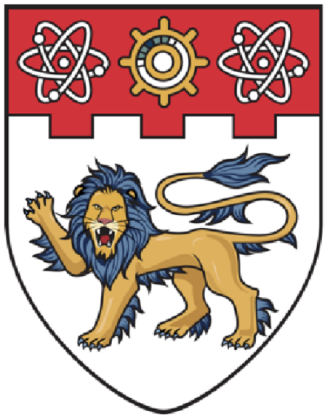


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering

SC2006 Software Engineering

Project Name - EatNow!

DSAI2 / Group 6

Team Anythinglor

Team members:

Ong Jun An (U2121799C) (Leader)

Chua Jing Jie Justin (U2122610G) (Assistant Leader)

Jewel Chin (U2122021C)

Sean Lim (U2121484D)

Contents

1. Product Description

1.1 Purpose

EatNow! aims to provide real time updates on the food deals available nationwide. Just like any other food applications, our application will display a variety of restaurants to allow users to make reservations or purchase orders in any restaurant of the user's choice. But what makes our application special is its review and users' following system that facilitates interaction between users. This is to encourage users, especially food lovers, to fully utilise our application as a platform to discuss and popularise food trends, while providing food services.

1.2 Scope

Our EatNow! Application allows users to view a feed of restaurants' reviews, which are created by other users. Users can view the details of the review by clicking on the review item. The review can be liked, commented, reported if deemed inappropriate by a registered user.

The application also displays the details of a wide range of restaurants in Singapore, as well as its associated reviews. Registered users can book a reservation or order food from their desired restaurant of choice. They can also leave a review for the restaurant.

The application also reveals a list of users, as well as the reviews that they have created. Registered users can follow other users to view a feed of filtered reviews based on these users.

Users can create their own user account to unlock several features of the application, as mentioned above. Users can customise their account by adding a profile picture.

1.3 Users and Stakeholders

The stakeholders of this project consist of food and beverages (f&b) restaurants, android mobile users and EatNow!. Information about food restaurants will be collected through the application with the assistance of Google Places API. Android mobile users will use the application to search for restaurants to reserve or purchase food from their desired restaurants, as well as viewing the available food deals. They will also get to experience the social media aspects of the application by exchanging reviews and following other users, once they register with our application. By attracting users to constantly check-in to our application through its social media, EatNow! strives to bring f&b restaurants closer to users to publicise those which are well-received. F&b restaurants will also utilise our application to create food trends by persuading influencers, and post their food deals to market their products. On the other hand, EatNow! also strives to provide a seamless platform for food services between restaurants and users.

1.4 Assumptions and Constraints

1.4.1 EatNow! Project

The project will utilise Google Firebase Realtime Database as a cloud platform for our application. Thus, the database is limited to JSON format. For example, images should be stored in ASCII bit strings. On the other hand, the structure of the database must be crafted to uniquely position each entity object. In our project, each entity object is inserted into the JSON database based on their unique identification number.

To showcase the application's deals functionality, the project will also utilise Telegram as a platform to draw food deals data. The project's application will display food deal messages from the Telegram channel, EatNow!, which hosts food deals available nationwide.

To update the project's cloud platform, the project will also utilise Python modules, including Telethon API, Google Maps API, and OneMapAPI, as backend services, which will be continuously running in the background to extract the details of new food deals from the EatNow! Telegram channel.

1.4.2 Application

EatNow! application currently supports Android OS running Android Lollipop (API level 21).

EatNow! is currently only available in English.

EatNow! should also be designed to display a large dataset of restaurants in Singapore, which holds more than 10,000 data points.

Users' mobile devices should have internet connection when using the application.

Users are required to register for or login into their accounts if they want to access additional functionalities such as making a reservation.

1.5 Initial UI Mockups

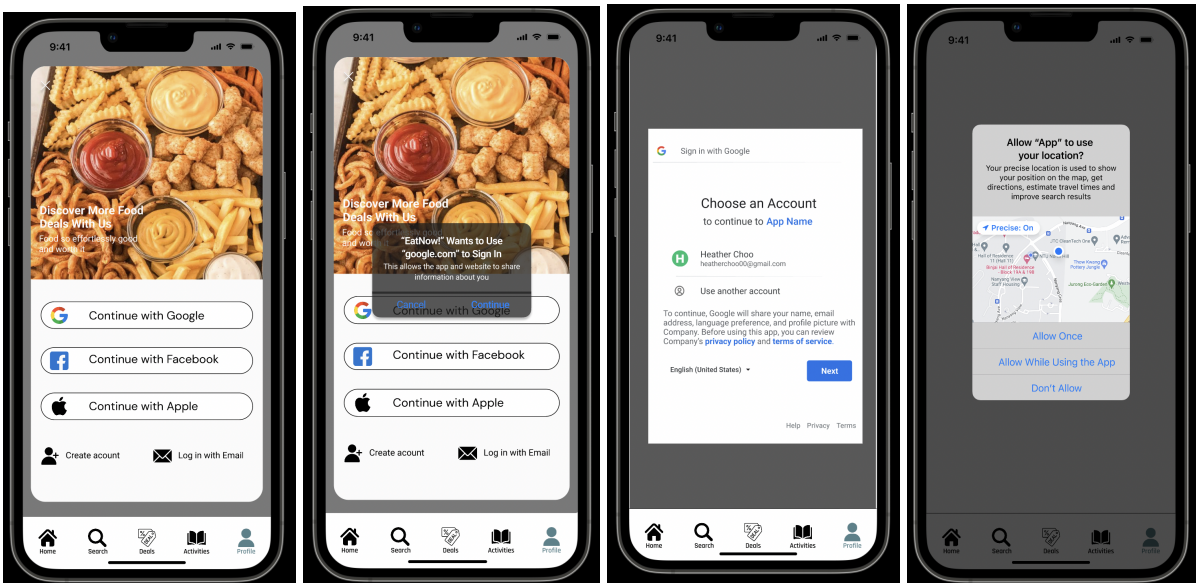


Figure 1: Login and Registration Page

Figure 1 shows the user interface for login and registration. To facilitate this process, users are allowed to login and register with Google, Facebook or Apple ID. Users are also given the option to register for and login into an account by manually keying in details into the system. After registration and login, users will be prompted to give location permissions.

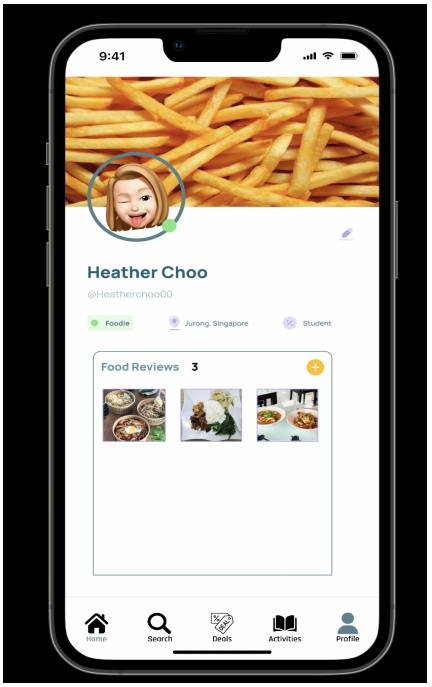


Figure 2: Profile Page

Figure 2 shows the interface for our profile page. This page shows users' profile images, which can be changed. There is also a section displaying food reviews made by the user.

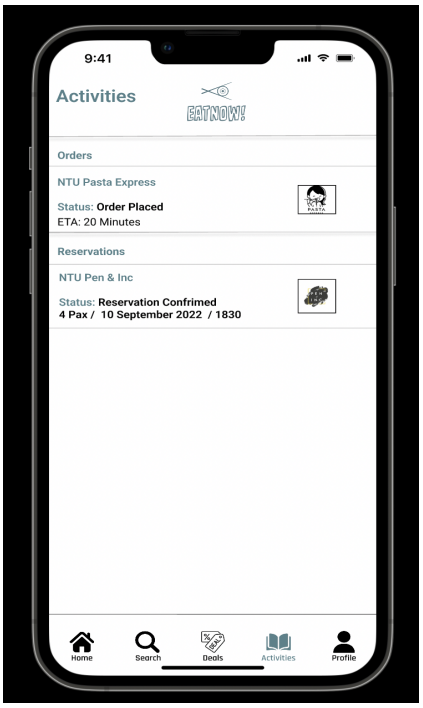


Figure 3: Activities Page

Figure 3 shows the user's activities page, which displays the reservations and orders made by users.

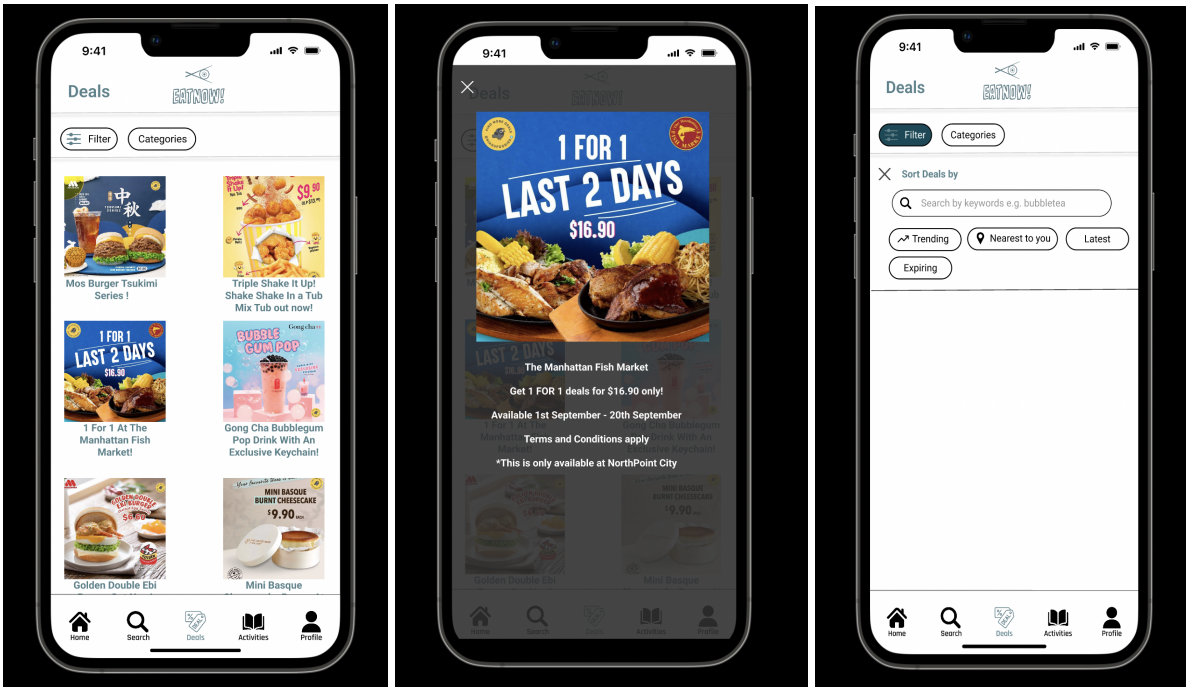


Figure 4: Deals Page

Figure 4 shows the deals page where users can view the different deals available. By tapping on a deal, details on deals will popup. There is also a filter option for users to filter the deals according to brand, cuisines, and location.

Figure 5: Search Page

Figure 5 shows the search page which displays popular restaurants to dine at and a list of restaurants. Users are able to search for their user profiles and desired restaurants. This page allows users to filter restaurants based on category, area, and whether or not it's reservable. Categories include cuisines and prices. Users can also filter restaurants near them using location services.

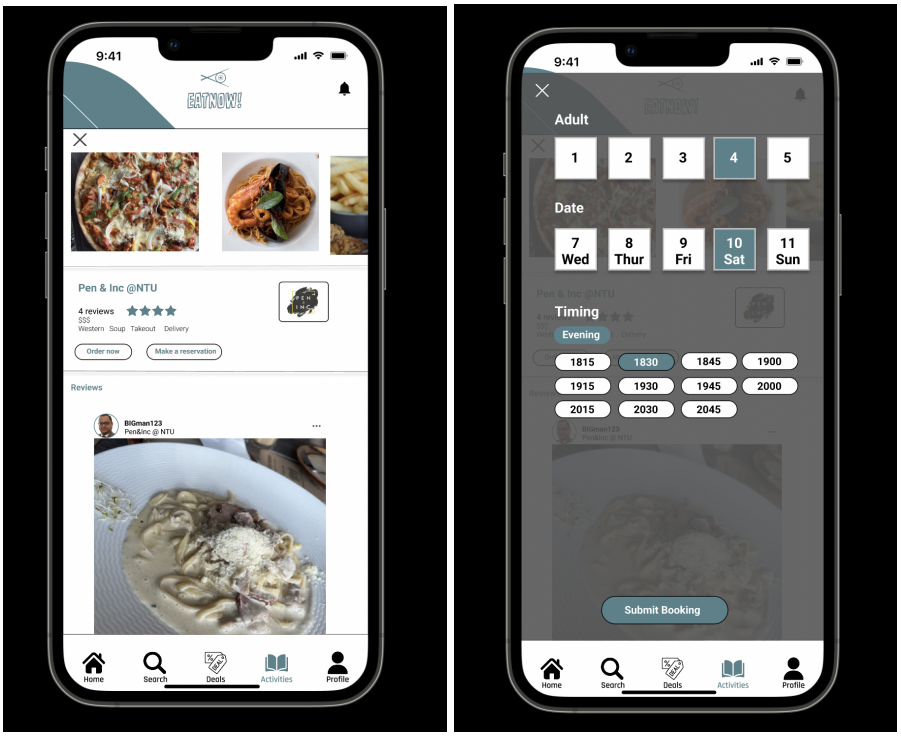


Figure 6: Restaurant reservation UI

Figure 6 shows the restaurant reservation UI. This page displays details and reviews of a particular restaurant. Users may make a reservation at a restaurant by selecting the number of diners, date and time.

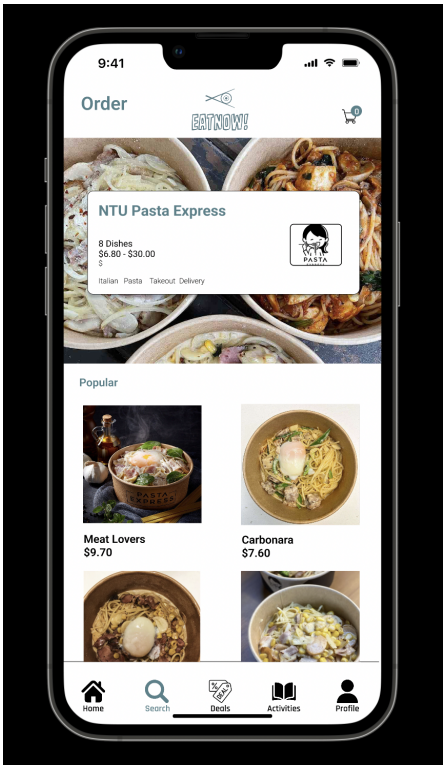


Figure 7: Placing orders UI

Figure 7 shows the UI for placing food orders. Upon selecting a restaurant to order food from, users can select the food items they want to order before placing the order.

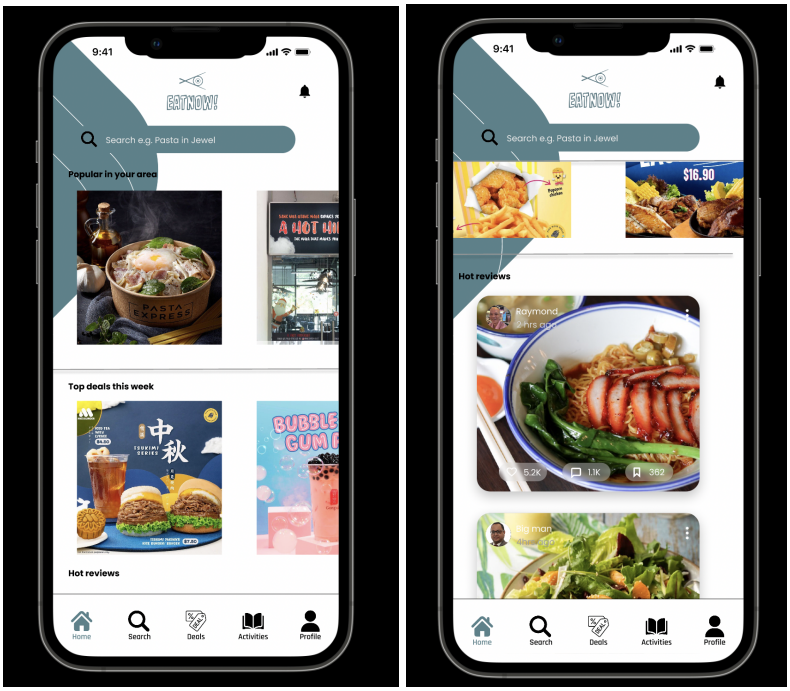


Figure 8: Home Page

Figure 8 shows the home page which displays restaurants popular in the user's current area, weekly top deals, and reviews made by other users of the app.

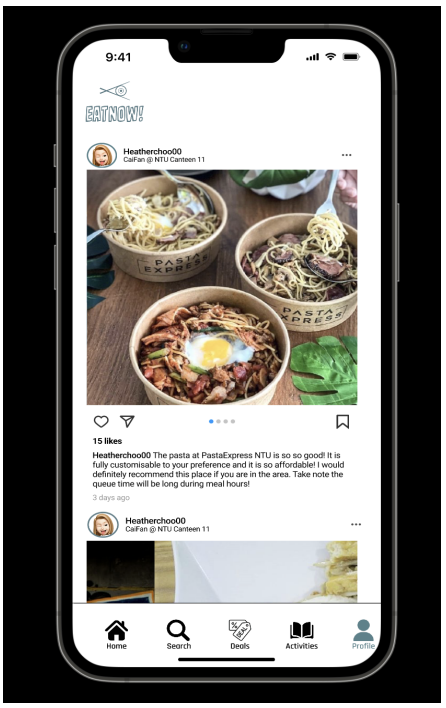


Figure 9: Reviews UI

Figure 9 shows the UI when users view a review. This page displays detailed review comments. Users are able to like, save, comment on and forward a review.

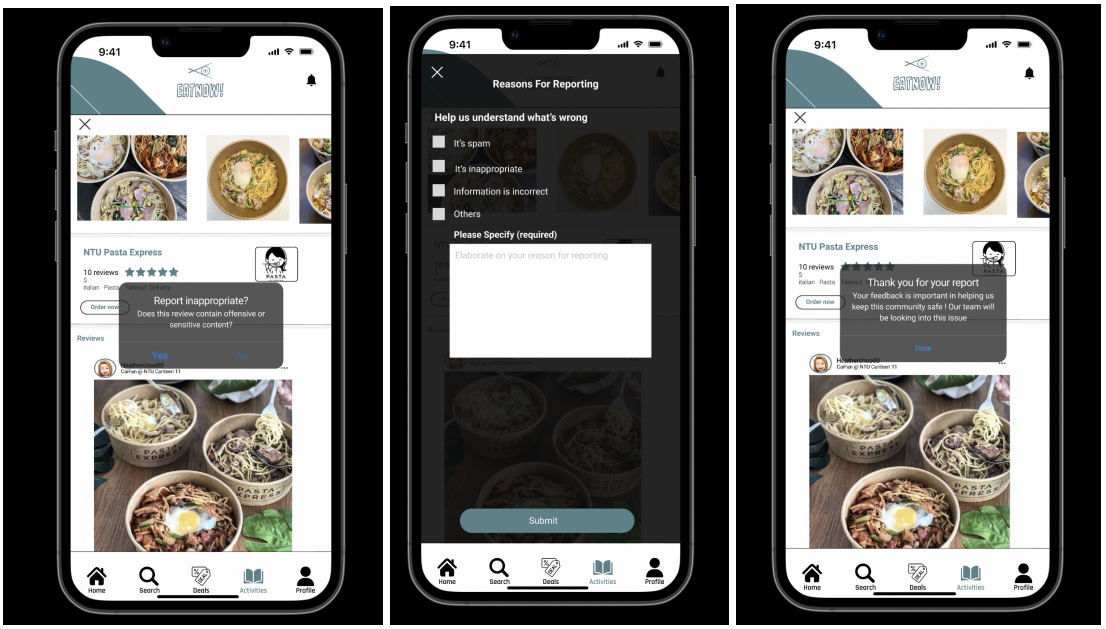


Figure 10: Report Review

Figure 10 shows the UI for reporting a review. Users are able to report reviews they deem inappropriate. Users can indicate the reasons for reporting and provide specific reasons for why a particular post should be taken down. After submitting the review, a popup thanking the user will be displayed.

2. Functional Requirements

1. Data System

1.1. The system must extract details of food deal messages from the Telethon API into a Realtime Firebase, in the following format:

Image of food deal
Details of food deal
Time of message sent
Cuisine of food in the deal
Timeframe of food deal
Location of food deal

1.1.1. The system must update the food deal details whenever there is a new food deal message.

1.2. The system saves account details into the Realtime Firebase in the following format:

User's Account ID
Email
Follower count
Following count
Password
Reviews by user in the form of ReviewIDs
Username

1.3. The system saves reservation details into the Realtime Firebase in the following format:

User's Account ID
Reservation ID
Restaurant ID
Date of Reservation Request
Number of Pax
Date of Reservation

1.4. The system saves reviews made by users into the Realtime Firebase in the following format:

User's Account ID
Caption
Date of review
Image
Number of likes
Rating
Restaurant ID
Review ID
Comment

1.5. The system must extract details of restaurants in Singapore into the Realtime Firebase in the following format:

Restaurant's address
Area
Category
Features
Image link
Latitude
Longitude
Name of restaurant
Phone number to enquire
Postal Code
Ratings (out of 5)
Restaurant ID
Number of reviews
Number of existing reservations
Restaurant website link

1.5.1. The system must update the restaurant details monthly.

2. Account

2.1. The application must allow users to register for an account.

2.1.1. The application must ensure that users fill up a registration form using the following format:

Email Address
Username
Create Password
Confirm Password

2.1.2. During registration, the application must validate the account registration.

2.1.3. The email address must not be in use.

2.1.4. The username must be unique.

2.1.5. The username must be 6-20 characters long.

2.1.6. The password must follow the criteria below:

2.1.6.1. The password must be 8-20 characters long.

2.1.6.2. The password must have at least 1 alphabet (a - z)

2.1.6.3. The password must have at least 1 special character (!?#$%&*)

2.1.6.4. The password must have at least 1 digit (0 - 9)

2.2. The application must allow users to sign in to their account.

2.2.1. The application must ensure that users fill up their account details in the following format:

Username
Password

2.2.2. The application must validate the account details.

2.3. The user must be able to add their profile picture.

2.3.1. The image must be selected from the user's device's gallery.

2.3.2. If no image is included, a default image must be used.

2.4. The account profile must also display a list of reviews created by the user.

3. Display deals

3.1. The application must display the deals according to the timeframe.

3.2. The application must display the photo of the deal from the original source, with its respective brand underneath it.

3.3. When clicked on the photo of the deal, the application must provide a summarised description of the deal in a pop-up in the following format:

Image of the deal
Brand
Deal details
Cuisine
Timeframe
Location(s)

3.4. The application must remove deals that have expired.

4. Display Restaurants

4.1. The application must display a selectable list of restaurants sorted based on ratings.

4.1.1. The rating for each restaurant is calculated through dividing the total sum of reviews' ratings by the number of reviews;

4.1.2. Each row in the list must display the following information:

Name of restaurant
Image of restaurant
Number of reviews
Rating of restaurant
Categories of restaurant

4.2. The application must display the full details of the restaurant upon selection.

4.2.1. The display must include:

Name of restaurant
Image of restaurant
Number of reviews
Rating of restaurant

General area of restaurant
Categories of restaurant
Features of restaurant
Phone number of restaurant
Website link of restaurant

4.2.2. The display must also display a list of reviews for that restaurant.

5. Display Users

5.1. The application must display a selectable list of users sorted based on followers.

5.1.1. Each row in the list must display the following information:

Username of user
Profile image of user

5.2. The application must display the following details of the user upon selection:

Username of user
Profile image of user
Number of followers
Number of followings

5.2.1. The display must also display a list of reviews created by that user.

6. Filtering

6.1. The application must allow the user to filter the restaurants.

6.1.1. The filters must include:

Price rate of restaurant
Cuisine of restaurant
General area of restaurant
Features of restaurant

6.1.2 The filters can be stacked

6.2. The application must allow the user to filter the deals.

6.2.1. The filters must include:

Brand of F&B restaurant hosting the deal
Cuisine of food that the deal includes
General area of deal

.

6.2.2. The filters can be stacked.

7. Searching

7.1. The application must allow the user to search for restaurants.

7.1.1. The search must be performed by matching the restaurants' name.

7.2. The application must allow the user to search for users.

7.2.1. The search must be performed by matching the users' name.

8. Reviews

8.1. A Review feed must be displayed on the home page.

8.1.1. The application must allow users to filter the review feed.

8.1.2. The filter must include:

Viewing all reviews
Viewing reviews created by users that the user is following

8.2. The full details of the review must be displayed upon user's selection.

8.2.1. The review must be clickable from all displays.

8.2.2. The application must display the following details of the reviews:

Image of review
Date of review
Number of likes of review
Rating of review
Caption of review
Text of review
Image of user that created the review
Name of user that created the review
Image of restaurant the review is created for

Name of restaurant the review is created for

8.3. The selected review must be interactable by the user.

8.3.1. The application must allow users to like or unlike the review

8.3.2. The application must allow users to comment on the review.

8.3.3. The application must allow users to report the review if inappropriate.

8.3.4. The application must allow users to bookmark the review.

8.4. The application must allow users to add a review upon viewing a restaurant.

8.4.1. The review must include the following information:

Caption
Text
Rating

8.4.2. The number of characters for the review's caption must be capped at 100.

8.4.3. The number of characters for the review's text must be capped at 300.

8.4.4. The rating for the review must be capped at 5 with steps of 0.5..

8.4.5. The application must allow the user to include an image for the review.

8.4.5.1. The image must be selected from the user's device's gallery.

8.4.5.2. If no image is included, a default image must be used.

9. Following System

9.1. The application must allow users to follow other users upon viewing the user's profile.

9.1.1. The application must not allow the user to follow his/herself.

9.2. The application must allow the user to view his/her followers.

9.3. The application must allow the user to view his/her followings.

10. Reservations

10.1 The application must enable users to book a reservation at restaurants upon viewing a restaurant.

10.1.1. The reservation must include the following information:

Selected date of reservation
Selected timeslot of reservation

Number of pax

10.1.2. The selectable range of dates for the reservation must be capped at 30 days.

10.1.2.1. The application must only allow the user to book a date from the following day onwards.

10.1.3. The selectable range of timeslots for the reservation must be capped based on the restaurant's opening hours.

10.1.3.1. The time intervals of each timeslot must be 30 minutes.

10.1.4. The number of pax must be limited to a range between 1 to 10.

10.2. The application must display a list of the user's active reservations in the following format:

Image of Restaurant that the reservation is for
Date of Reservation
Number of Pax
Date of Creation

11. Orders

11.1. The application must enable users to order food menu items for collection at restaurants upon viewing a restaurant.

11.1.1. The menu must include the following information:

Name of the food
Price of the food in Singapore Dollars ($), rounded off to 2 decimal places
Description of the food

11.1.2. The selection of food menu items can be stacked.

11.2. The application must also enable users to order the deal upon viewing a food deal.

11.2.1. The application must allow selection of restaurants with the food deal.

11.2.2. Upon selection, the application must forward to the restaurant's menu display page for further selection of food menu items.

12. Payment

12.1. The application must enable users to make a payment after the order has been made.

12.1.1. Payment must be done only through credit card.

12.1.2. Details of the payment must be displayed to the user. These include:

User's selected food menu items
Total price of selected food menu items in Singapore Dollars ($), rounded off to 2 decimal places
Collection details
Credit card information, including number and holder's name

12.2. The application must allow users to add a credit card for payment.

12.2.1. The application must display an interface for user's input in the following format:

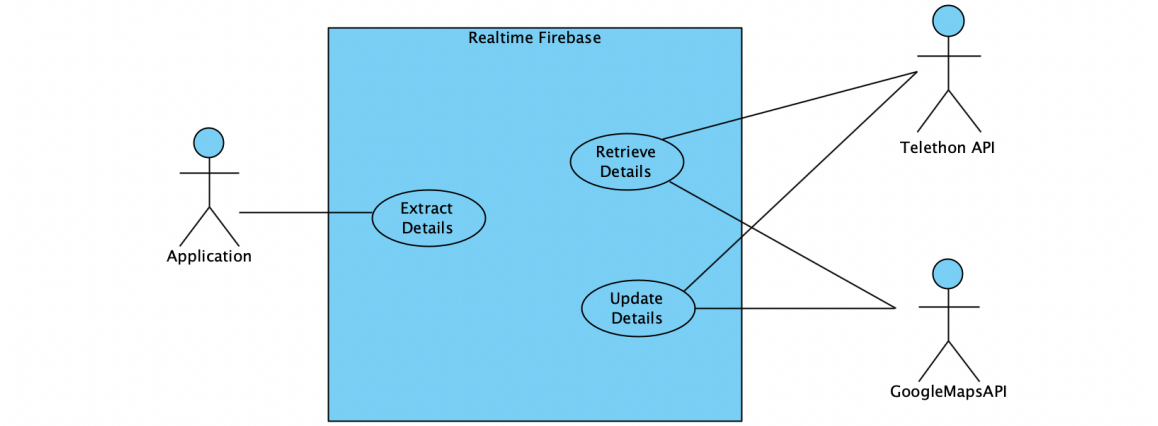
Credit Card Number (16 Digits)
Expiry Date
Card Holder's Name
Credit Card Verification Value 2 (CVV2)

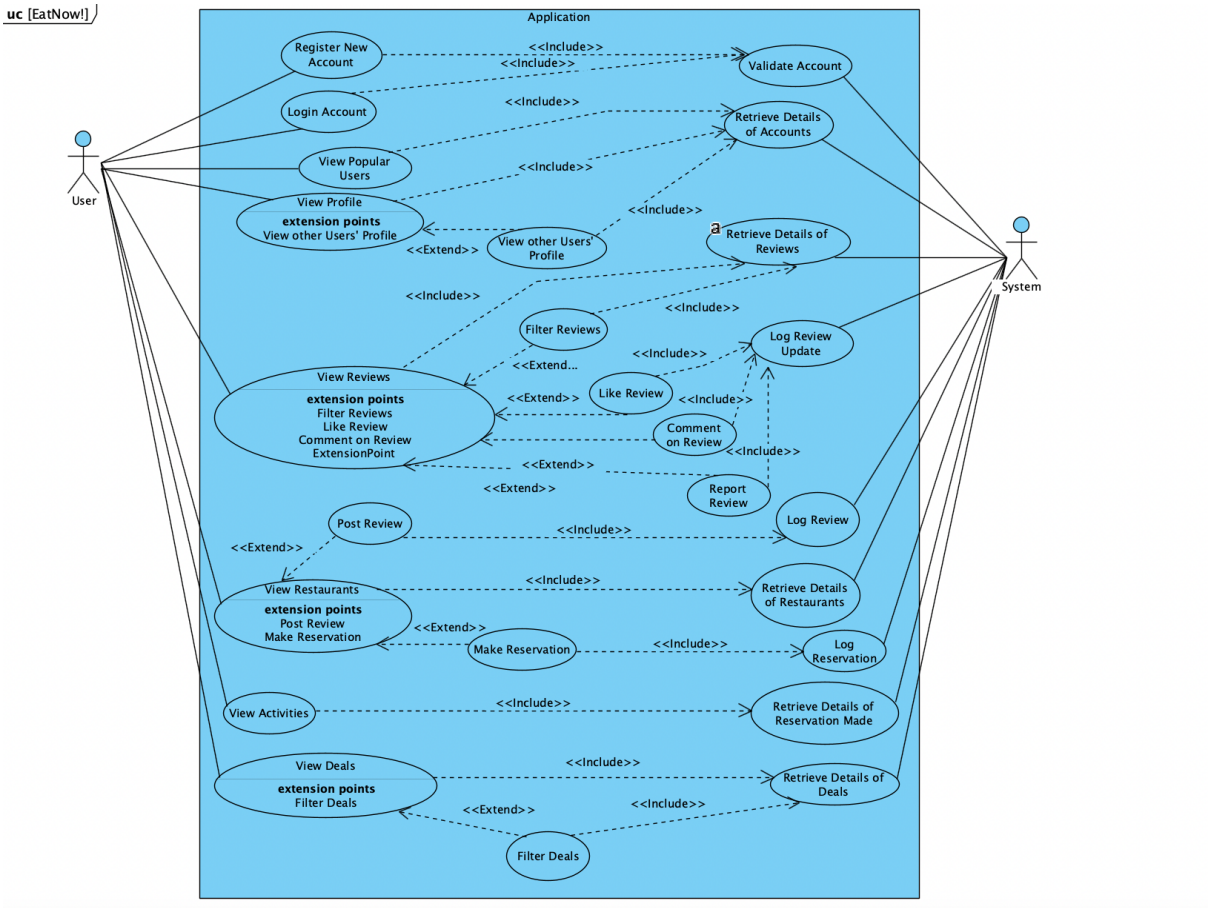
12.2.2. Upon adding the credit card, the application must validate the user's credit card information.

12.2.3. The application must not store the user's credit card information.

12.3. The application must display a receipt upon confirming the order.

2.1. Use Case Diagrams





2.2. Use Case Description

Use Case ID:	1		
Use Case Name:	Extract Details of Food Deals		
Created By:	Justin Chua	Last Updated By:	Ong Jun An
Date Created:	7 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	System, TelethonAPI
Description:	System extracts the relevant details of the food deals from the Telethon API through the channel, 'SG EatNow!'
Preconditions:	1. System first launches to create the Realtime Firebase
Postconditions:	1. System successfully input data into the Realtime Firebase
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. System extracts the food deal messages via the TelethonAPI. 2. System extracts the relevant details of food deals from the messages. 3. System adds the data of food deals into the Realtime Firebase.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	2		
Use Case Name:	Update Details of Food Deals		
Created By:	Justin Chua	Last Updated By:	Ong Jun An
Date Created:	7 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	System, TelethonAPI
Description:	System receives new food deal messages from the Telethon API, and extracts details of the new food deal to update the Realtime Firebase.
Preconditions:	1. New message is received through the channel, 'SG EatNow!'.
Postconditions:	1. System successfully adds new data into the Realtime Firebase.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. New message is added into the channel,'SG EatNow!'. 2. System is prompted to update the food deal database via TelethonAPI. 3. System extracts relevant details of food deals from the new message. 4. System adds the data of food deals into the Realtime Firebase.

	5.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	1. The Realtime Firebase is kept running.
Notes and Issues:	-

Use Case ID:	3		
Use Case Name:	Extract Details of Restaurants		
Created By:	Ong Jun An	Last Updated By:	Ong Jun An
Date Created:	21 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	System, GoogleMapsAPI
Description:	System extracts the relevant details of the Food & Beverages restaurants from the GoogleMapsAPI, and adds the data to the Realtime Firebase.
Preconditions:	1. System first launches to create the Realtime Firebase
Postconditions:	2. System successfully input data into the Realtime Firebase
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. System extracts the details of all restaurants in Singapore via GoogleMapsAPI. 2. System further filters the relevant details of the restaurants from the dataset. 3. System adds the data of restaurants into the Realtime Firebase.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	4		
Use Case Name:	Update Details of Restaurants		
Created By:	Ong Jun An	Last Updated By:	Ong Jun An
Date Created:	21 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	System, GoogleMapsAPI
Description:	System extracts the relevant details of the Food & Beverages restaurants from the GoogleMapsAPI on a monthly basis.
Preconditions:	1. System is prompted to update the restaurant database.
Postconditions:	1. System successfully inputs data into the Realtime Firebase.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. System verifies that the last update was a month ago. 2. System will extract the details of all restaurants in Singapore via GoogleMapsAPI.

	3. System replaces the data of restaurants in the Realtime Firebase.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	1. The Realtime Firebase is kept running.
Notes and Issues:	2. Since all restaurants in Singapore are extracted, the runtime will be long.

Use Case ID:	4		
Use Case Name:	Account Registration		
Created By:	Justin Chua	Last Updated By:	Jewel Chin
Date Created:	5 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	User, System
Description:	User registers for a new account.
Preconditions:	1. User account must not exist in the database. and 2. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	-
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. User selects ‘Don’t have an account?’ 2. ‘Join Us!’ registration pop-up appears. 3. User enters desired username, email address, password. 4. User enters the password again for confirmation of the password. 5. User presses the ‘Register’ button.
Alternative Flows:	AF-S2-2: User selects registration via Google. 1. System displays Google login page. 2. User enters a Google email and password. 3. User presses the 'Register’ button. 4. System checks the validity of Google account. 5. Return to step 6. AF-S4: User can choose to allow the password to be unhidden. 1. Return to step 4. AF-S5: System detects that either username, email address, and/or password is invalid. 1. System displays an error message in red above the respective input. 2. Return to step 2. AF-S5: System detects that the password does not match the confirmation password.

	1. System displays an error message in red above the respective input. 2. Return to step 2.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	5		
Use Case Name:	Check Registration		
Created By:	Justin Chua	Last Updated By:	Jewel Chin
Date Created:	7 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	System
Description:	System validates registration details and creates an account for users.
Preconditions:	1. User account must not exist in the database. and 2. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. User account is successfully created and stored in the database.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. System validates login details. 2. System validates account availability. 3. System creates an account for the user in the database.
Alternative Flows:	AF-S1-1: System detects that username exists in the database. 1. System displays an error message “Username is taken.” 2. User re-enters registration details. 3. Return to step 1. AF-S1-1: System detects that email exists in the database. 4. System displays an error message “Email is taken.” 5. User re-enters registration details. 6. Return to step 1. AF-S1-2: System detects weak password. 1. System displays error message “Password is weak. - The password should be 8-20 characters long. - The password should have at least 1 alphabet. - The password should have at least 1 special character. - The password should have at least 1 digit.” 2. User re-enters registration details. 3. Return to step 1.
Exceptions:	-
Includes:	-

Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	6		
Use Case Name:	Account Login		
Created By:	Justin Chua	Last Updated By:	Justin Chua
Date Created:	8 Nov 2022	Date Last Updated:	8 Nov 2022

Actor:	User, System
Description:	User signs into existing account.
Preconditions:	1. User account must exist in the database. and 2. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. User is logged into account and is directed to profile page.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. User clicks on app icon to launch app from home screen. 2. User enters email and password. 3. User clicks ‘Log in’ button. 4. System logs user into account and displays profile page.
Alternative Flows:	AF-S1: User selects Forgot password. 1. System prompts user to enter registered email. 2. User enters email. 3. System resets password of account and sends default password to user’s email. 4. User retrieves new password from email. 5. Return to step 2.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	7		
Use Case Name:	Check Login		
Created By:	Justin Chua	Last Updated By:	Justin Chua
Date Created:	8 Nov 2022	Date Last Updated:	8 Nov 2022

Actor:	System
Description:	System validates registration details and creates an account for users.
Preconditions:	1. User account must exist in the database. and

	2. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. User account is successfully logged into.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. System validates login details. 2. System validates account existence. 3. System logs into account and displays profile page.
Alternative Flows:	AF-S1-1: System detects that username does not exist in the database. 7. System displays an error message “Invalid login” 8. User re-enters login details. 9. Return to step 1. AF-S1-2: System detects that password does not exist in the database. 10. System displays an error message “Invalid login” 11. User re-enters login details. 12. Return to step 1.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	8		
Use Case Name:	Change profile photo		
Created By:	Justin Chua	Last Updated By:	Justin Chua
Date Created:	8 Nov 2022	Date Last Updated:	8 Nov 2022

Actor:	User
Description:	User changes profile image from default to desired image.
Preconditions:	1. User account must exist in the database. And 2. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. Profile photo is changed to desired profile image.
Priority:	Medium
Frequency of Use:	Monthly
Flow of Events:	1. User navigates to profile tab. 2. User clicks on settings icon. 3. User selects the option to change profile image. 4. System opens user’s phone photo gallery. 5. User selects desired image. 6. System sets selected image as new profile photo.
Alternative Flows:	-
Exceptions:	-

Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	9		
Use Case Name:	View Activities		
Created By:	Justin Chua	Last Updated By:	Justin Chua
Date Created:	8 Nov 2022	Date Last Updated:	8 Nov 2022

Actor:	User, System
Description:	System displays User's upcoming reservations and orders.
Preconditions:	1. User account must exist in the database. and 2. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. Users can view their upcoming reservations and orders.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. User navigates to activities tab. 2. System displays upcoming reservations and orders from database.
Alternative Flows:	-
Exceptions:	AF-S2: User has no reservations or orders. 1. System displays a blank page with "No upcoming activities" text.
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	10		
Use Case Name:	Display Deals		
Created By:	Justin	Last Updated By:	Ong Jun An
Date Created:	7 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	System
Description:	System displays deals under Deals tab.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. System displays the list of deals available.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. User navigates to the Deals tab. 2. System displays all available deals according to how recent the deal is.

	3. Clicking on an available deal will show a pop-up page that displays the deal's specific details.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	11		
Use Case Name:	Filtering the deals available		
Created By:	Ong Jun An	Last Updated By:	Ong Jun An
Date Created:	7 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	User
Description:	To allow the User to filter the deals available by categories provided by the System.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	2. A filtered list of deals will be displayed by the system based the User's selection.
Priority:	High
Frequency of Use:	1-2 times daily
Flow of Events:	1. System will provide a selection of categories that can be filtered by the System. 2. Users can filter the list of deals based on brand, cuisines and location. 3. Users will select on the types of filters. 4. System will display the filtered list of deals based on the User's selection.
Alternative Flows:	AF-S4: User provides a change in the types of filters to be used. 1. Filter is changed based on user's selection. 2. Return to step 4. AF-S4: User taps on the 'Reset filters' button. 1. User can remove all filters in his/her selection. 2. Return to step 1.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	12		
Use Case Name:	View Reviews		
Created By:	Sean Lim	Last Updated By:	Jewel Chin
Date Created:	1 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	User

Description:	View reviews made by other users. Users should be able to interact with the reviews.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. Reviews seen should be filtered out with new reviews
Priority:	High
Frequency of Use:	1-2 times daily
Flow of Events:	1. Users will click into the restaurant of their choice 2. Users will then be able to view the reviews of that restaurant made by other users.
Alternative Flows:	AF-S1: Users can also view reviews at the home page 1. Users scrolling down the home page will view trending reviews recently made by other users. 2. Users can choose to filter to view only reviews made by users they are following.
Exceptions:	-
Includes:	1. Like a Review 2. Comment on a Review 3. Report a Review
Special Requirements:	-
Assumptions:	1. Reviews are constantly made by users.
Notes and Issues:	-

Use Case ID:	13		
Use Case Name:	Like Reviews		
Created By:	Sean Lim	Last Updated By:	Justin Chua
Date Created:	1 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	User
Description:	User must have the option to interact with the reviews made by other users in terms of “likes”. The more “likes” the review has, the algorithm will deem the post trending and it will more likely appear at the top of other user’s home page.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data) And 2. User must be signed in.
Postconditions:	1. Number of likes on a review will increase.
Priority:	High
Frequency of Use:	1-2 times daily
Flow of Events:	1. User view the reviews made by other users in the homepage. 2. User clicks on review for detailed comments. 3. Users is able to tap on the Like icon and give the review a like. 4. The number of likes for that review will increase.
Alternative Flows:	AF-S1: The user is not signed in. 1. User double tap to like a review without being signed in. 2. Users will be redirected to the account login/ registration page.
Exceptions:	-

Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	14		
Use Case Name:	Comment a Review		
Created By:	Sean Lim	Last Updated By:	Ong Jun An
Date Created:	1 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	User
Description:	Users must be able to comment on a post.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data) And 2. Users must be signed in.
Postconditions:	1. Number of comments must increase. 2. Comment is added into the review.
Priority:	High
Frequency of Use:	1-2 times daily
Flow of Events:	1. User clicks on the comment button. 2. System must display a pop-up textbox window for user's inputs 3. User inputs the text for the comment. 4. User clicks on the confirm button. 5. Comment is added into the review.
Alternative Flows:	AF-S1: The user attempts to input more than 200 characters. 1. System must limit the textbox to 200 characters. 2. User is unable to input more than 200 characters. 3. Return to step 4.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	15		
Use Case Name:	Report Review		
Created By:	Sean Lim	Last Updated By:	Justin Chua
Date Created:	1 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	User
Description:	Users will be able to report a post. The post may contain offensive or sensitive content. The post will then be reviewed and may be taken down if deemed offensive or sensitive.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data)

	2. Users must be signed in.
Postconditions:	1. The post will be flagged in the system.
Priority:	Medium
Frequency of Use:	1 time weekly
Flow of Events:	1. User clicks on a review to view detailed comments. 2. User deems the review to have offensive or sensitive content. 3. User presses the Report button. 4. Users are prompted to submit a brief description of why the user found the post offensive or sensitive. 5. A popup notification informs the user that the report has been submitted.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	1. There will be someone to review the reports and deem which reviews are indeed offensive or sensitive.
Notes and Issues:	-

Use Case ID:	16		
Use Case Name:	Following A Friend		
Created By:	Ong Jun An	Last Updated By:	Justin Chua
Date Created:	21 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	User, Friend
Description:	To allow the user to be able to follow another user.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data) And 2. User must log-in to his/her account.
Postconditions:	1. System has successfully followed a user. 2. System will update the following count of the user.
Priority:	High
Frequency of Use:	3-4 times weekly
Flow of Events:	1. User searches for the username of the user he wishes to follow in the Search page. 2. System will show the user and their profile picture under the Popular Users section. 3. User will click on it and will be sent to the user's profile page. 4. User will click the "Follow" button.
Alternative Flows:	AF-S1: No search result was given. 1. User will edit the search query input. 2. Return to step 1.
Exceptions:	AF-S4-1: The user is not signed in. 1. System displays a message to prompt the user to log in/create an account. 2. Return to step 3.

	AF-S4-2: User attempts to follow own profile 1. System displays error popup “You cannot follow your own profile”. 2. Return to step 3.
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	17		
Use Case Name:	Booking a Reservation with a Restaurant through EatNow!		
Created By:	Ong Jun An	Last Updated By:	Jewel Chin
Date Created:	7 Sep 2022	Date Last Updated:	8 Nov 2022

Actor:	User, Restaurant
Description:	To allow the user to be able to book a reservation with a restaurant specified by the user
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data) And 2. User must log-in to his/her account.
Postconditions:	1. System has successfully booked a restaurant for the user. Or 2. System is unable to make a reservation due to the restaurant’s unavailability.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. User browses through the Restaurants section of the Search page and picks a restaurant. 2. User is redirected to a page which shows the restaurant’s details. 3. User clicks on the “Make a Reservation” button. 4. Users will be shown a pop-up. 5. User will pick a date from the calendar provided and select the number of pax. Users will click on the View Timeslots button. 6. Users will be shown a pop-up of the available time slots. 7. User will validate and confirm the details of the booking by tapping on the tick button on the top right. 8. System will notify the Restaurant on the details of the booking. 9. System will update the user’s active reservations in the Activities page.
Alternative Flows:	AF-S3: User is not signed in. 1. System displays an error message, prompting the user to log in/create an account. 2. Return to Step 2. AF-S5: Time slots are all unavailable. 1. Users will not be able to make a reservation. 2. Return to step 3.
Exceptions:	EX1-Restaurant is unavailable for booking.

	1. System will notify the User with a message “Restaurant is unavailable for booking.”.
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	18		
Use Case Name:	Search for Restaurants and Users		
Created By:	Justin Chua	Last Updated By:	Justin Chua
Date Created:	8 Nov 2022	Date Last Updated:	8 Nov 2022

Actor:	User, System
Description:	Users can search for restaurants and user’s profiles in database.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. System displays search results of restaurants and user profiles based on user’s search input.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. User navigates to search tab. 2. User inputs restaurant name or profile usernames in search bar. 3. System searches database using search input. 4. System displays search results.
Alternative Flows:	-
Exceptions:	1. System displays blank page if none of the restaurants or account usernames in database match the search input.
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	19		
Use Case Name:	Filter Restaurants		
Created By:	Justin Chua	Last Updated By:	Justin Chua
Date Created:	8 Nov 2022	Date Last Updated:	8 Nov 2022

Actor:	User, System
Description:	Users can filter the restaurants based on selected filters
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data)
Postconditions:	1. System displays a list of restaurants based on filters selected.
Priority:	High
Frequency of Use:	Daily
Flow of Events:	1. User navigates to search tab.

	2. User clicks on filter button. 3. User can select filters to restaurants such as category (cuisines and price), area, and features (reservable or non-reservable). 4. After selection, users click on ‘set filter’. 5. System displays list of restaurants from database that pass the filters.
Alternative Flows:	-
Exceptions:	1. System may not display any restaurants if none of the restaurants in database pass the filters set.
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

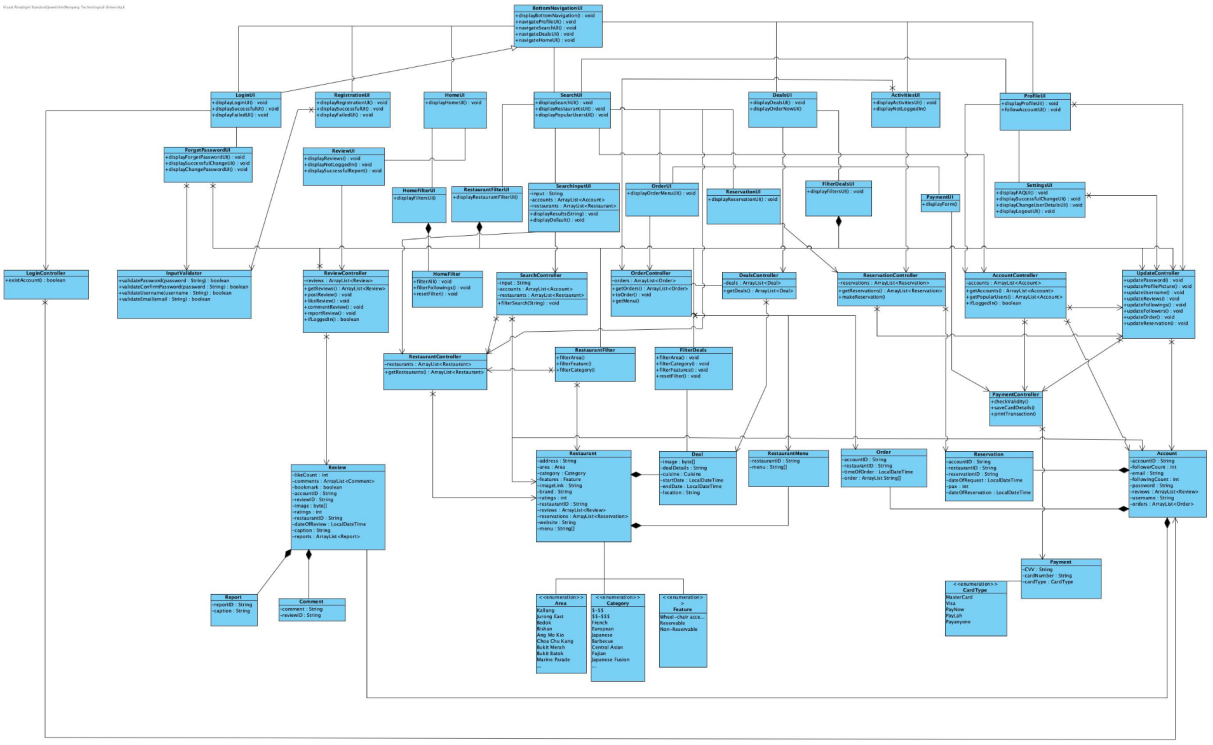
Use Case ID:	20		
Use Case Name:	Order Food		
Created By:	Justin Chua	Last Updated By:	Justin Chua
Date Created:	8 Nov 2022	Date Last Updated:	8 Nov 2022

Actor:	User, System
Description:	Users can order food through deals tab or search tab.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data) 2. Users must be logged into their accounts.
Postconditions:	1. System displays payment page for users to make payment to confirm successful order.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. User navigates to deals tab. 2. User clicks on desired deal. 3. User presses on ‘Order now’ button. 4. System displays food menu for user to select. 5. After selection, users click on ‘Make Payment’ button. 6. System displays payment page.
Alternative Flows:	AF-S1: User navigates to search tab 1. User searches for desired restaurant. 2. User clicks on restaurant they want food to be ordered from. 3. Return to step 3.
Exceptions:	EX-S5: User only wants to view food menu 1. User presses back button and do not click on ‘make payment’ button.
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	21		
Use Case Name:	Order Payment		
Created By:	Justin Chua	Last Updated By:	Justin Chua
Date Created:	8 Nov 2022	Date Last Updated:	8 Nov 2022

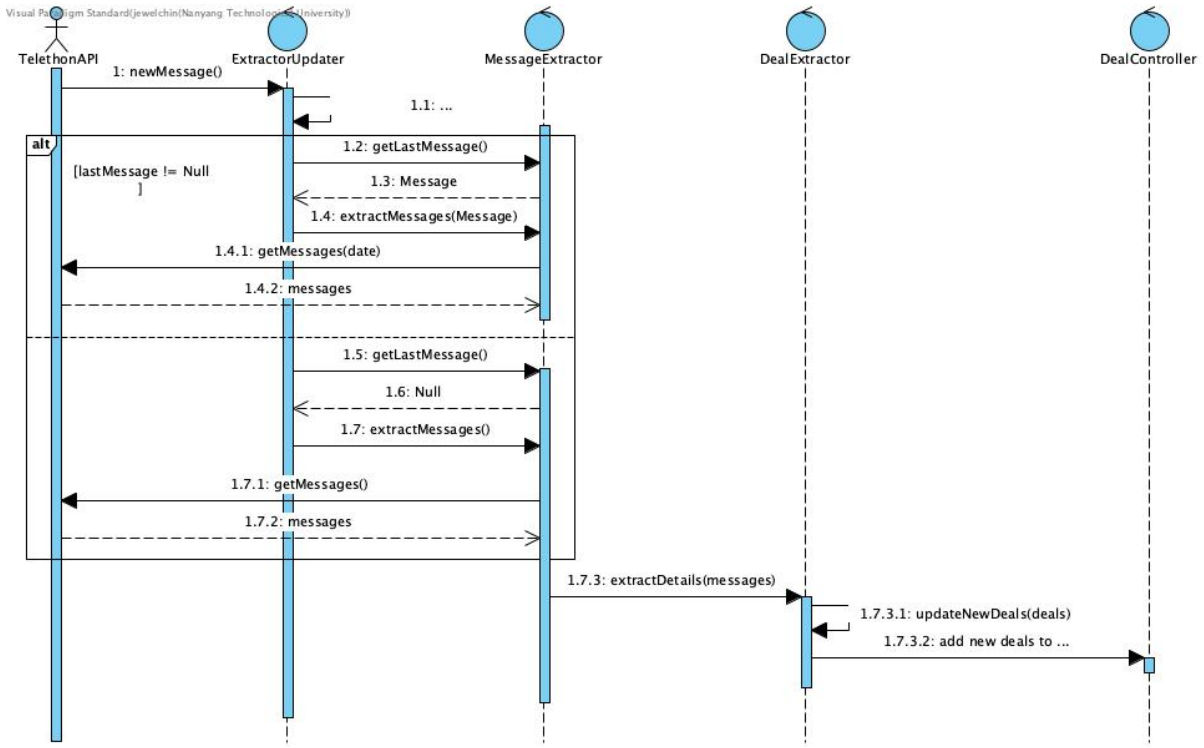
Actor:	User, System
Description:	Users can make payment online for the food they want to order.
Preconditions:	1. Mobile device must be connected to an internet connection (Wifi/Cellular Data) 2. Users must log into their accounts.
Postconditions:	1. System successfully records food order and payment. 2. Food order appears in activities tab.
Priority:	Very High
Frequency of Use:	Daily
Flow of Events:	1. User presses ‘make payment’ button after selecting the food to be ordered. 2. System displays payment page. 3. User enters card details. 4. User clicks on ‘Proceed to pay’ button. 5. System displays confirmation receipt.
Alternative Flows:	AF-S4: Card details invalid 1. System displays error message “Invalid card”. 2. Return to step 3.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

2.3. Class Diagram

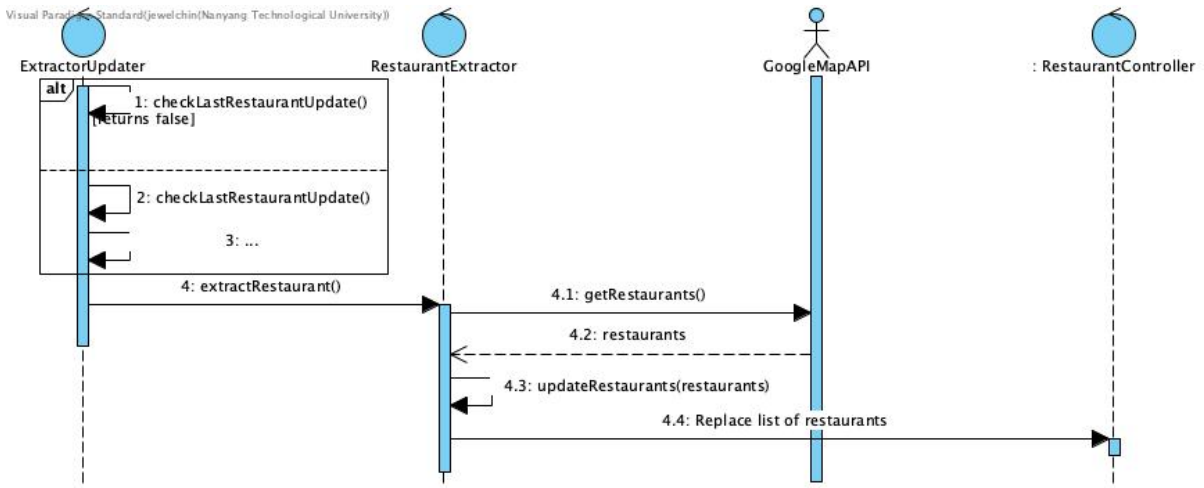


2.4. Sequence Diagram

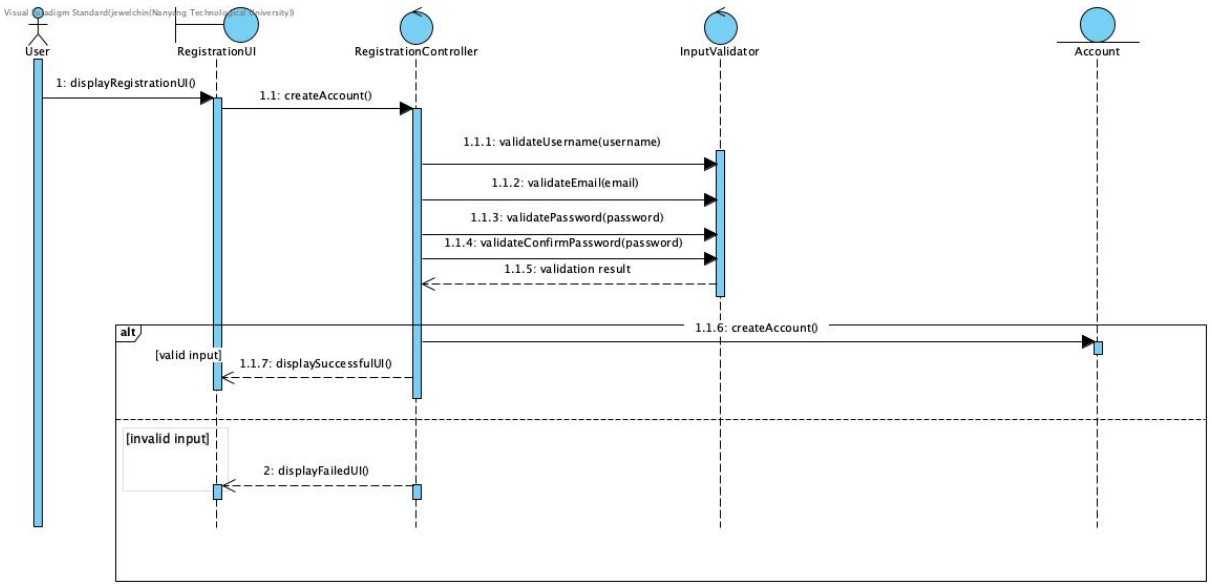
1. Extract and Update Food Deals



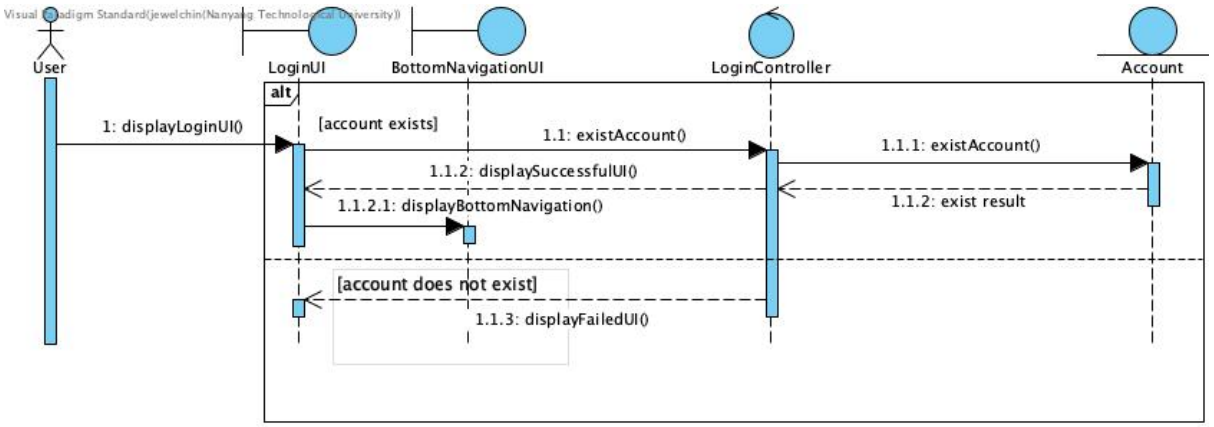
2. Extract and Update Restaurants



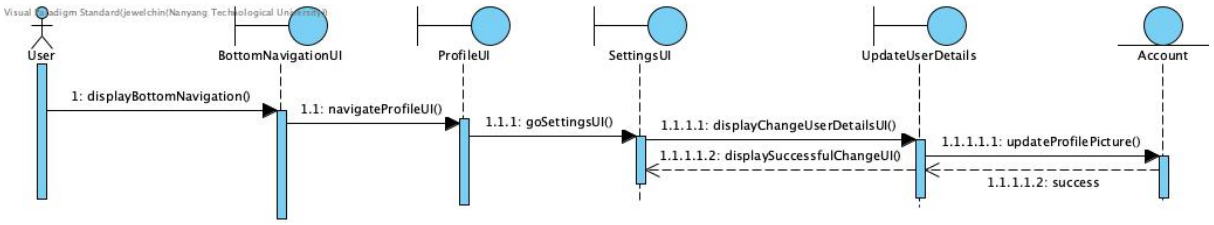
3. Account Registration with Checks



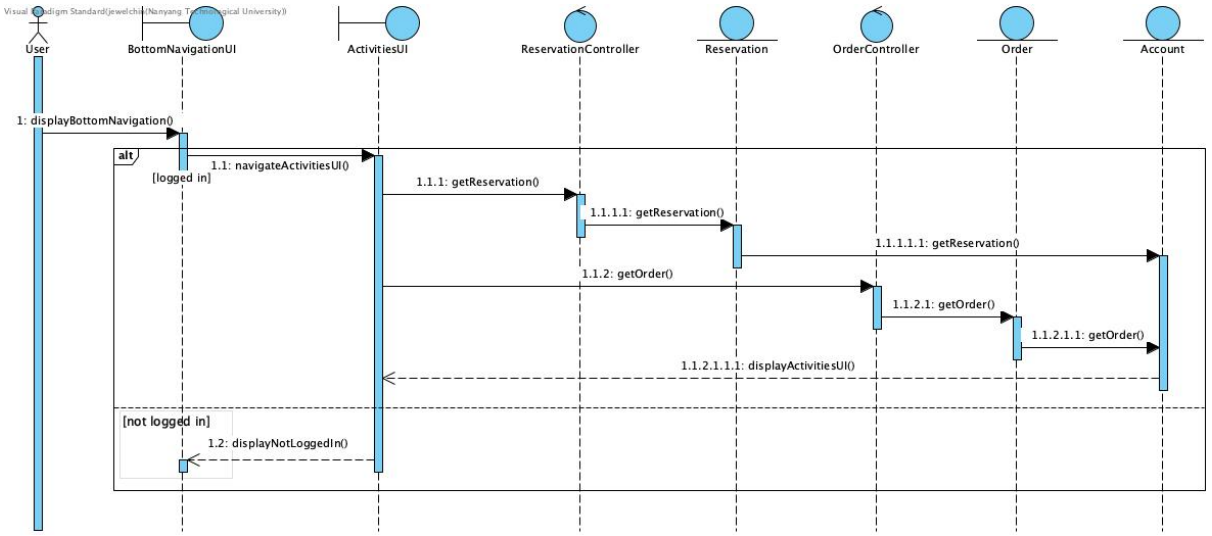
4. Account Login with Checks



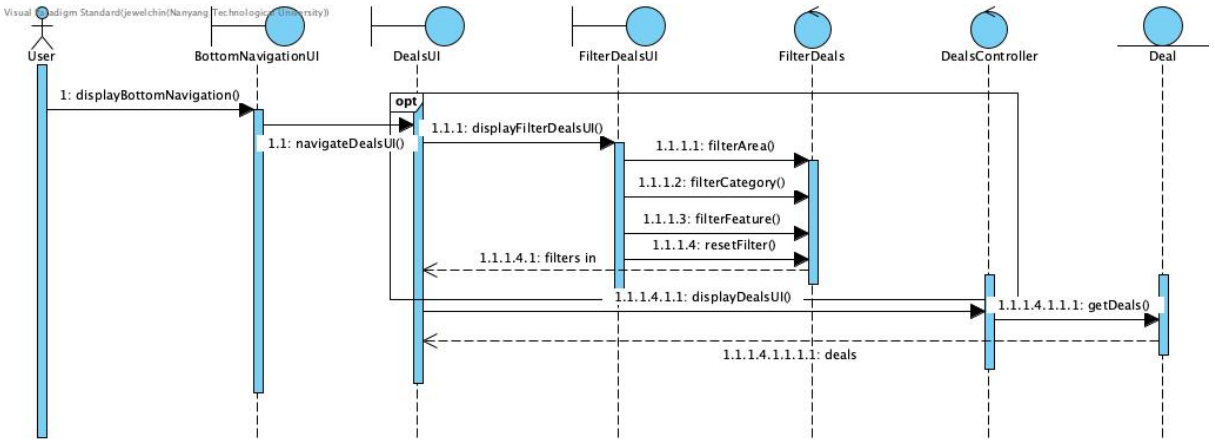
5. Change Profile Photo



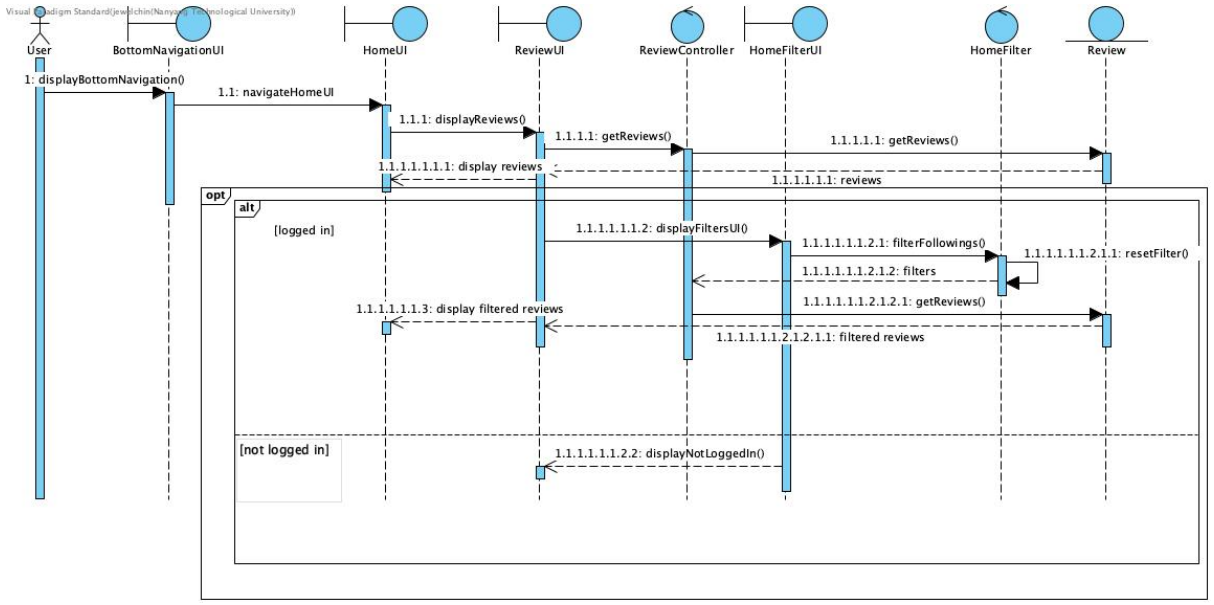
6. View Activities



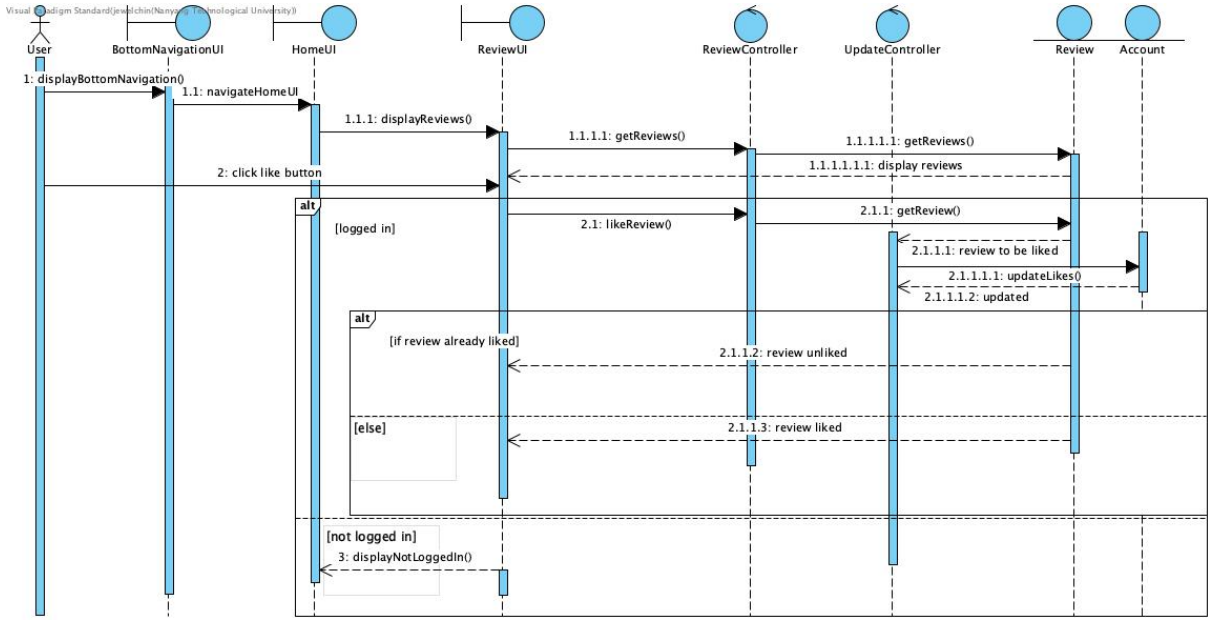
7. Display and Filter Deals



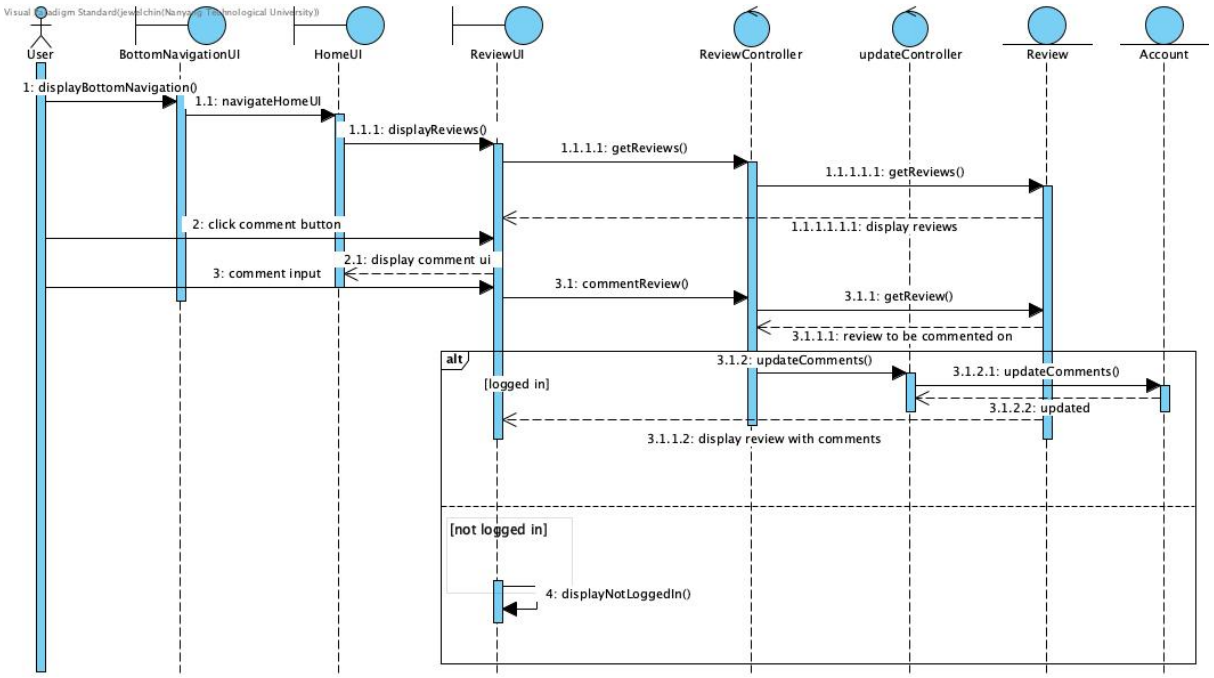
8. View and Filter Reviews



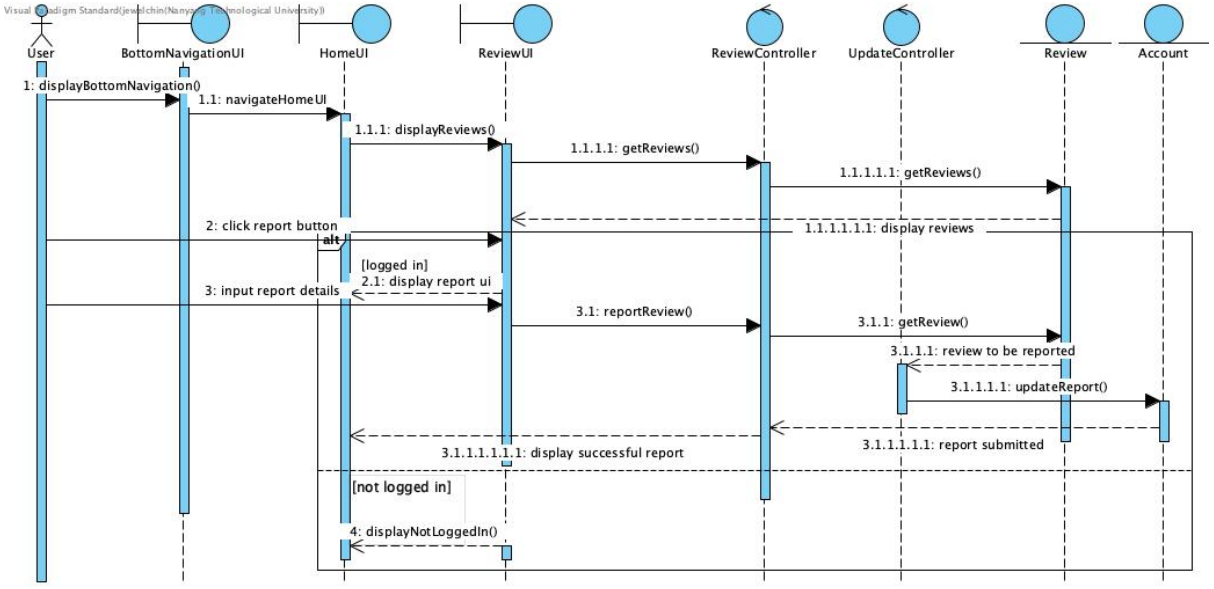
9. Like Reviews



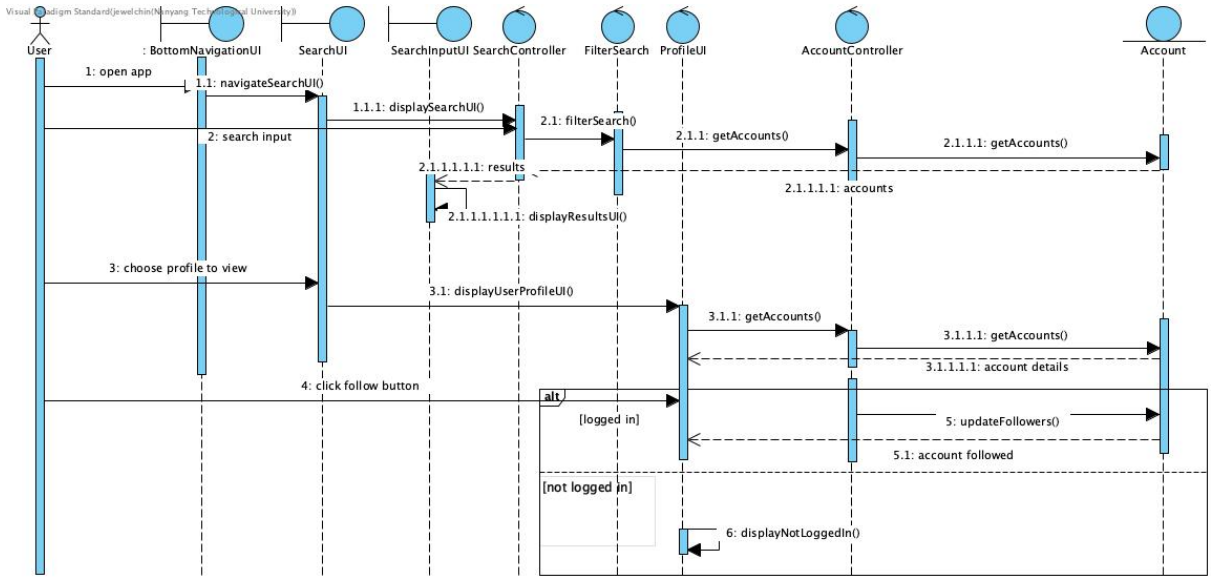
10. Comment Reviews



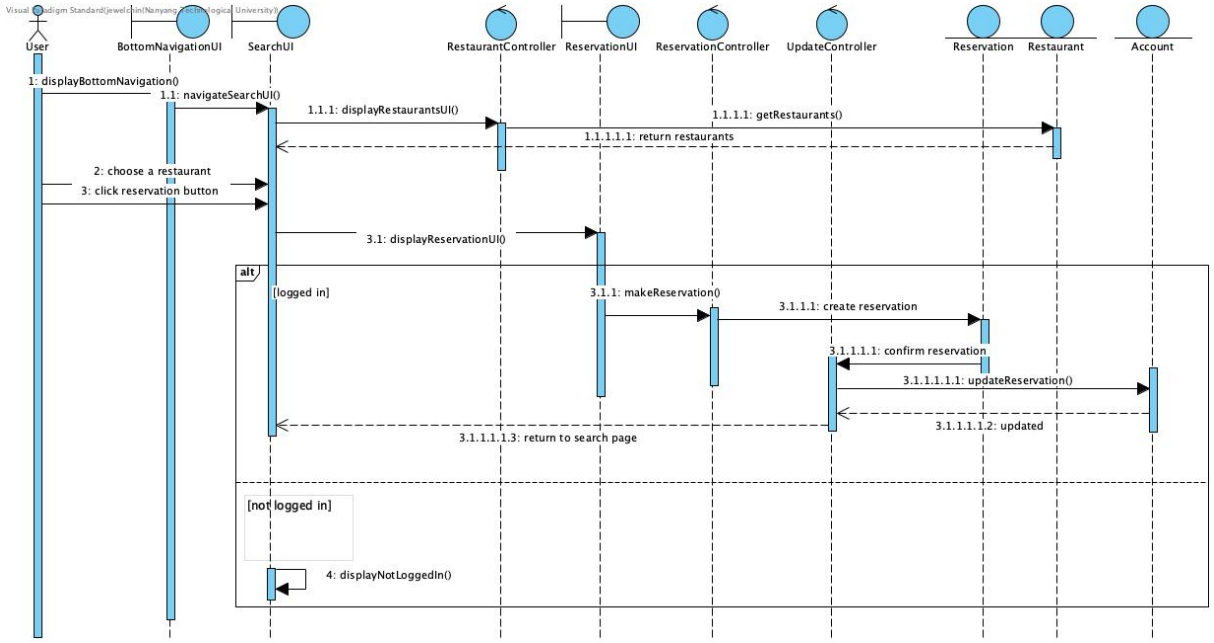
11. Report Reviews



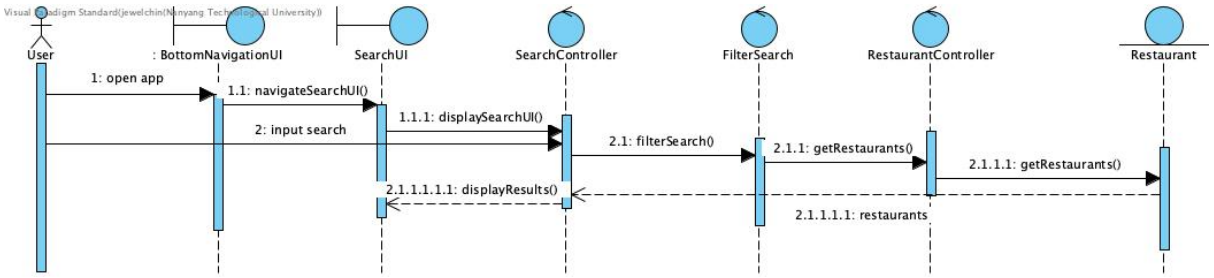
12. Search and Follow User



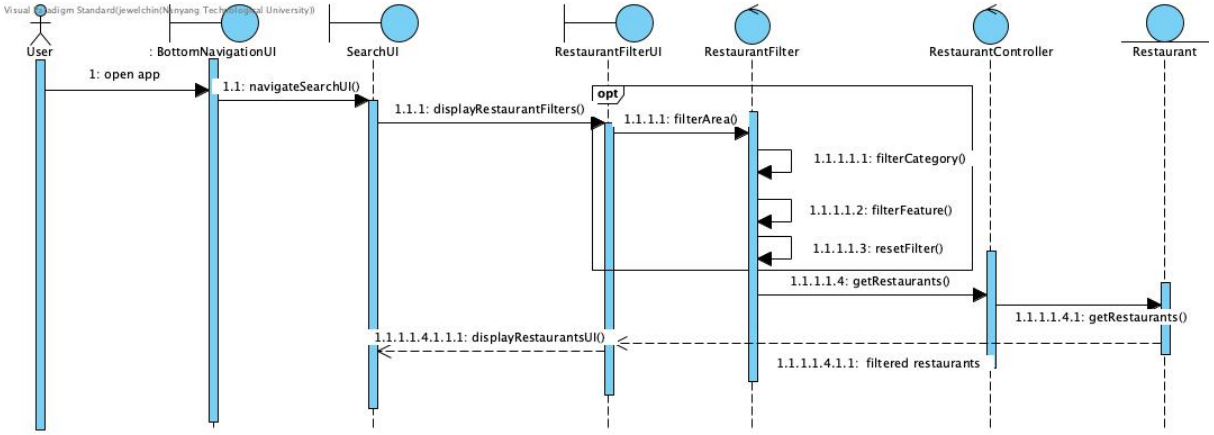
13. Booking Reservation



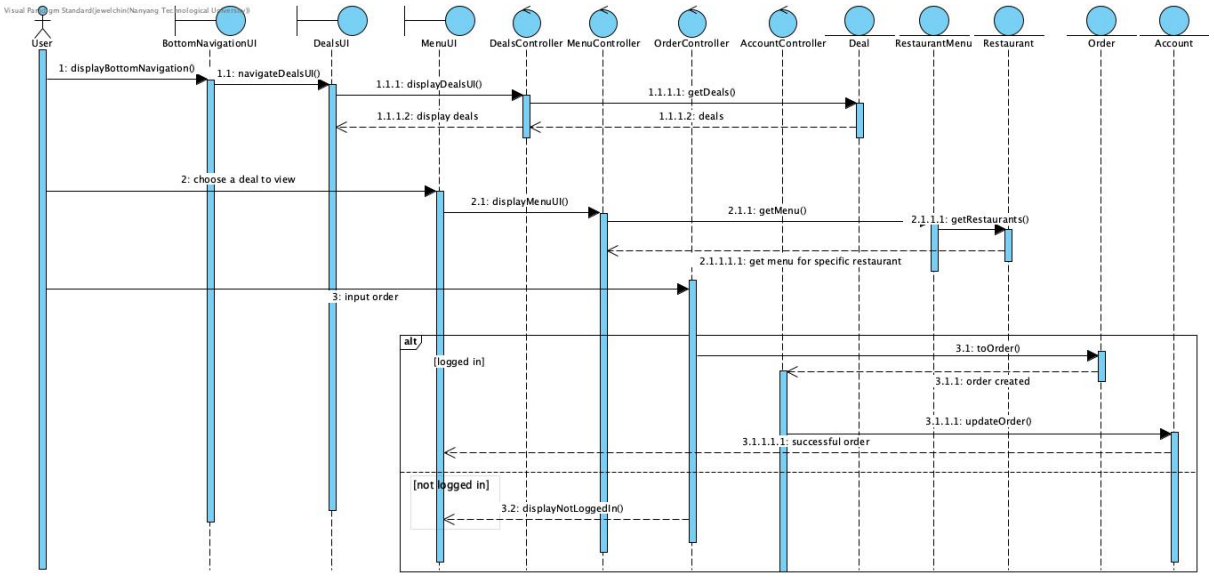
14. Search for Restaurant



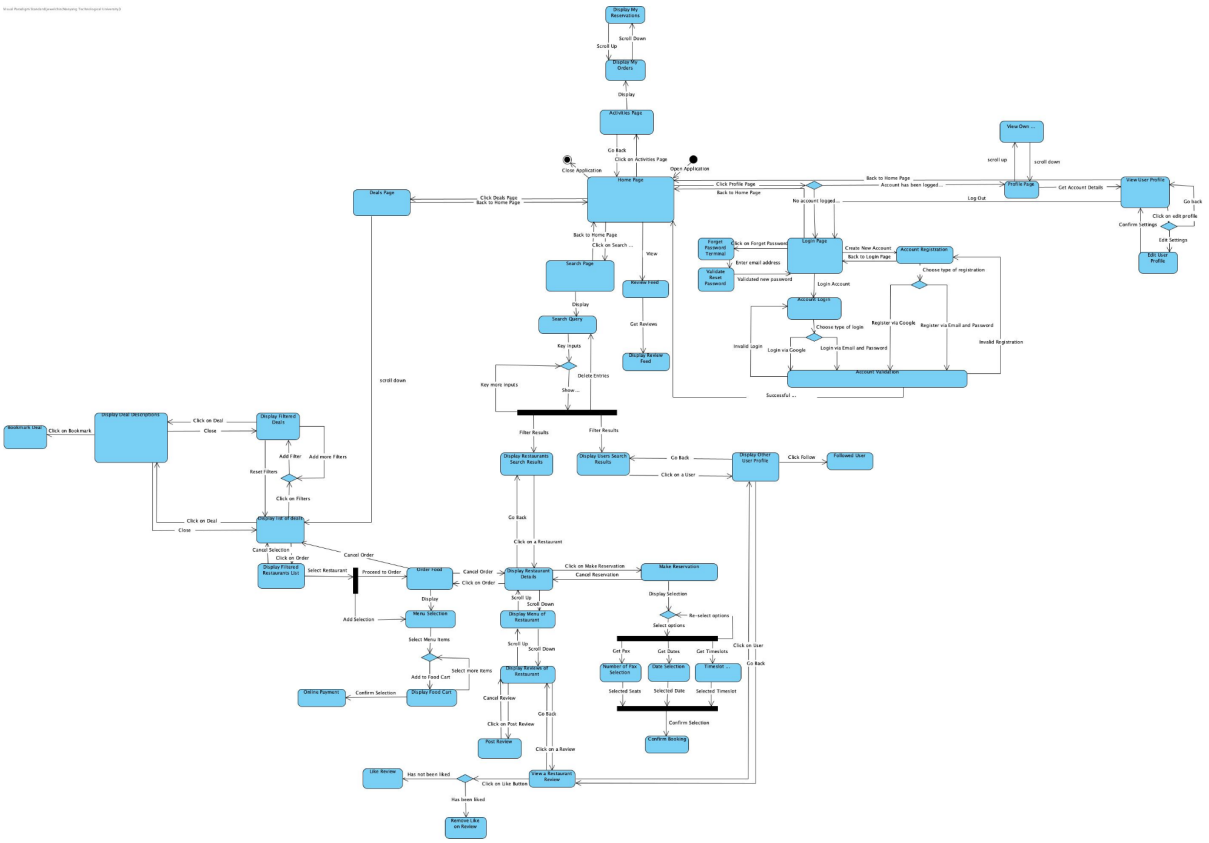
15. Filter Restaurants



16. Order Food through Deals Page



2.5. Dialog Map



3. Non-Functional Requirements

1. Performance Requirements

1.1. Deals Extraction System

1.1.1. Update Function - The system must extract deals from its relevant source, and update the database within 10 seconds, whenever there is new data.

1.2. Application

1.2.1. Display Deals Function - The application must display all current deals within 3 seconds.

1.2.2. Filter Function - The application must filter deals, based on the user's selection of category, within 3 seconds.

1.2.3. Like Function - The application must change the colour of the 'like' icon from default to red when the user clicks on it, within a second.

1.2.4. Book Reservation Function - When the booking is confirmed, the details must be present in the Activities page, within 3 seconds.

1.2.5. The application must not crash when the user opens it.

1.2.6. The application must be able to be maintained with little downtime.

2. Security Requirements

2.1. Data Extraction System

2.1.1. The system must not extract any data that may breach internet privacy. This includes usernames, profile pictures, comments, and private links.

2.2. Application

2.2.1. The application must only be able to access the data in the data extraction system. This is to prevent any potential leakages of private data.

2.2.2. The application will mask the password field in order to prevent any potential shoulder surfing.

3. Software Quality Attributes

3.1. Adaptability

3.1.1. The application should display the correct information based on the user's requests.

3.2. Flexibility

3.2.1. The application should be able to handle bugs and crashes, rebooting to a previous saved state until the system is operating again.

3.3. Interoperability

3.3.1 The application and the data extraction system must be able to exchange information needed for the application to function.

3.4. Maintainability

3.4.1. System and Application must have all infrastructure in code to simplify changes to provisioning and deployment processes.

3.5. Portability

3.5.1 The application should be compatible with all Android systems running Android Lollipop (API level 21).

3.6. Reliability

3.6.1. The data extraction system must be accurate and kept up to date with all information available in its relevant source.

3.6.2. The application must not take longer than 10 seconds to load.

3.7. Reusability

3.7.1. The code library used for coding the application should be generic enough to be easily used in other systems with minimal changes to be reused.

3.8. Robustness

3.8.1. The application should be able to cope with errors during execution.

3.8.2. Previous states must be saved to reboot when faced with errors with execution.

3.8.3. The application must have test cases to deal with erroneous input from the user.

3.9. Testability

3.9.1. The system should be easy to test and find bugs by having 98% unit test coverage for backend and frontend code.

3.10. Usability

3.10.1. 60% of users must find the information on deals displayed within the application to be concise and easy to read.

3.10.2. 60% of users must find the searching algorithm to be easy to use to find their desired restaurants or users.

3.10.3. The application must have a “Frequently Asked Question (FAQ)” section to guide users in using it.

3.10.4. 60% of users must find the FAQ section to be comprehensive and easily accessible.

4. Interface Requirements

4.1 User

EatNow! is suitable for all users with a device who is looking to eat smart and spend wisely on food by making use of food deals displayed by our application. However, EatNow! doesn't support non-english users.

4.2 Hardware

EatNow! requires a minimum of 1 gigabyte of RAM on the user's hardware device to function.

EatNow! requires hardware devices to have internet access when running.

4.3 Software

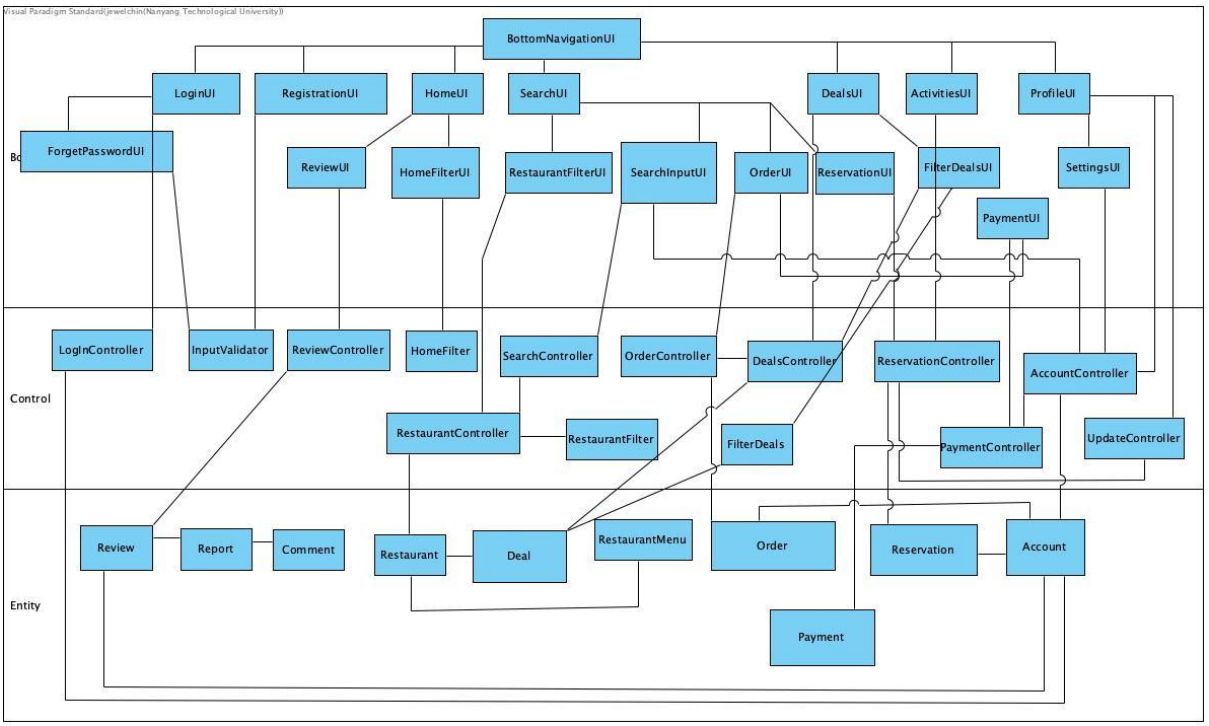
EatNow! is being designed to work on Android devices.
EatNow! uses external parties like Telethon API and GoogleMaps API to retrieve data for EatNow!'s database.
EatNow! uses RealTime Firebase to store data.

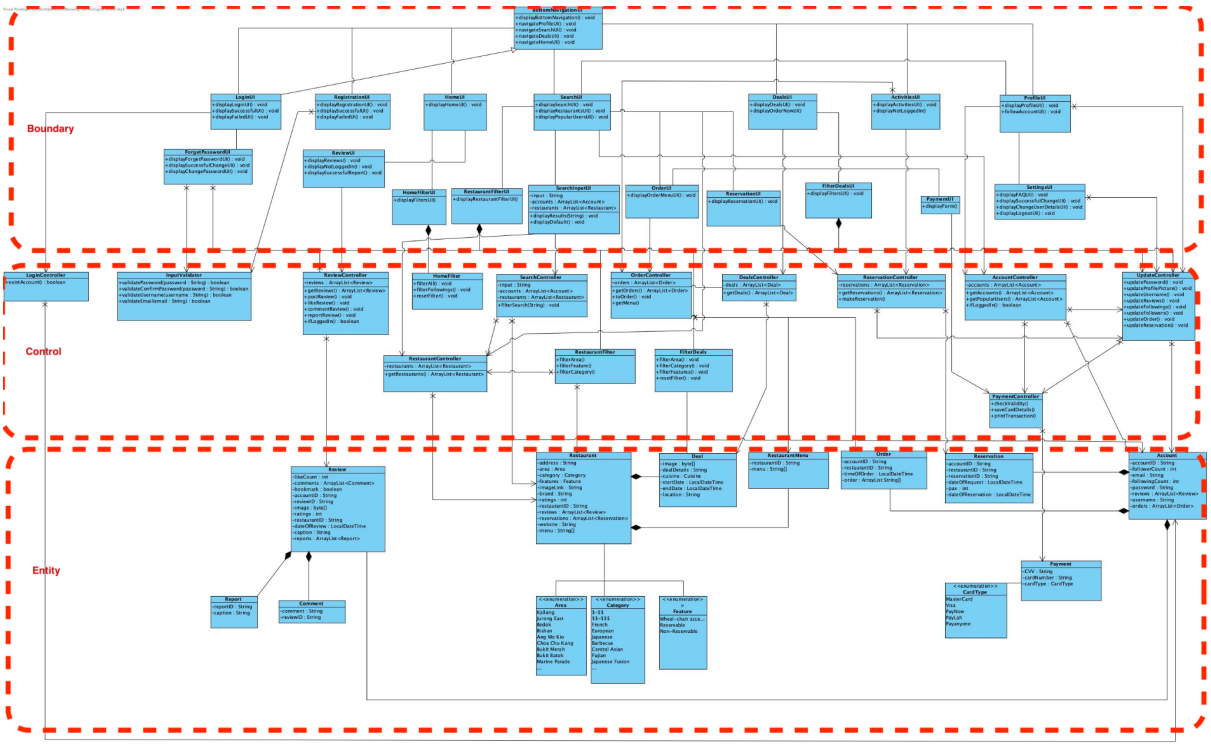
4.4 Communication

EatNow! requires the user's mobile device to be connected to an internet connection (Wifi/Cellular Data). All features, such as orders, reservations, and interaction with reviews and other users will be accessible through the application.
EatNow! allows users to register with any email.

5. Architecture Design

5.1 System Architecture Diagram



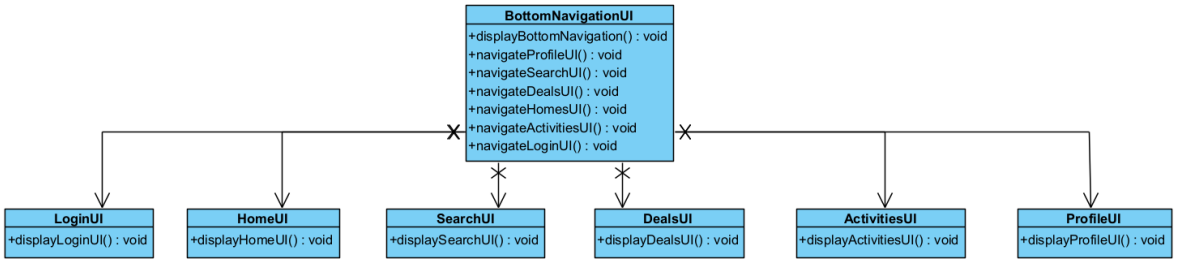


5.2 Design Patterns

1. Bottom Navigation UI Factory Design Pattern

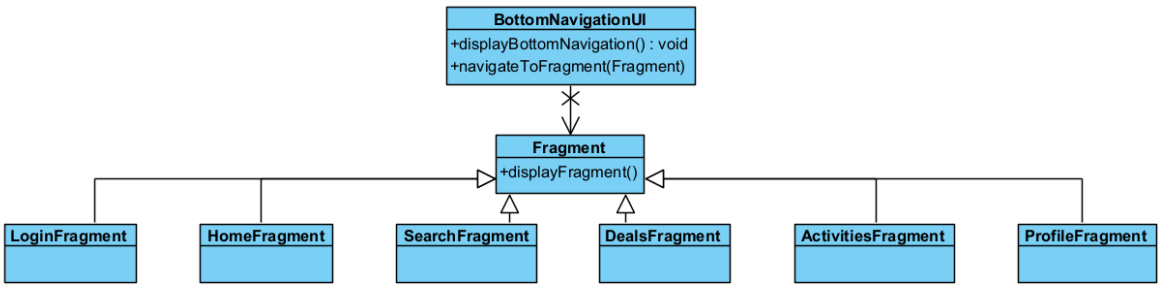
Problem

The code logic for displaying the user's interface is coupled in the Bottom Navigation UI. This makes it difficult to extend, replace or add any new potential interfaces into our application. Considering that the interface display is dependent on the user's input at runtime, the Factory Design Pattern is better suited to decouple the application's interface selection.



Solution

We defined a parent class (Fragment) for the UI subclasses, namely HomeFragment, SearchFragment, DealsFragment, ActivitiesFragment, LoginFragment, and ProfileFragment. The logic for instantiating these UI objects based on the user's input is implemented in the Bottom Navigation UI, while the navigation method (navigateToFragment) will be modified to receive arguments of objects of the Fragment parent class.



This allows greater flexibility in controlling the type of UI objects created, and feasibility in changing the creation logic, so that UI subclasses can be easily replaced or added.

This is also a representation of the Open-Closed Principle.

2. RecyclerView View Holder Pattern

Problem

During the scrolling of entities in our application, findViewById will be called for each creation of view, which will slow down the performance of our application. This is because the application will have to locate the view elements to update the view. This operation is expensive.

Solution

Our application utilises the RecyclerView view model to display the entities. Each RecyclerView view model consists of ViewHolders, which can be subclassed to adapt to our entities' interface.

In the example of displaying deals, a class (DealsViewHolder) is implemented to hold references of the necessary subviews to display the entity (deal).

The ViewHolder object will be constructed to instantiate the view objects.

```
// ViewHolder class for deals
public class DealsViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {

    // deal's image
    ImageView imageView;
    // deal's brand
    TextView textView;

    public DealsViewHolder(@NonNull View itemView) {
        super(itemView);

        imageView = itemView.findViewById(R.id.deals_image);
        textView = itemView.findViewById(R.id.deals_text);

        // To set image as clickable
        imageView.setOnClickListener(this);
    }
```

Thus, during the scrolling of RecyclerView, the ViewHolder object is accessed, instead of calling findViewById each time.

This will increase the performance of displaying our entities in our application.

3. RecyclerView Adapter Structural Pattern

Problem

Our application requires 3 different ViewHolder classes for the layouts of displaying the reviews. The implementation of 3 separate classes to adapt each ViewHolder distinct object to the RecyclerView may not be efficient, as each class holds the same attributes, including the arraylist of reviews to be displayed.

```
// ViewHolder class for reviews in Home Page
public class HomeReviewsViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
    // UIs
    ImageView reviewImage;
    ImageView accountImage;
    TextView usernameText;
    TextView likesText;
    TextView dateText;
```

```
// ViewHolder class for reviews in Restaurant Page
public class RestaurantReviewsViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener

    // UIs
    ImageView reviewImage;
    ImageView accountImage;
    TextView usernameText;
    TextView likesText;
    TextView dateText;
    TextView reviewCaption;
    RatingBar ratingBar;
```

```
// ViewHolder class for reviews in Account Page
public class AccountReviewsViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {

    // UIs
    ImageView reviewImage;
    TextView likesText;
    TextView reviewCaption;
    RatingBar ratingBar;
```

Solution

We defined an adapter class for our reviews (ReviewsAdapter) to map the RecyclerView to each ViewHolder.

```
public class ReviewsAdapter extends RecyclerView.Adapter implements ErrorPopUp {

    // Context of activity to display reviews
    Context context;
    // Layout Type 1 : Home Feed, 2 : Restaurant Review, 3 : Profile Review
    int layoutType;
    // List of reviews
    ArrayList<Review> reviews;
```

This is accomplished by subclassing the ReviewsAdapter under the RecyclerView Adapter, and overriding the onCreateViewHolder and onBindViewHolder methods to specify the type of ViewHolder to be utilised. These methods will bind the view to be displayed to the RecyclerView.

```
// Based on view type / layout type, setup layout and view holders
@NonNull
@Override
public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    if (viewType == 1) {
        // If Home
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.reviews_home_item, pare
        return new HomeReviewsViewHolder(view);
    } else if (viewType == 2) {
        // If Restaurant
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.reviews_restaurant_item
        return new RestaurantReviewsViewHolder(view);
    } else {
        // If Account
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.reviews_user_item, pare
        return new AccountReviewsViewHolder(view);
    }
}
```

```java
// Setting view holder parameters based on view holder type
@Override
public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {
    // Get review based on position
    Review review = reviews.get(position);
    // If there is no review
    if (review == null) {
        return;
    }
    // Retrieving account data based on account id (if needed)
    int account_id;
    Account selectedAccount;
    // If Home
    if (holder instanceof HomeReviewsViewHolder) {
        account_id = review.getAccount_id();
        selectedAccount = accountMap.get(account_id);
        // Set ImageView based on review's image
        if (review.getImage() != null) {
            try {
                Bitmap bmp = ImageConverter.convertASCIItoBitmap(review.getImage());
                ((HomeReviewsViewHolder) holder).reviewImage.setImageBitmap(Bitmap.createScaledBitmap
            } catch (Exception e) {
            }
        }
```

This Adapter Structural Pattern allows flexibility to change the review's ViewHolder based on the type of layout the application requires, as well as reusability of code.

6. Data Dictionary

Term	Definition
User	User refers to a person who uses the application.
Data System	Data system refers to a system that retrieves, updates and provides details about food deals, account information, and any other information linked to a user's account (e.g. Food preferences, reviews, and history)
ID	ID refers to the identification number that uniquely recognises each entity (e.g. Account) as separate objects.
Application	Application refers to the EatNow mobile application.
Deals	Deals are discounts, given by food and beverage (f&b) establishments.
Image/Photo	Image refers to a picture of a food deal, stored as ASCII Character Bit String.
Message	Message refers to a string of characters, and emojis that depict the details of the food deal.
Time	Time is recorded as Singapore Standard Time (SST), which is GMT +8.
Location	Location refers to a place where a person can redeem a food deal.
Timeframe	Timeframe refers to a food deal's start to end date.
Registration	Registration is a feature that allows users to create an account to bookmark their favourite deals, view their history, post a review, leave a rating, and share their reviews.
Account	A user's account will have an email address, username, password, follower count, following count and the account id.
Category	It includes price rate in the format $/$$/$$$, type of cuisine and type of restaurant.
Area	Area is a general term to group several places together. For example, there are places like City Square mall, Kallang Wave mall, and Kallang Leisure Park that will be categorised under the area of Kallang.
Features	A restaurant can be reservable or non-reservable. It can have other features like wheelchair-accessible, types of car parks available nearby, availability of reservations booking, free wifi, types of payment and many more.
Display	Display is a feature that shows the list of available food deals. It

	will show the photo of the deal, and summarised description of the deal.
Filtering	Filtering is a feature that allows users to filter the list of food deals according to category, area and features.
Type of restaurant	It includes fast food, bars, cafes and many more.
Popular Users	Popularity is judged based on the number of followers a user has.
Review	Review is a feature that allows users to comment on the food deals, and leave a rating.

7. Testing

7.1 Black Box Testing

---

1. Equivalence Class Testing

1.1. Register function

Equivalence Classes

Input Data	Valid	Invalid
Username	Any string input with at least 6 characters	Empty string or string with less than 6 characters or more than 20 characters
Email	Any valid email	Empty or emails with wrong format
Password	Any string input with at least 8 characters, at least 1 uppercase, lowercase character, digit and special character, no more than 20 characters	Empty string or any string less than 8, no uppercase or lowercase character or no digits or no special characters or more than 20 characters

Test cases:

Username	Email	Password	Expected Results	Actual Results
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123Abcd!”	Welcome fullyvelcro12	Welcome fullyvelcro12
“”	“fullyvelcro@gmail.com”	“123Abcd!”	Invalid Registration	Invalid Registration
“fullyv”	“fullyvelcro@gmail.com”	“123Abcd!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“”	“123Abcd!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro.com”	“123Abcd!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcrogmail.com”	“123Abcd!”	Invalid Registration	Invalid Registration

“fullyvelcro12”	“fullyvelcro@gmail.com “	“123Abcd!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123abcd!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro@gmail.com”	“”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro@gmail.com”	“Abcdefg!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123abcd!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123ABCD!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123Abcde”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro@gmail.com”	“1Ab!”	Invalid Registration	Invalid Registration
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123Aaaaaaaa aaaaaaaaa!”	Invalid Registration	Invalid Registration

1.2. Account Log in

Equivalence Classes

Input Data	Valid Input	Invalid Input
Email	Any string that is in database	Empty string or string not within database
Password	Associating string that corresponds with Username string in database	Empty string or string that does not correspond with Username in database

Test cases:

Username	Password	Expected Output	Actual Output
“fullyvelcro@gmail.com”	“123Abcd!”	Welcome fullyvelcro12	Welcome fullyvelcro12
“”	“123Abcd!”	Invalid Login	Invalid Login
“fullyvelcro@gmail.com”	“”	Invalid Login	Invalid Login

1.3. Review Creation

Equivalence Classes

Input Data	Valid Input	Invalid Input
Rating	Rating from 0.5-5	Empty rating
Caption	Any string input	Empty string
Comment	Any string input	Empty string

Test Cases:

Rating	Caption	Comment	Expected Output	Actual Output
“5”	“Nice”	“Damn good!”	Review created	Review created
“”	“Nice”	“Damn good!”	No rating given	No rating given
“5”	“”	“Damn good!”	No caption given	No caption given
“5”	“Nice”	“”	No comment given	No comment given

2. Boundary Value Testing

2.1. Username Character Length

Username	Email	Password	Username Length	Expected Output	Actual Output

“fullyvelcro12”	“fullyvelcro@gmail.com”	“123Abcd!”	13	Registration Successful	Registration Successful
“fullyvelcro121212121”	“fullyvelcro@gmail.com”	“123Abcd!”	20	Registration Successful	Registration Successful
“fully”	“fullyvelcro@gmail.com”	“123Abcd!”	5	Registration Unsuccessful	Registration Unsuccessful
“fullyvelcro1212121212”	“fullyvelcro@gmail.com”	“123Abcd!”	21	Registration Unsuccessful	Registration Unsuccessful

Valid Boundary Values {6,20}
Invalid Boundary Values {5,21}

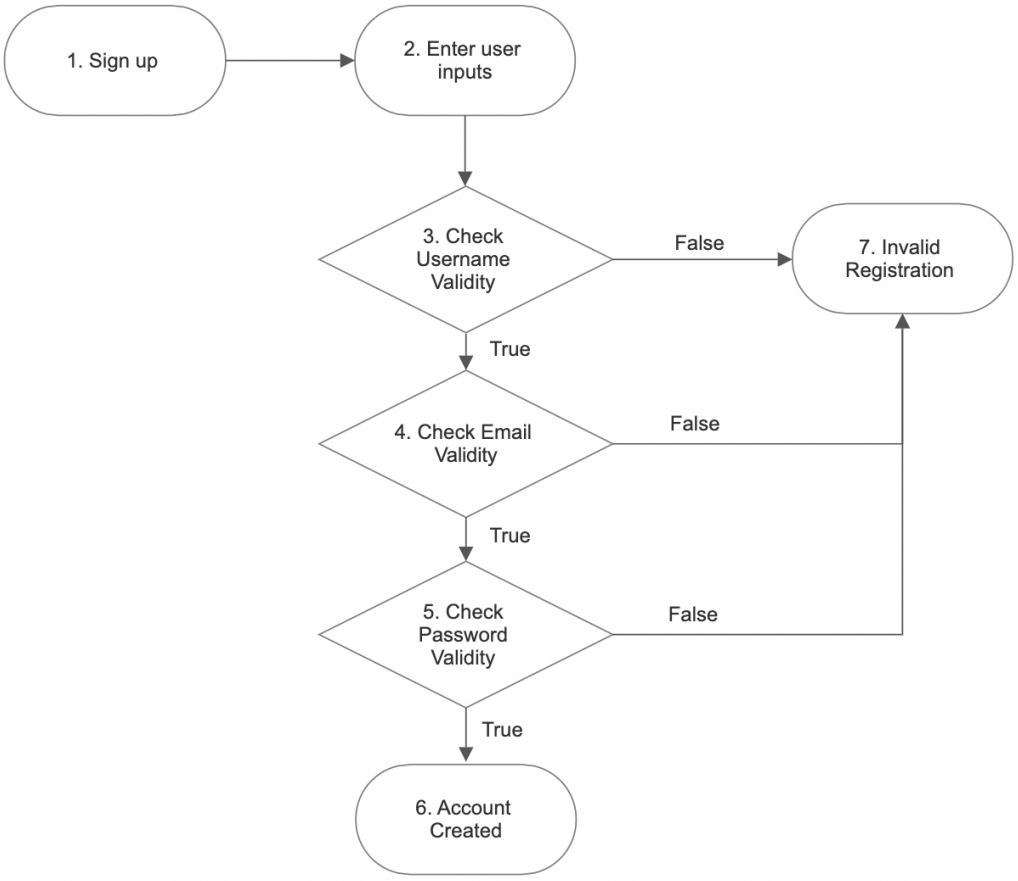
2.2. Password Character Length

Username	Email	Password	Password Length	Expected Output	Actual Output
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123Abcd!”	8	Registration Successful	Registration Successful
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123Aaaaaaaaaaaaaaaaa!”	20	Registration Successful	Registration Successful
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123Abcd”	7	Registration Unsuccessful	Registration Unsuccessful
“fullyvelcro12”	“fullyvelcro@gmail.com”	“123Aaaaaaaaaaaaaaaaaa!”	21	Registration Unsuccessful	Registration Unsuccessful

Valid Boundary Values: {8, 20}
Invalid Boundary Values: {7, 21}

7.2 White Box Testing

1. Account Creation



Basis Path

a) {1,2,3,4,5,6}
b) {1,2,3,7}
c) {1,2,3,4,7}
d) {1,2,3,4,5,7}

Basis Path a)
This scenario occurs as the main flow of a typical successful account creation. The user inputs a valid username, password and email.

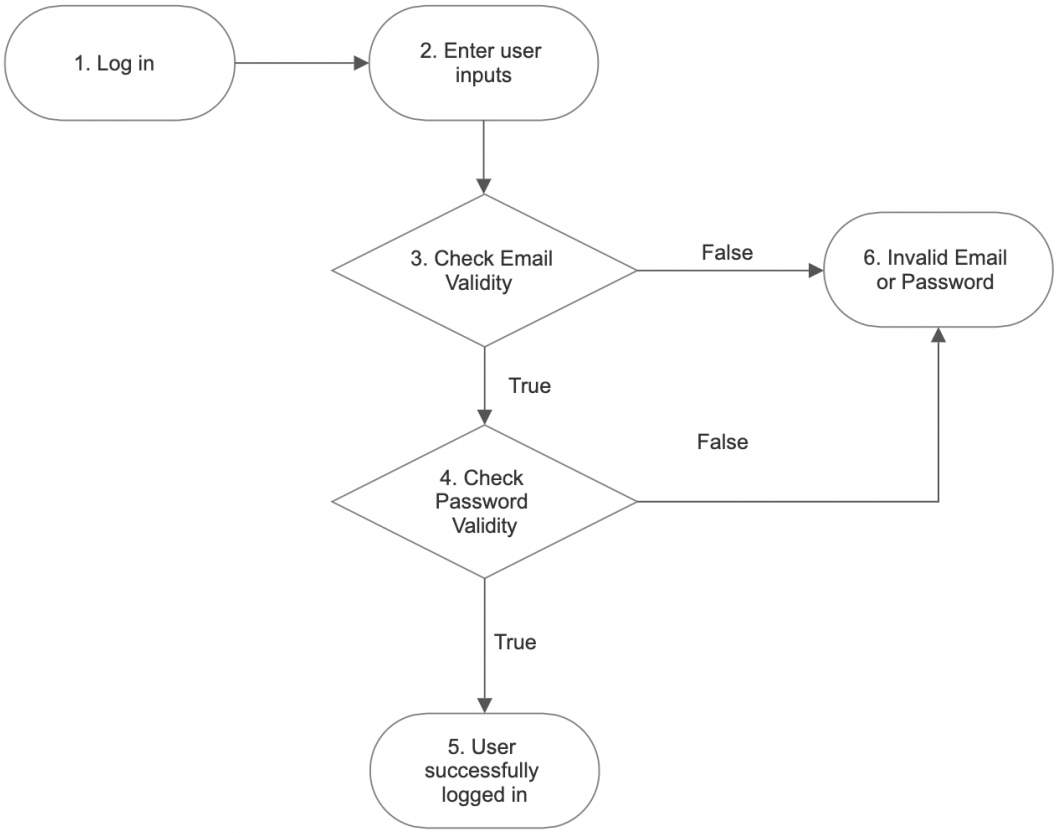
Basis Path b)
This scenario occurs when the user inputs an invalid username during account registration.

Basis Path c)
This scenario occurs when the user inputs a valid username but an invalid email address during account registration.

Basis Path d)
This scenario occurs when the user inputs a valid username and email but an invalid password during account registration.

Cyclomatic Complexity
Since all decision points are binary: Cyclomatic complexity = |3| + 1 = 4

2. Account Log In



Basis Path
a) {1,2,3,4,5}
b) {1,2,3,6}
c) {1,2,3,4,6}

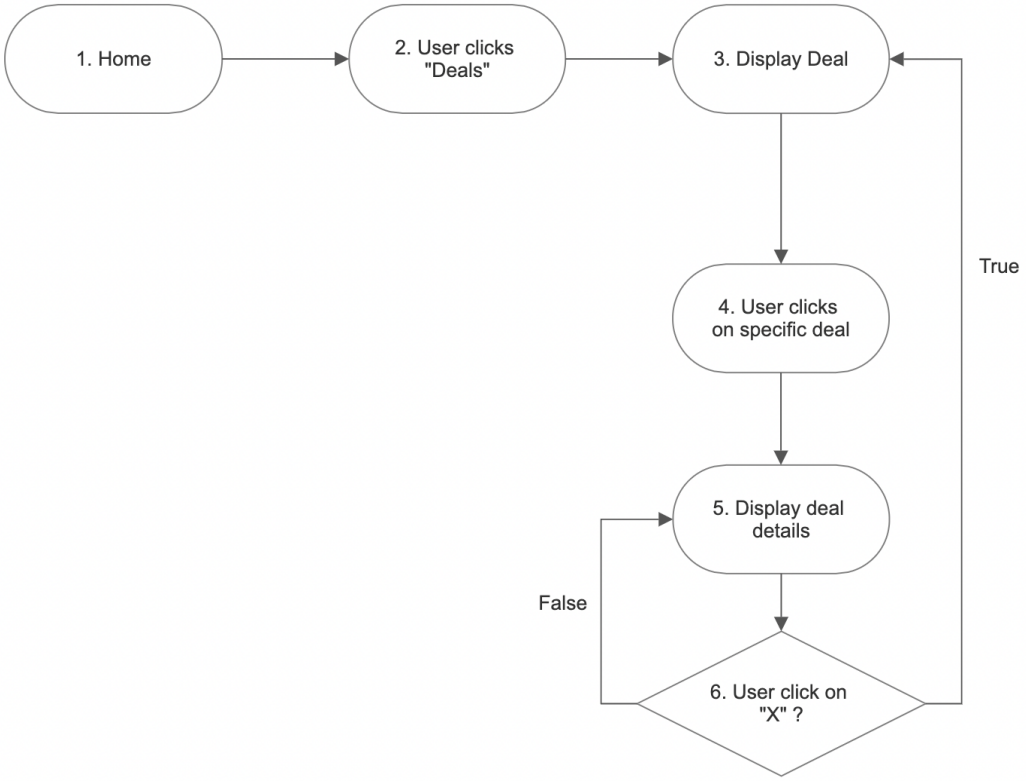
Basis Path a)
This scenario occurs as the main flow of a successful account login. This happens when the user enters a valid username and password.

Basis Path b)
This scenario occurs when the user inputs an invalid username during account login.

Basis Path c)
This scenario occurs when the user inputs a valid username but an invalid password during account login.

Cyclomatic Complexity
Since all the decision points are binary: Cyclomatic complexity = |2| + 1 = 3

3. Deals Click



Basis Path
a) {1,2,3,4,5,6}
b) {1,2,3,4,5}

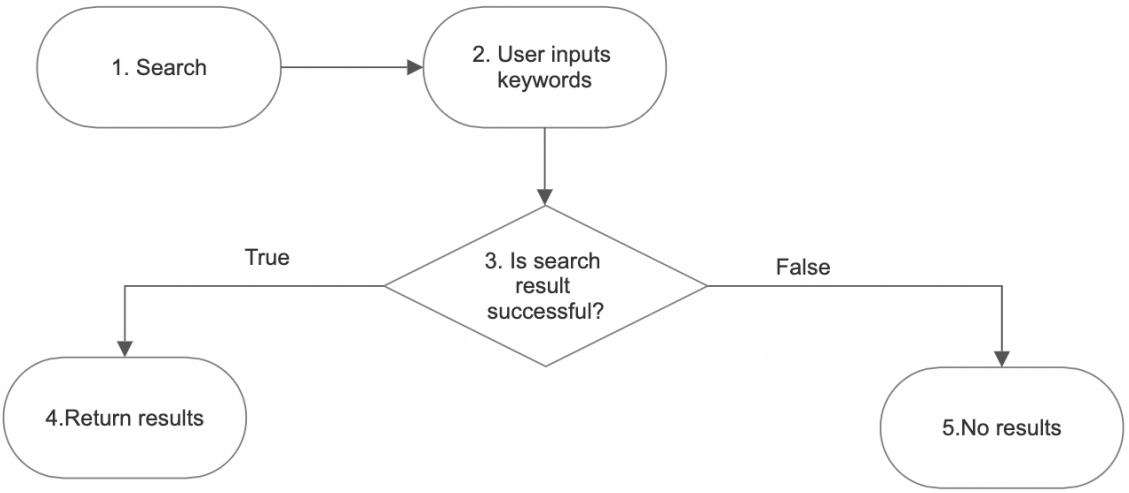
Basis Path a)
This scenario is the typical flow where the user views the details available, clicking on a specific deal to view more details before closing it and viewing more deals. This checks the functionality of the deals page in displaying deals and its specific details.

Basis Path b)
This scenario is when the user views the deals available and clicks on the specific deals to view more details. However, the user does not exit the pop up.

Cyclomatic Complexity

Since all the decision points are binary: Cyclomatic complexity = |1| + 1 = 2

4. Search Restaurants



Basis Path

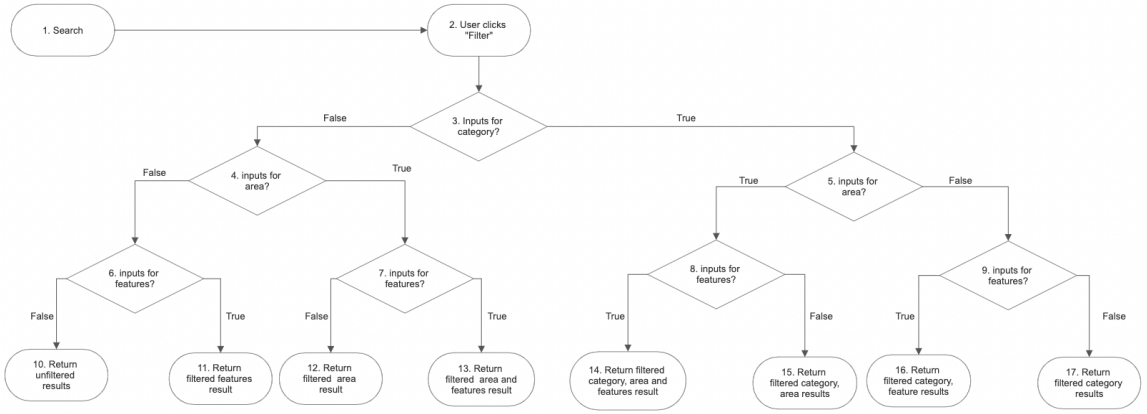
a) {1,2,3,4}
b) {1,2,3,5}

Basis Path a)
This is the typical scenario where the user inputs keywords of the restaurants into the search bar and the system displays the results of corresponding restaurants with the keywords

Basis Path b)
This is the scenario where the user inputs keywords of the restaurants into the search bar; however, no restaurants with the corresponding keywords are found. Displaying no results.

Cyclomatic Complexity
Since all decision points are binary: Cyclomatic complexity = |1| + 1 = 2

5. Filtering Restaurants



Basis Paths

a) {1,2,3,5,8,14}
b) {1,2,3,5,8,15}
c) {1,2,3,5,9,16}
d) {1,2,3,5,9,17}
e) {1,2,3,4,7,13}
f) {1,2,3,4,7,12}
g) {1,2,3,4,6,11}
h) {1,2,3,4,6,10}

Basis Path a)
This is the typical scenario where the user filters the list of restaurants by category, area and features.

Basis Path b)
This is the scenario where the user filters the list of restaurants by category and area.

Basis Path c)
This is the scenario where the user filtered the list of restaurants by category and features.

Basis Path d)
This is the scenario where the user filtered the list of restaurants by only category.

Basis Path e)
This is the scenario where the user filtered the list of restaurants by area and features.

Basis Path f)
This is the scenario where the user filtered the list of restaurants by only area.

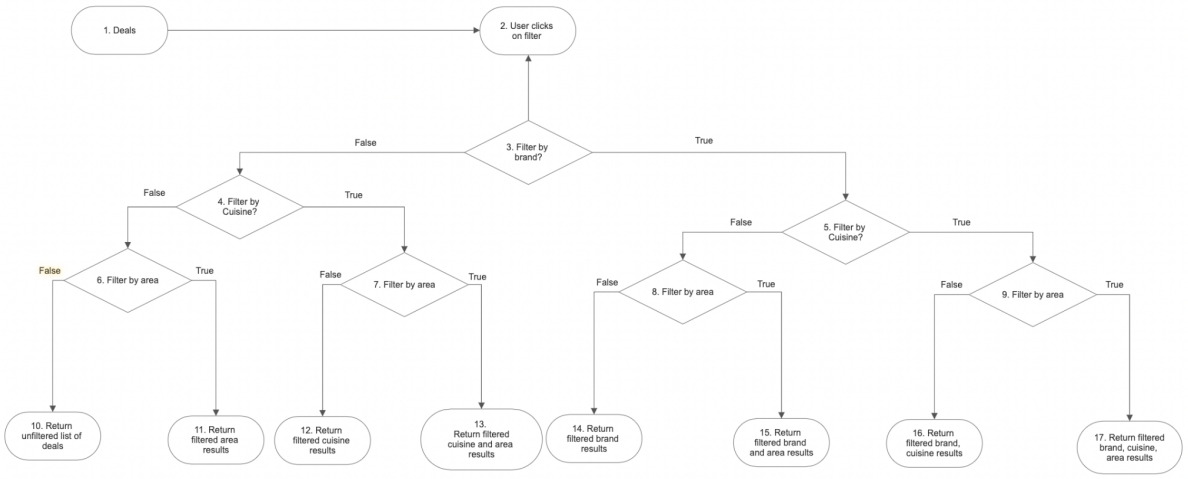
Basis Path g)

This is the scenario where the user filtered the list of restaurants by only features.

Basis Path h)
This is the scenario where the user does not filter the list of restaurants.

Cyclomatic Complexity
Since all decision points are binary: Cyclomatic complexity = |7| + 1 = 8

6. Filtering Deals



Basis Paths

a) {1,2,3,5,9,17}
b) {1,2,3,5,9,16}
c) {1,2,3,5,8,15}
d) {1,2,3,5,8,14}
e) {1,2,3,4,7,13}
f) {1,2,3,4,7,12}
g) {1,2,3,4,6,11}
h) {1,2,3,4,6,10}

Basis Path a)
This is the typical scenario where the user filters the deals by all 3 categories of brand, cuisine and location

Basis Path b)
This is the scenario where the user filters the deals by brand and cuisine only.

Basis Path c)
This is the scenario where the user filters the deals by brand and area only.

Basis Path d)
This is the scenario where the user filters the deals by brand only.

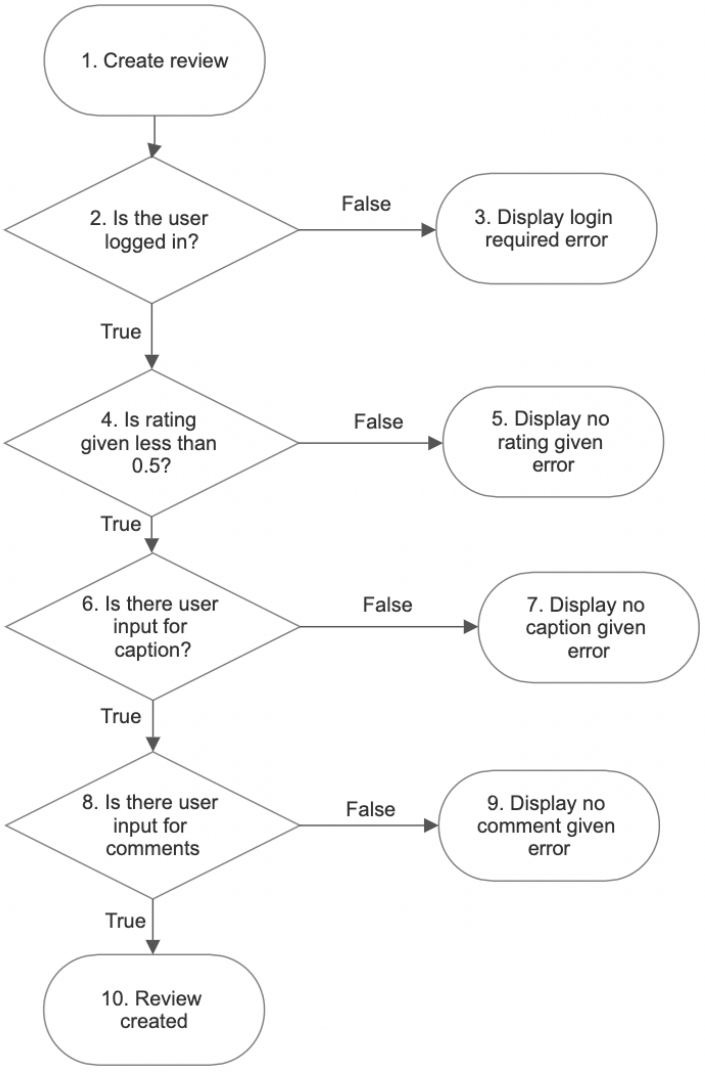
Basis Path e)
This is the scenario where the user filters the deals by cuisine and area only.
Basis Path f)
This is the scenario where the user filters the deals by cuisine only.

Basis Path g)
This is the scenario where the user filters the deals by area only.

Basis Path h)
This is the scenario where the user did not filter the deals at all.

Cyclomatic Complexity
Since all the decision points are binary: Cyclomatic complexity = |7| + 1 = 8

7. Create Review



Basis Paths
a) {1,2,4,6,8,10}
b) {1,2,3}
c) {1,2,4,5}

d) {1,2,4,6,7}
e) {1,2,4,6,8,9}

Basis Path a)
This is the typical scenario where the user enters a valid input for rating, caption and comment leading to a successful creation of a review

Basis Path b)
This is the scenario where the user is not logged in and is trying to create a review, it will prompt the user to log in first before creating a review.

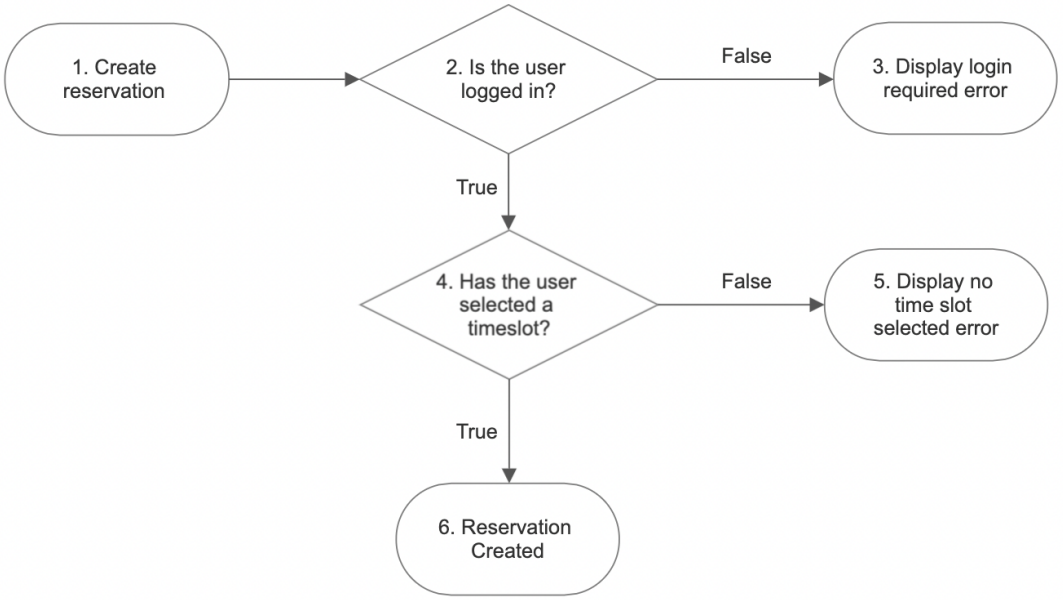
Basis Path c)
This is the scenario where the user enters an invalid input for rating, prompting an error in review creation.

Basis Path d)
This is the scenario where the user enters an invalid input for caption, prompting an error in review creation.

Basis path e)
This is the scenario where the user enters an invalid input for comments, prompting an error in review creation.

Cyclomatic Complexity
Since all the decision points are binary: Cyclomatic complexity = |4| + 1 = 5

8. Create Reservation



Basis Paths

a) {1,2,4,6}
b) {1,2,3}
c) {1,2,4,5}

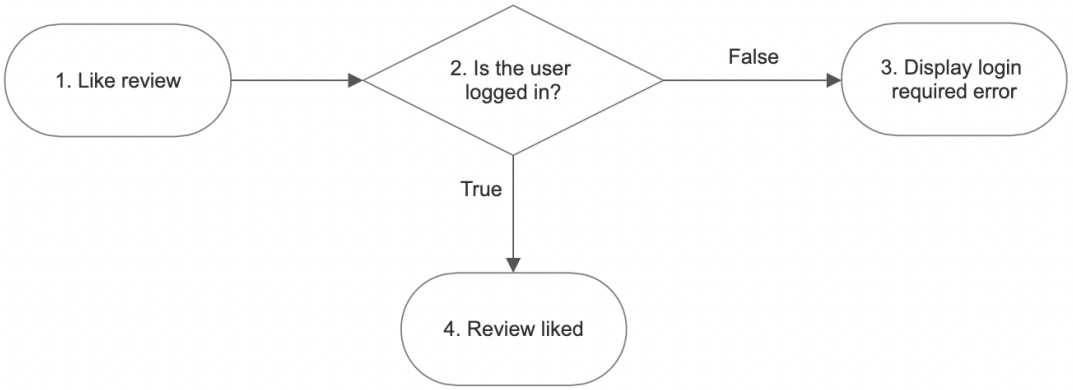
Basis Path a)
This is the typical scenario where the user is already logged in, the user selects a time slot and successfully creates a reservation.

Basis Path b)
This is a scenario where the user is trying to create a reservation without being logged in. It will display an error message prompting the user to log in.

Basis Path c)
This is a scenario where although the user is logged in, the user is trying to create a reservation without selecting a time slot. It will display an error message, prompting the user to select a time slot.

Cyclomatic Complexity
Since all decision points are binary: Cyclomatic complexity = |2| + 1 = 3

9. Liking Reviews



Basis Paths

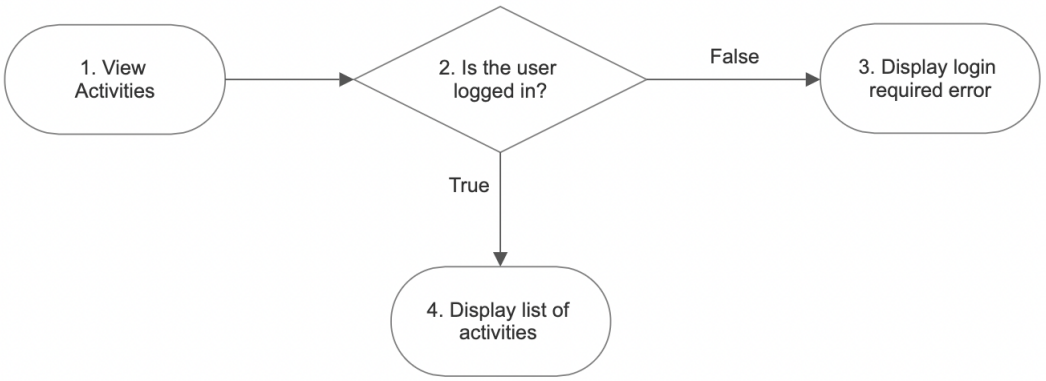
a) {1,2,4}
b) {1,2,3}

Basis Path a)
This is the typical scenario where the user is already logged in. When viewing a review, the user is able to like the review, increasing the number of likes by 1.

Basis Path b)
This is the scenario where the user is trying to like a review without being logged in. The system will display an error message, prompting the user to log in first before the review can be liked.

Cyclomatic Complexity
Since all decision points are binary: Cyclomatic complexity = |1| + 1 = 2

10. View Activities



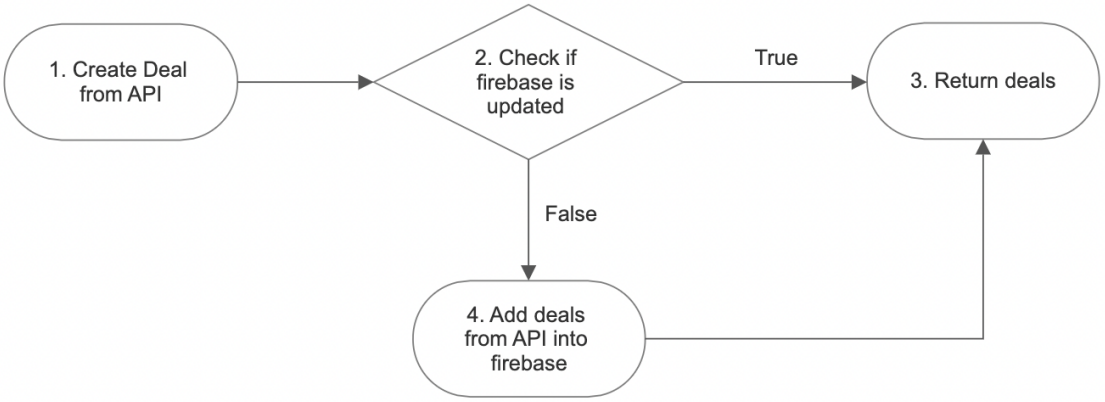
Basis Paths
a) {1,2,4}
b) {1,2,3}

Basis Path a)
This is the typical scenario where the user is already logged in. When trying to view the activities section, the user is able to access it and see the list of activities under the user.

Basis Path b)
This is the scenario where the user is trying to view activities under the user without being logged in. The system will display an error message, prompting the user to log in first before the user can see the activities.

Cyclomatic Complexity
Since all decision points are binary: Cyclomatic complexity = |1| + 1 = 2

11. Create Deals



Basis Paths

a) {1,2,4,3}
b) {1,2,3}

Basis Path a)

This is the typical scenario where the system constantly checks if it is updated. With a constant flow of new deals, the system will check if the firebase is updated, and add in the corresponding new deals into the system.

Basis Path b)

This is the scenario where the firebase is already updated and there are no new deals to add into the system.

Cyclomatic Complexity

Since all decision points are binary: Cyclomatic complexity = |1| + 1 = 2

8. Appendix

8.1 Appendix A : Backend Services Python Modules

The application utilises Python version 3.9 and above.

The details of the required Python modules for the project's deal extraction system are as follow:

Module	Version	Description
Telethon	1.25.2	Telethon API allows interaction between our Firebase database and our EatNow! Telegram channel to extract food deal messages from the channel. Please refer to the documentation: https://docs.telethon.dev/en/stable/
onemapsg	1.0.3	Onemapsg API provides a concrete list of planning areas in Singapore, as well as returning geolocation results of search queries. This facilitates the extraction of areas that the food deal messages are being held at. Please refer to the documentation: https://pypi.org/project/onemapsg/
googlemaps	4.6.0	Similar to onemapsg, Google Maps API returns geolocation results of search queries. This is utilised when onemapsg does not return any result. Please refer to the documentation: https://github.com/googlemaps/google-maps-services-python
firebase firebase_admin	3.0.1 6.0.1	Firebase API allows interaction to our project's firebase real time database to update and add deal messages into the cloud. Please refer to the documentation: https://firebase.google.com/docs/reference/admin/python

You may refer to the python modules of our deal extractor system in the 'DealExtractorSystem' file under the 'SourceCode' folder.

8.2 Appendix B : Demonstration of Application

For more information and detailed demonstration of the EatNow! application, please refer to 'EatNow! Application Demonstration.mp4' under DemoVideo folder.

8.3 Appendix C : Future Developments

Due to unavailability of data and time constraint, our application has yet to fully implement the ordering system.
Thus, as part of this project's future prospects, EatNow! hopes to partner with various food and beverages (f&b) restaurants to outsource its data from them, while maintaining its financial stability through potential investments and commissions.
EatNow! also hopes to shift its database to a robust and organised system, where retrieval and querying of data is fast and efficient when handling large amounts of data in our project.