Software Requirements Specification

For

Rideshare System

Version 1.0 approved

Fengxiang Zhao, Yida Liu

Sep 21th 2018

Introduction

In this project, we would like to design a Android application that provides ride-sharing for students in Case Western Reserve University (CWRU). The riders (students who have cars) could use the application to post their trip information and they will be matched to some riders (students who do not have cars), whom the riders would take during the trip. As international students, we understand that how hard it could be to live without a car here. Therefore, we proposed this software system that is aimed to help more students. This application runs solely on the goodwill and CWRU students could use it free of charge.

This document contains the requirement specification for this project. It is composed by the following sections: Requirements, Design, Testing Plan, Responsibility and schedule. Requirement section covers the required functionalities of the software system. Design section contains a top-level overview on the design and a draft UI of the application. Testing plan section contains the test specifications that covers the functionalities of the software.

Definition

Client: Android App that used by users.
Server: Out backend server implemented by Django REST Framework.
Driver: User who would like to share their trips.
Rider: User who join trips shared by drivers.

Requirements

Functional Requirements

1. Client
    1.1. Trip Schedule
        1.1.1. Rider schedule contains the origin, destination, estimated time of trip.

1.1.2. Driver schedule contains the origin, destination, estimated time of trip, and driver's time constraint.

1.1.3. After the user specifies the trip information, a "send request" button will show up on the bottom of the screen.

1.2. Map

1.2.1. Client shows the Map using Google Maps API.

1.2.2. User can change the area in the map by dragging along the map.

1.2.3. User can use the input box on top of the screen to search an address or keywords; suggested results will be available for select.

1.2.4. After the user enters the origin and destination, the two points will be shown in the form of pin-points on the map; estimated trip duration will be shown.

1.3. Settings

1.3.1. Trips - Show the trips related to current login account

1.3.2. Profile - Show the information related to current login account and could be updated by user.

1.3.3. About - Show the current build/version information, and author.

1.3.4. Sign out - Invalidate the user token and bring the user back to the login page.

1.4. Login

1.4.1. Users should be able to login with the correct combination of credentials.

1.4.2. Users should be able to register new accounts with account name, email address.

1.4.3. Users should be able to reset their password using their email address if they forget the password.

1.4.4. Users' emails should be verified before they could use the rideshare service.

2. Server

2.1. Trip schedules

2.1.1. The server stores all posted and validated trip details.

2.1.2. The server provides trip information to the authenticated user only; trip can only be viewed by the users who created it.

2.1.3. The server runs an algorithm that matches the rider with driver by the constraints, including driver's time constraints, trip schedule, origin and destination.

2.2. Database

2.2.1. Contains information for registered users

2.2.2. Passwords should not store in plaintext to ensure the security.

2.2.3. Stores trip information of users, which includes start/destination, date and time, related user, number of seats, ID, status(Ready/Full/PassTrip)

3. Client-Server Communication

3.1. Driver's client can post driver schedule to the server; the driver will be notified if one or more rider is matched to the driver.

3.2. Rider's client can post rider schedule to the server; the server immediately returns a list of matched driver that rider could choose.

3.3. The server uses Google Cloud Message to push message to the client.

3.4. Client obtain the result message from the server after posting information and display the result to the user.

Other Non-functional Requirements

1. The email verification should be sent out immediately after the registration.
2. The Android application should have a smooth transition between activities.
3. User's request should be processed in a timely fashion; typically the result should be displayed to the user within 1 minute or less for a better user experience.

Charts And Diagrams

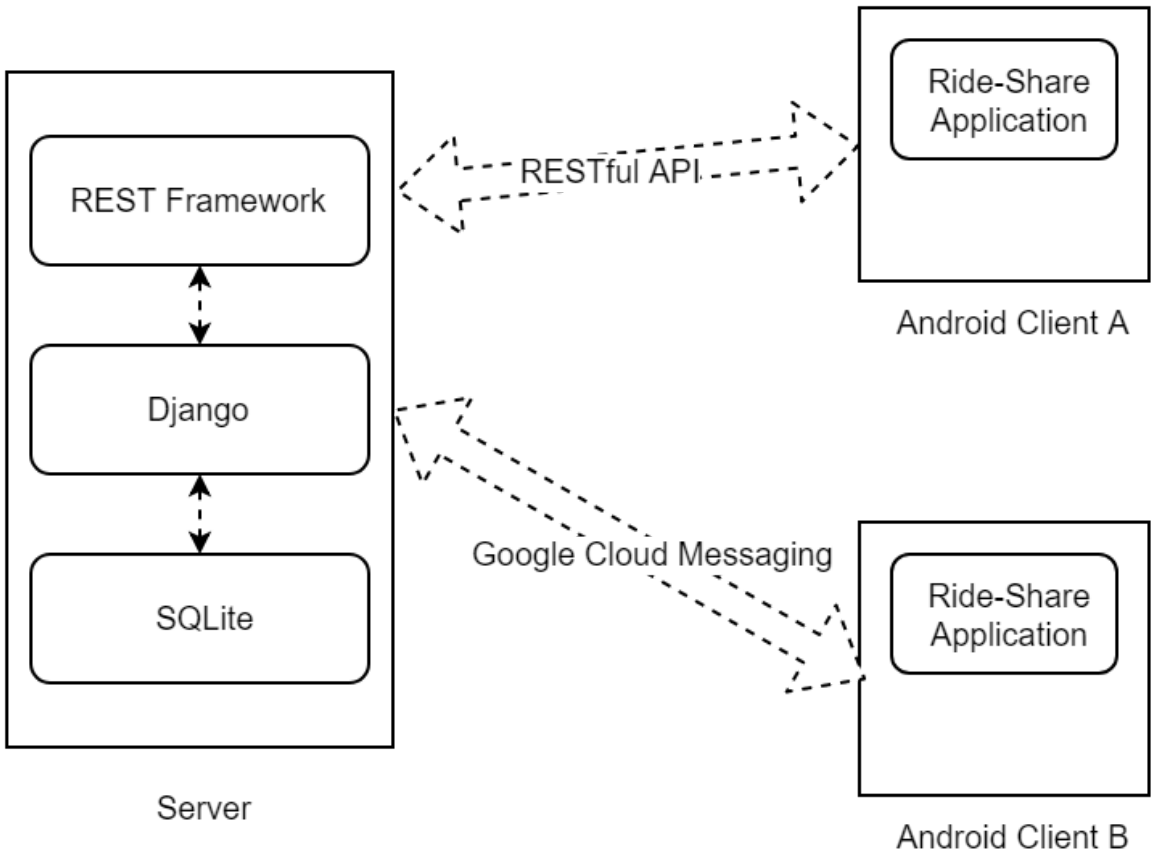

Figure 1: An Top-level overview on the communications between client and server

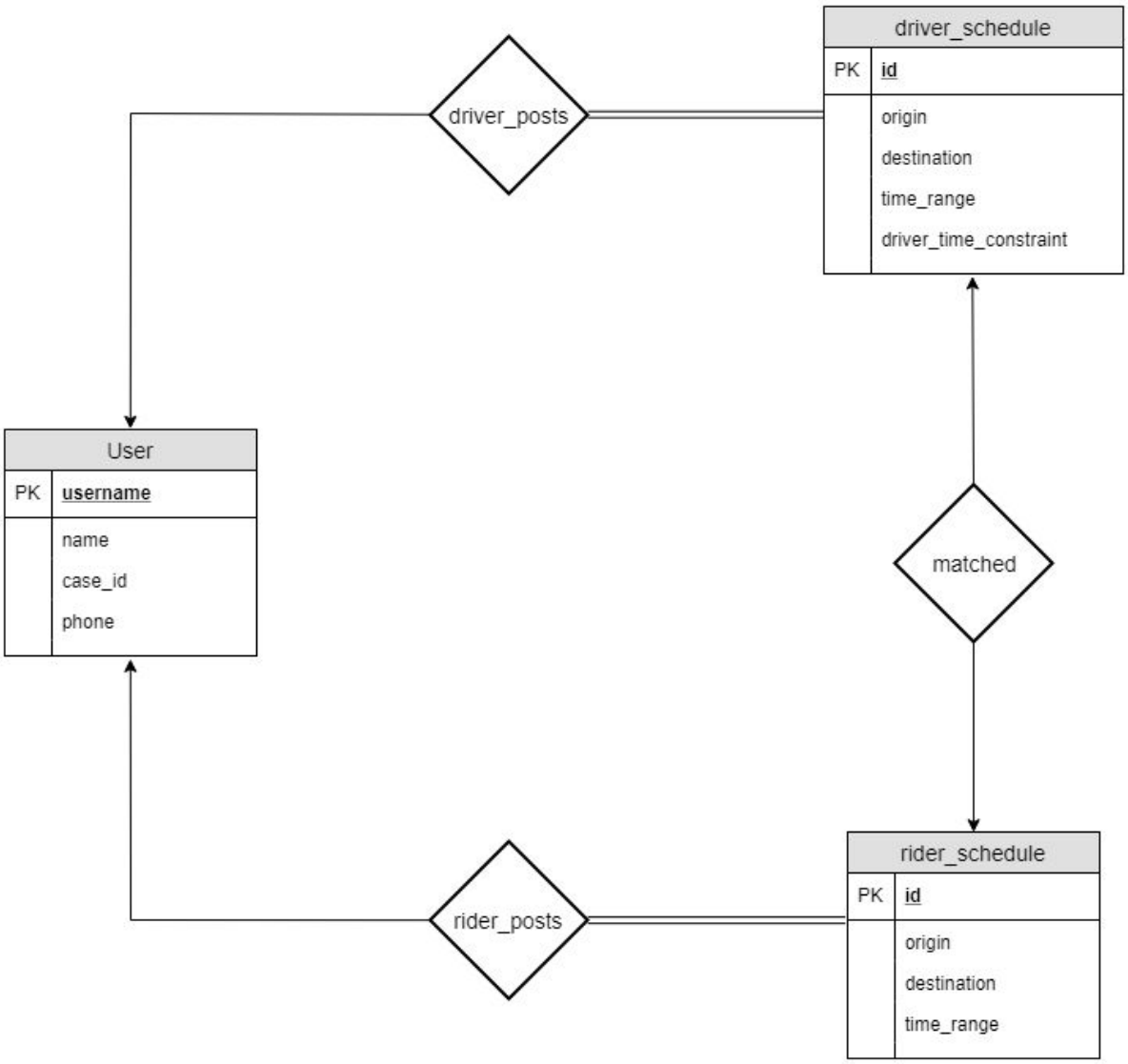

Figure 2: An E-R diagram of the data model on backend

Design

User Interview Design

The UI is implemented to facilitate users interact with map and make the basic operation. It consists with a map as the main page. Screenshots are listed below.

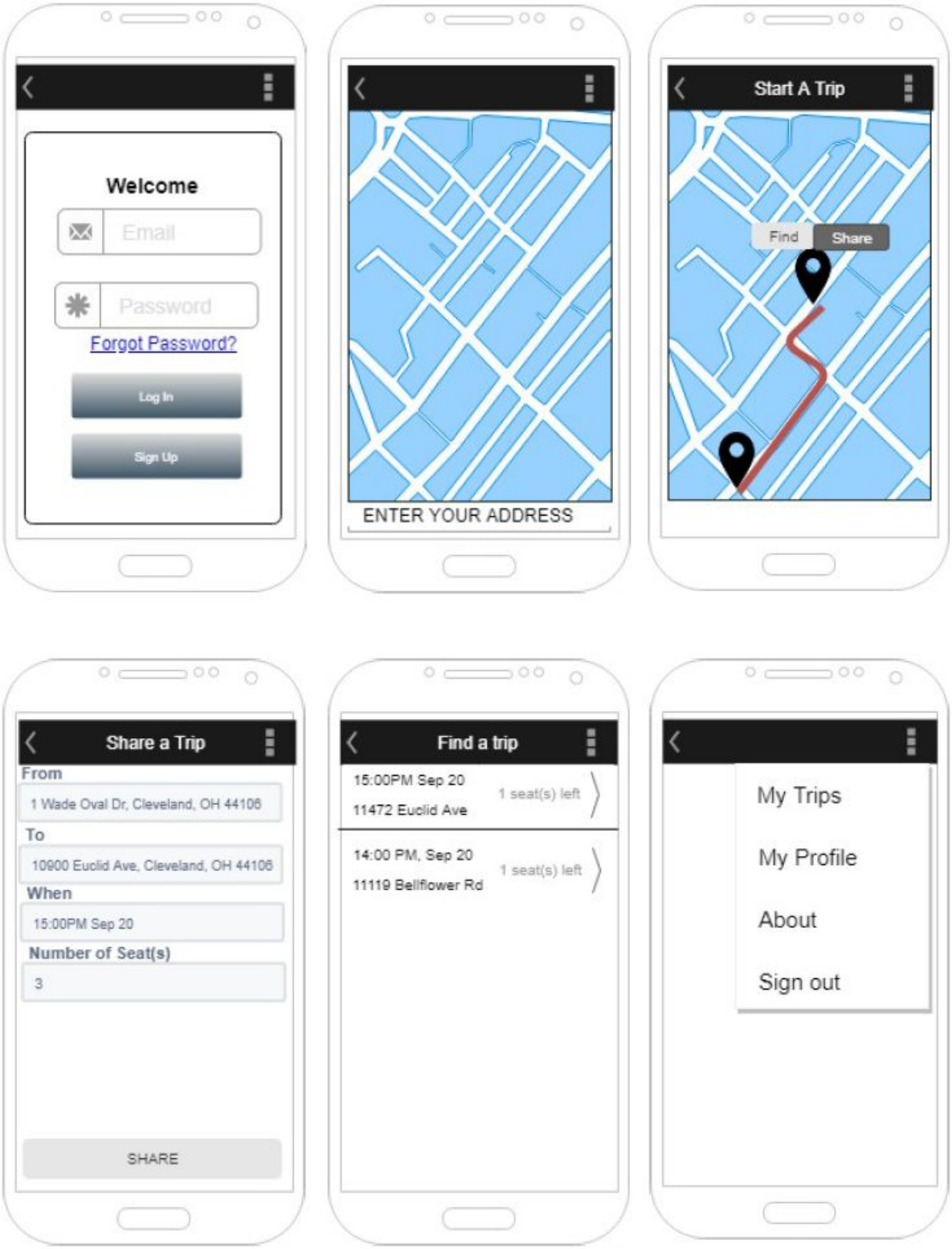

Figure 3. A draft UI design

Testing Plan

This section describes how we plan on testing the code for the Rideshare Program. Following this plan should ensure that the risk of having a defect in the program is minimized and confirm consistency of the product itself in performance and reliability. For this project, we will conduct unit test during the course of development and integration test after the main components are finished.

Functional Test

Functional testing is used to test the characteristics and operational behavior of the system to determine that they meet the design requirements. Various functions that need to be tested without considering the internal structure and code of the entire software. Generally, from the interface and architecture of the software product, test cases are written according to requirements. Below are some sample cases for functional test, but not limited to these.

Test ID	Description	Expected Result
1.	Client attempts to log on using credentials.	Server returns the sign in result. If the credential matches, the client was moved to the main page.
2.	Client attempts to add a new trip request.	If the request condition is valid, it is added into the database.
3.	Client attempts find a ride under given constraints.	Server returns the matching result and client presents the parsed results to the user correctly.
4.	Client attempts to remove a trip request.	If the client is logged-in and the trip request belongs to the logged-in user, the trip request is deleted from the database

5.	New user attempts to register.	The information of new user should be added to database if the information provided is unique in the database.
6.	User attempts to search a location by keywords, potentially fuzzy.	Client should present a list of results that mostly matched the key words.
7.	User attempts post a trip but there exists a trip that posted by him/her which has time conflict	The server should reject this POST operation and client should show an error message
8.	A user try to post a trip with time time of past	The server should reject this POST operation and client should show an error message

Integration Test

We will conduct integration test in our system. Integration test is used to test the combination of module. The main goal of integration test is to find problems related to the interface. If data passes through the interface, many problems may appear. We will require the integration test along the functional data and control flow paths. First, the input to the function is integrated in the bottom-up mode. The output of each function is then integrated in a top-down manner. The main advantage of this approach is the adaptability of early release with limited functionality.

Responsibility and Schedule

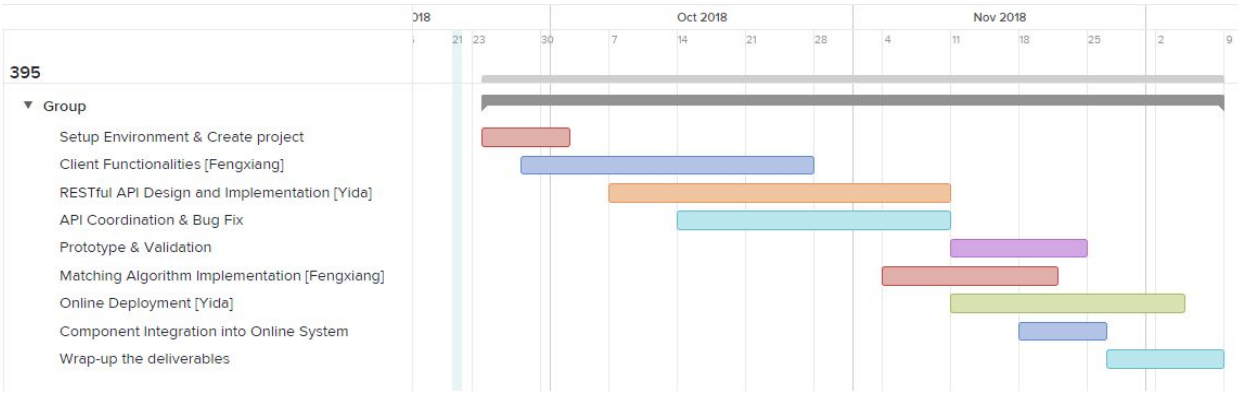

Figure 4: A Gantt chart includes the timeline, the milestones, and the individual responsibility of the project. Tasks without member's name shall be collaborative works.

Yida will work on the backend functionality with Django REST Framework, database management, and user registration. Fengxiang will work on the client functionality includes UI design and implement, protocol between client and server, design and implement trip matching algorithm. We will work cooperatively on including integration of client and server, testing, deployment of the service.