**7وثيقة التصميم لـ:**

# “Y-A-Social-Media-App”

**اعداد:**

ساجد نجيب تنتوش 2211844915

محمد عادل الكيلاني 2211845885

محمد مصطفى الزعلوك 2211853980

ابراهيم عبدالوهاب الشريف 2211844042

رجب خالد الكبتي 2211849923

محمد الامين عمر القعود 2211846082

النسخة 1.0

21/6/2025

**Contributor’s table:**

|  |  |  |
| --- | --- | --- |
| Section Title | Section Number | Contributor name |
| Introduction | 1 | Sajed Najib Tantush |
| Data Design | 2.1-2.2 | Sajed Najib Tantush & Mohammed Adel Elkilani |
| Architectural and component-level design | 3-3.1 | Rajb Khalid Alkubti |
| Description for Component User Management | 3.2-3.2.4 | Mohammed Adel Elkilani |
| Description for Component Post Management | 3.3-3.3.4 | Ibrahim Abdelwahab Elsharif |
| Description for Component Likes | 3.4-3.4.4 | Sajed Najib Tantush |
| Description for Component Comments | 3.5-3.5.4 | Sajed Najib Tantush |
| Description for Component Follow System | 3.6-3.6.4 | Rajb Khalid Alkubti |
| Description for Component Search & Discovery System | 3.7-.3.7.4 | Mohammed Mustafa Elzaalouk |
| Description for Component Admin | 3.8-3.8.4 | Mohammed Alamin Alqaoud |
| Description for Component Sharing | 3.9-3.9.4 | Sajed Najib Tantush |
| Description of the user interface | 4.0-4.1 | Sajed Najib Tantush |
| SignIn & SignUp | 4.1.1.1 | Mohammed Adel Elkilani |
| Profile Settings | 4.1.1.2 | Mohammed Adel Elkilani |
| Security Settings | 4.1.1.3 | Mohammed Adel Elkilani |
| Follow Suggestions | 4.1.1.4 | Mohammed Mustafa Elzaalouk |
| Explore Page | 4.1.1.5 | Mohammed Mustafa Elzaalouk |
| Connections | 4.1.1.6 | Rajb Khalid Alkubti |
| Posting Interface | 4.1.1.7 | Ibrahim Abdelwahab Elsharif |
| Profile Page | 4.1.1.8 | Mohammed Alamin Alqaoud |
| Admin User & Post Management | 4.1.1.9 | Mohammed Alamin Alqaoud |
| The Home Page | 4.1.1.10 | Sajed Najib Tantush |
| Interface design rules | 4.2 | Sajed Najib Tantush |
| Components available | 4.3 | Sajed Najib Tantush |

# Introduction

* 1. **Purpose**

The purpose of this document is to provide a detailed software design for a media web app named **Y** This application allows users to create and interact with posts, follow other users, and engage through comments, likes, and reposts.

The Software Design Document (SDD) serves as a blueprint for developers, designers outlining how the system will be structured and implemented.

* 1. **Scope**

The social media application will support core features such as user authentication (sign-up/login), creating and managing posts, commenting, liking, reposting, and following other users.

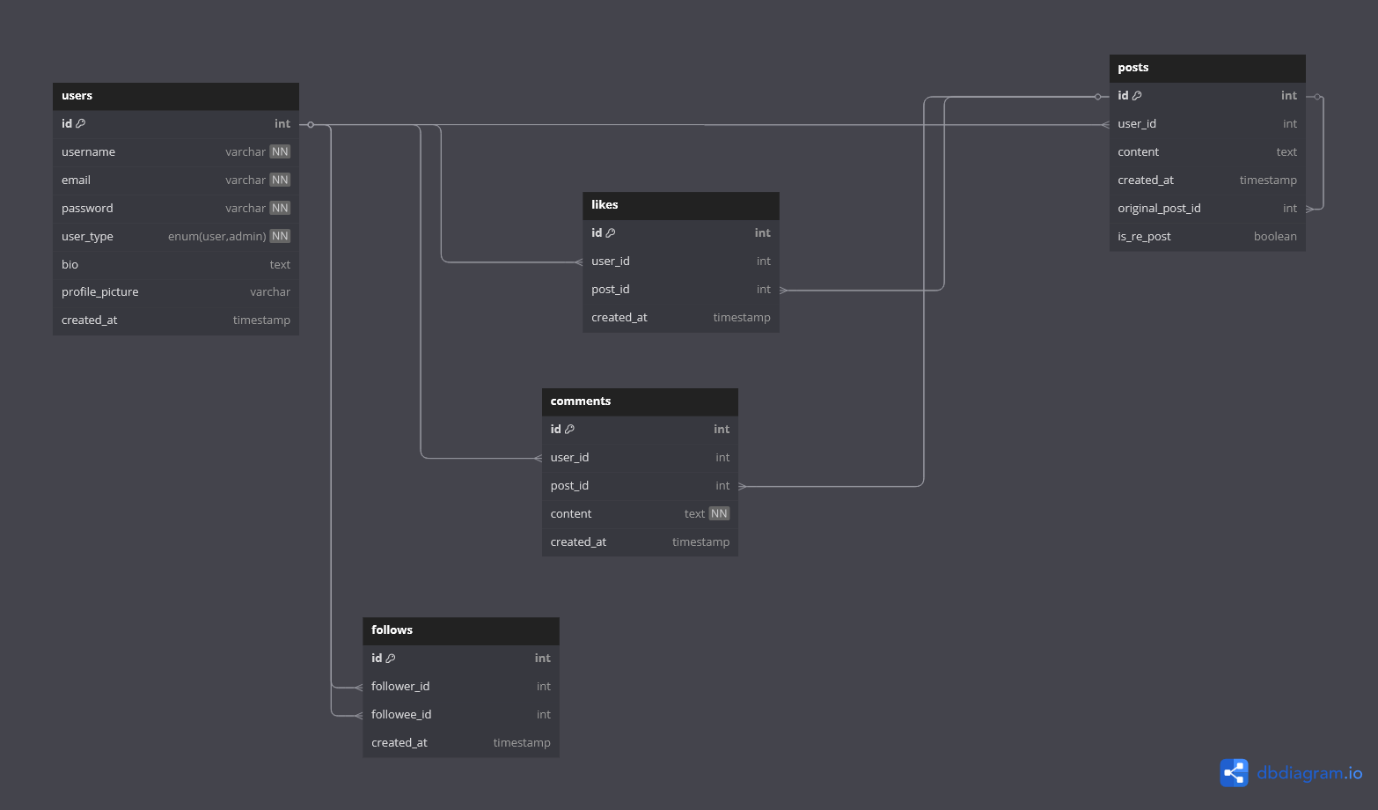
The platform will provide an admin interface for content and user moderation.

This document focuses on the architectural and component-level design of the system, including database design, major classes and their interactions, and interface structure.

# . Data design

A description of all data structures including internal, global, and temporary data structures.

## Database diagram (ERD)



|  |
| --- |
|  |
| Figure 1: ERD Diagram for the system |

## Data Dictionary

Describe the data structures including the database of the system

### Users Table

It is used to keep the common data between all users. Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Description** | **Type** | **Length (bytes)** | **Notes** |
| id | Unique Identifier for Users | INT | 11 | primary key, Auto Increment |
| username | User’s display name | VARCHAR | 30 |  |
| email | User Email Address | VARCHAR | 90 | Unique |
| password | Hashed Password | VARCHAR | 255 |  |
| bio | A user’s bio | TEXT |  |  |
| profile\_picture | A user’s profile Picture | VARCHAR | 255 |  |
| user\_type | User Role | ENUM |  | Value: ‘user’, ‘admin’ |
| created\_at | Time account was created at | TIMESTAMP |  |  |

#### Table Schema

public function up(): void

{

Schema::create('users', function (Blueprint $table) {

$table->id();

$table->string('username')->unique();

$table->string('email')->unique();

$table->string('password');

$table->text('bio')->nullable();

$table->string('profile\_picture')->nullable()->after('bio');

$table->enum('user\_type', ['user', 'admin']);

$table->timestamp(‘created\_at’);

});

}

### Posts table

The Post table is used to keep the websites Posts data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Description** | **Type** | **Length (bytes)** | **Notes** |
| id | Unique Identifier for Posts | INT | 11 | primary key |
| user\_id | Reference to the post author | INT | 11 | Foreign key to users table |
| content | Content of the post | TEXT |  | Nullable(supports reposts) |
| created\_at | Post creation time | TIMESTAMP |  |  |
| original\_post\_id | Refrence to the orginal post | INT | 11 | Foreign key to post.id |
| is\_re\_post | If post is a repost | BOOLEAN |  | Default is false |

#### Table schema

public function up(): void

{

Schema::create('posts', function (Blueprint $table) {

$table->id();

$table->foreignId('user\_id')->constrained('users')->onDelete('cascade');

$table->text('content')->nullable();

$table->timestamp('created\_at');

$table->foreignId('original\_post\_id')->nullable()->constrained('posts')->onDelete('cascade');

$table->boolean('is\_re\_post')->default(false);

});

}

### Likes Table

The Likes table is used to keep the websites Post’s Likes data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Description** | **Type** | **Length (bytes)** | **Notes** |
| id | Unique Identifier for Likes | INT | 11 | primary key |
| user\_id | Reference to the User who liked the post | INT | 11 | Foreign key to users table |
| post\_id | Reference to the post id | INT | 11 | Foreign key to the post’s id |
| created\_at | Timestamp when like was made | TIMESTAMP |  |  |

#### Table Schema

public function up(): void

{

Schema::create('likes', function (Blueprint $table) {

$table->id();

$table->foreignId('user\_id')->constrained('users')->onDelete('cascade');

$table->foreignId('post\_id')->constrained('posts')->onDelete('cascade');

$table->timestamp('created\_at');

$table->unique(['user\_id', 'post\_id']);

});

}

### Comments Table

The Comments table is used to keep the websites Post’s Comments data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Description** | **Type** | **Length (bytes)** | **Notes** |
| id | Unique Identifier for Comments | INT | 11 | primary key |
| user\_id | Reference to the User who commented on the post | INT | 11 | Foreign key to users table |
| post\_id | Reference to the post id | INT | 11 | Foreign key to the post’s id |
| content | Body of the comment | TEXT |  | NOT NULL |
| created\_at | Timestamp when comment was made | TIMESTAMP |  |  |

#### Table Schema

public function up(): void

{

Schema::create('comments', function (Blueprint $table) {

$table->id();

$table->foreignId('user\_id')->constrained('users')->onDelete('cascade');

$table->foreignId('post\_id')->constrained('posts')->onDelete('cascade');

$table->text('content');

$table->timestamp('created\_at')

});

}

### Follows Table

The Follows table is used to keep the websites individual user followers data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Description** | **Type** | **Length (bytes)** | **Notes** |
| id | Unique Identifier for Follower | INT | 11 | primary key |
| follower\_id | User who is following | INT | 11 | Foreign key to users table |
| followee\_id | User who is being followed | INT | 11 | Foreign key to users table |
| created\_at | Timestamp when Follow was made | TIMESTAMP |  |  |

#### Table Schema

public function up(): void

{

Schema::create('follows', function (Blueprint $table) {

$table->id();

$table->foreignId('follower\_id')->constrained('users')->onDelete('cascade');

$table->foreignId('followee\_id')->constrained('users')->onDelete('cascade');

$table->timestamp('created\_at')->useCurrent();

$table->unique(['follower\_id', 'followee\_id']);

});

}

# Architectural and component-level design

A description of the program architecture is presented.

## 3.1 System Structure

A detailed description the system structure chosen for the application is presented.

### 3.1.1 Architecture diagram

A pictorial representation, using a UML component diagram, of the architecture is presented.

|  |
| --- |
|  |
| Figure 2: ERD Diagram for the system |

## 3.2 Description for Component User Management

This component is responsible for handling all user-related operations within the system. It manages user registration, authentication (login/logout), profile updates (including bio and profile picture), password changes, account deletion, and session validation. It plays a core role in enabling secure access to the social media platform and interacts with other components like posts, comments, and follows through user IDs.

### 3.2.1 Processing narrative (PSPEC) for component User Management

The User Management Component is responsible for:

* Registering new users with username, email, password
* Validating user credentials and managing login sessions
* Logging users out and invalidating their session/token
* Allowing users to update their profile details (username, bio, profile picture)
* Changing passwords securely after validation
* Managing session creation and termination (login/logout)
* Fetching user profile data (own and public profiles)
* Interacting with the Follow and Post system for user-based views

This component ensures that only authorized users can access their data and that registration constraints (unique email/username) are enforced.

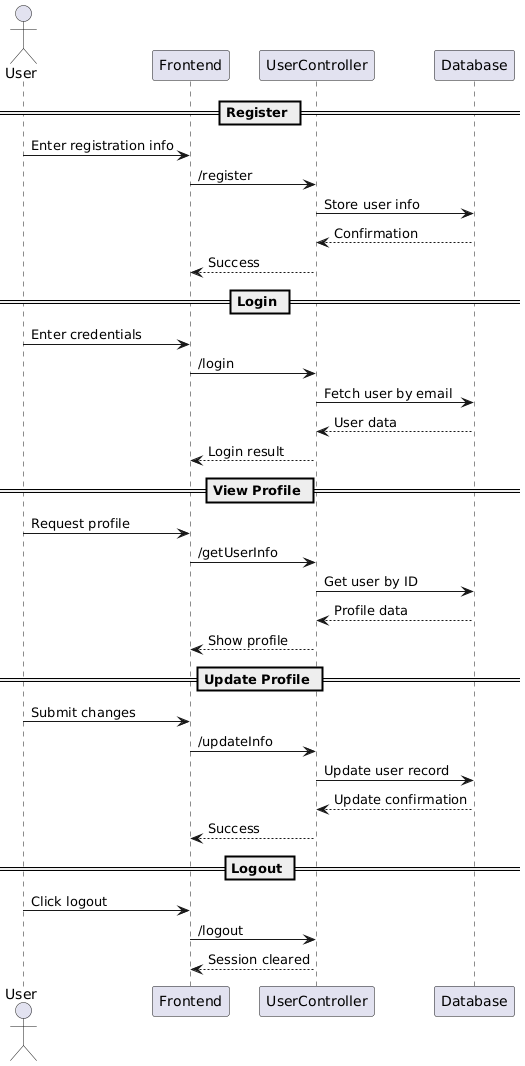
### 3.2.2 Component User Management interface description.

* **Register()**
  + **Input**: username, email, password, bio, profile\_picture
  + **Output**: Success or error message
  + **Purpose**: Create a new user account with the provided details
* **Login()**
  + **Input:** username, password
  + **Output:** session token or error
  + **Purpose:** Authenticates the user and initiates a session
* **Logout()**
  + **Input:** session token
  + **Output:** success or error message
  + **Purpose:** Ends the user’s session
* **getUserInfo()**
  + **Input:** user\_id or session token
  + **Output:** user data (e.g., username, bio, email, profile picture)
  + **Purpose:** Retrieves the user's profile information
* **updateInfo()**
  + **Input:** user\_id, new\_username, new\_bio, new\_profile\_picture
  + **Output:** success or error message
  + **Purpose:** Updates the user's profile details
* **updatePassword()**
  + **Input:** user\_id, old\_password, new\_password
  + **Output:** success or error message
  + **Purpose:** Changes the user's password after validating the old one
* **deleteUser()**
  + **Input:** user\_id
  + **Output:** success or error message
  + **Purpose:** Deletes the user account and all related data
* **isLoggedIn()**
  + **Input:** session token
  + **Output:** boolean
  + **Purpose:** Checks if the session is active and valid

### 3.2.3 Component User Management processing detail

* **register(username, email, password, bio, profile\_picture)**
  + **Validate input fields.**
  + **Check if email or username already exists.**
  + **Hash the password securely.**
  + **Store user in the database with the given data.**
  + **Return success or error.**
* **login(email, password)**
  + **Validate input.**
  + **Retrieve user by email.**
  + **Compare password with stored hash.**
  + **If valid, create a session/token.**
  + **Return token or error.**
* **logout(session\_token)**
  + **Invalidate the session or token.**
  + **Return success message.**
* **getUserInfo(user\_id or session)**
  + **Retrieve user data from database.**
  + **Return profile fields (username, bio, email, profile\_picture).**
* **updateInfo(user\_id, new\_username, new\_bio, new\_profile\_picture)**
  + **Validate updated values.**
  + **Check for username uniqueness if changed.**
  + **Update user record in database.**
  + **Return success or error.**
* **updatePassword(user\_id, old\_password, new\_password)**
  + **Verify old password matches current one.**
  + **Hash and store new password.**
  + **Return confirmation or error.**
* **deleteUser(user\_id)**
  + **Remove user from the database.**
  + **Automatically delete related posts, likes, comments, and follows.**
  + **Return confirmation.**
* **isLoggedIn(token)**
  + **Check session or token validity.**
  + **Return true if session is active, false otherwise.**

### 3.2.4 Interaction Diagrams



## Description for Component Post Management

This component handles all operations related to creating, editing, deleting, listing, and retrieving posts. It also manages reposting functionality and connects with other components like likes, comments, and user profiles.

### Processing narrative (PSPEC) for component Post Management

The **Post Management Component** is responsible for enabling users to interact with the core content of the platform — posts. This includes creating new posts, editing or deleting existing ones, listing all posts, viewing specific posts, and reposting content from other users. Each post is associated with a user and may optionally link to another post when it's a repost. The component ensures that content is stored, retrieved, and managed securely and efficiently.

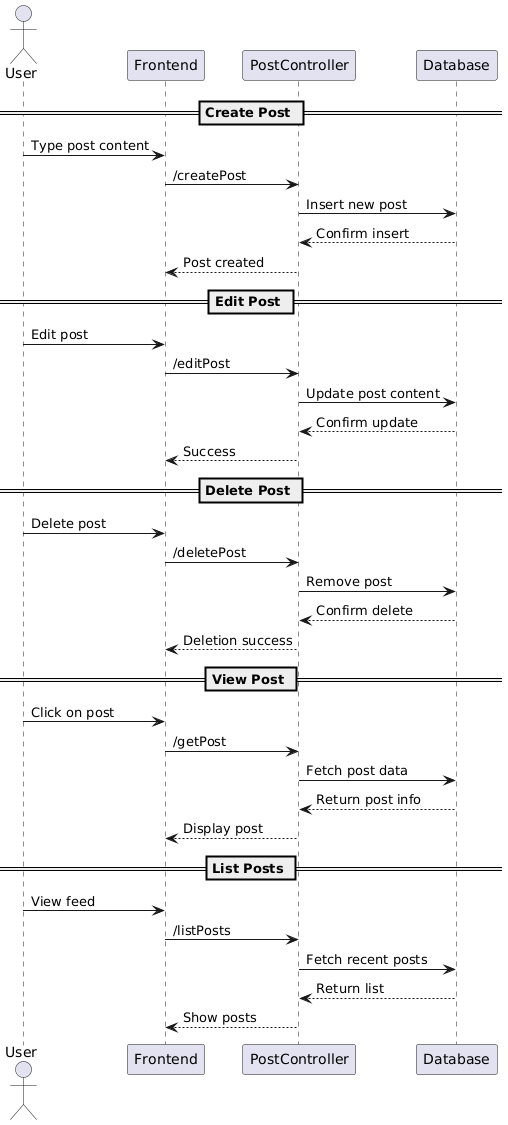
### Component User Management interface description

* **createPost()**
  + **Input:** user\_id, content, original\_post\_id *(optional for reposts)*
  + **Output:** success or error message
  + **Purpose:** Creates a new post by a user.
* **editPost()**
  + **Input:** post\_id, new\_content
  + **Output:** success or error message
  + **Purpose:** Updates the content of an existing post.
* **deletePost()**
  + **Input:** post\_id
  + **Output:** success or error message
  + **Purpose:** Deletes the specified post.
* **getPost()**
  + **Input:** post\_id
  + **Output:** post data
  + **Purpose:** Fetches a single post's details.
* **listPosts()**
  + **Input:** limit, offset
  + **Output:** list of posts
  + **Purpose:** Returns a list of recent posts (optionally paginated).
* **getUserPosts()**
  + **Input:** user\_id
  + **Output:** list of posts
  + **Purpose:** Returns all posts created by a specific user.

### Component Post Management processing detail

* createPost(user\_id, content, original\_post\_id)
  + Validate input.
  + If original\_post\_id is present, mark as repost.
  + Save the post in the database with timestamp and user reference.
  + Return success or error.
* editPost(post\_id, new\_content)
  + Verify the user owns the post.
  + Update the content in the database.
  + Return success or error.
* deletePost(post\_id)
  + Confirm ownership or admin rights.
  + Delete the post.
  + Related likes/comments are deleted via cascade.
  + Return confirmation.
* getPost(post\_id)
  + Retrieve post data from the database.
  + Return content, author, date, and repost info.
* listPosts(limit, offset)
  + Query the database for posts with pagination.
  + Return ordered list (most recent first).
* getUserPosts(user\_id)
  + Query the database for all posts by the user.
  + Return the list of posts.

### Interaction Diagrams



## Description for Component Likes

The **Likes Component** is responsible for allowing users to like or unlike posts within the social media platform. It tracks user engagement with content and helps determine post popularity. Each like is linked to a user and a post, and a user can like a post only once.

### Processing narrative (PSPEC) for component Likes

The **Likes Component** manages the logic behind user interactions with posts in the form of likes. It ensures that a user can only like a post once, and handles the toggling between like and unlike states. It is used to fetch total like counts for posts and to check if a user has already liked a specific post. It interacts primarily with the users and posts components.

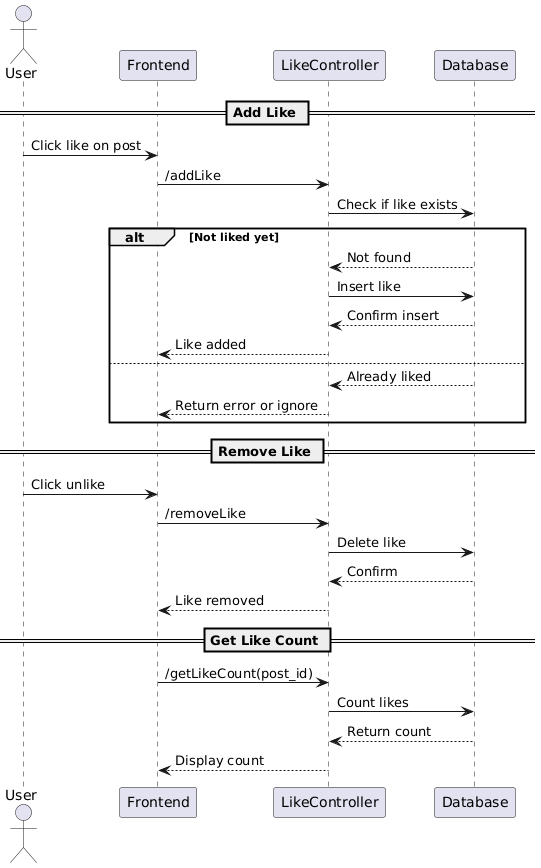
### Component Likes interface description

* **addLike()**
  + **Input:** user\_id, post\_id
  + **Output:** success or error message
  + **Purpose:** Adds a like to a post by a user if not already liked.
* **removeLike()**
  + **Input:** user\_id, post\_id
  + **Output:** success or error message
  + **Purpose:** Removes a like from a post.
* **hasLiked()**
  + **Input:** user\_id, post\_id
  + **Output:** Boolean
  + **Purpose:** Checks whether a user has liked a post.
* **getLikeCount()**
  + **Input:** post\_id
  + **Output:** integer (number of likes)
  + **Purpose:** Returns the total number of likes for a post.

### Component Likes processing detail

* addLike(user\_id, post\_id)
  + Check if user has already liked the post.
  + If not, insert like into the database.
  + Return success or error.
* removeLike(user\_id, post\_id)
  + Verify the like exists.
  + Delete the like from the database.
  + Return success or error.
* hasLiked(user\_id, post\_id)
  + Query database for existing like.
  + Return true if found, false otherwise.
* getLikeCount(post\_id)
  + Query database for number of likes on a post.
  + Return integer count.

### Interaction Diagrams



## Description for Component Comments

The **Comments Component** manages user-generated replies on posts. It supports adding, retrieving, and displaying comments under posts and ensures each comment is associated with a user and a post.

### Processing narrative (PSPEC) for component Comments

The **Comments Component** allows users to engage in discussions through comments on posts. Each comment is tied to both a user and a post.

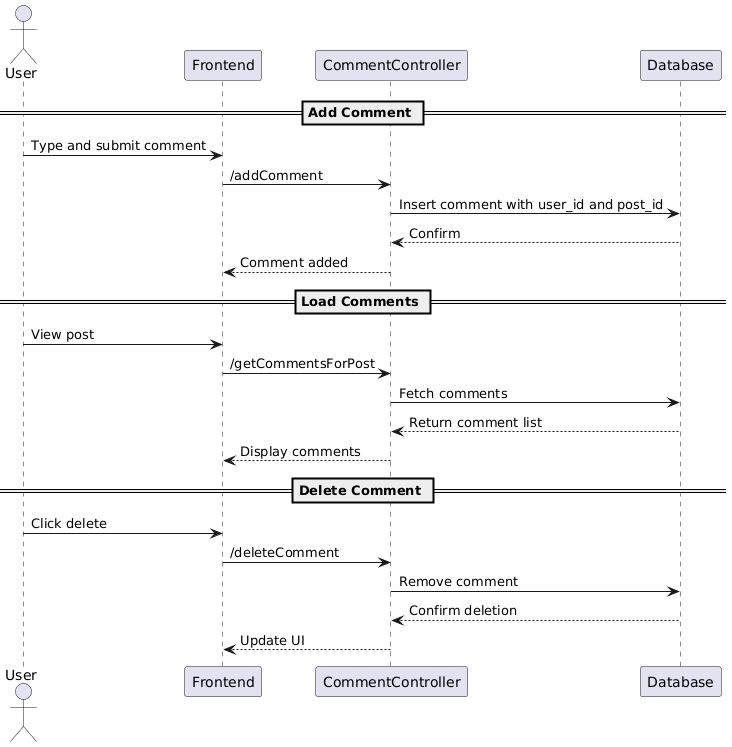
### Component Comments interface description

* **addComment()**
  + **Input:** user\_id, post\_id, content, parent\_comment\_id *(optional)*
  + **Output:** success or error message
  + **Purpose:** Adds a comment or reply to a post.
* **getCommentsForPost()**
  + **Input:** post\_id
  + **Output:** list of comments
  + **Purpose:** Returns all comments (and replies) for a post.
* **deleteComment()**
  + **Input:** comment\_id
  + **Output:** success or error message
  + **Purpose:** Deletes a comment

### Component Comment processing detail

* addComment(user\_id, post\_id, content, parent\_comment\_id)
  + Validate input.
  + If parent\_comment\_id is set, confirm parent exists.
  + Insert comment into the database with optional parent reference.
  + Return success or error.
* getCommentsForPost(post\_id)
  + Query comments with post\_id.
  + Sort comments by timestamp.
  + Structure as flat or nested list (if replies exist).
  + Return list.
* deleteComment(comment\_id)
  + Check ownership or permissions.
  + Delete comment and optionally its replies.
  + Return confirmation.

### Interaction Diagrams



## Description for Component Follow System

The **Follow System Component** manages the relationships between users by allowing one user to follow or unfollow another. It keeps track of followers and followees and provides counts and lists used across the platform, such as in user profiles and activity feeds.

### Processing Narrative (PSPEC) for Follow System

The **Follow System Component** enables user-to-user connections by supporting operations such as follow, unfollow, and retrieval of follower/followee data. It ensures that each user can follow another only once, and that mutual or one-way relationships are correctly recorded. This component plays a key role in content filtering, feed generation, and social features of the platform.

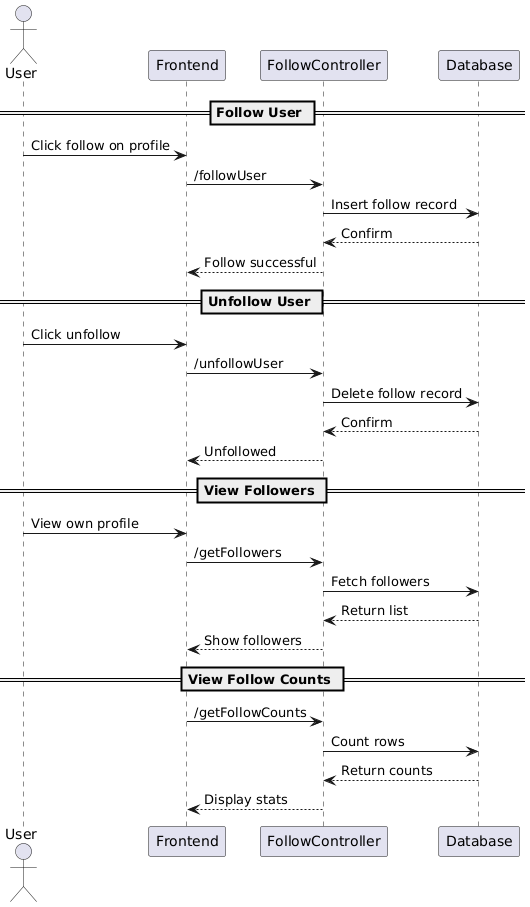
### Component Follow System interface description

* **followUser()**
  + **Input:** follower\_id, followee\_id
  + **Output:** success or error message
  + **Purpose:** Follows a user, unless already followed.
* **unfollowUser()**
  + **Input:** follower\_id, followee\_id
  + **Output:** success or error message
  + **Purpose:** Unfollows a user if already followed.
* **getFollowers()**
  + **Input:** user\_id
  + **Output:** list of users
  + **Purpose:** Returns a list of users following the specified user.
* **getFollowing()**
  + **Input:** user\_id
  + **Output:** list of users
  + **Purpose:** Returns a list of users that the specified user is following.
* **getFollowCounts()**
  + **Input:** user\_id
  + **Output:** number of followers and followings
  + **Purpose:** Returns numeric counts for quick display.

### Component Follow System processing detail

* followUser(follower\_id, followee\_id)
  + Check if the relationship already exists.
  + If not, insert new record into follows table.
  + Return success or error.
* unfollowUser(follower\_id, followee\_id)
  + Check if the relationship exists.
  + If so, delete the record.
  + Return confirmation.
* getFollowers(user\_id)
  + Query the follows table for users following user\_id.
  + Return list of user profiles.
* getFollowing(user\_id)
  + Query the follows table for users user\_id is following.
  + Return list of user profiles.
* getFollowCounts(user\_id)
  + Count the number of followers and followings.
  + Return the two integers.

### Interaction Diagrams



## Description for Component Search & Discovery System

This component handles simple searching capabilities within the application. Users can search for other users by typing their username (with or without the @ symbol) or search post content by keywords. There are no filters or advanced search options.

### Processing Narrative (PSPEC) for Search & Discovery System

The **Search & Discovery Component** is a lightweight feature that enables users to find other users by username and posts by basic keyword matching. It powers the search bar in the application, returning a combined result set for both users and posts depending on the query. It is designed for performance and simplicity, with no support for sorting or filters.

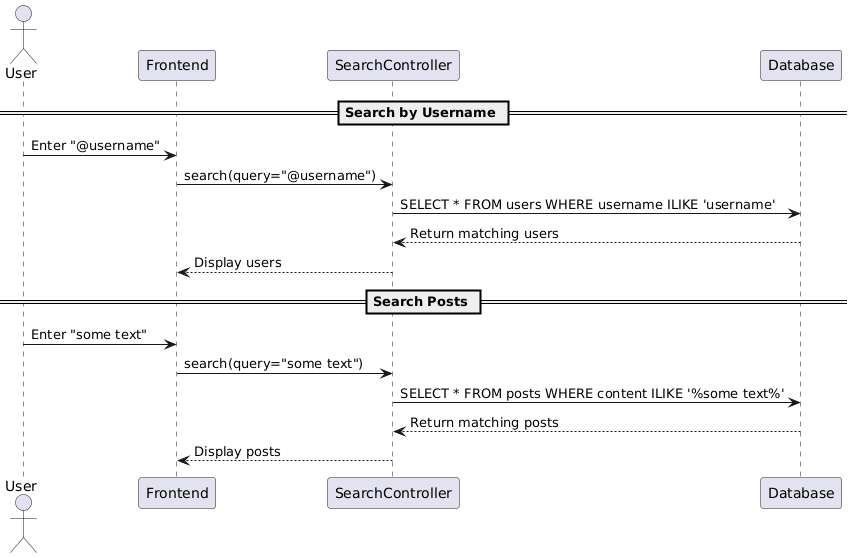
### Component Search & Discovery System interface description

* **search(query)**
  + **Input:** query (string)
  + **Output:** list of matching users or posts
  + **Purpose:** Returns results where the query matches either a username (@username) or content in a post.

### Component Seach & Discovery System processing detail

* search(query)
  + If query starts with @, search the users table by username.
  + Else, search the posts table for content containing the keyword.
  + Return matching users or posts as appropriate.

### Interaction Diagrams



## Description for Component Admin

The **Admin Component** provides administrative control over the web application. It enables the admin to manage users, monitor content the admin has access to privileged operations that regular users cannot perform.

### Processing Narrative (PSPEC) for Admin Component

The **Admin Component** is responsible for managing users and overseeing the health of the application. Admins can view all users, delete user accounts, and manage Posts. This component ensures that the application remains moderated and allows the admin to respond to misuse or content violations.

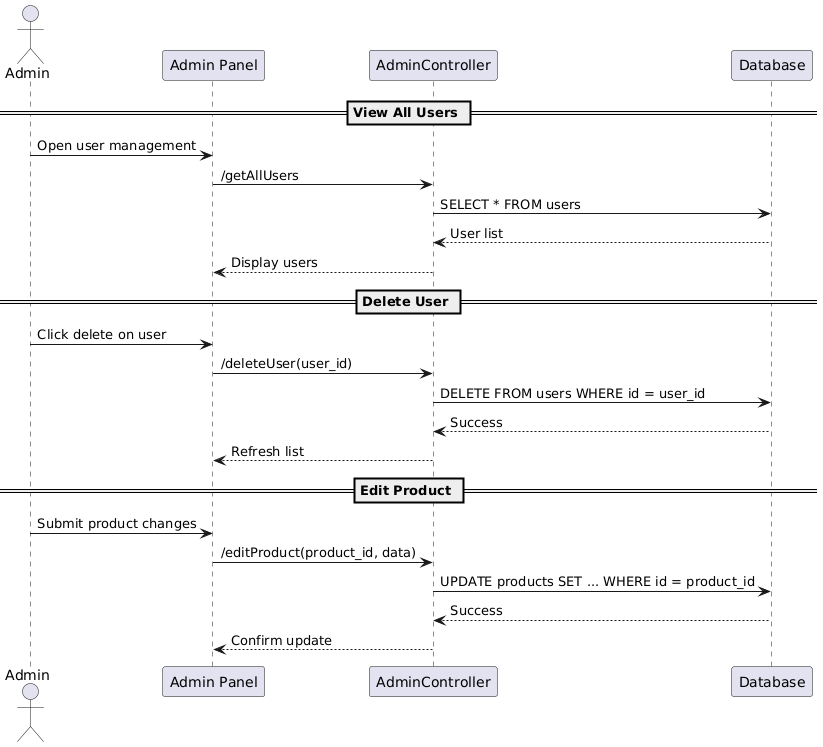
### Component Admin interface description

* **getAllUsers()**
  + **Input:** none
  + **Output:** List of users
  + **Purpose:** Retrieves a full list of registered users for monitoring.
* **deleteUser(user\_id)**
  + **Input:** user\_id
  + **Output:** Success or failure message
  + **Purpose:** Deletes a user and all associated data.
* **getAllPosts()**
  + **Input:** none
  + **Output:** List of posts
  + **Purpose:** Displays all posts in the system for review.
* **deletePost(post\_id)**
  + **Input:** post\_id
  + **Output:** Success or failure message
  + **Purpose:** Deletes a post reported or flagged by the system or users.

### Component Admin processing detail

* getAllUsers()
  + Query the users table.
  + Return the full list with pagination.
* deleteUser(user\_id)
  + Verify user exists and is not an admin.
  + Delete the user and related data.
* editUserInfo(user\_id, updatedData)
  + Validate input.
  + Update the user record with the new values.
* getAllProducts()
  + Query the product table.
  + Return all entries.
* deleteProduct(product\_id)
  + Check if the product exists.
  + Delete it from the database.
* editProduct(product\_id, updatedData)
  + Validate product info.
  + Update the database record.

### Interaction Diagram



## Description for Component Sharing

This component allows users to share (repost/retweet) existing posts created by others. It enables content circulation and visibility throughout the platform while linking the shared post back to the original author. It is integrated with the post system and user interface to display reposted content properly.

### Processing Narrative (PSPEC) for Sharing Component

The Sharing Component is designed to handle the functionality that enables users to share or repost existing content created by other users on the platform this component ensures that a shared post remains linked to its original source by storing a reference to the original post.

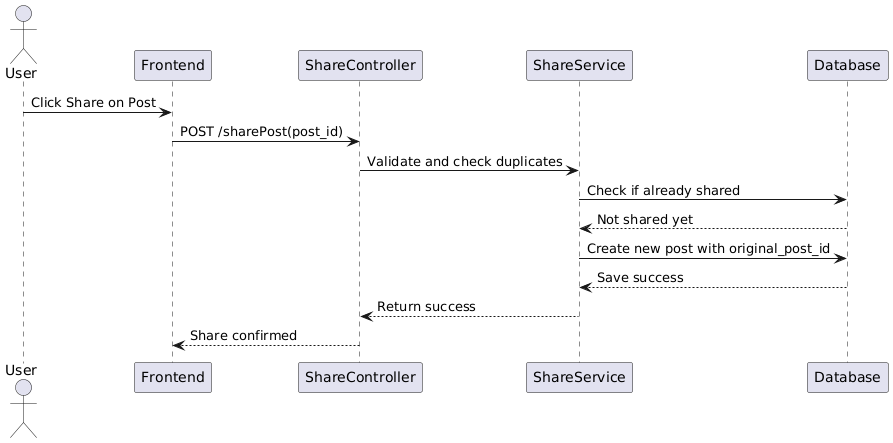
### Component Sharing interface description

* sharePost()
  + **Input:** user ID, post ID to share, optional message
  + **Output:** success or error message
  + **Purpose:** To allow a user to share an existing post, optionally with their own comment. Creates a new post record referencing the original.
* **getSharedPosts()**
  + **Input:** user ID or original post ID
  + **Output:** list of shared posts
  + **Purpose:** To retrieve all reposts made by a user or all reposts related to a specific post.

### Component Sharing processing detail

* sharePost(user\_id, post\_id, optional\_text)
  + Check if the post exists and user hasn’t already shared it.
  + Create a new post entry with original\_post\_id referencing the shared post.
  + Set is\_re\_post = true.
  + Return success or error.
* getSharedPosts(user\_id)
  + Query all posts where user\_id matches and is\_re\_post = true.
  + Return list of shared post data.

### Interaction Diagram



# 4.0 User interface design

## 4.1 Description of the user interface

The user interface of **Y** a social media application is designed to be clean, intuitive the main home screen is structured into three primary sections the left sidebar contains navigation tabs for key features such as Home, Profile, Explore and Settings. The central area is dedicated to post viewing and interaction with two tabs labeled **"For You"** and **"Following"** The "For You" tab displays posts from all users, curated for discovery, while the "Following" tab shows posts exclusively from users the person follows this layout allows for seamless switching between global content and personalized feeds.

on the right-side users can find a search bar at the top to look up other users by their handle (@username) or find posts by keywords. Below that, the section displays suggested users to follow based on activity and relevance.

users can update their password, bio and profile picture through the profile settings interface There is also a dedicated interface for viewing a user’s followers and followees.

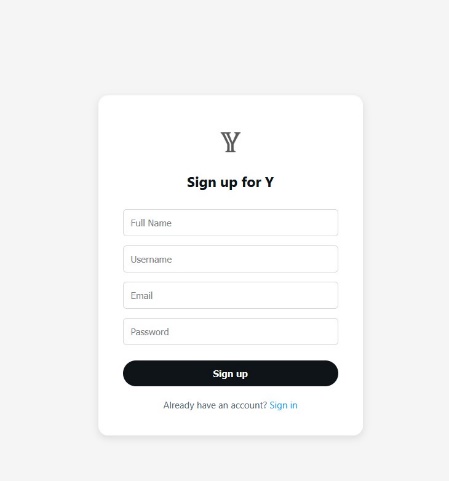
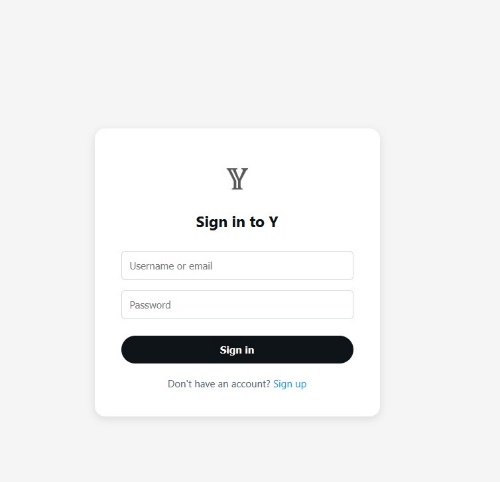
For users with the admin role, the interface includes two tabs

* One tab provides a complete list of all posts, allowing admins to review and delete any content that violates platform guidelines.
* Another tab displays a list of all users, giving admins the ability to manage user accounts, including deletion if necessary.

### 4.1.1 Screen images

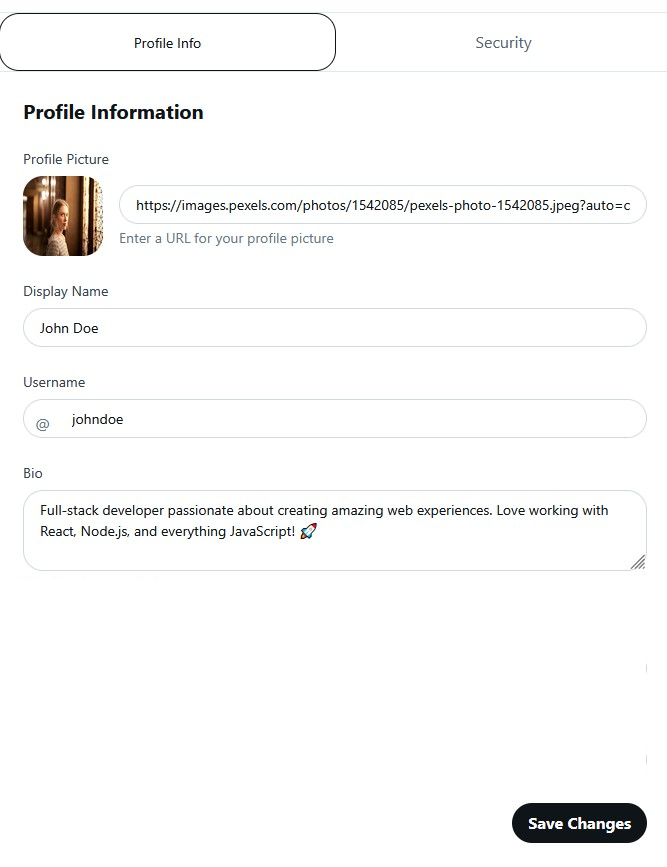
#### SignIn & SignUp

The app opens with two separate screens: the Sign In interface where existing users enter their username and password to access their account, and the Sign-Up interface where new users register by providing a username, full name ,email, and password. Both interfaces are designed to be simple.

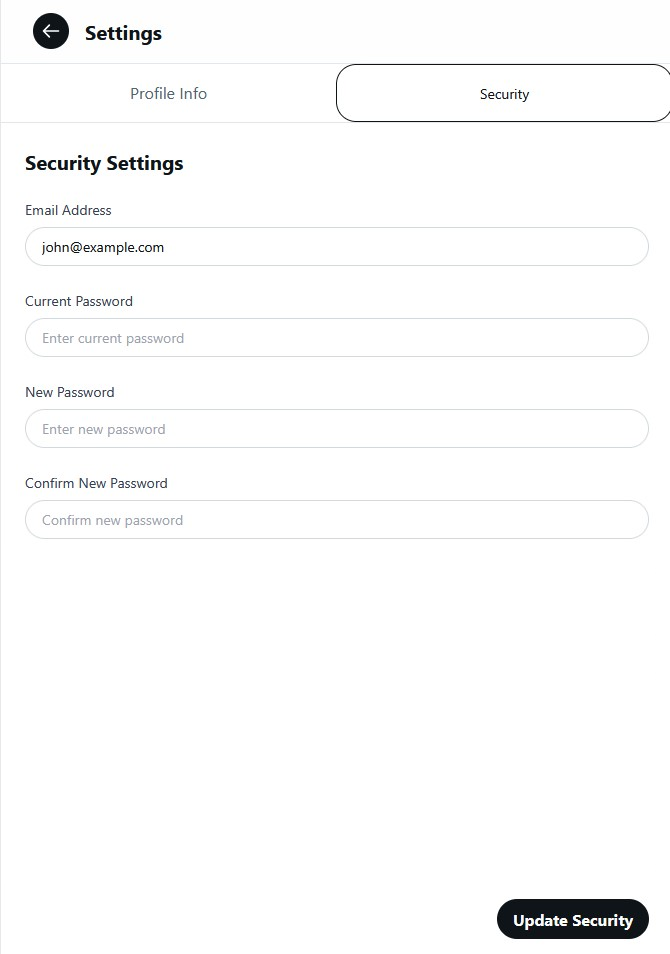
#### Profile Settings

The Profile Settings interface allows users to update their personal information, including their username, display name, profile picture, and bio. The design is straightforward, providing editable fields and an easy way to save changes.



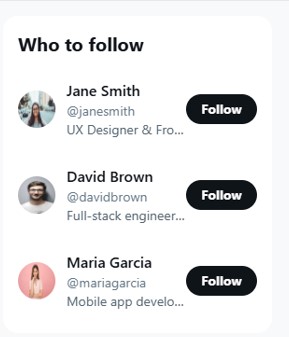
#### Security Settings

The Security Settings interface enables users to update their password and change their email address. It includes secure input fields with validation to ensure data integrity.



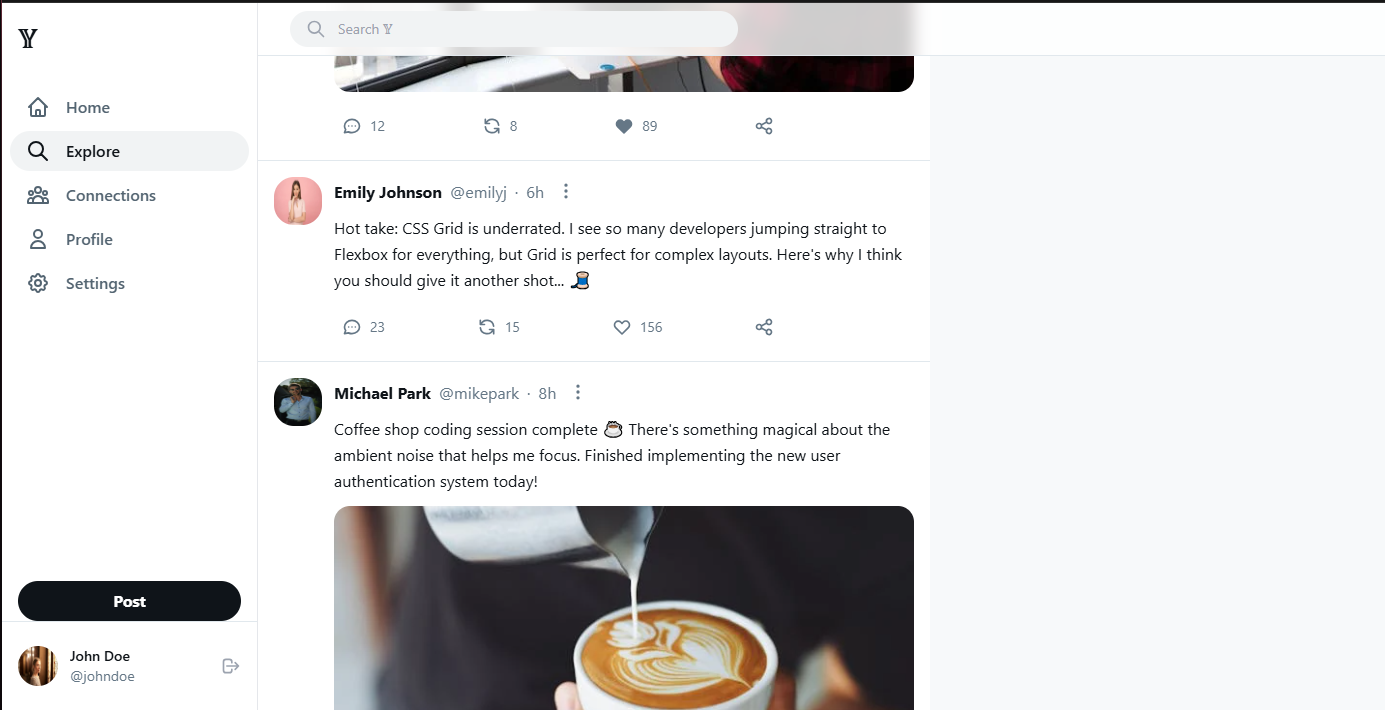
#### Follow Suggestions

The Follow Suggestions component appears on the right side of the main site interface. It displays three randomly selected users recommended for the current user to follow. This compact section helps users discover new connections.



#### Explore Page

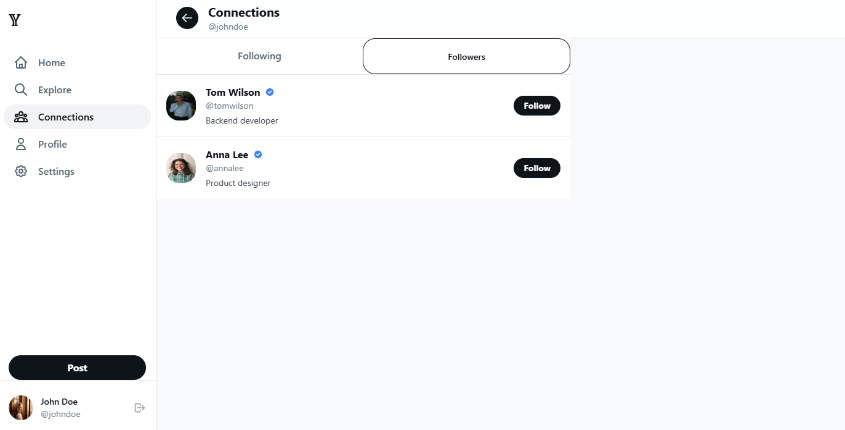
The Explore Page displays search results when a user uses the search bar. It shows relevant posts and user profiles based on the entered keywords or user handles.

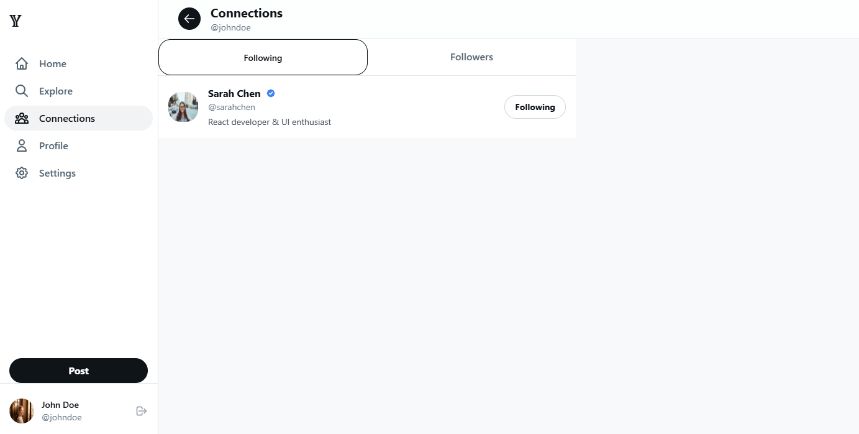


#### Connections

The Connections Tab contains two sub-tabs: Followers and Following.

The Followers tab lists all users who follow the current user, while the Following tab displays all users that the current user follows





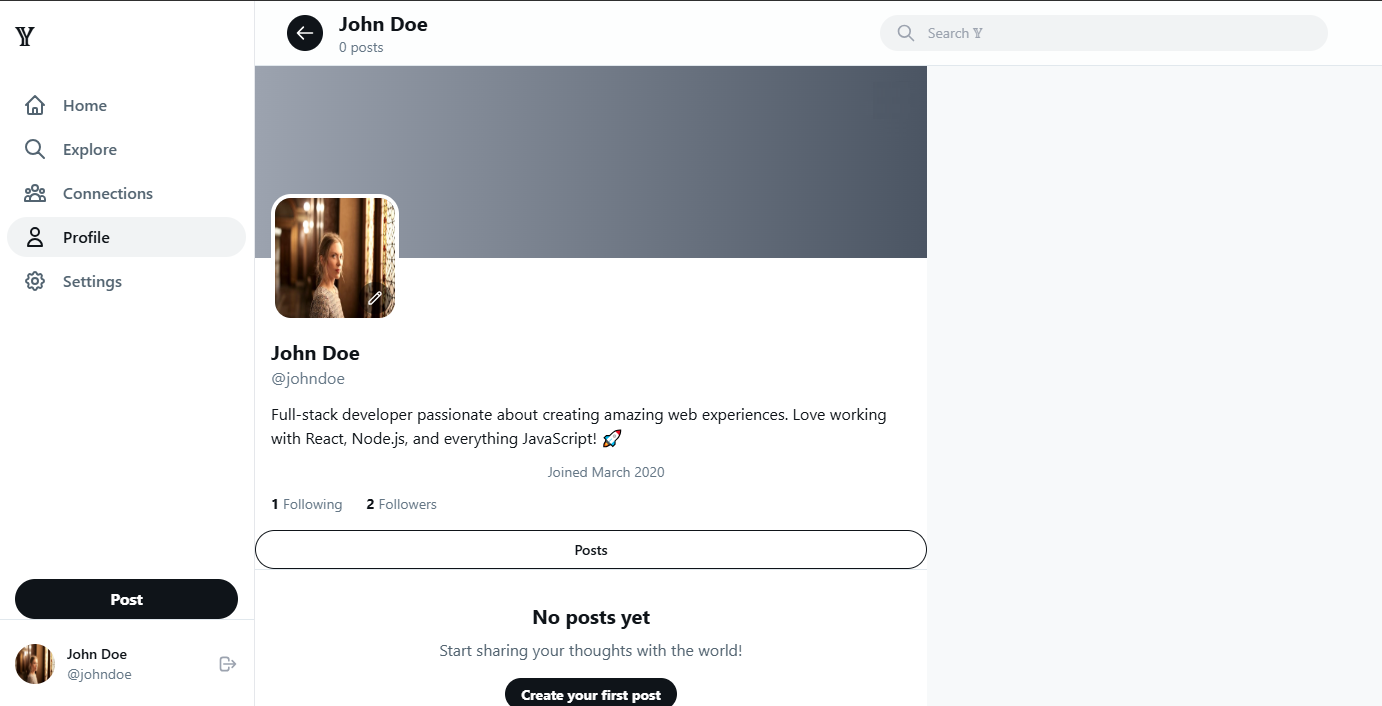
#### Posting Interface

The Post Component is located on the user’s home page and allows users to create new posts. It includes a text input area and an option to attach an image.



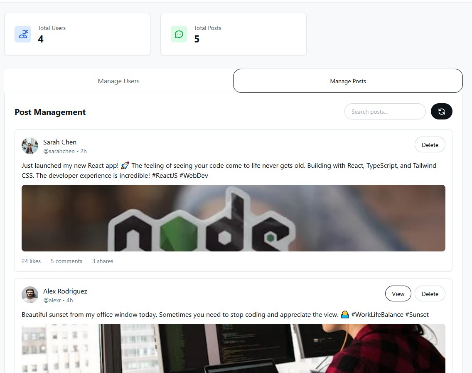
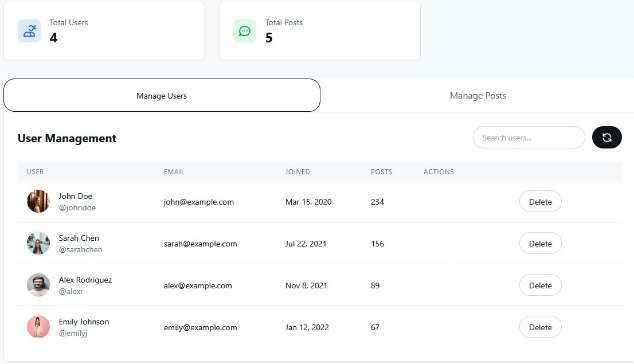
#### Profile Page

The Profile Page displays a user's information, including their profile picture, username, display name, bio, and post history. It also shows follower and following counts and follow/unfollow the user if viewing someone else's profile.



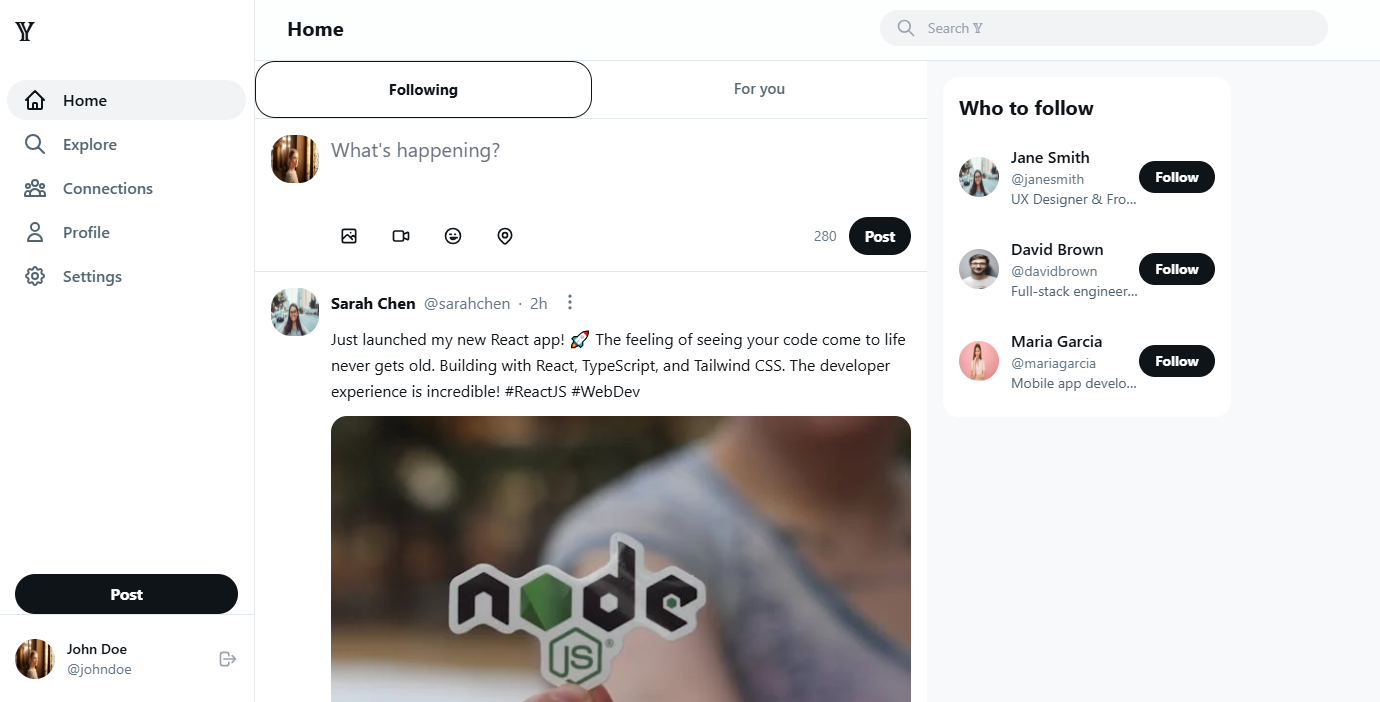
#### Admin User & Post Management

The Admin Page is accessible only to admin users and contains two tabs: Users and Posts. The Users tab displays a list of all registered users with the option to remove any account. The Posts tab lists all posts on the platform, allowing the admin to delete inappropriate or harmful content.

#### The Home Page

The Home Page is the main interface of the app and is divided into three sections. The left sidebar contains navigation tabs such as Home, Profile, and Settings. The center panel displays the post feed with two tabs For You (showing posts from all users) and Following (showing posts from followed users). The right **panel** includes the search bar and follows suggestions.



## 4.2 Interface design rules

The user interface of **Y** is designed to be simple, clear, and easy to use. The layout is consistent across all pages, helping users learn and navigate the app quickly. Buttons, colors, icons, and fonts are reused to keep everything familiar.

## 4.3 Components available

The user interface of **Y** is built using standard, reusable GUI components the following components are available for use throughout the application:

* **Navigation Bars** – Used on the left side for main app navigation
* **Post Cards** – Display individual posts with content, user info, likes, reposts, and comment buttons.
* **Buttons** – For actions such as like, repost, follow, submit, edit, delete.
* **Forms** – Used for login, registration, post creation, profile editing, and settings updates.
* **Input Fields** – Text fields, email/password inputs, text areas for posts, and file uploads for images or profile pictures.
* **Search Bars** – Present on the home and explore interfaces for finding users or posts.
* **Avatars & User Cards** – Show user profile picture, name, and interaction buttons (follow/unfollow).
* **Tabs** – Used in feeds ("For You" and "Following"), admin views (Users and Posts), and profile pages (Followers and Following).