Lab3

for

TransitEase

Version 3 approved

Prepared by Ashwin, Dave, Jun Heng, Jonathan

Nanyang Technological University

08/10/24

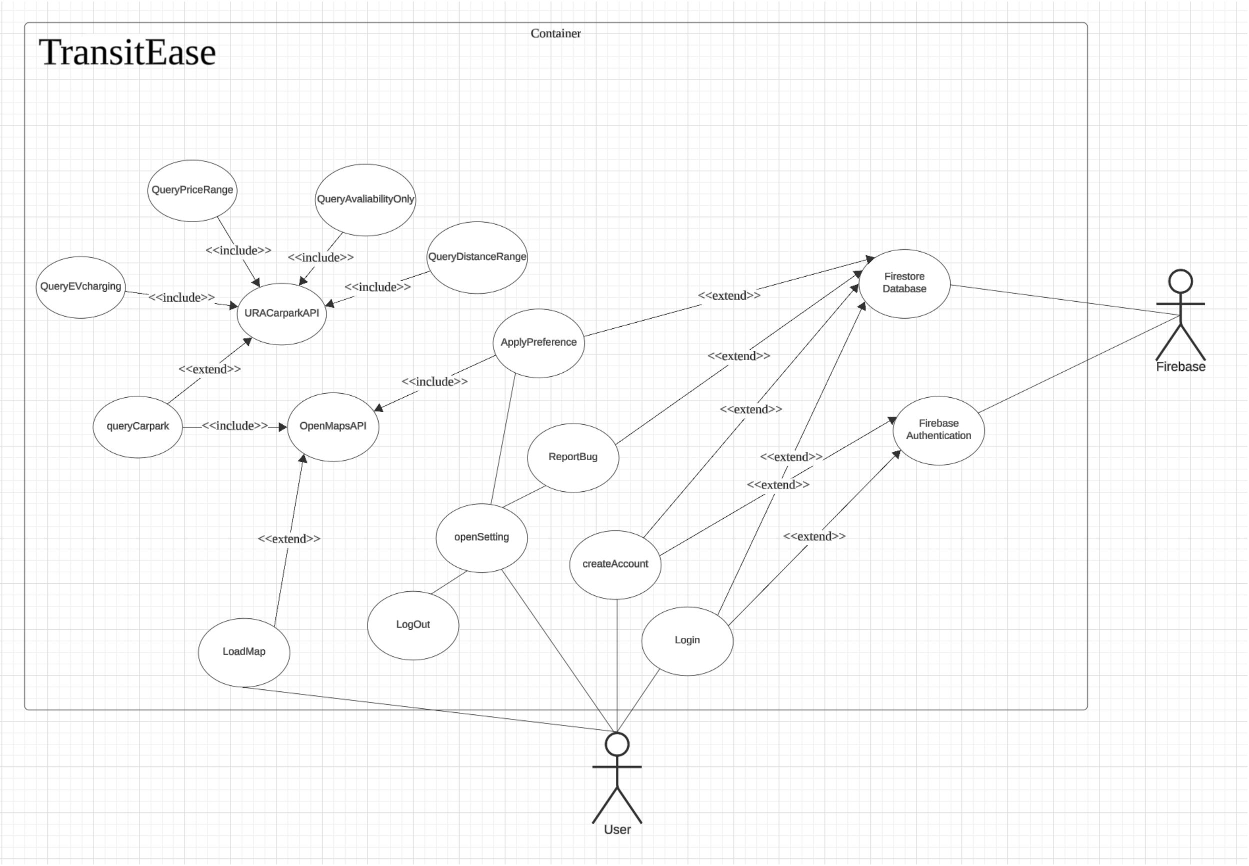
Revision History

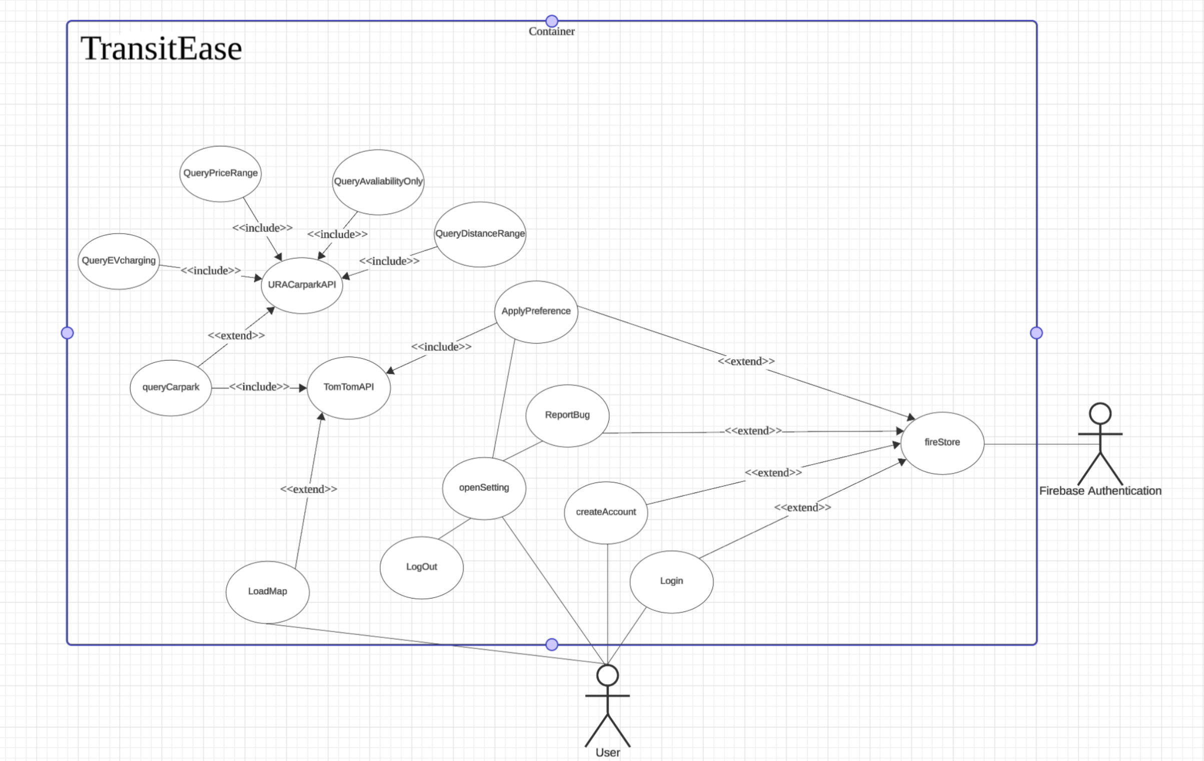
Name	Date	Reason For Changes	Version
Ashwin	22/10	Modified Use Case Diagram and added Dialog Map	3.0
Dave			3.0
Jonathan			3.0
Goh Jun Heng	22/10	Address Design Issues and develop Design Model	3.0

1. Table Of Contents

1. Table Of Contents 1
2. Complete Use Case Diagram 2
3. Use Case Descriptions 4
4. Design Model 12
5. Key Design Issue 17
6. Application Skeleton 18

2. Complete Use Case Diagram




Container
TransitEase
QueryPriceRange
QueryAvaliabilityOnly
QueryDistanceRange
QueryEVcharging
URACarparkAPI
<<include>>
<<include>>
<<include>>
<<include>>
<<extend>>
queryCarpark
<<include>>
TomTomAPI
ApplyPreference
<<include>>
<<extend>>
ReportBug
<<extend>>
fireStore
openSetting
createAccount
<<extend>>
<<extend>>
<<extend>>
LogOut
LoadMap
Login
Firebase Authentication
User

3. Use Case Descriptions

1. LogInEmail

Use Case ID:	1		
Use Case Name:	LogInEmail		
Created By:	Dave Goh	Last Updated By:	Dave Goh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Allows users to log in using an existing account via email credentials.
Preconditions:	The user must have already registered an account.
Postconditions:	The user gains access to the app's main interface upon successful login.
Priority:	High
Frequency of Use:	1
Flow of Events:	1. User selects "Login with Email" in the user interface. 2. User enters an email and password. 3. System validates credentials with Firebase authentication. 4. Users are granted access upon successful authentication.
Alternative Flows:	NIL
Exceptions:	1. Incorrect password. 2. Username does not exist.
Includes:	Firebase authentication API
Special Requirements:	Integration with Firebase Authentication.
Assumptions:	User has already created an account before.
Notes and Issues:	None

2. LoginWGoogle

Use Case ID:	2		
Use Case Name:	LoginWGoogle		
Created By:	Ashwin Suresh	Last Updated By:	Ashwin Suresh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Enables users to log in using their Google account credentials.
Preconditions:	The user must have a valid Google account.
Postconditions:	The user gains access to the app's main interface upon successful login.
Priority:	High
Frequency of Use:	Once per session
Flow of Events:	1. User selects "Login with Google" in the user interface. 2. User is prompted to grant app permissions to access their Google account. 3. System authenticates user via Google authentication. 4. User logs in successfully.
Alternative Flows:	NIL
Exceptions:	User denies app permission to access their Google account.
Includes:	
Special Requirements:	Integration with Google Authentication.
Assumptions:	User has a valid and accessible Google account.
Notes and Issues:	None

3. CreateAccount

Use Case ID:	3

Use Case Name:	CreateAccount		
Created By:	Ashwin Suresh	Last Updated By:	Ashwin Suresh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Allows new users to create an account using their email.
Preconditions:	The email must not be registered in the system.
Postconditions:	A new account is created, and the user can log in with their credentials.
Priority:	High
Frequency of Use:	Typically used once by new users
Flow of Events:	1. User selects "Create Account" in the user interface. 2. User enters a valid email, password, and confirms the password. 3. System checks for existing account using Firebase API. 4. Account is created successfully if email is unique.
Alternative Flows:	NIL
Exceptions:	1. Email already exists. 2. Passwords do not match.
Includes:	Firebase API
Special Requirements:	Integration with Firebase.
Assumptions:	1. User provides a valid email. 2. User inputs matching passwords.
Notes and Issues:	None

4. LoadMap

Use Case ID:	4		
Use Case Name:	LoadMap		
Created By:	Ashwin Suresh	Last Updated By:	Ashwin Suresh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Loads the map interface showing the user's current location.
Preconditions:	User must have granted location permissions.
Postconditions:	The map displays the user's location.
Priority:	High
Frequency of Use:	Every time the map is loaded
Flow of Events:	1. User is prompted to grant location permission if not already granted. 2. User grants permission. 3. Map is loaded and centred on the user's current location. 4. User's location is indicated on the map.
Alternative Flows:	NIL
Exceptions:	User denies location permission. No network connectivity.
Includes:	TomTomAPI
Special Requirements:	Location permission must be granted
Assumptions:	The user has internet access
Notes and Issues:	None

5. QueryNearbyCarpark

Use Case ID:	5

Use Case Name:	QueryNearbyCarpark		
Created By:	Dave Goh	Last Updated By:	Dave Goh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	System
Description:	Queries and displays nearby carparks based on the user’s location.
Preconditions:	Network connectivity and location permissions are granted.
Postconditions:	The user can view nearby carparks with details.
Priority:	High
Frequency of Use:	Frequently used when searching for parking.
Flow of Events:	1. System queries URA Carpark API for nearby carpark information based on the user’s location. 2. Car Parks are displayed on the map in proximity to the user’s location. 3. Car Park information such as distance, rate, EV charging capability, and capacity are displayed.
Alternative Flows:	NIL
Exceptions:	1. No network connectivity. 2. Location permissions are not granted.
Includes:	1. URA Carpark API 2. TomTom API
Special Requirements:	Integration with URA and TomTom APIs.
Assumptions:	User has internet access.
Notes and Issues:	None

6. ReportBug

Use Case ID:	6		
Use Case Name:	ReportBug		
Created By:	Ashwin Suresh	Last Updated By:	Ashwin Suresh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Allows users to submit a bug report.
Preconditions:	User must be logged in.
Postconditions:	Bug report is saved in the database for review.
Priority:	High
Frequency of Use:	Used occasionally.
Flow of Events:	1. User navigates to the settings menu. 2. User selects the "Report Bug" option. 3. User enters the bug description in a text field. 4. User submits the bug report. 5. Report is stored in the database.
Alternative Flows:	NIL
Exceptions:	1. Text field contains invalid characters. 2. Report exceeds 1000 characters. 3. No network connectivity.
Includes:	Firebase API
Special Requirements:	Integration with Firebase.
Assumptions:	User has internet access.
Notes and Issues:	None

7. ApplyPreferences

Use Case ID:	7		
Use Case Name:	ApplyPreferences		
Created By:	Dave Goh	Last Updated By:	Dave Goh
Date Created:	17/09/24	Date Last Updated:	17/09/24

Actor:	User
Description:	Allows users to modify their app preferences.
Preconditions:	Users must be on the preferences page.
Postconditions:	Changes are saved to the user's profile in the database.
Priority:	High
Frequency of Use:	Used frequently to adjust preferences.
Flow of Events:	1. User navigates to the preferences page. 2. Users modify their preferences. 3. System updates the changes in the user database.
Alternative Flows:	NIL
Exceptions:	No network connectivity.
Includes:	None
Special Requirements:	None
Assumptions:	User has internet access
Notes and Issues:	None

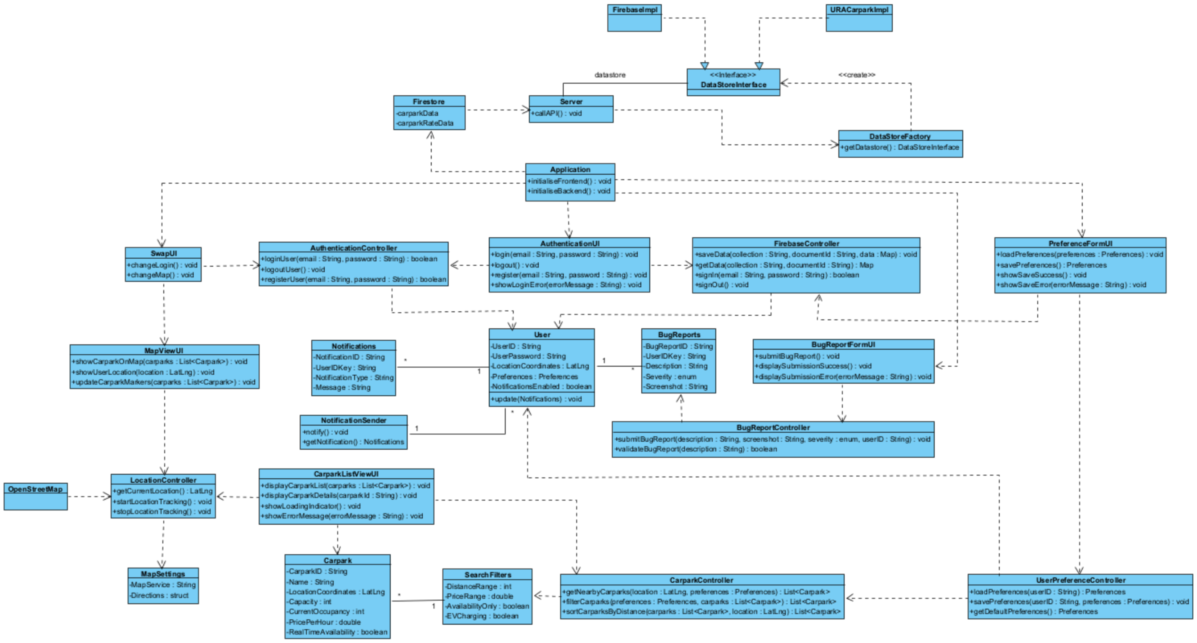
8. Logout

Use Case ID:	8		
Use Case Name:	Logout		
Created By:	Dave Goh	Last Updated By:	Dave Goh
Date Created:	17/09/24	Date Last Updated:	17/09/24

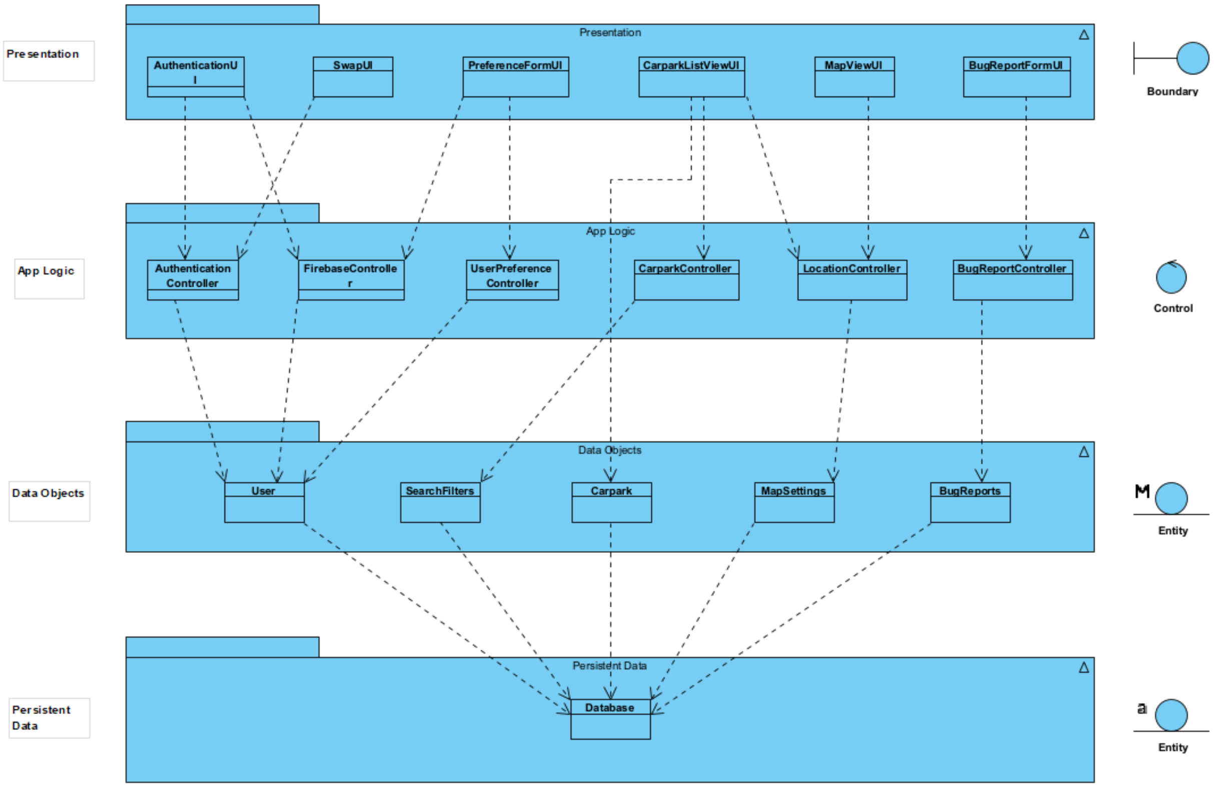
Actor:	User
Description:	Logs the user out of the app.
Preconditions:	The user must be logged in.
Postconditions:	User is redirected to the login screen.
Priority:	High
Frequency of Use:	Frequently used at the end of a session.
Flow of Events:	1. User selects "Logout." 2. Application logs out the user and redirects them to the login page.
Alternative Flows:	NIL
Exceptions:	No network connectivity.
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

4.Design Model

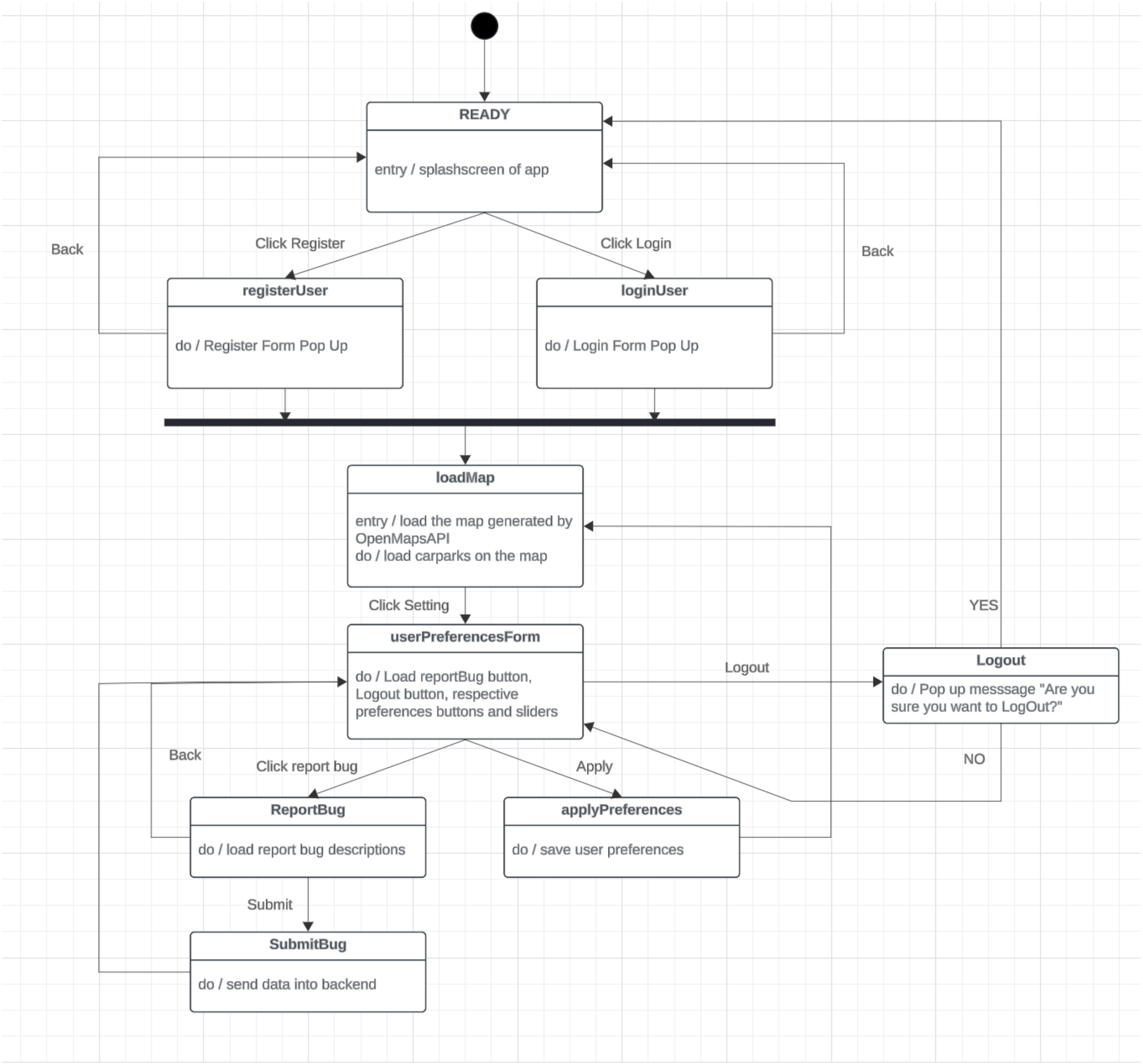
4.1 Class Diagram


FirebaseImpl
URACarparkImpl
datastore
<<Interface>>
DataStoreInterface
<<create>>
Firestore
-carparkData
-carparkRateData
Server
+callAPI() : void
DataStoreFactory
+getDatastore() : DataStoreInterface
Application
+initialiseFrontend() : void
+initialiseBackend() : void
SwapUI
+changeLogin() : void
+changeMap() : void
AuthenticationController
+loginUser(email : String, password : String) : boolean
+logoutUser() : void
+registerUser(email : String, password : String) : boolean
AuthenticationUI
+login(email : String, password : String) : void
+logout() : void
+register(email : String, password : String) : void
+showLoginError(errorMessage : String) : void
FirebaseController
+saveData(collection : String, documentId : String, data : Map) : void
+getData(collection : String, documentId : String) : Map
+signIn(email : String, password : String) : boolean
+signOut() : void
PreferenceFormUI
+loadPreferences(preferences : Preferences) : void
+savePreferences() : Preferences
+showSaveSuccess() : void
+showSaveError(errorMessage : String) : void
MapViewUI
+showCarparkOnMap(carparks : List<Carpark>) : void
+showUserLocation(location : LatLng) : void
+updateCarparkMarkers(carparks : List<Carpark>) : void
Notifications
-NotificationID : String
-UserIDKey : String
-NotificationType : String
-Message : String
*
1
User
-UserID : String
-UserPassword : String
-LocationCoordinates : LatLng
-Preferences : Preferences
-NotificationsEnabled : boolean
+update(Notifications) : void
1
*
BugReports
-BugReportID : String
-UserIDKey : String
-Description : String
-Severity : enum
-Screenshot : String
BugReportFormUI
+submitBugReport() : void
+displaySubmissionSuccess() : void
+displaySubmissionError(errorMessage : String) : void
NotificationSender
+notify() : void
+getNotification() : Notifications
1
*
BugReportController
+submitBugReport(description : String, screenshot : String, severity : enum, userID : String) : void
+validateBugReport(description : String) : boolean
OpenStreetMap
LocationController
+getCurrentLocation() : LatLng
+startLocationTracking() : void
+stopLocationTracking() : void
CarparkListViewUI
+displayCarparkList(carparks : List<Carpark>) : void
+displayCarparkDetails(carparkId : String) : void
+showLoadingIndicator() : void
+showErrorMessage(errorMessage : String) : void
MapSettings
-MapService : String
-Directions : struct
Carpark
-CarparkID : String
-Name : String
-LocationCoordinates : LatLng
-Capacity : int
-CurrentOccupancy : int
-PricePerHour : double
-RealTimeAvailability : boolean
*
1
SearchFilters
-DistanceRange : int
-PriceRange : double
-AvailabilityOnly : boolean
-EVCharging : boolean
CarparkController
+getNearbyCarparks(location : LatLng, preferences : Preferences) : List<Carpark>
+filterCarparks(preferences : Preferences, carparks : List<Carpark>) : List<Carpark>
+sortCarparksByDistance(carparks : List<Carpark>, location : LatLng) : List<Carpark>
UserPreferenceController
+loadPreferences(userID : String) : Preferences
+savePreferences(userID : String, preferences : Preferences) : void
+getDefaultPreferences() : Preferences

4.2 System Architecture

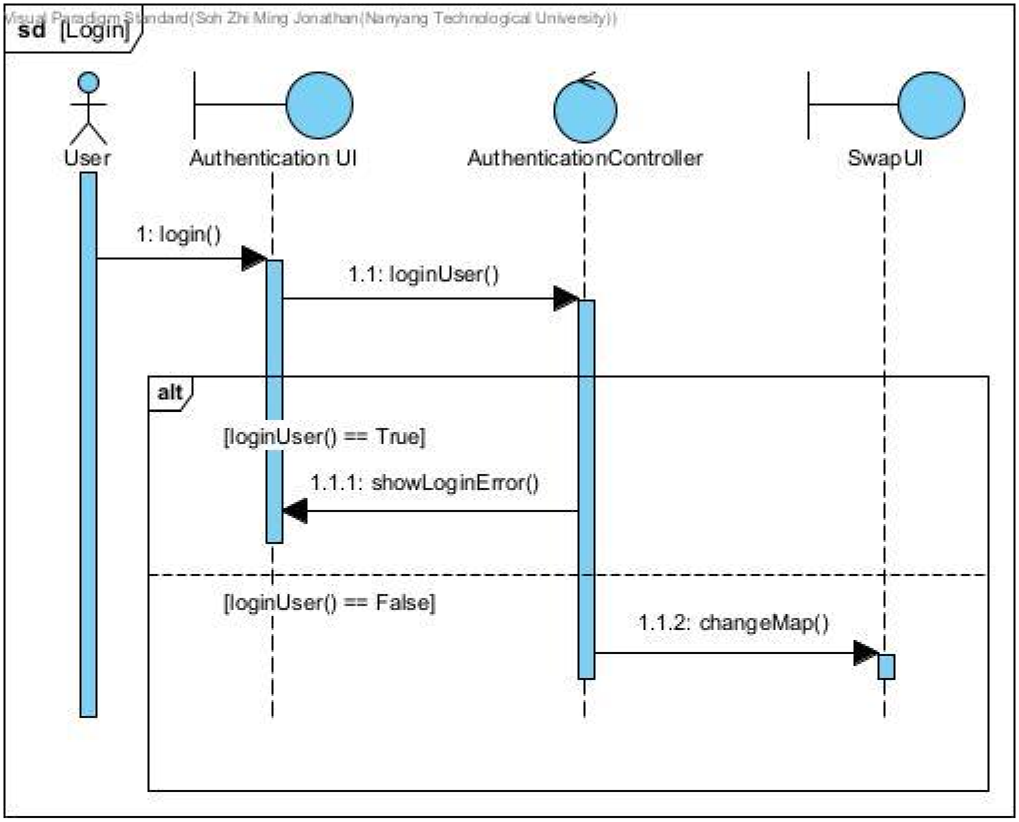


4.3 Dialog Map

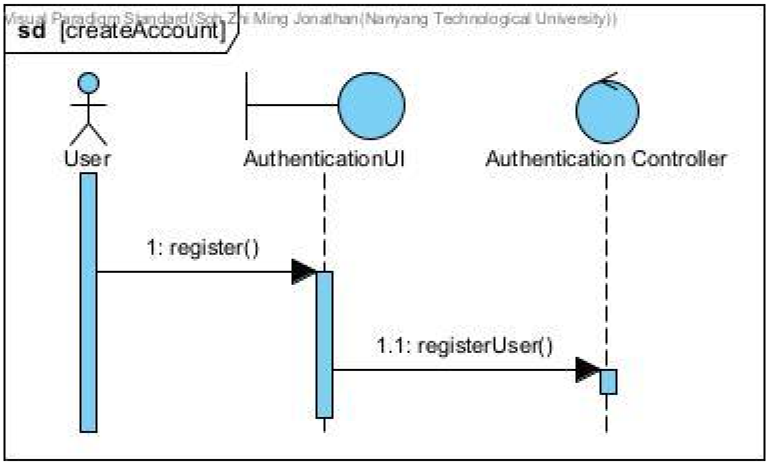

READY
entry / splashscreen of app
Back
Click Register
Click Login
Back
registerUser
do / Register Form Pop Up
loginUser
do / Login Form Pop Up
loadMap
entry / load the map generated by OpenMapsAPI
do / load carparks on the map
Click Setting
YES
userPreferencesForm
do / Load reportBug button, Logout button, respective preferences buttons and sliders
Logout
Logout
do / Pop up messsage "Are you sure you want to LogOut?"
Back
Click report bug
Apply
NO
ReportBug
do / load report bug descriptions
applyPreferences
do / save user preferences
Submit
SubmitBug
do / send data into backend

4.4 Sequence Diagram for Use Cases

4.4.1. Login

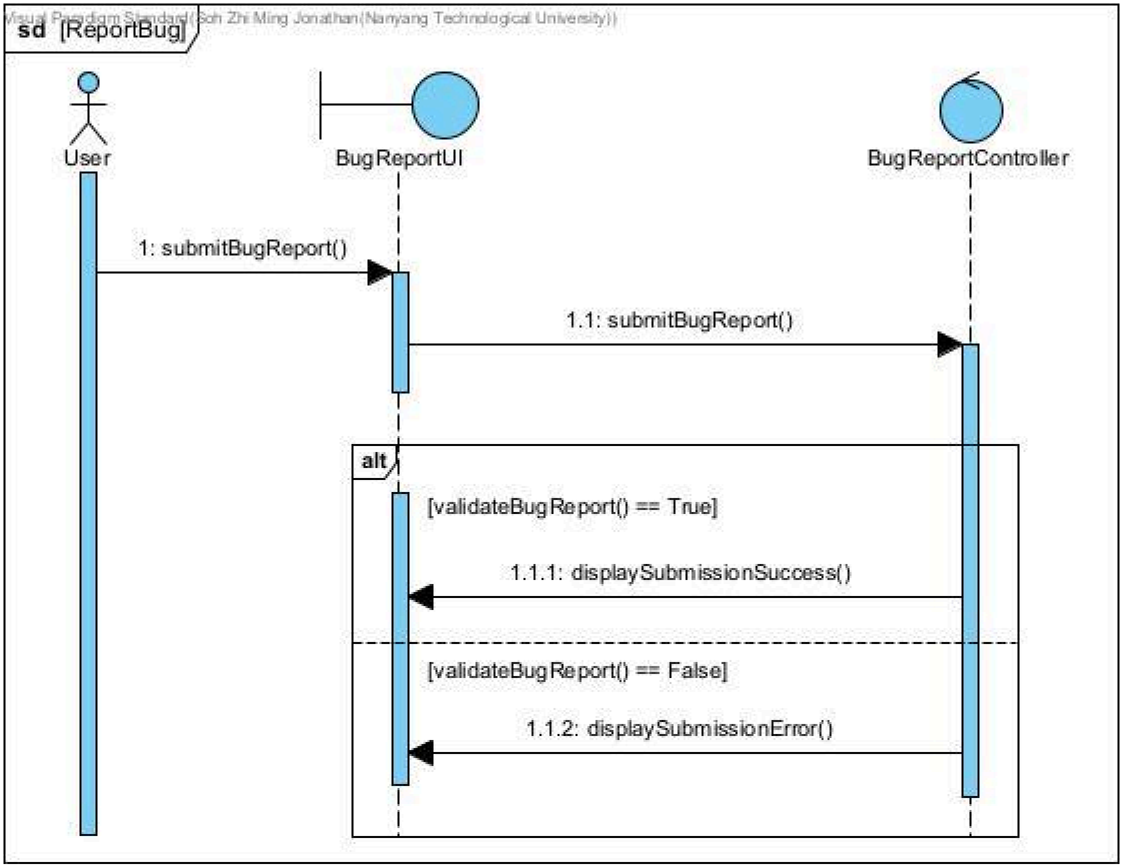


4.4.2. createAccount

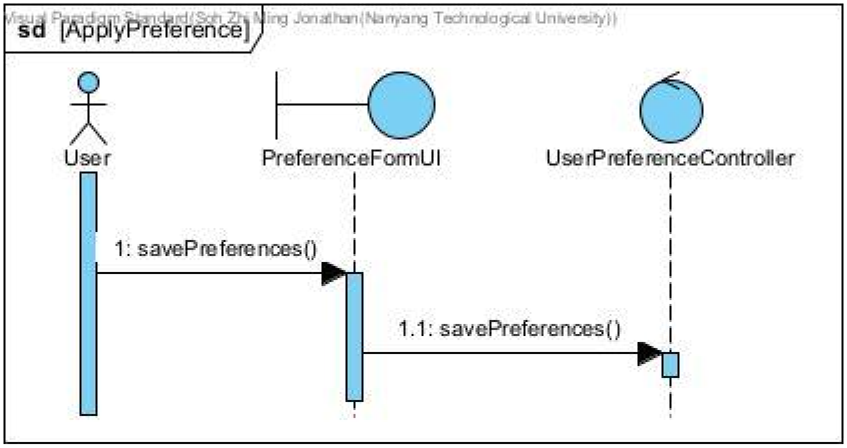


Visual Paradigm Standard(Soh Zhi Ming Jonathan(Nanyang Technological University))
sd [createAccount]
User
AuthenticationUI
Authentication Controller
1: register()
1.1: registerUser()

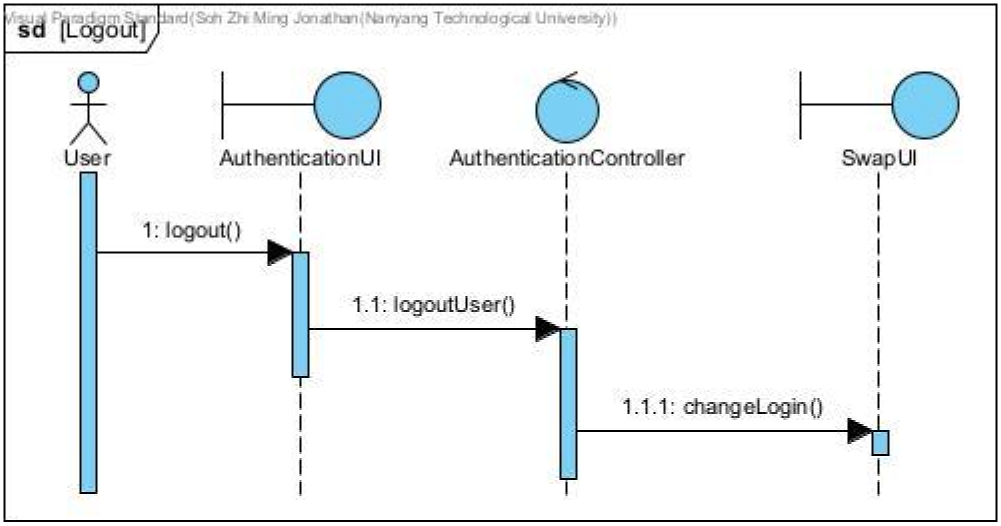
4.4.3 ReportBug


Visual Paradigm Standard(Soh Zhi Ming Jonathan(Nanyang Technological University))
sd [ReportBug]
User
BugReportUI
BugReportController
1: submitBugReport()
1.1: submitBugReport()
alt
[validateBugReport() == True]
1.1.1: displaySubmissionSuccess()
[validateBugReport() == False]
1.1.2: displaySubmissionError()

4.4.4 ApplyPreference



4.4.5 LogOut



5.Key Design Issue

5.1 Design constraints

- API handling required a lot of computational power due to the sheer amount of data given out per call. Furthermore, the API requires re-validation every 24 hours which meant to combat this issue, we offloaded the scheduling and computational requirements into a Docker container that handles parsing, validation and writing to the database.

- API formatting for location was in SVY21 format, which is localized to Singapore. However, our Application required a conversion to WGS84 for every instance of a carpark, which required further computational power.

5.2 Design patterns

In the Design Model, two new design patterns were added: The Factory Pattern and the Observer Pattern with a Push-Update Notification mechanism.

- The Factory Pattern was implemented to add backend capabilities of the Application. The Application takes Carpark and Carpark Rate data from the database Firestore, while a server in the backend calls the URA Carpark and Firebase APIs and pushes the data into Firestore. The Factory Pattern decouples class selection between the implementation classes for URA Carpark and Firebase APIs (URACarparkImpl and FirebaseImpl respectively) and ensures instantiation of either implementation class at runtime.

- The Observer Pattern was implemented in the Notification system, as a NotificationSender class was added to provide capabilities to send notifications to users (observers). The Push mechanism was preferred here because users need not call back to see the full messages in each notification. While the types of notifications to send were not detailed thoroughly, it can be assumed that most notifications could provide information that would be important to most users, such as changes in carpark details or availability that could be useful for commuters who frequently use carparks and may need to visit particularly crowded places. The Push mechanism would be useful in such cases since information would be more readily available to users.

6. Application Skeleton