

**Mini Social Media**

**Software Design Document**

**Project Code: Midia**

**Document Code: Midia– v1.0**

– Hanoi, June 2025 –

**Table of Contents**

[Chapter 1. Introduction 3](#_heading=h.9ti5nc7dh7ad)

[1.1 Purpose 3](#_heading=h.rbt1t8nfn4qh)

[1.2 Definitions, Acronyms and Abbreviations 3](#_heading=h.us5lu39synca)

[Chapter 2. System Architecture 4](#_heading=h.jrkhvans8bfu)

[2.1 System Overview 4](#_heading=h.b7tly5h4vjyw)

[2.3 Assumptions 4](#_heading=h.ytvqf6s17fbv)

[2.4 Design Constraints 5](#_heading=h.yty8jwmbqute)

[Chapter 3. Software Architecture Design 5](#_heading=h.n3n9uzl8l4vv)

[3.1 Architectural Representation 5](#_heading=h.y36w4bjrie57)

[3.2 Software Architecture 5](#_heading=h.va6oizlbxnvt)

[3.2.1 Process View 8](#_heading=h.iv2tdb4g1cf6)

[3.2.2 Logical View 10](#_heading=h.1qdd7z44tz6)

[3.2.3 Development View 10](#_heading=h.swib2hahcem6)

[3.2.4 Deployment View 12](#_heading=h.jgcau85olvye)

[Chapter 4. Detailed Component Design 14](#_heading=h.mlw66z1i50j4)

[4.1 Authentication & Authorization 14](#_heading=h.jg565dm9c6re)

[4.2 User Settings 14](#_heading=h.391se5dbsyyr)

[4.2.1 User Settings 14](#_heading=h.mmvv2f71dzax)

[4.2.1.2 Class/Component Design 17](#_heading=h.7f3hqrpv44kz)

[Chapter 5. Database Design 18](#_heading=h.u1htwwd1w28v)

[5.1 Database Design 18](#_heading=h.ke381dcyqmap)

[5.2 Database Detail 18](#_heading=h.ca05pcrjrd5x)

[5.2.1 User 18](#_heading=h.gsmhyad63s0c)

[5.2.2 UserProfile 19](#_heading=h.q75pm468yoiq)

[5.2.3 UserStats 19](#_heading=h.6e4o54d0bt8b)

[5.2.4 Post 19](#_heading=h.jtrubdrdb606)

[5.2.5 Comment 20](#_heading=h.s1qb6f9cb9r4)

[5.2.6 Attachment 20](#_heading=h.kqv1avkvj6l9)

[5.2.7 Role 20](#_heading=h.9fj7n0yc6391)

[5.2.8 Permission 20](#_heading=h.7wa70dapyw69)

# Chapter 1. Introduction

## 1.1 Purpose

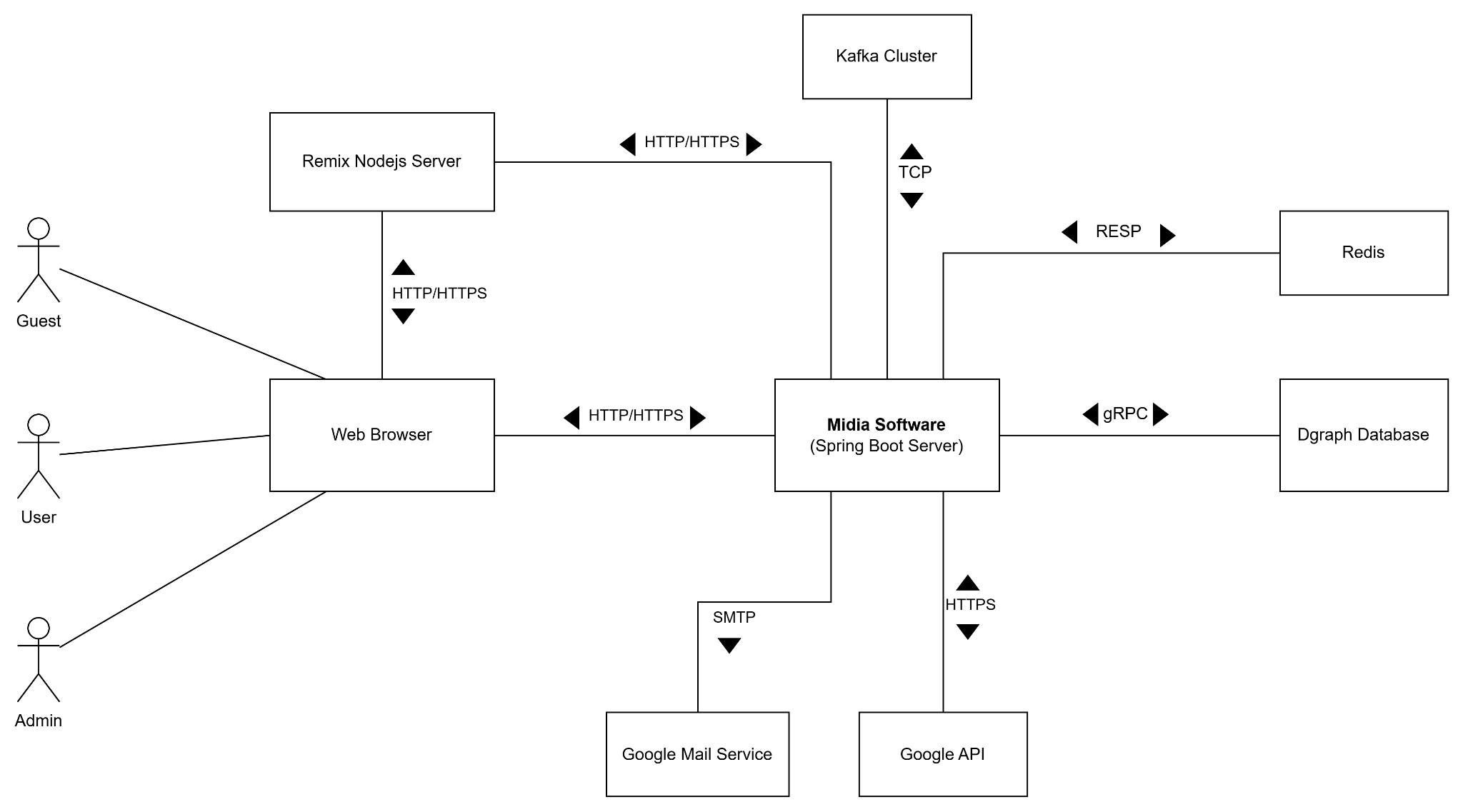
This document specifies the detailed software design for the Midia social‑media web application (React.js frontend, Spring Boot & Dgraph backend). It provides the architectural overview, module designs, data models, and component interactions required by the development team to implement Release 1.0 of the product.

## 1.2 Definitions, Acronyms and Abbreviations

| **Acronyms** | **Definition** |
| --- | --- |
| Midia | Mini Social Media Application |
| JWT | JSON Web Token |
| API | Application Programming Interface |
| SSR | Server-Side Rendering; a key feature of the Remix framework where web pages are rendered on the server instead of the browser. |
| DB | Database |
| Dgraph | A distributed graph database, serving as the database for Midia's interconnected social data. |
| Redis | An in-memory data store used for high-speed caching of frequently accessed data, like user sessions or popular posts. |
| Cassandra | A distributed NoSQL database used specifically for the high-throughput requirements of the messenger feature. |
| Kafka | A distributed event streaming platform used as a message broker to handle asynchronous tasks, like sending emails. |
| HTTP | Hypertext Transfer Protocol; the foundation of data communication for the World Wide Web, used for all API requests. |
| JSON | JavaScript Object Notation; the lightweight format used for data exchange between the Spring API and the Remix client. |
| OAuth | Open Authorization; an open standard for access delegation, used for authentication via the Google Authorization Server. |
| DM | Direct Message; a private chat feature common in social media, which the "messenger" component will handle. |

# Chapter 2. System Architecture

## 2.1 System Overview

**

**Figure 2-1**: System Architectural Overview

As described in the figure 2-1, the system will have the following components

* **Midia Software**: this is a developing software which will be built on the Spring MVC with Netflix DGS framework and deployed as a cloud native Docker container.
* **Remix Nodejs Server:** this is a Nodejs server that will be used for server-side rendering of React applications and communicating with Midia Software and deployed as a cloud native Docker container.
* **Google API:** this is the collection of apis provided by Google Cloud that the Midia software shall integrate with in order to perform some functionalities
* **Redis:** this is the Redis in-memory noSQL key-value store that will be used for caching data.
* **Dgraph Database:** this is the main noSQL graph database of the Midia software.
* **Kafka Cluster:** this is the cluster of Kafka servers that Midia software will use for performing some asynchronous tasks.
* **Google Mail Service**: This is a mail server that the Midia software shall integrate with in order to authenticate the user and send email notification.

## 2.3 Assumptions

Target browsers: Chrome, Firefox, Edge (latest versions).

Stable internet connection ≥ 10 Mbps.

## 2.4 Design Constraints

* All client–server communication over HTTPS.
* Response time ≤ 2s per API call.
* Image preview ≤ 1 s latency.

# Chapter 3. Software Architecture Design

## 3.1 Architectural Representation

The Midia system follows the Clean Layered Architecture, which separates concerns into multiple layers to improve maintainability and scalability. The architecture is described using the following views:

**Use-Case View**

* Actors: Guest, User, Admin.
* Use cases: Register, Login, View Feed, Post, Like, Comment, Follow, Manage Account, ...
* Represented with Use Case Diagrams.

**Logical View**

* Layers:
  + **Domain**: Business models (User, Post, Comment).
  + **Application**: Use case logic (e.g., FollowUser, CreatePost).
  + **Interface**: Exposes controllers and endpoints.
  + **Infrastructure**: Contains configs, DB access and external service
* Represented with Class and Package Diagrams.

**Process View**

* Sequence: API Controller -> Business Logic -> Repository -> DB
* Represented with Sequence and Activity Diagrams.

**Deployment View**

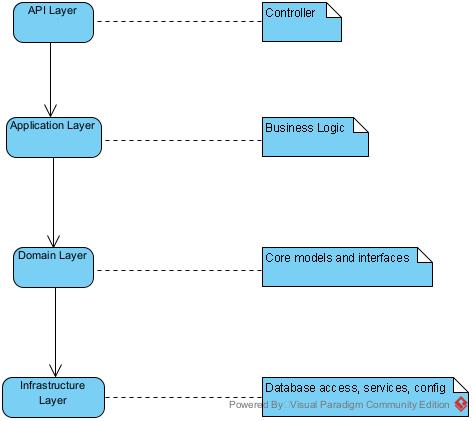
* Components: Frontend (ReactJS), Backend (Spring Boot), Database (PostgreSQL + DGraph),.
* Represented with Deployment Diagrams.

**Implementation View**

* Folders: /domain, /application, /business, /infrastructure.
* Represented with Component Diagrams or directory structure.

## 3.2 Software Architecture

The **Midia** system adopts the **Clean Layered Architecture Pattern**, which emphasizes separation of concerns, dependency inversion, and clear boundaries between layers.



**1. API Layer (Presentation Layer)**

* **Packages**: api
* **Responsibilities**:
  + REST API: Exposes traditional HTTP endpoints via controllers.
  + GraphQL API: Exposes flexible queries/mutations via resolvers.
  + Both interact with the same service layer.

#### **2. Application Layer (Use Case Layer)**

* **Package**: business
* **Responsibilities:**
  + Contains reusable business logic and orchestrates use cases.
  + Called by both REST and GraphQL APIs.

#### **3. Domain Layer**

* Package: domain
* **Responsibilities:**
  + Encapsulates business rules and core data structures.
  + Contains entity classes (e.g.,User, Post) and repository interfaces.

#### **4. Infrastructure Layer**

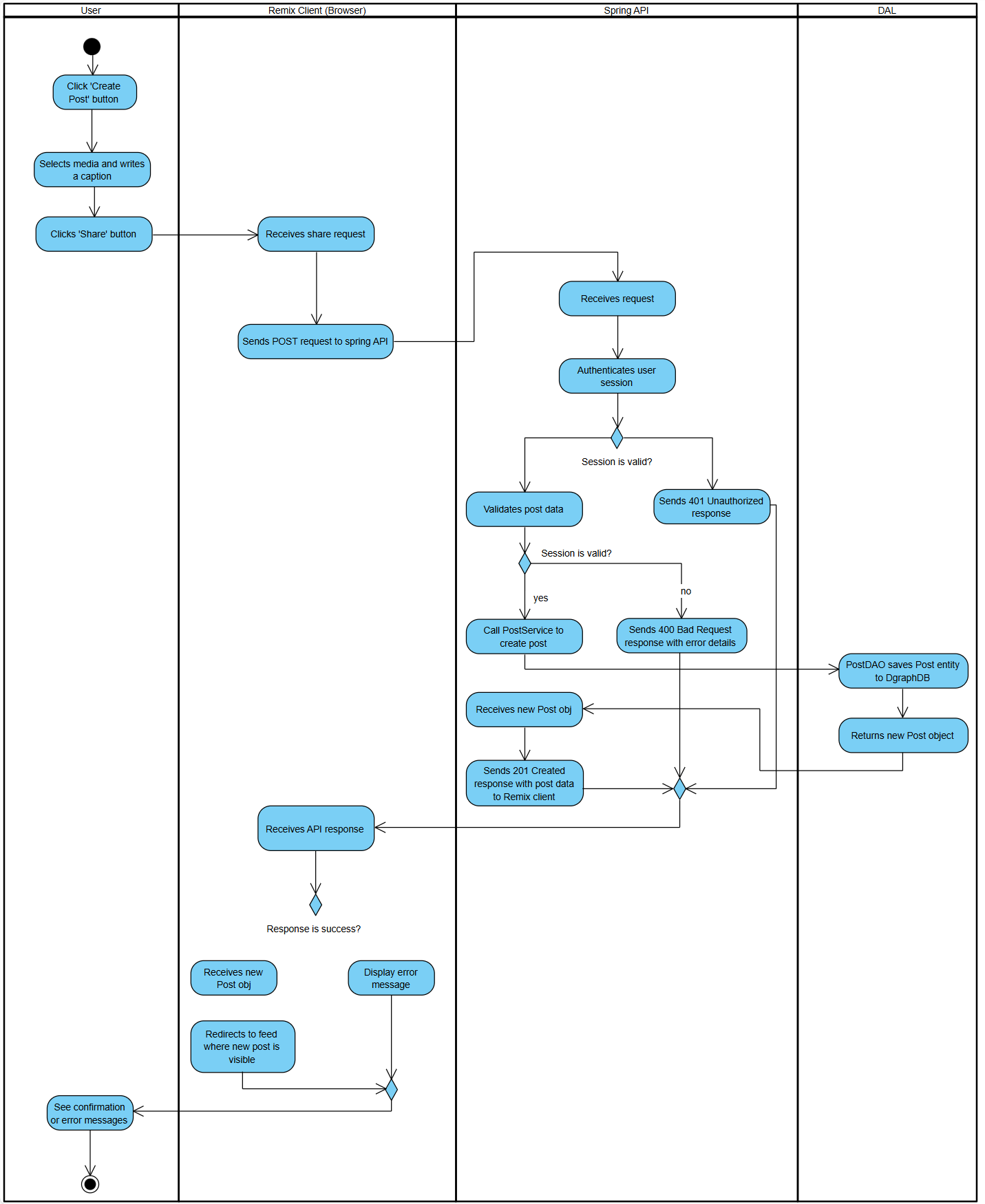
* **Package**: infrastructure
* **Responsibilities:**
  + Implements repository interfaces.
  + Provides database access (e.g., Dgraph), external APIs, configurations.

### 

### 3.2.1 Process View

##### 3.2.1.1 Create Post

##### 3.2.1.2 Create Post Process

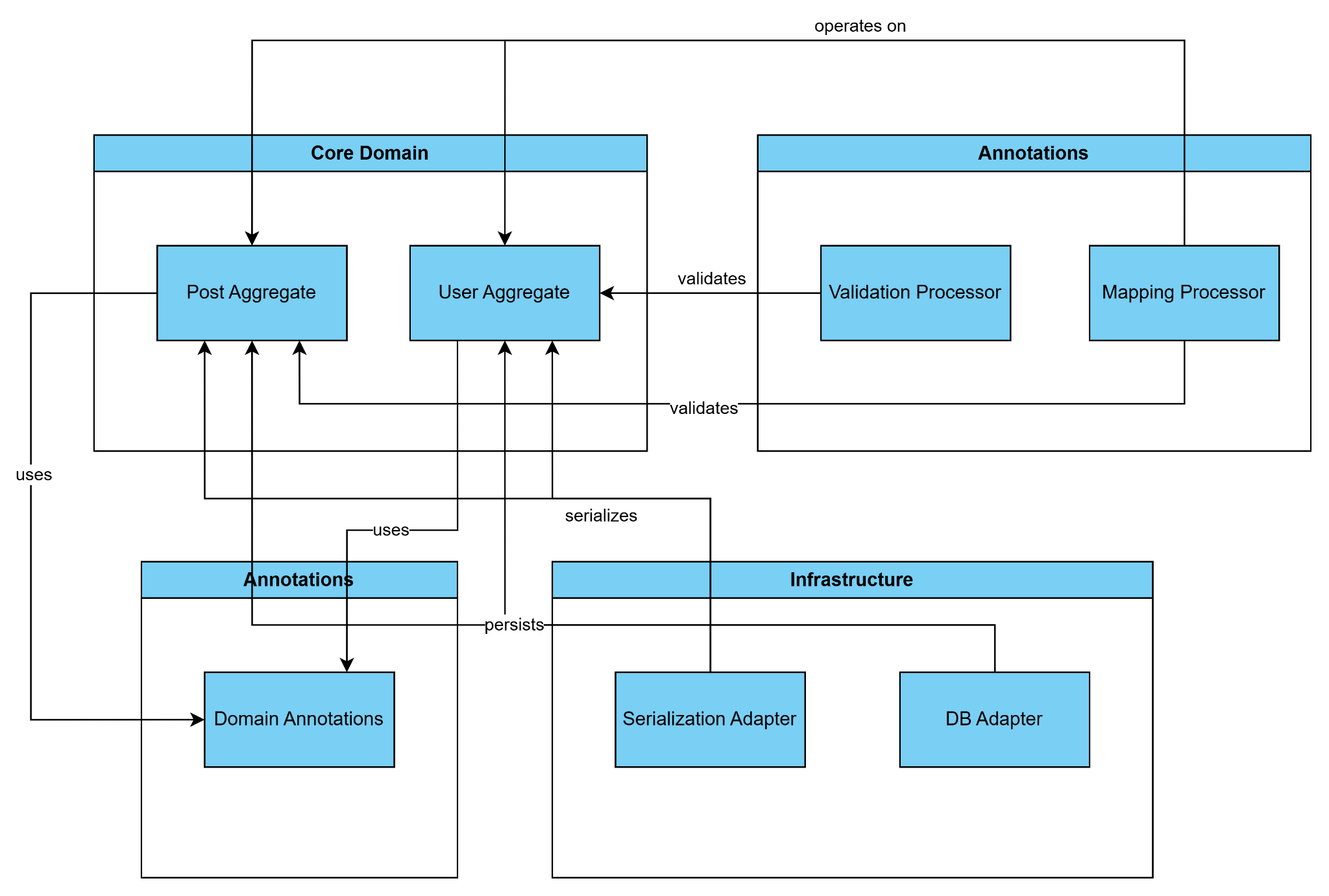


**Figure 3-3**: Create Post process

As described in Figure 3-3, the "Create Post" process will have the following steps:

* **Step 1:** A logged-in **User** initiates this process by navigating to the "Create Post" screen within the Midia application running on their browser (**Remix Client**).
* **Step 2:** The user selects media, writes a caption for the post, and clicks the "Share" button.
* **Step 3:** The **Remix Client** component receives this action. It packages the user's session token, the post caption, and the media file into a multipart/form-data HTTP POST request and sends it to the /api/posts endpoint on the **Spring API**.
* **Step 4:** The **Spring API** receives the request. It first validates the user's session token to ensure they are authenticated.
  + If the session is invalid, it returns a 401 Unauthorized error, and the process ends.
  + If the session is valid, the API proceeds to validate the post's content (e.g., checking for prohibited content, caption length).
* **Step 5:** If the post data is valid, the PostController in the Spring API calls the createPost method on the PostService.
* **Step 6:** The PostService component orchestrates the business logic. It invokes the PostDAO (Data Access Object) to save the new post's information into the **Dgraph Database**.
* **Step 7:** After the database confirms the successful creation of the post record, the PostDAO returns the newly created Post object to the PostService.
* **Step 8:** The **Spring API** constructs a 201 Created HTTP response, including the data of the new post in JSON format, and sends it back to the **Remix Client**. (Optionally, at this stage, the API could also publish a "NewPostCreated" event to **Kafka** to trigger asynchronous notifications for the user's followers).
* **Step 9:** The **Remix Client** receives the success response. It displays a confirmation message to the user (e.g., "Your post has been shared") and updates the UI, typically by redirecting the user to their feed or profile where the new post is now visible.

### 3.2.2 Logical View

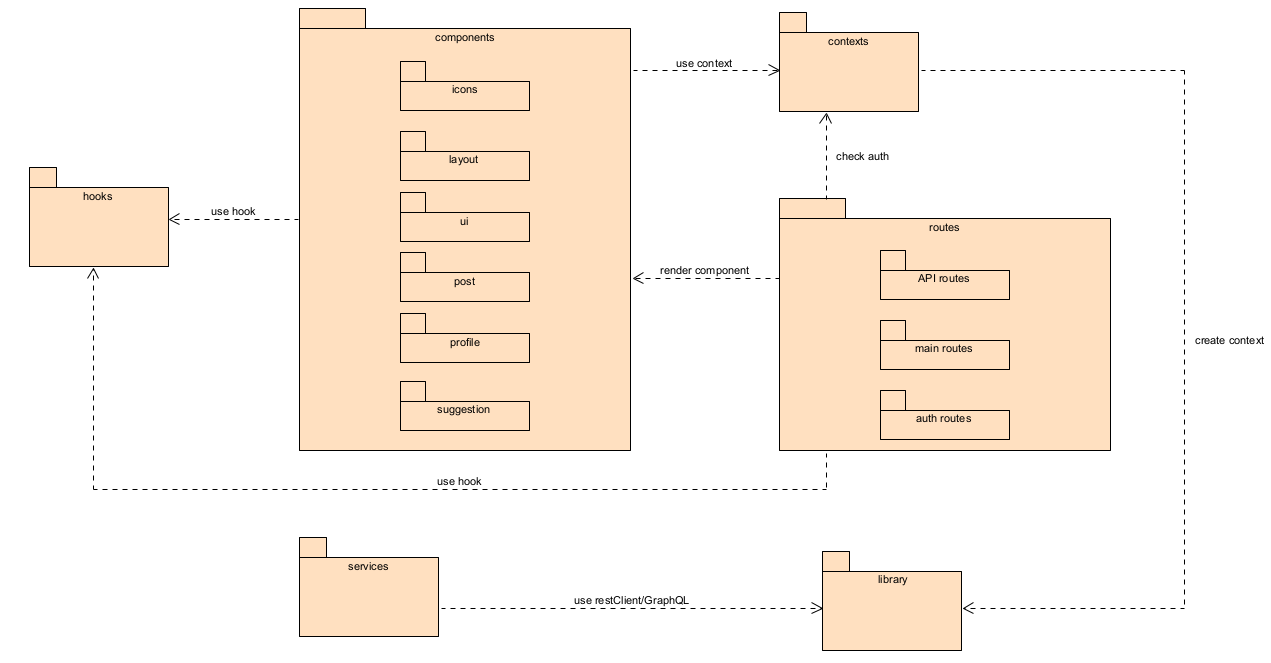


**Figure 3-1**: FAP Logical Overview

The logical view diagram shows the main subsystems of the application: Core Domain, Application Logic, Infrastructure, and Annotations. Core Domain contains business entities like User and Post aggregates. Application Logic includes processors that use and process domain objects to implement business rules. Infrastructure provides adapters for database and serialization, handling technical concerns outside the core logic. Annotations are used by domain objects to enrich their metadata.

### 3.2.3 Development View

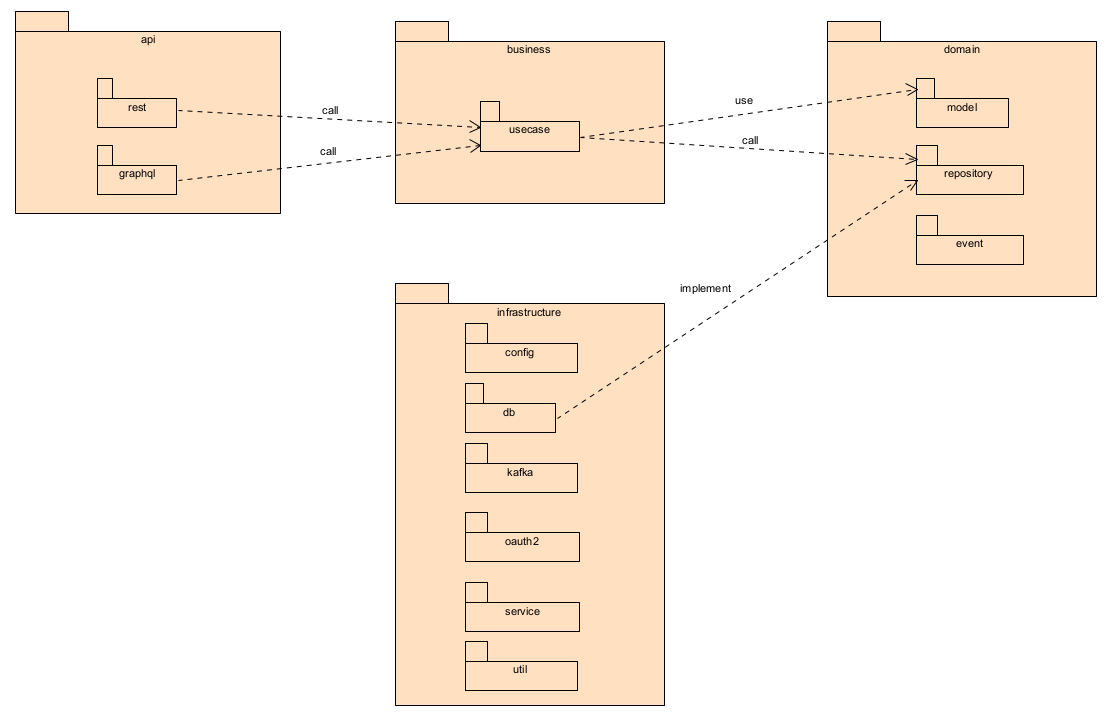
#### 3.2.3.1 Package Diagram



**Figure 3-1:** Midia Frontend Package Diagram

***Packages Description***

| **No** | **Package** | **Description** |
| --- | --- | --- |
| 1 | components | Contains UI components such as layout, post, profile, suggestions, and icons |
| 2 | hooks | Custom React hooks used to share logic between components or routes. |
| 3 | routes | Defines the application's routing paths (main, auth, API). |
| 4 | contexts | Provides and shares global state using React Context API. |
| 5 | services | Handles tasks such as API calls, backend interactions, or supporting logic. |
| 6 | library | Common utility libraries like helpers, validators, constants, etc. |



**Figure 3-2:** Midia Backend Package Diagram

***Packages Description***

| **No** | **Package** | **Description** |
| --- | --- | --- |
| 1 | rest | Exposes RESTful HTTP endpoints for client interaction |
| 2 | graphql | Handles GraphQL requests and schema definitions. |
| 3 | usecase | Contains application-specific business logic (use cases). |
| 4 | model | Defines core business entities. |
| 5 | repository | Defines interfaces (contracts) for data access |
| 6 | db | Implements data access logic using databases or external storage |
| 7 | service | Provides external services such as email sending, file storage, or third-party integrations. |
| 8 | config | Contains application configuration files and setup classes |
| 9 | kafka | Manages Kafka integration for asynchronous communication and event streaming. |
| 10 | oauth2 | Handles OAuth 2.0 authentication and authorization flows. |
| 11 | util | Provides utility classes and helper methods shared across the backend. |

### 3.2.4 Deployment View

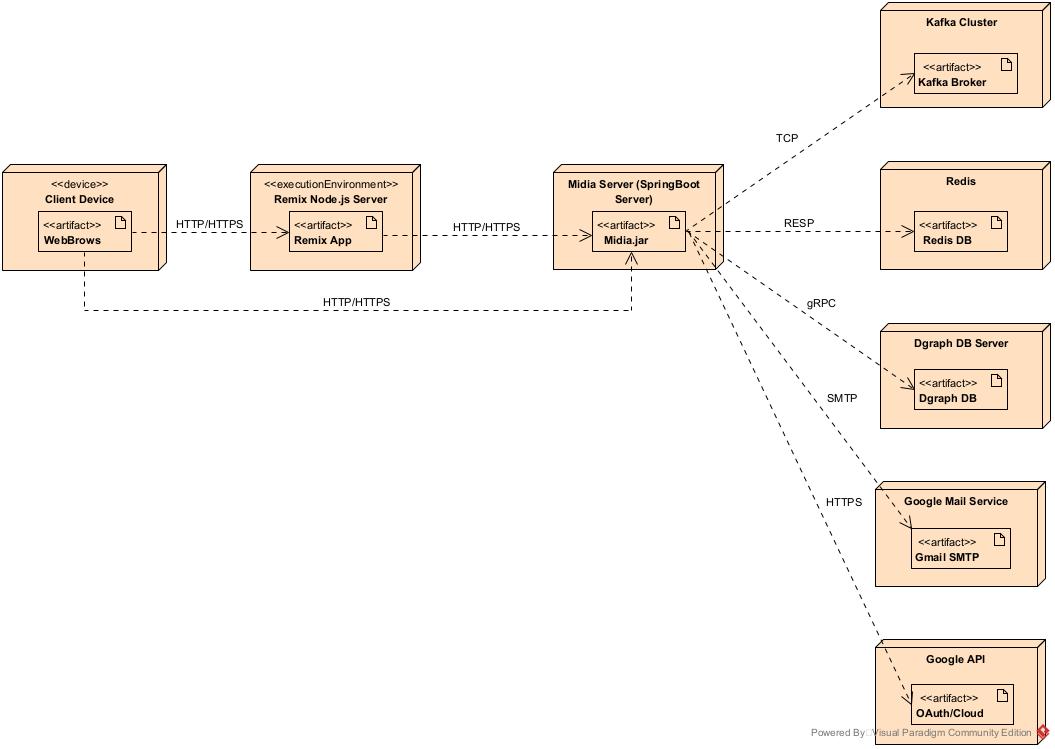
**

Figure 3-4: FAP Deployment structure

***Configuration Description***

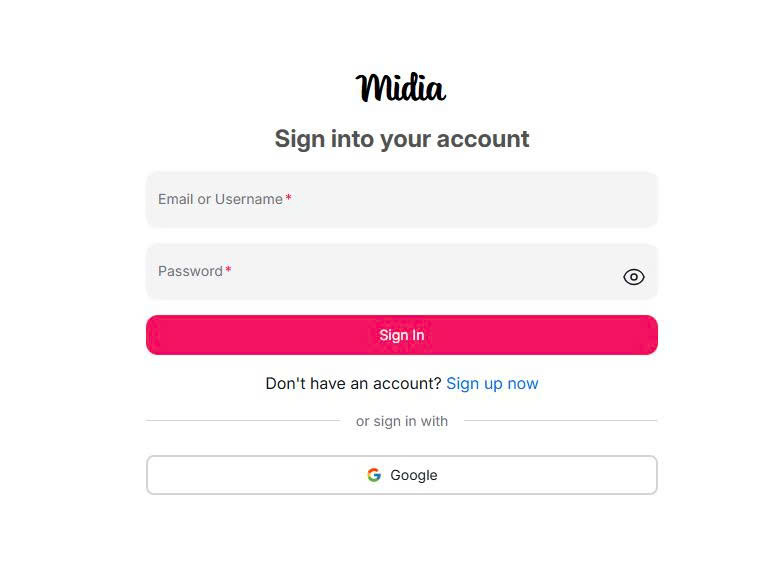
| **No** | **Component/Node** | **Description** |
| --- | --- | --- |
| 01 | Client Device | Represents the end-user device (e.g., computer, smartphone). The user interacts with the application through a web browser, sending HTTP/HTTPS requests to the Remix App. |
| 02 | Remix Node.js Server | Hosts the Remix frontend application on a Node.js server. It receives client requests and renders pages or API responses. It forwards certain backend API requests to the Midia Spring Boot server. |
| 03 | SpringBoot Server | Acts as the core backend service. It handles business logic, processes API requests, and communicates with multiple infrastructure services |
| 04 | Kafka Cluster | Facilitates asynchronous messaging and event-driven architecture by acting as a distributed message broker. Communicates with the Midia server via TCP. |
| 05 | Redis | Provides fast in-memory key-value storage, typically used for caching, session storage, or rate limiting. Communicates with the Midia server using the RESP protocol. |
| 06 | Dgraph DB Server | A distributed graph database used to store and query relational data efficiently. The Midia server interacts with it via gRPC. |
| 07 | Google Mail Service | Enables the system to send emails (e.g., user verification, notifications) using the SMTP protocol through Google Mail. |
| 08 | Google API | Provides integration with Google’s OAuth for user authentication and access to various Google Cloud services. Midia server communicates via HTTPS. |

# Chapter 4. Detailed Component Design

## 4.1 Authentication & Authorization

### 4.1.1 Login

#### 4.1.1.1 Screen Design



**Figure 4-1.1**: Screen Design of Follow Suggestions

**Table 4-1.1: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | Username | Label | Yes | N/A | Display usernames of accounts. |
| 2 | Password | Label | Yes | N/A | Display password of accounts. |
| 3 | Sign In | Button | No | N/A |  |
| 4 | Sign up now | Link | No | N/A | Navigate to sign up page. |
| 5 | Google | Button | No | N/A | Sign in via google. |

#### 

#### 4.1.1.2 Class Design

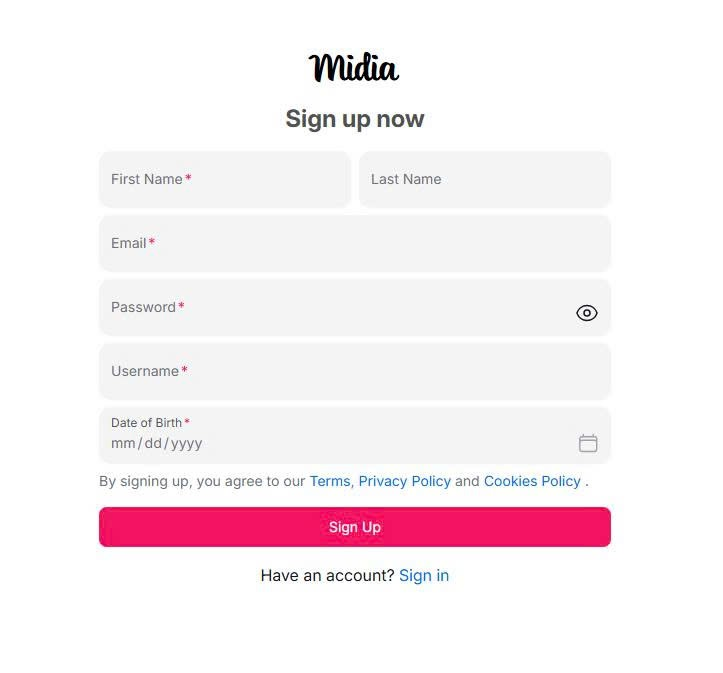
##### 4.1.1.2.1 Classes/Component Structure

##### 4.1.1.2.2 Classes/Component Description

#### 4.1.1.3 Object Interactions

### 4.1.2 Register

#### 4.1.2.1 Screen Design



**Figure 4.1.2**: Screen Design of Register

**Table 4.1.2: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | First Name | Input | Yes | 30 | Enter user’s first name |
| 2 | Last Name | Input | No | 30 | Enter user’s last name |
| 3 | Email | Input | Yes | 30 | Enter user’s email |
| 4 | Password | Input | Yes | 30 | Enter user’s password |
| 5 | Username | Input | Yes | 20 | Enter user’s username |
| 6 | Date of birth | Date | Yes | N/A | Choose user’s date of birth in calendar |
| 7 | Sign up | Button | Yes | N/A | Submit button to create a new account |
| 8 | Sign in | Button | No | N/A | Navigate user to login page |

#### 4.1.2.2 Class Design

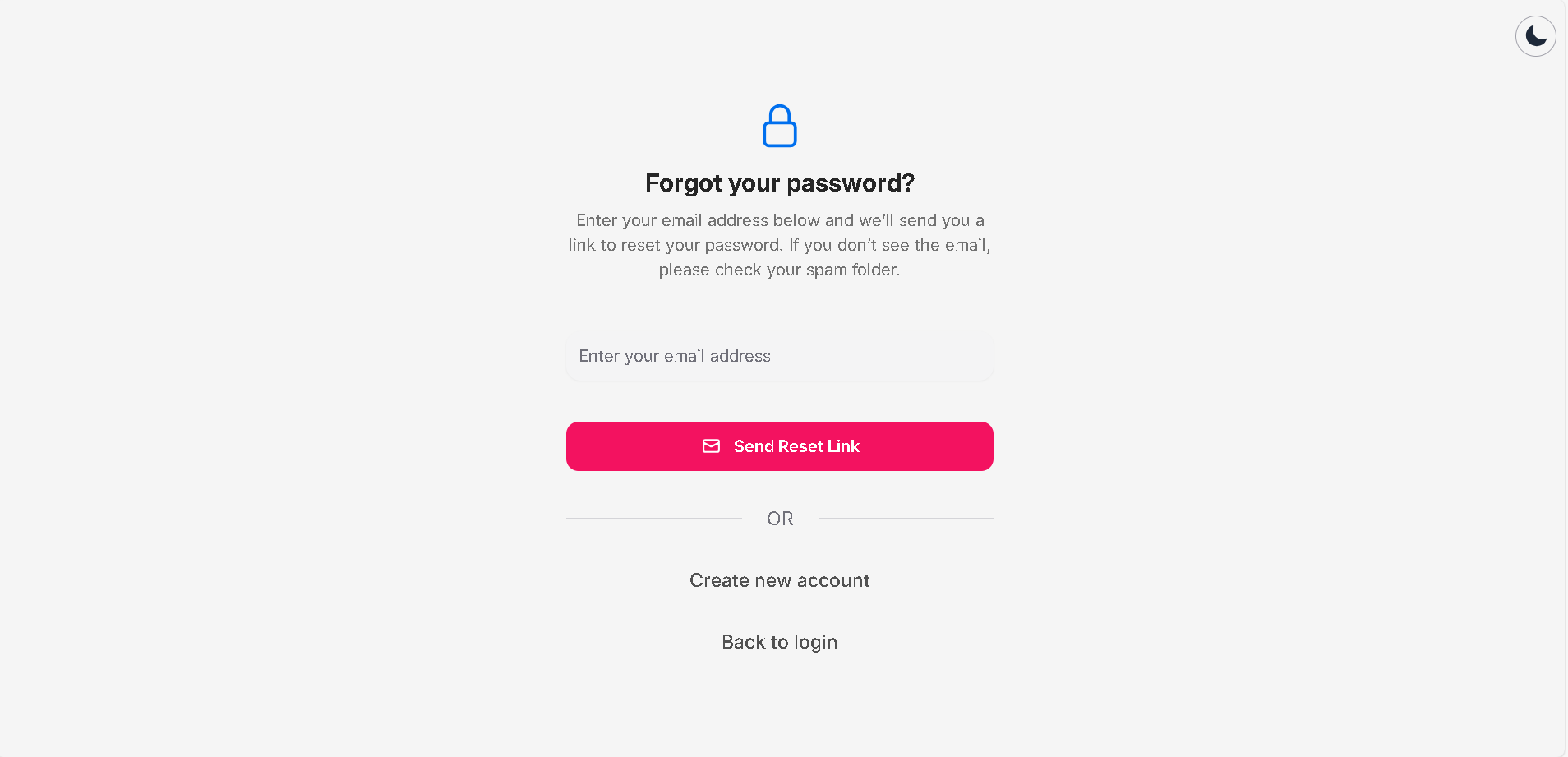
##### 4.1.2.2.1 Classes/Component Structure

##### 4.1.2.2.2 Classes/Component Description

#### 4.1.2.3 Object Interactions

### 4.1.3 Forgot Password

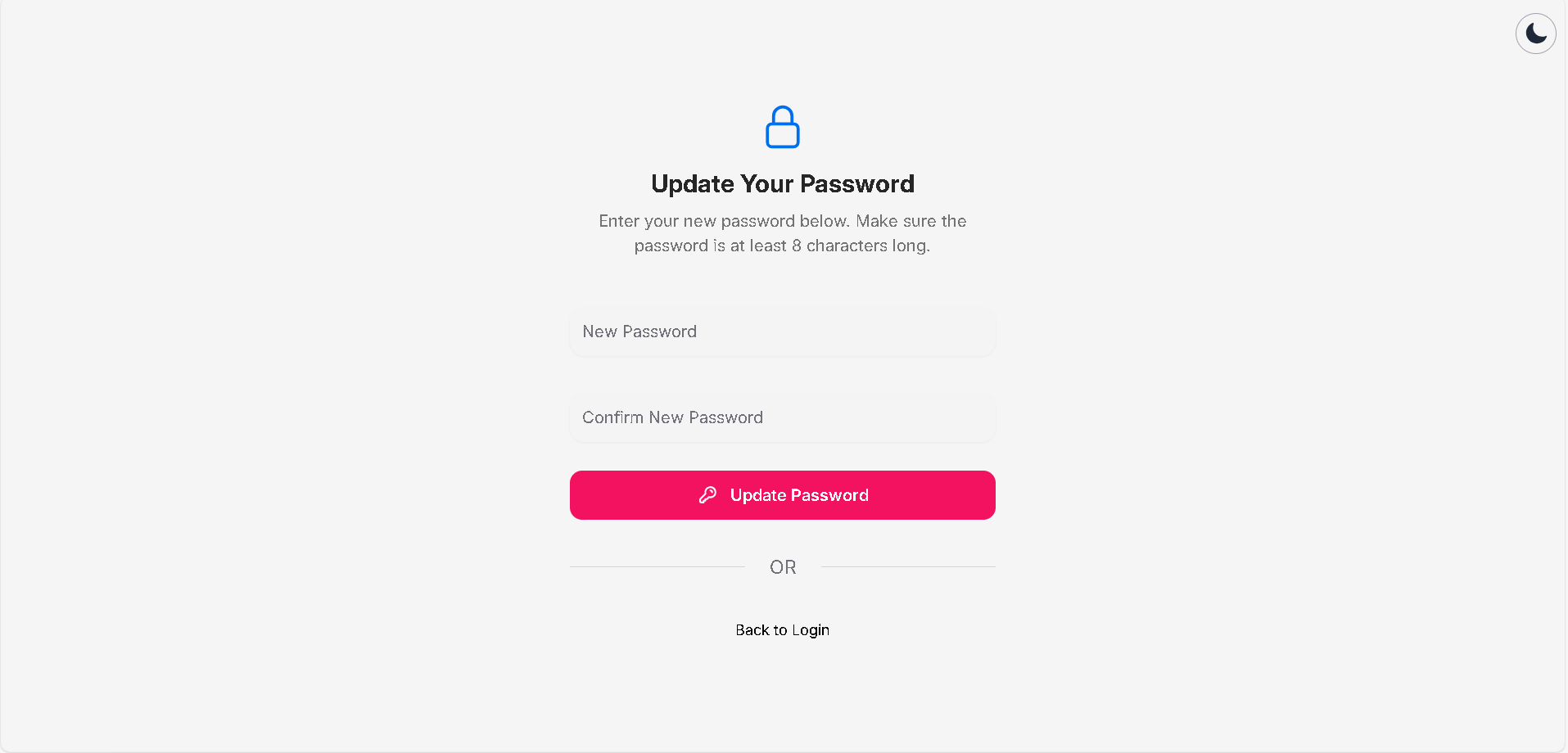
#### 4.1.3.1 Screen Design



**Figure 4.1.3-1**: Screen Design of forgot password

**Table 4.1.3: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | Theme Switcher | Button | No | N/A | Switch the theme of the system |
| 2 | Email address | Input | Yes | 30 | Enter user’s email |
| 3 | Send Reset Link | Button | Yes | N/A | Confirm email and send reset password URL |
| 4 | Create new account | Link | No | N/A | DIrect user to register page |
| 5 | Back to login | Link | No | N/A | DIrect user to login page |

****

**Figure 4.1.3-2**: Screen Design of update password

**Table 4.1.3-2: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | Theme Switcher | Button | No | N/A | Switch the theme of the system |
| 2 | New password | Input | Yes | 8 | Enter user's new password |
| 3 | Confirm new password | Input | Yes | 8 | Enter user's confirm new passwordL |
| 4 | Update password | Button | Yes | N/A | Confirm user to update the new password |
| 5 | Back to login | Link | No | N/A | DIrect user to login page |

#### 

#### 4.1.3.2 Class Design

##### **4.**1**.**3**.2.1 Classes/Component Structure**

##### 4.1.3.2.2 Classes/Component Description

#### 4.1.3.3 Object Interactions

## 4.2 User Settings

### 4.2.1 User Settings

#### 4.2.1.1 Screen Design

##### 

**Figure 4-2.1**: Screen Design of User Settings

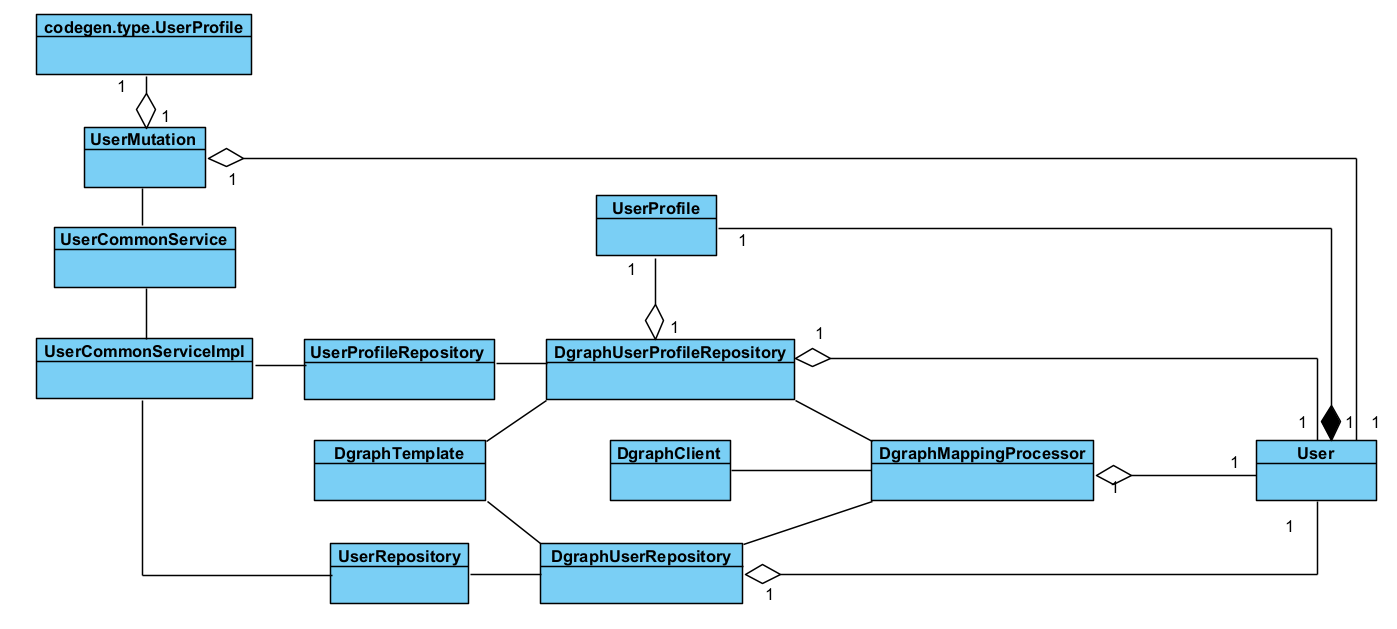
**Table 4-2.1:** **User Settings Definition**

| **No** | **Object/Control Name** | **Type** | **Required** | **Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | fullName | String | Yes | 100 | Display real name of user who has logged in |
| 2 | username | String | Yes | 100 | Display username of account which has logged in |
| 3 | website | String | No |  | Display users’ website link. This field can be edited if users are using a mobile app and optional |
| 4 | bio | String | No |  | Display short biography of users. Users can only type about 150 characters. This field is optional |
| 5 | email | String | Yes |  | Display user’s email. Email must follow the format <name>@<domain>. This field cannot be edited |
| 6 | phoneNumber | String | Yes | 11 | Display user’s phone number. Phone number must contain 11 digits. |
| 7 | gender | String | Yes |  | Display the user's gender. This field contains Male, Female and Prefer not to say |
| 8 | suggestion | Boolean | No |  | Allow users to see similar account suggestions on their profiles |
| 9 | learnMore | Link | Yes |  | Display the link that navigates to the information about changing username policies |
| 10 | changeProfilePhoto | URL | Yes |  | Display the link that navigates to change profile photo screen |
| 11 | submit | Button | Yes |  | Allow to save all changes |
| 12 | deactivateAccount | Boolean | Yes |  | Allow users to deactivate their account if they want |
| 13 | editProfile | Button | Yes |  | Allow users to open the edit profile box. They can change their information stored in the system |

#### 

#### 4.2.1.2 Class Design

##### 4.2.1.2.1 Classes/Component Structure



##### 4.2.1.2.2 Classes/Component Description

###### **4.2.1.2.1 UserMutation Class**

| Class | UserMutation | | | |
| --- | --- | --- | --- | --- |
| **Description** | A GraphQL mutation component for user-related operations such as registration, profile editing, and follow toggling. | | | |
| **Base Class** | None (uses annotations to define behavior) | | | |
| **Constructor** | @RequiredArgsConstructor` (Lombok auto-generates a constructor with all final fields) | | | |
| **Prototype** | @DgsComponent` class with GraphQL mutation methods | | | |
| **Source File** | UserMutation.java | | | |
| **Namespace** | Package: as defined in your project: `dev.huyhoangg.midia.api.mutation` | | | |
| **Attributes** | Name | Type | Description | |
| userCommonService | UserCommonService | Handles core user operations such as registration, profile editing, and fetching user info. | |
| userInteractionService | UserInteractionService | Handles user interaction logic like follow/unfollow. | |
| **Methods** | Name | Input | Output | Description |
| registerUser | RegisterUserInput input | RegisterUserResp | Registers a new user using the provided input data. |
| editUserProfile | EditUserInput input | UserProfile | Updates both user profile and account information. Requires authentication. |
| toggleFollow | String targetUserId | Boolean | Toggles follow/unfollow action for the current user on the target user. Requires authentication. |

##### 

###### **4.2.1.2.2 UserCommonService Class**

###### 

| Class | UserCommonService | | | |
| --- | --- | --- | --- | --- |
| **Description** | Interface that defines user-related business logic operations. | | | |
| **Base Class** | None | | | |
| **Constructor** | Not applicable (interface) | | | |
| **Prototype** | public interface UserCommonService | | | |
| **Source File** | UserCommonService.java | | | |
| **Namespace** | dev.huyhoangg.midia.business.user | | | |
| **Attributes** | Name | Type | Description | |
|  |  | No attributes in interface | |
| **Methods** | Name | Input | Output | Description |
| registerUser | RegisterUserInput | RegisterUserResp | Registers a new user. |
| getUserProfile | String userId | UserProfile | Fetches profile for the given user ID. |
| getMyInfo | None | User | Retrieves information about the currently authenticated user. |
| getCurrentUserUid | None | String | Retrieves current user's UID from the security context. |
| verifyEmail | String token, String id | boolean | Verifies user's email via token. |
| getUserById | String id | User | Retrieves user by ID. |
| getUserByUsername | String username | User | Retrieves user by username. |
| getUserByEmail | String email | User | Retrieves user by email. |
| getUserByPost | Post post | User | Retrieves user by associated post. |
| searchUserByKeyword | String kw | List<User> | Searches users by keyword. |
| editUserProfile | String userId, UserProfile profile | UserProfile | Updates user profile data. |
| editUserInformation | String id, String username, String email | User | Updates username/email information. |

##### 

###### **4.2.1.2.3 UserCommonServiceImpl Class**

| Class | UserCommonServiceImpl | | | |
| --- | --- | --- | --- | --- |
| **Description** | Implementation of `UserCommonService` providing full business logic. | | | |
| **Base Class** | Implements `UserCommonService` | | | |
| **Constructor** | `@RequiredArgsConstructor` (auto-generated via Lombok) | | | |
| **Prototype** | `@Service public class UserCommonServiceImpl implements UserCommonService` | | | |
| **Source File** | UserCommonServiceImpl.java | | | |
| **Namespace** | dev.huyhoangg.midia.business.user | | | |
| **Attributes** | Name | Type | Description | |
| userRepository | UserRepository | Handles CRUD operations on user entities. | |
| passwordEncoder | PasswordEncoder | Encodes and verifies passwords. | |
|  | roleRepository | RoleRepository | Fetches user role data. | |
|  | userEventProducer | UserEventProducer | Publishes events such as email verification. | |
|  | redisTemplate | RedisTemplate<String, String> | Manages Redis caching for verification tokens. | |
|  | userMapper | UserMapper | Maps between domain and GraphQL types. | |
|  | upRepo | UserProfileRepository | Repository for profile data. | |
|  | suRepo | SearchUserRepository | Search-related user repository. | |
| **Methods** | Name | Input | Output | Description |
| registerUser | RegisterUserInput | RegisterUserResp | Validates and registers a new user, raises events. |
| getUserProfile | String userId | UserProfile | Retrieves profile from domain model and maps to GraphQL type. |
| getMyInfo | None | User | Authenticates and fetches user data for current context. |
| verifyEmail | String token, String id | boolean | Validates email token and updates user verification status. |
| getUserById | String id | User | Looks up user by ID. |
| getUserByUsername | String username | User | Looks up user by username. |
| getUserByEmail | String email | User | Looks up user by email. |
| getUserByPost | Post post | User | Finds user associated with a post. |
| getCurrentUserUid | None | String | Retrieves UID of currently authenticated user. |
| searchUserByKeyword | String kw | List<User> | Performs keyword-based user search. |
| editUserProfile | String userId, UserProfile profile | UserProfile | Updates and maps user profile. |
| editUserInformation | String userId, String username, String email | User | Updates user's login credentials. |

###### 

###### **4.2.1.2.4 UserRepository Class**

| Class | UserRepository | | | |
| --- | --- | --- | --- | --- |
| **Description** | Interface for CRUD and custom query operations on `User` domain model. | | | |
| **Base Class** | None | | | |
| **Constructor** | Not applicable | | | |
| **Prototype** | public interface UserRepository | | | |
| **Source File** | UserRepository.java | | | |
| **Namespace** | dev.huyhoangg.midia.domain.repository.user. | | | |
| **Attributes** | Name | Type | Description | |
|  |  | No attributes in interface | |
| **Methods** | Name | Input | Output | Description |
| existsByEmail | String email | boolean | Checks if email already exists. |
| existsByUsername | String username | boolean | Checks if username already exists. |
| findById | String id | Optional<User> | Finds a user by ID. |
| findByEmail | String email | Optional<User> | Finds a user by email. |
| findByUsername | String username | Optional<User> | Finds a user by username. |
| findUserByUid | String uid | Optional<User> | Finds user by UID. |
| save | User user | User | Persists user to the database. |
| findSocialAccountsByUserUid | String uid | Collection<SocialAccount> | Retrieves social accounts linked to a UID. |
| findByPostId | String postId | Optional<User> | Finds a user by post ID. |

###### **4.2.1.2.5 DgraphUserRepository Class**

| Class | DgraphUserRepository | | | |
| --- | --- | --- | --- | --- |
| **Description** | Repository implementation for managing `User` and `SocialAccount` data in Dgraph using high-level query building, mutation handling, and JSON mapping. | | | |
| **Base Class** | dev.huyhoangg.midia.domain.repository.user.UserRepository.java | | | |
| **Constructor** | DgraphUserRepository(DgraphMappingProcessor, DgraphTemplate, ObjectMapper, UserMapper) | | | |
| **Prototype** | @Repository`, `@Slf4j`, `@RequiredArgsConstructor | | | |
| **Source File** | DgraphUserRepository.java | | | |
| **Namespace** | dev.huyhoangg.midia.infrastructure.db.persistence | | | |
| **Attributes** | Name | Type | Description | |
| mappingProcessor | DgraphMappingProcessor | Converts Java objects to/from Dgraph-compatible JSON. | |
| dgraphTemplate | DgraphTemplate | Handles read and write transactions to Dgraph. | |
| objectMapper | ObjectMapper | Jackson mapper for parsing and constructing JSON. | |
| userMapper | UserMapper | Maps domain `User` objects to GraphQL representation. | |
| **Methods** | Name | Input | Output | Description |
| existsByEmail | String email | boolean | Checks if a user with the given email exists. |
| existsByUsername | String username | boolean | Checks if a user with the given username exists. |
| findById | String id | Optional<User> | Retrieves a user by their ID. |
| findByEmail | String email | Optional<User> | Retrieves a user by their email. |
| findByUsername | String username | Optional<User> | Retrieves a user by their username. |
| save | User user | User | Persists or updates a user entity in Dgraph. |
| findUserByUid | String uid | Optional<User> | Finds a user by their Dgraph UID. |
| findSocialAccountsByUserUid | String uid | Collection<SocialAccount> | Retrieves social accounts associated with the given user UID. |
| findByPostId | String postId | Optional<User> | Retrieves the author of a post by post ID. |
| suggestUsers | String userId, Integer first, Integer offset | List<User> | Suggests users based on social graph (friends of friends). |
| findUsersOrderByTotalFollowersExceptUser | String userId, Integer first, Integer offset | List<User> | Retrieves top users ordered by follower count, excluding the given user. |
| updatePassword | String userId, String newPassword | void | Updates password for all user nodes matching the ID. |
| findAllPaginated | Integer first, String after | Connection<User> | Returns paginated list of users ordered by creation time. |

###### **4.2.1.2.6 DgraphUserProfileRepository Class**

| Class | DgraphUserProfileRepository | | | |
| --- | --- | --- | --- | --- |
| **Description** | Repository implementation to manage `UserProfile` updates for a given `User` entity in Dgraph. | | | |
| **Base Class** | dev.huyhoangg.midia.domain.repository.user.UserProfileRepository.java | | | |
| **Constructor** | Uses Lombok’s `@RequiredArgsConstructor` to inject `DgraphMappingProcessor` and `DgraphTemplate` as final dependencies. | | | |
| **Prototype** | `@Repository`, `@Slf4j`, `@FieldDefaults(level = PRIVATE, makeFinal = true)` | | | |
| **Source File** | DgraphUserProfileRepository.java | | | |
| **Namespace** | dev.huyhoangg.midia.infrastructure.db.persistence | | | |
| **Attributes** | Name | Type | Description | |
| mappingProcessor | DgraphMappingProcessor | Used for converting Java objects to/from Dgraph JSON. | |
| dgraphTemplate | DgraphTemplate | Executes Dgraph read and write operations. | |
| **Methods** | Name | Input | Output | Description |
| updateProfile | UserProfile profile, String userId | UserProfile | Updates the avatar and full name of the user profile identified by `userId` in Dgraph. Throws `UserNotExistsException` if no user found. |

###### **4.2.1.2.7 DgraphMappingProcessor Class**

| Class | DgraphMappingProcessor | | | |
| --- | --- | --- | --- | --- |
| **Description** | Handles mapping between annotated Java objects and Dgraph-compatible JSON structures. Provides serialization and deserialization support for Dgraph nodes. | | | |
| **Base Class** | None | | | |
| **Constructor** | `DgraphMappingProcessor(ObjectMapper objectMapper)` – initialized via Spring injection (Lombok `@RequiredArgsConstructor`). | | | |
| **Prototype** | @Component | | | |
| **Source File** | DgraphMappingProcessor.java | | | |
| **Namespace** | dev.huyhoangg.midia.dgraph.processor | | | |
| **Attributes** | Name | Type | Description | |
| objectMapper | ObjectMapper | Jackson object mapper used to serialize/deserialize JSON. | |
| **Methods** | Name | Input | Output | Description |
| toDgraphNode | <T> T entity | String | Converts an annotated object to its JSON representation for Dgraph. Throws if not annotated with `@DgraphNode`. |
| fromDgraphNode | String json, Class<T> clazz | <T> | Parses JSON into a Java object of type `T`. Validates `dgraph.type` and annotation. |
| fromDefaultQueryResponse | String json, Class<T> clazz | Collection<T> | Converts a Dgraph query response into a collection of objects of type `T`. Handles parsing and validation. |
| fromDgraphRelationship | JsonNode jsonNode, Class<T> clazz | <T> | Placeholder for mapping Dgraph relationships. Not implemented. |

###### **4.2.1.2.8 DgraphTemplate Class**

| Class | DgraphTemplate | | | |
| --- | --- | --- | --- | --- |
| **Description** | A Spring component providing transaction management and execution utility methods for Dgraph database interactions. It encapsulates read-only queries, mutations, upserts, and retry logic, with JSON and transaction error handling. | | | |
| **Base Class** | None | | | |
| **Constructor** | `DgraphTemplate(DgraphClient dgraphClient)` – automatically injected by Spring using `@RequiredArgsConstructor`. | | | |
| **Prototype** | @Component | | | |
| **Source File** | DgraphTemplate.java | | | |
| **Namespace** | dev.huyhoangg.midia.dgraph.processor | | | |
| **Attributes** | Name | Type | Description | |
| dgraphClient | DgraphClient | Injected client for communicating with Dgraph. | |
| **Me** | Name | Input | Output | Description |
| executeReadOnlyQuery | FunctionWithJsonException<Transaction, R> | R | Executes a read-only query and handles `JsonProcessingException`, rethrows as `RuntimeException`. |
| executeReadOnly**thods**  QueryReturnOptional | FunctionWithJsonException<Transaction, R> | Optional<R> | Executes a read-only query; returns `Optional.empty()` on failure. |
| executeMutation | FunctionWithJsonOrTxnException<Transaction, R> | R | Executes a mutation with automatic discard and error mapping. |
| executeMutationWithRetry | FunctionWithJsonOrTxnException<Transaction, R> | R | Executes a mutation and retries on conflict (max 3 times). |
| executeMutation (void) | ReturnVoidFunctionWithJsonOrTxnException<Transaction> | void | Executes a mutation returning void; handles all transaction cleanup. |
| executeUpsert | `String query`, `DgraphProto.Mutation mutation` | void | Executes an upsert with a single mutation. |
| executeUpsertReturnSuccess | `String query`, `DgraphProto.Mutation... mutations` | boolean | Executes an upsert with multiple mutations and returns true on success. |
| **Functional Interfaces** | Name | Input | Output | Description |
| FunctionWithJsonException<T, R> | T txn | R | Function interface allowing `JsonProcessingException`. |
| FunctionWithJsonOrTxnException<T, R> | T txn | R | Function interface allowing both JSON and transaction exceptions. |
| ReturnVoidFunctionWithJsonOrTxnException<T> | T txn | void | Void-returning function with JSON and transaction exception handling. |

###### **4.2.1.2.9 User**

| Class | User | | | |
| --- | --- | --- | --- | --- |
| **Description** | Represents a user in the system, mapped to a Dgraph node. It encapsulates personal information, relationships, and authentication data. | | | |
| **Base Class** | DgraphBaseModel | | | |
| **Constructor** | No-args constructor and all-args constructor provided by Lombok. | | | |
| **Prototype** | public class User extends DgraphBaseModel | | | |
| **Source File** | User.java | | | |
| **Namespace** | dev.huyhoangg.midia.domain.model.user | | | |
| **Attributes** | Name | Type | Description | |
| id | String | The UUID identifier of the user. | |
| username | String | The username of the user. | |
| profile | UserProfile | The profile object associated with the user. | |
| email | String | The email address of the user. | |
| password | String | The hashed password of the user. | |
| stats | UserStats | Statistical information related to the user. | |
| role | Role | Role assigned to the user for permission handling. | |
| followings | Set<User> | Set of users that this user is following. | |
| followers | Set<User> | Set of users that follow this user. | |
| socialAccounts | Set<SocialAccount> | Set of social media accounts linked to the user. | |
| isLocked | Boolean | Indicates whether the user account is locked. | |
| isEmailVerified | Boolean | Indicates whether the user's email has been verified. | |
| createdAt | Instant | Timestamp of when the user was created. | |
| updatedAt | Instant | Timestamp of the last update to the user. | |
| deletedAt | Instant | Timestamp of when the user was deleted (soft delete). | |
| **Methods** | Name | Input | Output | Description |
| Getters/Setters | None | Various types | Generated by Lombok for all attributes. |
| toString | None | String | Returns a string representation of the User object. |
| builder | None | User.UserBuilder | Enables fluent API for constructing `User` instances. |

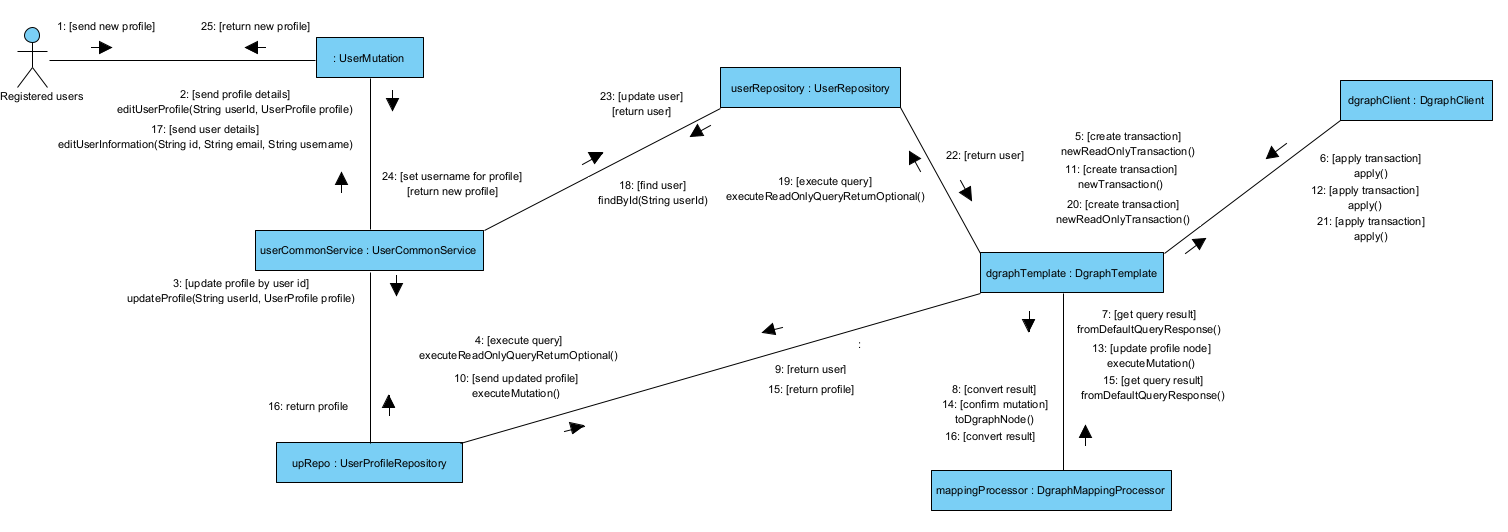
###### **4.2.1.2.10 UserProfile**

| Class | UserProfile | | | |
| --- | --- | --- | --- | --- |
| **Description** | Represents a user profile containing personal information stored in Dgraph. Designed to be mapped as a Dgraph node. | | | |
| **Base Class** | DgraphBaseModel | | | |
| **Constructor** | No-args constructor and all-args constructor via Lombok annotations (`@NoArgsConstructor`, `@AllArgsConstructor`) | | | |
| **Prototype** | `@DgraphNode`, POJO with builder pattern and Jackson annotations | | | |
| **Source File** | UserProfile.java | | | |
| **Namespace** | dev.huyhoangg.midia.domain.model.user | | | |
| **Attributes** | Name | Type | Description | |
| fullName | String | Full name of the user (`user\_profile.full\_name`) | |
| phoneNumber | String | User's phone number (`user\_profile.phone\_number`) | |
| birthDate | LocalDate | Date of birth of the user (`user\_profile.dob`) | |
| bio | String | Short biography or introduction (`user\_profile.bio`) | |
| avatarUrl | String | URL of the user's avatar image (`user\_profile.avatar\_url`) | |
| gender | String | Gender of the user (`user\_profile.gender`) | |
| **Methods** | Name | Input | Output | Description |
| Getters/Setters | None | Various types | Generated by Lombok for all attributes. |
| toString | None | String | Returns a string representation of the User object. |
| builder | None | User.UserBuilder | Enables fluent API for constructing `User` instances. |

###### **4.2.1.2.11 UserProfile - codegen**

| Class | UserProfile | | | |
| --- | --- | --- | --- | --- |
| **Description** | A data model representing user profile information including personal details, biography, and avatar. | | | |
| **Base Class** | None | | | |
| **Constructor** | UserProfile()  UserProfile(String fullName, String phoneNumber, String birthDate, String bio, String avatarUrl, String username) | | | |
| **Prototype** | public class UserProfile | | | |
| **Source File** | UserProfile.java | | | |
| **Namespace** | dev.huyhoangg.midia.codegen.types | | | |
| **Attributes** | Name | Type | Description | |
| fullName | String | Full name of the user (`user\_profile.full\_name`) | |
| phoneNumber | String | User's phone number (`user\_profile.phone\_number`) | |
| birthDate | LocalDate | Date of birth of the user (`user\_profile.dob`) | |
| bio | String | Short biography or introduction (`user\_profile.bio`) | |
| avatarUrl | String | URL of the user's avatar image (`user\_profile.avatar\_url`) | |
| username | String | Unique username of the user | |
| **Methods** | Name | Input | Output | Description |
| Getters/Setters | None | Various types | Generated for all attributes. |
| toString | None | String | Returns a string representation of the User object. |
| equals | (Object o) | boolean | Compares this profile with another for equality |
| hashCode | None | int | Returns the hash code of the object |
| newBuilder | NOne | Builder | Returns a new builder for constructing a `UserProfile` |
| Builder.fullName | String | Builder | Sets full name in builder |
| Builder.phoneNumber | String | Builder | Sets phone number in builder |
| Builder.birthDate | String | Builder | Sets birth date in builder |
| Builder.bio | String | Builder | Sets bio in builder |
| Builder.avatarUrl | String | Builder | Sets avatar URL in builder |
| Builder.username | String | Builder | Sets username in builder |
|  | Builder.build | None | UserProfile | Builds a new instance of `UserProfile` |

#### 4.2.1.3 Object Interactions



#### 

## 4.3 Notification

### 4.3.1 Check Recent Notifications

#### 4.3.1.1 Screen Design



**Figure 4.3.1**: Notification modal

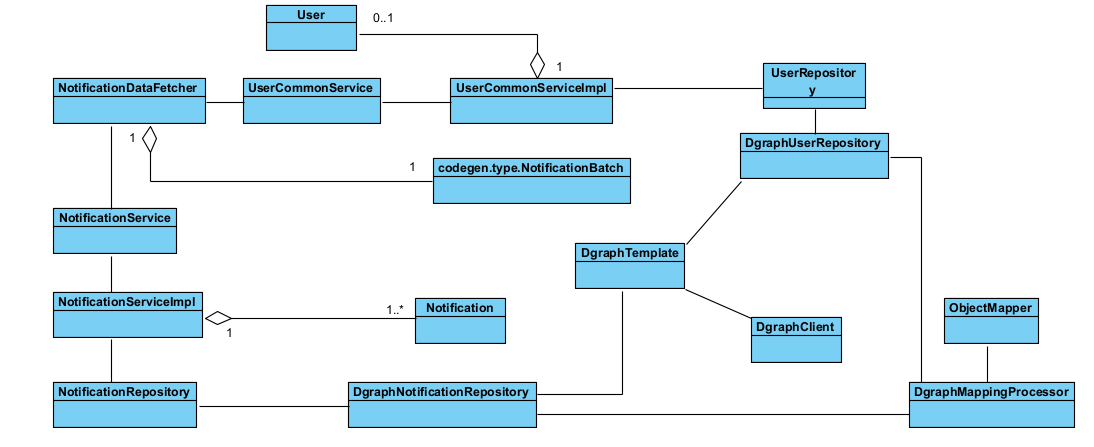
**Table 4.3.1: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | ctaButtonLabel | String | No | N/A | Caption on the call-to-action button when the notification requires a response (e.g., **“Follow”**, **“Following”**). Hidden for non-interactive notifications. |
| 2 | targetURL | URL | Yes | 2083 | With this field in place, every notification item can seamlessly deep-link to the related post or user profile. |
| 3 | timeAgoLabel | String | Yes | 10 | Relative time stamp (“1 d”, “3 h”). Concise ≤ 5 chars plus a trailing space if displayed inline. |
| 4 | activityMessage | String | Yes | 200 | Full descriptive sentence that follows the actors (e.g., *“started following you.”*, *“… like your post.”*). Includes punctuation. |
| 5 | actorNamesText | String | Yes | 100 | Human-readable list of actors shown in bold (e.g., *“roirin\_femlivart2931ec”* or *“Vincent, Leo and 12 others”*). |
| 6 | sectionHeader | String | Yes | 20 | Group label that partitions the list chronologically (e.g., “Yesterday”, “This Week”). |

#### 

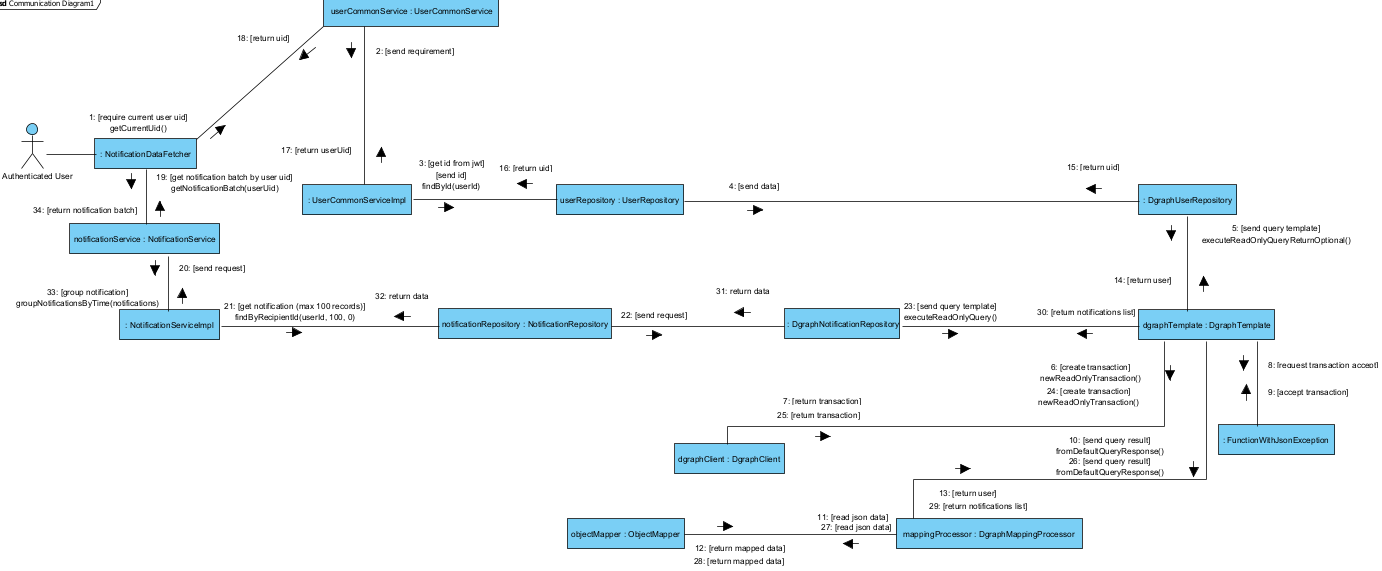
#### 4.3.1.2 Class Design

##### 4.3.1.2.1 Classes/Component Structure



##### 4.3.1.2.2 Classes/Component Description

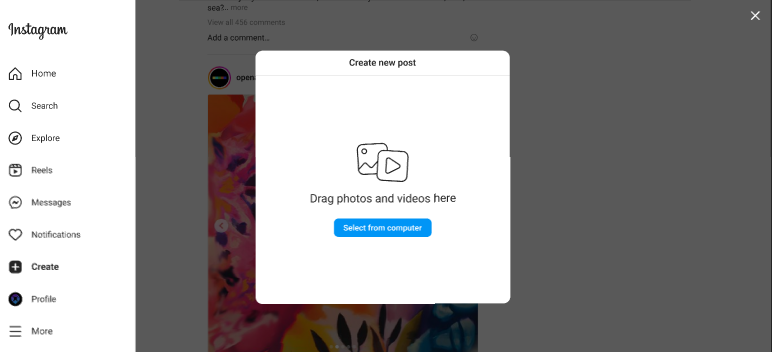
#### 4.3.1.3 Object Interactions



## 4.4 Content Management

### 4.4.1 Create post

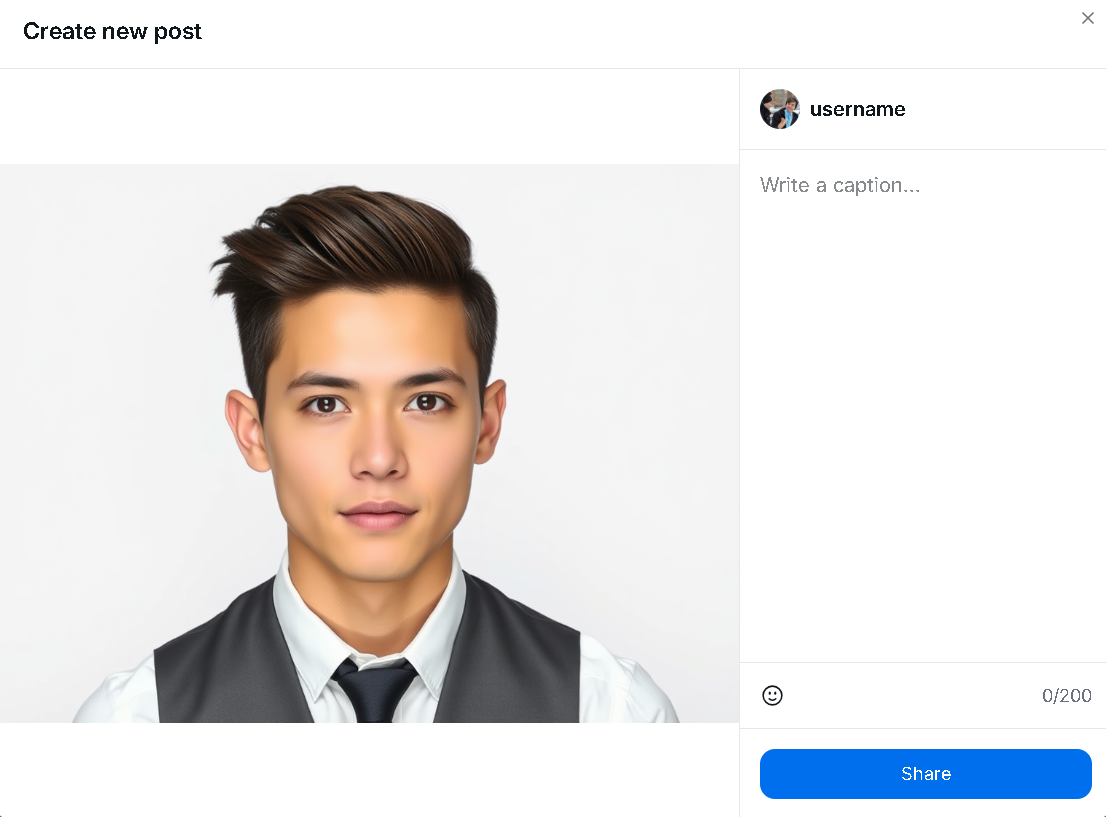
#### 4.4.1.1 Screen Design

****

**Figure 4.4.1-1**: Screen Design of Create Post

**Table 4.4.1-1: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | uploadImage | Button | Yes | N/A | Import image or video from system |

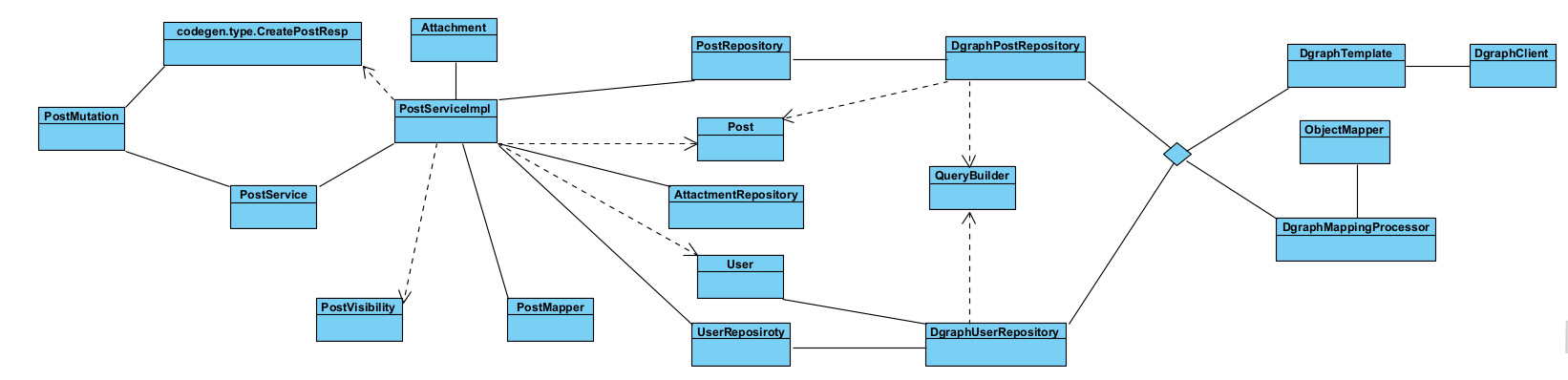
******

**Figure 4.1.1-2**: Screen Design of post sharing

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | postImage | image | Yes | N/A | Display image of post. Can click to image to tag other user |
| 2 | captionText | text area | Yes | 255 | Write the caption of the post |
| 3 | emojiButton | button | No | N/A | Choose emoji for the caption for more beautiful caption |
| 4 | shareButton | button | Yes | N/A | Submit to create a new post |

#### 4.4.1.2 Class Design

##### 4.4.1.2.1 Classes/Component Structure

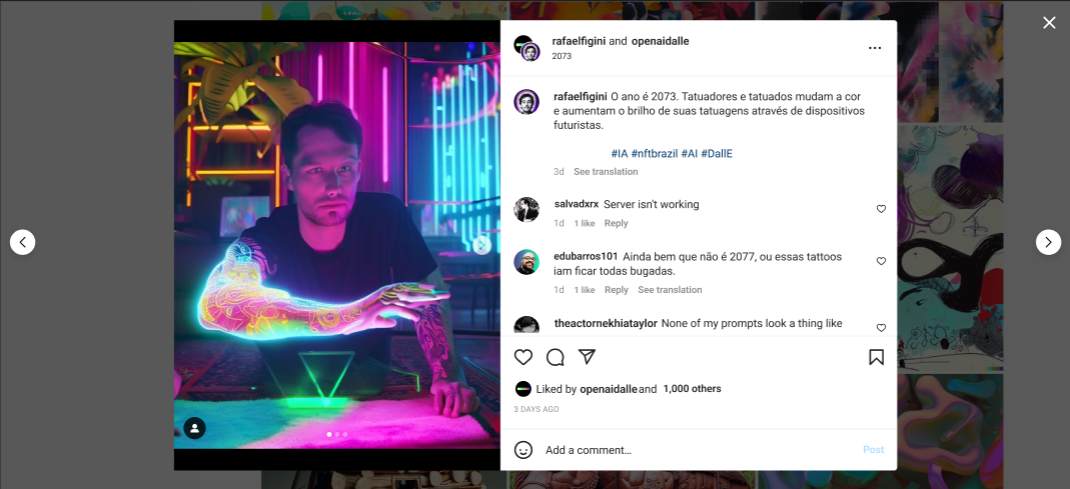


##### 4.4.1.2.2 Classes/Component Description

#### 4.4.1.3 Object Interactions

### 4.4.2 View Post Details

#### 4.4.2.1 Screen Design



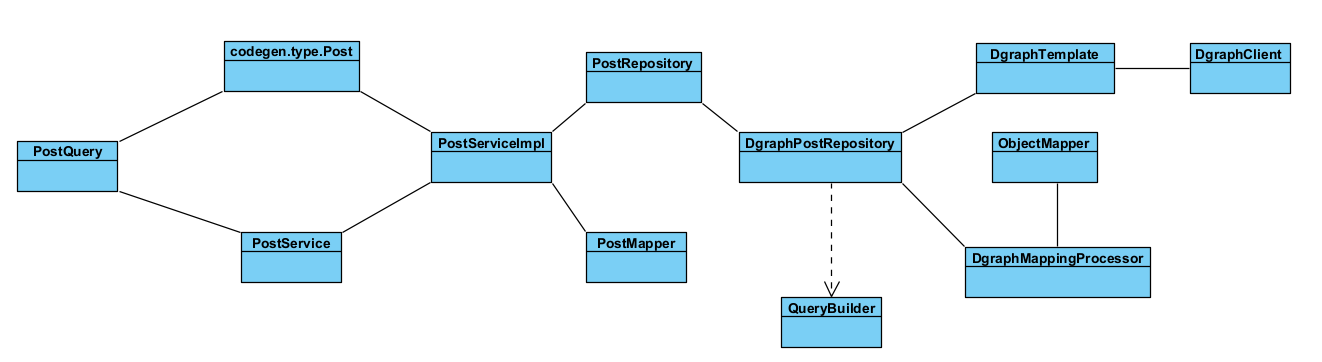
**Figure 4.4.2**: Screen Design of Post Details

**Table 4.4.2: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | postImage | Image | Yes |  | Display images of a post |
| 2 | nextImageButton | Button | Yes |  | Allow users to move to next image of this post if there are more than 2 images |
| 3 | previousImageButton | Button | Yes |  | Allow users to move to previous image of this post if there are more than 2 images |
| 4 | userAvatar | Image | Yes |  | Display users’ avatars |
| 5 | usernameLabel | Label | Yes |  | Display users’ username |
| 6 | fullNameLabel | Label | Yes |  | Display the current user’s full name |
| 7 | postTimeLabel | Label | Yes |  | Display how long this post was created |
| 8 | postContentLabel | Label | Yes |  | Display the description of image that users import in a post |
| 9 | commentField | Label | No |  | Display the comments of other users |
| 10 | commentTimeLabel | Label | Yes |  | Display how long this comment was posted |
| 11 | commentLikeIcon | Icon | Yes |  | Display how many likes that this comment has |
| 12 | replyLink | Link | Yes |  | Allow users to answer someone’s comment |
| 13 | likeIcon | Icon | Yes |  | Allow users to like comments |
| 14 | commentIcon | Icon | Yes |  | Allow users to create a comment |
| 15 | shareIcon | Icon | Yes |  | Allow users to share this post |
| 16 | likeCountLabel | Label | Yes |  | Display the number of liked that users’ post have and their friends’ username |
| 17 | feelingIcon | Icon | Yes |  | Allow users to add some icons in their comment |
| 18 | addCommentInput | Text | Yes |  | Allow users to type their comment |
| 19 | postCommentButton | Button | Yes |  | Allow users to post their comment |

#### 4.4.2.2 Class Design

##### 4.4.2.2.1 Classes/Component Structure



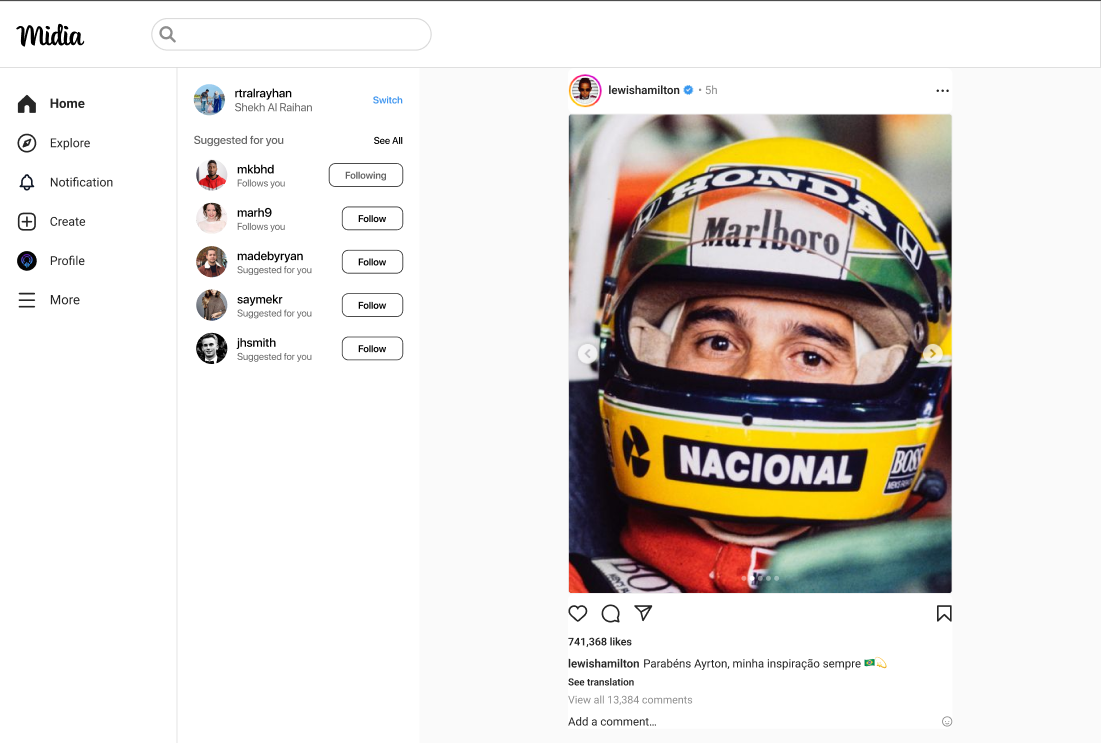
##### 4.4.2.2.2 Classes/Component Description

#### 4.4.2.3 Object Interactions

## 4.5 Follow system

### 4.5.1 Follow Suggestions

#### 4.5.1.1 Screen Design



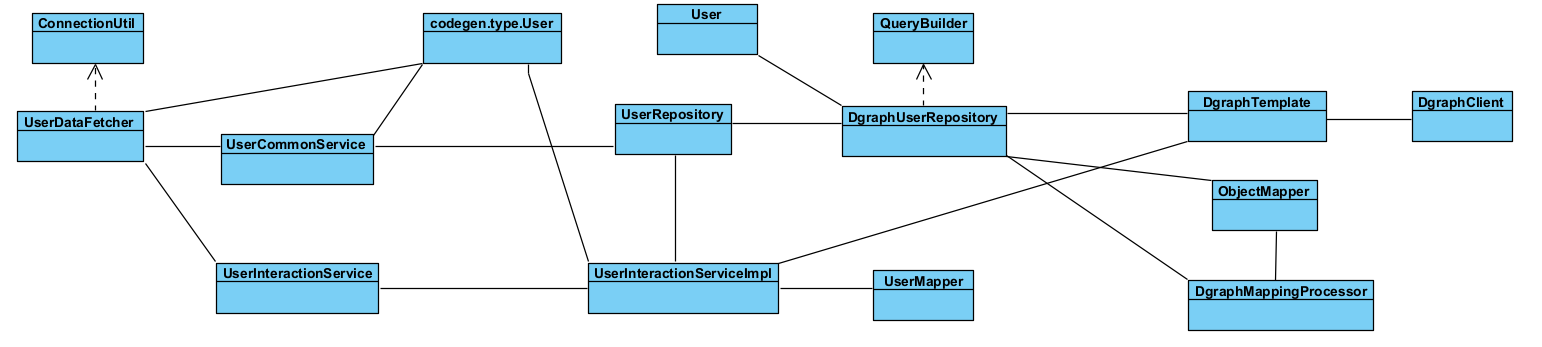
**Figure 4.5.1**: Screen Design of Follow Suggestions

**Table 4.5.1: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | usernameLabel | Label | Yes |  | Display usernames of accounts. |
| 2 | fullNameLabel | Label | Yes |  | Display the full name of user who is using the website |
| 3 | avatarImage | Image | Yes |  | Display accounts’ avatars |
| 4 | relationshipStatusLabel | Label | Yes |  | Displays a label next to another user's name in lists or search results.  **Follows you**: Indicates that the user is currently following you. Helps you quickly identify mutual connections.  **Suggested for you**: Shown for users that Instagram recommends based on your behavior, interests, or mutual connections. These users are not necessarily following you. |
| 5 | switchAccountLink | Link | Yes |  | Allow users to use another saved account or login with unsaved account |
| 6 | seeAllLink | Link | Yes |  | Navigate to screen that list all accounts that users can follow |
| 7 | followButton | Button | Yes |  | Allow users to start follow or cancel follow other accounts  **Follow:** Displayed when the user *has not followed* the account. Clicking it sends the following request.  **Following**: Displayed when the user *is currently following* the account. Clicking it may open an "Unfollow" option or directly trigger the unfollow action (depending on the design). |

#### 4.5.1.2 Class Design

##### 4.5.1.2.1 Classes/Component Structure



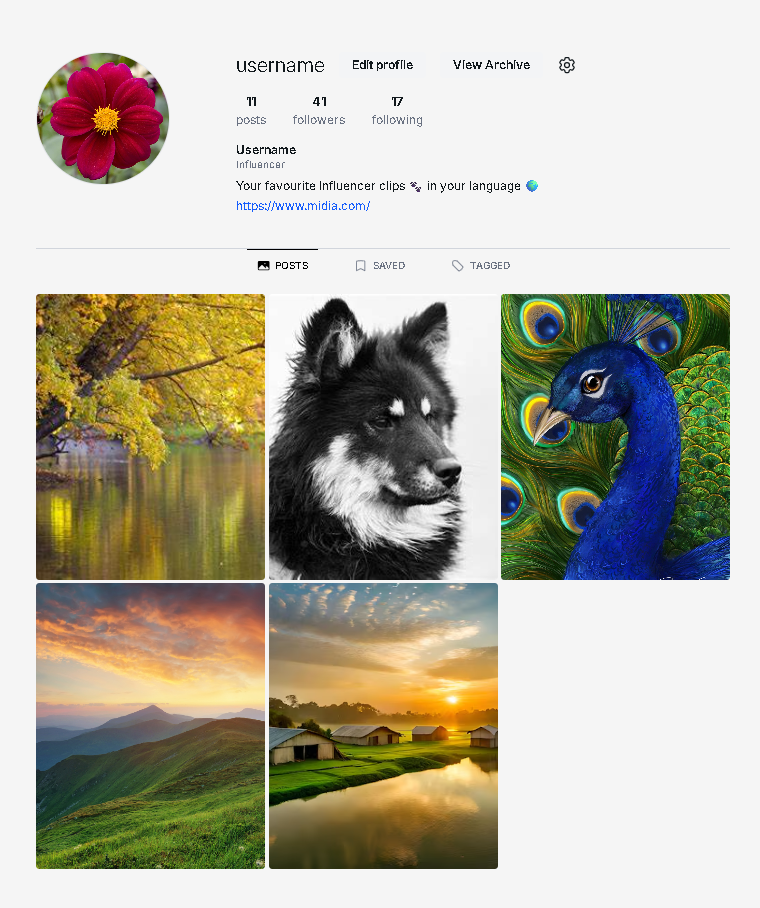
##### 4.5.1.2.2 Classes/Component Description

#### 4.5.1.3 Object Interactions

## 4.6 Profile Management

### 4.6.1 Profile page

#### 4.6.1.1 Screen Design



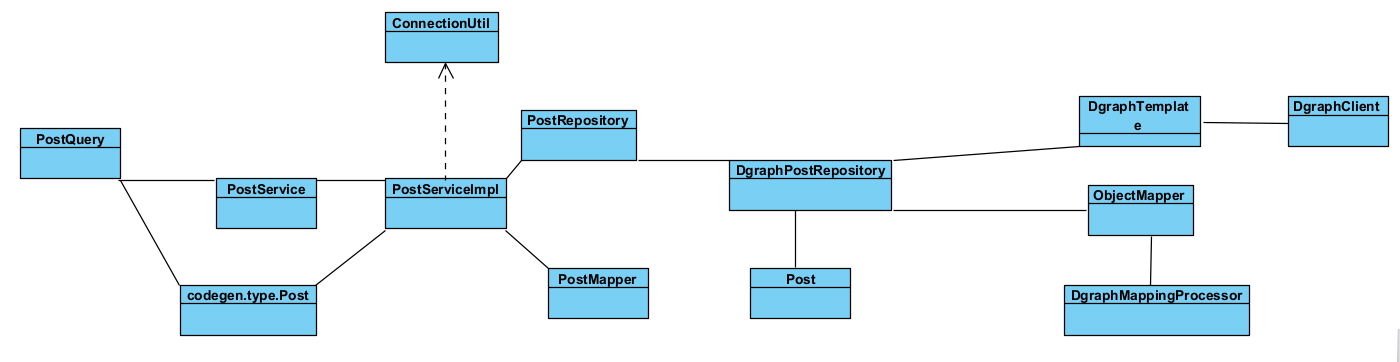
**Figure 4.6.1**: Screen Design of Follow Suggestions

**Table 4.6.1: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | usernameLabel | Label | Yes | N/A | Display usernames of accounts. |
| 2 | editProfileButton | Button | No | N/A | Navigate to edit profile page |
| 3 | avatarImage | Image | Yes | N/A | Display accounts’ avatars |
| 4 | postCountText | Text | Yes | 10 | Display number of posts of that user |
| 5 | followerCountText | Text | Yes | 10 | Display number of followers of that user |
| 6 | followingCountText | Text | Yes | 10 | Display number of following user of that user |
| 7 | profileDescriptionText | Text | No | 100 | Display the description of that user profile |
| 8 | profileLinkText | Text | No | 100 | Display the link of other profile page |
| 9 | postNavigationLink | Link | No | N/A | Display all the post of that user |
| 10 | savedNavigationLink | Link | No | N/A | Display all the saved post of that user |
| 11 | taggedNavigationLink | Link | No | N/A | Display all the post of that user that been tagged by others |
| 12 | imagePost | Image | Yes | N/A | Display the image of the posts. When hover, show the number of comment and like. When click, show modal of image, comment and number of like. User can interact inside that modal |
| 13 | settingsButton | Button | No | N/A | Navigate user to User Setting screen page |

#### 4.6.1.2 Class Design

##### 4.6.1.2.1 Classes/Component Structure



##### 4.6.1.2.2 Classes/Component Description

#### 4.6.1.3 Object Interactions

### 4.6.2 Edit Profile

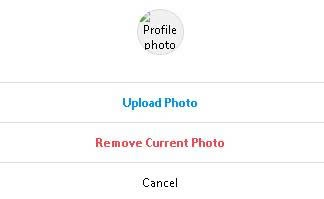
#### 4.6.2.1 Screen Design

##### 

**Figure 4.6.2-1**: Screen Design of Edit Profile

**Table 4.6.2-1: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | avatarImage | Image | Yes |  | Display accounts’ avatars |
| 2 | usernameText | Text | Yes |  | Display username of account which has logged in |
| 3 | fullNameLabel | Label | Yes |  | Display the full name of user who is using the website |
| 4 | changePhotoButton | Button |  |  | Allow users to replace the old avatar in the popup screen |
| 5 | websiteText | Text | No |  | Display users’ website link. This field can be edited if users are using a mobile app and optional |
| 6 | bioText | Text | No |  | Display short biography of users. Users can only type about 150 characters. This field is optional |
| 7 | genderDropdown | Dropdown box | Yes |  | Display the user's gender. This field contains Male, Female and Prefer not to say |
| 8 | showAccountSuggestionsSwitch | Switch | No |  | Allow users to see similar account suggestions on their profiles |
| 9 | submitButton | Button | Yes |  | Allow to save all changes |



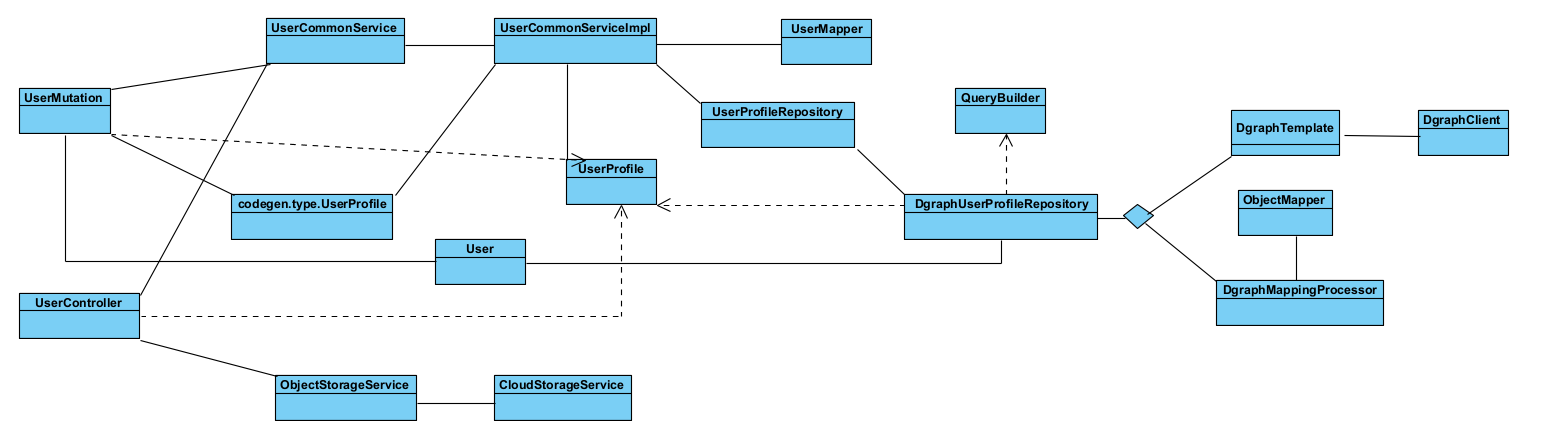
**Figure 4.6.2-2**: Screen Design of Change photo popup screen

**Table 4.6.2-2: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | profilePhotoImage | Image | Yes |  | Display accounts’ avatars |
| 2 | uploadPhotoButton | Button | Yes |  | Allow users to upload a new photo |
| 3 | removePhotoButton | Button | Yes |  | Allow users to remove the current avatar |
| 4 | cancelButton | Button | Yes |  | Allow users to cancel all changes and go back to previous screen |

#### 4.6.2.2 Class Design

##### 4.6.2.2.1 Classes/Component Structure



##### 4.6.2.2.2 Classes/Component Description

#### 4.6.2.3 Object Interactions

## 4.7 Home Features

### 4.7.1 Landing home page

#### 4.7.1.1 Screen Design

##### 

**Figure 4.7.1**: Screen Design of Home page

**Table 4.7.1: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | usernameLink | Text(Link) | Yes | 20 | The display name of the user who posted. Clickable to navigate to profile page. |
| 2 | postImageText | Text | Yes | N/A | The main image of the post. |
| 3 | postCaptionText | Text | No | 200 | The description or caption of the post. |
| 4 | likeCount | Integer | Yes | N/A | The number of likes for the post. |
| 5 | commentInput | Text Input | No | N/A | Input field where user can type a comment. |
| 6 | commentLinkButton | Icon Button | No | N/A | Link to view all comments of the post. |
| 7 | postTimestampDropdown | Dropdown box | Yes | N/A | Opens a dropdown menu with actions (e.g., report, share). |
| 8 | followButton | Button | No | N/A | Button to follow the suggested user. |
| 9 | moreOptionsButton | Icon Button | No | N/A | Opens a dropdown menu with actions (e.g., report, share). |
| 10 | suggestedUsernameLink | Link | Yes | 20 | Username of the suggested account to follow. |
| 11 | viewAllCommentsButton | Button | Yes | N/A | Opens the comment section to view all comments. |
| 12 | reactionIconsGroup | Icon Group | No | N/A | Icons for actions: like, comment, share. |
| 13 | commentPlaceholderText | Placeholder text | No | N/A | Placeholder text in the comment input, e.g. “Add a comment…” |
| 14 | suggestionsCardGroup | Card Group | No | N/A | Cards showing user suggestions with avatar, name, and follow button. |
| 15 | navigationMenuButtons | Button group | Yes | N/A | Sidebar with navigation: Home, Explore, Notification, Create, Profile, More. |
| 16 | profilePictureButton | Image button | Yes | N/A | Avatar image of the user. Clickable to view their profile. |

#### 4.7.1.2 Class Design

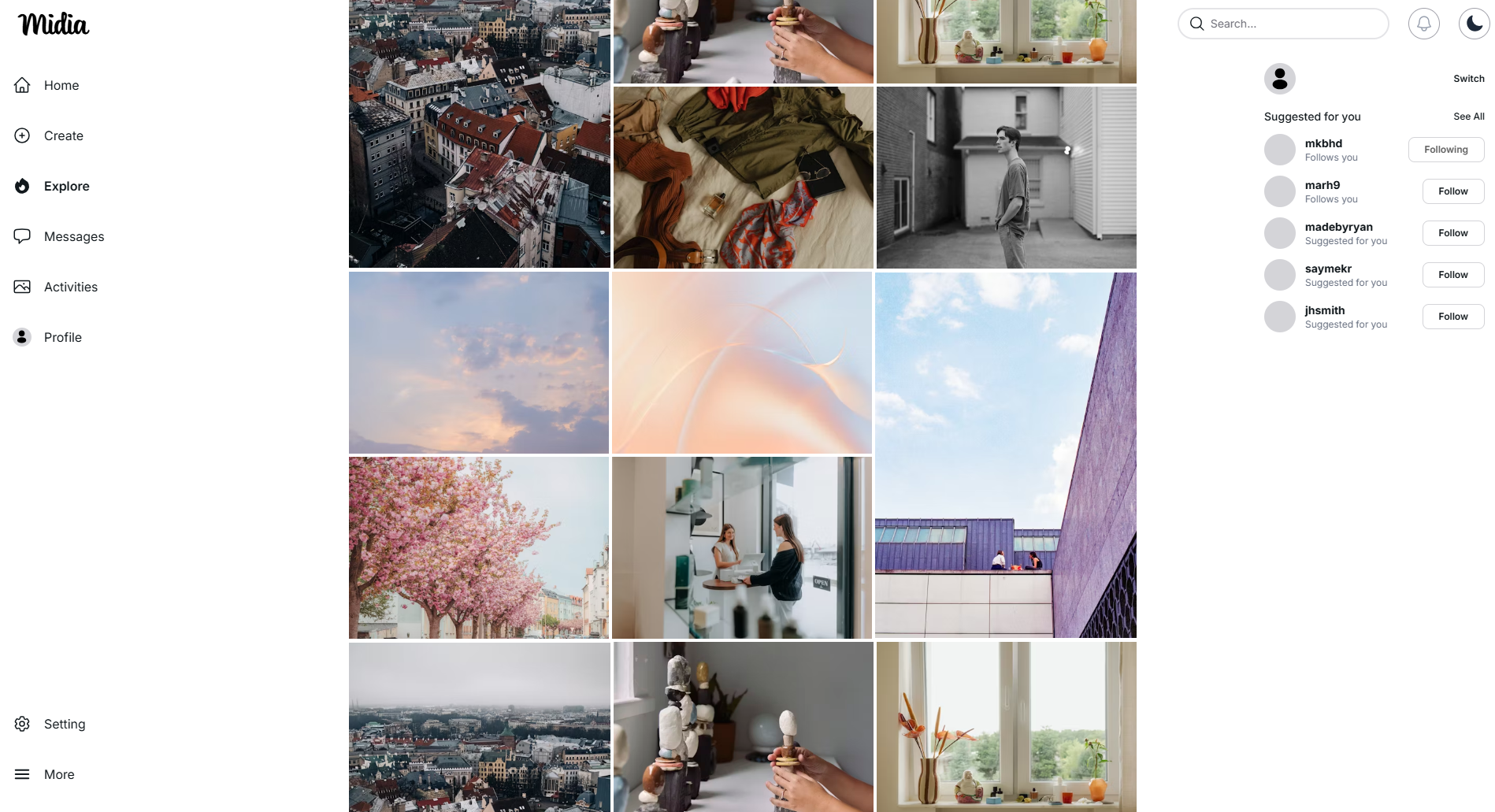
##### 4.7.1.2.1 Classes/Component Structure

##### 4.7.1.2.2 Classes/Component Description

#### 4.7.1.3 Object Interactions

### 4.7.2 Explore page

#### 4.7.2.1 Screen Design



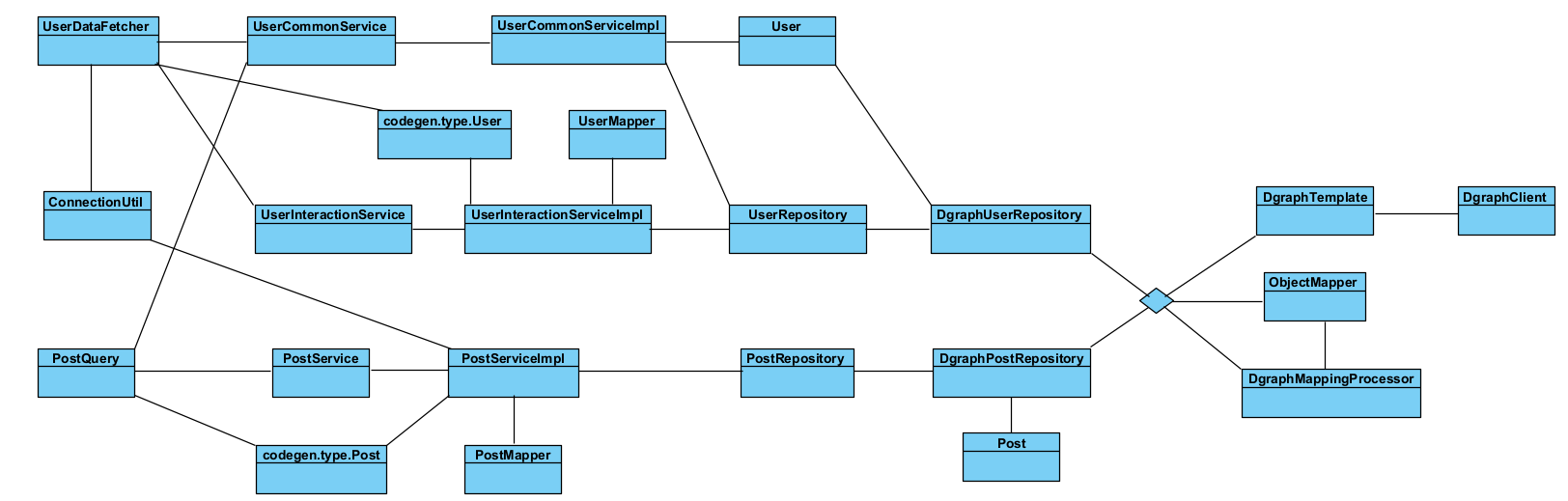
**Figure 4.7.2**: Screen Design of Explore page

**Table 4.7.2: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | postImageText | Text | Yes | N/A | The main image of the post. |

#### 4.7.2.2 Class Design

##### 4.7.2.2.1 Classes/Component Structure

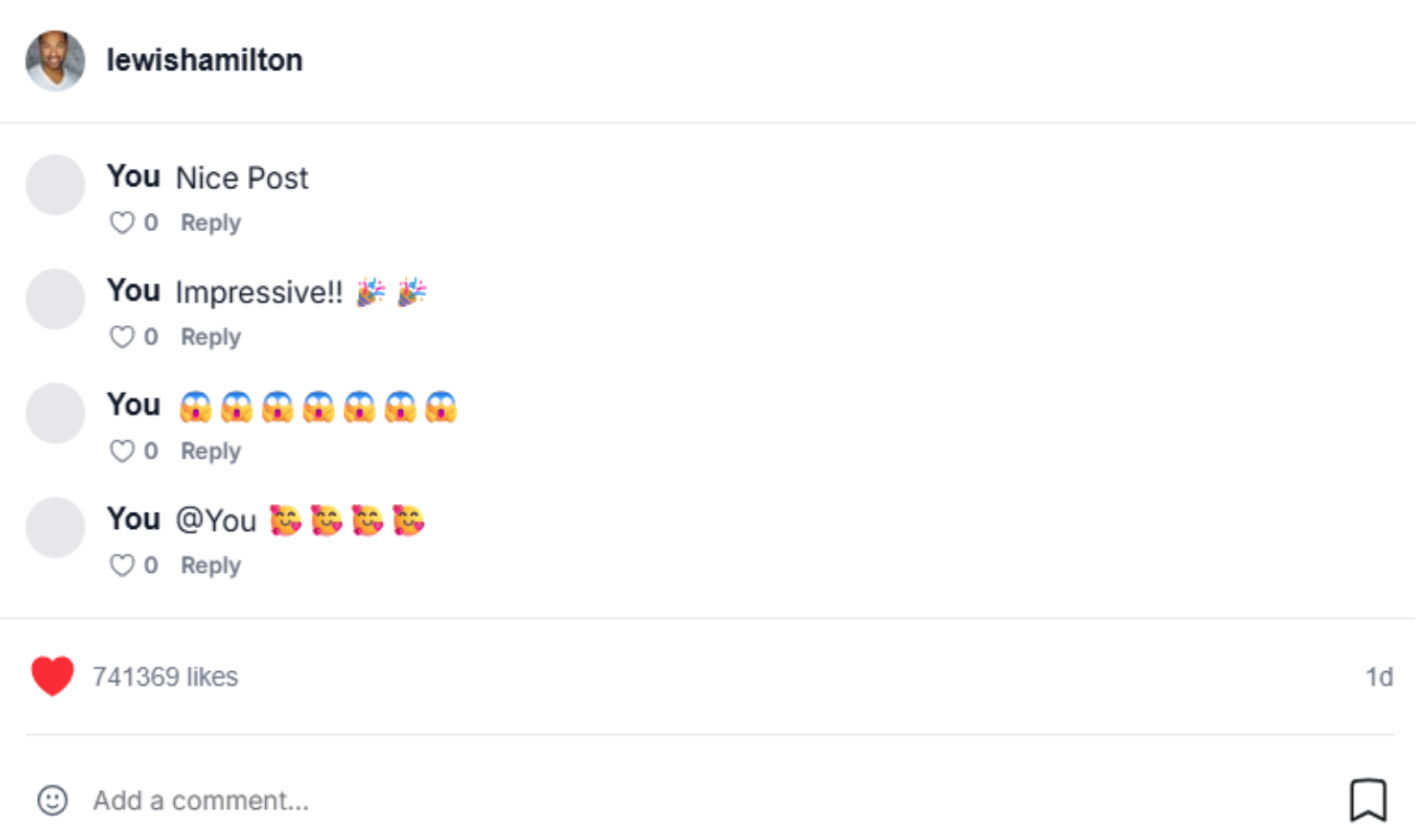


##### 4.7.2.2.2 Classes/Component Description

#### 4.7.2.3 Object Interactions

### 4.7.3 Comment Section

#### 4.7.3.1 Screen Design

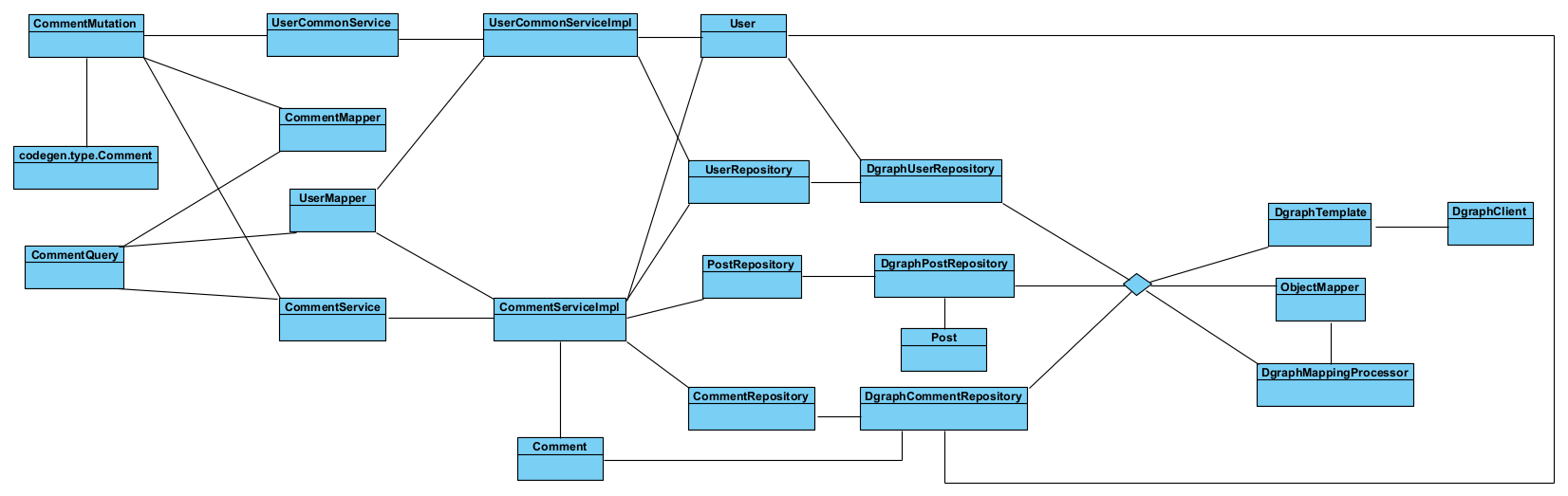


**Table 4.7.3: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | usernameLink | Text(Link) | Yes | 20 | The display name of the user who posted. Clickable to navigate to profile page. |
| 2 | postCaptionText | Text | No | 200 | The description or caption of the post. |
| 3 | likeCount | Integer | Yes | N/A | The number of likes for the post. |
| 4 | commentInput | Text Input | No | N/A | Input field where user can type a comment. |
| 5 | postTimestampText | Text | Yes | N/A | Represent created time of post |
| 6 | reactionIconsGroup | Icon Group | No | N/A | Add Icons for comment |
| 7 | commentPlaceholderText | Placeholder text | No | N/A | Placeholder text in the comment input, e.g. “Add a comment…” |
| 8 | heartIconButton | Button | No | N/A | like/unlike post |
| 9 | bookmarkButton | Button | No | N/A | Save post |

#### 4.7.3.2 Class Design

##### 4.7.3.2.1 Classes/Component Structure



##### 4.7.3.2.2 Classes/Component Description

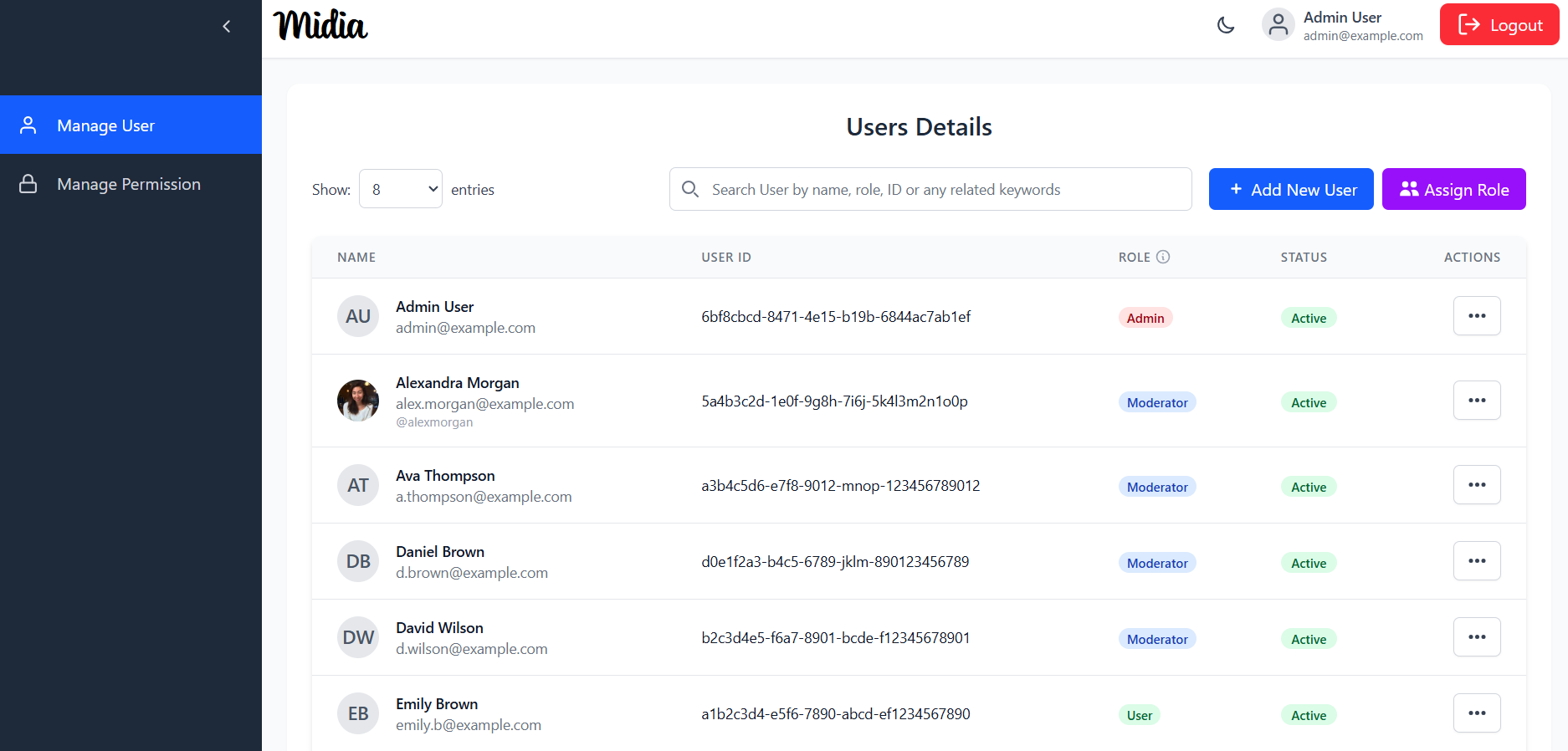
#### 4.7.3.3 Object Interactions

## 

## 4.8 Admin Features

### 4.8.1 View user list

#### 4.8.1.1 Screen Design



**Figure 4.8.1**: Screen Mockup of View User List

**Table 4.8.1: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | showEntriesDropdown | Dropdown | Yes | N/A | A dropdown to select the number of user records displayed per page. |
| 2 | searchInput | Text Input | No | 100 | A search bar to filter the user list by keywords such as name, role, or ID. |
| 3 | userNameWithAvatar | Text (with avatar) | Yes | N/A | Displays the user's avatar, full name, and email address. |
| 4 | userId | Text | Yes | N/A | The unique identifier for the user. |
| 5 | userRoleTag | Tag | Yes | 20 | A colored tag indicating the user's assigned role (e.g., Admin, Moderator). |
| 6 | accountStatusChip | Chip | Yes | 15 | A colored tag showing the user's account status (e.g., Active). |
| 7 | actionMenu | Button (Menu) | Yes | N/A | A menu button (three dots) that reveals a list of actions for the specific user, including Edit, Toggle status, View Details. |

#### 4.8.1.2 Class Design

##### 4.8.1.2.1 Classes/Component Structure

##### 4.8.1.2.2 Classes/Component Description

#### 4.8.1.3 Object Interactions

### 

### 4.8.2 View user detail

#### 4.8.2.1 Screen Design

### 

**Figure 4.8.2**: Screen Mockup of View User Detail

**Table 4.8.2: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | avatar | Image | Yes | N/A | The user's profile picture displayed at the top. |
| 2 | fullName | Text | Yes | 100 | The complete name of the user. |
| 3 | email | Text | Yes | 100 | The user's primary email address. |
| 4 | role | Chip | Yes | 20 | A tag indicating the user's assigned role (e.g., Moderator). |
| 5 | userId | Text | Yes | N/A | The unique, non-editable system identifier for the user. |
| 6 | username | Text | Yes | 50 | The user's unique username. |
| 7 | emailVerificationStatus | Icon + Text | Yes | N/A | Indicates whether the user's email has been verified. |
| 8 | accountStatus | Icon + Text | Yes | N/A | Shows the current status of the user's account (e.g., Active, Locked). |
| 9 | profileInformation | Section | Yes | N/A | A container for the user's personal details like phone number, date of birth, and bio. |
| 10 | statistics | Number | Yes | N/A | A section displaying key metrics: total followers, total following, and total posts. |
| 11 | connectedTimeline | Text | Yes | N/A | A list of social media accounts (e.g., Google, Facebook) linked to the user's profile. |
| 12 | accountTimeline | Text | Yes | N/A | Displays the timestamps for when the account was created and last updated. |
| 13 | closeButton | Button | Yes | N/A | A button to dismiss or close the user profile view. |

#### 4.8.2.2 Class Design

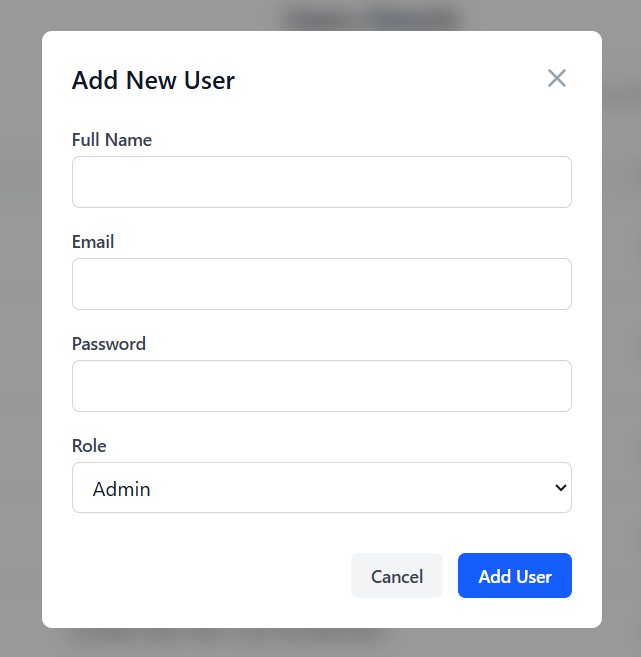
##### 4.8.2.2.1 Classes/Component Structure

##### 4.8.2.2.2 Classes/Component Description

#### 4.8.2.3 Object Interactions

### 4.8.3 Add new user

#### 4.8.3.1 Screen Design



**Figure 4.8.3**: Screen Mockup of Add New User

**Table 4.8.3: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | password | Text Input | Yes | 100 | An input field for setting the user's password. The characters should be masked. |
| 2 | fullName | Text Input | Yes | 100 | The complete name of the user. |
| 3 | email | Text Input | Yes | 100 | The user's primary email address. |
| 4 | role | Dropdown | Yes | N/A | A dropdown menu to select and assign a role (e.g., Admin, Moderator, User) to the new user. |
| 5 | cancel | Button | Yes | N/A | A button that discards any input and closes the "Add New User" modal. |
| 6 | addUserBtn | Button | Yes | N/A | The primary action button that submits the form to create the user account. |

#### 4.8.3.2 Class Design

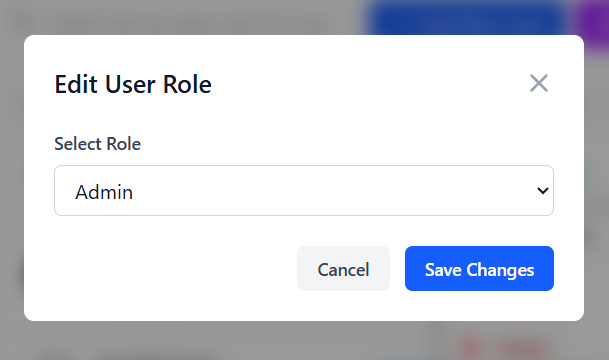
##### 4.8.3.2.1 Classes/Component Structure

##### 4.8.3.2.2 Classes/Component Description

#### 4.8.3.3 Object Interactions

### 4.8.4 Edit user role

#### 4.8.4.1 Screen Design



**Figure 4.8.4**: Screen Mockup of Add New User

**Table 4.8.4: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | selectRole | Dropdown | Yes | N/A | A dropdown menu to select a new role for the user. It should be populated with all available system roles (e.g., Admin, Moderator, User). |
| 2 | cancelButton | Button | Yes | N/A | A button to close the modal and discard the role change. |
| 3 | saveChangesButton | Button | Yes | N/A | The primary button that confirms the selection and updates the user's role. |

#### 4.8.4.2 Class Design

##### 4.8.4.2.1 Classes/Component Structure

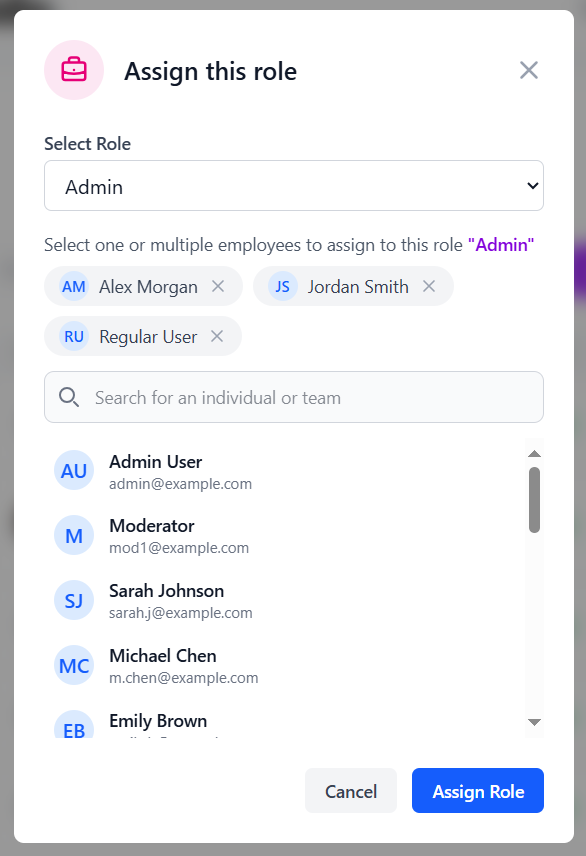
##### 4.8.4.2.2 Classes/Component Description

#### 4.8.4.3 Object Interactions

### 

### 4.8.5 Assign role to user

#### 4.8.5.1 Screen Design



**Figure 4.8.5**: Screen Mockup of Assign role

**Table 4.8.5: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | selectRole | Dropdown | Yes | N/A | A dropdown menu to select a new role for the user. It should be populated with all available system roles (e.g., Admin, Moderator, User). |
| 2 | selectedUsers | Tag | Yes | N/A | A display area showing all users currently selected for the role assignment. Each user appears as a removable tag. |
| 3 | userSearch | Text Input | No | 100 | A search field to find and filter individuals or teams from the list of available users. |
| 4 | availableUsersList | List | Yes | N/A | A scrollable list showing all users who can be selected. Clicking a user in this list adds them to the "Selected Users" area above. |
| 5 | cancelBtn | Button | Yes | N/A | A button to close the modal without saving any changes. |
| 6 | assignRoleCancelButton | Button | Yes | N/A | The primary action button to confirm and apply the selected role to all chosen users. |

#### 4.8.5.2 Class Design

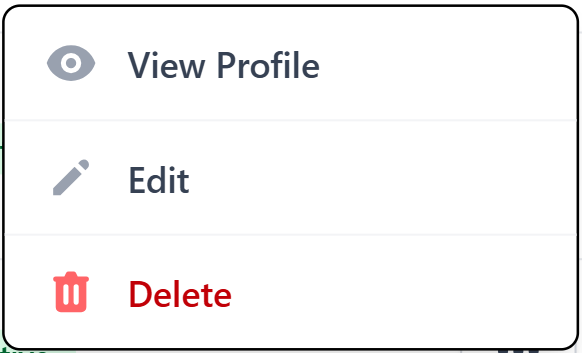
##### 4.8.5.2.1 Classes/Component Structure

##### 4.8.5.2.2 Classes/Component Description

#### 4.8.5.3 Object Interactions

### 4.8.6 Toggle status

#### 4.8.6.1 Screen Design



**Figure 4.8.6**: Screen Mockup of toggle status

**Table 4.8.6**: **Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | toggleStatusButton | Menu | Yes | N/A | An option in the user "Actions" menu. The text would dynamically change to "Deactivate" for an active user and "Activate" for an inactive one. |
| 2 | confirmationDialog | Modal | Yes | N/A | A system alert that would appear after clicking the "Toggle Status" button, asking the admin to confirm the action. |
| 3 | confirmationMessage | Text | Yes | 150 | The text within the dialog, such as "Are you sure you want to deactivate this user's account?". |
| 4 | okButton | Button | Yes | N/A | The button that confirms the status change. |
| 5 | cancelButton | Button | Yes | N/A | The button that aborts the action and closes the confirmation dialog. |

#### 4.8.6.2 Class Design

##### 4.8.6.2.1 Classes/Component Structure

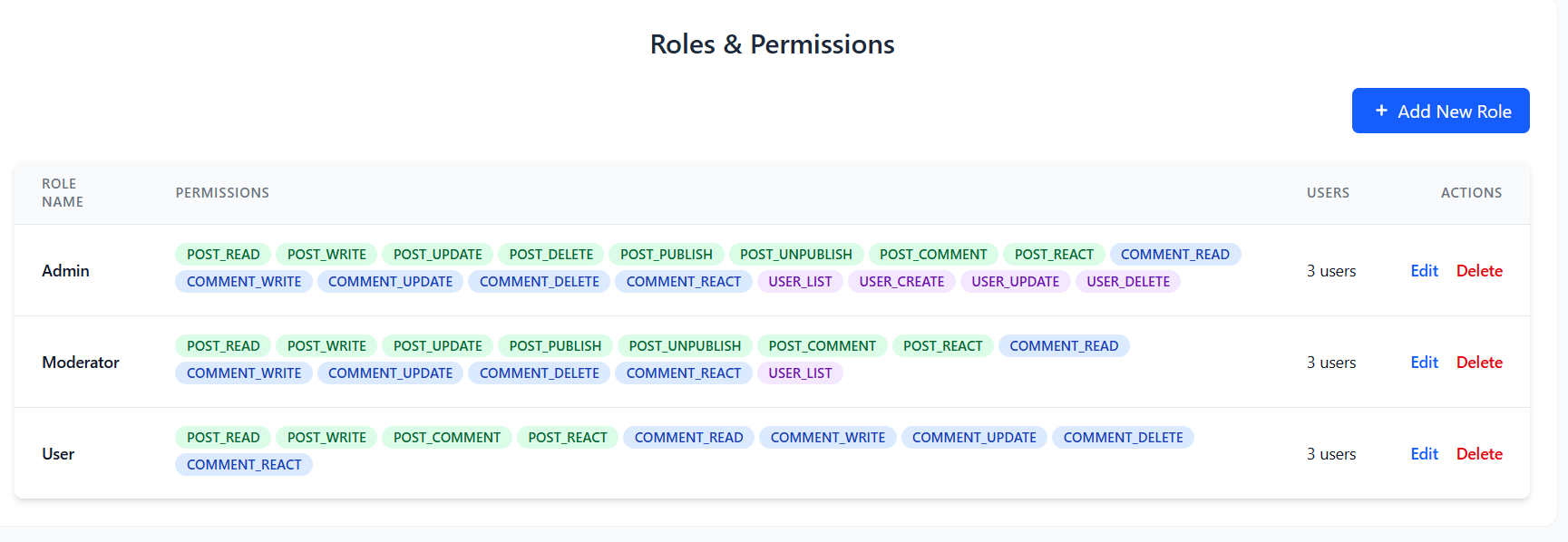
##### 4.8.6.2.2 Classes/Component Description

#### 4.8.6.3 Object Interactions

### 

### 4.8.7 View roles & permissions

#### 4.8.7.1 Screen Design



**Figure 4.8.7**: Screen Mockup of view roles and permissions

**Table 4.8.7: Screen Definition**

| **#** | **Object/Control Name** | **Type** | **Mandatory** | **Max Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | addNewRoleButton | Button | Yes | N/A | A primary button that opens a form or modal to create a new role and assign permissions to it. |
| 2 | roleName | Text | Yes | 50 | The unique name of the role (e.g., Admin, Moderator, User). This is the primary identifier for the role. |
| 3 | permissions | Chip List | No | N/A | A collection of colored tags, where each tag represents a specific permission (e.g., POST\_READ, USER\_DELETE) granted to that role. |
| 4 | users | Text | Yes | N/A | A text display showing the total count of users currently assigned to the corresponding role. |
| 5 | actions | Buttons | Yes | N/A | A set of actions ("Edit", "Delete") that can be performed on the role to modify its name/permissions or remove it from the system. |

#### 4.8.7.2 Class Design

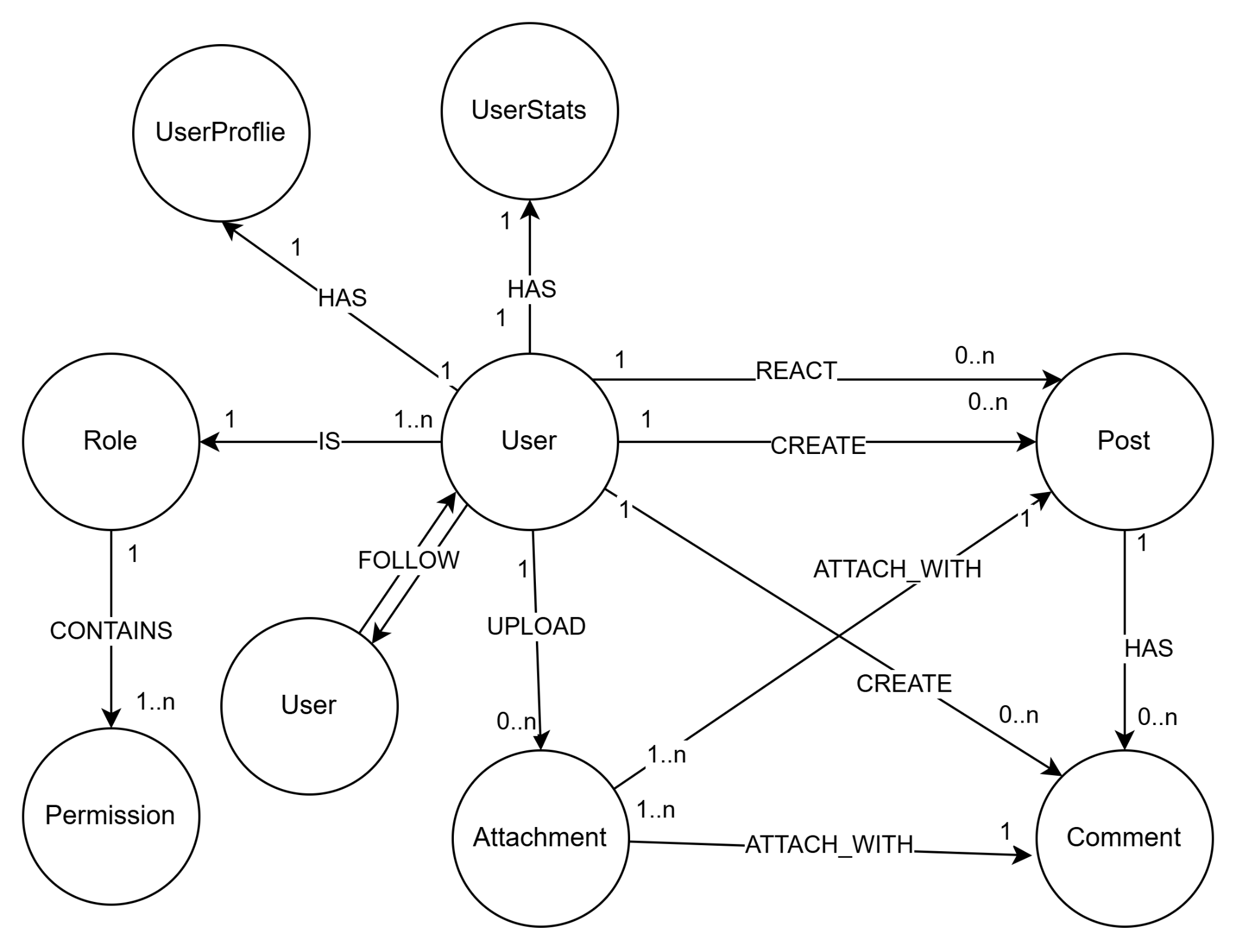
##### 4.8.7.2.1 Classes/Component Structure

##### 4.8.7.2.2 Classes/Component Description

#### 4.8.7.3 Object Interactions

# Chapter 5. Database Design

## 5.1 Database Design



## 5.2 Database Detail

### 5.2.1 User

| **#** | **Attribute name** | **PK** | **Type** | **Mandatory** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | id | x | uuid | Yes | Primary key that uniquely identifies each user. |
| 2 | user\_name |  | string | Yes | Display name of the user |
| 3 | email |  | string | Yes | User’s email address |
| 4 | password |  | string (hashed) | Yes | Encrypted password |
| 5 | created\_at |  | timestamp | Yes | Timestamp when the user was created. |
| 6 | updated\_at |  | timestamp | No | Timestamp when the user was updated. |
| 7 | deleted\_at |  | timestamp | No | Soft-delete timestamp. If set, the account is considered deleted. |
| 8 | is\_locked |  | boolean | Yes (default: false) | Indicates whether the user's account is locked |
| 9 | profile |  | Reference | Yes | Reference to user’s profile |
| 10 | stats |  | Reference | Yes | Reference to user’s stats |
| 11 | followers |  | Set<Reference> | No | A set of references to other users who follow this user. |
| 12 | followings |  | Set<Reference> | No | A set of references to other users this user follows |
| 13 | uid |  | string | Yes | Automatic unique generated internal id |

### 5.2.2 UserProfile

| **#** | **Attribute name** | **PK** | **Type** | **Mandatory** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | uid | x | string | Yes | Automatic unique generated profile id |
| 2 | avatar |  | string | Yes | URL or reference to avatar image |
| 3 | bio |  | string | No | User’s biography |
| 4 | user\_id |  | Reference | Yes | Associated user |

### 5.2.3 UserStats

| **#** | **Attribute name** | **PK** | **Type** | **Mandatory** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | uid | x | string | Yes | Automatic unique generated profile id |
| 2 | num\_posts |  | int | Yes | Number of posts made by the user |
| 3 | num\_following |  | int | Yes | Number of users the user is following |
| 4 | num\_followers |  | int | Yes | Number of followers |
| 5 | user\_id |  | Reference | Yes | Associated user |

### 5.2.4 Post

| **#** | **Attribute name** | **PK** | **Type** | **Mandatory** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | id | x | uuid | Yes | Primary key that uniquely identifies the posts. |
| 2 | caption |  | string | Yes | The content of the posts |
| 3 | created\_at |  | timestamp | Yes | Time when the post was created |
| 4 | visibility |  | enum | Yes | Visibility setting of the post (‘PUBLIC’,‘PRIVATE’, ‘FRIENDS’) |
| 5 | deleted\_at |  | timestamp | No | Time when post is deleted. |
| 6 | updated\_at |  | timestamp | No | Timestamp when the post was updated. |
| 7 | author\_id |  | Reference | Yes | Reference to user who create the post |

### 5.2.5 Comment

| **#** | **Attribute name** | **PK** | **Type** | **Mandatory** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | id | x | uuid | Yes | Primary key that uniquely identifies the comments. |
| 2 | content |  | string | Yes | The text content of the comment |
| 3 | created\_at |  | timestamp | Yes | Time when comment was created |
| 4 | update\_at |  | timestamp | No | Timestamp when the comment was updated. |
| 5 | deleted\_at |  | timestamp | Yes | Timestamp when the comment was created. |
| 6 | post\_id |  | Reference | Yes | Associated post |
| 7 | user\_id |  | Reference | Yes | User who wrote the comment |

### 5.2.6 Attachment

| **#** | **Attribute name** | **PK** | **Type** | **Mandatory** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | id | x | uuid | Yes | Primary key that uniquely identifies the attachment. |
| 2 | title |  | String | No | The title of the attachment |
| 3 | description |  | string | Yes | The detail text for the attachment |
| 4 | original\_link |  | string | Yes | Link to the original file (full size, original quality) |
| 5 | created\_at |  | datetime | Yes | Time when the attachment was created. |
| 6 | metadata |  | json | No | Additional information (e.g. size, resolution, duration, format). |
| 7 | optimized\_link |  | string | No | Link to the optimized,compressed version. |
| 8 | delete\_at |  | timestamp | No | Time when the attachment is deleted |

### 5.2.7 Role

| **#** | **Attribute name** | **PK** | **Type** | **Mandatory** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | uid | X | string | Yes | Unique ID of the role |
| 2 | name |  | enum | Yes | Role name (default: ‘GUEST’, ‘USER’, ‘ADMIN’) |

### 5.2.8 Permission

| **#** | **Attribute name** | **PK** | **Type** | **Mandatory** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | uid | X | string | Yes | Unique ID of the permission |
| 2 | name |  | string | Yes | Name of permission |
| 3 | role\_id |  | Reference | Yes | Which role this permission belongs to. |

### 