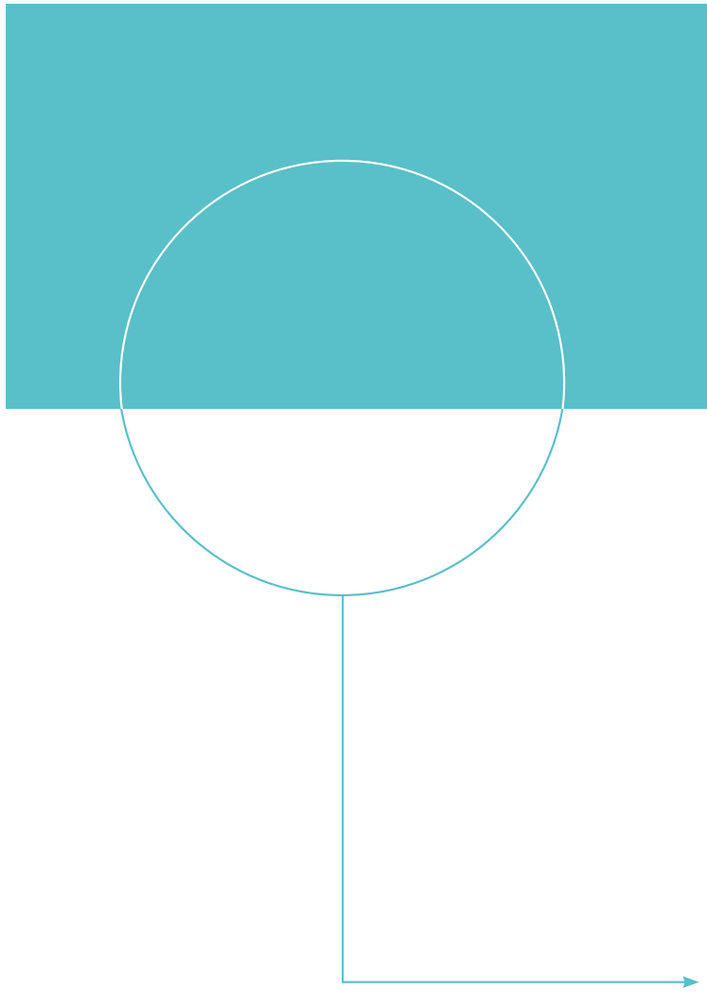


Master's thesis

2019

Matz-Leander Solheim Wiik

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Matz-Leander Solheim Wiik

NAP App - A mobile application for booking autonomous vehicles.

A research on the development of an app for booking autonomous cars and peoples trust in them.

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NAP App - A mobile application for booking autonomous vehicles.

A research on the development of an app for booking autonomous cars and peoples trust in them.

A research on the development of an app for booking autonomous cars and peoples trust in them.

Master of Science in Informatics
Submission date: June 2019
Supervisor: Frank Lindseth

Norwegian University of Science and Technology
Department of Computer Science

Summary

Fully autonomous cars are a hot topic throughout the world. These vehicles work by combining different types of LIDARs and radars to control the vehicle and make decisions, resulting in no need for human interaction. It can lead to less traffic, reduced emissions, increased security and more jobs. It is still in the testing phase, but a wide range of car manufacturers, tech companies and universities are researching within the field. With the AI technology getting more advanced, it is assumed that it is not that many years until they are getting commercialized. In order for manufacturers to earn revenue off them, it is important that it earns peoples trust and satisfies their needs.

In this thesis, it is developed a system for handling the booking of real autonomous cars. This is based on a previous project that could only simulate the booking. The main goal is to develop a system where it works on real cars. In order to solve this, two mobile apps were developed. One app is for handling the communication between the user and car, taking care of all the functionality in regard to the bookings. The other app is for the autonomous car, distributing its position to all the users. This was done by using the most trending technologies within the field of mobile app development. It is also conducted research in order to evaluate if such an app in any way could have an effect on peoples trust in fully autonomous cars and its success factors.

In order to find out how such an app should be designed, what content it should have and it's functions, both a background study and qualitative research were performed. The first iteration was evaluated with 5 co-students in order to find possible usability improvements. The qualitative test was conducted after improving the usability problems and features missing from iteration one, and resulted in a final usability score and some indications on how it can affect peoples trust. The results indicate that the application has potential and most people wanted to book an autonomous car using it but has some room for improvement. It does not seem like the app is affecting people's trust, but there are however some features that could be implemented to solve this.

Sammendrag

Autonome biler er et populært tema over hele verden. Disse kjøretøyene fungerer ved å kombinere forskjellige typer LIDAR og radarer for å kontrollere kjøretøyet og ta beslutninger, noe som ikke medfører behov for menneskelig interaksjon. De kan føre til mindre trafikk, reduserte utslipp, økt sikkerhet og flere jobber. Bilene er fortsatt i testfasen, men et stort utvalg av bil produsenter, teknologibedrifter og universiteter forsker innen feltet. Med at AI teknologi blir mer avansert, kan det antas at det ikke er så mange år før de blir kommersialisert for allmenheten. For at produsentene skal tjene penger på dem, er det det viktig at autonome biler har tillit fra allmenheten.

I denne oppgaven er det utviklet et system for å håndtere bestilling av ekte autonome biler. Dette er basert på et tidligere prosjekt som bare kunne simulere bestillingen. Hovedmålet er å utvikle et system der det fungerer på ekte biler. For å løse dette, er det utviklet to mobilapplikasjoner. En app er for å håndtere kommunikasjonen mellom brukeren og bilen, den skal utføre all funksjonalitet i forbindelse med bestillingen. Den andre appen er for den autonome biln, som distribuerer sin posisjon til alle brukerne. Dette ble gjort ved å bruke de mest moderne teknologiene innen mobil app utvikling. Det er også gjennomført forskning for å vurdere om en slik app på noen måte kunne ha en effekt på om folk kan stole på autonome biler og appens suksessfaktorer.

For å finne ut hvordan en slik app skal utformes, hvilket innhold den skal ha og hvilke funksjoner som er nyttig, ble både bakgrunnsundersøkelse og kvalitativ forskning utført. Den første iterasjonen ble evaluert med 5 medstudendter for å finne mulige feil i brukervennligheten. Den kvalitative testen ble utført etter å ha forbedret bruksproblemer og funksjoner som manglet fra iterasjon en, og resulterte i en endelig måling av brukervennligheten og noen indikasjoner om hvordan det kan påvirke folks tillit. Resultatene indikerer at applikasjonen har potensial og de fleste ønsket å bestille en autonom bil med den, men den har noe rom for forbedring. Det virker ikke som appen påvirker folks tillit, men det er noen funksjoner som kan implementeres for å løse dette.

Preface

This master's thesis was completed in the last year of the master of science in informatics at the Department of Computer Science and Information Science at the Norwegian University of Science and Technology (NTNU).

First and foremost, I would like to thank the professor supervisor from the department, Frank Lindseth, for valuable guidance and feedback through the work on this project.

I am very grateful to all the participants who took the time to participate in the user tests. This has been very valuable for this thesis.

Finally, I would like to thank my family who have supported me through all these years. I would also like to thank my lovely girlfriend who has given me a lot of support and has sticked with me during this busy period.

Matz-Leander Solheim Wiik
Trondheim, 01. June 2019

Table of Contents

Summary i

Sammendrag ii

Preface iii

Table of Contents ix

List of Tables xi

List of Figures xiv

Abbreviations xv

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Project and Context . . . 2
- 1.3 Research Questions . . . 3
- 1.4 Contributions . . . 4
- 1.5 Report Outline . . . 5

2 Background 7
- 2.1 Autonomous Cars . . . 8
  - 2.1.1 What is an autonomous car? . . . 8
  - 2.1.2 How do autonomous cars work? . . . 9
  - 2.1.3 Societal Benefits . . . 10
  - 2.1.4 Societal Challenges . . . 11
- 2.2 Trust . . . 12
  - 2.2.1 Trust in Driverless Cars . . . 12
- 2.3 Related Work . . . 13
  - 2.3.1 Waymo . . . 13
  - 2.3.2 Uber . . . 14

2.3.3 Lyft . . . 15
2.3.4 Nuro . . . 16
2.3.5 Summary of existing solutions . . . 16
2.4 Technology Review . . . 17
2.4.1 Smartphones . . . 17
2.4.2 Server . . . 17
2.4.3 Database . . . 18
2.4.3.1 SQL Databases . . . 18
2.4.3.2 NoSQL databases . . . 19
2.4.4 Mobile Application . . . 19
2.4.4.1 Native Application . . . 19
2.4.4.2 Hybrid Application . . . 19
2.4.5 Web Application . . . 20
2.4.6 Summary of Technology Review . . . 20
2.5 Designing an Enjoyable Application . . . 21
2.5.1 Cluttering . . . 21
2.5.2 Feature overload . . . 22
2.5.3 Design for Touch . . . 22
2.5.4 UI Feedback . . . 23
2.5.5 Accessibility . . . 23
2.5.6 Fast and Responsive Loading . . . 23
2.5.7 Legible Text . . . 24
2.5.8 Summary of Design Theory . . . 24

3 Methodology and Implementation 25
3.1 Process . . . 26
3.1.1 Development Process . . . 26
3.1.2 Research Process . . . 27
3.2 Development Overview . . . 28
3.3 Iteration 1 . . . 28
3.3.1 Planning and Requirements . . . 28
3.3.2 Analysis and Design . . . 30
3.3.2.1 React Native . . . 30
3.3.2.2 Expo . . . 31
3.3.2.3 Redux . . . 32
3.3.2.4 MySQL . . . 33
3.3.2.5 Node with Express . . . 33
3.3.2.6 REST API . . . 35
3.3.2.7 Database Model . . . 35
3.3.2.8 Overall Architecture . . . 36
3.3.2.9 Design . . . 37
3.3.3 Implementation . . . 37
3.3.3.1 Server and Database . . . 38
3.3.3.2 NAPApp . . . 38
3.3.4 Testing and Evaluation . . . 40
3.4 Iteration 2 . . . 41

3.4.1 Planning and Requirements . . . 41
3.4.1.1 Requirement specification - NAP App . . . 41
3.4.1.2 Requirement specification - NAP App server . . . 42
3.4.1.3 Requirement specification - NAP App admin . . . 42
3.4.1.4 Software and Hardware Requirements . . . 43
3.4.2 Analysis and Design . . . 43
3.4.2.1 Google Cloud Platform . . . 43
3.4.2.2 Vue Js . . . 45
3.4.2.3 Bcrypt . . . 46
3.4.2.4 Oauth2 . . . 46
3.4.2.5 JSON web token . . . 47
3.4.2.6 Websockets . . . 48
3.4.2.7 Database Model . . . 49
3.4.2.8 Overall Architecture . . . 50
3.4.3 Implementation . . . 51
3.4.3.1 Navigation . . . 51
3.4.3.2 Implementing REST API and database on Google Cloud Platform . . . 52
3.4.3.3 Authentication . . . 53
3.4.3.4 Real-time Booking of Real cars . . . 53
3.4.3.5 Administration Tool . . . 54
3.4.4 Testing and Evaluation . . . 54
3.4.4.1 User Test . . . 55
3.4.4.2 Alpha Test . . . 55

4 Results **57**
4.1 iteration 1 . . . 58
4.1.1 Solution for Requirement Specification . . . 58
4.1.2 Device Test . . . 63
4.1.3 Evaluation . . . 64
4.2 Iteration 2 . . . 65
4.2.1 Design Improvements . . . 65
4.2.2 Solution for Requirement Specification . . . 68
4.2.2.1 Requirement Solutions for Nap App . . . 68
4.2.2.2 Requirement Solutions for Nap App Admin . . . 75
4.2.2.3 Database and Server hosting in Google Cloud Platform 78
4.2.3 Evaluation . . . 81
4.2.3.1 Alpha test . . . 81
4.2.3.2 User test . . . 81

5 Discussion **91**
5.1 Development . . . 92
5.1.1 Front-end development . . . 92
5.1.1.1 React Native . . . 92
5.1.1.2 Expo . . . 92
5.1.1.3 Vue.Js . . . 93

5.1.2 Back-end Development . . . 93
5.1.2.1 Node.Js . . . 93
5.1.2.2 Google Cloud Platform . . . 93
5.1.3 Functionality of the app . . . 94
5.2 Success Factor . . . 94
5.2.1 Usefulness . . . 94
5.2.1.1 Safety . . . 95
5.2.1.2 Efficiency . . . 95
5.2.1.3 Pricing . . . 95
5.2.1.4 Environment friendly . . . 95
5.2.1.5 Carpooling . . . 95
5.2.2 Usability . . . 96
5.2.2.1 Generally intuitive . . . 96
5.2.2.2 Choosing destination . . . 96
5.2.2.3 Choosing location by tapping on the map . . . 96
5.2.2.4 Displaying time estimate before booking . . . 96
5.2.3 Experience . . . 96
5.3 Trust in autonomous cars . . . 97
5.3.1 Reasons for trust . . . 97
5.3.1.1 Experience using ADAS . . . 97
5.3.1.2 Experience and Testing . . . 97
5.3.1.3 Accidents . . . 97
5.3.1.4 Bugs or Faults . . . 98
5.3.2 Improving the trust . . . 98
5.3.2.1 Feedback from other users . . . 98
5.3.2.2 Information about the cars . . . 98
5.3.2.3 Testing and Security . . . 99

6 Conclusion and Further Work **101**
6.1 Conclusion . . . 101
6.2 Future Work . . . 102
6.2.1 Choose destination using viewport . . . 102
6.2.2 Carpooling . . . 102
6.2.3 Pricing . . . 102
6.2.4 Car information . . . 103
6.2.5 Time estimate before choosing pickup location . . . 103
6.2.6 User Experiences . . . 103

Bibliography **105**

Appendix A - Persona **113**

Appendix B - Github Repositories **114**

Appendix C - Nap App user test tasks **115**

Appendix D - Google Cloud billing for May 2019 **116**

Appendix E - User test questionnaire **117**

Appendix G - Demonstration video **123**

Appendix H - Answers from questionnaire **124**

List of Tables

3.1 Functional requirements of iteration 1 29
3.2 Non-functional requirements of iteration 1 29
3.3 Functional requirements - NAP App of iteration 2 41
3.4 Non-functional requirements - NAP App of iteration 2 42
3.5 functional requirements - NAP App server of iteration 2 42
3.6 functional requirements - NAP App admin of iteration 2 43
3.7 non-functional requirements - NAP App admin of iteration 2 43

4.1 Solved requirements for figure 4.1 59
4.2 Solved requirements for figure 4.2 60
4.3 Solved requirements for figure 4.3 61
4.4 Solved requirements for figure 4.4 62
4.5 Devices the app was tested on 63
4.6 Solved requirements for figure 4.9 68
4.7 Solved requirements for figure 4.11 70
4.8 Solved requirements for figure 4.12 71
4.9 Solved requirements for figure 4.13 72
4.10 Solved requirements for figure 4.15 74
4.11 Solved requirements for figure 4.16 75
4.12 Solved requirements for figure 4.17 76
4.13 Solved requirements for figure 4.18 77
4.14 Requirements not solved for the admin tool 78
4.15 User background overview 82
4.16 User test procedure and time estimate 82
4.17 Average SUS score per question 85
4.18 Correlation between ADAS experience and trust 86

List of Figures

2.1 Different levels of autonomy 8
2.2 The Nuro vehicle 9
2.3 Different technology used in an autonomous car. 10
2.4 Percent of drivers afraid of driverless vehicle 12
2.5 Waymo one car and screenshot from Waymo app **Source:** Waymo 13
2.6 Screenshot from the Uber app **Source:** The Verge 14
2.7 Autonomous car used by Uber **Source:** Archpaper 15
2.8 Screenshot from the Lyft app. **Source:** Lyft 15
2.9 The tiny autonomous cars are trailed by "shadow cars" in case something goes wrong. **Source:** New York Times 16
2.10 Illustration of how the back-end architecture works. **Source:** Upwork Global 18
2.11 The clear tab bar (right) is much better than the cluttered one (left). **Source:** Apple 22
2.12 10 by 10 mm is a good minimum touch target size. **Source:** UXmag . . . 22
2.13 The faster your app, the better the experience will be. **Source:** Google . . 23

3.1 Illustration of the process developing an app. **Source:** Testing Excellence 26
3.2 Stages used in Oates research model. **Source:** (Oates, 2006) 27
3.3 Apps using React Native from June 2018. **Source:** Appbrain 30
3.4 Workflow developing a react native app in Expo **Source:** Expo docs . . . 31
3.5 React component tree and Redux flow **Source:** Quora 32
3.6 Entity relational diagram from iteration 1 36
3.7 System architecture from iteration 1 37
3.8 Different zones and regions in Google Cloud Platform **Source:** Google . 44
3.9 Vue reactivity model **Source:** Vue 45
3.10 OAuth2 authorization flow **Source:** Digital Ocean 47
3.11 Illustration showing how WebSockets work. **Source:** Medium 48
3.12 Entity relational diagram for iteration 2 49
3.13 System architecture for iteration 2 51

4.1 Map screen from iteration 1 58
4.2 Address and pointer screen from iteration 1 60
4.3 Booking screen from iteration 1 61
4.4 Screenshots showing how the user can track the booked car 62
4.5 Screenshots from both an Android and iOS device 63
4.6 Map screen from iteration 2 65
4.7 Address input screen from iteration 2 66
4.8 Car booking design changes 67
4.9 Login and Sign up screens. 68
4.10 Sign in details either using Google or LinkedIn 69
4.11 Map status 70
4.12 Simultaneous bookings by different users 71
4.13 Car moving in real-time 72
4.14 Screenshots from the app running inside the car 73
4.15 Profile screen showing booking history 74
4.16 Login page from admin tool 75
4.17 User overview in admin tool 76
4.18 Cars overview in admin tool 77
4.19 App engine traffic May 2019 78
4.20 Graphs showing CPU and memory usage from the server 79
4.21 Latency between client and server, May 2019 79
4.22 Graphs showing total connections, CPU usage and memory usage 80

Abbreviations

AAA	=	American Automobile Association
ADAS	=	Advanced driver-assistance systems
ADS	=	Automated Driving Systems
AI	=	Artificial Intelligence
API	=	Application Programming Interface
CSS	=	Cascading Style Sheet
GUI	=	Graphical User Interface
HTML	=	HyperText Markup Language
HTTP	=	Hypertext Transfer Protocol
HTTPS	=	Hypertext Transfer Protocol Secure
iOs	=	iPhone Operating System
IOT	=	Internet of Things
JSON	=	JavaScript Object Notation
JWT	=	JSON Web Token
NAP	=	NTNU Autonomous Perception
NoSQL	=	Non relational Structured Query Language
NPM	=	Node Package Manager
PHP	=	PHP: Hypertext Preprocessor
REST	=	REpresentational State Transfer
SDK	=	Software Development Tool
SQL	=	Structured Query Language
SSD	=	Solid State Drive
SSL	=	Transport Layer Security
SUS	=	System Usability Scale
TCP	=	Transmission Control Protocol
UI	=	User Interface
URL	=	Uniform Resource Locator
UX	=	User Experience
VM	=	Virtual Machine
W3C	=	World Wide Web Consortium

Chapter 1

Introduction

This chapter introduces the report, motivation of the thesis, the research questions, research methods and contributions

1.1 Motivation

2018 was a big year within autonomous vehicles. Waymo was on their way to launch a driverless taxi service and Uber had started to log data based on their self-driving vehicles (Carson, 2017; Lee, 2017). Because of some setbacks, both had to delay their development, while they took time to evaluate their systems and algorithms. Even if the development had stagnated by the biggest companies, the promotion around autonomous vehicles are still growing and now we can see more smaller companies and universities starting to develop and test more cars (Lee, 2018). According to various sources it is one of the hottest trends within the tech area (BBVA, 2019; Panetta, 2018). With the growth of autonomous cars, we can get a lot of benefits in the society. These can be reducing accidents based on driving fatigue, Lidars could potentially perceive the environment better than humans, which results in less traffic. Since most of these vehicles are electric, reduced CO2 emissions can also be achieved.

While autonomous vehicles brings a lot of positive factors to the society it is still in an early stage and one of the big problems is earning the peoples trust (Luehrs, 2018). Some of the reasons is because we are putting a lot of responsibility in the hands of a machine while we humans don't have any control over the driving ourselves (Kaur and Rampersad, 2018). In recent times we also had some accidents with Tesla autopilot (Stewart, 2018) and a self-driving Uber (Salinas, 2018) which greatly influenced peoples trust in fully autonomous cars. While these accidents can be a big factor on why so many people don't trust fully autonomous cars, it also seems like the majority of the people want to control the vehicle themselves (Kaur and Rampersad, 2018).

In this project, it would be interesting to see if people would trust or use an autonomous car if the offer is available. Also asking people for other AV usage reasons. It will be evaluated whether a system combined with mobile applications can help to make the booking of autonomous cars, and which technologies can solve this. It needs to be a full stack containing both mobile apps and functionality to communicate with server and database. It is important that it is implemented in such a way, that makes it possible to use this with real autonomous cars. For example since Uber arrived on the market, people are booking more and more rides. This is not on the cost of the Taxi industry, since the total Uber bookings are increasing, but the taxi industry has not experienced any decline(Iqbal, 2019). The problem presented here are the motivation for conducting this research. The goal in this project is to develop a seamless and intuitive app, test it on users and get their feedbacks. Based on this, we will look into why people are more interested in riding autonomous cars.

1.2 Project and Context

The task is given as follows:

[NAPApp] - Booking service for autonomous cars. *In this project, the goal is to develop an app for booking autonomous cars. The mobile application should handle the communication between the cars and the end users. It also needs to handle all functionality for*

car reservations, and should include both car distribution and route planning. The project should also include an app for the car which broadcasts its position to the users and an administration panel for admins.

The first phase of the project will consist of a theoretical study of existing apps within the area. It also includes studies on how to efficiently develop an app of this scale. The second phase focuses on implementing a prototype using the technologies found in phase one. The third and final phase will evaluate and test the prototype.

This master thesis is based on work that was done in a previous thesis (Mathisen, 2018). Here, it was developed an iOS prototype app for simulating the booking of autonomous cars. It was also conducted studies on how the sharing economy could be influenced by the technology.

During the work on this master's thesis, the app was developed from the ground up, using elements from the previous master's thesis. It was made sure that the new app is working on both Android and iOs and could be used in production. This means that the app could be used live on both users and cars. It was developed two mobile applications. One app is for the users, where the main goal is to perform booking and route planning. The other app is used for the cars, broadcasting it's position to the users. Furthermore the system consists of a server, connecting the app to a database and a webapp for admins. The webapp is a tool for admins where they can add/remove cars, manage users and get an overview over rides.

In the end the app was tested on users to gather data which can determine the user experience and the effects the app has on its users. This data was then used to answer the research questions listed in the next section.

1.3 Research Questions

The main goal of this thesis is to develop a software system for booking autonomous cars. It will also do research on people's trust in fully autonomous cars, and their reasons for either trusting them or not. If people easily have access to an application for booking, will they disregard their trust or not?

The following research questions are defined for this project:

RQ1: In what way is it possible to develop an application for the booking of genuine autonomous cars, using modern technology?

RQ2: Which factors affect the success of an autonomous car booking app?

RQ3: To what extent can the app affect whether users rely on autonomous cars?

1.4 Contributions

The contributions from the project includes a second and final version of the app NapApp that was developed in (Mathisen, 2018), n app to be used in the autonomous car, a server running in Google Cloud Platform, feedback from users testing the app, as well as suggestions on how to further improve the system. Furthermore, this thesis also provides some research on the success factors and, how the app can have affect on the users trust in autonomous cars.

1.5 Report Outline

The chapters in the thesis are structured as follows:

Chapter 1 - Introduction Introduces the report, motivation of the thesis, the research questions, research methods and contributions

Chapter 2 - Prestudy Contains theory for the material used in the thesis. Furthermore, a review of technology relevant to the thesis is conducted. From there theories in how to design enjoyable apps are presented.

Chapter 3 - Methodology and Implementation Presents the methodology used for the development, and the research methods used to evaluate the system. Furthermore the technology used in the development, collection of data and how the experiment was executed will be described here.

Chapter 4 - Results Presents the results from the development. These are measured up to the requirements in chapter 3. Furthermore the results from the user tests are also presented

Chapter 5 - Discussion Discusses the results from chapter 4 against research questions presented in chapter 1.

Chapter 6 - Conclusion and Future Work Presents the conclusion of the project as well as suggestions for future work.

Chapter 2

Background

The background study is done in order to get a better understanding of the topic, a better overview over existing solutions, best solutions in modern technology and the foundation for the overall design. In this chapter, it will first present the theory around the project, before giving a presentation over already existing solutions. Furthermore, a review over technology that needs to be used in order for the thesis to be carried out, lastly it will look into theories for designing an application that is fun to use.

2.1 Autonomous Cars

An autonomous car (sometimes called self-driving car or driverless car) is a vehicle that uses cameras, LIDARs and radars in order to get from one point to another without a human driver (Rouse, 2018). It is a popular subject throughout the world, this is probably because it might bring a lot of changes to the society we know. It can solve a lot of problems like reducing traffic accidents caused by human drivers. Also it can help the world market by providing necessary services within shipping and transportation.

We can see that vehicles with automated driving systems (ADS) has been researched on since at least the 1920s. The first prototype cars appeared in the 1980s. These cars could drive by itself in one lane with the speed of 31 kilometres per hour, later in the 80s they added obstacle avoidance and day/nighttime conditions. The research continued throughout the 90s and the technology got more and more advanced. In 2017 Audi released their first fully autonomous car. They claimed it could drive up to 60 kilometres per hour without a human driver. It was the first car to reach level 3 automated driving. You can read more about the automated driving levels in the next section. In December 2018 Waymo was the first company to commercialize a fully autonomous taxi service in the U.S, using an AI driver that had driven for 10.000.000 miles. (Wikipedia, 2019d)

2.1.1 What is an autonomous car?

A lot of cars in the market now use Advanced driver-assistance systems(ADAS), although it provides important safety mechanisms such as pre-collision warnings, steering assistance, object detection and automated braking it still needs a human driver. The US National Highway Traffic Safety Administration (NHTSA) has defined different levels of car autonomy. This is to set a standard for autonomous vehicle testing. (Hendrickson, 2019) in figure 2.1 the different levels are defined as follows, starting with level 0:

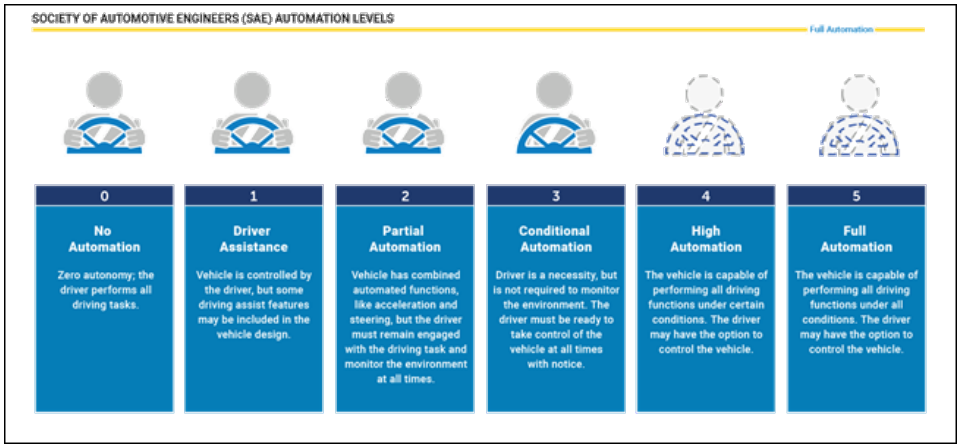


Figure 2.1: Different levels of autonomy

The vehicles this thesis is centered around are cars level 5 automation. Full automation would say that the car could fully drive by itself even or without a passenger. As of now the only fully automated cars on the road are made by a company called Nuro (Nuro, 2019). Currently they are being tested for delivering groceries picked up from a store to the customers. They can only travel short distances as for now and they don't have any human passengers.
Currently most of the cars being developed by Lyft, Uber and Google falls under the level 4 capability. These requires a safety driver and they are being tested on level 2 and 3 standards. In the Waymo vehicles there are no safety driver, but these are only being tested on dry conditions. (Hendrickson, 2019)

Figure 2.2: The Nuro vehicle

2.1.2 How do autonomous cars work?

Autonomous vehicles use a wide range of technologies. As explained earlier in the chapter they use cameras, LIDARs and radars in order to get from one point to another. In today's autonomous vehicles they are used together with each other to get a level of autonomy that manages to produce a vehicle that falls under level 4 or 5 in figure 2.1.
For example Tesla's autonomous car has a feature called "autopilot", that uses cameras in order to get a 360-degree vision. It also uses ultrasonic sensors and front-facing radar in order to analyze its surroundings, like for example Snow or rain. (Armstrong, 2018)
It uses the radar sensors to detect other vehicles nearby. The video cameras detects traffic light, road signs, other cars, pedestrians and other obstacles. The lidar sensors helps the car to stay on the road by detecting the edges of the road and lane markings. Lidars does this by measuring the time difference or changes in the wave phase between an emitted laser signal and a reflected light (Wikipedia, 2019a).
It uses ultrasonic sensors in the wheels in order to detect kerbs. They work by emitting an ultrasonic wave and receives the wave reflected back from the kerb (Keyence, 2019). To

manage and analyze all of the data collected by the sensors and radars the car uses a central computer. When analyzing the data, it controls the steering, braking and acceleration based on it. In figure 2.3 you can see an illustration of all the different sensors and radars used on the car.

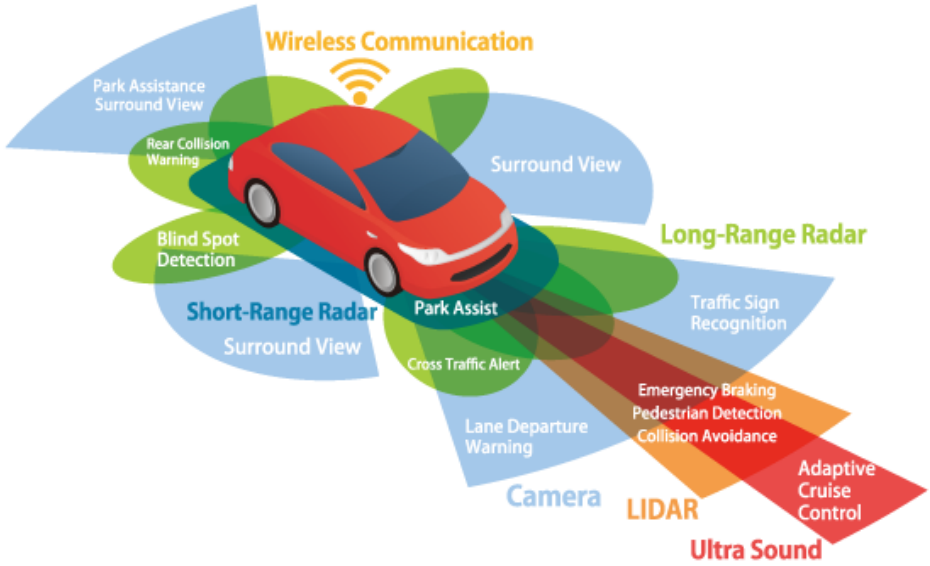


Figure 2.3: Different technology used in an autonomous car.

A key component for the cars to work for a certain standard is the 5G cellular network. It is still in the testing phase, but has started to be released in some cities around the world. 5G is the same as we know as today's 4G network for mobile phones, but it can transfer data at a much faster rate. According to reports 5G are going to be atleast 700% faster and have a lot better response times than today's 4G network (Gozalvez, 2015). When the autonomous cars can start to use 5G networks we can get a lot more robust and faster communication between one car to another. Also we are getting more IoT devices in our society, where everything from a garbage can to traffic lights will be connected to high-speed network, enabling for even more communcation between objects in the traffic (Chong and Ng, 2016; Dividend, 2018)

2.1.3 Societal Benefits

By getting fully autonomous cars on the roads we get a lot of benefits in our society. One of the most important benefits are decreased accidents. According to (Crist and Voege, 2018) advanced driverless cars are predicted to reduce accidents by ninety percent. A big factor is that we can completely eliminate drunk and distracted driving accidents.

Since driverless use electric power and be programmed, we can obtain reduced emissions by driving more efficiently (Greenblatt and Saxena, 2015; Ohio-University, 2019).

We can also assume that with the growth of electric cars, most of the driverless cars in the future will use an electric battery, this will also affect the total emission.

We can get less traffic on the roads since driverless cars can calculate both the best route and how to avoid most of the traffic. According to a study done by (Ohio-University, 2019), Americans living in urban areas spend 6.9 billion hours in traffic, incur 160 billion dollars in congestion costs and waste 3.1 billion gallons of fuel. With driverless cars most of these aspects will be reduced.
There are some other societal benefits, but these are considered the most important ones.

2.1.4 Societal Challenges

With the challenges we are facing right now I don't think we are quite ready for having fully autonomous cars on the roads.
As of right now 71 percent of Americans fear fully autonomous vehicles and only 19 percent would put their kids in them according to a survey done by AAA (American Automobile Association) (Naughton, 2019). More about trust in autonomous vehicles can be found in the next section.
Another big challenge is weather conditions. Falling rain and snow can interfere with the cars radars and LIDARs, and need to be further tested before commercializing driverless cars (Kiss and Berecz, 2019). The technology isn't 100 percent safe and reliable yet. In a California pilot program (Ohio-University, 2019) they experienced these issues with the different manufacturers:

- Google experienced one incident every 1.244 miles.
- Volkswagen experienced one incident every 57 miles.
- Nissan experienced one incident ever 14 miles.

Most of these accidents involve clashes with other cars and objects. Only a few of these have inflicted damage to humans. In order to start utilizing the technology in today's society these numbers are too high, but the different manufacturers are continuing to improve and find bether solutions as we speak.

2.2 Trust

Trust can be defined as one part is relying on the actions of a second part. For the second part action to be performed in association with the first part, the second part needs to have the first parts trust. Trust can be a factor in the relationship between people. Humans often uses trust in order to judge another individual. (Wikipedia, 2019e)
Nowadays we see a lot of questions of trust between human and machines. As the development of AI are increasing, people needs to trust them in order for applying them to our everyday life (IBM, 2019a). Wehn a person develops trust, they believe in the ability of the tehnology to behave in the way the person is expecting.

2.2.1 Trust in Driverless Cars

The social attitude in 2019 toward driverless cars are not good. According to AAA's annual survey, 71 percent of the American people are afraid to ride in a fully self-driving vehicle (Edmonds, 2019) and in a study done by HERE in 2018, 62 percent did not trust them (HERE, 2018). People may find it difficult to lose the control when they are not behind the driving wheel. Having to deal with more automated systems and reading about it in media seems to affect peoples opinion about it. Also according to the survey, experience seems to impact what people think about automated technology. A lot of cars in todays market are equipped with ADAS and people who uses it seems to have a better comfort level when riding. Furthermore consumers who uses ADAS systems are about 68 percent more likely to trust the features, compared to consumers without them (Edmonds, 2019).
It seems that people gain more trust by experience in this study, using the Tesla autopilot, they gain more trust in automated technology overall (Koskinen et al., 2019).

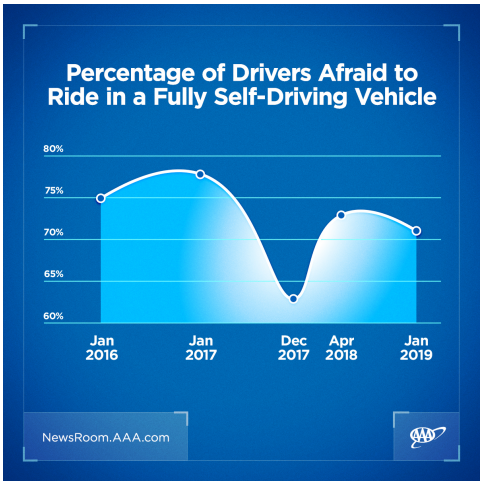


Figure 2.4: Percent of drivers afraid of driverless vehicle

The good part is that 53 percent of the people in the survey trusts automated vehicles with low speed in limited areas like airports and 44 percent trust autonomous vehicles

without any driver, like delivery of groceries. But it seems that Americans are ready to take the technology into their daily lives, just in a steady pace. According to Brannon in (Edmonds, 2019) ”Hands-on exposure in more controlled, low-risk environments coupled with stronger education will play a key role in easing fears about self-driving cars.” Although this could help people gain more trust in autonomous vehicles we also need to improve the security even more and we need to do more testing.

Like we can see in figure 2.4 the trust level towards driverless cars have remained mostly unchanged. But the attitude had some drastic change in 2017, this could probably be because of the accidents that occurred with Tesla autopilot in 2016 (Wikipedia, 2019b).

2.3 Related Work

In this section research has been conducted in order to understand what has been done earlier in the field of autonomous vehicle booking. A small selection of existing solutions will be briefly presented, explained and evaluated.

2.3.1 Waymo

Waymo is a company that specializes in self-driving technology. It started out as a Google project and became a stand-alone subsidiary in December 2016 (Wikipedia, 2019g). In 2017 Waymo started testing a taxi service with autonomous vehicles in Phoenix, Arizona. Later on they started testing the service in 25 cities across the U.S over a time-span of nine months. The testing had such a big success that in December 2018, they launched a commercial self-driving car service called Waymo One. As of now it is only available in Arizona and people in metropolitan areas can use the waymo app for pickups. (Wikipedia, 2019g) As seen in figure 2.5 a picture of the car used in the Waymo One project and a screenshot from the app used to book these cars.

Figure 2.5: Waymo one car and screenshot from Waymo app
Source: Waymo

The technology equipped on the Waymo cars are LIDARs, giving 360 degree views and lasers detecting objects up to 300 meters away. The cars also has short range lasers for detecting close objects like cars driving on the side. It's also equipped with a radar to keep track of other vehicles and objects in motion. (Wikipedia, 2019g)

2.3.2 Uber

Uber is a taxi service offering peer-to-peer ridesharing. You can hire a private driver to pick you up and take you to your destination just by using their mobile application. Often your driver is nearby to pick you up, and you can even watch your drivers route while waiting for the pickup (Uber, 2019). The drivers can be all kinds of people either using it as a part time job or full time job. It allows for some income and an opportunity to meet new people in your city.

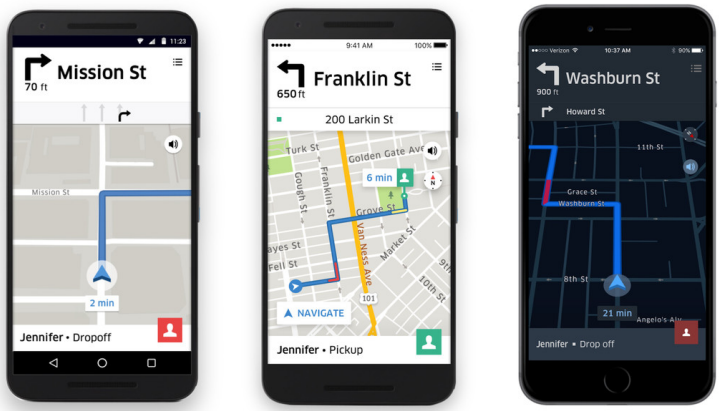


Figure 2.6: Screenshot from the Uber app
Source: The Verge

In 2016 Uber launched their first self-driving service. Using Volvo XC90 SUV's programmed for self-driving they started testing in San Francisco. Later in 2016 the state of California stopped the program and Uber started testing in Arizona instead. In 2017 They had a plan to buy 24.000 of the XC90 to continue their development, unfortuntely the year after one of their self-driving Volvo's killed a person which resulted in a temporary pause in the testing (Wikipedia, 2019f).

Now Uber has put their cars on the roads again to continue their developent in starting their self-driving taxi service. This time they are testing in San Francisco, driving in limited areas, reduced driving speed and also having a safety driver taking control of the vehicle if something is about to happen. It's been difficult for them getting the cars back on the roads, struggling to pass safety tests. (Conger, 2018)

Figure 2.7: Autonomous car used by Uber
Source: Archpaper

2.3.3 Lyft

Lyft is similar to Uber, a Taxi service offering peer-to-peer ridesharing, or a transportation network. In this category, they are the second biggest company and you can either book or drive cars, scooters or bicycles. They offer the same services as Uber, but the difference between these two is that Lyft are offering their platform to companies producing driverless cars. For example, Waymo, Ford, General Motors and other companies are now using Lyft's platform for booking autonomous vehicles. Using a big actor like Lyft in this market to make more people try driverless cars is a big step in making more people want to try it out.(Wikipedia, 2019c)

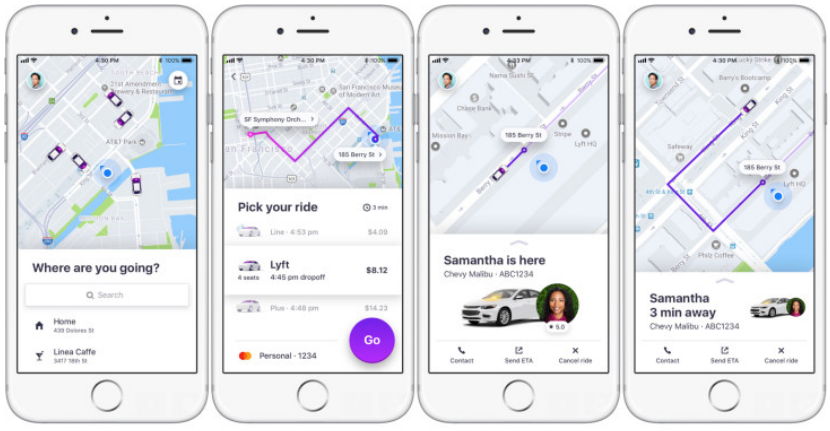


Figure 2.8: Screenshot from the Lyft app.
Source: Lyft

2.3.4 Nuro

Nuro is a startup company developing autonomous delivery vehicles. They started in January 2018 and has developed small vehicles not even half the size of a Volkswagen Beetle, it has no driver and no passengers, its only purpose is to deliver groceries to people. It's designed to only carry one cargo, with space of 12 grocery bags. They have already partnered with big supermarket chains such as Kroger and Fry's Food and Drug store, delivering their products to customers. (Metz, 2018)

What differs Nuro from Waymo and Uber, are that they are developing a driverless service not for passengers, but other practical uses - in this case, food delivery. By using smaller vehicles as the size of a golf chart, they don't need a lot of space and they can drive at slower speeds reducing the risk of big accidents (Metz, 2018). As of now this service are still in the testing phase, using shadow cars with real drivers following the small Nuro vehicles if something were to happen.

Figure 2.9: The tiny autonomous cars are trailed by "shadow cars" in case something goes wrong.
Source: New York Times

2.3.5 Summary of existing solutions

We can see on all of these existing solutions that driverless vehicles and booking services are still in the testing phase. The technology seems to be in the right place, but this is something that can't have any security flaws as people's lives are at stake. The development seems to be going forward, by taking more and more baby steps into an ideal driverless taxi solution. There are also many more companies developing driverless cars, such as General Motors and Ford, but all of the companies are at the same stage in development.

It also seems that more companies are starting to use the technology for other driverless solutions, such as transportation of groceries, picking up laundry, post delivery and transportation of beer (Metz, 2018). These solutions are all in early stage but it seems like Nuro

are the ones that's been tested the most. Most of the companies working with this technology think that we will have driverless taxi cars driving on roads aroung 2025. (Metz, 2018)

2.4 Technology Review

The main goal of this project is to develop a service for booking autonomous cars. In order to do this, a review of relevant technology for this project is needed. This section will provide a brief explanation of these solutions.

2.4.1 Smartphones

In today's smartphones we have access to a variety of different tools such as sensors. This makes smartphones an interesting choice when developing an application, because of more opportunities not possible on a traditional computer. One of the most important sensors for this project, are the sensors for tracking the position. In a smartphone we have (GPS, WiFi, and cellular network). Other useful sensors are accelerometer, gyroscope, microphone and cameras. (DeviceSpecifications, 2018)

A lot of similar apps like the one developed in this project use the GPS to track the users location. In our case we are also relying that the GPS is so accurate that it can track the position of the driverless car.
The possibilities we are gaining from the hardware and sensors in the modern smartphones are letting us develop a wide variety of applications we can use wherever we are. That being said, when designing an application for a smartphone, it can be difficult because of the limited space on the screen, but a lot of different design patterns has been developed in order to get an enjoyable user exeperience.

2.4.2 Server

The server or back-end is the machinery that works behind the scenes of what the users see or interact with. The reason why we need a server is because we need something that can pick out the appropriate data for the user. For example when a user logs in with their username and password, the server need to check if these details match with a registered user in the database. If the details match, user details gets returned to the client. In our project a server's gonna be necessary, since the users will do a lot of requests either to add or get data from a database.(Wodehouse, 2019)

Usually for a server to work, we have to implement some kind of script or code that is running continuous on a computer. Figure 2.10 illustrates how the back-end server are working from a front-end request to the database. There are also different kind of back-end servers, some servers are physical units stored in your workplace or area, while other servers are stored and managed by big companies. These are often called cloud servers, they can either be physical or virtual. The virtual machines(VM) are created, using virtualization software to create a physical server into multiple virtual servers. The

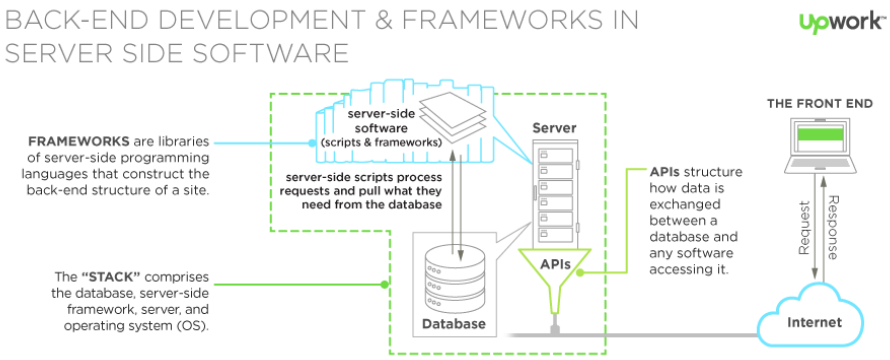


Figure 2.10: Illustration of how the back-end architecture works.
Source: Upwork Global

resources from the physical unit are spread throughout all the virtual units. (IBM, 2019b) What people are choosing are usually up to what would be most cost effective between having your own physical server or a virtual server hosted by someone else.

2.4.3 Database

In order to have all of the data organized in different forms, a database is required. It also makes all of the data accessible in many possible ways. In order to store large chunks of data you will need a database, yes it is possible to use a spreadsheet, but as the data gets large enough, a spreadsheet is not enough to keep things organized. Furthermore, database languages such as MySQL also validates an update before recorded. This provides good accuracy managing your data and reducing flaws when editing it.

Just like the servers we have different kinds of databases. they are often split into two groups: Relational databases (SQL) and non-relational databases (NoSQL). While both are viable options, there are certain key differences between them.

2.4.3.1 SQL Databases

An SQL database use structured query language for manipulating data. This is one of the most used options today and provides great documentation for how to use it. Because of this it is a safe choice for complex queries of data since it's been used for a long time, however it is more restrictive because you have to create pre-defined schemas of how the data are gonna be organized, and all of your data has to follow the same structure (Xplenty, 2017). If you want more structure in your data, this is the best choice. Since it is more restrictive it is easier building relational data. This can provide some ease of use, as the data is more divided into tables, and presents an overview over all the relations between them.

2.4.3.2 NoSQL databases

Non relational query language (NoSQL) has no pre-defined scehma, however you can use it if you like. You can store unstructured data in different ways: By columns, document-oriented, graph-based or organized with keys. In NoSQL a table is defined as a document and they can have their unique structure since we're not defining any structure before manipulating the data. You can add fields all the time without the need for changing other documents since each document is not dependant on another like the tables in SQL (Xplenty, 2017).

2.4.4 Mobile Application

A mobile application is a piece of software running on a smartphone. Just like any software running on a computer, this has the same goal: To cover some needs of a user. Using a mobile application has several advantages, like having access to the phone's hardware like mentioned in section 2.4.1. A mobile application can make use of the device performance in a more efficient way compared to for example a web application running in the web browser. They can also be developed in such a way that they are working even when the phone is disconnected from the internet. However, the negative side of mobile applications is that changes to an application doesn't come automatically, you have to update the application through an integrated software like Googles Play store or Apples app store.
Also there are two kinds of mobile applications, native and hybrid apps.

2.4.4.1 Native Application

A native application is a software developed to perform a task on a particular environment platform. This application is developed using a software development tool (SDK) for a specific framework, platform or operating system (Dua, 2019). Android apps are built using the Java development kit and Java programming language, iOS apps are built using iOS SDK and Swift or Objective C language. The benefits of a fully native application are performance, because native code are the fastest solution to get the most out of the phone's hardware. It allows accessing the devices features without using plugins and if an update for the SDK allows for new features, it will be available here first (Patro, 2018). However there are some drawbacks as well. A native app for each platform needs to be coded in different languages, increasing the development time. This also causes the deployment of the app challenging and you require different skills from the developers, which could increase the cost. (Patro, 2018)

2.4.4.2 Hybrid Application

Hybrid apps are almost the same as a native app. The main differences is that Hybrid apps are built with web technology like HTML, CSS and JavaScript (Dua, 2019). The app is running in a web view, which is a view of a web browser inside a native application. To access the native features of the phone such as the camera, you need to use a native plugin in your framework (Patro, 2018). The most important aspect and why most people are developing a hybrid app is because you get a single code base for an application released on

different platforms (Patro, 2018). The biggest benefits of developing a hybrid application are probably that the same team can deliver an app for any platform. You can almost use the same codebase if you already have developed a web application and want to transform it into a native app, you just need to resize the screens. Biggest drawbacks are probably the performance, hybrid apps will struggle with animations and 3d (Patro, 2018). They will also suffer a bigger response time when pressing buttons since it is running inside a webview. Also, if a plugin for the chosen framework is not available you won't get access to the hardware unless you develop the plugin yourself.

2.4.5 Web Application

Web apps are launched and runs on a web browser, they are essentially a website designed to look like an application. Instead of installing a piece of software on the device, users are interacting with the app through the browser or webview. They are usually easy to build, maintain and a cheaper solution compared to a native app; However, they require a web browser and the performance are usually slower than native apps (Dua, 2019).

But what differs a webapp from a normal website? You can say that a normal webpage are designed to show a lot of information, while a web app are condensing it to improve functionality (Dua, 2019). You could say that they are a mix between a native app and a website. Using the same technology as normal websites, but acts like an application. This provides easy development, using known technologies such as HTML, CSS and JavaScript to create an app. Cmpared to native development, you don't need any development kit to create a web app, they are quite straightforward and quick to build; However they don't offer the devices utilities in the same scale as a native app (Dua, 2019).

2.4.6 Summary of Technology Review

By gathering information about existing technology, I've gained some ideas for how to develop the application needed to finish the project. Furthermore, I've gained an understanding of the opportunities and limitations of the technology we have.

By finding the capabilities of the smartphone, it is very likely that it will fulfill the needs of running the application. By researching the different ways of how to develop an application for a smartphone I have better ideas on what to choose. While both web and native apps could work for this purpose, having the performance of a native app would be an advantage. One part of the project is to develop a tool for admins, managing users, cars and bookings. This could be hard to manage on a smartphone, therefore using a web app for this purpose would be the best solution.

By researching the different kind of servers, the best way to host a server would be in the cloud. A server from the university could have been given, but then a lot of time would have been spent on setup. Using a cloud hosting service, the server would be configured and ready to use immediately. Also using a cloud platform would provide more tools for the project with different API's, this will be described in chapter 3. For the database both SQL or NoSQL would be a good choice, but since we will have some relations between

users, cars and bookings, a relational databse such as SQL would be the smartest choice.

The findings in this technology research have given more insight into useful aspects when developing this service.

2.5 Designing an Enjoyable Application

People are engaging their smartphones more than ever before. The average person in the US uses 5 hours per day on a smartphone, and the majority of that time is spent on apps and websites (Perez, 2017). With this, consumers expect more and more of an application. The difference between a good and a bad application is usually the user experience(UX) (Babich, 2018). With the growth of application usage, people expect fast loading times, ease of use and feedbacks. If you want your app to be successful, the UX should be a major focus when developing (Babich, 2018). There are many things to consider when creating a design for a mobile application, in this section it's described the most important topics within UX design.

Reading through Google's best practices of UX design (Gove, 2019) gave a lot of good pointers on what to do and not to do when design UX. This is based on their material design which is widely used across different applications. It is also the standard template used in Android apps. Also reading through Apple's human interface guidelines helped a lot in defining what good UX is (Apple, 2019). Apple are known to create good user experiences with their design on iOS and OSX, so by reading through their guideleines, new ideas are gained on how to design an app.

2.5.1 Cluttering

One of the most important factors when creating a design is not to have too much clutter (Babich, 2018). By cluttering the interface, you give the users too much information, the more components, the more complicated the users think it is (Ketterman, 2019). People don't like clutter even on desktop apps and webpages, even though it is more accepted there, keep in mind that mobile phones have smaller screens. The main focus to avoid this problem is to keep the content to a minimum, only present the users with what they need (Babich, 2018). If you need a lot of data on a screen, then you can use a technique of progressive disclosure. This is wrapping information inside an element and by for example touching the element, the information will expand and by touching it again it will be disclosed (Babich, 2018). It's also important to reduce user input. For a lot of people it is not comfortable typing on a smartphone screen, and it's often quite error prone (Babich, 2018). Ways to reduce this is to use autocomplete when typing, and design inputs such as you write one input at a time. For example when registering at Facebook, you get one screen for your name. When your name is confirmed you get a new screen with email, and so on. This is a good design for both reducing the amount of inputs you get served, but also reducing the number of errors.

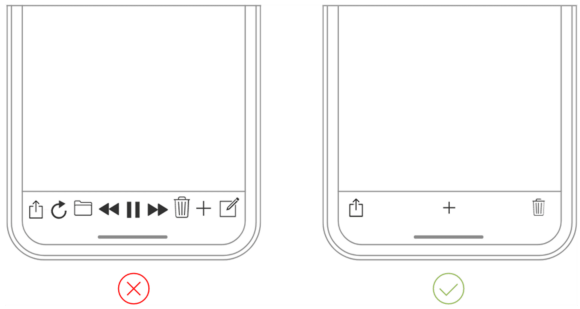


Figure 2.11: The clear tab bar (right) is much better than the cluttered one (left).
Source: Apple

2.5.2 Feature overload

A lot of apps has the tendency to add as many features as possible, because they try to cover all the needs of the users (Ketterman, 2019). This will in the end create less of an experience for the users. In a study they found out that the reason why old applications such as Microsoft Word don't have as many users as it had is because of feature overload (Olchówka, 2014). According to a psychological concept called the paradox of choice, the more choices we are given, the less happy we get (Olchówka, 2014). The best way to reduce this is to keep the features focused on the main objective of the app. Then you design the app in such a way that these features are both intuitive and enjoyable. (Ketterman, 2019)

2.5.3 Design for Touch

Users can find it uncomfortable using an app where the buttons don't respond when they are tapping on them. Error taps tend to happen because of too small touch targets (Babich, 2018). According to MIT's study when designing a touch target, the average size of finger pads are between 10-14mm and fingertips are 8-10mm, this makes a touch target with a minimum of 10x10mm good enough (Dandekar et al., 2003).

Figure 2.12: 10 by 10 mm is a good minimum touch target size.
Source: UXmag

2.5.4 UI Feedback

A well designed application will always in some way interact with its users. Lack of feedback can confuse users, and they will start to wonder if something in the app has happened after a tap. To solve this it is important to use different kinds of feedback based on the UI component the user are gonna interact with. These forms of feedback can be sound, visualizations, vibrations or haptics. (Ketterman, 2019)

2.5.5 Accessibility

According to (Ketterman, 2019) accessibility is probably one of the most overlooked aspects of UX design. In the world today about 15% of the population lives with some form of disability, and 2-4% of those experience significant difficulties in functioning (WHO, 2011). By having an accessible design you allow users of all abilities to use an application successfully. Always design an application such as people with hearing loss, vision loss and other dissabilities can use the app (Babich, 2018). W3C's content accessibility guidelines (W3C, 2019) is a free resource, and should be followed when creating a design.

2.5.6 Fast and Responsive Loading

Loading times are one of the most import aspects in making users come back to the app. People are getting more used to modern technology, and today 47% of users expects loading times in 2 seconds or less (Corona, 2019).

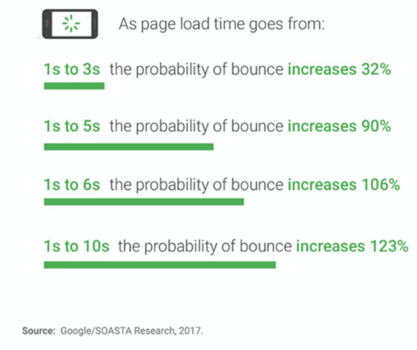


Figure 2.13: The faster your app, the better the experience will be.
Source: Google

As presented on figure 2.13, as loading time increases, the chance of users stopping to use the app increases. That's why, this should be a priority when developing an application. If the first thing that happens when a user loads the app is slow loading, this can affect the overall impression the user will have in the rest of the test as well. There are some factors that can cause slow loading times, such as bad internet connection, but these are not something the development can solve. But slow loading times caused by app performance should be avoided at all costs.

2.5.7 Legible Text

One of the most significant roles in design is its content. In most cases the reason why people use an app is because of the content it provides. But only presenting the content is not enough, we have to make sure that it's easy to digest (Babich, 2018). In order to provide a consistent UX across multiple mobile devices, we have to make sure we choose a sufficient typeface in multiple sizes and weights (Ketterman, 2019). A safe bet in this case is to use the default fonts the platform is providing. Apple uses the San Francisco font to provide a good reading experience (Apple, 2019), while Google uses Roboto and Noto fonts (Gove, 2019).

Using a legible font size is also important when presenting text. According to (Apple, 2019) you should at least use a font size of 11 points for a great reading experience. It also says that the text lines shouldn't be squeezed too tight and insufficient contrast makes the text blaned more with the background. You should always go for a contrast ratio of 4.5:1 for body and image text.

2.5.8 Summary of Design Theory

All of these topics within design theory will be important when designing a mobile application. There are many considerations we need to take into account to make the user experience as good as possible, and to make the users return to the app.

It is quite clear from the research that keeping an app minimalistic is the key. Furthermore, defining what the clear goal for the app is as important. Based on the research, mainly focusing on the goal and not a lot of extra features makes users more interested in using it. It's also important to remember that some users will have disabilities, and when designing we should have this in the back of our mind. Creating a user friendly experience on such a small screen is challenging. The key to manage this is to always think less is more and always consider what information you are putting on the screen and why.

In the end I've understood a lot of important principles about app design that will guide me through the development. It's also important to think that the design don't need to be flawless from the start, but improved through feedbacks from user testing.

Chapter 3

Methodology and Implementation

This chapter presents the methodology used for the development, and the research methods used to evaluate the system. Furthermore the architecture of the whole service will be presented and a briefing of the technology that will solve the service in the best way. Lastly the collection of data and how the experiment was executed will be described here.

3.1 Process

This section describes how both the research and the development process were done.

3.1.1 Development Process

The development process can be seen on figure 3.1. This is an iterative process, meaning that the development begins by specifying and implementing just a part of the software, then reviewed to identify further improvements. This process is then repeated, producing a new version of the software.

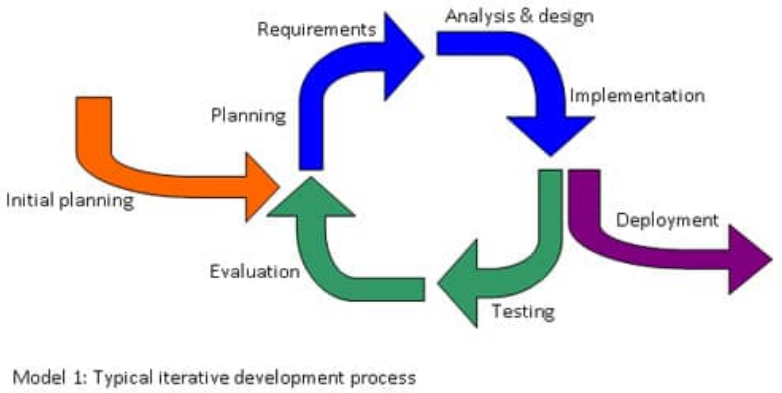


Figure 3.1: Illustration of the process developing an app.
Source: Testing Excellence

The process starts off with planning and requirements. Here the initial planning to map out the specification documents of the development project are done. Furthermore the software and hardware requirements are established.(Powell-Morse, 2016)

Once the planning and the requirements are complete, an analysis of the appropriate logic, database models and the overall architecture is done. The design stage also occurs here, defining the technical requirements which involves languages, services, etc that will be carried out in order to meet the requirements in the analysis stage.(Powell-Morse, 2016)

With the planning and analysis finished, the actual implementation of the software is carried out. All of the work done in the previous stages are now coded and implemented. For each time a build has been implemented, the next step is to go through some testing to identify and find potential bugs. Here the software was continually tested with 2-3 co-students while doing a bigger user test in the end. This is because of the time limits conducting the thesis. (Powell-Morse, 2016)

Lastly a thorough evaluation of the development up to this stage is conducted. This allows the development team and the users to examine where the project is at. At this development

process the evaluation are done based on the feedbacks from the testing. (Powell-Morse, 2016)

After all of these stages are carried out, the feedbacks from the evaluation process is brought back to the planning and requirement stage and the entire process is repeated.

3.1.2 Research Process

When defining the research process, the research model created by (Oates, 2006) was followed.

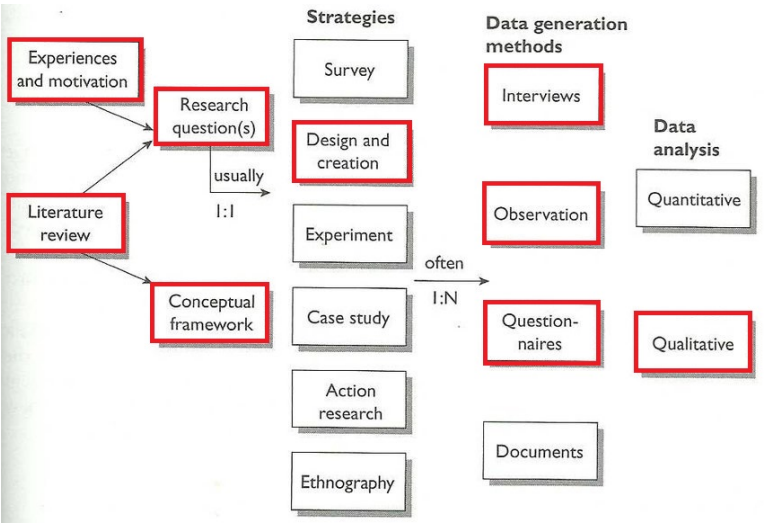


Figure 3.2: Stages used in Oates research model.
Source: (Oates, 2006)

Following this model, defining the research questions for the thesis was based on the motivation in chapter 1 and the literature review in chapter 2. Furthermore a conceptual framework was created in order to find out how to develop the system for doing the research, which is described more in section 3.2.

Next phase is the design and creation of the software which was done in the development process.

Lastly the software was evaluated through user testing with questionnaires, interviews and observation. This evaluation is both for defining the usability of the software but also topics regarding the research questions. The data analysis was mainly qualitative because regarding the research questions, impressions and opinions is a better way of measuring the research instead of raw numbers and statistics.

3.2 Development Overview

Autonomous vehicles as described in chapter 1 and 2 has a big potential in the society, and in order for the average user to use this technology, a tool for communicating between them is necessary. In this project it is facilitated through a mobile application. The idea is that people registers themselves in an app, and they will become members in a share pool of self-driving cars. Then, the users can see a map of their location as well as available cars, booked cars and the opportunity to book a ride (Mathisen, 2018).

This project is a part of the NTNU Autonomous Perception (NAP) project and focuses of developing the car booking software. This includes the app communicating between users and cars called NAP app, the NAP car app for receiving data from users and updating car position, a server, database and an app for admins. What's so important with this system, is that it's the only part of the NAP project where the users communicate with the cars. To book a car the user chooses the destination, either by typing the address or choosing with a marker, then the system will assign the best available car and calculate the route.

Throughout the development of the application, and some consultation with the projects supervisor Frank Lindseth, it was decided that this should be based on the project done in (Mathisen, 2018), but this time it should not be a simulation, but a software that is live and ready to use. Furthermore it was decided to start over again from start and redevelop it in order to make it compatible with both Android and iOS, because the previous project only worked on Android. This is why the development process was done in two iterations. The first iteration was to redevelop the app made by (Mathisen, 2018) and make sure it supports both Android and iOs. Then in iteration two, the main focus is to make sure this app can be used with multiple users, booking cars at the same time, and that it works on real cars.

3.3 Iteration 1

Following the model illustrated in figure 3.1 this section gives a description on how all the different stages was solved. The main focus for this iteration is to redevelop the application made by (Mathisen, 2018) and make sure it will be compatible with multiple devices.

3.3.1 Planning and Requirements

A lot of the project context and requirements were already given by the description and through consultation with the supervisor. Furthermore an analysis of the previous thesis done by (Mathisen, 2018) was conducted. But in order to fully understand the users for the application a persona was created. This is a fictional and at the same time realistic description of a target user of the app. The real value of it is to fully understand the goals and motivation of the people who will be using the app. This persona can be examined in appendix A.

Next up is to define the requirements. These are very much based on the requirements in (Mathisen, 2018) because as mentioned, this first iteration is a redevelopment of the same app. It covers both the functional and non-functional requirements, there has been some tweaks, removing those requirements that wasn't implemented by (Mathisen, 2018), since these can be added in the next iteration.

ID	Functional Requirement	Priority
FR1	The user should be able to see its own position on the map	High
FR2	The user should be able to see the position of available cars on the map	High
FR3	The user should be able to request a ride to a specified address	High
FR4	The user should be able to request a ride to a specied location on the map	High
FR5	The application should locate the nearest car to the user	High
FR6	The application should calculatea the best directions for the specified ride	High
FR7	The application should display directions and time estimates to the user	High
FR8	The user should be able to change the pickup location before booking	Medium
FR9	The user should be able to modify the drop-off location before booking	Medium
FR10	The user should be able to track the booked car	High
FR11	The application should distribute cars in a traffic and environmentally optimal way	Mediun

Table 3.1: Functional requirements of iteration 1

Some requirements were removed, these are editing drop-off and pickup after a booking, deciding pickup time and managing multiple bookings simultaneously. This is because these requirements weren't met in the previous app, but will be added in the next iteration.

ID	Non-functional Requirement	Priority
NFR1	The application should be easy to use	High
NFR2	The application should be able to run on most devices	High
NFR3	The application should have a simple and consistent design	High

Table 3.2: Non-functional requirements of iteration 1

The requirements for supporting several users simultaneously and displaying cars in real time were removed. Following these requirements it was easy to understand what was needed in order to finish iteration 1.

Following the requirements elaboration, a review of both software and hardware requirements were done. For hardware it was necessary having a laptop for both developing and running a test server. Also to be able to continuously run the app and test the code, both an Android and iOS device were necessary. The development could have been done through an emulator, but the performance using emulators are often much slower compared to using a real device. Furthermore this iteration was heavily influenced by the work done by (Mathisen, 2018).

3.3.2 Analysis and Design

At this stage it was chosen to first choose the development technology, then create the overall architecture and database models based on that.

3.3.2.1 React Native

React native is a framework for developing mobile apps for both Android and iOS by using JavaScript only. It is built upon the popular web library called ReactJS, both developed and backed by Facebook. In other words, developers who are comfortable developing for web can now develop native mobile apps. Normally, app development involves using Java for Android and Swift/Objective-C for iOS. With React Native, this requirement is removed, leading to fully native apps for both platforms developed in much less time and sharing the same codebase.

The use of the framework are steadily rising in popularity, with more and more larger companies starting to use it for their apps.

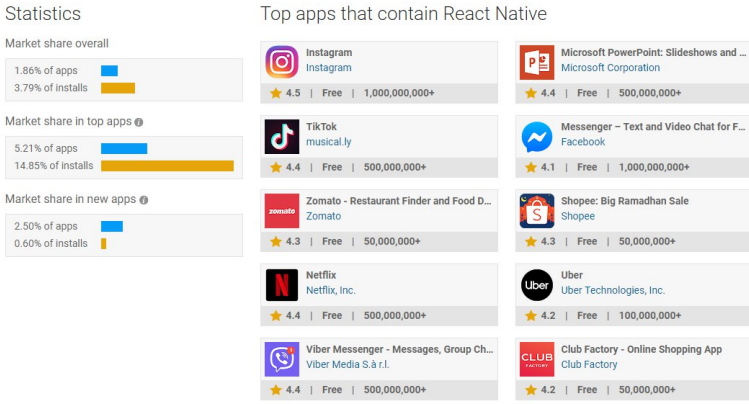


Figure 3.3: Apps using React Native from June 2018.
Source: Appbrain

As seen in figure 3.3, 15 percent of the top downloaded apps are coded in React Native. Also big apps such as Instagram and Netflix are using it, making it even more trustworthy. In this case, the best example is Uber using React Native and according to the research done in chapter 2, this app will probably be quite similar to Uber.

The performance of React Native is quite good, this was one of the biggest requirements before starting the development. It is able to achieve 60 frames per second because the logic runes on the JavaScript thread, separate from the main UI thread. This means that the application can achieve great performance. (Facebook, 2019)

The reasons why it was chosen to develop this in React Native is first, the previous app developed by (Mathisen, 2018) was developed in it and it performed quite well. Further-

more, using the same code base for two different platforms, faster development time can be achieved.

3.3.2.2 Expo

Expo SDK is a toolchain built around React Native in order to help you quickly get started with the development of an app. It provides tools for simplifying the development. In order to use hardware functionality, these methods are already implemented in Expo, providing a single line of code and you can for example use the GPS, Camera, notifications, etc. It also provides some services and user interfaces usually only available as third party packages. (Kruhlyk, 2019)

```
1 import { Location, Permissions } from 'expo';
2 Permissions.askAsync(Permissions.LOCATION);
3 Location.getCurrentPositionAsync({});
```

Listing 3.1: Code snipped from React Native using Expo

Showing in this code snipped, just a single line: Location.getCurrentPositionAsync is retrieving the coordinates from the device. This makes it easy to access the hardware on the device.

Other positive features in the kit is a client that runs in the webbrowser, showing app status, devices the app is running on, a QR code to open the app and switching between production/dev mode.
The app could be launched through an Expo app that is installed from Google Play or Apple store. It allows you to run the application without having it published on these app markets, proving quite useful for testing purposes.
It also provides over the air reloading of the app. This means changes in the code while the app is running will automatically reload the app.
This SDK was chosen because of previous experiences, knowing it works quite well on

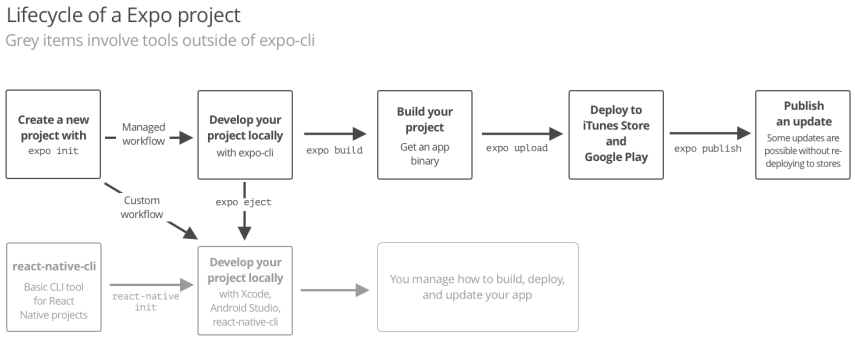


Figure 3.4: Workflow developing a react native app in Expo
Source: Expo docs

both platforms. All of these combined was the biggest reason for using this tool. However,

one big drawback is if you want to use a module that aren't supported by Expo, it will be unavailable until implemented. For example, background location fetching weren't added to Expo until February, making it impossible to do until then. But after further consultations with the supervisor, it was decided that the features missing was not gonna be a problem for the development.

3.3.2.3 Redux

Redux is a open-source library for managing application state. It is maintaining and updating the state of the apps components. It is important to consider if Redux is necessary for the app, because it can become confusing and a lot of boilerplate code is necessary for it to work. In React and React Native we have a vertical dataflow, this means that a parent component can send data to a child, but not to it's siblings. In React everything is a component, imagine how confusing it can get based on figure 3.5 if the parent wants to send data to child 6. Compared to Redux, where all components has the same data store, accessing the same state whenever they want. (Kapoor, 2019)

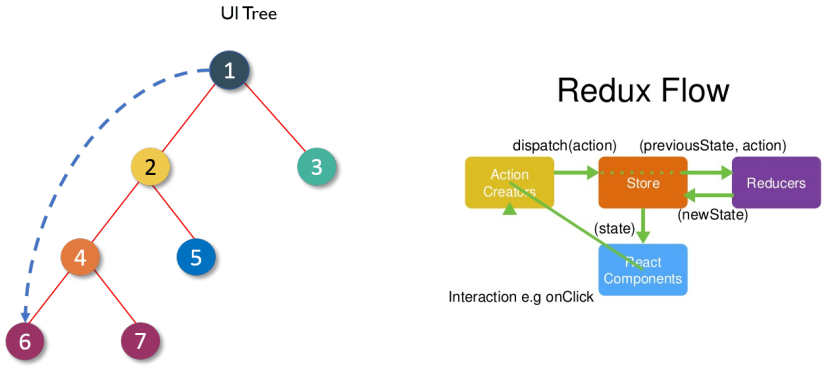


Figure 3.5: React component tree and Redux flow
Source: Quora

In order to understand Redux, there are a few concepts that need explaining. In this case, when booking a car, you get assigned a car for picking you up. So here the booking is an action, which is one of Redux's main concepts. After the booking is done, you wait for the assignment of a car, now there is a term called dispatch that sends the action to a reducer. The reducer looks at the action, determining what to do next. In this case finding the nearest car, then updating the global state and at last returning the state to the store. When the store is updated, the component is also updated.

Redux hasn't been necessary in this iteration, because the app is just a single screen. But thinking ahead, the app will have multiple screens with login and signup. The best solution is to use it from the start because it is quite problematic implementing it later on.

3.3.2.4 MySQL

In order to manage data in a database, a language for doing this is necessary. MySQL is an open source structured query language created by Oracle Corporation. It is a language that is used to manage data in relational databases. This means a database using relations(also called tables) to store and validate the data.
The SQL language was developed by Edgar F. Cobb and in his breakthrough paper, he describes it as a database where objects could be constructed and queried. SQL is meant to creath both data and the schema for that data, which describes fields in columns. A single record is called a row. (Shiff and Rowe, 2018)

It is possible to use MySQL with a lot of different back-end or front-end solutions. By for example running a local server, it is possible to set up a MySQL database. Also most of the big cloud hosting companies such as Amazon or Google has support for MySQL. (Oracle, 2018)

The reason why SQL was chosen over NoSQL which is described in chapter 2, is because SQL is a better choice for any system that will benefit from a predefined structure and set schemas. It's also the better option if all the data must be consistent without leaving room for error. Since there are gonna be real users in this system and have a relation with bookings, it is important that there are no errors. However it is possible to do all this in NoSQL, but SQL databases has proven themselves for over 40 years and use standards that are well defined and documented. (Shiff and Rowe, 2018)

3.3.2.5 Node with Express

Node
Node is an open-source, cross-platform, runtime environment that allows developers to create scalable network applications. These applications are running on the server-side, coded only in JavaScript. Node has a lot of benefits, one of them is great performance. It was designed to optimize throughput and scalability in web or mobile applications. (Node, 2019)
In the following example, many connections can be handled concurrently. For each connection a callback is fired, but if there is no work to be done, Node will sleep.

```
1  const http = require('http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
```

Listing 3.2: Hello World example in Node. Source: (Node, 2019)

This is in contrast to today's more common models where thread-based networking is relatively inefficient and difficult to use. (Node, 2019)
The Node package manager (NPM) provides access to a lot of useful packages. These brings extra features to the app, for example cryptation functions. (Mozilla, 2019a)

Node is quite versatile, it is available on Windows, Mac, Linux, Solaris, WebOS, etc. Furthermore it is well supported by many web hosting providers, they also often provide documentation for how to host Node apps on their services. (Mozilla, 2019a)

Express
Express is the most popular framework used with Node. It makes Node more minimalistic and easier to develop in, providing a set of robust features for web and mobile applications. (Express, 2019)
It also makes it easier to do requests to the server, by writing handlers with different HTTP requests at different url paths. For example by writing http://localhost/cars, you will receive a list of all cars, and http://localhost/rides gives a list over all registered rides in the database.
It also sets the settings for the web application, like the port used for connecting, and the database connection. (Mozilla, 2019a)

Express provides methods to specify what function is called for a particular HTTP verb(GET, POST, SET, etc.) and URL patterns(Route). It also has support for cookies, sessions and users, and you can use any kind of database technology that's supported by Node. (Mozilla, 2019a)
The next bit of code shows how much simplified the code in listing 3.2 is with Express.

```
1  var express = require('express');
2  var app = express();
3
4  app.get('/', function(req, res) {
5    res.send('Hello World!');
6  });
7
8  app.listen(3000, function() {
9    console.log('Example app listening on port 3000!');
10 });
```

Listing 3.3: Hello World example in Node and Express. Source: (Mozilla, 2019a)

The first require lines are importing the express module and creates an express application. The middle part shows a route definition. The app.get() method is invoked whenever a get request with the path '/' relative to the root site. After that a response with the string "Hello World" is transferred back to the client.
The final part launches the server on port 3000, logging text in the console if it is successfully launched.

The reason for why Node and Express was chosen is first, that it is coded in JavaScript. Using the same language as the client-side will reduce time spent with "content-shift" between different languages when writing both client and server-side code.

Another reason is that compared to PHP which was used in the previous project, Node proves to have better performance. Having support for multithreading, distributing a process between multiple threads (Chrzanowska, 2017).

3.3.2.6 REST API

First of all it is important to understand what an API is. An API (application Programming Interface) is used for different pieces of software to communicate with each other. Either internally in an app, or externally to connect to a service. The API is usually coded in a form of programming language, in this case JavaScript on the Node server. (Silva, 2019)

REST stands for REpresentational State Transfer and is used for manipulating data using several stateless operations. These operations use the HTTP protocol and are divided between these operations (Silva, 2019):

- POST (Create new source of data)
- GET (Retrieve data)
- PUT (Replaces a piece of data that currently exists)
- PATCH (Update or modify data)
- DELETE (Remove data)

By using the operations listed above, it is possible to create a pattern to easily understand the code rapidly and maintain it afterwards.
In the following code, when doing a request with url/car the REST API can see that it is a get function, knowing that it needs to get some piece of data from the database and send it back to the client.

```
1  app.route('/car')
2      .get(function (req, res, next) {
3          connection.query(
4              "SELECT * FROM `car`",
5              function (error, results, fields) {
6                  if (error) console.log(error);
7                  res.send(results);
8              }
9          )
10 });
```

Listing 3.4: Method in Node and Express, using REST pattern

3.3.2.7 Database Model

Before starting the implementation and setting up the system architecture, it is beneficial to have a model for the database. This model is based on the model made in (Mathisen, 2018) because the scope in this iteration is to make it work on the new back-end solution and on multiple mobile devices.

The first table is the car table, which stores all the information about the autonomous cars. It stores a unique ID, license plate number, coordinates, booked status and a foreign key linking it to the car model table.
The reason why an own table for the brand is implemented, is to reduce the number of duplicate records on the car brand, because multiple cars in the database could be off the same brand and model. It could be discussed to also split the model into an own table. This is a one to many relationship, the car can only have one brand, but the brand can be linked to many cars.

Next table is the ride table. This table stores records of all the times a user has booked a car. Like all other tables this has a unique id. Further, it has two foreign keys from both the car and user table, linking them together. It also stores different coordinates and timestamps from start, pickup and destination locations. One change here is adding a booked timestamp, having a record of when the booking happened. This table has a one to many to both car and user table. This means that in order to record a ride, it need one car and user, but the user and car can have many rides.

Lastly the user table stores records about the users. Storing email, password, name and the timestamp the user was created. The email here is set to unique, such that a user can't create multiple records. The created timestamp was added in order to know when the user registered.

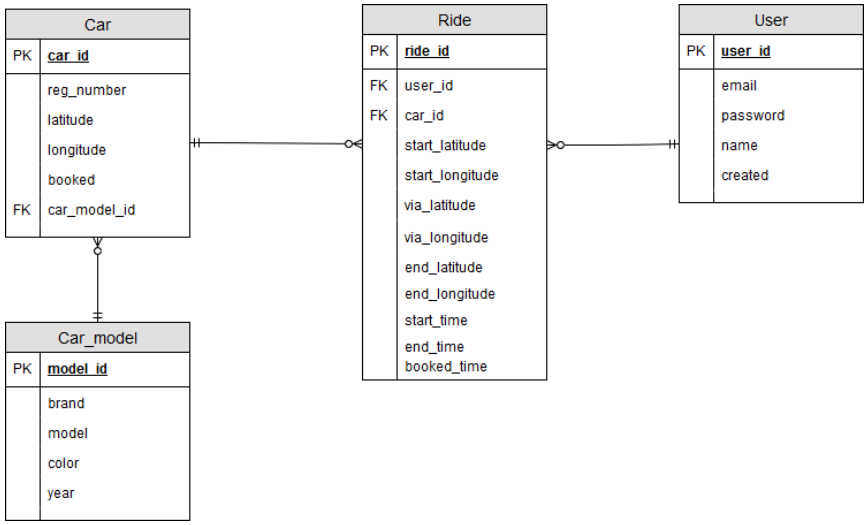


Figure 3.6: Entity relational diagram from iteration 1

3.3.2.8 Overall Architecture

The whole software is based on a three-tier client-server architecture, illustrated in figure 3.7. A three-tier client/server is a type of multi-tier computing architecture in which an entire application is distributed across three different computing layers or tiers. It divides the

presentation, application logic and data processing layers across client and server devices (Techopedia, 2019). The presentation layer in this case are the clients or mobile devices. This layer are responsible for visual presentation and interaction with the user, and allows the user to interact with the application logic layer. They are distributing data between each other based on what the user does. The application logic layer takes care of getting the right data from the data layer, based on what the user are requesting. and lastly the data layer are responsible for storing all the data, usually in a database. The reason for using three layers of data processing is to reduce the amount of processing and load in one single layer, distributing the work between three entities. (Techopedia, 2019)

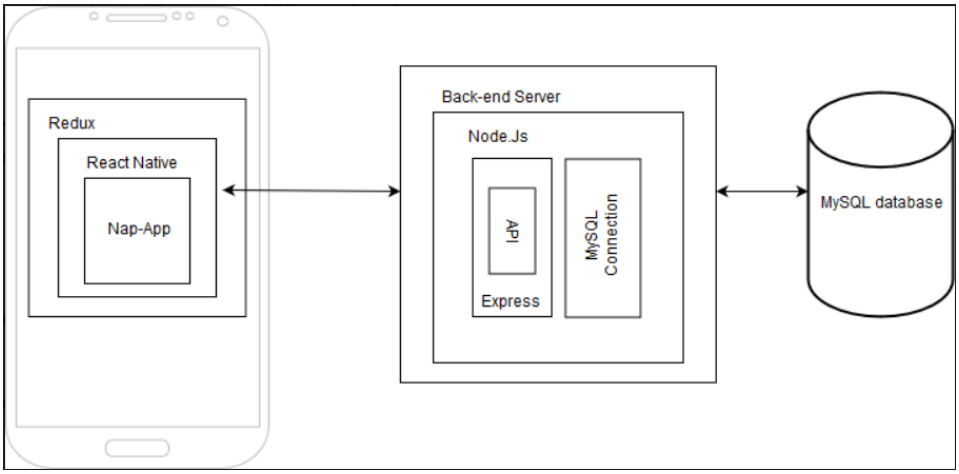


Figure 3.7: System architecture from iteration 1

In this case the presentation layer consists of a mobile device running either iOS or Android. The app is coded and compiled through the React-Native language, converting JavaScript code into native Java for Android and Objective-C for iOS. Furthermore the app is wrapped in a Redux store, managing all of the apps state. If the client needs to retrieve or update some data in the database, it needs to first send a request to the back-end server. The API running on the back-end server, coded in Node and Express checks what kind of request this is and based on the operation, it manages the data from the database and sends a response back to the client. The database contains all data about cars, users and rides.

3.3.2.9 Design

At this stage it was not conducted any design of a prototype. Having an agreement with the supervisor, the most important goal was to get it up and running.

3.3.3 Implementation

This section presents the implementation details for this iteration. Here the main focus was to get the previous app available on multiple devices and develop a new back-end solution

3.3.3.1 Server and Database

In this stage both the Node and database server were running locally on the development computer. First off, the database had to be created on the local machine. Then the different tables were created based on the database design, using MySQL. In the next example, it is displayed how to create a table. This was also done for the other tables in the entity relation model.

```
1 CREATE TABLE user(
2   user_id INT AUTO_INCREMENT,
3     email VARCHAR(50) NOT NULL UNIQUE,
4     password VARCHAR(20) NOT NULL,
5     name VARCHAR(50) NOT NULL,
6     created timestamp DEFAULT CURRENT_TIMESTAMP,
7     primary key(user_id)
8 );
```

Listing 3.5: SQL code for creating user table

The REST API implemented in Node and Express, has the responsibility to retrieve, modify, add or delete data in the database. This is an API where the clients can send HTTP requests to communicate with the database. The entire API can be checked out in Appendix B - NapApp-server. The main focus this time was retrieve a list of available cars and send it to the clients. Also creating bookings, marking cars as booked and retrieve a new list was implemented. Furthermore adding new records of rides was implemented. The code in listing 3.4 is the Node.Js code to retrieve all cars, when Node gets them from the database, they are sent back to the client.

3.3.3.2 NAPApp

In this first iteration, the main focus is to take the app implemented in (Mathisen, 2018), modify it to work with Expo which was explained in section 3.3.2.2. And the new back-end solution made with Node and Express. One of the main reasons for doing this is because the previous version of the app couldn't be tested on iOS devices, since it had to be compiled on a MAC computer. Also the SDK's used for developing this app weren't documented, making it difficult to find out which SDK to use when continuing the development. With Expo all of these challenges can be easily solved, making testing easy on any device and not needing any SDK. All of the features involving directions and location are still the same as the ones that were used in (Mathisen, 2018), using Google API's for geolocation and getting directions. It is recommended reading section 3.5.3 in that report to get a better understanding of those. After evaluating them it was decided that these solutions were good enough to use the map and find directions between the user and car.
Changing from PHP to Node in the back-end proved to be a better solution increasing performance and efficiency after doing some research.

Changing to Expo

To change the entire app into the Expo framework proved to have some difficulties. It was not enough to just copy the same source code used in (Mathisen, 2018) and add it into a new project. It was necessary to do refactoring on the code, for example using the React

Native Maps package that was used earlier are now included in Expo. Also React Native had some updates between the previous version and now, making some of the methods deprecated and had to use new ones. For example the next code snippet shows the code for getting the user location. This code was completely changed compared the previous Nap-App project.

```
export const fetchCurrentLocation = (onClick) => (
    async (dispatch: Function) => {
        dispatch(fetchLocationRequest);
        let {status} = await Permissions.askAsync(Permissions.LOCATION);
        if (status === 'granted') {
            let position = await Location.getCurrentPositionAsync({});
            if (position) {
                let region = {
                    latitude: position.coords.latitude,
                    longitude: position.coords.longitude,
                    latitudeDelta: 0.05,
                    longitudeDelta: 0.05,
                };
                let coordinates = {
                    latitude: position.coords.latitude,
                    longitude: position.coords.longitude,
                };
                if ((position.coords.latitude > 63.5 || position.coords.latitude <
                     63)
                    && (position.coords.longitude > 11 || position.coords.longitude <
                         9.5)) {
                    coordinates = {
                        latitude: 63.419567,
                        longitude: 10.401914,
                    };
                    region = {
                        latitude: 63.419567,
                        longitude: 10.401914,
                        latitudeDelta: 0.05,
                        longitudeDelta: 0.05,
                    };
                }
                if (onClick === true) {
                    await dispatch(setRegion(region))
                } else {
                    await dispatch(setCurrentInitialRegion(region))
                }
                await dispatch(fetchAddress(coordinates, 'current'));
            } else {
                dispatch(fetchLocationError);
            }
        } else {
            dispatch(fetchLocationError);
        }
    }
);
```

Listing 3.6: Code for getting user location

No documentation was conducted in the previous project, having to understand all the code was also quite challenging, but since a lot of code were changed, it didn't matter that much.

Also at the start of version 2, a lot of changes were done to Redux. This meant that refactoring of all the redux code from the previous project had to be done. Redux is quite hard to understand, so in this project, all of the Redux code were implemented from the ground up.

3.3.4 Testing and Evaluation

After the implementation was finished, a test with 4-5 co-students was conducted. It was not done any questionnaires measuring the functionality since the goal was to check if the app was working well on multiple mobile devices. It was necessary to find people with different brands and operative systems on their devices, to find out how the app performed with different screen sizes and architectures.
A more systematical test should have been done in order to document the results. The design and usability were already tested in the previous project, not being necessary at this time, but will be more important in the next iteration.

3.4 Iteration 2

As iteration 1 was finished and evaluated, the development of the next iteration could be performed. As the application works on both mobile platforms such as Android and iOS, it is time to make sure it works outside a testing environment. Making users have the opportunity to create user accounts, Being available for download and have an architecture that makes sure the app works live and with multiple users and cars simultaneously.

3.4.1 Planning and Requirements

As the typical users still are the same since iteration 1, the persona described in appendix A remains unchanged.

For the requirements specifications, the ones that were implemented in iteration 1 are now removed, and new requirements are added after further consultation with the projects supervisor. now, three different requirement specifications were made, because in this iteration a back-end server and an app for administrators are also developed.

3.4.1.1 Requirement specification - NAP App

ID	Functional Requirement	Priority
FR1	The user should be able to create an account	High
FR2	The user should be able to log in	High
FR3	The user should be able to log in using Google or LinkedIn	High
FR4	The user can't book a car unless signed in	Medium
FR5	The user should be able to change the pickup location after booking	Low
FR6	The user should be able to change the drop-off location after booking	Low
FR7	The application should display other booked cars	Medium
FR8	The user should be able to decide pickup time	Medium
FR9	The application should manage multiple bookings simultaneously	High
FR10	The application should support several users simultaneously	High
FR11	The application should display cars in real time	High
FR12	User should see all the previous bookings	Low

Table 3.3: Functional requirements - NAP App of iteration 2

As for the non-functional requirements, they stay pretty much the same because these are requirements that all the iterations needs. There are however added some additional requirements.

ID	Non-functional Requirement	Priority
NFR1	The application should be easy to use	High
NFR2	The application should be able to run on most device	High
NFR3	The application should have a simple and consistent design	High
NFR4	App data should be secured and encrypted	Medium
NFR5	It should be easy to modify the app	Low

Table 3.4: Non-functional requirements - NAP App of iteration 2

3.4.1.2 Requirement specification - NAP App server

In order to use the app outside testing environment, a server is required. This is because the server takes care of all the communication between the user and database. For example, if one person books a car, then the other users of the app needs to see that it's booked. If this is runned locally, they can't observe it. The server doesn't have any non-functional

ID	Functional Requirement	Priority
FR1	Server should be able to manipulate user data	High
FR2	Server should be able to manipulate car data	High
FR3	Server should be able to manipulate rides data	High
FR4	Communication with the server should be secure	High
FR5	Should be able to communicate with simultaneous users	High
FR6	User data such as password must be encrypted	High

Table 3.5: functional requirements - NAP App server of iteration 2

requirements, because at this stage the most important aspect is that it works. A non-functional requirement could be performance.

3.4.1.3 Requirement specification - NAP App admin

It was specified that the software also needs an administration tool. This is meant to be a tool for editing data, adding new cars, change user roles, etc. Not only a sufficient tool for administrators, but also for the development.

ID	Functional Requirement	Priority
FR1	Should be able to add user, and user role	High
FR2	Be able to add cars	High
FR3	Edit cars	Medium
FR4	Get an overview over registered cars	High
FR5	Get an overview over rides	Medium
FR6	Only user with admin role should be able to sign in	High

Table 3.6: functional requirements - NAP App admin of iteration 2

ID	Non-functional Requirement	Priority
NFR1	Should be available on all devices, but primarily PC	High
NFR2	Secure login	High
NFR3	Available from everywhere	Medium

Table 3.7: non-functional requirements - NAP App admin of iteration 2

As for iteration 2, these are the requirements set for all the different software pieces needed.

3.4.1.4 Software and Hardware Requirements

These requirements hasn't changed much since iteration 1. The biggest change is to set up a live server and database. Also finding a platform for hosting the app..
Since the software uses some API's from the Google cloud systems, it was decided in collaboration with the supervisor that probably the best solution is to use a cloud server in the same platform. This includes both hosting for the server code, and database server. This will be described further in section 3.4.2.

3.4.2 Analysis and Design

In this section a description about technology necessary for carrying out the new requirements are presented. Furthermore, it presents the changes done to the architecture and database models. Finally, an overview over design of new components and how the app was evaluated and tested are described.

3.4.2.1 Google Cloud Platform

In order for multiple devices to have access to the server, the REST API coded in Node and the database needs to be hosted somewhere devices can communicate with it. As described in chapter 2, it was chosen to use a cloud service for server hosting. The project's supervisor adviced to use this service for hosting because the other teams working with the NAP project are already using it. Also, since the project is already using API's for

location and direction fetching, having these in the same service with the servers will give a better overview, reducing communication latency between server and API.

Google's Cloud Platform provides a scalable and reliable infrastructure for computing and deploying web apps. It covers application, storage and computing services for both back-end, mobile and web applications. It is widely trusted and Google is using it for their own services such as YouTube (Cloud, 2019).
The whole platform consists of a physical hardware infrastructure which is computers, hard drives, solid state drives and networking. All of these are stationed in Google's data centers. The way this hardware is available to the customers is in the form of virtual machines, as an alternative to customers using their own physical infrastructure. What this platform is offering are software and hardware products provided as services the users choose depending on their needs. They are offering over 50 services in the categories of compute, storage and databases, networking, big data, machine learning, security and management tools. They can either be used independently or in a combination. (Meier, 2017)

All of the hardware distributing the different services are regional across multiple zones. They are spread across different zones that are isolated from each other to prevent the spreading of errors. By having a server in a zone close to the geographic domain of the users, reduces network latency between user and server. In the next figure it is illustrated all the zones distributed across the world. (Meier, 2017)

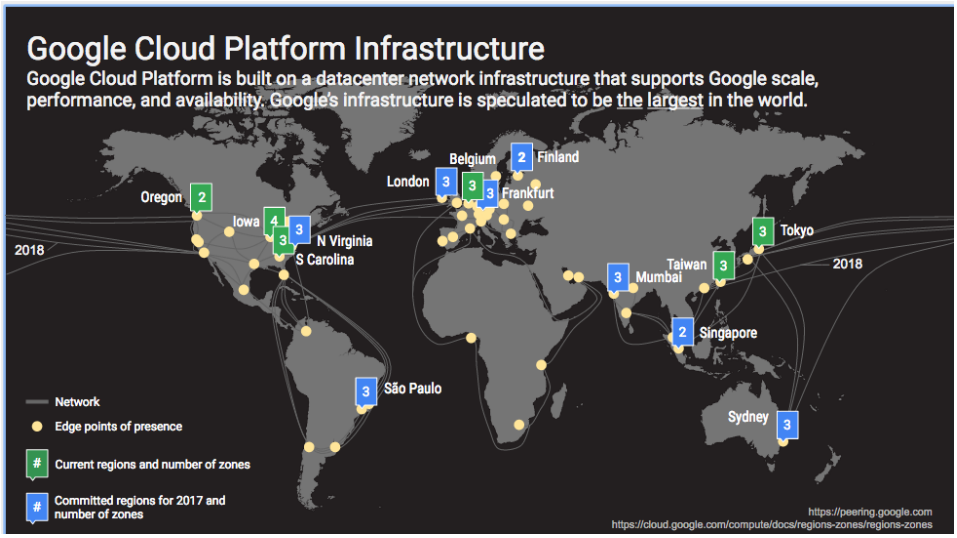


Figure 3.8: Different zones and regions in Google Cloud Platform
Source: Google

The services required for this project are the app engine, compute engine, cloud SQL and geolocation API's.

The app engine is a service for deploying server-side code on a Google virtual machine. This is a scalable solution because as the load on the server increases, it is always possible to increase the power of the hardware.

Compute engine is necessary for calculating the directions between different coordinates. As the client requests coordinates, the compute engine handles the calculation of the route.

Cloud SQL is the same as App engine, but an own virtual machine for hosting databases. Since the App engine and SQL server are hosted on the same hardware at a Google data center, the latency for the communication between the two instances is low. It will also be performed daily backups of all the data stored, making it a reliable hosting service if the data should be lost.

3.4.2.2 Vue Js

Vue is a web framework for building user interfaces. Compared to other frameworks such as React, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, this means the presentation layer that the user sees. It is also easy to learn and integrate with other libraries. It is a good tool to use with single page applications, which is web applications where the data are updating without needing to refresh the page. (Vue, 2019b)

Vue works with separating the HTML code and it's own javascript code, unlike React where it is combined. When a javascript object is passed as a data option to the vue instance, Vue will walk through all of its properties and convert them to getters/setters.
These are invisible to the user, but under the hood they change the view when properties are accessed or modified. Properties are all the data belonging to a view. Every component instance has a watcher instance, which observes any properties "touched" during the component's render as dependencies. When the dependency's setter is triggered, it notifies the watcher, resulting in the component to re-render, showing the new properties. In the next figure, the virtual DOM tree is the components shown to the user. (Vue, 2019a)

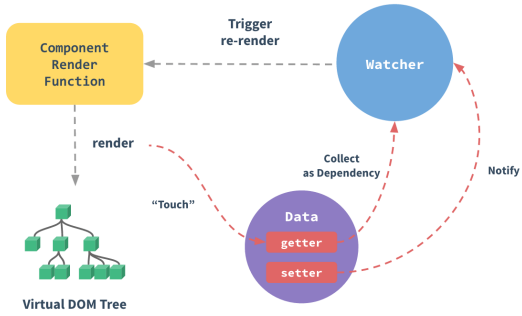


Figure 3.9: Vue reactivity model
Source: Vue

In other words, Vue is quite similar to React. They have the same purpose, it is a framework to code user interfaces in Javascript. The reason why Vue was used in this project is that a web app for administration tools is necessary. In web development, it is important to keep being updated on new frameworks and tools making the development easier and less time-consuming. According to Stack overflow global developer survey(StackOverflow, 2019), Vue is rising in popularity. Making it the most trending web framework. Despite being quite new, 15 percent of the total projects on Github are using it. Also being liked by 73 percent of the developers in the survey.

3.4.2.3 Bcrypt

Since user registration are implemented in this iteration it is important that passwords are not stored in plain text. In case the data should be leaked, anyone having access to it will know the different passwords of the users. The most common way to solve this is to store the password as a hashed random generated string like the below example
hash(”paul”) = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

What bcrypt features is salted hashing. This means first it creates random bytes(salt) and combines this with the password before hashing, creating unique hashes across each user's password. It also allows to choose the value of salt rounds, which gives control over the cost of processing the data. The higher the number is, the longer it takes for the machine to calculate the hash. Bcrypt is a robust solution and is widely used across modern applications (Paul, 2018). The following listing shows how it is used in code. First generating the salt with 10 rounds, then hashing input password with the salt.

```
1 const salt = await bcrypt.genSaltSync(10);
2 const hashedPassword = await bcrypt.hashSync(password, salt);
```

Listing 3.7: Bcrypt code example

3.4.2.4 Oauth2

In order to make the user log in using their Google or LinkedIn account, a famework called OAuth2 was implemented.

OAuth2 is an authorization framework that enables applications to obtain access to user information from different services such as Facebook, Google, Twitter, LinkedIn, etc. It's quite common that big services such as these provide support to OAuth2. It works by delegating user authentication to the service that hosts the user account. Third-party applications such as Nap-App need to be authorized by these in order to use it. It is possible to use OAuth2 with web, desktop, and mobile apps. (Anicas, 2014)

OAuth2 flow:

1. The application requests authorization to access resources from the user.

2. Application receives an authorization grand if the user accepted.

3. The application or server requests an access token from the service(Facebook, Google, etc) by presenting the login info to that service.

4. If the identity is recognized, the service sends an access token back to the application or server. This token is usually stored on the device or database

5. The application can now request data from the service by presenting the access token.

6. If the token is valid, the requested data are sent back to the application.

This flow can differ according to which service the data is requested from, but they all follow this same idea.

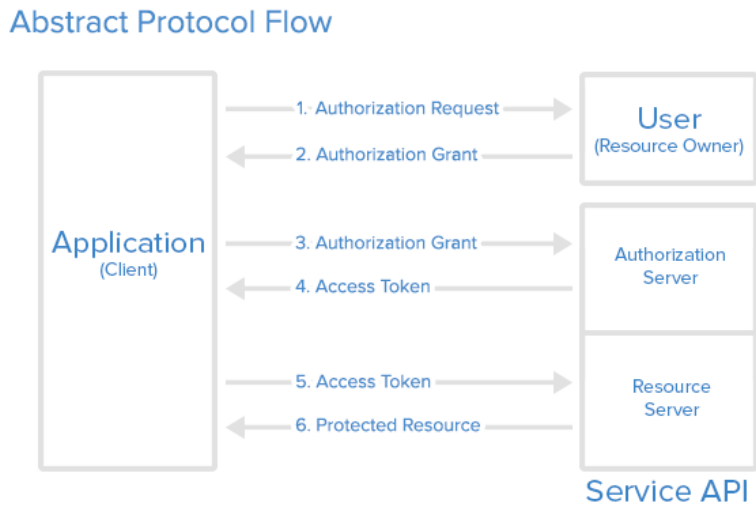


Figure 3.10: OAuth2 authorization flow
Source: Digital Ocean

OAuth2 proved to be the most efficient solution for login through Google and Linkedin. Based on the documentation on the framework and how easy it was to implement with Node and Express.

3.4.2.5 JSON web token

JSON Web Token (JWT) is an open standard that defines a way of securely transmitting information between parties as a JSON object. It can be considered as trusted because the token is digitally signed. This is done by signing it with a secret key, which is not accessible by other parties than the client itself. The whole token is then hashed into a random string, making each user have a unique token based on their id. (jwt.io, 2019)

It is often used for authorization. Each time a user has signed in, a request will include the JWT, allowing users to access resources that are only permitted with that token.
It is also a good way securely exchanging information between parties. Since it is signed, you can be sure the senders are who they are. Listing 3.8 shows how the token is created in the server. (jwt.io, 2019)

```
1 const createToken = function(auth) {
2     return jwt.sign({
3             id: auth.id,
4             role: auth.role
5         }, process.env.JWT_SECRET);
6 };
```

Listing 3.8: JWT signing example

Knowing that in this system, an administration panel are gonna be implemented. By adding the role to the JWT, the server can distinguish bnetween the different users.
Also since users need access to their booking history, we need to authorize with the server that this is a real user that is already signed in. The server then checks if this user has the privelages to get the requested rides.

3.4.2.6 Websockets

As the main goal in this iteration is to make the app work with multiple users booking cars simultaneously and to observe the car positions as they drive. The entire structure on the data requests had to be changed. In order to do this, WebSockets proved to be a possible solution to the problem.

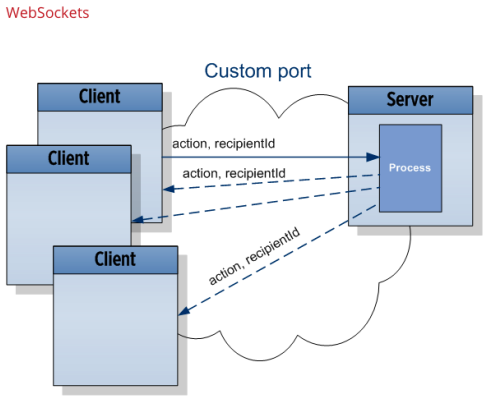


Figure 3.11: Illustration showing how WebSockets work.
Source: Medium

WebSockets is an advanced technology that makes it possible to open two-way interactive communication between a client and a server (Mozilla, 2019b). With this a client can

send messages to the server and receive responses without polling, i.e. without having to constantly check for a reply, causing unnecessary traffic. A good example is having an email service open and a new e-mail will automatically be visualized without any interaction from the client. It uses a long-held single TCP(Transmission control protocol) socket connection to be established between client and server, allowing for messages to be instantly distributed. This is done with minimal overhead such as a traditional HTTP request, resulting in low latency connection (Navada, 2018).
A request to a WebSocket connection is sent to the server from one or multiple clients through a process called handshake. If the server accepts the handshake, the WebSocket connection is established, allowing data to be transmitted in real-time back to the client. This connection is constantly open until terminated. This means if multiple clients are connected to the same WebSocket, and one of them is editing some data, the changes will be pushed back to the other clients.(Navada, 2018)
This will be useful for the user to see if a car is booked or not.

3.4.2.7 Database Model

In order to make the application work towards the requirement specifications, some changes were made with the database model. This model is based on the previous model created in iteration 1.

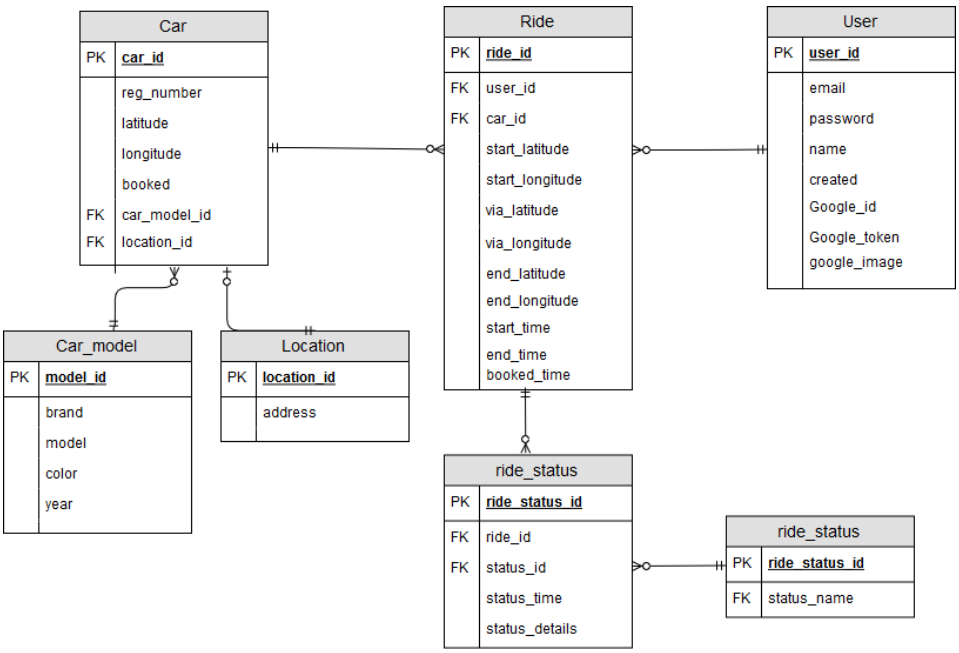


Figure 3.12: Entity relational diagram for iteration 2

In order for the administrator to see the different car positions, a new record of the address

was implemented. To avoid duplicate records of the address, the creation of a new table with a unique location id and address was necessary. Besides that, the car table remains the same.

Further on, the specifications indicated that users should have access to a history of previous rides. In order to solve this, the tables: timestamp and description were implemented. Having one mandatory to many optional between the ride table, the different statuses are divided into four strings: new ride, ride started, ride ended and ride canceled, which was stored in an own table, again to reduce duplicates in the ride status table.

Lastly, the changes done to the user table was because of the requirements for registration and sign in using a Google profile. The id of the Google profile, access token, and the profile image UTL is stored here. The LinkedIn service is working in a different way, not needing to store it in the database.

Further improvements to this database could be storing all the different timestamps and coordinates in an own table. It was decided to not structure it that way for now, because the differences between these numbers are big enough to keep them in their respective tables.

3.4.2.8 Overall Architecture

Keeping the same architecture as in section 3.3.2.8, figure 3.13 shows the integration with Google Cloud Platform. The client remains the same as there are no changes to it within the architecture. The server and database however are now hosted in Google Cloud Platform. The Node server is running through Google app engine on a virtual machine which are connected to a stationary server in Finland. The SQL server are also stationed here, making communication between the app engine and database fast and reliable.

This time, two new apps are gonna be developed. NapApp car is the app that is running inside the car. Its main task is to consistently broadcast the cars position to the database, in order for the users to observe where the car is at all times.
The other app is a web application, used for system administrators to edit data, and have an overview over available cars.

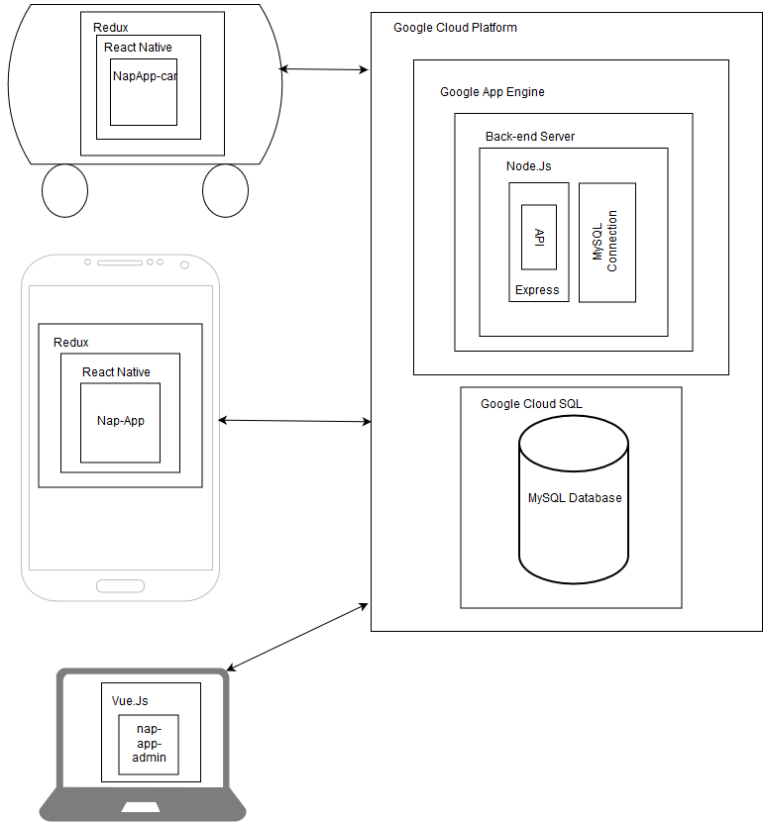


Figure 3.13: System architecture for iteration 2

3.4.3 Implementation

This section describes the implementation of the requirements presented in section 3.4.1. The main goal of this iteration is to make the system work on real users and applications. This means that the system should be functional enough for booking an autonomous car if it is available.

3.4.3.1 Navigation

In the previous iteration, only one screen was available to the user. Knowing that a registration and login page is necessary, some form for navigation needed to be implemented. This is not something that is already implemented in React Native or Expo. Reading the documentation of Expo(Expo, 2019), a package for React Native called React Navigation(React-Navigation, 2019) is recommended to use for app navigation.
It provides a wide variety of navigation methods. These can be everything from a component positioned at the bottom of the screen, where you can change screen according to

what you press. A Stacknavigator where the screens are divided into different layers, or a drawer navigator that animates into the screen from the side. Since this application does not need many screens it was decided to use the bottom navigator, choosing between map and profile screen.

This package works by implementing a routes file in JavaScript. Then importing the chosen screen components to the route's component and further decide where they are gonna be positioned. The example in 3.9 shows how to create the bottom navigator.

```
export const Routes = createBottomTabNavigator({
    Map: {
        screen: mapScreen,
        navigationOptions: {
            tabBarIcon: ({tintColor}) =>
            <FontAwesome name="map-o"
            size={20}
            color={tintColor}/>
        }
    },
    Profile: {
        screen: createStackNavigator({
            profile: {
                screen: LoginContainer,
                navigationOptions: {
                    headerTitle: 'Profile'
                }
            }
        }),
        navigationOptions: {
            tabBarIcon: ({tintColor}) =>
            <FontAwesome name="user"
            size={20}
            color={tintColor}/>
        }
    }
}, {
    lazy: false
});
```

Listing 3.9: React Navigation example

3.4.3.2 Implementing REST API and database on Google Cloud Platform

In order to use the Google Cloud Platform as the hosting service, the REST API and database had to be implemented on it. The database could be hosted by creating an SQL server through the console. Choosing different specifications for the virtual machine according to how much computing power needed. Since the database is just for testing purposes, a VM with only 600MB of memory and 10GB SSD was created. This is the lowest offer available, also the cheapest. As the load and data increases, it is always possible to increase the computing power and storage volume on the VM. It is also possible to increase the memory on the machine. The benefit of that is increasing the cache. This means that more data can be stored in the cache, resulting in faster loading of the data. Since the test database used locally used the same language as the SQL server in Google

Cloud, it was possible to upload the entire database. Keeping the same tables, relations, and data. An IP address is distributed to the user creating the database. This is used to connect to the database, either through a software client or from a web server. Keeping it secure, a validated user had to be created in order to connect through the IP address.

The REST API was uploaded on the Google App Engine, offering a new virtual machine for running the server code. Upon the creation of the VM, a new URL is distributed back to the developer, used for doing HTTP requests. All the requests are now secure, using the HTTPS protocol. Since both the app engine and database are running on the same Google Cloud account, they both communicate securely with each other. It was not possible to choose between different settings on the VM, but it will change the power according to how much processing the app does. Since all the different API's for location and directions already was tied to the same account, these remained unchanged. They will appear in the same overview of the transactions for billing.

3.4.3.3 Authentication

In the application, there are two ways of authentication. One is done by registering email and the desired password, the other one is used by either signing in with Google or LinkedIn credentials.

In order to sign in using email and password. A sign-up screen was created where the user first types their email, then writing the password two times. The data are then transferred to the server for validation. The server first checks if the user is already signed in, if the email already exists in the database, an error message will be responded back to the user. If the email does not exist in the database, the server is hashing the password with bcrypt explained in section 3.4.2.3. This makes sure that the password is not stored in plain text. If the user is successfully stored in the database, a response will be sent back to the user, confirming the registration.
The response, in this case, is a JSON web token. This is used both for the application to observe if the user is signed in, and validation towards server requests. The application is changing its state based on if the token is stored or not.
The database uses the same technology for signing in. When the user types their email and password, the server first decrypts the stored password, then checks if the strings match.

For signing in using Google or LinkedIn, the user first types in their credentials for the service chosen. If the authentication is successful, the client will receive an access token, like the example in section 3.4.2.4. The token is then sent to the server. If there is a user record with the token, a JSON web token is responded back. If there is no record, the server will create a new user, storing the access token.

3.4.3.4 Real-time Booking of Real cars

The most important goal for this iteration was to make it work in a real-life scenario, booking a "real" car. Different users should be able to see the car move around the map, and they should be able to book cars simultaneously, observing which cars are available or not.

In order to implement this, the server code had to be changed from iteration 1. All of the data requests from the client were done with HTTP requests, making real-time updates impossible.

In order to make the cars broadcast their positions to the users, a new application was developed. This is the NapApp-Car application, which is an app that runs on an iPad inside the car. The task for this application is to update the database with its location as it drives. It is done by for every 10 seconds, the car sends its coordinates to the server. The server observes which car it is by checking the car id from the request, and stores this in the respective car table. When a car updates its position, the server distributes the cars to the clients with web sockets, making the cars update in real-time. The app inside the car also shows which user has booked the respective car.

Web sockets are also used when the user's book cars. When a booking occurs, other users will observe that one of the cars on the map are booked. When a ride has finished, the other users will see that the car is available again.

3.4.3.5 Administration Tool

In order for the system administrator to have an overview of the data, the requirement specification required a tool for that. In order to find out how this was gonna be implemented, a meeting with the supervisor was conducted. It came to an agreement that a web-application was probably the best solution for this. The administrator is most probably gonna use a computer for this since managing a lot of data on a mobile device could be problematic. To carry this out, it was developed in a framework called Vue.js, explained in section 3.4.2.2.

Vue proved to be a quite efficient tool for developing the web-application. Only using about a month developing it, and it was easy for the developer to learn. It has some features which are similar to React, but the state manager is already implemented. This means that it was not necessary to import an external library such as Redux in order to manage the app's state. Since the state manager is built in the framework, it has much more seamless integration with the overall development. The code for this app can be reviewed under Appendix B - Admin app 6.2.6

3.4.4 Testing and Evaluation

In order to evaluate the app, a user test was conducted in order to test the functionality of the application, the usability and questions regarding the users trust in autonomous cars. The test was mainly qualitative, making the user do a set of tasks while the interviewer observed for possible problems with the UI. After completion of the tasks, the user answered a questionnaire regarding the usability and their experience using the app.
Another test of the app was conducted in order to find out if it would work on a real car. This was not a user test, but an alpha test of the overall system.

3.4.4.1 User Test

In order to test the functionality and usability of the app, a test including both a set of tasks and a questionnaire was conducted. The questionnaire also includes questions about the participant's thoughts about the product and their opinions and reasons for why they either trust or do not trust autonomous cars.

When choosing user samples for a qualitative test, (john W. Creswell, 1998) recommends between 5-25 participants and (Morse, 1994) recommends at least 6 participants. A qualitative test was chosen in order to get a more detailed understanding of the usability of the app. This test also included an interview about the research questions. According to (Morse, 1994), most problems can be found with just six participants, as they tend to find the same faults or errors.

In order to find the usability of the application, the System Usability Scale(SUS) was used. It is a simple and easy way to measure the usability of a system. Using ten questions, where each question has a weight between 1 to 5. 1 represents strongly disagree and 5 represents strongly agree. Participants can choose whichever number between 1-5 as they please. It allows testing a wide variety of products, including both hardware and software (usability.gov, 2019).
The total score is calculated by, for each odd-numbered questions, subtract 1 from the score. For each even-numbered question, subtract their value from 5. Take these new values which you have found, and add up to a total score. In the end, you multiply this by 2.5. This will result in a score from 1 to 100 (Thomas, 2015). Having a SUS score above 68 would be considered above average.(usability.gov, 2019)

The test was done by arranging a meeting with the participants. They were presented with what the purpose of the app was, and a sheet that told what tasks they should do. The interviewer observed what they were doing and made notes along the way. When the participants completed the tasks, they were asked if they could answer any questions. These included their experiences with ADAS and autonomous vehicles, whether they rely on autonomous vehicles or not, reasons why they trust/do not trust it, SUS tests, product issues, and finally any features that could affect their trust in self-driving cars. The questionnaire can be found in Appendix E - User test questionnaire.

3.4.4.2 Alpha Test

The main idea is that users should be able to book real cars using the Nap app. In order to evaluate this, an alpha test with two volunteers was conducted. One of the testers was simulating a car by running the Nap car app. The other two acted as standard users, wanting to get picked up and travel somewhere. Since we didn't have access to a real car, the test was done in the city center of Trondheim, reducing the distances between the testers.

Both the apps were launched through the Expo client app and had a connection to the server, running in Google Cloud Platform. The test was carried out by the volunteers communicating by telephone. The users should book a trip in the city center of Trondheim.

Then the car must be informed that it is booked and that it should pick up a person. The person who simulated the car should then follow the route to the person and the user should then see that the ”car” is moving towards him. When the car arrives at the user, he must confirm that the car has arrived.

This test was designed to find out if the app can work with real users and cars. For it to work, it is important that all data is sent to each endpoint and that the app is updated in real-time

Chapter 4

Results

This chapter presents the results of the whole project. The first part focuses on the results of the development. This is done by measuring the app against the requirement specifications presented in Chapter 3. The second part presents the results from both the user and alpha tests.

4.1 iteration 1

As there was not conducted any advanced user tests in iteration 1, the app was tested on a set of different devices. This was done in order to determine if it is functional across different platforms. It was performed a small user test with 4-5 co-students, evaluating if the app was functional and also getting some feedback on design improvements. The previous application was used as a prototype under the development of version 2. The most important goal was to make it work across platforms and using a new backend solution. A presentation on how the requirement specifications were solved in the app is also presented here.

4.1.1 Solution for Requirement Specification

This section will present how different requirement specifications were solved. Different screens from the first iteration will be linked up to the requirements. Keep in mind that some of these screenshots do have a navigation bar at the bottom. This was not implemented in iteration 1, but lacking screenshots from it, some are from the early stages of iteration 2. However, the functionalities presented are still the ones from the first iteration.

Map Screen

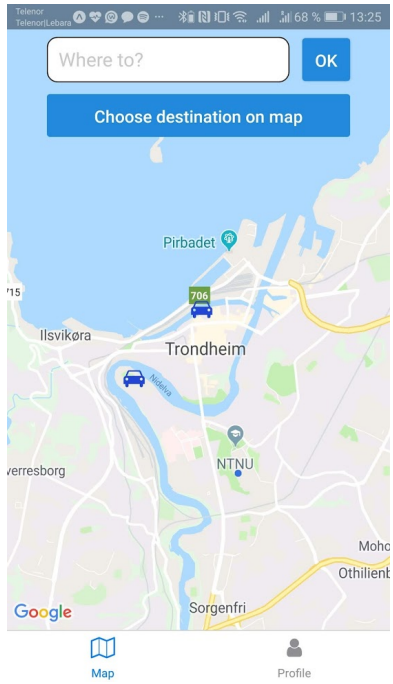


Figure 4.1: Map screen from iteration 1

ID	Functional Requirement	Priority
FR1	The user should be able to see its own position on the map	High
FR2	The user should be able to see the position of available cars on the map	High
FR11	The application should distribute cars in a traffic and environmentally optimal way	Medium

Table 4.1: Solved requirements for figure 4.1

This is the start screen from iteration 1. First off, the users can observe their position with the small blue dot under NTNU. All the available cars that can be booked are presented as blue cars. If they are not available, they will not be presented. All the cars are distributed by the system administrator by typing an address. If the address is not present, the car will not be stored in the database.

Choosing Address

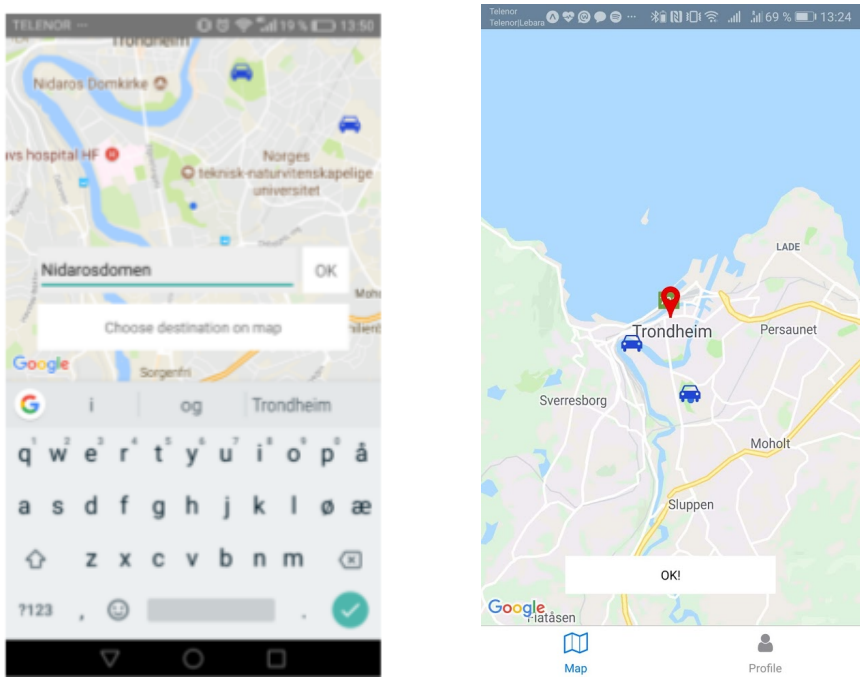


Figure 4.2: Address and pointer screen from iteration 1

ID	Functional Requirement	Priority
FR3	The user should be able to request a ride to a specied address	High
FR4	The user should be able to request a ride to a specied location on the map	High

Table 4.2: Solved requirements for figure 4.2

These requirements were solved by adding a textbox where the user types in the desired address. The user also had the opportunity to press the ”Choose location on map” button in order to see a pointer, where the user could click on the desired pickup or destination point. Both these solutions were used for choosing pickup and destination.

Booking Screen

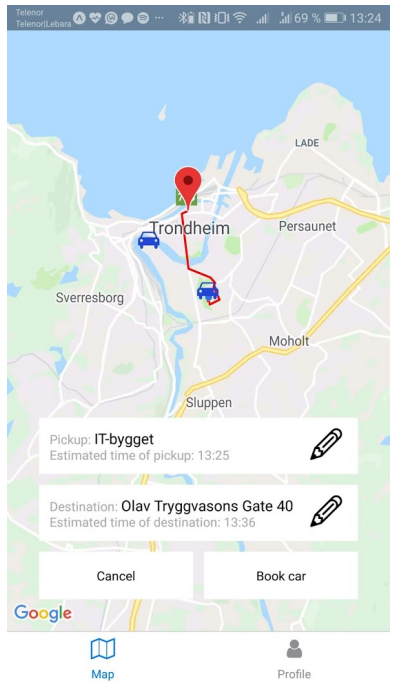


Figure 4.3: Booking screen from iteration 1

ID	Functional Requirement	Priority
FR5	The application should locate the nearest car to the user	High
FR6	The application should calculate the best directions for the specied ride	High
FR7	The application should display directions and time estimates to the user	High
FR8	The user should be able to change the pickup location before booking	Medium
FR9	The user should be able to modify the drop-off location before booking	Medium

Table 4.3: Solved requirements for figure 4.3

As the user chooses pickup and destination address illustrated in 4.2, the application will locate the nearest available car. This is done by calculating the distance from pickup location and the cars using the Google API for location. The app also shows the directions in the form of a red line, linking the car, pickup, and destination such that the user can observe the travel route. It is trusted that Google's API for directions is reliable enough to calculate the best route, as Google Maps is trusted among people. Using this API, the app also receives the time estimate for a car to travel between these distances. It is presented to the user together with the address, resulting in a complete booking overview. Lastly, by pressing the pencil button it is possible to edit both pickup and destination address. If the user presses these, the screen will be redirected to the address screen shown in figure

4.2. The reason for using the pencil button is because it is often recognized for editing information.

Car Tracking

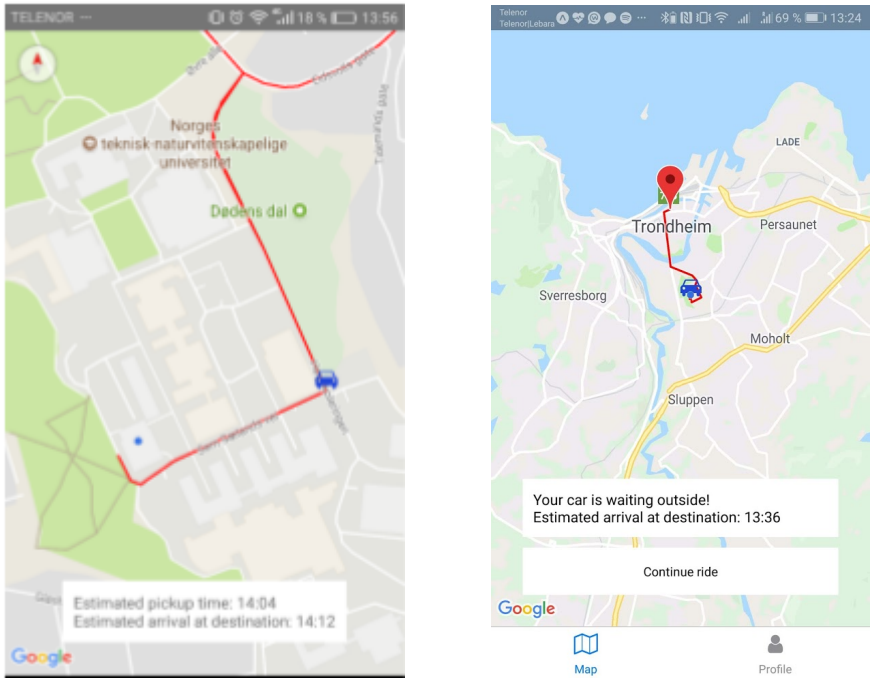


Figure 4.4: Screenshots showing how the user can track the booked car

ID	Functional Requirement	Priority
FR10	The user should be able to track the booked car	High

Table 4.4: Solved requirements for figure 4.4

Different screens are shown to the user depending on where the car is. If the car is driving, the left screen on figure 4.4 are shown. When the car arrives, the user can confirm that by pressing the ”Continue ride” button. Afterward, a similar screen like the first one is presented, showing the estimated time to destination. When the car arrives at the destination, the user confirms it by pressing ”Finish Ride”. This is just a simulation of the whole process. It is just a car that animates to the user location inside the app. This is not used in a real car.

4.1.2 Device Test

The redeveloped application was tested on a set of different devices, having different screen sizes and operating systems. Some co-students were asked if they wanted to cooperate by testing the app on their devices. The developer was also allowed by an electronics store to test the app on their demo phones. Since the Expo framework was used for the development, the testers could easily download an app called Expo client on their phones. With this app, they could sign in to the developer account used for this project and launch the app. Table 4.5 shows all the devices that managed to run the application without any problems.

Phone	Number	OS	Status
Apple iPhone X	1	iOS 12	Runs
Apple Apple iPhone XS	2	iOS 12	Runs
Huawei Mate 9 pro	1	Android 8	Runs
Samsung Galaxy S8	1	Android 8	Runs
Huawei p20 pro	3	Android 8	Runs
Sony Xperia CZ2 Compact	1	Android 8	Runs
Apple iPhone 7	1	iOS 11	Runs

Table 4.5: Devices the app was tested on

By doing this test, it was clear that Expo was a reliable framework for cross-platform testing. Proving useful for future development. Figure 4.5 Shows the app running on both and Android and iOS devices.

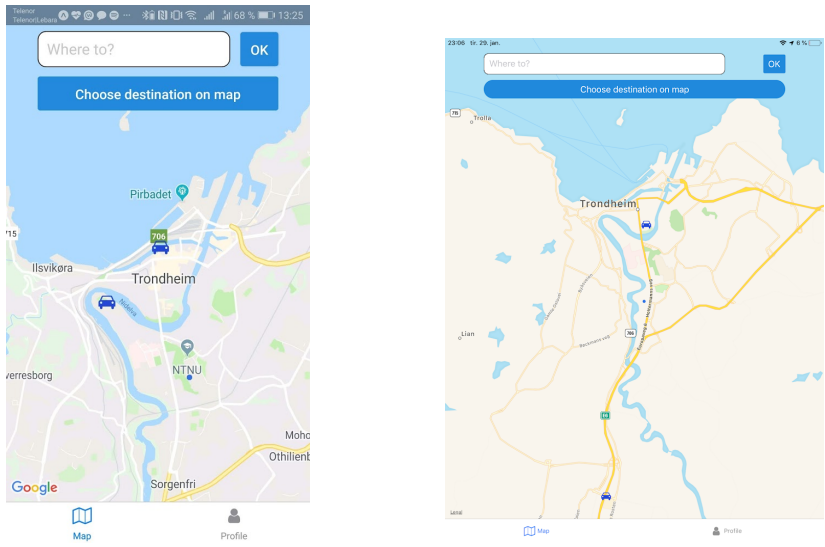


Figure 4.5: Screenshots from both an Android and iOS device

4.1.3 Evaluation

As the app in this iteration was just tested on 4-5 users without doing any questionnaire, there are no results. However, Some oral feedbacks were given on the functionality of the app.

The users were all able to book a car. Saying that the application was easy to use, not needing to do a lot of button presses before doing their bookings. They had a good overview of all the available cars and had no struggle finding out how to book it. That said, the users conducting this test were all students of the same institute, having a background in computer science. This probably makes them familiar with such products. However, some feedbacks were given for design improvements.

All of the users mentioned that the current location marker was barely visible. As it had a fixed height, users with big screens struggled to identify it. Some of the users also mentioned that a button to set your current location could be a useful feature.

Some users had some problems with typing either pickup or destination address, as the keyboard appeared over the input box. This made it impossible to observe if they typed the wrong or correct address. A suggestion to resolve this was to make the input box animate to the top of the screen while typing. Also using Google places for suggestions of addresses as the user types.

Two of the users would like to see all cars, not only the ones available. They recommended using different colors for booked and available cars. The reason for this was to get an impression of how many cars could be booked in the future, and to evaluate if the product is a viable option when the traffic is busy.
Lastly, there was some feedback that the buttons did not look like buttons, but just squared components. This made them unsure if they were just for information or pressable.

4.2 Iteration 2

This part presents the results from the second iteration of the development. It includes an overview of improvements done based on the feedback from iteration 1. A description of how the different requirement specifications related to iteration 2 was solved. This is done by showing different screenshots from the app after iteration 2, and which requirements they represent. Lastly, a presentation of the results from the evaluation. These are both the results from the user tests and the alpha tests. They are both done in collaboration with different users. It was not done any device test at this time, as the app was running fine on different devices from the previous iteration.

4.2.1 Design Improvements

Based on the feedback received in section 4.1.3, some changes were made to the app. These are mainly design changes as they do not have anything to do with the requirement specifications. These are improvements done to the features which were already implemented and described in section 4.1.1. The screenshots presented here are taken from an iPad 2018 version and Huawei Mate 9 Pro.

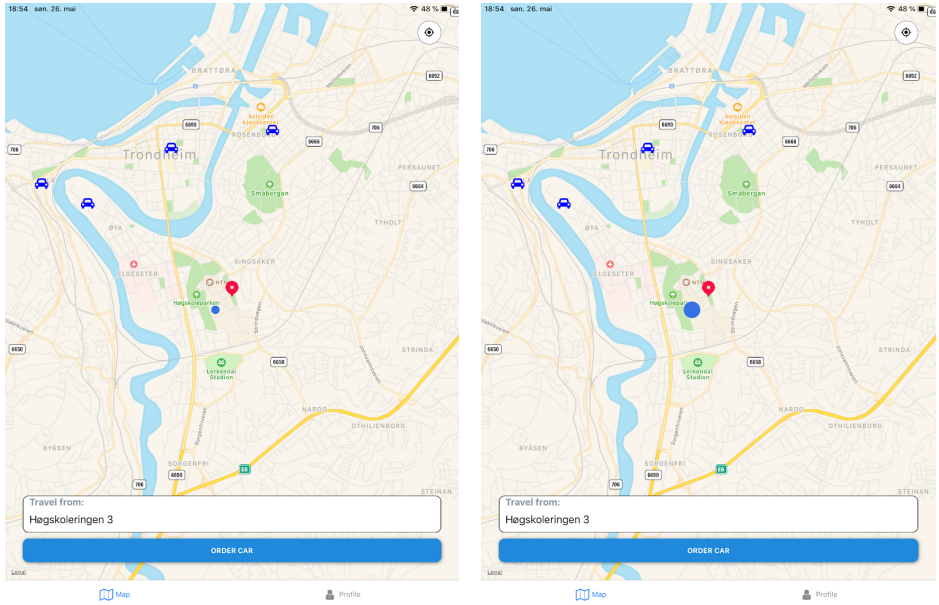


Figure 4.6: Map screen from iteration 2

First off, an animation was added to the blue dot showing the users location. The location marker is always getting bigger and smaller. This is done with the benefit of making the user notice that something is moving. People will automatically pay more attention to a moving object, reducing confusion about where they are. If the user chooses to press on

the blue dot, it will tell that it is their position.

Another big change is that the map automatically chooses a pickup location with a marker. Not needing to press a button before doing it. From testing the app in iteration 1, everyone decided to choose pickup with the marker. By making it default, one extra touch and feature were reduced. The user can, however, choose to enter a pickup address by pressing the text input field.

A border was added to the text input field. In the previous iteration, it looked like a ”hole” in the map component. Now it is clearly observed as an overlay component to the map. By adding a small border and some shadowing to the button, it is more clear that it is something that the user can press.

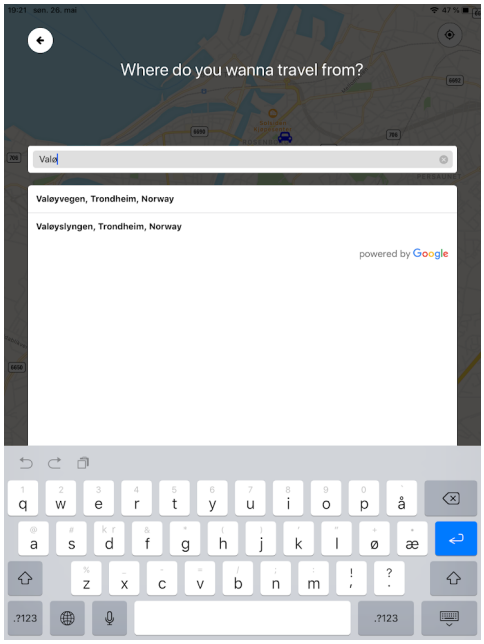


Figure 4.7: Address input screen from iteration 2

To solve the problem where the keyboard overlayed the text input from iteration 1, a modal will now be visible if text input is chosen. When the user starts to type the address, different suggestions will appear based on what the user is typing. These suggestions are fetched from the Google locations API. When one of the suggestions is pressed, a data object will be fetched making the map move to the desired pickup address. This feature is also used when entering the destination address, with the intention of preserving a consistent design.

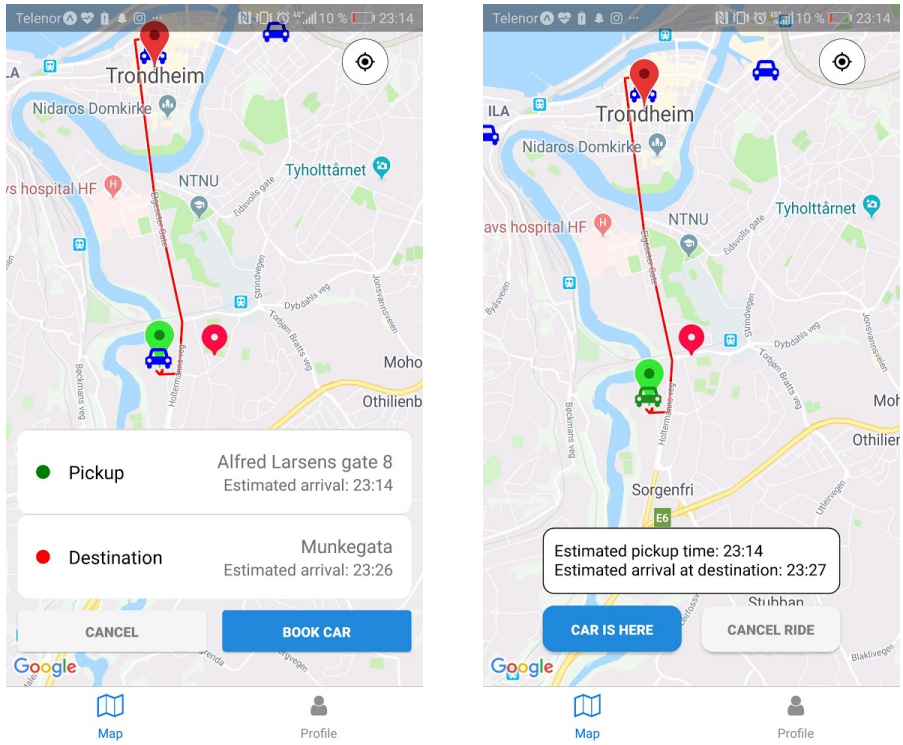


Figure 4.8: Car booking design changes

Further improvements were done the booking overview and the tracking screen. The information boxes have better contrast, showing a dot with a color respective to the color of the marker for either pick up or destination. Different colors were used on the buttons, as the blue color resembles a confirmation, and the cancel button is designed with grey in order to indicate that it is canceling an operation. It is also implemented some shadows on the buttons, as this helps with indicating that it is a button.
The tracking screen now has a border around the information box, Making it more noticeable.

The car that the respective user has booked is now changed to a green color. Cars that another user has booked are marked with red. This is done in order to distinguish between the different statuses of the cars. Furthermore, animations were added as the different components appear.

In order to navigate between different screens, a navigator component was added to the bottom of the screen. This is a familiar design that many people are used to as a lot of known apps are using it. This is a viable solution when having a small number of screens, as they appear more clearly.

4.2.2 Solution for Requirement Specification

This part presents the solved requirement specifications. A screenshot from the application is presented along with the respective requirements it solved. The solutions for both the Nap app and administration tool are described.

4.2.2.1 Requirement Solutions for Nap App

Registration and Signing in

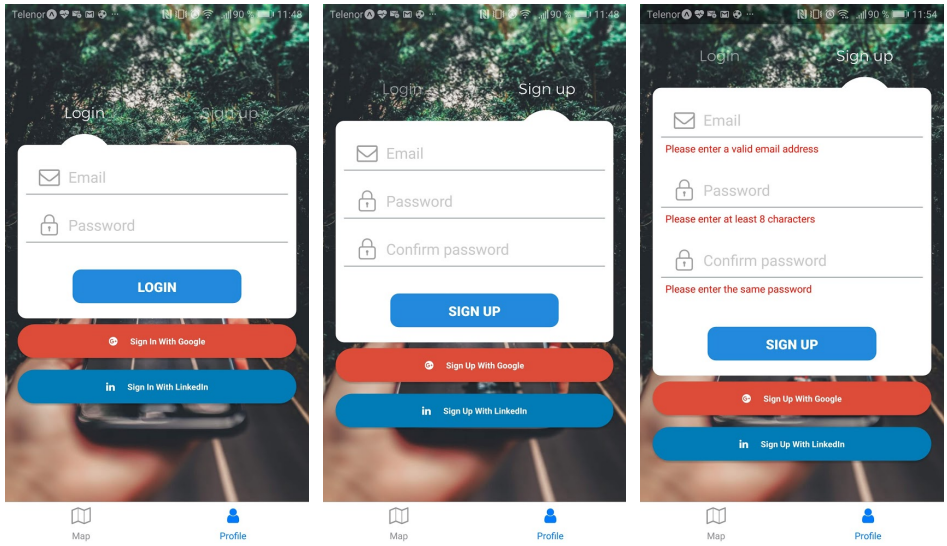


Figure 4.9: Login and Sign up screens.

ID	Functional Requirement	Priority
FR1	The user should be able to create an account	High
FR2	The user should be able to log in	High
FR3	The user should be able to log in using Google or LinkedIn	High

Table 4.6: Solved requirements for figure 4.9

The implementation of a new screen called profile allows the user to either sign up or sign in to the app. This is all done in one single screen as the user does not need to navigate through different screens, making it less cumbersome. By pressing one of the tabs over the white information box, the screen will change with an animation depending if the user wants to sign in or sign up. If there is some validation error with the server, error messages will appear under the input where the error was registered. This whole process is done with an HTTP request to the server, sending the information the user typed. If the server

validates the authorization, a JSON web token will be sent back to the user.

If the user wants to sign in or register through Google or LinkedIn, there are two more buttons under the input box. By pressing these, the user first needs to give the app permission to use OAuth, then they will be redirected to new screens where they sign in using either Google or LinkedIn credentials. These are illustrated in figure 4.10. The same process happens here, if the validations are successful, a JSON web token will be sent back to the user. If the app has received the token, the profile screen will change.

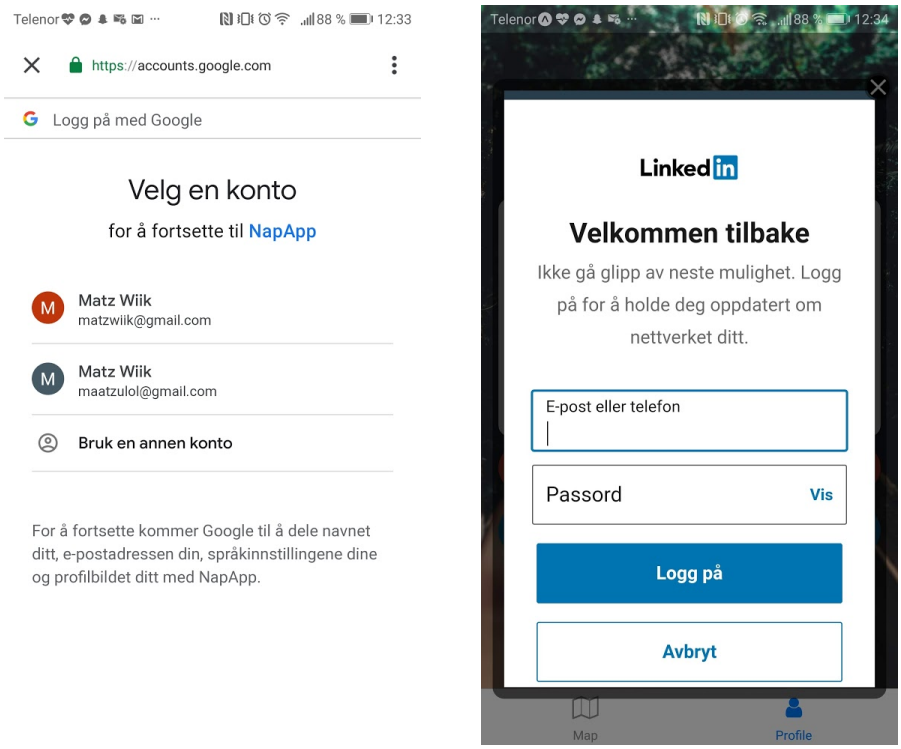


Figure 4.10: Sign in details either using Google or LinkedIn

User Status and Booked Cars

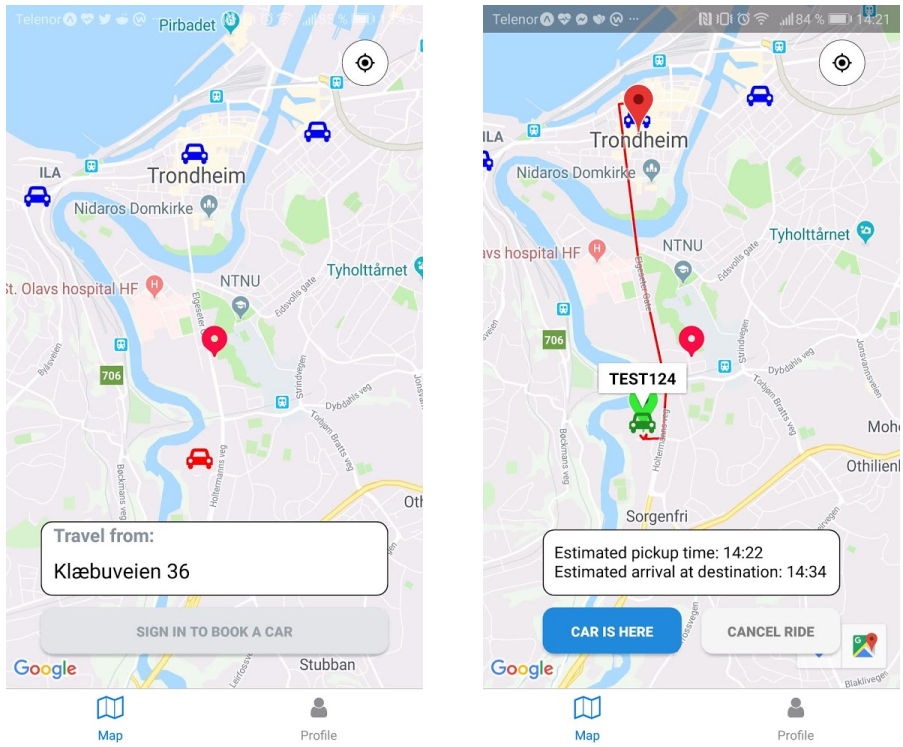


Figure 4.11: Map status

ID	Functional Requirement	Priority
FR4	The user can't book a car unless signed in	Medium
FR7	The application should display other booked cars	Medium

Table 4.7: Solved requirements for figure 4.11

If the user has not signed in, the application will not have stored a JSON web token. When launching the app, it will check for the token in the app's state. If not available, the book car button will be disabled, telling the user to sign in if they want to perform this action. The user can still navigate through the map and check where the cars are driving.

The blue cars are the available ones, the red cars are the ones booked by other users and a green car is the one booked by the signed-in user. If a red car is the closest to the user and he/she chooses to perform a booking, the distance will be calculated to the nearest blue car. If the user signs in and the red car in the left image on figure 4.11 is the one booked, the directions will show up and the car will turn green.

Simultaneous bookings with multiple users

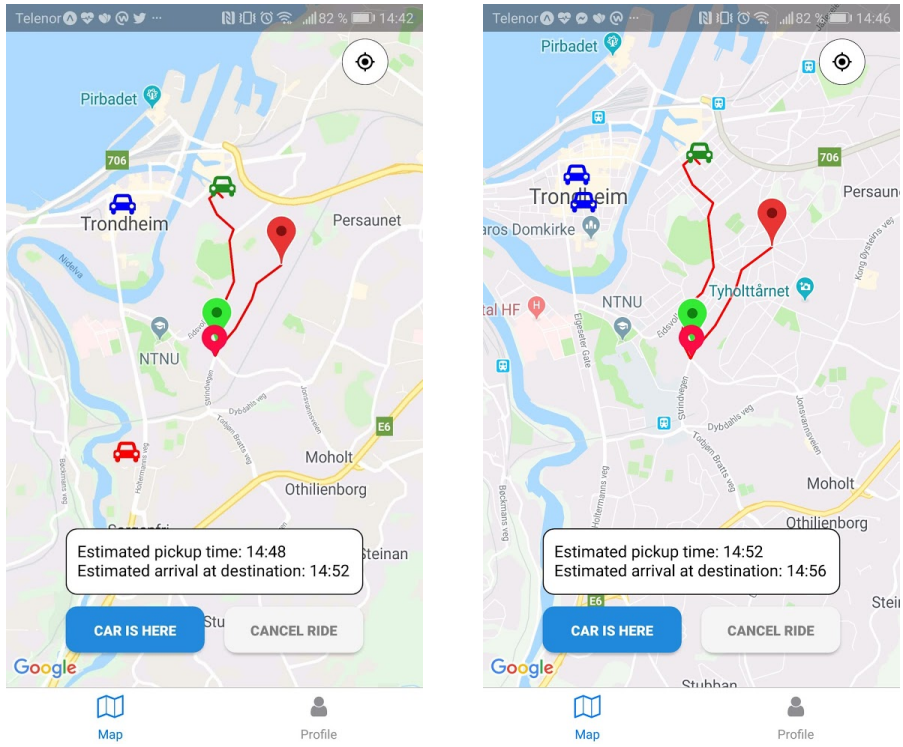


Figure 4.12: Simultaneous bookings by different users

ID	Functional Requirement	Priority
FR9	The application should manage multiple bookings simultaneously	High
FR10	The application should support several users simultaneously	High

Table 4.8: Solved requirements for figure 4.12

The screenshots in figure 4.12 shows the map when different users are performing bookings. The left image shows a user performing a booking(green) while someone else has booked another car(red). While the user waits for the arrival of the booked car, the other car has reached its destination. When a user performs a booking, the client will send the data to the server using WebSockets. The server will register the changes and broadcast it back to all the other clients. This makes them able to observe all the changes in real-time, making The map update automatically as changes happen.

Real-Time Updates of Cars

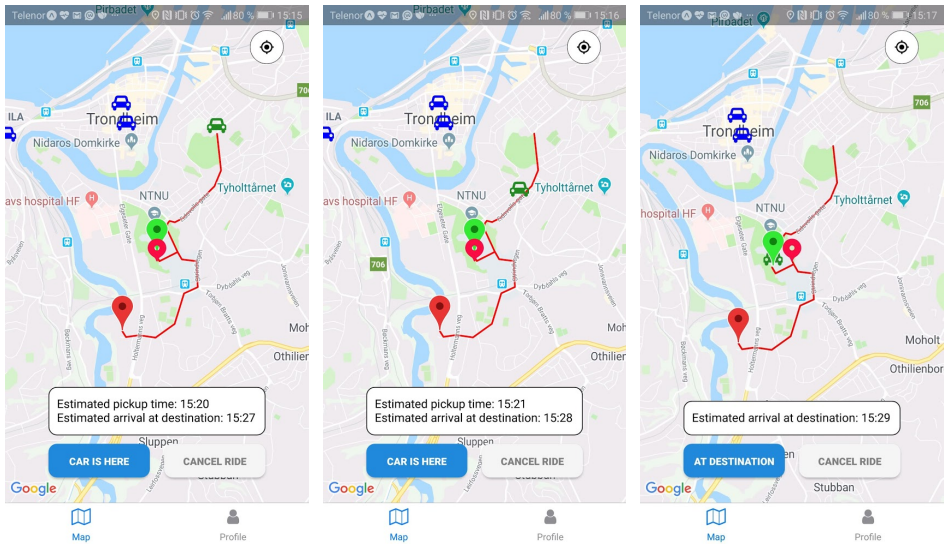


Figure 4.13: Car moving in real-time

ID	Functional Requirement	Priority
FR11	The application should display cars in real-time	High

Table 4.9: Solved requirements for figure 4.13

Users can observe where the different cars are driving by paying attention to the changes on the car icons. For every 10 seconds, the cars will update their position to the database. The clients are getting the cars through WebSockets, meaning that when a car updates its position, the user will see the new location on the map. Figure 4.13 shows a booked car moving towards the user and the pickup location.

In order for the car to update its position, a new application was developed. The idea is that it runs on an iPad that is stationed inside the vehicle. This application is connected to the same server as the users. For every 10 seconds, it will check its coordinates and update the position in the database. When the update has been performed, the cars are redistributed to the users, making them able to see where the cars are at all times. It calculates the route in the same way as the Nap App, using Google API for directions. The coordinates for current, pickup and destination positions are all stored in the database. This makes it possible for the computer inside the car to drive to these locations as well. Figure 4.14 illustrates the app running in the car. This is just a map, where it will show the name of the user it is supposed to pick up. The main functionality is to update the car's position.

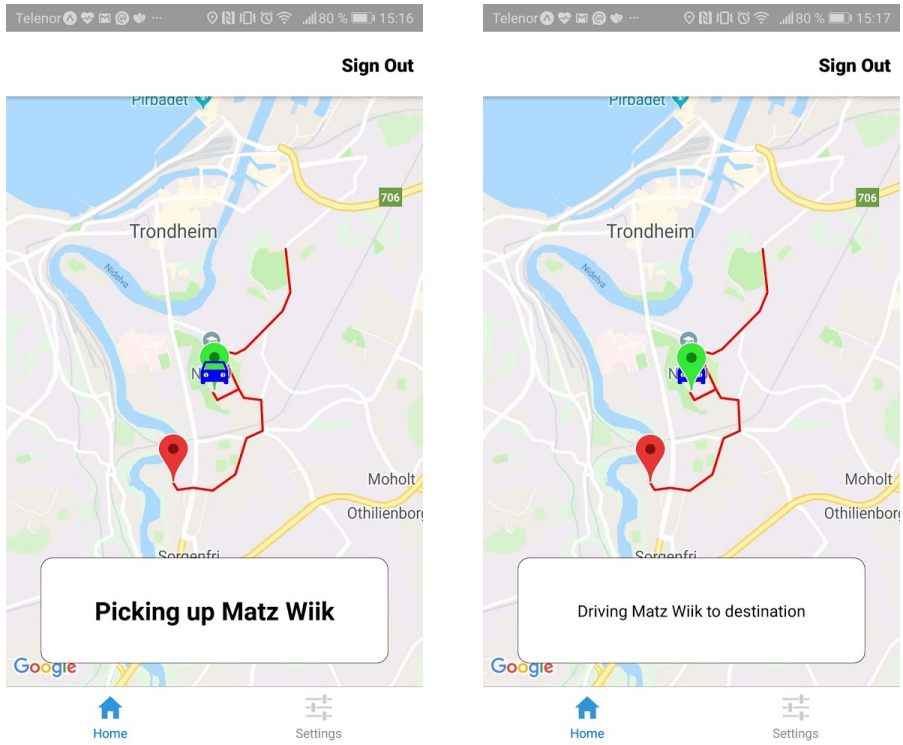


Figure 4.14: Screenshots from the app running inside the car

User Profile

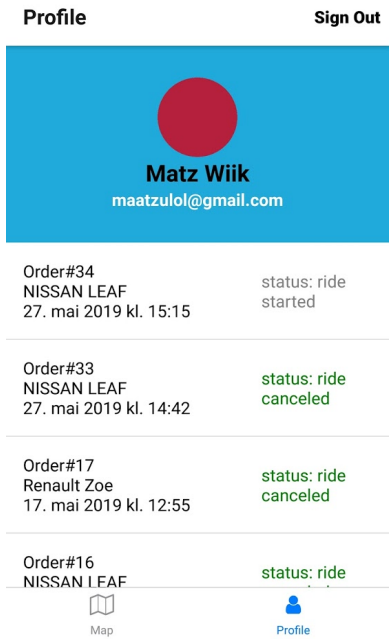


Figure 4.15: Profile screen showing booking history

ID	Functional Requirement	Priority
FR12	User should be able to see all the previous bookings	Low

Table 4.10: Solved requirements for figure 4.15

After the user has signed in, the user profile will be visible instead of the sign-in page. The history of the previous bookings performed by the user is presented here. This is presented in the form of the date, car model and the status. Finished rides are shown with green text and active rides with grey text. There is not much else in the profile page as the scope does not define anything else, but more functionality can be implemented. For example user settings.

4.2.2.2 Requirement Solutions for Nap App Admin

Sign in page

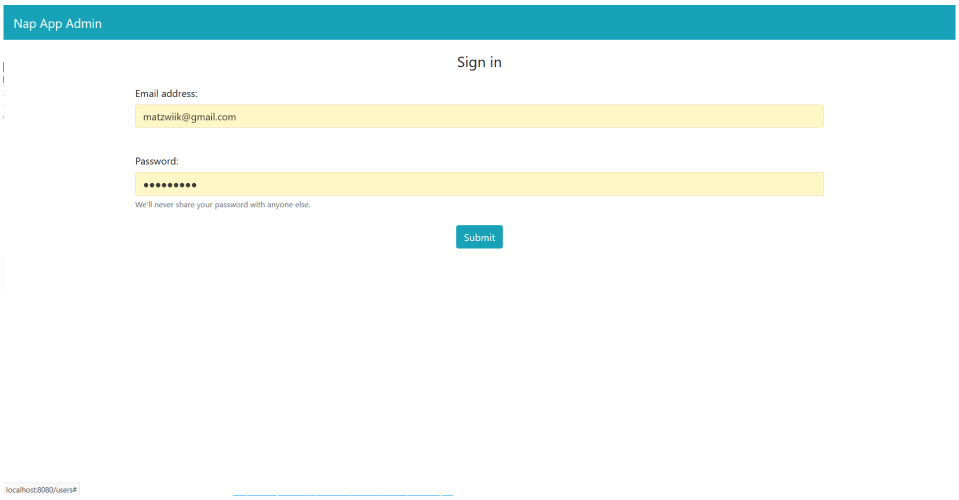


Figure 4.16: Login page from admin tool

ID	Functional Requirement	Priority
FR6	Only users with admin role should be able to sign in	High

Table 4.11: Solved requirements for figure 4.16

The first page appearing when launching the admin page is a sign in page. Only users with role id 1 are able to sign in here. Role id 1 equals admin and role id 2 is a standard user. If login credentials are wrong, an error message will be displayed back to the user. This is done in the same way as user login in the Nap App, displayed in figure 4.9.

User administration page

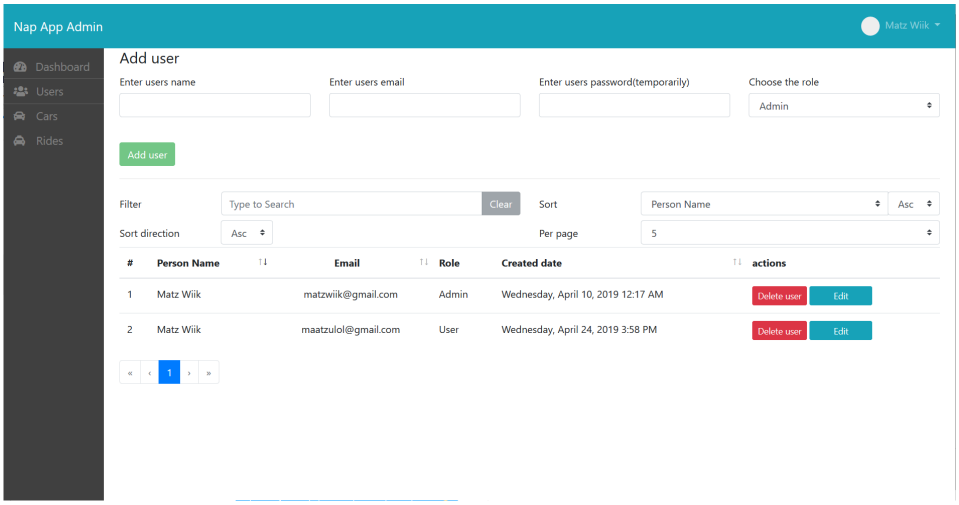


Figure 4.17: User overview in admin tool

ID	Functional Requirement	Priority
FR1	Should be able to add user, and user role	High

Table 4.12: Solved requirements for figure 4.17

All the actions are performed on the same screen. By adding a user, the admin needs to type in the information at the top of the screen. When the user gets stored, they will appear in real-time in the user list. It is implemented different functions for filtering users, and the admin also has the opportunity to delete and edit them. It is possible to decide how many users should appear in the list, and sort them based on name, email, role and creation date. If the admin chooses to delete a user, a confirmation screen will appear in case of a miss click. There is no way the admin can get a hold of the user's password, as this is private. It is not either possible to edit the password.

Cars administration page

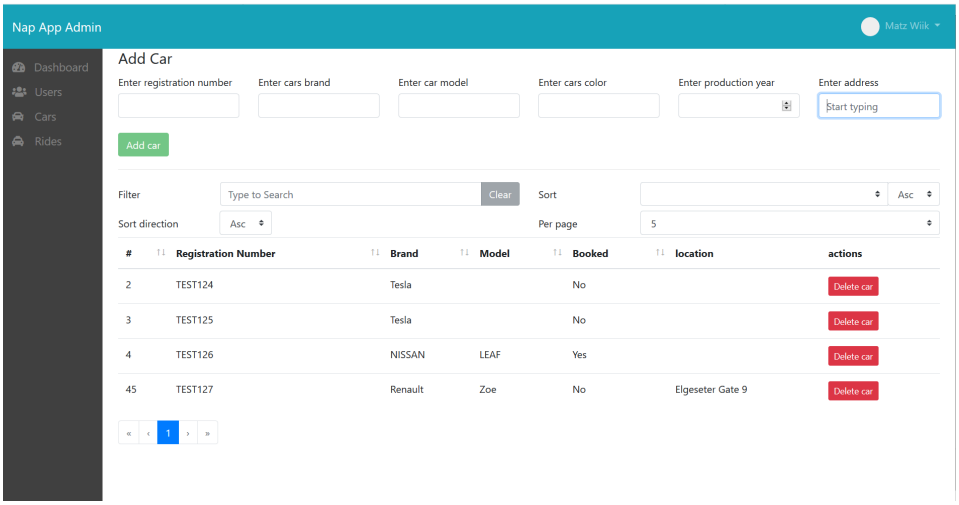


Figure 4.18: Cars overview in admin tool

ID	Functional Requirement	Priority
FR2	Should be able to add cars	High
FR4	Get an overview over registered cars	High

Table 4.13: Solved requirements for figure 4.18

This screen is similar to the user administration page. It is possible to add new cars with the form on top of the screen. The admin types the car information and chooses its destination by typing the address in the address input field. Location recommendations will be visible as the admin types, getting locations from the Google Places API. Upon choosing one of the addresses, coordinates will be received from the API and stored in the database.

The admin will also have an overview of registered cars. It is possible to sort these based on the different fields and delete a car.

Requirements not implemented

ID	Functional Requirement	Priority
FR3	Edit cars	Medium
FR5	Get an overview over rides	Medium

Table 4.14: Requirements not solved for the admin tool

As the Nap App was not working live with real users and cars when this application was developed, some of the features are missing. These were also the ones with medium priority, not focusing too much on them. At this point, there were not any rides available in the database as it was impossible to book cars. As the app now works in a live environment, finishing the admin tool is now possible.

4.2.2.3 Database and Server hosting in Google Cloud Platform

Upon validating if the Google Cloud Platform is a viable tool for database and server hosting, different details from the load on the servers will be presented. These include CPU usage, memory usage, storage usage, latency and connections to the back-end. Lastly, an overview of the total billings of the services will be presented. The graphs will display the stats from May 2019, the user tests were conducted on the first of May, measuring the loads based on that.

App Engine

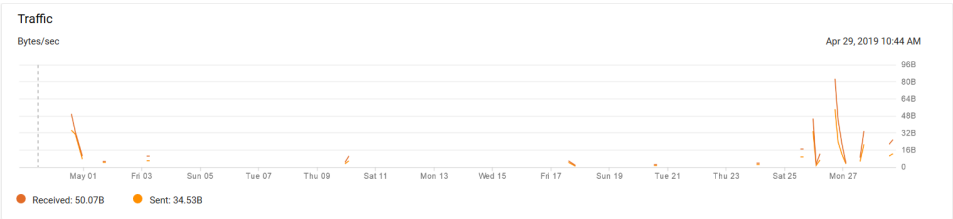


Figure 4.19: App engine traffic May 2019

As seen in figure 4.19 there has not been too much traffic in May. The graph displayed around the first of May is when the user tests were conducted. It was not possible to display previous months, checking the traffic when doing the alpha test.

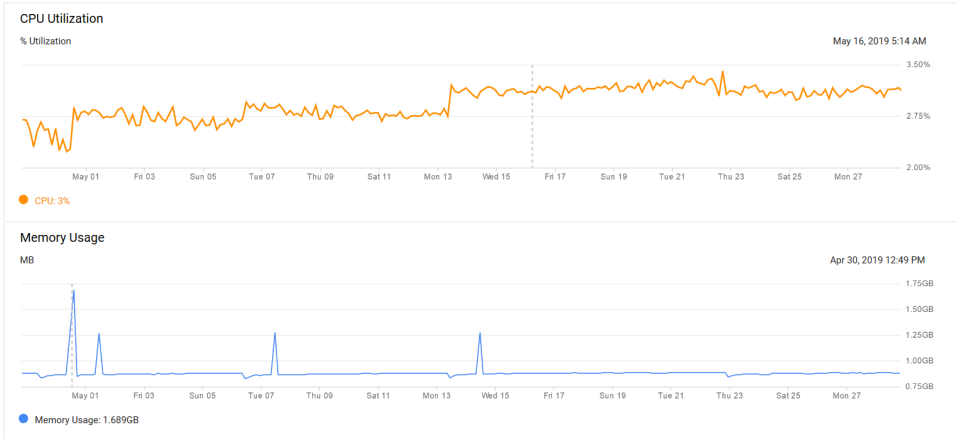


Figure 4.20: Graphs showing CPU and memory usage from the server

Figure 4.20 shows the CPU and memory usage across May 2019. It does not appear that the CPU usage has any correlation with the traffic, as the CPU usage stays almost the same throughout the entire month. The period where the traffic increases, the CPU usage does not increase. One big factor is that the load on the server is minimal, probably having none effect on the CPU. The memory, however, correlates with the traffic, using about 1.6 GB as the user tests were done.

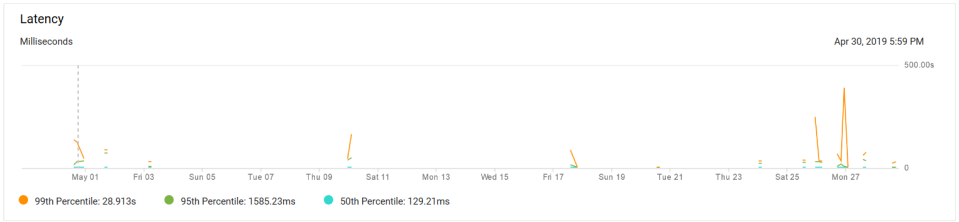


Figure 4.21: Latency between client and server, May 2019

As the traffic increased, the latency between client and server increased. According to the graph, the 50th percentile shows that the latency lies around 129ms. Google Cloud Platform docs describes the 50th percentile as the median, and this is the stat that should be looked at in order to measure the overall latency. The 95th percentile represents that only 5 percent of the total requests had a latency of 1500ms or more.

Database

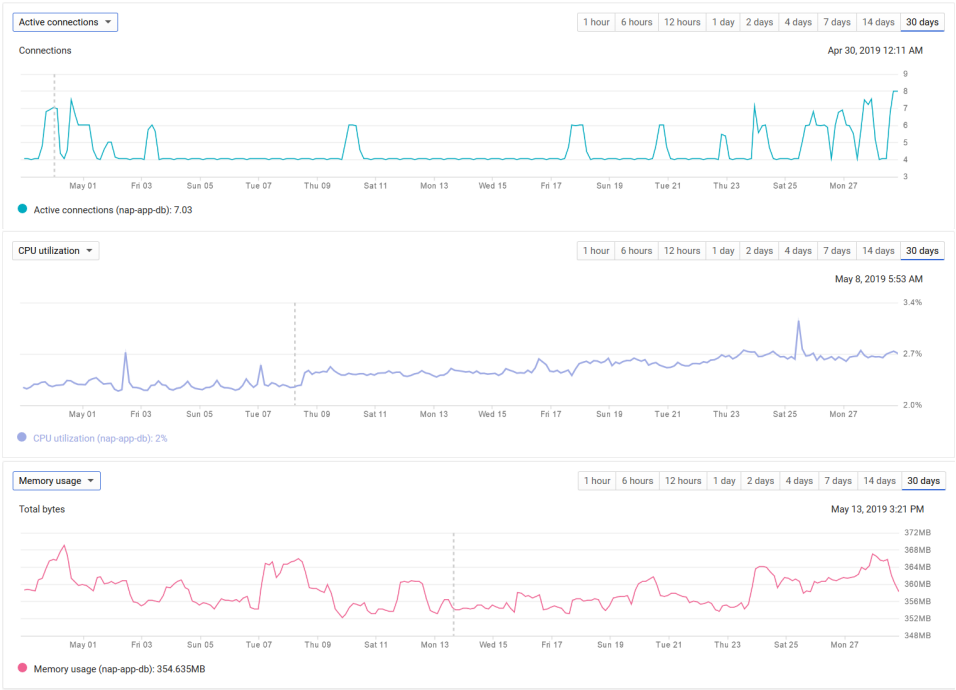


Figure 4.22: Graphs showing total connections, CPU usage and memory usage

The database had an average of seven active connections. Upon performing the user tests, it had at the most 8 active connections at the same time. It does not appear that both the CPU and Memory are increasing as operations are done in the database. Both the CPU and memory usage graphs are staying consistent, most probably just the machine using the capacity. As the load increases, the user will probably increase.

Billing

The billing transactions display the cost of using the different services in the Google Cloud Platform. The App Engine is the most costly service, having a price of 650 NOK just by running and 87 NOK based on the operations it does. This will increase as the load on the server is increasing. The pricing for the database also includes a price for the number of hours it runs and a cost for the total storage. The transactions for the different API's are based on how many times they are used. For example, the geocoding API had 557 counts for a total price of 24.13 NOK. This means that for each time a geocoding is requested, it will cost 0.04 NOK. The billing transaction for May 2019 can be further examined in Appendix D.

4.2.3 Evaluation

This section presents both the experience doing the alpha test and the feedback given by the users participating in the user tests.

4.2.3.1 Alpha test

The alpha test proved that it was possible to use the Nap app in a real-life scenario. By having two users book cars at the same they in the city center of Trondheim, they could identify that they got offered a car and that another booking happened on the app screen. The person who simulated being a car was running the Nap car app, and he could observe the booking through his mobile device. He then had to follow the directions given to the user who performed the booking. As the ”car” moved towards the pickup point, the user that performed the booking could observe the ”car” moving towards him on the map screen. When the ”car” arrived, he then could confirm that the car is waiting outside. The other person who performed a booking could also see the car move while he was waiting for his ride.

Videos with the demonstration of the two different apps can be found in Appendix G - Demonstration video.

Identified Problems

As the main functionality proved to work, there was however some identified problems. The Nap car app had some problems with Android devices. When the screen went black, it did not update its position to the database, resulting in no changes in the Nap app. This could be a problem related to the background fetching function in the Expo framework, as this is a feature that was recently added. When the screen was turned on again, it still did not update its position. If the app was running on an iPhone, no problems occurred.

Some stuttering was noticed in the Nap app. Sometimes, the car did not update for every 10 seconds. While it did for the majority of the time, some times it could take 20-30 seconds to observe changes. Using an interval for updating the car’s position every 10 seconds could be to low.

4.2.3.2 User test

The user test was done by gathering participants. They got a briefing over what the purpose of the app is, and a set of tasks displayed in Appendix C. After the task was done, they did a questionnaire regarding the usability and features related to both improve the app and their trust in autonomous cars.

4.2.3.2.1 Participants

In total, ten participants participated in the user test. In order to make the sample of participants as representative as possible for the user group, it was made sure that all of the

participants did not own a car. As not owning a car, will probably increase their motivation to use this app. All of the users owned their own smartphone, and the app was launched on their own device. It was made sure that the participants had different backgrounds, as this will be more representable for the overall user group.

All of the participants were familiar with the use of a smartphone, and they used it multiple times in a day. About half of the participants did not trust autonomous cars and the other half was not sure. Most of the users were not familiar with ADS systems and they had some prior experience with taxi booking apps.

Background	Values
Age	under 20 - 40 years
Smartphone usage	Multiple times a day
Experience using taxi booking apps	80 percent
Trusts autonomous cars	50 percent maybe - 50 percent no
Experience using ADS	40 percent some - 60 percent no

Table 4.15: User background overview

4.2.3.2.2 Procedure

The user test was conducted between 01 and 02 May. It was held in different places in Trondheim, based on where the participant could meet. Since their own smartphone was used, a lot of equipment was not needed.

Procedure	Approximate time
Introdoction to concept and app	5 minutes
Execution of tasks	10 minutes
Answering questionnaire	10 minutes

Table 4.16: User test procedure and time estimate

Introduction to concept and app

The test started with the participant being introduced with the concept and why they participate in this test. It was clearly explained that all of the user information is deleted after the test. Furthermore, they were asked if they could complete a set of tasks and a questionnaire. If the participant accepted the terms, the person was told that it is not the user being tested, but the application itself.

Furthermore, it was told how the participant can load the app on their own phone and if the person could think loudly while performing the tasks

Execution of tasks

When the participant started doing the tasks given in Appendix C, the interviewer observed how the different tasks were performed. While observing, notes were taken of problems that occurred when the execution of the tasks. It was not possible for the participant to ask questions if he/she got stuck. After the tasks were completed, the participant could ask any question as pleased.

Answering questionnaire

In the end, the participants answered a questionnaire regarding their background and their thoughts about the application. It can be further examined in Appendix E. The participants could express their opinions about the app in the form of a SUS scale and some questions. They also answered some questions regarding their trust in autonomous cars.

4.2.3.2.3 Identified Problems

Manual registration and sign in

During the first user test it was discovered that manual login using email and password did not work. Nothing happened when the user tried to register. This was probably because of a bug related to the server code. For the rest of the users, it was asked if they had a Google or LinkedIn account, and the majority had an account on these services. If they did not want to sign in using these, they were offered to use the interviewer's account.

Changing location before booking

The feature for editing a location after pickup and destination were chosen did not work. All of the participants tried to press on the information boxes with locations, but nothing happened. This was most likely due to a bug in the app. A more thorough test should have been done by the developer before the user tests were held. Participants were able to change locations by canceling the booking, then booking again. This proved to be a difficult solution to the problem.

4.2.3.2.4 Observation

During the user test the interviewer was observing what the user did. This gave a deeper insight into how a user will use the system based on the tasks given.

Registration and login

Registration and login were intuitive. All the users went straight to the profile page and chose to either sign in using credentials or Google/LinkedIn. There was a problem where one user could not log in with email and password, but this was more related to a bug, not the user. The user could also clearly notice that they had to sign in to book a car. Also

the text for login and sign up was not visible because of low contrast with the background image

Navigating the map

All of the users managed to choose a pickup point by navigating the map. There was however some problems regarding navigating back to the current location and choosing a pickup location. When the user tried to book from their current location, everyone tried to tap on the blue dot instead of the current location button at the top of the screen. Also, one user tried to choose a pickup point by tapping on a point on the map, not by navigating with the marker.

Perform a booking of a car

Most users managed to book a car without having any issues. They managed to choose a pickup location either by using the marker or typing the address. In order to choose a destination, there were some issues when they typed the address. The app uses Google Places autocomplete, giving suggestions on addresses as they type. In order to choose a full address with the number, they had to type the whole string. This made it a bit cumbersome choosing destination.

Editing booking

All of the users experienced problems editing a booking as a bug occurred with the app. It should be able to tap on either the pickup or destination address to change the booking, but this did not work in the test. It was however solved by canceling and ordering it again.

Tracking the ride

There was no problems identified for tracking the car. All users understood that it was on its way to them and that they had to confirm when it arrived.

Checking booking history

Most of the users had no problem locating the history. Two users used some extra time noticing it because it was missing a header or title at the profile page, not indicating that it was the history. Also, one user did not find out how to update the list. This was done by dragging it downwards.

4.2.3.2.5 Usability

In order to measure the usability, the questionnaire contained SUS questions following the standard. These scores can define the usability of the system. The combined score will show the overall usability. Table 4.17 shows the average sus score for all the participants.

SUS Question	Average SUS Score
I think that I would like to use this system frequently	77.5
I found the system unnecessarily complex	92.5
I thought the system was easy to use	90.0
I think I would need the support of a technical person to be able to use this system.	87.5
I found the various functions in this system were well integrated.	65.0
I thought there was too much inconsistency in this system.	62.5
I would imagine that most people would learn to use this system very quickly.	90.0
I found the system very cumbersome to use.	80.0
I felt very confident using the system	80.0
I needed to learn a lot of things before I could get going with this system.	95.0
Total average SUS score	**82.0**

Table 4.17: Average SUS score per question

4.2.3.2.6 About the product

One part of the questionnaire contained questions about the participant's thoughts about the product. These were questions that had a scale between 1 and 5, the same as the SUS scoring where 1 is low and 5 is high. The scores are divided based on what the participant answered with the background in either smartphone use, experience using ADAS and general trust in autonomous cars.

	Average						
Question	**Total**	**Experience using taxi app**		**Used ADAS**		**Trusts autonomous car**	
		Yes	No	Some	No	Maybe	No
This product will make my everyday life easier.	3.40	3.50	3.38	2.33	3.86	4.00	2.80
I would book an autonomous car using this system.	4.20	5.0	4.00	3.75	4.50	5.00	3.40
This system influenced my trust in autonomous cars.	1.70	3.50	1.25	1.75	1.66	2.20	1.20

4.2.3.2.7 Trust in autonomous cars

	Experience using ADAS	Trust autonomous car
P1	No	No
P2	Some	Maybe
P3	No	Maybe
P4	No	Maybe
P5	Some	No
P6	No	No
P7	No	Maybe
P8	No	Maybe
P9	Some	No
P10	Some	No

Table 4.18: Correlation between ADAS experience and trust

The questions are organized so that based on what the participants answered, the answers are divided into different categories. This model wants to be followed for the other questions as well. Their answers will then be presented with the associated category.
The question is formulated in this way: *Can you list a reason for why you trust/don't trust autonomous cars?*

Lack of information - Lack of information about autonomous cars

P1: *Not enough information about how safe it is.*

Accidents - Afraid of accidents happening

P2: *Distrust to autonomous systems in general, and especially in situations with potentially fatal outcome.*

P5: *Accidents*

Testing - Not enough testing has been done.

P4: *Skeptical to the technology, has not been fully tested yet or seen in action.*

P7: *haven't been tested much, after a couple of years with more testing/own experiences i think i will be a lot more secure*

Experience - Not enough experience, or do not know anyone that has experienced it.

P5: *Do not know anyone that has used it.*

P6: *Has no experience with it and knows no one who has experience with it.*

P7: *Feeling uncertain about it. Has neither tried it yourself nor heard the experience of others about it.*

Bugs or faults - Afraid of some technical problems to the machine .

P8: *Do not trust because of the system suddenly going down then can even get on the wrong road, the car could be locked down*

P10: *I would like to drive the car myself, do not trust a machine doing it because of all bugs and faults on current machines on our society.*

4.2.3.2.8 Utility value of the app

In order to find out why the users would use the app, this question were answered: *Why would you use or not use this service instead of a traditional taxi booking service?*

Safety - The autonomous car needs to be safer than a taxi.

P1: *As or right now, I can't think of a reason to use it instead of a taxi. Then it needs to be safer*

P3: *Security*

P6: *If it turns out to be safer than a man driving, then I would use it for safety reasons*

P9: *If it had been the way it is now, then I would not use it, but if it is safe it must be cheaper.*

Efficient Driving - Autonomous car will arrive faster than a taxi.

P2: *I would think that not all taxi drivers drives the best possible route, and maybe a machine would calculate it better?*

P3: *Best route to destination*

P7: *If this service proved to be much faster and reliable than a normal taxi service. No queues*

Price - It needs to be cheaper than a standard taxi service

P4: *If the service is less expensive, then maybe I would try in a limited area first.*

P8: *If it is cheaper*

P9: *If it had been the way it is now, then I would not use it, but if it is safe it must be cheaper.*

4.2.3.2.9 Suggestions for improvement of the app

In order to find out how to improve the app, this question were answered by the participants: *Is there anything that should be added or removed to make the product more attractive for you to use?*

Choose destination using viewport - Use the viewport marker or click the map to choose destination.

P2: *Should be possible to choose the destination by moving the viewport, such as with pickup location.*

P5: *Choose destination with the map*

P7: *dynamically moving markers for start position, as well as clicking for setting end location*

Improvement of the UI - using a more consistent UI and less steps doing tasks.

P3: *More consistent / implemented design in the app, some components do not look the same.*

P4: *General bug fix. A more "intuitive" UI when choosing destinations both from and to. Was a little "tiring" that one had to go through several screens to choose from and to destinations.*

Carpooling - be able to share a ride with other users.

P6: *Carpooling with other users*

P9: *Carpooling*

Pricing - Display pricing for a ride.

P9: *Price for an order*

Car information - Display information about the car and the equipment it has.

P10: *some information about the car and what kind of equipment it has as I have kids.*

Time Estimate - Display time estimate before doing a booking.

P10: *I would like to see the time estimate before I did a booking*

4.2.3.2.10 Improve the users trust in autonomous cars

In order to evaluate whether there are some features that could be implemented to affect the users trust in autonomous cars, this question were answered by the participants: *Are there any features that could be implemented to affect your trust in autonomous cars?*

User experiences - Display how other users experienced their rides.

P1: *I would like to see what other users thought of their previous rides.*

P2: *User reviews in some format.*

P3: *User reviews in appstore.*

P6: *Information with experience from other users had been helpful. Had I seen that others trust that I would also.*

P7: *feedback from other users, maybe for the specific car*

P8: *Rating/reviews*

P10: *It could also be nice to see what other pepls thoughs are, did they have a pleasant ride maybe I would try it.*

Information about the cars - Stats and information about the different cars.

P3: *Live view of cars not in use. Stats about cars.*

P5: *information about self-driving cars, statistics on how many people have used a car in Trondheim the last 24 hours / week*

P8: *information about the car's safety system, state of how long the car has been in operation, faults / defects, how many times it has traveled the stretch before.*

P9: *Information about the cars that are driving. How long they have been in operation and whether they have had any mistakes or accidents before.*

Security - Security features that makes the users feel more safe.

P2: *support chat with actual human operators.*

P7: *camera in the car? notice someone (of your choice) that you have been picked up and is in the car (until you get out), show where you are and status of the car so that if anything happens they can call for help*

P8: *Information about the cars security system*

P10: *I would like to know if the car has any safety driver that can take control in case something should happen*

Test - Have the opportunity to test a car before using the service.

P5: *Be able to test the car in a limited area*

P6: *Would be practical to test on a real car before using the app*

Chapter 5

Discussion

This chapter discusses the results from chapter 4, in more detail. The results will be discussed in relation to the research questions described in chapter 1.3. In addition, the results will be linked to the pre-study conducted in chapter 2.

5.1 Development

RQ1: *In what way is it possible to develop an application for the booking of genuine autonomous cars, using modern technology?*

To answer this question, the different results from the development process and the resulting application were evaluated.

5.1.1 Front-end development

5.1.1.1 React Native

React Native proved to be an efficient solution for creating mobile apps. By sharing the same codebase with Android and iOS, the developer could write the code once and run it on multiple platforms. This eliminates the need to write the code again and again. This process is directly related to reducing development time. If this was a commercial app, the framework could be helpful in reducing development costs because of the time saved. In theory, if a developer needs to learn both Java and Swift to develop an app the costs could be reduced by 50%.

In order to solve the project, two apps had to be developed. It is ideal to use some of the same components on both apps in order to keep the same profile. React Native proved useful as the components from one app could be used on the other. This resulted in that the development time of the nap-car app only took one week as it was possible to use a lot of the same components from the Nap app.

It allowed the use of third party packages. In this project, it was used packages for navigation and UI elements. By using these, it was not necessary to develop them, but just implementing the navigation and UI components necessary.

Since it was not developed any app using native language, it is now known if the app would perform better. It is thinkable that an app developed in React Native performs worse since the app is coded in JavaScript and needs to be compiled into the native language of the device.

5.1.1.2 Expo

By using Expo it was easy to use hardware sensors such as GPS, as this could be accessed with just a single line of code. It also brought some extra features such as the map screen, icons, and app loading indicator. With Expo, it is possible to develop on either Windows, Linux or Mac. This is done by launching the code in the Expo-CLI and the app could be launched through the Expo client app. This was also the feature used when testing. The participants did not need to connect with a cable, they could just scan the barcode with the expo app and the app would launch on their phones. With the hot reloading feature, the app would update as changes were done in the code. This means that a lot of time is saved compared to standard React Native, where the code needs to be compiled before launching

the app on a device.

If Expo is chosen for the React Native project you are however committed to using it. It is not possible to convert the project to vanilla React-Native. This means that if a feature is missing in React Native, you have to wait until it gets available. At the start of the project, the background location fetching was not available, but it was added in February. This was useful in order for the Nap car app to work. If it had not been available until then, it would be impossible to use the Expo framework. During a demo of the app with an autonomous car, Apple had removed support for location fetching through the expo client, which meant that the demo could not be performed. Therefore an evaluation of the features needed is advised before deciding on using it.

5.1.1.3 Vue.Js

Since JavaScript was used for the native applications, using it for the web framework resulted in a short-term development. It could be discussed to use React.Js for the web-app since React Native was already used. The developer decided to test Vue, as it proved to be a trending framework. It was easy to learn Vue as React was already known. Vue takes elements that are good about React and changes the more cumbersome elements. A good example is the state manager. In order to use a global state manager in React, Redux had to be used. It proved to be a lot of boilerplate implementing it and the developer needed to keep track of a lot of files. In Vue, this was implemented in a much easier way.

5.1.2 Back-end Development

5.1.2.1 Node.Js

For the purpose, Node.Js worked well for coding the server functionality. By using JavaScript on the server-side, one could use only one language for the whole development. This means that it is possible to do full-stack development only coding in JavaScript. It is completely free to use and is supported on most platforms and services. It could have been possible to create the back-end solution in a framework such as Meteor.js(Meteor, 2019). It is built upon Node, but instead of implementing functions from the ground up, Meteor offers pre-developed features and functions such as authentication. This could reduce development time significantly, but like Expo, you would be committed to it.

5.1.2.2 Google Cloud Platform

By hosting the server code and database on Google Cloud Platform, the app could always have a connection with the server and database. The deployment of the server was easily done with one command, and a secure connection between the client and server was given automatically by Google in the form of an SSL connection. daily data was backed up which made it possible to feel confident that something was not lost. Although it was not done any heavy load on the server during testing, the servers running in the platform did not use a lot of computing power. Since a number of API's were used from Google, there is an overview of these as well as both server and database usage. This proved to be useful

when reviewing transaction history for the use and payment of services. It did not seem to be an expensive solution, as prices are relatively low. Again there were not so many requests, so a larger test with several users should have been done.

In view of all the rumors that Google contains user data, one can ask whether it is prudent to use the service. It is conceivable that Google takes care of store data in terms of a number of requests and which phones have requested a location etc. They also have access to the databases, which is a great source of information.

5.1.3 Functionality of the app

From the results of both the alpha and user test, the application worked well for its purpose. During the alpha test, a simulation of a real car was performed in the best possible way. The main functionality for booking and tracking the car worked well. There were some minor faults with registration and editing locations, but these did not result in not achieving the goal and can be easily fixed with some bug fixes. Although some improvements were recommended by the participants in the user test, They were positive in using the app for booking an autonomous car with a score of 4.20. The Nap car app did not work properly on Android, but this was a problem related to Expo, but not the development.

The app should have been tested on more users simultaneously. Evaluating if the system as it is now could handle it. It would also be interesting to see if the server could handle a larger simultaneous load, as the alpha test was just done with three simultaneous users. The user test was not done with a simulation of a car. They had to pretend that the car arrived and picked them up, then being driven to the destination. A bigger scale the alpha test should have been done in order for the user to feel that they booked a real car.

5.2 Success Factor

RQ2: *Which factors affect the success of an autonomous car booking app?*

In order to answer this question, the data from the questionnaire has to be analyzed. In the user test, some question was asked in regard to reasons why they would use the app. Furthermore, a SUS score is analyzed in order to evaluate the usability of the app.

5.2.1 Usefulness

The total score from the question: *This product will make my everyday life easier* had an average of 3.40, indicating that most of the participants did not have an opinion about it. This could be a pointer to that the participants do not know if the app is useful for them or not. Interestingly, among the participants who might rely on autonomous cars, the score was 4.00 and 2.80 for the ones that don't rely on them.

In order to find reasons for why the participants would use this app instead of a traditional taxi app, they answered this question: *Why would you use or not use this service instead of a traditional taxi booking service?*

5.2.1.1 Safety

The biggest factor for why they would use the service proved to be for security reasons. P1 answered: *"As or right now, I can't think of a reason to use it instead of a taxi. Then it needs to be safer"*. This indicates that he would not use it now, but if it turns out to be safer, then it could have been relevant. P6 answered: *If it turns out to be safer than a man driving*. The interesting part here is that one of the main reasons for using autonomous cars is to eliminate human errors. This could mean that people don't think it is safer as of now. Two more participants answered that safety is a big factor. One participant also said that it would be nice to use it in a limited area first.

5.2.1.2 Efficiency

P2 answered: *I would think that not all taxi drivers drive the best possible route, and maybe a machine would calculate it better?* and P7 answered: *If this service proved to be much faster and reliable than a normal taxi service.* This could indicate that people would use it if it is more efficient than a normal taxi, driving to the user with the best possible route. It also seems that the participants think a machine would drive more efficiently than a human. Even though they don't trust machines completely, they think it is smarter in some areas. P3 also answered that he would use it if it's faster.

5.2.1.3 Pricing

In order to use this instead of a normal taxi, pricing is also relevant. P4 answered: *If the service is less expensive then maybe I would try in a limited area first.* P8 and P9 also answered that it has to be cheaper. If the price is the same, they would probably use a normal taxi. It would have been interesting to observe how much cheaper the service should be in order for them to use it. If it is a big difference in price, would a person that does not trust autonomous cars use them?

5.2.1.4 Environment friendly

P3 answered reducing CO2 emissions by driving the best route and using electronic cars could be beneficial. Maybe more people would think this is a factor, as it is a big debate around the world.

5.2.1.5 Carpooling

P7 answered: *maybe it could be shared with someone else, splitting the bill?*. P6 and P9 also answered carpooling when asked for features to be added. This is something that could be relevant to the environment and pricing. As the car will drive less if a route can be combined between multiple people, the pricing will also be reduced as the users share the bill.

5.2.2 Usability

5.2.2.1 Generally intuitive

The app proved to be intuitive. Achieving an average SUS score of 82 which is above the average described in section 3.4.4. It should have been done two different calculations, one with the persons having experience with taxi booking apps, and one without. Since this app is quite similar to such apps, it would be interesting to see if there was a large deviation on the SUS score. When asked if they would like to use this system to book an autonomous car, the score of 4.20 is achieved. This means that they are quite positive in using it. A question in regard to what their thoughts about the product should have been asked in order to describe why they did like or not like with the UI. When asked about the improvements of the app P3 answered: *More consistent/implemented design in the app, some components do not look the same*. P4 answered: *General bug fix. A more "intuitive" UI when choosing destinations both from and to.* This indicates that improvements could have been done with the UI, but further testing should have been performed to evaluate it thoroughly.

5.2.2.2 Choosing destination

P2 answered: *It should be possible to choose the destination by moving the viewport, such as with pickup location.* P5 also answered something similar. This problem also occurred under the observation. The participants thought it was difficult to write a destination. They would like to choose a destination by using the cursor, such as when choosing the pickup location. This would reduce the number off steps as a screen pops up after choosing pickup.

5.2.2.3 Choosing location by tapping on the map

P7 answered: *dynamically moving markers for start position, as well as clicking for setting end location*. This could be an intuitive feature for someone as probably a lot of people have used Google Maps. There it is possible to tap a location on the map and information or a marker will be shown. It is also thinkable that for some it can save time. Instead of targeting a location with a marker, it could be faster for someone to just tap the location.

5.2.2.4 Displaying time estimate before booking

P7 answered: *I would like to see the time estimate before I did a booking.* This could improve the usability by reducing one extra tap. As it is now, you have to choose pickup and destination before the time estimate is displayed. by showing the time estimate before choosing a destination, one can reduce the number of taps by one. Some users may just want to see how long it will take for a car to arrive.

5.2.3 Experience

All participants had a lot of experience using a smartphone. 80% had experience with similar apps, such as taxi booking. This could have affected their opinions and usability

about the app. It is conceivable that since they have experience with it, much of the functionality they recommended can be of great use. In terms of usability, it would have been interesting to test it on someone who does not have so much experience with similar apps.

5.3 Trust in autonomous cars

RQ3: *To what extent can the app affect whether users rely on autonomous cars?*

In order to evaluate this question, a background check on peoples trust in autonomous cars was conducted in section 2.2. Participants in the user test also answered questions related to why they trust autonomous cars or not. It is interesting to see if there is any correlation between the findings in section 2.2 and what the participants answered. They also answered whether there are any features that could have been implemented to influence their trust.

5.3.1 Reasons for trust

P1 mentioned that it is not enough information about how safe it is. This answer is tied with the other reasons, as they can be used for information about safety.

5.3.1.1 Experience using ADAS

One of the factors presented in section 2.2 is that experience using ADAS could have an effect on whether people rely on autonomous cars. Based on the results of the interviews, it did not turn out to be the case in this research. Table 4.18 presents that only two people without ADAS experience don't trust autonomous cars, and one person with experience trusts it. It is thinkable that another result can be achieved with bigger sample size.

5.3.1.2 Experience and Testing

Both experience and testing of the app proved to be a big factor for why people don't rely on autonomous cars. Four participants mentioned that this is a reason for not trusting them. P7 answered: *Feeling uncertain about it. Has neither tried it yourself nor heard the experience of others about it.* P4 answered: *Skeptical to the technology, has not been fully tested yet or seen in action.* This fits well with what was mentioned in section 2.2. When people have gained some experience with autonomous cars, they feel more confident in using it. P6 answered: *Has no experience with it and knows no one who has experience with it.* This can mean that social influence can have an effect. If acquaintances have had experiences with autonomous cars, it may have an impact on their trust

5.3.1.3 Accidents

There has not been a lot of accidents, but some has occurred. When these happen and the majority of people don't trust autonomous cars in general, it will have a negative effect. P2 answered: *Distrust to autonomous systems in general, and especially in situations with*

potentially fatal outcome. after a known case of an accident related to an autonomous car happened, the general trust went down. This can be observed in figure 2.4.

5.3.1.4 Bugs or Faults

P8 answered: *Do not trust because of the system suddenly going down then can even get on the wrong road, the car could be locked down*. P10 answered: *I would like to drive the car myself, do not trust a machine doing it because of all bugs and faults on current machines on our society*. This is an indication that some people think that a machine is more error-prone compared to a human. It can be difficult for a person to rely blindly on a machine. Fatal accidents can occur in traffic and then it is important to know how good the safety is. This can be tied with the safety factor in section 5.2.1.1

5.3.2 Improving the trust

The product developed during this thesis did not seem to have effect on the user's trust. When asking the question: *This system inuenced my trust in autonomous cars*, it got an average score of 1.70. A place where the score stood out was among those who had experience with taxi booking apps. Here it got an average score of 3.50. This may be an indication that the NAP app is similar to taxi booking apps, which causes them to feel familiar with the system. To find out which features could affect whether users rely on autonomous cars, they answered this question: *Are there any features that could be implemented to affect your trust in autonomous cars?*

5.3.2.1 Feedback from other users

Seven of the participants mentioned that feedback from other users in some way would affect their trust. P6 answered: *Information with experience from other users had been helpful. Had I seen that others trust that, I would also.* P10 answered: *It could also be nice to see what other peoples thoughts are, did they have a pleasant ride maybe I would try it.* Since such a large proportion believe that such functionality can be useful, it can be assumed that this will be a good feature for affecting their trust. We know that such features on apps such as Uber are very useful for a person to rely on a driver. If a self-driving car has good feedback, then it is conceivable that the same applies here.

5.3.2.2 Information about the cars

It seems that information about cars is useful for achieving the users' trust. Another important point is to know how much the car has run and whether it has had any faults or accidents before. P8 answered: *Information about the cars that are driving. How long they have been in operation and whether they have had any mistakes or accidents before*. P9 answered something similar to P8. Based on P5's answer, social influence can also have an impact. He answered: *information about self-driving cars, statistics on how many people have used a car in Trondheim the last 24 hours/week.* Again by observing what other users do in the app, could have an effect.

5.3.2.3 Testing and Security

Some users said that security features are important for them. Some of the answers are irrelevant and impossible to implement in the app, such as a camera in the car and having a safety driver. The app could, however, provide information about a safety driver. P7 answered: *notice someone(of your choice)that you have been picked up and is in the car (until you get out), show where you are and status of the car so that if anything happens they can call for help*, this could be combined with P2 answer: *Support chat with actual human operators.* P5 and P6 would also like to test the car in a limited area. It is possible to give the users one free ride when they register an account, then they could use this for testing purposes.

Chapter 6

Conclusion and Further Work

This chapter presents a conclusion based on the analysis and discussion in the preceding chapters by answering the research questions. It also presents suggestions for further work in order to improve the system.

6.1 Conclusion

In this thesis, the concept of developing an app for booking genuine autonomous cars has been developed. Research has also been done on the success factors of such an app and how it can lead people to rely more on autonomous cars.

The objective of this task was to answer the following research question:

RQ1: To what extent is it possible to develop an application for the booking of genuine autonomous cars, using modern technology?

RQ2: Which factors affect the success of an autonomous car booking app?

RQ3: To what extent can the app affect whether users rely on autonomous cars?

To answer these questions three applications were developed over two iterations, using the most modern and trending technologies within the field of app development. The applications were named of the parent project NTNU Autonomous Perception for autonomous vehicles, these are NAP app, NAP car app, and NAP admin. With the combination of React-Native, Vue.Js, Redux, Expo, and Node, the same language could be used over the entire stack. This resulted in a lot of time being saved during development. By evaluating the app with two user tests and one field test, the application received a good usability score and they would like to book an autonomous car using the system. Some users experienced some problems with both the UI and functionality, meaning that there are some improvements that could be done. The different applications worked well for its purpose

and it is recommended to look into these systems if someone else were to develop an app.

There are many alternatives to such a solution. Although people did not have an opinion if it would make everyday life easier, there are some factors that could affect it. If people choose to use autonomous cars instead of a traditional taxi service, it is important that they feel safer. Other important factors are that it must be more efficient so that people have the opportunity to save time. Price is also a factor that affects success, This solution must be less expensive than the options, but it is unclear how much cheaper it must be. All of these factors can be solved with for example carpooling.

In order for getting users to use such a system, it is important that it earns their trust. It seems that people need more experience with autonomous cars in order to rely on them. An important factor here is that social relationships can play a major role, if someone else says they have had a good experience, this can affect another user. People are generally also a bit afraid of accidents. The app that was developed in this project did not prove to have such a great effect on whether people rely on autonomous cars or not, but there was feedback on some features that might play a role. Based on social relationships, feedback from other users could affect their trust. They would also like to know information about how long the car has been in effect and how many users it has driven. It could be concluded that such an app can have effect on peoples trust by implementing these features.

6.2 Future Work

In this section, future work for NAP App is recommended.

6.2.1 Choose destination using viewport

Most users requested this function. As it is now, it is only possible to choose a destination by typing an address. Most of the participants said that the overall usability could be improved by also using the viewport to choose a destination. It is already implemented when choosing pickup, and since React Native is used, the components could be re-used.

6.2.2 Carpooling

As efficiency and price proved to be success factors for the app, carpooling could be implemented to solve this. Also, some users recommended this when they gave feedback on improvements. It could be researched if the Google Maps API has some features already solved for using carpooling in an app.

6.2.3 Pricing

As price is a success factor and some users mentioned pricing as a feature to be added, it should be implemented. An example of a solution is when a user has performed a booking, a price can be visible along with the booking information.

6.2.4 Car information

Information about the car and what equipment it has proved to be important for some users. By displaying stats about the car on how long it has been operative, distance driven and how many other users that have booked it, seems to help the users trust in autonomous cars. One user would also like to know if it had the necessary equipment to bring along his kids.

6.2.5 Time estimate before choosing pickup location

By showing time estimate as the users move the viewport around the map could improve the usability of the system. This can be solved by for example implementing a component over the marker that displays how many minutes the nearest car would use to the location.

6.2.6 User Experiences

The most important factor to improve the general trust in autonomous cars seems to be social influence by other users. By implementing a feature where a user rates the car and the ride, store it in the database and make other users see the different feedbacks on a car before booking, could be a solution for this. A good example is examining how Uber solves this by rating the drivers.

Bibliography

Anicas, M., 07 2014. An introduction to oauth 2. https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2, accessed: 2019-05-20.

Apple, 2019. Human interface guidelines. https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/, accessed: 2019-05-13.

Armstrong, J., 11 2018. How do driverless cars work? https://www.telegraph.co.uk/cars/features/how-do-driverless-cars-work/, accessed: 2019-05-04.

Babich, N., 02 2018. A comprehensive guide to mobile app design. https://www.smashingmagazine.com/2018/02/comprehensive-guide-to-mobile-app-design/, accessed: 2019-05-09.

BBVA, 01 2019. Nine technology trends in 2019. https://www.bbva.com/en/nine-technology-trends-in-2019/, accessed: 2019-05-03.

Carson, B., 12 2017. Uber's self-driving cars hit 2 million miles as program regains momentum. https://www.forbes.com/sites/bizcarson/2017/12/22/ubers-self-driving-cars-2-million-miles/#15aafaa5a4fe, accessed: 2019-05-03.

Chong, H. F., Ng, D. W. K., 2016. Development of iot device for traffic management system. 2016 IEEE Student Conference on Research and Development (SCOReD).

Chrzanowska, N., 09 2017. Node.js vs. php: Which environment to choose for your next project? https://www.netguru.com/blog/nodejs-vs-php, accessed: 2019-05-20.

Cloud, G., 2019. Google cloud platform overview. https://cloud.google.com/docs/overview/, accessed: 2019-05-20.

Conger, K., 12 2018. Uber's driverless cars return to the road after fatal crash. https://www.nytimes.com/2018/12/20/technology/uber-driverless-cars-return.html, accessed: 2019-05-09.

Corona, B., 2019. 20+ web design statistics small business owners should know (2018). https://www.bluecorona.com/blog/20-web-design-facts-small-business-owners/, accessed: 2019-05-14.

Crist, P., Voege, T., 2018. Safer roads with automated vehicles?

Dandekar, K., Raju, B. I., Srinivasan, M. A., 2003. 3-d finite-element models ofhuman and monkey fingertips toinvestigate the mechanics oftactile sense. Journal of Biomechanical Engineering 125, 682–691.

DeviceSpecifications, 2018. Device specifications. https://www.devicespecifications.com/, accessed: 2019-05-12.

Dividend, L., 2018. The future of self-driving cars: How far can autonomous cars take us? https://www.landmarkdividend.com/self-driving-car/, accessed: 2019-05-04.

Dua, K., 04 2019. The complete guide to mobile app development: Web vs. native vs. hybrid. https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/, accessed: 2019-05-12.

Edmonds, E., 03 2019. Three in four americans remain afraid of fully self-driving vehicles. https://newsroom.aaa.com/2019/03/americans-fear-self-driving-cars-survey/, accessed: 2019-05-07.

Expo, 2019. Routing navigation. https://docs.expo.io/versions/latest/guides/routing-and-navigation/, accessed: 2019-05-21.

Express, 2019. Express. fast, unopinionated, minimalist web framework for node.js. https://expressjs.com/, accessed: 2019-05-20.

Facebook, 2019. Performance. https://facebook.github.io/react-native/docs/performance, accessed: 2019-05-15.

Gove, J., 2019. What makes a good mobile site? https://developers.google.com/web/fundamentals/design-and-ux/principles/, accessed: 2019-05-13.

Gozalvez, J., 2015. Samsung electronics sets 5g speed record at 7.5 gbs [mobile radio]. IEEE Vehicular Technology Magazine 10, 12–16.

Greenblatt, J. B., Saxena, S., 2015. Autonomous taxis could greatly reduce greenhouse-gas emissions of us light-duty vehicles.

Hendrickson, J., 01 2019. What are the different self-driving car "levels" of autonomy? https://www.howtogeek.com/401759/what-are-the-different-self-driving-car-levels-of-autonomy/, accessed: 2019-05-04.

HERE, 2018. Consumer acceptance of autonomous vehicles.

IBM, 2019a. Building trust in ai. https://en.wikipedia.org/wiki/Waymo, accessed: 2019-05-07.

IBM, 2019b. What is a cloud server? https://www.ibm.com/cloud/learn/what-is-a-cloud-server, accessed: 2019-05-12.

Iqbal, M., 05 2019. Uber revenue and usage statistics (2018). http://www.businessofapps.com/data/uber-statistics/, accessed: 2019-05-04.

john W. Creswell, 1998. ualitative inquiry and research design: Choosing among five traditions. Sage publications.

jwt.io, 2019. What is json web token? https://jwt.io/introduction/, accessed: 2019-05-21.

Kapoor, S., 2019. What is redux and who uses it? https://www.quora.com/What-is-Redux-and-who-uses-it/, accessed: 2019-05-15.

Kaur, K., Rampersad, G., 2018. Trust in driverless cars: Investigating key factors influencing the adoption of driverless cars. Journal of Engineering and Technology Management 48, 87–96.

Ketterman, S., 2019. Mobile ux design – best practices, constraints, and working with developers. https://www.toptal.com/designers/ux/mobile-ux-design-best-practices, accessed: 2019-05-14.

Keyence, 04 2019. What is an ultrasonic sensor? https://www.keyence.com/ss/products/sensor/sensorbasics/ultrasonic/info/, accessed: 2019-05-04.

Kiss, G., Berecz, E. C., 01 2019. Questions of security in the world of autonomous vehicles.

Koskinen, K. M., Lyrra, A., Mallat, N., Tuunainen, V., 01 2019. Trust and risky technologies: Aligning and coping with tesla autopilot.

Kruhlyk, Y., 2019. Expo vs vanilla react native: What to choose for your project. https://apiko.com/blog/expo-vs-vanilla-react-native/, accessed: 2019-05-15.

Lee, T. B., 07 2017. Waymo makes history testing on public roads with no one at the wheel. https://arstechnica.com/cars/2017/11/fully-driverless-cars-are-here/, accessed: 2019-05-03.

Lee, T. B., 12 2018. The hype around driverless cars came crashing down in 2018. https://arstechnica.com/cars/2018/12/uber-tesla-and-waymo-all-struggled-with-self-driving-in-2018/, accessed: 2019-05-03.

Luehrs, A., 08 2018. Seven problems self-driving cars need to overcome. https://www.smithslawyers.com.au/post/self-driving-car-problems, accessed: 2019-05-03.

Mathisen, H., 2018. A mobile application for bookingautonomous vehicles.

Meier, R., 2017. What is google's cloud platform? https://medium.com/@retomeier/what-is-googles-cloud-platform-d92a9c9e5e89, accessed: 2019-05-20.

Meteor, 2019. The fastest way to build javascript apps. https://www.meteor.com/, accessed: 2019-05-22.

Metz, C., 12 2018. A toaster on wheels to deliver groceries? self-driving tech tests practical uses. https://www.nytimes.com/2018/12/18/technology/driverless-mini-car-deliver-groceries.html?module=inline, accessed: 2019-05-09.

Morse, J. M., 1994. Designing funded qualitative research.

Mozilla, 2019a. Express/node introduction. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction, accessed: 2019-05-20.

Mozilla, 2019b. The websocket api (websockets). https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, accessed: 2019-05-20.

Naughton, K., 03 2019. Americans still fear self-driving cars. https://www.bloomberg.com/news/articles/2019-03-14/americans-still-fear-self-driving-cars, accessed: 2019-05-07.

Navada, Y., 01 2018. What are websockets? https://medium.com/front-end-weekly/what-are-websockets-7bf0e2e1af2, accessed: 2019-05-20.

Node, 2019. About node.js. https://nodejs.org/en/about/, accessed: 2019-05-20.

Nuro, 2019. Delivering the future of local commerce, autonomously. https://nuro.ai/, accessed: 2019-05-04.

Oates, B. J., 2006. Researching information systems and computing.

Ohio-University, 2019. The future of driving. https://onlinemasters.ohio.edu/blog/the-future-of-driving/, accessed: 2019-05-07.

Olchówka, B., 03 2014. Beware of feature overload: A case study. https://www.uxmatters.com/mt/archives/2014/03/beware-of-feature-overload-a-case-study.php, accessed: 2019-05-14.

Oracle, 03 2018. The main features of mysql. https://dev.mysql.com/doc/refman/8.0/en/features.html, accessed: 2019-05-20.

Panetta, K., 10 2018. Gartner top 10 strategic technology trends for 2019. https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2019/, accessed: 2019-05-03.

Patro, N., 03 2018. Choose the best—native app vs hybrid app. https://codeburst.io/native-app-or-hybrid-app-ca08e460df9, accessed: 2019-05-12.

Paul, R., 08 2018. How bcryptjs works. https://medium.com/@paulrohan/how-bcryptjs-works-90ef4cb85bf4, accessed: 2019-05-20.

Perez, S., 2017. U.s. consumers now spend 5 hours per day on mobile devices. https://techcrunch.com/2017/03/03/u-s-consumers-now-spend-5-hours-per-day-on-mobile-devices/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=7bVHOQMbs5-z7CY4GC0oiA, accessed: 2019-05-09.

Powell-Morse, A., 12 2016. Iterative model: What is it and when should you use it? https://airbrake.io/blog/sdlc/iterative-model, accessed: 2019-05-31.

React-Navigation, 2019. Getting startedn. https://reactnavigation.org/docs/en/getting-started.html, accessed: 2019-05-21.

Rouse, M., 05 2018. self-driving car (autonomous car or driverless car). https://searchenterpriseai.techtarget.com/definition/driverless-car, accessed: 2019-05-04.

Salinas, S., 12 2018. Uber's self-driving cars are back on the road, nine months after a fatal accident. https://www.cnbc.com/2018/12/19/uber-resumes-self-driving-car-tests-nine-months-after-fatal-accident.html, accessed: 2019-05-04.

Shiff, L., Rowe, W., 03 2018. Nosql vs sql: Examining the differences and deciding which to choose. https://www.bmc.com/blogs/sql-vs-nosql/, accessed: 2019-05-20.

Silva, M. H. D., 2019. Creating a secure rest api in node.js. https://www.toptal.com/nodejs/secure-rest-api-in-nodejs, accessed: 2019-05-20.

Stewart, J., 03 2018. Tesla's autopilot was involved in another deadly car crash. https://www.wired.com/story/tesla-autopilot-self-driving-crash-california/, accessed: 2019-05-04.

Techopedia, 2019. Three-tier client/server. https://www.techopedia.com/definition/23813/three-tier-clientserver, accessed: 2019-05-20.

Thomas, N., 2015. How to use the system usability scale (sus) to evaluate the usability of your website. https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-o accessed: 2019-05-21.

Uber, 2019. A guide for how to use uber. https://www.uber.com/us/en/ride/how-it-works/, accessed: 2019-05-09.

usability.gov, 2019. System usability scale (sus). https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html, accessed: 2019-05-21.

Vue, 2019a. Vue in depth. https://vuejs.org/v2/guide/reactivity.html, accessed: 2019-05-20.

Vue, 2019b. What is vue.js? https://vuejs.org/v2/guide/, accessed: 2019-05-20.

W3C, 2019. Web content accessibility guidelines 2.0. https://www.w3.org/TR/WCAG20/, accessed: 2019-05-14.

WHO, 2011. World report on disability.

Wikipedia, 04 2019a. Lidar. https://en.wikipedia.org/wiki/Lidar, accessed: 2019-05-04.

Wikipedia, 05 2019b. List of self-driving car fatalities. https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities, accessed: 2019-05-07.

Wikipedia, 2019c. Lyft. https://en.wikipedia.org/wiki/Lyft, accessed: 2019-05-31.

Wikipedia, 05 2019d. Self-driving-car. https://en.wikipedia.org/wiki/Self-driving_car, accessed: 2019-05-04.

Wikipedia, 05 2019e. Trust (social science). https://en.wikipedia.org/wiki/Trust_(social_science), accessed: 2019-05-07.

Wikipedia, 2019f. Uber. https://en.wikipedia.org/wiki/Uber, accessed: 2019-05-08.

Wikipedia, 2019g. Waymo. https://newsroom.aaa.com/2019/03/americans-fear-self-driving-cars-survey/, accessed: 2019-05-08.

Wodehouse, C., 2019. A beginner's guide to back-end development. https://www.upwork.com/hiring/development/a-beginners-guide-to-back-end-development/, accessed: 2019-05-12.

Xplenty, 09 2017. The sql vs nosql difference: Mysql vs mongodb. https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2, accessed: 2019-05-12.

Appendix A - Persona

Name	John
Gender	Male
Age	40
Occupation	Financial Analyst
location	London, UK

Description

John is a financial analyst who has a hectic lifestyle with lots of meetings and he also works a both with financial consulting with his customers. Even though he is not good with technology he isn't hesitating with learning new things. Since he has a busy schedule he always plans his days.

User Goals

- As I have an hectic lifestyle, I want to spend short time booking a car
- I should have an option to always change my pickup or destination point.
- As I am weak in directions I want to track my ride.

Appendix B - Github Repositories

- NapApp V2 - `https://github.com/matwii/NapApp_V2`
- Autonomous Car App - `https://github.com/matwii/NapApp_Car`
- NapApp Server - `https://github.com/matwii/NapApp_server`
- Admin app - `https://github.com/matwii/nap-app-admin`

Appendix C - Nap App user test tasks

Nap App user test tasks

Nap-App

The software you are about to test is a mobile application used for booking autonomous cars. The idea is that many people are sharing a pool of self-driving cars, and they use this app for either driving them somewhere or transporting materials. The users decides where they want to go and then the app will allocate a car and find the best possible directions for them. This app is meant to be used for real cars, but they are not available yet, therefore a person simulating a car will move around depending on where the car is supposed to drive.

Tasks

Do the following tasks as well as you can. The interviewer cannot answer any questions. The whole purpose is to see how well you perform the tasks without any help. The interviewer will write down notes if there are any possible faults. Keep in mind that it is not you we are testing but the application. We are not keeping any of the information, all the user details will be deleted after the test.

- If you have a Google or LinkedIn account create a user in Nap-App using these tools.
- If you don't a Google or LinkedIn account, create a user with your email and password.
- If creation of a user was a success, sign out.
- Try to book a car from your current position to Julianus Holms veg 26, Trondheim.
- If the car is at your destination, confirm that.
- Confirm that you arrived to the destination.
- You don't want a ride from your current position, but from Lerkendal Stadium. Travel to Munkegata, Trondheim
- First you want to book a ride from your current position and travel to Prinsens gate, Trondheim, but you change your mind and want to travel from Lerkendal Stadium instead of your current position.
- Find your booking history.

Well done! Now, please answer the questionnaire

Appendix D - Google Cloud billing for May 2019

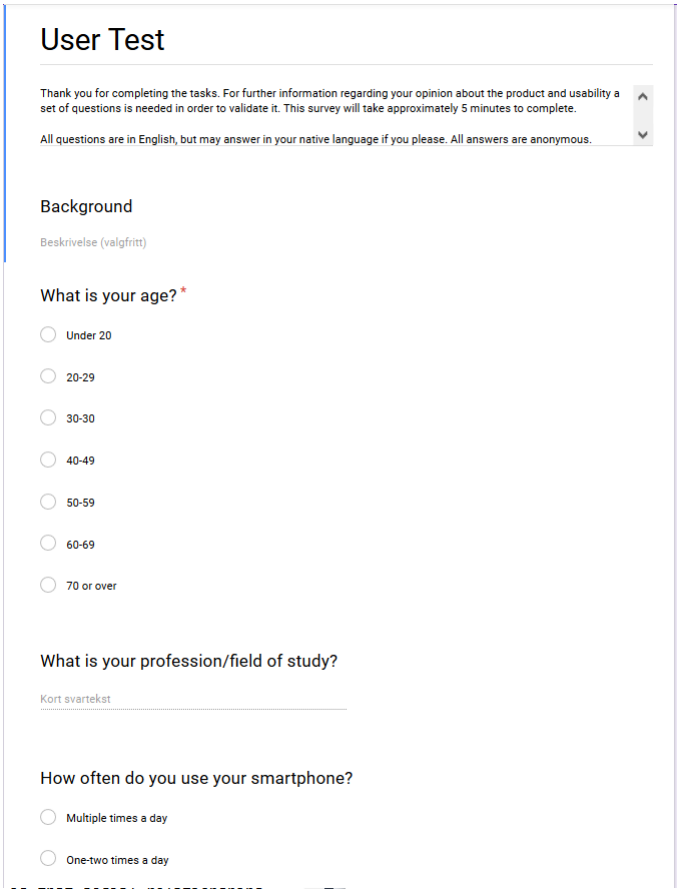
May 1 – 28, 2019

Ending balance: −NOK 0.24

Date	Description	Amount (NOK)
May 1 – 28, 2019	Credit Maps Free Tier (Source:nap-app [nap-app-229311])	−NOK 48.67
May 1 – 28, 2019	Places API Autocomplete without Places Details - Per Session: 1 Count [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 0.15
May 1 – 27, 2019	Credit FreeTrial:Credit-016791-C3862A-D6BA48 (Source:nap-app [nap-app-229311])	−NOK 817.65
May 1 – 27, 2019	Cloud SQL Storage PD SSD for MySQL DB in Finland: 8.71 Gibibyte-months [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 14.11
May 1 – 27, 2019	Cloud SQL MySQL DB generic Micro instance with burstable CPU running in Finland (with 30% promotional discount): 648 Hours [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 65.12
May 1 – 27, 2019	App Engine Flex Instance RAM EMEA: 1296.594 Gibibyte-hours [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 87.61
May 1 – 27, 2019	App Engine Flex Instance Core Hours EMEA: 1296.594 Hours [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 650.34
May 1 – 27, 2019	Cloud Storage Multi-Regional Storage Europe: 0.437 Gibibyte-months [Currency conversion: USD to NOK	NOK 0.10

May 1 – 27, 2019	Geocoding API Geocoding: 557 Counts [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 24.13
May 1 – 27, 2019	Directions API Directions: 194 Counts [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 8.41
May 1 – 27, 2019	Places API Places Details: 31 Counts [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 4.57
May 1 – 27, 2019	Places API Contact Data: 35 Counts [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 0.91
May 1 – 27, 2019	Places API Atmosphere Data: 35 Counts [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 1.51
May 1 – 27, 2019	Places API Autocomplete - Per Request: 312 Counts [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 7.65
May 1 – 26, 2019	Cloud SQL Storage PD Snapshot in Finland: 0.445 Gibibyte-months [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 0.34
May 1 – 26, 2019	Cloud SQL Network Internet Egress from Finland to EMEA: 0.014 Gibibytes [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 0.02
May 1 – 26, 2019	Places API Places - Nearby Search: 4 Counts [Currency conversion: USD to NOK using rate 8.663] (Source:nap-app [nap-app-229311])	NOK 1.11
		Starting balance: NOK 0.00

Appendix E - User test questionnaire



User Test

Thank you for completing the tasks. For further information regarding your opinion about the product and usability a set of questions is needed in order to validate it. This survey will take approximately 5 minutes to complete.

All questions are in English, but may answer in your native language if you please. All answers are anonymous.

Background

Beskrivelse (valgfritt)

What is your age? *

- Under 20
- 20-29
- 30-30
- 40-49
- 50-59
- 60-69
- 70 or over

What is your profession/field of study?

Kort svartekst

How often do you use your smartphone?

- Multiple times a day
- One-two times a day

- [ ] 4-5 times a week

Do you have any prior experience using Taxi booking apps?

- [ ] Yes
- [ ] No

Do you have any driving experience with Advanced driver-assistance systems (ADAS) or Autonomous cars? *

- [ ] Yes
- [ ] No
- [ ] Some
- [ ] I don't know

Would you trust a fully autonomous car to drive you somewhere? *

- [ ] Yes
- [ ] No
- [ ] Maybe

Can you list a reason for why you trust/don't trust autonomous cars?

Lang svartekst

Usability

In this part you answer questions regarding how usable this app was. This is a standardized system Usability Scale (SUS), describing how useful the system is

describing how useful the system is.

I think that I would like to use this system frequently

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I found this system unnecessarily complex.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I thought the system was easy to use.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I think I would need the support of a technical person to be able to use this system.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I found the various functions in this system were well integrated.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I thought there was too much inconsistency in this system.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I would imagine that most people would learn to use this sytem very quickly.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I found the system very cumbersome to use.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I felt very confident using the system

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

I needed to learn a lot of things bnefore I could get going with this system.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongy Agree

About Nap-App

In this part, you will answer some questions regarding the application.

This product will make my everyday life easier.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I would book an autonomous car using this system.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

This system influenced my trust in autonomous cars.

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Why would you use this service instead of a traditional taxi booking service?

Lang svartekst

Is there anything that should be added or removed to make the product more attractive for you to use?

Lang svartekst

Are there any features that could be implemented to affect your trust in autonomous cars?

Lang svartekst

Appendix G - Demonstration video

- NAPApp V2 - `https://vimeo.com/339766772`
- NAPApp car app - `https://vimeo.com/339766642`

Appendix H - Answers from questionnaire

- Answer spreadsheet - https://docs.google.com/spreadsheets/d/1_U89Z0q5jR-UPYz7Pkt6YxBEB0ffCtb-9zYsKBwbkXM/edit?usp=sharing