NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

College of Computing and Data Science

SC2006 - Software Engineering

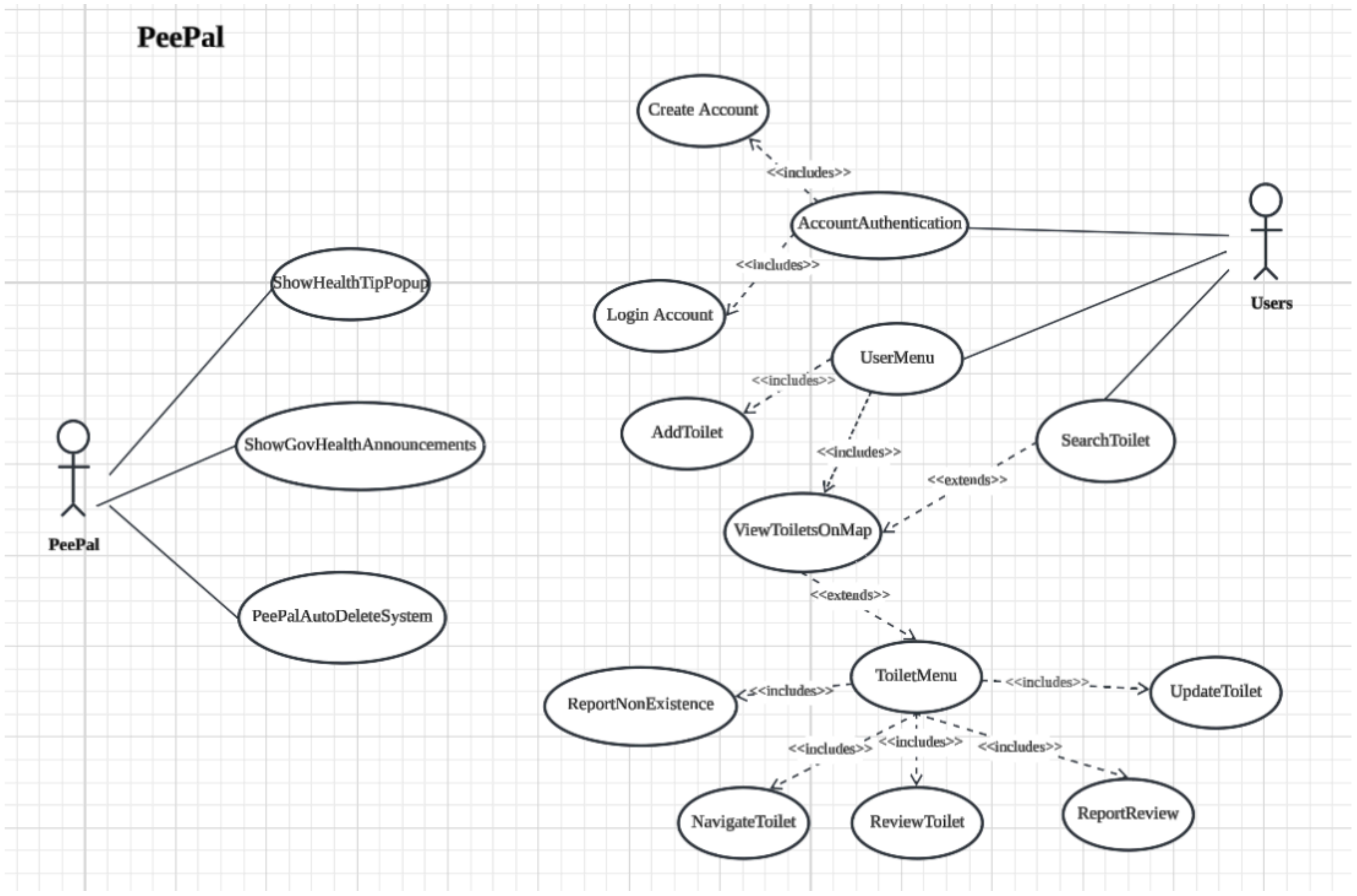
Lab 3 Deliverables

FDAB Team 1

Name	Matriculation No.
Adam Soh Shi Jie	U2320112A
Joyce Lee Jia Xuan	U2320463F
Soong Jun Shen	U2340460H
Tan Kai Hooi	U2420694C
Liew Jia Wei	U2320233G

1. Complete Use Case Model

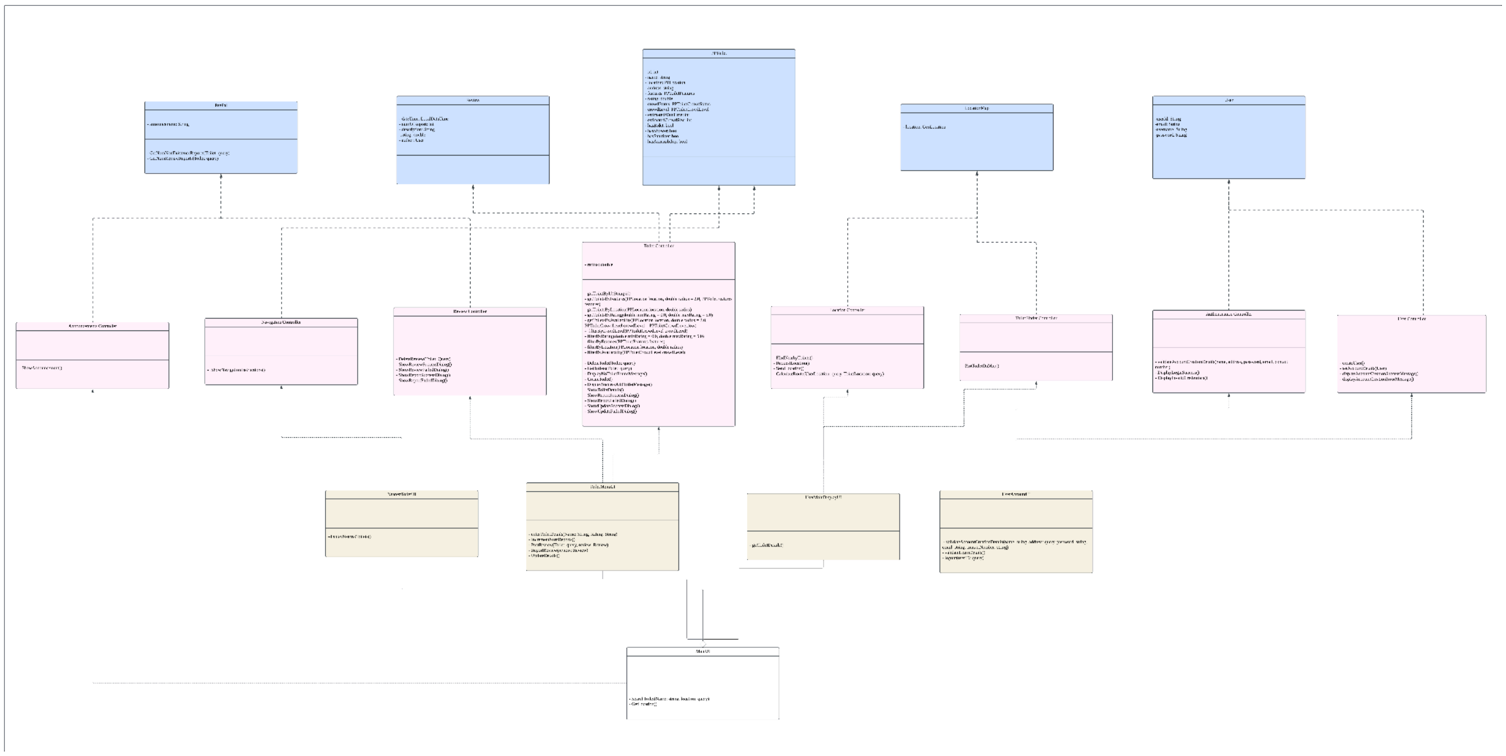
If the image is unclear, please refer to the raw PNG file that is uploaded together with this document.



2. Design Model

A. Class Diagram

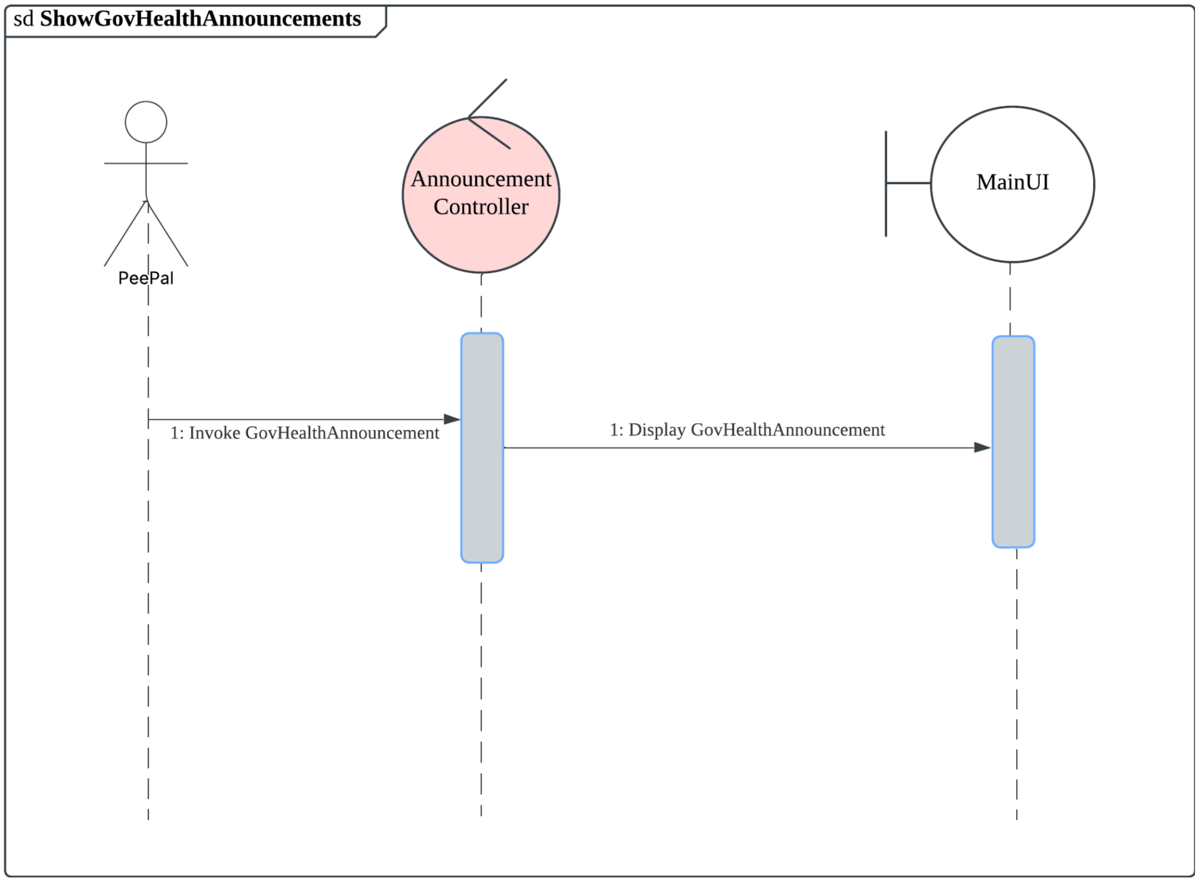
If the image is unclear, please refer to the raw PNG file that is uploaded together with this document.



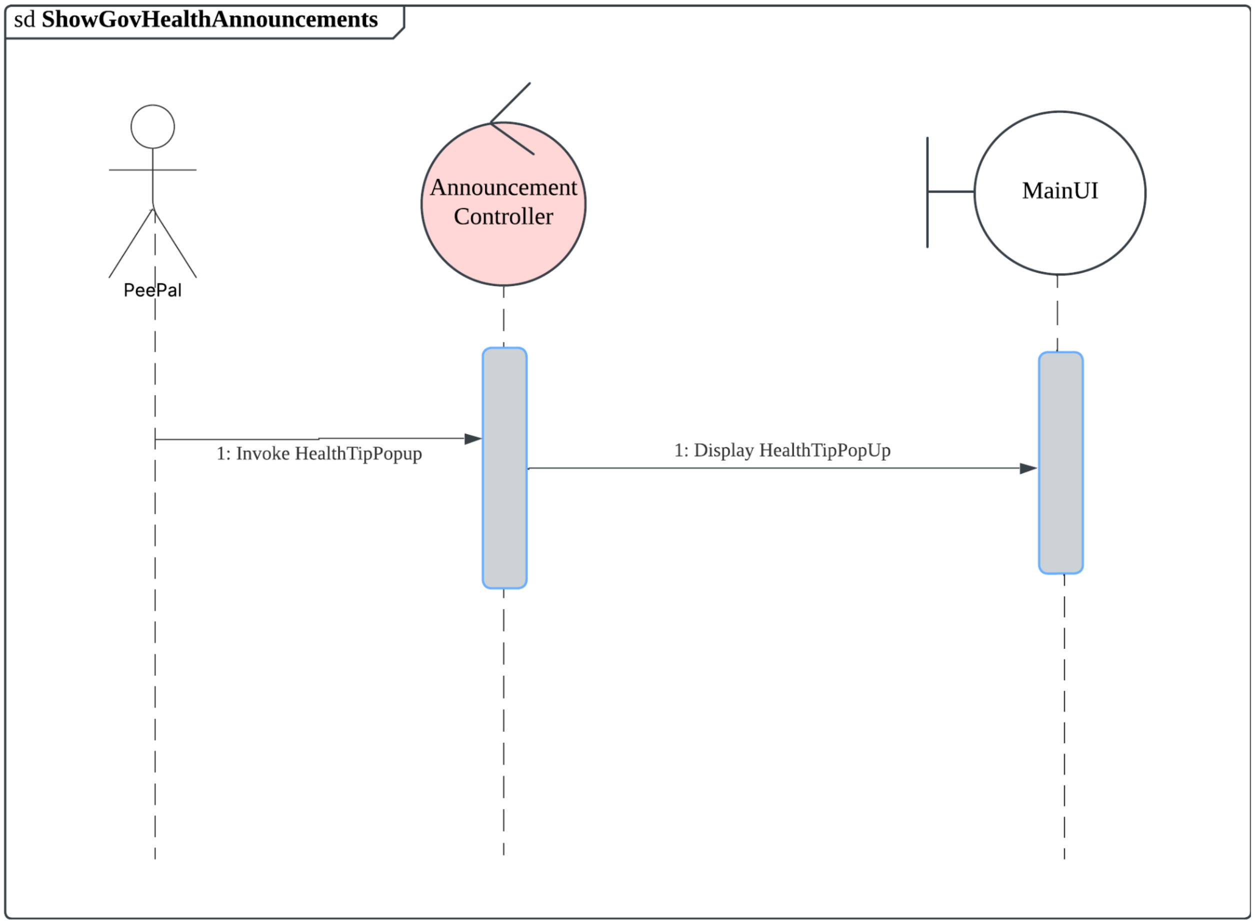
B. Sequence Diagrams

I. For Use Cases under I (PeePal)

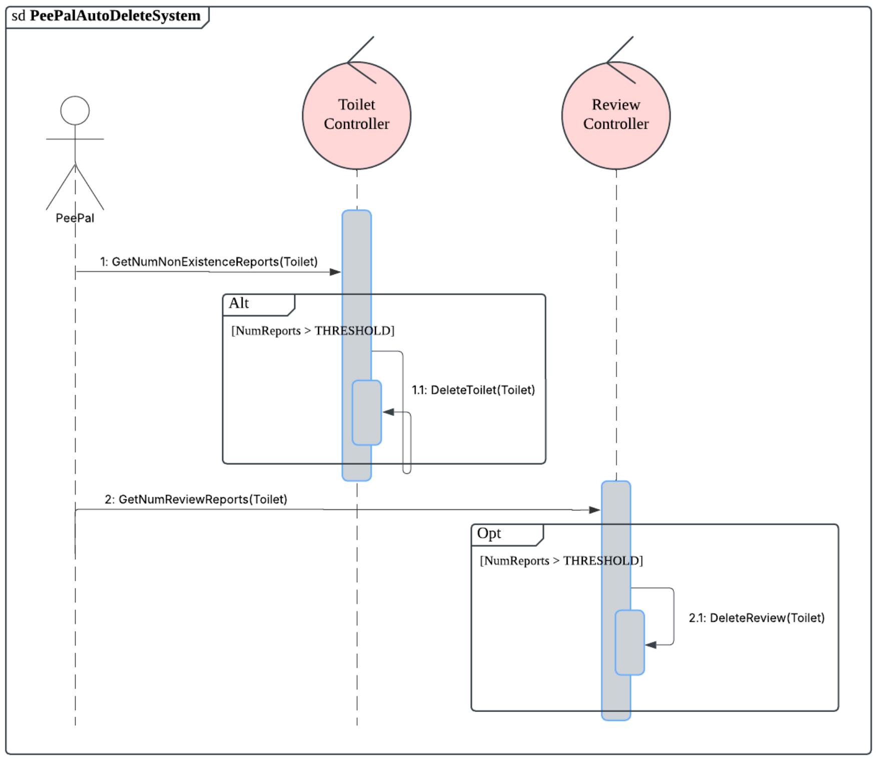
I.I ShowGovHealthAnnouncements


sd ShowGovHealthAnnouncements
PeePal
Announcement Controller
MainUI
1: Invoke GovHealthAnnouncement
1: Display GovHealthAnnouncement

I.II ShowHealthTipPopup

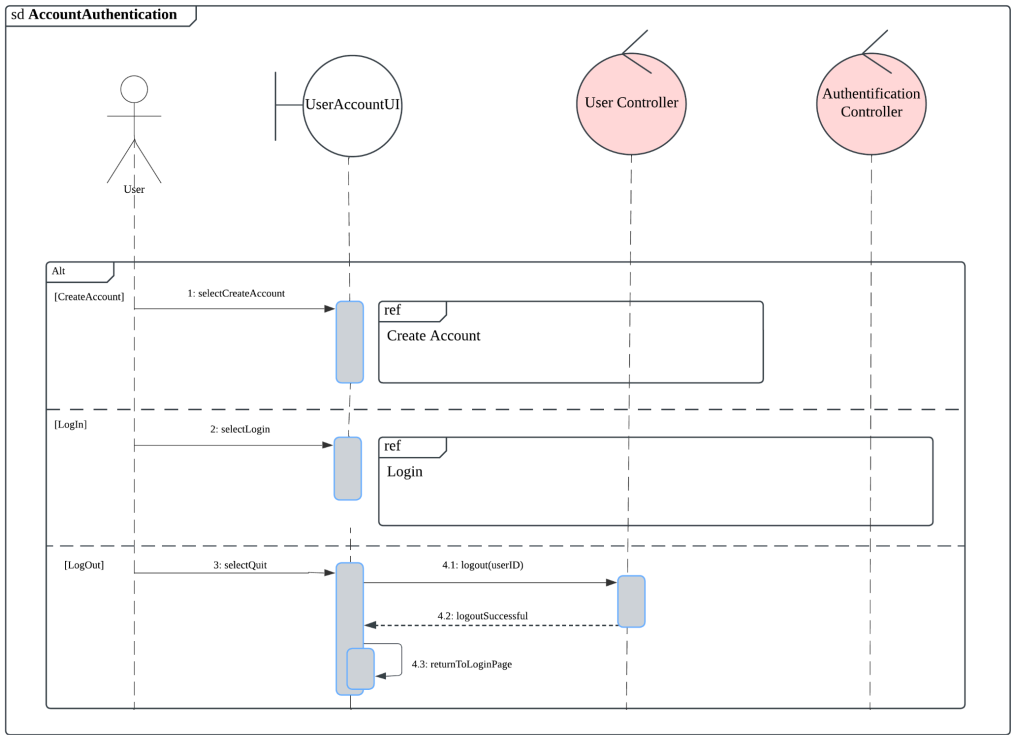


I.III PeePalAutoDeleteSystem

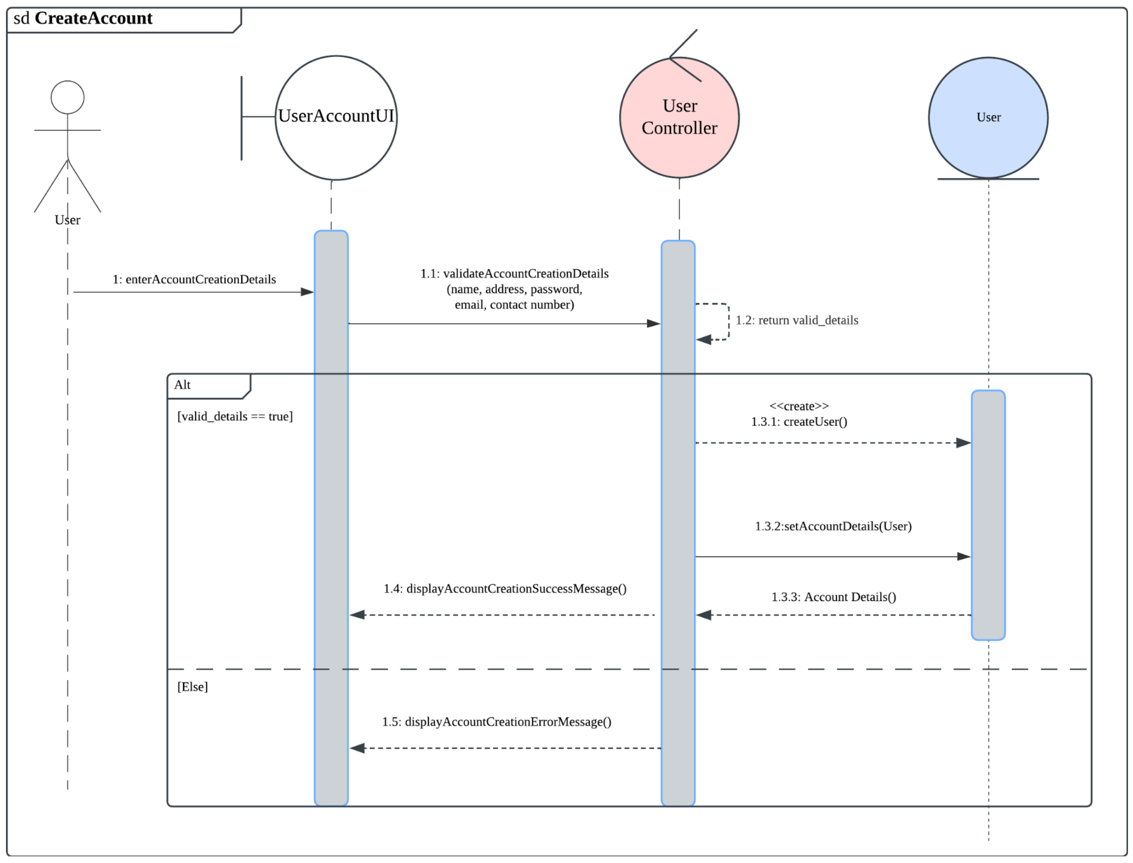

sd PeePalAutoDeleteSystem
PeePal
Toilet Controller
Review Controller
1: GetNumNonExistenceReports(Toilet)
Alt
[NumReports > THRESHOLD]
1.1: DeleteToilet(Toilet)
2: GetNumReviewReports(Toilet)
Opt
[NumReports > THRESHOLD]
2.1: DeleteReview(Toilet)

II. For Use Cases under II (User Authentication)

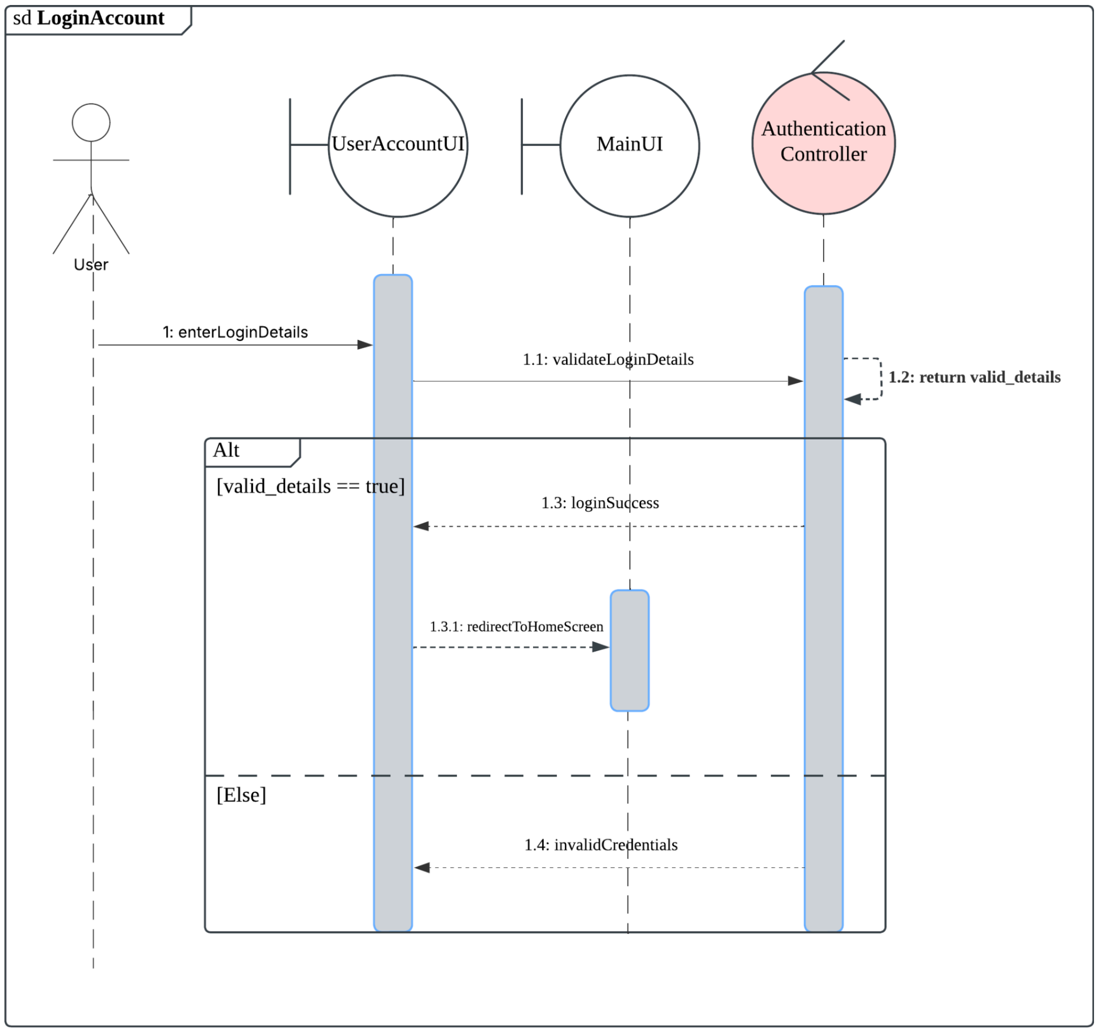
II.I AccountAuthentication



II.II CreateAccount

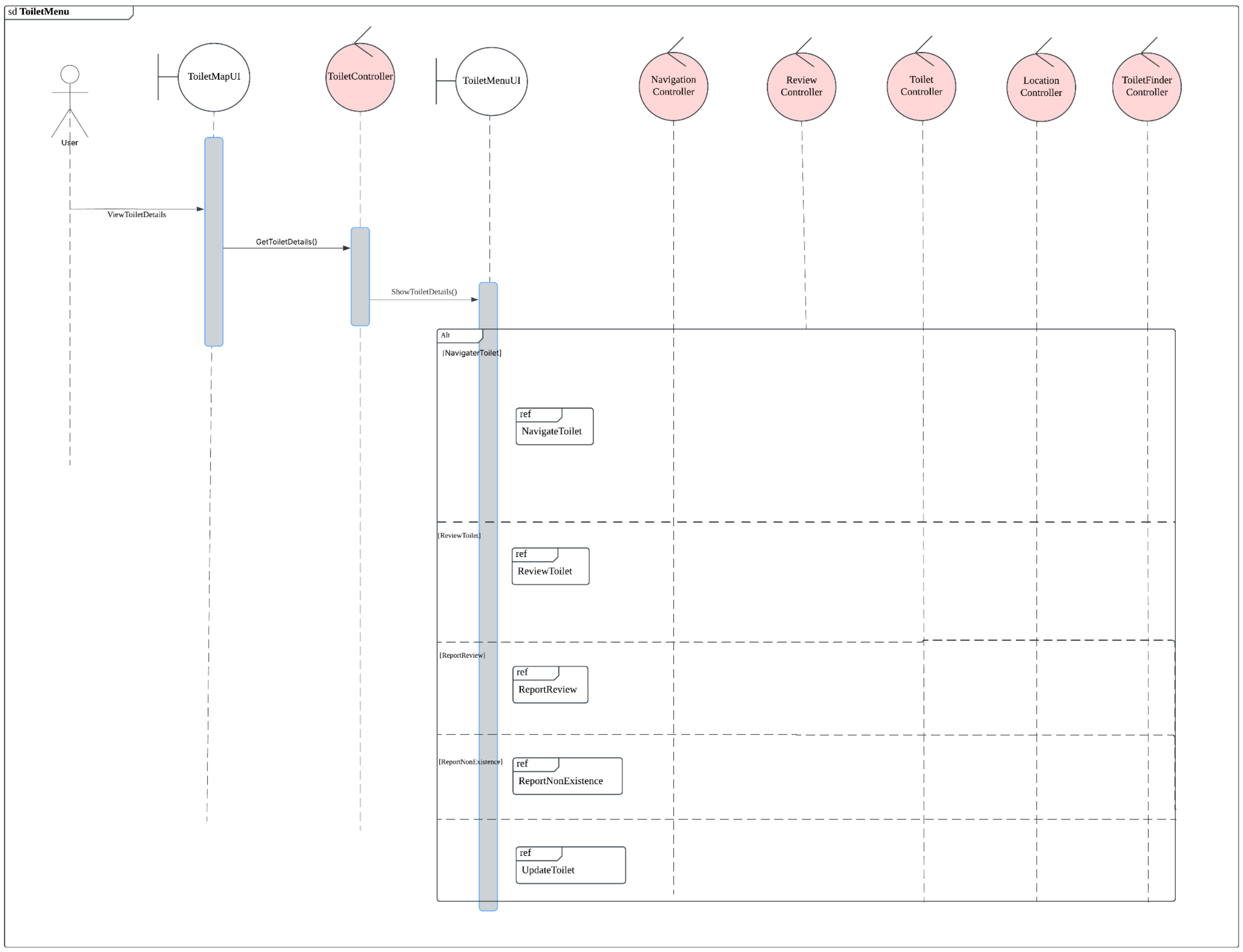

sd CreateAccount
User
UserAccountUI
User Controller
User
1: enterAccountCreationDetails
1.1: validateAccountCreationDetails (name, address, password, email, contact number)
1.2: return valid_details
Alt
[valid_details == true]
<<create>> 1.3.1: createUser()
1.3.2:setAccountDetails(User)
1.4: displayAccountCreationSuccessMessage()
1.3.3: Account Details()
[Else]
1.5: displayAccountCreationErrorMessage()

II.III LoginAccount

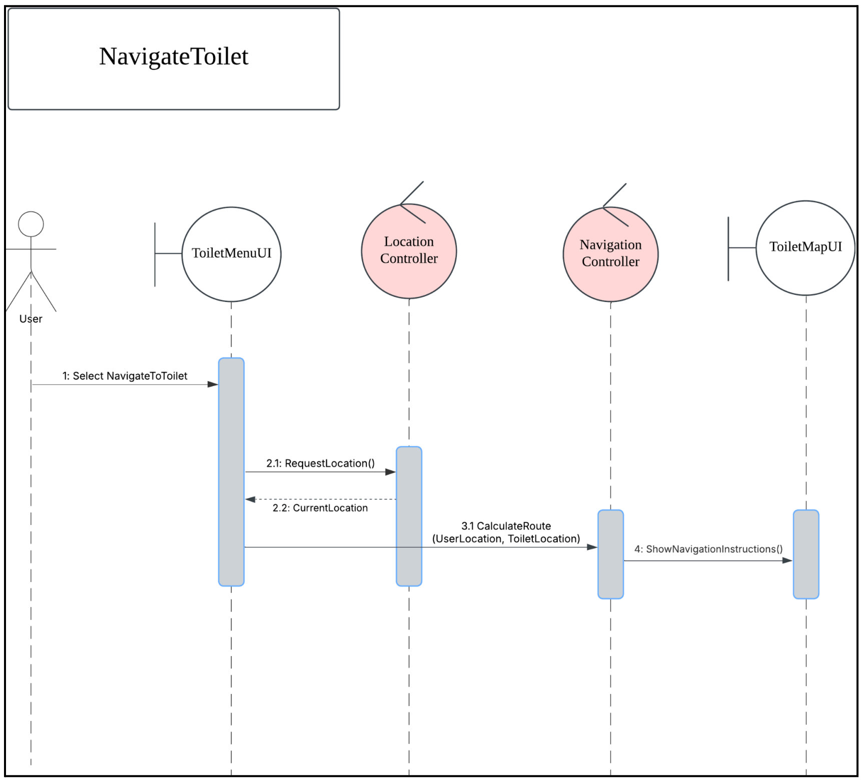


III. For Use Cases under III (ToiletFeatures)

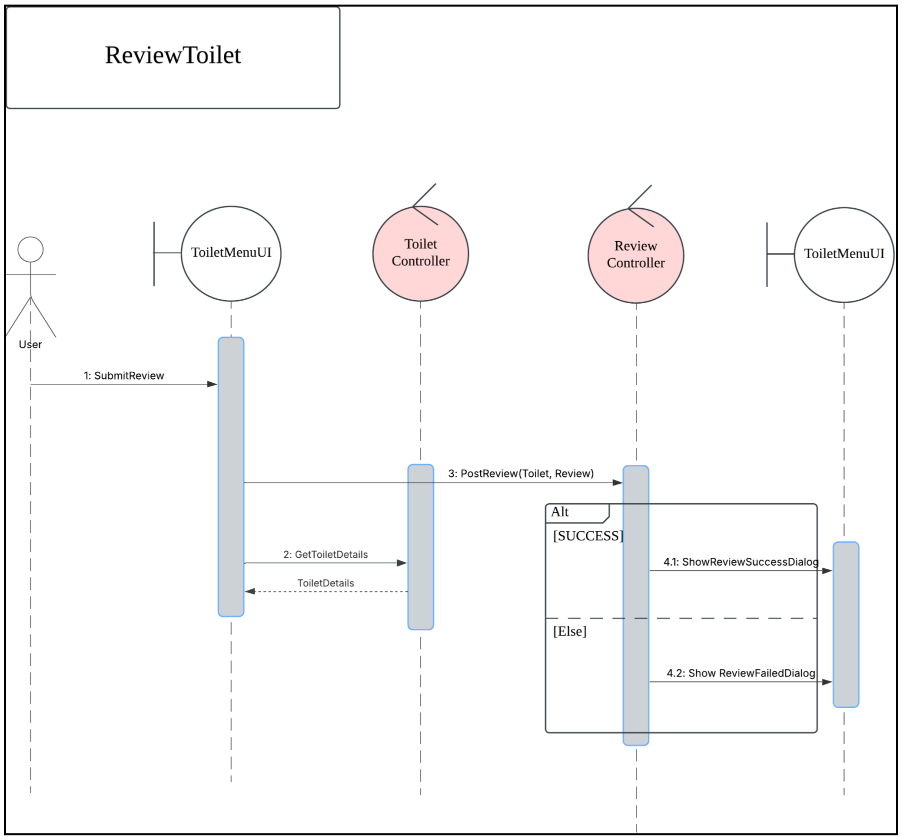
III.I ToiletMenu



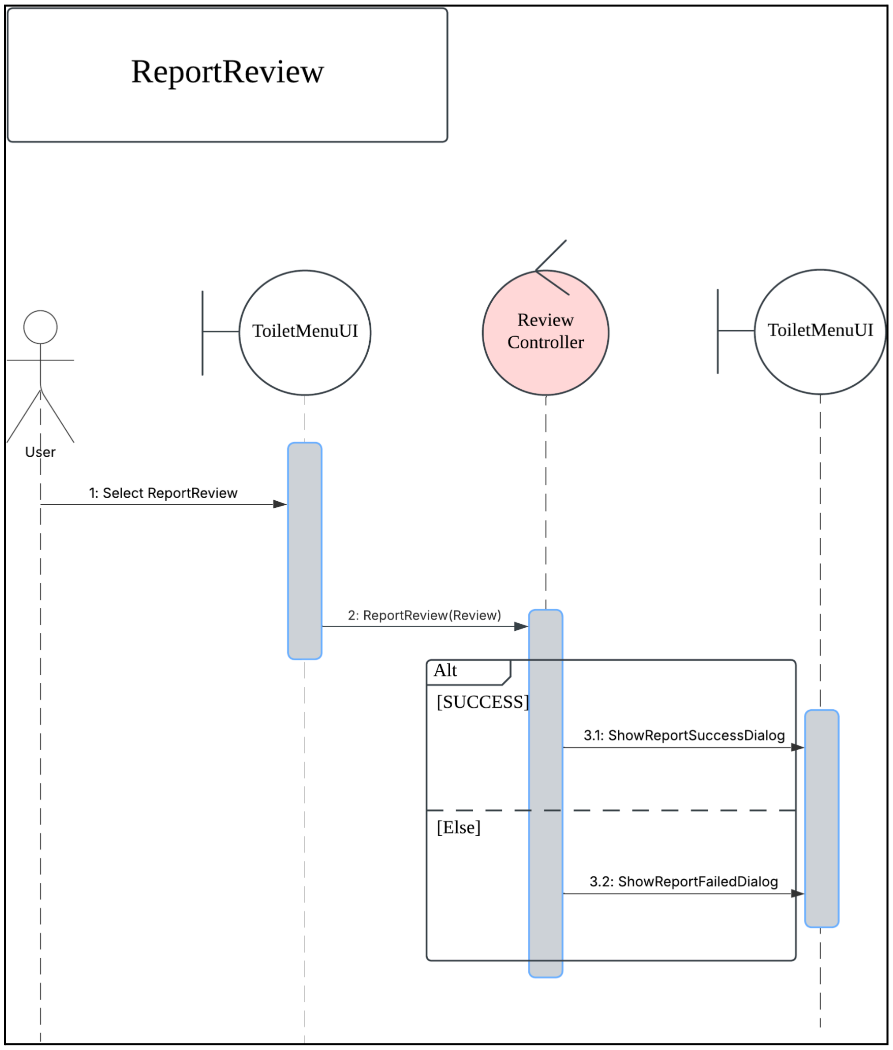
III.II NavigateToilet



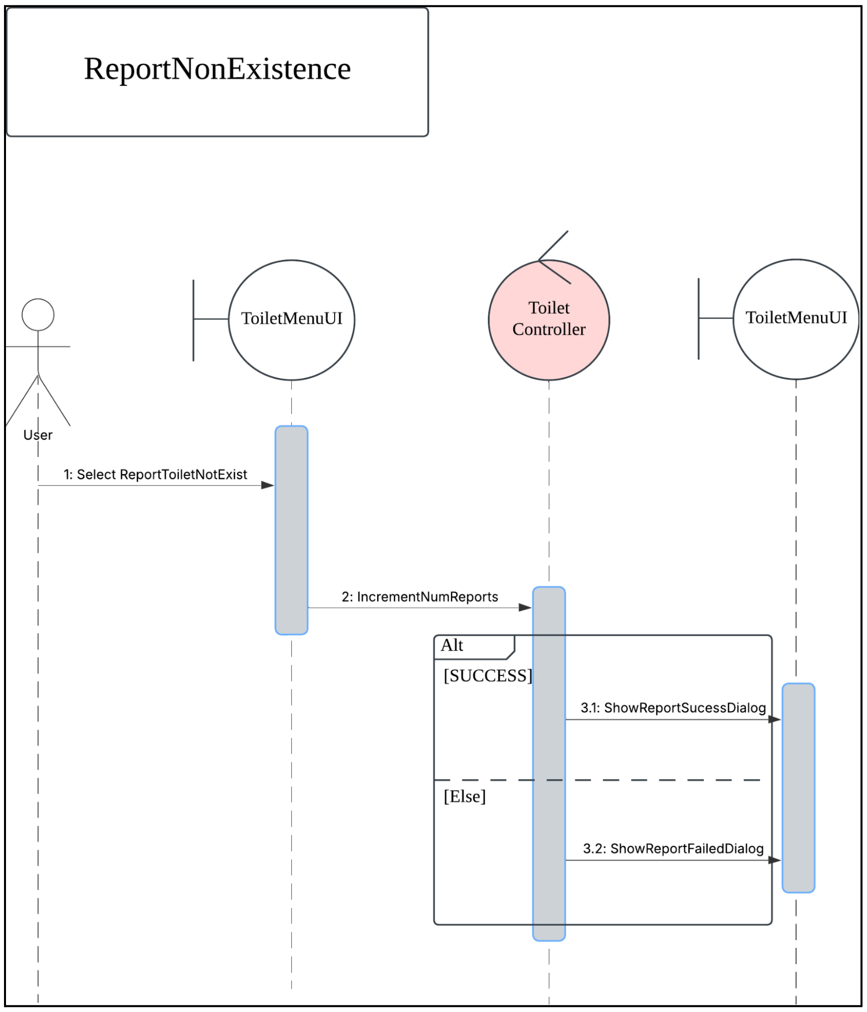
III.III ReviewToilet



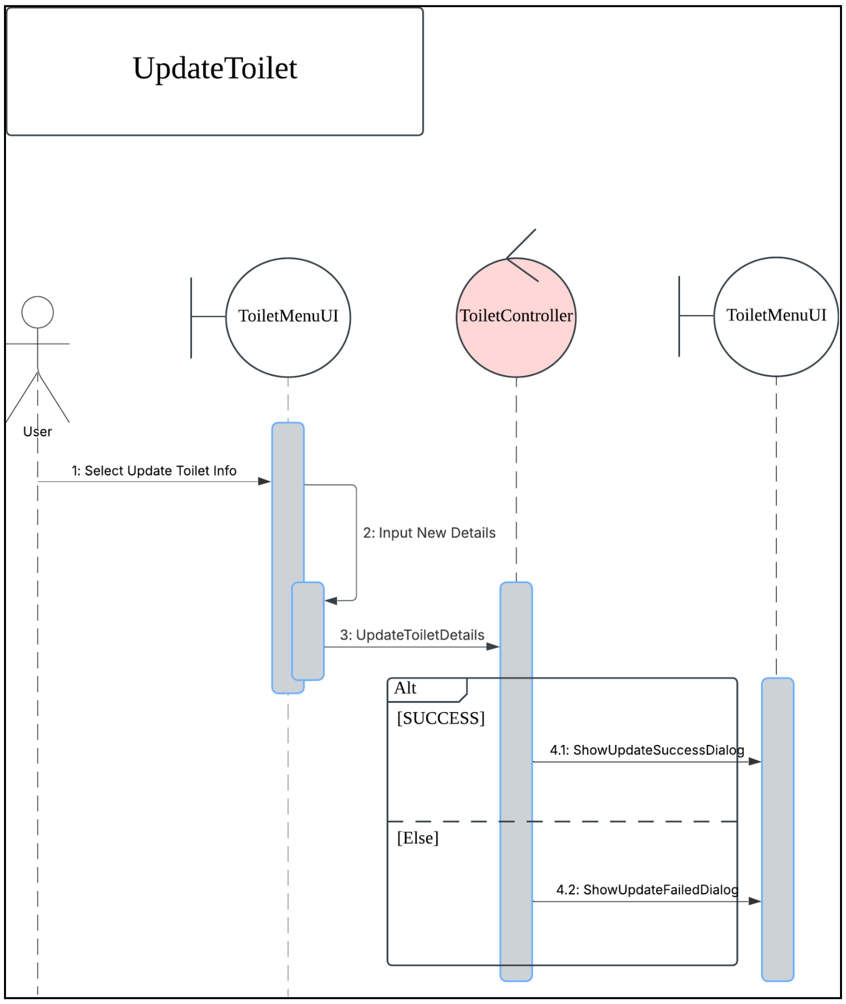
III.IV ReportReview



III.V ReportNonExistence

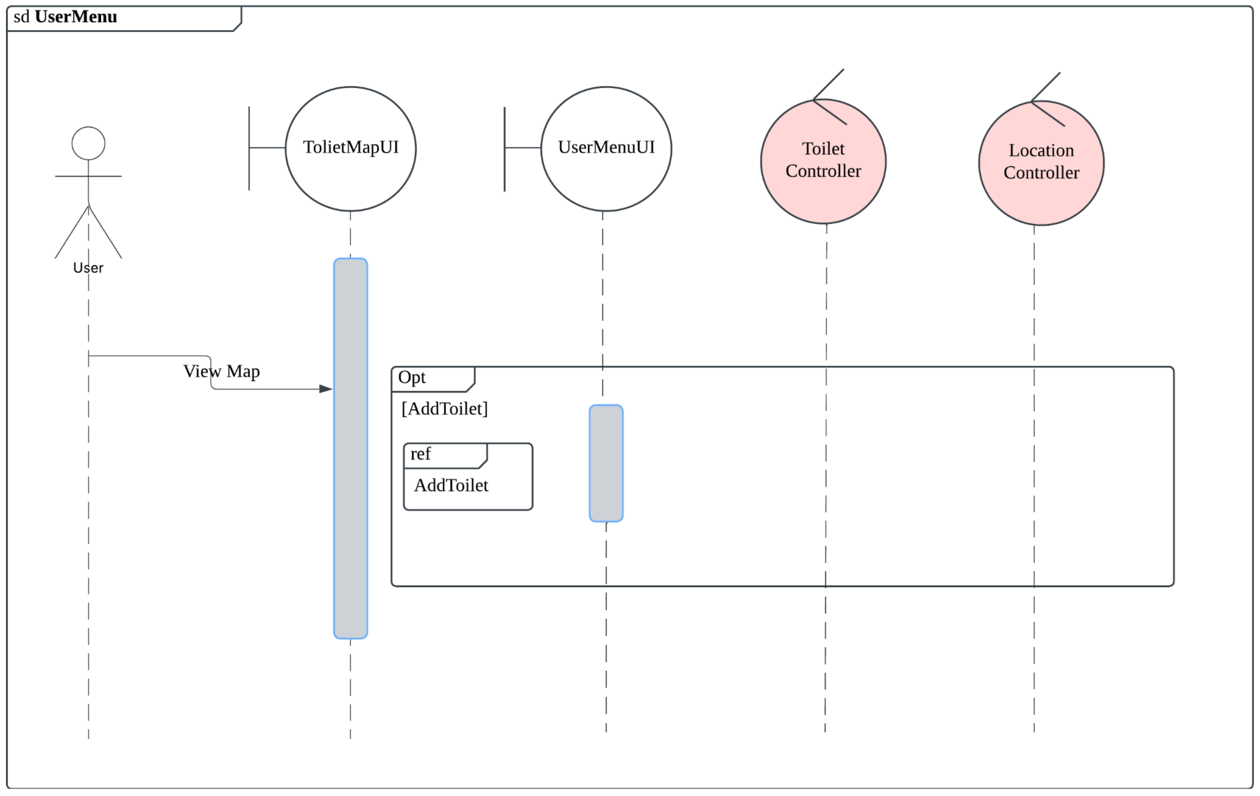


III.VI UpdateToilet

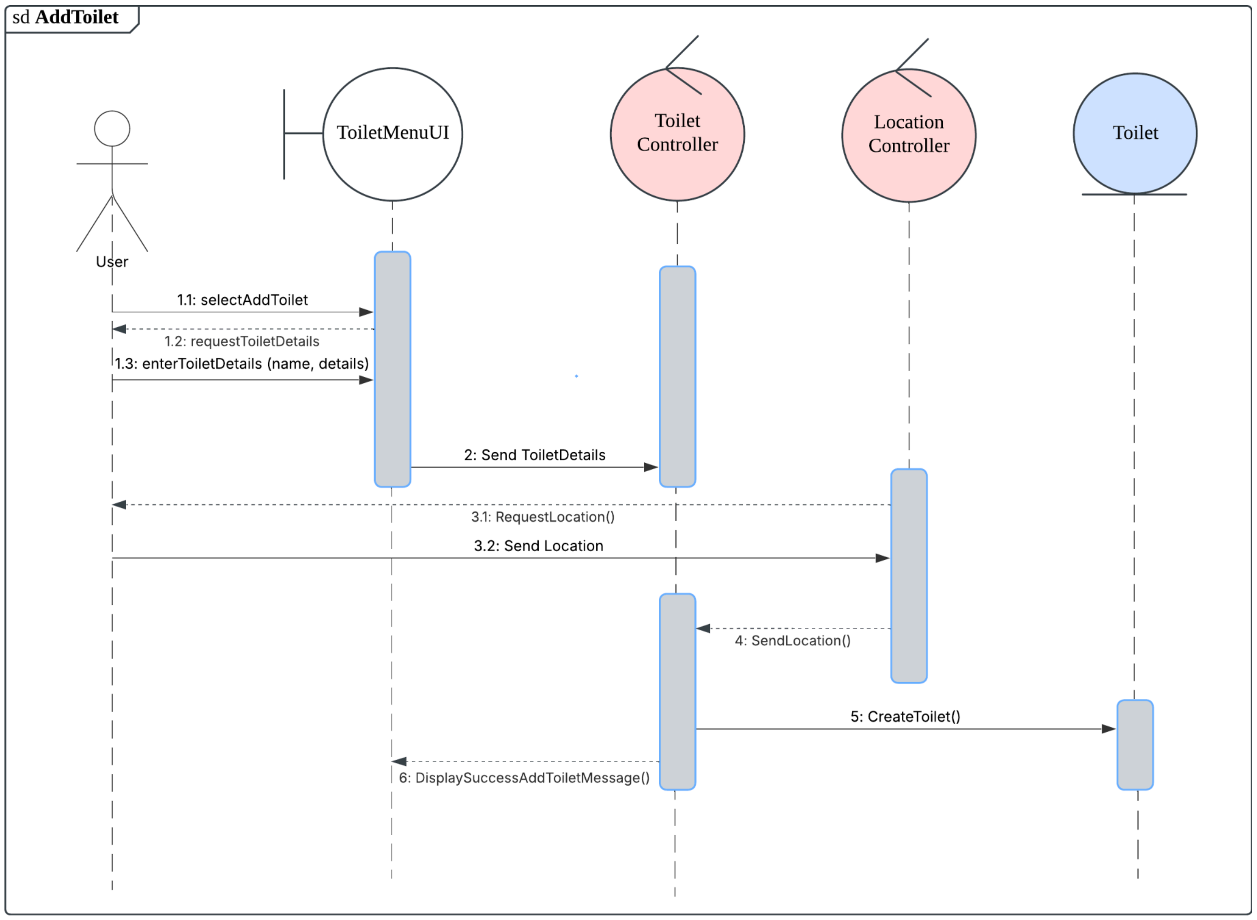


IV. For Use Cases under IV (Users)

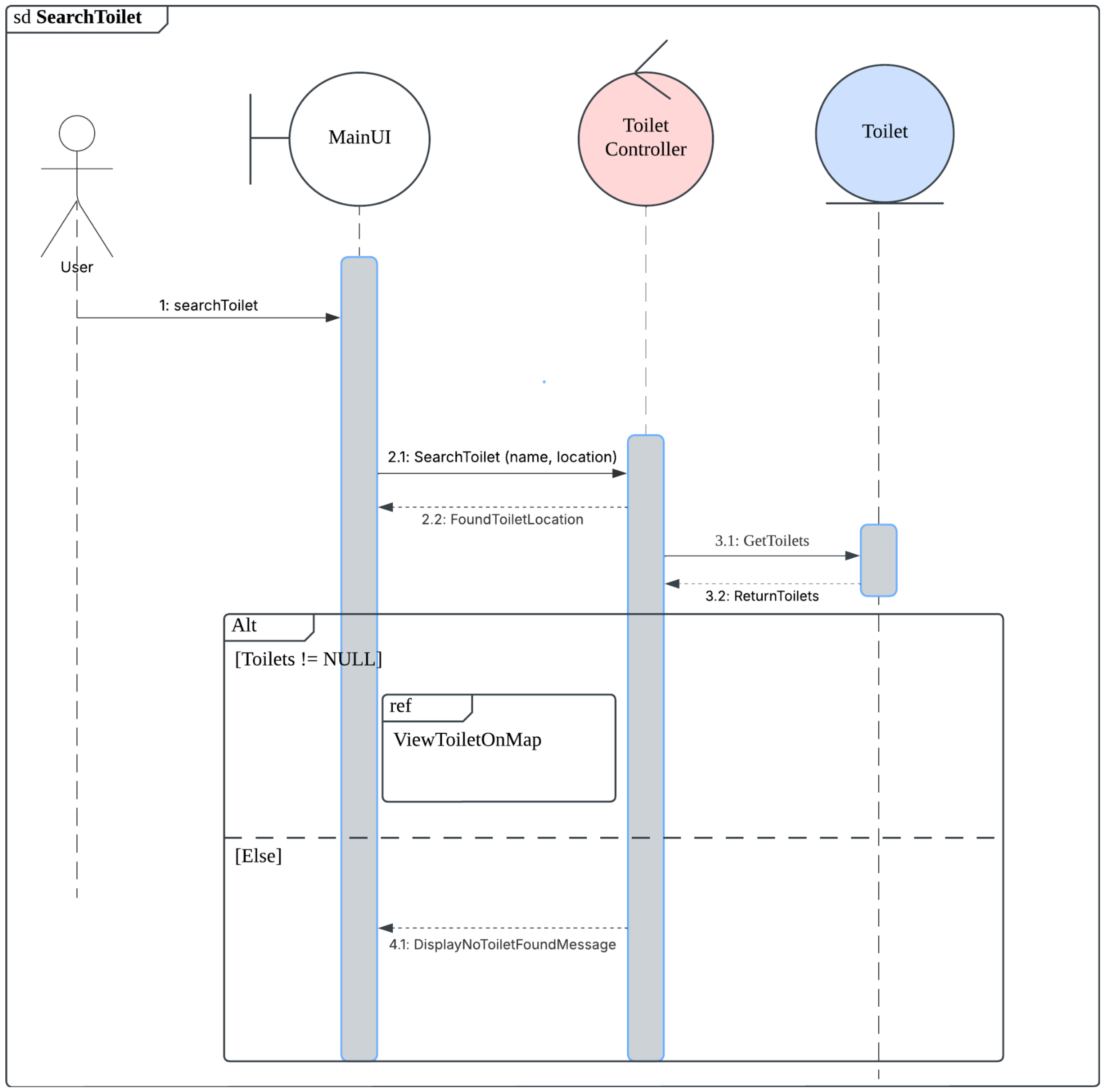
IV.I UserMenu



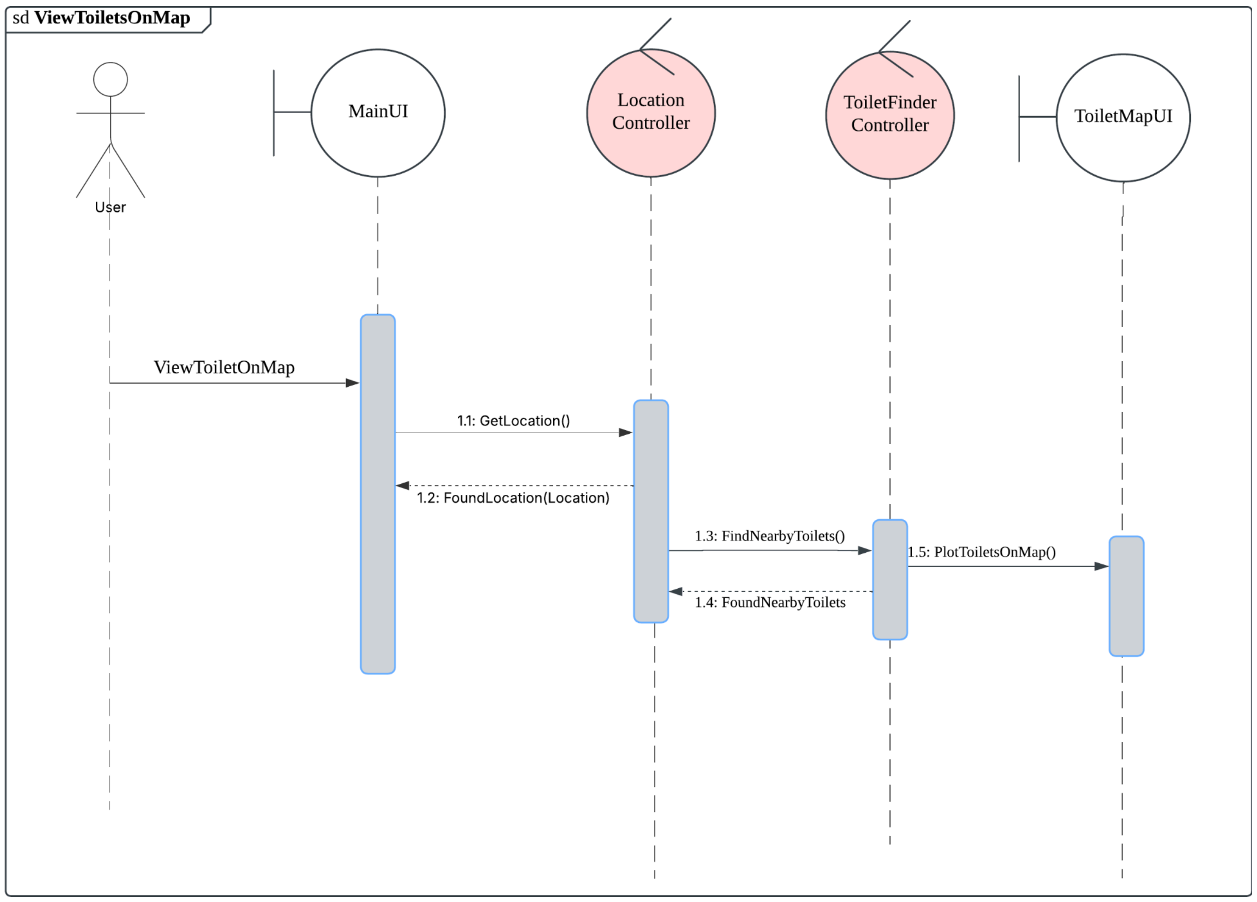
IV.II AddToilet



IV.III SearchToilet

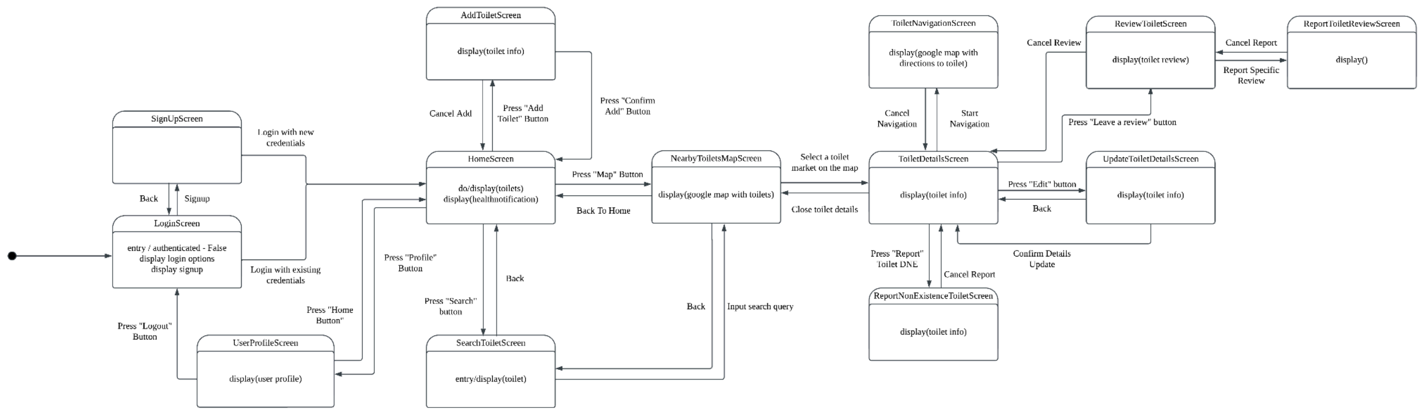


IV.IV ViewToiletOnMap



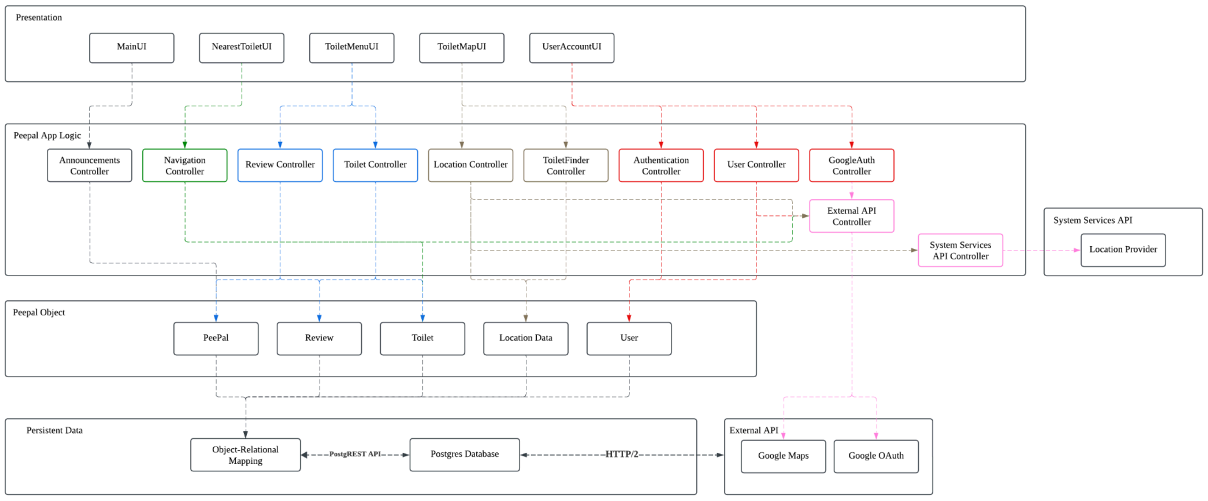
C. Dialog Map Diagram

If the image is unclear, please refer to the raw png file that is uploaded together with this document.



3. System Architecture

If the image is unclear, please refer to the raw .png file that is uploaded together with this document.



Presentation Layer

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

The layer consists of:

1. **MainUI**
   MainUI will consist of the services of the Announcement Controller.
2. **NearestToiletUI**
   NearbyToiletUI is part of the user interface in MainUI, and it allows Users to interact by calling NavigationController.
3. **ToiletMenuUI**
   ToiletMenuUI is part of the user interface in MainUI, and it allows the User to interact by calling Review Controller and Toilet Controller.
4. **ToiletMapUI**
   ToiletMapUI is part of the user interface in MainUI, and it allows the Users to see the user's location.
5. **UserAccountUI**
   UserAccountUI is part of the login page in MainUI, and it allows the user to log in via the Authentication Controller or sign up via the User Controller.

App Logic Layer

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

This layer consists of:

1. **AnnouncementController**
   AnnouncementController is called by MainUI to display announcements to Users.
2. **NavigationController**
   Display the routes to the selected toilet using Google Maps.
3. **ReviewController**
   Delete reviews that have been flagged too many times automatically.
4. **ToiletController**
   Fetch toilet information from the database.
   Manage user-submitted data about the toilet, e.g. reviews or if they wish to update anything.
   Connected to ToiletUI to help display toilet information.
5. **LocationController**
   Fetch user location via system services API controller.
6. **ToiletFinderController**
   Use location data from the location controller to determine the nearest toilets.
   Fetch and display search results from a user's specific query.
7. **AuthenticationController**
   AuthenticationController is called by UserAccountUI for users to log in. This might include various login methods, including Google Authentication.
8. **UserController**
   UserController is called by UserAccountUI for users to sign up. This might include various signup methods, including Google signup.
9. **GoogleAuthController**
   Allow users to log in via Google Authentication by using the service of GoogleAuthController**.**
10. **ExternalAPIController**
    Handle communication with third-party APIs:
    - Fetch location data from Google Maps.
    - Manage authentication via Google OAuth for user login.
11. **SystemServicesAPIController**
    Fetch location data from phone's internal GPS.

App Object Layer

This layer contains the entity classes that will be called by the App Logic Layer to implement its logic. The entity classes are stored in the database of the persistent layer.

This layer consists of:

1. **PeePal**
   System contains announcement information, number of non-existent toilet reports.
2. **Review**
   Review contains ratings, text, photos, and the number of reports.
3. **Toilet**
   Toilet contains all information regarding the toilet, location, toilet features, and toilet rating.
4. **LocationMap**
   LocationMap contains all information regarding the user's location, including current location and objects of nearby toilets.
5. **User**
   User contains all information regarding the user. These include name, email, and last login.

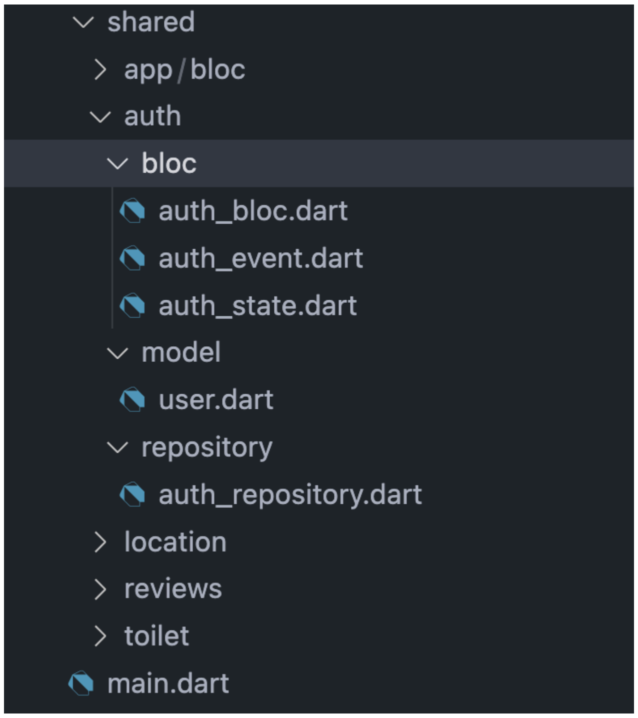
Persistent Data Layer

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

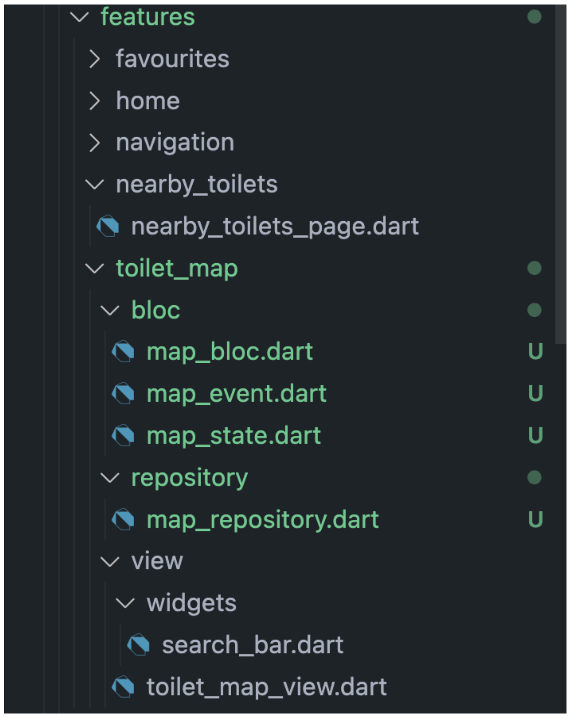
4. Application Skeleton

Please refer to the source code uploaded in the github repository for the application skeleton.

A. Frontend



```
shared
  app / bloc
  auth
    bloc
      auth_bloc.dart
      auth_event.dart
      auth_state.dart
    model
      user.dart
    repository
      auth_repository.dart
  location
  reviews
  toilet
main.dart
```



```
features
  favourites
  home
  navigation
  nearby_toilets
    nearby_toilets_page.dart
  toilet_map
    bloc
      map_bloc.dart          U
      map_event.dart         U
      map_state.dart         U
    repository
      map_repository.dart    U
    view
      widgets
        search_bar.dart
      toilet_map_view.dart
```

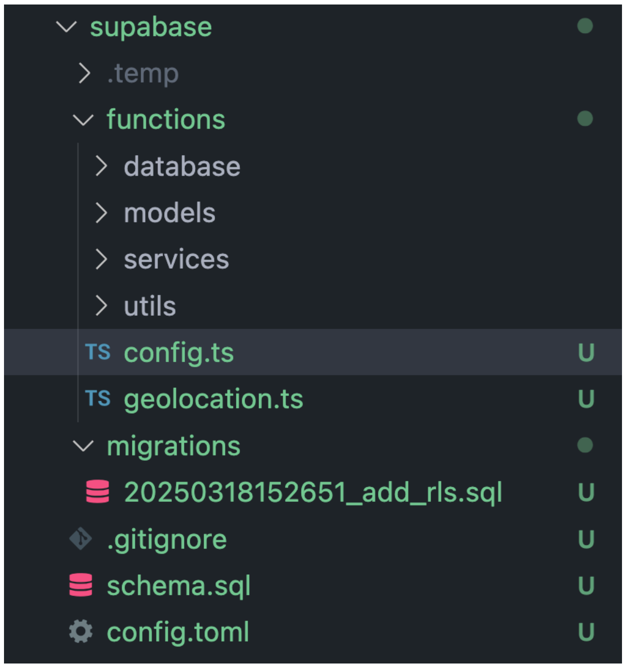
Our frontend follows the Clean Architecture principles, emphasizing a clear separation of concerns. The codebase is structured using a domain-first approach, where each domain folder contains three distinct layers:

- **UI Layer**: Responsible for presentation, with components that listen to and react to state changes.
- **Controller/Logic Layer**: Managed using the Business Logic Component (BLoC) pattern, ensuring that business logic remains separate from the UI.
- **Data Access Layer**: Abstracted through the Repository pattern, providing a clean interface for data retrieval and storage.

Data flows via the following dependency chain, from the repository to the BLoC (controller) to the view. For shared components, the UI layer is replaced with the **Model layer,** which encapsulates the data models used by the application, and handles serialisation logic for network requests to the backend.

This architecture ensures maintainability, scalability, and a clear distinction between concerns, enhancing the overall development workflow.

B. Backend



```
supabase
  .temp
  functions
    database
    models
    services
    utils
    config.ts                       U
    geolocation.ts                  U
  migrations
    20250318152651_add_rls.sql      U
  .gitignore                        U
  schema.sql                        U
  config.toml                       U
```

The backend consists of several components:

- **Database**: Contains logic and types for the object-relational-mapping between database objects and TypeScript objects.
- **Models**: Encapsulates the data models used by the application, and handles serialisation logic for network requests.
- **Services:** Contains the logic to compute and modify the business objects (models), and fetching data from external APIs where needed.
- **Utils:** Contains utility methods used through the backend, such as math functions.
- **Migrations:** Contains the database migrations used by the Postgres database. The migrations refer to the incremental changes to the database schema via SQL statements.
- **Endpoints:** The root of the functions folder consists of the actual RESTful endpoints that can be invoked by the client application, and the mapping to corresponding business logic (eg: geolocation.ts).

5. Appendix

Key Design Issues

A. **Identifying and Storing Persistent Data (Section 7.4.2)**
   - Relational database
     i. User
        - **userId** (str), email (str), username (str), password (str)
     ii. Toilet
        - **toiletId** (str), name (str), location (GeoLocation), amenities (List), numOfNonExistenceToiletReport (int), review (Review[])
     iii. Review
        - **dateTime** (localDateTime), numOfReports (int), description (string), author (User)
     iv. PeePal
        - **AnnouncementID** (int), Announcement (str)
     v. LocationData
        - **LocationID** (int), longitude (int), latitude (int)

Tech Stack

Frontend

- Flutter (Target iOS and Web)
- State Management: Business Logic Component (BLoC) with ReactiveX Streams
- Architecture: Separation of concerns, View depends on BLoCs, BLoCs manage state act as the business logic layers and depend on repositories for data, repositories act as the data access layer.

Backend

- Backend as a service (BaaS).
- Serverless functions written TypeScript via Deno runtime, as middleware to access external APIs and handle server side compute.
- BaaS handles authentication and issues JSON Web Tokens (JWTs) to clients for authorisation.

Database

- Postgres with PostGIS to handle location-based queries.
- pl/pgSQL: Complex queries using stored procedures.
- Data access control is handled via Postgres R ow Level Security Rules (RLS).