CS 307 Design Document BoilerRide

Team 17

Nadeem Mahmood

Ming-Da Liu Zhang

Karan Teja

Konstantin Diego Pandl

Srishti Gupta

Natasha Tyagi

1 Purpose

Currently, Purdue college students and other affiliates without cars find it particularly hard to commute on and off campus around Purdue, as well as other universities, in a cheaply and timely manner. Sharing a ride can help save money, be environmentally friendly and be fun. Unfortunately, there is no common central platform for it at Purdue as there are in other parts of the world. However, there are still many students, including international, that do not have a car that would appreciate ride sharing over taxis, shuttle services, trains or buses. With such a large student base at Purdue, which ensures a selected and secure user group, we will develop a mobile application that satisfies these needs. Boiler-Ride, as a mobile application, will allow students to ask for and offer rides to and from campus to get company or travel at a reduced cost in comparison to other services.

The system consists of an Android application and a backend web-server to connect passengers and drivers. The application will be defined by the components of the system, their individual functionalities, and the interaction between the different components. The system that we are designing will manage the ride sharing process by connecting clients together through a shared service hosted on a server.

1.1: Functional Requirements

1. Allow users to authenticate themselves.
    a. Create an account using their email(gmail, yahoo).
    b. Confirm their Purdue email.
2. Allow users to offer rides.
    a. Users will be able to make a post with descriptive information about the ride they are offering.
        i. Basic required information will include the users start location, their destination, and the date of travel.
        ii. Additional information can be optionally included.
3. Allow users to find rides.
    a. A user will be able to search for rides based on their search criteria.
    b. Invite other users having the same destination.
4. Allow users to write reviews about other users.
    a. Reviews can be written about other users(drivers) after a trip has been completed.
    b. Users can read reviews written about other users.
5. Allow users to edit their profile.
    a. Users can edit personal information to be included on their profile.
    b. Users can set their privacy settings.

General Objectives

Usability

One of the main advantages of BoilerRide compared to similar solutions (like the Purdue Facebook rideshare group) is the increased usability, which comes from the clear App design and the ride seeker and offerer matching.
The App should be easy-to-use even for new users.

Security

We want to keep the user data secure. Therefore, we don’t save payment information but use third party payment apps to process payments. We want to keep the personal information like name, email, phone number on a secure place. Furthermore, security is increased because all users are affiliated to the purdue community.

Performance

BoilerRide should have real time performance to organize ride shares quickly. Notifications of matching should come out immediately.

2 Design Outline

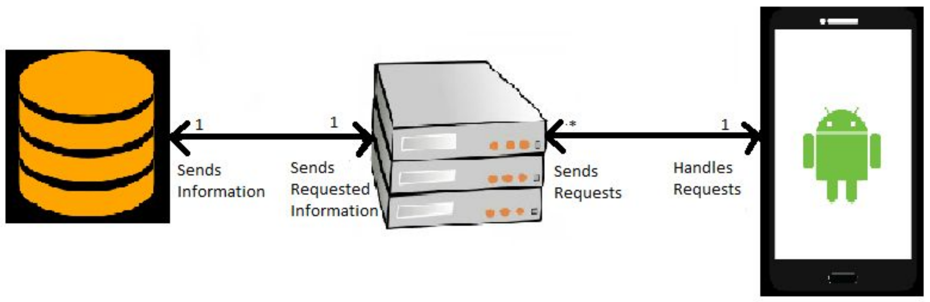
BoilerRide is an Android application that will allow drivers and passengers to connect with one another with posts and using geolocation to find fellow Purdue riders going to the same destination as you. It will utilize a Client-Server design paradigm, with a single server handling numerous client requests. The server side will use the Model-View-Controller (MVC) software architectural pattern and access a database where all user information, ride information, among other data is stored and accessed.

The system we are designing will consist of three major components:

- Client
- Server
- Database

2.1 High Level Overview of the System

We will have numerous clients that are connected to our central server. Each client will send requests to the server and then the server will need to decide which action the server should take depending on what the request was; either to upload a new screen page, send a request/query to the database for missing data or even possibly both.



2.2 System Component Details

The functionality of the three major system components are given in further detail in this section.

Server

The server will be hosted remotely using Firebase and will manage our API, implemented using Java .The server will have the following purposes:

- The server acts as an interface for communication between the Android application and the database.
- The server will accept requests from the client to access or modify information contained in the database.
- The server will send responses to the client after requests are made.
- The server will connect with the database to access, store, and modify data relevant to the system.
- The server ensures the secure transmission of data between the system components.

Client

The Android application will be the user interface for the clients that allows them to communicate with the server by sending requests and see the responses sent back from the server. The Android application will have the following purposes:

- ❏ Send requests to the server.
- ❏ Receive responses from the server and update the UI as needed.
- ❏ Allow the user to perform the following actions:
    - Authenticate themselves for use of the application using purdue email address
    - Update their profile information.
    - Update their application settings.
    - Search for upcoming rides being offered by other users.
    - Post an offer for a ride for other users.
    - View their past completed rides.
    - Write reviews and give ratings to other users.

Database

The database will store persistent information that is relevant to the system. Information to be stored will include the following:

- User credentials for login and API access tokens.
- User access level and function access level map.
- User settings including notification preferences.
- User profile information including name, profile image, purdue email address, phone number, and unique identifiers.
- User reviews and message.
- Feedback given from the user about the application.

3 Design Issues

3.1 Functional Issues:

Issue 1: Map API decision

Option 1: Open Maps

Option 2: Google Maps

Option 3: mapQuest

Option 4: Yahoo

Option 5: Bing

Decision: We decided for Google Maps. Google Maps is easy to integrate in an Android app since it is also offered by Google and provides high quality and up to date maps with unparalleled documentation for developers.

Issue 2: Which email services to allow

Option 1: Only Purdue email accounts

Option 2: Any email account

Decision: We decided to go with allowing account creating with only Purdue emails to ensure that only Purdue affiliates could use the application and as a security measure.

Issue 3: User Profiling

How to let users know each other and develop trust or sense of safety?

Option 1: Every user has a profile accessible by other users

Option 2: Every user has end of trip reviews written for them or written by them

Option 3: Every user has an overall rating

Decision: We decided to incorporate all the options listed above. Every user must have a short profile publicly accessible. The user also has information like phone number, email etc. which are not released on the profile page but released to specific users when travelling along with them. The overall rating will be an average on individual ratings earned by the user either as a Rider or as a Driver. And it would be the best to have reviews at the end of the ride.

Issue 4: User classification

Classify users as Riders or as Drivers.

Option 1: Classify users as Drivers or Riders during the time of registration

Option 2: Classify users and let them switch between Driver and Rider within the app

Decision: Students who own and drive cars on campus may very well need rides to go to airports leaving their own cars at their rental/dorm/residences. Adding an extra switch between looking for rides (rider) and posting for rides (driver) may be unnecessarily time consuming. Thus we decided to not classify users as Drivers or Riders explicitly on the front-end but we rather assume that a person is a Driver when he posts for a trip and a Rider when he searches/requests to book a ride.Adding an extra switch between looking for rides (rider) and posting for rides (driver) may be unnecessarily time consuming.

Issue 5: Development Platform

Option 1: Ionic

Option 2: Android (Material design)

Option 3: iOS

Option 4: Bootstrap

Decision: Our very first intent was to just make one mobile application that works on all platforms and this would enable us to write and maintain less code and at the same time give users across different mobile platforms the same feel and experience. After a lot of doing and undoing we finally decided to develop a native Android application which would run smooth on various Android versions and offers convenient and familiar Material design.

<u>3.2 Non-Functional Issues:</u>

Issue 6: Backend API

Option 1: Amazon Web Services

Option 2: Firebase

Option 3: Parse

Decision: We decided for Firebase. Firebase is free, easy to use and fits our needs.

Issue 7: App programming language and framework

Option 1: Java
Option 2: C# (e.g. xamarin)
Option 3: C++ (Lighthouse, QT)
Option 4: JavaScript, HTML (Webapp)

Decision: We decided to use Java. Java Android development is the most convenient and most common, as it is also the standard by Google.

Issue 8: Programming IDE

Option 1: Android Studio
Option 2: Eclipse
Option 3: AIDE

Decision: We decided to use Android Studio. Android Studio is developed by Google, therefore it is the most up to date and stable platform for Android development

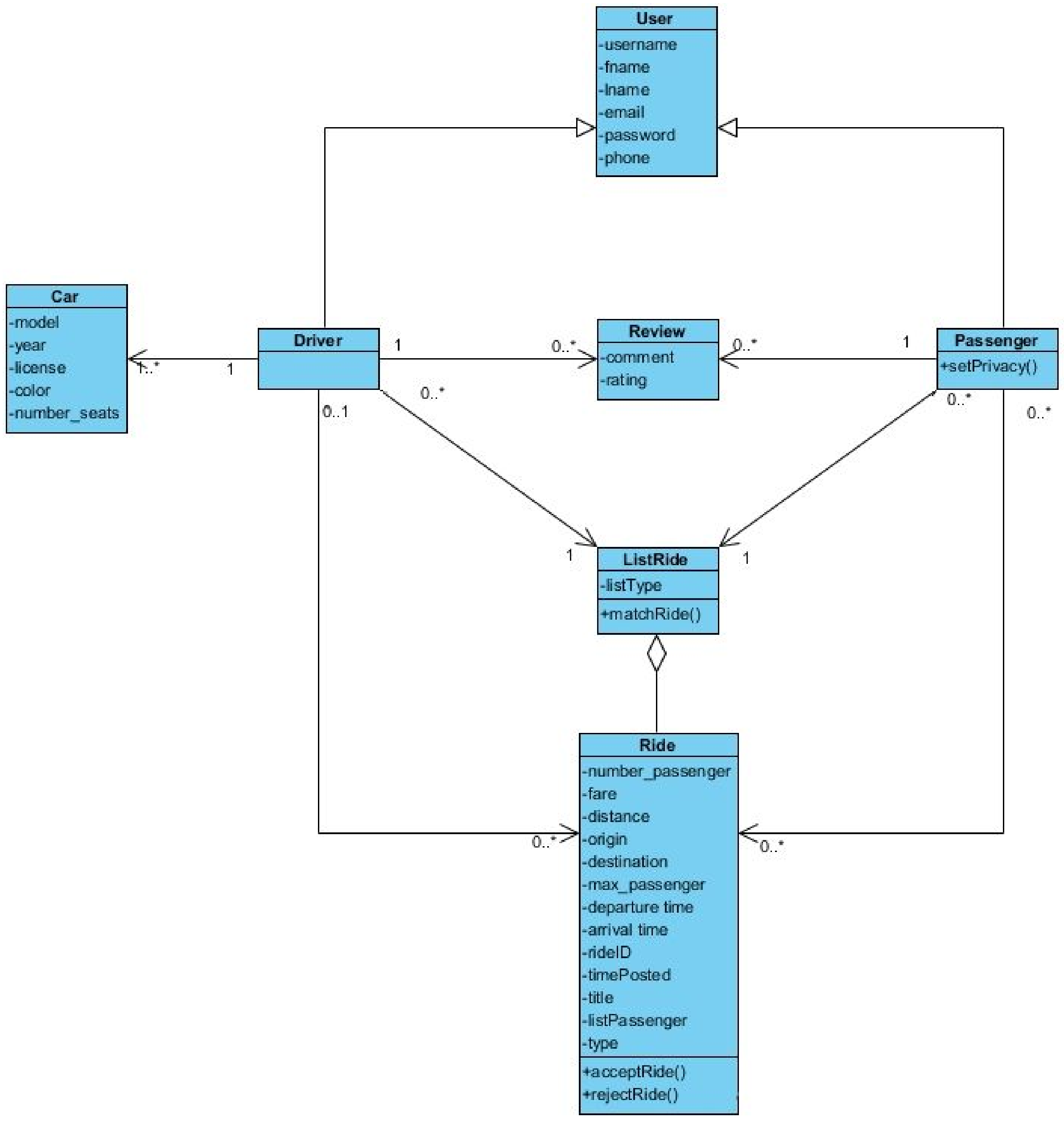
Issue 9: Target Android version

Option 1: Android 1.x and higher
Option 2: Android 2.x and higher
Option 3: Android 4.x and higher
Option 4: Android 5.x and higher

Decision: We decided to use develop for Android 4.x and higher. This allows us to use the modern Android Material Design, which is intended for Android 5, with support libraries. It would not be easy to do this wit Android versions lower than 4.0, furthermore, most devices today have a version of Android 4.0 or higher.

4 Design Details

4.1 Class Diagram



4.2: Description of classes and models:

User:

Entity that encapsulates the system’s main users(Drivers, Passengers)
It contains information about the users such as their username, first name, last name, email, password and phone number.
The username will identify a user uniquely.

Driver:

Entity that represents a driver. The Driver class inherits from User. It is associated with the classes Car, Review, ListRide and Ride. A driver can offer or accept a ride.

Passenger:

Entity that represent a passenger in our system. The Passenger class inherits from User. The passenger has association with Review, ListRide and Ride.
The passenger has an operation (in addition to the getters and setters of the attributes in User: setPrivacy() - The passenger has the option to set his privacy settings; that is if he wants the other passengers to see his ride or contact details .

Ride:

Entity that stores the information about a ride. It contains information about the fare, distance, origin, destination, maximum number of passengers, departure time, arrival time, ID that identifies uniquely the ride, title of the ride, time in which the ride was posted/offered and a list of passengers/driver that accepted the ride, type of ride(offered by driver, requested by passenger).
It has one operation: acceptRide() - when a passenger accepts a ride, his information will be stored that ride’s list. If the maximum number of passengers is achieved, the ride won’t let more passengers to accept it.
If the driver accepts the passenger’s ride request, his information is added to that ride’s list and no other driver can view this ride.

Review:

Entity that represents a comment and rating on a particular driver or passenger. A comment will be an explanation of the experience of the passenger during the ride with the driver and vice versa. A rating will be an assessment of the ride on a scale of 5.

Car:

Entity that represents the information about a car; the make, year, license number, color and number of seats.

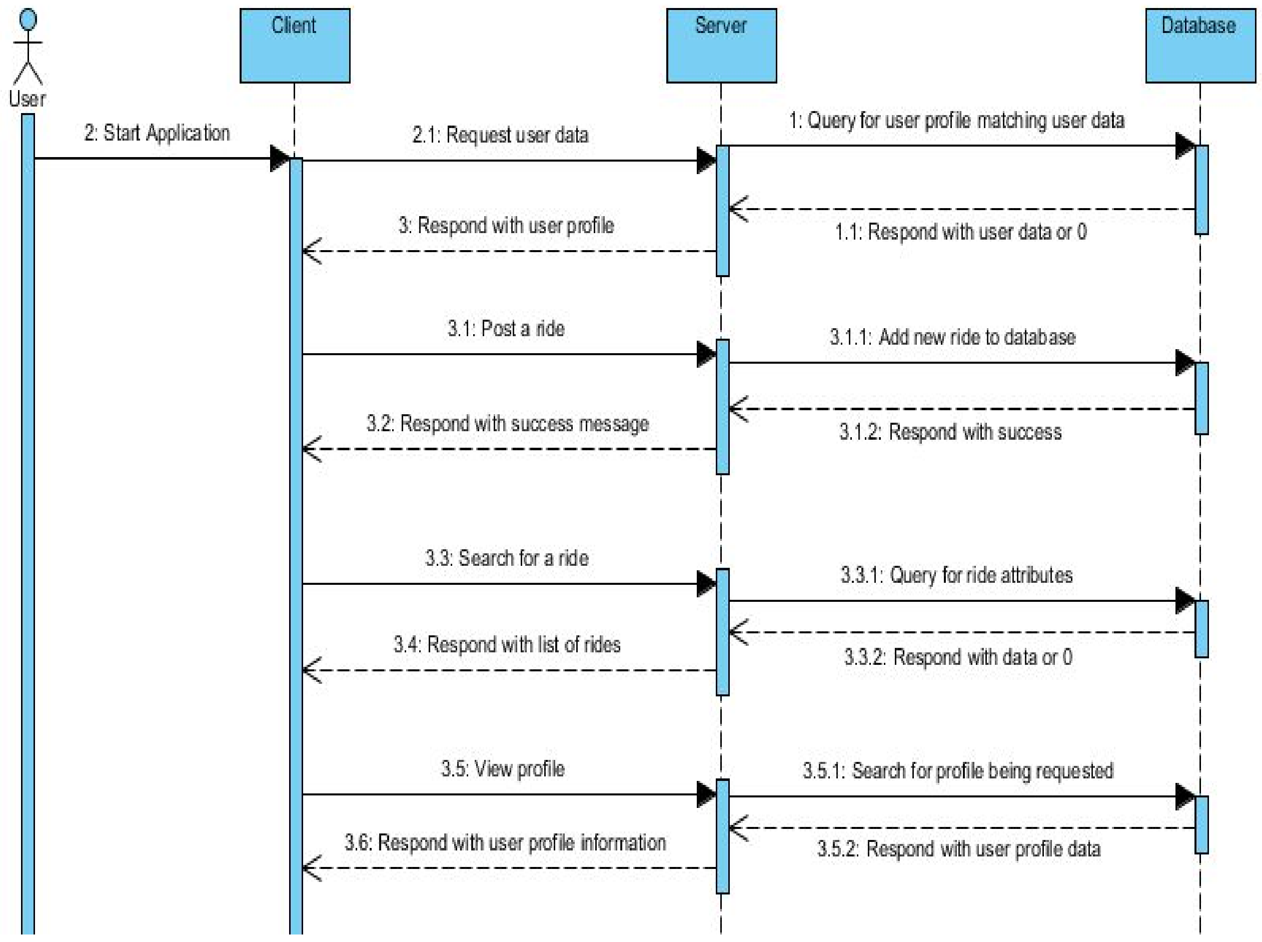
ListRide:

Entity that represents the list of matched rides as well as the list of the offered and requested rides.
It has an operation: matchRide()- it will match a ride according to the filter specified by the user (time, origin and destination, distance, price).

4.3: Broad Overview of Sequence of Events (Sequence Diagram)

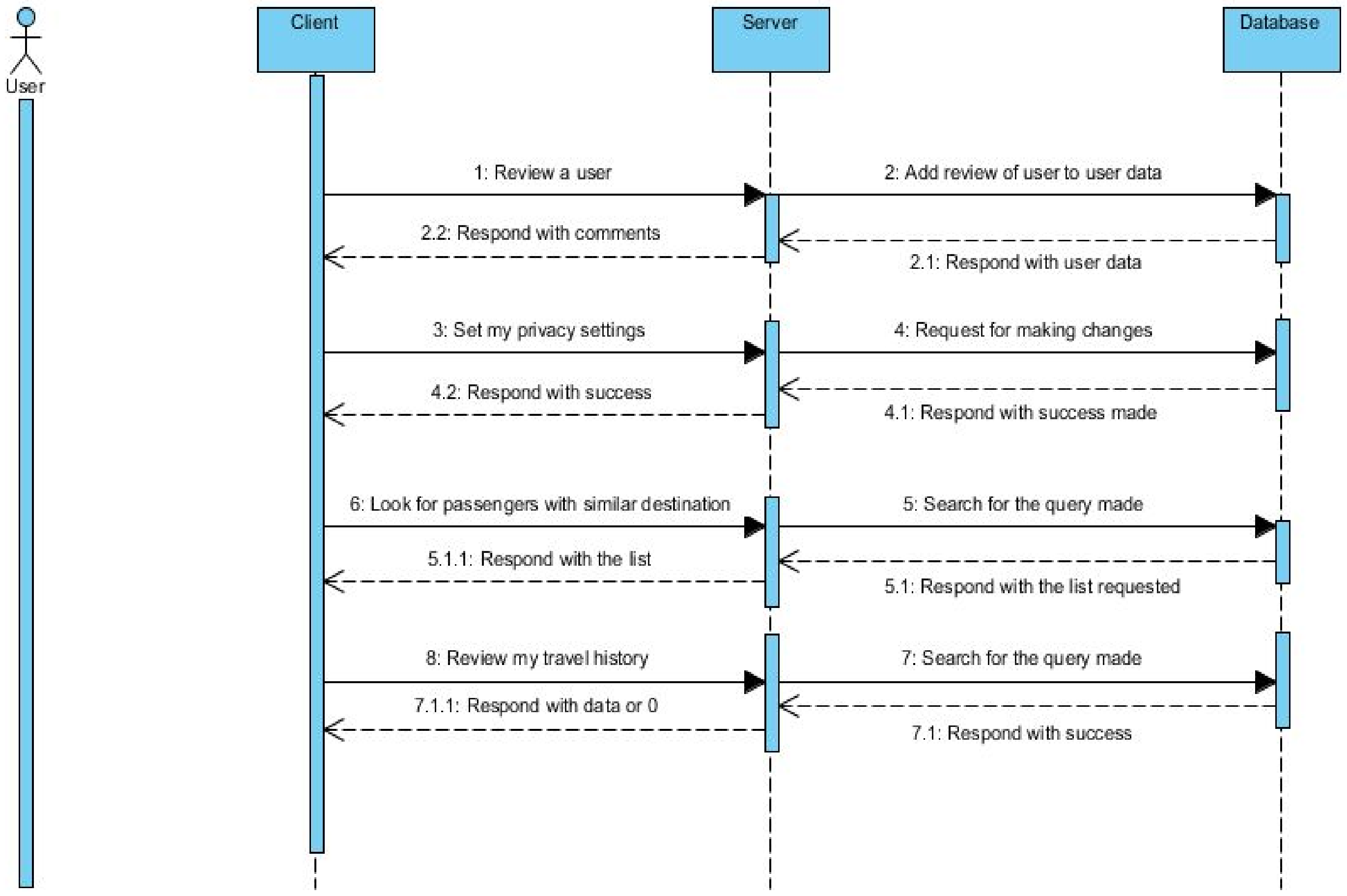
The following figure shows an overview of the typical sequence of events that will occur when a user accesses our app; showing the relationship between the client, server, and database. The client requests information from the server, the server will generate the new information if it posses it. If the server does not have the resources to complete the request, it will send a request to the database for the needed data to complete the request.

Sequence Diagram Part 1:


User
Client
Server
Database
2: Start Application
2.1: Request user data
1: Query for user profile matching user data
1.1: Respond with user data or 0
3: Respond with user profile
3.1: Post a ride
3.1.1: Add new ride to database
3.1.2: Respond with success
3.2: Respond with success message
3.3: Search for a ride
3.3.1: Query for ride attributes
3.3.2: Respond with data or 0
3.4: Respond with list of rides
3.5: View profile
3.5.1: Search for profile being requested
3.5.2: Respond with user profile data
3.6: Respond with user profile information

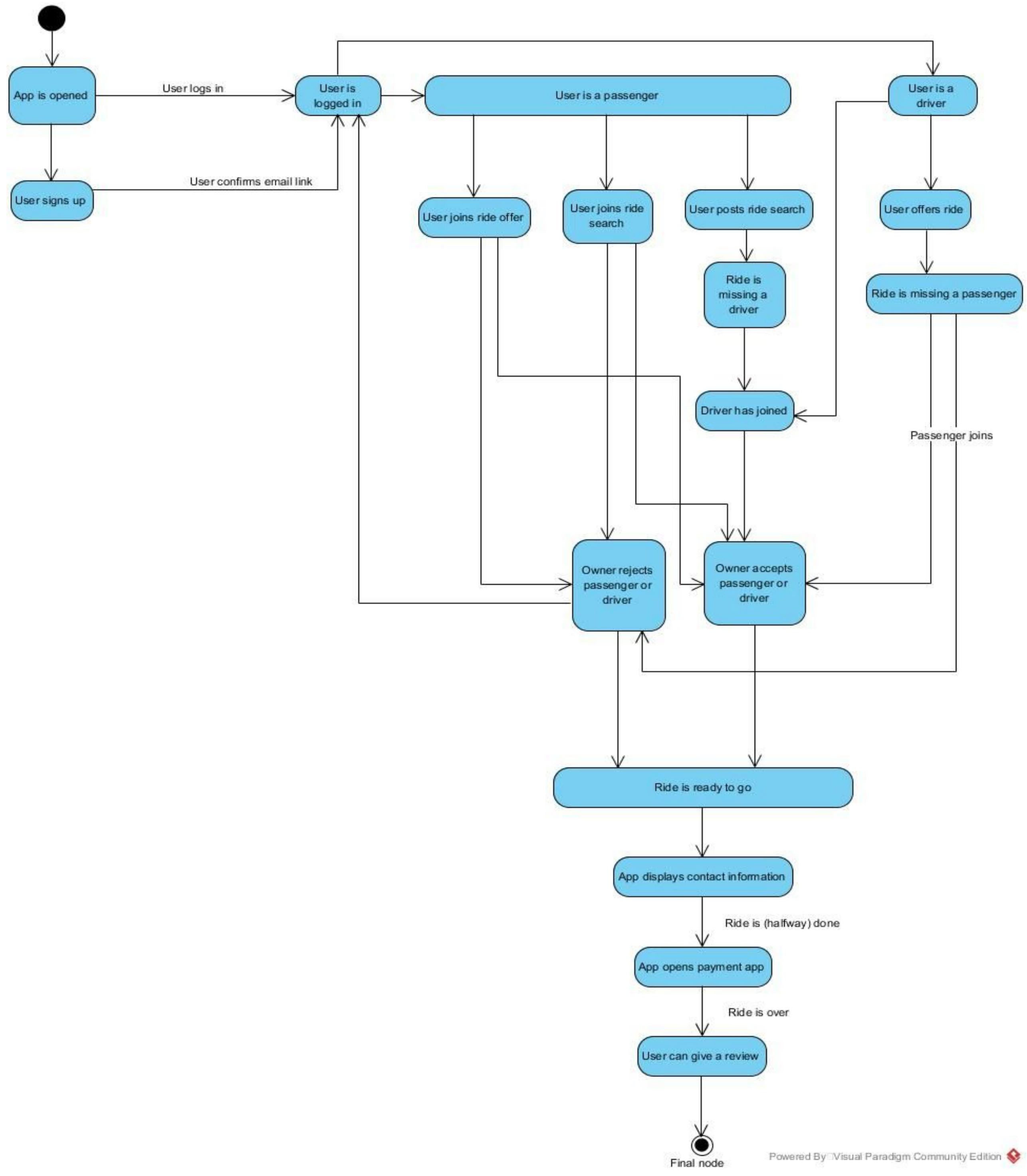
Sequence diagram Part 2:

A continuation of the sequence diagram on the previous page.


User
Client
Server
Database
1: Review a user
2: Add review of user to user data
2.2: Respond with comments
2.1: Respond with user data
3: Set my privacy settings
4: Request for making changes
4.2: Respond with success
4.1: Respond with success made
6: Look for passengers with similar destination
5: Search for the query made
5.1.1: Respond with the list
5.1: Respond with the list requested
8: Review my travel history
7: Search for the query made
7.1.1: Respond with data or 0
7.1: Respond with success

4.5: State diagram

State diagrams are used to give an abstract description of the behavior of the system. This behaviour is analyzed and represented as a series of events that occur in one or more possible states. Hereby, in our state diagram interaction between different classes is explained like when app is opened then user signs up and if he is new one then he creates an account and administrator sends him the verification email.


App is opened
User logs in
User is logged in
User is a passenger
User is a driver
User signs up
User confirms email link
User joins ride offer
User joins ride search
User posts ride search
User offers ride
Ride is missing a driver
Ride is missing a passenger
Driver has joined
Passenger joins
Owner rejects passenger or driver
Owner accepts passenger or driver
Ride is ready to go
App displays contact information
Ride is (halfway) done
App opens payment app
Ride is over
User can give a review
Final node
Powered By Visual Paradigm Community Edition