IM3080 Design and Innovation Project

AY2022/23 Semester 1

Project Report

Title: NyMe Carpooling

GitHub: https://github.com/yuno-cchi/DIP_22S1

Submitted by: Sahassapon Manussawin

Supervisor: Erry Gunawan

Table of content

Table of figures

1. Background and Motivation

Before the COVID-19 pandemic, carpooling was a popular concept where the carpool driver shares their car with others to get to a common location. It has multiple benefits including reducing carbon footprint. It also provided non-drivers by occupation the ability to fetch passengers to destinations that were on their way. Grab was the default platform for booking these hitch rides as it was a dedicated platform for ride booking. After being faced with a global pandemic, people avoided interacting with strangers and hence the commodity of ride-hitching dropped. As we adapt to the new changes that post-lockdown brought, ride-hitching has been struggling to pick up.

Hence, during our ideation session, we wanted to fill the gap that the pandemic had created, as well as bring new features unseen in any other applications which came before and build an app that is dedicated to ride-hitching, bridging the gap between drivers and riders.

2. Objective

To create a mobile application that is made convenient for both drivers and riders to schedule rides. The cost of the rides depends solely on distance and the average price of fuel.

3. Review of Literature/Technology

GrabHitch, which is a sub feature of Grab transport feature, allows users to either browse bookings near them or plan ahead by creating routes to see passengers that match their intended journey. Given that Hitch drivers are mostly working professionals, they enjoy complete flexibility to choose when and with whom they want to share their ride.

4. Design and Implementation

4.1. Design Consideration and Choice of components

4.1.1. UI design

The colour theme is white background with coral pink as the highlight, which provides a clean interface while signifying warmth and friendliness.

Gestalt's law is also applied in the UI design to eliminate redundant elements, ensuring the tidiness of the overall design.

The clean interface ensures easier readability and usability. And the highlight color naturally attracts users' attention, serving as an effective way to guide users' behaviour. Making it intuitive for the user to navigate within the application.

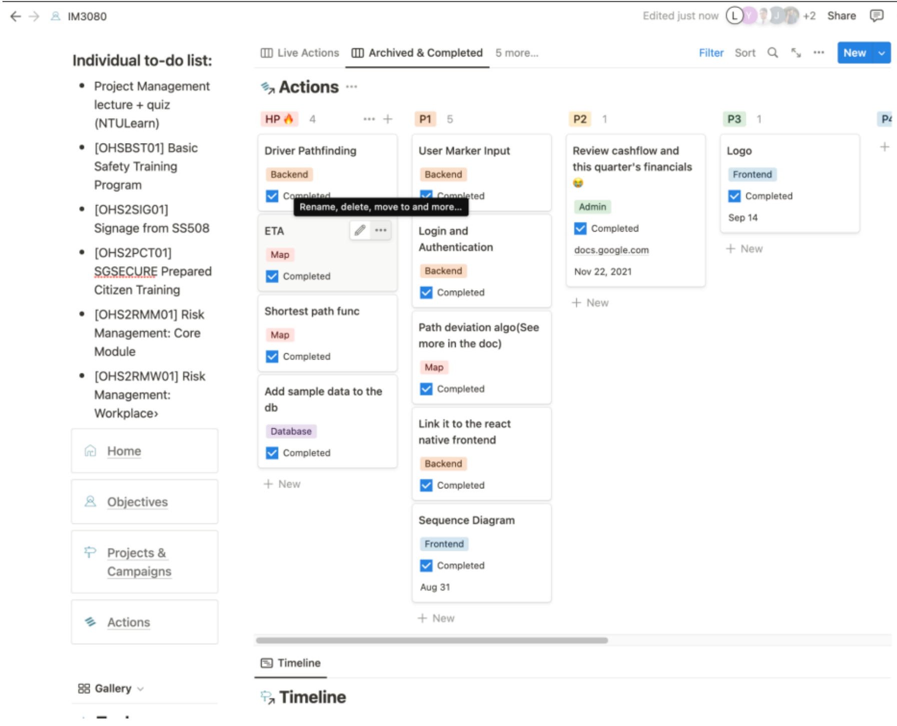
4.1.2. Planning and Project management

We adopted the Evolutionary model. Mobile applications are explicitly designed to accommodate a product that evolves over time. They are iterative, characterized in a manner that enables us to develop increasingly more complete, accurate or better versions of the mobile application.

Specifically, the prototyping sub-model is heavily utilized in this project. Prototyping begins with communication. We defined the overall objectives for the mobile application, identified the requirements and outlined areas where further definition was mandatory.

A prototyping iteration is planned quickly, and modelling ("quick design") occurs. The prototype is deployed and then evaluated. Feedback from our professor-in-charge is used to refine requirements for the mobile application. Iteration occurs as the prototype is tuned to satisfy the needs of the mobile application.

In order to track our progress, we used Notion to organise project details. With the help of Notion, we were able to keep track of all the tasks that each sub-group needed to complete for the application. We were also able to complete the task sequentially, from the highest priority to the lowest.



IM3080 | Edited just now | +2 | Share

Individual to-do list:

- Project Management lecture + quiz (NTULearn)
- [OHSBST01] Basic Safety Training Program
- [OHS2SIG01] Signage from SS508
- [OHS2PCT01] SGSECURE Prepared Citizen Training
- [OHS2RMM01] Risk Management: Core Module
- [OHS2RMW01] Risk Management: Workplace›

Home | Objectives | Projects & Campaigns | Actions

Gallery

Live Actions | Archived & Completed | 5 more... | Filter | Sort | New

Actions

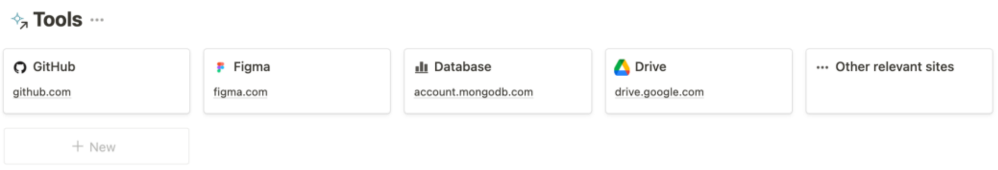
HP 🔥 4	P1 5	P2 1	P3 1
Driver Pathfinding — Backend — ✓ Completed	User Marker Input — Backend — ✓ Completed	Review cashflow and this quarter's financials 😆 — Admin — ✓ Completed — docs.google.com — Nov 22, 2021	Logo — Frontend — ✓ Completed — Sep 14
ETA — Map — ✓ Completed	Login and Authentication — Backend — ✓ Completed	+ New	+ New
Shortest path func — Map — ✓ Completed	Path deviation algo(See more in the doc) — Map — ✓ Completed		
Add sample data to the db — Database — ✓ Completed	Link it to the react native frontend — Backend — ✓ Completed		
+ New	Sequence Diagram — Frontend — ✓ Completed — Aug 31		
	+ New		

Rename, delete, move to and more...

Timeline

Timeline

Figure 1 Task table



Tools

GitHub github.com	Figma figma.com	Database account.mongodb.com	Drive drive.google.com	Other relevant sites

+ New

Figure 2 Tools

The framework chosen is React Native due to the ability to do rapid testing and cross-platform development. For the UI design, Figma is used, and MongoDB is our database of choice.

4.1.3. Promotional Video and Poster

The video is designed in a way to showcase the customer experience of the mobile application. It is meant to be a marketing video that aims to inspire people to learn more about our product and the services that we provide.

We made a scripted storyboard, took the video, and edited it using Premiere Pro and Adobe Audition.

We created two posters. One poster states our objectives and the project model that we used. The other poster is designed as a marketing poster. In the marketing poster, we included the benefits that users can get from using our mobile application which aims to attract our target audiences' attention.

We used "Canva" as a tool to aid us in designing our posters.

We used the AIDA(Awareness, Interest, Desire and Action) approach for both our video and poster. AIDA stands for Attention, Interest, Desire and Action. We first grab the targeted audiences' attention by stating something that involves them and sustain their attention by stating the benefits. Subsequently, give them motivation and call to action to download the mobile application.

4.1.4. Frontend

4.1.4.1. Login and Registration

The Login and Registration screens are implemented as forms that have their inputs validated via JavaScript functions.

During registration, the raw input is validated using Regular Expressions (RegEx), and if the input is deemed to be valid the input is passed by an asynchronous JavaScript function into our MongoDB database.

During a login attempt, another asynchronous JavaScript function checks to see if the entered credentials are in the database.

4.1.4.2. Screen Navigation and Notification

Navigation between different screens is achieved by the use of a stack. In the app, there are buttons to navigate to the next screen and the previous screen, these two functions are implemented by pushing the new screen into the stack and popping respectively.

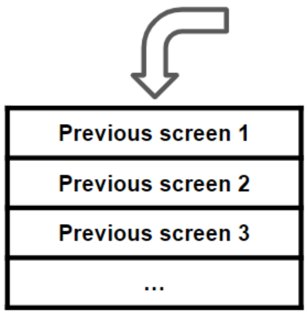


Figure 3 Stack diagram

Notification is achieved by having a database search prior to loading the calendar screen to retrieve the dates a user might schedule. Upon knowing the dates, a notification is scheduled with the use of the internal clock of the phone.

4.1.4.3. Map Navigation and Best route algorithm

Geo Navigation is aided by the use of Google API to get an array of desirable coordinates representing waypoints based on a set of criteria namely traffic condition and distance.

The array of coordinates is used in MapView API which allows access to native map features such as Apple map or Google map. The route is plotted with a set of vectors with the MapViewDirection library.

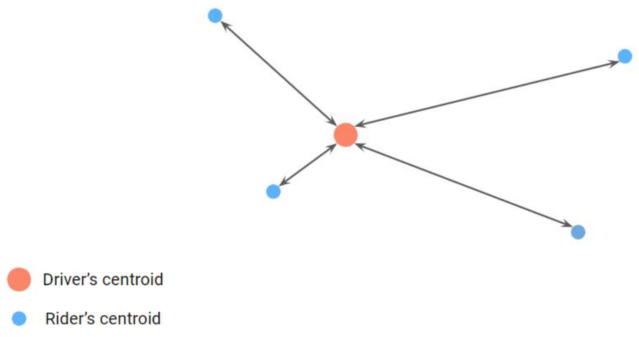


Figure 4 Coordinate-based route searching

The algorithm used to classify the best route determines the goodness of the route based on a variable called “centroid”.

Centroid is the mathematical expression of the sum of the starting point and the destination divided by half.

$$Centroid = \left(\frac{latitude_{start} + latitude_{destination}}{2}, \frac{longitude_{start} + longitude_{destination}}{2}\right)$$

Figure 5 Centroid calculation

The algorithm searches the database with the centroid difference of 0.2 from itself and sorts the routes according to the difference between two centroids, the least being the best.

4.1.4.4. Scheduling function

The scheduling function is designed for riders to put out their future hitching request, and drivers have the ability to choose which request to accept.

In order for users to keep track of their upcoming rides, we have designed a calendar screen for users to view their pending and confirmed requests that consist of Starting point and destination, Time of ride and Price

The scheduling function works hand in hand with the notification function, by prompting users when there is a ride coming up in the near future.

4.1.5. Backend

4.1.5.1. Database and Data Tables

The database used is an online, cloud-service-based, No-SQL database known as MongoDB. As it is online and requires no SQL, this makes project management easier and data handling, making the app scalable in the long run.

Moreover, with MongoDB being a cloud-service-based database platform, the security of the database is already high, making it sufficient for this app.

In order for the app to function, there are three 3 tables implemented in the database that is used for the functionalities of the app:

1. “User” table: this table serves for user authentication and registration, as such, this table stores user data.
2. “Ride” table: this table serves users intending to get a ride by posting their destination for drivers to see
3. “Drive” table: this table serves for drivers by showing them their destinations, pick-up points and the ride requests selected from the “Ride” table.

Overall, these three 3 database tables are the basis for almost all of the app’s functions, and these tables will be where most of the app’s data will be stored and utilised by other hosts using this app.

4.1.5.2. API and Linking to Database

In order for the database to be tethered to the application, an Application Programming Interface (API) needs to be created. For this, MongoDB is able to provide a connection link to the API that was developed in the project using Node.JS.

This API is created in a Model-View-Controller (MVC) method, where the Model files in the backend API represent each table and its attributes in the MongoDB database. The Controller files are to handle HTTP requests made from the client to the database, typically in a form of CRUD functions (Create, Retrieve, Update and Delete). The View, however, is done at the frontend instead.

The Node.JS API utilises an Express server file that connects the API’s MVC to the database itself. As such, this establishes the connection of the API to the project’s MongoDB database.

4.1.5.3. Hosting

As the API is locally done in the project, this would mean that the app would require to manually, and locally boot it, the moment an instance of the app is active. This would be highly unfeasible and therefore, the solution is hosting the API.

The technology used is another cloud-based hosting platform called Heroku. Heroku’s functionalities are to host files by deploying them on a hosted link accessible anywhere via the internet, as well as setting commands to run upon deployment. This fits in the use-case of the API where it is required to run in order to have the app access to the API itself and the database that is linked to it.

With the hosted API up and running, the link is used in the app's functionalities to fully access the database and allow the API to handle the app's data, and display data from the database if required. Therefore, Heroku's hosting of the API enables a communication link between the app frontend to the backend, as well as the database.

5. Final Design

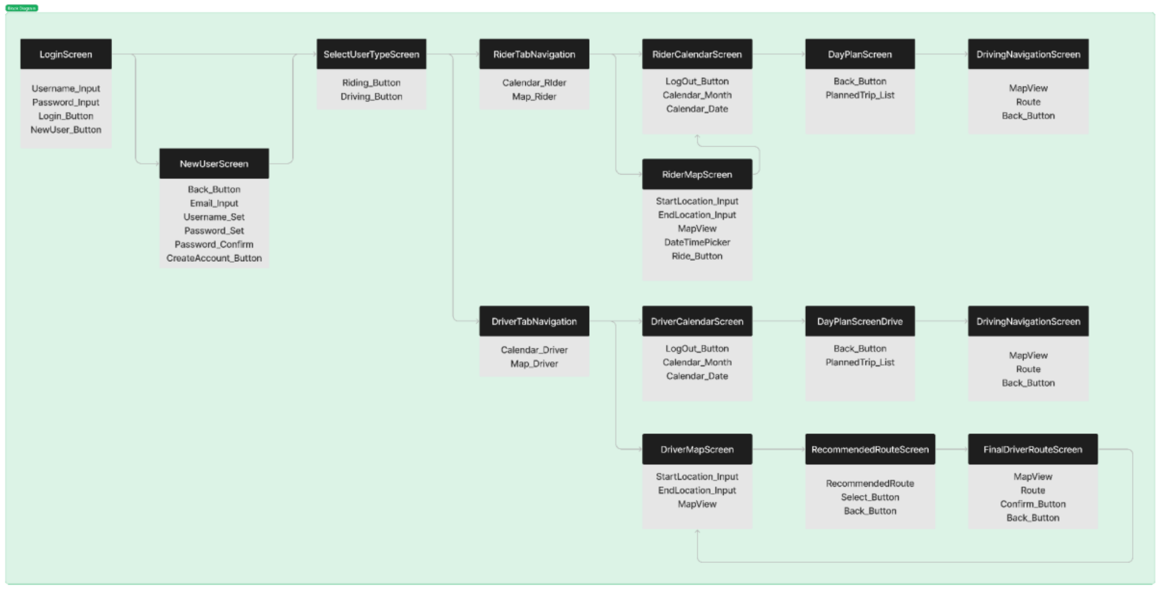


Figure 6 Application block diagram

After the user chooses the role, they will be directed to the respective screens. Both are using tab navigation, containing a calendar screen and either driver set of screens or rider's set based on the selection type. Refer to Figure 6 for more information.

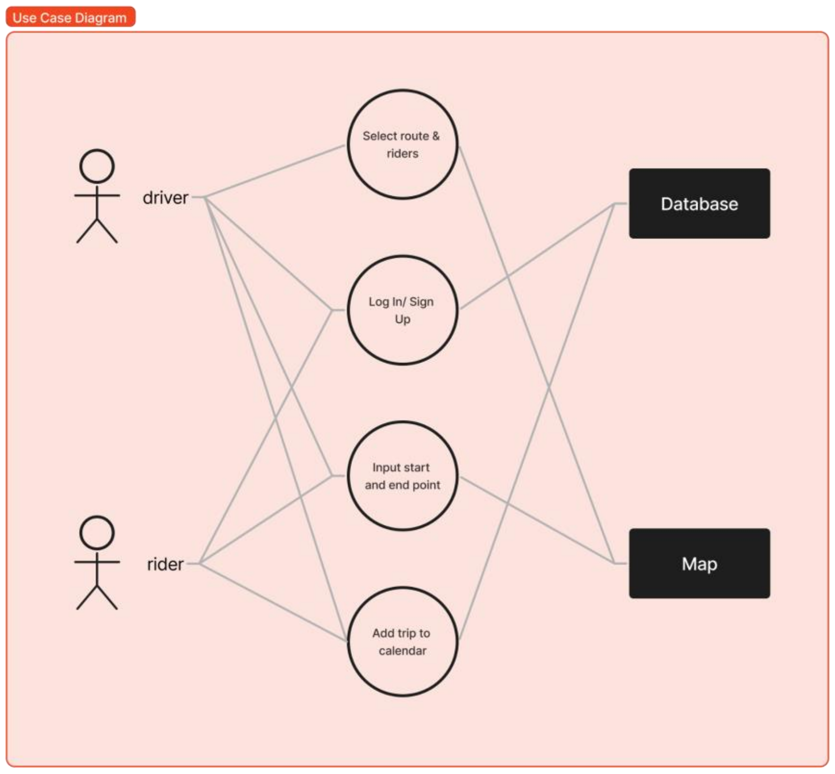


Figure 7 Use-case diagram

Users are classified into drivers and riders based on their selection after each login. While both are able to log in/sign up, input start addresses and destination address, and add trips to the calendar, drivers can select routes and riders after selecting the start and end addresses. Users' account information and trips' information are stored and accessible in the database. The Map functionality shares much of the same functionality but additional recommended routes function for the driver. Refer to Figure 10 for information on driver full list of functionalities.

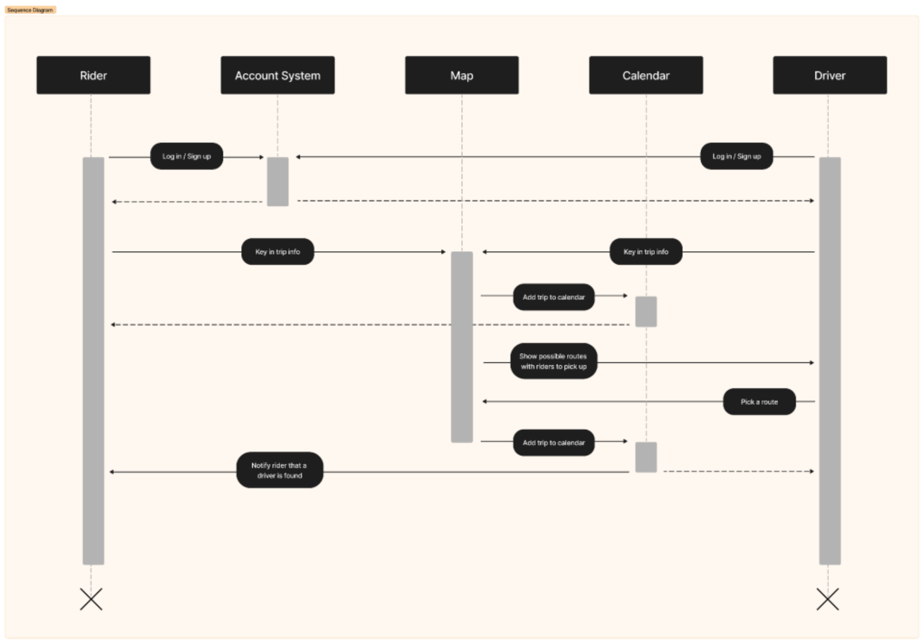


Figure 8 Sequence diagram

Figure 8 depicts the lifespan of the components as well as its following sequences.

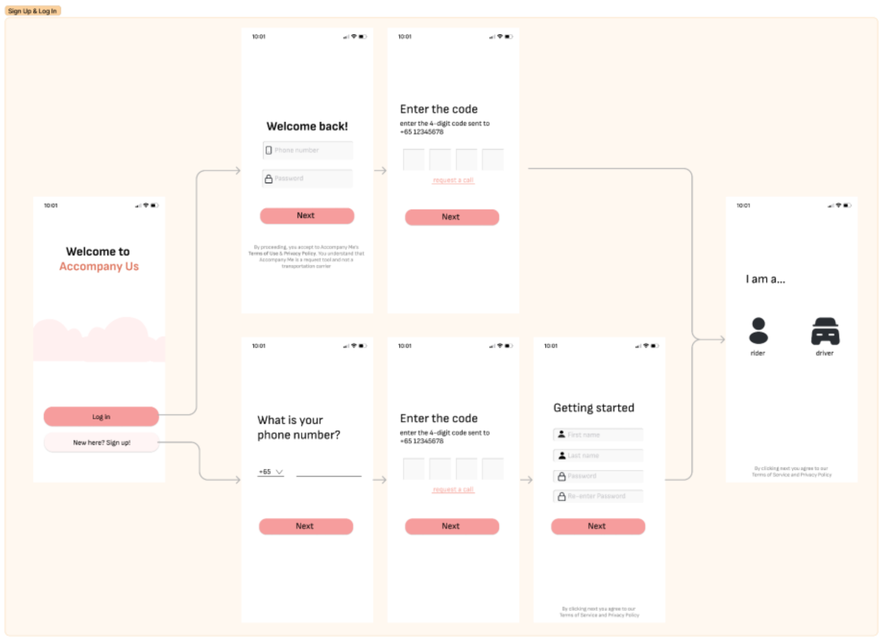


Figure 9 User wireframe(Type less)

Figure 9 indicates the action a user might take before selecting the user type for the following session, either a driver or a rider. The type of user is not permanent and can be changed with each session.

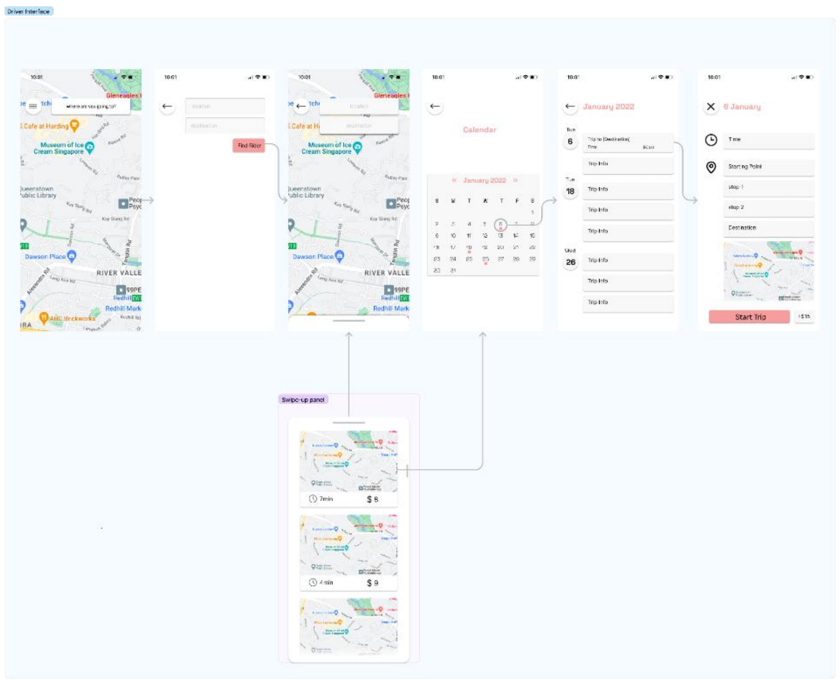


Figure 10 Driver wireframe

The figure above describes the set of actions a driver type can take following the user selection screen in Figure 9.

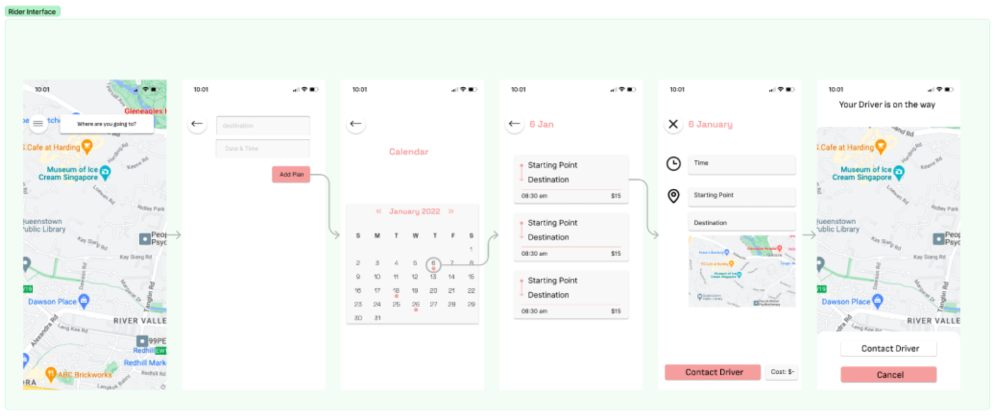


Figure 11 Rider wireframe

Figure 11 shows the full sequence of actions a rider type can go through after the user selection screen.

6. Discussion

6.1. Different ways of searching the best route

There are a few search algorithms initially considered for finding the best routes such as Depth First Search, Breadth First Search and A star. All of them yield very promising results when finding the best route but had a noticeably long search time, and, for BFS, more memory space. Moreover, the searching algorithm above requires far more query call to both Google API and Database API which increases the operation cost exponentially.

A compromise is necessary for reasonable operation cost and price for the customers, thus the heuristic search approach is preferred. The time complexity and space complexity are reduced substantially to constant time and space.

6.2. Schedule notification over Push notification

The use of push notification was considered early on in the project lifecycle, it is preferred in terms of functionality and flexibility but is discouraged upon looking at the cost and software assimilation.

Setting up a push notification system requires the use of another database which proves to be a great challenge due to the rising cost and limited knowledge of integrating nonSQL and SQL together.

6.3. The downside of React Native

Discuss React Native being a barebone framework compared to Flutter which has most packages attached, and why react native is selected.

React Native is a great software development framework which allows rapid prototyping, but it does suffer in terms of core components which are readily available in the first-party library like Flutter. React Native is a community-driven platform, with many critical features such as Map, Server communication and Notification made by third-party developers, some with slightly different naming conventions. The issue of naming is extremely critical and has hurdled the project since the start.

6.4. Software testing

At the end of the project lifecycle, extensive testing is carried out on the software. The testing in focus is decision-based testing and load testing.

Decision-based tests focus on passing parameters between screens, this is due to the nature of each coder, despite the attempt to standardise the naming convention, there are times whereby variables are named out of the rules. This leads to the parameters in the subsequent screens being undefined.

Load on the server is tested by retrieving the routes within a certain distance from the centroid. With the custom algorithm is request call for both the database and map are minimal, numbering around on average 100 API calls per user around 20 searches.

7. Conclusion and Recommendation

7.1. Conclusion

The original goals of developing a carpooling application with scheduling features have been wholly achieved. Moreover, innovative features such as best-route recommendations are implemented successfully. The features which were not in the original draft are also included namely Serverless Notification, Additional route optimisation, Time-based selection, GPS navigation and User authorisation.

7.2. Recommendation for Future Works

7.2.1. Improved Notification System

All notifications within the application are handled locally, this has the benefit of lower cost and eliminates the use of another database. However, notifications are slightly delayed, on average around 5 seconds, and it requires additional memory usage on the user's device. Hence, some

7.2.2. Native development kit over React Native

React Native excels at making multi-platform applications but some features native to the platform itself are extremely difficult to access such as native map magnetic heading.

7.2.3. Periodic data purge

The database currently stores every data request by the users without any deletions for used data, the routes which are expired or travelled. It is highly recommended that the used data is cleaned every year to minimise storage usage and improve searching efficiency.

7.2.4. Economy-Aware Fare System

In the app's current state, the fare is linearly calculated with respect to the distance travelled. In possible future implementations, we could possibly use an API to estimate the current gas prices. With this information, we can more fairly and intelligently calculate the fare.

An example implementation of this method could be:

$$\text{Fare per rider} = \frac{\text{Dollars per liter of gas} \times \text{Liters of gas consumed per km} \times \text{km travelled}}{\text{number of riders} \times \alpha}$$

$$\text{where } 0 < \alpha < 1 \text{ is a constant that decreases as the number of riders increases}$$

Figure 12 Fare calculation

This would encourage more riders to carpool so that fares for them decrease, and the driver has the incentive to pick up more than one rider – which would be better for the environment.

8. Appendix

8.1. User Guide

8.1.1. General

8.1.1.1. Account Creation

The registration screen is accessible via the “New User?” button near the bottom of the Login page. To make a new account, you must enter a new username that meets the following requirements:

- Must be 5-25 characters long
- Must not contain whitespace (spaces, tabs, etc.)
- Must contain only alphanumeric characters or full stops/periods
- Must not begin or end with a full stop/period
- Must not contain consecutive full stops

You must also enter a valid email address.

Next, you must enter a password that meets the following requirements:

- Must be 6-25 characters long
- Must only contain alphanumeric characters and some special characters: (.!?@#$%^&*()-+=<>)

Lastly, you must correctly enter your password.

If all your inputs are valid then after pressing the Register button, you will be redirected to the Account Type Selection screen.

8.1.1.2. Login

Enter a valid username and password and press the “Login” button.

8.1.1.3. Account type selection

Select either the Rider or Driver button, depending on if you are riding or driving.

8.1.2. Driver

8.1.2.1. Calendar

The calendar screen depicts planned “driving” routes in monthly and daily format. For daily format, it encapsulates the details of the route: Price, Time and Locations respectively.

8.1.2.2. Route Selection

Destinations and start are required to be filled in order to proceed to the route selection screen. Travelling time is set as the local current date by default. Route selection screen allows a user to select up to 3 riders, selection can be omitted. Once the routes are selected, a finalised route is presented before confirmation.

8.1.3. Rider

8.1.3.1. Calendar

The calendar screen depicts planned “riding” routes in monthly and daily formats. For daily format, it encapsulates the details of the route: Price, Time, and Locations respectively.

8.1.3.2. Route Request

Both fields, destination and start have to be filled in order for a request to be successful, an alert is prompted otherwise. Indicated riding time initialised at the user’s local time.

8.2. Maintenance Guide

The application is constructed to support both iOS and Android, screens and components that are platform specific are followed by "_{platform_name}" such as "ScreenA_ios".

Data is sent between screens by deconstructing the navigation object into two parameters namely navigation and route. Within the source code, they are in the format as such {navigation, route} which are present in every screen.

The project follows the standard Java naming convention by Oracle.

The Application uses the paid Google Maps API and must be given a new API key if and when the API key is invalid.

The MongoDB database must also be changed on an if-needed basis

8.3. Source Code

The source code of the project resides in the "main" branch on the GitHub repository, other branches, and with the experimental version in the "copy_main" branch. All other branches are for isolated testing purposes.

GitHub link: https://github.com/yuno-cchi/DIP_22S1