NTNU
Norwegian University of
Science and Technology

IoT-nøkkelskap - åpning av elektronisk nøkkelskap med mobilapplikasjon

Forfattere
Markus Sørlien Johansen
Fredrik Hatletvedt
Andreas Bradahl

Bachelor i ingeniør - data
20 ECTS
Institutt for Datateknikk og Informatikk
Norges teknisk-naturvitenskapelige universitet,

16.05.2018

Veileder Frode Haug

Sammendrag av Bacheloroppgaven

Tittel:	**IoT-nøkkelskap - åpning av elektronisk nøkkelskap med mobilapplikasjon**
Dato:	16.05.2018
Deltakere:	Markus Sørlien Johansen Fredrik Hatletvedt Andreas Bradahl
Veiledere:	Frode Haug
Oppdragsgiver:	Electric Time Car
Kontaktperson:	Dag Solhaug, dag@electrictimecar.com, +47 90101344
Nøkkelord:	IoT, Bluetooth, Xamarin.Forms, Raspberry Pi, Mikrocontroller, IMT, Bacheloroppgave
Antall sider:	108
Antall vedlegg:	8
Tilgjengelighet:	Åpen

Sammendrag:

Denne rapporten beskriver moderniseringen av oppdragsgivers (Electric Time Car AS) programvare *CarAdmin*. CarAdmin har mange bruksområder, men dette prosjektet har tatt utgangspunkt i CarAdmins bilparkfunksjon. Denne lar brukerne av en bedrifts bilpark (de ansatte) ta ut og levere bilnøkler ved behov, samtidig som CarAdmin til enhver tid har oversikt over hvilke biler som er i bruk, og hvem som bruker dem. Målet med moderniseringen er at CarAdmin-brukerne ikke skal trenge å logge seg inn på datamaskinen som i dag står plassert ved siden av bilnøkkelskapene, men istedet kunne bruke sin egen smarttelefon og CarAdmins mobilapplikasjon for å hente ut og levere bilnøkler. For å kunne kommunisere mellom brukernes smarttelefoner og nøkkelskapene, så måtte en mikrocomputer fungere som et trådløst tilgangspunkt på nøkkelskapet. Oppdragsgiver ønsket at Bluetooth skulle brukes som kommunikasjonsteknologi, og denne kommunikasjonen ble dermed implementert både på mikrocomputeren og på mobilapplikasjonen. CarAdmins mobilapplikasjon var ikke ferdig da dette prosjektet startet, og det ble derfor nødvendig å utvikle en konseptapplikasjon for å teste systemet underveis. Nøkkelskapenes låsemekanismer ble styrt av en mikrokontroller. Logikken for styring av mikrokontrolleren ble også implementert på mikrocomputeren. Resultatet av dette prosjektet er en «pakke» med logikk og kode som på sikt kan implementeres i CarAdmin.

Summary of Graduate Project

Title:	**IoT-nøkkelskap - åpning av elektronisk nøkkelskap med mobilapplikasjon**
First Date:	First 16.05.2018
Authors:	Markus Sørlien Johansen Fredrik Hatletvedt Andreas Bradahl
Supervisor:	Frode Haug
Employer:	Electric Time Car
Contact Person:	Dag Solhaug, dag@electrictimecar.com, +47 90101344
Keywords:	IoT, Bluetooth, Xamarin.Forms, Microcontroller, Raspberry Pi, IMT, Thesis
Pages:	108
Attachments:	8
Availability:	Open

Abstract:

This report describes the modernization of the contractor's (Electric Time Car AS) software *CarAdmin*. CarAdmin has many applications, but this project has used CarAdmin's carpool functionality as a basis. This functionality lets the users of the system (the employees of a firm) retrieve and return car keys, while CarAdmin at the same time has an overview over which cars are available, which employee has which car etc. The goal of the modernization is that the users of CarAdmin should not have to use the external computer that is placed by the key lockers today to log in and retrieve a car key, but instead use their own smart phones and the CarAdmin mobile application. To communicate between the smart phones and and the key lockers, a microcomputer has to function as a wireless access point on the locker. Our contractor wanted Bluetooth to be used as the technology for communication, so this was implemented in both the microcomputer and on the mobile application. CarAdmin's mobile application was unfinished at the start of this project, making it necessary to develop a concept application to test the system while working on it. A microcontroller manages the key lockers' opening and locking mechanisms, and the logic for this had to be implemented on the microcomputer. The result of this project is a «package» of logic and code which eventually can be implemented in CarAdmin.

Forord

Vi vil gjerne takke vår oppdragsgiver Electric Time Car AS og deres utviklerteam for å gi oss muligheten til å jobbe for dem under prosjektperioden og for å være åpne for endringer vi har måttet gjøre i prosjektet.

Vi vil også gjerne takke Frode Haug, vår veileder under prosjektperioden, for å hjelpe og veilede under prosjektet, samt også for å stille seg tilgjengelig under hele perioden.

Innhold

Figurer

Tabeller

Listings

1 Innledning

1.1 Bakgrunn

Electric Time Car AS (heretter kalt «ETC» eller «oppdragsgiver») er en softwareleverandør som holder til på Gjøvik. De leverer løsninger for deling av transportsmidler og ressursstyring innen bilhold til bedrifter og offentlig sektor. Brukerne av systemet forholder seg til nøkkelskap hvor bilnøkler hentes og leveres før og etter bruk. CarAdmin[1], som er deres hovedprodukt, samler spredte bilparker i felles virtuelle parkeringsplasser som enkelt lar en bedriftsleder eller lignende ha full oversikt over bedriftens bilpark til enhver tid. I dag kommuniserer brukerne med nøkkelskapene via en datamaskin som bruker lavnivåprotokoller som f.eks. RS485, TTL og USB. I disse IoT-tidene[2] er det likevel ønsket å modernisere dette, og tilby et system hvor brukerne kan hente ut og levere nøkler ved hjelp av mobiltelefonen sin. Det er det dette prosjektet, modernisering av oppdragsgivers eksisterende løsning, har som hovedmålet.

1.2 Avgrensning

Oppdragsgiver ønsket å gjøre sin nåværende løsning mer IoT-vennlig. I dag har ETC utplassert en fysisk datamaskin ved hvert av sine nøkkelskap hvor brukeren som skal hente ut en bilnøkkel må logge seg på. Moderniseringen som var ønsket ville tillate brukerne av CarAdmin å hente ut en bilnøkkel ved bruk av sin smarttelefon og CarAdmin-applikasjonen.

Som følge av at CarAdmin-systemet allerede er et etablert produkt, har prosjektet naturlige avgrensninger og begrensninger. Blant annet har ETC allerede en egen CarAdmin-brukerdatabase med tilhørende innlogging-/autentiseringsmekanismer. ETC har også utviklet en egen algoritme for utvelgelse av bil (dette skjer i sanntid når brukeren trenger bilen). Oppdragsgiver ønsket at denne algoritmen fortsatt ble benyttet, noe som betydde at det ikke ble nødvendig å tenke på hvordan «riktig» bil ble valgt. Det har vært diskutert om denne logikken kunne og burde ligge på mikrocomputeren, men på grunn av tidsbegrensningen i prosjektet ønsket oppdragsgiver å stå for dette selv.

Selve mobilapplikasjonen var enda under utvikling av ETC. Dette betydde at prosjektet ikke ville inkludere utvikling av en selvstendig applikasjon, men at løsningen heller skulle implementeres i deres applikasjon.

[1] www.caradmin.no
[2] Internet of Things

1.3 Oppgavedefinisjon

Oppgaven gikk ut på å gjøre det mulig å kommunisere med nøkkelskap fra en smarttelefon. Det skulle også finnes og tilpasses hardware til dette formålet. Hardwaren skulle programmeres slik at nøkkelskapene, oppdragsgivers server og mobilapplikasjonen kunne kommunisere sammen. Det måtte dermed også legges til funksjonalitet på mobilapplikasjonen for å utføre skanning og oppkobling mot nøkkelskapene. Oppgaven hadde visse krav som *skulle* være på plass i den ferdige løsningen. Kravene som oppdragsgiver hadde satt er som følger:

- ETCs uttaksrutiner/-algoritme ligger på deres server, og vår løsning må kommunisere med denne for åpning av riktig skap
- Skapet kan ikke åpnes av en bruker som ikke er i nærheten av det.
- Programvaren i mikrocomputeren må kommunisere med systemene ETC bruker i dag, blant annet server, mikrokontroller og applikasjon.
- Løsningen skal kunne tilpasses mot en iOS-/Android-applikasjon som allerede er under utvikling av ETC.
- Løsningen skal benytte seg av Bluetooth-teknologi for kommunikasjon mellom applikasjon og mikrocomputer.
- ETC ønsker plug-and-play-hardware, som i praksis betyr at komponentene skal være klare til bruk (det er for eksempel uaktuelt å lodde på komponentene).
- Det ferdige produktet skal være enkelt å montere hos kunden.

Den ferdige løsningen skal, som vist i figur 1, erstatte den gamle løsningen, der en ekstern datamaskin benyttes (figur 1a), med en løsning som på figur 1b. Løsningen skal få plass på innsiden av et nøkkelskap, og gjøre det mulig å kobles til via Bluetooth med mobil.

1.4 Formål

Formålet med prosjektet var i hovedsak å modernisere den allerede eksisterende løsningen for kommunikasjon mellom komponentene som var benyttet under uttaksrutinene av kjøretøy.

Vi ønsket denne oppgaven ettersom den ville gi oss god erfaring med å jobbe mot eksisterende systemer, noe som sannsynligvis vil komme godt med i fremtidig arbeidsliv. I tillegg fant vi oppgaven godt egnet for dataingeniører, da vi kunne benytte tilegnet kunnskap gjennom tre års studier. Allerede etter første møte med oppdragsgiver hørtes dette ut som en spennende og lærerik oppgave som vi så frem til å komme i gang med.

(a) Dagens løsning.

(b) Mål for løsning.

Figur 1: Før og etter: brukeren slipper nå å forholde seg til datamaskinen på det øverste bildet.

1.5 Læringsutbytte

I tillegg til læringsutbyttet beskrevet i *BIDAT39*[3]s emnebeskrivelse, ble det definert følgende egne læringsmål:

- Prosjektplanlegging
- Bruk og programmering av mikrocomputere.
- Tilegne kunnskap om utvikling for mobile plattformer (fortrinnsvis iOS og Android)
- Få erfaring med programmering mot Bluetooth-teknologi.
- Tilegne kunnskap om sikkerhet ift. «Internet of Things»-enheter.
- Få erfaring i å programmere mot eksisterende APIer og løsninger.
- Bruk og kombinasjon av systemutviklingsmetodikker (*Fossefall* og *Scrum*).

1.6 Målgruppe

1.6.1 Målgruppe for produkt

Det endelige produktet er ment for oppdragsgiver, for kommunikasjon mellom komponentene som blir brukt i deres allerede eksisterende system.

[3]BIDAT39: Bacheloroppgave for dataingeniører

1.6.2 Målgruppe for rapport

Rapporten er skrevet for alle som har interesse av emnet, og også de som ønsker å gjenskape produktet. Vi har forsøkt å benytte et så enkelt språk som mulig, slik at det ikke skal være nødvendig med bakgrunnskunnskaper innenfor IT for å forstå rapporten. Fremmedord blir forklart i fotnoter, samt i en «fremmedordbok» (vedlegg B).

1.7 Akademisk bakgrunn

Prosjektgruppen består av: Andreas Bradahl, Fredrik Hatletvedt og Markus Sørlien Johansen. Alle gruppemedlemmer har ulike bakgrunner og erfaringer, men felles for alle er at vi har gjennomført det treårige bachelorstudiet for dataingeniører ved NTNU Gjøvik. Enkelte av de obligatoriske emnene har endret seg under den aktive utdanningsplanen til medlemmene under utdanningen, men ettersom disse fagene har omtrent de samme aspektene, har de gitt samme læringsutbytte. Utover de obligatoriske emnene har hvert medlem også deltatt i tre valgfag. Fredrik og Markus har hatt programvaresikkerhet, mens Andreas har hatt objektorientert systemutvikling. Gruppemedlemmene har med dette utarbeidet kunnskaper og erfaringer innenfor blant annet C++, C, Go og Python, som vi ser for oss vil være nyttig i dette prosjektet.

1.8 Rammer

1.8.1 Tidsrammer

- NTNU har satt en tidsramme fra 10. januar til 16. mai for ferdigstilling av prosjektet.
- ETC ønsker at grunnfunksjonaliteten i systemet skal være ferdig til midten av april, slik at ferdigstillingen av rapporten ikke skal gå utover arbeidet på det faktiske produktet.
- NTNU har satt krav om tre statusrapporter som leveres til veileder for å se prosjektets tilstand og fremdrift slik at det kan sammenlignes mot foreliggende planer.

1.8.2 Tekniske rammer

- Oppdragsgiver foretrekker at det benyttes Java som programmeringsspråk der det passer, men er åpne for alternativer dersom det er grunner til det.

1.8.3 Økonomiske rammer

- NTNU Gjøvik bistår ikke med økonomiske utlegg som oppstår i prosjektperioden.

1.9 Prosjektorganisering

I en liten gruppe hvor alle tilsynelatende har det samme grunnlaget, vil det i praksis bety at ethvert gruppemedlem (inkl. prosjektleder) er «allround-arbeidere», med det samme ansvaret for at prosjektet gjennomføres etter planen. Det er likevel noen definerte roller i prosjektet. Så godt som det har latt seg gjøre er Scrum[4]-terminologi blitt benyttet, både ift. roller og andre relevante oppgaver/hendelser. I visse tilfeller ble noe av dette ungått - prosjektet har f.eks. ingen *Scrum Master*, som i Scrum er den som har ansvar for opplæring og guiding i korrekt utførelse av Scrum.

Følgende roller er definert i prosjektet:

- **Prosjektleder:** Fredrik Hatletvedt
  Prosjektlederen har ansvar for å fordele oppgaver, samt passe på at tidsfrister blir holdt. Prosjektleder har også hovedansvaret for kommunikasjon mellom gruppa og oppdragsgiver/veileder.
- **Resterende gruppemedlemmer:** Andreas Bradahl og Markus Sørlien Johansen
  Fungerer i praksis som «allround-arbeidere», men også med ansvar for at prosjektet gjennomføres som planlagt.
- **Product Owner:** ETC v/Dag Solhaug
  Product Owner er en Scrum-rolle som i prosjektets tilfelle kan oversettes fritt til «oppdragsgiver». Han skal være behjelpelig under utviklingen av løsningen ved å uttrykke funksjonelle ønsker/krav, samt gi konstruktive tilbakemeldinger underveis i produktutviklingen.
- **Veileder:** Frode Haug
  Veilederens rolle er å være behjelpelig i forhold til prosjektets gang, ved å komme med råd og pekepinner for en så god prosess som mulig (hovedsakelig ift. selve rapporten).

1.10 Rapport

1.10.1 Ordbruk

- «ETC» og «oppdragsgiver» brukes om hverandre.
- «Raspberry Pi» og «mikrocomputer» brukes om hverandre.
- «Mikrokontroller» og «kontroller» brukes om hverandre.
- Det benyttes i all hovedsak engelsk når det gjelder informatikkterminologi (f.eks. «Bluetooth», og ikke «Blåtann»).
- Enkelte av teknologiene og verktøyene som er benyttet i prosjektet bruker liten forbokstav i navnet sitt. I så fall skrives navnet med liten forbokstav også i denne rapporten.

[4] Scrum: arbeidsmetodikk som er mye brukt i programvareutvikling.

1.10.2 Rapportstruktur

Rapporten består av sju hovedkapitler med egne underkapitler. Strukturen er gjort på følgende måte:

1.0 Innledning
Innledningsdelen inneholder bakgrunnen og grunnlaget for selve oppgaven. Organisering av prosjektet, rammer og prosjektets deltakere dekkes også her.

2.0 Kravspesifikasjon
Kapittelet inneholder de funksjonelle og ikke-funksjonelle kravene som er satt til løsningen..

3.0 Analyse
Analyser gjort før og under utvikling for vurdering av ulike aspekter ved oppgaven.

4.0 Design og arkitektur
Viser hvordan systemet kommuniserer, der det er beskrevet i detalj hvilken systemarkitektur som er benyttet og hvordan de ulike delene er designet.

5.0 Implementering
Kapittelet beskriver implementeringen av løsningen, med beskrivelser av språk, programmer og teknologier som ble benyttet.

6.0 Testing og kvalitetssikring
Her er det beskrevet hvordan de forskjellig delene av løsningen ble testet, og hvilke verktøy som ble benyttet.

7.0 Avslutning
I dette kapittelet evalueres arbeidet som er gjort i prosjektet, hvor det også diskuteres og drøftes om hva som kunne ha blitt gjort annerledes, i tillegg til resultater og hva som bør jobbes videre på.

Vedlegg
Vedlegg for rapporten.

2 Kravspesifikasjon

I dette kapittelet gjennomgås det hvilke krav om funksjonalitet som er satt til den ferdige løsningen. Dette innebærer funksjonelle og supplementære krav, i tillegg til use case-beskrivelser som ytterligere vil beskrive funksjonaliteten i systemet.

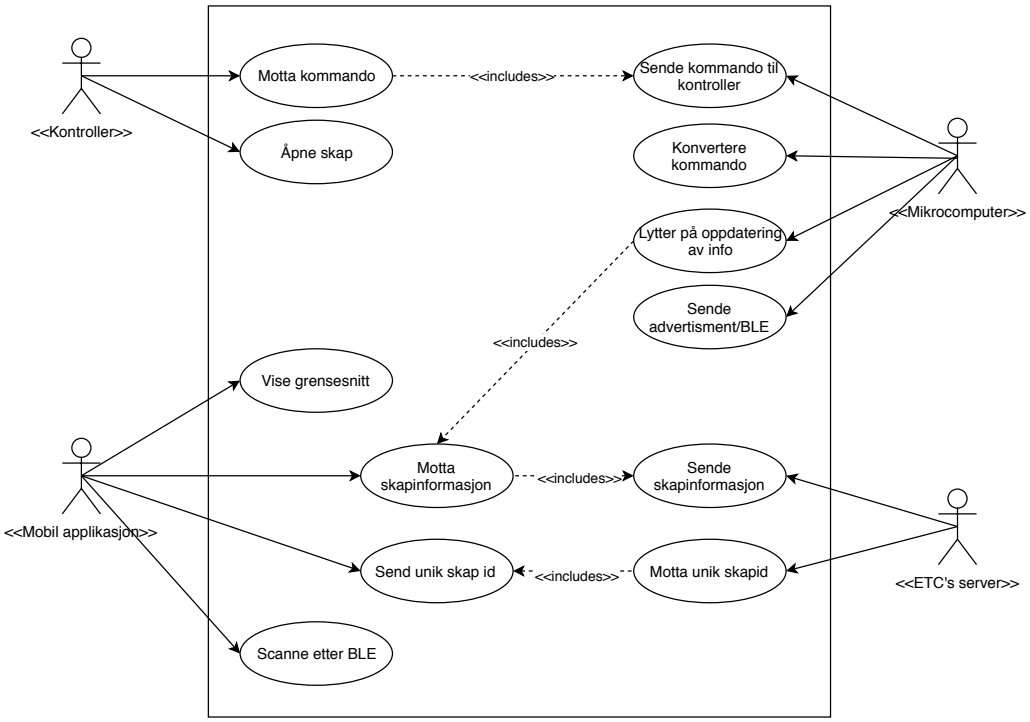
2.1 Funksjonelle krav

En del krav måtte være på plass for at systemet skulle fungere som forventet. Mange av kravene har dukket opp underveis, etterhvert som oppdragsgiver har gitt mer informasjon om hva som er ønsket. Det har også dukket opp uforutsette utfordringer, som resulterte i at enkelte krav måtte endres på underveis i utviklingen.

Applikasjonen skal kunne gjennomføre en filtrert skann av Bluetooth-enheter, for så å koble seg opp mot disse. Dette skal gjøre det mulig å kommunisere frem og tilbake over Bluetooth mellom mobiltelefon og mikrocomputer. Oppdragsgivers server vil benytte informasjonen som er hentet ut fra mikrocomputeren til å finne riktig skap. Deretter sendes denne informasjonen tilbake til mobilapplikasjonen, som igjen sender den til mikrocomputeren.

Mikrocomputeren skal lytte på oppdateringer av informasjon sendt fra mobilapplikasjonen, som må konverteres til en kommando og deretter sendes til programmet som håndterer åpning av skap.

2.2 Use case-diagram



Figur 2: Use case-diagram som viser funksjonaliteten i systemet.

2.2.1 Aktører

«Mikrocomputer»
En mikrocomputer av typen Raspberry Pi 3B som er hovedenheten i systemet og har som rolle å styre mikrokontrolleren i skapet. Denne skal tilby et Bluetooth grensesnitt for identifisering av skap for smarttelefoner og vil sende og lytte på kommandoer sendt fra mobiltelefonen over Bluetooth.

«Kontroller»
Oppdragsiveren krever at det skal brukes en egenutviklet kontroller som styrer låsemekanismen i skapet. Denne styres av kommandoer som vil bli sendt fra mikrocomputeren.

«ETCs server»
Oppdragsgiver har utviklet egen logikk som ligger på deres server. Denne logikken tar seg av bestillingene av kjøretøy. Prosjektet skal benytte seg av denne logikken for å bestemme hvilken kommando som skal kjøres og deretter hvilket skap som skal åpnes.

«Mobilapplikasjon»
Oppdragsgiver ønsker at det lages funksjonalitet for både iOS og Android som senere kan implementeres i deres egen mobilapplikasjon, som ennå er i utviklingsfasen. Denne funksjonaliteten vil benytte seg av Bluetooth-kommunikasjon for å identifisere deres skap.

2.3 Overordnede use case-beskrivelser

Use case	Vise grensesnitt
Aktører	«Mobilapplikasjon», «ETCs server»
Hensikt	Vise bruker et enkelt grensesnitt.
Beskrivelse	Før brukere skal bestille kjøretøy skal det vises et grensesnitt der brukeren kan velge å skanne etter et skap.

Use case	Skanne etter Bluetooth-enheter
Aktører	«Mobilapplikasjon»
Hensikt	Skaffe oversikt over hvor en bruker befinner seg.
Beskrivelse	Applikasjonen skal gjøre det mulig å finne ut hvor en bruker befinner seg. Ved bruk av Bluetooth gjøres dette ved å scanne etter en advertisement-pakke med en bestemt ID. Denne ID-en sendes inn som en del av identifiseringen av lokasjon og tilgang i applikasjonen.

Use case	Send unik skap-ID
Aktører	«Mobilapplikasjon»
Hensikt	Mobilapplikasjon finner ut hvilket skap den er i nærheten av.
Beskrivelse	Mobilapplikasjonen må kunne sende en unik 128-bits ID for skapet den er i nærheten av til CarAdmin-serveren.

Use case	Motta unik skap-ID
Aktører	«CarAdmin-server»
Hensikt	Motta den unike ID-en fra mobilapplikasjonen.
Beskrivelse	CarAdmin-serveren skal ta i mot den unike 128-bits ID-en sendt fra mobiltelefonen til en bruker, for så og håndtere denne informasjonen.

Use case	Sende skapinformasjon
Aktører	«CarAdmin-server»
Hensikt	Sende verdier til mobiltelefon.
Beskrivelse	CarAdmins server behandler brukerens bestilling og finner dørnummer på skapet brukeren er i nærheten av. Deretter sendes informasjon om hvilket skap og dør som skal åpnes tilbake til telefonen.

Use case	Motta skapinformasjon
Aktører	«Mobilapplikasjon»
Hensikt	Motta skapinformasjon for åpning av skap.
Beskrivelse	Mobilapplikasjonen mottar informasjon om riktig skap fra CarAdmin-serveren, som skal sendes videre til mikrocomputeren over Bluetooth.

Use case	Lytte på oppdatering av informasjon
Aktører	«Mikrocomputer»
Hensikt	Mikrocomputer venter på nøkkelskapverdier.
Beskrivelse	Mikrocomputeren skal kunne lytte på oppdateringer fra serveren, for å finne ut av hvilket skap brukeren befinner seg ved, samt hvilken skapdør som skal åpnes.

Use case	Sende advertisement/BLE
Aktører	«Mikrocomputer»
Hensikt	Sende ut en spesifikk ID for lokasjon
Beskrivelse	Mikrokontroller skal sende ut en advertisement-pakke med ID for bedriften og en spesifikk ID for den bestemte mikrocomputeren. Denne brukes for å finne riktig skap.

Use case	Sende kommando til kontroller
Aktører	«Mikrokontroller»
Hensikt	Sende informasjon om skap som skal åpnes
Beskrivelse	Mikrocomputeren skal sende informasjon til kontrolleren om hvilket skap som skal åpnes. Kommandoen videresendes til et eget program som utfører den.

Use case	Motta kommando
Aktører	«MikroKontroller»
Hensikt	Utføre kommando
Beskrivelse	Mikrokontrolleren utfører kommando som er tilsendt.

Use case	Åpne skap
Aktører	«Mikrokontroller»
Hensikt	Åpne skap for en bruker
Beskrivelse	Kontrolleren åpner en luke i et skap basert på tilsendt informasjon.

2.4 Supplementære krav

Prosjektet skulle forholde seg til det allerede eksisterende systemet som oppdragsgiver hadde ved prosjektstart. Dette ville si at det skulle være mulig å kommunisere med server og benytte applikasjonen som allerede er under utvikling. Ettersom oppdragsgivers applikasjon var skrevet i Xamarin, burde også konseptapplikasjonen utvikles i Xamarin.

Applikasjonen skulle fungere på både Android og iOS. Derfor kunne ikke koden benytte seg av kode som resulterte i at noe vil fungerte med det ene systemet, men ikke det andre. Det var derfor stilt krav til at biblioteker som ble tatt i bruk skulle fungere for begge enhetene, for ikke å gjøre jobben i prosjektet og oppdragsgiver mer krevende enn den trengte å være. Koden skulle også være samlet på ett sted, noe som ville være enklere å forholde seg til.

Det var satt krav til at både server og mobiltelefon skulle ha tilgang til internett. Dette var fordi oppdragsgivers applikasjon var laget som en webapplikasjon, som benyttet seg av serveren deres for oppkobling.

Koden som ble skrevet utenfor applikasjonen skulle skrives i Java, med mindre dette ikke lot seg gjennomføre på en god måte.

Nøkkelskapene skulle ikke kunne åpnes av en bruker som ikke var i nærheten av det. Dette ville si at applikasjonen skulle kunne filtrere bort skap som ikke er innenfor en angitt rekkevidde.

Løsningen skulle benytte seg av Bluetooth-teknologi for kommunikasjon mellom mobilapplikasjonen og mikrocomputeren. For at mobilapplikasjonen skulle kunne hente ut/-sende informasjon fra/til mikrocomputeren måtte enhetene kobles sammen med Bluetooth. Denne prosessen vil skje etter endt skanning, og henter ut informasjon om de ulike tjenestene som er satt opp.

Det burde benyttes plug-and-play-hardware. Dette ville si at komponentene enkelt skulle la seg byttes ut uten ekstra tiltak som for eksempel lodding. Derfor måtte det gjøres vurderinger angående hvilke mikrocomputer som skulle benyttes i løsningen.

Det ferdige produktet skulle enkelt kunne monteres ute hos kundene, ved at formfaktoren er liten, programvaren er godt forklart og det kan settes opp med riktig programvare på en enkel måte.

2.4.1 Lisenser

Ettersom oppdragsgiver tenker å benytte dette systemet til kommersielt bruk, var det viktig at det ble satt krav til hvilke kodebiblioteker som kunne benyttes. For at prosjektet skulle kunne bruke ekstern kode uten å måtte gi fra seg rettigheten til deler eller hele kildekoden, ville det benyttes kodebiblioteker som tilfredsstilte dette.

3 Analyse

Oppgaveteksten forklarte lite om *hvordan* oppgaven skulle løses. Fordelen med dette var at det gav stor frihet når det kom til teknologivalg og løsningsmetodikk. Ulempen var at, for å få et så optimalt system som mulig, så krevdes det mye research ift. hvilke teknologier som var tilgjengelige, hvilke som passet for dette prosjektet, hvilke som var enklest å bruke ift. andre delsystemer osv. Research-fasen i prosjektet var altså omfattende.

3.1 Valg av teknologier

Prosjektet hadde gitte krav til hva som skulle gjennomføres. For å realisere disse ble flere typer teknologier vurdert:

- Mikrocomputer-teknologi
- Operativsystem som skal benyttes på mikrocomputeren
- Kommunikasjonsteknologi mellom applikasjon og mikrocomputer
- Programmeringsspråk som skal benyttes på og mellom modulene

3.1.1 Valg av mikrocomputer

I forhold til valg av mikrocomputer, ble det vurdert flere typer fra forskjellige produsenter. Oppdragsgiver hadde allerede ved første møte forslag til en mikrocomputer, *Raspberry Pi 3B*, men ønsket likevel alternativer skulle undersøkes. Mikrocomputeren skulle muliggjøre det som er blitt satt som krav i kapittel 2. Alternativer som ikke oppfylte disse kravene ble ikke vurdert. Mikrocomputerne som oppfylte disse kriteriene er som følgende:

- Raspberry Pi 3B
- Raspberry Pi Zero WH
- NanoPi-2
- Intel® Galileo Gen. 2 Board
- Minnowboard 3

Under prosjektet ble overnevnte mikrocomputerne vurdert, hvor det ble lagt vekt på pris, eksisterende dokumentasjon, integrerte komponenter, funksjonaliteter i kortet, og brukervennlighet. Siden ingen av gruppemedlemmene hadde noen relevant erfaring med mikrocomputere, var vi avhengig av god dokumentasjon. Derfor falt valget på en av Raspberry Pi-enhetene som har vært på markedet en god stund, og har dokumentasjon på det meste. Når det kom til Raspberry Pi-enhetene, så er Zero WH billigere enn 3B,

Figur 3: Raspberry Pi 3B

men manglet enkelte funksjonaliteter prosjektet var avhengig av. Zero WH har også færre tilkoblinger, noe som gjør det vanskeligere å koble til og ville krevd bruk av unødvendige overganger. Derfor valgte vi å benytte Raspberry Pi 3B, som vist i figur 3.

Arduino ble også vurdert som et alternativ, men da dette er ikke en fullverdig mikrocomputer, passer den ikke til dette prosjektet. Denne ble derfor tidlig faset ut.

3.1.2 Valg av operativsystem på mikrocomputer

Det ble valgt å gå for Raspberry Pi sitt offisielle operativsystem *Raspbian Lite*. Dette er et operativsystem som er basert på Linux Debian som også er godt dokumentert, noe som var en viktig faktor, i og med at ingen hadde noen relevant, tidligere erfaring med Raspberry Pi. Lite-versjonen av Raspbian er en lettvektsversjon uten det grafiske skrivebordet, men som istedet baserer seg på input og output via kommandolinjen.

3.1.3 Valg av kommunikasjonsteknologi

En bruker av CarAdmin-applikasjonen skal kunne etablere en tilkobling mellom sin smarttelefon og et nøkkelskap som er utplassert der hvor brukeren er. Det finnes flere muligheter for å oppnå en slik tilkobling. Bluetooth-teknologi blir flere ganger nevnt i oppdragsgivers oppgavebeskrivelse, og dette var derfor teknologien som det ble umiddelbart tatt utgangspunkt i.

	Proksimitet	Pålitelighet	Sikkerhet	Brukervennlighet	Tilgang	Tot.
BT	4	3	3	4	5	19
NFC	5	4	4	4	3	20
QR	2	4	2	3	5	16

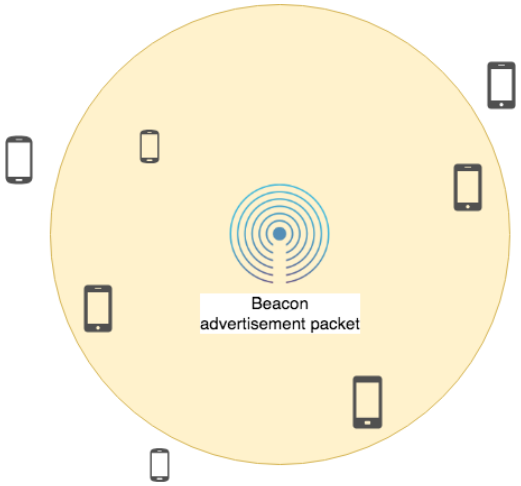
Tabell 1: Viser hvordan de forskjellige kommunikasjonsteknologiene måler opp mot hverandre, basert på en poengsum fra 1 (lavest) til 5 (høyest).

For å være sikker på at produktet som leveres fungerte på en så optimal måte som mulig, spesielt ift. daglig bruk, var det likevel ønsket å se på andre kommunikasjonsteknologier. Disse ble satt opp mot Bluetooth, for å se på eventuelle fordeler og ulemper mellom de forskjellige teknologiene, basert på «jobben» den skal gjøre (kravene som stilles) for produktet. De mest relevante kravene ift. tilkoblingsteknologi var følgende:

- Teknologien skal kunne forsikre om at en bruker befinner seg i nærheten av det aktuelle nøkkelskapet.
- Teknologien skal være brukervennlig, hvilket i denne sammenhengen betyr at det skal være enkelt å koble seg til et nøkkelskap, uten nevneverdige forsinkelser, feilmeldinger el.l.
- Teknologien skal finnes på de aller fleste av dagens smarttelefoner
- Teknologien skal være sikker. Det skal ikke være mulig for uvedkommende å få tak i dataene som overføres.

I tillegg til Bluetooth ble det også gjennomgått både NFC (Near-Field Communication) og QR (Quick Response) som mulige teknologier. Disse har varierende styrker og svakheter ift. de nevnte kravene. Tabell 1 viser i grove trekk hvordan de forskjellige teknologiene gjør det på hvert område. Skalaen går fra 1-5, hvor 1 er laveste poengsum og 5 er høyeste poengsum. Merk at disse tallene er egne estimater, basert på kunnskapen gruppen hadde ervervet seg om hver av dem.

Basert på tallene i tabell 1, så var det faktisk NFC teknologien som kom ut med høyest poengsum. NFC regnes som en pålitelig, rask og svært enkel teknologi, som i utgangspunktet ville latt en bruker av CarAdmin-applikasjonen legge mobiltelefonen sin på et nøkkelskap, og så går resten «av seg selv». Det er neppe tilfeldig at både Apple og Android benytter seg av NFC-teknologi når det gjelder mobilbetaling (som forøvrig har mange av de samme kravene som prosjektet har). Bakdelen med NFC per i dag er at den er mindre tilgjengelig enn både Bluetooth og QR. De fleste nyere smarttelefoner kommer med NFC-støtte, men det finnes fortsatt en god del smarttelefoner uten støtte. I samråd med oppdragsgiver ble totalvurderingen derfor slik at Bluetooth ble valgt som kommunikasjonsteknologi mellom smarttelefon og nøkkelskap. Det var allikevel et mål at systemet skulle utvikles som moduler med lave koblinger, og at en enkeltmodul enkelt kan byttes ut med en annen modul. Det burde derfor være mulig å erstatte Bluetooth som kommunikasjonsteknologi med NFC, dersom det viser seg at NFC ville fungert på en bedre måte.



Figur 4: Beacon-enheten i sentrum annonserer en datapakke, og alle Bluetooth-enheter innen en viss radius kan fange opp denne.

Beacon

En «bonusfunksjon» ved bruk av Bluetooth er såkalte *Beacons*. En beacon er en liten hardware-enhet som sender ut et BLE-signal (BLE = Bluetooth Low Energy). Signalet kan plukkes opp av alle enheter med Bluetooth v4.0 eller nyere. beacon-teknologien gir flere spennende muligheter, også med tanke på prosjektet. For eksempel kan en beacon automatisk varsle en bruker om at han er i nærheten av et nøkkelskap. En annen fordel er også at dataoverføringer kan skje uten Internett-tilkobling, noe som potensielt kan tillate en bruker å hente ut en bilnøkkel selv om nettilkoblingen feiler (dette var forøvrig et ønske fra oppdragsgiver). Det eneste som kreves er at brukeren befinner seg i nærheten av en beacon. Figur 4 viser hvordan beacon-teknologien fungerer i praksis.

Som applikasjonsutvikler for iOS og Android finnes to relevante protokoller ift. beacon-teknologien. iOS og Apple har sin *iBeacon*, mens Android og Google har sin *Eddystone*. I dette prosjektet falt valget på Google og Eddystone, av den enkle grunn at den tilfredsstiller løsningskravene på en bedre måte enn hva iBeacon gjør. Eddystone er multiplattform, og kan uten tilpasning benyttes på alle enheter med støtte for BLE-beacons, noe iBeacon ikke kan «out-of-the-box». Eddystone er open-source, med kildekoden tilgjengelig på GitHub[1]. Den viktigste grunnen til at Eddystone ble valgt fremfor iBeacon er likevel pakkeformatet. iBeacon sender ut et BLE-signal som er synlig for alle kompatible enheter. Eddystone, på den annen side, gir flere valgmuligheter ift. pakkeformat. På grunn av tidsrammen i dette prosjektet, ble likevel den «enkleste» varianten benyttet, som i praksis fungerer på samme måte som iBeacon. I delkapittel 7.3 beskrives noe om hvordan en sikrere og mer avansert versjon av Eddystone fungerer.

3.2 Biblioteker

For å gjøre ting litt lettere for i prosjektet og for oppdragsgiver, ble det benyttet visse biblioteker. Bibliotekene ble brukt for å legge til nyttig funksjonalitet uten å måtte lage alt selv der det ikke holdt med de allerede eksisterende mulighetene i kodespråkene. Da var det enklere å bruke biblioteker som allerede hadde mye av den ønskede funksjonaliteten.

3.2.1 Kontroller

Koden for kontrolleren, *LockerControllerUsb_java.java*, benytter seg av et kodebibliotek for kommunikasjon med USB-enheter, kjent som *usb4java*. usb4java baserer seg sterkt på kodebiblioteket libusb og har også implementert høynivåimplementasjonen javax-usb (JSR-80). Til tross for at den manglet grundig dokumentasjon, var det dette biblioteket som var enklest å bruke og utnytte for å kommunisere med skapkontrolleren, en HID-enhet[1], fra ETC.

3.2.2 Mobilapplikasjon

Mobilapplikasjonen som skulle brukes måtte ha mulighet for å skanne etter advertisement-pakker fra en BLE-beacon. Optimalt sett var det ønsket å finne et bibliotek som var enkelt å bruke, anvendelig og som fungerte på både iOS og Android med samme kode. Det ble derfor vurdert flere biblioteker som hadde mulighet for dette. Det ble først vurdert biblioteker som tillot å arbeide direkte på kommunikasjon med beacon, men det fantes ingen biblioteker som ville gjøre denne jobben både på iOS og Android. Siden mobilapplikasjonen også skulle kunne koble seg opp mot Bluetooth-enheter, måtte det uansett vurderes et bibliotek som hadde disse mulighetene. I tabell 2 vises de bibliotekene som ble tatt i betraktning og vurderingene som har blitt gjort for hver av dem.

Det ble til slutt valgt å benytte Xamarin Bluetooth LE i prosjektet, ettersom dette passet best for formålet for prosjektet, og fungerte for både iOS og Android. Dette biblioteket benytter seg av Apache License 2.0 som blir beskrevet senere i delkapittel 3.3.3. Denne lisensen gir mulighet for at biblioteket kan brukes til kommersielt bruk.

[1] HID (Human Interface Device): enheter som kan gjenkjennes uten spesifikke drivere i de fleste operativsystemer.

Navn	Enheter støttet	Det positive	Det negative	konklusjon
AltBeacon[2]	Android	Støtter kommunikasjon med alle typer beacon, inkludert Eddystone. Tilpasset for Java og C#	Fungerer bare for Android.	Vil ikke kunne brukes i prosjekt sammenheng, grunnet kun støtte av Android.
Universal Bluetooth Beacon[3]	Android, Windows	Enkel kommunikasjon mot alle beacon typer.	Fungerer ikke mot iOS for øyeblikket, iOS funksjonalitet kommer senere.	Ettersom det ikke er lagt til funksjonalitet for iOS vil ikke denne kunne brukes.
Xamarin Bluetooth LE[4]	Android, iOS, Windows	Støtter de fleste enheter. Enkel kommunikasjon mot Bluetooth. Har mulighet for å se på advertisement pakker.	Må gjøre manuell skann og filtrere ut enheter for å finne beacons.	Fungerer bra i sammenheng med prosjektet med tanke på hva biblioteket skal benyttes til.
Native Android	Android	Full støtte for Android	Må ha separat kode for iOS og Android. Vanskelig å benytte i prosjektet.	Blir vanskelig å bruke i prosjektet, det vil ta lenger tid å få funksjonaliteten som er nødvendig.
Native iOS	iOS	Full støtte for iOS	Må ha separat kode for iOS og Android. Vanskelig å benytte i prosjektet.	Blir vanskelig å bruke i prosjektet, det vil ta lenger tid å få funksjonaliteten som er nødvendig.

Tabell 2: Vurdering av Bluetooth bibliotek

3.3 Lisenser

Når det ble benyttet andres arbeid i prosjektet, måtte det tas hensyn til lisensene som var knyttet opp mot dette. Disse lisensene kan være veldig forskjellige fra hverandre og ha ulike krav for hvordan man kan benytte dem. Lisensene som er brukt i dette prosjektet er rettet mot biblioteker for bruk i programmering. Disse bibliotekene ble benyttet for å gjøre det enklere for oss og for sluttbruker å sette seg inn i koden uten å måtte lage alt selv fra grunnen av.

3.3.1 GNU GPL - General Public License

GNU GPL [5] er en gratis, copyleft lisens for programvare og andre verk. Denne lisensen krever at programvare som bruker kode utgitt med GNU GPL lisensen ikke kan benyttes til kommersielt bruk og at programvaren alltid må være åpen og gratis. Om kodesnutter eller kodebibliotek med GNU lisens brukes i prosjektet, er det påkrevd at prosjektet må havne under samme GNU lisens. Dette vil desverre skape problemer for prosjektet, hvor enkelte deler av denne lisensen strider i mot det ønskede produktet i prosjektet.

Det har dermed blitt valgt å være varsom når det ble brukt kode som var dekket av GNU GPL lisensen for å ikke overkomplisere utviklingen og for å slippe å måtte gi ut de viktigere delene av koden som «open-source». BlueZ er blant bibliotekene som er dekket av denne lisensen i prosjektet.

3.3.2 GNU LGPL - GNU Lesser General Public License

GNU LGPL [6] er en enklere utgave av GNU GPL lisensen, hvor den blant annet tillater kommersiell bruk under visse forhold. Den er mer rettet for å beskytte modifikasjoner, endringer og forbedringer på selve koden som bruker denne lisensen. Prosjekter som ønsker å bruke slik kode eller hjelpebibliotek med GNU LGPL lisens trenger ikke å bruke samme lisens, med mindre man modifiserer verk som er dekket av denne lisensen og selv da trenger bare delene som bruker den modifiserte utgaven av koden bruke denne lisensen og være åpen. Andre krav som stilles er at man ikke har lov til å hindre bruk av en nyere utgave av den lisensierte koden og at det skal være mulig å reversere koden for vedlikehold og testing.

3.3.3 Apache Version 2.0 License

Apache lisensen er en ́Permissive ́lisens som vil si at man ikke trenger å gi fra seg egen kode ved bruk, i motsetning til copyleft lisensene som er mer restriktive på hva som er lov til å gjøre med programvaren. Ettersom programvaren som ble lagd skal benyttes til kommersielt bruk, passet denne lisensen bra for dette formålet. Apache lisensen [7] ble i prosjektet brukt i sammenheng med biblioteket Xamarin Bluetooth LE, og for å kunne benytte Eddystone beacon teknologien.

3.3.4 MIT License

MIT lisensen[8] er en gratis programvare lisens brukt på open source programmer. Programmer som bruker biblioteker under denne lisensen kan distribueres, publiseres og selges kommersielt uten restriksjoner. Det er også mulig å modifisere koden uten å måtte legge ut koden for andre å se.

I prosjektet brukes blant annet kodebiblioteket «usb4java[9]» for USB kommunikasjon i Java som ligger under denne lisensen.

4 Design og Arkitektur

I dette kapittelet utledes arkitekturen og oppbyggingen av systemet. Først forklares det store bildet og de delsystemene som dette prosjektet skal implementeres i. Så utledes det i større detalj, og sees på hvordan delsystemene kommuniserer med hverandre. Til prosjektet ble boken Software Engineering[10] benyttet og kunnskapene gruppen har oppbygget seg i *IMT2243 - Systemutvikling*, som bakgrunn for dette.

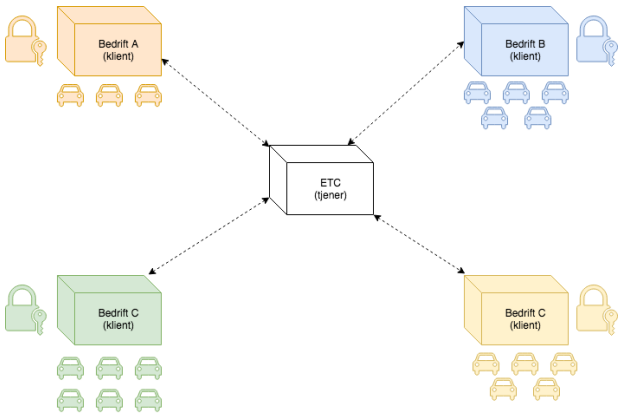
4.1 Eksisterende system

Denne oppgaven handler i utgangspunktet om å modernisere oppdragsgiverens eksisterende løsning. Arkitekturen og samhandlingen mellom hovedkomponentene i systemet er derfor i grove trekk allerede fastsatt. For å forstå valg og beslutninger tatt i prosjektet, så er det viktig å ha et overordnet bilde av komponentene i systemet og sammenhengene mellom dem.

I svært grove trekk, så kan CarAdmin-systemet sees på som vist i figur 5: en klient/tjener-arkitektur hvor ETCs server, med brukerdatabase og logikk, fungerer som en tjener for de forskjellige bedriftene (klientene) som benytter seg av CarAdmin. Bil-ikonene under hver bedrift representerer bilparken deres, mens nøkkelen og låsen representerer nøkkelskapet deres. Bilparkene og nøkkelskapene er samlokalisert, mens uthentingsalgoritmen og logikken ligger hos ETC. Hadde man lagt til et datamaskin-ikon hos hver av bedriftene, så hadde figuren vist systemet slik det fungerer i dag. Datamaskin-ikonet hadde da representert den fysiske datamaskina som i dag står utplassert ved hvert nøkkelskap, hvor brukerne av CarAdmin må logge seg på hver gang de ønsker å bruke en bil.

Med prosjektet var det ønsket å «erstatte» datamaskin-ikonet, og lage en mer moderne løsning hvor brukerne heller benytter seg av smarttelefonen sin for å gjøre interaksjon med nøkkelskapet.

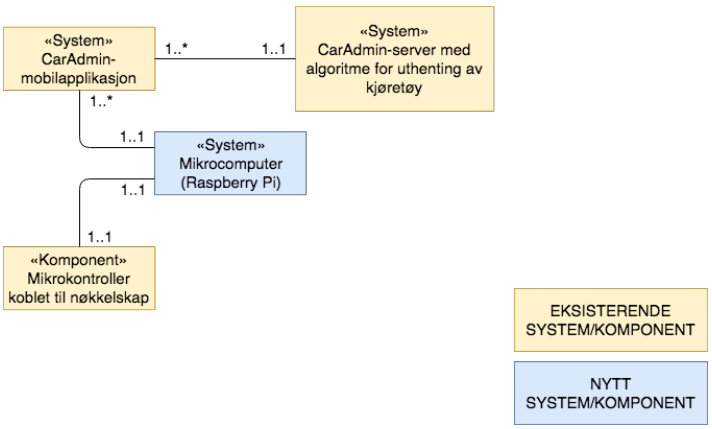
Det har i all hovedsak blitt sett på oppdragsgivers komponenter som «svarte bokser». I praksis betyr dette at det i liten grad har vært nødvendig å forholde seg til hva som faktisk foregår *inne i* de enkelte komponentene, men heller hva slags input de tar og hva slags output de produserer.



Figur 5: Systemet i grove trekk: ETC fungerer som tjener, mens brukerne fungerer som klienter.

Komponentene som prosjektet har forholdt seg til er som følgende:

- **ETCs server**: inneholder brukerdatabasen til CarAdmin, samt logikk som finner «rett» bil til brukeren. Gir en bruker svar på hvilke biler som er tilgjengelige for brukeren, og ved slutten av prosessen hvilket skapnummer og hyllenummer en spesfikk nøkkel ligger i.
- **ETCs egen mikrokontroller**: håndterer åpning av nøkkelskapene som ETC benytter seg av.
- **CarAdmin-mobilapplikasjon**: webapplikasjon skrevet i Xamarin (C#). Med «webapplikasjon» menes det at CarAdmin benytter sin nåværende webapplikasjon (som kjøres i en standard nettleser), og viser denne i deres egen native applikasjon. I skrivende stund er dette en uferdig applikasjon under utvikling, og fungerer mer som et «skjelett» som logikken skal implementeres i.



Figur 6: Kontekstdiagram som viser grensene og kardinaliteten i systemet.

Kontekstdiagrammet i figur 6 viser grensene og kardinaliteten. De gule komponentene eksisterer allerede, men må likevel tilpasses det nye systemet. Den eneste komponenten som legges til i systemet er den blå mikrocomputeren.

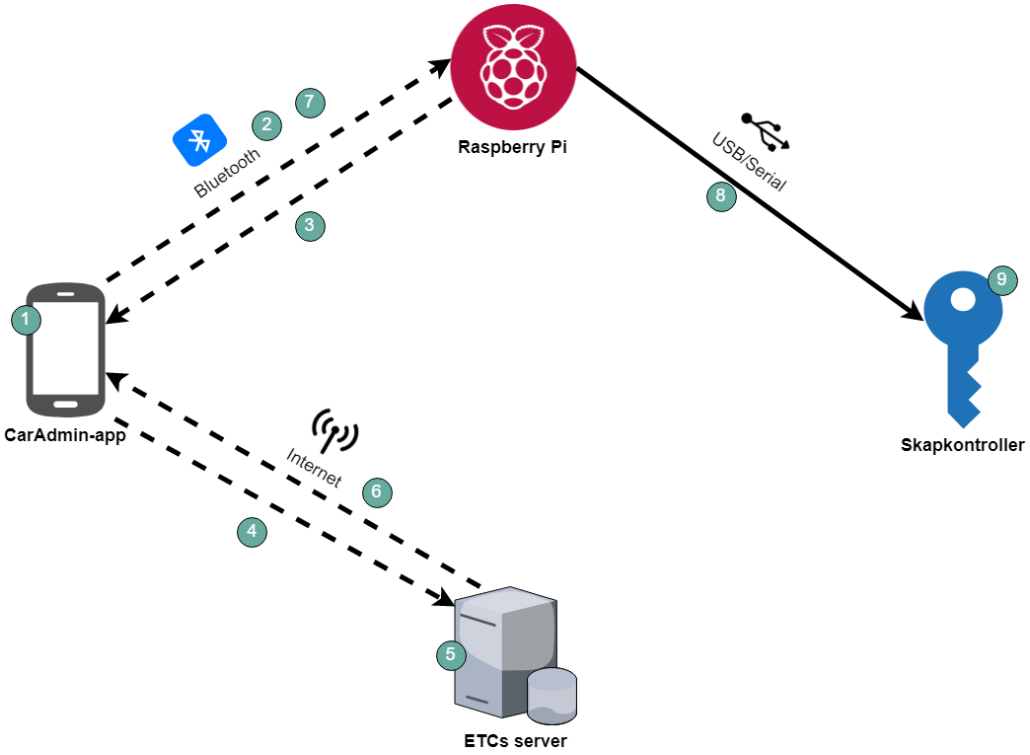
4.2 Modernisering av systemet

Oppdragsgiver ønsket med dette prosjektet en modernisering av det daværende systemet, hvor brukerne av CarAdmin på en enkel måte kan hente ut et kjøretøy ved å bruke sin egen smarttelefon. Dette innebærer at den nevnte datamaskinen som i dag står utplassert ved nøkkelskapene skal erstattes. Erstatningen skjer i form av en mikrocomputer (Raspberry Pi 3), som plasseres inne i nøkkelskapene, usynlig for brukeren. Mikrocomputeren annonserer konstant et Bluetooth Low Energy-signal som forteller mobilapplikasjonen hvilket nøkkelskap den er i nærheten av.

Det er et relativt komplekst system med mye datatrafikk frem og tilbake mellom komponentene. For brukeren er dette i stor grad usynlig, men det er likevel viktig at brukeropplevelsen er god. I dette prosjektets sammenheng betyr det hovedsakelig at dataflyten er optimal, og at det ikke oppstår tilkoblingsproblemer, tilfeldige frakoblinger, forsinkelser osv. Dette vil oppleves frustrerende for en sluttbruker som bare ønsker å hente ut en bilnøkkel, og en del av valgene som er gjort ift. arkitektur og design er derfor tatt med hensyn til dette.

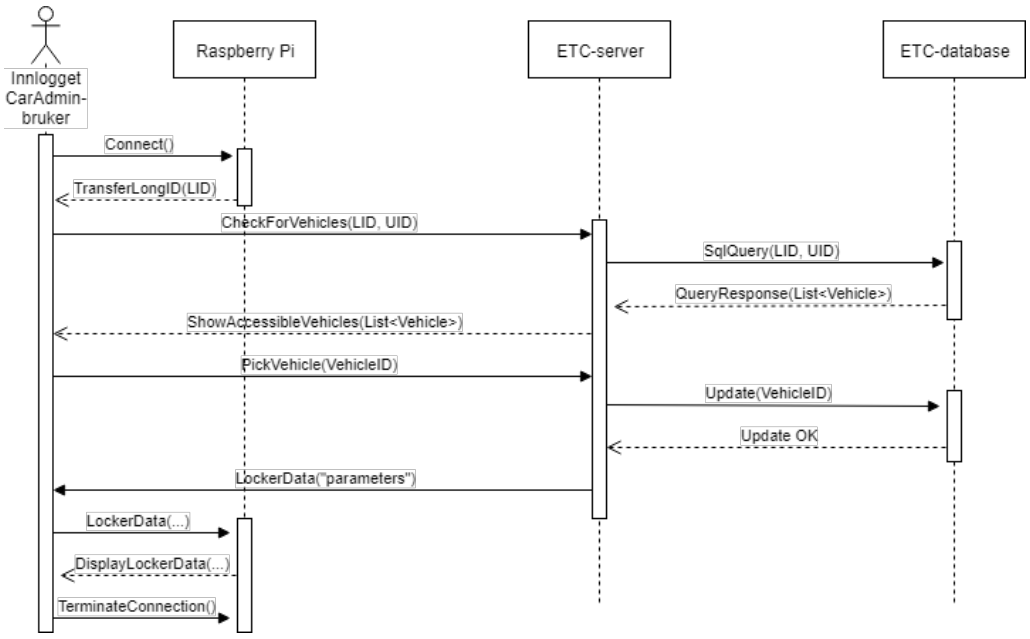
Figur 7 viser med nummerne hvilken rekkefølge uttaksprosessen følger. Flyten i systemet er slik:

- En bruker åpner CarAdmin applikasjonen og trykker på «Ta ut bil»-knappen
- CarAdmin applikasjonen oppdager via BLE at den er i nærheten av en Raspberry Pi, og forsøker å koble seg på dette.
- Dersom mobil applikasjonen har koblet seg på Raspberry Pi via BLE, så overføres automatisk nøkkelskapets ID til CarAdmin applikasjonen.
- CarAdmin applikasjonen sender automatisk nøkkelskapets ID til ETCs server via HTTP.
- ETCs server finner ut hvilke kjøretøy som er tilgjengelig for den enkelte bruker ved det spesifikke skapet.
- ETCs server svarer med valgene som brukeren kan ta.
- Bruker trykker på ønsket kjøretøygruppe (og evt. andre valg) i CarAdmin applikasjonen, og sender dette tilbake til ETC-serveren, som oppdaterer kjøretøydatabasen med valget som er gjort.
- ETCs server kjører algoritmen som bestemmer hvilket kjøretøy brukeren skal få, og sender skapnummer og hyllenummer tilbake til mobilapplikasjonen.
- CarAdmin applikasjonen sender de mottatte verdiene til Raspberry Pien som den fortsatt er koblet til via BLE.
- Raspberry Pi tar i mot og behandler skapnummer og hyllenummer, og sender disse videre til en biblioteksfunksjon som tilhører mikrokontrolleren (som fysisk er koblet til Raspberry Pi via USB/Serial).
- Biblioteksfunksjonen bruker skapnummer og hyllenummer for å åpne korrekt skap
- CarAdmin applikasjonen viser så hvilket skap og hvilken hylle som er åpen, og brukeren kan hente ut nøkkelen. BLE-koblingen termineres.



Figur 7: Oversikt over det moderniserte systemet og dets flyt.

Sekvensdiagrammet i figur 8 viser i større detalj hvilke komponenter som kommuniserer med hverandre, og også hvilke komponenter som er aktive til et hvert tidspunkt i uthentingsprosessen.



Figur 8: Sekvensdiagram som viser prosessen for uthenting av et kjøretøy.

5 Implementasjon

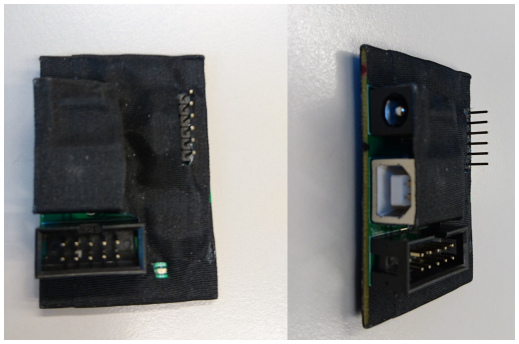
Dette kapittelet tar for seg gjennomførelse og implementeringen av selve løsningen. Det innebærer også hvilke vanskeligheter som oppstod underveis i prosjektet og hva som ble gjort for å oppnå det som skulle gjøres.

5.1 Raspberry Pi

For å komme i gang med Raspberry Pi, måtte den først kobles opp mot en ekstern skjerm for å kunne settes opp for oppkobling gjennom SSH. Skolenettverket på NTNU i Gjøvik gjorde det ikke mulig for å koble til Raspberry Pi via SSH, ettersom brannmuren på nettverket ville stoppe denne pakken fra å nå målet sitt. For å komme utenom dette, kan man på Raspberry Pi koble til via hostnavnet «raspberrypi.local» som gjør det mulig å koble rett til uten å være på samme nettverket. Dette krevde nedlasting av Apples print tjeneste kalt «Bonjour»[11] for å fungere.

5.1.1 Kontroller

Oppdragsgiveren hadde som et av deres grunnleggende krav at prosjektet skulle bruke deres egenutviklede kontroller, figur 9. Denne kontrolleren styrer låsemekanismen i skapene deres og styres den dag i dag av kode som kjører eksklusivt på Windows.



Figur 9: Skapkontrolleren

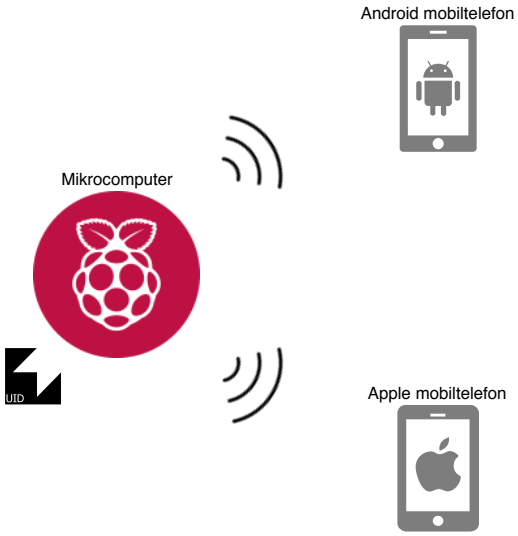
En av utfordringene med kontrollerbiblioteket var at det skulle kunne kjøres på et system som kjører en distribusjon av UNIX/Linux. Koden oppdragsgiver brukte for å styre kontrolleren viste seg å ikke fungere på dette systemet og ville dermed trenge å bli skrevet om fra grunnen av for å støtte systemet. Denne delen av prosjektet ble skrevet i Java,

grunnet ønske fra oppdragsgiver, samt gode eksempler og veiledning tilgjengelig på nett. En annen bonus ved å skrive det i Java var at systemet ville kunne kodes til å virke både på Linux og Windows. Underveis oppstod det en utfordring med å sende kommandoer til kontrolleren, da det var vanskelig å vite om kommandoene ble godkjent eller ikke før det faktisk ble sendt inn en gyldig kommando. Dette viste seg å kreve noe ekstra tid for å finne ut av.

En mer detaljert beskrivelse av noe av koden kan leses i 5.4.4.

5.1.2 Beacon-oppsett

Ettersom prosjektet var veldig bredt med tanke på hva som skulle gjøres og at forskjellige komponenter skulle kommunisere for å få jobben gjort, ble det i første omgang prioritert å få dem til å fungere. Det ble derfor valgt å benytte Eddystone sin UID beacon istedet for den mer sikre EID. Grunnen til dette var at denne er enklere å sette opp og krevde ingen ekstra funksjonaliteter i applikasjonen utenom skanning av pakken. EID på sin side måtte ha benyttet Googles API for å holde på de hemmelige nøkklene for både applikasjonen og Raspberry Pi enheten. For å sette opp selve advertisement pakken, som sendes ut på Bluetooth, ble det nødvendig å sette seg inn i dokumentasjon for Eddystone og hvordan dette ville fungere i praksis. Denne informasjonen er lett tilgjengelig på internett ettersom Eddystone ligger ute med åpen kildekode, og prosjektsiden til Eddystone på GitHub[1] kunne tilby god informasjon til hvordan den kunne benyttes. Selv om denne siden hadde god dokumentasjon på selve Eddystone, stod det lite om hvordan dette faktisk skulle settes opp på en Bluetooth enhet. Det ble derfor brukt et annet eksempel[12] som benytter seg av en Eddystone-URL som blir satt opp på samme måte, men det som faktisk sendes ut er forskjellig. Figur 10 viser visuelt hvordan beacon fungerer i løsningen, der mikrocomputeren sender ut en melding som kan oppfattes av både Apple og Android enheter via applikasjonen.



Figur 10: Visning for hvordan mikrocomputeren skal fungere som en beacon

Det ble senere oppdaget at denne måten å gjennomføre prosjektet på ikke lot seg gjøre. Dette var fordi den unike skap ID-en som oppdragsgiver bruker for å identifisere skapene var lenger enn hva som kunne utnyttes av den unike 128 bits ID-en som er tilbudt i Eddystone. Denne ville måtte deles opp for å kunne brukes for å finne både et skap tilhørende oppdragsgiver og for å finne selve skapet. Dette ville si at selv om både den unike skap ID-en og Eddystones lengde var like, ville disse ikke kunne brukes om hverandre. Det ble derfor valgt å gå for en mer tradisjonell Bluetooth tilkobling der en bruker først kobler seg på Bluetooth og deretter leser ID-en. Ved å gjøre det på denne måten var prosjektet ikke lenger bundet til lengdebegrensningen for ID-er i Eddystone og kunne heller benytte seg av en 128bit lengde der den kunne holde på mye mer informasjon. Selv om dette endret på måten den unike skap ID-en ble hentet ut, ble Eddystone fortsatt benyttet for å identifisere gyldige skap, da dette ville fortsatt tillate det å enklere søke opp spesifikke ID-mønstre for å filtrere ut riktig skap.

For å sette opp mikrocomputeren til broadcast av advertisement pakker i form av Eddystone, måtte det lages et bash script. Dette scriptet, som vist i koden 5.1 sjekker først om en Bluetooth enhet ikke er koblet til. Om dette ikke er tilfellet vil den deretter slå på Bluetooth, sette denne til å akseptere oppkoblinger fra enheter i form av en GATT tilkobling(se delkapittel 5.1.4), og deretter vil den starte broadcast av pakken.

Listing 5.1: Oppsett for Eddystone og start av broadcasting av pakken

```
1 #!/bin/bash
2
3 if [ ! -d /sys/devices/platform/soc/3f201000.serial/tty/
     ttyAMA0/hci0/hci0\:* ]
4 then
5         sudo hciconfig hci0 up
6         sudo hciconfig hci0 leadv 0
7         sudo hcitool -i hci0 cmd 0x08 0x0008 1e 02 01 06 03
             03 aa fe 15 16 aa fe 00 e7 00 01 02 03 04 05 06
             07 08 09 01 02 03 04 05 06
8 fi
```

I tabell 3 vises hvilke parametere advertisment pakken inneholder, og forklaring på hva de er.

Parameter	Beskrivelse
0x08	#OGF = Operation Group Field = Bluetooth Command Group = 0x08
0x008	#OCF = Operation Command Field = HCI_LE_Set_Advertising_Data = 0x0008
1e	Lengden i hexadecimal for antall bytes som følger
02	Lengde for det som følges
01	Flag data type verdi
06	Flag data
03	Lengde for det som følger
03	komplett liste av 16-bit Service UUIDs data type verdi
aa	16-bit Eddystone UUID
fe	16-bit Eddystone UUID
15	Lengde for det som følger
16	Service Data data type verdi
aa	16-bit Eddystone UUID
fe	16-bit Eddystone UUID
00	Eddystone-UID
e7	TX Power
00 - 09	Egensatt 10-byte verdi for namespace
01 - 06	Egensatt 6-byte verdi for identifikator

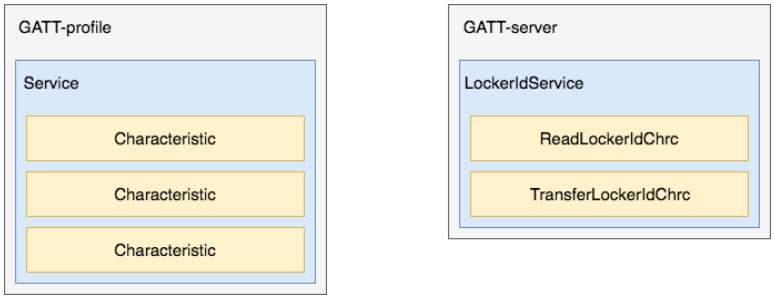
Tabell 3: Innhold i Eddystone advertisment pakken

Når noen kobler seg på Bluetooth enheten, stoppes advertisement av pakken. På grunn av dette måtte det en måte til for å gjøre slik at dette ble slått på igjen om denne skulle vært avskrudd. Dette viste seg å være vanskeligere enn først antatt. Det ble først vurdert å bruke udev[1] for å kunne kjøre et script når noen kobler seg opp over Bluetooth. Udev gjør dette mulig siden den monitorerer endringer i enheter som kobles til eller av. Siden dette tok mye tid å finne ut av uten å få det til å kjøre slik som ønsket, ble det heller vurdert å bruke crontab[2] til å gjøre jobben. Denne brukes for å lage en sjekk i scriptet på endringer i enhets mappen, hvor crontab sjekker en gang hvert minutt om dette skal kjøres.

5.1.3 BlueZ

BlueZ[13] er en Bluetooth-stack utviklet med hensyn på Linux, og ble i prosjektet brukt på Raspberry Pi-enheten for å gjøre Bluetooth-kommunikasjon mulig. BlueZ ble i første omgang brukt for å kringkaste BLE-pakken som mobilapplikasjonen kobler seg opp mot. Etter hvert benyttet prosjektet seg også av BlueZ for selve koblingen og datautvekslingen mellom mobilapplikasjonen og Raspberry Pi-enheten.

[1] udev er en «device manager» i Linux
[2] crontab er en tidsbasert «job scheduler» i Unix-baserte systemer



Figur 11: Bildet til venstre viser en generisk GATT-servers oppbygning, mens bildet til høyre viser prosjektets GATT-server.

5.1.4 GATT

GATT (Generic Attributes)[14] definerer hvordan Bluetooth Low Energy-enheter utveksler data *etter* at tilkobling har skjedd mellom to BLE-enheter. Enhetene kan da kommunisere via sine *services* og *characteristics*.

En **service** er en logisk gruppering av en eller flere characteristics. Hver service har en unik 128-bits UUID-identifikator (offisielle Bluetooth-services har 16-bits predefinerte UUID).

En **characteristic** er den minste enheten (altså det laveste nivået) i GATT-konseptet. Den inneholder også en 128-bits UUID, en verdi, et sett av egenskaper som sier noe om hvilke operasjoner som er støttet av den spesifikke characteristic, og et sett av tillatelser med tanke på sikkerhet. Characteristics inneholder ofte også *descriptors*.

Descriptors inneholder metadata eller konfigurasjonsflagg relatert til en characteristic. Descriptors ble ikke benyttet i dette prosjektet.

For å gjøre det mulig å lese og overføre data mellom CarAdmin-applikasjonen og Raspberry Pi trengtes førstnevnte å gjøres til en GATT-klient og sistnevnte til en GATT-server. Det ble tatt utgangspunkt i BlueZ sin eksempelkode for oppsett av GATT-servere[15], og vår GATT-server ble konfigurert med en service (`LockerIdService`), og to characteristics (`ReadLockerIdChrc` og `TransferLockerIdChrc`). Figur 11 viser hvordan den logiske oppbygningen av GATT-serveren ser ut. `ReadLockerIdChrc` inneholder nøkkelskapets unike skap-ID. GATT-klienten henter ut denne, og sender den videre til CarAdmin-serveren. Serveren finner ut hvilket nøkkelskap brukeren er i nærheten av, og sender et svar tilbake med dataene som trengs for å åpne riktig skapdør. Disse dataene skrives tilbake til GATT-serveren via `TransferLockerIdChrc`, som igjen sender disse videre til et annet program for behandling. Det fantes også allerede et eksempel[16] på oppsett for å kunne kjøre denne koden på en Raspberry Pi-enhet. Koden som ble brukt som utgangspunkt var skrevet i Python, noe gruppen hadde noe erfaring med fra bl.a. *IMT3881 - Vitenskapelig programmering*. Det tok derfor ikke like lang tid å sette seg inn i de fleste delene av denne koden. Koden benyttet seg av *D-Bus*-mekanismen (ved å bruke biblioteket *pydbus*), som tillater flere prosesser å kommunisere med hverandre.

5.2 Programmeringsspråk og IDE

Oppdragsgiveren ønsket at prosjektet i utgangspunkt skulle programmeres så mye som mulig i Java som programmeringsspråk. Siden dette var språket de benyttet seg mest av, ville dette gjøre det betydelig enklere for dem å sette seg inn i koden som skrives og videreutvikle den. I tilegg var det benyttet C# i Xamarin i sammenheng med mobilapplikasjonen deres de har lagd, og Python i forbindelse med GATT-serveren. Derfor ble dette også noe som måtte tas i bruk i forbindelse med prosjektet.

5.2.1 IntelliJ IDEA og Java

Ettersom det i utgangspunkt skulle brukes Java som programmeringsspråk, ville det være viktig med å ha en IDE som ville gjøre kodingen enklere, uten at det tilbyr inkonsekvenser for utviklingsprosessen. Det ble derfor benyttet Intellij IDEA, ettersom Intellij IDEA allerede er kjent som et bra utviklerverktøy og var også benyttet av oppdragsgiveren.

I sammenheng med IntelliJ ble det tatt en vurdering om prosjektet skulle ta i bruk Maven eller Gradle som byggesystem for programmene som ble skrevet. Disse to ble vurdert fordi de begge har muligheter for å enkelt legge til «dependencies» for ulike biblioteker som ble tatt i bruk under prosjektet. Både Maven og Gradle ville la seg gjøre for bruk under utviklingen, men oppsettet på disse to var ganske så forskjellig fra hverandre.

Maven:
Maven er et automatisk byggesystem som blir brukt for å fortelle systemet hvordan programmet skal bygges opp og hvilke avhengigheter som må på plass. Når programmet blir bygget, vil Java biblioteker og plug-ins dynamisk lastet ned fra et Maven repository. I denne sammenhengen ble det brukt Maven Central Repository.

Gradle:
Gradle er også et automatisk byggesystem i likhet med Maven, men benytter seg ikke av XML, men istedet benytter seg av et domene-spesifikt språk basert på Groovy. Gradle prosjekter fungerer på samme måte som Maven ved at det dynamisk lastes ned biblioteker og plug-ins når programmet bygges.

Det ble tidlig avgjort at Gradle var byggesystemet som burde brukes. Dette var fordi Maven føltes veldig tungvindt å bruke og det var vanskeligere å sette seg inn i bruken og oppbyggingen av programmene. Ved å laste ned Gradle, ville man også enkelt kunne lage nye Gradle prosjekter ved hjelp av en enkel kommando som ville sette opp strukturen og filene som trengtes for å kjøre programmene. Det var også enkelt å sette inn dependencier som skulle brukes i programmet, ettersom det kun måtte til en linje per dependency som dynamisk ble lastet ned når programmet ble bygget.

5.2.2 Xamarin og C#

Mobilapplikasjonen skulle fungere både på iOS og Android. Det ble benyttet Xamarin Forms som er skrevet i C# for dette formålet. Forms applikasjoner er bygget opp på den måten at all programkode for den faktiske applikasjonen som skal kjøres gjøres på et sted, hvor både iOS og Android har to forskjellige måter å starte denne koden på. På grunn av dette måtte det vurderes biblioteker som både ville fungere på iOS og Android slik at det ikke ville bli problemer for en av platformene å kjøre programmet. Oppdragsgiver, Electric Time Car, hadde allerede laget en applikasjon som løsningen skulle implementeres i. Ettersom denne var laget for Web og enda var på et tidlig stadium i utviklingen, gjorde den det vanskelig å teste koden i. Det ble derfor valgt å skrive en egen applikasjon for testing, for å gjøre det enklere å sette seg inn i både språk og hvordan Xamarin programmer var satt opp. Det ble først prøvd med å lage applikasjonen uten å bruke forms, men det ble fort funnet ut av at dette ikke ville la seg gjøre ettersom dette ville resultert i to separate applikasjoner som måtte kjøres hver for seg, en for iOS og en for Android. Hadde denne løsningen blitt brukt, ville vedlikehold og utvikling på platformene kostet mye mer tid og resurser enn det som hadde vært nødvendig.

Det gikk mye tid til å sette seg inn i oppbyggingen av Xamarin Forms i Xamarin prosjekter. Det ble også oppdaget at det var en mindre feil med Visual Studio 2017 som gjorde det vanskelig å legge til sider i Forms applikasjonen. Denne feilen kom på grunn av en manglende pakke i Visual Studio 2017 som måtte kjøres når programmet ble bygget. Ekstra-pakken som måtte installeres, kalt «Universal Windows Platform Development», inneholder tillegsfunksjonaliteter til Visual Studio 2017. Denne pakken blir som regel benyttet i sammenheng med cross-platform applikasjoner der også Windows blir brukt. Ettersom det skulle lages en applikasjon for iOS og Android hadde denne pakken egentlig ikke vært nødvendig, men siden det ble programmert på Windows trengtes denne pakken for at programvaren skulle kunne kompileres.

Xamarin hadde en god del feil og utfordringer under utviklingen av applikasjonen. Blant annet klagde den på manglende deklareringer av funksjoner og variabler, selv om disse egentlig allerede var gyldige. Disse feilene var ikke på grunn av feil fra vår side, men problemer med selve Xamarin, noe som i retur gjorde feilsøking vanskeligere ettersom det ikke var sikkert vi faktisk hadde begått en feil i første omgang. Feilene som Xamarin kom opp med, lot seg som regel fikse ved å bruke «clean» og «re-build», og noen ganger måtte problemene løses ved å starte programmet eller datamaskinen på nytt.

5.2.3 NuGet

For å legge til biblioteker i Xamarin benyttet vi NuGet[17] som er en pakkebehandler i .Net applikasjoner. Denne ble også benyttet for å oppdatere bibliotekene/pakkene.

5.2.4 Python

Python ble benyttet i koden for GATT-serveren på Raspberry Pi enheten. Det ble vurdert flere programmeringsspråk for å gjøre denne jobben, men ettersom Bluez allerede var skrevet i Python, ble dette vurdert over de andre mulighetene som eksisterte.

5.3 Applikasjon

5.3.1 Konseptapplikasjon

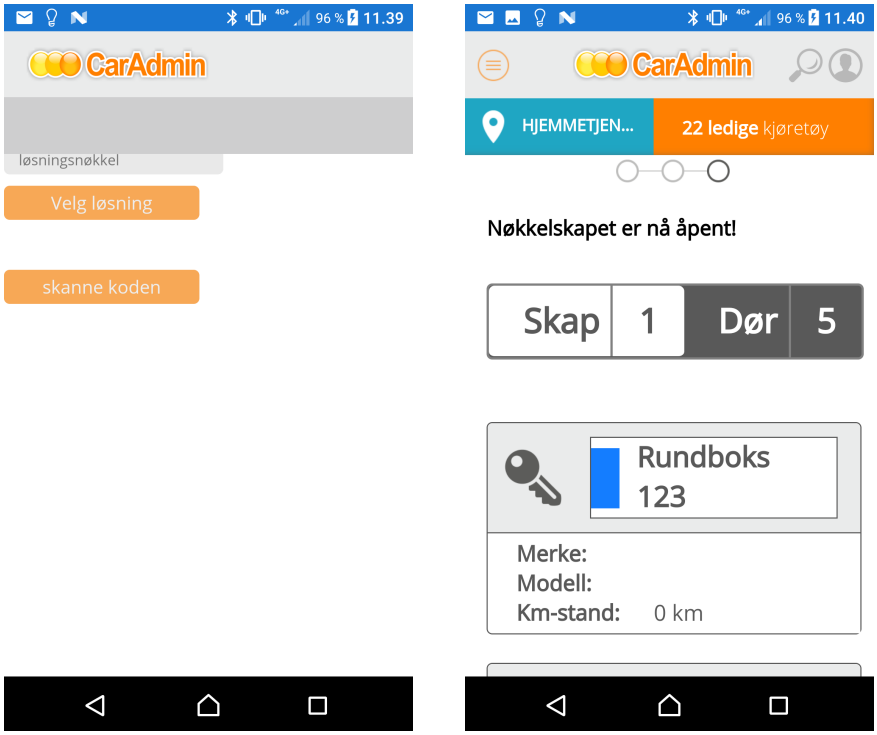
Som sagt tidligere i delkapittel 5.2.2, ble det lagd en konseptapplikasjon slik at det ville være enklere å teste funksjonaliteter som skulle på plass i den endelige løsningen. Denne skulle gjøre det mulig å skanne etter bestemte skap, for å filtrerte ut beacon enheter som tilhører ETC. I konseptapplikasjonen ville en liste over tilgjengelige skap innenfor en gitt rekkevidde bli vist på skjermen til brukeren. I denne listen kunne man velge ut «skapet» man ønsket å koble seg til, deretter henter applikasjonen ut tjenestene skapet hadde å by på. Disse tjenestene var i form av en service og to characteristic som nevnt i 5.1.4. Både service og characteristics ble hentet ut på samme måte, men dette kommer vi tilbake til i kapittel 5.4. Når informasjon om tjenestene er hentet ut vil prosessen for å hente ut den unike ID-en startes, deretter vil konseptapplikasjonen også foreta seg av en sending av denne ID-en til en testserver som var satt opp for testing av server kommunikasjonen. Denne serveren sender så tilbake verdiene som blir benyttet når det skrives til skapet, og GATT-serveren utfører arbeidet videre.

Applikasjonen benytter seg av Xamarin Bluetooth LE biblioteket som ble valgt ut tidligere i prosjekt perioden, som det også ble snakket om i delkapittel 3.2.2. Dette biblioteket måtte brukes for at Bluetooth kommunikasjonen skulle kunne foregå både på Android og iOS, og gjøre det mulig for å oppnå kravene som var satt.

5.3.2 Implementering i oppdragsgivers applikasjon

For å få en bedre visning av applikasjonen og benytte den i et mer reelt scenario, ble koden implementert inn i oppdragsgiver applikasjon for uthenting av kjøretøyer. Implementeringen bydde på problemer med tanke på listen som i konseptapplikasjonen ble vist på mobilskjermen. I implementeringen ble det heller benyttet elementer rett fra listen ved å ta ut det første elementet istedet for å få opp en liste på skjermen der brukeren ville velge skapet selv.

Figur 12a viser hvordan CarAdmin applikasjonen ser ut, der brukeren benytter seg av knappen «skanne koden» før man logger inn, for å skanne etter Bluetooth enheter. Hvis en enhet blir funnet, kobler applikasjonen seg på denne og henter ut informasjon om tjenestene på samme måte som i konseptapplikasjonen. Når dette er gjort, kan brukeren gå gjennom prosessen for uttak. Når brukeren har bestilt et kjøretøy og er klar for å ta ut nøkkelen, trykker han på knappen som viser skap og dør ID, som vist i figur 12b, som



(a) Skanning av skap

(b) Åpning av skap

Figur 12: CarAdmin applikasjonen fra ETC

igjen vil starte funksjonen for å skrive kommando til Bluetooth enheten for å åpne selve skapet. Når prosessen er ferdig vil applikasjonen koble seg av Bluetooth tilkoblingen og brukeren må foreta en ny skann hvis et skap skal åpnes.

5.4 Kodeeksempler

For å forklare konseptene bedre er det tatt med noen eksempler fra kildekoden til de ulike programmene som ble utviklet i prosjektperioden. Disse vil være med på å forklare bedre hvordan systemet fungerer og hvordan de ulike delene kommuniserer med hverandre.

5.4.1 Konseptapplikasjon

For at applikasjonen skulle fungere som forventet, måtte det settes en del tillatelser for å få Bluetooth til å fungere. Disse trengtes bare å settes opp på Android, da det viste seg å ikke være nødvendig å sette opp for vanlig bruk på iOS. For noen Android enheter må det også settes tillatelser for å aksessere lokasjonsdata. For å gjøre skanningen mindre krevende er det også satt opp til bare å skanne etter Bluetooth Low-Energy på Android enheter. I koden 5.2 vises manifest filen for Android i applikasjonen for hvordan tillatelsene er satt opp.

Listing 5.2: Utsnitt fra AndroidManifest.xml

```
<uses-permission android:name="android.permission.
    ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.
    ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.BLUETOOTH"
    />
<uses-permission android:name="android.permission.
    BLUETOOTH_ADMIN"/>
<uses-feature android:name="android.hardware.bluetooth_le"
    android:required="true"/>
```

Ettersom applikasjonen må skanne etter spesifikke enheter, måtte det fortsatt gjennomføres en full skann som finner alle enheter i nærheten. Deretter gjøres det en filtrering av disse for å finne den riktige enheten basert på de forutsetningene som er blitt satt. Denne sjekker om den er av typen servicetype, som Eddystone benytter. Den dobbeltsjekker dette ved å se at lengden faktisk stemmer, og ser på avstanden til enheten for å bestemme rekkevidde. Koden 5.3 følger hvordan dette er gjort ved å benytte en `EventHandler<DeviceEventArgs>`. Denne informasjonen blir videre sjekket opp mot oppdragsgivers navnsetting, og om dette stemmer, blir den lagret i en liste som blir brukt for oppkobling.

Listing 5.3: EventHandler funksjonen fra MainPage.xaml.cs

```
private EventHandler<DeviceEventArgs> Handler()
{
  // Making a new handler to hold the information
  EventHandler<DeviceEventArgs> handler = (obj, a) =>
  {
    foreach (AdvertisementRecord record in a.Device.
        AdvertisementRecords)
    {
    // Only add advertisement records of this specific kind
      if (record.Type == AdvertisementRecordType.ServiceData
          && record.Data.Length >= ExpectedPacketSize && a.
          Device.Rssi >= ExpectedRssiValue)
      {
        ...
      }
    }
  }
}
```

Når en enhet har blitt koblet til, vil applikasjonen hente ut tjenestene som enheten byr på i form av en service og to characteristics. I kodesnutten 5.4 vises hvordan servicen og en av characteristicsene blir hentet ut.

Listing 5.4: GetServices og GetWriteCharacteristic fra MainPage.xaml.cs

```
1  // Gets the service that will be used for read and write
2  private async Task<IService> GetServices(IDevice advDevice)
3  {
4    var service = await advData.Device.GetServiceAsync(Guid.
       Parse("0000180d-0000-1000-8000-00805f9b34fb"));
5    return service;
6  }
7
8  // Gets the characteristics values used for the write
     function
9  private async Task<ICharacteristic> GetWriteCharacteristics(
     IService service)
10 {
11   var characteristicWrite = await service.
       GetCharacteristicAsync(Guid.Parse("00002a39
       -0000-1000-8000-00805f9b34fb"));
12   return characteristicWrite;
13 }
```

Etter at informasjonen om den aktuelle characteristicen er hentet ut kan denne verdien brukes videre. For å kunne skrive til enheten, sendes et Byte array inn med `WriteCharacteristic` og denne blir videre sendt over til enheten for behandling som vist i koden 5.5

Listing 5.5: WriteCharacteristic fra MainPage.xaml.cs

```
1  // Makes a byte array and writes to the characteristic on
     the device
2  private async Task WriteCharacteristic(ICharacteristic
     writeChar, Byte[] byteValue)
3  {
4    if (byteValue != null && byteValue.Length ==
       ExpectedResponseLength)
5    {
6      await writeChar.WriteAsync(byteValue);
7    }
8    else
9    {
10     Console.WriteLine("Value exceeds expectations");
11   }
12 }
```

Deretter vil mobiltelefonen kobles av Bluetooth oppkoblingen og stå klar til en ny skann.

5.4.2 Implementering i oppdragsgivers kode

For å implementere koden fra konseptapplikasjonen inn i CarAdmin applikasjonen måtte det legges til en måte å kjøre koden. Dette gjøres via en knapp på websiden til oppdragsgiver, som starter prosessen for oppkobling og henting av informasjon/tjenester fra Bluetooth enheten. Deretter skal det ved uthenting av kjøretøy kjøres en kommando som sender informasjonen til mikrocomputeren for åpning av en dør på et bestemt skap. Koden 5.6 viser hvordan dette blir kjørt, men de faktiske dataene som ble brukt vises ikke.

Listing 5.6: Web navigation in MainPage.xaml.cs

```
1  private void WebBrowserNavigating(object sender,
       WebNavigatingEventArgs e)
2  {
3    if (e.Url.StartsWith("Valid argument")
4    {
5      if (command == "connectBT")
6      {
7        if (bluetoothCom == null)
8        {
9          Console.WriteLine("Warning! Instance is null");
10       }
11       else
12       {
13         // Searches for bluetooth devices and connects
14         BluetoothConnect();
15       }
16     }
17     else if (command == "openLocker")
18     {
19       if (data.Length == cmd_len)
20       {
21         openLocker('valid data');
22       }
23       else
24       {
25         Console.WriteLine("Error: Wrong amount of commands")
               ;
26       }
27     }
28   }
29 }
```

5.4.3 Konverterer

For å unngå at behandling av data skulle skje eksklusivt i kode som er dekket av GNU GPL3, ble det laget et mellomledd mellom kontrollerbiblioteket og GATT-serveren. Denne koden har som oppgave å motta og bygge opp den endelige kommandoen som sendes til kontrollerbiblioteket som styrer skapkontrolleren.

5.4.4 Styring av skap

For å kunne styre skapet og sende kommandoene til kontrolleren ville det bli nødvendig å identifisere selve kontrolleren. For å gjøre dette, søkes det gjennom alle de tilkoblede enhetene for å finne ut av hvilken enhet som matcher kravene. Dette gjøres ved hjelp av `findDevice()` funksjonen. Denne har tatt inspirasjon fra 'usb4java's kodeeksempel[18] på hvordan man skal iterere over alle enhetene som er koblet til systemet og hente ut adressen til en spesifikk enhet.

Listing 5.7: `findDevice()`

```
try {
  // Iterate over all devices and scan for the right one
  for (Device device : list)
  {
    DeviceDescriptor descriptor = new DeviceDescriptor();
    result = LibUsb.getDeviceDescriptor(device, descriptor);

    if (result != LibUsb.SUCCESS)
    {
      throw new LibUsbException("Unable to read device
          descriptor", result);
    }

    if (descriptor.idVendor() == vendorId && descriptor.
        idProduct() == productId)
    {
      DeviceDetails(device, list);
      return true;
    }
  }
} catch (Exception e) {
  // Free the DeviceList on error
  LibUsb.freeDeviceList(list, true);
}
```

Når enheten er blitt funnet, lagres disse verdiene unna slik at de kan brukes senere for å kommunisere med enheten. For å kunne gjøre det mulig å kommunisere med selve enheten, blir det nødvendig å lage en 'Handle' som kan brukes til å sende og motta data fra.

Listing 5.8: `DeviceHandle()`

```
/**
 * Sets up a DeviceHandle that can be used to send and
    receive data to and from a device
 * @param  device  The address of the device
 * @return Returns A DeviceHandle for the current device
 */
private DeviceHandle deviceHandle(Device device)
{
  int result;
  DeviceHandle handle = new DeviceHandle();
```

```
10    System.out.println("deviceHandle(" + device + ")␣setup:␣
          Initializing␣handle");
11
12    // Open a new DeviceHandle to communicate with the device
13    result = LibUsb.open(device, handle);
14    if (result != LibUsb.SUCCESS)
15    {
16      throw new LibUsbException("Unable␣to␣open␣USB␣device",
            result);
17    }
18
19    // Reset the device
20    System.out.println("Resetting␣the␣device");
21    result = LibUsb.resetDevice(handle);
22    if (result != LibUsb.SUCCESS)
23    {
24      throw new LibUsbException("Unable␣to␣reset␣device␣handle
            ", result);
25    }
26
27    // Check if a kernel driver is attached to the interface.
          If so, we will need to detach it.
28    System.out.println("Checking␣if␣a␣kernel␣driver␣is␣
          attached");
29    result = LibUsb.kernelDriverActive(handle, dev_interface);
30    if (result == 1)
31    {
32      System.out.println("Warning!␣Kernel␣driver␣detected");
33      detachKernelDriver(handle, dev_interface);
34    }
35
36    // Check if the device is set to the correct configuration
37    System.out.println("Checking␣device␣configuration");
38    result = LibUsb.getConfiguration(handle, IntBuffer.
          allocate(8));
39    if (result != dev_conf)
40    {
41      LibUsb.setConfiguration(handle, dev_conf);
42      System.out.println("New␣device␣configuration␣set");
43    }
44
45    System.out.println("Attempting␣to␣claim␣Interface");
46    result = LibUsb.claimInterface(handle, dev_interface);
47    if (result != LibUsb.SUCCESS)
48    {
49      throw new LibUsbException("Unable␣to␣claim␣interface",
            result);
50    }
51
52    System.out.println("deviceHandle(" + device + ")␣setup:␣
          Handle␣initialized␣(" + handle + ")");
53    return handle;
54  }
```

På grunn av måten UNIX/Linux behandler HID enheter må man kunne sjekke om den er koblet til en såkalt Kernel driver. En Kernel driver er bare en standard driver for kommunikasjon, men denne er desverre ikke støttet da man ikke kan behandle kontrolleren som en minepenn. For å fikse dette, kjøres en kommando for å koble fra denne driveren før programmet fortsetter som vanlig.

Listing 5.9: `detachKernelDriver()`

```
1  /**
2   * Detach an active kernel driver from a device
3   *
4   * @param handle          A DeviceHandle to the device
5   * @param deviceInterface The interface  to detach
6   */
7  private void detachKernelDriver(DeviceHandle handle, int
       deviceInterface)
8  {
9    int result;
10
11   // Detach device from kernel. We will not be able to
         communicate with an attached device
12   System.out.println("Attempting to detach kernel driver");
13   result = LibUsb.detachKernelDriver(handle, deviceInterface
         );
14   if (result !=LibUsb.SUCCESS && result !=LibUsb.
         ERROR_NOT_SUPPORTED && result !=LibUsb.ERROR_NOT_FOUND)
15   {
16     throw new LibUsbException("Unable to detach kernel
           driver", result);
17   }
18 }
```

Når alt dette er blitt utført skal det være satt opp en brukbar 'Handle' som kan brukes til å sende og motta data til og fra enheten. Denne koden er kodet til å fungere for en hvilken som helst HID enhet, og ikke bare spesifikt for oppdragsgiverens skapkontroller.

På grunn av ønske fra oppdragsgiver, vil ikke eksempler på hvordan data sendes og mottas for selve kontrolleren vises. Dette er av sikkerhetsmessige grunner da prosjektet er tiltenkt å brukes videre i et av deres produkter.

5.4.5 GATT-server

GATT-serveren har, som beskrevet i delkapittel 5.1.4, to characteristics som lar applikasjonen lese en byte verdi, og skrive en byte verdi tilbake til enheten. For å lese fra enheten er det satt opp en characteristics klasse `ReadLockerIdChrc` som har en vilkårlig UUID brukt for å identifisere characteristicen. Denne characteristicen er satt opp til å bare ta seg av 'read' kommandoer, og vil returnere en bestemt byte verdi som skal fungere som en unik ID for et skap. I koden 5.10 vises hvordan klassen definerer den returnerte verdien når applikasjonen leser fra enheten:

Listing 5.10: Characteristic for lesing fra enheten

```
def ReadValue(self, options):
  # Return a 128bit unique locker ID
  return
  [
    0x18, 0x04, 0x02, 0x18, 0x04, 0x02, 0x18, 0x04,
    0x02, 0x18, 0x04, 0x02, 0x18, 0x04, 0x02, 0x18
  ]
```

For å skrive en byte verdi tilbake til enheten, måtte det også settes opp en egen klasse, `TransferLockerIdChrc`, som også har en vilkårlig UUID og er satt opp med å bare ta imot 'write' kommandoer isteden for 'read'. Koden under viser også hvordan denne håndterer informasjonen sendt inn til enheten. Denne tar imot en kommando av gitt lengde som blir formatert til hexadecimal på formatet «xFF» og deretter sendes dette med som en parameter i konverterings funksjonen `LockerReceiver-all.jar`.

Listing 5.11: Characteristic for skriving til enheten

```
def WriteValue(self, value, options):

  if len(value) != ExpectedLength:
    raise InvalidValueLengthException()

  for i in range(0, 'Length'):
    response+='{}{:02x}'.format('x', int(value[i]))

  # Runs the program with the correct arguments
  call(["java", "-jar", "./LockerReceiver-all.jar", response
     ])
```

5.5 Verktøy

Utenom verktøyene som ble brukt for programmering av løsningen, ble det også benyttet følgende verktøy i prosjektperioden:

- **ShareLaTeX**: Lage rapporten og generere PDF filen brukt for å levere prosjektoppgaven. Denne benyttet seg av strukturen funnet i malen for bachelor rapporter ved NTNU i Gjøvik.
- **Google Drive**: Nettbasert skyløsning brukt for deling av diverse dokumenter benyttet under prosjektet.
- **Draw.io**: Enkelt nettbasert verktøy for å lage diagrammer, UML-modeller og andre figurer i rapporten.
- **NextCloud**: Skybasert fillagrings- og delingsklient som ble brukt for deling av lokalt lagrede ressurser for prosjektet.
- **FileZilla**: FTP klient som ble brukt for å overføre og redigere filer over SSH på mikrocomputeren.
- **PuTTY**: En SSH og Telnet klient som kan brukes til å fjernstyre systemer via terminal. Denne ble benyttet for å administrere og kode på Raspberry Pi enheten.

Det ble også benyttet verktøy underveis til å teste løsningen. Disse vil bli beskrevet i større detalj i kapittel 6.

6 Testing og kvalitetssikring

Vi hadde lite erfaring med systematiske tester fra studietiden, som f.eks. enhetstester (unit tests) eller integrasjonstester. Pga. tidsrammen i dette prosjektet ble det tatt en avgjørelse om å heller ikke settes inn i dette. Dette valget ble gjort på bakgrunn av at brukerinteraksjon i svært liten grad er en del av prosjektet, og at mesteparten av input og flyt ble basert på automatikk. Det har naturligvis vært nødvendig å teste at funksjonaliteten likevel gjør det den skal, men dette ble gjort på en mer ikke-formel måte av på prosjektgruppen.

6.1 Applikasjon

Ettersom applikasjonen skulle fungere både for Android og iOS, måtte denne også testes mot begge platformene. Applikasjonen ble testet til å fungere på alle gruppemedlemmenes telefoner som innebar to Android mobil og en iOS mobil.

6.2 Serverkommunikasjon

Vi hadde ikke tilgang til oppdragsgivers server, hvor logikken for uthenting av et kjøretøy lå. For å få testet at flyten i systemet fungerte som den skulle, ble det derfor nødvendig å emulere serverlogikken ved å lage en egen mock-server. Det ble laget et REST API i Golang for denne delen, grunnet tidligere erfaring med dette språket fra *IMT2681 - Cloud Technologies* gjorde det enkelt å sette opp.

I praksis fungerte mock-serveren som en listener, som ventet på en nøkkelskap-ID fra mobilapplikasjonen. Mobilapplikasjonen hentet ID-en til nøkkelskapet via Bluetooth, og sendte denne til mock-serverens listener-adresse, som ble definert som en *route* i Go-programmet. Det ble valgt å benytte en tredjeparts HTTP-router (også kalt *multiplexer*)[19], da denne viste det seg å være enklere og mer fleksibel å bruke enn den innebygde Go-routeren. Koden i 6.1 viser oppsett av mock-serveren:

Listing 6.1: Oppsett av mock-server i Go

```
1 // Initialize new http router
2 router := httprouter.New()
3
4 // Listens for a locker ID on "/locker" path
5 router.POST("/locker", LockerIDListener)
6
7 http.ListenAndServe("localhost:8080", router)
```

Da en HTTP-request av typen POST ankom på */locker*-routen, så ville funksjonen `LockerIDListener` kjøres, for så å behandle requesten inne i denne. `LockerIDListener` dekoder JSON-dataene og legger de i objektet `lockerID`, som er av typen `LockerID`. Etter ønske fra oppdragsgiver, så er deler av denne koden ikke vist i detalj, da dette potensielt vil kunne gjøre det enklere å utsette serveren deres for ondsinnede handlinger. Koden er derfor istedet forklart med ord.

Listing 6.2: Håndtering av POST-requests på mock-server

```
1 func LockerIDListener(w http.ResponseWriter, r *http.Request
      , p httprouter.Params)
2 {
3   // Empty LockerID object to decode JSON into
4   lockerID := LockerID{}
5   json.NewDecoder(r.Body).Decode(&lockerID)
6   // More code
7 }
```

Etter at `lockerID` er fylt med en nøkkelskap-ID, sendes objektet til en ny funksjon `calculateDoorID`, som «kalkulerer» verdiene som trengs (verdiene er hardkodet i testene). `calculateDoorID` returnerer verdiene, og disse skrives som en HTTP-response. Response-pakken sendes tilbake til mobilapplikasjonen, hvor den sendes videre til mikrocomputeren. På mikrocomputeren behandles verdiene ytterligere, før de brukes til å åpne en fysisk skapdør.

I konseptapplikasjonen er funksjonen `PostId(System.Byte[] bytes)` som vist i koden 6.3, som tar seg av jobben for å sende og motta fra mock-serveren, satt opp til å sende den unike skap ID-en etter å ha hentet denne ut fra mikrocomputeren. Funksjonen tar i mot et byte array, som så videre sendes til serveren, og som deretter gjør jobben som beskrevet over til å returnere verdien som funksjonen videre returnerer for bruk i `WriteCharacteristic` som beskrevet tidligere i delkapttel 5.4.1.

Listing 6.3: Sending av post request og behandling av svar

```
1  public async Task<Byte[]> PostId(System.Byte[] bytes)
2  {
3      // Converts the byte array to string
4      string id = BitConverter.ToString(bytes).Replace("-",
           string.Empty);
5
6      // Sets mocked Eddystone-ID to the message
7      var message = new GetKeyMessage(btId: id);
8
9      // Creates URI for web service
10     var uri = new Uri(string.Format("https://bb81c306.ngrok.
           io/locker", string.Empty));
11
12     /*
13      * Serialises the message, converts it to byte arrey (
            content).
14      * content is sent asynchronously to the URI trough
            PostAsync.
15      */
16     try
17     {
18         var json = JsonConvert.SerializeObject(message);
19         var content = new StringContent(json, Encoding.UTF8,
               "application/json");
20         var response = await client.PostAsync(uri, content);
21         if (response.IsSuccessStatusCode)
22         {
23             var respContent = await response.Content.
                   ReadAsByteArrayAsync();
24             Console.WriteLine("JSON-data␣was␣sent,␣Response:
                   ␣{0}", BitConverter.ToString(respContent));
25             ConfirmText.Text = "Sending␣successful";
26             ConfirmText.IsVisible = true;
27             return respContent;
28         }
29         else
30         Console.WriteLine("Error␣while␣sending␣the␣JSON-data
               ");
31     }
32     catch (HttpRequestException ex1)
33     {
34         Console.WriteLine("Http␣Error:␣{0}", ex1);
35     }
36     return null;
37 }
```

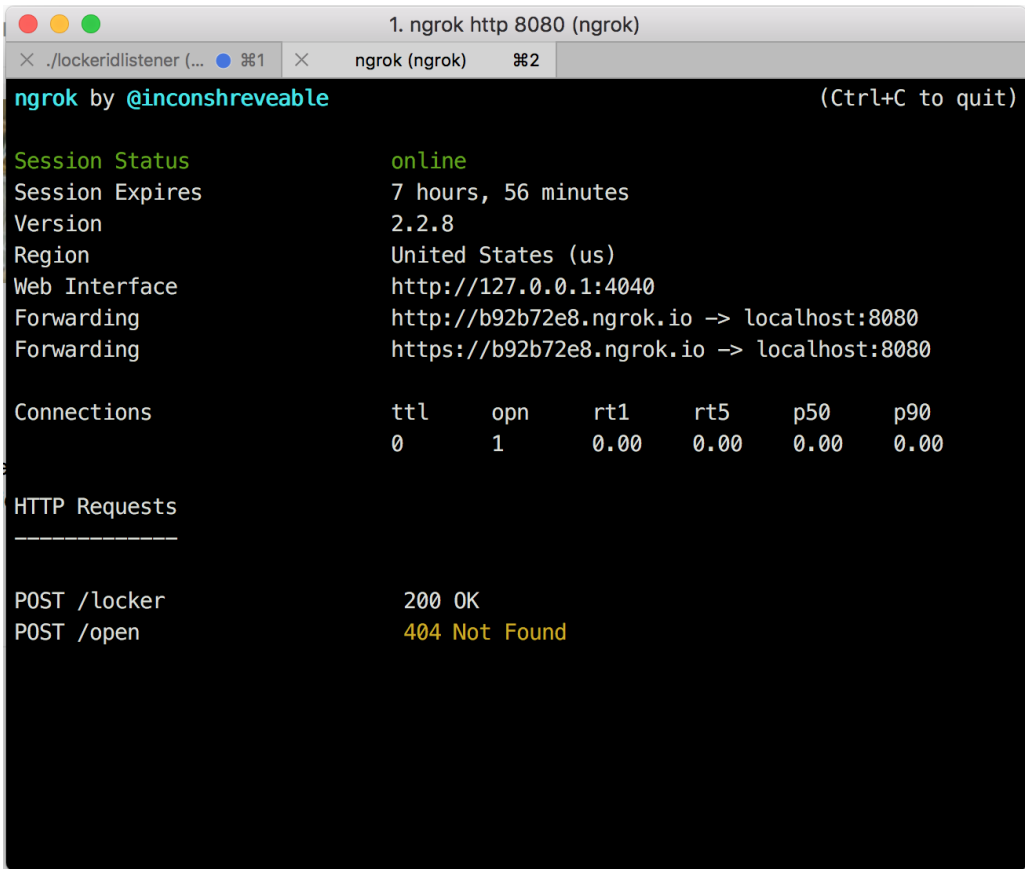
6.3 Utviklingsverktøy

Det ble underveis i prosjektet nødvendig å bruke enkelte verktøy for å tillate utvikling og testing av deler av systemene.

6.3.1 ngrok

Under prosjektet ble det oppdaget at utvikling og testing av nettverkskommunikasjon kunne bli en utfordring på grunn av brannmuren på nettverket systemet kjørte på. Da det ikke ville være mulig å få tilgang til å endre de nødvendige innstillingene i brannmuren, ville det ikke være mulig å åpne direkte kommunikasjon mellom enhetene i programmene og testserveren i prosjektet.

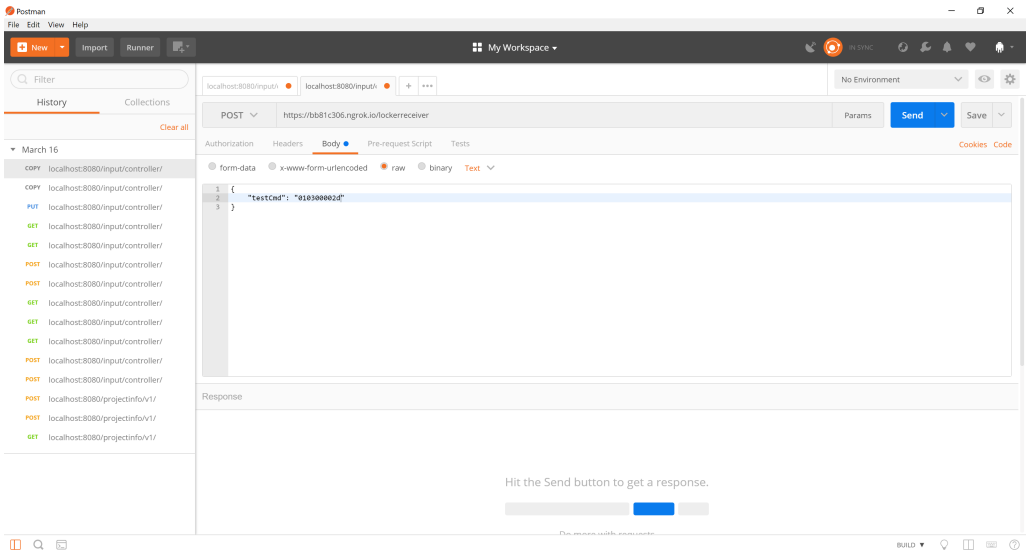
Det er her ngrok[20] kommer inn. ngrok er en tjeneste som oversetter datamaskinens interne, lokale adresse til en fullt brukbar, ekstern adresse for kommunikasjon til tross for NATs og brannmurer. Mock-serveren som ble nevnt tidligere i dette kapitlet ble bl.a. kjørt via ngrok.



Figur 13: Eksempel på kjørende ngrok-server.

6.3.2 Postman

Postman[21] er et utviklerverktøy som lar brukere teste og utvikle nettverksbaserte API-er uten å måtte sette opp avanserte testsystemer selv. Postman har verktøy som gir mulighet for å teste funksjonalitet for hele API-ets utviklingstid, fra design til testing og helt frem til produksjon. Postman ble brukt av under utvikling av kommunikasjon mellom server og mobil, og viste seg nyttig for å kunne finne alternativer til overføring av dataen som skulle sendes i systemene.

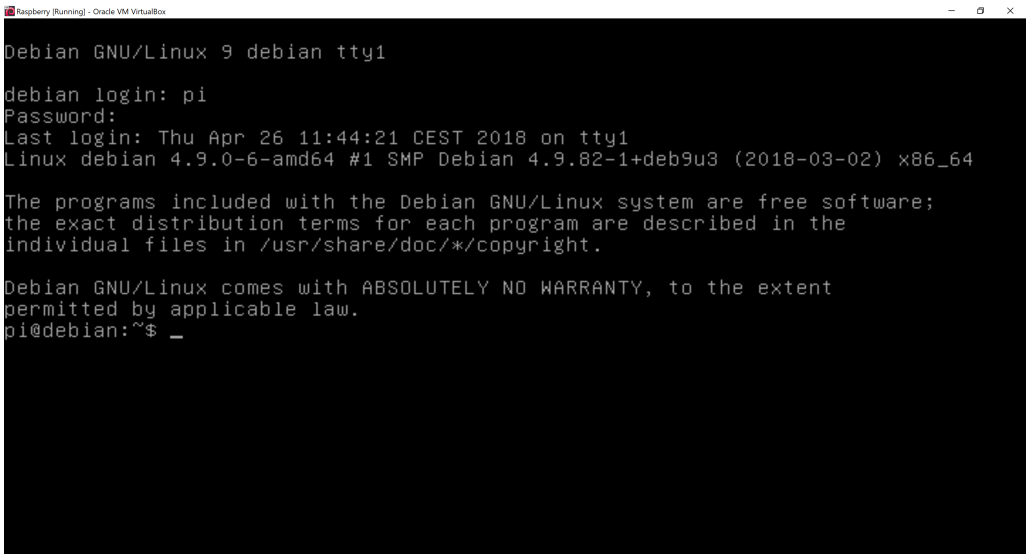


Figur 14: ngrok-serveren testes med JSON i Postman.

6.3.3 Oracle VM VirtualBox

Da prosjektet kom til det punktet hvor programvare skulle utvikles for mikrocomputeren, støtte gruppen på et mindre hinder; gruppen bestod av tre utviklere og hadde bare en enhet å dele på. Dette viste seg å være delvis overkommelig så lenge arbeid på mikrocomputeren ble gjort i samme lokale, men om noen skulle jobbe hjemme ble det mye planlegging og prioritering på hvem som skulle ha tilgang til enheten. Det ble bestemt at det å kunne kjøre egne lokale, virtuelle systemer ville være en nødvendighet for god arbeidsflyt. Etter noe lesing om relevante teknologier, ble valgt å gå for Oracle VM VirtualBox[22] for å kunne kjøre Linux distribusjonen kjent som Debian.

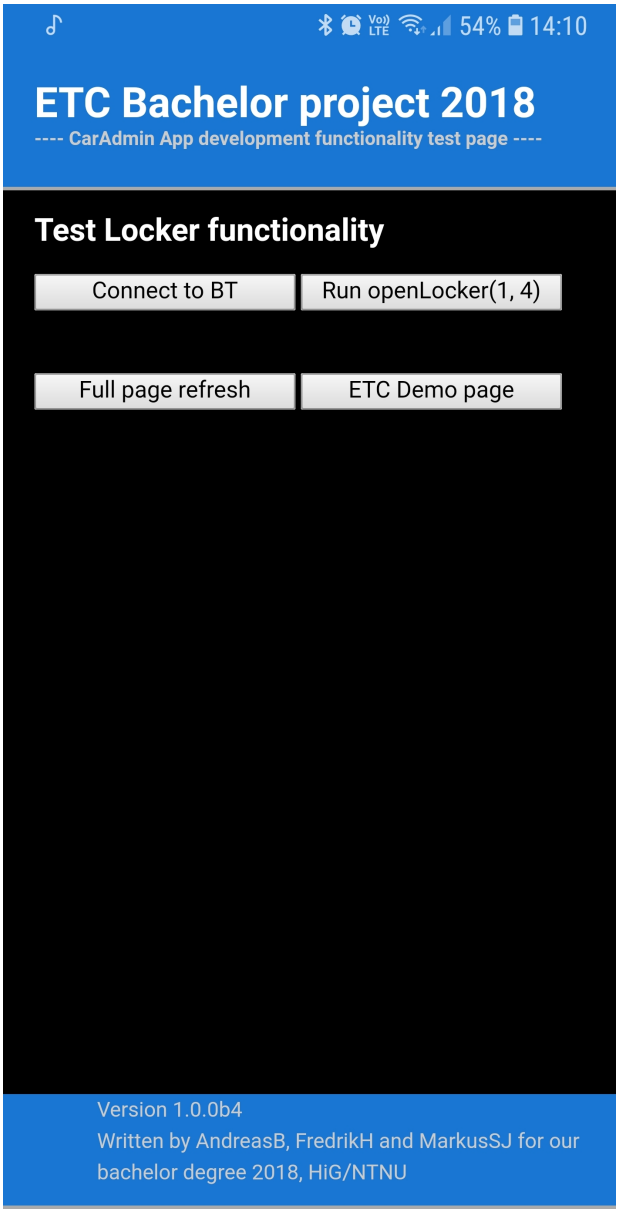
Oracle VM VirtualBox er en programvare som er kjent som en Virtual Machine. En Virtual Machine gjør det mulig å kjøre programvare på lukkede, virtuelle systemer uten å trenge dedikert, fysisk utstyr å kjøre på. Ved å kunne utnytte denne teknologien, kunne alle programmere på et Raspberry Pi lignende system hver for seg uten å måtte ha egne mikrocomputere. Dette forenklet både arbeidsflytt og sparte kostnader på innkjøp for utviklingsverktøy til prosjektet.



Figur 15: Innloggingsbildet i Debian kjørende på VirtualBox.

6.3.4 HTML Testside (ETC_APP_INTEGRATION-TESTING)

For å kunne teste koden implementert i oppdragsgivers eksisterende applikasjon, ble det utviklet en nettside med kode fokusert på å teste funksjonaliteten i applikasjonen. Denne nettsiden ble utviklet i HTML og kjører noen enkle JavaScript funksjoner som blir snappet opp av applikasjonen som den reagerer på.



Figur 16: Startbildet på testsiden.

7 Avslutning

I dette kapitlet evaluerer vi oppgaven mer helhetlig. Dette innebærer blant annet diskusjon av resultatene vi har oppnådd. Er det noe vi kunne eller burde gjort annerledes i løpet av prosessen, som kunne gitt andre resultater? Vi ser også på hva som ville vært det neste naturlige steget i prosessen: altså hva som bør gjøres for å optimalisere det nåværende produktet.

7.1 Diskusjon og drøfting

7.1.1 Resultater

Den endelige løsningen vår inneholder alle de grunnleggende kravene som ble satt i kapittel 2. I tillegg til grunnfunksjonaliteten som oppdragsgiver ønsket, så fikk vi også implementert en ekstrafunksjon. Det var et ønske om at systemet skulle fungere selv om mikrocomputerne ikke hadde tilkobling til internett. Systemet ble derfor designet med hensyn på dette.

På den opprinnelige prosjektplanen (vedlegg C), så var det også ønsket at en administrator skulle kunne koble seg på mikrocomputeren via SSH. Dette var også en ekstrafunksjonalitet. Denne kom vi dessverre ikke i mål med, da andre deler av prosessen tok mer tid enn forutsett. En annen ekstrafunksjon som ble nevnt var muligheten for å oppdatere programvaren på mikrocomputeren, *uten* å være i nærheten av den. Av samme grunn som over, så ble heller ikke dette implementert.

Oppgavens aspekter har under hele prosjektet vært klart, og det har ikke blitt gjort store endringer i hvordan denne funksjonaliteten skal foregå. De endringene som faktisk har måttet blitt gjort, har vært i form av hvilke komponenter som skal kommunisere sammen på hvilken måte, for å best mulig tilpasse dette til kravene som er satt.

7.1.2 Alternativer

Underveis ble det tatt flere valg om hvordan systemet skulle ende opp. Dette ble flere ganger revidert, på bakgrunn av tilbakemelding fra oppdragsgiver, eller at noe ikke ville la seg gjøre innenfor tidsrammene som var satt.

Et stykke ut i prosjektperioden oppdaget vi at vi ikke kunne benytte oss av beacons slik vi først hadde tenkt. Dette ble tidligere beskrevet i delkapittel 5.1.2. Her ble det også beskrevet problemer med oppstart av Bluetooth etter at en mobil har koblet av enheten.

Løsningen skulle i utgangspunktet fungere slik at det var mikrocomputeren som kommuniserte med CarAdmin-serveren. Dette ville tatt for lang tid å fikse opp i ettersom vi måtte legge til nye funksjonaliteter for å omgå en brannmur som gjorde at vi ikke fikk sendt informasjon fra serveren til mikrocomputeren. Løsningen vi allerede hadde på dette tidspunktet var i for seg en ferdig løsning der vi fikk kommunikasjonen til å gå hele veien gjennom systemet via testserveren vår. På grunn av at brannmuren ville stoppet pakken som ble sendt, måtte denne bli sendt fra mikrocomputeren istedet, som igjen vil si at det måtte til en initialisering av kommunikasjon mellom disse enhetene når en pakke skulle ha blitt sendt/mottatt. Vi unngikk derfor dette helt ved å endre til en løsning der serverkommunikasjonen går mellom server og mobil.

Nær slutten av prosjektperioden viste det seg at lisensen som ble benyttet for biblioteket usb4java ikke var oppdatert på deres hovedside, men hadde i senere tid blitt endret til MIT-lisens. Dette hadde ikke noen stor betydning for prosjektet, men ville gitt oss mulighet til å forenkle implementasjonen av kodebiblioteket brukt i denne delen. Dette var fordi vi da ikke ville trengt å forholde oss til at koden måtte lastes opp for andre å se om det ble gjort endringer i biblioteket eller linket til det statisk.

7.2 Kritikk av oppgaven

I oppgaveteksten er det nevnt at oppgaven passer for grupper med størrelse på en til tre personer. Vi var tre personer i vår gruppe, og har vanskeligheter med å se for oss at resultatet vårt ville blitt det samme med bare én eller to på gruppa. Da måtte vi i så fall fra tidligere hatt gode kunnskaper om teknologi og metoder som ligger utenfor vårt pensumområde.

Den første utfordringen var å utvikle biblioteksfunksjoner for skapkontrolleren. Selv om vi fikk kildekoden for det allerede eksisterende kontrollerbiblioteket, så var det en relativt stor jobb å «oversette» denne fra C# til Java, også med tanke på at den nå skulle kjøre på en ARM-prosessor på en Raspberry Pi.

Bluetooth Low Energy var også en omfattende teknologi som det var relativt krevende å forstå. BLE består av mange protokoller som virket i forskjellige faser av kommunikasjonsprosessen. Disse måtte i tillegg implementeres på to forskjellige plattformer, både på mikrocomputeren og i mobilapplikasjonen.

Å utvikle på mobilplattform var også en ny erfaring. Selv om det har mye til felles med utvikling av skrivebordsapplikasjoner, så var det også mange nye konsepter som måtte tas hensyn til og læres.

7.3 Videre arbeid

For at løsningen skal tas i bruk i kommersielt, må det videre tenkes på sikkerhet i Bluetooth-kommunikasjon, og optimalisering av koden som er blitt utviklet, for at den skal tilpasses bedre hvordan oppdragsgiver skal benytte seg av dette videre. Videre vil vi forklare noen komponenter i løsningen vi har laget, som kan forbedres og evt. benytte andre måter for sikkerhet, noe som er viktig å jobbe videre med for at ikke noen skal kunne utnytte løsningen i form av sikkerhetshull. Koden vi har utviklet er ment som en byggestein for ETC, som kan jobbes videre på for å få en mer komplett løsning for åpning av skapene under uttaksprosessen av kjøretøy.

7.3.1 Eddystone

I analysedelen (kapittel 3) forklarte vi hvorfor vi benytter oss av Eddystone som Beacon-teknologi. Raspberry Pi-enheten kringkaster en datapakke med en forhåndsbestemt og hardkodet identifikator. Dette er *Eddystone-UID*. Google har også et sikrere alternativ, *Eddystone-EID*. Denne har, i motsetning til Eddystone-UID, ikke en forhåndsbestemt identifikator, men istedenfor en ID som forandrer seg med jevne mellomrom. Dette hindrer at uvedkommende kan utnytte datapakken, da det kreves kommunikasjon med en kryptert bakgrunnstjeneste for å kunne bruke signalet til noe fornuftig. Eddystone-EIDs offisielle sider bruker følgende som eksempler på situasjoner hvor den bør brukes:

- Dersom man ønsker å hindre at andre skal benytte seg av din Beacon
- Sikre personvern i situasjoner hvor brukeren har en BLE-enhet på seg
- Dersom bedrifter som leaser en Beacon ønsker en «power off»-knapp
- Dersom det trengs et sterkt signal om at en bruker befinner seg på en spesifikk lokasjon, uten at det er mulig å *spoofe*[1] signalet.

Ved prosjektslutt overleveres koden til oppdragsgiver med dokumentasjon om hvordan den brukes, hvordan den settes opp og hvilke krav som gjelder for hver del av prosjektet.

7.4 Evaluering

7.4.1 Innledning

Dette var første gang vi tre hadde jobbet sammen på gruppe. Markus og Fredrik hadde allerede jobbet på gruppe i større prosjekter tidligere. Selv om dette var tilfellet, jobbet vi bra sammen fra starten av. Gruppen har heller ikke hatt store uenigheter underveis, og alle har hjulpet hverandre der noe har vært uklart.

[1] Spoofing: en person eller et program utgir seg for å være en annen ved å etterlikne denne.

Oppgaven i sin helhet har for det meste gått bra, selv om det har gått mer tid til utvikling enn å skrive på rapporten. Vi har i alle tilfeller blitt enige i spørsmål som har dukket opp og for tilnærminger for løsningen. Det har heller ikke vært noen konflikter i gruppen.

7.4.2 Organisering

Med tanke på organisering av dokumenter, har vi ikke gjort noen endringer ut ifra hva som allerede var oppført i prosjektplanen, vedlegg C. Vi har alle byttet på hvem som skal ha ansvar for kommunikasjon angående møter og sammendrag mellom oss og oppdragsgiver, der Andreas og Markus har stått for det meste av denne kommunikasjonen, og hvor Fredrik stod for det meste for den kommunikasjon angående det tekniske i oppgaven.

Prosjektgruppen har for store deler av prosjektet jobbet på skolen, med visse unntak med tanke på påskeferie, sykefravær og røde dager. Det har bare vært krav om å jobbe på arbeidsdager, men vi har også gjort en del arbeid utenom dette, som ikke har kommet med i arbeidsloggen, vedlegg H.

For å strukturere rapporten på en god måte ble det boken «Bacheloroppgaven»[23] benyttet. Denne hadde mange gode råd ift. skriving og regler som blir brukt for å utforme rapporten bedre.

Med tanke på Scrum har dette i liten grad blitt fulgt utover sprintene som ble satt opp i fremdriftsplanen i prosjektplanen, vedlegg A. Dette vil si at vi benyttet oss av sprinter der det dukket opp nye krav underveis, men benyttet oss mer av en Fossefall-tilnærming for selve arbeidet. Det har til tider vært misforståelser og uklarheter mellom oppdragsgiver og gruppa, spesielt ift. krav og funksjonalitet. Dette er noe vi kan ta selvkritikk på. Sett i ettertid, så burde vi ha vært bedre på å dokumentere og visualisere hva vi tenker, og hvordan vi oppfatter oppdragsgivers ønsker. Her var det en stor fordel at vi benyttet oss av sprinter, og hadde jevnlige møter med oppdragsgiver. Dette ga oss mulighet for å oppklare misforståelsene på et tidlig tidspunkt, og forandre produktet i riktig retning. I vårt tilfelle, måtte vi flere ganger endre på oppsettet for hvordan kommunikasjonen fungerte, som igjen førte til at vi måtte omskrive deler av koden. Etter hver sprint har det alltid vært noe å vise til, og vi har prøvd å følge det som var nevnt i fremdriftsplanen A så godt det har latt seg gjøre.

7.4.3 Fordeling av arbeidet

Organiseringen av arbeidet har vært gjort på måten som nevnt i kapittel 1.9 og har vært slik gjennom hele prosjektet. Dette har gjort at vi har jevnt fordelt oppgavene for alle gruppemedlemmer og alle har hatt ting å gjøre til en hver tid.

Utenom dette hadde vi også delt opp arbeidet slik at Fredrik hadde ansvaret for Java-programmene, og Markus og Andreas hadde ansvaret for Xamarin.Forms-applikasjonen og Bluetooth-kommunikasjonen. Vi føler at det var nødvendig å dele disse opp slik at vi fokuserte på ulike deler for å kunne fullføre prosjektet. Dette vil si at vi alle har deler av

prosjektet vi har mer kunnskaper innenfor enn noen av de andre på gruppen. Vi kunne nok vært bedre på å fordele arbeidet mellom hverandre ved å benytte prosjektorganiseringsverktøyer. Trello ble ikke brukt i den grad vi tenkte, men mer for å faktisk si hvilke oppgaver som eksisterte, og ikke hvilke som var knyttet til de ulike prosjektdeltakerne.

7.4.4 Prosjekt som arbeidsform

Denne prosessen har gitt oss nye erfaringer innenfor større prosjektarbeid, noe som vil komme godt med i arbeidslivet. Å måtte jobbe i gruppe med flere personer har gjort at vi har blitt bedre til å strukturere arbeid og roller i en mindre prosjektgruppe. Å ha prosjekt som arbeidsform gjør at vi har ekstra fokus på å gjennomføre et godt prosjekt, som videre fører til et godt produkt. Det å benytte seg av prosjekter, er en normal måte for utviklingsbedrifter å arbeide på. Dette gjør at vi får et mer realistisk bilde av hvordan det faktisk blir å jobbe i en utviklerjobb.

7.5 Konklusjon

Prosjektet har i sin helhet vært interessant og utfordrende. Det har også vært veldig lærerikt, og vi har fått stort utbytte med tanke på ferdigheter og kunnskaper vi har oppnådd i løpet av denne perioden. Vi har fått god bruk for de kunnskapene vi har oppnådd gjennom tre år ved NTNU i Gjøvik.

Slik vi ser det, så har vi gjennomført det prosjektoppgaven etterspør på en måte vi kan være fornøyde med. Vi fikk dessverre ikke implementert all ekstrafunksjonaliteten som ble nevnt ved prosjektstart, men til gjengjeld tenker vi at produktet som *er* ferdig, ble gjort på en skikkelig måte.

Vi håper at oppdragsgiver får nytte av dette prosjektet og kan benytte seg av produktet vi har laget.

Bibliografi

[1] Google. Eddystone. https://github.com/google/eddystone. [Online; Aksessert 11. mars 2018].

[2] Chris Riesgo. Android AltBeacon Library. https://github.com/chrisriesgo/Android-AltBeacon-Library. [Online; Aksessert 10. mars 2018].

[3] Universal Bluetooth Beacon Library. https://github.com/andijakl/universal-beacon. [Online; Aksessert 10. mars 2018].

[4] Bluetooth LE plugin for Xamarin. https://github.com/xabre/xamarin-bluetooth-le. [Online; Aksessert 10. mars 2018].

[5] GNU General Public License. https://www.gnu.org/licenses/gpl-3.0.en.html. [Online; Aksessert 27. februar 2018].

[6] GNU Lesser General Public License. https://www.gnu.org/licenses/lgpl-3.0.en.html. [Online; Aksessert 27. februar 2018].

[7] Apache License Version 2.0. https://www.apache.org/licenses/LICENSE-2.0. [Online; Aksessert 27. februar 2018].

[8] The MIT License. https://opensource.org/licenses/MIT. [Online; Aksessert 16. mars 2018].

[9] usb4java — USB library for Java based on libusb 1.0. https://github.com/usb4java/usb4java. [Online; Aksessert 15. mars 2018].

[10] Ian Sommerville. 2016. *Software Engineering TENTH EDITION*. Pearson Education Limited.

[11] Bonjour Print Services for Windows v2.0.2. https://support.apple.com/kb/DL999?viewlocale=en_US&locale=en_US. [Online; Aksessert 15. mai 2018].

[12] Yamir Encarnacion. Raspberry Pi 3 as an Eddystone URL beacon. https://hackaday.io/project/10314-raspberry-pi-3-as-an-eddystone-url-beacon. [Online; Aksessert 11. mars 2018].

[13] Bluez — Official Linux Bluetooth protocol stack. http://www.bluez.org/. [Online; Aksessert 9. april 2018].

[14] GATT Overview. https://www.bluetooth.com/specifications/gatt/generic-attributes-overview. [Online; Aksessert 9. april 2018].

[15] example-gatt-server. https://github.com/RadiusNetworks/bluez/blob/master/test/example-gatt-server. [Online; Aksessert 4. april 2018].

[16] Running BLE GATT Server Example on Raspbian Stretch. https://scribles.net/running-ble-gatt-server-example-on-raspbian-stretch/. [Online; Aksessert 9. april 2018].

[17] A introduction to NuGet. https://docs.microsoft.com/en-us/nuget/what-is-nuget. [Online; Aksessert 2. mai 2018].

[18] Finding USB devices. http://usb4java.org/quickstart/libusb.html. [Online; Aksessert 16. mai 2018].

[19] Httprouter. https://github.com/julienschmidt/httprouter. [Online; Aksessert 7. mai 2018].

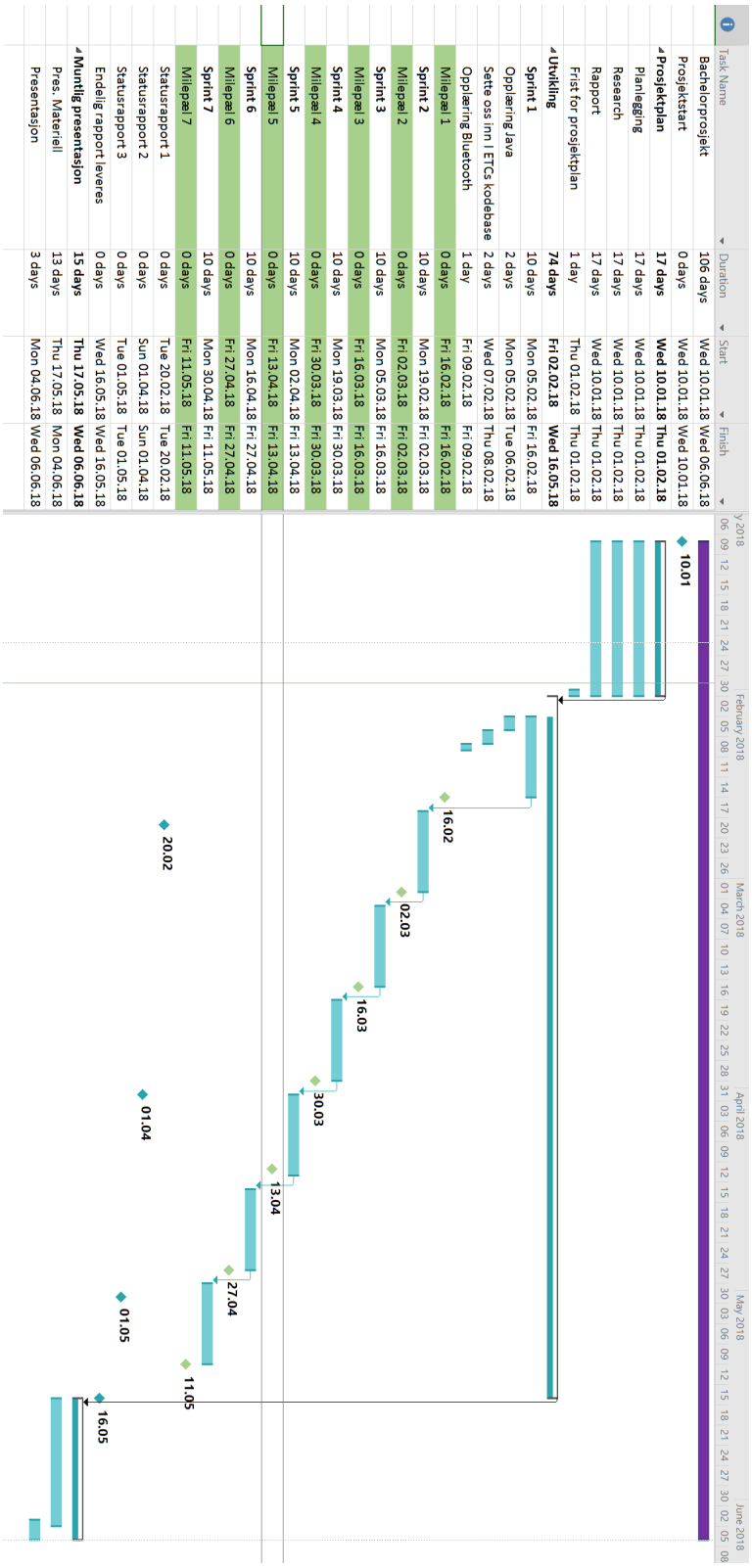
[20] Secure Introspectable Tunnels to Localhost. https://ngrok.com. [Online; Aksessert 26. april 2018].

[21] Postman | API Development Environment. https://www.getpostman.com/. [Online; Aksessert 26. april 2018].

[22] Oracle VM VirtualBox. https://www.virtualbox.org/. [Online; Aksessert 26. april 2018].

[23] Aage Rognsaa. 2016. *Bacheloroppgaven - Skriveråd og regler for utformingen*. Universitetsforlaget.

A Fremdriftsplan



Figur 17: Gantt-diagram

B Ordbok

Her gir vi en kort beskrivelse av en del av ordene fra rapporten som vi ser for oss at en leser ikke nødvendigvis vet betydningen av. Forhåpentligvis har vi dekket de fleste, og hvis leseren ikke finner igjen et ord som han stusser over, så håper vi at det kommer tydelig frem fra sammenhengen hva betydningen av det er.

Android: det mest brukte operativsystemet på smarttelefoner i dag. Samsung, Huawei, Sony og mange flere smarttelefonprodusenter benytter seg av Android.

API: et grensesnitt som man kan programmere mot.

ARM-prosessor: prosessortype som brukes på mobile enheter, f.eks. smarttelefoner og Raspberry Pi.

Avhengighet (eng: dependency): i programvaresammenheng, så betyr avhengighet at èn del av et program er avhengig av en anne del av et program.

Bash: et kommandolinjebasert grensesnitt for kommunikasjon med operativsystemet. Brukes på mange Linux-distribusjoner, bl.a. Raspbian.

Beacon: en BLE-enhet som sender ut advertisement-pakker med informasjon.

Bibliotek (eng: library): i programmeringssammenheng blir «bibliotek» brukt for å beskrive en pakke som gir muligheter for ekstrafunksjonalitet i et programmeringsspråk.

Bilpark: i denne sammenhengen betyr «bilpark» en samling av biler, gjerne eid av en bedrift eller lignende.

Bluetooth: trådløs kommunikasjonsteknologi som lar enheter utveksle data over korte distanser.

Bluetooth Low Energy (BLE): variant av Bluetooth som er mindre energikrevende enn ordinær Bluetooth. Brukes i enheter som kommuniserer mye, f.eks. hjertefrekvensmålere, pulsklokker, beacons osv.

Brannmur: mekanisme som kan stenge og tillate trafikk til og fra et nettverk.

Byggesystem (eng: build system): programvare som automatisk «bygger» strukturen i en programvare. Byggesystemet tar hånd om bl.a. avhengigheter, tester og mappestruktur. **CarAdmin**: programvare utviklet av ETC som holder orden på en bilpark.

Copyleft: et ordspill på *copyright*, som i grove trekk sier at hvem som helst fritt kan distribuere et arbeid, så lenge alle forgreninger av arbeidet også kan distribueres fritt.

Crontab: tidsbasert scheduler på Unix-baserte systemer.

Enhetstest (eng: unit test: en test bygget for å sørge for at en liten del av et program

fungerer som det skal.

ETC: softwareleverandør. Utvikler av CarAdmin. Vår oppdragsgiver i dette prosjektet.

Git: variant av versjonskontroll.

Heksadesimal: tallsystem med base 16. Forenkler i mange sammenhenger representasjon av data. Prefikses ofte med «0x».

HID (Human Interface Device): enheter som kan gjenkjennes uten spesifikke drivere i de fleste operativsystemer.

HTTP-route: i prosjektets sammenheng er en «route» en URL. Dersom en forespørsel ankommer denne URLen, så settes en reaksjon i gang.

HTTP-router: håndterer HTTP-forespørsler på HTTP-routes som er satt på HTTP-routeren.

IDE (Integrated Development Environment): programvaremiljø for utvikling av ny programvare.

Integrasjonstest (eng: integration test): flere separate enheter settes sammen og testes som om de var en enhet.

iOS: operativsystem som Apples iPhone bruker.

IoT (Internet of Things): begrep brukt om «hverdagsenheter» som er koblet til internett, og som kan sende og motta data.

Kernel: delen av operativsystemet som brukeren ikke har direkte tilgang til, og som gir muligheter for å kontrollere maskinvaren mer direkte.

Linux: open source-operativsystemer bygget rundt Linux-kjernen. *Raspbian*, det offisielle operativsystemet på Raspberry Pi, er en Linux-distribusjon.

Listener: en del av programvaren som konstant venter på («lytter») input.

Mikrocomputer: en liten og ofte rimelig datamaskin med innebygget minne, prosessor og et lite utvalg av I/O-enheter.

Mikrokontroller: en liten og enkel datamaskin som ofte konfigureres for å gjøre én spesifikk oppgave.

Mock: en modell/replika av et eksisterende system/produkt.

.NET: rammeverk som brukes på Microsoft-plattformen.

NFC (Near-Field Communication): trådløs kommunikasjonsteknologi som bl.a. benyttes i Apple og Samsungs betalingsløsninger via smarttelefon.

Open source: en programvare hvor kildekoden er åpen for alle. Open source-programvare benytter seg av forskjellige typer lisenser, som spesifiserer hvordan man kan benytte seg av den spesifikke lisensen.

Plugin: ekstrafunksjonalitet for en programvare. Kan lastes inn separat.

QR (Quick Response): en todimensjonal strekkode som kan skannes av et kamera og parses i en applikasjon.

RS485: elektrisk standard som ofte benyttes i serielle kommunikasjonssystemer. Også kjent som *TIA-485(-A)* og *EIA-485*.

Scrum: arbeidsmetodikk med et iterativt fokus, og tett dialog med oppdragsgiver.

Serial: dataoverføringsmekanisme som sender èn og èn bit sekvensielt.

Sprint: Iterasjonene i Scrum kalles for «sprinter», og er typisk fra 7 til 30 dager.

USB (Universal Serial Bus): industristandard som definerer kabler, kontakter og protokoller for tilkobling, dataoverføringer og strømtilførsel mellom enheter.

Use case: metode brukt for å beskrive interaksjonen mellom diverse aktører i en programvare.

Versjonskontroll: holder orden på versjoner og revisjoner av programvare.

C Prosjektplan

NTNU
Norwegian University of
Science and Technology

Prosjektplan

Forfattere
Markus Sørlien Johansen
Andreas Bradahl
Fredrik Hatletvedt

Bachelor i ingeniør - data
20 ECTS
Institute for Datateknikk og Informatikk
Norges teknisk-naturvitenskapelige universitet,

01.02.2018

Veileder Frode Haug

Innhold

Introduksjon til prosjektplan

Forord

Prosjektplanen fungerer som en tentativ plan for hvordan vi ser for oss å gjennomføre prosjektet. I tillegg til å være et verktøy for vår egen del, både for å strukturere arbeidet og få oversikt over oppgaven, så er det et dokument som i grove trekk viser veileder og oppdragsgiver hvordan vi ser for oss å løse problemet som beskrives. Det er som nevnt en tentativ plan, og det er svært sannsynlig at endringer vil forekomme, både i løsning av oppgaven og med tanke på tidsrammene som gis i Gantt-diagrammet i kapittel 6.

Informasjon om deltakere

Bachelorgruppen består av 3 tredjeårsstudenter ved Dataingeniør - NTNU i Gjøvik.

- Andreas Bradahl
- Fredrik Hatletvedt
- Markus Sørlien Johansen

Veileder: Frode Haug v/NTNU i Gjøvik
Oppdragsgiver: Electric Time Car AS

1 Mål og Rammer

1.1 Bakgrunn

Electric Time Car AS (ETC) er en softwareleverandør som holder til på Gjøvik. De leverer løsninger for deling av transportsmiddler og ressursstyring innen bilhold til bedrifter og offentlig sektor. Brukerne av systemet forholder seg til nøkkelskap hvor nøkler hentes og leveres før og etter bruk. CarAdmin[1], som er ETCs hovedprodukt, samler spredte bilparker i felles virtuelle parkeringsplasser som enkelt lar en bedriftsleder eller lignende ha enkel oversikt over bedriftens bilpark til enhver tid. I dag kommuniserer brukere med nøkkelskapene via en datamaskin som bruker lavnivåprotokoller som f.eks. RS485, TTL og USB. I disse IoT-tidene[2] er det likevel ønskelig å modernisere dette, og tilby et system hvor brukerne kan hente ut og levere nøkler ved hjelp av mobiltelefonen sin. Det er denne oppgaven, altså moderniseringen av den eksisterende løsningen, som vi har tatt på oss i dette prosjektet.

1.2 Prosjektmål

Prosjektmålene deles opp i *læringsmål*, *effektmål* og *resultatmål*. Læringsmålene forteller hva slags kunnskap vi ønsker å tilegne oss gjennom prosjektperioden. Effektmålene beskriver hvilke effekter og gevinster ETC ønsker å oppnå ved å innføre løsningen som er beskrevet over. Resultatmålene, på en annen side, sier noe om hvordan det konkrete produktet skal muliggjøre at effektmålene nås.

1.2.1 Læringsmål

I tillegg til læringsutbyttet beskrevet i BIDAT39s emnebeskrivelse, har vi definert følgende læringsmål:

- Bruk og programmering av mikrocomputere (mest sannsynlig Raspberry Pi 3).
- Tilegne oss kunnskap om utvikling for mobile plattformer (fortrinnsvis iOS og Android)
- Få erfaring med programmering mot Bluetooth-teknologi.
- Tilegne oss kunnskap om sikkerhet ift. «Internet of Things»-enheter.
- Få erfaring i å programmere mot eksisterende APIer og løsninger.
- Bruk og kombinasjon av systemutviklingsmetodikker (*Fossefall* og *Scrum*).

1.2.2 Effektmål

Med dette produktet ønsker ETC følgende:

- Reduserte kostnader ved at brukerne aksesserer nøkkelskapene fra egne enheter.
- Mer IoT-vennlig og tidsriktig løsning enn den som benyttes i dag.
- Enklere hverdag for CarAdmins brukere.

[1] www.caradmin.no
[2] Internet of Things

1.2.3 Resultatmål

Gruppa ønsker å levere et produkt som tilfredsstiller disse resultatmålene:

- Nøkkelskap kan åpnes ved bruk av mobiltelefon (ved vellykket brukerautentisering).
- Nøkkelskap skal kunne oppdateres via en ekstern kilde.
- Løsningen skal være intuitiv og enkel å bruke.
- Løsningen skal enkelt kunne utvides med andre kommunikasjonsteknologier (f.eks. NFC i tillegg til/istedenfor Bluetooth)

1.3 Rammer

1.3.1 Tidsrammer

- NTNU har satt en tidsramme fra 10. januar til 16. mai for ferdigstilling av prosjektet.
- ETC ønsker at grunnfunksjonaliteten i systemet skal være ferdig til midten av april, slik at ferdigstillingen av rapporten ikke skal gå utover arbeidet på det faktiske produktet.
- NTNU har satt krav om tre statusrapporter som leveres til veileder for å se prosjektets tilstand og fremdrift slik at det kan sammenlignes mot foreliggende planer.

1.3.2 Tekniske rammer

- Oppdragsgiver foretrekker at vi benytter Java som programmeringsspråk, men de er åpne for alternative forslag dersom det er grunner til det.

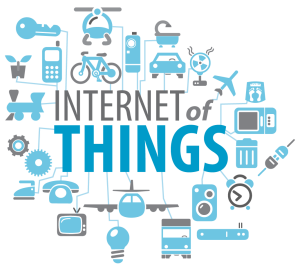
1.3.3 Økonomiske rammer

- NTNU Gjøvik bistår ikke med økonomiske utlegg som oppstår i prosjektperioden.

2 Omfang

2.1 Fagområde

Begrepet «Internet of Things» (IoT) blir mye brukt i dagens samfunn, og Wikipedias beskrivelse[1] av det lyder omtrent slik: *IoT er nettverket av fysiske enheter, transportmidler, hjemmeelektronikk og andre enheter med innebygget elektronikk, software, sensorer og nettverkskonnektivitet som tillater disse enhetene å kommunisere og utveksle data*. Ekspertene er uenige når det kommer til estimater av IoT-enheter, men det kan virke realistisk at det innen 2020 er ca. 30 milliarder tilkoblede enheter i IoT-nettverket[2].



Figur 1: Visualisering av «Internet of Things»

Grunnene til at dette blir et mer og mer relevant begrep er mange: bredbånd og høyhastighetsnettverk er i dag tilgjengelig i store deler av verden, og man betaler stadig mindre både for tilkobling og data, men også teknologi generelt. Om lag 80% av den norske befolkningen er i besittelse av en smarttelefon i dag[3], og bruksområdene for disse øker stadig. Ikke bare kan man sende videosnutter av barnebarnet til bestemor, eller spille sjakk mot en virtuell utgave av Magnus Carlsen på hytta, men i dag kan kaffetrakteren fjernstyres via hjemmenettverket, butikkbetalingen kan gjøres via NFC-teknologi[1], og varmeanlegget i bilen kan automatisk aktiveres når vekkerklokka (som forøvrig også er en telefon) ringer om morgenen. Vi ønsker oss en stadig enklere hverdag, og på mange måter er det dette IoT handler om.

[1] NFC: Teknologi som tillater datautveksling mellom to elektroniske enheter

2.2 Avgrensning

ETC ønsker å gjøre sin nåværende løsning mer IoT-vennlig. I dag har ETC utplassert en fysisk datamaskin ved hvert av sine nøkkelskap hvor brukeren som skal hente ut en bilnøkkel må logge seg på. Moderniseringen som ønskes vil tillate brukerne av CarAdmin å hente ut en bilnøkkel ved bruk av sin smarttelefon og ETCs egen app.

Som følge av at CarAdmin-systemet allerede er et etablert produkt, så har prosjektet vårt noen naturlige avgrensninger og begrensninger. Blant annet har ETC allerede en egen CarAdmin-brukerdatabase med tilhørende innloggings-/autentiseringsmekanismer. ETC har også utviklet en egen algoritme for utvelgelse av bil (dette skjer i sanntid når brukeren trenger bilen). De ønsker at denne algoritmen fortsatt benyttes, noe som betyr at vi ikke trenger å tenke på hvordan «riktig» bil velges. Det har vært diskutert om denne logikken kan/bør ligge på mikrocomputeren, men pga. tidsbegrensningen i prosjektet har vi gått bort fra denne muligheten.

Selve mobilapplikasjonen utvikles av ETC, og vi trenger derfor ikke å ha full oversikt over mobilutvikling, da vår løsning i praksis skal integreres i den eksisterende CarAdmin-appen.

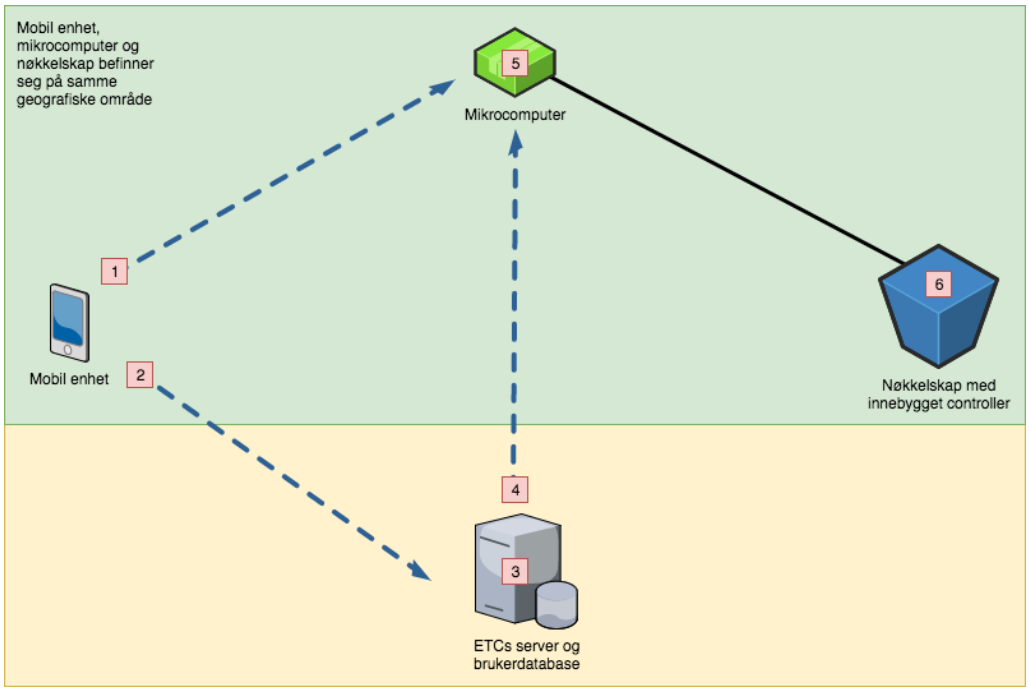
2.3 Oppgavebeskrivelse

Oppgaven som ETC har lansert har visse krav som *skal* være på plass i den ferdige løsningen. Så lenge vi holder oss innenfor disse, så er oppdragsgiver åpen for forslag både ift. nye løsninger, samt implementering av de allerede fastsatte kravene. Kravene som ETC har satt er som følger:

- ETCs uttaksrutiner/-algoritme ligger på deres server, og vår løsning må kommunisere med denne for åpning av riktig skap
- Skapet kan ikke åpnes av en bruker som ikke er i nærheten av det (dette må sikres på et vis).
- Programvaren i skapene oppdaterer «seg selv». Dette kan f.eks. skje ved at oppdateringer som legges på en remote server automatisk pushes til nøkkelskapene, eller ved at nøkkelskapene sjekker etter oppdateringer på faste tidspunkter.
- Mulighet for en administrator å fjernstyre mikrocomputeren via SSH.
- Programvaren i mikrocomputeren må kunne kommunisere med systemene ETC bruker i dag, deriblant server og brukerdatabase.
- Løsningen skal kunne tilpasses mot en iOS-/Android-app som allerede er under utvikling av ETC.
- ETC ønsker plug-and-play-hardware, som i praksis betyr at komponentene skal være klare til bruk (det er f.eks. uaktuelt å lodde på komponentene).
- Det ferdige produktet skal være enkelt å montere ute hos kunden.

Som nevnt er ETC åpne for forslag når det kommer til løsning og implementering, i tillegg til eventuell ny funksjonalitet. I oppgavebeskrivelsen nevnes Bluetooth som en mulighet for kommunikasjon mellom den mobile enheten og mikrocomputeren i nøkkelskapet, og det er i utgangspunktet denne teknologien vi vil utvikle med hensyn på.

Figur 2 viser hvordan grunnfunksjonaliteten til løsningen kommer til å se ut, og beskrives i grove trekk slik:

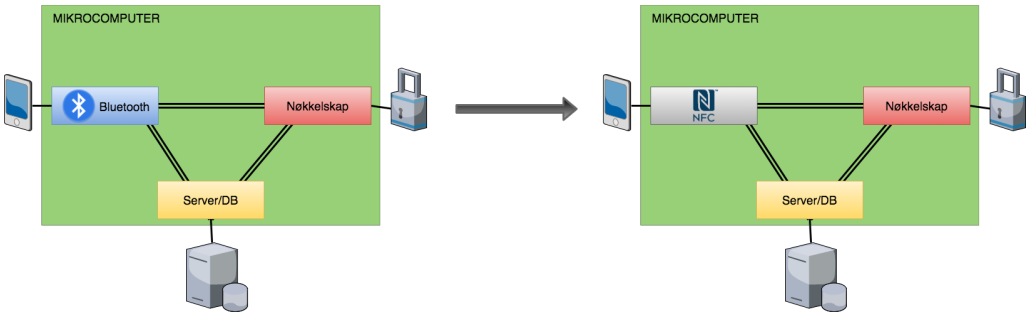


Figur 2: Kommunikasjon mellom komponentene i systemet

1. Brukeren logger inn på CarAdmin-appen og kobler seg til nøkkelskapet via Bluetooth.
2. CarAdmin-appen viser tilgjengelige transportmidler, og brukeren velger ønsket type. Input-dataene sendes til ETCs server.
3. ETCs server mottar input-dataene og evaluerer reservasjonen. ETCs unike uttaksalgoritme bestemmer så hvilket transportmiddel brukeren får, og dermed hvilken skuff som skal åpnes.
4. ETCs server sender data som forteller hvilken bil og skuff som skal åpnes til mikrocomputeren.
5. Mikrocomputeren sjekker via Bluetooth at brukeren faktisk *er* koblet opp mot skapet og utfører logikk for åpning av rett skuff.
6. Skuff låses opp i skapet og brukeren kan hente ut nøkkel.

Utover det som er beskrevet over vil vi også se på muligheter for å utvide løsningen ytterligere. Vi har bl.a. sett for oss at det kunne være av interesse å gjøre et dypdykk i sikkerhet ift. IoT-systemer. En annen mulighet er å utvide kommunikasjonsformen, slik at løsningen f.eks. kan benyttes med NFC, en teknologi som av mange anses som en bedre løsning enn Bluetooth ift. låseteknologi.

Vi ser for oss et modulbasert system, hvilket vil si at funksjonaliteten består av flere mer eller mindre «utskiftbare» delsystemer. I praksis betyr dette at et delsystem kan oppdateres eller erstattes, uten at de resterende delsystemene vil måtte skrives om. Figur 3 viser på en svært forenklet måte hvordan dette kan foregå: den blå Bluetooth-modulen erstattes med den grå NFC-modulen, *uten* at dette påvirker grensesnittene til hverken nøkkelskapet eller ETCs server.



Figur 3: Eksempel hvor Bluetooth-modulen erstattes med en NFC-modul

3 Prosjektorganisering

3.1 Ansvarsforhold og roller

I vår lille gruppe hvor alle mer eller mindre har det samme grunnlaget, så vil det i praksis bli slik at ethvert gruppemedlem (inkl. prosjektleder) er «allround-arbeidere», med det samme ansvaret for at prosjektet gjennomføres etter planen. Vi har likevel noen definerte roller i prosjektet. Som vi kommer til i del 4.1.1, så benytter vi oss av en Scrum-variant. Så godt det lar seg gjøre vil vi også benytte oss av Scrum-terminologien, både ift. roller og andre relevante oppgaver/hendelser. I visse tilfeller unngår vi derimot dette - vi har f.eks. ingen *Scrum Master*, som i Scrum er den som har ansvar for opplæring, guiding og opplæring i korrekt utførelse av Scrum.

Følgende roller er definert i prosjektet:

- **Prosjektleder:** Fredrik Hatletvedt
  Prosjektlederen har ansvar for å fordele oppgaver dersom det blir behov, samt passe på at frister blir holdt. Prosjektleder har også hovedansvaret for kommunikasjon mellom gruppa og oppdragsgiver/veileder.
- **Resterende gruppemedlemmer:** Andreas Bradahl og Markus Sørlien Johansen
  Fungerer i praksis som «allround-arbeidere» med det samme ansvaret som prosjektleder.
- **Veileder:** Frode Haug
  Veilederens jobb er å være til hjelp angående prosjektets gang, ved å komme med råd og pekepinner for at prosessen skal gå best mulig (hovedsakelig ift. selve rapporten).
- **Product Owner:** ETC v/Dag Solhaug
  Product Owner er en Scrum-rolle som i vårt tilfelle kan oversettes fritt til «oppdragsgiver». Han skal være til hjelp under utviklingen av løsningen ved å uttrykke funksjonelle ønsker/krav, samt gi konstruktive tilbakemeldinger underveis i produktutviklingen.

3.2 Rutiner og regler i gruppa

3.2.1 Generelle rutiner

1. Alle dokumenter som opprettes skal lagres i en felles Google Docs-mappe som alle gruppemedlemmer har tilgang til.
2. Møtereferater skrives og legges i Google Drive-mappen like i etterkant av hvert møte, og sendes så til Product Owner og/eller veileder.
3. Alle kilder som benyttes noteres fortløpende i et eget felles referansedokument, sammen med en beskrivelse av kilden.

3.2.2 Rutiner ift. utvikling

- Når kode legges til det offisielle BitBucket-repositoriet skal en engelsk commit-melding beskrive enkelt hva som ble lagt inn/endret.

- Master Branch skal kun benyttes til fungerende kode.
- Ved ny funksjonalitet, endringer på eksisterende kode og/eller bugfiksing, skal det opprettes en egen Branch for dette (det skal *ikke* jobbes direkte på Master Branch). Når funksjonalitet/endring er fullført opprettes Pull Request som resterende gruppemedlemmer kan se over.
- Alle variabelnavn og kommentarer i koden skal være på engelsk.
- «In progress»-tavla i Trello skal til enhver tid aldri ha mer enn 5 aktive oppgaver.

3.2.3 Regler

1. Det forventes 30 timers arbeidsuker pr. gruppemedlem.
2. Alle gruppemedlemmene plikter å gjennomføre de kursene som er planlagt, i tillegg til å sette seg inn i dokumentasjon som gruppen sammen bestemmer er relevant for prosjektet.
3. Endringer i prosjektdokumentasjon (f.eks. kode eller rapport) må tas i plenum.
4. Gi beskjed om endringer angående oppmøte og fravær i så god tid som mulig.
5. Møteplikt i alle typer møter, evt. gi beskjed i god tid slik at møter kan flyttes eller kanselleres.
6. Ved faglige uenigheter skal gruppen forsøke å komme til enighet. Evt. fattes en flertallsavgjørelse som det kreves lojalitet til.
7. Det avholdes morgenmøter hver arbeidsdag i Sprinten. Disse har en tidsvarighet på 5-10 minutter, status siden forrige arbeidsdag samt plan for dagen beskrives.
8. Basetid er fra kl. 08:00 - 16:00 fire dager pr. uke. Dette innebærer at det ikke er forventninger/plikt til arbeid utenfor dette tidsrommet med mindre det er enighet om dette.
9. Det skal tilstrebes å jobbe sammen på skolen minst tre dager pr. uke.
10. Hvert gruppemedlem skal føre egen timelogg i et felles Google Docs-dokument for hver arbeidsdag for å holde oversikt over hva som ble gjort og antall timer vedkommende har jobbet.
11. Mandag morgen lages en Sprint Backlog for inneværende Sprint. Denne planen brukes som utgangspunkt for dagsplanene som legges på morgenmøtene resten av uka.
12. Et medlem vil ikke kunne bli kastet ut av gruppen mot prosjektslutt (21 dager før ferdigstilling av prosjektet).

3.2.4 Konsekvenser

Ved gjentatte brudd på gruppereglene vil det kalles inn til gruppemøte der alle gruppemedlemmer må være tilstede. Det vil deretter diskuteres alvorlighetsgraden og hvordan problemet skal løses.

- Hvis møtet ikke løser problemet vil det bli sendt en skriftlig advarsel til vedkommende angående hvilke regelbrudd som gjelder, alvorligheten av regelbruddet og konsekvensen dette har på videre arbeid.
- Hvis problemet fortsatt ikke er løst vil et møte med gruppemedlemmene og veileder holdes, der mulig ekskludering av gruppen vil bli diskutert.
- Hvis gruppen ender opp med å splittes, skal alt arbeid som vedkommende har sittet på bli levert til resten av gruppen.

4 Planlegging, oppfølging og Rapportering

4.1 Hovedinndelingen av prosjektet

4.1.1 Valg av systemutviklingsmodell

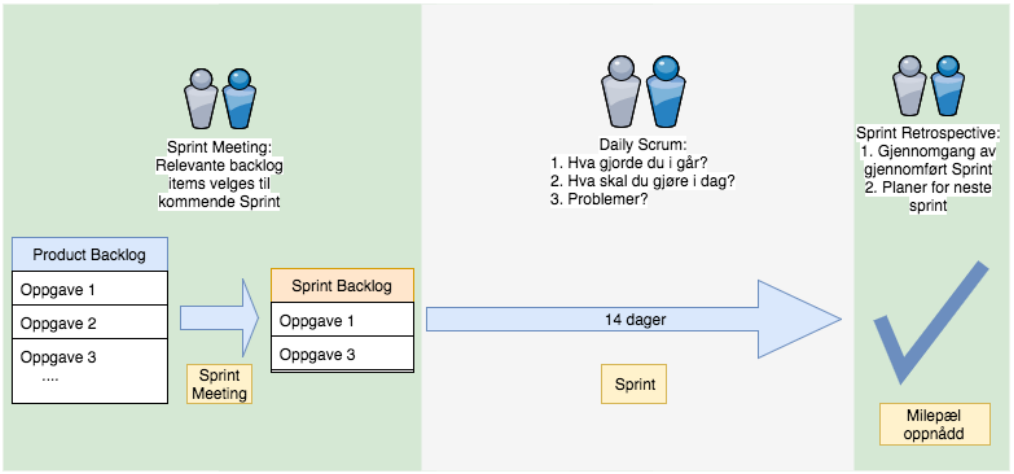
Som nevnt i 2.3 er den essensielle funksjonaliteten i programvaren avklart og forstått av både oppdragsgiver og prosjektdeltakerne, mens det fremdeles er åpenhet rundt andre funksjonaliteter og muligheter.

Grunnfunksjonaliteten i prosjektet vil hovedsakelig bli utviklet ved bruk av en plandrevet prosessmodell (fossefallsmodellen), men dog ikke helt etter boka. I praksis betyr dette at vi i grove trekk kommer til å gjennomføre èn og èn fase, med mer eller mindre klare skillelinjer. Da vi alle er uerfarne i forhold til både utvikling og metodikk, så anser vi det likevel som sannsynlig at det oppstår situasjoner hvor vi har behov for å gå tilbake og endre på tidligere avgjørelser. Dette gjør at vi tillater oss å bruke en «rundere» variant av fossefall, med innslag av smidig utvikling.

Vi har alle noe teoretisk erfaring med Scrum fra tidligere, og det faller seg derfor naturlig å benytte oss av denne modellen. Vi har valgt å gå for Sprinter med 14 dagers intervaller. Sprintene starter på mandag, og er ferdige etter 10 arbeidsdager (fredag kveld). Dette gir oss tid til å arbeide i vårt eget tempo over en lengre periode, men samtidig få hyppige tilbakemeldinger fra Product Owner (ETC), noe vi synes er greit mtp. uerfarenheten vår.

Hver Sprint startes mandag morgen med et *Sprint Meeting*, hvor *Sprint Backlogen* for den kommende Sprinten velges ut basert på vår *Product Backlog*. Ved den første tirsdagen etter hver Sprint gjennomføres et *Sprint Retrospective Meeting* hvor resultater, situasjoner og evt. problemer fra den gjennomførte Sprinten gjennomgåes. Vi kommer til å benytte oss av *Daily Scrums*: morgenmøter på ca. 10 minutter hvor planen for dagen og evt. utfordringer som har dukket opp gjennomgåes. Arbeidsoppgavene ift. grunnfunksjonaliteten deles også opp i Backlog Items og legges til Product Backlogen. På denne måten har vi en oversikt over de enkelte delsystemene, og kan bruke disse som et utgangspunkt for hver Sprint. Ian Sommerville nevner denne typen hybridmodell som et mulig valg i prosjekter hvor deler av systemet er godt kjent[4].

Figur 4 visualiserer hvordan vi ser for oss at Scrum-prosessen gjennomføres. Fase 1 innebærer produksjon av Sprint Backlog, i fase 2 gjennomføres selve Sprinten, og i fase 3 har vi forhåpentligvis et resultat å vise frem fra den nettopp gjennomførte Sprinten.



Figur 4: Slik tenker vi oss Scrum-prosessen

4.2 Plan for statusmøter og beslutningspunkter i perioden

Det er planlagt å ha møter med veileder en gang i uken i oppstarten, slik at vi får gjort unna prosjektplanen og får mest mulig informasjon angående hvordan denne kan forbedres. Dette vil være med på å gjøre prosessen videre enklere ved å ha en god og strukturert plan.

Det er planlagt å ha møter med oppdragsgiver en gang i uken når utviklingen av prosjektet starter for fullt. Det vil med andre ord bli ett møte før/etter hver Sprint, og ett møte underveis i hver Sprint.

5 Organisering av Kvalitetssikring

5.1 Dokumentasjon, standardbruk og kildekode

5.1.1 Dokumentasjon

Mesteparten av dette prosjektet dokumenteres gjennom selve bachelorrapporten. Dette gjelder hovedsakelig teori rundt oppgaven (f.eks. ift. teknologivalg), metodebruk og relevant bakgrunnsinformasjon.

5.1.2 Kildekode og standardbruk

Kildekode og koderelatert dokumentasjon finnes i vårt Bitbucket-repository [5]. Dersom ETC har egne standarder for kildekode og kommentering vil vi benytte oss av disse. Vi kommer også til å se på nytten av en *Linter*[1] for å ha en felles standard på kildekoden vi ender opp med å skrive.

5.2 Verktøy

- **Draw.io**
  Draw.io er et enkelt nettbasert verktøy for å lage diagrammer, UML-modeller og andre figurer i rapporten.
- **Google Drive**
  Google Drive er en nettbasert skyløsning som vi benytter for å dele diverse dokumenter.
- **Microsoft Project**
  Tilbyr å opprette Gantt-diagrammer på en enkel og oversiktlig måte.
- **Nextcloud**
  Nextcloud er en skybasert fillagrings- og delingsklient som vi bruker for deling av lokalt lagrede ressurser for prosjektet.
- **ShareLaTeX**
  Vi benytter ShareLaTeX for dokumentering av oppgaven, da dette gir oss mulighet for å benytte standard LaTeX-maler laget for bachelorrapporten. NTNU i Gjøvik har allerede slike maler i et Git-repository[6] som alle har mulighet for å benytte seg av. I tillegg gir ShareLaTeX oss mulighet for samskriving på en enkel måte.
- **Trello**
  Vi bruker Trello for prosjektplanlegging, for enkel oversikt over Product Backlog/Sprint Backlog og oppgavefordeling.

5.3 Versjonskontroll

Vi har valgt å benytte oss av Bitbucket for å håndtere versjonskontroll. Grunnen til dette er at NTNU i Gjøvik tilbyr studentene sine *Academic Plan* helt gratis, noe som tillater private repositories. Dette er relevant, da ETC ønsker at deler av deres egen kode holdes privat. Bitbucket er også brukt av samtlige prosjektdeltakerne tidligere, og det er derfor et system vi kjenner til.

[1] Lint: Verktøy som automatisk analyserer kildekode ift. feil og formatforbedringer.

5.4 Risikoanalyse

Ved store prosjekter er det alltids en sjanse for at noe kan gå galt. Derfor er det viktig å ha en god oversikt over eventuelle risikoer som kan dukke opp underveis i prosjektet. Vi har derfor satt opp noen risikoer og gradert de med tanke på sannsynlighet og konsekvensen disse vil ha.

5.4.1 Gradering av risiko

Graderingsskjemaet (figur 1) viser sammenhengen mellom konsekvens og sannsynligheten som er med på å bestemme graden av risiko.

Konsekvens/ Sannsynlighet	**Liten**	**Middels**	**Høy**	**Kritisk**
Lite sannsynlig		7, 8	6	3
Noe Sannsynlig			2, 4	
Sannsynlig		5	1	
Meget sannsynlig				

Tabell 1: Tabell for for gradering av risiko

5.4.2 Identifisere risikoer

De mulige risikoene er fargekodet etter hvordan graderingen er satt opp, og fargekodene i tabellen nedenfor baserer seg på denne. Vi har valgt å ta for oss risikoene som i hovedsak har en høy risiko, men har også tatt med noen med lavere risiko som vi tenkte det var verdt å tenke på.

Indeks	**Beskrivelse**	**Sannsynlighet**	**Konsekvens**	**Tiltak**
1	Ikke fullført arbeid i slutten av en sprint.	Sannsynlig	Høy	Forutse arbeidsmengde på forhånd. Tidlig finne ut om oppgavene lar seg gjøre.
2	Løsningen blir ikke fullført i tide.	Noe sannsynlig	Høy	Alltid jobbe med oppgaven og følge planen.
3	Rapporten blir ikke fullført i tide.	Lite sannsynlig	Kritisk	Holde rapporten oppdatert, alltid skrive underveis og følge/revidere planen.
4	En eller flere gruppemedlemmer hindres i å delta over en lengre periode.	Noe sannsynlig	Høy	Diskutere fravær på forhånd og oppdatere medlemmet på fremgangen ved neste oppmøte.
5	Mangel på kompetanse.	Sannsynlig	Middels	Kursing/konsultere fagpersoner.
6	Tap av kode.	Lite sannsynlig	Høy	Ta jevnlig backup og ikke utvikle for mye kode på en gang.
7	Blir nødt til å skrive om deler av dokumentasjon og/eller koden.	Lite sannsynlig	Middels	Jevnlige møter med oppdragsgiver.
8	ETC avbryter prosjektet.	Lite sannsynlig	Middels	Opprettholde avtaler og ha god kommunikasjon med oppdragsgiver. Omstrukturere prosjektet om nødvendig/mulig.

Tabell 2: Risikotabell

6 Plan for gjennomføring

6.1 Gantt-skjema

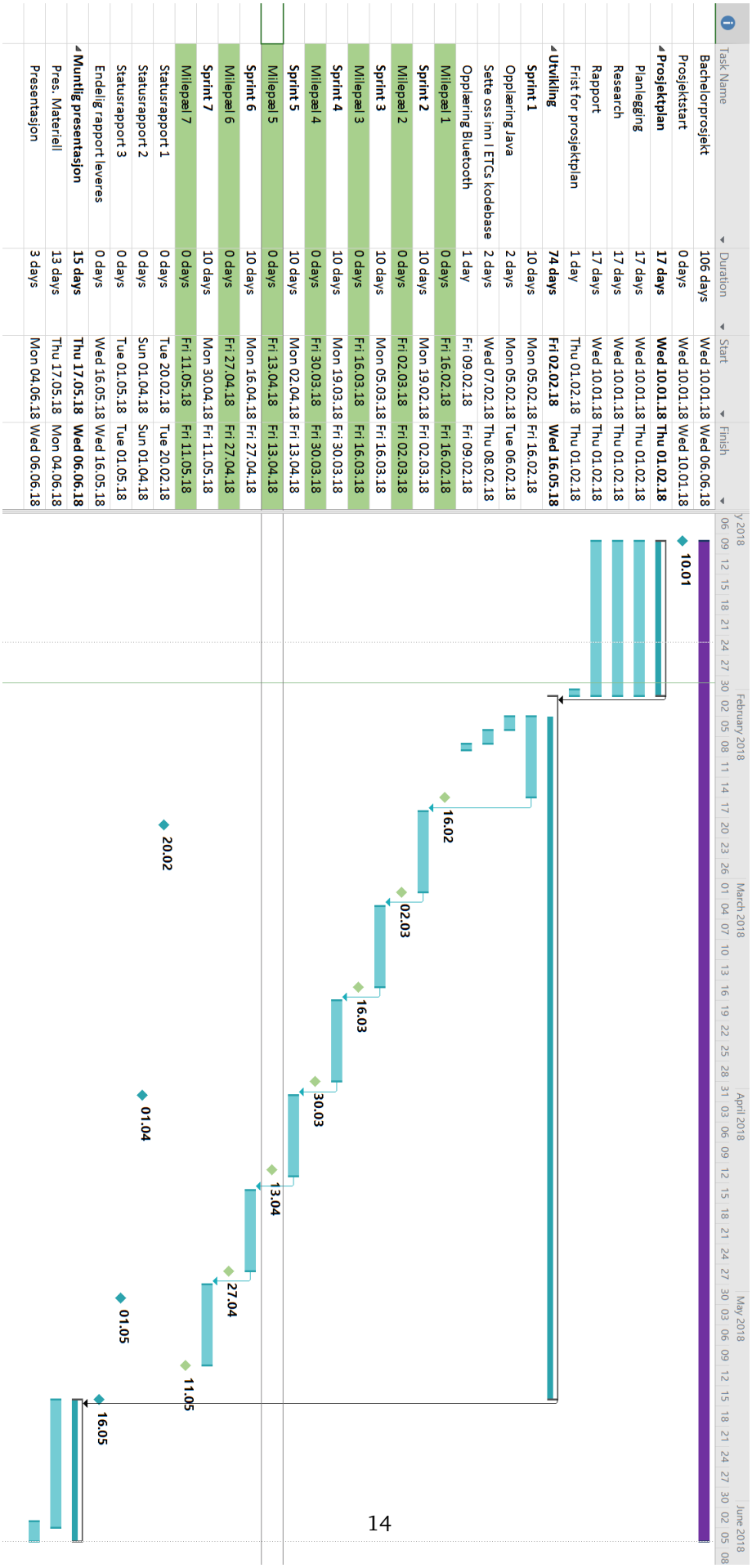
Figur 5 viser vår nåværende plan for gjennomføringen av prosjektet. Det er sannsynlig at denne endres i løpet av prosjektperioden.

6.2 Kommentar til tidsplan

Etter hver Sprint skal en milepæl være gjennomført. Milepælene fungerer som foreløpige «statusrapporter» og sier noe om hvordan fremdriften er ift. den faktiske planen.

Milepælene er som følger:

1. 16.02.18: Fullført opplæring (Java, Bluetooth, ETCs kodebase) og oppnådd grunnlegende kommunikasjon med skapkontroller.
2. 02.03.18: Testet funksjonalitet med skap og implementert Bluetooth-kommunikasjon mellom mobil enhet og mikrocomputer.
3. 16.03.18: Integrert kommunikasjon mellom skap og server.
4. 30.03.18: Optimalisert kode og kommunikasjon med tanke på sikkerhet.
5. 13.04.18: Hovedfunksjonalitet for systemet skal være på plass.
6. 27.04.18: Ekstrafunksjonalitet skal være planlagt og implementert.
7. 11.05.18: Ekstrafunksjonalitet skal være ferdig dokumentert.



Task Name	Duration	Start	Finish
Bachelorprosjekt	**106 days**	**Wed 10.01.18**	**Wed 06.06.18**
Prosjektstart	0 days	Wed 10.01.18	Wed 10.01.18
Prosjektplan	**17 days**	**Wed 10.01.18**	**Thu 01.02.18**
Planlegging	17 days	Wed 10.01.18	Thu 01.02.18
Research	17 days	Wed 10.01.18	Thu 01.02.18
Rapport	17 days	Wed 10.01.18	Thu 01.02.18
Frist for prosjektplan	1 day	Thu 01.02.18	Thu 01.02.18
Utvikling	**74 days**	**Fri 02.02.18**	**Wed 16.05.18**
Sprint 1	10 days	Mon 05.02.18	Fri 16.02.18
Opplæring Java	2 days	Mon 05.02.18	Tue 06.02.18
Sette oss inn I ETCs kodebase	2 days	Wed 07.02.18	Thu 08.02.18
Opplæring Bluetooth	1 day	Fri 09.02.18	Fri 09.02.18
Milepæl 1	0 days	Fri 16.02.18	Fri 16.02.18
Sprint 2	10 days	Mon 19.02.18	Fri 02.03.18
Milepæl 2	0 days	Fri 02.03.18	Fri 02.03.18
Sprint 3	10 days	Mon 05.03.18	Fri 16.03.18
Milepæl 3	0 days	Fri 16.03.18	Fri 16.03.18
Sprint 4	10 days	Mon 19.03.18	Fri 30.03.18
Milepæl 4	0 days	Fri 30.03.18	Fri 30.03.18
Sprint 5	10 days	Mon 02.04.18	Fri 13.04.18
Milepæl 5	0 days	Fri 13.04.18	Fri 13.04.18
Sprint 6	10 days	Mon 16.04.18	Fri 27.04.18
Milepæl 6	0 days	Fri 27.04.18	Fri 27.04.18
Sprint 7	10 days	Mon 30.04.18	Fri 11.05.18
Milepæl 7	0 days	Fri 11.05.18	Fri 11.05.18
Statusrapport 1	0 days	Tue 20.02.18	Tue 20.02.18
Statusrapport 2	0 days	Sun 01.04.18	Sun 01.04.18
Statusrapport 3	0 days	Tue 01.05.18	Tue 01.05.18
Endelig rapport leveres	0 days	Wed 16.05.18	Wed 16.05.18
Muntlig presentasjon	**15 days**	**Thu 17.05.18**	**Wed 06.06.18**
Pres. Materiell	13 days	Thu 17.05.18	Mon 04.06.18
Presentasjon	3 days	Mon 04.06.18	Wed 06.06.18

Figur 5: Gantt-diagram

Bibliografi

[1] Wikipedia - Internet of Things.
https://en.wikipedia.org/wiki/Internet_of_things.
[Online; Accessed 1. februar 2018].

[2] Popular Internet of Things Forecast of 50 Billion Devices by 2020 is Outdated.
https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated.
[Online; Accessed 1. februar 2018].

[3] Andel som har smarttelefon.
http://www.medienorge.uib.no/statistikk/medium/ikt/379.
[Online; Accessed 1. februar 2018].

[4] Sommerville, I. 2016. *Software Engineering - Tenth Edition*. Pearson.

[5] 2018. Bitbucket.
https://bitbucket.org/MarkusSJ/bacheloroppgave_etc.
[Online; accessed 11-January-2018].

[6] bachelor-thesis-NTNU.
https://github.com/COPCSE-NTNU/bachelor-thesis-NTNU.
[Online; Accessed 11-January-2018].

D Prosjektavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

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

ELECTRIC TIME CAR AS

(oppdragsgiver), og

MARKUS S. JOHANSEN, FREDRIK HATLETVEDT OG ANDREAS BRADAHL

(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 10.01.18 til 16.05.18.

   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 L SOLHAUG

Student(er) (signatur): Markus S. Johansen dato 19/1-2018

Fredrik Hatlebrekke dato 19.01.2018

Andreas Bradahl dato 19.01.2018

_____ dato _____

Oppdragsgiver (signatur): Dag L Solhaug dato 18/1-2018

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 _____

E Statusrapporter

Statusrapport 1:

1. Status for:
   - Planlegging (jfr. fremdriftsplan)
     Planen er uendret ift. fremdriftsplanen i Prosjektplandokumentet. Det har med andre ord ikke dukket opp noe som gjør at planen må endres.

   - Organisering av gruppens arbeid og ansvarsområder
     Gruppens arbeid er organisert i Trello. Nå i starten har det meste av arbeidet handlet om research, så organiseringen av arbeidet har ikke vært like vesentlig som vi ser for oss at det vil bli på et senere tidspunkt.

   - Klargjøring av problemstillingen
     Problemstillingen har vært relativt klar for oss i en stund nå, og vi har ikke endret på denne fra prosjektplanen.

   - Løsningsmetode
     Vi har for det meste jobbet felles nå i starten for å komme i gang med diverse systemer.

   - Rapportskriving
     Vi har ikke startet på rapporten, men har laget mal for dette.

3. Muligheter. Trusler/problemer.
Det har dukket opp noen tekniske «utfordringer», bl.a. i forhold til Java og virtualisering, og vi har brukt noe mer tid på dette enn hva vi hadde sett for oss.

4. Hva er avsluttet. Hvilke oppgaver er ferdige.
Har laget et kodegrunnlag i Java som tillater kommunikasjon med USB-enheter. Dette kan benyttes i alle OS som støtter Java. Hovedmålet i første sprint var å oppnå grunnleggende kommunikasjon med mikrocontroller, og dette er også noe vi har oppnådd.

5. Hva er under arbeid.

Omskriving av oppdragsgivers kode fra C# til Java. Koden som den er, skal benyttes på Raspberry Pi og fungerer nå bare for Windows, denne må derfor skrives om til å fungere i Linux (Raspbian). Vi driver også med opplæring i Java og Gradle (Intellij IDEA).

6. Er tidsfristene
   - Overholdt
   - Overskredet
   - Kritiske(“Brenner det et blått lys”)

Tidsfristene er overholdt, selv om det har dukket opp uforutsette utfordringer underveis i prosessen.

7. Hva med motivasjonen:
   - Gruppens samarbeid, arbeidsformer og organisering.
   - Forhold som oppmuntrer eller frustrerer.

Gruppen samarbeider bra, og vi har alltid noe å gjøre. Vi arbeider hovedsakelig sammen på skolen, da mye per nå handler om å skaffe oss et felles grunnlag ift. oversikt over prosjektet. I forhold til planen vi hadde laget på forhånd, så har arbeidet under denne sprinten blitt noe mer ustrukturert enn vi hadde tenkt. Det har dukket opp tekniske utfordringer underveis, noe som vi naturligvis forutså at skulle skje, da dette er et helt nytt kunnskapsområde for alle på gruppen.

8. Hvordan oppleves veilederkontakt:
   - Fungerer den
   - Fornøyd/misfornøyd

Veilederkontakten fungerer som forventet, og vi er hittil fornøyde.

Statusrapport 2:

1. Status for:
   - Planlegging (jfr. fremdriftsplan)
     Planen er uendret ift. fremdriftsplanen i Prosjektplandokumentet. Det har med andre ord ikke dukket opp noe som gjør at planen må endres.

   - Organisering av gruppens arbeid og ansvarsområder
     Gruppens arbeid er organisert i Trello. Utover dette blir det gjort en del muntlige diskusjoner, samtaler for fordeling av oppgaver.

   - Klargjøring av problemstillingen
     Problemstillingen har vært relativt klar for oss i en stund nå, og vi har ikke endret på denne fra prosjektplanen.

   - Løsningsmetode
     Vi har for det meste jobbet felles på skolen som gruppe, og delt opp oppgavene slik at alle har noe å jobbe med. Før vi jobber videre på nye deler tar vi planlegging i plenum med diagrammer og beskrivelser for at alle skal henge med.

   - Rapportskriving
     Vi har kommet et stykke på rapporten. Vi har skrevet en god del på kravspesifikasjon, implementering av det vi har gjort og analyse av forskjellige teknologier valgt underveis. I tillegg har vi lagt til noen av vedleggene: Møtereferater, prosjektplan, osv.

3. Muligheter. Trusler/problemer.
Det har dukket opp noen problemer angående kommunikasjonen mellom systemene som må endres på.

4. Hva er avsluttet. Hvilke oppgaver er ferdige.
Kommunikasjon gjennom alle komponentene i systemet, fra mobil til det faktiske nøkkelskapet. Grunnleggende serverlogikk for mocking av ETC sin server. Ettersom det har blitt gjort endringer i flere ledd i systemet har vi måttet jobbe videre med å ferdigstille disse.

5. Hva er under arbeid.

Rapport med med tanke på Design og Arkitektur og implementering av den endelige løsningen. Ny revisjon av Bluetooth-kommunikasjon.

6. Er tidsfristene
   - Overholdt
   - Overskredet
   - Kritiske(“Brenner det et blått lys”)

Tidsfristene er overholdt, selv om det har dukket opp uforutsette utfordringer underveis i prosessen.

7. Hva med motivasjonen:
   - Gruppens samarbeid, arbeidsformer og organisering.
   - Forhold som oppmuntrer eller frustrerer.

Gruppen samarbeider bra, og vi har alltid noe å gjøre. Vi arbeider for det meste på skolen, der det kommer opp diskusjoner og samtaler om det vi driver med underveis. Vi har måttet endre litt på visse deler av løsningen underveis, ettersom noen av de delene vi har laget ikke var ønsket av ETC. Vi har også måttet omstrukturere på deler av kommunikasjonen, noe som tok tid.

8. Hvordan oppleves veilederkontakt:
   - Fungerer den
   - Fornøyd/misfornøyd

Veilederkontakten fungerer som forventet, og vi er hittil fornøyde.

Statusrapport 3:

1. Status for:
   - Planlegging (jfr. fremdriftsplan)
     I følge planen skulle vi nå jobbe med ekstrafunksjonalitet. Med tanke på at noe av ekstrafunksjonaliteten (mulighet for at systemet fungerer selv om Raspberry Pi ikke har internett) ble implementert inn i hovedfunksjonaliteten på et tidligere tidspunkt, så har dette tatt noe lenger tid enn først antatt. Perioden frem til nå har derfor gått med til finpuss av systemet. Tiden nå går derfor hovedsakelig til rapportskriving.

   - Organisering av gruppens arbeid og ansvarsområder
     Gruppens arbeid er organisert i Trello. Utover dette blir det gjort en del muntlige diskusjoner, samtaler for fordeling av oppgaver.

   - Klargjøring av problemstillingen
     Problemstillingen har vært relativt klar for oss i en stund nå, og vi har ikke endret på denne fra prosjektplanen.

   - Løsningsmetode
     Vi har for det meste jobbet felles på skolen som gruppe, og delt opp oppgavene slik at alle har noe å jobbe med. Før vi jobber videre på nye deler tar vi planlegging i plenum med diagrammer og beskrivelser for at alle skal henge med.

   - Rapportskriving
     Vi har for øyeblikket fullt fokus på rapport, og har kommet godt på vei med den.

3. Muligheter. Trusler/problemer.
Det har dukket opp noen uforutsette hindringer mens vi har implementert applikasjonen i oppdragsgivers applikasjon. Det tok noe lenger tid å implementere dette ettersom vi fortsatt ikke har altfor gode kunnskaper i å bruke Xamarin, og det var også noen problemer i forhold til serveroppkobling fra applikasjonen.

4. Hva er avsluttet. Hvilke oppgaver er ferdige.

Det meste av koding er avsluttet. Det er fortsatt noe småfiksing og kommentering som må på plass, men dette er ikke hovedfokus nå. Implementering i oppdragsgivers applikasjon er nå ferdig, ved at den gjør jobben den skal gjøre. Systemet fungerer fra start til slutt, noe som er testet via oppdragsgivers serversystem.

5. Hva er under arbeid.

Rapporten er nå under arbeid og vi har kommet godt på vei. Det mangler fortsatt å starte på avslutningskapittelet og legge til kode og forklare denne.

6. Er tidsfristene
   - Overholdt
   - Overskredet
   - Kritiske(“Brenner det et blått lys”)

Tidsfristene er overholdt, selv om det har dukket opp uforutsette utfordringer underveis i prosessen.

7. Hva med motivasjonen:
   - Gruppens samarbeid, arbeidsformer og organisering.
   - Forhold som oppmuntrer eller frustrerer.

Gruppen samarbeider bra, og vi har alltid noe å gjøre. Vi arbeider fortsatt for det meste på skolen, der det kommer opp diskusjoner og samtaler om det vi driver med underveis. Noe problemer i forhold til kommunikasjon med server, noe som har resultert i å bruke mer tid enn forventet og sending av mail frem og tilbake for å klarne opp i dette.

8. Hvordan oppleves veilederkontakt:
   - Fungerer den
   - Fornøyd/misfornøyd

Veilederkontakten fungerer som forventet, og vi er hittil fornøyde.

F Oppgavetext

Oppdragsgiver

Oppdragsgiver: ETC, Electric Time Car AS
Kontaktperson: Dag L Solhaug
Adresse: Studieveien 2, 2815 Gjøvik
Telefon: +47 901 01 344
Epost: dag.solhaug@electrictimecar.com

IoT - Fjernstyre nøkkelskap

ETC er en lokal Software leverandør som bla leverer løsninger for bilpool og ressursstyring innen bilhold, EL-sykler ol til bedrifter og offentlig sektor. Våre kunder etterspør i større og større grad muligheten til å bruke mobilene til å kunne ta ut og levere kjøretøy i våre nøkkelskap.

For å få til dette trenger vi å kunne fjernstyre våre nøkkelskap. Her ser vi for oss å benytte hyllevare av type microcomputere som kan kobles inn i nøkkelskapene, med mulighet for å kunne koble til nettverk over WIFI og LAN, samt blåtann.

Oppgaven

Gjøre det mulig å fjernstyre nøkkelskap, fra server/mobil, som i dag hovedsakelig kommuniserer på lavnivå protokoller RS485, TTL og USB.
Finne og tilpasse hardvare til formålet.
Programmere hardvare for kommunikasjon med nøkkelskapene, vår server og telefon via blåtann.

Programmering
ETC vil bistå prosjektet i gjennomføringen av oppgaven. Jevnlige møter og oppgavefordelinger avtales underveis.

Studentgruppen vil gjennom prosjektet få erfaring med:

- Microcomputere
- Programmering
- Serielle protokoller
- Blåtann
- Sikkerhet i IoT
- Integrasjon med eksisterende løsning API

Oppgaven passer best for en gruppe på 1 - 3 personer.

ETC har gjennomført bacheloroppgave ved HiG/NTNU siden 2004 hvor flere av oppgaven har fått topp karakter. Alle bacheloroppgaver har resultert i ansettelser i ETC, eller ansettelser for en eller flere av studentene som følge av referanse fra ETC etter endt bacheloroppgave.

Vi er for tiden i vekst og har planer om å utvide staben av utviklere i 2017 her på Gjøvik, lokal tilhørighet vektlegges.

G Møtereferater

G.1 Møtereferater - Oppdragsgiver

18.01.2018 - Oppstartsmøte

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC)

Agenda:

- Klargjøring av oppgave (Use-cases)
- Kartlegging av mål
- Relevant teknologi (programmeringsspråk, maskinvare, annet)
- Mobile plattformer (iOS/Android)
- CarAdmins bruksmønster
- Nøkkelskapets grensesnitt
- Risikovurdering
- Oppfølging mellom møter og møtehyppighet

Kommentarer:

Klargjort en del ift. selve oppgaven, bl.a. flyten i systemet og hva som eksisterer av systemer og løsninger pr i dag. Vi ble enige om at fase 1 hovedsakelig bør dreie seg rundt utvikling mellom mikrocomputer og mikrokontroller i nøkkelskap. Flyten mellom komponenter beskrives mer detaljert i prosjektplanen som skal være ferdig innen 1. februar. Foreløpige teknologiske valg blir også beskrevet grundigere i prosjektplanen. Målene med oppgaven er relativt klare.

30.01.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC)

Agenda:

- Mer klarhet ift. teknisk løsning
- Produksjon av Product Backlog
- Standarder som eksisterer i ETC
    f.eks. i forhold til kildekode, kommentering, verktøyer osv.

Kommentarer:

Fått demonstrert uttaksprosedyre, og fått klarhet i hvorfor vår skisserte løsning ikke fungerer med deres system. Vi skal se på uttaks- og valglogikken som en «svart boks» på ETCs server, og heller fokusere på input/output til denne. Hovedfokus blir på grunnløsning frem til midten av april, og evt. utvidelser (f.eks. dypdykk ift. sikkerhet og/eller NFC) etter dette. Ble enige om at vi (gruppa) produserer en Product Backlog, og at oppdragsgiver heller ser over og kommer med innspill. Kode og evt. kodestandarder tas opp på et møte når utvikler er til stede. Fast møtedag tirsdager klokken 09:00 fremover (invitasjoner sendes via Outlook).

06.02.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC) Øyvind Flatval (utvikler, ETC)

Agenda:

- Kontrollerkode
- Tilgang til generell kodebase
- Standarder og verktøy

Kommentarer:

Får tilsendt kontrollerkode i løpet av de nærmeste dager. Enige om at det kan være greit å benytte seg av en mocked mobilapplikasjon før vi begynner å kode mot den virkelige applikasjonen. Java benyttes så fremt det er mulig.

20.02.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC) Øyvind Flatval (utvikler, ETC)

Agenda:

- Status
- Teste kode mot skap

Kommentarer:

Fikk info. om hvordan svarene fra mikrokontroller skal se ut. Enige om at vi per i dag er i rute iht. prosjektplan. Har nå begynt å se på Bluetooth-teknologi ift. initiell kommunikasjon mellom smarttelefon og Raspberry Pi.

27.02.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Øyvind Flatval (utvikler, ETC)

Agenda:

- Teste kode for nøkkelskap mot fysisk nøkkelskap
- Generell status

Kommentarer:

Nøkkelskapkoden fungerte, og vi greide å låse opp hver enkelt hylle fra mikrocomputeren. Ved en anledning trengte den flere forsøk på å fungere, og dette må vi finne ut av.

06.03.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC)

Agenda:

- Status
- Presentere skapkontroller via Raspberry Pi

Kommentarer:

Fikk klarhet rundt hvordan administratorsiden kan knyttes til applikasjonen, og at Id'ene til skapet hentes via Bluetooth og føres inn i resten av bestillingene via applikasjonen. Ikke helt i rute iht. prosjektplan, men har kommet godt i gang med Bluetooth delen. Ser videre på Bluetooth og administrering av Raspberry Pi via Bluetooth.

13.03.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC)

Agenda:

- Status
- Videre arbeid

Kommentarer:

Gikk gjennom hvordan vi har tenkt å jobbe videre. Ble enige om å fokusere på å få kommunikasjonen fullstendig på plass før vi tar for oss ekstra arbeid. Ser videre på kommunikasjon mellom mobil, server og klient.

20.03.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC) Øyvind Flatval (utvikler, ETC)

Agenda:

- Status
- Vise demo av systemet slik det er i dag
- Tilbakemelding
- Videre arbeid

Kommentarer:

Fikk vist frem demo av hvordan systemet vi har laget fungerer per dags dato. Ble enige om å se videre på kommunikasjon mellom mobil og mikrocomputer, og mikrocomputer og server. Ble også enige om at neste møte blir over påske, da jobbing under påsken blir

gjort hver for oss og det blir jobbet en del mindre.

03.04.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC)

Agenda:

- Status
- Videre arbeid

Kommentarer:

Gikk gjennom hvordan vi har tenkt videre og hva vi skal jobbe med til neste gang. Ettersom det nermer seg siste innspurt før grunnfunksjonalitet bør være på plass, tenker vi å sende informasjon og tanker i forhånd til løsninger fortløpende, der det er mulig å komme med innspill.

11.04.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC)

Agenda:

- Status
- Beskrivelse av nåværende system
- Videre arbeid

Kommentarer:

Gikk gjennom hvordan systemet vårt fungerer i dag, og hva vi kommer til å jobbe med for å ferdigstille dette før neste møte. Ble enige om at vi fullfører systemet slik som siste diagrammer viser, for å få en endelig fungerende løsning som kan demonstreres, forhåpentligvis neste tirsdag.

17.04.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Øyvind Flatval (utvikler, ETC)

Agenda:

- Demo av system
- Presentasjon og demo-app

Kommentarer:

Systemet pr i dag ble demonstrert, og fungerte som forventet. Det ble diskutert hvordan vi skal implementere vårt system inn i ETC-appen, og vi ble enige om at ETC gir oss en med enkle menyvalg (f.eks. «Bestill bil» og «Velg biltype»), men uten annen logikk. Denne demo-applikasjonen implementeres inn i vår nåværende app, slik at det under en demonstrasjon av applikasjonen ser ut som om vi kommuniserer med ETCs servere, men at dataene i realiteten er mockede (fra vårt eget system).

24.04.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC)

Agenda:

- Status ift. implementering av vår Xamarin-kode i ETCs Xamarin-kode
- Bildebruk i rapport

Kommentarer:

Ble enige om hvordan vi implementerer vår appkode i CarAdmin-appen. I grove trekk:

1. CarAdmin-appen startes, og når «Skanne kode»-knappen aktiveres, så kobler appen seg på nærmeste CarAdmin-nøkkelskap. Optimalt bør brukeren få opp en liste over de tilgjengelige nøkkelskapene, og velge selv hvilket han vil koble seg på. Dette er en løsning vi jobber med nå.
2. Etter at brukeren er koblet til et nøkkelskap, så gjennomføres prosessen med å ta ut en bil.

3. Når prosessen er ferdig, så skal et JavaScript kalle på en Xamarin-funksjon openLocker(int LockerID, int DoorID) med to argumenter bestemt av uthentingsalgoritmen på ETCs server.
4. openLocker(int LockerID, int DoorID) trigger oversendelse av dataene den har mottatt til Raspberry Pi, som på dette tidspunktet har nok informasjon til å åpne rett skapdør.

Ble ellers enige om at det er greit å benytte bilder av skap samt skjermbilder fra CarAdmin-appen i vår rapport.

04.05.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Dag L. Solhaug (daglig leder, ETC)

Agenda:

- Visning av full flyt i systemet via caradmin applikasjonen
- Bilder av skap
- Hvor vi skal levere kode og rapport

Kommentarer:

Viste frem hvordan demoen vil se ut for åpning av skap gjennom caradmin applikasjonen. Ble enige om å endre teksten på knappen fra "skanne koden" til "Finn skap" eller lignende. Fikk tatt bilder av skap som vi kan benytte i rapporten. Ble også enige om at det ikke er behov om flere møter med mindre noe dukker opp senere, og at vi leverer fra oss kildekode I ETCs repository sammen med rapporten.

G.2 Møtereferater - Veileder

11.01.2018 - Oppstartsmøte

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Frode Haug (prosjektveileder)

Agenda:

- Oppstartsmøte med generell kartlegging

- Veileders rolle
- Relevante fagpersoner
- Utviklingsmetodikk
- Faste møter

Kommentarer:

Gjennomgått oppgavetekst. Ble enige om at den er vag, og at hovedfokus i januar blir å kartlegge hva vi skal lage. Frode har fortalt litt om veileders rolle, og hva vi kan forvente av ham. Mariusz Nowostawski, Simon McCallum og Kjell Are Refsvik nevnt som potensielle fagpersoner innen mobilutvikling. Enige om at utviklingsmetodikk ventes med til oppgaven er klarere. Timelogg bør være maks 2 linjer pr dag (kort og konkret). Grupperegler bør etableres så fort som mulig. 3 stk statusrapporter (ca. 20. februar, 1. april og 1. mai). Fagområde, avgrensning og oppgavebeskrivelse bør være maks 2 A4-sider. Faste møter med Frode hver torsdag klokken 09:00 (kan evt. flyttes til fredag).

18.01.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Frode Haug (prosjektveileder)

Agenda:

- Går gjennom agendaen for kartleggingsmøte med ETC

Kommentarer:

Gjennomgått agendaskjemaet vi laget før kartleggsingsmøte/-samtale med ETC. Ble enige om at hovedfokus måtte bli på hva oppgaven gikk ut på, mens en del detaljer kan vente til senere.

25.01.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Frode Haug (prosjektveileder)

Agenda:

- Grupperegler signeres?

4 kopier (en til Frode)
- Prosjektplan muntlig presentasjon
- Loggbok - individuell eller for gruppa?
  individuell (både ift. timer og gjøremål)
- Fokus for prosjektplan/rapport: produkt vs prosess?
  kan vi bruke mål om læringsutbytte som resultatmål (f.eks. «Lære å utvikle applikasjoner for mobil»)
- Økonomi (f.eks. kalkulere en teoretisk pris for jobben)
  kun ift. faktiske økonomihensyn
- Hva forventes ift. litteraturbruk?
  Sommerville? Google Scholar?

Kommentarer:

05.04.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Frode Haug (prosjektveileder)

Agenda:

- Status
- Gjennomgang av statusrapport
- Product backlog

Kommentarer:

Fikk tatt opp status, og snakket om statusrapporten som ble levert med tanke på endringer i plan. Gikk gjennom hvordan product backlog skulle kunne implementeres ettersom denne ikke var gjort mye på fra før.

12.04.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Frode Haug (prosjektveileder)

Agenda:

- Status

- Kommunikasjon med oppdragsgiver
- Dokumentasjon angående dette

Kommentarer:

Gikk i hovedsak gjennom status. Tok også opp hvordan kommunikasjonen fungerer med oppdragsgiver og misforståelser i forhånd til oppgavens utforming, og hva som skulle ende opp som hovedgrunnlaget i løsningen.

26.04.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Frode Haug (prosjektveileder)

Agenda:

- Status
- Rapport struktur

Kommentarer:

Gikk gjennom status. Tok opp hvordan rapport strukturen bør se ut, med tanke på hva som skal inn i de forskjellige delene av rapporten. Ble enige om at vi må sende utkast av rapporten senest torsdag den 10. mai hvis Frode skal kunne se på den før rapporten skal leveres den 16. mai. Ble enige om å ha siste møte på fradag den 11. mai ettersom torsdagen er kristelig himmelfartsdag dag.

03.05.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Frode Haug (prosjektveileder)

Agenda:

- Status
- Tilbakemelding på statusrapport
- Tilbakemlding på rapport
- Hvordan inspera håndterer innleveringer

Kommentarer:

Fikk tilbakemelding på rapporten slik den er i skrivende stund. Ble enige i at strukturen er bra, men at noen små ting her og der må endres på. Fikk vite hvordan Inspera håndterer innleveringer, slik at om evt. problemer dukker opp siste dag er det alltids et levert dokument.

11.05.2018

Tilstede:

Markus S. Johansen (prosjektdeltaker) Fredrik Hatletvedt (prosjektdeltaker) Andreas Bradahl (prosjektdeltaker) Frode Haug (prosjektveileder)

Agenda:

- Tilbakemelding på 2. utkast av rapport
- Oppsett av referanser av litteratur

Kommentarer:

Fikk tilbakemelding på 2. utkast av rapporten. Ble enige om at struktur og det meste generelt er bra. Noen små ting må endres på, som skrivefeil og slik. Det må også legges til mer i avslutning og legges til referering til tidligere diskusjons deler i dette kapittelet. Dette er siste møtet med Frode og han er ikke tilgjengelig etter i dag.

H Arbeidslogg

Uke 2	**Sprint Første uke**
Oppsummering	Møte med Frode. Opprettet ett felles ShareLaTeX dokument, Trello plan og BitBucket gruppe. Startet på grupperegler. Utformet konseptskisse og sekvensdiagram.
Navn	**Antall timer**
Markus	12
Andreas	12
Fredrik	12

Uke 3	**Sprint Oppstartsfase**
Oppsummering	Oppstartsmøte med ETC for klargjøring av prosjektet. Mye av tiden ble brukt til å lese seg opp på relevante teknologier og muligheter som kunne brukes i prosjektet. Arbeidet med prosjektplanen.
Navn	**Antall timer**
Markus	33
Andreas	33
Fredrik	33

Uke 4	**Sprint Planleggingsfase**
Oppsummering	Hovedfokus denne uken har vært på å dokumentere oppgaven og jobbe med prosjektplanen. Lest opp på forskjellige muligheter for kommunikasjon mellom smarttelefon og Raspberry Pi (NFC eller Bluetooth). Hatt gjennomgang av prosjektplanen med veileder. Vært på kurs/ekstra forelesning (Professional Programming).
Navn	**Antall timer**
Markus	32
Andreas	34
Fredrik	30

Uke 5	**Sprint 1**
Oppsummering	Hovedfokus denne uken har vært på å fortsette på å jobbe med prosjektplanen. Møtt med oppdragsgiver for innspill angående prosjektplanen. Signert og levert prosjektavtalen og gruppereglene.
Navn	**Antall timer**
Markus	30
Andreas	30
Fredrik	27

Uke 6	**Sprint 1**
Oppsummering	Statusmøte med Electric Time Car. Her fikk vi blant annet tilgang til den delen av koden som i dag ble brukt til å styre kontrolleren. I løpet av uken ble kontrollerkoden skrevet om til å fungere for testing. Vi satte også opp egne lokale virtuelle maskiner med Debian til å testutvikle på.
Navn	**Antall timer**
Markus	31
Andreas	30
Fredrik	28

Uke 7	**Sprint 2**
Oppsummering	Statusmøte med Electric Time Car. Under møtet ble det diskutert alternativer med tanke på hvilke programmeringsspråk som var fåretrukket. Mesteparten av uken ble brukt på å lese seg opp på de forskjellige alternativene og løsningene vi kunne benytte i prosjektet, som for eksempel usb4java for å kommunisere med usb enheter i Java og Gson for å lage JSON pakker.
Navn	**Antall timer**
Markus	31
Andreas	12
Fredrik	28

Uke 8	**Sprint 2**
Oppsummering	Statusmøte med Electric Time Car. Fokus denne uken var på å utvikle koden til skapkontrollen i Java. Diverse Bluetooth løsninger og teknologier ble også utprøvd. Jobbet videre med rapporten
Navn	**Antall timer**
Markus	31
Andreas	31
Fredrik	34

Uke 9	**Sprint 3**
Oppsummering	Statusmøte med Electric Time Car. Oppdragsgiver hadde manglende oppmøte grunnet ferie som var glemt å informere om. Presenterte skapkontroller koden på Windows for første gang til hovedutvikleren deres suksessfullt. Hovedfokus denne uken var å jobbe med Bluetooth kommunikasjonen og lage en mobilapplikasjon for dette formålet. Det ble også jobbet med å ferdigstille kontroller koden og vi fikk startet på hovedrapporten.
Navn	**Antall timer**
Markus	32
Andreas	32
Fredrik	30

Uke 10	**Sprint 3**
Oppsummering	Statusmøte med Electric Time Car, hvor vi blant annet presenterte skapkontroller koden på Linux for daglige leder og diskuterte videre fremgang i prosjektet. Jobbet videre på rapport og Bluetooth kommunikasjon.
Navn	**Antall timer**
Markus	31
Andreas	0
Fredrik	28

Uke 11	**Sprint 4**
Oppsummering	Statusmøte med Electric Time Car, hvor vi diskuterte hva vi hadde fått gjort og hva som kom til å skje videre. Hovedfokus denne uken har vært å få kommunikasjon over internett og jobbet videre med Bluetooth kommunikasjonen. Møte med veileder.
Navn	**Antall timer**
Markus	33
Andreas	31
Fredrik	31

Uke 12	**Sprint 4**
Oppsummering	Statusmøte med Electric Time Car, hvor vi viste frem et prototype system for kommunikasjonsmodellen. Under møtet kom vi frem til at vi ble nødt til å gjøre endringer i hvordan systemet kommuniserer. Mesteparten av uken ble brukt til å planlegge og tilrettelegge for denne endringen, noe som innebar større endringer i logikk som var gjort hittil. Møte med veileder.
Navn	**Antall timer**
Markus	33
Andreas	33
Fredrik	30

Uke 13	**Sprint 5**
Oppsummering	PÅSKEFERIE. Det som har vært berørt denne uken har vært rapport.
Navn	**Antall timer**
Markus	2
Andreas	0
Fredrik	4

Uke 14	**Sprint 5**
Oppsummering	Statusmøte med Electric Time Car. Her diskuterte vi vårt nye forslag til kommunikasjon mellom systemene. Videre jobbet vi for å implementere for denne modellen. Underveis gjorde vi en ny endring, som vi også varslet oppdragsgiver angående. Vi jobbet også videre med rapporten.
Navn	**Antall timer**
Markus	26
Andreas	23
Fredrik	20

Uke 15	**Sprint 6**
Oppsummering	Statusmøte med Electric Time Car, hvor hovedtema var den nye kommunikasjonsmodellen. Fått grunnleggende kommunikasjon mellom alle enhetene i systemet.
Navn	**Antall timer**
Markus	33
Andreas	32
Fredrik	28

Uke 16	**Sprint 6**
Oppsummering	Statusmøte med Electric Time Car, hvor vi diskuterte implementasjon med hovedutvikleren deres angående en mer ferdig implementasjon med applikasjonen deres. Videre utvikling på denne delen.
Navn	**Antall timer**
Markus	24
Andreas	22
Fredrik	24

Uke 17	**Sprint 7**
Oppsummering	Statusmøte med Electric Time Car, hvor vi diskuterte fremdrift og videre skifte av fokus til rapporten. Fikk implementert løsningen vår med oppdragsgivers mobil applikasjon som også er under utvikling.
Navn	**Antall timer**
Markus	35
Andreas	35
Fredrik	35

Uke 18	**Sprint 7**
Oppsummering	Statusmøte med Electric Time Car, hvor vi presenterte implementasjonen vår helt ut i systemet og fikk tatt noen bilder for dokumentasjonen. Ble enige om at det var liten grunn til å holde flere møter fremover da vi har gått over til å fokusere på rapporten. Møte med veileder angående første utkast av rapporten. Skrevet videre på rapporten.
Navn	**Antall timer**
Markus	29
Andreas	25
Fredrik	28

Uke 19	**Sprint Ferdigstilling**
Oppsummering	Møte med veileder angående andre utkast av rapporten. Videre fokus og arbeid på rapporten.
Navn	**Antall timer**
Markus	30
Andreas	25
Fredrik	28

Uke 20	**Sprint Ferdigstilling**
Oppsummering	Ferdigstilling og innlevering av bachelorrapporten.
Navn	**Antall timer**
Markus	21
Andreas	20
Fredrik	23