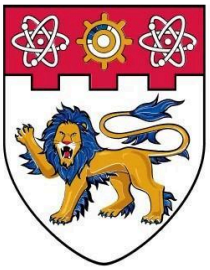


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

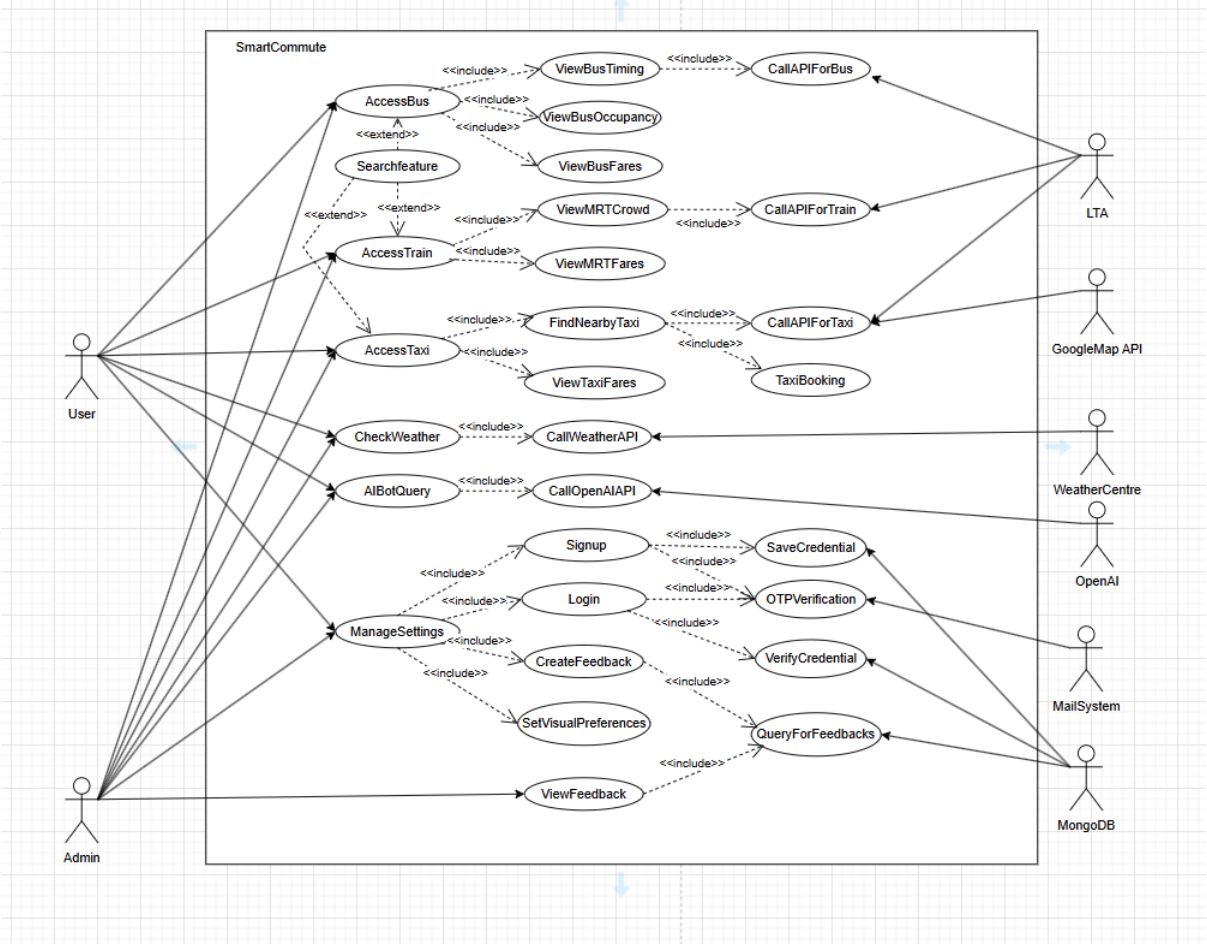
SC2006 – Software Engineering

Lab 3 Deliverables

Lab Group	SCEX
Team	SmartCommute
Members	AMANDA RAE JOSEPHINE (U2420764F)
	AW YONG WING KIAN, ALVIN (U2223300F)
	CHAN ZI HAO (U2222242B)
	IVAN CHENG LI HAO (U2221078L)
	JACE SEOW WEN HUI (U2222469F)

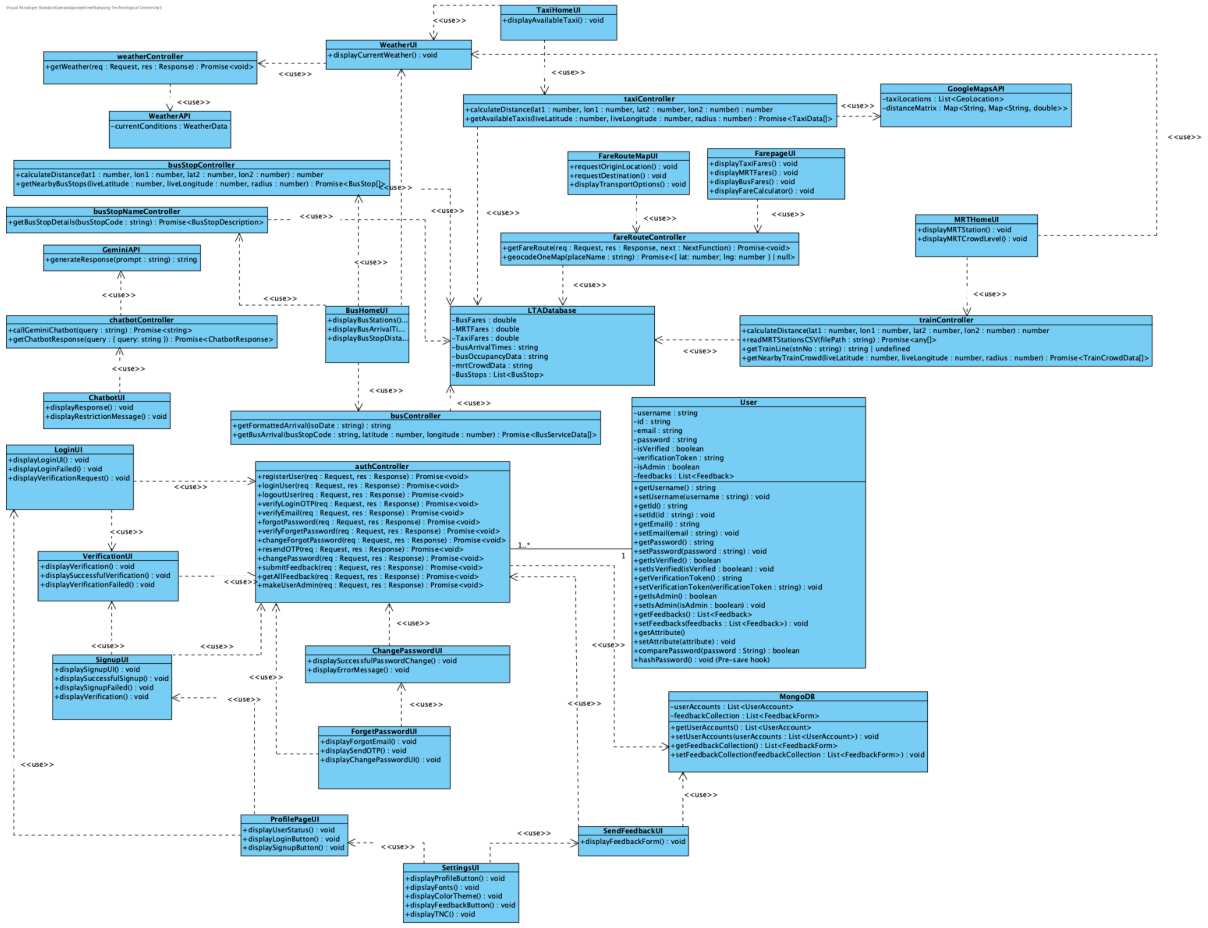
Table of Contents

Complete Use Case model

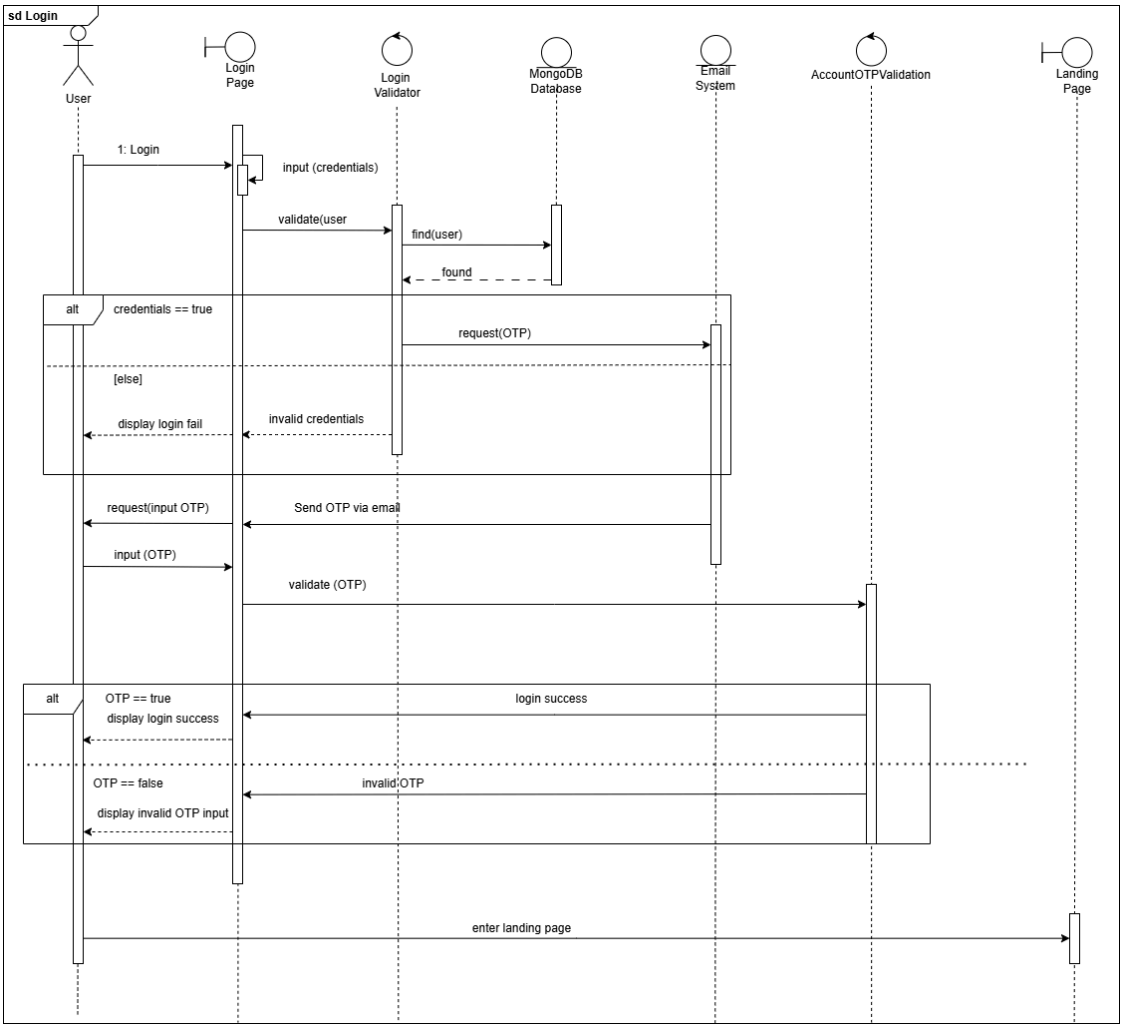


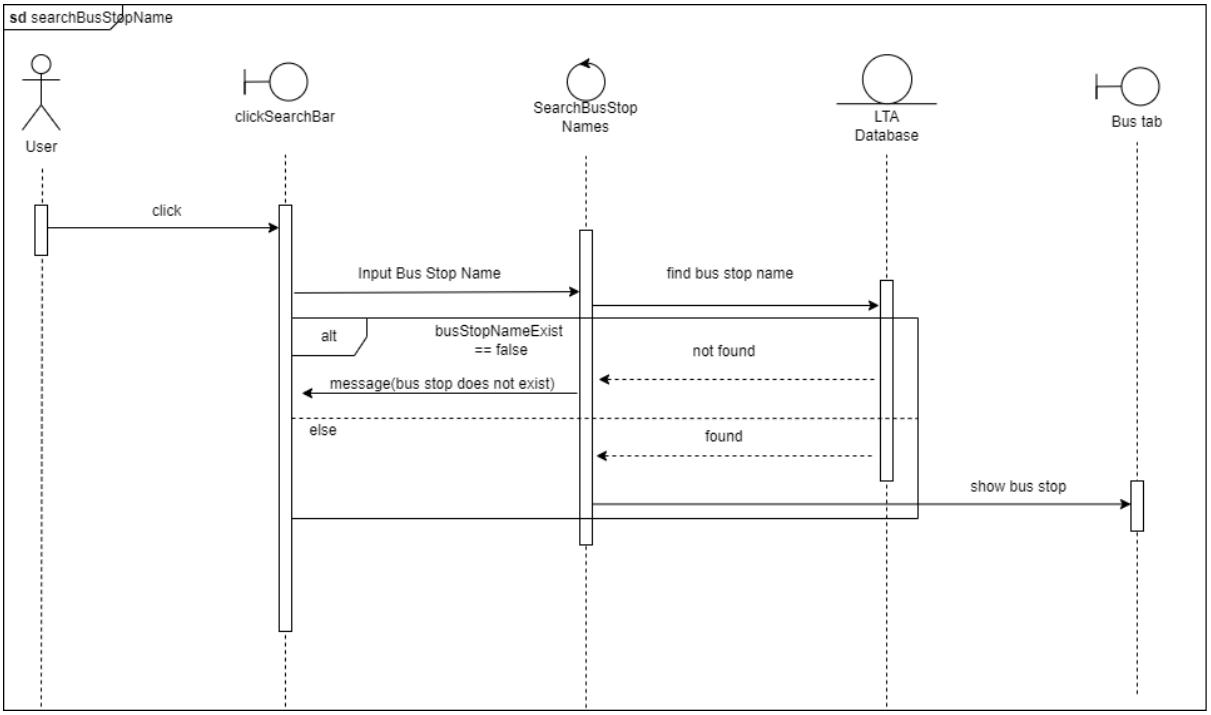
Design Model

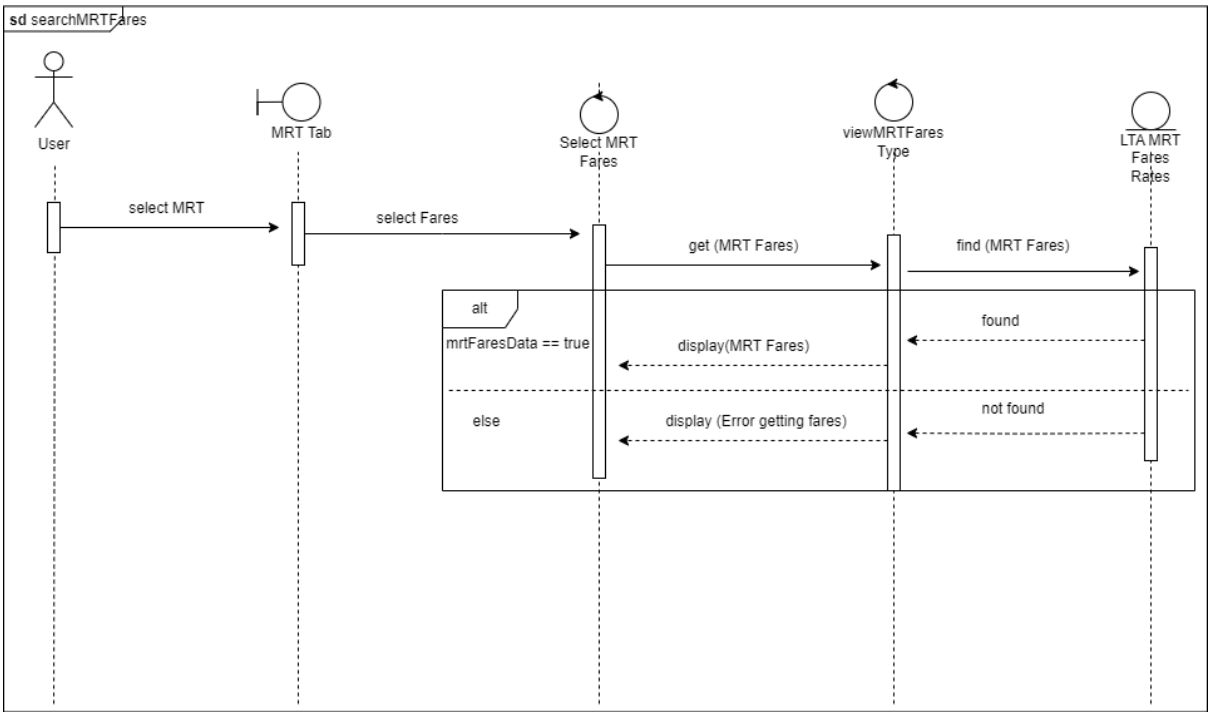
Class diagram

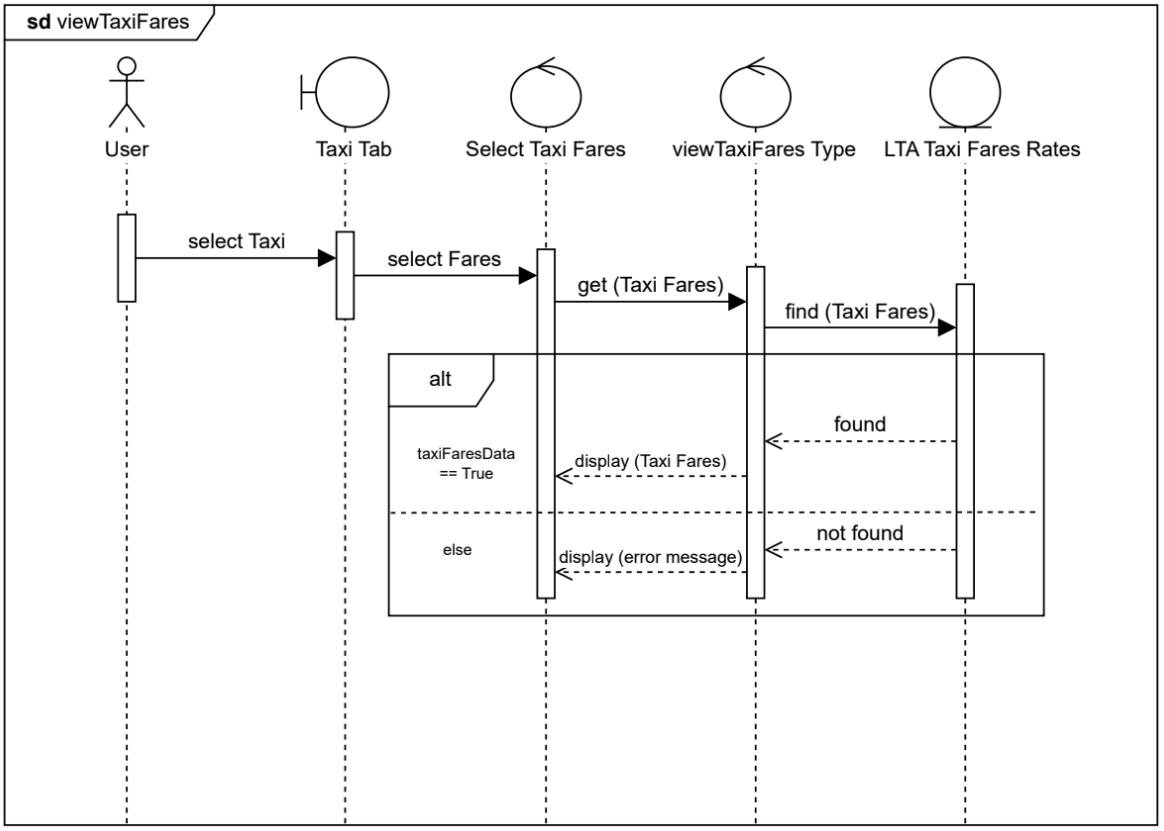


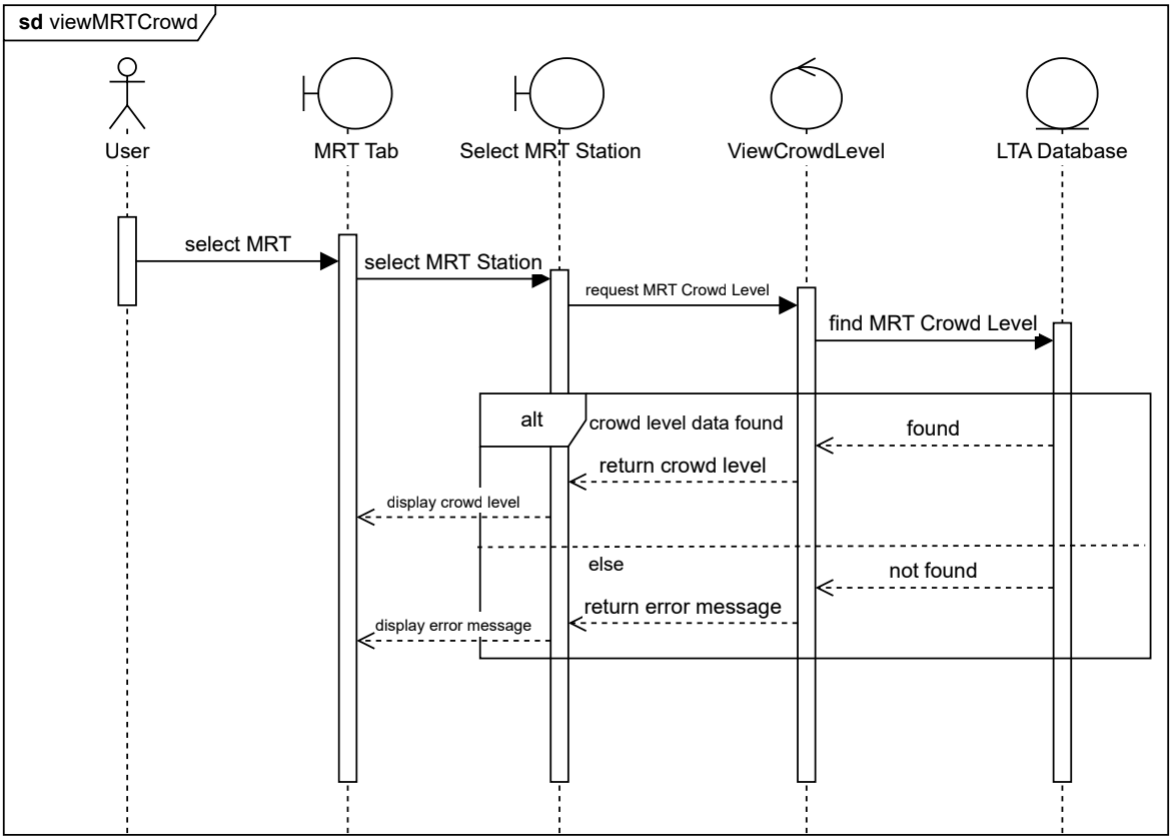
Sequence diagrams

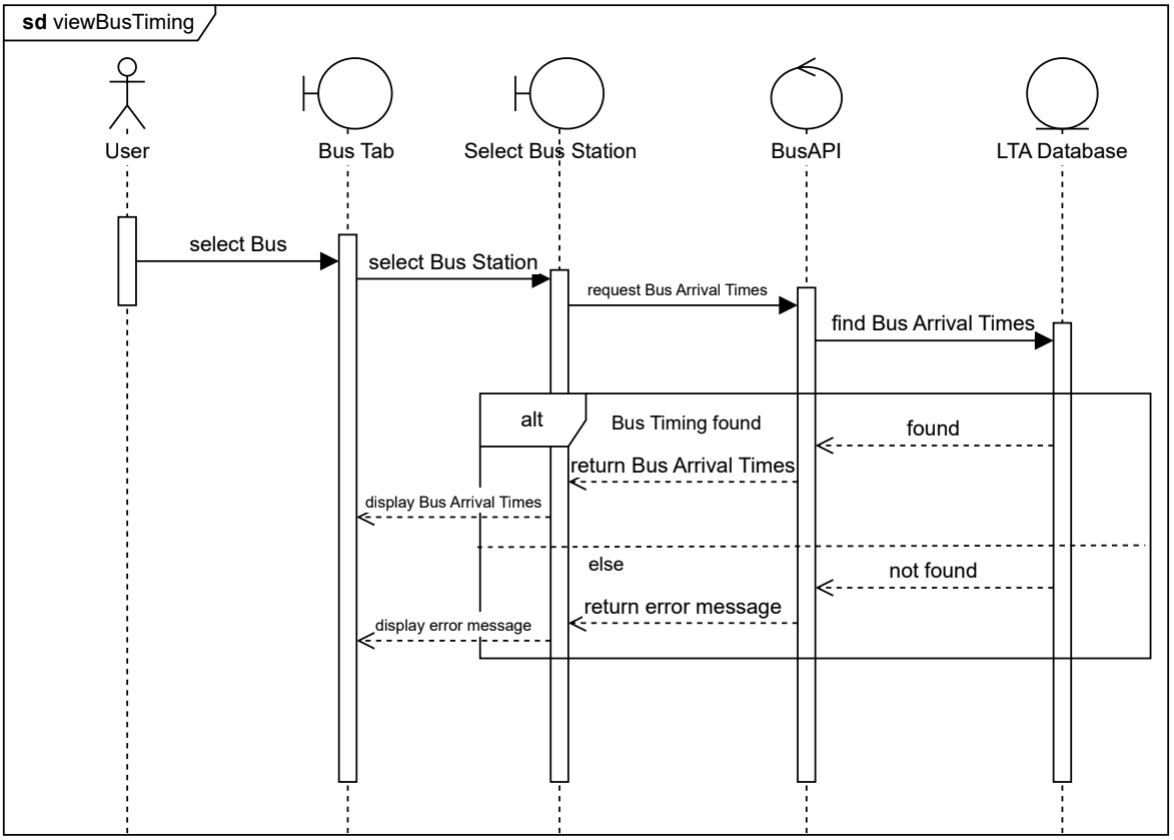


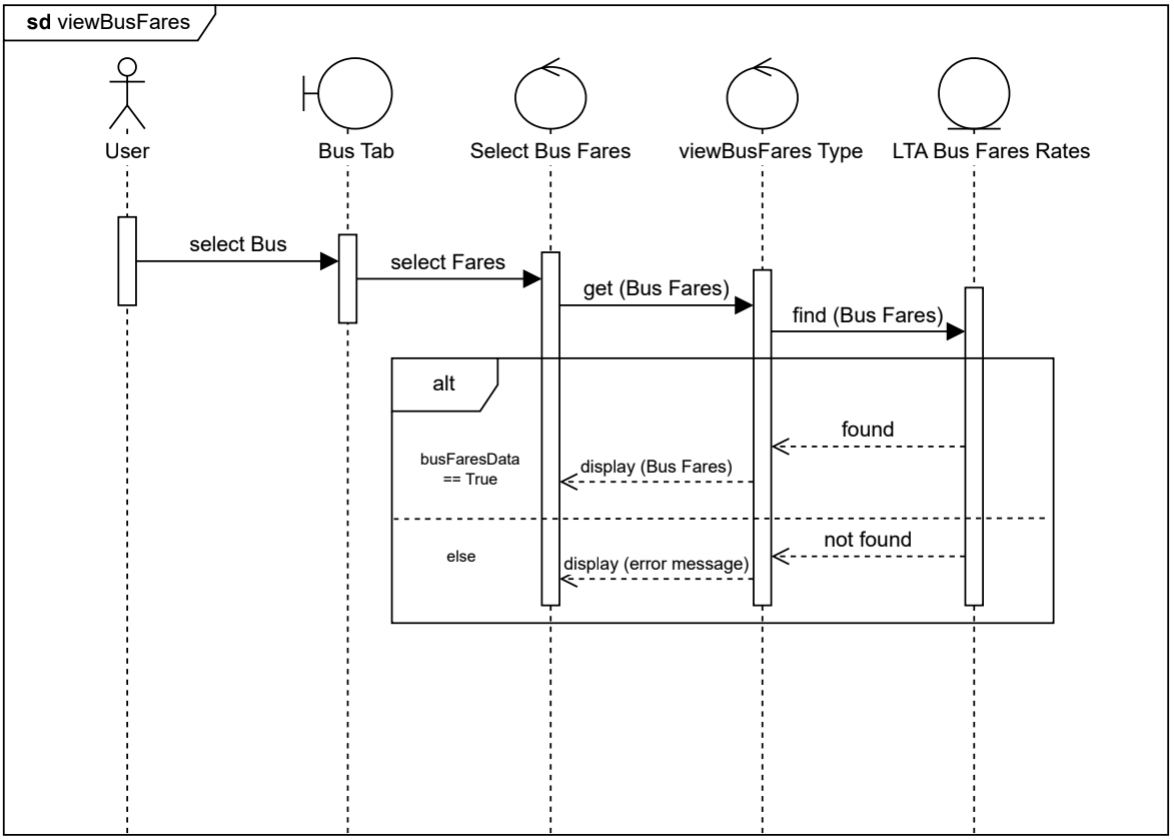

sd searchBusStopName
User
clickSearchBar
SearchBusStop Names
LTA Database
Bus tab
click
Input Bus Stop Name
find bus stop name
alt
busStopNameExist == false
not found
message(bus stop does not exist)
else
found
show bus stop

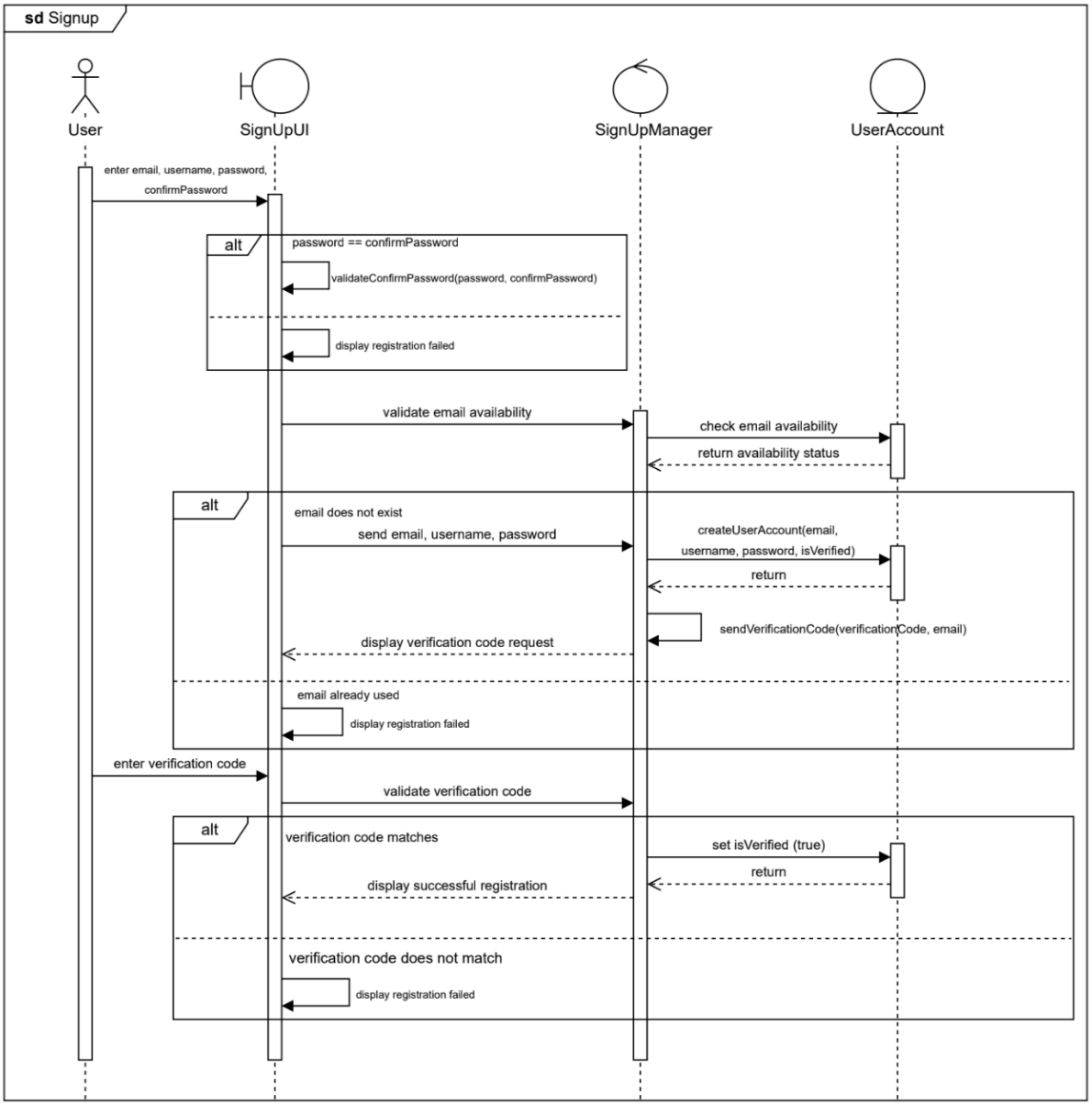

sd searchMRTFares
User
MRT Tab
Select MRT Fares
viewMRTFares Type
LTA MRT Fares Rates
select MRT
select Fares
get (MRT Fares)
find (MRT Fares)
alt
mrtFaresData == true
found
display(MRT Fares)
else
not found
display (Error getting fares)

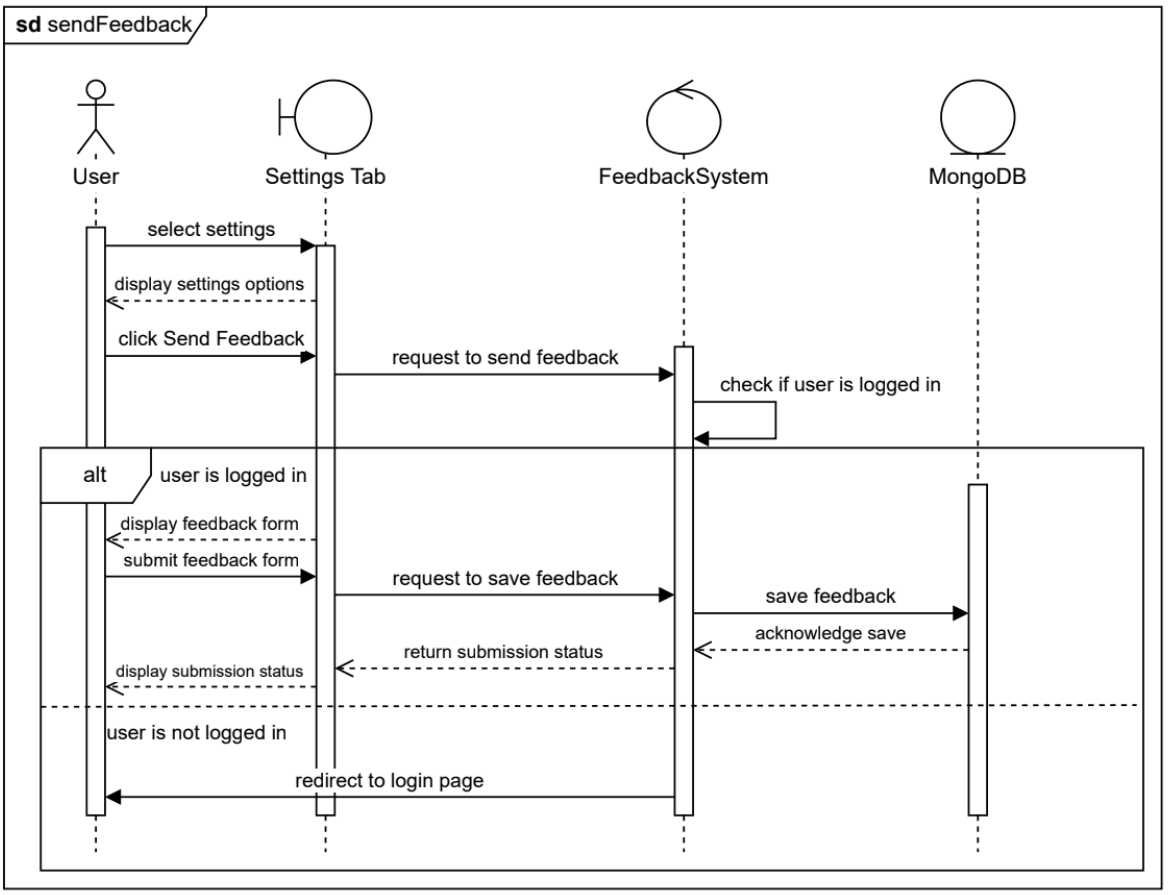

sd viewTaxiFares
User
Taxi Tab
Select Taxi Fares
viewTaxiFares Type
LTA Taxi Fares Rates
select Taxi
select Fares
get (Taxi Fares)
find (Taxi Fares)
alt
found
taxiFaresData
== True
display (Taxi Fares)
else
not found
display (error message)

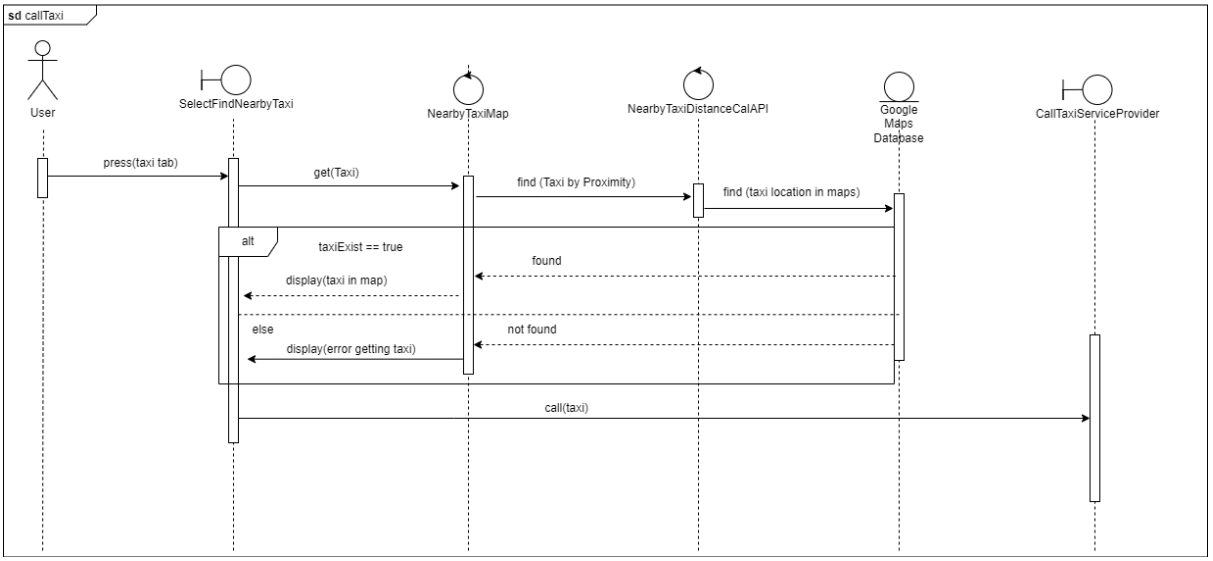

sd viewMRTCrowd
User
MRT Tab
Select MRT Station
ViewCrowdLevel
LTA Database
select MRT
select MRT Station
request MRT Crowd Level
find MRT Crowd Level
alt
crowd level data found
found
return crowd level
display crowd level
else
not found
return error message
display error message

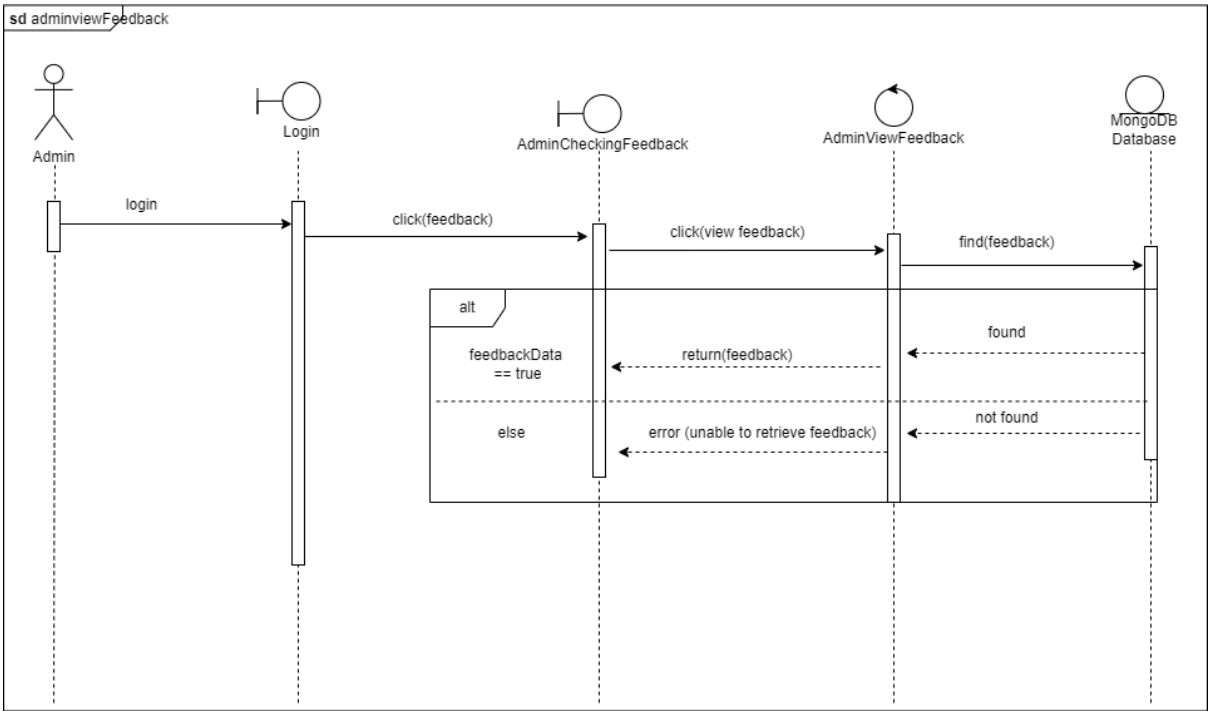

sd viewBusTiming
User
Bus Tab
Select Bus Station
BusAPI
LTA Database
select Bus
select Bus Station
request Bus Arrival Times
find Bus Arrival Times
alt
Bus Timing found
found
return Bus Arrival Times
display Bus Arrival Times
else
not found
return error message
display error message

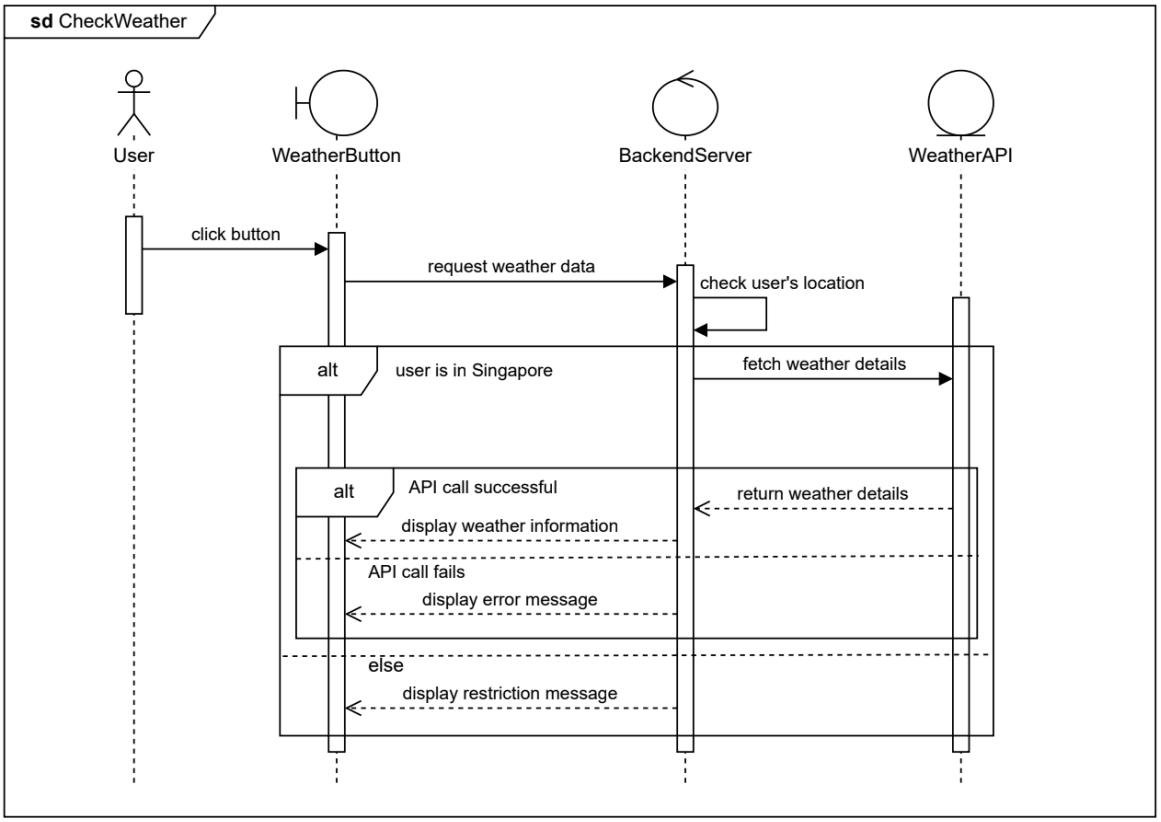

sd viewBusFares
User
Bus Tab
Select Bus Fares
viewBusFares Type
LTA Bus Fares Rates
select Bus
select Fares
get (Bus Fares)
find (Bus Fares)
alt
found
busFaresData
== True
display (Bus Fares)
else
not found
display (error message)

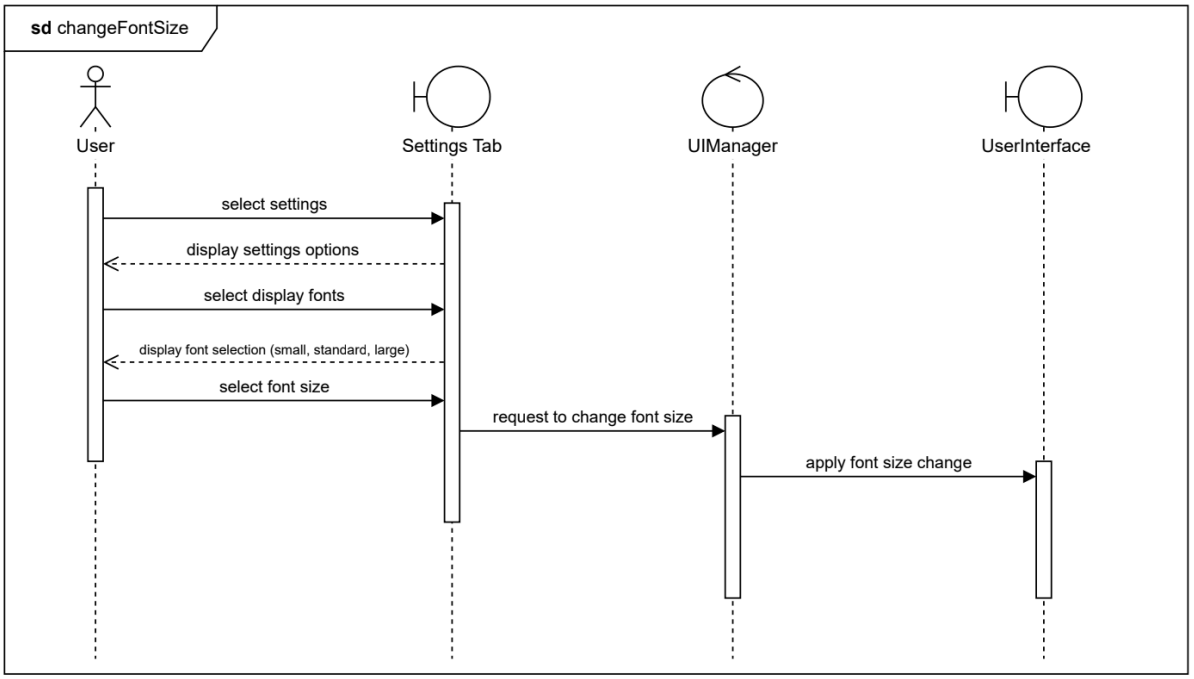

sd Signup
User
SignUpUI
SignUpManager
UserAccount
enter email, username, password, confirmPassword
alt
password == confirmPassword
validateConfirmPassword(password, confirmPassword)
display registration failed
validate email availability
check email availability
return availability status
alt
email does not exist
send email, username, password
createUserAccount(email, username, password, isVerified)
return
sendVerificationCode(verificationCode, email)
display verification code request
email already used
display registration failed
enter verification code
validate verification code
alt
verification code matches
set isVerified (true)
return
display successful registration
verification code does not match
display registration failed

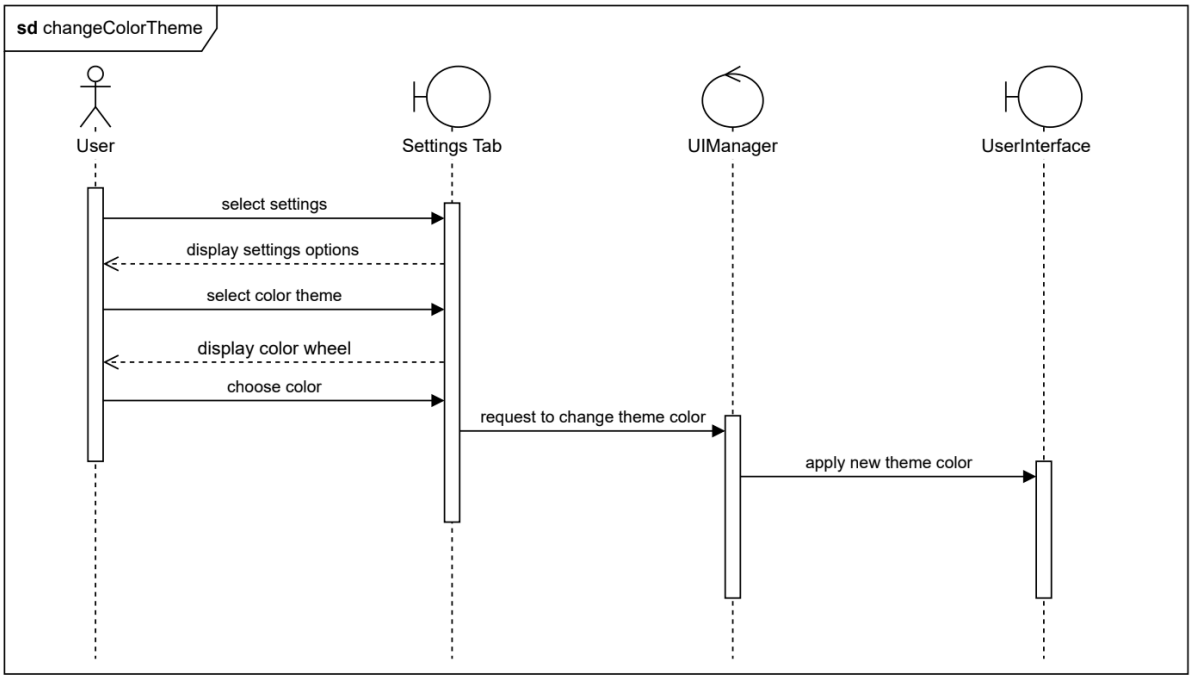

sd sendFeedback
User
Settings Tab
FeedbackSystem
MongoDB
select settings
display settings options
click Send Feedback
request to send feedback
check if user is logged in
alt
user is logged in
display feedback form
submit feedback form
request to save feedback
save feedback
acknowledge save
return submission status
display submission status
user is not logged in
redirect to login page

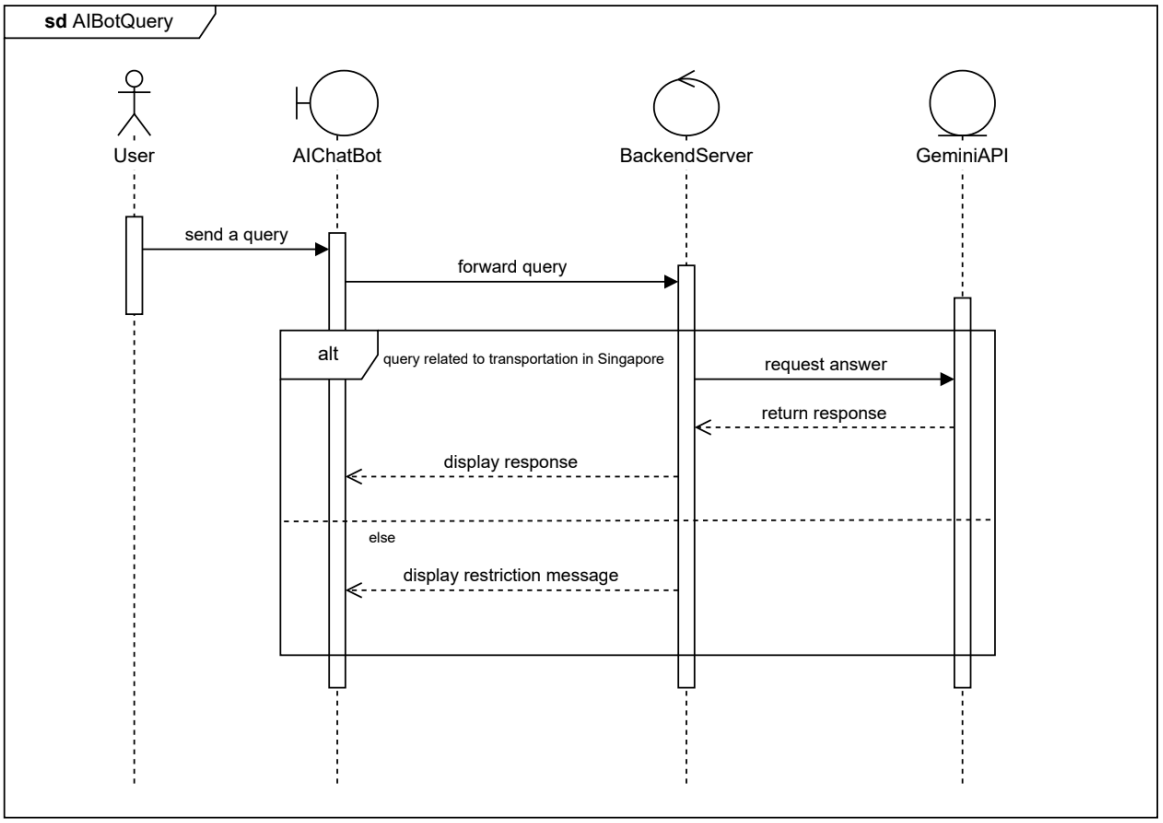

sd callTaxi
User
SelectFindNearbyTaxi
NearbyTaxiMap
NearbyTaxiDistanceCalAPI
Google Maps Database
CallTaxiServiceProvider
press(taxi tab)
get(Taxi)
find (Taxi by Proximity)
find (taxi location in maps)
alt
taxiExist == true
found
display(taxi in map)
else
not found
display(error getting taxi)
call(taxi)

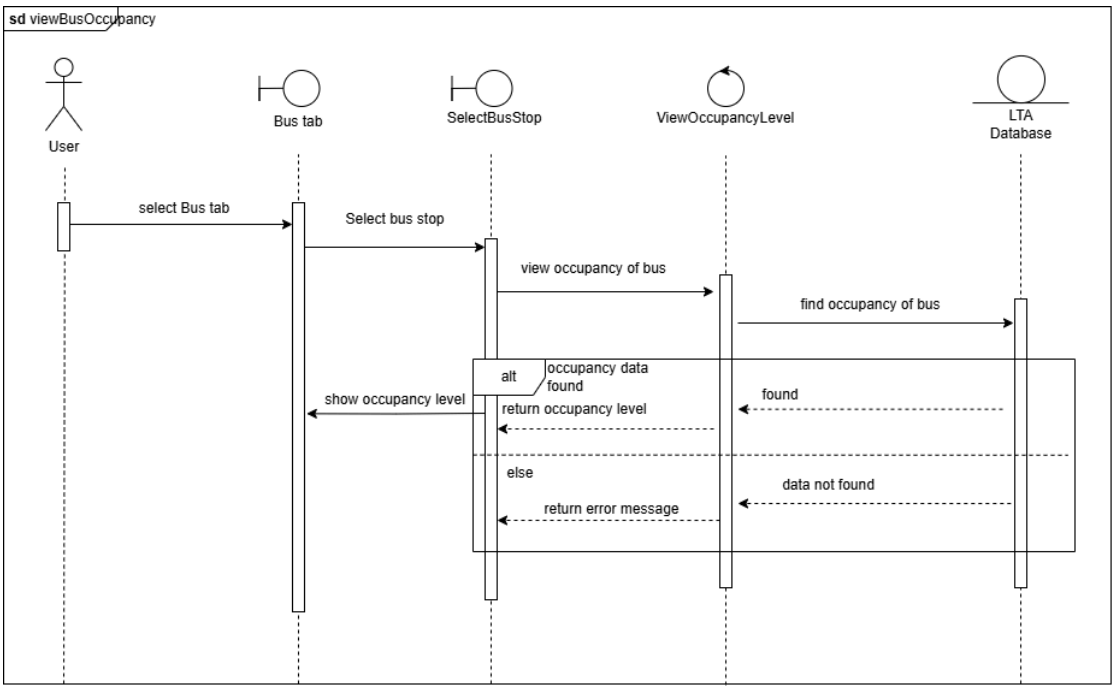

sd adminviewFeedback
Admin
Login
AdminCheckingFeedback
AdminViewFeedback
MongoDB Database
login
click(feedback)
click(view feedback)
find(feedback)
alt
found
feedbackData == true
return(feedback)
else
not found
error (unable to retrieve feedback)


sd CheckWeather
User
WeatherButton
BackendServer
WeatherAPI
click button
request weather data
check user's location
alt
user is in Singapore
fetch weather details
alt
API call successful
return weather details
display weather information
API call fails
display error message
else
display restriction message


sd changeFontSize
User
Settings Tab
UIManager
UserInterface
select settings
display settings options
select display fonts
display font selection (small, standard, large)
select font size
request to change font size
apply font size change


sd changeColorTheme
User
Settings Tab
UIManager
UserInterface
select settings
display settings options
select color theme
display color wheel
choose color
request to change theme color
apply new theme color


sd AIBotQuery
User
AIChatBot
BackendServer
GeminiAPI
send a query
forward query
alt
query related to transportation in Singapore
request answer
return response
display response
else
display restriction message



Dialog map

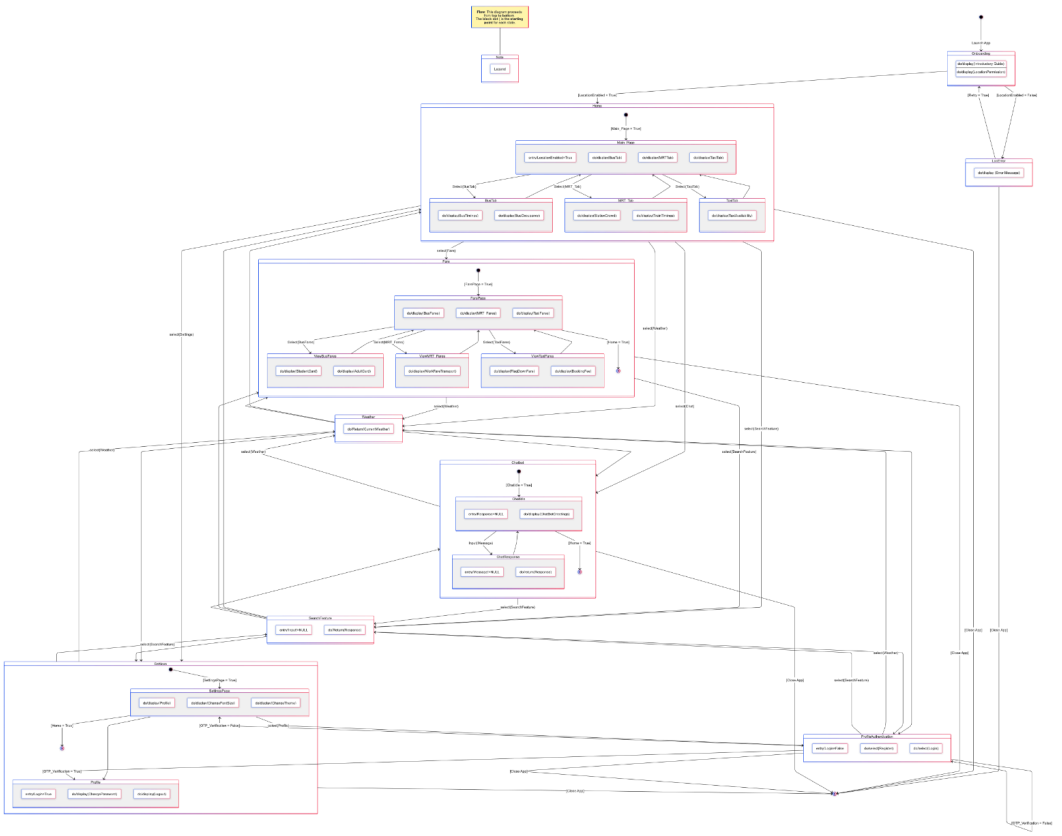


Figure 1: Initial Dialog Map (Please refer to a separate PDF file for a clearer illustration)

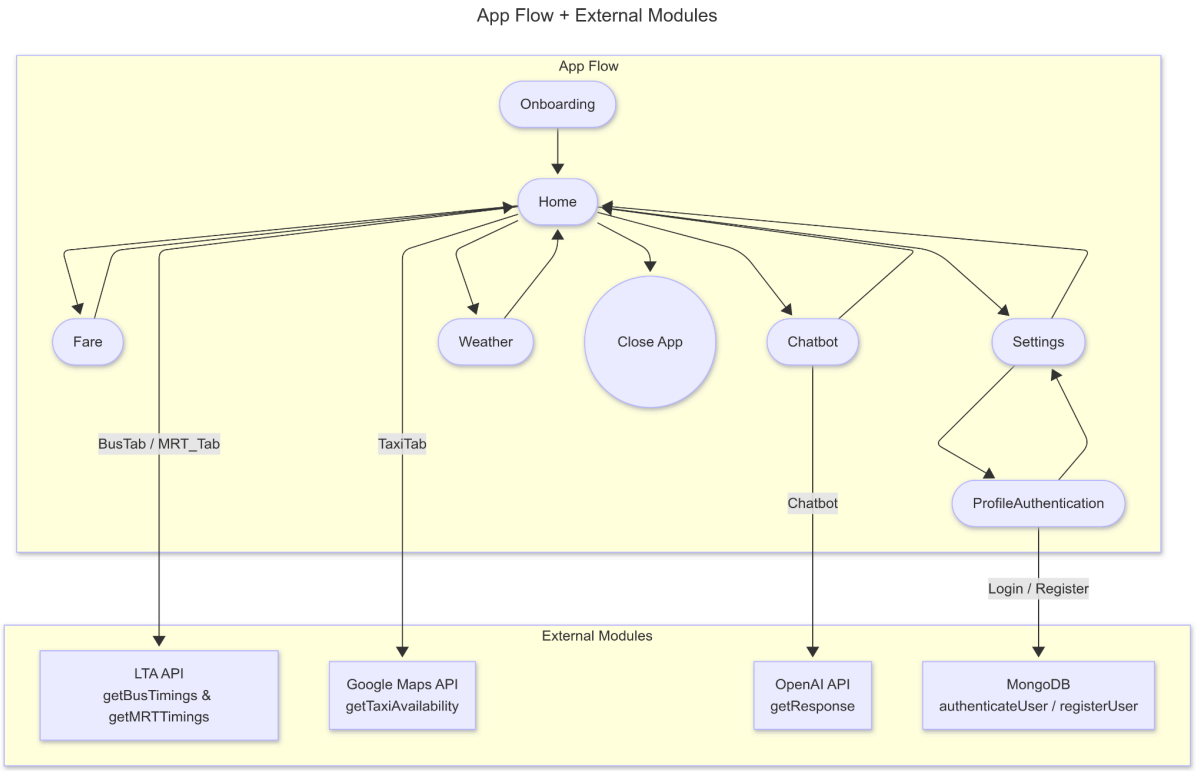
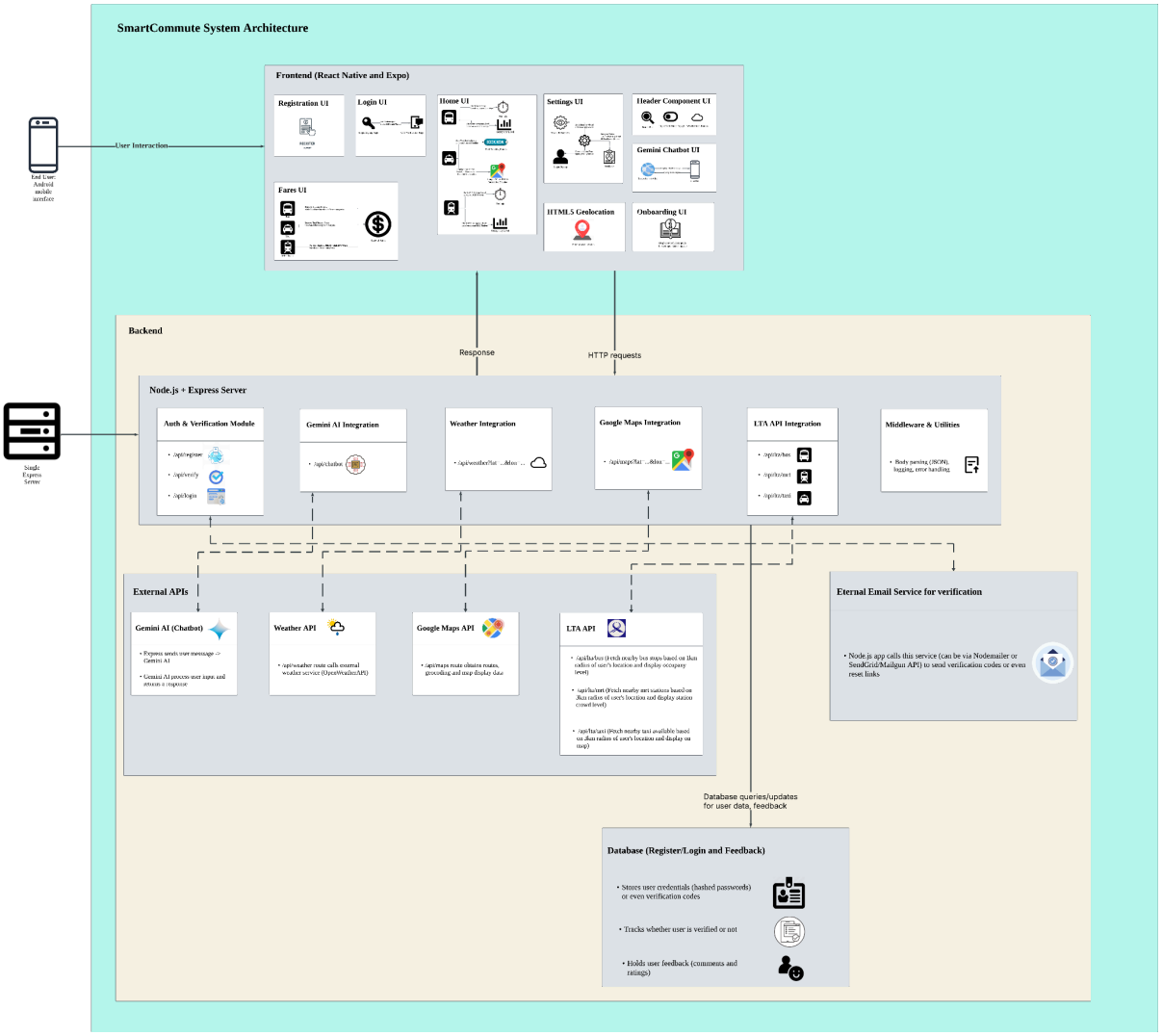


Figure 2: Initial Dialog Map extension showing the app flow interaction with external modules

System Architecture



Application Skeleton

Backend (application/backend/)

1. src/controllers/

Responsible for handling logic behind API endpoints.

File	Description
authController.ts	Handles user authentication (login, logout, password reset).
busController.ts	Retrieves bus routes and real-time updates.
busStopController.ts	Provides nearby bus stop data and estimated timings.
busStopNameController.ts	Fetches or modifies bus stop names and IDs.
chatbotController.ts	Processes chatbot queries, returns route/fare suggestions.
fareRouteController.ts	Calculates and provides fare estimates for routes.
taxiController.ts	Manages taxi availability.
trainController.ts	Retrieves seat availability.
weatherController.ts	Fetches current weather conditions and forecasts.

2. src/models/

Contains schema-like TypeScript classes.

File	Description
User.ts	Defines the structure and types for a user object (email, name, roles).

3. src/middleware/

Houses logic that executes between incoming requests and final route handling.

File	Description
auth.ts	Handles authentication checks (e.g., token validation, session verification) and ensures secure access to protected resources.

4. src/routes/

Defines API endpoints and maps them to controllers.

File	Description
authRoutes.ts	Auth-related API routes (e.g., /login, /signup).
busRoutes.ts	API routes related to bus travel.
chatbotRoutes.ts	Chatbot communication API route.

fareRoute.ts	Exposes endpoints to fetch fare calculation data.
taxiRoutes.ts	Endpoints related to taxi services.
trainRoutes.ts	Handles endpoints for retrieving seat availability.
weatherRoutes.ts	Fetches current weather conditions and forecasts.

5. src/services/
Utility logic or helper modules.

File	Description
api.ts	Centralized API request/response handler or wrappers.

Frontend (application/frontend/)

1. components/
Reusable, modular UI parts.

File	Description
BusStopSearch.tsx	Component for searching nearby bus stops.
Bushomelayout.tsx	Home layout specifically for bus interface.
Layout.tsx	Generic page layout wrapper with styling.
ServerIP.tsx	UI and logic to configure app server IP address.
Taxihomelayout.tsx	Layout used for taxi feature landing.
TrainHomeLayout.tsx	Layout used for train crowd level.
ThemeContext.tsx	Used to set theme colors and dark/light mode.
locationservice.tsx	To fetch location coordinates, used for getting nearby taxis/trains/buses.

2. screens/
Full app views/screens rendered through navigation.

File	Description
ChangePassword.tsx	Screen to change user password.
Chatbotpage.tsx	UI for interacting with the built-in chatbot.
FareRouteMap.tsx	Map showing route and fare visually.
Farepage.tsx	Shows estimated fare breakdown for a trip.
Forgetpasswordpage.tsx	Recovery screen to request password reset link.
Permissionsscreen.tsx	Screen to request for permission for location.
ProfilePage.tsx	To display user’s details (premium/lite) and to send feedback.
Homepage.tsx	Homepage that shows taxi/bus/train details, it encompasses the bus/train/taxi layout.
Onboardingscreen.tsx	Initial page, when u first install the application to display what the application does.

Settingpage.tsx	Settings page for user to view profile and set applications theme color.
Loginpage.tsx	Page to login as a user to access feedbacks.
LandingPage.tsx	Initial landing page when opening app.
ProfilePageAdmin.tsx	To display admin's profile page and view feedback submitted.
SendFeedbackpage.tsx	Screen for users to send feedback for the application.
Signuppage.tsx	Signup page for application.
successfulpage.tsx	Page to display login/register successfully.
verificationpage.tsx	OTP page to generate and enter otp when logging in/registering.

3. data/

JSON files storing app data.

File	Description
fares.json	Static fare price table used for offline calculation.
publicTransportTimings.json	Estimated arrival and departure times for transport.

4. styling/

Contains all the style resources that define the visual design of the application, ensuring consistent look and feel across different components or screens.

File	Description
Bushomelayout.styles.ts	Stylesheet for bus timings.
Taxihomelayout.styles.ts	Stylesheet for taxi home layout.
Trainhomelayout.styles.ts	Stylesheet for train home layout.
Farepagedynamic.styles.ts	Stylesheet for farepage.
Homepage.styles.ts	Stylesheet for homepage.
Layout.styles.ts	Stylesheet for layout overlay.
Sendfeedback.styles.ts	Stylesheet for sending feedback page.
Settingpage.styles.ts	Stylesheet for settings page.

Appendix

Technology Stack Used

- Backend:
  - Node.js – JavaScript runtime for scalable server-side applications
  - Express – Web framework for building RESTful APIs
  - TypeScript – Strongly typed superset of JavaScript
  - Jest – Testing framework for backend logic
- Frontend:
  - React Native – Cross-platform mobile app development
  - Expo – Framework and platform for universal React apps
  - TypeScript – Type-safe components and props
  - Tailwind CSS – Utility-first styling (via NativeWind)
  - i18n – JSON-based internationalization

Recommendations

1. Ensure modularity in both frontend and backend by separating concerns clearly.
2. Use environment variables for configurations like server IP and API base URLs.
3. Implement error handling middleware in the backend for consistency.
4. Add form validation and user feedback (e.g., Toasts, alerts) in the frontend.
5. Consider automated testing (unit + integration) using Jest and React Testing Library.
6. Add Docker support for easier deployment and environment setup.