UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

How to use an app to Nudge people to choose more green transportation?

—

Ingvild Kristiane Myrvang

INF-3990 Master's Thesis in Computer Science - October 2021

Abstract

Making green transport choices are more important than ever to save the environment. To accomplish this there must be a reduction in emissions of greenhouse gases and other pollutants. How to make people choose green transportation, when it is inconvenient, is an interesting challenge. Most would rather take the easy way out to avoid the extra planning and estimation of how to reach their destination. Thus, they take the car, because they know how long it will take, no need for planning. What if instead there was an app that did the planning for you? Telling you when you had to leave to reach the bus, get on your bike or to be on time walking. What if it could tell you the weather too so that you would know what to take with you? What if you could tell the app that you need more or less time to walk to that bus stop based on the last time you walked there. This thesis will be about solving such problems through the design and the implementation of an app that will make travel planning easier.

Acknowledgments

A special thanks to Øystein S. Jacobsen for helping me with everything while working on this thesis.
I´m also thankful to thank Kristian Mikalsen and Sean Bakaitis for reading through my thesis
I would like to thank my supervisor Anders Andersen for introducing this project to me and his feedbacks on this thesis.
Lastly, thanks to student organization Imladris for being a place to discuss and be social and to all of my friends and family for their support along the way.

Table of Contents

Table of tables:

Table of figures:

1 Introduction

This thesis will be using nudge to make people choose greener transportation. But what is nudging? Nudge was first defined as [1] «any aspect of the choice architecture that alters people´s behavior in a predictable way without forbidding any options». This thesis will focus on the so-called green nudges, and these nudges [2] «aim at encouraging people to voluntarily contribute to a public good, namely, environmental protection».
Why is there a need to nudge people to choose green transportation? One reason is that [3] «main environmental issues in towns and cities are related to the predominance of oil as a transport fuel, which generates CO_2, air pollutant emissions and noise». CO_2 is one of the gasses which acts as a greenhouse gas in the atmosphere, making the earth warmer. This is known as the greenhouse effect, [4] «is a leading factor in keeping the Earth warm because it keeps some of the planet's heat that would otherwise escape from the atmosphere out to space». Of [5] «the most important greenhouse gasses in the atmosphere are water vapor H_2O, carbon dioxide CO_2, methane CH_4, nitrous oxide N_2O and ozone O_3». The problem is that people have been emitting CO_2 into the atmosphere and making the earth warmer than it should be by natural processes. This is known as global climate change and is shown by the climate change report [6] to be manmade. More CO_2 in the atmosphere will only make this process go faster, so one goal for humanity is to reduce the quantity of CO_2 in the atmosphere. This is where nudging may make it easier for people to choose green transportation.
Road dust is another pollutant that comes from the use of motorized road vehicles. These vehicles typically have spiked winter tires, resulting in the production of road dust when the roads are bare. This road dust can trouble people suffering from asthma, they might have trouble with breathing while there is lot of road dust in the air, and they have use asthma medicine to breath properly. Nudging people to choose green transportation instead of personal cars can therefore also help to reduce the amount of road dust.
Nudging can be both for digital and nondigital, where the difference between them is that digital will do its nudging through digital environment and non-digital will not do its nudging through digital environment. This app is going to be using digital nudges. According to

Weinmann, Schneider and Brocke [7] «Digital nudging is the use of interface design elements to guide people´s behavior in digital choice environments».

1.1 Problem definition

With all the pollution that comes from cars there is a need to move towards more environmentally friendly transportation, like walking, biking, and using public transportation. The problem is that it is difficult to incorporate these greener transportation options in everyday travel, when one has to do an extra effort to plan around these environmentally friendly transportations.

This thesis will with the use of digital nudging try to solve this problem by making an app that will try to influence people to choose green transportation.

This app will be the main focus in this thesis and will be explored how it was designed, implemented, tested and at the end if it was successful in influencing people to make greener transportation choices.

1.2 Goal

To solve the problem the goal is to develop an app that can be used for travel planning and nudge its users towards green transportation.

Therefore, the first goal is:

The app will need to be able to let users select an origin and a destination for their travel. The origin and destination should be possible to identify by having the user either writing in the address or choosing a location on a map. In its first version the app will be limited to the city Tromsø. The map will have a possibility to see establishment as well as the addresses for locations when zooming in. To help the users when writing in addresses or location an autocomplete feature will be implemented. The autocomplete should come with suggestions based on the first letter the user writes and update for each successive one.

The second goal is:

To implement a way to find the users location so the app can easily find the starting point. The user's current location will have a marker that displays the address of the location.

The third goal is:

> *Offer a way for the user to select their preferred transportation mode. A direction feature must take all the inputs into consideration and show possible green travel routes. This feature will use the information to calculate the distance and duration of the travel routes. It will also need real-time updates while traveling to show the remaining time, and distance and offer new possible routes from the current location.*

The fourth goal is:

> *There will be a possible to click on an establishment to get more information about it. This will tell the user some things about the establishment that will make them use the app. Since they can see name, rating and if it is open now and this will give them enough information to see if they want to go there and then they can choose it as a place to travel to. Also, there will be a possibility to get personalized walking/biking time.*

The fifth goal is:

> *To have a possible integration with calendars, such as Apple´s Calendar, might also be implemented with a notification feature so the users can easily get to their appointments.*

The sixth goal is

> *The app will also have the possibility to use the data from the GPS trackers in the buses in Tromsø. This will update the travel information about where the buses are at the moment and the user can see this information and plan around it.*

The seventh goal is:

> *To integrate some weather data (i.e., rain, wind, sun, and snow) such that users may be notified of weather conditions. This integration is important since it will make it possible to show skiing as a green travel option during winter conditions.*

1.3 Approach

This thesis is about making an app that will nudge people into choosing green transportation. Therefor the approach will be how this app was designed and implemented.

This app was implemented with the programming language Swift and the IDE Xcode for iOS phones. Swift was chosen as the language to program this app in, and iOS was chosen since Swift is made by Apple and there are more of documentation for Swift with iOS. Xcode is made with iOS in mind and is developed by Apple. More information about Swift and Xcode will be presented in -Section 2.2 and -Section 2.1.1. Xcode was used in designing the app, and Swift with Xcode were used to do the necessary implementation to the design. The app´s design and implementation will be presented in -Section 3 and 4.

1.4 Method

The starting point for the method for this thesis is to first understand nudging and everything that comes with this concept. Then the next step is to design the app and then implementing this design so that the app can influence people towards the choice of green transportation. Finally test the app with the author doing some test to see if it works and then with some other testers to find out if this app was successful in nudging people to choose greener transportation. How these tests were done will be presented in -Section 5.1 and the result of the test will be presented in -Section 5.2.

1.5 Contribution

Despite not having all features defined in -Section 1.2, the app forms an important foundation that others can build on in the future. This foundation contains a method of acquiring the address that the user wants to travel from and to, using autocomplete and a map. The user´s preferred travel mode is also taken into consideration. This information is used to create possible routes and calculate the distance and duration of each route. Users may also hit a button to update their location while traveling, triggering route recalculations and updating distance and duration estimates. When the user clicks on a business, they receive information such as the name, rating about that business and if it is open right now. There will be a way for the user to have their preferred travel mode stored for the next time. The user may additionally put in the time they used on the route, and this will show up the next time they travel this route. This app is able to influence people to some extend towards the choice of greener transportation choice, where this choice of transportations will be walking and biking.

In the future someone can implement rest of the features in goal and make the app better than this protype.

1.6 Outline

This thesis is structured as followed:
2 – Technical Background: Presenting nudge, some programming terms, the programming language Swift, some tools used with Swift like IDE, CocoaPods, Alamofire, SwiftyJSON and Googles APIs.
3 – Design: Explains the way the app was designed but does not go into how this was done but explains what each controller does.
4 – Implementation: Explains how things were implemented into the app. Start first with how things were implemented into in the storyboard in Xcode. Then goes on to explain how ViewController, MapsViewController and RoutesViewController were implemented. This section will also take up thing as programming language, libraries, problems, the way errors was handled and the choice of using Google Maps.
5 – Discussion: Here different subjects will be discussed, these subjects will be what was achieved of the goal, learned from them and shown from the author in these goals. Then how the app was tested, the result from the testing the app, why Tromsø, the choice of colors in the app, rejection on nudges on the bases of freedom of choice, the ethical dilemma of nudges, if there are any other concerns of nudges, why it was developed in iOS and if the app was successful in nudging.
6 – Future Work: Here all that was not implemented in goals will be talked about for implementation in the future and with the result from discussion
7 – Conclusion: Here the conclusion of this thesis will be talked about, and if the app was able to nudge user to take green transportation.

2 Technical background

2.1 Nudge

Nudging is about pushing someone in a direction that is a better choice for society, without forbidding any options [1]. It all comes down to how the choices are presented to people. People will usually choose the option that is easier or takes less effort on their part. If the choice requires less mental effort but still some physical effort it will feel easier for people to choose it. For example, a sport watch that tells you that you need to move, or that you have been inactive for too long. This nudge could make you walk around more and get some exercise in your day.
The one creating the nudge is a choice architect, Thaler and Sunstein state [1] «that a choice architect has the responsibility for organize the context in which people make decisions». Thaler and Sunstein go on to say that there [1] «are many parallels between choice architecture and more traditional form of architecture and that there is no neutral design». There will always be something of that person in their design. One can have several biases when designing things, but one can try to add less of the biases and themself into a design. One can try to think what other people will think about the design and how this will affect them. They go on to say [1] «that everything matters when designing the nudge», which is the case for this app, especially to color and this will be discussed in -Section 5.4.
Nudging as a concept is called libertarian paternalism, libertarian part of the nudge is that people need to be able to choose for themselves and to not be part of the nudge if they want to [1]. The paternalistic part of the nudge is that it will try to make people lives longer, healthier and better [1].
Libertarian is all about the freedom of choosing but can there be an illusion of freedom of choice. This will be discussed in -Section 5.5.2, but what is freedom? Freedom is defined in Cambridge dictionary as [8] «the condition or right of being able or allowed to do, say, think, write, etc. whatever you want to, without being controlled or limited». With freedom comes the right to freedom of speech, freedom to choose, freedom of information, freedom to life, the freedom to be free,

etc. Freedom itself as a concept has been limited by human rights and laws, to ensure that everybody´s freedom is upheld equally. This is something that also apply to freedom of choice, one person is not free to do whatever choice they like if it restricts someone else freedom in any way.
Nudging is usually used for good things like making people be healthier, using less energy on heating and power, choosing green transportation, or saving up to pensions. But this might not always be the case which is why there is the discussion of the ethics of nudging, this will be discussed later in -Section 5.5.2.
The form for nudging utilized by this app is digital nudging, [9] «digital nudging is also about selecting and combining the right set of information in the given context, so people have better and more relevant information to base their choices on». Weinmann, Schneider and Brock say [7] that this type of nudging is mostly used in digital environment but it can be applied beyond this and that digital nudges being increasingly used to influence real world behavior. There are apps that use nudges, these are mostly for health and are called (for example mHealth, (Mobile Health) apps). Not all of these apps use nudge but some of them do. These [10] «applications have the potential to assist patients in adhering to their physician's advice in chronic disease management through the use of persuasive nudge».
In digital nudging [11] to design choices to nudge users the nudge will need to define a goal, understand the users, and design and test the nudge. For apps that use digital nudging ease of use is important, as this will help in nudging the users. Nudging is about making choices easier; an app must portray this in its design. There are a lot of design decisions a choice architect will need to think about in design the app that will nudge the users. For example, where to place graphical widgets and their sizes. The size of the users´ device play an important role in such decisions, here it is very important to think about the size of phones. The most important thing in design will be how the users are going to interact with the app.
Apps as digital nudges will focus more on the design of that app, since it needs to be visually pleasing, not boring, and at the same time not distracting. Apps will often collect information about the user and create a tailored profile for them so they can be used better. Apps that do this to nudge are therefore utilizing smart nudging. Smart nudging is defined as [12] «digital nudging, where the guidance of user behavior is tailored to be relevant to the current situation of each individual user».

2.2 Swift

Swift is a new language based on Objective-C, produced by Apple Inc. [13] «Swift is a general-purpose programming language built using a modern approach to safety, performance, and software design patterns».
Stated on Apple´s website [13] Swift is built to make programming easier both in reading and writing the code. On memory Swift [13] «managed it automatically and there is no need for semicolons» unlike in C. Swift [13] «also borrows from other languages, for instance named parameters brought forward from Objective-C are expressed in a clean syntax that makes APIs in Swift easy to read and maintain». Regarding the safety of Swift, it is stated on Apple´s website that [13] «swift was designed from the outset to be safer than C-based languages and eliminates entire classes of unsafe code». On Apple´s website it is also stated that [13] «variables are always initialized before use, arrays and integers are checked for overflow». Also, [13] «syntax is tuned to make it easy to define your intent — for example, simple three-character keywords define a variable (var) or constant (let)». Platforms that swift can be used on [13] «open-source Swift can be used on the Mac to target all of the Apple platforms: iOS, macOS, watchOS, and tvOS». They also inform on their website [13] that they also have Swift for some Linux operative systems, and they have also Swift for Windows 10. To get swift for any Mac OS one only need to install Xcode. For Linux and windows 10 it is information about how to install Swift and what else is need for each of these OS´s in [14].

2.1.1 Xcode

Xcode is an IDE (integrated development environment) that is developed by Apple and that is used by programming language Swift to develop apps. Integrated development environment is [15] «a program that is used with a programming language, that helps with debugging, testing and running the code». This program [15] «will also suggest things while one is writing the code and telling you when you have written something wrong». It will also [15] «give you more tools for helping with developing». Xcode is [16] «integrated with the Cocoa and Cocoa Touch frameworks, Xcode is an incredibly productive environment for building apps for Mac, iPhone, iPad, Apple Watch, and Apple TV».

2.3 API

Application Programming Interfaces (API) will be used in implementation of the features to Google that will be used in this app, like map, places, geocoding, directions, distance and duration. There is only one way to implement Google´s features and that is with using API´s. But what is API? API (Application Programming Interface) [17] «defines how an outside programmer can add functionality or services to an application or other type of software created by others, so that the coder does not have to familiarize themselves with the application's source code».

2.4 HTTP and HTTPS

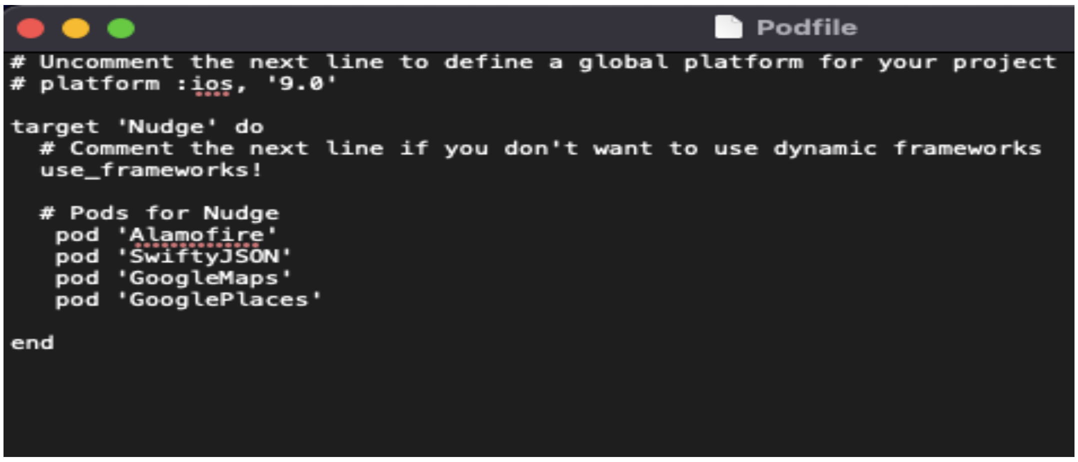
HTTP (Hypertext Transfer Protocol) [18] «is a communication protocol that is primarily used to transfer HTML documents between servers and clients using a transport protocol, usually the Transfer Control Protocol (TCP)». HTTPS is a secure version of HTTP that [19] «uses HTTP in a combination with Transport Layer Security (TLS) or Secure Sockets Layer (SSL), which encrypts the communication session so that an unauthorized person cannot eavesdrop or change data during transmissions over open networks». HTTPS will be used to access the data provided by two of Googles APIs; the Directions API and Distance Matrix API. Here there where two choices in implementation and that was either using HTTP or HTTPS to access the URL. Here HTTPS was chosen because of its safety.

2.5 CocoaPods

CocoaPods is a library (Gem) in Ruby the programming language and it is used to [20] «manages library dependencies for your Xcode projects». On the CocoaPods website it is informed that [20] «the dependencies for your projects are specified in a single text file called a Podfile». CocoaPods will resolve dependencies between libraries, fetch the resulting source code, and then link it together in an Xcode workspace to build the project. To install CocoaPods on MacOS one has to use the terminal. The easiest way to install it is to use the following command in the terminal:

```
sudo gem install cocoapods
```

This will also install a default version of Ruby on MacOS as well as CocoaPods. This is because libraries in the programming language are dependent on the programming language and won´t work without it. On CocoaPods official website states detail of other ways of installing CocoaPods if needed [20]. To use CocoaPods one will first need a Podfile, [21] «the Podfile is a specification that describes the dependencies of the targets of one or more Xcode projects». The file should simply be named Podfile and one can use the terminal to create it, simply navigate to the folder one would want this Podfile in and then write the command `pod init´ in the terminal. Pods themselves will be the outside library that needs to be install into a Xcode project. All the required pods will be written into the Podfile, as displayed in Figure 1. To install the pods in the Podfile go to the folder where the podfile is located at and then write in the command `pod install`.



```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'Nudge' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for Nudge
   pod 'Alamofire'
   pod 'SwiftyJSON'
   pod 'GoogleMaps'
   pod 'GooglePlaces'

end
```

Figure 1 How the Podfile looks for this project.

CocoaPods was chosen because it was the easiest option and there was more information about how to use it with Swift and Google.

2.5.1 Alamofire

Alamofire is one of the many libraries that is available for installation as a pod through CocoaPods. Alamofire is created by the Alamofire Software Foundation [22]. Alamofire itself can act as a library in a Xcode project [23] and «provides an elegant and composable interface to HTTP network requests». [23] «It does not implement its own HTTP networking functionality». Alamofire [23] «builds on top of Apple's URL Loading System provided by the Foundation framework». Alamofire itself can be installed without CocoaPods, on Alamofire´s official website [24] it is written that it can be installed with Carthage and manually if one doesn´t want to or can´t use CocoaPods. The Alamofire library is going to be used in this app to easier implement the features for this app. This will be because it does the work that is needed to access the HTTPS request to the Directions API and the Distance Matrix API.

2.5.2 SwiftyJSON

In Swift dealing with JavaScript Object Notation (JSON) can be difficult and messy. SwiftyJSON is a library that [25] «makes it easier and less messy to deal with JSON». JSON [26] «is a text format and it is based on the programming language JavaScript but is completely independent». To install SwiftyJSON you can either do it using CocoaPods (by adding it as a pod into the Podfile), using Carthage or by doing it manually. SwiftyJSON is going to be used in this app with JSON to make it easier to work with JSON.

2.6 Google Maps

The app require a map that visualize directions, which is offered by Google at a cost which is reasonable. There will a discussion of why Google was chosen to use as the map in Section 4.9, since there is a choice between Google and Apples own map. The author does not know if there is any other than these two that can be used as maps in implementation of apps. Google was chosen as the map to use for implementing this app and the features that was used in this app will be talked about below when talking about Google. Not all Google´s features where required and these will not be talked about, and this is because they were not necessary for the app to nudge people to choose green transportation. The only one not used here that can be investigated in the future when someone continue the work on this app is Maps Elevation API.
Google offers many APIs on their cloud platform, which may be used with their Maps SDK for iOS API. Not all of these APIs are required to use the Maps SDK and some of them can even be used without it. [27] «With the Maps SDK for iOS, you can add

maps based on Google maps data to your application». The SDK [27] «automatically handles access to the Google Maps servers, map display, and responds to user gestures such as clicks and drags». With the SDK [27] «you can also add markers, polylines, ground overlays and info windows to your map». These objects provide additional information for map locations and allow user interaction with the map. To get the Google Maps library into a Xcode project one can either do it through CocoaPods as a pod in the Podfile or through Carthage or install it manually and add it to the project.
Google Places were also used as an API in this app, this is one of the APIs that can be used without the Maps SDK. [28] «The Places SDK for iOS allows you to build location-aware apps that respond contextually to the local businesses and other places near the device». The Places SDK is called Places API in Google Cloud Platform and with it [28] «you can build rich apps based on places that mean something to the user». With Google Place one can have the autocomplete feature when searching for places and addresses. To get Google Place library into the Xcode project one can use the same methods as for Google Maps SDK.

Other APIs that were used in this app are Directions API, Distance Matrix API and Geocoding API. The Geocoding API can either geocode and address into a location or reverse geocode a location into an address. It provides a direct way to access these services via an HTTP request [29] «that is returned in JSON or XML format». The Directions API [30] «is a web service that uses an HTTP request to return JSON or XML-formatted directions between locations». With this [30] «you can receive directions for several modes of transportation, such as transit, driving, walking, or cycling». You can use this API [30] «for direction calculations that respond in real time to user input (for example, within a user interface element) ». The Distance Matrix API [31] «is a service that provides travel distance and time for a matrix of origins and destinations». The API returns information based on the recommended route between start and end points, as calculated by the Google Maps API, and consists of rows containing duration and distance values for each pair. This API like the geocoding API and directions API also uses an HTTP request and is return in JSON or XML format. These three APIs use HTTPS for security, but it is also possible to use HTTP. Here the output for both Directions API and Distance Matric API were chosen to be JSON, this is because it is slighter easy to use in Swift than

XML and used more be other people. Also, there is an outside library for parsing JSON in Swift that make JSON even easier to use, this being SwiftyJSON.

3 Design

This app was designed and made in a particular way to nudge the user toward the choice of green transportation. This design and making of this app will now be talked about.
The author chose in to make the apps in four different views that the user interacts with, this is because the size of phone screens. Users will need to see everything in this app properly and this is the reason for dividing things in this app out in more than one view. In one view the user can click buttons and textfields that will send them to other views, in another view the user can write in an address or place and in the other two views the user can view the map. One allows the user to choose an address or place from the map, and the other will have a map with the directions for the planed route. These views will be named view, autocomplete, map, and routes´ view
The app will start in the view that look like this Figure 2, the user will first need to give access to the app to send notification, give access to the data in the calendar and the access to the location of the user. The only thing that was design with this access notifications are the text that show up, the rest of it is a default design from Apple. This way the user know they are giving access to the app. To see how this access notifications looks like see Figure 3.

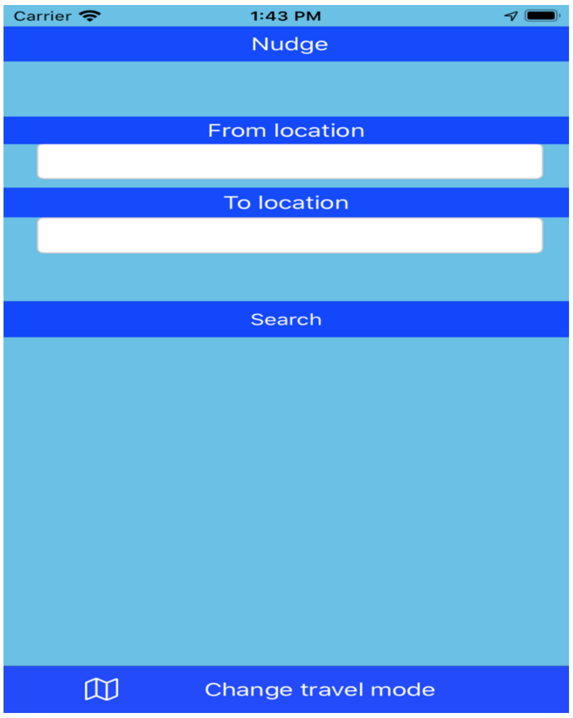
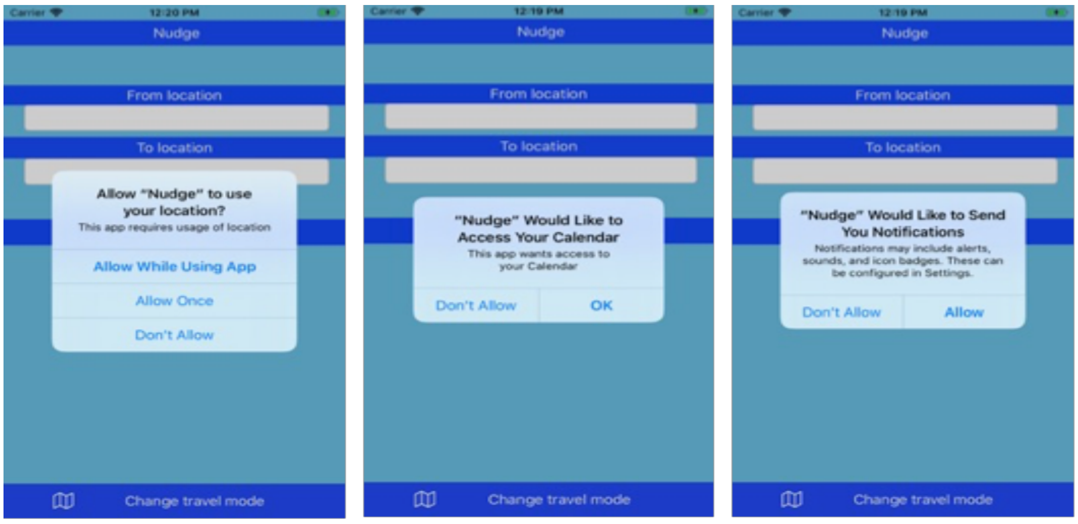


Figure 2 How the view the app will start in looks like



Allow "Nudge" to use your location?
This app requires usage of location

"Nudge" Would Like to Access Your Calendar
This app wants access to your Calendar

"Nudge" Would Like to Send You Notifications
Notifications may include alerts, sounds, and icon badges. These can be configured in Settings.

Figure 3 How these access notifications looks like

The view consists of a label, that is on the top of the screen, this is the name of the app. Then Two other labels that will tell the user which textfield that is from and to. Then two textfield that will be empty until the user chose and address from the map or autocomplete. When the user clicks the textfield they will be sent to autocomplete, see Figure 5. The labels will be near the textfields, so the user understand these are what the app refers as from and to. Then a button called search so that the user understand they will now search for a route between from and to. This button will either send the user to routes or show an alert that tells the user that both textfields need addresses in them. This button will only be a little down from textfields, so the user understands the button is connected to the textfields. To see how this alert will look like in the app see, Figure 6. To see what happens when both textfields have addresses and the search button gets clicked see Figure 7. Then at the bottom the button for the map will be, this will be a picture of a map, so the user understands what this button is for. When this button is clicked, the user will first need to choose from or to, so that this address can be put back in the right textfield. To see how this looks like in the app for the user see Figure 8. Then after the user choose what they want the user will be taken to the map. Then the last button will let the user choose their preferred transportation. To see how this look like see Figure 9. To see a description with arrows to all labels, textfields and button look at Figure 4.

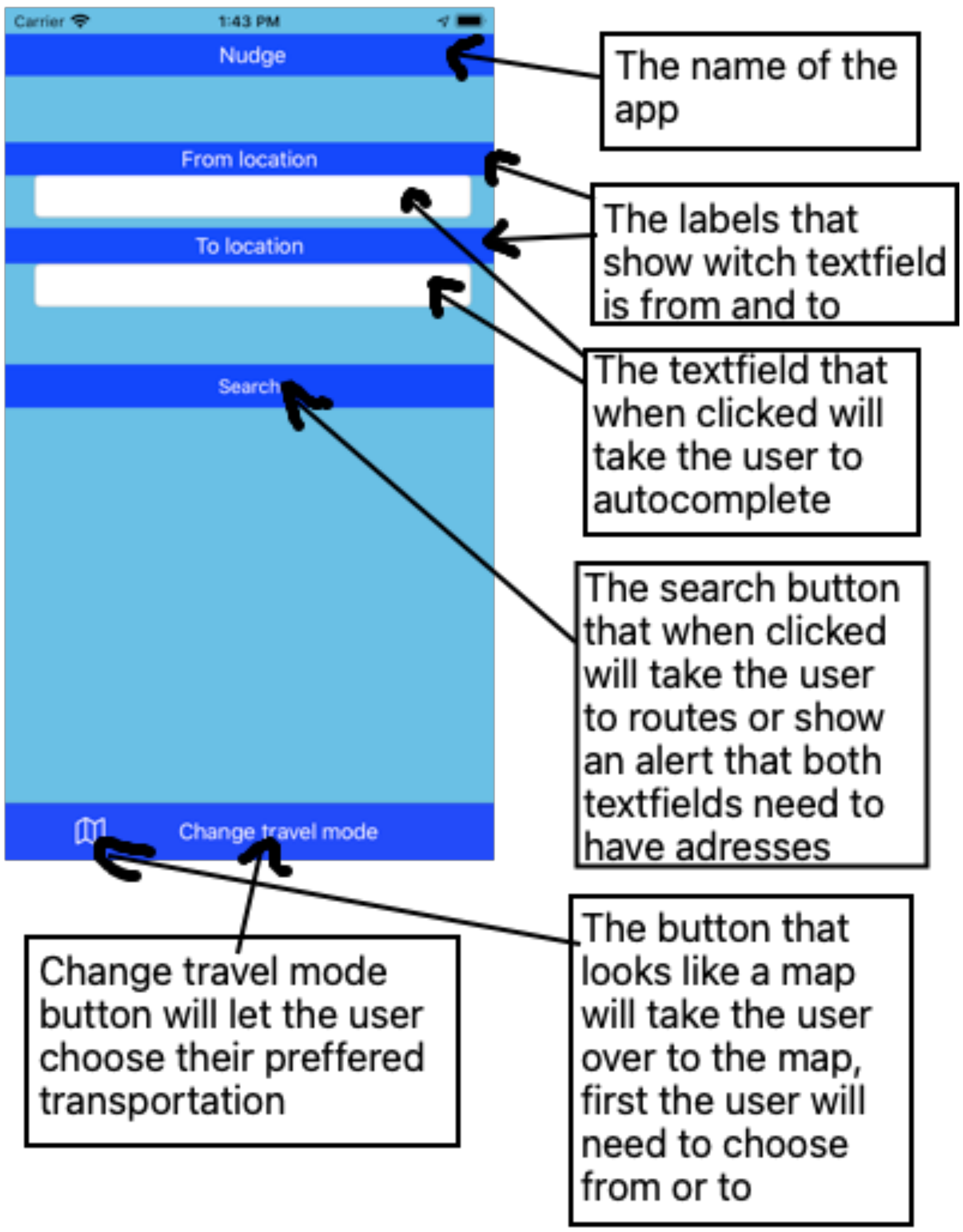

Carrier
1:43 PM
Nudge
From location
To location
Search
Change travel mode
The name of the app
The labels that show witch textfield is from and to
The textfield that when clicked will take the user to autocomplete
The search button that when clicked will take the user to routes or show an alert that both textfields need to have adresses
Change travel mode button will let the user choose their preffered transportation
The button that looks like a map will take the user over to the map, first the user will need to choose from or to

Figure 4 All the labels, textfields and buttons with arrows and descriptions

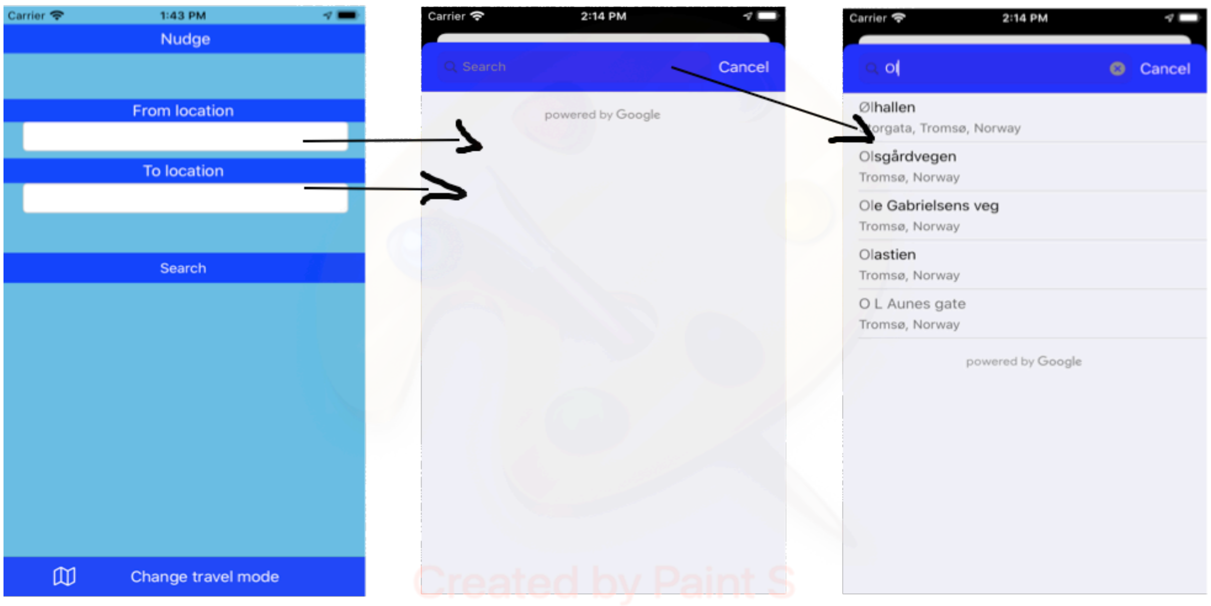
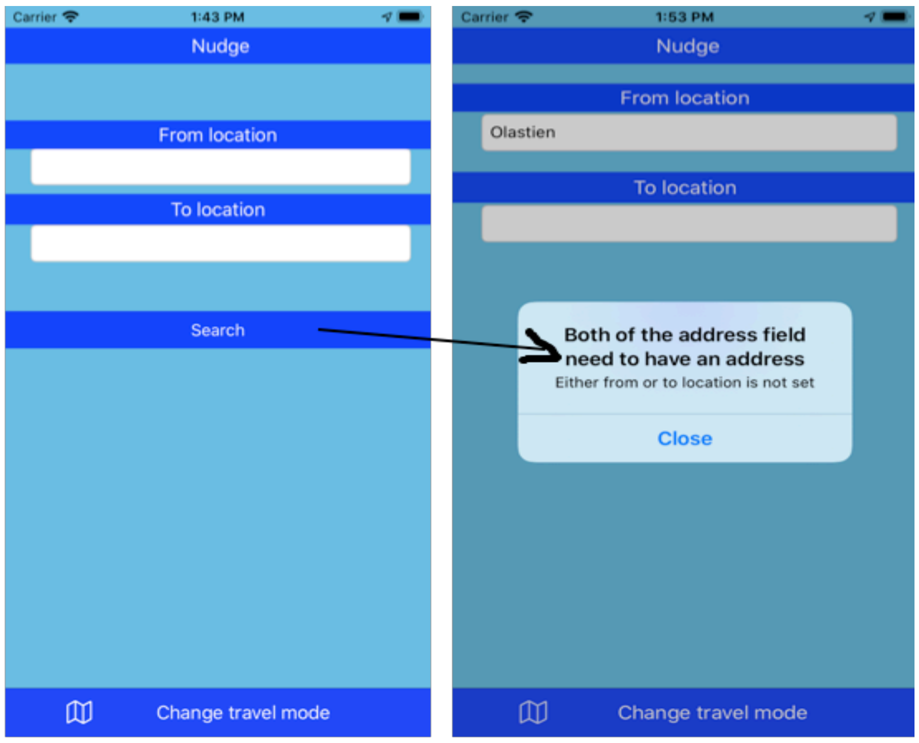


Figure 5 This picture will show what happened when the user click on of the textfields. The user will be taken to the autocomplete this transition is show with arrows from the textfield to the autocomplete. It also shows how it looks like when the user writes in a letter, the autocomplete will give suggestion from the first letter.



Both of the address field need to have an address

Either from or to location is not set

Figure 6 What the alert the user will be meet with if they click on the search button with either none or only one textfield have an address.

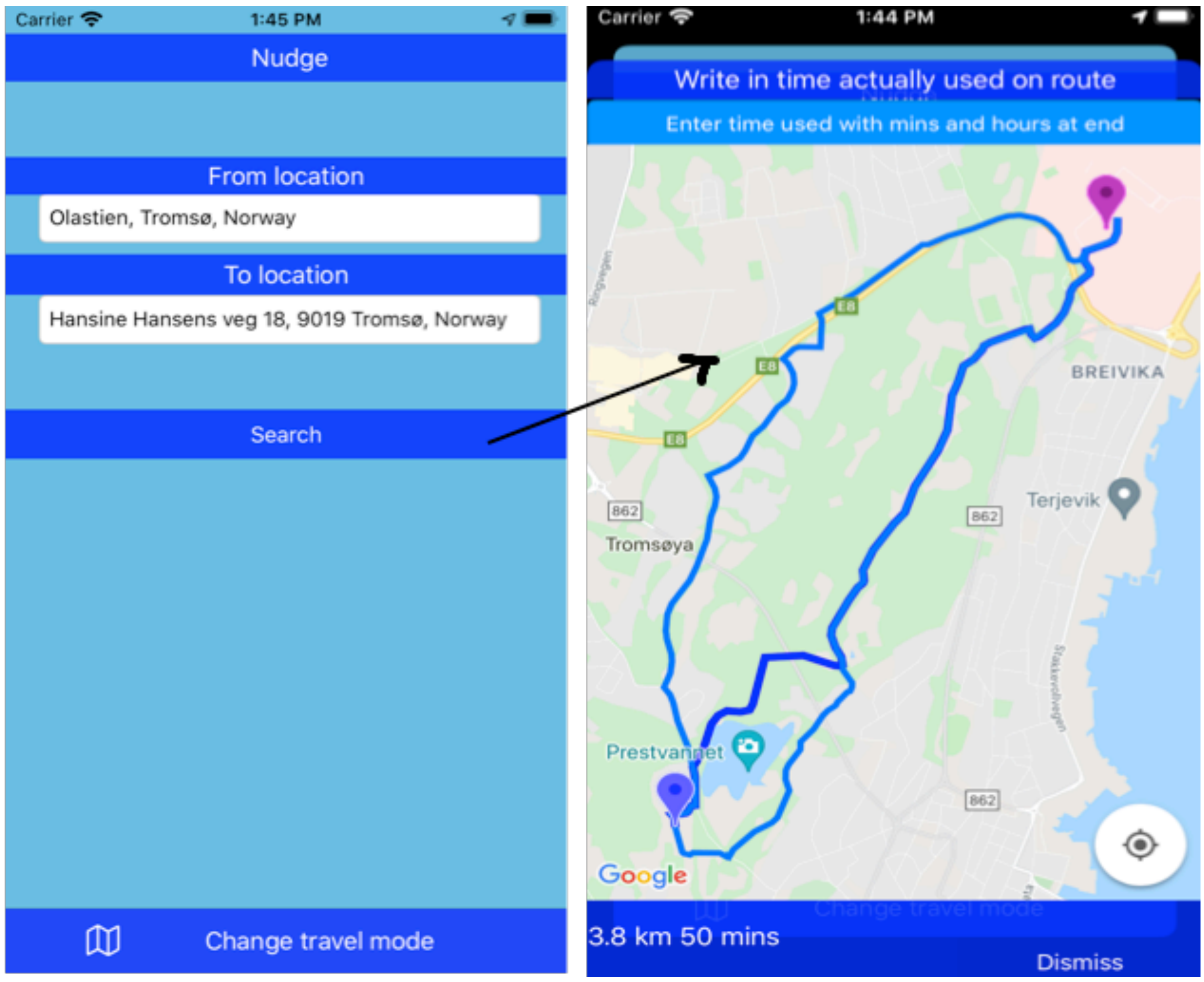


Figure 7 When both the textfields have addresses and the user click the search button they will be taken over to routes, this transition is shown with an arrow.

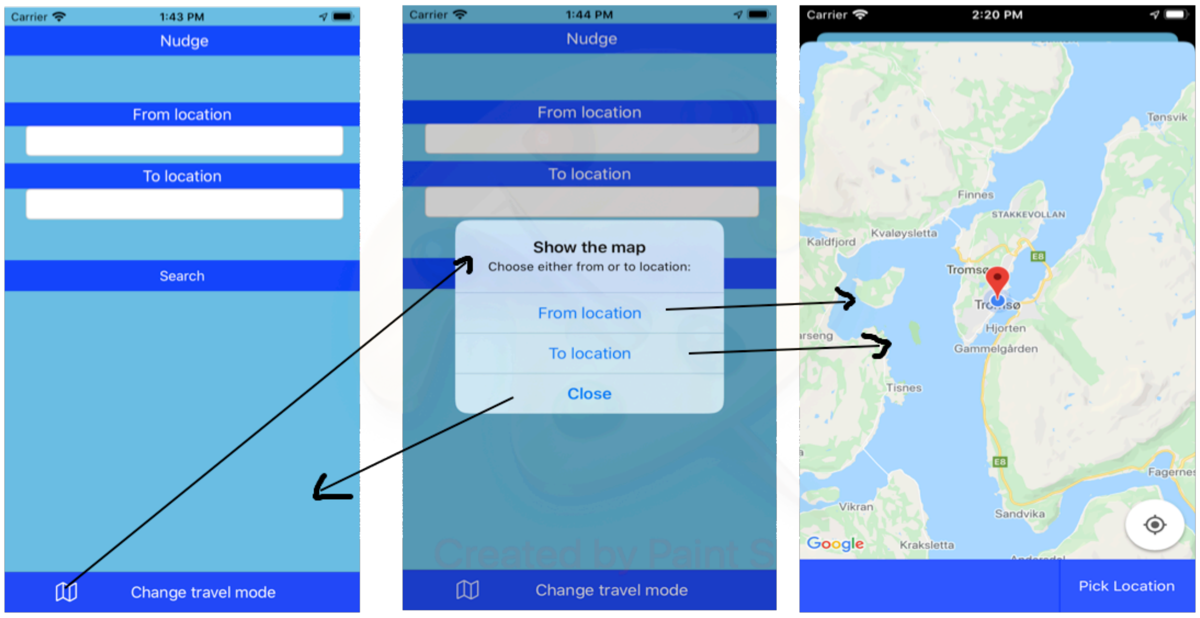


Figure 8 When the map button gets clicked the user will get up a white notification box they can choose from location, to location or close. When the user clicks the from or to, they will be taken to the map this transition is shown with arrows. If the user clicks on close they will be taken back to view and this transition is shown with an arrow.

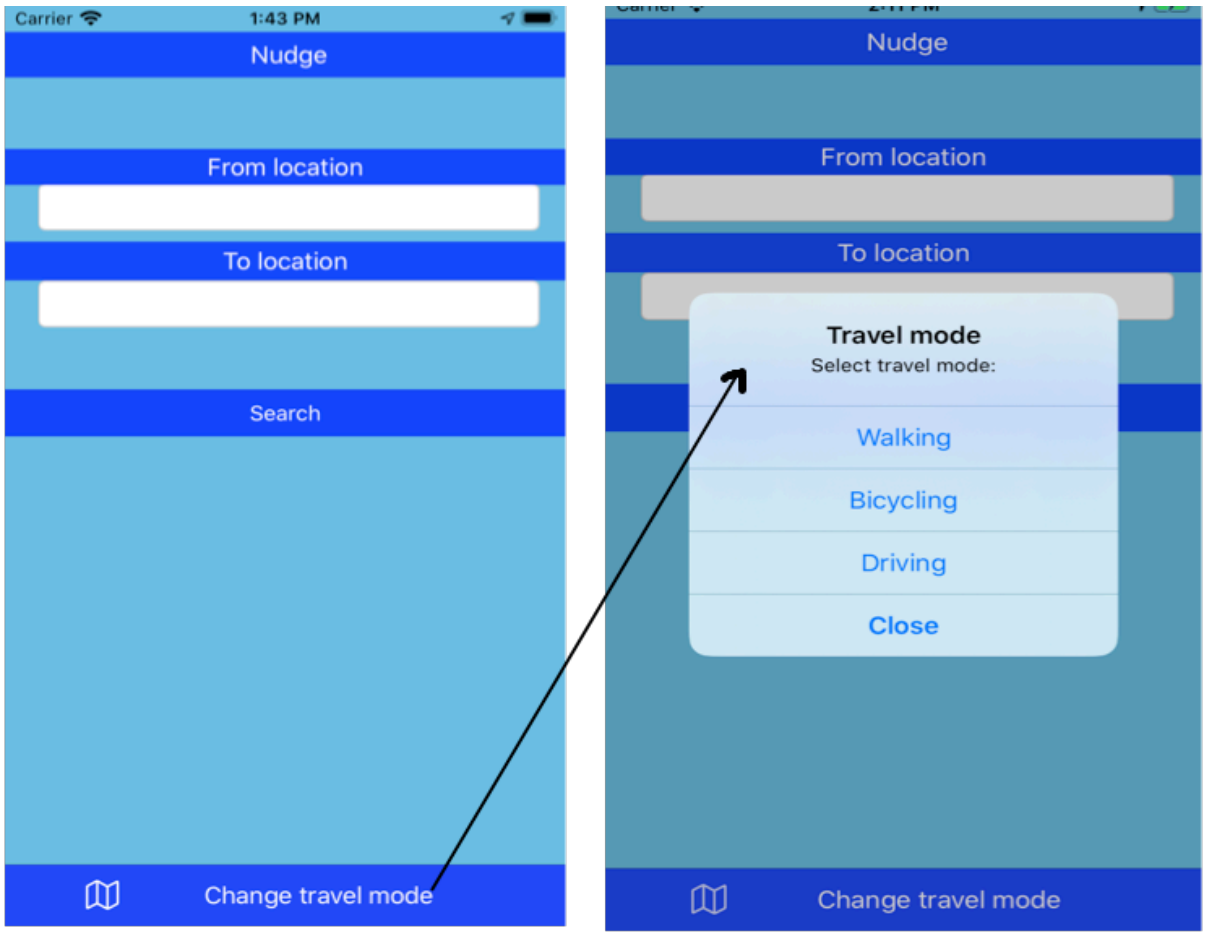


Figure 9 when the user click on the change travel mode they will be meet with a white notification box where they can click on the transportation they want to utilize. They can choose between walking, bicycling, and driving and when they choose something the app will take them back to view. They can also close the notification box by clicking on close if they want to choose none.

The Autocomplete view will have a search field that the user can write in and give the user suggestion on the first letter typed and then when the user chose an address or place it will go back to the view and the address will be in the textfield that the user clicked on to be sent to autocomplete view. See how this autocomplete looks like by seeing Figure 5, it will show also how it look like when the user writes in a letter and that the autocomplete comes with suggestions.

The map view will have a map with a marker on it, this marker will show the address to where it is localized on the map. When the user clicks on the map the marker will move to where the user clicked. A button that when clicked will find user location and the map will now be localized to where the user is. Another button that when clicked will pick the location that the marker is on and go back to view and the address to the marker will now be in the corresponding textfield. To see how this look like in the app see Figure 10. A label that will give information about business when they get clicked on the map by the user. To see how this looks like in the app see Figure 11.

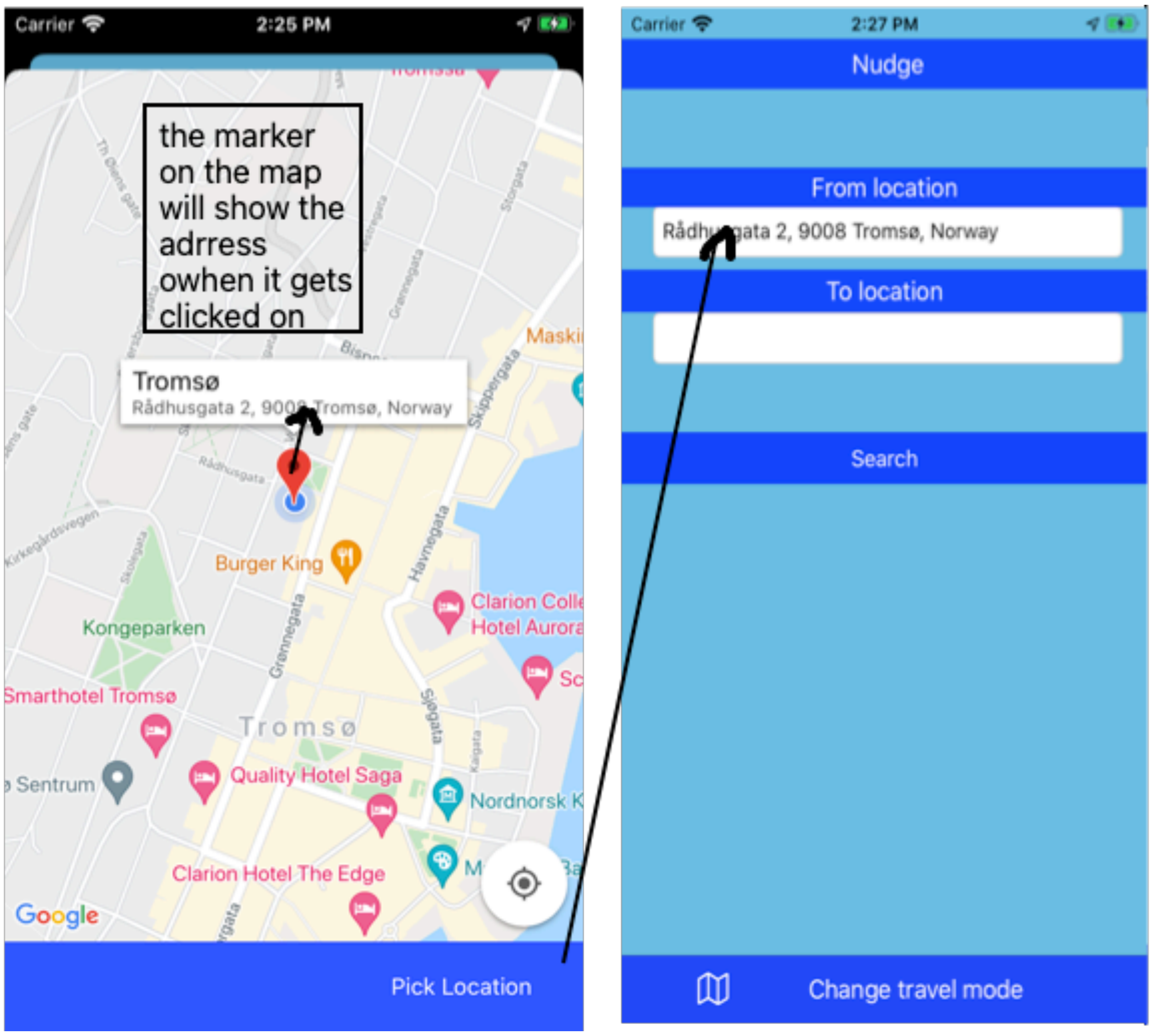


Figure 10 The map with a marker on it, the marker will in the start be localized to the user current location. When the marker gets clicked on it will show the address to where it is localized. Then when the user clicks pick location, the user will be taken back to view and the address to where the marker where will now be in the textfield. All transition is shown with arrows.

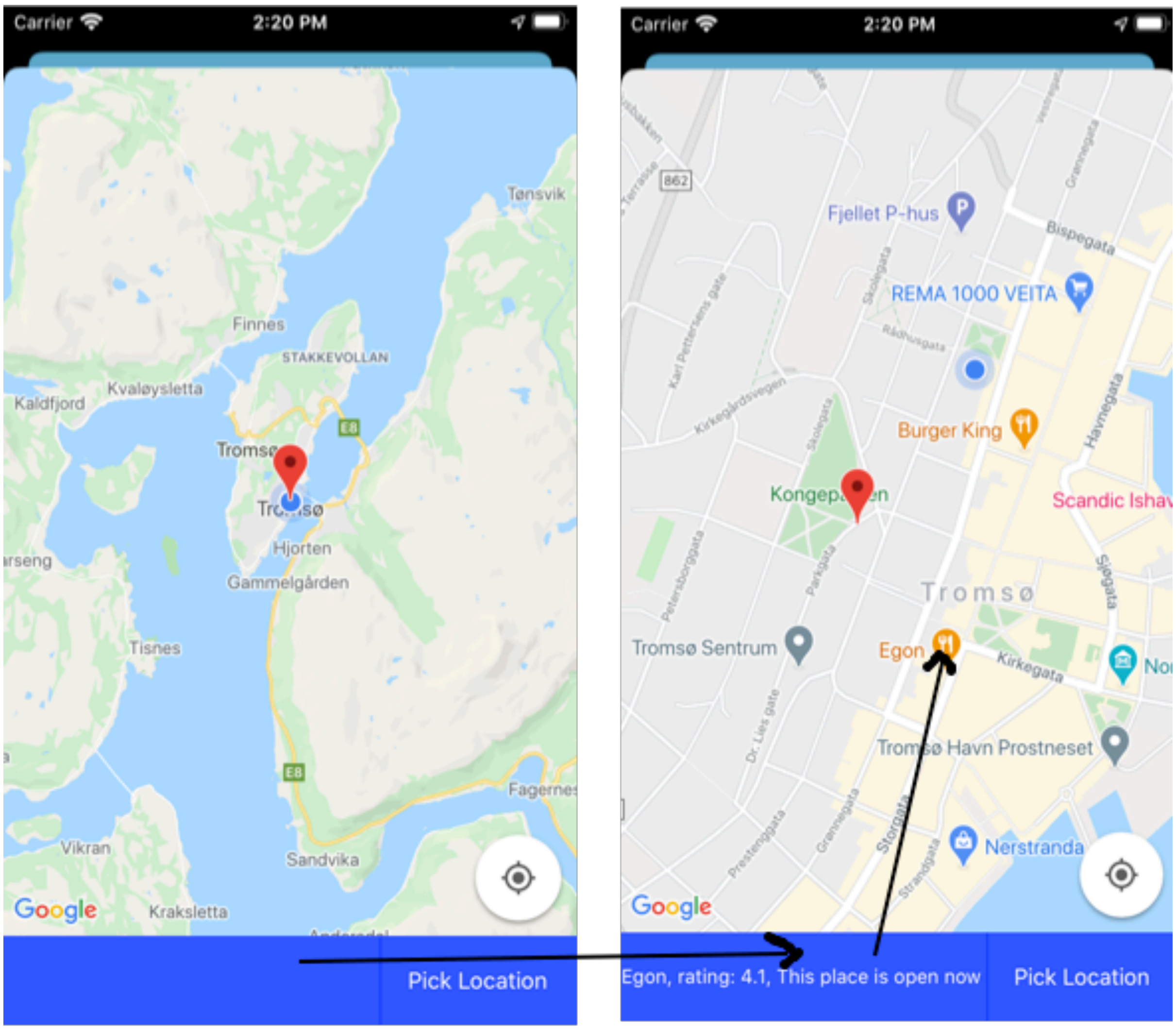


Figure 11 When a business gets clicked on the label will get a text with the information about the business like name, rating and if it is open now.

The routes' view will have a map with a drawn route with the input that the user gave. It will have two markers that will be the address that was put in the textfields. It will have a label that tells the user what the textfield is for. It will have a textfield that when clicked on the user can write in the time they used on the planed route. When the user clicks on this textfield, it will open a keyboard that the user can use to write the time they used, to see how this look like see Figure 13. It will have a button that when clicked on will localize the user on their current location. It will have a label that tells the user the distance and duration of the route. It will have a button dismiss that when clicked will go back to the view so the user can plan a new route if needed. To see a description of the labels, textfield, map and the buttons see Figure 12.

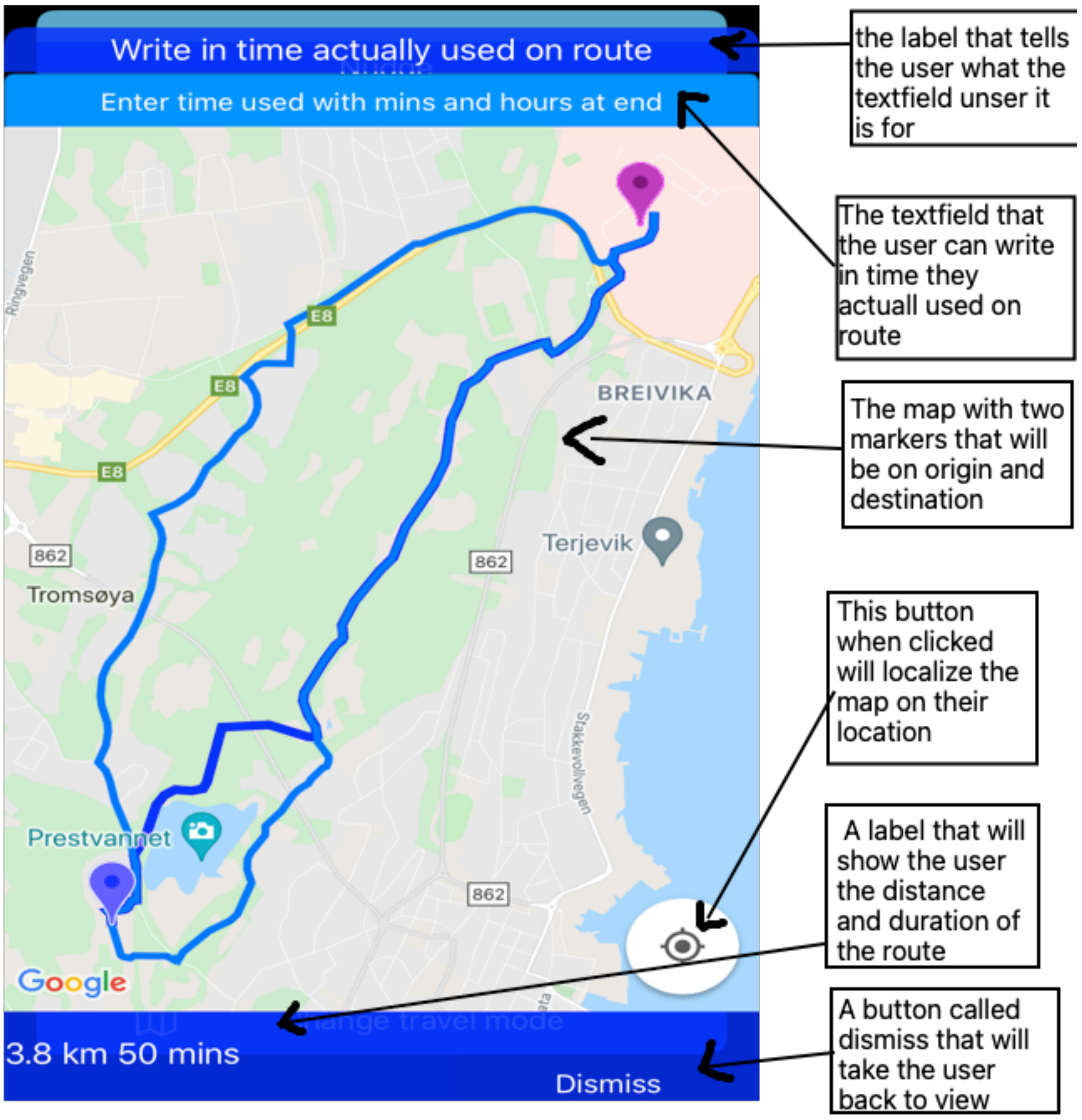

Write in time actually used on route
Enter time used with mins and hours at end
E8
E8
E8
BREIVIKA
862
Tromsøya
Terjevik
862
Prestvannet
862
Google
3.8 km 50 mins
Dismiss
the label that tells the user what the textfield unser it is for
The textfield that the user can write in time they actuall used on route
The map with two markers that will be on origin and destination
This button when clicked will localize the map on their location
A label that will show the user the distance and duration of the route
A button called dismiss that will take the user back to view

Figure 12 This picture show the route view with a description about all the labels, textfield, and buttons do and arrows to all of them.

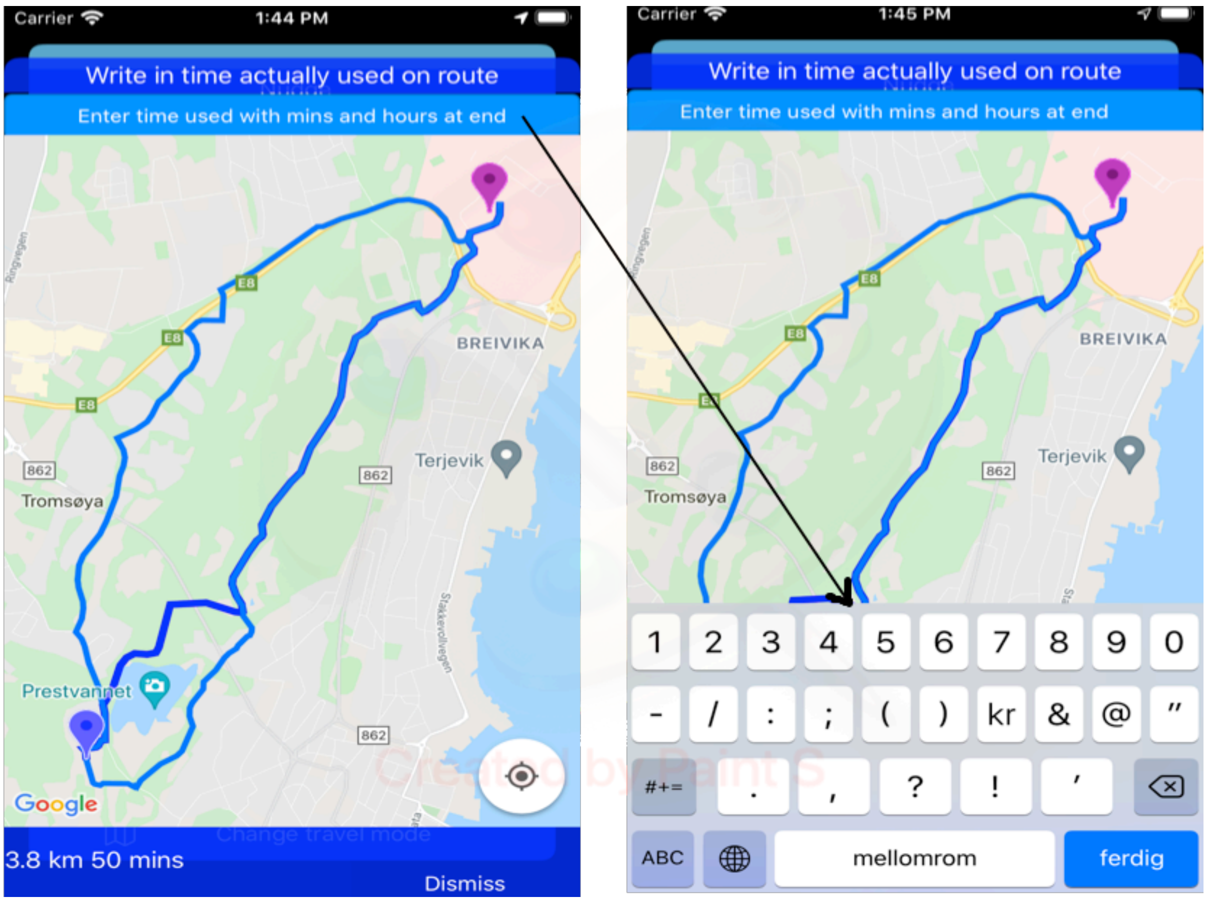

Carrier
1:44 PM
Write in time actually used on route
Enter time used with mins and hours at end
BREIVIKA
Terjevik
Tromsøya
Prestvannet
Google
3.8 km 50 mins
Change travel mode
Dismiss
Carrier
1:45 PM
Write in time actually used on route
Enter time used with mins and hours at end
BREIVIKA
Terjevik
Tromsøya
ABC
mellomrom
ferdig

Figure 13 When the user click the textfield a keyboard will show up and they can write in the time they actually used.

To see a complete picture of all the views and how they interact with each other see Figure 14. Here there will be arrows that show what happens when the user clicks on a buttons or textfields.

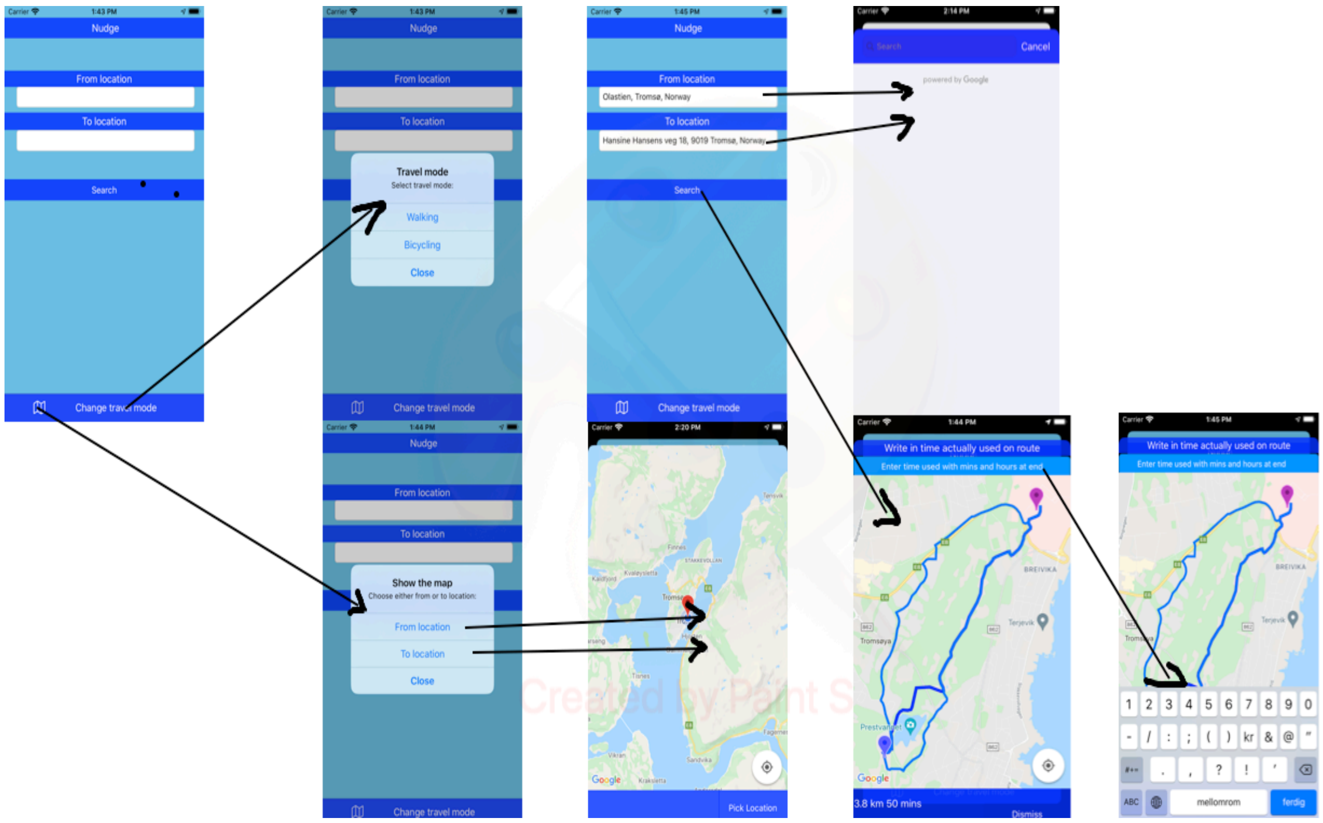


Figure 14 In this figure there will be an overview over what happens when the user clicks different buttons and textfields, this will be shown with arrows.

4 Implementation

The app was implemented in a specific way to be able to all the things to nudge people towards a green transportation. How this app was implemented will now be talked about.

4.1 How this app was made

This app was made using Xcode and the first thing that had to be done to start the making of this app is to make a new project in Xcode. When one makes a new project in Xcode one gets the choice of which application one wants to make Figure 15.

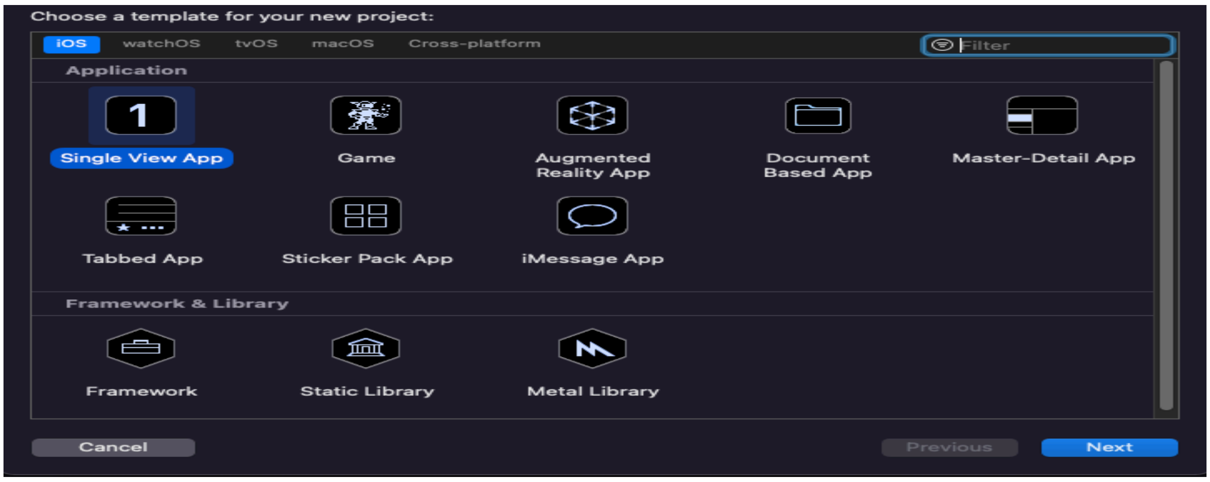


Figure 15 How it looks at the start of creating a new project in Xcode.

Here the choice of making a single view app was made, which just means that the project starts with one view controller. More views can be added if the app one is designing needs it. This app will start up in a window and then will make other windows pop up when the user clicks buttons and textfields. These windows are called view controllers and will be the size of the version of iPhone one choses to go with. Here Xcode will have different iPhone version to the newest and to the oldest. The iPhone version that was chosen for this thesis is iPhone SE 2nd generation, this is because this is the iPhone the app will be tested on and the design of the app most match the phone. When the project has been made, go to the storyboard, this is where the design of the app is made, in here there will already be a `ViewController` with a view Figure 16.

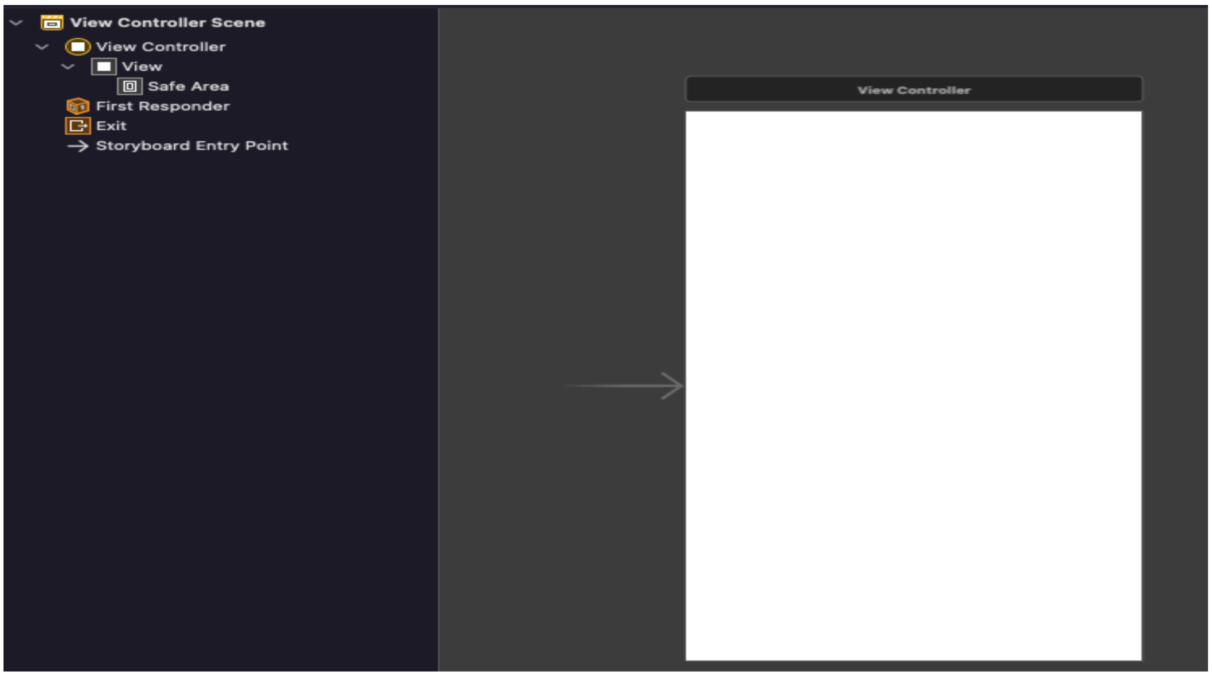


Figure 16 How this ViewController looks in the storyboard.

4.1.1 Designing the app in storyboard

The first thing that was added is a label that was put at the top of the screen and dragged out, so it went all the way to the edges. In this label the name of the app got written, which is Nudge. This text also got centred, because then the text is where it is natural to look. The reason for the label getting dragged out all the way to the end of the screen so it is visible and a nice way of creating a way to put it apart from the background. Then another label was added, a little bit under the previous label, that also got dragged all the way to the edges. Then "From location" got written into this label and this text also got centred. Then a textField got added under the second label, this textField got dragged out but not all the way to the edges. The label before the textfield will tell the user that this textfield is where they put in where they want to travel from. In this textField nothing is written since it is going to take in user input. The details of how the textfield works will be talked about later in making the code for the `ViewController`. Then a new label was added, this label is a little way down from the textField and got dragged out all the way to the edges, in this label it got written "To location" and it also got centered. Then another textField got added a little way down from the third label, this textField got dragged out a little but not all the way to the edges and so it aligned with the previous textField. This label with the textfield will also inform the user that they can interact with this textfield to put in where they want to travel to.

Then a button got added a little way down from the recently added textField, this button got dragged out all the way to the edges and then search got written into this button and the text got centered. Then a toolbar got added, this was then moved to the end of the screen and was dragged out, so it covered at the edges. In this toolbar a fixed space was added, then a bar button was added. The fixed space was added because then it is space between the button and the edge of the screen. This bar button will just have the image of a map, since this button is going to make the app show the map either from location or to location. Then a bar button was added, into this it was written “change travel mode”, then a fixed space was added between this bar button and the previous bar button. Then a flexible space was added after the "change travel mode" button. Then every label, button and the toolbar background color were changed to blueberry blue, and the text color was changed to white color. Then the view´s background color got changed to sky blue, so it is a little lighter blue then the rest is. Then there will be a need to add constraints so that it stays this way int the runtime for the app and, so it stays the same for all versions of iPhones. To add this constraint, it will be easiest to select each of the labels, textfields, button and toolbar (the buttons in the toolbar will not need constraint since they will be at the same place as the toolbar) separately and then right click on the symbol that looks like a tie fighter Figure 17.When this is right click it will come up a text that looks like this Figure 18, under Selected Views click on add missing constraints.

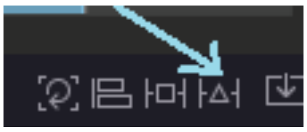


Figure 17 how this symbol looks like

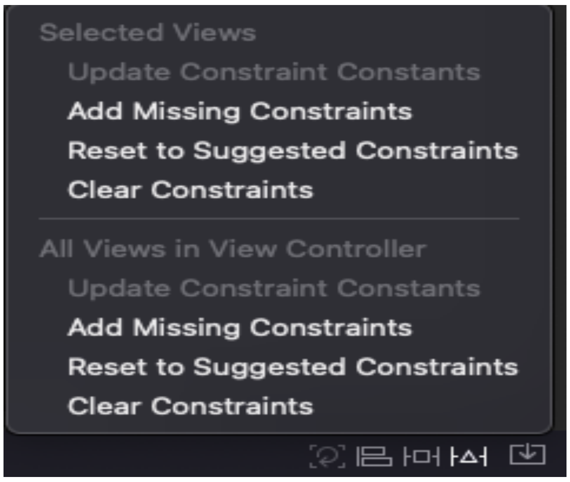

Selected Views
Update Constraint Constants
Add Missing Constraints
Reset to Suggested Constraints
Clear Constraints
All Views in View Controller
Update Constraint Constants
Add Missing Constraints
Reset to Suggested Constraints
Clear Constraints

Figure 18 How it the text that shows up when it is right clicked looks like

After all this the first `ViewController` is done, and this is the controller that the app will start in when it gets launched. The end result will look like Figure 19 and to get this end result it has to be designed in the way that is explained above.

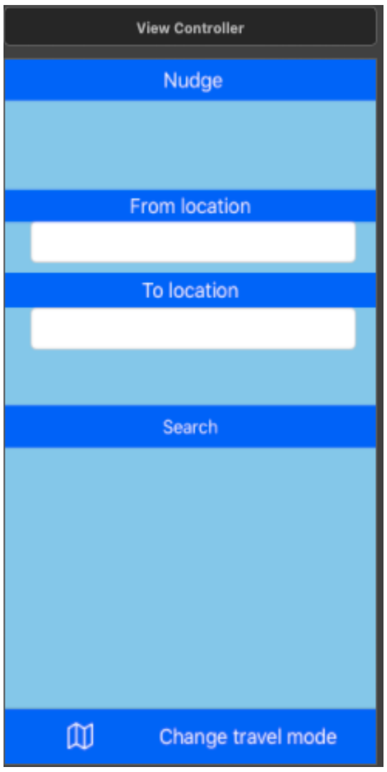


Figure 19 How the first view will look like.

The app will need more view controllers, so two more view controllers got added in the main storyboard. These view controllers each got a name `MapsViewController` and `RoutesViewController`, they also got

their own storyboard id which is the same as their name. The `ViewController` that was started with also got a storyboard id which is the same as its name. These storyboard ids will help with connecting the controllers with each other, and go to different views in the app.
The `MapsViewController`, is the view controller that will show the map and allow you to pick a place from the map and will also show the previously mention information about business, like name, rating and if it is open right now. So here a button got added, that will be called "pick location", and will be at the bottom of the screen all the way to the right and will be dragged someway up and to the left so the text is visible. The button background color got changed to blueberry blue and the text color gets changed to white. Then a label got added and put at the bottom to the left and dragged someway up and to the right where the button is, it will be dragged as close as possible without covering the button. Then the text will be removed from the label, and the background color will be changed to the same as the button and the text color will be changed to white. Also, the number of lines the label can take in will be changed to 2 and the size of the text to systems 13.0. This will be so that it can display the whole info for the business, since the length of business names varies. Then a view got added, this got dragged out, so it covered everywhere on the screen except at the very bottom where the previous added button and label are. There is no need for color on the background since the map will be added into in the runtime for this controller. Then the constraints get added to all of them, so they stay at the same position and no clipping happens at run time and it also looks the same on other versions of iPhone. The end result will look like Figure 20, this will not be the way the app looks when it is launched since the map will be where the white is. How to get this map in this controller will be explained later, but to get the end result of this view controller the design that is described above need to be used.

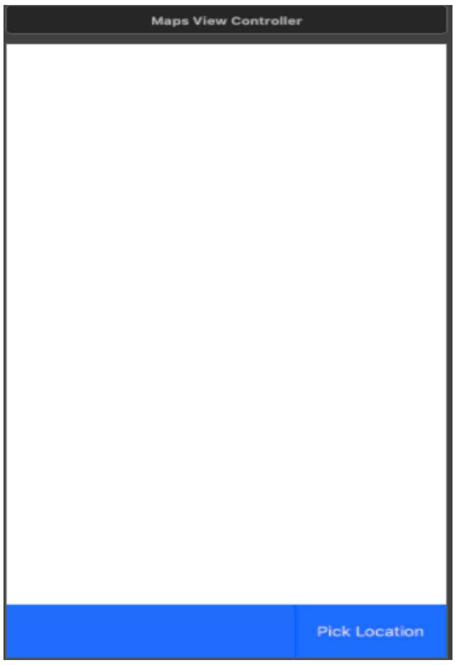
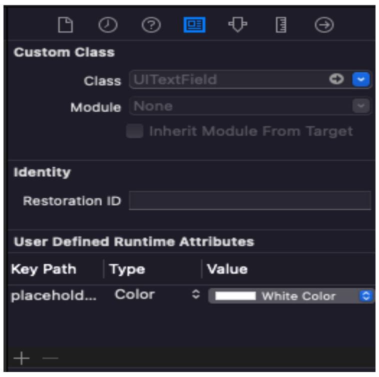


Figure 20 How the second view will look like.

The `RoutesViewController`, is the view controller that will show the routes between the two location that were written in the textField in the view controller and will show information about distance and time and be able to take in input from the user if duration is different than displayed duration (this duration will then be saved and used later for the user). It will also have a way of dismissing the route and go back to the view controller to make a new one. A button got added at the very bottom at the left and is called “dismiss” (since it will be used to dismiss the routes controller). The dismiss button got dragged out, so the text is visible. The background color got changed to blueberry blue and the text color got changed to white. Then a label got added at the very bottom at the left, then it got dragged a little up and to the button without covering it. Then the background color and the text color got changed to the same as button, and then the text is removed. Then a label got added at the top and got dragged out all the way to the screen and its background color and text color got changed to the same as the button. Then in the label the text was replaced with “Write in time actually used on route”. Then a textfield got added at the bottom of this label, this textfield gets dragged out all the way to the edge and then the background color got changed to aqua and the text color got changed to white. The color was chosen so that the textfield will be visible and stand out. The keyboard to the textfield got changed to number and punctuation and set the return key to done. In the placeholder write in this "Enter time used with mins and hours at end". This will be so that the user gets up the number part of the keyboard and so they can click done to exit out from the textfield. The placeholder is there so they can see what this textfield is for and what to write in it, but it will not come up as text when they write in the textfield and then they don´t need to remove this text before writing. Then in the custom class to the textfield in the storyboard Figure 21, in the user defined runtime attributes add a new key. In the text to this key write in placeholderLabel.textColor and change the type to color and then change the color to white. This will make the placeholder text be the color white in the runtime for the app. The reason for the white text color is so that the placeholder text is visible. Then a view was added and dragged so it covered every edge except the bottom and the top and got dragged as close to the button and the label as it could without covering it and got dragged the other way as close as it could to the textfield. Then the constraints got added to all of them, so they stay the same place in run time and in other versions of iPhone. The end result will look like Figure 22, to get this end result the design steps described above need to be followed.


Custom Class
Class UITextField
Module None
Inherit Module From Target
Identity
Restoration ID
User Defined Runtime Attributes

Key Path	Type	Value
placehold...	Color	White Color

Figure 21 how this looks in the storyboard

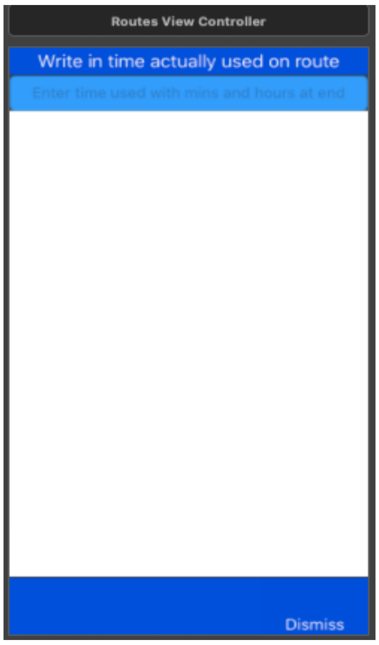

Routes View Controller
Write in time actually used on route
Enter time used with mins and hours at end
Dismiss

Figure 22 How the third view will look like.

4.2 Making the files and the classes for the controllers and connecting things from storyboard

After the main storyboard is finished it is time to connect things into their corresponding classes, but first two more files need to be added to the project. One will be called MapsViewController and the other RoutesViewController. These files will only have one imported library in them which is Foundation, but this library is not needed so it will be removed. Both of them will need to import the UIKit library. Then in the MapsViewController file make a class called MapsViewController and that will inherit from the UIViewController class (therefore the UIKit library was needed, since this class is in this library). This will make MapsViewController a subclass of the UIViewController, and let it access functions that are in that class it inherited from.

Then in the RoutesViewController file also make a class, but this time called RoutesViewController and will also inherit from the UIViewController class. There will already be a ViewController file that has the imported library UIKit, and it will also have a class called ViewController class in it. This class will already be set to inheriting from the UIViewController. Now some of the things in the storyboard need to be connected with the classes to the different controllers. So first the ViewController, here both the textfields will be connected into the ViewControllers

class. They will each be connected as variables at the top inside the class, the first connected will be the first textfield that was added in the storyboard and this textfield will be named fromLocationTextfield. The second one will be connected and will be named toLocationTextfield. They are given these names so that it is known which textfield is from the location and which is to location. Then the three buttons will each be connected as functions. The first one connected is the search button's function will be called searchRoutes. The bar button with just a map as image will be connected and called showTheMap. The bar button called change travel mode will be connected and it functions name will be changeTravelMode. The function searchRoutes will direct the user over to RoutesViewController where their route gets drawn. The function showTheMap will first make the user choose from or to location and then direct the users over to the MapsViewController where they can pick their location from the map. The function changeTravelMode will give the user the option to choose which mode they want to travel with. How these function does this will be presented in 4.3.2, 4.3.3 and 4.3.4.
Then in the MapsViewController class the second UIView in the MapsViewController will be connected as a variable and called mapView. The reason for this being a variable is because it will display the map, and this is done thought the class GMSMapView. It will need to display this map though another variable because otherwise the map will show up without some essential thing. It will be map without a marker and the location button and it will no longer be centered to Tromsø. This will be a map that can´t be used, so for this reason this mapView will be a UIView when it is connected and will use another variable that will be GMSMapView that will be display on the frame to the mapView.
Then the label will be connected as a variable and called businessInfo. Then the button pick location will be connected as a function and called pickLocation.
Then in the RoutesViewController class first connect the textfield and call it timeActuallyUsed. Then the rest of the connections will be the same as for the MapsViewController, expect the label and the button (this will be the label at the bottom). The label will be called info and the button function will be called clearScreen (it can´t be named dismiss, since it already exists an in-built function in swift that is named that). This function when clicked on will return the user to the view controller. In this function the inbuilt function dismiss will be called, which takes in animated and completion. Set animated to be true and completion to be nil.

After this it is time to close Xcode and open the terminal and go to the folder for this project, then in the terminal write init pod, which will create a new cocoa podfile. Then open that pod file and write in pod 'name', where the name will be the name of the library that needs to be installed through cocoapods. These libraries are GoogleMaps, GooglePlaces, Alamofire, SwiftyJSON.
In AppDelggate, before the initialization of the class, import both GoogleMaps and GooglePlaces. Then in the class, in the function didFinishLaunchingWithOptions call GMSServices.provideAPIKey("api key") and call GMSPlacesClient.provideAPIKey("api key"). The api key will be gained through google cloud platform, to get a working API key there needs to be a billing account linked here. This is because Google has a pay as go to use all their APIs.
ViewController, MapsViewController and RoutesViewController will need to import GoogleMaps, GooglePlaces, and CoreLocation, RoutesViewController will also need to import Alamofire and SwiftyJSON. ViewController will need to also import EventKitUI, EventKit and UserNotifications.

4.3 Implementing the app in ViewController

In `ViewController`, before the initialization of the class but after all the libraries are imported make an enumeration called *TravelModes*. Inside *TravelModes* define three different cases walking, bicycling, driving and default. Enumerations [32] «defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code»
This `ViewController` will need to take in the protocol *UITextFieldDelegate*, to manage the textfields. It will also need to take in the protocol *CLLocationManagerDelegate*, to be used with the location manager object, which is used for location. Then inside the class define a variable called *fromLocation* and set it to be optionals (its value needs to be defined later) of type *CLLocationCoordinate2D*, which is a structure that will make this variable into a coordinate. Optionals [33] «represents two possibilities: Either there *is* a value, and you can unwrap the optional to access that value, or there *isn't* a value at all». They are used when the value is absent, this might be because it needs to be set later. It must be unwrapped before it is used, except for when it is set to be something, and it might need to be checked to see if it has been set yet. Then define a variable called *toLocation* and set it to be

optionals of *CLLocationCoordinate2D*. Then create a variable called *currentLocation* and set it to be optionals of the `CLLocation` class. Then define a variable called *locationManager* and set it to class `CLLocationManger`, this will help when tracking the user's location when they are using the app.
Then define a variable called *fromOrToLocation* and set it to true, which will make this variable be type Boolean and will make it start with being true. A Boolean is a type that can only be two things either true or false. Then define a variable called *travelModes* that is set to be the default case from the enumeration Travelmodes *(TravelModes.defualt)*. Then define a variable called center and set it to be `UNUserNotificationCenter` class *current* method. Then define a variable called *titles* and set it to be an empty array that is the type string. Then define a variable called *startDates* and set it to be an empty array of the type Date. Then define a variable called *locations* and set it to be an empty array of the type string. Then define a variable called *eventStore* and set it to be the class `EKEventStore`. Then define a variable called *calendar* and set it to be the `Calendar` class *current* method. Then define a variable called *intervalBetween* and set it to be an optional of type Date. Then define a variable called *eventStartDate* and set it to be an optional of type Date. Then define a variable called *eventTitle* and set it to be an optional of type string. Then define a variable called *eventLocation* and set it to be an optional of type string. Then define a variable called *date* and set it to be an optional of type Date.

4.3.1 Setting up the view to the controller and locations

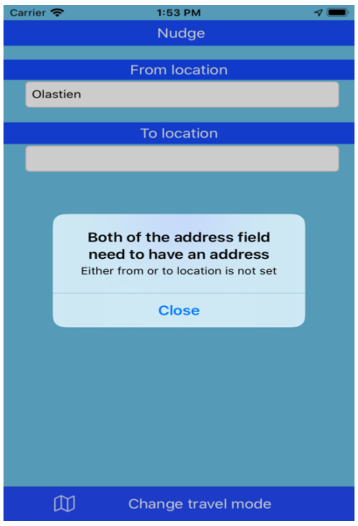
Then define a function called *viewDidLoad*, which will already be an inbuilt function in Swift that belongs to the `UIViewController` class, that `ViewController` will inherit from it. Since this is a function that is inherit from a class there will be a need to call the original function, because if not this definition of the function will override the original things in that function. Since the original function calls the view and makes it load in this is not something that is wanted, so the super of *viewDidLoad* will be called in here. Then after that in the function one can now do what one wants to be done when the view loads in. Here there are some variables that needs to be set up that were declared before this function. So, the textfields delegates will be set to be themself. If these are not set, they will be nothing, and this will be a

problem. Then the *location manager* delegate will be set to itself. If it is not set it will be nothing, and this will be a problem. Then after these things are set, there will be a need for location to be set, call the *location manager* function called *requestWhenInUseAuthorization*, this function will be used to request an authorization for location. Then call the function *startUpdatingLocation*, which will update the location to the user's current location. For the location to complete work there is also a need to put something into info.plist, the following privacy permissions need to be added into the plist file:
The key: Privacy - Location When In Use Usage; The key: Privacy - Location Always and When In Use Usage Description; The key: Privacy - Location Always Usage Description. The description for all these keys will be: This app requires usage of location. These will make the user be able to click when the location should be used, either in only the app, or always, they can also choose never, but then the app will try to force them to turn on location when the directions routes get drawn. Since it will need location to tell the person which roads they should take.
Then define the inbuilt function *locationManager (_ manager: CLLocationManager, didFailWithError error: Error)*. Inside this the error needs to be handled, the way to do this is create an alert with the title "Error:" and the message "\(error)". The message will show what the variable error is, the way the error is handled will be talked more about later.
Then after this function define the inbuilt function *locationManager (_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation])*, in this function set *currentLocation* to be last property to the locations array. This will set the *currentLocation* to be the last position to the user. Then call on four functions that will be defined later, *requestNotification*, *checkStatusAndGetAllEvents*, *getEventForToday* and *scheduleNotification*. All these functions will not take in anything and will not return anything. For the calendar and notification to work there is something that is needed in info.plist. add keys NSCalendarsUsageDescription and NSContactsUsageDescription with the description: This app wants access to your Calendar. Then add the key NSRemindersUsageDescription with the

description: This app wants access so it can send you reminders. These will give the user the option to either give access or not through a pop-up window.

4.3.2 Implementing the function for the search button

Then inside the `RoutesViewController`, inside the class define two variables that are coordinates by using the *CLLoctionCoordinate2D* and name them *originCoordinate* and *destinationCoordinate* and let them be optionals. Then back to the `ViewController`, in the function *searchRoutes* it needs first to check if both coordinates (from and to) are not nil. If they are not nil, create a constant called storyboard and set it to be the `UIStoryboard`, then create another constant called *VC* and set it to a function belonging to the `UIStoryboard` called *instantiateViewController (withIdentifier: (the storyboard id belonging to RoutesViewController)*. Then the coordinates to the `RoutesViewController` needs to be set to the coordinates to the `ViewController`, then the `RoutesViewController` can be presented. If one or both coordinates are nil, define a constant called alert and set it to be the `UIAlertController` class (which will create an alert), all calls on classes will have parenthesis, in these there will be a need to write in three inputs title, message and *preferedStyle*. Make the title say, "Both of the address field need to have an address" and the message say, "Either the from or to location is not set", the style can be set to alert. There are two styles one can use depending on how one wants the alert to show up, these styles are alert and action sheet. They look different when showing up on the screen but there is a bug with the style action sheet on IOS 12 and up, that has not been fixed yet (as far as the author have investigated it in 2021). This means that for this app the style that is going to be used for all alerts will be the style alert. Define a constant called *closeAction* and set it to be the `UIAlertAction` class (which creates an action to the alert), in the parenthesis there can be three inputs title, style and handler. All actions can have a handler or not, this handler will create a callback where one can put in things that one wants to happen when the action gets clicked, but there will be no need for this in this action. The title will say “close”, and the style will be set to cancel, this will make the action close. Then add this action to the alert by using the method *addAction* and put in *closeAction* in the parenthesis to the method. Then present the alert by using the inbuilt function *present* which takes in three things, in the parenthesis write in the alert, then animated: true and completion: nil. This will make the alert show up on the screen when the button is clicked and one or more of the coordinates are missing. See how it looks in *Figure 23*.



Both of the address field need to have an address

Either from or to location is not set

Figure 23 What the alert looks like when running the app.

4.3.3 Implementing the function to the map button

Then in `MapsViewController` define a variable called *originOrDestination* and set it to be optionals of type String, then define a variable called *prevVC* and set it to be the `ViewController` class. Then in `ViewController` inside *showTheMap*, create a constant called *storyboard* and set it to be the `UIStoryboard`, then create another constant called *VC* and set it to a function belonging to the `UIStoryboard` called *instantiateViewController (withIdentifier: (the storyboard id belonging to MapsViewController).* Then create and alert with the title "Show the map" and message "Choose either from or to location:". Then create an action by using the *UIAlertAction* function which takes in a title, style, and handler. With the title “From location” and the style default. Here it will be a need for a handler, in this callback set *originOrDestination* (belonging to the `MapsViewController`, use the variable *VC* to get it) to be "Origin_location" and then call an inbuilt function called *present* that will present the `MapsViewController`. Then create another action with the title “To location” and style default, here there will also be a need for a handler, so in the callback to the handler set the *originOrDestination* to "Destination_location", then present the `MapsViewController` by using the function present. Then create an action with the title "close” and the style cancel, that will close the alert without changing anything. Then add all these actions to the alert and present the alert on the screen.

4.3.4 Implementing the function to the change travel mode button

In the function *changeTravelMode* create an alert with the title "Travel mode" and message "Select travel mode:". Then create an action with the title "walking" and style default, this action will need a handler, so in the callback to the handler set *travelModes* to be the case walking from the enumeration *TravelModes*. Then create another action with the title "bicycling” and style default, and in its handlers, callback set the case to be bicycling. Then create an action with the title "close" and the style cancel, that will close the alert. Then add all these actions to the alert and present the alert. Then create another action with the title "driving” and style default, and in its handlers, callback set the case to be driving. Then create an action with the title "close" and the style cancel, that will close the alert. Then add all these actions to the alert and present the alert.

4.3.5 Implementing the calendar and notification feature

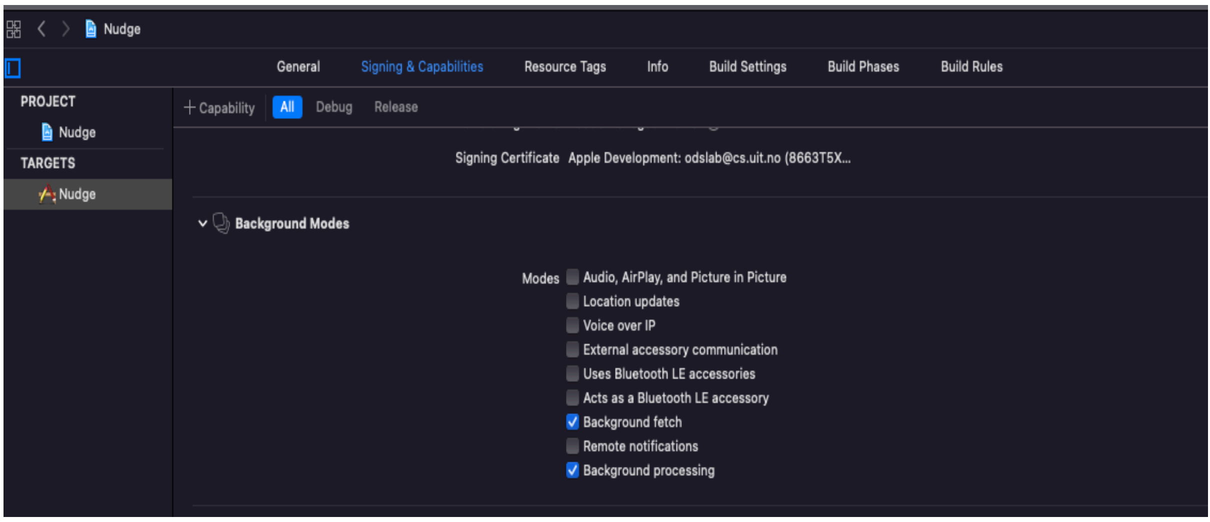
Then define the function called requestNotification, in this function call on center requestAuthorization function. This function will take in an array, in this array call on the propety alert and sounf. This function will have a call-back in this callback check if error is not nil, then present an alert for the user. This alert will have the title "Permission not granted" and the message "this app will not be able to give any notifications". This alert will have a close action with the title "close", add this to the alert and then present it.

Then define the function called scheduleNotification, in this function define a constant called content and set it to be the UNMutableNotificationContent class. Then set content title property to be the string "You have an event today". Then set content subtitle property to be the string "It is now 2 hours to your event starts". Then chekc if eventTitle is not nil, if it is not then set content body property to be eventTitle. Then set content sound property to be UNNotificationSound default property. Then chcek if intervalBetween is not nil, if it is not define a variable called dateComp and set it to be DateComponents class. Then set dataComp.month to be calendar component function that takes in the property month and intervalBetween. Then set dataComp.day to be calendar component function that takes in the property day and intervalBetween. Then set dataComp.hour to be calendar component function that takes in the property hour and intervalBetween. Then set dataComp.minute to be calendar component function that takes in the property minute and intervalBetween. Then define a constant called trigger and set it to be UNCalendarNotificationTrigger function which take in dataMatching and repeats. Put in dataComp for dataMatching and for repats put in true, then define a constant called request and set it to be UNNotificationRequest which take in identifier, content and trigger. For idenfier put in the string "identifer", for content put in content and for trigger put in trigger. Then call on center add function which take in an input so put in request. This function will have a call-back and in the call-back check if error is not nil if it is then create n alert with the title "Permission not granted" and message "this app will not be able to check calendar for events". Then create a action for closing with the title "close", add this action to the alert and then present the alert.
Then define a privat function called createDate which will take in an Int called year and will return an Date. In this function define a variable called components and set it be the DateComponents class. This will create a new instance of this class that will not override the last version. Then set components year property to be year and set components timezone property to be the TimeZone function which take in one input, put in zero here. Then return calendar date function as an optional and the function will take in components as its input.

Then define a privat function called get and in this function define a contant called calendars and set it to be eventStore calendars function which take in a input, in here put in the property event. Then create a for loop with the variable calender and will iterate through calendars. In this for loop create a guard statement with calender allowsContentModifications property. Then create a else statement and in this else call the continue statement, this will tell the guard loop to stop an continue the iteration of the code. Then outside the else but still in the for loop define a constant called startYear and set it to be calendar component function which will take in the year property and Date structure. Then deifne a constant called start annd set it to be createDate function which will take in an int year, put in startYear. Then define a constant callled end and set it to be createDate function and for year put in 2025. Apple will only allow and end year that is four year from the start date. Then define a constant called predicate and set it to be eventStore predicateForEvents function which will take in (start, end, and an array of the type Calendar, put it calender). Then create a constant called events and set it to be eventStore events function which will take in a input called matching, put in predicate here. Then create a for loop that will iterate over all event in events, in the for loop append event title property to the titles array. Append event startDate to the startDates array, and then check if event location property is not nil and then if it is not append event location to the location array.
Define the function called checkStatusAndGetAllEvents, and in the function define a constant called currentStatus and set it to be EKEventStore authorizationStatus function which will take in EKEntityType event property. Thn create a switch statement with currentStatus, define a a case authorized and in this case call on the get function. Then define a case notDetermined in this case call on eventStore requestAccess function which will take in event. Then create a call-back with accessGranted and error and then check if accessGranted, if it is then call on the function get. If not create an alert with title "Could not get access" and the message "Change settings to Allow access", define a close action with the title close and add this to the alert and present the alert. Then define a case restricted in this case creat an alert with the title "Access is resticted" and the message "if this is not what you wanted go to setting and change to Allow". Define a close action with the title "close",

add this action to the alert and present the alert. Then define a case denied and in this case create an alert with the title "Access is denied" and the message "if this is not what you wanted go to setting and change to Allow", have close action to this alert and present the alert. Then define a case default and in this create an alert with the title "Error:" and the message "Something went wrong" and create a close action to this alert and present this alert.
Define the function getEventForToday, in this function create a constant called timeZone and set it to be TimeZone current method. Then set calendar timeZone property to be timeZone, then set date to be the Date class. Then create a for loop that will iterate over all instances of startDate in startDates. In this for loop create another for loop that will go through all location in locations. Then in this for loop create a for loop that will go through all title in titles, then in this for loop check if date is less than startDate, then if it is then set eventStartDate to be startDate. Then set eventTitle to be title, set eventLocation to be location and call a function calles findDestinationCoordinate that will be defined later. This function will not take in any inputs or return anything, then create a constant called oneHourBefore and set it be a Double and the value 7200, which is two hours in seconds. Then set intervalBetween to be startDate addingTimeInterval function which will take in one input, put in oneHourBefore with a minus before it so the value will be substrackted.
Define the function findDestinationCoordinate, in the function create a constant calles geocoder and set it to be CLGeocoder class. Then define a variable called lat to be an optional of the type CLLocationDegrees, then define a variable called lon to be an optional of the type CLLocationDegrees. Thne call on geocoder geocodeAddressString function which will take in one input, put in eventLocation, then in the function create a callback with placemarks and error. Then in the call-back deifne a constant called placemark and set it to be optional of placemark first property. Then set lat to be the latitude property access through coordinate property to location which is the property to placemark, set both placemark and location to be optional. Then set lon to be longitude property access through coordinate that is a property of location that is a property to placemark. Also here set placemark and location to be optional. Then set   toLocation to be CLLocationCoordinate2D function

which will take in latitude and longitude, put in lat and lon as inputs here. Then set toLocationTextfield text property to be eventLocation.
Then in the project setting under Signing & Capabilities add capability called background modes and click for Background fetch and Background processing. This will let the notification get sent in the background when they are needed to be sent to the phone by the app, look Figure 24 at to see how this look like in the project settings.



Signing Certificate Apple Development: odslab@cs.uit.no (8663T5X...

Background Modes

Modes
- [ ] Audio, AirPlay, and Picture in Picture
- [ ] Location updates
- [ ] Voice over IP
- [ ] External accessory communication
- [ ] Uses Bluetooth LE accessories
- [ ] Acts as a Bluetooth LE accessory
- [x] Background fetch
- [ ] Remote notifications
- [x] Background processing

Figure 24 How it looks in the project setting with both for Background fetch and Background processing enabled.

4.3.6 Implementing the autocomplete feature to the textfields

Then in the `ViewController` file, after the end of the class, create an extension and call it `ViewController` and let it take in the protocol *GMSAutocompleteViewControllerDelegate*. This protocol is need for using the `GMSAutocompleteViewController` class, which is used for getting autocomplete feature. Then inside the extension define the inbuilt function *textfieldDidBeginEditing*, this will be called when one of the textfields are clicked on. This function will take in an `UITextfield` that can be used to know which of the ones are getting clicked. Then inside the function define a constant called *autocompleteController* and set it to be the `GMSAutocompleteViewController` class. Then create a constant called *filter* and set it to be `GMSAutocompleteFilter` class, this will help with filtering the autocomplete list. This class will have some properties that can be set to filtering the autocomplete list to Norway and to only Tromsø. To do this set *filter country* property to "NO". Then set the delegate to the

autocompleteController to be itself. Then create a constant called *north* and set it to be the result of the inbuilt function *CLLocationCoordinate2DMake(),* which takes in two parameter *latitude*(69.776753) and *longitude*(19.280129) and these the author got from Google Maps. This function will take in these two values and format it to a coordinate data structure. Then create a constant called *south* and also set it to be the result of the function *CLLocationCoordinate2DMake()*, but this time with *latitude*(69.552071) and *longitude*(18.565459). Then set the *filter locationRestriction* property to be the function *GMSPlaceRectangularLocationOption()* which takes in two coordinates, so put in the coordinate *north* and the coordinate *south*. Then set the *autocompleteController* autocompleteFilter to be *filter*, so that all the *filter*s get added to this *autocompleteController* class. This all needs to restrict the *filter* for the autocomplete. Now the autocomplete and the textfields needs to be taken care of. So, first check if the `UITextfield` is the *fromLocationTextfield*, if this is the case then *fromOrToLocation* to be true. Then call on an instance method called *resignFirstResponder* on the *fromLocationTextfield*, this will make it no longer be the first responder on its window. Then present the *autocompleController* in a new window over the previous one, this window will show a search field, with a cancel button at the end, and under it a table where suggestions will be for places and addresses. If the `UITextfield` was not *fromLocationTextfield*, it must be *toLocationTextfield* (it can´t be nil, since one of textfield had to be clicked to call on this function), so in the else set *fromOrToLocation* to be false. Then call on *resignFirstResponder* on the *toLocationTextfield*, and then present the autocomplete table in a new window.
Then nothing more is needed in this function, but there are still things that needs to be taken care of. Like when the place gets chosen, also when the cancel button gets clicked. So, inside the extension define the inbuilt function *viewController (_ viewController: GMSAutocompleteViewController, didAutocompleteWith place: GMSPlace)*. Inside this function check if *fromOrToLocation* is true, if it is then set the *fromLocationTextfield text* property to be `GMSPlace` class (which is named place from the function) *formattedAddress* property, which will be the address of the chosen place or

address. Then set *fromLocation* to be *place's coordinate* property, this will be the coordinate to the chosen place or address. In the else fromOrToLoaction is false, then set *toLocationTextfield* text to be place formattedAddress property. Then set the *toLocation* to be place coordinate property. Then at the end of function outside the if and else statements, call dismiss, so that it will dismiss the `GMSAutocompleteViewController` when something gets chosen. Then in the extension define the inbuilt function *viewController (_ viewController: GMSAutocompleteViewController, didFailAutocompleteWithError error: Error)*, in here handle the error. To tell the users that something is wrong, it will show an alert with the title "Error:" and the message "\(error. localizedDescription)", the message will tell what error is. Then create a close action with the title "Close" and style cancel, so that this alert can be closed, then add this action to the alert and then present the alert on the screen.

Then in the extension define the inbuilt function *wasCancelled (_ viewController: GMSAutocompleteViewController)*, in this function call on the inbuilt function *dismiss*, so it can dismiss the `GMSAutocompleteViewController` when the cancel button gets clicked and bring the user back to the `ViewController`.

To edit how the colors looks in the GMSAutocompleteController view, go to the AppDelegate. Then in the function *didFinishLaunchingWithOptions* access the variable *barTintColor* through the `UINavigation` class and the appearance function and set it to be `UIColor` blue. This will make the search bar color to blue (it exists only two types of blue in Swift UIColor, and it is system blue, which is light blue and blue, see Figure 25 to see how this color blue looks like). Then access the variable *tintColor* also through the `UINavigation` class and the appearance method and set it to be `UIColor` white. Then access the variable *barStyle* like the one before and set it to be the enumeration *UIBarStyle* default case. Then to get the text white in the search bar, do this: define a constant called *searchBarTextAttributes* and set it to be type *[NSAttributedString.Key: AnyObject]* and then set it to be *NSAttributedString Key foregroundColor rawValue* with the `UIColor` white and *NSAttributedString Key font rawValue UIFont systemFont*. Then set *defaultTextAttributes* access through the `UITextfield` and the appearance function (which takes in the `UISearchBar`) to be *searchBarTextAttributes*. See how it looks in Figure 25.

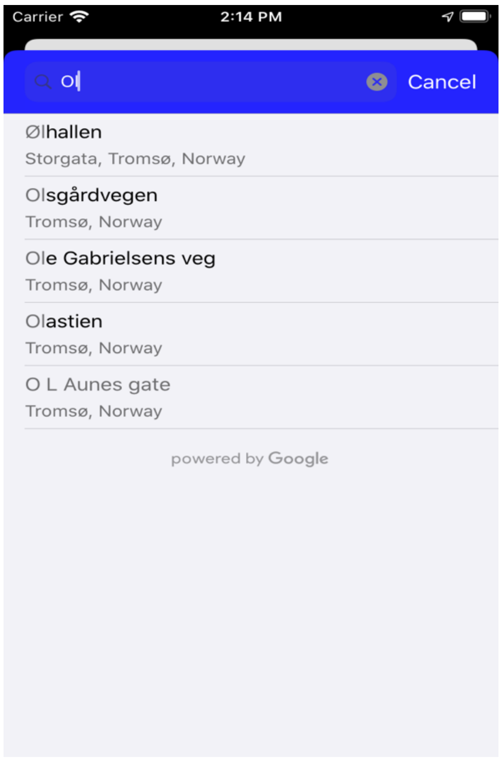


Figure 25 What the autocomplete controller looks with search text and suggestions

4.4 Implementing the app in MapsViewController

Then everything with autocomplete and the first View is done. Now the `MapsViewController` needs to be fixed, so there can be a map where the user can choose the location, instead of searching for it with the autocomplete. So, where the class `MapsViewController` is defined make it take in these two protocols *CLLocationManagerDelegate* and *GMSMapViewDelegate*. The *CLLocationMangerDelegate*, to be used with the location manager object, which is used for location. The last protocol is for the `GMSMapView` class, which is for the map, so its delegate will also help with doing stuff that is needed on the map. Then in the class create a variable called *googleMapView* and set it to be optionals of the `GMSMapView` class. Then create a variable called *geocoder* and set it to be the `GMSGeocoder` class. This class will help with reverse geocoding latitude and longitude into a readable address. Then create a variable called *marker* and set it to be `GMSMarker` class, this class will be used for creating a *marker* on the map. Then create a variable called *address* and set it to be optionals of the string type. Then create a variable called *city* and set it to be the same as *address*. Then create a variable called *currentCoordinate* and set it to be optionals of the *CLLoctionCoordinate2D* structure. Then create a variable called *currentLocation* and set it to be optionals of the `CLLocation` class. Then define a variable called *locationManager* and set it the class `CLLocationManger`, this will help when tracking the person location when they are using the app.

4.4.1 Setting up the view and location

Then override the inbuilt function called *viewDidLoad*, then in the function calls on the super of *ViewDidLoad* function (which calls the original function which was inherited from the `UIViewController` class). Then set the *address* to be an empty string and do the same with the *city*. Then the location manager delegate will be set to itself. If it is not set it will be nothing, and this will be a problem. Then after these things are set, there will be a need for location to be set, call the location manager function called *requestWhenInUseAuthorization*, this function will be used to request an authorization for location. Then call the function *startUpdatingLocation*, which will update the location to the user's current location.

Then after the function *ViewDidLoad*, define the inbuilt function *locationManager (_ manager: CLLocationManager, didFailWithError error: Error)*. Inside this the error needs to be handled, the way to this is create an alert with the title "Error:" and the message "\(error)". The message will show what the variable error is, the way the error is handled will be talked more about in 4.8. Also create a close action to this alert so the user can close this alert.

Then after this function define the inbuilt function *locationManager (_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation])*, in this function set *currentLocation* to be last property to the locations array. This will set the *currentLocation* to be the last position to the user.

4.4.2 Setting up the map

Then create a function named *setUpMap*, this function will set up the map with a *marker*. In this function create a constant called *latitude* and set it to be *currentLocation coordinate latitude*, then create another constant called *longitude* and set it to be *currentLocation coordinate longitude*. This coordinate will be the user coordinate, since *currentLocation* will be where the user is located. Then set *currentCoordinate* to be *CLLocationCoordinate2D* structure which takes in two inputs *latitude* and *longitude*, so here the two newly constants will be put in. Then check if *currentCoordinate* is not nil, since it can´t never be nothing and if it is something is wrong, therefore it will be an else to the if statement that handles the error. For now, in the else create an alert with title "Error:" and message "current location is nothing, make sure positions and GPS is turned on". This alert will have a close action and will be present for the users, this way the users will know something is wrong and that they can try to make sure their GPS and positions is turned on. In the if statement create a new constant called *camera* and set it to be `GMSCameraPosition` class *camera* function, this function takes in three inputs *latitude*, *longitude* and *zoom*. Put in the two constants *latitude* and longitude and set *zoom* to be 10.0. This is how much the camera starts zoomed out, and it needs to be 10 so the user can see the whole city. The zoom levels inputs values can be from 1 and up to 20, and the higher the zoom value is the closer the zoom level of the map will be. Then set *googleMapView camera* to be the newly created camera and set *googleMapView mapType* to be normal. This means that the map will be the normal map, there are others map types to choose from. A feature where one chooses which map type one wanted could be implemented to create it more friendly to the users. This is something to think about when someone is continuing the work on this app. Then call on *geocoder* (which is the `GMSGeocoder` class) *reversGeocodeCoordinate* function, which takes in a coordinate and a completionHandler. To get the coordinate use *CLLocationCoordinate2D* struct and put in the *latitude* and *longitude* constants into it. The completion handler is there to handle the callback which is invoked with the reverse geocoding result. For that reason, after the curl bracket write *response*, *error* in, and click enter between curl

bracket and response. Then still inside the callback if error is nil (which mean there is no error), if (create a new constant called result) is the response? *firstResult().* The question mark is for unwrapping the optionals response, *firstResult ()* will get the first result for the response (which get it from the *reversGeocodeCoordinate* function). Then in the if statement set *googleMapView* delegate to itself, then create a constant called *lines* and set it to be *result lines* (let this be optionals) and cast it to the type of array that takes in type string. Then set the *city* to be *result locality*, this locality will be where the user is located now. Then set *marker* (`GMSMarker` class) position property to be the current coordinate, do this by using the *CLLocationCoordinate2D* struct and put in the constant's *latitude* and *longitude* in the input field. Then set *marker snippet* property (this property will show a text when the marker is selected) to be *lines joined* (which is a method and takes in a separator, put in here "/n"). The separator will create a separation between every new line in the text to the snippet. Then set *marker title* property to be *result locality*, make sure this is not nil by using two question marks (this is the way of checking a value is nil without an if statement), if it is it will return a string that says "Unavailable". While if it is not nil the string of text that is in locality will be shown instead on the marker title. Then set the *address* to be *lines joined* with the separator "/n". Then *marker* will have its own map property set to be the *googleMapView*. Then create an else statement to the previous if statement, this means the unwrapping, unwrapped nil. So, in this else create an alert with the title "Error:" and message "please try again later" and with a close action and present this alert.
This will be everything that is needed for this function, so outside this function override the inherit (this was inherit from the `UIViewController` class) function *viewDidAppear* (which takes in a Boolean, in this function call the super (the original function) of the *viewDidAppear(true)* function, if this is not called this define will override the original function completely, and this not something that is wanted. Then after this set *googleMapView* to be `GMSMapView` class, the reason why it is set now and optionals before this is because now the view has appeared, and it can be set. Then set *googleMapView* to be `GMSMapView` *(frame: mapView.frame),* this will make

it so that the frame to the *googleMapView* will be the *mapView* frame. Then set *googleMapView.isMyLocationEnabled* to be true, the *isMyLocationEnabled* is for creating the blue location dot on the map. Then to create the location button set the property *myLocationButton* (belonging to the settings which is a property of *googleMapView*) to be true, when this it is clicked it will center the map back to the blue dot. Then add a subview of *googleMapView* to the view, this will make google maps show up on the view. Then call on the *setUpMap* function.

4.4.3 Implementing the function to the button pick location

Then in the function *pickLocation* (which is connected through the storyboard), check if *currentCoordinate* is not nil. If it is just dismissing this view and go back to the previous. If it is not then creating a switch statement with *originOrDestination* variable, then create two cases and on default. Set the first case to be "Origin_location", in this case set *prevVC* (which is the previous View) *fromLocationTextfield* text property to be *address*. Then set *prevVC fromLocation* to be *currentCoordinate*, then dismiss this view and go to the previous view. Set the second case to be "Destination_location", then in this case set *prevVC toLocationTextfield* text property to be *address* and set *prevVC toLocation* to be *currentCoordinate*. Then dismiss the `MapsViewController` and go back to the `ViewController`.

4.4.4 Implementing so the marker moves when the user click somewhere on the map

Then outside the class create an extension of `MapsViewController`, in this extension define the inbuilt function *mapView (didTapAt coordinate: CLLocationCoordinate2D)* belonging to *GMSMapViewDelegate* protocol. This function will be called when the user taps a location on the map where the current marker is not at. In this function the marker needs to be moved to the new location, so set the *currentCoordinate* to be coordinate that the function takes in (since this will be the new coordinate the user selected on the map). Then call on the *geocoder reversGeocodeCoordinate*, which takes in *CLLocationCoordinate2D* and completionHandler. For the

CLLocationCoordinate2D (which takes in *latitude* and longitude) put in coordinate *latitude* and coordinate longitude. For the completionHandler do as in setUpMap *reversGeocodeCoordinate*, and in this *reversGeocodeCoordinate* function callback do the same as in *setUpMap*.

4.4.5 Making the information about the business show up the label

Then in the extension define the inbuilt function *mapView (didTapPOIWithPlaceID placeID: String, name: String, location: CLLocationCoordinate2D)* belonging to the *GMSMapViewDelegate* protocol. Then create a constant called fields and set it to be `GMSPlaceField (rawValue:)`. Here one can put in as many fields one wants from the `GMSPlace` object, which mean which information one wants from the place that Google has access to. For what that needs to be done here only three fields will be necessary and these are *opningHours, utcOffsetMinutes* and *rating*. Then create a constant called *placesClient* and set it to be `GMSPlaceClient` class, then call the function *fetchPlace (which takes in placeID, GMSPlaceField, sessiontoken and a callback)* that belongs to *placesClient*. For the *placeID* put in the *placeID* that already is here by the inbuilt function *mapView*, then for `GMSPlaceField` put in field, for *sessionToken* set it to nil and then the callback put in (place: GMSPlace? error: Error?) in. Then continuing still in the callback check if error is there (if it is not nil), if it is then make an alert with the title "Error:" and message "\(error.localizedDescription)" and also make a close action and add it to the alert and then present the alert. Then continuing in the callback check if the `GMSPlace` (with using the variable place) is not nil, then if it is not creating a constant called *isOpen* and set it to be place *isOpen* function (this function will return one of the cases from the enumeration called *GMSPlaceOpenStatus*). Then create a constant called *openString* and set it to be an empty string, then create a switch statement with *isOpen* constant, then create two cases and one default. Make one of the cases open from the enumeration *GMSPlaceOpenStatus*, in this case set the *openString* to be the string "This place is open now". Then make the other case closed from the same enumeration as open, then in the case set the *openString* to be the string "This place is closed now". Then in the default case set *openString* to be the string "Open and close time is not available". Then after this switch statement is finished creating a constant called *rating* and set it to be place *rating* property. Then create a constant called *ratingString* and set it to be the string "rating: \(rating)" (this will convert the constant rating with is an int into an string with some text before to tell the user that this number is the rating, Then set the *businessInfo* text to be name (get this from the inbuilt function) plus this string ", " (this is to create a comma and a space between every new information thing about the business) plus ratingString plus this string ", " plus *openString*. Now when the user taps a business it will come a text in the label with name, rating and if it is open, closed, or unknown. Then the `MapsViewController` is done, and now only the `RoutesViewController` is left.

4.5 Implementing the app in RoutesViewController

So, in the `RoutesViewController` where the class is defined make it takes in two protocols *CLLocationMangerDelegate* and *GMSMapViewDelegate.*
Then in the class create a variable called locationManger and set it to be optionals of the `CLLocationManager` class. Then create a variable *currentLocation* and set it to be optionals of the `CLLocation` class. Then create a variable called *currentCoordinate* and set it to be optionals of the *CLLocationCoordinate2D* protocol. Then create a variable called *prevVC* and set it to be `ViewController` class, since there are things needed from this class. Then create a variable called *googleMapView* and set it to be optionals of the `GMSMapView` class. Then create a variable called *geocoder* and set it to be `GMSGeocoder` class. Then create a variable called *polyline* and set it to be an empty array which takes in `GMSPolyline` class type. Then create a variable called *transferPolyline* and set it to be optionals of the type string. Then create a variable called *city* and set it to be optionals of the type string, and then create a variable called *originAddress* and set it to be optionals of the type string. Then create a variable called *destinationAddress* and set it to be optionals of the type string. Then create a variable called *originMarker* and set it to be optionals of the `GMSMarker` class, then create a variable called *destinationMarker* and set it to be

optionals of the `GMSMarker` class. Then create a variable called *totalDistance* and set it to be optionals of the type string, and then create a variable called *totalDuration* and set it to be optionals of the type string. Then create a variable called *storedOrigin* and set it to be an empty array which takes in the type String. Then create a variable called *storedDestination* and set it to be an empty array which takes in the type String. Then create a variable called *storedTime* and set it to be an empty array which takes in the type String. Then create a variable called *storedMode* and set it to an empty string, then create another variable called *changedDuration* and set it to an empty string. Then create a variable called *origin* and set it to be an empty string, then create another variable called *destination* and set it to be an empty string. Then create a variable called *timeChanged* and set it to be false.

4.5.1 Setting up the view and the location

Then define the inbuilt function *viewDidLoad* and in this function call on the super of *viewDidLoad* function. Then set the *city* to be an empty string, then set *originMarker* to be the `GMSMarker` class, the reason it was first set as optionals is so it will be only defined when this view did load. Then set *destinationMarker* to be the `GMSMarker` class, then set the delegate to *timeChanged* textfield to be itself. Then set up everything that will be needed for location when the view is loaded in, so set *locationManger* to be `CLLocationManger` class. Then set the *locationManager* delegate to itself, then set *desiredAccuracy* (belonging to the *locationManager*) to be *kCLLocationAccuracyBest* variable (which is a global variable in the library CoreLocation). Then call on the function *requestWhenInUseAuthorization* belonging to the *locationManager*, and then call on the function *startUpdatingLocation* also belonging to the *locationManger*.
Then define the inbuilt function *viewDidAppear* (which takes in a Boolean), then in this function call the super of the *viewDidAppear* function and put in true as the input. Now as the view is appearing the *googleMapView* can be defined, so set the *googleMapView* to be `GMSMapView` class and make it takes in a frame, where this frame is *mapView* (which is from the storyboard `RoutesViewController`). Then it is time to fix so that the location dot for the current location as well as the button. So set the *isMyLocationEnabled*

(belonging to the *googleMapView*) to be true and set the *myLocationButton* (belonging to the setting of *googleMapView*) to be true. Then add a subview to the view of *googleMapView*, do this by using the *addSubview* method belonging to the view and make it takes in *googleMapView*. Then call on three functions that will be defined later, this function is *createMarkersForRoute, drawRoute* and *calculateTotalDistanceAndDuration*. All of them will takes in inputs, *createMarkersForRoute* will takes in a Boolean called i*sCurrent*, as an input here put in false. *DrawRoute* and *calculateTotalDistanceAndDuration* will takes in a string called *origin*. Set *origin* to be a string and inside this string write in *latitude* and *longitude* to *originCoordinate* and use backslash opening parenthesis and closing parenthesis to get the value and have a comma between them. Then put *origin* in as input in both *drawRoute* and *calculateTotalDistanceAndDuration*.
Then it is time to fix the rest for location so define the inbuilt function *locationManger (didFailWithError error)* and in this function the same alert will be created as in `MapsViewController` version of this function. The alert will also have a close action and will be presented if this function gets called, which it only will if the location failed. Then define the function *locationManger (didUpdateLocations locations),* and in here set *currentLocation* to be locations (which is an array of the type `CLLocation`) last property. This is the last element in the array, which will be the user's current position. Also call on the function *createMarkersForRoute (put in true in here as input)*, *drawRoute* and *calculateTotalDistanceAndDuration* and for the input string create a constant called *current* and set it to be a string and inside the string put in *currentLocation* coordinate *latitude* and *longitude* and use backslash and opening parenthesis and closing parenthesis to get the value on both and have a comma between them.

4.5.2 Setting up the map

Create the function called *createMarkersForRoute* and this function will take in a Boolean called *iscurrent* and will not return anything. Then inside the function create a constant called *latitude* and set it to be the *latitude* of

coordinate to *currentLocation*. Then create a constant called *longitude* and set it to be *longitude* of coordinate of *currentLocation*. Then set *currentCoordinate* to be *CLLocationCoordinate2D* which takes in *latitude* and *longitude* so put in the newly created l*atitude* and *longitude*. Then create a constant called camera and set it to be GMSCameraPosition class´s *camera* function (which takes in CLLoctionCoordinate2D and a *zoom*). For the *CLLocationCoordinate2D* put in *currentCoordinate* and for the zoom put in 9.0, then set *googleMapView* camera to be the newly created camera. Then call on a function that will be defined later called *reverseGeoCode*, this function will take in these constants' *latitude, longitude, marker* and *startOrEnd*. Put in *originCoordinate latitude* and *longitude* for *latitude* and *longitude*, for *marker* put in *originMarker* and for *startOrEnd* put in true. Then call this function again but now with *destinationCoordinate latitude* and *longitude*, *destinationMarker* and false.
Then create the function called *reverseGeoCode* and it will take in a double called *latitude*, another double called *longitude*, a GMSMarker called *marker* and a Boolean called *startOrEnd*. Then inside this function call on the function *reversGeocodeCoordinate* belonging to the GMSGeocoder class (use the *geocoder* to access it). This function takes in a *CLLocationCoordinate2D* (put in the constants *latitude* and *longitude*) and completionHandler. The completionHandler is there to handle the callback which is invoked with the reverse geocoding result. So, for that reason after the curl bracket write *response, error* in, and click enter between curl bracket and *response*. Then in the callback check if error is nil, if it is not handling the error by creating an alert with a close action and then present the alert. If it is then set the delegate to *googleMapView* to itself, then create a constant called *lines* and set it to be *lines* (as optionals) to *result* and set it as an array that takes in string. Then set the *city* to be *result locality* property, then set *marker* position property to be *CLLocationCoordinate2D* with the *latitude* and *longitude* taken in by the function. Then set *marker* snippet property to be lines *joined* function with separator string /n. Then set *marker title* property to be *result locality*, make sure this is not nil by using two question marks (this is the way of checking a value is nil without an if statement), if it is it will return a string that says

"Unavailable". While if it is not nil the string of text that is in locality will be shown instead on the marker title. Then check if *startOrEnd* is true if it is then set *originAddress* to be *lines joined* by separator string and inside the string write /n so it will separate at every new line. Then set the *marker icon* property to be *markerImage* function belonging to `GMSMarker`, this function takes in `UIColor` so put in blue. Then if *startOrEnd* is false set *destinationAddress* just the same way as *originAddress*, then set marker icon to purple by using `GMSMarker` *markerImage* function. Then outside the if and else statements put still in the callback set *marker map* property to be the *googleMapView*.

4.5.3 Getting the distance and duration about route and displaying it in the label

Then define the function *calculateTotalDistanceAndDuration* and make it takes in a string called *origin*. Then inside this function set *destination* to be a string and, in this string, write in *destinationCoordinate latitude* and *longitude* by using backslash opening parenthesis and closing parenthesis on both and comma between them. Then create a constant called *defaults* and set it to be the *standard* property belonging to the class `UserDefualts`. Then call on a function that will be defined later called *getStoredTime*, this function will not take in anything or return anything. Then create a constant called *travel* and set it to be `ViewController` (use the *prevVC* to access it) travelMode. Then create a variable called *tempTravelString* and set it to be an empty string, then create a switch statement by using *travel rawValue*. This switch statement will have three cases and one default (there must always be a default with switch statement). One of the cases will be *TravelModes walking rawValue*, in this case set t*empTravelString* to be the string "walking", then set *storedMode* to be *tempTravelString*. The second case will be *TravelModes bicycling rawValue* and in this case set the *tempTravelString* to be the string "bicycling", then set *storedMode* to be *tempTravelString*. The third case will be *TravelModes driving rawValue* and in this case set the *tempTravelString* to be the string "driving", then set *storedMode* to be *tempTravelString*. Then in the default case check if *defaults string* function (that takes in a string called *forKey*, in this string write in StoredMode) is not nil. If this is true, then set the *tempTravelString* to be *defaults* *string* function with the same input as in the if statement. This will get the last stored mode that the user used. In the else statement set the *tempTravelString* to be "walking", this will nudge the user to walk more. Then after the switch statement create a constant called *URL* and set it to be this string: "https://maps.googleapis.com/maps/api/distancematrix/json?units=metric&origins=\(origin)&destinations=\(destination)&mode=\(tempTravelString)&key=api_key". This is an URL that will access Google Maps API Distance Matrix, which will help with finding the distance for the route. This API has three parameters that are required for using it, these are *origin*, *destination,* and a *key* (this is the API key that one gets through the Google Cloud Platform). There are also a lot of optional parameters one could use for this API, but only two will be used in this app. These parameters are *units* (which will be which unit system one wants to use for the app) and *mode* (which is the mode that was used for traveling). Then continuing in the function, call on *responseJSON* (which takes in a completion handler) belonging to *request* (which takes in URL) belonging to the Alamofire library (use AF to access it). For the completion handler create a callback with *Response*, then in this callback check if the case success (belonging to the response enumeration, which belongs to Alamofire) is *result* property belonging to *Response*. Then create a do statement with a catch statement if it fails. This is because it needs to try to unpack the json data, so the first thing inside the do statement needs to be create a constant called *json* and set it to be try *JSON* (which takes in data, and this data will be *Response data*, make it optionals). Then counting in the do statement after the try, create a constant called *rows* and set it to be an *arrayValue* (this is from the SWiftyJSON library) of the index string rows in the array json. The *arrayValue* will access the value of *rows* since it will be optionals. Then create a for loop with a constant *i* and set it to be 0 and set it to iterate as long as *i* is less then count of *rows* (which is how big the array is) and let *i* increment with each iteration. Then in the for loop create a constant called *row* and set it to be the index *i* in the array *rows*. Then create a constant called *elements* and set it to be *arrayValue* of the index string *elements* in the array *row*. Then create a for loop with *j* and set it start at 0 and let it increment with each iteration and make the for loop go as long as *j* is less then count of the array *elements*. In the for loop create a constant called *element* and set it to be the *elements* array with the index *j*. Then create a constant called *duration* and set it to be a dictionary of the index string duration in the array *element*. Then check if *timeChanged* is true if it is then set *totalDuration* to be *changedDuration*. This will set *totalDuration* to be the stored time. If it is false then set *totalDuration* to be a *stringValue* of the string index text of the array *duration*, let both *duration* and the index be optionals. *StringValue* belongs to SwitfyJSON and will access the actual value of text. Then create a constant called *distance* and set it to be a dictionary of index string *distance* of the array *element*. Then set *totalDistance* to be the *stringValue* of the string index text of the array *distance* and set the array and the index to be optionals. Then set info text to be *totalDistance* plus a string with space and a comma plus *totalDuration*. This is everything that is needed for accessing the distance and duration from Google on the user created route.

4.5.4 Setting up the function for getting the stored time the user put in last time

Create the function called `getStoredTime`, in this function create a constant called *defaults* and set it to be the *standard* property belonging to the class `UserDefualts`. Then create a constant called *tempStoredOrigin* and set it to be *defaults* `array` function, that takes in a string called *forKey* and in this string write in StoredOrigin. Then create a constant called *tempStoredDestination* and set it to be *defaults* `array` function, that takes in a string called *forKey* and in this string write in StoredDestination. Then create a constant called *tempStoredTime* and set it to be *defaults* `array` function, that takes in a string called *forKey* and in this string write in StoredTime. Then check if *tempStoredOrigin* is not nil and *tempStoredDestination* is not nil and *tempStoredTime* is not nil. This will make sure that it will get the *storedTime* when all the stored variables are there, since the *storedTime* will need to be with a corresponding *origin* and *destination*. In the if statement create a for loop with *i* and set it to start at 0 and iterate until it reaches the count to the optionals of *tempStoredOrigin*. Then in this for loop create another for loop with *j* and set it to start at 0 and iterate until it reaches the count to the optionals of *tempStoredDestination*. Then in this second for loop create a third for loop with *k* and set it to start at 0 and iterate until it reaches the count to the optionals of *tempStoredTime*. Then in the third for loop check if the optionals of *tempStoredOrigin* at index *i* (typecast as a String) is equal to the string *origin* and the optionals of *tempStoredDestination* at index *j* (typecast as String) is equal to the string *destination*. In this if statement set *changedDuration* to *tempStoredTime* array at index *k* (typecast as a String), then set *timeChanged* to be true.

4.5.5 Setting up the function for drawing up the route

Then create the function *drawRoute*, this function will take in a string called Origin and it will not return anything. In this function create a constant called *destination* to be string and inside the string write in the value of *destinationCoordinate latitude* and *longitude*. To get the value use backslash opening parenthesis and closing parenthesis on both and use comma between them to. Then create a constant called *travel* and set it to be `ViewController´s` travelMode (use *prevVC* to access it since it is an instance of the `ViewController` class). Then create a variable called *tempTravelString* and set it to be an empty string. Then create a switch statement with *travel rawValue*, this switch statement will have three cases and one default. The first case will be *TravelModes* case *walking rawValue*, in this case set *tempTravelString* to be the string walking. The second case will be *TravelModes* case *bicycling rawValue*, in this case set the *tempTravelString* to be the string bicycling. The third case will be *TravelModes* case *driving rawValue*, in this case set the *tempTravelString* to be the string driving. Then in the default set the *tempTravelString* to be the string walking. Then the switch statement is done now create a constant called *URL* and set it to be string and inside this string put this URL: https://maps.googleapis.com/maps/api/directions/json?origin=\(origin)&destination=\(destination)&mode=\(tempTravelString)&alternatives=true&key=api_key. This URL will access the API directions belonging to Google Maps, this API have three parameters that are required for using it, these are *origin,* *destination,* and *key* (which is an API key one gets through Google´s Cloud platform). There are also some optional parameters one could use here, for this app only *mode* (mode that are travel with) and *alternatives* (a Boolean, that is either true or false, when true let the route have more than one route) are used. Then continuing in the function, call on *responseJSON* (which takes in a completion handler) belonging to *request* (which takes in URL) belonging to the Alamofire library (use AF to access it). For the completion handler create a callback with *Response*, then in this callback check if the case success (belonging to the *response* enumeration, which belongs to Alamofire) is *result* property belonging to *Response*. Then create a do statement with a catch statement if it fails. This is because it needs to try to unpack the *JSON* data, so the first thing inside the do statement need to be create a constant called *json* and set it to be try *JSON*(which takes in data, and this data will be *Response data*, make it optionals). Then after the try, create a constant called *routes* and set it to be an *arrayValue* (this is from the SwiftyJSON library) of the index string routes in the array *json*. The *arrayValue* will access the value of *routes* since it will be optionals. Then create a for loop with a constant *i* and set it to be 0 and set it to iterate as long as *i* is less then count of *routes* (which is how big the array is) and let *i* increment with each iteration. Then in the for loop create a constant called *route* and set it to be the index *i* in the array *routes*. Then create a constant called *routeOverviewPolyline* and set it to be a dictionary of index string *overview_polyline* of the array *route*. Then create a constant called *points* and set it to be *stringValue* of index string *points* of the array *route*, set both the array and the index as optionals. Then create a constant called *path* and set it to be the `GMSPath` *init* function which takes in *fromEncodedPath* (which is the type `GMSPath`), so put in optionals of *points* in here. Then create a constant called *polyline* and set it to be `GMSPolyline` *init* function which takes in a path (which is the type GMSPath) so put in a newly created constant *path*. Then set *polyline isTabbable* property to be true. Then check if i is 0 (this if statement will have an else), which means the for loop is at the first iteration. If it is then set *polyline strokeColor* property to blue, then set polyline *strokeWidth* property to be five. Then set *transferPolyline* to be *points*, then check if *googleMapView* is not nil. If it is not, then create a constant called *bounds* and set it to be *GMSCoordinateBounds* which takes in a *GMSPath* so put in path as optionals. Then call on *animate* function belonging to *googleMapView* (set this to optionals), the function will take in `GMSCameraUpdate` class function *fit* which takes in *bounds* and a constant called *withPadding* (which is a float) set this to be fifty point zero. Then in the else to the if check for i is 0, set *polyline strokeColor* property to systemBlue color (which is light blue). Then set *polyline strokeWidth* property to be four. Then outside the else statement but still in the for loop call on the inbuilt function *append* (which will take in polyline) on the array polylineArray. Then set *polyline map* property to be the *googleMapView*.

4.5.6 Setting up so the user can click the other polylines and make it more visible than the others

Then define the inbuilt function *mapView (*which takes in a *mapView* of the type `GMSMapView` and *did Tap overlay* of the type GMSOverLay*)*. Then inside the function call on *createMarkersForRoute* (in here put in true), then create a constant called *current* and set it to be a string and inside the string write the value of *latitude* and *longitude* belonging to *coordinate* belonging to *currentLocation*, use comma between them and use backslashes opening parenthesis closing parentheses to get the value in the string. Then check if by using the *isKind* function on overlay, in the *isKind* function put in a version of `GMSPolyline` which is itself. This will check if the user taped a polyline on the map. Then in the if statement call on the *clear* function belonging to *mapView* which will clear the map for markers, polyline, and ground overlays. Then call on the function *createMarkersForRoute*. Then create a for loop with i starting at 0 and get plussed one up with iteration until it is no longer a smaller number than polylineArray´s count. Then in the for-loop check if the overlay is equal to index i in polylineArray, if it is not then it goes to the else statement. In the else statement call on *calculateTotalDistanceAndDuration* and put in *current* as input, then set *strokeColor* property to the *polylineArray* (use *i* as index) to be the color systemBlue. Then set *strokeWidth* property to the *polylineArray* with *i* as index to be four, then set *map* property to the *polylineArray* with *i* as index to be *mapView*. Then in the if statement check if *strokeColor* property to *polylineArray* (use *i* as index) is blue. If it is then set *map* property to the *polylineArray* using *i* as the index to be *mapView*. If it is not then call on *calculateTotalDistanceAndDuration* and put in *current* as input, then set *strokeColor* property to the *polylineArray* (use *i* as index) to be blue. Then set *strokeWidth* property to the *polylineArray* with i as index to be five, then set *map* property to the *polylineArray* with *i* as index to be *mapView*. Then set *transferPolyline* to be the *encodedPath* function belonging to *path* (set this as optionals) property belonging to the *polylineArray* (with using i as index, since this will access the current polyline). Then check if *googleMapView* is not nil, if it is not then creating a constant called *bounds* and set it to be `GMSCoordinateBounds` class with the input *path* property (as an optional) to *polylineArray* (with using i as the index). Then call on *animate* function belonging to *googleMapView* (as optionals) with the input *fit* function belonging to `GMSCameraUpdate`, with the input *bounds* and *withPadding*, set this to be fifty point zero.

4.5.7 Managing the textfields in the extension

Then outside the class create an extension to `RoutesViewController` that takes in the *UITextFieldDelegate* protocol. Then in this extension define the inbuilt function *textFieldShouldReturn*, which takes in an `UITextfield` class called *textField* and returns a Boolean. Inside this function check if the *textField text* property is not nil. If it is not then, call the method called *insert* on the array *storedTime*, this method takes in an element (to be added into the array) and a position called *at*. For the element write in *textField text* property as optionals, for the *position* at write in the number zero. Do the same (except with different elements) for the arrays *storedOrigin* (for the element write in origin) and *storedDestination* (for the element write in *destination*). Then create a constant called *defaults* and set it to be the *standard* property belonging to the class `UserDefualts`. Then use the method *set* belonging to *defaults* to store the three arrays *storedTime*, *storedOrigin* and *storedDestination* and store the variable *storedMode*. This method will take in any value and a string called *forKey*. For the first call on this method put in for the value the array *storedTime* and in the string called *forKey* put in StoredTime. For the second call on this method for the value put in the array *storedOrigin* and in for the string put in StoredOrigin. Then for the third call on this method for the value put in *storedDestination* and for the string put in StoredDestination. Then for the last call on this method put in for the value put in the string *storedMode* and for the string put in StoredMode. Then outside the if statement and after it call *resignFirstResponder* on the *textField* and then return true. This will make the user be able to go out of the textfield, even if they have not written anything. This storing will not take that much space so it can be saved on the phone, and it will not need to be encrypted since it does not need to be stored safely like a password. This data will still be safe here since it will be stored in memory on the phone and can only be accessed through the phone and not the internet.

4.6 Libraries and programming language

This app was implemented in the IDE Xcode and the programming language used to implement this app was primarily Swift, but some Ruby was also used. Ruby was used to get other libraries that does not belong to Swift or Ruby by using Ruby´s library CocoaPods. These other libraries where Google Maps, Google Place, Alamofire and SwiftyJSON and were installed by using Cocoapods as pods in the podfile. These libraries were then imported into the code in Xcode. There were some of Swift´s own libraries that were also used for implementing this app, these being UIKIT and CoreLocation.

4.7 Problems

Thing were not intuitive during the implementation of this app. There were a lot of problems along the way, and they were not always so easy to solve.

4.7.1 Problems with Google´s API

The first issue was getting everything of the API from Google to work right away, except for Google Maps. The issue was that one needs to have an account that has billing to use the rest of the APIs at that time, but for Google Maps one only need an API key through the Google platform and have the API enabled (this is not the case anymore at the time of writing this thesis in May in 2021). The cases are that it has not always been this way with the Google APIs, there was a change like three years ago (in 2017) where one had to have a billed account to use the rest of the APIs. This information was not easy to find out of this at first, and this was especially a problem with Google Places, and it would not just work. A lot of the tutorials about Google Places didn´t mention this, since they were often more than three years old, and Google has only recently started to inform about this on their website for Google Places iOS. There was also a problem with getting Maps to work properly after updating Xcode to version 11.5 and the solution to this problem was a URL schematic that was needed in info.plist, this is not required anymore (in 2021). But at the time it was required to be able to have a functioning map, but it the author thought that there was something wrong with the version of Xcode. Since the problem started when running the app after a new version of Xcode had been installed, therefor an older version of Xcode 11.0 was downloaded and installed. This fix didn't work at all, and the author had to consider abandoning Google Maps and use iOS maps instead. First the reason for why Google Maps were not working were investigated before abandoning it totally. One solution was found on Google Maps iOS page and that was that a schematic had to be used to make it work. Now Google Maps only works if one has a billed account and Google informs about this on their webpage, the URL schematic can be used to use Google Maps without API key and a billed account. This will only open their version of Google Maps and can´t be personalized to the app that someone makes. As one can see Google changes now and then what their APIs will need to be run, so the best option when something suddenly doesn't work is to check

their documentation and see if there is any information there about it and then check Stack Overflow.
Another problem was connected to the API keys from the Google Cloud Platform, they would just not work in the URLs. The solution to this were that it was a need to have a wait time of 10 minutes after making them before they can be used.

4.7.2 Problem´s with CocoaPods

Another problem that came up was with CocoaPods, it would not install the pods that were needed (like Google Maps). To try and solve this the Mac was reset to a previous version, in hope that this fixes the problem with CocoaPods. Since it suddenly did not work as it should after some update from apple on Mac OS. It did not fix the problem, so other solutions to this problem were looked into. To fix this problem start by installing the Xcode Command Line tools in the terminal on the Mac. What was written to install it was this:

```
xcode-select –install
```

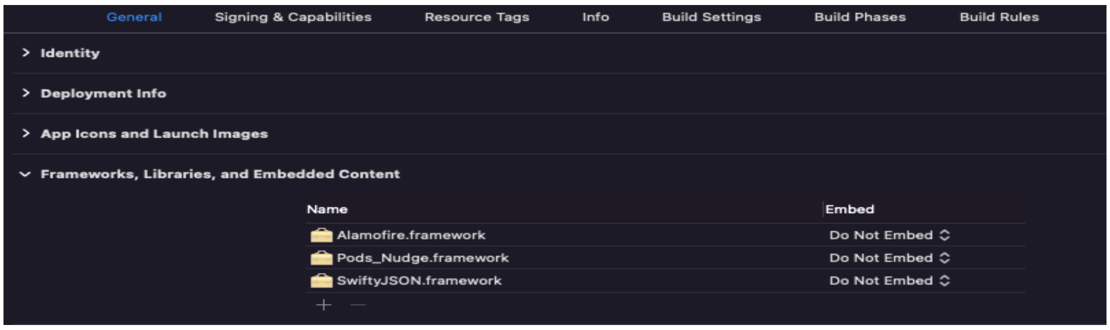
After installing the command line tools make sure the Xcodebuild worked as it should, to do that this needed to be written in the terminal:

```
sudo xcodebuild -license accept
```

This would install the Xcodebuild needed for CocoaPods. Then after that the pod setup needs to be written in the terminal to setup the CocoaPods environment. After all this it worked to install the pods through CocoaPods.

4.7.3 Problem´s with Alamofire and SwiftyJSON

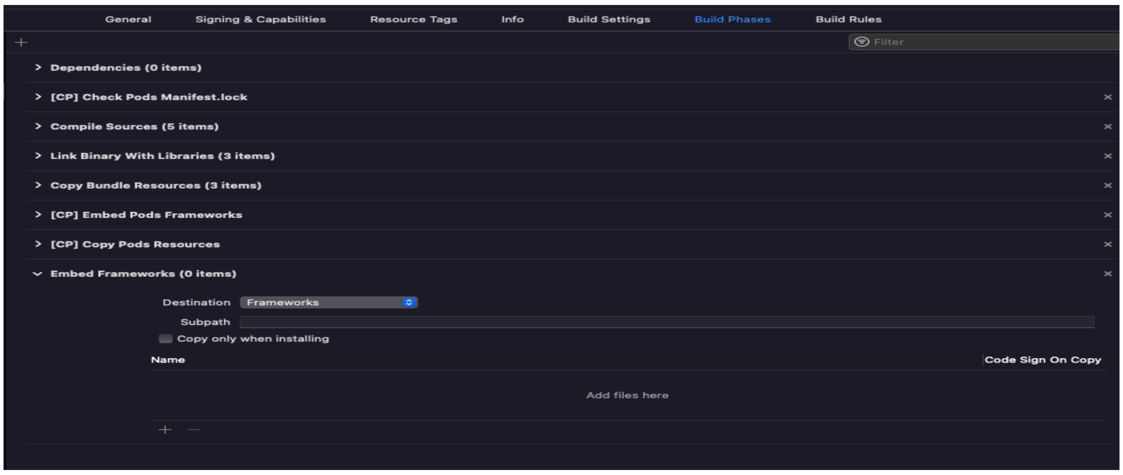
There has been some problem in using Alamofire and SwiftyJSON, and they are were usefully to use with the implementation of the code. Alamofire and SwiftyJSON are pods that are available through Coacopods. The problem was with importing the Alamofire and the SwiftyJSON library in the code. The author figured out how to solve this problem and it is now working. It was solved through installing the pods again, to do this one needed to write pod install in the terminal in the folder to the project and click enter and let it resolve. Then after that is done write in the terminal pod update click enter then let it resolve. Open xcworkspace, this will open the Xcode project with the pod files. Then in Xcode go to the project settings, then add both Alamofire and SwiftyJSON in the project settings general under frameworks, libraries and embedded content. If they are already there, then remove them first and then add them again Figure 26.



Name	Embed
Alamofire.framework	Do Not Embed
Pods_Nudge.framework	Do Not Embed
SwiftyJSON.framework	Do Not Embed

Figure 26 How the project setting looks like after the Alamofire and SwiftyJSON framework is added.

Remove Alamofire and SwiftyJSON frameworks from build phase embedded framework Figure 27.



Dependencies (0 items)
[CP] Check Pods Manifest.lock
Compile Sources (5 items)
Link Binary With Libraries (3 items)
Copy Bundle Resources (3 items)
[CP] Embed Pods Frameworks
[CP] Copy Pods Resources
Embed Frameworks (0 items)
Destination: Frameworks
Subpath
Copy only when installing
Name | Code Sign On Copy
Add files here

Figure 27 How the project settings for build phase embed frameworks look after Alamofire and SwiftyJSON have been removed.

This is a problem that keeps showing up now and then, and this might need to be fixed by doing the solution that is mentation above before the project is built and run. Why this keeps happening is not something that was figured out, it might be that Xcode cleans the build folder and removes the reference to Alamofire and SwiftyJSON. The solutions do make the error go away and make the project be built and run again.

4.8 Error Handling

The way that errors were handled in this app was by just creating an alert that would show up for the user that had information about the error and what was wrong. But this is not a good enough way of handling error if this app is ever going to be put on the App Store and be commercialized. There should be more done here, like having a support feature. This support can be a chat, or through email or through phone. If there is no support the user can´t get help when the error happens, and it will make them not use the app or use it less. The reason why it is just an alert that show up is because this was an easy way to debug when testing the app and an easy way to implement error handling for now.

4.9 The Choice between Google Maps and Apple´s own maps

There were two maps that where considered for implementation in this app. The options were Google Maps and iOS own map called MapKit. Since this app was developed as an iOS app the MapKit would have been easier to use and implementing when developing the app. On the other hand, MapKits is not that good at seeing trails like the Lysløypa in Tromsø. Which is a trail one can use to bike and walk across Tromsø island. Tromsø is a town that consists of the mainland and two islands, where one is called Kvaløya and the other refers to the island or the Tromsø island. Most people live or work on Tromsø island, it is where the city center is and where most businesses are as well as the University of Tromsø and other public institutions. This is why it really important to have a map that can see the trails, since this can be the fastest and easiest way to take when walking or biking to places on the island. Google Maps has the information about the trails and are able to create route through the trails, and even though it is less simple to use Google Maps when making an iOS app it was better to use more time on the map implementation than having a map that will not tell its users of routes that are faster and less known.

5 Discussion

In the start of this thesis there were set some goals that were going to be implemented into the app to nudge people towards green transportation. The author is now going to talk about if all of these where achieved, what was learned from them and what the author have shown through these goals.

The first goal is:

> *The app will need to be able to let users select an origin and a destination for their travel. The origin and destination should be possible to identify by having the user either writing in the address or choosing a location on a map. In its first version the app will be limited to the city Tromsø. The map will have a possibility to see establishment as well as the addresses for locations when zooming in. To help the users when writing in addresses or location an autocomplete feature will be implemented. The autocomplete should come with suggestions based on the first letter the user writes and update for each successive one*

The first goal was achieved the test user where able to select an origin and a destination for their travel, this both through the map and an autocomplete feature. From the information from the testers the author was able to show that they can make a simple design that is not boring but not too distractive.

The second goal is:

> *To implement a way to find the users location so the app can easily find the starting point. The user's current location will have a marker that displays the address of the location.*

The second goal was achieved, the app is able to finds the user location and displaying the address. The app need's location permission to be granted from the user as well as having the GPS for the phone to be on while the app is being used.

The third goal is:

> *Offer a way for the user to select their preferred transportation mode. A direction feature must take all the inputs into consideration and show possible green travel routes. This feature will use the information to calculate the*

distance and duration of the travel routes. It will also need real-time updates while traveling to show the remaining time, and distance and offer new possible routes from the current location.

The third goal was achieved, the user was able to select their preferred transport mode. The route was calculated with the user inputs and offered updates in both remaining time and distance. Through this goal it was learned that everything can be fixed with some time and patient. Through this the author was able to show a simple design with the help of Google.

The fourth goal is:

There will be a possible to click on an establishment to get more information about it. This will tell the user some things about the establishment that will make them use the app. Since they can see name, rating and if it is open now and this will give them enough information to see if they want to go there and then they can choose it as a place to travel to. Also, there will be a possibility to get personalized walking/biking time.

The fourth goal was achieved, the user was able to get information about establishment when clicking on them and put in personalized walking/biking time. Through here it was learned how to work with time and inputs from users. Here the author showed they could make a simple design that the user could interact with.

The fifth goal is:

To have a possible integration with calendars, such as Apple Calendar, might also be implemented with a notification feature so the users can easily get to their appointments.

The fifth goal was achieved, it can get events from Apple´s calendar. The app will send a notification 2 hours before the event and put the location for the event into the destination textfield. The notification is set to 2 hour and the reason why is so that the user will have some time to make themselves ready and then to travel to destination. This is because the notification would not show up if it was in findDestinationCoordinate function and because of this the duration could not be added to the interval the notification will show up for the user. The findDestinationCoordinate function is important for the duration since the duration of the route must be found between current location and the events location and it will need the coordinate for this. Before this function is called events location will be nil and therefor duration cannot be found. The Author don´t know exactly why the notification don´t want to show up in this function, it might be that it gets called too late. Therefore, it set to 2 hours, if the user has 1 hour travel time, they will then have 1 hour to make themself ready to leave for the event.

The sixth goal is

> *The app will also have the possibility to use the data from the GPS trackers in the buses in Tromsø. This will update the travel information about where the buses are at the moment and the user can see this information and plan around it.*

The sixth goal was not achieved and through this it was learned that implementing buses into an app is not easy and will take some time to figure out how to implement.

The last seventh goal is:

> *To integrate some weather data (i.e., rain, wind, sun, and snow) such that users may be notified of weather conditions. This integration is important since it will make it possible to show skiing as a green travel option during winter conditions*

The seventh goal was not achieved and through this it was learned that weather data is not easy to implement and that it will take some time to implement it into the app.

5.1 How the app was tested

The app was tested on an iPhone SE (2nd generation) by the author, the project was added to this iPhone. An origin (olastien 4) and destination (UiT) were provided in, this gave possible routes that could be traveled. It calculated four different routes and the route through "lysløpa" was chosen and walked. To see if there would be any problem while walking in the "lysløpa" since it will be important as a walking and biking route. The author thought the app was easy to use, and all the features functioned as intended. There was a need for other people to test this app as well, to see if it was able to nudge them toward choosing green transportation. Due to the covid-19 restrictions, voluntary test subjects cuold not easily be found since this app needed to be tested in person with the iPhone. This app was tested with three participants that were given the iPhone and told what the app was supposed to do by the author. Then these participants tested all the features and was asked questions about the different features and the design of the app. All the answers and feedback from the three participants are in Table 1. The question the where asked where what do you think of design of the app, what do you think about color, and how is it interacting with the different features. The iPhone had to be disinfected before giving it to others and when getting it back due to the covid-19 pandemic.

5.2 Result

The author had an easy time using the app and it gave three different routes to walk from olastien 4 to UiT. The time for the route between prestvannet student housing to UiT were about one hour instead of the 45 min the app stated the route would require. It also updated the location to the current position and the remaining time left to walk when the location button was clicked. When other people got to test the app, the feedback that was focused on how it was to use and interact with the app. The testers were also asked if this app would make the choice of green transportation easier. Some of the feedbacks from the testers of the app is presented in Table 1. The questions were asked in Norwegian and the answers were given in Norwegian. Both question and answers have been translated to English.

From tester one:	From tester two:	From tester three
Simple Design	The textfield in routes should be white and the text color should be black	Thought it was an interesting app
Not disturbing	The name of the app should be another color to display this is the name, another color blue.	Different time on different polylines, since they might be factor like uphill and downhill that play an effect on the time used on route.
The button for changing the transportation mode needs to give feedback on what is chosen as a transport mode	The search button should be another color or moved to where the other buttons are, the show map button can´t not just be a map	
That the dismiss button should be renamed back or return	The textfield should maybe remove information when going	

	back from the RoutesViewController	
Text needs to be bigger for weaker sight people		
Maybe have experience point, and more point for walking		

Table 1 show the feedback from testers

Based on the received feedback much future work remains for the app so that it can nudge better to make the choice of green transportation. It will also need an info banner about what it does since this does not come across in the app by itself, and the author had to explain what it did to the testers.

5.3 Why Tromsø?

There are four reasons for selecting Tromsø as the base town for the Nudge app. The first one is that it is a town that has very many walking and biking routes that many people that live here don't know about to full extent. The second reason is that since Tromsø only has walking, biking and busses as a green choice it would not be that much to implement, for example a city like Oslo has also trains and trams that would need to be looked at during the design of the app. The third reason is that the creator of this app happen to live in Tromsø, and knowledge of walking and biking routes is needed to take the best decision during the design of the app, for example recognizing that Lysløypa is really important for walking and biking routes (and also skiing in the winter). The fourth reason is that the app had to be tested someway on an iPhone, and since the app works with a given localization it had to be Tromsø since its creator lived here and could test it here. So, for these reasons Tromsø was selected as the town for this app, and why the *filter* for the autocomplete was restricted to Tromsø.

5.4 The choice of colors for the app

When designing the app, the question of what colors to use came up and what color would be best to use. When it comes to products and branding the color of it will have something to say whether it will be bought [34]. Singh goes on to say that [34] «people make up their minds within 90 seconds of their initial interactions with either people or products». He also says that [34] «about 62-90 percent of the assessment is based on colors alone». The choice was made to go for the color blue in the app, and this is because it there is a lot of apps that use this color (Facebook, twitter, PayPal, even the traveling planner called Troms reise that is used in Tromsø) and it seem to have a positive psychotically effect on people and are not too distracting.

5.5 Should nudges be uses by both business and the government and why do some want to reject nudges as a concept?

There is a need to discuss some things that make people want to reject the concept of nudge being used on people by both businesses and governments. One of the things that people reject nudges based on is the freedom of choice. Another is that they reject it based on the nudge being used unethically on people. These rejections will now be discussed separately, and what can be done about these rejections that some people have, so that the nudges can still be used. There are also some other concerns that people might have that also will be discussed.

5.5.1 The rejection of nudges on the bases of freedom of choice

Rejection of paternalism and any nudge will usually come from those that favor the freedom of choice [1]. They reject them on the basis that nudge breaks the freedom of choices. These people believe that people can choose for themself the best option and want there to be as many choices as possible. This is mostly because most of them are economists that believe that people are rational beings that can always make the best option (they say they take the best choice for themself therefore others must do so too). Others just want to have their freedom to do what they want and don´t know what freedom entails. Economists are wrong, people don´t always do what is in the best interest for themself, and too many options can make people too overwhelmed and make them choose nothing or the option that is easiest. It is also the fact that people usually don´t have all the information they need to make the best decision for themselves.
There should always be an option with the nudge to not be a part of it. If this is not a choice in a nudge, then that nudge can be rejected on the basis that it

breaks the right to freedom of choice. The app design in this thesis does not break any right to freedom of choice, even if localization must be on to use the app. This is because it will be their choice if they install this app and use it, they can always uninstall it.

5.5.2 The ethical dilemma of nudging

Some people will try to reject nudges based on nudge being unethical. This will have something to say for this app since it uses nudge and if nudge is unethical then the app is unethical. Therefore, there must be a discussion about nudge being ethical or unethical to figured out if the app is unethical. There are four commitment that are pointed to when it comes to concern about nudging: welfare, autonomy, dignity, and self-government [35]. These are the commitments one should keep in mind when considering whether the nudge is ethical or not. Some of these commitments will be examined more thoroughly, to determine whether a nudge is ethical or not. Welfare in the perspective of nudging is that nudges are the best for the welfare for people in general. Will certain nudges help people in making optimal choices for their health and happiness in the long run? This welfare commitment is connected to what nudges are supposed to be about and are connected to the other commitments to see whether the nudge is ethical or not.
For the commitment autonomy there are some who believe that nudge have no autonomy for the people that using the nudge. For example, that nudge have no autonomy and it make people having less control over their actions [36]. There are other for example Sunstein that says that if there is going be autonomy there have to be an informed choice and that lack of information is when the nudge lack autonomy [35]. This fit with the freedom of information that is one of the rights under freedom, this right will also have restriction to protect people´s sensitive information to protect their life and their privacy. There is some information that can´t be told because they are credit card numbers, security numbers, passwords, etc. this is the type of information that need to be secure and need to be withheld to protect people´s private life and economy from other´s that would use it to their advantage or unlawful means. There is other information that needs to be withheld to protect people´s life.

This can be information about criminal records, here it can be sealed records or witness protection. Types of information may be sensitive because it about children, and will be sealed to protect them, since this is not something they can do for themselves.
There are concerns about nudges that can be manipulative, since they change the behavior to people and that in this way nudges are not ethical. Manipulation can be tied to the commitment autonomy, since it is manipulation that violates the autonomy that any person has [37].
About manipulation and nudge it is said that something that alter someone´s behavior is not necessarily manipulative [35]. If you warn someone about a danger or remind someone of a meeting you are not manipulating them [35]. Some other concerns when it comes to nudge and manipulation is that [38] «nudge seeks to exploit imperfections in human judgement and to this extent it is manipulative». Sunstein say that [35] «an action might generally be counted as manipulative if it lacks transparency». For an action to be manipulative it must attempt to influence people so that it undermines their choice over that action [35].
Dignity can be tied in with welfare in way where people can have a loss in welfare because of dignity. For example, if they feel that have been humiliated or treated disrespectfully [35]. This insult on dignity is a not something that is tied to one person's feeling about it, and it should not be any part of the nudge unless it has extremely good reason for being there. [35] There can be an objection to some nudges but not nudges generally [35]. Some nudges can be unethical if they are not looking out for the welfare of people and tries to manipulate them or break any of the right that comes with freedom or try to restrict them (of course without protecting any other´s rights, since they sometimes need to be restricted to protect other people´s freedom´s right´s). The author will say that any nudge that does this is not a nudge by the concept and is unethical. This is because the concept of nudges is all about being for the welfare of people, and they are not that if they are trying to fool any people to take a choice that is not in their best interest. Since nudges is all about helping people to take a choice that is the best for the welfare for the people [35].

The app description, if it gets put up on the App Store, will need to declare what it does and why it does it on it is information page on the App Store. This way people will get the complete information about the app. Some might say that the app is manipulative on the bases of colors, but an app is not manipulative just because of the colors that is used in the design. This app does not use the color blue to be manipulative, it uses it because blue is a color that many people don´t find distractive and it highlights the white textfield and the places where there are informative text or buttons that does something for the user of the app.

The author believes that there can be an illusion of freedom of choice, and that it is this that people refer to when talking about the ethics of nudging. What is illusion of freedom of choice? The author definition on this is that somebody present a choice that is not actually a choice at all, it might seem like it, but it is an illusion. Substein talked about a country that says everyone has a choice to opt out (if they explicitly indicated this is not what they want) of the nudge or the default leader forever will be person x [35]. This is not a choice at all, it might seem so, it has two options, but it is not really so. Most people do not want to go from a country that is run on democratic ideas to a country that is run on authoritarian ideas, they want to be free to choose who is gone represent them in the government and that leader for this government can change every fourth year or something like that. It is an illusion of freedom of choice since this kind of choice will remove the freedom to choose for everyone in the country. This would break on the right for everyone to have freedom and the right that comes with this. There should be some rules for nudges, so that no one can make a nudge that is not really a nudge. To protect people´s freedom there should always be rules in place so it can´t be misused by any person, company, or government. Since any concept can be misused if there are not rules or laws in place for them not to be misused by anyone. One of these rules should be that nudges as a choice architecture can never do something that will infringe on the democracy or the right´s to freedom that people have. Where this right´s that should be followed should be the Universal Declaration of Human Rights, which [39] «is a milestone document in the history of human rights». It is a document that was made by representatives from different people all over the earth with different cultural backgrounds [39]. A nudge that does this should be consider a non-nudge and to be unethical and will have to be removed.

5.5.3 Other concerns about nudge

Some might ask why nudge should be used to solve problems like global climate change, health, and economy. If people feel they have a choice rather than a punishment can help to reduce the amount of greenhouse gases. Making goods or transportation utilizing green energy cheaper provides an economic incentive to change. The government should try to give people a carrot rather than the stick, which is why they should use nudges.

5.6 Why was the app made for iOS?

The programming language Swift was the best choice for programming languages for making app´s, and it is used on iOS. Xcode is easy to use as an IDE with Swift and is easy to use when designing apps. Swift is a programming language which is easy to learn and use, and Xcode was not difficult to use at all in designing the app and making it. At the time the most used programming language for making app´s in Android was Java, which is a problematic programming language because it can have memory leaks and is not that safe to use. Swift does not have these problems at all. Swift is designed to be safe, for example on Swift website [13] it is informed that Swift never lets a value be nil and that a value will need to be checked before it is doing something to avoid compile-time error. Optionals [13] «may contain nil, but Swift syntax forces you to safely deal with it using question mark to indicate to the compiler you understand the behaviours and will handle it safely». In the technical background in the section 2.2, Swift it was discussed in detail why it is safe to use and how it handles values, arrays, and memory.
In the future someone can develop this app for Android also so it can be used be more people.

5.7 Did the app nudge?

This app is only a protype that someone can build on in the future and be able to nudge people even better. This app was able to nudge, and in it is design it was thought about how the user were going to use this app. It was design to be easy to interact with, easy to use and a color that made it easy for user to see where inputs and buttons are. In the test this was achieved, since the tester thought it was easy to interact with and that the colors were not distractive. Even if there are things in the feedbacks from the tester on what can be improved on, this does not mean the app was not successful in nudging. It only means there are things that can be improved in the future in the design in the app to be able to nudge even better. The first design of big apps like this will not be without anything that can be improved. Apps come with update that improve them in the way they are design and how user interact with them. This is also the case for app that nudge the users. Look at Pokémon Go, that uses nudge to make people walk more. It has evolved a lot since it first started, all to make it better for the user of the app and nudge them in walking. For example, the app now check your speed and you need to tell the phone that you are a passenger, to make people not drive and play Pokémon Go. This is because it is dangerous to use one's phone and drive at the same time and Pokémon Go want its user and other people to be safer in traffic.

5.8 Summary

To summarize what have been talked about in this thesis, there were seven goals established in the begin of this thesis. Five of these was achieved and that the app was able to nudge people towards the choice of green transportation, which was shown in the result. This thesis has concluded that nudging can ethical if there are rules in place for nudging and that this app follow the rules and are nudging ethical. The thesis has shown designs and implementations discissions and why thing was chosen. For example, Google Maps, the color of the app and why the map is restricted to Tromsø have been things that have been disscussed in this thesis and all have been able to show why these choices where taken. This thesis has also talked about how this app is a protype and that apps always improve more after they are released. If this app is released the further work and feedbacks in result should be taken into consideration for implementation.

6 Future work

A possible improvement is to fix the calendar notification so that the notification shows up one hour plus the travel time before the calendar event. Another thing to possible consider is that the user can maybe put in the time they want to have before the travel time, so they can decide for them self how long they need to make themselves ready.
Further improvements include looking into public transportation, to provide public transportation as a mode of travel. The buses have live GPS tracking which can be used to track the busses and see when they arrive. When the user inputs their location and destination they could find bus stops near their locations and see when the bus leaves.
Another improvement will be to that user can get remaining time left on the time they actually use on the route, when they update their location on the travel. This updating of time would either have to plus or minus the difference between the time the user actually used (user´s time) and the time Google think the user will use on the route (Google´s time). Where it would be plus if the user´s time was longer than Google´s, and minus if the user´s time was shorter than Google. Also adding something so that user don´t need to take the time themselves, so they can hit a button that start time and then a button that end times and then save the time between hitting the start and end button.
The route utilized while traveling could be saved, although storage space requirements must be taken into consideration. If it takes too much space to store such data, it might have to be saved in the cloud. This cloud has to be very secure to make sure this information is not leaked, and it would need to require a login for users so that the cloud could make sure to give the right user the right information.
Implementing the weather service as mention in goals could also help users plan their day. The app could, for example, notify the user stating that it will rain today, or that it is windy. The weather can also tell a person that the winter seasons have started, that skiing is an option, and suggest viable winter routes. An implementation of this would require the skiing to be implemented for itself, since the only road to take here in Tromsø would be through Lysløypa. The calculated travel time for skiing should also change based on the snow and weather conditions.

To reduce the overall numbers of cars in use, future version of the app could suggest carpooling. By examining people in the vicinity and determining that somebody got the same destination in mind, the app could suggest grouping up to reduce overall vehicle usage. The app could also have a way to split the money used on gas for traveling, parking and/or tolls depending on the number of people that travel together.

7 Conclusion

This thesis started with a problem definition:

> *In this thesis with the help of digital nudging attempt to make a solution that will influence people to choose green transportation. This attempt to a solution will be done through designing and implementing an app.*

This problem definition was solved through making an app that would use nudge to make people to choose greener transportation and in this it was successful when looking at the feedback from testers.

There was also set some goals that where set for implementation for this app and even if not all of them are implemented. The goals that where not achieved are:

The sixth goal:

> *The app will also have the possibility to connect with the GPS trackers in the Tromsø buses. This will be used to tell the user when they are going, where they are going, how far away they are and how long they will take to be at the end position in real time.*

The last seventh goal:

> *The possibility to integrate some (like rain, wind, sun, and snow) weather data services that will show a notification about the weather. This integration will be important since it will make it possible to show skiing as a green travel option in the winter.*

Even though not all goals are implemented into this app it is able to nudge the user, and in the future the rest of the goals can be implemented into this app. This app is only a protype and where the foundation of this app is made. In the future someone can continue the work on this and get it out on the market.

If this app is implemented for Android, the things in future work and feedback from testers should be taken into consideration for implementation.

Bibliography

[1] R. H. Thaler and C. R. Sunstein, Nudge; Improving Decisions About Health, Wealth, and Happiness, New Heaven: Yale University press, 2008.

[2] C. Schubert, "Green nudged: Do they work? Are they ethical?," *Ecological Economics,* vol. 132, pp. 329-342, 2017.

[3] Commission of the European Communities, Towards a new culture for urban mobility; Green Paper COM(2007) 551 final, Brussel, Belgium: European Commission, 2007.

[4] W. K. Darkwah, B. Odum, M. Addae, D. Koomson, B. Kwakye Danso, E. Oti-Mensah, T. Asenso and B. Buanya, "Greenhouse Effect: Greenhouse Gases and Their Impact on Global Warming," *Journal of Scientific Research and Reports,* vol. 17, no. 6, pp. 1-9, 2018.

[5] J. Mamen and R. Benestad, "drivhuseffekten i Store norske leksikon," snl.no, 26 Mars 2020. [Online]. Available: https://snl.no/drivhuseffekten. [Accessed 28 June 2021].

[6] R. Pachauri and A. Reisinger, "IPCC, 2007: Climate Change 2007: Synthesis Report. Contribution of Working Groups I, II and III to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change," Geneva, Switzerland, IPCC, 2008, p. 104.

[7] M. Weinmann, C. Schneider and J. v. Brocke, "Digital Nudging," *Business & Information Systems Engineering,* vol. 58, pp. 433-436, 2016.

[8] "Cambridge Dictionary," Cambridge University Press, [Online]. Available: https://dictionary.cambridge.org/dictionary/english/freedom. [Accessed 25 April 2021].

[9] A. Andersen, R. Karlsen and W. Yu, "Green Transportation Choices with IoT and Smart Nudging," in *Handbook of Smart Cities: Software Services and Cyber Infrastructure*, Switzerland, Cham: Springer International Publishing, 2018, pp. 331-354.

[10 W.-y. Chao, "Usability Engineering Framework for Persuasive Mobile Health Apps to
] Effectively Influence Dietary Decisions of Older Adults," Purdue University Graduate School, 15-Dec-2020.

[11 C. Schneider, M. Weinmann and J. vom Brocke, "Digital nudging: Guid- ing online user
] choices through interface design," *Commun. ACM,* vol. 61, pp. 67-73, 2018.

[12 R. Karlsen and A. Andresen, "Recommendations with a Nudge," *Technologies,* vol. 7,
] no. 2, 2019.

[13 "About Swift," Apple Inc., [Online]. Available: https://swift.org/about/#swiftorg-and-
] open-source. [Accessed 12 April 2021].

[14 "Using Downloads," Apple Inc., [Online]. Available: https://swift.org/download/#using-
] downloads. [Accessed 14 April 2021].

[15 E. Rossen, "programmeringsverktøy i Store norske leksikon," snl.no, 4 August 2020.
] [Online]. Available: https://snl.no/programmeringsverktøy. [Accessed 20 April 2021].

[16 Apple Inc., [Online]. Available: https://developer.apple.com/xcode/ide/. [Accessed 20
] April 2021].

[17 E. Rossen, "API i Store norske leksikon," snl.no, 31 July 2020. [Online]. Available:
] https://snl.no/API. [Accessed 12 May 2021].

[18 H. Dvergsdal, "HTTP i Store norske leksikon," snl.no, 16 January 2021. [Online].
] Available: https://snl.no/HTTP. [Accessed 12 May 2021].

[19 M. Bartnes, "HTTPS i Store norske leksikon," snl.no, 25 November 2019. [Online].
] Available: https://snl.no/HTTPS. [Accessed 6 July 2021].

[20] "Getting started," CocoaPods, [Online]. Available: https://guides.cocoapods.org/using/getting-started.html. [Accessed 20 April 2021].

[21] "The Podfile," CocoaPods, [Online]. Available: https://guides.cocoapods.org/using/the-podfile.html. [Accessed 22 April 2021].

[22] A. S. Foundation, "Alamofire/Foundation," GitHub, [Online]. Available: https://github.com/Alamofire/Foundation. [Accessed 22 April 2021].

[23] A. S. Foundation, "Alamofire/Documentation/Usage.md," GitHub, [Online]. Available: https://github.com/Alamofire/Alamofire/blob/master/Documentation/Usage.md. [Accessed 22 April 2021].

[24] A. S. Foundation, "GitHub/Alamofire," GitHub, [Online]. Available: https://github.com/Alamofire/Alamofire#features. [Accessed 22 April 2021].

[25] "SwiftyJSON," GitHub, [Online]. Available: https://github.com/SwiftyJSON/SwiftyJSON. [Accessed 22 April 2021].

[26] "JSON," [Online]. Available: https://www.json.org/json-en.html. [Accessed 22 April 2021].

[27] "Maps SDK for iOS; Overview," Google, [Online]. Available: https://developers.google.com/maps/documentation/ios-sdk/overview. [Accessed 20 April 2021].

[28] "Places SDK for iOS; overview," Google, [Online]. Available: https://developers.google.com/maps/documentation/places/ios-sdk/overview. [Accessed 20 April 2021].

[29] "Geocoding API; overview," Google, [Online]. Available: https://developers.google.com/maps/documentation/geocoding/overview?hl=en_US#ReverseGeocoding. [Accessed 20 April 2021].

[30] "The Directions API overview," Google, [Online]. Available: https://developers.google.com/maps/documentation/directions/overview?hl=en_US. [Accessed 20 April 2021].

[31] "Distance Matrix API; overview," Google, [Online]. Available: https://developers.google.com/maps/documentation/distance-matrix/overview?hl=en_US. [Accessed 20 April 2021].

[32] "Enumerations," Apple Inc., [Online]. Available: https://docs.swift.org/swift-book/LanguageGuide/Enumerations.html. [Accessed 11 April 2021].

[33] "The Basics," Apple Inc, [Online]. Available: https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html. [Accessed 11 April 2021].

[34] S. Singh, "Impact of color on marketing," *Management Decision,* vol. 44, no. 6, pp. 783-789, 1 July 2006.

[35] C. R. Sunstein, "The Ethics of Nudging," *Yale Journal on Regulation 32 Yale J. on Reg.,* no. 2, pp. 413-450, 2015.

[36] L. Bovens, "The Ethics of Nudge," in *Preference Change: Approaches from Philosophy, Economics and Psychology (T. Grune-Yanoff and S. O. Hansson, eds.)*, vol. 42, Dordrecht: Springer Netherlands, 2009, pp. 207-219.

[37] T. M. Wilkinson, "Nudging and Manipulation," *Political Studies,* vol. 61, no. 2, pp. 341-355, 2012.

[38] T. Goodwin, "Why We Should Reject 'Nudge'," *Politics,* vol. 32, no. 2, pp. 85-92, 2012.

[39] U. Nations, "Universal Declaration of Human Rights," United Nations, [Online]. Available: https://www.un.org/en/about-us/universal-declaration-of-human-rights. [Accessed 26 April 2021].