Jaguar Ride

CS M117 Project Report

**Abstract**

In this class, we studied the principles underlying wireless communication systems relevant to digital data communications. To apply what we have learnt in CS M117, our team (of three students) designed and developed an Android App, Jaguar Ride, that helps matching drivers with people who need a lift. Jaguar utilizes JAVA’s real-time GPS module to locate riders and drivers, and the Google Maps API to navigate drivers to riders. For front end, we used the Android Studio to design the user interface, and for back end we set up an AWS parse server to achieve communications between users. This report in detail explains the design process and implementation of our project, as well as motivations and the challenges our team has been through.

**Introduction & Motivation**

I, like other students who don’t own a car, always have trouble travelling from school to other places. Travelling with traffic transportation is too slow, while calling ubers all the time is too expensive. However, Jaguar Ride can save us from this dilemma. Jaguar Ride is an Android App that matches drivers with people who need a lift. However, unlike Uber or Lyft who charges users directly when they request for a ride, Jaguar Ride is totally free and leaves the freedom of how to share the cost to the driver and rider themselves. This feature can be very useful among students. For example, if Tom, who doesn’t have a car, wants to travel from Westwood to Orange County to spend the weekend with his family, there might be hundreds of driving students who are willing to give him a ride and share the gas cost. In this scenario, Jack can easily find a driver through Jaguar Ride and leave for home together with him on Friday.

The application can be separated into two parts - the driver side and the rider side. When a user logs into the app, he can choose to become a driver or a rider by choosing which side he wants to proceed to. On the rider side, the user can see his location on the map and request for a ride, while on the driver side, the driver can see a list of all nearby riders and choose one request to accept. Once the driver accepts a request, Jaguar Ride will navigate him to the rider through Google Maps. In this way, Jaguar Ride allows both people who drive and who does not drive to find their “car buddies” efficiently.

**Theoretical Background of Wireless Technologies**

**WIFI**

WIFI is an alternative network to wired network which is commonly used for connecting devices in wireless mode. Wi-Fi uses a radio technology known as 802.11, which can transmit data over short distances using high frequencies. 802.11 operates on either 2.4GHz or 5GHz depending on its type. The network’s central point is the access point, which is a router with transmitting antennas which route the transfer of data.

Typically, the range of this Wi-Fi access point to any Wi-Fi capable devices is about 300 feet outdoors and 150 feet indoors. This estimated range does not take into account any obstructions which may block the signal, including walls, solid objects or trees. The more obstructions in the signal’s path from the base station, the shorter the range will be. The lab we did in the discussion also indicated the same thing since the WIFI signal decreased as the intensity of microwave, which was a noise in this case, became stronger.

**GPS**

The Global Positioning System (GPS) is a satellite-based navigation system that consists of 24 orbiting satellites, each of which makes two circuits around the Earth every 24 hours.

The satellites transmit a high-frequency signal containing information packets with precise time at which each is transmitted. Receivers pick up the signal and use a system of trilateration to compute the position, comparing the time difference between transmission and reception of each packet, thereby calculating the distance to each satellite. As you move, the distance from the satellites changes, generating a small difference in the time course, which is used to update the location.

Because GPS provides real-time, three-dimensional positioning, navigation, and timing 24 hours a day, 7 days a week, all over the world, it is used in numerous applications, including GIS data collection, surveying, and mapping.

**Functionality & Manual**

**Hardware Requirement**

Running Jaguar Ride requires Google Play. During the development process, we used the Nexus 5X model emulator. To achieve full functionalities of Jaguar Ride, please use an emulator model with an API version no earlier than API 24.

**Login**

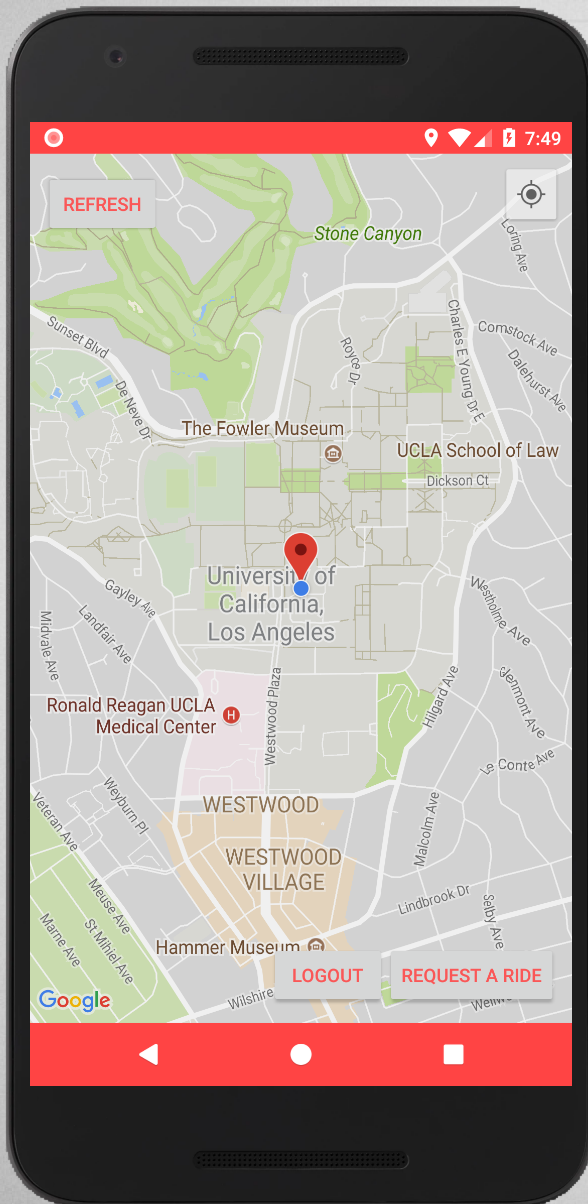
Our application, Jaguar Ride, consists of two main parts, the rider interface and the driver interface. Both rider and driver interfaces are connected to a user login page. On the login page, the user can choose to become a rider or driver with a switch (Figure 1). Since the application is still under the alpha test phase which does not have a large user pool, we have three username-password combinations, stored locally, to login to the app. They are:

“xuezhouyang” pin: 12345

“yangpochao” pin: 54321

“wangchengyu” pin: 11111

If a wrong username tries to login, the system will print out “wrong username”. Similarly, wrong username and password combination cannot access the application, either. However, as we moved on developing the app, we found the login system quite unnecessary, because the app does not charge users any fees and thus security is not the major concern.

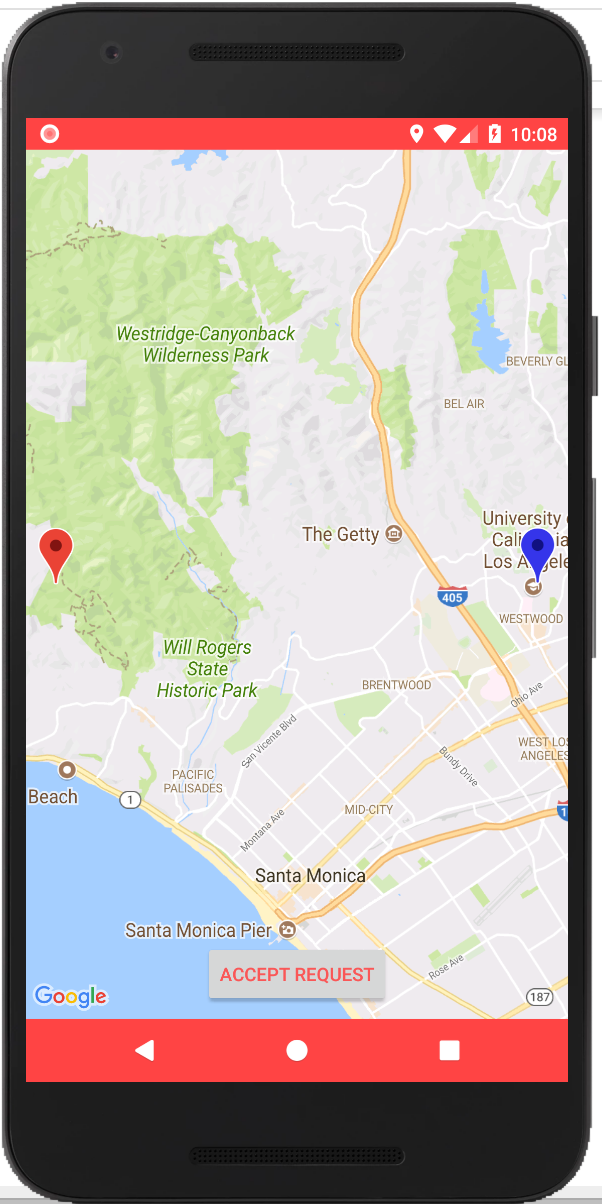
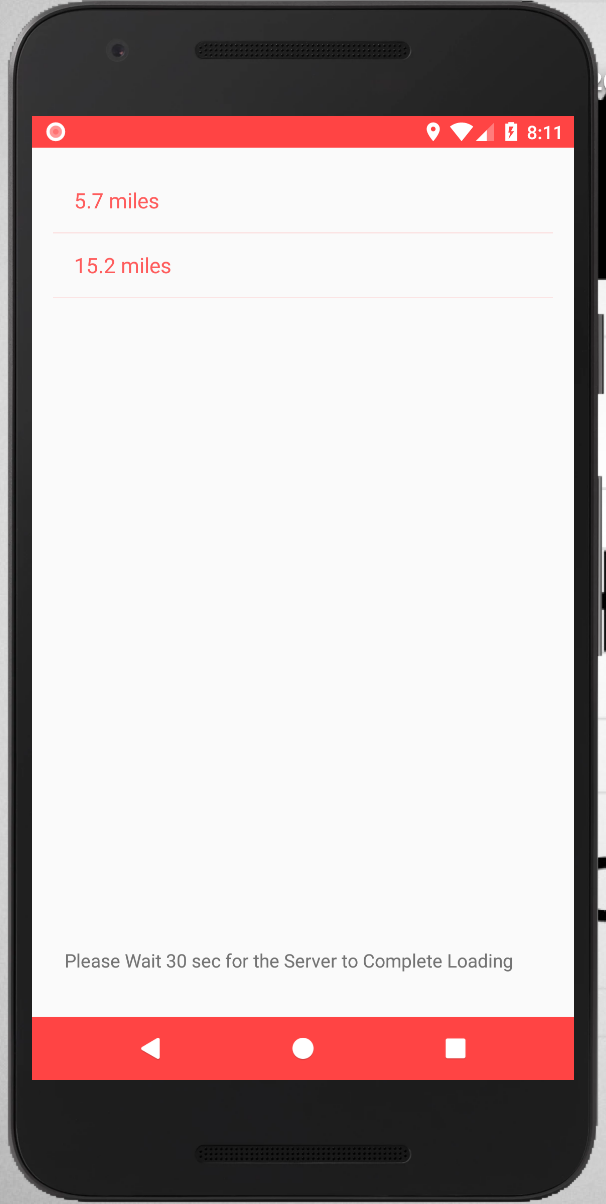
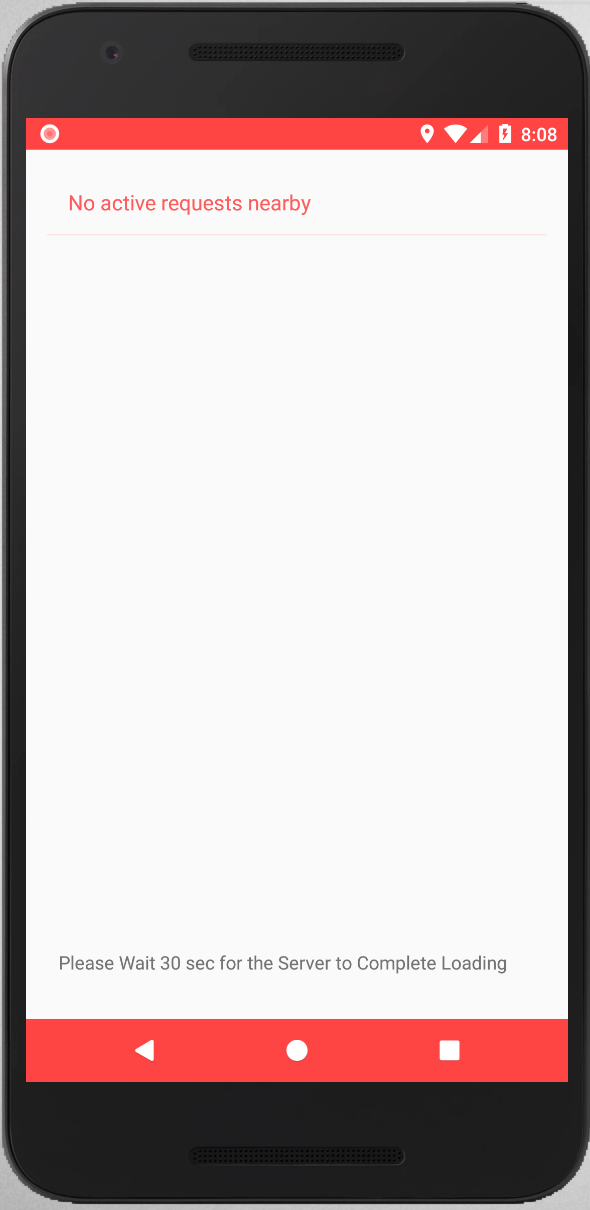
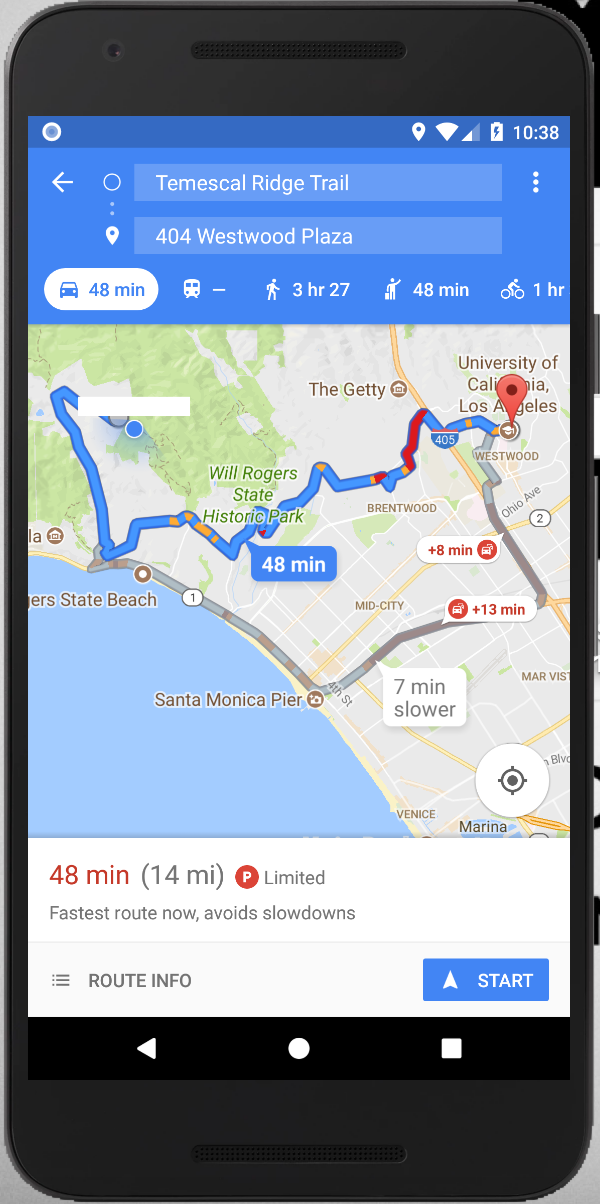
Figure 1, Login Page Figure 2, Driver Interface

**Rider Interface - Making Request**

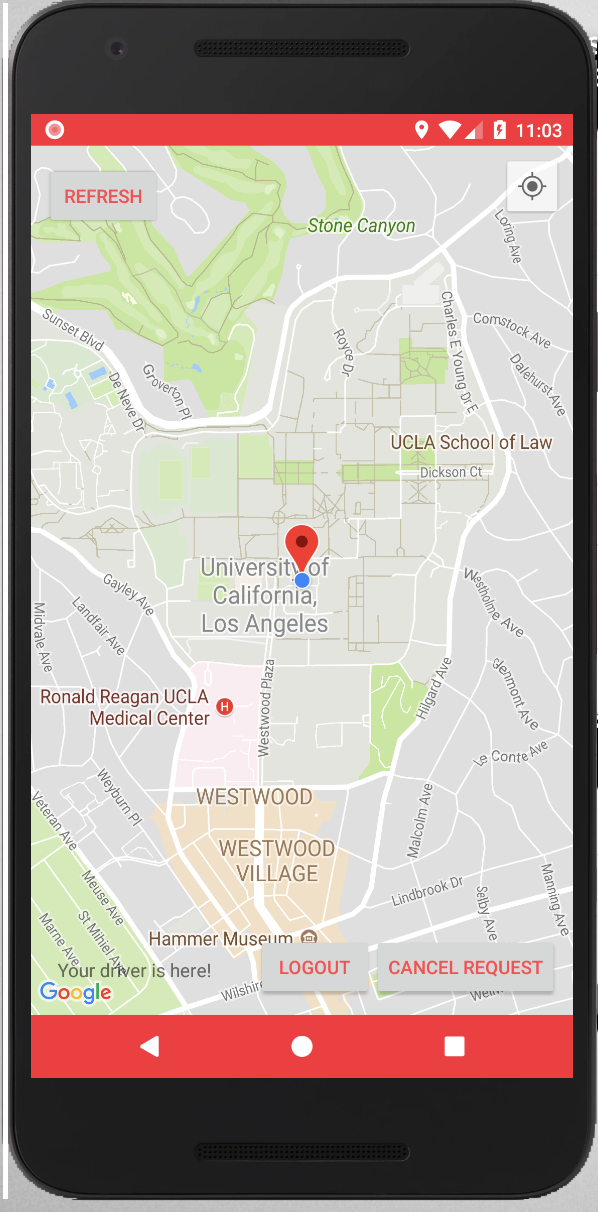
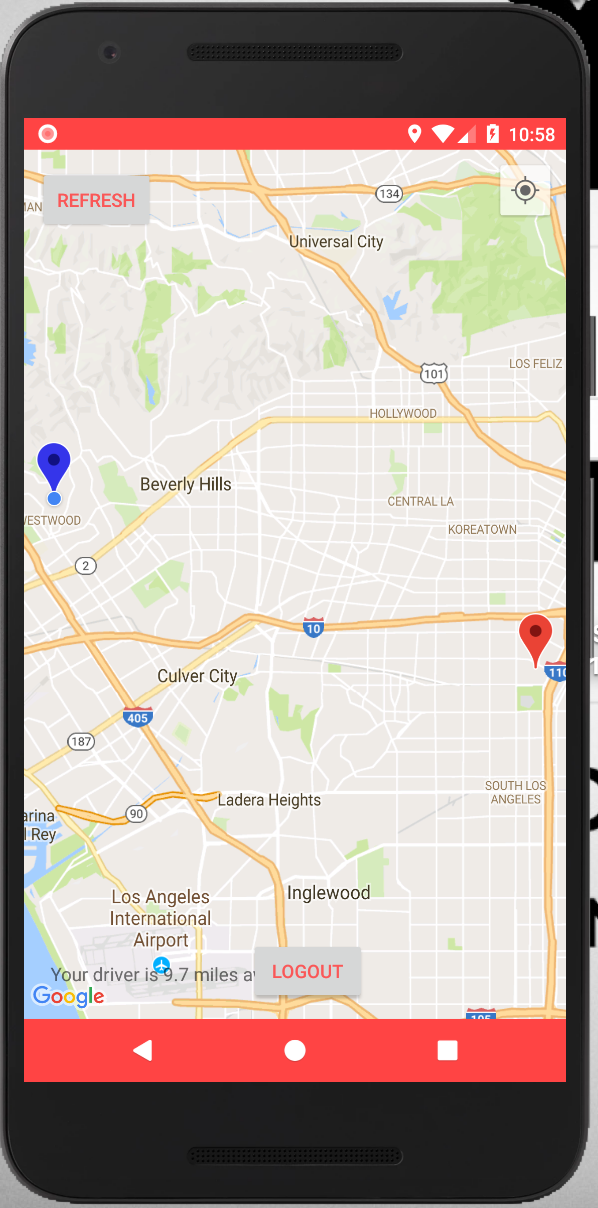
After logged in as a rider, the user can see his current location on the map (Figure 2). The map interface operates similarly as the google map interface. Users can move around or zoom in and out to inspect the surrounding area. (Zooming on the emulator is Ctrl + dragging mouse). To request a ride, the user simply clicks on the “Request A Ride” button, and his request will show up on the “request list” of the driver’s interface (Figure 3).

**Driver Interface - Viewing & Accepting Request**

After logged in as a driver, the user can see a list of surrounding riders (Figure 3). If no one makes a request, the driver will see an empty list, saying “No Active Request Nearby” (Figure 4). On the list, the driver can see the distance from himself to any rider. He can inspect the position of a rider simply by clicking him on the list, and it will redirect the driver to a map that shows both his location and the rider’s location (Figure 5). After that, the driver can either go back to the request list for more options, or accept this request by clicking on the “Accept Request” button. Once the driver accepts the request, Jaguar Ride will redirect the driver to Google Maps to navigate him to find the rider (Figure 6). Once a request is accepted by a driver, it will be removed from the request list. This ensures that no two drivers can pick up the same rider.

Figure 3 Figure 4 Figure 5 Figure 6

**Rider Interface - Waiting for the Ride**

Figure 7 Figure 8

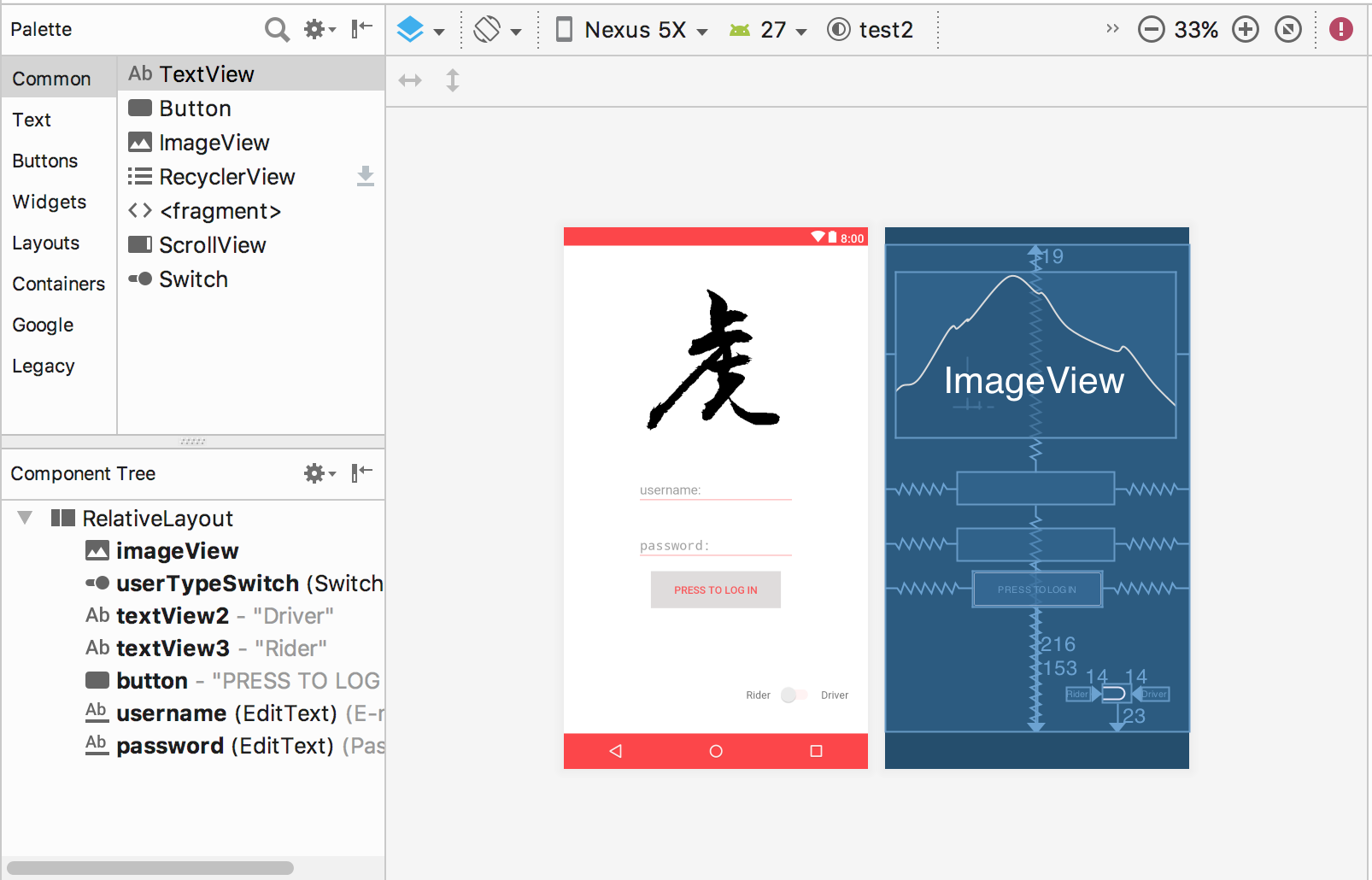
When the rider’s request is accepted by a driver, the rider would be able to see the driver’s location on his phone (Figure 7). At the bottom left corner, Jaguar Ride also shows how far is the driver away. When the driver arrives to the rider, At the bottom left corner, the app will print out “Your Driver Is Here!” to remind the rider (Figure 8). Once the rider and driver meet each other successfully, the goal of Jaguar Ride is achieved.

**Front End Implementations**

The design of front end for this application can mainly be divided into four parts: user login page (home page), rider-view page, driver-view page and Google Map navigation page. Despite the fourth part, which we simply use the well-formed Google Map API front end implementation, the first three parts are designed via Android Studio’s Theme Editor which can be used to create and modify themes, adjust themes for different resource classifiers and visualize the effect of color changes on common UI elements.

**User Login Page**

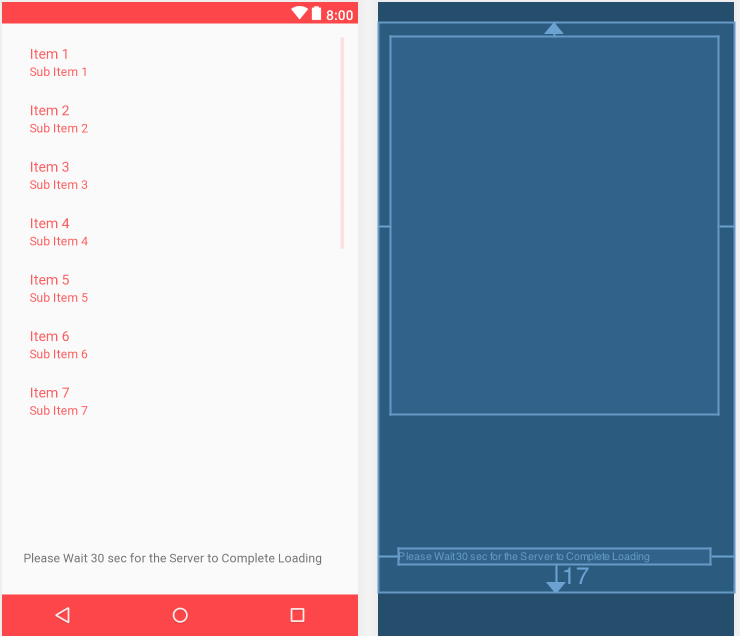
Here we fully utilize the amazing functions that the theme editor provides. The Theme Editor's main screen is divided into two sections. The right side of the editor shows what specific UI elements, such as the app bar or a raised button, look like when the current theme is applied to them. The left side of the editor displays the name of the current theme being previewed, the module where the theme is defined, and the settings for theme resources, such as Theme parent and colorPrimary. Relative codes are written in the text area and test2 is the customized theme our group designed for this application.



For this page, we use an imageview element to represent our application’s logo, which means Jaguar in Chinese letter. Also, we use edittext element to get the username and password mechanism work. Below that, a button is drew for actual login if being clicked on. There is also a switch on the lower-right side which enables the user to choose to become either a rider or a driver.

**Rider and Driver View Page**

In the rider view page, we implements several buttons which allow the user to refresh the page, logout from current account and most importantly, request a ride, and place them on different corners of the page to be user-friendly. At the center of the page is a pre-allocated fragment for Google map activity.



In the driver view page, we use the listview tool to show all the ride request as a list for the driver to view and select. In both page there is a navigation bar at the bottom so that the user can go back to the previous page if anything goes wrong.

**Back End Implementations**

**Parse Server**

We use the Parse Server to develop the backend of this application. Parse Server uses MongoDB as its database. However, we do not need to directly get access to MongoDB. We simply pass the information as a ParseObject and save, the server will do the database storing and querying for us. We deploy the server on AWS Parse Server Certified by Bitnami. It’s free for small traffic.

There are two database schemas we create.

|-- User -> username -> RiderOrDriver ->location

|-- Request -> Ridername -> location -> Drivername

**Database Behaviour**

When the user logs in, he needs to choose whether he is a driver or rider. The database will record his username and usertype(driver or rider). According to different user types, the user will be redirected to different pages.

When the rider presses the Requestbutton, a new Request Object is saved to the database along with the rider’s username and current location.

After the driver logs in, the database will query all the requests that have not been accepted yet and sort all the requests by the distance from the driver from the shortest to the most distant.

If a driver accepts one request, the driver name will be saved to the request schema.

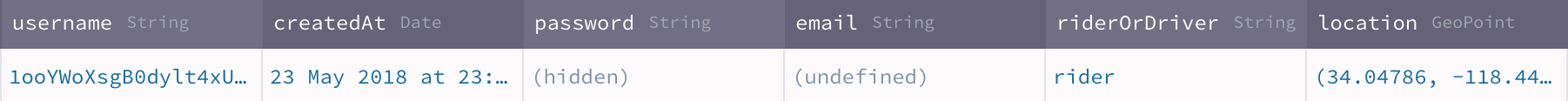
**Data Structures and Implementations**:

In the rider file, we keep two boolean values requestActive and driverActive. If the database finds the request got a driverName, the driverActive will be set to true, and then in the background, the program will compute the distance between the rider and the driver until their distance is less than 0.1 mile. The requestActive is used to judge whether the rider has requested a service. If requestActive is set to true, it means that his request has been stored to the database and wait for the driver to accept; if the requestActive is set to false, it means that maybe the rider cancels his request, and the database has to query all the requests with the rider’s username and delete that request.

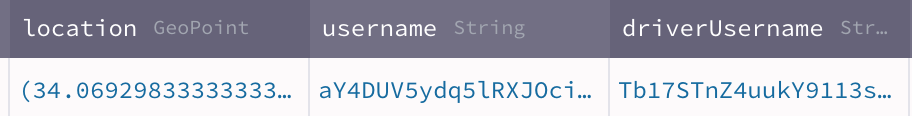
In the navigation file, we create a listener that repeatedly locates the driver and the rider. With those two coordinates, the rider could see his location and the driver’s location in the map while he is waiting.

As mentioned in the “functionality” section, Jaguar Ride currently has three builtin users, whose username-password combinations are stored at local. The builtin users utilize the anonymous login mechanism, where a random username is generated by the server when a user tries to login.

User schema fields:

****

Request schema fields:



**Challenges**

A early but critical challenge we encountered was that Google Maps API didn’t work in our emulator. We double-checked the code, which were modified from the official online documentations and found no errors. And after doing some research on Stackoverflow, we found out that since the Google Maps SDK for Android is distributed as part of the Google Play services SDK, so we need to first download Google Play services SDK via the Android SDK Manager. Another reason for the disfunction was that we didn’t use correct API versions for the emulator. Because the Google Play images are available starting with Android Nougat (API 24), so we need to switch to a newer version of emulator. After we acquired a new emulator image, we update Google Play Services and were finally able to get the map and navigation working well.

**Conclusion**

In conclusion, we implemented the features and functionalities listed in the proposal. The project successfully utilized wireless network connection to implement real time data communication between clients and server and explored the Google Map API to implement real time location changes. Besides some minor issues regarding the handling of exceptions and several areas for future development, the project is a great success and has been a valuable learning experience for every group member.

**Contribution**

Chengyu Wang: Implemented driver functions and set up parse server.

Zhouyang Xue: Implemented rider functions and set up rider-driver communications.

Pochao Yang: Front-end development and UI design. Researched in Google Maps API and GPS location.