React To Spring

Forum
Software Architecture Document

Version 2.0

Revision History

Date	Version	Description	Author
2/12/2024	1.0	First version of Software Architecture Document: Defines the foundational software architecture for the project	Team
26/12/2024	2.0	Revised version of the first one Changes: 1. Add packages diagram for Logical view including controller, service, model diagram. (section 2 : Logical View) 2. Add Deployment and Implementation View (section 5, 6: Deployment and Implementation)	Team

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document is to define the software architecture for the Forum project, inspired by Voz.The architecture addresses functional requirements, such as user management and discussion threads, and non-functional requirements like security and reliability.
Link to diagrams:
https://drive.google.com/file/d/14j9eX_JNr7VoIL7XxQL4tJnLKLIxY2Sv/view?fbclid=IwY2xjawGRKqxleHRuA2FlbQIxMAABHSsjczPGF-jHlAwFD6d1-sceT3bp10dlmp_CHHF9Z7-YvxtHYi3ksIVUUA_aem_x7BgsrRlsE724mnmID2WHA

1.2 Scope

This document provides an overview of the architecture for an online forum system. The forum system is developed by the React To Spring team as part of the Introduction to Software Engineering course. Readers will learn about the technologies, frameworks, and tools used in designing the system, including its core functionalities such as user management, discussion thread creation and management and notification management.

1.3 Definitions, Acronyms and Abbreviations

1.4 References

Applicable references are:
• https://www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch.htm?fbclid=IwAR2M5pRqlMp1o4HH_c0d47MylS6nbyV39b8h11BoHZyI8qaahIkDrQYoOmM
• https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/
• https://sudonull.com/post/79895-Honest-MVC-on-React-Redux-Developer-Soft-Blog

1.5 Overview

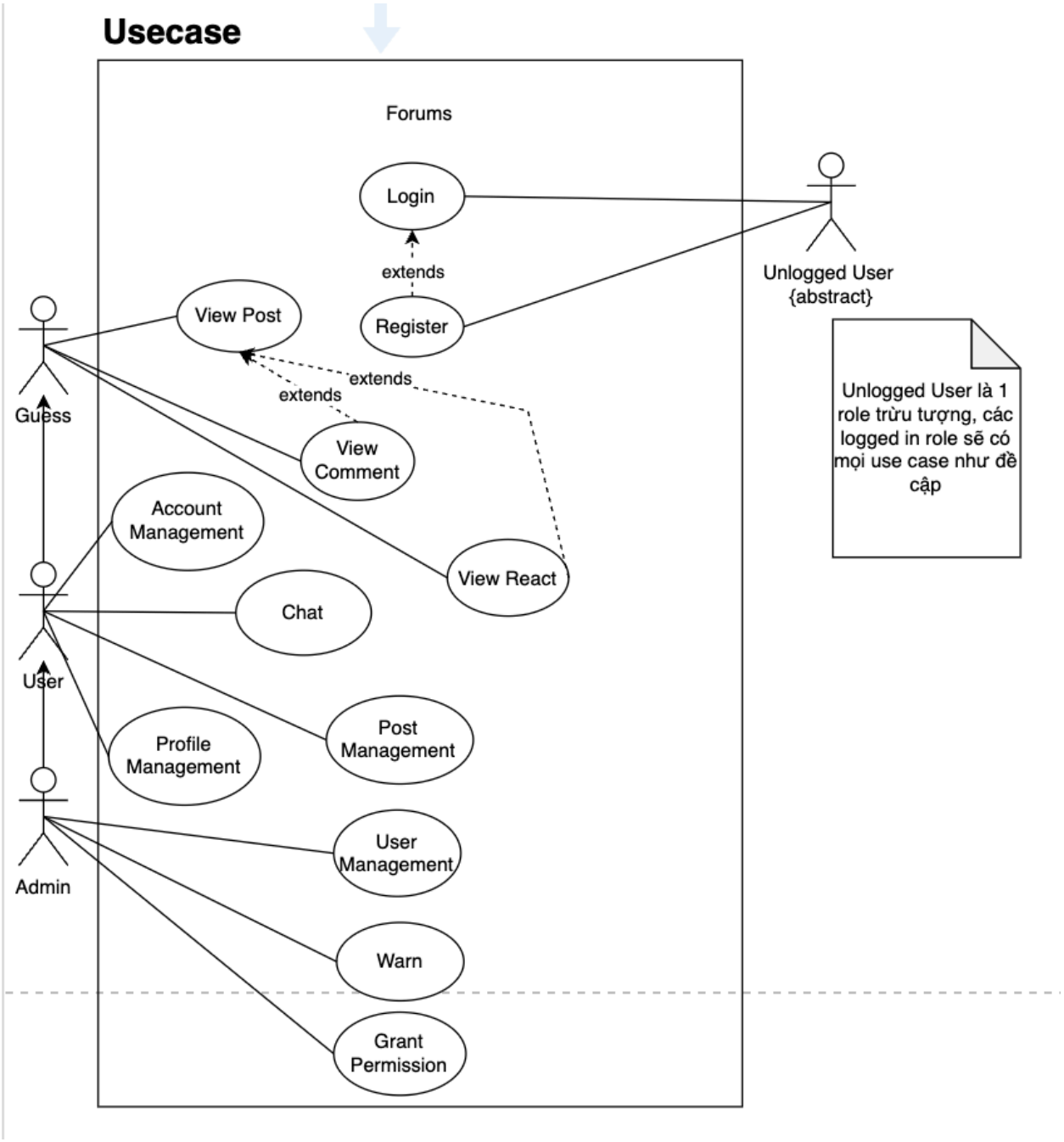
This document provides an overview of the forum system's architecture, starting with design principles and technical requirements. It details the system's components, including the backend API, and database structure. The architecture follows the layered design pattern, separating concerns for better maintainability. Each section builds upon the previous one to give a holistic view of the system.

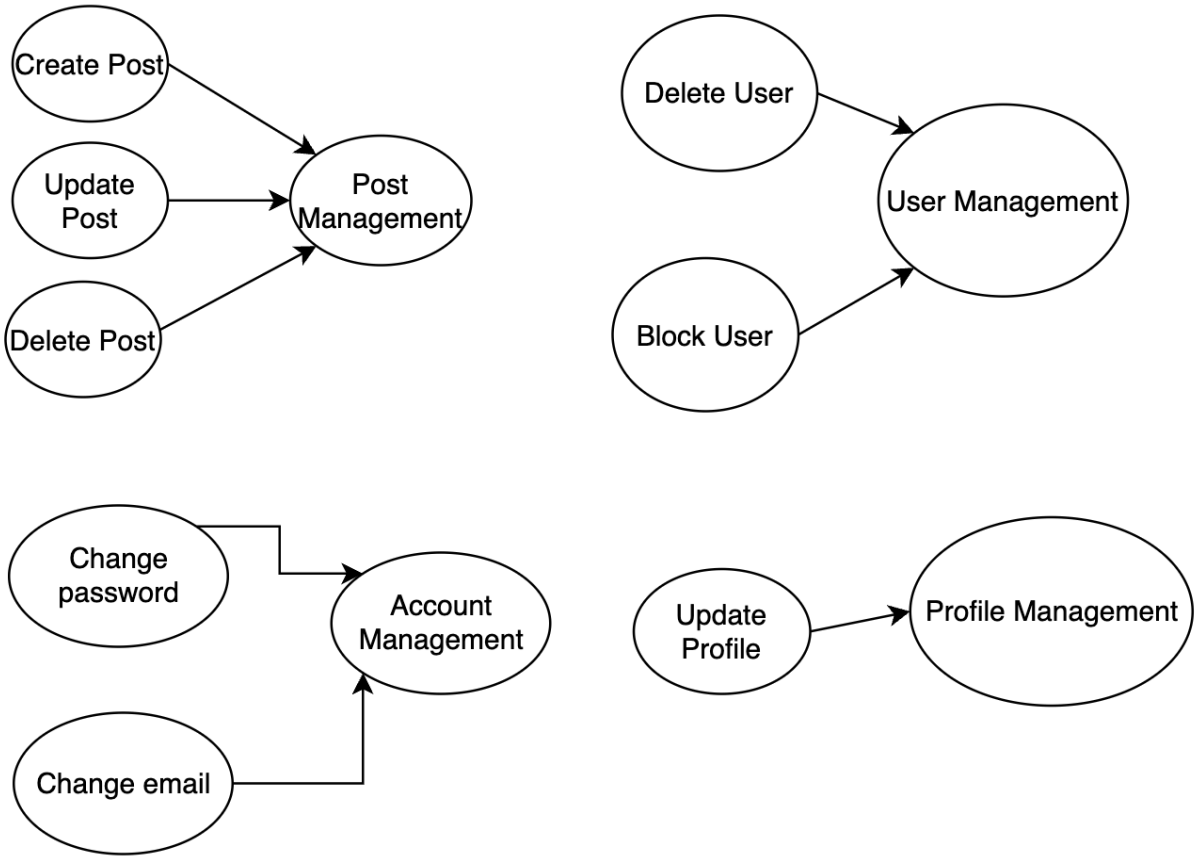
2. Architectural Goals and Constraints

- Security and Privacy: The system must ensure strong protection of user data and personal information. All sensitive data, including user credentials, should be encrypted. The system must have robust security mechanisms, such as secure authentication and authorization, to prevent unauthorized access and attacks.
- Performance: The system must provide fast and responsive performance, especially during peak usage. This includes efficient database queries, low latency, and optimized code that ensures a smooth user experience.
- User-Friendliness: The user interface must be intuitive, modern, and easy to navigate.
- Use of Off-the-Shelf Products: The system will utilize established technologies such as React Library, MongoDB, ReactJs and RESTful APIs, ensuring reliability and industry-standard practices for the development process.
- Client-Server Model: The system will follow a client-server architecture, where the client interacts with the server over the internet. This constraint ensures centralized management, easier maintenance, and better control over the data.
- Team Structure and Collaboration: The project will be developed by a fixed team of 5 members who will collaborate closely throughout the development process.
- Schedule: The architecture must allow for timely delivery of the project, with phases aligned to meet deadlines for each milestone. The system should be modular, enabling parallel development by different team members while ensuring integration at key stages.

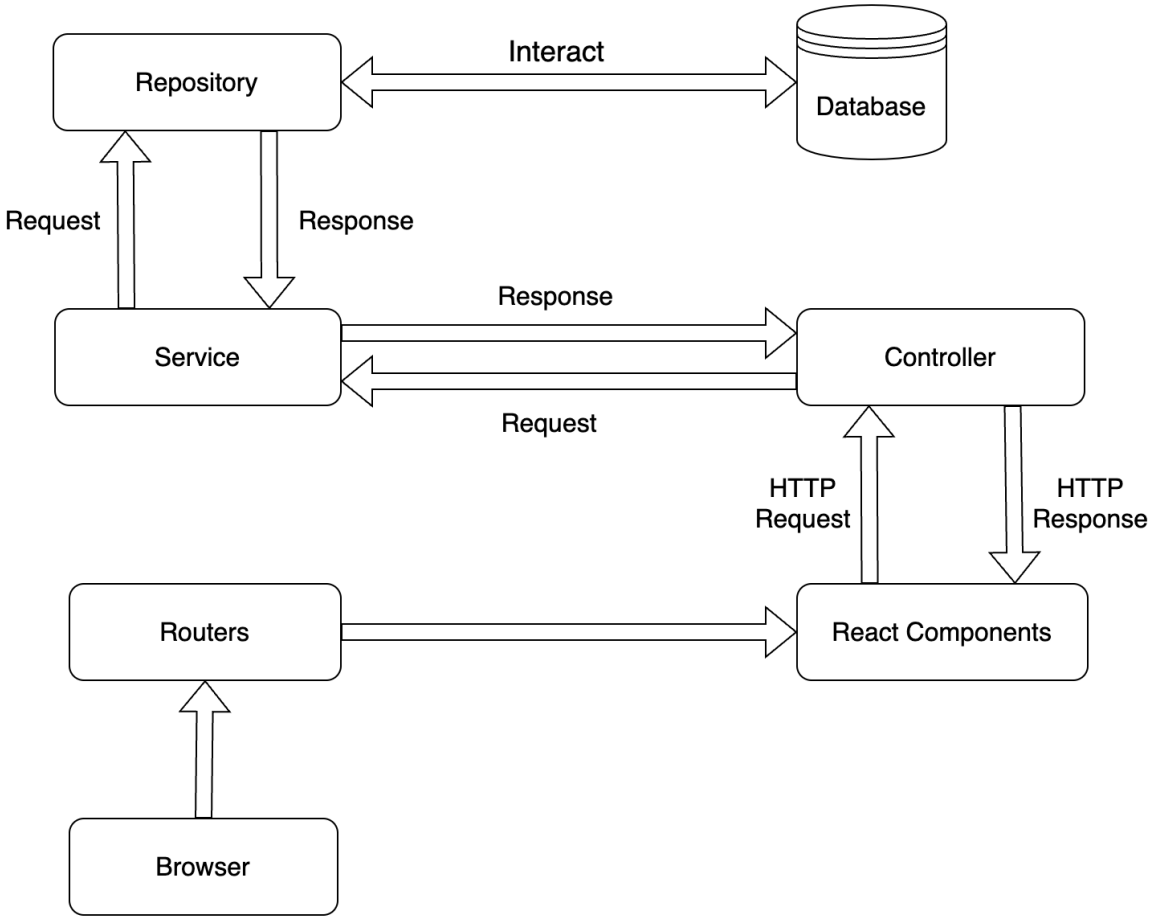
- Legacy Code and Reuse: The coding practices are defined and designed according to standards to ensure consistency among team members.

3. Use-Case Model



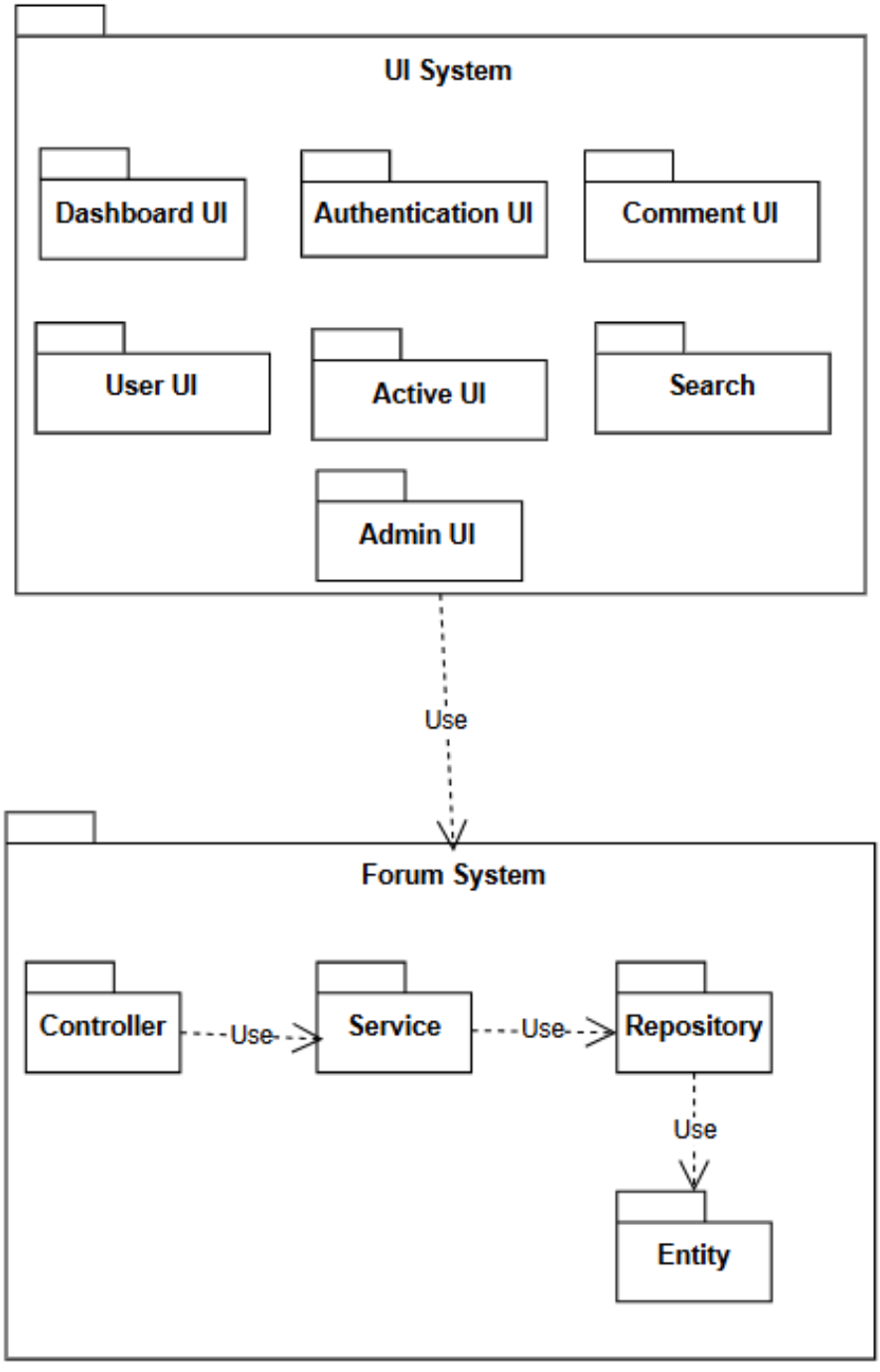

Create Post
Update Post
Delete Post
Post Management
Delete User
Block User
User Management
Change password
Change email
Account Management
Update Profile
Profile Management

4. Logical View



- **Database:** Use MySQL for Identity Information and MongoDB for other data like: user profile, post, comment, notification, message …
- **Repository:** Directly interact with database by SQL (MySQL) and NoSQL(MongoDB)
- **Service:** Handles business logic of the application and is an intermediary between the controller and the repository.
- **Controller:** Define entry point for HTTP requests. Receive HTTP request from client and return HTTP response to client.
- **React Components:** Interact with Server through HTTP request. Render UI based on router and application state.
- **Routers:** Manages routing on the client side based on the URL. Maps paths to React Components.
- **Browser:** Provides user interface to interact with the application. Sends requests by typing URLs, clicking links, or interacting with UI elements.

Package Diagram



4.1 Component: Database

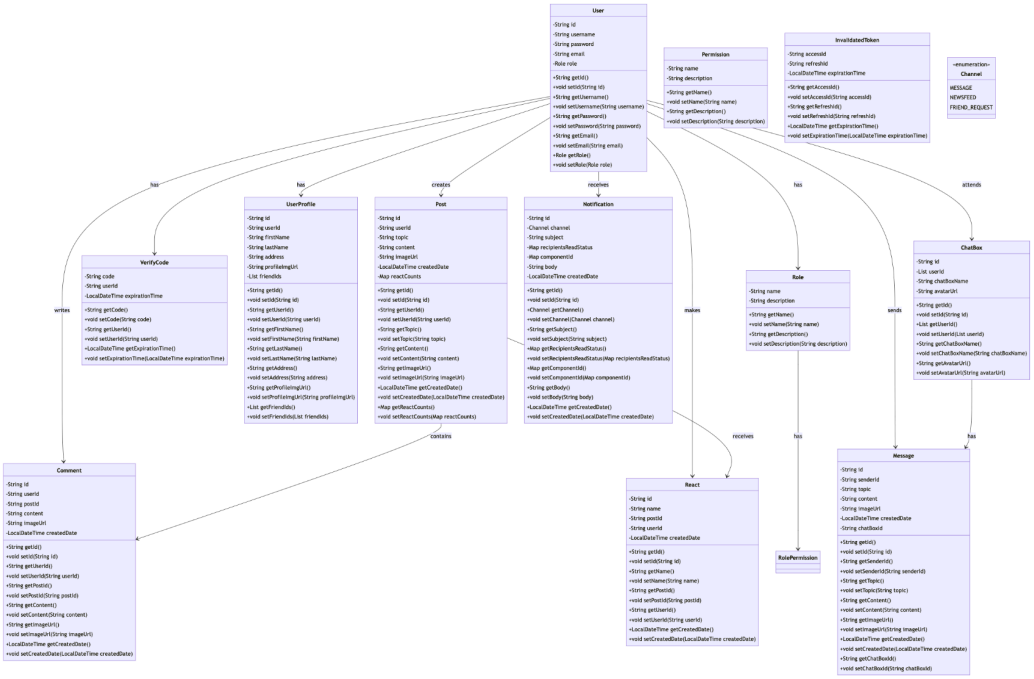
- **Responsibilities:** Use MySQL for Identity Information and MongoDB for other data like: user profile, post, comment, notification, message …

- **Main Entities:**
  a. **User:** saves user's account information includes: username, password, email.
  b. **Role:** saves application's roles.
  c. **Permission:** saves roles' permissions.
  d. **VerifyCode:** saves verify code used to verify email when registering, change password, update email. If it is still in the database, the email isn't verified, the activity like login, change password, update email will not be successful.
  e. **Invalidated Token:** saves expiration token, invalid token (the token that is saved when logging out or refreshing token).
  f. **UserProfile:** saves user profile includes: first name, last name, address, date of birth…
  g. **Post:** saves post information includes: title, content, image urls (the images are saved in Cloud), created date, id of the post creator.
  h. **Comment:** saves comment information includes: content, image urls (the images are saved in Cloud), created date, post id (the post that the comment belongs to), id of the comment creator.
  i. **React:** saves created date, id of the reaction creator, post id (the post that the reaction belongs to).
  j. **Notification:** saves application notification like new post, add friend request, new message, new comment, post report…
  k. **NotificationRecipient:** save notification of each user
  l. **ChatRoom:** saves chat room that user creates.
  m. **Message:** saves message in each chat room.
  n. **AddFriendRequest:** saves the information of add friend requests
  o. **ReportViolatingPostsRequest:** saves the information of add friend requests
- **Class diagram:**

  Link: [Online FlowChart & Diagrams Editor - Mermaid Live Editor]

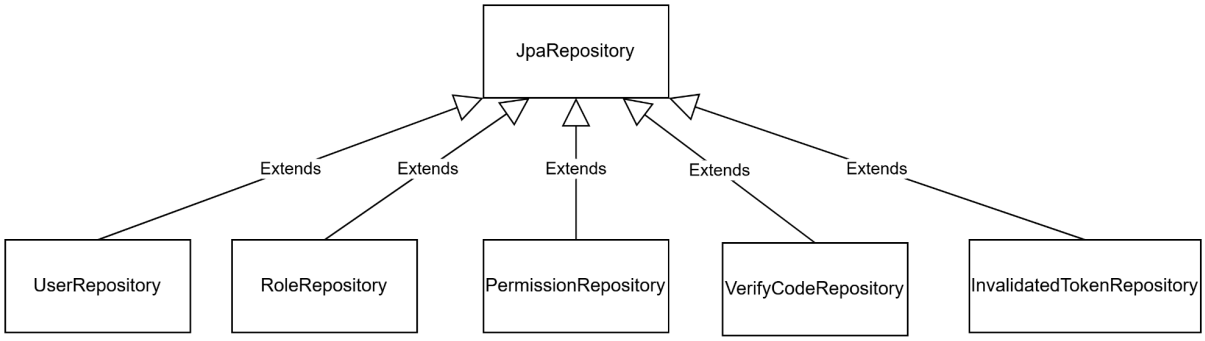
  Instruction: Click on the Code section and press Enter to display the Diagram

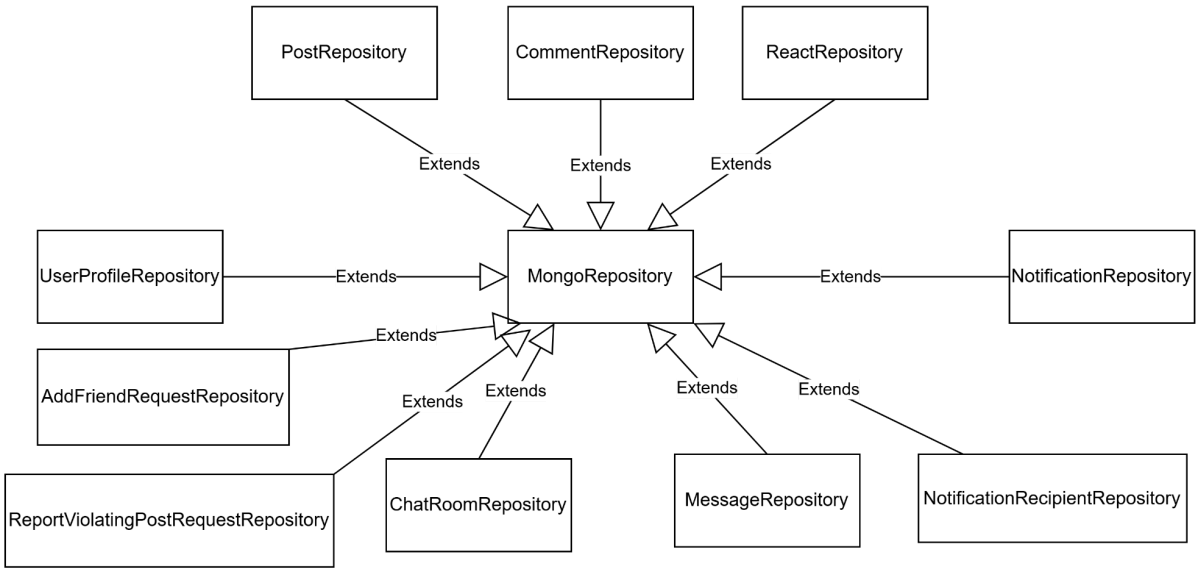
  Demo image:



4.2 Component: Repository

- **Responsibilities:** Directly interact with database by SQL (MySQL) and NoSQL(MongoDB)
- **Class Diagram:**

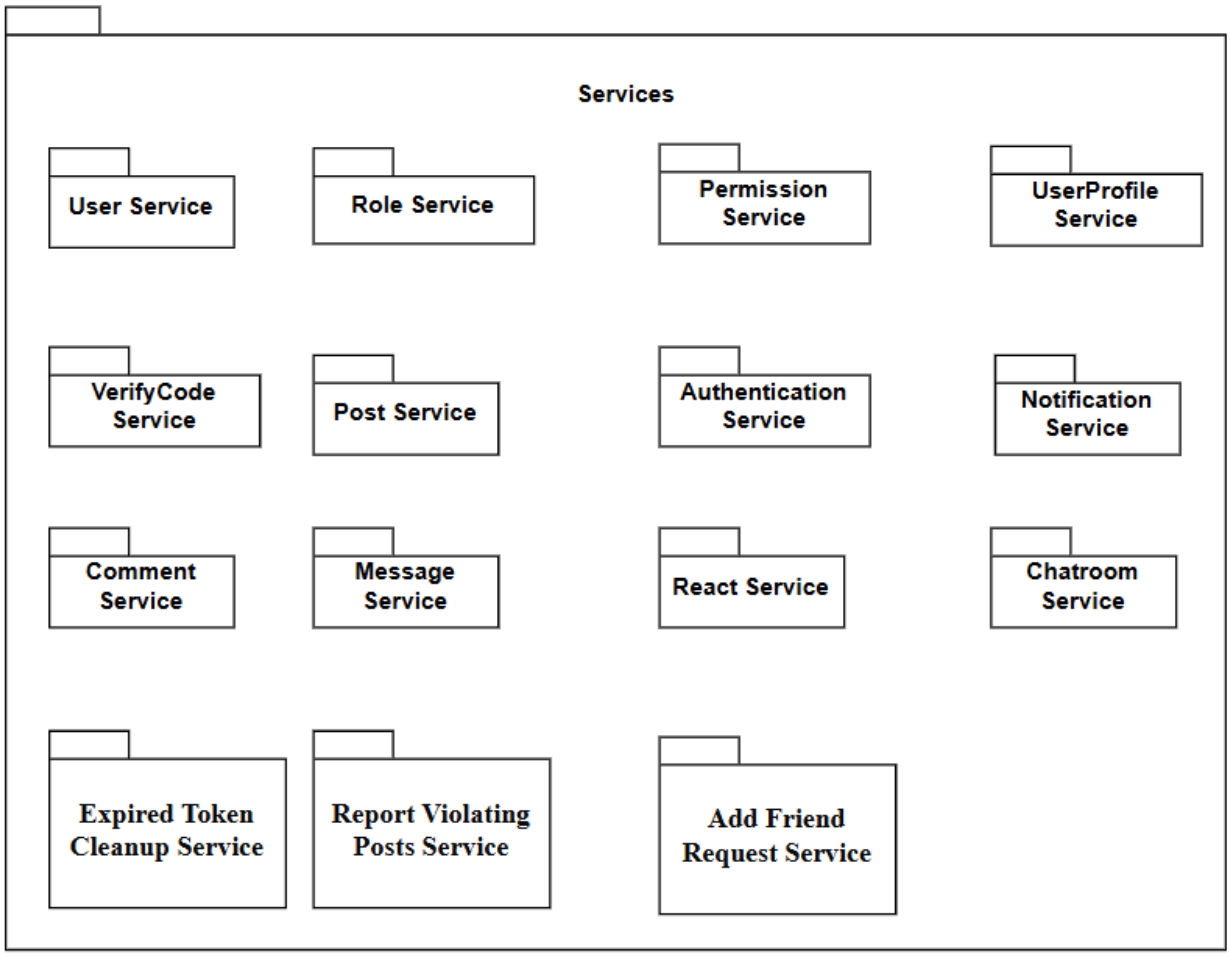




- **Key Interfaces:**
  a. **UserRepository:** provides methods for CRUD operation on User entity.
  b. **RoleRepository:** provides methods for CRUD operation on Role entity.
  c. **PermissionRepository:** provides methods for CRUD operation on Permission entity.
  d. **VerifyCodeRepository:** provides methods for persisting verify code in database to mark as account hasn't been verified and delete verify code in database in verifying account feature.
  e. **InvalidatedTokenRepository:** provides methods for invalidating and cleaning up invalid token in database.
  f. **UserProfileRepository:** provides methods for CRUD operation on User Profile.
  g. **PostRepository:** provides methods for CRUD operation on Post entity.
  h. **CommentRepository:** provides methods for CRUD operation on Comment entity.
  i. **ReactRepository:** provides methods for CRUD operation on React entity.

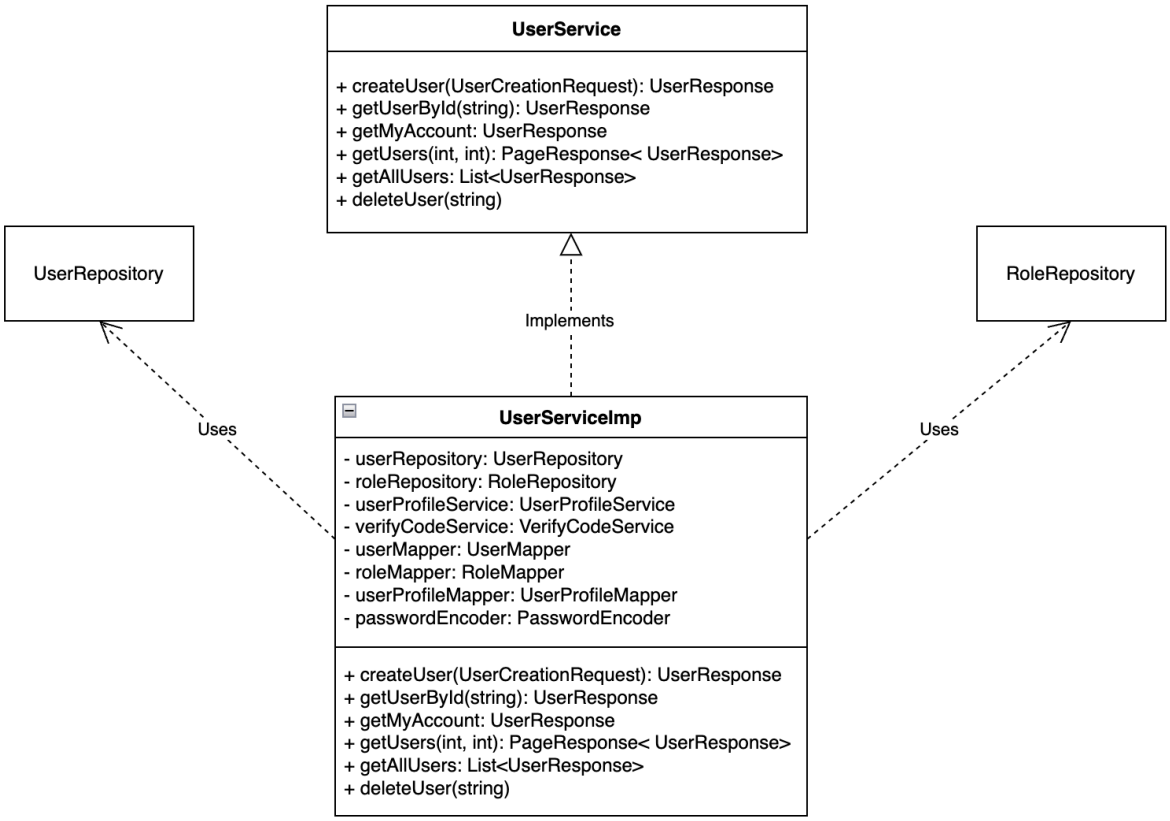
j. **NotificationRepository:** provides methods for CRUD operation on Notification entity.
k. **NotificationRecipientRepository:** provides methods for CRUD operation on NotificationRecipient entity.
l. **ChatRoomRepository:** provides methods for CRUD operation on ChatRoom.
m. **MessageRepository:** provides methods for CRUD operation on Message.

4.3 Component: Service



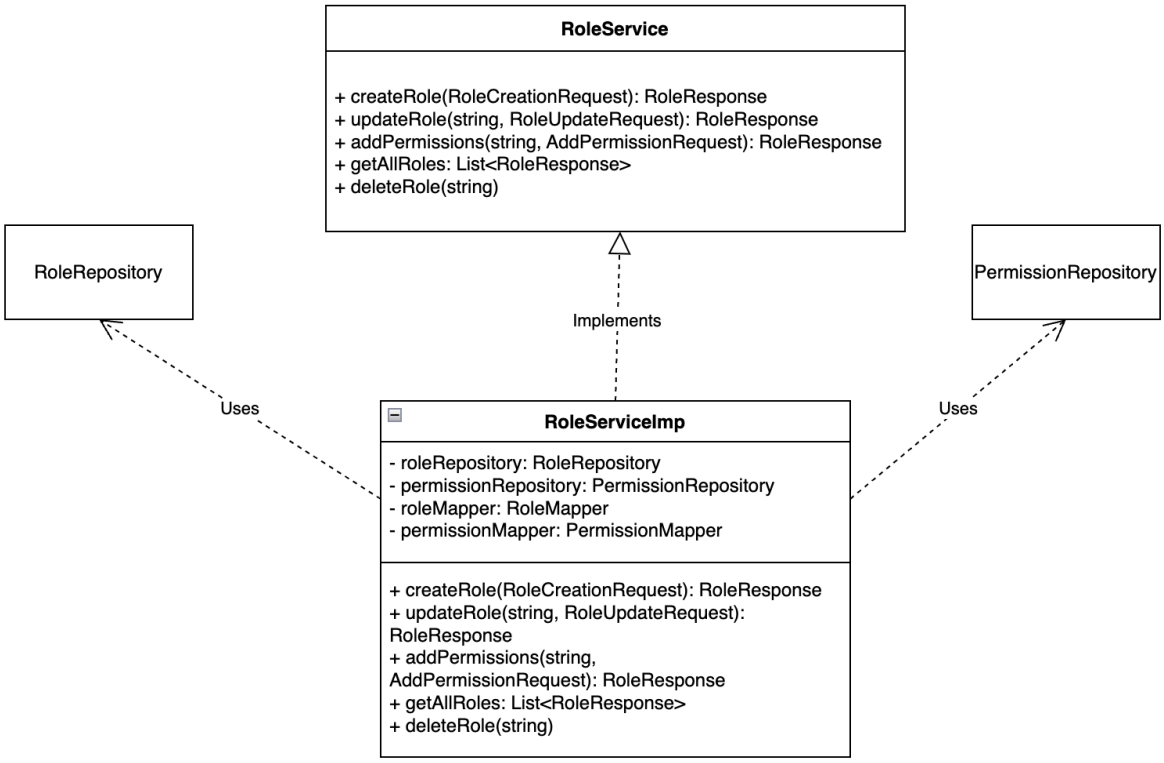
- **Responsibilities:** Handles business logic of the application and is an intermediary between the controller and the repository.
-

- **Main Classes:**
  a. **User Service:**
    - **Responsibilities:** Manages user-related actions like registration, login, account update.
    - **Class Diagram:**



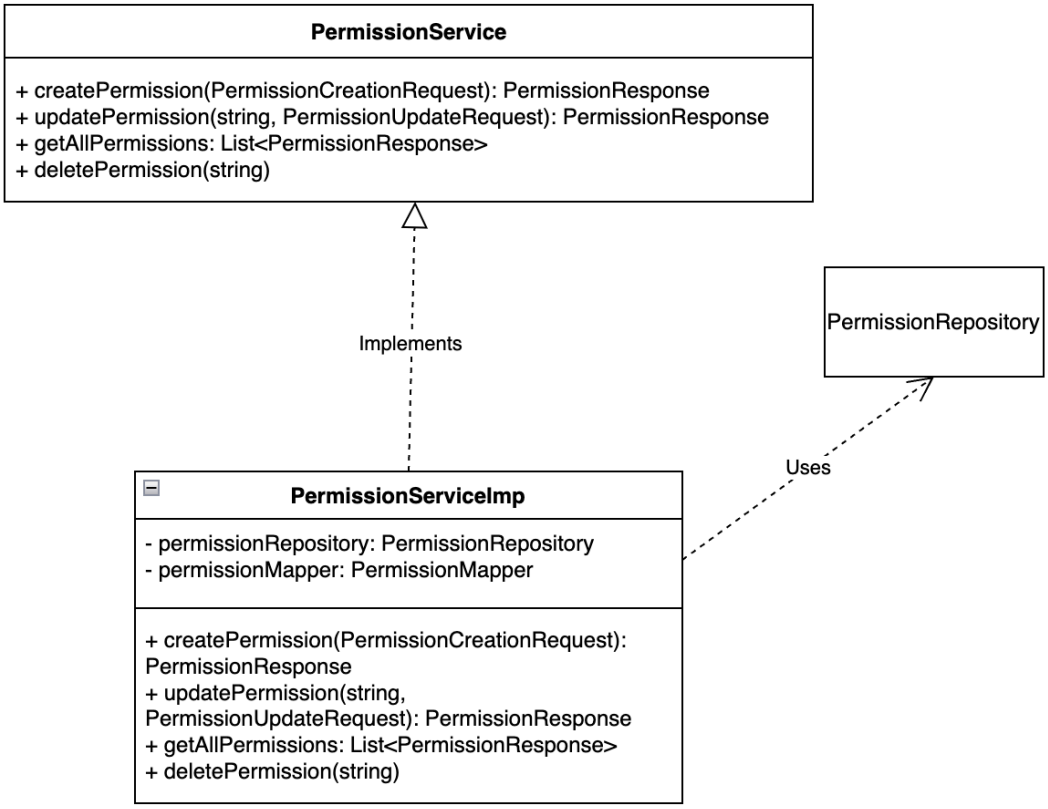
b. Role Service:

- **Responsibilities:** Manages application's roles (CRUD)
- **Class Diagram:**



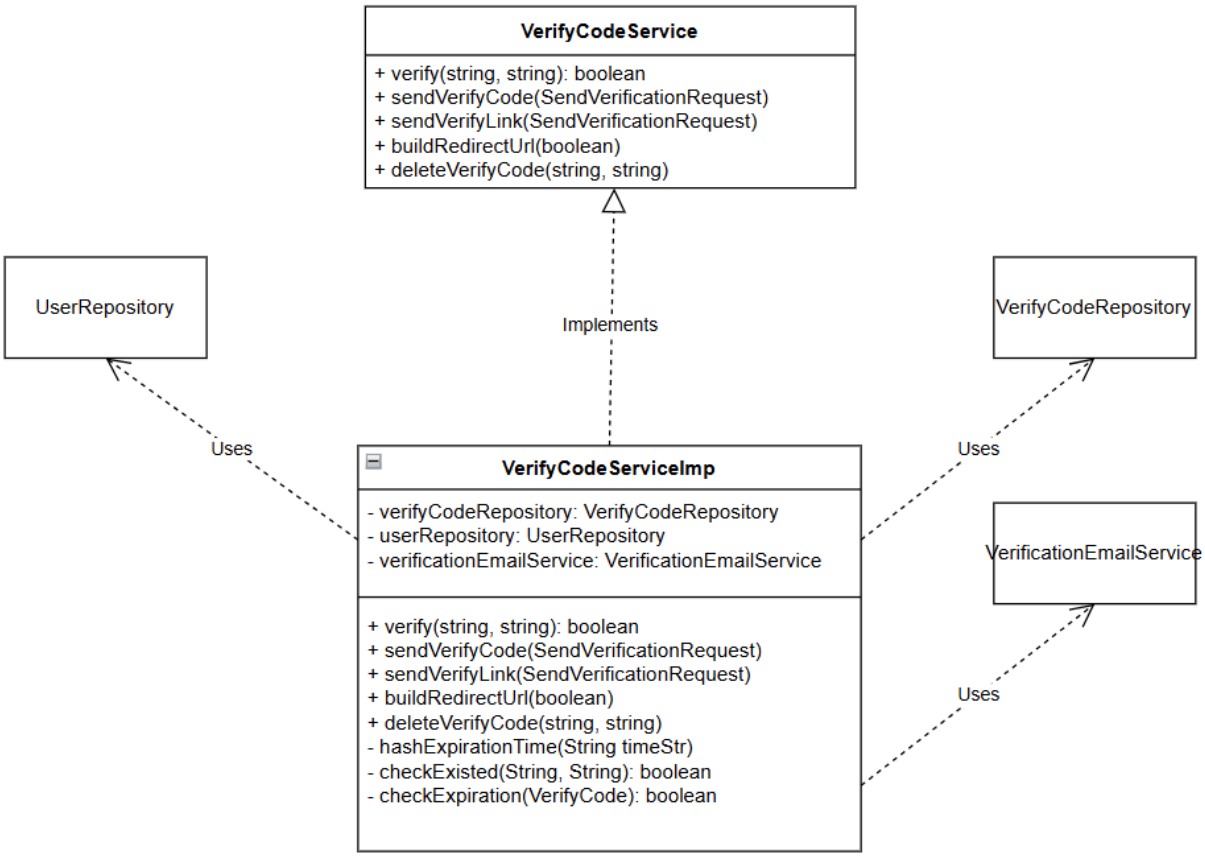
c. Permission Service:

- **Responsibilities:** Manages application's permissions (CRUD)
- **Class Diagram:**



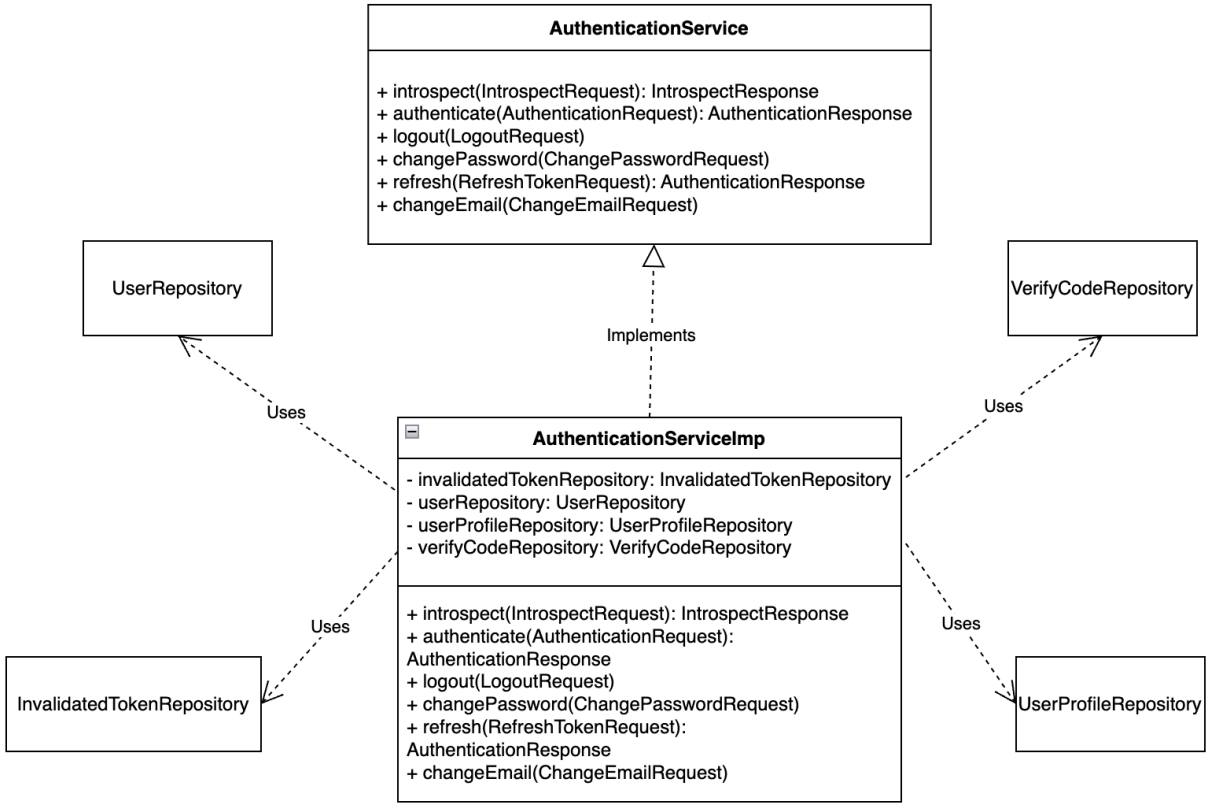
d. VerifyCode Service:

- **Responsibilities:** Manages verifying email operation like: Ensures the code is still valid, sends verification code and links for users updating credentials.
- **Class Diagram:**



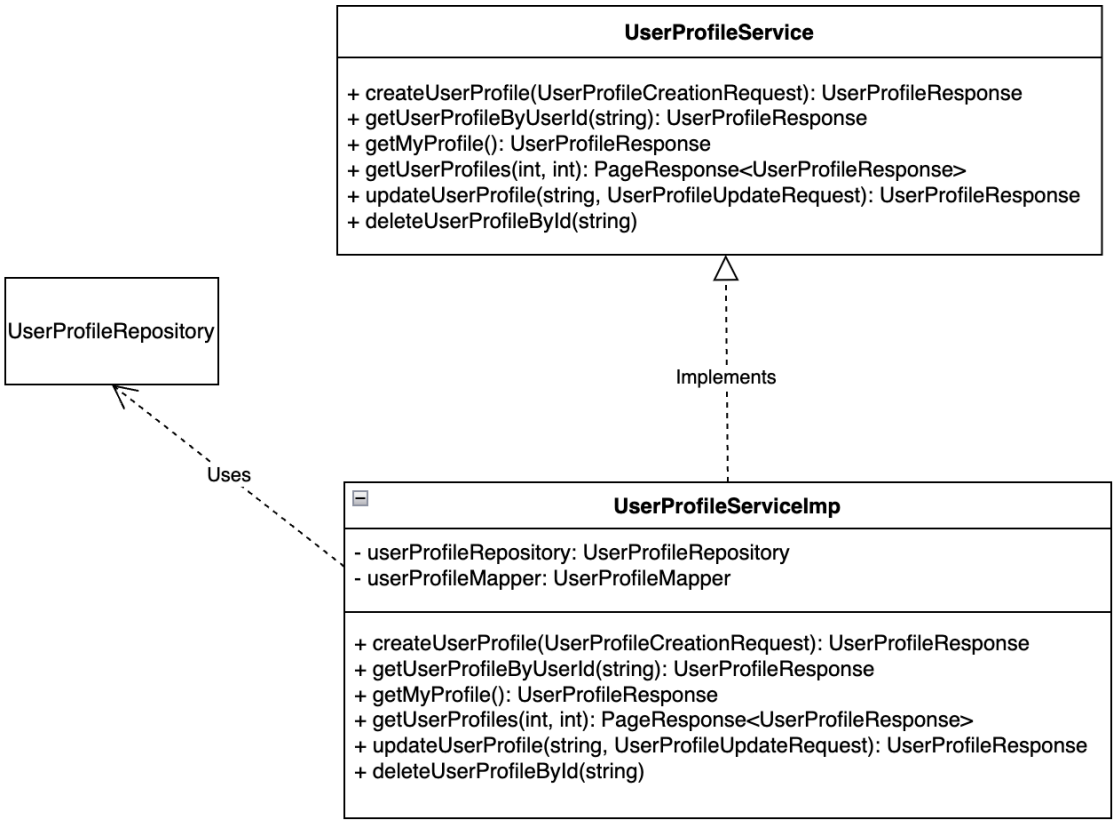
e. AuthenticationService

- **Responsibilities:** Handles authentication feature of the application like authenticate account (check valid username, password and send jwt to client), logout, change password, change email, refresh token, check validity of token.
- **Class Diagram:**



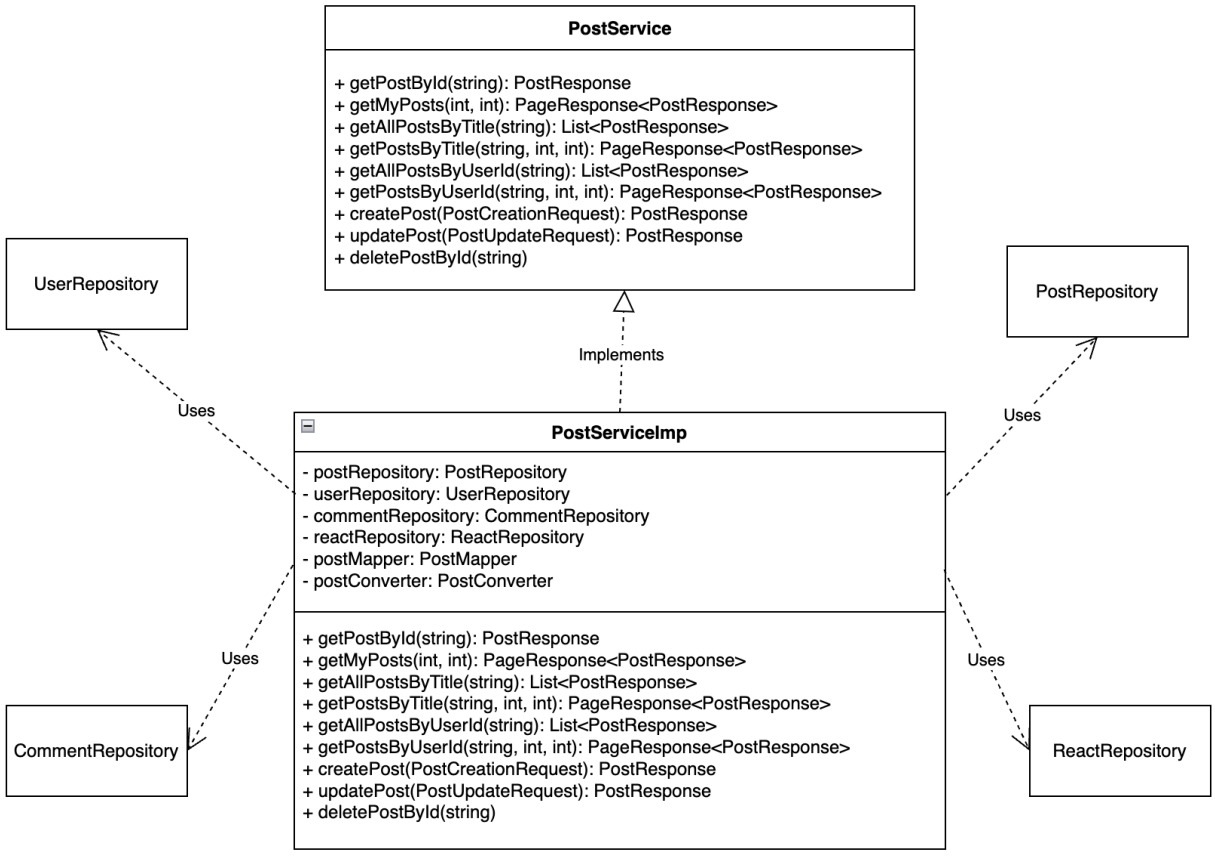
f. UserProfile Service:

- **Responsibilities:** manages user profile (CRUD).
- **Class Diagram:**



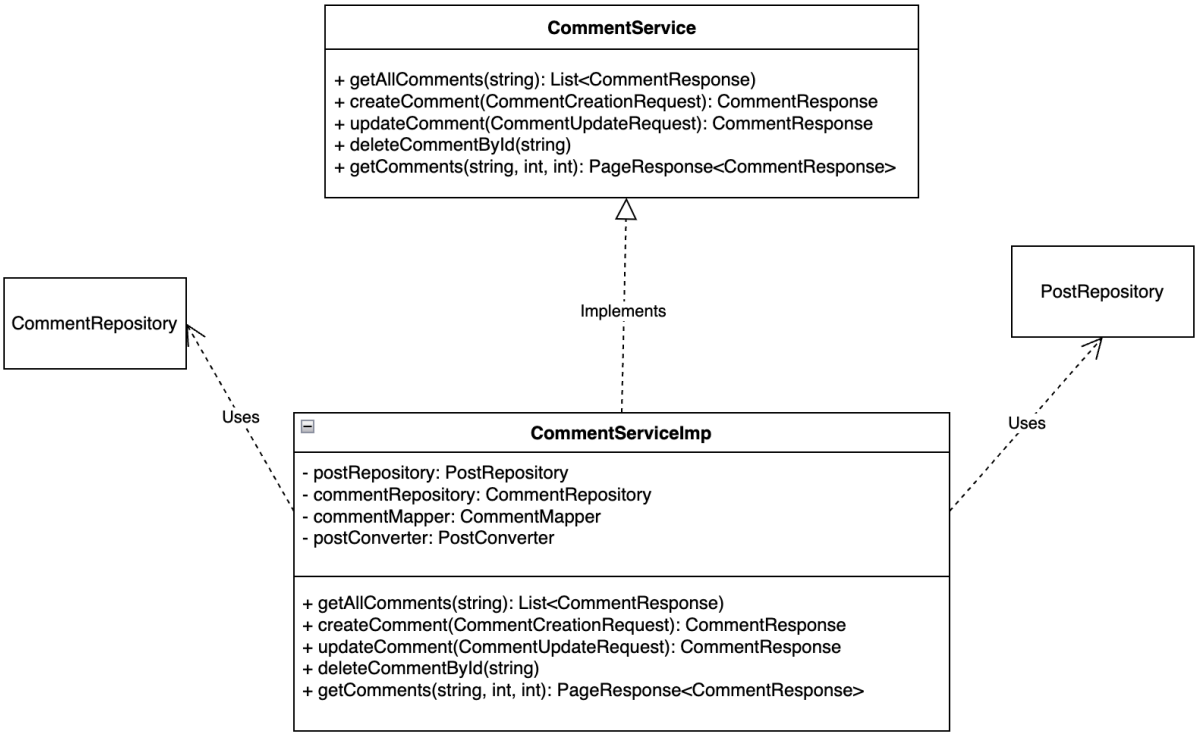
g. PostService:

- **Responsibilities:** manages post CRUD operation
- **Class Diagram:**



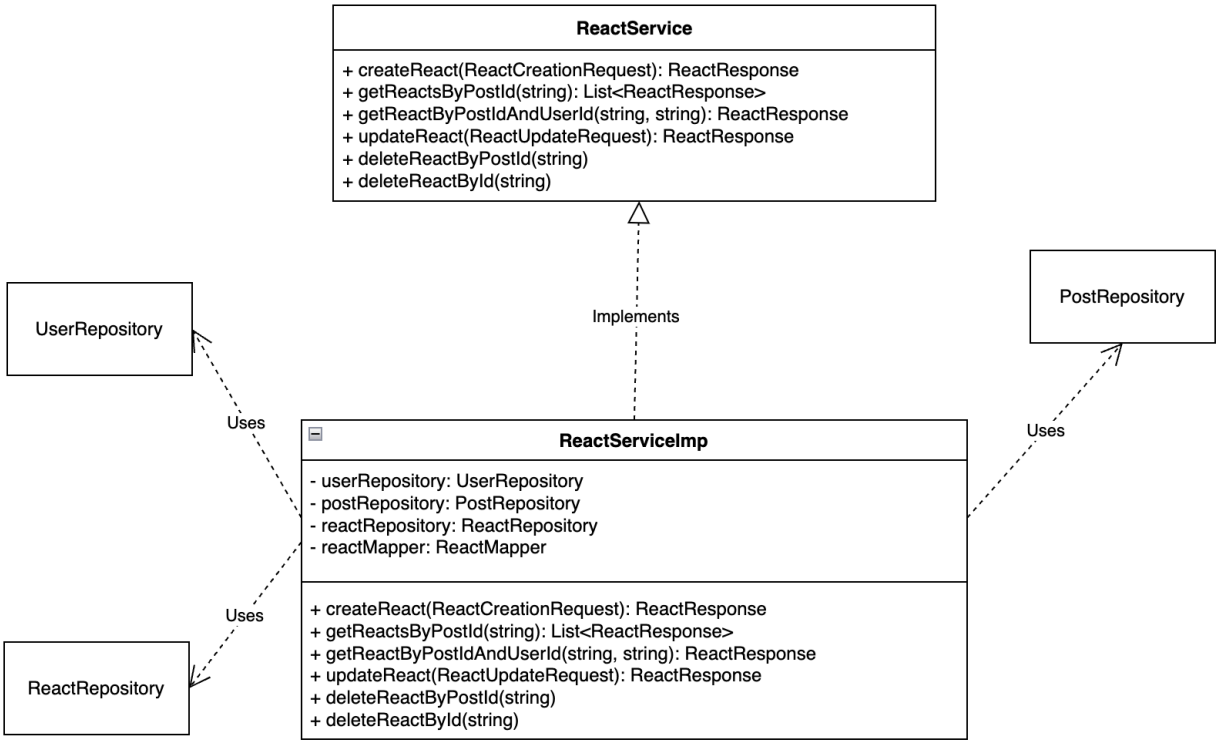
h. Comment Service:

- **Responsibilities:** manages posts' comments' CRUD operation.
- **Class Diagram:**



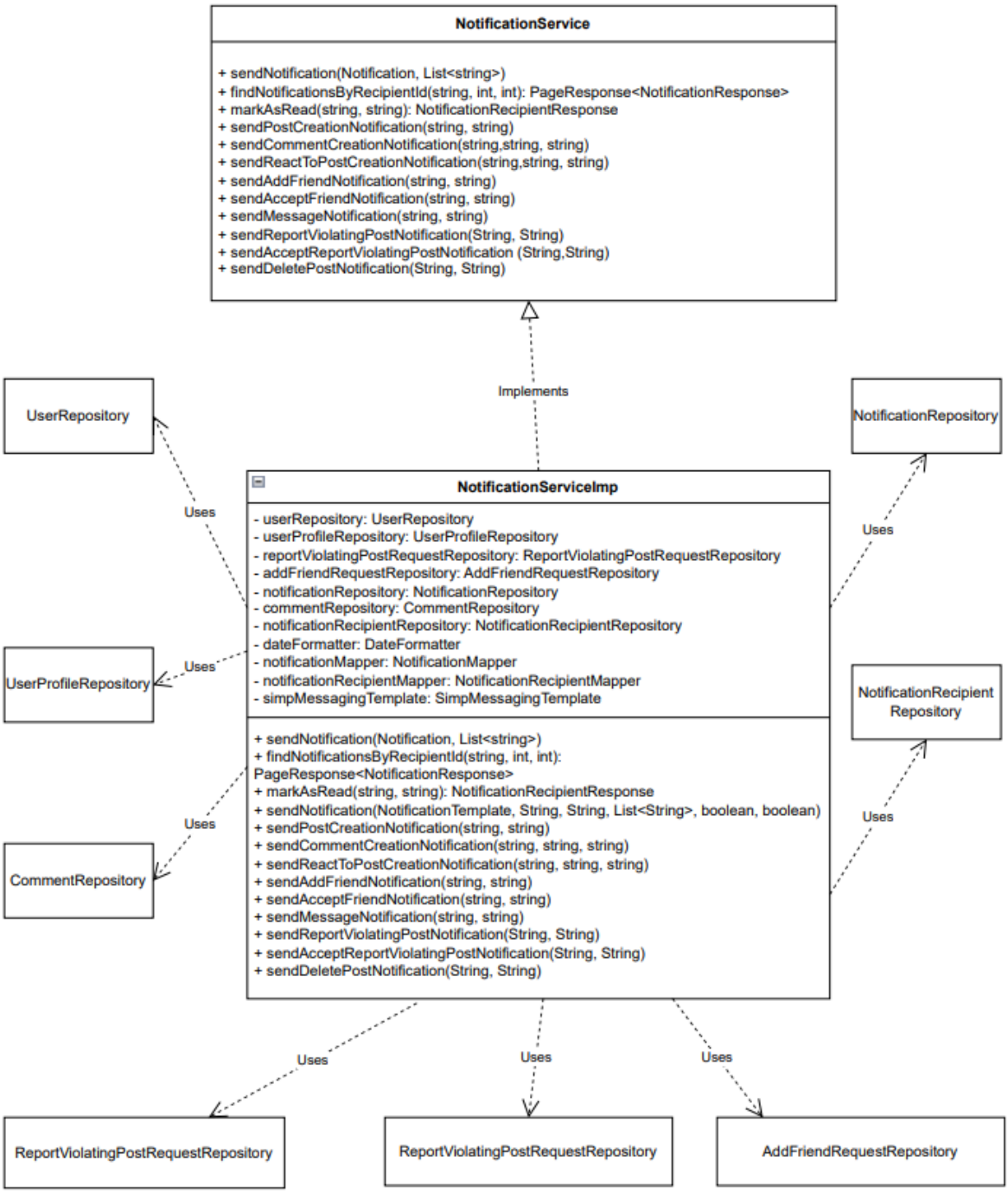
i. React Service:

- **Responsibilities:** manages posts' reacts' CRUD operation.
- **Class Diagram:**



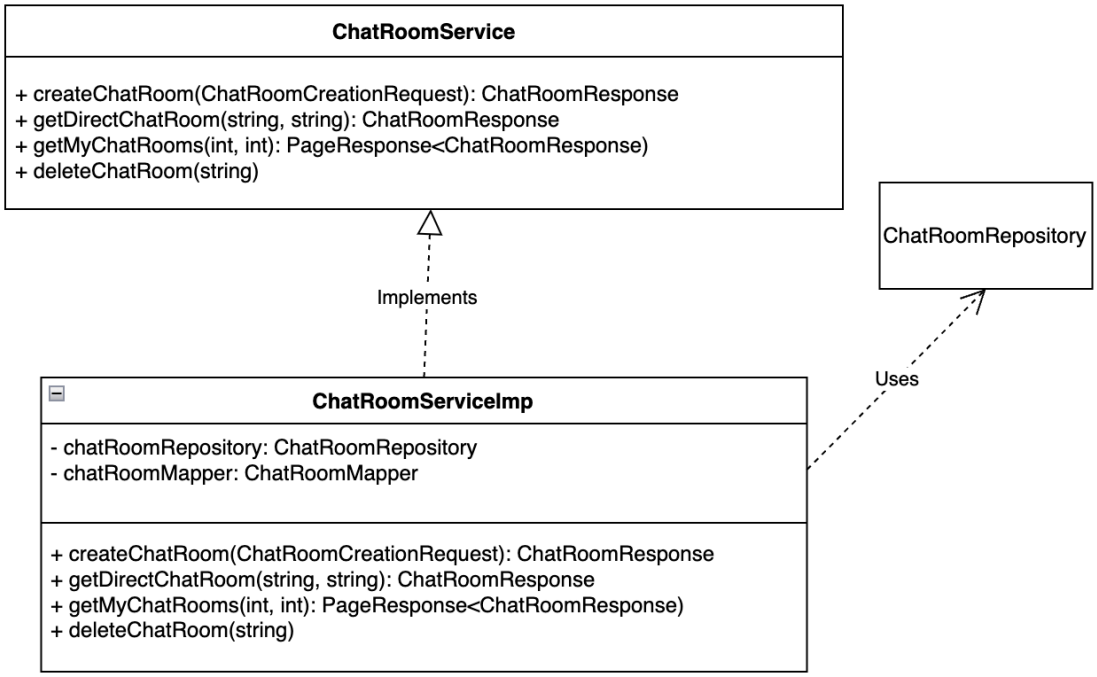
j. Notification Service:

- **Responsibilities:** Handle sending notifications to inform users about activities such as creating a new post, reacting, sending messages, and sending friend requests.
- **Class Diagram:**



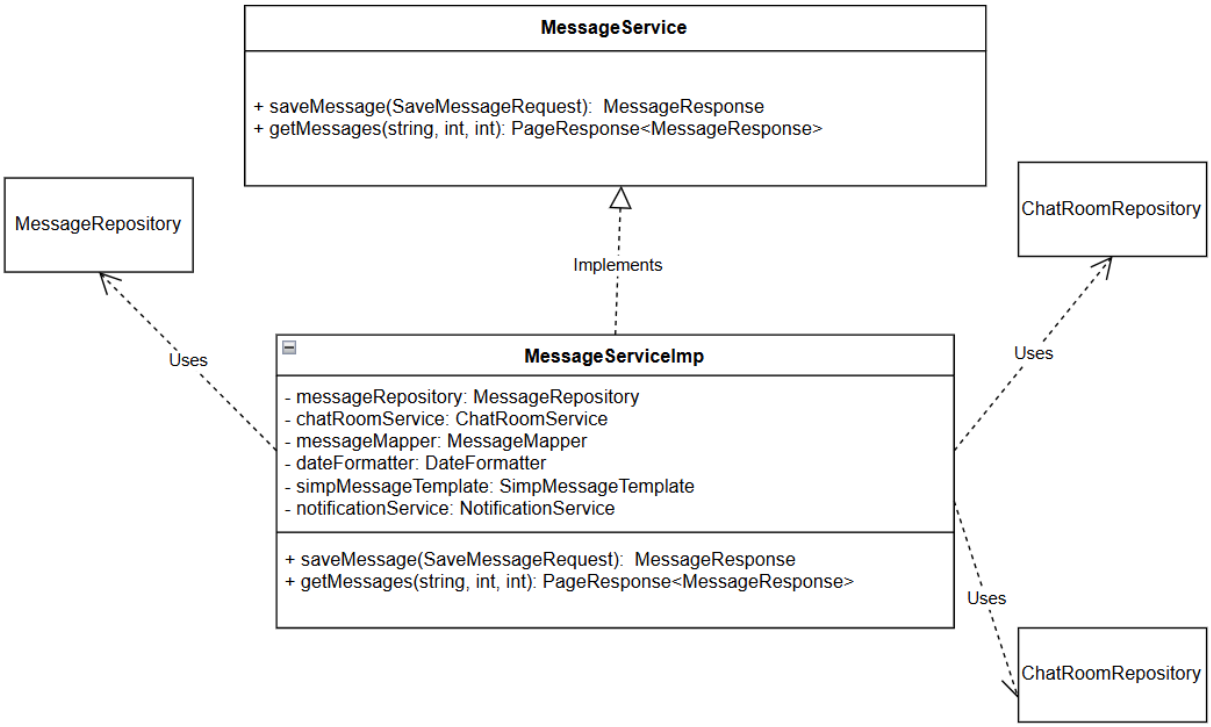
k. ChatRoom Service:

- **Responsibilities:** manage chat room CRUD operation
- **Class Diagram:**



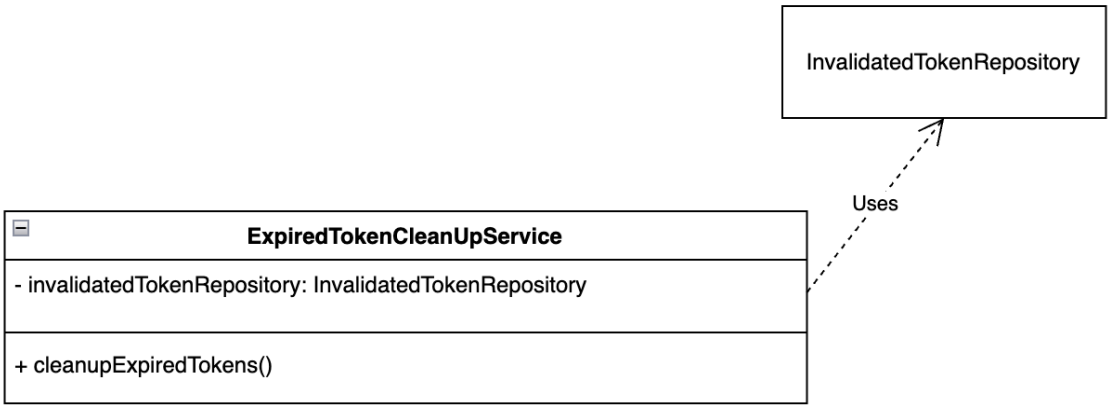
l. Message Service

- **Responsibilities:** manages message CRUD operation
- **Class Diagram:**



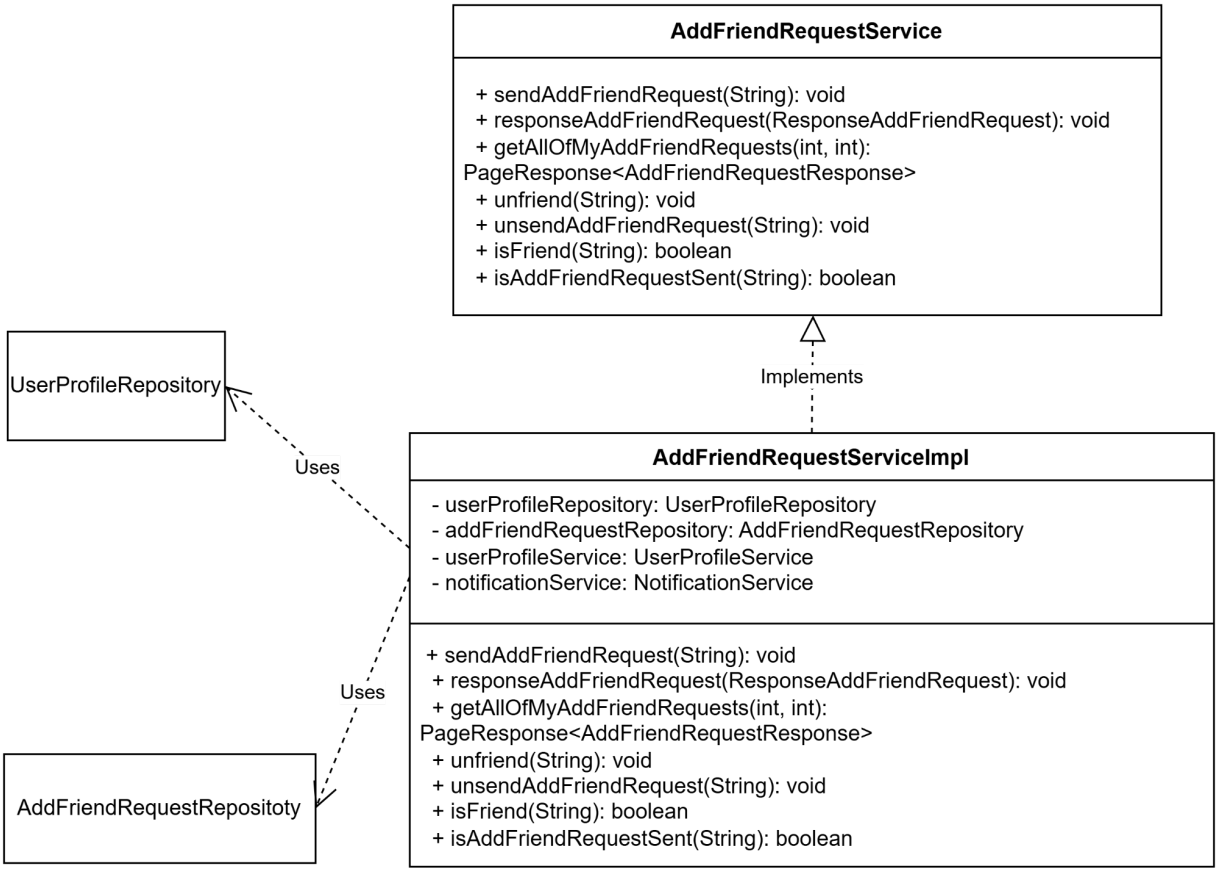
m. Expired Token Cleanup Service

- **Responsibilities:** Deletes Expiration Token in database.
- **Class Diagram:**



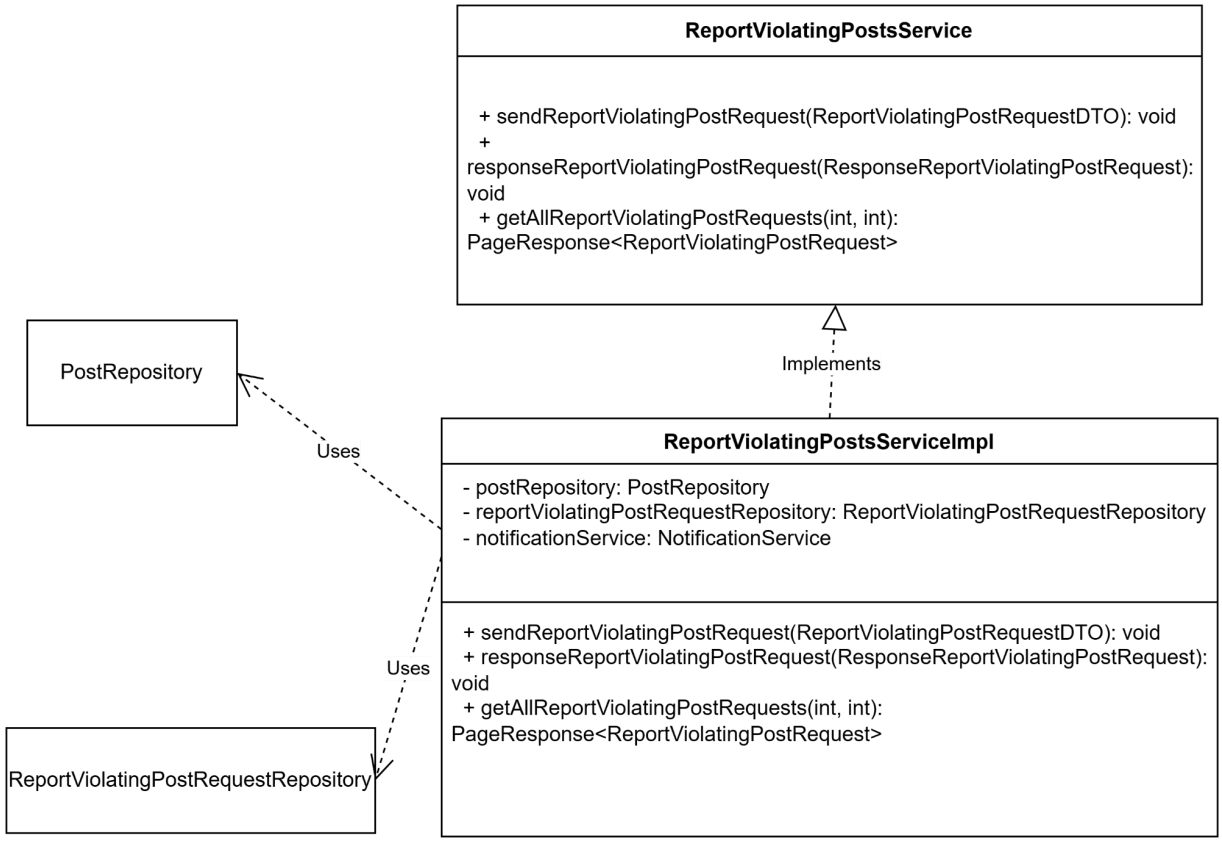
n. Add Friend Request Service

- **Responsibilities:** Manage add friend requests
- **Class Diagram:**

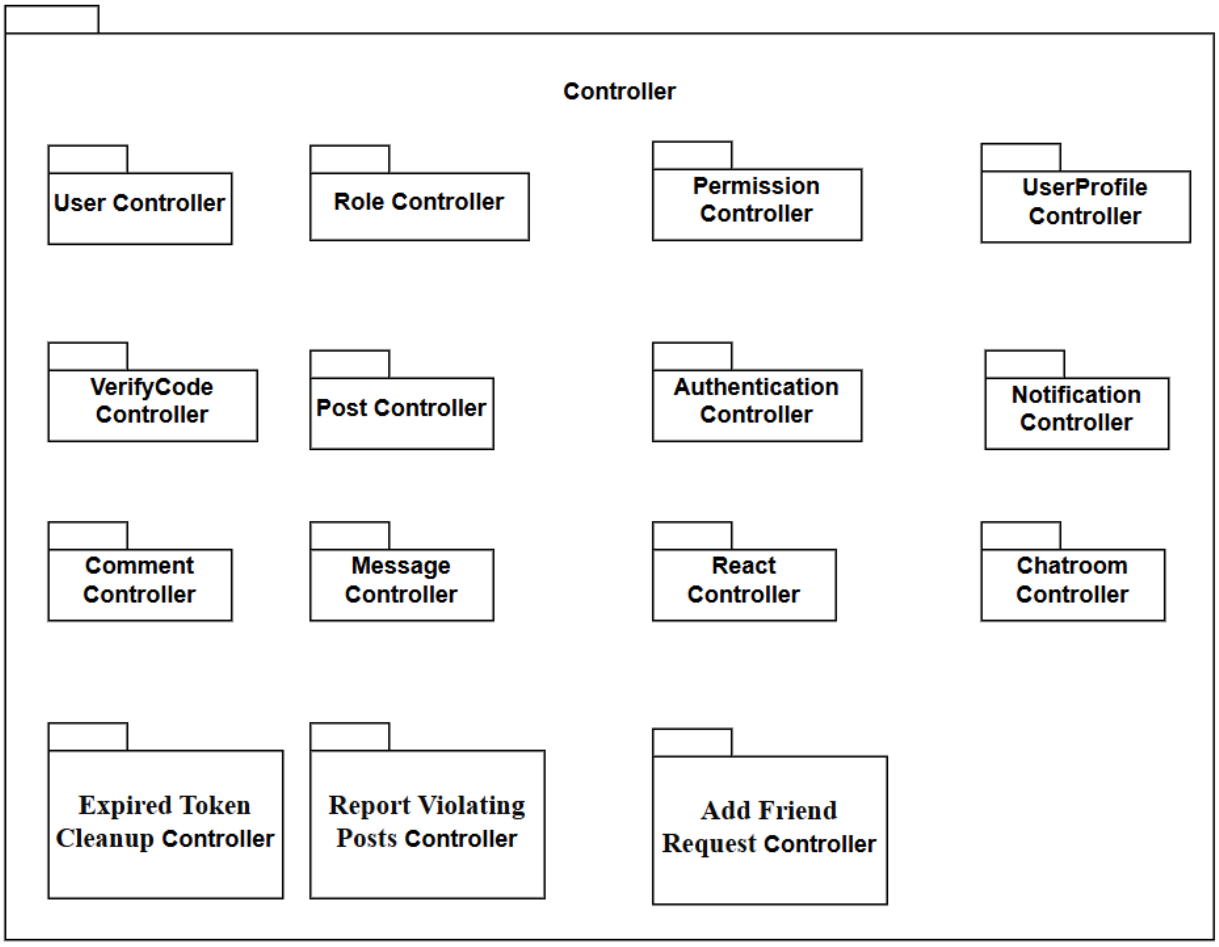


o. Report Violating Posts Service

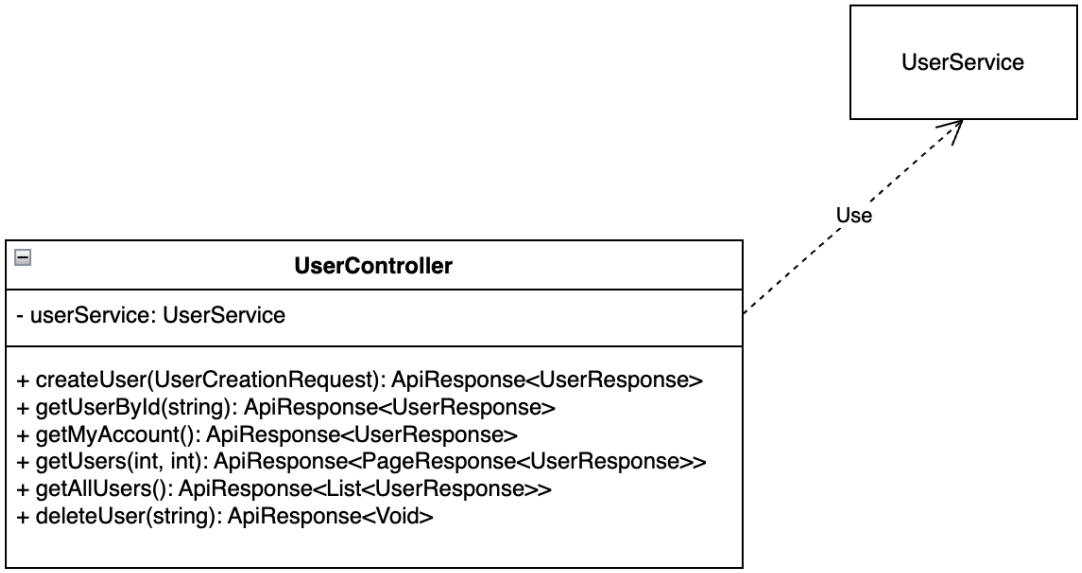
- **Responsibilities:** Manage report violating posts
- **Class Diagram:**



4.4 Component: Controller

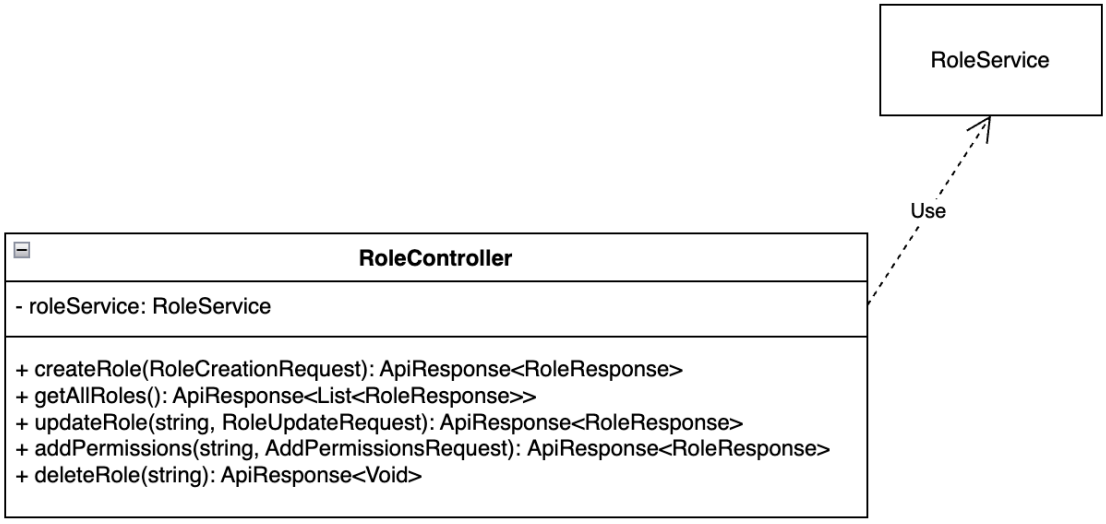


- **Responsibilities:** Define entry point for HTTP requests. Receive HTTP request from client and return HTTP response to client.
- **Main Classes:**
    a. **UserController:**
        - **Responsibilities:** Defines user-related endpoints, is an intermediary between User Service and Client
        - **Class Diagram:**



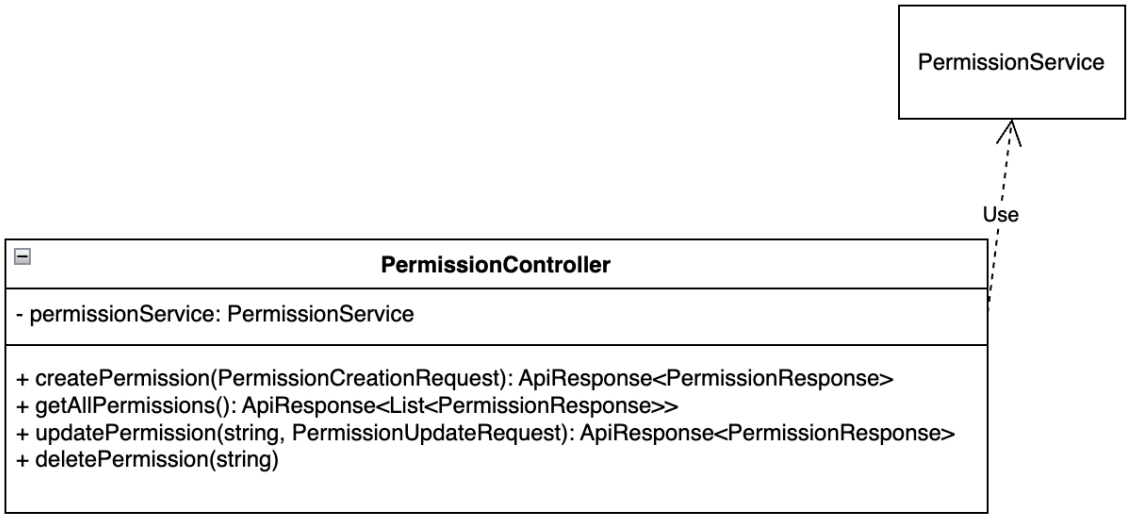
b. RoleController:

- **Responsibilities:** Defines role-related endpoints, is an intermediary between Role Service and Client
- **Class Diagram:**



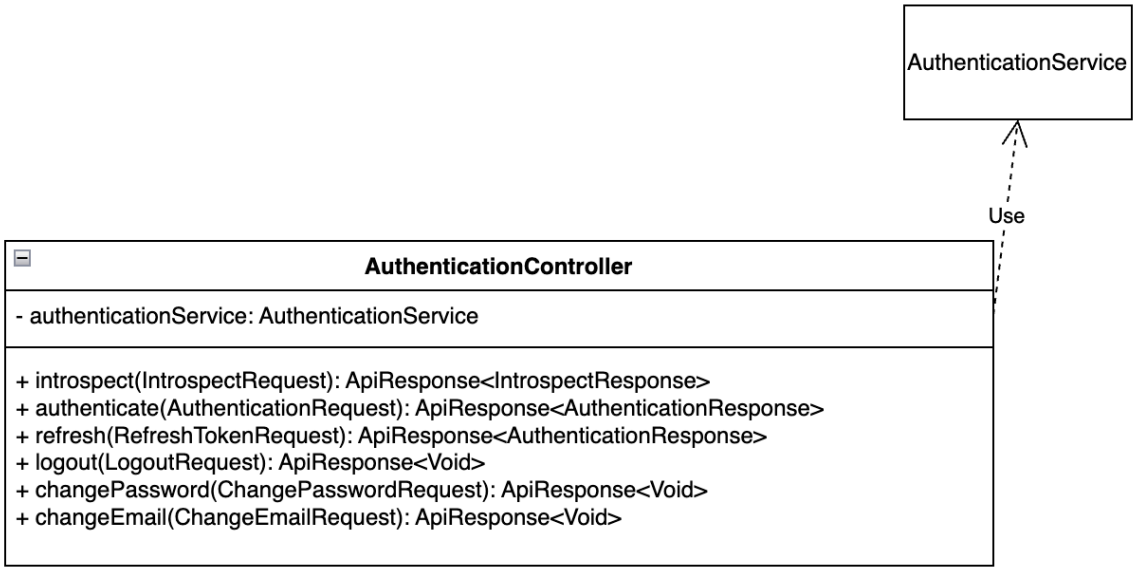
c. PermissionController:

- **Responsibilities:** Defines permission-related endpoints, is an intermediary between Permission Service and Client
- **Class Diagram:**



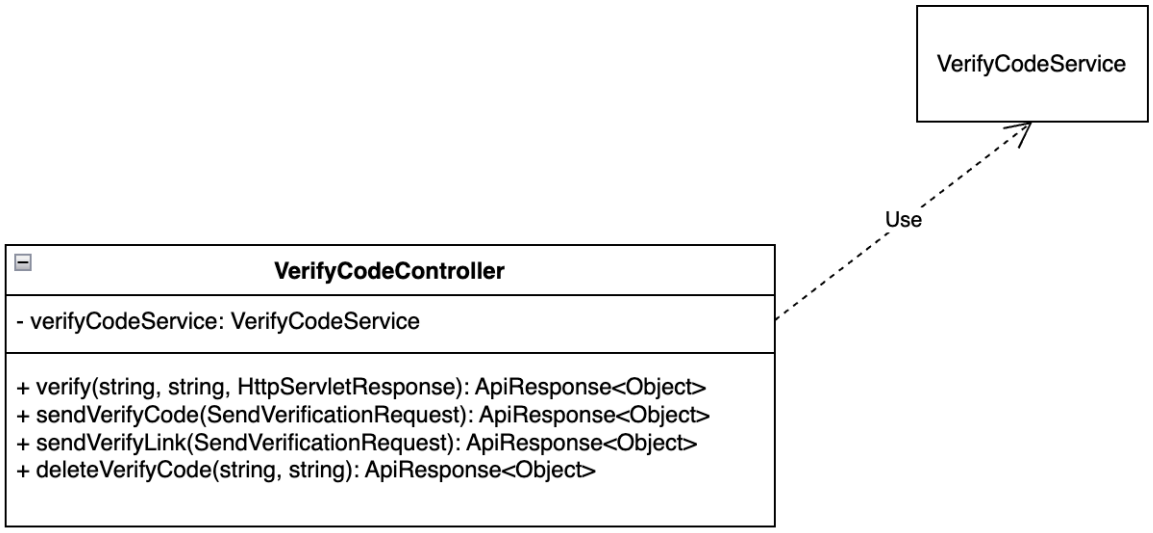
d. Authentication Controller:

- **Responsibilities:** Defines authentication-related endpoints, is an intermediary between Authentication Service and Client
- **Class Diagram:**



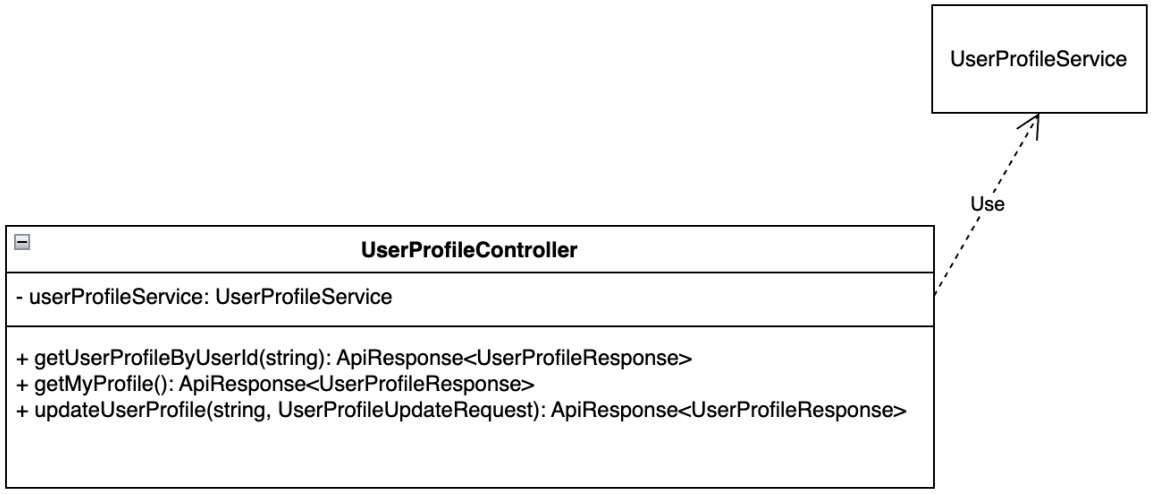
e. VerifyCode Controller

- **Responsibilities:** Defines verifycode-related endpoints, is an intermediary between VerifyCode Service and Client
- **Class Diagram:**



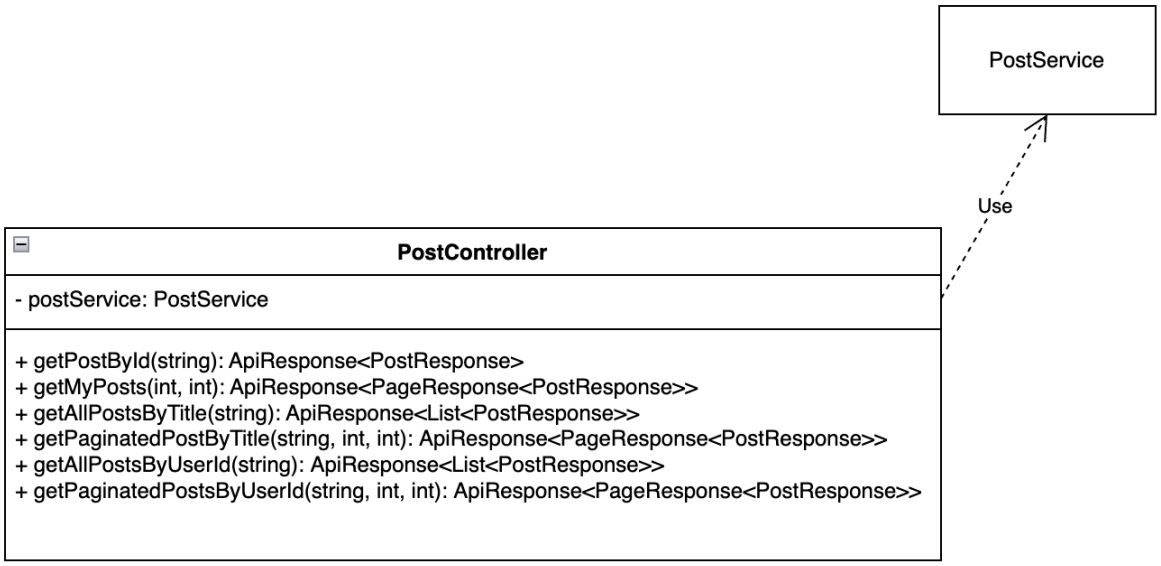
f. UserProfile Controller

- **Responsibilities:** Defines userprofile-related endpoints, is an intermediary between Userprofile Service and Client
- **Class Diagram:**



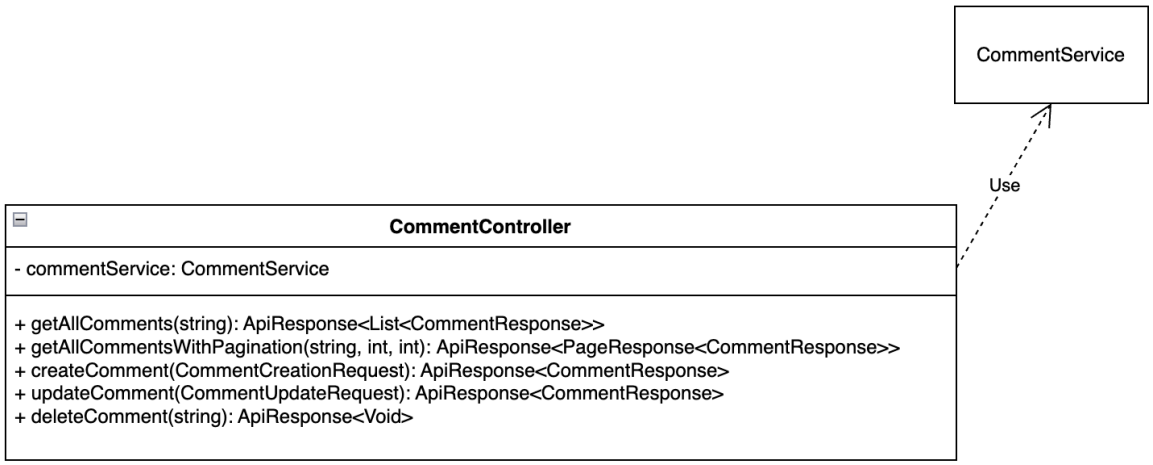
g. Post Controller:

- **Responsibilities:** Defines post-related endpoints, is an intermediary between Post Service and Client
- **Class Diagram:**

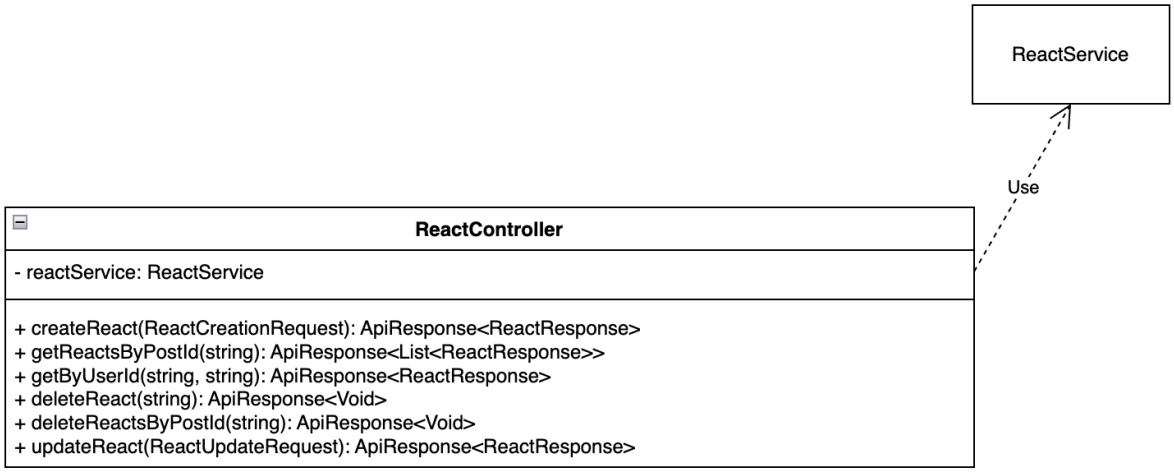


h. Comment Controller:

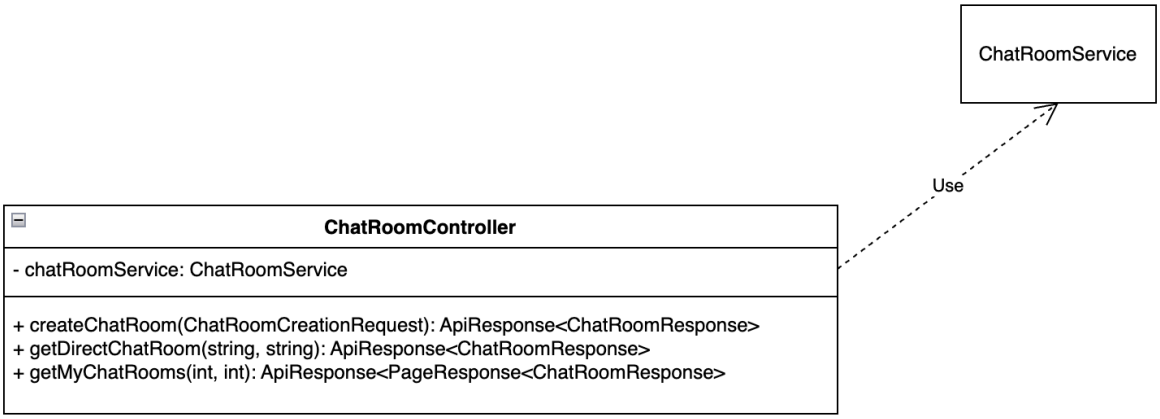
- **Responsibilities:** Defines comment-related endpoints, is an intermediary between Comment Service and Client
- **Class Diagram:**



i. **React Controller:**
   - **Responsibilities:** Defines react-related endpoints, is an intermediary between React Service and Client
   - **Class Diagram:**

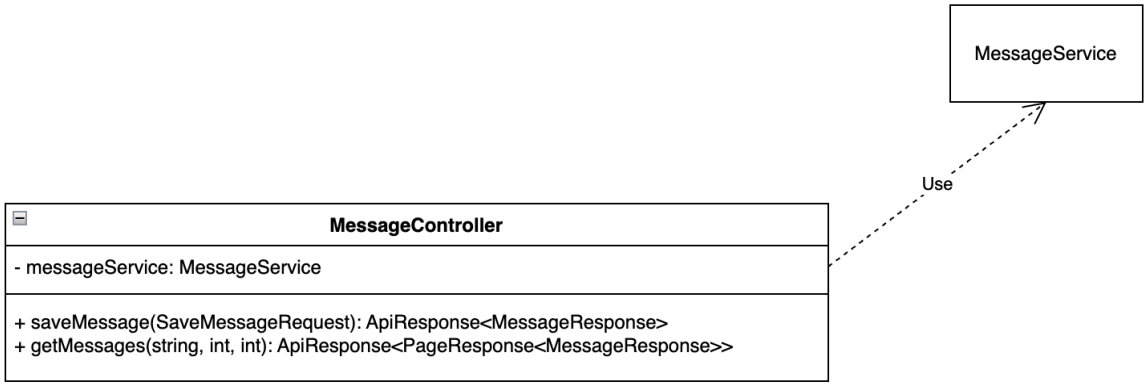


j. **ChatRoom Controller:**
   - **Responsibilities:** Defines chatroom-related endpoints, is an intermediary between ChatRoom Service and Client
   - **Class Diagram:**



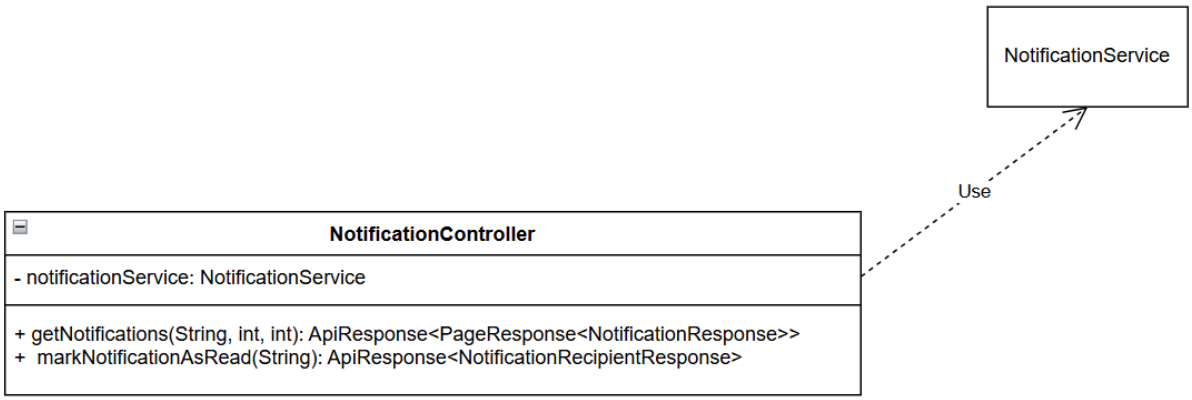
k. MessageController

- **Responsibilities:** Defines message-related endpoints, is an intermediary between Message Service and Client
- **Class Diagram:**



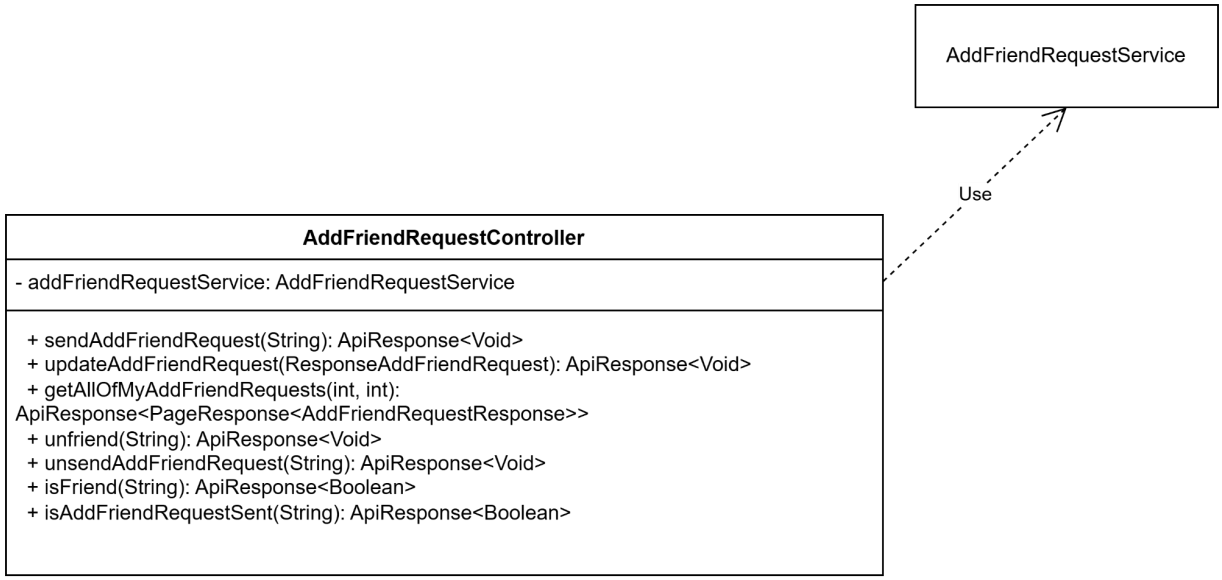
l. NotificationController

- **Responsibilities:** Defines notification-related endpoints, is an intermediary between Notification Service and Client
- **Class Diagram:**



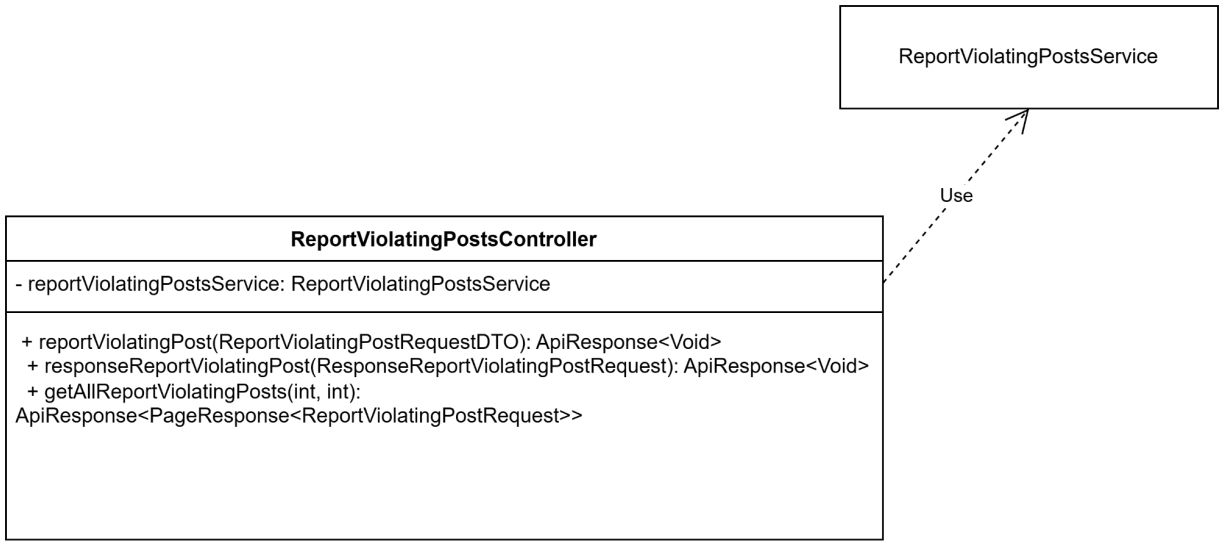
m. AddFriendRequestController

- **Responsibilities:** Defines addfriend-related endpoints, is an intermediary between AddFriendRequestService and Client
- **Class Diagram:**



n. ReportViolatingPostsController

- **Responsibilities:** Defines report posts-related endpoints, is an intermediary between ReportViolatingPoststService and Client
- **Class Diagram:**



4.5 Component: Views

- **Responsibilities:**
  - Rendering the user interface.
  - Managing its internal state and using props passed from parent components.
  - Handling user events and interactions.
  - Passing data to child components and maintaining the flow of information.
  - Ensuring reusability and performance optimization.
- **Some main components:**
  a. **Login Component:**
     - **Responsibilities:**
       - Display the login interface.
       - Manage the state of the user's input (email, password).
       - Handle the login event (send a login request to the server).
       - Validate and show error messages if necessary.
     - **Child:**
       - **Login form component:** This is where users enter their login details (email and password). It contains input fields and a submit button.

  b. **Register Component:**
     - **Responsibilities:**
       - Display the user registration form.
       - Manage the state of user input (email, username, password, first name, last name, address).
       - Handle the registration event (send a registration request to the server).
       - Validate and show error messages if necessary.
     -
     - **Child:**
       - **Register form component:** This is where users enter their registration details (email, username, password, first name, last name, address). It contains input fields and a submit button.

  c. **Dashboard Component:**
     - **Responsibilities**
     - **Child:**
       - **Post form component:** This component is used to create a new post. It displays the form, including input fields and the submit button.
       - **Menu component:** This component displays the user's navigation menu (Dashboard, Profile, Activity, Message). Users can manage the menu options and navigate to different pages in the app.
       - **Post component:** Displays user posts. Render a list of posts, each post showing title, content, image, and action buttons (like, comment, share).
       - **Search Component:** The search input field where users type their search query. Send a search request to the server and display the results like the posts or the users

  d. **Message Component:**
     - **Responsibilities:**
       - Manage and display user messages.
       - Provide the interface to send and receive messages.
       - Handle the sending of messages.
     - **Child:**
       - **Message Box Component:** Displays the message content. Render the sent and received messages, differentiating between sender and receiver.
       - **Message Input Component:** Provides the input field for users to send a new message. Manage the message content and send it to the Message Component.

e. **Activity Component:**
   - **Responsibilities:**
     - Display the user's activity (e.g., new posts, friend requests, events related to the user).

f. **User Component:**
   - **Responsibilities:**
     - Display detailed user information (profile, personal details, etc.).
     - Allow users to edit their personal information (e.g., change avatar, change password).

4.6 Component: Routers

- **Responsibilities:** The Router component in React is responsible for managing the navigation and rendering of components based on the URL. It acts as the bridge between the URL path and the components that should be displayed for that path.
- **Some main routes:**

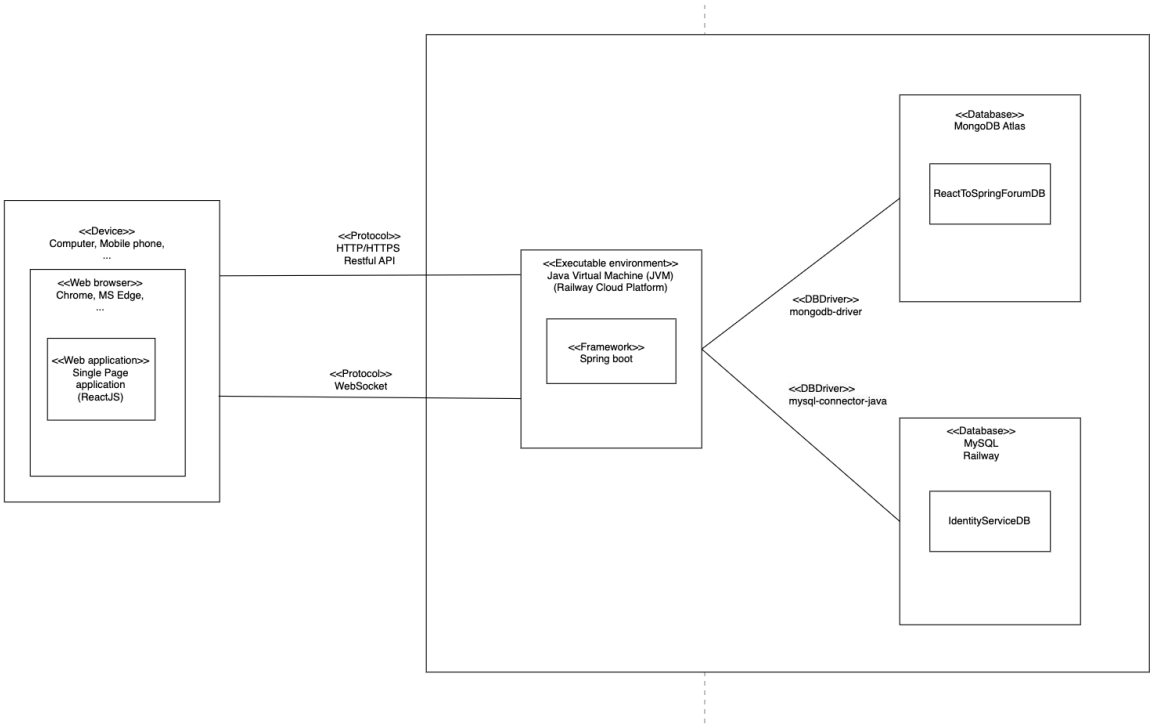
a. **/ (Home route):**
   - **Parent Component: LayoutDefault**
   - **Child Routes:**
     - /: Render the dashboard page, typically showing the list post.
     - /user: Render the user profile page, showing user details and preferences.
     - /activity: Render the activity page, displays user activity such as new posts, friend requests, and events related to the user

b. **/login:**
   - **Component:** LoginRegister
   - **Description:** Renders the login or registration page, allowing users to log into or create an account.

c. **/verification-success:**
   - **Component:** VerificationSuccess
   - **Description:** Displays a success message after a successful verification process (e.g., email confirmation).

d. **/verification-fail:**
   - **Component:** VerificationFailed
   - **Description:** Shows an error message when verification (e.g., email confirmation) fails.

5. Deployment



Deployment Description:

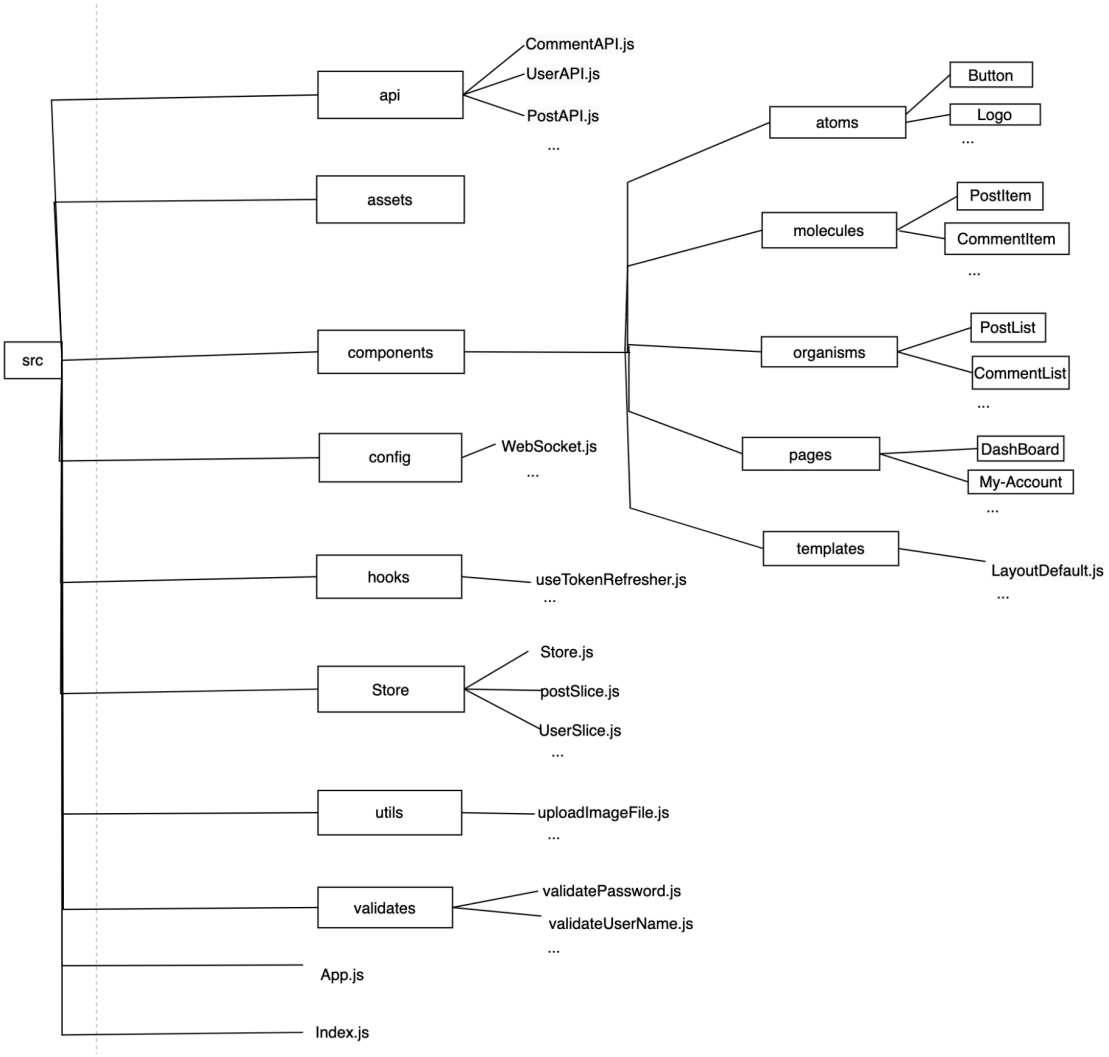
1. **Client Side**:
    - **Devices**: The web application can be accessed via devices such as computers or mobile phones.
    - **Web Browsers**: Users interact with the application using browsers like Chrome or Microsoft Edge.
    - **Web Application**: A single-page application built using **ReactJS**. Communication with the backend occurs through **RESTful APIs** (HTTP/HTTPS) and **WebSocket** protocols.
2. **Backend**:
    - **Execution Environment**: The backend application runs on the **Java Virtual Machine (JVM)**.
    - **Framework**: The backend is developed using **Spring Boot**, which provides RESTful API services and WebSocket communication.
3. **Database Layer**:
    - **MongoDB (Deployed on MongoDB Atlas)**: A NoSQL database, connected to the backend using the **mongodb-driver**.
    - **MySQL (Deployed on Railway)**: A relational database, connected to the backend using the **mysql-connector-java**.

The backend acts as a bridge between the frontend (ReactJS application) and the databases, handling requests, responses, and persistent data storage.

6. Implementation View

6.1 Front End

- Constructor directory



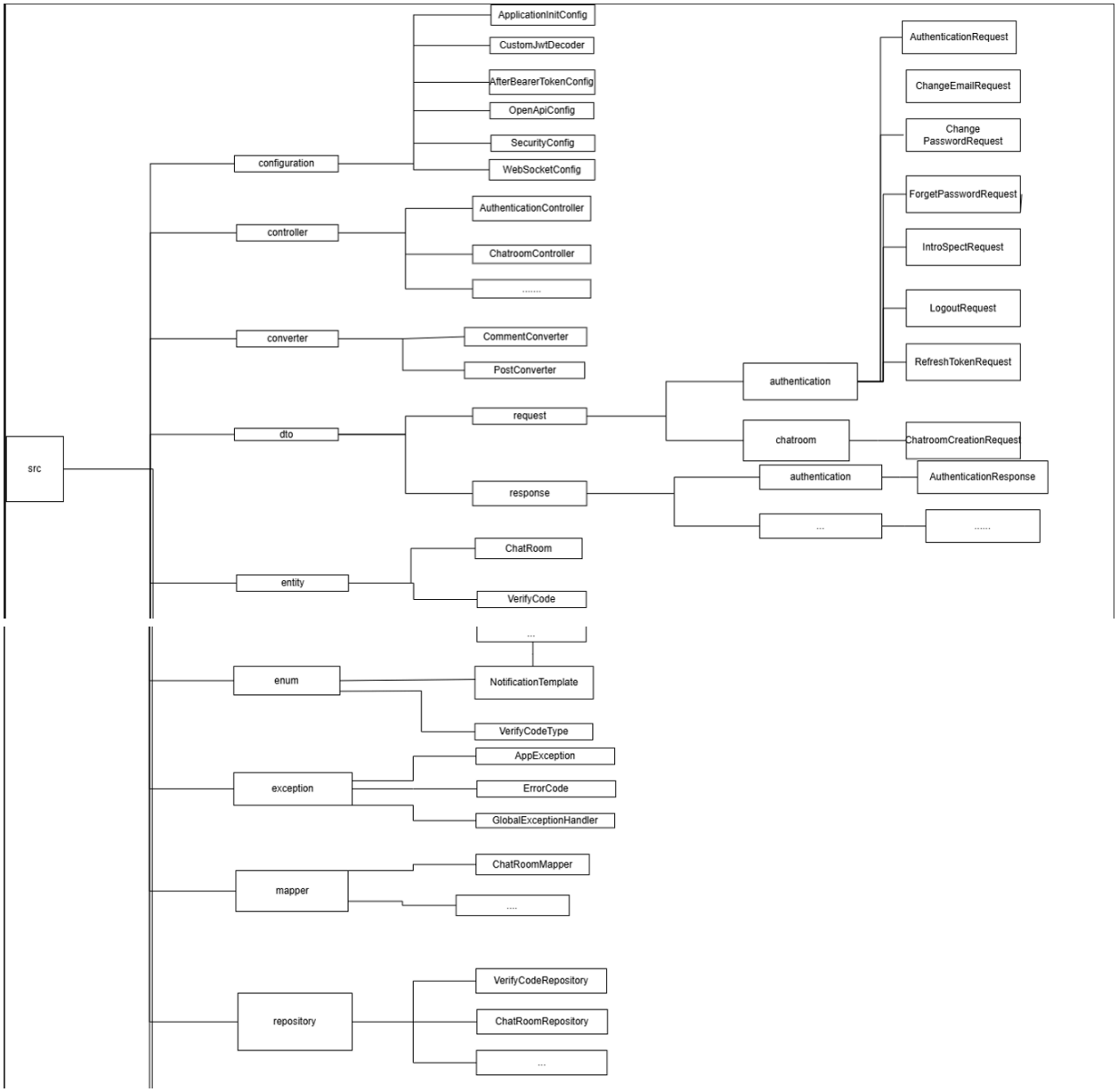
- Folder api: Defines files containing api processing functions to retrieve data from the BackEnd.
- Folder assets: Contains project resources such as logo, background…
- Folder components: This is the most important folder, where code snippets that define interfaces, operations, and event handling are contained. Inside is divided into 5 subfolders:
    - atoms: Defines small, reusable components in many places such as buttons, loading,...
    - molecules: Defines small components (larger than atoms) that are repetitive, such as a post, a comment. Molecules can be reused in atoms.
    - organisms: Defines components larger than molecules, such as post list, comment list,... Often reuse molecules.
    - pages: Defines the interface of the main pages, content can be taken from organisms.
    - templates: Defines default layouts.
- Folder config: Contains configuration files.
- Folder hooks: Contains self-defined hooks
- Folder store: Define slices to manage global state. Convenient for easily transferring data from one

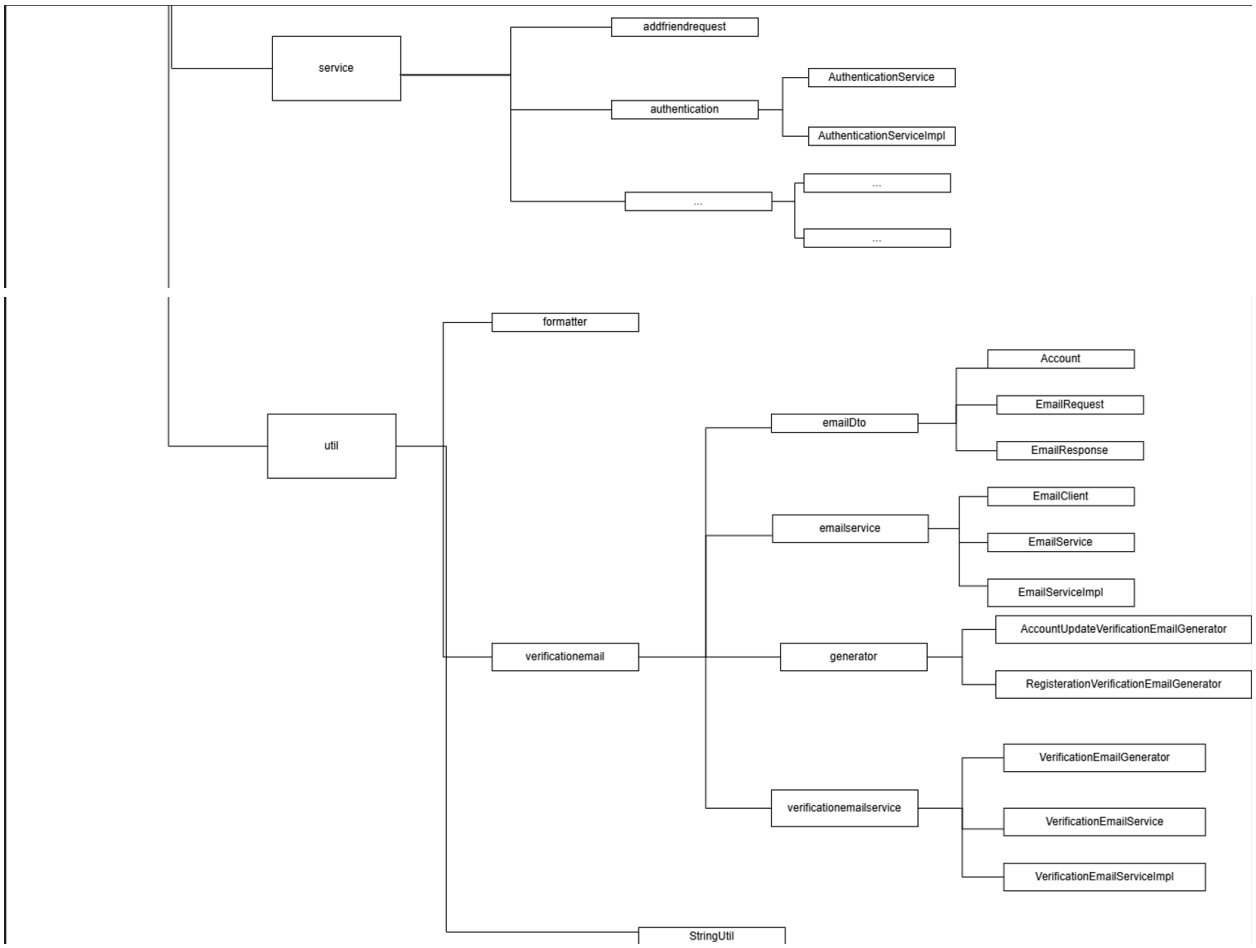
component to another

- Folder utils: contains reusable widgets in multiple places.
- Folder validates: contains functions used to check data..

6.2 Back End

- Constructor directory





- Folder configuration: Contains files to configure the application including application-specific settings such as database, authentication such as security, JWT, OpenAPI for API documentation and WebSocket for real-time communication.
- Folder controller: Contain files handling receiving requests from client and return the result to client.
- Folder converter: contains files that handle custom converter
- Folder dto : contains files that define data structure of request and response data.
- Folder entity: contains files that define data structure mapping to database
- Folder enums: Contains files defining enumerations, which are constant values for NotificationTemplate and VerifyType
- Folder mapper: Contains files responsible for mapping data from entity to response data and from request data to entity.
- Folder repository: Contains files that handle interactions with the database.
- Folder service: Contains files that implement business logic and coordinate operations, interacting with the database to return data to the controller.
- Folder util: Contains utility files to help business logic operations such as date formatting, string manipulation and sending verification emails.