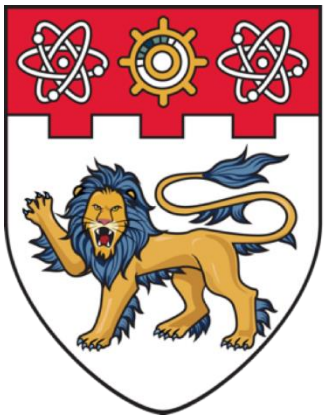


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering

CZ2006 Software Engineering

Software Project Documentation

Lab Group:	**SS2**
Supervisor:	**Zhang Jie**
Instructor:	**Li Xiaoming**
Team Name:	**SkyForce**
Project Name:	**Hang Out SG**
Team members:	**Chen Yinya (U1722474A)**
	He Yuhao (U1722945E)
	Li Bingzi (U1722793H)
	Luo Jinqi (U1722733J)
	Ma Xiao (U1722964K)
	Wang Zijian (U1722613E)

Table of Contents

Acknowledgement

We are team SkyForce from School of Computer Science and Engineering, Nanyang Technological University. This application HangOut SG is based on our course project of CZ2006 Software Engineering. It is a mobile utilities application designed to help Singaporeans to find interesting places to hang out in their spare time. During the software development process, we dedicated ourselves to ensure the best system performance and provide the best user experience. We kept examining the functionalities of various components and making adjustment continuously. With all these efforts, we hope our application helps every user to make their spare time more fruitful.

We would like to give our sincere gratitude to lab supervisor Prof Zhang and Dr. Li for their professional suggestions and guidance during project development. Special thanks also go to Dr. Liu and Dr. Sim for their valuable teaching in the lectures.

SkyForce,

Apr 7th, 2019

1. Introduction

1.1 Purpose

Hang Out Singapore, or HOS in short, is an open-source Android application that provides multiple services to both local customers and vendors. HOS aims to build up an information platform for customers who plan to hang out during free time. It provides online advertising and promotion solutions for local small-scale vendors. By offering simple but powerful tools for users to plan and reserve hang-out activities, HOS is pursuing the goal of accelerating the progress of Smart Nation Movement.

1.2 Target Users

HOS is developed on a requirement-driven basis.

1.2.1 For local customers
Our team noticed that long periods of time are often wasted on deciding where to go when friends plan to hang out together. HOS aims to simplify the searching, planning and reserving process.

1.2.2 For small vendors
Nowadays, many start-up shops and individual vendors find it hard to publicize their amazing products, delight food and services. HOS aims to provides a promotion platform publicizing small local shops, their attractive products and easy reservations.

1.2.3 For back-end management administers

Many Data Engineers suffered from badly-managed backend database. HOS aims to build up a clean and relational database storing customer and vendor information.

1.3 Intended Audience and Reading Suggestions

Front-end developers may use the documentation to review, suggest and implement new features to HOS. The HOS document provides the detailed guideline to the architecture of the application. Project testers shall take this documentation for building test strategies. The documentation provides testers with methodically-organized testing process. Back-end engineers may use the documentation to check the implementation of Databases, APIs and corresponding methods. The APIs are encapsulated and well-documented for developers to implement.

1.4 Product Scope

HOS provides all its services within the scope of Mainland Singapore. In short, the services are designed with the consideration of locality-oriented activities that last no more than 24 hours. By HOS mainly focuses on two aspects:

1.4.1 The usability with high efficiency of searching and planning for local customer users.

1.4.2 The effective promotion with a visible display platform for start-up vendor users.

1.5 References

[1] “Unified Modeling Language (UML) | An Introduction,” *GeeksforGeeks*, 01-Mar-2019. [Online]. Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/. [Accessed: 08-Mar-2019].

[2] “Android Developers,” *Android Developers*. [Online]. Available: https://developer.android.com/. [Accessed: 13-Mar-2019].

[3] “Carpark Availability,” *Data.gov.sg*. [Online]. Available: https://data.gov.sg/dataset/carpark-availability. [Accessed: 12-Mar-2019].

[4] “Spring Projects,” *Spring*. [Online]. Available: https://spring.io/projects/spring-boot. [Accessed: 01-Mar-2019].

[5] “Spring Projects,” *Spring*. [Online]. Available: https://spring.io/projects/spring-boot. [Accessed: 01-Mar-2019].

[6] *Google Maps Platform | Google Developers*. [Online]. Available: https://developers.google.com/maps/documentation/. [Accessed: 28-Feb-2019].

[7] “Adding Android Banner Ads - javatpoint,” *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/android-banner-ads. [Accessed: 08-Mar-2019].

[8] “Package javax.persistence,” *javax.persistence (Java(TM) EE 7 Specification APIs)*, 01-Jun-2015. [Online]. Available: https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html. [Accessed: 23-Mar-2019].

[9] Ramotion, “Ramotion/cardslider-android,” *GitHub*, 08-Feb-2019. [Online]. Available: https://github.com/Ramotion/cardslider-android. [Accessed: 12-Mar-2019].

[10] “HTTP request methods,” *MDN Web Docs*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. [Accessed: 08-Mar-2019].

2. Overall Description

2.1 Product Perspective

Hang Out Singapore is a self-contained, open source product. This product is first developed and deployed on Android platform. Its main responsibilities are to provide recommendations, reservation and plan managing functions for users sign in as customers and provide shop information uploading and reservation managing for users sign in as vendors. The usability of this product allows vendors and customers in Singapore to connect in a simple way. It provides with more options for locals when they decide to hang out while also helps vendors to gain more income.

2.2 Product Functions

The following are the main features that are included in Hang Out Singapore：

- Sign in system: the system allows users to create customer or vendor accounts in the system and providing features of viewing and updating profiles. When users sign in, the UI will direct them to different activity pages designed separately for customers and vendors.
- Recommendation system: the system will recommend shops to customers.
- Reservation and plan system: the system allows customers to make reservation on shop's detail page and also add to his plan. Customer can also create an empty plan and add plan items to it later.
- Shop verification system: vendors can upload their shop information through this system. When administrator approves, customers can see it.

2.3 User Classes and Characteristics

We assume there are two main classes of our users. One is customer, who is considered to have basic experience of using general mobile apps. This product is very easy to use and only requires novice level knowledge of mobile apps. Another class is vendor, who is expected to be familiar with the reservation managing system.

2.4 Operating Environment

The HOS application is built on Android Studio. It operates on mobile phones with android operating system installed. Since its Minimum API Level is API15: Android 4.0.3 IceCreamSandwich, the app will run on approximately 100% of android devices. Therefore, the app is well-adapted. The app will need coordination with the photo gallery of the phone. It will coexist with other apps as long as it is compatible with the operating system.

2.5 Design and Implementation Constraints

Database used: MySQL
Communication protocols: The communication protocol between frontend and backend is HTTP.

Security considerations: The frontend must access datastore using access token.
Design convention: The application uses Spring Boot framework to build the backend. The backend is connected to frontend using API.

2.6 User Documentation

The software targets at general public and follows. Thus, it is very easy to use. The user do not need documentation to understand how to use the app.

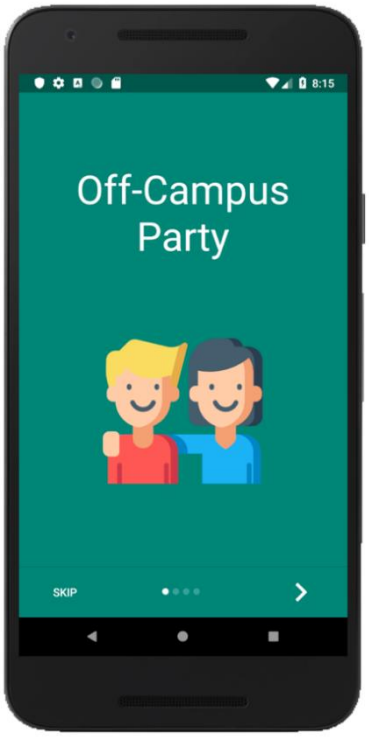
2.7 Assumptions and Dependencies

The server must have MySQL running.
The administrator must have the qualifications to verify vendor's shop certificate.
The users must have a mobile phone with android operating system high than 4.0.3.
The users must connect to internet.

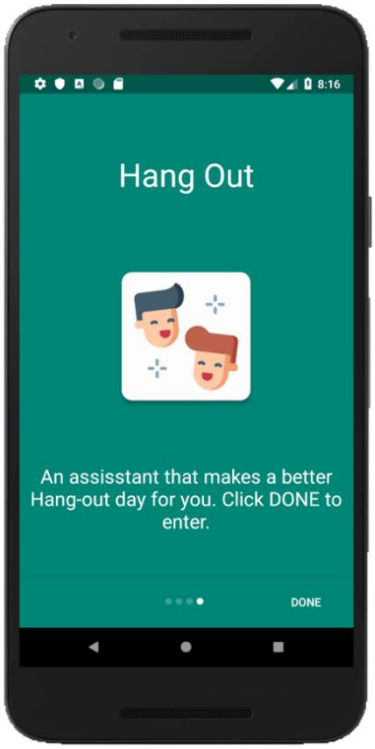
3. External Interfaces

3.1 User Interfaces

3.1.1 This is the welcome page when user first open the app. It gives a preview of the functionality of the software.

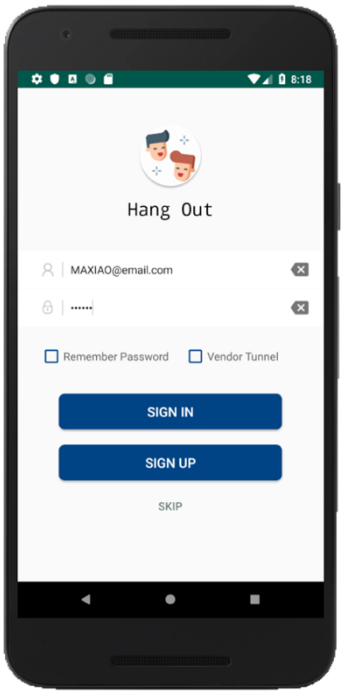


Welcome Page

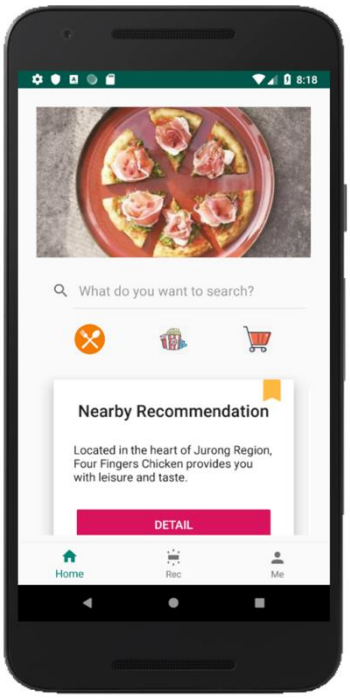


Welcome Page

3.1.2 User can sign in or sign up at the left page. He can go to plaza entrainment or restaurant by clicking the corresponding icon on the home page.

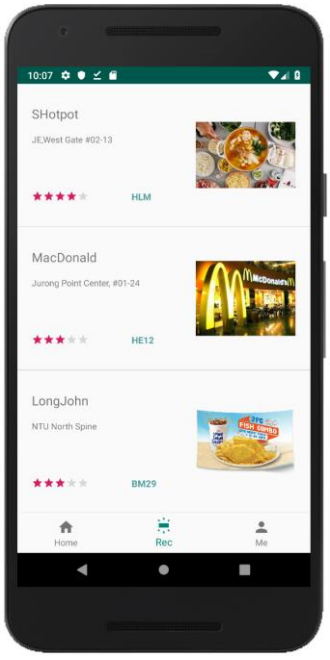


Customer sign in

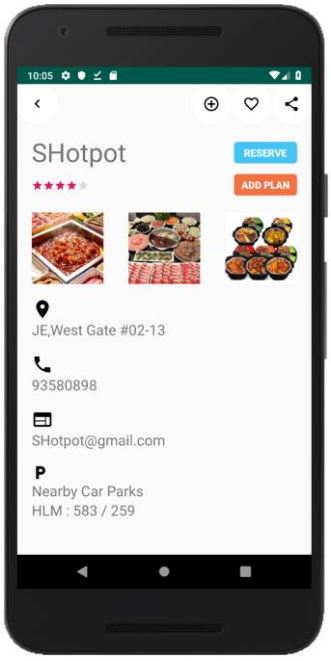


Customer home page

3.1.3 The left is the recommendation page. Customer can click into a shop to see its detail. The shop detail page displays location, contact number and nearby carpark capacity. He can also make a reservation or add the shop to plan by clicking ‘reserve’ or ‘add plan’ button.

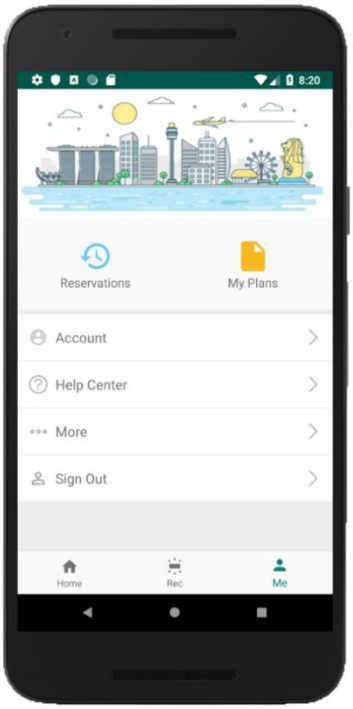


Recommendation

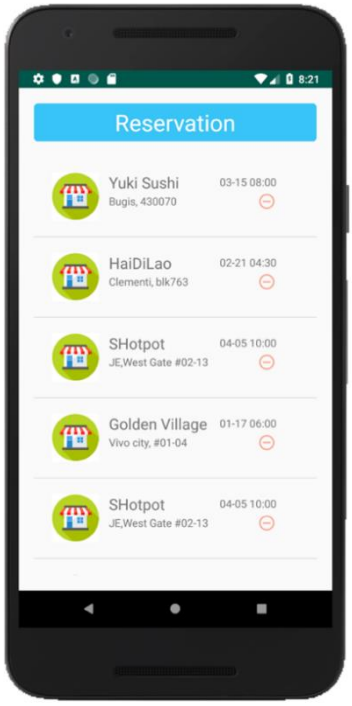


Shop detail

3.1.4 This is ‘me’ page. Customer can retrieve all his information here. The right page displays customer’s reservation history.

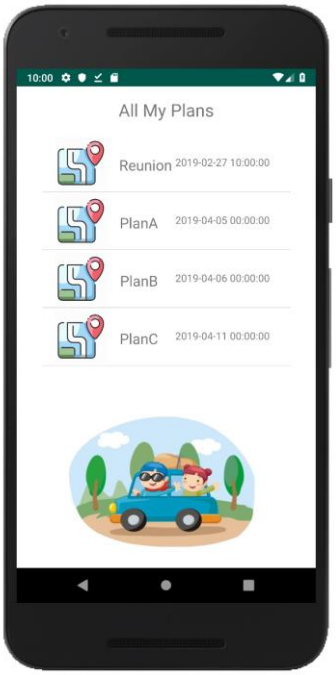


Me page

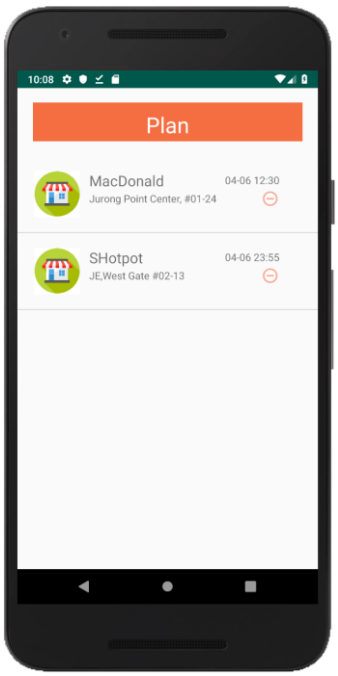


Reservation history

3.1.5 This is ‘me’ page. Customer can retrieve all his information here. The right page displays customer’s reservation history.

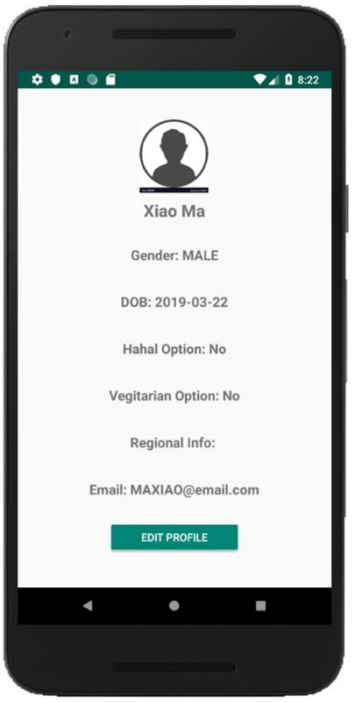


Plan history

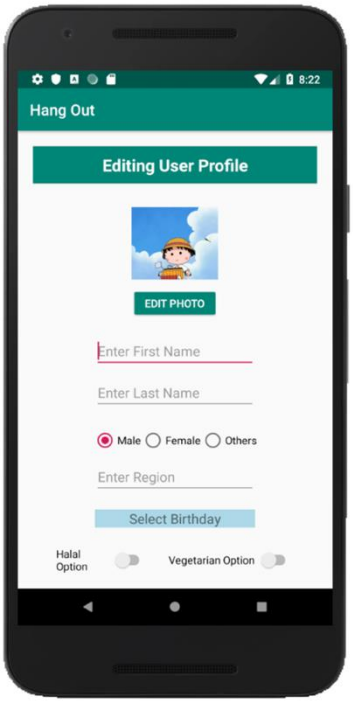


Plan detail

3.1.6 This is the customer profile page. me' page. Customer can edit his information and upload photo from photo album.

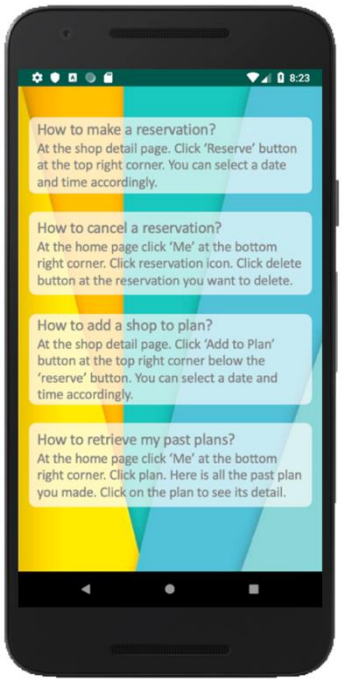


Customer profile page



Edit customer profile

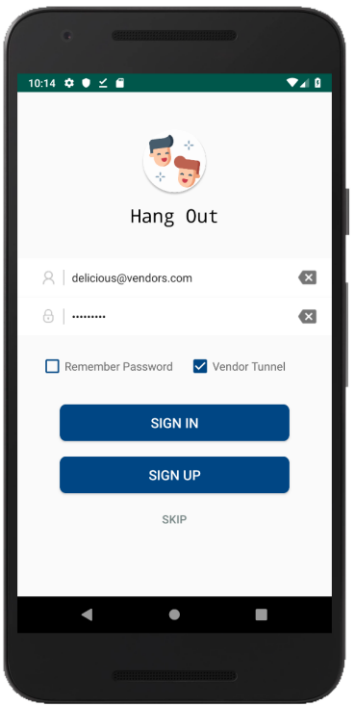
3.1.7 This is the help centre and more page. They display useful information about the software.



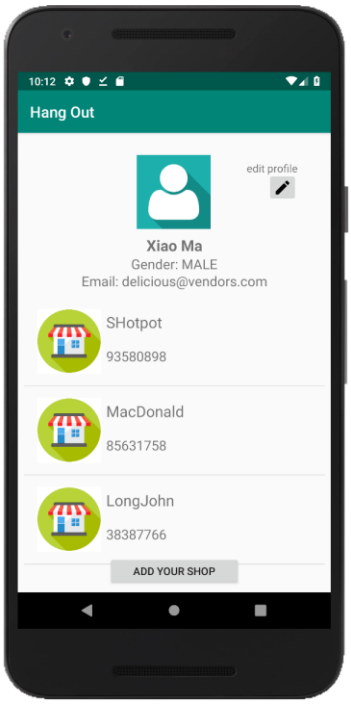
Help center

Home

3.1.8 Vendor can sign in to the application by checking the vendor tunnel at the sign in page.

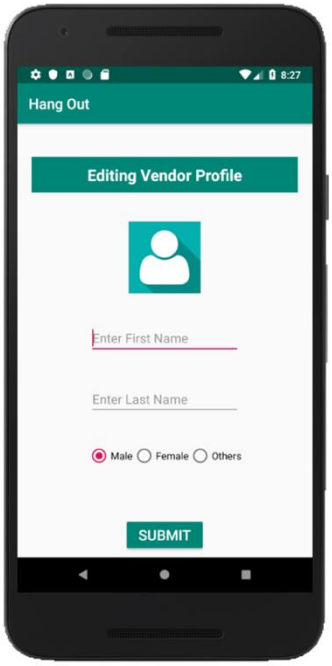


Vendor login page

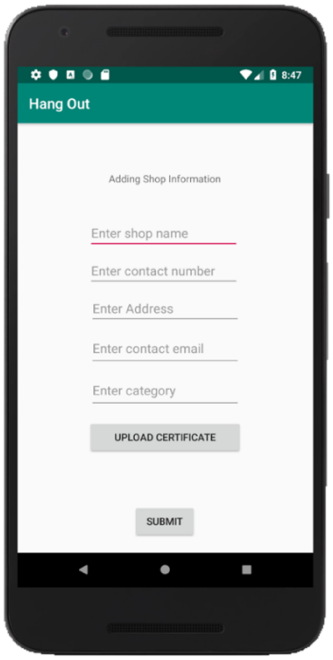


Vendor home page

3.1.9 Vendor can edit his profile and add new shops to his account.

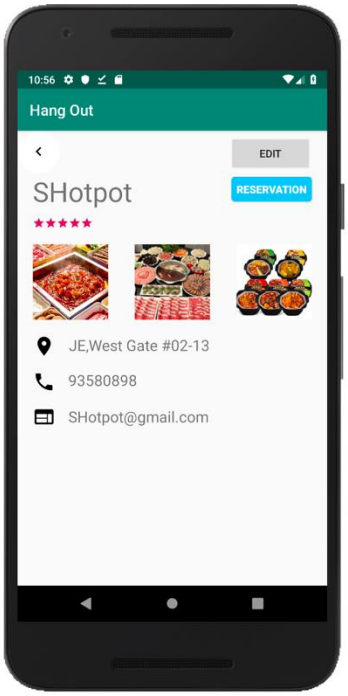


Edit profile page

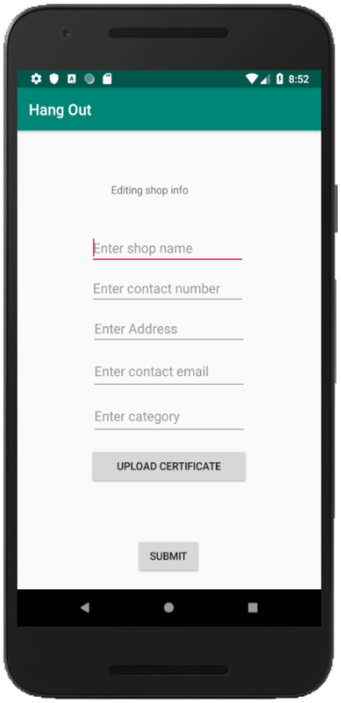


Add shop

3.1.10 Shop detail page displays shop information. Vendor can edit the information by clicking ‘edit’.

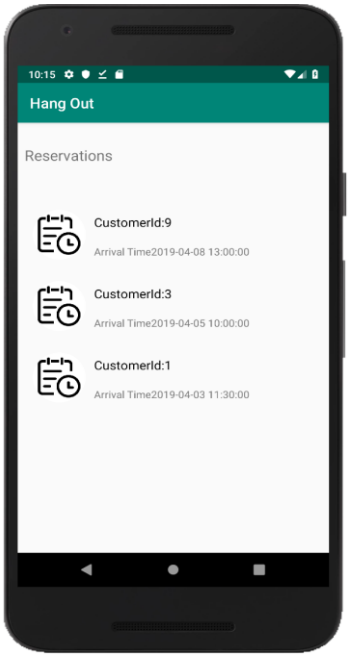


Shop detail page



Edit shop information

3.1.11 Vendor can check all the reservation of a shop by clicking ‘reservation’ at shop detail page.



View all reservations

3.2 Hardware Interfaces

The hardware interface of HOS is mainly based on the touch screen where direct manipulations are supported. Users can touch the screen to provides inputs that loosely correspond to real-world actions. Swiping, tapping, pinching, and reverse pinching are called to manipulate on-screen objects or methods. The response to the user input is conducted after when users finish typing on the virtual keyboard. Internal hardware, such as GPS modules and Wi-Fi modules.

3.3 Software Interfaces

Android Studio supports the construction of HOS front-end. Android coded in original Java grammars provides a variety of pre-built UI components. HOS is built with structured layout objects and UI controls that allow the users to operate on the graphical user interface of the application. Google Map is embedded to support HOS as the external data sources. Emulators of different Android versions are used in the test stages.

3.4 Communications Interfaces

HOS back-end servers are hosted by Spring Boot. The transmission, storage and transaction records of data is supported by HTTP protocols. Google Chrome is applied to support the external links embedded in HOS.

4. System Features and Functional Requirements

4.1 Onboarding to Application

1.1 The application shall display introductions to users with graphics and texts.
1.2 The application shall ask for permission from users to access GPS modules.
1.3 The application shall ask for permission from users to access memory.

4.2 Registration and Login

2.1The application shall display “sign in”, “sign up” and “skip” when users first use this application.
2.2 The application shall display “vendor tunnel” checkbox.
    2.2.1 The application shall perform customer sign in/up checkbox is not ticked.
2.3 The application shall be operating with the user registration.
    2.3.1 The application shall permit users to register using email address.
    2.3.2 Users shall fill in other information.
        2.3.2.1 The information shall contain the password.
        2.3.2.2 The information shall contain the name.
        2.3.2.3 The information shall contain the sex.
        2.3.2.4 The information shall contain the region.
        2.3.2.5 The information shall contain the day of birthday.
        2.3.2.6 The information shall contain the die preference.
2.4 The application shall be operating without the user registration.
    2.4.1 This must be only for customers.
    2.4.2 The customer general homepage shall show up if the user logs in without registration.
    2.4.3 The user shall not be able to access and set personal information without signing in.
    2.4.4 The user shall not be able to access supply personal recommendation without signing in.
    2.4.5 Any plan and reservation shall not be done without signing in.
2.5 The user shall be able to skip registration page and directly get into customer home page. But some functions shall not be available. (please refer to 2.3)

4.3 Recommendation Functionalities

3.1 The application shall work on users’ preference.
    3.1.1 The recommendation shall respect users’ religion.
    3.1.2 The recommendation shall respect users’ diet.
3.2 The application shall have the information of Singapore major restaurants.
    3.2.1 The application shall have the knowledges of restaurants name.
    3.2.2 The application shall have the knowledges of restaurants location.
    3.2.3 The application shall have the knowledges of restaurants rating.
    3.2.4 The application shall have the knowledges of restaurants carpark capacity.
    3.2.5 The application shall have the knowledges of restaurants photo cover.
    3.2.6 The application shall have the knowledges of restaurants email address.
    3.2.7 The application shall have the knowledges of restaurants phone number.
3.3 Recommendation function shall only be available after the user logs in.
3.4 Recommendation list item shall be clickable, which leads to the corresponding shop detail page.
3.5 The shop detail page contains the detail information of the shop.

4.4 Planning Making

4.1 The schedule shall permit users to add places to plan list.
- 4.1.1 The user must log in before utilising this function.
- 4.1.2 The user shall add place to plan at the shop detail page.
- 4.1.3 The user shall choose date and time for each place added in to the plan list.
  - 4.1.3.1 Past date shall not be available.

4.1.4 The user shall choose which plan to add in.
- 4.1.4.1 The user shall create an empty plan if no plan is created.
- 4.1.4.2 The user shall create an empty plan if no desired plan is created.

4.2 The users shall remove places from plan list.
- 4.2.1 The user shall utilise this function in the corresponding plan list.
  - 4.2.1.1 The shops in the plan history list shall not be removed.
- 4.2.2 The place removed shall be able to be added to the plan again.
  - 4.2.2.1 The user shall add place to plan at the shop detail page.

4.3 The planning function shall be open only to logged in customers.

4.4 The “plans” shall be found in Me page, which is only available after the customer has logged in.
- 4.4.1 The users shall remove the place in the “plans”.
- 4.4.2 The users shall see the plan history.

4.5 Reservation Making

5.1 The schedule shall permit users to add places to reservation list.
- 5.1.1 The user must log in before utilising this function.
- 5.1.2 The user shall add place to reservation at the shop detail page.
- 5.1.3 The user shall choose date and time for each place added in to the plan list.
  - 5.1.3.1 Past date shall not be available.
- 5.1.4 Only one reservation list is available, which contains all the reservations added.

5.2 The users shall remove places from reservation list.
- 5.2.1 The place shall be able to be reserved again.

5.3 The reservation function shall be available only to logged in users.

5.4 The “reservations” shall be found in Me page, which is only available after the customer has logged in.
- 5.4.1 The users shall see all the places added in the reservation list.
- 5.4.2 The users shall remove the place from the reservation list.

4.6 Profile Updates

6.1 The user shall be able to update profile.
- 6.1.1 The user shall be able to change avatar.
- 6.1.2 The user shall be able to change first name.
- 6.1.3 The user shall be able to change last name.
- 6.1.4 The user shall be able to change sex.
- 6.1.5 The user shall be able to change region.
- 6.1.6 The user shall be able to change day of birthday.
- 6.1.7 The user shall be able to change halal option.
- 6.1.8 The user shall be able to change vegetarian option.

6.2 All the information shall be valid after the user submit the changes.

6.3 The user shall log in with new password if he/she changed the password.

4.7 Shop Details

7.1 Shop detail shall contain shop information.
- 7.1.1 The shop information contains the shop name.
- 7.1.2 The shop information contains the shop rating.
- 7.1.3 The shop information contains the shop photo gallery.
- 7.1.4 The shop information contains the shop location.
- 7.1.5 The shop information contains the shop phone number.
- 7.1.6 The shop information contains the shop email address.
- 7.1.7 The shop information contains the shop government API carpark capacity.

7.2 The reservation shall be done on the shop detail page.
7.3 The plan shall be done on the shop detail page.

4.8 Onboarding to Application

8.1 The application shall display introductions to venders with graphics and texts.
8.2 The application shall ask for permission from venders to access memory.

4.9 Registration and Login

9.1The application shall display “sign in”, “sign up” and “skip” when users first use this application.
9.2 The application shall display “vendor tunnel” checkbox.
- 9.2.1 The application shall perform vendor sign in/up checkbox is ticked.

9.3 The application shall be operating with the user registration.
- 9.3.1 The application shall permit users to register using email address.
- 9.3.2 Users shall fill in other information including password, name and sex.

9.4 The application shall only be operating with the user logged in.
- 9.4.1 The skip button shall not be available for vendors.

4.10. Manage shops

10.1 The vendor shall be able to declare the ownership on a shop.
- 10.1.1 The certificate shall be needed for a declaration.
- 10.1.2 The shop shall be added only after receiving the permission from admin.

10.2 The vender shall be able to update shop information.
- 10.2.1 The vender shall be able to update shop name.
- 10.2.2 The vender shall be able to update shop description.
- 10.2.3 The vender shall be able to update shop location.
- 10.2.4 The vender shall be able to update shop email.
- 10.2.5 The vender shall be able to update shop phone number.
- 10.2.6 The vender shall be able to update shop photo gallery.

10.3 The vender shall be able to delete the shop.
- 10.3.1 All the shop information shall be removed.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 The application shall take within 3 seconds to load in.
5.1.2 The introduction message shall take within 1 second to load in.
5.1.3 The sign up and sign in page shall take within 1 second to load in.
5.1.4 The validation email shall take within 30 seconds to arrive.
5.1.5 The personal nickname shall take within 1 second to finish.
5.1.6 The personal avatar shall take within 2 seconds to finish.
5.1.7 Each module helping message shall take within 1 second to load in.
5.1.8 Each function module shall take within 1 seconds to load in.
5.1.9 Detail about a specific restaurant or visiting place shall take within 2 seconds to load in.
5.1.10 The research result shall take within 2 seconds to load in.
5.1.11 The schedule shall take within 0.5 second after users drag the place in.
5.1.12 The route planning shall take within 3 seconds to find short and less-traffic-jam paths based on different travel mode.
5.1.13 Social media API shall take within 2 seconds to switch to corresponding application.
5.1.14 All exception handling message shall take within 0.5 second to load in.
5.1.15 Plan records shall take within 1 second to load.
5.1.16 It shall support at most 100 people to use it at the same time.
5.1.17 The database shall store at most 1000 users with average 10 records for each user.

5.2 Safety Requirements

5.2.1 The application shall be extended in the future.
- 5.2.1.1 New user interface design and format shall be adopted.
- 5.2.1.2 New recommendation algorithm and route planning algorithm shall be adopted.
- 5.2.1.3 New features shall be adopted.
  - 5.2.1.3.1 Users shall add friends via this application and chat.
  - 5.2.1.3.2 Users shall form a group and make a plan together.
  - 5.2.1.3.3 Users shall vote for preferred place and plan.
  - 5.2.1.3.4 Users shall write reviews on the place they have been to.
  - 5.2.1.3.5 Users shall make a reservation for restaurants.
- 5.2.1.4 New API shall be added.
  - 5.2.1.4.1 Translating API shall be implemented to translate into multiple languages.
  - 5.2.1.4.2 Taxi application API shall be merged.

5.2.2 SkyForce team shall maintain the system.
5.2.3 The application shall get IOS version and desktop version when it gets popular.

5.3 Security Requirements

5.3.1 APIs shall be implemented.
- 5.3.1.1 Google map API shall be used to get the location, path, and information.

5.3.1.2 Carpark Availability API, based on real-time data from Singapore government database, shall be used to get car park capacity.
5.3.1.3 Weather Forecast API, based on real-time data from Singapore government database, shall be used to get the destination weather condition and temperature.
5.3.1.4 Telegram, WhatsApp and Facebook API shall be used to share plans.
5.3.1.5 Recommendation algorithm shall take the ranking, distance from current location, whether, personal preference into account.
5.3.1.6 Route planning algorithm shall take current time, and road traffic condition data from Singapore government database, and google map routes into account.

5.3.2 The system shall handle exceptions.
5.3.2.1 Reminding message shall display when the user signs up with an invalid email address.
5.3.2.2 Reminding message shall display when the user signs up with an email address that has been registered.
5.3.2.3 Reminding message shall display when the user inputs the wrong verification code.
5.3.2.4 Reminding message shall display when the user type two different passwords when setting or resetting the password.
5.3.2.5 Reminding message shall display when the user signs in with an incorrect password.
5.3.2.6 Reminding message shall be displayed when the user provides nickname with blank or over 50 character.
5.3.2.7 Reminding message shall display when the user who has not logged in accesses recommendation.
5.3.2.8 Reminding message shall display when the user who has not set preference accesses recommendation.
5.3.2.9 Reminding message shall be displayed when the user searches for a place that cannot be found in google map.
5.3.2.10 Reminding message shall be displayed when the user taps search with no content.
5.3.2.11 Reminding message shall display when the user taps schedule arrangements with no places liked.
5.3.2.12 Reminding message shall display when the user who drags the time slot out of limit or two slots clashes.
5.3.2.13 Reminding message shall display when the user taps route planning before arranging schedules.
5.3.2.14 Reminding message shall display when the user taps “finish” on route planning page before choosing routes.
5.3.2.15 Reminding message shall display when the user books an item that has no quantity left.

5.3.3 The application must not leak user information.
5.3.3.1 User phone number and email address shall not be seen or used by others.
5.3.3.2 User plane record shall only be used for personal recommendation.
5.3.3.3 User GPS location shall only be used for personal recommendation.
5.3.3.4 User information shall be deleted if the user account is deleted.

5.3.4 Restart shall be acceptable when failure happens

5.4 Software Quality Attributes

5.4.1 The helping message.
5.4.1.1 The helping message of the introduction shall contain three pages.
5.4.1.1.1 The first page shall say welcome user.
5.4.1.1.2 The second page shall explain where each module is, and own function.

5.4.1.1.3 The third page shall explain where the setting, user-login, helping-message are, and own function.

5.4.1.2 The helping message on navigating through each module, shall be displayed, when the user first opens each module.

5.4.1.2.1 The helping message for each icon for each place shall explain its function.

5.4.1.2.2 The helping message for schedule arrangement shall display how to drag plans to time slot.

5.4.1.2.3 The helping message for route planning shall display how to choose transportation way between two places.

5.4.1.2.4 The helping message for sharing shall display how to share to friends.

5.4.1.3 The user shall refer to the helping message button when needs help further.

5.4.1.4 The user shall tap the screen to view next helping message.

5.4.1.5 The user shall tap the screen to exit when it reaches the last page.

5.4.1.6 The ”SKIP” choice must display at the bottom for user to skip when the helping message is on.

5.4.2 Information on each target place.

5.4.2.1 Each place shall display the local car park capacity.

5.4.2.2 Each place shall display distance from the current location there.

5.4.2.3 Each place shall display the photo gallery.

5.4.2.4 Each place shall display the detailed address.

5.4.2.5 Each place shall display the contact way.

5.4.2.6 Each place shall display the customer review.

5.4.3 Free setting.

5.4.3.1 The user shall use this application with a login account or as a guest.

5.4.3.1.1 Recommendation shall be on, frequent visit records shall be stored when the user has logged in and.

5.4.3.1.2 Recommendation shall be off, visit records shall not be stored when the user has not logged in.

5.4.3.2 The user shall enable and disable certain functionalities.

5.4.3.2.1 Functionalities shall hide when the user disable them.

5.4.3.2.2 Functionalities shall display when the user enable them.

5.4.3.2.3 Functionalities shall contain speech recognition, help-message, recommendation and route-plan.

5.4.3.3 The user shall fill up personal information or leave blank after signing in.

5.4.3.3.1 The user shall fill up avatar or leave blank.

5.4.3.3.2 The user shall fill up nickname or leave blank.

5.4.3.3.3 The user shall fill up personal preference or leave blank.

5.4.3.3.4 The user shall choose gender or leave unchosen.

5.4.3.4 The user shall register at any time when he/she is using this application as a guest.

5.4.3.5 The application shall allow the user to delete the account.

5.4.3.5.1 The account information associated with the account shall be deleted in the database.

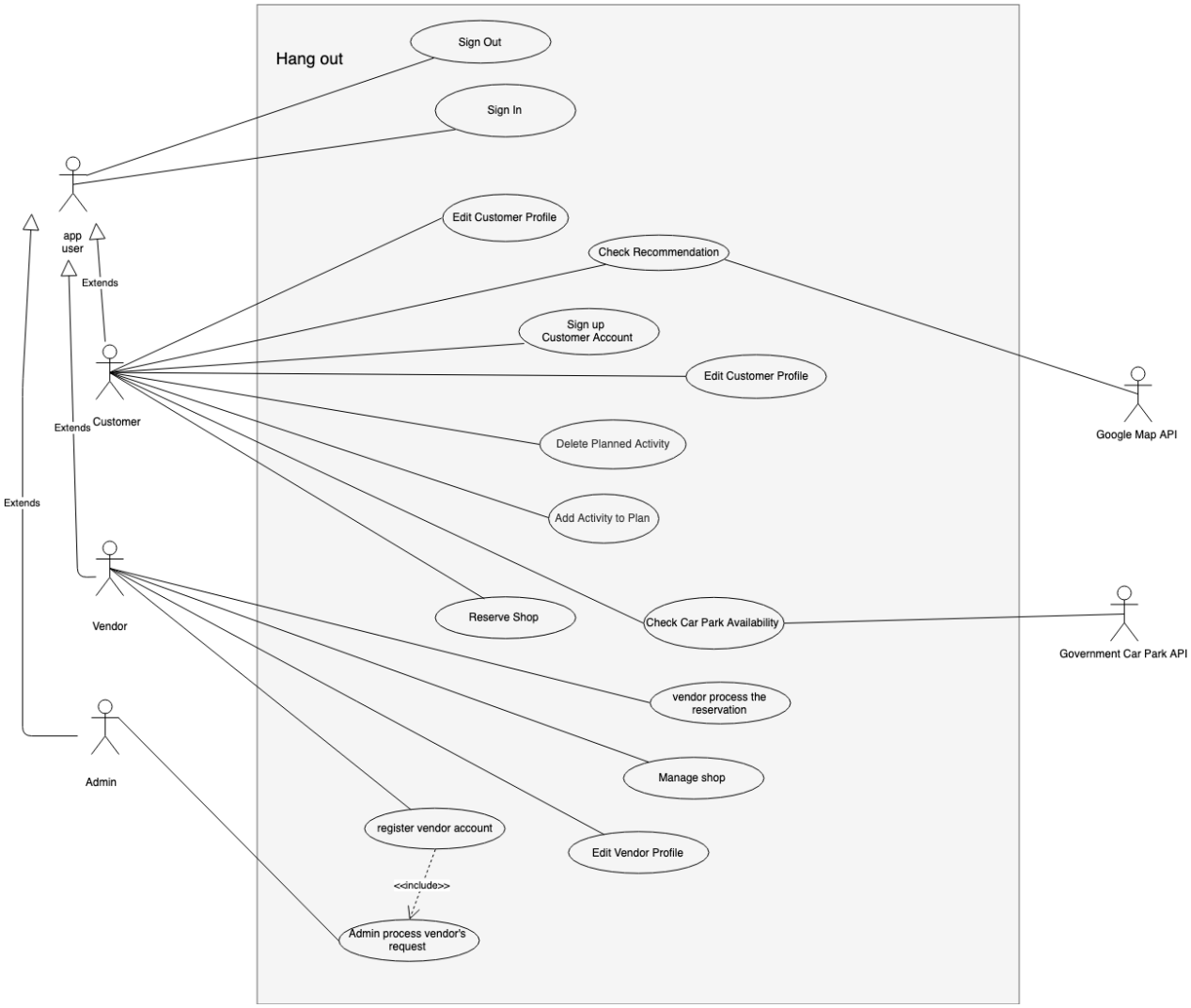
5.4.3.5.2 The frequent-routes associated with the account shall be deleted in the database.

5.4.3.5.3 The frequent visited places associated with the account shall be deleted in the database.

5.4.4 This application is free of change.

6. Use Cases

6.1 Use case diagram



6.2 Use cases list

- UC001 Sign Up Customer Account
- UC002 Sign in
- UC003 Check Recommendation
- UC004 Add Activity to Plan
- UC005 Delete Planned Activity
- UC006 Delete Plan
- UC007 Edit User Profile
- UC008 Sign out
- UC009 Register vendor account
- UC010 Login vendor account
- UC011 Check Car Park Availability
- UC012 Reserve Shop
- UC013 Manage Shop
- UC014 Edit Vendor Profile
- UC015 Vendor process the reservation

6.3 Future improvements list

- UC016 Search
- UC017 Routes Planning
- UC018 Sharing Schedule
- UC019 Reset Password
- UC020 Delete Account
- UC021 Login administration account
- UC022 Admin process the vendor’s registration

6.4 Use Case Descriptions

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC001</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Sign Up Customer Account</td></tr>
<tr><td>Created By:</td><td>Ma Xiao</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>05/02/2019</td><td>Date Last Updated:</td><td>06/04/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User (Customer) (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The app user sign up to the app as a customer and get an account</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has download the application, and
2. The user has a personal email.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">A app has successfully registered an customer account in the app.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Edge use case (seldom used)</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The user proceeds to sign up on the sign in page and choose to sign up as customer.</td><td>2. The system navigates to the sign up page.

3. The system requests the user to sign up with email, password, personal information including name, gender, birthday, regional preference, vegetarian/hahal preference.</td></tr>
<tr><td>4. The user inputs the personal email , personal information and set the account password.</td><td>5. The system create the user and save the user into backend.</td></tr>
<tr><td></td><td>6. The system reports to user that the sign up is successful.
7. The system returns to the sign in page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S4: If the user inputs an invalid email address to sign up.
1. The system displays a message “Invalid email address”.
2. The system returns to beginning of step 4.

AF-S4: If the user inputs an email address that has already been used for signing up.
1. The system displays a message “The email has been used for signing up”.</td></tr>
</table>

	2. The system returns to beginning of step 4.
Exceptions:	EX1: The user wants to cancel button during the signing up process. 1. The sign up process terminates and system returns to log in page.
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	AF-S4 is future improvement.

Use Case ID:	UC002		
Use Case Name:	Sign in		
Created By:	Ma Xiao	Last Updated By:	Li Bingzi
Date Created:	05/02/2019	Date Last Updated:	06/04/2019
Actor:	Application User (Customer or Vendor) (initiating actor)		
Description:	The app user sign in to the app		
Preconditions:	1. The user has the app. 2. The customer or vendor has an approved account.		
Postconditions:	The user has successfully signed in to the application.		
Priority:			
Frequency of Use:	Edge used case(seldomly used)		
Flow of Events:			

Actor Steps	System Steps
1. The user proceeds to sign in after the welcome message.	2. The system navigates to the Sign in page.
3. The user chooses to sign in as customer or vendor.	

<table>
<tr><td></td><td>
<table>
<tr><td>4. If the user selects the mode “Customer”, then “Sign in as Customer” is performed.

5. If the user selects the mode “Vendor”, then “Sign in as Vendor” is performed.</td><td></td></tr>
<tr><td></td><td>7. The system verifies the user’s account and password and the user is signed in.</td></tr>
<tr><td colspan="2">Sign in as Customer:
1. The system requests the user to enter email and the password.
2. The user inputs the email and the password.

Sign in as Vendor:
1. The system requests the user to enter email and the password.
2. The user inputs the email and the password.</td></tr>
</table>
</td></tr>
<tr><td>Alternative Flows:</td><td>AF-S7: If the user inputs an invalid email address that has not been used to sign up or the user account id that does not exist
1. The system displays an error message accordingly.
2. The systems returns to step 4, 5 or 6 depends on the sign in mode.

AF-S7: If the user inputs an invalid password.
1. The system displays a message “The password is incorrect, please try again”
2. The systems returns to step 4, 5 or 6 depends on the sign in mode.</td></tr>
<tr><td>Exceptions:</td><td></td></tr>
<tr><td>Includes:</td><td></td></tr>
<tr><td>Special Requirements:</td><td></td></tr>
<tr><td>Assumptions:</td><td></td></tr>
<tr><td>Notes and Issues:</td><td></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC003</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Check Recommendation</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>06/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User (initiating user)
Google API(participating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">System recommend places and restaurants according to user’s information. The user check those recommendation in the home page.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has downloaded the app.
2. The user has an account via UC001.
3. The user has signed in via UC002.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The user is able to add these recommendations to schedule.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Scenario Use Case</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td></td><td>1.The system fetch user info data from database of the app.

2. The system calculates the restaurants and visiting places for recommendation.</td></tr>
<tr><td>3. The user enters the recommendation page.</td><td>4. System displays recommendations in form of pictures and text on the landing page.</td></tr>
<tr><td>5. If the user goes into one of the recommendations.</td><td>6. System enters the corresponding page of the item.</td></tr>
<tr><td>7. The user chooses to go back.</td><td>8. The system goes back to the home page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S3: The user click the recommendation on the home page.
1. The system displays the shop detailed information using Google API.
2. The system returns to home page after user click return button.</td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
</table>

Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC004		
Use Case Name:	Add Activity to Plan		
Created By:	Ma Xiao	Last Updated By:	Li Bingzi
Date Created:	10/02/2019	Date Last Updated:	06/04/2019
Actor:	Application User		
Description:	The user adds the activity for the schedule planning.		
Preconditions:	1. The user has downloaded the app. 2. The user has an account. 3. The user has signed in.		
Postconditions:	1. The user adds the activity he/she likes and the activity is ready to be planned into the schedule.		
Priority:			
Frequency of Use:	Daily Use Scenario		

<table>
<tr><td>Flow of Events:</td><td>
<table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The user picks an outing activity in the recommended list or in the search result.</td><td>2. They system displays the detailed information about the activity and gives user a “add to plan” choice to add the activity to the plan list.</td></tr>
<tr><td>3. The user proceeds to add to existing plan.</td><td>4. The system displays all the existing plans of the user.
5. The system provide the choice of create a new plan.</td></tr>
<tr><td>5. The user chooses the plan he/she wants to add the activity into.</td><td>6. The system prompts the user to choose a time slot.</td></tr>
<tr><td>7. The user chooses the time slot.</td><td>8. If successful, the system informs the user the add activity is successful.
9. The system stays in current page.</td></tr>
</table>
</td></tr>
<tr><td>Alternative Flows:</td><td>AF-S2: If the user has already added this activity to the liked activity list
1. The system does not display the “liked” choice
2. The system displays the “remove from liked” choice
3. The process returns to the step 1, and the user will see the “liked” choice in the activities that he/she has not added to liked activity list.

AF-S3: If user choose to create a new plan.
1. The system prompts user to enter plan name.
2. The user enters the new plan name.
3. The system creates a new plan and add the activity into the new plan.

AF-S6: If the action failed
4. The system displays the failure message.
5. The process returns to step 3.</td></tr>
<tr><td>Exceptions:</td><td></td></tr>
<tr><td>Includes:</td><td>UC003</td></tr>
<tr><td>Special Requirements:</td><td></td></tr>
<tr><td>Assumptions:</td><td></td></tr>
<tr><td>Notes and Issues:</td><td></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC005</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Deleted Planned Activity</td></tr>
<tr><td>Created By:</td><td>Ma Xiao</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>10/02/2019</td><td>Date Last Updated:</td><td>06/04/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User</td></tr>
<tr><td>Description:</td><td colspan="3">The user delete the activity he/she added to the plan before.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has downloaded the app.
2. The user has an account via UC001.
3. The user has signed in via UC002.
4. The user has added the activities to plan via UC004.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">1. The activity is removed from the plan.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Use Scenario</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The user navigates to the “my plan” page.
2. The user navigates to a specific plan.</td><td>3. They system displays the activities that the user added into the plan.</td></tr>
<tr><td>4. The user chooses the activity that he/she wants to remove from the list.</td><td>5. The system displays the liked activities list without the activity being deleted and the delete activity is successful.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3"></td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC006</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Delete Plan</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>06/02/2019</td><td>Date Last Updated:</td><td>06/04/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The user delete a plan.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has downloaded the app.
2. The user has an account via UC001.
3. The user has signed in via UC002.
4. The user has a plan via UC004.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The plan is deleted in database</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Use Scenario</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The user navigates to “view my plan” page.</td><td></td></tr>
<tr><td>2. The user chooses to delete the plan he/she wants to delete in the list.</td><td>3. System displays “ Do you really want to discard this plan?” and offers yes/no option.</td></tr>
<tr><td>4. The user selects “yes”.</td><td>5.1 System delete the plan in the database.
5.2 System returns to “view my plan” page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S4: The user selects “no”
1. System returns to step 1.</td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC007</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Edit User Profile</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>06/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The user edit his/her profile information on profile page</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has downloaded the app.
2. The user has an account via UC001.
3. The user has signed in via UC002.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">1. Changes of the user information appears on his/her profile page.
2. The app is able to use new data to provide personal recommendation.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Necessary Case</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1.The user proceeds to edit personal information on profile page.</td><td>2.1 System displays an information edit page.
2.2 System displays the information fill-in in form of choices
2.3 The system displays the original information.</td></tr>
<tr><td>3. The user fill in new information.</td><td>4. system allows user to confirm the action in order to proceed.</td></tr>
<tr><td>5. The user confirms.</td><td>6. System displays “ successfully edited” for 2 seconds.
7. System returns to the profile page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3"></td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
</table>

Special Requirements:	
Assumptions:	
Notes and Issues:	

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC008</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Sign out</td></tr>
<tr><td>Created By:</td><td>Ma Xiao</td><td>Last Updated By:</td><td>Ma Xiao</td></tr>
<tr><td>Date Created:</td><td>06/02/2019</td><td>Date Last Updated:</td><td>11/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User</td></tr>
<tr><td>Description:</td><td colspan="3">The app user sign out his/her account</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has an account and,
2. The user has already signed in to the application via UC002</td></tr>
<tr><td>Postconditions:</td><td colspan="3">1. The user continue using the application without signing in
2. The system shall not perceive any information about the user</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Edge user case (seldomly used)</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The signed in user navigates to the user profile page.
2. The user choose to “Sign Out”.</td><td>3. The system displays a widget with message “Confirm Sign Out?” and two choice buttons “Confirm” and “Cancel”.</td></tr>
<tr><td>4. The user clicks the “Confirm” button.</td><td>5. The sign out is successful.
6. The system displays the sign in page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S4: If the user click “Cancel” to cancel the signing out process.
1. The system displays the user profile page with the user’s information.
2. The sign out process returns to step 2.</td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC009</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Register Vendor Account</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>23/02/2019</td><td>Date Last Updated:</td><td>06/04/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Vendor (initiating actor)
Admin (participating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The vendor app user request register account with a shop license. Admin approve or reject the request.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">The vendor has the app.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The vendor is able to log in to the app.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Edge Case</td></tr>
<tr><td>Flow of Events:</td><td colspan="3">
<table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The vendor proceed to register.</td><td></td></tr>
<tr><td></td><td>2. The system requests the vendor with name, email and password.</td></tr>
<tr><td>3. The vendor fills in email and password.</td><td>4. The system sends the request along license to Admin.</td></tr>
<tr><td>5. Admin receives the request.</td><td>6. The system prompts Admin to select Accept or Reject using the included use case UC023 Admin Approve the Vendor’s Registration.</td></tr>
</table>
Accept:
<table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td></td><td>1. The system stores the account data in database.</td></tr>
<tr><td></td><td>2. The system updates the vendor via email that the account has been registered.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>Reject:<table><tr><th>Actor Steps</th><th>System Steps</th></tr><tr><td></td><td>1. The system updates the vendor via email that the registration has been rejected.</td></tr></table></td></tr>
<tr><td>Alternative Flows:</td><td></td></tr>
<tr><td>Exceptions:</td><td></td></tr>
<tr><td>Includes:</td><td></td></tr>
<tr><td>Special Requirements:</td><td></td></tr>
<tr><td>Assumptions:</td><td></td></tr>
<tr><td>Notes and Issues:</td><td></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC010</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Login Vendor Account</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>23/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Vendor (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The vendor login to the app.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The vendor has the app.
2. The vendor has an approved account via UC009.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">1. The vendor is able to upload items and manage account.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Scenario</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The vendor navigates to Login page and choose the vendor tunnel.</td><td>2. The system requests the vendor to Login with email address and password.</td></tr>
<tr><td>3. The vendor types in email address and password.</td><td>4. The system verifies the account email and password.</td></tr>
<tr><td></td><td>5. The system navigates to home page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S4: If the user inputs an invalid email or password.
1. The system displays a message “The password or the email is incorrect, please try again”
2. The systems returns to step 2.</td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC011</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Check Carpark Availability</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>23/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Customer (initiating actor)
Government Car Park API</td></tr>
<tr><td>Description:</td><td colspan="3">Customer check the nearest car park and its availability in shop detail page.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The customer has the app.
2. The customer has logged in via UC002.</td></tr>
<tr><td>Postconditions:</td><td colspan="3"></td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Scenario</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The user navigates to shop detail page.</td><td>2. The system gets real time information of the nearby carpark including: Car Park name, and current availability.</td></tr>
<tr><td></td><td>3. The system displays the name and availability on the shop detail page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3"></td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC012</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Reserve Shop</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>23/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">App user (initiating actor)
Vendor (participating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The app user sends reservation to the vendor via the app.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The app user has logged in via UC002.
2. The corresponding vendor has logged in via UC010.
3. The vendor has put an item on the app for reservation via UC013.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The vendor is able to reject or accept the reservation.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Scenario</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The app user navigates to a certain item page.</td><td></td></tr>
<tr><td>2. The app user proceeds to reservation.</td><td>3. The system prompts the user to enter reservation information.</td></tr>
<tr><td>4. The user enters reservation information.</td><td>5. The system prompts user to confirm or cancel the reservation.</td></tr>
<tr><td>6. The user confirms the reservation.</td><td>7. The system sends the reservation to the vendor who put the item on the app.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3"></td></tr>
<tr><td>Exceptions:</td><td colspan="3">EX-S6: If the user cancels the reservation.
Exit the use case.</td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC013</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Manage Shop</td></tr>
<tr><td>Created By:</td><td>Ma Xiao</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>13/02/2019</td><td>Date Last Updated:</td><td>06/04/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Vendor (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The vendor add the items they offer to other users to reserve.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The vendor has an account on the application
2. The vendor has logged in to the application as vendor using vendor account.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The vendor has successfully added a shop he/she offer to other users and the item can be seen and reserved by all users.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Use Case (frequently used)</td></tr>
<tr><td>Flow of Events:</td><td colspan="3">
<table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The vendor proceeds to add or edit the shop.</td><td>2. The system navigates to the add/edit shop page.</td></tr>
<tr><td>3. If the vendor chooses to add shop, then the Add Shop sub-flow is performed.

4. If the vendor chooses to edit shop, then the Edit Shop sub-flow is performed.</td><td></td></tr>
<tr><td colspan="2">Add Shop
1. The vendor chooses to add a shop.
2. The system requires the input fields: shop name, contact number, contact email, shop address, category, certificate.
3. The vendor key in the required information to add a new shop.
4. The system add a new shop from the acquired information.

Edit Shop
1. The vendor chooses a shop and chooses to edit the shop.
2. The system requires the input fields: updated shop name, updated contact number, updated contact email, updated shop address, updated category and updated shop certificate.
3. The vendor key in the required information to edit shop information.
4. The system edit the shop using the acquired information.</td></tr>
</table>
</td></tr>
</table>

Alternative Flows:	AF-AddShop-S2: If the vendor is adding a shop and does not input a required field. 1. The system displays a message “This field is required”. 2. The system returns to beginning of step 5. AF-EditShop-S2: If the vendor is adding a shop and does not input a required field. 1. The system displays a message “This field is required”. 2. The system returns to beginning of step 5.
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC014</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Edit Vendor Profile</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>06/04/2019</td><td>Date Last Updated:</td><td>06/04/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Vendor (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The vendor edit his/her profile</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The vendor has an account on the application
2. The vendor has logged in to the application as vendor using vendor account.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The vendor has successfully</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Use Case (frequently used)</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table><tr><th>Actor Steps</th><th>System Steps</th></tr><tr><td>1.The vendor proceeds to edit personal information on profile page.</td><td>2.1 System displays an information edit page.
2.2 System displays the information fill-in in form of text fields and choices, including name and gender.</td></tr><tr><td>3. The user fill in new information.
4. The user submits.</td><td>5. System displays “ successfully edited” for 2 seconds.
6. System returns to the profile page.</td></tr></table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3"></td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
<tr><td></td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC015</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Vendor process the reservation</td></tr>
<tr><td>Created By:</td><td>Ma Xiao</td><td>Last Updated By:</td><td>Ma Xiao</td></tr>
<tr><td>Date Created:</td><td>13/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Vendor</td></tr>
<tr><td>Description:</td><td colspan="3">The vendor processed a user’s reservation of an item offered in his/her shop. If the vendor accept the reservation, the uses receives the reservation success message. If the vendor reject the reservation, the user receives the reject message and the reason for reject.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The vendor has an account on the application
2. The vendor has logged in to the application as vendor using vendor account.
3. A user has reserved an item in the vendor’s shop</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The vendor has successfully processed the item reservation and the user will get the notification about the result of their reservation.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Use Case (frequently used)</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The vendor proceeds to manage reservation on the reservations page.</td><td>2. The system navigates to the reservations page (reservations page is only visible to vendor that provide the reserved item)</td></tr>
<tr><td>3. The vendor chooses a reservation to process and check the reservation details.</td><td>4. The system gives vendor two choices, APPROVE and REJECT.

4. The system shows the message saying that “the reservation is successfully confirmed” and the reservation is moved to the “accepted reservations” page.

5. The system send the user who made the reservation an notification saying that “The reservation is confirmed, you are good to go, enjoy!”</td></tr>
<tr><td>5. If the vendor selects the choice APPROVE, then the Approve</td><td></td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>Customer Reservation sub-flow is performed.

6. If the vendor selects the choice REJECT, then the Reject Customer Reservation sub-flow is performed.</td><td></td></tr>
<tr><td colspan="2">Approve Customer Reservation
1. The vendor chooses to approve the reservation.
2. The system shows the message “the reservation is successfully confirmed” and the reservation is moved to the “accepted reservations” page.
3. The system sends the user who made the reservation a notification saying that “The reservation is confirmed, you are good to go, enjoy!”

Reject Customer Reservation
1. The vendor chooses to reject the reservation.
2. The system shows the message “the reservation is rejected”.
3. The system sends the user who made the reservation a notification “The reservation cannot be approved, maybe you can try to make another reservation in different time”</td></tr>
</table>
</td></tr>
<tr><td>Alternative Flows:</td><td></td></tr>
<tr><td>Exceptions:</td><td></td></tr>
<tr><td>Includes:</td><td></td></tr>
<tr><td>Special Requirements:</td><td></td></tr>
<tr><td>Assumptions:</td><td></td></tr>
<tr><td>Notes and Issues:</td><td>Approve and Reject is included in future improvement of the app.</td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC016</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Search</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>06/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The user searches for location/activity.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has downloaded the app.
2. The user has an account via UC001.
3. The user has signed in via UC002.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">1. The user enters the information page of the item searched.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Use Scenario</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table><tr><th>Actor Steps</th><th>System Steps</th></tr><tr><td>1. The user click the search box on top of the home page and enter name of the location/activity .</td><td>2. If the System has found the location/activity, System presents the results in order of relevance below the search box.</td></tr><tr><td>3. The user goes into the desired location/activity.</td><td>4. System enters the information page of the item searched.</td></tr></table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S2: If no such place is found
1. Display a message “ No results, please try another”.
AF-S3: If the user chooses to go back.
1. System goes back to the home page.</td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC017</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Routes Planning</td></tr>
<tr><td>Created By:</td><td>Ma Xiao</td><td>Last Updated By:</td><td>Ma Xiao</td></tr>
<tr><td>Date Created:</td><td>06/02/2019</td><td>Date Last Updated:</td><td>11/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User
Google Map API
Government Car Park Availability API
Government Weather API</td></tr>
<tr><td>Description:</td><td colspan="3">The user plans some appropriates routes for his/her hang out places.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">The user has made a plan with places to go and the planned time via UC006 .</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The system generates several ways to get to all the places user wants to go.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily used case (frequently used)</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The user navigates to the “View My Plans” page.

2. The user picks one outing plan and chose “Plan Routes”.</td><td>3. The system gets the user’s preferred transportation method and all the information about the user’s plan including the places and planned time to go to every place.

4. The system calls Google Map API, Government Car Park Availability API, and Government Weather API to generate feasible routes plans.

5. The system displays all the feasible plans.</td></tr>
<tr><td>6. The user chooses one of the plans and choose “Take it as My Routes”</td><td>7. The routes planning is successful and the system displays the chosen routes inside the user’s hang out plan.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S2: If the user does not have any hangout plans yet.
1. The system displays “Hey, find out some outing plan first”
2. The process returns to step 1</td></tr>
<tr><td>Exceptions:</td><td colspan="3">EX1: The system cannot generate routes for user that fits the user’s schedule (for example: 10 minutes from NTU to Changi Airport)</td></tr>
</table>

	1. The system displays “Please check your scheduled time and places again. It seems not a feasible plan” 2. The system returns to the My plans page.
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC018</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Sharing Schedule</td></tr>
<tr><td>Created By:</td><td>Ma Xiao</td><td>Last Updated By:</td><td>Ma Xiao</td></tr>
<tr><td>Date Created:</td><td>06/02/2019</td><td>Date Last Updated:</td><td>11/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User</td></tr>
<tr><td>Description:</td><td colspan="3">The user shares a hang-out plan to other social media platform</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has made a plan with places to go and the planned time via UC004.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">1. The outing plan is shared to the desired social media</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily use case (frequently used)</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The user navigates to the “View My Plans” page.

2. The user picks one outing plan and choose to “Share”</td><td>3. The systems pops up a widget from the bottom of the screen and shows social media platforms for user to choose.</td></tr>
<tr><td>4. The user chooses a social media to share the plan.</td><td>5. The systems jumps to the social media the user chooses.</td></tr>
<tr><td>6. The user choose a particular contact or group to share the plan.</td><td>7. The system forwards the plan to the target the user chooses and the sharing is successful.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S4: If the user clicks “Cancel” in the sharing widget.
1. The system returns to step 1.</td></tr>
<tr><td>Exceptions:</td><td colspan="3">EX1: If the user does not have any social media installed on the device.
1. The sharing process cannot be proceed and is terminated.</td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
<tr><td>Assumptions:</td><td colspan="3"></td></tr>
<tr><td>Notes and Issues:</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC019</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Reset Password</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>06/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Application User (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">The user reset password of his/her account.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">1. The user has downloaded the app.
2. The user has an account via UC001.
3. The user has signed in via UC002.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">1. The password change is recorded in database.
2. The account is ready for signing in with new password.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Edge Case</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The user navigates to profile page.</td><td></td></tr>
<tr><td>2. The user opts to change password.</td><td>3. System display a page, require the user to enter current password.</td></tr>
<tr><td>4. The user enters current password.</td><td>5. If the password is correct, display second page for user to enter new password.</td></tr>
<tr><td>6. The user enters new password twice.</td><td>7.If the entered passwords for two times are the same, system changes the password in database.
8.1 System displays message to inform user the password is changed.
8.2 System returns to the profile page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S5: If the password entered is incorrect
1. Display the message “Incorrect password”
2. System returns to step 3.
AF-S6: If two passwords are not the same
1. Display the message “Inconsistent password”
2. System returns to step 5.</td></tr>
</table>

	AF-S6: : If the new password is not valid 1. Display the message “Invalid password. 2. System returns to step 5.
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC020		
Use Case Name:	Delete Account		
Created By:	Ma Xiao	Last Updated By:	Ma Xiao
Date Created:	06/02/2019	Date Last Updated:	11/02/2019
Actor:	Application User		
Description:	The user delete his/her account and the system will delete all records about this account.		
Preconditions:	1. The user has an account on the app 2. The user has signed in to the account		
Postconditions:	1. The app user can not use the deleted account to sign in 2. All the information about this account is deleted from the system’s database.		
Priority:			
Frequency of Use:			

<table>
<tr><td>Flow of Events:</td><td>
<table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The signed in user navigates to the user profile page.

2. The user chooses the “Delete Account”.</td><td>3. The system displays a warning message “Are you sure to delete this account? Note that all the information about this account will be deleted” and gives user two choices, “Confirm” and “Cancel”.</td></tr>
<tr><td>4. The user clicks “Confirm”.</td><td>5.1 The system send the verification code to the email that the user used to sign up for this account.
5.2 The system displays the user’s account ID and two input fields requiring the password of the account and the verification code.</td></tr>
<tr><td>6. The user inputs the correct account password and verification code and choose “Confirm Delete”.</td><td>7. The delete account is successful. The system displays the sign in page.</td></tr>
</table>
</td></tr>
<tr><td>Alternative Flows:</td><td>AF-S4: If the user clicks the “Cancel” button:
1. The system displays the user profile page with the account information.
2. The delete account process return to step 1.

AF-S6: If the account credentials the user inputs is incorrect.
1. The system displays the message “Incorrect account password”
2. The system returns to step 5.

AF-S6: If the verification code the user inputs is incorrect.
1. The system displays the message “Incorrect verification code”
2. The system returns to step 5.</td></tr>
<tr><td>Exceptions:</td><td>EX1: If the user inputs incorrect account password for three times.
1. The system displays “Delete account failed.”
2. The delete account process terminates and the system displays the user profile page with the information about the signed in account.</td></tr>
<tr><td>Includes:</td><td></td></tr>
<tr><td>Special Requirements:</td><td></td></tr>
<tr><td>Assumptions:</td><td></td></tr>
<tr><td>Notes and Issues:</td><td></td></tr>
</table>

<table>
<tr><td>Use Case ID:</td><td colspan="3">UC021</td></tr>
<tr><td>Use Case Name:</td><td colspan="3">Login Administration Account</td></tr>
<tr><td>Created By:</td><td>Li Bingzi</td><td>Last Updated By:</td><td>Li Bingzi</td></tr>
<tr><td>Date Created:</td><td>23/02/2019</td><td>Date Last Updated:</td><td>23/02/2019</td></tr>
<tr><td>Actor:</td><td colspan="3">Admin (initiating actor)</td></tr>
<tr><td>Description:</td><td colspan="3">Admin logs into the system.</td></tr>
<tr><td>Preconditions:</td><td colspan="3">The admin has a valid admin account.</td></tr>
<tr><td>Postconditions:</td><td colspan="3">The admin is able to approve vendor’s registration request.</td></tr>
<tr><td>Priority:</td><td colspan="3"></td></tr>
<tr><td>Frequency of Use:</td><td colspan="3">Daily Scenario</td></tr>
<tr><td>Flow of Events:</td><td colspan="3"><table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The vendor navigates to Login page.</td><td>2. The system requests the vendor to Login with an ID and a password.</td></tr>
<tr><td>3. The vendor types in ID and password.</td><td>4. The system verifies the account ID and password.</td></tr>
<tr><td></td><td>5. The system navigates to landing page.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows:</td><td colspan="3">AF-S4: If the user inputs an invalid email address that has not been used to sign up or the user account id that does not exist
1. The system displays a message “Email address not found” or “user account not found”.
2. The systems returns to step 2.

AF-S4: If the user inputs an invalid password.
1. The system displays a message “The password or the account id is incorrect, please try again”
2. The systems returns to step 2.</td></tr>
<tr><td>Exceptions:</td><td colspan="3"></td></tr>
<tr><td>Includes:</td><td colspan="3"></td></tr>
<tr><td>Special Requirements:</td><td colspan="3"></td></tr>
</table>

Assumptions:	
Notes and Issues:	

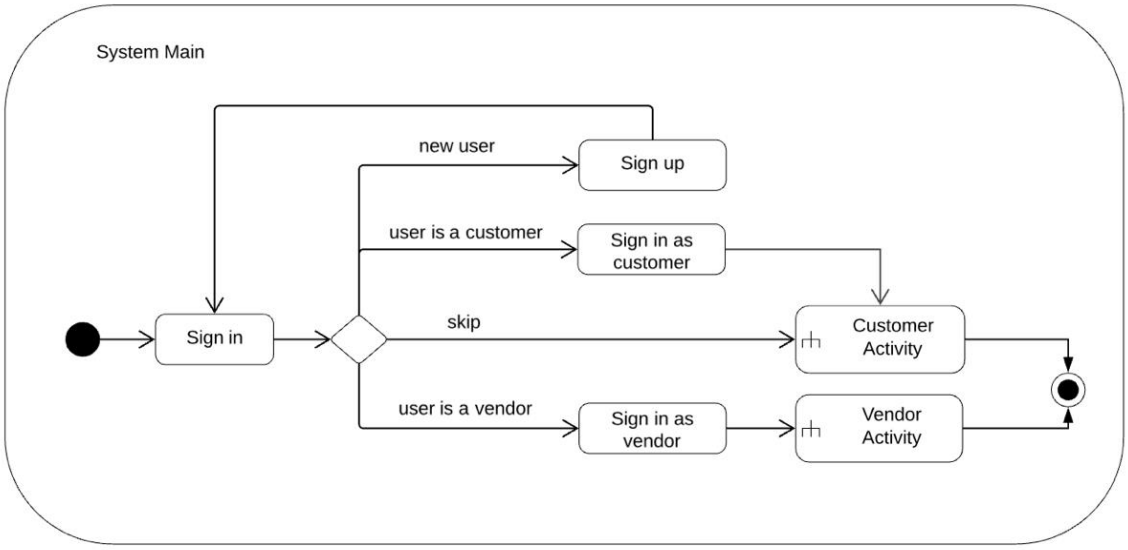
Use Case ID:	UC022		
Use Case Name:	Admin process the vendor's registration		
Created By:	Ma Xiao	Last Updated By:	Ma Xiao
Date Created:	13/02/2019	Date Last Updated:	23/02/2019
Actor:	Admin		
Description:	The admin process a vendor's registration. The admin can either approve the registration or reject the registration.		
Preconditions:	1. The admin has logged in to the application as admin using admin account. 2. A vendor has a registration request.		
Postconditions:	The admin has successfully processed the vendor's reservation and the vendor will get the notification about the registration result. If the registration has been approved, the vendor can begin the business on the platform. If the registration has been rejected, the reason is given and the vendor should make the registration again		
Priority:			

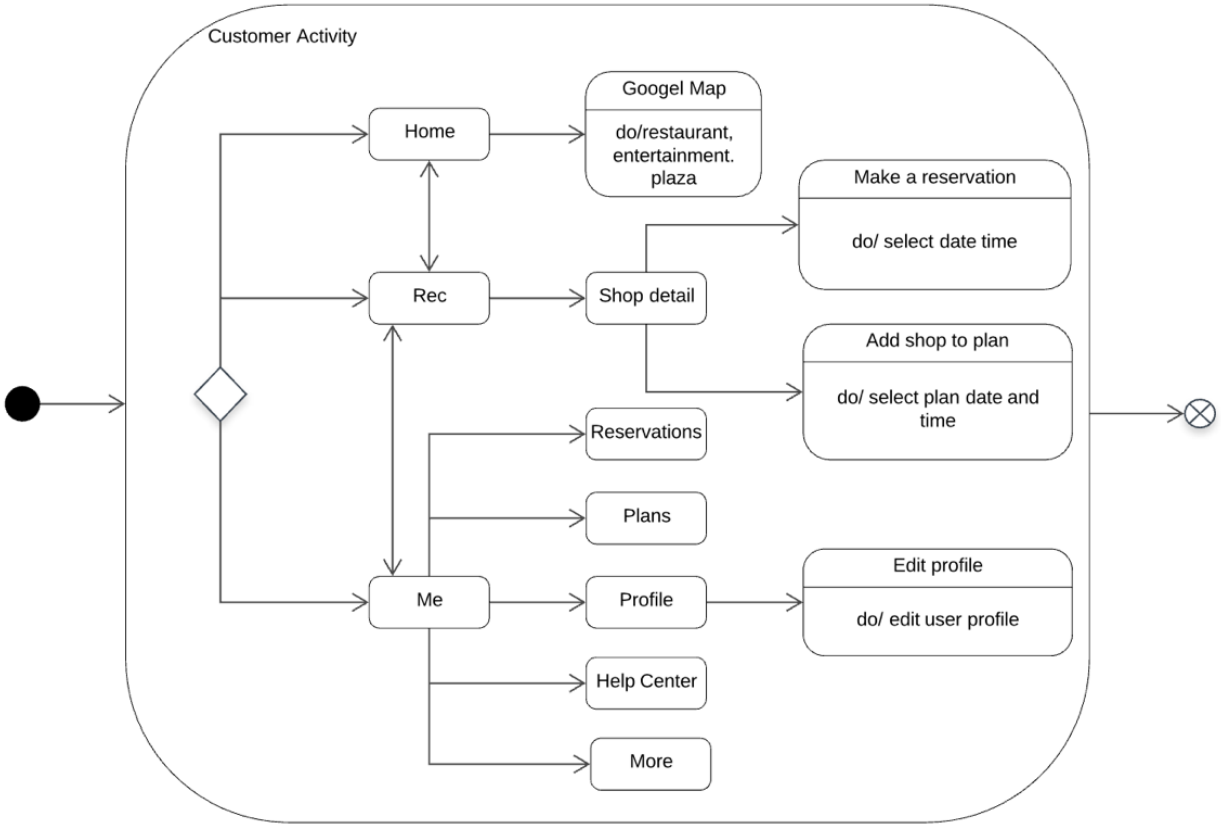
<table>
<tr><td>Frequency of Use:</td><td>Daily Use Case (frequently used)</td></tr>
<tr><td>Flow of Events:</td><td>
<table>
<tr><th>Actor Steps</th><th>System Steps</th></tr>
<tr><td>1. The admin proceeds to vendor registration management page.</td><td>2. The system navigates to the vendor registration management page (the page is only visible to admin).</td></tr>
<tr><td>3. The admin picks up a vendor registration and check the details of the registration.</td><td>4. The system gives the admin two choices APPROVE and REJECT.</td></tr>
<tr><td>5. If the admin selects the activity APPROVE, then the Approve Vendor Registration sub-flow is performed.

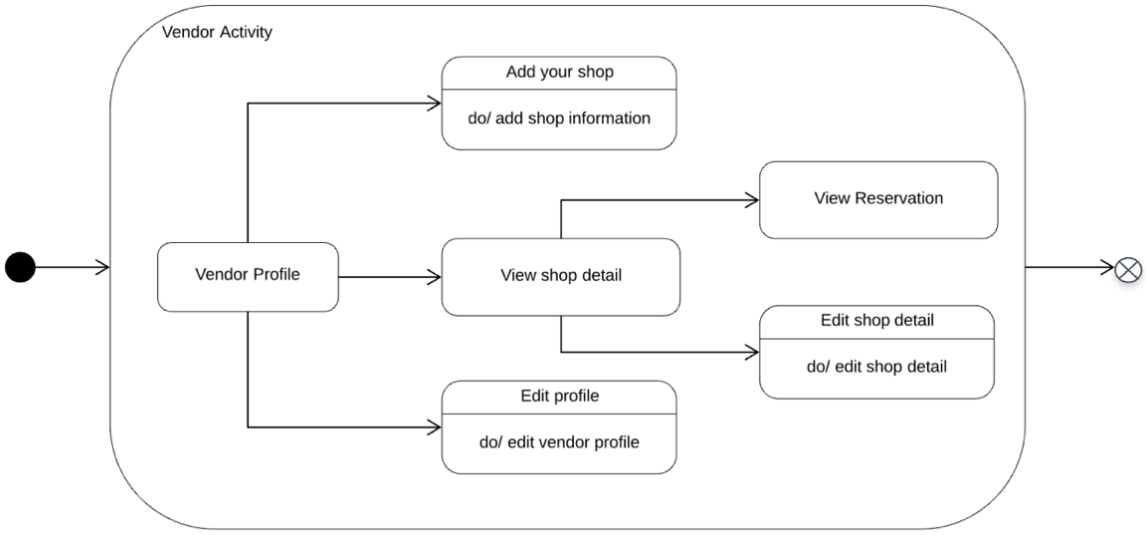
6. If the admin selects the activity REJECT, then the Reject Vendor Registration sub-flow is performed.</td><td></td></tr>
<tr><td colspan="2">Approve Vendor Registrations
1. The admin chooses to approve the vendor registration.
2. The system sends the approval message to the vendor
3. The vendor will receive the approval message and the vendor can begin their online business in the system.

Reject Vendor Registrations
1. The admins chooses to reject the vendor registration.
2. The system sends the reject message to the vendor
3. The vendor will receive the reject message and the rejection reason.</td></tr>
</table>
</td></tr>
<tr><td>Alternative Flows:</td><td></td></tr>
<tr><td>Exceptions:</td><td></td></tr>
<tr><td>Includes:</td><td></td></tr>
<tr><td>Special Requirements:</td><td></td></tr>
<tr><td>Assumptions:</td><td></td></tr>
<tr><td>Notes and Issues:</td><td></td></tr>
</table>

6.5 Dialog Map







6.6 Testing

Both blackbox testing and whitebox testing are used to test our application.

Black box testing is used to test software functionality without knowing the internal structure of the system. Major test cases are carried out to test system behaviour and performances. This is to analysis client requirements and specifications.

Whitebox is used to do unit testing which will test the individual component of related units. We use this whitebox testing techniques to verify that the code does what is intended to do and analysis the internal implementation of the system. We choose different inputs to exercise paths through the code and determines the appropriate outputs.

Test case 1. User sign in and sign up

Test ID	**Description**	**Expected Result**	**Actual Result**
1	1. key in correct customer name 2. key in correct password	Successfully login as customer. Go to customer home page.	pass
2	1. key in correct customer name 2. key in wrong password	Deny login. Display “Incorrect customer account or password”.	pass

3	1. key in wrong customer name 2. key in wrong password	Deny login. Display "Incorrect customer account or password".	pass
4	1. key in correct customer name 2. key in correct password 3. check 'vendor tunnel'	Deny login. Display "Incorrect vendor account or password".	pass
5	1. key in correct vendor name 2. key in correct password 3. check 'vendor tunnel'	Successfully login as vendor. Go to vendor home page.	pass
6	1. click skip	Go to customer home page.	pass

Test case 2. Make a reservation

Precondition: user is at a shop's detail page.

Test ID	**Description**	**Expected Result**	**Actual result**
7	1. clicks reserve button 2. select a date in the past	Request is denied.	pass
8	1. clicks reserve button 2. select a date in the future 3. select a time 4. click submit 5. go to 'me' page 6. click 'Reservations'	Successfully made the reservation. Toast message "made a reservation successfully". User can check the new reservation at 'Me —> Reservations' page.	pass
9	1. clicks reserve button 2. select a date 3. clicks 'cancel'	Request is not completed. Go to shop detail page.	pass
10	1. clicks reserve button 2. select a date in the future 3. select a time 4. clicks 'cancel'	Request is not completed. Go to shop detail page.	pass

Test case 3 Add a shop to a plan

Precondition: user is at a shop's detail page.

Test ID	Description	Expected Result	Actual result
11	1. click add plan 2. select an existing plan A 3. select a valid date 4. select a time 5. click submit	Successfully add the item to plan. Toast message "item successfully add to this plan". User can check the plan item at 'Me —>plans —> specific plan' page.	pass
12	1. click add plan 2. click add to new plan 3. key in plan name 4. select a date for the plan 5. click submit 6. select the new plan 7. select a date for the item 8. select a time for the item 9. click submit	Made a new plan and successfully add the item to the new plan. Toast message "item successfully add to this plan". User can check the plan item at 'Me —>plans —> specific plan' page.	pass
13	1. click add plan 2. select an existing plan A 3. select a valid date 4. click cancel	Request is not completed. Go to confirm plan item page.	pass
14	1. click add plan 2. select an existing plan A 3. select a valid date 4. select a time 5. click cancel	Request is not completed. Go to confirm plan item page.	pass

Test case 4 Customer edit profile

Precondition: user is signed in as customer and is at edit profile page

Test ID	Description	Expected Result	Actual result
15	1. click edit photo 2. open a photo album to select a photo	The user photo is change	pass
16	1. key in new first name 2. key in new last name 3. key in new region	First name, last name or region changed in user profile	pass
17	1. select female	Gender is changed in user profile	pass

18	1. click 'select birthday' 2. select a date in the past	Birthday information is changed in user profile	pass
19	1. click 'select birthday' 2. select a date in the future	Request is denied. Invalid birthday	pass
20	1. switch 'vegetarian option' or 'halal option'	Vegetarian option or halal option is changed in user profile	pass

Test case 5 Vendor add a shop

Precondition: user is signed in as vendor and is in add shop page

Test ID	**Description**	**Expected Result**	**Actual result**
21	1. key in complete shop information: name, contact number, address, email and category 2. click upload certificate 3. select a photo from photo album to upload 4. click submit	The shop is added in vendor profile page	pass
21	1. key in incomplete shop information (some attributes not filled in) 2. click upload certificate 3. select a photo from photo album to upload 4. click submit	Shop information is incomplete. Request is denied.	pass
23	3. key in complete shop information: name, contact number, address, email and category 4. click submit	Shop certificate is not uploaded. Request is denied.	pass

White box testing

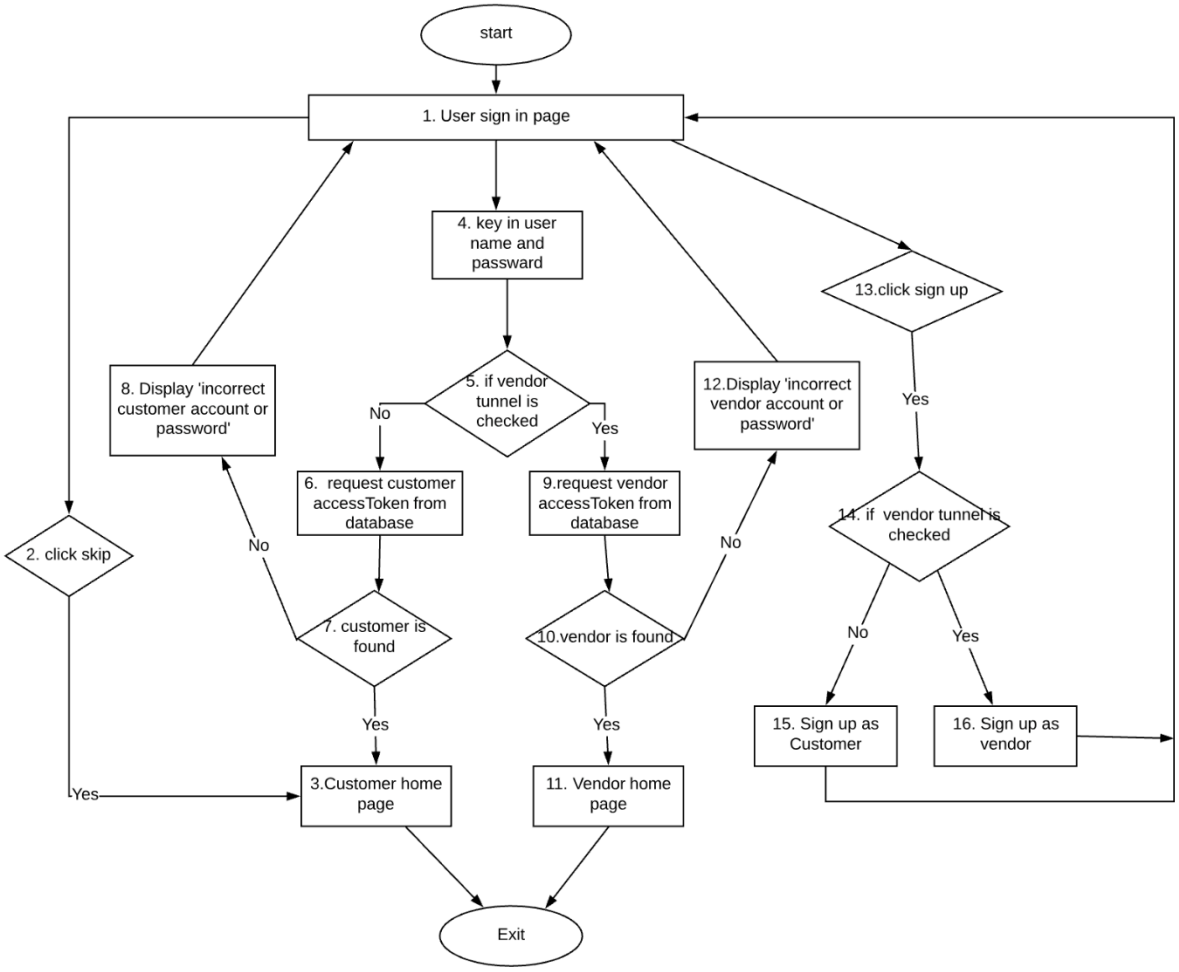
Test Case 1. User login

1. Pseudocode

```
if (click 'skip')
   redirect to Customer home page;
if (click 'sign up'){
   if( vendor tunnel is checked )
```

```
        redirect to vendor sign up page;
    else
        redirect to Customer sign up page;

    user finish fill in sign up detail;
    redirect to sign in page;
}

if (key in user name and password) {
    if( vendor tunnel is not checked ){
        request customer accessToken from database;
        if ( customer is not found ){
            show message "Incorrect customer account or password";
        } else if ( customer is found ){
            redirect to Customer home page;
        }
    }

    // vendor tunnel is checked
    else {
        request vendor accessToken from database;
        if ( vendor is not found ){
            show message "Incorrect vendor account or password";
        } else if ( vendor is found ){
            redirect to vendor home page;
        }
    }
}
```

2. Control Flow Graph



3. Basic Paths

   Path 1: 1—2—3
   Path 2: 1—4 –5—6—7—3
   Path 3: 1—4 – 5 —6—7—8—1(—4—5—6—7—3)
   Path 4: 1—4—5—9—10—11
   Path 5: 1— 4—5—9 —10—12—1
   Path 6: 1—13—14—15—1
   Path 7: 1—13—14—16—1

4. Test Cases for Basic Paths

Path 1

User click skip

Execute result:

Redirect to customer home page.

Path 2

User inputs correct customer name and password, the vendor tunnel is not checked. Customer assess token is then requested from database. The customer is valid.

Execute result:

Redirect to customer home page.

Path 3

User inputs incorrect customer name and password, the vendor tunnel is not checked. Customer assess token is requested from database. The customer is invalid.

Execute result:

Access is denied. User stays at sign in page.

Path 4

User inputs correct vendor name and password, the vendor tunnel is checked. Vendor assess token is then requested from database. The vendor is valid.

Execute result:

Redirect to vendor home page.

Path 5

User inputs incorrect vendor name and password, the vendor tunnel is checked. Vendor assess token is then requested from database. The vendor is invalid.

Execute result:

Access is denied. User stays at sign in page.

Path 6

User clicks sign up with vendor tunnel unchecked. User is redirected to sign up page to fill in sign up information. Upon successful sign up, the data is stored in database.

Execute result:

Redirect to sign in page.

Path 7

User checks vendor tunnel and clicks sign up. User is redirected to sign up page to fill in sign up information. Upon successful sign up, the data is stored in database.

Execute result:

Redirect to sign in page.

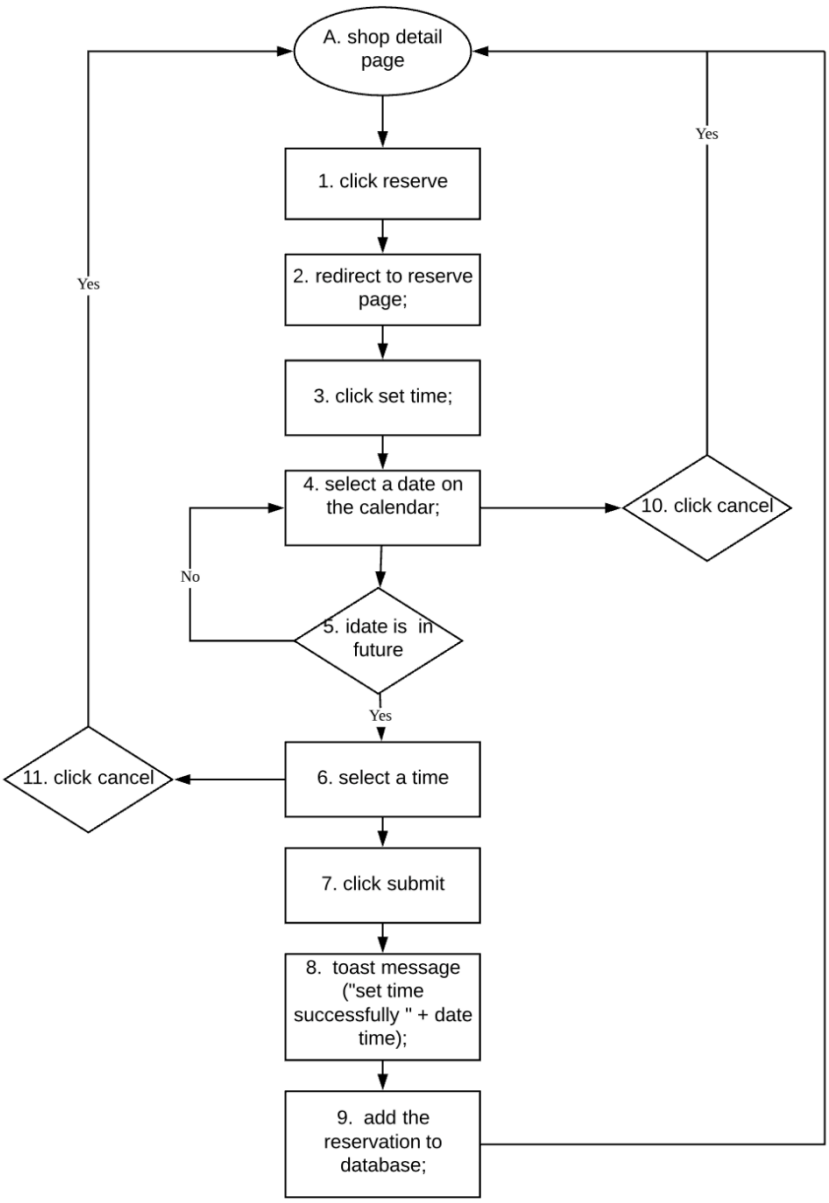
Test case 2. Make a reservation

1. Pseudocode

   (precondition: user is at shop detail page;)

```
click reserve;
redirect to shop detail page;
click set time;
select a date on the calendar;
if (click cancel) {
redirect to reserve page;
}

if (date is in the past) {
   request is denied;
} else {
   select a time;
   if (click cancel) {
     redirect to reserve page;
   }
   click submit;
   toast message ("set time successfully " + date time);
}
add the reservation to database;
redirect to reserve page;
```

2. Control Flow Graph



3. Basic Paths

   (precondition: user is at shop detail page A)

   Path 1: A—1—2—3—4—5—6—7—8—9—A

   Path 2: A—1—2—3—4—10—A

   Path 3: A—1—2—3—4—5—6—11—A

4. Test Cases for Basic Paths

Path 1

User clicks reserve button at shop detail page, and he is redirected to reserve page. User clicks set time button and a date picker dialog pops out. User selects a valid date on the calendar. Subsequently selects a time and clicks submit.

Execute result:

The system toasts a message “set time successfully + [selected date time] “. The new reservation is added to database. New reservation will be displayed at ‘me —> reservations’ page.

Path 2

User clicks reserve button at shop detail page, and he is redirected to reserve page. User clicks set time button and a date picker dialog pops out. User clicks cancel to quit the request.

Execute result:

Redirect to shop detail page.

Path 3

User clicks reserve button at shop detail page, and he is redirected to reserve page. User clicks set time button and a date picker dialog pops out. User selects a valid date on the calendar. Subsequently selects a time and clicks cancel to quit the request.

Execute result:

Redirect to shop detail page.

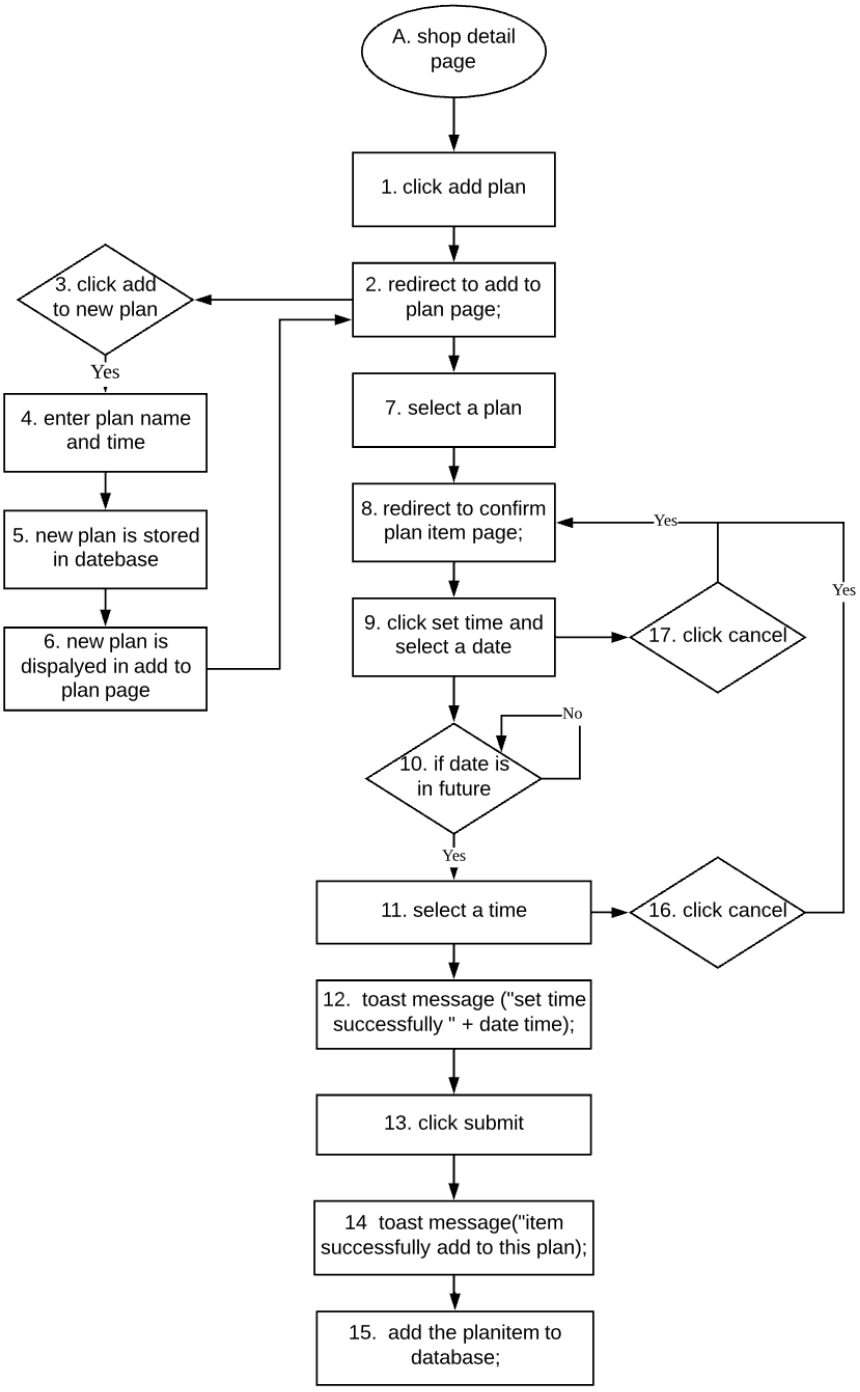
Test case 3 Add a shop to a plan

1. Pseudocode

```
precondition: in shop detail page;

click add plan;
redirect to addPlan page;

if (click add to new plan) {
   enter plan name and time;
   new plan is stored in database;
   new plan is displayed in add to plan page;
}

select a plan to add;
redirect to confirm plan item page;
```

```
click set time;
select a date on the calendar;

if (click cancel) {
   redirect to confirm plan item page;
}

if (date is in the past) {
   request denied;
} else {
   select a time;
   if (click cancel) {
      redirect to confirm plan item page;
   }
}

toast message ("set time successfully" + date time);
click submit;
toast message ("item successfully add to this plan");
store data in database;
```

2. Control Flow Graph



3. Basic Paths

(precondition: user is at edit profile page)

Path 1: A—1—2—7—8—9—10—11—12—13—14—15

Path 2: A—1—2—3—4—5—6—2

Path 3: A—1—2—7—8—9—17—8

Path 4: A—1—2—7—8—9—10—11—16—8

4. Test Cases for Basic Paths

Path 1

User clicks add plan button at shop detail page and is redirected to add to plan page. User selects an existing plan then redirected to confirm plan item page. User selects a valid date and time on the picker dialog and click submit.

Execute result:

The system toasts message “set time successfully + [selected date time] “. After user clicks submit, system toasts message “item successfully add to this plan”. The plan item is added to database accordingly. Plan item will be displayed at ‘me —> plans—> [specific plan]’ page.

Path 2

User clicks add plan button at shop detail page and is redirected to add to plan page. User clicks add to new plan. User inputs plan name and time and is redirected to add to plan page. Then continues on path 1.

Execute result:

New plan is stored in database and displayed in add to plan page.

Path 3

User clicks add plan button at shop detail page and is redirected to add to plan page. User selects an existing plan then redirected to confirm plan item page. User clicks set time and then clicks cancel to quit the request.

Execute result:

Redirect to confirm plan item page.

Path 4

User clicks add plan button at shop detail page and is redirected to add to plan page. User selects an existing plan then redirected to confirm plan item page. User selects date and time. Then clicks cancel to quit the request.

Execute result:

Redirect to confirm plan item page.

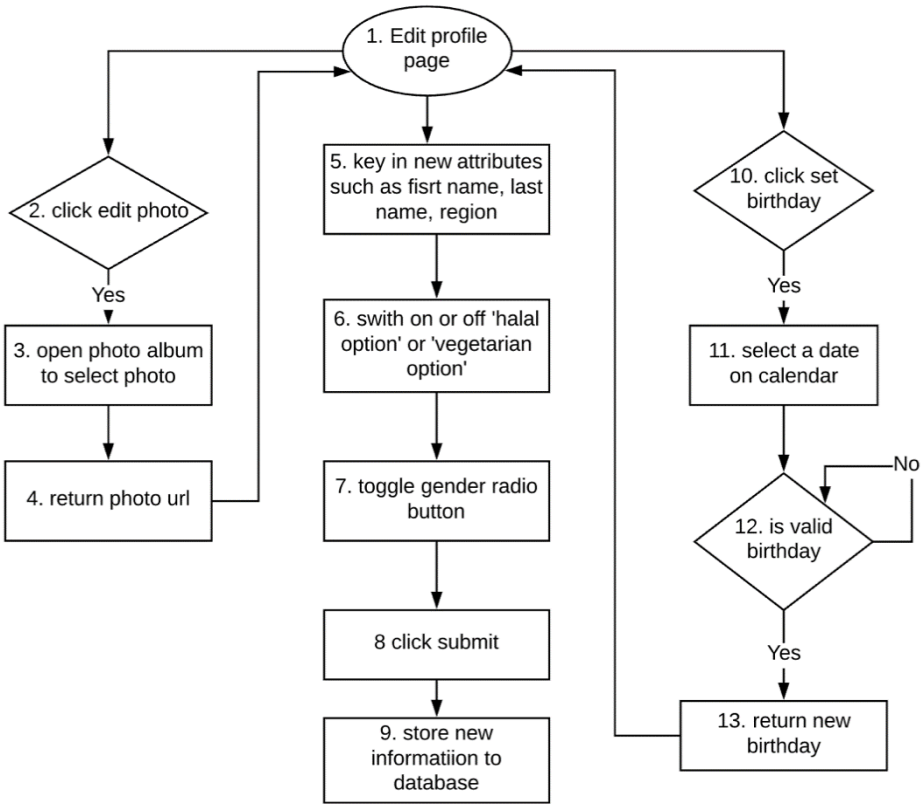
Test Case 4. Edit customer profile

Precondition: user is signed in as customer and is at edit profile page

1. Pseudocode

```
key in new attributes;
change halal and vegetarian switch to edit preference;
if (click edit photo) {
   go to photo album and select a photo;
   get URL of the new photo;
}

if (click select birthday) {
   date picker dialog pops out;
   if (date is in the future){
      request denied;
   } else {
      change users birthday;
   }
}

click submit to store new information in database;
```

2. Control Flow Graph



3. Basic Paths

(precondition: user is at edit profile page)

Path 1: 1—5—6—7—8—9

Path 2: 1—2—3—4—1—5—6—7—8—9

Path 3: 1—10—11—12—13—1—5—6—7—8—9

4. Test Cases for Basic Paths

Path 1

User keys in the attributes he wants to edit such as first name, last name, region. He can switch on or off halal option and vegetarian option to change preference settings. Furthermore, he or she can toggle gender radio button to change gender setting. After clicking ‘submit’ button, the new information will be stored to database

Execute result:

New information is stored to database and displayed at user profile page.

Path 2

User clicks edit photo button. User is redirected to photo album to select a new photo. The photo URL is returned. And user continues on path 1.

Execute result:

User photo is changed. New information is stored to database and displayed at user profile page.

Path 3

User clicks set birthday button. User selects a date on calendar. Only valid birthday can be chosen and returned. User continues on path 1.

Execute result:

User birthday is changed. New information is stored to database and displayed at user profile page.

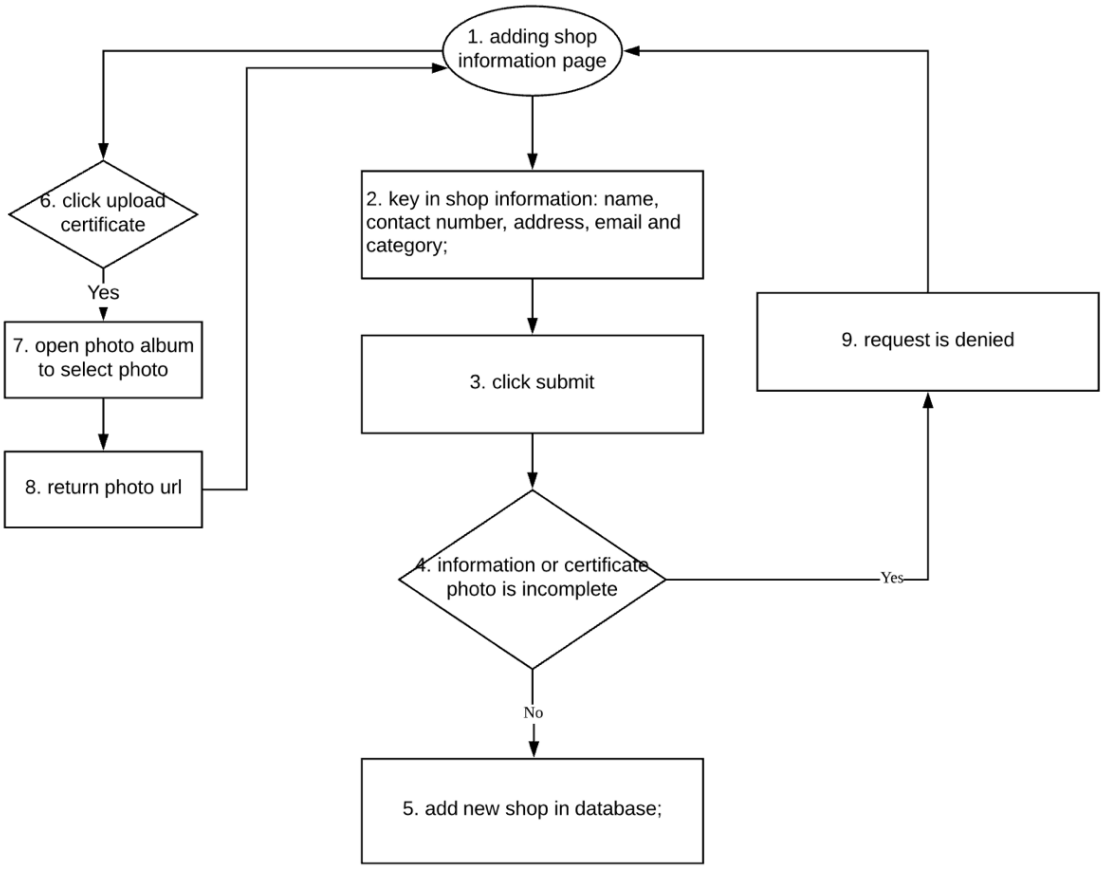
Test case 5. Vendor add a shop

Precondition: user is signed in as vendor and is at add shop page

1. Pseudocode

```
key in shop information: name, contact number, address, email and category;

if (click upload certificate) {
   go to photo album and select a photo;
   get URL of the new photo;
}

click submit;

if (information or certificate photo is incomplete) {
   request is denied;
}

add new shop in database;
```

2. Control Flow Graph



3. Basic Paths

   (precondition: user is at edit profile page)

   Path 1: 1—6—7—8—1—2—3—4—5

   Path 2: 1—6—7—8—1—2—3—4—9—1

   Path 3: 1—2—3—4—9—1

4. Test Cases for Basic Paths

Path 1

User keys in complete shop information: name, contact number, address, email and category. User clicks upload certificate and selects a photo from photo album. Photo URL is returned to the system. User clicks submit. The system examines the information and found it complete.

Execute result:

The new shop is added in database and displayed at vendor profile page.

Path 2

User keys in incomplete shop information. Some attributes of name, contact number, address, email and category are not filled in. User clicks upload certificate and selects a photo from photo album. Photo URL is returned to the system. User clicks submit. The system examines the information and found it incomplete.

Execute result:

Shop information is incomplete. The request is denied.

Path 3

User keys in shop information: name, contact number, address, email and category. User clicks submit. The system examines the information and found no certificate photo is uploaded.

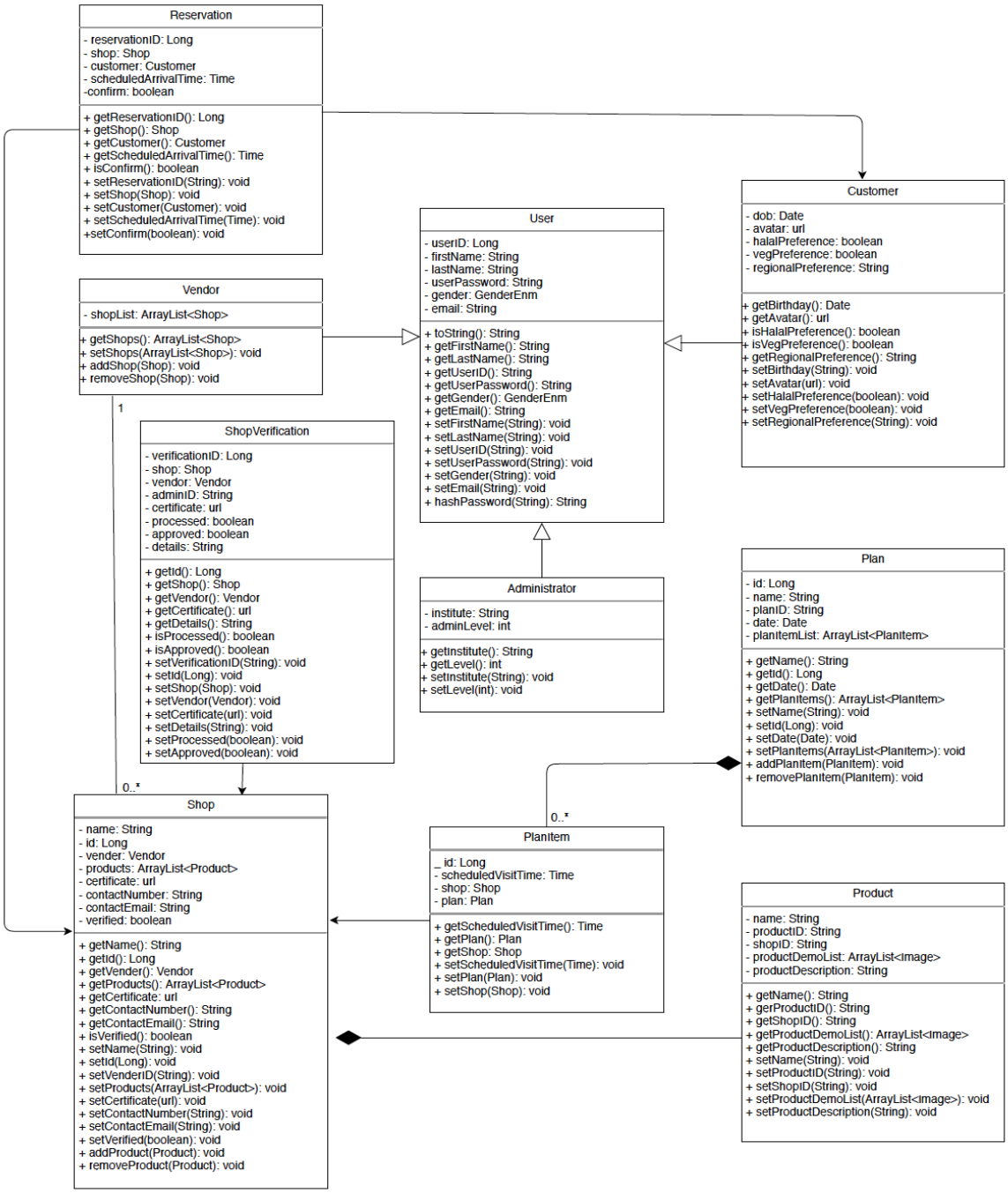
Execute result:

Shop information is incomplete. The request is denied.

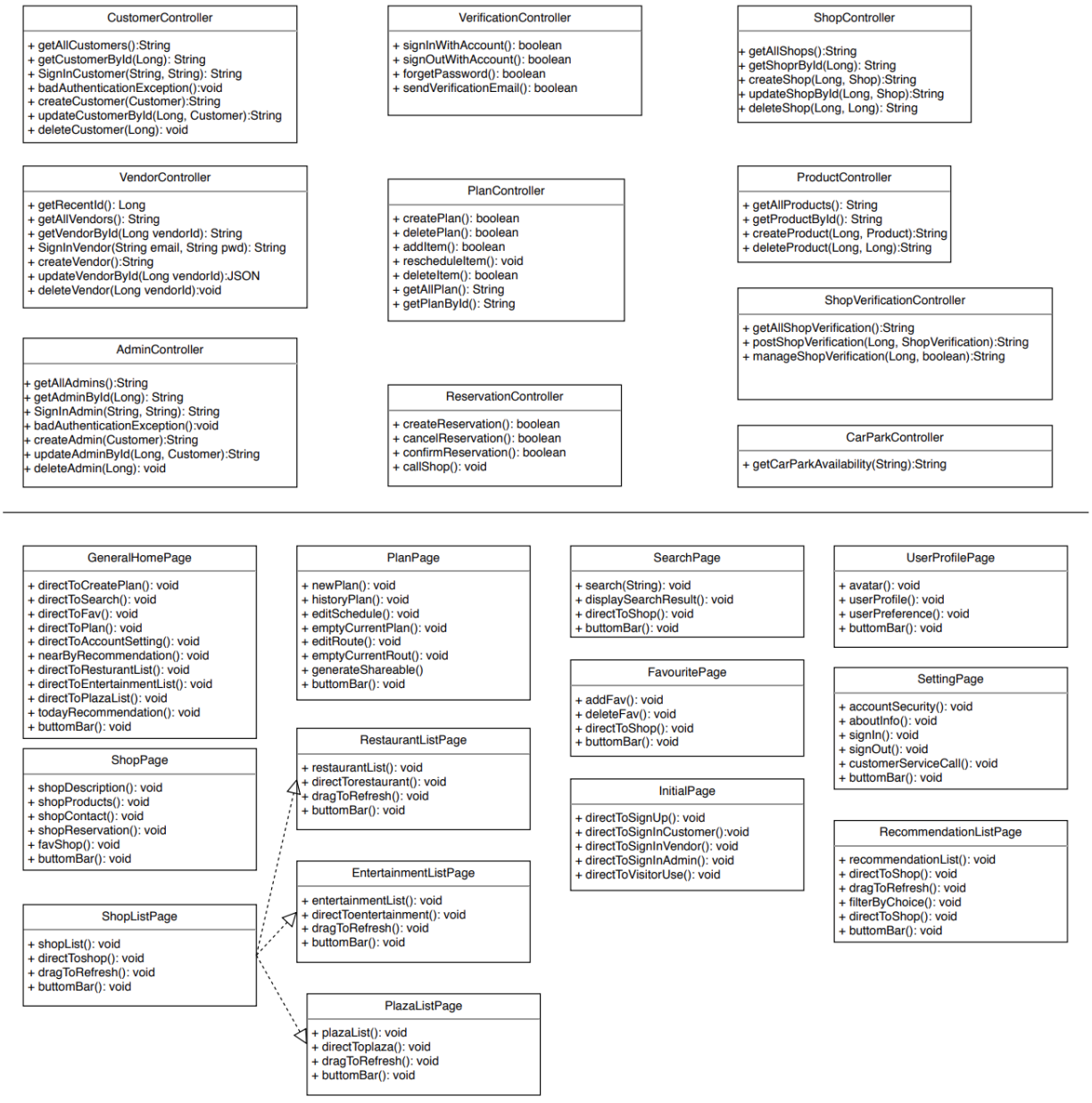
7. Design Models

7.1 Conceptual Models – Class Diagrams

7.1.1 Class Diagram for entity classes



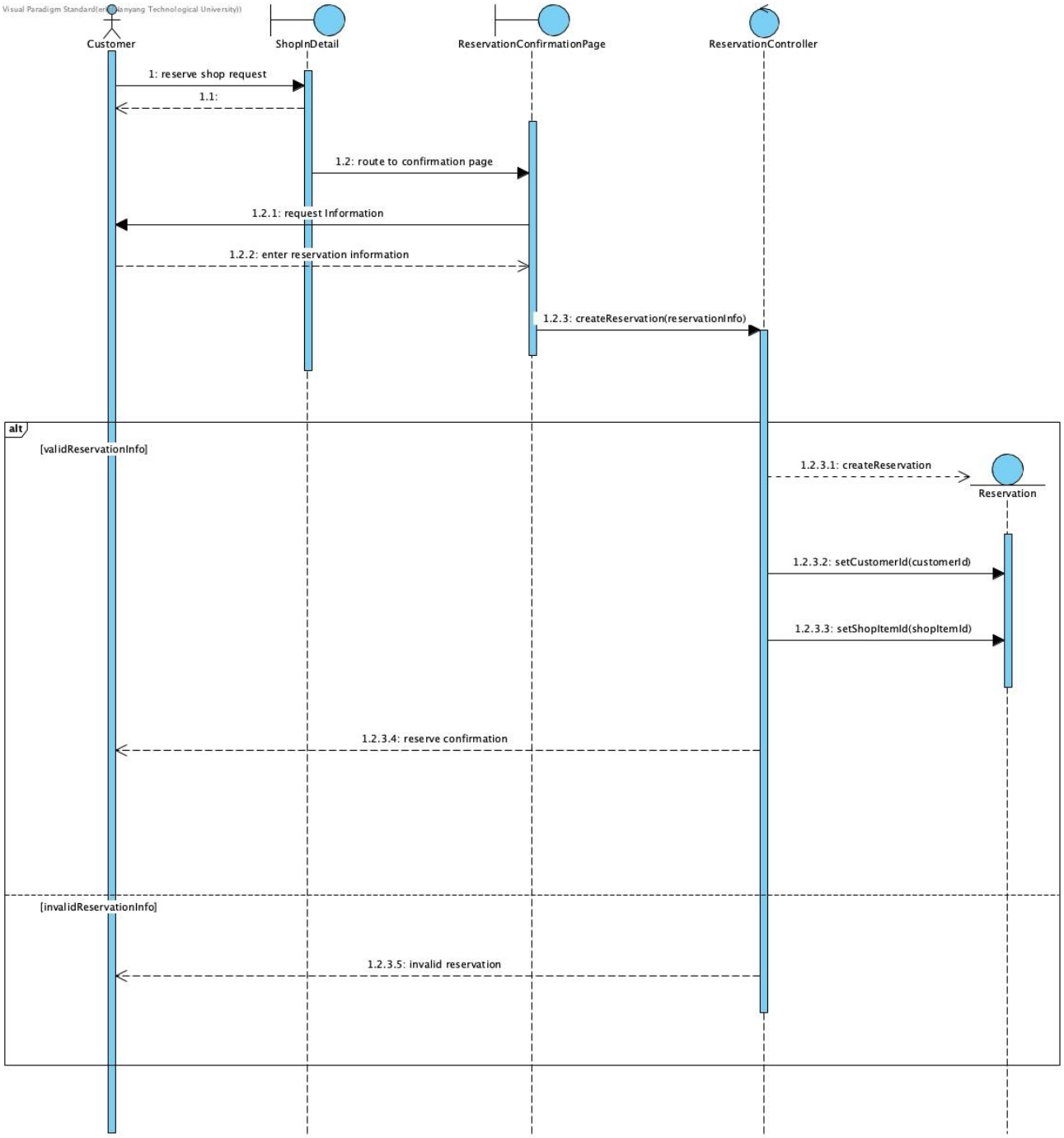
7.1.2 Class diagram for boundary and control classes



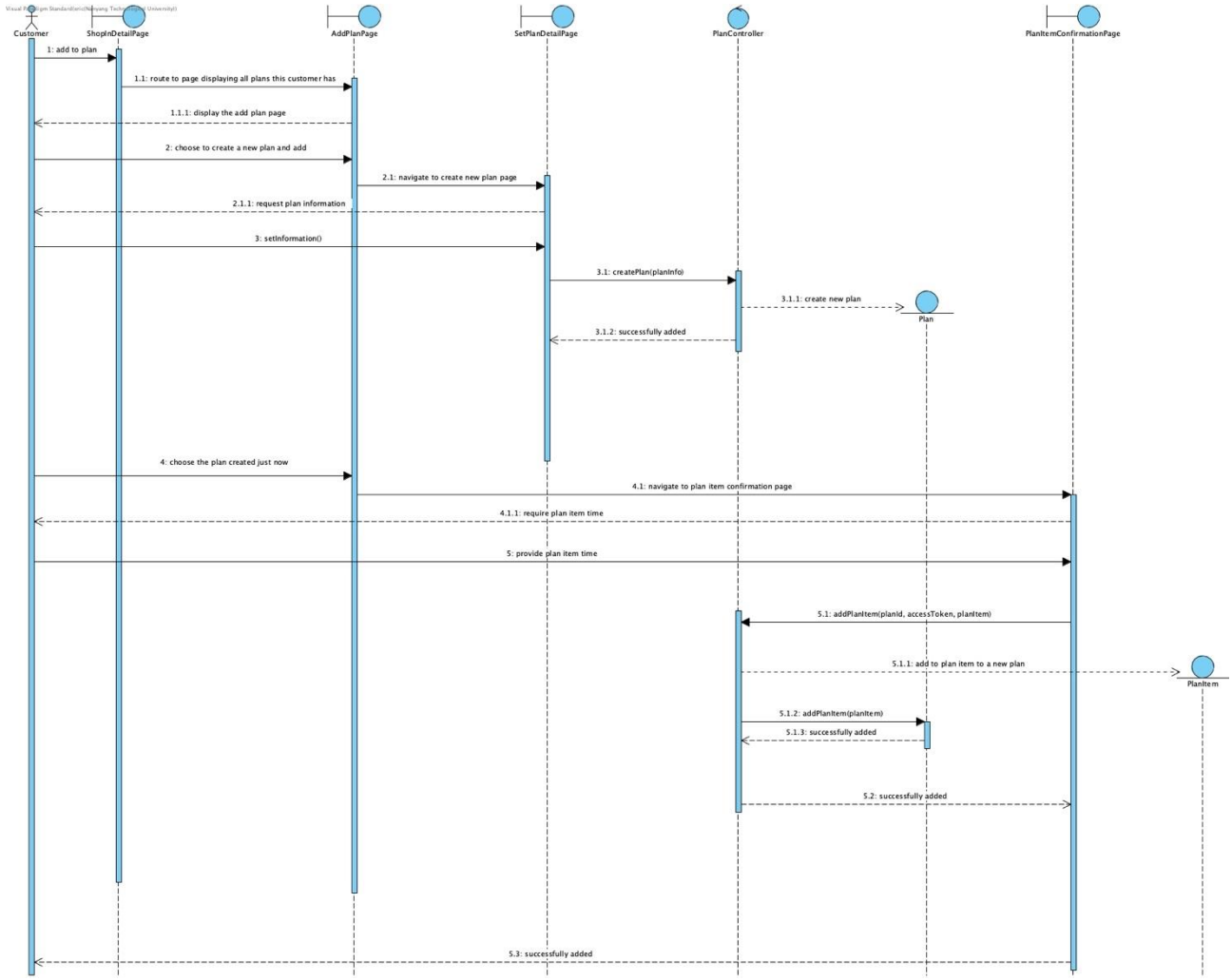
7.2 Dynamic Model – Sequence Diagrams

Here are sequence diagrams for some key use cases

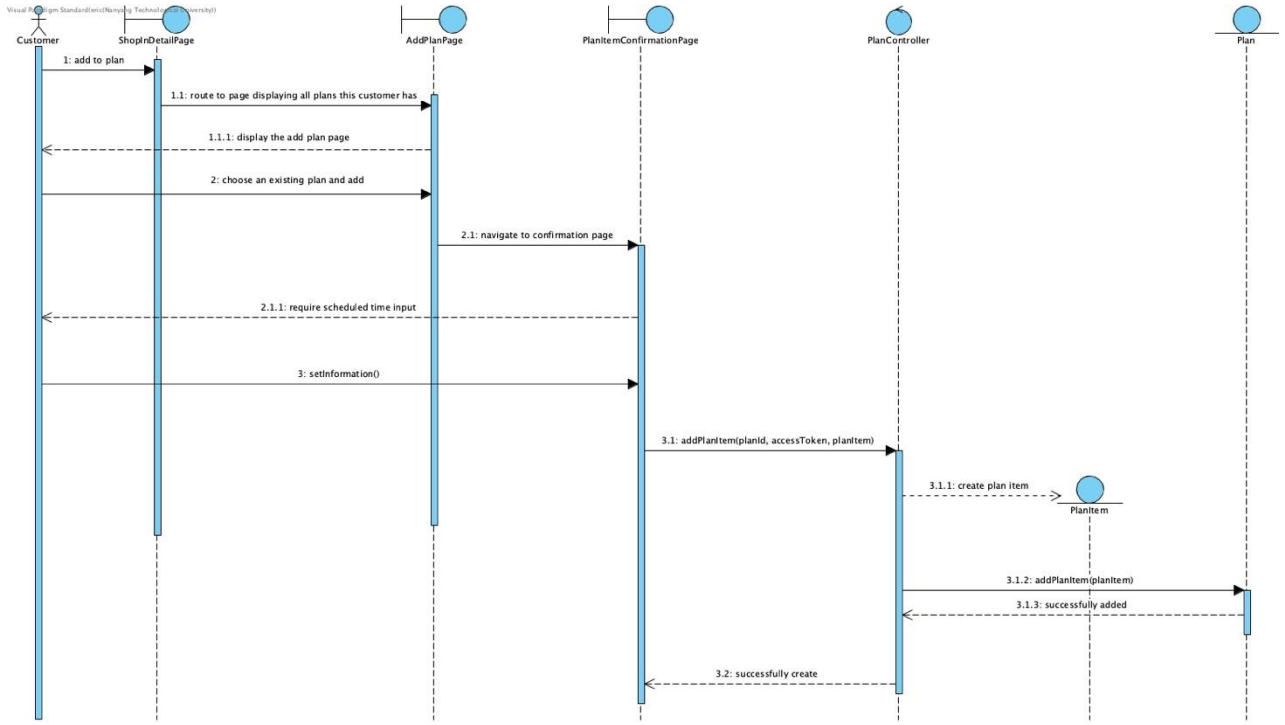
7.2.1 Customer Reserve Shop



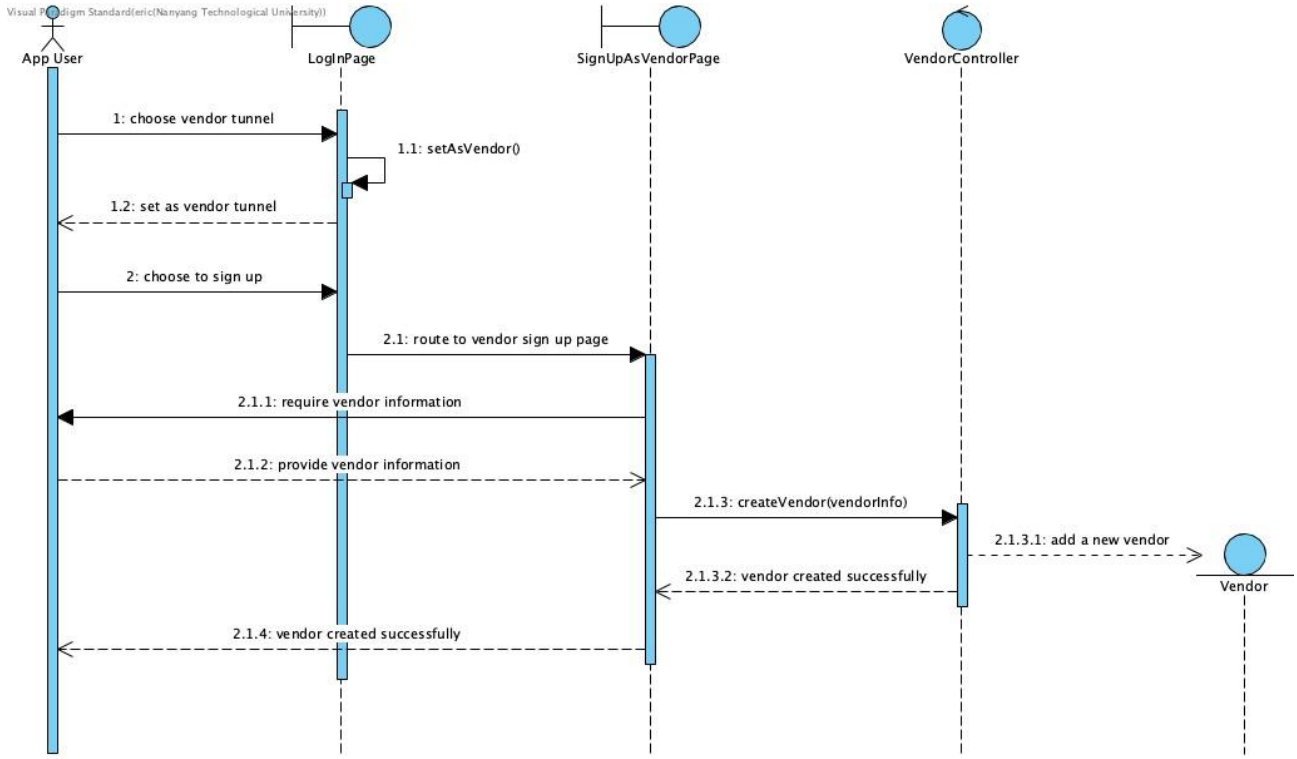
7.2.2 Customer Add Plan Item to a new Plan



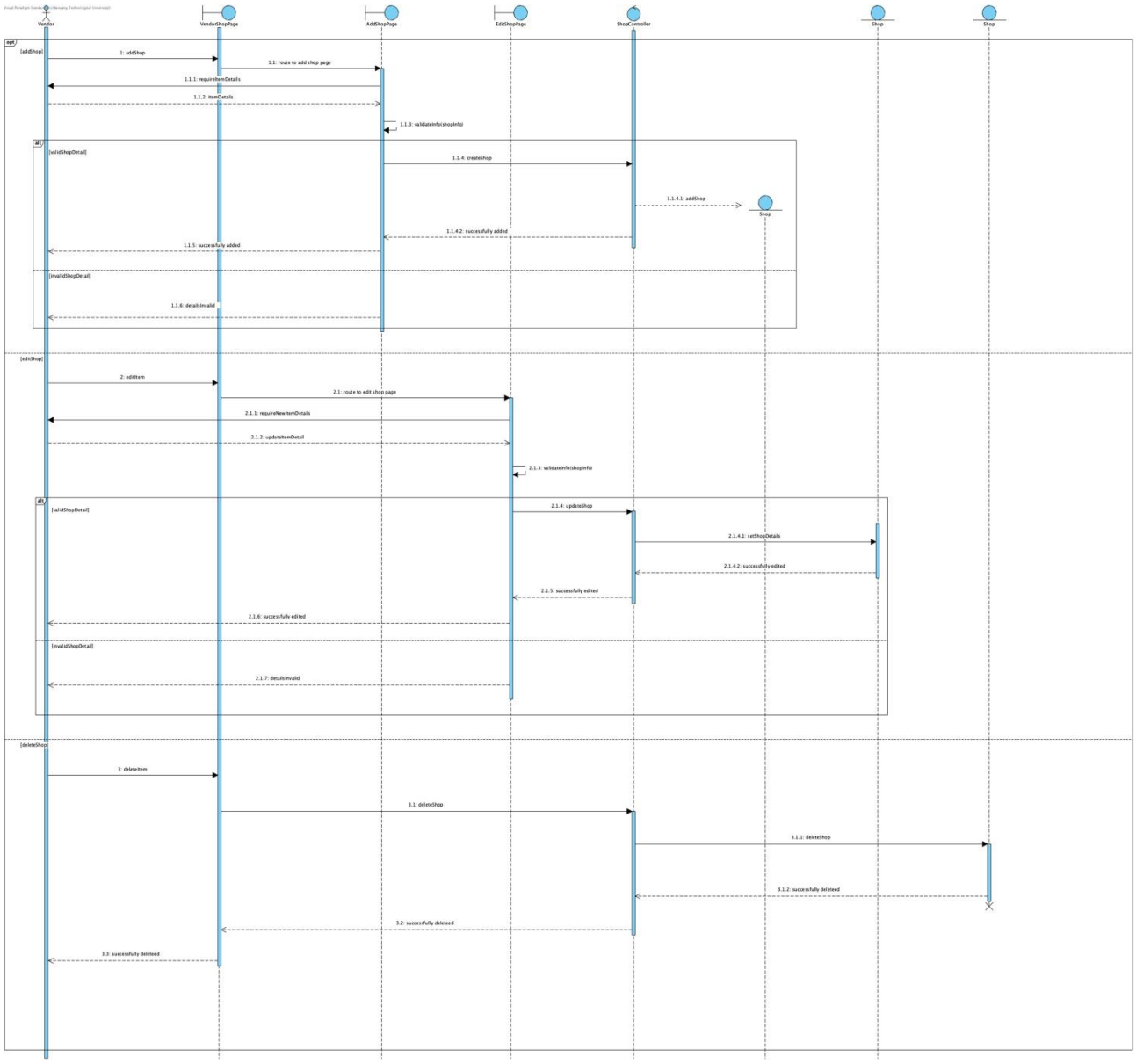
7.2.3 Customer Add Plan Item to an Existing Plan



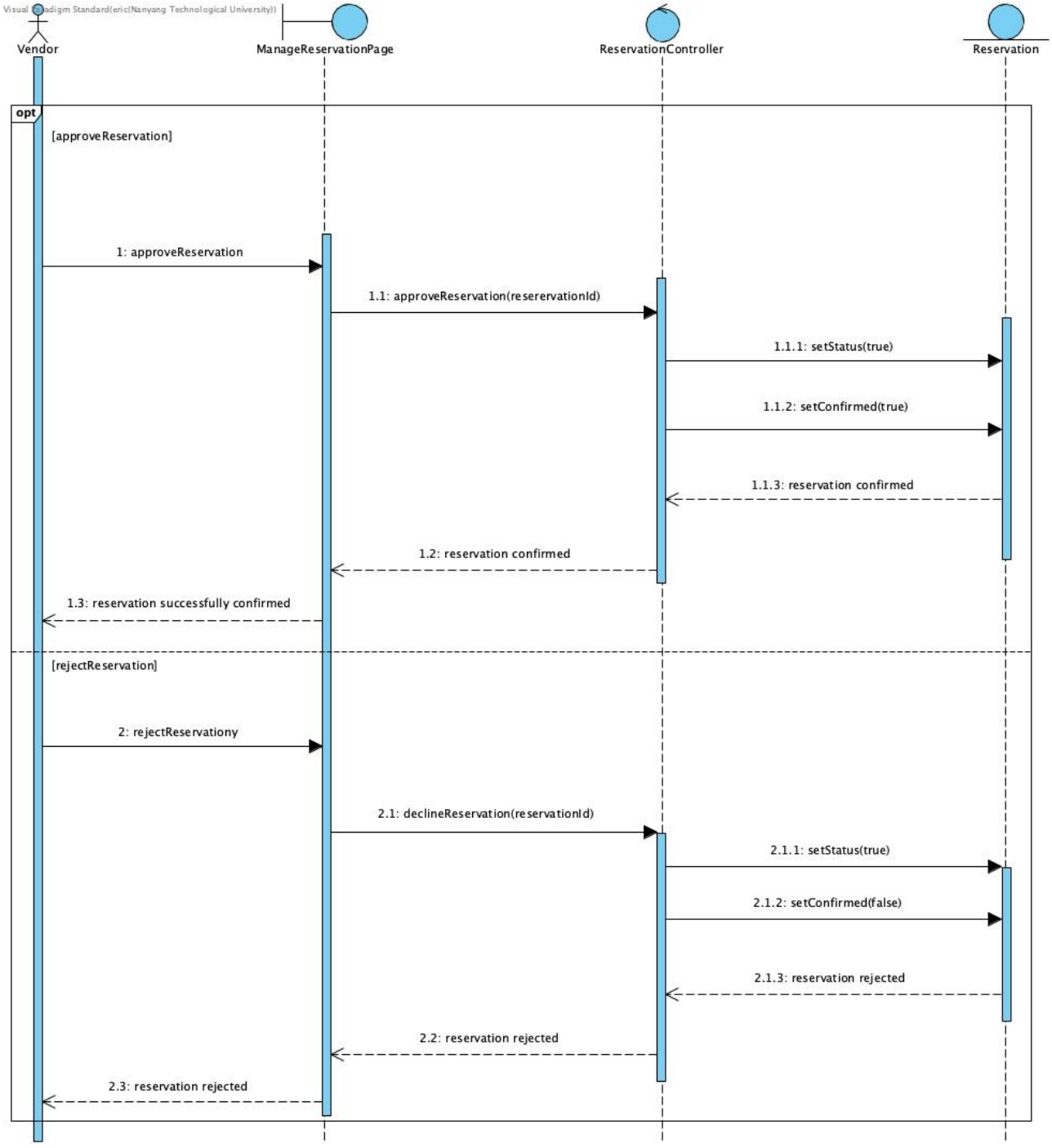
7.2.4 Sign Up Vendor



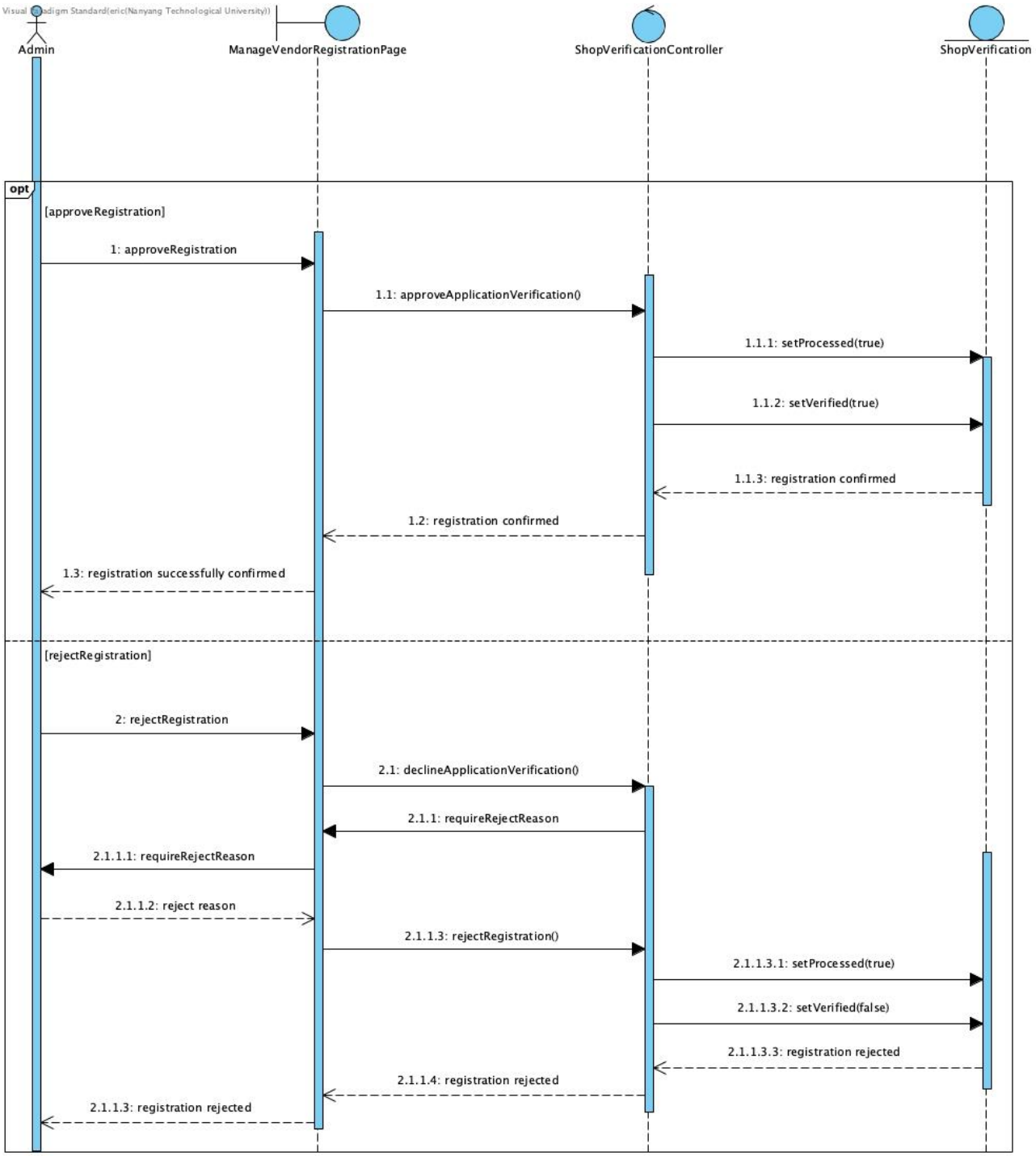
7.2.5 Vendor Manage Shop



7.2.6 Vendor Process Reservations (Future Improvement)

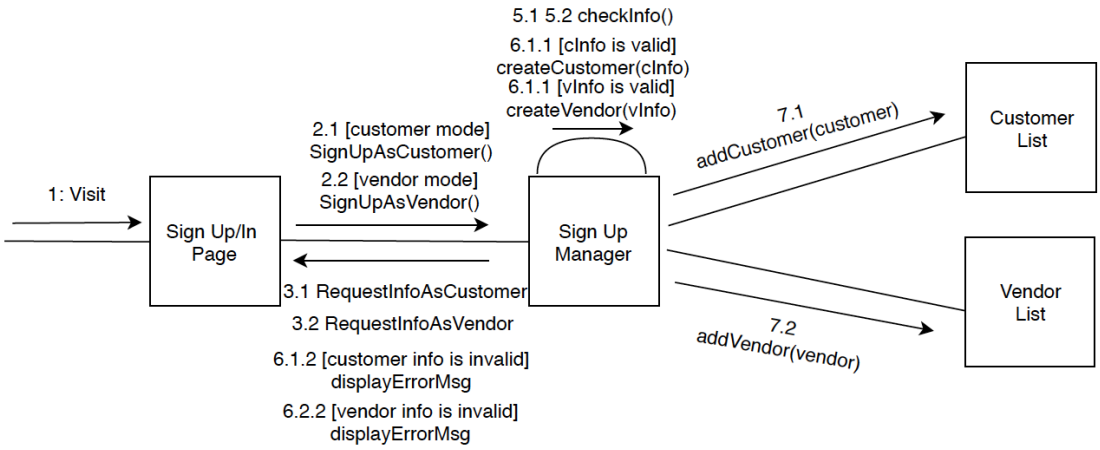


7.2.7 Admin Process Shop Verification (Future Improvement)



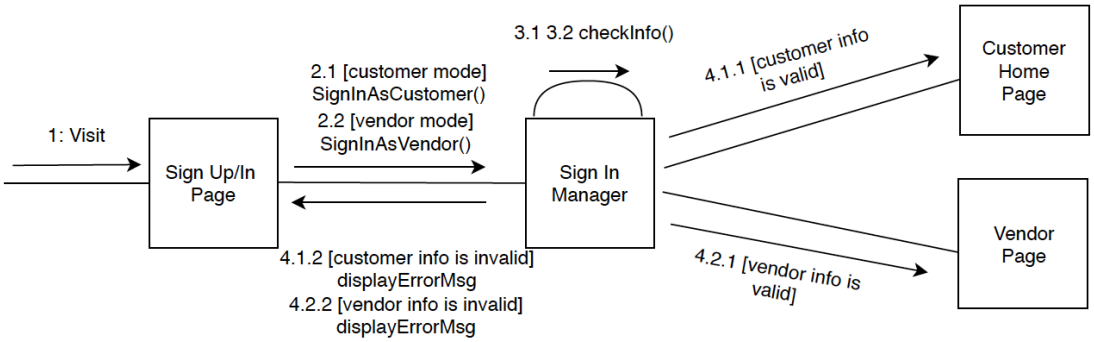
7.3 Dynamic Models – Communication Diagrams

7.3.1 User Sign Up



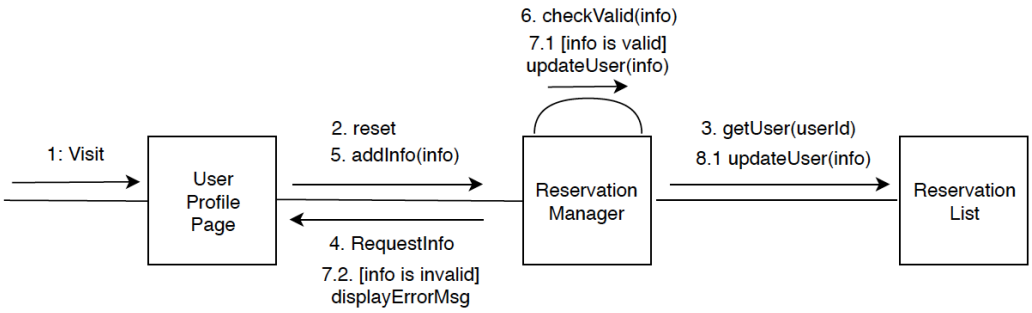
Users sign up

7.3.2 User Sign In



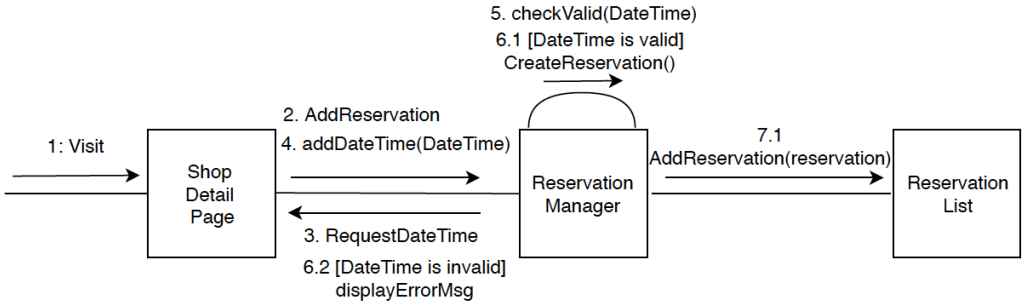
Users sign in

7.3.3 User Edit Profile



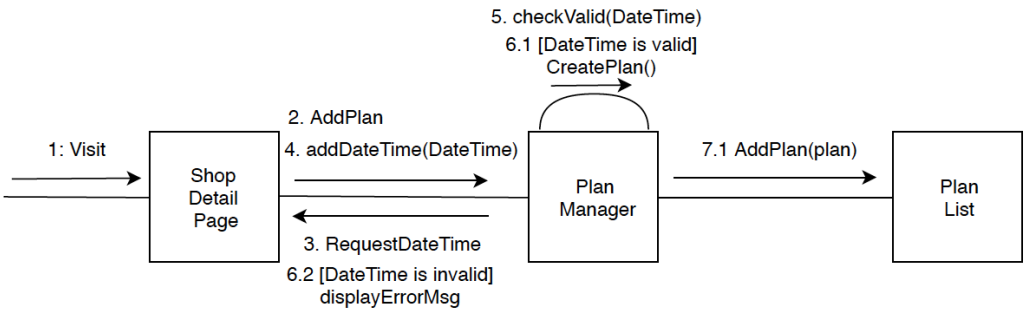
Users edit profile

7.3.4 Customer Make Reservation



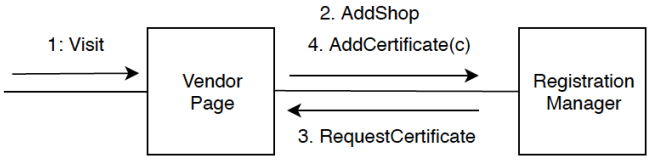
Customers make a reservation

7.3.5 Customer Add Plan Item



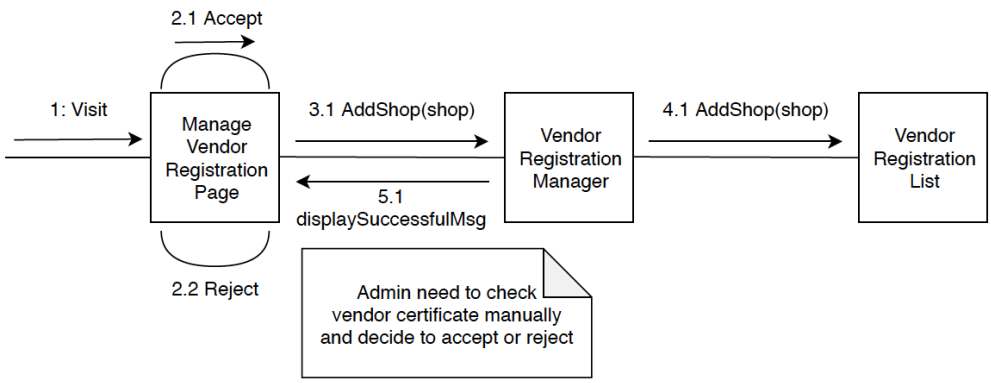
Customers add a shop to a plan

7.3.6 Vendor Add a Shop



vendor declare shop

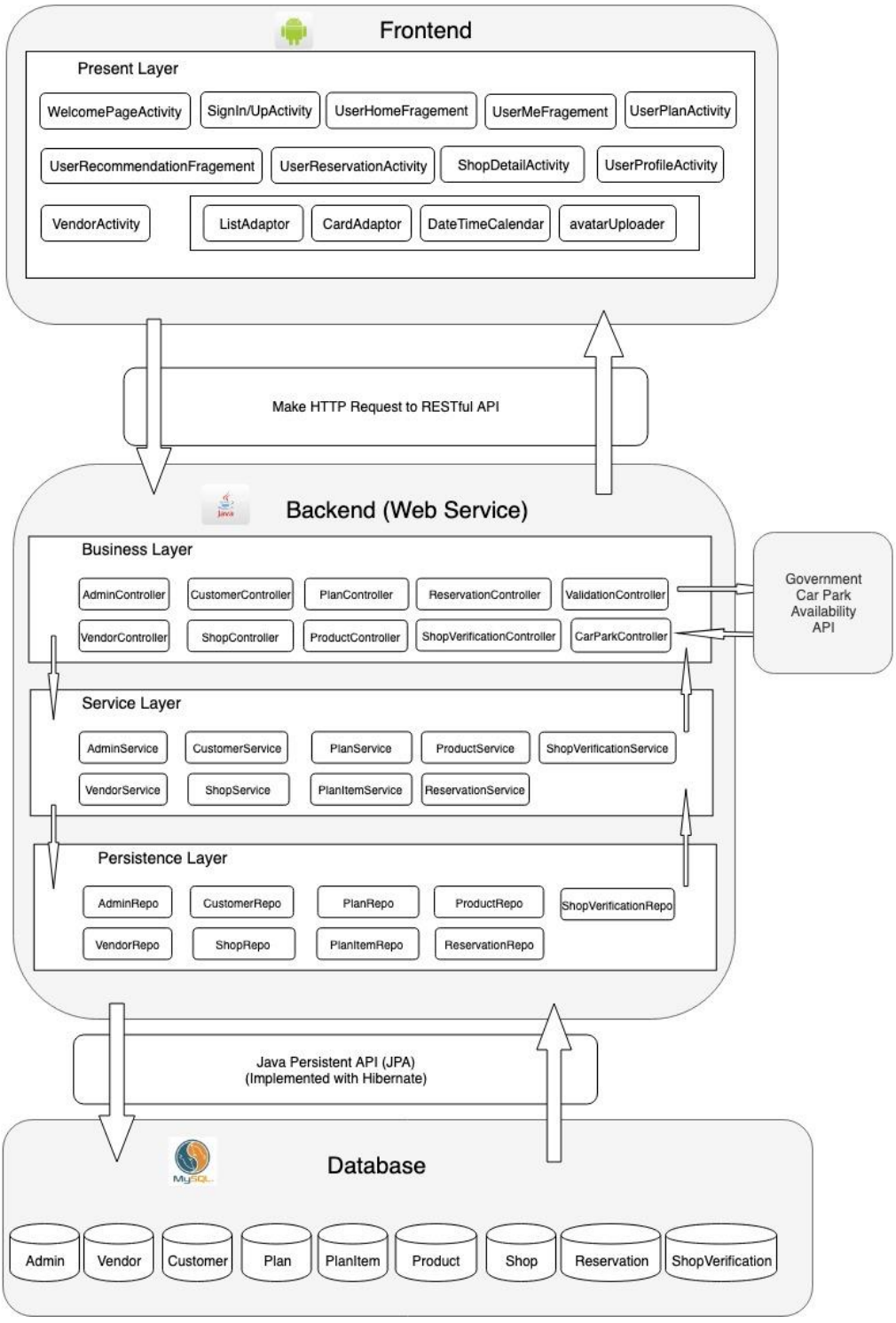
7.3.7 Admin Process Shop Verification



Admin process vendor registration request

8. Design Concerns

8.1 System Architecture Diagram



We built design patterns to achieve the high performance of class composition. Inheritance, extension and overrides are used to compose interfaces by different patterns. In our application, patterns define how to compose objects and obtain the functionalities.

8.2 Structural design patterns

8.2.1 Adapter

In our application, adapters are widely used to match interfaces of different classes to build up our target features. We use adapters to convert the interface of a class into another interface of clients. Adapters in our application also lets classes work together to implement comprehensive features.

8.2.2 Decorator

Our team focuses on the good use of decorators. Decorators dynamically define the actions, responsibilities and roles of objects. They can attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative way to extend existed functionalities.

8.2.3 Private Class Data

The keyword ‘private’ limits the external access and improves the security and reasonability of a class. Private Class encapsulates class data and permits safe initialization.

8.2.4 Bridge

Bridges decouple the abstraction from its implementation. In our application, it publishes interfaces in an inheritance hierarchy and buries implementation correspondingly.

8.3 Behavioural design patterns

8.3.1 Iterator

The iterators provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation. The Java standard library abstraction makes it possible to decouple collection classes and algorithms. We use iterators to perform polymorphic traversals and iterative searches.

8.3.2 Interpreter

Given the embedded Java interpreter in Android Studio, our team applies objective-oriented Java grammars to let the interpreter interpret corresponding sentences. By mapping the variable domain to our code segments, the interpreter permits the program be of the object-oriented design style.

8.3.3 Null Objects

The intent of a Null Object is to encapsulate the absence of an object by providing a substitutable alternative that offers the suitable “Do-Nothing” default. In short, the Null Object pattern makes use of a collaboration that already exists some instances. These instances should do nothing in order to handle null-operations from the client.

8.3.4 State

The State allows our objects to alter behaviours when detecting the change of internal values or memory. Under the situation, objects will appear to change its class states. In our application, the State can be referred as an object-oriented state machine wrapper. This wrapper contains a polymorphic wrappee and relative Contexts.

8.3.5 Strategy

The Strategy defines a series of algorithms by encapsulating each one and making them interchangeable. Strategy lets the algorithm vary independently from the clients or events that call for algorithms. The Strategy in our application is applied by different client tabs. It captures the abstractions and bury implementation details in corresponding classes.

8.3.6 Visitor

The Visitor represent an operation to be performed differed by the roles of object. Visitors lets our application define a new operation without changing the classes of the UI on which it operates. Visitor builds the classification between different clients (e.g. Vendors, Customers or Admins). It is one of most important techniques that we used to separate user inputs by types. In short, The Visitor allow us to build features independently.

8.3.7 Template Method

The Template Method defines the skeleton of an algorithm in the operation. It refers to the general steps of an algorithm. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. In short, our application takes base classes as 'placeholders', and sub-classes as the implementation the ‘placeholders’.

8.4 Creational design patterns

8.4.1 Abstract Factory

In our implementation, the Abstract Factory provides an interface for creating families of related or dependent objects without specifying their concrete classes. In short, it is like the collection of all abstract relations that link to different sub-classes.

8.4.2 Builder

The Builder separate the construction of a complex object from its method representations. Thus, in our application, the same construction method can create different instance representations by putting different initialization values. The Builder helps us to create instances efficiently.

8.4.3 Prototype

The Prototype specifies the kinds of objects and create new objects by different constructor implementations. In our application, prototypes declare abstract base classes and maintain a dictionary of all "cloneable" concrete derived classes. Any class that requires constructors will extract and evolve them from the abstract base class.

8.4.4 Singleton

The Singleton ensures that some certain class has only one instance and provides a global access method to this instance. That is, the Singleton in our application permits the unique initialization on first use. To be specific, the user access token, user id and user sign-in status are uniquely initialized and altered. They should only have one global instance for all related classes and methods. By declaring the instance as private or static data member, the Singleton principle provides a general way to encapsulate important global initializations and corresponding accessors to the instance.

9. Software Design Principles

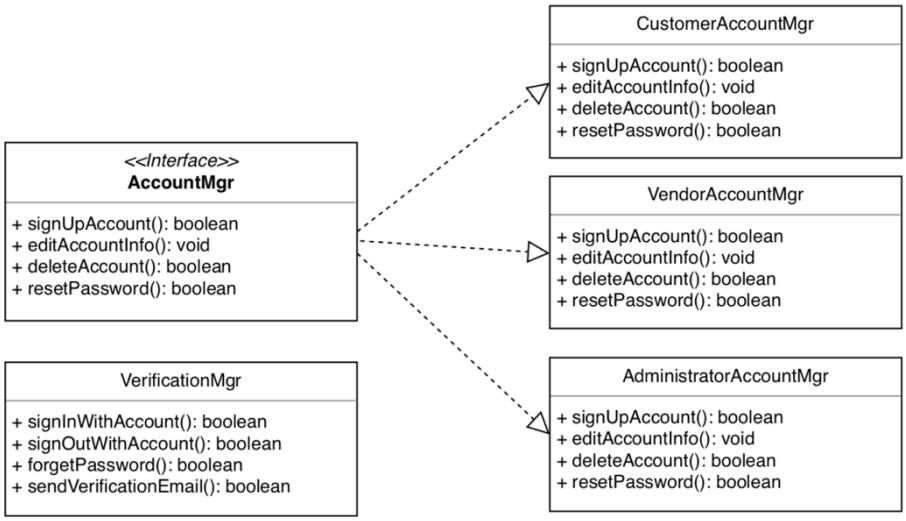
SOLID is one of the most popular sets of design principles in object-oriented software development. We applied these five principles in the practice of front-end and back-end software engineering. SOLID is the mnemonic acronym for the following:

9.1 Single Responsibility Principle

The principle addresses that a class should have one, and only one, reason to change. In our application, we create different managers to manage features respectively.

9.2 Open/Closed Principle

The principle addresses that the application should be open for extension and closed for modification. For the practice, our team takes care on the specific Interface and implements it in multiple styles to satisfy the needs of different types of user.



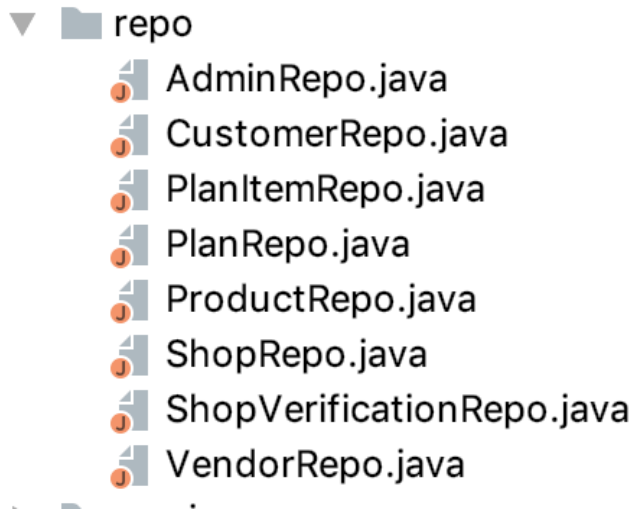
9.3 Liskov Substitution Principle

The principle addresses that parent classes should be easily substituted with their child classes without blowing up the application. As we create the user class as the parent class, it can be substitute by child classes (e.g. Vendor, Customer or Admin) without influencing the application performance negatively.

```java
@Entity
@Table(name = "Vendor")
public class Vendor extends User {

    @OneToMany(mappedBy = "vendor", cascade = CascadeType.ALL, orphanRemoval = true)
    @JsonManagedReference
    private List<Shop> shops = new ArrayList<>();


    public Vendor(Long id, String firstName, String lastName, GenderEnm gender, String
        super(id, firstName, lastName, gender, email, password);
        this.shops = shops;
    }

    public Vendor() {
    }

    @Override
    public String toString() {
        return "Vendor{" +
                "id=" + super.getId() +
                ", firstName='" + super.getFirstName() + '\'' +
                ", lastName='" + super.getLastName() + '\'' +
```

9.4 Interface Segregation Principle

The principle addresses that many client-specific interfaces are better than one general-purpose interface. We follow the principle to create many specific interfaces for different features.



9.5 Dependency Inversion

The principle addresses that high-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions. In our application, High-level modules which provide complex logic are reusable and unaffected by changes in low-level modules which provide utility features. This follows the principle and refines our objective-oriented structures.

```java
public interface PlanItemRepo extends JpaRepository<PlanItem,Integer>{
    // find a planItem by id
    PlanItem findById(Long id);

    // insert one PlanItem
    PlanItem save(PlanItem planItem);

    // delete one user
    void delete(PlanItem planItem);

}
```

```java
@Service
public class PlanItemServiceImpl implements PlanItemService{

    @Autowired
    PlanItemRepo planItemRepo;

    @Override
    public List<PlanItem> findAll() { return planItemRepo.findAll(); }

    @Override
    public PlanItem findPlanItemById(Long id) { return planItemRepo.findById(id); }

    @Override
    public PlanItem save(PlanItem planItem) { return planItemRepo.save(planItem); }

    @Override
    public void delete(PlanItem planItem) { planItemRepo.delete(planItem); }
}
```