System Modeling and Design

CRAISELION : Handong Team Meeting Archiving and Exchange Web Platform

Version:	1.0
Date:	2024-05-11
Client:	Student
Confidential:	No
developer:	our name

Team

Name	Email	ID	Role
Seokjae Ma	21800239@handong.ac.kr	21800239	Project Manager
Donggyu Kim	22000063@handong.ac.kr	22000063	Scrum Master
Sechang Jang	21900628@handong.ac.kr	21900628	Documenter
Junhyeok Choi	21900764@handong.ac.kr	21900764	Developer
MinSeo Lee	22100503@handong.ac.kr	22100503	Developer

Version

Version	Date	Description
1.0	11-05-24	First version

Table of Contents

1 Introduction

1.1 Project Background

Handong University hosts a variety of Residential Colleges (RCs), within which numerous team meetings are regularly held. These meetings play a crucial role in fostering community formation and interaction among students. However, the current system faces several significant challenges.

First, there are difficulties in exchanging information between teams within the same RC, and even more so between different RCs. This lack of interaction within and between RCs hinders the development of a community-centered culture at Handong, which the university aims to promote.

Second, there is a scarcity of guidance provided to team leaders, leading to challenges in leadership and team management. This situation diminishes the efficiency and effectiveness of team activities, negatively affecting student engagement and enthusiasm.

Third, the preservation of materials and information generated during team meetings throughout the year is not adequately addressed. Due to the absence of a proper archiving system, important documents are often lost, posing significant challenges to long-term project management and the reuse of materials.

To resolve these issues, we propose a web platform that facilitates communication and exchange among teams or RCs and preserves information and materials related to team meetings, thereby aiding in the development of the entire community at Handong University. This platform will serve as an essential resource for the student support team and the students (team executives), who are the main stakeholders, in enhancing the team culture and leadership development within Handong University.

2. SYSTEM CONTEXT & INTERACTION

2.1 General Overview

Handong University is taking a significant step forward in enhancing community interaction and team collaboration within its Residential Colleges (RCs). The university recognizes the critical role that efficient communication and proper management of team meetings play in fostering a community-centered culture. To address the existing challenges such as poor information exchange between teams, lack of guidance for team leaders, and insufficient archiving of meeting materials, the university proposes to develop a dedicated web platform.

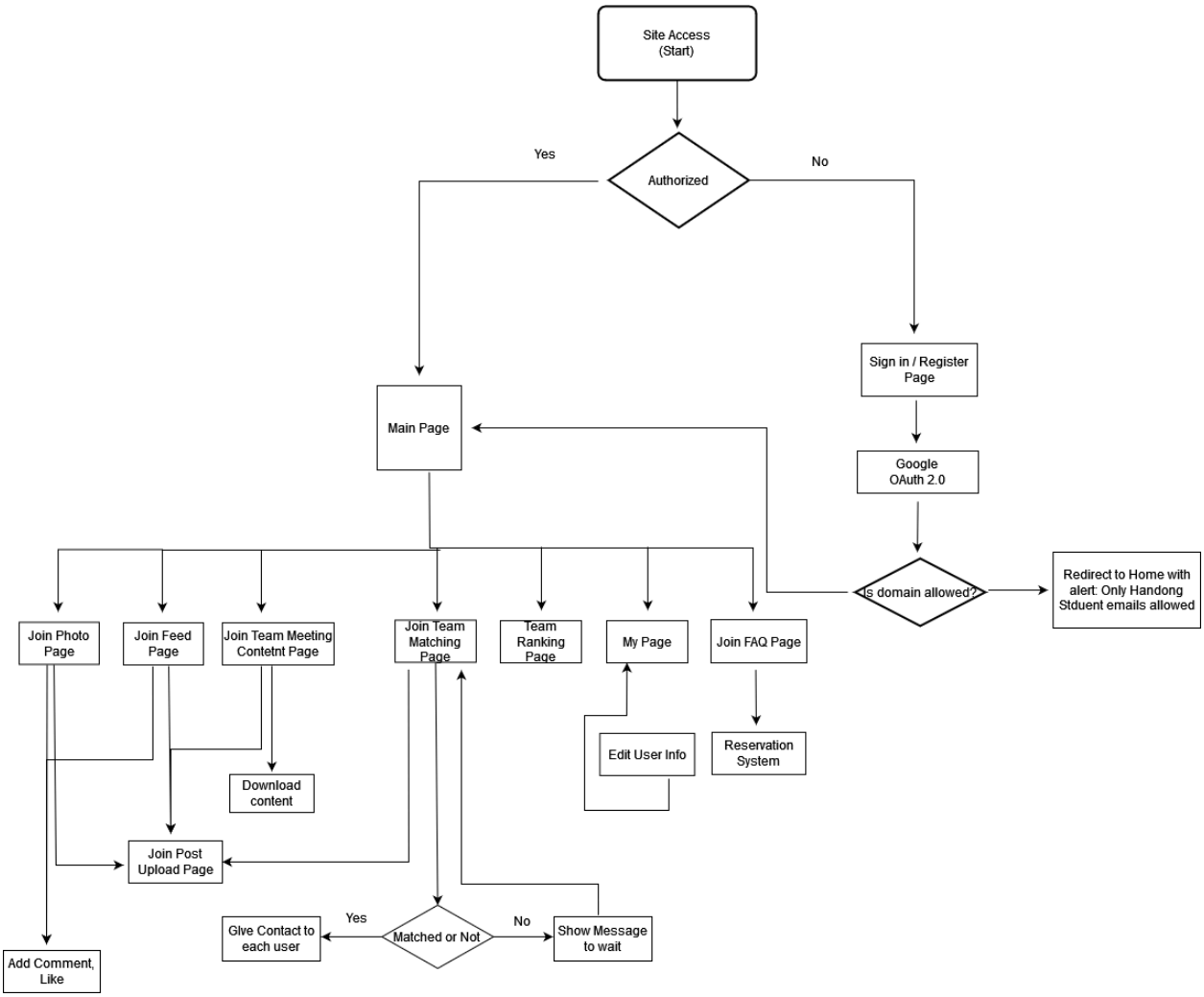
Purpose of the Platform: The primary objective of this web platform is to facilitate seamless communication across different teams and RCs, provide essential tools and resources for team leaders, and establish a robust archiving system for storing and managing meeting documents and materials. By doing so, the platform will not only enhance the efficiency of team activities but also boost student engagement and leadership development across the university.

A **flowchart** is a type of diagram that represents a workflow or process. It shows the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting, or managing a process or program in various fields.

The flowchart in the image you provided represents the process of accessing and navigating a website:

1. **Site Access**: The process begins with accessing the site.
2. **Authorization**: The system checks if the user is authorized. If not, the user is directed to the “Sign in / Register Page” or uses “Google OAuth 2.0”.
3. **Main Page**: Once authorized, the user is directed to the main page. From here, the user can navigate to various other pages like “Join Photo Page”, “Join Feed Page”, “Join Team Board/Content Page”, etc.

4. **User Interaction**: The user can download content, add comments, and like posts. Some pages also allow users to upload content.
5. **User Info and Reservation System**: Users can edit their info and use the reservation system.



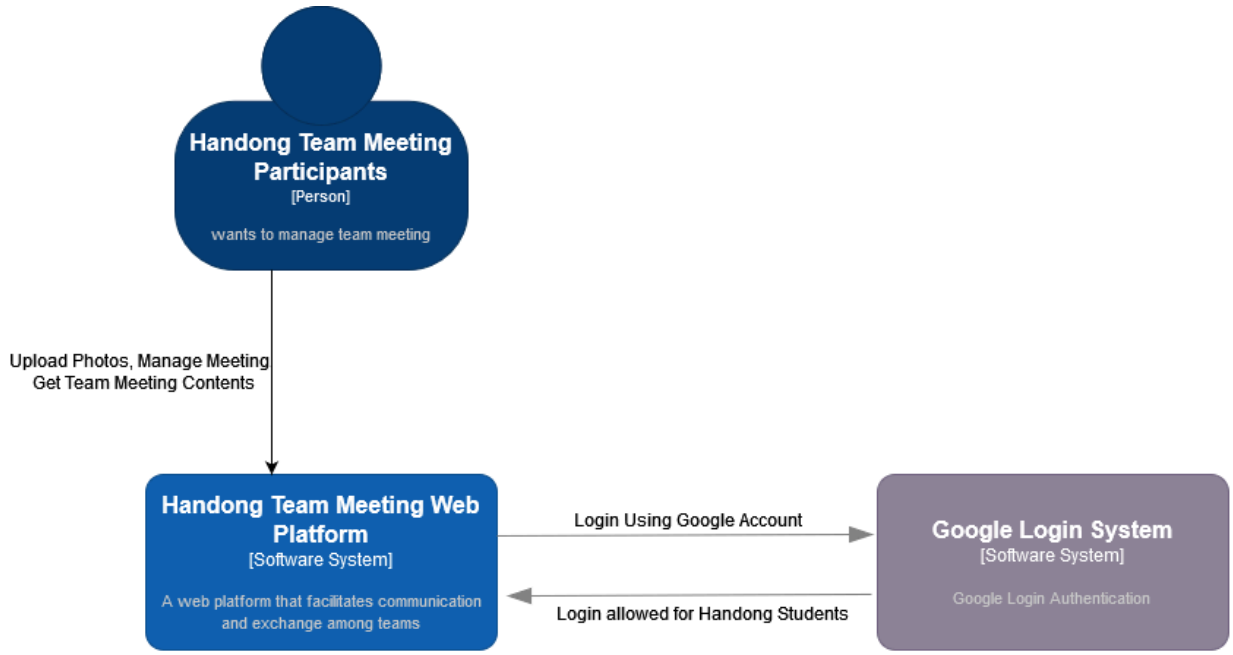
3.1 C4 Model

The detailed technical design is constructed using the C4 model. This slowly zooms in on the different parts of the application. This model is divided into four layers, which are context, containers, components, and code. We are going to draw for Context, Container, and Component for system modeling and design.

In SYSTEM CONTEXT & INTERACTION section, we are going to see the context part.

3.1.1 Context Diagram

- **System:** Handong University Team Collaboration Platform
- **Purpose:** Facilitate communication and collaboration within and across Residential Colleges (RCs), improve leadership guidance, and archive essential documents.
- **Primary Users:** Students (team executives) and Handong Students



Level 1. Context Diagram

3. ARCHITECTURAL DESIGN

3.1 Technical Stack and Framework

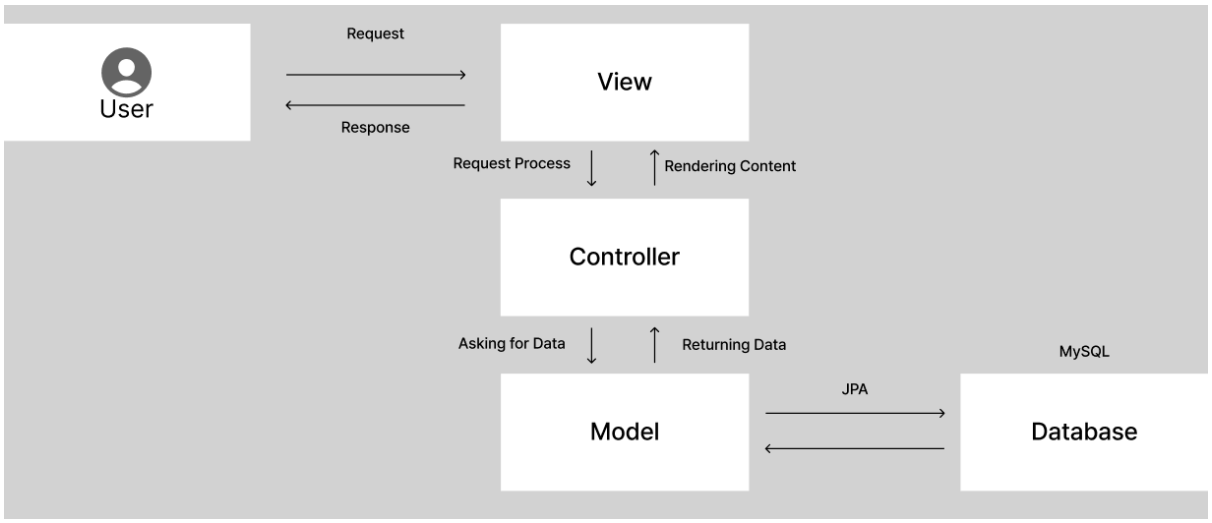
1. **Frontend Development:** The user interface of the platform will be built using **React**, renowned for its scalability and flexibility. React's component-based architecture will allow for a responsive, user-friendly experience across various devices, catering to the needs of students and team leaders alike.
2. **Backend Services:** The backend of the platform will be developed using **Spring Boot**, chosen for its rapid development capabilities and its extensive suite of features for building robust web services. Spring Boot will facilitate easy integration with other systems and databases at Handong University.
3. **Database Management: MySQL** will serve as the primary database management system due to its reliability and efficiency in handling complex data structures necessary for storing user data, meeting records, and archival materials.
4. **Security and Compliance:** To ensure the security of data transmission and user authentication, **Spring Security** will be implemented. It will provide comprehensive security configurations to safeguard personal and operational data against potential threats.
5. **Document Storage and Archiving:** For managing and archiving digital assets and meeting documents effectively, the platform will integrate a **document management system** that supports tagging, categorization, and easy retrieval of archived materials.

By developing this platform, Handong University aims to create an ecosystem that not only supports but actively promotes a vibrant community culture. This system will be an invaluable tool for students and faculty alike, providing them with the necessary resources to manage their activities more effectively and preserve the wealth of knowledge and collaboration fostered within the university's RCs.

3.2 MVC Pattern

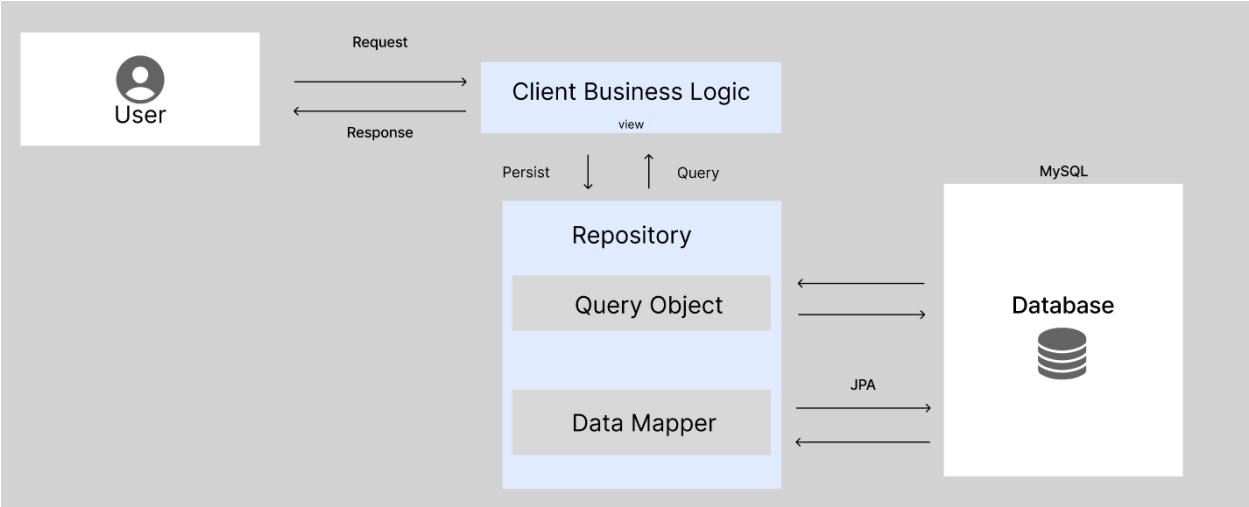
The Model-View-Controller (MVC) is a design pattern used in software engineering. It separates an application into three interconnected components:

- **Model:** This handles data and business logic. It interacts with the database, as shown in your image. The model retrieves data from the database using MySQL and JPA and returns the data to the controller.
- **View**: This presents data to the user. It's responsible for the user interface (UI) components. In the context of the previous flowchart, the view would be responsible for displaying the various pages like "Photo Page", "Feed Page", etc.
- Controller: This acts as an intermediary that handles input from the user and updates the Model and View accordingly. In your image, the controller is shown processing requests and rendering content.



In the context of the previous question, the MVC pattern could be used to structure the website. The flowchart could represent the user's journey through the View, while the Controller handles the logic of what page to display next, and the Model manages the data related to the user and the content of the pages.

3.3 Repository Pattern



The Repository Pattern is a design pattern that mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.

In the diagram you provided, the Repository Pattern is illustrated as follows:

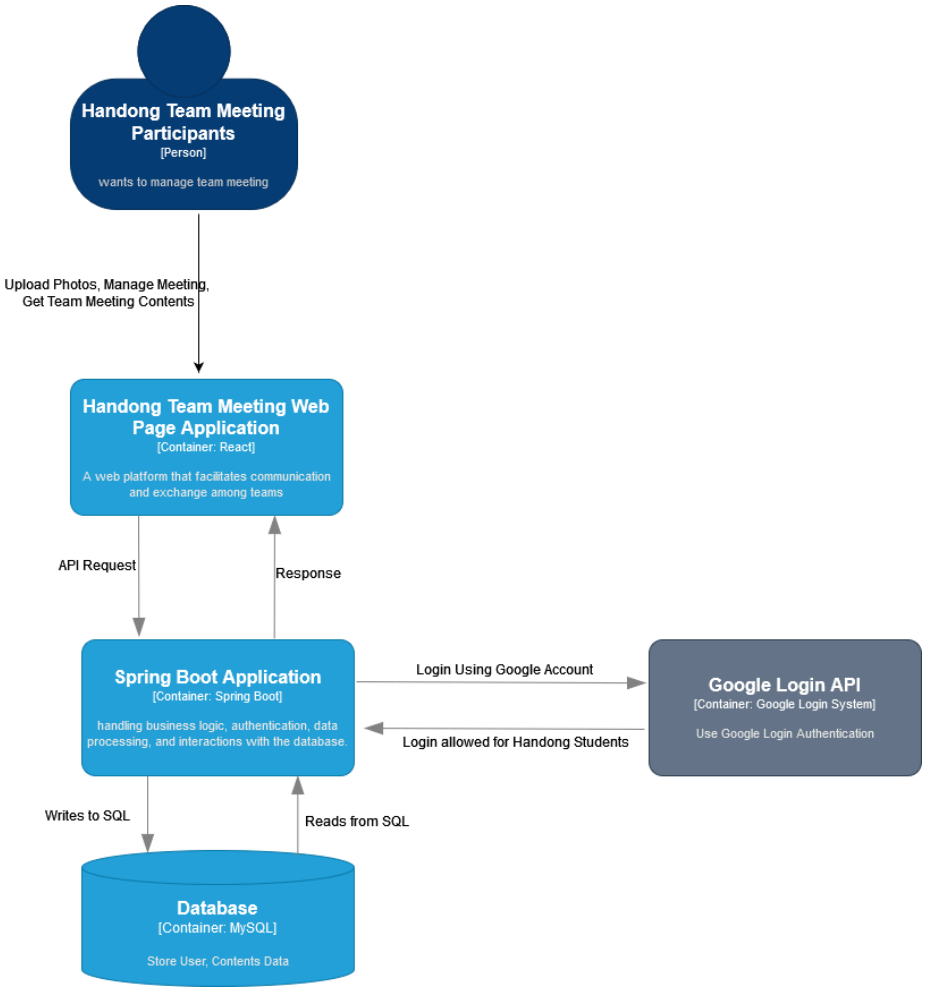
- User: The user interacts with the application, sending requests and receiving responses.
- Client Business Logic: This layer handles the business logic of the application. It receives user requests and interacts with the repository to retrieve or persist data.
- Repository: This is the core of the Repository Pattern. It communicates with the data layer to perform CRUD operations (Create, Read, Update, Delete). Inside the repository, there are several components:
- Query Object: This object handles specific queries to the database.
- Data Mapper: This component maps the data from the database to objects that the application can use, and vice versa.
- JPA (Java Persistence API): This is a standard interface for accessing databases in Java. It's used to interact with the database.
- Database: This is where the data is stored. The database in the diagram is a MySQL database.

4. Object Class Identification

4.1.1 Container Diagram on C4 model

Web Application:

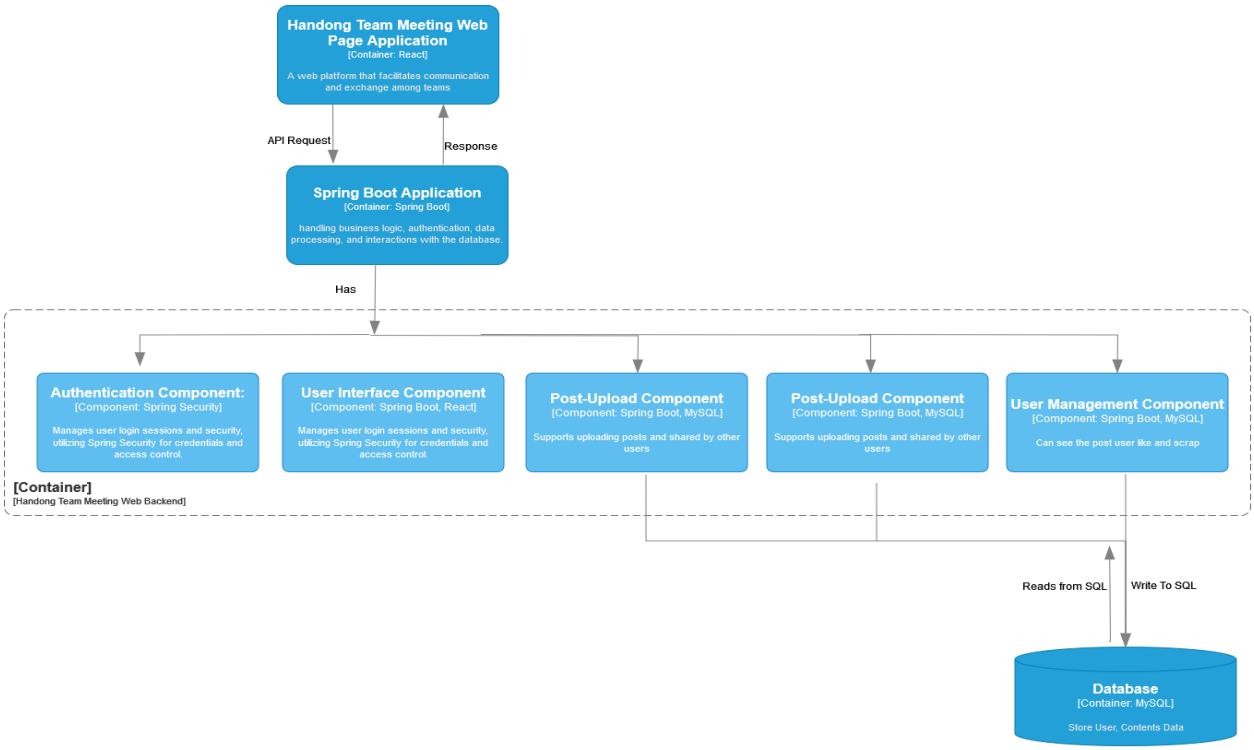
- **Frontend:** Developed with React, serves the dynamic content and interfaces directly with the users through their web browsers.
- **Backend:** Spring Boot application, handling business logic, authentication, data processing, and interactions with the database.
- **Database:** MySQL database, stores user data, meeting records, and archived documents.



4.1.2 Component Diagram on C4

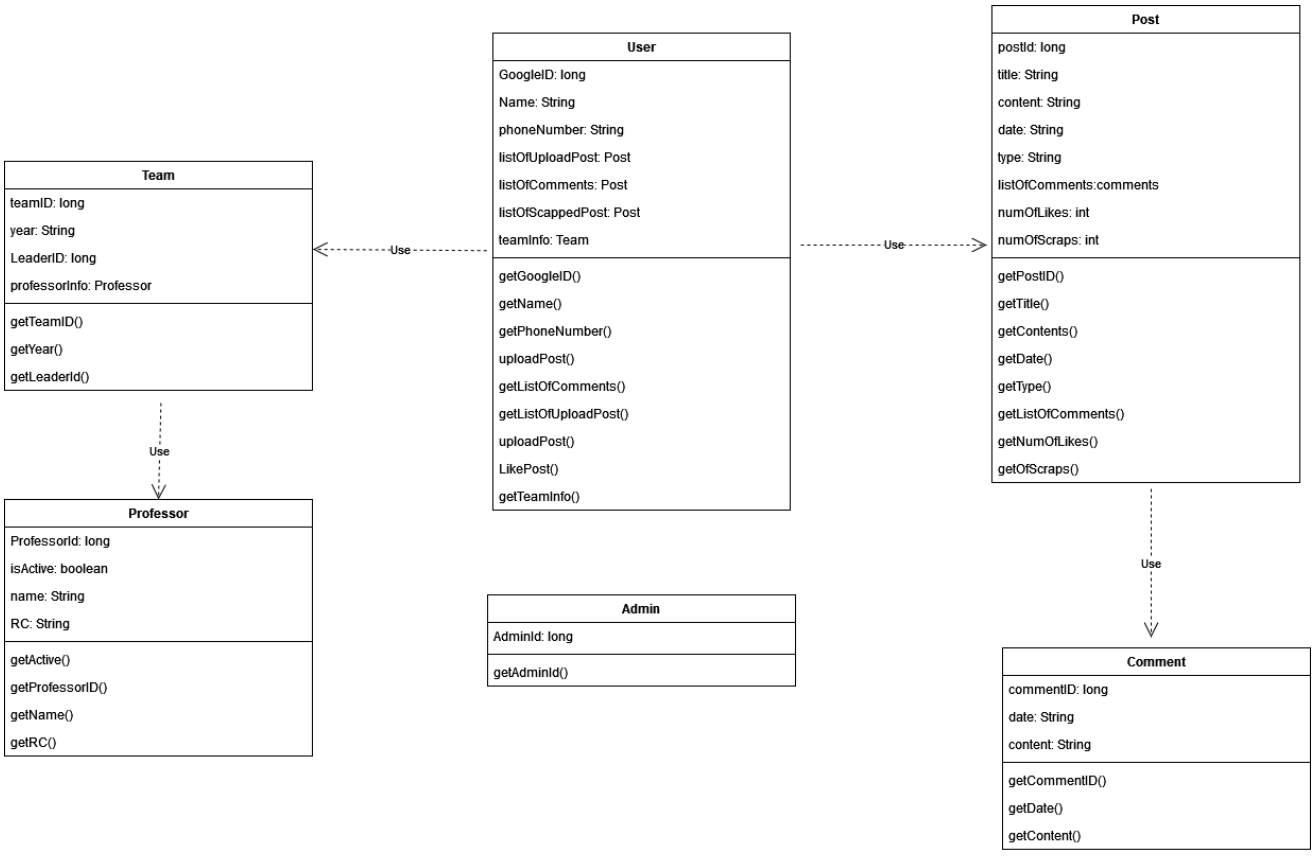
Brief Web Application Components:

- **User Interface Component:** Handles the rendering of the user interface and user interactions.
- **Authentication Component:** Manages user login sessions and security, utilizing Spring Security for credentials and access control.
- **Meeting Management Component:** Supports scheduling, updating, and tracking of inter-team meetings.
- **Document Storage Component:** Interfaces with the Document Management System to facilitate document upload, categorization, and retrieval.
- **Post-Upload Component:** Supports uploading posts and shared by other users
- **User Management Component:** Can see the post user like and scrap



4.2 Class Diagram

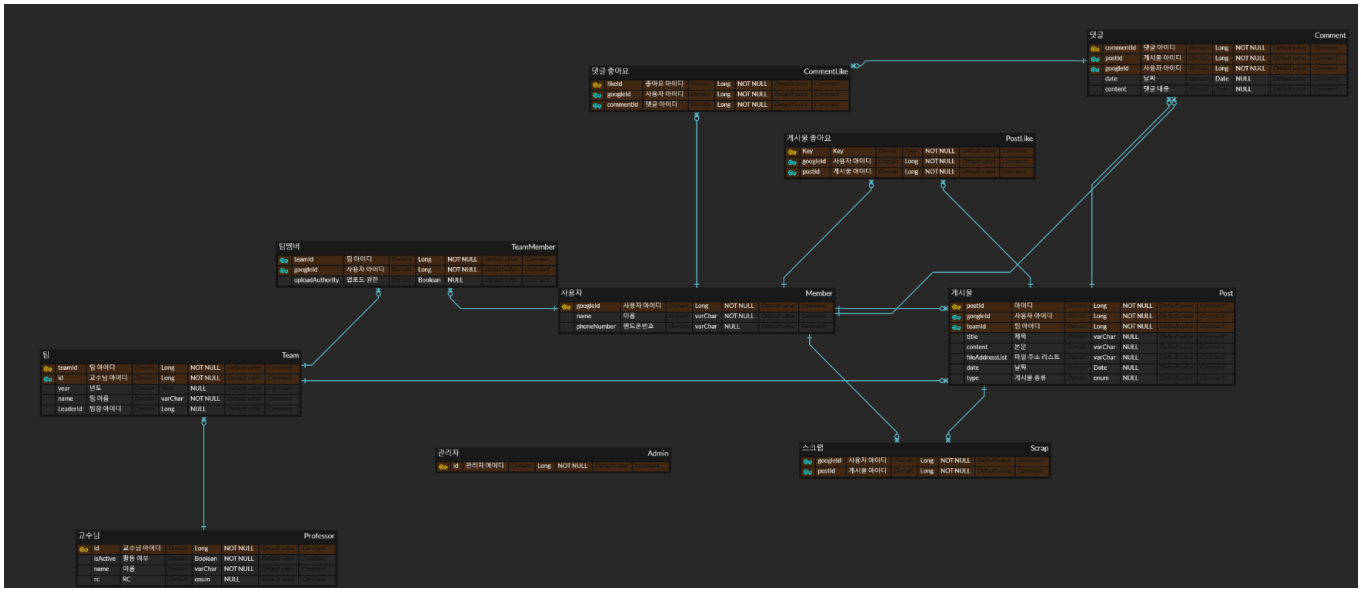
- **User Class**: This class has attributes like GoogleID, Name, and phone number. It has methods for getting ID, name, and phone number, and for uploading posts and comments.
- **Team Class**: This class has attributes like teamID, year, and leaderID. It has methods for getting a team, professor, team ID, and leader ID.
- **Professor Class**: This class inherits from the User class and has additional attributes like professorId, isActive, name, and pic. It has methods for getting professor ID, name, pic, and CO.
- **Admin Class**: This class has an attribute adminId and a method for getting adminId.
- **Post Class**: This class has attributes like postId, title, content, and date. It has methods for getting post ID, title, content, date, backdrop, list of comments, number of likes, and scrapes.
- **Comment Class**: This class has attributes like commentId, date, and content. It has methods for getting comment ID, date, and content.

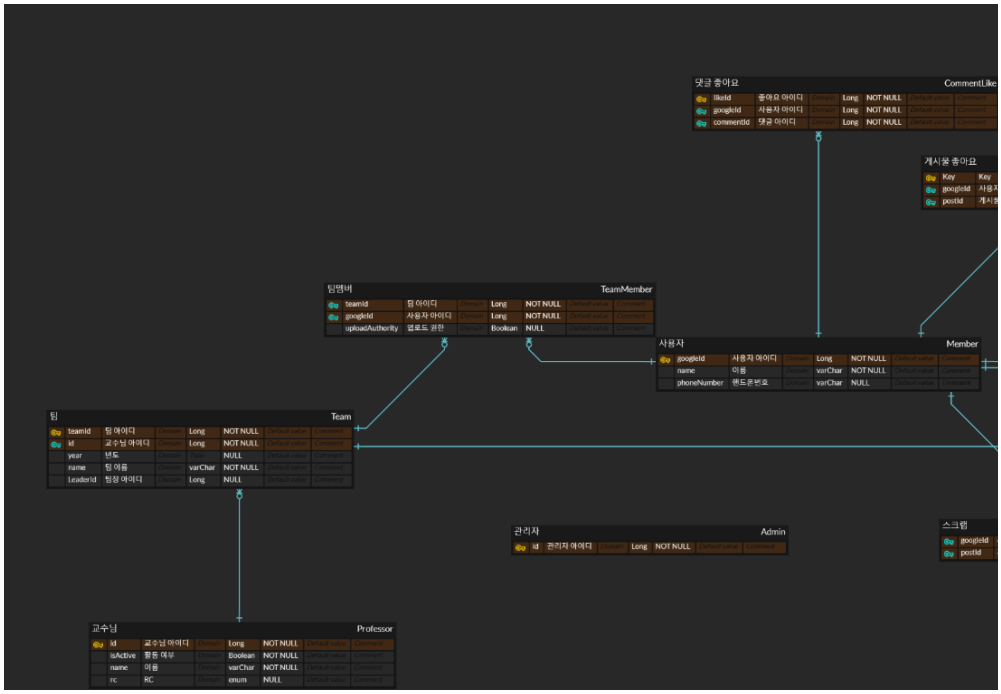


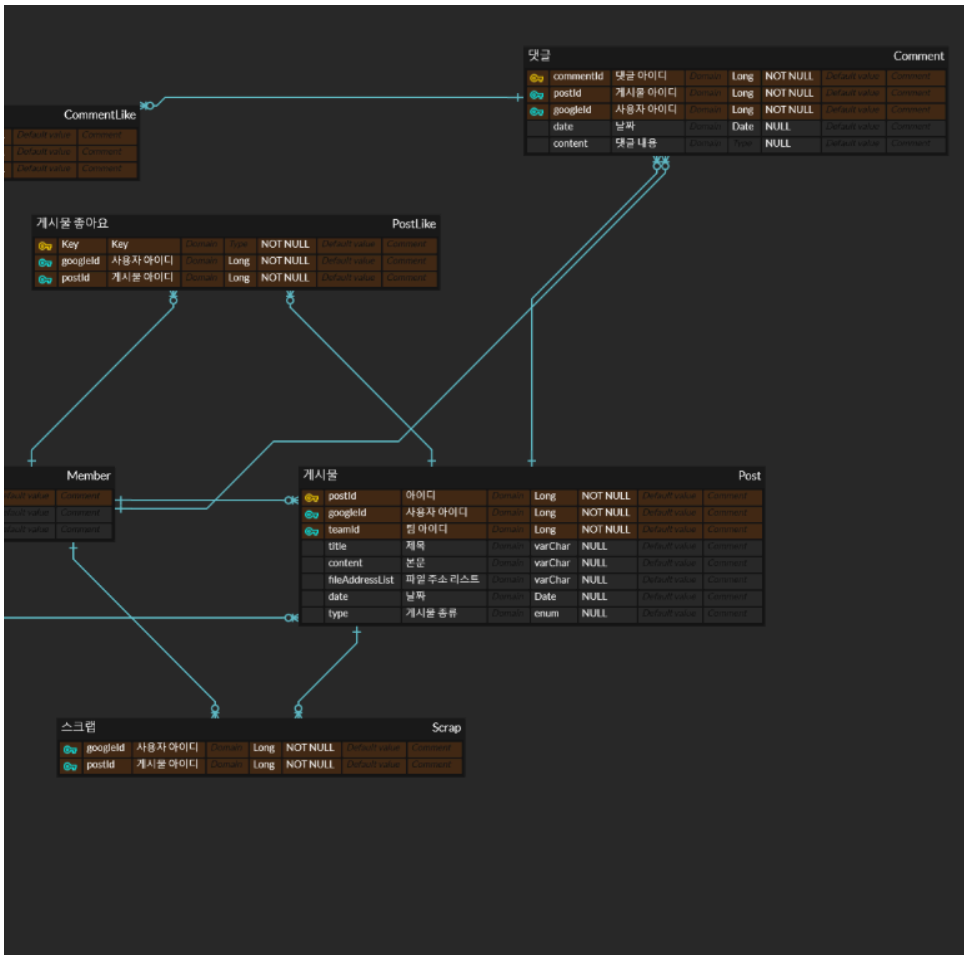
4.3 ER Diagram

This ER Diagram shows the overall architecture of DB.
Each user has their own ID, can upload posts, and can see the posts that they liked and scraped.
Also, it stores team information and professor information.

Refer to the ER diagram for the details.




댓글 좋아요
CommentLike
팀멤버
TeamMember
사용자
Member
팀
Team
관리자
Admin
스크랩
교수님
Professor


CommentLike
댓글
Comment
게시물 좋아요
PostLike
Member
게시물
Post
스크랩
Scrap

4. Design Model

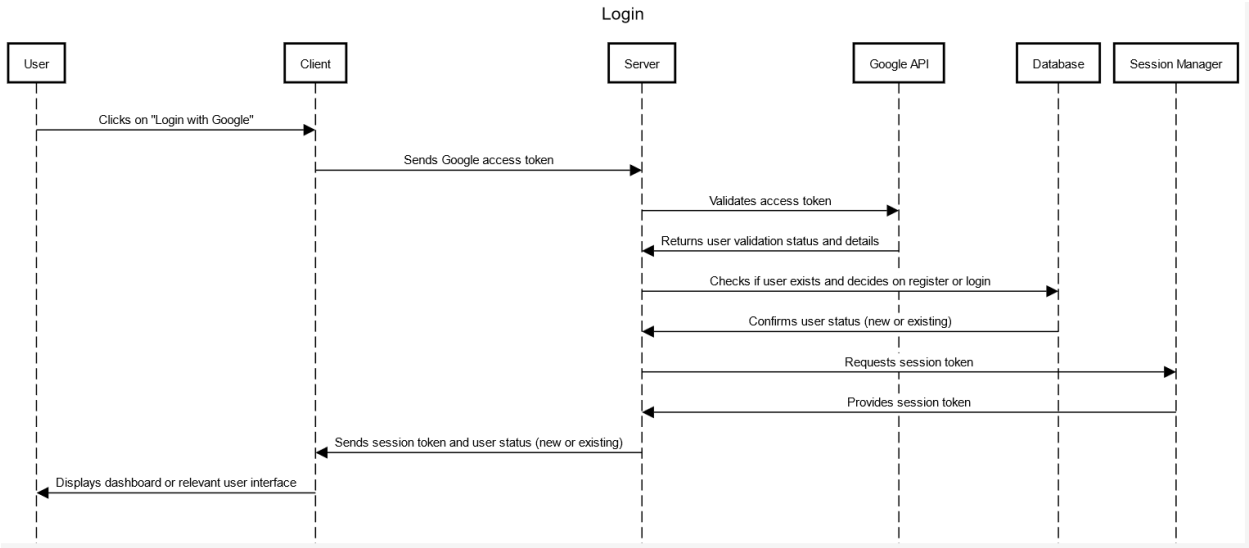
4.1 API Details

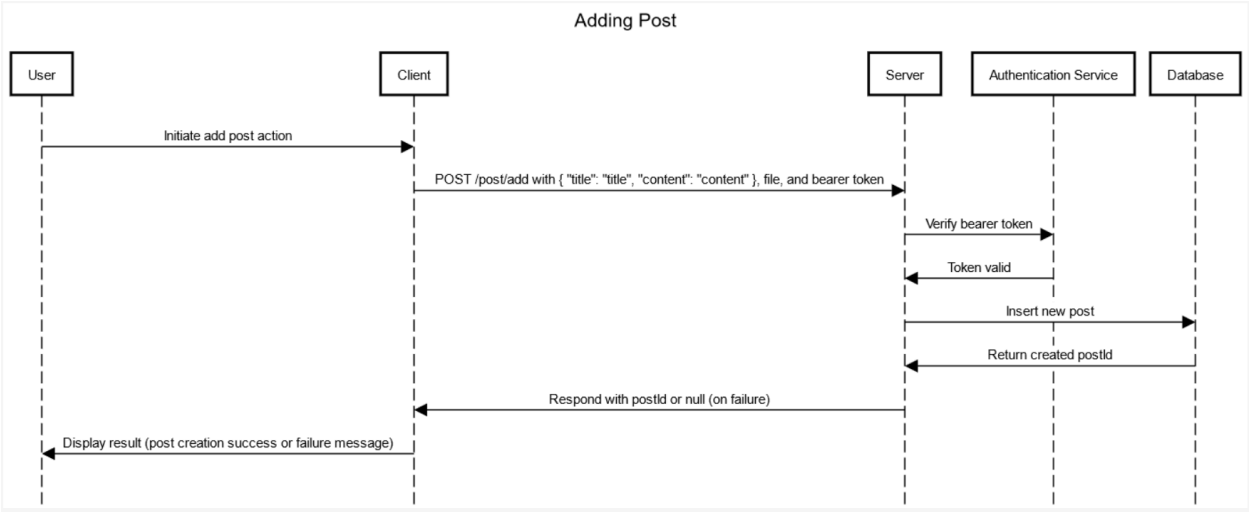
Request URL	Request Type	Authorization	Description	Return Value
code/google/{registerId}	POST	None	Google login endpoint. Retrieves user info from Google server for signup/login	Spring Security token
post/add	POST	Only users with manager permissions	Only users with manager permissions can create posts. The token must be sent to the bearer.	Created postId or null if failed
post/delete/{id}	DELETE	Manager permissions + same team, Admin can delete directly	Request Param for post ID required, along with the token.	Deleted postId or null if failed
post/get/all/{index}	GET	No restrictions	Provides information about all posts, paginated by 10s.	PostResponseList
post/get/{id}	GET	No restrictions	Provides information about a single post.	PostResponse
post/update/{id}	PATCH	Manager permissions + same team, Admin can update directly	Request Param for postId required, the token used to determine user permissions for the update.	Updated postId or null if failed
team	GET	No restrictions	Token required.	PictureResponseList or null if failed
comment/add/{postId}	POST	No restrictions, Token required	Successfully adds a comment and returns comment ID.	Comment id or null if failed
comment/delete/{commentId}	DELETE	Only comment creator and admin	Token required.	Deleted commentId or null if failed
comment/update/{commentId}	PATCH	Only comment creator	Token required.	Updated commentId or null if failed
material/get/all/{index}	GET	No restrictions	Provides information about all posts, paginated by 10s.	PostResponseList
material/get/{id}	GET	No restrictions	Provides information about a single post.	PostResponse
material/add	POST	Only users with manager	Only users with manager permissions can create posts. The token must be sent in the bearer.	Created postId or null if failed

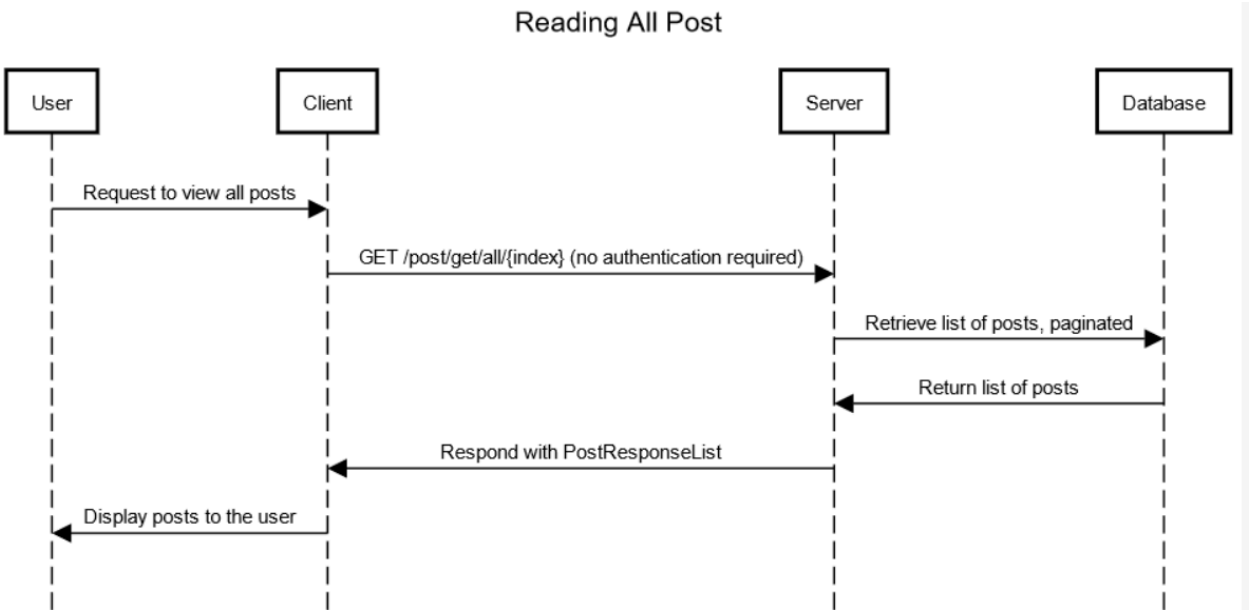
		permissions		
material/delete/{id}	DELETE	Manager permissions + same team, Admin can delete directly	Request Param for post Id required, along with token.	Deleted postId or null if failed
material/update/{id}	PATCH	Manager permissions + same team, Admin can update directly	Request Param for postId required, token used to determine user permissions for the update.	Updated postId or null if failed
faq/get/all/{index}	GET	No restrictions	Provides information about all posts, paginated by 10s.	PostResponseList
meet/get/all/{index}	GET	No restrictions	Provides information about all meeting posts, paginated by 10s.	PostResponseList
meet/get/{id}	GET	No restrictions	Provides detailed information about a specific meeting post.	PostResponse
meet/add	POST	Admin and manager permissions	Only users with admin or manager permissions can create meeting posts. The token must be sent in the bearer.	Created postId or null if failed
meet/delete/{id}	DELETE	Manager + same team, Admin can delete directly	Request Param for post Id required, along with the token.	Deleted postId or null if failed
meet/update/{id}	PATCH	Admin and manager permissions, same team possible	Request Param for postId required, the token used to determine user permissions for the update.	Updated postId or null if failed
like/add/{postId}	POST	No restrictions request param, and tokens required	Adds like to the specified post and returns the postId.	PostId or null if failed
like/delete/{postId}	DELETE	No restrictions request param, and tokens required	Removes like from the specified post and returns the postId.	PostId or null if failed
like/get	GET	No restrictions, token required	Returns a list of posts the user has liked.	PostLikeList or null if failed
star/add/{postId}	POST	No restrictions request param, and tokens required	Adds a star to the specified post and returns the postId.	PostId or null if failed
star/delete/{p	DELETE	No restrictions request param,	Removes a star from the specified post and returns the postId.	PostId or null if failed

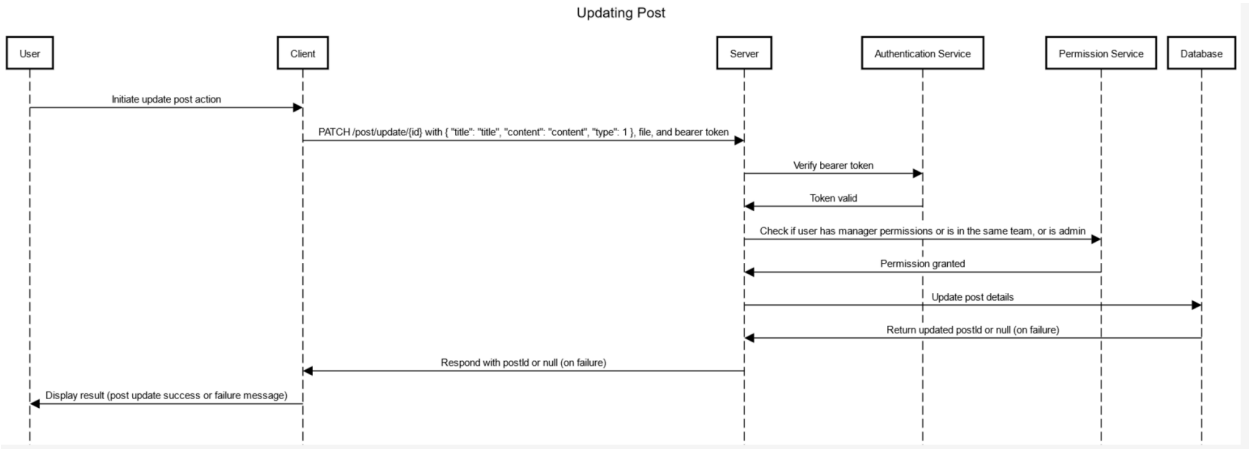
ostId}		and tokens required		
star/get	GET	No restrictions, token required	Returns a list of posts the user has starred.	StarList or null if failed
team/add	POST	No restrictions, token required	Adds a new team with the specified members. The token must be provided.	TeamId or null if failed

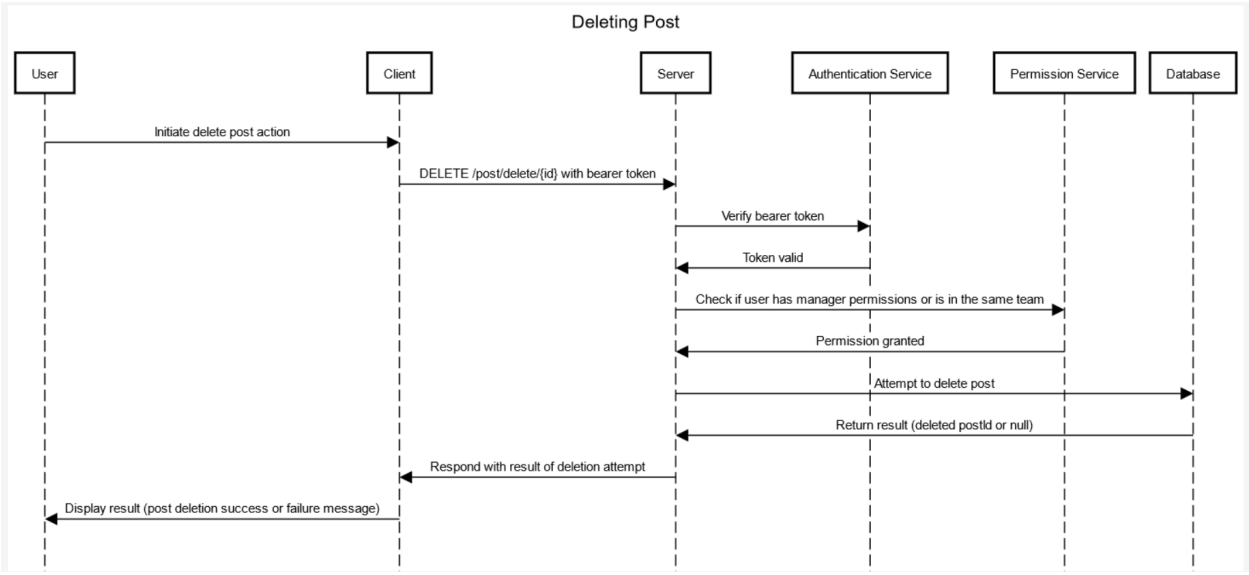
4.2 Sequence Diagram

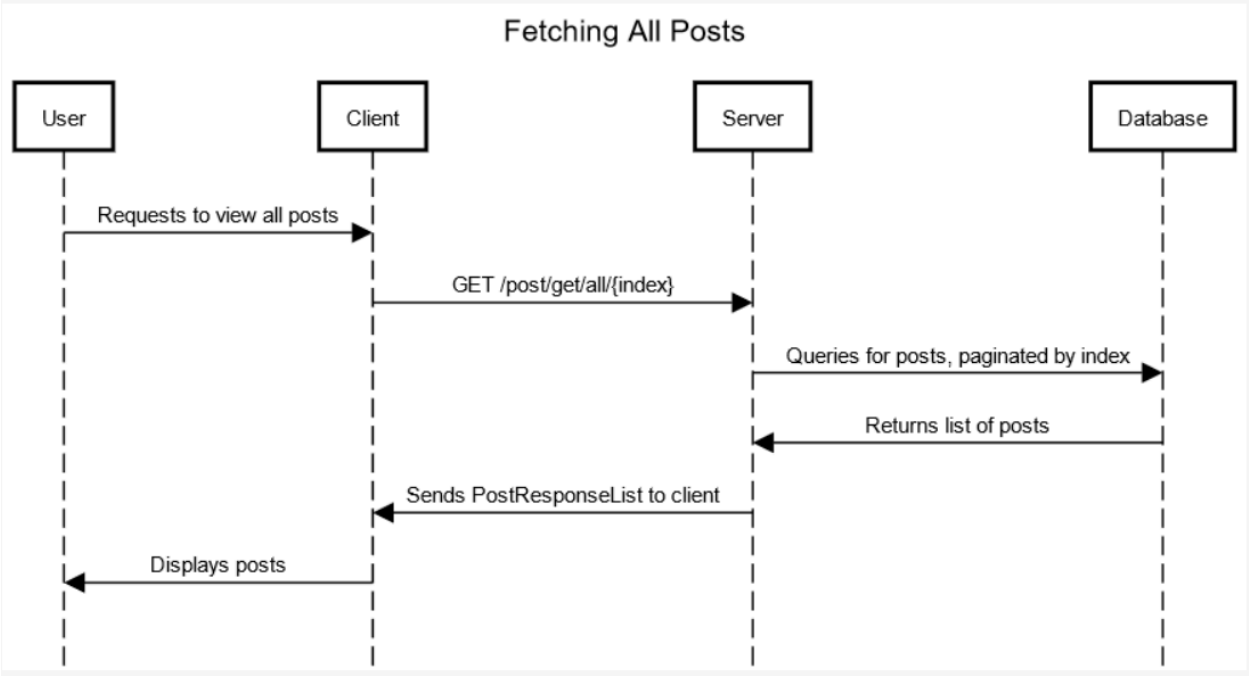


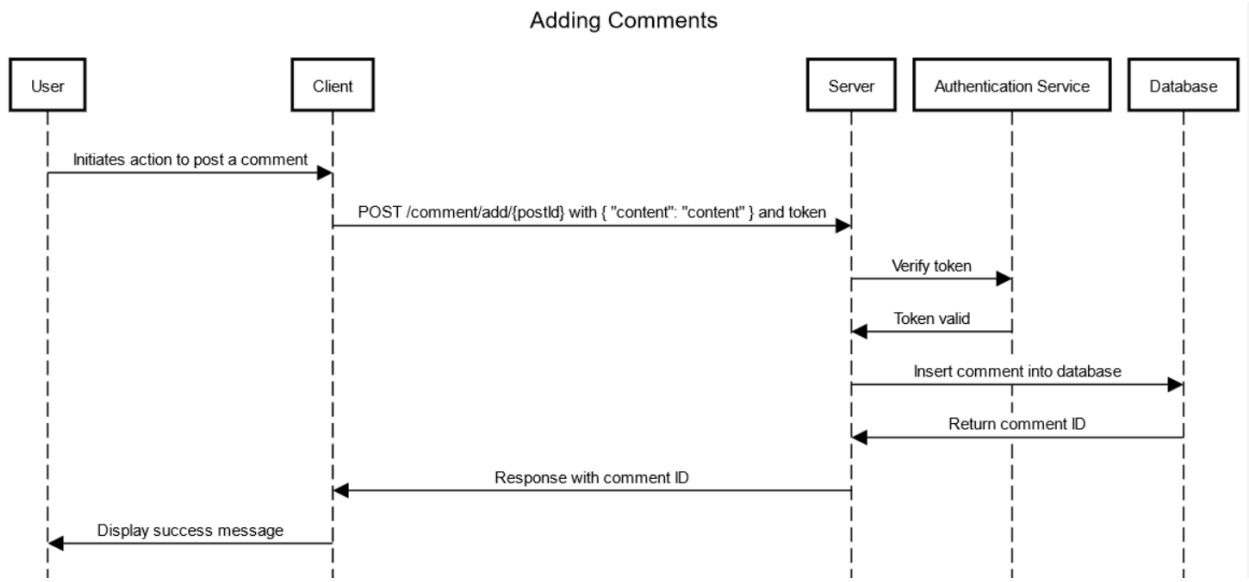


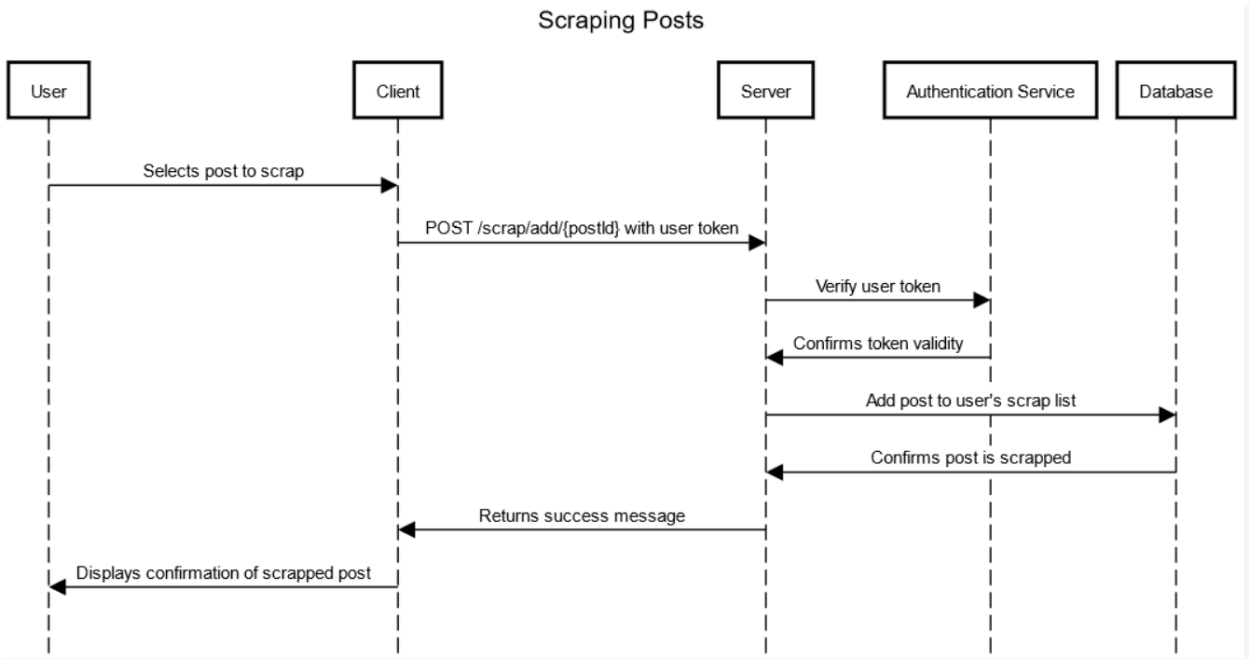

Reading All Post
User
Client
Server
Database
Request to view all posts
GET /post/get/all/{index} (no authentication required)
Retrieve list of posts, paginated
Return list of posts
Respond with PostResponseList
Display posts to the user


Updating Post
User
Client
Server
Authentication Service
Permission Service
Database
Initiate update post action
PATCH /post/update/{id} with { "title": "title", "content": "content", "type": 1 }, file, and bearer token
Verify bearer token
Token valid
Check if user has manager permissions or is in the same team, or is admin
Permission granted
Update post details
Return updated postId or null (on failure)
Respond with postId or null (on failure)
Display result (post update success or failure message)


Deleting Post
User
Client
Server
Authentication Service
Permission Service
Database
Initiate delete post action
DELETE /post/delete/{id} with bearer token
Verify bearer token
Token valid
Check if user has manager permissions or is in the same team
Permission granted
Attempt to delete post
Return result (deleted postId or null)
Respond with result of deletion attempt
Display result (post deletion success or failure message)


Fetching All Posts
User
Client
Server
Database
Requests to view all posts
GET /post/get/all/{index}
Queries for posts, paginated by index
Returns list of posts
Sends PostResponseList to client
Displays posts


Adding Comments
User
Client
Server
Authentication Service
Database
Initiates action to post a comment
POST /comment/add/{postId} with { "content": "content" } and token
Verify token
Token valid
Insert comment into database
Return comment ID
Response with comment ID
Display success message


Scraping Posts
User
Client
Server
Authentication Service
Database
Selects post to scrap
POST /scrap/add/{postId} with user token
Verify user token
Confirms token validity
Add post to user's scrap list
Confirms post is scrapped
Returns success message
Displays confirmation of scrapped post