



Bilkent University
Department of Computer Engineering

Senior Design Project

CarBuds

FINAL REPORT

Ahmet Emre Nas	21402357
Doğukan Altay	21400627
Ali Osman Çetin	21302483
Aras Heper	21302248

Supervisor: Uğur Doğrusöz

Jury Members: Ercüment Çicek, H. Altay Güvenir

Innovation Expert: Doğukan Şengül

1.Introduction	4
2.Architecture	5
2.1. Overview	5
2.2. Subsystem Decomposition	5
2.3. Hardware-Software Mapping	6
2.4 Design Decisions	7
1.1. User Interface Design Decisions.	7
1.2. Algorithm Design Decisions	8
2.5. Client / Frontend Architecture	8
2.5.1. Architectural Pattern	8
2.5.2. Distribution & Deployment	8
2.6. Service / API Architecture	9
2.7. Database Architecture	9
2.8. Transaction Architecture	9
3.Contextual Impacts	9
3.1. Short Term Impacts	9
3.2. Long Term Vision	9
4.Contemporary Issues	10
4.1.Social Issues	10
5. Contemporary Tools Used	10
Git	10
Docker	10
Nginx	10
RabbitMQ	11
Google Cloud	11
Firebase	11
Google Maps API	11
PostgreSQL	11
6.Used Libraries, Solutions and Resources	12
7.User Manual	14
7.1 Login Page	14
7.2 Signup Page	16
7.3 Role Selection Page	17
7.4 Initial Profile Setup Page	18
7.5 Matchmaking Page	19
7.6 Profile Page	20
7.6 Matches Page	21
7.7 Edit Profile Page	22
7.8 Message Page	22

7.9 Match Info Page	23
8.Document References	24

1.Introduction

Hitchhiking has always been a common sight, “CarBuds” is a solution provided on mobile machines running by Android to transform this phenomenon into a more convenient form, while allowing its users to form new relationships.

“Carbuds” allows its users to register as, either or both, driver and passenger. The solution provides tools to help users visually recognize trips that will be done by other users satisfying the users criteria.

Users can also further investigate the available choices to make their trip with, can communicate with the candidates via the messaging service.

This document investigates the solution “Carbuds” by roughly laying the architectural details, suggests the socioeconomic impacts if the project succeeds to reach a critical popularity mass, and discusses possible contemporary issues related to project.

The reader can also find a User Manual at the section “7”, and all used libraries together with their licensing at the section ”6”.

2. Architecture

2.1. Overview

This section briefly presents the overall architectural and design decision taken for the implementation of the solution “CarBuds”. More elaborately, it tries to present the subsystem decomposition via a composition diagram(2.2) followed by hardware-software mapping(2.3). Design decisions are being discussed at the following sub-section(2.4), which is also followed by specific architectural decisions related to the subsystems(2.5-2.8).

2.2. Subsystem Decomposition

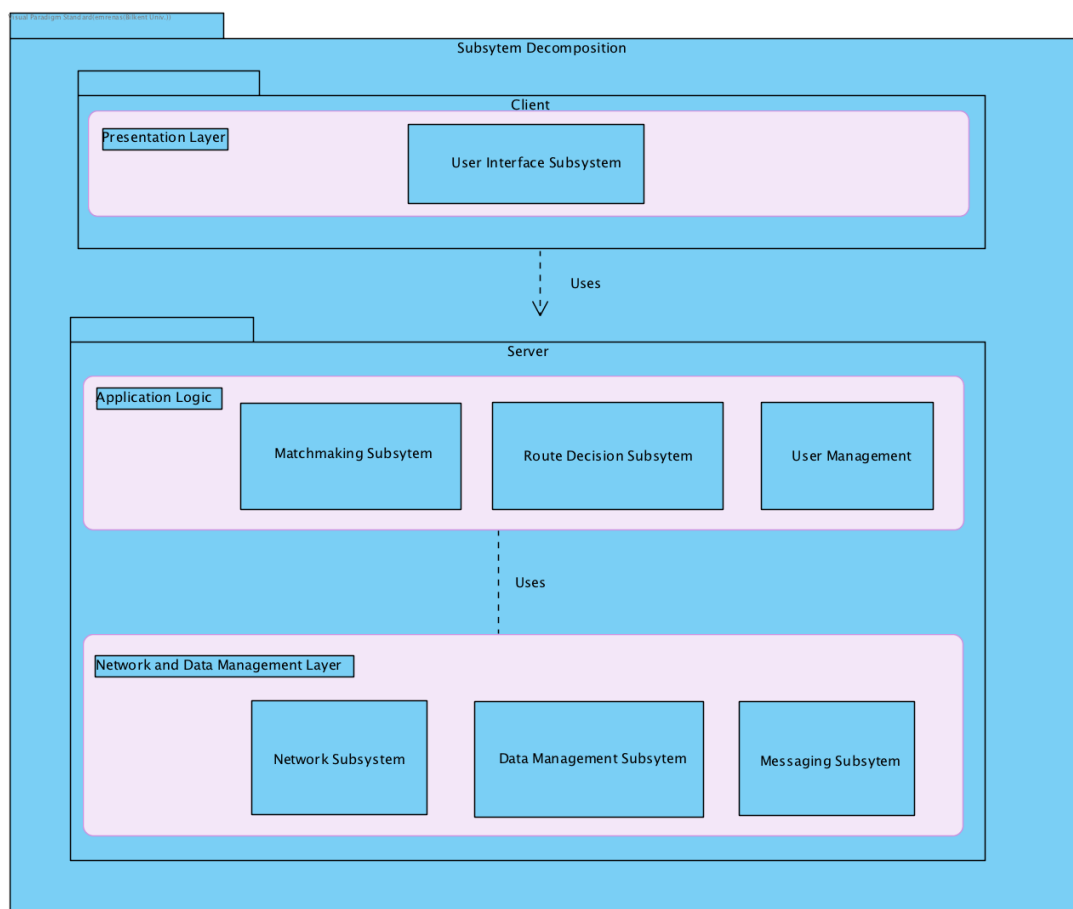


Image: System composition represented by the members decomposed to sub-parts.

2.3. Hardware-Software Mapping

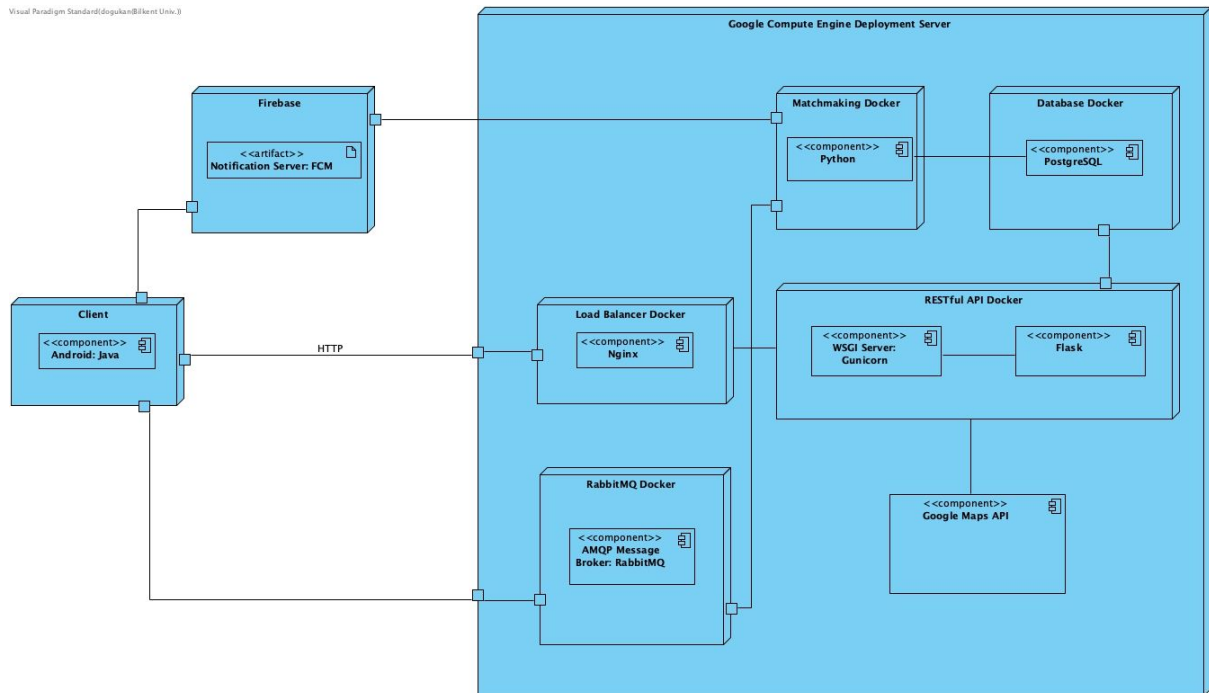


Image: Hardware-Software mapping represented by the composition diagram.

Carbuds application system consists of one client and two servers. All the communications between deployment server and client handled with HTTP protocol using RESTful API. For the in-app messaging service each client connects to the AMQP message broker server in order to communicate with each other using AMQP protocol. Carbuds backend consists of several docker containers for each different component needed. The reason behind is that with the docker containers each component can be scaled up easily on server clusters in case of a scalability problem and gives us the opportunity to maintain the version compatibility between each component in the deployment server. Android client uses JSON for sending and receiving data from/to the RESTful API server. In the backend server, nginx used as a load balancer and reverse proxy each http request to the WSGI server in the RESTful API container. Therefore, in case of huge traffic load balancer will ensure that the server will be reachable. Since client send all the data to the API, it has no direct connection to the database which increases the security of the system and the maintainability of the overall system. All database queries handled with the API. For route calculations, system uses Google Maps API.

API container also handles the preparation of the data for the matchmaking container. When a user creates a trip, API generates the route polyline for the trip and writes to the database and search for any possible match in the database. At the and, if API finds a possible match saves the possible

match to the database. In order to find a match there are several constraints, each user's trip start points should be in a circle with 1 Km radius, their routes' intersection should be more than 1 Km, their gender and music preferences should comply and their desired trip start times' difference should not exceed 1 hour in favor of the driver.

Client demands possible candidates from the API. After the user's decision, appropriate data will be sent to the API again. Matchmaking container works as a STIL. Constantly checks possible matches and if finds that both users liked each other it marks them as match and writes to the database server, creates a chatroom in RabbitMQ message broker and sends a notification to the each user using Firebase Cloud Messaging API through Firebase server. Android client opens the listeners for the match's chatroom and enables in-app messaging for the users. Also provides appropriate information about the trip.

2.4 Design Decisions

During design phase developers needed to agree on both user interface and algorithmic designs. This section contains information about design decisions and the impact of these decisions on the project development.

1.1. User Interface Design Decisions.

First discussion about user interface that we needed to solve was about how user should match with each other. Some of the team members suggested every possible candidate should be visible and can talk with each other's. But this would create a problem where some users can annoy other users. Therefore, we decided to develop our application in a way that both of the users who is going to talk needs to accept each other. Because of this we did not list every possible candidates to each user but designed a cardview which caused a matchmaking system. When user want to find possible driver or hitchhiker a picture of each users shown to user and if both of the user accept each other they can start talking with each other.

After this we discussed how can we accomplish our user-friendly interface goal. We know if there are many buttons on a page it decreases user-friendliness. Because of this we decided to design tabbed view on main page but in order to keep it simple we only added three tab to main page of the application. One of the tab for profile of users and settings, one tab is for the setting a new trip and matchmaking and the last one is for chatting with matched users.

Another discussion we encountered what should we display to users when they matched with each other. We decided to show them the preferences of other user. And the common route they will share when they decide to travel together. By this way hitchhiker can look the intersection of routes and decide to travel with his/her most suitable match.

1.2. Algorithm Design Decisions

The main and the most important algorithm for CarBuds is matchmaking. We can find the users' trip's routes however drawing these routes on Google Maps does not help us to find users who has a same or similar trip. In order to achieve this, we took polyline of routes from google maps. Polyline is compressed data of the all the points in a route. A detailed explanation of this algorithm can be found at:

<https://developers.google.com/maps/documentation/utilities/polylinealgorithm>[21].

By reversing this algorithm we could take every point on someone's route. After this we checked every users' routes who started a trip close to each other with each other and created possible matches.

2.5. Client / Frontend Architecture

2.5.1. Architectural Pattern

"Carbuds" app follows the MVC architectural pattern, as the application logic is tightly bound to I/O and user interaction, such a model is enforced, unless MVP or MVVM is used, which is not. Various I/O listeners listen to their respective sources, when triggered triggers the change in the data models, as such, changes either await the views or notify the watcher views to be consumed by the views associated with the models.

The implementation of the app is in Java using Android SDK.

2.5.2. Distribution & Deployment

To compile "Carbuds" successfully, the client machine must have android API version from 2.4 to 2.8.

The software will be distributed using Google's Play Store .

Internet connection is required to deploy the app, and to run the features successfully.

2.6. Service / API Architecture

The web server adheres to REST architecture, and as such the relevant states are being kept in either the data management tools, or on the client.

2.7. Database Architecture

Databases are implemented in PostgreSQL.

2.8. Transaction Architecture

- The messaging system between client machines and the server is provided by RabbitMQ provider through AMQP message queuing protocol to sustain the chat service. This connection is initiated by HTTP handshaking provided by Firebase.
- The client machine connects to web server hosting the service layer via HTTP calls.
- The web server hosting the service layer fetches data from or pushes data to database provider via PostgreSQL queries.

3.Contextual Impacts

3.1. Short Term Impacts

“CarBuds” allows easier, pre-planned hitchhiking trips; “CarBuds” helps hitchhiking evolve from a contingent way of transportation to a convenient way of transportation, possibly reducing the resources needed for the same amount of work, economically cutting down the private expenditure, especially from the lower income people who can’t afford cars, as such, slightly reducing the skewed asset distribution in the economy, also reducing the need for growth in the physical infrastructure.

Environmentally “CarBuds” helps to support more population with the same resources.

3.2. Long Term Vision

This project helps to empower the common idea, that through sharing, instead of strict ownership of the resource, the society becomes more productive and more socially secure. This project is another small step to establish such a value.

4. Contemporary Issues

4.1. Social Issues

It is very true that both classical hitchhiking and a technologically supported one such as “CarBuds” can be abused criminally by mal-intentioned individuals. However this commonality does not extend far; by supporting hitchhiking digitally, and by luckily creating an environment where software assisted hitchhiking is a norm; dependability and regulatability of each session of hitchhiking increases, uncontrolled or unaccounted hitchhiking ratio decreases. This solution, “CarBuds”, empowers both hitchhikers and drivers.

There is an uncalculated risk invoked in the project, which is the initial assumption that drivers desire to have passengers for no personal benefit other than humanitarian gratification and companionship, doing so without having those passengers on their immediate peripherals.

However, as every other innovative project should be, “CarBuds” is an unregulated experiment on the society, a step worth taking, especially since the only risk takers are the developers themselves.

5. Contemporary Tools Used

- Git

“Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency” [<https://git-scm.com>]. We used GitHub for merging everyone’s code during development.

- Docker

“Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package” [22]. We used it for deploying our Flask and Database server.

- Nginx

“NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more. It started out as a web server designed for maximum performance and stability. In addition to its HTTP server capabilities, NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and a reverse proxy and load balancer for HTTP, TCP, and UDP servers”[17]. Since we are developing a web server as a backend of carbuds app it is important to balance the network on the server. For this purpose we used Nginx.

- RabbitMQ

“RabbitMQ is a messaging broker - an intermediary for messaging. It gives your applications a common platform to send and receive messages, and your messages a safe place to live until received”[18] . RabbitMQ is a message-broker on top of amqp. We used rabbitmq for messaging between users.

- Google Cloud

The backend of the carbuds deployed in the compute engine which cloud computing service of Google Cloud. By using compute engine we ensured backend of the carbuds will always run and can be accessed from anywhere.

- Firebase

Firebase is a product of google and developed for developing mobile and web application without server-side of programming which includes real-time database, data storage and authentication service. However, we used it only for push notifications.

- Google Maps API

Carbuds rely on accurate location information of users in order to match users with the best possible candidates. In order to achieve this we used Google Maps for route information of users' trips.

- PostgreSQL

For database, we used PostgreSQL because of it's functionality on geographical data which helped us to match users who are close to each other.

6.Used Libraries, Solutions and Resources

- “Fast Android Networking” [1]

License: Apache 2.0, copyrighted on amitshekhariitbhu [2]

Explanation: Gives tools to quickly post network requests and receive their responses.

- “ImagePicker” [3]

License: MIT[4]

Explanation: Allows selection of images from the users file system.

- “Apache Commons IO” [5]

License: Apache 2.0, copyrighted on apache [6]

Explanation: Supports tools to interpret data streams.

- “Glide”, [7]

License: Apache 2.0, copyrighted on bumptech [8]

Explanation: A library for image fetching and displaying from the network.

- “RabbitMQ”, [9]

License: MPL 1. 1, copyrighted on Pivotal Software Inc.[10]

Explanation: A messaging solution used in the software to implement the chat service.

- “CardStackView” [11]

License: Apache 2.0, copyrighted on yuyakaido [12]

Explanation: An elaborate GUI library to display a list of elements.

- “RoundedImageView”[13]

License: Apache 2.0, copyrighted on Vincent Mi [14]

Explanation: Grants a GUI image viewing component that has soft edges.

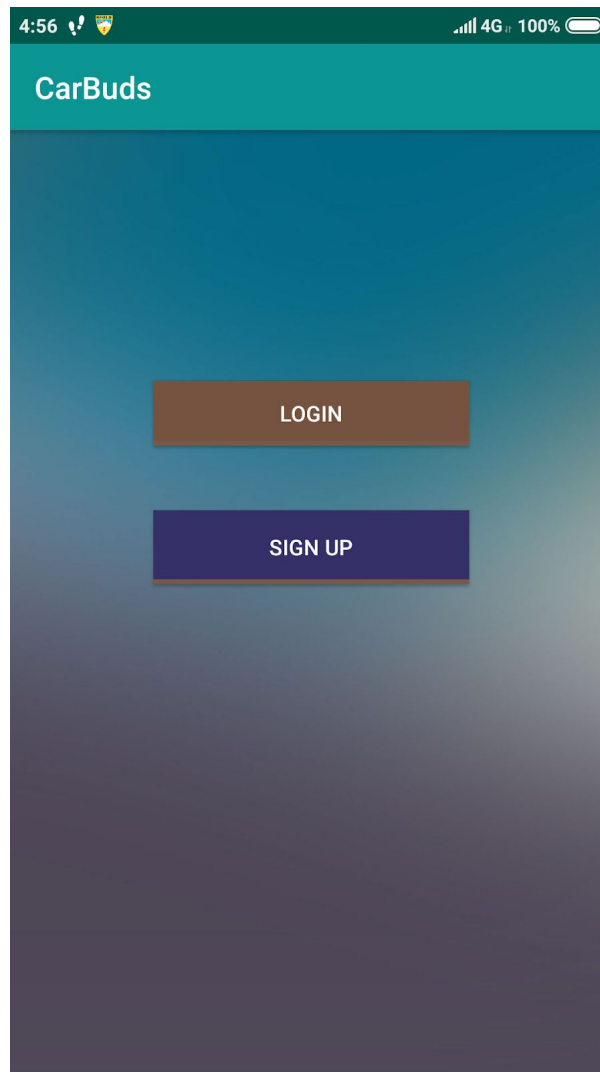
- “Nachos” [15]

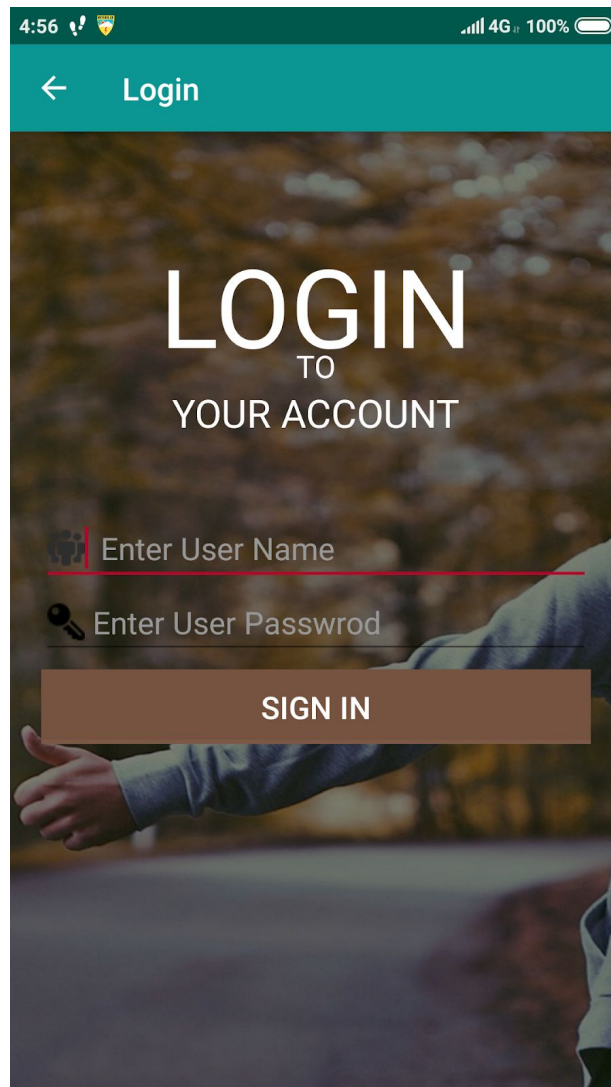
License: Apache 2.0, copyrighted on Hootsuite Media [16]

Explanation: A library to display chip-like text fields.

7. User Manual

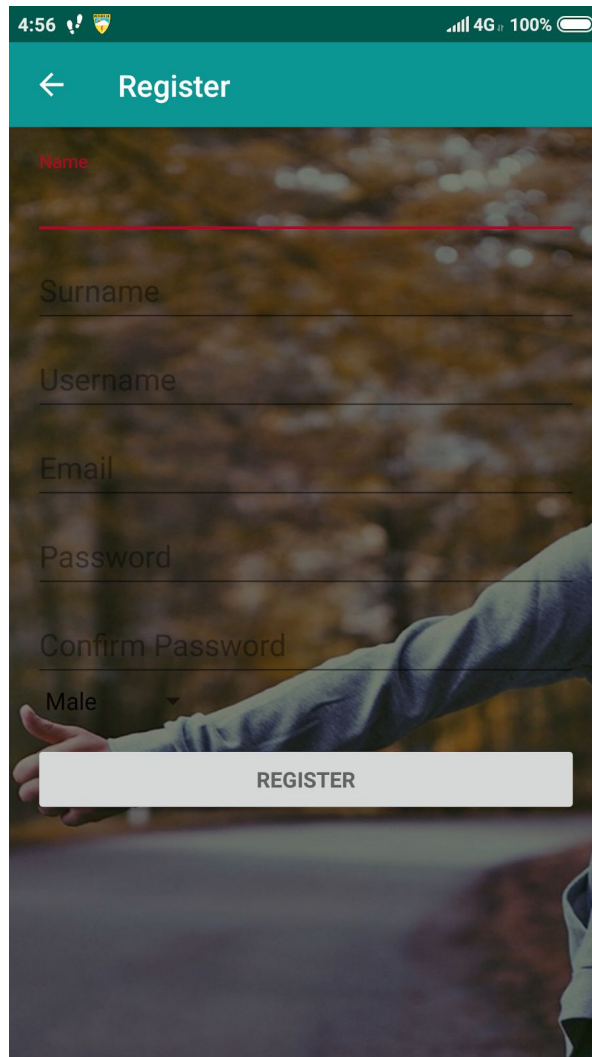
7.1 Login Page





First page of the Carbuds is “Login Page”. When a fresh user opens the app, they will be greeted with the login page. Also active users will be greeted with login page if they lost their sessions or logged out. Fresh users can use “Sign up” button to create an account. In order to gain access to the system, user need to provide its correct credentials. If credentials validated by the server their login will be successful, else they won’t get access.

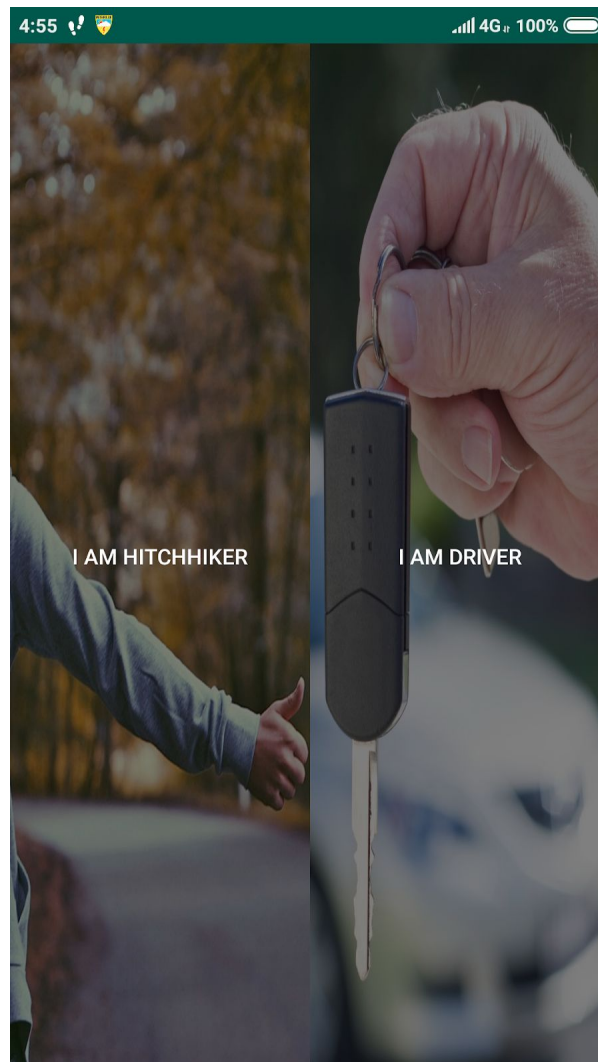
7.2 Signup Page



The screenshot shows a mobile application interface for a registration page. At the top, there is a teal header bar with a back arrow icon and the text "Register". Below the header, the background is a blurred image of a person's arm. The form consists of several input fields: "Name" (with a red label), "Surname", "Username", "Email", "Password", and "Confirm Password". Below these fields is a dropdown menu currently showing "Male". At the bottom of the form is a white button with the text "REGISTER". The status bar at the very top shows the time as 4:56, signal strength, 4G network, and 100% battery.

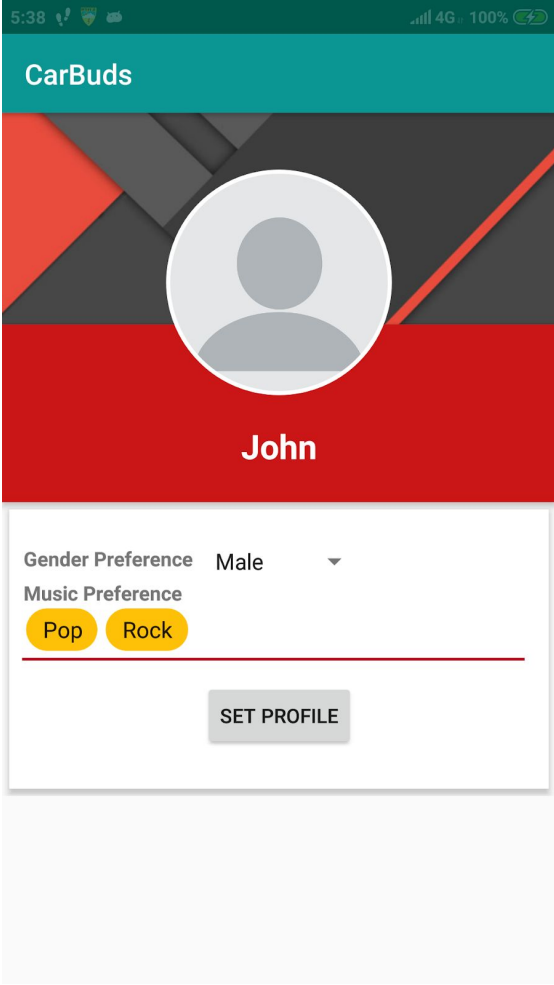
Fresh users can create an account by using Signup page. Each user need to provide several credentials about them. After signup, users will be redirected to the login page in order to access the system.

7.3 Role Selection Page



In Carbuds, each user should select a role whether as hitchhiker or driver. If users did not choose any role before, they will be greeted with the Role Selection page in order to choose a role.

7.4 Initial Profile Setup Page



The screenshot shows a mobile app interface for 'CarBuds'. At the top, a teal header bar contains the app name. Below it is a large red banner with a grey circular profile icon and the name 'John' in white. Under the banner, there is a white form area. The form contains a 'Gender Preference' dropdown menu set to 'Male', a 'Music Preference' section with two yellow buttons labeled 'Pop' and 'Rock', and a grey 'SET PROFILE' button at the bottom. The status bar at the very top shows the time as 5:38, 4G signal, and 100% battery.

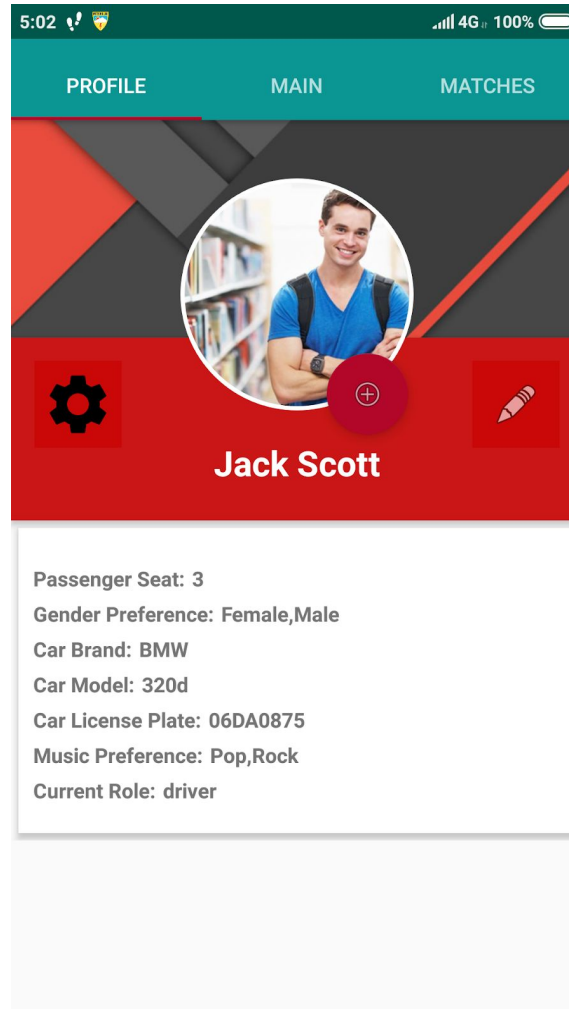
In Carbuds, in order to set trips and get matches each user should setup their preferred role profile. However, if a user does not have any prior profile for their preferred role, they will be greeted with the Initial Profile Setup page which changes depending on their role.

7.5 Matchmaking Page



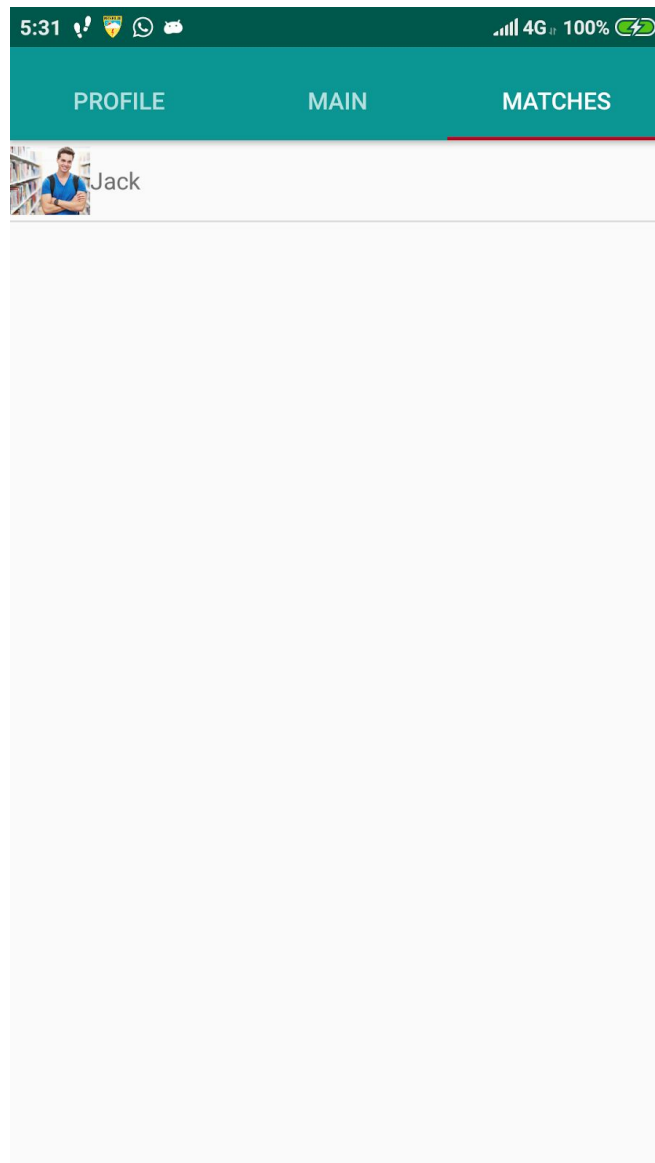
Matchmaking is one of the most important feature of the Carbuds. In Matchmaking page, users can like/dislike possible candidates depending on their current trips. Users can see the candidates' profile pictures, name, their trip start time and possible intersection of both parties trips over Google Maps. Users either swipe right to like or click like button and either swipe left or click dislike button in order to dislike the candidate.

7.6 Profile Page



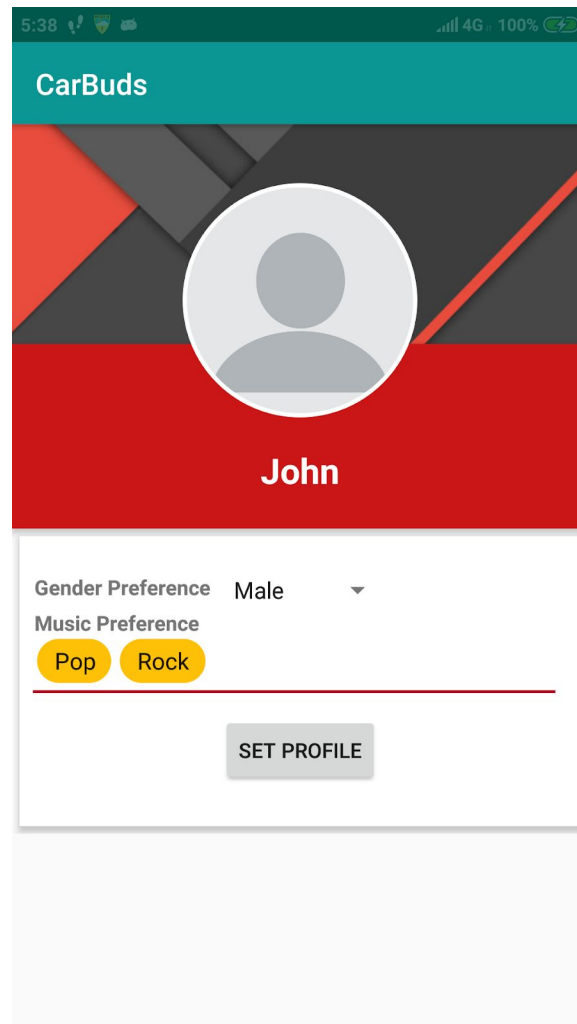
User can see their active role's profile information in Profile page. Also in this page, user can edit his/her profile information by clicking edit button. Users can change/add a profile picture by clicking add image button. In Profile page, CarBuds provide settings menu as a navigation bar. It can be activated by clicking gear button. In settings user can, logout, change role, cancel current trip and set a new trip.

7.6 Matches Page



Users can see their active matches in this page. In Matches page, each match listed as message list. Users can access the corresponding match's message page by clicking the elements in the list.

7.7 Edit Profile Page

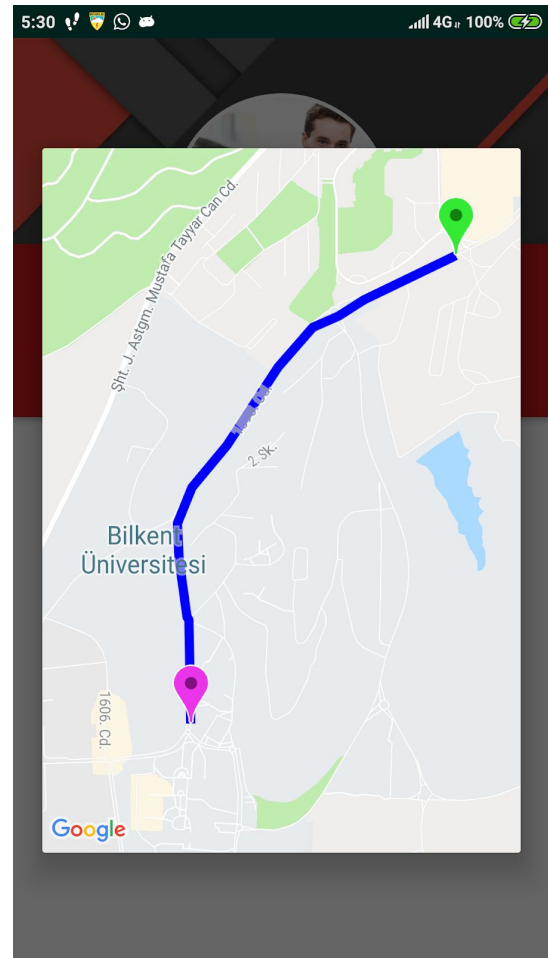
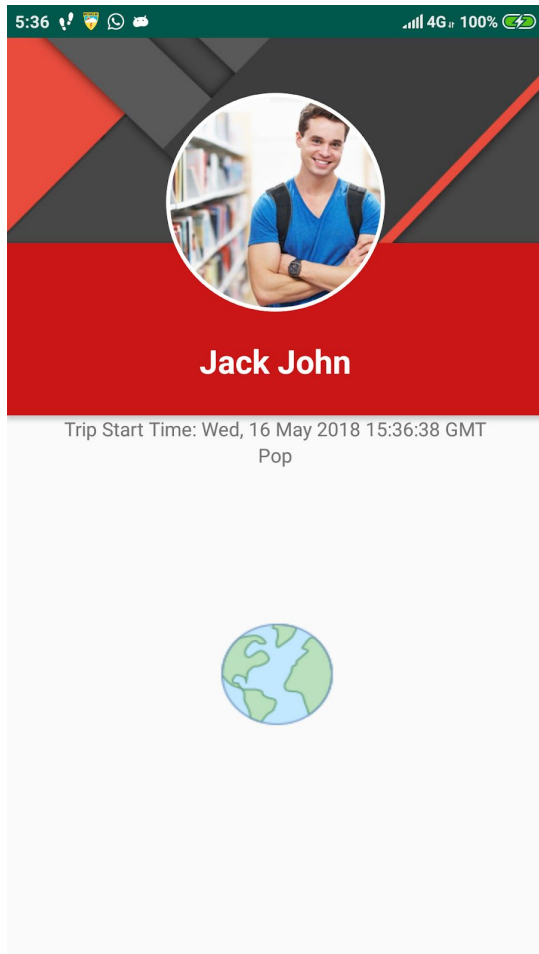


Users can edit their current profile information in this page. Edit Profile page changes depending on their active role.

7.8 Message Page

Carbuds provides users a in-app messaging service. Each match represented as messages for each user.

7.9 Match Info Page



Users can access the match informations in Match Info page. Users can see the trip route of the match and start time with the other user's profile picture.

8.Document References

- [1] Github, “Fast Android Networking”, December 2018 [Online]. Available:
<https://github.com/amitshekhariitbhu/Fast-Android-Networking>
- [2] Github, “Fast Android Networking”, December 2018 [Online]. Available:
<https://github.com/amitshekhariitbhu/Fast-Android-Networking/blob/master/LICENSE>
- [3] Github, “Android image picker”, December 2018 [Online]. Available:
<https://github.com/esafirm/android-image-picker>
- [4] Github, “ImagePicker”, December 2018 [Online]. Available:
<https://github.com/hyperoslo/ImagePicker/blob/master/LICENSE.md>
- [5] Github, “Apache”, December 2018 [Online]. Available:
<https://github.com/apache/commons-io>
- [6] Apache, “License”, December 2018 [Online]. Available:
<https://www.apache.org/licenses/LICENSE-2.0>
- [7] Github, “Glide”, December 2018 [Online]. Available:
<https://github.com/bumptech/glide>
- [8] Github, “Glide License”, December 2018 [Online]. Available:
<https://github.com/bumptech/glide/blob/master/LICENSE>
- [9] Github, “Rabbitmq”, December 2018 [Online]. Available:
<https://github.com/rabbitmq/rabbitmq-server>
- [10] Github, “Rabbit mq License”, December 2018 [Online]. Available:
<https://github.com/rabbitmq/rabbitmq-server/blob/master/LICENSE-MPL-RabbitMQ>
- [11] Github, “Card Stack View”, December 2018 [Online]. Available:
<https://github.com/yuyakaido/CardStackView>
- [12] Github, “Card Stack View License”, December 2018 [Online]. Available:
<https://github.com/yuyakaido/CardStackView/blob/master/LICENSE>
- [13] Github, “Round Image View”, December 2018 [Online]. Available:
<https://github.com/vinc3m1/RoundedImageView>
- [14] Github, “Roudn Image View license”, December 2018 [Online]. Available:
<https://github.com/vinc3m1/RoundedImageView/blob/master/LICENSE>
- [15] Github, “Nachos”, December 2018 [Online]. Available:
<https://github.com/hootsuite/nachos>
- [16] Github, “Nachos Lisence”, December 2018 [Online]. Available:
<https://github.com/hootsuite/nachos/blob/master/LICENSE>
- [17] Nginx, “Nginx”, December 2018 [Online]. Available:
<https://www.nginx.com/resources/glossary/nginx/>
- [18] Rabbitmq, “Rabbitmq Features”, December 2018 [Online]. Available:
<https://www.rabbitmq.com/features.html>
- [19] UML, “What is UML”, July 2005, [Online]. Available:
<http://www.uml.org/>
- [20] IEEE, “IEEE Citation Reference”, December 2018 [Online]. Available:
<https://iee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf>

- [21] Google, “Encoded Polyline Algorithm Format”, December 2018 [Online].Available:
<https://developers.google.com/maps/documentation/utilities/polylinealgorithm>
- [22] opensource.com, “What is Docker?”, December 2018 [Online].Available:
<https://opensource.com/resources/what-docker>