NTNU
Norwegian University of
Science and Technology

A mobile application for booking autonomous vehicles

Combining the sharing economy and self-driving cars

Hanne Mathisen

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Frank Lindseth, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Fully autonomous vehicles eliminate the human driver of cars by taking care of all driving tasks. The cars can view 360 degrees of their surroundings and make decisions based on them, which can optimize traffic and fuel usage, and also contribute to safer traffic and improve mobility in the society. These cars can be combined with the sharing economy, which enables the sharing of assets and resources among people to optimize usage, by offering a pool of autonomous vehicles as an on-demand service to members. For such a product to be used, the public must have a positive attitude towards it, and it must also be able to satisfy the needs of the society to replace personal vehicles and public transport.

This thesis evaluates how the combination of autonomous vehicles and the sharing economy can affect the society, and if a mobile application can take care of the communication between the cars and the end users. A mobile application to handle all car booking functionality was developed, including car distribution and route planning. The application is only an initial version, so more and expanded functionalities are needed for the application to fulfill all requirements.

Two user tests were conducted on the application, with two different sets of users. The first and qualitative test resulted in some suggestions for usability improvements to the application and a preliminary usability score. The second and quantitative test was conducted after fixing usability problems and adding functionality, and resulted in a final usability score and some indications of the public attitude. The results indicate that the application has potential and is easy to use, but with some room for improvement, and the general attitude of the public is positive, but an attitude change is needed for people to change from personal vehicles to shared autonomous vehicles.

Sammendrag

Autonome kjøretøy eliminerer den menneskelige sjåføren av biler ved å ta hånd om alle kjøreoppgaver. Bilene kan se 360 grader av omgivelsene og tar beslutninger basert på dem, noe som kan optimalisere trafikk og drivstoffbruk, og også bidra til tryggere trafikk og bedre mobilitet i samfunnet. Disse bilene kan kombineres med delingsøkonomien, som gjør det mulig å dele ressurser blant mennesker for å oppnå optimal bruk, ved å tilby et utvalg av autonome kjøretøyer som en on-demand biltjeneste til medlemmer. For at et slikt produkt skal bli brukt, må publikum ha en positiv holdning til det, og det må også kunne tilfredsstille samfunnets behov for å erstatte private personbiler og offentlig transport.

Denne oppgaven evaluerer hvordan kombinasjonen av autonome kjøretøyer og delingsøkonomien kan påvirke samfunnet, og om en mobilapplikasjon kan ta hånd om kommunikasjonen mellom bilene og sluttbrukerne. En mobilapplikasjon for å håndtere all funksjonalitet for bilreservasjoner ble utviklet, og inkluderer både bildistribusjon og ruteplanlegging. Applikasjonen er en første versjon, så det er behov for flere og utvidede funksjoner for at applikasjonen skal kunne oppfylle alle krav.

To brukertester av applikasjonen ble utført, med to forskjellige sett av brukere. Den første kvalitative testen resulterte i flere forslag til forbedringer av brukervennlighet og en foreløpig brukbarhetsscore. Den andre kvantitative testen ble utført etter å ha fikset noen brukervennlighetsproblemer og lagt til ny funksjonalitet, og resulterte i en endelig brukbarhetsscore og noen indikasjoner på den offentlige holdningen. Resultatene indikerer at applikasjonen har potensial og er lett å bruke, men med noe rom for forbedring, og den generelle holdningen til publikum er positiv, men en holdningsendring er nødvendig for at folk skal bytte ut personbiler med delte autonome biler.

Preface

This is the Master's thesis as the final part of the study programme Computer Science at the Norwegian University of Science and Technology (NTNU). The work was carried out during the spring semester of 2018.

I would like to thank my supervisor, Frank Lindseth, for his guidance throughout the project. I would also like to express my gratitude to everyone who participated in the user testing, and finally, I would like to thank my mother, both for her support during these years, and also for her help by making her coworkers test my application.

Hanne Mathisen

Trondheim, June 2018

Table of Contents

Abstract **i**

Sammendrag **iii**

Preface **v**

Table of Contents **x**

List of Tables **xi**

List of Figures **xiv**

Abbreviations **xv**

1 Introduction **1**

1.1 Motivation and project description 1

1.2 Project goals and research questions 2

1.3 Contributions 2

1.4 Thesis outline 2

2 Background **5**

2.1 Car usage and road safety 5

2.2 Sharing economy 9

2.2.1 Shared mobility 11
2.3 Autonomous vehicles 12
2.4 Usability 16
2.4.1 System Usability Scale 17
2.5 Shared mobility and autonomous vehicles: Existing solutions 17
2.5.1 Waymo 17
2.5.2 Uber 19
2.5.3 BlaBlaCar 20
2.5.4 ZipCar 21
2.5.5 Bilkollektivet 21
2.5.6 Nabobil 21
2.6 Mobile app development: Platform and technologies 22
2.6.1 Mobile application 22
2.6.2 Web application 24
2.6.3 Choosing the best tools 26

3 Methodology and implementation **27**
3.1 Overview 27
3.2 Technology 28
3.2.1 React Native 29
3.2.2 Redux 30
3.2.3 MySQL 33
3.2.4 PHP 34
3.3 Requirements 35
3.4 Architecture 35
3.4.1 React-Redux 37
3.4.2 Client-Server 37
3.5 Implementation 38

3.5.1 Database . . . 39
3.5.2 REST API . . . 40
3.5.3 NAP App . . . 41
3.6 Evaluation . . . 47
3.6.1 Number of testers . . . 47
3.6.2 First test . . . 47
3.6.3 Second test . . . 48

4 Results **49**
4.1 NAP App . . . 49
4.1.1 Mobile devices . . . 49
4.1.2 First iteration views . . . 49
4.1.3 Second iteration improvements . . . 52
4.1.4 User flow diagram . . . 54
4.2 Evaluation . . . 54
4.2.1 First user test . . . 54
4.2.2 Second user test . . . 57

5 Discussion **63**
5.1 RQ1: How can autonomous cars and the sharing economy satisfy the needs of the society? . . . 63
5.2 RQ2: To what extent is a mobile application perceived as a usable tool for booking cars? . . . 65
5.3 RQ3: To what extent is it feasible to develop a mobile application for simulation of car booking? . . . 67
5.4 Limitations . . . 68

6 Conclusions and future work **69**
6.1 Conclusion . . . 69

6.2 Future work 70
6.2.1 iOS 70
6.2.2 Cars 70
6.2.3 Ridesharing 71
6.2.4 Car distribution 71
6.2.5 User profiles and login 71
6.2.6 Pricing and payments 72
6.2.7 Addresses 72
6.2.8 Rides 72
6.2.9 Web service for admins 72

Bibliography **75**

A GitHub repositories **83**
A.1 REST API 83
A.2 NAP App 83

B Get rides REST API **85**

C MapComponent **87**

D User test tasks **89**

E Questionnaire for user test 1 **91**

F Questionnaire for user test 2 **97**

G Demonstration video **103**

H Responses to user tests **105**
H.1 User test 1 105
H.2 User test 2 105

List of Tables

2.1 SAE's six levels of automation 13

3.1 Functional requirements 36

3.2 Non-functional requirements 36

4.1 Tested mobile devices 50

4.2 SUS score by discipline, first user test 56

4.3 Participants by discipline, second user test 58

4.4 SUS score per question, second user test 59

4.5 SUS score per discipline, second user test 59

4.6 Average and median for product questions, per car owner status 60

List of Figures

2.1 Registered private cars in Norway since 1951 6

2.2 Total vehicle-kilometres in Norway since 2005 7

2.3 Registered private cars and average vehicle-kilometres per car in Norway since 2005 7

2.4 Fatalities and severe injuries in Norwegian road traffic accidents from 1991 to 2017 8

2.5 Waymo's FireFly 18

3.1 Car booking use case 28

3.2 React Native bridge 30

3.3 React Native component on iOS 32

3.4 React-Redux architecture 37

3.5 Three-tier client-server architecture 38

3.6 Entity-relationship diagram 40

4.1 Requesting ride views, iteration 1 51

4.2 Editing destination views, iteration 1 51

4.3 Ride simulation views, iteration 1 52

4.4 Requesting ride views, iteration 2 53

4.5 Ride simulation views, iteration 2 54

4.6 User flow diagram 55
4.7 Answers regarding easier life, first user test 57
4.8 Answers regarding potential signup, first user test 57
4.9 Participant car usage, user test 2 58

Abbreviations

ADAS	=	Advanced Driver Assistance System
ADS	=	Automated Driving System
API	=	Application Programming Interface
AV	=	Autonomous Vehicle
CSS	=	Cascading Style Sheets
DBMS	=	Database Management System
DOM	=	Document Object Model
FR	=	Functional Requirement
HTML	=	HyperText Markup Language
HTTP	=	HyperText Transfer Protocol
ICT	=	Information and Communication Technology
IDE	=	Integrated Development Environment
JSON	=	JavaScript Object Notation
JSX	=	JavaScript XML
NAP	=	NTNU Autonomous Perception
NFR	=	Non-Functional Requirement
NHTSA	=	National Highway Traffic Safety Administration
REST API	=	Representational State Transfer Application Programming Interface
SQL	=	Structured Query Language
SUS	=	System Usability Scale
UI	=	User Interface
UN	=	United Nations
URI	=	Uniform Resource Identifier
XAML	=	eXtensible Application Markup Language
XML	=	eXtensible Markup Language

Chapter 1

Introduction

1.1 Motivation and project description

People all over the world are driving more and more, and the number of cars in the world continues to increase (World Health Organization 2015). However, the average distance driven yearly by each car decreases, and cars are on average parked more than 90% of the time (Bates & Leibling 2012, Barter 2013, Statistics Norway 2018*d*). The sharing economy, which enables the sharing of assets and resources to optimize usage, has huge potential in decreasing the number of cars needed, and can also be economically beneficial for people choosing to share cars rather than own a private car (Hamari et al. 2016, Schor 2016).

Developments in technology and communications have led to advances in the car industry, with tools to help the driver with some of the driving tasks, such as automatic transmission, cruise control and parking assistants (Rödel et al. 2014, NHTSA 2018). Several companies and researchers are working on the next advancement, the fully autonomous vehicle, which takes care of all driving tasks and enables the occupants to spend commuting and traveling time doing other things. Self-driving cars can also eliminate human error and choice made in traffic, which is the cause of most serious accidents, and thereby significantly reduce the number of people killed in traffic (Fagnant & Kockelman 2015, World Health Organization 2015, NHTSA 2018).

Combining autonomous cars with the sharing economy can lead to significant benefits for the society and can change the way cars are used and owned, by being able to share a pool of cars that can pick up and drop off the user when and where needed. This business model can

increase mobility in the society, improve traffic and decrease congestion, reduce the need for parking spaces and have positive effects in communities and for the environment (Bertoncello & Wee 2015, Fagnant & Kockelman 2015, NHTSA 2018). A mobile app to handle the communication between end users and cars can make the realization of this more manageable, and could potentially also make people's life easier. This project will try to develop such an app and investigate how people's attitudes to both the app and the product is.

1.2 Project goals and research questions

The goal of this thesis is to research how autonomous vehicles and the sharing economy can be combined, the effect this could have for people's life and the society in general, as well as to develop and test an app for booking autonomous cars, to investigate how people's attitude towards this business model is.

The following research questions (RQs) are defined for the project:

RQ1: How can autonomous cars and the sharing economy satisfy the needs of the society?

RQ2: To what extent is a mobile application perceived as a usable tool for booking cars?

RQ3: To what extent is it feasible to develop a mobile application for simulation of car booking?

1.3 Contributions

The contributions from this project include a first version of a mobile application to handle ride booking for autonomous cars, as well as suggestions to how to further implement this app. Further, this thesis also provides some insights about how the product of sharing a pool of on-demand autonomous vehicles is perceived by the public, and how this product could help change the way the society solves transportation.

1.4 Thesis outline

This thesis contains six chapters, where the remaining five are structured as follows:

Chapter 2 - Background introduces the theory and initial research for the material used in the project.

Chapter 3 - Methodology and implementation presents the technology, implementation details, and methodology used for the development of the app and the execution of the user testing.

Chapter 4 - Results presents the results of the development and the user tests.

Chapter 5 - Discussion discusses the results presented and the research questions.

Chapter 6 - Conclusions and future work presents the conclusion of the project as well as recommendations and suggestions for future work.

Chapter 2

Background

This chapter presents the background information needed to design and develop a mobile application to implement the sharing economy for an autonomous vehicle. The first section presents some statistics about car usage in Norway today. The next section describes the sharing economy, especially the concept of shared mobility, before the third section briefly explores autonomous vehicles and the potential benefits and challenges related to these new technologies. The fourth section presents the concept of usability. Further, the fifth section presents some existing solutions, before some popular platforms and technologies used for mobile application development are considered in the sixth section of this chapter.

2.1 Car usage and road safety

People are very dependent of their cars, and in Norway, with big distances and limited access to public transportation outside the big cities, as much as 88% have access to at least one car in their household (Hjorthol et al. 2014, p. 7). The amount of registered private vehicles in Norway has experienced a linear growth the past 60 years, shown in figure 2.1, and reached more than 2.7 million cars by the end of 2017 (Statistics Norway 2018*b*). Many people use cars often, but mostly as the driver of the car; 55% of all travelling in 2014 was done as the driver of a car, whereas only 8% was travelled as a passenger in a car (Hjorthol et al. 2014, p. 25). The car occupancy rate is therefore low, with an average of 1.55 people in each car (Nordbakke et al. 2014, p. 9). This of course varies depending on what the purpose of the travelling is, and is significantly lower on car rides associated with work, with a car occupancy rate of 1.15. Work

travelling is also the most predictable type of travelling; 31% of work trips starts between 7:00 and 9:00 in the morning, and 25% starts between 15:00 and 17:00 (Hjorthol et al. 2014, p. 39).

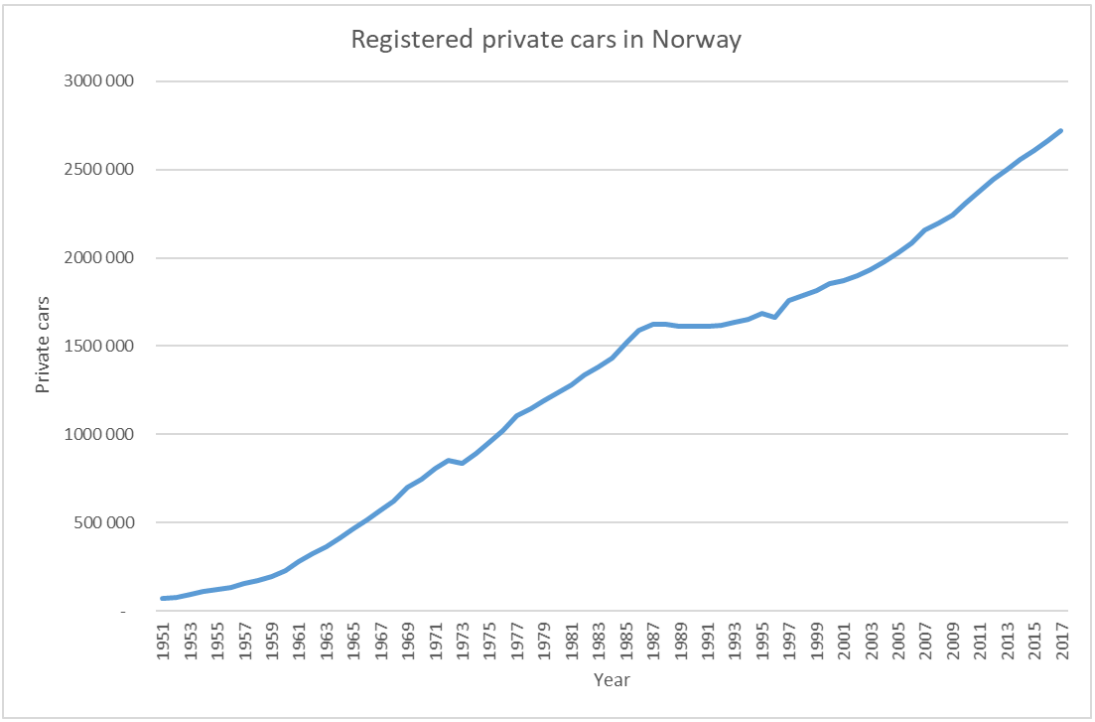


Figure 2.1: Registered private cars in Norway since 1951
Source: Statistics Norway (2018*b*)

More kilometres are also driven each year, but this trend is slowing down, which can be seen in figure 2.2. A vehicle-kilometer is a kilometre driven, with or without load, by a car, and in 2017, a total of 35.3 billion vehicle-kilometres were driven by private cars in Norway (Statistics Norway 2018*d*). Although the total number of vehicle-kilometres have increased slightly each year, the number of cars have increased even more, resulting in a constant decrease of average vehicle-kilometres per car, which can be seen in figure 2.3. The average distance driven by a private car in Norway in 2017 was 12 228 km. (Statistics Norway 2018*d*). This can be used to find the average amount of time each car is used, and consequently, the average amount of time each car is parked.

To find the average car usage, the average speed is also needed. Speed limits in Norway are usually 30 km/h within residential areas, 50 km/h on roads in urban areas and 80 km/h in more sparsely populated areas (Høye et al. 2012, chap. 3.11). As it is challenging to find a correct number for the average speed of all cars, the calculations will be done for 30 km/h, but the average speed of all car rides could be higher than this. Assuming an average speed of 30 km/h,

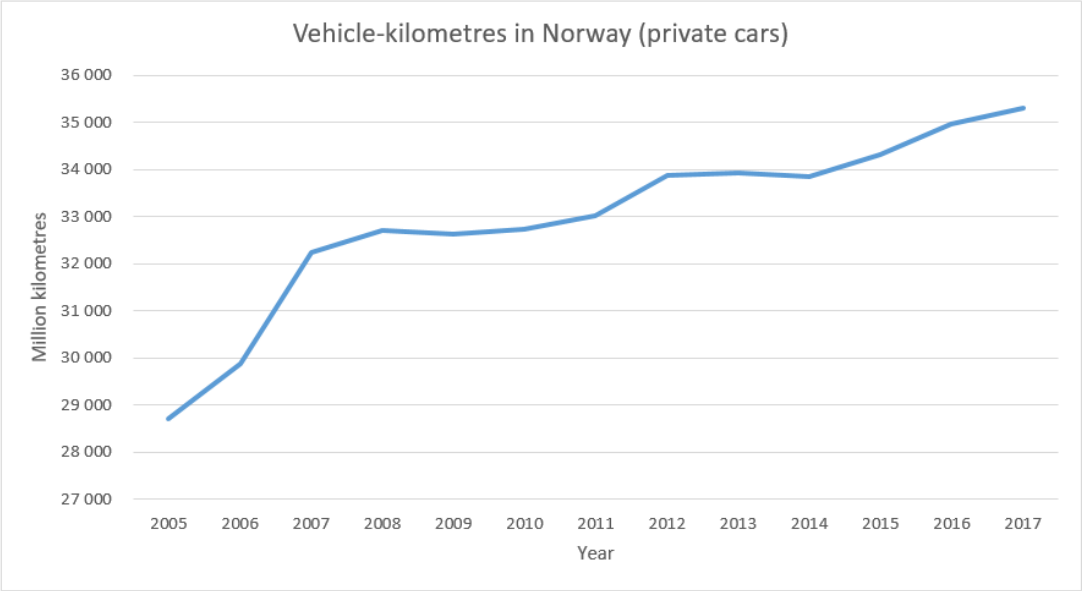


Figure 2.2: Total vehicle-kilometres in Norway since 2005
Source: Statistics Norway (2018*d*)

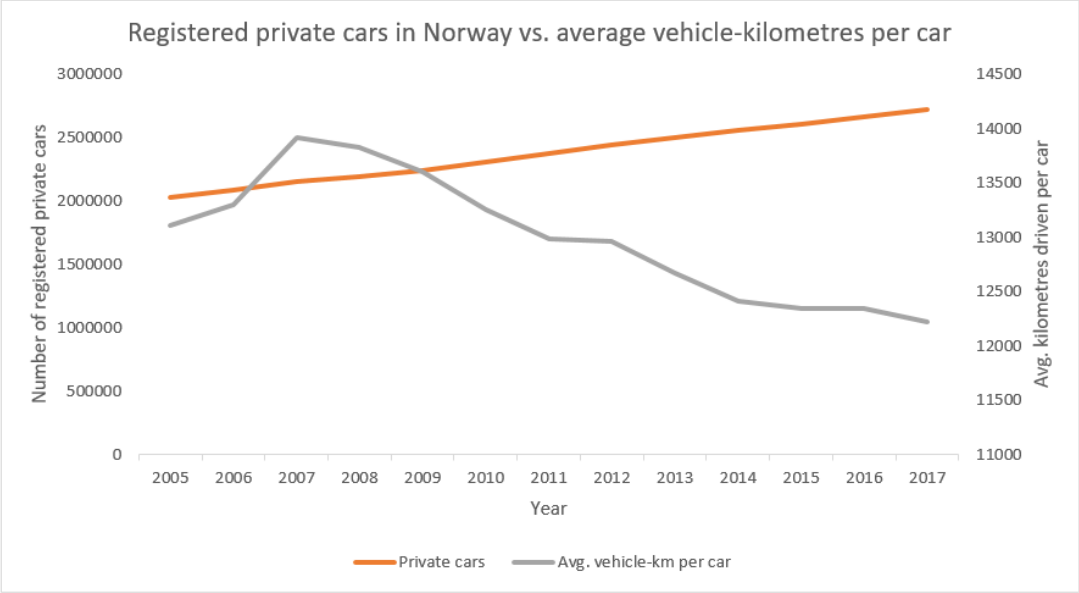


Figure 2.3: Registered private cars and average vehicle-kilometres per car in Norway since 2005
Source: Statistics Norway (2018*b*,*d*)

this gives a total time of $\frac{12228km}{30km/h} = 407.60$ hours of car usage in 2017. Since a year has 8760 hours, a personal car in Norway is used $\frac{407.60h}{8760h} = 4.65\%$ of the time on average. Usually, private cars are not used during the night, so for a more realistic view, one can assume that a normal car will be used between the hours of 7:00 and 21:00, in total 14 hours each day. This means that there are 5110 active hours each year, which gives a new percentage of $\frac{407.60h}{5110h} = 7.98\%$, with an average speed of 30 km/h. Even with a low average speed, this indicates that a private

car in Norway is parked for more than 90% of the day (in addition to 10 hours each night). This confirms international research on the topic, which found that cars are, on average, parked around 95% of the time (Bates & Leibling 2012, Barter 2013).

Although people in Norway drive more than earlier, the number of people killed and seriously injured in traffic is decreasing (Statistics Norway 2018*c*). The decrease can be seen in figure 2.4, going from 304 killed and 1148 severely injured in 1999 to 106 killed and 665 severely injured in 2017. In 2001, Vision Zero was adopted by the Norwegian government and the Norwegian Public Roads Administration, with the goal of no fatalities or severe injuries in traffic (Elvebakk & Steiro 2007). This has led to more legislation and focus on safer roads, cars and traffic behavior, as well as a higher awareness of traffic safety among the population.

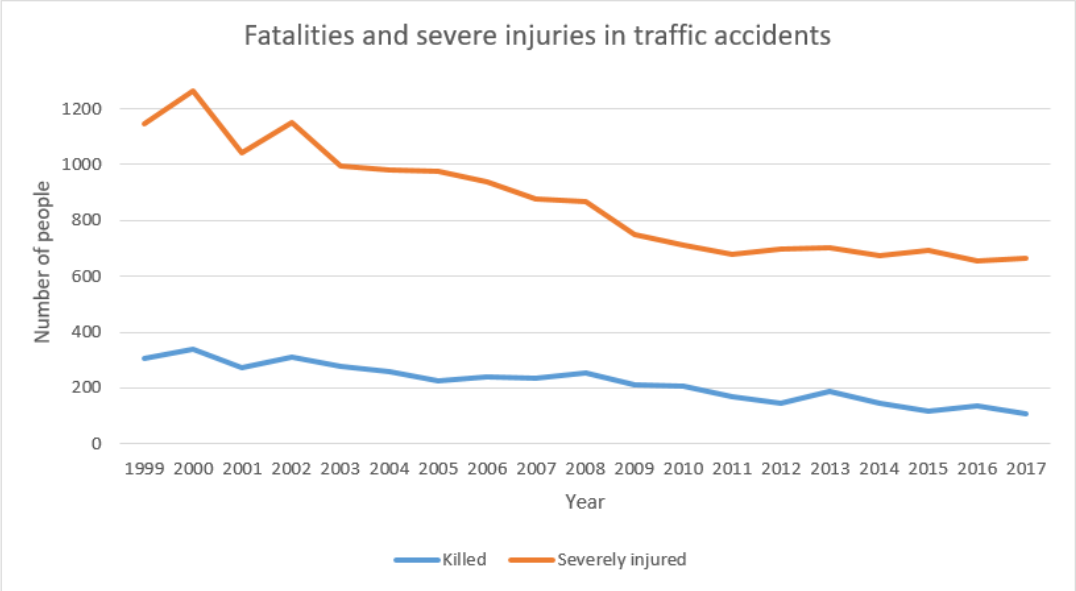


Figure 2.4: Fatalities and severe injuries in Norwegian road traffic accidents from 1991 to 2017
Source: Statistics Norway (2018*c*)

Globally, traffic accidents are a leading cause of death, in fact it is the most common cause of death for people aged 15-29 (World Health Organization 2015, p. x). The average for the world is 17.4 road traffic fatalities per 100 000 population (World Health Organization 2015, p. 6), and Norway is therefore one of the safest countries to drive in, with a rate of 2.1 per 100 000 population, based on the 109 traffic fatalities and a population of 5.3 million in 2017 (Statistics Norway 2018*a*). In many countries, such as Norway, the number of traffic-related fatalities has decreased, but in total, global deaths have plateaued the past ten years (World Health Organization 2015, p. 2). Since there has been a global population and motorization

increase, lives have probably been saved the past few years, but the number of traffic-related deaths is still rising, especially in low-income countries.

The World Health Organization further identifies five key behavioral risk factors in traffic: speed, drunk-driving, lack of usage of motorcycle helmets, lack of seat belt usage, and child restraints, and they also talk about distracted driving, from, e.g., mobile phones, and safety of cars and roads as risks (World Health Organization 2015). Out of the 17 Sustainable Development Goals and 169 targets set by the United Nations (UN) in 2015, two targets are related to road safety and mobility (World Health Organization 2015, p. 3). Target 3.6 aims to ”halve the number of global deaths and injuries from road traffic accidents” (The Global Goals 2018*b*) by 2020, and target 11.2 aims to ”provide access to safe, affordable, accessible and sustainable transport systems for all, improving road safety, notably by expanding public transport, with special attention to the needs of those in vulnerable situations, women, children, persons with disabilities and older persons” (The Global Goals 2018*a*) by 2030. Road and traffic safety are therefore recognized as important factors when it comes to achieving a better world for everyone.

2.2 Sharing economy

During the past few years, general views on consumerism have changed, leading to alternative ways to consume goods and services. Drivers of this attitude change include the global financial crisis of 2008, increasing awareness of climate changes, as well as a greater desire for social embeddedness and public consumption (Heinrichs 2013, Hamari et al. 2016). The sharing economy has emerged as a result of both the aforementioned as well as developments in information and communications technologies (ICT). As people also are becoming more concerned about the ecological, societal and developmental consequences of the market society, the sharing economy has become a more attractive alternative for many consumers.

The sharing economy includes several very different activities, so finding one accurate definition is difficult. Hamari et al. (2016) looks at the sharing economy as an umbrella concept that covers all ICT technologies and developments supporting the sharing of consumption of goods and services through online platforms. Heinrichs (2013) further elaborates that the sharing economy is referring to the economic and social systems that use information technology to give both individuals, corporations, non-profits and government information that enables the

distribution, sharing and reuse of excess goods and services. The sharing economy uses market intelligence to create a more collaborative and sustainable society, which could disrupt the mainstream consumerism while also optimizing resource use and improving social embeddedness. Examples of sharing economy activities include bike- and car sharing schemes and peer-to-peer activities like swapping clothes or renting rooms.

This new economic system has attracted much attention, and people are divided on how they feel about it. Boosters mention several benefits, such as empowerment of ordinary people, more efficiency in the society and a lower carbon footprint, while critics talk about how sharing economy technologies are exploitative and more about economic self-interest than sharing (Schor 2016). The sharing economy could mitigate problems such as poverty, hyperconsumerism, and pollution in today's capitalist society, but as long as new products are on the market with a low price, some people might not be interested in sharing and rather own products themselves. (Hamari et al. 2016).

Historically, people have chosen not to consume ethically because of economic and institutional reasons; the price for ethical products have been significantly higher, and legislators have not tried to control the consumption, manufacturing or imports of unethical products, with for example regulations or taxes (Hamari et al. 2016). The sharing economy could overcome these problems, and economic gain could be a potential motivation to participate in sharing. Some platforms can distribute the value across the supply chain, which would give less profit to middlemen, and services such as Airbnb, where people can rent out a spare bedroom or their entire home for a short period, usually requires fewer fees than a hotel would (Schor 2016). The sharing economy is also a way for people to earn money by renting out their property, which could be a strong motivational factor for some. Another motivational factor is the enjoyment of the activity (Schor 2016). The sharing economy can contribute to better social connections in the community by people getting to know their neighbors, and can also be a way to make new friends. Couchsurfers have reported that they grew close friendships with the people whose couch they borrowed, and co-riding is also a way to get to know new people. The commitment to social transformation can also be a motivational factor, many highlight the value of collaboration and sharing as they are critical of capitalism and how the market works.

2.2.1 Shared mobility

Shared mobility covers all the sharing economy activities carsharing, ridesharing and bike sharing. Co-owning properties have been widely accepted for decades, but sharing cars have not been popular until recent years (Cohen & Kietzmann 2014). In big cities, with a high and growing population density, sharing vehicles and riding together is especially interesting, as this could reduce traffic, congestion and pollution problems. Traditional transportation policies have focused on reducing commute times, but emerging technologies challenge this perspective. Decreasing the commute time can result in increased use of private vehicles, which leads to an escalated need for parking and road maintenance, as well as more air pollution in the city.

Carsharing in general means that many people share one or more cars, and several different business models for how this is solved exists (Cohen & Kietzmann 2014). A common for-profit model is a business-to-consumer model, where a company owns several vehicles and distributes them at appropriate parking spots around a city. Members often pay a monthly or yearly fee and can locate the most convenient car on their smartphone, book it and unlock it with their membership card. They drive it only for the time they need it, and usually pay a sum based on how long they use the car, the distance driven, or a combination of both. Some companies use a roundtrip model, where the car must be parked at the same spot it was picked up, while others employ a one-way model, where the car can be parked somewhere close to the user's destination. This carsharing model has two goals in general; the company wants to generate and maximize profits while also supporting sustainable mobility. The idle time of the vehicles is minimized, and the economic costs and benefits of the cars are distributed among all members, making it potentially cheaper to always have access to a car compared to owning a personal vehicle. Since the companies support more sustainable mobility, they often make deals with the city to get better parking spaces for a cheaper price, and other privileges such as reduced tolls. A non-profit version of the same model also exists, where the members all contribute resources to manage the carsharing without anyone expecting a financial gain. A third business model is a peer-to-peer carsharing scheme, where private individuals own cars they rent to other private individuals. Some intermediary using the web or mobile technology is used to connect owners with potential renters.

Ridesharing is not about renting or borrowing a car, but rather a seat in a car. Carpooling

is a popular activity, where vehicle owners allow people going to and from the same or similar locations at the same time to ride in the same vehicle (Cohen & Kietzmann 2014). The owners are in general not seeking profit here, but the passengers often pay a small sum to help the driver with expenses such as fuel and other costs associated with owning a car. Carpooling also contributes to reduced traffic congestion, as a standard car potentially can replace five cars if people are carpooling to and from work during rush hours. Historically, carpooling had to be arranged either personally with people the driver knew, or through designated meeting places, but with recent ICT developments, this process has become easier. New carpooling schemes use social networks and mobile geolocation to find both drivers and potential riders, and this can enable real-time co-riding with strangers, potentially making carpooling more attractive and thereby reducing traffic and pollution.

Sharing vehicles could potentially reduce one's carbon footprint compared to owning a personal vehicle. Owning a car which is always available might result in more driving than necessary, whereas participating in carsharing could mean that one only uses a car when it is needed and might use a bike or public transport for shorter trips when possible. However, a study has shown conflicting results on the effect of carsharing on greenhouse gas emissions (Martin & Shaheen 2011). Some of the households in the study had significant reductions in such emissions, but the majority participating in the carsharing scheme ended up with slightly increased emissions. The carsharing led to an expanded availability of cars for people who usually would not own a personal vehicle, and the possibility to get cheap rides were shifting people away from public transport. Even though the average greenhouse gas emission per household decreased, carsharing might work against one of its purposes.

2.3 Autonomous vehicles

Historically, vehicles have been entirely dependent on a human driver, only responding to the driver's commands and unable to make decisions (Gerla et al. 2014). The past years' developments in technology, communications and software have introduced Advanced Driver Assistance Systems (ADAS) and Automated Driving Systems (ADS), designed to help the driver with parts or all of the driving (Rödel et al. 2014, NHTSA 2018). Existing ADAS include helping tools such as automatic transmission, parking assistants, and navigation systems, which

already are well-used, while ADS can be autonomous vehicles (AV) capable of doing all the driving, but these are still being researched, developed and tested. The extent to which ADAS or ADS are used in vehicles differ, and the National Highway Traffic Safety Administration (NHTSA) in the US use six levels of automation when classifying vehicles, described in table 2.1.

Automation level	Who drives	Description
Level 0	Human driver	The human does all the driving.
Level 1	Human driver, assist from system	An ADAS can sometimes assist with either steering or braking/accelerating, but not simultaneously.
Level 2	Human driver, assist from system	An ADAS can control steering and braking/accelerating simultaneously under some circumstances. The driver pays full attention and performs the rest of the driving task.
Level 3	Human driver and system	An ADS can do all of the driving task under some circumstances. The driver must be ready to take control at any time when requested. In other circumstances, the driver performs the driving task.
Level 4	System and human driver	An ADS can do all the driving in certain circumstances. The driver need not pay attention in those circumstances.
Level 5	System	An ADS can do all the driving in all circumstances. The occupants are just passengers and need never be involved in driving.

Table 2.1: SAE's six levels of automation
Source: NHTSA (2018)

This thesis focuses on fully autonomous vehicles (level 5), which can disrupt the whole car industry, and potentially bring with it huge benefits within road safety, traffic and mobility, among others (Bertoncello & Wee 2015, Fagnant & Kockelman 2015, NHTSA 2018). Despite all these benefits for both individuals and society, there are also many challenges and obstacles to overcome before fully autonomous cars are available to the public.

Almost all traffic accidents today are caused or contributed to by human error or choice (Fagnant & Kockelman 2015, World Health Organization 2015, NHTSA 2018). Self-driving cars can mitigate the human factor; they can be programmed to follow all speed limits and traffic rules, they will not be affected by drinking or drugs, and they can react faster than humans can,

which could significantly decrease the number of people killed or injured in traffic (Fagnant & Kockelman 2015). The fatalities and injuries resulting from traffic accidents are very expensive for the society, both regarding loss of life and decreased quality of life, but also concerning lost workplace productivity, so reducing accidents will also incur an economic benefit (NHTSA 2018).

Roads containing AVs can also lead to better flow in traffic, and reduce congestion, fuel usage, and emissions (NHTSA 2018). By anticipating other vehicles' braking and acceleration, AVs can achieve smoother speed adjustments, and they can also decrease the gap between cars, causing a more efficient road and intersection usage (Fagnant & Kockelman 2015). However, for these benefits to be maximized, a high percentage of AVs in traffic is required, along with some vehicle-to-vehicle or vehicle-to-infrastructure communication. By reducing traffic congestion and optimizing driving patterns, commuting can take shorter time, and with fully automated vehicles, the time spent on the road can also be used more productively (Bertoncello & Wee 2015, NHTSA 2018).

The mobility in the society will potentially increase with fully autonomous vehicles, by providing everyone with the possibility to use a car (Fagnant & Kockelman 2015, NHTSA 2018). Elderly, people with disabilities or people too young to drive can use the cars the same way other people can, which can make independent living possible for more people, and also allow more people to work, improving their quality of life. Combined with car sharing and/or ride sharing, AVs as an on-demand service could also replace the need for a personal car for many households, and Fagnant & Kockelman (2018) found that with an optimized fleet size, each shared AV could replace ten privately owned cars in cities. The number of parking spaces can then be reduced, both because there are fewer cars, but also because each car will be used for a higher percentage of the day. Additionally, AVs will not need a parking space with room to open the doors, which also reduces the size of the parking lots (Bertoncello & Wee 2015).

Several challenges are related to the technology used. Fully automated vehicles rely on a variety of sensors, cameras, radars and more to observe the environment and identify all objects and their speed, which is very critical to do correctly and can also be very difficult (Gerla et al. 2014, NHTSA 2018). People around the car can be of all shapes and sizes, moving at different speeds, or sitting, standing or laying on the ground, while other objects can be difficult to separate from one another; a square-shaped object in the road could be a box made

of cardboard, or a block of concrete, which lead to very different consequences if crashed into (Fagnant & Kockelman 2015). In addition to this, the weather conditions around the car could lead to sensor noise, making it more difficult to see the surroundings and thus challenging to make the correct decisions in traffic. The technology used is also costly, making the cars not affordable for most people without a significant decrease in price.

Another problem people are worried about with autonomous vehicles is the security of the software and the privacy of the data (Gerla et al. 2014, NHTSA 2018). Malicious attacks must be prevented, as they could be fatal if human passengers have no chance to take control over the software in the vehicle. Attacks on an entire vehicle fleet or terrorist attacks could have severe consequences, but security breaches are more commonly used for espionage and not sabotage (Fagnant & Kockelman 2015, p. 177). With security measures, like those used in, e.g., air traffic control systems, the risk of attacks can be significantly reduced. Much data is gathered from the rides taken, and how this data is stored, shared and used is a big concern for many. Data about a person's routes, destinations and travel times can be used for tracking, surveillance or targeted advertising, but it could also be used to improve traffic and reduce prices (Fagnant & Kockelman 2015, p. 178).

For autonomous vehicles to be used, it requires people to trust the cars enough to accept handing all control over to the car, and for people to keep using the cars, it is essential that their attitude and experiences are positive. Rödel et al. (2014) studied people's general attitude and experiences with all levels of autonomous vehicles to find out what factors affected their attitudes and how the potential for autonomous cars was. They found that people's acceptance decreased as the autonomy level increased, but in general people were very positive and had good experiences with familiar cars. People who had experience with different ADAS also tended to have a higher acceptance for more autonomous functions. This could mean that the general acceptance for autonomous vehicles will increase as people get more familiar with vehicles that include higher levels of autonomy.

Not only does the public have to be positive, but governments must also change laws for autonomous cars to become publicly available and be a part of the traffic (Fagnant & Kockelman 2015). Insurance companies will have to change their policies based on who should be held liable if an accident occurs (Bertoncello & Wee 2015, Fagnant & Kockelman 2015). As humans no longer control the cars, they can probably not be held liable if an accident occurs, and it might

be the car manufacturers that need insurance in case of a technical error. The ethics behind the choices the car should make is also a challenge; the car might act differently if an animal jumps in front of the car than if it is a human, and it might not prioritize its passengers over people outside the car (Fagnant & Kockelman 2015). These algorithms are also something lawmakers must take into account when designing regulations.

Research is still needed on the subject, and although some manufacturers have come a long way and the possible benefits are huge, there are still many challenges to overcome and regulations to be changed before automated vehicles will be publicly available (NHTSA 2018).

2.4 Usability

Usability is the ”extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”, as defined by the International Organization for Standardization (2010), and is an important element in any application today. Poor usability, meaning that the application is challenging to use or that controls are difficult to master, for example, negatively affects the user's overall experience and interaction with the app, and can in the worst case lead to the user refraining from using the system.

The usability of a mobile application depends upon its user interface (UI), which is the part of the system the user can see and interact with. It is therefore essential to have a good UI design that increases the user's confidence to keep using the app. The effectiveness of a UI refers to how accurate and complete users can achieve specified goals, while the efficiency is about the resources needed to achieve those goals, and the satisfaction is decided by how comfortable and positive the users are towards using the product (International Organization for Standardization 2010). For the UI to achieve this, it should be organized logically and consistently, so that it does not require much of the user. Nielsen (1995) defined ten heuristics to design usable UIs, including recognition over recall (users should understand, not remember, how to use the system), minimalist design, error prevention and help to recognize, diagnose and recover from errors, and user control and freedom, among others.

2.4.1 System Usability Scale

Since usability depends upon the specified users and context of use, the testing of it is also dependent on this, so it is difficult to compare the usability of different systems (Brooke 1996). A huge variety of scales to measure usability exists, and one of the most used scales is the system usability scale (SUS). SUS is a standardized, simple scale aiming to give a global view of the usability of a system. It consists of 10 statements, where the participants, after testing the system, rate how much they agree to each, ranging from strongly disagree to strongly agree. A SUS score can then be calculated based on the responses, where all answers are given a score between 0 and 4, where 4 is the best. For the odd-numbered questions, strongly disagree is given a score of 0, while strongly agree is given a score of 4, and the even-numbered questions are scored oppositely: strongly agree is given a score of 0, while strongly disagree is given a score of 4. The score is summed up and multiplied by 2.5, to provide an overall score between 0 and 100. This score is not a percentage, in fact, 68 is considered an average SUS score. This means that a system scoring above 68 (on average across all test participants) have above average usability. The SUS score only implies how usable a system is, it is not diagnostic and does not mean where potential problems might be. It is, however, a reliable tool for measuring usability, and can be used to test a variety of systems, including both hardware and software.

2.5 Shared mobility and autonomous vehicles: Existing solutions

This section presents some of the many existing solutions for products and applications working on autonomous vehicles or offering shared mobility.

2.5.1 Waymo

Waymo is a company working with self-driving technology, initially started as a Google project in 2009, but became an independent company in 2016 (Waymo 2018*a*). The name originates from their goal, which is to develop a new way forward in mobility. They aim to improve mobility for people around the world with their fully autonomous vehicles, by making it easier

and safer for everyone to move around. Waymo's cars contain several sensors and computers to see the world around it, all 360 degrees (Waymo 2018*d*). The data from these are combined and used with artificial intelligence and machine learning, based on more than 5 million miles of test driving in different surroundings, to make predictions and tell the car how to navigate through traffic (Waymo 2018*e*). When making decisions, Waymo emphasizes that not only the safety of the passenger(s) is taken into account; the car is programmed to make sure that everyone in and around the car feels secure.

In addition to cars from big car manufacturers such as Lexus and Chrysler, Waymo has also designed their own vehicle, named Firefly, which is a complete self-driving vehicle containing custom sensors and computers, but no steering wheel or pedals, and can be seen in figure 2.5 (Waymo 2018*c*). Firefly completed the world's first fully autonomous ride on public roads in 2015, done in the busy streets of Austin, Texas. Waymo continues to work on cars driving in regular traffic, and has since 2017 had an early rider programme, where the public in and around Phoenix, Arizona can sign up to test the cars in their everyday life, driving on public roads without anyone in the driver seat, and they also plan to expand to other cities eventually (Waymo 2018*b*).

Figure 2.5: Waymo's FireFly
Source: Waymo

2.5.2 Uber

Uber is a ride service company aiming to make transportation easier, safer and more affordable, while also reducing traffic congestion in cities (Uber 2018*d*). Since its beginning in 2009, the company has grown worldwide and has 3 million drivers and 75 million riders spread in over 600 cities in 65 countries, resulting in 15 million trips each day. Uber itself does not own any vehicles or hire any drivers, but is a social networking platform in the form of a mobile app, used as an intermediary to connect drivers with potential passengers in real-time, based on their location (Cohen & Kietzmann 2014). This is a way for the private drivers to earn extra revenue using their private vehicle, where Uber takes up to 20% of the transaction.

The rider enters a destination in the app and chooses wanted ride option (Uber 2018*b*). The different options offered are in different price ranges, from economy class uberX for up to four people, to uberXL with affordable SUVs for up to 6 riders, to premium uberLUX with luxury cars and professional drivers. One has to book the entire car for all these options, no matter how many passengers are riding. A ridesharing option also exists, uberPOOL, which matches the passenger with other riders heading the same direction. Ridesharing saves costs as the price is shared, helps reduce traffic since fewer cars are needed, but might take longer time as one has to pick up and drop off other passengers on the way.

The rider is then matched with a driver and can see info such as the driver's picture and ratings, vehicle details and estimated price, which is set by Uber and not the driver, for the ride (Uber 2018*c*). The rider can then track the car's arrival on the map, and the app also handles payment, so no money is given directly to the driver. When the ride is finished, one can rate the driver, which helps make Uber safe, reliable and enjoyable for everyone. Initially, the matching between rider and driver was based on ridesharing, where the driver was going the same direction as the rider (Cohen & Kietzmann 2014), but now, most drivers operate more like taxi drivers, and some have quit their job to become full-time Uber drivers. Uber is therefore now more similar to a taxi service, but many would say that their product is better than a typical taxi company; one can book a ride with a few taps on a smartphone, the payment is more convenient, the price is often lower than for a taxi, and one rates the driver and ride afterwards, which ensures high standards (Christensen et al. 2015).

In addition to the services offered now, Uber is also working with self-driving technologies

to implement in their products in the future (Uber 2018*a*). They are planning to use this both for trucks to move goods safely and cost-effective, and in their cars, to transport people more efficiently and reduce traffic congestion, to make rides more affordable and accessible, as well as making traffic safer and reduce the number of accidents and lives lost in traffic.

2.5.3 BlaBlaCar

BlaBlaCar is the world's biggest long-distance carpooling platform and serves as an intermediary connecting drivers with passengers (BlaBlaCar 2018). The company does not own any vehicles, but is an online platform for web and mobile where drivers can post available seats in their private car, and passengers can look for available seats to their destination. BlaBlaCar suggests a fixed price per available seat, which takes distance, fuel, tolls and other expenses, such as insurance and maintenance, into account, but the driver can adjust this price within certain limits. This is done to ensure that the driver does not make a profit from the ride, since the general goal of carpooling is to split the costs between all passengers. Because of this, it is also not possible to rent out more than three seats per ride. All payment is made online, making it safer for both driver and passenger, as the passenger pays before the ride and the driver receives the payment after the ride. BlaBlaCar also receives a 12% commission for each transaction, but it is considered a non-profit company as drivers are not generating any revenue. Both the driver and the passenger have to use their real name in their profile, and they often also include a picture and short description of themselves and their car. In addition to this, they also develop a reputation based on previous ratings. After a ride has ended, the driver can rate the passenger(s), and opposite, making it a safer platform, as the user can read previous ratings before booking a ride.

In addition to the economic motivation, as the cost in general is cheaper than the corresponding train or bus ticket, and the ride also is cheaper for the driver compared to driving alone, users are motivated to choose carpooling from ecological and social reasons (Farajallah et al. 2016). Carpooling reduces road congestion and greenhouse gas emissions as it minimizes the amount of cars on the roads, but it is also a social venue to meet and get to know new people. The name BlaBlaCar actually comes from how talkative the driver wants the ride to be, ranging from ”bla” and not very talkative, via ”blabla” if they would like to talk a little, to ”blablabla” if

they are very talkative (BlaBlaCar 2018). As the driver and passenger(s) will share a small car for possibly several hours, the social aspect is an important factor when booking a ride.

2.5.4 ZipCar

ZipCar was one of the world's first successful carsharing companies, and is said to be the innovator of the concept of carsharing (Schor 2016). It started as an independent company who had cars placed around cities for members to use on an hourly basis, but has now become a subbrand of the car leasing company Avis. ZipCar has a for-profit business-to-consumer roundtrip business model (Cohen & Kietzmann 2014, p. 284), where people pay a monthly or yearly fee to be a member, and can reserve cars online, pick it up and rent it for a few hours or days, before they park in the its reserved spot, and pay online for the time they used the car (ZipCar 2018*a*). This business model provides a more affordable access to cars, since all maintenance, insurance, gas and parking prices are taken care of by ZipCar, and members only pay for using the car when they need it, not the whole year around (ZipCar 2018*b*). Several similar companies exist around the world, for example DriveNow, a German-started carsharing company owned by the BMW group in Europe (DriveNow 2018).

2.5.5 Bilkollektivet

Bilkollektivet is a Norwegian business-to-consumer carsharing company, owning vehicles in several of the big cities (Bilkollektivet n.d.). It works similarly to ZipCar, with a membership fee covering expenses such as insurance, maintenance, fuel, toll and parking, and other than that, members only pay for the time they use the car and the distance they drive. The big difference from ZipCar, however, is that Bilkollektivet is non-profit and owned by only the members, meaning that the price each member pays, goes directly to the car park and the service, no company is trying to earn a profit from its members.

2.5.6 Nabobil

Nabobil is a Norwegian peer-to-peer carsharing company, where people can rent out their private vehicle to people in need of a car for a short period (Nabobil n.d.). Its business model is

built upon the philosophy of the sharing economy of using resources more efficiently. People who own a car, but don't need it all the time, can rent out their car for some fee, which will then help them cover expenses such as insurance and maintenance. At the same time, people who do not own a car, but need one once in a while, can borrow their neighbor's car and pay a small sum, which will be much cheaper than owning a car themselves. The renter has to pay before he receives the car, and the owner receives the payment after the car is returned, making it safer for both parts. The renter also has to register a credit card in case extra expenses are incurred during the rental period.

2.6 Mobile app development: Platform and technologies

One of the goals of this project is to develop an app to handle car bookings, and this can be done in multiple ways with different technologies. This section therefore compares native apps for Android and iOS to web applications and presents some advantages and drawbacks of each. Some popular technologies for the development of these are also described. This is followed by an evaluation of these, resulting in a decision on what platform and technology to use for the development of the app.

2.6.1 Mobile application

Mobile applications are programs developed to be installed and used on a particular mobile device. Developing a mobile app have several advantages, including access to built-in hardware, such as camera and GPS, functionality and application programming interfaces (API) offered by the platforms, as well as third-party libraries (Budiu 2013). Since the app runs directly on the device, the performance of native apps is usually better than other alternatives, and they can often also be accessible when the device is offline. A drawback of native apps is the distribution and updating of mobile applications. They must be installed via application stores such as App Store and Google Play, and changes made to the app must be packaged in a new version which each user must update to. Several programming languages, frameworks and recommended Integrated Development Environments (IDEs) to develop mobile applications exists, mainly separated into native and cross-platform development.

Native development

One option is to develop native, platform-specific apps for the two most popular mobile platforms, iOS and Android (Gartner 2017), by using tools and technologies as recommended by the individual vendors.

Native iOS applications can be developed using the Xcode IDE with Swift (or Objective-C) as the programming language (Apple 2018*a*). Swift is a powerful programming language, aiming to make it easy to write and maintain correct programs, and to be the best language for several different use cases (Apple 2018*b*). It is said to be easy to learn, and is developed to be safe, fast and expressive. A disadvantage of developing native iOS applications is that Xcode is only available for Mac, making it impossible to do with e.g., a Windows computer.

Android Studio is the official IDE for Android development, using Kotlin as its official programming language, but it is also possible to use Java, or a combination of both (Android 2018*a*,*b*). Kotlin is, similarly to Swift, developed to be a modern, expressive and safe programming language to improve performance and make faster apps. Contrary to iOS development, there is no restriction on the tools or machine, one does not have to use Android Studio but can use the IDE the developer prefers, where popular choices include IntelliJ or Eclipse.

Both native apps for iOS and Android have the advantage that functionality and APIs for the platform, as well as third-party libraries, can be used directly in the code, they have the highest performance as they do not require any abstraction level, and also the smallest footprint since the app package only contains the code and some configuration (Friberg 2017, Apple 2018*a*, Android 2018*b*). However, developing for both iOS and Android requires two separate projects with no code reuse for the two platforms. It also requires knowledge of different programming languages, frameworks, and classes and components that can be used.

Cross-platform development

Another option is to use a cross-platform framework, enabling the sharing of code between the apps. Two promising frameworks are React Native by Facebook and Xamarin by Microsoft.

React Native uses only JavaScript to develop mobile apps for Android and iOS, and one can use any IDE of choice (Facebook 2018*c*). Small, reusable and declarative components are written in JavaScript and rendered to higher-level, platform-specific components on each

platform (Occhino 2015). This results in an app looking and behaving like any native app, because the same native components are used, only wrapped with JavaScript. One can also write modules in native code when needed to optimize the app, and since React Native is open source, the whole community can contribute to new modules, making it a richer framework (Witte & von Weitershausen 2015).

Xamarin uses .NET and C# to develop apps for iOS, Android and Windows, and one has to use Visual Studio as the IDE (Xamarin 2018). One can choose to develop the UI native for each platform, or use Xamarin.Forms to share as much code as possible for the UI as well (Microsoft 2018). This can be done with XAML (eXtensible Application Markup Language) to define the UI, binded to a C# code-behind-file, and renders to native UI components. It is also possible to define custom renderers for Android and iOS where needed, also done in C#, which means that the developers do not have to learn several technologies and programming languages.

Both frameworks use a shared codebase for both platforms, and they also offer the opportunity to build some of the app native when needed, to optimize the application. This means that one could need to write parts of the code in the native language to access all functionality or to follow design guidelines for the OS (Friberg 2017). Thus, the amount of code sharing might not be as helpful as anticipated before the project starts, but compared to having two separate projects, it is considered a huge advantage of choosing cross-platform frameworks to develop mobile applications that feel and look like native applications. A disadvantage for the cross-platform apps is that even though there is no need for a Mac to write code, one still needs a Mac to build the iOS app.

2.6.2 Web application

Web applications are websites that can look like native apps, but are accessible through a web browser when the device is connected to the Internet (Budiu 2013). HTML (HyperText Markup Language) is the standard language for the development of web pages, often used together with JavaScript for the logic and CSS (Cascading Style Sheets) for styling, but multiple new technologies and frameworks exist, enabling a faster development of more complex web apps. Since they are accessible through browsers on all devices, web apps only require one code base, and they are also easily distributed and updated as there is no need to install the apps on

the devices. However, they lack the use of platform-specific UI elements, and the native mobile gesture recognizers are challenging to achieve (Occhino 2015). The performance is also usually poorer on web apps than native apps, as it is difficult to parallelize work on different threads.

Angular is an open-source JavaScript framework, developed by Google and designed for the development of apps for both web, mobile and desktop (Google 2017). It extends the static data of HTML, which allows creating dynamic and interactive applications by using components that describe what should be displayed where, and how, in the app (Sidorenko 2017). It is considered easy to learn, but also powerful enough to use when developing complex single-page web apps. Since Angular is just a frontend framework, other frameworks and technologies are needed to build a full web app, and although the components allow for code of high quality, much code is needed for a single component (*The Good and the Bad of Angular Development* 2017).

React is a JavaScript library, developed by Facebook to build user interfaces for web applications (Facebook 2017), and is the predecessor of React Native. React uses reusable components, which are JavaScript functions that take some input and return a React element to appear on the screen (*Components and Props* n.d.). The components manage their own state and express what the application should look like, rather than what Document Object Model (DOM) operations are needed. A virtual DOM compares the previous state with the desired state and updates the DOM in the most efficient way, which means that React only updates the parts where changes are made, making the app very fast. Since React is a small solution to build the UI, additional frameworks, libraries and tools are needed to build complete web applications.

Elm is a functional programming language that compiles to JavaScript (Czaplicki 2017*a*). It has the same conceptual model as React and also uses a virtual DOM, but requires less setup and does not depend on JavaScript. Elm focuses on performance, robustness and usability for the developer, and aims to simplify the creation of responsive graphical user interfaces (GUI) and improve maintainability for web applications (Czaplicki & Chong 2013). HTML is described in a declarative way, making the creation and combination of text, images and videos to a multimedia display easier (Czaplicki 2017*b*). Since Elm is a relatively new programming language, there is a chance of incompatible changes requiring rewriting of code, and although Elm offers high-quality libraries and the amount of third-party libraries is growing, the Elm community is still small.

2.6.3 Choosing the best tools

Several good technology alternatives to develop an app for booking autonomous vehicles exists, and both mobile and web applications have good potential to fulfill the requirements of the app. However, since the geolocation of the device is essential for the app, and good performance also is very important, a mobile application would suit better than a web application. The app should be available for at least the two dominating mobile operating systems, iOS and Android, and because of limited time and resources, it would therefore be beneficial to develop a cross-platform app where the two apps could share all code logic. Looking at React Native and Xamarin, both technologies can accomplish this, and React Native was chosen due to its component-based architecture, which is well suited for the car booking app's requirements. React Native is further described in section 3.2.1.

Chapter 3

Methodology and implementation

This chapter presents the methodology used for the development of the mobile application. This includes a presentation of the intended product, a description of the chosen technologies, the requirements for the application, the architecture and implementation details, as well as a description of how the user tests were executed.

3.1 Overview

Combining the sharing economy and fully autonomous vehicles as a product for the public has enormous potential, and can be facilitated by a mobile application. The idea is that people become members and share a pool of self-driving cars by paying a yearly fee, and they can use an app to book cars whenever needed. Members download the app and create user profiles containing information, such as payment details and pre-defined addresses like their home and work addresses. Once signed in, users can see a map containing their location as well as available cars nearby, with the opportunity to book a ride.

This project is a part of the NTNU Autonomous Perception (NAP) project and focuses on developing the car booking application for this project, called NAP App. The application is the only part of the system the users communicate with and takes care of all communication with the database and cars. To book a car, the user enters the address of the wanted destination or place a pin on the map, and the system will assign the best available car and calculate the route, both based upon the user's current location. It is also possible to change the pickup location and

time, and this will result in a re-calculation of the route and car distribution. Cars are distributed among users in the most optimal way for both traffic, environment and the user itself. It will also be possible to share a car if several users are traveling the same direction at the same time.

In addition to the annual membership fee, users pay a fee for each ride, depending on the route traveled and whether or not the car was shared for parts of the journey. This happens when the ride ends as the user exits the car, and all payment is also taken care of by the application. The ride booking is the most critical use case for the mobile application, and a use case diagram for how a single user books a ride can be seen in figure 3.1.

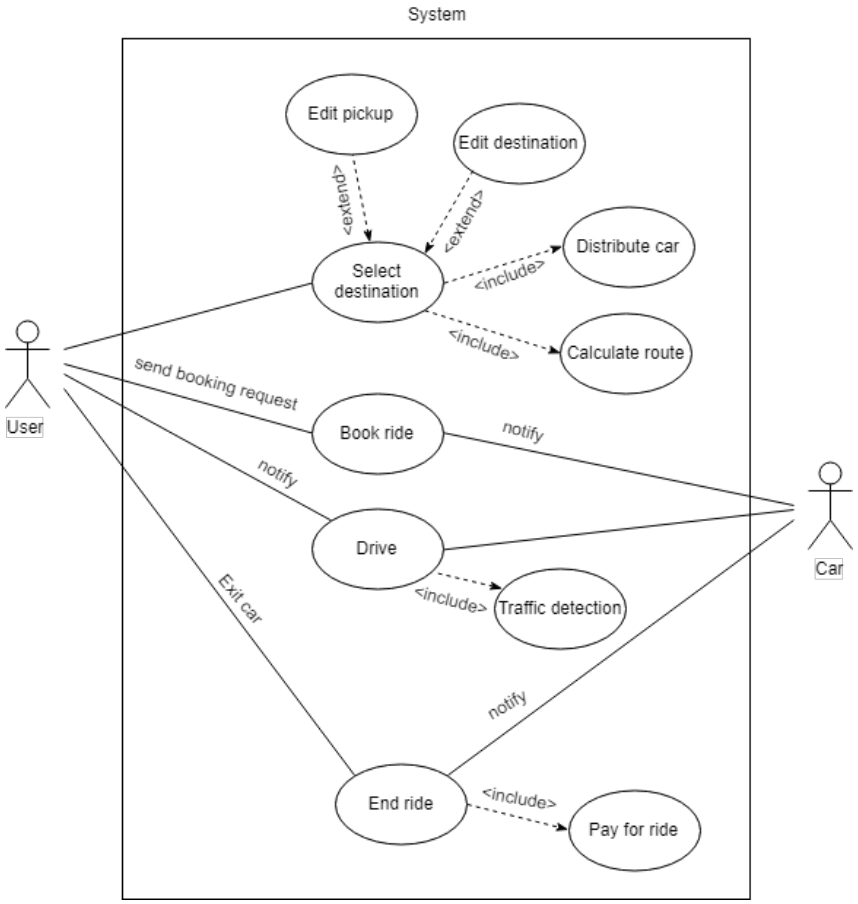


Figure 3.1: Car booking use case

3.2 Technology

This section presents the technologies used to develop the mobile app.

3.2.1 React Native

It was chosen to develop mobile applications for Android and iOS with React Native, to have one main code base for both apps and still achieve high performance. React Native is a cross-platform framework based on React for the web, where one can develop mobile applications identical to native developed apps, but using JavaScript (Facebook 2018*c*). One uses similar declarative components representing single views as with React, but instead of rendering to a virtual DOM, the React Native ”bridge” invokes the Objective-C APIs for iOS and Java APIs for Android, resulting in components rendered as the real, native UI components for each platform (Eisenman 2015). React Native also supports all native APIs, which means that the application can access features like the geolocation of the device or the camera.

Using the host platform API's makes React Native able to achieve 60 frames per second and therefore a significantly better performance than frameworks using web views (Facebook 2018*b*). The logic runs on the JavaScript thread, separate from the main UI thread, so the application can achieve both high performance and capability without affecting the user experience. This is an advantage for this project, as lots of re-rendering is necessary, at the same time as API calls and calculations must be done. However, complex calculations and re-rendering of expensive component subtrees can lead to several frames being dropped, resulting in the app appearing to freeze, and this should be avoided as much as possible to ensure a good user experience.

React Native functions like an abstraction layer between the code and the rendering, with the bridge providing an interface to the native UI elements (Eisenman 2015). A React component returns some markup describing what it should look like, which is translated to fit its host platform. For example, a <View> in a React Native component will be translated to an UIView on iOS and an android.view on Android. Some components can, however, be specified for each platform if React Native does not offer the interface to translate it, or some parts of the app need to be optimized (Facebook 2018*c*). By being forced to break the application into discrete components, it becomes easier to maintain and iterate on the product, and the applications are also easier to scale (Occhino 2015).

An example of the code for a simple React Native component can be seen in listing 1. The constructor sets the initial state for the component, in this case, ”greeting” is set to an empty

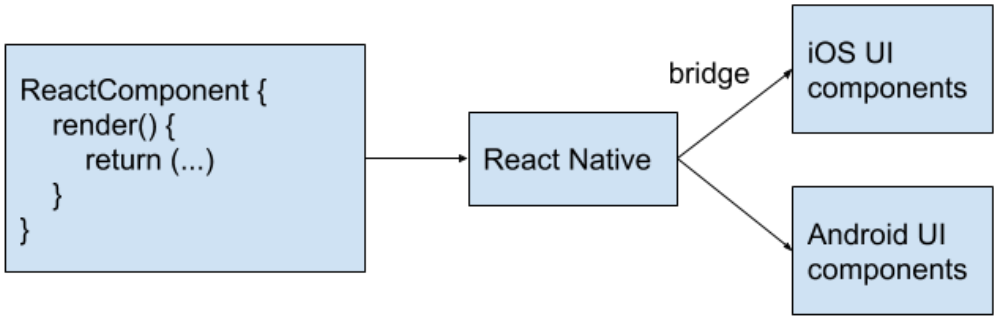


Figure 3.2: React Native bridge

string in line 7. The render function in line 14 to 24 is what returns the markup being translated into native UI components. The code for this is not plain JavaScript, but JSX (JavaScript XML), an extension for JavaScript which embeds XML (eXtensible Markup Language) (Facebook 2018*a*). This is due to React's focus on separation of concerns instead of separation of technologies, so the markup, style, and behavior of each component are in the same file (Eisenman 2015). This component contains a view with a text input field and a text field for displaying a greeting to the user. The initial output for this on iOS is shown in figure 3.3a. The function onTextChange() in line 10-12 is called when the user has entered some text in the text input field and submitted it, and updates state. When the state changes, the component is re-rendered, shown in figure 3.3b after the user has submitted "World".

The styling of components can also be seen in this example, for the <View> component, and is also done with JavaScript. The names and values for different style attributes typically match styling done with CSS (Facebook 2018*d*). It does not contain all CSS rules, as React Native instead focuses on simplicity and consistent support (Eisenman 2015).

3.2.2 Redux

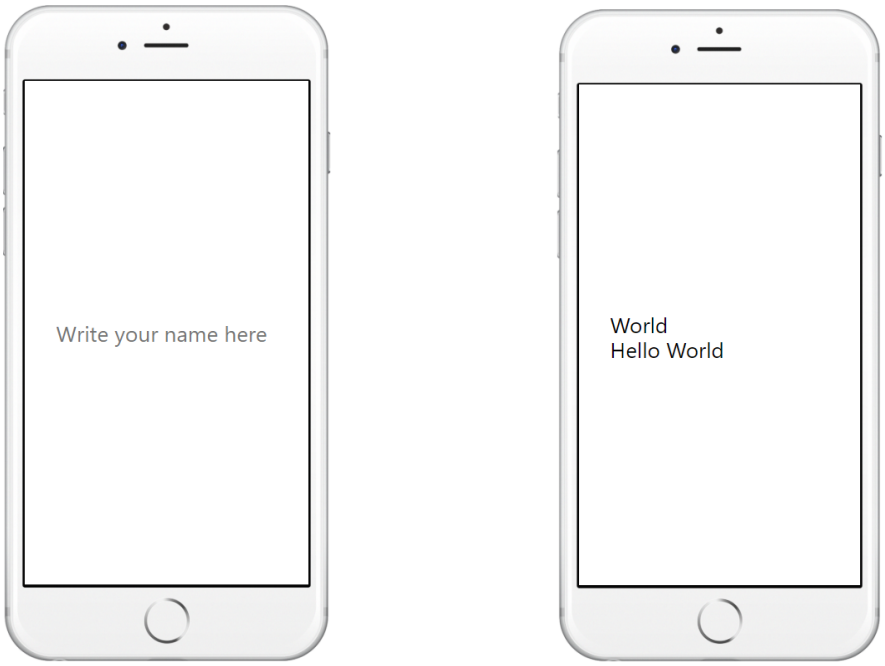
Redux is a state container for JavaScript apps, designed to assist with development of applications that should behave consistently, run across varying environments and be easy to test (Redux 2018*a*). React already contains some state management, but Redux also introduces predictable data flow and a more consistent architecture, simplifying the development. Additionally, Redux comes with functions to make the developer experience better, such as time travel debugging and logging of all updates, live code editing, and much more.

```
import React, { Component } from 'react';
import { View, TextInput, Text, StyleSheet } from 'react-native';

export default class Hello extends Component {
  constructor() {
    super();
    this.state = { greeting: '' };
  }

  onTextChange(name) {
    this.setState({ greeting: 'Hello ' + name })
  }

  render() {
    return (
      <View style={styles.container}>
        <TextInput
          placeholder="Write your name here"
          onSubmitEditing={event => this.onTextChange(event.nativeEvent.text)}
        />
        <Text>{this.state.greeting}</Text>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'flex-start',
    padding: 20,
  }
})
```

Listing 1: React Native component

A store is introduced to the application, where the entire state is stored as an object tree (Redux 2018*e*). This state can only be changed by actions being dispatched, which is then processed by a reducer function handling how every possible action will alter the application state. The rest of the application can only read the state, not modify it. Since the car booking application will contain a significant amount of data changing rapidly, and it will be inefficient to store the entire state in a top-level component, Redux is a good choice to make the data flow simpler and thereby improve performance. Several components must use the same data in different ways, and by allowing them to subscribe to the Redux store, all components can use precisely the parts of the state it needs, and ignore the rest.

A code snippet of a simplified Redux application for a Todo-list (where the only possible



(a) View before text submission **(b)** View after text submission

Figure 3.3: React Native component on iOS

interaction is to add a new todo) can be seen in listing 2. This application is so simple that Redux would not be necessary, but it illustrates the concept well. The state of the application is stored in objects, combined in a tree, but in this case, the state only consists of a list with todos (line 2). Since the user only can add new todos, this is the only action necessary for the application. Actions are plain JavaScript objects with a type field and a payload, as in line 5-10, and this is what the reducer looks at to update the state (Redux 2018*b*). For more complex cases, it is common to use action creators. One can also include some middleware, which makes it possible to run code after dispatching an action, but before it reaches the reducer.

Actions are dispatched in components, through their container. The component itself is not included in this example as it will be similar to the component in listing 1. The container can be seen in line 13-23 and uses the React-Redux connect function to wrap the AddTodoComponent and connect it to the store. The function mapStateToProps is called every time the state changes and returns an object with props for the component, in this case, the list of todos in the state. To be able to dispatch the addTodo-action, mapDispatchToProps is used and is only called when the component is created. This function returns an object with functions that use dispatch, here onSubmit(task), so when this function is called in the component, the addTodo-action will be dispatched.

Since there only is one component and one action, only one reducer is needed, but for bigger apps, several reducers would be necessary, usually combined to a rootReducer. A reducer is a pure function only relying on the input it receives and contains all state logic for the app (Redux 2018*d*). The reducer in this example, in line 26-39, looks at the type of action, and if it is of type "ADD_TODO", it makes a copy of the current state, adds the action payload to the list of todos, and this new state is then returned. The current state is stored in the Redux store, which is created with the createStore-function, as in line 45, taking the rootReducer as an argument. The store then notifies subscribers, which is the components wrapped with connect(), and they can then retrieve the new state and re-render.

3.2.3 MySQL

A MySQL database was chosen to store data about the cars, users, and rides booked by users. MySQL is an open source SQL (Structured Query Language) database management system (DBMS) by Oracle Corporation (Oracle 2018*b*). The relational MySQL databases stores the data in different tables instead of in one place and this model allows for a flexible programming environment. A DBMS such as MySQL is needed to add, access and process data stored in a computer database.

There are several ways to access the data stored in a MySQL server. Depending on the need, one option is to use SQL and either enter the statements directly or embed them into code in another language (Oracle 2018*b*). Another option is to use language-specific APIs that hide the SQL syntax, with language options including C++, Java, PHP, Python, Ruby and others (Oracle 2018*a*). Since the MySQL database software supports multiple back ends, many client programs and administrative tools, as well as a variety of APIs, it is a highly usable solution for database handling and works in both client-server and embedded systems.

MySQL servers can run in several different environments, such as a web server, or a local desktop or laptop computer (Oracle 2018*b*). It is designed to be able to handle huge databases faster than other solutions and has become the most popular SQL DBMS, used with success in big and demanding environments. It is fast, reliable, scalable and easy to use, and this along with its connectivity and security makes it a popular and well-suited alternative for accessing databases on the Internet, and also a good fit for NAP App.

```
// State:
{ todos: ["Write thesis", "Buy groceries"] };

// Actions:
export const addTodo = (task) => (
  {
    type: ADD_TODO,
    payload: { task },
  }
);

// Container:
import { connect } from 'react-redux';
import { addTodo } from './actions';
import AddTodoComponent from './components';

const mapStateToProps = (state) => ({ todos = state.todos });

const mapDispatchToProps = (dispatch) => ({
    onSubmit: task => dispatch(addTodo(task))
});

export default connect(mapStateToProps, mapDispatchToProps)(AddTodoComponent);

// Reducer:
export const reducer = (state, action) => {
    switch (action.type) {
        case ADD_TODO:
            return {
                ...state,
                todos: [
                    ...state.todos,
                    action.payload.task,
                ]
            };
        default:
            return state;
    }
};

// Store:
import { createStore } from 'redux';
import reducer from './reducers';

const Store = createStore(reducer);
```

Listing 2: React-Redux application
Source: Redux (2018*a*)

3.2.4 PHP

To add, retrieve and modify data stored in the MySQL server, it was chosen to develop a REST API (Representational State Transfer Application Programming Interface) for the back-

end. REST is an architectural description containing some constraints and guidelines for the web, and a web API conforming to this architectural style is a REST API (Masse 2011). APIs are used by client programs to communicate with web services, so the API facilitates this information exchange by exposing a set of data and functionality from the web service, and is in that way an interface to a database for other programs to use. The client sends a request, identifying which resource it wants with a Uniform Resource Identifier (URI), and what operation it wants to perform with an HTTP (HyperText Transfer Protocol) verb such as GET or POST, and the REST API performs the operation on the resource and sends a response to the client.

This can be done using multiple technologies and programming languages, one of which is PHP, an open source general-purpose scripting language suited for web development (Achour et al. 2018). It is commonly used for server-side scripting, meaning that all code is executed on the server, which generates HTML, JSON (JavaScript Object Notation) or some other file and sends it to the client. The client will therefore not see the underlying code for the file generated, making PHP a suited alternative for writing REST APIs. The REST APIs are hosted on a web server, accessible using HTTP requests.

3.3 Requirements

This section presents the functional and non-functional requirements of the application, as defined for this project. A functional requirement is any required or desired feature, and non-functional requirements are quality goals of the application. Each requirement includes a priority level, where a high priority is used for features considered a necessity, and a low priority implies only a desired, but not critical feature. The functional requirements (FR) are presented in table 3.1, and the non-functional requirements (NFR) are displayed in table 3.2.

3.4 Architecture

This section describes the architectures of the application.

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 specified location on the map	High
FR5	The application should locate the nearest car to the user	High
FR6	The application should calculate 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 change the pickup location after booking	Low
FR11	The user should be able to change the drop-off location after booking	Low
FR12	The user should be able to decide the pickup time	Medium
FR13	The user should be able to track the booked car	High
FR14	The application should manage multiple bookings simultaneously	Medium
FR15	The application should distribute cars in a traffic and environmentally optimal way	Medium

Table 3.1: Functional 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 devices	High
NFR3	The application should have a simple and consistent design	High
NFR4	The application should support several users simultaneously	High
NFR5	The application should display the booked car's location in real time	High
NFR6	The application should display all cars' locations in real time	Medium

Table 3.2: Non-functional requirements

3.4.1 React-Redux

Since the app uses Redux for state management and data flow, it is written in a React-Redux architecture pattern, as shown in figure 3.4. This pattern describes the data flow in the application and contains a store, containers with associated components, actions, and reducers (Redux 2018*c*). The store contains the current state of the application at all times, which is passed to all React components through their containers. User interaction with components can through the container dispatch actions or action creators, which can handle API calls and dispatch new actions based on their response. Reducers compare the current state with the type of the dispatched action to update the state in the store.

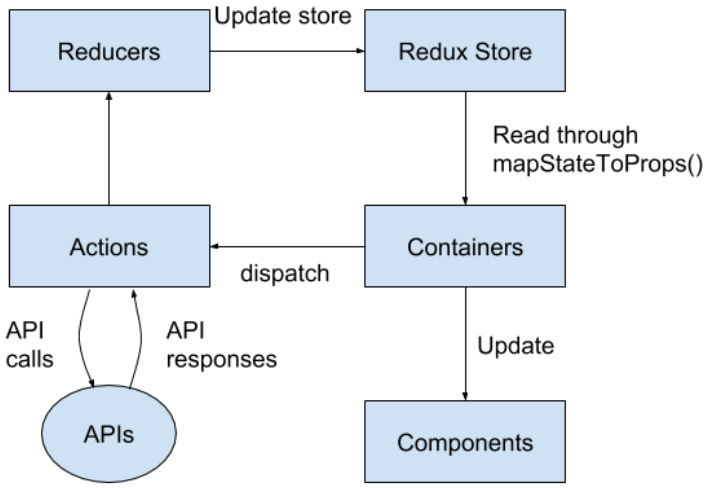


Figure 3.4: React-Redux architecture

3.4.2 Client-Server

The software for the application is structured in a three-tier client-server architecture, illustrated in figure 3.5. Three-tier architectures enable separating the functionality of the application into three layers (IBM 2018). They often consist of a client layer, an application logic layer, and a data layer. The data layer stores data for the application, typically in some database. The application logic layer can access the database based on requests from the clients and is where most of the logic and processing happens. Finally, the clients are responsible for presentation and interaction with the user, and allows the user to interact with the application logic layer, but not directly with the data layer. Using three tiers with an intermediary between clients and

database can reduce the load on databases and improve performance.

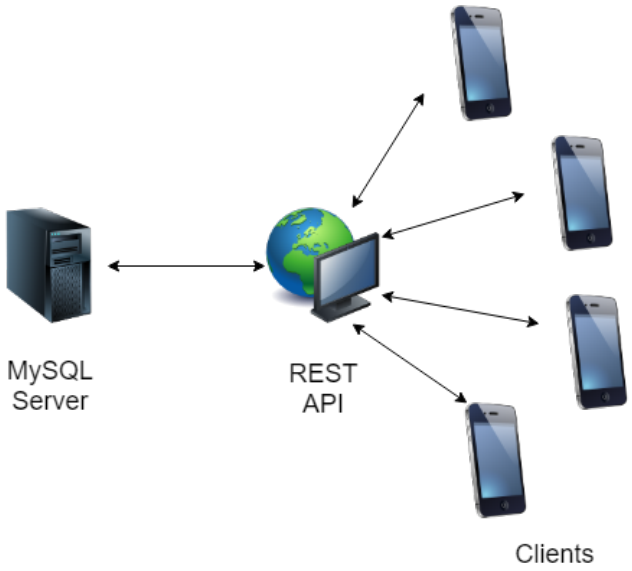


Figure 3.5: Three-tier client-server architecture

The database layer in this application consists of the MySQL server and contains all data about cars, users, and rides. The REST API, written in PHP, is part of the application logic and provides the data from the database to the client mobile applications. The rest of the application logic, like the calculation of rides, is implemented in the front-end application and is written in JavaScript. The clients, in this case, are the mobile apps, developed using React Native and JavaScript, and is only responsible for rendering the data received from the logic layer to the user, as well as handling all user interaction. Based on these user interactions, a client sends a request to the REST API to access or modify some data. The database waits for requests in forms of queries from the REST API and responds accordingly, and the REST API further handles the response before a new response is sent to the client. After receiving the response, the client updates the presentation to the user by re-rendering the data.

3.5 Implementation

This section describes the implementation details for the application. The implementation was done in two iterations, where the first one focused on the car booking functionality, and the second one focused more on the database and connecting the app to the REST API, as well as correcting bugs and usability problems from iteration one. The first part of this section will

describe the implementation of the database, while the next presents the REST API and how it is used to access data from the database. Finally, the third part discusses the implementation details of the mobile application.

3.5.1 Database

The first table needed in the database was a table to store information about the cars. Since not much information about each car was needed in the first stage of the development, the car entities stored in the database only contains a few attributes. The most important ones are its ID, coordinates of its current position and a boolean value storing whether the car is booked or available. The registration number of the car is also added, and although this is a unique attribute, a separate ID-field was used as the primary key. The database also contains some physical information about the car, such as the car brand and type, color and registration year. More attributes can be added to expand the functionality of the application.

For several users to be able to book cars simultaneously, there was also a need to create a user profile, and therefore a user table was added to the database. Similarly to the cars, the users only contain a few attributes in the database. The email is used as the username and has to be unique for every user. The database also stores a password for each user, encrypted with MySQL's password-function [1]. In addition to this, each user also has the opportunity to store their home address, to enable them to tell the application to go home instead of having to type their address manually every time. This could be expanded so that users could have several pre-defined addresses to book rides to and from.

It was also decided to store all booked and completed rides in the database. Each ride is booked by a user and driven by a car, so the table includes foreign keys to the other two tables. Both the user and car entities have unique attributes such as the registration number and email address, but they use separate ID-values, as this leads to faster joins, can be more efficient and is not subject to change. If for example, a user changes his or her email address, this would need to be updated in both the user table and in every ride the user has booked, but a separate ID-field will never need to change. In addition to the primary key and foreign keys, the table contains information about the start, pickup and end coordinates, as well as the start, pickup and end

[1] https://dev.mysql.com/doc/refman/5.6/en/password-hashing.html

times for the ride. The final entity-relationship diagram for the database can be seen in figure 3.6. As the name implies, this diagram also says something about the relationship between the entities. In this case, every ride is booked by exactly one user and driven by exactly one car, whereas any user can book none or many rides, and every car can drive none or many rides.

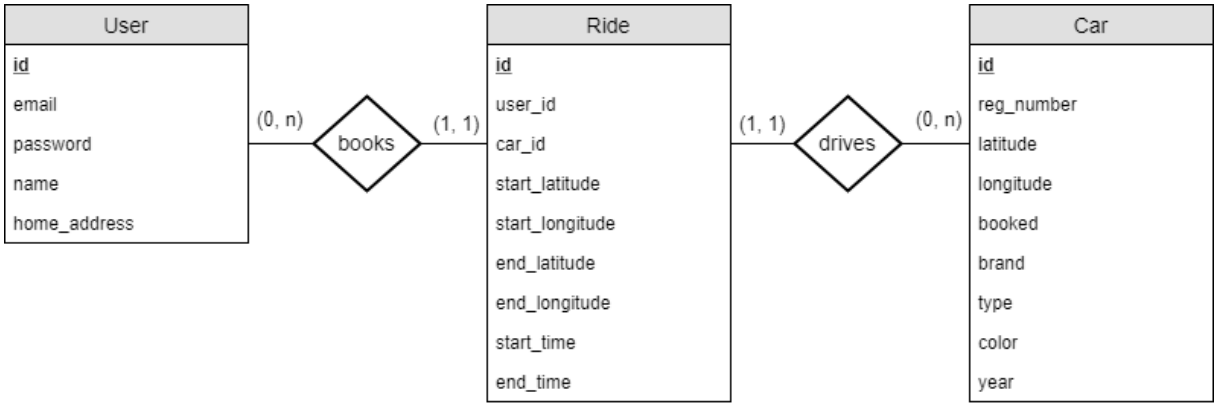


Figure 3.6: Entity-relationship diagram

3.5.2 REST API

The REST API implemented in PHP contains functionality to retrieve, add, and modify all objects in the database. During the development and testing of the application, it was hosted on an NTNU web server[2], available for students to host simple web applications. An API for the cars was first implemented. The most important part for the first functionalities implemented in the application was to retrieve information about all or one specific available car(s), as well as update a single car's booked-status and position. The API also includes functionality to add and delete cars, as this may be needed in the future. The code for the entire REST API is included in the GitHub repository linked to in appendix A.1.

When users and rides were added to the database, APIs to retrieve, add, update and delete these were also implemented. The user API is very similar to the car API, as the only functions needed is to add and delete a user, modify some attributes of an existing user, or retrieve data about one specific or all user(s). The API for a ride also contain these functions, but in addition, it contains functionality to retrieve all rides within a specific time period, for a specific user or by a specific car, or a combination of these. The code to retrieve rides can be seen in appendix B, where the query includes a join on all three tables, so the API returns data about the user

[2]http://folk.ntnu.no/

booking the ride and the car driving it, in addition to data about the ride itself. The JSON response for a single ride is shown in listing 3.

```
[
    {
        ride_id: "24",
        car_id: "1",
        reg_number: "AB12345",
        user_id: "1",
        name: "Test Testesen",
        start_latitude: "63.435230",
        start_longitude: "10.412560",
        start_time: "2018-06-07 10:55:52",
        via_latitude: "63.430550",
        via_longitude: "10.419922",
        via_time: "2018-06-07 10:59:57",
        end_latitude: "63.430134",
        end_longitude: "10.393510",
        end_time: "2018-06-07 11:08:38"
    }
]
```

Listing 3: JSON response for rides API

3.5.3 NAP App

The first step of the development process focused on implementing the core of the ride-booking, which is the most critical functionality needed for the app. This included adding a user at one location, a car at another location and enabling the user to book the car to drive to a third location. This was later expanded to include several cars, where the closest car to the user is chosen. This section describes the implementation of the essential components needed for this functionality. Because a Mac is needed to run the iOS-app, the focus was on the Android app, but the app should also work in iOS with some minor adjustments to the code. The code for the entire application can be seen in the GitHub repository linked to in appendix A.2.

Map

A module called React Native Maps[3] was used for the map in the application. This is an additional package that needs to be installed to React Native and renders to Google Maps on Android and Apple Maps on iOS. Using this makes the app more familiar for the users, as

[3]https://github.com/react-community/react-native-maps

it is the native map of the device they are using, and the module supports all known gestures to zoom and move around the map. The coordinates for the region displayed on the screen can be set in code, so the initial region of the map is set to the user's position. Markers can easily be placed on specified coordinates on the map. The UI components for the route and all cars are implemented as separate components, but they are subcomponents of the map and therefore included under the MapView. An excerpt of code for the map component is included in appendix C.

Destination and pickup locations

The functionality for setting wanted destination and pickup locations are implemented in two ways. If the user wants to type the address of the location, this is solved with text input components, but it is also possible to choose the location by placing a marker on the map. This is implemented by placing a marker in the middle of the screen, and the user has to zoom and/or move the map around until the marker is placed over the wanted location. This was chosen to enable the user to be more precise, as simply pressing the map can result in the map registering the wrong address. By default, the pickup location is set to the user's current location and is displayed with a blue dot, implemented as a marker with an image. When editing the pickup location, the blue dot is therefore moved to the new pickup location. The destination location is displayed on the map with a default red marker. Both these markers are also subcomponents of the map component, but the address input field and button to choose the location on the map is implemented as a separate component.

Directions

To get directions for the rides, Google's Directions API [4] was used, where one can search for directions between two end destinations with several via-locations. The locations can be submitted as coordinates, string addresses or places IDs. Google's Geocoding API[5] was also used to translate between coordinates and textual addresses for a more accurate and informative experience for the user. These APIs use HTTP requests and returns a JSON response containing information about distance, time, waypoints, HTML instructions for each step of each leg on the

[4]https://developers.google.com/maps/documentation/directions/intro
[5]https://developers.google.com/maps/documentation/geocoding/intro

route and a polyline for the route, among others. The HTTP request is sent every time the user submits a new end or pickup location. The polyline returned for each booking request from the user is encoded to latitude, longitude-pairs in JavaScript by using the additional package Mapbox Polyline[6]. The coordinates are then displayed using the built-in Polyline from React Native Maps, and the directions component with this polyline is included in the map component. The function to retrieve directions is included in listing 4.

```
export function fetchDirections(startCoordinates, endCoordinates, address) {
  return function (dispatch) {
    dispatch(fetchDirectionsRequest());
    const start = `${startCoordinates.latitude},${startCoordinates.longitude}`;
    const end = `${endCoordinates.latitude},${endCoordinates.longitude}`;
    return fetch(`https://maps.googleapis.com/maps/api/directions/
                  json?origin=${start}&destination=${end}&key=${API_KEY}`)
    .then(
       response => response.json(),
    )
    .then((myJson) => {
       const dir = getPoints(myJson.routes[0]);
       const duration = myJson.routes[0].legs[0].duration.value;
       dispatch(fetchDirectionsSuccess(endCoordinates, address, dir, duration));
    });
  };
}
```

Listing 4: JavaScript code to retrieve directions

Cars

The cars are vital elements in the application, but as fully autonomous vehicles licensed to drive on the road not yet are available, the cars in the application have to be simulated. As described earlier, the cars used in the application are fake cars stored in the database, accessible through the REST API. When the application is loaded, the cars are retrieved from the database and mapped to car components through the car list component, which is added as a sub-component of the map.

The distribution of the cars is essential functionality for the application. For this first version of the application, this is solved by finding the car with the shortest route to the pickup location, in terms of duration. To improve performance, this is only calculated the first time the user requests a ride, and every time the user updates the pickup position, or when the location of a

[6]https://github.com/mapbox/polyline

car has changed. The code for how this is solved can be seen in listing 5. This function returns both the car object, the direction and the duration, and this is later stored in state, so when the user requests a ride, only the directions from the pickup location to the destination has to be calculated.

```
export async function fetchBestCar(available, pickup) {
  let duration = Infinity;
  let bestCar = null;
  let directions = [];
  const destination = `${pickup.latitude},${pickup.longitude}`;
  await Promise.all(available.map(async (car) => {
    const start = `${car.coordinate.latitude},${car.coordinate.longitude}`;
    const resp = await fetch(`https://maps.googleapis.com/maps/api/directions/
                json?origin=${start}&destination=${destination}&key=${API_KEY}`);
    const respJson = await resp.json();
    const thisDuration = respJson.routes[0].legs[0].duration.value;
    if (thisDuration < duration) {
      duration = thisDuration;
      bestCar = car;
      directions = getPoints(respJson.routes[0]);
    }
  }));
  return [bestCar, directions, duration];
}
```

Listing 5: JavaScript code to assign car

Rides

The ride is another significant part of the application, as the primary goal is to book and complete rides. The ride is not stored as a separate component in the application, but all information about it is stored in the state, mostly under directions and car. The ride starts when the user presses the ”Book ride”-button, after requesting a ride from a pickup to a drop-off location. The best car is then reserved and simulates to drive to the pickup location, following the specified directions shown in the direction component. The car continues to drive from the pickup to the destination after the user has pressed the ”Continue ride”-button.

In the final app, the car driving should be based upon the actual movement of the car, but since this is not possible yet, the car is coded to move one step of the route at a time. The code for how this simulation is implemented is included in listing 6. The function receives a list containing all coordinates of the directions for the leg it is simulating, where the first coordinate is extracted for the car to move to, and the function is called recursively with the rest of the list

after a timeout. This is done for the user to see that the car is moving. The timeout is set to only 200 ms for the testing, so the user does not have to wait for the entire time the ride actually would have taken.

```
1  export function driveCar(directions, car, i, type, places) {
2    return (dispatch: Function) => {
3      if (directions.length === 0) {
4        if (type === 'Destination') {
5          dispatch(addRideToDatabase(car, places));
6          dispatch(setCarPosition(car, 0));
7        } else {
8          dispatch(setCarPosition(car, 1));
9        }
10       return;
11     }
12     const dir = [...directions];
13     const next = dir.shift();
14
15     let newi = i;
16     setTimeout(() => {
17       const newCar = {
18         id: car.id,
19         coordinate: {
20           latitude: next.latitude,
21           longitude: next.longitude,
22         },
23         regNr: car.regNr,
24       };
25       dispatch(moveCar(newCar));
26       dispatch(driveCar(dir, newCar, newi += 1, type, places));
27     }, 200);
28   };
29 }
```

Listing 6: JavaScript code to simulate ride

When all coordinates in the list have been removed, the simulation is over, and depending on if the car arrived at the pickup or destination location, different database calls are made. If the car is at the pickup location, the car position will be updated in the database, and the car status will also be set to ”booked” (line 8). Ideally, the car position should be updated at every step of the way, and the booking status should be updated once the ride is booked, but it was solved this way to improve the performance of the application when testing it, as continuous calls to the database would negatively affect the performance of the app. When real cars are added to the application, their position will have to be continuously updated in the app, and this simulation will not be needed.

When the car arrives at the destination, the car position is again updated, and the booking

status is now set to ”available”. In addition to this, the ride is here stored in the database, by dispatching the addRideToDatabase-function in line 5. The addRide-function called is included in listing 7. It takes the car used as well as an object containing all locations from the ride as arguments and creates a POST-request to send to the database. The start and pickup times are not stored in the app state, so they are calculated based on the estimated durations from Google since the simulated drive only takes a fraction of the actual duration. The user taking the ride is also stored in the database, but as no user profiles or login functionality is implemented in the app, this is hardcoded to be the user with ID 1.

```
1  export async function addRide(car, places) {
2    const request = new Request(`http://${HOST}/rides.php`, {
3      method: 'POST',
4      headers: {
5        Accept: 'application/json',
6        'Content-Type': 'application/json',
7      },
8      body: JSON.stringify({
9        car_id: car.id,
10       user_id: 1,
11       start_latitude: places.startCoordinates.latitude,
12       start_longitude: places.startCoordinates.longitude,
13       start_time: (Date.now() / 1000) - places.startTime,
14       via_latitude: places.pickupCoordinates.latitude,
15       via_longitude: places.pickupCoordinates.longitude,
16       via_time: (Date.now() / 1000) - places.pickupTime,
17       end_latitude: places.destinationCoordinates.latitude,
18       end_longitude: places.destinationCoordinates.longitude,
19       end_time: Date.now() / 1000,
20     }),
21   });
22   const response = await fetch(request);
23 }
```

Listing 7: JavaScript code to add ride to database

Known bugs

Addresses that exist in several cities have been a problem during testing. Street names like ”Kongens gate” and ”Prinsens gate” are common in many Norwegian cities, and the first result from the Google API is for the street in Oslo. Adding Trondheim in the text input field was a quick fix used during testing, but this is not implemented in the application.

3.6 Evaluation

After the development of the application, a user test was conducted to investigate how usable the app is, and to identify any problems with the graphical user interface. The user test was executed in two rounds, first a qualitative test to identify initial problems with the design or functionality, and secondly, a quantitative test to find a usability score after fixing problems and adding new functionality. After testing the app, the testers were asked to answer a questionnaire with questions both regarding their background and their experience with the application.

3.6.1 Number of testers

The number of testers needed to get significant results differ based on the type of test conducted. For a quantitative study, Nielsen (2012) recommends a minimum of 20 users to achieve statistically significant results, while a qualitative test aimed at identifying usability problems should have a group of 5 testers. Users tend to do very similar things, so for every new tester, less and less new problems will be identified (Nielsen 2000). With five testers, approximately 85% of problems will be found, which is a good amount for iterative design with several test phases. A second user test after fixing problems found in the first can both serve as quality assurance to verify that problems were solved, as well as identify new issues or problems not found in the first test. In addition to this, it could also provide more in-depth insights about the app, regarding things such as task flow or information architecture.

3.6.2 First test

The first user test was conducted on a small number of users, as the goal was to identify usability problems and investigate the potential to use the app. The testers were given a task sheet, which included a summary of the product as it is intended to be, as well as some tasks to complete using the app. These tasks were chosen so the users would have to try the different functionalities added, and they had to be executed on a mobile device given to them while being observed. They were not allowed to ask questions on how to complete the tasks. The task sheet is included in appendix D.

After completing the tasks, participants were asked to answer an online questionnaire, which

first surveys their background to see if different age groups or professions were more inclined to be positive towards the product. Further, they were asked to answer the SUS form. Although this test was qualitative, it was interesting to see how usable the first version of the app was, and to see if the SUS score was improved in the later test phase. Finally, the participants were asked some questions regarding the product and the functionality of the application, to identify possible problems they had executing the tasks. The questionnaire is included in appendix E.

3.6.3 Second test

The second test focused on finding how usable the application is, and the primary goal here was to find a SUS score, as well as further investigate people's attitude towards the product. Since this was a quantitative test, more participants were needed. The same task sheet, included in appendix D, were used for the second user test, but the testers were not observed while executing the tasks, meaning that they had no opportunity to ask questions or receive guidance. The implementation of the rides API was done before the execution of this test, so it was also possible to see if the testers were able to do all tasks based on the rides stored in the database.

The participants for this test were also asked to answer an online questionnaire after completion of the tasks. The first part of this questionnaire is similar to the first test questionnaire, asking about the participants' background and normal car usage. Further, they were also asked to fill out the SUS form. The final part of the questionnaire also here contains some questions about their attitude towards the product, but in more detail than in the first test, and they were also asked to identify why or why not they would like to use the product. This questionnaire is included in appendix F.

Chapter 4

Results

This chapter presents the results of the project. The first section focuses on the result of the development of the application, while the second section presents the results from the user tests. Since the development of the app was divided into two iterations, the results will also be displayed for each iteration.

4.1 NAP App

4.1.1 Mobile devices

The application has been tested on a selection of Android devices, both physical devices and emulators, shown in table 4.1. These devices cover some common Android devices, with different screen sizes and resolutions, and different versions of the Android operating system. Since the development was done on a Windows computer, the app has not been tested on any iOS devices.

4.1.2 First iteration views

The main functionality of the application is the ability for users to book a ride to a specific destination, from a specific pickup location. The views for how to achieve this after the first iteration of the development are presented in this section. The start screen of the app is displayed in figure 4.1a. It consists of a map with all available cars and the user's current location as a

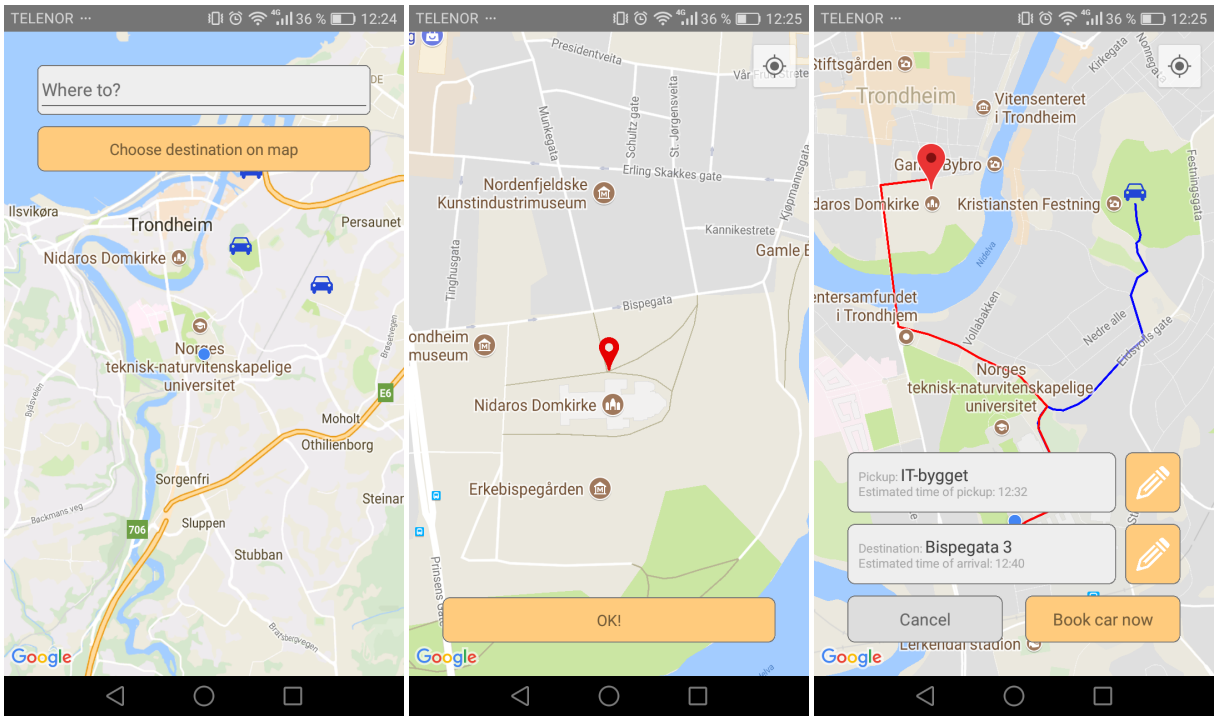
Phone	Operating system	Physical/emulator
Huawei P9	Android 6.0	Physical
Huawei P9 Lite	Android 7.0	Physical
Google Pixel XL	Android 6.0	Emulator
Nexus 5X	Android 6.0	Emulator
Samsung Galaxy S8 Plus	Android 8.0	Physical
Samsung Galaxy XCover 4	Android 7.0	Physical

Table 4.1: Tested mobile devices

blue dot. The user is presented with two alternatives: he can type the destination address in the text field, or tap the button to set the destination with a pin on the map. If pressing this button, a view with the pin in the middle of the screen is displayed, and the user can then zoom and move the map, and press ”OK” when the pin is over the wanted destination (figure 4.1b). After choosing the destination, either way, the route from the closest car via the user's position to the destination is calculated and displayed, along with two infoboxes with pick-up and drop-off addresses and time estimates, as well as buttons to change either locations, and to book or cancel the ride (figure 4.1c). Canceling the ride will result in the start screen being displayed again.

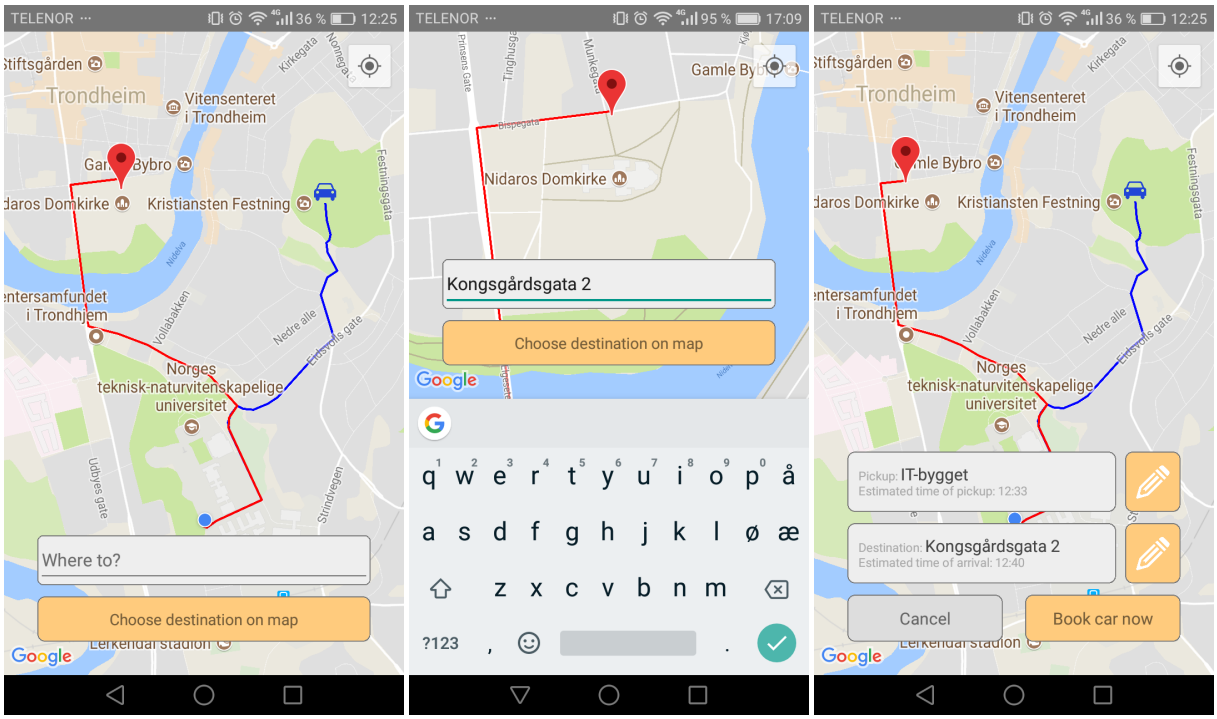
If the user chooses to edit the destination, the view in figure 4.2a is displayed, and a similar view will be shown if he chooses to edit the pickup location. Also here, the user has the choice between typing the address or placing a pin on the map. When tapping the text input field, the keyboard is displayed as shown in figure 4.2b. After updating either the destination or pickup location, the new route is calculated and displayed with updated info about the destination or pickup address and time estimate, as in figure 4.2c.

Finally, when the user is happy with both the pickup and drop-off locations, the ride can be booked by pressing the ”Book car now”-button. The car in the app will then simulate driving according to the directions given, and a new infobox with estimated pickup and drop-off times are displayed, as in figure 4.3a. It is not possible to zoom or move around the map when the car is driving. When the car arrives at the pickup location, the user is notified and has to manually continue the ride from the app, with the button displayed in the view in figure 4.3b. After continuing the ride, only the estimated arrival time is shown while the car continues the



(a) Start screen **(b)** Set destination with marker **(c)** Directions screen

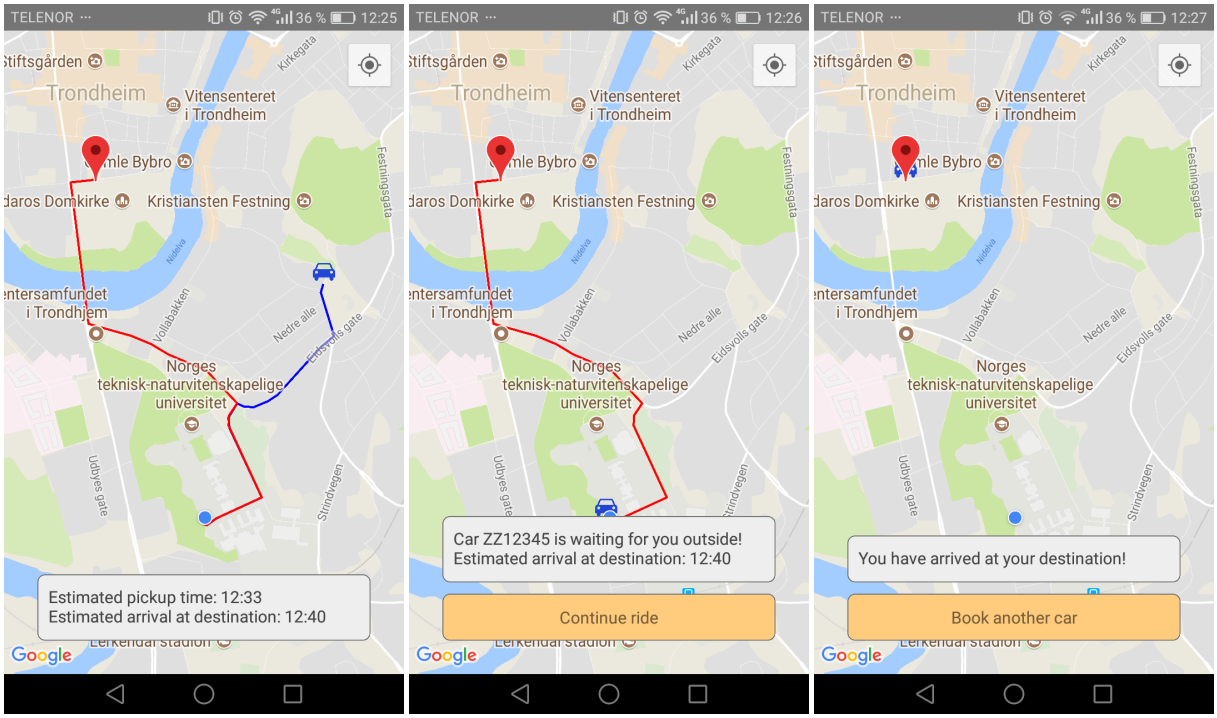
Figure 4.1: Requesting ride views, iteration 1



(a) Edit destination **(b)** Typing address **(c)** Updated directions screen

Figure 4.2: Editing destination views, iteration 1

simulation drive towards the final destination. Finally, the car and user arrives at the destination, and the user can choose to book a new ride in the app, with the button in the view in figure 4.3c. If the button is pressed, the app returns to the start screen, but with an updated position of the car. Since the app does not communicate with the database, closing the app and opening it will result in the cars being set at their original positions.



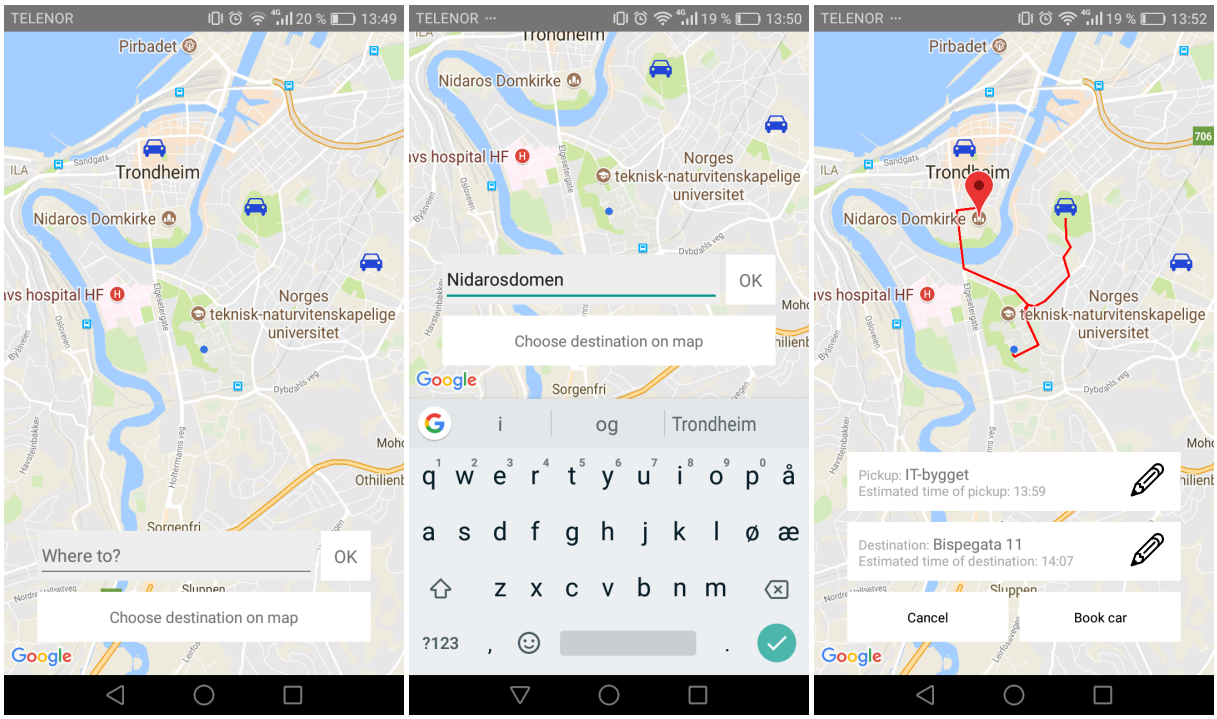
(a) Drive to pickup **(b)** Pickup screen **(c)** User arrived at destination

Figure 4.3: Ride simulation views, iteration 1

4.1.3 Second iteration improvements

Based on feedback from the user test after the first iteration, some design adjustments were made to the application in the second iteration. The database and all communication with it were also added in this step and implemented in the app. First, the design has changed a little, to make it more universal and clean. For example, all orange elements are removed, and the address input component is moved to the bottom of the screen also for the start screen, as shown in figure 4.4a. The placement of pins on the map works the same way as described for iteration one, but an ”OK”-button is added next to the text input field for the address (figure 4.4b), as feedback from the testing after iteration one revealed that several users mistook the other button for an ”OK”-button. The view containing the best car and directions can be seen in figure 4.4c,

with some design changes from the same view after iteration one (figure 4.1c). It is also now possible to tap the entire infobox to change either the pickup or drop-off location instead of just a button.

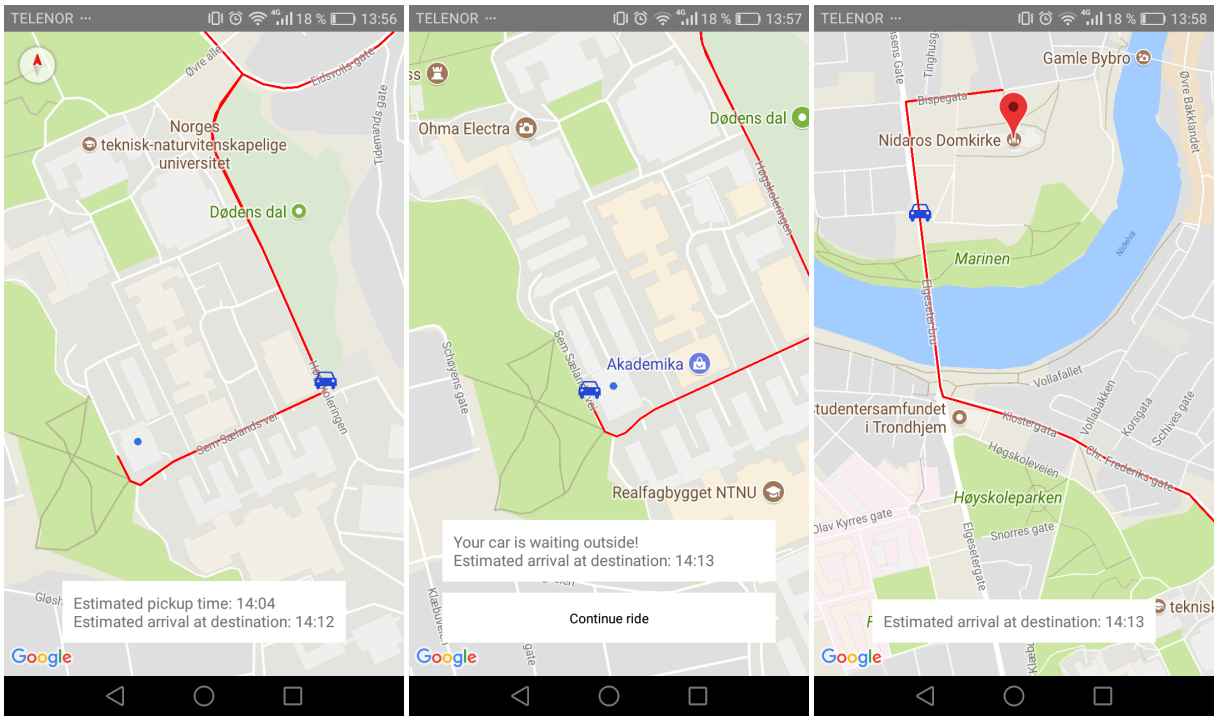


(a) Start screen **(b)** Typing address **(c)** Directions screen

Figure 4.4: Requesting ride views, iteration 2

The ride itself does not appear very different to the user, except for a couple of small changes. The route the car already has driven is not removed, both to improve the performance a bit and for the user to keep track of how far the car has driven. The most significant change, however, is that the bug in iteration one, where it was not possible to move or zoom the map once the ride has started, is fixed. The simulation apart from this is solved in the same way, and the same information is given to the user. A couple of views from the new drive simulation and notifications can be seen in figure 4.5. Finally, when the ride is over, the app immediately changes to the start screen with the updated position of the car. A demonstration video can be seen in the YouTube video linked to in appendix G.

The biggest improvement of the second iteration is the communication with the REST API and therefore also the database. For simplicity in this version of the app, the car's position is set when the user is picked up and when the ride is over, not at every single step of the ride. The ride is also saved to the database when it is finished. This means that if the user closes the



(a) Drive to pickup **(b)** Pickup screen **(c)** Drive to destination

Figure 4.5: Ride simulation views, iteration 2

app during the ride, it will not be saved to the database. As no signup or login functionality has been implemented, the user part of the ride is hardcoded, so all rides in the database are taken by the user with ID 1.

4.1.4 User flow diagram

The diagram in figure 4.6 describes the flow a user can take to book and complete a ride in the final system.

4.2 Evaluation

This section presents the feedback given from the users in the two user tests.

4.2.1 First user test

The test was conducted as an online questionnaire after the user was given a set of tasks he or she performed. The questionnaire for the first user test is included in appendix E, and the

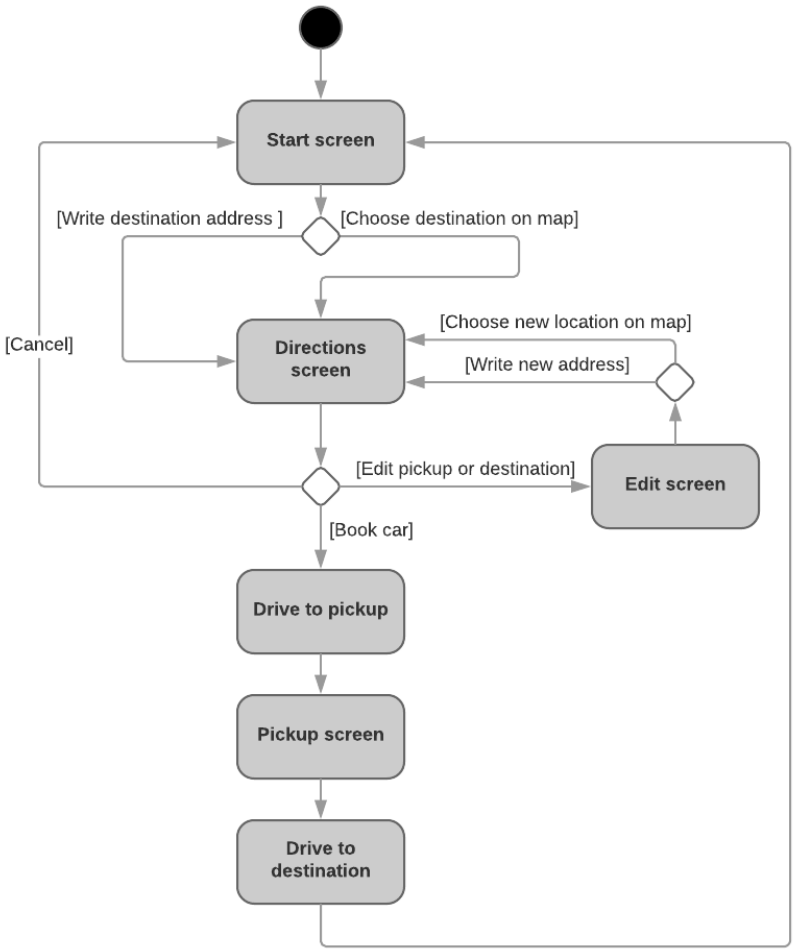


Figure 4.6: User flow diagram

responses are linked to in appendix H.1.

Participants

In total nine people tested the application in the first user test. Since this was a qualitative test, more testers were not needed to identify problems with the UI. 7 of the participants were students between ages 20 and 29, while one was between 40 and 49 years old, and one under the age of 20. When it comes to their line of study or profession, five was in the field of computer science, and the remaining four in different occupations. It was also asked whether or not the participants owned a personal car, where three did, three were planning on purchasing a car in the future, and the remaining three did not own a car. Out of the participants owning a car, one commonly use it once or twice each day, one usually use the car a few times each month, and one participant almost never uses their car.

Identified problems

Multiple problems with the UI was identified by the participants. Five of the participants mistook the ”Choose destination on map” for an OK-button when entering the address, and requested a separate and more visible OK-button to choose the address of their destination or pickup location. A couple of participants were not able to change the pick-up location because of this. Another problem with the application that a couple of participants mentioned was the ability to zoom or move the map while the car is driving, as this was not possible. One of the tasks was to specify the drop-off location to the east entrance of a building (while the main entrance is at the west side), and three users were not able to figure out how to do so, while some others did not try. One user also mentioned that the design of the app could be improved.

SUS score

A SUS score was calculated to find the potential usability of the app. The resulting SUS score for the computer scientists and the others, as well as the total score, is presented in table 4.2.

Discipline	**SUS Score**
Computer Science	84.50
Other	68.75
Total score	**77.50**

Table 4.2: SUS score by discipline, first user test

Attitude towards product

The participants were asked about their attitude towards the product, particularly if they believed it would make their life easier and if they think they would have signed up for it when it becomes available. The answer distribution for this can be seen in figure 4.7 and 4.8.

In addition to these questions, the participants were also asked to identify anything that could be added or removed to make the product more attractive to use. Regarding the cars, one participant requested more information about the cars, such as its color, type and size, and another suggested the opportunity to choose what car size to book, as one sometimes might need a bigger car with more luggage room or car seats. The price of the ride was also mentioned,

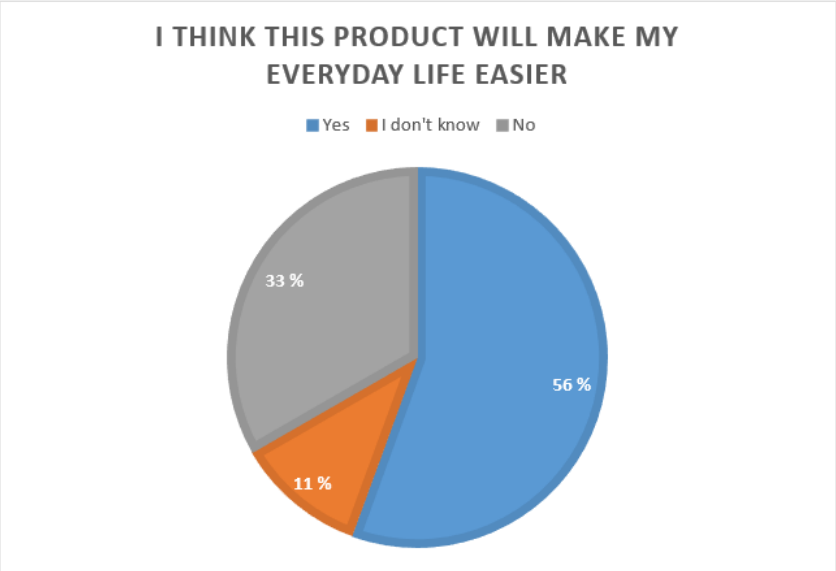


Figure 4.7: Answers regarding easier life, first user test

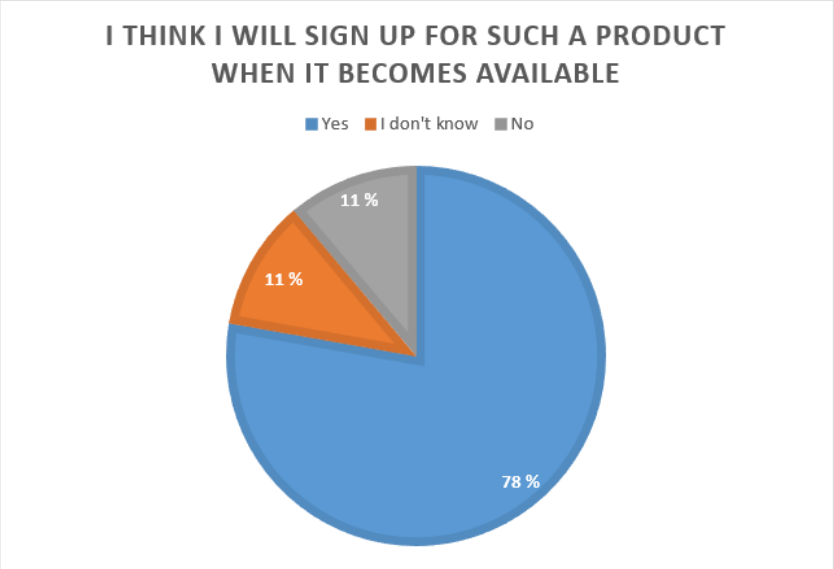


Figure 4.8: Answers regarding potential signup, first user test

as the current version does not include any information about this, so it was suggested that including some calculations would improve the product. A fourth suggestion was the ability to plan a ride in the future, not just exactly when the car is booked.

4.2.2 Second user test

The second test was also conducted as an online questionnaire after the user had completed a set of tasks. The questionnaire is included in appendix F, and the responses are linked to in

appendix H.2.

Participants and background

To evaluate the usability of the system, people from several different disciplines tested the application. The majority of the in total 18 participants were in the field of computer science or IT, including both students and software and system developers, and table 4.3 shows the distribution among respondants of the user test. As in the first user test, the participants were asked about their usual car habits. 61.1% of the respondents own a personal car, and their normal car usage can be seen in figure 4.9.

Discipline	Participants
Computer science/IT	3
Computer science/IT (student)	6
Other profession	5
Other profession (student)	1
Retired	3

Table 4.3: Participants by discipline, second user test

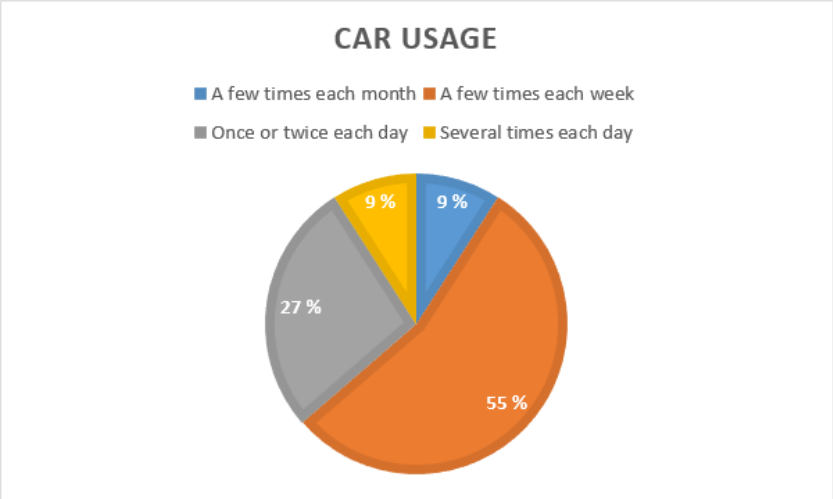


Figure 4.9: Participant car usage, user test 2

SUS Score

The questionnaire contained the standard SUS questions focusing on the usability of the application. The resulting scores per item in the questionnaire can be seen in table 4.4, where each item has received a score between 0 and 4 and 4 is the most positive for the application, and the total SUS scores per discipline are displayed in table 4.5.

Question	**Score**
I think that I would like to use this system frequently	2.72
I found the system unnecessarily complex	3.55
I thought the system was easy to use	3.44
I think that I would need the support of a technical person to be able to use this system	3.44
I found the various functions in this system were well integrated	2.94
I thought there was too much inconsistency in this system	3.61
I would imagine that most people would learn to use this system very quickly	3.55
I found the system very cumbersome to use	3.83
I felt very confident using the system	3.00
I needed to learn a lot of things before I could get going with this system	3.55
Total SUS score	**84.16**

Table 4.4: SUS score per question, second user test

Discipline	**Score**
Computer science/IT	91.95
Other profession	79.58
Retired	70.00
Total SUS score	**84.16**

Table 4.5: SUS score per discipline, second user test

Attitude towards product

The second test contained more detailed questions about the product and how it could impact the participants' everyday life as well as their community. The answers to these questions, both divided by the car owner status of the participants and for all participants combined, can be seen in table 4.6, where each statement received a score between 1 and 5, 5 being the best possible score.

	Average			Median		
Question	Total	Car	No car	Total	Car	No car
This product will make my everyday life easier	3.39	2.91	4.14	3	3	4
I will sign up for such a product when it becomes available	3.61	3.00	4.57	4	3	5
This product will remove my need for a personal vehicle	2.78	2.20	3.71	3	2	4
My community will benefit from this product	4.06	3.82	4.43	4	4	5

Table 4.6: Average and median for product questions, per car owner status

Textual feedback

The participants were also asked to list reasons for using or not using the product, as well as providing suggestions for functionality to add or remove to both the product itself and the mobile application, to make it more attractive to use. They could also write down any problems they had with completing the tasks, and as the rides were stored in the database, it could be checked whether or not they were able to complete the tasks as intended. Finally, the participants also had the opportunity to provide additional comments, if they had any.

Multiple participants mentioned the ease of use as a good reason to use the product. Environmental reasons and the opportunity to carpool was also mentioned by several, and some mentioned that it could be both cheaper and less work than owning a personal car. Other reasons mentioned was that it could be practical and flexible to use, for example when doing errands in the city or for elderly living in rural areas with minimal access to public transport.

Those participants who were negative to signing up for such a product all owned a car

themselves, and some mentioned that they would rather use public transport or their car. One participant mentioned that it could be used as an alternative to taxis if the price is right, but that it would not replace the need for a private car. The security factors of the product were also mentioned as a possible downside with such a product, and the feeling of being watched as the app would know the user's position at any time as well as their travel patterns.

Most participants did not have any problems completing the tasks, but some said that they struggled a bit at first to figure out how to book a car, and tried to press the map where they wanted to go. Some also mentioned that they did not figure out how to specify a particular side of a building as the drop-off destination. The latter was also confirmed by the rides in the database, where most of the testers were dropped off at the main entrance of Lerkendal Stadium, instead of the east entrance as specified in the tasks.

Some participants also made suggestions for functionalities and possible additions to the product and app to make it more interesting to sign up. More information about the car, such as the type and amount of seats, was mentioned by several. Because of the problems mentioned, several users also suggested to include some user manual with demonstration and instructions the first time one uses the app. The security aspect was mentioned here as well, as a user wanted more information about the privacy and security of the application. It was also mentioned to add auto-complete functionality in the address fields as a potential improvement that would make it easier for users to use the app.

Chapter 5

Discussion

This chapter discusses the research questions defined in chapter 1.2, based on the results presented in chapter 4 as well as some of the background research presented in chapter 2.

5.1 RQ1: How can autonomous cars and the sharing economy satisfy the needs of the society?

The background research and answers from the user tests regarding the respondents' attitude towards the product were evaluated to answer the first research question. The results show promise, and the combination of autonomous cars and the sharing economy can lead to several benefits for the society, but an attitude change is required for people to be willing to give up their personal vehicles.

Accessible transportation is an important need for people, but the amount and quality of public transportation vary, and few ridesharing platforms for daily trips exists, so private vehicles are often used. The number of cars on the roads thus keeps increasing, resulting in more traffic congestion and time spent in traffic. The cars are however not used efficiently, as they are parked more than 90% of the time on average, which means that fewer cars are needed in total in communities, and sharing a pool of cars could therefore be a good solution to satisfy the transportation needs of the society and even improve the current situation. Combined with autonomous cars, an on-demand service can be offered, which could further enhance mobility as these cars can communicate and optimize their driving and the total flow of traffic, reducing

congestion.

The safety of people is another major need in today's society, and both the UN and several individual countries have defined goals to reduce the number of people killed in traffic. Although this number already is decreasing, a lot of people are still killed or seriously injured in traffic accidents, often because of human error or choice in traffic. Autonomous vehicles can eliminate these factors and thereby help to achieve these goals and contribute to safer overall traffic. This does require a high rate of autonomous vehicles on the roads, as the benefits are higher if all human errors can be eliminated and the cars can communicate.

The environment and a sustainable planet is an essential need of today's society, and fuel emissions are a big problem in many parts of the world, contributing to poor air quality and increased climate changes. Autonomous vehicles can contribute to reducing emissions with their optimized driving and road usage, and combined with ridesharing fewer cars are needed, further reducing fuel emissions. As most commute trips happen in single occupant vehicles in the morning and afternoon, carpooling has especially huge potential here. Several participants of the user test mentioned the environmental factor as a reason to use the product, indicating that it is important to people when choosing transportation method, and autonomous carsharing could, therefore, satisfy the environmental need of the society.

The car owners among the respondents to the user test were negative towards giving up their private vehicle and replacing it with shared autonomous cars, even though most of them used their car less than daily. They mentioned the convenience of owning a car and did not think that the product would make their lives easier, whereas the participants who did not own vehicles believed their lives would become more comfortable with such a product. This indicates that an attitude change is needed in society for such a product to become a success, but the fact that those who do not own cars are positive shows promise, and could mean that the next generations believe that this product can satisfy their transportation needs.

Although some of the respondents were negative towards using the product themselves, there was a consensus that it would benefit their communities, where the score of 4.06 out of 5 indicates that the public believes that there is a need for the product in most communities. Some mentioned the mobility of everyone as an advantage, which is backed up by research and can contribute to a society with fewer differences between people. Other factors mentioned were the ease of use and price, which could have been highlighted more to the participants of the user

tests, as being a part of a carsharing scheme can reduce car-related costs significantly, especially if the car only is used a few times each week.

All in all, there are many needs of the society when it comes to transportation, mobility, and safety. This combination of the sharing economy and autonomous vehicles as an on-demand service, with the help of the public and governments, as changes in laws and regulations are required for the cars to become publicly available, could potentially replace private vehicles in the future and satisfy all such needs, but the convenience of owning a car is an essential factor for people and needs to be addressed in this product as well.

5.2 RQ2: To what extent is a mobile application perceived as a usable tool for booking cars?

To answer this question, a first version of a mobile application to handle all communication between end users and cars was developed from scratch. The application was tested through two separate user tests, to evaluate if it is usable for booking rides. For the product to become a success, it is important that the cars can be booked easily, so it was interesting to investigate whether a mobile app could handle this functionality and if all users were able to book rides without problems. To measure the usability, a usability test using the System Usability Scale was carried out, presented in sections 4.2.1 and 4.2.2.

The first user test aimed at identifying usability problems, but a SUS score was also calculated. This gave a result of 77.5, which is an above average score, but there was a significant difference between the score for the participants in the field of computer science (84.5) and the rest of the participants (68.75). A score of 68.75 is an average usability score, and this implies that the application was not that usable for the entire target group, but that it had potential. Because there only was 9 participants for this user test, the numbers are not considered significant, but gives an indication on how usable the system is. The comment section supported this indication, as the comments were positive, but with some suggestions for possible improvements.

The second user test, after some improvements had been made to the app, aimed at finding the final usability score, and resulted in a total score of 84.16. The number of participants for this test was 18, with some distribution among age groups and professions, giving a more

representative usability result than the first test. The application should, however, have been tested on an even bigger range of people to cover more of the intended target group. The usability score is a definite improvement from the first test, and indicates an application with good usability, but still with room for improvement. The computer scientists gave the app a total score of 91.95, which is considered very good, whereas the group of retired participants only gave a score of 70, slightly above average. The rest of the participants resulted in a score of 79.58, which is also considered good usability. The app shows potential even for the older part of the society, but their answers indicate that they could need some more guidance to be able to use the application. As one of the benefits of this business model is to improve mobility also for older people who cannot drive, it is important that they too are able to use the application without problems.

The scores for the separate statements in the SUS questionnaire were also analyzed, to evaluate if they could say something about how usable the app is. Each statement receives a score between 0 and 4, where 4 is the best and indicates a close to perfect system. Only two statements received scores lower than 3, including ”I think that I would like to use this system frequently” and ”I found the various functions in the system well integrated”. The first statement receiving a lower score does not necessarily mean that the application is not usable for its purpose, it could also imply that the whole product is not attractive to the user, as discussed in section 5.1. The second statement indicates that better integration of functionalities in the application could make it more usable for car booking. The statement receiving the significantly highest score was ”I found the system very cumbersome to use”, with a score of 3.83, which means that most people did not find the system cumbersome, but rather easy to use with the purpose of booking cars.

The textual responses to the user tests also indicated that most users were able to complete the different car booking tasks without significant problems, but several users reported difficulties with specifying a particular side of the building to be dropped off at. This indicates that the functionality to place a pin on the map for a more specific location was either challenging to find or difficult to master, and is a clear subject for improvement. Some participants in the tests also had problems finding out how to start requesting a ride, but once they figured it out, they found it easy to manage. One of the usability heuristics is that systems should prefer recognition over recall, and this problem indicates that the recognition is not perfect for this app, and should be

improved, so the users understand what to do instead of having to remember it.

Apart from these two problems, most users reported that they found it easy to complete the tasks. Their answers imply that the app achieves both high effectiveness as most users were able to accomplish the goal of booking a ride, high efficiency as they do not require many resources to do so, and relatively high satisfaction as their comments about the application was positive. Some participants also mentioned that the application was practical and easy to use, which confirms these indications. The good usability score of 84.16 also supports that the application is a usable tool to book rides, but that there is some room for improvement.

5.3 RQ3: To what extent is it feasible to develop a mobile application for simulation of car booking?

The third question is answered based on the development process and the resulting application. The app development was done using the cross-platform framework React Native, to save time and resources by having a common codebase for both iOS and Android. Since the app only was tested on Android, it is unknown if this was an advantage, or if it would have been better to use the native IDEs and languages. The Android application achieved a good usability score, but some requirements were not prioritized to implement as getting to know the frameworks and technologies took more time than anticipated. Especially Redux was not familiar, and it could have been better to use technologies already well-known to the developer. Ideally, more technologies should have been tested in more detail, to be sure that the right decision was made. However, the final result received positive feedback, indicating that the technologies used were well-suited for this purpose.

Most of the requirements defined in section 3.3 were fulfilled, but some were as mentioned not prioritized in this first version of the app. Out of the functional requirements, all with high priority were met, but some of the medium and low priority requirements were not implemented. More specifically, FR10, FR11, FR12, FR14, and FR15 were not accomplished in this version of the app. A couple of the non-functional requirements were also not met, particularly NFR4 and NFR6. Several of the less prioritized requirements were regarding the functionality for several users to use the app and book cars simultaneously, and none was explicitly about the

main car booking functionality. An application for only car booking is therefore achievable to develop and simulate using the chosen technologies.

From the feedback given in the user tests, the application worked well for its purpose, and although most participants only were a little positive to use the app, this was probably more due to the entire product and not because the application did not work to book cars. This indicates that it is feasible to use an app for this purpose, and as the application received positive feedback, it is likely that the solution for the development was well-chosen. When it comes to the simulation part of the application, this worked well for this version, but it should have been tested to continuously update the cars' position in the database and refresh the application, to see if the performance of the application was still good.

5.4 Limitations

As the project was started from scratch with no previous work done, a lot of time was spent getting up to speed, resulting in less time for the development. Also, the technology research to find the best choices for the requirements was time demanding, and as both React Native, Redux and PHP were unfamiliar to the developer, some time was spent getting to know these, instead of time spent focusing on the development of the application.

For the user tests, the participants were found in and around the developer's network. Ideally, a more diverse group of users should have tested the app, and they should have been unknown to the developer to get rid of any bias that could occur. Additionally, the participants only received a short description of the product without many details. If they had known more about the product and the benefits of it, they could might have been more positive towards it.

Chapter 6

Conclusions and future work

This chapter presents the conclusion for this project as well as recommendations for future work.

6.1 Conclusion

In this thesis, the concept of combining fully autonomous vehicles with the sharing economy has been researched, and a product where people become members to share a pool of self-driving cars has been introduced. A mobile application is needed for members to book rides with the autonomous cars, and the goal of this project was to investigate how such an application could best be developed, as well as studying how usable a car booking application is as a tool to organize transportation, and examine people's attitudes towards the product. Three research questions were defined to address these goals.

- RQ1: How can autonomous cars and the sharing economy satisfy the needs of the society?
- RQ2: To what extent is the mobile application perceived as a usable tool for booking cars?
- RQ3: To what extent is it feasible to develop a mobile application for simulation of car booking?

To answer these questions, the first version of this application was developed using state-of-the-art technologies. The application was named NAP App, based on the parent project

NTNU Autonomous Perception for autonomous vehicles. The technologies used include React Native as a cross-platform framework to develop native applications, Redux to handle state management and data flow, MySQL to implement the database, and PHP to develop REST APIs for communication between the application and the database. The application was developed with a focus on being easy to use for the users, and this was tested through two rounds of user tests, with a short development phase in between to fix usability problems.

The results show that the combination of autonomous cars and the sharing economy can lead to huge benefits and satisfy the needs of the public both as individuals and a community, but an attitude change among people is required for the benefits to be maximized. The application received positive feedback and was given a good usability score, and this along with the fact that all participants in the user test managed to book cars indicates that the mobile application is a usable tool for booking cars. Some users did, however, struggle with some tasks and had a little trouble figuring out how to start, so although the application is easy to use, both the UI and the functionalities can be improved and expanded. As the app received positive feedback and was able to handle the main feature to book and simulate a ride with an autonomous car, as well as store and update information in the database, it can be concluded that it is highly feasible to develop and use a mobile application for this purpose.

6.2 Future work

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

6.2.1 iOS

As the application only has been tested on Android devices, it should be prioritized to get it running on an iOS device and test both all functionality and the UI before adding any more functionality.

6.2.2 Cars

Many users requested more information about the cars before they booked them, as different trips require different types of cars. This could be implemented as an option when requesting a

ride, so the user could specify if there is a need for a particular type of car for that ride. More information about the physical appearance of the car could also help the user identify the car when being picked up. Additionally, the cars' position should be updated continuously, so the user can see the cars moving at all times when using the application.

6.2.3 Ridesharing

For the community to gain maximum benefits and for rides to be cheaper, the possibility for ridesharing or carpooling should be researched further and implemented in the app. This could include both several members sharing a vehicle from the same pickup location to the same destination, or an algorithm for picking up several members who are going the same direction at the same time, but from different starting points.

6.2.4 Car distribution

The current car distribution algorithm only takes the distance between the cars and the user booking a ride into account. When several users are added, and there is a possibility that several users are requesting rides at the same time, or even sharing a ride from multiple pickup and drop-off locations, the algorithm should be expanded to take this into account as well. This needs to be researched in more detail and tested, but looking at it as a flow network and using an adapted maximum flow algorithm might be a good place to start. It is possible that machine learning could be beneficial to use here.

6.2.5 User profiles and login

To facilitate for several users booking cars at once and possibly also share cars, they need to be able to create a personal user profile and log in to the application. Some functionality for users are added to the database, but this is not implemented in the application and is considered a necessary step to increase the functionality and potential for the application to be used. The login functionality should be secure, and the privacy of the members should be considered.

6.2.6 Pricing and payments

The pricing of the product needs to be researched further, to find a reasonable price for both the general membership and a pricing model for rides. This should also be included in the app, as several of the testers mentioned that they would like to see details about this before booking a ride. Some also said that the price of the service was an essential factor when it comes to deciding whether to sign up or not. Payment opportunities should also be included in the app for each user, all payment must happen directly in the application since there are no drivers to pay directly.

6.2.7 Addresses

Several users requested suggestions when searching for addresses, similar to what is offered by Google, to improve the usability and accuracy. This could also solve the problem of some addresses giving results in other cities than intended, as the user could choose the correct address from the proposals. Some restrictions on where rides users should be able to book rides to could also be implemented here, only allowing, for example, booking to addresses within the same city or a certain radius.

6.2.8 Rides

In the current version of the application, it is only possible to cancel or edit a ride before it is booked, the user cannot change any details once the car has started driving. Adding more functionality here could improve the usability, with for example enabling the user to change his mind and cancel the ride at any time, or editing the pickup or drop-off location after the ride has started. Additionally, the user should be able to book a ride for a later point in time and to book recurring rides, e.g., every morning at 7:30 from home to work.

6.2.9 Web service for admins

A web service for administrators to add, remove or edit cars, retrieve statistics about rides, users, and cars, as well as being able to add, edit or remove users manually should also be considered to be implemented and connected to the REST API, in order to make the system more usable

for everyone, not just the end user. The REST API could also be expanded to retrieve more information or combination of information if needed.

Bibliography

Achour, M. et al. (2018), 'PHP Manual'. Online; accessed Apr. 25, 2018.
URL: *http://php.net/manual/en/*

Android (2018*a*). Online; accessed Feb. 13, 2018.
URL: *https://developer.android.com/studio/intro/*

Android (2018*b*). Online; accessed Feb. 13, 2018.
URL: *https://developer.android.com/kotlin/*

Apple (2018*a*). Online; accessed Feb. 13, 2018.
URL: *https://developer.apple.com/develop/*

Apple (2018*b*), 'About swift'. Online; accessed Feb. 13, 2018.
URL: *https://swift.org/about/*

Barter, P. (2013), '"Cars are parked 95% of the time". Let's check!'. Online; accessed Apr. 21, 2018.
URL: *https://www.reinventingparking.org/2013/02/cars-are-parked-95-of-time-lets-check.html*

Bates, J. & Leibling, D. (2012), 'Spaced out: Perspectives on parking policy', *Royal Automobile Club Foundation* .

Bertoncello, M. & Wee, D. (2015), 'Ten ways autonomous driving could redefine the automotive world', *Retrieved from McKinsey & Company website: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/ten-ways-autonomous-driving-could-redefine-the-automotive-world* . Accessed Mar. 8, 2018.

Bilkollektivet (n.d.). Online; accessed Apr. 24, 2018.
URL: *https://bilkollektivet.no/en/*

BlaBlaCar (2018), 'Blablacar'. Online; accessed Apr. 03, 2018.
URL: *https://www.blablacar.com*

Brooke, J. (1996), Sus: A 'quick and dirty' usability scale, *in* P. W. Jordan, B. Thomas, B. A. Weerdmeester & I. L. McClelland, eds, 'Usability evaluation in industry', Taylor & Francis, chapter 21, pp. 189–194.

Budiu, R. (2013), 'Mobile: Native apps, web apps, and hybrid apps'. Online; accessed Feb. 10, 2018.
URL: *https://www.nngroup.com/articles/mobile-native-apps/*

Christensen, C. M., Raynor, M. E. & McDonald, R. (2015), 'What is disruptive innovation', *Harvard Business Review* **93**(12), 44–53.

Cohen, B. & Kietzmann, J. (2014), 'Ride on! Mobility business models for the sharing economy', *Organization & Environment* **27**(3), 279–296.

Components and Props (n.d.). Online; accessed Feb. 11, 2018.
URL: *(https://reactjs.org/docs/components-and-props.html*

Czaplicki, E. (2017*a*), 'elm'. Online; accessed Feb. 9, 2018.
URL: *http://elm-lang.org/*

Czaplicki, E. (2017*b*), 'An introduction to elm'. Online; accessed Feb. 9, 2018.
URL: *https://guide.elm-lang.org/*

Czaplicki, E. & Chong, S. (2013), Asynchronous functional reactive programming for GUIs, *in* 'Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation', Vol. 48, pp. 411–422.

DriveNow (2018), 'About drivenow'. Online; accessed Apr. 24, 2018.
URL: *https://www.drive-now.com/de/en/about*

Eisenman, B. (2015), *Learning React Native: Building Native Mobile Apps with JavaScript*, O'Reilly Media, Inc.

Elvebakk, B. & Steiro, T. (2007), *Nullvisjonen - i teori og praksis*, Transportøkonomisk institutt.

Facebook (2017), 'React'. Online; accessed Feb. 11, 2018.
URL: *https://reactjs.org/*

Facebook (2018*a*), 'Learn the basics'. Online; accessed Feb. 13, 2018.
URL: *https://facebook.github.io/react-native/docs/tutorial.html*

Facebook (2018*b*), 'Performance'. Online; accessed Feb. 13, 2018.
URL: *https://facebook.github.io/react-native/docs/performance.html*

Facebook (2018*c*), 'React native'. Online; accessed Feb. 13, 2018.
URL: *https://facebook.github.io/react-native/*

Facebook (2018*d*), 'Style'. Online; accesssed Feb. 13, 2018.
URL: *https://facebook.github.io/react-native/docs/style.html*

Fagnant, D. J. & Kockelman, K. (2015), 'Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations', *Transportation Research Part A: Policy and Practice* **77**, 167–181.

Fagnant, D. J. & Kockelman, K. M. (2018), 'Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in austin, texas', *Transportation* **45**(1), 143–158.

Farajallah, M., Hammond, R. G. & Pénard, T. (2016), 'What drives pricing behavior in peer-to-peer markets? Evidence from the carsharing platform BlaBlaCar'.

Friberg, J. (2017), 'React native vs native in mobile app development'. Online; accessed Feb. 16, 2018.
URL: *https://www.varvet.com/blog/react-native-vs-native-in-mobile-app-development/*

Gartner (2017), 'Gartner says worldwide sales of smartphones grew 7 percent in the fourth quarter of 2016'. Online; accessed Oct. 15, 2017.
URL: *https://www.gartner.com/newsroom/id/3609817*

Gerla, M., Lee, E.-K., Pau, G. & Lee, U. (2014), Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds, *in* 'Internet of Things (WF-IoT), 2014 IEEE World Forum on', IEEE, pp. 241–246.

Google (2017). Online; accessed Feb. 11, 2018.
URL: *https://angular.io/*

Hamari, J., Sjöklint, M. & Ukkonen, A. (2016), 'The sharing economy: Why people participate in collaborative consumption', *Journal of the Association for Information Science and Technology* **67**(9), 2047–2059.

Heinrichs, H. (2013), 'Sharing economy: a potential new pathway to sustainability', *GAIA-Ecological Perspectives for Science and Society* **22**(4), 228–231.

Hjorthol, R., Engebretsen, Ø. & Uteng, T. P. (2014), *Den nasjonale reisevaneundersøkelsen 2013/14: nøkkelrapport*, Transportøkonomisk institutt.

Høye, A., Elvik, R., Sørensen, M. W. J. & Vaa, T. (2012), *Trafikksikkerhetshåndboken*, Transportøkonomisk institutt.

IBM (2018), 'Three-tier architectures'. Online; accessed May 12, 2018.
URL: *https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.*
tier.html

International Organization for Standardization (2010), 'ISO 9241: Ergonomics of human system interaction-part 210: Human-centred design for interactive systems'.

Martin, E. W. & Shaheen, S. A. (2011), 'Greenhouse gas emission impacts of carsharing in north america', *IEEE Transactions on Intelligent Transportation Systems* **12**(4), 1074–1086.

Masse, M. (2011), *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*, O'Reilly Media, Inc.

Microsoft (2018), 'Xamarin.forms'. Online; accessed Feb. 17, 2018.
URL: *https://docs.microsoft.com/en-us/xamarin/xamarin-forms/*

Nabobil (n.d.). Online; accessed Apr. 24, 2018.
URL: *https://nabobil.no/en*

NHTSA (2018), 'Automated vehicles for safety'. Online: accessed Mar. 8, 2018.
URL: *https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety*

Nielsen, J. (1995), '10 usability heuristics for user interface design', *Nielsen Norman Group* .

Nielsen, J. (2000), 'Why you only need to test with 5 users'. Online; accessed Apr 2, 2018.
URL: *https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/*

Nielsen, J. (2012), 'How many test users in a usability study'. Online; accessed Apr. 2, 2018.
URL: *https://www.nngroup.com/articles/how-many-test-users/*

Nordbakke, S., Uteng, T. P. & Hjorthol, R. (2014), *Reisevaneundersøkelsen 2013/14: Samling av faktaark*, Transportøkonomisk institutt.

Occhino, T. (2015), 'React native: Bringing modern web techniques to mobile'. Online; accessed Feb. 15, 2018.
URL: *https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/*

Oracle (2018*a*), 'The main features of MySQL'. Online; accessed Apr. 19, 2018.
URL: *https://dev.mysql.com/doc/refman/8.0/en/features.html*

Oracle (2018*b*), 'What is MySQL?'. Online; accessed Apr. 19, 2018.
URL: *https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html*

Redux (2018*a*). Online; accessed Mar. 6, 2018.
URL: *https://redux.js.org/*

Redux (2018*b*), 'Actions'. Online; accessed Mar. 6, 2018.
URL: *https://redux.js.org/basics/actions*

Redux (2018*c*), 'Basics'. Online; accessed Mar. 6, 2018.
URL: *https://redux.js.org/basics/*

Redux (2018*d*), 'Reducers'. Online; accessed Mar. 6, 2018.
URL: *https://redux.js.org/basics/reducers*

Redux (2018*e*), 'Store'. Online; accessed Mar. 6, 2018.
URL: *https://redux.js.org/basics/store*

Rödel, C., Stadler, S., Meschtscherjakov, A. & Tscheligi, M. (2014), Towards autonomous cars: the effect of autonomy levels on acceptance and user experience, *in* 'Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications', ACM, pp. 1–8.

Schor, J. (2016), 'Debating the sharing economy.', *Journal of Self-Governance & Management Economics* **4**(3), 7–22.

Sidorenko, V. (2017), '7 best frameworks for web development in 2017'. Online; accessed Feb. 12, 2018.
URL: *https://gearheart.io/blog/7-best-frameworks-for-web-development-in-2017/*

Statistics Norway (2018*a*), 'Key figures for the population'. Online; accessed Apr. 20, 2018.
URL: *https://www.ssb.no/en/befolkning/nokkeltall/population*

Statistics Norway (2018*b*), 'Registered vehicles statbank'. Online; accessed Apr. 20, 2018.
URL: *https://www.ssb.no/en/statbank/list/bilreg*

Statistics Norway (2018*c*), 'Road traffic accidents involving personal injuries'. Online; accessed Apr. 20, 2018.
URL: *https://www.ssb.no/en/statbank/list/vtu*

Statistics Norway (2018*d*), 'Road traffic volumes statbank'. Online; accessed Apr. 20, 2018.
URL: *https://www.ssb.no/en/statbank/list/klreg*

The Global Goals (2018*a*), '11 sustainable cities and communities'. Online; accessed Apr. 21, 2018.
URL: *https://www.globalgoals.org/11-sustainable-cities-and-communities*

The Global Goals (2018*b*), '3 good health and well-being'. Online; accessed Apr. 21, 2018.
URL: *https://www.globalgoals.org/3-good-health-and-well-being*

The Good and the Bad of Angular Development (2017). Online; accessed Feb. 10, 2018.
URL: *https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/*

Uber (2018*a*), 'Advanced Technologies Group'. Online; accessed Mar. 22, 2018.
URL: *https://www.uber.com/info/atg/*

Uber (2018*b*), 'Always the ride you want'. Online; accessed Mar. 22, 2018.
URL: *https://www.uber.com/ride/*

Uber (2018*c*), 'How Uber works'. Online; accessed Mar. 22, 2018.
URL: *https://www.uber.com/ride/how-uber-works/*

Uber (2018*d*), 'Our story'. Online; accessed Mar. 22, 2018.
URL: *https://www.uber.com/about/*

Waymo (2018*a*). Online: accessed Mar. 21, 2018.
URL: *https://waymo.com/*

Waymo (2018*b*), 'Be an early rider'. Online: accessed Mar. 21, 2018.
URL: *https://waymo.com/apply/*

Waymo (2018*c*), 'Journey'. Online: accessed Mar. 21, 2018.
URL: *https://waymo.com/journey/*

Waymo (2018*d*), 'Technology'. Online: accessed Mar. 21, 2018.
URL: *https://waymo.com/tech/*

Waymo (2018*e*), 'Waymo safety report - on the road to fully self-driving'. Online: accessed Mar. 22, 2018.
URL: *https://waymo.com/safety/*

Witte, D. & von Weitershausen, P. (2015), 'React native for android: How we built the first cross-platform react native app', *Facebook Code* . Online; accessed Feb. 15, 2018.

World Health Organization (2015), *Global status report on road safety 2015*, World Health Organization.

Xamarin (2018). Online; accessed Feb. 16, 2018.
URL: *https://www.xamarin.com*

ZipCar (2018*a*). Online; accessed Mar. 22, 2018.
URL: *https://www.zipcar.com/*

ZipCar (2018*b*), 'The genius of car sharing'. Online; accessed Mar. 22, 2018.
URL: *https://www.zipcar.com/carsharing*

Appendix A

GitHub repositories

A.1 REST API

https://github.com/hannemathisen/napapp-rest-api

A.2 NAP App

https://github.com/hannemathisen/NapApp

Appendix B

Get rides REST API

```php
<?php
function get_rides($id, $car_id, $user_id, $start_time, $end_time) {
    global $connection;
    $query = "SELECT * FROM rides as r INNER JOIN cars as c ON r.car_id = c.car_id
                INNER JOIN users as u ON r.user_id = u.user_id";
    if ($id != 0) {
        $query .= " WHERE r.ride_id=".$id." LIMIT 1";
    } else if ($car_id != 0  $user_id != 0  $start_time != 0  $end_time != 0) {
        $where = false;
        if ($car_id != 0) {
          $query .= " WHERE r.car_id=".$car_id;
          $where = true;
        }
        if ($user_id != 0) {
            if ($where) $query .= " AND r.user_id=".$user_id;
            else {
                $query .= " WHERE r.user_id=".$user_id;
                $where = true;
            }
        }
        if ($start_time != 0) {
            if ($where) $query .= " AND start_time>=FROM_UNIXTIME(".$start_time.")";
            else {
                $query .= " WHERE start_time>=FROM_UNIXTIME(".$start_time.")";
                $where = true;
```

```
            }
        }
        if ($end_time != 0) {
            if ($where) $query .= " AND end_time<=FROM_UNIXTIME(".$end_time.")";
            else {
                $query .= " WHERE end_time<=FROM_UNIXTIME(".$end_time.")";
            }
        }
    }

    $response = array();
    $result = mysqli_query($connection, $query);
    foreach ($result as $row) {
        extract($row);
        $ride_item = array(
            "ride_id" => $ride_id,
            "car_id" => $car_id,
            "reg_number" => $reg_number,
            "user_id" => $user_id,
            "name" => $name,
            "start_latitude" => $start_latitude,
            "start_longitude" => $start_longitude,
            "start_time" => $start_time,
            "via_latitude" => $via_latitude,
            "via_longitude" => $via_longitude,
            "via_time" => $via_time,
            "end_latitude" => $end_latitude,
            "end_longitude" => $end_longitude,
            "end_time" => $end_time,
        );
        array_push($response, $ride_item);
    }
    header('Content-Type: application/json');
    echo json_encode($response);
}
?>
```

Appendix C

MapComponent

```
1  class MapComponent extends React.Component {
2    componentWillMount() {
3      this.props.getCars();
4      this.props.getLocation();
5    }
6
7    render() {
8      return (
9        <MapView
10         region={this.props.region}
11         style={styles.map}
12         onRegionChangeComplete={reg => this.props.onRegionChange(reg)}
13       >
14         {this.props.destination &&
15         <Marker
16           coordinate={this.props.destination}
17         />
18         }
19         {this.props.pickup &&
20         <Marker
21           coordinate={this.props.pickup}
22           title="Pickup location"
23         >
24           <Image
25             style={{ width: 7, height: 7 }}
```

```
26              source={require('./location.png')}
27            />
28          </Marker>
29          }
30          <DirectionsContainer />
31          <CarListContainer />
32        </MapView>
33      );
34    }
35  }
```

Appendix D

User test tasks

About the product

The system you are about to test is a mobile application meant to facilitate for the integration of the sharing economy with autonomous vehicles. The idea is that many people are sharing a pool of self-driving cars, and they use this app when they need a ride somewhere. The user can book cars through the app by saying where one needs to go, and the app will then allocate and book a specific car to pick the user up. This can eliminate the need for a personal car, and can have huge impacts on traffic, the environment and the need for parking spaces, among others. As this is a first draft for the app, it only contains some simple functionality, but in the future it could also be interesting to include the possibility to book a car for a specific time in the future, as well as to share a car if, for example, two users are going the same direction at the same time.

Tasks

Do the following tasks as well as you can, and answer the questionnaire afterwards. If you have any problems with performing the tasks, please write them in the comments section of the questionnaire. If your current location is outside of Trondheim, it will be sat to NTNU, for testing purposes. Your position will not be updated when ”driving”. We are not testing you, but the app, the goal is to test how easy it is to use the system, so you cannot do anything wrong.

- You want to go to Nidarosdomen. Book a car to take you there.
- You have an appointment at Munkegata 23. Book a car to take you there, but you do not want to be picked up from your current location. If you live in Trondheim, you want to be

picked up from your home. If not, you want to be picked up at Trondheim Sentralstasjon.

- You are going to Lerkendal Stadion, which you know has the address Klæbuveien 125. You are meeting some friends there, so you want to be dropped off at the east entrance of the stadium (to the right on the map). Book a car to take you there.
- Play around as much as you want to!

Well done! Now, please answer the questionnaire.

Appendix E

Questionnaire for user test 1

User test

Thank you for completing the tasks! Now you will answer some questions about the app you just tried and the whole product described earlier. This survey will take approximately 5 minutes to complete.

All questions are in English, but you may answer in Norwegian if you would like to. All answers are anonymous.

This first section asks some questions about your background.

* Required

1. **How old are you?** *
   Mark only one oval.
   - ◯ Under 20
   - ◯ 20-29
   - ◯ 30-39
   - ◯ 40-49
   - ◯ 50-59
   - ◯ 60-69
   - ◯ 70 or over

2. **What is your profession/field of study?** *

3. **Do you own a car?** *
   Mark only one oval.
   - ◯ Yes
   - ◯ No
   - ◯ I am planning to buy one in the future

4. **If yes, how often do you usually use your car?**
Mark only one oval.

- ◯ Several times each day
- ◯ Once or twice each day
- ◯ A few times each week
- ◯ A few times each month
- ◯ Almost never

Usability

In this section you will answer questions regarding how usable the app was. This is a standardized System Usability Scale (SUS), which will result in a score between 0 and 100, saying something about how usable the system is.

5. **I think that I would like to use this system frequently** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

6. **I found the system unnecessarily complex** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

7. **I thought the system was easy to use** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

8. **I think that I would need the support of a technical person to be able to use this system** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

9. **I found the various functions in this system were well integrated** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

10. **I thought there was too much inconsistency in this system** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

11. **I would imagine that most people would learn to use this system very quickly** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

12. **I found the system very cumbersome (tungvint) to use** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

13. **I felt very confident using the system** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

14. **I needed to learn a lot of things before I could get going with this system** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

About the product

In this section, you will answer a few questions regarding the product.

15. **I think this product will make my everyday life easier** *

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ I don't know

16. **I think I will sign up for such a product when it becomes available** *

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ I don't know

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

_____

_____

_____

_____

_____

18. **Are there any functionalities in the app that should be added or removed to make the app easier to use?**

_____

_____

_____

_____

_____

Comments

In this final section, you can leave your additional comments (if you have any). All feedback is very useful!

19. **Did you have any problems or difficulties when doing the tasks?**

20. **Do you have any additional comments?**

Appendix F

Questionnaire for user test 2

User test

Thank you for completing the tasks! Now you will answer some questions about the app you just tried and the whole product described earlier. This survey will take approximately 5 minutes to complete.

All questions are in English, but you may answer in Norwegian if you would like to. All answers are anonymous.

This first section asks some questions about your background.

* Required

1. **How old are you?** *
   Mark only one oval.
   - ◯ Under 20
   - ◯ 20-29
   - ◯ 30-39
   - ◯ 40-49
   - ◯ 50-59
   - ◯ 60-69
   - ◯ 70 or over

2. **What is your profession/field of study?** *

3. **Do you own a car?** *
   Mark only one oval.
   - ◯ Yes
   - ◯ No
   - ◯ I am planning to buy one in the future

4. **If yes, how often do you usually use your car?**
Mark only one oval.

- ◯ Several times each day
- ◯ Once or twice each day
- ◯ A few times each week
- ◯ A few times each month
- ◯ Almost never

Usability

In this section you will answer questions regarding how usable the app was. This is a standardized System Usability Scale (SUS), which will result in a score between 0 and 100, saying something about how usable the system is.

5. **I think that I would like to use this system frequently** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

6. **I found the system unnecessarily complex** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

7. **I thought the system was easy to use** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

8. **I think that I would need the support of a technical person to be able to use this system** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

9. **I found the various functions in this system were well integrated** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

10. **I thought there was too much inconsistency in this system** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

11. **I would imagine that most people would learn to use this system very quickly** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

12. **I found the system very cumbersome (tungvint) to use** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

13. **I felt very confident using the system** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

14. **I needed to learn a lot of things before I could get going with this system** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

About the product

In this section, you will answer a few questions regarding the product.

15. **This product will make my everyday life easier** *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

16. **I will sign up for such a product when it becomes available** *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

17. **This product will remove my need for a personal vehicle** *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

18. **My community will benefit from this product** *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

19. **Please list some reasons why you would or would not like to use this product** *

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

21. **Are there any functionalities in the app that should be added or removed to make the app easier to use?**

Comments

In this final section, you can leave your additional comments (if you have any). All feedback is very useful!

22. **Did you have any problems or difficulties when doing the tasks?**

23. **Do you have any additional comments?**

Appendix G

Demonstration video

A demonstration of the final mobile app is found here: https://youtu.be/uNe6r05ugMQ

Appendix H

Responses to user tests

H.1 User test 1

https://docs.google.com/spreadsheets/d/1VD79a6Lq_NkZ3-q4K5B-XAp5k854nuLJ2ywtuoF1eKQ/edit?usp=sharing

H.2 User test 2

https://docs.google.com/spreadsheets/d/15Zi4talLRgCvYs9m-BypJ0myZDvW7oar_hoAO4oFnhU/edit?usp=sharing