NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

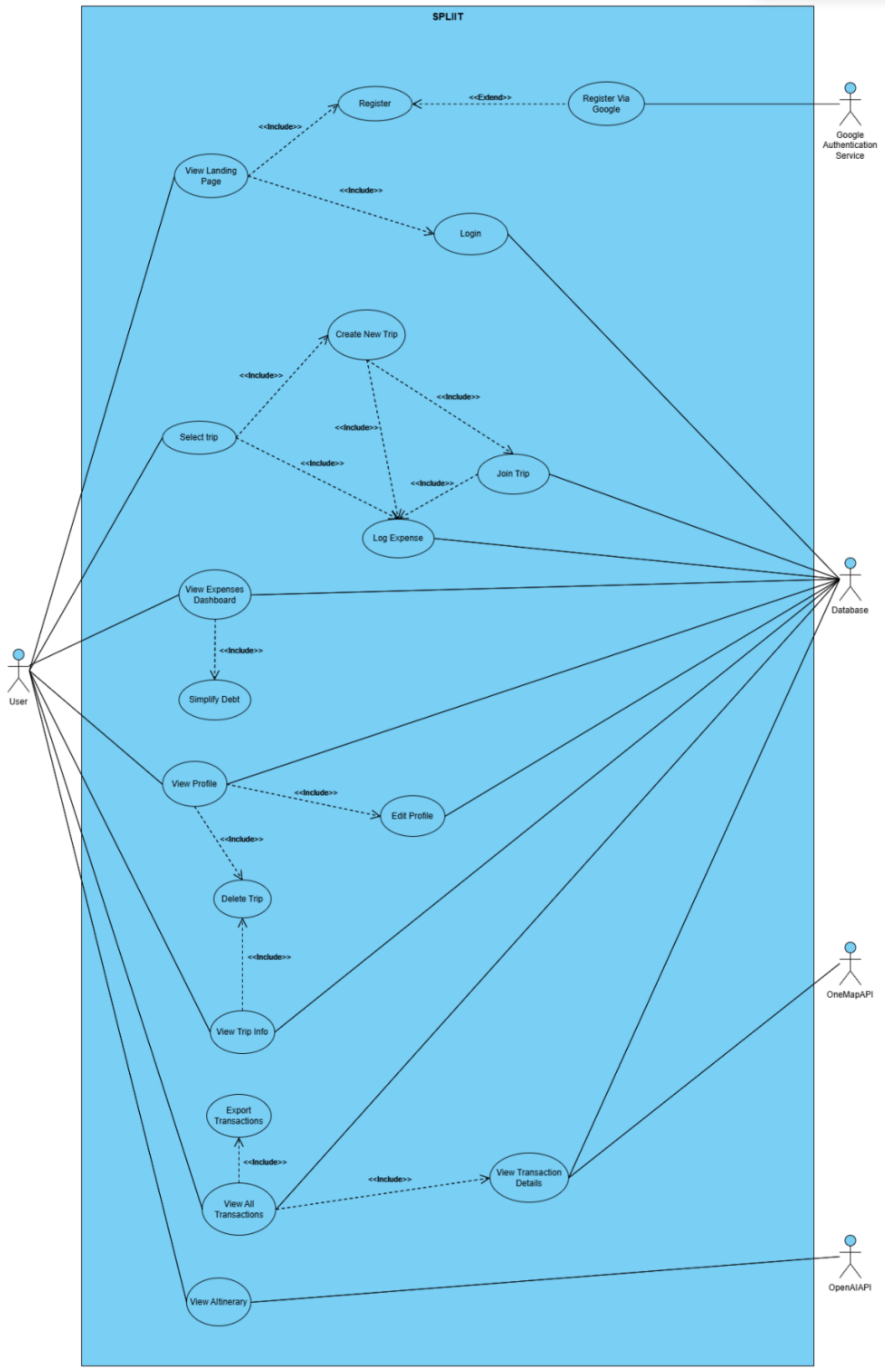
SC2006 – Software Engineering

Lab 3: Design and Implementation

Lab Group	SCMB	
Team	SPLIIT	
Members	Goh Jun Keat	
	Koh Ze Kai Leo	
	Madhumita Thiruppathi	
	Siah Yee Long	
	Teo Liang Wei, Ryan	
	Zhang Yichi	

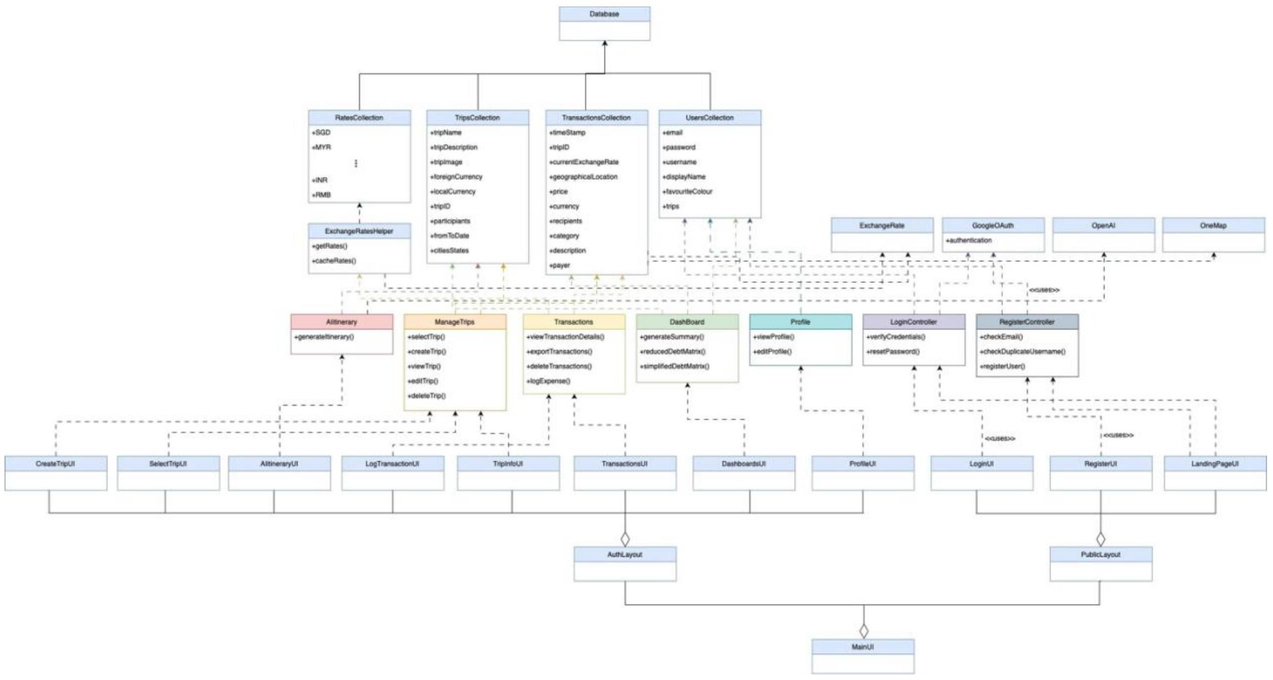
Table of Contents

Complete Use Case model



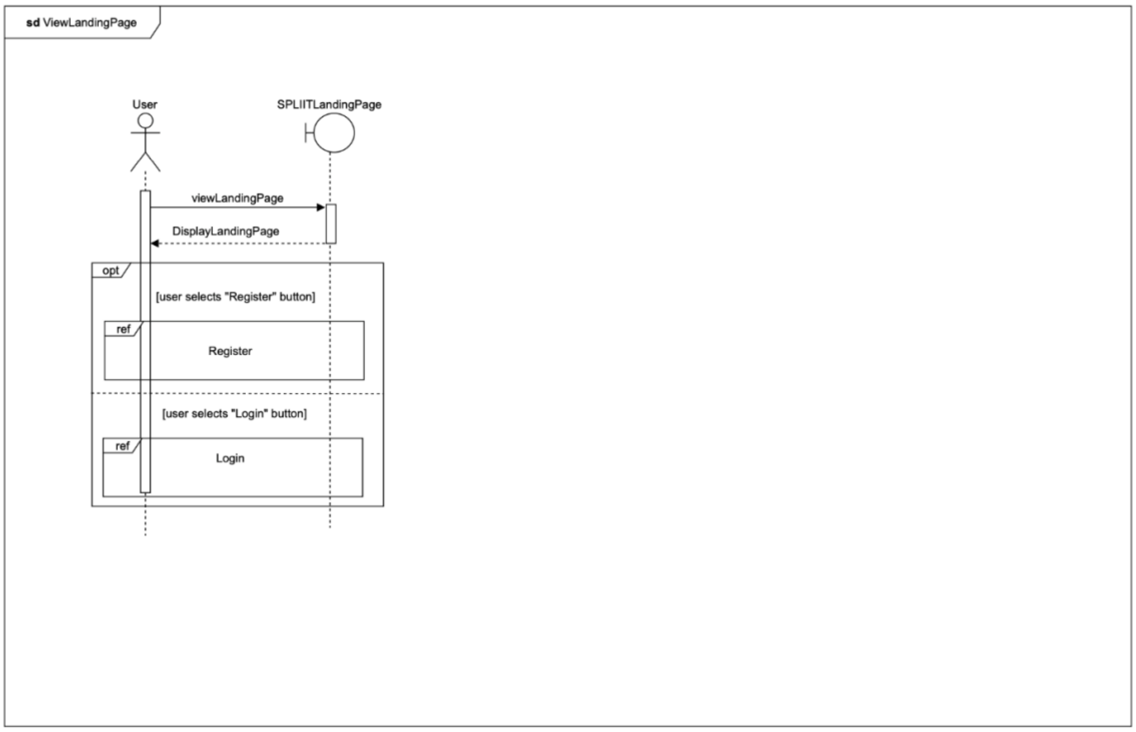
Design Model

Class diagram

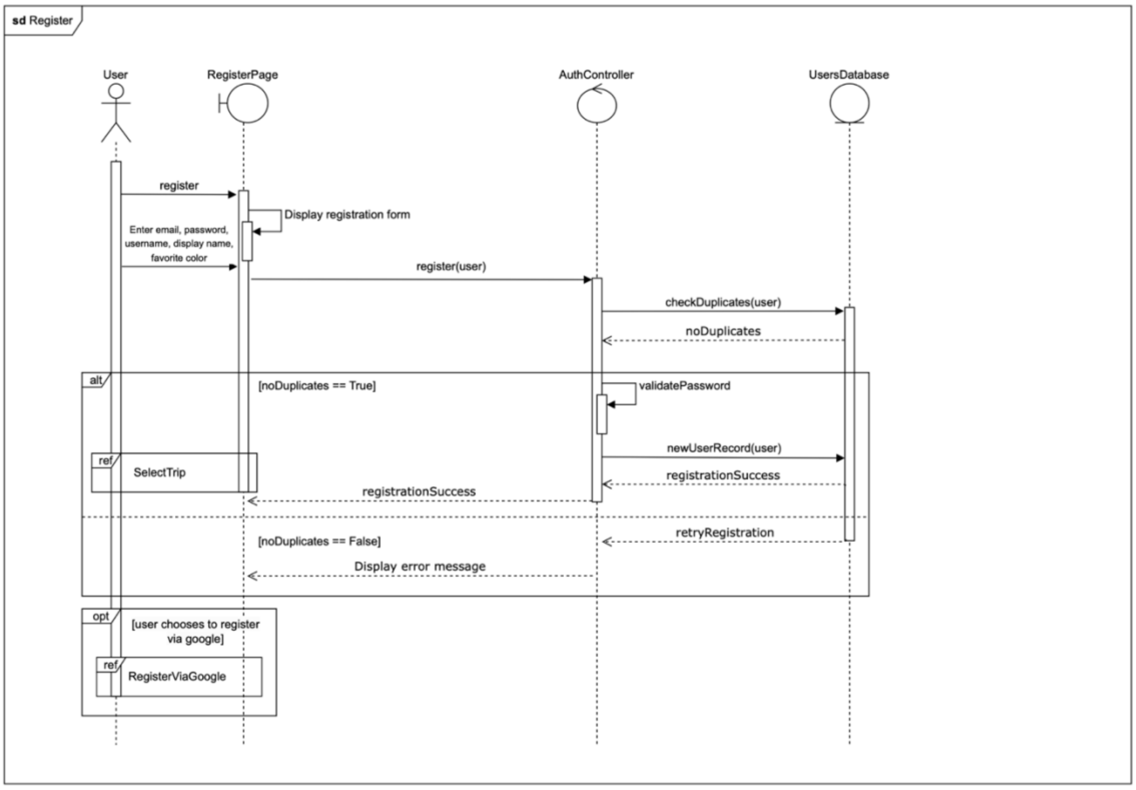


Sequence diagrams

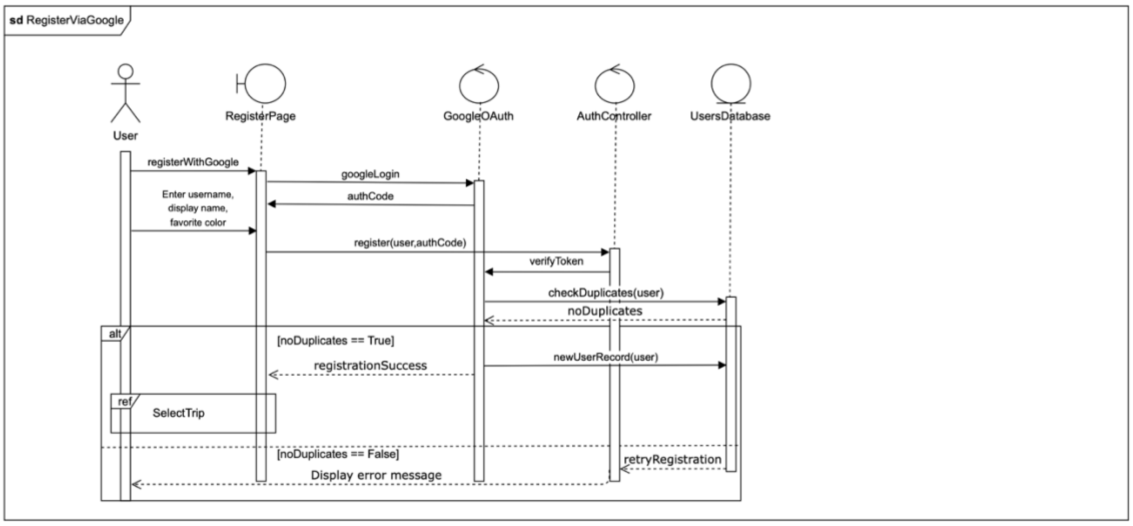
View Landing Page



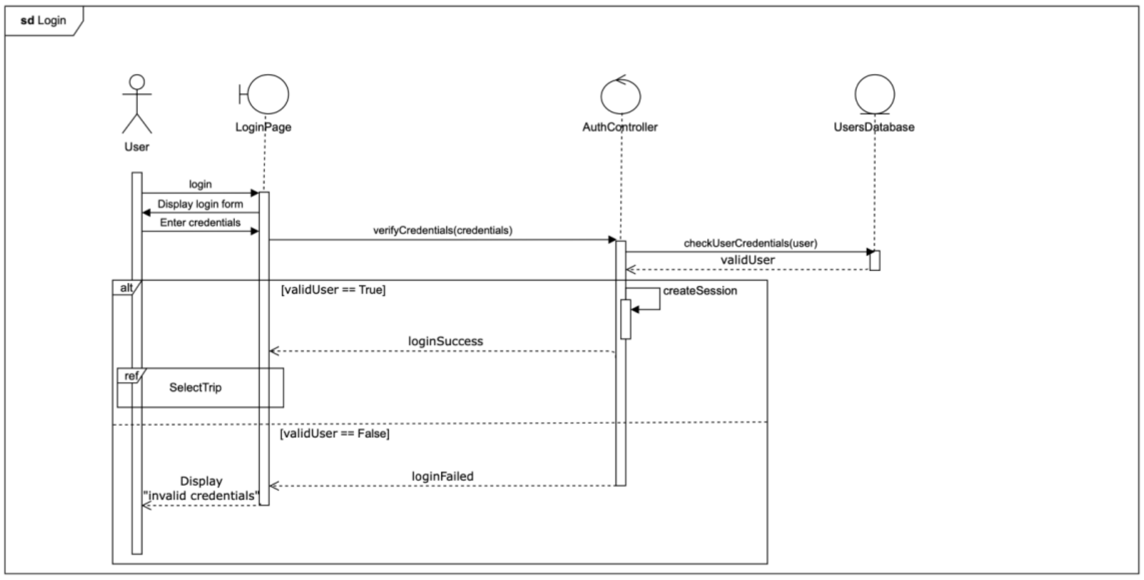
Register



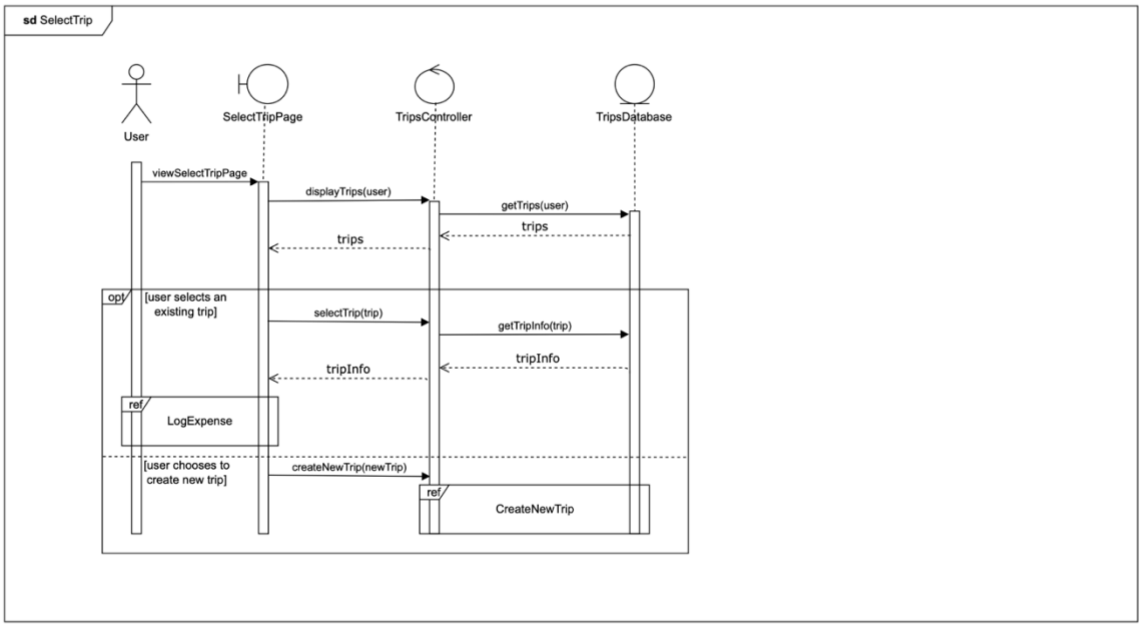
Register via Google



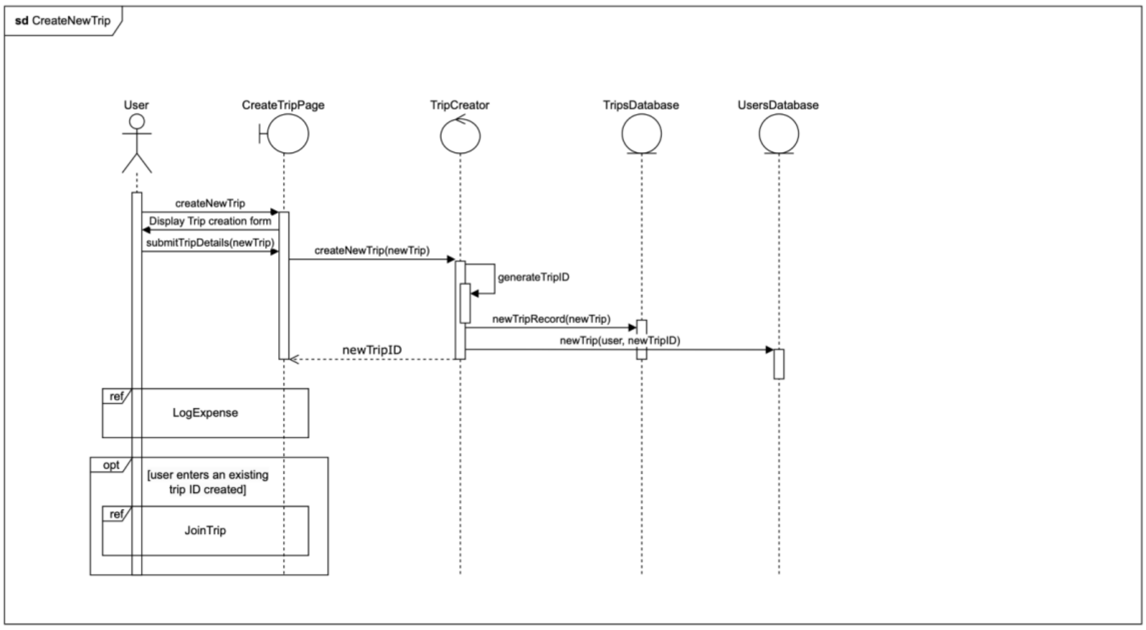
Login



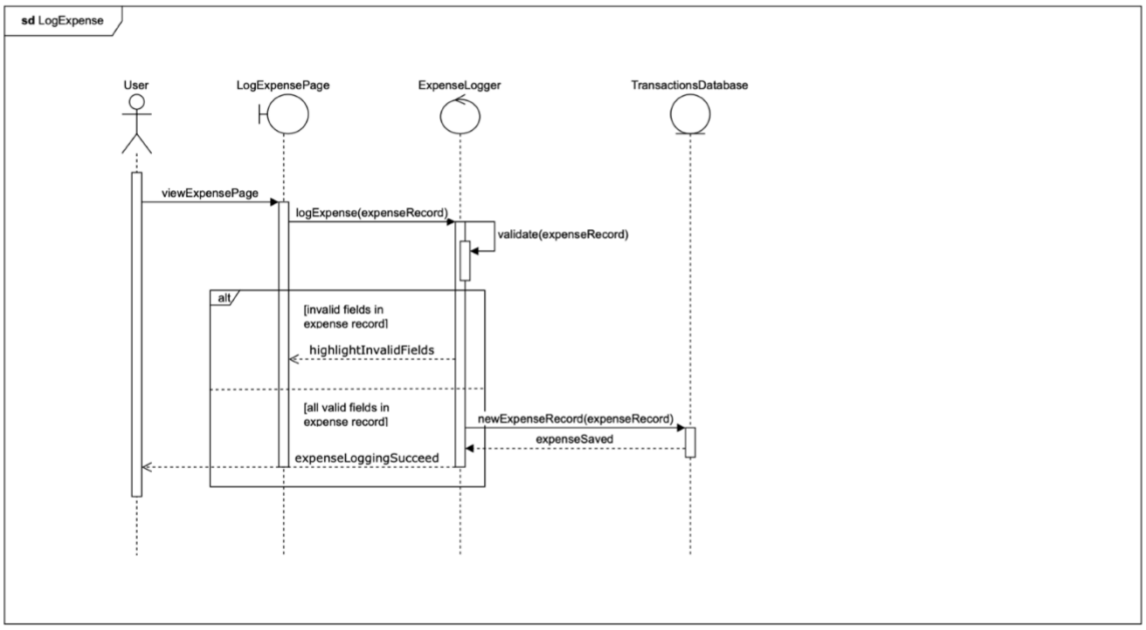
Select Trip



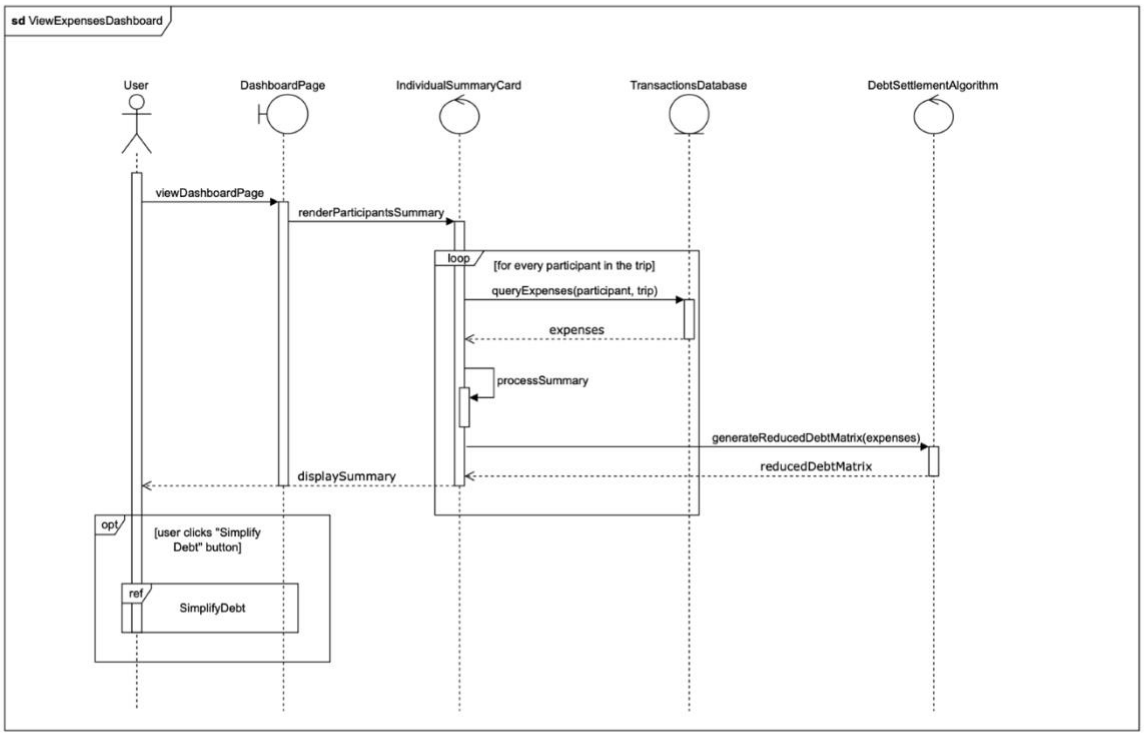
Create New Trip



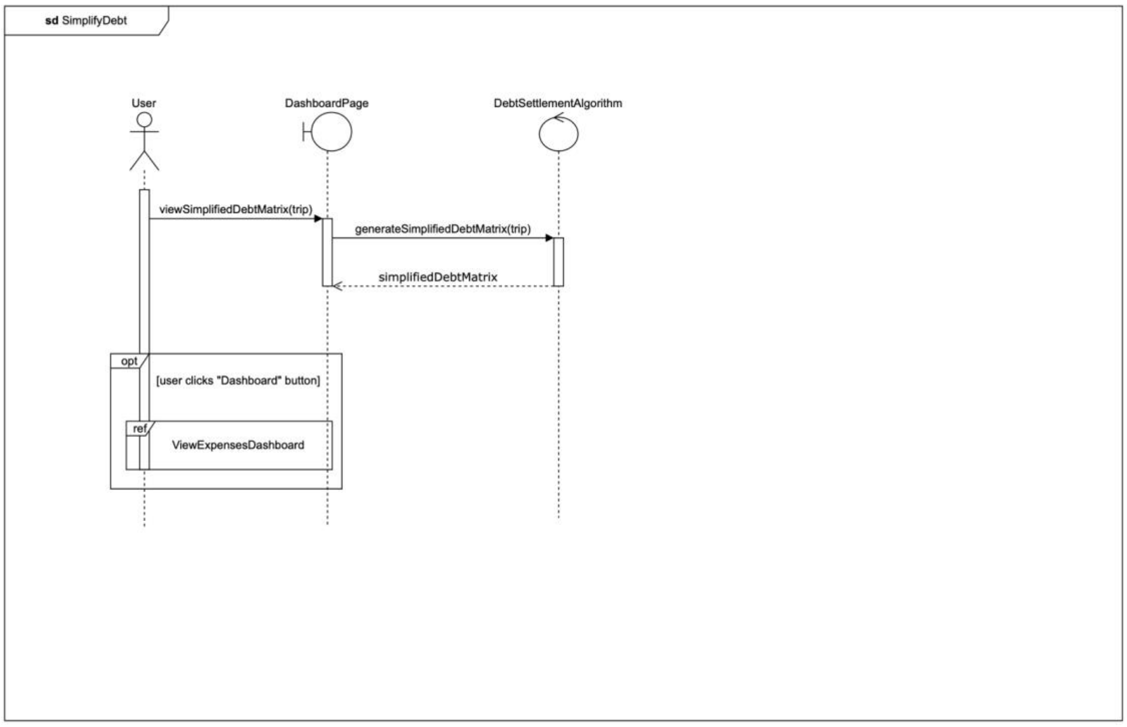
Log Expense



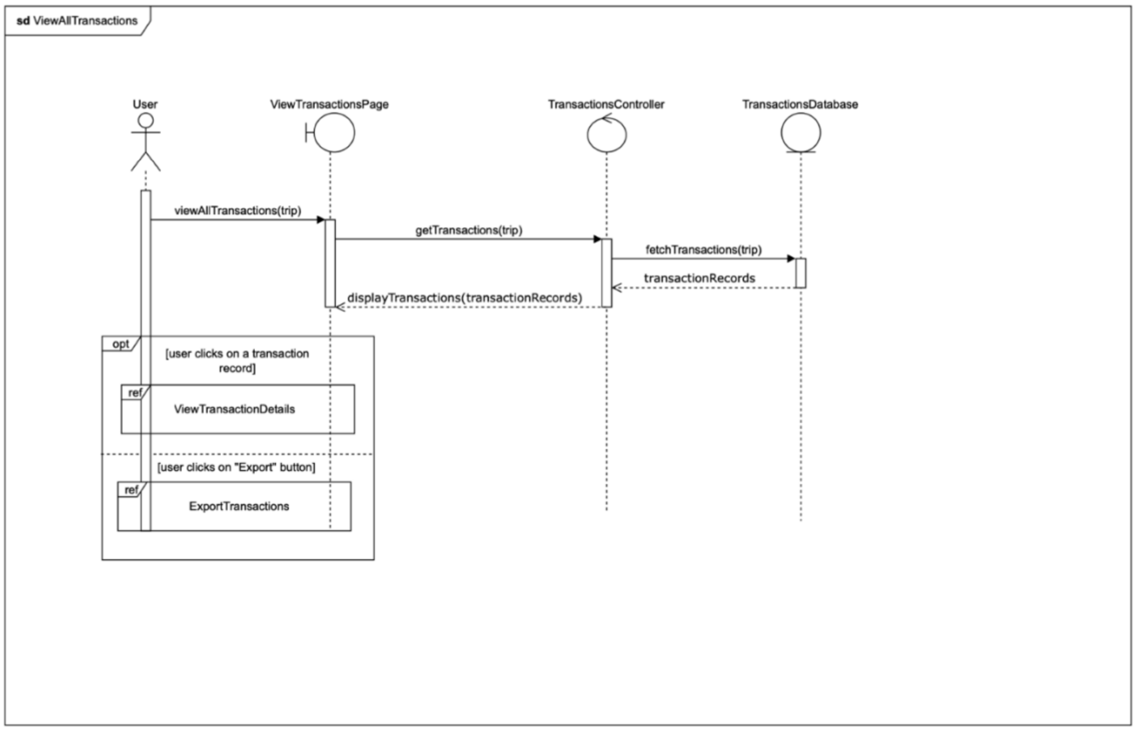
View Expenses Dashboard



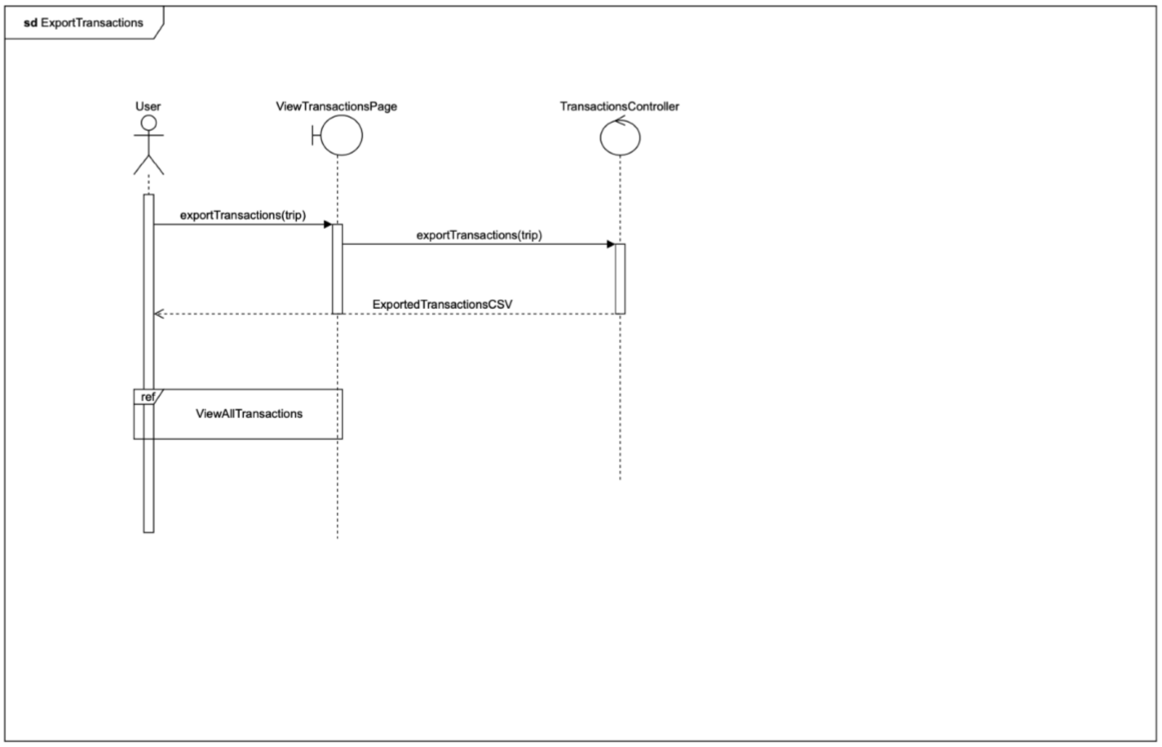
Simplify Debt



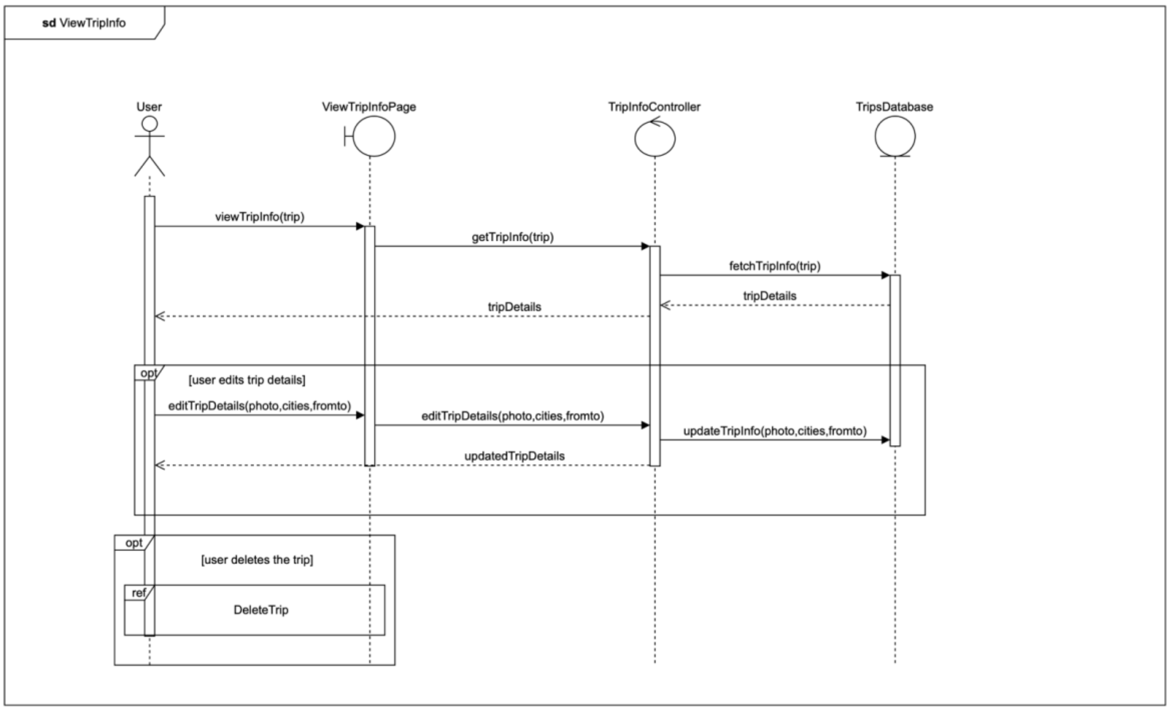
View All Transactions



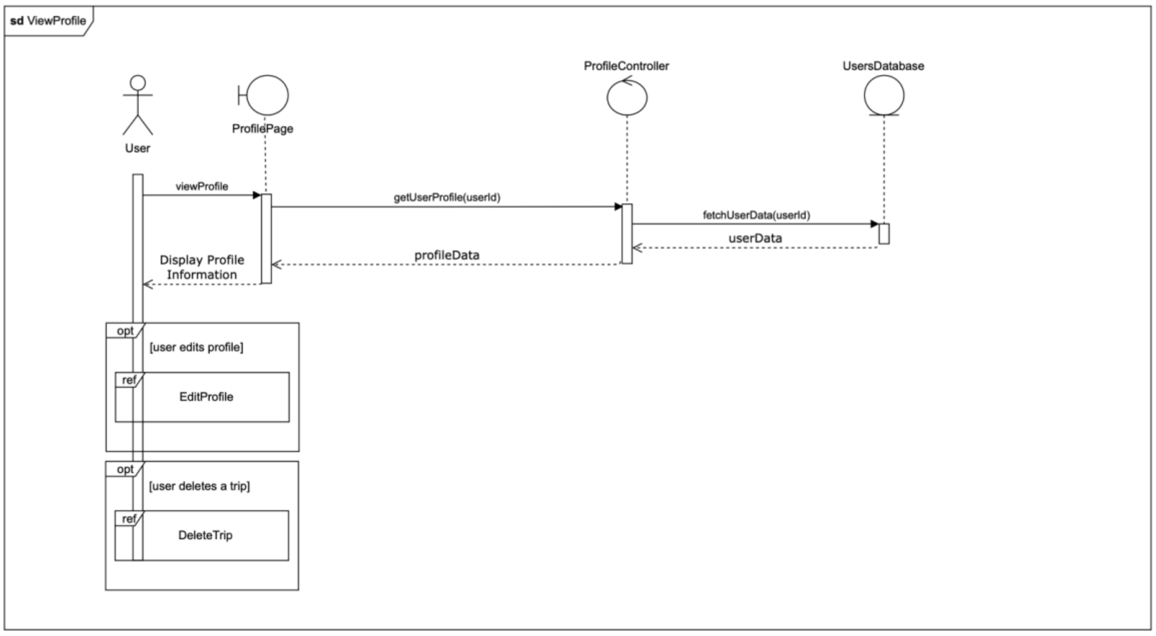
Export Transactions



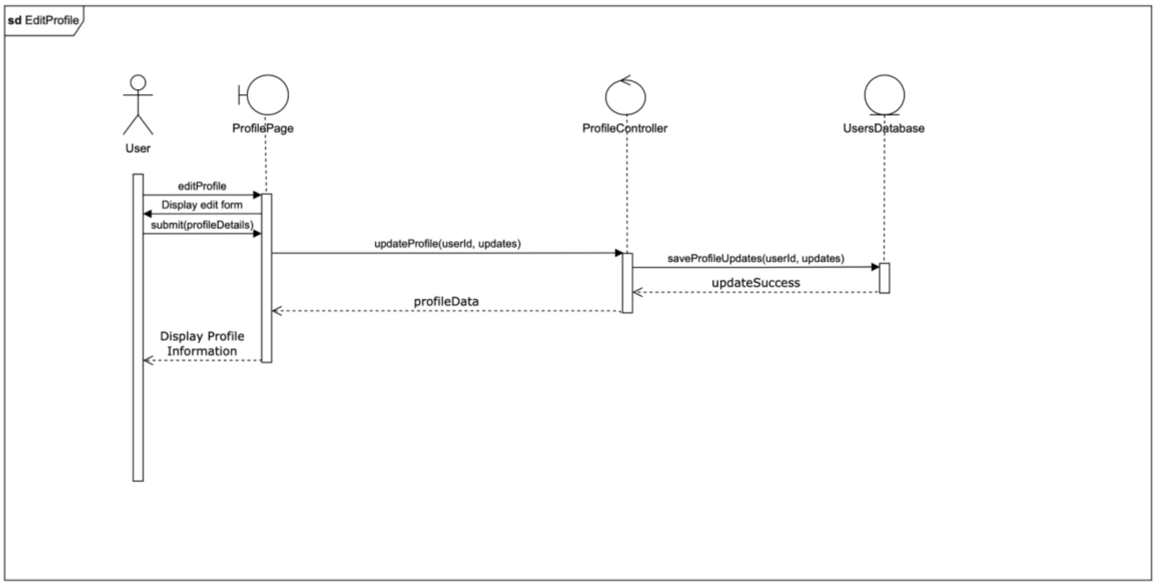
View Trip Info



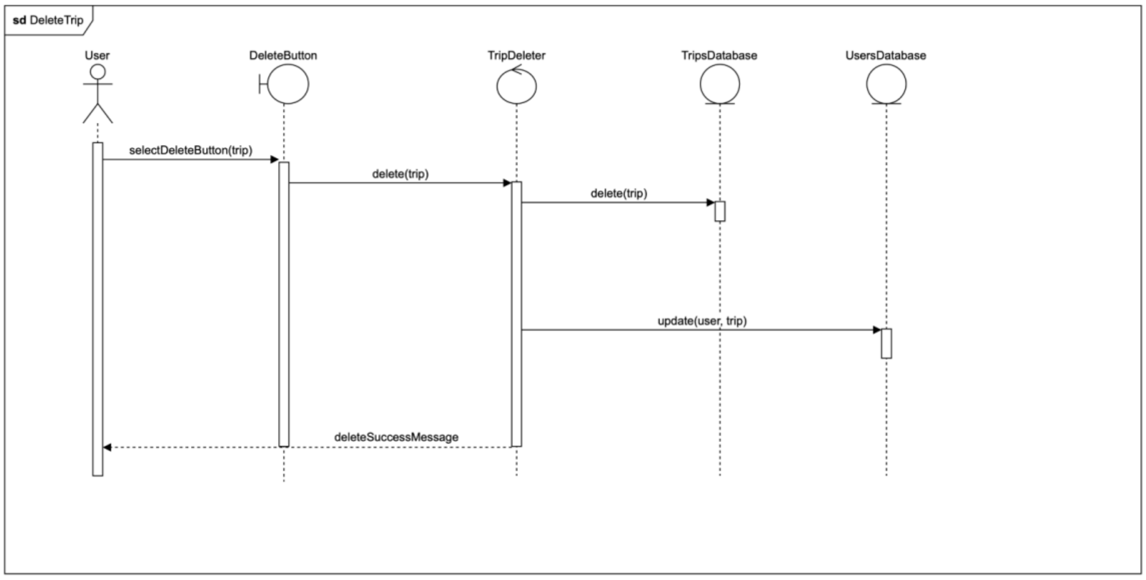
View Profile



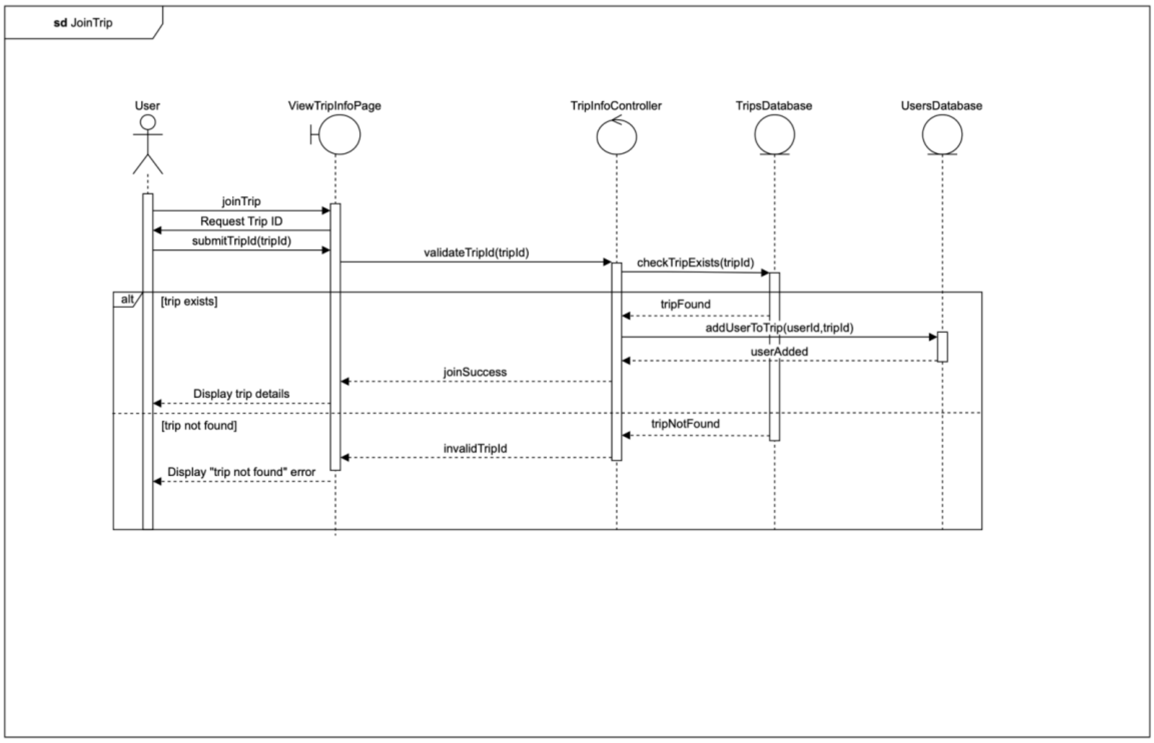
Edit Profile



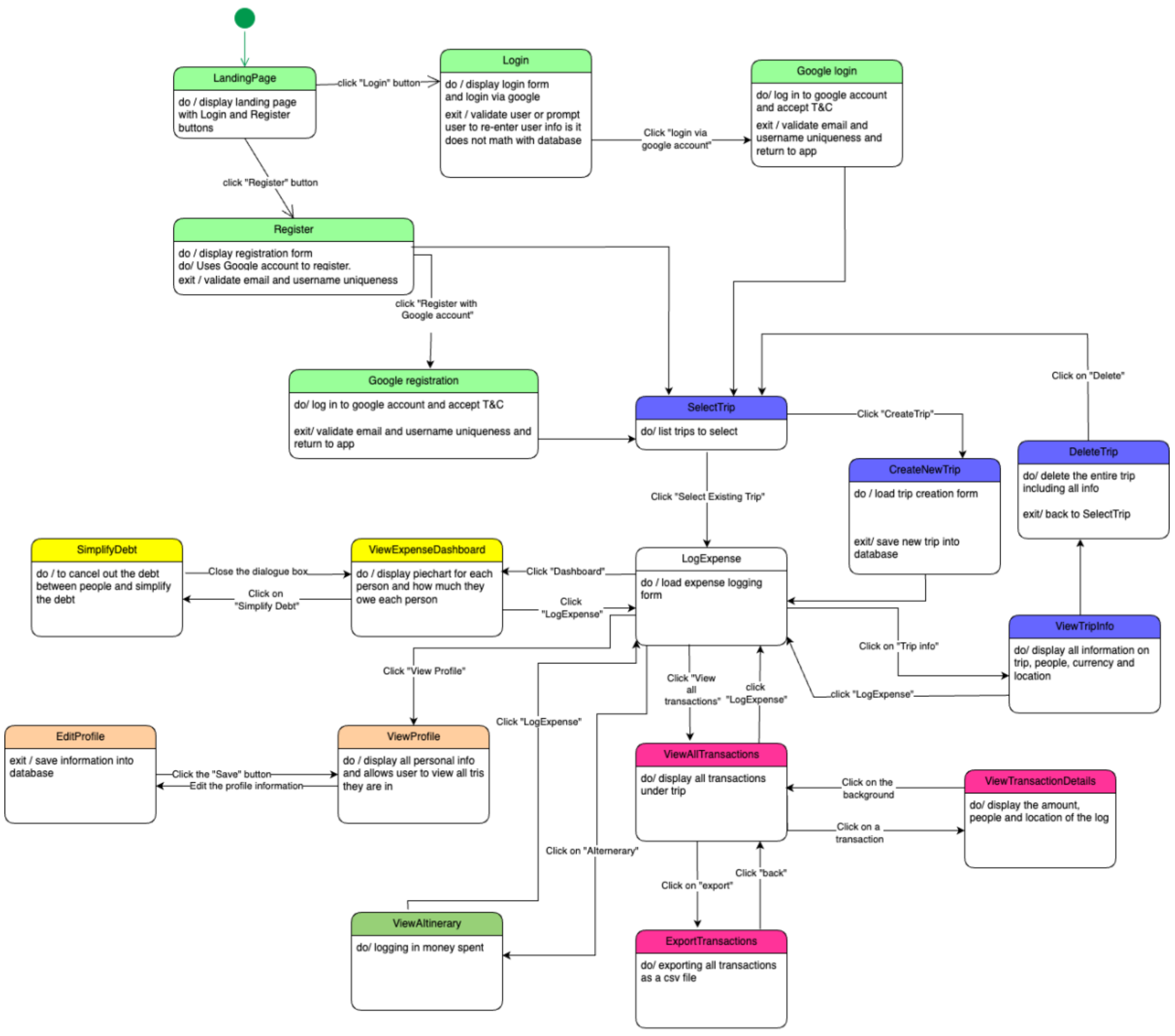
Delete Trip



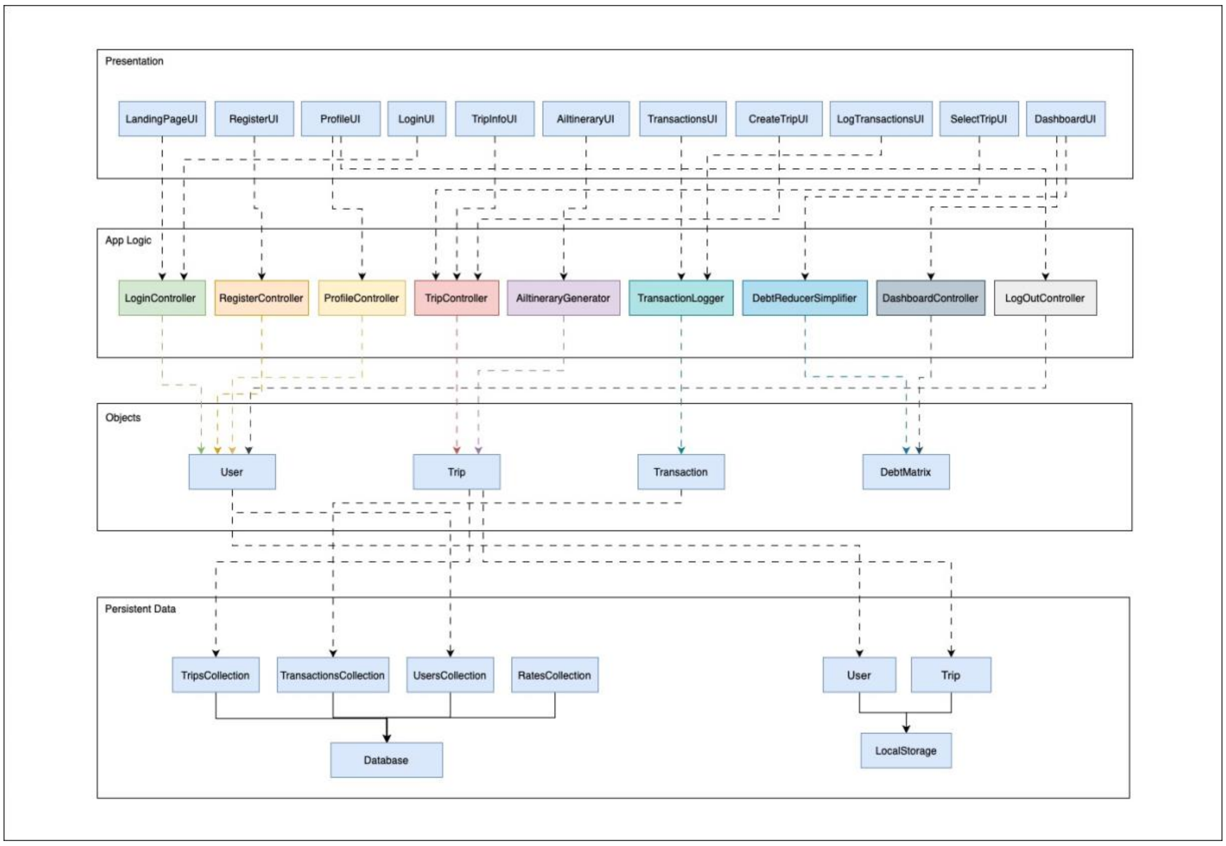
Join Trip



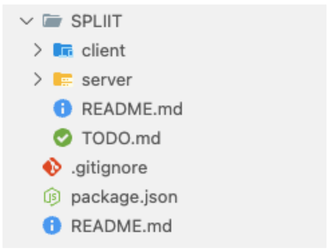
Dialog map


LandingPage
do / display landing page with Login and Register buttons
click "Login" button
Login
do / display login form and login via google
exit / validate user or prompt user to re-enter user info is it does not math with database
Click "login via google account"
Google login
do/ log in to google account and accept T&C
exit / validate email and username uniqueness and return to app
click "Register" button
Register
do / display registration form
do/ Uses Google account to register.
exit / validate email and username uniqueness
click "Register with Google account"
Google registration
do/ log in to google account and accept T&C
exit/ validate email and username uniqueness and return to app
SelectTrip
do/ list trips to select
Click "CreateTrip"
Click on "Delete"
CreateNewTrip
do / load trip creation form
exit/ save new trip into database
DeleteTrip
do/ delete the entire trip including all info
exit/ back to SelectTrip
Click "Select Existing Trip"
SimplifyDebt
do / to cancel out the debt between people and simplify the debt
Close the dialogue box
Click on "Simplify Debt"
ViewExpenseDashboard
do / display piechart for each person and how much they owe each person
Click "Dashboard"
Click "LogExpense"
LogExpense
do / load expense logging form
Click on "Trip info"
ViewTripInfo
do/ display all information on trip, people, currency and location
click "LogExpense"
Click "View Profile"
Click "LogExpense"
Click "View all transactions"
click "LogExpense"
EditProfile
exit / save information into database
Click the "Save" button
Edit the profile information
ViewProfile
do / display all personal info and allows user to view all tris they are in
ViewAllTransactions
do/ display all transactions under trip
Click on the background
Click on a transaction
ViewTransactionDetails
do/ display the amount, people and location of the log
Click on "Alternerary"
Click on "export"
Click "back"
ViewAltinerary
do/ logging in money spent
ExportTransactions
do/ exporting all transactions as a csv file

System architecture



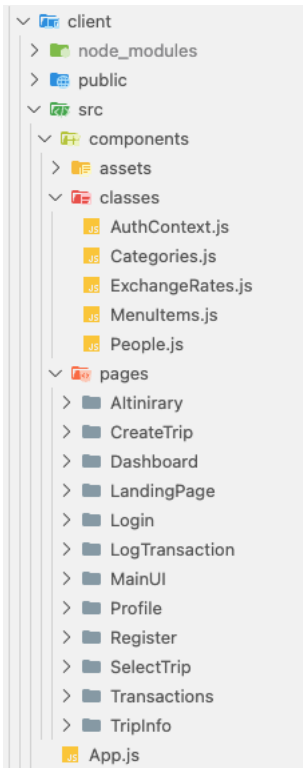
Application skeleton



```
v SPLIIT
  > client
  > server
    README.md
    TODO.md
  .gitignore
  package.json
  README.md
```

The SPLIIT application is split into the frontend (client) and backend (server). Other files such as README.md and TODO.md are used during development.
For detailed implementation, visit our GitHub page at: https://github.com/siahyeelong/SC2006-SPLIIT

Frontend



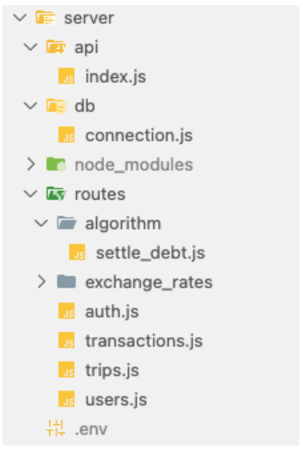
```
v client
  > node_modules
  > public
  v src
    v components
      > assets
      v classes
          AuthContext.js
          Categories.js
          ExchangeRates.js
          MenuItems.js
          People.js
      v pages
        > AItinirary
        > CreateTrip
        > Dashboard
        > LandingPage
        > Login
        > LogTransaction
        > MainUI
        > Profile
        > Register
        > SelectTrip
        > Transactions
        > TripInfo
      App.js
```

- The frontend is built with React. It consists of the relevant classes used and the different pages where information is presented on the webpage. Each component is modular and can be reused across pages, such as the SnackbarNotifs component.

- Within each page, the individual element components can be found. For example, a LoginForm and a GoogleLoginButton component is built so that it can be used in the login page.

- Each component also has the controller functionality to manage the relevant classes and call the relevant APIs.

Backend



```
v server
  v api
      index.js
  v db
      connection.js
  > node_modules
  v routes
    v algorithm
        settle_debt.js
    > exchange_rates
      auth.js
      transactions.js
      trips.js
      users.js
  .env
```

- The backend is built with Express JS, where it contains relevant routings to external and internal APIs.

- The main debt settlement algorithm which we have written is found under the ‘algorithm’ folder.

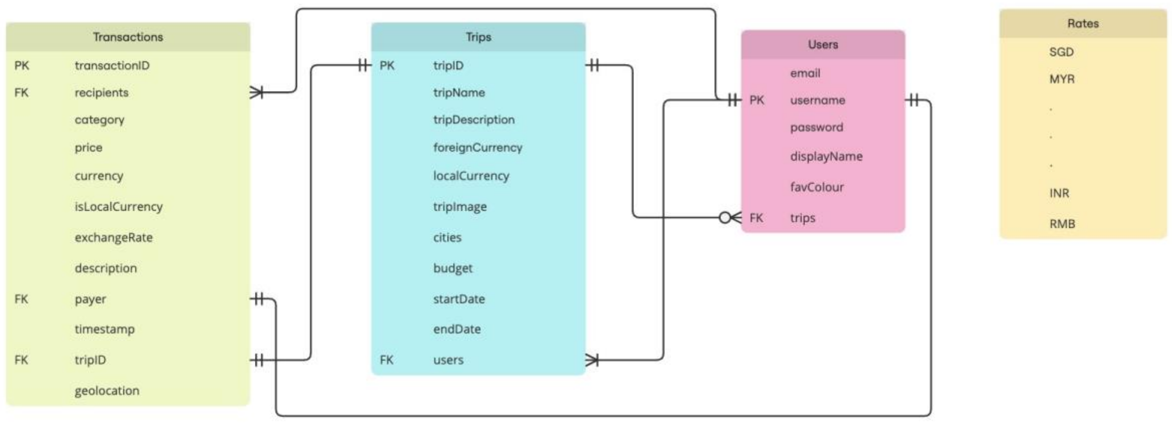
- The database used is MongoDB, where the relevant connection is found under the ‘db’ folder.

Key design issues

Data Persistence

There exist different levels of data persistence in our application.

- Application data:
  - Transactions:
    - Stores information regarding every transaction record
    - Stored in the TransactionsCollection in MongoDB
  - Users:
    - Stores information regarding every user and their associated trips
    - Stored in the UsersCollection in MongoDB
  - Trips:
    - Stores information regarding every trip and their associated participants
    - Stored in the TripsCollection in MongoDB
  - Exchange rates:
    - Stores every currency's exchange rate according to an exchange rate API to act as a cache
    - Stored in the RatesCollection in MongoDB



- Session data:
  - Current active user (username)
    - Stored in local storage in the frontend (client)
  - Current active trip (tripID)
    - Stored in local storage in the frontend (client)
  - User authentication
    - Refresh token
      - Expires in 30 days
      - Stored in HTTP-only cookies in the backend (server)
    - Access token
      - Expires in 1 hour or when user reloads page

- Stored in state memory in the frontend (client)

Access Control

User access in our application is simply split into an authenticated user or a public user. Public users are able to only access the landing, login, and register pages. Once authenticated, they are able to access the rest of the pages in the application.
Hence, an AuthLayout and PublicLayout are implemented to enable and restrict authenticated features to users. If an unauthenticated user attempts to access an authenticated page, the user will simply be redirected back to the landing page to log in.

Design patterns used

The **strategy pattern** was used to allow for interchangeability and future extensibility of our system. For example, user login can be achieved via two means – either regular authentication or through Google OAuth.

The **observer pattern** was used to enable one-to-many relationships. An example of an implementation is in the DashboardPage, where each participant's summary card subscribes to the simplified debt strategy API. When there is an update to the debt strategy (e.g. when someone logs a new expense), the algorithm adjusts the debt strategy accordingly and updates the individual participants' summary cards.

The **factory pattern** can be found in the low-level implementation of displaying the menu items in the sidebar. Instead of hard-coding the menu items like "log transaction", "dashboard", or "trip info", a list of menu items and their corresponding redirections is stored as a list at MenuItems.js. When rendering the sidebar, the list is iterated through to display the relevant tabs.

The **façade pattern** was implemented to provide a simplified interface to a complex system. In the backend, when a user has to be created, multiple functions are called before the user's registration is approved. It has to (1) check if the username is taken, (2) check if the email has been used, and (3) if the password meets the security requirement.