6CCS3PRJ Final Year
Environmental Advisor App

Final Project Report

Author: Shitin Kamalia

Supervisor: Dr. Kevin Lano

Student ID: 1860139

April 30, 2021

Abstract

The main motive behind developing this mobile application was to provide a platform that would allow individuals to track and monitor their daily Co2 emissions through various mode of transportation. This application would be of great benefit as it would not only educate and inform people but it will act as a strong advocate for future sustainability. The application allows the user to set-up their account through which they can monitor their emissions by logging their journeys into the application. The overall project was intended to push the sustainability agenda and attract as much audience to this great initiative as possible.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Shitin Kamalia

April 30, 2021

Acknowledgements

I would like to start by expressing my since gratitude to my supervisor, Dr. Kevin Lano, who has religiously assisted me throughout the entirety of my project. Due to his contagious enthusiasm of the subject and motivating personality I was able to learn and achieve in this project more than I had ever imagined, and to this I cannot fully embody my appreciation for him.
In addition to this, I would also like to take this chance to thank my family and friends who graciously supported and encouraged me even in the toughest times when they themselves were challenged by the adversity of the ongoing global pandemic. Through their communication and assurance I was able to find closure in the hardest times and was able to push through. Working on this project was truly an exceptional experience for me.

Contents

Chapter 1

Introduction

This project intends to educate and inform the general population about the significant rise of greenhouse gases which have been the main root in initiating the crisis we know today as global warming. As carbon dioxide makes up 85% of greenhouse gases, humankind must limit their carbon emissions per person. This application intends to follow and promote the novel plan set out by King's College London, which wants to provide an easy-to-use and accessible platform for all. The application allows users to calculate their daily carbon emissions and promote them to take up more sustainable mode whilst using transportation.

1.1 Motivation

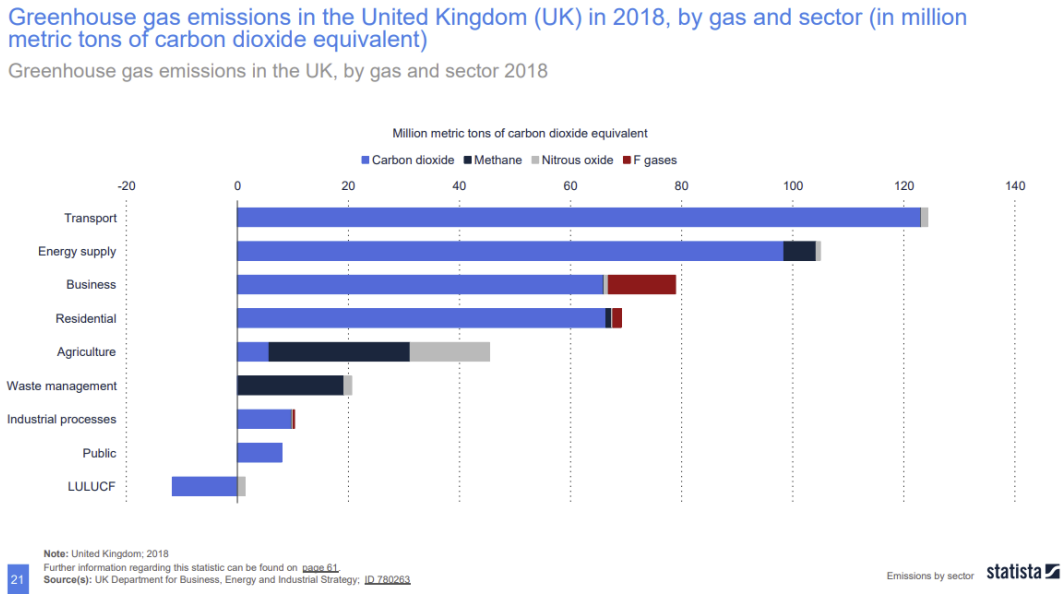
Carbon emissions are having a detrimental effect on the environment and every living thing on it. Temperatures are on a drastic rise causing sea levels to surge to dangerous water levels, affecting all land and sea creatures with it. Global warming has also impacted several countries economically due to frequent flooding and various other natural disasters that have devastated two significant industries, agriculture and fishing. A notable example of this is New England's major economic crisis because of climatic change, ergo plummeting the catch count in the lobster industry and coral reefs has been affected by rising sea levels. Moreover, global warming also poses a threat to the existence of the human race marking it as a threatened species. Major diseases reported as a cause of greenhouse gases are respiratory problems like asthma and several allergies. In addition to this, the rise of temperature has homed an ideal environment for mosquitoes to breed. This has led to an increase in vector-borne diseases like malaria, dengue etc.. Further to this, global warming has brought droughts and

famine in parts of the world, causing people to suffer and die of hunger. This is particularly astonishing in this age where technology is so advanced and distances have been cut short. However, still, 9% of the global population is struggling to have essential nourishment requisites. This project allows me to work on a subject that is of utmost importance and is meant to help in retaining a sustainable environment for future generations to come. Thus, I believe this project would be an excellent platform for me to use my knowledge and bring about a change I have always aspired to make.

Chapter 2

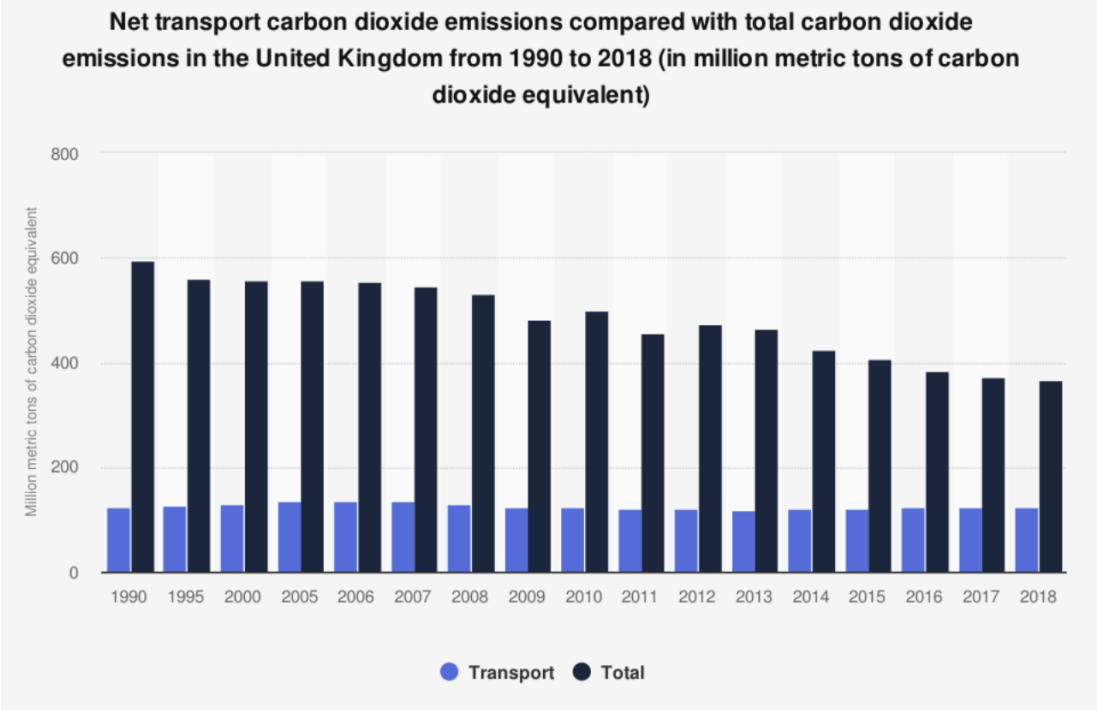
Background

2.1 Problem Overview



As indicated above, transportation is a significant contributor to carbon emissions, causing high pollution levels. With the increase in private cars owned by people, the pollution level has touched the sky demanding us to look for safer alternatives to sustain our only home. However, if we look at the market dynamics and the current usage, it is challenging to stop using fossil fuel altogether. We can attempt to reduce the number of fossil fuels we burn on an average in a day, but the complete omission of diesel and petrol has many hurdles.

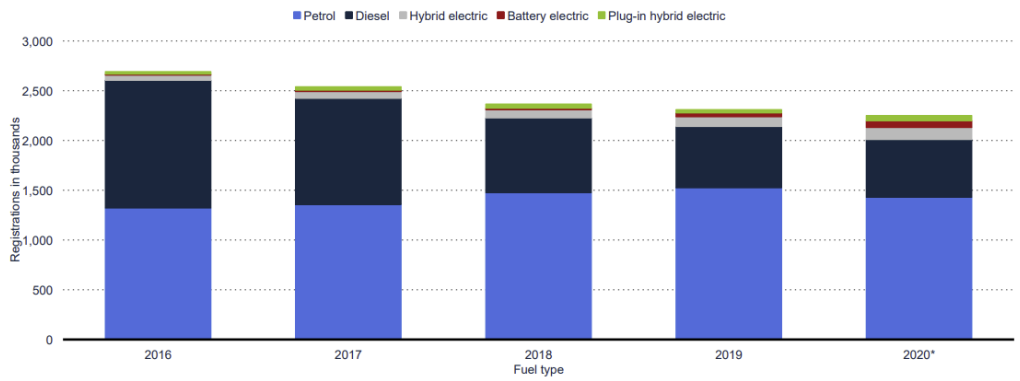
For instance, low demand for a commodity (oil in this instance) reduces the price of that commodity. However, a price reduction attracts all types of customer base and increases demand for the same commodity leading to higher consumption. Secondly, for many countries like the Middle-east, their primary source of income is oil mines and reservoirs. Thus they will continue to advertise, subsidise and export their oil for continuous income. Lastly, for many countries, especially third world countries, global warming and pollution aren't the main source of worries. It comes last in the list of food, security and protection. So, for these countries, economic progress is a bigger deal than global warming for survival. Nonetheless, we can still reduce the number of fossil fuels that transport burns and switch to more eco-friendly ways of travelling that save the planet and make it sustainable and give hope and motivation to future generations to continue building upon these eco-friendly techniques. The most basic solution to this problem would be to save trees, prevent wildfires and plant as many trees as we can. Trees reduce the amount of carbon dioxide in the air and give out more oxygen in return. Along with this, increased usage of wind and solar power energy to run engines, derivation of biofuels from organic waste of animals and setting a price for carbon so that it's not very easy to obtain. This will drastically reduce the number of greenhouse gases in the atmosphere.



As depicted above, there has been a plunge in carbon dioxide emission. Countries like the United Kingdom have taken many measures to curb down the same over the last two decades, but if you see carefully that the carbon dioxide from transport in the United Kingdom has been consistent over the past two decades. The increasing population over the last two decades should also be taken into account. Still, as there is a dip in the total carbon dioxide emission, this application has a significant scope to curb down the total emission by transport. Another main alternative to gasoline-run vehicles is electric vehicles. Electric vehicles have shown tremendous results and produce much less carbon emission than the fueled vehicles. However, the transport market is highly dominated by petrol and diesel vehicles, and it will take a lot of time for EVs to conquer it. Electric vehicles are a reasonably new concept to everyone, and it will take time for people to get used to them. Currently, owning an electric vehicle is unsuitable as there are hardly any electric vehicle charging stations. Electric vehicles tend to show inferior range in cold conditions and are much more expensive than the fueled vehicles. All sectors are working on making electric vehicles more affordable by reducing the battery cost, increasing the battery life and range and spreading more awareness on the matter of sustainability and depletion. Car companies are working on making electric cars appealing to all sectors of customers to attract a more extensive customer base and provide a broader range of styles and thus reduce the usage of gasoline in the future. Many countries like the US have already started providing incentives to people driving electric cars to attract more customers and also take measures to install EV chargers throughout the country. The author would also like to notice that there has been some rise in the sales over the past few years of electric vehicles (as depicted below); however, the market is still dominated by petrol and diesel vehicles, but people are getting more aware of this issue.

Passenger car registrations in the United Kingdom between 2016 and 2020, by fuel type (in 1,000 units)

New car units registered in the UK by fuel type 2016-2020



Although there has been a significant rise in awareness about carbon emission, there is no denial that only 1.1% of the total car sales in the United Kingdom are electric. The targeted dates for the ban of fueled vehicles in advanced countries like the United Kingdom is 2040, and according to a survey, an ordinary SUV generally lasts around 16-18 years with 200,000 miles and more. So still, electric vehicles have a long way to go to omit the use of fueled vehicles completely. More immediate solutions to prevent carbon footprint is by promoting means of transportation like biking or walking. This is not only eco friendly but also promotes daily exercise which is very much needed for a healthier lifestyle. Air travel also contributes largely to carbon emissions and as responsible citizens of the planet we should choose our mode of transport wisely and pack as light as possible since heavier aircraft, the more fuel it consumes. In order to reduce the usage of private cars, the government should spend their funds on improving the quality of public transport so that all sectors of people can utilize it. For example, the United Kingdom has an excellent public transport system where all routes are connected and transport from anywhere is easily accessible for everyone. It is really important for mankind to act upon this and start taking carbon emission into account from now to maintain a sustainable planet for our future generations., in which an application like this would be of much help.

2.2 About the Application

The principle intention of the project is to help the users estimate their carbon emission (a.k.a carbon footprint) while travelling from one location to the other. The app would not just estimate and calculate the emission but also persuade the user to choose a more conservative mode of transport, therefore help in diminishing a critical danger to mankind i.e. Global warming. The application is intended to spread mindfulness and awareness particularly among younger individuals. While monitoring their carbon impression just through their daily commute, will assist the users with reflecting upon their decision and spur them to take up a more sustainable mode of transport. In addition to this, the app will also implement calories calculation for sustainable transport like cycling or walking, in order to motivate the users to a healthy lifestyle.

2.3 Platform

When it came to deciding the platform on which the app should be developed, it was quite confusing for the author to choose between the two, Android app development or iOS app development. Since the aim of the application is to ”reduce the carbon emissions”, it was extremely important for the author that the application takes the bigger group into account. Over this perspective a lot of research was done. On researching about the same, out of all the smartphone users 71.82% of the crowd are using android based platform devices and only 21.43% uses iOS. The android app development also had few extra benefits over iOS, as Android is an open system. In Android, the developers get a lot more features easily accessible which might as well be restricted in iOS. Since it is one of the first mobile applications that the author was developing, he personally thought that it would not be a good idea to go for iOS app development as the deployment and publishing app on app store is way more difficult than google play store. Thus, the author decided to develop the application on the Android platform.

Chapter 3

Requirements & Specifications

3.1 Requirements

The primary thought behind this application is to build up a Map-based mobile application, enabling users to keep a log of their carbon footprint through their daily commute. Below mentioned are the key requirements of the project.

- The application should be developed in either android or ios platform.
- The application could be developed in any suitable language of their choice.
- The application should work in mostly all the mobile devices of chosen platform.
- The graphical user interface should be compatible with the chosen platform mobile devices.
- The application must allow the user to recover their account in case the user forgets their password.
- The application must allow the user to register with the application.
- The application must allow the user to login with the application.
- The application should allow the user to update their profile details.
- The application must use Google Map API for the map based interface.
- The application should seek the user's consent before using their location.
- The application must allow the user to define their journey (i.e The start and the end location of the journey).

- The application could give suggested places recommendation while the user is typing in a location.
- The application should allow users to choose location preference on map using the pin pointers.
- The application could draw the route between the start and end point of the trip.
- The application could display an animation on the drawn route.
- The application should store the trip journey of every user in the background allowing them to reflect upon the carbon emitted.
- The application should calculate the estimated co2 emissions during the run time and similarly display it to the user.
- The application should promote the use of more eco-friendly mode of transport.
- The application should calculate the estimated calories burnt during the run time and similarly display it to the user for cycling and walking.
- The application should be easy to use and search-able, thus inviting users from all age groups.
- The application should adhere to the Jakob Nielsen's 10 general principles for interaction design. (https://www.nngroup.com/articles/ten-usability-heuristics/)
- A user guide to the application should be produced along side with the project.

3.2 Specifications

Setting up the specification of all the requirements listed above is one of the most significant parts of the development process of this application. This phase allowed the author to look upon the functional and non-functional requirements mentioned by the supervisor and further allowed him to design and develop the application. This phase also allowed the user to prioritise his work as every requirement is given a significance rating and gives him a head start on the project.

Serial Number	Requirement Type	Requirement details	Specifications	Significance
R1	Functional	The application should be developed in either android or iOS platform.	The application should be specific to just one platform either iOS or android. The chosen platform for this application is Android.	High
R2	Non-Functional	The application could be developed in any suitable language of their choice.	The application can be developed in any coding languages if it is functional with the platform chosen and fulfils all the requirements. The chosen language for this application is Java.	High
R3	Non-Functional	The application should work in mostly all the mobile devices of chosen platform.	The application needs to be compatible with all the android device (i.e. the application should work with different versions of Android)	High
R4	Functional	The graphical user interface should be compatible with the chosen platform mobile devices.	The author needs to make sure that there is no hardcoded height and width and always uses the relative height and width, so the GUI of application remains unaltered in any android devices.	High
R5	Functional	The application must allow the user to recover their account in case the user forgets their password.	The author needs to have a forgot password functionality added to the login page to allow users to recover the password.	Medium

R6	Functional	The application must allow the user to register with the application.	The author needs to have a registration page link to login page, to allow new users to register. The database management system should be able to add new users.	High
R7	Functional	The application must allow the user to login with the application.	The author needs to implement a database handling system to facilitate the login authentication.	High
R8	Non-Functional	The application should allow the user to update their profile details.	The author needs to implement a profile update page through which the users can change their details and the same to be stored in the database.	Low
R9	Functional	The application must use Google Map API for the map-based interface.	The author needs to generate a key to facilitate the use of maps in the application before starting with the implementation.	High
R10	Non-Functional	The application should seek the user's consent before using their location.	The author needs to implement a dialog box which should be appeared as soon as the user starts the application, to seek consent for using location.	High
R11	Functional	The application must allow the user to define their journey (i.e. The start and the end location of the journey).	The author must implement different pages to allow the users to choose both the start and the destination.	High

R12	Functional	The application could give suggested places recommendation while the user is typing in a location.	The author could implement the Place API to allow a drop-down suggestion of places whilst the user types a location.	Medium
R13	Functional	The application should allow users to choose location preference on map using the pin pointers.	The author needs to implement an option which would allow users to access location through maps for both: Start and End locations	High
R14	Functional	The application could draw the route between the start and end point of the trip.	The author needs to implement the route drawn in the application. Thus, the author needs to parse the data abstracted and make the route.	Medium
R15	Functional	The application could display an animation on the drawn route.	The author could display moving vehicle animation on the route drawn adding aesthetics to application as well as attracting young users.	Medium
R16	Functional	The application should store the trip journey of every user in the background allowing them to reflect upon the carbon emitted.	The author must implement a separate page in which the user can see their past records and arrange them date wise.	High
R17	Functional	The application should calculate the estimated co2 emissions during the run time and similarly display it to the user.	The application needs to have a calculating co2 method, through which the co2 emitted could be calculated for all the journeys.	High

R18	Functional	The application should calculate the estimated calories burnt during the run time and similarly display it to the user for cycling and walking.	The same algorithm to calculate the co2 emissions can be used for an eco-friendly mode of transport like walking or cycling to count the calories (Approx. calories) of a person and could be similarly shown like the co2 emissions.	High
R19	Functional	The application should promote the use of more eco-friendly mode of transport.	The application needs to have some indication when choosing the mode of transport which would indicate the mode of transport is comparatively eco-friendlier and emits less carbon.	Medium
R20	Non-Functional	The application should be easy to use and search-able, thus inviting users from all age groups.	The application should follow principles designs methodologies as taught in Human Computer interaction module.	Medium
R21	Non-Functional	The application should adhere to the Jakob Nielsen's 10 general principles for interaction design.	The author needs to refer to the Jakob Nielsen's heuristic principle while designing the application, allowing the maximum usability to all kind of users.	Medium
R22	Non-Functional	A user guide to the application should be produced alongside with the project.	The author needs to make a detailed user guide to show the working of the application as well as step by step guide to the application.	High

Chapter 4

Software Architecture

4.1 Software Development Model

Coming to the development model to follow for the development of application, the author chose to follow the ITERATIVE WATERFALL MODEL. The following model was picked by the author because in this model, the requirements are implemented in fraction which makes the initial implementation of software easier. The simple implementation is then iteratively enhanced till the finished product is achieved. This appeared to be an ideal fit for the author as the initial requirements were well specified by the supervisor and the iterative waterfall model made the initial implementation of application simpler. In the initial stages, further functionality for each requirement was evolving and this model allowed the author to begin with the basic requirements without wasting time on specifying all the requirements. The author could always get back to any implementation and modify the same if he wished for it. For example, in the early stages of development, the animation was not considered for the application but later it allowed the author to get back and enhance the map requirement. Another major factor for choosing an iterative waterfall model by the author was that it's flexibility to change and add requirements. Although the following had it's setbacks as the author personally thinks that it required more management but overall the author is really satisfied with his choices.

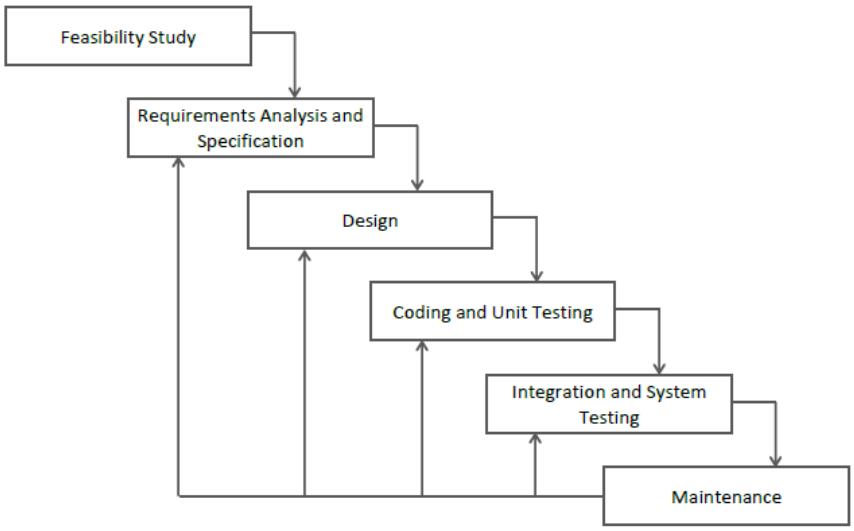
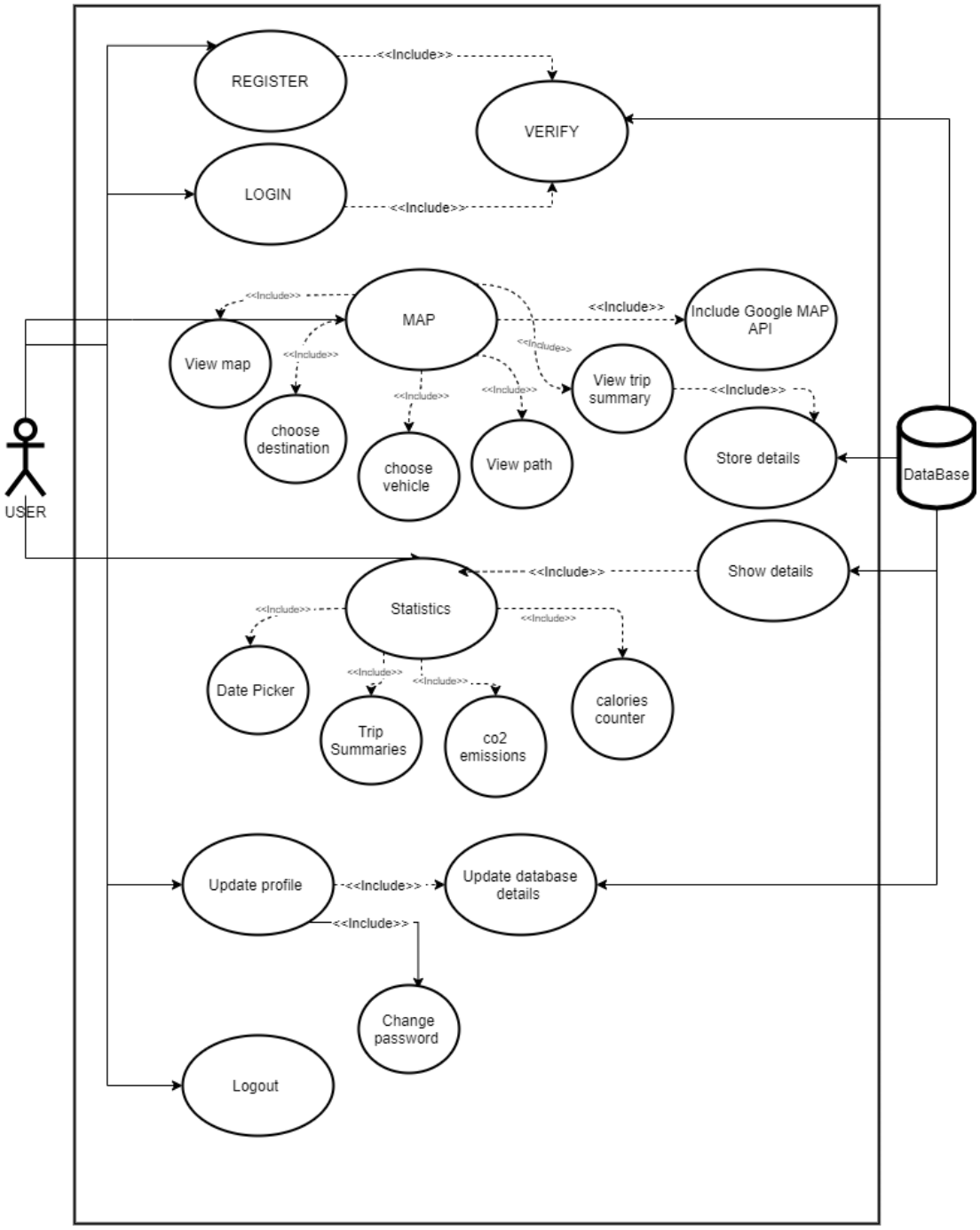


Figure 2: Iterative Waterfall Model

4.2 Use Case Diagram

Use case is a behavioural UML diagram which portrays the interaction between the actors and the system. The user acts as an actor to the application. The following use case diagram helped the author encapsulate the core functionalities that are needed to be implemented in the system as well as gather a brief glimpse of the flow. This turned out to be of great help before starting with the implementation of the application. The use case diagram for the application was created using `draw.io`

PROENV-Y



The above displayed use case diagram comprises the user as the primary actor and the database as the secondary actor. The use cases in the diagram define the core functionalities that are essential for the application.

- **Register** : It is important for the application to allow the users to make their account before signing up.

  Preconditions: The user needs to download the application beforehand.

  The register use case will take in all the details from the user and the verify use case is used to verify it before the details are stored in the database. (Secondary actor)

- **Login** : It is essential for the application to have a login option as each user will have different carbon emissions which needs to be stored separately in the database.

  Preconditions: The user needs to register with the application before signing up.

  The login uses the verify use case to check if the credentials entered by the users are correct from the database.

- **Map** : Map is the most important functionality of the application, that will allow the user to choose their destination as well as the calculation algorithm for the software will be embedded here.

  Preconditions: The user needs to have logged in the application.

  The map use case here is the main functionality that the author has hinted at. In the initial stages of developing an application when this use case diagram was made the author had connected the Map use case with few other functionalities (use cases) that the map might include.

  1. Google Map API: This is where the google maps will be connected to embed the map inside the application.
  2. View map: This functionality will allow the user to move the map according to their choice
  3. Choose Destination: This would allow the map to allow the user to specify their source and destination of the map as well as drop in suggestions while typing in the location .
  4. Choose vehicle: This would allow the user to choose between different modes of transportation.

- **Statistics** : The following functionality would retrieve the trip summaries of all the trips from the database and would allow all the users to see their trip history and the respective co2 emissions for the same.

Preconditions: The user needs to have slogged in the application as well as should have taken a trip from the map.

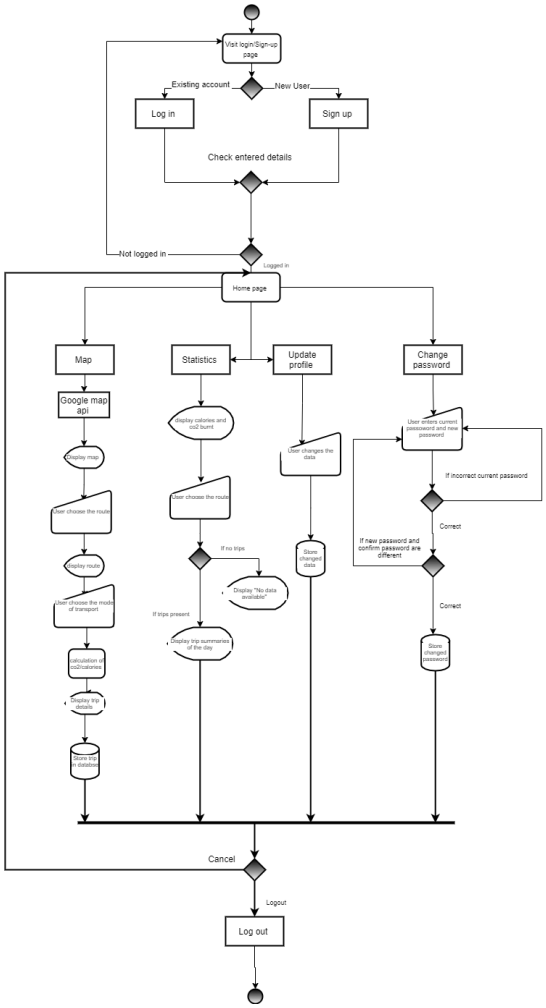
The statistics will include the following functionalities (Use cases)

1. DatePicker - The purpose of this is to allow users to go to specific dates and see the trip summaries.
2. Trip summaries - This will provided a list view of all the trips on a specific date
3. CO2 emissions - This will show the total co2 emitted by the user
4. Calories burnt- This will allow the user to see the total calories they burn so far by using the eco-friendly mode of transports with 0 emissions. (Walking/cycling)

- **Update Profile**: The following use case is very similar to register and by this functionality the user will be able to change their details like Name, DoB etc.., in case some user gives in a wrong detail while registering. This functionality would also allow the user to change to their password in case they wish to do so.
- **Logout**: The last functionality in the use case diagram is logout and this will allow the users to sign out of the application if they wish to and take them back to the login page, allowing them to switch to another account.

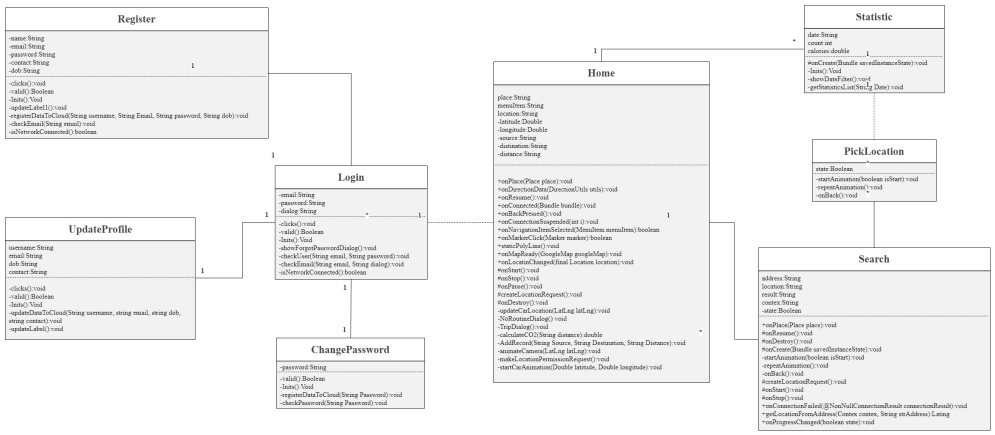
4.3 Activity Diagram

After understanding the key application functionalities to be included in the application with the use of USE CASE DIAGRAM, the author thought the next important step in development would be to understand and gain an abstract view of the flow of application. Hence, the author decided to create the activity diagram. Activity diagram is another type of behavioural UML diagram and can be referred to as an enhanced version of flowchart. It portrays the dynamic aspects of the system as well as conveys the flow of the application from one activity to another. It also manifests the pre and post conditions of the use case and helps the author decide on them. The activity diagram was a really supportive step taken during the development phase. The following Activity diagram was created using an online tool draw.io.



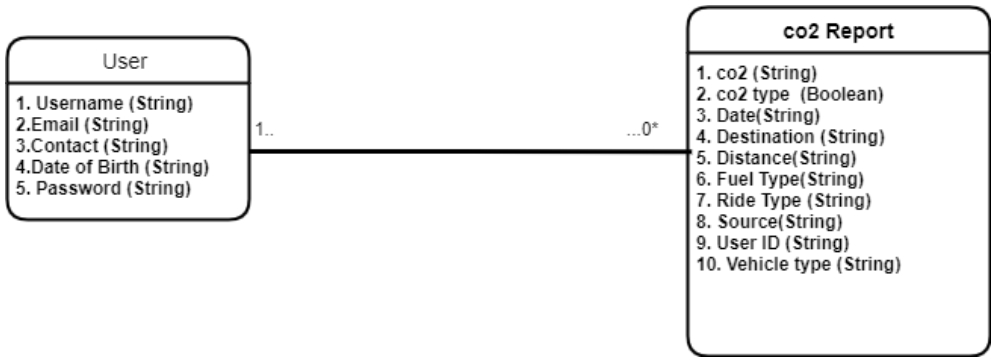
4.4 Class Diagram

Class Diagram is a behavioural UML diagram. It provides a static view of application and portrays a structure of an application class and its attributes. The below mentioned class diagram consists of the 8 classes and their attributes. This also portrays the relationship between the classes. The following class diagram was created using an online tool draw.io.



4.5 ER Diagram

ER stands for entity and relationship respectively. An entity could be anything (for example, place, event, person) that is applicable to a system. The ER diagram is a visual relationship between the different entities in the application and helps in explaining the logical structure of the database. Since the application uses a back-end database storage management, the author thought that it would be a nice idea to go about the steps of designing an ER diagram. The ER diagram helped the author design the database schema and also could prove to be helpful in troubleshooting databases. The following Entity-Relationship diagram was created using an online tool draw.io.



The database design of the application was kept very simple by the author. The database

just consisted of 2 entities.

- **User** : The following saved the user profile details.
- **Co2 Report**: The following saved the specific details of a user's trip.

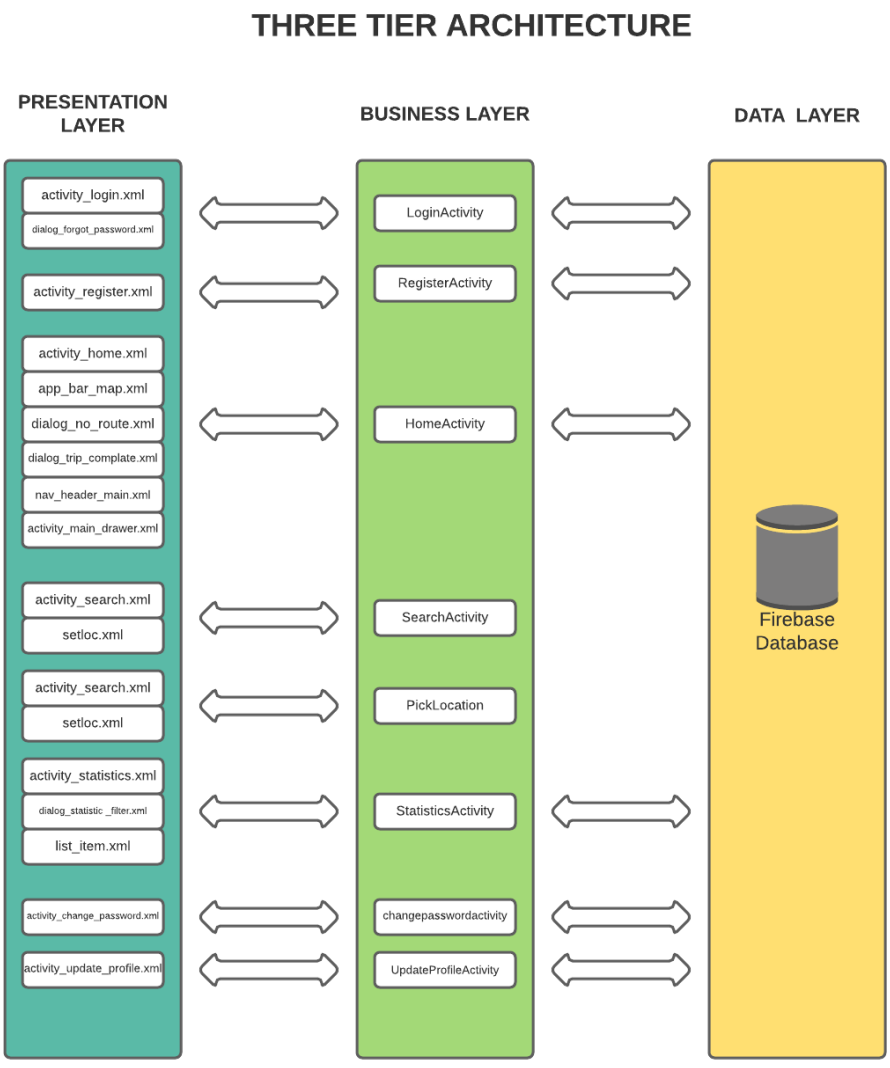
The relationship between the enities is one to many. Every user may have zero or many trips.

4.6 Three Tier Architecture

Three-tier architecture is a software architecture in which the application is divided into 3 tiers of logical computing. The following Three Tier Architecture diagram of the application was created using lucidchart.com

1. User Interface
2. Business Logic Layers
3. Data Storage Layers

The author throughout the process of development aimed at making the application flexible for future updates and improvement. Thus, the author tried to structure the code into 3 different tiers that would facilitate the same. The other benefit of using three tier architecture is the potential of improving something specific independently without affecting the other layers. The user interface of the application can be easily upgraded without affecting the business and data management layers of the system.


THREE TIER ARCHITECTURE
PRESENTATION LAYER
BUSINESS LAYER
DATA LAYER
activity_login.xml
dialog_forgot_password.xml
activity_register.xml
activity_home.xml
app_bar_map.xml
dialog_no_route.xml
dialog_trip_complate.xml
nav_header_main.xml
activity_main_drawer.xml
activity_search.xml
setloc.xml
activity_search.xml
setloc.xml
activity_statistics.xml
dialog_statistic _filter.xml
list_item.xml
activity_change_password.xml
activity_update_profile.xml
LoginActivity
RegisterActivity
HomeActivity
SearchActivity
PickLocation
StatisticsActivity
changepasswordactivity
UpdateProfileActivity
Firebase Database

Chapter 5

Design

Listing down the important requirements and getting an abstract view of the application through uml, the next crucial step in development of the application was the design phase. Having listed down all the requirements gave the author a heads up on few important but unsaid things like compatibility of the application with other android devices etc. The list of requirements and specifications allowed the user to identify the design of the application and the important functionalities that need to be kept in mind while developing the application. In this chapter, the author would be portraying his mind storming and ideas for the application and providing a prototype view of the application. The designing face was keenly looked at by the author where he had tried to apply the concepts learnt in the module - Human-Computer Interaction.

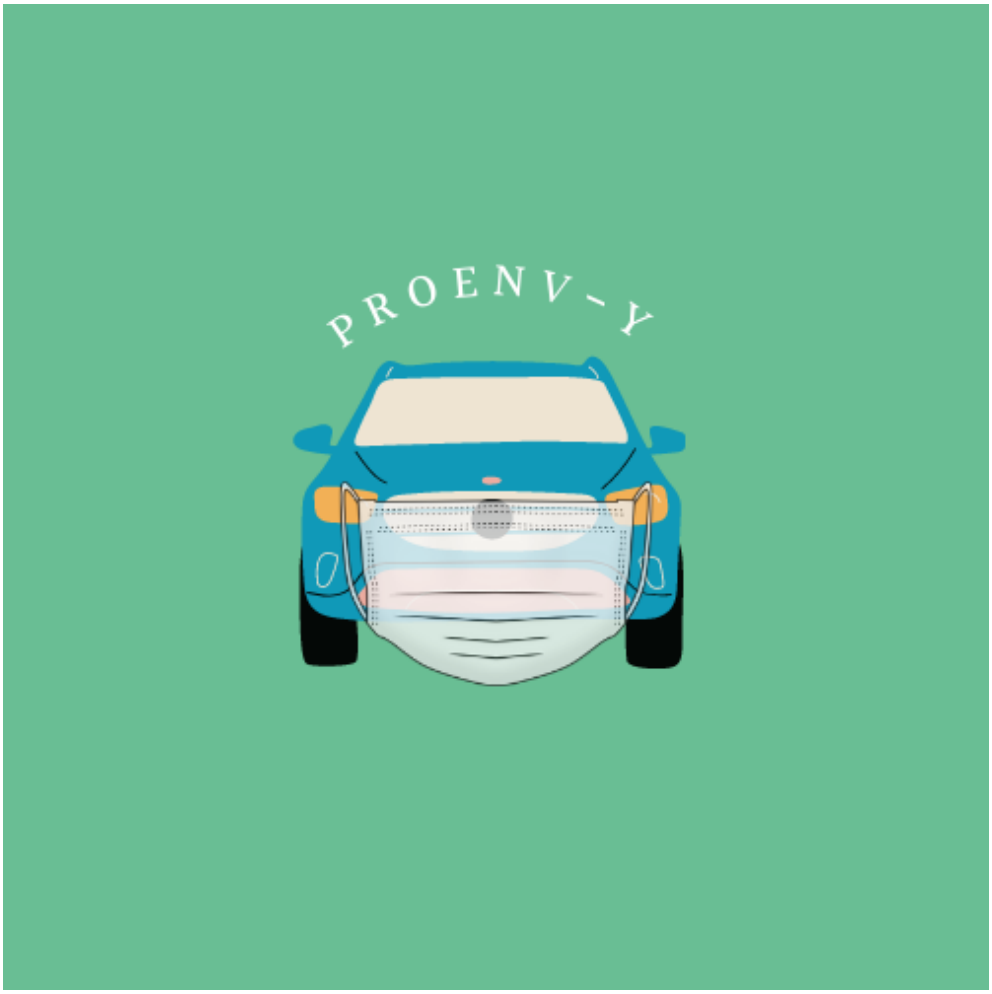
5.1 Application Name

One of the fundamental steps in designing an application is the choice of a fitting and appealing application name. it is important for the application name to consider its core features yet be memorable to the user. Another significant aspect that the author had in mind while considering the application name was that the name is not taken by any other application. Additionally, application names should also be easily searchable. Reflecting upon the core features and purpose of the application, the only motto that comes to mind is " Protecting the environment ". After a lot of playing around with the phrase "Protecting the environment" PROENV-Y (pronunciation: Proenvie) was decided as the application name by the author. The name "PROENV-Y" was not occupied by any other application in the

play store. fortunately, no domain existed with the accompanying name which made it easily searchable in any search engine and the first result could be directed to the application.

5.2 Application Logo

Another important aspect of designing an application is the choice of a fitting application logo. A well designed and appealing logo fabricates trust and attracts curious users to download the application and check the functionality of the application. In the beginning, deciding the name of the application seemed like a difficult job but it turned out to be the opposite when it came to designing the logo. Having decided on the name and looking at the core trait of the application, using a vehicle was considered to be the part of the logo. A lot of sample logos were created with the car as a part of the logo on canva but all of them turned out to be very mainstream or unattractive. Being resolute about having a very unique and appealing logo, I started noticing the surroundings for ideas. At some point while strolling, a board spread mindfulness about the Covid-19 pandemic and urging individuals to wear masks was noticed. That's how the association of mask with reducing the spread was made and using the mask in the application logo was contemplated. Fortunately, the thought and perception turned out great. Picking the colour scheme of the app logo was fairly easy as using environmental colours like green and blue would have been the most ideal decision. An exact hexadecimal colour code was used for green and blue respectively. While designing, the round logo was additionally remembered and that is the manner by which application name text was written in circular like motion. The actual logo gives a feeling of the core characteristics of the application and it could not get any more appropriate.



5.3 Personas

Before designing the application, an important step is to analyse the target audience of the application. Persona is a fictional representation of the potential user for the application. Creating personas helped the author pick out and assess the target audience for the application and also helped in bringing the common objectives and goals that needed to be fulfilled for all clients before actually developing and designing the application. For making the personas `www.canva.com` proved to be really helpful to the author allowing him to design the personas.

5.3.1 About the Personas

For the following mobile application - 4 personas were created targeting different users to contemplate on the design of the application.

- Matt, a 15 year old school student, who is worried about his grandparents health
- Julien, a 37 year old financial analyst and a mother of 2 children, worried about the future generation and keen to promote sustainable methods of travelling for health benefits.
- Ricky, a 25 year old environmentalist worried about the future generation and constant pollution degrading the air quality in his city.
- Jordan Freeman, a 75 year old retired businessman wants to spread awareness about the increasing pollution to save the future generation.

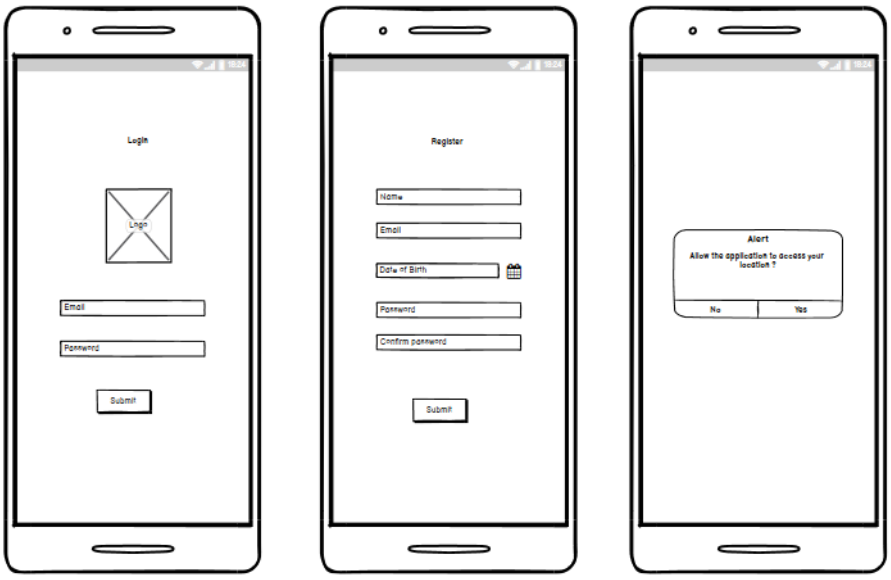
5.3.2 Conclusion

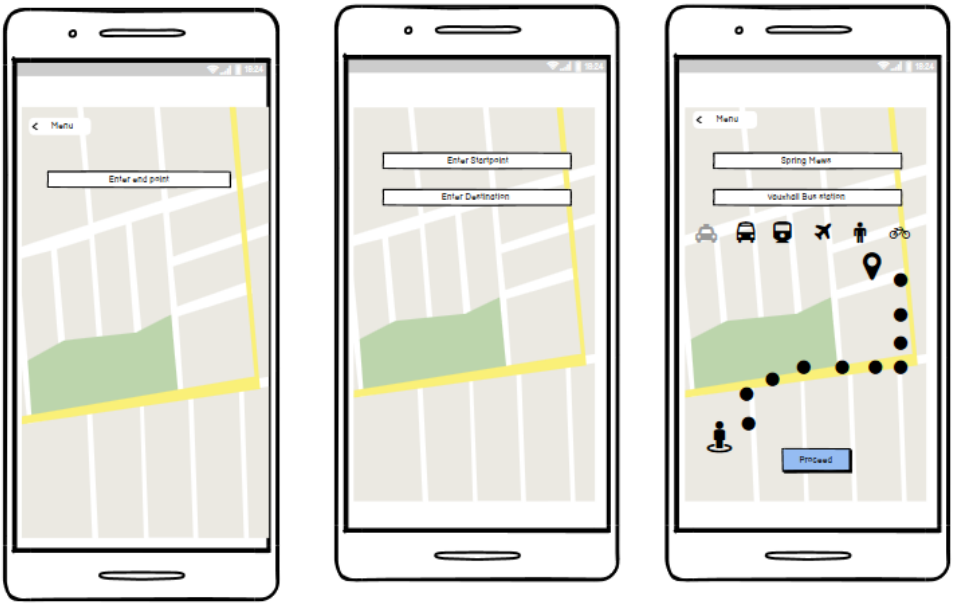
After analysing the target audience, it was assessed that most of these individuals are keen on protecting the environment for various reasons but mostly this app targets all kinds of

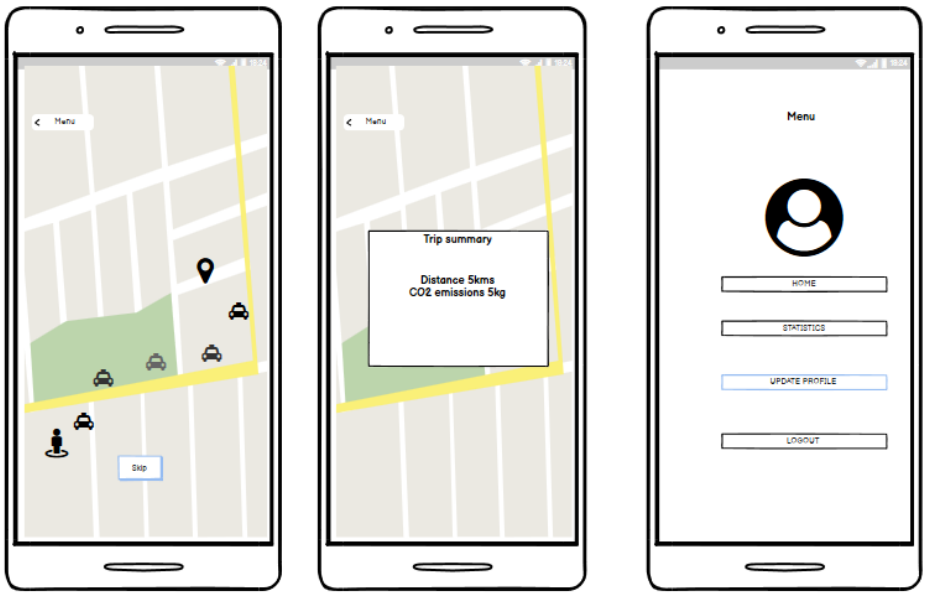
consumers, be it children, the older generation or working mothers. This gave the author an insight that since the target audience consists of people from different age groups, the app design should be appealing, easy to use and efficient, catering to every individual's needs.

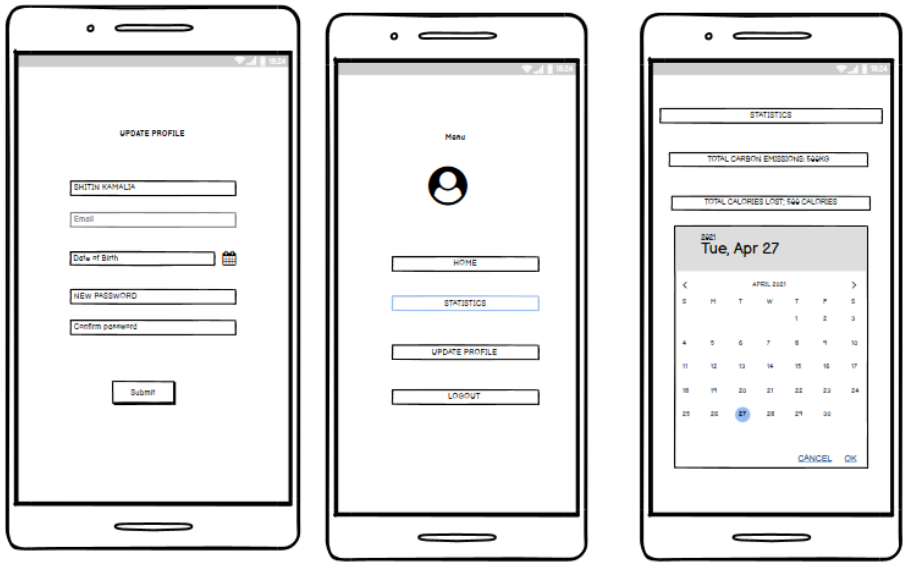
5.4 Prototyping

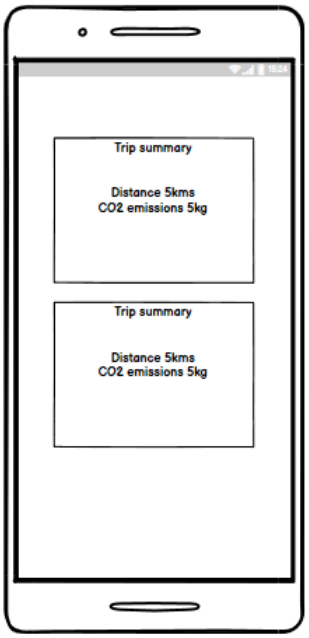
Prototyping of mobile applications is a versatile application model that shows how an application will function.The purpose of making the low-fidelity prototype for the application was to have a clear set of rough templates following which the XML layouts that need to be created for the actual application could be made. This also saved the author with the hassle of re-designing the final product and saving time. This process helped the user in understanding the flow of the application and maximise the efficiency of development. It also helped the author in visualizing the working of the application and portraying a functioning plan and layouts. The initial process of prototyping had just begun with a piece of paper and pen in which the author had tried to create different design structures for the application. Having the personas ready, it was important for the author to cater the needs of all the potential users. The rough sketches of prototypes were later analysed and after a lot of self-evaluation and reasoning, the author was able to come up with a rough design estimate. This rough design was later converted to an actual prototype using the software `Balsamiq`, to get a clearer glance at the application.


Login
Logo
Email
Password
Submit
Register
Name
Email
Date of Birth
Password
Confirm password
Submit
Alert
Allow the application to access your location ?
No
Yes


Menu
Enter end point
Enter Startpoint
Enter Destination
Menu
Spring Mews
Vauxhall Bus station
Proceed


Menu
Skip
Menu
Trip summary
Distance 5kms
CO2 emissions 5kg
Menu
HOME
STATISTICS
UPDATE PROFILE
LOGOUT


UPDATE PROFILE
SMITIN KAMALIA
Email
Date of Birth
NEW PASSWORD
Confirm password
Submit
Menu
HOME
STATISTICS
UPDATE PROFILE
LOGOUT
STATISTICS
TOTAL CARBON EMISSIONS: 5KG
TOTAL CALORIES LOST: 500 CALORIES
2021
Tue, Apr 27
APRIL 2021
CANCEL
OK



5.4.1 About the Prototype

The login and register pages were very mainstream for the application. Coming to the navigation of the application, the author wished to have a convenient navigation throughout the application,allowing the novice users (for example, the older generation) to get back in case of undesirable touch presses. A lot of consideration was made for the same. Initially, the author had decided to have navigation in the bottom of the screen but later the idea was rejected. Not only the author felt that it would make the application look complex but also this could result in a lot of undesirable clicks by the user while using the application. After a great deal was reasoning, click navigation bar was decided. This nav bar would not only avoid undesirable screen clicks but would also allow the application to have new functionality that could easily be listed down in the navigation. The author was already quite determined to have a simple design structure and the decision of keeping the map on the home page was made and the other functionalities to be linked up in the navigation bar. While prototyping the author was really perplexed about the simple design, as he felt that this might not attract the younger generation. Owing to this reasoning, the author came up with the idea of having an animation after setting up the route. Initially the author wished it to be live but owing to google maps restriction, the live tracing could not be implemented.

5.4.2 Conclusion

The process of prototyping was very crucial for the development of the application because the ProEnvy is a user-centric application and design plays an important role in an application. This process served as a booster for the author, allowing him to get a start on the development phase of the application and as mentioned before come up with appealing ideas in the application.

Chapter 6

Implementation

6.1 Development Tools

6.1.1 Front End: Android Studio

While thinking about the software to use in the application development, a lot of considerations were made by the author including Android-Eclipse and React Native however after a lot of research Android Studio automatically became the obvious choice. Android studio makes the app development and deployment very easy because of being an open source platform. It is relatively easy to use and comes with a Firebase assistant, which would be especially helpful in connecting firebase to my application. It also offers GUI (availability of drag and drop) and is comparatively a much more stable system than android eclipse. The android studio emulator also allows to test code on a variety of devices. Furthermore, react native, a relatively new app developing language allowing cross-platform development, had also been considered at length but due to its limitations and lack of community support compared to Android studio, the author dismissed the idea.

6.1.2 Back End: Firebase

It was essential for the application to have a backend storage management as it was required for the user to login and the back end system is a must for storing the emissions and trip details. While thinking about the backend system Firebase was an obvious choice but a lot of other options were still considered. Other options like php api and node api with domain and server space were considered but this would not only involve manual API creation but also the app would have to contact it and then make changes in the database. Whereas on the

other hand, Firebase is an open source Backend-As-A Service (BaaS) platform enabling a variety of tools to the developer. It is a product by Google so comes with good documentation and the backend storage is real time and faster than any other options considered. Thus Firebase was used in the backend management of the application

6.1.3 Emulators

Emulators are virtual devices usually built in the app developing IDE. In the following application, Android studio was used which already provides the developer with a range of devices of different sizes such as small devices, devices with high resolution etc. These enable the developer to check their program and it's compatibility with various devices. Following are the official devices supported by Android:

- Pixel
- Nexus
- Samsung
- Motorola

6.1.4 Google Map API

As the application involved map integration, it was essential to add a map based API in the application. It was already mentioned that in the application specification provided by the supervisor that the application should use a google map API. According to which Google Map API Android SDK was enabled in the application. Other map based API were looked at like Open Street but they were not as flexible as Google map API. Another important point was that the Google MAP and firebase were internally connected as the same account was used for configuration, which turned out to be really convenient for storing the user trip summaries and etc.

6.1.5 SDK

SDK stands for Software Development Kit. SDK is a collection of tools provided by the application (Android) that facilitates the development of the application. Normally the SDK includes varieties of items like libraries, documentation, processes and etc., that can be used by the developer to integrate into their application. The list of SDK used by the author in PROENV-Y can be seen in the build.gradle (module) under dependencies.

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'com.google.android.material:material:1.3.0-rc01'
    implementation 'androidx.navigation:navigation-fragment:2.2.2'
    implementation 'androidx.navigation:navigation-ui:2.2.2'
    implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.2.0'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'
    testImplementation 'junit:junit:4.13.1'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
    implementation 'com.google.firebase:firebase-analytics'
    implementation platform('com.google.firebase:firebase-bom:26.2.0')
    implementation 'com.google.firebase:firebase-firestore'
    implementation 'com.google.android.gms:play-services-maps:17.0.0'
    implementation 'com.google.android.gms:play-services-location:17.1.0'
    implementation 'com.google.android.gms:play-services-places:17.0.0'

    implementation('com.alibaba.android:ultraviewpager:1.0.7.7@aar') {
        transitive = true
    }
    implementation 'com.google.maps.android:android-maps-utils:0.5'

    //circle image
    implementation 'de.hdodenhof:circleimageview:3.0.0'
    implementation 'com.sothree.slidinguppanel:library:3.4.0'
    implementation 'com.github.tintinscorpion:Dual-color-Polyline-Animation:1.0'

    //Place API
    implementation 'com.google.android.libraries.places:places:2.0.0'
}
apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'
```

Dependencies

6.2 CO_2 / *Calories* Calculation

Due to fuel consumption and other fossil fuel combustion there is a high co2 emission quantity with the usage of various modes of transport. To tackle this aggravating issue strong efforts are being made to not only educate the general population about the significant drawbacks of these high co2 emissions but also alternative modes of transport are being discovered and encouraged in order to promote a greener future. The author of this application has done precisely so, he has managed to gather sophisticated data by processing and visualising it using the python jupyter notebook.

Respecting the given timeline for the project, the author was not able to gather and process enough data to calculate co2 emissions for every car specifically however the author programmed the application in a way that this feature could be added in the future, thus accounting for scalability. An example of this can be seen by considering the login and signup

functionality of the application which allows the user to set up a profile which can be used to account for future user preferences. The way calculations are currently being monitored is through the author's efforts of obtaining average emissions respective to car model and energy consumption type.

To get the best average value to calculate the co2 emissions the author did not only merely average out co2 emissions across a range of vehicles, but the author also used Python library Pandas to clearly map out co2 emissions and was able to sketch a graph thus helping to obtain a more real average for calculation. It must be highlighted here that all these functionalities have been programmed in a way that allows the ability of having specific co2 emissions if the app was to scale in the future.

To manage the data acquired, Python Jupyter notebook has been used through which meaningful tables and graphs had been constructed which aided in the main calculations of the application. A final table of all the calculated Co2 values has been included below:

```
In [6]: data = {'Fuel Type':  FuelType,
                'Compact Car (KG)': CC,
                'Mid Range Car (KG)': MRC,
                'Luxury/SUV/Van (KG)': LSVC
                }

        df = pd.DataFrame (data, columns = ['Fuel Type','Compact Car (KG)', 'Mid Range Car (KG)', 'Luxury/SUV/Van (KG)'])

        display(df)
```

	Fuel Type	Compact Car (KG)	Mid Range Car (KG)	Luxury/SUV/Van (KG)
0	Petrol	0.28	0.34	0.41
1	Diesel	0.24	0.31	0.39
2	Hybrid	0.21	0.26	0.30
3	PHEV	0.19	0.24	0.28
4	Electric	0.07	0.08	0.10

All the data before processing had been acquired from trusted and reliable sites. They helped the author draw strong and reasonable conclusions through which the author was able to come up with the final values, shown above. These values were then included in the code and are used in run time to calculate the amount of Co2 that is produced against a kilometer of transport using the respective fuel type.

In addition to this the author has also taken into consideration long distance travel i.e. through planes . For this the author wished to use the great circle distance. Basically, the great circle distance is followed by the flights and is the shortest distance between the 2 places on any part of Earth. But owing to the google maps API restriction and route generation requirement for animation, the user could not implement this and thus the co2 emissions by flight are calculated on the driving route and is only possible if the distance between the source and destination is more than 500 kms and there is a possible car route in between them.

The per kg/km data for flights, train and bus were taken from: `http://www.aef.org.uk/downloads/Grams/_CO2/_transportmodesUK.pdf`

The calories implementation was done in the similar way as CO2. In this the author looked up for average calories burnt by a person on per km walking and cycling and the same values were implemented. The average calories burnt per km walking was taken as 47 and for cycling 25.

6.3 Code Structure

When it comes to the code structure the author tried to achieve the boilerplate structure. Different parts of the codes were differentiated in different folders, for example Models in models, utils in Utils and Adapters in Adapters and the layouts in Layout. This structure enhances the readability of the code for others as well as increases scalability of the software. The author also tried to achieve MVC architecture for the application. MVC is an architectural design which tries and separates the application into 3 main logic.

1. **MODEL** - Data Related Logic
2. **VIEW** - User Interface of the application
3. **CONTROL** - Acts as a communicator between Models and Views.

MVC is the most widely used software architecture pattern and the author considered the following keeping the 3 tier architecture in the mind. Although the author was not able to implement pure MVC.

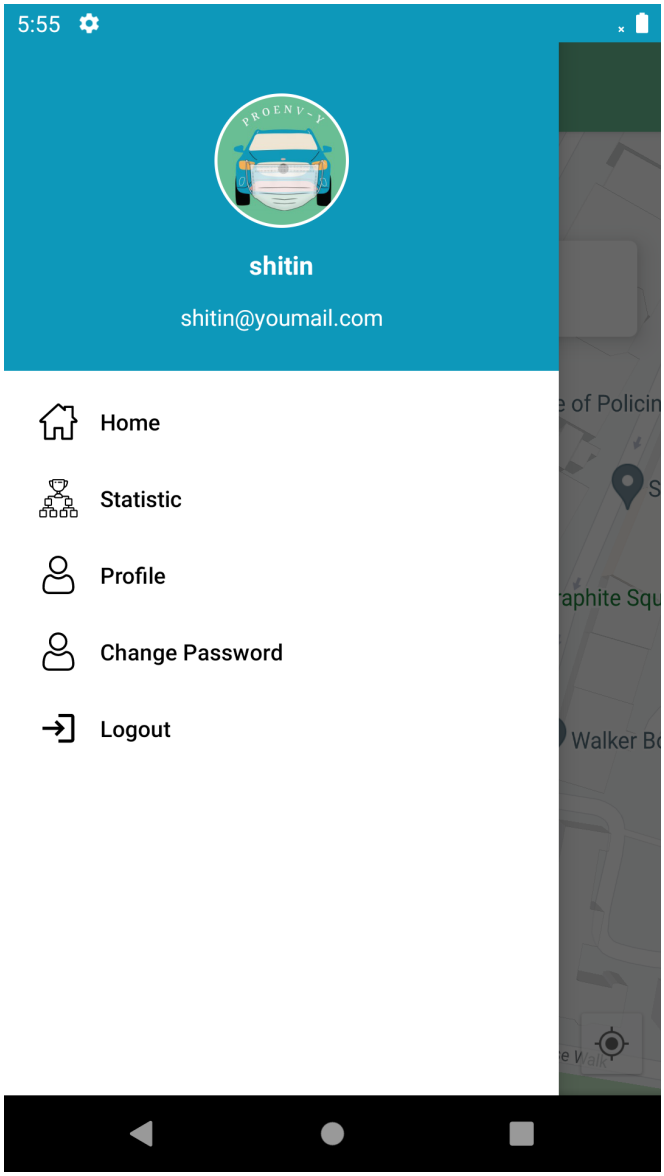
6.4 Code Implementation

6.4.1 Design

6.4.1.1 Graphical User Interface

When it comes to talking about the implemented design of application, the most important aspect is the XML layouts. In Android applications, screens are defined in XML layouts files which are managed by activities. The XML layouts were created in Android Studios itself and can be found in layouts under res folder in the source code. In all the XML layouts created the author tried to use mostly constraint layout to reduce the layout layers redundancy.

The user interface components used in the application by the author is fairly simple as the design phase (section 5) made the author realise the potential users and he aims to cater to everyone's needs.. The user interaction inside the application is fairly short (maximum no. of steps being 3) and only linear interactions is possible. A lot of xml layouts were created throughout the application and with the use of setVisibility they were accordingly displayed when needed. Another important aspect of mobile application is the UI navigation. The author aimed to reduce the use of forward/reverse navigation as much as possible and aimed to have lateral navigation throughout. Forward/ Reverse navigation may need specific coding whereas in lateral navigation there is no need for additional coding. In the Forward navigation, the screen movement is at successive layers of hierarchy which are normally implanted into buttons or using search and containers.In reverse navigation, the user moves backwards either chronologically or hierarchically. Forward/Reverse navigation was inevitable for a few parts in the application (for example in login and register pages) where it is necessary for the application to invoke activities via intents. Coming to the lateral navigation, in lateral navigation the screen movement is at the same level of hierarchy. The application after logging in has a navigation drawer on the top right corner which supports this. Initially the author thought of using a bottom navigation but the idea was dismissed as it would make the application look messy and could cause a problem in case the author decides to add any extra functionality. The navigation drawer is constant throughout the app and the user can anytime switch between the pages allowing them to get to any page whenever they wish to do so.

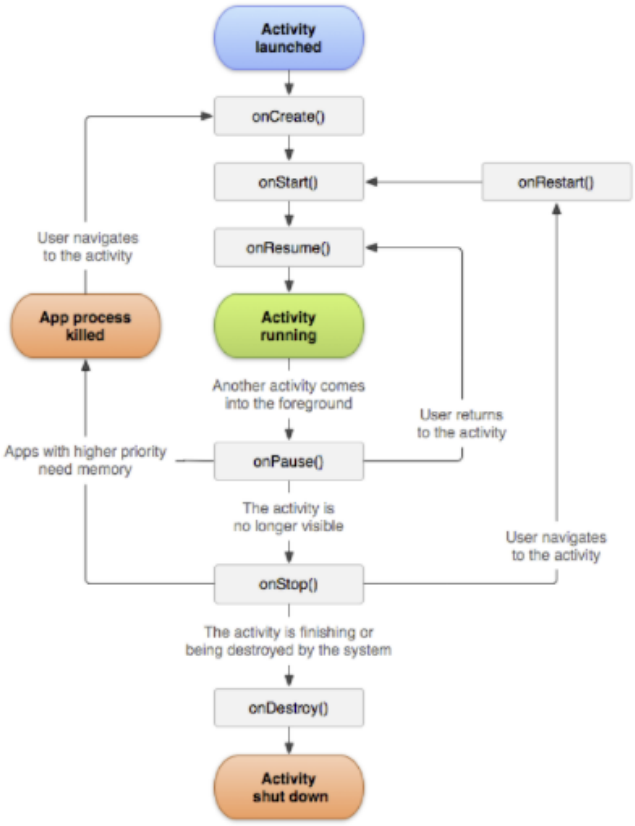


Navigation bar

6.4.1.2 Android Application Life Cycle

While developing the application, an important thing that the author constantly considered was the activity life cycle. While an application is being used, it goes through certain stages which comprise its life cycle. If the life cycle is not implemented appropriately, then this might hamper the application's accessibility and memory-efficiency. For example : Losing the user's progress while switching between other apps or while receiving a phone call or losing on RAM memory even when the user is not actively using the application.

The above mentioned graphs lists a core set of callbacks which is provided in the Activity class and the hierarchy in which it is usually called. These callbacks are invoked whenever the activity enters a new state.

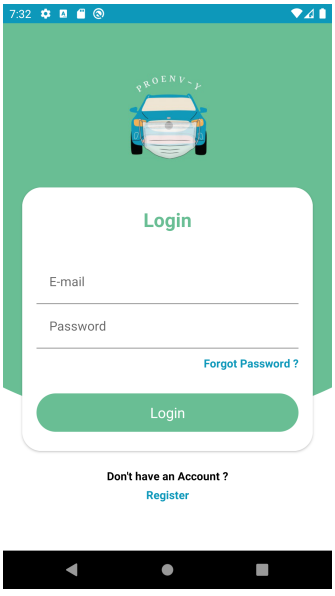


Activity Life Cycle

Core callbacks implemented	Purpose
onCreate()	This is the first method called when the activity is first created and usually comprises all the set up like creating views and etc.
onStart()	onStart() method is called when the activity is in visible state to the users and is followed by onResume().
onResume()	This method is called when the activity starts to interact with the user.
onPause()	This activity is called when another activity comes into the foreground. It is usually followed by onResume() if the activity comes back to front or also followed by onStop() in case the activity is no longer visible to the user.
onStop()	This method is called when the activity is no longer visible to the user i.e When a new pre-existing activity is started on top of this current activity.
onDestroy()	This method is the final method call before the activity is destroyed which may be the result of completion of the activity or the system is momentarily destroying the instance of the activity.
onRestart()	This method is called when the activity is stopped just before starting again.

Table 6.1: Activity Life Cycle

6.4.2 Login Page



Login

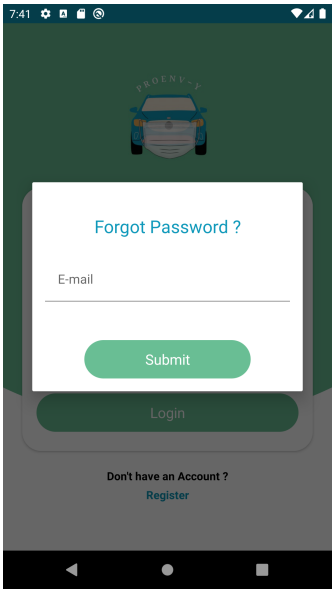
The majority of the code for login activity can be seen under **LoginActivity** in the java folder of the code. The user only requires the email and password credentials if they have

already registered with the application. The other options on the login pages are the option to register if the user does not have an account and the forgot password functionality, in case the user forgets their password credential. Application life cycle was kept in mind and implemented in **LoginActivity** (for example, **OnCreate()** method sets the layout, starts the firebase and calls all the initialising methods required).

```
private void checkUser(String email, String password) {
    progressDialog.show();
    db.collection( collectionPath: "users") CollectionReference
            .whereEqualTo( field: "Email", email) Query
            .whereEqualTo( field: "Password", password)
            .get() Task<QuerySnapshot>
            .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {

                    if (task.isSuccessful()) {
                        for (DocumentSnapshot ds : task.getResult()) {
                            if (ds.getString( field: "Email").equals(email)) {
```

Code Snippet of checkUser()

The **valid()** method contains the java validation for email and password. The **checkUser()** method checks whether the user has an account from the database and sets up the firebase connection to verify users. . The working of the login page is when the user enters his/her credentials and on clicking the submit button, the **checkUser()** and **Valid()** methods are called inside the **setOnClickListener** of **btnLogin** which is inside the **clicks()** method. Using the mentioned methods, the further processing of application takes place and in case of wrong input, the application throws an error respectively. (for example - Account does not exist or Invalid credentials).



Forget Password Dialog

The forget password functionality was added in the same class, where a special dialog was created in which the user can type in the registered email and the password would be displayed. For this a separate method **showForgotPasswordDialog()** and **checkEmail()** were created. **The showForgotPasswordDialog()** method was created to make the dialog and further the same method calls **checkEmail()**, in which the entered email is checked in the database and if the email exists the password is displayed or an error message is shown stating" This email is not found. Please enter your registered email".

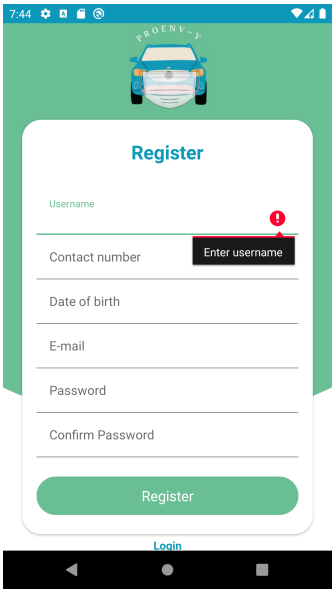
```
public void onComplete(@NonNull Task<QuerySnapshot> task) {
    Log.d(TAG,  msg: "checkingIfusernameExist: checking if username exists");

    if (task.isSuccessful()) {
        for (DocumentSnapshot ds : task.getResult()) {
            if (ds.getString( field: "Email").equals(email)) {
                progressDialog.dismiss();
                Constant.showAlertDailogBox( context: LoginActivity.this,  msg: "Your Password is : "+ds.get("Password"));
                dialog.dismiss();
```

Code Snippet of checkEmail()

The XML layout of this page can be seen under **activity_login.xml** . Creating the XML layout for the application was fairly simple as the author had created the prototypes before implementing the code which had given him a clear head scope of all the buttons and text view options to be implemented. The layout was simple and yet stylish. All the buttons and

content that needed to be displayed were placed according using the draw and drop feature of Android studio as well as coded.

6.4.3 Register Page



Register

Moving onto the register page. The register page is accessed by the user to make their account for the application in case they are a new user. The code for the register user can be seen under **RegisterActivity** in the activity folder. The text fields to be filled by the users are:

- Name
- Email
- Phone Number
- Date of Birth
- Password
- Confirm Password

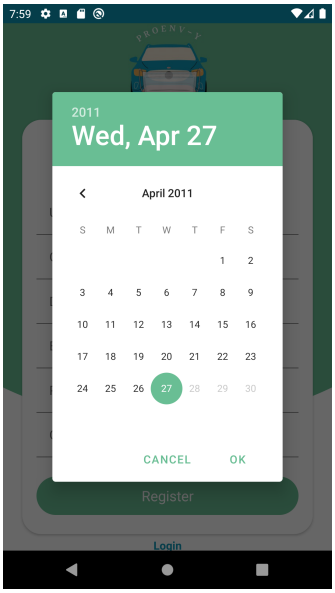
The user can also get back to the login page if he/she realises that they already have an existing account. Initially as soon as the user presses on the register button made available on the login page,the **RegisterActivity** is invoked and the **onCreate()** method is called

automatically. The **onCreate()** method initialises the layout and calls the other methods like **inits()** (initialises the buttons and other fields with the ids) and **clicks()** (sets up the setOnclickListener for the buttons available). Similar to **LoginActivity**, **RegisterActivity** also has **checkEmail()** and **valid()** methods. The **valid()** method is to check if all the data filled up by the user in the compulsory text fields are in the correct format (for example, if the email is correct or password is acceptable by the application.)The **checkEmail()** is intended to see if the user also has an account with the email address they are trying to register again before it invokes the **registerDataToCLoud()** method. **registerDataToCLoud()** method is one of the most important methods of this activity as it handles the data storing in the backend - firebase. The application also throws in the respective error messages for example, if the account already exists, so that the user is fully aware of what is going wrong with his/her process, hence increasing usability.

```
/**
 * Method for register data to firebase
 */
private void registerDataToCloud(String username, String email, String password, String dob, String contact) {
    Map<String, Object> user = new HashMap<>();
    user.put("Username", username);
    user.put("Email", email);
    user.put("Password", password);
    user.put("DOB", dob);
    user.put("Contact", contact);

    db.collection( collectionPath: "users")
            .document( documentPath: "user" + System.currentTimeMillis() / 1000).set(user)
            .addOnSuccessListener(aVoid -> {
                Toast.makeText( context: RegisterActivity.this,  text: "Registration successfully", Toast.LENGTH_SHORT).show();
                progressDialog.dismiss();
                startActivity(new Intent(getApplicationContext(), LoginActivity.class));
                finish();

            }).addOnFailureListener(e -> {
        Log.e(TAG,  msg: "registerDataToCloud: " + e.getMessage());
        progressDialog.dismiss();
       });
}
```

Code Snippet of registerDatatoCloud()

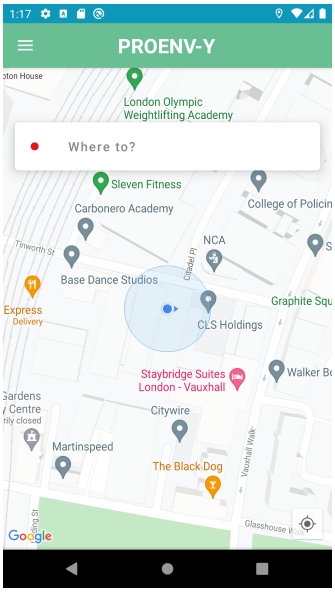
The author also made sure to put a calendar in the Date of Birth option to allow the users to easily fill in the date of birth. The calendar was implemented by having an **onclickListener** for DOB field which implements the calendar view. The minimum age was set to 10 by the author but this could be changed at any moment.



Date Picker for Date of Birth Field

The xml file used in this activity is quite similar to the one used for login, to maintain a standard design throughout the application. The author also made sure to include the progress dialog in the application allowing the user to see when the processing is happening. The progress dialog is dismissed and shown respectively whenever the process is happening. For example, when the application is storing the data on firebase, the progress dialog is visible and as soon as the processing is over the dialog is dismissed with the message " REGISTRATION SUCCESSFUL".

6.4.4 Home Page



Home Screen post logging in

The most important aspect of this application was the implementation of map. The author wanted to provide the best possible graphical user interface to the users, as this is the feature for which everyone would be downloading the application. The map comes with a lot of other functionalities like animation and other features for the map. These functionalities are spread across 3 main activity classes:

- **HomeActivity**
- **SearchActivity**
- **PickLocation**

HomeActivity has the main functionalities of the application. The home page has an integrated map and the other 2 activities mentioned above (SearchActivity and PickLocation) are solely used for app functionalities like picking up the location through filling in the dialog and through pin pointers as well as the data that needs to be stored for the formation of route and implementation of moving vehicle animation.
Throughout these classes, the author has put in a lot of animation like bottom to up, up to bottom, transit in and transit out to enhance the GUI of the application. The author has also given special attention to the use of back buttons on phones and accordingly created different outcomes for the same. For example, if the user presses the back button after setting

up the route and selecting the vehicle, the author checks if the map or carmarkers are present and removes them accordingly. Throughout the application, as mentioned before the boiler plate structure, so accordingly the utils and adapters were created. The utils were created to retrieve data and this helped in increasing the readability of the code whereas adapters for listing data.

In the above mentioned classes, the author has also implemented a few interfaces to allow him to use these methods. (For example: **OnMapReadycallBack**, side navigation call listener through **NavigationView.OnNavigationItemSelectedListener** and few others for google map related interfaces. **AppCompactActivity** is the base activity of this application.

Home page is the first page that opens up as soon as the user logs into the application. As the author tried to follow the iterative waterfall model, the use of Boilerplate structure proved to be really helpful. The implementation of the home page was done in stages. The use of prototypes created beforehand proved to be really helpful to the author as he had a sound idea design that needed to be followed throughout the application. The main content layout used here can be seen under **activity_home.xml** in the res folder. The **HomeActivity** has the following functionalities:

- Navigation drawer button on toolbar
- Destination card above the map
- The map
- Map Animation
- Trip dialog

6.4.4.1 Navigation

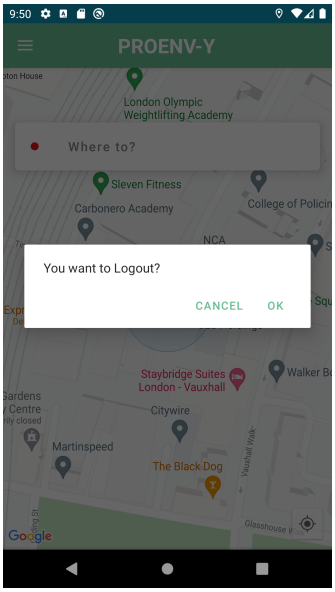
Coming to the application navigation. The app navigation was created in the **HomeActivity** class under **onCreate()** method. First of all the author had set the view of the activity followed by which the author had implemented the navigation bar. For the navigation bar, first the author had made an object of **DrawerLayout** and then set the navigation view. Followed by this, a toggle was created to allow reopening and closing of the navigation. The author did not wish to keep the navigation bar with just the buttons to shift to other pages but also make it appealing. On this thought the author created a header view for the navi-

gation bar and in this the author wished to display the application logo, user's name and his email. These were the initializing and setting up the navigation.

Moving onto the implementation of the navigation drawer. The author had created an xml file called **activity_main_drawer.xml** in which he made the clickable text of all the options available in the navigation bar. Later in the homepage under **onNavigationItemSelected()** method, the use of if else ladder and these clickable texts were given utilities like opening up new functionality by using startActivity and intent. The logout functionality was also implemented inside the navigation drawer and the author has also made sure to give up a pop up dialog seeking for logout confirmation.

```
if (id == R.id.nav_home) {
    startActivity(new Intent(getApplicationContext(), HomeActivity.class).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TA
} else if (id == R.id.nav_Profile) {
    startActivity(new Intent(getApplicationContext(), UpdateProfileActivity.class));
} else if (id == R.id.nav_Score_Board) {
    startActivity(new Intent(getApplicationContext(), StatisticsActivity.class));
} else if (id == R.id.nav_Change_Password) {
    startActivity(new Intent(getApplicationContext(), changepasswordactivity.class));
} else if (id == R.id.nav_Logout) {
    new AlertDialog.Builder( context: HomeActivity.this)
            .setMessage("You want to Logout?")
            .setCancelable(false)
            .setPositiveButton(android.R.string.yes, (dialog, which) -> {
                clearShaedPref( context: HomeActivity.this);
                startActivity(new Intent(getApplicationContext(), LoginActivity.class).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTI
```

Code Snippet of onNavigationItemSelected()



Logout Dialog

6.4.4.2 Destination Card

The purpose of keeping a destination card was to allow the users to enter the destination address. The destination card on the home page serves as the link to **SearchActivity** followed to **PickLocation** activity, in which the user is able to set the start point and the end point of the journey. The **destcard** comprises an image, denoting the destination(red dot) and the dest button, which is the test view with a hint as Where to. This can be found under **app_bar_map.xml** in the res folder.

Destination card

6.4.4.3 Map

It was extremely essential for the application to have a map on the home page, as the author feels that it's an important indication to the users about the purpose of the application. Initially just the map was created but later the author had decided to make use of the map and added moving vehicle animations to the same. To begin with the map implementation, the map was included in the **onCreate()** method of HomeActivity class using the **mapFragment**.In the **mapFragement** variable, the map was created using the **getSupportFragmentManager**. It returns the **FragmentManager** for linking with fragments connected with the activity. Followed by this the code checks if the **mapFragment** variable is not null and calls **getMapAsync** to load the map.

```
mapFragment = (SupportMapFragment) getSupportFragmentManager()
        .findFragmentById(R.id.map);
if (mapFragment != null) {
    mapFragment.getMapAsync( onMapReadyCallback: this);
}
```

Code Snippet of map creation

As soon as the map is loaded using the above mentioned code, the **onMapReady()** method

is called. This method enables the google map to be ready and checks for other things like permissions and etc.. and initialises the map. The **onMapReady()** method also calls the **setOnCameraIdleListener** to focus on the location entered by the user and calls the **setUpClusterer()** method. The **SetUpClusterer()** method was used to manage the cluster for animation related work.

```
public void onMapReady(GoogleMap googleMap) {
    this.googleMap = googleMap;
    this.googleMap.getUiSettings().setCompassEnabled(false);
    this.googleMap.getUiSettings().setMapToolbarEnabled(false);

    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission
            ( context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {...}
    this.googleMap.setMyLocationEnabled(true);

    View mapView = mapFragment.getView();
    if (mapView != null && mapView.findViewById(1) != null) {...}


    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {...}

    setUpClusterer();
}
```

Code Snippet of onMapReady()

Another important aspect was how will the author access the google map API for location related purposes. To access the **GoogleMapAPI** , initially a variable of **GoogleApiClient** was made and then a method **buildGoogleAPIClient()** was implemented. The method **buildGoogleAPIClient()** is called in the **onStart()** method of **HomeActivity**, as it is important for the API to be active throughout the application. The method **buildGoogleAPIClient()** after creating the google API client calls the connect method. The **onConnect()** method is called as soon as the connection is established which further sets up the API client with location and checks for permission.

```
/** Method for build google api client */
protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder( context: this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
    mGoogleApiClient.connect();
}
```

Code Snippet of buildGoogleApiClient()

Moving onto the permission request, method **makeLocationPermissionRequest()** and **onRequestPermissionResult()** were implemented. It is not possible for the application to function without the location request, so in case if it is not granted the application will seek permission again. Throughout the activity attention was given towards the activity life cycle, and similarly **onResume()**, **onStop()** methods were implemented in which the code checks for permissions and disconnects the API client. It was essential for the **onResume()** method to check for permission as it is absolutely important for phones with android 6 Marshmallow and above, to seek for run-time permissions.

```
public void onResume() {
    super.onResume();

    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        if (mGoogleApiClient != null && mGoogleApiClient.isConnected()) {
            LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient, mLocationRequest, locationListener: this);
        }
    }
```

Code Snippet onResume()

As soon as the user sets up the start location and destination from the **PickLocation** and **SearchActivity** respectively, the **onPlace()** method in **HomeActivity** is called. The **onPlace()** method firstly checks if the place is available. If the place is available, a direction of URL is created.

```java
@Override
public void onPlace(Place place) {
    this.placeGet = place;
    MapUtils.createBottomUpAnimation( context: this, destCard, bottomup);

    DisplayMetrics metrics = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(metrics);

    btn_proceed.setVisibility(View.VISIBLE);

    btn_proceed.setOnClickListener(view -> {
        if(SystemClock.elapsedRealtime() - lastTimeClick < defaultInterval){...}
        lastTimeClick = SystemClock.elapsedRealtime();
        if (btn_proceed.getText().toString().equals("Start")) {
            new Handler().postDelayed(() -> showMovingCab(polyLineList),  delayMillis: 100);
            linMode.setVisibility(View.GONE);
            btn_proceed.setText("Skip");
        } else {
            TripDialog();
        }
    });

    if (place.getSrcLatLng() != null) {
        String url = getDirectionsUrl(place.getSrcLatLng(), place.getDesrLatLng(),  mapType: "driving");
        DownloadTask downloadTask = new DownloadTask();
        downloadTask.execute(url);
    }
}
```

Code Snippet for onPlace()

To create the URL, the author has implemented a **getDirectionsURL** in which the latitude and longitude of start location and destination as well as map type is passed. A string was used to combine the whole URL along with google map API key.

```java
/** Method for prepare direction url */
private String getDirectionsUrl(LatLng origin, LatLng dest, String mapType) {
    String url = "";
    String strOrigin = "origin=" + origin.latitude + "," + origin.longitude;
    String strDest = "destination=" + dest.latitude + "," + dest.longitude;
    String strMode = "";
    String strKey = "key=" + "AIzaSyClEy7iOrtMOyD7_hLoALTz6o_1vur7mYc";

    if (mapType.equals("driving")) {
        strMode = "mode=driving";
        url = "https://maps.googleapis.com/maps/api/directions/json?" + strOrigin + "&" + strDest + "&" + strMode + "&" + strKey;
    } else {...}
    return url;
}
```

Code Snippet for getDirectionUrl()

The **setOnclicklistener** of the proceed button first checks the get of the button, if the text of the button is start, this means the page is still existing and the **showMovingCab ()** method (Discussed in section 6.4.4.4) is called and the polyLineList is passed and if not the

TripDialog() method is called. The TripDialog()method details can be seen in section 6.4.4.5. Since, now the URL was ready, the next step to be executed was to download the data with the use of URL.

A separate **DownloadTask** inner class was used and an object was created in the **OnPlace()** method for the same and the same was called.

```
/** Method for Aynctask to get direction data */
@SuppressLint("StaticFieldLeak")
private class DownloadTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... url) {
        String data = "";
        try {
            data = downloadUrl(url[0]);
        } catch (Exception e) {
            Log.d( tag: "Background Task", e.toString());
        }
        return data;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);
        ParserTask parserTask = new ParserTask();
        parserTask.execute(result);
    }
}
```

Code Snippet for DownloadTask

The data was now downloaded, the next step involved the parsing of data into the required format. For this, **another parsing innerclass** was used which was called by creating an object in the **downloadTask** URL.

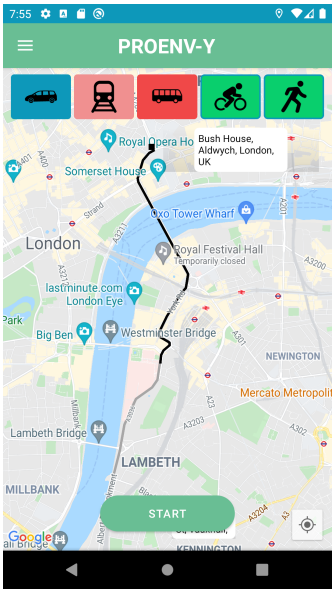
```
/** Method for Asyntask to parse direction data */
@SuppressLint("StaticFieldLeak")
private class ParserTask extends AsyncTask<String, Integer, List<List<HashMap<String, String>>>> {

    @Override
    protected List<List<HashMap<String, String>>> doInBackground(String... jsonData) {

        JSONObject jObject;
        List<List<HashMap<String, String>>> routes = null;

        try {
            jObject = new JSONObject(jsonData[0]);
            if (!jObject.getString( name: "status").equals("ZERO_RESULTS")) {
                DirectionsJSONParser parser = new DirectionsJSONParser();
                Log.e( tag: "JsonObject", jObject.toString());
                routes = parser.parse(jObject);
            } else {
                new Thread() {...}.start();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return routes;
    }
```

Code Snippet for ParserTask

After parsing the data, the next step involved copying the data into **polyLineList** which is a list of latitudes and longitudes. This was done under the **onPostExecute()** method of parser class and after copying the parsed data into polyLineList, the **staticPolyLine ()** method was called. Now the author had all the data needed by him for the further execution of the program. It is worth noting that the download and parsing inner classes are **async** and have been happening simultaneously.

The **staticPolyLine()** method first clears the google map, sets the camera, makes the route and calls the **startCarAnimation ()** method. The route between the source and destination is made by the code **MapAnimator.getInstance().animateRoute(googleMap, polyLineList)**

6.4.4.4 Map Animation



Route Drawn

Moving Cab Animation

The above shown image portrays the use of **startCarAnimation()** method. The options of different modes of transport were created beforehand in the layout (linMode and fab) but the visibility was disabled. As soon as the route is made (as shown in the screen image above) the linMode and fab (layouts) options are also made visible inside the **startCarAnimation ()** method. To promote the use of eco-friendly mode of transport, cycling and walking buttons were given green colour and expect car, other's were in increasing shade of red.

The darker the shade of red the more is the co2 emissions.

The **startCarAnimation ()** method basically sets up all the icons that can be seen in the map above. The text view on the map for source and destination with the addresses are created here. Another important thing that the author realised during the implementation of application, the options like walking and cycling were also available for large distances. This would not make sense, so the author decided to use an if else ladder in which basically when the distance is above 50 kms, the option to select the walk/cycle option is not permitted and the option of flight will only be available above 500kms.

```
double tempDistance = 0.0;
tempDistance = Double.parseDouble(RideDistance.replace( target: " km",  replacement: ""))

if (tempDistance > 500) {
    btn_Flight.setVisibility(View.VISIBLE);
    btn_Walking.setVisibility(View.GONE);
    btn_cycle.setVisibility(View.GONE);
} else if (tempDistance > 50 && tempDistance <= 500) {
    btn_Flight.setVisibility(View.GONE);
    btn_Walking.setVisibility(View.GONE);
    btn_cycle.setVisibility(View.GONE);
} else {...}

linMode.setVisibility(View.VISIBLE);
```

Code Snippet for the conditions on mode of transport buttons in startCarAnimation()

These conditions can easily be changed in the **startAnimation()** method itself. It also sets the visibility of proceed button to true. This was implemented here by the author, as it would allow the map data to be completely downloaded before the user clicks on the start button. It usually takes a little longer for large routes(above 500 Kms) to collect data. Lastly, the start car animation sets the visibility of linear layout which comprises all the vehicle selection buttons. Going back to the **onCreate()** method, all the modes of transport buttons were given a separate **onClickListener**. There are different routes available for different modes of transport, for example - Driving for cars and walking route. To implement this, in the **setOnclickListener**, similar to the **onPlace()** method the URL was created and downloaded. The difference here being that the map type passed into the **getDirectionsUrl()** was walking for the mode of transport - cycling and walking. The **setOnClickListener** also assigns the marker flag for each mode of transport which will be used later.

```
btn_Bus.setOnClickListener(view -> {
    markerFlag = "BUS";
    btn_Bus.setSelected(true);
    btn_car.setSelected(false);
    btn_Train.setSelected(false);
    btn_Flight.setSelected(false);
    btn_Walking.setSelected(false);
    btn_cycle.setSelected(false);
    if (placeGet.getSrcLatLng() != null) {
        String url = getDirectionsUrl(placeGet.getSrcLatLng(), placeGet.getDesrLatLng(), mapType: "driving");
        DownloadTask downloadTask = new DownloadTask();
        downloadTask.execute(url);
    }
});
```

Code Snippet for setOnClickListener for Bus button

Currently the Google MAP does not allow the implementation of train,bus or flight route, but in case it ever does, this could be easily changed here. Currently the bus, train and flight function on the same driving map type as the car. The actual implementation of animation begins when the user clicks on the Start button in **OnPlace()** activity and **showMoving-Cab()** method is called.

```
/** Method for show car animation & moving cab */
private void showMovingCab(final List<LatLng> cabLatLngList) {
    handler = new Handler();
    runnable = () -> {
        if (cabLatLngList != null && index < cabLatLngList.size()) {
            updateCarLocation(cabLatLngList.get(index));
            handler.postDelayed(runnable, delayMillis: 900);
            ++index;
        } else {
            handler.removeCallbacks(runnable);
            TripDialog();
        }
    };
    handler.postDelayed(runnable, delayMillis: 1500);
}
```

Code Snippet for showMovingCab()

In the **showMovingCab()** method, a runnable was created inside a handler, which works as a loop and in every 1500 milliseconds, the marker on the map (For eg, the car, or flight) updates its location. To implement the same, a method known as **updateCarLocation()** method is called.

```
/** Method for update pinpoint location to show animation */
private void updateCarLocation(LatLng latLng) {
    if (movingCabMarker == null) {
        movingCabMarker = addCarMarkerAndGet(latLng);
    }
    if (previousLatLng == null) {
        currentLatLng = latLng;
        previousLatLng = currentLatLng;
        movingCabMarker.setPosition(currentLatLng);
        movingCabMarker.setAnchor( v: 0.5f,  v1: 0.5f);
    } else {...}
    animateCamera(latLng);
}
```

Code Snippet for updateCarLocation()

The actual working of moving vehicle animation is based on the route formed. The route formed is stored in a polyline list. Now for example, if the user chooses the location as Strand and Guy's campus. This polyline list will not only include the latlng of these 2 places. It will consist of a list of latlngs. The source and destination are broken down into many parts. For example, from Stand to Shard will comprise of Waterloo bridge, blackfriars, Borough tube station and other locations in between. The animation implemented in the application uses the same concept. Using the same example as above, first the vehicle is moved from strand to waterloo bridge followed by waterloo bridge to BlackFiars and so on until it reaches the set destination. In this method if the marker is null then the marker was added, if previous latlng is null, then the animations starts from the starting and else the previous LatLng was initialized with the currentLatLng and the currentLanLng was initialized to the new latLng passed in the **showMovingCab()** method. The **updateCarLocation()** method called in **showMovingCab()** only passes one latLng and not the whole list.

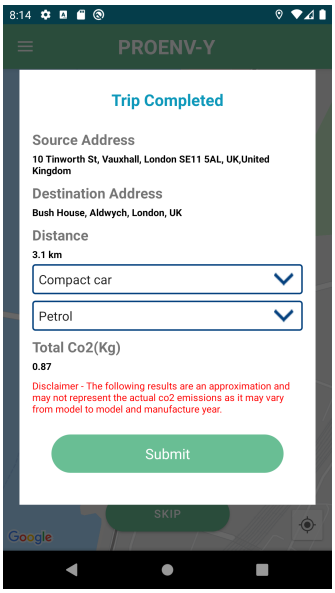
Next challenger during the implementation of animation was the different mode of transport. The author was determined to use separate icons for each mode of transport. To do this, another method was created called **addCarMarkerAndGet()**. This method returns a marker and the specific latLng is passed in the method call inside **updateCarLocation()**. Now the role of markerFlags assigned to **onClickListener** of every mode of transport comes into action. Basically this method, checks for the marker flags and similarly adds it on the map at the specific latLng.

```
/** Method for get marker on basis of selection */
private Marker addCarMarkerAndGet(LatLng latLng) {

    if (markerFlag.equals("CAR")) {
        return googleMap.addMarker(new MarkerOptions().position(latLng).anchor( v: 0.5f,  v1: 0.5f).flat(true).icon(BitmapDescriptorFactory.fromResource(R.mipmap.new_car_small)));

    } else if (markerFlag.equals("BUS")) {
        return googleMap.addMarker(new MarkerOptions().position(latLng).anchor( v: 0.5f,  v1: 0.5f).flat(true).icon(BitmapDescriptorFactory.fromResource(R.mipmap.ic_bus)));
    } else if (markerFlag.equals("TRAIN")) {
        return googleMap.addMarker(new MarkerOptions().position(latLng).anchor( v: 0.5f,  v1: 0.5f).flat(true).icon(BitmapDescriptorFactory.fromResource(R.mipmap.ic_train)));
    } else if (markerFlag.equals("FLIGHT")) {
        return googleMap.addMarker(new MarkerOptions().position(latLng).anchor( v: 0.5f,  v1: 0.5f).flat(true).icon(BitmapDescriptorFactory.fromResource(R.mipmap.ic_air_plan)));
    } else if (markerFlag.equals("WALK")) {
        return googleMap.addMarker(new MarkerOptions().position(latLng).anchor( v: 0.5f,  v1: 0.5f).flat(true).icon(BitmapDescriptorFactory.fromResource(R.mipmap.ic_man)));
    } else
        return googleMap.addMarker(new MarkerOptions().position(latLng).anchor( v: 0.5f,  v1: 0.5f).flat(true).icon(BitmapDescriptorFactory.fromResource(R.mipmap.ic_bike)));
}
```

Code Snippet for addCarMarketGet()

Further to this, an object of **valueAnimator** was made which provided a simple timing engine for running application and with the **animateCamera** method the camera positions were set. Overall the camera position was handled by **onCameraIdleListener**.

6.4.4.5 Trip Dialog



Trip Dialog for Car

Trip dialog is shown whenever the user presses the skip button when the animation begins or as soon as the animation ends (that is the vehicle reaches its targeted destination). The implementation of trip dialog can be seen under **TripDialog()** method. In this, a dialog was created and the height width of the mobile phone is taken. The layout of the dialog can be seen under **dialog_trip_complate**. All the widgets inside the dialog box were set. The application besides just calculating the co2 emissions, it also calculates the approximate calories

burnt by a person when using mode of transport like walking and cycling. So accordingly, it was important for the dialog to have specific data for specific vehicles. For example, the dialog should show calories instead of co2 emitted in the dialog box when the chosen mode of transport is walking. Coming to the car selection, the author had decided to put 2 spinners through which first the user could select the car types and then the fuel type. As the user enters the details, the co2 emission changes. The calculation of co2 for cars was specifically implemented in **calculateCO2()** method. In this method, basically by using the if else ladder the car model type and fuel type was checked and accordingly a specific value (basicCo2) which is the per kilometer co2 emissions was set and later the calculation of co2 was done and sent back to the **tripDialog()** method.

```
private double calculateCO2(String distance) {

    double basicValue = 0;
    double co2value = 0;
    if (FuelSelectedPos == 0 && VehicleSelectedPos == 0) {
        basicValue = 0.28;
    } else if (FuelSelectedPos == 0 && VehicleSelectedPos == 1) {
        basicValue = 0.34;
    } else if (FuelSelectedPos == 0 && VehicleSelectedPos == 2) {
        basicValue = 0.41;
    } else if (FuelSelectedPos == 1 && VehicleSelectedPos == 0) {
        basicValue = 0.24;
    } else if (FuelSelectedPos == 1 && VehicleSelectedPos == 1) {
        basicValue = 0.31;
    } else if (FuelSelectedPos == 1 && VehicleSelectedPos == 2) {
        basicValue = 0.39;
    } else if (FuelSelectedPos == 2 && VehicleSelectedPos == 0) {
        basicValue = 0.21;
    } else if (FuelSelectedPos == 2 && VehicleSelectedPos == 1) {
        basicValue = 0.26;
    } else if (FuelSelectedPos == 2 && VehicleSelectedPos == 2) {
        basicValue = 0.30;
    } else if (FuelSelectedPos == 3 && VehicleSelectedPos == 0) {
        basicValue = 0.19;
    } else if (FuelSelectedPos == 3 && VehicleSelectedPos == 1) {
        basicValue = 0.24;
    } else if (FuelSelectedPos == 3 && VehicleSelectedPos == 2) {
        basicValue = 0.28;
    } else if (FuelSelectedPos == 4 && VehicleSelectedPos == 0) {
        basicValue = 0.07;
    } else if (FuelSelectedPos == 4 && VehicleSelectedPos == 1) {
        basicValue = 0.08;
    } else if (FuelSelectedPos == 4 && VehicleSelectedPos == 2) {
        basicValue = 0.10;
    }

    if (distance.contains(" km")) {
        co2value = basicValue * Double.parseDouble(distance.replace( target: " km",  replacement: ""));
    } else {
        co2value = basicValue * (Double.parseDouble(distance.replace( target: " m",  replacement: "")) / 1000);
    }
    return co2value;
}
```

Code Snippet for calculatecCO2()

Similarly, in the **tripDialog()** method, the basicCO2 was set for each vehicle except the cars. The details to be entered in the dialog boxes were set. The author has also added the disclaimer messages for vehicles and an additional disclaimer message for cars which consume electricity. After all the selection, in the bottom of the dialog the users can see a submit button. As soon as the user clicks on the button, the data is saved in the database using the **AddRecord()** method. This method basically just sets up the firebase and adds all the trip dialog details to the firebase which can be seen in the statistics page.

```
/**
 * Method for add record in firebase
 */
private void AddTripRecord(String Source, String Destination, String Distance, String Co2, Dialog dialog, Boolean isCo2,
                           String FuelType, String VehicleType, String rideType) {

    String date = new SimpleDateFormat( pattern: "dd-MM-yyyy", Locale.getDefault()).format(new Date());
    Map<String, Object> user = new HashMap<>();
    user.put("UserID", getShaedPref( context: HomeActivity.this, spUserId));
    user.put("Source", Source);
    user.put("Destination", Destination);
    user.put("Distance", Distance);
    user.put("Co2Type", isCo2);
    user.put("Co2", Co2);
    user.put("RideType", rideType);
    if (markerFlag.equals("CAR")) {
        user.put("VehicleType", VehicleType);
        user.put("FuelType", FuelType);
    } else {
        user.put("VehicleType", "");
        user.put("FuelType", "");
    }
    user.put("Date", date);
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    db.collection( collectionPath: "Co2-Report")
            .document( documentPath: "Co2-Report" + System.currentTimeMillis() / 1000).set(user)
            .addOnSuccessListener(aVoid -> {
                dialog.dismiss();
                startActivity(new Intent(getApplicationContext(), HomeActivity.class).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK));

            }).addOnFailureListener(e -> {
        Log.e(TAG,  msg: "registerDataToCloud: " + e.getMessage());
        progressDialog.dismiss();
    });
}
```

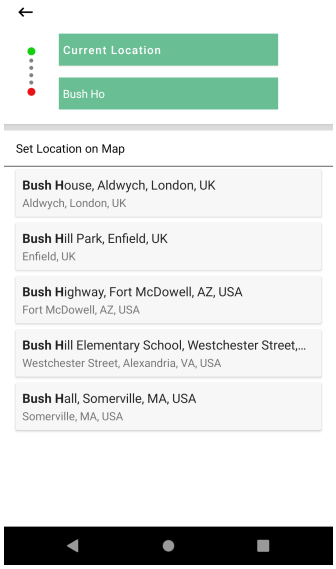
Code Snippet for AddTripRecorded()

6.4.5 Search Activity

Home Search

The class **SearchActivity** is implemented by the author to allow the user to set their destination. The **SearchActivity** comes to action as the user clicks on the destination card on the home page.

On clicking at the dest card, the above shown screen pops up where the use of **SearchActivity** comes into existence.The GUI components used in this activity can be found under activity_search.xml. Moving onto the implementation of the **SearchActivity**. First of all the author made sure to use the **PlaceAutoCompleteInterface**. This interface would allow the users to view the recommended places when they are typing in a destination address. The application also uses few animations here for which few methods were created throughout the activity. The main implementation of the code begins with **onCreate ()** method. Firstly, the **onCreate()** method checks for the width and height of the phone and with the help of a bundle, the data of the previous screen is passed on here. If the bundle is not empty, then the latitude and longitude is fetched. Followed by which an object of **GoogleAPIClient** is made and the contents of the screen is set. The GUI components used in this activity can be found under **activity_search.xml**. Similar to the **HomeActivity dest button**, the current location option on the screen also serves as a button link to the **PickLocation** activity and the **onClickListener** was set. The author wished to show the recommended destination as a drop down while the user types the location and in order to implement that **mRecycleView** and **mAdapter** were created.



Drop down list of recommended places

These are used to display the suggested places. To list down the suggested places, the author has used an adapter called **placeAutoComplete** adapter. The destination part is used as EditText which is given an **onClickListener** in which the input functionality and animations are activated. It was necessary to also implement the **addTextChangedListener**. The **addTextChangedListener** was added to allow the drop down menu of suggested places to show below when the user keeps typing in the location. In this further checks were implemented like if the text in the destination dialog is empty, the **mAdapter** was cleared and if there is some texts, the application checks if the map is connected and accordingly the adapter methods using the object (**mAdapter**) were filtered and using the **mRecyclerView** locations were displayed.

```
mSearchEdittext.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        if (count > 0) {
            if (mAdapter != null) {
                mRecyclerView.setAdapter(mAdapter);
                Log.e( tag: "mAdapterSize",  msg: mAdapter.getItemCount() + "");
            }
        } else {
            if (mAdapter != null) {
                mAdapter.clearList();
                mAdapter.notifyDataSetChanged();
                mRecyclerView.setAdapter(mAdapter);
                Log.e( tag: "mAdapterSize1",  msg: mAdapter.getItemCount() + "");
            }
        }
        if (!s.toString().equals("") && mClient.isConnected()) {
            mAdapter.getFilter().filter(s.toString());
        }
    }

    @Override
    public void afterTextChanged(Editable s) {

    }
});
```

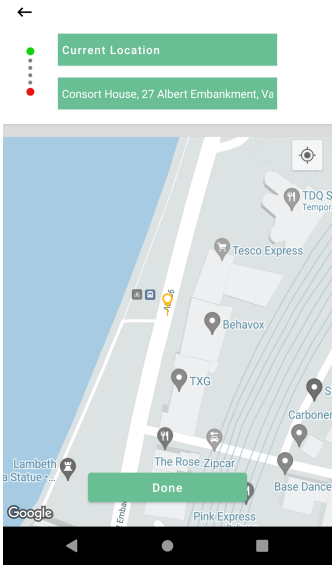
Code Snippet of addTextChangedListener

It was essential for application to take the address of the location apart from the latitude and longitude, the author used a **geocoder** in the same **onCreate()** method. **Geocoder** is a service which is used for conversion between address and coordinates of a location. With the use of **GeoCoder** the address was retrieved and was later stored in a string. When the user chooses a suggested place from the drop down menu, to retrieve the value of the destination address chosen, a method **onPlaceClick()** is implemented. The **onPlaceClick()** method basically fetches the data of that pin pointer location and goes back to the previous activity. The **onPlaceClick()** method also calls another method called **getLocationFromAddress()**. This method is used because the author required address details along with location(Longitude and latitude). The **getLocationFromAddress()** method is used to get the location of the address when the user selects the location on map. This method was again implemented with the use of **geocoder**.

```
LatLng ladw = new LatLng(loc.latitude, loc.longitude);
try {
    Geocoder selected_place_geocoder = new Geocoder( context: SearchActivity.this);
    List<Address> address;
    address = selected_place_geocoder.getFromLocation(ladw.latitude, ladw.longitude,  maxResults: 1);
    Address location = address.get(0);
    if (location != null) {
        if (location.getLocality() != null) {
            srcAdd = location.getAddressLine( index: 0) + "," + location.getLocality() + "," + location.getCountryName();
        } else {
            srcAdd = location.getAddressLine( index: 0) + "," + location.getCountryName();
        }
    }
} catch (Exception e) {
    e.printStackTrace();
```

Code Snippet of geocoder inside onCreate()

To easen up the storage of data, the author had created utils like **CustomModel** which has an object of **placeListener** and the place data could easily be retrieved later.The **onPlaceClick()** method finally calls the **back()** method.



Set Location on Map screen

Next important functionality that was added in the **SearchActivity** was to allow the users to choose location on map. For this a LinearLayout was set with an **onclickListener**. For a smooth transition into map, in the **onCLickListener** animations were created which disabled the frame, the visibility of **mRecycleView** was cleared and most importantly, the map was created. The frame is made active as soon as the animation to set location on map is ended. The frame contains the pin pointer used for drag and drop on the screen. When the user chooses to choose the destination through map, a pin pointer is made available and that

can be placed anywhere on the map.This part is implemented inside the **onMapReady()** method. As soon as the user drops in the pin-pointer, **onCameraIdle** method is called. The **onCameraIdle()** method first of all stores the latitude and longitude of the pin pointer. While using the pin-pointer, the location is retrieved and to retrieve the address, again the **geocoder** was used. The address was constructed and then this address is set to the edit-Text option. As the user chooses a pin-pointer location, simultaneously a done button is set visible. Similar code that of onPlaceClick was added to the **setOnclickListener** to be done which would save the data and call **back()** method.

```
done.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (done.getText().toString().equalsIgnoreCase( anotherString: "Done")) {
            if (placeCust != null) {
                Place place = new Place(new LatLng(placeCust.srcLat, placeCust.srcLon), googleMap.getCameraPosition().target);
                place.setSrcAddress(placeCust.getSrcAddress());
                place.setDestAddress(mSearchEdittext.getText().toString());
                CustomModel.getInstance().changeState(place);
            } else {
                Place place = new Place(new LatLng(loc.latitude, loc.longitude), googleMap.getCameraPosition().target);
                place.setSrcAddress(srcAdd);
                place.setDestAddress(mSearchEdittext.getText().toString());
                CustomModel.getInstance().changeState(place);
            }
            onBack();
        } else {
            Toast.makeText(getApplicationContext(), text: "Wait for Location to be fetched.", Toast.LENGTH_SHORT).show();
        }
    }
});
```

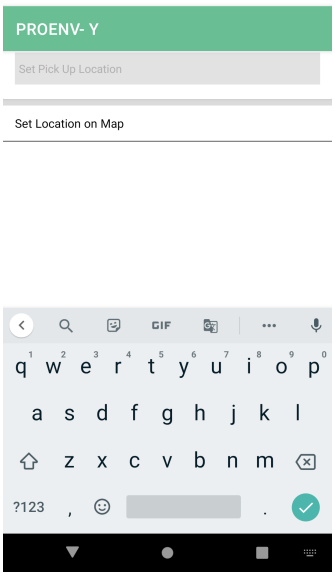
Code Snippet for onClick() for done button

The **onActivityResult()** method is called whenever the user tries to choose the location. It has 3 conditions:

- If the result of location is correct, all the data from the place is retrieved
- If the result of location is incorrect, an error is being displayed
- If the result of location is canceled, this means the user has canceled the operation.

The **onLocationChange()** method is called whenever the user changes the location.

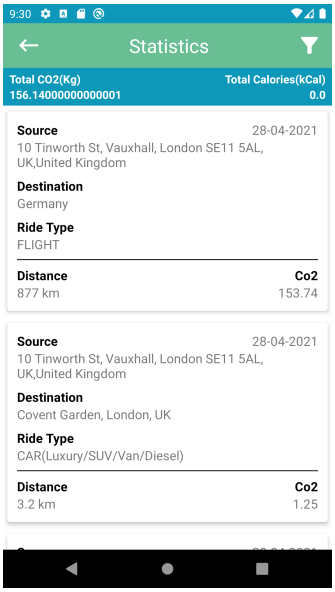
6.4.6 PickLocation



Pick Location

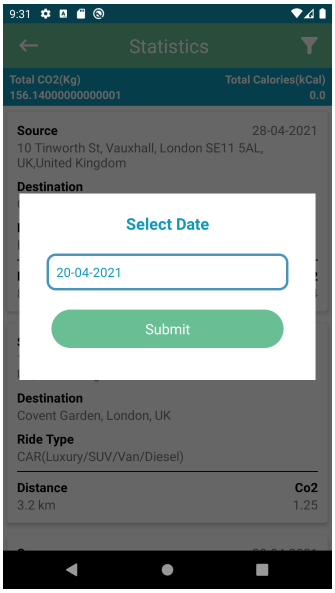
The implementation of **PickLocation** is very similar to the **SearchActivity** (Section 6.4.5). The purpose of this activity is to allow the users to change their start location instead of current location. The back method of **PickLocation** calls back to the **SearchActivity** where the destination can be chosen. The GUI components used in this activity can be found under **activity_pick.xml**.

6.4.7 Statistic Page



Statistics Home

Statistic activity comes into action when the user clicks on the statistics option in the navigation bar. The majority of code for the statistics page can be seen under **StatisticsActivity** in the java folder. The purpose of the statistics page is to display the past trip details and also show the carbon emissions as well as calories burnt by the user. This is a really important page, as it helps users reflect upon the main motivation of the application. The users can see trip vise co2 emissions in KG and also can see the total co2 emitted for a particular day. The author believes this to be an eye opener for the major crowd to reflect upon their trip details and carbon emissions emitted by them. The users can also track the calories burnt with each trip and help them account for the approximate value of the total calories burnt in the day. Again, a nice feature through which the users are motivated for walking and cycling which are also very eco-friendly. In this method, an object of firebase was made as the application involved database access to display all the trips. A separate adapter was used to display the list of trips which will be discussed later. The **onCreate()** method sets up the initial layout of the page, the **inits()** method called inside the **onCreate()** method initialises variables used throughout the page as well as sets the **onClickListener** for back. The i**nits()** method also sets the filter layout through which the user can choose a date. The code for date picker can be seen under the **showDateFilter()** method in which just the **datepicker** and its dialog is managed.

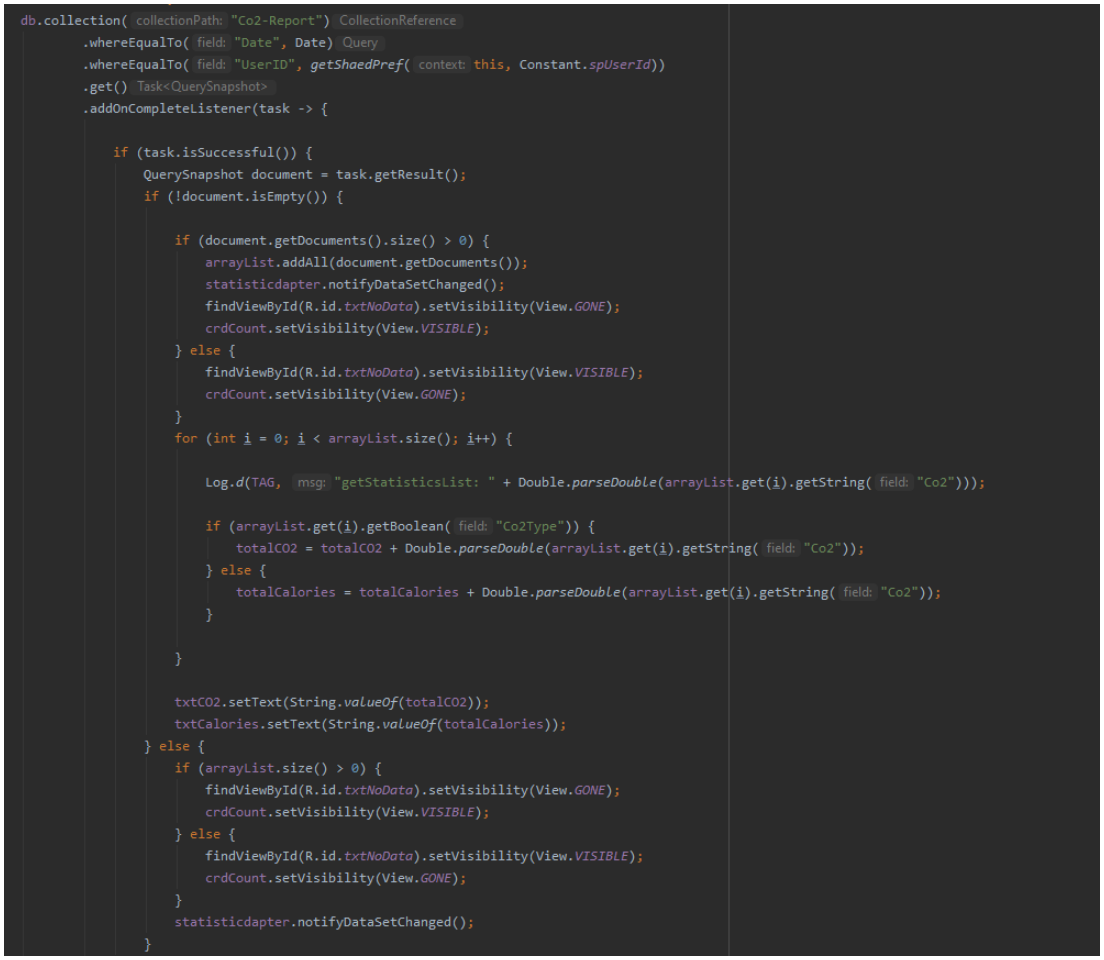


Filter Statistics

```
SimpleDateFormat dateFormatter = new SimpleDateFormat( pattern: "dd-MM-yyyy", Locale.getDefault());
TextView btn_submit = dialog.findViewById(R.id.btn_submit);
txtDate.setOnClickListener(v -> {
    final Calendar newCalendar = Calendar.getInstance();
    final DatePickerDialog StartTime = new DatePickerDialog( context: this, (view, year, monthOfYear, dayOfMonth) -> {
        Calendar newDate = Calendar.getInstance();
        newDate.set(year, monthOfYear, dayOfMonth);
        txtDate.setText(dateFormatter.format(newDate.getTime()));
    }, newCalendar.get(Calendar.YEAR), newCalendar.get(Calendar.MONTH), newCalendar.get(Calendar.DAY_OF_MONTH));
    StartTime.show();
});
btn_submit.setOnClickListener(v -> {
    getStatisticsList(txtDate.getText().toString());
    dialog.dismiss();
});

dialog.show();
}
```

Code Snippet for date filter

Initially on the statistics screen, the trip details of current date is shown, to display the data of a specific date, as the user chooses the date from calendar and clicks submit, the **getStatisticsList()** method is called and the chosen date is passed. Similarly in the statistic first screen the same method is called with the current date which displays the data.



```
db.collection( collectionPath: "Co2-Report") CollectionReference
        .whereEqualTo( field: "Date", Date) Query
        .whereEqualTo( field: "UserID", getShaedPref( context: this, Constant.spUserId))
        .get() Task<QuerySnapshot>
        .addOnCompleteListener(task -> {

            if (task.isSuccessful()) {
                QuerySnapshot document = task.getResult();
                if (!document.isEmpty()) {

                    if (document.getDocuments().size() > 0) {
                        arrayList.addAll(document.getDocuments());
                        statisticdapter.notifyDataSetChanged();
                        findViewById(R.id.txtNoData).setVisibility(View.GONE);
                        crdCount.setVisibility(View.VISIBLE);
                    } else {
                        findViewById(R.id.txtNoData).setVisibility(View.VISIBLE);
                        crdCount.setVisibility(View.GONE);
                    }
                    for (int i = 0; i < arrayList.size(); i++) {

                        Log.d(TAG, msg: "getStatisticsList: " + Double.parseDouble(arrayList.get(i).getString( field: "Co2")));

                        if (arrayList.get(i).getBoolean( field: "Co2Type")) {
                            totalCO2 = totalCO2 + Double.parseDouble(arrayList.get(i).getString( field: "Co2"));
                        } else {
                            totalCalories = totalCalories + Double.parseDouble(arrayList.get(i).getString( field: "Co2"));
                        }

                    }

                    txtCO2.setText(String.valueOf(totalCO2));
                    txtCalories.setText(String.valueOf(totalCalories));
                } else {
                    if (arrayList.size() > 0) {
                        findViewById(R.id.txtNoData).setVisibility(View.GONE);
                        crdCount.setVisibility(View.VISIBLE);
                    } else {
                        findViewById(R.id.txtNoData).setVisibility(View.VISIBLE);
                        crdCount.setVisibility(View.GONE);
                    }
                    statisticdapter.notifyDataSetChanged();
                }
```

Code Snippet of getStatisticsList()

getStatisticsList() is the most important method of this page which facilitates the use of databases which later display the contents using the **StatisticVehicle** adapter. In the **getStatiticList()** method, a query is being generated in which the user and the date is specified and contents are fetched from the database. Inside the same, **addOnCompleteListener** is set in which if the data fetched is stored in the document variable, the total co2 is calculated and the total calories is calculated using the **arrayList** which contains all the fetched data and the same are calculated. **Notifydatasetchanged** was used, as the data could be changed in the run time after initialisation, so as soon as the new data is received in the **arrayList** through the database, this method is called which informs the **adapter** about the **changed data** and the new updated data is displayed. Moving onto the **statisticsAdapter**, as mentioned before this is used to display the lists of data.The code for **statisticAdapter** can be found under the adapter folder. In the adapter, first of all there is a constructor in

which the arrayList and context are passed. **onCreateViewHolder()** method sets the xml layout that is used for printing. The xml layout used here is list_Item which can be found in the layout of the res folder. The text items that need to be displayed are found with the help of ViewHolder class implemented inside the adapter class. Followed by this, the contents are assigned in the **onBlindViewHolder()** method in which the object of **ViewHolder** class is passed and the position is passed. This method assigns the values to all the things in the dialog. The source, destination and time are common between all the modes of transport. Additionally, further checks were implemented for car in which the Fuel type and Vehicle type was also displayed and when it comes to walking and cycling, the co2 emitted is replaced by calories as these do not emit co2.

```
@Override
public void onBindViewHolder(ViewHolder holder, int position) {

    holder.txtsource.setText(arrayListData.get(position).getString( field: "Source"));
    holder.txtDestination.setText(arrayListData.get(position).getString( field: "Destination"));
    holder.txtDistance.setText(arrayListData.get(position).getString( field: "Distance"));
    holder.txtCO2.setText(arrayListData.get(position).getString( field: "Co2"));

    if (arrayListData.get(position).getString( field: "RideType").equals("CAR"))
        holder.txtVehicle.setText(arrayListData.get(position).getString( field: "RideType") + "(" + (arrayListData.get(position).getString( field: "Vehicle
                + "/" + arrayListData.get(position).getString( field: "FuleType")) + ")");
    else
        holder.txtVehicle.setText(arrayListData.get(position).getString( field: "RideType"));

    holder.txtDate.setText(arrayListData.get(position).getString( field: "Date"));
    if (arrayListData.get(position).getBoolean( field: "Co2Type"))
        holder.txtCo2Title.setText("Co2");
    else
        holder.txtCo2Title.setText("Calories");
}
```

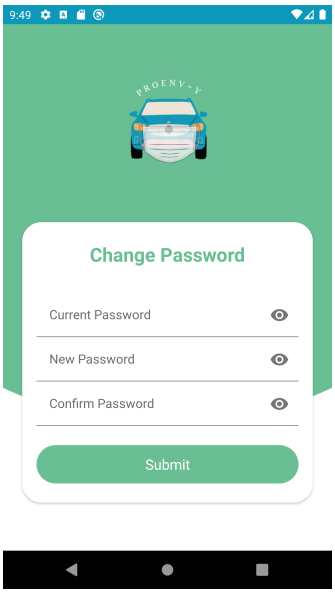
Code Snippet of binderView() method is adapter

6.4.8 Update Profile

Update Profile

The code for updating the profile can be seen under **UpdateProfileActivity**. The purpose of this functionality is to allow users to change their name and other fields like phone number whenever they wish to. The following would allow the user to change details if entered incorrectly during the registration process. The code is almost similar to the **RegisterActivity** and follows the same sequence of method calls. Author has made sure to disable the option for changing the email.

6.4.9 Change Password



Change Password

The code for the changing password can be seen under the **ChangePasswordActivity** in the java folder. The purpose of this functionality is to allow the users to change their password credentials. The fields to be filled by the users are current password followed by new and confirmed new password. The implementation of code is very similar to the login page. Initially the **onCreate()** method sets the layout, calls the i**nits()** method which initialises the fields and button on the app screen, call the **clicks()** method which sets the **onclick()** listener for the button and also starts the firebase. On clicking the submit method, initially the **valid()** method is called. The **valid()** method checks if the user has entered all the details as well as checks if the new password and the confirm new passwords are the same. On checking the above mentioned validations, the next thing the application should check if the Entered current password of the user matches. This would ensure that the passwords are only changed by the authenticated user. To ensure this, the application uses a **checkPassword()** method. Provided all the validations are met, the **click()** method calls the **checkPassword** method and passes the current password entered by the user. The purpose of **checkPassword()** method is to ensure that the password entered by the user matches with the database before changing it. The **checkPassword()** method firstly checks the same from the database and later calls the **registerDataToCloud()** method which makes the changes on the database and sets up the new password for the user. The author has made sure that the user is updated with messages like "**Password did not match**" or "**Enter password**"

throughout the page. The XML layout for the same can be seen under a**ctivity_change_password.xml** and is very similar to the one for login, maintaining the similar design throughout and giving a sense of uniformity to the users.

Chapter 7

Legal, Social, Ethical and Professional Issues

Throughout the project, the author has tried to follow any and all GDPR regulations set out by the European Commission and the British Courts. One functionality where there is ambiguity if GDPR was followed or not is the "forgot password" feature however what must be highlighted here is the fact that this feature has been listed in the limitations of the application and that this application does not hold any critical information thus ridding the possibility of privacy violation. The main purpose of the login functionality in the developed app was to serve as a registry system that would keep track of all Co2 emissions and calories burned.

As the developed application is supposed to be used on mobile devices, the usability test of the application was mandatory. As the usability test requires to get reviews about the app from a group set, the author employed students living in his accommodation to use the app and give their review. Whilst carrying out the review the author made sure that the confidentiality of all individuals is maintained thus the author made sure to not get any personal information.

Whilst developing the application the author referred to a number of websites and research papers which helped the author not only in terms of writing up the code of the application but it served to be a great help as it allowed the author to optimise his calculations and also design his app in a way that would be easy for people of all backgrounds to use. The bibliography of all these sources has been included in section 10.

Chapter 8

Results/Evaluation

8.1 Software Testing

8.1.1 Manual Testing

Positive Testing

In positive testing the application will be provided with a valid data set and will be checked if the application behaves in the way it is intended to do.
The above shown test is an example test performed by the user in which the functionality of map is tested with the moving pin-pointers. In total 17 positive tests were performed and the tables in green in Appendix A.3 are the positive tests.

Negative Testing

In Negative testing the application will be provided with improper data set and values to see how the application behaves with it. The purpose of negative testing is to see how the application behaves with the wrong data and prevents it from crashing.
The above shown example is a negative test performed by the author in which the author checks the change password functionality. In total 11 negative tests were performed and the tables in orange depict the negative tests.

8.1.2 Test on Emulators & Real Time Devices

The application was tested on various Android mobile devices to see how it functions with different dimensions and also how it works in landscape or portrait mode.

8.1.3 Usability Testing

Usability testing is an approach to perceive how simple it is to utilize something by a real user. The usability testing was performed by the author to test the interface of the application and measure its user experience. To perform the usability testing, the author had initially approached a few students staying in his accommodation. With their consent, the author first gave an insight of the application.

Followed by this, the students were given 10-20 mins to explore the app alone and on finishing it, the students were sent a survey form which had to be filled. It should be noted that the following were performed on a virtual device emulator as the author did not have an Android mobile device as well as no walk through was provided by the author.The students were suppose to explore the app on their own.The students were also told to be patient with the emulator as they are generally slow. The questionnaires and the surveys filled by the students can be found in Appendix A.3.

Later, on the responses, further SUS calculations were performed to find out the usability score of the application. Please note that the author wished to perform this test on a wider target audience of different age groups, but owing to the ongoing pandemic, the following could not be possible. The survey was based on the https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/ and then accordingly the scores were calculated by using the same website.

User	Calculations	SUS score
User1	[(4-1)+(5-4)+(4-1)+(5-2)+(4-1)+(5-2)+(4-1)+(5-2)+(4-1)+(5-2)]x2.5	70
User2	[(5-1)+(5-1)+(5-1)+(5-1)+(5-1)+(5-2)+(5-1)+(5-2)+(5-1)+(5-1)]x2.5	90
User3	[(4-1)+(5-1)+(5-1)+(5-1)+(4-1)+(5-2)+(5-1)+(5-2)+(4-1)+(5-1)]x2.5	87.5
User4	[(4-1)+(5-2)+(5-1)+(5-2)+(4-1)+(5-1)+(4-1)+(5-2)+(5-1)+(1-5)]x2.5	87.5

Table 8.1: SUS Calculation

Score	Grade
80 and above	A
70-80	B
60 -70	C
50 - 60	D
Upto 50	E

Overall the author was really satisfied with the results.

Average SUS: 83.75

Average Grade: A

8.2 Evaluation and Limitations

Serial Number	Requirement details	Evaluation	Limitations
R1	The application should be developed in either android or iOS platform.	The application was developed in Android OS. The reasons for the same can be read in Section 2.3	None
R2	The application could be developed in any suitable language of their choice.	To develop the application, the author used JAVA as the coding language.	None
R3	The application should work in mostly all the mobile devices of chosen platform.	The application was developed in Android SDK up to 29, minimum SDK being 21.	None
R4	The graphical user interface should be compatible with the chosen platform mobile devices.	The author did not hard code the height and width throughout the application and the same was successfully tested on different emulators of different sizes.	None
R5	The application must allow the user to recover their account in case the user forgets their password.	The author implemented a forgot password dialog on the Login page, in which when the user enters his email, the password shows up.	The author was not able to use the authentication offered by the Firebase and which made it difficult for the author to implement password recovery through email.
R6	The application must allow the user to register with the application.	The registration functionality was implemented in the application (Section 6.4.3) and test were performed to check the same.	None

R7	The application must allow the user to login with the application.	The login functionality was implemented in the application (Section 6.4.2) and test were performed to check the same.	The author was not able to implement the authentication by firebase and thus the author could not implement login through other platforms like google, Facebook etc..
R8	The application should allow the user to update their profile details.	The profile update option was implemented in the nav bar of the application. (Section 6.4.7)	The profile update does not allow the users to change their email address.
R9	The application must use Google Map API for the map-based interface.	The key was generated by the user through which the map interface was embedded. (Section 6.1.4)	None
R10	The application should seek the user's consent before using their location.	A dialog was implemented which is shown as soon as the user starts the application, seeking for the permission.	None
R11	The application must allow the user to define their journey (i.e. The start and the end location of the journey).	Separate pages were implemented which facilitated the user to pick their locations. (Section 6.4.5)	None
R12	The application could give suggested places recommendation while the user is typing in a location.	The author did implement the place API in the application.	None
R13	The application should allow users to choose location preference on map using the pin pointers.	A set location on map was implemented for both picking destination and pin pointers were made available.	None

R14	The application could draw the route between the start and end point of the trip.	The route was drawn between the source and destination and data parsing was performed for the same. (Read section 6.4.4.3)	It is essential to have a driving route between the 2 locations, so if there is no available route, the calculation of c02 cannot be implemented.
R15	The application could display an animation on the drawn route.	The animation was implemented in the application. (6.4.4.4)	None
R16	The application should store the trip journey of every user in the background allowing them to reflect upon the carbon emitted.	The author had implemented a separate page in which all the history could be seen with the total co2 and calories burnt count for the day.	None
R17	The application should calculate the estimated co2 emissions during the run time and similarly display it to the user.	The same was implemented by the user (read section 6.4.4.5 and 6.2)	The co2 calculation is based on co2 emitted kg/km for different vehicles. The co2 calculated for train and flights are on the driving route owing to google map restrictions
R18	The application should calculate the estimated calories burnt during the run time and similarly display it to the user for cycling and walking.	The same algorithm to calculate the co2 emissions can be used for an eco-friendly mode of transport like walking or cycling to count the calories (Approx. calories) of a person and could be similarly shown like the co2 emissions.	The app does not account for the user's height and weight, but rather uses an average value for the same.

R19	The application should promote the use of more eco-friendly mode of transport	The mode of transport buttons was given different colours. The eco-friendly ones were green and the rest of them were in increasing shade of red.	No colour was given to the car as the car had different fuel types and types. So, a standard colour could not be given. For example, an electric car emits less co2 compared to diesel.
R20	The application should be easy to use and search-able, thus inviting users from all age groups.	The author had performed detailed design phase in which the concepts of HCI were used. Read section 5	None
R21	The application should adhere to the Jakob Nielsen's 10 general principles for interaction design.	The following were kept in mind whilst developing and designing the actual application.	None
R22	A user guide to the application should be produced alongside with the project.	A detailed user guide was made by the author and the same can be found under Appendix B.	None

Chapter 9

Conclusion and Future Work

9.0.1 Conclusion

The given project required a thorough knowledge of the problem for which the application has been developed. Pollution, especially air pollution has always been an acute hazard to the health and well being of mankind. However, due to globalisation and economic growth of the society, no individual has given heed to pollution caused by motor vehicles they own and how it will impact future generations and their health. Working on this project further acquainted the author about the crisis as well as inspired him to go in further depth of the knowledge of the issue. Thus helping the author to aspire to take preventive measures on his own.

As this was the first time the author has created a mobile application, the project helped him gather and put in the knowledge gained over the past few years into action. This process of development presented a good opportunity for the author to realise and understand how various software architectures and models are applied and implemented on an industrial and professional level. The author was made aware of numerous new fundamental things like Activity Lifecycle, use of adapters and etc.. Less attention was given to the authentication of the application, as the app currently does not store any critical information that may be personal to the user. The major focus of the author, throughout the application, went into developing the main motive of the application i.e. calculation of carbon emissions.

The author, to the best of his capabilities, has fulfilled all functional and non-functional requirements that were decided between him and his supervisor. However it must still be noted that due to some underlying mitigating circumstances, for which the author received an extension, the author had limited ability to complete the project according to his initial wishes,

thus all limitations have been mentioned in the Future works . Some of these work are easy to implement due to the author's modular approach to code design, which was done inline with the author's belief in the future scalability of the application.
Overall, the author feels that through this project, he has developed some special interest towards app development and wishes to develop more applications in the future.

9.0.2 Future Work

The author feels that the app needs few things before the app is deployed on the Google Play Store. Most essentially the application needs to have a complete and adequate authentication through which the user data is stored safely and not cannot be accessed by any one. A well secure database, would also allow the developer to put in their critical information such as credit card details and the application could be connected to carbon offsetting schemes, so that the user could account for their burnt carbon emission immediately. Another important aspect, the application could work on the estimation of co2 emitted and calories burnt. The application could account for various like time taken, weight and height of the person to calculate calories. The data used for the calculation of co2 and calories are currently static and give an estimated results. A proper database with the co2 emissions of every car, for the calculation of emissions could be implemented to improve the results.
The mode of transport like flights and trains do not account for the actual distance between them (Great circle distance for flights) and currently is implemented on the driving map style owing to google maps restriction. This could be updated in future. Another interesting thing that could be implemented in the application is the live location tracking, instead of the animation. With the use of the Iterative Waterfall Model which was followed by the user, these functionalities could be easily added in the future, on the same application.

Chapter 10

Bibliography

The references are mentioned on the next page.

1. energy post .eu, 2016, why it's so difficult to reduce co2 emissions? (online). Available at: <https://energypost.eu/difficult-reduce-co2-emissions/ > [Accessed Nov,2020]
2. Sciencing, 2017., How humans disrupt the ecosystem. (online). Available at: <https://sciencing.com/humans-disrupt-ecosystem-5968.html> [Accessed Nov,2020]
3. Sciencing, 2018, Positive effects on the environment from going green (online). Available at: <https://sciencing.com/positive-effects-on-the-environment-from-going-green-5117214.html > [Accessed Nov,2020]
4. our worldindata, 2020. Cars, planes and trains: where do CO2 emissions from transport come from? (online). Available at<https://ourworldindata.org/co2-emissions-from-transport> [Accessed Nov,2020]
5. Epa.gov, Carbon pollution from transportation (online). Available at:< https://www.epa.gov/transportation-air-pollution-and-climate-change/carbon-pollution-transportation#:~:text=%E2%80%8BGreenhouse%20gas%20(GHG)%20emissions,terms%20than%20any%20other%20sector.> [Accessed Dec,2020]
6. The conversation, 2019., Why reducing carbon emissions from cars and trucks will be so hard (online). Available at: <https://theconversation.com/why-reducing-carbon-emissions-from-cars-and-trucks-will-be-so-hard-113230> [Accessed Dec,2020]
7. Statista, Carbon emission., (online). Available at: <https://www.statista.com/search/?q=carbon+emission+&qKat=newSearchFilter&sortMethod=idrelevance&isRegionPref=826&sortMethodMobile=idrelevance&statistics-group=1&statistics=1&forecasts=1&infos=1&topics=1&studies-reports=1&dossiers=1&groupA=1&xmo=1&surveys=1&toplists=1&groupB=1&branchreports=1&countryreports=1&groupC=1&expert-tools=1&cmo=1&mmo=1&co=1&tmo=1&amo=1&io=1&dmo=1&accuracy=and&isoregion=0&isocountrySearch=&category=0&interval=0&archive=1> [Accessed Dec,2020]
8. Bbc.co.uk, 2019., The five major challenges facing electric vehicles (online). Available at: <https://www.bbc.co.uk/news/uk-49578790> [Accessed Dec,2020]
9. Altexsoft, 2018., The good and the bad of Android app development (online). Available at:<https://www.altexsoft.com/blog/engineering/pros-and-cons-of-android-app-development/> [Accessed, Dec 2020]
10. Brainvire, Why android developers should prefer android studio over eclipse (online). Available at:<https://www.brainvire.com/android-developers-prefer-android-studio-eclipse/> [Accessed Dec,2020]
11. Nationalgeographic, 2015., Five reasons we need to act now on climate change (online). Available at: <https://www.nationalgeographic.com/news/2015/12/151204-climate-paris-disease-antarctic-arctic-ice-melt-acidification-fish-co2/> [Accessed Dec,2020]

12. Geektonight, 2020., Iterative waterfall model, software engineering (online). Available at: <https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.geektonight.com%2Fiterative-waterfall-model-software-engineering%2F&psig=AOvVaw2hBKgCJnAsDsH_1qF1G6An&ust=1608346827641000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCIDeuPzE1u0CFQAAAAAdAAAAABAX> [Accessed Dec,2020]
13. Theappsolutions, Android apps vs IOS apps- what and why is better in 2021? (online). Available at:<https://theappsolutions.com/blog/development/ios-vs-android/#:~:text=Advantages%20of%20developing%20Android,developing%20an%20intuitive%20user%20interface.> [Accessed Dec,2020]
14. Gs.statcounter, 20-21.Mobile operating system market share worldwide, (online). Available at: < https://gs.statcounter.com/os-market-share/mobile/worldwide> [Accessed Feb,2021]
15. Educative.io, What is firebase? Available at: < https://www.educative.io/edpresso/what-is-firebase> [Accessed Jan,2021]
16. Material.io, Understanding navigation (online). Available at: <https://material.io/design/navigation/understanding-navigation.html#lateral-navigation> [Accessed April,2021]
17. Creately, 2021., Ultimate entity relationship diagram tutorial (ER diagrams) (online). Available at: <https://creately.com/blog/diagrams/er-diagrams-tutorial/#:~:text=Entity%20relationship%20diagrams%20are%20used,their%20relationships%20with%20each%20other.&text=In%20the%20diagram%2C%20the%20information,attributes%20of%20a%20particular%20entity.> [Accessed April,2021]
18. Visual Paradigm, What is Activity Diagram? (online). Available at: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/> [Accessed April,2021]

19. Visual Paradigm, What is a Class Diagram? (online). Available at: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/> [Accessed April,2021]

20. Jinfonet, 3-Tier Architecture: A Complete Overview.(online). Available at: <https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/>[Accessed April,2021]

21. Guru99,MVC Tutorial for Beginners: What is, Architecture & Example(online).Available at: <https://www.guru99.com/mvc-tutorial.html> [Accessed April,2021]

22. ClearBridge,4 Benefits of Mobile App Prototyping.(online).Available at:<https://clearbridgemobile.com/4-benefits-of-mobile-app-prototyping/#:~:text=The

%20purpose%20of%20a%20prototype,a%20working%20design%20and%20layout.> [Accessed April,2021]

23. Interaction-Design-Foundation,Persona- A Simple Guide (Online). Available at:<https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them#:~:text=Personas%20are%20fictional%20characters%2C%20which,%2C%20experiences%2C%20behaviours%20and%20goals.> [Accessed April,2021]

24. CleverTap,What is an SDK(online). Available at:<https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them#:~:text=Personas%20are%20fictional%20characters%2C%20which,%2C%20experiences%2C%20behaviours%20and%20goals.>[Accessed April,2021]

25. NN/g, 10 Usability Heuristics for User Interface Design. Available at:<https://www.nngroup.com/articles/ten-usability-heuristics/> [Accessed March,2021]

26. Myclimate, Car CO2 emissions.(Online)Available at:<https://co2.myclimate.org/en/car_calculators/new>[Accessed April,2021]
27. UsabilityGeeks, How to use the system usability scale to evaluate your website. Available at:<https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/>[Accessed April,2021]
28. Github, uber-car animation android (online), Available at: <https://github.com/MindorksOpenSource/Uber-Car-Animation-Android (Car animation)>[Accessed April,2021]
29. Wingsquare, 2019. Android: Driving route from my location to destination in google maps (online). Available at: <https://www.wingsquare.com/blog/android-driving-route-from-my-location-to-destination-in-google-maps/> [Accessed Jan,2021]
30. Developer.android, Update UI components with navigation UI (online). Available at: <https://developer.android.com/guide/navigation/navigation-ui?gclid=CjwKCAjw7J6EBhBDEiwA5UUM2k3Xi2addClfoVwSi9KU8YtKFMuC7l7aLTf2B9ksn03pAwUzrTgHWxoC0CkQAvD_BwE&gclsrc=aw.ds#java> [Accessed Jan,2021]
31. Stackoverflow, Navigation drawer to switch activities instead of fragments (online). Available at: <https://stackoverflow.com/questions/19442378/navigation-drawer-to-switch-activities-instead-of-fragments> [Accessed Jan,2021]
32. Geeksforgeeks, 2021. How to create and add data to firebase firestore in android (online). Available at: <https://www.geeksforgeeks.org/create-and-add-data-to-firebase-firestore-in-android/> [Accessed Dec,2021]

33. Code.tutsplus,2021. How to code a navigation drawer for an android app (online). Available at:<https://code.tutsplus.com/tutorials/how-to-code-a-navigation-drawer-in-an-android-app--cms-30263> [Accessed Jan,2021]
34. javatpoint, Update and delete in javastore (online). Available at: <https://www.javatpoint.com/firebase-update-and-delete-in-firestore> [Accessed Jan,2021]
35. Youtube, 2019. Firebase tutorial- create, read, update, delete data (online). Available at: <https://www.youtube.com/watch?v=qA9L3_cK9Z0> [Accessed Jan,2021]
36. Developers.google, Quickstart adding a map (online). Available at: <https://developers.google.com/maps/documentation/android-sdk/start> [Accessed Jan,2021]
37. Developers.google, Get started (online). Available at: <https://developers.google.com/maps/documentation/places/android-sdk/start> [Accessed Jan,2021]