Technical University of Denmark

Erhvervsorienteret rutevejleder

Course:
Ingeniørprojekt

Mathias Nikolaj Nielsen
s134542

Forår 2018

Abstract

Atkins is currently developing Vejdirektoratets (VD) traffic map (https://trafikkort.vejdirektoratet.dk). This displays, among other things, real time information concerning traffic load, incidents and various winter services. As an extension to this traffic map, VD wants to expand the business orientered functionalities aimed at bus and truck drivers, and road trains.

This project is based on the current implementation and analyzes difficulties withholding the extension up until now. Current data sources and data model are analyzed in order to determine any improvements. Alternatives to the requested extension are explored along with the opportunities this can support.

The extension is developed as a prototype routeplanner. The prototype takes VD's data on ground clearance into account, combined with an available API for route planning, e.g. the Google API, which is currently used on the site, or an alternative open source product. Further some user interface and interactions are taken into account.

Based on the technology stack currently used in the system, skills gained through various courses is expected to be used, such as:

- 02267 - Webservice elective
- 02105 - Algorithms and data structures
- Various courses on Java and JavaScript programming.

Resumé

Atkins udvikler pt Vejdirektoratets (VD) trafikkort (https://trafikkort.vejdirektoratet.dk). Her vises blandt andet realtidsoplysninger vedr. trafikbelastning, trafikhændelser og vintertjenester. I forbindelse med dette trafikkort ønsker VD at udvide deres funktionalitet der fokuserer på erhvervschauffører, så som bus- og lastbilchauffører og modulvogntog.

Dette projekt tager udgangspunkt i den nuværende løsning og analyserer problemstillinger der har tilbageholdt implementationen af den ønskede funktionalitet. I forbindelse med analysen bliver der fokuseret på de nuværende datakilder og datamodel, for at undersøge eventuelle forbedringer. Hvilke alternativer der er til den ønskede løsning, og hvilke fremtidige udviklingsmuligheder dette kan være med til at understøtte.

Som en udvidelse til trafikkortet er der udviklet en prototype til en ruteplanlægger der tager hensyn til frihøjder på baggrund af vejdirektoratets data, kombineret med et ruteplanlægger API, fx. Googles, hvilket bliver brugt på trafikkortet pt. eller alternative open source produkter, og endelig hvordan det præsenteres for brugeren.

På baggrund af den teknologi-stack som trafikkortet er udviklet på, forventer jeg at bruge kompetencer opnået igennem:

- 02267 - Webservice valgfag
- 02105 - Algoritmer og datastrukturer
- Forskellige kurser om Java og JavaScript programmering.

Indhold

1 Indledning og Problemformulering

Denne rapport fokuserer på udviklingen af en prototype udvidelse til et eksisterende produkt (https://trafikkort.vejdirektoratet.dk) udviklet af Atkins Danmark A/S til Vejdirektoratet (VD). Trafikkortet er en AngularJS baseret webapp som blandt andet viser realtids trafik information og vinter relateret information.
Hovedfunktionaliteten fokuserer på private brugere af systemet med almindelige personbiler, og en dertilhørende rutevejledning baseret på Google's API.

Som en udvidelse til dette trafikkort ønsker VD ny funktionalitet som fokuserer på erhvervschauffører. Trafikkortet udstiller pt. data omkring frihøjder, vægtgrænser, tvangsruter for farligt gods og tilladte veje for modulvogntog. Funktionaliteten til planlægning af ruter til erhvervschauffører, på baggrund af udstillet data, mangler.

På baggrund af den ønskede funktionalitet og i samarbejde med Atkins er følgende problemstillinger opstillet.

- Hvordan data forbedres til at give et præcist grundlag for en rute?
- Hvordan en rutevejleder implementeres til samarbejde med VDs data om frihøjder og vægtgrænser og samtidigt sikre den alternative rute er valid?
  - Hvilket Framework kan bruges til at understøtte dette?
- Hvordan udstilles servicen optimalt til både indlandske og udenlandske brugere?

Raportens struktur læner sig opad normen indenfor OOAD[1], med et indledende analyse afsnit der fokuserer på analysen af de eksisterende datakilder, use cases af prototypen og dets funktionalitet.
Et design afsnit der følger op på use cases og hvordan valget af frameworks har påvirket designet, samt hvilke overvejelser der er gjort for at understøtte udvidelse af prototypen.
Implementerings afsnittet vil hovedsagligt understøttes af code snippets, og fokuserer på hvordan designet og funktionaliteterne er realiseret i koden og hvordan designet er blevet påvirket af det eksisterende kode.
Test afsnittet vil argumentere for at prototypen virker og følge op på use cases med dertilhørende test cases, samt konkludere på opfyldelsen af kravspecifikation.
Til sidst kommer der en perspektivering til hvordan prototypen kan udvides i fremtiden. Hvordan bedre erhvervsorienteret ruteplanlægning påvirker samfundet og erhvervslivet og hvordan dette aligner sig med Atkins strategi.

[1] **O**bjekt **O**rienteret **A**nalyse og **D**esign

2 Projektplan

Dette afsnit vil omhandle projektstyringen af projektet og hvilke beslutninger der er taget forudgående for projektet.

2.1 Tidsplan

Indledningsvis til projektet er der udarbejdet et Gantt chart, se figur 1, som forventet tidsplan for projektet.

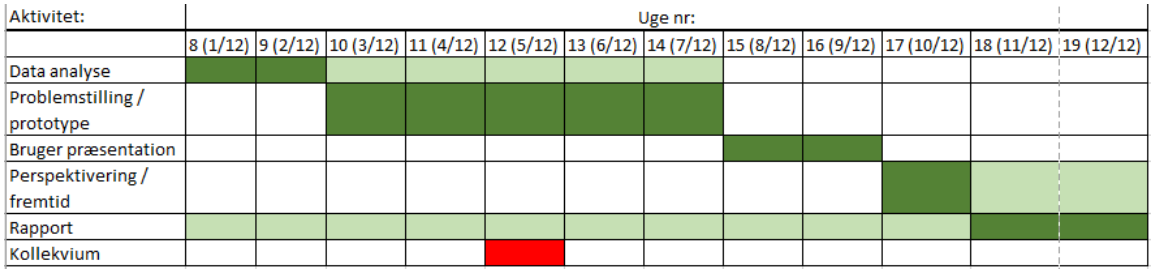

Figure 1: Gantt chart - Forventet tidsplan

Gantt forklaring

Venstre side lister alle forventede aktiviteter på baggrund af problemstilling defineret i section 1.

Farve forklaring

- Mørkegrøn - Fokus område i denne periode.
- Lysegrøn - Sideløbende opdatering i denne periode.
- Rød - Obligatorisk tidspunkt for Kollekvium.

Med prototypen som kernefunktionalitet i dette projekt er der henstillet 5 uger til fokus på udvikling af dette, hvis dette ikke er tilstrækkeligt kan der skæres på tiden til Bruger præsentationen, da dette tilhører bonus funktionalitet.

2.2 Projekt management

Som projektstyring til dette 1-mands projekt, har jeg valgt at benytte mig af KanBan, kombineret med et gratis online værktøj kaldet Asana[1], og har her oprettet to projekter, ét til prototype udviklingen og ét til rapport skrivningen.
Formålet med disse projekter er at holde styr på de TODOs og opgaver jeg skal huske, og derved nemmere kan overskue opgaverne, og de deadlines der er sat op ifølge Gantt diagrammet på figur 1. Workflowet for disse to projekter er næsten ens, og er beskrevet på figur 2.

Prototype projekt

1. TO DO
2. IN PROGRESS
3. TEST
4. DONE

Rapport projekt

1. TO DO
2. IN PROGRESS
3. DONE
4. PROOF READ

Figure 2: Projekt workflows

Alternativt til Asana findes der mange andre online projekt management systemer, fra tidligere projekter igennem praktikperioden og kurser på DTU, er der arbejdet med JIRA[2]. Asana er blevet valgt da det er gratis og kendskab til værktøjet er opnået igennem fritidsprojekter.

2.3 Udviklingsmetode

Som en del af overtagelsen af projektet fra Atkins, blev der lavet en ny branch under Atkins VCS[16], med en fuld kopi af projektet, dette gjorde det muligt at fortsætte i samme udviklingsstil som de originale udviklere fra Atkins.
Projektet indeholdte de nødvendige biblioteker, samt et python script der starter en udgave af trafikkortet op på en localhost webserver. Dette lagde grundlag for den løbende udvikling og test, ved redigering og implementering af de ønskede klasser og filer.
For at ændringerne trådte i kraft skulle browseren dumpe cached data og refreshe web applikation (CTRL + F5) og kombineret med Google Chrome Developer Tools[3] var det muligt at se resultatet og evt. console log i Chrome.
Under udviklingen af prototypen blev der brugt sideløbende user tests der tester det specifike komponent.

3 Analyse

Dette afsnit omhandler den indledende analyse af de nuværende datakilder og analyse af prototypen i form af use cases og kravspecifikation. Yderligerer undersøges vejdirektoratets ønsker til prototypen, og deres synspunkt på problemstillinger der har tilbageholdt denne funktionalitet til dato.

3.1 Nuværende datakilder

Som en del af det eksisterende trafikkort, er der udviklet et større integrations backend system som indhenter data fra de nødvendige management systemer i Vejdirektoratet. Denne integrations service indlæser data og transformerer det til GeoJSON[10] format og uploader det så herefter til en Google cloud storage service, som trafikkortet har adgang til.
Der er primært kigget på datakilder der pårører vægtgrænser og frihøjder, disse datakilder kommer ind i .XML format i en lang liste af elementer. Se figur 3 for frihøjde eksempel.

```
1  <gml: featureMember>
2    <TRA: bygvaerker_med_frihoejdefid=
3    "bygvaerker_med_frihoejde.fid--6100aeaf_161ad00e681_2466">
4      <TRA: BYGVAERK_ID>28723751</TRA: BYGVAERK_ID>
5      <TRA: BYGVAERKSART>Vejbrendebro</TRA: BYGVAERKSART>
6      <TRA: BYGVAERKSBETEGN>OFafHndbkstien</TRA: BYGVAERKSBETEGN>
7      <TRA: BYGREGNR>18232.0</TRA: BYGREGNR>
8      <TRA: FRIHOEJDE>4.4</TRA: FRIHOEJDE>
9      <TRA: GEOMETRI>
10       <gml: PointsrsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
11         <gml: coordinatesxmlns: gml="http://www.opengis.net/gml"decimal="."cs=","ts="
               ">10.5, 57.4312</gml: coordinates>
12       </gml: Point>
13     </TRA: GEOMETRI>
14   </TRA: bygvaerker_med_frihoejde>
15 </gml: featureMember>
```

Figure 3: Frihøjder - xml element

Under transformationen til GeoJSON format behandler den blandt andet GEOMETRI subobjektet, så geometri information snappes til Google maps opfattelse af vejens placering, dette skyldes at trafikkortet bruger Googles Rute API. Se figur 4 for resultatet af transformationen, for yderligere eksempel med vægtgrænse se bilag B.1.

```
{
   "type": "Feature",
   "properties": {
     "featureId": "28723751",
     "layerId": "layer_id_9A",
     "title": "OF af  Hndbkstien ",
     "regNo": "18232.0",
     "kind": "Vejbrende bro",
     "clearance": "4.4"
   },
   "geometry": {
     "type": "Point",
     "coordinates": [10.5,
     57.4312]
   }
 }
```

Figure 4: Frihøjder - xml element

3.2 Vejdirektoratets perspektiv

For at få vejdirektoratets (VD) perspektiv på problemstillingen blev der afholdt et møde med Thomas Mark De Laine[12], som agerer kontaktperson hos VD. Disse spørgsmål lagde grundlag for mødet:

1. Hvorfor er denne funktionalitet ikke blevet implementeret endnu?
2. Hvad skyldes denne usikkerhed på frihøjderne?
3. Hvor kommer alt jeres data fra, og hvad skyldes variationen i datatyperne?
4. Eksisterer der funktionalitet til deling af jeres data til udenlandske chauffører?
5. Mere info om jeres Trafikinfo App, hvem udvikler denne?

VDs ønsker til problemstillingen afviger en anelse fra den originale problemstilling. Konklusionen fra mødet med VD kombineret med den originale problemstilling kan deles op i tre kategorier:

1. Data
2. Funktionalitet
3. Præsentation

Data

På baggrund af spørgsmål 2 og 3, blev der fremlagt ny viden i form af introduktion til den interne database DANBRO[17] hos VD, indeholdende information om broer. Som tidligere beskrevet i afsnit 3.1, er DANBRO en af de management systemer som integrations servicen henter information fra. DANBRO databasen indeholder langt mere data end det der trækkes ud til trafikkortet. Det simple udtræk til trafikkortet indeholder kun den laveste højde på broen, hvor højderne i realiteten kan variere fra hvilken retning og hvilket spor der køres i. Som sikkerhedsmargin fratrækkes der 20cm fra den reelle frihøjde.

Funktionalitet

På baggrund af spørgsmål 1, 4 og 5, konkluderes det at den ønskede funktionalitet ikke er implementeret grundet udfordringer med hvordan den ekstra data i DANBRO skal udstilles på en brugervenlig måde igennem trafikkortet.
Funktionaliteten udstilles pt. ikke til udenlandske brugere, i tilfælde hvor udenlandske interessenter mangler information om frihøjder i forbindelse med en transport, allierer de sig med danske transport firmaer. I forlængelse af dette spørgsmål kom der information om denne branche, se bilag B.2 for uddybning.
På baggrund af tidligere erfaring hos VD ville evt. implementering af prototypen udviklet i dette projekt, blive implementeret samtidigt, både som en del af det online trafikkort og som en app, for at undgå fejl ved variation mellem produkterne.

Præsentation

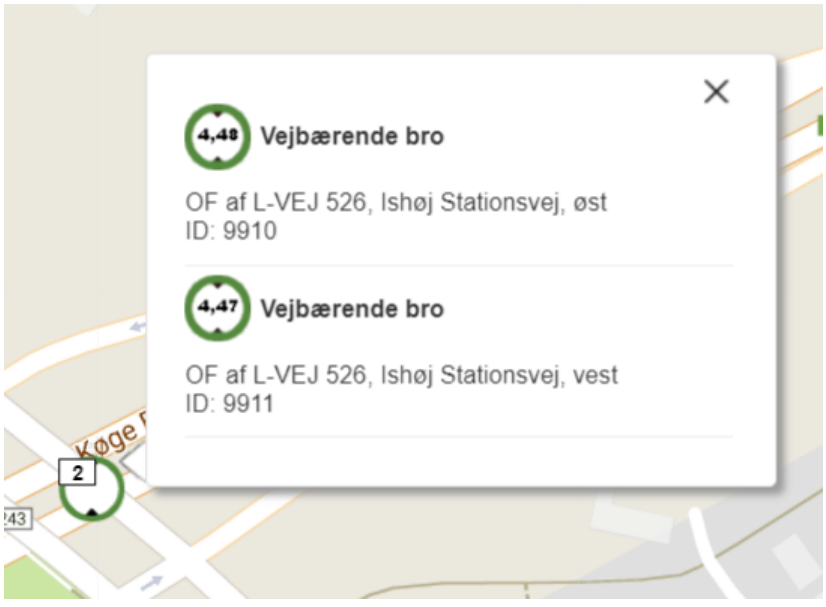

Figure 5: Udsnit af trafikkortet

I forlængelse af **Funktionalitet** afsnittet kan det ses på figur 5, hvordan frihøjde data udstilles på trafikkortet i dag. I eksemplet overlapper 2 frihøjde-skilte og derved grupperes de. Yderligere konkluderes det at beskrivelserne til frihøjderne kræver en vis grad af teknisk og intern forståelse.
Målgruppen for den erhvervsorienteret del af trafikkortet varierer og kræver derfor at implementationens formulering og præsentation af data kan forstås af alle interessanter, herunder vognmænd, broingeniører og mindre tekniske interessanter som buschauffører.

3.3 Funktionalitets konklusion

På baggrund af afsnit 3.1 og 3.2 opsummeres og redefineres de funktionaliteter der fokuseres på i dette projekt. For at visualisere det er der udarbejdet et diagram (Figur 6).

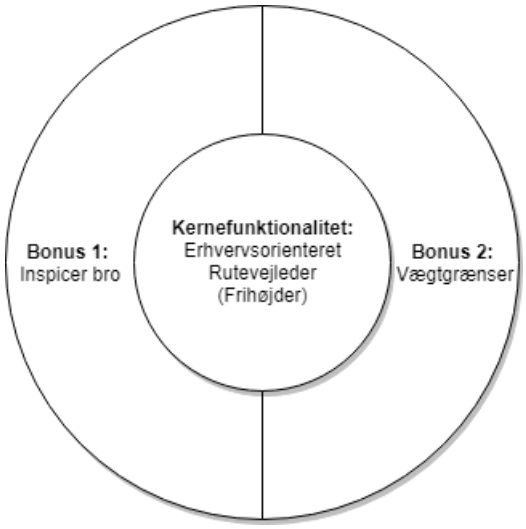

Figure 6: Funktionalitets oversigt

- **'Kernefunktionalitet'**: Er udledt af den indledende problemformulering og er indskrænket til at prototypen skal samarbejde med frihøjde data, og så vidt muligt returnere en rute på baggrund af dette.
- **'Bonus 1: inspicer bro':** Grundet VDs perspektiv på problemstillingen er der udledt en ekstra funktionalitets komponent.
- **'Bonus 2: Vægtgrænser':** Prototype interaktion med vægtgrænser er blevet nedprioriteret, da dette ligner kernefunktionaliteten og derfor er VDs ønsker blevet prioriteret højere.

3.4 Use cases

For denne prototype er der udarbejdet to use cases på baggrund af den indledende problemformulering og konklusionen fra afsnit 3.2 og 3.3.

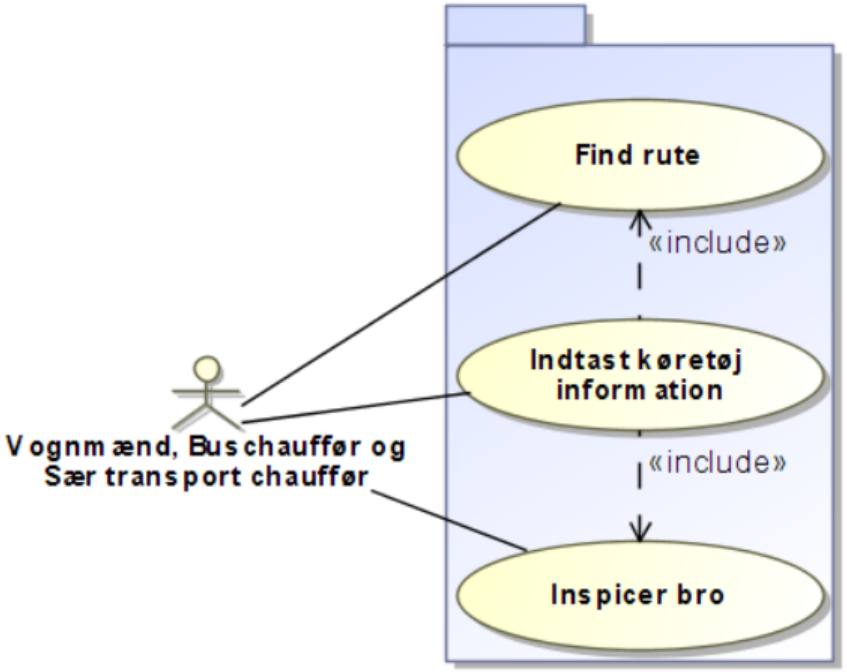

Figure 7: Use case diagram

På use case diagrammet ses en tredje use case "indtast køretøjsinformation", dette er eksisterende funktionalitet som er nødvendig for udførelsen af de resterende use cases.

3.4.1 Use case beskrivelse - Find rute

Introduktion
Use case beskrivelse for at finde en rute.

1. **Aktører**
   (a) Vognmænd (lastbil chauffører)
   (b) Buschauffører
   (c) Sær transport
      i. Tvangsruter for farligt godt
      ii. Tilladte veje for modulvogntog
2. **Preconditions**
   - Aktøren har information om den ønskede rute.
   - Aktøren har information om sit køretøj.

3. **Hovedforløb**
   (a) Aktøren åbner https://trafikkort.vejdirektoratet.dk.
   (b) Aktøren indtaster køretøjsinformation. [**Alt 1:** Brugeren glemmer at indtaste køretøjsinformation - rute validering punkt (d) fejler.]
   (c) Aktøren indtaster afgangs- og destinationsadresse.
   (d) Aktøren klikker på 'Valider rute'. [**Alt 2:** Brugeren klikker 'fjern rute'.]
4. **Postconditions**
   - Aktøren er givet en valid rute. [**Alt 3:** Ingen valid rute kunne findes.]
5. **Alternative forløb**
   - **Alt 1:** Ved validering leverer systemet en fejlmeddelse.
   - **Alt 2:** Afgangs- og destinationsadresse og rute nulstilles.
   - **Alt 3:** Aktøren får en fejlmeddelse.

3.4.2 Use case beksrivelse - Inspicere bro

Introduktion
Use case beskrivelse for at inspicere en bro og få mere information. Denne use case fokuserer særligt på frihøjder.

1. **Aktører**
   (a) Vognmænd (lastbil chauffører)
   (b) Buschauffører
   (c) Sær transport
      i. Tvangsruter for farligt godt
      ii. Tilladte veje for modulvogntog
2. **Preconditions**
   - Aktøren har adgang til trafikkortet.
3. **Hovedforløb**
   (a) Aktøren åbner https://trafikkort.vejdirektoratet.dk.
   (b) Aktøren vælger "Frihøjder" i drop down menuen og klikker 'Vis'.
   (c) Aktøren finder området der er interesse for.
   (d) Aktøren klikker på ikonet/broen.
4. **Postconditions**
   - Aktøren får vist en mere beskrivende popup om den specifikke bro.

3.5 Kravspecifikation

I forlængelse af afsnit 3.3 og 3.4 kan der opstilles en række krav til prototypen.

Kernefunktionalitet

1. Mulighed for at indtaste køretøjsinformation (frihøjde).
2. Mulighed for at indtaste afgangs- og destinationsadresse.
3. Systemet skal levere en valid rute på baggrund af foregående information.

Bonus 1

1. Den ekstra information skal præsenteres på en brugervenlig måde.
2. Den ekstra information skal interagere med ruten.
3. Systemet skal indhente yderligere information fra DANBRO.

Bonus 2

1. Mulighed for at indtaste køretøjsinformation (vægtgrænser).
2. Mulighed for at indtaste afgangs- og destinationsadresse.
3. Systemet skal levere en valid rute på baggrund af foregående information.

3.6 Delkonklusion

Afsnittet har analyseret den nuværende situation, hvordan kunden ønsker produktet skal udvikles i fremtiden og hvilke problemstillinger der har tilbageholdt funktionaliteten frem til dato. Denne analyse har lagt grundlag for en funktionalitets opdeling, med dertilhørende use cases og en kravspecifikation, bygget på den indledende problemstilling beskrevet i samarbejde med Atkins og på baggrund af interview med Thomas fra VD.

Tidsplanen i afsnit 2.1 er bygget på den indledende problemstilling og sætter 2 uger af til bruger præsentation, på baggrund af VD's ønsker konkluderes det at der som minimum skal udarbejdes et udkast til en løsning på denne funktionalitet, beskrevet som bonus 1 i afsnit 3.3.

4 Design

Dette afsnit vil omhandle de design beslutninger der er taget inden udviklingen, med udgangspunkt i analysen. Yderligere vil de relevante frameworks blive beskrevet og deres eventuelle påvirkning på designfasen.

Forud for udviklingen af prototypen var den eksisterende kodebase ukendt, dette har givet udslag undervejs i udviklingen og har påvirket det ønskede design. De aktuelle problemstillinger opstået undervejs vil blive belyst i dette afsnit og afsnit 5.

4.1 Frameworks

Dette afsnit vil belyse de beslutninger der er taget omkring valg af frameworks. Derudover vil teorien bag disse frameworks blive diskuteret og hvordan det har påvirket designet af prototypen.

4.1.1 AngularJS

Det eksisterende system er todelt, med en backend implementation i Java der håndterer forædling af data, frasortering af irrelevant information, samt forretningslogik angående abonnementer og adviseringer. Forretningslogikken implementeret i frontend systemet, er skrevet i AngularJS og indeholder blandt andet logikken vedrørende visualisering og håndtering af brugscenarier. Dette ændrede forudsætningerne for prototypen, da først antaget at forretningslogikken omkring ruteplanlægning var implementeret i Java backenden, hvilket resulterede i et øjeblikkeligt behov for at lære AngularJS og hvordan dette designes og udvikles ifølge best practices. Ressourcerne hertil indebærer forskellige guides, de vigtigste elementer beskrives i dette afsnit. *Se appendix A.1 for listen over ressourcer.*

AngularJS er et frontend framework udviklet af Google, AngularJS fokuserer på at effektivisere hjemmesider, ved at gøre det nemmere at genbruge kode og optimere backend kald ved at begrænse mængden af data der hentes, ved dynamisk ændring af html siderne. Samtidigt giver AngularJS en struktur der gør det nemt at teste. *Se AngularJS dokumentation for en mere detaljeret beskrivelse af AngularJS [6].*

En stor del af design beslutningerne er taget på baggrund af en Angular Team anerkendt community style guide [A.1-pkt.6], hvor standard opbygningen af et AngularJS program bygger på 3-lags modellen og MVVM arkitektur. Dette kommer til udtryk igennem nogle af de kernemoduler givet fra AngularJS biblioteket:

1. Directives
2. Controllers
3. Factories

4. Services

Sammenhængen af disse moduler og fordeling af ansvar kan ses på figur 8.

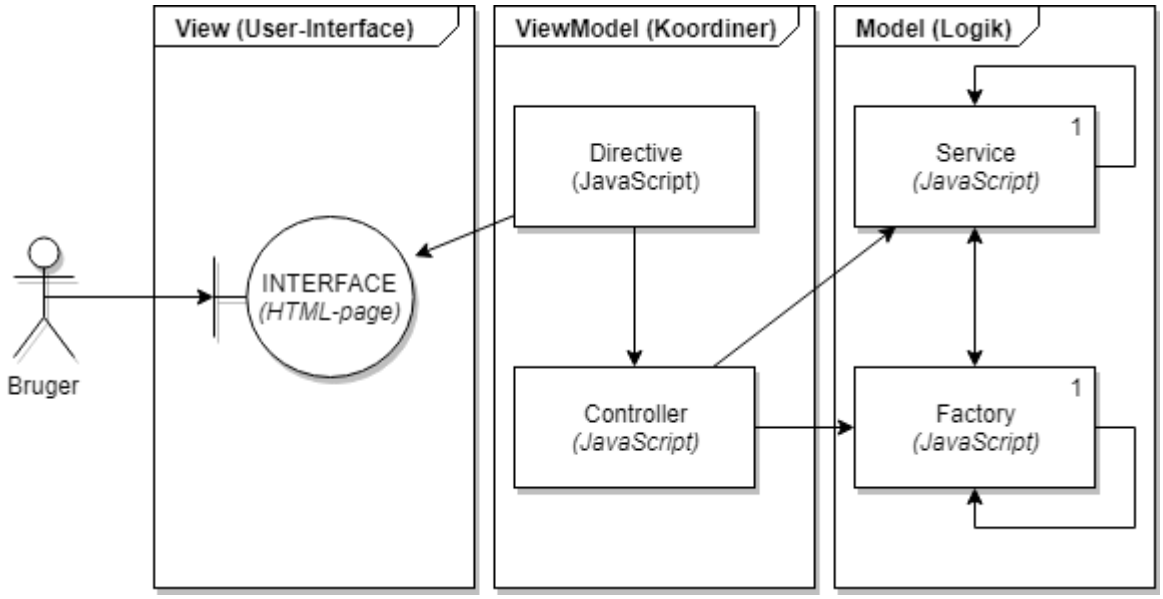

Figure 8: AngularJS kerne moduler

Som det ses på figur 8 er det tydeligt opdelt. User interfacet forbindes med en Controller som har den koordinerende rolle, igennem Directives, hvor disse custom Directives compiles til enten HTML elementer eller HTML attributes, som kan bruges og genbruges på andre HTML sider.
Model laget indeholder alt logik, og baseret på style guiden skal Factories indeholde forretningslogik, og alt data manipulation som fx. eksterne databasekald skal laves til Services, også kaldet DataServices.

Arkitekturen som ses på figur 8, vil være en gennemgående struktur for hver funktionalitet i AngularJS projektet, denne struktur vil derfor udgøre grundlag for klassediagrammet designet til prototypen.
Den overordnede arkitektur for et komplet AngularJS projekt, bygger på en stjerne topology hvor hver node er en funktionalitet med strukturen set i figur 8, som samles af AngularJS klassen **Module**.

Ved genbrug af funktionalitet på tværs af noderne kan Directives genbruges direkte på HTML siderne, eller logik-komponenter som Factories og Services kan ved hjælp af **Dependency Injection** [2] genbruges i andre dele af projektet og i andre Factories og Services.

[2]Dette kan betragtes som "import" af en klasse i Java

Alternativ

Alternativet til AngularJS ville omfatte en ændring af projekt scopet, da næsten hele den eksisterende kodebase er skrevet i AngularJS.
Denne scope ændring kunne resultere i en redifinition af projektet der tillader et andet framework, fx. ved at lave det som en ny backend service, dette ville dog stadig kræve en form for integration til Trafikkortet og dets AngularJS kode.

Hvis der ses bort fra dette projekt scope og blot ser på alternativerne til Angular(JS) på markedet, findes der en række frameworks, som leverer samme funktionalitet. Hertil kan nævnes ReactJS, som det alternativ jeg ville vælge, da ReactJS også kører på JavaScript og er blevet anbefalet af andre fra branchen.

4.1.2 Routeplanner

En essentiel del af projektet er en rutevejleder. For at effektivisere udviklingen og forhindre spildtid ved udviklingen af en ny rutevejleder fra bunden, diskuteres valget af ekstern rutevejleder API i dette afsnit.

Valgt: Det oplagte valg faldt på Google Maps Directions Api[4], da trafikkortet pt. benytter sig af denne. De fleste Directions API'er tilbyder lignende funktionaliteter, men for at komme i gang med projektet og benytte sig af de værktøjer, data og funktioner allerede i produktet, arbejdes der videre med Google Maps.
For at bruge deres API skal der generes en key, dette åbner op for deres service med et begrænset antal requests pr. dag.

```
Key: AIzaSyCOOD_ck2vTdtpoCkUF_GA5brvfv8A2P4c
```

Hvis der skulle opstå komplikationer med Googles API under udviklingen eller i fremtiden, er der taget design mæssige forholdsregler der gør det nemt at udskifte klassen der har alle API kald implementeret. Som beskrevet i afsnit 4.1.1 om best practices i forbindelse med eksterne kilder, har logikken sin egen Service klasse.

Alternativer: Som alternativ til Googles API, findes der en række open source produkter, samt andre tech firmaer som Bing, dog er Google en af de få API'er som tilbyder live trafik information.
Open source produkter som indledningsvis er blevet undersøgt kort til projektet:

- GraphHopper[9] - Dette produkt blev anbefalet igennem et andet Diplomingeniør projekt på DTU.
- OpenRouteService[15] - Som bygger på OpenStreetMaps blev brugt som alternativ kilde igennem praktikperioden.

4.2 Klassediagram

Det indledende klassediagram er designet på baggrund af kravspecifikationen i afsnit 3.5 og viden erfaret omkring AngularJS udvikling i afsnit 4.1.1 og grundlaget etableret på figur 8. Klassediagrammet er påvirket kraftigt af at programmet skal skrives i AngularJS, da JavaScript kombineret med Angular tilbyder andre egenskaber end Java.

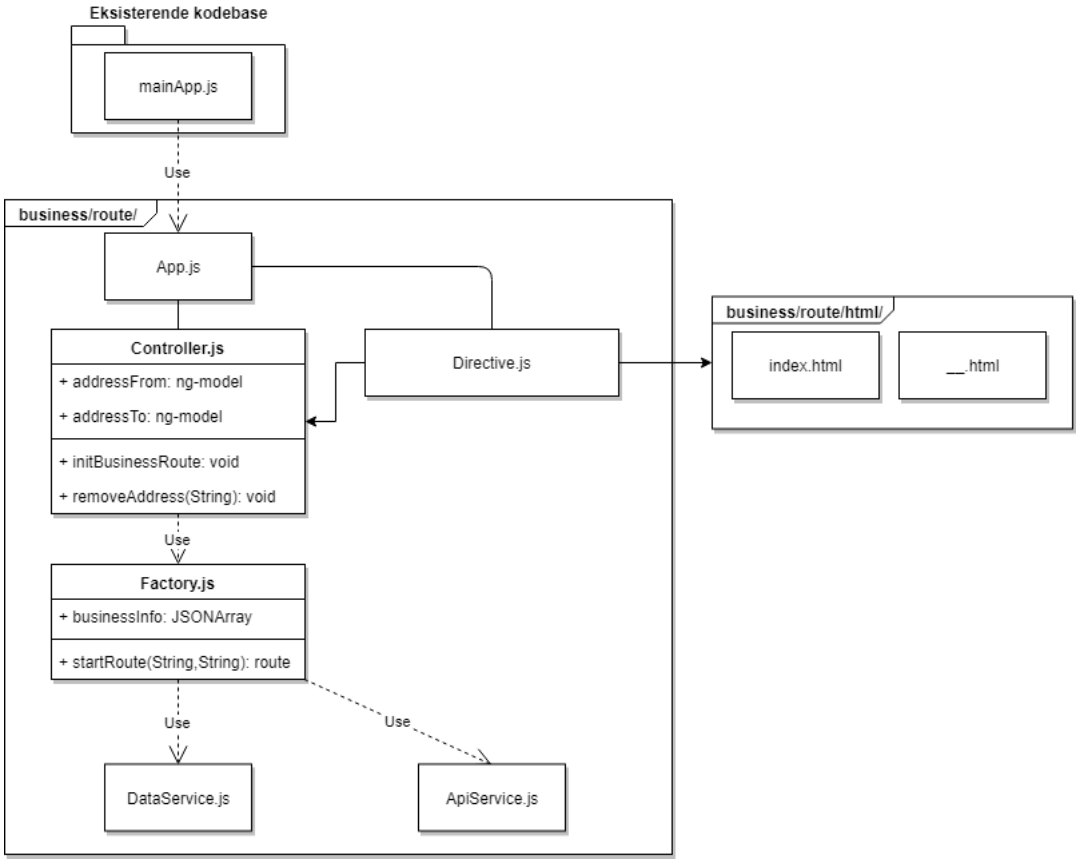

Figure 9: Indledende klassediagram

Ved nærmere inspektion af klassediagrammet, ses det at klasserne hænger sammen som dependencies og alle forhold er 1-til-1, da AngularJS Services og Factories er singletons. Som beskrevet i afsnit 4.1.1 opdeles det i noder pr. funktionalitet, dette klassediagram beskriver den ønskede struktur for noden som implementerer kernefunktionaliteten.

Intentionen er at isolere koden implementeret i dette projekt, fra den eksisterende kodebase som ses øverst i diagrammet, og bruge dependency injection til at flette det sammen.
Som en del af isolerings processen indkapsuleres koden i én folder, repræsenteret ved den relative sti beskrevet øverst i package beskrivelsen. Dette tillader at filerne kan have det samme navn som andre AngularJS klasser på tværs af det samlede projekt, og så gives de en unik identifikator som en del af implementationen, dette beskrives yderligere i afsnit 5.3.1.
Da klassediagrammet bygger på figur 8, kan der i forlængelse af beskrivelsen fra afsnit 4.1.1, tilføjes at det nederste lag, kaldet Service laget, har fået to klasser, en DataService klasse

indeholdende statisk data og hvis nødvendigt eksternt data, samt en ApiService klasse med alle eksterne API kald, som beskrevet i afsnit 4.1.2.
For at forbedre overblikket er 'model' laget fra figur 8 delt op i to forskellige lag i klassediagrammet, den anden halvdel af 'model' laget kaldes logik laget og indeholder i dette tilfælde kun en enkelt Factory klasse, som skal indeholde alt nødvendig logik i forbindelse med den ønskede kernefunktionalitet.
Den øverste del udgør så view-viewmodel, med en mere kompliceret komposition af klasserne. Dette står for præsentation til brugeren og koordinering mellem UI og JavaScript logik.
Implementationen og samarbejdet imellem Controllers, Directives og .html-pages forklares dybdegående i afsnit 5.2.3 med kode-snippets.

4.3 Delkonklusion

På baggrund af dette afsnitt kan det konkluderes, at de eksisterende frameworks har haft stor indflydelse på valget af frameworks for dette projekt, da der var stort behov for at bruge eksisterende funktionalitet. Kombineret med et design der bygger på best practices indenfor disse frameworks.

5 Implementering

Dette afsnit vil belyse de beslutninger der er taget under implementeringsfasen af projektet, samt hvordan designet er realiseret.
Indledningsvis vil trafikkortets tilstand forinden projektet blive diskuteret og hvordan dette har haft indvirkning på implementeringen. Derefter vil fokus ligge på implementeringen af prototypen, understøttet af kode udsnit og hvordan disse følger best practices diskuteret i afsnit 4.1.1.

Som en del af det eksisterende trafikkort var der allerede en privat rutevejleder, denne implementation har været en stor læring og inspirationskilde til implementationen af prototypen.

5.1 Trafikkortets tilstand

Trafikkortet er en forholdsvis stor webapplikation med mange forskellige funktionaliteter. Det er bygget af to udviklere fra Atkins, én til backenden og én til frontenden, med en kort implementeringsperiode på 4 måneder og det faktum at dette var det første AngularJS projekt i Atkins, har dette resulteret i en forholdvis rodet kodebase, som ikke altid overholder best practices, nogen af disse problemstillinger vil blive diskuteret i dette afsnit og hvordan jeg har tacklet disse problemer.

5.1.1 Map objektet

Som en vigtig del af trafikkortet bygger det meste af funktionaliteten på kortet, i form af visualisering og interaktion med kortet.
Den første komplikation kom ved realiseringen af den ønskede Controller.js (se figur 9), kaldet BusinessController fremadrettet, initialiseringen af Map objektet sker i den eksisterende Controller 'MapController.js':

```
$scope.vdtrafficmap = new google.maps.Map( document.getElementById('map_canvas'),
    mapOptions);
```

Code snippet 1: init map

Dette begrænser tilgangen til dette vdtrafficmap object fra andre AngularJS Controller klasser.
Som løsning til dette blev der afprøvet følgende workarounds:

- Tilgå MapController scopet fra BusinessControlleren, og så derved få fat på Map objektet. Dette mislykkedes da MapControlleren ikke benytter sig af et alias[3], men derimod benytter sig af angular servicen $scope.

[3]Ved brug af alias fanger man det scope/context og kan refere til dette scope igennem aliaset, se style-guide: [A.1-pkt.6a]

- Oprette et nyt Map object, på samme måde som kode udsnit 1, men da denne kommando hiver fat i samme HTML element, ødelægger det kortet for resten af applikationen.
- Til sidst blev der forsøgt at dependency injecte MapControlleren, men igen grundet $scope skabte dette uønsket kaos. Samt at Controller til Controller injection ikke var understøttet af standard injectoren.

Det konkluderes derfor at ønsket funktionalitet, designet til at være i BusinessControlleren måtte flyttes og flettes ind i den eksisterende MapController og derved tilpasse designet. *Se bilag B.3 for udsnittet af MapController med BusinessController logikken.*

5.1.2 Layers

Som en del af visualisering på kort (og GIS[4] generelt) bruges layers[5], disse layers indeholder et dataset og information om hvordan datasettet skal vises, vha. symboler og text labels.
I forbindelse med udviklingen af dette projekt skulle den eksisterende implementation omkring layers undersøges og særligt hvordan det layer indeholdende frihøjder hentes og fungerer.
De indledende undersøgelser af kodebasen gav id'et på frihøjde layeret:

```
CLEARENCE_ID = 'layer_id_9A'
```

Dette ID kunne så bruges til at hente et langt array indeholdende alle frihøjde Markers[6]:

```
var markers = $scope.markerClusterer.getMarkers(CLEARANCE_ID);
var mark = BusinessRouteFactory.findMarkerById(markers,20149);
```

Code snippet 2: Frihøjde Markers

Med dette array kunne enkelte Markers så findes ved at søge efter det ID, som vist på figur 5, her søges der specifikt efter 20149 af test årsager.

BusinessRouteFactory er en klasse implementeret som en del af prototypen, denne klasse vil blive diskuteret yderligere i afsnit 5.3

5.2 Best practices

Dette afsnit vil forklare hvordan arkitekturen og designet er udført ifølge best practices. Samtidigt vil nogle af koncepterne introduceret i afsnit 4.1.1 blive uddybet yderligere.

[4]Geografiske Informations Systemer
[5]Layers er mekanismen der bruges til at vise dataset på et kort.
[6]Markers er en google maps API klasse der specificerer et punkt

5.2.1 Factory

Kernefunktionaliteten består blandt andet af tre forskellige Factory klasser som alle følger de samme design principper, men med forskellige ansvars roller.
Interface princippet som det kendes fra Java eksisterer ikke i AngularJS, dette er dog kommet med Angular 2 der bygger på TypeScript. Men et princip implementeret som en del af Factories kan sammenlignes med Interfaces:

```
1  //Interface
2     var service = {
3         //Fields
4         map: map,
5         placesService: placesService,
6         businessInfo: businessInfo,
7         directionsService: directionsService,
8         travelMode: travelMode,
9         unitSystem: unitSystem,
10        directionsDisplay: directionsDisplay,
11
12        //Functions
13        setupAutoComplete: setupAutoComplete,
14        init: init,
15        setMarker: setMarker,
16        startRoute: startRoute,
17        initDirectionsDisplay: initDirectionsDisplay,
18        findMarkerById : findMarkerById,
19        validateRoute : validateRoute
20
21    };
22    return service;
```

Code snippet 3: Interface
Se bilag B.4 for kontekst med resten af klassen

På kode udsnit 3 ses et udsnit af BusinessRouteFactory klassen, dette dikterer hvilke funktioner og variabler som skal udstilles og være tilgængelige. Dette følger principperne om at holde callable members i toppen [A.1-pkt.6b] og funktion deklarationer [A.1-pkt.6c], disse principer kombineret giver en række fordele, som gør det nemmere at læse, udvide og genbruge fremadrettet.

Sammenlignet med klassen MapRouteFactory fra den eksisterende kodebase, som bruger function expressions:

```
1  mapRouteFactory.showAlert = function(message) {
2    alertModalContent.text(message);
3    alertModal.modal();
4  }
```

Code snippet 4: MapRouteFactory - function expressions

Hvor funktion deklaration og implementation sker samme sted, som i længden giver et dårligere overblik over hvad klassen kan og hvilke funktioner der eksisterer, samt alle funktioner og variabler er public.

5.2.2 Dataservices

ApiServicen og DataServicen er implementeret efter samme princip som kode udsnit 3. Derudover følger de princippet om isolering af data kald [A.1-pkt.6d]. Som eksempel se kode udsnit 5 hvor API kaldet samt validering af input data isoleres i en funktion:

```
1  function getAutoComplete(input, options) {
2    console.log('Getting AutoComplete');
3
4    if(input && options) {
5      return new google.maps.places.Autocomplete(input, options);
6    } else {
7      console.log('Error : Missing parameters');
8    }
9  }; //getAutoComplete
```

Code snippet 5: ApiService funktion eksempel

Det samme kan tilføjes til Dataservicen som indeholder alle datastrukturer nødvendige igennem applikation, som eksempel i udsnit 6 bruges dette JSONObject som opbevaring af kritisk information i BusinessRouteFactory klassen.

```
1  function getBusinessInfo(){
2    var toReturn = {
3      from : {},
4      to : {},
5      route : {}
6    };
7    return toReturn;
8  };
```

Code snippet 6: DataService funktion eksempel

Efter erfaring giver disse principper en række vigtige egenskaber:

- Det gør det nemt at finde datastrukturen, hvis der er noget som skal rettes, tilføjes eller slettes, da det hele er samlet ét sted.
- Ved evt. udskiftning af et API kan den enkelte ApiService klasse skiftes og den eksisterende kode kan bruge den samme kode på baggrund af Interface princippet

beskrevet i udsnit 3.

- Det gør det muligt at simulere data kald til et eksternt API eller database som ikke er færdig udviklet endnu.

Hvis problemstillingen udstillet i kode udsnit 1, havde fulgt principperne om Dataservices [A.1-pkt.6d] uddybbet her i afsnit 5.2.2, kunne det kritiske Map objekt være udstillet igennem enten en ApiService eller en DataService som derfra kunne være tilgængelig fra alle Controllers.

5.2.3 View-ViewModel

Den ønskede komposition af klasserne Directives og Controllers kombineret med .html sidderne set på figur 8, bygger på en række style guide principper som vil blive uddybet i dette afsnit.

Directives er det bindende led:

```
1  (function(){
2    'use strict';
3
4      angular
5        .module('mainApp.business')
6        .directive('toolTip',tooltipFunction);
7
8      function tooltipFunction(){
9        return {
10         restrict : 'E',
11         templateUrl : 'js/components/business/tooltip/index.html',
12         controller : 'TooltipController',
13         controllerAs : 'tooltipCtrl'
14       };
15     };
16
17 })();
```

Code snippet 7: Tooltip.Directive.js

Her ses det custom directive 'toolTip' indeholdende referencer til:

- En HTML fil:

```
templateUrl : 'js/components/business/tooltip/index.html'
```

- En Controller, samt alias [A.1-pkt.6a]:

```
controller : 'TooltipController',
controllerAs : 'tooltipCtrl'
```

"Restrict : 'E'" definerer at det skal compiles til et html **E**lement, alternativt kan 'A' bruges, som compiles til et html **A**ttribut. Navnet 'toolTip' i camelCase oversættes til dashes i html: <tool-tip> </tool-tip>, som det ses i udsnit 8 linje 4.

```
1  <hr class="mapviewer_panel_hr"/>
2  <div ng-init="initBusinessRoute()" class="nav nav-list">
3    <div class="title_new">S&oslash;g rute
4      <tool-tip></tool-tip>
5    </div>
```

Code snippet 8: index.html udsnit

Disse komponenter kombineret tillader tooltip.html at tilgå variabler og eksekvere funktioner i RouteController igennem dets alias, som det ses på udsnit 9 linje 5:

```
1 <img class="icon_img_tooltip routeinfotooltip"
2 tooltip-hide-trigger="mouseleave"
3 tooltip-show-trigger="click mouseover"
4 tooltips
5 tooltip-content="{{tooltipCtrl.tooltip}}"
6 tooltip-size="small"
7 tooltip-side="right"
8 tooltip-class="tooltips"
9 alt="" src="./images/icon/info.png" />
```

Code snippet 9: tooltip.html

Som refererer til udsnit 10 linje 12

```
1  (function(){
2    'use strict';
3
4    angular
5      .module('mainApp.business')
6      .controller('TooltipController',tooltipController);
7
8    function tooltipController(){
9      //VM capture
10     var self = this;
11
12     self.tooltip = "Tooltip goes here";
13   }; //RouteController
14 })(); // Closure
```

Code snippet 10: TooltipController

Kode udsnit 7 - 10 ligger grundlag for hvordan klasserne interagere og spiller sammen, derudover er der fremstillet argumentationer for hvordan disse følger design principperne om:

- High cohesion Directives files [A.1-pkt.6e].
- ControllerAs [A.1-pkt.6a].

Disse principper kan opsummeres til at give en overskuelig og forståelig kode. Eksempler fra den eksisterende kodebase med mangel på følgende principper er for store til rapporten, da klasser og html sider er op til 5000 tusind linjer kode.

5.3 Funktionalitet

Dette afsnit vil belyse det samlede overblik over de ønskede funktionaliteter beskrevet i afsnit 3.3, dette indebærer status for den enkelte funktionalitet, forhindringer undervejs, evt. interessante kode udsnit og hvordan det udvikles fremadrettet.

5.3.1 Funktionalitet isolering

I forlængelse af klassediagrammet uddybes isolerings processen, ved at belyse filstrukturen og unik identifikator/klasse navn princippet i AngularJS.

```
- /staticweb/vd-trafikkort/war/js/components/business/
        - app.js
        - id
            - id-entrance.html
            - Id.Directive.js
        - inspect
            - index.html
            - bro.jpg
            - Factory.js
            - Inspect.Directive.js
            - InspectController.js
        - route
            - index.html
            - Factory.js
            - Route.Directive.js
        - services
            - DataService.js
            - GoogleApiService.js
        - tooltip
            - index.html
            - Tooltip.Directive.js
            - TooltipController.js
```

Figure 10: Filstrukturen for projektet

Filstrukturen
Projektets filstruktur kan ses på figur 10, '/components/' indeholder alle komponenterne fra den eksisterende kodebase, og prototypen er så isoleret i '/business/'. Herunder er de enkelte elementer og funktionaliteter så isoleret yderligere.

- **id**: Indeholder filerne til valg af metode og indtast af id, som set på figur 11.

Figure 11: Id sektion

- **inspect**: Indeholder filerne til use casen 'inspicer bro' beskrevet som bonus 1.
- **route**: Indeholder filerne til use casen 'find rute' beskrevet som kernefunktionaliteten.
- **services**: Indeholder de nødvendige service klasser.
- **tooltip**: Indeholder logikken til tooltipet. Dette var originalt tænkt som værende 'BusinessController', men under implementationen og grundet de komplikationer der opstod undervejs, skrumpede funktionaliteten der var nødvendig i denne klasse.

Unik identifikator
På figur 10 bemærkes det at 'inspect' og 'route' begge indeholder en fil kaldet 'Factory.js', dette er muligt da de ligger i hver sin mappe, og angular compileren kigger på den angivede unik identifikator i koden, frem for filnavnet.
Denne unik identifikator defineres ved initialiseringen af modulet, som det kan ses på udsnit 11 og 12 (linje 1-6 i de respektive filer).

```
1 (function(){
2   'use strict';
3
4   angular
5     .module('mainApp.business')
6     .factory('BusinessRouteFactory',businessRouteFactory);
```

Code snippet 11: /route/Factory.js

```
1 (function(){
2   'use strict';
3
```

```
4  angular
5    .module('mainApp.business')
6    .factory('InspectFactory',inspectFactory);
```

Code snippet 12: /inspect/Factory.js

module('mainApp.business') bruges til at koble disse factories på modulet kaldet 'mainApp.business' defineret i app.js i root folderen '/business/.
Hver Factory får så deres unik identifikator ved den String givet i '.factory(String,Function)'-metoden, hhv. 'BusinessRouteFactory' og 'InspectFactory'. Function parameteren er så selve factory implementationen givet længere nede i filen.

Dette giver så adgang til disse factories igennem injectoren, ved unik identifikatoren som det ses på udsnit 13 linje 8 og 10, her får injectoren unik identifikatoren 'InspectFactory' som String, og i funktionen kommer den så som input parameter.

```
1  (function(){
2    'use strict';
3
4    angular
5      .module('mainApp.business')
6      .controller('InspectController',inspectController);
7
8    inspectController.$inject =
         ['GoogleApiService','InspectFactory','DataService','utilFactory']
9
10   function inspectController(GoogleApiService,InspectFactory,DataService,utilFactory){
```

Code snippet 13: InspectController l.1-10

5.3.2 Kernefunktionalitet

Kernefunktionaliteten fungerer til en vis grad. Den er pt. bygget op omkring Googles Directions API, som under implementeringen er blevet undersøgt yderligere og det er blevet konstateret at den ikke understøtter "negative viapunkter"/"Avoid areas", dette har været en ønsket change request[5] hos Google de sidste 10 år.
For at arbejde videre med projektet og det arbejde der var lavet indtil videre, blev andre aspekter af problemstillingen fokuseret på, og derved er der lavet et mindre workaround til dette.

Prototypen understøtter pt:

1. At indtaste adresser.
2. Efter indtastning af adresser requester den en rute fra Google API'et samt nogle alternative ruter.

3. Efterfølgende kan ruten så valideres via 2 metoder:
   (a) Metode 1: Om en angivet bro skaber problemer.
   (b) Metode 2: Om alle broer skaber problemer.
4. Hvis algoritmen finder en problem bro, søger den blandt de alternative ruter returneret fra step 2, og om disse alternative ruter indeholder en problem bro.
   - Hvis alle de alternative ruter indeholder problemer, smides der en fejl og konstatere at der ikke kan findes en alternativ rute.
   - Hvis en af de alternative ruter kan valideres, sættes denne som aktiv.

For at se koden der implementerer de enkelte trin, se bilag B.5.

Dette demonstrerer den ønskede funktionalitet i den indledende problemformulering, beskrevet som kernefunktionalitet.

Fremadrettet

På baggrund af information erfaret igennem projektet, kan det konkluderes at det ikke er muligt at løse problemstillingen optimalt ved brug af Googles Directions API, da det ikke understøtter denne funktionalitet.
Løsningforslag/workarounds til den manglende funktionalitet kunne indebære:

- Udvikle en algoritme der kan bruge de "almindelige viapunkter", til at tvinge Google ruten under broer som systemet ved er godkendte.
- Uddrage funktionaliteten i sit eget backend system og så integrere det med trafikkortet. Backenden kan så benytte sig af en anden Directions API end Googles, og så afslutningsvis snappe det til Googles opfattelse af vejkoordinater, inden det returneres til trafikkortet. Da koden udviklet i dette projekt er skrevet i næsten rent JavaScript, kan backenden med fordel udvikles i Node.js, da denne kombination af JavaScript biblioteker er en anerkendt stack i branchen, kendt som MEAN[13] (**M**ongoDB, **E**xpress, **A**ngular, **N**ode). Derudover kunne et evt. backend system til denne funktionalitet udstilles offentligt så interessanter som indlandske og udenlandske erhvervschauffører kunne benytte den.

5.3.3 Bonus 1

Der er udviklet et løsningsforslag til funktionaliteten beskrevet på baggrund af Vejdirektoratets perspektiv. Dette løsningforslag opfylder kravene pkt. 1 og 2 sat i afsnit 3.5 for Bonus 1. Udviklingen og implementeringen for denne funktionalitet vil blive beskrevet i dette afsnit.

Designet følger principperne beskrevet i afsnit 4.1.1.

Indhentning af den udvidede information fra DANBRO er ikke blevet implementeret (afsnit 3.5 Bonus 1 pkt.3), i stedet er der taget udgangspunkt i en enkelt bro med ID: 10048, udvidet information om denne bro er så indhentet manuelt fra DANBRO, se figur 12.

Figure 12: Udvidet frihøjde information om bro ID: 10048

Disse værdier er så hardcoded i DataServicen, se udsnit 14.

```
1   function getAdvancedBridgeInformation(){
2     var toReturn = {
3       vejside1: {
4         v : 4.53,
5         m : 4.62,
6         h : 4.7
7       },
8       vejside2: {
9         h : 4.57,
10        m : 4.67,
11        v : 4.77
12      }
13    };
14    return toReturn;
15  }
```

Code snippet 14: DataService.js udsnit

Interaktion med rute

I forbindelse med Bonus 1 pkt.2 fra afsnit 3.5 bliver information fra udsnit 14 så injectet ind i det originale mark object og overskriver den simple udgave af information, og det bliver så efterfølgende sendt med en ekstra variabel 'vejside2', videre til validateRoute() funktionen, se udsnit 15.

```
1 var mark = businessRouteFactory.findMarkerById(markers,10048);
2 mark.feature_properties = DataService.getAdvancedBridgeInformation();
3 businessRouteFactory.validateRoute(mark,truckHeight,0,'vejside2');
```

Code snippet 15: MapController.js udsnit

Logikken fra den originale valideringsmetode fra afsnit 5.3.2 pkt. 3a er så udvidet til også at håndtere hvis 'advanced' metoden er valgt i drop down menuen. Her kan det så validere på spor niveau om frihøjden kan godkendes. Og leverer evt. ekstra vejledning i en prompt, se figur 13.

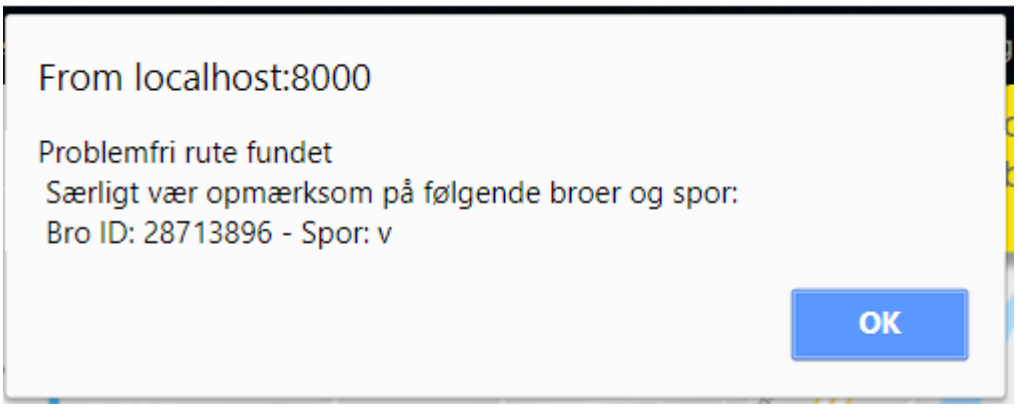

Figure 13: Ekstra vejledning til ruten

Bro ID'et i prompten er broens "featureId", da dette er identifieren i layer'et, og det originale Bro ID "10048" kun eksisterer i "description", dette ID er identifieren i DANBRO.
Det vides ikke om der findes en mapning af disse IDs.

Visualisering

Jævnfør afsnit 3.5 Bonus 1 pkt.1 er der udviklet et løsningsforslag til visualiseringen af dette. På samme vis som interaktionen med ruten, bygger dette på den hardcoded information fra udsnit 14.

Til sammenligning se den eksisterende løsning på figur 5 i afsnit 3.2.
Løsningsforslaget tager udgangspunkt i scenariet, hvor brugeren har kendskab til lastbil højden og ruten der eksisterer mellem 'Jyllingevej 200, Rødovre' og 'Islevdalvej 200, Rødovre', hvilket evalueres til interaktion med bro ID: 10048 - 'vejside2', i data settet.
For at forbedre visualiseringen på spor niveau, er perspektivet kørt ned, så der kigges på broen i fugleperspektiv i stedet for satellit perspektivet, se figur 14.

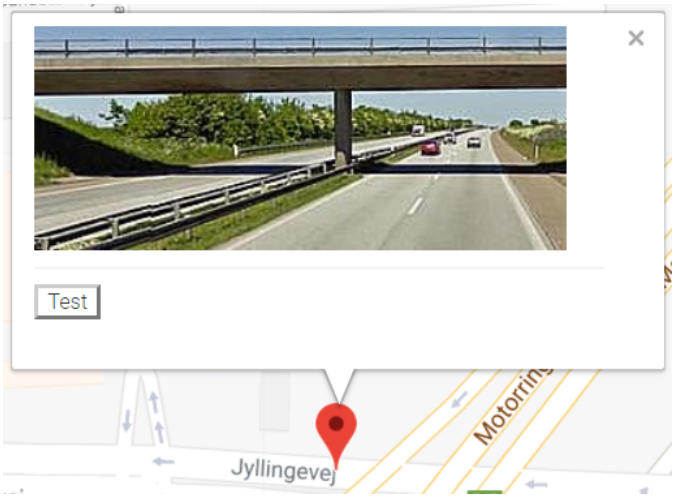

Figure 14: Bonus 1 - løsningsforslag

Dette giver så mulighed for at visualisere en højde restriktion på spor niveau, se figur 15, hvor funktionaliteten er testet med en lastbil højde på 4m & 67cm.

Figure 15: Bonus 1 - inkl. højde restriktions test

Her visualiseres et godkendt spor med en grøn bjælke på den dertilhørende frihøjde og henholdsvis en rød bjælke på afviste spor.

Alt kode til denne funktionalitet følger principperne beskrevet og er isoleret i undermappen "inspect".

Fremadrettet

Implementeringen til analysen af ruten og frihøjde på spor niveau er skrevet generisk (se udsnit 16 [7]), så når udvidet data kan hentes fra DANBRO burde det tilpasse sig og fungere på et nationalt plan.

Inspicering af bro skal erstatte de eksisterende ikoner og udvides til at understøtte begge

[7] roadSide = 'vejside2' fra udsnit 15, road = 'h','m','v' fra udsnit 14

sider af vejen enten samtidigt eller ved en switch, og udvides med det komplete datasæt fra DANBRO.

Rute interaktion kan udvides til at levere en modificeret udgave af den detaljeret rutebeskrivelse, så det nødvendige sporvalg eksisterer sammen med de enkelte steps leveret fra Google, som fx: "Tag mod vest ad Jyllingevej mod Vedersøvej".

```
1 for( var road in mark.feature_properties[roadSide] ){
2     var bridgeHeight = mark.feature_properties[roadSide][road];
3     if(bridgeHeight > truckHeight){
4         DataService.addAdvancedBridge(mark.featureId,road);
5          return false;
6     }
7 }
```

Code snippet 16: BusinessRouteFactory.js - function isMarkCausingProblems

5.3.4 Bonus 2

Der er ikke arbejdet på denne Bonus funktionalitet.

5.4 delkonklusion

På baggrund af dette afsnit kan der drages en række konklusioner:

- Det eksisterende trafikkort har haft positiv påvirkning på projektet, i form af inspiration til hvordan der arbejdes indenfor domænet, og negativ påvirkning ved den mindre optimale kodebase som har forhindret det ønskede design.
- Det indledende analyse arbejde omkring best practices er blevet benyttet kraftigt, og har givet positive resultater igennem alle faserne af implementeringen.
- Google Maps API'et viste sig ikke at være optimalt til at undgå specifikke punkter/områder. Men der blev udarbejdet et workaround for at showcase resten af prototypen.
- På trods af udfordringer og information erhvervet under implementation, er kerne- og bonus 1 funktionaliteten implementeret til bedste formåen.

6 Test

I dette afsnit vil de test cases udformet på baggrund af use cases i afsnit 3.4, blive beskrevet og der konkluderes på baggrund af resultatet.
Derudover vil test aspektet af en AngularJS applikation blive belyst og hvilke frameworks og fremgangsmetoder der kan bruges.

6.1 Test cases

Alle test cases bygger på en metode lært fra praktikperioden hos Atkins Danmark A/S, hvor testen udarbejdes på baggrund af use cases og kravspecifikationen og derved beviser test cases at systemet lever op til kravene.

Der udarbejdes én test case pr use case og evt. alternativt forløb til disse use cases.
Test casen har en række steps som hver har 3 sektioner:

1. Step beskrivelse
2. Test data
3. Forventet resultat

6.1.1 Test case 1: Find rute

På baggrund af use casen 3.4.1 og kravspecifikationen (afsnit 3.5) opsummeres de i en liste af acceptkriterer, alle acceptkriterer skal tilfredsstilles.

Acceptkritier

1. Mulighed for at indtaste køretøjsinformation[8].
   (a) **Alternativ:** Aktøren glemmer at indtaste køretøjsinformation - Systemet leverer en fejlmeddelse.
2. Mulighed for at indtaste afgangsadresse.
   (a) **Subcase:** Mulighed for at slette afgangsadressen.
3. Mulighed for at indtaste destinationsadresse.
   (a) **Subcase:** Mulighed for at slette destinationsadressen.
4. Systemet skal levere en valid rute.
   (a) **Alternativ:** Valid rute kunne ikke findes - systemet leverer en fejlmeddelse.

[8]Denne funktionalitet eksisterede i forvejen

#	**Step beskrivelse**	**Test data**	**Forventet resultat**
1	Åben trafikkortet.	`http://localhost:8000/index.html?showhead=false&showfooter=false`	Succes.
		Acceptkriterie 1:	
2	Prototypen arbejder med frihøjder, hvis ikke valgt som standard, vælg da "Frihøjder" i dropdown menuen.	Indtast ønsket lastbil højde hhv. 'm' og 'cm'.	Ved klik på 'vis' burde clearance layer'et blive loadet, med ikoner der indikerer om den er clear eller ej.
		Acceptkriterie 2 & 3 : *Se bilag B.6*	
3	Som det ses på billedet er der 2 felter til indtastning af 'Fra' og 'Til' (Bilag B.6 figur 16).	Påbegynd indtastning af vilkårlig adresse (Bilag B.6 figur 17).	Autocomplete forslag fra Google, vælg et forslag i 'Fra' og 'Til'.
		Acceptkriterie subcases 2.a og 3.a: *(Bilag B.6 figur 18)*	
4	På billedet ses 'Fra' og 'Til' udfyldt, samt et 'X' er blevet synligt. Dette 'X' sletter adressen i det felt der klikkes. For at fjerne begge adresser benyt 'Fjern rute'.		Ved sletning af en adresse eller begge, burde ruten bliver annulleret.
		Acceptkriterie alternativ 1.a: *(Bilag B.6 figur 18)*	
5	På billedet ses det at en normal rute er fundet, men brugeren har glemt lastbilhøjde. Klik 'Valider rute'.		Dette burde give en fejlmeddelse: 'Der blev ikke fundet nogle frihøjder' (Se bilag B.6 figur 19).
		Acceptkriterie 4:	
6	Indtast en lastbilhøjde for at undgå fejlmeddelsen fra step #5. Efter indtastning af test data klik 'valider rute'.	Som eksempel kan følgende data bruges: -'Fra' Jyllingevej 200, Rødovre -'Til' Islevdalvej 200, Rødovre - Lastbilhøjde 4m 50cm	Valideringen burde analysere den almindelige rute, rapportere at der er problemer på ruten (Bilag B.6 figur 20). Og herefter succesfuldt finde en alternativ problemfri rute (Bilag B.6 figur 21).

Acceptkriterie subcase 4.a:			
7	Indtast test data og klik 'Valider rute'.	Som eksempel kan følgende data bruges: -'Fra' Hulgårdsvej 128, København -'Til' Arne Jacobsens Allé 17, København - Lastbilhøjde 4m 50cm	Igen burde valideringen analysere den almindelige rute, rapportere at der er problemer på ruten (Bilag B.6 figur 20). Og herefter konstatere at den ikke kan finde en alternativ rute (Bilag B.6 figur 22).

6.1.2 Sub test case: Find rute *advanced*

I forlængelse af test case 1 kombineret med Bonus 1 funktionaliteten pkt.2, er denne sub test case lavet for at teste den ekstra funktionalitet introduceret ved den udvidede information interaktion med ruten.

Acceptkriterier

Acceptkriterierne udvider pkt.4 fra test case 1

1. Ruten skal tage hensyn til den udvidede information.
2. Systemet skal levere ekstra information om de broer der kræver særlig opmærksomhed mht. frihøjde.

#	**Step beskrivelse**	**Test data**	**Forventet resultat**
Acceptkriterie 1 & 2:			
1	Indtast en adresse der krydser den specfike bro, og vælg 'advanced' i drop down menuen,.	-'Fra' Jyllingevej 200, Rødovre -'Til' Islevdalvej 200, Rødovre	En rute der krydser den specifike bro, burde optræde på kortet.
2	Indtast en lastbilhøjde.	- Lastbilhøjde mellem 4m & 58cm og 4m & 76cm	
3	Afslut med 'Valider rute'.		Systemet leverer et specifikt spor for at cleare broen, samt ID'et på broen.
4	Indtast en lastbilhøjde.	Højde over 4m & 78cm	Systemet burde konkludere at broen skaber problemer og levere en anden valid rute.

6.1.3 Test case 2: Inspicer bro

På baggrund af use casen 3.4.2 og kravspecifikationen (afsnit 3.5) opsummeres de i en liste af acceptkriterer, alle acceptkriterer skal tilfredsstilles.

Acceptkriterier

Krav om request af data fra DANBRO er udeladt da dette ikke er implementeret.

1. Mulighed for at indtaste køretøjs information[9].
2. Mulighed for at inspicere en bro - I dette tilfælde findes der kun et enkelt punkt/bro der kan inspiceres, ved klik på "Test Inspicer Bro" centreres denne på kortet.
   (a) **Alternativ**: Casen afbrydes ved klik på "Reset Inspicer Bro" - Dette fjerner markøren og. zoomer ud.
3. Mulighed for at teste frihøjden.

#	**Step beskrivelse**	**Test data**	**Forventet resultat**
Acceptkriterie 1:			
1	Indtast køretøjsinformation.	Lastbilhøjde mellem 4m & 58cm og 4m & 76cm	Layer'et burde vise ikoner fordelt på kortet.
Acceptkriterie 2:			
2	Klik 'Test Inspicer Bro'.		Kortet burde være zoomet ind og et Mark burde dukke op på kortet.
Alternativ 2a:			
3	Klik 'Reset Inspicer Bro'.		Kortet burde være zoomet ud og markøren fjernet.
Acceptkriterie 2 *fortsættelse*:			
4	Klik på markøren og derefter på 'Test' knappen.		3 streger burde dukke op under broen med farvekode grøn eller rød hhv. om dette spor er godkendt eller ej.
5	Klik 'Clear', gentag step #1 med ny bro højde og Test igen.		

[9]Denne funktionalitet eksisterede i forvejen

6.1.4 Resultat

Test case 1: Find rute Opfylder alle acceptkritierne opstillet inklusiv alternative forløb og subcases.

Sub test case: Find rute *advanced* Opfylder alle acceptkritierne opstillet.

Test case 2: Inspicer bro Opfylder alle acceptkritierne opstillet inklusiv alternative forløb. Dog er funktionalitet om indhentning af data fra DANBRO udeladt.

6.2 AngularJS testing

AngularJS testing kategoriseres normalt i sammenhæng med TDD[10] eller BDD[11]. TDD kræver at testene skrives først og så implementeres logikken til at opfylde disse tests efterfølgende, derfor kategoriseres dette projekt som BDD.

Til BDD testing anbefales[8] en kombination af to frameworks, Karma[7] og Jasmine[11], guides og ressourcer brugt til opsætning og introduktion findes i appendix A.2.

Karma fungerer som motoren til testen og starter en webserver op der eksekvere source kode op mod test kode, og kan konfigureres til test på tværs af browsers.

Jasmine er JavaScript frameworket specielt designet til BDD testing, hvor testene skrives efter deres syntax.

Alternativ

Alternativt til de to frameworks anbefalet, eksisterer der selvfølgelig andre frameworks på markedet, som også kan teste JavaScript kode, her kan nævnes:

- Selenium `https://www.npmjs.com/package/selenium-webdriver`
- Jest `https://facebook.github.io/jest/`
- Cucumber `https://cucumber.io/`

Blandt mine kollegaer anbefales og bruges Selenium.

6.2.1 Karma & Jasmine testing

Udover installationen af Karma og Jasmine igennem npm[14], kræves en konfig fil der initieres igennem CLI'et[12] ved 'karma init'. Efterfølgende skal denne så rettes til med filestier:

[10] Test Driven Development
[11] Behaviour Driven Development
[12] Command Line Interface

```
files: [
    'js/bower_components/angular/angular.js',
    'js/app.js',
    'tests/*.js'
],
```

Ved kommandoen 'karma start' kører den alle tests i 'tests'-mappen med mønstret '*.js'. På udsnit 17 ses et test eksempel.

```
1 describe('example test', function() {
2   it('should be true', function() {
3     expect('foo').toBe('foo');
4   });
5 });
```

Code snippet 17: example.js

'describe' bruges som en overordnet funktion, til en række 'it' tests indenfor den samme kategori. Der kan tilføjes andre funktioner som fx. 'beforeEach' og 'it' testene kan bruge en række assert statements som gives fra Jasmine frameworket.

I forbindelse med unit testing af dette projekt, er det ikke lykkkedes at få komponenter til at spille sammen. Dette skyldes en række problemer med uvidst alvorlighedsgrad:

- I kombination med Jasmine og Karma anbefales[8] et bibliotek kaldet 'angular-mocks', der mocker de nødvendige dependencies på de klasser der testes, det har ikke været muligt at få biblioteket til at køre succesfuldt.
- Det eksisterende npm (bower) setup i projektet, har skabt problemer ved installationen af 'angular-mocks', da det skal tvinges til at køre samme version som det eksisterende 'angular' bibliotek. Dette har ikke været muligt.

Da det er uvidst præcist hvad fejlen er, har det ikke været muligt at udvikle og execute de ønskede unit tests. Forsøget på en unit test kan ses på udsnit 18.

```
1  (function(){
2    'use strict';
3
4    describe('firstFactoryTest',function(){
5      var routeFactory;
6
7      beforeEach(inject(function(BusinessRouteFactory){
8        routeFactory = BusinessRouteFactory;
9      }));
10
11
12     it('inspectFactory should be defined',function(){
```

```
13      expect(routeFactory).toBeDefined();
14    });
15  }); //describe
16 })();
```

Code snippet 18: factory.js

Den almindelige test givet ved eksempel på udsnit 17 kører igennem succesfuldt, men ved forsøget på inject af klassen der skal testes 'BusinessRouteFactory', fejler testen da 'inject' ikke kendes af compileren, 'inject' burde være givet af de angivede frameworks og biblioteker.

6.3 Delkonklusion

Det konkluderes at den ønskede funktionalitet og opførsel opsummeret i de respektive acceptkritier opfyldes af det udviklede system.
Der er dog ikke lavet test cases til funktionalitet som ikke er implementeret, her kan nævnes indhentning af udvidet data fra DANBRO og funktionalitet beskrevet som Bonus 2 i afsnit 3.3.

Yderligere konkluderes det at unit testing implementering er fejlet, grundet komplikationer med biblioteker og uvidste fejlkilder.

7 Konklusion

Konkluderende på projektet og opsummerende over alle delkonklusionerne, kan det siges at den originale problemstilling er blevet udfordret og udvidet, og velargumenterede løsningsforslag samt implementationer til disse problemstillinger, er blevet fremstillet igennem projektet.

Indledningsvis til projektet var problemstillingen defineret i samarbejde med Atkins og deres opfattelse, men igennem analysefasen blev det konkluderet at kundens synspunkt på problemstillingen varieret fra Atkins, disse synspunkter blev kombineret til en samlet problemstilling, indeholdende den originale vinkel med fokus på interaktion imellem data og rute, kombineret med den forbedret data allerede eksisterende i Vejdirektoratets (VD) interne database (DANBRO) og hvordan dette præsenteres for brugeren. På baggrund af den kombineret problemstilling blev der så udarbejdet en kravspecifikation til systemet.

Kravpsecifikationen satte en række funktionalitetskrav til systemet, som skulle opfyldes og understøttes i valget af frameworks, for at kunne arbejde med data fra VD og den eksisterende funktionalitet, faldt valget på de eksisterende frameworks, AngularJS og Google's Directions API. Disse frameworks lagde grundlaget for designet og derudover var dokumentationen dertil en stor hjælp under udviklingen af systemet, i form af best practice guidelines.

Disse best practice guidelines lagde grundlaget for introduktionen til det ukendte framework AngularJS og sænkede læringskurven. Ved udforskningen af den eksisterende kodebase fandt jeg en del variation i kodekvaliteten, som resulterede i nogle ændringer til designet, men den eksisterende kodebase gav samtidigt også en introduktion til hvordan der arbejdes indenfor domænet.
Efter designet var realiseret og delelementerne til rute-bro interaktionen var på plads, opstod der komplikationer med Googles Directions API, da data om broerne og den ønskede rute skulle arbejde sammen, da APIet ikke er i stand til at modtage negative viapunkter. Denne funktionalitet har været en request hos Google i over 10 år, dette kunne være undgået ved en mere dybdegående analyse af APIet forud for projektet.
På trods af disse komplikationer blev der arbejdet videre indenfor API'ets rammer, og på baggrund af dette, er der så udviklet en rutevejleder som kan levere en rute, analysere broerne på ruten, og hvis muligt og nødvendigt finde en alternativ rute.

Yderligere blev der udviklet en løsning til præsentations problemet fremstillet af VD, der udnytter den udvidede information fra DANBRO, og visualiserer frihøjderne fra et andet perspektiv, der gør det muligt at præsentere data på vejbane niveau.

Løsningsforslagene og implementeringen dertil, opsummeres i en række test cases der tager udgangspunkt i dertilhørende use cases og kravspecifikationen. Disse test cases tester applikationen i user test format og konkluderer at systemet fungerer efter hensigten.
Dog blev konkret test af koden i form af unit testing aldrig muligt grundet komplikationer med projekt setup, frameworks og eksterne biblioteker.

8 Perspektivering

Dette afsnit vil starte med en perspektivering, til hvordan prototypen kan bruges og udvides i fremtiden på baggrund af konklusionerne draget i denne rapport, derefter vil det blive perspektiveret til Atkins fremtidige strategi, og til sidst hvordan det relaterer til det generelle erhvervsliv. *I forlængelse af konklusionerne om fremtidig udvikling fra afsnit 5.3, opsummeres her et samlet forslag.*

Fremtiden om denne applikation kan udvides til en komplet understøttelse af ruteplanlægning for erhverschauffører, dette indebærer de fuldt ønskede funktionaliteter beskrevet som kernefunktionalitet, og bonus 1 og 2, samt udvidelse til de resterende aspekter af erhvervs transport, modulvogntog, transport af farlig gods, vindmølle moduler, og militære formål med henblik på mulige transportruter af tunge køretøjer som fx. kampvogne.
Yderligere kan rutevejlederen udviklet i dette projekt udvides til en mobil app fokuseret på Truckers, der indeholder ruterne de skal køre, mulighed for inkorporation af køre- hviletid, og rastepladser. Rutevejledning kan kombineres med en "Travelling salesman" algoritme der kan forbedre køretiden for chaufførerne, samt formindske benzin forbruget.

Strategien for Atkins kan alignes med fremtiden for applikation i form af implementation af nye teknologier som Machine Learning og IoT. **Machine learning** kan anvendes på de tidligere nævnte funktionaliteter omkring køre- hviletider, rastepladser og yderligere trafik data. Som den eksisterende funktionalitet til private kan trafik dataen kombineres med Machine learning til at forudsige trafik mønstre og levere ruter på baggrund af dette, denne anvendelse af machine learning benyttes allerede igennem det nuværende Google API, men ved en alternativ implementering, kunne dette være en ekstra feature.

Derudover kan **IoT** benyttes til indsamling af data ved overvågning af broer og andre enheder på vejnettet, og derigennem understøtte teknologier som Big Data og Machine learning. Ved overvågningen af broer kan IoT enheder udvides til automatisk overvågning af frihøjder og evt. skader på kritiske enheder langs vejnettet.

Generelt kunne en fokuseret udvikling af de nævnte funktionaliteter og aspekter af emnet have en bred effekt på erhvervslivet. Vejdirektoratets arbejde kunne lettes og avanceres i form af de nævnte anvendelsområder ved overvågning af vejnettet. Transport og logistik firmaer kunne få en pålidelig datakilde der reducerede ulykker og farlige situationer ved ulovlige opmålinger, samt reducere deres transport udgifter. Den voksende vindmølle industri kunne også benytte den uvidede information ved planlægning og transport af de meget lange vindmølle dele.

A Appendix

A.1 AngularJS Ressourcer

1. https://www.reddit.com/r/angular/
2. https://www.reddit.com/r/angularjs/wiki/index
3. https://docs.angularjs.org/api
4. https://www.youtube.com/channel/UCbn1OgGei-DV7aSRo_HaAiw
5. https://www.codeschool.com/courses/shaping-up-with-angularjs
6. https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.mdtable-of-contents
   (a) **Princip:** "ControllerAs"
   Link: `https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md#controlleras-view-syntax`
   (b) **Princip:** "Accessible Members Top"
   link: `https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md#accessible-members-up-top`
   (c) **Princip:** "Funktion Deklaration"
   link: `https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md#function-declarations-to-hide-implementation-details-1`
   (d) **Princip:** "Dataservices"
   link: `https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md#data-services`
   (e) **Princip:** "Directives"
   link: `https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md#directives`

A.2 Karma & Jasmine ressourcer

1. `https://ericnish.io/blog/set-up-jasmine-and-karma-for-angularjs/`
2. `https://medium.com/frontend-fun/angular-unit-testing-jasmine-karma-step-by-step-`
3. `https://cli.angular.io/`

B Bilag

B.1 Data - Vægt grænse

XML format - input

```
<gml:featureMember>
  <TRA:bygvaerker_med_vaegtbegraensning
      fid="bygvaerker_med_vaegtbegraensning.fid--6100aeaf_161ad00e681_2455">
    <TRA:BYGVAERK_ID>2.8725506E7</TRA:BYGVAERK_ID>
    <TRA:BYGVAERKSART>Vejbrende bro</TRA:BYGVAERKSART>
    <TRA:BYGVAERKSBETEGN>UF af Vandl. Rdding </TRA:BYGVAERKSBETEGN>
    <TRA:BYGREGNR>20598.0</TRA:BYGREGNR>
    <TRA:STOERSTE_SPAENDVIDDE>0.8</TRA:STOERSTE_SPAENDVIDDE>
    <TRA:GEOMETRI>
      <gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
        <gml:coordinates xmlns:gml="http://www.opengis.net/gml" decimal="." cs="," ts="
            ">8.7113,56.2371</gml:coordinates>
      </gml:Point>
    </TRA:GEOMETRI>
  </TRA:bygvaerker_med_vaegtbegraensning>
</gml:featureMember>
```

GeoJson format - transformeret output

```
  {
  "type": "Feature",
  "properties": {
    "featureId": "2.8725506E7",
    "layerId": "layer_id_9B",
    "title": "UF af Vandl. Rdding  ",
    "regNo": "20598.0",
    "kind": "Vejbrende bro",
    "span": "0.8"
  },
  "geometry": {
    "type": "Point",
    "coordinates": [8.7113,
    56.2371]
  }
}
```

B.2 Branche uddybning

Under mødet med Vejdirektoratet, fortalte Thomas kort om nogle af de problemstillinger Transport branchen oplever og hvordan de arbejder udenom dem.

Frihøjder ved broer er en dynamisk parameter, da de konstant påvirkes og opdateres ved ny asfalt og forstærkning af broen, og målingerne sker ikke lige så ofte som opdateringerne, derfor offentliggøres brohøjderne med en sikkerhedsmargin på 20cm.

Som et konkurrence element i transport branchen, kan det tilføjes at hvis selskab A kender den aktuelle frihøjde og selskab B ikke gør, kan dette resultere i at selskab A kan give en bedre pris på et transport tilbud og/eller at selskab B skal ud og måle broen som der er tvivl om, denne måling, fortæller Thomas, kan foregå på mindre lovlige måder. Derudover kan en brohøjde variere fra spor til spor, hvis broen er buet.

B.3 MapController implementering

```
1  //START ### Mathias ###
2  //var self = this; //VM Capture
3
4  //Fields
5  $scope.BusinessRouteFactory = BusinessRouteFactory;
6  this.idMethod;
7  $scope.idMethods = ['alle','specifik','advanced'];
8  this.specificId;
9  $scope.buttonText = 'Test Inspicer Bro';
10 $scope.testActive = false;
11
12 var businessAddressFrom;
13 var businessAddressTo;
14
15 //Functions
16 $scope.removeAddress = removeAddress;
17 $scope.initBusinessRoute = initBusinessRoute;
18 $scope.validateFunction = validateFunction;
19 $scope.hideTextField = hideTextField;
20 $scope.testInspectBridge = testInspectBridge;
21
22 //############### Function implementations ###############
23 function removeAddress(tag){
24   console.log("removeAddress",tag);
25
26   //clear HTML textareas
27   if(tag === undefined){
28       $scope.businessAddressFrom.value = "";
29       $scope.businessAddressTo.value = "";
30   } else if(tag == 'from') {
31       $scope.businessAddressFrom.value = "";
32   } else if(tag == 'to') {
33       $scope.businessAddressTo.value = "";
34   }
35
36   $scope.BusinessRouteFactory.removeMarkers(tag);
37   $scope.BusinessRouteFactory.removeAddress(tag);
38   BusinessRouteFactory.clearRoute();
39
40 };
41
42 function initBusinessRoute(){
43   BusinessRouteFactory.init($scope.vdtrafficmap);
44   InspectFactory.init($scope.vdtrafficmap);
45
46
```

```
47    $scope.businessAddressFrom = document.getElementById("business-address-from");
48    $scope.businessAddressTo = document.getElementById("business-address-to");
49
50    BusinessRouteFactory.setupAutoComplete($scope.businessAddressFrom,"from");
51    BusinessRouteFactory.setupAutoComplete($scope.businessAddressTo,"to");
52  };
53
54  function validateFunction(){
55    var truckHeight = utilFactory.getType("truckheight") / 100;
56    var markers = $scope.markerClusterer.getMarkers(DataService.CLEARANCE_ID);
57    //TODO: Reset saved advancedBridges
58
59    //Case 1: Error: Ingen frihjder
60    if(markers.length < 1){
61      $window.alert('Der blev ikke fundet nogle frihjder');
62    }
63
64    //Case 2: Error: Specific valgt og ID undefined
65    else if(this.idMethod == 'specifik' && this.specificId === undefined) {
66      $window.alert('Indtast et ID');
67    }
68
69    //Case 3: Succes: specifik valgt og ID indtastet
70    else if(this.idMethod == 'specifik' && this.specificId != undefined) {
71      console.log('specificId',this.specificId);
72
73      var mark = BusinessRouteFactory.findMarkerById(markers,this.specificId);
74      BusinessRouteFactory.validateRoute(mark,truckHeight,0);
75    }
76
77    //Case 4: Succes: ALle valgt
78    else if(this.idMethod == 'alle') {
79      console.log('Alle valgt');
80      BusinessRouteFactory.validateRoute(markers,truckHeight,0);
81    }
82
83    //Case 5: Advanced
84    else if(this.idMethod == 'advanced'){
85      var mark = BusinessRouteFactory.findMarkerById(markers,10048);
86      mark.feature_properties = DataService.getAdvancedBridgeInformation();
87      BusinessRouteFactory.validateRoute(mark,truckHeight,0,'vejside2');
88    }
89
90  }; //testFunction
91
92  function hideTextField() {
93    var toReturn = !(this.idMethod == 'specifik');
94    return toReturn;
95  };
```

```
96
97  function testInspectBridge(){
98    $scope.testActive = !$scope.testActive;
99    if($scope.testActive){
100     $scope.buttonText = 'Reset Inspicer Bro';
101     InspectFactory.focus();
102   } else {
103     $scope.buttonText = 'Test Inspicer Bro';
104     InspectFactory.reset();
105   }
106 };
107 //############### Function implementations ###############
108 //END ### Mathias ###
```

Code snippet 19: MapController udsnit

B.4 BusinessRouteFactory

Forkortet version af klassen, se vedlagt projekt for fuld source kode.

```
1  angular
2      .module('mainApp.business')
3      .factory('BusinessRouteFactory',BusinessRouteFactory);
4
5  //Dependency injection
6  BusinessRouteFactory.$inject = ['$window', '$timeout', 'mapRouteFactory',
       'GoogleApiService', 'DataService'];
7
8
9  function BusinessRouteFactory($window, $timeout, mapRouteFactory, GoogleApiService,
       DataService){
10
11     //List of Fields
12     var ApiService = GoogleApiService;
13     var map;
14     var placesService;
15     var directionsService;
16     var travelMode = ApiService.getTravelMode('DRIVING');
17     var unitSystem = ApiService.getUnitSystem('METRIC');
18     var businessInfo = DataService.getBusinessInfo();
19     var directionsDisplay;
20
21     var IMG_A_URL = DataService.IMG_A_URL;
22     var IMG_B_URL = DataService.IMG_B_URL;
23
24     var setMarker = setMarker;
25
26     //Interface
27     var service = {
28         //Fields
29         map: map,
30         placesService: placesService,
31         businessInfo: businessInfo,
32         directionsService: directionsService,
33         travelMode: travelMode,
34         unitSystem: unitSystem,
35         directionsDisplay: directionsDisplay,
36
37         //Functions
38         setupAutoComplete: setupAutoComplete,
39         init: init,
40         setMarker: setMarker,
41         startRoute: startRoute,
42         initDirectionsDisplay: initDirectionsDisplay,
43         findMarkerById : findMarkerById,
```

```
44          validateRoute : validateRoute
45
46      };
47      return service;
48
49      //Function implementations
50      function init(mapParam){...};
51
52      function setupAutoComplete(input,tag){...};
53
54      function startRoute(fromPlaceId,toPlaceId){...};
55
56      function function setMarker(location,tag) {...};
57
58      function initDirectionsDisplay(){...};
59
60      function findMarkerById(markers,id){...};
61
62      function validateRoute(mark,height){...};
63
64      //Helpers
65      function function createPolylineFromLegs(legs){...};
66
67  };
```

B.5 Kernefunktionalitet steps

Step 1: At indtaste adresser

```
var businessAddressFrom;
var businessAddressTo;

function initBusinessRoute(){
    $scope.businessAddressFrom = document.getElementById("business-address-from");
    $scope.businessAddressTo = document.getElementById("business-address-to");

  businessRouteFactory.setupAutoComplete($scope.businessAddressFrom,"from");
  businessRouteFactory.setupAutoComplete($scope.businessAddressTo,"to");
}
```

Code snippet 20: MapController

```
   function setupAutoComplete(input,tag){
     var autocomplete_options = DataService.getAutoCompleteOptions();
     var autocomplete = ApiService.getAutoComplete(input, autocomplete_options);
     autocomplete.bindTo('bounds',map);

     autocomplete
       .addListener(
         'place_changed',
         (function() {
           console.log("place changed",tag);
           var place = autocomplete.getPlace();

           if (!place.geometry) {
               $window.alert("Den indtastede adresse kunne ikke findes");
               return;
           }

           if(tag === "from"){
             businessInfo.from.place = place;
             setMarker(businessInfo.from.place.geometry.location,tag);
           } else if (tag === "to"){
             businessInfo.to.place = place;
             setMarker(businessInfo.to.place.geometry.location,tag);
           }
           //Start route
           if(businessInfo.from.place && businessInfo.to.place){
             this.startRoute(businessInfo.from.place.place_id,
                 businessInfo.to.place.place_id);
           }

         }).bind(this)
```

```
31        );
32    };
```

Code snippet 21: businessRouteFactory

Step 2: request route

Efter indtastning af adresser requester den en rute fra Google API'et samt nogle alternative ruter

```
1
2     function startRoute(fromPlaceId,toPlaceId){
3       console.log("Starting route function");
4       var routeParams =
            DataService.getRouteParams(fromPlaceId,toPlaceId,travelMode,unitSystem);
5
6       console.log("id's",fromPlaceId,toPlaceId);
7
8       DataService.requestRoute(directionsService,routeParams, function(response,status){
9         console.log('response: ', response);
10
11        if(status === ApiService.getDirectionsStatus('OK')){
12          dataLayer.push({
13            'event': 'create-route',
14            'gtm.element': routeParams,
15            'origin' : businessInfo.from.place.formatted_address,
16            'destination' : businessInfo.to.place.formatted_address
17          });
18
19          directionsDisplay.setMap(map);
20
21          businessInfo.route.response = response;
22
23          removeMarkers();
24          directionsDisplay.setDirections(response);
25          businessInfo.route.index = 0;
26
27        } else {
28          $window.alert("Der kunne ikke findes en rute.");
29        }
30
31      }); //requestRoute
32
33    }; //startRoute function
```

Code snippet 22: businessRouteFactory

Step 3 og 4: validering af rute

```
function testFunction(){
  var height = utilFactory.getType("truckheight") / 100;
  var markers = $scope.markerClusterer.getMarkers(DataService.CLEARANCE_ID);

  //Case 1: Error: Ingen frihjder
  if(markers.length < 1){
    $window.alert('Der blev ikke fundet nogle frihjder');
  }

  //Case 2: Error: Specific valgt og ID undefined
  else if(this.idMethod == 'specifik' && this.specificId === undefined) {
    $window.alert('Indtast et ID');
  }

  //Case 3: Succes: specifik valgt og ID indtastet
  else if(this.idMethod == 'specifik' && this.specificId != undefined) {
    console.log('specificId',this.specificId);

    var mark = businessRouteFactory.findMarkerById(markers,this.specificId);
    businessRouteFactory.validateRoute(mark,height,0);
  }

  //Case 4: Succes: ALle valgt
  else if(this.idMethod == 'alle') {
    console.log('Alle valgt');
    businessRouteFactory.validateRoute(markers,height,0);
  }
}; //testFunction
```

Code snippet 23: MapController

```
 function validateRoute(markers,height,validateRouteIndex){

  if(markers === undefined) {
    console.log('error in validateRoute markers is undefined');
    return;
  }
  var legs = directionsDisplay.getDirections().routes[validateRouteIndex].legs;
  var currentRoute = createPolylineFromLegs(legs);

  var isTrouble;
  if(Array.isArray(markers)){
    //Filter alle non-trouble marks away
    var troubleBridges = markers.filter(mark =>
        isMarkCausingProblems(currentRoute,mark,height));
    isTrouble = troubleBridges.length > 0;
```

```
15        algorithmImpl(isTrouble,validateRouteIndex,markers,height);
16
17      } else { //Check route for 1 mark
18        isTrouble = isMarkCausingProblems(currentRoute,markers,height)
19        algorithmImpl(isTrouble,validateRouteIndex,markers,height);
20      }
21    }; //validateRoute
```

Code snippet 24: businessRouteFactory

```
1
2     function algorithmImpl(isTrouble,validateRouteIndex,markers,height){
3       if(isTrouble){
4         if(validateRouteIndex == 0){
5           $window.alert('Problemer med frihjder p ruten, finder en ny rute');
6         }
7
8         businessInfo.route.index = calcNewRouteIndex(validateRouteIndex);
9         if(typeof businessInfo.route.index === 'string'){
10          $window.alert(businessInfo.route.index);
11          return;
12        }
13        validateRoute(markers,height,businessInfo.route.index);
14      } else {
15        $window.alert('Problemfri rute fundet');
16        console.log('Route index used: ',validateRouteIndex);
17        directionsDisplay.setRouteIndex(validateRouteIndex);
18      }
19    }; //algorithmImpl
```

Code snippet 25: businessRouteFactory

B.6 Prototype UI

Billede serie i forbindelse med Test case 6.1.1

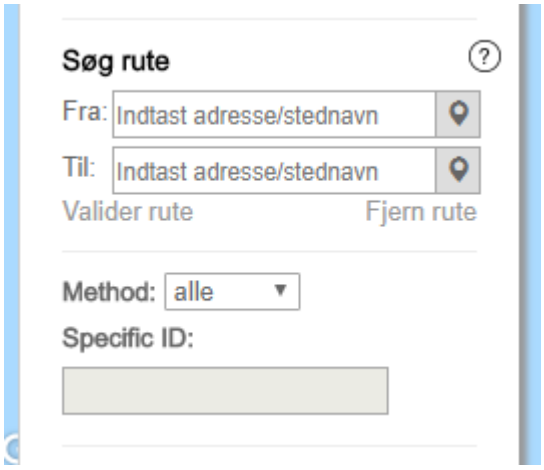

Figure 16: UI udviklet og arbejdet med til prototypen

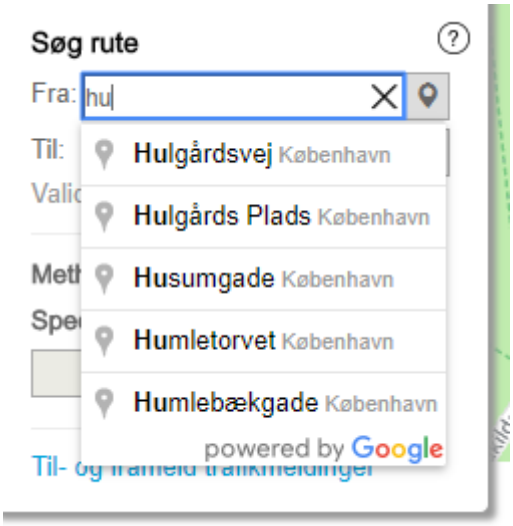

Figure 17: Autocomplete forslag fra Google

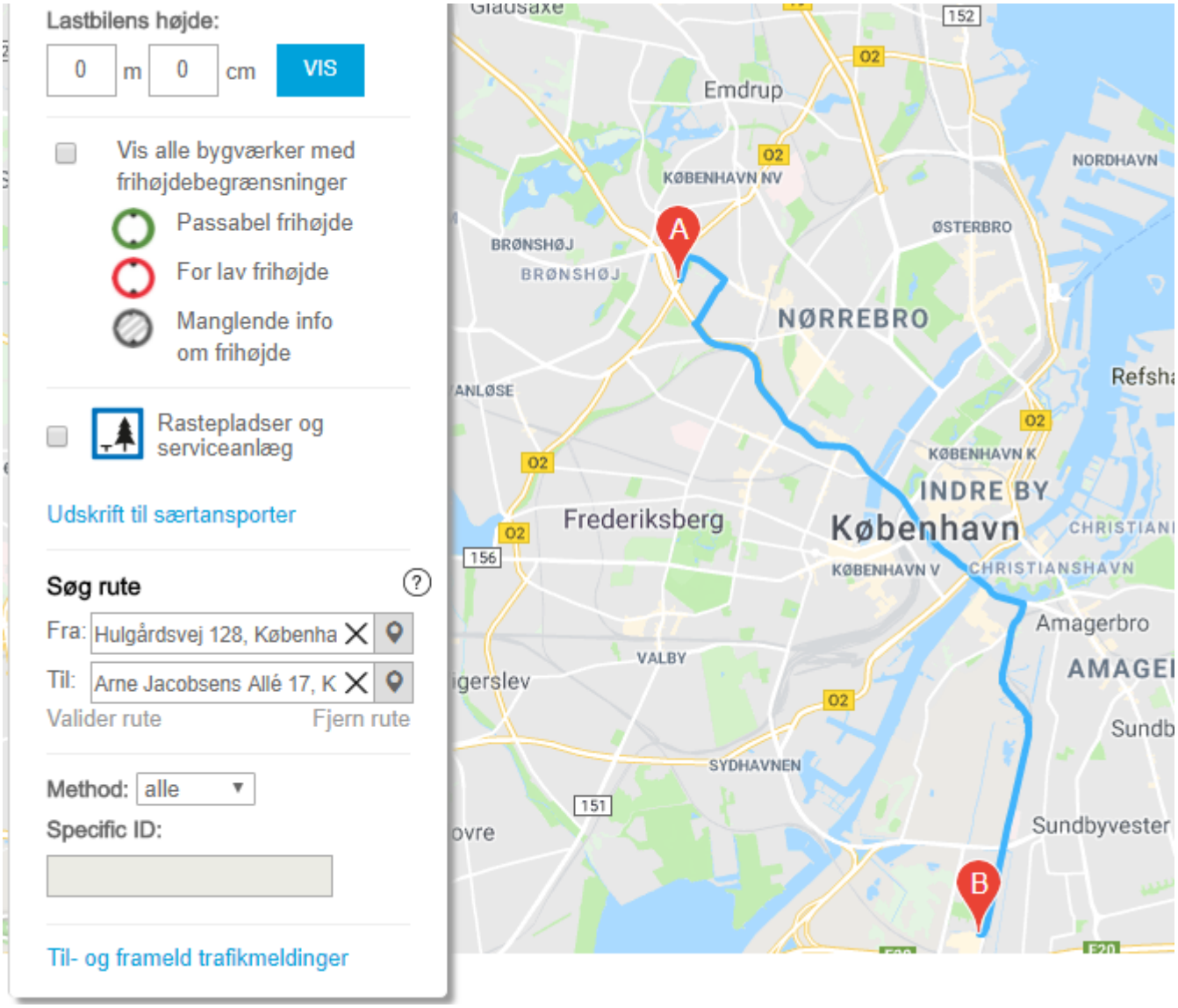

Figure 18: Fra og Til fundet - Rute på kortet

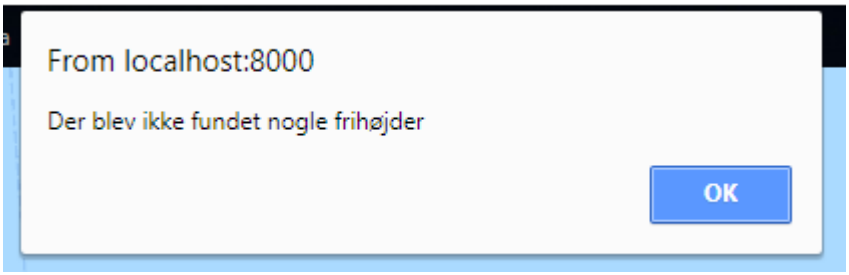

Figure 19: Fejlmeddelse til alternativ 1.a

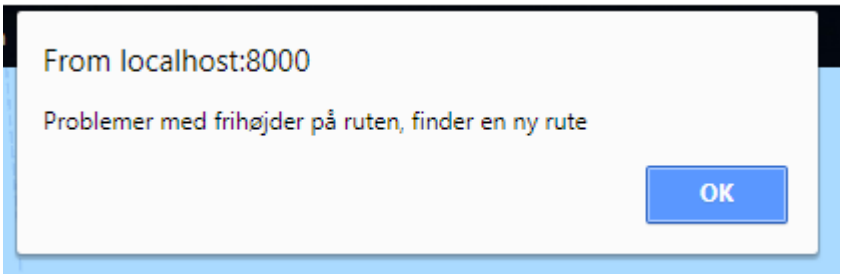

Figure 20: Fejlmeddelse til acceptkriterie 4

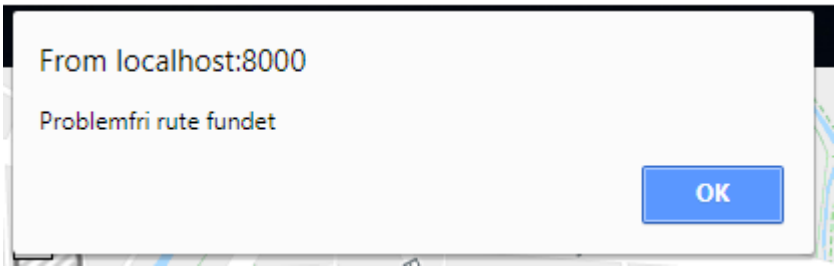

Figure 21: Succes besked til acceptkriterie 4

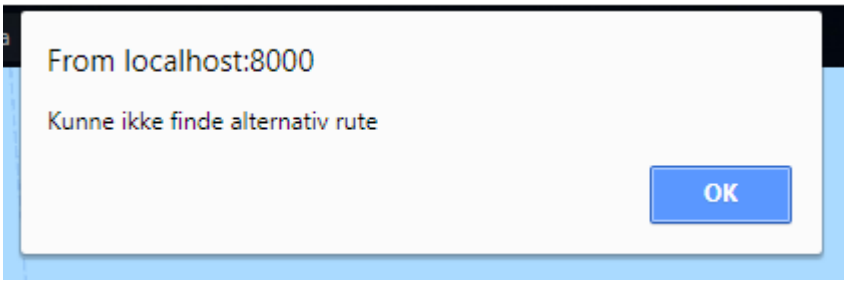

Figure 22: Fejlmeddelse til alternativ 4.a

Referencer

[1] Asana. Projekt management. URL: https://app.asana.com/.

[2] Atlassian. JIRA. URL: https://www.atlassian.com/software/jira.

[3] Google. Development Tools. URL: https://developers.google.com/web/tools/chrome-devtools/.

[4] Google. Google Maps API. URL: https://developers.google.com/maps/documentation/directions/intro?hl=en.

[5] Google. Issue tracker. URL: https://issuetracker.google.com/issues/35816642.

[6] Angular Team - Google. AngularJS Introduction. URL: https://docs.angularjs.org/guide/introduction.

[7] Angular Team - Google. Test runner. URL: https://karma-runner.github.io/2.0/index.html.

[8] Angular Team - Google. Unit Testing. URL: https://docs.angularjs.org/guide/unit-testing.

[9] GraphHopper. Open source directions API. URL: https://www.graphhopper.com/.

[10] Internet Engineering Task Force (IETF). GeoJSON Format. 2015 - 2016. URL: http://geojson.org/.

[11] Jasmine. Behavior Driven Development testing framework. URL: https://jasmine.github.io/index.html.

[12] Thomas Mark de Laine. Vejdirektorat kontakt. tmdl@vd.dk.

[13] MEAN.JS. URL: http://meanjs.org/.

[14] npm. URL: https://docs.npmjs.com/getting-started/what-is-npm.

[15] OpenRouteService. Open source directions API. URL: https://openrouteservice.org/.

[16] Tortoise SVN. Version Control System. URL: https://tortoisesvn.net/.

[17] Vejdirektoratet. DANBRO. URL: http://www.vejdirektoratet.dk/DA/vejsektor/ydelser/programmer/Danbro/Sider/default.aspx.