School of Computer Science & Engineering

CZ2006 Software Engineering

Project Name – SeeFOOD

SSP1/Group **1**

Team NTU SE-A+

Team Members:

Alvin Lee Yong Teck (Leader) (U1620768F)

Htet Naing (U1620683D)

Liu Jihao, Bryan (U1621401C)

Lim Kian Hock Bryan (U1620949L)

Johnathan Wong Wei Hong (U1622446E)

Lok Li Xin (U1622435K)

Contents

1. Product Description

1.1 Purpose

Our SeeFOOD app aims to facilitate the searching of nearby restaurants based on the GPS location of an Android mobile device. In addition, what makes our app unique is being able to retrieve nearby car park information given a particular restaurant.

Effectively, SeeFOOD serves two main purposes. They are as follows:

(1) The first purpose is to allow users to find nearby restaurants based on their mobile device's GPS location. By using our app, users will be able to find nearby restaurants ranging from fast foods to popular restaurants in a split of a second.

(2) The second purpose is to enable users to locate nearby carparks based on the proximity of a particular restaurant. Due to this feature, the required car park information will be retrieved and presented to our users in an efficient manner.

1.2 Scope

Our SeeFOOD app allows users to search nearby restaurants and also to access details of each restaurant merely by clicking on a restaurant that they are interested. Users will be able to find carparks that are located within the area of an interested restaurant and access their information such as carpark rates. Users can create their own user account with free-of-charge to unlock the favourite feature which allows them to bookmark their favourite restaurants.

1.3 Users and stakeholders

The stakeholders of this project consist of food restaurants, Android mobile users and SeeFOOD. Information of food restaurants will be collected through the application with the assistance of Google Places API. Android mobile users will utilize the application to search nearby restaurants and locate close by carparks of those restaurants. SeeFOOD strives to bring food restaurants closer to Android mobile users by providing a smooth and seamless platform for digital interaction between restaurants and mobile users as well as time and energy saving for our users to access carpark information.

1.4 Assumptions and constraints

Users should have a GPS-enabled Android device capable of running the application.

Users should have Internet access in order to use the application.

1.5 Constraints

Our SeeFOOD app currently supports Android OS running Android Lollipop (API level 21) and above.

As data.gov.sg do not provide all the car park rates and details resulting in limited search capability of listing all available local car parks.

The application is currently available only in English.

1.6 Initial UI Mockups

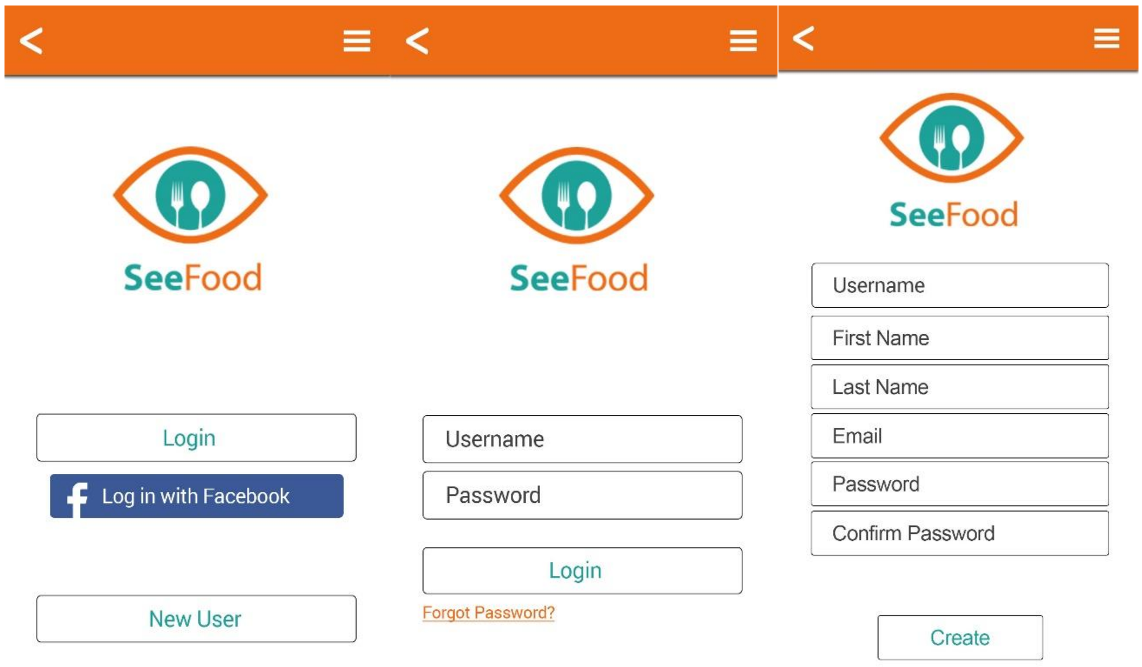


Figure 1: Login & Registration Screens

Figure 1 shows the user interface for login and registration. The user interface layout is kept simple and is designed to be similar to the login and registration screens used for other applications. This will enable the user to register for a new account as well as to login to the application with relative ease.

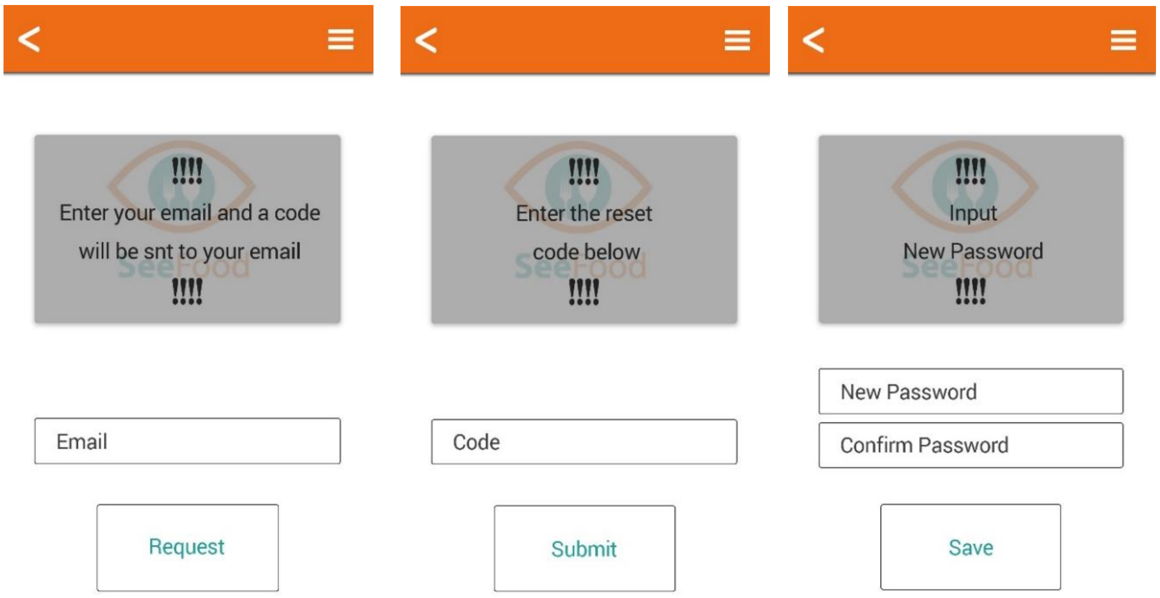


Figure 2: Forget Password Screens

Figure 2 shows the user interface for forget password. The forget password page will send a code to the user's email once the user requests for it. The subsequent page will prompt for the code that was sent to the user. Once the code has been verified, the page prompting for the new password will appear and the user will be able to set a new password for his/her account.

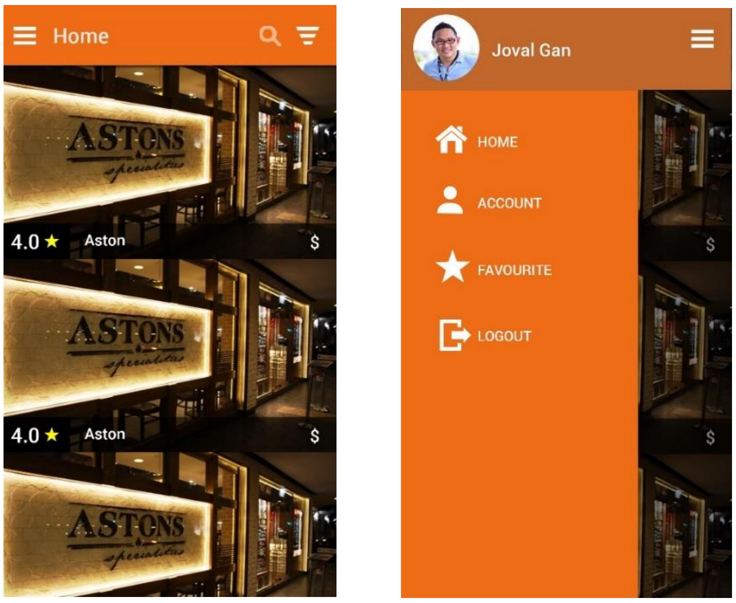


Figure 3: Home Screens

Figure 3 shows the user interface for the home page. The menu bar will show various options depending if the user is a guest or a logged in user. The home page will have an image slide show of the range of restaurants. The featured panel will show the promotions that certain restaurants is having together with its end date. The trending panel will show the restaurants that are currently being heavily searched by the users.

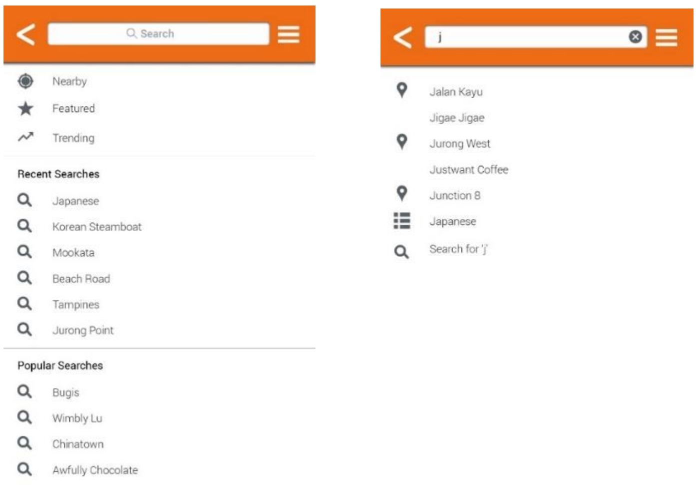


Figure 4: Search Screens

Figure 4 shows the user interface for searching. The search page will show options such as "nearby" which shows nearby restaurants, "featured" which shows featured restaurants, "trending" which shoes restaurants that are trending. The search page also shows the recent searches that the user searched previously. The search page will show various suggestions depending on the popular searches in the application. The searching example shows an example of how the screen will be shown should the user start to type in his/her search.

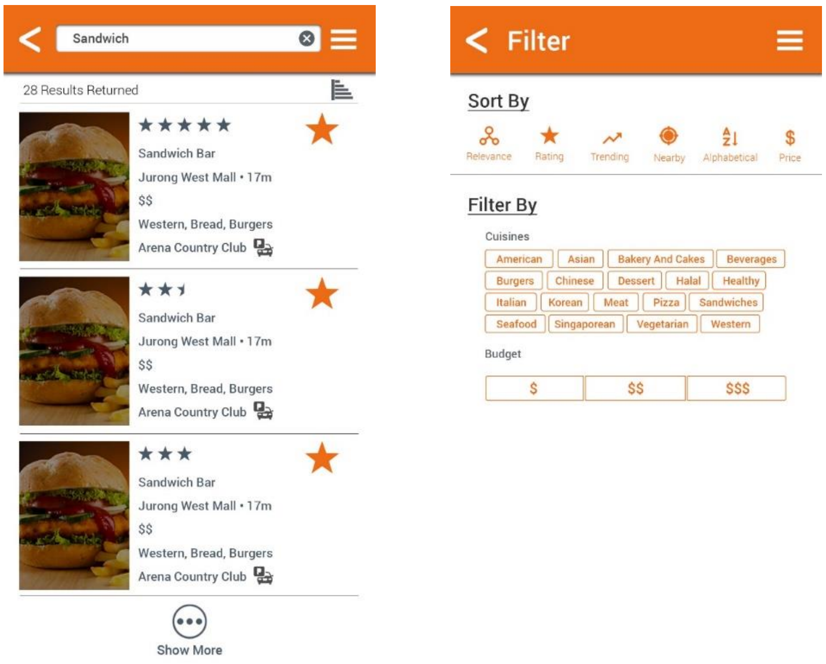
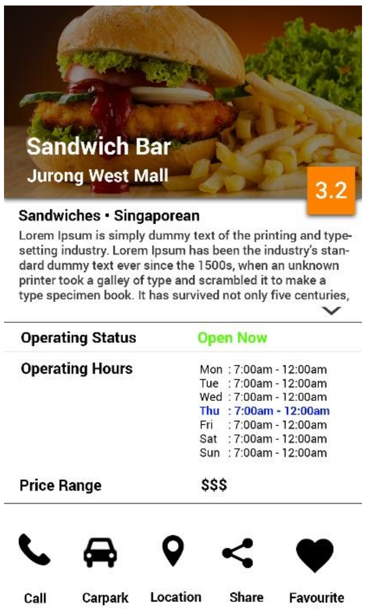


Figure 5: Search Results/Favourites & Sorting Screens

Figure 5 shows the user interface for the search results and favourites. The search results will return the relevant favourites first should the search be related. Favourites will return all the user's favourites and it will be sorted based on the most recent favourite. The search results and favourites will only display 5 results with the option to allow the user to show more results via a button. There is an option that will allow the user to sort the results based on the selected filter in the sorting screen.


Sandwich Bar
Jurong West Mall
3.2
Sandwiches • Singaporean
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries,

Operating Status	Open Now
Operating Hours	Mon : 7:00am - 12:00am Tue : 7:00am - 12:00am Wed : 7:00am - 12:00am Thu : 7:00am - 12:00am Fri : 7:00am - 12:00am Sat : 7:00am - 12:00am Sun : 7:00am - 12:00am
Price Range	$$$

Figure 6: Individual Restaurant Screens

Figure 6 shows the user interface for the individual restaurant. The screen will show the various restaurant information such as its rating, location, location on the map, opening hours, price range, phone number and website. The screen also shows the nearest carparks with the option to expand to see more about the carpark prices. There is also an option to show more carparks should the user wishes to view more carparks in the vicinity.

2. Functional Requirements

1. Main Menu

1.1. The system must be able to display the list of recommended restaurants.
1.2. The system must allow user to use the search function.
1.3. The system must allow user to register their account.
1.4. The system must allow user to login to their account.
1.5. The user must be able to select and retrieve more details of a particular restaurant.
  1.5.1. Information include:
    1.5.1.1. Name
    1.5.1.2. Popularity rating
    1.5.1.3. Type of cuisine
    1.5.1.4. Price rate
    1.5.1.5. Restaurant Location
    1.5.1.6. Restaurant distance from current location

2. Search

2.1. The user must be able to search for a variety of restaurants based on filters.
  2.1.1. Filters include:
    2.1.1.1. Name
    2.1.1.2. Popularity rating
    2.1.1.3. Type of cuisine
    2.1.1.4. Price rate
    2.1.1.5. Restaurant Location
    2.1.1.6. Restaurant distance from current location
2.2. The system must be able to auto-complete the user’s input within the search function.
2.3. The system must be able to display information of filtered restaurants.

3. Registration

3.1. The user must be able to register for a new account via our system registration.
  3.1.1. Information to include:
    3.1.1.1. Name
    3.1.1.2. Valid Email Address
    3.1.1.3. Password & Confirm Password
3.2. The user must be able to register for a new account via their Facebook account.
3.3. The system must validate all required fields have been filled up.
3.4. The system must validate the account availability.

4. Login

4.1. The user must be able to login using their email address and password.
4.2. The system must validate that both fields have been filled up.
4.3. The system must validate the email address and password are correct.
4.4. If the login information is incorrect, the system will display an error message.
4.5. The system will redirect the user to the application’s main menu.

4.6. The user must be able to request for change of password should he forget his password.

4.7. The system must be able to send email verification code upon user's request for change of password.

5. Favourites

5.1. The user must login first in order to use the favourites feature.

5.1.1. The user must be able to bookmark their favourite restaurants.

5.1.2. The user must be able to retrieve their list of favourite restaurants.

5.1.3. The user must be able to remove their favourite restaurants.

6. Retrieve Carpark Information

6.1. The user must be able to retrieve nearby carparks information.

6.1.1. Information include:

6.1.1.1. Carpark rates

6.1.1.2. Carpark location

6.1.1.3. Carpark distance from current location

6.1.1.4. Carpark distance from a particular restaurant

6.2. The user must be able to access information for alternative carparks in the vicinity of a particular carpark.

7. Interface with other systems

7.1. The system must be able to retrieve location from user's device via Geolocation API.

7.2. The system must be able to retrieve the list of restaurant's information from Google Places API.

7.3. The system must be able to retrieve the list of carpark information from Data.gov.sg.

8. Formats for Information to be processed

8.1. Popularity rating

8.1.1. Rating must be from 1-star to 5-stars and display average in one decimal places of accuracy (e.g. 4.7 stars)

8.2. Price rate

8.2.1. Price format must be in "S$" and two decimal places of accuracy (e.g. S$1.00)

8.3. Location

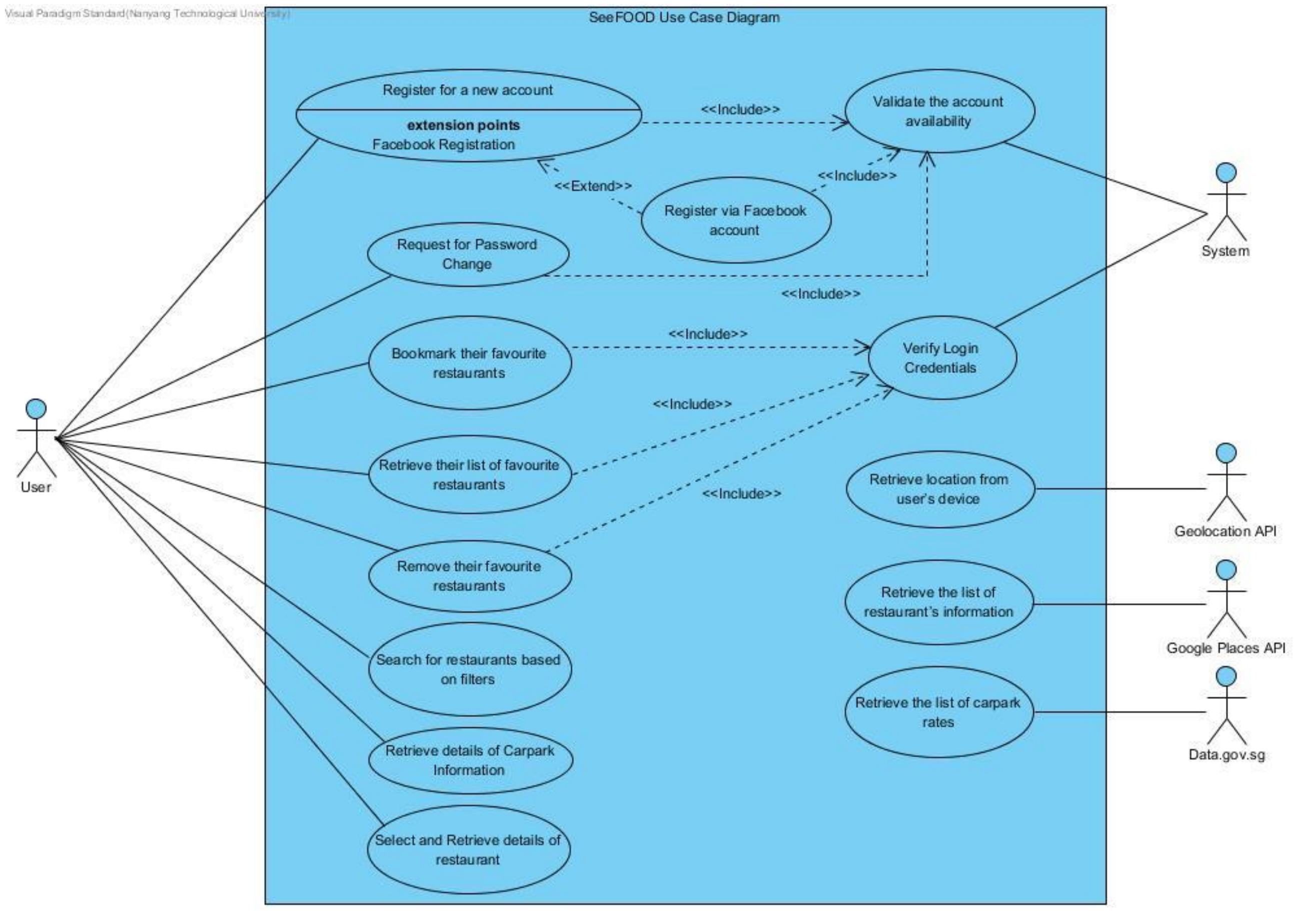
8.3.1. Distance format must be in "km" and two decimal places of accuracy (e.g. 2.05km) if the distance >= 1000m.

8.3.2. Distance format must be in "m" if the distance < 1000m.

8.4. Carpark rates

1. Time format must be in "hr" and "min" (First 1hr: S$2.00, subsequent 30mins: S$0.50)

2.1 Use Case Diagrams


Visual Paradigm Standard(Nanyang Technological University)
SeeFOOD Use Case Diagram
Register for a new account
extension points
Facebook Registration
<<Include>>
Validate the account availability
<<Extend>>
Register via Facebook account
<<Include>>
Request for Password Change
<<Include>>
System
Bookmark their favourite restaurants
<<Include>>
Verify Login Credentials
<<Include>>
User
Retrieve their list of favourite restaurants
<<Include>>
Retrieve location from user's device
Geolocation API
Remove their favourite restaurants
Retrieve the list of restaurant's information
Google Places API
Search for restaurants based on filters
Retrieve the list of carpark rates
Data.gov.sg
Retrieve details of Carpark Information
Select and Retrieve details of restaurant

2.2 Use Case Descriptions

Use Case ID:	1		
Use Case Name:	Verify Login Credentials		
Created By:	Htet Naing	**Last Updated By:**	Li Xin
Date Created:	6th September 2017	**Date Last Updated:**	6th November 2017

Actor:	System
Description:	Account authentication process through username and password.
Preconditions:	1. User account must already exist in the database. 2. Mobile must be connected to WiFi/Mobile Data.
Postconditions:	1. User is able to bookmark their favourite restaurants. 2. User is able to retrieve their list of favourite restaurants. 3. User is able to remove their favourite restaurants.
Priority:	Medium
Frequency of Use:	1 - 3 times per lifetime
Flow of Events:	1. User tap login via username. 2. User enters their credentials in the login interface. 3. User select the login button. 4. System validates the account by checking the user's credentials with the database. 5. System authenticates the user to login successfully.
Alternative Flows:	AF-S1: User tap login via Facebook account. 1. Return to Step 4. AF-S3: System detects empty username or password fields. 1. System displays error message "Username or password fields cannot be empty." 2. User filled up the required field(s) for username and password. 3. User select the login button to re-attempt login again. 4. Return to Step 2. AF-S4: User enters the wrong credentials 1. System displays error message "Invalid username or password. Please re-enter." 2. Return to Step 2.
Exception:	EX-AF-S1: User not logged in to their Facebook account. 1. System will prompt the user to enter their credentials to login to their Facebook account.
Includes:	1. Validate the account availability
Special Requirements:	-
Assumptions:	- Database can be referred to both System's database and Facebook's database.
Notes and Issues:	-

Use Case ID:	2		
Use Case Name:	Register for a new account		
Created By:	Alvin Lee	**Last Updated By:**	Bryan Liu
Date Created:	6th September 2017	**Date Last Updated:**	2nd November 2017

Actor:	User
Description:	Registration for a new user account.
Preconditions:	1. User account must not already exist in the database. 2. Mobile must be connected to WiFi/Mobile Data.
Postconditions:	1. A verification code will be sent to the user's registered email address.
Priority:	Medium
Frequency of Use:	1 - 3 times per lifetime
Flow of Events:	1. User enters a valid username, email address, first name, last name, password and confirm password fields. 2. User select the register button. 3. System validates the password and confirm password fields are identical. 4. System will send verification code to the registered email address.
Alternative Flows:	AF-S3: System detects mismatch between the password and confirm password. 1. System displays error message "Password and Confirm Password mismatch. Please re-enter." 2. Return to Step 1.
Exceptions:	-
Includes:	1. Validate the account availability.
Special Requirements:	-
Assumptions:	- Database can be referred to both System's database and Facebook's database.
Notes and Issues:	-

Use Case ID:	3		
Use Case Name:	Validate the account availability		
Created By:	Alvin Lee	**Last Updated By:**	Bryan Liu
Date Created:	6th September 2017	**Date Last Updated:**	10th November 2017

Actor:	System
Description:	System to validate the account availability, send verification code to the registered email address, and create the user account.
Preconditions:	1. User account must not already exist in the database. 2. Mobile must be connected to WiFi/Mobile Data.
Postconditions:	1. User account is successfully created in the system’s database. 2. User is able to login using their registered user account.
Priority:	Medium
Frequency of Use:	1 - 3 times per lifetime
Flow of Events:	1. System will validate the user account availability in the database. 2. System will create the user account in the database.
Alternative Flows:	AF-S1-1: System detects the user account is already existed in the database. 1. System will display error message “Account already exists.” 2. User will enter a new username. 3. User select the register button to re-attempt registration again. 4. Return to Step 1.
Exceptions:	-
Includes:	1. Validate the account availability.
Special Requirements:	-
Assumptions:	- Database can be referred to both System’s database and Facebook’s database.
Notes and Issues:	-

Use Case ID:	4		
Use Case Name:	Register via Facebook account		
Created By:	Alvin Lee	**Last Updated By:**	Bryan Liu
Date Created:	6th September 2017	**Date Last Updated:**	10th November 2017

Actor:	User
Description:	Registration for a new user account via user's Facebook account.
Preconditions:	1. User account must not already exist in the database. 2. Mobile must be connected to WiFi/Mobile Data.
Postconditions:	-
Priority:	Medium
Frequency of Use:	1 - 3 times per lifetime
Flow of Events:	1. User select "Login via Facebook" button. 2. User is already logged in to their Facebook account. 3. System will validate the account availability with respect to database. 4. User will be prompted to grant permission for System to access User's Facebook account. 5. User will grant the permission. 6. System will extract User data from User's Facebook account and save them in database. 7. User will be able to login successfully using their Facebook account.
Alternative Flows:	AF-S2: System detects user's mobile is not logged in to the Facebook account. 1. System will prompt User to enter their credentials to login to their Facebook account. 2. User provide their credentials to login. 3. Return to Step 3.
Exceptions:	-
Includes:	1. Validate the account availability.
Special Requirements:	-
Assumptions:	- Database can be referred to both System's database and Facebook's database.
Notes and Issues:	-

Use Case ID:	5		
Use Case Name:	Select and Retrieve details of restaurant		
Created By:	Johnathan	**Last Updated By:**	Johnathan
Date Created:	8th September 2017	**Date Last Updated:**	11th November 2017

Actor:	User
Description:	To select the indicated restaurants and retrieve more details of the restaurant.
Preconditions:	1. Mobile must be connected to WiFi/Mobile Data.
Postconditions:	1. The selected restaurant details will be displayed in the system.
Priority:	High
Frequency of Use:	0 - 20 times per day
Flow of Events:	1. System will display a list of nearby restaurants in the Main Menu. 2. User select from the lists of nearby restaurants to know more about the details of that restaurant. 3. System will display the details of the selected restaurant.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	6		
Use Case Name:	Bookmark their favourite restaurants		
Created By:	Johnathan	**Last Updated By:**	Bryan Lim
Date Created:	8th September 2017	**Date Last Updated:**	11th November 2017

Actor:	User
Description:	To bookmark their favourite restaurants.
Preconditions:	1. Mobile must be connected to WiFi/Mobile Data. 2. User must be logged in to their account.
Postconditions:	1. User's favourite restaurant will be saved into the database.
Priority:	High
Frequency of Use:	0 - 20 times per day
Flow of Events:	1. User navigate to the restaurant details page. 2. User select the favourite button to bookmark their favourite restaurant. 3. System will save the user's favourite restaurant into the database.
Alternative Flows:	-
Exceptions:	-
Includes:	1. Verify Login Credentials
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	7		
Use Case Name:	Retrieve their list of favourite restaurants		
Created By:	Htet Naing	**Last Updated By:**	Johnathan
Date Created:	8th September 2017	**Date Last Updated:**	9th November 2017

Actor:	User
Description:	To retrieve the list of their favourite restaurants
Preconditions:	1. Mobile must be connected to WiFi/Mobile Data. 2. User must login to their account.
Postconditions:	1. A list of User's favourite restaurants will be displayed on the Favourites page.
Priority:	High
Frequency of Use:	0 - 5 times per day
Flow of Events:	1. User navigate to the Favourites page. 2. System will retrieve the list of User's favourite restaurants from the database. 3. System will display a list of User's favourite restaurants on the Favourites page.
Alternative Flows:	-
Exceptions:	-
Includes:	1. Verify Login Credentials
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	8		
Use Case Name:	Remove their favourite restaurants		
Created By:	Johnathan	**Last Updated By:**	Bryan Lim
Date Created:	8th September 2017	**Date Last Updated:**	12th November 2017

Actor:	User
Description:	To remove their favourite restaurants from the list of bookmarks.
Preconditions:	1. Mobile must be connected to WiFi/Mobile Data. 2. User must be logged in to their account.
Postconditions:	1. User's selected favourite restaurants will be removed from the list of bookmarks.
Priority:	High
Frequency of Use:	0 - 20 times per day
Flow of Events:	1. User navigate to favourite page. 2. User select the favourite restaurant to be removed. 3. System will remove the favourite restaurant from the list of user's favourite restaurants and update the database.
Alternative Flows:	-
Exceptions:	-
Includes:	1. Verify Login Credentials
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	9		
Use Case Name:	Search for restaurants based on filters		
Created By:	Alvin Lee	**Last Updated By:**	Lixin
Date Created:	8th September 2017	**Date Last Updated:**	8th November 2017

Actor:	User
Description:	To search for a variety of restaurants based on filters.
Preconditions:	1. Mobile must be connected to WiFi/Mobile Data.
Postconditions:	1. A list of restaurant will be displayed in the system based on user's search filters.
Priority:	High
Frequency of Use:	0-20 times per day
Flow of Events:	1. User enters his query in the Search bar. 2. System will provide auto-completion/suggestion based on User's query. 3. User tapped the Search button without adding any filters. 4. System will display the list of restaurants based on User's input/filters.
Alternative Flows:	AF-S3: User tapped the Search button and modify the search by adding filters. 1. User can filter by Popularity rating, Type of cuisine, Price rate, Restaurant Location and Restaurant distance from current location. 2. System will ignore any user inputs in the Search bar once User has chosen any constraint through filters. 3. Return to Step 4.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

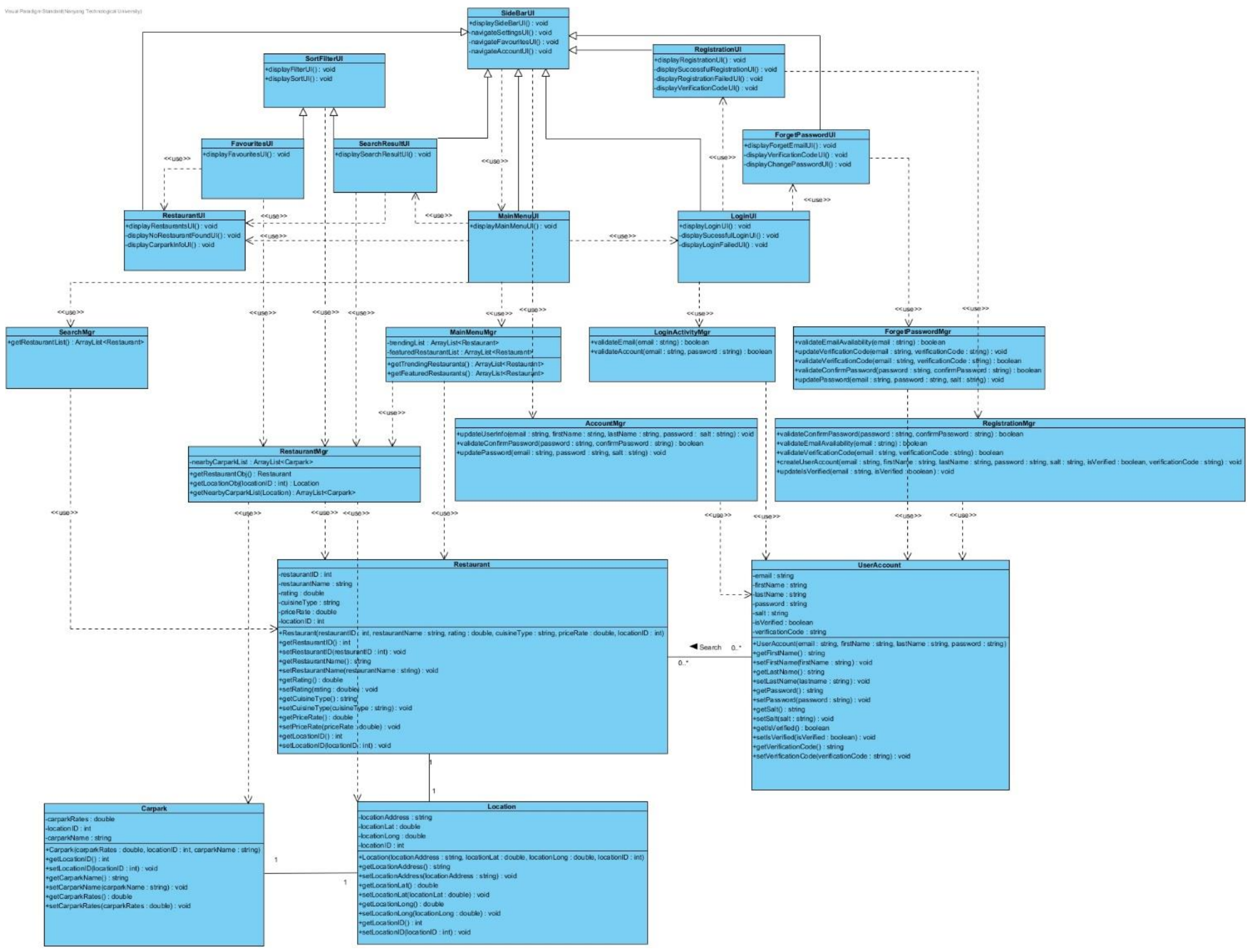
Use Case ID:	10		
Use Case Name:	Request for Password Change		
Created By:	Alvin Lee	**Last Updated By:**	Htet Naing
Date Created:	4th September 2017	**Date Last Updated:**	11th November 2017

Actor:	User
Description:	To allow User to change or reset their passwords in the event that they are unable to remember them.
Preconditions:	1. Mobile must be connected to WiFi/Mobile Data.
Postconditions:	1. System has successfully updated User's new password in the database and User is able to login.
Priority:	High
Frequency of Use:	3 - 5 times per year.
Flow of Events:	1. User type in their registered email address and tap the Request button. 2. System will check if the provided email address is matched with the one in database. 3. System will send the OTP code to User's registered email address. 4. User will enter the OTP code and tap the Submit button. 5. System will validate the OTP code. 6. User will enter the new password and confirm the new password and tap the Save button. 7. System will check if there is a mismatch between the two passwords. 8. System will save user's new password in database.
Alternative Flows:	AF-S2: The provided email address is not found in the database. 1. System will display a message, "Email is not found". 2. Return to Step 1. AF-S5: The entered code is invalid. 1. System will display a message, "Invalid OTP code". 2. Return to Step 4. AF-S7: There is a mismatch between the two passwords. 1. System will display a message, "Mismatch between the two passwords". 2. Return to Step 6.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	11		
Use Case Name:	Retrieve details of Carpark Information		
Created By:	Alvin Lee	**Last Updated By:**	Alvin Lee
Date Created:	8th September 2017	**Date Last Updated:**	11th November 2017

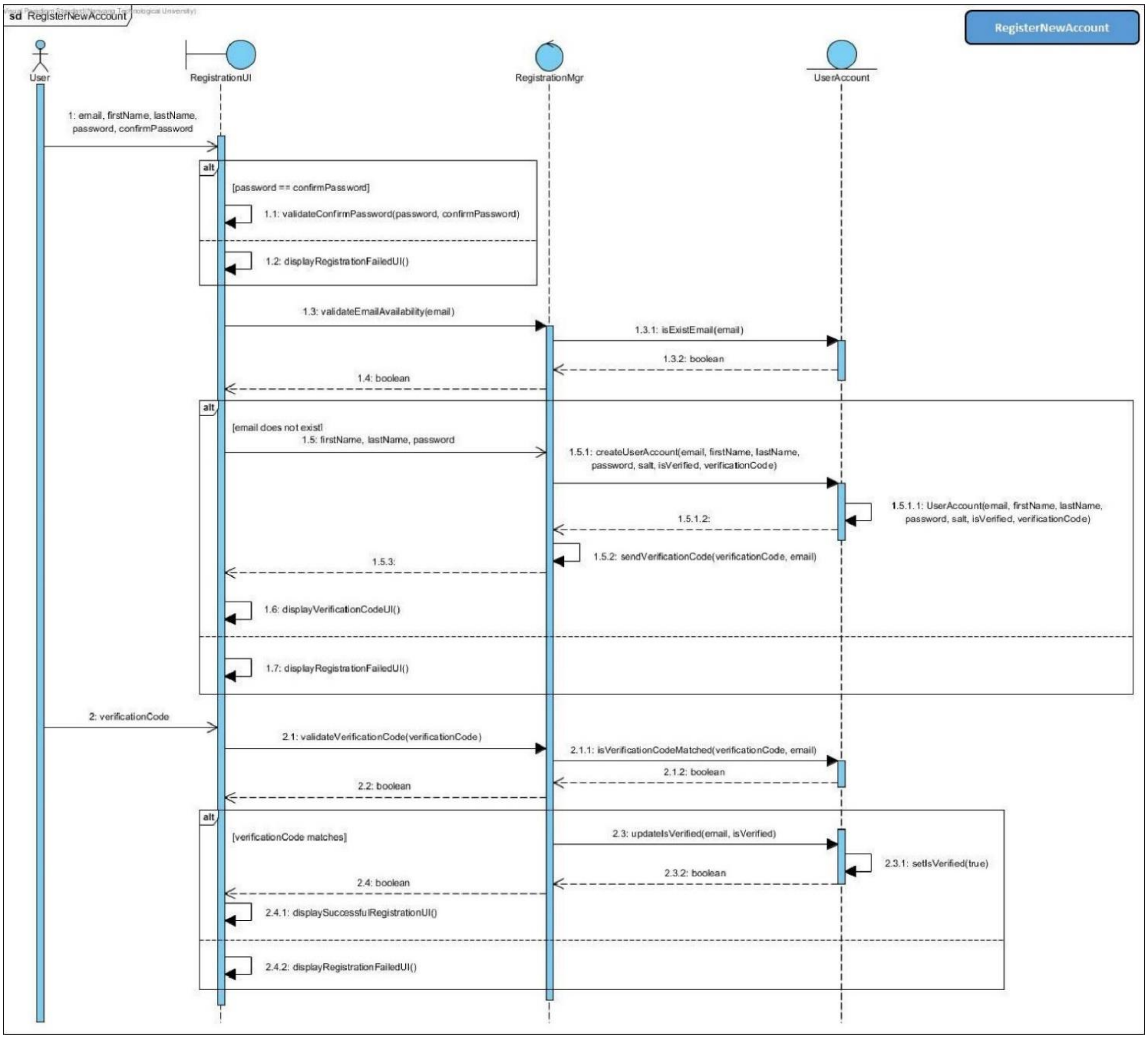
Actor:	User
Description:	To retrieve details of carpark information.
Preconditions:	1. Mobile must be connected to WiFi/Mobile Data.
Postconditions:	1. User will be able to retrieve the details of the carpark from the system.
Priority:	Medium
Frequency of Use:	0-10 times per day
Flow of Events:	1. User navigate to detailed view of a particular restaurant. 2. System will provide a selectable list of nearest parking lot. 3. User will select from the drop-down button. 4. System will display details of the selected carpark.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

2.3 Class Diagram

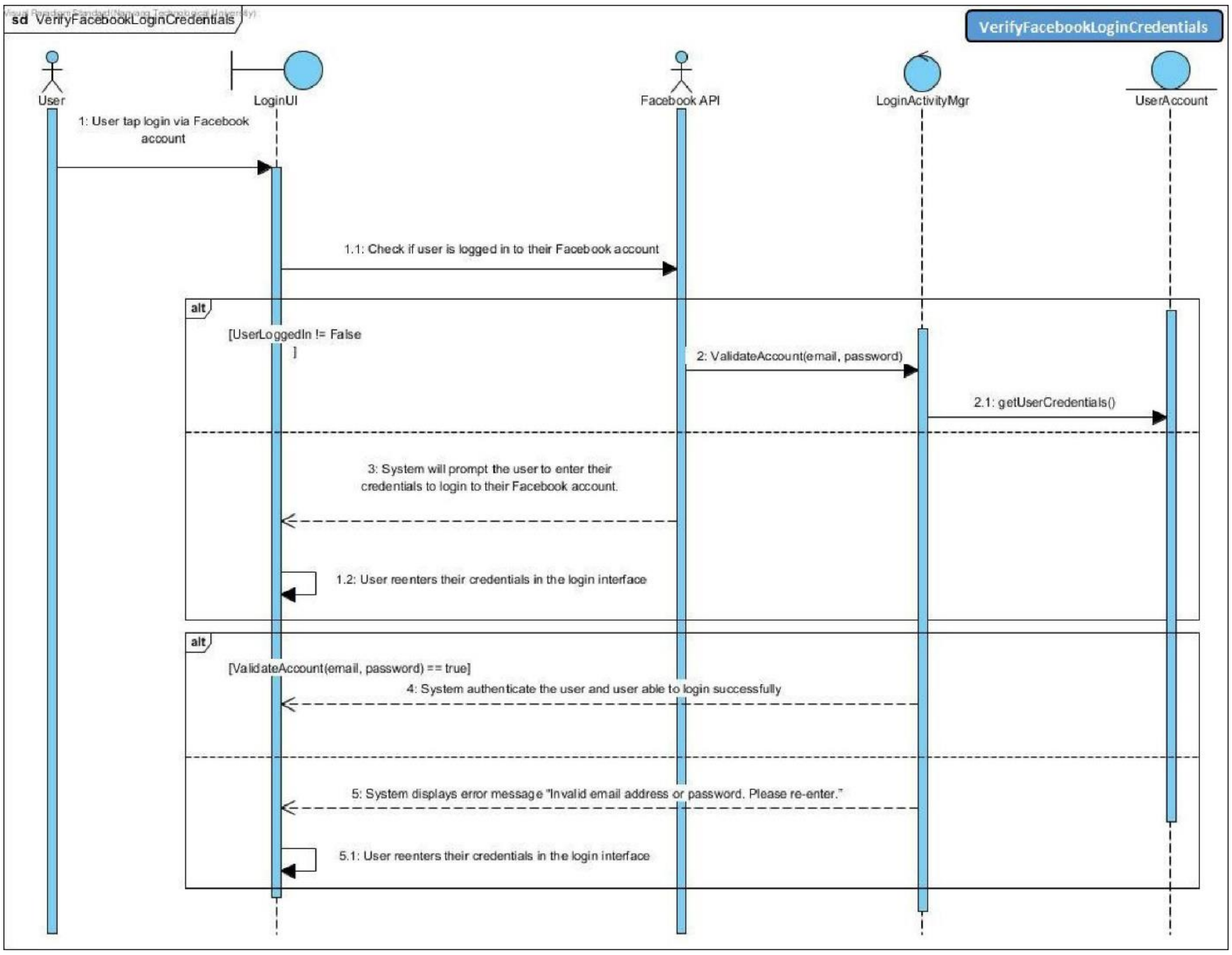

Visual Paradigm Standard(Nanyang Technological University)
SideBarUI
+displaySideBarUI() : void
-navigateSettingsUI() : void
-navigateFavouritesUI() : void
-navigateAccountUI() : void
SortFilterUI
+displayFilterUI() : void
+displaySortUI() : void
RegistrationUI
+displayRegistrationUI() : void
-displaySuccessfulRegistrationUI() : void
-displayRegistrationFailedUI() : void
-displayVerificationCodeUI() : void
ForgetPasswordUI
+displayForgetEmailUI() : void
-displayVerificationCodeUI() : void
-displayChangePasswordUI() : void
FavouritesUI
+displayFavouritesUI() : void
SearchResultUI
+displaySearchResultUI() : void
RestaurantUI
+displayRestaurantsUI() : void
-displayNoRestaurantFoundUI() : void
-displayCarparkInfoUI() : void
MainMenuUI
+displayMainMenuUI() : void
LoginUI
+displayLoginUI() : void
-displaySucessfulLoginUI() : void
-displayLoginFailedUI() : void
<<use>>
SearchMgr
+getRestaurantList() : ArrayList<Restaurant>
MainMenuMgr
-trendingList : ArrayList<Restaurant>
-featuredRestaurantList : ArrayList<Restaurant>
+getTrendingRestaurants() : ArrayList<Restaurant>
+getFeaturedRestaurants() : ArrayList<Restaurant>
LoginActivityMgr
+validateEmail(email : string) : boolean
+validateAccount(email : string, password : string) : boolean
ForgetPasswordMgr
+validateEmailAvailability(email : string) : boolean
+updateVerificationCode(email : string, verificationCode : string) : void
+validateVerificationCode(email : string, verificationCode : string) : boolean
+validateConfirmPassword(password : string, confirmPassword : string) : boolean
+updatePassword(email : string, password : string, salt : string) : void
AccountMgr
+updateUserInfo(email : string, firstName : string, lastName : string, password : salt : string) : void
+validateConfirmPassword(password : string, confirmPassword : string) : boolean
+updatePassword(email : string, password : string, salt : string) : void
RegistrationMgr
+validateConfirmPassword(password : string, confirmPassword : string) : boolean
+validateEmailAvailability(email : string) : boolean
+validateVerificationCode(email : string, verificationCode : string) : boolean
+createUserAccount(email : string, firstName : string, lastName : string, password : string, salt : string, isVerified : boolean, verificationCode : string) : void
+updateIsVerified(email : string, isVerified : boolean) : void
RestaurantMgr
-nearbyCarparkList : ArrayList<Carpark>
+getRestaurantObj() : Restaurant
+getLocationObj(locationID : int) : Location
+getNearbyCarparkList(Location) : ArrayList<Carpark>
Restaurant
-restaurantID : int
-restaurantName : string
-rating : double
-cuisineType : string
-priceRate : double
-locationID : int
+Restaurant(restaurantID : int, restaurantName : string, rating : double, cuisineType : string, priceRate : double, locationID : int)
+getRestaurantID() : int
+setRestaurantID(restaurantID : int) : void
+getRestaurantName() : string
+setRestaurantName(restaurantName : string) : void
+getRating() : double
+setRating(rating : double) : void
+getCuisineType() : string
+setCuisineType(cuisineType : string) : void
+getPriceRate() : double
+setPriceRate(priceRate : double) : void
+getLocationID() : int
+setLocationID(locationID : int) : void
Search
0..*
0..*
UserAccount
-email : string
-firstName : string
-lastName : string
-password : string
-salt : string
-isVerified : boolean
-verificationCode : string
+UserAccount(email : string, firstName : string, lastName : string, password : string)
+getFirstName() : string
+setFirstName(firstName : string) : void
+getLastName() : string
+setLastName(lastname : string) : void
+getPassword() : string
+setPassword(password : string) : void
+getSalt() : string
+setSalt(salt : string) : void
+getIsVerified() : boolean
+setIsVerified(isVerified : boolean) : void
+getVerificationCode() : string
+setVerificationCode(verificationCode : string) : void
1
1
Carpark
-carparkRates : double
-locationID : int
-carparkName : string
+Carpark(carparkRates : double, locationID : int, carparkName : string)
+getLocationID() : int
+setLocationID(locationID : int) : void
+getCarparkName() : string
+setCarparkName(carparkName : string) : void
+getCarparkRates() : double
+setCarparkRates(carparkRates : double) : void
1
1
Location
-locationAddress : string
-locationLat : double
-locationLong : double
-locationID : int
+Location(locationAddress : string, locationLat : double, locationLong : double, locationID : int)
+getLocationAddress() : string
+setLocationAddress(locationAddress : string) : void
+getLocationLat() : double
+setLocationLat(locationLat : double) : void
+getLocationLong() : double
+setLocationLong(locationLong : double) : void
+getLocationID() : int
+setLocationID(locationID : int) : void

2.4 Sequence Diagram

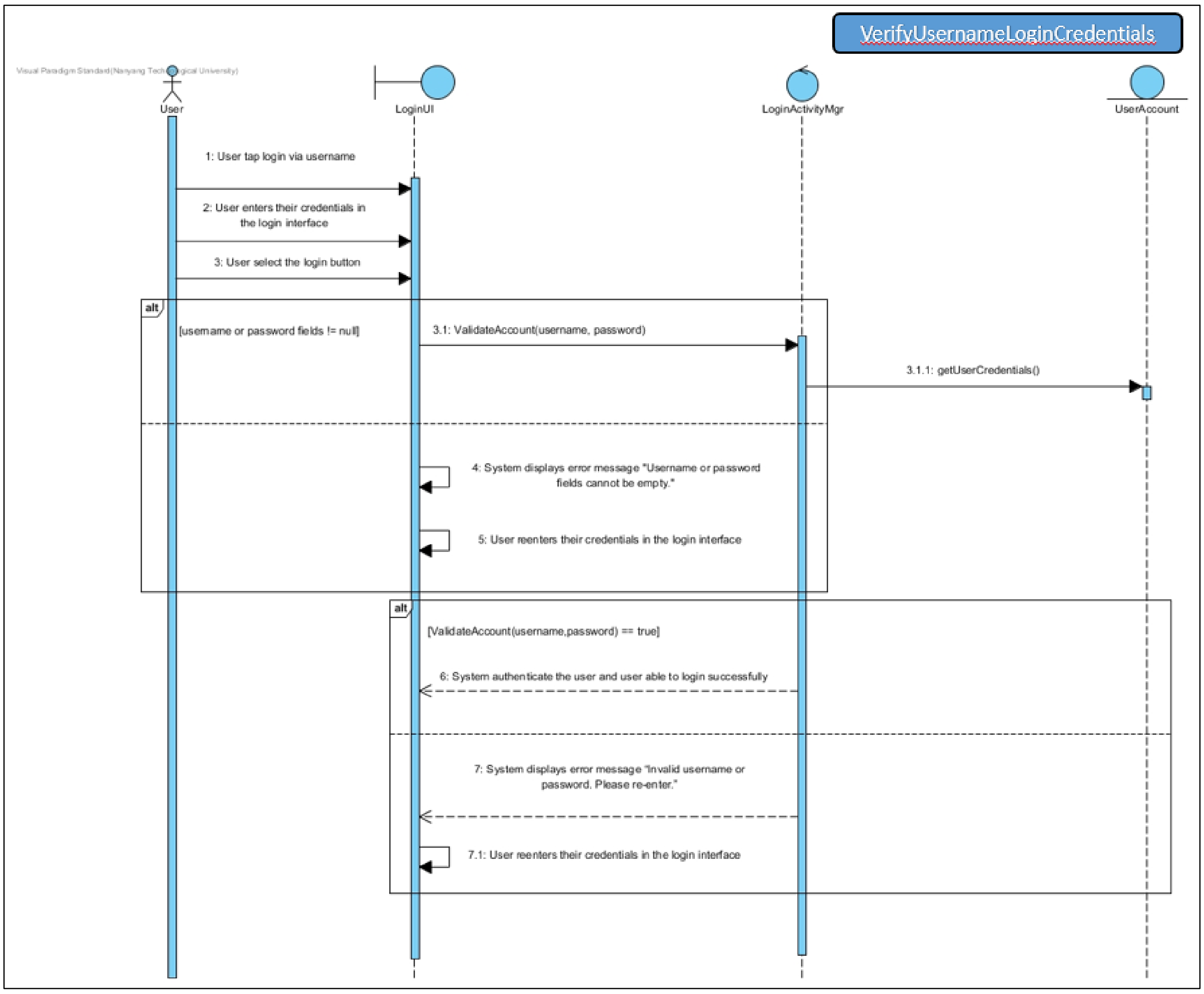
1. Register New Account


sd RegisterNewAccount
RegisterNewAccount
User
RegistrationUI
RegistrationMgr
UserAccount
1: email, firstName, lastName, password, confirmPassword
alt
[password == confirmPassword]
1.1: validateConfirmPassword(password, confirmPassword)
1.2: displayRegistrationFailedUI()
1.3: validateEmailAvailability(email)
1.3.1: isExistEmail(email)
1.3.2: boolean
1.4: boolean
alt
[email does not exist]
1.5: firstName, lastName, password
1.5.1: createUserAccount(email, firstName, lastName, password, salt, isVerified, verificationCode)
1.5.1.1: UserAccount(email, firstName, lastName, password, salt, isVerified, verificationCode)
1.5.1.2:
1.5.2: sendVerificationCode(verificationCode, email)
1.5.3:
1.6: displayVerificationCodeUI()
1.7: displayRegistrationFailedUI()
2: verificationCode
2.1: validateVerificationCode(verificationCode)
2.1.1: isVerificationCodeMatched(verificationCode, email)
2.1.2: boolean
2.2: boolean
alt
[verificationCode matches]
2.3: updateIsVerified(email, isVerified)
2.3.1: setIsVerified(true)
2.3.2: boolean
2.4: boolean
2.4.1: displaySuccessfulRegistrationUI()
2.4.2: displayRegistrationFailedUI()

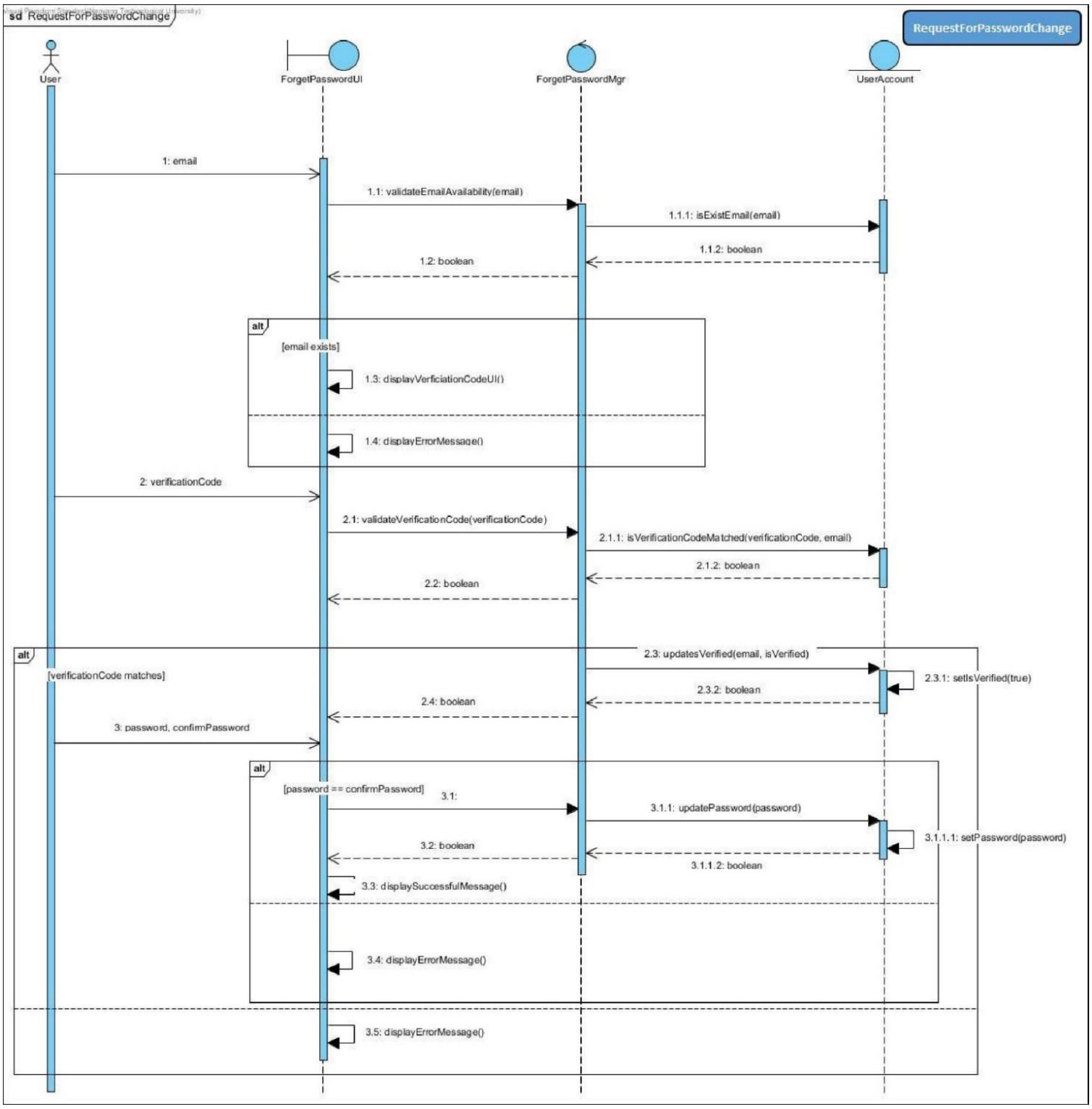
2. Verify Facebook Login Credentials


sd VerifyFacebookLoginCredentials
VerifyFacebookLoginCredentials
User
LoginUI
Facebook API
LoginActivityMgr
UserAccount
1: User tap login via Facebook account
1.1: Check if user is logged in to their Facebook account
alt
[UserLoggedIn != False]
2: ValidateAccount(email, password)
2.1: getUserCredentials()
3: System will prompt the user to enter their credentials to login to their Facebook account.
1.2: User reenters their credentials in the login interface
alt
[ValidateAccount(email, password) == true]
4: System authenticate the user and user able to login successfully
5: System displays error message "Invalid email address or password. Please re-enter."
5.1: User reenters their credentials in the login interface

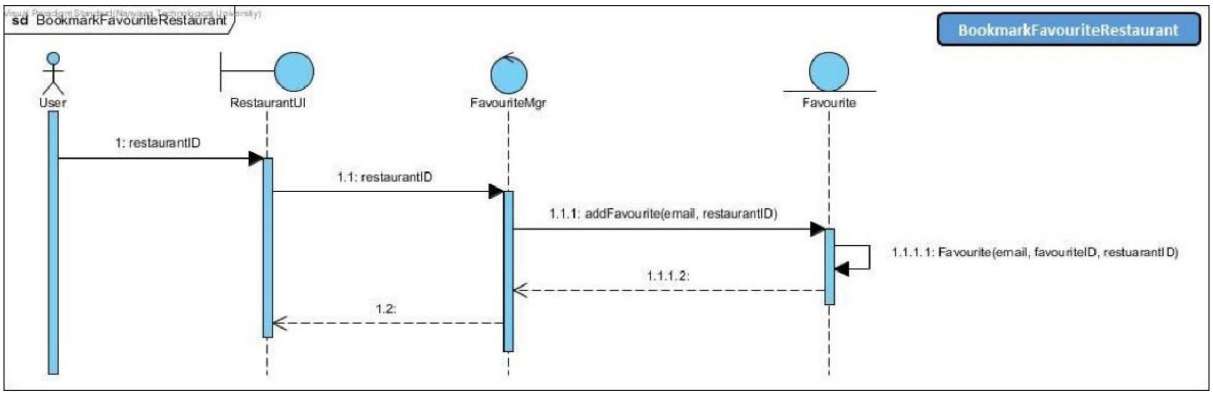
3. Verify Username Login Credentials


VerifyUsernameLoginCredentials
Visual Paradigm Standard(Nanyang Technological University)
User
LoginUI
LoginActivityMgr
UserAccount
1: User tap login via username
2: User enters their credentials in the login interface
3: User select the login button
alt
[username or password fields != null]
3.1: ValidateAccount(username, password)
3.1.1: getUserCredentials()
4: System displays error message "Username or password fields cannot be empty."
5: User reenters their credentials in the login interface
alt
[ValidateAccount(username,password) == true]
6: System authenticate the user and user able to login successfully
7: System displays error message "Invalid username or password. Please re-enter."
7.1: User reenters their credentials in the login interface

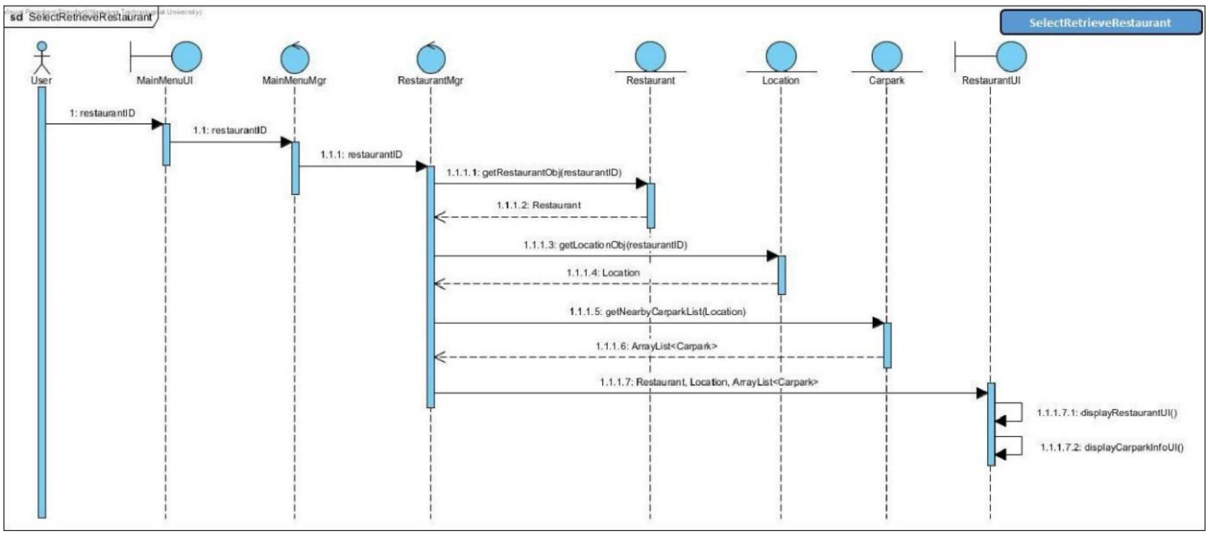
4. Request for Password Change


sd RequestForPasswordChange
RequestForPasswordChange
User
ForgetPasswordUI
ForgetPasswordMgr
UserAccount
1: email
1.1: validateEmailAvailability(email)
1.1.1: isExistEmail(email)
1.1.2: boolean
1.2: boolean
alt
[email exists]
1.3: displayVerficiationCodeUI()
1.4: displayErrorMessage()
2: verificationCode
2.1: validateVerificationCode(verificationCode)
2.1.1: isVerificationCodeMatched(verificationCode, email)
2.1.2: boolean
2.2: boolean
alt
2.3: updatesVerified(email, isVerified)
[verificationCode matches]
2.3.1: setIsVerified(true)
2.3.2: boolean
2.4: boolean
3: password, confirmPassword
alt
[password == confirmPassword]
3.1:
3.1.1: updatePassword(password)
3.1.1.1: setPassword(password)
3.2: boolean
3.1.1.2: boolean
3.3: displaySuccessfulMessage()
3.4: displayErrorMessage()
3.5: displayErrorMessage()

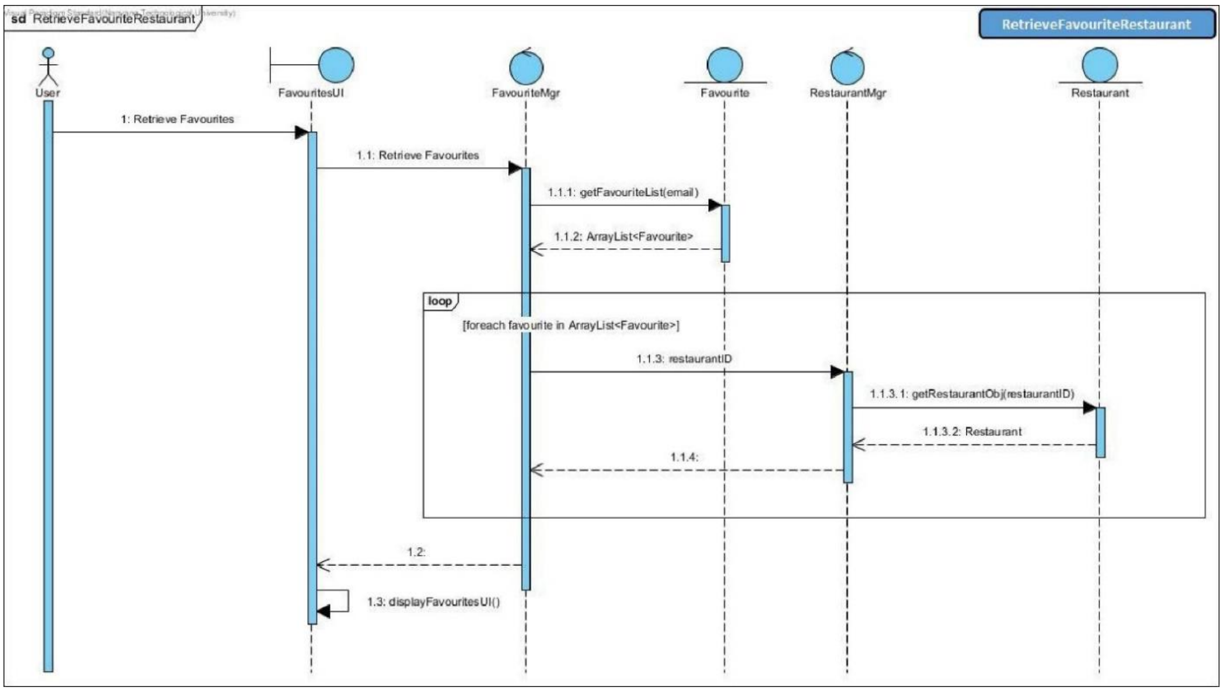
5. Bookmark Favourite Restaurant


sd BookmarkFavouriteRestaurant
BookmarkFavouriteRestaurant
User
RestaurantUI
FavouriteMgr
Favourite
1: restaurantID
1.1: restaurantID
1.1.1: addFavourite(email, restaurantID)
1.1.1.1: Favourite(email, favouriteID, restuarantID)
1.1.1.2:
1.2:

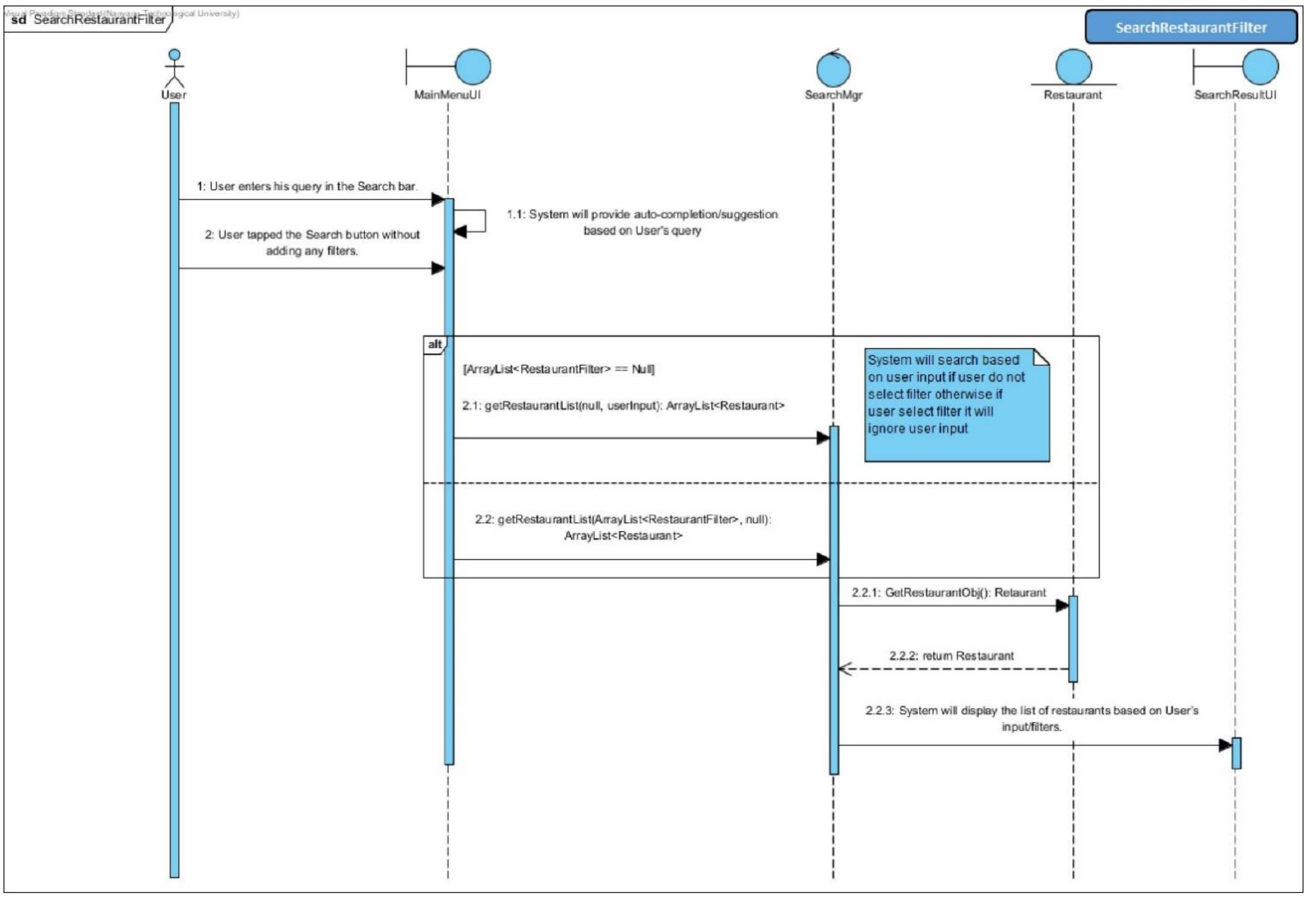
6. Select/Retrieve Restaurant


sd SelectRetrieveRestaurant
SelectRetrieveRestaurant
User
MainMenuUI
MainMenuMgr
RestaurantMgr
Restaurant
Location
Carpark
RestaurantUI
1: restaurantID
1.1: restaurantID
1.1.1: restaurantID
1.1.1.1: getRestaurantObj(restaurantID)
1.1.1.2: Restaurant
1.1.1.3: getLocationObj(restaurantID)
1.1.1.4: Location
1.1.1.5: getNearbyCarparkList(Location)
1.1.1.6: ArrayList<Carpark>
1.1.1.7: Restaurant, Location, ArrayList<Carpark>
1.1.1.7.1: displayRestaurantUI()
1.1.1.7.2: displayCarparkInfoUI()

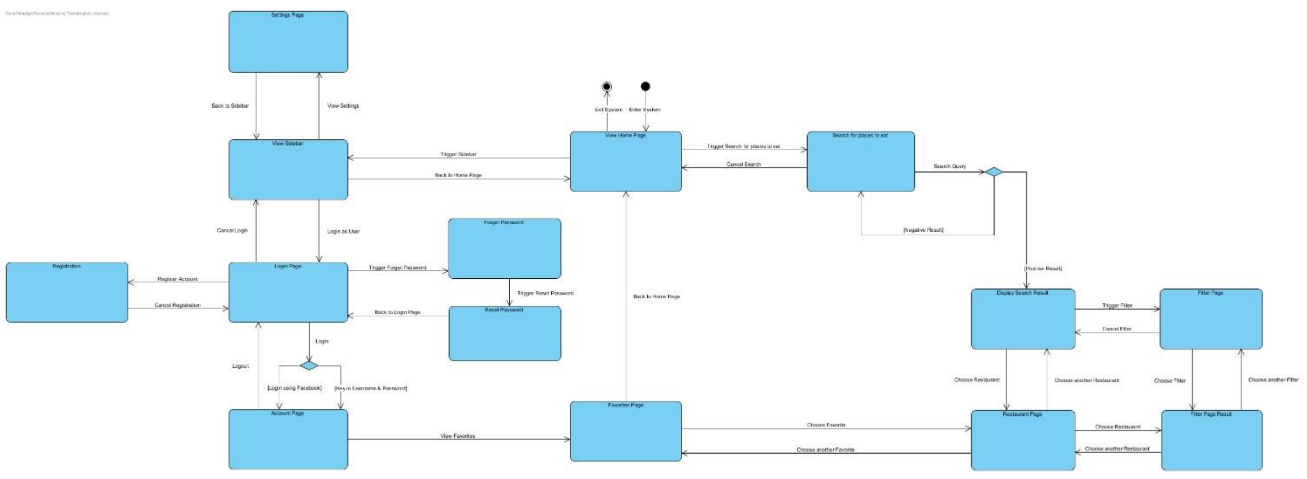
7. Retrieve Favourite Restaurant


sd RetrieveFavouriteRestaurant
RetrieveFavouriteRestaurant
User
FavouritesUI
FavouriteMgr
Favourite
RestaurantMgr
Restaurant
1: Retrieve Favourites
1.1: Retrieve Favourites
1.1.1: getFavouriteList(email)
1.1.2: ArrayList<Favourite>
loop
[foreach favourite in ArrayList<Favourite>]
1.1.3: restaurantID
1.1.3.1: getRestaurantObj(restaurantID)
1.1.3.2: Restaurant
1.1.4:
1.2:
1.3: displayFavouritesUI()

8. Search Restaurant Filter


sd SearchRestaurantFilter
SearchRestaurantFilter
User
MainMenuUI
SearchMgr
Restaurant
SearchResultUI
1: User enters his query in the Search bar.
1.1: System will provide auto-completion/suggestion based on User's query
2: User tapped the Search button without adding any filters.
alt
[ArrayList<RestaurantFilter> == Null]
2.1: getRestaurantList(null, userInput): ArrayList<Restaurant>
System will search based on user input if user do not select filter otherwise if user select filter it will ignore user input
2.2: getRestaurantList(ArrayList<RestaurantFilter>, null): ArrayList<Restaurant>
2.2.1: GetRestaurantObj(): Retaurant
2.2.2: return Restaurant
2.2.3: System will display the list of restaurants based on User's input/filters.

2.5 Dialog Map



3. Non-Functional Requirements

1. Usability Requirements

1.1. The system must reduce short-term memory load.
   1.1.1. To keep the display simple and allow user to retrieve their list of favourites easily.
1.2. The system must permit easy reversal of actions.
   1.2.1. The user must be able to bookmark and remove their favourite restaurants easily.
1.3. The system must offer informative feedback.
   1.3.1. To provide necessary feedback to the user when invalid inputs are detected.
   1.3.2. To display an appropriate error message when certain process fails.
1.4. The system must strive for consistency.
   1.4.1. A consistent sequence of actions is required for similar situations.
   1.4.2. A consistent visual layout must be adopted in the application (e.g. labels, fonts, and colours)

2. Performance Requirements

2.1. The system must not crash when the user opens the application.
2.2. The system must be able to return the display results to the user within 2 seconds.
2.3. The system must be able to maintain with little or no downtime occurred.
2.4. The system must be able to support internal locus of control.
   2.4.1. To give the user the sense of control of events occurring and the system will behave as what they expect.
   2.4.2. Fast responsive time in order to prevent the user from experiencing any lag or latency.

3. Security Requirements

3.1. The system will mask the password field in order to prevent any potential shoulder surfing.
3.2. The system will implement Secure Hash Algorithm (SHA) to perform salt-hashing on all the password before storing into the database.
3.3. Upon login, the system will perform salt-hashing on the input password and compare with the hashed password stored in their database.

4. Extendibility Requirements

4.1. The system must be designed with Model-View-Controller architecture and design patterns to support any future enhancements and facilitate extendibility.
4.2. To facilitate easy data access using other platforms, the data collected will be separated from the system and stored in an online database.

4. Interface Requirements

4.1 User

SeeFood works on all types of users who would like to search for nearby food to eat. Currently SeeFood doesn't support for special diet filters of nearby restaurants (e.g. Vegetarian, Halal, etc)

4.2 Hardware

SeeFood requires to work on the hardware devices with location service enabled to locate the user's current location.

4.3 Software

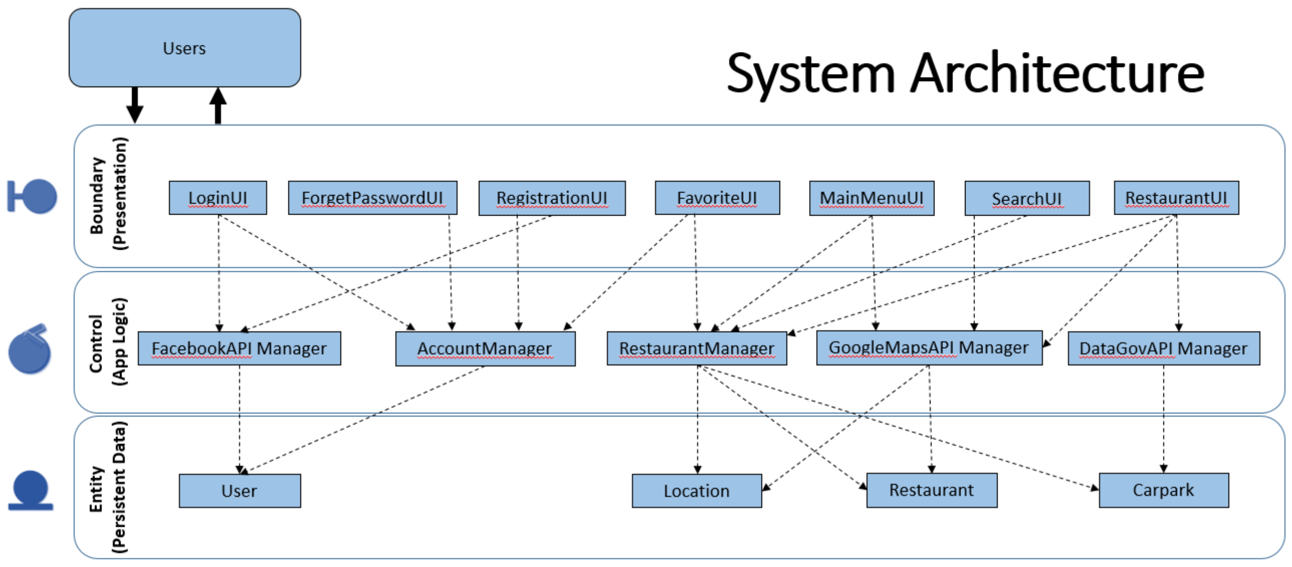
SeeFood is being designed to work on Android Devices.

4.4 Communication

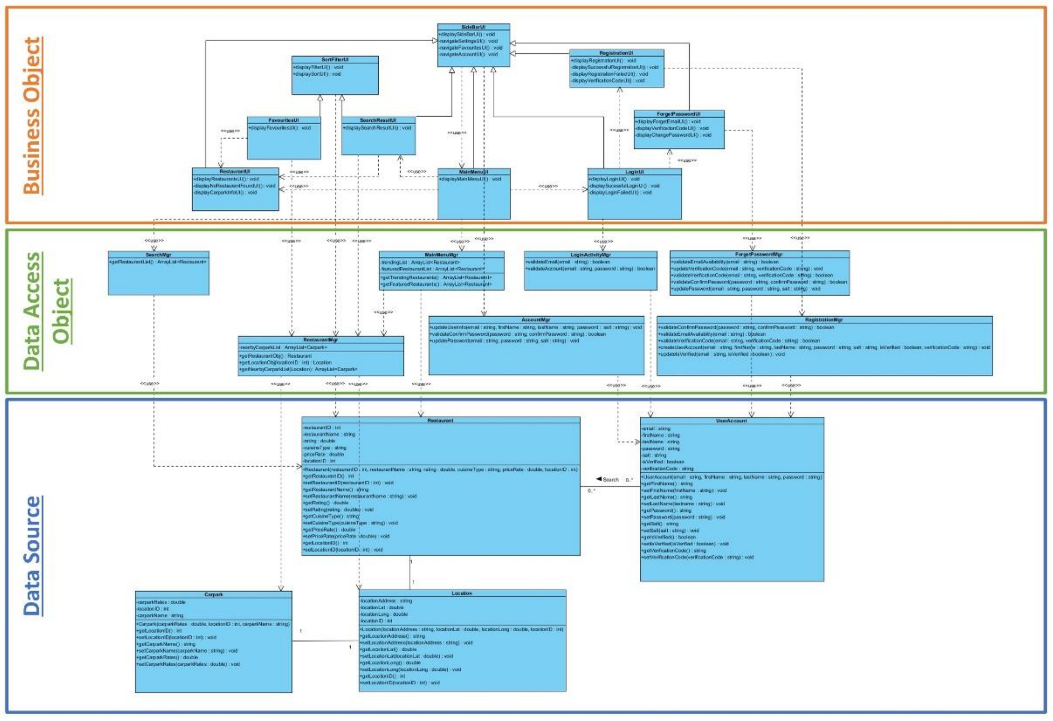
SeeFood will be accessed over the Internet. All features will be accessible through the application.

5. Architecture Design

5.1 System Architecture Diagram



Dependencies flow downwards

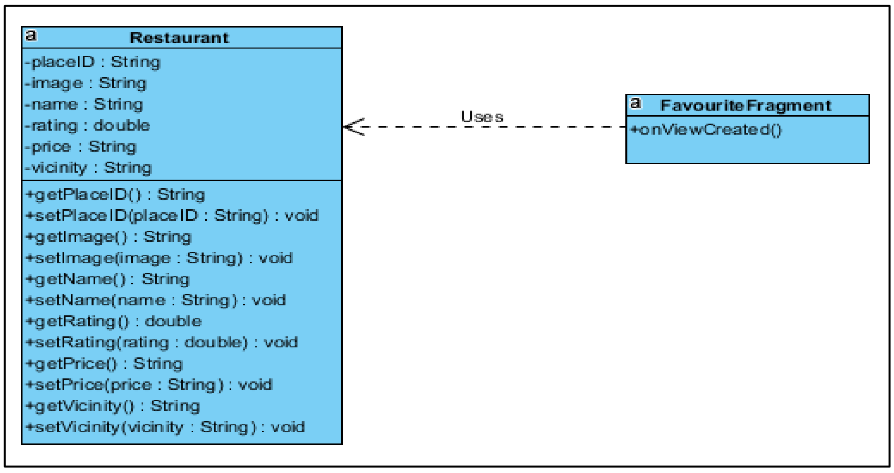


5.2 Design Pattern

1. Data Access Object Pattern

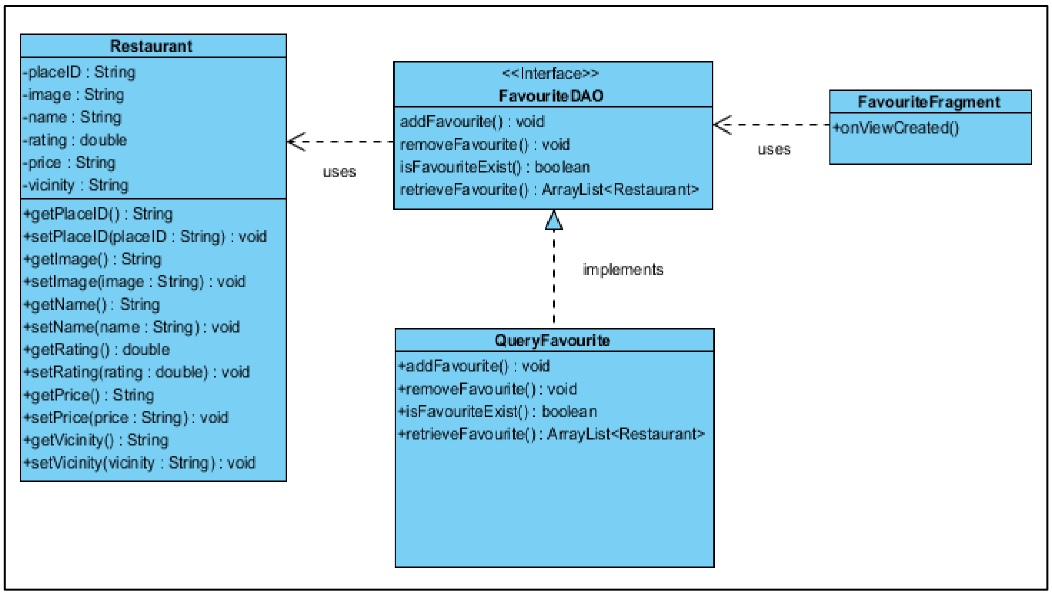
Problem

The code for business logic and data access logic is in the favourite fragment. This makes it difficult to replace or modify any data resources.



Solution

We separate the client interface (favourite fragment) from the data access mechanism. To do this, we define a generic client interface (FavouriteDAO) and having the implementation being done in the QueryFavourite. This allows data access mechanism to change independently of the code that uses the data. It isolates and decouple persistent mechanism from the rest of the application. This isolation supports the Single responsibility principle.



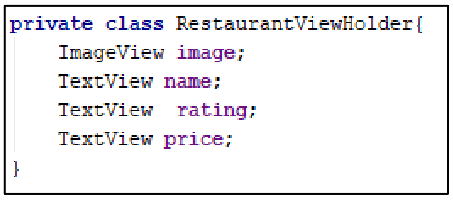
2. View Holder Pattern

Problem

During the scrolling of ListView, findViewById gets called every time, which can slow down performance as it must look up in the view hierarchy to find the view object and update every time. This operation alone is very expensive.

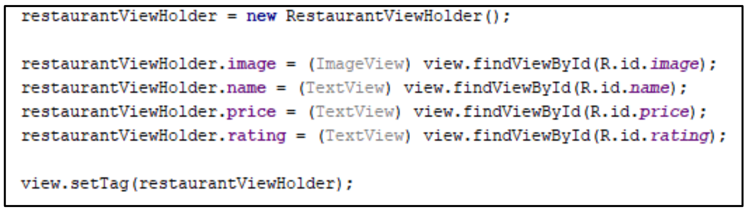
Solution

To resolve the slow scrolling of ListView, implementation of the View Holder pattern is required. First, a class must be created to hold the exact set of views.



```java
private class RestaurantViewHolder{
    ImageView image;
    TextView name;
    TextView  rating;
    TextView price;
}
```

Second, populate the View Holder with the view objects (eg. buttons).



```java
restaurantViewHolder = new RestaurantViewHolder();

restaurantViewHolder.image = (ImageView) view.findViewById(R.id.image);
restaurantViewHolder.name = (TextView) view.findViewById(R.id.name);
restaurantViewHolder.price = (TextView) view.findViewById(R.id.price);
restaurantViewHolder.rating = (TextView) view.findViewById(R.id.rating);

view.setTag(restaurantViewHolder);
```

Lastly, during the scrolling of ListView, the View Holder object can be accessed instead of calling findViewById every time. This greatly improves the performance of the ListView scrolling as the View Holder object can be accessed instead of having to traverse the view hierarchy.

6. Data Dictionary

Term	Definition
User	A user is a person who is using the application to find popular restaurants in Singapore and carparks nearby its location.
System	A system refers to the SeeFOOD Android mobile application.
Popularity Rating	Popularity Rating is the classification for restaurants in Singapore according to stars from 0 to 5, rounded to one decimal places of accuracy (e.g. 4.6 stars). It determines the popularity of the particular restaurants.
Feedback	Feedback is the response time between the user and the application.
Filter	Filter is a feature that processes information to exclude the types which are not wanted, and/or sort the information in order (e.g. Price Rate Low to High)
Search	Search is a feature that allow users to discover a variety of restaurants based on certain filters.
Favourites	Favourites is a feature that allow users to bookmark their favourite restaurants, and they are able to retrieve their list of favourite restaurants for future references.
Registration	Registration refers to a feature that allow the user to sign up for an account should he/she wants to use the Favourites feature.
Salt	A salt is a random generated string that is used as an additional input to a one-way function that "hashes" a password in cryptography.
Model-View-Controller	Model-View-Controller is a software architectural pattern for implementing user interfaces in the application.
Shoulder Surfing	Shoulder Surfing is a type of social engineering technique where another person spies on the user when they are typing their credentials to obtain their password.
GPS	Global Positioning System which will detect a user’s current location. Is used interchangeably with Wi-Fi positioning system.

7. Testing

7.1 Black Box Testing

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 using Facebook	The system displays the main menu for user to continue the operation	The system displays the main menu for user to continue the operation
3	Login without valid credentials	The system prompts the user to enter the credentials again	The system prompts the user to enter the credentials again
4	Login without filling up the required fields	The system prompts the user to fill up the required fields for logging in	The system prompts the user to fill up the required fields for logging in

b. Specific cases (Combination)

Username	Password	Expected Result	Actual Result
testuser	testpass	Successful login	Successful login
wronguser	testpass	Invalid email/password	Invalid email/password
Empty("")	testpass	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 fill up the required fields for registration	The system prompts the user to fill up the required fields for registration
3	Register with password mismatch	The system prompts the user re-enter the password	The system prompts the user re-enter the password

b. Specific cases (Email address)

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

c. Specific cases (Combination)

Email Address	**Username**	**First name**	**Last Name**	**Password**	**Confirm Password**	**Expected Result**	**Actual Result**
user@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
user@gmail.com	Empty(“”)	test	testing	testpass	testpass	Please fill in all required fields	Please fill in all required fields
user@gmail.com	testuser	Empty (“”)	testing	testpass	testpass	Please fill in all required fields	Please fill in all required fields
user@gmail.com	testuser	test	Empty(“”)	testpass	testpass	Please fill in all required fields	Please fill in all required fields
user@gmail.com	testuser	test	testing	Empty(“”)	testpass	Please fill in all required fields	Please fill in all required fields
user@gmail.com	testuser	test	testing	testpass	Empty(“”)	Please fill in all required fields	Please fill in all required fields
existing@gmail.c om	testuser	test	testing	testpass	testpass	Email has been used	Email has been used
user@gmail.com	existinguser	test	testing	testpass	testpass	Username has been used	Usernam e has been used
user@gmail.com	testuser	test	testing	testpass1	testpass2	Entered passwords do not match	Entered passwor ds do not match
user@gmail.com	testuser	test	testing	testpass2	testpass1	Entered passwords do not match	Entered passwor ds do not match

3. Favourites

Test Id	Scenario	Expected Result	Actual Result
1	Favourite a restaurant	The system saves the restaurant into the favourite list	The system saves the restaurant into the favourite list
2	Favourite an existing restaurant	The system prompts the user that the restaurant has been saved before	The system prompts the user that the restaurant has been saved before

4. Get user's current location

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

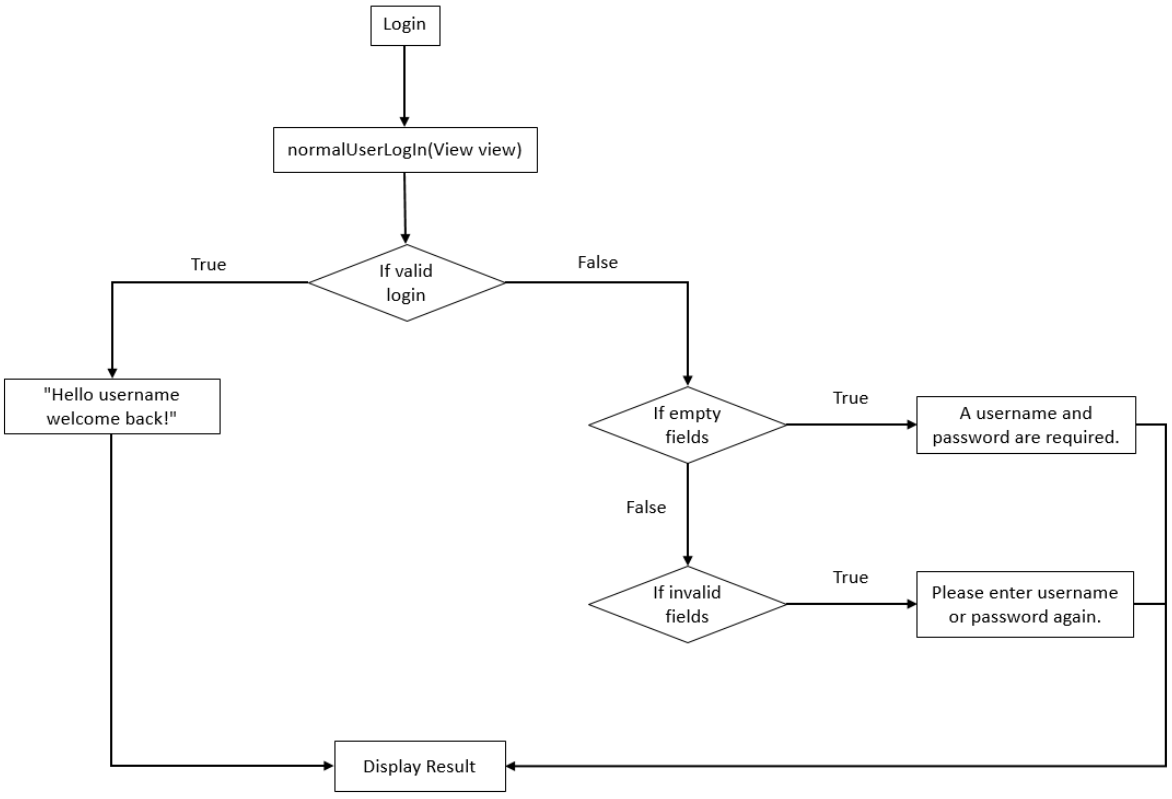
Test Input Latitude	Longitude	Expected Output	Actual Output
1.377728	103.738431	Choa Chu Kang Avenue 5	Choa Chu Kang Avenue 5
1.353466	103.867716	Serangoon Avenue 2	Serangoon Avenue 2
1.3483	103.6831	50 Nanyang Ave	50 Nanyang Ave
1.434145	103.786604	Woodlands Ave 5	Woodlands Ave 5

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

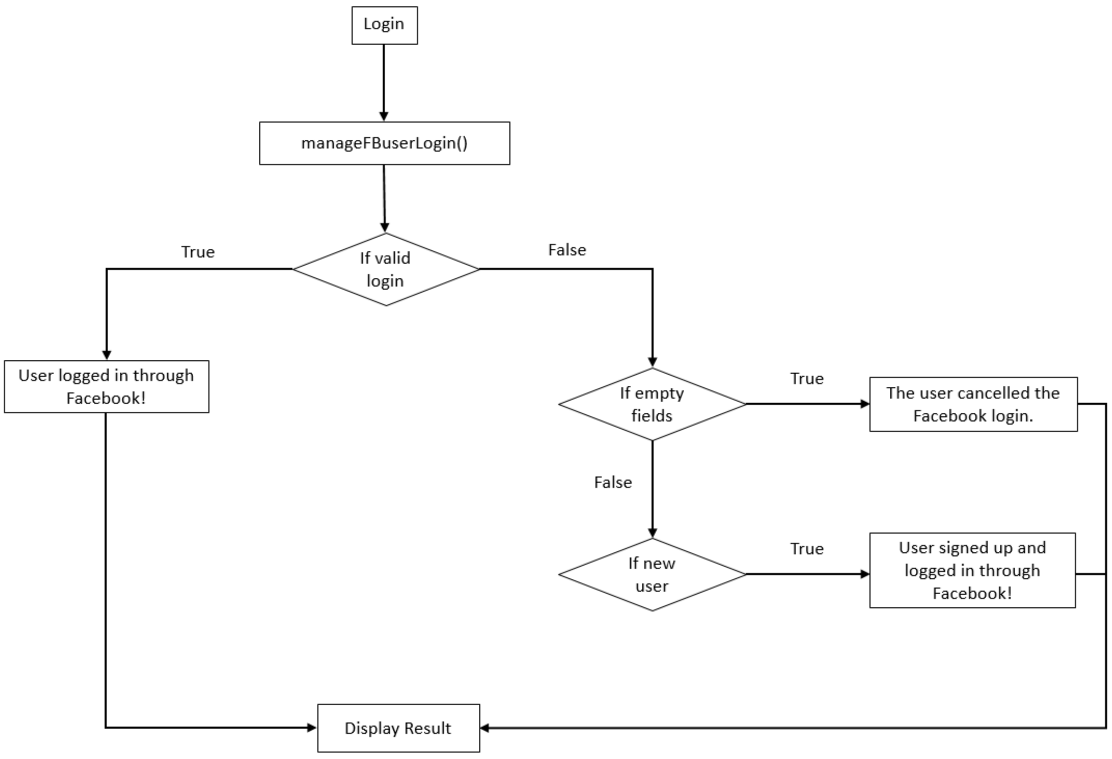
Test Input	Expected Result	Actual Result
Location service not allowed	Permission denied	Permission denied
Mobile GPS not activated	Position unavailable	Position unavailable
Unknown user coordinates	Unknown error	Unknown error

7.2 White Box Testing

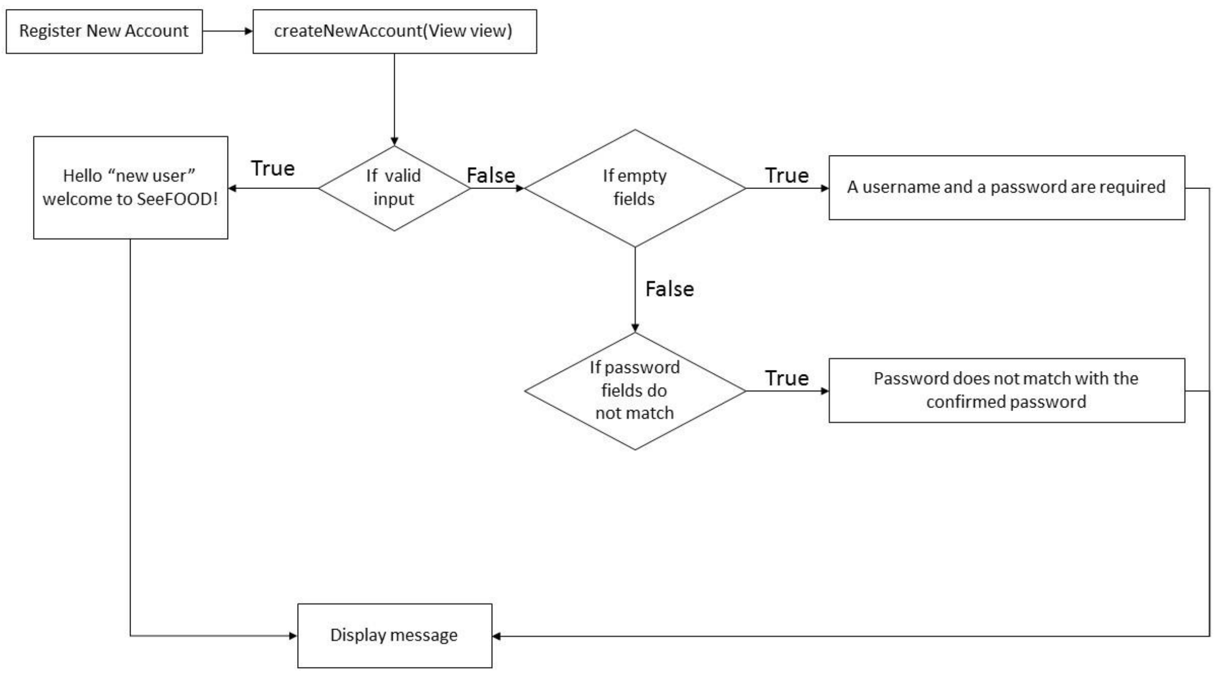
1. Normal User Login


Login
normalUserLogIn(View view)
If valid login
True
False
"Hello username welcome back!"
If empty fields
True
A username and password are required.
False
If invalid fields
True
Please enter username or password again.
Display Result

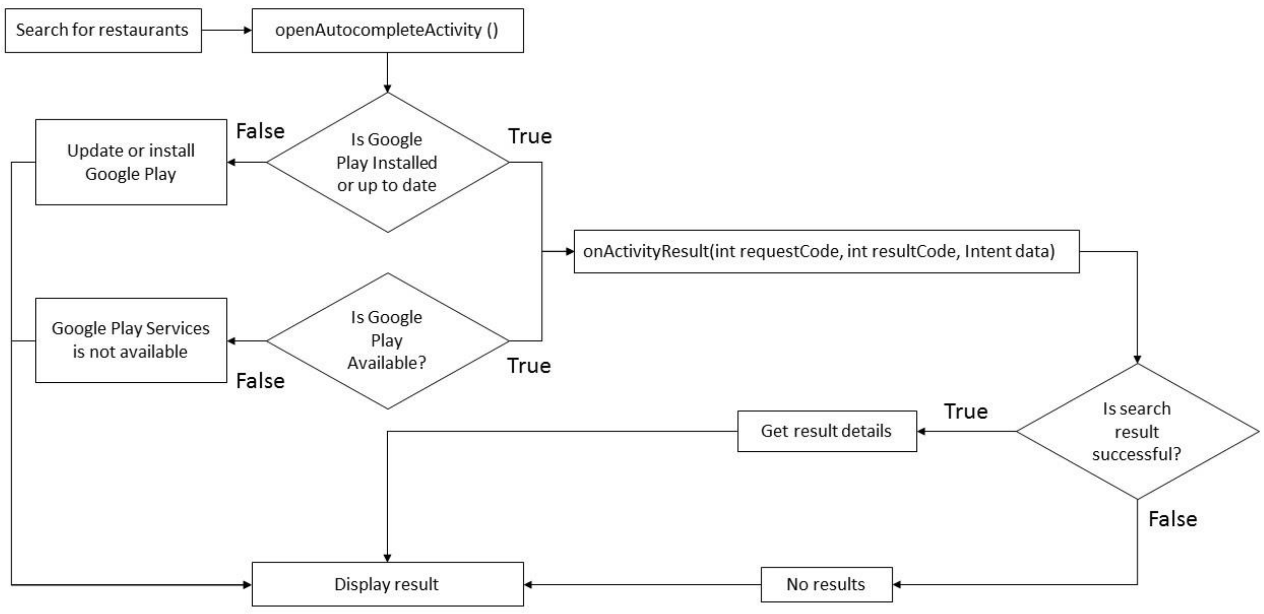
2. Facebook User Login


Login
manageFBuserLogin()
If valid login
True
False
User logged in through Facebook!
If empty fields
True
The user cancelled the Facebook login.
False
If new user
True
User signed up and logged in through Facebook!
Display Result

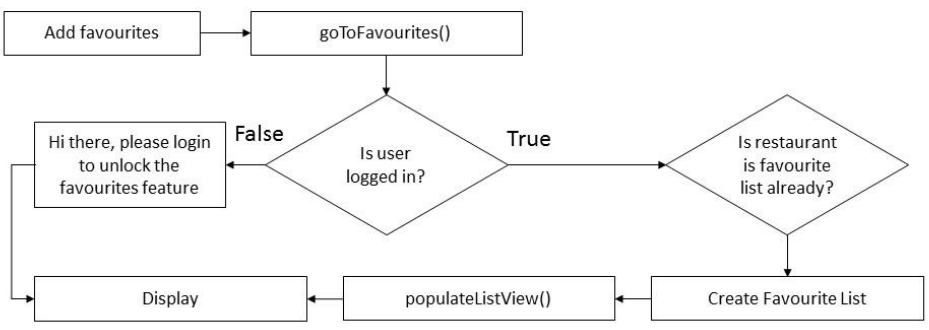
3. Register New Account


Register New Account
createNewAccount(View view)
If valid input
True
Hello "new user" welcome to SeeFOOD!
False
If empty fields
True
A username and a password are required
False
If password fields do not match
True
Password does not match with the confirmed password
Display message

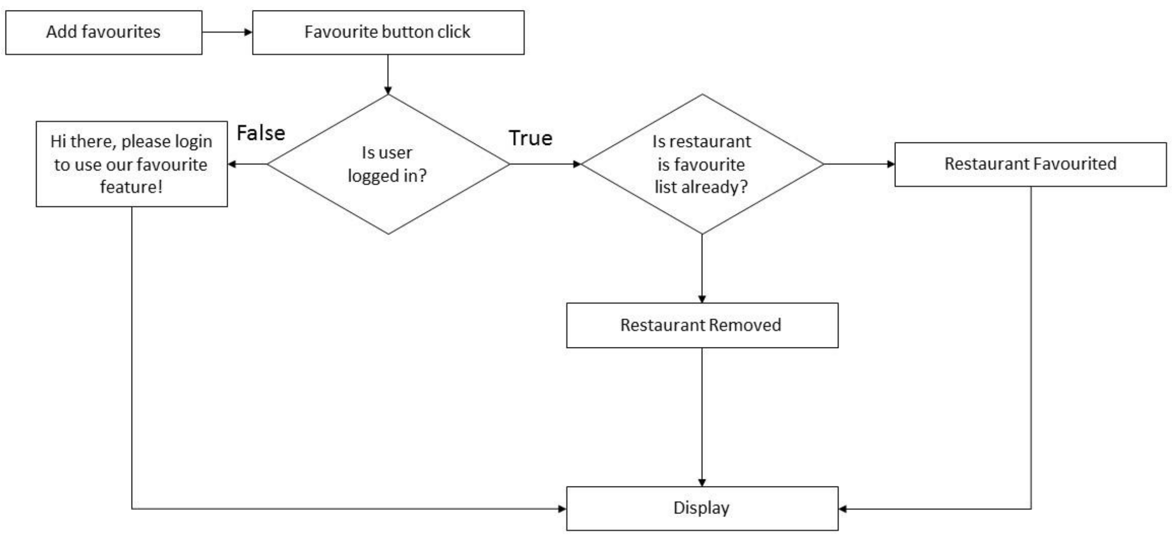
4. Searching


Search for restaurants
openAutocompleteActivity ()
Is Google Play Installed or up to date
False
Update or install Google Play
True
onActivityResult(int requestCode, int resultCode, Intent data)
Is Google Play Available?
False
Google Play Services is not available
True
Is search result successful?
True
Get result details
False
No results
Display result

5. View Favourites



6. Add favourites



8. Appendix

For more information and detailed demo of your SeeFOOD app, please refer to the YouTube link below for our video demo:

https://www.youtube.com/watch?v=QWGCBdcrFso