Avstand og bompengeberegningsmodul til HRessurs

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

Bacheloroppgave
Avdeling for informasjonsteknologi
Høgskolen i Østfold"

13.03.15

HØGSKOLEN I ØSTFOLD

Avdeling for Informasjonsteknologi
Remmen
1757 Halden
Telefon: 69 21 50 00
URL: www.hiof.no

BACHELOROPPGAVE

Prosjektkategori: Bachelorprosjekt	X	Fritt tilgjengelig
Omfgang i studiepoeng: 20		Fritt tilgjengelig etter
Fagområde: Informasjonsteknologi		Tilgjengelig etter avtale med oppdragsgiver

Tittel: Utvikling av reiseregnignsmodul	Dato: 17. mars 2015
Forfatterere: Robin Furu, Ingvild Bjørlo Karlsen, Christian Jacobsen, Glenn Bjørlo	Veileder: Terje Samuelsen
Avdeling / Program: Avdeling for Informasjonsteknologi	Gruppenummer: BO15-G02
Oppdragsgiver: Infotjenester AS	Kontaktperson hos oppdragsgiver: Petter Ekran

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

Figurliste **5**

Tabelliste **7**

Kodeliste **9**

1 Introduksjon **1**
1.1 Prosjektgruppen 1
1.2 Oppdragsgiver 2
1.3 Oppdraget 4
1.4 Formål, Metode og Leveranser 4
1.5 Rapportstruktur 7

2 Bakgrunnsanalyse **9**
2.1 Eksisterende løsninger samt regler og hensyn. 9
2.2 Teknologi 18

3 Design/utforming/planlegging **21**
3.1 Struktur/innhold/utforming 21
3.2 Uferdig tittel/kapitel - mal/leveranse 21

4 Implementasjon/produksjon/gjennomføring **22**
4.1 Alt i ett løsning 22
4.2 Innhenting av informasjon 30
4.3 SCRUM - utførelse 31

5 Test av produkt **32**
5.1 Testing 32
5.2 Evaluering 32

6 Diskusjon **34**

7 Konklusjon **35**

Register **37**

Figurer

1.1 Oversikt over Infotjenesters produkter 3
1.2 Grafisk fremvisning av tenkt metoder. 5
1.3 Gantdiagram - Planlagt fremdrift 7

2.1 Eksisterende løsning 9
2.2 Vår tenkte løsning 10
2.3 Skjermdump fra HRessurs 10

4.1 Viapunkter, CSS og Javascript 24
4.2 Viapunkter til å manipulere 25
4.3 Strukturen på CalculatedRoute 26
4.4 Strukturen fra Web Service til Ruteplantjenesten 28
4.5 Strukturen på interfacet 28
4.6 Viser eksempel på hvordan man henter Google API Autocomplete basert på steder. 29

Tabeller

Kodeliste

Kapittel 1

Introduksjon

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

90204584 - Invgilkb@hiof.no, 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

41524376 - robinaf@hiof.no, 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 Jacbosen

41758575 - chrisjac@hiof.no, er født og oppvokst i Fredrikstad og bor på Gressvik. Han har tidligere erfaring som teknikker 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

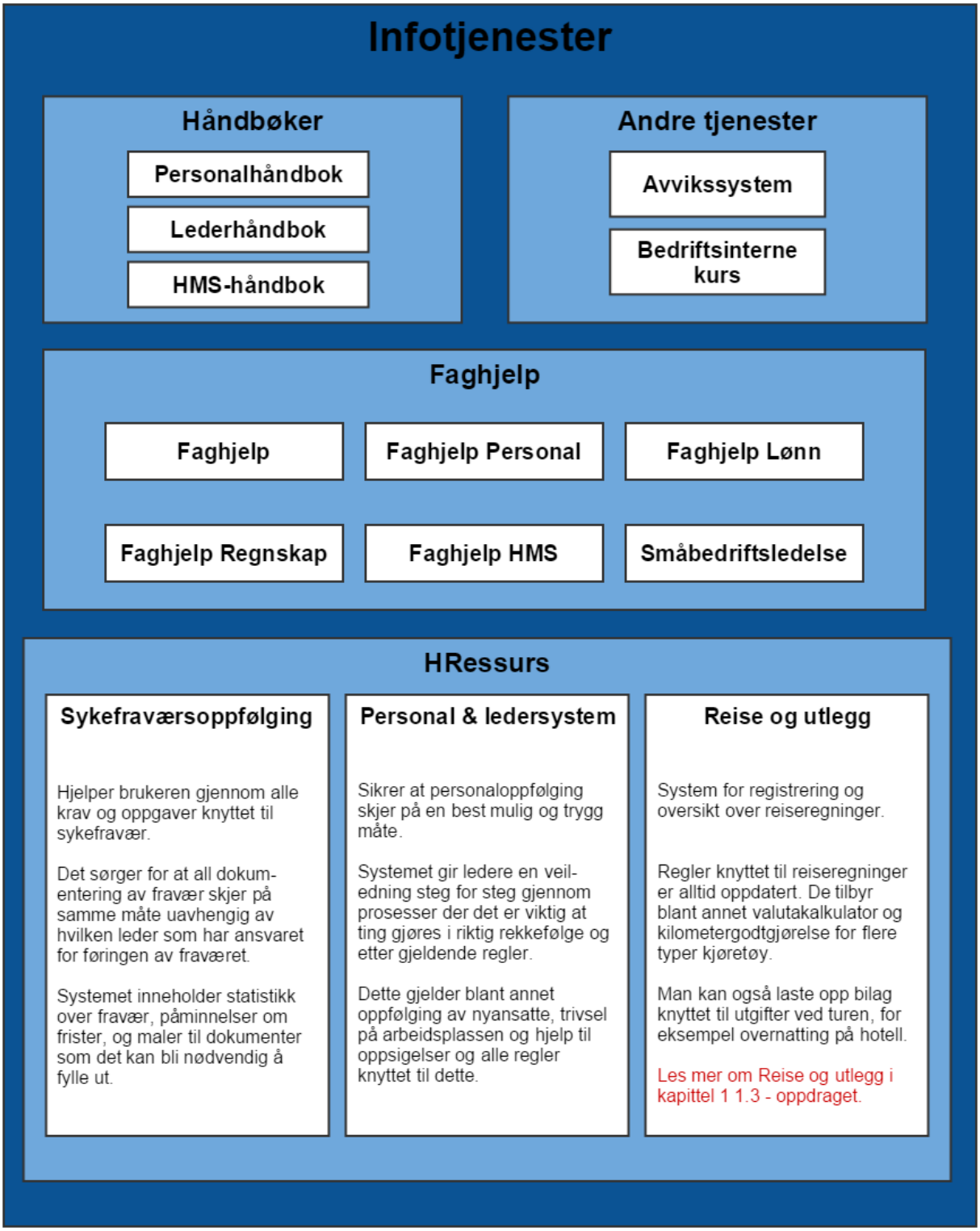
47384852 - glennab@hiof.no, er oppvokst i Askim og har bodd i Halden siden høsten 2014. Han ble uteksaminert fra Askim Videregående skole i 2005 og flyttet deretter til Lillehammer hvor han studerte film- og fjernsynsvitenskap. Senere flyttet han til Sarpsborg og begynte studiet DMpro. Han har nylig 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 og de veileder kundene med dette i tillegg til å tilby konsulenttjenester. 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 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.

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, Metode og Leveranser

1.4.1 Formål

Da man i dag enten selv må holde styr på og notere ned bompasseringer og kostnader underveis, eller ta i bruk ekstern teknologi for å beregne utgiftene i ettertid, er målet vårt å gjøre denne beregningen enklere for brukerne. Man skal slippe å notere bompasseringer underveis, og også slippe å ta i bruk eksterne løsninger for å beregne avstand og kostnader. I dag må kjøreruta også beskrives detaljert, men også denne jobben skal applikasjonen ta seg av. Alt brukeren må gjøre er å angi hvor han har kjørt i form av adresser på ruta. Start, stopp og viapunkter.

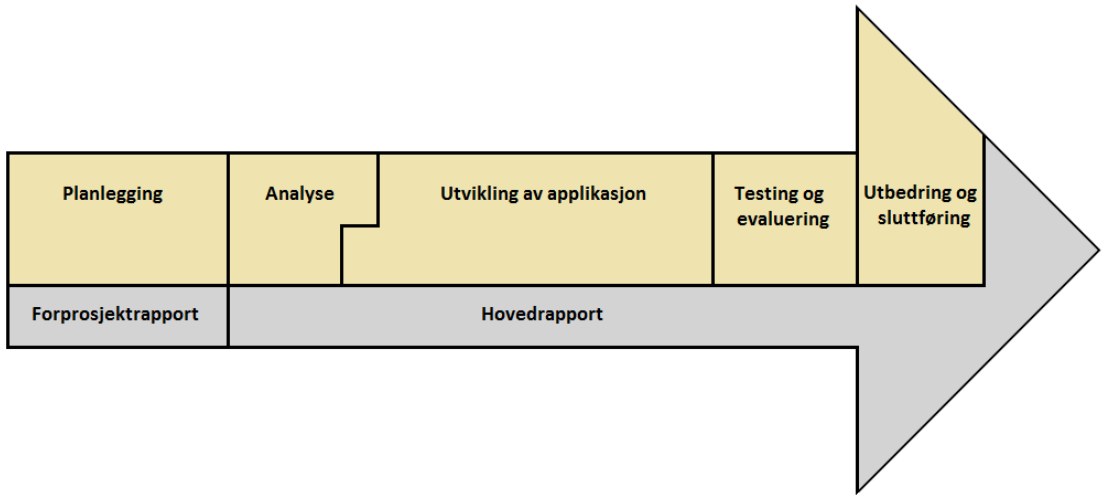
Hovedmål
: Gjøre reiseregning/beregning lettere

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 Metode

I dette kapittelet vil vi ta for oss metoden vi skal bruke for gjennomføring av prosjektet. I hovedsak kan prosjektperioden deles inn i fem klare faser, se ref. *Figur 1.2* 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 1.2: 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.

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.

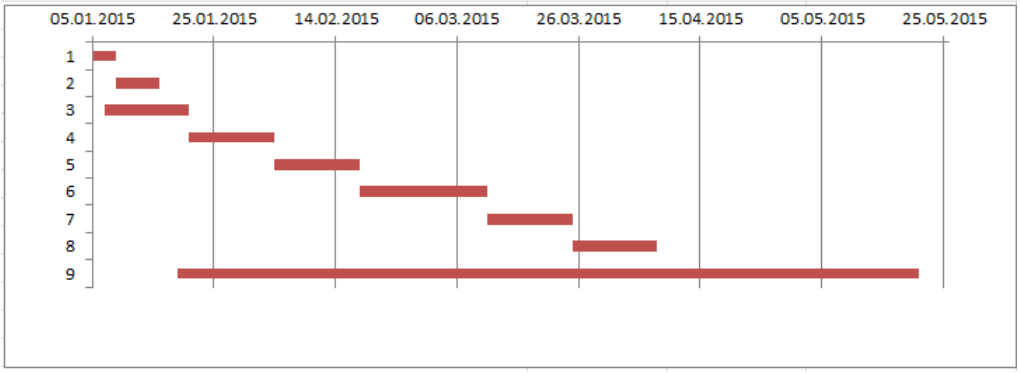
De første ukene vil være en typisk oppstartsfase, der vi vil bruke en del tid på planlegging og forprosjektrapporten, i tillegg til at alle må sette seg inn i programmeringsspråkene oppgaven baserer seg på. I denne perioden skal vi også finne ut om prosjektet er gjennomførbart, blant annet ved å finne 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. Vi vil her 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 erstattes 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.

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 lagret 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.

1.4.3 Fremdriftsplan

Aktivitet	Start dato	Slutt dato	Periode i dager
1. Hjemmeside for prosjektet	05.01.2015	09.01.2015	4
2. Forprosjektrapport	09.01.2015	16.01.2015	7
3. Sprint 1: Lage GUI frontend	07.01.2015	21.01.2015	14
4. Sprint 2: Koble GUI mot Google API og Ruteplantjenesten	21.01.2015	04.02.2015	14
5. Sprint 3: Koble sammen frontend og web service backend	04.02.2015	18.02.2015	14
6. Sprint 4: Lage via-punkter frontend. Sende request fra web service mot Ruteplantjenesten	18.02.2015	11.03.2015	21
7. Sprint 5: Retting og testing	11.03.2015	25.03.2015	14
8. Sprint 6: Design, tilpasning til HRessurs	25.03.2015	08.04.2015	14
9. Hovedrapport	19.01.2015	21.05.2015	122



Figur 1.3: Gantdiagram - Planlagt fremdrift

1.5 Rapportstruktur

Dette kapittelet er ikke ferdig utformet, det som kommer under er kun en kladd, som vil bli skrevet om når rapporten er tilnærmet ferdig.

Kapittel 1 - Introduksjon

Introduksjonskappitellet omhandler i all hovedsak informasjon om bachelorgruppen, hvordan gruppen er satt sammen av diverse kompetanser. Beskrivelse av oppgradgsgiver samt diverse opplysninger om firmaet. Oppdraget i seg selv, som hva vi har fått i oppdrag og utføre og hva som foreventes. Det står også listet endel formål og leveranser som blir brukt og tatt med i prosjektet samt hva vi har tenkt å levere og hvordan, pluss litt om hvilken type utviklingsmetode som er brukt i oppgaven

Kapittel 2 - Bakgrunnsanalyse

Bakgrunnsanalyse kapittelet inneholder genrell 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 kapitellet finner du utdypende informasjon om prosjektetsoppbyggning

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 - Systemtesting
I dette kapittelet beskriver vi resultatene rundt testing av systemet mot et knipe tilfeldige brukere.

Kapittel 6 - Diskusjon
I dette kapittelet 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 oppnåd og evt hva som ikke ble gjennomførtbart.

Kapittel 2

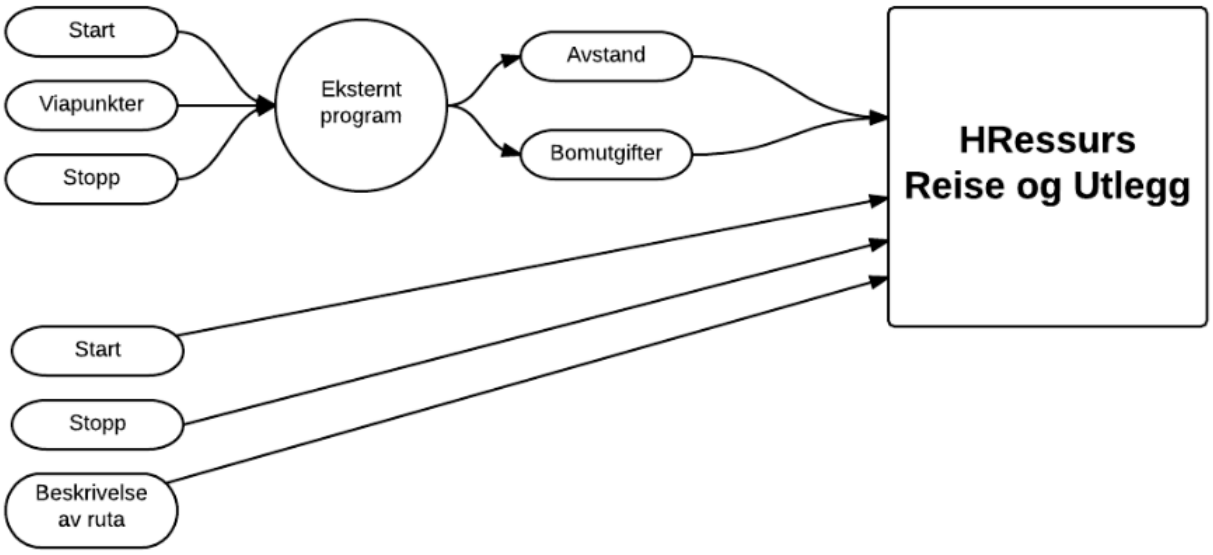
Bakgrunnsanalyse

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 Eksisterende løsninger

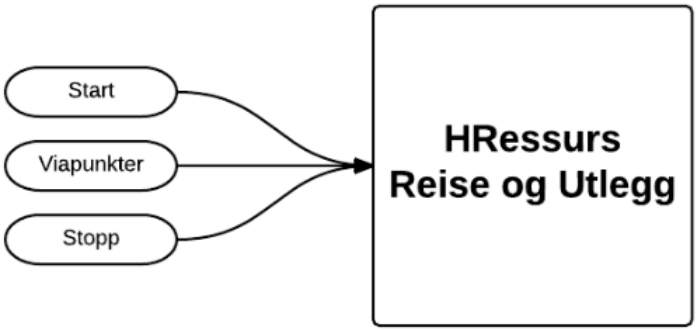
Dette er dagens løsning ved bruk av ekstern teknologi:



Figur 2.1: 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:

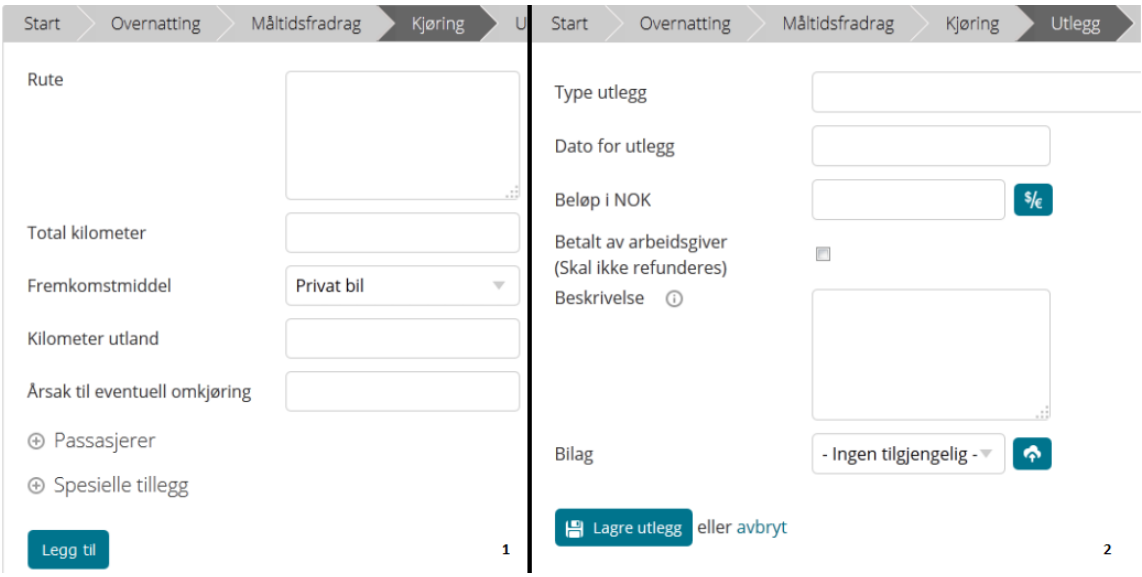
Dette vi være en bedre løsning både for de ansatte som kjører og firmaet, da beregningen foretas automatisk i det de registrerer reiseregningen. Løsningen er bedre for de



Figur 2.2: Vår tenkte løsning

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.

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 2.3* viser et skjermbilde av dagens HRessurs.



Figur 2.3: 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.

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 [1] 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» [3]Straffen for å ikke følge disse reglene er bot på 8000kr, og denne satsen dobles 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 [5] 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,[6],[7] 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 [8] 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 [9] er et selskap som tilbyr både forskudd og etterskuddsbetaling for sine abonnenter, men her fordeler rabatten seg på 10% for privatkunder med

[1] http://www.e6bompenger.no/Bomstasjonene-2.aspx

[2] http://www.ha-halden.no/__Vi_kommer_ikke_utenom_bompenger-5-20-19908.html

[3] http://www.autopass.no/obligatoriskbrikke/om-obligatorisk-brikke

[4] http://www.vegvesen.no/_attachment/181865/binary/348996?fast_title=Takster+i+bompengeanlegg.pdf

[5] http://www.autopass.no/Bompengeselskap

[6] http://www.ostfold-bompengeselskap.no/service/#oversikt-bomstasjoner

[7] http://www.ostfold-bompengeselskap.no/takster/

[8] http://www.svinesundsforbindelsen.no/autopass.html

[9] http://bompenger.no/Takster/Nyrabattstruktur.aspx

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 dobles taksten for å passere bomstasjonene i Trondheim sentrum. Disse tilhører prosjektet «Miljøpakke Trondheim» [10] Resterende bomstasjoner i Trondheimområdet [11] 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 [12] 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 [13] Denne praksisen finner man blant annet igjen i Haugalandspakken [14] som dekker bomstasjoner i Haugesund og på Karmøy, der man også kun betaler en passeringer 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 HRessurs og en eventuell autopassavtale han eller hun måtte ha. Etterskuddsfakturering 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 etterskuddsfakturering, 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.» [15] 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.

[10] `http://miljopakken.no/om-miljoepakken/om-organisasjonen`
[11] `http://miljopakken.no/om-miljoepakken/om-organisasjonen`
[12] `http://miljopakken.no/om-miljoepakken/maal`
[13] `http://miljopakken.no/om-miljoepakken/bompunkter`
[14] `http://www.haugalandspakken.no/service/#timesregel`
[15] `https://www.altinn.no/no/Starte-og-drive-bedrift/Drive/Arbeidsforhold/Lonn/Reiseutgifter/Dekning-etter-regning-/`

2.1.3 Eksisterende ruteplanleggingstjenester

Når vi var i første møtet med arbeidsgiver så fikk vi presentert et API fra vegvesenet som gjør det mulig å beregne bomutgiter og kjørelengde. Webtjenesten [16] deres virker ikke helt optimal, så vi bestemmer oss for å undersøke mulighetene våre ved å gjøre litt research rundt tilsvarende, eksisterende løsninger. Vi tar for oss andre nettbaserte tjenester til å beregne kjøreruter, for å se hvilken tilleggsinformasjon de kunne tilby utover å bare beregne kjøreruta. Vi tester NAF Ruteplanlegger [17] og veibeskrivelsestjenesten hos 1881 [18], Google Maps [19], Gule sider [20] og Kvasir [21]. Felles for alle er at de oppgir kjørelengde for strekningen, samt at de har en mulighet til å legge til via-punkter ved å angi stedsnavn. Google Maps og 1881 tilbyr i tillegg en grafisk løsning for å kunne dra kjøreruten for å endre veivalg. Bompengekostnader er det færre som viser. Kvasir og Gule Sider benytter samme tjeneste, og inkluderer ingen informasjon om bomstasjoner langs ruta. NAF og Google oppgir at det er bomvei på del av strekningen, uten å spesifisere nøyaktig hvor på strekningen denne befinner seg, ei heller prisen. 1881 oppgir prisen for vanlig bil og lastebil, i tillegg navn på bomstasjon og hvor den ligger (grafisk og med koordinater). Bompengekostnader er det færre som viser. Kvasir og Gule Sider benytter samme tjeneste, og inkluderer ingen informasjon om bomstasjoner langs ruta. NAF og Google oppgir at det er bomvei på del av strekningen, uten å spesifisere nøyaktig hvor på strekningen denne befinner seg, ei heller prisen. 1881 oppgir prisen for vanlig bil og lastebil, i tillegg navn på bomstasjon og hvor den ligger (grafisk og med koordinater).

For å finne ut om 1881 eller Ruteplantjenesten egner seg best, tar vi en nærmere sjekk på hvilken informasjon de inneholdt. Vi vet som beskrevet i *kap 2.1.2*, at det finnes både rabattordninger og tilleggspriser rundt omkring i Norge. For å finne ut om disse løsningene 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 totalkostnadene for reisen, benyttes vanlig takst uten rushtidtillegg. I veibeskrivelsen, står det beskrevet at det mellom 07:00 og 09:00, samt 15:00-17:00 er dobbel takst. .

1881 tilbyr en bedre kartløsning som er mer brukervennlig. Kartløsningen hos ruteplantjenesten er uoversiktlig, 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 «Storgata» blir «Storgata, Moss». Brukeren har ingen mulighet til å påvirke dette, utenom å rette opp i ettertid når han eller hun ser at byen er feil.

[16] `http://visveg.vegvesen.no/Visveg/`
[17] `https://www.naf.no/tjenester/ruteplanlegger`
[18] `http://www.1881.no/Kart/Veibeskrivelse/`
[19] `https://www.google.no/maps/dir/`
[20] `http://kart.gulesider.no/veibeskrivelse`
[21] `http://kart.kvasir.no/`

1881 tilbyr autofullførfunksjon, så et søk på «Storgata» gir ut en liste over de forskjellige «Storgata»-er som finnes. Man vil i tillegg raskt oppdage en eventuell skrivefeil i adressen, da adressen man ønsker å skrive inn, ikke lenger vil dukke opp blant forslagene.

1881 sitt system for veibeskrivelser innehar alle nødvendige data vi trenger. Vi konkluderer derfor med at 1881 egner seg best, med tanke på at det inneholder litt mer detaljinformasjon om noen av bomstasjonene, det er mer brukervennlig å søke, 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 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. 1881 opplyser at de har et API, som etter en gratis testperiode, ville koste penger. Ved nærmere undersøkelse viser det seg at dette kun er et API for person- og bedriftssøk, ikke for veibeskrivelser, og uten noe kartbasert innhold. Denne vil derfor ikke dekke de kravene til informasjon prosjektet vårt har. Ved nærmere undersøkelser av JSON-objektet Ruteplantjenesten returnerer, viser det seg at de har tilleggsinformasjon om navn og priser på alle bomstasjonene på ruta, objekter av typen nvdb:Bomstasjon. Hvis det er fem bomstasjoner på den angitte ruten vil JSON filen som returneres inneholde fem bomstasjonobjekter, hvert enkelt objet inneholder navn, takst litenbil, takst storbil. For å kunne liste ut alle bomstasjonene er det bare å søke gjennom JSON objektet etter navnet nvdb:Bomstasjonfor å få listet ut alle bomstasjoner på ruten.

For å oppfylle alle ønskene/kravene til prosjektet er vi avhengig av å bruke Google API til flere deler av vår webapplikasjon. Da Ruteplantjenesten som nevnt ovenfor ikke har en autofullføringsfuksjon, vil vi benytte Google Autocomplete, for å gjøre adressesøk mer brukervennlig. Med dette får vi også tilgang til koordinatene til adressen. Google og Ruteplantjenesten benytter ulike koordinatsyster som gjør at disse ikke kan kobles direkte, mer om hvordan vi håndterer Google Autocomplete og koordinathåndtering finnes i *kap. 4.1.5*. I hovedsak vil resten av applikasjonen basere seg på APIet til ruteplantjenesten. Det hadde 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 et ønske om å kunne vise ruten på kart, slik at brukeren kan kontrollere at den beregnede ruta stemmer. Vi forsøker å kombinere kjøreruteberegning hos Ruteplantjenesten, og vise denne på Google-kart. Et problem som dukker opp når vi kombinerer disse, er at kjøreretning på kartet kan vises forskjellig fra system til system. Det vil si at informasjonen om avstand og bompenger vi får fra Ruteplantjenesten vil stemme med det brukeren skriver inn, men ikke nødvendigvis med kjøreretningen som vises på Google-kartet, selv om begge er satt til å beregne korteste avstand. 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, via E6, 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 at de ikke støtter andre karttjenester kombinert med deres autocomplete-funksjon. [22] Det store spørsmålet er om vi virkelig har behov for en karttjeneste i vår app. Vi skal ikke 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 bekrefte reiseruten som ble beregnet da brukeren gjennomførte sin reise. Kan denne informasjonen vises på andre måter? I stedet for å hente informasjon om valgt reiserute fra Google, kan vi heller vise den i tekstformat fra Ruteplantjenesten. 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 en stiplet linje på et kart, og denne skal da kunne hentes ut fra veibeskrivelsen fra Ruteplantjenesten. Denne tekstlige beskrivelsen vil erstatte det nåværende feltet der kjøreruta beskrives detaljert, slik som beskrevet i *kap 4.1.5*.

2.1.4 Universiell utforming

1.juli 2014 ble nye regler for universell utforming av IKT-løsninger, «Retningslinjer for tilgjengelig webinnhald (WCAG) 2.0» [23]. Regelen innebærer at alle nye IKT-løsninger i Norge, både innen privat og offentlig sektor, organisasjoner og lag, skal være universelt utformet [24]. 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 [25].

Kravene innebærer at IKT-løsninger skal være tilgjengelige for alle, og det tas hensyn både med tanke på brukervennlighet slik at alle skal ha forutsetninger for å bruke løsningen, også de som har nedsatt oppfattelsesevne i form av dårlig syn og hørsel, fargeblindhet osv. 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. Det siste prinsippet, 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

[22] `kildekommer`

[23] `http://uu.difi.no/regelverk/tidsfristar-ny-og-eksisterande-ikt`

[24] `http://uu.difi.no/regelverk/kven-skal-folgje-krava`

[25] `http://uu.difi.no/veiledning/nettsider/krav-til-nettsider/oppbygging-av-wcag-20`

HTML, som vil bruke, blir som regel dette ivaretatt, men det er spesielt viktig å sikre god tilgjengelighet dersom man utvikler egne elemeter (widgets).

Vi har studert retningslinjene, for å vite hvilke hensyn vi må ta under utformingen av prosjektet. 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 de går mer på det overordnede i HRessurs, og applikasjonen vår kun vil utgjøre en veldig liten del iforhold til hele HRessurs. Vi vil rette oss inn etter HRessurs med tanke på fargevalg, tekstfarger osv. En spesifisert liste over hva vi har utelukket, samt hvorfor, finnes i slutten av dette delkapittelet.

Ettersom dette er en løsning der brukeren skal angi informasjon, er alle underpunktene «Retningslinje 3.3 – Inndatahjelp» spesielt viktige. 3.3.1 – Identifikasjon av feil, og 3.3.2 – Forslag ved feil er viktige. Vi må her sørge for at dersom brukeren taster inn en ugyldig adresse, må det bli gitt tydelig tilbakemelding om dette. Her vil vi også bruke en autofullførfunskjon for å foreslå adresser under intasting, for å forhindre feil. 3.3.2 – Ledetekster eller instruksjoner innebærer at alle inndatafelt skal ha ledetekst eller instruksjoner som angir hvilke data som skal angis hvor. Vi vil her tydeliggjøre hvor start, stopp og viapunkter skal angis, at kjøretøy må velges. I denne sammenhengen vil vi også nevne viktigheten av «1.3.2 – Meningsfylt rekkefølge» med tanke på viapunktene som skal angis. Rekkefølgen viapunktene angis i, har betydning for hvilken kjørerute som beregnes. Dette skal vi gjøre ved å poengtere viktigheten av rekkefølgen i form av instruksjonstekster for utfylling, samt gi brukerne mulighet til å sortere listen ved å kunne flytte tekstboksene til ønsket rekkefølge er oppnådd. Denne sorteringsmuligheten vil vi også tekstlig spesifisere. 2.1 – Tilgjengelig med tastatur, tilsier at applikasjonen må være tilgjengelig å betjene fra tastatur, slik som å navigere mellom tekstfelt med tabulator. Dette vil vi sørge for at er mulig å oppnå.

3.3.4 Forhindring av feil (juridiske feil, økonomiske feil, datafeil) er også et kapittel som er veldig viktig. Dette prinsippet går i all hovedsak ut på at dersom brukeren taster inn data som endrer brukerdata, her informasjon om brukerens reiseregning, skal det være en kontroll for at ikke uriktige data overskriver eksisterende. En av følgende tre mulige løsninger kan håndtere dette. Enten ved at prosessen med innsending av data kan reverseres, at brukeren har mulighet til å gå tilbake å rette opp i feil eller at brukeren må bekrefte innsending. 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 ønsker å legge de beregnede verdiene til i sin reiseregning.

Beregningen foregår i en oppdelt struktur, som beskrives nærmere i *kap. 3.1*. Dette innebærer at det er mange steder det kan oppstå feil, som vil gi en feilmelding (exception). Kriterie 3.1.1 – språk på siden, tar for seg at språk på hver side kan bestemmes programmeringsmessig. Her tenker vi en løsning som hvis applikasjonen gir en feilmelding, så vil denne slå opp på i koden for denne, og presentere en feilmelding på språket som valgt. I

utgangspunktet skal feilmeldingene som er relevant for brukerne presenteres på norsk, da HRessurs er lagt opp til administrering av norske firmaer. Det kan tenkes at applikasjonen en gang i framtiden skal oversettes til f.eks svensk, og da kan man enkelt endre dette i koden, og slippe å oversette alle mulige feilmeldinger manuelt. Kravene i 4.1 – Kompatibel, i forhold til kompabilitet med andre brukeragenter og tilsvarende teknologi vil være viktig å tenke på ved bruk av Ruteplantjenesten og Google API, implementering i forhold til nåværende HRessurs, og at applikasjonen skal være enkel å tilpasse til nye HRessurs.

Vi velger å gå bort i fra følgende retningslinjer: Retningslinjer 1.1 – Teskstalternativer, 1.2 – tidsbaserte medier og 2.3 – Anfall, er alle kapitler som er mer aktuelle i en mulitmedieaplikasjon, eller en applikasjon der grafikken spiller en stor del av opplevelsen. Retningslinje 1.4 – Mulig å skille fra hverandre, 2.2 – nok tid, 2.4 – navigerbar, 3.2 – Forutsigbar, går på blant annet navigering på websidene, fargevalg og kontraster, at brukerene 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 oppfyllt eller ikke, vil vi ikke studere nærmere. HRessurs skal som vi vil gå nærmere inn på i *kap. 2.2.2*, relanseres i ny drakt, siden nåværende HRessurs ikke er en ny webløsning etter 1.juli 2014, er den bare omfattet at regelverket for eksisterende løsninger, slik at den må følge reglene innen 2021. Når den relanseres trer regelverket i kraft, og da vil HRessurs omfattes av reglene. Vi vil ha retningslinjene i bakhodet dersom vi legger til nye funksjoner, men planen er at vår tilleggsapplikasjon skal ha et så rent brukergrensesnitt som mulig, og skal kunne skli rett inn i eksisterende HRessurs.

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 da måtte oppdragsgiver i så fall tilpasse den til sitt allerede eksisterende system. 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 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 dagens ordning for registrering av reise og utlegg er det naturlig at vi velger samme plattform som HRessurs allerede er basert på.

Men innenfor .Net rammeverket er det ulike måter å strukturere en webapplikasjon 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 brukerens 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 jobber med 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 nås 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.

Kapittel 3

Design/utforming/planlegging

3.1 Struktur/innhold/utforming

HER KOMMER DET MER / IKKE FULLFØRT KAP

Hvordan vi vil dele opp prosjektet (Eks HTML/JS/CSS – WEB SERVICE (C#) – INTERFACE – API... . Bør vel også være litt om hvorfor vi deler opp (vedlikehold, med tanke på utbytting av API...).

At vi ikke skal bruke asp.net og hvorfor?

3.2 Uferdig tittel/kapitel - mal/leveranse

Beskrivelse av hva de forskjellige leveransene skal inneholde.

HER KOMMER DET MER / IKKE FULLFØRT

Kapittel 4

Implementasjon/produksjon/gjennomføring

4.1 Alt i ett løsning

Vår første versjon baserer seg kun på HTML og javascript. Den bruker også 2 sepparerte API systemer. for den grafiske biten samt kordinater bruker vi Google Map API, og for all nyttig kjøredata håndteres det av Ruteplan tjenesten som er eid av Statens Vegvesen, vi er i kontakt med vegvesenet for å få opprettet en access til det lukkete API'et og får fri tilgang til det iløpet av hele Bachelor perioden.

Om løsningen er varig i etterkant med tanke på om vegvesenet lar private bedrifter bruke løsningen dere er ikke undersøkt. men vi jobber med saken.

Når man taster in start stop addresse bruker vi google sitt API til og auto fullføre de forskjellige søkene man sender inn i form av adresser. Når auto fullføring av søk er ferdig og man trykker beregn vil en funksjon kjøre som konverterer adressen til søket om til kordinater. Deretter konverteres kordinatene om til ett format som Ruteplantjenesten forstår, I det kordinatene er ferdig konvertert blir de byggdt inn i en URL string som vi sender til Ruteplan sitt REST API, hvor vi får returnert et JSON objekt med forskjellige type informasjon.

I det JSON objektet henter vi så ut total kjørelenge, beregnet tid samt navn på bomstasjoner og takstert for passeringer. Det blir så igjen returnert til HTML siden som skiserer ruten grafisk på ett kart og returnerer de spesifikke dataene.

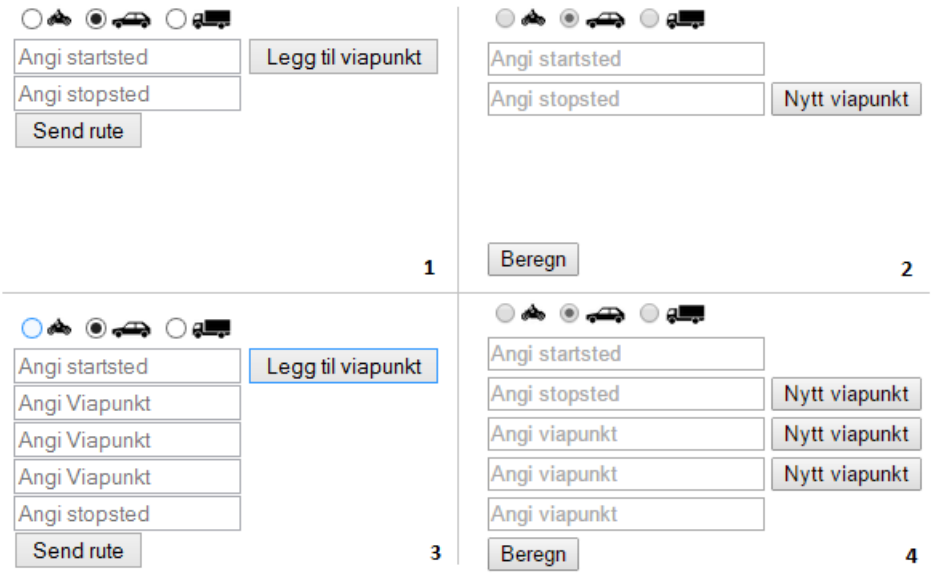
4.1.1 Utforming av viapunktliste

For at brukeren skal kunne legge inn en så nøyaktig reiserute som mulig, har vi valgt å legge til muligheten for å legge til via-punkter. Dersom en destinasjon ikke har en selvsagt rute, men kan nås ved å velge flere forskjellige strekninger, eller hvis brukeren har blitt tvunget til å kjøre en omvei, kan det være nødvendig å beskrive 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 teksfeltene 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 et 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. Alt dette kan løses på en bedre måte med Javascript-tellere som kan registrere antall trykk og dermed fjerne knappen etter for eksempel ett trykk. Vi har allerede 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 Jacascript 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, xxxxx. Forskjellen er at førstnevnte CSS setter felter som allerede eksisterer til synlig, mens den andre metoden lager ett nytt felt for hver gang som knappen trykkes.le Skjermbildene i *Figur.4.1* på neste side, illustrerer forskjellen på usynlige tekstfelt, og tekstfelt i liste.

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



Figur 4.1: Viapunkter, CSS og Javascript

Brukeren skal også kunne sortere via-punktene, 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 slette teksten i feltene, og i stedet ta tak i ett av feltene og 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 direkte, vi må gå en omvei. Dersom feltene legges inn som listeelementer og knyttes sammen med et bilde (i vårt tilfelle en hånd som viser at man kan manipulere feltene, se *figur.4.2*) 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.

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

Figur 4.2: Viapunkter til å manipulere

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 Autocompleete, ikke er i samme format som Ruteplantjenesten krever. Se mer om dette i *kap. 4.1.5* Pga utfordringene i forbindelse med converteren, og det at tekstfeltene ikke finnes fra begynnelsen, tester vi et annet alternativ for viapunktliste, som skal fungere bedre med tanke på Google autocompleete og konverteringen.
fortsettelse, må skrives..eks: Når vi skal knytte viapunktlista i javascript mot converteren og autocomplete dukker følgende problemer opp... dette kan man løse ved å gjøre xxxxxx, men da dukker følgende problemer opp som gjør at xxxxx (feks at dobbeltklikk for å slette, må fjernes). Dette kan man løse ved å bruke CSS-design i ren HTML sammen med Javascript til å sette et helt div-element med fem viafelt til display none". Deretter knytter man en knapp til en Javascript funksjon som setter elementet til display block". Dette vil gjøre feltet synlig. Trykkes knappen igjen vil elementet settes tilbake til display none. 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.

4.1.2 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 4.3: 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.4.3*. Når brukeren trykker knappen «Send rute», sendes verdiparet med intastet data til webservice. I Web service

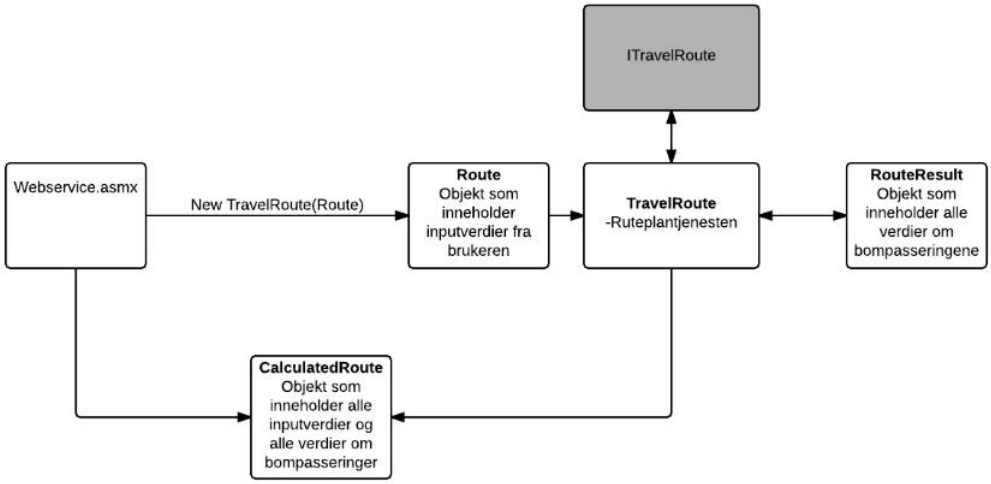
inisialiseres et nytt objekt av CalculatedRoute basert på inputverdiene og resultatet av den beregnede ruta. Dette sendes så tilbake til klientsiden, der vi parser (pakker ut) objektet, og fremstiller relevant informasjon for brukeren. Denne første løsning inneholder bare testdata, slik at bomstasjonene 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.

4.1.3 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. 4.4*

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



Figur 4.4: Strukturen fra Web Service til Ruteplantjenesten

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 4.5: 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 HttpWebRequest 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 tekslig 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.

4.1.4 Veien mot endelig brukergrensesnitt

Vi ser på muligheter for å presentere resultatene for brukerne. Ved å sette opp ett enkelt listesystem for utdyping av de forskjellige bompasseringene som blir foretatt under reisen har brukeren muligheten til å klikke seg inn på hver enkelt bomstasjon for å se de forskjellige takstene som er listet under hver enkelt "stasjon", listen blir generert av Javascript og skriver rett til HTML via det returnerte JSON objektet.

Skisse kommer

4.1.5 Google og konvertering

Google sin autocompleete er lagt til i tekstfeltene der adresser skal angis, 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 ERST89. For å kunne bruke koordinatene i Ruteplantjenesten, må koordinatene fra Google konverteres slik at vi kan bruke koordinatene til ruteberegning hos Rutenplantjenesten.//

For å oppfylle alle ønskene/kravene til prosjektet har vi har vært avhengig av å bruke Google API til flere deler av vår webapplikasjon. Enn så lenge vi vil benytte Googles autocomplete-funksjon, for å gjøre adressesøk mer brukervennlig. Google sin autocompleete er lagt til i tekstfeltene der adresser skal angis, 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. For å kunne benytte Google autocomplete 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. I hovedsak vil applikasjonen basere seg på APIet til ruteplantjenesten.

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

Figur 4.6: Viser eksempel på hvordan man henter Google API Autocomplete basert på steder.

Ruteplantjenesten foretar søk baser på koordinater, og benytter den europeiske standarden ETRS89 som måles i meter. Google derimot er basert på det amerikanske WGS84 som bruker grader. Vi må derfor ha en konverter som konverterer mellom de to forskjellige koordinatsystemene WGS84 og ETRS89. Konverteren vil få 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. 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.

4.1.6 Skrives senere:

Videre arbeidsforløp

Veien mot endelig GUI HTML-Webservice Versjon 3 osv

4.1.7 programmer vi har brukt:

- Visual Studio 2013
- Visuam Paradigm Free UML design tool
- Surveymonkey.com

4.2 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.

4.3 SCRUM - utførelse

Vårt produkt er utviklet på en standard som er satt av Infotjenester for utvikling av produkter, vi har benyttet en flerlags model innen programmering som generelt sett betyr at vi skal ha det mest mulig vedlikeholdbart samt at det skal være enkelt og bytte ut deler Vi utviklet en versjon 1 som vi benyttet til å få en bedre oversikt over hva eventuelle brukere føler de trenger i en slik modul, denne versjonen benyttet vi til å utarbeide en spørreundersøkelse som vi delte ut til et bredt spekter av personer, ut ifra alder og IT kunnskaper. Dette ga oss mye feedback på hvordan vi kan få produktet til et nytt nivå innen brukervennlighet. Dette tok vi med oss videre når vi ferdigstilte produktet og disse endringene ble gjort (. . .) (. . .) må utfylles når vi er så langt.

skal skrives om, Litt om foreløpig scrumresultater. litt om Backlog /userstories/-milestone og at vi definerer på git. Gjerne screenshot fra første milestone. Begrunnelse for hvorfor vi ikke har hatt klart deifnerte sprinter fra begynnelsen (pga å lære oss nye ting, analyse muligehter, finne ut om prosjektet var gjennomførbart osv)

Kapittel 5

Test av produkt

5.1 Testing

Vi kvalitets sikrer produktet vårt ved at vi hele tiden tester programmet, lar andre personer teste produktet og komme med tilbakemeldinger, samt at arbeidsgiver tester produktet på SCRUM Møtene.

5.2 Evaluering

I forbindelse med første utkast av programmet vårt velger vi å gjennomføre en undersøkelse blant et knipe forskjellige personer med forskjellig alder og kunnskap når det gjelder IT. Vi gjør dette for å få en bredere oversikt over hva som bør være med og hva som burde vært gjort annerledes. Vi velger å bruke en så bred brukergruppe som mulig slik at vi får mest mulig bredt spekter av meninger.

Temaer som vi skal ha med i undersøkelse **Spørsmålene vil endres når evalueringen er gjennomført**

1. Navn
2. Aldersgruppe
3. IT kunnskap
4. Brukervennlighet?
5. Mangler?
6. Hva var bra?
7. Er det noen funksjoner du syns burde være med?
8. Stedsforslag når man taster inn fra/Til, syns du dette bør være med?
9. Når ruten er tastet inn foretrekker du en grafisk fremstilling av ruten, med bruk av kart, eller foretrekker du en skriftlig rutebetegnelse?

Kapittel 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?

Kapittel 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.