

**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

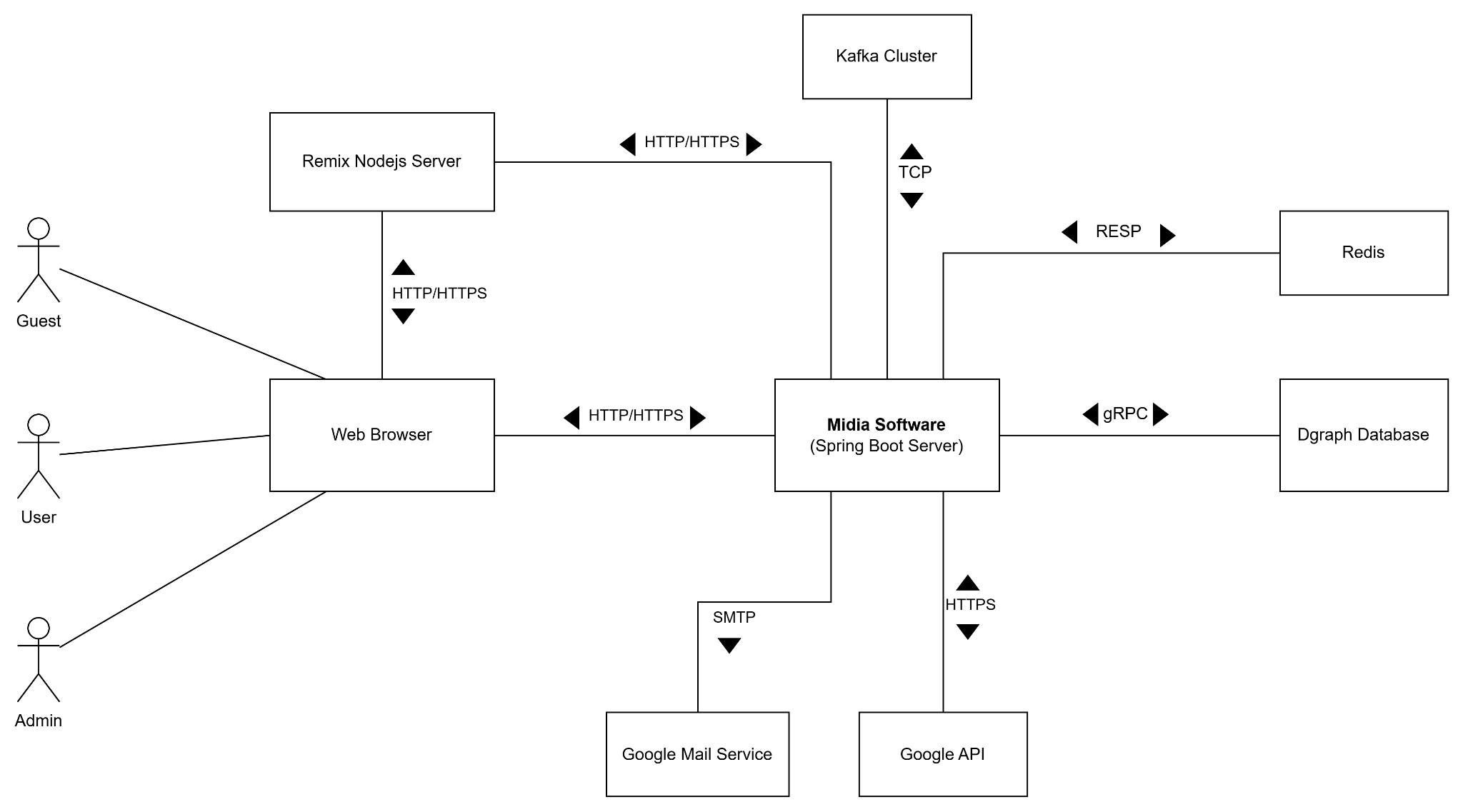
This document specifies the detailed software design for the Midia social‑media web application (React.js frontend, Spring Boot & Dgraph backend). It provides the architectural overview, module designs, data models, and component interactions required by the development team to implement Release 1.0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

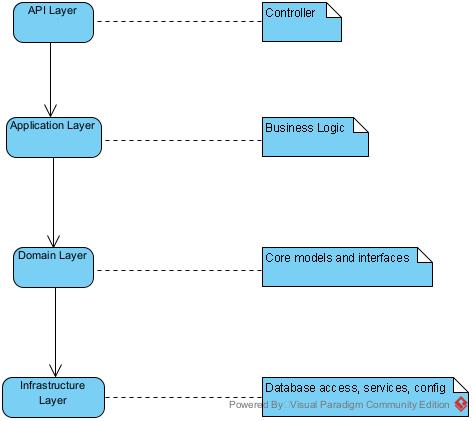
* Components: Frontend (ReactJS), Backend (Spring Boot), Database (PostgreSQL + DGraph),.
* Represented with Deployment Diagrams.

**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**

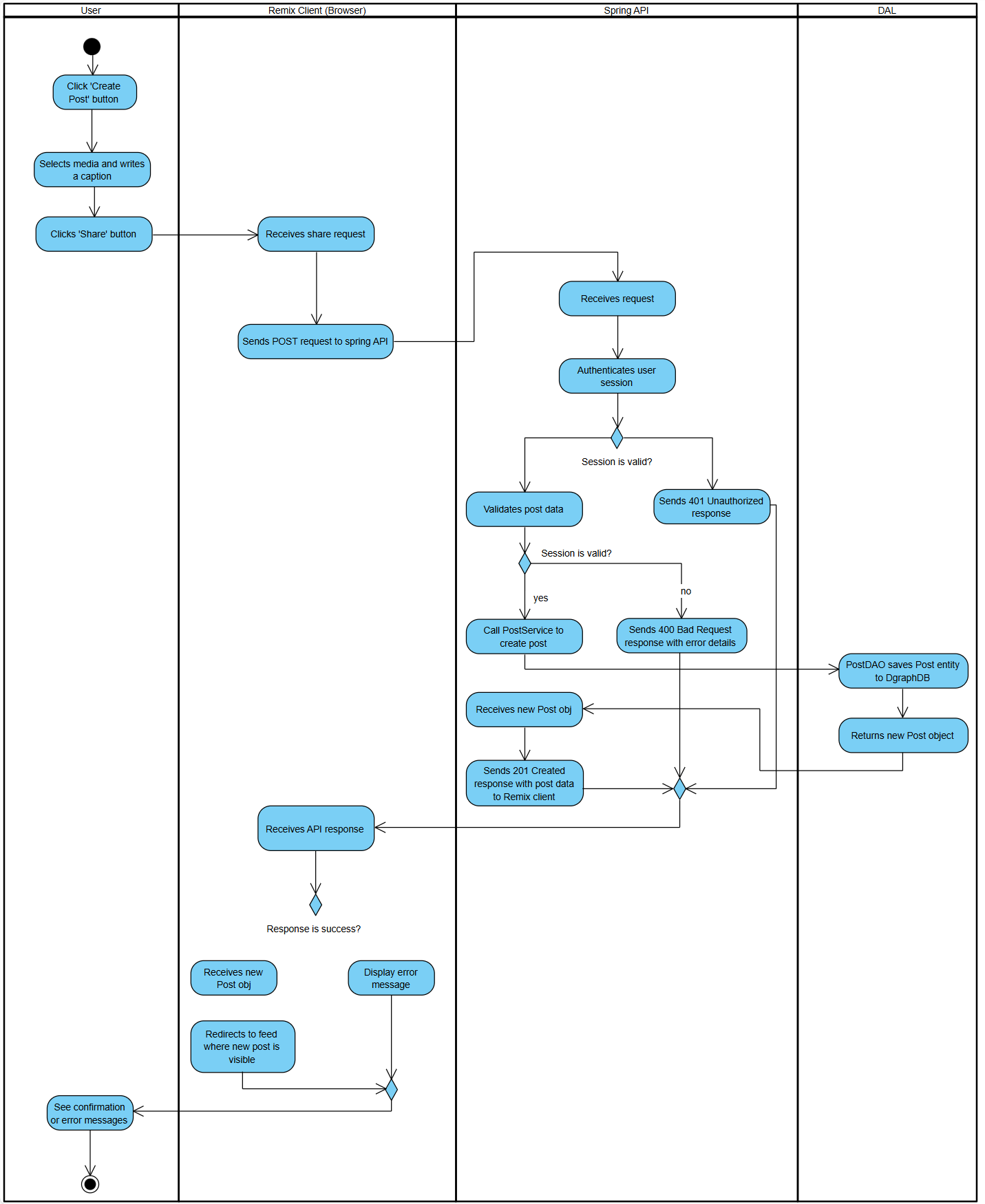
* **Package**: infrastructure
* **Responsibilities:**
  + Implements repository interfaces.
  + Provides database access (e.g., Dgraph), external APIs, configurations.

### 

### 3.2.1 Process View

##### 3.2.1.1 Create Post

##### 3.2.1.2 Create Post Process

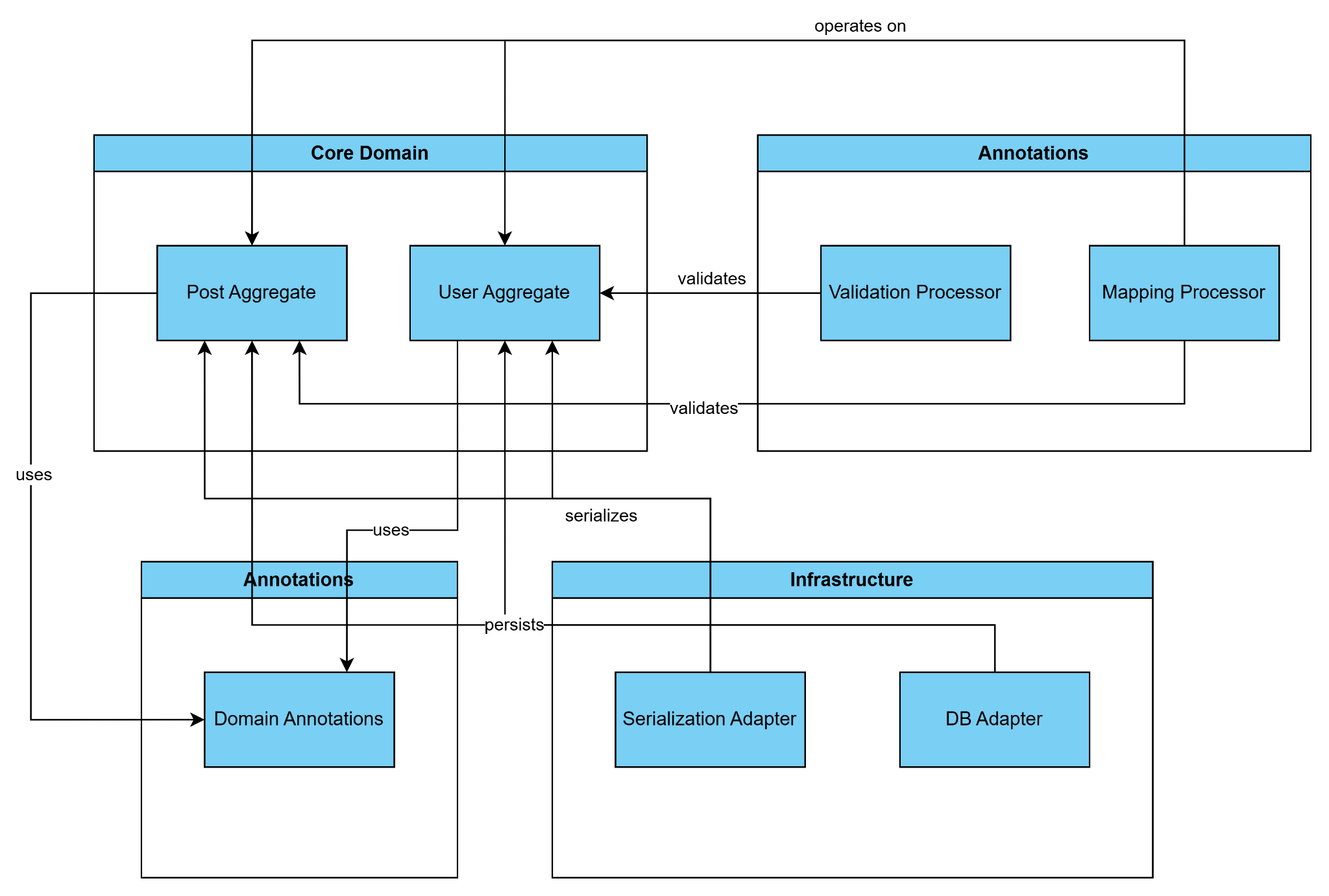


**Figure 3-3**: Create Post process

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

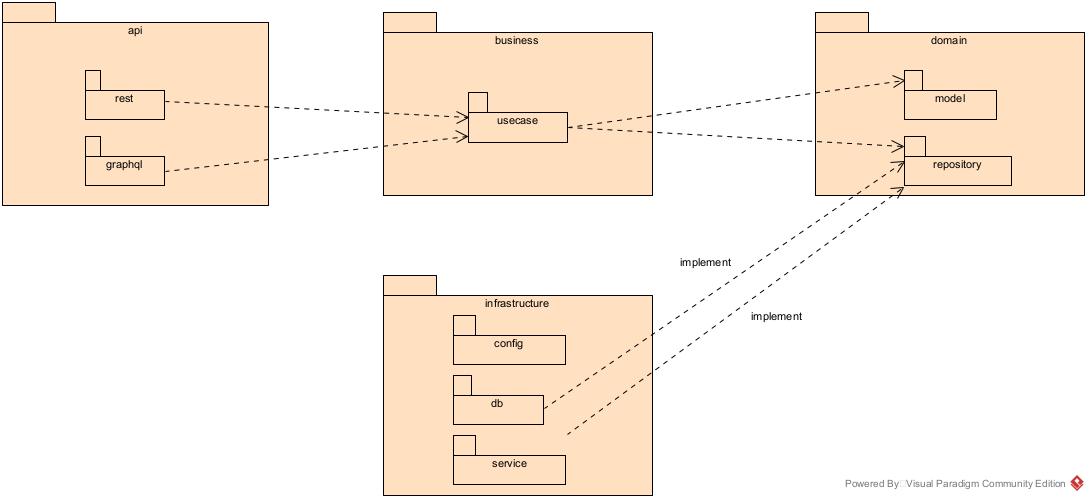


**Figure 3-1**: FAP Logical Overview

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 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 |

### 3.2.4 Deployment View

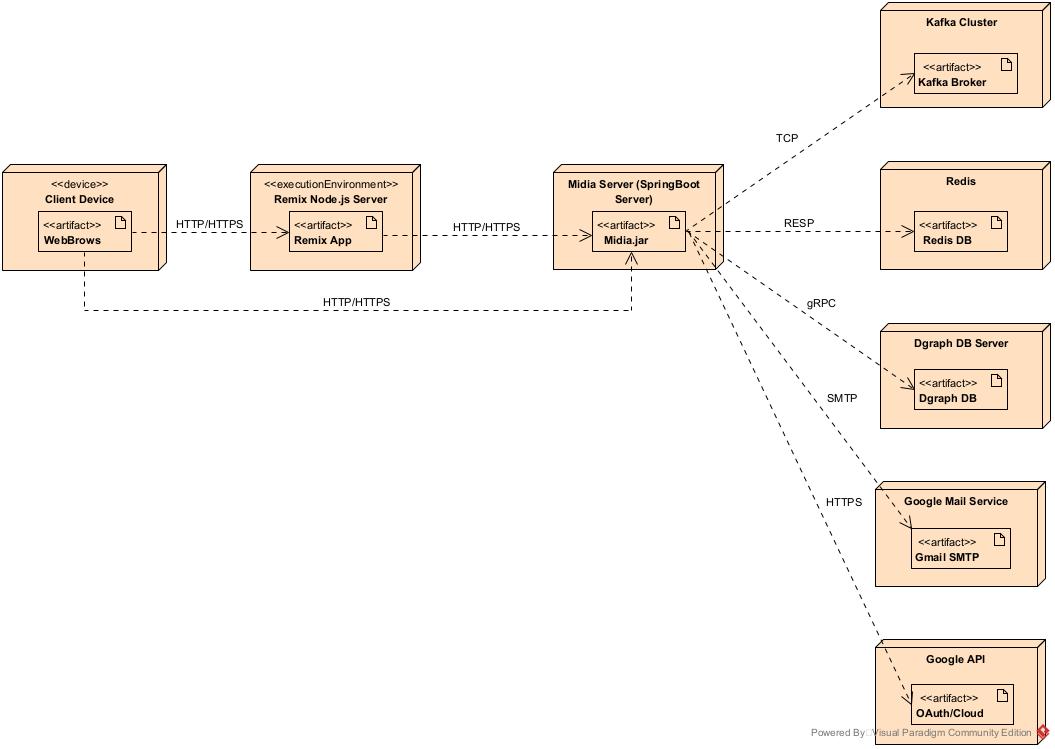
**

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

## 4.2 User Settings

### 4.2.1 User Settings

#### 4.2.1.1 Screen Design

##### 

**Figure 4-2**: Screen Design of User Settings

**Table 4-2:** **User Settings Definition**

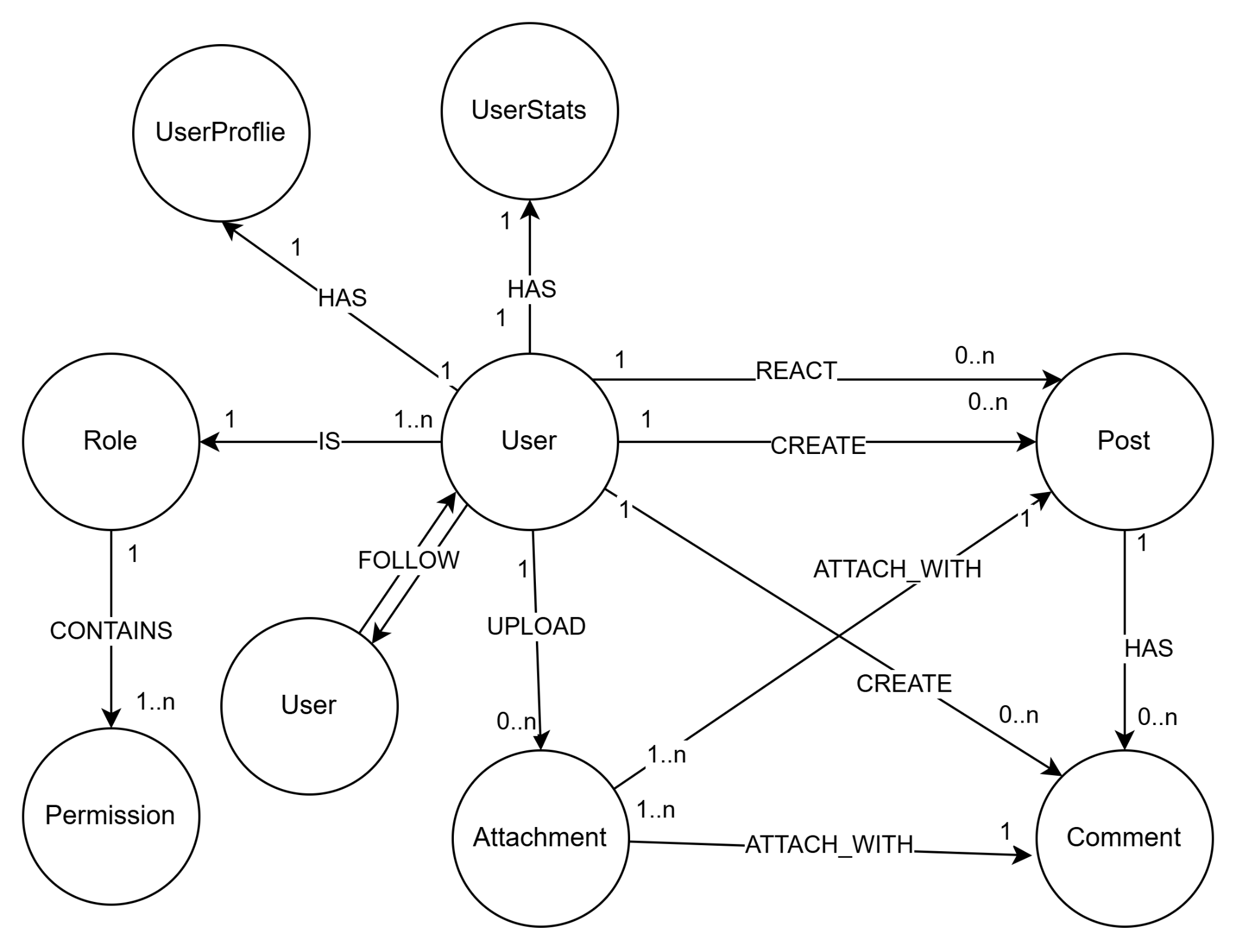
| **No** | **Object/Control Name** | **Type** | **Required** | **Length** | **Description** |
| --- | --- | --- | --- | --- | --- |
| 1 | Name | Text | Yes | 20 | Display real name of user who has logged in |
| 2 | Username | Text | Yes | 20 | Display username of account which has logged in |
| 3 | Website | Text | No |  | Display users’ website link. This field can be edited if users are using a mobile app and optional |
| 4 | Bio | Text | No |  | Display short biography of users. Users can only type about 150 characters. This field is optional |
| 5 | Email | Text | Yes |  | Display user’s email. Email must follow the format <name>@<domain>. This field cannot be edited |
| 6 | Phone number | Text | Yes | 11 | Display user’s phone number. Phone number must contain 11 digits. |
| 7 | Gender | Dropdown box | Yes |  | Display the user's gender. This field contains Male, Female and Prefer not to say |
| 8 | Show account suggestions | Checkbox | No |  | Allow users to see similar account suggestions on their profiles |
| 9 | Learn more | Link | Yes |  | Display the link that navigates to the information about changing username policies |
| 10 | Change profile photo | Link | Yes |  | Display the link that navigates to change profile photo screen |
| 11 | Submit | Button | Yes |  | Allow to save all changes |
| 12 | Temporarily deactivate my account | Link | Yes |  | Allow users to deactivate their account if they want |
| 13 | Edit profile | List item | Yes |  | Allow users to open the edit profile box. They can change their information stored in the system |

### 4.2.1.2 Class/Component Design

#### 4.2.2.1 Classes/Component Structure

# 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. |

### 