UNIVERSITY OF Cincinnati

Department of Electrical Engineering and Computer Science (EECS)

RollOut (CE27)

Kristin Van Deusen (CS)
Chad McDaniel (CompE)
Connor Silva (CompE)
Advisor: Dr. John Franco

April 26, 2019

Submitted in partial fulfillment of the degree of
Bachelor of Science in Computer Engineering and Computer Science

Team Member: Kristin Van Deusen _____ Date: 4/26/2019
Team Member: Chad McDaniel _____ Date: 4/26/2019
Team Member: Connor Silva _____ Date: 4/26/2019

Technical Advisor: Dr. John Franco _____ Date:_____

ACKNOWLEDGEMENTS

Teresa Hamad

- For being the best advisor, any could ever ask for.

All of our friends and family

- For supporting us along the way, as well as for participating in the usability tests for this project.

TABLE OF CONTENTS

Abstract

Ride sharing is not a new concept, given that taxis have been around for over a century, but ride sharing mobile applications have transformed how people can get from one place to another. Applications like Uber and Lyft have allowed people to easily hail a ride with a few movements of their fingers. Payment is already taken care of and if a user does not want to talk to anyone, they do not have to. The reason these applications have dominated the current market and are starting to put cab companies out of business (in certain areas), is due to their extreme ease of use.

Executive Summary

The reason ride sharing mobile application have dominated the current market is due to their ease and speed of use. The main idea that this project is based on is being able to even further reduce this simplicity while at the same time guaranteeing users they have picked the best option for either their budget or their time. This report then covers the various technical details about the design of the project. It is developed for the Android platform through the use of Android Studio. The application uses Google's Firebase for the storage of its data. Many challenges came up during the actual coding of this project but were largely overcome and the schedule created at the beginning of the project was maintained. Some future developments for the project are Facebook integration, group messaging, further expansion into the ride sharing market (e.g. Bird/Lime Scooters, Redbike, public transportation). In the concluding section, there is a graph showing the time saving capability of the RollOut application versus the current method of comparison shopping. In general, a user's time spend one-third of the time looking for a ride when using RollOut.

INTRODUCTION

The purpose of this report is to provide both a general overview of our senior design project as well as the extensive technical detail that went into the design, planning, and execution of the project. This project is a mobile phone application developed on the Android Platform that serves as a "one-stop shop" for people who desire a method of transportation. The app uses various ride-sharing APIs (i.e. Uber, Lyft, and Google Maps) and an algorithm to estimate cab fares, and then displays them all on one screen. It also allows users add other users as friends and then enables them to share their locations with each other to also be displayed on a map.

Problem/Need

This project is needed in order to simplify the process of securing a ride, and to ensure that the user is informed enough to make their best decision. Another, more specific, need for this simplicity is because drunk driving is still a huge problem facing the US. "In 2016, 10,497 people died in alcohol-impaired driving crashes", according to the CDC[1]. Ride sharing apps like Uber and Lyft have tremendously helped simplify the process of getting home safely, thus keeping more people off the road who shouldn't be driving. These apps have given consumers never-before-seen amounts of choice when they need a ride. While all these choices are great for consumers, having a one-stop-shop to make the best choice would both benefit the user and their wallet.

Solution

The idea outlined in this report is for one app that allows users to see all of their options yet will also be simple enough that with one touch someone can order a ride to their location. This app will pull in data from ride sharing apps like Uber and Lyft. It will estimate local cab company fares. Finally, it will also allow users to add their friends so that they can see their location on a live map so that they can meet up to share a cab or another service. Friends will be able to post that they are available to be a designated driver. Hopefully, with this added simplicity and the confidence that users are getting the best option, even more people will be encouraged to make the smart decision and not get behind the wheel.

[1] "Impaired Driving: Get the Facts | Motor Vehicle Safety | CDC Injury ...." 16 Jun. 2017, https://www.cdc.gov/motorvehiclesafety/impaired_driving/impaired-drv_factsheet.html. Accessed 5 Oct. 2018.

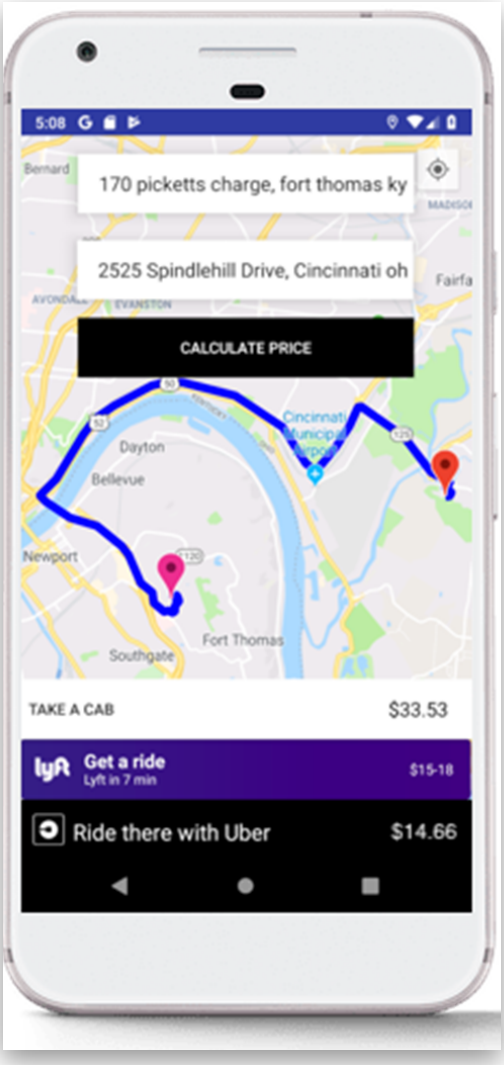

Figure 1: Screenshot of RollOut in Use

Credibility

Kristin Van Deusen (Computer Science)

Qualifications: Worked in a variety of fields including artificial intelligence, natural language processing, data analysis, web development and statistical analysis. Fluent in C#, Java, MATLAB, SQL, Javascript/Node.js, PHP, Swift, Objective C, SAP, Cucumber and Selenium. Proficient in C++ and Python.

Chad McDaniel (Computer Engineering)

Qualifications: Worked extensively in statistical analysis methods of big data and GUI development. Fluent in Python, MATLAB, and C++/C.

Connor Silva (Computer Engineering)

Qualifications: Worked in Software QA. Designed and performed tests on software/firmware. Proficient in MATLAB, C++.

The project team is composed of two computer engineering majors and one computer science major. All three have worked on apps and user interfaces before.

Kristin Van Deusen has worked in a variety of fields including artificial intelligence, natural language processing, data analysis, web development and statistical analysis. These skills are the real backbone of the app. She has experience in C# which will allow her to use Java effectively in Android Studio. Additionally, she has also developed iOS apps which will be vital in the development of this app.

Chad McDaniel has a specialty in data analysis and GUI development. He will be able to apply these data analysis methods when programming the background information processes, to better tailor the app experience to the users. He will also be able to use the GUI development skills learned in the design of the app.

Connor Silva has a background in Software QA and has designed (and performed) tests on software/firmware. These skills will be needed tremendously not only when the app is finished to test its quality, but also during its development to ensure that our code meets a certain quality of modularization and uniformity.

Project Goals/Brief Methodology

The main overall goal of this project to display every type of ride sharing option on one screen to enable users to pick the best option, with the main idea being centered around the simplification of finding a ride. Listed below are the objectives of the project from the Attribute List for the project which expands on the goals of this application.

Objectives

- Useful/functional
    - The application should provide a service of convenience to a user and must be fully operational.
- Safe/Secure
    - The application should keep users' information private and provide them with a direct link to Uber or Lyft where driver's are screened and monitored to ensure safety.

- Marketable
    - The application should be appealing to users to make sure an ecosystem is built for the users as well as have a easy to use UI.

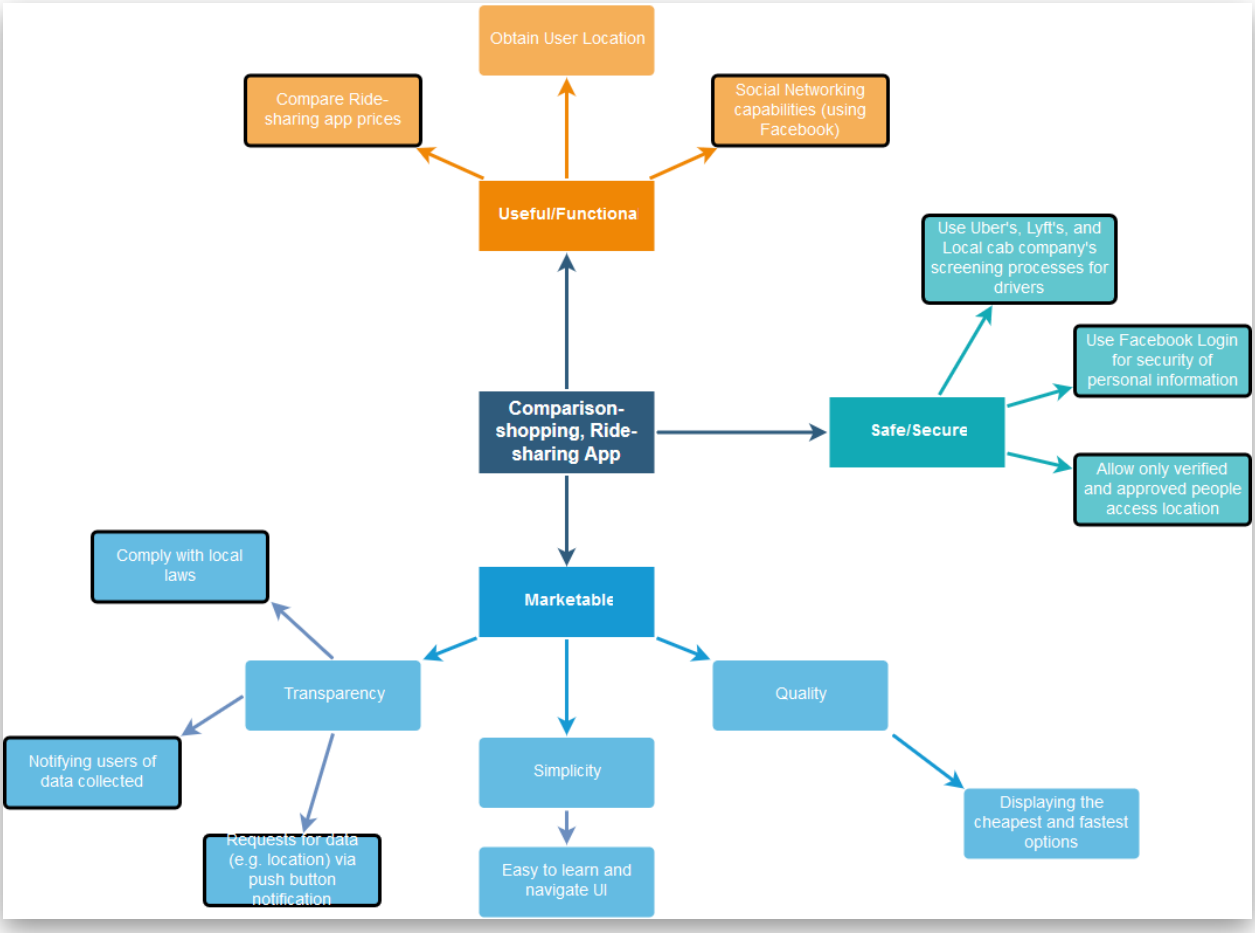

Figure 2: Objective Tree

DISCUSSION

Project Concept

The entirety of the design of the application is centered around the idea of speed and simplicity, while keeping a mindful eye on security. The goal of this concept was to successfully display available rideshare options to a user on one modular screen while also displaying a users' friends' locations on their map. By doing so, a user is able to quickly make a decision that best fits their needs (whether it be Uber, Lyft, a cab or carpooling or sharing a ride with a friend) all on one screen.

Design Objectives

A brief list of the primary design considerations is listed below. These considerations are based off of the three main objectives of the application listed above. Each function and constraint listed is directly related to a main object. This can be seen in the Object Tree figure above.

Functions:

- Request/obtain user's location
    - A push-button notification is to be displayed to the user, alerting them that this app will need to access their current location in order to be fully functional.
- Calculate price for Uber and Lyft based on location
    - The app will request this information from the ride sharing apps APIs and then will display it to the user.
- Show nearby friends
    - Users will be able to add friends to their profile and, with the proper permissions turned on, will be able to see each other's locations on the map.
- Save user location
    - The user's last location will be saved in Firebase. It will be used in order to determine a starting location for ride hailing and to display to approved friends.

Constraints:

- Request user's approval before collecting information (access to location, camera, etc.) via push button notification
- Comply with/utilize Uber API to Open Uber With Saved Location
- Comply with/utilize Lyft API to Open Lyft With Saved Location
- Comply with/utilize Google Maps API
- Comply with local laws

Methodology/Technical Approach

Upon downloading the mobile application, the user will be prompted to create an account. This account is stored within Firebase Firestore, which is a cloud database that easily integrates with Android Studio. The user will be prompted for various fields including their first and last name, their phone number, email, username and password. The combination of these fields will make it easy for account retrieval if one field is forgotten by a user. The user will also have the option to edit their profile and add an avatar image of their choosing. In the future, a user will be given the option to sign up for an account using their Facebook login. When opting for a Facebook account, the user's picture, name and friend list will be gathered to show them catered results. Once their account is created, the user will be prompted to login. The log screen will ask a user for a combination of either their phone number, username or email in addition to their password. Firebase Firestore will check these credentials to make sure they are valid in the cloud database and either grant access to the landing page or ask a user to register. Using Android's internal permission requesting, the user will then be prompted to allow the application to gather their current location. This location will be stored in Firebase Firestore for later use. The landing page will provide the user with an input field titled 'Where To?' this is where the user will enter their desired location. Once a user fills in this field, an input field will appear displaying their current location. The user will have the option to either use their current location or edit this value to be someplace else. Once the user uses the 'Calculate Price' button, RollOut utilizes Google's Geocoding library to convert the user's textual input into coordinates then feeds them into Google's Map SDK API. This API determines the route information between the two coordinates, including distance in miles as well as duration of the trip in current traffic by car. This information is fed into our custom cab algorithm obtained from Yellow Cab Cincinnati. The algorithm uses the distance and duration to calculate an accurate cab fare and presenting

it to the user as an option. Uber and Lyft's APIs work similarly. Each require a development client token to be connected to RollOut. Once the connection is created, each API requires ride parameters in order to calculate an estimated fare. These ride parameters include pickup location, drop-off location and type of ride (UberXL, UberX, Lyft XL, etc.). RollOut automatically chooses the cheapest option for the user. The pickup and drop-off location are obtained from the text input fields on the landing page. Again, these textual fields are Geocoded and turned into coordinates in order for Uber and Lyft to recognize them as locations. These APIs then provide a cost range for the ride as well as how far an available driver is. All of these services (Uber, Lyft and Yellow Cab Cincinnati) are provided to the user in the appearance of buttons. The user will then have the option to select with service they desire and click the associated button. When clicking Uber or Lyft, the user will be redirected to their Uber or Lyft Android mobile application. Both Uber and Lyft (whichever was selected by the user) will then open with the user's location preferences filled out as a result of each services' API. In addition to displaying the available rideshare options, their prices and distance away to the user; the user will also see their available friends' locations on their map. After creating and registering for an account, the user will have the option to go back and edit any information they desire in the settings tab - it's here where a user can add their friends to their friends list by phone number, email or username. When adding a friend, the desired person will get a notification say they've been requested as a friend and have the option to either accept or ignore the request ensuring the highest standard of security. If a user decides to accept a friend request, their location will then be shared with the person in which had requested it. The new friend's location will then be fed into their map. The way this works is by utilizing Firebase Firestore once again. Each user's current location is live fed into the cloud database every time RollOut is opened. Each user also has a friend list in the database in which reroutes to the friend's profile and their current location. In future implementations, the current user will be able to click on their friends and directly message them (internally) and see what their availability is or if they're interested in requesting a ride together. By incorporating Facebook as an element, a user will also have the ability to automatically import their friend list and request to see their friends' (who have RollOut) location. In addition, RollOut plans to incorporate local events on the user's map to promote local businesses and increase community engagement.

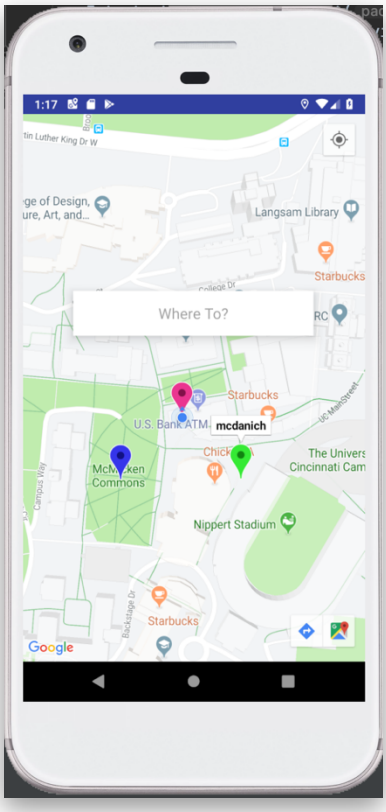

Figure 3: User's Friends Displayed on Map

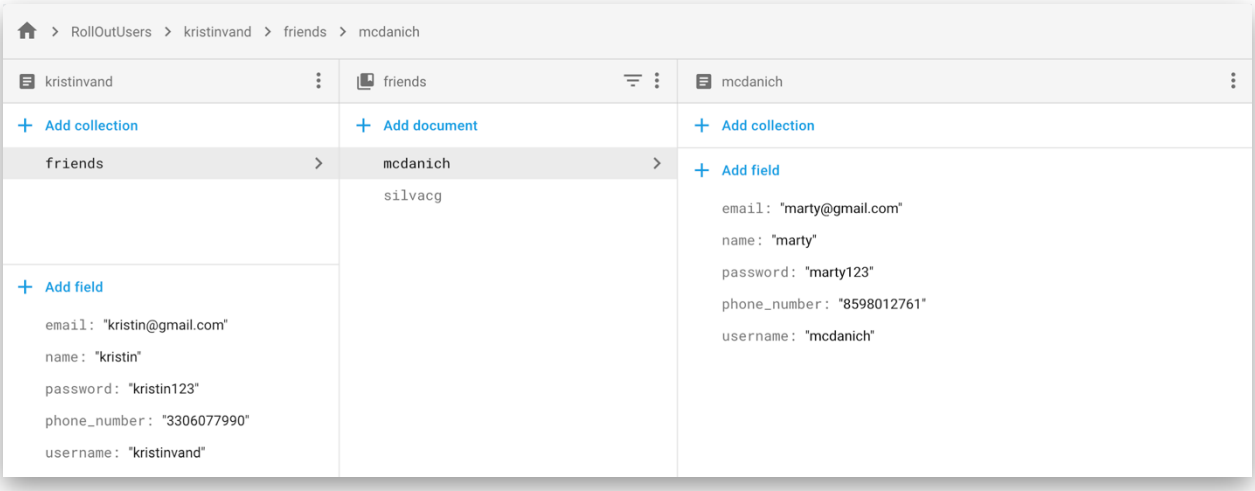

Figure 4: User's Friend List

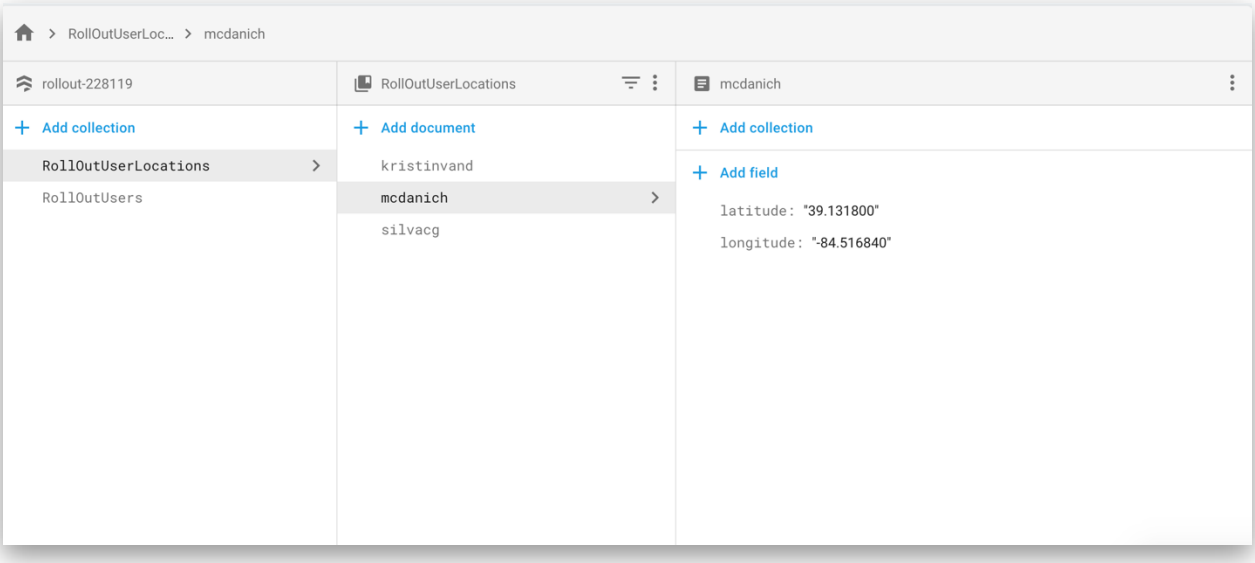

Figure 5: User's Stored Location

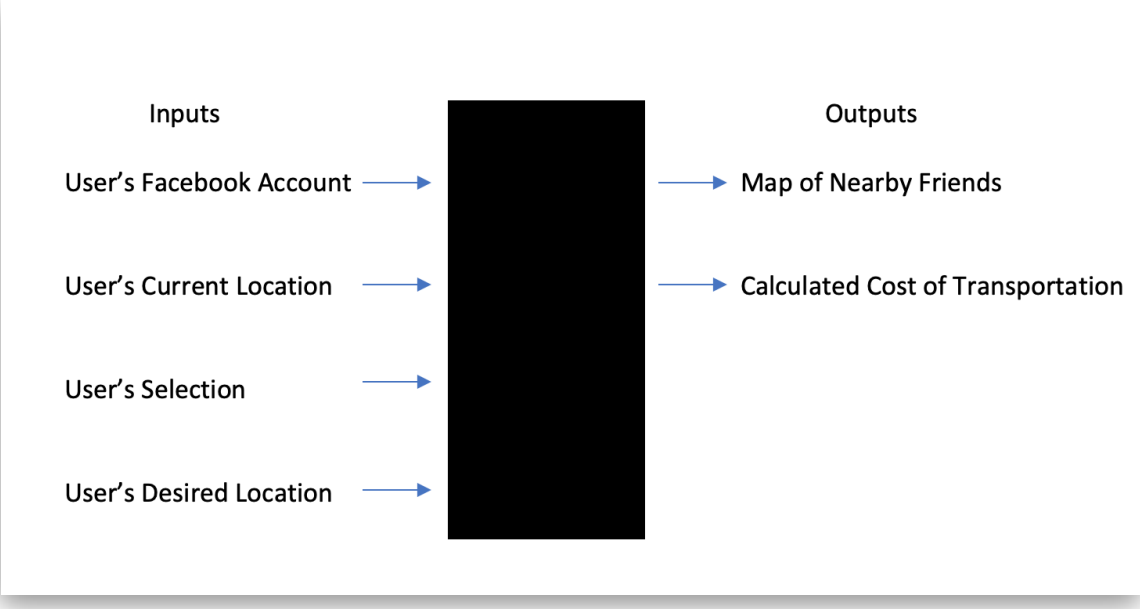

Figure 6: Black Box Diagram

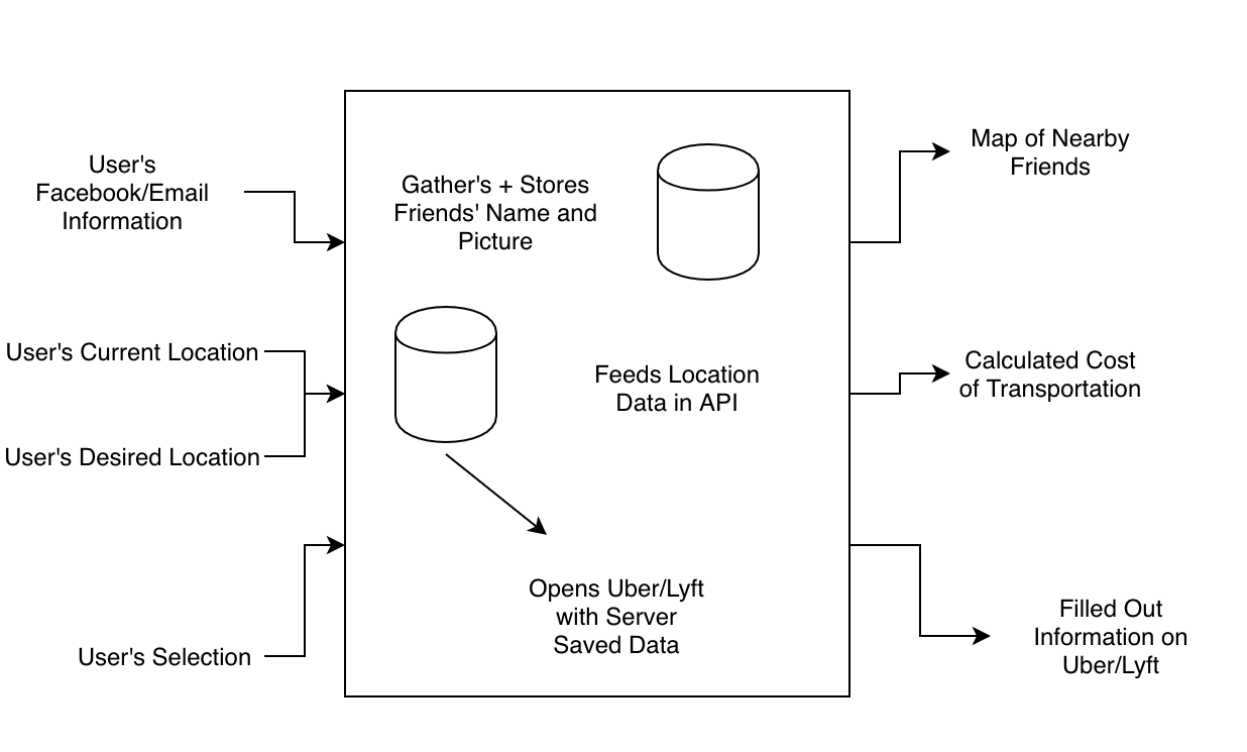

Figure 7: Glass Box Diagram

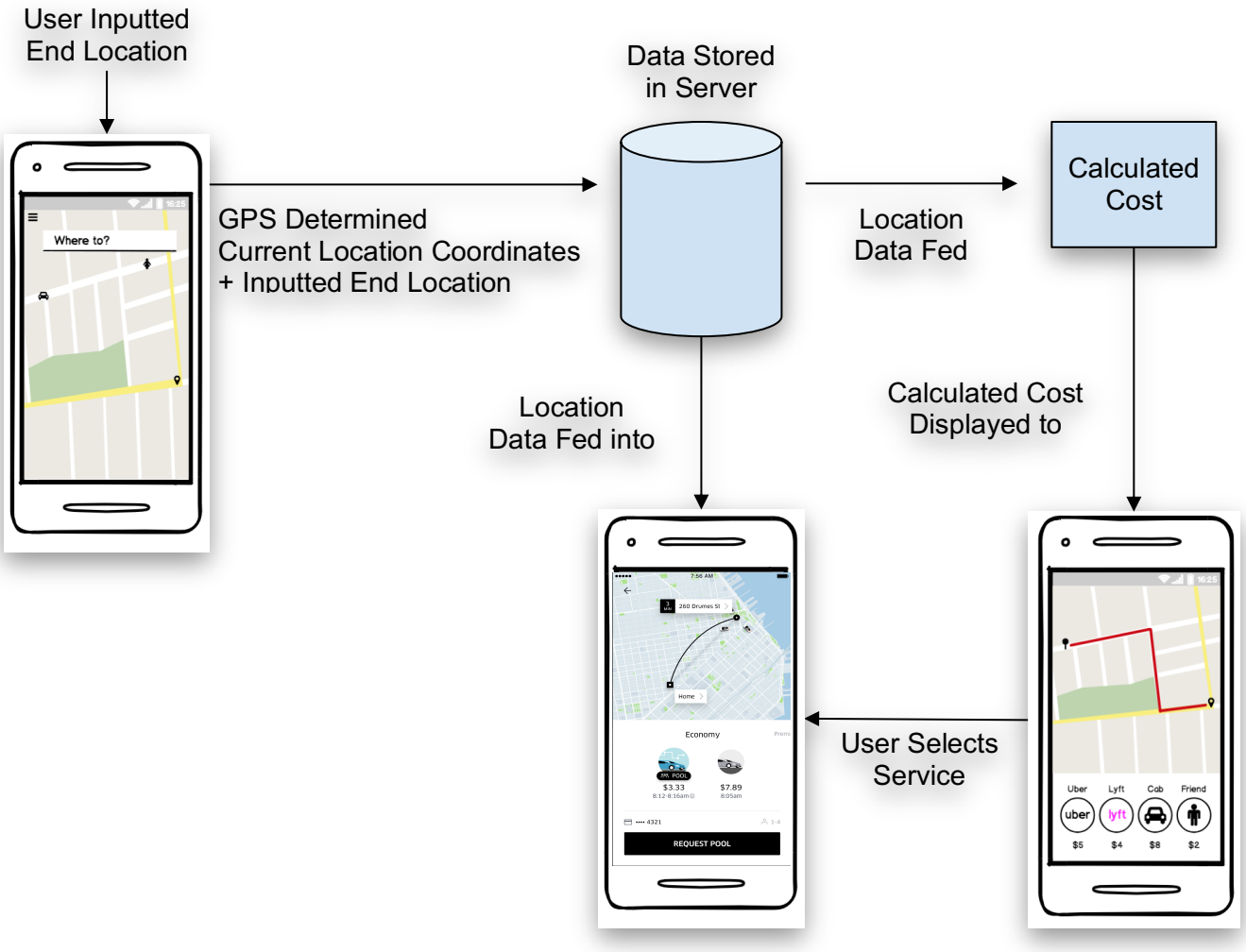

Figure 8: Functional Diagram

In compliance with the Android Studio framework, this application is broken up into activities. In the design of the application, there are three activities, the *Maps Activity*, *Login Activity*, and *Register Activity*. The following subsection will discuss the role of each activity.

Register Activity

The Register Activity is pretty self-explanatory in its name, here is where the logic pertaining to a user's registration is handled. Here is where the textual information from the text fields on the registration form are processed. Each of these fields creates a new collection in the Firebase Firestore cloud database. These fields

include: first and last name, email, phone number, username and password. What happens behind the scenes is that a hashmap is created with these values that gets fed directly into the database under a unique ID, which in this case in their users' username.

Login Activity

The Login Activity is also self-explanatory in its name. The login activity handles all logic associated with a user trying to use RollOut's login page. Here, the inputted text from a user is gathered and checked against the Firebase Firestore cloud database. If the provided combination of credentials (either username and password, email and password or phone number and password) isn't found in the database, the user will be prompted to register. If the credentials are found, the user will be displayed with RollOut's landing page.

Maps Activity

The Maps Activity is where the bulk of the logic is stored. It's here where permission handling is stored as well as location storage. Once the location is obtained using Android's internal location permission requesting, it's stored as a variable to be used in each ride sharing mobile applications' associated function. There are three functions held in this activity relating to the ride cost calculation - uberRide(), lyftRide() and cabRide(). These functions use the stored location data and determine a cost calculation through their associated APIs. Maps Activity also contains the logic to convert a user's desired location into a Geocoded location, meaning their text input is converted into longitude and latitude coordinates. This makes it possible for the APIs to use the location data properly. These locations are stored in RollOut's Firebase Firestore cloud database in this activity as well. Lastly, Maps Activity handles all of the Map logic such as camera bounds, and how the routes are displayed with polylines.

Budget

The initial team budget for this project was predicated on the assumption that in order to fully test the functionality of the application each group member would have to their own mobile device with an Android operating system installed. With this in mind various sourcing options for mobile phones were assessed.

Item	Quantity	Potential Suppliers	Estimated Cost per phone	Total Estimated Cost
Android Phone	3	Best Buy	$ 42.39	$ 127.17
		Micro Center	$ 31.79	$ 95.37
		AT&T	$ 127.19	$ 381.57
		Verison	$ 52.99	$ 158.97
		Craigslist	$ 50.00	$ 150.00
		Ebay	$ 31.79	$ 95.37
		Amazon	$ 47.69	$ 143.07
		Already Own	$ -	$ -
Android Development Environment	3	Andriod.com	$ -	$ -
Laptops	3	Already Own	N/A	N/A
Total Project Cost	$0.00			

Figure 9: Final Budget

After we shopped around some electronic stores for new android phones. But, due to ease and immediate availability, we decided to test our app using two phones that Chad McDaniel provided. In addition to this, we used an emulator through the Android Studio. Lastly, it was free to use the Android Development Environment on our laptops.

Timeline

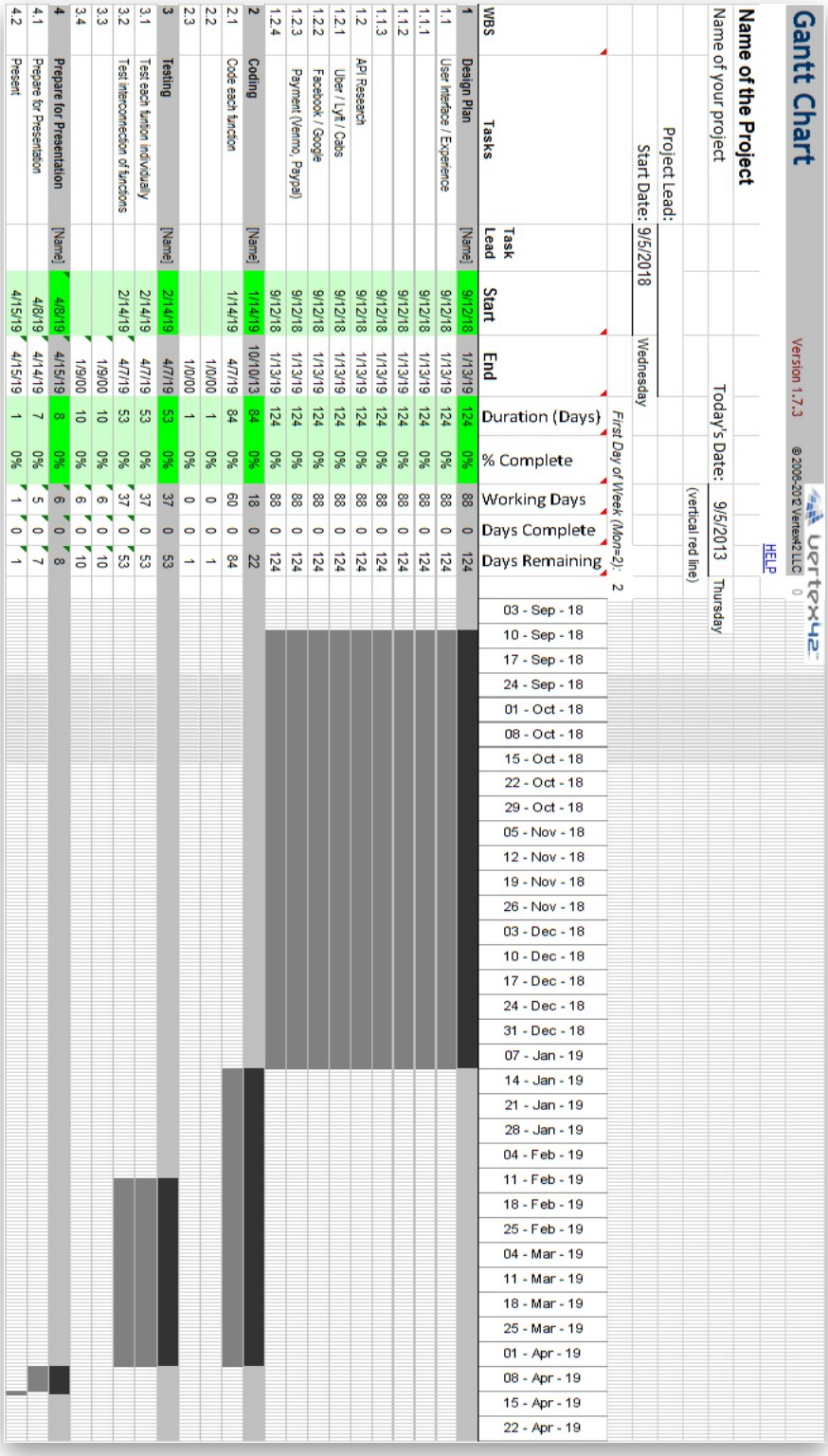

Figure 10: Gantt Chart

Overall, we were able to follow our schedule in a timely manner. The first semester was focused on developing ideas, fleshing out designs and functions, and research. The second semester is where we began to implement the functions designed in the first semester. Once we got a solid base down for our app we decided to begin testing using both an emulator through Android Studio and with actual phones that use Android OS. A few times we were delayed by setbacks. The biggest one came when dealing with the third-party API integration. There were in-app constraints from Uber and Lyft that we did not anticipate such as layout and use restrictions. Other delays were relatively minor and were settled either with deliberation between the group or further research.

Problems Encountered/Analysis of Problems Solved

Third-Party API Integration

When using the Uber and Lyft APIs, there were some logistical obstacles. First there was the display constraints; the formatting of the display of the Uber and Lyft prices was very strict. They needed to be a specific size and displayed in a specific way. This didn't create too big of a problem when using the emulator. However, since there are several different phones that use Android, making a universal display was tricky. Unfortunately, we weren't able to get around to changing this, but would be something to fix for future implementations. Secondly, if the user selected either an Uber or Lyft ride, it would have to open up that specific app in order to actually request the ride. Unfortunately, there was nothing we could do about this since Uber and Lyft won't allow us to open the request screen in our app. To get past this, we made it so when the user finds the ride he/she wants and the Uber or Lyft app opens up, all the information the user has inputted is already there, so all they need to is push the 'request ride' button.

Permission Requesting

One of the current issues in the digital world today is cyber security. This applies to our idea since we deal with user's private information. This information includes their login information to both Facebook and our app, profile pictures, location, and possibly credit card information. We decided to follow set standards that deal with security between apps and servers. The standard we followed was the TS 1.2 Compatibility.

Database Implementation

One set of challenges arose when it came to decide how to store user information. We needed a safe, secure way to store their data while also making it a bit easier on us to implement. We settled on Firebase since it met our criteria of being inexpensive in development, straightforward in implementation, and secure for the use of our app.

Learning Curve

Although we as a group have experience with software development, none of us had ever developed for Android before, so this became a big learning curve for us as we were figuring out how to actually develop this app as we were doing it. Hours of additional research were put into this app due to this fact. This slowed us down a decent amount but were able to apply our research to the app to make it work as we intended.

Android Studio

Using Android Studio posed its own set of problems that we did not think we would encounter. One of the issues that came up consistently was that this IDE would constantly need updating seemingly every time that launched it to work. Sometimes not updating it meant that certain functionalities of the program wouldn't work, and the debugging process was elongated for unnecessary reasons. Connection issues became prevalent when we were testing our app on actual phones. To run the app on our phones, we would need to plug it into the computer to upload what we had to the phones. However, there were sometimes issues with keeping the connection live between the phone and computer which would delay our testing process. To get around this, we decided to do most of our testing on the emulator.

Future Recommendations

Lime/Bird/Redbike Integration

With its rise in popularity, the next natural step is to integrate Lime/Bird electric scooters and Redbike stations into our app. Along with showing where the nearest scooters/stations are, we would also include the price approximation of the trip, the same way we do with ridesharing apps. In addition to this, we would add a

filter option so that the user would be able to sift through only the options they want instead of being bogged down with information**.**

Public Transportation

Following up on the Lime/Bird/Redbike integration, the next step we would take is to include public transportation. This would include buses, trains, trams, ferries, and other forms that a city provides. Again, the user would be able to sift through their preferred method of transportation with the filtering feature that we would include. Having public transportation added to our app increase its usability and would further streamline the user's decisions on travel.

Facebook Integration

To make inputting user information easier, we would implement a functionality where the user has the option to sign up/log in with Facebook. The information we would take from the users Facebook would be their profile picture and friends list. Additional security measures would be taken when handling this sensitive information.

iOS Application

In order to reach wider audience, developing the app for iOS would be necessary. We believe this would be simple as Kristin has experience in Xcode and developing for iOS devices.

Organized Events

One of the most significant additions we would make is to incorporate local events and activities that the user would be interested in. We would reach out to organizers, restaurants, venues, etc., to make the list, then have the user pick which event or activity they would like to catch a ride to.

Group Messaging

Continuing with the idea being able to share rides with friends, we would add a feature that would let users be able to get in touch with one another. Having this functionality in-app would make communication and making plans that much easier for our users.

Scheduling Rides

Another implementation we want to incorporate into our app is to be able to schedule a ride for later. The user would be able to pick a time and place for pick up. This can be very useful for users, especially when paired with the list of organized events.

One Button Ride-Hailing ("Drunk Mode")

This goes along with our idea of streamlining the process of finding a ride when the user is out drinking for the night. This feature would allow the user to open up the app and press one button that would automatically find and click on a ride. When setting this up, the user would input a destination (most likely their home), and a preference on which kind of ride they want the program to find (either the cheapest option or the fastest way to the destination). Then, all they have to do is press the "Drunk Mode" button, and a ride will quickly be found for them. The time between finding a ride this way and finding a ride a slower way could be the difference in deciding to get behind the wheel or getting home safely.

CONCLUSION

Results

Below is a histogram depicting the difference in the time it takes to find the best ride using Rollout versus opening Uber and Lyft (see Appendix E).

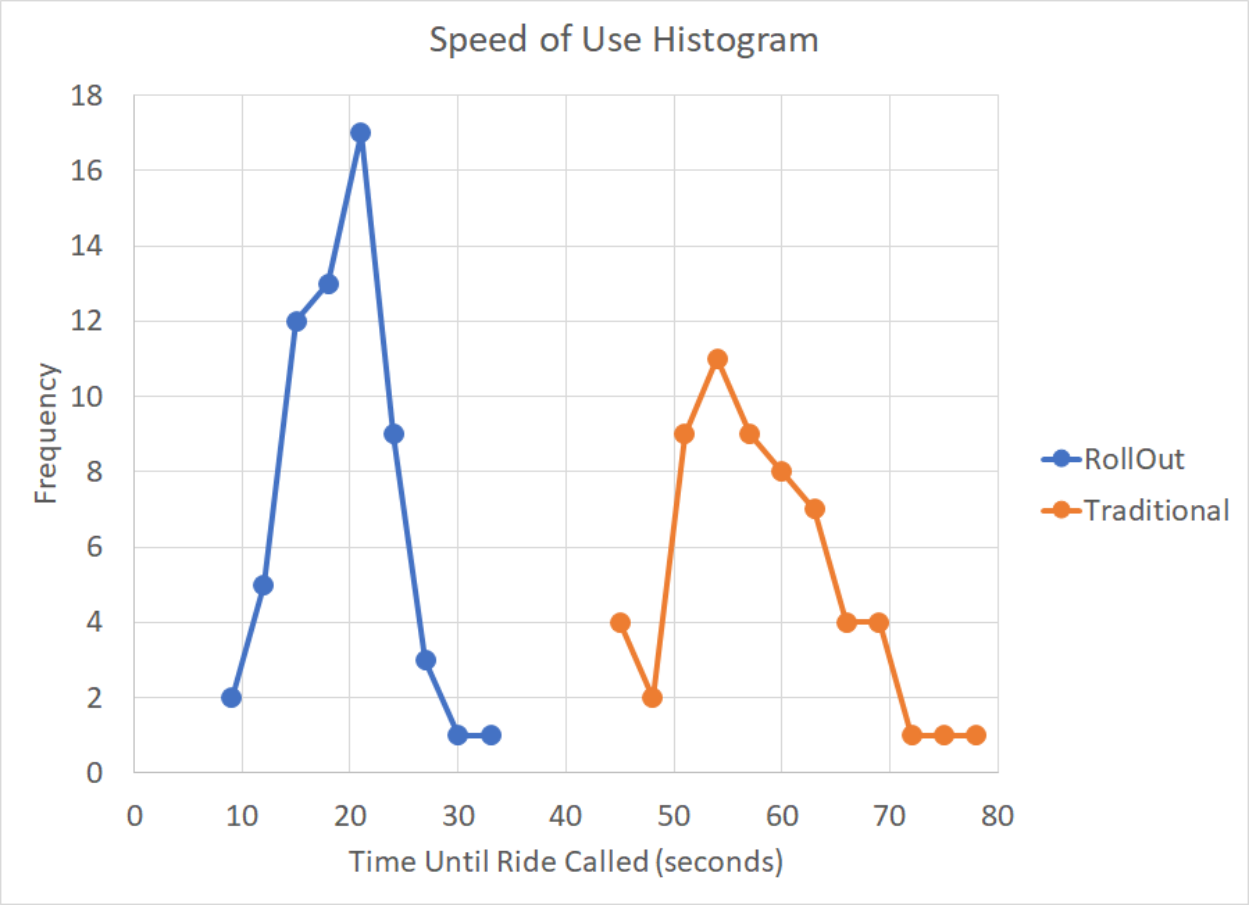

Figure 11: Usability Histogram

To test the usability of our app, we asked some friends and family to find the best (in this case best refers to most inexpensive) ride to a location using both the Lyft and Uber apps. Then, we asked them to find the best ride again using RollOut. We timed the two actions. The results showed that when using our app, the time it took to find the same ride took about 25 seconds fewer than using both the Lyft and Uber apps. This is a significant time not only in terms of being able to streamline the process of finding the

best ride, but also could be the difference between deciding to not get behind the wheel after a night of drinking.

References

"Impaired Driving: Get the Facts | Motor Vehicle Safety | CDC Injury ...." 16 Jun. 2017, Retrieved October 5, 2018, from CDC: ttps://www.cdc.gov/motorvehiclesafety/impaired_driving/impaired-drv_factsheet.html.

The main point of this article is simply a statement of facts about drinking and driving. This is a government page whose purpose is to educate the public about the effect drinking and driving has caused. The main conclusion to be taken from the article is that drinking and driving is still a very large problem in the United States. It is used in our argument to demonstrate the severity of the drinking and driving problem in the United States.

NHTSA. (n.d.). *Drunk Driving*. Retrieved December 12, 2018, from NHTSA: https://www.nhtsa.gov/risky-driving/drunk-driving

The main point of this article is to educate people on drinking and driving. This government page accomplishes this task through an all-encompassing discussion of the impact alcohol has on the ability to drive. It uses of statistics to show the current impact that drinking and driving has on the economy as well as how many people it affects per year. The main conclusion of this article is that not only is it illegal to drive drunk, but the that it is still a very large problem in the United States, and that the NHTSA is fully committed to getting rid of this problem plaguing the US. This is used in our argument for showing that drinking and driving is still a very large problem in the US.

Statista. (n.d.). *Facebook users worldwide 2018*. Retrieved December 12, 2018, from Statista: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

This web page has no overarching argument, but merely is a graph of the amount of active Facebook users over time. The purpose of the page is to show the progress of Facebook's market penetration, that they are the first social media site to reach the benchmark of one billion active users. This is measured by a user logging in within the past 30 days. This tool demonstrates the power and size of Facebook's platform and therefore indirectly shows a benefit of being associated with their brand. This is used in

our proposal for the justification of providing a Facebook login option to make accessibility easier.

Uber; MADD. (2015, January). *More Options. Shifting Mindsets. Driving Better Choices*. Retrieved December 12, 2018, from Uber: https://newsroom.uber.com/wp-content/uploads/2015/01/UberMADD-Report.pdf

This report demonstrates the impact that Uber has had on the amount of alcohol related driving accidents and deaths. Uber partnered with Mothers Against Drunk Driving (MADD) for this study showing the service to the community that Uber has provided by reducing the amount of people getting behind the wheel when they shouldn't. Though this article specifically talks about Uber's impact, a similar argument can be made for anything providing more, safe options to consumers. The article concludes that providing people with an easy alternative to driving, they have effectively reduced the number of drunk drivers on the road and therefore the amount of alcohol-related driving incidents (deaths and accidents). This argument is used to make a parallel argument for the potential life saving impact of our app.

Appendix A: Register Activity Java Code

```
1.  package com.example.kristin.rollout;
2.  import android.content.Intent;
3.  import android.support.annotation.NonNull;
4.  import android.support.v7.app.AppCompatActivity;
5.  import android.os.Bundle;
6.
7.  import com.google.android.gms.tasks.OnFailureListener;
8.  import com.google.android.gms.tasks.OnSuccessListener;
9.  import com.google.firebase.firestore.CollectionReference;
10. import com.google.firebase.firestore.FirebaseFirestore;
11.
12. import android.util.Log;
13. import android.view.View;
14. import android.widget.EditText;
15. import android.widget.TextView;
16.
17. import java.util.HashMap;
18. import java.util.Map;
19.
20. public class RegisterActivity extends AppCompatActivity {
21.
22.     private static final String TAG = "RegisterActivity";
23.     private CollectionReference rollOutDatabase = FirebaseFirestore.getInstance().colle
    ction("RollOutData");
24.     String userName;
25.     String userEmail;
26.     String userPhoneNumber;
27.     String userUsername;
28.     String userPassword;
29.
30.
31.     @Override
32.     protected void onCreate(Bundle savedInstanceState) {
33.         super.onCreate(savedInstanceState);
34.
35.         setContentView(R.layout.activity_register);
36.
37.         TextView loginLink = findViewById(R.id.loginLink);
38.
39.         loginLink.setOnClickListener(new View.OnClickListener() {
40.             @Override
41.             public void onClick(View v) {
42.                 Intent loginIntent = new Intent(RegisterActivity.this, LoginActivity.cl
    ass);
43.                 RegisterActivity.this.startActivity(loginIntent);
44.             }
45.         });
46.
47.     }
48.
49.
50.     public void registerUser(View v){
51.
52.         EditText userNameTextView = findViewById(R.id.userName);
53.         EditText userEmailTextView = findViewById(R.id.userEmail);
54.         EditText userPhoneNumberTextView = findViewById(R.id.userPhoneNumber);
```

```
55.         EditText userUsernameTextView = findViewById(R.id.userUsername);
56.         EditText userPasswordTextView = findViewById(R.id.userPassword);
57.
58.         userName = userNameTextView.getText().toString();
59.         userEmail = userEmailTextView.getText().toString();
60.         userPhoneNumber = userPhoneNumberTextView.getText().toString();
61.         userUsername = userUsernameTextView.getText().toString();
62.         userPassword = userPasswordTextView.getText().toString();
63.
64.         writeToDatabase();
65.
66.         Intent startMap = new Intent(RegisterActivity.this, LoginActivity.class);
67.         RegisterActivity.this.startActivity(startMap);
68.     }
69.
70.     public void writeToDatabase(){
71.
72.         Map<String, Object> userInfo = new HashMap<>();
73.
74.         userInfo.put("name", userName);
75.         userInfo.put("email", userEmail);
76.         userInfo.put("phone_number", userPhoneNumber);
77.         userInfo.put("username", userUsername);
78.         userInfo.put("password", userPassword);
79.
80.
81.         rollOutDatabase.document(userUsername).set(userInfo)
82.                 .addOnSuccessListener(new OnSuccessListener<Void>() {
83.                     @Override
84.                     public void onSuccess(Void aVoid) {
85.                         Log.d(TAG, "DocumentSnapshot successfully written!");
86.                     }
87.                 })
88.                 .addOnFailureListener(new OnFailureListener() {
89.                     @Override
90.                     public void onFailure(@NonNull Exception e) {
91.                         Log.d(TAG, "Error writing document", e);
92.                     }
93.                 });
94.     }
95. }
```

Appendix B: Login Activity Java Code

```
1.  package com.example.kristin.rollout;
2.
3.
4.  import android.content.Intent;
5.  import android.support.v7.app.AppCompatActivity;
6.  import android.os.Bundle;
7.  import android.view.View;
8.  import android.widget.EditText;
9.  import android.widget.TextView;
10.
11.
12. public class LoginActivity extends AppCompatActivity {
13.
14.     public static String userUsername;
15.     public static String userPassword;
16.
17.
18.     @Override
19.     protected void onCreate(Bundle savedInstanceState) {
20.         super.onCreate(savedInstanceState);
21.
22.         setContentView(R.layout.activity_login);
23.
24.         TextView registerLink = findViewById(R.id.registerLink);
25.
26.         registerLink.setOnClickListener(new View.OnClickListener() {
27.             @Override
28.             public void onClick(View v) {
29.                 Intent registerIntent = new Intent(LoginActivity.this, RegisterActivity
    .class);
30.                 LoginActivity.this.startActivity(registerIntent);
31.             }
32.         });
33.
34.     }
35.
36.     public void loginUser(View v) {
37.
38.         EditText userUsernameTextView = findViewById(R.id.userUsername);
39.         EditText userPasswordTextView = findViewById(R.id.userPassword);
40.
41.         userUsername = userUsernameTextView.getText().toString();
42.         userPassword = userPasswordTextView.getText().toString();
43.
44.         Intent loginIntent = new Intent(LoginActivity.this, MapsActivity.class);
45.         LoginActivity.this.startActivity(loginIntent);
46.
47.     }
48.
49.
50.
51.
52. }
```

Appendix C: Maps Activity Java Code

```
1.  package com.example.kristin.rollout;
2.
3.  import android.Manifest;
4.  import android.content.Context;
5.  import android.content.Intent;
6.  import android.content.pm.PackageManager;
7.  import android.location.Location;
8.  import android.net.Uri;
9.  import android.os.Build;
10. import android.support.annotation.NonNull;
11. import android.support.annotation.Nullable;
12. import android.support.v4.app.ActivityCompat;
13. import android.support.v4.app.FragmentActivity;
14. import android.os.Bundle;
15. import android.support.v4.content.ContextCompat;
16.
17. import android.util.Log;
18. import android.view.View;
19. import android.view.inputmethod.InputMethodManager;
20. import android.widget.Button;
21. import android.widget.TextView;
22. import android.widget.Toast;
23.
24. import com.example.kristin.rollout.directionhelpers.JSONParser;
25. import com.google.android.gms.common.ConnectionResult;
26. import com.google.android.gms.location.LocationListener;
27.
28. import android.location.Address;
29. import android.location.Geocoder;
30.
31. import com.google.android.gms.common.api.GoogleApiClient;
32. import com.google.android.gms.location.LocationRequest;
33. import com.google.android.gms.location.LocationServices;
34. import com.google.android.gms.maps.CameraUpdateFactory;
35. import com.google.android.gms.maps.GoogleMap;
36. import com.google.android.gms.maps.OnMapReadyCallback;
37. import com.google.android.gms.maps.SupportMapFragment;
38. import com.google.android.gms.maps.model.BitmapDescriptorFactory;
39. import com.google.android.gms.maps.model.LatLng;
40. import com.google.android.gms.maps.model.LatLngBounds;
41. import com.google.android.gms.maps.model.Marker;
42. import com.google.android.gms.maps.model.MarkerOptions;
43. import com.google.android.gms.maps.model.Polyline;
44. import com.google.android.gms.maps.model.PolylineOptions;
45.
46. import com.google.android.gms.tasks.OnCompleteListener;
47. import com.google.android.gms.tasks.OnFailureListener;
48. import com.google.android.gms.tasks.OnSuccessListener;
49. import com.google.android.gms.tasks.Task;
50. import com.google.firebase.database.DataSnapshot;
51. import com.google.firebase.database.DatabaseError;
52. import com.google.firebase.database.ValueEventListener;
53. import com.google.firebase.firestore.CollectionReference;
54. import com.google.firebase.firestore.DocumentReference;
55. import com.google.firebase.firestore.DocumentSnapshot;
56. import com.google.firebase.firestore.FirebaseFirestore;
```

```
57.
58. import com.lyft.lyftbutton.LyftButton;
59. import com.lyft.lyftbutton.RideParams;
60. import com.lyft.lyftbutton.RideTypeEnum;
61. import com.uber.sdk.android.core.UberButton;
62. import com.uber.sdk.android.rides.RideParameters;
63. import com.uber.sdk.android.rides.RideRequestButton;
64. import com.uber.sdk.rides.client.SessionConfiguration;
65. import com.uber.sdk.rides.client.ServerTokenSession;
66.
67. import com.lyft.networking.ApiConfig;
68.
69. import java.io.IOException;
70. import java.util.HashMap;
71. import java.util.List;
72. import java.util.Map;
73.
74. import com.example.kristin.rollout.directionhelpers.FetchURL;
75. import com.example.kristin.rollout.directionhelpers.TaskLoadedCallback;
76.
77.
78. public class MapsActivity extends FragmentActivity implements
79.         OnMapReadyCallback,
80.         TaskLoadedCallback,
81.         GoogleApiClient.ConnectionCallbacks,
82.         GoogleApiClient.OnConnectionFailedListener,
83.         LocationListener {
84.
85.     private GoogleMap mMap;
86.     private GoogleApiClient googleApiClient;
87.     private LocationRequest locationRequest;
88.     private Marker currentUserLocationMarker;
89.
90.     private LatLng currentLocationLongitudeLatitude;
91.     double currentLongitude;
92.     double currentLatitude;
93.     double dropoffLatitude;
94.     double dropoffLongitude;
95.     double pickupLatitude;
96.     double pickupLongitude;
97.     String pickupAddressString;
98.     Address dropoffAddress;
99.     Address pickupAddress;
100.
101.          final LoginActivity loginActivity = new LoginActivity();
102.          String userUsername = loginActivity.userUsername;
103.
104.          private static final String TAG = "MapsActivity";
105.
106.          double cabFare;
107.          String cabFare_string;
108.          Button cab_button;
109.          TextView cab_fare;
110.
111.          MarkerOptions dropoff_marker, pickup_marker;
112.          Polyline currentPolyline;
113.
114.          TextView dropoffLocationTextView;
115.          TextView pickupLocationTextView;
```

```
116.            TextView calculate_button;
117.
118.            com.uber.sdk.android.rides.RideRequestButton uber_button;
119.            com.lyft.lyftbutton.LyftButton lyft_button;
120.
121.            private static final int Request_User_Location_Code = 99;
122.
123.            public String LOCATION_LONGITUDE = "longitude";
124.            public String LOCATION_LATITUDE = "latitude";
125.
126.            public CollectionReference rollOutDatabase = FirebaseFirestore.getInstance()
    .collection("RollOutUsers");
127.            public CollectionReference rollOutLocationDatabase = FirebaseFirestore.getIn
    stance().collection("RollOutUserLocations");
128.
129.
130.            @Override
131.            protected void onCreate(Bundle savedInstanceState) {
132.                super.onCreate(savedInstanceState);
133.                setContentView(R.layout.activity_maps);
134.
135.
136.                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
137.                    checkUserLocationPermission();
138.                }
139.
140.                SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragment
    Manager()
141.                        .findFragmentById(R.id.map);
142.                mapFragment.getMapAsync(this);
143.
144.            }
145.
146.            // 'Where To' Click
147.            public void textInputButton(View v) throws IOException {
148.                dropoffLocationTextView = findViewById(R.id.dropoff_location);
149.                pickupLocationTextView = findViewById(R.id.pickup_location);
150.                calculate_button = findViewById(R.id.price_calculate);
151.                dropoffLocationTextView.setY(250);
152.                dropoffLocationTextView.setTextAlignment(2);
153.                dropoffLocationTextView.setPadding(50, 0, 0, 0);
154.                pickupLocationTextView.setVisibility(View.VISIBLE);
155.                pickupLocationTextView.setPadding(50, 0, 0, 0);
156.                calculate_button.setVisibility(View.VISIBLE);
157.
158.
159.                // Populates Current Location Text Field With Current Location
160.                Geocoder coder = new Geocoder(this);
161.                List<Address> currentLocationList;
162.                currentLatitude = currentLocationLongitudeLatitude.latitude;
163.                currentLongitude = currentLocationLongitudeLatitude.longitude;
164.                currentLocationList = coder.getFromLocation(currentLatitude, currentLong
    itude, 1);
165.                Address currentLocationAddress = currentLocationList.get(0);
166.                pickupAddressString = currentLocationAddress.getAddressLine(0);
167.                pickupLocationTextView.setText(pickupAddressString);
168.
169.                locationToDatabase();
170.
```

```
171.            }
172.
173.            public void getInput() throws IOException {
174.
175.                // Gets Text (Pickup and Dropoff Location) from Text View
176.                String dropoffLocationString = dropoffLocationTextView.getText().toStrin
    g();
177.                String pickupLocationString = pickupLocationTextView.getText().toString(
    );
178.
179.                // Geocodes Input into Address
180.                List<Address> dropoffList;
181.                List<Address> pickupList;
182.
183.                Geocoder coder = new Geocoder(this);
184.
185.                dropoffList = coder.getFromLocationName(dropoffLocationString, 1);
186.                pickupList = coder.getFromLocationName(pickupLocationString, 1);
187.
188.                dropoffAddress = dropoffList.get(0);
189.                pickupAddress = pickupList.get(0);
190.
191.                dropoffLatitude = dropoffAddress.getLatitude();
192.                dropoffLongitude = dropoffAddress.getLongitude();
193.
194.                pickupLatitude = pickupAddress.getLatitude();
195.                pickupLongitude = pickupAddress.getLongitude();
196.
197.
198.                createRide();
199.
200.
201.            }
202.
203.            public void cameraBounds(){
204.
205.                // Map Configurations
206.                pickup_marker = new MarkerOptions().position(new LatLng(pickupLatitude,
    pickupLongitude)).title("Pick Up");
207.                dropoff_marker = new MarkerOptions().position(new LatLng(dropoffLatitude
    , dropoffLongitude)).title("Drop Off");
208.                pickup_marker.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescripto
    rFactory.HUE_ROSE));
209.
210.                mMap.addMarker(pickup_marker);
211.                mMap.addMarker(dropoff_marker);
212.
213.
214.
215.                currentLocationLongitudeLatitude = new LatLng(pickupAddress.getLatitude(
    ), pickupAddress.getLongitude());
216.
217.                String url = getUrl(pickup_marker.getPosition(), dropoff_marker.getPosit
    ion(), "driving");
218.                new FetchURL(this).execute(url, "driving");
219.
220.                // Configuring Camera Bounds
221.                boolean hasPoints = false;
222.                Double maxLat = null, minLat = null, minLon = null, maxLon = null;
```

```
223.                if (currentPolyline != null && currentPolyline.getPoints() != null) {
224.                    List<LatLng> pts = currentPolyline.getPoints();
225.                    for (LatLng coordinate : pts) {
226.
227.                        maxLat = maxLat != null ? Math.max(coordinate.latitude, maxLat)
    : coordinate.latitude;
228.                        minLat = minLat != null ? Math.min(coordinate.latitude, minLat)
    : coordinate.latitude;
229.                        maxLon = maxLon != null ? Math.max(coordinate.longitude, maxLon)
    : coordinate.longitude;
230.                        minLon = minLon != null ? Math.min(coordinate.longitude, minLon)
    : coordinate.longitude;
231.                        hasPoints = true;
232.                    }
233.                }
234.
235.                if (hasPoints) {
236.                    LatLngBounds.Builder builder = new LatLngBounds.Builder();
237.                    builder.include(new LatLng(maxLat, maxLon));
238.                    builder.include(new LatLng(minLat, minLon));
239.                    mMap.moveCamera(CameraUpdateFactory.newLatLngBounds(builder.build(),
    48));
240.                }
241.
242.            }
243.
244.            public void createRide() {
245.
246.                lyftRide();
247.                uberRide();
248.                cabRide();
249.            }
250.
251.            // Calculate Price Click
252.            public void priceCalculateButton(View v) throws IOException {
253.                uber_button = findViewById(R.id.uber_button);
254.                lyft_button = findViewById(R.id.lyft_button);
255.                cab_button = findViewById(R.id.cab_button);
256.                cab_fare = findViewById(R.id.cab_fare);
257.                cab_fare.setText(cabFare_string);
258.                cab_fare.setVisibility(View.VISIBLE);
259.                cab_button.setVisibility(View.VISIBLE);
260.                uber_button.setVisibility(View.VISIBLE);
261.                lyft_button.setVisibility(View.VISIBLE);
262.                InputMethodManager imm = (InputMethodManager) getSystemService(Context.I
    NPUT_METHOD_SERVICE);
263.                imm.hideSoftInputFromWindow(v.getWindowToken(), 0);
264.                getInput();
265.                cameraBounds();
266.            }
267.
268.            public void uberRide() {
269.                // Uber Integration
270.                RideRequestButton requestButton = new RideRequestButton(this);
271.                UberButton uberButton = findViewById(R.id.uber_button);
272.
273.                RideParameters rideParams = new RideParameters.Builder()
274.                        .setPickupLocation(pickupLatitude, pickupLongitude, "", "")
275.                        .setDropoffLocation(dropoffLatitude, dropoffLongitude, "", "")
```

```
276.                        .setProductId("a1111c8c-c720-46c3-8534-2fcdd730040d")
277.                        .build();
278.
279.                SessionConfiguration config = new SessionConfiguration.Builder()
280.                        .setClientId("-8NCdgaNOlzqc9mTkp4WMU4l5cm0wp2a")
281.                        .setServerToken("ocXTg92LK-TjXCLC97lTJPly6WHyAbFbTdPnp1dV")
282.                        .build();
283.                ServerTokenSession session = new ServerTokenSession(config);
284.
285.
286.                requestButton.setRideParameters(rideParams);
287.                requestButton.setSession(session);
288.                requestButton.loadRideInformation();
289.            }
290.
291.            public void lyftRide() {
292.                // Lyft Integration
293.                ApiConfig apiConfig = new ApiConfig.Builder()
294.                        .setClientId("93hozlE-5V07")
295.                        .setClientToken("g7RBXeaGqQmVCk775iWW4ZQJA+I52Y46O5rYRa7b4GsMiqD
    nwjssYkydlycU4Fzs7CG3WnH+0K23DtCUxmeYHHWg9hgcvaJCWPd4TJov5DkPBXL2kO83Icw=")
296.                        .build();
297.
298.                LyftButton lyftButton = findViewById(R.id.lyft_button);
299.                lyftButton.setApiConfig(apiConfig);
300.
301.                RideParams.Builder rideParamsBuilder = new RideParams.Builder()
302.                        .setPickupLocation(pickupLatitude, pickupLongitude)
303.                        .setDropoffLocation(dropoffLatitude, dropoffLongitude);
304.                rideParamsBuilder.setRideTypeEnum(RideTypeEnum.CLASSIC);
305.
306.                lyftButton.setRideParams(rideParamsBuilder.build());
307.                lyftButton.load();
308.            }
309.
310.            public void cabRide() {
311.                // Cab Integration
312.
313.                JSONParser jsonParser = new JSONParser();
314.
315.                String distance = jsonParser.distance;
316.                String duration = jsonParser.duration;
317.
318.                distance = distance.replace(" mi", "");
319.                duration = duration.replace(" mins", "");
320.
321.                float distance_float = Float.valueOf(distance);
322.                float duration_float = Float.valueOf(duration);
323.
324.                cabFare = (3.75 + .25 * ((duration_float * 60) / 33) + 2.00 * (distance_
    float));
325.
326.                cabFare = Math.round(cabFare * 100.0) / 100.0;
327.
328.                cabFare_string = "$" + Double.toString(cabFare);
329.            }
330.
331.            public void callCab(View v) {
332.                Intent callIntent = new Intent(Intent.ACTION_CALL);
```

```
333.                callIntent.setData(Uri.parse("tel:5135492469"));
334.                if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CALL_PH
    ONE) != PackageManager.PERMISSION_GRANTED) {
335.                    return;
336.                }
337.                startActivity(callIntent);
338.            }
339.
340.            private String getUrl(LatLng origin, LatLng dest, String directionMode) {
341.                String str_origin = "origin=" + origin.latitude + "," + origin.longitude
    ;
342.                String str_dest = "destination=" + dest.latitude + "," + dest.longitude;

343.                String mode = "mode=" + directionMode;
344.                String parameters = str_origin + "&" + str_dest + "&" + mode;
345.                String output = "json";
346.                String url = "https://maps.googleapis.com/maps/api/directions/" + output
     + "?" + parameters + "&key=" + getString(R.string.google_maps_key);
347.                return url;
348.            }
349.
350.            public void locationToDatabase(){
351.
352.                Map<String, Object> userLocation = new HashMap<>();
353.
354.                userLocation.put(LOCATION_LONGITUDE, currentLongitude);
355.                userLocation.put(LOCATION_LATITUDE, currentLatitude);
356.
357.
358.                rollOutLocationDatabase.document(userUsername).set(userLocation)
359.
360.                        .addOnSuccessListener(new OnSuccessListener<Void>() {
361.                            @Override
362.                           public void onSuccess(Void aVoid) {
363.                                Log.d("Success", "DocumentSnapshot successfully written!
    ");
364.                            }
365.                        })
366.                        .addOnFailureListener(new OnFailureListener() {
367.                            @Override
368.                            public void onFailure(@NonNull Exception e) {
369.                                Log.d("Failure", "Error writing document", e);
370.                            }
371.                        });
372.
373.            }
374.
375.            @Override
376.            public void onMapReady(GoogleMap googleMap) {
377.                mMap = googleMap;
378.
379.                if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_F
    INE_LOCATION) == PackageManager.PERMISSION_GRANTED)
380.                {
381.
382.                    buildGoogleApiClient();
383.
384.                    mMap.setMyLocationEnabled(true);
385.
```

```
386.                 }
387.             }
388.
389.             public boolean checkUserLocationPermission(){
390.                 if(ContextCompat.checkSelfPermission(this,Manifest.permission.ACCESS_FIN
    E_LOCATION) != PackageManager.PERMISSION_GRANTED)
391.                 {
392.                     if(ActivityCompat.shouldShowRequestPermissionRationale(this,Manifest
    .permission.ACCESS_FINE_LOCATION))
393.                     {
394.                         ActivityCompat.requestPermissions(this, new String[]{Manifest.pe
    rmission.ACCESS_FINE_LOCATION}, Request_User_Location_Code);
395.                     }
396.
397.                     else
398.                     {
399.                         ActivityCompat.requestPermissions(this, new String[]{Manifest.pe
    rmission.ACCESS_FINE_LOCATION}, Request_User_Location_Code);
400.                     }
401.
402.                     return false;
403.                 }
404.
405.                 else
406.                 {
407.                     return true;
408.                 }
409.             }
410.
411.
412.             @Override
413.             public void onRequestPermissionsResult(int requestCode, @NonNull String[] pe
    rmissions, @NonNull int[] grantResults) {
414.                 switch (requestCode) {
415.                     case Request_User_Location_Code:
416.                         if(grantResults.length > 0 && grantResults[0] == PackageManager.
    PERMISSION_GRANTED)
417.                         {
418.                             if(ContextCompat.checkSelfPermission(this, Manifest.permissi
    on.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED)
419.                             {
420.                                 if(googleApiClient == null)
421.                                 {
422.                                     buildGoogleApiClient();
423.                                 }
424.                                 mMap.setMyLocationEnabled(true);
425.                             }
426.                         }
427.                         else
428.                         {
429.                             Toast.makeText(this,"Permission Denied", Toast.LENGTH_SHORT)
    ;
430.                         }
431.
432.                         return;
433.                 }
434.
435.             }
436.
```

```
437.            protected synchronized void buildGoogleApiClient(){
438.                googleApiClient = new GoogleApiClient.Builder(this)
439.                        .addConnectionCallbacks(this)
440.                        .addOnConnectionFailedListener(this)
441.                        .addApi(LocationServices.API)
442.                        .build();
443.
444.                googleApiClient.connect();
445.            }
446.
447.
448.            // Where Location is Being Updated
449.            @Override
450.            public void onLocationChanged(Location location) {
451.
452.
453.                if(currentUserLocationMarker != null)
454.                {
455.                    currentUserLocationMarker.remove();
456.                }
457.
458.                currentLocationLongitudeLatitude = new LatLng(location.getLatitude(), lo
    cation.getLongitude());
459.
460.                MarkerOptions markerOptions = new MarkerOptions();
461.                markerOptions.position(currentLocationLongitudeLatitude);
462.                markerOptions.title("User Current Location");
463.                markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescripto
    rFactory.HUE_ROSE));
464.
465.                currentUserLocationMarker = mMap.addMarker(markerOptions);
466.
467.                MarkerOptions martyMarker = new MarkerOptions().position(new LatLng(39.1
    31800, -84.516840)).title("mcdanich");
468.                martyMarker.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorF
    actory.HUE_GREEN));
469.                mMap.addMarker(martyMarker);
470.
471.                MarkerOptions connorMarker = new MarkerOptions().position(new LatLng(39.
    131794, -84.518396)).title("silvacg");
472.                connorMarker.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptor
    Factory.HUE_BLUE));
473.                mMap.addMarker(connorMarker);
474.
475.                mMap.moveCamera(CameraUpdateFactory.newLatLng(currentLocationLongitudeLa
    titude));
476.                mMap.animateCamera(CameraUpdateFactory.zoomBy(14));
477.
478.                if(googleApiClient != null)
479.                {
480.
481.                    LocationServices.FusedLocationApi.removeLocationUpdates(googleApiCli
    ent,this);
482.
483.                }
484.            }
485.
486.            @Override
487.            public void onConnected(@Nullable Bundle bundle) {
```

```
488.                locationRequest = new LocationRequest();
489.                locationRequest.setInterval(1100);
490.                locationRequest.setFastestInterval(1100);
491.                locationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCU
    RACY);
492.
493.                if (ContextCompat.checkSelfPermission(this,Manifest.permission.ACCESS_FI
    NE_LOCATION) == PackageManager.PERMISSION_GRANTED)
494.                {
495.
496.                    LocationServices.FusedLocationApi.requestLocationUpdates(googleApiCl
    ient,locationRequest,this);
497.                }
498.
499.            }
500.
501.            @Override
502.            public void onConnectionSuspended(int i) {
503.
504.            }
505.
506.            @Override
507.            public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {

508.
509.            }
510.
511.            @Override
512.            public void onTaskDone(Object... values) {
513.                if(currentPolyline != null){
514.                    currentPolyline.remove();
515.                }
516.                currentPolyline = mMap.addPolyline((PolylineOptions)values[0]);
517.
518.            }
519.
520.        }
```

Appendix D: Android Manifest XML Code

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.      package="com.example.kristin.rollout">
4.      <!-- To auto-
    complete the email text field in the login form with the user's emails -->
5.      <uses-permission android:name="android.permission.GET_ACCOUNTS" />
6.      <uses-permission android:name="android.permission.READ_PROFILE" />
7.      <uses-permission android:name="android.permission.READ_CONTACTS" />
8.      <!--
9.           The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
10.          Google Maps Android API v2, but you must specify either coarse or fine
11.          location permissions for the 'MyLocation' functionality.
12.     -->
13.     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
14.     <uses-permission android:name="android.permission.INTERNET" />
15.     <uses-
    permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />

16.     <uses-
    permission android:name="android.permission.ACCESS_COARSE_LOCATION_LOCATION" />
17.     <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
18.     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
19.     <uses-permission android:name="android.permission.CALL_PHONE" />
20.
21.     <application
22.         android:allowBackup="true"
23.         android:icon="@mipmap/ic_launcher"
24.         android:label="@string/app_name"
25.         android:roundIcon="@mipmap/ic_launcher_round"
26.         android:supportsRtl="true"
27.         android:theme="@style/AppTheme">
28.         <activity android:name=".RegisterActivity"></activity>
29.
30.         <activity
31.             android:name=".LoginActivity"
32.             android:label="@string/title_activity_login" >
33.
34.             <intent-filter>
35.                 <action android:name="android.intent.action.MAIN" />
36.
37.                 <category android:name="android.intent.category.LAUNCHER" />
38.             </intent-filter>
39.
40.         </activity>
41.
42.         <meta-data
43.             android:name="com.google.android.geo.API_KEY"
44.             android:value="@string/google_maps_key" />
45.
46.         <activity
47.             android:name=".MapsActivity"
48.             android:label="@string/title_activity_maps">
49.
50.         </activity>
51.
52.     </application>
```

```
53.
54. </manifest>
```

Appendix E: Usability Results Raw Data

RollOut	Normal		Bin	RollOut	Traditional
15.4	47.7		0		
10.8	63.8		3		
16.3	58.4		6		
16.3	58.7		9	2	
12.9	63		12	5	
19.9	45.4		15	12	
18	58.2		18	13	
13.5	59.2		21	17	
20.7	70		24	9	
20.7	51.8		27	3	
25.2	63.2		30	1	
18.4	59.3		33	1	
24.6	57.5		36		
11.4	57.8		39		
14.4	59.3		42		
14.7	60.3		45		4
15.8	58.9		48		2
16.3	54		51		9
20.3	60.6		54		11
21.7	57.1		57		9
13	53.1		60		8
11.5	58.4		63		7
15.6	57.1		66		4

18.3	61.1		69		4
13.2	66.5		72		1
14	64.2		75		1
17.1	52.1		78		1
20	62.6		81		
17	57		84		
15.8	53.6		87		
15.5	51.7		90		
18.8	63.3				
14	46.6				
16.4	59.5				
20.1	58.3				
17.2	43.1				
20.4	55.4				
13.9	57.5				
16.9	50				
15.9	60				
21	57.1				
13	61.4				
22.7	35.6				
16.6	36.5				
14.2	58				
24.8	49.8				
15.5	48				
21.4	66.4				
24.2	54.2				
11.2	54.3				
9.4	56.8				

13.4	54.3				
11.1	67.5				
15.2	49.2				
19.3	59.8				
14.4	61.6				
18.4	60.3				
20.3	60.8				
18.8	62.5				

Appendix F: Expo Poster