The University of Hong Kong

Final Year Project

Final Group Report

Building an Easy-to-use Ride-sharing App for HK

Supervisor: Dr. Huang Z.Y.

Student Name: Leung Hon Man, Anthony (3035278400)

Lau Chi Ho, Eric (3035326049)

Lau Chun Yin, Elven (3035294715)

Account: FYP18028

April 19, 2019

Abstract

In recent years, ride-sharing has been an alternative to public transportation or private car rental for commuters in countries around the world. It is believed that ride-sharing is a way to alleviate the traffic congestion and reduce the air pollution caused by vehicles [1]. However, Hong Kong, as a world-class city which is suffering from traffic congestion, has not exploited the benefits of ride-sharing. One of the reasons may be that there is no ride-sharing mobile application being extensively spread in Hong Kong until now.

In an effort to increase the popularity of ride-sharing in Hong Kong, this project aims to design and implement a fully featured handy ride sharing mobile application and related web services for people in Hong Kong, which adapting matching algorithms from cutting-edge research. Two algorithms are implemented and evaluated using some real-world test cases and a virtual grid world simulator in this project.

This project was conducted in group by three students in the Computer Science Department of The University of Hong Kong, they are Leung Hon Man (Anthony), Lau Chi Ho (Eric) and Lau Chun Yin (Elven).

Acknowledgments

We would like to express my greatest appreciation to Dr. Huang Z.Y. for his guide on this project, to Dr. Hubert T.H. Chan for his advices and assessment as a second examiner, to Dr. Joanna C.Y. Lee for the help in writing the academic report. This project would not have been possible without their help.

Furthermore, we would like to express my special gratitude to the Computer Science Department and the University of Hong Kong for offering me the world-class computer science curriculum and comprehensive learning facilities, which equipped me with a strong academic foundation and made the completion of the project possible.

Table of Contents

List of Figures

List of Tables

Abbreviations

DBMS	Database Management System
DevOps	Software Development and Information Technology Operations
FTP	File Transfer Protocol
ILP	Integer Linear Programming
JWT	JSON Web Tokens
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
UI	User Interface
UX	User Experience

1. Introduction

1.1 Motivation

Since Hong Kong is one of the densest cities in the world, traffic congestion has been a problem for commuters for a long time; despite the transportation network in Hong Kong is highly efficient and fully exploited, the traffic congestion problem is escalating [2]. However, ride-sharing may be a solution to the dilemma of Hong Kong's transportation. Studies demonstrate that adapting ride-sharing can significantly reduce the demand for vehicles, and thus it is a potential solution to traffic congestion [1, pp. 25-26]. Ride-sharing can reduce the number of empty seats in private vehicles and thus can alleviate traffic congestion.

Ride-sharing has been available in Hong Kong in recent years, but the usage of ride-sharing is not popular. A survey illustrates that there are only around 33 per cent non-car owners have ever used ride-sharing services before [3, pp. 40-41]. One possible reason of this situation is the fact that there is no free-of-charge, fully featured, safe, and easy-to-use ride-sharing mobile application which can encourage public to attempt the ride-sharing services for the first time in Hong Kong.

1.2 Concepts and Definitions

The term ride-sharing can be ambiguous since it is not stated in the laws of Hong Kong [3]. In this section, the definitions of ride-sharing and related concepts will be clarified.

Ride-sharing (or car-sharing) refers to any means of transportation in which a car owner (an individual or a company) provides a shared or non-shared ride to another individual or a small

group of people [3]. In this project, only two types of ride-sharing are concerned, they are car hailing and carpooling.

Car hailing (or ride-hailing) refers to a service that a driver with or without a Taxi license provides a ride to an individual or a small group of people to their destination with or without prior appointment [3]. Traditional Taxi service is one of the car hailing by definition, other examples are UberX and Lyft which use digital the platform to assign the ride requests to drivers.

Carpooling refers to a service which is the same as car hailing except that the driver will pick up new passengers with a similar direction during the ride if the capacity of the car allows.

Ride request (or simply request) refers to the requests that submitted by a mobile application user (passenger) who want to travel from origin to destination.

Trip (or ride, ongoing ride, matched ride, matched request) refers to the mission of the driver, appointed by the system after matching a passenger's ride request to the driver, to travel from one or more origins to one or more destinations.

1.3 Previous Works

Many research related to ride-sharing have been conducted and attempted to provide solutions to the dynamic matching problem appearing in the online digital platform of ride-sharing. There have been at least 38 academic papers related to ride-sharing since 2006, and most of the papers presented matching algorithms which make it possible to assign ride requests to suitable drivers instantaneously [4]. Two academic papers ([5] and [6]) published in 2018 were studied, but the content is not directly used in this project. A paper published in 2017 with the title "On-demand

high-capacity ride-sharing via dynamic trip-vehicle assignment" is the foundation of the matching algorithms in this project [7]. And the concept "shareability" used in the algorithm introduced by [7] is originated by [8] which is also studied.

There are several ride-sharing mobile applications available on the market such as Uber, LYFT, GRAB, etc. However, it seems the source code and technical details of those applications are not shared with the public. On the other hand, there are no open-source ride-sharing mobile applications are observed at this moment.

1.4 Objective and Scope

This project aims to deliver a free-of-charge, full-featured, safe, and easy-to-use ride-sharing digital platform to people in Hong Kong for the purpose of improving the transportation system. Only mobile devices are chosen as the medium of this digital platform. Therefore, the deliverables consist of two components, a front-end mobile application and back-end web services.

For the front-end mobile application, both Android and iOS platforms are targeted, and both car hailing and carpooling services will be available. The application will also include features of other ride-sharing mobile applications, such as users account management (i.e. login, signup, password reset, etc.), map view for easy locating the driver or passenger, and in-app communications among users, except the e-payment system.

The back-end web services will have all necessary functionalities (i.e. web server, database, and matching algorithms) that support the features presenting in the mobile application. In particular, two matching algorithms for rider-driver pairing will be implemented, analyzed, and evaluated.

1.5 Outline

This report will first present the design, implementation details, and testing approach of the ride-sharing mobile application and the back-end web server in the methodology section. Next, a review of the difficulties, and limitations of this project will be delivered. Finally, in the conclusion section, a summary of this report and a future plan will be made.

2. Methodology

In this section, the abstract-level design and implementation details of the ride-sharing digital platform are introduced.

2.1 Problem Definition and Algorithms

The definition of ride-sharing problem of this project aims to solve is a real-time driver-passenger matching problem. In every timeframe (let's say 6 seconds), the web server will receive a number of vehicles and ride requests which are passed to the matching engine for matching. The matching will be conducted by the match engine at an interval of I (let's say 60 seconds). The matching engine will first transform those requests and available vehicles to a set of Requests $R = \{r_1, r_2, \dots, r_n\}$ (each r is a tuple [o,d,t] where o is the origin of the request, d is the destination of the request and t is the request time) and a set of vehicles $V = \{v_1, v_2, \dots, v_m\}$ (each v is a tuple [l,p] where l is the location of the vehicle and p is a list of request assigned to v) respectively. Then, an optimal assignment of V and R will be computed, where V may have already carried a list of passenger p. The assignment is a set of mapping M between R and V(e.g. $M = \{ r_1, v_2 \}$) such that the cost function C is minimized, and a set of constraints Z is satisfied. The input and output format are shown as following:

Input:

Requests R

Vehicles V

Contraints Z

Output:

Requests R' that cannot be matched where $R' \subseteq R$

Match results $M = \{ m_1, m_2, ..., m_n \}$ where $m = (r, v)$

This project implemented two matching algorithms, an original greedy algorithm and an advanced algorithm from the academic paper "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment"[7] (Dynamic Assignment for short). Two algorithms are evaluated using both real-world test cases and a grid world simulator.

Algorithm 1: Greedy Algorithm

For the original greedy algorithm, the pseudo code is shown as below:

Steps:

1. $C \leftarrow \{ (r, v) \mid r \in R \wedge v \in V \}$
2. Sort C in asc order by $distance(r.o\ ,\ v.l)$
3. $M \leftarrow \{ c : c \in C \wedge r\ and\ v\ satisfy\ Z\ where\ c = (r, v) \}$
4. $R' \leftarrow \{ r : r \in R \wedge r \notin \{ r' : (r', v') \in M \} \}$
5. Return M, R'

The constraints set Z includes three constraints, 1. waiting time must be less than certain minutes, 2. the empty seat in the drivers must be greater than zero, and 3. the most important constraint is that requests must be sharable with the trips the driver carrying.

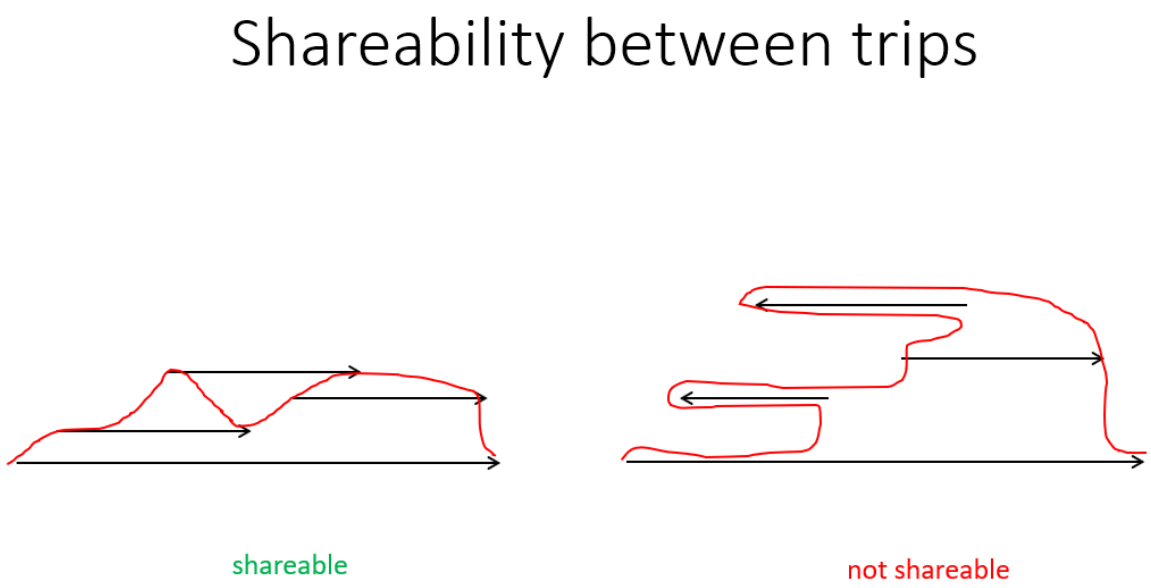
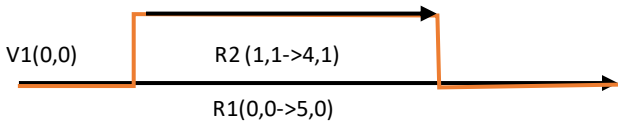


Figure 1 - Shareability between trips

The concept of whether a request and a driver have shareability or not is proposed by [8]. Trips are sharable if their best routes are shorter than their sum of individual distance between pickup and drop-off locations [8, p.13292]. In figure 1, the trips in left side are shareable because their best shared routes (red line) are shorter than the sum of their individual lengths (dark arrows). In contrast, for the trips in right, there does not exist a shared route (not a concatenated routes of individual trip) that is shorter than the sum of the individual lengths of trips, so those trips are not shareable. [7] introduced a variant that decides the shareability between a ride request and a vehicle, which also consider the current location of the vehicle when calculating the best shared route and individual route.

Algorithm 2: Dynamic Assignment Algorithm

The three terms are used in this subsection, they are **delay, waiting distance**, and **matched route**. It is better to use the following case to illustrate them.



In this case, there are two requests (R1 and R2) and one empty vehicle V1. V1 has a position of (0,0). R1 is a request from 0,0 to 5,0 and R2 is a request from 1,1 to 4,1.

The **matched route** (trip) is the orange line on the graph. Its route is 0,0->0,1->1,1->4,1->4,0->5,0 and has a distance of 7 units.

Delay is the sum of delay distance of the requests in a trip. Delay distance means the distance in the matched route that the passengers experienced in the ride minus the distance of the best route to serve the requests alone. In the case above, R1 will experience 7-5= 2 units distance delay but R2 doesn't experience any delay since R2 gets on the car at 1,1 and gets off the car at 3,1 which is the same as the best route to serve the R2 alone. The sum of delay is 2+0=2.

Waiting distance means the waiting distance each request experienced in a trip. In the case above, R1 didn't experience any delay as V1 have the same position with the origin of R1 but R2 experience 2 units delay distance as matched route is 0,0->0,1->1,1 and need to go 2 units distance to the origin of R2. In the algorithm, we only need to consider whether the largest waiting distance of a request in the trip violated the constraint or not because if the request which has the largest waiting distance in a trip does not violate the waiting distance constraint, other does not too.

Below are the set of constraints Z we will consider in the algorithm:

1. For each request R, the maximum waiting distance cannot be larger than Ω, which is a constant
2. For each request R, the maximum delay distance cannot be larger than Δ, which is a constant
3. For each vehicle, the maximum passengers cannot exceed the capacity of the car.
4. Rides will only be matched if they are shareable. Where the term shareability is same as that described in Algorithm 1.

The cost function C computes the sum of delay (δ_r) of the trips (both previous requests but still on the car and new assigned requests are considered) and sum of constant c_{ko} for the unassigned request. Formally,

$$C(\sum) = \sum_{v \in V} \sum_{r \in P} \delta_r + \sum_{r \in R_{ok}} \delta_r + \sum_{r \in R_{ko}} c_{ko}.$$

Where R_{ok} denote set of newly assigned requests R_{ko} denote set of unassigned requests.

Algorithm Steps:

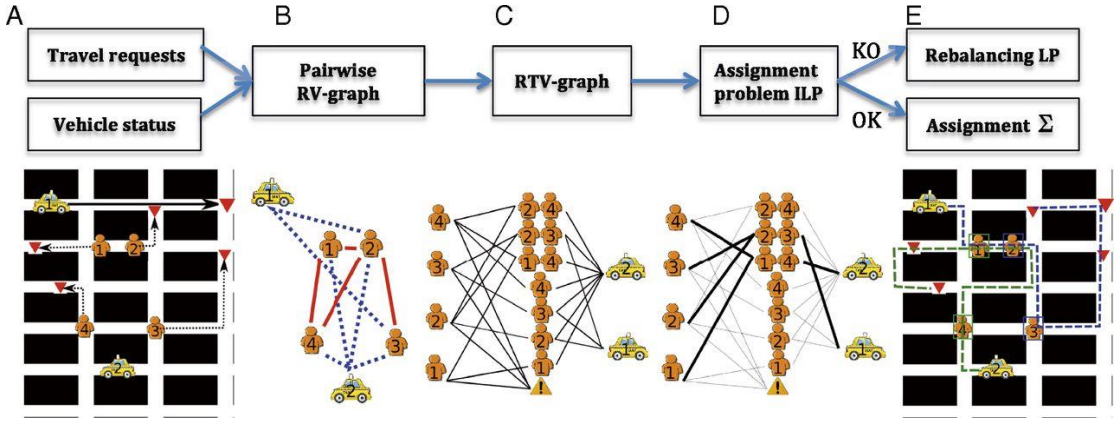


Figure 2 - Steps of Dynamic Assignment algorithms (capture from [7]). V1 is carrying a passenger to a destination and V2 is an empty car. Also, there are four different request 1,2,3,4.

The algorithm will receive a number of requests R and vehicle status V as an input (fig. 2A). The matching algorithm is divided into four parts – pairwise request-vehicle shareability graph (RV-

graph) (fig. 2B), feasible trip of vehicles serving requests graph (RTV-graph) (fig. 2C), solving ILP for assign the best trips (fig. 2D), and rebalance the remaining idle vehicles and unassigned requests (fig. 2E) [7].

In this project, we will set our constrain Z. The delay distance is set to 1km and maximum capacity of 2 requests in order to enhance user experience and increase the performance.

First, RV graph has two types of edge (Step B). The first type of edge is a vehicle connects to a request. A vehicle will connect to a request if the vehicle can serve the request while does not violate the constraints Z (Waiting time, delay, capacity, shareability). Note that a vehicle can be empty or already have passengers in the car. The second type of edge is a request connects to a request. Two requests are connected if they are shareable by a virtual car start from either request. The use of this step is to reduce the trips generated in the next step.

The output from the RV graph (2 types of edge) will be passed to the next step, which is forming RTV graph (Graph of candidate trips and pick-ups) (Step C). This step is to select all possible trips that satisfied the constraints Z. In order to reduce the possible cases for finding possible trips, two rules will be applied:

- Request will be added to a vehicle to make a new trip if the request vehicle pair is in the RV graph
- Request should form pairs with one of the requests in the same vehicle which can be found in the RV graph.

After all trips are generated, the trips will be checked incrementally to find the best route of the trips and the corresponding delay will be calculated and will be checked if it satisfied the constraints Z or not. The trips that satisfied the constraints Z will be append to the possible trip list (RTV-graph) with the delay to be the output. Note that a vehicle and a request can be in multiple trips. In order to save computation power, trips will only be checked if all the sub-trips in the trips are checked and satisfied the constraints (i.e.: in the possible trip list). For example, a trip $t =$

$\{v_1, r_1, r_2\}$ will only be checked if (v_1, r_1) and (v_1, r_2) are both in the possible trip list (RTV-graph).

After this step is the assignment step. There are two methods running in parallels. The first method is greedy assignment in respect to the delay and it may not output the optimal solution. The second method is using ILP (integer linear programming) and it will output the optimal solution. In the ILP method, a binary variable $E_{i,j} \in \{0,1\}$ is introduced. If $E_{i,j} = 1$, it means vehicle j is assigned to trip i. We need to find the values of all $E_{i,j}$ such that the cost function will return a minimum value and also satisfy each request and vehicle can only be assigned once only. The reason why 2 methods are running in parallels is because sometimes the ILP assignment will take a long time to compute the optimal assignment and exceed the time limit. In this case, we need to combine the partial result of ILP with the result of the greedy assignment to return a suboptimal result. However, the paper didn't mention how to combine the result and our ILP library (Pulp) doesn't support partial solution, we only run the ILP to give the optimal solution in the testing part. Note that we have also implemented the greedy assignment.

At last, the propose of rebalance (Step E) is to balance the number of idle vehicles fleet because there may occur non-random requests. For example, there will be many requests with similar route and the idle vehicles are far away from the requests' origin. In this case, the requests are not satisfied the constrain. Therefore, we will try to match those idle vehicles V_{idle} and unassigned requests R_{ko} in this step. This step will check for idle vehicles V_{idle} and unassigned requests R_{ko} and compute the minimum number of sum of waiting distance $\sum_{v \in V_{idle}, r \in R_{ko}} \delta^{w}_{v,r} y_{v,r} = \min(V_{idle}, R_{ko})$ where $y_{v,r} \in \{0, 1\}$. This rebalancing will stop until there are no unassigned rides or idle vehicles However, in our project, this step will not be run because our testing location is randomly generated, and the request can match with vehicle without rebalancing.

2.2 System Design

This subsection explains the overall architecture of the system, data flow diagram among all the parties, the network communication method used in this system, and the main features of the mobile applications.

2.2.1 Overall Architecture

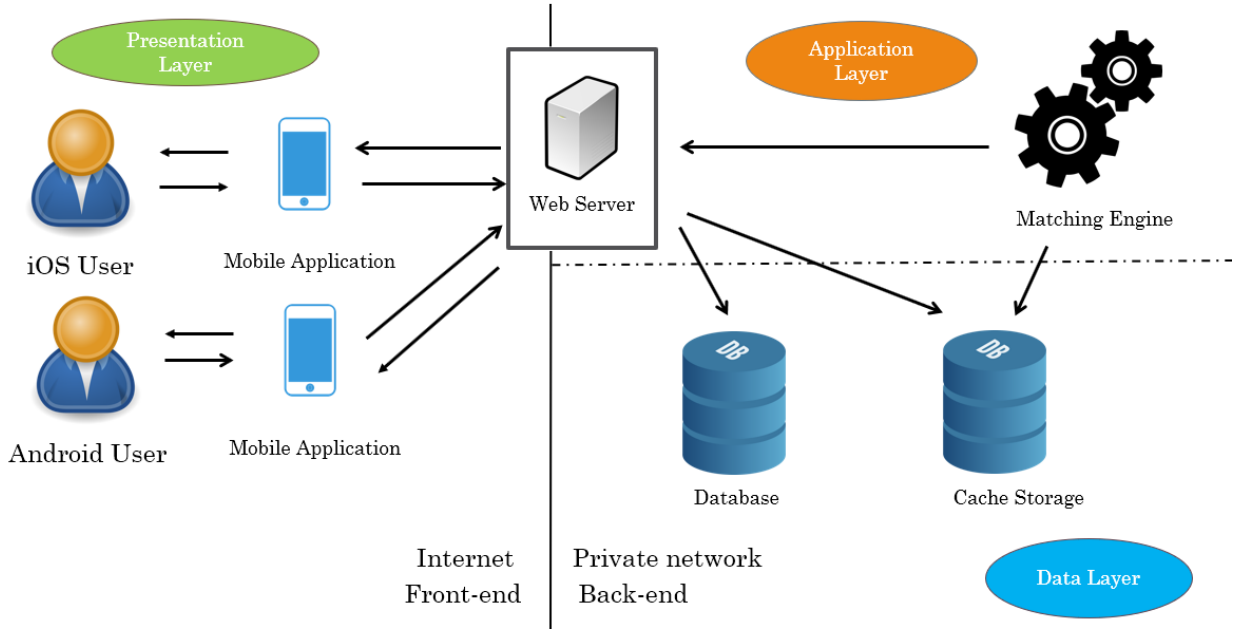


Figure 3 - System Architecture of Ride-sharing Digital Platform

The ride-sharing digital platform consists of two major parts as demonstrated in Figure 3, the front-end side and the back-end side.

On the front-end side, the mobile application in each user's mobile device directly interacts with the user to provide ride-sharing services and communicates with the web server on the back-end side to deliver users' information or ride requests.

On the back-end side, there are four running services, web server, matching engine, database, and cache storage which provide different functionalities and communicate with each other within the private network. This architecture, also known as the 3-tier architecture, is a common approach in IT industry. It separates the responsibilities into three categories, presentation layer, application layer, and data layer, with the benefits that the system are more flexible, scalable, and higher performance [9]. Mapping the architecture of this project to the 3-tier architecture, the mobile application is in the presentation layer, and the web server and matching engine are classified as application layer while the cache storage and database are in the data layer.

There are two reasons to separate matching engine from web server. First, matching is performed at specified intervals, for example, every 30 seconds, and it is not triggered by external event, therefore, the nature of matching engine is different from that of web server, they should not be in the same program (or process). Second, it is a good practice to decouple the matching engine from the system (which means no other components in the system knows its existence) if there are more than one version of matching algorithms. Because different matching algorithms may require different information from database and different implementation of the matching engine, separating matching engine from the system make the switch of matching algorithms easier and do not require restart of web server. To support the communications between web server and matching engine, the cache storage is used as a global, atomic, in-RAM storage which allow high speed read/write operations to two data structures, list and hash table.

2.2.2 Data Flow Diagram

Figure 4 shows the ways different parties communications with each other in and outside the system and describe the input and output of each parties. Note that the cache storage is divided into 3 sub-parts, ongoing ride set which stores the details of all matched rides, online driver set which stores the online drivers' locations, and Ride Request Queue that stores the requests waiting to be matched.

The users only send request to the web server when they want to have a ride, while the drivers update their location to the web server at pre-defiled time interval. Match engine is trigger regularly and it will read all the information it needs to perform matching, and it will send the match results to web server when the matching is done. When web server received the match results, it does not notify the end-users directly, but redirects the result to push notifications services providers (APNS for iOS, Google Cloud Messaging for Android). The push notifications services providers will send notification to users. At the same time, when passengers inquire all the driver locations or drivers inquire all the pending ride requests, the web server is able to provide the timely information after received the match results from matching engine.

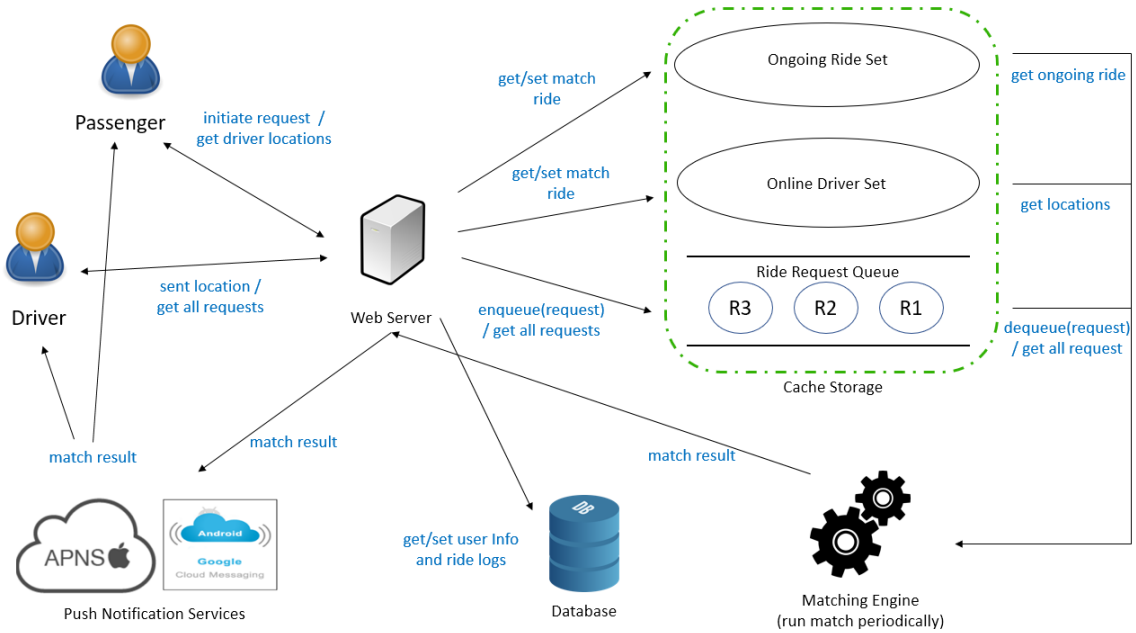


Figure 4 - Data Flow Diagram of the System

2.2.3 Network Communication Method

The communication method for the mobile application, web server, matching engine, and database will be REST. REST is the most common communication style of web services in which one computer uses a stateless network protocol like HTTP to send data to another computer. Another popular communication method is SOAP, which is more complicated and uses not only HTTP but also SMTP and FTP. The differences between REST and SOAP are shown in Table 1. It is illustrated that REST is much easier to implement, has better performance, involves fewer technologies and thus has lower learning overhead [10]. Therefore, REST is chosen as the only communication style for this project.

#	SOAP	REST
1	A XML-based message protocol	An architectural style protocol
2	Uses WSDL for communication between consumer and provider	Uses XML or JSON to send and receive data
3	Invokes services by calling RPC method	Simply calls services via URL path
4	Does not return human readable result	Result is readable which is just plain XML or JSON
5	Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.	Transfer is over HTTP only
6	JavaScript can call SOAP, but it is difficult to implement	Easy to call from JavaScript
7	Performance is not great compared to REST	Performance is much better compared to SOAP - less CPU intensive, leaner code etc.

Table 1 - Comparison Between SOAP and REST Communication Method [10]

2.2.4 Main Features of the Mobile Application

In the user perspective, there are in total 10 main features in the mobile application:

1. Account management for users (i.e. sign up, log in, change and reset password)
2. Profile management (i.e. upload avatar images, edit nickname, car plate, contact no., etc.)
3. Passengers can see all the driver locations on the map view
4. Passengers can find a driver in real time
5. Drivers can see all the ride requests on the map view
6. Drivers can find riders to initiate a ride
7. Drivers can continually match new riders with similar direction during a ride
8. Notification to users when match found
9. Match History
10. Providing contact method to matched passengers or drivers

2.3 Implementation

This subsection provides the details of the current implementation of the design demonstrated in the last subsection (i.e. section 2.1), including project structures, technologies choices, UI of the mobile application and user authentication process.

2.3.1 Project Structures

The source code is divided into two Git repositories, "Ridesharing-App-For-HK" for the code related to the mobile application development and "Ridesharing-App-For-HK-Back-End" for the code related to web server, database schema, matching engine/algorithms, and scripts for integration test. Those repositories is resided in GitHub for collaboration and easy access to development history, the link is https://github.com/eric19960304/Ridesharing-App-For-HK and https://github.com/eric19960304/Ridesharing-App-For-HK-Back-End respectively.

The project structure for "Ridesharing-App-For-HK" is as following:

- README.md
    - some important information about the repository
- assets
    - for storing the images materials
- App.js
    - project entry point
- config.js
    - for configuring the web server url
- src/boot
    - scripts here is executed when the app is first opened
- src/helpers

    - storing utility modules that can be used at various other modules
- src/screens
    - storing pages' layout and logic
- src/theme
    - storing the code for styling
- … (can ignore)

The project structure for " Ridesharing-App-For-HK-Back-End" is as following:

- README.md
    - some important information about the repository
- images
    - for storing the images materials
- app.js
    - project entry point
- config.js
    - for configuring various parameters of the web server
- src/controllers
    - storing end-point logic
- src/db
    - storing database clients
- src/helpers
    - storing utility modules that can be used at various other modules
- src/models
    - storing database schema
- src/middlewares
    - storing reusable components that can be used in controllers
- /matchEngine
    - code for matching algorithms, matching engine, and simulator
- /testing
    - code for integration test

- … (can ignore)

The list of end-point of the web server is shown in table 2:

End-point	HTTP Method	Remark
/auth/login	POST	
/auth/signup	POST	
/auth/signup/activate/:token	GET	A temporary account activation link is sent to user's email after login (:token is random generated UUID4 string)
/auth/reset-password/request	POST	
/auth/reset-password/:token	GET	A temporary link for user to fill in reset password form in their web browser
/auth/reset-password/:token	POST	Same link as previous but to receive the reset password form data submitted by users.
/api/secret/google-map-api-key	POST	For mobile app to get google API key to use map services, it will be expensive if there are many users.
/api/driver/update	POST	For drivers to update their location and get all the pending/matched request origin and destination.
/api/driver/get-all-drivers-location	POST	For passengers to get all/matched the driver location.
/api/rider/real-time-ride-request	POST	
/api/user/edit-profile	POST	
/api/user/edit-profile-with-password	POST	
/api/user/push-token	POST	For push notification to work.
/api/user/unread-messages-count	POST	For users to get how many unread system messages.

/notify-match-result/real-time-ride	POST	For internal used only. When matching engine found mapping, it will send the result to this end-point.

Table 2 - End-points of the Web Server

The mobile application is on development stage and requires some work to make it available in Apple App Store and Android Play Store. The instructions to run the mobile app, web server, matching engine, simulator, and testing are written in README.md file in root directory for each repository.

2.3.2 Technology Choices

	Mobile Application	**Web Server**	**Database**	**DevOps**
Programming Language	JavaScript	JavaScript	NoSQL	N/A
Frameworks (or libraries)	React Native, NativeBase	Node.js, Express.js	MongoDB	N/A
Tools	Expo	Nodejs Development Server	Robo 3T (DB browser)	Google Cloud, Git, GitHub

Table 3 - List of programming languages, frameworks, and tools used in development

There are many tools and technologies available for implementing the design described in the previous subsection, yet, not all of them are suitable for this project. Thus, choices have been made carefully to ensure the development process will run smoothly, and no compatibility conflict among software libraries will occur. The selected programming languages, frameworks, and tools for this project are shown in table 3.

For the web server, JavaScript is used as the programming language with Node.js runtime environment. JavaScript is a popular choice for building web servers for mobile applications, because it has better performance, lower learning curve, more support on the Internet, compared to other choices like Python and Java.

For the database, MongoDB is chosen to be the DBMS. Compared to other available DBMSs such as PostgreSQL, MySQL, Oracle NoSQL Database, etc., MongoDB is much more suitable for this project since it is easy to learn, flexible, stable, and free-to-use.

For the mobile application, JavaScript will be the programming language with React Native and NativeBase framework. Using JavaScript for mobile application development is beneficial. Not only the development skill set can be aligned with the web server development to reduce the learning cost, but also that it is supported for both Android and iOS mobile platform development, and thus only one code base needs to be written.

Since the development of matching engine can only be done after fully understood the matching algorithms mentioned in section 1.3, there is no implementation detail for the matching engine yet, attributed to the academic papers regarding the matching algorithms are not fully explored.

2.3.3 UI/UX of the App

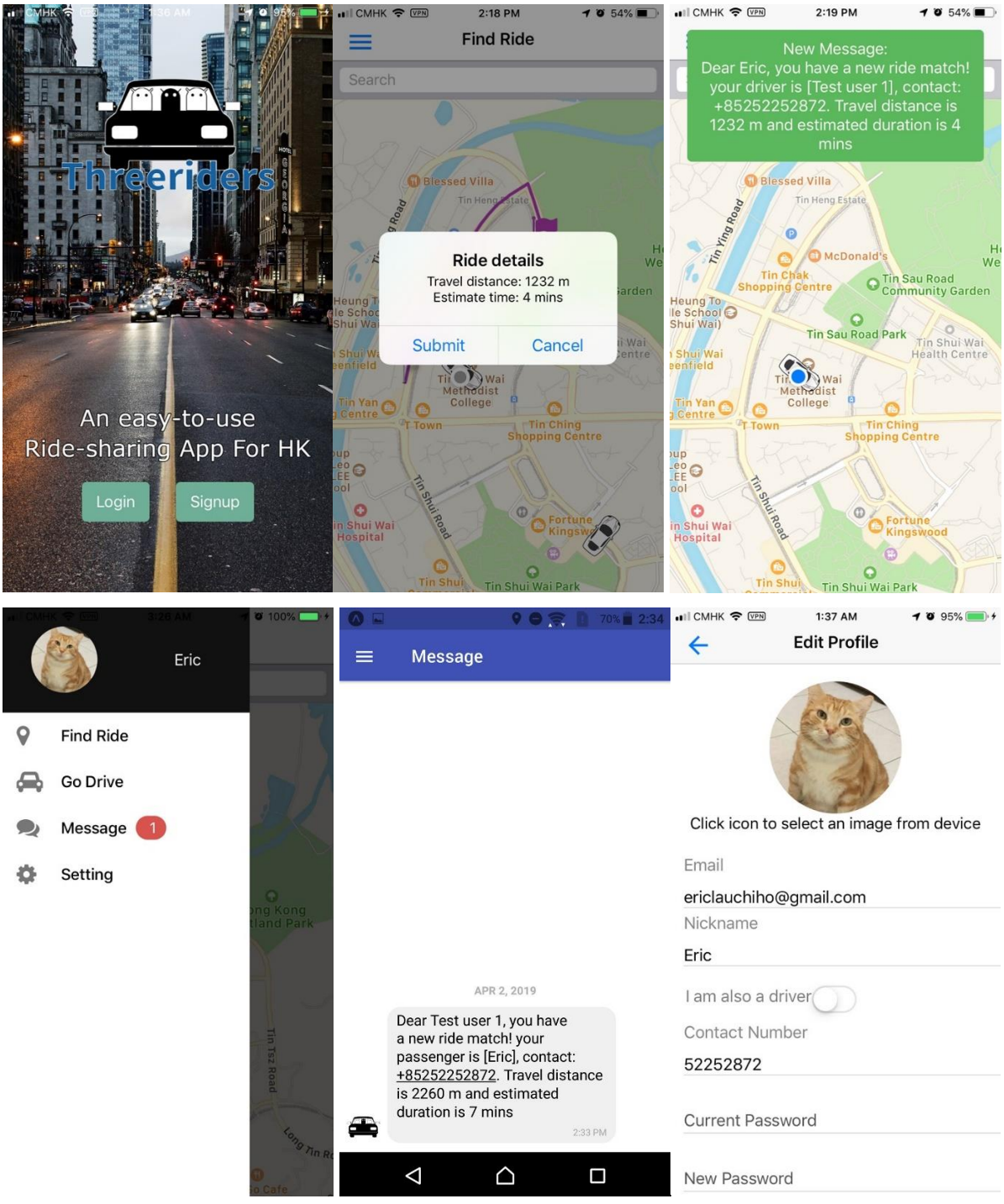


Figure 5 - Screenshots of the Mobile Application

Figure 5 displayed some screenshot of the UI of mobile application, including initial welcome page, find ride page (submit requests, notification of getting matched), system messaging, and edit profile page.

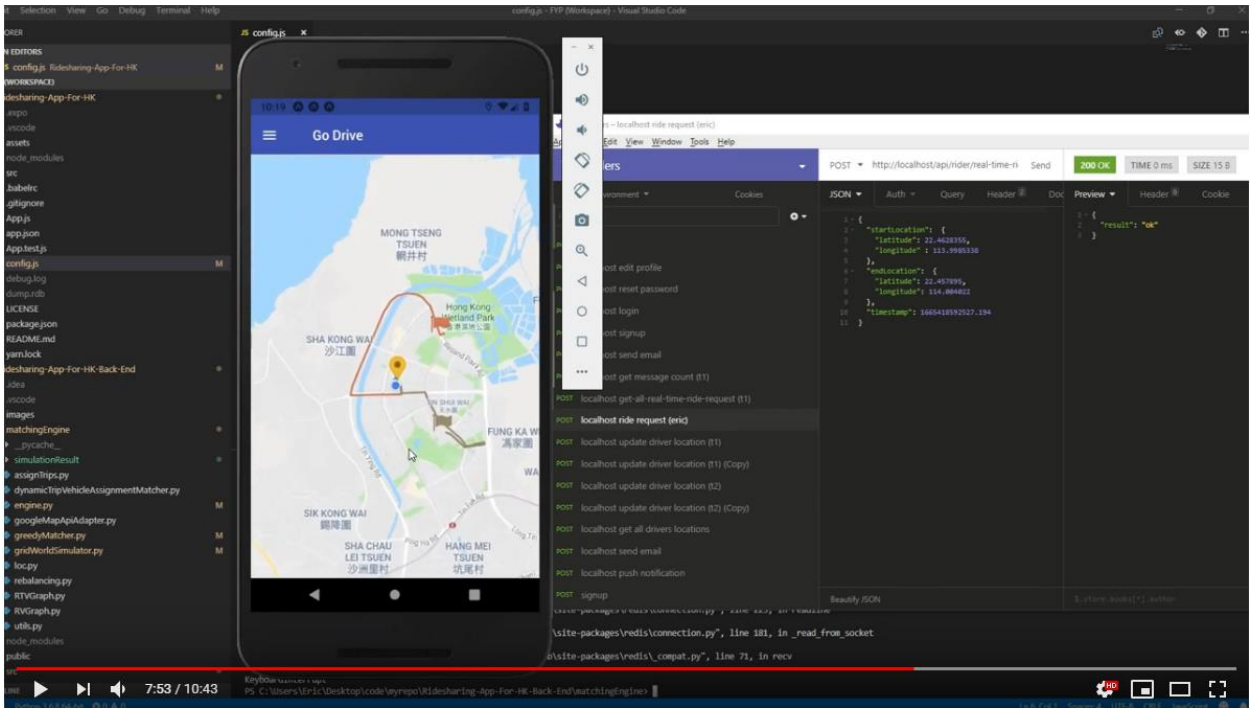


Figure 6 - Screenshots of Demo Video on YouTube

However, it is difficult and inefficient to explain the flow of using the mobile application in this report. Instead, a 10-min video (as shown in figure 6) demonstrating how to run and how to use the mobile application is made and uploaded on YouTube at the link: https://youtu.be/lWX9K4Xj9gU.

2.3.4 User Authentication Process

The user authentication process in software requires special attention, because this process involved users' sensitive information which are their passwords. For this project, users' privacy

and server-side security will be well handled by adopting industry standard for implementing user authentication feature.

When a user signs up in the mobile application, his or her email and password are sent to the web server via SSL and there is no third party can obtain those data. After the web server received those data, instead of storing the password into database in plain text, the web server keeps the hashed value, which is calculated by the password received and a random value, into database to ensure nobody can derive the original password from the database record.

When the user logins in the mobile application, the email and password entered will again be sent to the web server and used to generate another hashed value. By comparing the freshly generated hashed value and the hashed value in database, the identity of the user can be confirmed. Upon successful login, a JWT token will be generated and sent back to user. JWT is an encrypted string that is used for Token-based Authentication in which a user only needs to log in for once to get a JWT and mobile application can present it to the web server for each request [11]. The authentication method using JWT is suitable for mobile application because it only requires the user to log in once, easy to implement, and are reasonably safe [11].

3 Testing and Results

3.1 Unit and Manual Test

Both unit test and manual test have been conducted for the project. The mobile application is tested manually using Expo, which is a mobile application build tool for React Native framework. Each end point of the web server mentioned in 2.3.1 is tested using a RESTful call test tools called Insomnia (figure), which can easily make GET and POST HTTP/HTTPS requests to web servers. For the matching engine, each module is tested individually using simple test cases in `if __name__ == "__main__":` block.

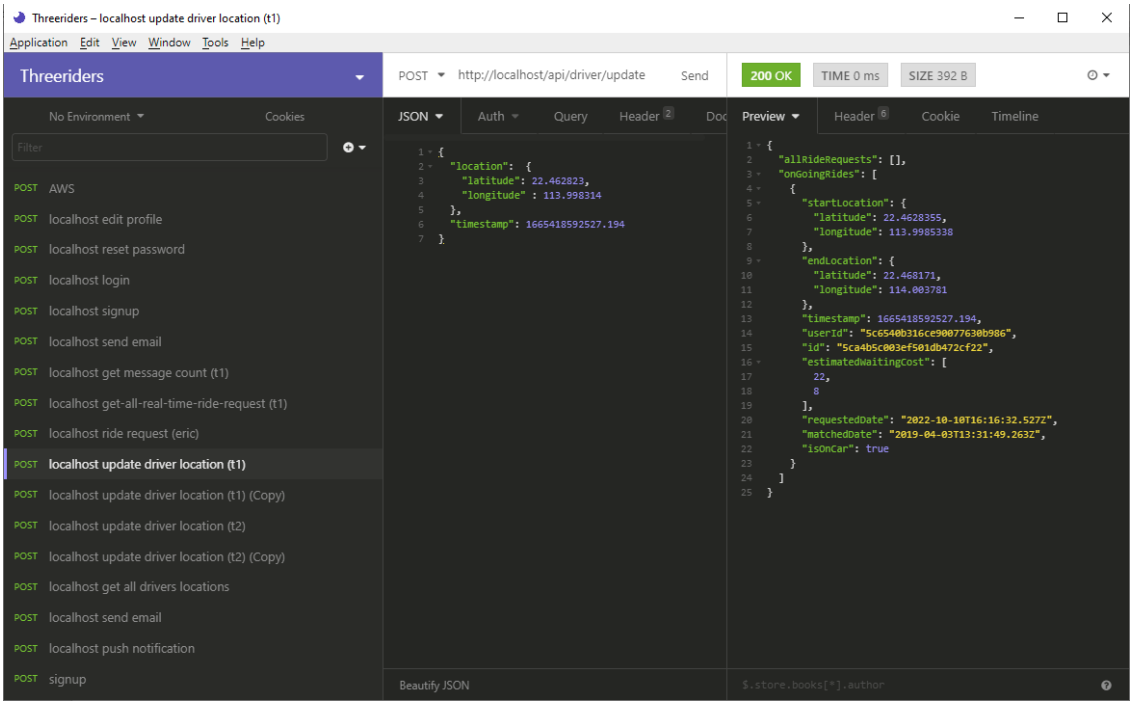


Figure 7 - Screenshot of Insomnia Test Tool

3.2 Integration Test

Two cases shown in figure 8 and 9 are created for testing whether the system can work as whole, the test script is located at *Ridesharing-App-For-HK-Back-End/test/caseStudy.py*.

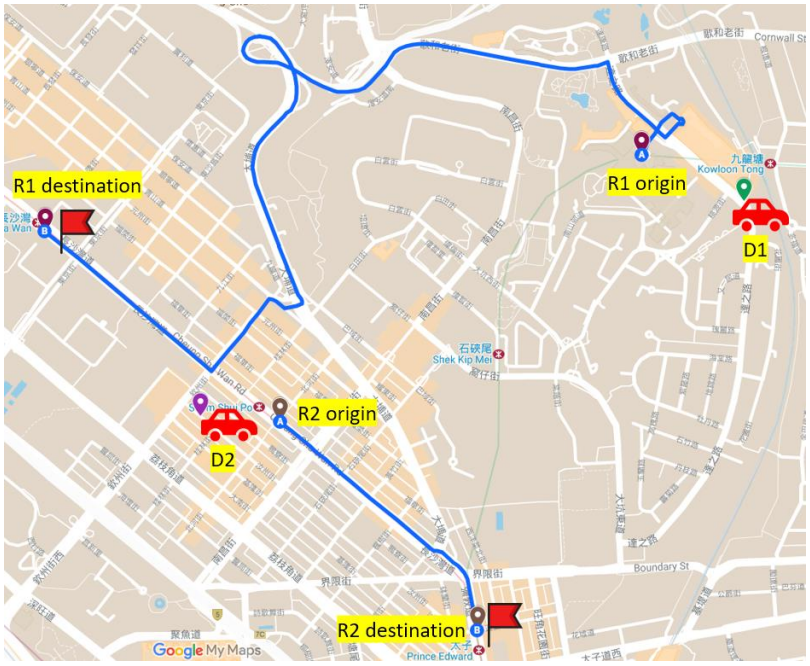


Figure 8 - Integration test - case 1

Figure 9 - Integration test - case 2

For both cases, D1, D2, D3… represents driver 1, driver 2, driver 3, etc. And R1, R2, R3, … represents request 1, request 2, request 3, … etc. We used the same constraints sets Z for both algorithms to solve the two cases and observed the following results.

For case 1, the optimal solution is (R1, D1) and (R2, D2) which means assigning request 1 to driver 1, request 2 to driver 2. Both the Greedy and the Dynamic Assignment algorithms give the optimal assignment.

For case 2, D2 has matched with R2, and the passenger of R2 has already get in D2's car. Although R3 is closest to D2, but R3's destination is completely opposite to R2's destination where D2 must go to. Therefore, the optimal assignment should be (R1, D2) and (R3, D1). In this case, the Dynamic Assignment algorithm gives the optimal assignment while the Greedy algorithm give the mappings (R1, D1) due to its greediness.

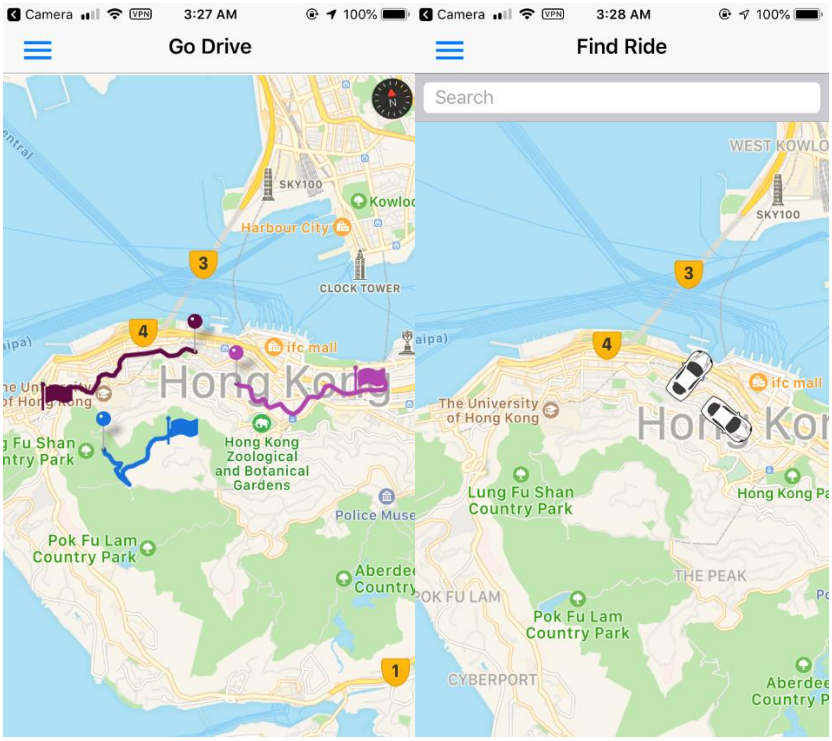


Figure 10 - Integration test - Mobile App Screenshots

Furthermore, the integration test can also be reflected in the mobile application as shown in figure 10.

In short, the system success work as a whole and it can be observed that the Dynamic Assignment algorithm give better solution than the Greedy algorithm in case 2.

3.3 Evaluation of the Matching Algorithms

To decide which algorithms to use in the system, evaluation of the performance on the two algorithms is needed, but computational efficiency is not the focus of this evaluation because the code of the two algorithms is not optimized yet. In order to evaluate the Greedy and Dynamic Assignment algorithms, a way to accurately estimate the travel distance from one location to another and travel distances from multiple sources to multiple destinations. Google Maps API will be the only reliable option for this, however, the pricing of Google Maps Distance Matrix (gives distances from multiple sources to multiple destinations) is not affordable for students.

Therefore, a grid world simulator is built to simulate the peak traffic time in Hong Kong. In the grid world, the common setting is that all vehicles can only move in four directions, east, south, west and north and move exactly *s* units of space every time unit, where *s* is the speed of the vehicles. The moved distance is calculated by the formula $distance(x1, y1, x2, y2) = |x1-x2| + |y1-y2|$ where (x1, y1) and (x2, y2) are the coordinates of two locations. Vehicles will move at random if there is not matched request (trip), and when there is matched request, a best route will be calculated and being followed. Others common parameters used in this experiment is shown in table 4.

Parameters / Variable	Simulator value	Approximate to real world value
Time unit	1	6 seconds
Grid World Width x Height	5000 x 1000	5 km x 1 km
Vehicle's Speed	82	50 km/h
Car's Capacity	2	2
Max. Delay Distance*	1000	1km
Max. Waiting Distance*	1000	1km
Interval to run the matching	10	60 seconds
Drivers' initial location	Random	N/A
Requests' l_p and l_f	Random	N/A

Table 4 - Parameter used in Evaluating Matching Algorithms

(*: Those two are constraints parameters. Max. Waiting Distance means the maximum distance between request's start location and driver current location, which is used for both Greedy and Dynamic Assignment algorithms. Max. Delay Distance is the maximum distance that a request could incur by affecting others ongoing rides on a driver)

There are two experiments conducted, one is to simulate the peak traffic time, another is to benchmark the two algorithms with various driver-request ratios.

For simulating the peak traffic time, the number of drivers is fixed at 100 during the whole process. From t=1 to t=200, 5 requests are generated at every time units. Therefore, there are 1000 requests generated in total. Matching is conducted at an interval of 10 time units. Match rate, share rate, accumulated unhandled requests, and delay (waiting time and total delay) were chosen as the metrics to compare the Greedy and Dynamic Assignment algorithms. The definitions of those metrics are shown as following:

Match rate = # of requests that is matched / # of total requests at a time frame

Share rate = # of trips that shares the car with another trip / # of total trip at a time frame

Waiting time = pickup time of a request - time a request gets matched

Total delay = arrival time of a request - time a request gets matched

(Note: for unhandled requests, their waiting time will also be counted into total delay and waiting time)

Figure 11 - Result of the Peak Traffic Time Experiment

Figure 11 is the result of the peak traffic time experiment, it shows that the average total delays a passenger experienced using the Dynamic Assignment algorithm is less, and the total delay (which included waiting time) is actually dominated by waiting time. The matching rate and share rate of

Dynamic Assignment are generally higher than Greedy, while the Dynamic Assignment has less accumulated unhandled requests.

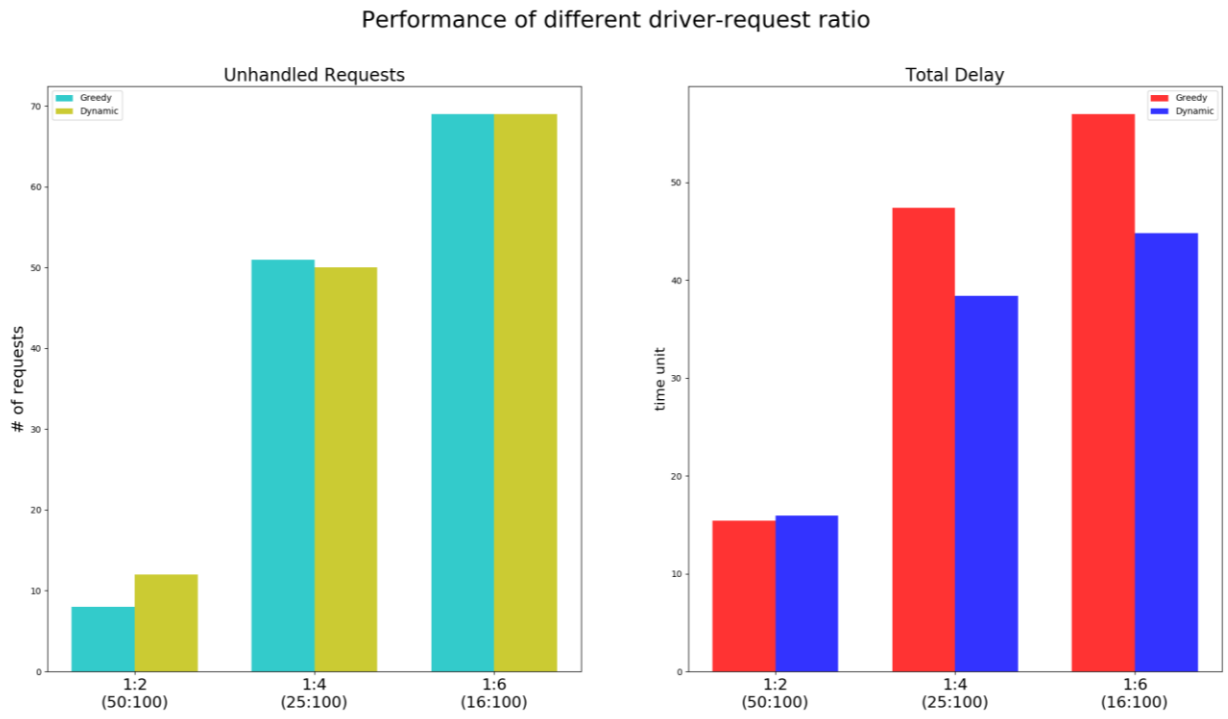


Figure 12 - Result of the Benchmark Experiment

For the benchmark experiment, both the number of driver and requests are fixed at the beginning, and there is only one matching at t=0, all the unhandled requests will be left alone, but their waiting time will be counted into total delay. Moreover, 3 simulations with 1:2, 1:4, and 1:6 **driver-request ratios** were conducted. The actual number of drivers and requests were 50:100, 25:100, and 16:100 respectively. The result shown in figure 12 illustrates more shortage of drivers it is, better the Dynamic Algorithm performs.

In conclude, the Dynamic Assignment algorithm has better performance (except in computational time) compared to the Greedy algorithm.

4. Limitations and Difficulties

In this project, the biggest difficulties encountered are regarding to understanding the academic papers and selecting proper parameters and setting to run the grid world simulation. First, it is difficult for the team members to understand the academic papers related to matching algorithms for the ride-sharing system since those papers consist of a number of advanced mathematics topics in linear algebra, calculus, and number theory which are not taught in any undergraduate courses in HKU. To be able to implement the matching algorithms in those papers, self-study of those unfamiliar topics is needed to fill the knowledge gap. Secondly, it is chaotic and time consuming to develop and run the simulator since there are too many options and parameter choices and the execution of simulation take very long time. Moreover, attributed to the insufficient experiences in software development of the team members, it was often the case that a great amount of time had been wasted on learning some frameworks, for example, Redux and Firebase, but it was realized at the end that those frameworks may not suitable for this project.

Regarding the limitations encountered, one frustrating fact is the pricing of Google Map API services, it charges a lot of money (5 US dollars per 1000 travel distance inquiries). However, there is no suitable alternative can be found to replace Google Map API services that can provide accurate travel distance for Hong Kong (for example MapQuest). Therefore, some features are not production-ready, for example, the matching engine cannot serve too much requests due to budget constraint.

To sum up, the project has difficulties in matching algorithms and testing related aspects.

5. Conclusion and Future Works

This project presents a free-of-charge, full-featured, safe, and easy-to-use ride-sharing digital platform for smartphone users with the mission of reducing the traffic congestion by utilizing the capacity of private cars in Hong Kong.

In methodology section, the abstract level design of the whole system and the details of implementation were explained. In section 3, the testing approach and results were reported. Furthermore, limitations and difficulties encountered were discussed in section 4.

The future works could be making the mobile application production ready and publish to iOS's App Store and Android's Play Store. The UI/UX of the mobile application and the time efficiency of the matching algorithm could also be improved in the future.

6. Division of Labor

In this project, group members cooperated using several project managements, team collaboration, and version control tools, such as Trello, Skype, GitHub, Google Drive, and WhatsApp. To minimize the workload of maintaining progress and contributions documentations, only a list of responsibilities is made to keep track of the contribution of each member. Table 5 lists the works that mainly done by each member.

Leung Hon Man's work	Lau Chi Ho's work	Lau Chun Yin's work
App's Search Page Alarm Message	App's Login & Reset Password Pages	App's Search Ride Page: Driver Location Icon
App's Signup Page	App's Push Notification	App's Search Ride Page: Ride Request Path
App's Google Distance Matrix API Adapter	App's Go Drive Page	App's Message Page
App's Edit Profile Page	Web Server End-points	Web Server: System Messaging (Socket.io)
Dynamic Matching Algorithm: RTV-graph	Integration Test	Dynamic Matching Algorithm: RV-graph
Dynamic Matching Algorithm: Assignment Problem ILP	Greedy Algorithm & help with ILP implementation	Dynamic Matching Algorithm: Rebalance
	Grid World Simulator	

Table 5 - List of Responsibilities of Each Member

References

[1] Stefansdotter, A., Danielsson, C., Nielsen, C. K., & Sunesen, E. R. (2015). Economic benefits of peer-to peer transport services [Online]. Available: https://www.copenhageneconomics.com/dyn/resources/Publication/publicationPDF/0/320/1441009386/economics-benefits-of-peer-to-peer-transport-services.pdf. [Accessed Oct. 16, 2018].

[2] Hong Kong Transport Advisory Committee. (2014). Report on Study of Road Traffic Congestion in Hong Kong [Online]. Available: http://www.thb.gov.hk/eng/boards/transport/land/Full_Eng_C_cover.pdf. [Accessed Oct. 16, 2018].

[3] A. Belz, E. Healey, and K. Hudgins. (2016). Car Sharing: A Feasibility Study in Hong Kong [Online]. Available: https://web.wpi.edu/Pubs/E-project/Available/E-project-030716-041007/unrestricted/Car_Sharing-_A_Feasibility_Study_in_Hong_Kong.pdf. [Accessed Oct. 17, 2018].

[4] Federal Highway Administration. (2015). Initial Stage Reference Search: Real-time ridesharing [Online]. Available: https://www.fhwa.dot.gov/publications/research/ear/15069/15069.pdf. [Accessed Oct. 17, 2018].

[5] C. Dutta, and C. Sholley. (2018). Online Matching in a Ride-Sharing Platform [Online]. Available: https://arxiv.org/pdf/1806.10327.pdf. [Accessed Oct. 17, 2018].

[6] Z. Y. Huang, N. Kang, Z. H. G. Tang, X. W. Wu, Y. H. Zhang, and X. Zhu. (2018). How to Match when All Vertices Arrive Online [Online]. Available: https://arxiv.org/pdf/1802.03905.pdf. [Accessed Oct. 13, 2018].

[7] J. Alonso-Mora et al. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment [Online]. Available: https://www.pnas.org/content/114/3/462. [Accessed Oct. 20, 2018].

[8] Santi P, et al. (2014) Quantifying the benefits of vehicle pooling with shareability networks [Online]. Available: https://www.pnas.org/content/111/37/13290. [Accessed Jan. 13, 2019].

[9] 3-Tier Architecture: A Complete Overview. (n.d.). Jinfonet Software [Online]. Available: https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/. [Accessed Oct. 20, 2018].

[10] Main differences between SOAP and RESTful web services in java. (2017). Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/2131965/main-differences-between-soap-and-restful-web-services-in-java. [Accessed Oct. 21, 2018].

[11] K. Lathiya. (2018, Feb 22). Node Js JWT Authentication Tutorial From Scratch [Online]. Available: https://appdividend.com/2018/02/07/node-js-jwt-authentication-tutorial-scratch/. [Accessed Oct. 22, 2018].

Appendix 1 - project schedule

Deadline	Tasks	
Inception		
30/09/2019	Research on ride-sharing related papers	✓
30/09/2019	Research on programming languages / frameworks / technologies available for building the ride-sharing system	✓
30/09/2019	FYP Website	✓
30/09/2019	Detailed project plan	✓
Elaboration		
31/12/2019	Study of Mobile App Development Framework	✓
31/12/2019	Web server's login, register, reset password features	✓
31/12/2019	Database setup	✓
31/12/2019	Mobile Application's login, register, reset password features	✓
31/12/2019	Matching engine with a simplified algorithm	✓
31/12/2019	Detailed interim report	✓
Construction		
14/04/2019	Matching engine with cutting-edge algorithms	✓
14/04/2019	Mobile Application with full features	✓
14/04/2019	Web server with full features	✓
14/04/2019	Testing and performance evaluation of the system	✓
14/04/2019	Final report	✓

(Note: finished tasks are marked with ✓ symbol)