Bilkent University
Department of Computer Engineering

# Senior Design Project

*CarBuds*

# LOW LEVEL DESIGN 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

October 8, 2018

# Table of Contents

# 1. Introduction

In Universities, students create networks in order to solve their problems collectively. The problems can vary in terms of buying used/old books from other students or exchange stuff and most of the time sharing a ride. Most of these networks established in internet environment which not mean to use for a specific problem of students. Thus, Student can have a hard time to find help or another student fast and precise. Finding a ride or sharing your car with another person rises several issues, the most important one is the security. Inside the university campus most of the drivers can be sure that the hitchhiker most probably another student of the same school. However, outside the campus no one can assure that the hitchhiker even a student. To solve this problem students reach each other through un-specified networks and this arises another issue, time.

CarBuds is a mobile application that any university student can register to reach hitchhikers or can be hitchhiker to find a ride to a specific location. The application offers to find a car or a hitchhiker very quickly while assuring the validity of the other person's whether he or she a student of the proposed university. Therefore, with the help of the Carbuds students can reach each other for sharing a ride in a fast and secure way.

## 1.1 Object design and trade Off

### 1.1.1 Security vs Usage Count

Our users will be authenticated with university e mails. So only university members can use this application. If people could be to use all emails maybe there will be a lot more users but in this case users' security will be a problem.

### 1.1.2 Functionality vs Usability

Our first aim is developing a working matchmaking application. At first design of application may not be seen very good. When we made a working program then we will consider adding new functions. As a result we can say that we prefer usability over functionality.

### 1.1.3 Interface documentation guidelines

In this document we used this class interface format

| |
|---|
| **Class ClassName** |
| Class description |
| **Properties** |
| Properties and explanation |
| **Methods** |
| methods() and explanation of methods |

## 1.2 Engineering standards

When we preparing class interfaces, diagrams, database diagrams, we used UML guidelines [1] which used very commonly by engineers. Also our report is based on IEEE report format [2].

### 1.2.1 Definitions, acronyms and abbreviations

**MySQL:** Widely used, an open source database management system.

**SHA256:** A hashing algorithm that used for encrypting data in one direction, mostly used for password storing.

**HTTP:** HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

**TCP/IP:** Transmission Control Protocol/Internet Protocol, is a suite of communication protocols used to interconnect network devices on the internet.

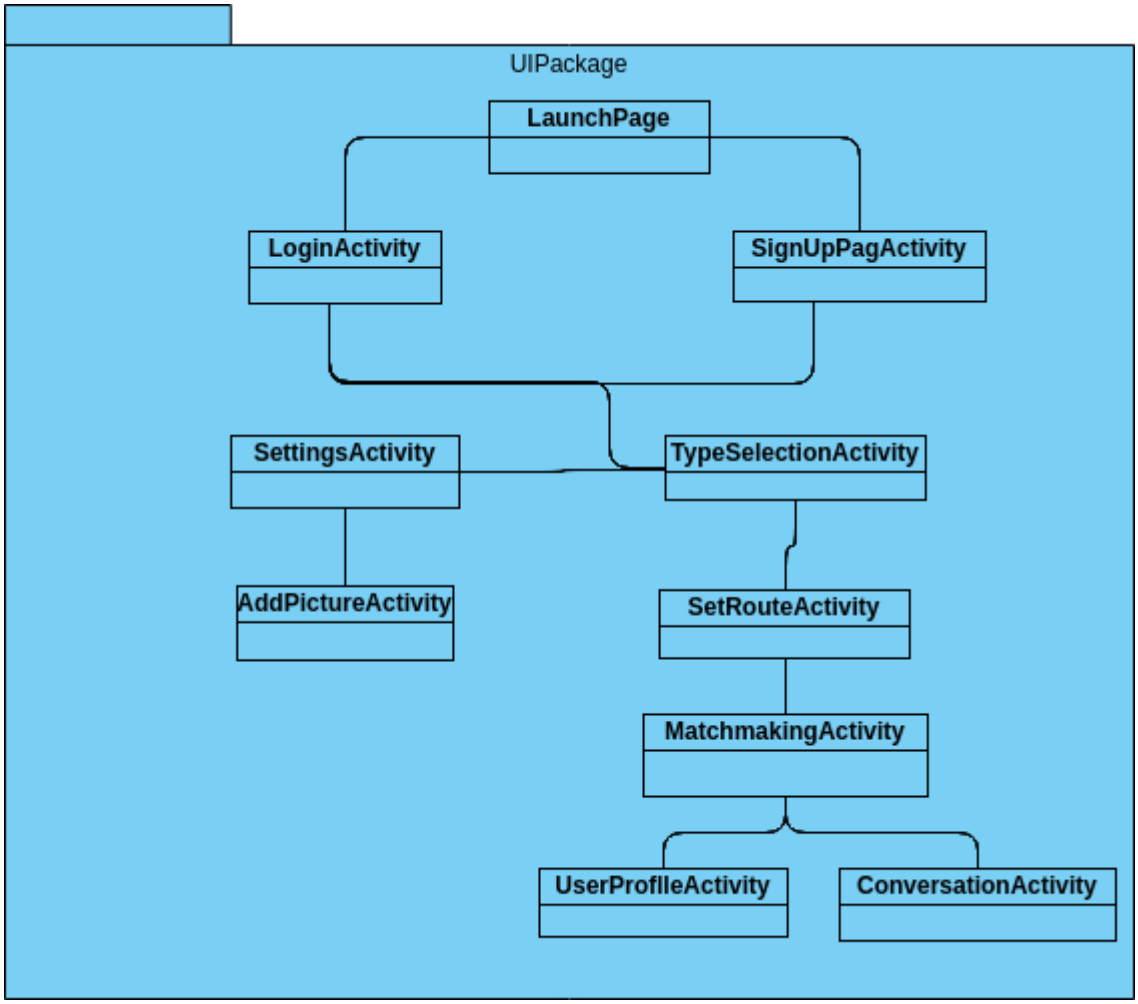
# 2. Packages

## 2.1 Client

Client side of Carbuds is an android application. With this application users can sign up and login their accounts, state their routes and they can be able to select users who goes same direction. After matchmaking they will be able to send messages with application.

### 2.1.1 View

In view side of application there will be user interfaces. This pages will provide interaction with user.




**LaunchPage:** Shows initial screen for application

**LoginActivity:**This class is responsible for login page. In this screen user will enter userid and password.

**SignUpActivity:** If user doesn't have any account user will create an account with this activity.

**SettingsActivity:** This class is responsible for showing app settings and user settings. Also user can change this settings with this activity.

**AddPictureActivity:** This class is a picking image class. User can change or add profile picture with using this class. This class provides camera and access to gallery

**TypeSelectionActivity:**This class is responsible for choosing usage type of application. In this screen user will select purpose of usage that time

**SetRouteActivity:**This class is responsible for route selection for both car owner and hitchhiker.

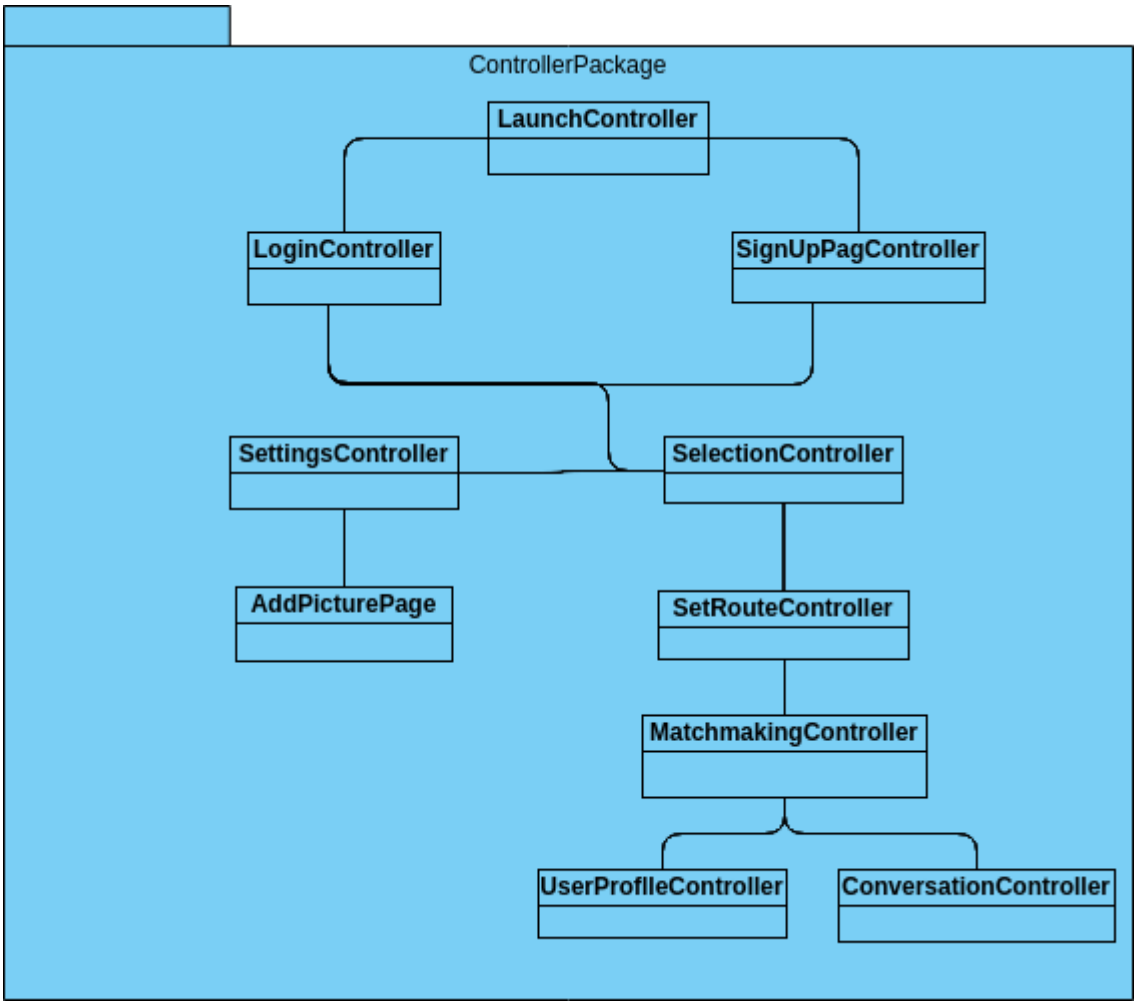
**MatchMakingActivity:** In this activity users will see bunch of users who has same route, in order. If both user approves each other they will be matched.

**UserProfileActivity:** This class is responsible for showing users profiles to other users.

**ConversationActivity:** This class is responsible for creating a connection between users after matchmaking

### 2.1.2 Controller

This package will be responsible managing user activities and creating connection with server.




**LoginController:**This class sends userId and password which entered by users to the server. After confirmation this controller gives access the application.

**SignUpController:** This class checks given information and sends them to the server. When server accepts creation of account this controller gives access to the application.

**SettingsController:** This controller get user information and settings from the server. If user changes any settings this controller send updates to the server.

**AddPictureController:** This class is responsible for sending pictures to server.

**TypeSelectionController:**This class is responsible for making arrangements after selection usage type

**SetRouteController:**This controller checks route and sends it to the server,

**MatchMakingController:** This controller gets users who has same preferences and concurrent routes.

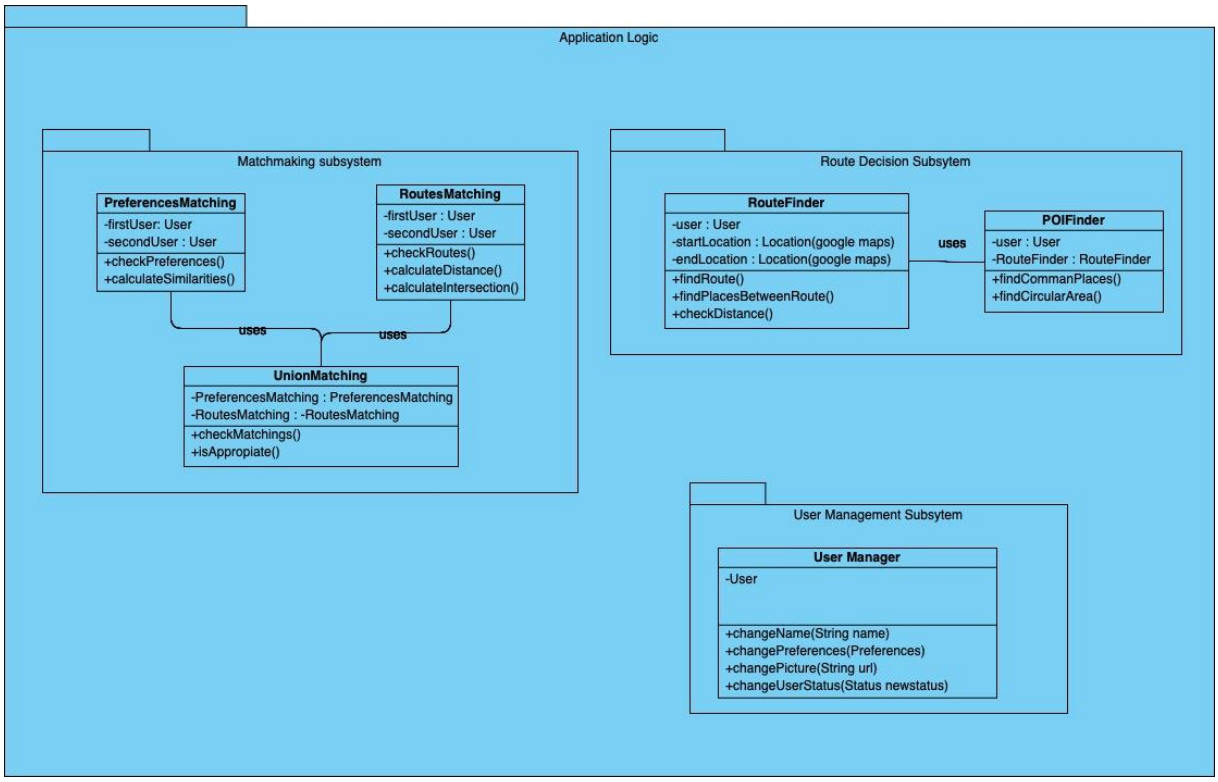
**UserProfileController:** This class is responsible for getting users profiles from the server.

**ConversationController**: This class is responsible for sending and receiving messages between users.

## 2.2 Server

### 2.2.1 Logic

This layer contains matchmaking, route decision and user management subsystems. Matchmaking subsystem is the place where matching the users happens according to user's preferences. Route decision subsystem use matchmaking subsystem in order to get the detail of the user's location and getting directions for the driver to take the passenger. Since route decision subsystem communicate with matchmaking subsystem there are in the same layer. User management subsystem is for managing user information.




The Matchmaking Subsystem is the main algorithmic logic subsystem in the application. Since CarBuds depends on matchmaking between a hitchhiker and a driver, in this subsystem all user preferences will be take in consideration during the matchmaking process. This subsystem generates appropriate amount of possible matchmaking candidates for the users.

After user decides his/her target destination both as hitchhiker or driver system firstly processes the routes of each user in order to make them quarriable for the matchmaking subsystem. Since Hitchhiking does not only involve finding an exact ride to the target destination, hitchhikers benefit some commonly used routes in order to make their journey to the target destination easier. Subsystem processes each route starting and finishing points and

deciding possible midpoint stops in the route or alternative routes that still can be applicable for the users.

Each user has its own profile page for both as hitchhiker and as driver that contains their information that they want to share with other users. Also, on the other hand, users have their preferences for their rides in order to be satisfied for most of the time for their trips. User Management Subsystem handles all the user data related operations and provide proper data for other application logic subsystems.

### 2.2.2 Network And Data

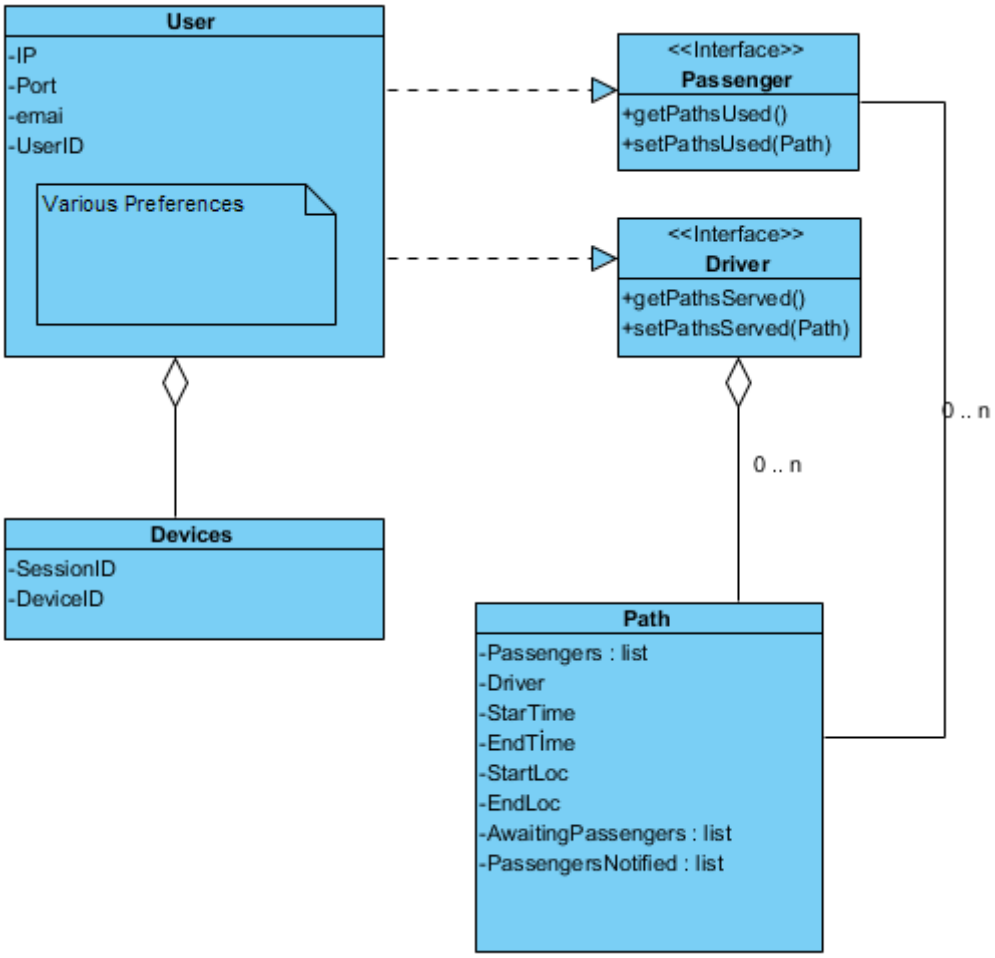



Figure: Class overview of the App server sided data, populated from the database

Network Subsystem handles the communication operations between server, database and the connection between server and the client. Between client and server system uses HTTP Protocols in order to send and receive information

which will be rendered to the user interface in the client side. Also handles all the query operations that comes from client side.

Messaging Subsystem handles the real time chatting between users of the application. Since each user is able to send messages to their matching users, Carbuds system handles the concurrency with a socket that runs with TCP/IP protocol which allows to process messages concurrently in real time. This provides a smoother and more responsive messaging.

Data Management Subsystem handles the database query operations for needed subsystems. Each subsystem depends on this subsystem because it contains and provides all the information about users' authentication credentials, matchmaking preferences, private messages and logs of the system.

# 3. Class Interfaces

In this section there will be class interfaces.

## 3.1 Client

In this section there will be clients classes. But throughout development process this class names can be change because of android structure.

### 3.1.1 View

| **Class** LaunchPage | |
|---|---|
| Shows initial screen for application | |
| **Properties** | |
| backgroundImage | background |
| loginButton | button for login page |
| singupButton | button for signup page |
| **Methods** | |
| | |

| **Class** LoginActivity | |
|---|---|
| This class is responsible for login page. In this screen user will enter userid and password. | |
| **Properties** | |
| emailArea | email area for user input |
| passwordArea | password area for user input |
| SubmitButton | this button for submitting user email and password |
| **Methods** | |
| | |

| **Class** SignUpActivity | |
|---|---|
| If user doesn't have any account user will create an account with this activity. | |
| **Properties** | |
| | |
| emailArea | email area for user input |
| passwordArea | password area for user input |
| SubmitButton | this button for submitting user email and password |
| | |
| **Methods** | |
| | |

| **Class** SettingsActivity | |
|---|---|
| This class is responsible for showing app settings and user settings. Also user can change this settings with this activity. | |
| **Properties** | |
| | |
| SettingsTable | It shows settings |
| | |
| **Methods** | |

| | |
|---|---|
| changeSetting(rowNumber) | user can change setting |

| **Class** AddPictureActivity | |
|---|---|
| This class is a picking image class. User can change or add profile picture with using this class. This class provides camera and access to gallery | |
| **Properties** | |
| cameraButton | It opens camera for taking picture |
| galleryButton | It opens gallery for choosing picture |
| **Methods** | |

| **Class** TypeSelectionActivity | |
|---|---|
| This class is responsible for choosing usage type of application. In this screen user will select purpose of usage that time | |
| **Properties** | |
| hitchhikerButton | it selects user type as hitchhiker |
| carOwnerButton | it selects user type as carOwner |

| **Methods** |
| --- |
| |

| **Class** SetRouteActivity | |
| --- | --- |
| This class is responsible for route selection for both car owner and hitchhiker with usign google map api | |
| **Properties** | |
| | |
| googleAPIConnection | It connect google map API with our user interface |
| **Methods** | |
| | |
| checkPreferences() | Check two users preferences |
| calculateSimilarities() | Looks for two people's preferences and calculate their similarities based on what they want and what they do not want |

| **Class** MatchMakingActivity |
| --- |
| In this activity users will see bunch of users who has same route, in order. If both user approves each other they will be matched. |
| **Properties** |

| | |
|---|---|
| userImagePanel | it is shows other users images |
| approveButton | This is positive button for matchmaking |
| declineButton | This is negative button for matchmaking |

**Methods**

| | |
|---|---|
| bringNext() | This method shows next user |
| showUserProfile() | This method opens User Profile |

**Class** UserProfileActivity

This class is responsible for showing users profiles to other users.

**Properties**

| | |
|---|---|
| userInformationTable | This table shows user information |

**Methods**

**Class** ConversationActivity

This class is responsible for creating a connection between users after matchmaking

**Properties**

| | |
|---|---|
| incomingMessageCell | it is for incoming messages |
| outgoingMessageCell | it is for outgoing messages |
| messagesTable | it is for messages ui |

**Methods**

| | |
|---|---|
| slide() | it is for showing old messages |
| writeMessage() | it is for sending new messages |

### 3.1.2 Controller

**Class** LoginController

This class sends userId and password which entered by users to the server. After confirmation this controller gives access the application.

**Properties**

| | |
|---|---|
| email | email for user input |
| password | password for user input |
| isFilled | it is a boolean which shows both e mail and password is filler |

**Methods**

| |
|---|
| |

| **Class** SignUpController |
|---|
| This class checks given information and sends them to the server. When server accepts creation of account this controller gives access to the application. |
| **Properties** |
| |

| | |
|---|---|
| email | email for user input |
| password | password for user input |
| isUserCreated | a boolean for user creation information |

| |
|---|
| **Methods** |
| |

| **Class** SettingsController |
|---|
| This controller get user information and settings from the server. If user changes any settings this controller send updates to the server. |
| **Properties** |

| | |
|---|---|
| SettingsTable | current settings |

**Methods**

| | |
|---|---|
| changeSetting(rowNumber) | user can change setting |
| getSettings() | it gets settings from server |

**Class** AddPictureController

This class is responsible for sending pictures to server

**Properties**

| | |
|---|---|
| picture | current picture |
| nextPicture | nextPicture |

**Methods**

**Class** TypeSelectionController

This class is responsible for making arrangements after selection usage type

**Properties**

| | |
|---|---|
| selection | it store selection result |
| | |

**Methods**

| **Class** SetRouteController | |
|---|---|
| This controller checks route and sends it to the server, | |
| **Properties** | |
| routeInformation | it stores route information which choosen by user |
| **Methods** | |
| sendRoute() | It sends route to the server |

| **Class** MatchMakingController |
|---|
| In this activity users will see bunch of users who has same route, in order. If both user approves each other they will be matched. |
| **Properties** |

| | |
|---|---|
| userImages | it stores user images |
| approvedusers | it stores approved users |

**Methods**

| | |
|---|---|
| sendApproveRequest() | when user approve a user it send request |
| bringMoreUsers() | it brings more users from server |

**Class** UserProfileController

This class is responsible for getting users profiles from the server.

**Properties**

| | |
|---|---|
| userInformationTable | This table shows user information |

**Methods**

**Class** ConversationController

This class is responsible for creating a connection between users after matchmaking

**Properties**

| | |
|---|---|
| incomingMessages | it is for incoming messages |
| outgoingMessages | it is for outgoing messages |
| userID | it is for user ID |
| **Methods** | |
| sendMessage | it sends new message |
| checkfornewMessage | it checks for new message |

## 3.2 Server

### 3.2.1 Logic

| | |
|---|---|
| **Class** PreferencesMatching | |
| Checks similarities of two users | |
| **Properties** | |
| firstUser | First user to check |
| secondUser | Second user to check |
| **Methods** | |

| | |
|---|---|
| checkPreferences() | Check two users preferences |
| calculateSimilarities() | Looks for two people's preferences and calculate their similarities based on what they want and what they do not want |

| |
|---|
| **Class** RoutesMatching |
| Checks two users routes to find if they are suitable match |
| **Properties** |
| firstUser — First user to check<br>secondUser — Second user to check |
| **Methods** |

| | |
|---|---|
| checkRoutes() | Check two users routes |
| calculateDistance() | Calculate the distance between two users |
| calculateIntersection() | Check if driver has a intersected route for hitchhiker |

| |
|---|
| **Class** UnionMatching |
| Takes both PreferencesMatching and RouteMatching in order to decide if two user is suitable for each other |
| **Properties** |
| |

| | |
|---|---|
| PreferencesMatching | Checker for preferences |
| RoutesMatching | Checker for routes |

| |
|---|
| **Methods** |

| | |
|---|---|
| checkMatchings() | Compares two matchings |
| isAppropriate() | If both preferences and routes matching is appropriate for both user returns success |

| **Class RouteFinder** | |
|---|---|
| Find routes for users | |
| **Properties** | |
| User | User whole route will be calculated |
| StartLocation | Start Location of the user |
| EndLocation | End Location of the user |
| **Methods** | |
| findRoute() | Finds the route with the help of Google maps api between start and end location of the user |
| findPlacesBetweenRoute() | Finds the places between start and end location |
| checkDistance() | Find the distance between start and end location |

| **Class** POIFinder | |
|---|---|
| Finds point of interests in the route | |
| **Properties** | |
| User | User of the route owner |
| RouteFinder | Route of the user |
| **Methods** | |
| findCommanPlaces() | Finds point of interests in the route for finding suitable driver for hitchhiker's route |
| findCircularArea() | Finds a way if the hitchhiker's destination is close to driver's destination but not exactly same |

| **Class** UserManager | |
|---|---|
| Manages users profile and settings | |
| **Properties** | |
| User | User to be modified |
| **Methods** | |
| changeName() | Change user's name |
| changePreferences() | Change user's preferences |
| changePicture() | Change or add Picture URL for user |
| changeUserStatus() | Change user's status from driver to hitchhiker of from hitchhiker to driver |

### 3.2.2 Data

1. Authentication
    a. getUserLogInStatus
        i. Parameters: UserID or ICCID

ii. Response: The list of “AuthDevice”’s with attributes LoggedIn = true, ICCID = param or other devices used by the user, SessionID

b. getUserAuthDataFromEmail
   i. Parameters: Email
   ii. Response: The list of “AuthDevice”’s with attributes LoggedIn = true, ICCID = param or other devices used by the user, SessionID

c. createUser
   i. Parameters: Email, password.
   ii. Response: Full AuthUserInfo data of the created user.

d. logUserIn
   i. Prameters: ICCID, UserID, password
   ii. Response: SessionID

e. (Various Setters)
   i. Parameters: Primary keys, attribute name, value
   ii. Response: Changed entry

2. CoreFunctionality
   a. (Various setters)
      i. Parameters: Primary keys, attribute name, value
      ii. Response: Changed entry
      iii. Note: Setters on “Path” trigger change on “Enrolled” and “AwaitingEnrollment” tables.
   b. (Various destructors)

i. Note: Deletion of User removes the served Paths. Removes associated entries on “Enrolled” and “Awaiting Enrollment”

c. (Various getters)

i. Parameters: Primary keys, attribute name, value,

ii. Response: The list of elements with Primary keys, and attributes.

d. (Various inserters)

i. Parameters: Primary keys

ii. Response: Created entry

3. Essential Members

a. Entities

i. User: UserID and other user data.

ii. Path: PathID and other path data.

b. Relations

i. Serves( User - Path): Paths served by the user as a driver.

ii. Enrolled ( User - Path): Paths used by the user.

iii. Awaiting Enrollment( User - Path ): Users awaiting the drivers approval to be enrolled into the path.

c. Triggers

i. Any addition to “Enrolled” triggers checking and adjustment of “Awaiting Enrollment”

ii. Any removal of an entry from “Serves” triggers adjustment of “Enrolled” and “Awaiting Enrollment”
iii. Any change in an entry from the “Path” table triggers adjustment of “Enrolled” and “Awaiting Enrollment” parameters (not shown above).
iv. Deletion of User deletes the served paths, deletes the entries from “Enrolled” and “Awaiting Enrollment” tables.

4. Authentication Members
    a. Entities
        i. AuthUserInfo: User ID, regularly permanent credentials
        ii. AuthDevice: Devices used by the user.
    b. Relations:
        i. Uses( AuthUserInfo - AuthDevice)
5. Logging Members
    a. Entities
        i. Shadow tables for all tables together with extra 3 attributes, an element creation timestamp, a change timestamp, and a deletion timestamp.
    b. Triggers
        i. Sustaining the shadow tables.
6. Messaging Members
    a. Entities
        i. UserConnectionInfo: IP and port numbers.

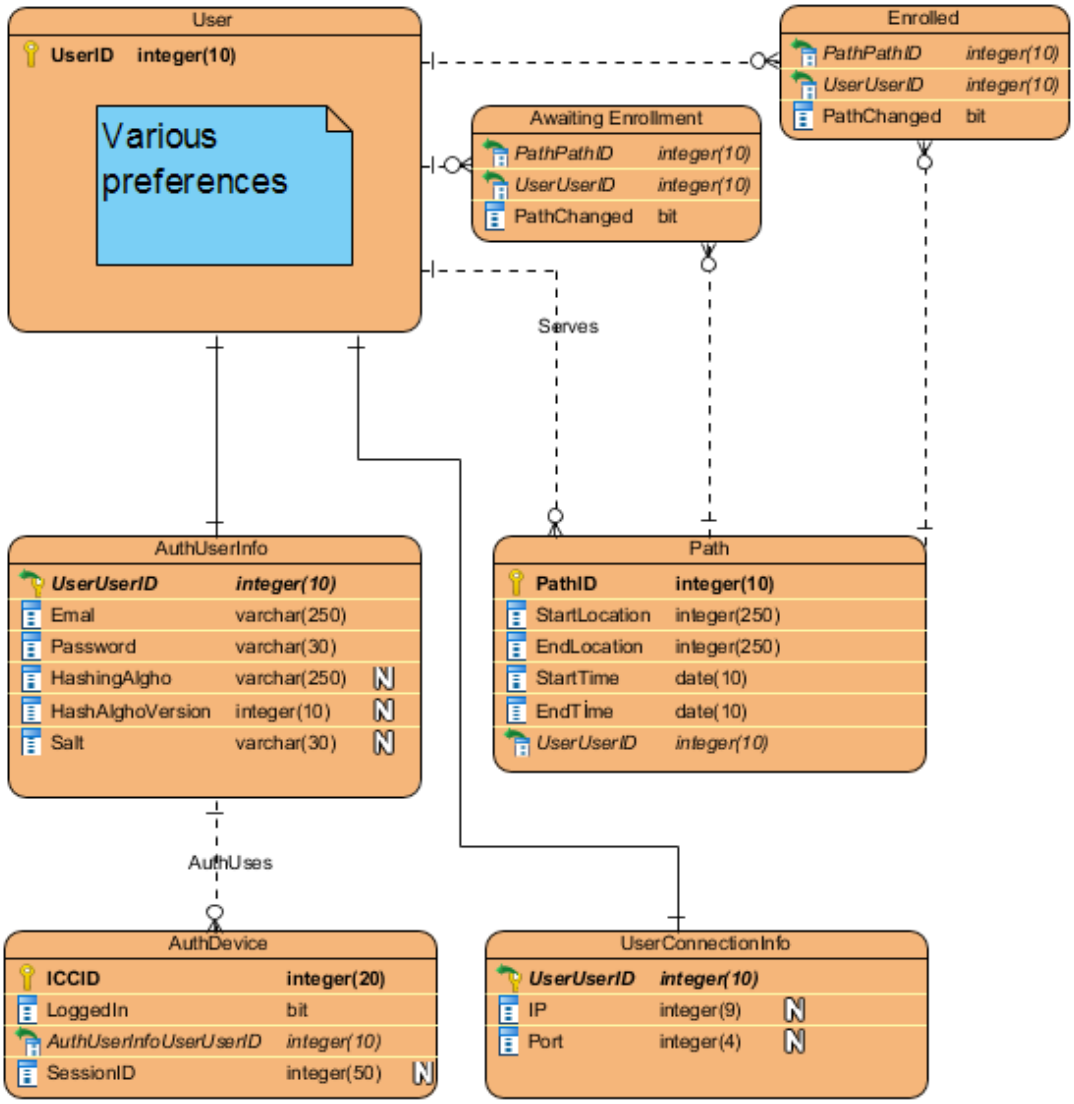



Figure: ER diagram of the relational database, logging is excluded.

## 4. Glossary

## 5. References

[1] UML, “What is UML” July 2005. [Online]. Available:

http://www.uml.org/ [Accessed 5-Oct-2018]

[2] IEEE, “IEEE Citation Reference” [Online]. Available:

https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf
[Accessed 5-Oct-2018]