Taxi Sharing Application

An Honours Project submitted to
The School of Computer Science in partial fulfillment of the requirements for
Bachelor of Computer Science Honours with
Software Engineering Stream

Carleton University
April 22nd, 2019

Mohamed Gahelrasoul - 101007118

Honours Project Supervisor: Dr. Doron Nussbaum

Abstract

The objectives of this project were to research and develop a taxi ride share application that was capable of matching drivers with multiple passengers and displaying all necessary user information. The driver application would display all relevant passenger info, such as their name, number, pickup location, and drop-off location. This project implements a backend to utilize a real-time, cloud-hosted database to store and sync data between drivers and passengers, acting as the highway between both the passenger side and the driver side. The project goals were focused majorly on the implementation for the functionality of a taxi sharing application, as well as the user interface for a driver.

Acknowledgments

I’d like to acknowledge the services and tools of various organizations, such as Google APIs and Firebase, and Stack Overflow. I would like to thank Dr. Doron Nussbaum for his encouragement and support on this project. A grateful acknowledgement to Andrew Dodge for developing the passenger application and for his continued efforts in this back-and-forth process. A quick thank you to past professors and peers for their help and support leading up to this project. Of course, my gratitude goes to my family for supporting me from the very beginning.

Table of Contents

Contents

List of Figures

Chapter 1: Introduction

Transportation is a major concern in the modern world, where according to Statistics Canada, the average Canadian household paid $12,707[1] on transportation in 2017, and of that they spent an average of $11,433 on private transportation. On top of that, households on average spent $2,142 on gasoline and other resources of fuel. Of the spending that went towards transportation, $1,274 were towards public transportation, which includes taxis, buses, trains, and air fare. Considering the constantly increasing price of taxi and bus fares, it is fair to assume that the amount of money spent has risen since then. For those that are unable to acquire a personal vehicle and are reliant on other forms of transportation, getting around can be costly and inconvenient. Attempting to get around quickly while also keeping comfort and privacy in mind can be difficult at times, as public transportation is not the most efficient method of travel.

In more recent years, taxi service companies have started cropping up and creating a service for regular drivers to act as taxis for regular, all through applications designed to handle ride requests and payment for each ride. This project that has been undertaken is for the venture of creating a taxi sharing application. The purpose of this project is to attempt to create a user-friendly and simple app capable of allowing separate parties to split taxi fare and ride together. This method of transportation utilizes the ease and comfort of ordering a taxi, while also reducing the costs of the trip by sharing the ride, though this may increase the travel time. For many people that prioritize saving their money and living frugally, having the option of sharing a taxi and paying a bit less at the possible expense of their time, then they will gladly save the difference and accept the added travel time required for another passenger.

Problem Definition

This honours project was designed and developed throughout the semester to be a driver designated app, easily accessible and usable. The application is developed and produced on the

[1] Statistics Canada. (2018, December 12). Survey of Household Spending, 2017. Retrieved from https://www150.statcan.gc.ca/n1/daily-quotidien/181212/dq181212a-eng.htm

Android operating system. The early focus of the project was towards research and design of a possible application, while using other similar apps as a reference point. Figuring out the required functionality, accessibility for different devices, the application's layouts, and more, were all taken into consideration for the projects design. The taxi sharing system has three components: Passenger App, Driver App, and back end system. The intent of this project is to focus on the Driver App and general framework for the backend system. The Driver app will be developed on an Android device. The main objectives are: 1. create a driver interface that is easily usable and accessible; 2. Creating a backend and database to store and retrieve passenger information. The driver's side will be developed by myself, while the passenger side will be developed by another student, Andrew Dodge. However, a good portion of the backend will be implemented as much as possible alongside my efforts on the driver's side, in the hopes that both the driver and passenger will be able to utilize the same backend and communicate between each other.

Motivation

As society moves towards more advanced technologies, we look for new possibilities of solutions to our struggles in our everyday lives. Regarding the topic of taxi services, more and more companies are establishing themselves to expand the industry and diversify the market, giving way to new applications that can pave the road further. The motivation behind this project is to create a simple and efficient app to use when in need of a taxi, adding to it the aspect of sharing a ride with other passengers to reduce costs. As autonomous vehicles are being developed at an increasing rate, this project holds the hopes of being used to be expanded as the basis for an autonomous taxi sharing app that will not require a driver. The plan would be to develop a software capable of directing a self-driving car to pick up and drop off passengers at specified locations, without having a driver to oversee the service. Though this technology seems far too advanced, it is a very palpable concept as technology quickly improves. Furthermore, such an application would lessen the number of active cars required to service all the requests made, reducing costs and congestion in major cities. This application would also take into consideration the ride cost for each passenger to reduce a passenger's fee while also netting the same amount, if not more. If there were a way to pay less than the generally charged amount, then it would be worthwhile to the passenger to share the ride and possibly increase their travel time, in order to pay a fraction of the price. As said, while this application would save the user some of the cost, the driver would still be receiving at least the same amount, as if it were one passenger. Keeping this in mind, the project will be made in the perspective of the driver to provide them with all the information required to complete the trip, as well as some backend to bring together the passenger and driver side.

Contributions

To summarize the contributions towards the completion of this project, we implement a system to allow for a taxi ride service. The majority of the system was implemented in order to have taxi sharing possible, with two individual apps to represent the end user and the driver. These two applications are joined by a backend utilizing a database to communicate and exchange information. The driver aspect that is handled by me was developed with a focus on design and ease of usage. To accomplish this, research and time was taken to gather similar apps in the market and determine the successful aspects of their respective systems.

Chapter 2: Background

Overview

This section will be covering the ideas and tools required to understand and develop this project. The chapter will discuss some applications that are available on app stores for use towards the taxi service industry. To describe the process in starting this project and creating the application, review the differences in taxi services and the functionality of their respective applications, and how they influenced the design and played a part in the development of the functionality of the system.

Definitions

As a precursor, the term taxi sharing is an expansion on the structure of taxi services, where a passenger can share a ride with multiple passengers. This addition in service is the main difference between common taxis and shared taxis, wherein passengers requesting a taxi from the same location with destinations approximate to each other are matched to share a ride. Extending that match process, rides that have been started and find nearby requests on the way with a destination close to the original destination will be picked up. So, rather than select a regular taxi that prioritizes privacy, comfort, and rapidness, there is the option to select a shared taxi that prioritizes comfort and reduced cost. This incentive helps users save money on trips, lessen the supply while maintaining the demand, and save drivers time while increasing their profit. Keeping this as the baseline for the project, matching passengers based on destination and displaying the necessary information to complete the ride were the goals for this project.

Previous Work

To give a better visualisation of previously developed taxi services, different application systems will be discussed, looking at the distinguished companies in the market currently as points of reference towards this project's system. Though there are plenty of similar apps out there, the ones that were heavily looked at were Uber and Lyft, arguably the two biggest players. Others to consider are ones such as Via, a shuttle service similar to those mentioned above, and Waymo One, a taxi service initially started by Google using a self-driving car. These differing company applications are topics of interest when researching on how to design and implement a taxi sharing service, using them to gain an understanding of some of the fundamentals in such a service. This can be things such as the information required to be a driver, the layout and views of the pages, or the matching and routing between a driver and the passenger. For this project, the companies Uber and Lyft have services that offer taxi sharing, which will be looked at and described for reference.

The most predominant company with the largest number of users is Uber, boasting 15 million completed rides each day[2]. This company application is one of the most widely used across the world, being available in 65 countries and 600 cities. Clearly, they are doing something right, so using them as a tool will hopefully yield results. Still, Uber is known mostly as just a taxi service, however, they do have something more relating to the project at hand: UberPool. UberPool is a taxi sharing service capable of matching riders together heading in the same direction and splitting the cost and ride[3]. This service will let users to pay a lesser price by potentially picking up and dropping off other passengers along the way, which will likely increase the ride duration. The system matches you with the closest driver, allows you to set the meeting point, it displays the estimated ride time, and calculates the cost of the fare before agreeing to the ride. Somewhere along the route, there may be detours to pick up passengers as requests come in for the closest driver with seating available. Once a passenger's destination is reached, they will be dropped off and the ride will continue, with the possibility of more passengers being picked up and dropped off along the way. This application provides with some possible functionality to implement in moving forward, looking at the tools necessary to create the ride sharing capability, such as closeness of drivers to passengers and passengers' destination proximity to one another.

[2] Iqbal, M. (2019, February 27). Uber Revenue and Usage Statistics (2018). Retrieved from http://www.businessofapps.com/data/uber-statistics/
[3] Uber. (n.d.). UberPool vs. UberX - How Does UberPool Work? Retrieved from https://www.uber.com/en-CA/ride/uberpool/

Another major player in the taxi service industry is Lyft, the less popular but just as competitive company, holding its position as Uber's rival. "As of January 2019, Lyft claimed 29% of the ride-hailing market in the US, compared to Uber's 69%, according to Second Measure." [Iqbal, 2019] More popular within North America, this app is available in 350 US cities and in a handful of cities in Ontario, with their taxi services yet to reach other countries. Lyft has also implemented their own variation of a taxi sharing service, labelling it as Lyft Line and offering its services in the US cities where Lyft is available. Lyft Line is functionally the same as UberPool and other taxi sharing programs, allowing for passengers to share a ride and reduce their costs. However, there is a significant difference between UberPool and Lyft Line, that being the decision to pick up passengers in the middle of a ride. Where Uber will actively search for new passengers as the ride progresses to maximise profits even if it takes you out of your way a bit, whereas Lyft will only match you with others initially and keep the route set to save you time. However, if a passenger is requesting a ride and their travel path would be the same as a ride taking place, then the system will accept that passenger since it won't add any detours, taking minimal additional time. From a systems consideration, both UberPool and Lyft Line have systems that are viable depending on the prioritization of the company and the capabilities of the team designing the system's algorithm. In regard to the interfaces, from a driver's perspective there are no major differences that would impact the usage of either company's respective applications. In essence, here is your passenger, here is the pick up point, the path to the destination(s), and the fare for the ride.

A different idea to the ones put in place by Uber and Lyft companies alike, is the product that Via offers. While it is also a ride share, Via implements a shuttle like system, wherein passengers will be picked up from a certain location and dropped off on a specific point on a path closest to their destination[4]. This shuttle service is similar to that of a ride share, however the pick up and drop off locations are predetermined, making the route constant when travelling. Users will be able to board the shuttle for a small fee and are given the time until pick up and time for destination arrival. Currently it is only available in Chicago, Washington D.C, and New York City, however it still offers an interesting point of consideration despite being on a small scale. Rather

[4] Via Transportation. (2019). Smarter shared rides - Download the app now! Retrieved from https://ridewithvia.com/

than having multiple pick up points as taxi sharing services do, they will minimize the time on detours and pick up by allocating a pick up location for quick boarding and departure. Via guarantees at most a five-minute wait for a shuttle service, keeping the time spent waiting to a minimum. Passengers will develop a routine, knowing exactly when and where they will be riding their shuttle to get to their location. This makes it easier for the driver in the long run, keeping their routing the same and ensuring that there are no issues when finding passengers or trying to drop them off at unfamiliar locations. For a system's optimality, having one pick up location for multiple passengers is ideal, eliminating the need for detours, while also having one drop off location will reduce the trip's duration.

Looking ahead at the possibilities for autonomous ride share services, there is one project that is currently yielding promising developments, Waymo One. Originally developed by a subsidiary of Google in 2009, the Waymo One is now a standalone project that is deploying self-driving rides in Phoenix, Arizona[5]. In its simplest form, this service is a self-driving taxi service the same as any other, though its use of autonomy is still to be extended to other cities and locations. This self-driving service allows passengers to set a pick up location and a destination, at which point they are given an estimated time and a fee for the ride. The user is given basic controls over the ride, with functionality to "Lock", "Start ride", "Pull over", and "Help", while also being given an onboard support system to contact a live Waymo agent at anytime. Though this technology is still being developed, tested, and improved, it paves the way for an autonomous transportation system in the future. As it is, Waymo One only accepts one passenger at a time, but with further developments, there could be someday a self-driving taxi sharing service used around the word. This project has concepts of autonomy in mind but no real aspects of such a system will be put in place, it is simply something to consider when looking at possibilities in the future for taxi services.

[5] Waymo. (2018). Waymo. Retrieved from https://waymo.com/

Software Development

Beginning this endeavour, its important to review and understand the relevant tools to completing the project. Designing the interface for the system from scratch is difficult, so we look towards a prototyping program to help lay out the main pages of the application. Using a prototyping tool called Balsamiq Mockups, we can create low-fidelity mockups of the design of the system. This tool is useful in allowing users to make different lay outs depending on the console they are using and program they are attempting. Developers have the capability to draw their program and add different features and images as they go along to expand upon the design of their application, without needing to write any code. This gives us a frame of reference when entering production of the code that will deal with the looks of the program.

Leading into development of the system, its vital to be working with the proper programming tools to ensure success. This project is fully developed using Android Studio, an integrated development environment (IDE) officially created by Google for its Android operating system and built on JetBrains' IntelliJ IDEA. There are two programming languages available when using Android Studio, either Java or Kotlin. Java is a general-purpose programming language that is object-oriented. Kotlin is a general-purpose programming language made fully operable with java, however because of its type inference – the automatic detection of a data type of an expression – it allows for a more simple and concise syntax. For the purpose of this project, the entirety of it is programmed using Java as a means of convenience for implementing, due to former years of practice coding in said language. When developing through Android Studio, the process involves two different components. The first component is the interface aspect, where the developer designs and populates the pages through a blueprint screen. This screen displays the current layout and blueprint of the application page and allows for immediate additions and deletions by selecting elements and manipulating them. Android Studio will then add the resulting changes to the file as the appropriate source code, known as the extensible markup language, or xml for short. While it is quite easy to use and clear to see the changes, it is just as possible to design the individual screens by writing the specific xml code directly. To add functionality to these screens and display the wanted information, a java file is created alongside its xml file. This java file is where all the programming occurs, referencing the front-end and implementing the back-end to communicate the relevant information between the two. To run and test this software, Android Studio will compile the program in to a Gradle-based system. That compilation can then be taken and created into an Android Package, APK, which is then installed on an Android device and run. Android Studios gives the choice of running the APK on a physical phone running Android, or if they'd like, on a virtual Android device of their choosing. If a physical phone is used, it simply needs to be plugged in to the machine and selected when the program is ran for the APK to be installed and started. This method is best for viewing the application on an actual device that would be used in normal circumstances, while using other functionality of the phone, such as network and GPS. In the other case, where the developer uses a virtual machine, it allows them to test for different types of phones running varying versions of the Android operating system. This virtual phone is built into Android Studio, and requires the operating system version to be downloaded, at which point the phone is saved locally and can be used at any time. In any case, both methods allow for software and interface testing, where actions taken will be noted in the programming environment, and any errors caused will be documented for fixing, or as said in software developing, debugging.

Considering how this program was intended on having two sides, both the driver's side and the passenger's side, and each is being developed separately, there must be a manner of communication between the two. A very useful tool in back-end communication and online database management is the Firebase platform initially developed by Firebase, Inc., which was then acquired by Google in 2014. This platform is a mobile and web application develop, and one of its uses is as a real-time database, a cloud-hosted NoSQL database, which you can store data as JSON object, and manage and sync between users in real-time[6]. Firebase is an essential tool to the success of the communication between the driver and riders, wherein any information required is stored in the database and referenced whenever needed. If a user creates an account, all their information is stored in the database, and for signing in, user authentication is also handled by Firebase, as it manages accounts that are created and checks for matching credentials. While, Firebase acts as a middleman between the driver side and the passenger side, allowing for an

[6] Google. (2019). Firebase Realtime Database | Firebase Realtime Database | Firebase. Retrieved from https://firebase.google.com/docs/database/

exchange of information, the actual matching of driver and rider are handled by the back-end system, which is relayed to the Firebase server, and is then passed on to the awaiting application.

Chapter 3: Main Contributions and Design

The focus of this chapter will be to discuss the plans for this project, going over the targets set out to be reached by the end of development. Discussion on this chapter will be about the initial goals for the system and what functionality was hoped to be brought to life, though that may not be the reality of the completed application. The specifications of the system were not given, so the proceeding section will be an overview of what would be attempted to tackle this project. To expand on the design and functionality of the system in detail, Chapter four will cover all implementations.

Main Objectives

The basis of the project is to develop a shared taxi service for riders through an Android mobile application. This application is meant to bring users together that are requesting a ride from the same location, and pair them with a driver to take them to their respective destinations, and exchange the relevant information needed for all parties involved in this trip. Passengers were to use the app starting from a set location at the Ottawa International Airport, to make pairing easier like the service of the Via shuttle. Ideally, users would be close to each other to reduce the wait times and detours necessary for pickups, so having a common start point is most optimal. Once a ride was to be found, there would be a need for information to be exchanged between both parties, the driver and the rider. This information would only be what is relevant for the interaction to be as smooth and efficient as possible. The driver side information is sent to the server, stored in the database, and then the passenger side is to utilize that data. Likewise, the passenger side is to relay their info to the server, update the database, and provide the needed data to the driver side to ensure a successful operation. Such a scenario would look like the following from the system's perspective:

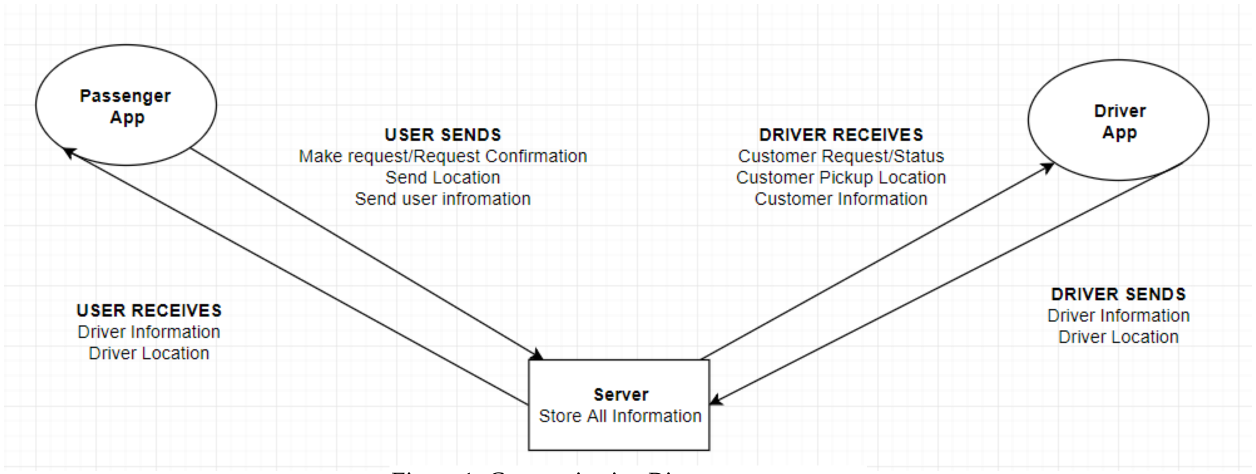
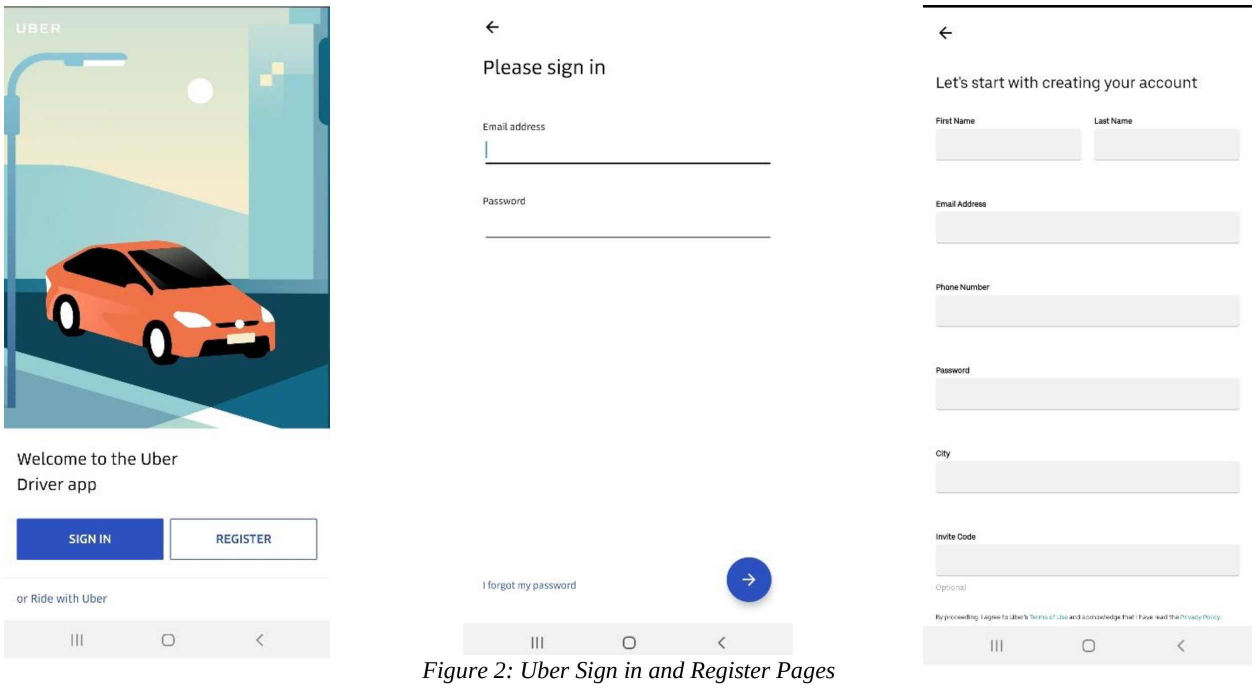


Figure 1: Communication Diagram

Looking at the driver's perspective of the system, the first aspects of the application that were considered for development required an opening screen that showed the application title and gave users the option to sign in or sign up. First time users would create an account and enter all the information required, such as email, password, name, and number to name a few. This information should be stored on a database for easy retrieval when needed, as well as to allow for authentication when returning to the app and signing in. Consider something such as the "Uber Driver" application, which asks the required personal information and gives the most basic of layouts.



By proceeding, I agree to Uber's Terms of Use and acknowledge that I have read the Privacy Policy.

Figure 2: Uber Sign in and Register Pages

This layout also corresponds to the Lyft system, as both Uber and Lyft use a similar design throughout both apps. After which, the driver will be given a main page that consists of a map where most the functionality will be based off. The goal is to have a pop up that displays a ride request to drivers, displaying relevant passenger(s) information, such as destination, name, and number, thus giving them the option to accept or decline a ride. However, looking at it in an autonomous manner, there would be no need for accepting or declining a ride since the system will take any ride that is optimal. Once a ride is to be accepted, the pickup location will be set on the map, with the path and approximate time to get there as well. After the rider has been picked up, the map will update to display the routing to the passenger's destination and the estimated time to reach it. This figure gives a rudimentary idea of an expected exchange of data between the driver and passenger.

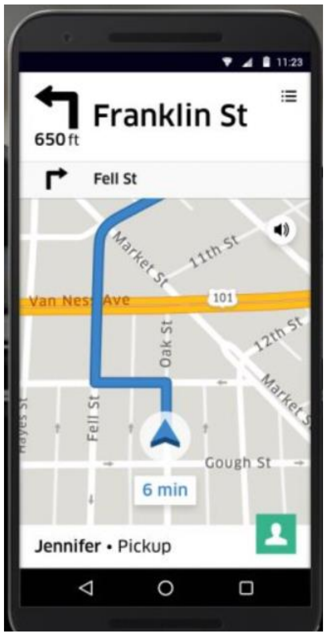


Figure 3: Uber Passenger Pickup Screen

This is a key component to the system, bringing the two parties together. Still, a main concern is "how does the system decide to pair these parties together?". Or more specifically, what are the constraints or parameters that the system has in place to govern the most optimal pairing?

To determine a method of driver-rider matching, the simplest form of optimization is based on the distance between a passenger and the nearest available driver. In a practical world, it would make the most sense to ensure optimality by matching passengers with the closest driver, to reduce wait times and increase the income made when drivers are able to accept more rides. However, in an effort to save riders time and allow drivers to find jobs at a faster rate, Uber has developed a system that avoids the first come first serve methodology, and instead uses a batched matching of multiple passengers and drivers[7]. This batched matching method is intended to compare different possibilities in pairings based on location, street traffic, and other variables to cut down on the time not spent on the ride for both the driver and passenger.

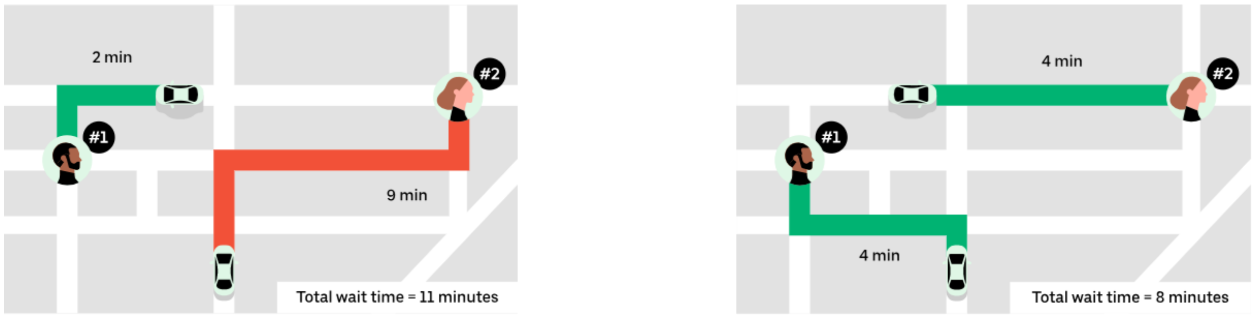


Figure 4: First to request vs Batched matching

Design

A major component to any application is the design of the interface, where a poor user interface can become detrimental to the success of the app. Regardless of the system implementation, consumers will avoid a product if it is difficult to use and not intuitively easy to navigate. With that in mind, it's important to come up with a front-end that is simple and usable. Due to the driver having less functionality than a passenger, there is less content necessary to be placed on the main map screen, making it less of a hassle to keep clean. Looking at previous applications that were mentioned, and considering the layout of Figure 3, there is a need for a design that will produce the required passenger information when needed and avoiding clutter will make it a much better experience for the driver. Being given the rider profile, written steps, and map directions ensure that the driver is going to complete the trip successfully and be able to contact the rider in case of any unforeseen circumstance to coordinately accordingly. Using Balsamiq, it is possible to create a mock-up - such as those in Figure 5 - to replicate features from applications that offer taxi services and use as a template when developing the actual user interface.

[7] Uber. (2019). Matching. Retrieved from https://marketplace.uber.com/matching

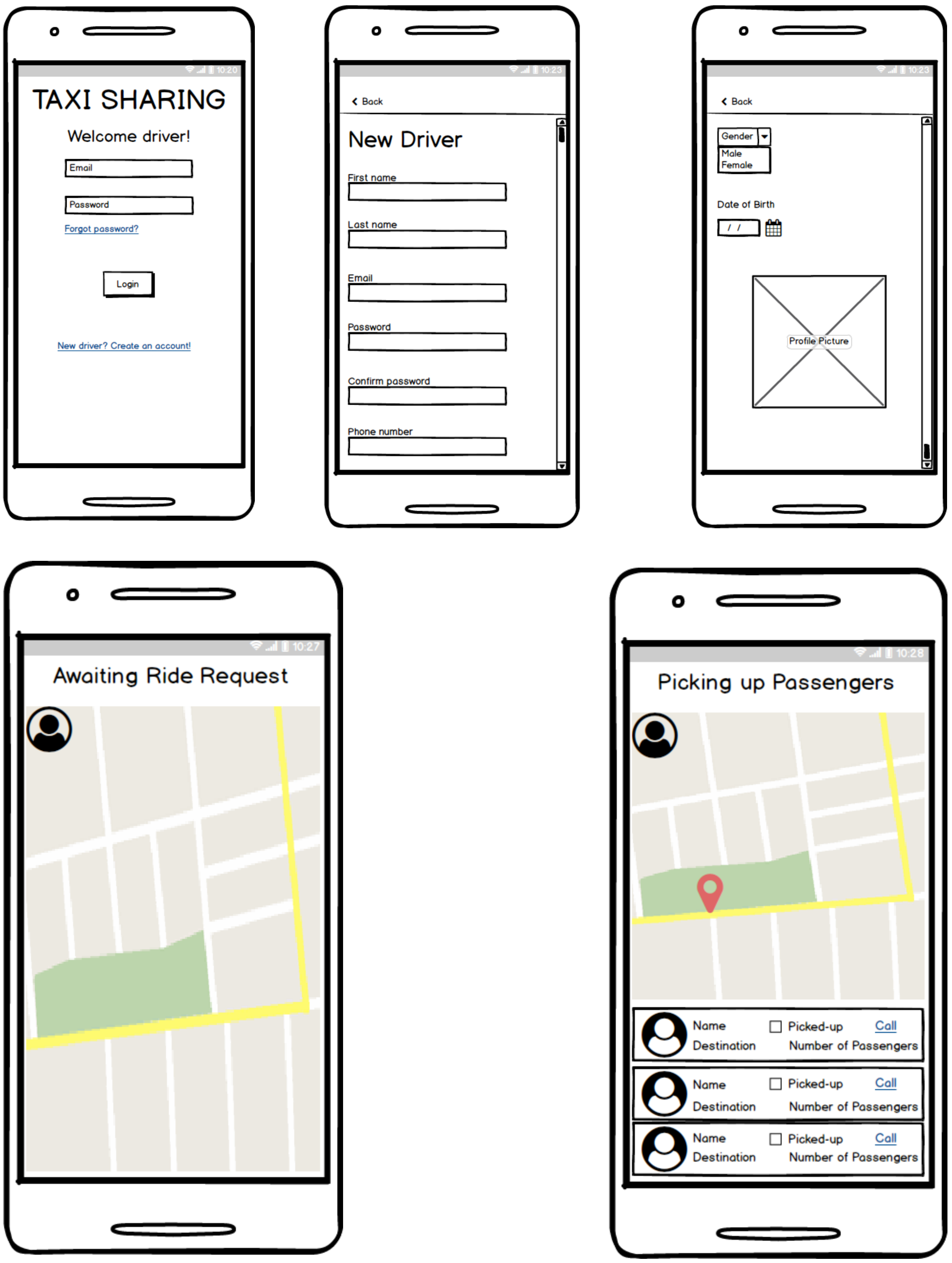


Figure 5: Balsamiq Mock-ups of Driver App

Chapter 4: Implementation and Errors

This chapter covers the overall implementation of the project, discussing the objectives mentioned previously. It will describe the development of the application's functionality from its conception to its final stages, including the setup to the system to allow for implementation and testing. After which, the chapter entails the development process towards completing aspects of the driver interface and back-end system. This is then followed by the testing between both the driver side and the passenger side, using the passenger app developed by Mr. Dodge to check the functionality of the system.

Setting up the Environment

Once the requirements had been established and understood for the taxi sharing system, the required software and tools were researched and organized to begin the project. Developing an Android app with a focus on Google Maps requires several services. All the necessary and relevant information for setting up is listed below.

- Environment
  - Android Studio version 3.3
    - IntelliJ IDEA 2018.2.2 (IDE base for Android Studio)
  - Balsamiq markup (Prototyping tool) version 3.5.16
  - Android version 9.0 (Target Android version)
  - Java version 1.8 (Programming language)
  - Firebase (database) version 16.0.8
    - Authentication version 16.2.0
    - Cloud Storage version 16.0.1
    - Geofire (Location Service Storage) version 2.1.1[8]
    - Realtime Database version 16.0.1
  - Google Play Services
    - Google Maps API version 16.0.0

[8] Firebase. (2018, April 23). Firebase/geofire-java. GitHub repository: https://github.com/firebase/geofire-java

        - Google Location Services version 16.0.0
    - Bumptech Glide (Image Display and Storage) version 4.9.0[9]
    - Windows 10 Operating System
    - Samsung Galaxy S9+ (Phone)
        - Android Version 9

Using these applications and services is out of both necessity and convenience. Services offered such as Google Maps and Locations, Firebase, and Bumptech Glide were used out of necessity to make the system as functional as possible, since key functions require those services. Tools that were used out of convenience were things such as the Samsung phone, the S9+, or Balsamiq markups. These were readily available and easily usable. The phone is personal property, making it a useful testing tool for its dimensions, as well as having the latest Android Operating System. However, this application could have been developed and tested on any Android phone running Android 5.1 or higher as that was the minimum SDK version that was supported by certain features of the project. The latest Android OS version currently is Android 9, the target SDK version, to allow for more modern functionality within the system. Programming language selected was Java due to familiarity and having no prior knowledge on the Kotlin syntax which is also useable in the Android Studio IDE. Java 1.8 was used as it is the latest default version when programming in Java.

To utilize a back-end system capable of storing user information, the best tool available to update user data in real-time was the Firebase database service. Due to it being a Google platform, Firebase has easy integration with Android Studio, giving developers access to services such as authentication, database management, cloud storage, and Geofire. The authentication aspect was used to verify a user's login to the application, comparing the entered credentials to the ones stored when the account was created. At the creation of a new account, the user's information will be added and viewable in the database in real-time, where the email and password will be used for verification. The one element that is not accessible is the password for the account. The cloud storage's use was to store a user's selected profile pictures, which would be displayed to identify them to the passenger. To properly access the image within the storage system, the Bumptech glide

[9] Bump Technologies. (2019, April 02). Bumptech/glide. Github repository: https://github.com/bumptech/glide

Github implementation by Bumptech Technologies was added to fetch and display the saved picture. Simply by adding an implementation line of the repository in the app's build gradle, it is possible to utilize the Glide functionality. The last Google platform mentioned was used to determine and store the geographical location of passengers within the database under the passenger's saved data.

Implementation

When a developer creates a new activity, there are two files that are created. The first is the one seen in Figure 6 below, which is the xml file. This is where all the front-end aspects are created, displayed, and managed. There are many different components to the interface that can be used on the screen when designing the layout, such as input fields, buttons, images, and fragments. On the other hand, the Java file that is created alongside the xml file is where all the back-end is implemented. In order to give the user interface any sort of functionality, the code developed in the connected java file must utilize the elements added to the xml file. Each element is allowed a unique id which is then used in the implementation to give it a specific method that will allow it a certain response depending on the action. By writing these methods for every screen of the app, the behaviour of the system begins to come together as each interactive element is referenced. Based on the prototype's displayed in Figure 5, the design of the application begins with the homepage, the initial screen the user is met with when they open the app. To code this, Android Studio provides a blueprint page as mentioned before, where the developer may add different elements to the screen and control them dynamically, which the IDE then writes the resulting xml code from the selected elements. In the figure below, it is possible to see the constraints and relations the different elements share thanks to the blueprint, as well as the app's actual visual representation when on a device.

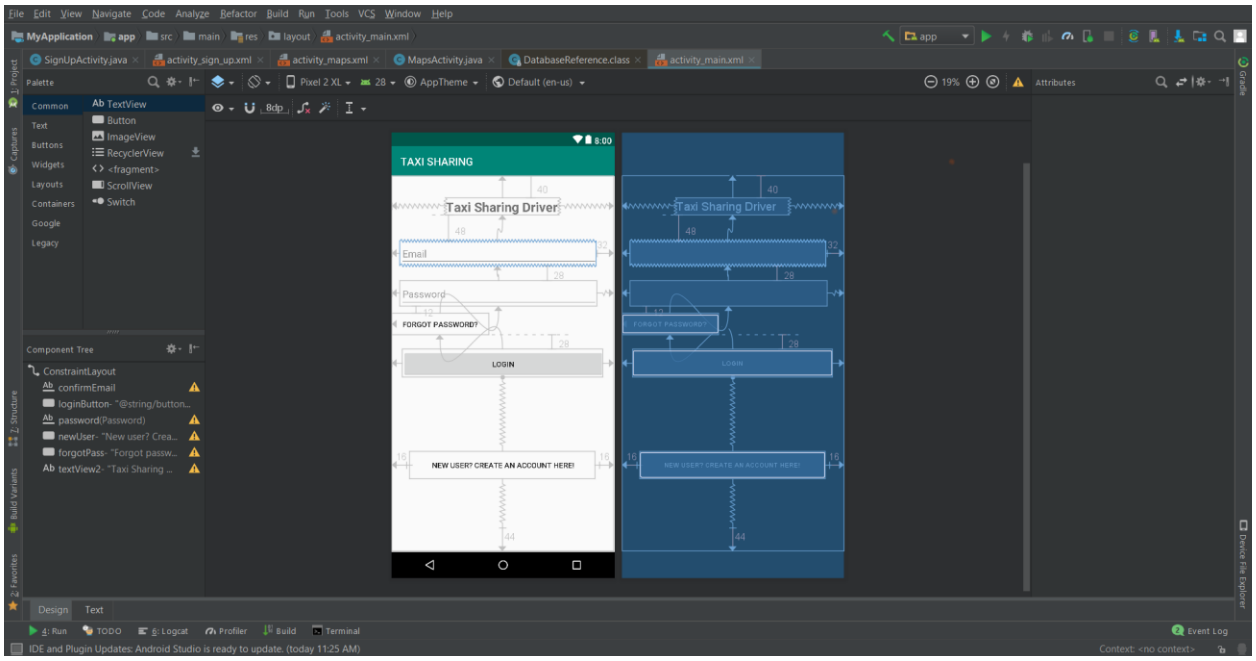
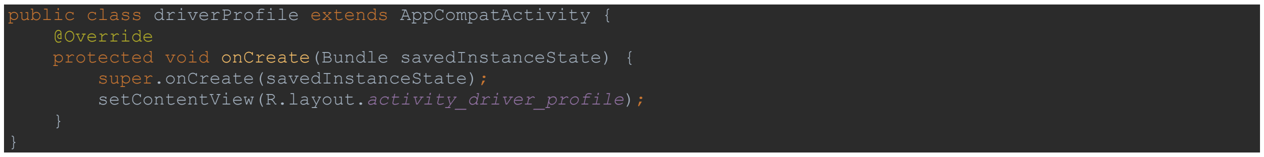


Figure 6: Home screen blueprint, XML file

Once all the designing has been completed for the screen, it is possible to create an instance of the xml page. To dictate what is made when the screen first appears, there exists an onCreate() method that will allow the developer to set the view of the screen by indicating the xml page



```
public class driverProfile extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_driver_profile);
    }
}
```

Figure 7: onCreate method instantiates xml file

Similarly, to change specific elements within the entire view rather than the entire view. Declare a variable of that element type, and set it to the wanted element by using the findViewbyId() method, while also type casting the method return to the element type.

In order to change between screens, the application utilizes elements such as buttons to alternate the view. There exist two implementations that are easily incorporated in order to change the current screen. In either case, a button must be created initially, and though an identification tag is ideal for all elements, the first method of button usage does not require one. In said approach, a method is written within the java file and assigned to the button by adding an onClick(). Once the button is pressed, the onClick tag will activate the method assigned to it. However, the shortcoming of this is that the onClick will only accept a View object as a parameter, meaning pop-ups and other changes happening in the same screen are not applicable. The other approach is used much frequently throughout the application, where it is required that the button have an identification tag. The purpose of this is to create a variable and use the findViewbyId() method as mentioned above. This method takes in an element id as a parameter and makes the implementation using the variable much shorter and easier to follow. Having created said variable, it is now possible to attach listeners to it which allows for much greater element functionality. The button listener is now waiting for an action to occur. Once the button is pressed, a callback method is invoked. This implementation allows for a larger variety of functionality when the user interacts with the screen. This is used in place for something such as the logout button, implemented in the main page of the application which is seen in the screenshots available in the appendix.

Having managed to design the layout and switch between screens, it becomes the goal to allow for user profiles to be created, as well as saved. In order for a user to access the application, they must first have an account created with the required information. First, to create an account, users are directed to a signup screen that has several input text fields, each one with a placeholder prompting them for the correct placement of their information. The implementation was written to check for any empty fields or mismatching emails or passwords, and a small error message informs them of their mistake. As a driver that will be interacting with many different people daily, it is also helpful to add a picture of yourself as well. To store and access all this information, it was convenient and efficient to use Google's Firebase database and cloud storage services. The Firebase functionality was imported and allowed for usage of the database. Once the user confirms all their information and no input fields are left blank, an instance of the database is created. To store all the user data given, a new HashMap is created, and each field is added to it with a key and value as strings. The database instance is then referenced, and the HashMap of information is added to the database under the specified heading. This functionality is presented:

```
Map drivers = new HashMap();
drivers.put("Name", driverName);
drivers.put("Email", driverEmail);
drivers.put("Phone Number", driverPhone);
drivers.put("Car Make", driverCar);
drivers.put("License Plate", driverPlate);
drivers.put("Rating", 0.0f);
DriverDb.setValue(drivers);
```

The only piece of the user's information not shown there is the picture attribute. A picture can be chosen by clicking on the profile image button, which activates the listener. This listener will attempt to open the user's images to allow them to use one as their profile picture. However, to do so, the user must first give permission to the application. Once a picture is chosen though, the database is unable to store the image, where instead it stores a link pointing to the image in the cloud storage. Using Bump Technologies' glide implementation, it is possible to use the URL saved in the database to access the storage and display the image. For initial testing, the fields shown above were made lenient, though the functionality still checks every field. Once everything has been entered, if any error occurs when attempting to create the profile, a toast will appear to notify the user that the signup was not successful. Early testing showed that there seemed to be an issue with the database when creating the account, though looking through the logs showed that there was a problem with the password space. The security measures of Firebase include enforcing at least a 6-character password, which was then made a boundary when users create their password. Once it is possible for users to create a profile and save all their information, or if they had already done so, the main screen would start. Briefly touching on the sign in capabilities, users will enter their email and password to gain access to the application. If this is unsuccessful due to an authentication error, a brief message appears notifying them of the failed login.

To properly implement the maps activity in Android Studio that drivers would use most often, it was necessary to go through the Google Maps API to utilize those services. Creating the activity was easy as it just required the API key to be added to the project manifest. However, getting the map activity to run on any devices was not possible, as the maps page would crash immediately. It was found that after the project was initially created, Android Studio had been updated, along with it its map activities. The app build relied on an older version that had since been updated and was no longer usable. Realizing this difference in versions, a quick change made it possible to load the maps activity. The application opens up to a map, where most of the functionality is found. To properly use the map activity and create a taxi app, the system must be able to work with locations. The application presents the user with a quick permission prompt, asking for usage of its location as seen in Figure 8 below. If agreed to, the app will then be allowed to access the user's location, to which the camera then focuses the map to their current location.

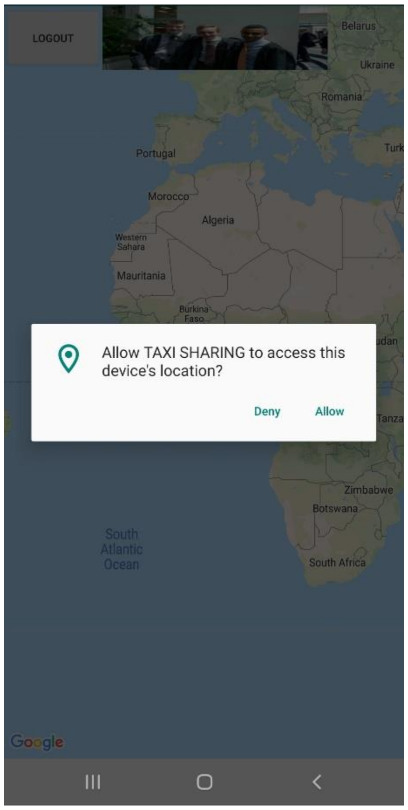


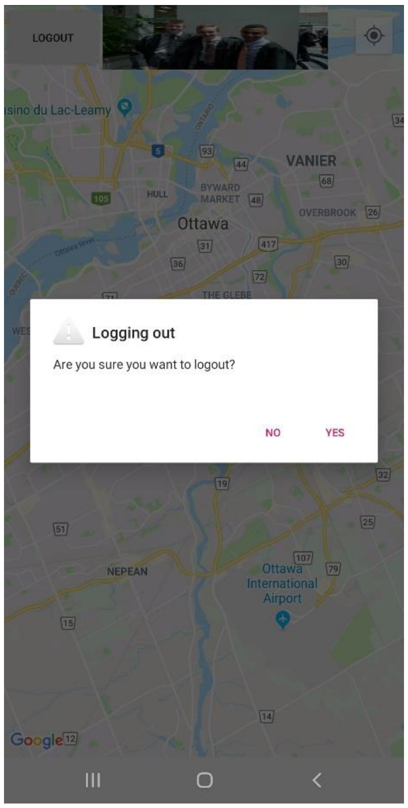
Figure 8: Accessing device location

Following the creation of the maps screen is the creation of the map for both sides, that being the driver and passenger's map page interface. So, to create any kind of working system, the back-end implementation must bring the driver and rider together. It was possible to set a driver as available for a ride in the database by creating an attribute that stated those free for service. On the other side, being implemented by Mr. Dodge, is the passenger's ability to search for a ride. To do this, it was necessary that a passenger searches for ride, and in doing so, adding an attribute to their database data that stated they were searching for a ride. Following that, the system searches for the driver closest to the customer, by taking a 1km radius of the passenger and increasing that whenever a driver is not found, alike to Lyft Line[10]. If multiple drivers are within that range, then the one closest will be given the ride. Having found a driver, the system then assigns that driver by having the passenger side place their customer id within the driver's "customerRideId" attribute, allowing them to identify their soon-to-be rider. Having been assigned a job and a reference to the rider, it is possible for the driver to obtain the relevant information needed about their passenger. By using that customer id, the data deemed necessary can then be provided to the driver once the match has been found. When the driver has gotten the rider information, a map marker is placed indicating the passenger's position and destination, and vice-versa, the driver's current position is viewable by the rider. This is given by the latitude and longitude found in the customer's database data. To display information of the passenger, a cardView is added to the maps activity xml. This card activity allows for different elements to be displayed cleanly. TextViews of user info and an image they provided are added to the xml page, which is then populated using the customer id given to the driver. The information the driver would find useful were name, phone number, and destination, as well the cost of the ride. A useful tool of course would be to display the pathing to the rider from the driver's location. Unfortunately,

[10] Lyft Engineering. (2016, February 02). Matchmaking in Lyft Line. Retrieved from https://eng.lyft.com/matchmaking-in-lyft-line-9c2635fe62c4

due to a fee of use by Google’s Directions API[11], it was decided that directions would be implemented using Google’s Directions API tutorial, though the actual functionality of it is unknown. This implementation can be found in the getRouteToMarker() method, where the key attribute would need to be replaced with a purchased key.

At this point it was considered to develop a better interface for the driver since the main implementation of a taxi service was complete. One thing which was added was the driver profile picture located at the top of their maps screen, which acted as a button to access their information and edit it if desired. A component that was also common and practical, a logout button was introduced to ensure that drivers could exit out if they so preferred, which also removed the driver from the database segment displaying them as available for work. To do this, a listener was attached to the button and on clicking it, it would open a prompt asking the user if they would like to indeed logout. Pressing no would simply make the prompt disappear, but on pressing yes, the driver would be taken to the sign in screen. This agreement would call disconnect(), a method used to stop the constant calls for driver location, and to remove the drive from the “driverAvailable” segment of the database. Once the driver was disconnected, Firebase’s authentication server allowed for easy signing out by simply calling the Firebase instance’s signOut() method. This will bring the application back to the sign in page.



Logging out

Are you sure you want to logout?

Figure 9: Logout prompt check

The progress though slow, testing proved the application to be capable and fulfilling of a taxi app. Unfortunately, there still remained the task of making it a taxi share service app, as opposed to simply a taxi service app. While implementing the driver-passenger system and creating pairings, it was attempted to incorporate multiple passengers towards one ride. Similar to how the pairing was made through the database, passengers would search and attempt to find a rider. It was thought that those

[11] Google. (2019). Directions API. Retrieved from https://developers.google.com/maps/documentation/directions/start

that found the same rider would be added to a list of passengers for each driver. Instead we found that it was not possible to store said list in the database as something that could be iterated through, and that only the first passenger would be saved to the database segment made. While the code was implemented to accept more than one passenger, it was not conceivable with the lack of time and knowledge to find a strategy that would match more than one rider with one driver.

Chapter 5: Conclusion

The first half of the project entailed research into the possibilities available for this program, based on similar applications out there. Initially considering the design of the app, a priority was placed on smaller objectives that were consistent with any app. These were things such as the layout, the user information, and the overall look. Users were able to navigate the pages easily and the usability was informative throughout. Still, it can be said that too much time was spent on less than vital features, such as creating an account with text input checks, or a profile image. The maps screen requires a better layout to make more use of the spacing, while also making sure to keep minimal clutter. Of course, while developing an application system, the frontend becomes almost obsolete when the project's future applicability would involve a driverless taxi service.

However, realizing that to achieve a favourable app, there still needed to be a significant amount of backend functionality in place. The objectives with which the project had started off can be said to have almost been met. Attempting the project required research into the design and layout of user interfaces, and more so for the implementation of the backend to allow for communication between both Mr. Dodge's passenger side, and the driver's side. It was important to put in time towards finding the proper tools and services to make the system as capable as possible. Visualising and creating a simple app to allow a driver's success as a driver, while implementing the backend communication, proved to be challenging when working with unfamiliar software and design. The implemented functionality of the system

References

Google. (2019). Firebase Realtime Database | Firebase Realtime Database | Firebase. Retrieved from https://firebase.google.com/docs/database/

Google. (2019). Directions API. Retrieved from https://developers.google.com/maps/documentation/directions/start

Iqbal, M. (2019, February 27). Uber Revenue and Usage Statistics (2018). Retrieved from http://www.businessofapps.com/data/uber-statistics/

Iqbal, M. (2019, March 25). Lyft Revenue and Usage Statistics (2019). Retrieved from http://www.businessofapps.com/data/lyft-statistics/

Lyft Engineering. (2016, February 02). Matchmaking in Lyft Line. Retrieved from https://eng.lyft.com/matchmaking-in-lyft-line-9c2635fe62c4

Statistics Canada. (2018, December 12). Survey of Household Spending, 2017. Retrieved from https://www150.statcan.gc.ca/n1/daily-quotidien/181212/dq181212a-eng.htm

Uber. (n.d.). UberPool vs. UberX - How Does UberPool Work? Retrieved from https://www.uber.com/en-CA/ride/uberpool/

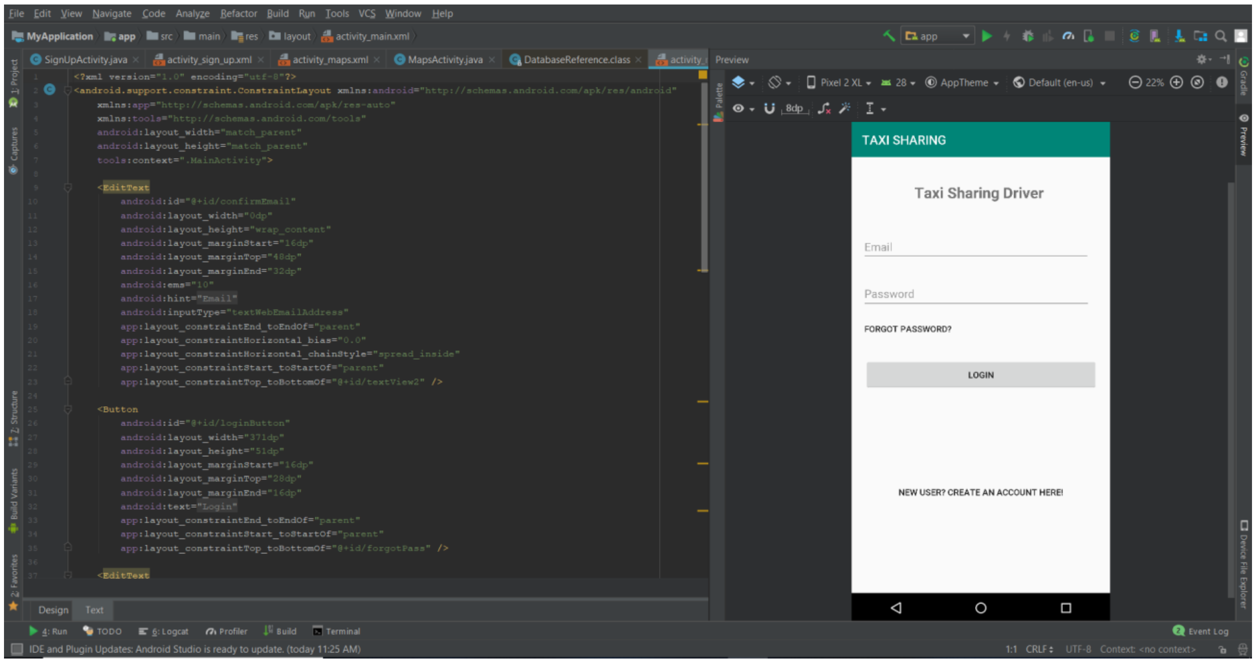
Uber. (2017, March 15). A New Navigation Experience for Drivers | Uber Newsroom Australia. Retrieved from https://www.uber.com/en-AU/newsroom/a-new-navigation-experience-for-drivers/

Uber. (2019). Matching. Retrieved from https://marketplace.uber.com/matching

Via Transportation. (2019). Smarter shared rides - Download the app now! Retrieved from https://ridewithvia.com/

Waymo. (2018). Waymo. Retrieved from https://waymo.com/

Appendix



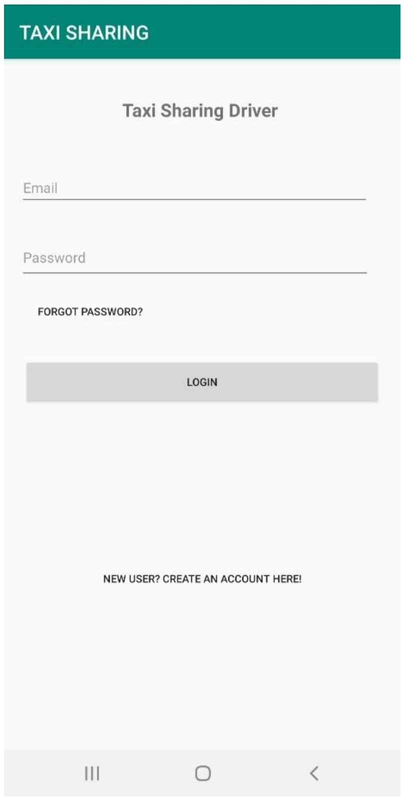
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/confirmEmail"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="48dp"
        android:layout_marginEnd="32dp"
        android:ems="10"
        android:hint="Email"
        android:inputType="textWebEmailAddress"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintHorizontal_chainStyle="spread_inside"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2" />

    <Button
        android:id="@+id/loginButton"
        android:layout_width="371dp"
        android:layout_height="51dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="28dp"
        android:layout_marginEnd="16dp"
        android:text="Login"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/forgotPass" />

    <EditText
```

IDE and Plugin Updates: Android Studio is ready to update. (today 11:25 AM)


TAXI SHARING

Taxi Sharing Driver

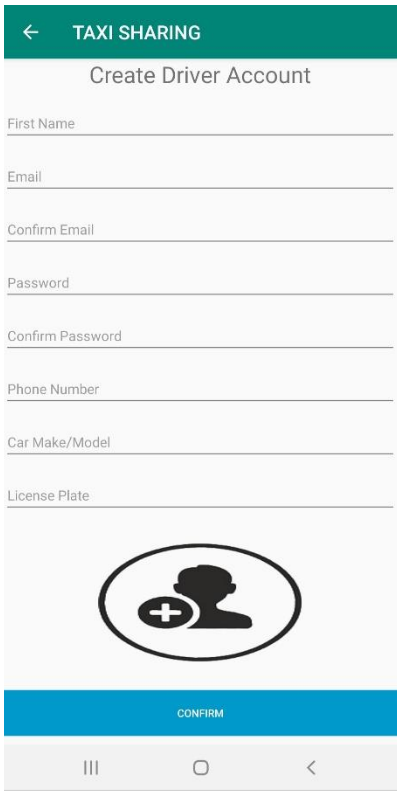
Email

Password

FORGOT PASSWORD?

LOGIN

NEW USER? CREATE AN ACCOUNT HERE!


TAXI SHARING

Create Driver Account

First Name

Email

Confirm Email

Password

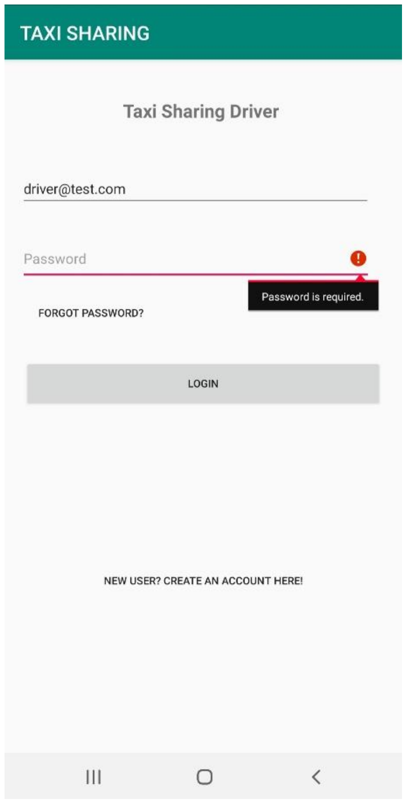
Confirm Password

Phone Number

Car Make/Model

License Plate

CONFIRM


TAXI SHARING

Taxi Sharing Driver

driver@test.com

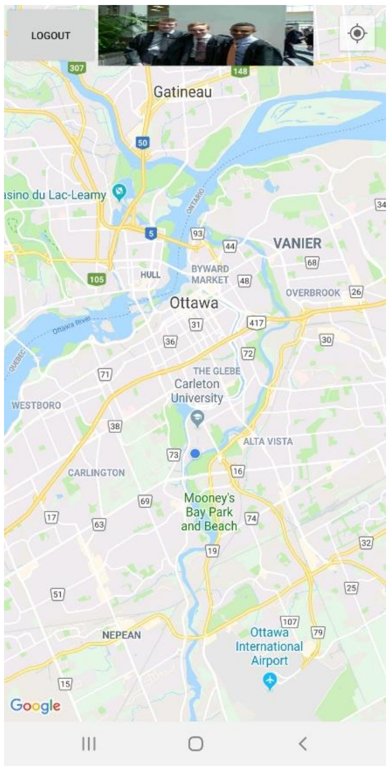
Password

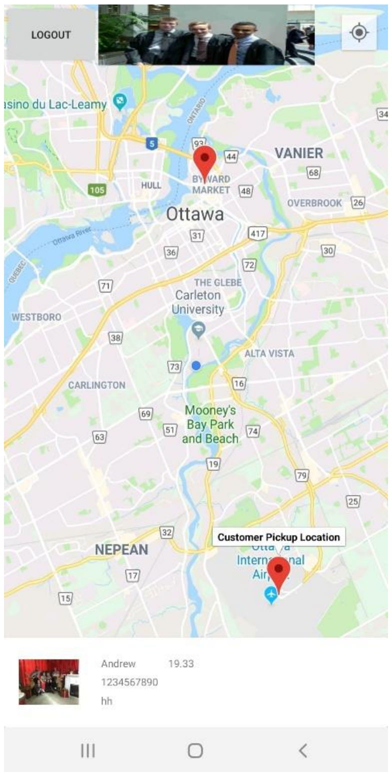
Password is required.

FORGOT PASSWORD?

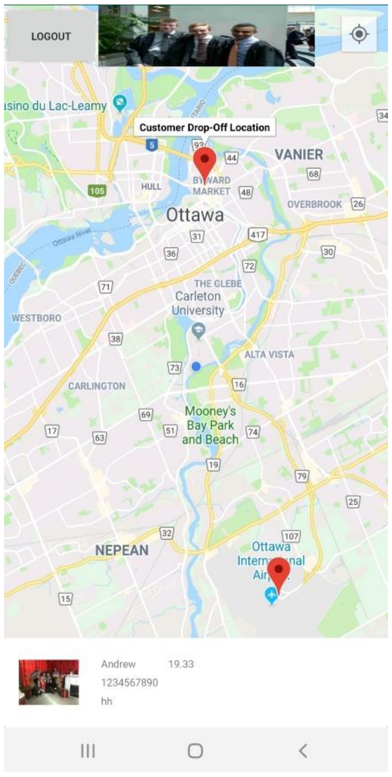
LOGIN

NEW USER? CREATE AN ACCOUNT HERE!





Andrew 19.33
1234567890
hh



Andrew 19.33
1234567890
hh