Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Sindre Østrem
Knut Grøstad

Salgsstøtte til CarAdmin

Bacheloroppgave i Dataingeniør
Veileder: Frode Haug
Mai 2019

NTNU
Kunnskap for en bedre verden

Sindre Østrem
Knut Grøstad

Salgsstøtte til CarAdmin

Bacheloroppgave i Dataingeniør
Veileder: Frode Haug
Mai 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Sammendrag av Bacheloroppgaven

Tittel:	**Salgstøtte til CarAdmin**
Dato:	20.05.2019
Deltakere:	Knut Grøstad Sindre Østrem
Veiledere:	Frode Haug
Oppdragsgiver:	Electric Time Car
Kontaktperson:	Dag Solhaug, dag.solhaug@eletrictimecar.com, 90101344
Nøkkelord:	Java, Spring, Angular, MariaDB, API, Fullstack
Antall sider:	39
Antall vedlegg:	5
Tilgjengelighet:	Åpen

Sammendrag:

Denne oppgaven beskriver et full-stack utviklingsprosjekt der vi lagde en web-applikasjon for salg av kjøretøy som en utvidelse av Electric Time Car sin løsning for flåtestyring. Dette innebar utviklingen av en databasestruktur, API og web-applikasjon. Teknologier som ble brukt var MariaDB for databasen, Java og Spring for API, samt Angular og Material Design for webutviklingen. Gjennom oppgaven oppnådde vi de læremålene vi hadde satt for oss selv. Desverre nådde vi ikke våre egne eller oppdragsiverens mål med web-applikasjonen. Dette kommer trolig av våre manglende kompetanse innen webutvikling ved prosjekt start og våre valg av rammeverk der vi kunne ha valgt verktøy som effektiviserer utviklingsprosessen.

Summary of Graduate Project

Title:	**Sale support for CarAdmin**
Date:	20.05.2019
Authors:	Knut Grøstad Sindre Østrem
Supervisor:	Frode Haug
Employer:	Electric Time Car
Contact Person:	Dag Solhaug, dag.solhaug@eletrictimecar.com, 90101344
Keywords:	Java, Spring, Angular, MariaDB, API, Fullstack
Pages:	39
Attachments:	5
Availability:	Open

Abstract: This thesis describes a full-stack development project where we were tasked to create a sale web application for vehicles. The web application was meant to serve as additional functionality for Electric Time Car's software for fleet management. This lead us to develop a database structure, API and a web application. To achieve this we used MariaDB for our database, Java with the Spring framework to develop our API and Angular with Angular material for web development. Although we achieved our learning goals, our web application did not meet the our own goals or the goals of our employer. We attribute this to our lack of previous experience with web development and that we might have been more efficient by choosing frameworks and technologies that better facilitates rapid development.

Forord

Vi vil gjerne takke Frode Haug for konstruktiv tilbakemelding, oppmuntring og tilgjengelighet. Vi vil også takke vår oppdragsiver, Dag Solhaug for å være villig til å tilpasse oppgaven etter våre ønsker og kompetanse i tillegg til å være svært imøtekommende og hyggelig.

Innhold

Figurer

Tabeller

Listings

1 Introduksjon

1.1 Fagområde

1.1.1 Fagområde

[1] Electric Time Car (ETC) er en softwareleverandør som holder til i Gjøvik. De har ett bilpark system med navn CarAdmin som hjelper bedrifter, i hovedsak offentlig sektor, med å administrere kjøretøy til sine ansatte. Kommuner er et godt eksempel. De har mange kjøretøy og forskjellige ansatte som benytter dem.

«CarAdmin er en helhetlig løsning for kjøretøyoppfølging av alle kjøretøy for privat og offentlig sektor. CarAdmin er tilpasset ulik bruk med forskjellige løsninger for alt fra kjøretøypooler til serviceoppfølging av gressklippere. Ønskes det mer automatisert oppfølging kan dette gjøres via GPS som i vår elektroniske kjørebok.» ([1])

Caradmin er delt inn i fire moduler:

- **Kjøretøyregisteret** holder oversikt over alle kjøretøy i Norge. Det inneholder diverse informasjon angående kjøretøyet, f.eks. km-stand, tid for neste service og lokasjon.
- **Kjøreboka** loggfører kjøreturer ved hjelp av GPS. Dette gjør det enkelt å foreta kjøregodtgjørelse.
- **Fellesbil** er utviklet for situasjoner der flere sjåfører deler et kjøretøy. Modulen gir mulighet til å låse nøkler i et nøkkelskap i forbindelse med uttak og innlevering av kjøretøyene. Fellesbil er med på å sikre god ansvarsfordeling mellom sjåførene.
- **Flåte** gir en oversikt over hvilke biler som befinner seg i nærheten, eller nærme et målpunkt. Det gis mulighet for å se kjørerute i kart. Denne modulen brukes i forbindelse med ulike oppdrag, og oppdragsgiver trenger å se hvem som kan ta på seg oppdraget.

1.1.2 Avgrensning

ETC hadde ønsker om å utvide og forbedre brukeropplevelsen på sitt system, CarAdmin. De ville gjøre det enklere for sine kunder å administrere salg og kjøp av kjøretøy. I den forbindelse fikk vi i oppgaven som skulle realisere dette ønsket. ETCs langtidsplan var å ha en komplett løsning som administrerer kjøretøyflåten for sine kunder og mente dette er en naturlig ekspansjon av sitt produkt.

Ettersom ETC sin kodebase var utviklet over 15 år ville et stort ressursbruk i oppgaven gått til å gjøre seg kjent med kodestrukturen i CarAdmin systemet. På bakgrunn av mangel på Java erfaring innad i gruppen og antall gruppemedlemmer, skulle vår løsning utvikles adskilt fra CarAdmin systemet. Dette betød at vår løsning skulle være uavhengig og separert fra CarAdmin systemet. En annen konsekvens av mangel på java-kompentanse og kompetanse innen webutvikling innad i gruppen ble oppgaven begrenset til kun en salgstøttemodul.

[1] Fagområdet er et samarbeid mellom gruppe 4 og 5 grunnet samme oppdragsgiver.

1.1.3 Oppgavebeskrivelse

Oppgaven gikk ut på å lage en webapplikasjon for salgsstøttemodul for ETC sitt CarAdmin system. Systemet skulle være en markedsplass for CarAdmin sine kunders kjøretøy. Kjøretøyene skulle kunne markedsføres til alle interesserte ved hjelp av en annonse som legges ut i systemet. Interesserte kjøpere skulle kunne søke etter og finne relevante annonser. Annonsene skulle inneholde informasjon som beskriver kjøretøyet og informasjon om selgeren. Kunder skulle kunne administrere egne biler og annonser. En administrator rolle skulle kunne administrere kunder. Det skulle undersøkes om mulighetene for at annonser i salgsmodulen kan legges direkte inn i FINN.no systemet.

1.2 Formål

Formålet med prosjektet var å tilføye det eksisterende CarAdmin systemet flere funksjoner og utvide systemets bruksområder. Hensikten ved dette var å øke interessen i produktet til potensielle kunder og øke eksisterende kunders tilfredsstillelse med produktet. For vår del var det hovedsakelig muligheten til å få erfaring med fullstack-utvikling som gjorde oppgaven attraktiv. Spesielt med tanke på å utvikle i Java som vi mener vil komme godt med i fremtidig arbeidsliv.

1.2.1 Effektmål

- Ekspandere CarAdmin sitt bruksområder og funksjonalitet med salg og kjøp av kjøretøy inn i kundenes bilpark.
- Gjøre det enklere for kunder å selge og kjøpe nye kjøretøy. Redusere kundenes avhengighet på leasingselskaper for administrering av kjøp og salg i kjøretøyflåten.
- Antall kommuner som abonnerer på CarAdmin ville øke med 20% i løpet av 3 år som følge av løsningen.

1.2.2 Resultatmål

- Grensesnittet på webapplikasjonen skulle følge retningslinjer for design beskrevet av Angular Material [2].
- Webapplikasjonen skulle inneha de funksjonalitetene som var ønsket av ETC.
- API til systemet skulle være dokumentert og kunne brukes av ETC.
- Med vår webapplikasjon ville CarAdmin systemet gå mot å dekke alle behov ved flåteadministrasjon.

1.2.3 Læringsmål

- Erfare systemutviklingsprosess i praksis med et prosjekt som var relevant i forhold til det fremtidige arbeidslivet.
- Få erfaring med backend-utvikling i Java med Mariadb og Tomcat og andre verktøy.
- Få erfaring med frontend-utvikling i Angular7/Typescript.
- Utvikle løsninger ved bruk av ekstern API og utvikle vår egen API.

1.3 Hvorfor valgte vi denne oppgaven

Oppgaven ble beskrevet som en fullstack-utviklingsoppgave med hovedsakelig Java som utviklingsmiljø. Dette gjorde oppgaven interessant for oss av flere grunner. For det første var vi mest interessert i mer praktisk oppgave til forskjell fra en mer teoretisk eller analytisk oppgave. En annen grunn var at Java er et relevant utviklingsmiljø som er attraktivt på jobbmarkedet. En tredje

grunn var at oppgaven hadde mange momenter ved seg hvor vi hadde liten til ingen erfaring i og dermed tenkte vi at oppgaven ville være lærerik og spennende.

1.4 Målgruppe

1.4.1 Målgruppe for produkt

Produktet var ment til å brukes av kunder av CarAdmin og personer som var interessert i å kjøpe kjøretøy.

1.4.2 Målgruppe for rapport

Rapporten kunne være av interesse for personer som skal utvikle noe lignende eller er interessert i utviklings-stacken vi har brukt. Den kan også være til nytte for personer som er interessert i å lære seg web-utvikling. Rapporten skulle være forståelig for alle med basiskunnskap innen systemutvikling og webutvikling. Fremmedord og spesielle uttrykk er forklart i en egen ordbok i vedlegg A

1.5 Egen bakgrunn og kompetanse

Gruppen som stod bak prosjektet bestod av Knut Grøstad og Sindre Østrem. Vi gikk i samme klasse på dataingeniørlinjen ved NTNU Gjøvik. Utover de obligatoriske emner har vi begge hatt «Programvaresikkerhet». Knut Grøstad har i tillegg hatt «Introduksjon til Kunstig intelligens». Vi stilte begge svakt når det gjelder utvikling innenfor webløsninger. Ingen av oss har utviklet programvare i Java tidligere og vi hadde begge liten eller ingen erfaring med utvikling i Typescript/Javascript. Knut Grøstad hadde noe erfaring i backend-utvikling i PHP og Rust.

1.5.1 Hva måtte læres?

I dette prosjektet var det mye som ble nytt for begge gruppemedlemmer. Ettersom vår bakgrunn var mer eller mindre lik så gjelder følgende for hele gruppen. Det virker ikke sånn i retrospekt men det var mye å sette seg inn i da vi tok i bruk Java og Angular.

Java

- Basis kunnskaper i Java som for eksempel deklarasjoner og arv.
- Lære seg nye konsepter og bruken av for eksempel «Beans» (A), «Servlets» (A) «Annotations» (A)
- Største delen var bruken av Spring rammeverket og dets mange moduler og tilsynelatende hundrevis av metoder og funksjonaliteter.
- Utvikle en «RESTful API» (A)

Angular 7

- Utvikling av en klient generelt.
- Bruke «templates» effektivt slik at de kunne brukes flere plasser og ikke er en engangsvare.
- Oppsett, organisering av løsningen.
- Bruk av en «CLI» for enklere produsering av nødvendige filer og struktur.
- Toveis databinding og hvordan de brukes.
- Hvordan man setter opp en forespørsel mot en server.
- Skrive TypeScript, som er ganske likt JavaScript.

Angular Material

- Hvordan bruke Material i samsvar med Angular.

- En del Material spesifikke «tags».
- Bruken av skjema spesielt modulen «NgForms».
- Bruke toveis databinding.

1.6 Rammer

1.6.1 Fremdriftsplan

Bacheloroppgaven hadde oppstart 10.01.19 og avsluttet 20.05.19 da endelig bachelorrapport skulle leveres.
I løpet av oppgaven skulle veileder motta tre statusrapporter på følgende datoer:

- 20.02
- 01.04
- 01.05

Produktet skulle bli lervert til ETC normalt sammen med bacheloroppgaven hvis ikke annet er avtalt.

1.6.2 Teknologisk

ETC satt krav til bruk av JavaEE, Tomcat og MariaDB. Når det gjaldt på klient delen så stod vi fritt til å velge rammeverk selv.

1.7 Prosjektorganisering

I vår gruppe på to delte vi mye av ansvaret for gjennomføringen av bacheloroppgaven. Dette inkluderte ansvar om å oppholde tidsfrister, holde avtaler, aktiv deltagelse og opplæring i teknologi som måtte være relevant for gjennomførelse av oppgaven.

Vi fordelte det slik at Knut Grøstad var prosjektleder med kommunikasjonansvar og Sindre Østrem var sekretær med ansvar for dokumentasjon av møter og beslutninger. Utenom dette ble arbeidsoppgaver fordelt etter hvert gruppemedlems preferanser om oppgaver som oppsto. Vår veileder, Frode Haug representerte NTNU sine interesser og var til hjelp med råd og veiledning av arbeidsprosessen for å nå de målene som var forventet av NTNU. Vår oppdragsgiver ETC, ble representert av Dag Solhaug som var dagligleder for ETC. Dag skulle fremme ETC sine interesser i oppgaven og kom med kravspesifikasjoner, konstruktive tilbakemeldinger og teknisk rådgivning/hjelp.

1.8 Rapporten

1.8.1 Rapportstruktur

Introduksjon Det første kapittelet gir leseren en innføring i fagfeltet som avgrenses videre til en konkret oppgavebeskrivelse. Her får leseren informasjon om ulike rammer og mål for oppgaven og blir informert om praktiske aspekter ved rapporten.

Prosesss I dette kapittelet blir leseren informert om hvilken arbeidsmetode som ble fulgt for å best mulig løse oppgaven og nå de målene som ble satt. Det blir argumentert for hvorfor vi tror denne metoden er den mest optimale i forhold til de rammene vi måtte forholde oss til.

Kravspesifikasjon Her blir leseren informert om hvilke krav det ble satt til web-applikasjonen av oppdragsiveren i samarbeid med oss. Kravene blir beskrevet i form av use case diagrammer og tabeller.

Design og arkitektur Kapittelet presenterer arkitekturen for web-applikasjonen i overordnet forstand og i detalj. Leseren blir presentert hvordan datastrukturen er satt opp og logikken bak dette. Databasestrukturen blir forklart og presentert i form av en figur.

Implementasjon Leseren får en innføring i hvordan web-applikasjonen er implementert i form av kodeeksempler i henhold til arkitektur og design som er valgt.

Diskusjon og konklusjon I dette kapittelet diskuterer vi resultatene i implementasjonen i forhold til de målene vi har satt oss. Vi diskuterer valg vi har tatt i løpet av oppgaven og konsekvensene av disse. Videre diskuterer vi oppgaven, evaluerer eget arbeid og videre arbeid. Til slutt konkluderer vi med de lærdommerer og erfaringer som oppgaven har ført til og oppnådd resultat iforhold til oppgavebeskrivelsen.

2 Scrumban prosessen

2.1 Hvorfor vi valgte Scrumban

Ved valg av utviklingsmodell vurderte vi både sekvensielle og smidige modeller. Problemet med en sekvensiell modell som fossefall er at kravspesifikasjon for utviklingsprosjektet må være klare før utviklingen begynner. I møter med oppdragsgiver var det tydelig at ny funksjonalitet og nye ideer kunne oppstå under utviklingen. Det var også åpent for og ønskelig at vi, utviklerene kunne komme med våre egne ideer iløpet av utviklingsprosessen. Vi var avhengig av at oppdragsiver ga oss tilbakemelding av arbeidet vi har gjort slik at vi kunne forbedre produktet. Dette var viktig for oss på grunn av mangel på erfaring med webutvikling. I tillegg var det potensielt flere funksjonaliteter i prosjektet som krevde avtaler med eksterne aktører som ikke var på plass ved oppstart av prosjektet. Dette gjorde at en sekvensiell modell ble feil for vårt prosjekt. Vår modell måtte være fleksibel og kunne brukes i en dynamisk utviklingsprosess. På grunn av denne vurderingen har vi sett på smidige modeller som Scrum og Scrumban som mulig utviklingsmodell for vårt prosjekt.

Scrum er en smidig utviklingsmodell som deler utviklingsprosessen i mindre iterative deler, kalt sprinter. I en sprint har man daglige møter og for enhver sprint må det planlegges. Scrumban er en blanding av metodene Scrum og Kanban. Den kombinerer de elementære delene av Scrum og fleksibiliteten til Kanban. Scrumban har en kontinuerlig utviklingsprosess med korte perioder for planlegging og lengre perioder for utgivelser.

Scrum er en litt mer restriktiv metode en Scrumban. Alle oppgaver er ofte bundet til en frist, men i Kanban og Scrumban bruker man en pull metode ettersom oppgaver blir ferdigstilt. En begrensning for antall oppgaver i utvikling (Work-In-Progress) reduserer risikoen for «for mange baller i luften» og sørger for at oppgaver blir ferdigstilt før man begynner på nye oppgaver. I Scrum tilbakestiller man prosesstavlen etter hver sprint. Etter planleggingsmøte legger man inn nye oppgaver i backlog. Man kan ikke legge inn nye oppgaver i en sprint som er i gang. I Scrumban har man «on-demand» planlegging dersom backloggen er tom eller er under et bestemt antall. Man vil da kunne fylle opp To-do/backlog med nye oppgaver som pushes fra produktbacklog. Det er ikke et krav til «planning-poker» og tidsestimering i Scrumban.

I vårt prosjekt var en kontinuerlig utviklingprosess foretrukket for å kutte ned på tid brukt på estimering og scrum-ritualer. Da vi var et team på to ville vi helst bruke så mye tid på selve utviklingen som mulig. Tids-estimering av oppgaver var i vårt tilfelle vanskelig ettersom vi manglet erfaring på å utvikle større prosjekter. Derfor var «planning on demand» foretrukket i vårt tilfelle. I tillegg er var vi ikke godt nok kjent med vårt valgte utviklings-stack til å gjøre gode vurderinger med tanke på tidsestimering.

Scrum er ofte beskrevet som best for grupper på tre til ni eksperter[3], Scrumban er et mer fleksibelt rammeverk og passer best egnet for rask utvikling som er typisk for oppstartsbedrifter[4]. Alt i alt ga Scrumban vår gruppe ett godt rammeverk for hektisk systemutvikling med rutiner for planlegging og forbedring.

2.2 Hvordan vi brukte Scrumban

I vårt prosjekt brukte vi Trello som er en virtuelt planleggingstavle. Tavlen bestod av følgende kolonner:

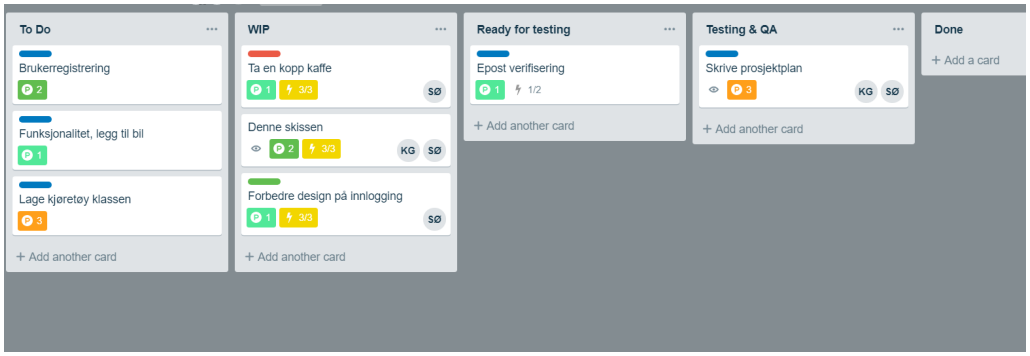
- Product backlog
  Inneholdte funksjonaliteter som så langt er planlagt av oppdragsgiver i samarbeid med oss.
- To-do list
  Inneholdte oppgaver, en enkel eller en del av en funksjonalitet. Oppgavene ble planlagt etter et planleggingmøte på bakgrunn av innholdet i «product backlog». Oppgaver ble tildelt en prioritering fra en til tre hvor tre er viktigst.
- Work-in-progress (WIP)
  Ferdig planlagt oppgave ble trukket inn i WIP. Ble valgt av gruppemedlem på bakgrunn av preferanse. Kolonnen hadde en begrensning på fire oppgaver.
- Klar-for-testing
  Oppgavene som var ferdig utviklet ble trukket fra WIP og var klargjort for testing. Kolonnen hadde begrensning på fire oppgaver.
- Testing og kvalitetsikring
  Nye moduler ble testet med funksjonelle tester og «static-code analyzer». Om modulen måtte forbedres flyttes den tilbake til to-do listen.
- Ferdig
  Kolonnen inneholdte ferdigstilte og testet funksjonalitet.

Figur 1 viser eksempelbruk for tavlen

Når To-do listen blir tom betyr dette at nye oppgaver måtte planlegges for at to-do listen skulle fylles opp igjen. Oppgavene skulle skilles i forskjellige typer oppgaver med fargekoder:

- Oppdateringer: Rød
- Forbedringer: Grønn
- Nye funksjonaliteter: Blå

Ferdigstilte moduler ble presentert og vurdert ca. hver 14. dag i møte med oppdragsiver (ETC).



Figur 1: Eksempel bruk for trello tavle

3 Kravspesifikasjon

Oppdragsgiveren ga oss en rekke krav til produktet vi skulle utvikle. Vi utarbeidet en kravspesifikasjon ut ifra disse kravene. Kravspesifikasjonen ble delvis skrevet i retrospekt eller underveis i henhold til utviklingsmodellen vi valgte.

3.1 Overordnede krav

- Webapplikasjonen skulle være på norsk.
- skulle kunne brukes på PC, nettbrett og mobil.

3.2 Funksjonelle krav

De funksjonelle kravene som ble gitt av oppdragsgiver ble igjennom oppgavens varighet endret, da funksjoner ble lagt til eller fjernet. De funksjonelle kravene ble i stor grad påvirket og/eller foreslått av oss.

- Kunder skulle kunne legge inn, endre på og slette data om et kjøretøy i systemet på vegne av sin bedrift.
- Kunder skulle kunne opprette, slette og endre på en annonse av et kjøretøy på vegne av sin bedrift.
- Potensielle kjøpere skulle kunne søke etter annonser og se annonser hvor de kan finne informasjon om kjøretøyet og kontaktinformasjon om selger.
- Kunder og bedrifter skulle kunne endre på sine egne opplysninger.
- Kunder kunne sende en opprettet annonse til FINN.no slik at det opprettes samme annonse på deres system automatisk.
- En administrator skulle kunne opprette, endre og slette brukere og bedrifter.

3.3 Ikke-funksjonelle krav

3.3.1 Tilgjengelighet

Webapplikasjonen skulle være tilgjengelig for alle til enhver tid utenom planlagt nedetid. Anonyme brukere har kun tilgang til å søke og se på annonser. Kunder har tilgang til hele systemet bortsett fra administrator funksjoner og må opprettes av en administrator.

3.3.2 Effektivitet

Den ferdig utviklede webapplikasjonen skulle effektivisere og forenkle prosessen ved salg av kundenes kjøretøy. Den skulle utvide CarAdmin systemet og gjøre produktet mer attraktivt for potensielle kunder.

3.3.3 Personvern og sikkerhet

Systemet skulle være utviklet etter prinsipper i programvaresikkerhet beskrevet av Leblanc Howard i boken «Writing Secure Code»[5]. Vi nevner de viktigste:

- Begrense angreps overflaten ved for eksempel data validering.
- Etablere sikre standarder ved for eksempel kreve kompleksitet til passord.

- Sikre at at brukergrupper kun har den tilgangen de trenger.
- Systemet feiler på en sikker måte.

Et annet krav er at systemet skulle følge datatilsynets råd angående krav til personvern[6].

3.3.4 Brukergrensesnitt

Systemet skulle være enkel å bruke. skulle være lik utforming som feks FINN.no så overgangen for CarAdmin kunder skulle være enkel.

3.3.5 Dokumentasjon og kode

Dokumentasjonen på koden skulle være på norsk. Navn på variabler, klasser og funksjoner i koden skulle være på engelsk.

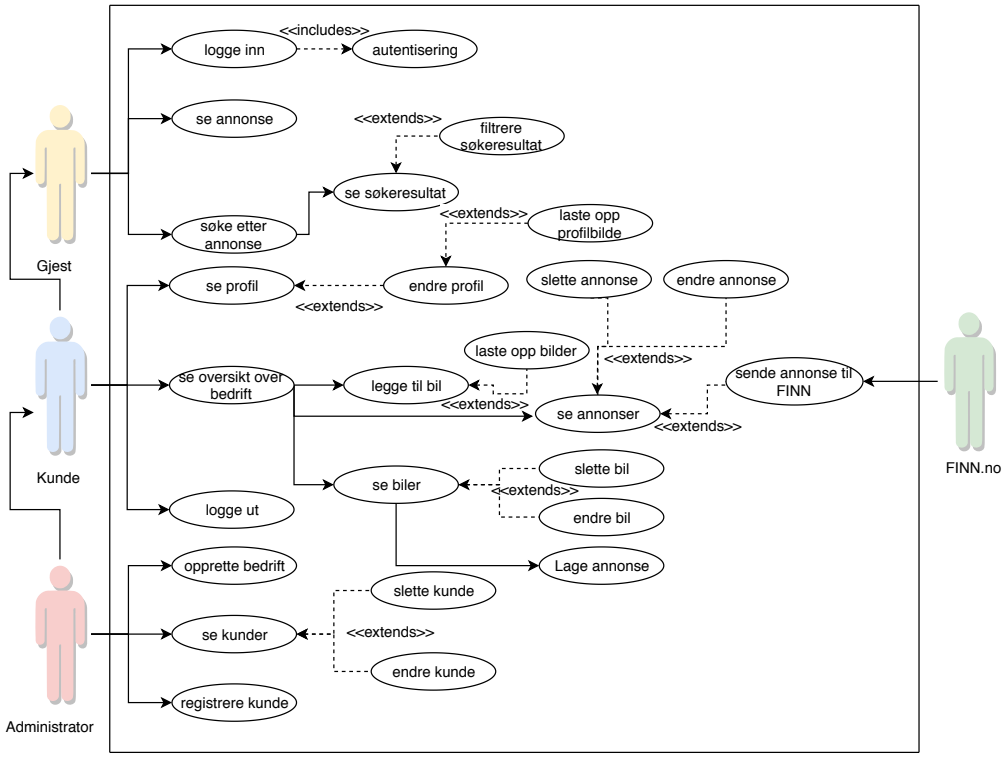
3.4 Database

Et krav fra oppdragsgiver er at databasen er av type MariaDB som er en fork av MySql. Databasen skulle følge standarden innen utforming av relasjonell database [7]. Databasen skulle bruke lignende datamedlemmer som databasen i CarAdmin systemet beskrevet i vedlegg E.2.

3.5 Use case

Figur 2 beskriver funksjonaliteten i systemet ved hjelp av aktører og use case.

3.5.1 Use case-diagram



Figur 2: Use case diagram for webapplikasjonen

3.5.2 Høynivå use case

Aktøren «bruker» er en fellesbetegnelse på brukerene gjest, kunde og administrator. I tabellene 1 til 9 forklarer vi use case på høy nivå.

Use case	Søke etter annonse
Aktør	Bruker
Beskrivelse	Brukeren skulle kunne søke etter tittel på annonsen eller på modell og bilmerke i et søkefelt. Brukeren vil så kunne initiere ett søk som vil gi brukeren ett søkeresultat

Tabell 1: Høynivå use case for å søke etter en annonse

Use case	Se annonse
Aktør	Bruker
Beskrivelse	Bruker har klikket seg inn på en annonse og blir møtt av: • Annonse informasjon. • Informasjon om bilen. • Informasjon om selger.

Tabell 2: Høynivå use case for å se annonse

Use case	Logge inn
Aktør	Bruker
Beskrivelse	Brukere har mulighet til å logge inn dersom de oppgir et gyldig brukernavn og passord som er registrert i systemet :

Tabell 3: Høynivå use case for å logge inn i systemet

Use case	Se oversikt over bedrift
Aktør	Kunde
Beskrivelse	Kunden navigerer seg til bedrifts-siden og blir møtt av: • Oversikt over bedriftens biler og annonser • Funksjon for å legge til, slette eller endre biler. • Funksjon for å legge til, slette eller endre annonse. • Funksjon for å sende annonse til FINN.no

Tabell 4: Høynivå use case for å se oversikt over bedrift

Use case	Se profil
Aktør	Kunde
Beskrivelse	Kunder skulle kunne gå inn på sin egen profil hvor de finner data som lagres om se selv. Her her de videre mulighet til å endre på egen bruker data om det er ønskelig.

Tabell 5: Høynivå use case for å se profil

Use case	Legge til bil
Aktør	Kunde
Beskrivelse	Kunder skulle kunne legge til en ny bil. Kunden ville få opp en oversikt over informasjon som må fylles inn for å legge inn en ny bil i systemet. Når kunden har lagt inn all data som kreves ville kunden kunne bekrefte opprettelsen av ny bil. Om et problem oppstår underveis ville kunden bli informert om dette.

Tabell 6: Høynivå use case for å legge inn ny bil

Use case	Se biler
Aktør	Kunde
Beskrivelse	Kunder skal kunne se en oversikt over alle bilene som bedriften har lagt inn i systemet. Herfra kunne kundene innad i bedriften ha muligheten til å gjøre endringer på en bestemt bil eller å fjerne en bestemt bil. Herfra kunne kunden også lage en ny annonse basert på en bestemt bil for å legge denne ut for salg.

Tabell 7: Høynivå use case for å se biler

Use case	Se annonser
Aktør	Kunde
Beskrivelse	Kunder skulle kunne se en oversikt over alle annonser som bedriften har lagt inn i systemet. Herfra kunne kundene innad i bedriften ha muligheten til å gjøre endringer på en bestemt annonse eller å fjerne en bestemt annonse. Herfra kan kunden også velge å sende annonsen til FINN.no slik at en annonse blir opprettet på FINN.no platformen på brukeren knyttet til bedriften.

Tabell 8: Høynivå use case for å se annonser

Use case	Se kunder
Aktør	administrator
Beskrivelse	Administrator skal kunne få se en oversikt over alle kunder som er registrert i systemet. Herfra har administratoren mulighet til å deaktivere enkelte kunder eller endre data på enkelte kunder.

Tabell 9: Høynivå use case for å se kunder

3.5.3 Lavnivå use case

Vi har gått i dybden på utvalgte use case i tabellene 10 til 11 .

Use case	Filtrere søkeresultat
Aktør	Bruker
Pre betingelser	Gitt ett søkeord i søkefeltet og mottatt ett søkeresultat
Post betingelser	Brukeren blir vist et filtrert søkeresultat.
Beskrivelse	Bruker har mulighet til å filtrere søkeresultatet etter flere kriterier som feks farge på bil, motortype (elbil, hybrid, diesel, bensin), pris, osv.
Detaljert hendelsesforløp	1. Ved siden av skoleresultatet vil det være en filter funksjon hvor man kan velge ulike filter som skal påvirke hvilke annonser som vises frem. 2. Brukeren velger ett eller flere filter slik at søkeresultatet endrer seg for å tilpasse filteret. 3. Brukeren ender opp med et filtrert søkeresultat.

Tabell 10: Lav nivå use case for å se filtrere søkeresultat

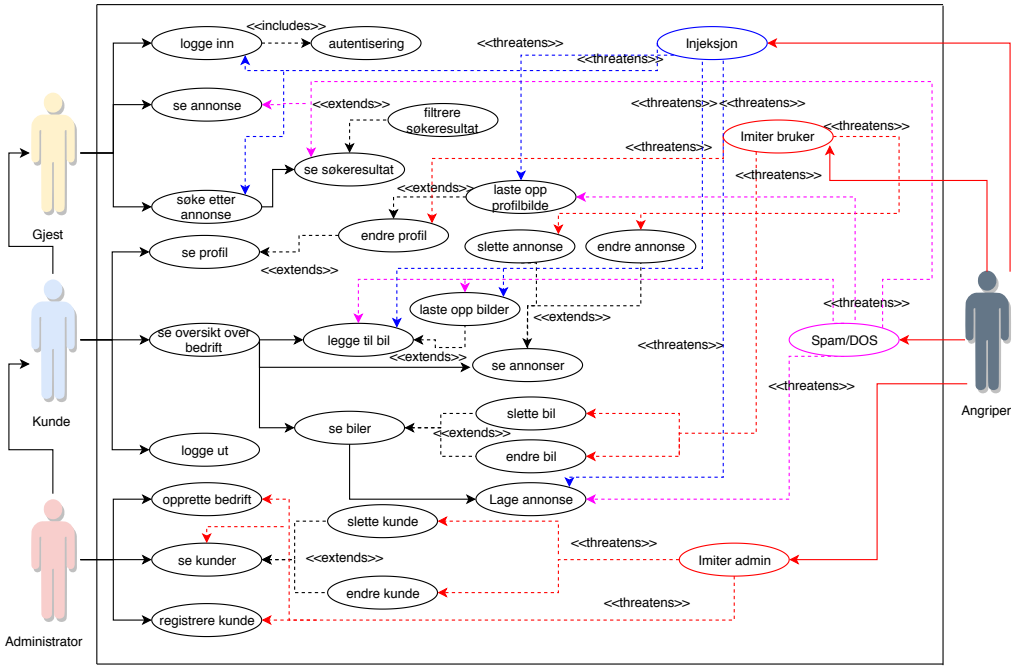
Use case	Sende annonse til FINN
Aktør	Kunde
Pre betingelser	En annonse for bilen er allerede opprettet og data knyttet til verifisering i FINN.no systemet er gitt.
Post betingelser	Kunden får bekreftet at annonsen er lagt til hos FINN.no
Beskrivelse	Kunder med egen bedriftsavtale med FINN.no har muligheten til å sende annonser direkte over til FINN.no markedsplassen og legge biler ut for salg.
Detaljert hendelsesforløp	1. Kunden velger å sende annonsen over til FINN.no 2. Ett skjema opprettes og sendes til FINN.no hvor det verifiseres 3. Kunden får tilbakemelding på om et problem oppstod eller om det lykkes med å opprette annonsen.

Tabell 11: Lav nivå use case for å se sende annonse til FINN.no

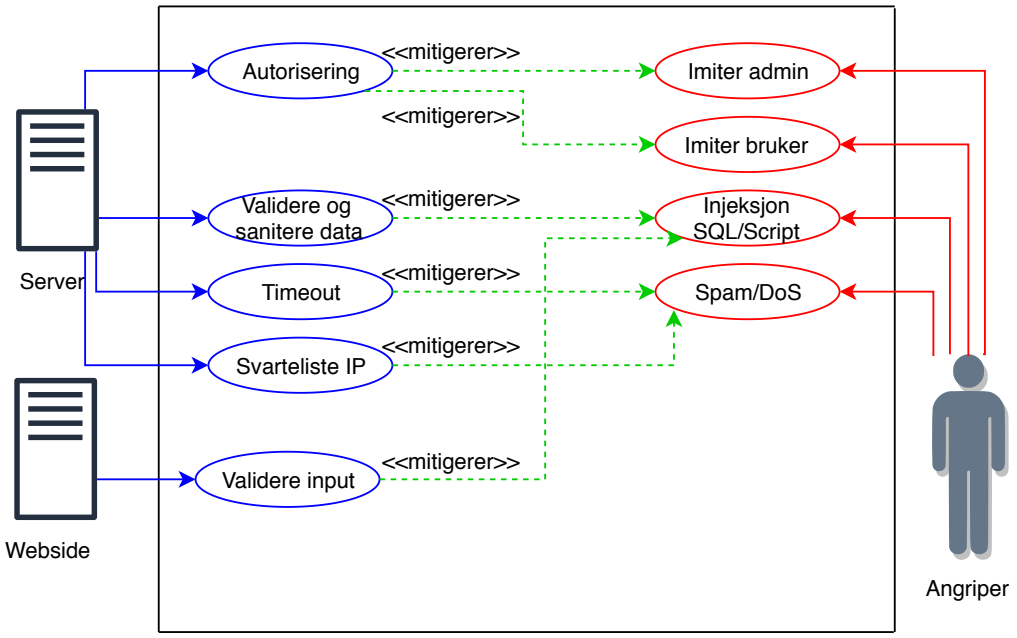
3.6 Misuse case

Et misuse case diagram brukes for sette seg inn i og oppdage mulige sårbarheter løsningen kan ha. Her brukte vi aktøren «angriper» for personen som prøver å gjøre ugagn i løsningen vår. Det første diagrammet 3 beskriver sårbarheter og det andre 4 beskriver tiltak som kan forhindre angrep.

3.6.1 Misuse case diagrammer



Figur 3: Misuse case diagram for webapplikasjonen



Figur 4: Misuse case diagram for mitigasjon av trusler

3.6.2 Misuse case forklaringer

I tabellene 12 til 15 gikk vi mer i detalj på enkelte misuse case for å kartlegge sårbarheter og potensielle mitigasjoner som kunne implemeteres.

Misuse case	Imiter bruker
Aktør	Angriper
Beskrivelse	Angriperen prøver å få tilgang til det en spesifikk kunde har tilgang til. Dette gir angriperen kundens annonser og lagrede data og kontroll over de.

Mulige sårbarheter	• Endre profilen
	• Kjøretøy
	  ◦ Endre kjøretøy
	  ◦ Slette kjøretøy
	• Annonser
	  ◦ Endre annonse
	  ◦ Slette annonse

Mitigasjon	Autorisering av brukeren

Tabell 12: Misuse case for imitering av kunder

Misuse case	Imiter admin
Aktør	Angriper
Beskrivelse	Angriper prøver å få tilgang til en admin konto med høye privileger i systemet. Med disse privilegiene risikeres hele systemet. Ikke bare kan angriperen ødelegge hele systemet men også se mye persondata.

Mulige sårbarheter	• Kunder
	  ◦ Se alle kunder
	  ◦ Slette en kunde
	  ◦ Endre på en kunde
	  ◦ Registrere nye kunder
	  · Registrere ny admin konto
	• Opprette bedrifter

Mitigasjon	Autorisering av brukeren

Tabell 13: Misuse case for imitering av admin

Misuse case	Injeksjon
Aktør	Angriper
Beskrivelse	Ved alle felter vi godtar input, kan en angriper forsøke seg på kode injeksjon på siden. Ved for eksempel SQL-injisering for å prøve å data ut av databasen vår, også tilgang til å slette/endre data, lage egne admin kontoer osv.

Mulige sårbarheter	• Innloggingsskjema
	• Annonsesøk
	• Bildeopplastning
	  ◦ Profilbilde
	  ◦ Kjøretøybilder
	• Lage kjøretøy objekt skjema
	• Lage annonse skjema

Mitigasjon	• Validering og sanitering av input:
	  ◦ Valider at input er hva som forventes.
	  ◦ Nekte ulovlige tegn som kan brukes ved injisering.
	  ◦ Sanitere alle forespørsler til databasen.
	  · Parametrisere dataene.

Tabell 14: Misuse case for injeksjon ved alle input

Misuse case	Spam/DOS
Aktør	Angriper
Beskrivelse	DOS (Denial of Service): Oversvømme serveren med forespørsler slik at siden vår opplever svekket funksjon eller i verste fall kræsj på serveren, som legger ned alle tjenester på siden.

Mulige sårbarheter	• Lage annonse
	• Legge til nye kjøretøy
	• Laste opp bilder
	  ◦ Profilbilde
	  ◦ Kjøretøybilder
	• Søke annonser
	• Se annonser

Mitigasjon	• Timeout ved mange forespørsler. • Svarteliste problematiske IP-er

Tabell 15: Misuse case for spam

4 Analyse

4.1 Valg av teknologier

Kravene som ble satt av ETC når det kommer til anvendelse av teknologier var som følger:

- JavaEE i serveren.
- Databasen skal være i MariaDB.
- Webapplikasjonen skal kjøres på en TomCat server.

Gitt disse kravene ga det oss muligheten til fritt velge hvilke rammeverk vi mente passet best til prosjektet og som vi var interessert i å bruke. Ved valg av disse teknologiene la vi vekt på følgende kriterier:

- Rammeverket hadde elementer i seg som vi hadde kjennskap til eller lignet på noe vi hadde jobbet med før.
- Relevanse og popularitet i webutviklingsmiljøet innenfor Java. Spesielt i forhold til etterspørselen for rammeverket på jobbmarkedet.
- Kvalitet og tilgjengelighet på dokumentasjon, eventuelle kurs og kodeeksempler.
- Brukervennlighet og vanskelighetsgrad for nybegynnere.
- Vi unngikk rammeverk som genererer kode. Dette gjorde vi fordi vi mente dette kunne skape forvirring når vi ikke var kjent med utviklingsmiljøet og minke læringsutbytte.

4.1.1 Maven eller Gradle

Maven og Gradle er eksempler på automatiserte byggesystem verktøy. Verktøyet kan hjelpe utviklere forstå hvordan programvare er bygd og hvilke «dependencies» som brukes. Maven[8] konfigureres med en XML fil som beskriver prosjektet som skal bygges, dependency i forhold til programvare ved tredje-part moduler og deler, rekkefølgen på og eventuelle «plugins» som kreves. Gradle[9] konfigurerer det samme men med ett domene-spesifisert språk som er basert på språket Groovy. Gradle har endel funksjoner som Maven ikke har, som for eksempel inkrementell bygging og andre konfigurasjons alternativer med tanke på kompilering. XML-miljøet er noe vi allerede var kjent med, dette var ikke tilfelle med Groovy. Tilleggsfunksjonene som Gradle kunne tilby var ikke, etter vår vurdering, relevant til vårt prosjekt. Når det kommer til popularitet kan det se ut som at Maven har den største delen av markedet [10].På bakgrunn av dette valgte vi å bruke Maven i dette prosjektet.

4.1.2 Rammeverk for webutvikling på serversiden

Alternativer vi har vurdert er Spring, OpenXava og CUBA.platform. OpenXava er et rammeverk designet for rask utvikling. Rammeverket genererer database-struktur og brukergrensesnitt etter selvlagde Java klasser. OpenXava er lett å bruke, har god dokumentasjon og det finnes en god mengde med kode-eksempler og how-to guider på YouTube. Med en kvart million nedlastninger er OpenXava den mest brukte domene-baserte rammeverket for Java[11].

Spring er et rammeverk for webutvikling i Java miljøet. Rammeverket består av rundt 20 moduler som man kan velge å bruke ettersom et behov oppstår. Rammeverk som Spring har

ofte som mål å effektivisere prosesser og enkeltgjøre implementasjonen av ellers kompliserte eller tidkrevende å utvikle funksjonalitet. Spring er ganske altomspennende når det kommer til webutvikling og har moduler for alt fra sikkerhet og cloud-teknologi til database- og testverktøy. Det at Spring er så heldekkende kan forklare hvorfor rammeverket er så populært blant Java-utviklere[10]. Spring rammeverket er et av de mest brukte rammeverk uavhengig av språk-miljø ifølge Stackoverfolow survey 2018[12]. Angivelig har Spring en ganske bratt læringskurve og det kan være vanskelig for nybegynnere i Java å sette seg inn i.

CUBA.platform er et rammeverk som bygger på Spring. Rammeverket er på lik linje med OpenXava designet for rask utvikling hvor en polymer klient applikasjon blir generert fra en API struktur. Rammeverket er ikke gratis men har en gratis versjon med noe begrenset funksjonalitet.

Vi valgte til slutt å bruke Spring med en rekke moduler. Dette valget tok vi på bakgrunn av populariteten til rammeverket, tilgjengelighet på dokumentasjon, kurs tilgjengelig på internett og at minimalt med kode ville bli generert. Vi var klar over at det ville bli mye å sette seg inn i og mye som måtte læres, men vi var motivert og villig til å investere den tiden dette ville kreve av oss.

Av Spring sine 20 moduler har vi tatt i bruk disse:

Spring boot

Modulen initierer en spring applikasjon med mappestruktur og forhåndsvalgte dependencies. Modulen gir oss ett skjellet vi kan bygge videre på. Genererer minimalt med kode.

Spring MVC

Gir oss Model-View-Controller arkitektur og komponenter som kan brukes til å utvikle webapplikasjoner. Rammeverket er designet rundt en DispatcherServlet som tar seg av HTTP forespørsler og svar. Denne modulen brukte vi til å utvikle vår REST API controller.

Spring data

Spring data modulen gir oss en rekke verktøy når det gjelder å integrere databasen inn i systemet. Modulen legger til et abstraksjonsnivå til kommunikasjon og interaksjon med databasen. Ved hjelp av «Hibernate» (A) kan vi generere en databasedesign ut ifra Java klasser. Spring «JPA» (A) er et verktøy som gir kommunikasjon med databasen et abstraksjonsnivå som bidrar til effektivisering av utviklingen av metoder som kommuniserer mellom et Java objekt og databasen.

Spring security

Bidrar til autentisering- og autorisasjons funksjonalitet til webapplikasjonen. Gir beskyttelse mot en rekke type angrep. Skal angivelig være lettvint å konfigurere og tilpasse til prosjekt.

Testing

Spring har egne moduler for testing, både for unittester og tester for funksjonalitet.

4.1.3 RESTful API

Spring rammeverket er tilpasset og er ment til å brukes brukes med en RESTful API arkitektur. Se vedlegg A for definisjonen på RESTful API.

4.1.4 Token basert eller cookie basert autentisering

Cookie basert autentisering er en metode som har eksistert lengde og som har blitt brukt mye. Metoden kjennetegnes ved at den er «stateful». Det vil si at en «session» må bli lagret både på serveren og på klienten. En database må lagre hver aktive «session», mens på klienten blir en «cookie» laget som inneholder en data som kan kjenne igjen en «session».

Token basert autentisering eller autentisering med «JSON Web Tokens(JWT)» (A) som vi har valgt å sette oss inn i, har blitt mer populært i det siste. Token basert autentisering er «stateless», det vil si at serveren ikke lagrer informasjon om innloggede brukere. Alle forespørsler som krever autentisering må derimot inneholde en token for å verifisere brukeren. JWT har flere overtak over cookie basert autentiseringen. Med «CORS» (A) blir det trivielt å la andre systemer integrere med vår API. Med JWT har også muligheten til å lagre og sende med all slags type data i en token. I en cookie derimot så lagres kun «session id». Cookies må sjekkes i databasen for hver gang de skal sjekkes. Tokens derimot kan sjekkes uten å aksessere databasen.

Basert på disse argumentene valgte vi å ta i bruk JWT for autentisering i prosjektet vårt.

4.1.5 Postman

Postman er et enkelt verktøy for å sende forespørsler til servere, slik som vår API. Dette gjør slik at vi slipper å må ha fungerende klient for å teste forespørsler mot API kontrolleren. Postman gjør det også mye enklere å sende inn testdata som man kan bruke på siden vår hvis det skulle trengs.

4.1.6 Rammeverk for webutvikling på klientsiden

Hvorfor Angular

Det finnes mange rammeverk å velge i når det kommer til klient-siden. De mest populære rammeverkene er React og Angular [12]. Vi har også sett på Thymeleaf. Rammeverkene varierer i kompleksitet, læringskurve og utviklings-miljø. React [13] er et bibliotek til Javascript, Angular [14] er et rammeverk som bruker Typescript. Thymeleaf er et java web-template generator som virker som et «view» lag i en MVC arkitektur. Generatoren tilbyr full integrasjon med Spring rammeverket og skrives direkte i HTML med attributter og tags.

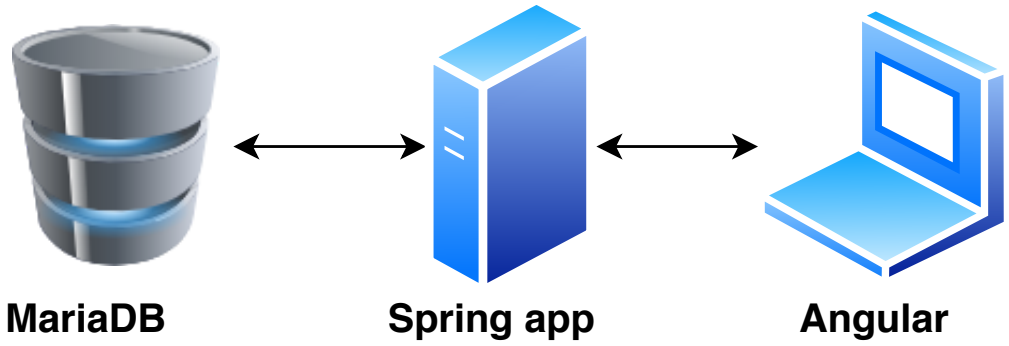
Til forskjell fra Javascript er Typescript mer som et programmeringsspråk vi er kjent med, slik som C++ eller Java. Typescript er et språk som kompileres, er mer objektorientert enn Javascript og det har «statisk typing» (A), som tvinger korrekt bruk av datatyper. Kompileringen gjør at man kan finne feil før det blir lagt ut på siden, som for eksempel datatyper som blir feil. Javascript er et «tolket språk» (A) som bare skjer i nettleseren der scriptet blir kjørt. Noe som kan gjøre det vanskeligere å feilsøke.

Begge har hver sin kraftfulle «CLI» (A) som gjør utviklingen mye enklere ved å automatisere noen deler av utviklingen. Eksempel på automatisering er generering av nye filer med skjellett og mappestruktur. Vi endte med å velge bort Thymeleaf på grunn av vår interesse for å lære rammeverk eller bibliotek som vi mente var mest relevant for fremtidig jobbsøking og hobby-programmering.

For oss som for det meste hadde jobbet med kompilerte språk med statiske typer og objekt-orientert programmering så virket Angular som det naturlige valget å ta, selv om vi har endel erfaring med Python som deler enkelte aspekter med Javascript der man har løse typer og som er et tolket språk.

Vi valgte tilslutt Angular for det virket for oss som det rammeverket som var best utstyrt for

oss og jobben. Etter de valgene vi tok med tanke på rammeverk og teknologier kunne vi skissere en grov arkitektur av hvordan webapplikasjonen ville se ut i figur 5.



Figur 5: Grov arkitektur planlagt etter valg av rammeverk.

Angular Material

Angular Material er et brukergrensesnitt med et komponent bibliotek for Material Design. Det har sine egne pakker og moduler som gir Google-utseende. Vi har valgt å bruke dette for å ikke selv måtte bruke så mye tid på CSS og stilarbeid.

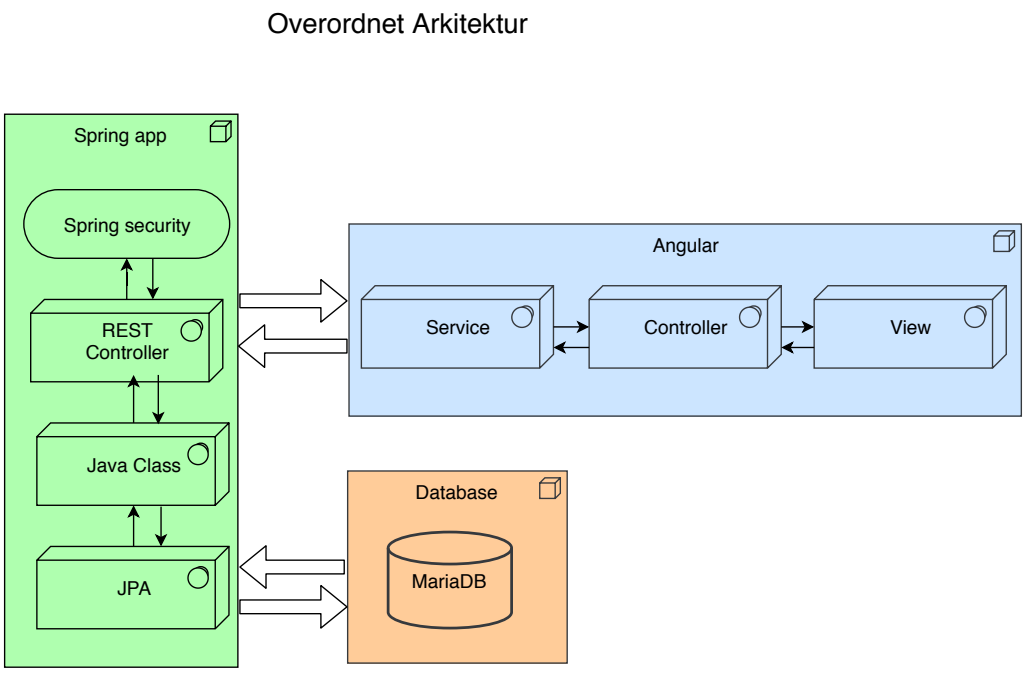
5 Design og arkitektur

5.1 Overordnet arkitektur

5.1.1 Valg av arkitektur

Da vi skulle velge i overordnet arkitektur stod det mellom en monolittisk struktur eller en klient-server struktur. En klient-server struktur har den fordel at den kan på enkelt vis utvides til «micro-services» (A) om dette er ønskelig i fremtiden. Vi så for oss at dette kunne være nødvendig hvis systemet skulle utvides med en beslutningsstøtte-modul og tilføres ytterligere funksjonalitet i salgsmodulen. «Mircoservices» har høyere oppetid ved oppdateringer og vedlikehold. En monolittisk arkitektur kan være tidskrevende og dyrt å konvertere til en klient-server struktur. På en annen side skal en monolittisk struktur være enklere å utvikle. Vi satte oss inn i «CORS filter» (A) funksjonalitet i Spring rammeverket og slo fast av dette ikke ville være krevende å konfigurere. Vi valgte tilslutt å benytte oss av en klient-server struktur på grunn av skalerbarheten til denne arkitekturen iforhold til en monolittisk struktur.

Figur 6 viser webapplikasjonens overordnet arkitektur.

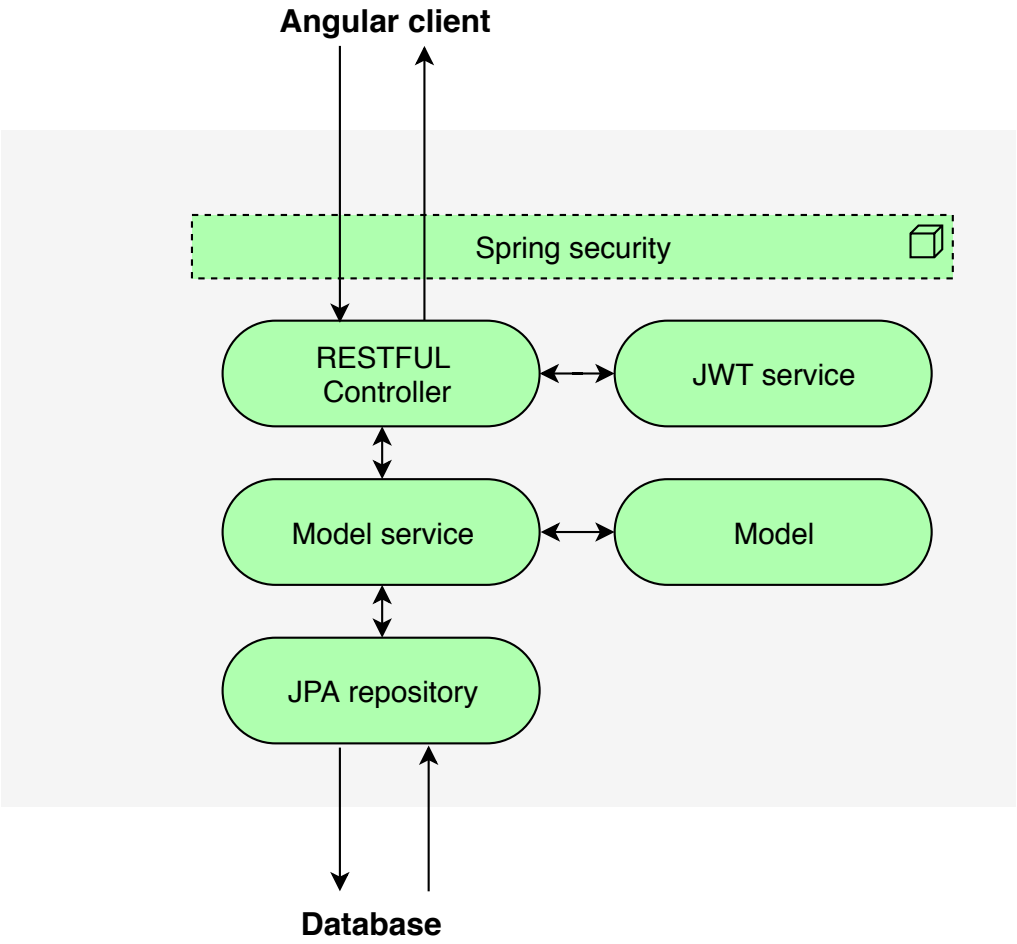


Figur 6: Overordnet arkitektur for webapplikasjonen

Angular klienten består av en MVC struktur, hvor en kontroller sender HTTP forespørsler og mottar responser til og fra vår REST API. Modell modulen brukes av kontrolleren til å konvertere responser og forespørsler til objekter. «View» bruker objekter til å lage ett brukergrensesnitt i nettleseren til brukeren.

Ved forespørsel til serveren vil «Spring security» sjekke om forespørselen krever autentisering. Hvis autentisering kreves må forespørselen inneholde en token i header. Token valideres og om brukeren innehar de rette tillatelsene får brukeren behandlet sin forespørsel.

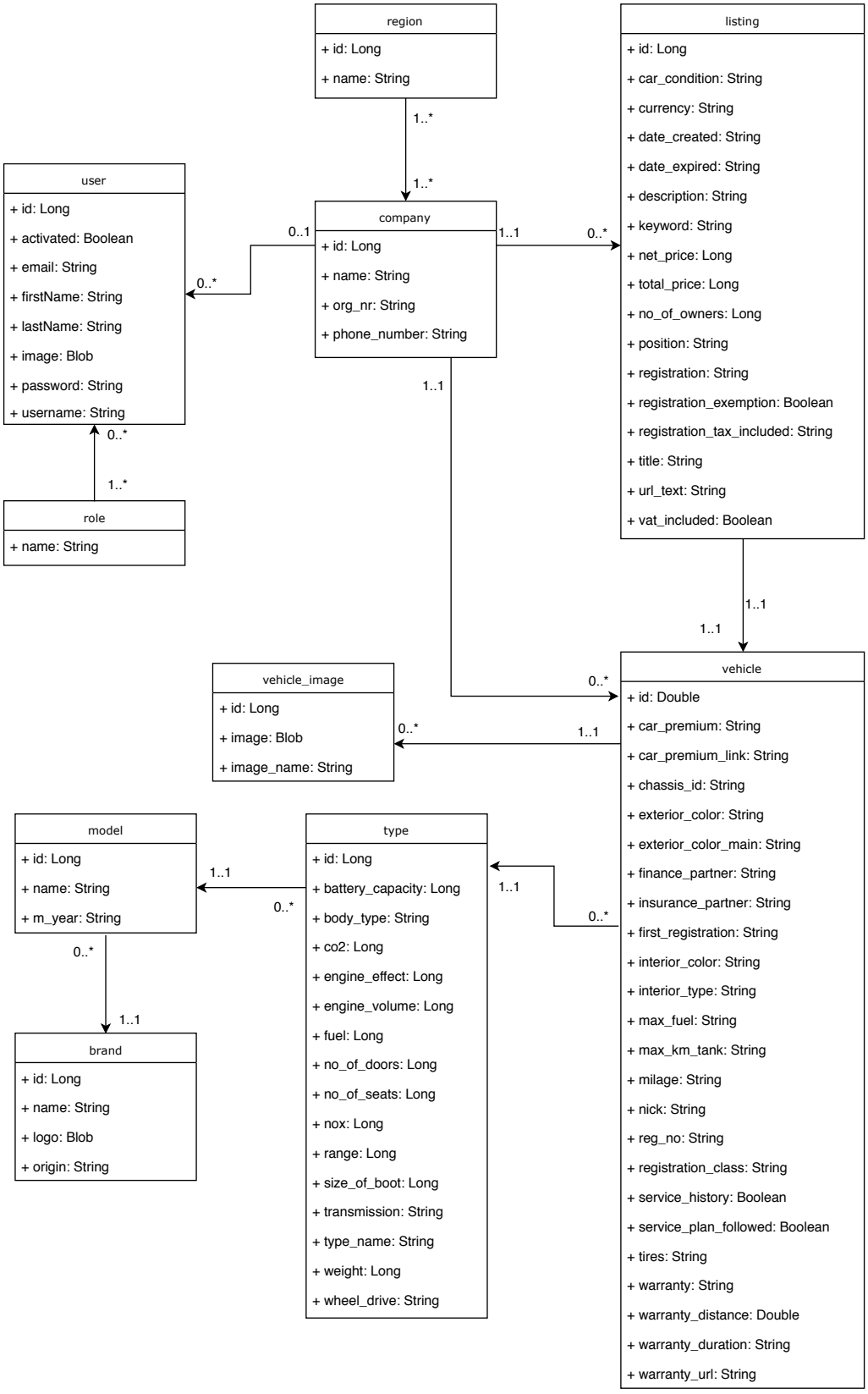
REST kontrolleren vil så behandle forespørselen ved hjelp av Modell modulen og dets tilhørende «JPA repository» som tar seg av alle databaseoperasjoner. Tilslutt vil REST kontrolleren gi et svar til klienten om status på forespørselen. Se figur 7 for en visuell forklaring.



Figur 7: Arkitektur over spring applikasjonen

5.2 Design

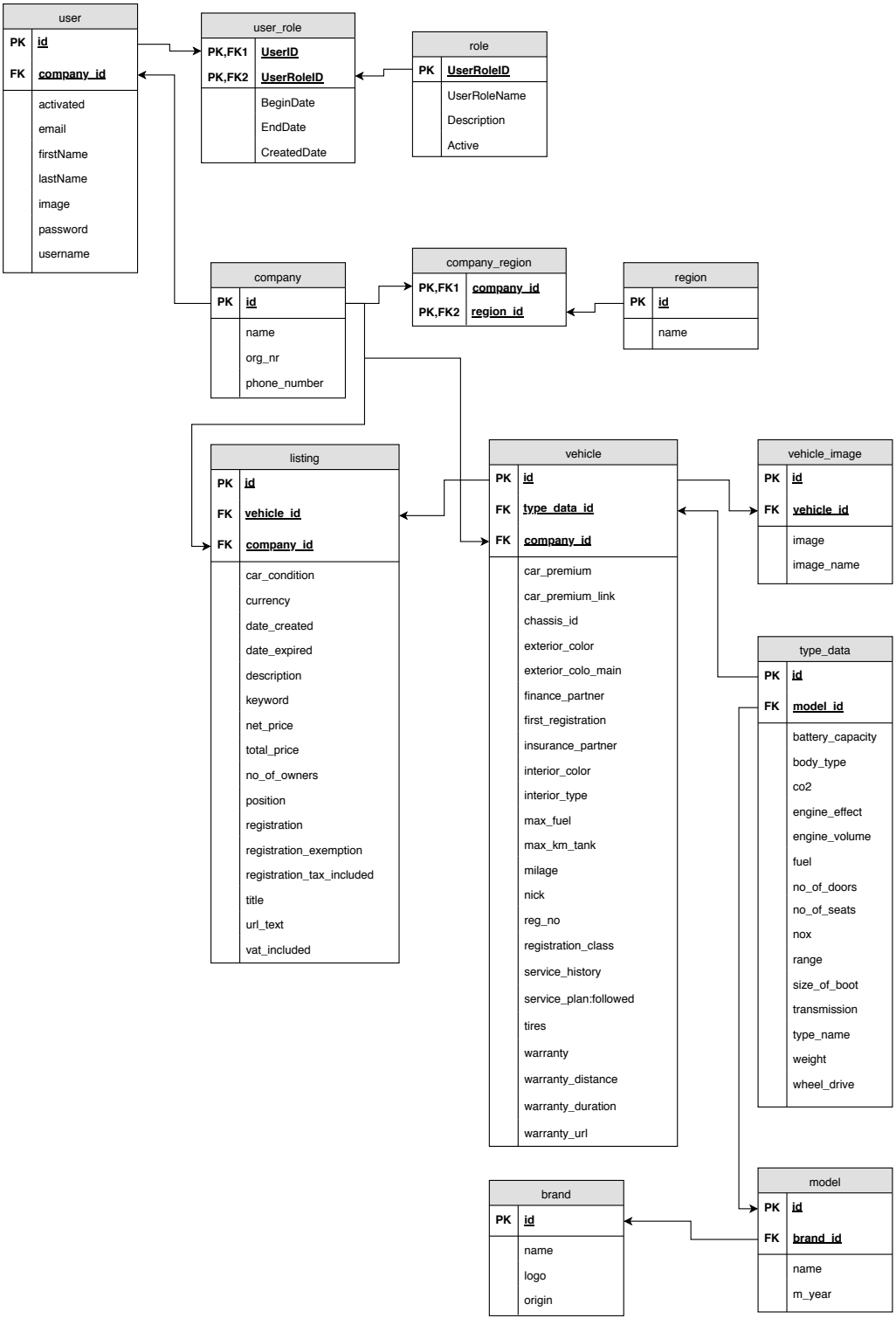
Designet på klasser i figur 8 er delvis basert på CarAdmin sin databasestruktur fra vedlegg E.2 og FINN.no sine krav på innsendelse av annonse til deres API i DTD-skjemaet i vedlegg E.1. Logikken bak designet er at kjøretøy ofte er en underkategori av en modell type som igjen er en underkategori av en modell som videre gjerne har en produsent. Et kjøretøy eies av en bedrift og en bedrift har flere ansatte eller her kalt brukere. En bedrift har annonser som inneholder ett kjøretøy de selv eier. Bedrifter holder til i en eller flere landsdeler. Brukere kan ha en eller flere roller, som bestemmer hvilke funksjonaliteter brukeren har tilgang til. Anonyme brukere blir ikke lagret i systemet. Kjøretøy kan ha flere bilder.



Figur 8: Klassediagram for webapplikasjon

5.3 Database

Databasestrukturen i figur 9 ble uformet fra klassediagrammet i figur 8 i henhold til kravene vi beskrev i kravspesifikasjonen 3.



Figur 9: Database diagram for webapplikasjonen i henhold til 3NF

6 Implementasjon

For å vise frem implementasjonen av webapplikasjonen har vi beskrevet en håndful av funksjonaliteter og vist prosessen gjennom systemet.

6.1 JPA

For at Spring data rammeverket skal kunne generere en database for oss må vi ta i bruk «annotations» for å markere klasser og klassemedlemmer. Som ett eksempel på implementasjon av JPA kan vi se på listing 6.1 hvor vi viser frem utvalgte klasse medlemmer. Klassen «company» markeres som en «entity» i en database struktur. Videre erklærer vi klassen som en tabell i databasen. Primærnøkkel blir satt til id og vi sier at id skal auto-inkrementeres. Vi deklarerer en til mange forhold med @oneToMany der «mappedBy» spesifiserer hvor fremmednøkkel skal ligge. I ett mange til mange forhold vil vi trenge en ny tabell for å normalisere strukturen. Vi konfigurerer den nye tabellen med «annotation» og konfigurerer fremmednøkler. Klassens metoder for interaksjon med databasen finnes i et JPA-repository som du kan se eksempel på i listing 6.2.

6.2 Registrering, skjema og posting

For å registrere en bruker i vår løsning må man være en administrator. Så på vår administrator side har vi lagt inn ett registreringsskjema som tar seg av dette. For å validere datatypene har vi satt krav i skjemaet om disse. Som du kan se i i listing 6.3 setter vi typen til «number» så gjør at vi bare godtar tall. Den er i tillegg merket «required» slik at skjemaet ikke kan bli sendt uten den. Resten av skjemaet finner du i vedlegg B.1

Listing 6.3: Input linje for tall i skjema

```
1 <input matInput placeholder="Selskaps-ID" type="
    ↪ number" [(ngModel)]="user.company_id" #
    ↪ company_id name="company_id" required/>
```

Dette skjemaet må vi sende videre til vår API. Det gjør vi via angular modulen «HttpClient» slik i listing 6.4, der «user» variablen er fra skjemaet over og «apiUrl» er adressen til API serveren vår.

Listing 6.4: Post forespørsel til API

```
1 this.http.post(`${this.globals.apiUrl}/auth/signup`,
    ↪ user);
```

Før forespørselen mottas av API kontrolleren sjekkes det om brukeren har administrator rollen som kreves for å registrere nye brukere. Dette gjøres i Spring security konfigurasjonen i listing 6.6. Hvis brukere innehar administrator-rollen sendes forespørselen til REST API kontrolleren i listing 6.5. «Annotation» @Valid og @RequestBody validerer input til serveren og beskytter mot «XSS» (A). Her valideres passordet som er gitt for den nye brukeren og dets data sendes videre for ytterligere validering i «Userservice». Funksjonen i «UserService» finnes i vedlegg B.3. Dersom den nye bruker data-en aksepteres legges den nye brukeren inn i databasen via

Listing 6.1: Klasse deklarasjon med JPA + Hibernate

```
1  @Entity
2  @Table(name = "company")
3  public class Company implements Serializable {
4
5      private static final long serialVersionUID = 1L;
6
7      @Id
8      @GeneratedValue(strategy = GenerationType.IDENTITY
           ↪ )
9      private Long id;
10
11     @Column(name = "name")
12     private String name;
13
14     @JsonIgnore
15     @OneToMany(mappedBy = "company")
16     private Set<Vehicle> vehicles = new HashSet<>();
17
18     @ManyToMany
19     @JoinTable(name = "company_region",
20             joinColumns = @JoinColumn(name = "
                ↪ company_id", referencedColumnName =
                ↪ "id"),
21             inverseJoinColumns = @JoinColumn(name = "
                ↪ region_id", referencedColumnName = "
                ↪ id"))
22     private Set<Region> regions = new HashSet<>();
23 }
```

Listing 6.2: Company JPA repository

```
1
2  @SuppressWarnings("unused")
3  @Repository
4  public interface CompanyRepository extends
       ↪ JpaRepository<Company, Long> {
5
6      Company findCompanyById(Long id);
7  }
```

Listing 6.5: API funksjon for å registrere ny bruker

```
1 @PostMapping("/signup")
2 @ResponseStatus(HttpStatus.CREATED)
3 public void registerAccount(@Valid @RequestBody
    ↪ ManagedUserVM managedUserVM) {
4     userService.registerUser(managedUserVM,
        ↪ managedUserVM.getPassword());
5 }
```

Listing 6.6: Konfigurasjon av Spring security

```
1  @Override
2  protected void configure(HttpSecurity http) throws
     ↪ Exception {
3      http.cors().and().csrf().disable().
4            authorizeRequests()
5            .antMatchers("/api/auth/signin").permitAll()
6            .antMatchers("/api/listings/**", "GET").
               ↪ permitAll()
7            .antMatchers("/api/auth/signup").hasRole("
               ↪ ADMIN")
8            .anyRequest().authenticated()
9            .and()
10           .exceptionHandling().
               ↪ authenticationEntryPoint(
               ↪ unauthorizedHandler).and()
11           .sessionManagement().sessionCreationPolicy(
               ↪ SessionCreationPolicy.STATELESS);
12
13     http.addFilterBefore(authenticationJwtTokenFilter
         ↪ (), UsernamePasswordAuthenticationFilter.
         ↪ class);
14 }
```

ett «JPA UserRepository». Om den nye brukeren ble lagt inn uten problemer vil API kontrolleren sende en 201 http status («created») tilbake til front-end. Dersom dette ikke skjer vil front-end motta en feilmelding.

6.3 Innlogging og autorisering

For å få logget inn så sender vi ett skjema med brukernavn og passord over til vårt API, slik som i registrering 6.2. Som man kan se i konfigurasjonen av Spring security 6.6 så er logg inn funksjonen i API kontrolleren åpen for forespørsler uten noen form for autorisering. Logg inn funksjonen tar et «loginform» objekt som parameter. Objektet inneholder brukernavn og passord samt sørger for at de er lovlige verdier. «authenticationManager» funksjonen fra Spring rammeverket sjekker om passordet stemmer overens med det brukernavn og passord som er lagret i databasen. Om passordet er riktig så vil en JWT genereres fra brukerens brukernavn i JwtProvider B.2. Om brukeren blir autentisert vil serveren svare med HTTP status 200 (ok) sammen med token og brukerens egen data. Koden for API funksjonen finnes i listing 6.7.

Listing 6.7: Login funksjon i API

```
@PostMapping("/signin")
public ResponseEntity<?> authenticateUser(@Valid
    ↪ @RequestBody LoginForm loginRequest) {
    Authentication authentication =
        ↪ authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                  loginRequest.getUsername(),
                  loginRequest.getPassword()
            )
    );
    // Sjekker om brukeren er autentisert
    SecurityContextHolder.getContext().
        ↪ setAuthentication(authentication);
    String jwt = jwtProvider.generateJwtToken(
        ↪ authentication);
    String username = jwtProvider.
        ↪ getUserNameFromJwtToken(jwt);
    CustomUserDetails customUserDetails = (
        ↪ CustomUserDetails) userDetailsService.
        ↪ loadUserByUsername(username);
    return ResponseEntity.ok(new JwtResponse(jwt,
        ↪ customUserDetails));
}
```

Token med brukerinfo blir sendt tilbake til brukeren og lagres lokalt. Med brukeren får vi tilsendt brukerens rolle. Denne rollen bruker vi i frontend til å bestemme hva man ser på siden. For eksempel har administrator en «administrer» knapp i sidemenyen som en vanlig bruker ikke har. Denne rollen brukes også for å bestemme hvilke sider man har tilgang på.

Listing 6.9: Eksempel på bruk av rolle til å begrense tilgang

```
@NgModule({
  imports: [RouterModule.forRoot([
    { path: '', component: HomeComponent },
    { path: 'register', component: RegisterComponent,
        ↪ canActivate: [RoleGuardService], data: {
        ↪ expectedRole: 'ROLE_ADMIN' }},
```

Listing 6.9 er ett utdrag fra vår routing modul, den håndterer adresser og hvilke komponenter som ligger på den adressen. Vi har satt adressen «register» til å kalle rollesjekken vår som vist i listing 6.8, der «canActivate» funksjonen henter rollen fra lokalt lager og sjekker den opp mot den medsendte rollen «expectedRole» og nekter eller tillater adgang til siden.
I listing 6.10 bruker vi om brukeren er logget inn til å bestemme om vi viser «Logg inn» knappen eller ikke. Det skjer ved en «*ngIf statement» som sjekker om brukeren finnes.

Listing 6.10: Bruk av roller for å gjemme innhold

```
<a mat-list-item (click)="toggleSidenav()" routerLink
    ↪ ="/login" *ngIf="!currentUser">Logg inn</a>
```

Listing 6.8: Funksjon for sjekk av rett rolle for bruk ved adgangskontroll

```
1  canActivate(route: ActivatedRouteSnapshot): boolean
     ↪ {
2    const expectedRole = route.data.expectedRole;
3    let asExpected = false;
4    const token = JSON.parse(localStorage.getItem('
       ↪ currentUser')).token;
5    const tokenPayload = decode(token);
6    console.log(tokenPayload.role);
7    for (let i = 0; i < tokenPayload.role.length; i++)
       ↪ {
8      if (tokenPayload.role[i].authority ===
         ↪ expectedRole) {
9        // Rolle som forventet
10       asExpected = true;
11     }
12   }
13   if (!this.authenticationService.isAuthenticated()
       ↪ || !asExpected) {
14     this.router.navigate(['']);
15     return false;
16   }
17   return true;
18 }
```

6.4 Bildevisning

Bildevisning er en viktig del av det å selge biler. Vi valgte å lagre bildene på var database med datatypen «Blob» (A) for å unngå mye data på webserveren, siden hver bil som skal selges har mest sannsynlig flere bilder i ett album som viser frem bilen.

For å laste opp bilder lager vi objekter av de som passer til vår database slik som i listing 6.11, som da postes til vårt API. Du kan se begge funksjonene som hjelper ved opplasting i vedlegg B.6.

Listing 6.11: Hvordan vi formaterer bildet for å bli sendt til API'et

```
1  this.vImage = {
2    image: this.imageArr[i],
3    imageContentType: fileExt,
4    vehicle_id: this.vehicle.id
5  };
```

Når brukeren sender en post forespørsel til vårt API vil den bli møtt av funksjonen i listingen 6.12. Post dataen legges i et hjelpe objekt kalt ImageUpload. ImageUpload består av en «Blob», som er selve filen, filtypen og kjøretøy id til bilde. I ImageUpload sin konstruktor blir filen konvertert fra base64 string til blob.

Ved en «Get» forespørsel til serveren får vi bilder som tilhører ett enkelt kjøretøy på samme format som «ImageUpload», men selve bildedelen er formatert rart, så vi må ta høyde for det. Vi fant ut at filen vår ble sendt tilbake annerledes en forventet. For det første ble den ikke sendt som «Blob» men som «Base64» (A). I tillegg til dette var den formatert fil for å være

Listing 6.12: Opplast bilde funksjon i API

```
@PostMapping("/vehicle-image")
public ResponseEntity<VehicleImage>
    ↪ createVehicleImage(@RequestBody ImageUpload
    ↪ imageUpload) throws URISyntaxException {
    VehicleImage album = new VehicleImage();
    album.setVehicle(vehicleRepository.findVehicleById
        ↪ (imageUpload.getVehicle_id()));
    album.setImage(imageUpload.getImage());
    album.setImageContentType(imageUpload.
        ↪ getImageContentType());
    VehicleImage result = vehicleImageRepository.save(
        ↪ album);
    return ResponseEntity.created(new URI("/api/
        ↪ vehicle-albums/" + result.getId()))
        .headers(HeaderUtil.createEntityCreationAlert(
            ↪ ENTITY_NAME, result.getId().toString()))
        .body(result);
}
```

Listing 6.13: Funksjon for å rette på filen fra server

```
convertImage(dataImage: string, dataContentType:
    ↪ string) {
    const base64String = JSON.stringify(dataImage);
    const startString = '"dataimage/' +
        ↪ dataContentType + 'base64';
    const base64Data = base64String.substring(
        ↪ startString.length, base64String.length - 1)
        ↪ ;
    return '
        ↪ ' + base64Data;
}
```

«Base64». Den kom på formen «dataimage/jpegbase64$FILE"» når den skulle ha kommet slik: «$FILE». Der $FILE er selve fildataene. Dette gjorde at vi måtte lage en egen funksjon for å fikse dette om du kan se i listing 6.13. Etter denne konvertering fungerte bildene som forventet.

For bildefremvisning på front-end brukte vi pakken «ngx-gallery», link til pakkens «github-repository» finner du å bibliografien [15]. Det er en pakke for Angular som hjelper å vise frem bilder som ett galleri/album. Dette er den enkleste måten vi fant for å vise frem flere bilder som ett album.

6.5 Finn.no integrasjon

Vi har implementert en metode for å sende XML-skjema med de korrekte elementene til FINN.no for å opprette en annonse på deres platform. Vi bruker hjelpe-objekter for å lage XML-skjemaet som kreves i FINN.no sin dokumentasjon, se vedlegg E.1. For å konvertere objekt til et XML-

Listing 6.14: Bruk JAXB «annotations» i Java klasse

```
@XmlRootElement
public class Motor_price {
    @XmlAttribute
    private String registrationtax_included;
    @XmlAttribute
    private String roadtax_included;
    @XmlElement
    private Long total;
    @XmlElement
    private Long registration;
}
```

Listing 6.15: objekt konvertert til XML string

```
@PostMapping("/listing-finn")
@ResponseStatus(HttpStatus.CREATED)
public void sendToFinn(@RequestBody ListingXML
    ↪ listingXML) throws JAXBException {

    JAXBContext jaxbContext = JAXBContext.newInstance(
        ↪ ListingXML.class);
    Marshaller jaxbMarshaller = jaxbContext.
        ↪ createMarshaller();
    jaxbMarshaller.setProperty(Marshaller.
        ↪ JAXB_FORMATTED_OUTPUT, true);

    StringWriter sw = new StringWriter();
    jaxbMarshaller.marshal(listingXML, sw);
    String xmlString = sw.toString();
}
```

skjema bruker vi biblioteket JAXB(Java Architecture for XML Binding). JAXB har «annotations» som vi kan bruke til å markere klasse medlemmer som elementer eller attributter se listing 6.14 for eksempel klasse. I listing 6.15 konverterer vi objektet som er markert til XML ved hjelp av «marshalling». Den resulterende XML-teksten sendes så til FINN.no API ved hjelp av en «post» forespørsel. Funksjonaliteten er ikke implementert på front-end.

6.6 Utseende og oppsett på nettside

6.6.1 Oppsett

Måten vi har satt opp siden er ved komponenter som Angular kaller det. Her er for eksempel hele koden for hjemmesiden vår i listing 6.16. Her har vi en av to material kort som vises, avhengig av om man er logget inn eller ikke. En «alert» komponent som viser eventuelle feilmeldinger, og et nytt kort som refererer til en søkeresultat komponent. Dette er for å kunne gjenbruke komponenter flere forskjellige plasser på vår nettside Hvis vi vil kunne vi ha hatt søkeresultater pakket inn på hvilken som helst side vi ville ha hatt den på.

Listing 6.16: Html for hjemmesiden

```
<mat-card>
  <mat-card-title *ngIf="!currentUser">Velkommen</mat-card-title>
  <mat-card-title *ngIf="currentUser">Hei {{currentUser.user.firstName}}!</mat-card-title>
  <alert></alert>
</mat-card>
<mat-card><app-search-result></app-search-result></mat-card>
```

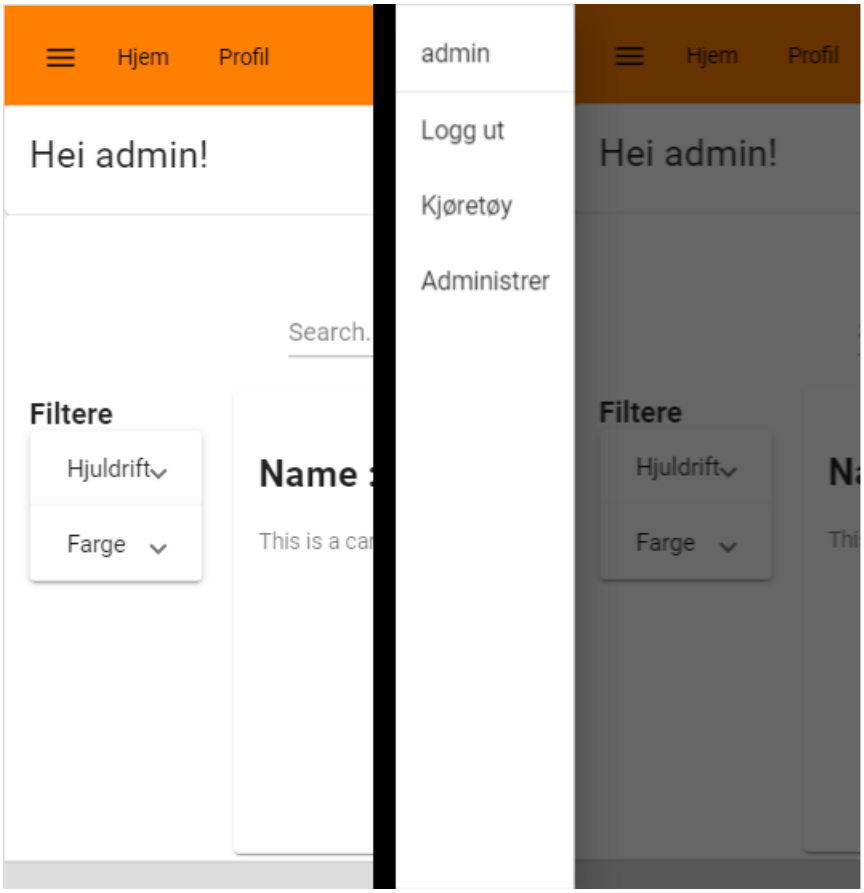
Dette oppnår vi via routeren vår, som er en standardpakke i Angular. Routeren vår bestemmer utifra adressen på nettsiden hva som skal vises der den ligger. Som vi ser fra tidligere i listing 6.9 så linkes en tom adresse til vår hjemmeside. Og det er da dit det linkes fra hovedsiden vår som du kan se i vedlegg B.5 med html taggen «router-outlet».

I vår hovedside som du kan se eksempel på i listing 6.17 så har vi omringet hele vår nettside med en «sidenav», som som vi kommer tilbake til, og at vi har en «header» og en «footer» fil som omringer vår «router-outlet» som nevnt i ovenfor. Dette gjør at hver eneste side i vår løsning har disse komponentene uten å må inkludere de for hver side. Dette blir i bunn og grunn en «Single Page Application», der hele siden ligger på en side og «routeren» tar seg av hva som lastes inn på siden.

Sidemenyen er en komponent i Material men implementasjonen var litt spesiell da den måte omringe hele applikasjonen for å fungere slik vi ville, altså at den dekker over hele siden og ikke bare mindre komponenter. Vi har laget ett eksempel på hovedside med en link i sidenavigasjons-menyen i listing 6.17. Og utseende på denne menyen kan du se i figur 10. Vi måtte lage en egen «service» for å knytte sidenavigasjonen opp mot toppmenyen slik at hamburgerknappen åpner sidemenyen.

Listing 6.17: Eksempel hovedside med en link i sidemenyen.

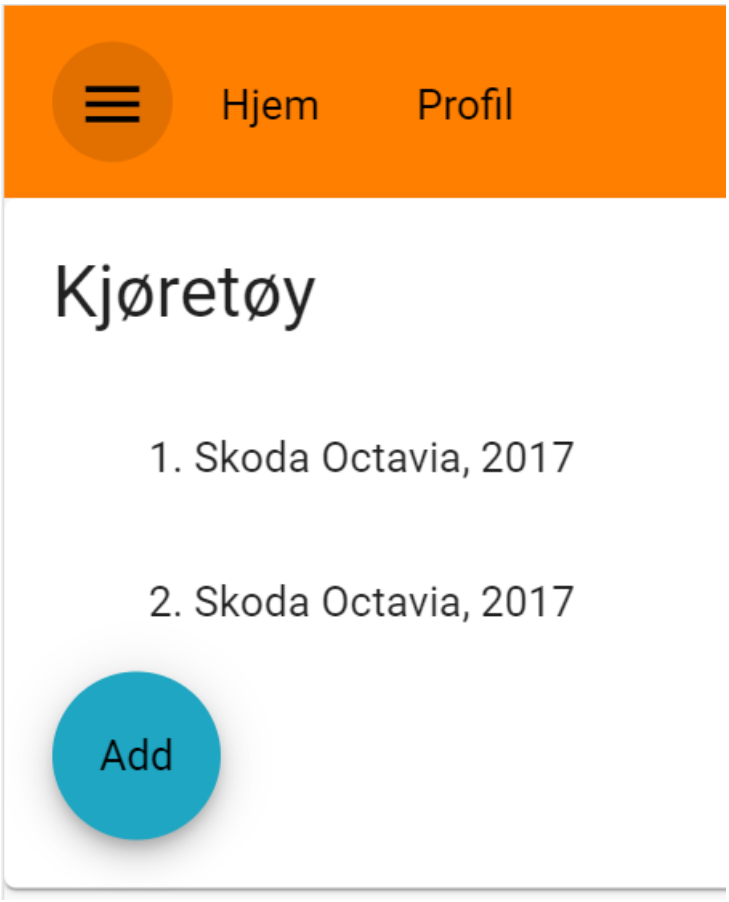
```
<mat-sidenav-container fullscreen>
  <mat-sidenav-content>
    <app-header></app-header>
    <router-outlet></router-outlet>
    <app-footer></app-footer>
  </mat-sidenav-content>
  <mat-sidenav #sidenav mode="push" [(opened)]="opened" [fixedInViewport]="true">
    <mat-nav-list>
      <a mat-list-item *ngIf="currentUser" routerLink="/user-profile/{{currentUser.id}}">{{currentUser.firstName}}</a>
    </mat-nav-list>
  </mat-sidenav>
</mat-sidenav-container>
```



Figur 10: Sidenavigasjonsmenyen for en administrator.

6.6.2 Utseende

For utseende av applikasjonen har vi tatt valget å bruke Angular Material fra Google. Dette er det Google bruker for mesteparten av sine applikasjoner. For å bruke dette må vi først definere ett «theme» som bestemmer fargene som vil bli brukt i vår applikasjon. Da definer man hovedfarge, sekundærfarge og en varselfarge. Vi definerte fargene etter det som ETC selv har brukt på sin side. Så oransje, blå og til slutt rød for varselfarge ved feil. Så brukte vi Material sine «html-tags» på siden, og utseende skjer etter disse. Fargene kan du se i figur 11



Figur 11: Eksempel farger i applikasjonen

7 Diskusjon og konklusjon

7.1 Resultater

I denne oppgaven ble kildekoden og den påfølgende web-applikasjonen et resultat av det arbeidet vi gjorde innefor de rammene vi definerte i delkapittel 1.6. I delkapittel 1.2 satte vi oss en rekke mål for vår løsning både for oss og for oppdragsgiver. I denne delen diskuterer vi hvorvidt vårt arbeid og løsning nådde disse målene.

7.1.1 Resulatmål

Et av våre resultatmål var å følge retningslinjene for design av Material Design fra Google [2]. Ved bruk av Angular Material sine komponenter på front-end, vil vi si vi har fulgt retningslinjene til Material Design til ett tilfredsstillende nivå.

Et annet mål var om vår løsning inneholdt de funksjonalitetene som ETC ønsket. Dette målet nådde vi desverre ikke som følge av de rammer som ble satt. De funksjonalitetene som ikke ble implementert er:

- Manglende side for bedrift, og tilsvarende infoside for kontaktinformasjon til bedriften.
- Funksjonalitet for å lage en annonse på siden, da dette var planlagt å være en del av bedriftsiden.
- Knytte kjøretøy opp mot bedrifter slik at bare de kan redigere sine egne registrerte biler.
- Søkefeltet vårt fungerer ikke lenger slik det gjorde tidligere av ukjente grunner.
- Vi har for få filtrere for søk til at det skal være hensiktsmessig å bruke. Burde blitt utarbeidet mer.
- Redigering av bruker på front-end er ikke ferdig implementert.
- Ferdigstille implementasjonen av å sende annonser til FINN.no

Det tredje resultatmålet var at serverens API skulle være godt dokumentert og kunne brukes av ETC. Dette målet oppfylte vi delvis. Som beskrevet i implementasjonen er serveren konfigurert med et CORS filter og en REST API slik at det kan kommunisere med tredjepart applikasjoner med HTTP forespørsler. Når det kommer til dokumentasjon er mesteparten av API funksjonene kommentert, men her var det enkelte mangler som måtte utbredes. Serveren mangler også funksjonelle og unit tester. I kravspesifikasjonen i kapittel 3 nevnte vi en rekke mitigasjoner som vi planla å implementere. Her fikk vi implementert alle mitigasjonsmetoder bortsett fra DDOS beskyttelse, noe som bidro til at vi bommet på det målet vi satte oss.

Et siste resultatmål vi satte oss var at vår løsning skulle bidra til at CarAdmin skulle dekke kunders behov innenfor biladministrasjon. Med den løsningen som vi presenterer på tiden denne rapporten skrives kan vi ikke si at vi nådde dette målet, men vi kan si at det har potensiale til å bidra til det i fremtiden med videre utvikling.

7.1.2 Effektmål

Når det kommer til effektmål ble dette vanskelig å svare på som følge av de tidsrammer som foreligger. Om det blir enklere for kundene av CarAdmin med kjøp og salg av kjøretøy er vanskelig å si noe om uten noe form for testing og påfølgende tilbakemelding. Det samme gjelder

eventuelle økonomiske gevinster for ETC.

7.1.3 Læringsmål

I læringsmål 1.2.3 skrev vi rekke mål som vi lagde til oss selv og ved utførelsen av denne oppgaven. Vi skrev at vi skulle erfare en systemutviklingsprosess i praksis med et prosjekt som var relevant for fremtidig arbeidsliv. Dette vil vi påstå at vi gjorde men med enkelte unntak. Mye tid gjennom oppgaven gikk til å lære seg språk, rammeverk og nye konsepter. Det ble ikke brukt så mye tid på å faktisk utvikle og ferdigstille funksjonalitet i forhold til det vi ser for oss er realiteten på en arbeidsplass.

Når det kommer til målene om å få erfaring innen web-utvikling både på server og klient siden så nådde vi disse målene på en god måte. Vi har gått fra å være uvitende om på dette område til å inneha en grei kompetanse som vi begge ønsker å utvikle videre. I læringsmål har vi også nevnt at vi ville utvikle løsninger ved bruken av ekstern API og utvikle vår egen.

Når det kommer til å utvikle med et eksternt API så kan vi si at vi ikke kom helt i mål. På grunn av tidsrammer fikk vi ikke tid til å ferdig utvikle funksjonaliteter som integrerte vår løsning til FINN.no platformen. Vår egen API fikk vi utvikle som nevnt tidligere.

7.2 Alternativer

I løpet av oppgaven har vi gjort endel undersøkelser og vurderinger på valg av teknologier, utviklingsmodell og arkitektur. I dette delkapittelet diskuterer vi konsekvensene av disse valgene og hvilke valg vi hadde gjort andeledes i retrospekt.

7.2.1 Rammeverk og teknologier

I analyse kapittelet 4 diskuterte vi valget av rammeverk for webutvikling på både server og klient siden. Utover de kravene ETC ga oss stod vi helt fritt til å velge rammeverk og teknologier selv. Dette har hatt både positive og negative innvirkninger på utførelse av oppgaven. De positive innvirkningene ville ha vært større om vi hadde tidligere erfaring eller kunnskap fra teknologier innen webutvikling, noe ingen av oss hadde. Mulighetene og alternativene ble derfor mange og det ble brukt mye tid på å sette seg inn i og forstå teknologier som ikke ble brukt.

Til server siden valgte vi Spring rammeverket på bakgrunn av dets relevans innenfor Java-miljøet, tilgjengelige resursser og fordi vi ikke ville bruke kode generatorer. En konsekvens av dette var at utvikling gikk tregt. Ikke bare måtte vi sette oss inn i Java, men også ett rammeverk som i store deler av utviklings perioden opplevdes som vanskelig og frustrerende. Frustrasjonen lønnet seg etterhvert da vi økte vår forståelse av rammeverket men dette krevde mye tid av oss.

Det er mulig at de alternative rammeverkene vi diskuterte i kapittel 4 som tilbyr et høyere abstraksjonsnivå til kodebasen og et fokus på rask utvikling ville ha effektivisert utviklingsprosessen. I dette valget prioriterte vi kanskje våre egne interesser for å få erfaring i et populært rammeverk over å velge det rammeverket som mest sannsynlig ville ha oppnådd de målene vi satte til løsningen og ETC sine forventninger. Utover Spring rammeverket i sin helhet er vi fornøyde med de modulene vi tok i bruk. Spesielt Spring data som var med å effektivisere utviklingen av metoder for å kommunisere med databasen.

I analyse kapittelet, delkapittel 4.1.6, diskuterte vi også valget av rammeverk for webutvikling på klient delen. Igjen valgte vi Angular på bakgrunn av relevans i webutviklingsmiljøet, og dens likhet med programmeringsspråk som vi hadde tidligere erfaring med. En god del tid ble brukt på fikling med komponenter fra Angular og Material. Det viste seg underveis at en god del av

dokumentasjonen var manglende for komponentene som Material har og vi måtte søke ganske mye på eksempler for å finne ut hvordan man bruker de. Da vi har brukt mye tid på å lære oss Angular og Material, kunne ett av de andre alternativene som vi vurderte spart oss tid, og da vært mye bedre for oss.

7.2.2 Utviklingsmodell

I kapittelet prosess 2 diskuterte vi valget av utviklingsmodell. Dette valget var vi godt fornøyd med og fleksibiliteten dette tilførte prosjektet ble viktig for oss slik vi trodde det ville bli. Det som kunne blitt gjort annerledes var at vi kunne brukt modellen mer aktivt i prosjektet vårt. I realiteten ble det brukt av og til i starten og veldig lite på mot slutten da vi hadde det travelt. Det er muligheter for at utviklingsprosessen hadde vært mer effektiv hadde vi fulgt modellen bedre.

7.2.3 Arkitektur

I arkitektur og design kapittelet 5 diskuterte vi valg av arkitektur for vår web-applikasjon. Vi står fast ved at klient-server arkitekturen med en REST API arkitektur på server delen var det riktige valget. Dersom vi valgte Thymeleaf og gikk for en monolittisk arkitektur ville API mistet sin anvendelighet. Vi hadde også mistet muligheten til å potensielt konvertere web-applikasjonen til en «microservice» arkitektur dersom ETC ønsket dette i fremtiden.

7.2.4 Lagring av bilder

For vår løsning valgte vi å lagre bildene direkte på databasen vår med hjelp av datatypen «Blob». Dette har kanskje ikke vært det beste valget vi kunne gjøre da vi fikk en del problemer med å få dataene tilbake fra databasen.

For mye tid ble brukt på å finne ut av problemer med bildene fra databasen. Vi s på dette som en ganske essensiell funksjon å få til i prosjektet vårt siden det er en salgsside. Lang tid ble brukt bare på feilsøke frem og tilbake i ett forsøk på å få bildene til å vise seg. Vi klarte det til slutt men det ble for mye fokus på det når vi kunne gjort en god del annet arbeid på resten av siden og kanskje fått gjort det som skulle til for at siden skulle bli brukbar.

Alternativ måte til lagring av bilder hadde vært å ha de liggende på webserveren i filsystemet, for så å referere til de via databasen, og slik få koblingen mellom kjøretøy og de rette bildene. Det er det vi skulle ha gjort da vår løsning resulterte i mye sløst tid for å finne ut av problemer.

7.2.5 Autentisering av brukere

I implementasjonskapittelet 6 beskriver vi vår implementasjon av JWT autentisering. Denne implementasjonen var en av de første modulene vi utviklet på server siden i Java. Denne implementasjonen tok mye ressurser fra oss da dette var vårt første møte med Spring security rammeverket, endel Java konsepter, tokens og HTTP statuser. Uten autentisering ville ikke webapplikasjonen kunne ha de funksjonalitetene som oppdragsiver krevde. Dermed var dette en essensiell funksjonalitet. Alternativene til å implementere denne funksjonaliteten selv var å bruke rammeverk som har slike funksjonaliteter generert. Dette ville ha fridd opp en betydelig mengde arbeidstimer til å nå de resultatmålene vi hadde satt oss. På en annen side var implementasjonen av denne funksjonaliteten svært lærererikt for oss, den ga oss et godt innblikk i Java, Spring og JWT.

7.3 Oppgavekritikk

Oppgaven ble beskrevet som en oppgave for to til fire personer. Vi var to personer på gruppen og den ble alt for stor for vår del. Oppgaven omhandlet originalt også en modul for besluttningshjelp

som bestod av et system som skulle hjelpe kunder til å velge nye kjøretøy å kjøpe utifra et sett med krav eller fra et kjøretøy som skulle erstattes. Denne delen av oppgaven falt bort sammen med andre spennende deler som nevnes i videre arbeid 7.4. Dette kommer av at vi ikke hadde den Java- og webutviklingskompetansen som krevdes av et slikt prosjekt. Som dataingeniørstudenter var vi mest interessert i noe litt mer spennende enn den oppgaven vi tilslutt endte opp med.

7.4 Videre arbeid

I tillegg til de mangler i web-applikasjonen som ble beskrevet i delkapittel 7.1.1 har vi gjennom oppgaven kommet på ideer og potensielle funksjonaliteter vi tror vil forbedre web-applikasjonen.

- Utvidet integrasjon med FINN.no hvor man kan starte og stoppe annonser fra applikasjonen.
- En prisantydningsfunksjon som gir kunden et forslag til salgspris på et kjøretøy kunden har lagt inn. Enten basert på eksisterende kjøretøy database eller en «webscraper» (A).
- Funksjonalitet for generering av et annonsebanner i HTML som kan legges inn på eksterne websider og fungere som en annonse for kunders egne kjøretøy.
- Kjøpstøtte ved hjelp av eksternt API eller database.

Videre må web-applikasjonen bli klargjort for produksjon før den kan brukes av faktiske kunder.

7.5 Evaluering av gruppens arbeid

Vi arbeidet med dette prosjektet i litt over fire måneder med en gruppe på to gjennom prosjektet samarbeidet vi på en tilfredstillende måte og vi kommuniserte godt når det gjaldt faglige diskusjoner eller praktiske hindre. Gruppemedlemmene møtte opp og gjorde fordelte arbeidsoppgaver til avtalt tid. Vi har begge holdt motivasjonen oppe på tross av perioder med sykdom og frustrasjon knyttet til læringsprosessen. I disse tidene var vi flinke til å støtte hverandre. Organiseringen innad i gruppen som beskrevet i introduksjonen 1.7 har fungert bra. Som følge av delt ansvarsområdet fikk vi begge utforsket det fagområdet som interesserte oss mest. Etter vår mening jobbet vi jevnt og målrettet. Vi balanserte emne godt med andre emner vi gjennomførte samtidig.

7.6 Konklusjon

Gjennom bacheloroppgaven undersøkte og vurderte vi ulike rammeverk, teknologier og arkitekturer som var best egnet for vår oppgave. Videre lærte vi oss Java, Spring, Angular og ulike konsepter innen webutvikling. I parallell med læringsprosessen planla og utviklet vi en web-applikasjon ut i fra oppdragsgivers beskrivelse med innspill fra oss selv. Gjennom denne læringsprosessen oppnådde vi de læringsmålene vi satte for oss selv i planleggingsfasen. Det å utvikle et produkt som et oppdrag fra et reelt firma ga oss viktig erfaring med tanke på fremtidig arbeidsliv. Resultatmålene nådde vi derimot ikke som følge av tidsrammer og som en konsekvens av gruppens manglende kompetanse innen webutvikling før starten av prosjektet. Det er mulig at resultatmålene ville ha blitt nådd dersom vi hadde benyttet oss av rammeverk med større fokus på rask utvikling. Web-applikasjonen vi lagde må videreutvikles og forbedres om den skal kunne utnyttes og tas i bruk av oppdragsgiver.

Bibliografi

[1] E. T. C. AS. (2019) Caradmin produkter. [Online]. Available: http://www.caradmin.no/index.php/en/produkter

[2] Google, "Material design," https://material.io/design, 2019.

[3] K. Schwaber and J. Sutherland, "The scrum guide," https://www.scrumguides.org/scrum-guide.html, 11 2017, [På nett; besøkt 31. januar 2019].

[4] P. Ilic, "Scrum vs. kanban vs. scrumban," https://www.linkedin.com/pulse/scrum-vs-kanban-scrumban-petar-ilic, [På nett; besøkt 31. januar 2019].

[5] M. Howard and D. LeBlanc, *Writing secure code*. Microsoft Press, 2009.

[6] "Krav." [Online]. Available: https://www.datatilsynet.no/regelverk-og-verktoy/veiledere/programvareutvikling-med-innebygd-personvern/krav/

[7] "Database normalization in mysql: Four quick and easy steps." [Online]. Available: https://www.computerweekly.com/tutorial/Database-normalization-in-MySQL-Four-quick-and-easy-steps

[8] B. Porter, J. v. Zyl, and O. Lamy, "Maven – welcome to apache maven." [Online]. Available: https://maven.apache.org/

[9] Gradle features. [Online]. Available: https://gradle.org/features/

[10] E. Paraschiv, "The state of java in 2018," Nov 2018. [Online]. Available: https://www.baeldung.com/java-in-2018#build

[11] "Java web framework for rapid application development of enterprise applications." [Online]. Available: https://www.openxava.org/

[12] "Stack overflow developer survey 2019." [Online]. Available: https://insights.stackoverflow.com/survey/2019#technology-_-web-frameworks

[13] Wikipedia contributors, "React (javascript library) — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=React_(JavaScript_library)&oldid=895095953, 2019, [På nett; besøkt 12. mai 2019].

[14] ——, "Angular (web framework) — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Angular_(web_framework)&oldid=894910458, 2019, [På nett; besøkt 12. mai 2019].

[15] Łukasz Gałka, "ngx-gallery," https://github.com/lukasz-galka/ngx-gallery, 2019.

A Vedlegg definisjoner

CLI	Command-line interface, ett grensesnitt i kommandolinjen
Statisk typing	Datatypene er definert til variabler og sjekkes ved kompilering om de stemmer underveis.
Dynamisk typing	Variabler får den datatypen som passer dataene.
Kompilert språk	Språket kompileres til kjørbar kode før det brukes.
Tolket språk	Språket blir tolket og kjørt der og når det brukes.
MVC	Model-View-Controller, arkitekturmønster delt i tre lag.
Blob	Binary Large Object, samling av binær data.
Base64	Datatype som representerer binær data i tekstformat. En karakter i Base64 representerer 6 bits.
Github-Repository	Mappe for prosjekter som ligger på nettsiden Github.
Bean	Java klasse som innkapsler flere objekter og som følger visse konvensjoner.
Artifact	En fil, som regel av typen JAR som blir distribuert til ett Maven repository
Plugin	Tilføyer funksjonalitet til å manipulere en artifact.
Dependency	Eksterne biblioteker i Java-miljøet.
DTO	Data Transfer Object". Objekt som brukes til å flytte data mellom lag i programvare.
JWT/token	JSON Web Token er en JSON basert standard for å lage tokens som inneholder informasjon om en brukeres rettigheter til avgang. En server lager en token basert på en brukeres rettigheter for så sende den til brukeren. Brukeren kan så bruke denne nøkkelen til å få adgang til serveren.
CORS filter	(Cross-Origin Resource Sharing) Standard for å tillate forespørsler på tvers av domener.
Microservice	En arkitektur hvor en større applikasjon er bygd som en gruppe med modulære komponenter eller tjenester.
Servlet	I Java er en servlet en klasse. Klassen opererer som en platform for håndtering av visse typer HTTP forspørsler. Brukes til å implementere web applikasjoner.
RESTFUL API	(Representational State Transfer Application Program Interface) En arkitektur innen webdesign som bruker HTTP forespørsler til å «GET» ,«PUT», «POST» og «DELETE» data.
Annotation	Brukes til å tilføre meta data til Java kode. Brukes typisk til å gi instruksjoner til kompliatoren.

Hibernate	Et verktøy som konverterer et objekt orientert domene modell til en relasjonell database.
Marshalling	Prosessen med å transformere minnesrepresentasjonen av et objekt til et dataformat som er egnet for lagring eller overføring.
JPA	JPA er en java-spesifikasjon for tilgang til, vedvarende og styring av data mellom Java-objekter og en relasjonsdatabase.
Webscraper	Ett script som ekstrakter data fra websider og lagrer det i en database eller regneark for å så senere bli sendt til analyse eller annet bruk.
XSS	Cross Site Scripting, det å kjøre skripter via en nettside til en annen.

B Vedlegg kodeeksempler

Listing B.1: Input linje for tall i skjema

```
<div class="center-card">
  <mat-card class="register-card" role="form">
    <mat-card-title>Registrer bruker</mat-card-title>
    <mat-card-content>
      <form #userForm="ngForm" (ngSubmit)="onSubmit(
        ↪ userForm.value)">
        <mat-form-field>
          <input matInput placeholder="Fornavn" type="
            ↪ text" [(ngModel)]="user.firstName" #
            ↪ firstName name="firstName" required/>
        </mat-form-field>
        <br>
        <mat-form-field>
          <input matInput placeholder="Etternavn" type=
            ↪ "text" [(ngModel)]="user.lastName" #
            ↪ lastName name="lastName" required/>
        </mat-form-field>
        <br>
        <mat-form-field>
          <input matInput placeholder="E-post" type="
            ↪ email" [(ngModel)]="user.email" #email
            ↪ name="email" ngModel email required/>
        </mat-form-field>
        <br>
        <mat-form-field>
          <input matInput placeholder="Brukernavn" type
            ↪ ="text" [(ngModel)]="user.username" #
            ↪ username name="username" required/>
        </mat-form-field>
        <br>
        <mat-form-field>
          <input matInput placeholder="Passord" type="
            ↪ password" [(ngModel)]="user.password" #
            ↪ password name="password" required/>
        </mat-form-field>
        <br>
        <mat-form-field>
          <input matInput placeholder="Selskaps-ID"
            ↪ type="number" [(ngModel)]="user.
            ↪ company_id" #company_id name="company_id
            ↪ " required/>
        </mat-form-field>
        <br>
        <mat-form-field>
```

```
31          <mat-select placeholder="Rolle" type="text"
               ↪ [(ngModel)]="user.role" #role name="role
               ↪ " #role name="role" required multiple>
32            <mat-option [value]="'user'">Bruker</mat-
                 ↪ option>
33            <mat-option [value]="'admin'">Admin</mat-
                 ↪ option>
34          </mat-select>
35        </mat-form-field>
36        <mat-card-actions>
37          <button type="submit" [disabled]="!userForm.
               ↪ form.valid" mat-raised-button>Register</
               ↪ button>
38        </mat-card-actions>
39      </form>
40      <alert></alert>
41    </mat-card-content>
42  </mat-card>
43 </div>
```

Listing B.2: JWT provider servide

```
1  package com.etc.trader.service.jwt;
2
3  import io.jsonwebtoken.*;
4  import org.slf4j.Logger;
5  import org.slf4j.LoggerFactory;
6  import org.springframework.beans.factory.annotation.
      ↪ Value;
7  import org.springframework.security.core.
      ↪ Authentication;
8  import org.springframework.stereotype.Component;
9
10 import com.etc.trader.service.CustomUserDetails;
11 import java.util.Date;
12 /*
13 Generer en JWT
14 Validerer en JWT
15 Henter brukernavn fra JWT
16  */
17 @Component
18 public class JwtProvider {
19
20    private static final Logger logger = LoggerFactory
         ↪ .getLogger(JwtProvider.class);
21
22    @Value("$trader.app.jwtSecret")
23    private String jwtSecret;
24    // Tid til token løper ut
25    @Value("${trader.app.jwtExpiration}0")
26    private int jwtExpiration;
27
28    public String generateJwtToken(Authentication
         ↪ authentication) {
```

```
29
30          CustomUserDetails userPrincipal = (
                ↪ CustomUserDetails) authentication.
                ↪ getPrincipal();
31
32          return Jwts.builder()
33                  .setSubject((userPrincipal.getUsername())
                        ↪ )
34                  .setIssuedAt(new Date())
35                  .setExpiration(new Date((new Date()).
                        ↪ getTime() + jwtExpiration))
36                  .claim("role", userPrincipal.
                        ↪ getAuthorities())
37                  .signWith(SignatureAlgorithm.HS512,
                        ↪ jwtSecret)
38                  .compact();
39      }
40
41      /**
42       *
43       * @param token: bearer token
44       * @return username: brukernavn til bruker.
45       */
46      public String getUserNameFromJwtToken(String token
            ↪ ) {
47          return Jwts.parser()
48                  .setSigningKey(jwtSecret)
49                  .parseClaimsJws(token)
50                  .getBody().getSubject();
51      }
52
53      public boolean validateJwtToken(String authToken)
            ↪ {
54          try {
55              Jwts.parser().setSigningKey(jwtSecret).
                    ↪ parseClaimsJws(authToken);
56              return true;
57          } catch (SignatureException e) {
58              logger.error("Invalid␣JWT␣signature␣->␣
                    ↪ Message:␣{}␣", e);
59          } catch (MalformedJwtException e) {
60              logger.error("Invalid␣JWT␣token␣->␣Message:␣
                    ↪ {}", e);
61          } catch (ExpiredJwtException e) {
62              logger.error("Expired␣JWT␣token␣->␣Message:␣
                    ↪ {}", e);
63          } catch (UnsupportedJwtException e) {
64              logger.error("Unsupported␣JWT␣token␣->␣
                    ↪ Message:␣{}", e);
65          } catch (IllegalArgumentException e) {
66              logger.error("JWT␣claims␣string␣is␣empty␣->␣
                    ↪ Message:␣{}", e);
67          }
```

```
68
69          return false;
70      }
71  }
```

Listing B.3: userservice funksjon for registrering

```
1   public void registerUser(UserDTO userDTO, String
        ↪ password) {
2       userRepository.findOneByUsername(userDTO.
            ↪ getUsername().toLowerCase()).ifPresent(
            ↪ existingUser -> {
3
4           throw new LoginAlreadyUsedException();
5
6       });
7       userRepository.findOneByEmailIgnoreCase(userDTO.
            ↪ getEmail()).ifPresent(existingUser -> {
8
9           throw new EmailAlreadyUsedException();
10
11      });
12
13      User newUser = new User();
14      // Krypterer passord
15      String encryptedPassword = passwordEncoder.encode(
            ↪ password);
16      newUser.setUsername(userDTO.getUsername().
            ↪ toLowerCase());
17      newUser.setPassword(encryptedPassword);
18      newUser.setFirstName(userDTO.getFirstName());
19      newUser.setLastName(userDTO.getLastName());
20      newUser.setEmail(userDTO.getEmail().toLowerCase())
            ↪ ;
21      newUser.setActivated(true);
22      // Bruker blir medlem av bedrift
23      newUser.setCompany(companyRepository.
            ↪ findCompanyById(userDTO.getCompany_id()));
24
25      Set<String> strRoles = userDTO.getRole();
26      Set<Role> roles = new HashSet<>();
27
28      strRoles.forEach(role -> {
29          switch (role) {
30              case "admin":
31                  Role adminRole = roleRepository.
                        ↪ findByName(RoleName.ROLE_ADMIN)
32                          .orElseThrow(() -> new
                                ↪ RuntimeException("Fail!␣->␣
                                ↪ Cause:␣User␣Role␣not␣find."))
                                ↪ ;
33                  roles.add(adminRole);
34
35                  break;
```

```
36            case "pm":
37                Role pmRole = roleRepository.findByName(
                      ↪ RoleName.ROLE_PM)
38                        .orElseThrow(() -> new
                              ↪ RuntimeException("Fail!␣->␣
                              ↪ Cause:␣User␣Role␣not␣find."))
                              ↪ ;
39                roles.add(pmRole);
40
41                break;
42            default:
43                Role userRole = roleRepository.findByName
                      ↪ (RoleName.ROLE_USER)
44                        .orElseThrow(() -> new
                              ↪ RuntimeException("Fail!␣->␣
                              ↪ Cause:␣User␣Role␣not␣find."))
                              ↪ ;
45                roles.add(userRole);
46        }
47    });
48
49    newUser.setRoles(roles);
50    // Ny bruker lagres til databasen
51    userRepository.save(newUser);
52 }
```

Listing B.4: Globals

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable()
4  export class Globals {
5    apiUrl: String = 'http://localhost:8080/api';
6  }
```

Listing B.5: Hovedside for nettsiden

```
1  <mat-sidenav-container fullscreen>
2    <mat-sidenav-content>
3      <app-header></app-header>
4      <router-outlet></router-outlet>
5      <app-footer></app-footer>
6    </mat-sidenav-content>
7    <mat-sidenav #sidenav mode="push" [(opened)]="
         ↪ opened" [fixedInViewport]="true">
8      <mat-nav-list>
9        <a mat-button style="height:␣0;position:␣
             ↪ absolute"></a>
10       <a mat-list-item *ngIf="currentUser" routerLink=
             ↪ "/user-profile/{{currentUser.id}}">{{
             ↪ currentUser.firstName}}</a>
11       <mat-divider *ngIf="currentUser"></mat-divider>
12       <a mat-list-item (click)="toggleSidenav()"
             ↪ routerLink="/login" *ngIf="!currentUser"
             ↪ >Logg inn</a>
```

```
13      <a mat-list-item (click)="toggleSidenav()" (
           ↪ click)="logout()" *ngIf="currentUser">Logg
           ↪ ut</a>
14      <a mat-list-item (click)="toggleSidenav()"
           ↪ routerLink="/vehicle-list" *ngIf="
           ↪ currentUser">Kjøretøy</a>
15
16      <a mat-list-item (click)="toggleSidenav()"
           ↪ routerLink="/user-admin"
17        *ngIf="isAdmin">Administrer</a>
18    </mat-nav-list>
19  </mat-sidenav>
20 </mat-sidenav-container>
```

Listing B.6: Metode for opplasting av bilde fra front-end

```
1  fileChange(event: any) {
2    if (event.target.files) {
3      this.ourFile = event.target.files;
4      const reader = new FileReader();
5      reader.onload = (eventR: any) => {
6        this.url = eventR.target.result;
7        this.imageArr.push(this.url);
8        console.log(this.url);
9      };
10     reader.readAsDataURL(event.target.files[0]);
11   }
12 }
13
14 upload() {
15   for (let i = 0; i < this.imageArr.length; i++) {
16     const imgString = JSON.stringify(this.imageArr[i])
          ↪ ;
17     const fileExt = imgString.substring('{data:image/'
          ↪ .length, imgString.indexOf(';base64'));
18     this.vImage = {
19       image: this.imageArr[i],
20       imageContentType: fileExt,
21       vehicle_id: this.vehicle.id
22     };
23     this.imageService.postImage(this.vImage).pipe(
          ↪ first())
24       .subscribe(
25         data => {
26           this.alertService.success('Image␣posted');
27         }, error => {
28           this.alertService.error(error);
29         }
30       );
31 }
```

C Vedlegg prosjekt-plan og -avtale

C.1 Prosjektplan

Prosjektplan

Knut Grøstad
knutgro@stud.ntnu.no

Sindre Østrem
sindrost@stud.ntnu.no

12. mai 2019

Innhold

1 Mål og rammer

1.1 Bakgrunn

≪ETC er et innovativt IT-selskap med nyskapende løsninger for administrasjon av fellesbiler som benyttes av 3 eller flere sjåfører i små og store bilparker. CarAdmin som er hovedproduktet samler også spredte bilparker i en felles virtuell parkeringsplass som enkelt lar en person holde daglig oppfølging med hele bedriftens bilpark.≫ ([1])

I forbindelse med gjennomføring av BIDAT39 Bacheloroppgave for Dataingeniør har ETC lagt ut et oppgaveforslag som ble godkjent av NTNU og som vår gruppe valgte som oppgave. I oppgavebeskrivelsen skriver de at de ønsker å få utviklet nye funksjoner for sitt system. Disse funksjonene skal tilsammen tilføye CarAdmin et verktøy for salg av kjøretøy og et besluttningsverktøy for kjøp av kjøretøy.

1.2 Prosjektmål

1.2.1 Effektmål

- Ekspandere CarAdmin sine bruksområder og funksjonalitet med salg og kjøp av kjøretøy inn i kundenes bilpark.
- Gjøre det enklere for kunder å selge og kjøpe nye kjøretøy. Redusere kundenes avhengighet på leasingselskaper for administrering av kjøp og salg i kjøretøyflåten.
- Antall kommuner som abonnerer på CarAdmin vil øke med 20% i løpet av 3 år som følge av løsningen.

1.2.2 Resultatmål

- Løsningen skal følge konvensjoner for GUI beskrevet i retningslinjene i Apple human interaction [2]
- Løsningen skal inneha de funksjonalitetene som er beskrevet av ETC.
- APIen til systemet skal være dokumentert og kunne brukes av ETC.
- ETC sin løsning går mot å dekke alle behov ved biladministrasjon.

1.2.3 Læringsmål

- Erfare systemutviklingsprosess i praksis med et prosjekt som er relevant i forhold til det fremtidige arbeidslivet.
- Få erfaring med backend-utvikling i Java med Mariadb og Tomcat og andre verktøy.
- Få erfaring med frontend-utvikling i Angular7/Typescript.
- Utvikle løsninger ved bruk av ekstern API og utvikle egen API.

1.3 Rammer

1.3.1 Tidsrammer

Bacheloroppgaven har oppstart 10.01.19 og avsluttes 20.05.19 da endelig bachelorrapport skal leveres.
I løpet av oppgaven skal veileder motta tre statusrapporter på følgende datoer:

- 20.02
- 01.04
- 01.05

Produktet leveres til ETC normalt sammen med bacheloroppgaven hvis ikke annet er avtalt.

1.3.2 Tekniske rammer

ETC har satt krav til bruk av JavaEE, Tomcat og MariaDB. Når det gjelder på front-end så står vi fritt til å velge rammeverk selv.

2 Omfang

2.1 Fagområde

[1] Electric Time Car (ETC) er en softwareleverandør som holder til i Gjøvik. De har ett bilpark system med navn CarAdmin som hjelper bedrifter, i hovedsak offentlig sektor, med å administrere kjøretøy til sine ansatte. Kommuner er et godt eksempel. De har mange kjøretøy og forskjellige ansatte som benytter dem.

≪CarAdmin er en helhetlig løsning for kjøretøyoppfølging av alle kjøretøy for privat og offentlig sektor. CarAdmin er tilpasset ulik bruk med forskjellige løsninger for alt fra kjøretøypooler til serviceoppfølging av gressklippere. Ønskes det mer automatisert oppfølging kan dette gjøres via GPS som i vår elektroniske kjørebok.≫ ([3])

Caradmin er delt inn i fire moduler:

- **Kjøretøyregisteret** holder oversikt over alle kjøretøy i Norge. Det inneholder diverse informasjon angående kjøretøyet, f.eks. km-stand, tid for neste service og lokasjon.
- **Kjøreboka** loggfører kjøreturer ved hjelp av GPS. Dette gjør det enkelt å foreta kjøregodtgjørelse.
- **Fellesbil** er utviklet for situasjoner der flere sjåfører deler et kjøretøy. Modulen gir mulighet til å låse nøkler i et nøkkelskap i forbindelse med uttak og innlevering av kjøretøyene. Fellesbil er med på å sikre god ansvarsfordeling mellom sjåførene.
- **Flåte** gir en oversikt over hvilke biler som befinner seg i nærheten, eller nærme et målpunkt. Det gis mulighet for å se kjørerute i kart. Denne modulen brukes i forbindelse med ulike oppdrag, og oppdragsgiver trenger å se hvem som kan ta på seg oppdraget.

2.2 Avgrensning

ETC har ønsker om å utvide og forbedre brukeropplevelsen på sitt system, CarAdmin. De vil gjøre det enklere å administrere salg av kjøretøy i flåten til kunden og i tillegg ha et verktøy som hjelper kunden til ta beslutninger angående innkjøp av nye kjøretøy. Vår løsning skal ha et veldokumentert og fungerende API. ETC langtidsplan er å ha en komplett løsning som administrerer kjøretøyflåten for sine kunder og mener dette er en naturlig ekspansjon av sitt produkt.
Ettersom ETC sin kodebase er utviklet over 15 år vil et stort ressursbruk i oppgaven gå til å gjøre seg kjent med kodestrukturen i CarAdmin systemet. På bakgrunn av mangel på Java erfaring innad i gruppen og antall gruppemedlemmer skal vår løsning utvikles adskilt fra CarAdmin systemet. Dette betyr at vår løsningen må bygges fra bunn av.

[1] Fagområdet er et samarbeid mellom gruppe 4 og 5 grunnet samme oppdragsgiver etter råd fra veileder.

2.3 Oppgavebeskrivelse

Vi deler vår løsning inn i en salgsmodul og en kjøps-beslutningsmodul. Salgsmodulen skal være en markedsplass for kjøretøy som legges inn av brukere. Markedsplassen skal ha funksjoner som er typisk for slike platformer som feks FINN.no. Brukere vil kunne søke etter kjøretøy med ulike filtre og krav. Eksempler på dette kan være kilometerstand, antall seter, lokasjon og pris. Brukere skal kunne se historiske data til kjøretøyene, som feks skademeldinger, kjøp/salg historikk og heftelser.
Det skal undersøkes om det er mulig å overføre et kjøretøy som er lagt ut for salg direkte til FINN.no uten å måtte legge inn annonse manuelt.

Beslutningsmodulen skal være et verktøy som på bakgrunn av et valgt kjøretøy som er til salgs, vil foreslå et kjøretøy som kan erstatte dette kjøretøyet. Kjøretøyet som kvalifiseres som mulig erstatter skal være så lik original kjøretøyet som mulig når det gjelder krav og egenskaper. Kilometerstand er også en faktor som spiller inn på å finne en god erstatter. Krav som kan stilles til biler er størrelse, miljøvurdering, type dekk, om det er 4- eller 2-hjulstrekk, innebygd GPS, bombrikke, motorvarmer osv.

Annen funksjonalitet:

- E-post tjeneste på siden som kan brukes til å sende ut anbud til kjøretøyleverandør.
- Reklameringsverktøy for salg av egne biler. Skal kunne legges inn på selgers egen hjemmeside.
- Prisestimering/vurdering av biler som skal legges på salg. Ser på liknende biler til salgs eller skal det undersøkes om eksisterende databaser kan være til hjelp.

3 Prosjektorganisering

3.1 Ansvarsforhold og roller

I vår gruppe på to deler vi mye av ansvaret for gjennomføringen av bacheloroppgaven. Dette inkluderer ansvar om å oppholde tidsfrister, holde avtaler, aktiv deltagelse og opplæring i teknologi som måtte være relevant for gjennomførelse av oppgaven.
Vi har fordelt det slik at Knut Grøstad er prosjektleder med kommunikasjonansvar og Sindre Østrem er sekretær med ansvar for dokumentasjon av møter og beslutninger.
Utenom dette blir arbeidsoppgaver fordelt etter hvert gruppemedlems preferanser om oppgaver som oppstår.
Vår veilder, Frode Haug representerer NTNU sine interesser og er til hjelp for å råde og veilede arbeidsprosessen for å nå de målene som forventes av NTNU.
Vår oppdragsgiver ETC, representeres av Dag Solhaug som er dagligleder for ETC. Dag skal fremme ETC sine interesser i oppgaven og komme med kravspesifikasjoner, konstruktiv tilbakemeldinger og teknisk rådgivning/hjelp.

3.2 Rutiner og regler i gruppa

3.2.1 Forventninger av deltagelse

1. Hvert medlem forventes å møte opp hver dag kl. 09:00 på campus med mindre annet er avtalt. Dagen varer normalt til 16:00 men dette kan justeres i løpet av dagen.
2. Delta aktivt i planlegging, koding, rapportskriving og på presentasjonen.
3. Ved fravær fra oppmøte skal medlemmet si ifra i god tid, helst 24 timer før.
4. Alle medlemmer forventes å møte opp på morgenmøte hver dag, i statusmøte og i møte med veileder eller oppdragsgiver ved avtalt tid.

3.2.2 Forventninger til gjennomførelse

1. Et morgenmøte hver dag og et statusmøte hver mandag med mindre annet er avtalt.
2. En sekretær utnevnes og skriver referat fra hvert statusmøte.
3. Referatet skal kort beskrive status og planen fremover.
4. Ved uenigheter skal det føres saklige argumenter og gruppen skal komme til enighet. Evt. gjennom et kompromiss.
5. Er man ikke fornøyd med et gruppemedlems arbeidsinnsats, skal man straks følge rutinene beskrevet i rutiner ved regelbrudd.

3.2.3 Rutiner ved regelbrudd

1. Eventuelle problemer tas opp i morgenmøte, hvor den man mener begår regelbrudd blir informert om dette.
2. Skulle ikke dette utbedre situasjonen skal den som begår regelbrudd få en skriftlig advarsel signert av de andre i gruppen. Advarselen skal inneholde:
   (a) Hvilke regelbrudd som er begått, konkrete tiltak for å utbedre problemet.
   (b) Det avtales møte med veileder.
   (c) Eventuell ekskludering av gruppen ved gjentatte regelbrudd etter andre rutiner er gjennomført.

4 Planlegging, oppfølging og rapportering

4.1 Valg av systemutviklingsmodell

Ved valg av utviklingsmodell har vi vurdert både sekvensielle og smidige modeller. Problemet med en sekvensiell modell som fossefall er at kravspesifikasjon for utviklingsprosjektet må være klare før utviklingen begynner. I møter med oppdragsgiver er det tydelig at ny funksjonalitet og nye ideer kan oppstå under utviklingen. Det er også åpent og ønsket at vi, utviklerene kan komme med våre egne ideer iløpet av utviklingsprosessen. Vi er avhengig av at oppdragsiver gir oss tilbakemelding til arbeidet vi har gjort slik at vi kan forbedre produktet. Dette er viktig for oss pga mangel på erfaring med utvikling i Java. I tilegg er det flere funksjonaliteter i prosjektet som krever avtaler med eksterne aktører som ikke er på plass ved oppstart av prosjektet. Dette gjør at en sekvensiell modell blir feil for vårt prosjekt. Vår modell må være fleksibel og kunne brukes i en dynamisk utviklingsprosess. På grunn av denne vurderingen har vi sett på smidige modeller som Scrum og Scrumban som mulig utviklingsmodell for vårt prosjekt.

Scrum er en smidig utviklingsmodell som deler utviklingsprosessen i mindre iterative deler, kalt sprinter. I en sprint har man daglige møter og for enhver sprint må det planlegges. Scrumban er en blanding av metodene Scrum og Kanban. Den kombinerer de elementære delene av Scrum og fleksibiliteten til Kanban. Scrumban har en kontinuerlig utviklingsprosess med korte perioder for planlegging og lengre perioder for utgivelser.

Scrum er en litt mer restriktiv metode en Scrumban. Alle oppgaver er ofte bundet til en deadline, hvor på i Kanban og Scrumban bruker man en pull metode ettersom oppgaver blir ferdigstilt. En begrensning for antall oppgaver i utvikling (Work-In-Progress) reduserer risikoen for for mange baller i luften"og sørger for at oppgaver blir ferdigstilt før man begynner på nye oppgaver. I Scrum resetter man prosesstavlen etter hver sprint. Etter planleggingsmøte legger man inn nye oppgaver i backlog. Man kan ikke legge inn nye oppgaver i en sprint som er i gang. I Scrumban har man ≪on-demand≫ planlegging dersom backloggen er tom eller er under et bestemt antall. Man vil da kunne fylle opp To-do/backlog med nye oppgaver som pushes fra produkt-backlog. Det er ikke et krav til ≪planning-poker≫ og tidsestimering i Scrumban.

I vårt prosjekt vil vi foretrekke en kontinuerlig utviklingprosess for å kutte ned på tid brukt på estimering og scrum-ritualer. Da vi er et team på to vil vi helst bruke så mye tid på selve utviklingen som mulig. Tids-estimering av oppgaver er i vårt tilfelle vanskelig ettersom vi mangler erfaring på å utvikle større prosjekter. Derfor vil vi i vårt tilfelle foretrekke planning on demand. I tilegg er vi ikke godt nok kjent med vårt valgte utviklingsstack til å gjøre gode vurderinger. Scrum er ofte beskrevet som best for grupper på 3-9 eksperter[4], Scrumban er et mer fleksibelt rammeverk og passer best egnet for ≪fast-paced≫ utvikling som er typisk for oppstartsbedrifter[5]. Alt i alt gir Scrumban vår gruppe et godt rammeverk for hektisk systemutvikling med rutiner for planlegging og forbedring.

4.2 Hvordan vi skal bruke Scrumban

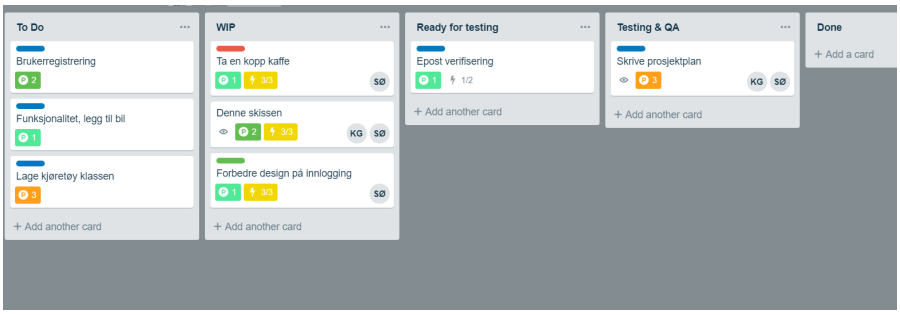
I vårt prosjekt vil vi bruke Trello som er virtuelt planleggingstavle(1). Tavlen vil bestå av følgende kolonner:

- Product backlog
  Inneholder funksjonaliteter som så langt er planlagt av oppdragsgiver i samarbeid med oss.
- To-do list
  Inneholder oppgaver, en enkel eller en del av en funksjonalitet. Oppgavene blir planlagt etter et planleggingmøte på bakgrunn av innholdet i ≪product backlog≫. Oppgaver blir tildelt en prioritering fra 1-3 hvor 3 er viktigst.
- Work-in-progress
  Ferdig planlagt oppgave trekkes inn i WIP. Velges av gruppemedlem på bakgrunn av preferanse. Kolonnen har en begrensning på 4 oppgaver.
- Klar-for-testing
  Oppgavene som er ferdig utviklet trekkes fra WIP og er klargjort for testing. Kolonnen har begrensning på 4 oppgaver.
- Testing og kvalitetsikring
  Nye modulen blir testet med funksjonelle tester og ≪static-code analyzer≫. Om modulen må forbedres flyttes den tilbake til to-do listen.
- Ferdig
  Kolonnen inneholder ferdigstilte og testet funksjonalitet.

Når To-do listen blir tom betyr dette at nye oppgaver må planlegges for at to-do listen skal fylles opp igjen. Oppgavene skal skilles i forskjellige typer oppgaver med fargekoder:

- Oppdateringer: Rød
- Forbedringer: Grønn
- Nye funksjonaliteter: Blå

Ferdigstilte moduler presenteres og vurderes hver 14. dag i møte med oppdragsiver (ETC).



Figur 1: Brettet vi skal bruke i utviklingen

4.3 Plan for statusrapportering

I gjennomførelsen av bacheloroppgaven har vi avtalt med veileder at vi møtes hver mandag 10:00. Denne avtalen kan endres etter gruppens behov. Vi planlegger også møter med oppdragsgiver etter behov.
Veileder skal også motta tre statusrapporter med frister 20. Januar, 01. April og 01. Mai.

5 Organisering av kvalitetssikring

5.1 Dokumentasjon, standardbruk og kildekode

All dokumentasjon skal skrives på norsk. Dokumentasjon forbundet med kildekode og API skal bo i github repositoriumet sammen med kildekoden. Selve kildekoden skal skrives på engelsk og følge etablerte konvensjoner i JavaEE og Typescript(Angular) miljøene. Dokumenter som møteagenda, møtereferater, timelogg og andre dokumenter forbundet med bacheloroppgaven lagres i en Google drive mappe som deles med gruppens medlemmer. Selve Bacheloroppgaven og prosjektplanen skrives i Overleaf som er et Latex redigeringsprogram. Bacheloroppgaven skal skrives i NTNU sin Latex mal. Referanser skal følge Vancouver metoden.

I utvikling prosessen skal ≪branching≫ utnyttes i git for optimal konfigurasjon styring. ≪Branches≫ brukes til utvikling av ny funksjonalitet for web-applikasjonen, dersom funksjonaliteten er ferdig, testet og godkjent kan den ≪merge≫ med ≪master≫. Bug fiksing skal også ≪branches≫ fra ≪master≫. Gruppens medlemmer skal gjøre hyppige ≪commits≫ til utviklings-≪branch≫ for å minke risikoen for tap av kode. Dette er i tråd med etablerte konvensjoner om konfigurasjon styring [6].

5.2 Verktøy

- Git:
  Brukes til konfigurasjonstyring av kildekode. Alle gruppens medlemmer er komfortable og godt kjente med git fra tidligere prosjektoppgaver.
- IntelliJ IDEA:
  Miljø for programvare utvikling i Java. Det foretrukne miljøet innad i gruppen. Brukes også av utviklerene i ETC.
- Trello:
  Trello er en web-applikasjon som er et verktøy for prosjektplanlegging og gjennomføring. Brukes i agile programvare utvikling.
- Draw.io
  Draw.io er en web-applikasjon for diagram tegning. Applikasjonen har flere maler til UML, Usecase og andre diagrammer for bruk i programvare design.
- Team gantt Team gantt er en web-applikasjon for enkel tegning av gantt skjema.

5.3 Risikoanalyse (identifisere, analysere, tiltak, oppfølging)

5.3.1 Tabellanalyse

Risiko	Sannsynlighet	Konsekvens	Vurdering
1. Tap av kildekode/data/dokumentasjon	Lav	Katastrofalt	Middels
2. Feilestimering av tid	Middels	Farlig	Middels
3. Kommunikasjonssvikt med oppdragsgiver og veileder	Lite	Mindre Farlig	Lav
4. Sykdom i gruppen	Middels	Farlig	Middels
5. Avtale med Finn.no ikke lar seg gjøre	Lite	Ufarlig	Lav
6. Rapporten blir forsinket pga programutviklingen	Lite	Katastrofalt	Middels
7. Kompetansesvikt	Lav	Mindre Farlig	Lav

Tabell 1: Risikovurderingstabell

5.3.2 1. Tap av kildekode/data/dokumentasjon

Tiltak: Lagring på skyen. Bruker github så å pushe kode når man er ferdig med ett arbeid. Jobbe direkte på skyen med annet arbeid. Lagre lokale kopier ved jevne mellomrom.
Oppfølging: Hente det på skyen og fortsette der arbeidet er kommet.

5.3.3 2. Feilestimering av tid

Tiltak: Holde veileder og oppdragsgiver oppdatert på framgangen og følge deres råd. Oppfølging: Ta bort ikke essensielle funksjonaliteter som vi ikke får tid til.

5.3.4 3. Kommunikasjonssvikt med oppdragsgiver og veileder

Tiltak: Møte opp og planlegge møter. Sørge for å kommunisere jevnlig.

5.3.5 4. Sykdom i gruppen

Tiltak: Alle på gruppen må jobbe mot allsidighet i prosjektet. Gruppens medlemmer innehar kunnskap ved alle aspekter i prosjektet, slik at hvis en på gruppen blir utilgjengelig kan man fortsette arbeidet.

5.3.6 5. Avtale med Finn.no ikke lar seg gjøre

Tiltak: Gjøre klar tilrettelegging for videre integrasjon, eller finne alternative løsninger.

5.3.7 6. Rapporten blir forsinket pga programutviklingen

Tiltak: Jobbe i forhold til at det skal skrives rapport. Dokumentere kravspesifikasjoner og design bestemmelser gjennom prosjektet. Avslutte utviklingen tidlig om nødvendig. Oppfølging: Avslutte all utvikling og skrive på rapporten.

5.3.8 7. Kompetansesvikt

Tiltak: Egenopplæring i verktøyene vi skal bruke. Eventuelt lære hverandre. Oppfølging: Andre på gruppen kan ta over arbeidet hvis det viser seg vanskelig.

6 Plan for gjennomføring

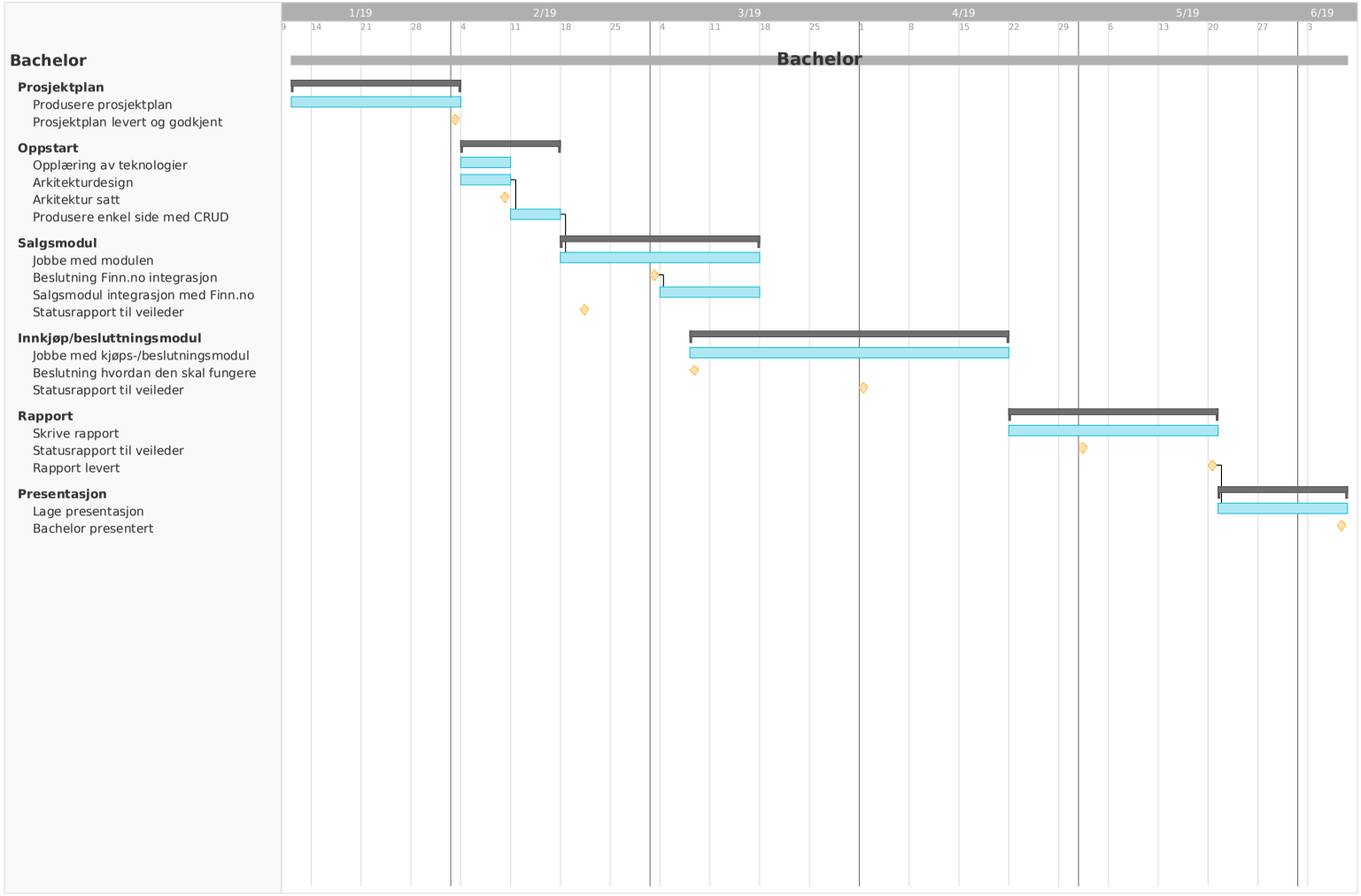
6.1 Aktiviteter

- Opplæring i java/angular.
- Design av arkitektur.
- Basic front-end & back-end.
- Webdesign.

- Salgsmodul.
  - Mulig integrering med Finn.no sitt API
- Kjøp/beslutningsmodul.
- Rapportskriving.
- Forberede presentasjon.

6.2 Gantt skjema

Vår planlegging når det gjelder tidspunkter er ikke endelige. Vi behandler dette mer som retningslinjer for når vi burde ha kommet så langt. Milepæler og beslutningspunkter inngår i gantt skjema. Se gantt skjema på side 13.



Referanser

[1] E. T. C. AS. (2019). Om etc, electric time car as, side: http://www.electrictimecar.com/modules/content/index.php?id=12 (sjekket 22.01.2019).

[2] Apple. (2019), side: https://developer.apple.com/design/human-interface-guidelines/macos/overview/themes/ (sjekket 28.01.2019).

[3] E. T. C. AS. (2019). Caradmin produkter, side: http://www.caradmin.no/index.php/en/produkter (sjekket 21.01.2019).

[4] K. Schwaber og J. Sutherland. (nov. 2017). The scrum guide, side: https://www.scrumguides.org/scrum-guide.html (sjekket 31.01.2019).

[5] P. Ilic. (7. mai 2015). Scrum vs. kanban vs. scrumban, side: https://www.linkedin.com/pulse/scrum-vs-kanban-scrumban-petar-ilic (sjekket 31.01.2019).

[6] V. Driessen. (5. jan. 2010). A successful git branching model, side: https://nvie.com/posts/a-successful-git-branching-model/ (sjekket 28.01.2019).

C.2 Prosjektavtalen

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

ETC, Electric Time Car AS

(oppdragsgiver), og

KNUT GRØSTAD

SINDRE ØSTREM

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 10.01.19 til 20.05.19.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): FRODE HAUG

Oppdragsgivers kontaktperson (navn): DAG SOLHAUG

Student(er) (signatur): Knut Grostad dato 16.01.19

Sindre Gystrem dato 16.01.19

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): Dag L Solhaug dato 16.01.19

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

D Vedlegg status underveis

D.1 Statusrapporter

Statusrapport ved bacheloroppgave

22.02.2019

Knut Grøstad | Sindre Østrem

Status per 22.02

Fremdriftsplan og planlegging:

Prosjektplanen er ferdig utformet, levert og godkjent av veileder.
Dermed er planlegging ferdigstilt etter tidsfristen satt 1.februar.
Ingen overordnet arkitektur er blitt satt enda.
Avtalen vi ønsker med Finn.no er ikke klar på dette tidspunkt.

Organisering av gruppens arbeid og ansvarsområder

Organisering av gruppens arbeid og ansvarsområder ble definert i prosjektplanen.
Per i dag så blir de krav og beskrivelser definert i prosjektplanen fulgt av gruppens medlemmer.

Klargjøring av problemstillingen

Problemstillingen er uendret siden utformelsen av prosjektplanen. Den vil mest sannsynlig endre seg etter hvor langt vi kommer i utviklingen og hvilke avtaler med eksterne api-leverandører vi får.

Løsningsmetode

Vi er godt igang med koding av web applikasjonen. Etter ca 15 arbeidsdager med prøving og feiling med Java og tilhørende rammeverk har vi en stabil backend modul med brukere og brukerrettigheter. I dette tidspunkt implementeres metoder for biler og annonser.
På front-end bruker vi angular som har vært en læringsprosses. Nå har vi laget en solid grunnmur for resten av siden. Per nå har vi registrering, innlogging, legge til og slette kjøretøy på siden vår.

Rapportskrivning

Vi er ikke startet med rapporten på dette tidspunkt.

Oppsummering av punktene over

Vi er godt i gang men har ett par ting som mangler. Avtalen med Finn burde settes på plass og en arkitektur burde bli satt som vi kan følge. Stort har vi holdt de tidsfrister vi har definert i prosjektplanen. Vi er muligens noen dager etter på utviklingsplanen definert i gantt-skjemaet. Grunnen til dette er en periode med sykdom i gruppen i uke 7.

Muligheter. Trusler/Problemer?

Et problem kan være om vi fortsetter å utvikle etter samme tempo som da vi startet med kodingen. Dette er ikke veldig sannsynlig ettersom vi lærer underveis og får en større forståelse av Java og Angular.
En trussel vi kan se for øyeblikket er at avtalen med Finn.no om integrasjon ikke går gjennom.

Motivasjon

Gruppens samarbeid fungerer bra. Kommunikasjon rundt organisering har også vært bra. Gruppens medlemmer møter og jobber når de skal, om ikke annet er avtalt/kommunisert.
En oppmuntrende faktor er at både Angular og Java kodingen har gått fra å være vanskelig og knotete til å bli forståelig og interessant å jobbe med.

Opplevelse av veilder

Vi er veldig fornøyde med Frode, vår veileder. Møtene tar sted når de skal og vi opplever god kommunikasjon rundt innleveringer og møter.
Oppdragsgiver, Dag har gitt oss ganske så frie tøyler når det gjelder utviklingen av applikasjonen. Dette gjør at vi ikke har hatt så mange møter med Dag. Vi er også fornøyde med Dag, både når det gjelder å svare på spørsmål og kommunikasjon.

Statusrapport ved bacheloroppgave

12.04.2019

Knut Grøstad | Sindre Østrem

Fremdriftsplan og planlegging:

Dette gjenstår:

- Backend:
    - Skrive ferdig unittester
    - Api dokumentasjon, mer utfyllende kommentarer, Java docs
    - Finn integrasjon, om vi får svar fra Finn.no og om tiden strekker til
- Frontend
    - Admin verktøy
        - Lag/oppdater/slett bruker
        - Lag/oppdater/slett selskap
    - Selskap side
        - Lag/oppdater/slett annonse
        - Lag/oppdater/slett/bildeopplasting bil
    - Filtrering av søk

- Rapportskrivning

Organisering av gruppens arbeid og ansvarsområder

Organisering av gruppens arbeid og ansvarsområder ble definert i prosjektplanen.
Per i dag så blir de krav og beskrivelser definert i prosjektplanen fulgt av gruppens medlemmer.

Klargjøring av problemstillingen

Den såkalte “beslutningsmodulen” får vi ikke tid til. Produktet som blir ferdig er en webside som Finn.no for salg av biler/kjøretøy med funksjonaliteter for brukere, selsskapssider som har tilgang til funksjoner som å registrere inn kjøretøy, lage en annonse av kjøretøyet på siden, slette og redigere kjøretøy og dens annonser. Opplastning av bilder som festes til kjøretøyet og dens annonse. Side med søkefunksjoner og filtre som skal gjøre det enklere å finne det man leter etter.

Rapportskrivning

Vi retter fokus på rapportskriving etter påske.

Muligheter. Trusler/Problemer?

Et problem kan være at vi kanskje må utvikle samtidig som vi skriver rapport for å få produktet ferdig. Ettersom det meste som gjenstår er frontend koding kan dårlig tid ha en negativ effekt på utseende og brukervennligheten på webapplikasjonen.

Motivasjon

Gruppens samarbeid fungerer bra. Kommunikasjon rundt organisering har også vært bra.
Gruppens medlemmer møter og jobber når de skal, om ikke annet er avtalt/kommunisert.
En oppmuntrende faktor er at både Angular og Java kodingen har gått fra å være vanskelig og knotete til å bli mer forståelig og interessant å jobbe med.

Opplevelse av veileder

Vi er veldig fornøyde med Frode, vår veileder. Møtene tar sted når de skal og vi opplever god kommunikasjon rundt innleveringer og møter. Vi er også veldig fornøyde med vår oppdragsgiver Dag, som har gitt oss ganske frie tøyler når det gjelder utviklingen av applikasjonen. Han har også vært flink til å tilpasse oppgaven i henhold til våre evner i Java.

Statusrapport ved bacheloroppgave

06.05.2019

Knut Grøstad | Sindre Østrem

Fremdriftsplan og planlegging:

Dette gjenstår:

- Backend:
  - Api dokumentasjon, mer utfyllende kommentarer, Java docs
- Frontend
  - Bilder fra database, da dette er en viktig del av en salgsside ønskes dette fikset før leveringen.
- Programmvaren generelt
  - Fullføre bedriftssider der bedrifter er knyttet til kjøretøy og kan generere annonser av disse. Per nå har vi ingen bedriftsside som planlagt som tar seg av det å lage annonser. Alle brukere har og per nå tilgang til å redigere alle kjøretøy i løsningen vår grunnet dette.
- Rapportskrivning
  - Introduksjonen og kravspesifikasjonen er ferdig. Resten av rapporten må skrives

Fokus fremover blir rapporten.

Organisering av gruppens arbeid og ansvarsområder

Organisering av gruppens arbeid og ansvarsområder ble definert i prosjektplanen.
Per i dag så blir de krav og beskrivelser definert i prosjektplanen fulgt av gruppens medlemmer.

Rapportskrivning

Vi har fått ned introduksjonen og kravspesifikasjonen vår, og må komme oss i gang med å skrive resten av rapporten. Vi skjønner vi har gitt oss selv dårlig tid til dette.

Muligheter. Trusler/Problemer?

Et problem er at vi ser vi ikke kommer helt i mål med vår løsning. Rapporten må skrives og vi får ikke mye tid til å ferdigstille selve programvaren vår

Motivasjon

Gruppens samarbeid fungerer bra. Kommunikasjon rundt organisering har også vært bra.
Gruppens medlemmer møter og jobber når de skal, om ikke annet er avtalt/kommunisert.
Motivasjon vår ligger nå mot å få skrevet ferdig rapporten, programvaren blir som den blir.

D.2 Referat fra møter

Referat fra møter med Dag

16.01

- MariaDB og TomCat
- Velge rammeverk selv, men holde seg til Java.
- Salgsmodul først
    - Pushe over data til feks finn og intern side
- Kravspec
    - Hva inn først, tilpass angående kjøretøy mot utstyr.
    - Mulig kontakt med finn og finne ut mulighetene rundt deres system
- Gode diagrammer om siden og dens sider
    - Vise at vi har god struktur

24.01

- Ta med besluttnigsmodulen
    - Klarer vi å matche krav til biler.
    - Bruk statistikk til det som er brukt før og gi forslag til nye.
    - Integrere e post anbud?
    - Type behov.
        - Størrselse
        - Miljø
        - Type dekk
        - 2-4 hjulstrekk
        - bombrikke
- Mål
    - Naturlig bredding av produktet de har
    - Caradmin skal bli et produkt som dekker hele kjøretøyholdet
        - Finne ut hva man trenger
        - Trenger man så mange biler?
            - Bruker man nokk biler.
- Salgsprosess
    - Leasing innad i kommunen
- FInn
    - Forklar at det er studentprosjekt. Det kommer ikke til å være noen kunder for det første.
    - Se på forslag på salgssider til biler. Kanskje det ligger noe der man kan bruke.
- Brukere
    - Selgere mer administrator for bedrift
    - Kjøp kan bare være gjestebrukere.
- Kommune website
    - “Embedd” tag for sider der de skal kunne vise fram sine biler til salgs.
    - Bare vise sine biler som er til salgs.
- Engelsk navnsetting
    - Kommentert på norsk

D.3 Timebok

	Knut	Beskrivelse	Sindre	Beskrivelse
10-01-2019	6	Første møte, orientering og begynnelse på planlegging	6	Første møte, orientering og begynnelse på planlegging
11-01-2019	5	Avtalt møte med etc, sendt cv/karakterer, lært litt java	5	Avtalt møte med etc, sendt cv/karakterer, lært litt java
12-01-2019				
13-01-2019				
Uke 2	11		11	
14-01-2019	5	Ble bedre kjent med java, klasser, error handling	5	Java, ajax, javaservlet
15-01-2019	6	Mer java	6	Forberede spørsmål til etc, java ajax
16-01-2019	7	Møte med etc, lest bacheloroppgaver, finn.no	7	Møte med etc. Prosjektplan. Finn.no mulighet
17-01-2019	7	Undersøkt rammeverk; angular, Springboot, JHipster	7	Se på angular, spring.
18-01-2019	2	Syk, kommuniserte med FINN.no, skrev på prosjektplan	4	Testet ut jhipster, spring. Prøving og feiling
19-01-2019				
20-01-2019				
Uke 3	27		29	
21-01-2019	7	Prosjektplan jobbing	6	Prosjektplan, fagområde, møte med Frode
22-01-2019	2	Prosjektplan jobbing	2	Prosjektplan, how to cite a site in latex
23-01-2019	3,4	Prosjektplan jobbing, undersøkte bil api'er	3,5	Spørsmål til ETC, bynt på risikodelen.
24-01-2019	6	Prosjektplan, møte etc	6	Prosjektplan, Etc møte.
25-01-2019	6	Prosjektplan	6	Prosjektplan
26-01-2019				
27-01-2019				
Uke 4	24,4		23,5	
28-01-2019	7	Prosjektplan	7	Frode, plan
29-01-2019	4	Prosjektplan	4	Plan
30-01-2019	2	Prosjektplan	2	Plan
31-01-2019	6	Prosjektplan	6	Plan
01-02-2019	3	Prosjektplan, leverer prosjektplan	3	Plan
02-02-2019				
03-02-2019				
Uke 5	22		22	
04-02-2019	3	Møte med frode, fikse prosjektplan	3	Småfiks på plan.
05-02-2019	5	Java/Angular/Springboot	5,5	Angular/Spring/Java, lage sideeee
06-02-2019	5	Bruker model, metoder	4,5	Angular
07-02-2019	6,5	Auth model, metoder	6,5	Angular, fake backend, user login&registration, vehicles
08-02-2019	3	Syk, login funksjon	6,5	Stirra meg ijæl på backend integrasjon, Angular
09-02-2019				
10-02-2019				
Uke 6	22,5		26	
11-02-2019	0	Syk	0	Syk
12-02-2019	5	login	0	Syk
13-02-2019	6	Bruker login	0	Syk
14-02-2019	5	Bruker registrering	0	Syk
15-02-2019	5	Feilsøking, autorisasjon	0	Syk
16-02-2019				
17-02-2019				
Uke 7	21		0	
18-02-2019	5	Unit test for bruker model	5	Angular
19-02-2019	6	Ferdigstilt api for bruker funksjonalitet	5	Angular
20-02-2019	4	Designer database	0	Syk
21-02-2019	0		0	Syk
22-02-2019	0		0	Syk
23-02-2019				
24-02-2019				
Uke 8	15		10	
25-02-2019	6	Database design, sette seg inn i JPA (bibliotek)	0	Syk
26-02-2019	5	Bil modell,funksjoner, tester	0	Syk
27-02-2019	5	Flere brukerfunksjoner, tester	5	Se på Backend og bli kjent med den. Login
28-02-2019	5	Endringer i autorisasjon	7	Knote med login og registrering mot backend.
01-03-2019				
02-03-2019				
03-03-2019				
Uke 9	21		12	

04-03-2019	5	Sessions og cookies	5	Angular, db
05-03-2019	5	Feilsøking login, vurdert auth biblioteker for java	5	Angular, vehicle detail, smått me notater til dag.
06-03-2019	6	Sett på oauth2 som alternativ til auth bibliotek	6	Møte med Dag, knote med bildealbum i angular
07-03-2019	4	Begynt å implementere oauth2	4	Angular, bildealbum
08-03-2019	2	Syk,Går bort fra oauth2, implemeterer token basert auth		
09-03-2019				
10-03-2019				
Uke 10	22		20	
11-03-2019	6	Implemeterer token basert auth	6	Bildealbum, angular
12-03-2019	5,5	Database endringer basert på FINN.no og ETC sitt design	5,5	Bildealbum, angular, gådd fra blob til file storage
13-03-2019	0		4	
14-03-2019	4,5	Implentert Brand, Model, TypeData	4,5	Søkefunksjon for vehicles
15-03-2019	0		0	
16-03-2019				
17-03-2019				
Uke 11	16		20	
18-03-2019	0	Syk med halsbetennelse hele uken	2	Angular søk & filter
19-03-2019	0	-	3	Angular søk & filter
20-03-2019	0	-	2	Angular søk & filter
21-03-2019	0	-	2	Angular søk & filter
22-03-2019				
23-03-2019				
24-03-2019				
Uke 12	0		9	
25-03-2019	5	Service klasser, lager api funksjoner	5	Generelt på front-end
26-03-2019	7	Service klasser til vehicle,listing,company	8	Generelt på front-end
27-03-2019	6	Api vehicle,listing,company	7,5	Generelt på front-end
28-03-2019	8	Migrert backend pga autorisasjons bug jeg ikke klarer å fikse	8	Generelt på front-end
29-03-2019			4	Generelt på front-end
30-03-2019				
31-03-2019				
Uke 13	26		32,5	
01-04-2019	7	Autorisasjon tester, API tester	6,5	Generelt på front-end
02-04-2019	6	Service og model tester	5	Generelt på front-end
03-04-2019	3	Modell endringer	3	Bilder
04-04-2019	5	Bildeopplasting	7	Bilder, byttet fra filsystem til blob igjen
05-04-2019				
06-04-2019				
07-04-2019				
Uke 14	21		21,5	
08-04-2019	7	Database endringer, Frontend feilsøking	7,5	Bilder
09-04-2019	8	Frontend brukerprofil, sette seg inn i angular	6	Bilder
10-04-2019	0	Syk	3	Bilder
11-04-2019	6	Filopplastning	2	Bilder
12-04-2019				
13-04-2019				
14-04-2019				
Uke 15	21		18,5	
15-04-2019		Påske	0	Påske
16-04-2019		Påske	0	Påske
17-04-2019		Påske	0	Påske
18-04-2019		Påske	0	Påske
19-04-2019		Påske	0	Påske
20-04-2019		Påske	0	Påske
21-04-2019		Påske	0	Påske
Uke 16	0		0	
22-04-2019		Påske	0	Påske
23-04-2019	0			
24-04-2019	6	Opprettet rapport, begynt på analyse	6	Rapport
25-04-2019	0		4	Rapport
26-04-2019	3	bruker profil, jobbet hjemmefra, syk	0	
27-04-2019	0			

28-04-2019				
Uke 17	9		10	
29-04-2019	6	Bruker profil	6	Småfiks angular
30-04-2019	5	Bildeopplastning	5	Småfiks angular
01-05-2019	6	Kravspesifikasjon, use cases diagram	2	Rapport
02-05-2019	6	Kravspesifikasjon, use cases diagram	6	Rapport
03-05-2019	3	Syk, Introduksjon	3	Rapport
04-05-2019				
05-05-2019				
Uke 18	26		22	
06-05-2019	5	Rapport planlegging	5	Møte med Frode. Planlegging rapport. misuse case
07-05-2019	6	Analyse delen	8	Misuse case diagrammer og forklaringer med mitigasjon
08-05-2019	7	Analyse delen	4	Rapport
09-05-2019	6	Analyse delen	4	Rapport
10-05-2019	6	Arkitektur figurer	3	Rapport
11-05-2019	2	Arkitektur figurer	6	Rapport
12-05-2019	2	Arkitektur figurer	8	Rapport
Uke 19	34		38	
13-05-2019	6	Siste møte med frode, arkitektur del	5	Rapport
14-05-2019	5	Ferdigstilt arkitektur og design del	10	Rapport, angular
15-05-2019	5	Implementasjon	8	Rapport, fix på bilder
16-05-2019	6	Implementasjon	7	Rapport
17-05-2019	6	Implementasjon	3	Rapport
18-05-2019	7	Finpuss, bedre analysedel, definisjoner	6	Rapport
19-05-2019	8	Diskusjon og konklusjon, sammendrag, levere	8	Rapport
Uke 20	43		47	
20-05-2019				
Total	381,9		372	

E Ekstern datastrutur

E.1 Finn.no DTD-skjema

Listing E.1: Finn.no DTD

```
<!-- DOCTYPE IAD.IF Car Finntech

Changelogg:
23.06.2006 Iad-version created   EAS
26.03.2009 Added the REREGISTRATION_EXEMPTION field fibeulve
-->
<!-- Definition of the 'top-element'; A transfer is -->
<!-- a IAD-if element with one or more OBJECTs -->
<!-- Top element -->
<!ELEMENT IAD.IF.NEWCAR (HEAD, OBJECT+)>
<!-- The element HEAD identifies a partne and a provider -->
<!ENTITY % HEAD SYSTEM "http://www.iad.no/dtd/IADIF-head20.dtd">
%HEAD;
<!ELEMENT OBJECT (OBJECT_HEAD, NEWCAR)>
<!-- OBJECT_HEAD- definition is common in IADIF-objecthead20.dtd -->
<!ENTITY % OBJECT_HEAD SYSTEM "http://www.iad.no/dtd/IADIF-objecthead20.dtd">
%OBJECT_HEAD;
<!-- Information on car -->
<!ELEMENT NEWCAR (CAR_MODEL, YEAR_MODEL,BODY_TYPE,
    REGISTRATION_CLASS, MILEAGE, MOTOR_PRICE,
    EXTERIOR_COLOUR_MAIN, EXTERIOR_COLOUR?,
    INTERIOR_COLOUR?, EQUIPMENT*, NO_OF_DOORS?,
    NO_OF_SEATS?, ENGINE, TRANSMISSION,
    SIZE_OF_BOOT?, WEIGHT?, WHEEL_DRIVE,
    DESCRIPTION?, MOREINFO*, NO_OF_OWNERS?, REGNO?,
     WARRANTY?, WARRANTY_DURATION?,
    WARRANTY_DISTANCE?, CONTACT, CAR_LOCATION?,
    CAR_SALESFORM?, CAR_SERVICE_HISTORY?)>
<!-- Car make/model -->
<!ELEMENT CAR_MODEL (MAKE, MODEL, MODEL_SPECIFICATION ?)>
<!-- The Mark must match a reference value -->
<!ELEMENT MAKE (#PCDATA)>
<!-- The Model must match a reference value -->
<!ELEMENT MODEL (#PCDATA)>
<!-- better spec of model, e.g. Turbo -->
<!ELEMENT MODEL_SPECIFICATION (#PCDATA)>
```

```
<!-- 4 digits -->
<!ELEMENT YEAR_MODEL (#PCDATA)>
<!-- Registration month, e.g 10.97 -->
<!ELEMENT REGISTRATION_FIRST (#PCDATA)>
<!-- Tells the body type of the car, predefined reference values -->
<!ELEMENT BODY_TYPE (#PCDATA)>
<!-- Tells registration class where the car currently belongs.Predefined set of reference values -->
<!ELEMENT REGISTRATION_CLASS (#PCDATA)>
<!-- Integer, km as unit -->
<!ELEMENT MILEAGE (#PCDATA)>
<!-- Price information -->
<!ELEMENT MOTOR_PRICE ((TOTAL | NET), REGISTRATION, CURRENCY?)>
<!ATTLIST MOTOR_PRICE
   REGISTRATIONTAX_INCLUDED (yes | no) #IMPLIED
   REREGISTRATION_EXEMPTION (yes | no) #IMPLIED
   ROADTAX_INCLUDED (yes | no) #IMPLIED
   VAT_INCLUDED (yes | no) #IMPLIED>
<!ELEMENT TOTAL (#PCDATA)>
<!ELEMENT NET (#PCDATA)>
<!ELEMENT REGISTRATION (#PCDATA)>
<!-- Currency for all prices ISO 3 letter code eg. NOK -->
<!ELEMENT CURRENCY (#PCDATA)>
<!-- Size, color etc. -->
<!-- Main car colour -->
<!ELEMENT EXTERIOR_COLOUR_MAIN (#PCDATA)>
<!-- Spesific colour description -->
<!ELEMENT EXTERIOR_COLOUR (#PCDATA)>
<!ELEMENT INTERIOR_COLOUR (#PCDATA)>
<!ELEMENT NO_OF_DOORS (#PCDATA)>
<!ELEMENT NO_OF_SEATS (#PCDATA)>
<!ELEMENT SIZE_OF_BOOT (#PCDATA)>
<!-- Integer kg as unit -->
<!ELEMENT WEIGHT (#PCDATA)>
<!-- EQUIPMENT - several occurrences may be included by repeating the
    tag. Search_value should match a reference value, description_value is free text -->
<!ELEMENT EQUIPMENT (#PCDATA)>
<!ATTLIST EQUIPMENT SEARCHKEY CDATA #IMPLIED>
<!-- Engine -->
<!ELEMENT ENGINE (EFFECT?, VOLUME?, FUEL)>
<!-- Effect with HP as unit -->
<!ELEMENT EFFECT (#PCDATA)>
<!-- Volume in litre, numbers with . as decimal -->
<!ELEMENT VOLUME (#PCDATA)>
<!-- eg. (petrol,diesel,gas,electricity) -->
<!ELEMENT FUEL (#PCDATA)>
<!-- Searchkey should match a reference value, a description as free text should be within the
```

```
    ↪ tag -->
<!ELEMENT TRANSMISSION (#PCDATA)>
<!ATTLIST TRANSMISSION SEARCHKEY (manual|automatic|
    ↪ semiautomatic) #IMPLIED >
<!-- Searchkey should match a reference value, a
    ↪ description as free text should be within the
    ↪ tag -->
<!ELEMENT WHEEL_DRIVE (#PCDATA)>
<!ATTLIST WHEEL_DRIVE SEARCHKEY (frontwheeledrive|
    ↪ rearwheeledrive|fourwheeledrive) #IMPLIED >
<!-- General description of car -->
<!ELEMENT DESCRIPTION (#PCDATA)>
<!-- Add an external url, urltext will become the
    ↪ clickable text in the webpage -->
<!ELEMENT MOREINFO (URL, URLTEXT?)>
<!ELEMENT URLTEXT (#PCDATA)>
<!-- Information on "take_over" and owner -->
<!ELEMENT REGNO (#PCDATA)>
<!-- Text max 15 characters -->
<!ELEMENT WARRANTY (#PCDATA)>
<!-- Number of months -->
<!ELEMENT WARRANTY_DURATION (#PCDATA)>
<!-- Number of km -->
<!ELEMENT WARRANTY_DISTANCE (#PCDATA)>
<!-- Text max 255 characters -->
<!ELEMENT NO_OF_OWNERS (#PCDATA)>
<!-- Integer < 20 -->
<!-- Information on CONTACT -->
<!ELEMENT CONTACT (NAME?, PHONE?, MOBILE?, EMAIL?,
    ↪ FAX?, URL?)>
<!ATTLIST CONTACT PHONESALESRESERVATION (yes|no) "no"
    ↪ >
<!ELEMENT NAME (#PCDATA)>
<!-- Text -->
<!ELEMENT PHONE (#PCDATA)>
<!-- Text -->
<!ELEMENT MOBILE (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!-- Text -->
<!ELEMENT FAX (#PCDATA)>
<!ELEMENT URL (#PCDATA)>
<!-- Text -->
<!ELEMENT CAR_LOCATION (#PCDATA)>
<!-- Text -->
<!ELEMENT CAR_SALESFORM (#PCDATA)>
<!-- Text -->
<!ELEMENT CAR_SERVICE_HISTORY (#PCDATA)>
<!-- Text -->
<!ELEMENT CO2 (#PCDATA)>
```

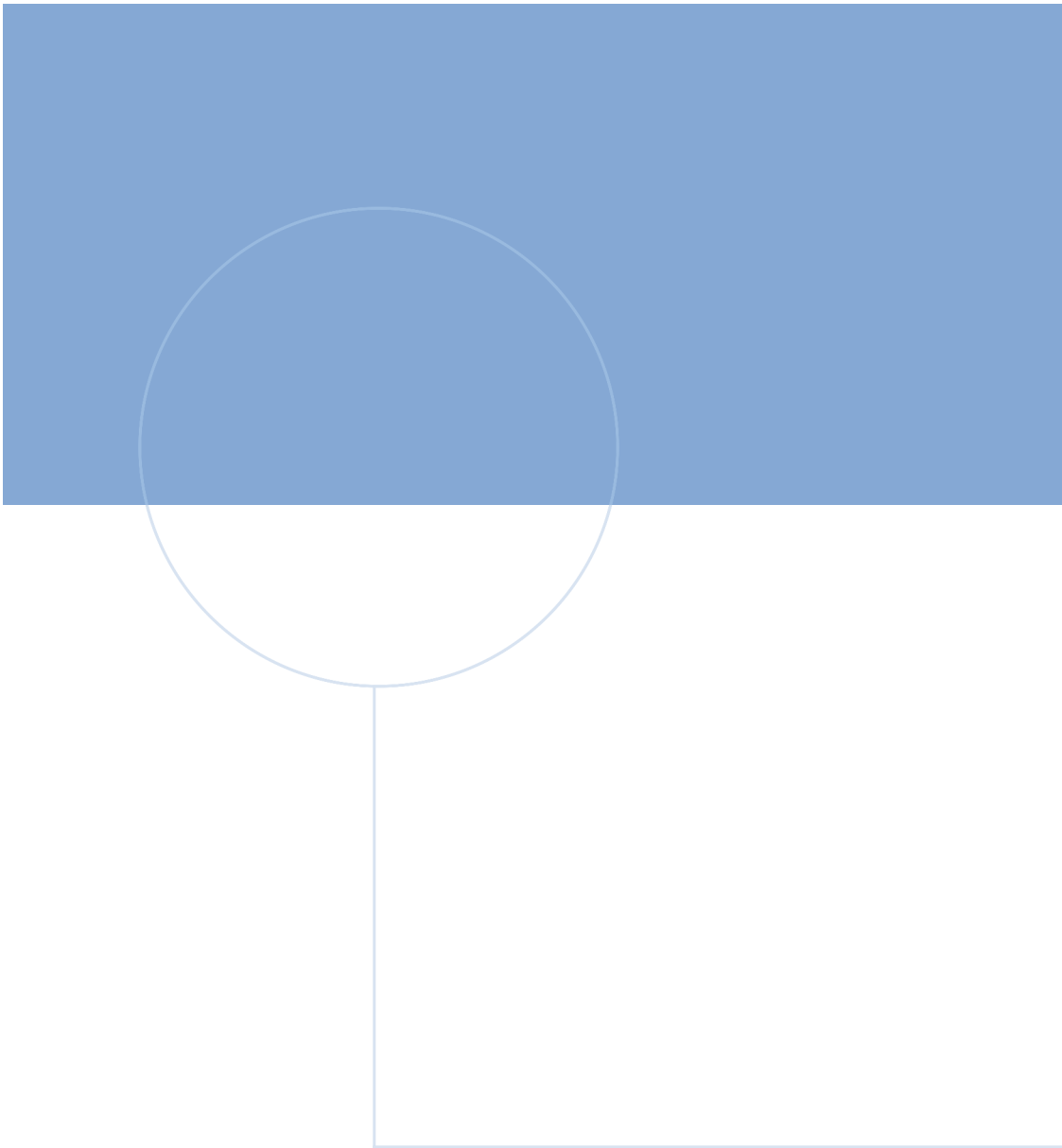
E.2 CarAdmin databasestruktur

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET NAMES utf8 */;
/*!50503 SET NAMES utf8mb4 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

CREATE TABLE IF NOT EXISTS `assetdeal` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `assetno` int(11) NOT NULL,
  `listprice` decimal(10,2) DEFAULT NULL,
  `costprice` decimal(10,2) DEFAULT NULL,
  `restvalue` decimal(10,2) DEFAULT NULL,
  `leasestop` datetime DEFAULT NULL,
  `financecompany` int(11) DEFAULT NULL,
  `rentalcustomer` int(11) DEFAULT NULL,
  `inactive` tinyint(4) NOT NULL DEFAULT 0,
  `rentalprodno` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `car` (
  `carno` int(11) NOT NULL AUTO_INCREMENT,
  `removed` datetime DEFAULT NULL,
  `cardID` varchar(11) DEFAULT NULL,
  `locationID` int(11) DEFAULT NULL,
  `created` datetime DEFAULT NULL,
  `updated` datetime DEFAULT NULL,
  `standaruser` int(10) NOT NULL,
  `role` int(10) unsigned DEFAULT NULL,
  `officeHourFrom` varchar(5) DEFAULT NULL,
  `officeHourTo` varchar(5) DEFAULT NULL,
  `includeWeekends` tinyint(1) DEFAULT NULL,
  `tenth` datetime DEFAULT NULL,
  `warningTimeFrom` varchar(20) DEFAULT NULL,
  `warningTimeTo` varchar(20) DEFAULT NULL,
  `warningTimeIncludeWeekends` tinyint(1) DEFAULT NULL,
  `assettype` int(11) DEFAULT 1,
  `costunit` int(11) DEFAULT 0,
  `odometerHour` double DEFAULT NULL,
  `damageprofile` int(10) DEFAULT NULL,
  `invoiceReceiverId` int(11) DEFAULT NULL,
```

```
  `invoiceDealId` int(11) DEFAULT NULL,
  `externId` int(11) DEFAULT NULL,
  `externUpdated` datetime DEFAULT NULL,
  `costReferenceId` int(11) DEFAULT NULL,
  `assetModelId` int(11) DEFAULT NULL,
  `externalCarId` varchar(50) DEFAULT NULL,
  `statisticsUpdated` datetime DEFAULT NULL,
  `vatcode` int(11) NOT NULL DEFAULT 0,
  `revision` int(11) NOT NULL DEFAULT 0,
  PRIMARY KEY (`carno`)
) ENGINE=InnoDB AUTO_INCREMENT=545 DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `carequipment` (
  `carID` varchar(20) NOT NULL DEFAULT '',
  `equipmentID` int(11) DEFAULT NULL,
  `price` float DEFAULT NULL,
  PRIMARY KEY (`carID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `general` (
  `carID` varchar(30) NOT NULL DEFAULT '',
  `nick` varchar(30) NOT NULL DEFAULT '',
  `brand` varchar(30) DEFAULT NULL,
  `color` varchar(30) DEFAULT NULL,
  `make` varchar(30) DEFAULT NULL,
  `model` varchar(30) DEFAULT NULL,
  `tires` varchar(30) DEFAULT NULL,
  `maxfuel` int(11) DEFAULT NULL,
  `fuelpin` varchar(30) DEFAULT NULL,
  `department` varchar(30) DEFAULT NULL,
  `posDesc` text DEFAULT NULL,
  `insuranceCost` float DEFAULT NULL,
  `insuranceCostTimescale` int(11) DEFAULT NULL,
  `lockerID` int(11) DEFAULT NULL,
  `chassisID` varchar(25) DEFAULT NULL,
  `maxKmTank` int(11) DEFAULT NULL,
  `fuelSystem` varchar(20) DEFAULT NULL,
  `gearSystem` varchar(20) DEFAULT NULL,
  `drive` varchar(20) DEFAULT NULL,
  `interior` varchar(25) DEFAULT NULL,
  `lockerNo` int(11) NOT NULL DEFAULT 1,
  `carGroup` int(11) DEFAULT NULL,
  `carNo` int(11) NOT NULL DEFAULT 0,
```

```
  `ownership` varchar(10) NOT NULL DEFAULT 'Unknown',
  `barcode` varchar(500) DEFAULT NULL,
  `AccountingRef` int(11) DEFAULT NULL,
  PRIMARY KEY (`carNo`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `maintenance` (
  `carNo` int(11) NOT NULL DEFAULT 0,
  `leaseEndDate` datetime DEFAULT NULL,
  `leaseStartDate` datetime DEFAULT NULL,
  `replacedByVehicle` int(11) NOT NULL DEFAULT 0,
  `replacingVehicle` int(11) NOT NULL DEFAULT 0,
  `originLocation` int(11) NOT NULL DEFAULT 0,
  `originVehicleGroup` int(11) NOT NULL DEFAULT 0,
  `originLockerNumber` int(11) NOT NULL DEFAULT 0,
  `originLockerDoor` int(11) NOT NULL DEFAULT 0,
  `sumKm` float NOT NULL DEFAULT 0,
  `sumHours` float NOT NULL DEFAULT 0,
  `avg_startDate` datetime DEFAULT NULL,
  `avg_startKm` float NOT NULL DEFAULT 0,
  `avg_startHour` float NOT NULL DEFAULT 0,
  PRIMARY KEY (`carNo`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `technical` (
  `carNo` int(11) NOT NULL DEFAULT 0,
  `currentPosition` varchar(30) NOT NULL DEFAULT '',
  `defaultPosition` varchar(30) DEFAULT NULL,
  `fuel` float NOT NULL DEFAULT 0,
  `host` varchar(30) NOT NULL DEFAULT '',
  `mileage` double DEFAULT NULL,
  `stateRef` int(11) NOT NULL DEFAULT 0,
  `avgfueluse` float DEFAULT NULL,
  `locked` tinyint(1) NOT NULL DEFAULT 0,
  `notdriveable` tinyint(1) NOT NULL DEFAULT 0,
  `co2` double NOT NULL DEFAULT 0,
  `nox` double NOT NULL DEFAULT 0,
  `environmentClass` varchar(100) DEFAULT '',
  `particlefilter` tinyint(4) NOT NULL DEFAULT 0,
  PRIMARY KEY (`carNo`),
  KEY `stateRef` (`stateRef`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
/*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */;
/*!40014 SET FOREIGN_KEY_CHECKS=IF(@OLD_FOREIGN_KEY_CHECKS IS NULL, 1,
@OLD_FOREIGN_KEY_CHECKS) */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```



NTNU

Kunnskap for en bedre verden