ProfileBook
Capstone Project Report

Submitted by: Venkatesh Modugumudi

Batch: WIPRO NGA-.Net Full Stack Angular -FY26 -C3

Instructor: Mr. Ramesh Nediyadath

Date: 7th September 2025

Table of Contents

Abstract

The ProfileBook project is a web-based application developed using ASP.NET Core for the backend and Angular for the frontend. The main objective of this system is to provide users with a secure platform to create, manage, and update their personal and professional profiles. It simplifies profile management by offering features such as user registration, authentication, role-based access, and profile editing.

The backend is responsible for handling business logic, authentication, and database communication, while the frontend ensures a responsive and user-friendly interface. Data is stored securely using a structured database, and security is enhanced through proper validation and access control.

This project demonstrates the integration of modern web technologies to build a scalable and efficient system. It not only helps users maintain their personal details in a structured way but also showcases best practices in full-stack development, including component-based design, RESTful APIs, and database-driven applications.

Introduction & Problem Statement

ProfileBook is a web-based application built with **ASP.NET Core** and **Angular** to help users create, manage, and update their personal and professional profiles. Unlike static documents, it provides a secure, user-friendly, and responsive platform with features like authentication, role-based access, and structured data storage. The project demonstrates modern full-stack practices, including RESTful APIs, database design, and component-based development.

Problem Statement

Users often struggle to maintain and update profiles using static documents, which are difficult to manage, insecure, and not easily accessible. There is a need for a simple, secure, and scalable platform to store and update personal information anytime. ProfileBook addresses this by offering a **web-based solution** with authentication, structured data management, and an intuitive interface for seamless profile handling.

Objectives

- **Develop a secure web application** using ASP.NET Core (backend) and Angular (frontend).
- **Provide user authentication and role-based access control** to ensure data privacy and security.
- **Design and implement a structured database** for storing and managing user profile information efficiently.
- **Create a responsive and user-friendly interface** that allows easy creation, editing, and viewing of profiles.
- **Integrate RESTful APIs** to enable smooth communication between the frontend and backend.
- **Ensure scalability and maintainability** by following best practices in full-stack development.

System Requirements & User Stories

- **User Registration & Login** – Users must be able to create an account and log in securely.
- **Role-Based Access Control** – Admins can manage users; normal users can only manage their own profiles.
- **Profile Management** – Users can create, edit, update, and view their personal/professional profiles.
- **Data Validation** – Input fields should be validated (e.g., email format, required fields).
- **Secure Authentication** – Passwords must be stored securely, and sessions should be protected.
- **Responsive UI** – The application must work on desktops, tablets, and mobile devices.
- **Database Integration** – All profile data must be stored in a relational database.
- **Search & View Profiles** – (Optional) Users should be able to search or view profiles with permissions.

User Stories

- **As a new user,** I want to register an account so that I can access the application.
- **As a registered user,** I want to log in securely so that I can access my profile.
- **As a user,** I want to create and update my personal profile so that my information is always current.
- **As a user,** I want to view my profile details so that I can confirm my stored information.
- **As a user,** I want the system to validate my inputs (like email, phone) so that I don't make mistakes.
- **As an admin,** I want to manage users (create, update, delete, or assign roles) so that I can control access.
- **As a system,** I need to securely store passwords and personal data so that user privacy is maintained.
- **As a user,** I want to access the application on any device so that I can manage my profile anytime, anywhere.

System Architecture

The **Profile Book** project uses a **three-tier architecture**:

- **Frontend (Angular)** – Provides a responsive UI for registration, login, and profile management. It communicates with the backend through **REST APIs**.
- **Backend (ASP.NET Core Web API)** – Handles business logic, authentication, role-based access, and CRUD operations. Uses **JWT authentication** for security.
- **Database (SQL Server / RDBMS)** – Stores user details, roles, and profile data, managed through **Entity Framework Core**.

Flow:

- User interacts with Angular (browser).
- Angular sends API requests to ASP.NET Core backend.
- Backend processes requests and communicates with the database.
- Data is returned and displayed on the frontend.

[USER BROWSER] → [ANGULAR FRONTEND] → [ASP.NET CORE BACKEND] → [DATABASE (SQL SERVER)]

Frontend Design (Angular)

The Frontend of **Profile Book** is developed using **Angular** with a component-based structure and modular services. It ensures a responsive UI and smooth interaction with the backend via REST APIs.

Admin Module

- **Groups** – Manage user groups and roles.
- **Post Approval** – Admins can review and approve/reject posts.
- **Reports** – View system reports and flagged content.
- **Users** – Manage registered users (view, update, delete).

User Module

- **Login / Register** – Secure authentication and new user registration.
- **Posts** – Create, view, and update posts.
- **Profile** – Manage personal information and settings.

Core Files

- **app.config.ts** – Stores global configuration settings.
- **app.route.ts** – Defines routing between components.
- **app.ts** – Root application file.
- **app.html** – Main template layout.
- **app.css** – Stylesheet for UI design.

Services

- **API Service** – Centralized service for backend communication.
- **Auth Service** – Handles login, token storage, and role checks.
- **JWT Interceptor** – Attaches JWT token to HTTP requests for secure API access.

Backend Design (ASP.NET Core)

The Backend of **Profile Book** is built using **ASP.NET Core Web API**, following a layered architecture with **Controllers, Models, and DTOs**. It handles authentication, business logic, and database communication while exposing RESTful endpoints to the Angular frontend.

Controllers

- **AuthController** – Provides **Register** and **Login** methods for secure user authentication and JWT token generation.
- **GroupController** – Handles creation and management of user groups.
- **MessageController** – Enables sending and retrieving messages.
- **PostsController** – Manages post creation, approval, and updates.
- **ProfilesController** – Handles profile creation, updates, and image uploads.
- **ReportController –** Allows users to report suspicious or fake accounts, storing reports for admin review.
- **UsersController** – Manages user details, roles, and updates.

Models

- **Comment, Group, Post, Profile, Report, User** – Represent database entities and define relationships.

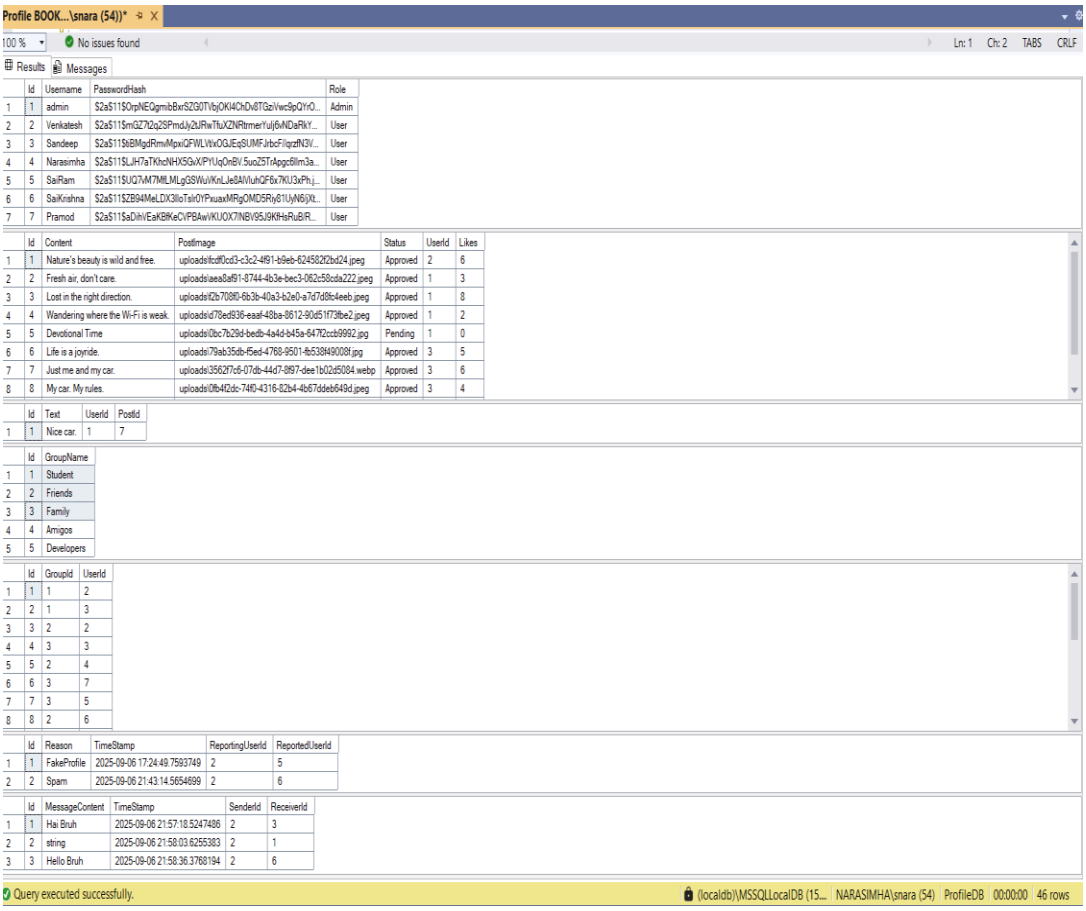
DTOs (Data Transfer Objects)

- **CommentDto** – Transfers comments associated with posts, including details such as comment text, user, and timestamp.
- **SendMessageDto** – Defines the payload for sending messages between users.
- **PostCreateDto** – Defines structure for creating posts.
- **ProfileImageUploadDto** – Used for uploading profile images.
- **UpdateUserDto** – Handles user update requests securely.

Database Design & ERD

The database of **Profile Book** is designed using a **relational model** to ensure structured storage and easy retrieval of data. It maintains relationships between users, posts, comments, reports, and groups.

Main Tables:



Profile BOOK...\snara (54))*

100 % — No issues found — Ln: 1 Ch: 2 TABS CRLF

Results | Messages

	Id	Username	PasswordHash	Role
1	1	admin	$2a$11$OrpNEQgmibBxrSZG0TrbjOKI4ChDx6TGzVwc9pQYrO...	Admin
2	2	Venkatesh	$2a$11$mGZ7t2q2SPmdJy2tJRwTfuXZNRtmerYulj6vNDaRkY...	User
3	3	Sandeep	$2a$11$tBMgdRmvMpxiQFWLWxOGJEqSUMFJrbcFIlqrzfN3V...	User
4	4	Narasimha	$2a$11$LJH7aTKhcNHX5GxXPYUqOnBV.5uoZ5TrApgc6llm3a...	User
5	5	SaiRam	$2a$11$UQ7vM7MtLMLgGSWuVKnLJe8AIVluhQF6x7KU3xPhj...	User
6	6	SaiKrishna	$2a$11$ZB94MeLDX3lIoTslr0YPxuaxMRgOMD5Ry61UyN6jXt...	User
7	7	Pramod	$2a$11$aDihVEaKBfKeCiPBAwiKUOX7INBV95J9KfHsRuBiR...	User

	Id	Content	PostImage	Status	UserId	Likes
1	1	Nature's beauty is wild and free.	uploads\fcdf0cd3-c3c2-4f91-b9eb-624582f2bd24.jpeg	Approved	2	6
2	2	Fresh air, don't care.	uploads\aea8af91-8744-4b3e-bec3-062c58cda222.jpeg	Approved	1	3
3	3	Lost in the right direction.	uploads\f2b708f0-6b3b-40a3-b2e0-a7d7d8fc4eeb.jpeg	Approved	1	8
4	4	Wandering where the Wi-Fi is weak.	uploads\d78ed936-eaaf-48ba-8612-90d51f73fbe2.jpeg	Approved	1	2
5	5	Devotional Time	uploads\0bc7b29d-bedb-4a4d-b45a-647f2ccb9992.jpg	Pending	1	0
6	6	Life is a joyride.	uploads\79ab35db-f5ed-4768-9501-fb538f49008f.jpg	Approved	3	5
7	7	Just me and my car.	uploads\3562f7c6-07db-44d7-8f97-dee1b02d5084.webp	Approved	3	6
8	8	My car. My rules.	uploads\0fb4f2dc-74f0-4316-82b4-4b67ddeb649d.jpeg	Approved	3	4

	Id	Text	UserId	PostId
1	1	Nice car.	1	7

	Id	GroupName
1	1	Student
2	2	Friends
3	3	Family
4	4	Amigos
5	5	Developers

	Id	GroupId	UserId
1	1	1	2
2	2	1	3
3	3	2	2
4	4	3	3
5	5	2	4
6	6	3	7
7	7	3	5
8	8	2	6

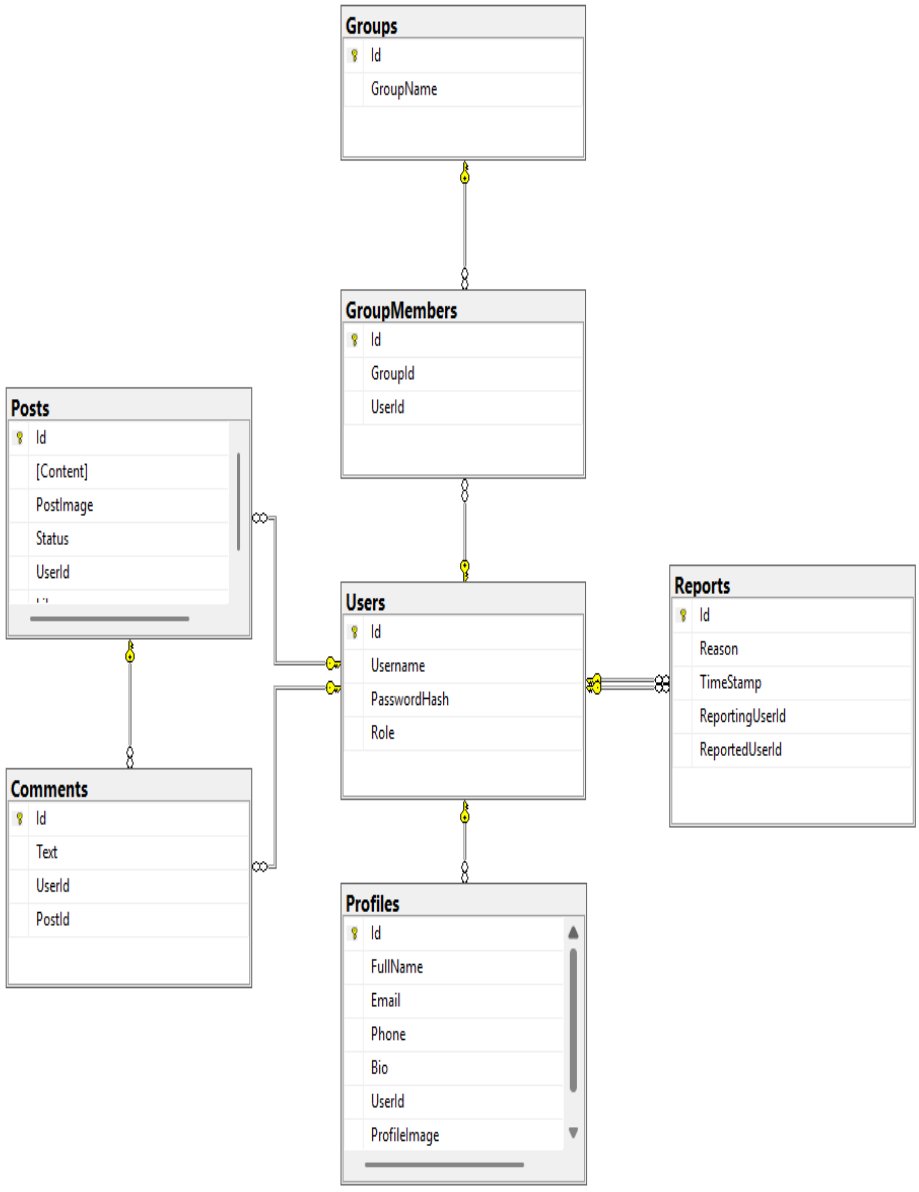
	Id	Reason	TimeStamp	ReportingUserId	ReportedUserId
1	1	FakeProfile	2025-09-06 17:24:49.7593749	2	5
2	2	Spam	2025-09-06 21:43:14.5654699	2	6

	Id	MessageContent	TimeStamp	SenderId	ReceiverId
1	1	Hai Bruh	2025-09-06 21:57:18.5247486	2	3
2	2	string	2025-09-06 21:58:03.6255383	2	1
3	3	Hello Bruh	2025-09-06 21:58:36.3768194	2	6

Query executed successfully. — (localdb)\MSSQLLocalDB (15... | NARASIMHA\snara (54) | ProfileDB | 00:00:00 | 46 rows

ERD (Entity Relationship Diagram):

- Boxes: Users, Profiles, Posts, Comments, Groups, Messages, Reports.
- Connectors: Show 1:1, 1:N, and M:N relationships.

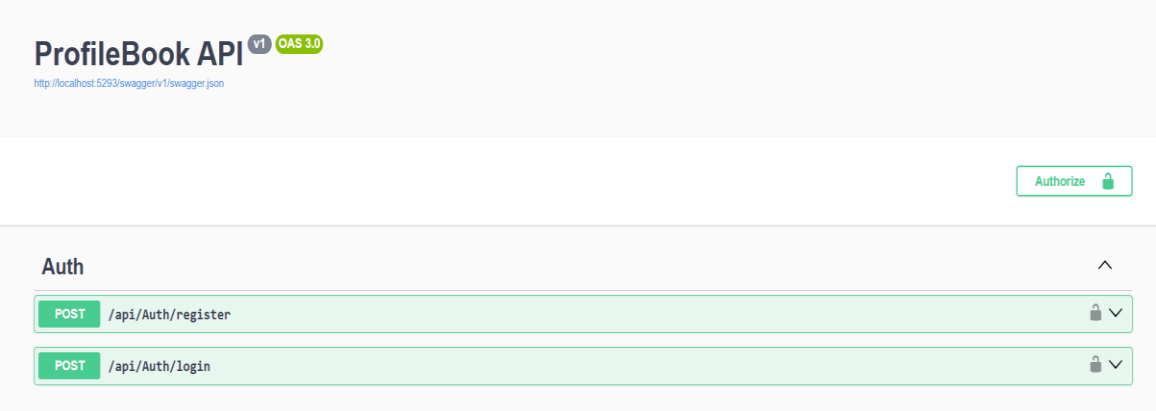


API ENDPOINTS:

AuthController:

- **/api/auth/register (POST)**
  Registers a new user by taking username, email, and password. Returns success message if registration is successful or an error if the email already exists or validation fails.

- **/api/auth/login (POST)**
  Authenticates a user with email and password. Returns a JWT token for authorized access to protected routes.



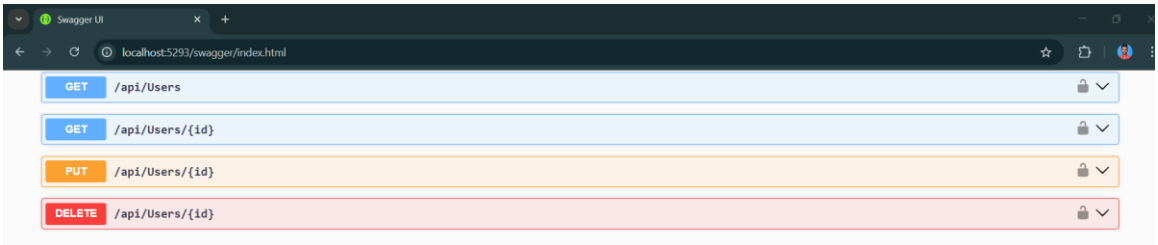
UsersController

- **/api/users (GET)**
  Retrieves a list of all registered users. Useful for admin or listing users in the app.

- **/api/users/{id} (GET)**
  Retrieves details of a specific user by their ID.

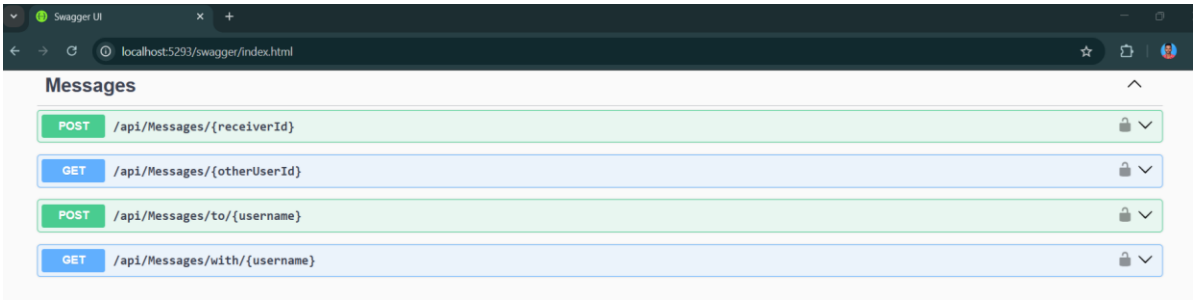
- **/api/users/{id} (PUT)**
  Updates user information like username or email. Requires authentication.

- **/api/users/{id} (DELETE)**
  Deletes a user from the system. Usually restricted to admin users.



MessagesController

- **/api/Messages/{receiverId} (POST)**
  Sends a message to a specific user identified by receiverId. Accepts the message content in the request body.
- **/api/Messages/{otherUserId} (GET)**
  Fetches all messages between the current user and another user identified by otherUserId.
- **/api/Messages/to/{username} (POST)**
  Sends a message to a user identified by their username. Accepts message content in the request body.
- **/api/Messages/with/{username} (GET)**
  Fetches all messages exchanged with a specific user identified by username.

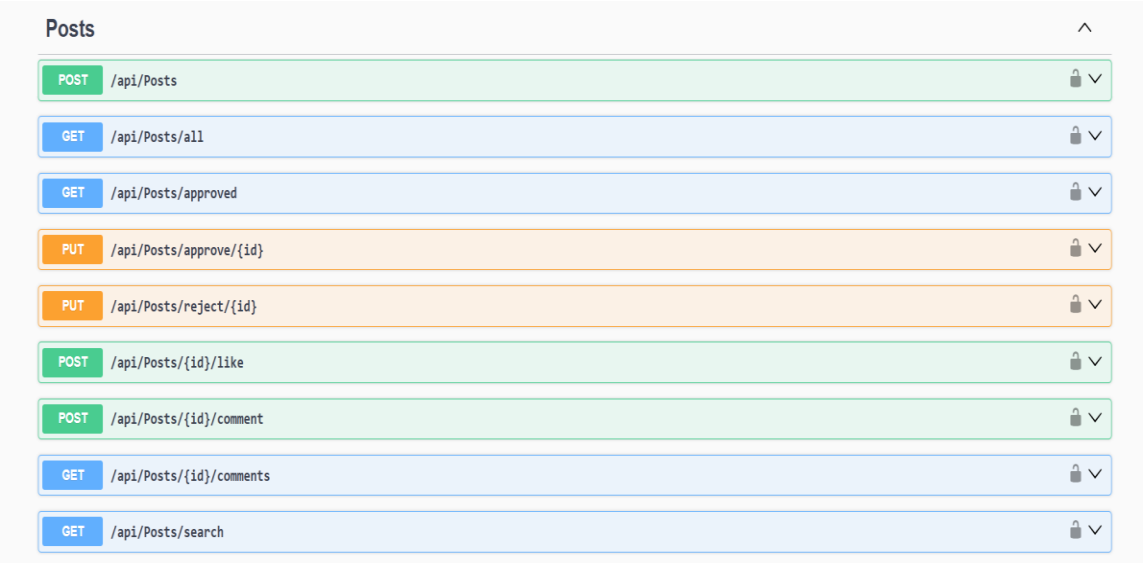


PostsController

- **/api/Posts (POST)**
  Creates a new post. Accepts post data (title, content, etc.) in the request body.
- **/api/Posts/all (GET)**
  Fetches all posts in the system, including both approved and pending posts.
- **/api/Posts/approved (GET)**
  Fetches only approved posts for public display.
- **/api/Posts/approve/{id} (PUT)**
  Approves a post by its ID. Usually restricted to admin users.
- **/api/Posts/reject/{id} (PUT)**
  Rejects a post by its ID. Usually restricted to admin users.
- **/api/Posts/{id}/like (POST)**
  Likes a specific post by its ID and updates the like count.
- **/api/Posts/{id}/comment (POST)**
  Adds a comment to a specific post. Accepts comment text in the request body.

- **/api/Posts/{id}/comments (GET)**
  Fetches all comments for a specific post.

- **/api/Posts/search (GET)**
  Searches posts by keyword. Accepts query parameters like ?q=keyword.



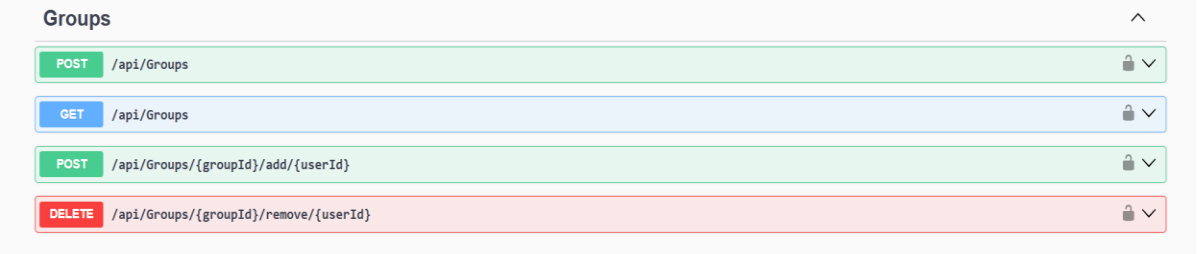
GroupsController

- **/api/Groups (POST)**
  Creates a new group. Accepts group details (like name and description) in the request body.

- **/api/Groups (GET)**
  Fetches all groups in the system.

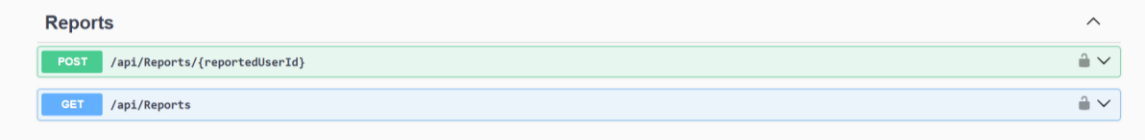
- **/api/Groups/{groupId}/add/{userId} (POST)**
  Adds a specific user to a group. groupId identifies the group, and userId identifies the user to add.

- **/api/Groups/{groupId}/remove/{userId} (DELETE)**
  Removes a specific user from a group. groupId identifies the group, and userId identifies the user to remove.



ReportsController

- **/api/Reports/{reportedUserId} (POST)**
  Reports a specific user. Accepts the reason for reporting in the request body.
- **/api/Reports (GET)**
  Fetches all reports. Usually restricted to admin users.

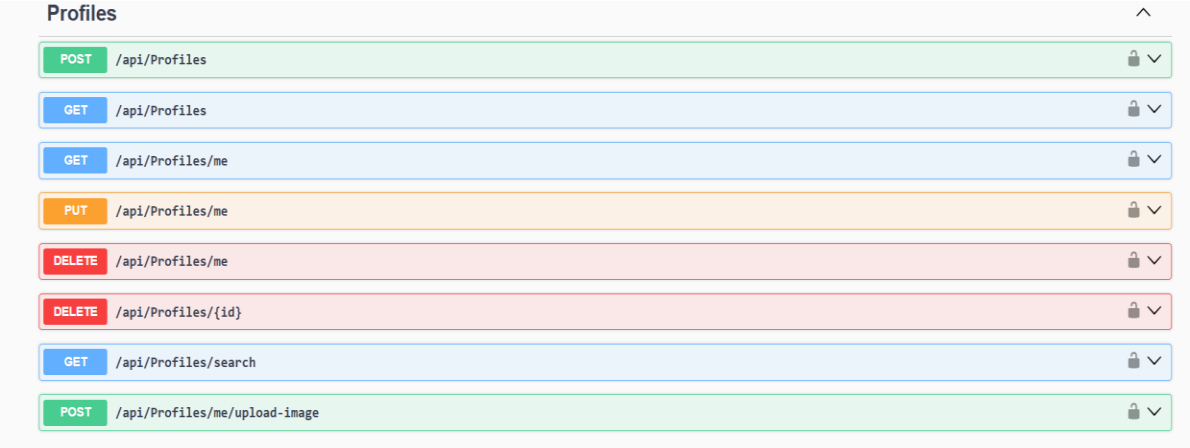


ProfilesController

- **/api/Profiles (POST)**
  Creates a new profile. Accepts profile details (like bio, avatar, etc.) in the request body.
- **/api/Profiles (GET)**
  Fetches all user profiles in the system.
- **/api/Profiles/me (GET)**
  Fetches the profile of the currently logged-in user.
- **/api/Profiles/me (PUT)**
  Updates the current user's profile. Accepts updated profile details in the request body.
- **/api/Profiles/me (DELETE)**
  Deletes the current user's profile.
- **/api/Profiles/{id} (GET)**
  Fetches the profile of a specific user by their ID.

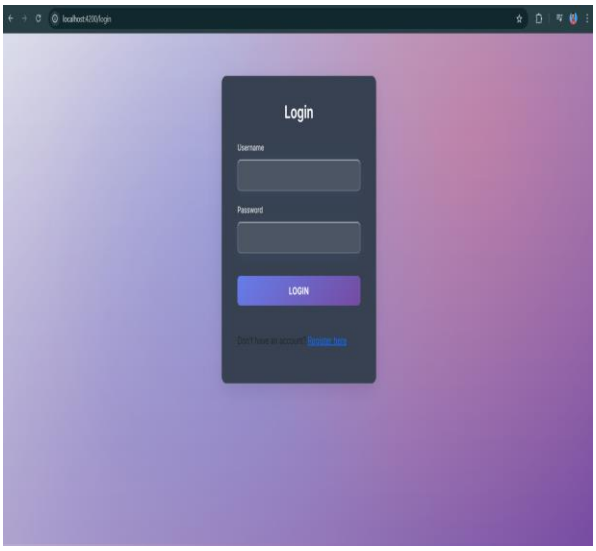
- **/api/Profiles/search (GET)**
  Searches profiles by keyword (like name or bio). Accepts query parameters.

- **/api/Profiles/me/upload-image (POST)**
  Uploads or updates the profile image for the current user. Accepts an image file in the request body.

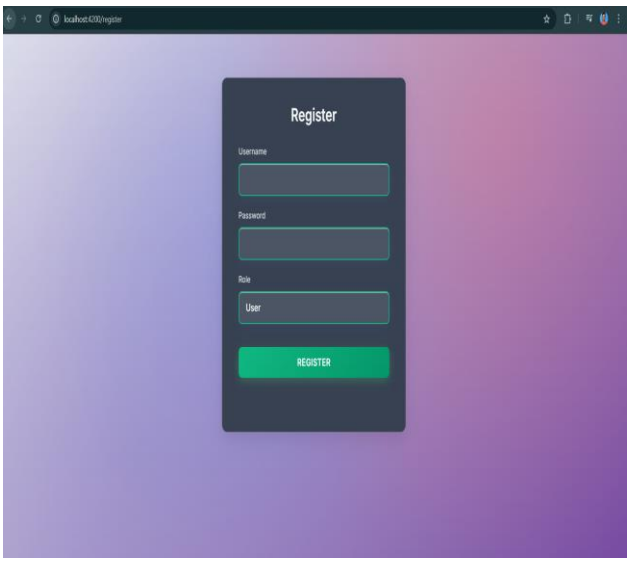


Demo Plan & Screenshots:

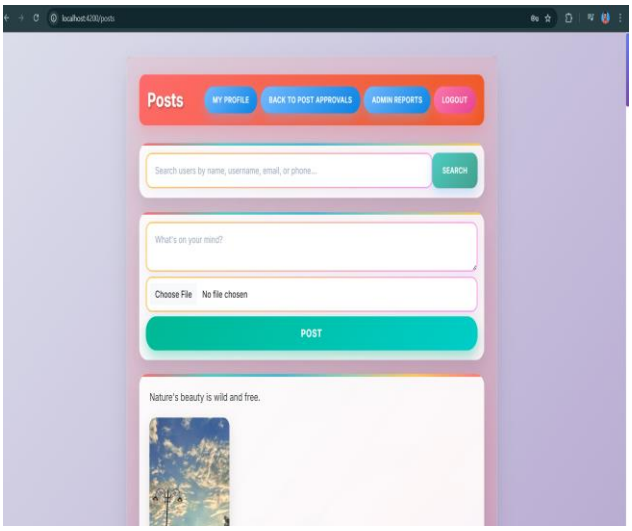
Login:

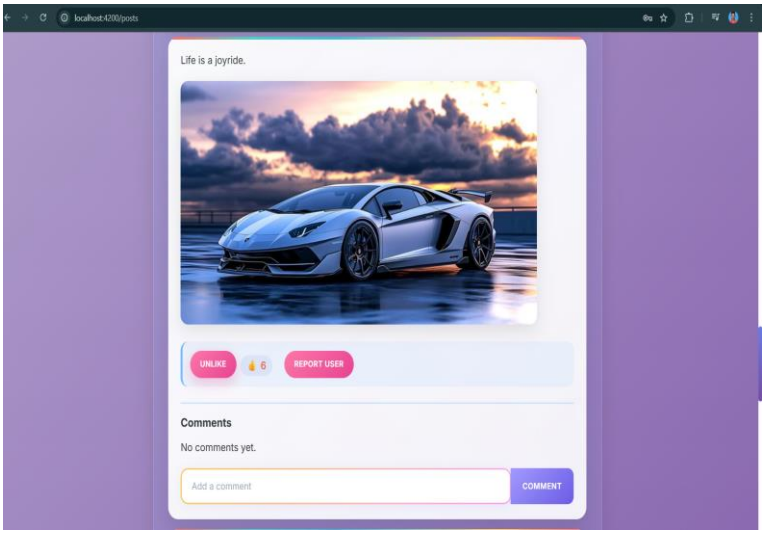


Register:

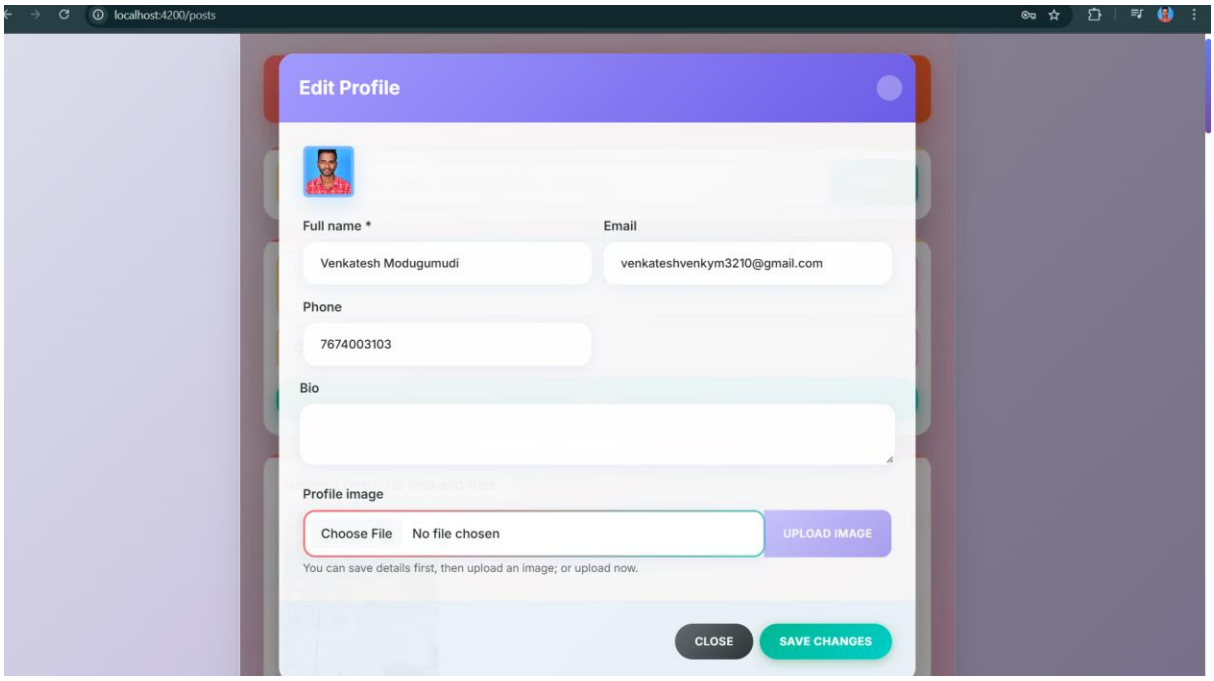


Posts:

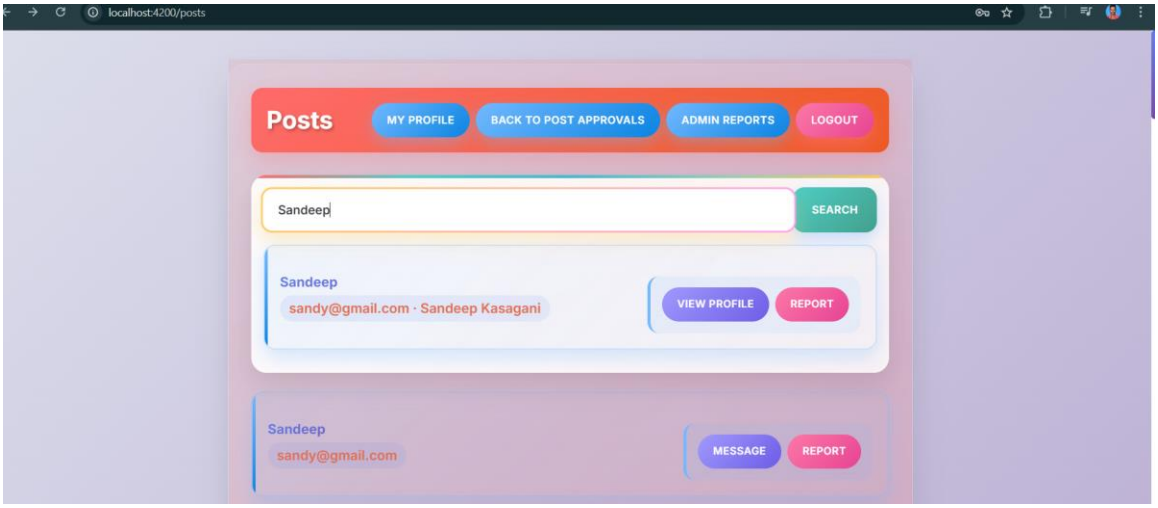




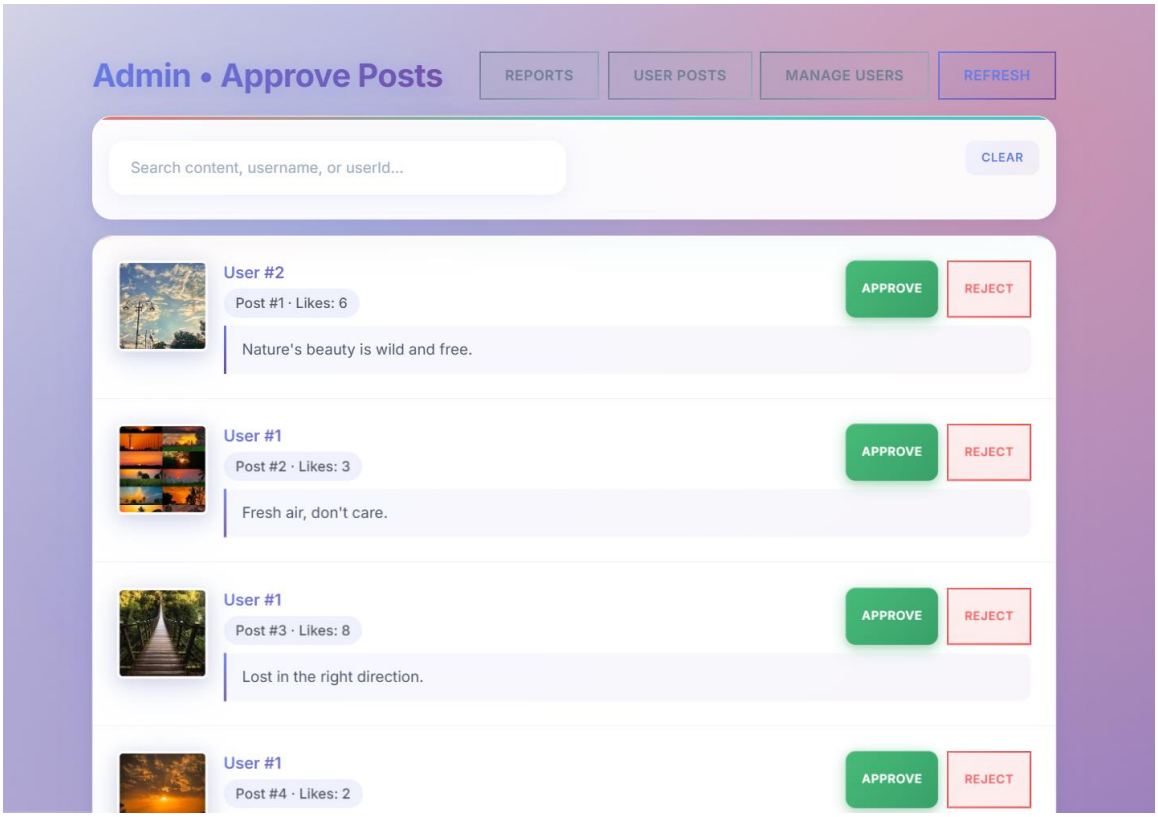
Edit Profile:



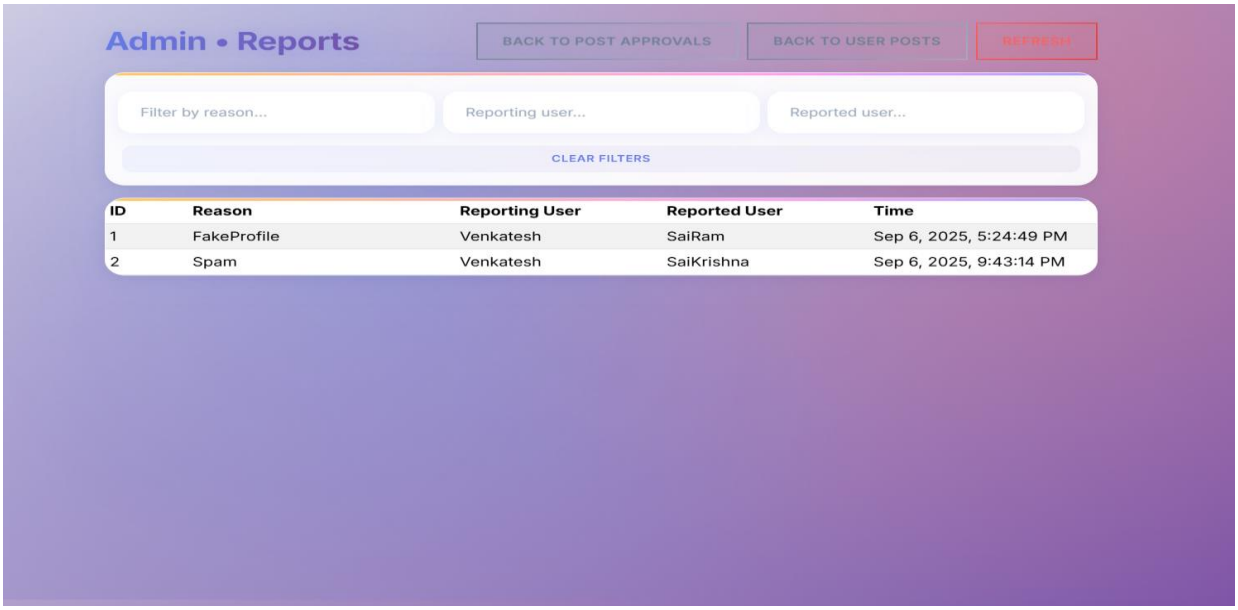
Search User:



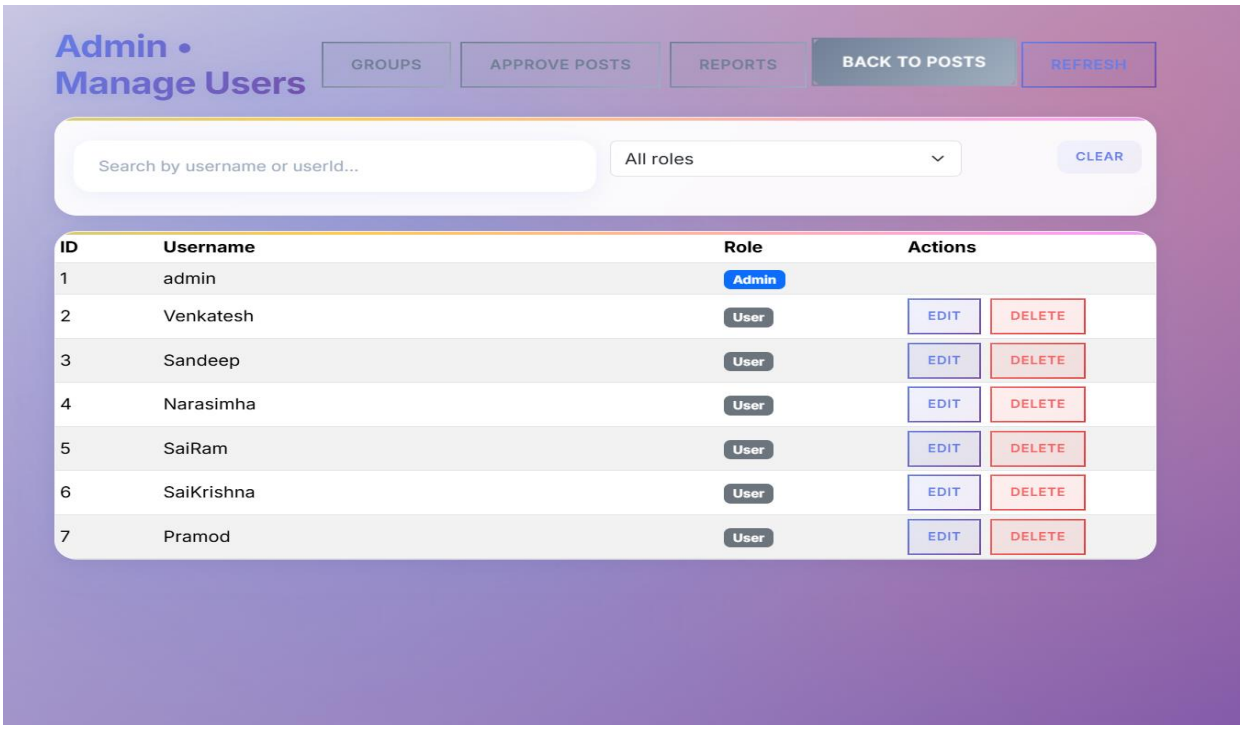
Admin Approve Posts:



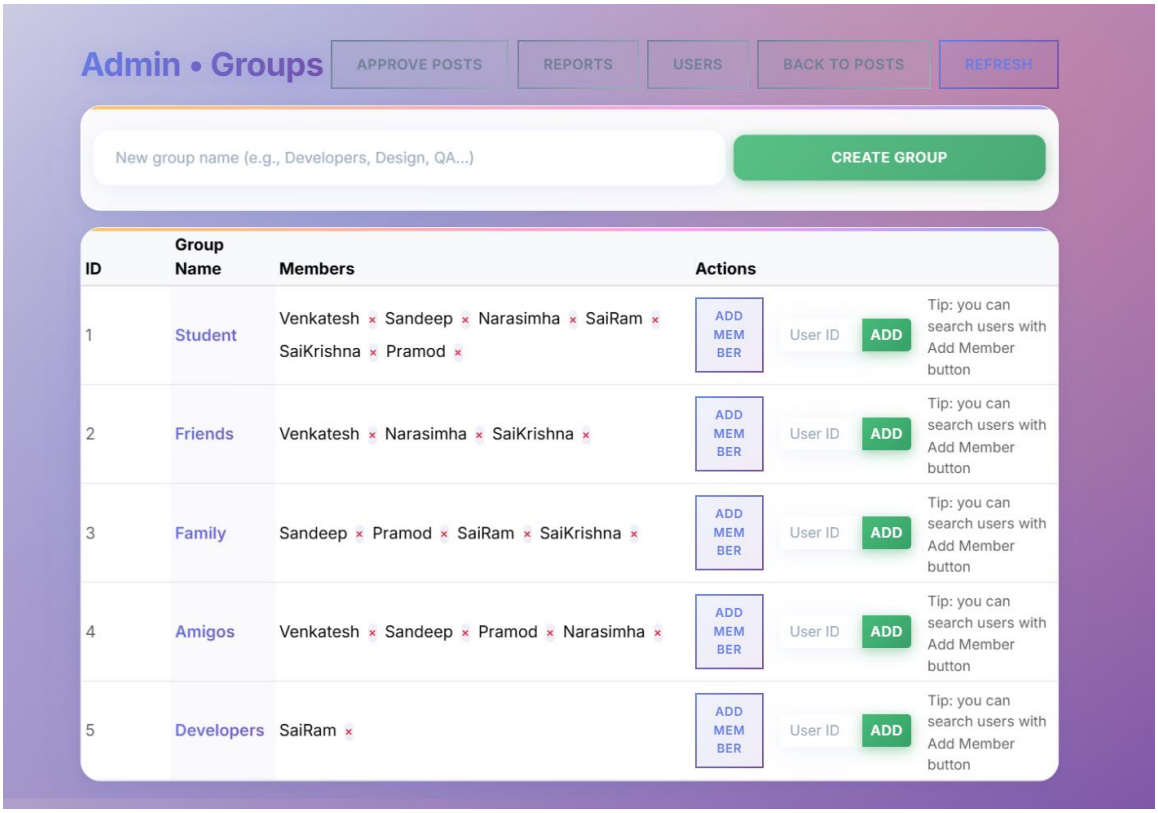
Admin Report:



Admin Manage Users:



Admin Make Groups For Users:



Testing & Security

- **Manual Testing**
    - Each API endpoint was tested using **Postman** or **Swagger UI** to ensure correct functionality.
    - Inputs such as valid/invalid data, missing fields, and edge cases were verified.
    - Responses, status codes, and error messages were checked for accuracy.
- **Unit Testing**
    - Key controller methods (e.g., Posts, Auth, Users) were covered with **unit tests** using frameworks like **xUnit** or **NUnit**.
    - Business logic and data access layers were tested separately to ensure correctness.

- **Integration Testing**
    - Tested end-to-end flows such as **user registration → login → create post → comment** to ensure multiple endpoints work together.
    - Verified database updates reflect accurately after API calls.

Security

- **Authentication & Authorization:** Protected endpoints use **JWT tokens**; only authorized users can access restricted routes.
- **Role-Based Access:** Admins can manage posts, users, and reports; regular users have limited access.
- **Input Validation:** API requests are validated to prevent invalid or malicious data.
- **Data Protection:** Passwords are **hashed**, and sensitive data is never exposed.
- **Common Attack Prevention:** Measures against **SQL Injection** and **XSS** are implemented.

Conclusion & Future Work

Conclusion

The project successfully implements a **full-featured social interaction platform** with user authentication, posts, comments, groups, messaging, profiles, and reporting functionality. All API endpoints are functional and secured with **JWT authentication** and **role-based access control**. The system allows smooth interaction between users while ensuring data integrity and security.

Future Work

- **Enhanced Security:** Implement features like **two-factor authentication** and more robust input sanitization.
- **Real-Time Features:** Add **real-time messaging and notifications** using WebSockets or SignalR.
- **Advanced Search & Filters:** Improve search functionality with filters for posts, users, and groups.
- **Admin Dashboard:** Develop a **comprehensive dashboard** for monitoring reports, user activity, and content approval.
- **Mobile Support:** Build a **mobile-friendly version** or mobile app for better accessibility.