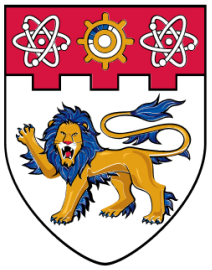


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2006 Software Engineering

Lab Group : Z60

Lab Supervisor : Lin GuoSheng

Lab Teaching Assistant : Zhang Zhao Qi

Date of Submission : 16/04/2023

Prepared By:

Name & Matriculation No.	Signature
Palaniswamy Tarun Kumar (U2121932F)	
Hashil Jugjivan (U2120599F)	
See Jian Rong Ben (U2120360L)	
Tham Holdon (U2121491G)	
Xu Junzhe (U2120174A)	
Weng Pei He (U2120920A)	

Table Of Contents

---

1. Software Requirements Specification

1.1 Introduction

1.1.1 Purpose

This document specifies the software requirements for "W.G.T" (We Got This), a web application that helps users find, review, and manage information about restaurants. This SRS covers the entire system, including all related components.

1.1.2 Document Conventions

This document follows standard typographical conventions, with code snippets and file paths in a monospaced font. Each requirement statement has its own priority, not inheriting from higher-level requirements.

1.1.3 Intended Audience and Reading Suggestions

This document is intended for project managers, developers, testers, documentation writers, marketing staff, and users. The document is organized into sections detailing requirements, system features, and other pertinent information.

1.1.4 Product Scope

"W.G.T" is a web application designed to help users search for restaurants, view and submit reviews, find nearest car parks, and manage their user accounts. The application aims to provide an easy-to-use interface for discovering new dining options and sharing experiences with others.

1.1.5 References

- Django Web Framework: https://www.djangoproject.com/
- Django Documentation: https://docs.djangoproject.com/en/
- Django Templates: https://docs.djangoproject.com/en/topics/templates/

1.2 Overall Description

1.2.1 Product Perspective

"W.G.T" is a self-contained web application designed to help users find and review restaurants. The application serves as a standalone product and is not a part of any larger system. A simple diagram showing major components and external interfaces is provided below:

User <---> W.G.T Web Application <---> Database

1.2.2 Product Functions

- Search for restaurants based on location, cuisine, or name
- View detailed restaurant information, including address, cuisines, and user reviews
- Find nearest carparks and get directions to each one
- Create user accounts and manage personal profiles
- Submit reviews and ratings for restaurants

1.2.3 User Classes and Characteristics

- Casual users: Infrequent users who search for restaurants, read reviews, and get directions, but do not contribute reviews or ratings

- Registered users: Regular users who have created accounts, contribute reviews, and manage their profiles
- Administrators: Users responsible for managing and maintaining the application, including user account, reviews management and restaurant content moderation

1.2.4 Operating Environment

The "W.G.T" web application will operate in a web server environment, supporting various web browsers and devices. The software will be developed using the Django web framework and is designed to be compatible with multiple operating systems, including Windows, macOS, and Linux. The software is also mobile-friendly and can automatically adjust frame sizes.

1.2.5 Design and Implementation Constraints

- The application must be developed using the Django web framework.
- The application must use a SQLite3 database for data storage.
- The application must be responsive and compatible with various web browsers and devices.
- The application must comply with relevant data privacy regulations and security best practices.

1.2.6 User Documentation

User documentation will include:

- A user manual describing the features and functionality of the application
- Online help and FAQs available within the application
- Video tutorials demonstrating how to use the application

1.2.7 Assumptions and Dependencies

- It is assumed that the restaurant data will be obtained from a reliable and regularly updated source (Grab restaurant database).
- It is assumed that users have access to a stable internet connection.
- The application is dependent on the Django web framework and SQLite3 database for its core functionality.
- The application's performance may be affected by the availability and stability of third-party libraries or components used in development.

1.3 External Interface Requirements

1.3.1 User Interfaces

The user interface of the "W.G.T" web application will follow modern design principles and provide a responsive layout for various devices and screen sizes. The user interface will consist of:

- A main page with a link to the find nearest restaurant page
- A side-bar page displaying a list of functionalities
- A find nearest restaurant page to help users find their preferred restaurant
- A search page to search for relevant restaurants
- A leave review page for users to leave a review for a selected restaurant
- Detailed restaurant pages with information, zuser reviews, ratings, and directions to nearest carparks
- User account pages for managing profiles and reviews
- Contact Us page and FAQ page to help user be familiar with the application
- Administrator account page for managing user accounts and restaurants
- Common UI elements include a navigation bar, standard buttons (e.g., submit, cancel), help tooltips, and error message displays.

1.3.2 Hardware Interfaces

The "W.G.T" web application does not directly interact with hardware components, as it is a web-based application running in a web browser. The application will be designed to be compatible with various devices, including desktop computers, laptops, tablets, and smartphones.

1.3.3 Software Interfaces

- The "W.G.T" web application will interact with the following software components:
- Django web framework (version: 4.1.4) for the core application development
- SQLite3 for database management and storage
- External restaurant data source (Grab data base) for obtaining restaurant information
- External map API (Google Map) for directing users to the selected restaurant and car park
- Django's built-in authentication system for user account management
- Data items/messages exchanged between the application and external components include search queries, restaurant data, user authentication data, and user-generated content (reviews, ratings).

1.3.4 Communications Interfaces

The application will support email notifications for user account-related activities (e.g., password reset, account confirmation). In the demo stage, “W.G.T” is currently running at LocalHost. However, in the future implementation, the "W.G.T" web application will utilize standard web communication protocols, such as HTTP/HTTPS for data transfer and user interactions. User authentication will be secured using HTTPS and secure cookies.
All communication will adhere to relevant security and encryption standards, ensuring data privacy and protection.

2. Software Distribution

2.1 Required functionality

2.1.1 Functional Requirements

- 1. User Account Registration
  - 1.1 User must register with an email not already in the database
  - 1.2 User must register with a username not already in the database
  - 1.3 User must register with a password of at least 8 characters long
  - 1.4 System validates that all fields are entered correctly
  - 1.5 System validates that there are no conflicts with the database
    - 1.5.1 System shows an invalid prompt if there are conflicts
    - 1.5.2 System requests user to address conflicts
  - 1.6 Account System must record new account in database

- 2. User Account Login
  - 2.1 User must login with username and password
  - 2.2 System validates that all fields are entered correctly
  - 2.3 System validates that information entered matches with database
  - 2.4 System redirects user to the main page after successful validation
  - 2.5 If username entered is not in database, system will inform user that the username inputted was incorrect
  - 2.6 If the username entered is in the database but the password entered is wrong, System will inform User of the error.
    - 2.6.1 User is allowed a total of five tries to input the correct password

- 3. Account Recovery
  - 3.1 System sends an email verification associated with that username after the fifth failed attempt to reset the User's password
    - 3.1.1 E-mail verification link expires 5 minutes after e-mail was sent
  - 3.2 Users are able to reset their password by clicking on "Forgot your password?" button on login screen
    - 3.2.1 System sends an email to the user to allow them to reset their password
    - 3.2.2 E-mail verification link expires 5 minutes after e-mail was sent
    - 3.3 Account System must be able to change user's password in database

- 4. Location Service
  - 4.1 System must be able to gather the User's location through their device's location function
  - 4.2 System must allow User to input their location manually

- 5. Home Page
    - 5.1 User must be able to navigate to the find nearest restaurants page.

- 6. Map
    - 6.1 System must display a map, centred on the location the User has entered/provided by the location service
        - 6.1.1 Map shows area around User in 3 kilometre radius
    - 6.2 User must be able to scroll through the map
    - 6.3 User must be able to zoom in and out of the map

- 7. View User's Reviews
    - 7.1 System must display all reviews made by User
    - 7.2 System must allow User to edit review made by User
    - 7.3 System must allow User to delete review made by User

- 8. Review Making
    - 8.1 User must be able to write reviews for restaurants
        - 8.1.1 Reviews are limited to 300 characters long
    - 8.2 User must be able to add ratings to restaurants
        - 8.2.1 Ratings are on a scale from 1 to 5
    - 8.3 User must be able to edit reviews
    - 8.4 User must be able to edit ratings.

- 9. View Nearby Restaurants
    - 9.1 System must be able to retrieve restaurant information from database provided
    - 9.2 System must be able to display restaurants in a 3 kilometer radius from the location specified by User
        - 9.2.1 The displayed restaurants must show their cuisine type, proximity and ratings
    - 9.3 User must be able to input a restaurants name manually to view its information
    - 9.4 The restaurants should be displayed on the map overlay
    - 9.5 System must be able to filter the restaurants to display
        - 9.5.1 The filters are "cuisine" , "proximity" or "rating"
        - 9.5.2 Proximity must range from 0.5 kilometer to 5 kilometer
    - 9.6 System must be able to display a random restaurant in a 3 kilometer radius if the User selects the "random" button
        - 9.6.1 The random button takes in to account any filters applied on the System by the User
    - 9.7 System must display the map as specified in section 5

- 10. User Profile Page
    - 10.1 User must be logged in to their account to access this page.

        - 10.1.1 System must redirect users to login page if they are not logged in
        - 10.1.2 System must redirect users back to profile page after successful login
    - 10.1 User must be able to view information about his profile
    - 10.2 User must be able to view their past reviews in a table. (section 6)
    - 10.3 User must be able to update their particulars
    - 10.3.1 User must be able to update their email address
    - 10.4 User must be able to change their password
        - 10.4.1 User must input their current correct password first before attempting to change their password
        - 10.4.2 New Password must be at least 8 characters long
        - 10.4.3 System will inform the user if password does not meet requirements
        - 10.4.4 If new password does not meet the password requirements, system will re-prompt the user to key in the new password again.

- 11. View Specific Restaurants
    - 11.1 User must be able to select a restaurant to view more information
        - 11.1.1 Information such as cuisine type, distance, rating and opening hours must be displayed
    - 11.2 User must be able to read reviews left by other users
    - 11.3 User must be able to add reviews to the restaurant
    - 11.4 User must be able to add ratings to the restaurant
        - 11.4.1 Ratings are on a scale from 1 to 5

- 12. View Nearby Parking
    - 12.1 System must show available parking spaces that are at most 1 kilometer away from the restaurant
        - 12.1.1 System must have a filter to lower search radius up to 0.5 kilometers
        - 12.2 User must be able to select a specific parking space from those displayed to navigate to.
        - 12.3 System must display the number of available lots and total number of lots at parking space specified

- 13. Pathfinding
    - 13.1 User must be able to select “Get directions” at either the “Specific Restaurant” page or “View Nearby Parking” page
        - 13.1.1 System must redirect User to Google Maps, inputting the location for the User

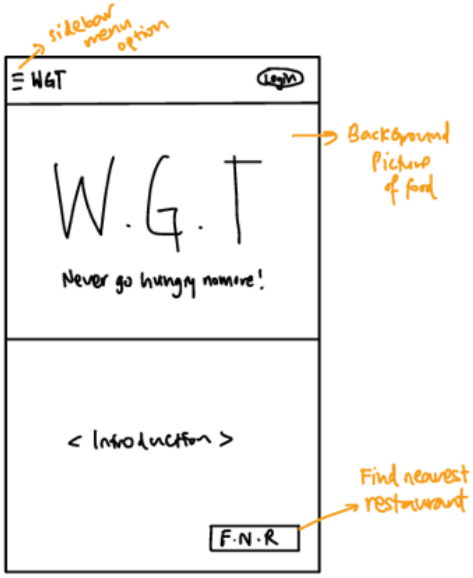
- 14. Frequently Asked Questions (FAQ)
    - 14.1 System must be able to display frequently asked questions

2.1.2 Non-Functional Requirements

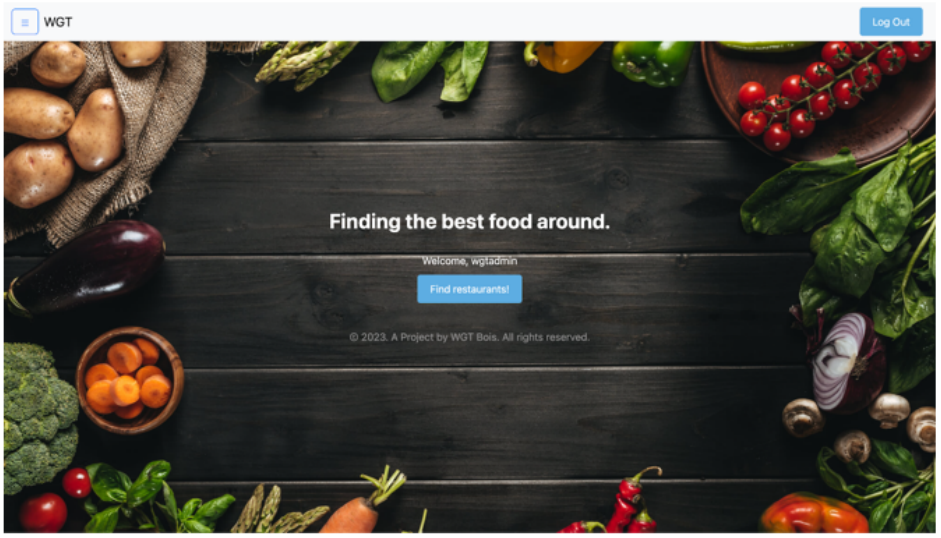
- 15.1 The system must be able to support up to 100 concurrent users
- 15.2 The system must take up to 3 seconds to load under normal network conditions
- 15.3 The system must be supported by the common web-browsers used
- 15.4 The system must scale according to the screen size of the device used by the User
- 15.5 The database must support standard SQL/MySQL queries

2.2 UI Mockups

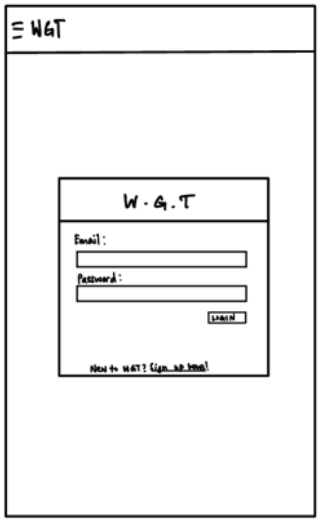
1.Original main page



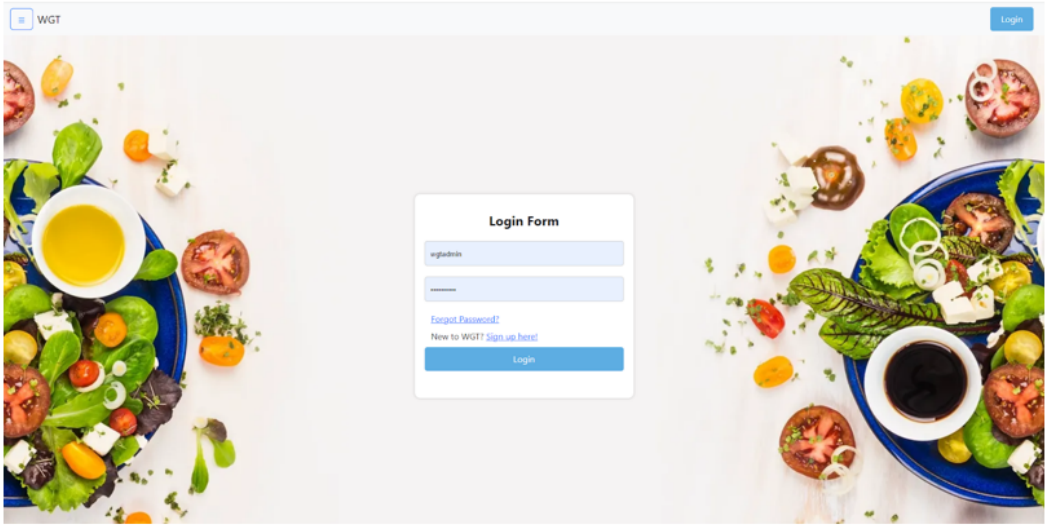
2.Final main page



3. Original login page

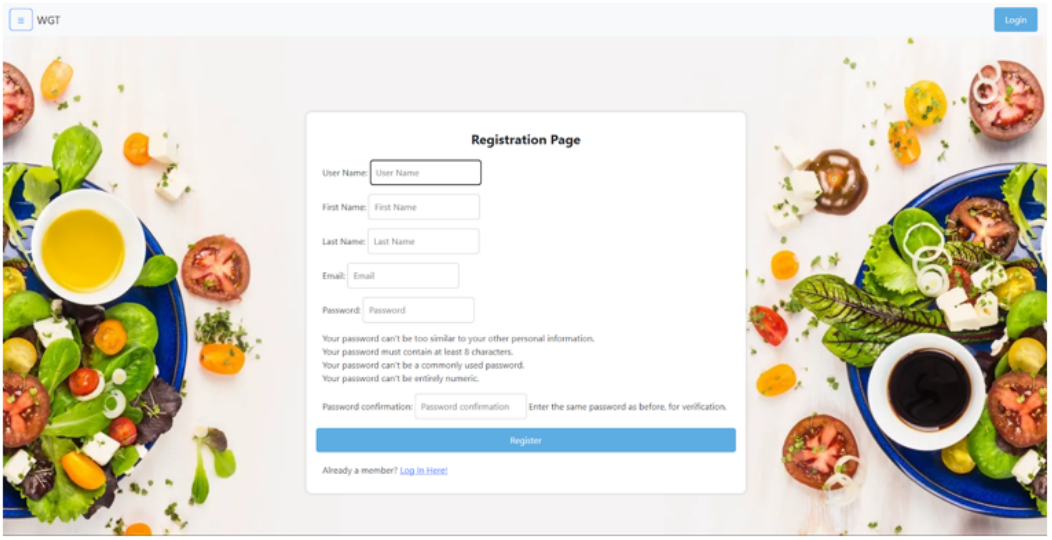


4. Final login page



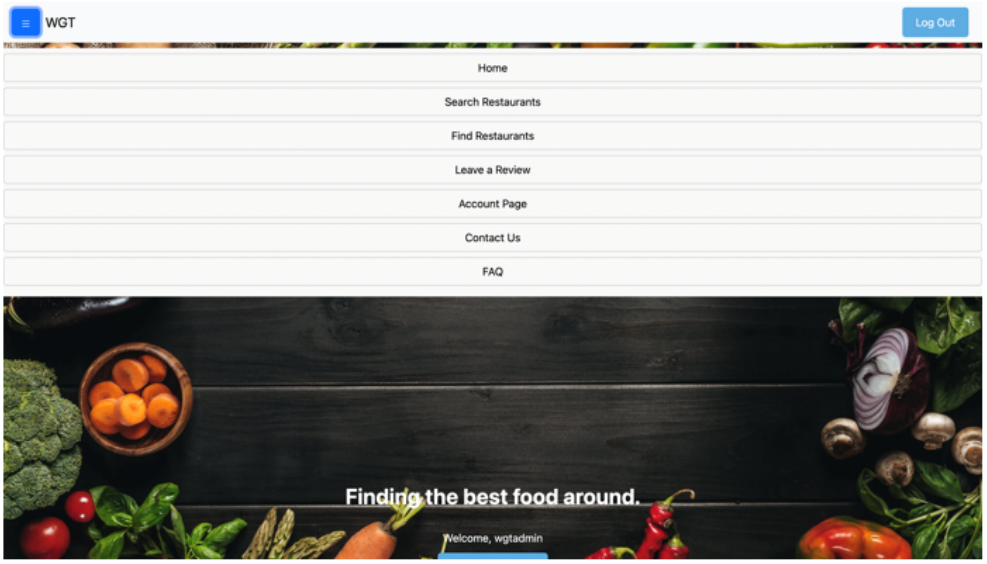
5. Original registration page

6. Final registration page

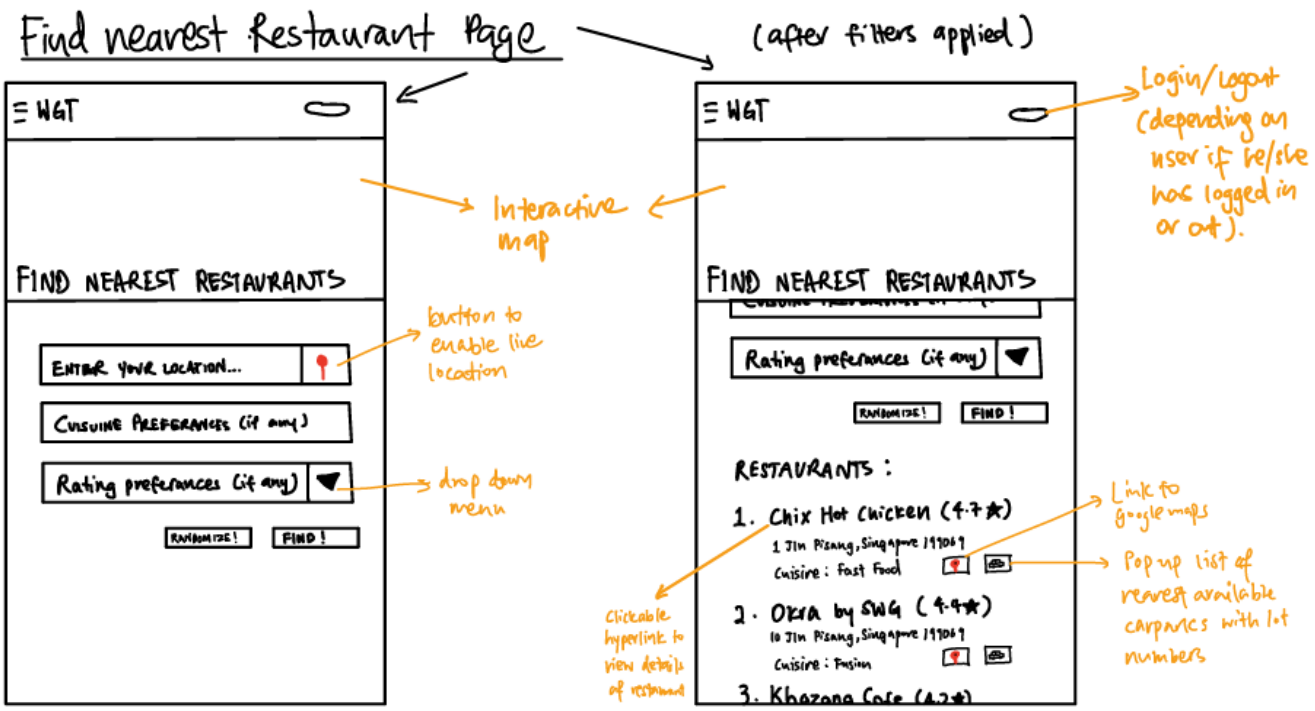


7. Original side bar

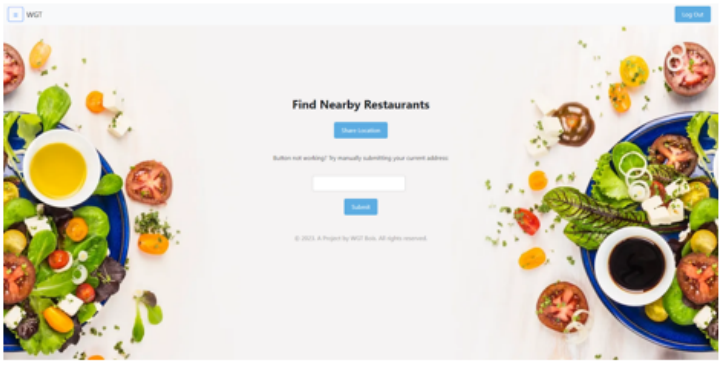
8. Final side bar

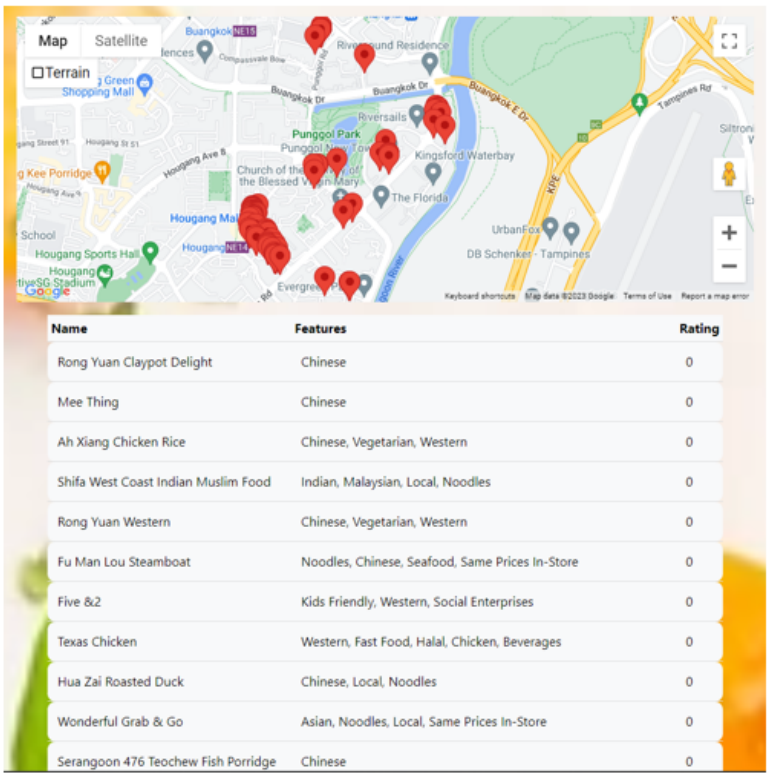


9. Original find nearest restaurant page 10. Original find restaurant page (After filtering)



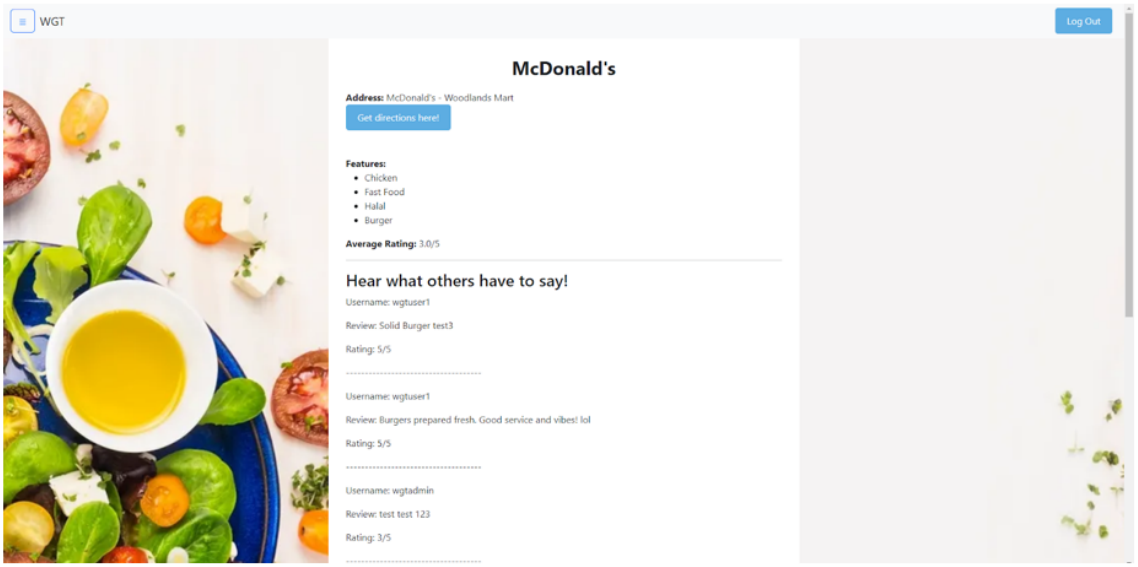
11,12,13. Final find nearest restaurant page



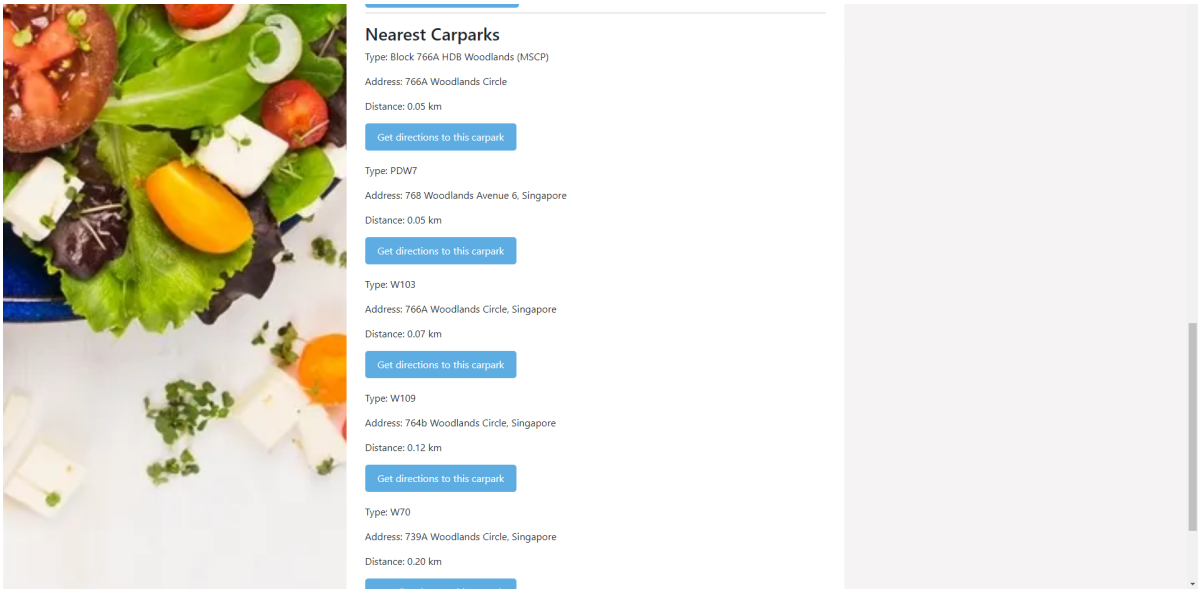


14. Original restaurant page

15. Final restaurant page (1)

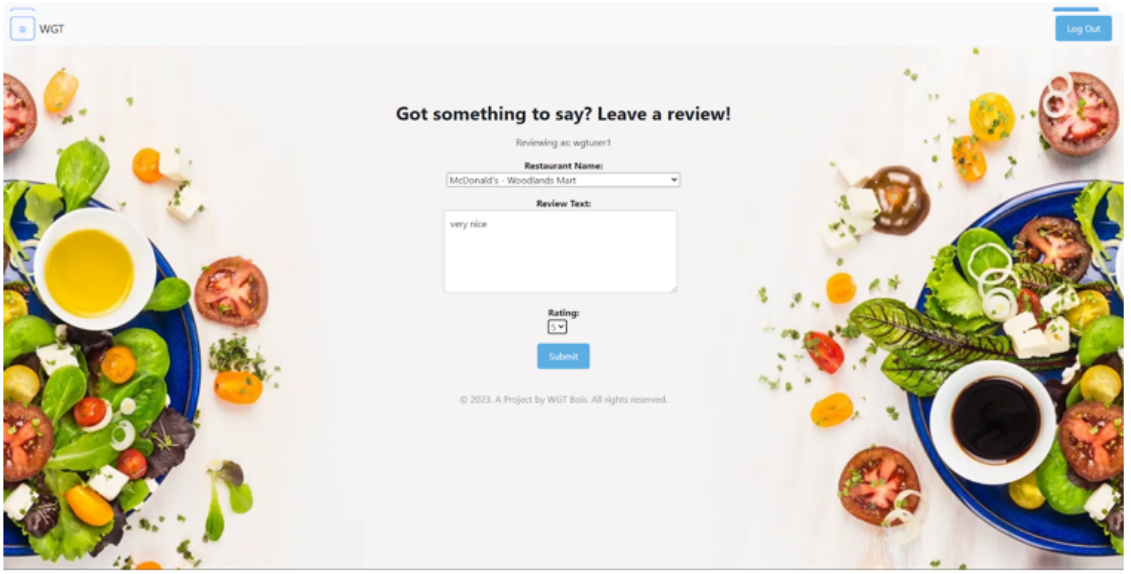


16. Final restaurant page (2)

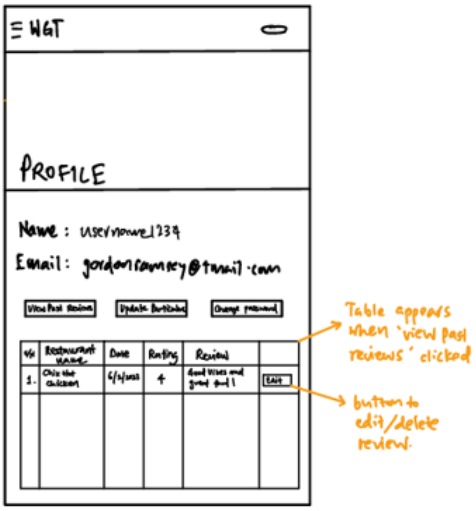


17. Original leave review page

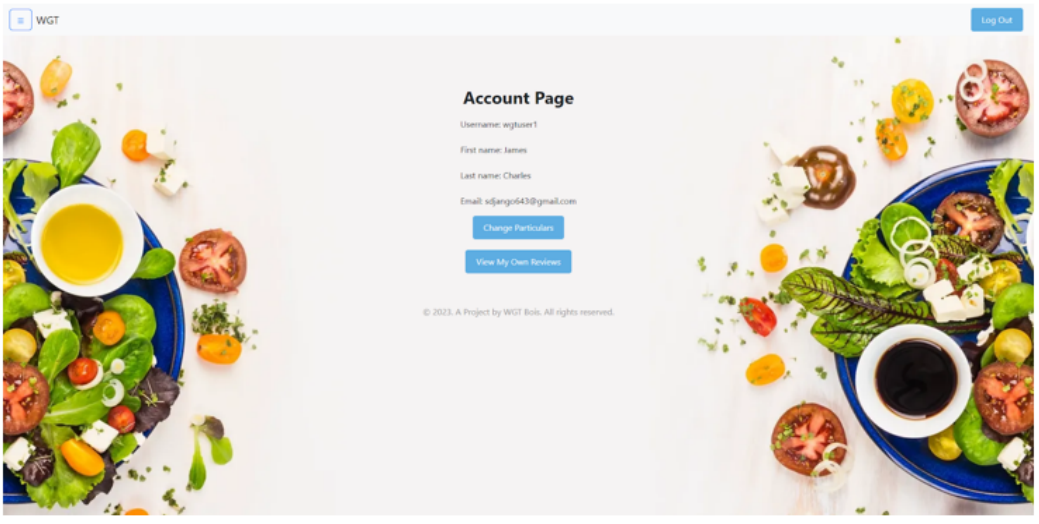
18. Final leave review page



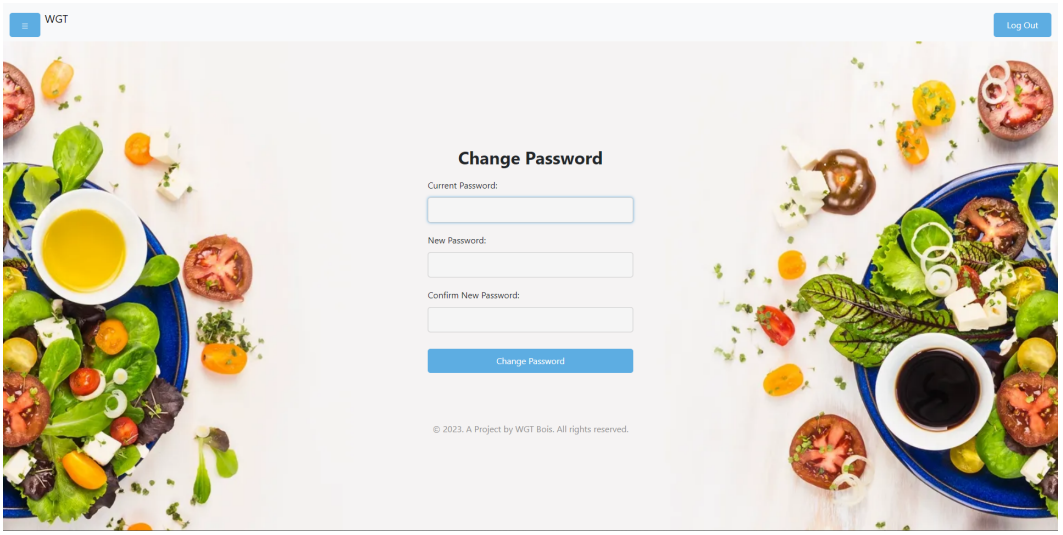
19. Original account page



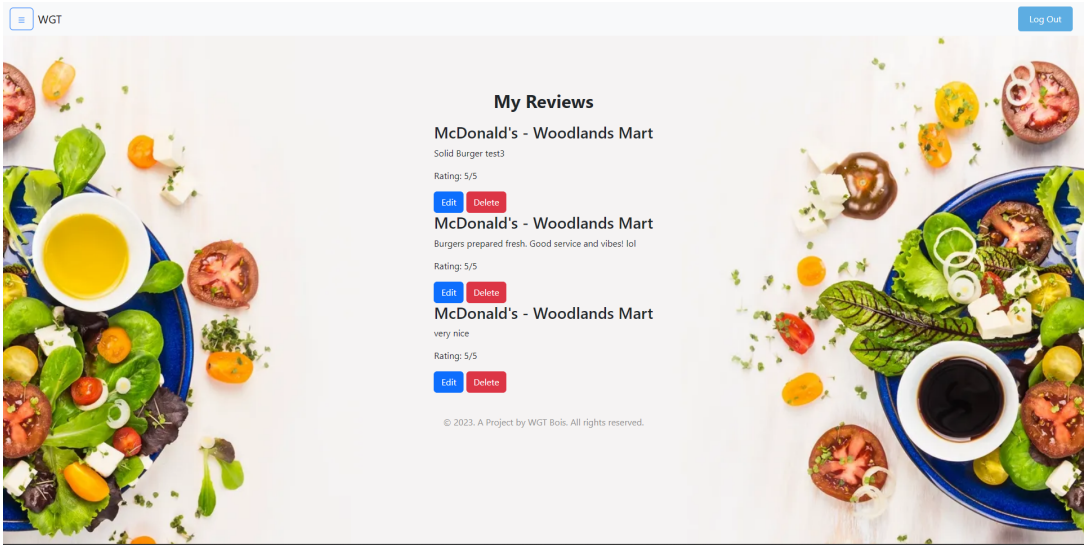
20. Final account page



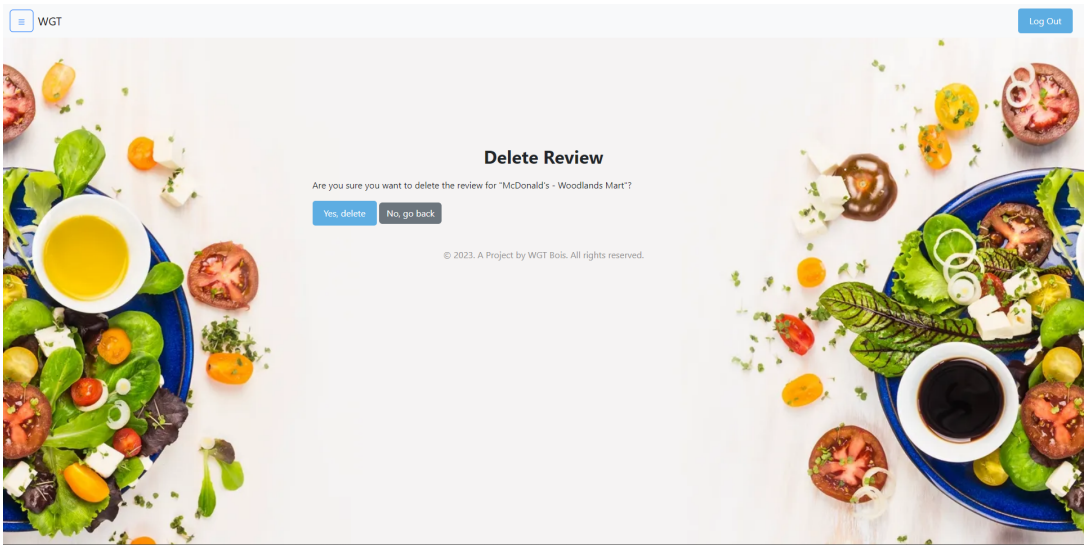
21. Final account page (change password)



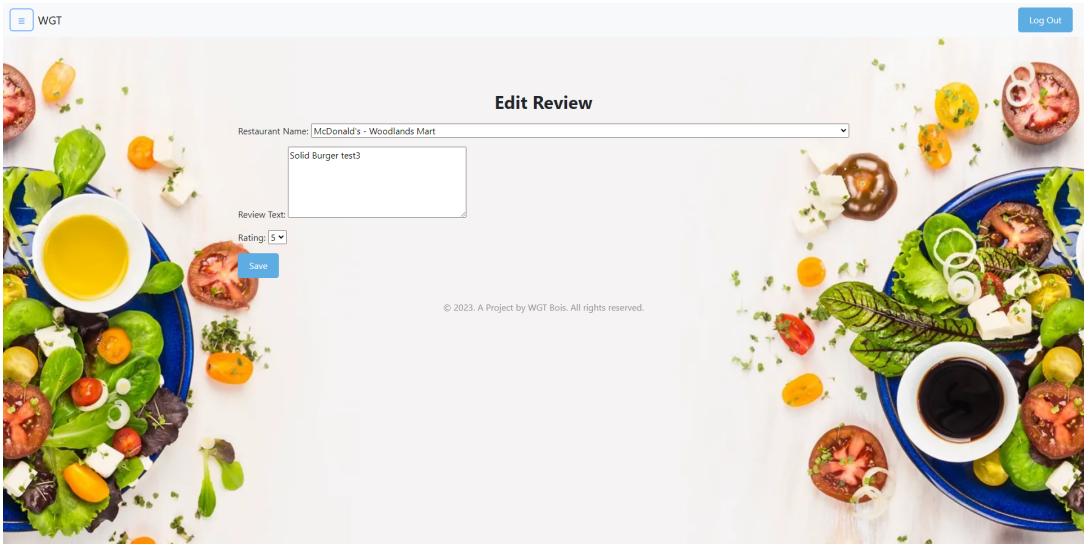
22. Final account page (view own reviews)



23. Final account page (delete review)

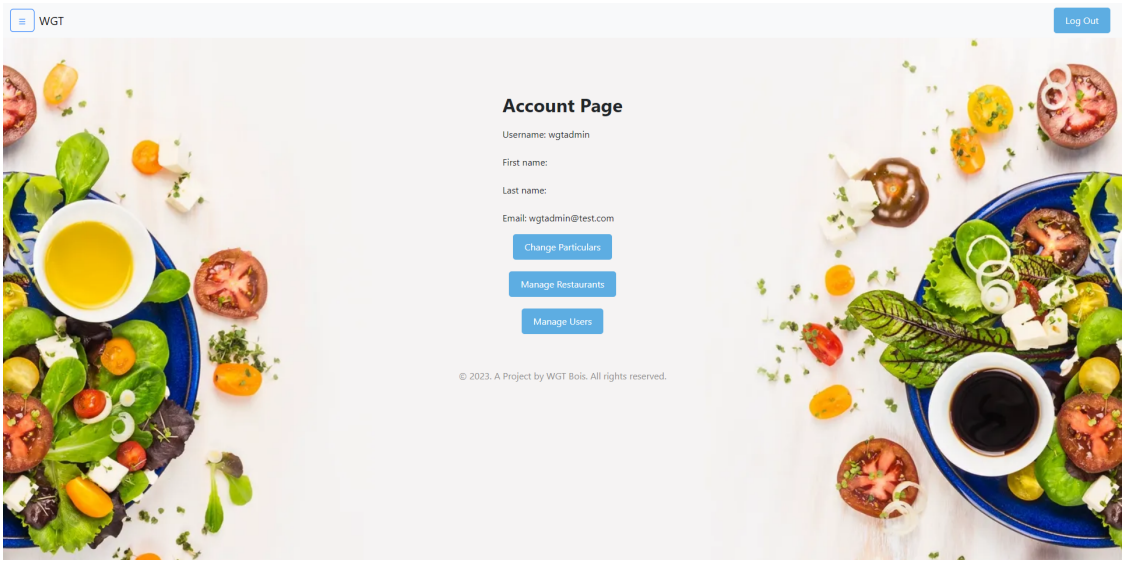


24. Final account page (edit review)

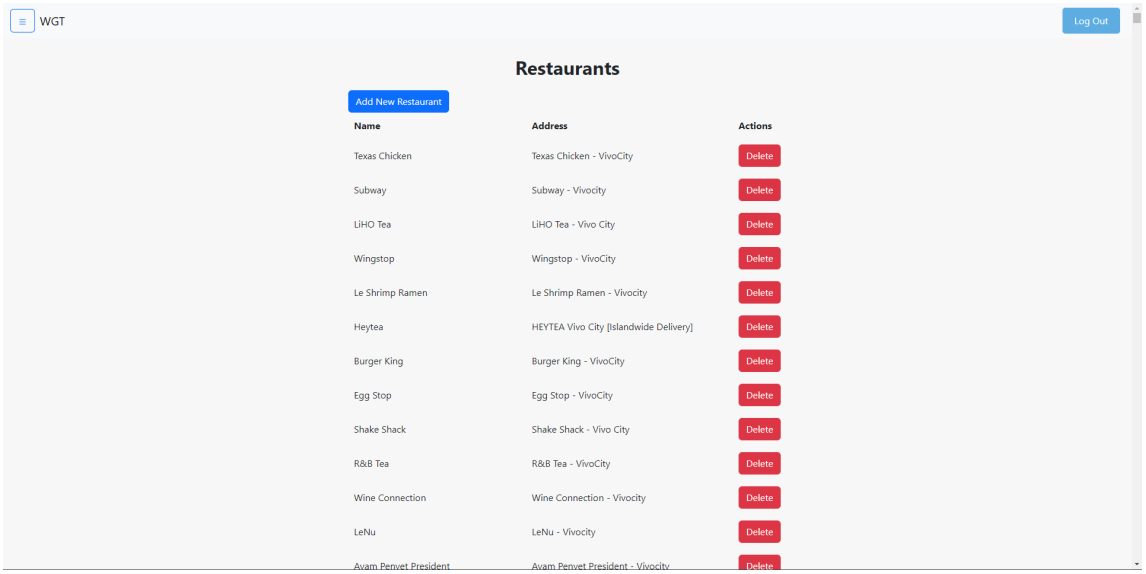


2.2.1 Admin Account's Additional Accessibility

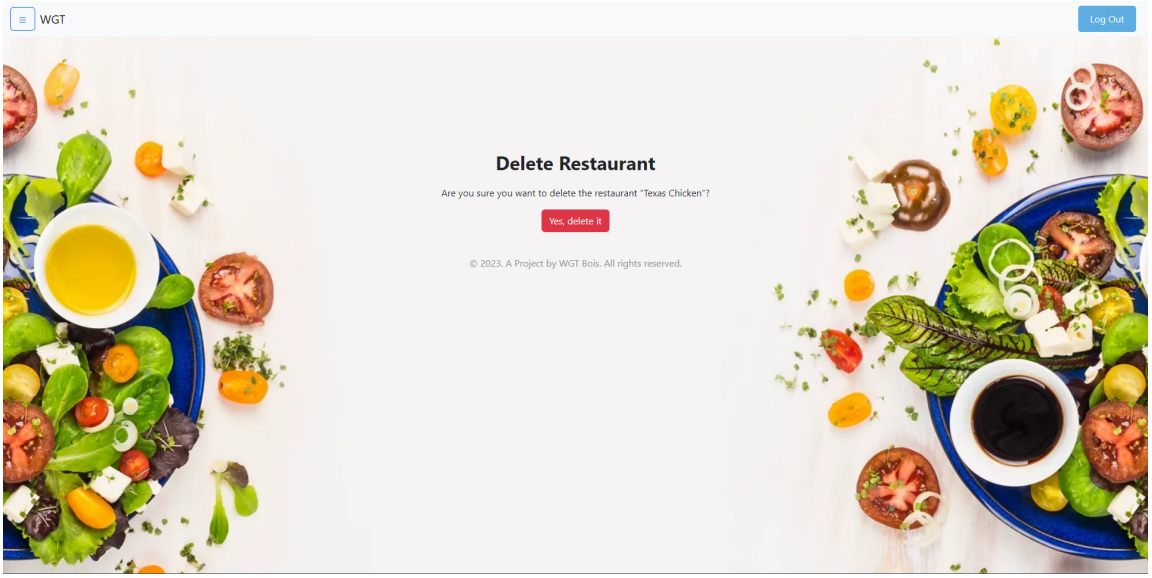
25. Admin management page



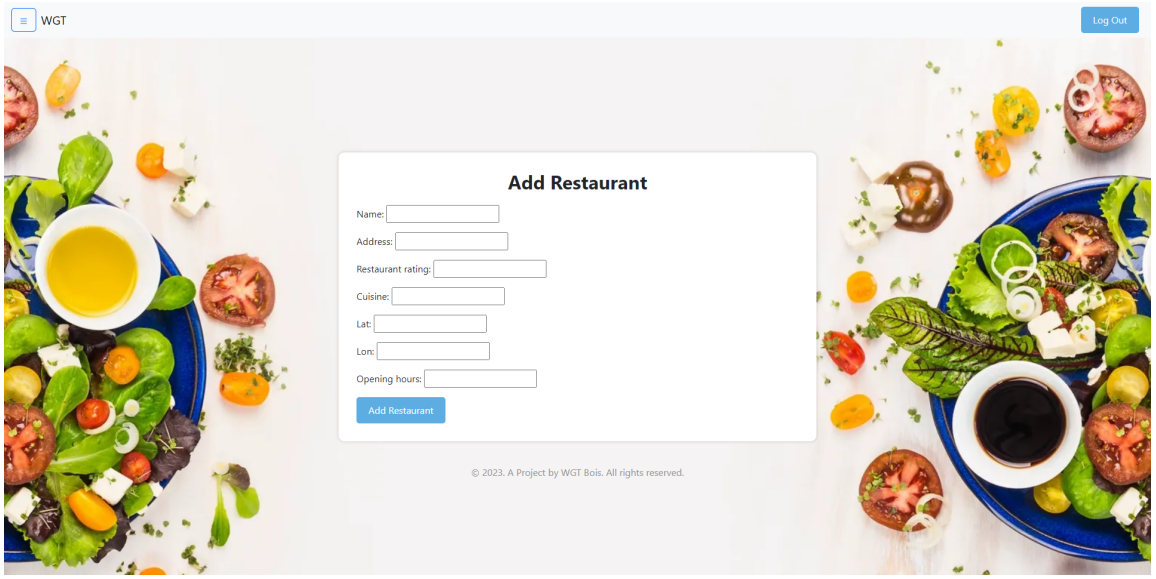
26. Admin page (view restaurant)



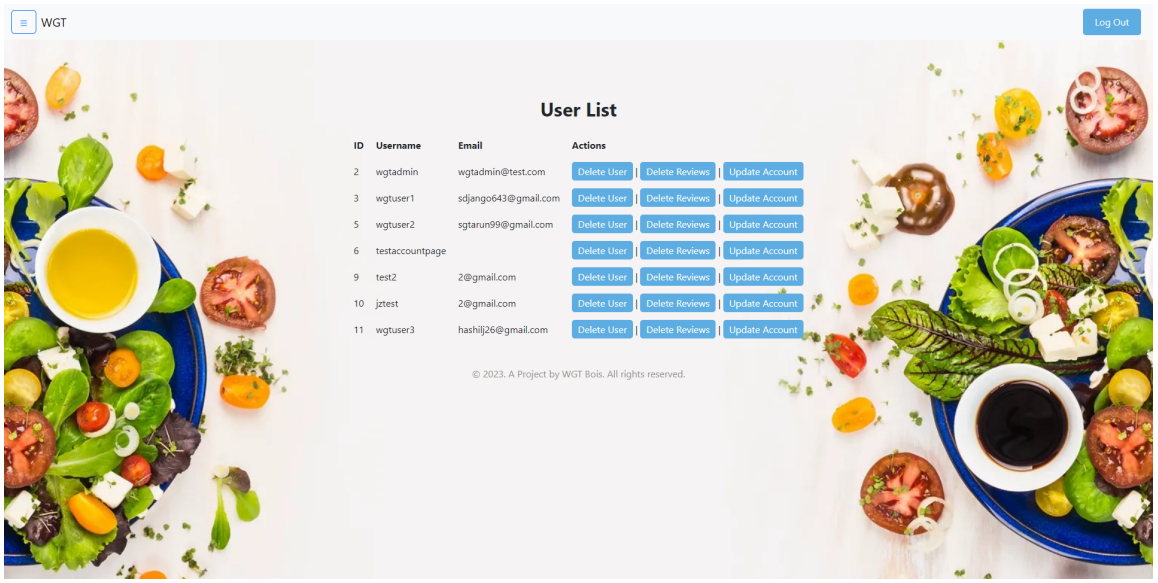
27. Admin page (delete restaurant)



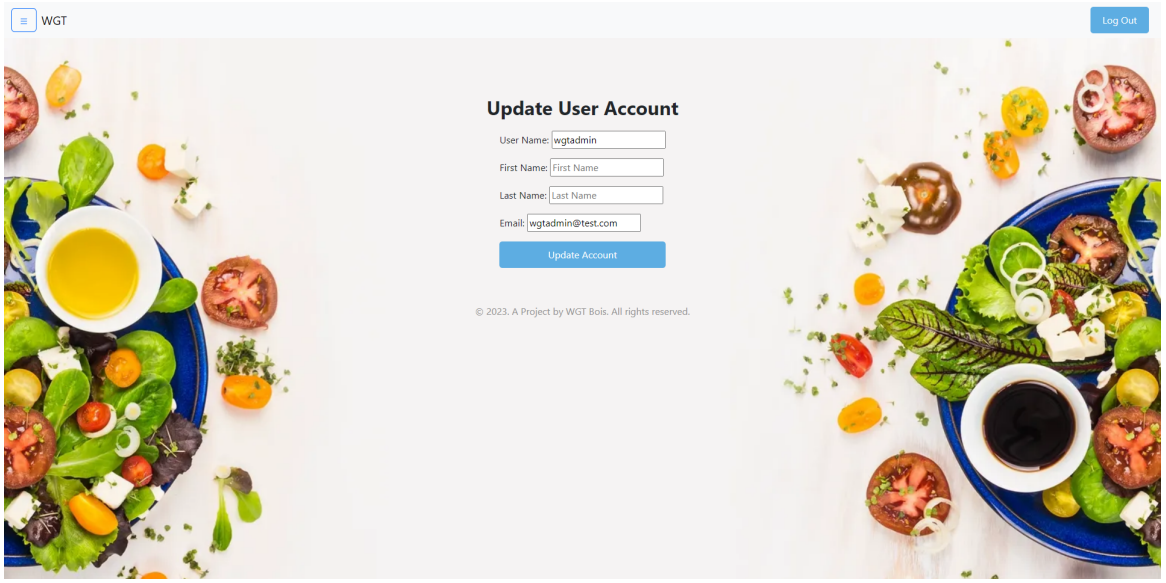
28. Admin page (add restaurant)



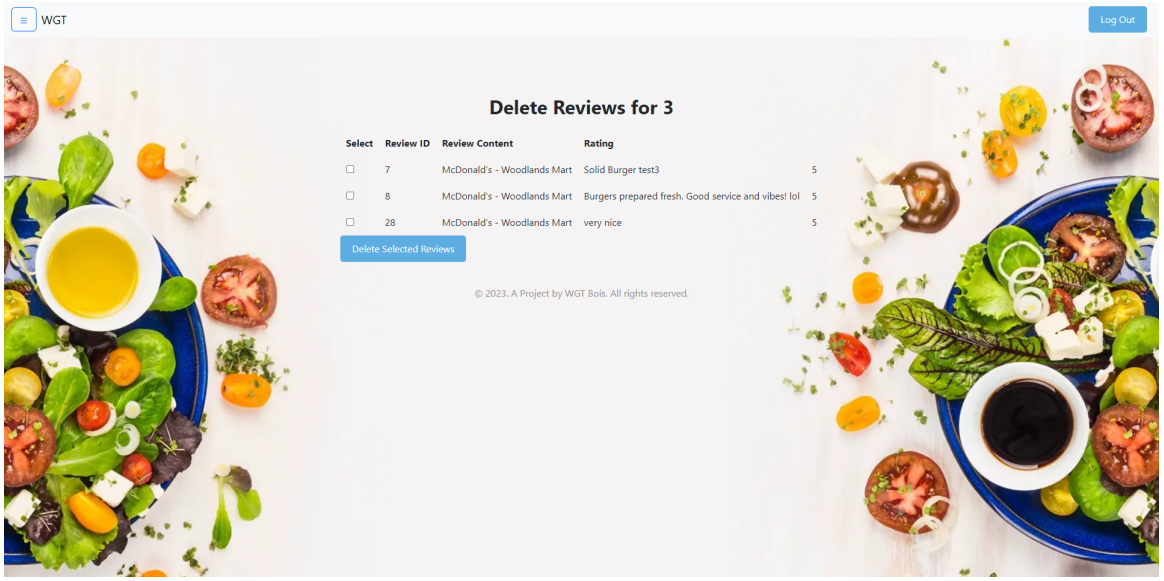
29. Admin page (view users)



30. Admin page (update user account)



31. Admin page (delete user reviews)



3. Technical Documentations

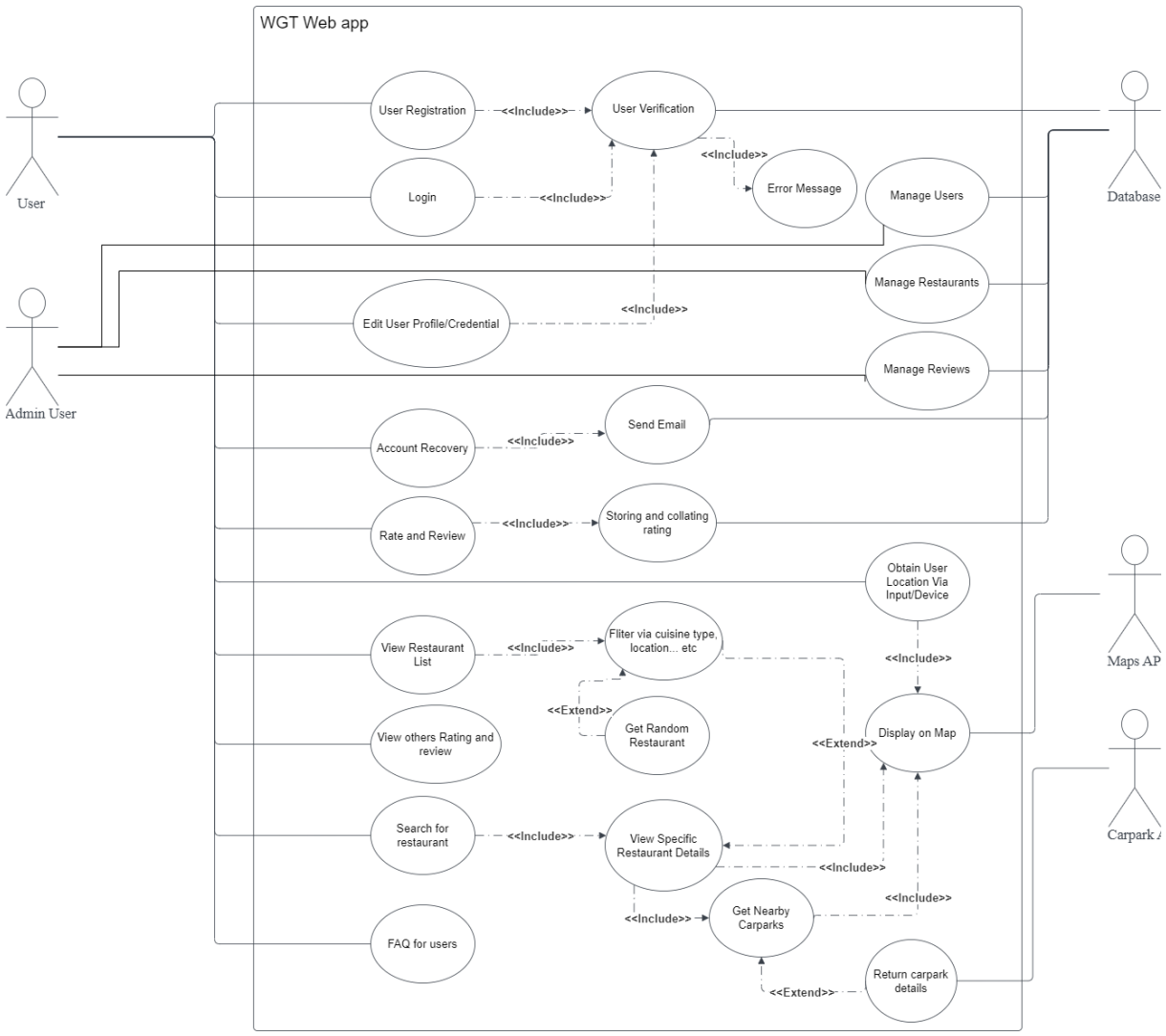
3.1 Data Dictionary

Term	**Definition**
Cuisine	A style or method of cooking that is associated with a specific culture, or a type of food. eg. Western, Chinese, Indian, BBQ, Pasta. etc.
Rating	A numerical value representing the overall quality of a restaurant on a scale of 1 to 5, with 5 being the best and 1 being the worst.
Review	A written evaluation of a restaurant by a user, including a rating and comments.
Map	A diagrammatic representation of Singapore; displaying its physical features, roads, towns, restaurants, places of interest etc.
Carpark	A designated area for people to park their cars.
User	Human user of a system to find food establishments.
Location	The current geographical coordinates of the user, determined by user input or user's device.
Address	Details about the specific location of a restaurant,
Proximity	How near the various food establishments and supermarkets are to the user. It can be limited to a range as well: 1km, 2km, 3km, 4km etc.
Restaurant	A place where people pay to sit and eat meals that are cooked and served on the premises.
System	The system refers to the WGTFood web application that is optimised for all screen types.
Filter	Filter is a feature that processes information to exclude the types which are not wanted, and/or sort the information in order. (example: Rating high to low, Price low to high, Cuisine etc)
Search	Search is a feature that allows users to find and discover a variety of restaurants based on their own combination of filters.
GPS	Global Positioning System which will detect a user's current location.

Latitude	The angular distance of a place north or south of the earth's equator.
Longitude	The angular distance of a place east or west of the earth's prime meridian.

3.2 Use Case Model

3.2.1 Use Case Diagram



3.2.2 User Case Descriptions

Use Case ID:	1		
Use Case Name:	User Account Registration		
Created By:	Pei He	Last Updated By:	Pei He
Date Created:	5/2/22	Date Last Updated:	10/4/22

Actor:	User, Database System
Description:	User registration process
Preconditions:	User is on login page, user is not already logged in
Postconditions:	User creates a new account.
Priority:	High
Frequency of Use:	
Flow of Events:	1) User clicks on “Sign up here!” 2) System displays a registration page, and the user is prompted to enter his username, first name, last name, email, and password and to confirm his password 3) User enters the required login information and clicks on the register button 4) Database system verifies that the email address is valid, and that the passwords match and is at least 8 characters long 5) Database system checks that the email is not in the system database 6) Database System encrypts user details and logs new account into database 7) User is redirected to login page
Alternative Flows:	AF- 3) : User enters an e-mail that is already in database 1. System informs User that e-mail has already been registered 2. System returns to 2) AF- 3) : User enters a username that is already in database 1. System informs User that username inputted has already been used 2. System returns to 2) AF- 3) : User enters a password that is not 8 characters long 1. System informs User that password inputted has to be at least 8 characters long 2. System returns to 2) AF- 3) : User enters passwords that do not match 1. System informs User that passwords have to match 2. System returns to 2)
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	2		
Use Case Name:	User Account Login		
Created By:	Holdon	Last Updated By:	Pei He
Date Created:	05/02/2023	Date Last Updated:	10/4/23

Actor:	User, Database System
Description:	User login process
Preconditions:	User has a registered account; user is not logged in
Postconditions:	User successfully logs in
Priority:	High
Frequency of Use:	
Flow of Events:	1) User clicks on the “login” button 2) User enters their username and password into the fields 3) Account System validates login credentials 4) System redirects User to main page, if validated
Alternative Flows:	AF - 2): User inputs invalid credentials 1. If User inputs a username not in the database, System will inform User that username is not in database 2. If User inputs a username in the database but inputs the wrong password, System will inform User of error
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	3		
Use Case Name:	Account Recovery		
Created By:	Holdon	Last Updated By:	Pei He
Date Created:	05/02/2023	Date Last Updated:	10/4/2023

Actor:	User, Database System
Description:	User has ability to recover account if they forget their password
Preconditions:	User has an account in the system; user is on the login page.
Postconditions:	User resets their password.
Priority:	High
Frequency of Use:	
Flow of Events:	1) User clicks on the “Forgot Password?” option. 2) User is prompted to enter their email. 3) User keys in their email. 4) System sends email with password reset link to user and displays success message. 5) User clicks on the link sent in his email. 6) System displays the password reset web page, prompting users to enter a new password and confirm it 7) User enters a new password and confirms it 8) System verifies that the passwords match and are at least 8 characters long. 9) Database System updates user database with new password 10) User is redirected to login page with a success message.
Alternative Flows:	AF - 5) : User does not click on email link 1. If user does not click on link 5 minutes after e-mail was sent, the link will expire 2. Use case terminates AF - 7) : User’s new passwords do not match 1. System informs user that the passwords do not match and requests that the user try again 2. System returns to 6) AF - 7): User’s passwords are less than 8 characters long 1. System informs user that the new password is not strong enough 1. System returns to 6)
Exceptions:	
Includes:	
Special Requirements:	

Assumptions:	
Notes and Issues:	

Use Case ID:	4		
Use Case Name:	Location Service		
Created By:	Holdon	Last Updated By:	Pei He
Date Created:	05/02/2023	Date Last Updated:	9/4/23

Actor:	User, Maps API
Description:	User must allow System to get their device location for the application to function
Preconditions:	User is has not entered their location, and is on the find nearest restaurants page.
Postconditions:	System stores user's location; user is redirected to Find Nearest Restaurants page.
Priority:	High
Frequency of Use:	
Flow of Events:	1) System prompts user to share their location 2) User clicks on "share location" button, or manually inputs their address to search box. 3) System records User's current location 4) System displays user location on a map. 5) System displays filter options, and "search" and "I'm feeling lucky" buttons.
Alternative Flows:	AF - 1): If user has not enabled/does not have device location 1. The system prompts the user to allow device location 2. User allows device location 3. System returns to 3), otherwise it alerts user to use manual input instead.
Exceptions:	
Includes:	Find Nearby Restaurants
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	5		
Use Case Name:	Accounts Page		
Created By:	Holdon	Last Updated By:	Pei He
Date Created:	05/02/2023	Date Last Updated:	23/3/2023

Actor:	User, Database System
Description:	Accounts page displays information related to the user and offers the user the option to update particulars or view his reviews.
Preconditions:	User is logged in.
Postconditions:	User is on the accounts page.
Priority:	Medium
Frequency of Use:	
Flow of Events:	1) User selects “Accounts Page” from sidebar. 2) System obtains user particulars from database system. 3) System displays user particulars. 4) System displays buttons for user to “Change Particulars” and “View my reviews”. 5) Database System displays reviews previously made by the User, as well as buttons to edit or delete the reviews for each specific review.
Alternative Flows:	
Exceptions:	
Includes:	View Own Reviews, Settings Update
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	6		
Use Case Name:	View Own Reviews		
Created By:	Holdon	Last Updated By:	Holdon
Date Created:	05/02/2023	Date Last Updated:	23/03/2023

Actor:	User, Database System
Description:	Database system contains all the reviews left by the User for reference
Preconditions:	User is in the “Accounts” page; User is logged in
Postconditions:	User views
Priority:	Medium
Frequency of Use:	
Flow of Events:	1) User selects “View my reviews” on his/her user profile page 2) System obtains user reviews from database system. 2) System displays reviews previously made by the User, as well as buttons to edit or delete the reviews for each specific review.
Alternative Flows:	
Exceptions:	
Includes:	Editing/Deleting Reviews
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	7		
Use Case Name:	Editing/Deleting Reviews		
Created By:	Holdon	Last Updated By:	Pei He
Date Created:	05/02/2023	Date Last Updated:	30/3/23

Actor:	User, Database System
Description:	Reviews left by Users can be edited or deleted if the User chooses
Preconditions:	User is logged in; User has made reviews.
Postconditions:	User makes changes to existing review
Priority:	Medium
Frequency of Use:	
Flow of Events:	1) User selects to edit specific review on “View Reviews” page 2) User edits previously made review 3) User edits previously given rating 4) System updates review and rating in database. 5) System displays success message.
Alternative Flows:	AF 2): User wants to delete review 1. User selects to delete specific review on “View Reviews” page 2. System requests for user confirmation 3. User confirms the request. 4. System removes the review and updates review database 5. System displays success message.
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	8		
Use Case Name:	Settings Update		
Created By:	Holdon	Last Updated By:	Holdon
Date Created:	05/02/2023	Date Last Updated:	05/02/2023

Actor:	User, Account System
Description:	User can be able to change their password, email, or other particulars.
Preconditions:	User is on accounts page, user is logged in.
Postconditions:	User exits settings menu
Priority:	Medium
Frequency of Use:	
Flow of Events:	1) User selects “Change Particulars” on accounts page. 2) System displays input forms for user to update particulars, as well as a “Change Password” button. 3) User updates his particulars and clicks submit. 4) System validates the updated particulars. 5) System sends updated user particulars to database system. 6) System displays success message to user.
Alternative Flows:	AF 4): User submits invalid particulars 1. System displays update failure message AF 2) : User wants to change password 1. User selects “Change Password” 2. User inputs current password 3. User inputs new password, which is 8 characters or longer 4. System displays success message if password change is successful, or failure message if user inputs wrong password or invalid new passwords.
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	9		
Use Case Name:	Find Nearby Restaurants		
Created By:	Holdon	Last Updated By:	
Date Created:	05/02/2023	Date Last Updated:	

Actor:	User, Database System, Maps API
Description:	System shows User all restaurants in their vicinity
Preconditions:	Location service is enabled for user.
Postconditions:	User can view nearby restaurants or select a specific restaurant page to view.
Priority:	High
Frequency of Use:	
Flow of Events:	1) User can update filter based on cuisine types, dietary restrictions, proximity and minimum rating or just use the default options, and clicks “Search”. 2) System retrieves information of restaurants from the database based on the filters from database system. 3) System calls Map API, which displays retrieved restaurants on an interactive map. 4) A list of the restaurants also displayed below the map. Clicking any food establishment in the list brings the user to the specific food establishment page
Alternative Flows:	AF 1): User wants to update filters. 1. User selects filter options and clicks “Search” again 2. System goes to 2) AF 3): User wants to select a random restaurant 1. User selects the “I’m feeling lucky” button 2. Restaurant System accounts for any filters in place 3. Restaurant System displays a specific restaurant at random
Exceptions:	
Includes:	Restaurant Page
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	10		
Use Case Name:	Restaurant Page		
Created By:	Holdon	Last Updated By:	Pei He
Date Created:	05/02/2023	Date Last Updated:	30/03/2023

Actor:	User, Database System
Description:	User can see details about a specific restaurant
Preconditions:	User clicks a restaurant on the restaurant list or on interactive map from Find Nearest Restaurants page or search results.
Postconditions:	User exits the restaurant page
Priority:	High
Frequency of Use:	
Flow of Events:	1) User selects a specific restaurant on list or map. 2) System queries database for restaurant information and reviews 3) System displays the restaurant's cuisine, reviews and rating. 4) System displays buttons for user to get directions, leave a review, or view carparks.
Alternative Flows:	
Exceptions:	
Includes:	View Nearby Parking, Pathfinding, Add Review
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	11		
Use Case Name:	View Nearby Parking		
Created By:	Holdon	Last Updated By:	Pei He
Date Created:	05/02/2023	Date Last Updated:	30/3/23

Actor:	User, Maps API
Description:	User can view parking spots near the restaurant selected
Preconditions:	User is on a restaurant page
Postconditions:	User can view nearby carparks and get directions to them.
Priority:	Medium
Frequency of Use:	
Flow of Events:	1) User selects “View Carparks” 2) Maps API retrieves parking information for carparks in 500m radius 3) System displays nearby carparks, as well as button to get directions to the carparks.
Alternative Flows:	
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	12		
Use Case Name:	Pathfinding		
Created By:	Holdon	Last Updated By:	Pei He
Date Created:	05/02/2023	Date Last Updated:	30/3/23

Actor:	User, Database System, Maps API
Description:	User can find a route from their current location to the specific restaurant or specific parking space they want to get to
Preconditions:	User selects “Get Directions Here!” at specific restaurant page User selects “Get Directions to this Carpark” at specific parking space page.
Postconditions:	User exits application
Priority:	Medium
Frequency of Use:	
Flow of Events:	1) User chooses to obtain directions to somewhere. 2) System obtains coordinates of restaurant from database or parking space from maps API 3) System calls Maps API with the coordinates 4) Google maps opens which shows the user the navigation path to the coordinates.
Alternative Flows:	
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	13		
Use Case Name:	Add review		
Created By:	Pei He	Last Updated By:	Pei He
Date Created:	06/02/23	Date Last Updated:	30/3/23

Actor:	User, Database System
Description:	User can add a review to a restaurant they went to
Preconditions:	User is logged in
Postconditions:	User is returned to the restaurants page
Priority:	High
Frequency of Use:	
Flow of Events:	1) User selects “add review” on the restaurants page or sidebar 2) System prompts the user to select a rating (1-5), as well as any comments about the food establishment 3) User inputs the rating and comments, and clicks submit 4) Review system adds the review to its database 5) Restaurant system recalculates a new average rating, and displays it accordingly 6) The message “review successfully added” is displayed to the user
Alternative Flows:	
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	14		
Use Case Name:	Admin user settings		
Created By:		Last Updated By:	
Date Created:	06/02/23	Date Last Updated:	

Actor:	Admin, Database System
Description:	Admin can manage users and their reviews
Preconditions:	User is logged in, user is an Admin, user selects manager users
Postconditions:	Admin exits settings menu
Priority:	High
Frequency of Use:	
Flow of Events:	1) System displays all users in the database 2) Admin selects “delete user” on specific user 3) System asks admin to confirm deletion 4) Admin acknowledges 5) System returns to manage user page 6) User exits settings menu
Alternative Flows:	AF 2) Admin selects “delete reviews” on specific user 3) System displays all reviews made by user 4) Admin selects reviews to delete 5) System deletes reviews selected 6) System goes to 5) AF 2) Admin selects “update account” on specific user 3) System displays account details of specific user 4) Admin updates details of specific user and clicks on “update account” 5) System goes to 5)
Exceptions:	
Includes:	
Special Requirements:	

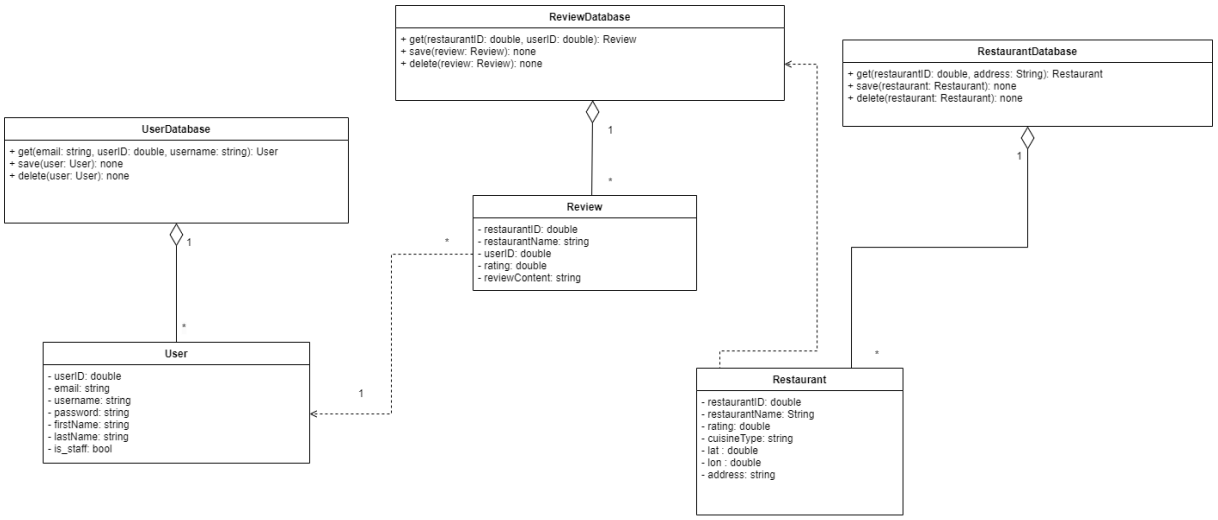
Assumptions:	
Notes and Issues:	

Use Case ID:	15		
Use Case Name:	Admin restaurant settings		
Created By:		Last Updated By:	
Date Created:	06/02/23	Date Last Updated:	

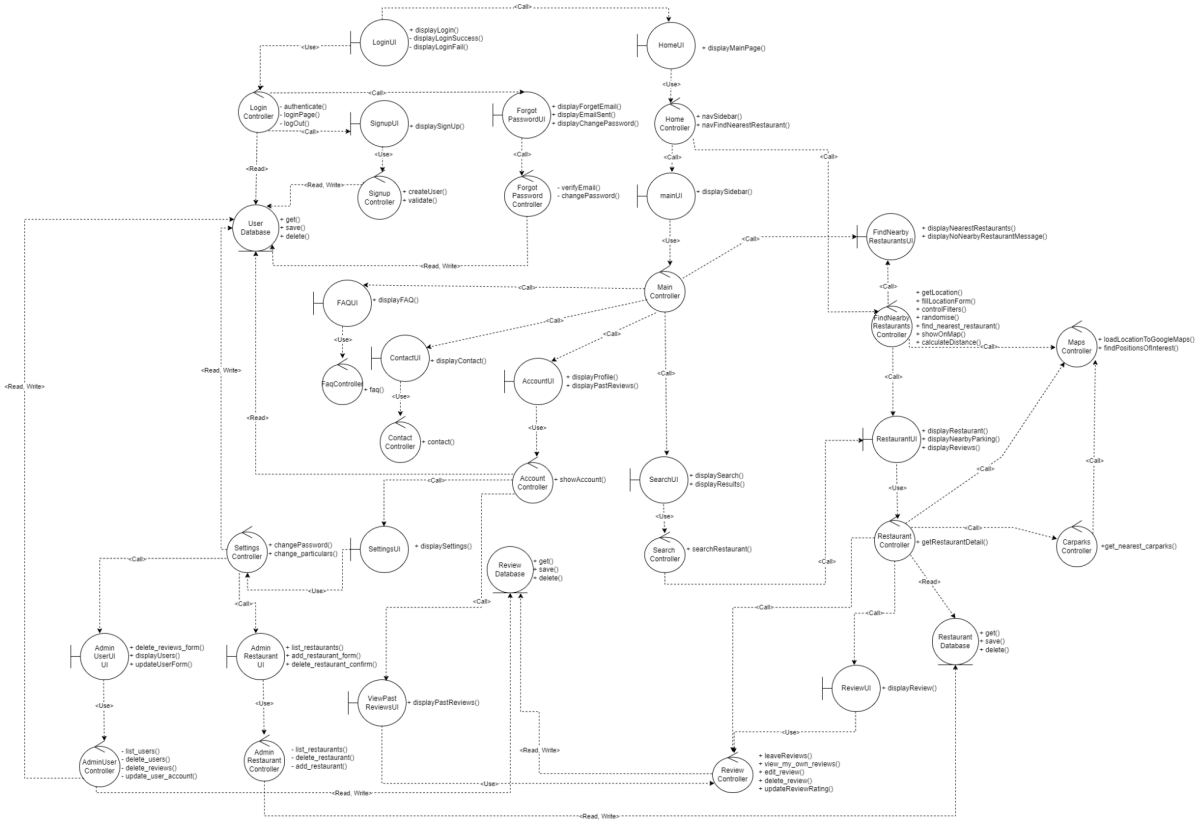
Actor:	Admin, Database System
Description:	Admin can manage manage restaurants
Preconditions:	User is logged in, user is an Admin, user selects “manage restaurants”
Postconditions:	Admin exits settings menu
Priority:	High
Frequency of Use:	
Flow of Events:	1) Admin selects “Add new restaurant” 2) Admin enters in restaurant details 3) System adds restaurant into database 4) System returns to manage restaurants page
Alternative Flows:	AF 1) Admin selects “delete restaurant” on specific restaurant 2) System ask for confirmation of deletion 3) Admin acknowledges 4) System returns to manage restaurants page
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3 Conceptual Model

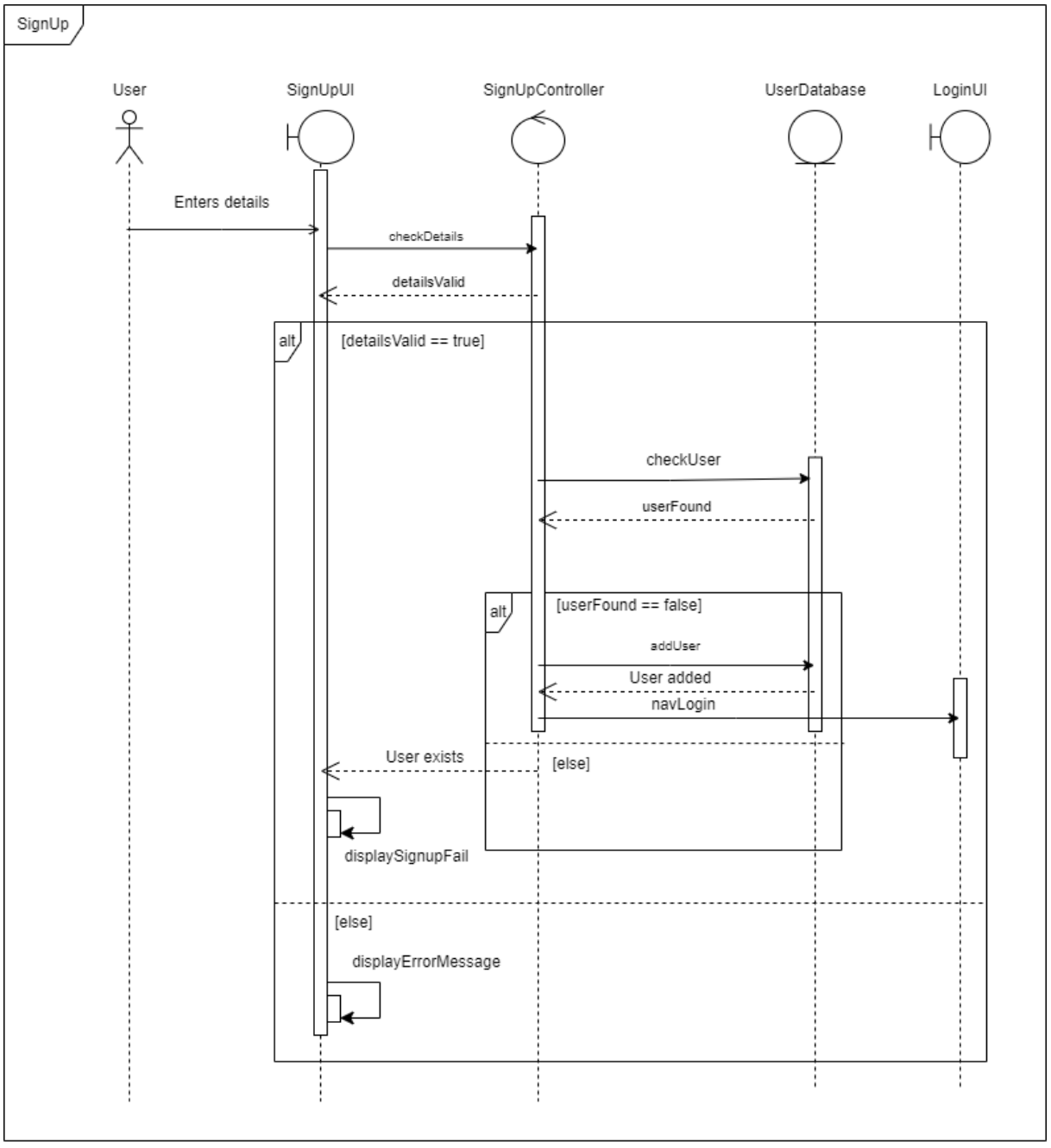
3.3.1 Entity Classes

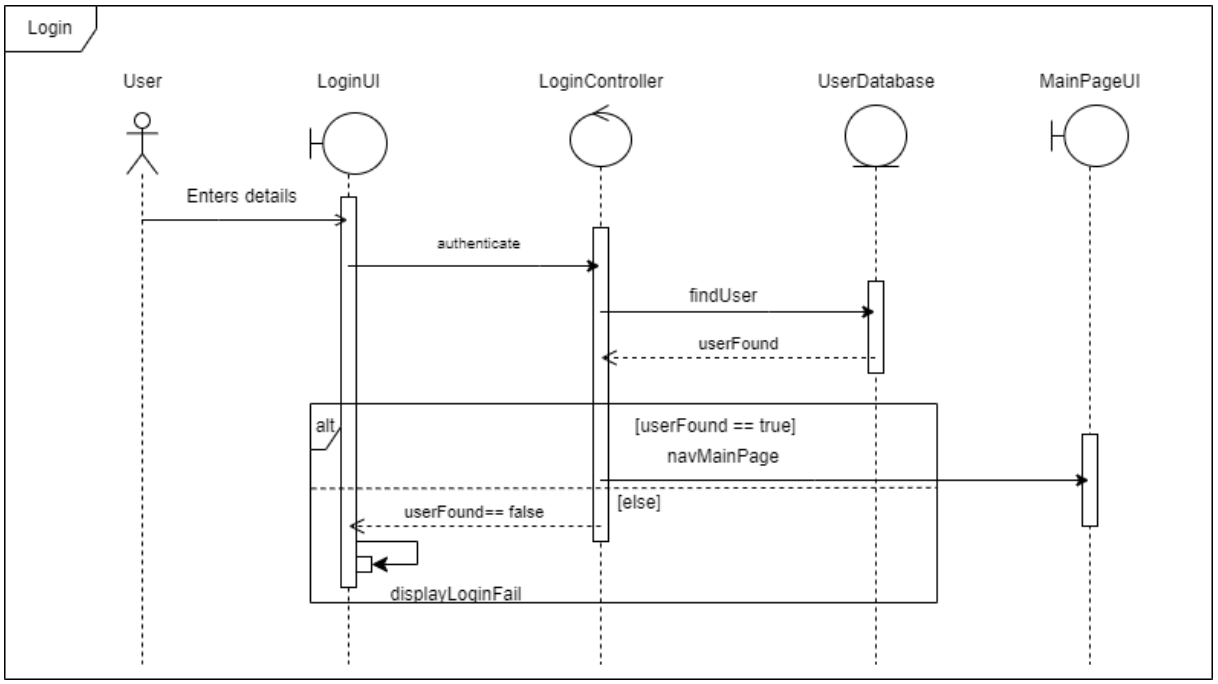


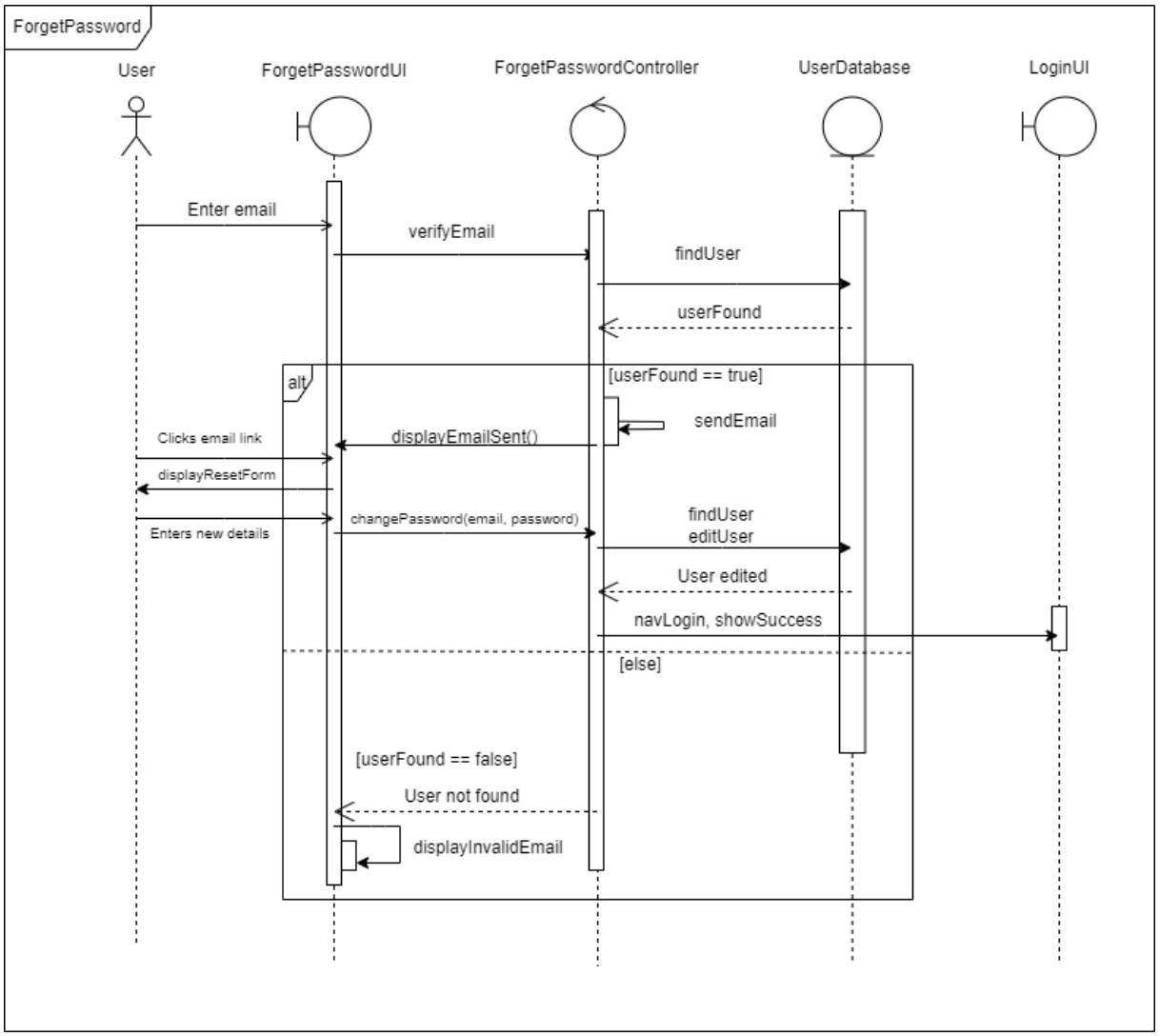
3.3.2 Boundary Classes and Control Classes

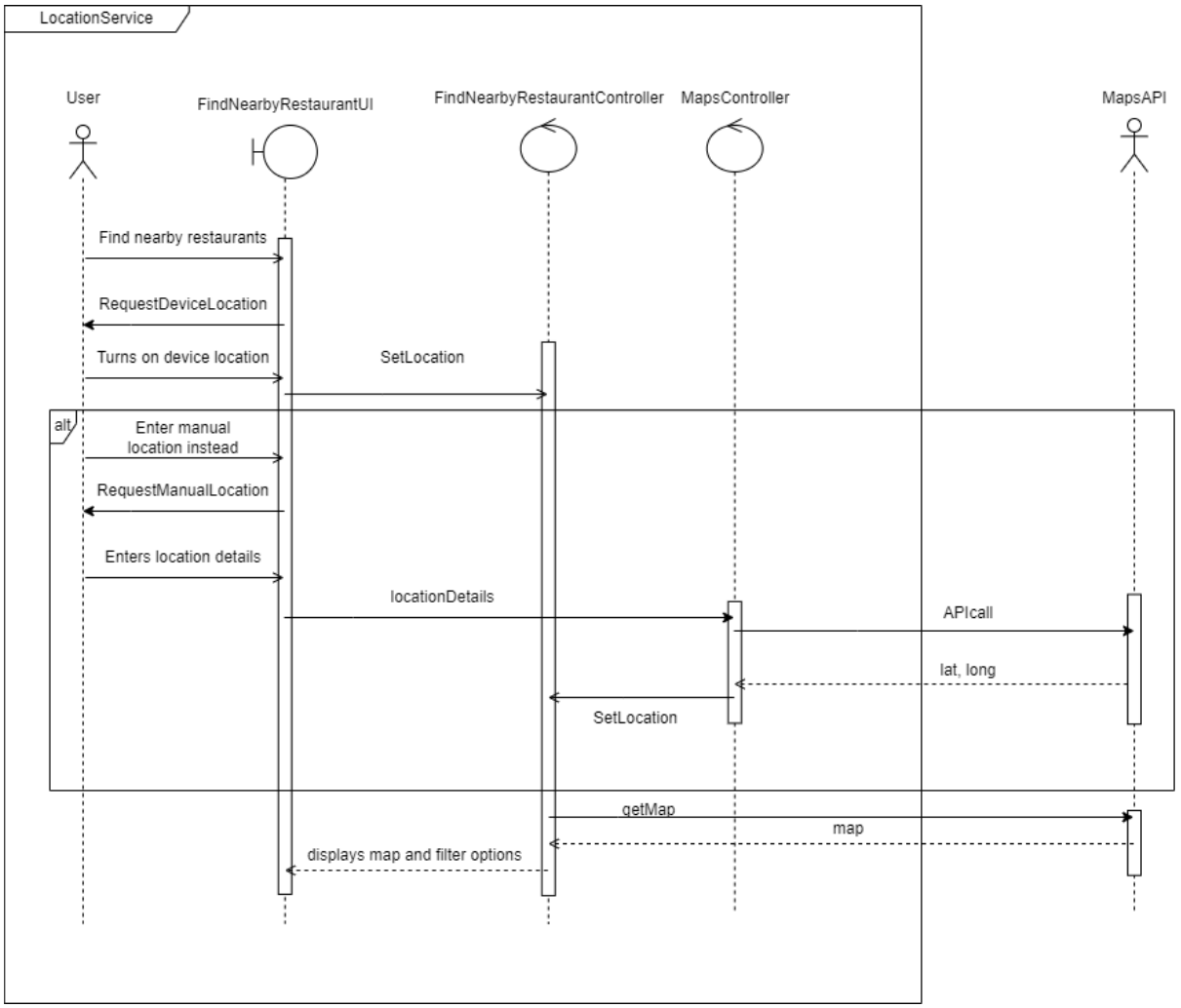


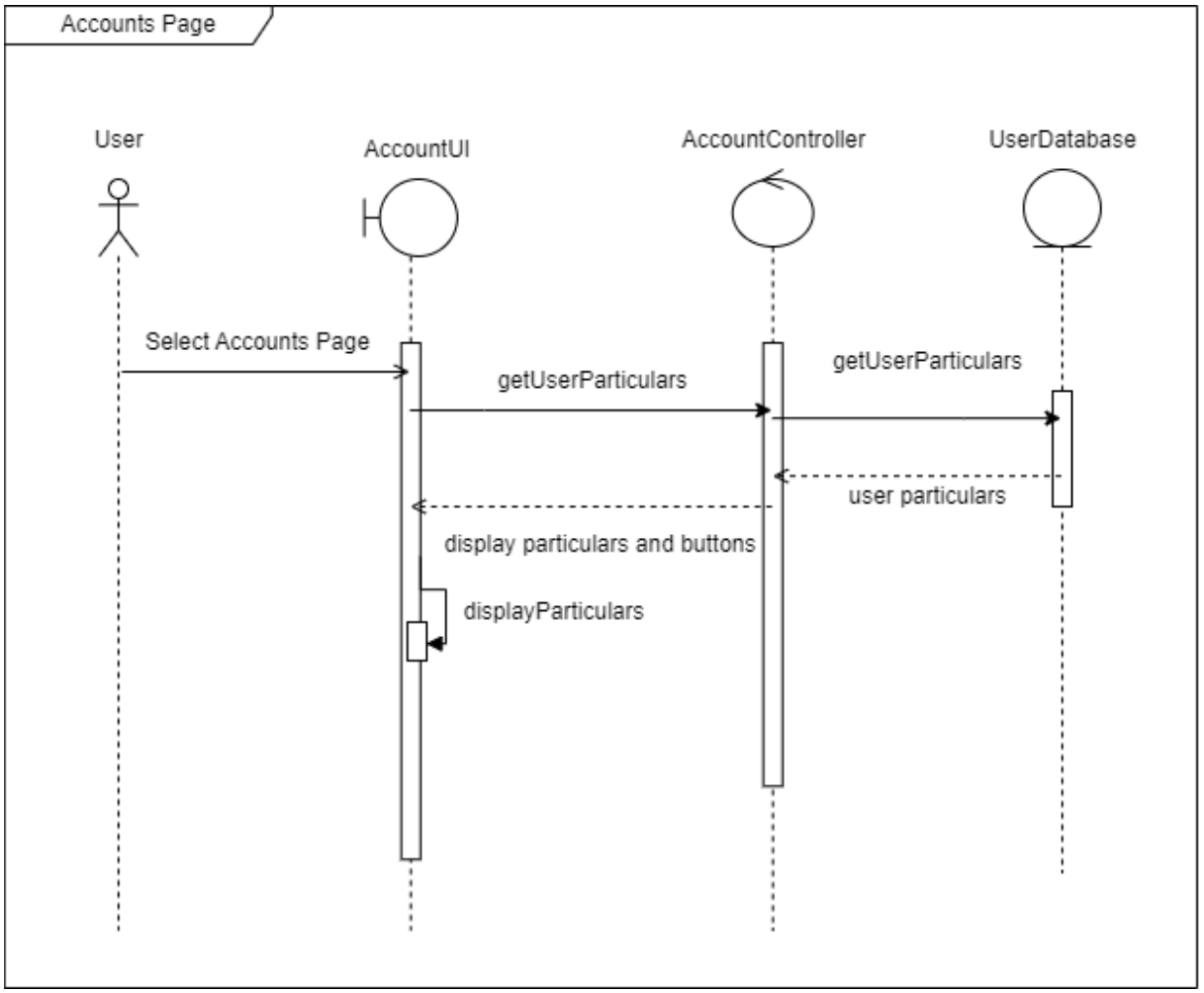
3.4 Sequence Diagrams

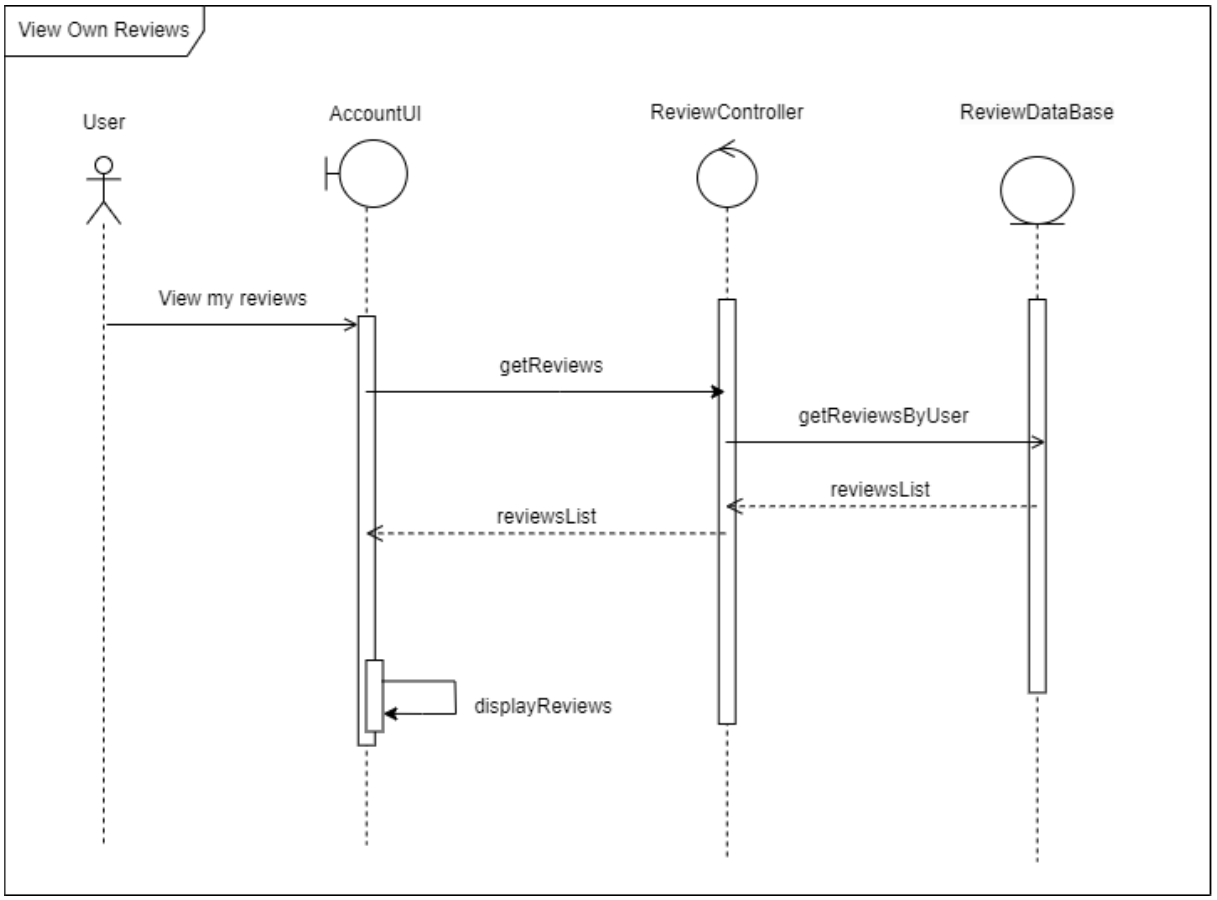


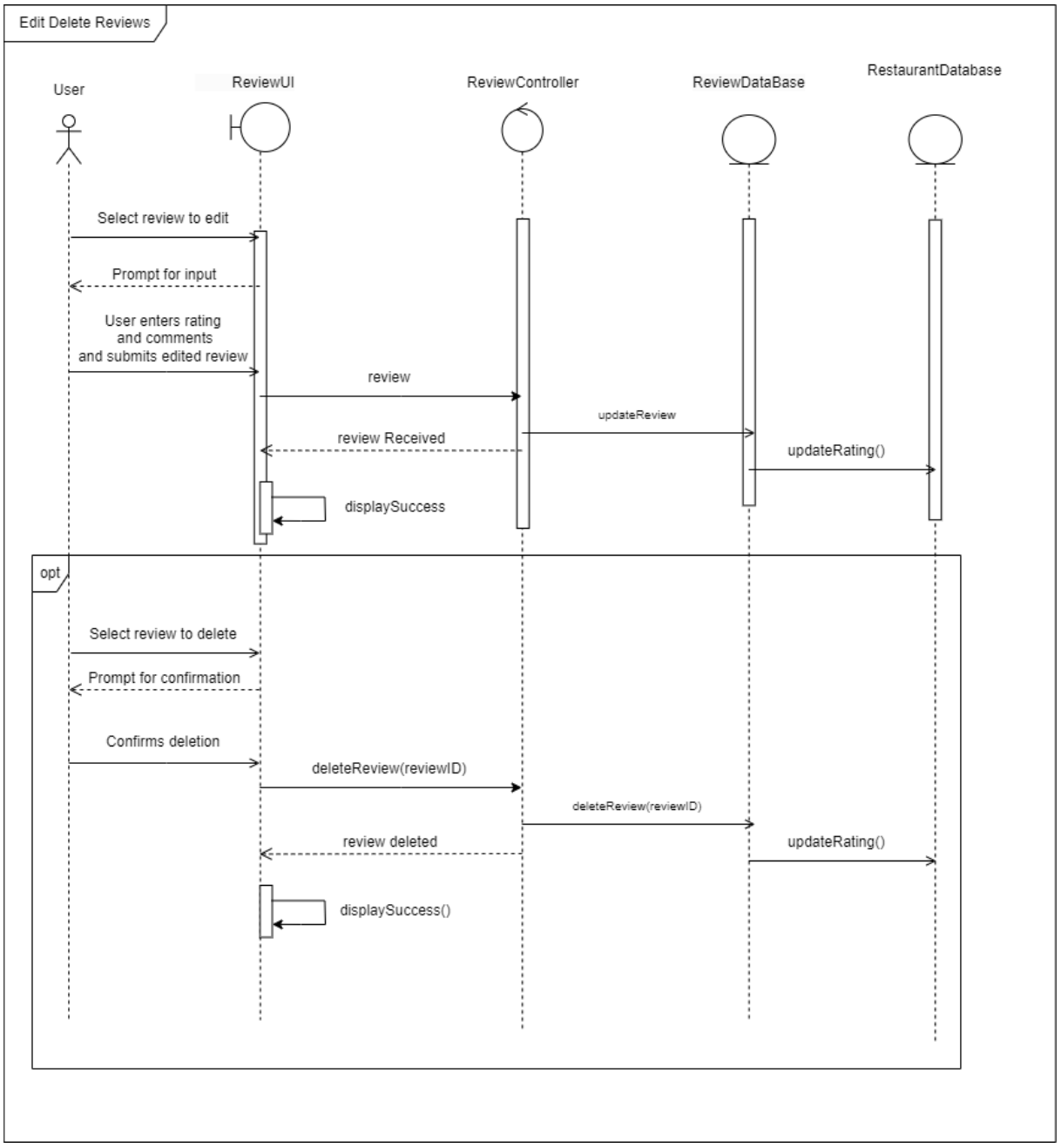

Login
User
LoginUI
LoginController
UserDatabase
MainPageUI
Enters details
authenticate
findUser
userFound
alt
[userFound == true]
navMainPage
[else]
userFound== false
displayLoginFail

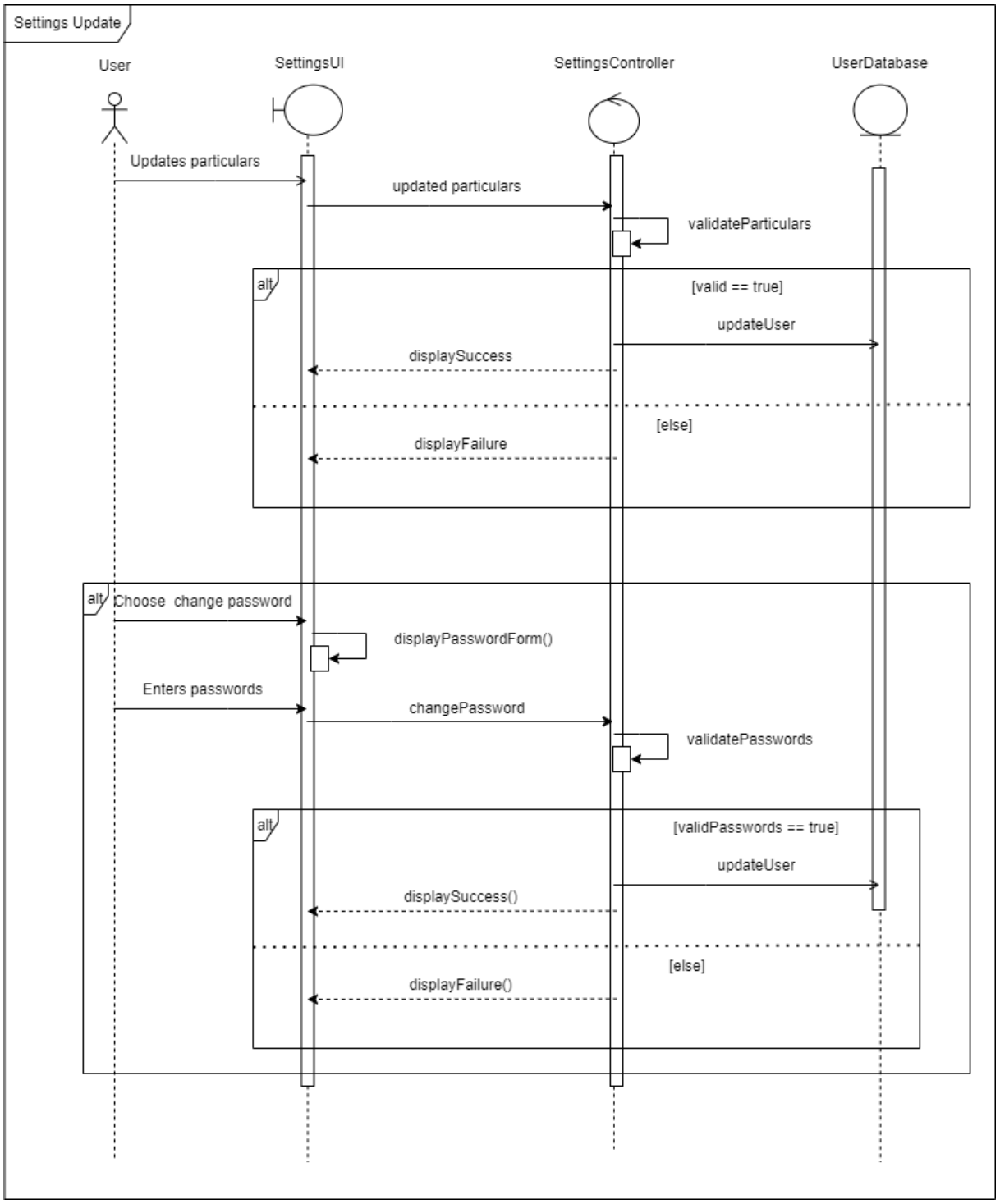

ForgetPassword
User
ForgetPasswordUI
ForgetPasswordController
UserDatabase
LoginUI
Enter email
verifyEmail
findUser
userFound
alt
[userFound == true]
sendEmail
Clicks email link
displayEmailSent()
displayResetForm
changePassword(email, password)
findUser
editUser
Enters new details
User edited
navLogin, showSuccess
[else]
[userFound == false]
User not found
displayInvalidEmail

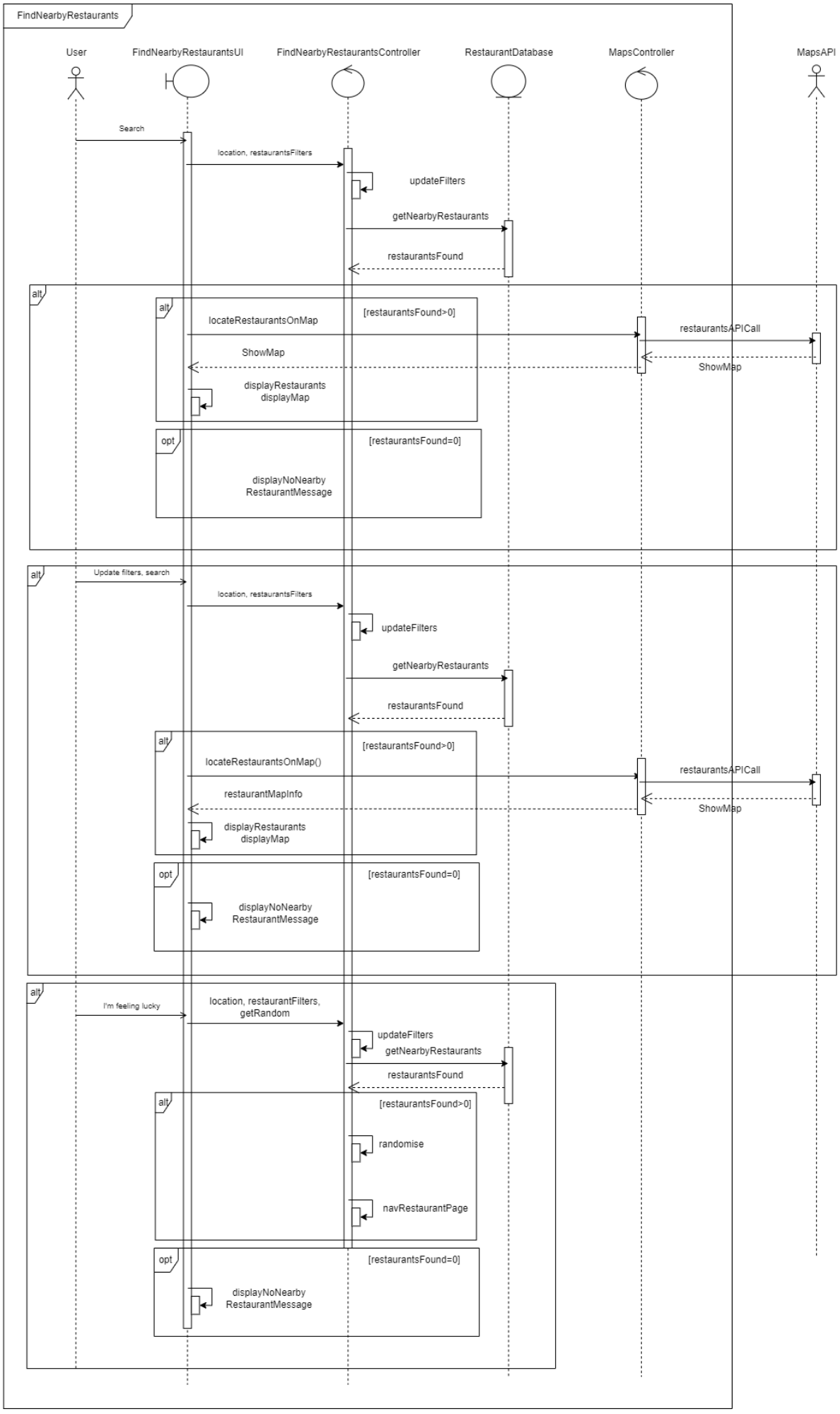

LocationService
User
FindNearbyRestaurantUI
FindNearbyRestaurantController
MapsController
MapsAPI
Find nearby restaurants
RequestDeviceLocation
Turns on device location
SetLocation
alt
Enter manual location instead
RequestManualLocation
Enters location details
locationDetails
APIcall
lat, long
SetLocation
getMap
map
displays map and filter options

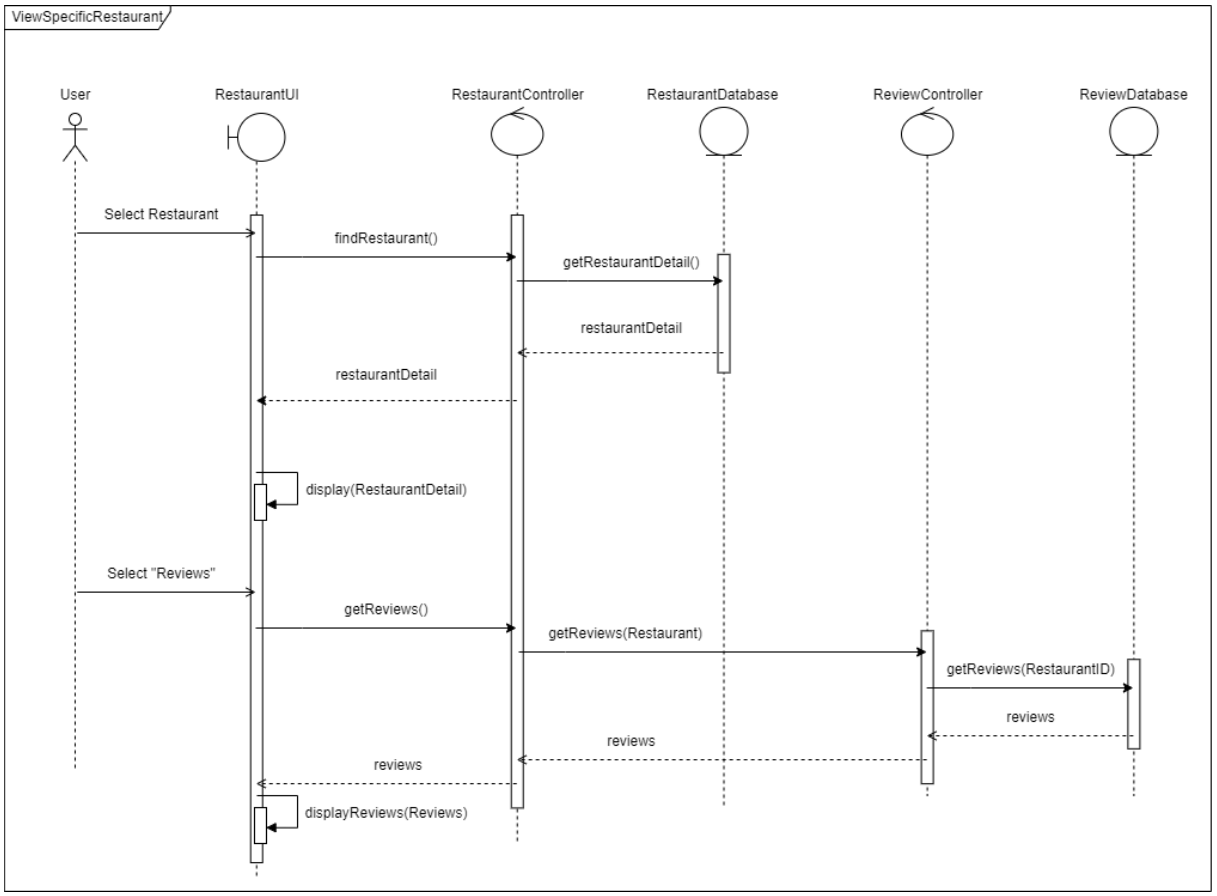

Accounts Page
User
AccountUI
AccountController
UserDatabase
Select Accounts Page
getUserParticulars
getUserParticulars
user particulars
display particulars and buttons
displayParticulars

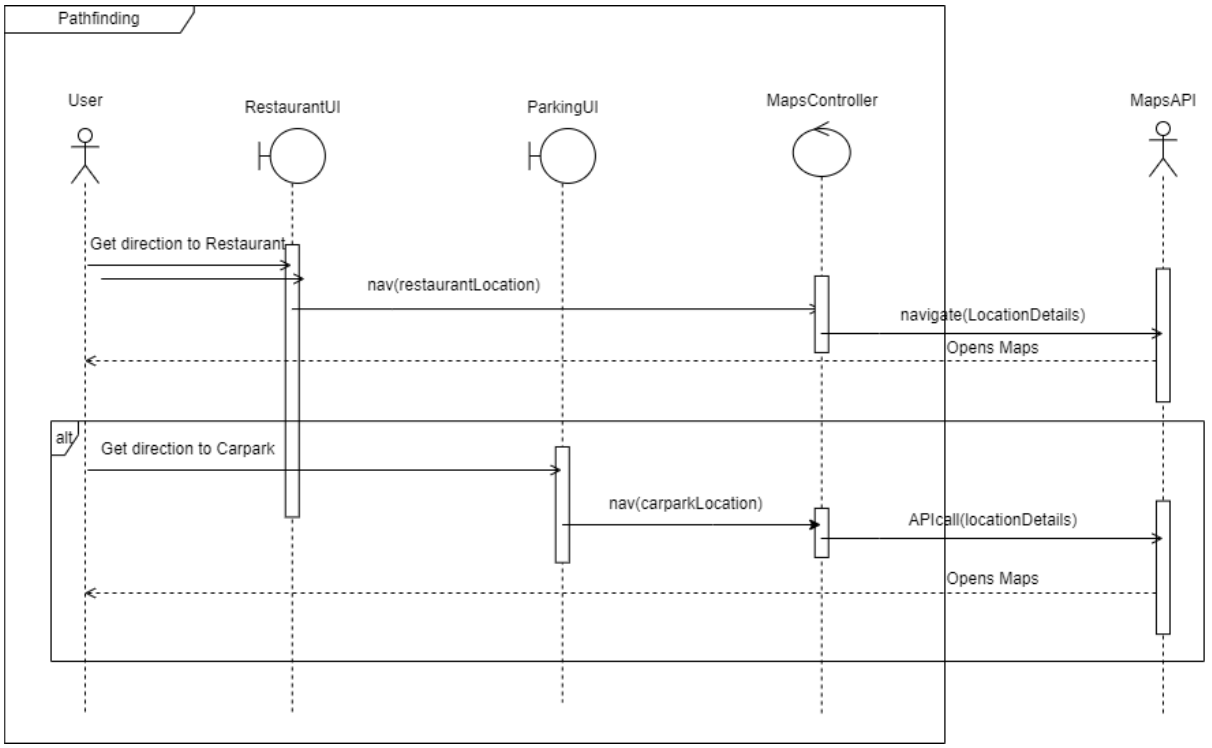

View Own Reviews
User
AccountUI
ReviewController
ReviewDataBase
View my reviews
getReviews
getReviewsByUser
reviewsList
reviewsList
displayReviews

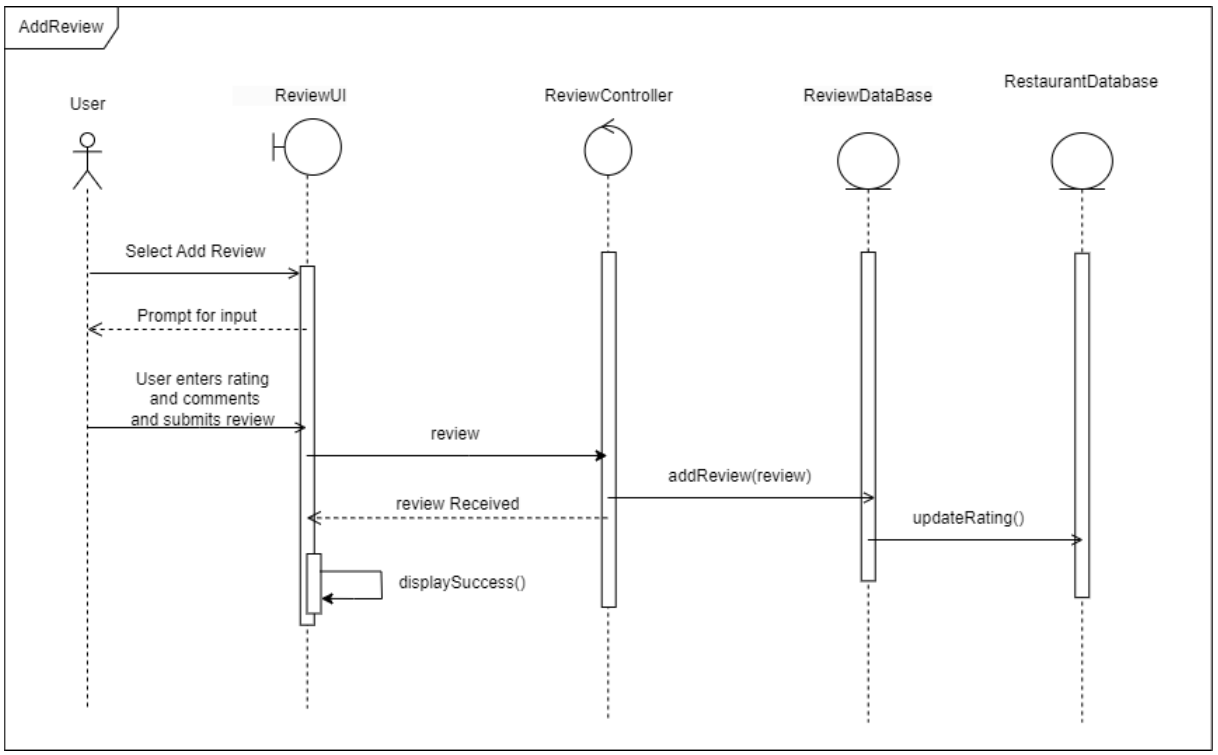

Edit Delete Reviews
User
ReviewUI
ReviewController
ReviewDataBase
RestaurantDatabase
Select review to edit
Prompt for input
User enters rating
and comments
and submits edited review
review
updateReview
review Received
updateRating()
displaySuccess
opt
Select review to delete
Prompt for confirmation
Confirms deletion
deleteReview(reviewID)
deleteReview(reviewID)
review deleted
updateRating()
displaySuccess()

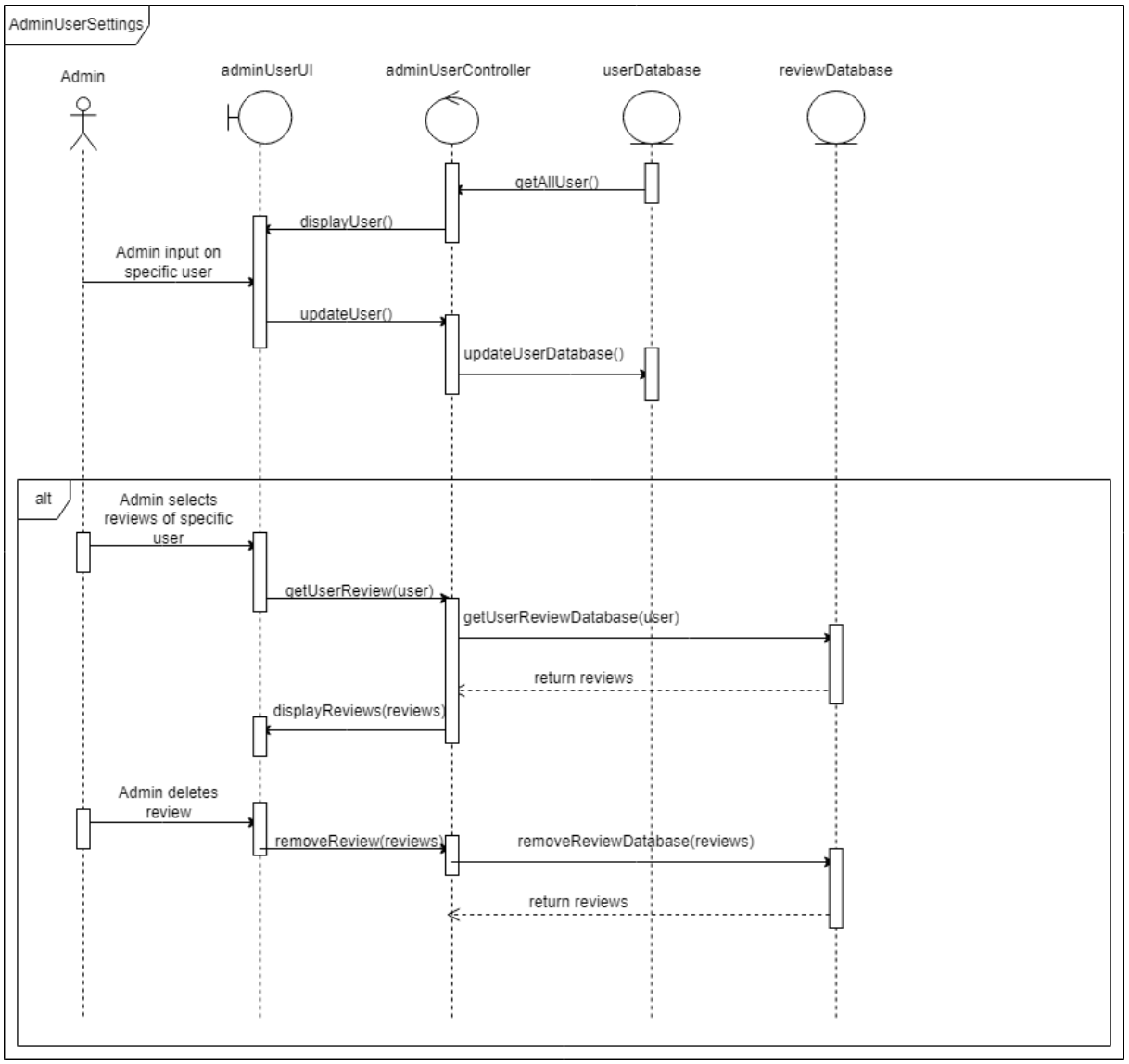

Settings Update
User
SettingsUI
SettingsController
UserDatabase
Updates particulars
updated particulars
validateParticulars
alt
[valid == true]
updateUser
displaySuccess
[else]
displayFailure
alt
Choose change password
displayPasswordForm()
Enters passwords
changePassword
validatePasswords
alt
[validPasswords == true]
updateUser
displaySuccess()
[else]
displayFailure()

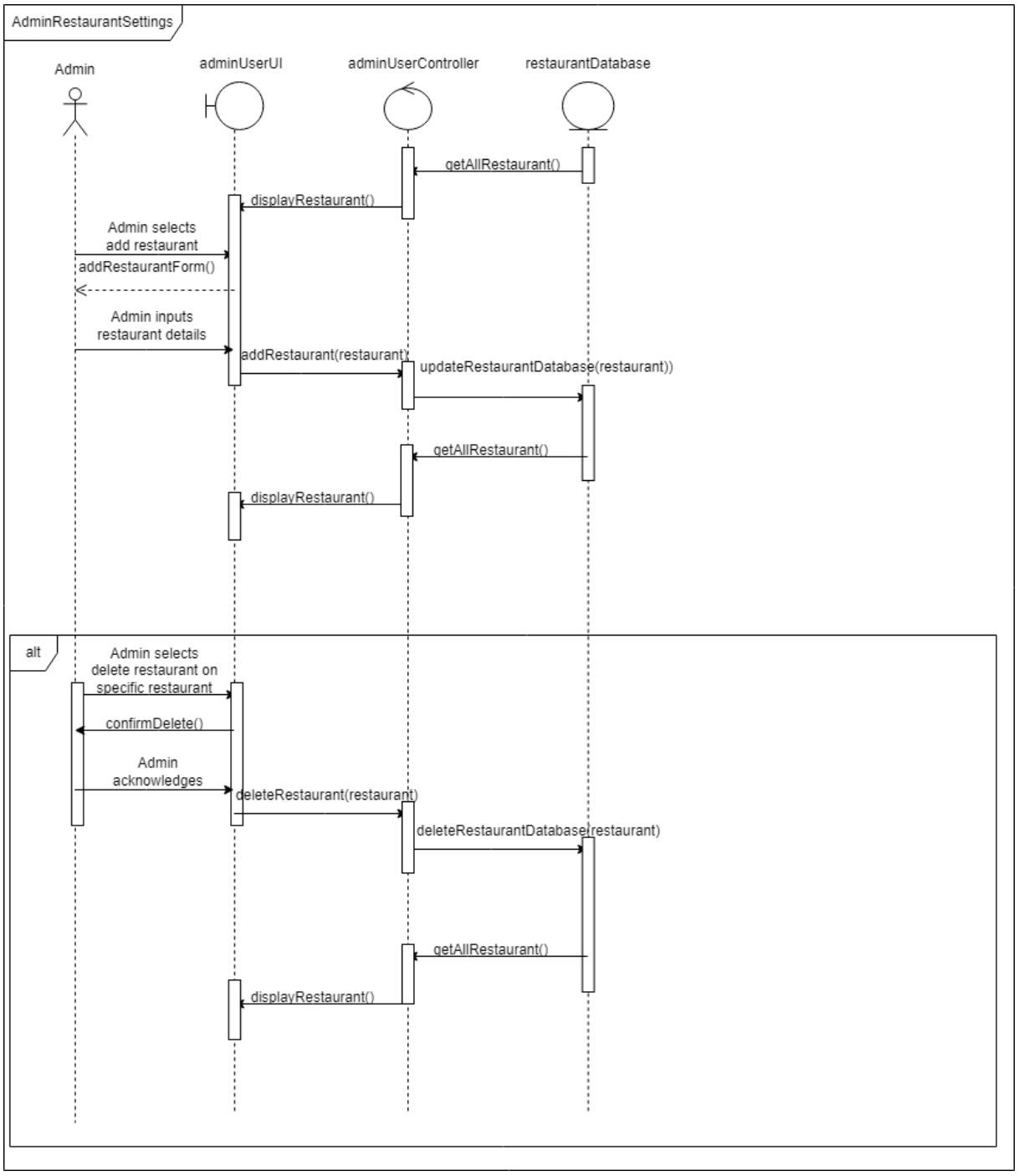

FindNearbyRestaurants
User
FindNearbyRestaurantsUI
FindNearbyRestaurantsController
RestaurantDatabase
MapsController
MapsAPI
Search
location, restaurantsFilters
updateFilters
getNearbyRestaurants
restaurantsFound
alt
alt
[restaurantsFound>0]
locateRestaurantsOnMap
restaurantsAPICall
ShowMap
ShowMap
displayRestaurants
displayMap
opt
[restaurantsFound=0]
displayNoNearby
RestaurantMessage
alt
Update filters, search
location, restaurantsFilters
updateFilters
getNearbyRestaurants
restaurantsFound
alt
[restaurantsFound>0]
locateRestaurantsOnMap()
restaurantsAPICall
restaurantMapInfo
ShowMap
displayRestaurants
displayMap
opt
[restaurantsFound=0]
displayNoNearby
RestaurantMessage
alt
I'm feeling lucky
location, restaurantFilters,
getRandom
updateFilters
getNearbyRestaurants
restaurantsFound
alt
[restaurantsFound>0]
randomise
navRestaurantPage
opt
[restaurantsFound=0]
displayNoNearby
RestaurantMessage


ViewSpecificRestaurant
User
RestaurantUI
RestaurantController
RestaurantDatabase
ReviewController
ReviewDatabase
Select Restaurant
findRestaurant()
getRestaurantDetail()
restaurantDetail
restaurantDetail
display(RestaurantDetail)
Select "Reviews"
getReviews()
getReviews(Restaurant)
getReviews(RestaurantID)
reviews
reviews
reviews
displayReviews(Reviews)

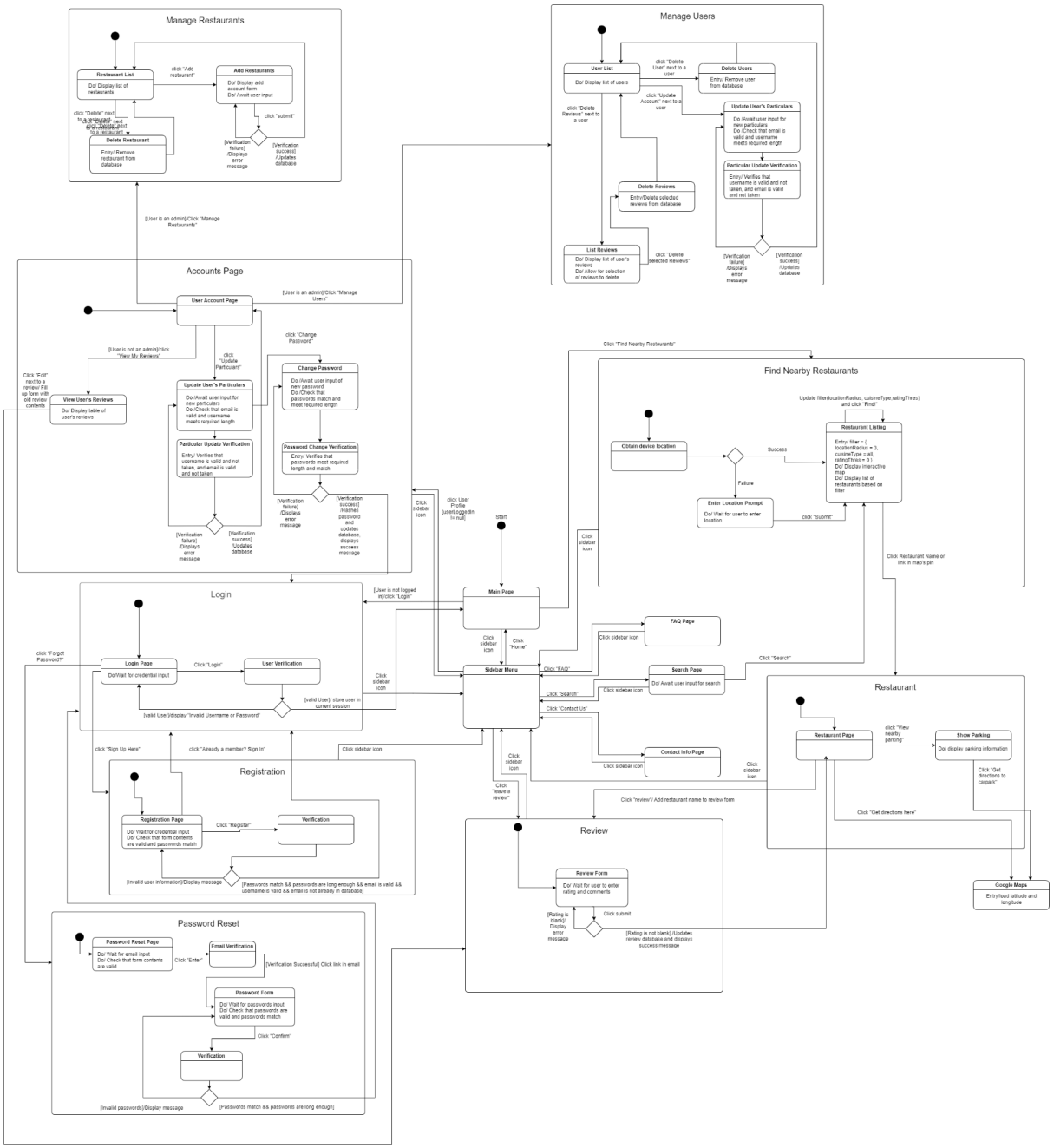

Pathfinding
User
RestaurantUI
ParkingUI
MapsController
MapsAPI
Get direction to Restaurant
nav(restaurantLocation)
navigate(LocationDetails)
Opens Maps
alt
Get direction to Carpark
nav(carparkLocation)
APIcall(locationDetails)
Opens Maps


AddReview
User
ReviewUI
ReviewController
ReviewDataBase
RestaurantDatabase
Select Add Review
Prompt for input
User enters rating
and comments
and submits review
review
addReview(review)
review Received
updateRating()
displaySuccess()


AdminUserSettings
Admin
adminUserUI
adminUserController
userDatabase
reviewDatabase
getAllUser()
displayUser()
Admin input on specific user
updateUser()
updateUserDatabase()
alt
Admin selects reviews of specific user
getUserReview(user)
getUserReviewDatabase(user)
return reviews
displayReviews(reviews)
Admin deletes review
removeReview(reviews)
removeReviewDatabase(reviews)
return reviews

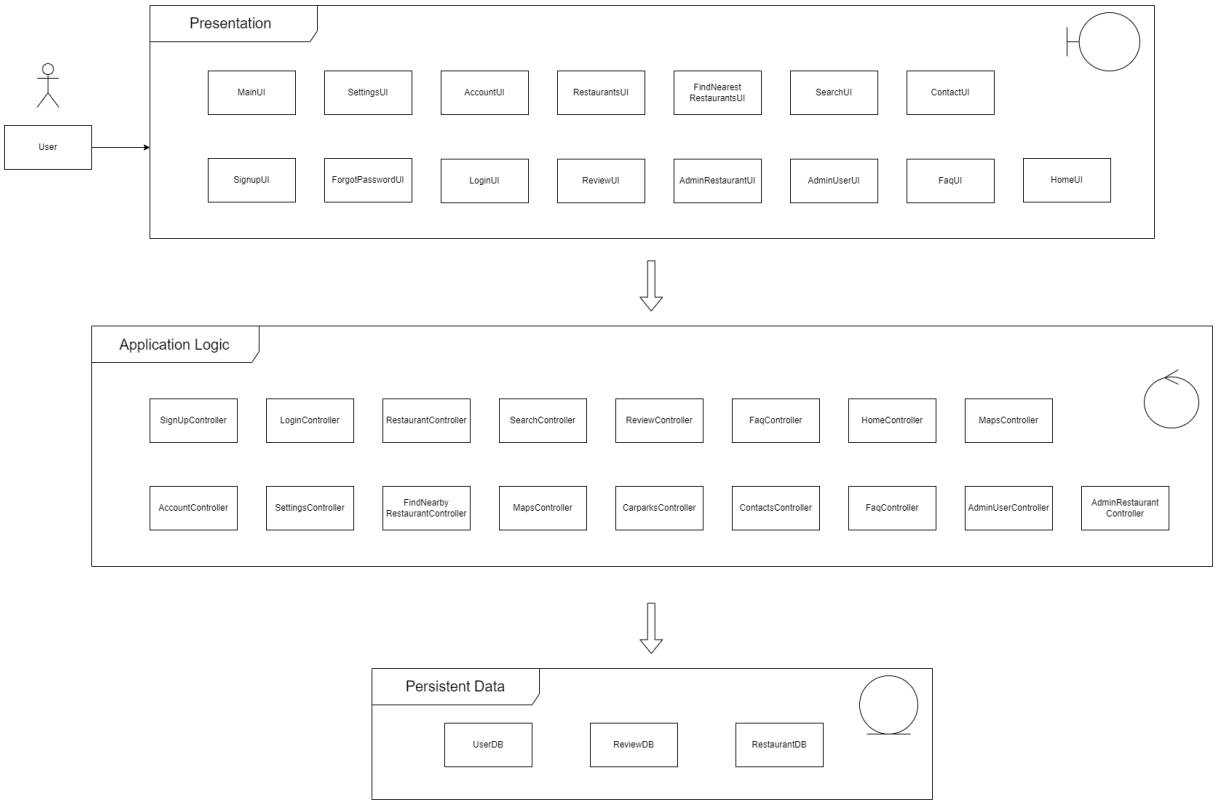

AdminRestaurantSettings
Admin
adminUserUI
adminUserController
restaurantDatabase
getAllRestaurant()
displayRestaurant()
Admin selects
add restaurant
addRestaurantForm()
Admin inputs
restaurant details
addRestaurant(restaurant)
updateRestaurantDatabase(restaurant))
getAllRestaurant()
displayRestaurant()
alt
Admin selects
delete restaurant on
specific restaurant
confirmDelete()
Admin
acknowledges
deleteRestaurant(restaurant)
deleteRestaurantDatabase(restaurant)
getAllRestaurant()
displayRestaurant()

3.5 State Machine Diagram



3.6 Design Model

3.6.1 Software Architecture



We used a 3-layer software architecture for W.G.T, which was implemented using Django's web framework.

The persistent data layer is implemented with a sqlite3 database, and the classes (UserDB, ReviewDB, RestaurantDB) are realized with Django's models.

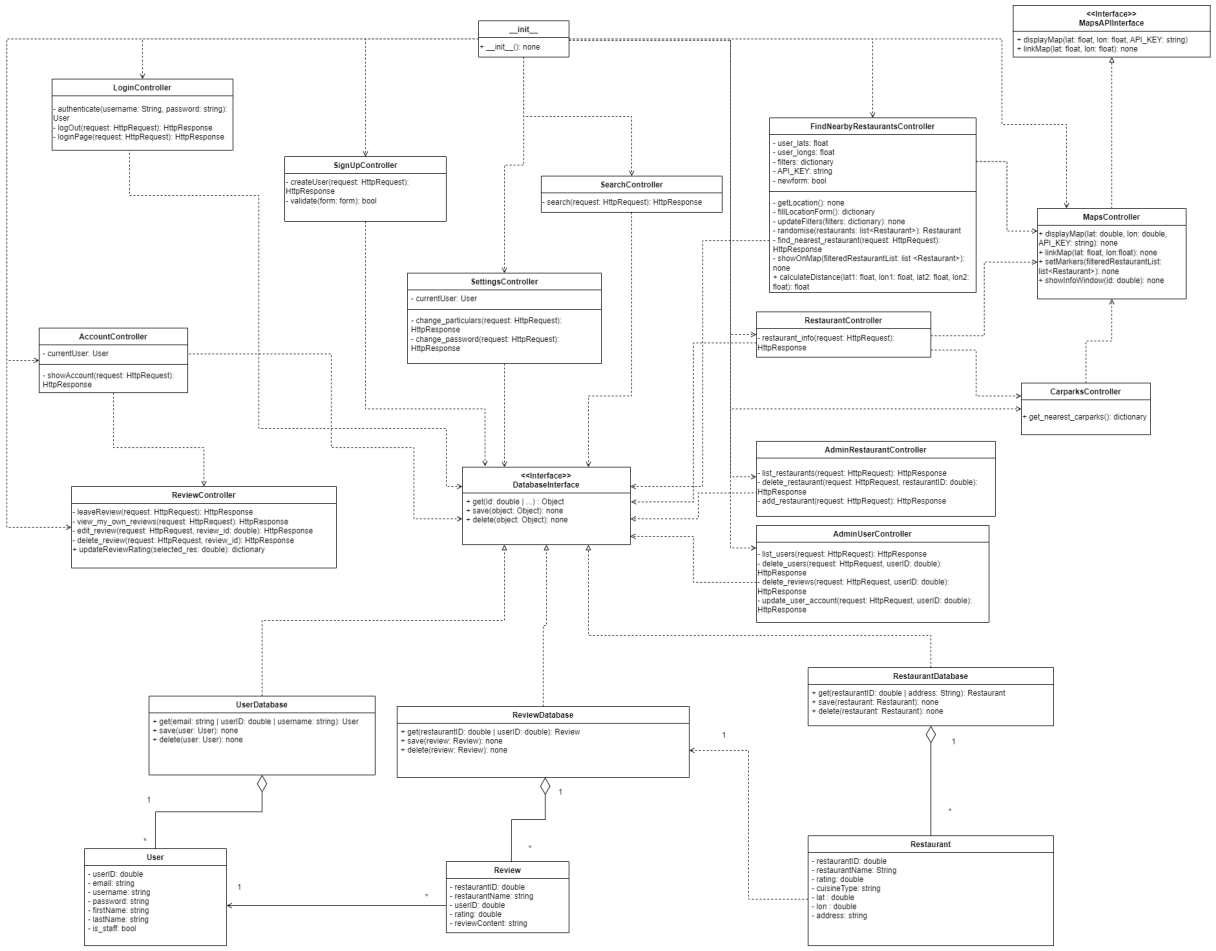
The application logic layer is mostly implemented with Python. Various controllers are used to each implement a specific function.

The presentation layer consists of HTML templates, which are styled with CSS and javascript. The user can only interact with this layer.

User input from the presentation layer is propagated to the controller layer using HTTP Request objects, and the controller layer then accesses or modifies data in the persistent data layer using its own logic.

Since each layer is entirely separate from the others, there is a separation of concerns. There is also reduced dependency between classes, allowing for easier testing and debugging.

3.6.2 Design Class Diagram



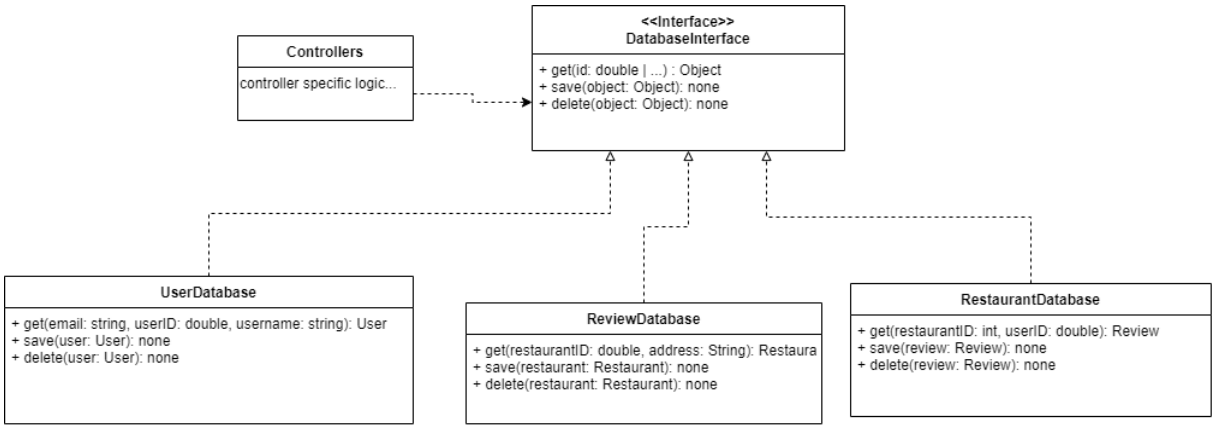
3.6.3 Design Patterns

Strategy Pattern

Problem

In our application, we use different databases that store different types of data, and have various controllers which access and interact with these databases in different ways. In some cases, such as in ReviewController, the controller may access more than one database. Linking the controllers individually to each database would lead to tight coupling, and cause problems during debugging and testing of the code.

Solution



To remedy this problem, we make use of the strategy pattern. We define a unified interface for the various “strategies” that each controller makes use of, which in this context refers to the databases it may access. This decouples the logic of the controller from the various databases, since it makes use of the same generic methods through the interface.

This allows for loose coupling, and also adheres to the open-close principle, allowing for easy extensibility. For example, if we were to implement a future feature of recording users’ past visits to restaurants for the purpose of training an AI-powered recommendation engine, we can easily add the necessary controllers and database through the interface, without having to change previous code.

Command Pattern

Problem

Our app uses many forms, through which various types of user data can be sent: usernames, emails, restaurant filter options, review comments, and so on. It also requests for data: such as when viewing restaurants and reviews. Designing ways to pass these data directly from the UI classes to the controller classes is time consuming and inflexible, especially when accounting for the different cases during runtime, such as erroneous input, for example.

Furthermore, data and input has to move from controller to controller, some data required at different stages of the code. For example, when leaving a review, the information of the last restaurant is needed so that users do not need to look through the drop down list and find the specific restaurant they have in mind.

Solution

```
def searchRestaurant(request):
    res = restaurant.objects.all() #get all restaurant objects in database
    results=None

    if request.GET.get('search'): #get restaurant search
        search = request.GET.get('search')
        results = restaurant.objects.filter(address__contains=search)
    context = {'res': res, 'results' : results}  #pass res into html
    return render(request, 'base/search_restaurant.html', context)
```

All information is encoded in a ‘request’ object. This decouples the interaction between the UI and controller classes: the UI classes can be configured to simply send the requested data to the request object, which the controller then handles according to its own logic.

```
request.session['selected_res'] = {
    'id': selected_res.id,
    'name': selected_res.name,
    'address': selected_res.address,
    'restaurant_rating' : selected_res.restaurant_rating,
    'lat': selected_res.lat,
    'lon': selected_res.lon,
    'cuisine': selected_res.cuisine
}
```

This request object can also store temporary data, which every controller can retrieve to interact with and store or update any attributes for other controllers to make use of. This decouples the controllers from each other as well. In this case, the storing of the selected restaurant in request.session allows us to decouple the interaction between RestaurantController and ReviewController in selecting which restaurant to pass to the review form.

3.6.4 Design Principles

Open-Closed Principle
Code is allowed to be extended for more functionality but closed to modification. Future improvements to the program can be added to the code without modifying the current code, such as by adding new templates and their corresponding controllers.

Interface Segregation Principle
Each controller only imports from libraries and methods it requires to function, and does not import additional methods it does not need. This reduces dependency across the various classes, allowing for loose coupling.

Modularity
There is separation of concerns between classes: each class is modular and carries out a specialized function.

Abstraction
Data passed into controllers does not need to know the implementation details, just that it has passed data required to that controller. This helps to reduce complexity and increases reusability. On top of that, code in controllers can be modified without affecting the user.

Reusability
Functions from a controller can be reused to help logic in other controllers by importing those methods. The use of a “main” HTML template also allows for consistent styling across the various UI classes without reusing code.

3.7 Software Engineering Practices

We utilized the scrum method to develop our project. Every week, we laid out what tasks were outstanding and what could be realistically accomplished during that week. Tasks are then completed throughout the week. Since everyone was busy with other modules and projects, daily meetings were unfeasible. Instead we communicated on Telegram regularly tasks completed or any issues we ran into during development. At the end of the week, we met on Zoom to review the sprint and product completed so far, and reflect on any difficulties during this sprint. Afterwards, we planned the next sprint for the next week.

For effective collaboration and version control, we used GitHub to manage our code properly. Each member maintained their own branch from main, coding and pushing changes to their branch. Tested work is then merged with main to prevent any overwriting of other people's work.

4. Testing Documentation

This documentation explains the various activities performed as part of the Testing of the ‘WGT’ application.

4.1 Test Plan

Multiple Testing will be implemented to ensure functionality

4.1.1 Types of testing performed

Smoke Testing is conducted upon receiving a deployed build in the Test environment to ensure that the primary features are functioning correctly, and the build can be accepted for further testing.

System Integration Testing verifies that the entire application is functioning as per the specified requirements, with critical business scenarios tested to ensure that essential functionality works as intended and without errors.

Regression Testing is performed after deploying a new build for testing, including any defect fixes or enhancements. This type of testing is done on the entire application and not only on the newly added functionality or defect fixes. It ensures that the existing functionality continues to work correctly after the addition of new features and fixes. The test cases for new functionality are added to the existing test cases and executed.

4.2 Test Cases and Test Results

4.2.1 Black Box Test Cases

---

1. Login
    a. Generic cases

Test Id	Scenario	Expected Result	Actual Result
1.	Login with valid account username and password.	The system displays the main menu for user to continue the operation	The system displays the main menu for user to continue the operation
2.	Login without valid credentials.	The system prompts the user to enter the credentials again.	The system prompts the user to enter the credentials again.
3.	Login without filling up the required fields	The system prompts the user to fill up the required details for logging in	The system prompts the user to fill up the required details for logging in

b. Specific cases (Combination)

Username	Password	Expected Result	Actual Result
testuser	password1234	Successful login	Successful login
wronguser	password1234	Invalid email/password	Invalid email/password
Empty(“”)	password1234	Please fill in all required fields	Please fill in all required fields
testuser	Empty(“”)	Please fill in all required fields	Please fill in all required fields
testuser	wrongpass	Invalid email/password	Invalid email/password

2. Registration
    a. Generic Cases

Test Id	Scenario	Expected Result	Actual Result
1.	Register with valid account username and password	The system displays the main menu for user to continue the operation	The system displays the main menu for user to continue the operation
2.	Register with incomplete fields	The system prompts the user to full up the required fields for registration	The system prompts the user to full up the required fields for registration
3.	Register with password mismatch	The system prompts the user to re-enter their password	The system prompts the user to re-enter their password

    b. Specific cases (Email Address)

Test Id	Email Address	Expected Result	Actual Result
1	testuser@gmail.com	**Approve**	**Approve**
2	testuser	**Reject**	**Reject**
3	test@login	**Reject**	**Reject**

c. Specific Cases (Combination)

Email Address	Username	First Name	Last Name	Password	Confirm Password	Expected Result	Actual Result
testuser@gmail.com	testuser	test	testing	testpass	testpass	Created new user	Created new user
Empty(“”)	testuser	test	testing	testpass	testpass	Please fill in all required fields	Please fill in all required fields
testuser@gmail.com	Empty(“”)	test	testing	testpass	testpass	Please fill in all required fields	Please fill in all required fields
testuser@gmail.com	testuser	Empty(“”)	testing	testpass	testpass	Please fill in all required fields	Please fill in all required fields
testuser@gmail.com	testuser	test	Empty(“”)	testpass	testpass	Please fill in all required fields	Please fill in all required fields
testuser@gmail.com	testuser	test	testing	Empty(“”)	testpass	Please fill in all required fields	Please fill in all required fields
testuser@gmail.com	testuser	test	testing	testpass	Empty(“”)	Please fill in all required fields	Please fill in all required fields
existinguser@gmail.com	testuser	test	testing	testpass	testpass	Email has been used	Email has been used
testuser@gmail.com	existinguser	test	testing	testpass	testpass	Username has been used	Username has been used
testuser@gmail.com	testuser	test	testing	testpass1	testpass2	Entered passwords do not match	Entered passwords do not match
testuser@gmail.com	testuser	test	testing	testpass2	testpass1	Entered passwords do not match	Entered passwords do not match

3. Get user's current location
    a. Display user's geo location **(Valid Inputs)**

Test Input Latitude and Longitude		**Expected Output**	**Actual Output**
1.350460	103.681040	50 Nanyang Walk	50 Nanyang Walk
1.302145	103.901533	39 Amber Road	39 Amber Road
1.312578	103.938321	26 Bayshore Road	26 Bayshore Road
1.296980	103.888289	55 Meyer Road	55 Meyer Road

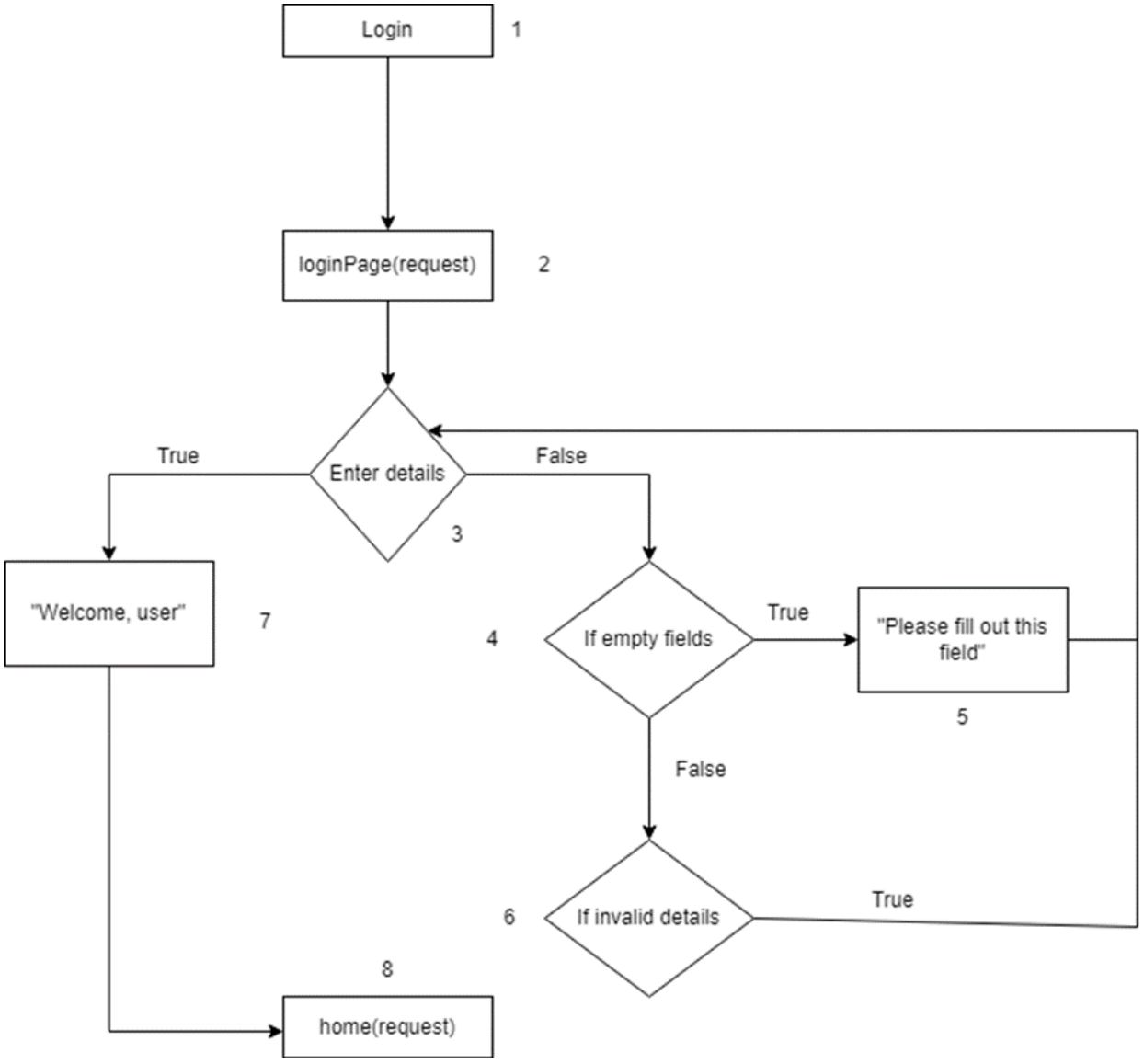
    b. Display user's geo location **(Invalid Inputs)**

Test Input	**Expected Result**	**Actual Result**
Location service not allowed	Permission denied	Permission denied
Mobile GPS not activated	Position unavailable	Position unavailable
Unknown user coordinates	Unknown error	Unknown error

4.2.2 White Box Test Cases

---

1) User Login



Basis path

1. 1-2-3-7-8
2. 1-2-3-4-5-7-8
3. 1-2-3-4-6-7-8

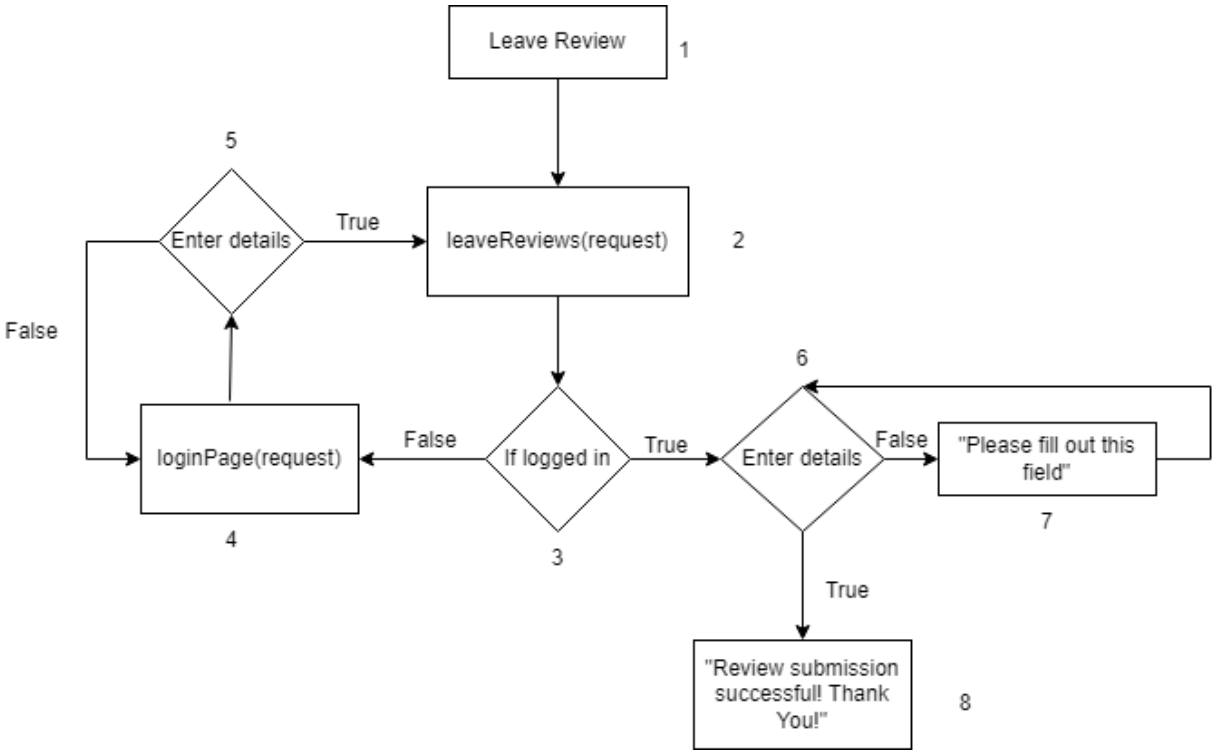
Generic Test Case

1. Enter Login page
   Enter correct login details and click login

2. Enter Login page
   Enter nothing and click login
   Enter correct login details and click login

3. Enter Login page
   Enter incorrect login details and click login
   Enter correct login details and click login

2) Leave Review



Basis path

1. 1-2-3-6-8
2. 1-2-3-4-5-2-3-6-8
3. 1-2-3-4-5-4-5-2-3-6-8
4. 1-2-3-6-7-6-8

Generic test cases

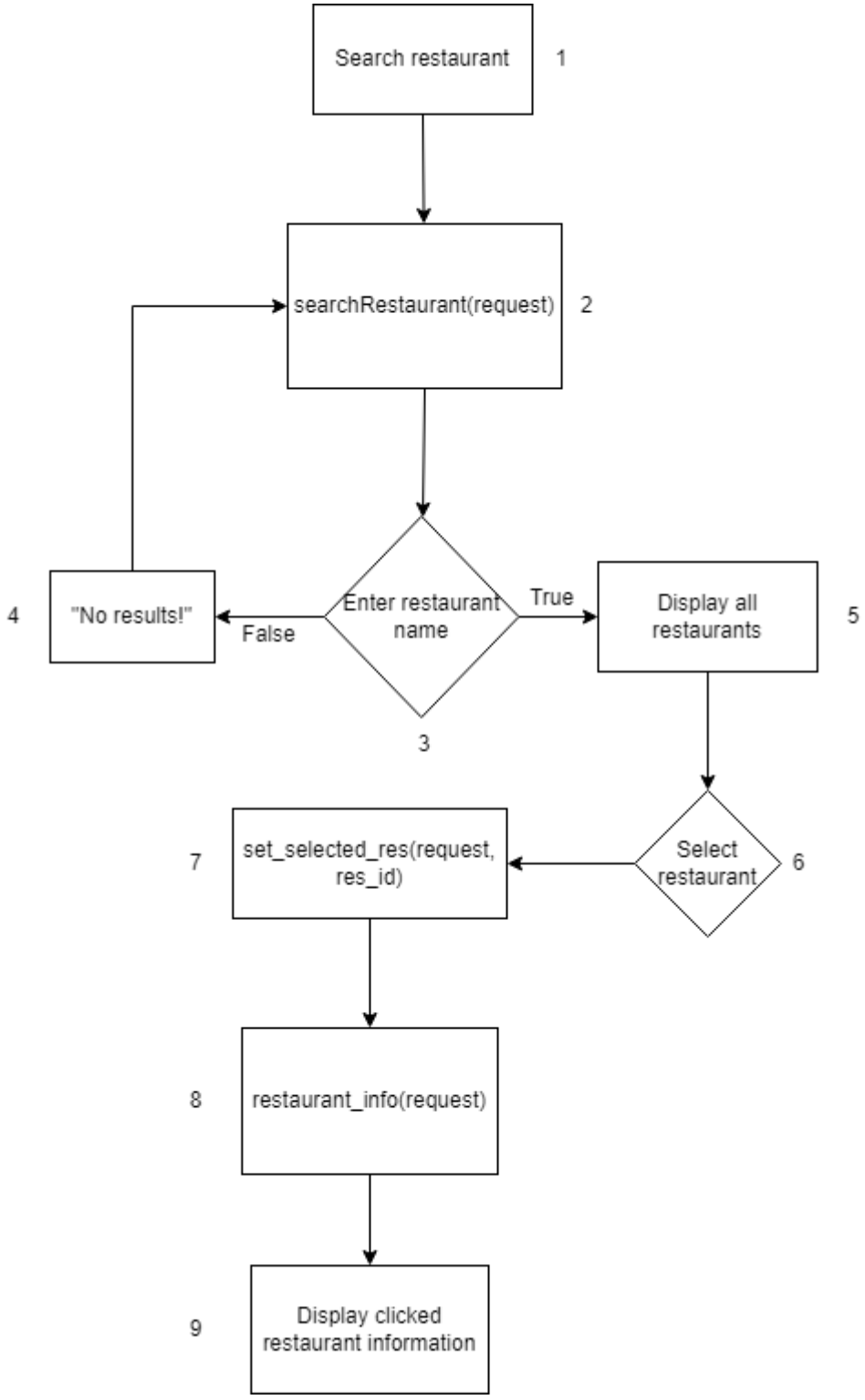
1. Be logged in
   Fill in all fields in review page

2. Not be logged in
   Enter valid user credentials
   Fill in all fields in review page

3. Not be logged in
   Enter invalid user credentials
   Enter valid user credentials
   Fill in all fields in review page

4. Be logged in
   Leave review text blank
   Fill in all fields in review page

3) Search Restaurant



Basis path

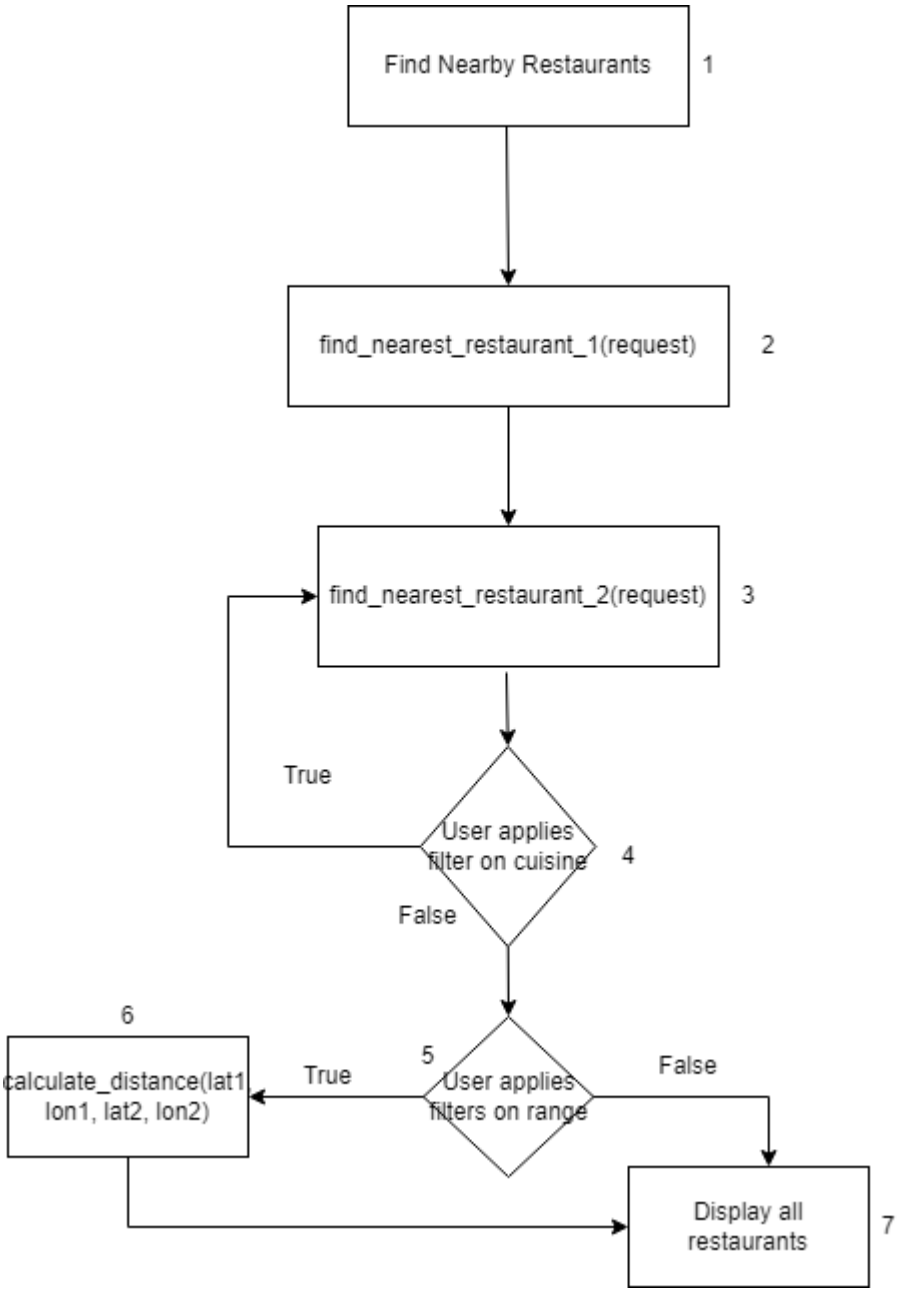
1. 1-2-3-5-6-7-8-9
2. 1-2-3-4-2-3-5-6-7-8-9

Generic test cases

1. Enter valid restaurant name
   Click on restaurant in link

2. Enter invalid restaurant name
   Enter valid restaurant name
   Click on restaurant in link

4) Find Nearby Restaurants



Basis path

1) 1-2-3-4-5-7
2) 1-2-3-4-3-5-7
3) 1-2-3-4-5-6

Generic Test Cases

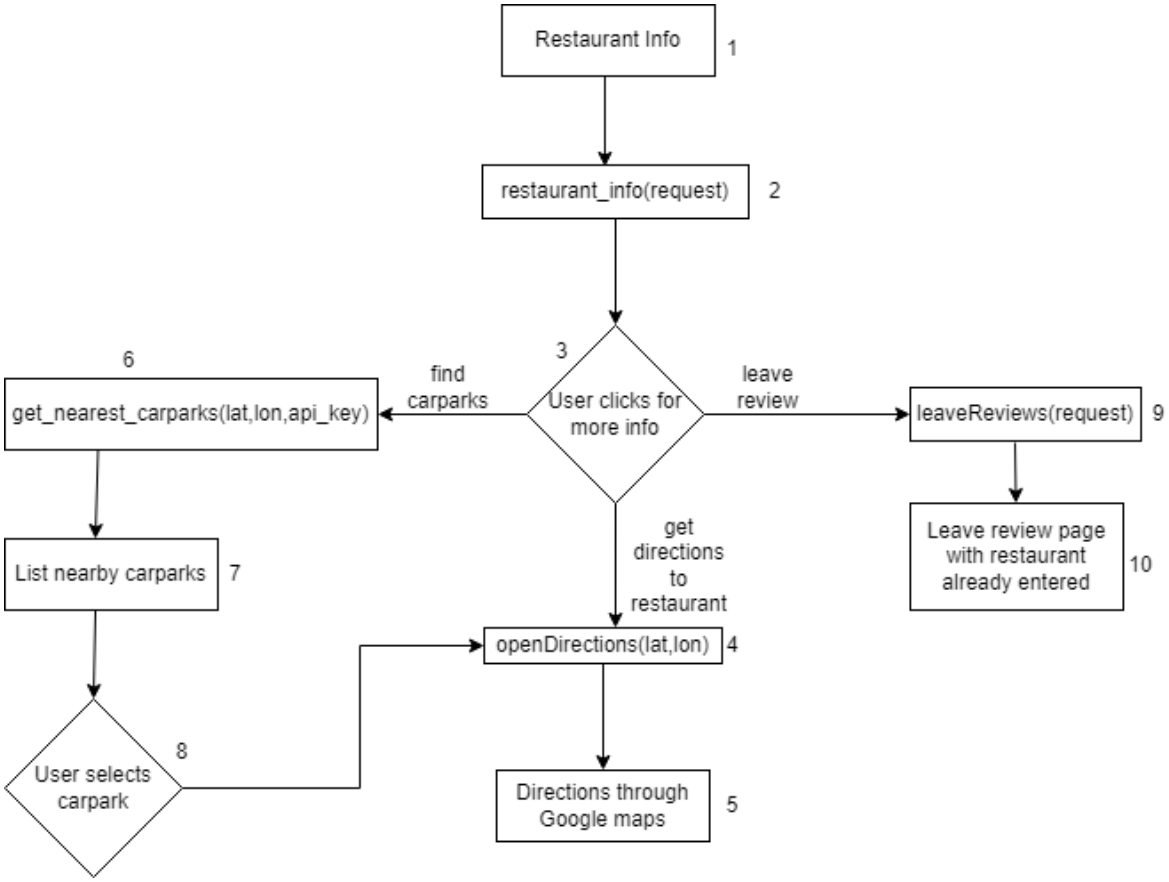
1) Enter find nearest restaurant page
   Share or enter location
   Click on search
   All restaurants displayed

2) Enter find nearest restaurant page

Share or enter location
Select cuisine filters
Click on search
All restaurants within constraint displayed

3) Enter find nearest restaurant page
Share or enter location
Click on search
Select range limitation
All restaurants within constraint displayed

5) Restaurant info



Basis path

1) 1-2-3-4-5
2) 1-2-3-6-7-8-4-5
3) 1-2-3-9-10

Generic Test Case

1) User selects specific restaurant
System renders restaurant information
User selects “get directions here”
System redirects them to google maps with coordinates inputted

2) User selects specific restaurant
   System renders restaurant information
   User selects “view carparks”
   System renders top 5 nearest carparks
   User selects a carpark in the list
   System redirects them to google maps with coordinates inputted

3) User selects specific restaurant
   System renders restaurant information
   User selects “leave a review”
   System redirects them to leave a review page with restaurant already inputted

5. Appendices

W.G.T Demonstration of features video https://youtu.be/b2Kk5jqNsd0
W.G.T Demonstration of test cases video https://youtu.be/JtLx_-BizDQ