A Proof of Concept of a Carpooled Ridesharing System for the USC Community

Nicolas Paolo M. Pepito[1], Christian Benedict C. Gonzales[1], Charles R. Ausejo Jr.[1],
Engr. Van B. Patiluna[2], Engr. Elline L. Fabian[2]

[1]*Undergraduate Student, Department of Computer Engineering, University of San Carlos, Talamban Cebu City, Philippines*
[2]*Faculty Member/Adviser, Department of Computer Engineering, University of San Carlos, Talamban Cebu City, Philippines*

Abstract— **With the steady rise of Cebu's economic growth, the number of cars in the province had also risen in number. Because of this, the roads and highways are more condensed than ever making heavy traffic one of the main daily struggles of a commuting Cebuano. With this, what if we were able to use these cars in a more efficient manner wherein more people are accommodated per vehicle? The aim of this study is to develop a mobile app that tests this concept by allowing car owners within the University of San Carlos Cebu, whether they be professors, students, or staff, to share their empty seats with USC community members along their route in exchange of a reward in the form of points. The points rewarded will be dependent on the number of passengers the driver voluntarily accommodated along with the corresponding distance of each passenger's travel. Furthermore, this paper aims to bring about a glimpse of the concept's marketability based on how the system is received as a whole by the USC community.**

Keywords—transportation, android, app, mobile, traffic

I. Introduction

Ridesharing is not a new concept. In fact, it is well-ingrained in Filipino culture in the form of the Philippine jeepney, a post-World War II innovation and the undisputed "King of the Road" [1]. The concept of a Philippine jeepney is simple: people going the same route share a ride and in exchange they pay a cheaper fare than what they would've if they were to hail a public transport for themselves alone. However, Filipinos are also familiar with the other side of the coin when it comes to PUJs. One immediate con that comes to mind is the sheer hassle of actually hailing and getting in a jeepney. There are approximately 3,665 traditional PUJs operating in Cebu Province and in the three highly-urbanized cities of Mandaue, Lapu-Lapu and Cebu [2]. Although the number of PUJs is high, the number of commuters will always beat the number of available seats in PUJs. In result, commuters are more likely to experience the pain of not being able to control when they can hail a ride. However, there is an overlooked solution that commuters miss - the possibility of hailing a ride with a private vehicle.

Surely the private vehicles are harder to hail since they are bought to be obviously private. However, with the flexibility of modern technology, a medium can be built to inform possible ride-hailers that they would be paying for the service a private vehicle driver willingly offers. This concept is the ride-sharing system by utilizing the available space in a private vehicle. Private vehicles provide more comfort for the passenger and at the same time the fare is being shared between another ride-hailer. Brian C. states that the environmental benefits of ride-sharing can be both in terms of reductions in emissions and the vehicle kilometers travelled [3]. Individual benefits also include shared travel costs, travel time savings from high occupancy vehicle lanes, and reduced commute stress [4]. The application that will be developed by the proponents shall enable Members of the USC community to experience the same benefits mentioned without having to pay. Instead, the application shall have a unique reward system. Unlike the current ridesharing apps of today in the likes of Angkas, Uber, and Grab, this mobile app aims for users to focus more on the act of sharing to their peers more than to use the app as a means of earning money. Hence, this study aims to gauge the feasibility of a carpooled ride-sharing system by designing, building, then observing and measuring how it fares when deployed to real users taken from the USC community.

A. *Review of Related Literature*

This chapter focuses on literature review, with the purpose of describing relevant studies regarding ridesharing as well as key concepts associated with the subject. As the researchers had read and studied literary sources, related articles were found to have created carpooling applications that are similar with the concepts behind this study. Utilizing these as references would grant the proponents knowledge pertaining to one of its primary goals and objectives, how similar carpooling applications are built in past researches.

On **Real Time City Scale Taxi Ridesharing**, *Shuo Ma and Yu Zheng* developed a taxi-sharing system that accepts taxi passengers' real-time ride requests sent from smartphones and schedules taxis to pick them up via ridesharing, subject to time, capacity, and monetary constraints, in turn, guarantees that passengers pay less, and drivers earn more compared with no taxi-sharing used [5].

Sanjay Khunger & *Darryl Carr*'s **Real-Time Ride Share System** describes the architecture behind these ride-share systems. The setup includes receiving a ride request from one or more riders, which involves the rider location and the rider destination, identifying a beginning waypoint and an ending waypoint for vehicle travel, determining a route based on the beginning and ending waypoints, choosing a rider whose rider location or rider destination is geographically closest to the determined vehicle route, presenting the information to the driver, and if the driver accepts, alerting the rider of the acceptance [6].

For complex travel routes, an algorithm that stands out with the rest is provided by *Niels Agatz, Alan Erera and Martin Savelsbergh* on their **Dynamic Ride-Sharing: A Simulation Study in Metro Atlanta**. The paper states two arrangements when it comes to assigning a rider to a driver, the Single Driver, Multiple Rider, and the Single Rider, Multiple Driver. In Single Rider, Multiple Rider, drivers may be willing to provide rides to several riders on a trip. Although this may mean comfort for the riders, this on the other hand means that the pickup and drop-off of the riders contribute to the rise of complexity of routing decisions. In Single Rider, Multiple Driver, the riders transfer between drivers, hence travelling with more than one driver to reach the drop-off point. This is sometimes called as the multi-hop rideshare system [7].

On how the society reacted towards these ride-sharing applications, **Factors That Affect Filipinos' Mentality On Ride Sharing** by *Laiza Limpin* concluded that the growing demands of transportation means, and its practical importance, paved way into its wide acceptance in the country. This is brought out as the findings revealed that the lack of adequate mass transport contributed to the increasing acceptability in the country, further, it also revealed that people perceive the ride-share system as a sustainable way of travelling. However, with its representation being seen to promote green image, its true effects have not been thoroughly investigated. The findings of this study could contribute to this research as it is seen to have implications towards the improvement of ride-sharing services in the country [8].

Issues on ride-share system acceptance is discussed by *S. Vayouphacks's* **Ridesharing in Developing Countries: Perspectives from India and Thailand**. Although its considerable benefits, there are still concerns that impact how ridesharing platform operators adapt to society, these include trust and safety between passengers, ridesharing drivers, and the operators, and legal and regulatory challenges. Summing it all, it is found that innovation causes social issues due to its rapid infiltration of new advancements in society. Although the research local is from India and Thailand, a similar case is presented in a study conducted by *Yuana et al.* on ridesharing in Indonesia and the Philippines. Lastly, although of different geographies, demography, and culture, as Philippines also is a developing country, their experiences may also in one way or another, represent ours [9].

B. *Goals and Objectives*

The core goal of this research is to determine the feasibility of a carpooled ride-sharing system by designing and building a mobile platform intended for the said service and deploying it, open for installation and usage for all members of the USC community. In particular, the proponents aim to meet the following objectives:

1. Build a full-stack mobile application, complete with a user interface and backend to service the clients with the minimum features required to completely deliver the intended carpooled ridesharing service.

2. Design and build the core algorithm that matches the compatible drivers with riders that require the service.

3. Encourage camaraderie and friendship among USC community members by providing a point system as incentive to users of the app, fair to both drivers and riders. Furthermore, these points will be convertible to CES.

4. Design a secure authentication system that allows the app to be exclusively usable to members of the USC community only.

C. *Scope and Limitations*

Though the mobile application will be built with a cross-platform framework (see *Software Design* below), the proponents will focus on optimizing the app for the Android platform. Given this, the study is also therefore limited to Android users as its subjects. The reasons for this decision include the following:

- None of the proponents own or have access to a Mac computer, which is a requirement for iOS development

- To minimize the budget. An Apple developer account costs around 5,000 Php per year.

The features that will be built into the application will be the minimum required to achieve the intended purpose of the application as detailed in Chapter 2. This is the main goal of the study. However, should time permit, additional features may be added.

Upon deployment, the only users allowed to install and use the app are those that are within the USC community including but not limited to the professors, staff, students and employees. In-app authentication checks will be added to ensure exclusivity (See *Authentication and Exclusivity* for details). In line with this, app testers external to the development team (the proponents), will be taken from the USC community as well.

The reward system will be purely point-based. This means that passengers will be granted a free ride in exchange for granting points to the drivers. Furthermore, to make this reward system more tangible, points will be convertible to CES points for USC students. However, the proponents shall only be concerned in generating points through the application. The conversion of points from the application to CES points would be taken care of by the University's own guidelines or decision. As a final note, this paper, and therefore its proponents, absolutely do not condone the exchange of money among users as a means of payment upon using the services provided by the app in adherence to existing rules and regulations set by the LTFRB.

II. Methodology

A. *Conceptual Framework*

The proposed system's general concept can be succinctly described by detailing a general timeline of events that happens at every transaction as shown in Fig. 1. Note that though not explicitly shown in the diagram, a driver is able to accommodate multiple passengers, so long that they can be picked up and dropped off along the driver's route.

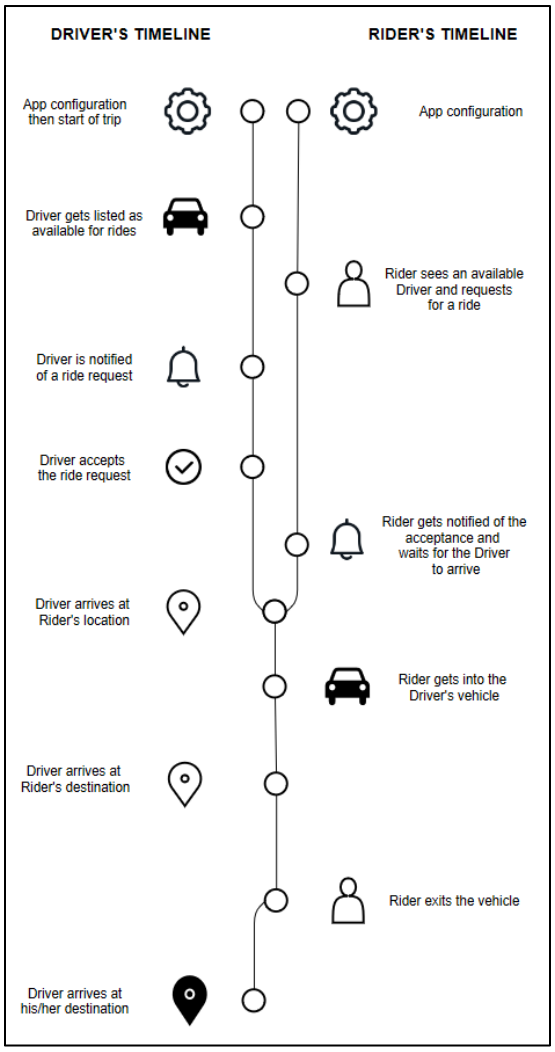
DRIVER'S TIMELINE
RIDER'S TIMELINE
App configuration then start of trip
App configuration
Driver gets listed as available for rides
Rider sees an available Driver and requests for a ride
Driver is notified of a ride request
Driver accepts the ride request
Rider gets notified of the acceptance and waits for the Driver to arrive
Driver arrives at Rider's location
Rider gets into the Driver's vehicle
Driver arrives at Rider's destination
Rider exits the vehicle
Driver arrives at his/her destination

Fig. 1. Conceptual Framework / General Timeline

The first thing in every transaction is for a user to configure the following settings on the app:

- To ride or to drive
- The date and time of the trip
- The starting location and the destination

After configuring the settings above, one final option is presented to the user depending on whether the user has chosen to be a rider or a driver. If the user opted to be a driver, the user will be asked to input the maximum number of people that he/she intends to accommodate. Otherwise, the user will be asked to input the number of people that he/she will be riding with (defaults to '1' indicating the user is travelling alone).

All the drivers that are currently traveling will be listed in the cloud as available. The riders would be able to see which drivers are along their route. A rider can request two or more of these drivers but will ultimately complete the transaction with one. Upon request, the driver will instantaneously be notified and will be presented the option to either accept or reject the ride request. The rider will only be notified of the drivers who accepted his/her request. The mobile app will indicate a rendezvous point as to where the rider will be picked up. The rider will be notified upon the requested driver's arrival. A driver is able to cater to requests up to the set maximum number of accommodations. If the driver's passenger count reaches that number, the driver will be listed as unavailable in the database until the current load is decreased i.e., a passenger had been dropped off. The system will ensure to pair a driver with passengers that will be dropped before he/she arrives at his/her intended destination or at the very same destination itself.

Finally, as mentioned above, the payment system will be in points. The points will be awarded to both ends of the transaction (Driver and Rider) upon completion. Riders are automatically awarded once they step in the Driver's vehicle while Drivers are awarded points after a Rider 'pays' it to them when they are successfully transported to their intended destination.

B. *Program Flow*

Fig. 2 shows all the main states the client mobile app can take. There is one state not shown in the diagram which is the state that occurs when the user is not authenticated. When this happens, the app presents the user a log-in screen. Otherwise, the app will begin at the home state. Here a user can configure to either ride or to drive, the starting and the end destination then the desired date and time of the trip. After that, the app will take on a different state path depending on whether the user has opted to be a rider or a driver.

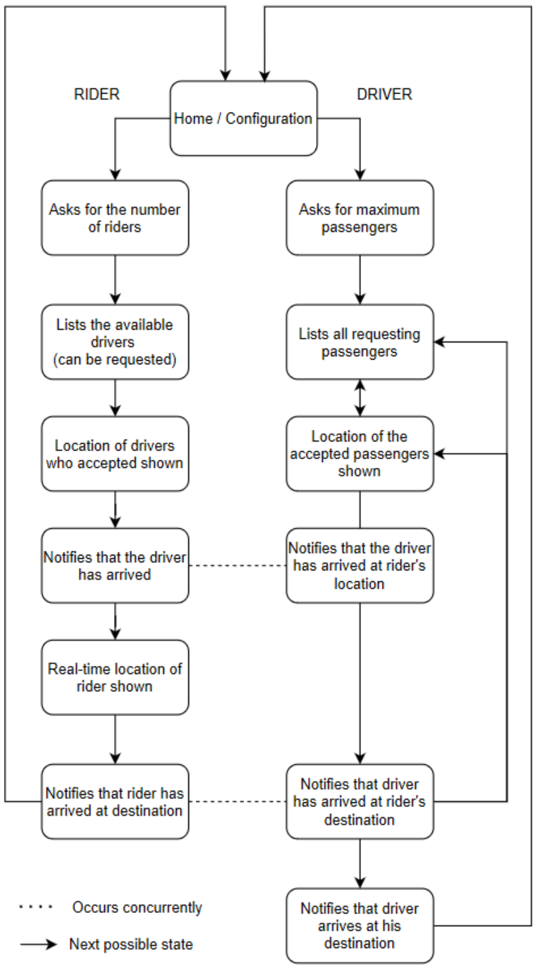
RIDER
Home / Configuration
DRIVER
Asks for the number of riders
Asks for maximum passengers
Lists the available drivers (can be requested)
Lists all requesting passengers
Location of drivers who accepted shown
Location of the accepted passengers shown
Notifies that the driver has arrived
Notifies that the driver has arrived at rider's location
Real-time location of rider shown
Notifies that rider has arrived at destination
Notifies that driver has arrived at rider's destination
Occurs concurrently
Next possible state
Notifies that driver arrives at his destination

Fig. 2. App state diagram

Say the rider option was chosen, the user will be asked one last choice indicating how many will be riding with him/her. After this, all drivers that are along the way of the rider will be listed as available. A rider can request a ride among these drivers. Once a driver accepts a request, the rider will be notified and the real-time location of the driver will be shown on the rider's screen. Once the driver arrives at his location, the rider will again be notified. The app will confirm that the rider is currently riding and the final notification happens when the rider arrives at the intended destination. After which the app returns to the home state.

On the other hand, if the user opted to be a driver, similar to a rider, the driver will also be asked one last option indicating the maximum number of passengers the driver intends to accommodate. After this, a screen containing all the requesting passengers are shown. The driver can choose to accept any of these requests. In the same screen, the accepted passengers are indicated and they real-time location shown, hence the bidirectional arrow in Fig. 2. When the driver arrives at a rider's location, the app will notify the driver. Every time a rider arrives at his/her set destination, the rider will be notified to drop off that rider. The final notification happens when the driver arrives at his/her set destination after which, the app returns to the home state ready for the next transaction.

The client mobile app's program flow is emphasized in this section as this will ultimately be what the users will be interacting with. However, it is also important to note that in order for all of the features in the client app to function correctly, a corresponding server app would also have to be built. In general, the server application will listen to the following events:

1. A user had just finished configuring the app for one's trip and therefore is either looking for a ride or looking to share a ride.
2. A rider requests for a list of eligible drivers
3. A rider requests a driver
4. A driver accepts a rider

These events will be triggered on the client app based on user action. See Appendix 2 for the corresponding action for each triggered event. Backend logic will sort the requests and serve the appropriate data depending on the set configuration.

C. *Software Tools*

Fig. 3. Flutter / Firebase stack

Two main technologies will be utilized for building the mobile app: Flutter and Firebase. Flutter is a relatively new technology that enables cross-platform app development that allows developers to create apps for Android, iOS and the web using one codebase. However, for the purposes of this study, the mobile app will be optimized for the Android platform (see *Scope and Limitations* for more details). Flutter will comprise most of the client application including the UI, the UI controller and the data interface. On the other hand, the backend will be built with the Firebase suite especially utilizing Firestore, a real-time document-based database. Firebase will handle all server requests from the client app and will manipulate the data as needed for the purposes of the app.

Services provided by Google Maps APIs such as Geocoding API, which converts addresses into geographic coordinates to be placed on a map, and converting geographic coordinates into human-readable address, and Place Autocomplete service in Places API which returns place predictions for subjects in motion, will be used for route planning and real-time navigation.

As for the version control system, the proponents will utilize Git. A private remote repository will be set up in GitHub with each proponent added as a collaborator. Furthermore, the project will adhere to the Gitflow process (Fig. 4) to keep the whole development experience clean, organized and professional.

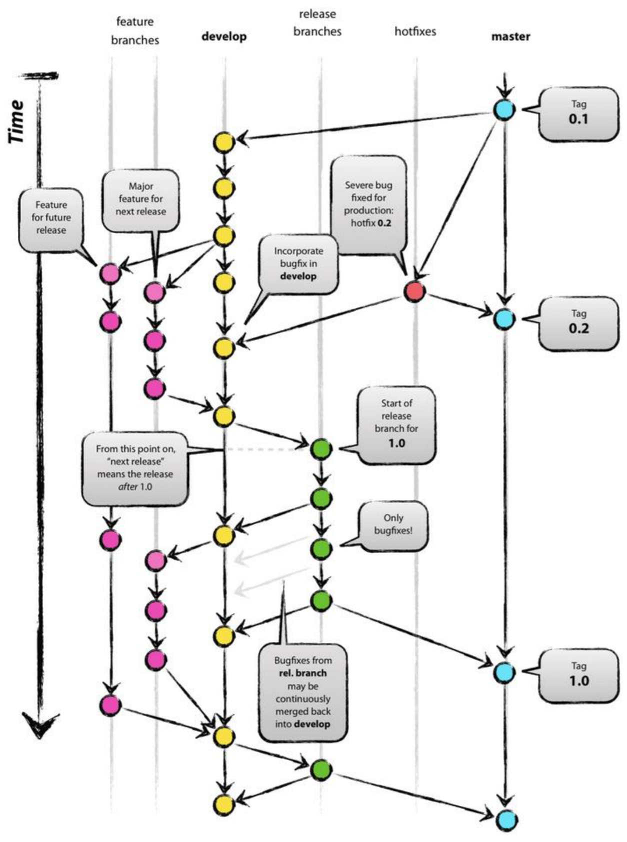
feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Fig. 4. The Gitflow process [10]

D. *Software Framework*

For the overall architecture of the mobile app, the researchers will adhere to the Model View Controller or the

MVC Architecture (Fig. 4). Adhering to this architecture will ensure testability and a robust organization of the mobile application's code. With this, there is a clear separation of concerns dividing the software code into three parts:

- Model, which typically reflects real-world things and hold the raw data of the application,
- View, which comprises the components responsible for directly interacting with the end user, and
- Controller, the logic behind the UI (View) and the controller of its state.

.

Appendix 1 shows the flowchart of the whole system. It includes all the important decision branches and on where to drive the client app. As for the system's corresponding server application, Appendix 2 shows all the events the server application listens to and the corresponding actions once a certain event is triggered. Appendix 3 shows a rough draft of how the user interface will be structured on the app. This is subject to further changes.

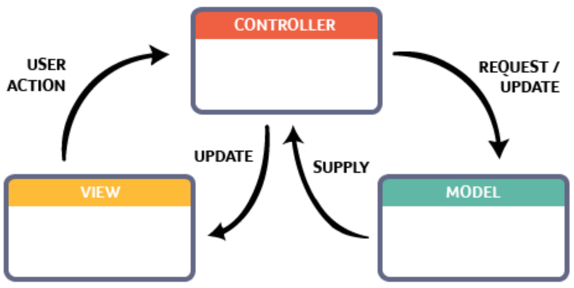

Fig. 5. MVC Architecture

E. Distance of a Point from a Line

Before diving into the core algorithms of the application, it's first important to note a basic yet very useful mathematical fact. Given two points that form a line, the minimum distance from this line to a certain point can be calculated with a formula derived from the area of a triangle. Recall that the formula for the area of a triangle is

$$A = \frac{1}{2}bh \tag{1}$$

The height h yields the shortest distance between the apex of a triangle (top vertex) and its base. By rearranging the formula, we get

$$h = \frac{2A}{b} \tag{2}$$

Therefore, given points (x_1, y_1) and (x_2, y_2) that form a line, the distance of this line from point (x_0, y_0) can be calculated using the formula:

$$minDistance = \frac{|(x0-x2)(y1-y0)-(x0-x1)(y2-y0)|}{\sqrt{(x2-x1)^2+(y2-y1)^2}} \tag{3}$$

where the numerator is twice the area of the triangle of vertices (x_0, y_0), (x_1, y_1) and (x_2, y_2) and the denominator is the base derived by calculating the distance between points (x_1, y_1) and (x_2, y_2). This formula will be used in the algorithm to calculate the distance between a coordinate (point) and a route segment (line) connected by two coordinates. Details of the derivation of this equation is at Appendix 4.

F. Core Algorithms

To build the system, two core algorithms have to be designed and coded using the software tools mentioned above. These algorithms will achieve the following:

- Return a list of compatible drivers to requesting riders i.e., drivers whose route is also along the rider's point of origin and destination.
- Calculate the number of points to be awarded both to drivers and passengers after a successful transaction.

The first and the most important algorithm that will be incorporated into the mobile application will be on how the system matches compatible drivers and riders to one another. Note that the algorithm only has to determine which drivers are compatible with a certain rider's set point of origin and destination, the other way around (determining which riders are compatible with a driver's route) is not necessary. The reason for this is best described in terms of the app's user interface. Remember that when a rider requests a ride, that rider is presented with drivers that are compatible to his or her set configuration. Hence, the pairing algorithm is required here. On the other hand, a driver only has to be presented with the riders that requested for his or her services.

As this algorithm will be coded with a language optimized for object-oriented programming (Dart), it is only fitting to describe the algorithm in an object-oriented perspective. Two objects are necessary for this algorithm, aptly named as *Driver* and *Rider*. The *Driver* will have a property that stores the *Route* which contains an ordered list of coordinates (latitude and longitude points) beginning with the Driver's starting location and ending with his/her destination. Remember that any point on Earth can be described in terms of latitude and longitude. Given this, a route can then be expressed as an interconnection of latitude and longitude points, effectively dividing a route into line segments. This concept is visualized in Fig. 6, which shows a theoretical route consisting a total of six route segments. See Appendix 5 for an example of a real-world route and the coordinates that comprise it as returned by the Google Maps Directions API.

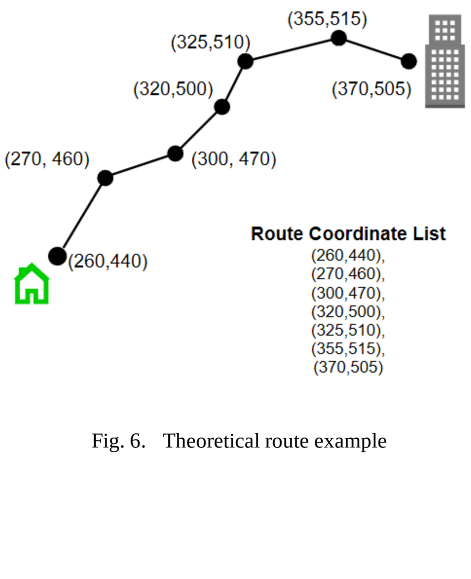

Fig. 6. Theoretical route example

The other properties that are necessary for the compatibility calculation between a Driver and a Rider will be the Rider's *startLoc* (starting location) and *endLoc* (ending location). An oversimplified diagram that shows what the algorithm does behind the scenes is shown in Figure 7 below.

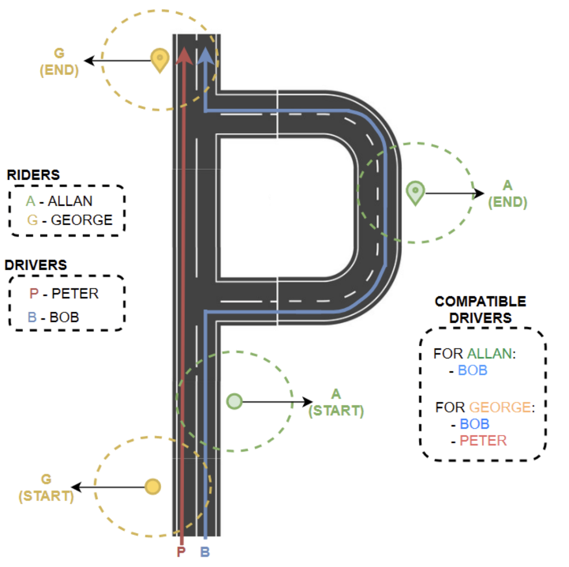

Fig. 7. Driver-Rider Pairing Visualized

Basically, the algorithm queries the database for available Drivers within a certain bounded area (in this case, Metro Cebu), whose Route property contains a route segment that is within walking distance of the Rider's *startLoc* and also within walking distance of the Rider's *endLoc*. The walking distance will be fixed at 50m. First, the algorithm iterates through each of a *Driver.Route's* ordered list of coordinates, then checks whether any of those coordinates are within 50m of the Rider's *startLoc*. If such coordinates exist, then the system proceeds to test the Route with respect to the *endLoc*. If there are none, the system determines the two nearest adjacent coordinates, instead. This is now where the formula derived from above comes to play. These two coordinates form a triangle with the *startLoc* at the apex. The natural curvature of the Earth can be ignored here as it's safe to say that the points aren't far enough for this to be a problem. The minimum distance between the *startLoc* and the two coordinates will be the height of the formed triangle. If this height measures to be within 50m then the system carries on with testing for the *endLoc*. If not, then the Driver is not compatible. The same process will be done for the *endLoc*, beginning with the testing if any coordinates are within 50m and if none are, the triangle formula. Finally, if both the *startLoc* and the *endLoc* are within 50m of the Route, then the Driver is compatible with the requesting Rider. If there are any Drivers that match with this query, they will be sent as JSON data containing details of the Drivers such as their name, photo, current location, and anything else required for by the app. This JSON data will be handled accordingly. A flowchart detailing this algorithm is shown at Appendix 6.

The second main algorithm to be designed is rather simple in comparison to the first one but equally as important. This second algorithm aims to perform a calculation of the number of points to be rewarded to both drivers and riders after a successful transaction i.e., a *Rider* was successfully transported from one place to another by a *Driver*. How points are calculated specifically for a *Rider* and a *Driver* are further discussed below.

In the driver's perspective, points to be awarded are based on how taxi fares are calculated. Four (4) points will be rewarded per 300 meters and another two (2) points per minute of travel per passenger. Unlike a taxi, there are no 'flag fall' points to be awarded. The travel duration that will be used for the calculation will also be through an estimation provided for by the Google Maps API. This way, the points to be rewarded are already determined at the start of the journey. These mimics the feature of current ridesharing apps that allow a passenger to have the fare already determined even before they set foot on the driver's vehicle.

As for riders, points are calculated simply to how many more riders they brought with them. Riders are awarded 10 points per person including themselves. So, if a rider decides to ride alone, he will be rewarded 10 points, if he rides with one more pal then he'll be rewarded 20 points, if there's three of them then 30 and so on. The extra riders that the ride hailer brought with him will not be awarded any points. Appendix 7 illustrates the algorithm for the point system.

G. *Authentication and Exclusivity*

As mentioned above, the mobile app will be exclusive to within the USC Community only. In order to enforce this, the system would have to find a way to differentiate a Carolinian from someone outside the community. Fortunately, there is one commodity that every certified USC member will surely have: a USC email. But having a USC email is not enough to ensure that one is indeed a member. What stops any outsider from blatantly searching for a valid USC email, say a professor's from the official USC website, and use it to impersonate a USC professional. To avoid this, the mobile app will require a user to actually authenticate themselves to make sure that they are the actual owner of the USC email account. To do this, the proponents would utilize one of the tools provided for by the Firebase suite: FireAuth. Behind the scenes, FireAuth utilizes the OAuth protocol. It is an open authorization framework that allows an app to practically outsource the authentication of a user securely. It is a widely recognized industry-standard as documented by Hardt (2012) in RFC 6749 [11]. OAuth is what's at work whenever one presses the "Sign in with Facebook" or the "Sign in with Google" button to register to a certain app. The task of authenticating a user is given to a trusted authentication provider such as Facebook, Google or Microsoft. Now, going back in context, remember that each USC account is encompassed in an organizational Google Workspace bought and managed by the USC administration. This essentially renders each USC account to be a valid Google account belonging to the same organization: The University of San Carlos. The mobile app can then therefore outsource the authentication to Google to validate that it indeed is a valid and existing account. Of course, that would also mean inputting one's password to the said account. Once a user is authenticated, data regarding the user will then be sent to the mobile app. Data that is willingly and knowingly shared by the user such as the user's profile picture, full name, email and even the organization that the account belongs to (if any). Given this, if a user of the system had successfully authenticated themselves by signing in with Google, and the account is a verified USC account, i.e., belongs to the "University of San Carlos" organization as returned by the OAuth response, then that user is eligible to use the service. On the other hand, if any of the aforementioned authentication checks fails, the user is automatically signed out of the application.

H. Testing Plan

As the application is built from the ground up, it will also be subjected to the following stages of tests:

1. Unit Testing
2. Integration Testing
3. Acceptance Testing

For the first two stages of tests, unit and integration testing, the built-in testing tools provided by the Flutter framework will be used for writing the tests. Unit tests will be written for all the models and classes in the application even at the early stages of development. Once a working app is built wherein the minimum features specified are met, the application will then be subjected to integration testing. Here, the proponents will still use the tools provided for by the Flutter framework to test the mobile app 'as a whole', testing it as to how everything integrates with one another including the individual screens, models, buttons, network requests and other relevant components of the app. Finally, the app will be subjected to acceptance testing. This stage is divided into two phases: *alpha* and *beta*. During the *alpha* acceptance testing stage, the mobile application will be deployed to the Google Play Store (alpha release) and the proponents would try to install and use the app themselves and test it as to how it fares in the 'real-world'. Each would take turns as to who will act as a Driver, and who will act as Riders. The designated Driver will drive around in a vehicle while Riders will attempt to hail a Driver through the mobile app. The second phase of the acceptance testing is the beta stage. The mobile application will now be released to a wider set of users (beta release). This is now where the proponents would invite interested members of the USC community to install, use and provide feedback regarding the mobile app.

I. In-App Feedback System

During the beta testing phase of integration testing, the main objective is to uncover as many bugs or usability issues as possible in controlled setting. In collecting user-feedbacks during this stage, an in-app feedback system will be implemented on the application (*see Appendix 8 for in-app user feedback UI)*. Conveying users' feedback especially during the early stages of a mobile application can speed up the development process [12], thus helping developers find potential issues and make improvements based on the insights given. Data gathered will be stored at the Firebase database and will be used to improve current release and help create priorities for future releases, this ensures the technology roadmap and planning are as optimal as they should be.

References

[1] Katrina Escalona, (September 2017) How the Jeepney Became a Filipino National Symbol from https://theculturetrip.com/asia/philippines/articles/how-the-jeepney-became-a-filipino-national-symbol/.

[2] Mitchelle Palaubsanon (January 2021) 80 more modern jeepneys to ply Metro Cebu routes from https://www.philstar.com/the-freeman/cebu-news/2021/01/11/2069672/80-more-modern-jeepneys-ply-metro-cebu-routes.

[3] Brian Caulfield. 2009. Transportation Research Part D: Transport and Environment. Elsevier. https://www.sciencedirect.com/science/article/abs/pii/S1361920909000893.

[4] Shaheen, S., Cohen, A., & Bayen, A. (2018). The Benefits of Carpooling. UC Berkeley: Transportation Sustainability Research Center. http://dx.doi.org/10.7922/G2DZ06GF Retrieved from https://escholarship.org/uc/item/7jx6z631

[5] Ma, S., & Zheng, Y. (2017, September). Real Time City Scale Taxi Ridesharing. Microsoft. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ridesharing-TKDE-yuzheng.pdf

[6] Khunger, S., & Carr, D. (2014, April 1). Real-Time Ride Share System. United States Patent. https://patentimages.storage.googleapis.com/b8/0a/a9/b3fb80874a516f/US8688532.pdf

[7] Agatz, N., Erera, A., & Savelsbergh, M. (2011). Dynamic Ride-Sharing: A Simulation Study in Metro Atlanta. Science Direct. https://www.sciencedirect.com/science/article/pii/S1877042811010895

[8] Limpin, L. (2018, June 28). Factors That Affect Filipinos' Mentality On Ride Sharing. Workshop on Computer Science and Engineering. http://www.wcse.org/WCSE_2018/W066.pdf

[9] Vayouphack, S. (2020). Ridesharing in Developing Countries: Perspectives from India and Thailand. AUT. https://openrepository.aut.ac.nz/bitstream/handle/10292/13296/Simmavanh%27s%20Dissertation.pdf?sequence=1&isAllowed=y

[10] Driessen, V. (2010). A Successful Git Branching Model. https://nvie.com/posts/a-successful-git-branching-model/

[11] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, https://tools.ietf.org/html/rfc6749

[12] M. Litwin, *Mobile App Survey: Complete Guide with Question Examples*, Survicate. Accessed on: April 9, 2021. [Online]. Available: https://survicate.com/mobile-app-survey/mobile-app-survey-questions/