NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC2006 - Software Engineering

All lab deliverables (combined)

Lab Group	SCS4
Team	SmartCommute
Members	Hoang Viet Thinh (U2420343J) Darrell Ma Wei Ze (U2222176G) Ivan Wong Jia Ming (U2422775A) Aster Wang Ziran (U2421510F) Jayaraj Kishore Kumar (U2423796B) Jarrett Goh Xiang Zheng (U2422600H)

Table of contents

1) Documentation of functional and non-functional requirements

A) Functional Requirements

1. **User Account**:

1.1. SmartCommute allows users to access basic features without creating account (anonymous mode):

1.1.1. If user is in anonymous mode, SmartCommute shall allow accessing #3 (General Information of a Station)

1.1.2. It allows access to accessibility mode, except for the saving preference paths feature.

1.2. SmartCommute allows users to create account
    1.2.1. Registration requires email and password input
    1.2.2. SmartCommute displays error message if email format is invalid (does not match the regex)
    1.2.3. SmartCommute displays error message if email already exists in the database
    1.2.4. SmartCommute displays error message when password's length < 8 characters
    1.2.5. When all registration requirements are met, SmartCommute shall hash the password and store user credentials in the database.
    1.2.6. After successful registration, SmartCommute redirects user to log-in page
1.3. SmartCommute provides logged-in users with personalized features
    1.3.1. If users logged in, they can use all features in anonymous mode
    1.3.2. If users logged in, they can add their preference paths to the dashboard for fast access, SmartCommute will save it into the database
    1.3.3. SmartCommute shall remember user preference of Accessibility Mode

2. Data filter option

2.1 SmartCommute allows user to filter the data by typing content into a search box

2.1.1 User can enter the name of a station, to filter data for real-time, forecast and lift maintenance data for that specific station

2.1.2 Autocomplete feature should provide suggestions as the user is typing. This autocomplete feature will be provided by an external API service.

2.1.3 If the user enters a station name that is not recognized by the autocomplete service, the application shall display a "No results found" message

2.2 SmartCommute has an option button menu for real-time and forecast crowd-level filter

        2.2.1 The available status options shall be: green(low), yellow (moderate), red(high)
        2.2.2 Selecting a category shall update the display to show only the stations that match the chosen status
        2.2.3 The user shall be able to switch between the real-time and forecast modes using the same menu

    2.3 SmartCommute shall provide a filter to allow users to view only stations or lines affected by service breakdowns
        2.3.1 A toggle button shall be provided for the filter. When enabled, the display shall isolate and highlight affected stations/lines
        2.3.2 When the toggle is disabled, all stations/lines shall be displayed according to the other active filters
        2.3.3 If no breakdowns are currently detected, enabling the toggle shall display the message “There are no breakdowns at the moment.”

    2.4 SmartCommute shall provide a reset filter button that clears all applied filters
        2.4.1 Selecting the reset button shall remove any search terms, crowd-level filters, and breakdown filters currently active
        2.4.2 After reset, the application shall return to displaying all available data by default
        2.4.3 The reset button shall be present on the same screen as the other filter option buttons, without any additional navigation

3. Station General Information Display

SmartCommute allows Users to view comprehensive information for individual MRT stations when Users click on that station:

3.1. SmartCommute displays real-time crowdedness status
- 3.1.1. If the crowd level data is available, SmartCommute shows crowd level with corresponding color indicators: green(low), yellow (moderate), red(high)
- 3.1.2. If the crowd level data is not available, display NA

3.2. SmartCommute shows the forecast of crowdedness in a 30-minute interval
- 3.2.1. If the forecast data is available, SmartCommute shows crowd level with corresponding color indicators: green(low), yellow (moderate), red(high)

        3.2.2. If the forecast data is not available, display NA

    3.3. If there is service interruption, SmartCommute displays the "message" attribute from the data to get specific details about the circumstance.

    3.4. If there is lift under maintenance, SmartCommute displays maintenance information:

        3.4.1. Display the "LiftId" attribute to show the specific lift

        3.4.2. Display the "LiftDesc" attribute to provide a detailed description

        3.4.3. If Accessibility Mode is active, highlight outage and add extra time to travel route

4. Notifications and Alerts

4.1 SmartCommute shall provide users with notifications and alerts for train disruptions, route re-planning or lift outages, on the dashboard page
4.2 If the user is in anonymous mode, the alerts will only be displayed at the relevant stations
4.3 If the user is logged in, SmartCommute shall provide additional alert management features:
4.3.1 Subscribe to alerts for specific stations or routes
4.3.2 Enable push notification in app where notifications will include timestamp and events at the relevant stations
4.3.3 System will automatically remove expired or resolved notifications

B) <u>Non-functional Requirements</u>

Usability	1. General layout of the User-Interface (UI) ● The general layout and positioning of the User-Interface (UI) elements such as: data filter options and stations information

	display, shall allow users to locate and access any primary function within **3 seconds of viewing the main screen**. 2. Mobile device responsiveness ● The application shall support both desktop and mobile device screen resolutions ● Elements shall automatically resize or reposition to fit the screen without **overlapping, truncation, horizontal scrolling, or layout distortion** on common devices
Reliability	The system shall remain operational and display the most recently cached data when external APIs are temporarily unavailable, instead of showing errors or blank screens. Once the APIs become available again, the system shall automatically refresh with the latest data.
Performance	The system shall return filtered results (search box, crowd-level filter, breakdown filter) within **2 seconds for average user queries**, defined as those containing **one or two filter criteria** and **no more than one keyword**.
Supportability	The database must be replaceable with any commercial product supporting standard SQL queries.

2) Data Dictionary

Term	Definition
User	A person who uses the SmartCommute application to access crowd-level data, plan journeys, and manage preferences.
Account	A registered user's personal profile associated with SmartCommute, created through email and password registration.
Station’s General Information	Information related to an individual MRT station, including real-time crowd density, forecast of crowdedness, service interruptions and lift maintenance details.
Filter Options	Parameters that allow users to refine and limit the displayed transport data (e.g. crowd-level, station information and breakdown status) according to specific criteria.
Anonymous Mode	Allows users to access basic features (general information of station) of SmartCommute without creating an account.
Accessibility Mode	User mode provides the additional feature to highlight lift maintenance/outages, and automatically adds extra time to travel routes.
Preference paths	Saved paths for quick access to frequently used station information in SmartCommute.
Alerts	Notifications provided by SmartCommute for train disruptions, route re-planning, or lift outages,

	displayed based on user mode.

3) Use Case Model (Use Case Diagram + Use Case Descriptions)

A) Use Case Diagram

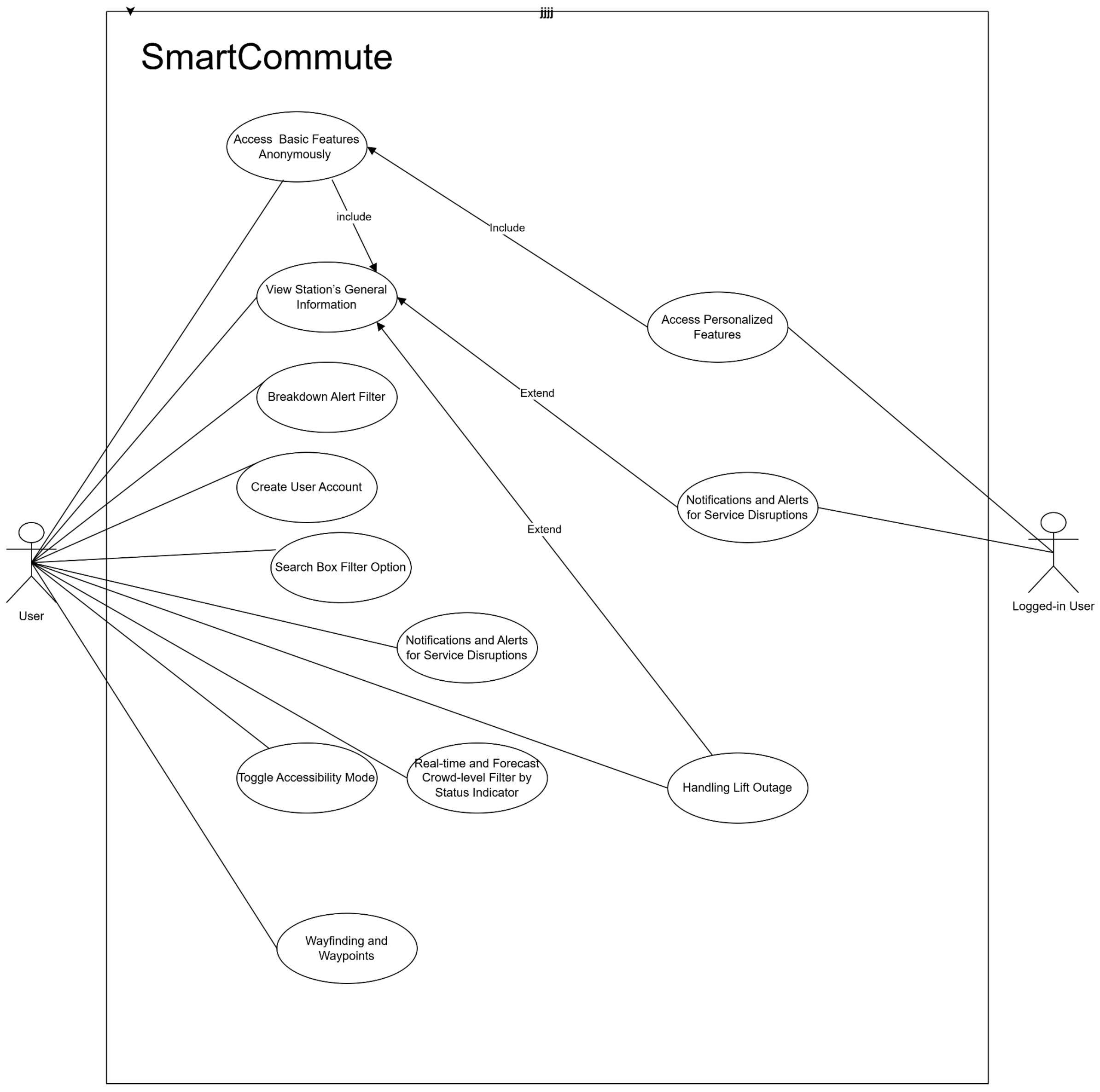

jjjj
SmartCommute
Access Basic Features Anonymously
include
Include
View Station's General Information
Access Personalized Features
Breakdown Alert Filter
Extend
Create User Account
Notifications and Alerts for Service Disruptions
Extend
Search Box Filter Option
User
Logged-in User
Notifications and Alerts for Service Disruptions
Toggle Accessibility Mode
Real-time and Forecast Crowd-level Filter by Status Indicator
Handling Lift Outage
Wayfinding and Waypoints

Diagram Access Link:
https://drive.google.com/file/d/1TNwJTmNh3ouxDIGFX8opWkzzPQrZIxb4/view?usp=sharing

B) Use Case Descriptions

Use Case 1.1

Use Case ID:	#1.1		
Use Case Name:	Access Basic Features Anonymously		
Created By:	Hoang Viet Thinh	Last Updated By:	Hoang Viet Thinh
Date Created:	8/9/2025	Date Last Updated:	19/9/2025

Actor:	User (Commuter)
Description:	Allows users to access basic features of SmartCommute without creating an account in anonymous mode
Preconditions:	User has access to the SmartCommute interface
Postconditions:	User shall view all basic features without data saved
Priority:	High
Frequency of Use:	Multiple times daily
Flow of Events:	1.User accesses SmartCommute 2.System displays main interface without requiring login 3.User shall access basic features listed in #3, which are: real-time crowd density, 30-minute forecast crowdedness, getting interruption service and lift maintenance alerts. 4.User shall access accessibility mode without saving data
Alternative Flows:	1.1.AC.1: If user attempts to save preferences, system denies action and prompts for account login
Exceptions:	1.1EX.1: Internet Connection is lost, system displays re-connect request
Includes:	#3 (Station General Information Display)
Special Requirements:	None
Assumptions:	User has internet connection
Notes and Issues:	None

Use Case: 1.2

Use Case ID:	#1.2		
Use Case Name:	Create User Account		
Created By:	Hoang Viet Thinh	Last Updated By:	Hoang Viet Thinh
Date Created:	8/9/2025	Date Last Updated:	16/9/2025

Actor:	User (Commuter)
Description:	Allows users to create an account providing email and password
Preconditions:	1.User is on SmartCommute registration page 2.Database connection is available
Postconditions:	1.User credentials are hashed and stored in the database 2.User is re-directed to the login page
Priority:	High
Frequency of Use:	Once per user
Flow of Events:	1.User navigates to registration page 2.System prompts for email and password input 3.User enters email and password 4.System validates email 5.System check password's length 6.If valid, system hashes password and stores credentials in database 7.System re-directs user to login page
Alternative Flows:	1.2AC.1: If user cancels registration, system returns to homepage
Exceptions:	1.2EX.1: If email is invalid, system displays error message: "The email format is invalid" 1.2EX.2: If email already exists, system displays error message: "The email is used by another user" 1.2EX.3: If password length < 8 characters, system displays error message: "Your password must have >= 8 characters".
Includes:	None

Special Requirements:	Hashing algorithm for password must be a secure algorithm
Assumptions:	User provides accurate email and password
Notes and Issues:	None

Use Case 1.3

Use Case ID:	#1.3		
Use Case Name:	Access Personalized Features		
Created By:	Hoang Viet Thinh	Last Updated By:	Hoang Viet Thinh
Date Created:	8/9/2025	Date Last Updated:	21/10/2025

Actor:	Logged-in User
Description:	Access personalized features after login
Preconditions:	User has successfully logged in with valid credentials
Postconditions:	User shall access all personalized features and their saved preference paths data
Priority:	Medium
Frequency of Use:	Multiple times daily by logged-in users
Flow of Events:	1.User logs in with credentials 2.System authenticates information 3.System displays the dashboard with all available features and user's saved preference paths 4.User shall add preference path to the dashboard 5.System saved the preference data into database for future session
Alternative Flows:	1.3.AC.1: If user removes a preference, system updates dashboard accordingly
Exceptions:	1.3.EX.1: If the credential does not match with the database, display error message : “Your email or password is incorrect”
Includes:	#1-1
Special Requirements:	System must ensure secure session management
Assumptions:	None
Notes and Issues:	None

Use Case 1.4

Use Case ID:	#1.4		
Use Case Name:	Remember Accessibility Mode toggled on or off		
Created By:	Darrell Ma	Last Updated By:	Darrell Ma
Date Created:	08/09/2025	Date Last Updated:	08/09/2025

Actor:	User (Commuter)
Description:	User toggles Accessibility Mode during journey planning and the system saves the preference of the user for future sessions.
Preconditions:	1. User has opened the journey planner 2. SmartCommute is able to read user profile
Postconditions:	Accessibility Mode preference is remembered and stored, which will be applied for future journey planning.
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. User opens journey planner 2. System displays option to toggle Accessibility Mode 3. User toggles on Accessibility Mode 4. System remembers the toggle 5. System confirms toggle is activated
Alternative Flows:	3a. User toggles off Accessibility Mode
Exceptions:	None
Includes:	None
Special Requirements:	Toggle must be clearly visible and accessible
Assumptions:	None
Notes and Issues:	None

Use Case 2.1

Use Case ID:	#2.1		
Use Case Name:	Search box filter option		
Created By:	Jarrett Goh	Last Updated By:	Jarrett Goh

Date Created:	8/9/2025	Date Last Updated:	8/9/2025

Actor:	User (Commuter)
Description:	Allows user to filter the current display data by entering a value into the search box
Preconditions:	User has access to the SmartCommute interface
Postconditions:	1. System shall provide autocomplete suggestions as the user is typing 2. Users shall only be able to view the data with values that matches the value entered into the search box 3. Users shall not be able to view the data with irrelevant fields (with data parameters that does not match the value the user has entered into the search)
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. User clicks on the search box container 2. User enters a search value 3. System will apply filter to the data being displayed, and output the relevant data 4. If there is no matching data, the system will display a message “No matching data found”.
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	Autocomplete suggestions API is up to date
Notes and Issues:	Credibility and accuracy of the API

Use Case 2.2

Use Case ID:	#2.2		
Use Case Name:	Real-time and forecast crowd-level filter by status indicator		
Created By:	Jarrett Goh	Last Updated By:	Jarrett Goh
Date Created:	8/9/2025	Date Last Updated:	8/9/2025

Actor:	User (Commuter)

Description:	Allows user to filter the current display data by the current status of the crowd-level for both real-time and forecast data
Preconditions:	User has access to the SmartCommute interface
Postconditions:	The displayed data shall be immediately updated to reflect all the data with matching status
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. User selects a particular status indicator filter value 2. The displayed data shall be updated immediately 3. If there is no matching data, the system shall display a message "No matching data found".
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

Use Case 2.3

Use Case ID:	#2.3		
Use Case Name:	Breakdown alert filter		
Created By:	Jarrett Goh	Last Updated By:	Jarrett Goh
Date Created:	8/9/2025	Date Last Updated:	8/9/2025

Actor:	User (Commuter)
Description:	Allows users to enable a filter to view an isolated display of all the stations/lines affected by a particular breakdown.
Preconditions:	User has access to the SmartCommute interface
Postconditions:	The displayed data shall be immediately updated to reflect all the data matching the filter
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. User clicks the toggle button 2. If filter is inactive, immediately update the displayed data to isolate the affected stations/lines 3. If there is no matching data, the system will display the message "There are no breakdowns at the moment"

	4. If is already active, remove the filter, and display all the data without the filter
Alternative Flows:	None
Exceptions:	None
Includes:	None
Special Requirements:	None
Assumptions:	None
Notes and Issues:	None

Use Case 3.1

Use Case ID:	#3.1		
Use Case Name:	View Station's General Information		
Created By:	Hoang Viet Thinh	Last Updated By:	Hoang Viet Thinh
Date Created:	8/9/2025	Date Last Updated:	16/9/2025

Actor:	User (Commuter)
Description:	Allows users to view comprehensive information for individual MRT stations when clicking on a station
Preconditions:	User has access to SmartCommute interface and selects a station
Postconditions:	Station information modal is displayed
Priority:	High
Frequency of Use:	Multiple times daily
Flow of Events:	1.User clicks on a station 2.System displays real time crowd density with color indicators if data is available, else "NA" 3.System displays 30-minute forecast crowdedness with color indicators, else "NA" 4.If service interruption occurs, system displays message to describe the situation 5. If lift is under maintenance, system displays the lift ID, and a description about the situation
Alternative Flows:	None
Exceptions:	3.EX.1: If API fails, system displays a generic error message
Includes:	None
Special Requirements:	1.Color indicators must be accessible for colorblind users.

	2.Data must be refreshed every 60 seconds
Assumptions:	API data is updated as schedule (update per 10 minute)
Notes and Issues:	Verify API reliability

Use Case 3.2

Use Case ID:	#3.2		
Use Case Name:	Handling Lift Outage		
Created By:	Darrell Ma	Last Updated By:	Darrell Ma
Date Created:	08/09/2025	Date Last Updated:	05/11/2025

Actor:	User (Commuter with Accessibility Mode on)
Description:	When lift outage occurs, system will notify user and provide alternatives, or add travel time
Preconditions:	1. Accessibility Mode toggled on 2. Facilities Maintenance API is integrated
Postconditions:	User receives timely alert and can continue with an alternative route
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. Lift Outage is reported. 2. System matches the outage to the current route of the user. 3. Alert displayed to user with outage details (station name, exit, duration). 4. User taps “Show Alternatives” or similar prompt. 5. System recalculates accessible route options, considering barrier-free paths and expected delay. 6. System displays new results on the route recommendation screen/map interface, showing: - Original route - Alternative route 7. User selects a preferred route option to continue the journey. 8. System updates the active route and estimated arrival time, confirming the change to the user.
Alternative Flows:	None
Exceptions:	False outage reports
Includes:	None
Special Requirements:	1. Alerts include additional travel time 2. Clear alert screen

Assumptions:	Accurate outage data
Notes and Issues:	None

Use Case 4.1

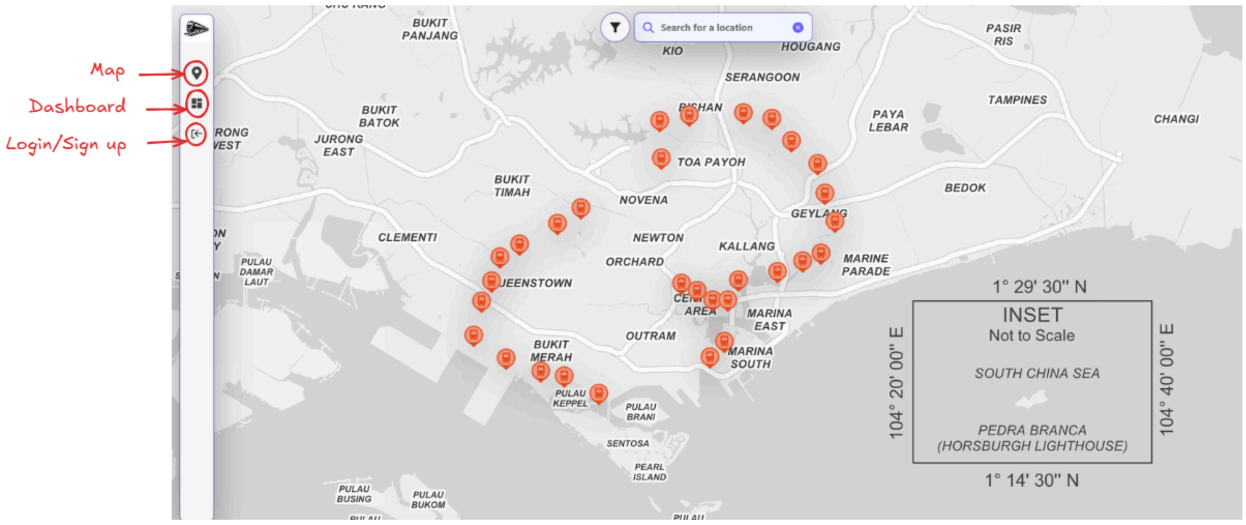
Use Case ID:	#4.1		
Use Case Name:	Pop-up notifications for Service Disruptions		
Created By:	Darrell Ma	Last Updated By:	Darrell Ma
Date Created:	08/09/2025	Date Last Updated:	05/11/2025

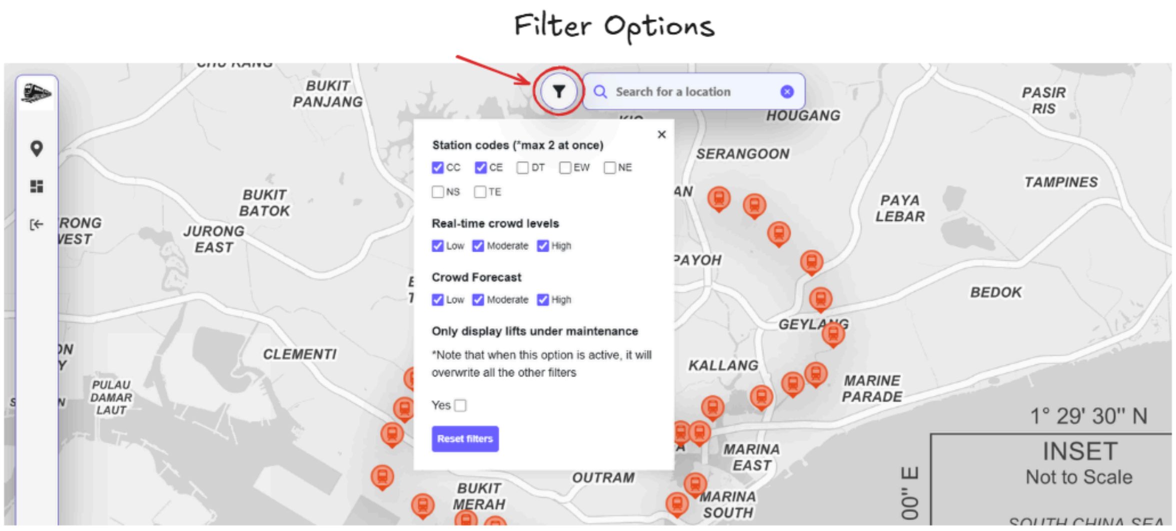
Actor:	User (Anonymous or Logged-in)
Description:	SmartCommute provides real-time notifications for service disruptions, delays, or lift outages. Logged-in users may subscribe to notifications for specific stations or routes and view notifications on their dashboard. Anonymous users can only view notifications while accessing relevant station information pages.
Preconditions:	Train Service Alert API is integrated
Postconditions:	1. Users are made aware of disruptions or outages in a timely manner. 2. Relevant notifications are displayed in the application and can be acknowledged or dismissed.
Priority:	High
Frequency of Use:	Moderate (triggered when disruptions or outages occur)
Flow of Events:	1. System receives disruption or outage data from API 2. System identifies affected stations or routes 3. System generates notification message with timestamp and details 4. Anonymous users: Notification displays on station information page Logged-in users: Notification displays on dashboard via subscribed notification channels 5. User presses the “Acknowledge” or “Dismiss” button to close the notification. 6. System marks the notification as read
Alternative Flows:	None
Exceptions:	API downtime

Includes:	None
Special Requirements:	1. Notifications must include disruption message, timestamp and affected stations 2. Expired notifications must be automatically removed once service resumes
Assumptions:	API data is accurate and updated
Notes and Issues:	None

4) UI Mockup

Map (Home Page)





×

Station codes (*max 2 at once)

[✓ CC] [✓ CE] DT EW NE NS TE

Real-time crowd levels

[✓ Low] [✓ Moderate] [✓ High]

Crowd Forecast

[✓ Low] [✓ Moderate] [✓ High]

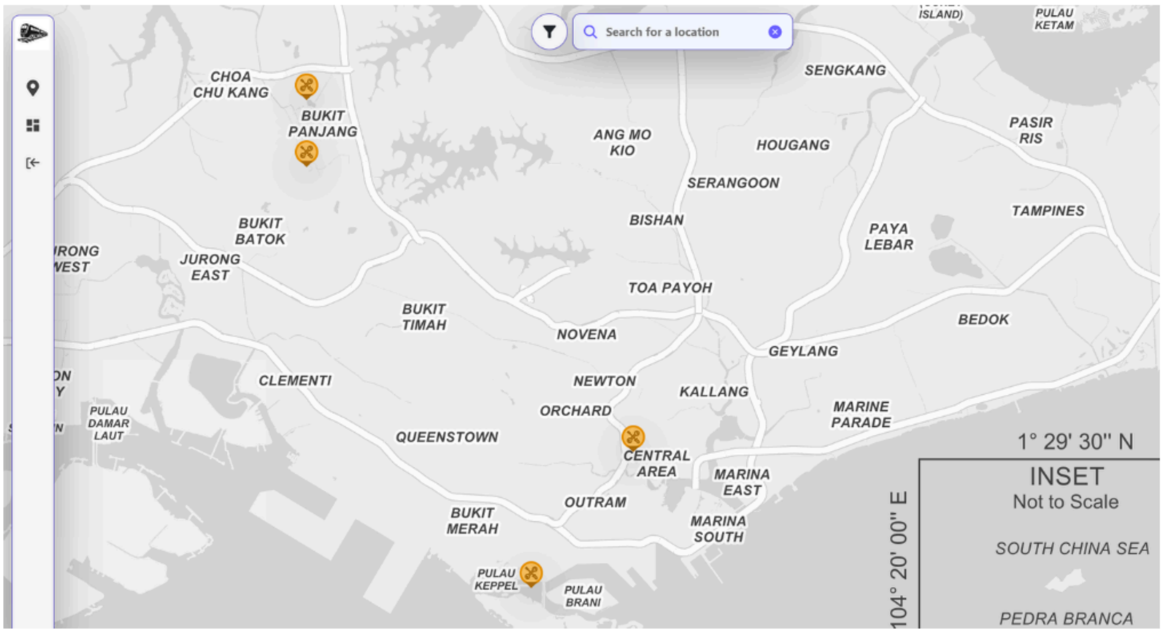
Only display lifts under maintenance

*Note that when this option is active, it will overwrite all the other filters

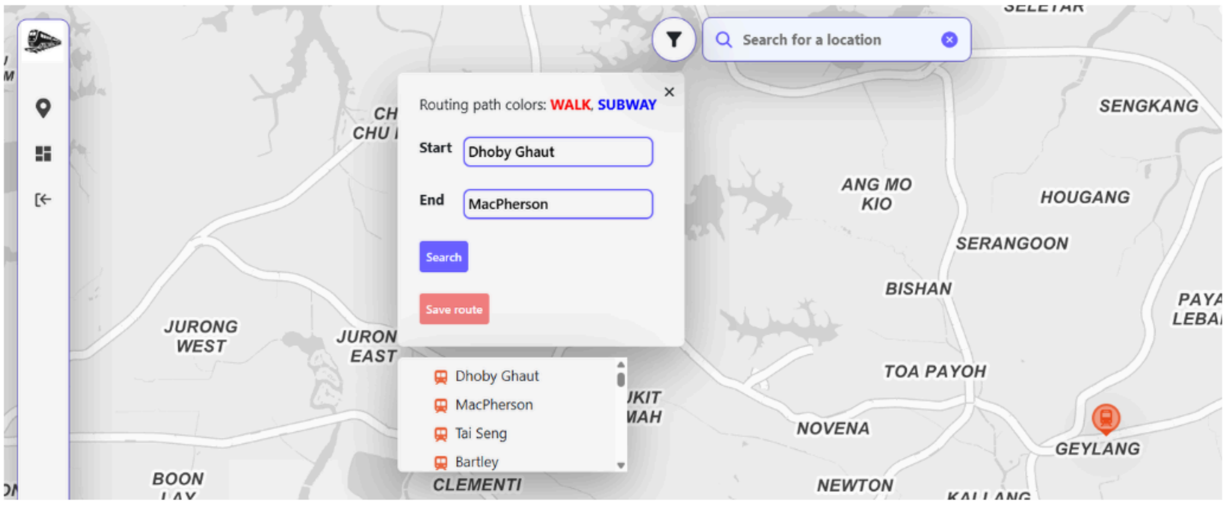
Yes

Reset filters

Example: Searching for stations that are under maintenance



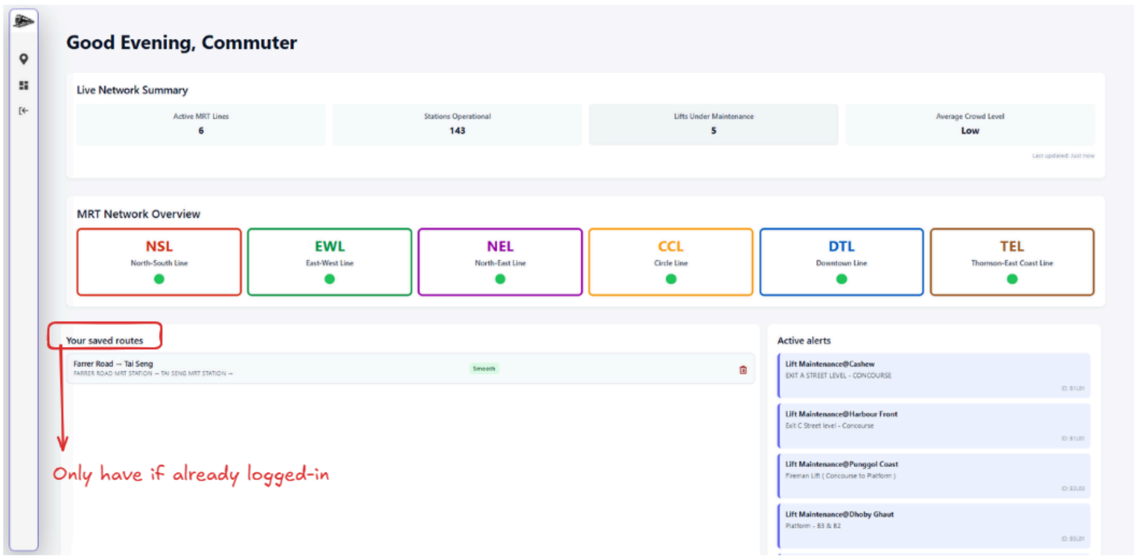
Generate personal preference route (if already logged-in)


Routing path colors: **WALK**, **SUBWAY**

Start: Dhoby Ghaut

End: MacPherson

Dashboard



Good Evening, Commuter

Live Network Summary

Active MRT Lines	Stations Operational	Lifts Under Maintenance	Average Crowd Level
6	143	5	Low

Last updated: Just now

MRT Network Overview

NSL	EWL	NEL	CCL	DTL	TEL
North-South Line	East-West Line	North-East Line	Circle Line	Downtown Line	Thomson-East Coast Line
●	●	●	●	●	●

Your saved routes

Farrer Road → Tai Seng
FARRER ROAD MRT STATION → TAI SENG MRT STATION → Smooth

Only have if already logged-in

Active alerts

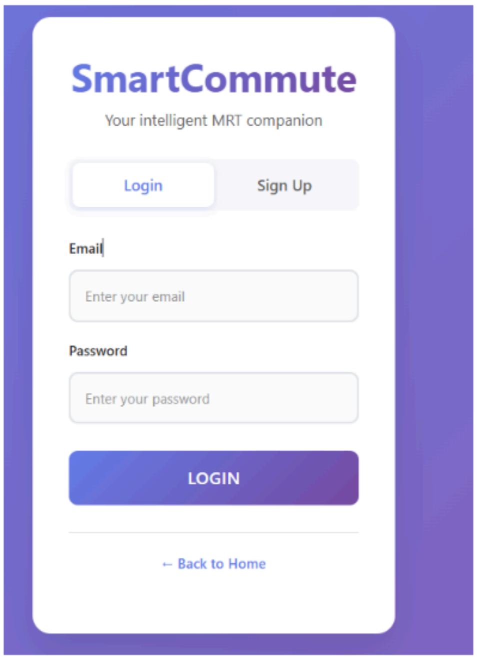
Lift Maintenance@Cashew
EXIT A STREET LEVEL - CONCOURSE

Lift Maintenance@Harbour Front
Exit C Street level - Concourse

Lift Maintenance@Punggol Coast
Fireman Lift (Concourse to Platform)

Lift Maintenance@Dhoby Ghaut
Platform - B3 & B2

Login/Sign Up



SmartCommute

Your intelligent MRT companion

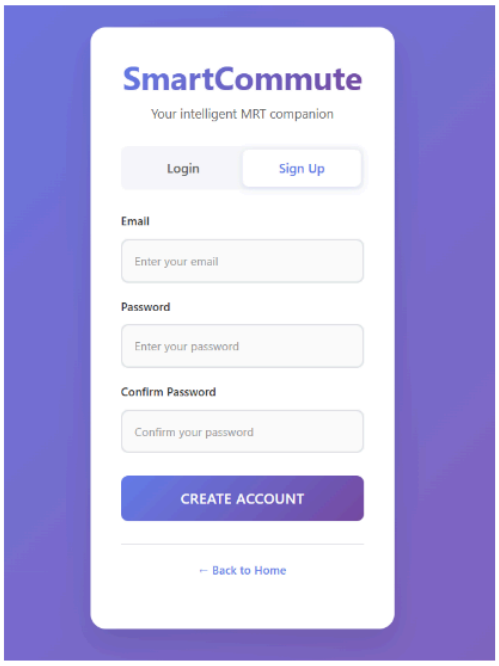
Login | Sign Up

Email: Enter your email

Password: Enter your password

LOGIN

← Back to Home



SmartCommute

Your intelligent MRT companion

Login | Sign Up

Email: Enter your email

Password: Enter your password

Confirm Password: Confirm your password

CREATE ACCOUNT

← Back to Home

MockUp Access Link:
https://excalidraw.com/#json=rbW5ymuIbJiCIKWp9zNoR,ovzmkY8ijlepGYLGR94QhA

5) Class Diagram of Entity Classes

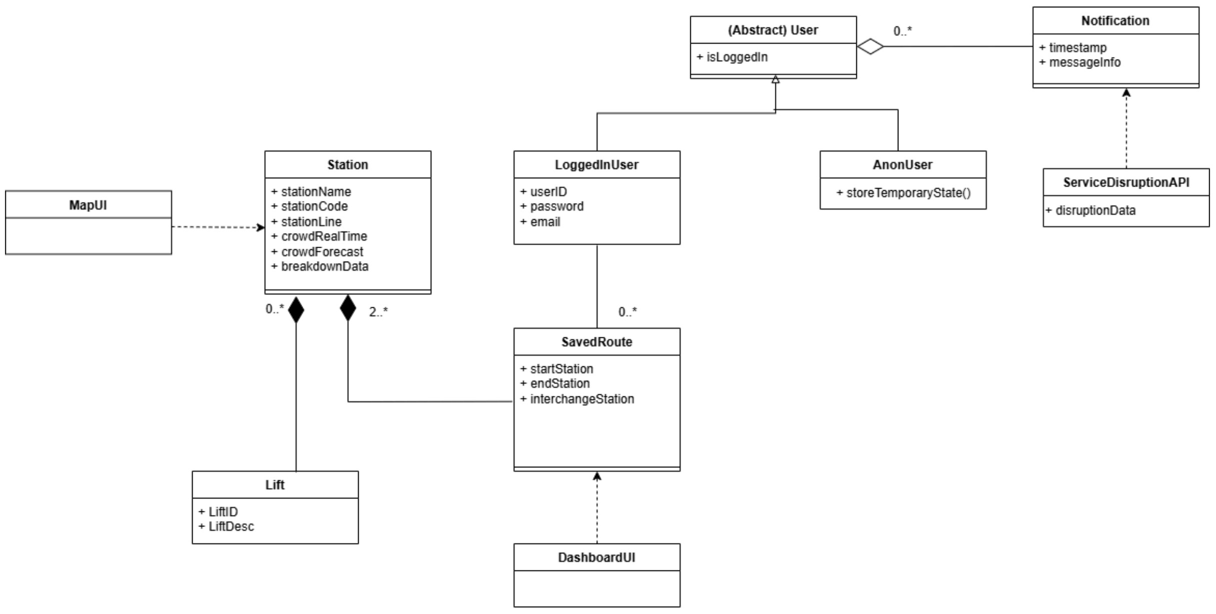


Diagram Access Link:
https://drive.google.com/file/d/1GzQe1HUrKSpWm03o91pbmqIGrYQAtgVM/view?usp=drive_link

6) Key Boundary Classes and Control Classes

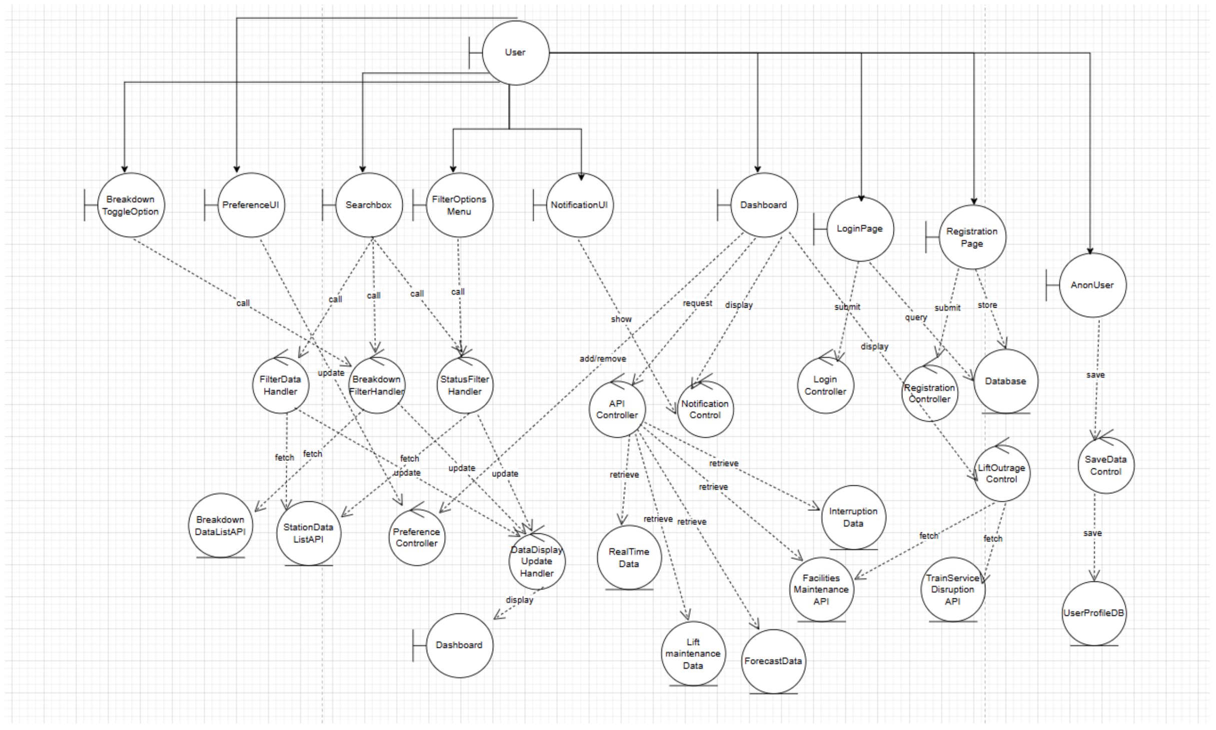


Diagram Access Link:
https://drive.google.com/file/d/1CfCV3tM8DxWZxD2oneYVx_Qt8iVvGrW9/view?usp=sharing

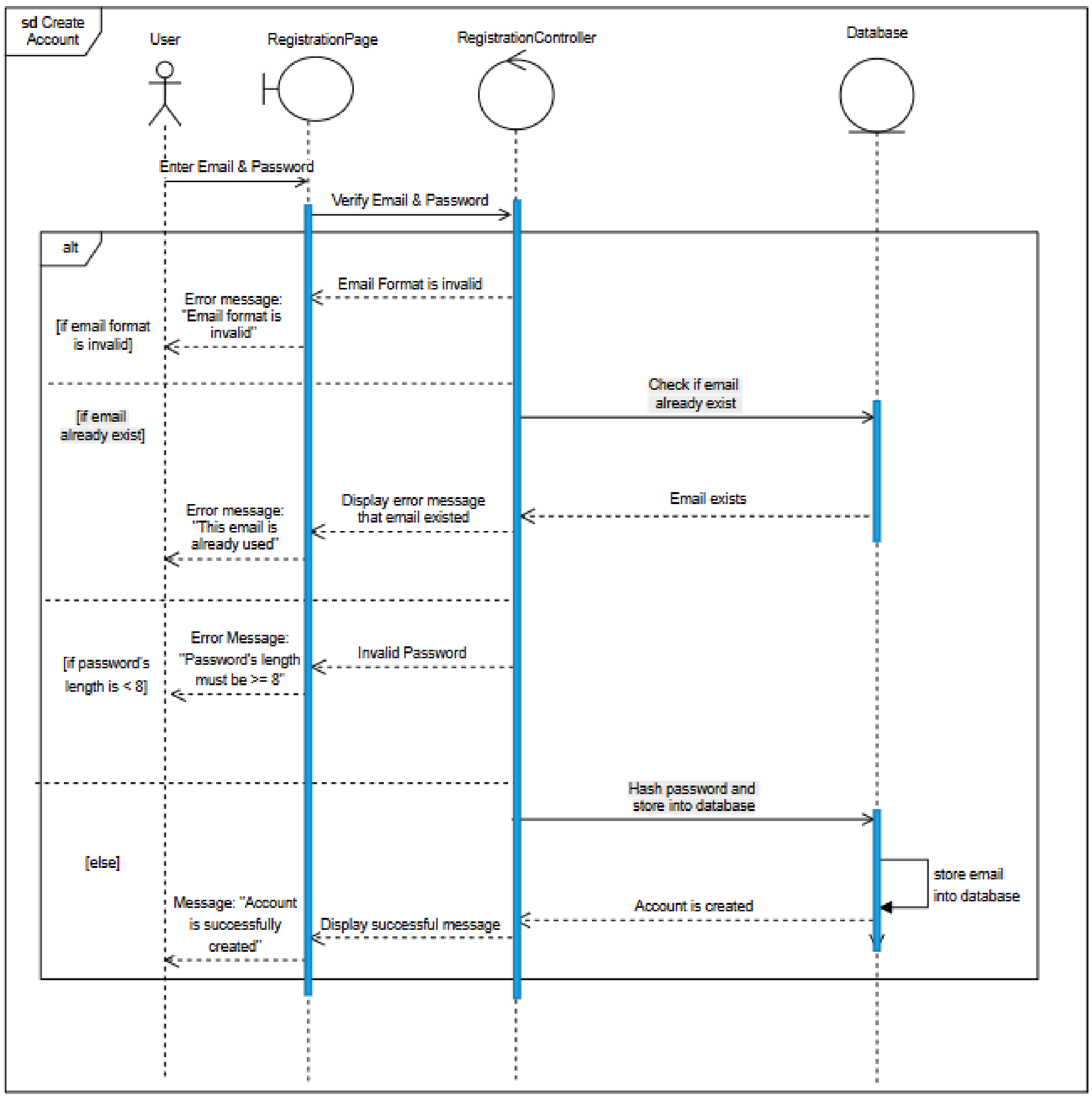
7) Sequence Diagrams

All diagrams access link:

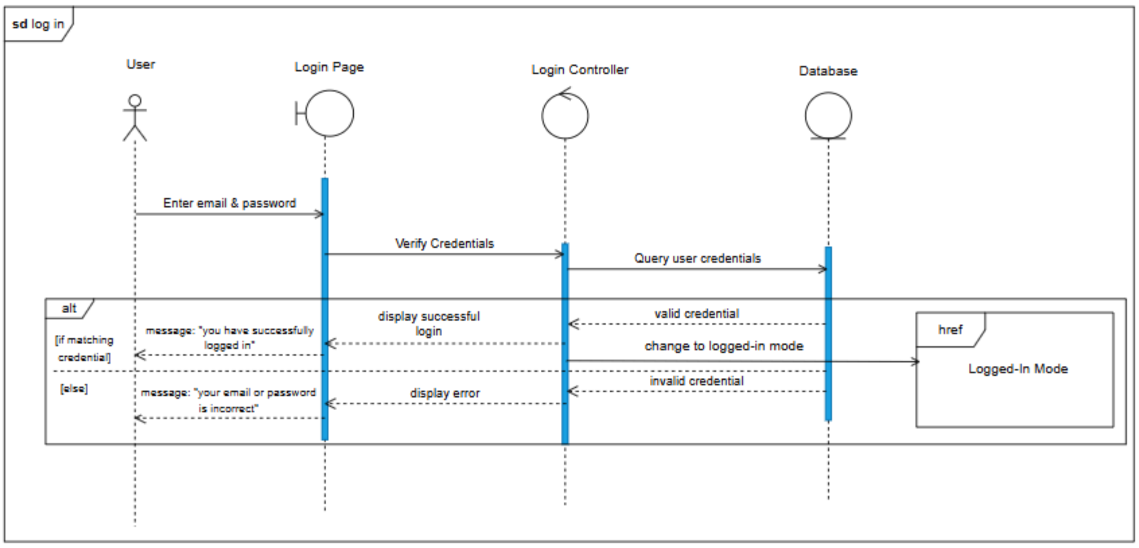
https://app.diagrams.net/#G1rq-3fzCsSFxKXBo552-YlxmEEF7C0DL6#%7B%22pageId%22%3A%22dyexEaZFBwcejvGt6OZu%22%7D

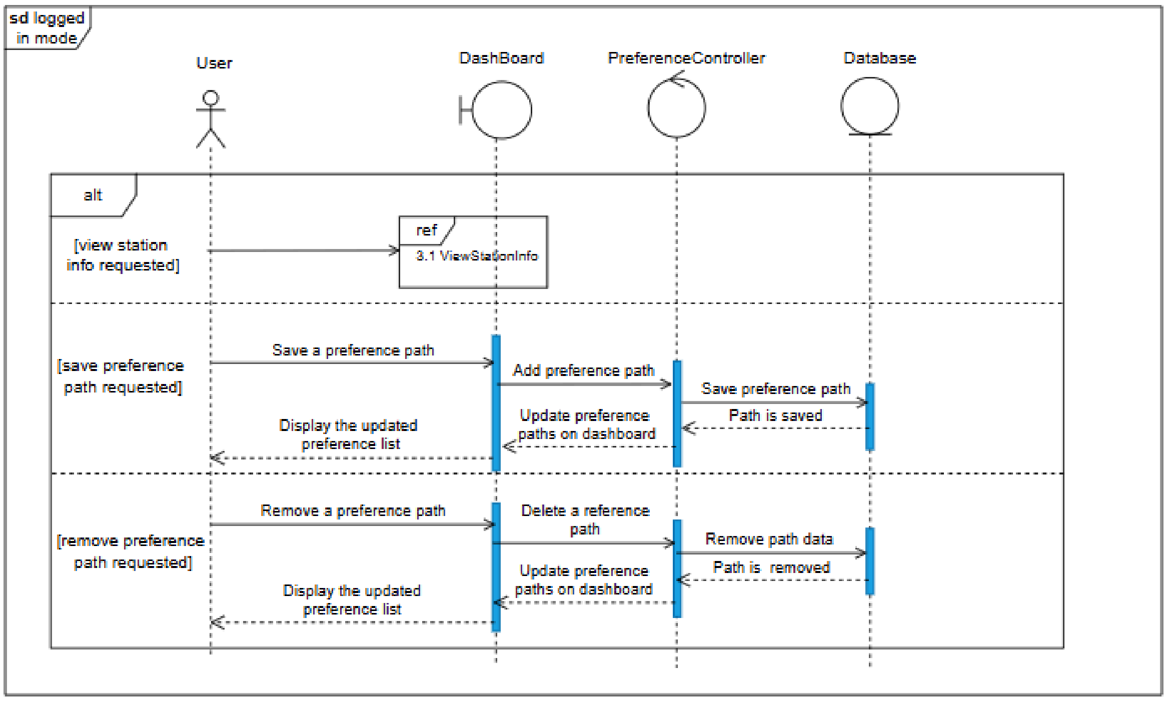
Functional Requirement #1:

1.2) User Create Account


sd Create Account
User
RegistrationPage
RegistrationController
Database
Enter Email & Password
Verify Email & Password
alt
Email Format is invalid
Error message: "Email format is invalid"
[if email format is invalid]
Check if email already exist
[if email already exist]
Email exists
Display error message that email existed
Error message: "This email is already used"
Error Message: "Password's length must be >= 8"
Invalid Password
[if password's length is < 8]
Hash password and store into database
[else]
store email into database
Account is created
Display successful message
Message: "Account is successfully created"

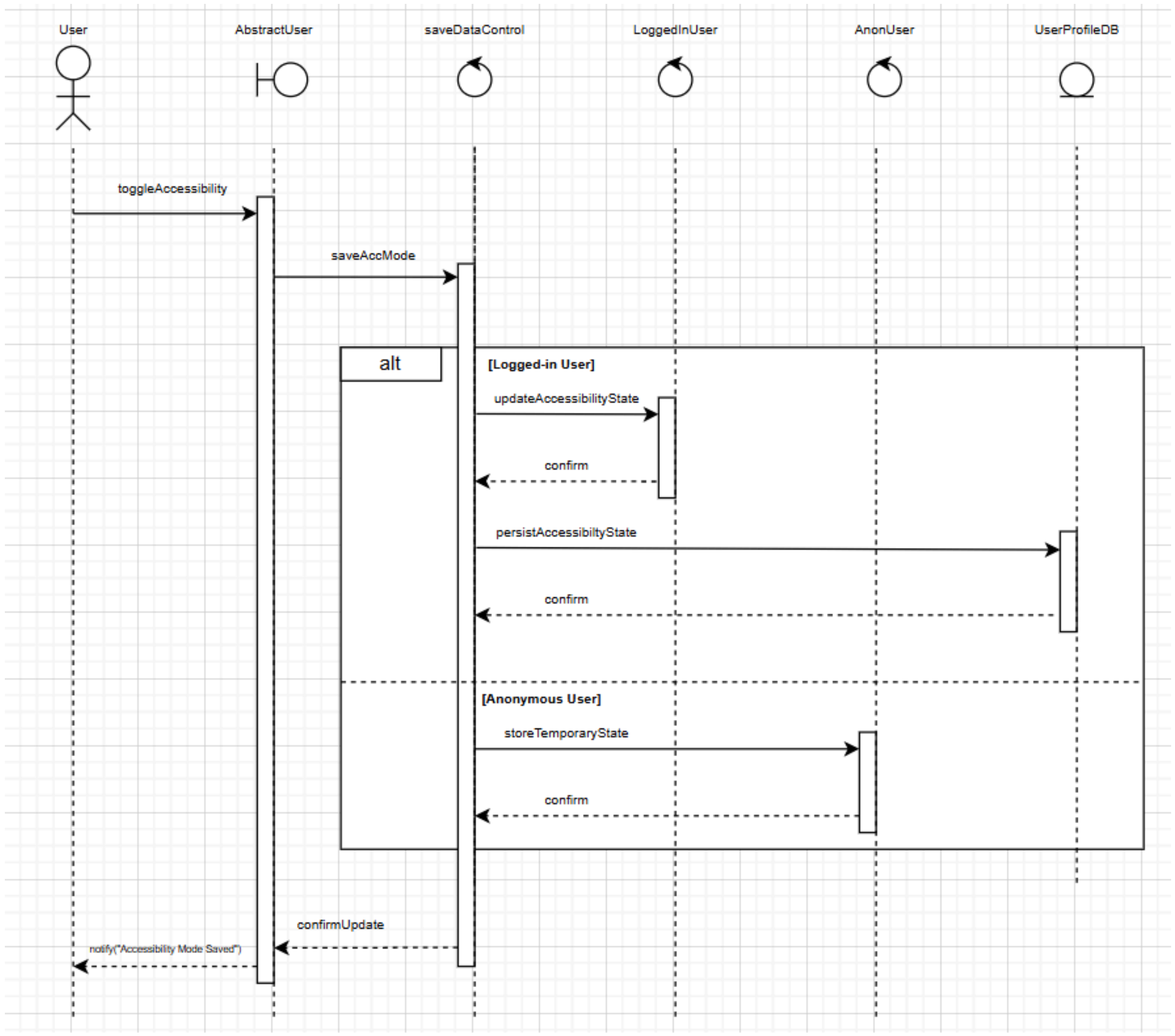
1.3) Logged in User





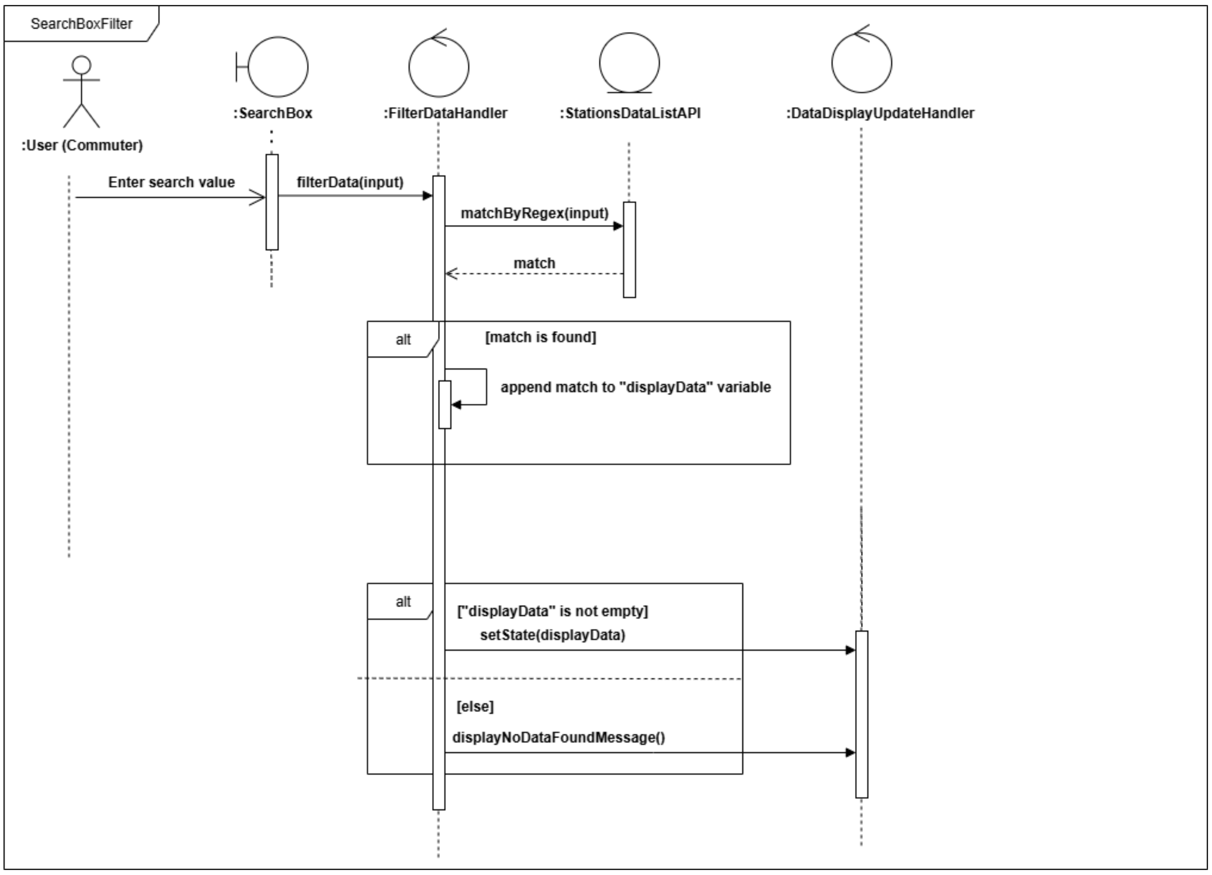
sd logged in mode
User
DashBoard
PreferenceController
Database
alt
[view station info requested]
ref
3.1 ViewStationInfo
[save preference path requested]
Save a preference path
Add preference path
Save preference path
Path is saved
Update preference paths on dashboard
Display the updated preference list
[remove preference path requested]
Remove a preference path
Delete a reference path
Remove path data
Path is removed
Update preference paths on dashboard
Display the updated preference list

1.4) Remember Accessibility Mode toggled on or off

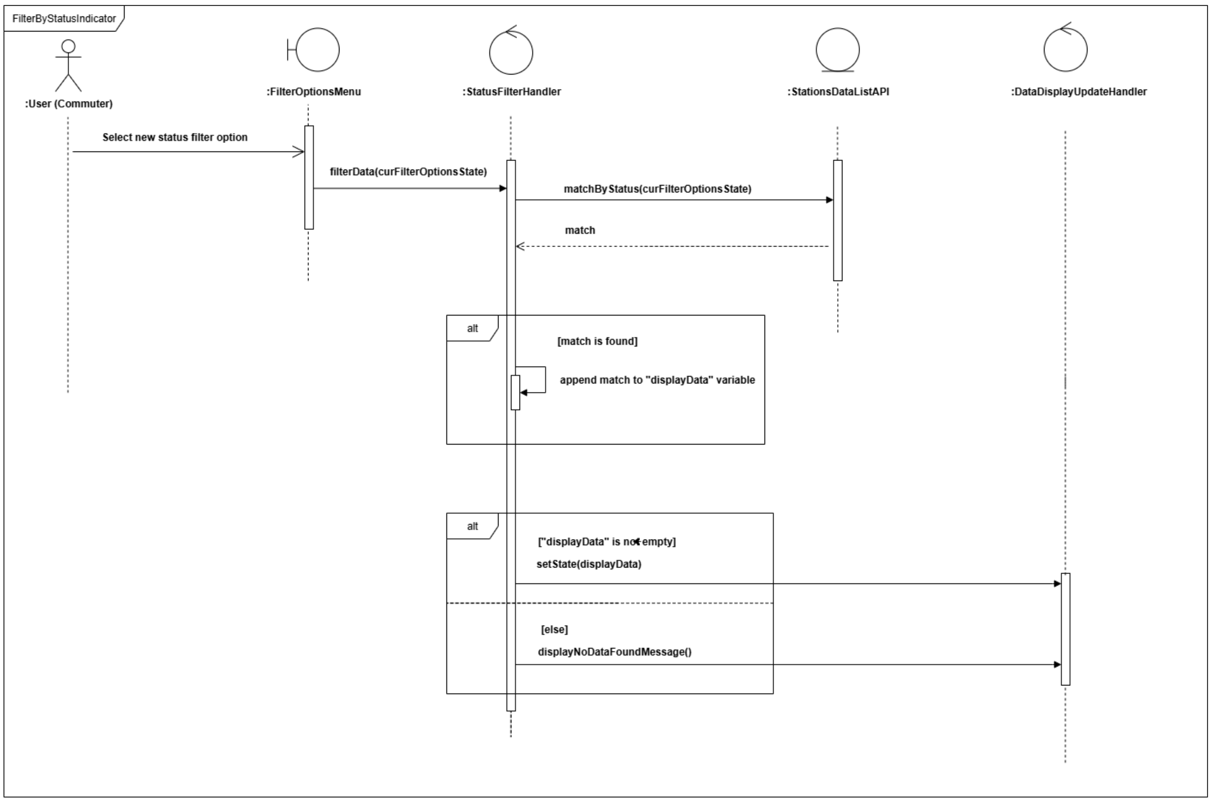


<u>Functional Requirement #2:</u>

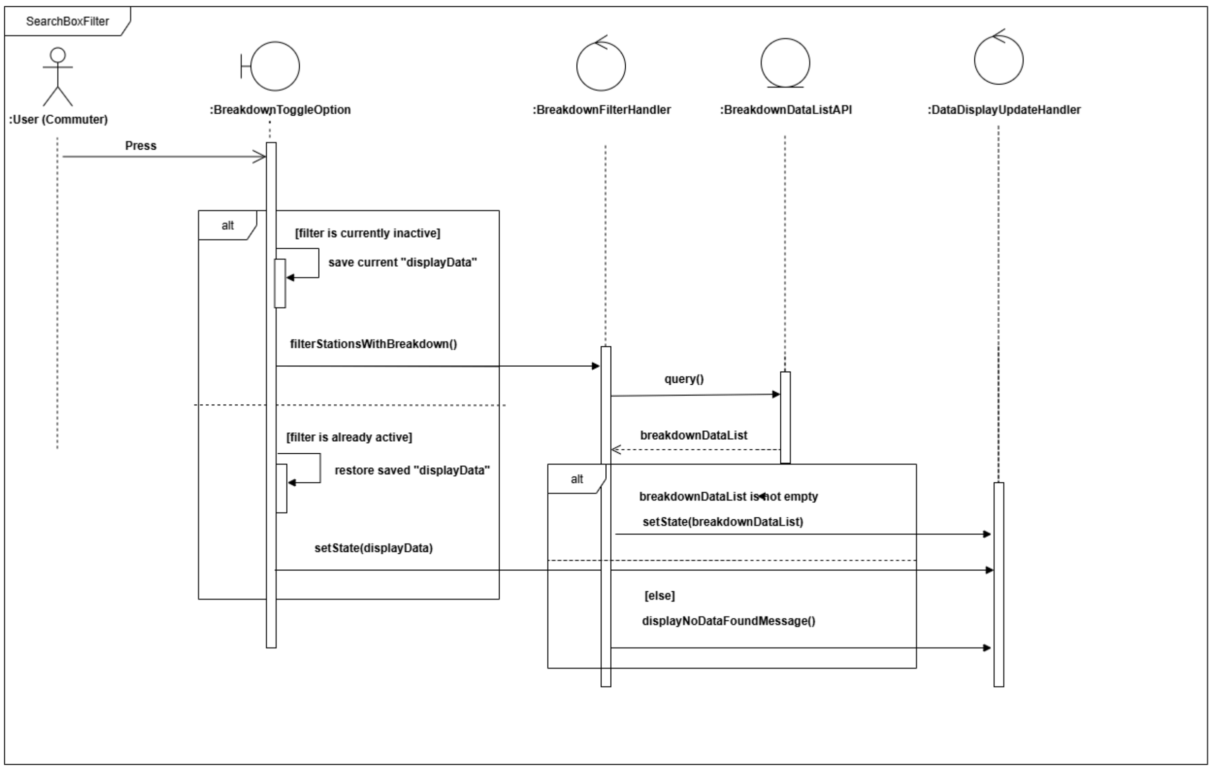
2.1) Search box filter option



2.2) Real-time and forecast crowd-level filter by status indicator

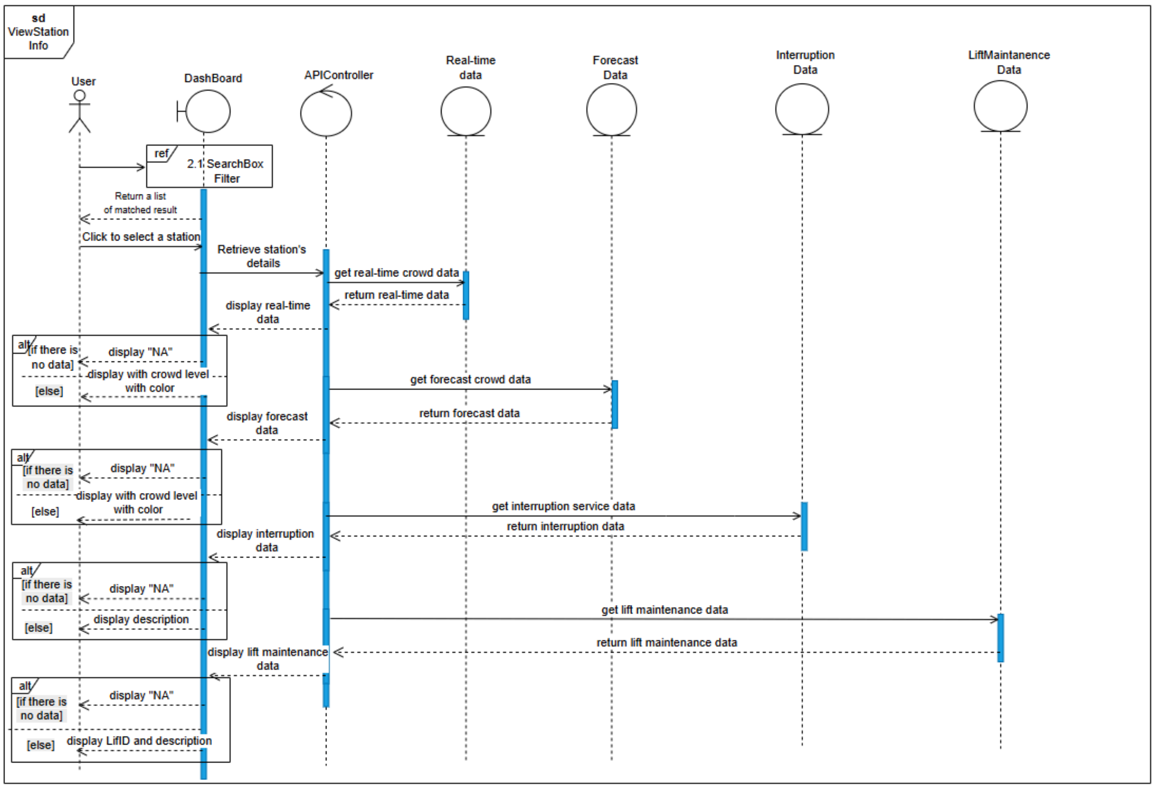

FilterByStatusIndicator
:User (Commuter)
:FilterOptionsMenu
:StatusFilterHandler
:StationsDataListAPI
:DataDisplayUpdateHandler
Select new status filter option
filterData(curFilterOptionsState)
matchByStatus(curFilterOptionsState)
match
alt
[match is found]
append match to "displayData" variable
alt
["displayData" is not empty]
setState(displayData)
[else]
displayNoDataFoundMessage()

2.3) Breakdown alert filter

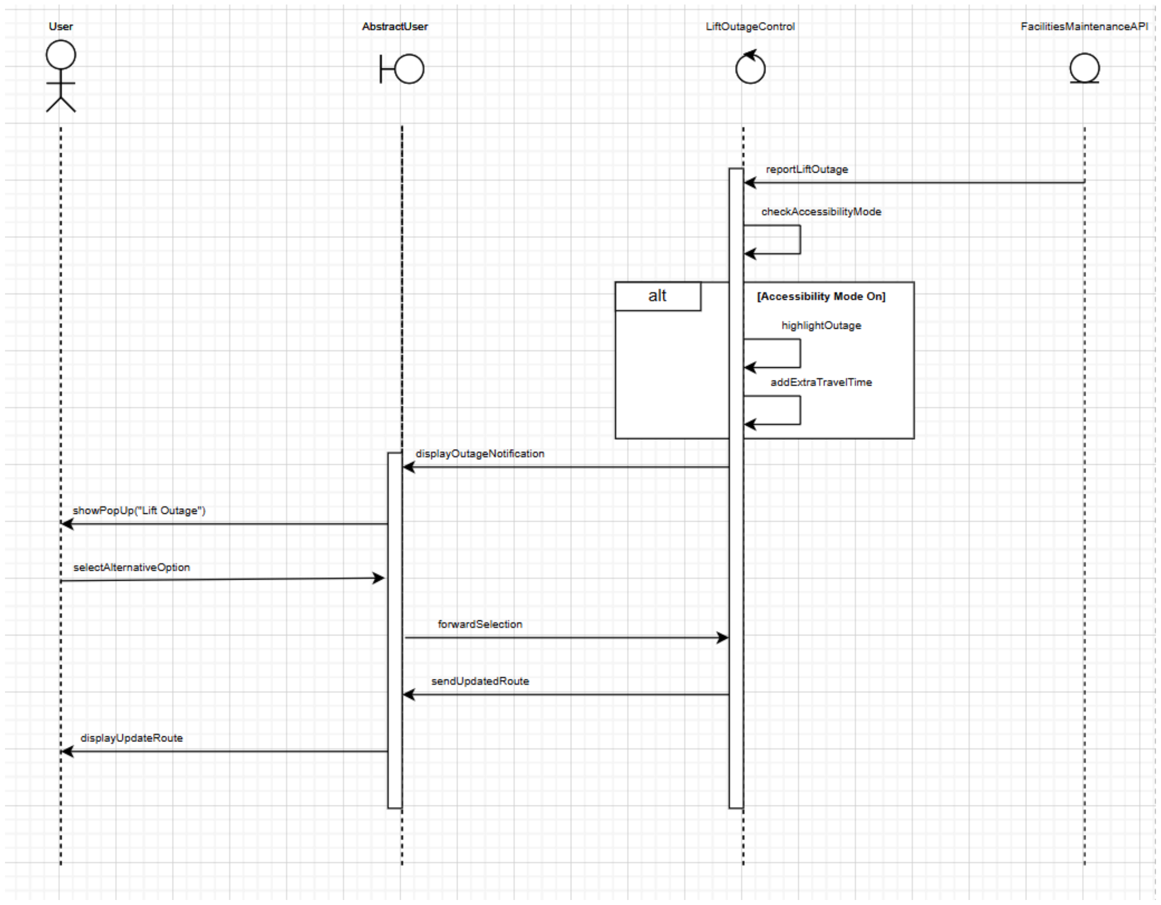

SearchBoxFilter
:User (Commuter)
:BreakdownToggleOption
:BreakdownFilterHandler
:BreakdownDataListAPI
:DataDisplayUpdateHandler
Press
alt
[filter is currently inactive]
save current "displayData"
filterStationsWithBreakdown()
query()
breakdownDataList
[filter is already active]
restore saved "displayData"
alt
breakdownDataList is not empty
setState(breakdownDataList)
setState(displayData)
[else]
displayNoDataFoundMessage()

Functional Requirement #3:

3.1) View Station General Information

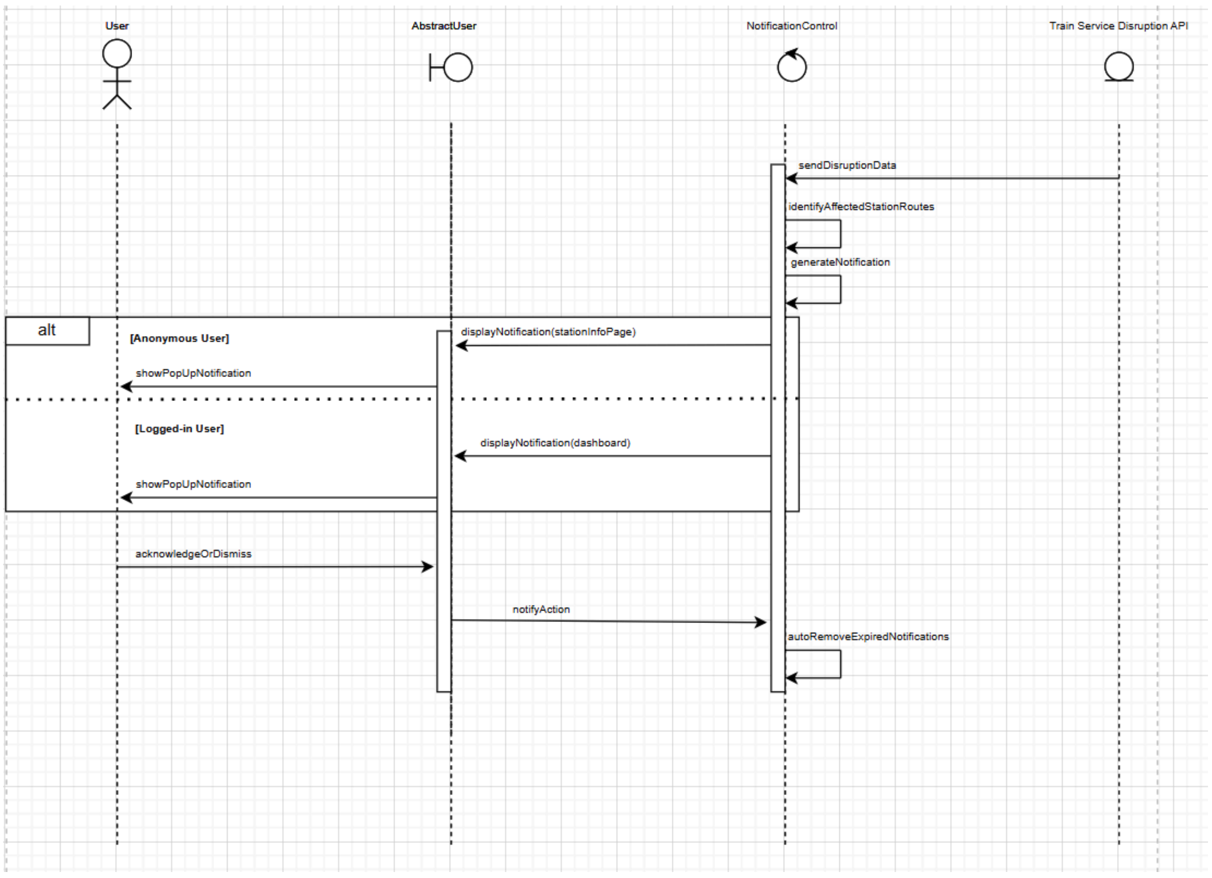


3.2) Handling Lift Outage

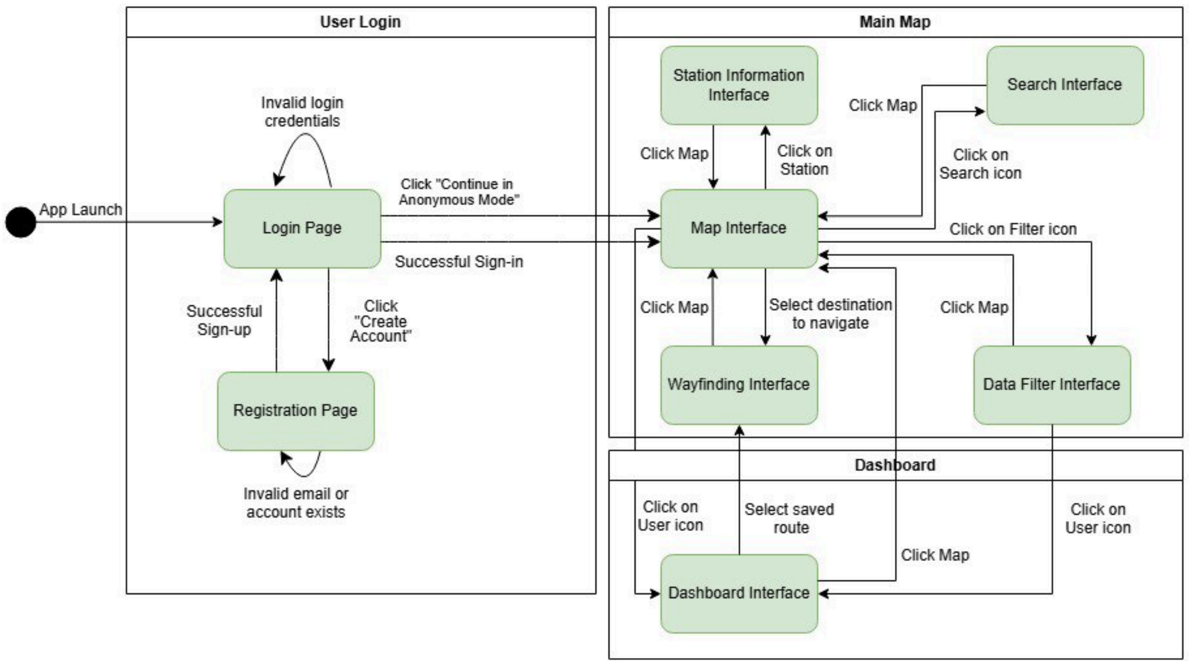


Functional Requirement #4:

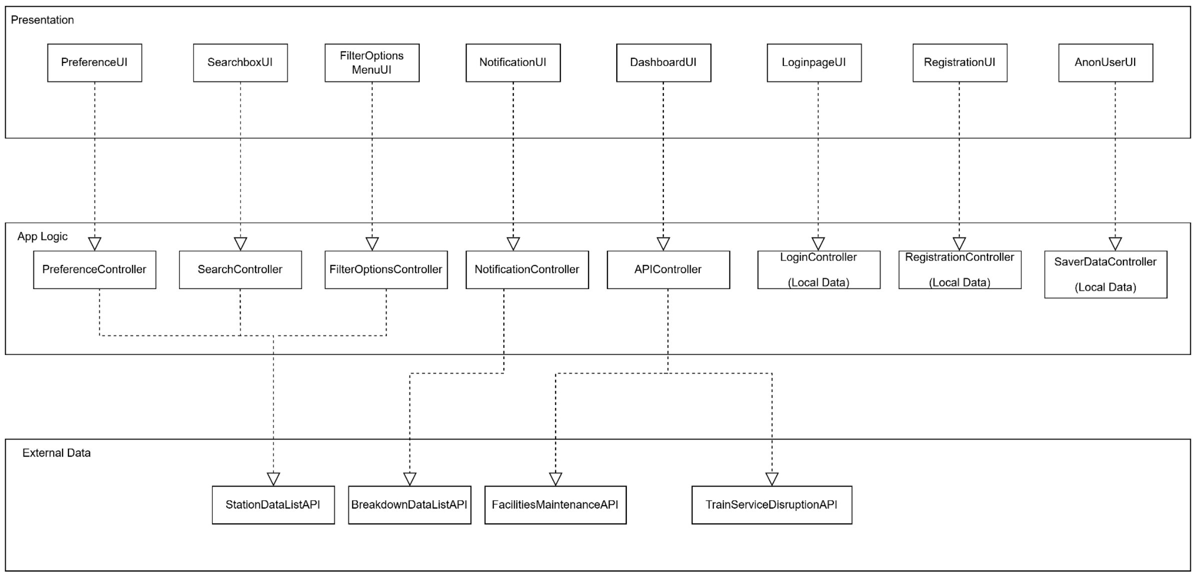
4.1) Pop-up notifications for Service Disruptions



8) Dialog Map



9) System Architecture



Presentation Layer

This layer is mainly responsible for the interaction between Users and SmartCommute. The
different UIs will then call for the respective controllers to run the App Logic.
This layer consists of:

1. **PreferenceUI**

Allows Users to save preferred routes or toggle accessibility mode by using the service of PreferenceController

2. **SearchboxUI**
   Allows Users to search for stations with SearchController
3. **FilterOptionsMenuUI**
   Allows Users to filter station search results by preference with FilterOptionsController
4. **NotificationUI**
   Shows notifications sent by NotificationController
5. **DashboardUI**
   Allows users to access saved paths quickly with the services of APIController
6. **LoginpageUI**
   Allows Users to login by using the service of LoginController
7. **RegistrationUI**
   Allows Users to register by using the service of RegistrationController
8. **AnonUserUI**
   Allows Users to access features without having to log in, with the services of SaveDataController

App Logic Layer

This layer contains all the controller classes that will provide the presentation layer with its services. The controller classes will request for entities from the Object Layer if necessary to run
its logic.
This layer consists of:

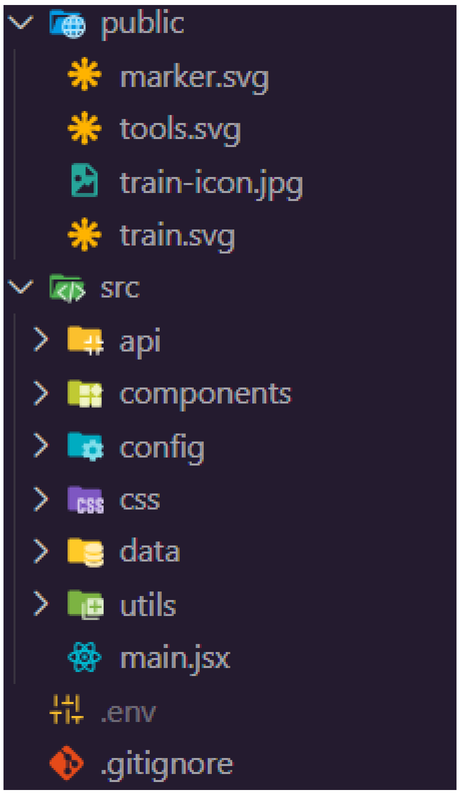
1. **PreferenceController**
   Called by PreferenceUI to allow Users to view, edit, delete and save preferred paths
2. **SearchController**
   Called by SearchUI for Users to search for stations
3. **FilterOptionsController**
   Called by FilterOptionsMenuUI to return stations with the filter options applied
4. **NotificationController**
   Called by NotificationUI to send updates and alerts to the User
5. **APIController**
   Called by DashboardUI to display information about Stations and allow for quick access to Preferences

6. **LoginController**
   Called by LoginUI to allow User to login
7. **RegistrationController**
   Called by RegistrationUI to allow User to register
8. **SaveDataController**
   Contains logic for saved preferences

Persistent Data Layer
This layer contains the database that will store all of the entities.

10) Application Skeleton

A) *Front-end*



```
public
    marker.svg
    tools.svg
    train-icon.jpg
    train.svg
src
  > api
  > components
  > config
  > css
  > data
  > utils
    main.jsx
.env
.gitignore
```

public/

Static assets accessible directly by the browser. Contains images like compass.png (for navigation/direction features), location.png (for station markers), and train-icon.jpg (for UI elements).

src/components

Reusable React components like station cards, filter menus, search boxes, notification alerts, login/registration forms, and accessibility toggle buttons.

src/css

Stylesheets for styling your application, including layouts, color indicators (green/yellow/red for crowd levels), and responsive design.

src/api

Contains API integration functions for backend communication. Handles requests like authentication (login/register), station data, routing, alerts, and facilities using Axios, serving as the bridge between the frontend and backend.

src/config

Stores configuration files in JSON format, including API endpoints and filter options. This centralizes settings that may change across different environments or need to be easily modified without touching the core code.

src/data

Contains static JSON data files like crowd level indicators, station codes, and station-to-color mappings. This data is used for reference, translations, and mapping values throughout the application.

src/utils

Provides utility functions for common operations like localStorage management. These reusable helper functions simplify recurring tasks across the application and improve code maintainability.

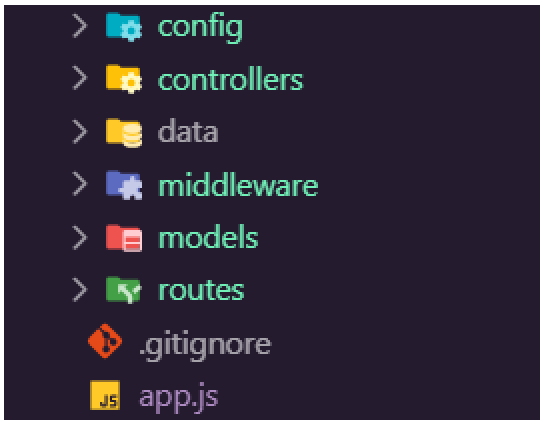
App.jsx

Root component that sets up routing, manages global state (user login, accessibility mode), and renders main layout structure.

main.jsx

Entry point that renders the App component into the DOM and sets up React with any global providers or configurations.

B) Back-end



```
> config
> controllers
> data
> middleware
> models
> routes
  .gitignore
  app.js
```

config/

Configuration settings like database connections, environment variables, API keys, and authentication settings.

controllers/

Business logic that handles requests, processes data, and sends responses. Implements features like registration, station data retrieval, filtering, and notifications.

data/

Static or reference data such as MRT station lists, line information, and fixed datasets.

middleware/

Functions that execute during requests for authentication checks, input validation, error handling, and applying user preferences.

models/

Database schemas defining how data (users, stations, preferences, alerts) is structured and stored.

routes/

API endpoint definitions that map URLs to their corresponding controller functions.

11) Black Box Testing

1) Control class to be tested: ***authController***

-The authController class is responsible for managing user authentication and registration in SmartCommute, implementing the core user account functionality that distinguishes between anonymous and authenticated users.

-Its two primary functions:

+)User Registration (signup):

- Validates required registration data (email and password)
- Ensures email format validity using regex pattern matching
- Enforces password security with minimum length requirement (8 characters)
- Prevents duplicate accounts by checking existing email addresses
- Securely hashes passwords before storage
- Creates new user accounts with default accessibility settings

+) User Authentication (login):

- Validates user credentials (email and password)
- Retrieves and verifies hashed passwords
- Generates JWT tokens for authenticated sessions
- Returns user profile data for personalized features

-This control class is crucial because it serves as the gateway between anonymous and authenticated access, enabling personalized features, such as saving preference paths

2) Equivalence Classes and Boundary Values Testing:

1) Login Function:

Parameter	Valid Equivalence Class	Invalid Equivalence Class
Email	Proper email format, exists in database	Missing OR not in database
Password	Matches user's password in database	Missing OR wrong password

2) Register Function:

Parameter	Valid Equivalence Class	Invalid Equivalence Class
Email	Correct format, not in database	Missing OR wrong format OR already exists in database
Password	Has >= 8 characters	< 8 characters
Confirm password	Matches the password	Does not match the password

3) Test cases and results

a) Login Function

N.o	Test input	Expected Result	Actual Result	Pass
1	**(valid)** Email: “andrich123@gmail.com” **(valid)** Password: “TestingPassword123” *(assume that these inputs are in data base)*	Successful login	Successful login	Yes
2	**(invalid)** Email: “” **(valid)** Password: “TestingPassword123”	Message error: “Please fill email and password fields”	Message error: “Please fill email and password fields”	Yes
3	**(invalid)** Email: “andrich.com” **(valid)** Password: “TestingPassword123”	Message error: “Your email or password is incorrect”	Message error: “Your email or password is incorrect”	Yes

4	**(valid)** Email: “andrich123@gmail.com” **(invalid)** Password: “”	Message error: “Please fill email and password fields”	Message error: “Please fill email and password fields”	Yes
5	**(valid)** Email: “andrich123@gmail.com” **(invalid)** Password: “siuuuu12345678”	Message error: “Your email or password is incorrect”	Message error: “Your email or password is incorrect”	Yes

b) Sign Up Function

N.o	Test input	Expected Result	Actual Result	Pass
1	**(all valid inputs)** -Email: “MessiGoat@gmail.com” -Password: “Password12345” -Confirm Password: “Password12345”	Account is successfully created	Account is successfully created	Yes
2	**(all inputs are**	Message error:	Message error:	Yes

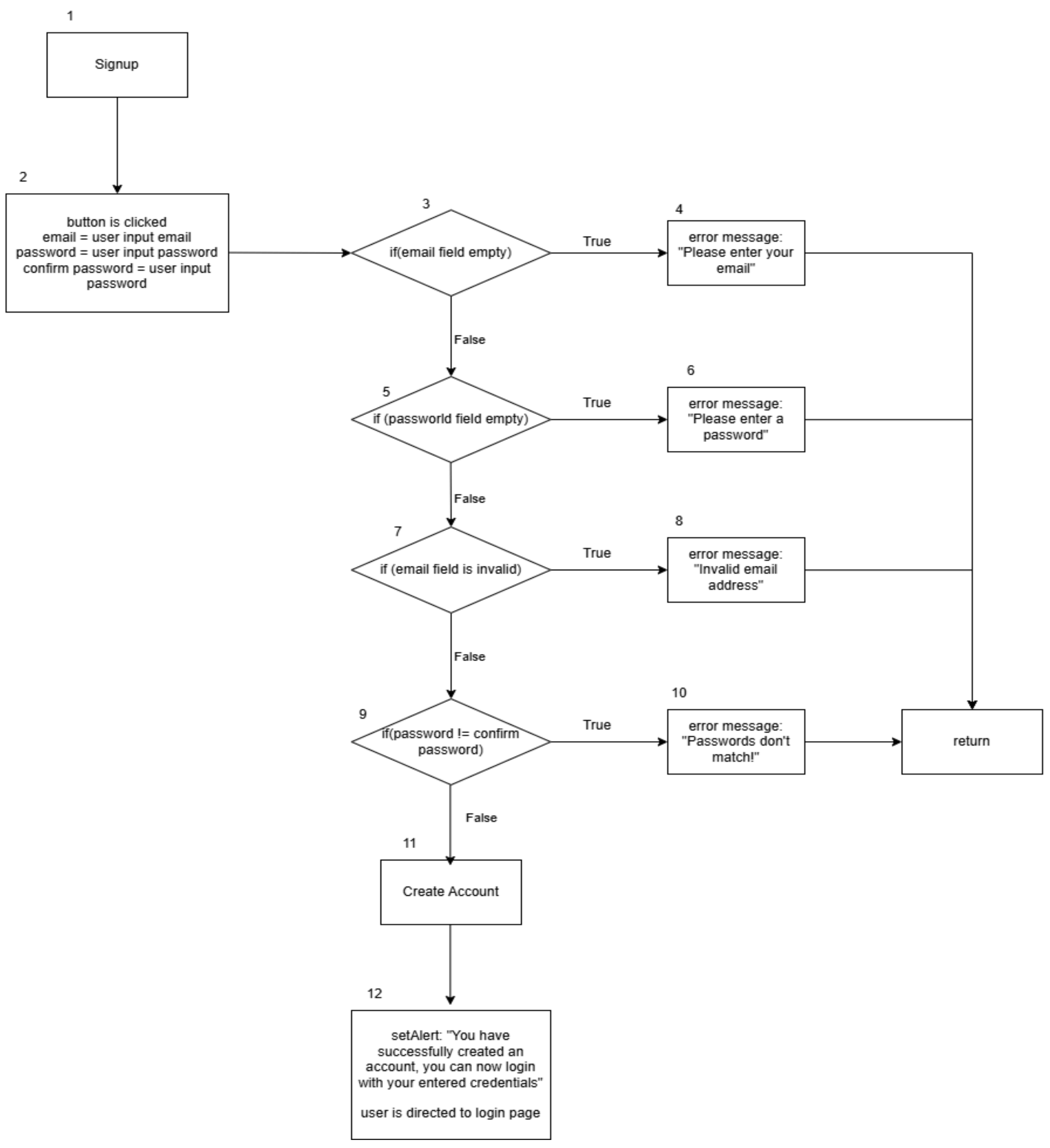
	valid except email) -Email: “”	“Please fill email and password fields”	“Please fill email and password fields”	
3	**(all inputs are valid except password)** -Password: “”	Message error: “Please fill email and password fields”	Message error: “Please fill email and password fields	Yes
4	**(all inputs are valid except email)** -Email: “MessiGoatgmail?com”	Message Error: “Email format is invalid”	Message Error: “Email format is invalid”	Yes
5	**(all inputs are valid except password)** -Password: “messi”	Message Error: “ Password must be at least 8 characters long”	Message Error: “ Password must be at least 8 characters long”	Yes
6	**(all inputs are valid except email)** -Email: “andrich123@gmail.com” (*Assume that the email “andrich123@gmail.com” already exists in the database)*	Message Error: “This email is already in use”	Message Error: “This email is already in use”	Yes
7	**(all inputs are valid except confirm password)**	Message Error: “Passwords don’t match”	Message Error: “Passwords don’t match”	Yes

	-Password: "Password12345" -Confirm Password: "Password"			

12) White Box Testing

1) Signup

Method: Creating account
Control Flow Graph:
https://drive.google.com/file/d/1xg7siPlaDBO4cs0yBTc6BFzsnbLfoOaT/view?usp=sharing


1
Signup
2
button is clicked
email = user input email
password = user input password
confirm password = user input password
3
if(email field empty)
True
4
error message: "Please enter your email"
False
5
if (passworld field empty)
True
6
error message: "Please enter a password"
False
7
if (email field is invalid)
True
8
error message: "Invalid email address"
False
9
if(password != confirm password)
True
10
error message: "Passwords don't match!"
return
False
11
Create Account
12
setAlert: "You have successfully created an account, you can now login with your entered credentials"
user is directed to login page

<u>Basis Path Testing</u>

Cyclomatic Complexity = |decision points| + 1 = 4 + 1= 5

Basis Paths

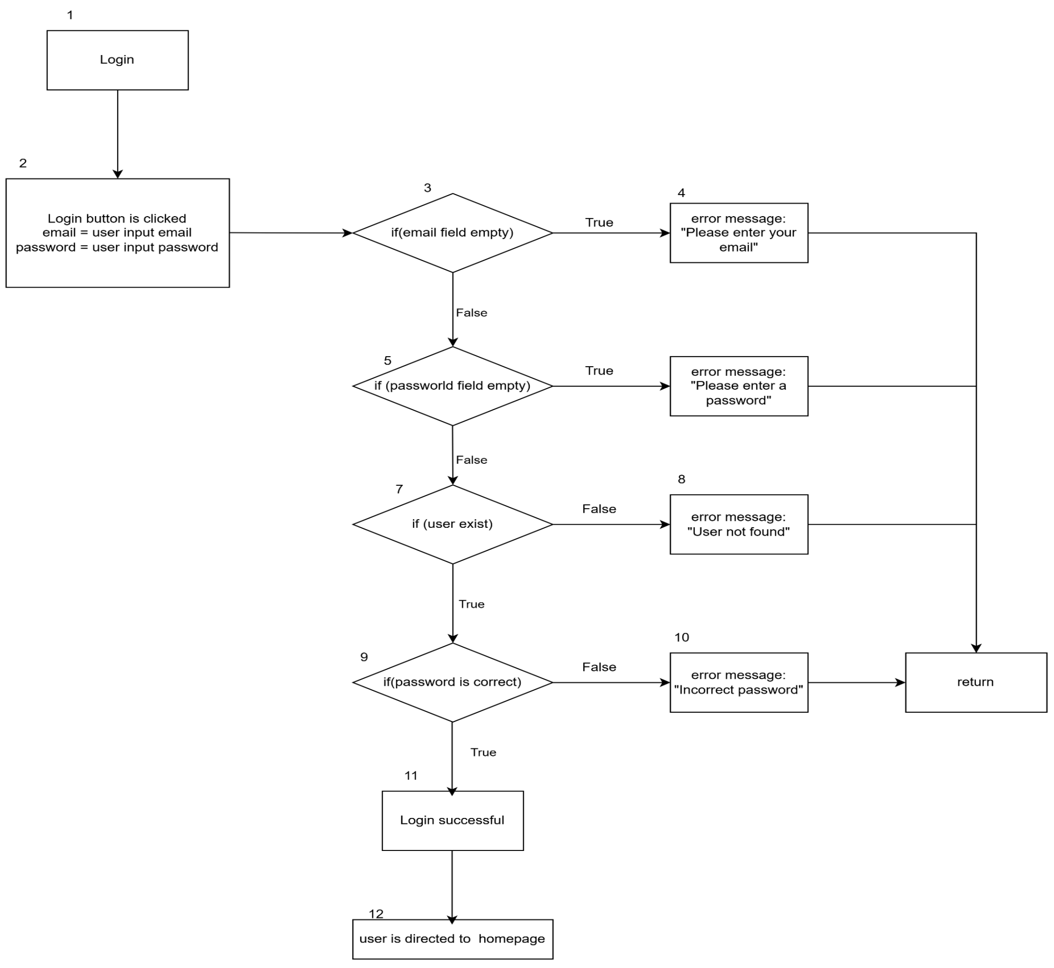
1. Baseline path: 1, 2, 3, 5, 7, 9, 11, 12
2. Basis Path: 1, 2, 3, 4, 11
3. Basis Path: 1, 2, 3, 5, 6, 11
4. Basis Path: 1, 2, 3, 5, 7, 8, 11
5. Basis Path: 1, 2, 3, 5, 7, 9, 10, 11

Test Cases and Results

No.	Test Input	Expected Output	Actual Output	Pass?
1	email = “someone@gmail.com” password = “123456” confirm password = “123456’	"You have successfully created an account, you can now login with your entered credentials"	"You have successfully created an account, you can now login with your entered credentials"	Yes
2	email = “” password = “123456” confirm password = “123456”	“Please enter your email”	“Please enter your email”	Yes
3	email = “someone@gmail.com” password = “” confirm password = “”	“Please enter a password”	“Please enter a password”	Yes
4	email = “someone” password = “123456” confirm password = “123456”	“Invalid email address”	“Invalid email address”	Yes
5	email = “someone@gmail.com” password = “p455w0rd” confirm password = “passw0rd”	“Passwords don’t match”	“Passwords don’t match”	Yes

2) Login

Method: Login to account
Control Flow:
Graphhttps://drive.google.com/file/d/1xMv9MTDilmvOCc0yelKD2S0qN3jWuQzJ/view?usp=sharing


1
Login
2
Login button is clicked
email = user input email
password = user input password
3
if(email field empty)
True
4
error message: "Please enter your email"
False
5
if (passworld field empty)
True
error message: "Please enter a password"
False
7
if (user exist)
False
8
error message: "User not found"
True
9
if(password is correct)
False
10
error message: "Incorrect password"
return
True
11
Login successful
12
user is directed to homepage

Basis Path Testing

Cyclomatic Complexity = |decision points| + 1 = 4 + 1= 5

Basis Paths

6. Baseline path: 1, 2, 3, 5, 7, 9, 11, 12
7. Basis Path: 1, 2, 3, 4, 11
8. Basis Path: 1, 2, 3, 5, 6, 11
9. Basis Path: 1, 2, 3, 5, 7, 8, 11
10. Basis Path: 1, 2, 3, 5, 7, 9, 10, 11

Test Cases and Results

No.	Test Input	Expected Output	Actual Output	Pass?
1	email = "someone@gmail.com", password = "password123"’	"You have successfully logged in."	"You have successfully logged in."	Yes
2	email = "", password = "password123"	“Please enter your email”	“Please enter your email”	Yes
3	email = "someone@gmail.com", password = ""	“Please enter a password”	“Please enter a password”	Yes
4	email = "nonexistentuser@gmail.com", password = "password123"	"Error: User not found."	"Error: User not found."	Yes
5	email = "someone@gmail.com", password = "wrongpassword"	"Error: Incorrect password."	"Error: Incorrect password."	Yes