», sendes verdiparet med intastet data til webservice. I Web service initialiseres et nytt objekt av CalculatedRoute basert på inputverdiene og resultatet av den beregnede ruta. Dette sendes så tilbake til klientsiden, der vi parser (pakker ut) objektet, og fremstiller relevant informasjon for brukeren. Denne første løsning inneholder bare testdata, slik at bomstasjonene får navn fra Bom0 til Bom1, og prisene for hver av bomstasjonene generes tilfeldig med randomfunksjon. Prisen for litenbil, 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.

Da web service ikke returnerer flerdimensjonale array, har vi valgt å returnere JSON. JSON har den fordelen at selv om selve elementet er fledimensjonalt, kan objektet returneres som en endimensjonal tekststreng, som vi kan parse i javascript for å hente ut objektet. Web servicen 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 jsonobjektet.

## 4.4 HTML - Web Service Versjon 2

Vi har et sprintmøte med Petter, får bekreftet at det skal la seg gjøre å endre slik at web service returnerer 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 jsonobjekt 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 i midten av januar, se *Figur. 4.3*



Figur 4.3: Strukturen fra Web Service til Ruteplantjenesten

Vi begynner først med å implementere strukturen i vår eksisterende løsning som er beskrevet i *kap 4.1.3, HTML – Webservice versjon1*. All kode flytter vi så 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 er fokuset å utbedre applikasjonen backend, og for å kunne teste applikasjonen raskere underveis, definerer vi et ferdig objekt i web service, slik at vi bare trenger å trykke på en knapp. Da slipper vi å angi verdier hver gang vi tester, men vi får samtidig testet at ingenting går galt i forbindelsen mellom klientsiden og web service. I WebService initialiserer vi et nytt objekt av klassen TravelRoute med Route-objektet som inputparameter. All kode fra HTML – WebService versjon 1, fordeler vi i de metodene de hører hjemme. All data er foreløpig fortsatt dummydata. All kode 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 intefacet kan sees i figuren under:

```
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 4.4: 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. Første versjon for å teste inneholder URL med koordinater. Koordinatene legger vi så i det ferdigdefinerte inputobjektet, der koordinatene når converteren, beskrevet i *kap 4.1.5*, er ferdig. Denne returnerer JSON. Da en metode kun skal ha en oppgave, og search allerede gjør en forespørsel mot Ruteplantjenesten, sender vi JSON-tekststrengen videre som input til metoden CalculateResult. Her bytter vi nå ut koden som generer tilfeldige bomstasjoner og legger informasjonen inn i objektet Result, med kode som parser JSON-objektet, og legger reell informasjon om bomstasjonene inn i objektet Result. Tilslutt kjøres metoden CalcResult, som bygger objektet med all data om reisen, som returneres til klientsiden. Dette endelige objektet inneholder tekstlig start, stopp og viapunkter, samt all informasjon om bompasseringer og avstander. Det tekstlige innholdet om start, stopp og via skal senere byttes ut med en mer detaljert rutebeskrivelse som skal erstatte det nåværende tekstfeltet der ruta må spesifiseres tekstlig.

## 4.5 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 via-punkter. 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 lage ruten ved å legge til flere punkter 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 via-punkter 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 via-punkter på fem, slik at det totale antall steder som kan registreres er sju.

For å angi viapunkter begynner 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 en gang som brukeren kanskje ikke har behov for. Vi benytter først en løsning med en knapp til hvert tekstfelt. Tekstfeltene og knappene eksisterer, men er usynlige. 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. Vi har funnet ut at en bedre løsning er å bruke Javascript til å lage nye felter, slik vi går nærmere inn på i det neste avsnittet.

Vi bruker Javascript til å lage nye tekstfelt som settes inn i en ordnet liste. Denne 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. Forskjellen er at førstnevnte metode med CSS setter felter som allerede eksisterer til synlig, mens den andre metoden lager ett nytt felt for hver gang som knappen trykkes. Innenfor CSS finnes det hovedsakelig 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 skal vises. Førstnevnte gjemmer bare elementet slik at det alltid vil ta opp plass selv om det ikke er synlig for brukeren. Skjermbildene i *Figur.4.5* 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.

Figure 4.5 shows four panels illustrating the development of a route planning interface:

- Panel 1:** Initial form with input fields for 'Angi startsted' and 'Angi stopsted', a 'Send rute' button, and a 'Legg til viapunkt' button.
- Panel 2:** Added a 'Nytt viapunkt' button next to the 'Angi stopsted' field.
- Panel 3:** Added multiple 'Angi Viapunkt' input fields and a 'Beregn' button.
- Panel 4:** Final version with multiple 'Angi Viapunkt' input fields, 'Nytt viapunkt' buttons, and a 'Beregn' button.

Figur 4.5: 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 han 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. Vi referer til biblioteket sammen med en «sortable» funksjon som automatisk henter inn koden vi trenger fra JQuery-biblioteket. Dessverre kan ikke tekstfelt flyttes på, 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.4.6*) kan man derimot flytte dem rundt. Brukeren klikker på bilde som ligger ved siden av tekstfeltet og flytter på dette. Tekstfeltet følger med, siden bilde ligger i samme listeelement som det tilhørende tekstfeltet.

Figure 4.6 shows a form titled "Via:" with two input fields labeled "Angi en posisjon" and a "Beregn" button. Each input field has a hand icon next to it, indicating it is draggable.

Figur 4.6: Viapunkter til å manipulere

Noen utfordringer angående å legge til nye felter er hvordan validering skal gjøres og hvordan informasjonen skal behandles før den sendes videre til webservice som ligger back-end. Blant annet må man legge inn logikk som håndterer feilmeldinger som kommer

til å oppstå dersom ikke alle feltene som er lagt til fylles ut. Grunnen til dette er at koordinatene som hentes ut fra adressen fra Google Autocomplete, ikke er i samme format som Ruteplantjenesten krever. Se mer om dette i *kap. 4.1.5* På grunn av utfordringene i forbindelse med konverteren, og det at tekstfeltene ikke finnes fra begynnelsen av, tester vi et annet alternativ for viapunktliste, som skal fungere bedre med tanke på Google Autocomplete og konverteringen.

Selv om det tilsynelatende virker mer fornuftig å lage nye input-felter når brukeren vil legge til viapunkter, så oppstår det store problemer på klientsiden i enkelte weblesere samt i forhold til Google Autocomplete. Det virket mest hensiktsmessig å ikke ha flere felt skjult for brukeren dersom han ikke har behov for å bruke dem. Vår foretrukne løsning ble derfor å bruke Javascript til å lage nye elementer ettersom det er behov for dem, men løsningen kommer altså ikke uten utfordringer. Det første problemet som oppstår ved denne løsningen begrenser seg til nettleseren Mozilla Firefox, men er et såpass alvorlig problem at vi ikke kan overse det. Dersom en bruker lager nye input-felt 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. Dette 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 tab-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 viapunkt-feltene, må man trykke seg gjennom hele runden av diverse elementer før man kommer tilbake til det elementet man vil redigere. 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.

Metoden skaper også to problemer i forhold til Google Autocomplete. Det første gir en feilmelding tilbake til nettleseren fra Google som sier at «Autocomplete kan ikke kjøre fra et element som ikke eksisterer». Dette betyr at Google «tror» input-feltet som skal gi brukeren forslag på steder og byer ikke eksisterer. Grunnen er at funksjonen som lager nye input-felt kjører samtidig med funksjonen som henter Google Autocomplete. Selv om sistnevnte funksjon er lagt etter førstnevnte funksjon, det vil si at funksjonen som lager nye felter kaller på den andre funksjonen som skal hente Autocomplete, kjører skriptet såpass raskt at Autocomplete gjør jobben sin før input-feltet er laget. Feilen oppdages riktignok aldri av brukeren, og den oppstår alltid kun ved det første input-feltet. Dersom brukeren vil legge til flere viapunkter og dermed lager opptil flere input-felt oppstår ikke feilen på de nye feltene. Men dersom serveren fra Google ved en anledning skulle bruke lenger tid på å svare, vil kanskje ikke Autocomplete-funksjonaliteten fungere på andre felter heller, og brukeren blir da nødt til å laste inn hele siden på nytt, legge til start- og stopp verdiene om igjen, samt viapunkter. Dette er en feil vi helst vil at brukeren skal slippe å oppleve.

Den andre feilen som oppstår i forhold til Google 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 for brukeren. Han fyller




ut først kun ett felt, og velger å slette det andre han la 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 én 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 legge til, så slette og så legge til felter for viapunkter igjen. Han 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. Listen over feil har rett og slett blitt for lang til at vi kan gå for valgte fremgangsmåte for via-felter.

Den alvorligste feilen har ikke direkte å gjøre med hvordan felter for viapunkter blir tilgjengeliggjort for brukeren, men kan løses ved nettopp å forandre måten dette gjøres på. Dermed vil vi slå to fluer i en smekk. La oss si en bruker vil legge til maks antall viapunkter. Han 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 programmet kunne stoppe opp. Og denne feilen har skjedd flere ganger, rett og slett fordi for mye informasjon behandles på en gang. Dette er det alvorligste problemet vi har støtt på front end. Men ved å lagre noe informasjon underveis mens brukeren fyll inn det han har bruk for, vil vi kunne unngå denne feilen. Samtidig vil vi løse problematikken angående viapunkt-feltene.

Vi har valgt å gå litt tilbake, og kombinere CSS med Javascript igjen. Vi behøver ikke å ha mer enn ett felt for viapunkter som eksisterer når siden lastes inn, men som er usynlig for brukeren. Velger han å legge til viapunkter trykker han knappen én gang og feltet blir synlig. Siden feltet blir laget når siden lastes inn, vil ikke Google registrere feltet som ikke-eksisterende. På høyre side av input-feltet 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 konverteren og lagret i en egen sortèrbar liste (en html ul-liste) under selve input-feltet. Samtidig tømmes innholdet i input-feltet 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 programmet 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 input-felter. Som nevnt kan ikke input-felter flyttes rundt på av brukeren, men må knyttes opp mot et annet type element. I tillegg kan brukeren enkelt slette listen over viapunkter og begynne på nytt dersom han har behov for det, siden verdien(e) i den endelige beregningen ikke hentes fra input-felt, men fra en html-liste. Da er det bare å lagre nye steder i ul-listen via input-feltet.

Som beskrevet tidligere har 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 la seg løse i selve brukergrensesnittet,

**Kjøretøy:**  


**Start:**

**Stopp:**

Figur 4.7: Viapunkter med ett input-felt.

altså hvordan skal brukeren kunne sortere listen. Men en annen utfordring er hvordan programmet skal registrere en ny rekkefølge på verdiene 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 input-felter som vi knyttet sammen med et bildeelement for i det hele tatt å muliggjøre sortering, mens nå sorterer vi liste-elementer (ul-liste) som får verdier fra ett input-felt. 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. Det vil si at vi benytter oss av multidimensjonale arrays. Vi har et «hovedarray» som lagrer opptil fem andre arrays. Hvert array som lagres inneholder tre verdier: stedsnavn, x-koordinater og y-koordinater. Stedsnavnet lagres som en string (ren tekst) direkte fra input-feltet. Et array kjennetegnes ved såkalte «klamme parenteser».

```
[[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. Det er ikke nødvendig å skrive ut koordinatene til brukeren, da dette bare vil være overflødig informasjon. 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 denne nye rekkefølgen i stedet. Ved å bruke JQuery metodikk kan man telle seg gjennom hver «li» fra null og oppover, og slik samle verdiene som finnes i vår ul-liste. Hver «li» 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 de konverterte verdiene og stedsnavnene da de først ble lagret av brukeren. Denne funksjonen vil sortere elementene basert på et «key»-element, altså en nøkkel som skal gjenkjennes. Dersom denne står i en annen posisjon enn opprinnelig vil det nye arrayet inneholde de samme verdiene, men i den siste gjeldende rekkefølgen. Husk at det opprinnelige arrayet består av flere arrayas 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.

The figure consists of four panels, numbered 1 to 4, arranged in a 2x2 grid. Each panel shows a user interface for a travel point list. The interface includes a 'Start:' input field, a 'Stopp:' input field, and a list of points. The points are represented as text boxes with '+' and '-' buttons next to them. The list is an unordered list (ul) where the order of points can be changed by clicking the buttons.

- Panel 1:** Start: Moss, Norge; Stopp: Kristiansand, Norge; List: Son, Norge, Drammen, Norge.
- Panel 2:** Start: Moss, Norge; Stopp: Kristiansand, Norge; List: Angi viapunkt ved behov, Drammen, Norge, Son, Norge.
- Panel 3:** Start: Moss, Norge; Stopp: Kristiansand, Norge; List: Angi viapunkt ved behov, Son, Norge, Drammen, Norge.
- Panel 4:** Start: Moss, Norge; Stopp: Kristiansand, Norge; List: Angi viapunkt ved behov, Son, Norge, Drammen, Norge.

Figur 4.8: Viapunkter med ett input-felt og en egen ul-liste.

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 tilfeldigvis vil legge til et ekstra viapunkt, 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 han legge til viapunktene på nytt. Start og stopp trenger han derimot ikke å legge til på nytt. Så lenge man ikke laster siden på nytt vil disse verdiene fortsatt være tilgjengelig i input-feltene. Det er heller ikke nødvendig å legge til funksjonalitet som tømmer disse input-feltene da de ikke har noe med sortering å gjøre.

For at brukeren ikke skal kunne legge til så mange viapunkter han vil, 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 han ikke kan legge 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 resettes. 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.

## 4.6 Veien mot endelig brukergrensesnitt

**Innholdet er ferdig, men teksten må skrives om fra blandet fortid/nåtid til nåtid**

Brukergrensesnittet vårt baserer seg mye på grunnlag av HResurs sitt brukergrensesnitt. Dette gjør vi fordi det endelige målet med denne oppgaven er å få applikasjonen vår integrert i HResurs. Designet er en 2 delt hovedside med enkel navigering frem og tilbake. 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. Når brukergrensesnittet er ferdig vil det bli direkte implementert i infotjenster sitt system, og gjennomgå grundigere testing internt i bedriften før det evt kan bli publisert for kunder.

dette er vår testversjon; *figur.4.9*

The mockup shows a web interface for a travel application. On the left, there is a form with the following elements:

- Kjøretøy:** A row of five radio buttons with icons for different vehicle types: a car, a motorcycle, a bicycle, a truck, and a bus.
- Start:** A text input field with the placeholder text "Angi startsted".
- Stopp:** A text input field with the placeholder text "Angi stopsted".
- Eventuell kommentar:** A larger text area for additional comments.
- Buttons:** Two blue buttons at the bottom: "Legg til viapunkt" and "Beregn".

On the right side, there is a large map area with a tabbed interface at the top containing four tabs: "Kart", "Resultat", "Kjørerute", and "Bomstasjoner". The "Kart" tab is currently selected. Below the map, there is an "Info" section with a warning message:

**Info**

*Dobbelsjekk at all informasjon stemmer i forhold til reiseruten  
Hvis reiseruten er korrekt, trykk bekreft. Hvis ikke, prøv å legge til et eller flere viapunkter og beregn rute på nytt*

At the bottom of the right panel, there is a blue button labeled "Bekreft rute".

Figur 4.9: Tidlig fremstilling av Brukergrensesnitt

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 this is a button labeled "Gå tilbake til mine reiser" and a "test" link. The main content area has a horizontal progress bar with steps: Start, Overnatting, Måltidsfradrag, Kjøring (highlighted), Utlegg, Tillegg, and Fullfør. The "Kjøring" step is active, showing a form with the following fields: "Rute" (a large text area), "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 expandable sections: "Passasjerer" and "Spesielle tillegg", each with a plus icon. A "Legg til" button is at the bottom left. On the right, an "Info" section provides instructions: "Kjøreruten skal beskrives så nøyaktig som mulig, med adresser for hvert delstopp. Kjøreruten 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." An example is given: "F.eks. Per sitter på 50 km, mens Kari sitter på 30 km. Da legger du inn 80 km med passasjer."

Figur 4.10: Hressurs design

figur.4.10 viser hvordan det nåværende systemet til Infotjenester ser ut, og det er det vi i utgangspunktet prøvde å ta stilling til. 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 form titled "Kart informasjon" (Map information). It contains the following fields and controls: "Start:" with a text input "Angi startsted"; "Stopp:" with a text input "Angi stopsted"; "Biltype:" with three radio buttons (car, motorcycle, truck); "Via:" with a text input "Via punkt"; a "Beregn" button; "Avstand:"; "Bompenger:"; "Du kjørte fra:"; "via:"; and "til:". To the right of the form is a map of Norway and Sweden, showing major cities like Bergen, Oslo, and Göteborg. The map is labeled "Norge Norway" and "Sver Sverige". The Google logo is visible at the bottom left of the map. The map data is dated 2015.

Figur 4.11: Tidlig brukergrensesnitt

Før vi bestemte oss for å lage et brukergrensesnitt som er mest mulig likt HRESSURS så hadde vi denne test modellen *figur.4.11* Denne er veldig lite funksjonell i forhold til det vi nå sitter igjen med. Vi har blandt annet oppgradert utseende på siden, gjort selve funksjonaliteten enklere og lagt til muligheten for opptil 5 viapunkter samt at ruten blir tegnet opp i kartet. Man kan nå også legge til en kommentar til reisen av hensyn til senere kontroll, f.eks hvorfor brukeren av systemet måtte kjøre om Gjøvik på vei til Lillehammer eller lignende.

Vi har også gjort om til en TAB basert visning på forsiden av hensyn til plass besparelse *figur.4.12*. Litt av hovedgrunnen til at vi valgte å legge all kjøredata i forskjellige tabber er for å minske informasjonen vi må presse inn på en side. Dette gjorde det også enklere for brukeren av systemet å ha god oversikt, noe som gjenspeiler ett brukevennligsystem. I tabbene formidler vi informasjon om ruten som ble lagt inn av bruker, her har vi med informasjon om Start, viapunkter, stopp, avstand, total bompengestnad og en beregnet kjøretid, vi har også en tab som lager en tekstlig beskrivelse av ruten, mer om dette i *kap.4.7.1*

The screenshot displays a web application for route planning. It features a top navigation bar with four tabs: 'Kart', 'Resultat', 'Kjørerute', and 'Bomstasjoner'. The 'Resultat' tab is currently selected. Below the tabs, the interface is divided into two main sections. The left section contains route statistics: 'Avstand: 582km', 'Total tid: 504min', 'Bompenger: 199kr', and a section titled 'Du har kjørt strekningen:' with details 'Start: gressvik' and 'Stopp: Trondheim S, Trondheim, Norge'. Below this is a 'Kommentar:' field. The right section, under the 'Bomstasjoner' sub-tab, lists toll stations along the route with their names and prices: Raukerud (23), E6 Europavegen (31), HOVINMOEN (19), Andelva (19), Tømte (15), Ørbekk (20), Espå (21), Kolomoen (23), Klett-E6 (10), E6 Tonstad (8), and Bjørndal (10). Above the statistics, a map shows the route from Bergen to Stockholm, with a red line indicating the path. The map includes labels for 'norge Norway', 'Bergen', 'Oslo', 'Stockholm', 'Göteborg', and 'Baltic Sea'. A Google logo is visible in the bottom left of the map area.

Figur 4.12: Innhold Tabvisning

Kartet i applikasjonen vår er satt til skjult frem til man beregner ruten. Grunnen til det er at kartet sliter med innlasting når man forstørrer eller forminsker bilde mens plottingen skjer, derfor løste vi det med å kun vise det etter plottingen er ferdig. Plottingen av den angitte ruten ser slik ut *figur.??* Mer om dette kommer også i *kap.4.7.2*

## 4.7 Fremstilling av beregnet kjørerute

### 4.7.1 Tekstlig fremstilling

Ruteplantjenesten er i hovedsak en ruteplanleggingstjeneste. Dette innebærer at innholdet tjenesten returnerer, bærer preg av å være tiltenkt reiseplanlegging og ikke dokumetering i ettertid slik som vår applikasjon skal. *Figur.4.13* viser et utklipp av en veibeskrivelse fra Halden til Drammen. Med mindre man har et kart man kan følge med på, eller følger beskrivelsen mens man kjører, 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 1time, og E6 har kun en tekstlinje, da det er 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, mens der kjører man til sammenligning bare i ca 10minutter.

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 4.13: 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 100-200m på og hvilken avkjørsel det er ut av hver enkelt rundkjøring, når veien man følger er den samme. Det er mer relevant å vite de store veiene, og det gir en overordnet og en 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.4.14.

<b>K2880</b> Ta til høyre på Hannibal Sehesteds Gate,
<b>F22</b> Ta til høyre på Fv22,
<b>R21</b> 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,
<b>E6</b> Hold til høyre på E6, I rundkjøring, ta andre avkjørsel og fortsett på E6,

Figur 4.14: Gruppert, tekstlig beskrivelse av kjøreruta.

Å gruppere etter veinummer gir med ett en bedre oversikt over hvor man har kjørt. For brukeren som skal kontrollere reiseruta vil dette vært et mye bedre alternativ enn beskrivelsen i figur xxxx, 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 xxxx 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 tomme arrayer slik som dette «[[[[[[]]]]]]», istedenfor den tekstlige beskrivelsen på formen «[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.

## 4.7.2 Grafisk fremstilling

### Konvertering av koordinater

Som nevnt i kapittel xXxxx, 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 plote 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 plote kun basert på angitte viapunkter i noen tilfeller slik som når 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 trafikkinformasjon(???) 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, så lik som mulig som ruta som er beregnet. Google har en mulighet for å angi viapunkter 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, som beskrevet i kap xxxx, 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 platform 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 en klasse DecompressGeometry. Det er ikke mulig å legge til kun denne klassen, i vårt eksisterende prosjekt, da den inneholder en WebReferanse til Arcgis. Ved å prøve å legge til denne Webreferansen i vårt prosjekt Ruteplantjenesten, får vi bare feilmeldinger om at referansen ikke finnes. For å få tatt vare på webreferansen 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 Debugvinduet, 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 på formen UTM 33N. Google krever koordinatene på desimalform, latitude og longitude. Dette er i likhet med converteren beskrevet i kap xxx, en omstendig omregning, slik at vi også her tar utgangspunkt i et ferdig eksempel. Vi gjør om eksempelkoden til å ta koordinatene fra DecompressGeometry og utmsone 33 som inputparametere, og legge resultatet inn i en ny multidimensjonal array. Vi legger til en ny attributt(????) på objektet som returneres til klientsiden, slik at vi får returnert koordinatene. Konverteringa kjøres en gang pr 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.

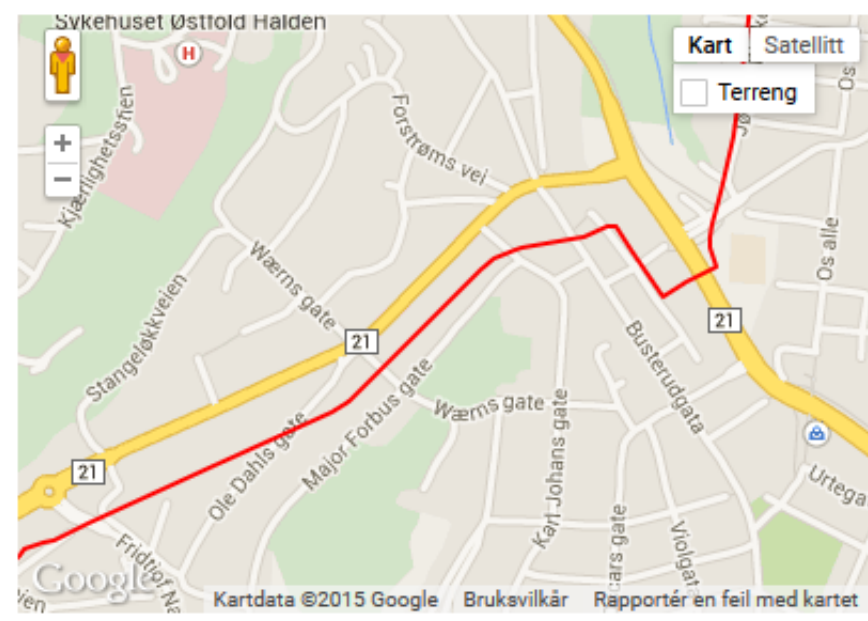
### Google Viapunkter

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.4.15* 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.

**Google Polyline** Valget faller på å prøve Googles polyline-funksjon. Denne plotter en mengde koordinater på kartet, og trekker en linje mellom disse. Jo flere koordinater, jo mer nøyaktig blir den inntegnede linja iforhold 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 først 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.4.16* viser.



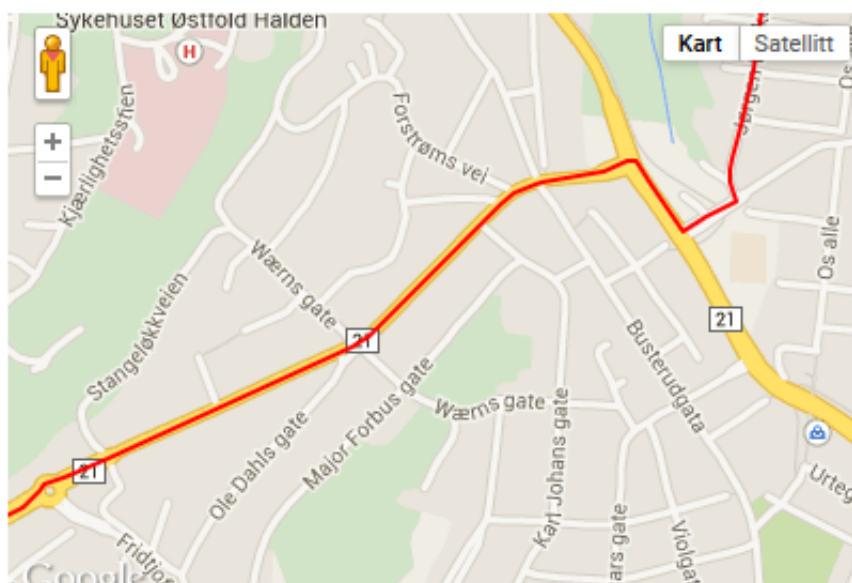
Figur 4.15: Kjørerute fram og tilbake på E6



Figur 4.16: 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 den veiene det er mest sannsynlig at en kjøreruta langs en polyline er. Problemet med denne 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 4.17. 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





Figur 4.18: Polyline følger kjørerute, uten forskyvning

at vi kan være sikre på at ruta vises slik den skal, og at den alltid vises. Dersom vi bruker polyline, spiller det ingen rolle linja havner utenfor veien, linja vises uansett. Ved å bruke alle koordinatene fra Ruteplantjenesten blir det tett nok mellom koordinatene til at selv å kjøre rett fram i en rundkjøring, blir tydelig på kartet. Ulempen er som nevnt under avsnittet «Konvertering av koordinater», at tidsbruken øker jo lenger strekningen er. Ved bruk av polyline trengs alle koordinatene for å få en nøyatig linje, men vi anser viktigheten av en korrekt rutevisning som alltid fungerer for å være større enn at det må gå superraskt.

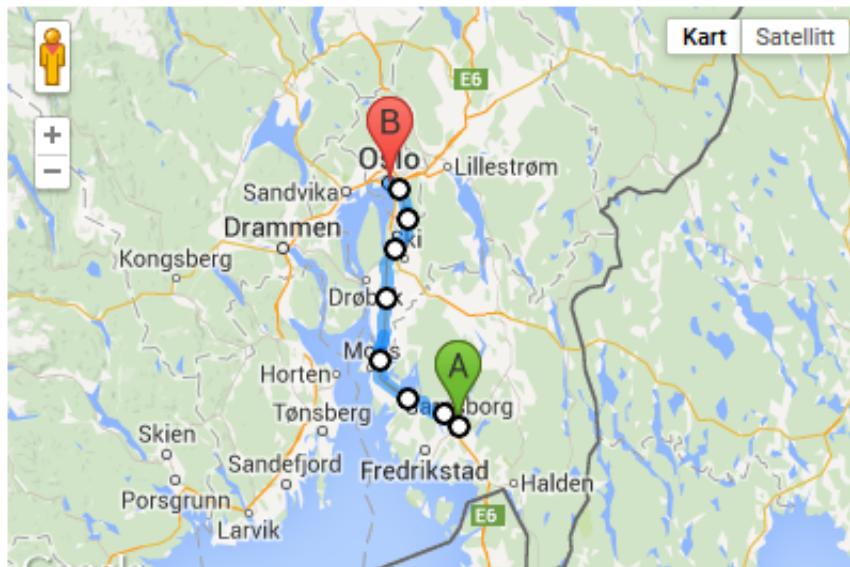
Utsende på den plottede kjøreruta har også betydning for at vi velger polyline framfor viapunkter. *Figur.4.19* viser hvordan kjøreruta ser ut etter å bli plottet med viapunkter. På en kort strekning slik som Sarpsborg – Oslo, der de åtte punktene har en kort avstand mellom hverandre, fremstår viapunktene som forstyrrende elementer. På en lengre strekning vil punktene fordeles utover et større område, slik at de ikke fremstår like forstyrrende.

*Figur.4.20* 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.

### Optimalisering av koden

Etter et møte hos infotjenester der vi demonstrerte vår foreløpige applikasjon for Petter, 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 stort 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, er i fellesskap å prøve å finne ut hvorfor beregningene tar så lang tid. Petter forelår at det kan være dekomprimeringsmetoden som tar tid, da denne som





Figur 4.19: Sarpsborg - Oslo, fremstilt med viapunkter



Figur 4.20: Halden - Oslo, fremstilt som polyline

nevnt tidligere i kapitlet, foregår med en webreferanse 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 foregå via Googles server, skal foregå på infotjenester sin server. Fra testing og debugging, har vi derimot slått fast at dekomprimeringen går veldig raskt, og at det først er når konverteringa starter, at beregningene begynner å gå sakte. Det kan være mange grunner til at konverteringa 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 en gang. En reiserute fra Halden til Kirkenes

inneholder ca 30 000 koordinater, og dersom vi kan beregne de fleste verdiene som trengs til konverteringa, bare en gang, istedenfor 30 000 ganger, kan dette føre til en raskere beregning.

Som en måte kontrollere koordinatene som ble beregnet, la vi i implementeringsfasen av konverteringsmetoden, på en debug-utskrift. Når vi så går i gang med å optimalisere konverteringa, 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, enn en kort, men det går vesentlig raskere, og hastigheten er uten tvil akseptabel.

Samtidig som vi jobber videre på å utbedre kode, begynner noen på gruppa å se på mulighetene for å publisere applikasjonen, slik at vi kan få testet denne. Det dukker raskt opp et problem med delprosjektet DecodeGC. Pga DecodeGC lar ikke prosjektet se publisere, og vi har dermed enda en viktig grunn til at vi må gå bort fra DecodeGC og NAServeren. Vi vil gå nærmere inn på publiseringen i kap 4.9. 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 denne 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 feil sted i koden, og at uriktig kodestruktur gjorde at koden ikke fungerte. Denne var første ferdige C#-kodesnutt vi testet, og etter å ha vært innom en del forskjellige eksempler i forbindelse med konverteringen, har vi fått en mye bedre forståelse for hvordan vi best mulig kan ta i bruk kodesnutter.

## 4.8 Tilpassninger til HResource - Veien mot implementering

### 4.8.1 Optimalisering av koden - Javascript

Etter de siste endringene vedrørende viapunkter, har vi gjennomgått all Javascript kode frontend 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 skriptene, 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 samlet det meste av Javascript koden i to egne filer som refereres til i index-fila. 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 er ikke veldig mottakelig for redigering, og vi er nødt til å beholde både globale variabler og 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 kalkulator for hver av koordinatene vi har trengt konvertert inne i konverterskriptet.



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 måtte forandre for mye på Google APIet. Det lot seg gjøre ved å sende med start og stopp verdiene som parametere til selve funksjonen. I stedet for å opprette globale variabler for lengde- og breddegrader hver gang Google APIet kjøres, 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 i meter. Kalkulatoren kan få disse verdiene på to måter: enten hente variabler fra et 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 to av de 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. Men 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 til lik de globale variablene, setter man dem til lik parameterne i stedet. Eksempelet under viser hvordan dette ser ut i kode for start-koordinatene.

Kodeeksempel 1: De globale variablene lages ved å putte «window» foran variabelnavnet. Disse sendes ikke som parametere.

```
window.longstart = long1;  
window.latstart = lat1;  
functionkonverter();
```

Kodeeksempel 2: Variablene sendes som parametere og trenger derfor ikke å være globale.

```
var longstart = long1;  
var latstart = lat1;  
functionkonverter(longstart, latstart);
```

Vi sender variablene som parametere ved å sette variablene inn i parentesen som etterfølger funksjonsnavnet. I konverteringsskriptet ser starten av funksjonen nå slik ut:

```
functionkonverter(y,x);
```

Som sagt forventer denne nå å få inn parametere. Vi har valgt å sette de til y og x, men det kunne stått hva som helt der. De lokale variablene i konverteringsfunksjonen settes til lik henholdsvis y og x, i stedet for til lik to globale variabler utenfor funksjonen. Y og

x vil for hver gang funksjonen kjører inneholde ulike verdier. I vårt tilfelle vil de først inneholde koordinatene for start, og deretter koordinatene for stopp.

### 4.8.2 Optimalisering av koden - C#

I vår applikasjon har vi benyttet mange lister, blant annet nøstede 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 vært 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 dekomprimeringa og konverteringa. Her er koordinatene lagret som en liste med array av double (desimaltall). For å hente ut første koordinat må vi gjøre det på 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 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 nøstede lister og lister med array.

Vi tilpasser også koden slik at objektet som returneres til klientsiden, inneholder et eget objekt som kan sendes legges 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 pr 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 hvert trinn i løkka. Ved bruk av foreach beregnes lista kun en gang, før programmet går gjennom hele lista/arrayet.

I skolesammenheng har for-løkker blitt brukt oftere enn foreach, og fordelene med foreach har Petter gjort oss oppmerksom på, da han ble obs på at vi konsekvent brukte for-løkker framfor foreach.

### 4.8.3 Tilpassning av Brukergrensesnitt

Vi sender over skjermbilder av vår applikasjon, til HResurs-ansvarlig xxxxxxxx, som skal se på hvilke endringer som bør gjøres før implementasjon. Her vil det komme en nærmere beskrivelse når vi får mer informasjon.

## 4.9 Publisering på skolens server

For å få testet applikasjonen uten å kjøre den via Visual Studio, utviklingsverktøyet vi har benyttet, er vi nødt til å laste opp applikasjonen på en server hos Høgskolen i Østfold. Det er ikke bare å laste den opp som en vanlig nettside, ettersom vi benytter oss av en web service som må ligge og kjøre i bakgrunn. Det er denne som foretar alle forespørslene mot Ruteplantjenesten, og uten at web servicen kjører, er det inntasting av steder som fungerer. Ved å publisere applikasjonen gjennom et publiseringsverktøy i Visual Studio, skal applikasjonen være klar til å lastes direkte opp på en server. Dette fungerer ikke for oss, web servicen vil ikke kjøre.

Vi ta kontakt med IT-ansvarlig og får beskjed om at det bare skal være å laste opp på frigg, der vi allerede har en katalog der vi har lastet opp nettsiden vår. Det er dette vi allerede har gjort. Etter feilsøking og flere samtaler med IT-ansvarlig finner vi ut at feilen oppstår siden frigg er Linuxbasert, mens webservicen vi skal kjøre fra serveren er en windows-web service.

Vi sender så en forespørsel til webansvarlig for windows, for å få opprettet et domene på Donau (Windows skoleserver). Når vi omsider får tilgang, fungerer det heller ikke. Det viser seg at serveren kjører et for gammel .NET-rammeverk (2.0), og vi må igjen vente. Når serveren blir oppdatert til 4.5 rammeverket som applikasjonen vår er laget i, fungerer alt som det skal, og vi er klare for å brukerteste applikasjonen. Vi måtte utesette brukertestinga i ca en uke pga ventetid med oppretting av domene og Framework feil.

## 4.10 Implementering

Her vil vi beskrive prosessen med å implementere applikasjonen mot HResurs. Vil finne sted i begynnelsen av mai.

## 4.11 Hvordan kan applikasjonen videreutvikles?

Hvordan kan applikasjonen utbedres ved et senere tidspunkt? Hvilke ideer har vi, men som vi ikke har gjennomført/evt testet så smått?

Ide 1:

Ferjekostnader på strekningen - slå opp priser på ferjene på strekningen. Må også angis antall personer i bilen, da det ofte er pris for bil med fører, og tillegg for pasasjerer

Ide 2:

Hva hvis kjøreruta går innom Sverige? Beregne bompriser i Norge, km i Norge etter norske satser, og km i sverige etter utenlandssatser

Ide 3:

Mobiltilpasset versjon av applikasjonen. Mest sannsynlig er dette allerede overordnet i HRessurs.

## 4.12 Programmer, verktøy og hjelpemidler

### 4.12.1 programmer og nettbaserte verktøy vi har brukt:

- Visual Studio 2013
- Visuam Paradigm Free UML design tool
- Surveymonkey.com
- Github
- Dropbox
- Google Forms

### 4.12.2 Innhenting av informasjon

Den informasjonen vi har trengt for å kunne fullføre arbeidet har blitt innhentet hovedsakelig fra nett. Glenn organiserte også et møte med en av hans forelesere Per Bisseberg ved HIOF for å få flere synspunkt på vårt arbeid, samt mer informasjon om hvordan ASP.Net applikasjoner med web service virker.

- Stackoverflow: Dette nettsamfunnet har vi benyttet til alle deler av arbeidet
- Youtube: Her finner man mange gode tutorials. Særlig har jeg lett etter informasjon på hvordan man bruker en web service i Visual Studio.

- Google Developers: Fremgangsmåte for å bruke Googles autocomplete-funksjon (og Google Maps).
- W3School: Mye grunnleggende programmering, spesielt innen HTML og Javascript, er beskrevet her, med tilleggsfunksjon for å teste/modifisere eksemplene.

## 4.13 SCRUM - utførelse

**Dette kapittelet er ikke ferdig formulert, skal beskrives mye mer nøye, få med grafisk fremstilling av gjennomføringen av sprinter og Issues i Github samt mer forklarende, for øyeblikket på stikkordsform. Her skal det diskuteres hvordan det gikk, om vi rakk alt, hva som eventuelt ble utsatt til neste sprint og hvorfor dette skjedde**

### **4-18 feb**

Request mot ruteplantjenesten til webservice, webservice(serverside) og brukergrensesnitt (klientside) skal fungere sammen. Dummy data ferdig kordinater, henter stedsnavn fra klientside. Hente ut avstand og bompenger.

### **18feb - 12 mars**

Request mot ruteplantjenesten til webservice, webservice(serverside) og brukergrensesnitt (klientside) skal fungere sammen. Dummy data ferdig kordinater, henter stedsnavn fra klientside. Hente ut avstand og bompenger.

### **12 mars – 24 mars**

Flytte logikk fra webservice til TravelRoute(implementerer ITravelRoute) hente koordinater fra klientside til start og stopp. Hente ut alle bomstasjoner på ruta, Feilhåndtering på request mot ruteplantjenesten.

### **24 mars – 8 april**

Veibeskrivelse, Plotte rute på kart feilhåndtering med forskjellige søk, konvertering av koordinater tilbake til LatLng 8 april – 15 april Komprimert veibeskrivelse, eget objekt som returneres til HResurs (ett objekt med bomutgifter og ett objekt med kjørerute) gjøre om Array og Liste struktur til Objekt struktur

### **8 april – 15 april**

Komprimert veibeskrivelse, eget objekt som returneres til HResurs (ett objekt med bomutgifter og ett objekt med kjørerute) gjøre om Array og Liste struktur til Objekt struktur

## 5. Test av Applikasjon

### 5.1 Brukerevaluering

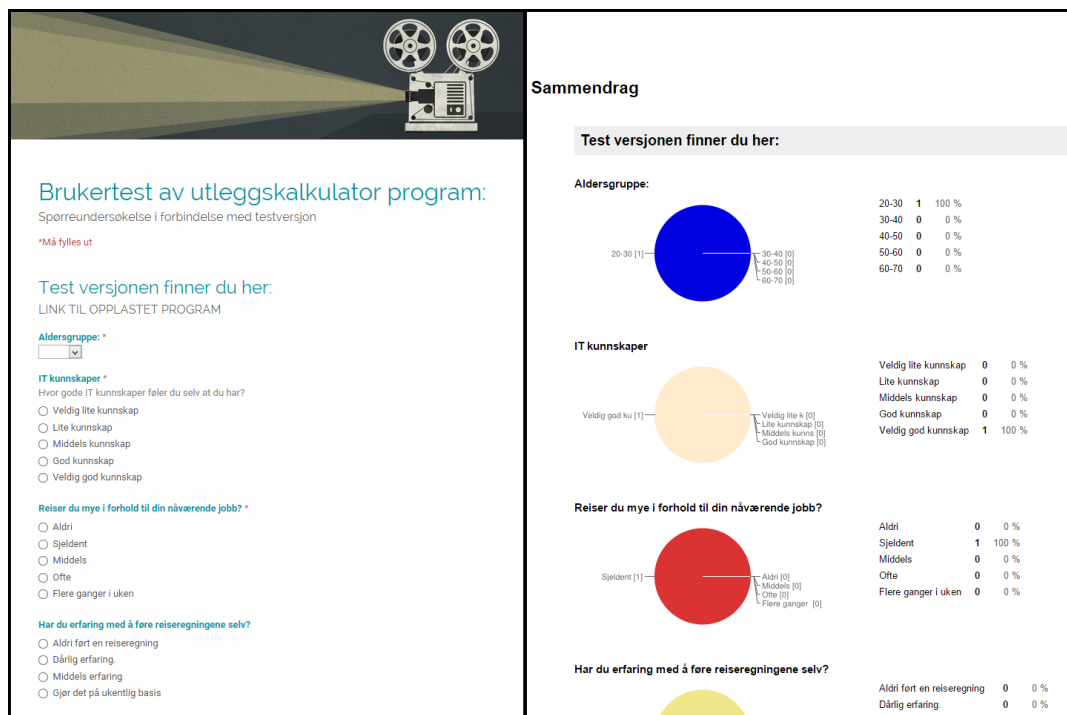
Gjennom hele prosjektprosessen har vi kvalitetssikret applikasjonen vår ved hjelp av SCRUM-møtene. Da har vi i samarbeid med oppdragsgiver gått igjennom funksjonalitet i forhold til brukergrensesnittet og design og struktur på koden. Det har gjort at vi hele veien har vært fleksible for endringer og at vi har fått et bredere perspektiv igjennom store deler av utviklingsprosessen. Når det nå nærmer seg implementering, har vi valgt å foreta en brukerevaluering av applikasjonen. Vi planlegger å gjøre dette for å få relevante meninger fra de som vil komme til å bruke applikasjonen i fremtiden. Det kan være veldig lett for oss som utviklere å låse oss fast i vår egen tankegang av hvordan ting skal være. Vi utvikler koden og brukergrensesnittet, slik at vi vet nøyaktig hvordan det funker og i hvilken rekkefølge ting må utføres for å fungere. Det viser seg ofte at det i en bruker setting burde vært gjort på en annen måte, da applikasjonen ikke er selvforklarende for andre enn oss selv.

På serveren på donau, som beskrevet nærmere i kap xxxx, publiserer vi siste versjon av applikasjonen. Denne er mer eller mindre ferdig til å implementere i HRessurs, det mangler bare siste finpussen for at brukervennligheten skal være optimal.

Vi har bestemt oss for å lage ett spørreskjema for å få tilbakemeldinger fra testpersonene våre. Da har vi konkrete tilbakemeldinger og forholde oss til, og vi kan få svar på de spesifikke tingene vi er interessert i å vite. Vi har mulighet til å formulere spørsmålene, slik at de er lukkede nok til at vi kun får svar på det vi lurte på, men at de samtidig er åpne nok til at vi kan få et utdypende svar. Ved en muntlig tilbakemelding blir det fort et svar av typen: Det fungerte helt greit”, uten nærmere utdypning.

Vi begynte tidlig å se på mulighetene for å opprette en spørreundersøkelse på nett. Vi testet da SurveyMonkey. I ettertid velger vi heller å bruke Google Forms til spørreundersøkelsen. Her er det mulig å stille flere spørsmål enn i SurveyMonkey som har en begrensning på ti spørsmål. Google Forms gir oss i tillegg en enkel og ren, både tekstlig og grafisk fremstilling av spørsmålene og resultatet. Her kan vi få frem grafer bare ved et trykk. ref. *figur.5.1*

I forbindelse med spørsmålene og de forskjellige bakgrunnene til personene som tester applikasjonen og besvarer undersøkelsen vår, er vi nødt til å bestemme hvilke tilbakemeldinger vi skal fokusere på når vi utbedrer feil og mangler som kommer fram under testingen. Vesentlige feil må utbedres uansett, men vi er nødt til å ha bestemt hvilke tilbakemeldinger vi skal legge vekt på dersom det er store sprik i meninger om applikasjonen. Vi vil da legge vekt på de tilbakemeldingene hvor det kommer frem at personen har middels til gode IT kunnskaper og som allerede er, eller vil kunne være potensielle brukere av applikasjonen. Dette gjelder for eksempel de som ofte fører reiseregninger selv.



Figur 5.1: Eksempel fra Skjema og fremvisning



Vi kommer frem til at vi først ønsker å kartlegge testpersonen bakgrunn, før vi stiller spørsmål angående applikasjonen. Spørreundersøkelsen blir derfor delt inn i to deler, bakgrunn og applikasjon.

**Generelle spørsmål:**

1. Aldersgruppe:
2. IT kunnskaper
3. Reiser du mye i forhold til nåværende jobb?
4. Har du erfaring med å føre reiseregningene selv?

**Applikasjonen:**

1. Var applikasjonen lett forståelig?
2. Opplevde du feilmeldinger, problemer eller treghet når du testet?
3. Er det noen funksjoner du føler at mangler?
4. Er det noe du synes kunne vært gjort annerledes?
5. Hva var bra, eventuelt dårlig?

## **5.2 Resultater av brukerevaluering:**

Denne seksjonen kommer så fort brukerevalueringen er gjennomført og resultatene er behandlet.

## 6. Diskusjon

### **Skrives når vi nærmer oss ferdig.**

Her skal det dokumenteres at vi har lært noe undersis, ikke bare levert et produkt til oppdragsgiver. I hvilken grad ble målene nådd? Ble leveransene fullført? Hvordan metoden fungerte?

Bør nevne om flerdelt prosjekt, at vi på skolen kun har hatt programmering på et nivå. Lært å splitte opp. Diskutere for om løsningen oppfyller alle deler, om målet med enklere reiseregistreing er nådd. Fikk vi levert alt som er beskrevet under leveranser? Hva var med i hver av leveransene iforhold til hva som skulle være med? Hvordan fungerte metoden (Scrum?), hadde vi noe nyttig av det? Ble det noen endringer underveis pga scrum? Hva fungerte bra/ikke bra? Hva ville vi gjort annerledes? Hvilke problemer oppsto?

Alle forskjellige viapunker, mange måter å gjøre det samme på. Mye tid på å finne den beste måten å løse dette på, både med tanke på kodemessig og brukergrensesnitt, slik at dette også skulle. Autocomplete-problemer, ikke kunne trykke i tekstboks i firefox osv.

## 7. Konklusjon

Sammendrag av diskusjonskapittelt. Legge vekt på det viktigste vi fant ut/lærte. Godt diskusjonskapittel = konklusjonen kan holde med en side. Legg vekt på tydelig språk. Sensor leser sannsynligvis først sammendrag, så konklusjon. Hvordan ble produktet iforhold til våre/oppdragsgivers forventning? Gjenta de viktigste punktene fra diskusjonskapittelet. Hva bør bli gjort videre ved en evt videreføring av prosjektet? Evt gi råd for de som skal jobbe videre med det.

# Bibliografi

- [1] Bomstasjoner langs E6  
<http://www.e6bompenger.no/Bomstasjonene-2.aspx>
- [2] Artikkel om bompenger Halden:  
[http://www.ha-halden.no/\\_vi\\_kommer\\_ikke\\_utenom\\_bompenger-5-20-19908.htm](http://www.ha-halden.no/_vi_kommer_ikke_utenom_bompenger-5-20-19908.htm)
- [3] Autopass:  
<http://www.autopass.no/obligatoriskbrikke/om-obligatorisk-brikke>
- [4] Takster Bompenger:  
[http://www.vegvesen.no/\\_attachment/181865/binary/348996?fast\\_title=Takster](http://www.vegvesen.no/_attachment/181865/binary/348996?fast_title=Takster)
- [5] Bompengeselskaper:  
<http://www.autopass.no/Bompengeselskap>
- [6] Bomstasjoner Østfold:  
<http://www.ostfold-bompengeselskap.no/service/#oversikt-bomstasjoner>
- [7] Takster Østfold:  
<http://www.ostfold-bompengeselskap.no/takster/>
- [8] Svinesundsforbindelsen, autopass:  
<http://www.svinesundsforbindelsen.no/autopass.html>
- [9] Rabatter:  
<http://bompenger.no/Takster/Nyrabattstruktur.aspx>
- [10] Miljøpakken:  
<http://miljopakken.no/om-miljoepakken/om-organisasjonen>
- [11] Miljøpakkens mål:  
<http://miljopakken.no/om-miljoepakken/maal>
- [12] Miljøpakken bompunkter:  
<http://miljopakken.no/om-miljoepakken/bompunkter>
- [13] Haugesunds regler:  
<http://www.haugalandspakken.no/service/#timesregel>
- [14] Reiseregningsregler i bedrift  
<https://www.altinn.no/no/Starte-og-drive-bedrift/Drive/Arbeidsforhold>
- [15] Vis veg, ruteplanlegger:  
<http://visveg.vegvesen.no/Visveg/>
- [16] NAF's ruteplanlegger:  
<https://www.naf.no/tjenester/ruteplanlegger>

[17] 1881's ruteplanlegger:

<http://www.1881.no/Kart/Veibeskrivelse/>

[18] Google Maps:

<https://www.google.no/maps/dir/>

[19] Gulesiders ruteplanlegger:

<http://kart.gulesider.no/veibeskrivelse>

[20] Kvasir:

<http://kart.kvasir.no/>

[21] Universiell utforming - regler:

<http://uu.difi.no/regelverk/tidsfristar-ny-og-eksisterande-ikt>

[22] Universiell utforming - regler:

<http://uu.difi.no/regelverk/kven-skal-folgje-krava>

[23] Universiell utforming - krav til nettsider:

<http://uu.difi.no/veiledning/nettsider/krav-til-nettsider/oppbygging-a>





