The University of Hong Kong
COMP4801 Final Year Project
Final Report
Building an easy-to-use ride-sharing app for Hong Kong

Name: Lau Chun Yin (3035294715)
Members’ UID: 3035326049
3035278400
Date of Submission: 14/4/2019
Abstract

Ride-sharing apps has developed rapidly in recent year, for example, Uber and Lyft has publish ride-sharing in other country. However, there are only car hailing apps which provide one-to-one matching in Hong Kong. This cannot fit the Hong Kong crowded environment. Therefore, this project aims to building the ride-sharing app using different algorithm for matching two or more group of riders.

In the project, the application is built in both IOS and Android platform. The user can register with email and password and login with registered information to be a rider or driver. The request will send to the server compute the ride-sharing algorithm. The paper will introduce the function design and the algorithm design.
Acknowledgment

I want to present my greatest appreciation to my supervisor Dr. Huang Zhiyi who advising me during the project and the second examiner Dr. Hubert Chan for commenting our project. I also want to thank my CAES lecturer Joanna Lee for teaching me on report writing.
Table of Contents

Abstract ........................................................................................................................................... 2

Acknowledgment ............................................................................................................................. 3

List of Figures .................................................................................................................................. 6

List of Table .................................................................................................................................... 7

Abbreviation ..................................................................................................................................... 7

1 Introduction ..................................................................................................................................... 8

1.1 Background ................................................................................................................................. 8

1.2 Motivation ................................................................................................................................... 8

1.3 Objective and Scope ..................................................................................................................... 9

1.4 Outline ......................................................................................................................................... 10

2 Methodology .................................................................................................................................. 11

2.1 Application System Design ........................................................................................................ 11

2.2 Matching Algorithm Design ...................................................................................................... 14

2.2.1 Algorithm 1 – Greedy algorithm .......................................................................................... 14

2.2.2 Algorithm 2 – Dynamic trip-vehicle assignment - Problem Definition ................................. 14

2.2.3 Algorithm 2 – Dynamic trip-vehicle assignment - Problem Statement ............................... 14

2.2.4 Algorithm 2 – Dynamic trip-vehicle assignment ................................................................. 16

3 User Interface and server implementation .................................................................................... 19

3.1 Register and Login function ....................................................................................................... 19

3.2 Find ride function ....................................................................................................................... 22

3.3 Messaging Function .................................................................................................................... 24

3.4 Edit Profit Function ................................................................................................................... 26

3.5 Go Drive Function ...................................................................................................................... 27

4 Evaluation ..................................................................................................................................... 28

4.1 Server Response Test .................................................................................................................. 28

4.2 Algorithm case study .................................................................................................................. 28
List of Figures

Figure 1: Presentation Layer, Application Layer and Data Layer structure .................11
Figure 2: The work flow of four step of the matching algorithm [11].........................16
Figure 3: Shareability of two trips.................................................................17
Figure 4: Workflow of registration......................................................................19
Figure 5: Workflow of Login...............................................................................20
Figure 6: The JSON web token (JWT) procedure utilized in the user authorized
function...........................................................................................................21
Figure 7: Map view and sending request...............................................................23
Figure 8: Message pop-up window and review message page..............................24
Figure 9: Edit profile page and the MongoDB user data collection......................26
Figure 10: Go Drive screen capture....................................................................27
Figure 11: Testing POST request for signup and login........................................28
Figure 12: test case 1........................................................................................29
Figure 13: test case 2........................................................................................30
List of Table

Table 1: Performance between Redis and MongoDB [6] .........................................13
Table 2: Compare table between grid world and real world ....................................31
Table 3: Performance testing on fix driver and continues requests ............................32
Table 4: Benchmark experiment of two algorithm .................................................33

Abbreviation

JSON – JavaScript Object Notation
JWT – JSON web token
API – Application programming interface
ILP – Integer Linear Program
UI – User Interface
1 Introduction

1.1 Background

In recent year, mobile computing is invented, it changes the people’s life style – we like to share property with others through the Internet. Therefore, ride-sharing service is introduced. Ride-sharing need a matching algorithm to match two or more group of riders with similar route to share a car. It is different from ride-hailing which is one-to-one matching for a ride through mobile or web which is similar to taxi service.

In the world, there are some popular ride-hailing companies such as Uber, Lyft and Grab etc. [1] These companies offer mainly car hailing service. In recent year, they have provided the new ride-sharing service in some cities such as New York, Los Angeles and Beijing etc. [2][3] However, in Hong Kong, Uber is the only company providing service among these worldwide companies and only provide the car hailing service.

1.2 Motivation

This paper chooses to build the ride-sharing service instead of car hailing service because car hailing service is not suitable for Hong Kong environment. For example, Hong Kong traffic is very crowded which has more than 730,000 vehicles are driving on the 2,100 km road [4] and those cars consist of a driver only or with one passenger. Therefore, ride-sharing apps can encourage people to sharing the car with two or more people instead of driving alone or using one to one matching like the taxi service. If more people are sharing a car, the number of one-rider cars on the road will be reduced and it will have fewer traffic congestions.
Also, Hong Kong is not legal for carrying passenger with getting reward and there are some drivers arrested by the government [5]. The ride-sharing apps that discussed in this paper is a non-profit application and there will not be in-app money transfer. We want to encourage the government and people pay attention to the raising of ride-sharing and make Hong Kong transport system better.

An easy-to-use ride sharing app can benefit to rider and driver. It will be more convenient to riders because they can choose the specific origin and destination and it will save time without changing public transports. Driver can save some money by sharing the riding cost when they open their private vehicle to the public.

1.3 Objective and Scope

To improve the public transport system in Hong Kong, this project aims to build an easy-to-use ride sharing app in both IOS and Android platform for Hong Kong. The application will mainly focus on design and implementing the matching algorithm for suggesting possible match after the users choose their start and end destinations. Also, making the app easy-to-use, this project builds an interactive user interface with the login page and the map for them to choose the origin and destination.

At the first state, this project will have a basic greedy algorithm try to pairwise rider and driver. After that, the project attempts to build the dynamic algorithm discussed in the “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment” paper publish in 2017[11]. This paper will compare the performance between existing matching
algorithms and the greedy algorithm and showing the result. Due to the budget and the 
hardware performance, the matching algorithm will only be able to handle two groups of 
riders with at most two people.

1.4 Outline

The paper will discuss about our ride-sharing application. First, this paper will describe 
how we design the application and the basic functions to make users easy-to-use. It 
includes detail framework of our application and which kinds of a programming language 
are we going to utilize.

Next, the paper will focus on the detail of the ride matching algorithm. At the beginning, 
the paper will introduce the ride-sharing problem that need to solve. Then, we will further 
discuss about the matching algorithm to solve the problem step by step. Finally, this 
paper will discuss the testing method and the result we obtain.
2 Methodology

2.1 Application System Design

The design of the application contains three layers – Presentation Layer, Application Layer and Data Layer (Fig. 1). They are commonly used in building application with client-server system and they work together by sending request and response. The layers structure helps easily maintenances because they can work independently.

![Figure 1: Presentation Layer, Application Layer and Data Layer structure](image)

The Presentation Layer is the front-end layer which consist of the user interface. This layer will be implemented by React Native Expo. It can generate the application to IOS and Android version with similar interface design and functions. Also, it is the programming language which can render the different style of application text, button and figure of the application and send request when performing user action. For example, the user creates his own account and having ride-sharing through the app. Additionally, this
paper will install and utilize React Native Expo’s additional application packages into our project such as Google Map packages. Therefore, this paper will choose React Native Expo to build user interface.

The Application Layer is the back-end layer containing the functional business logic. In this project, NodeJS with ExpressJS framework is the main part of the application layer. It is at the middle part of figure 1 which means it acts as a bridge for handling the request of the presentation layer, retrieving and sending data to the data layer, and the matching engine is built by Python language. NodeJS is the server-language build-up based on JavaScript. It is easy for handling JavaScript Object Notation (JSON) requests by the application and data layers. Nowadays, more and more people are using NodeJS.

The other part of the application layer is the matching engine using Python as the programming language. In this project, separating matching engine and the server is because matching algorithm takes time to compute, and therefore, handling request and matching can run independently. Also, Python provides a large number of extended libraries which include calculation functions that will be utilized in the project such as NumPy for calculating, matplotlib for plotting graphs.

The Data Layer is the database storage system and data access. Our design of the database consists of two parts – MongoDB and Redis. MongoDB stores the data in JSON format, which is easy to read by humans. However, MongoDB retrieve data speed will be much slower than Redis, as represented by the graph in Table 1. Therefore, MongoDB is utilized for storing the data that is not frequently modified and the human-readable data
such as personal information. Another database tool Redis can store the data in the temporary storage, so it is easier and faster for the machine to perform computing. Therefore, Redis is utilized for dynamic data such as the driving route.

Table 1: Performance between Redis and MongoDB [6]

Our project’s the database and backend server host on the virtual machine which is running the Ubuntu 16.04 operating system and rented by Google Cloud platform for free 300USD credit.
2.2 Matching Algorithm Design

This paper will design a simple greedy algorithm first and compare to the algorithm reference to “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment procedure” [11].

2.2.1 Algorithm 1 – Greedy algorithm

For each request $r$, sort the distance$(r$.origin, r.destination). Then, compute the each vehicles and r which stratified three constraints include waiting time must be less than certain minutes, the empty seat in the drivers must be greater zero, and the most important constraint is that requests must be sharable (will explain below) with the trips the driver carrying. Then, return the matched result.

2.2.2 Algorithm 2 – Dynamic trip-vehicle assignment - Problem Definition

A request $r$ is defined by a tuple $\{o_r, d_r, t_r\}$, indicating its origin $o_r$, its destination $d_r$, and the request $t_r$.

A vehicle $v$ is given by a tuple $\{q_v, P_v\}$, indicating its current position $q_v$ and its passengers $P_v = \{p_1, ..., p_n\}$. A passenger $p \in P$ is same as a request $r$.

A trip $t = \{v, o_{r_1}, o_{r_2} ... , d_{r_n}\}$ is which $o_{rn}$ is the request origin and $d_{rn}$ is the request destination. t is a set of requests origin and destination that can be combined and served by vehicle with best route.

2.2.3 Algorithm 2 – Dynamic trip-vehicle assignment - Problem Statement

In this work we propose a general framework for the multi-vehicle multi-request routing capable of addressing three main problems of ride-sharing.
Problem: At every time-stamp (6 second), there will be a number of vehicles and requests that contain the origin and destination. Then, the engine will combine them to a set of ride requests \( R = \{r_1, r_2, ..., r_n\} \) and a set of vehicles \( V = \{v_1, v_2, ..., v_m\} \) and compute for matching every 10 time-stamps (60 seconds). Compute the optimal assignment of \( V, R \), which minimized the cost function \( C \) and satisfied the constrain \( Z \).

The cost function \( C \) computes the sum of delay distance \( \delta_r \) of a set of previously assigned passenger \( P \), delay distance \( \delta_r \) of newly assigned request and sum of constant \( c_{ko} \) all over unassigned request. Formally,

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

Where \( R_{ok} \) denote set of newly assigned requests \( R_{ko} \) denote set of unassigned requests. We want to minimize this \( C \) function.

We will consider the following set of constraints \( Z \):

- For each request \( r \), the waiting distance \( \delta_{r \in R,v \in V} \) that is the distance between driver and \( r \)'s origin \( \leq \) constant.
- For each request \( r \), the maximum delay distance that is the distance more to pick up another rid < constant
- For each vehicle, the maximum capacity < constant.

Output: a set of trips \( T = \{t_1, t_2, ..., t_n\} \) indicating the trips with best shared route, which minimized the cost function \( C \) and satisfied the constrain \( Z \).
2.2.4 Algorithm 2 – Dynamic trip-vehicle assignment

Solving above problem with a set of ride requests $R = \{r_1, r_2, ..., r_n\}$ and a set of vehicles $V = \{v_1, v_2, ..., v_m\}$. 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) [11].

Figure 2: The work flow of four step of the matching algorithm [11]. A simple case of vehicles and requests. V1 is now carrying a passenger to a destination and V2 is an unassigned car. Also, there are four different request 1,2,3,4.

First, the RV-graph is check for the shareability of request and vehicle. The request and vehicle need to stratify the constrain of the waiting distance $\delta_{R,R,V} < \delta_{\text{const}}$, and maximum capacity < constant (Due to the performance, in our project it is 2). Then, the algorithm will consider two requests’ share route and the separated route. If the share distance (orange line in figure 3) is smaller than sum of separated distance of two requests (two black arrows in figure 3), then they are shareable.
Figure 3: Shareability of two trips

Second, RTV-graph is finding the feasible trip $T$ of the shared vehicle and requests and return delay distance that is the distance travel more to pick up the other passenger. In this part, we will consider the case of the best route to share the passenger minus the separated distance. For example, in figure 3, the best route is R1 origin -> R2 origin -> R2 destination -> R1 destination - R2 origin -> R2 destination - R1 origin -> R1 destination (orange line in figure 3 is best share distance, $\therefore$ delay distance = (1+2) km).

Third, the RTV-graph will output the result and assign the trip in this step. The initial state, the assignment will in greedy way that is sort the cost of edge in RTV and assigned for the vehicle which have minimum delay time. The other more optimal assignment is using Integer Linear Program (ILP) to solve $\min \sum_{i,j \in RTV} C_{i,j} + \sum_{k \in \{1,...,n\}} C_{ko} x_k$, for $x_k \in \{0,1\}$. Finding the minimum edge of RTV-graph and penalty of $C_{ko}$ which is unassigned request. However, the ILP solver takes very long time, therefore, the paper will use the greedy assignment only.

At last, the propose of rebalance 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 \( R_{ko} \) to compute the sum of waiting distance

\[
\sum_{v \in V_{idle}, r \in R_{ko}} \delta_{v,r}^W = \min(V_{idle}, R_{ko}).
\]

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.
3 User Interface and server implementation

We are going to introduce the function implemented in the application in the follow part. The finished functions include register and login function, find ride function, messaging function, go drive function and edit profit function.

3.1 Register and Login function

This project has implemented the register and login function used for identifying the user and it can retrieve their personal data during registration.

Figure 4 is showing the first page of the app. At the page, user can choose to click login or Sign-up button. The first time he/she utilize the app, he/she should create an account first. After user fill in all the require information – email, username and password, the information will be sent to the server and stored into the database through encrypted manner.

Figure 4: Workflow of registration
After registration, the server sends the email to the user and user need to click on the link to activate the account. If the account is not activated, the user cannot login to his/her account. Then, go back to the first page of application and click the login button for redirect to figure 5 login page. Users need to enter their email and password to login, and then the data will be sent to have the server logic to have validation. To implement the Authorization, this paper is referencing NodeJS JWT Authentication Tutorial [5]. Start from the top of figure 6, it is user can fill in their login form including their registered email and password. Then, below the user login form, it will be sent to the server for verifying the credential to get the JSON web token (JWT). After that, JWT will be store into the client’s local storage. JWT is the token that is put in front of the message in every request by the client for authentication. If it is a valid JWT, it will return a find ride page when successful login page showing on the third capture of figure 5.

![Figure 5: Workflow of Login](image)

**Generate User Login form for verification**
Figure 6: The JSON web token (JWT) procedure utilized in the user authorized function
3.2 Find ride function

The project needs a map function to help to perform place finding and route-finding functions. This project chooses to add the Google Map application programming interface (API) into the app. It is the pre-implement real ma that built by Google. The application will choose to add three types of Google Map API into the app – map view, place finding and routing.

First, the map view is utilized for showing the road of Hong Kong. Since the application is building the application for Hong Kong, setting the latitude and longitude be 22.2855 and 114.1577 respectively. The Google API need to register an account and need to pay for the function [7]. The account will generate a unique API key and insert into the application. The map will show the user current location. The map view has the same function as Google Map which can drag to move the map and zoom in and out.

Second, in the map, user can search place in the search field and it will redirect to the resulting place. Then, user can add the origin and destination by tapping on to the map or showing on figure 7. The pin is the origin and the flag is the destination. The result will return the shortest path between starting point and ending point by calling Google Direction API [7] and pass these two coordinates into API. If the user confirms the path, he will choose to submit and send the request to server wait for matching.
Figure 7: Map view and sending request
3.3 Messaging Function

After ride request send out, the matching engine start finding possible driver and match with the request. User need to wait for the server response to your request. If the trip is
successfully matched, it will save the message into the database and send it back to the
driver and rider showing in first two screen capture on Figure 8. The pop-up message
only shows for few seconds, but user can navigate to message page to check for previous
message. The message function is reference socket.io [8] to set up. The message page
will reload message that saved before after user navigate to the message page present by
last two capture in figure 8.
3.4 Edit Profit Function

The edit profile page (Figure 9) is the page for user to update the personal information. There are some fields that is not necessary required during registration such as avatar, driver status and contact number. The above information is used for identifying user and help contacting to the driver. If user want to share their vehicle, he/she need to submit the information in this page. After submitting, the MongoDB will be update showing on figure 9. In addition, for protection manner, user can update the new password and email in the edit profile page.
3.5 Go Drive Function

The go drive page is implemented for driver’s usage. Driver need to accept to be a driver in the edit profile page. When navigate to this page, driver will change the status to online and wait for request. In the go drive page (figure 10), driver can see all requests with shortest path (color line) and his/her current location (blue spot). After matching to a request, it will show that request only. Driver can follow the map to the request’s origin to pick up the passenger.

Figure 10: Go Drive screen capture
4 Evaluation

4.1 Server Response Test

Testing for server handling the request, we use the Insomnia software to test for sending request in the personal computer. In figure 11, we test for the procedure of registration and log in and the NodeJS server is showing proper response. After testing result is acceptable, the code will upload to the Google Cloud server.

![POST request for signup and login](image)

Figure 11: Testing POST request for signup and login

4.2 Algorithm case study

For testing the correctness of the algorithm, we create the two real world situations on the map and it is shown below. Also, this paper will compare the result between greedy and dynamic algorithm.
The first case is shown in figure 12, is the simple case. In this case, there are two requests (R1, R2) and two drivers (D1, D2) without any matched passenger. The optimal solution should be driver D1 match to request R1 and D2 match to R2. It is because the origin of two requests is far away and their route do not satisfy the shareability rule. These two requests should be handled with two drivers. In the case, since D1 is close to R1 origin and similarly D2 is close to R2 origin, we should assign a closer vehicle to the request. If it is not do so, the waiting time will be more. We have tested our algorithm that match D1 -> R1 and D2 -> R2.
Figure 13: test case 2

The second case, there are four request (R1, R2, R3, R4) and two drivers (D1, D2) in figure 13. D2 is assigned to R2 and is carrying R2 to the destination. Currently, the requests and vehicles are getting online, and the algorithm will compute the new match result. Request R1 will be have two choices either match to D1 or D2. In our greedy algorithm, R1 will match to D1 because the driver is the nearest one. However, in our dynamic trip-vehicle assignment algorithm, the route of R1 and R2 is sharable. Therefore, R1 is match to D2 with passenger R2. Then, R3 match to D1. Last request R4 is far away from any driver and it remain unmatched. The result is acceptable because that result is what we want.

In the case study, the dynamic matching algorithm give a better result than greedy algorithm.
4.3 Algorithm performance

Besides perform the case study, we generate many random requests and drivers in the grid world to simulate the request. We use the grid world testing compare the greedy and dynamic algorithm.

Due to utilize Google API for calculating the map distance need a lot of credit, we built the grid world instead of real world. The grid world is vertex have (x, y) coordinate and the vertex can only move on north, east, south, west four direction. The distance between two coordinates is calculate by manhattan distance that is $|x_1-x_2| + |y_1-y_2|$.

The request and vehicle will generate randomly at one time unit. If the vehicle is unmatched, it will move at random direction. If the vehicle is matched with two requests, it will be treated as offline. The comparison between grid world and real world can reference to table 2.

<table>
<thead>
<tr>
<th>Variable comparison</th>
<th>Grid World</th>
<th>Real World</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex representation</td>
<td>x, y coordinate</td>
<td>latitude, longitude</td>
</tr>
<tr>
<td>Time unit</td>
<td>1 time unit</td>
<td>6 seconds</td>
</tr>
<tr>
<td>Maximum Waiting Distance</td>
<td>1000 units</td>
<td>1000m</td>
</tr>
<tr>
<td>Distance</td>
<td>manhattan distance</td>
<td>Shortest path of the road</td>
</tr>
</tbody>
</table>

Table 2: Compare table between grid world and real world
Table 3: Performance testing on fix driver and continues requests.

(Match rate means the percentage of number of requests that matched. Share rate means the rate of trips that shares the car with two passengers. Waiting time is the time waiting for the vehicle arrive until matched. Total delay is time between sending out a request and a request gets matched.)

The first experiment is simulating the real world crowned traffic situation. There are 100 fixed drivers in the grid world waiting for request. Five new requests will continuous send out from $t=0$ to $t=199$. Match engine will compute the result every 10 units.
Table 3 is showing the result between greedy and dynamic algorithm in terms of match rate, share rate, unmated request, and waiting & delay time. As there are more and more requests online, the match rate will drop down because there are no more vehicles getting online. The performance of the dynamic algorithm is slightly better than the greedy algorithm. It is because the delay and waiting time means that dynamic algorithm matches the request with nearest drivers and better sharing route.

Table 4: Benchmark experiment of two algorithms

(Match rate means the percentage of number of requests that matched. Total delay is time between sending out a request and a request gets matched.)
The other is the benchmark experiment, there are fix number of 2:1, 1:1, and 1:2 driver-request ratios were conducted (actual number is 100:50, 50:50, and 25:50). This experiment will only compute the algorithm once and the result is given in table 4. The rate of match of dynamic algorithm is higher than greedy algorithm in the rate of 2:1, 1:1. However, if there are less vehicles on the road, dynamic algorithm will have similar value to greedy. For the total delay, if there are equal or fewer drivers than request, the delay time will be fewer. Therefore, the dynamic algorithm can perform better in term of matching rate and the delay time will be fewer if fewer drivers are online.
5 Limitations and difficulties

The limitations and difficulties we found while working on the project will be discussed in this part. First one is hard to find real-life vehicle-request data in Hong Kong. The second one is the performance is limited due to the limited budget.

5.1 Lack of real-life data

It is hard for us to retrieve many sample real-life data in Hong Kong. Consequently, the application will be different from the reality. To solve this problem, we attempt to use some estimated data for the testing. For example, generate case study manually to test for the algorithm logic is correct or not. Then, generate random requests and random vehicles which simulate real-life requests getting online to test for the performance.

5.2 Limited computational resources

There are limited computational resources due to the limited budget. Our server set up on the Google Cloud Platform is the free for 12 months version and API calling can use up to 300 USD credits. If testing the real-life situation and utilize Google API for calculating the real-life distance, the credit will burn out easily. To solve this problem, our testing only test for the grid world simulation instead of real-life situation.
6 Conclusion

In conclusion, this project aims to build an easy-to-use ride sharing app for Hong Kong because the existing riding-sharing app in Hong Kong is not suitable for the city with high density. If more people are using ride-sharing, it can reduce one-rider cars on the road and can benefit to the environment. This project hopefully can encourage the public concern about the raising of ride-sharing and make Hong Kong transport system better.

This paper has discussed the functions of the ride-share application include register and login function, find ride function, messaging function, go drive function and edit profit function.

In addition, this project will also implement the matching algorithms that can automatically match two or more of riders with similar routes. This paper implements two algorithm – greedy algorithm and dynamic algorithm. This testing result is dynamic algorithm perform better than greedy algorithm.

This part will first discuss the about the future planning for this project. One thing we want to improve is the matching algorithm. Although the algorithm build in this paper can match two groups of riders, the algorithm is not the optimal algorithm. The advance version of the algorithm should be running faster and computing more capacity compare to our algorithm. Also, there are more and more research paper will publish focus on ride-sharing algorithm. There will be better way to have ride-sharing. We hope that the team will follow making improvement and the application can publish to the public.
7 Reference


Appendix 1 – project schedule

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

(Note: finished tasks are marked with ✓ symbol)