**Full Stack Development with MERN**

**Project Documentation**

**1. Introduction**

* **Project Title:** Connectify
* **Team Members:**

Kshitij Choudhary - Frontend Developer, Quality Assurance Engineer/Tester

Chidambaram Suresh - Frontend Developer, UI/UX Designer

Aakarsh Sinha - Backend Developer, Database Administrator

Ashank Sethi - Backend Developer, Database Administrator, Security Manager

**2. Project Overview**

* **Purpose:** The purpose of this project is to develop a full-fledged social media application using the MERN stack (MongoDB, Express.js, React, Node.js). The app aims to provide users with a platform where they can create, share, and interact with posts, fostering a sense of community and engagement among users.
* **Features:** User Authentication and Authorization - Secure user registration and login using JWTs.

User Profile Management - Profile Picture, Bio Section, Post Wall

Post Creation and Management - Create posts with support for multimedia content.

Comments Management - Users can comment on posts.

Likes - Users can like posts and view likes count.

Home Feed - The user can see relevant posts by scrolling down.

Search Functionality - Users can search for other users

Friends Management - Sending friend requests, accepting friend requests, maintaining a list of

friends.

Direct Messaging - Real-time messaging between two users. (Websockets)

Responsive Design - Compatibility with various devices and screen sizes

**3. Architecture**

* **Frontend:** The app primarily uses functional components with React Hooks for state and lifecycle management. Utilizes useState, useEffect, useContext, and other hooks for managing local component state and side effects. Common UI elements and functionalities are encapsulated in reusable components. Separation of concerns is maintained. Implements client-side routing using React Router,. Uses CSS-in-JS solutions such as styled-components and Tailwind. Uses Axios to make HTTP requests to the backend server. Uses WebSockets for real-time updates.

* **Backend:** An Express.js server is set up to handle incoming HTTP requests and route them to appropriate handlers. Uses various middleware for request parsing, authentication, error handling, and more. Routes are organized in a modular fashion. Follows RESTful principles for defining API endpoints. Controllers contain the logic for handling specific routes. Uses MongoDB as the database and Mongoose for defining schemas and interacting with the database. Uses JSON Web Tokens (JWT) for user authentication and session management. Middleware for verifying JWT tokens and ensuring that routes are accessible only to authenticated users. Uses Socket.io for real-time communication.

* **Database:** MongoDB is the database for storing user data, posts, comments, and other entities. Mongoose is an Object Data Modelling (ODM) library for MongoDB to define models and interact with the database. The User schema defines the structure of user documents in the database which includes email, password, username, profile picture. The Post schema captures information about user posts which includes userId, likes, caption, username, image, comments, date. The messages model stores information about messages sent between users. The data includes sender, receiver, message, and timestamp. The friends model stores userId and the array of friends of the user.

**4. Setup Instructions**

* **Prerequisites:** Node.js, Express.js, MongoDB, Mongoose, jsonwebtoken, cors, dotenv, body-parser, express-validator, react, react-router-dom, axios, nodemon, socket.io, zod, express-async-handler.
* **Installation:**

Step 1 - Clone the Repository

- git clone https://github.com/Aakarsh-Sinha/Connectify.git

Step 2 - Navigate to Project Directory

- cd Connectify

Step 3 - Install Server-side (Backend) dependencies

- cd server

- npm install

Step 4 - Set up Environment Variables

- touch .env

Add necessary environment variables

NODE\_ENV=development

PORT=<INT>

JWT\_SECRET=<STRING>

MONGO\_URI=<STRING>

Step 5 - Install Client-side (Frontend) Dependencies

- cd ../client

- npm install

Step 6 - Navigate to Root Directory

- cd..

Step 7 - Install Root Directory Dependencies

- npm install

Step 8 - Set up Environment Variables

- touch.env

Add necessary environment variables

NODE\_ENV=development

PORT=<INT>

JWT\_SECRET=<STRING>

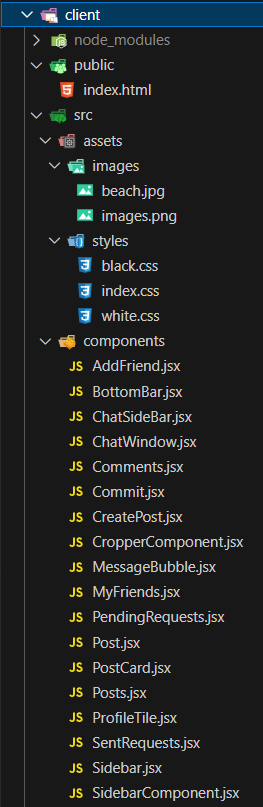
MONGO\_URI=<STRING>

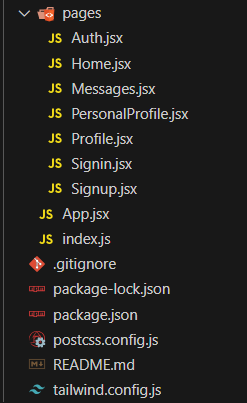
Step 8 - Run Frontend and Backend concurrently

- npm run dev

**5. Folder Structure**

* **Client:**





index.html: The main HTML file that serves the React application.

**src/**

This is the main directory for the React application source code.

**assets/**

* images/: Directory for storing image files used in the application.
* styles/: Global CSS files and theme configurations.

**components/**

This directory contains reusable UI components. .

* ChatWindow.jsx: Component for displaying chat between two users.
* Comments.jsx: Component for displaying and managing comments
* ProfileTile.jsx: Component for displaying user profiles.
* Posts.jsx: Component for displaying and managing posts.

**pages/**

This directory contains components that represent full pages in the application.

* Home.jsx: Home page of the application.
* Signin.jsx: Login page for user authentication.
* Signup.jsx: Signup page for new users.
* Messages.jsx: Page for messaging other users.

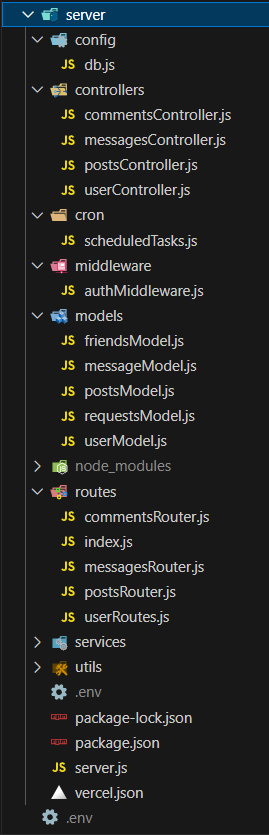
**App.js**  **x**

The root component of the application, contains the main layout and routing logic.

**index.js**

The entry point of the React application, rendering the root component into the DOM.

* **Server:**



**config/**

This directory contains configuration files.

* db.js: Contains the MongoDB connection logic using Mongoose.

**controllers/**

This directory contains controller files that handle the logic for each route.

* messagesController.js: Handles logic related to messages.
* postsController.js: Handles logic related to posts.
* userController.js: Handles logic related to user profiles and user data.
* commentsController.js: Handles logic related to comments on posts.

**cron/**

Contains scripts that run scheduled tasks

**middleware/**

This directory contains middleware functions.

* authMiddleware.js: Middleware for verifying authentication tokens.

**models/**

This directory contains Mongoose schema definitions for different collections.

* userModel.js: Schema for user data.
* postsModel.js: Schema for posts, including likes and comments
* friendsModel.js: Schema for friends
* messagesModel.js: Schema for messages
* requestsModel.js: Schema for friend requests

**routes/**

This directory contains route definitions.

* messagesRouter.js: Routes for messages-related endpoints.
* postsRouter.js: Routes for post-related endpoints.
* userRoutes.js: Routes for user-related endpoints.
* commentsRouter.js: Routes for comment-related endpoints.

**utils/**

This directory contains utility functions and helpers.

**.env**

A file for storing environment variables.

**server.js**

The entry point of the backend application. It sets up the Express server, connects to the database, and configures middleware.

**6. Running the Application**

* Provide commands to start the frontend and backend servers locally.
  + **Frontend:** npm start in the client directory.

     npm run client in the root directory.

* + **Backend:** npm start / nodemon server.js / node server.js in the server directory.

        npm run server in the root directory.

* + **Frontend and Backend :** To run concurrently, npm run dev in the root directory.

**7. API Documentation**

**User Authentication**

* **POST /api/user/signup** - Registers a new user.
* **POST /api/user/signin** - Authenticates a user and returns a token.

**User Management**

* **GET /api/getusers/ -** Retrieves user information by userID.
* **GET /api/user/getprofile** - Retrieves user profile by userID.
* **GET /api/user/getpfp –** Retrieves user profile picture by userID.
* **POST /api/user/updatepfp –** Updates user profile picture.

**Friend Management**

* **GET /api/user/getfriends –** Retrieves all the friends of a user.
* **GET /api/user/sentrequests –** Retrievesthe active friend requests sent by a user.
* **GET /api/user/receivedrequests –** Retrieves the active friend requests received by a user.
* **POST /api/user/acceptrequest –** Accepts friend requests sent by other users.

**Posts Management**

* **POST /api/posts/createpost** – Creates a new post by authenticated user.
* **GET /api/posts/getposts** – Retrieves posts to display in user feed.
* **GET /api/posts/userposts** – Retrieves all the posts created by a specific user through userID.
* **POSt /api/posts/like/:postID –** Allows the user to like a post.

**Comments Management**

* **GET /api/comments/:postID** – Retrieves all the comments on a post by postID.
* **POST /api/comments/:postID** – Creates a new comment on a post.

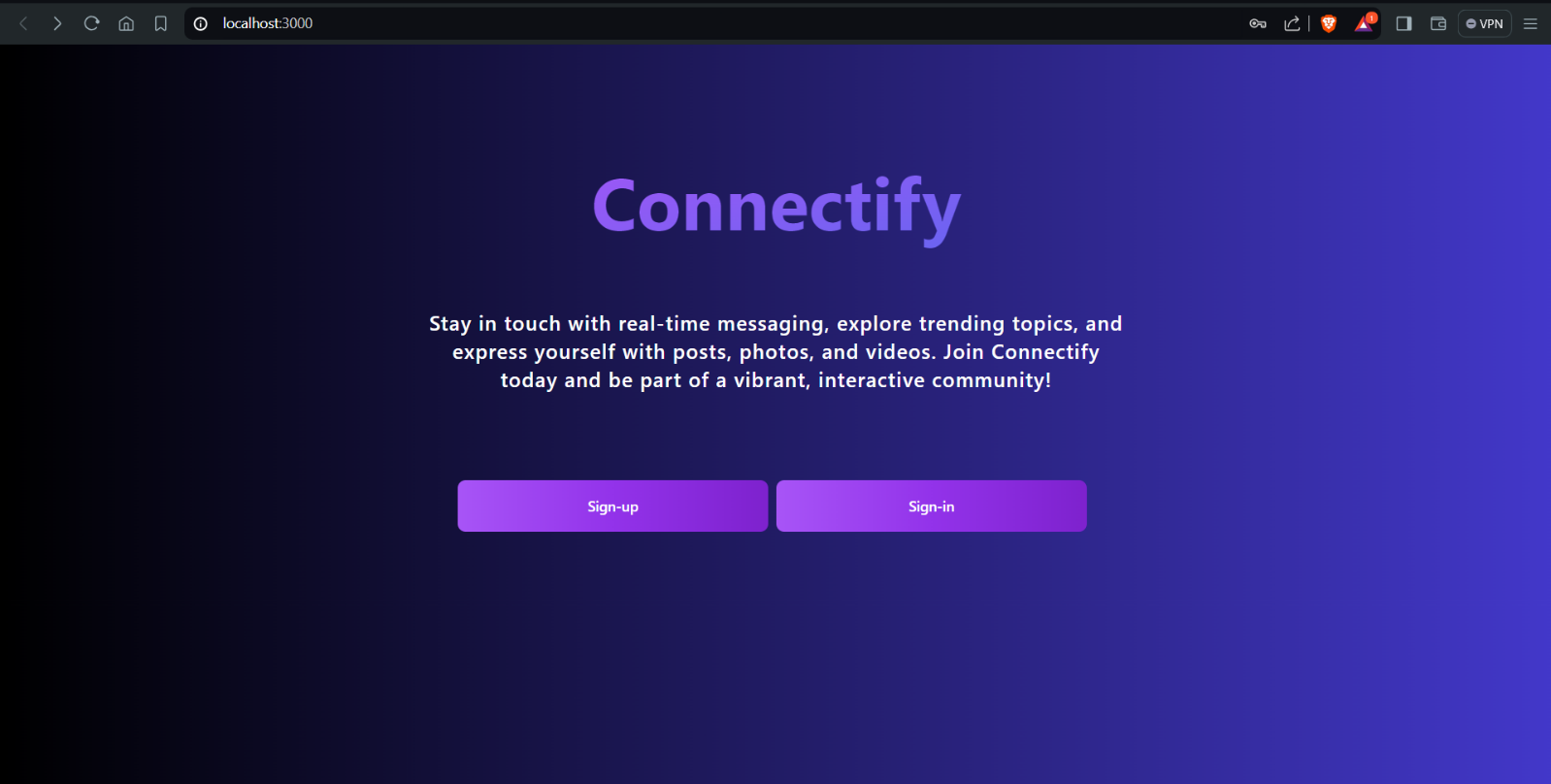
**Direct Messaging**

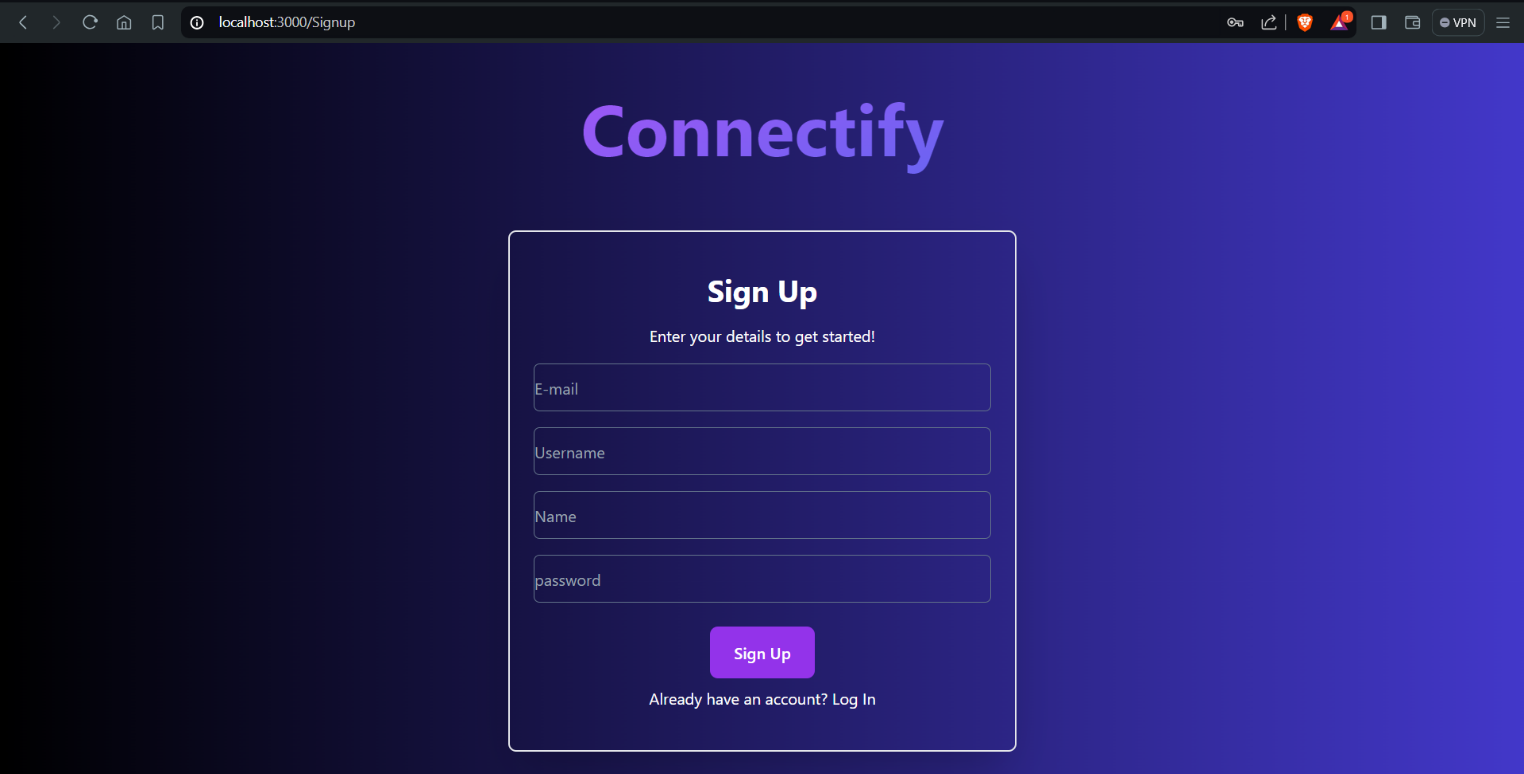
* **GET /api/comments/:postID** – Retrieves all the comments on a post by postID.
* **POST /api/comments/:postID** – Creates a new comment on a post.

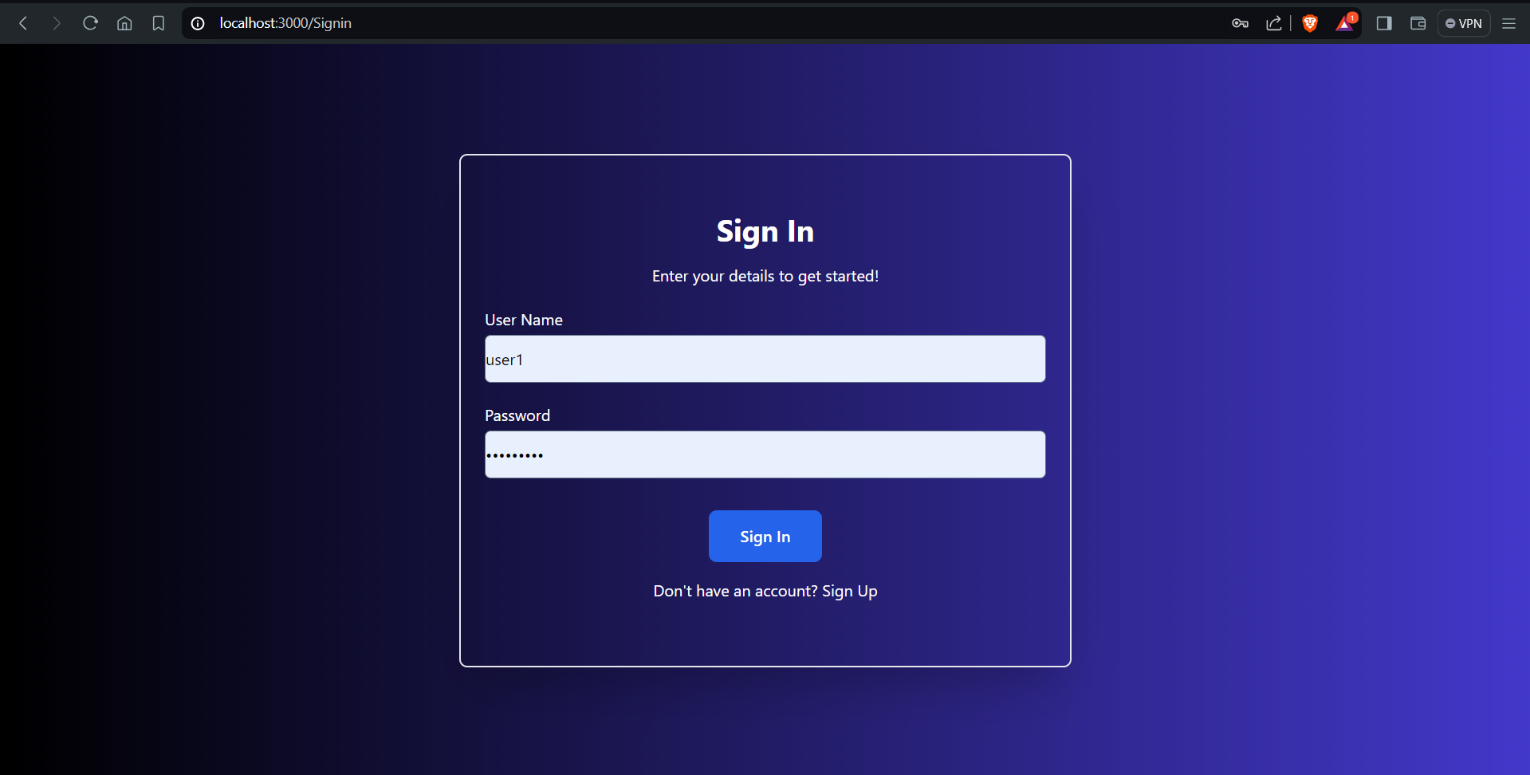
**8. Authentication**

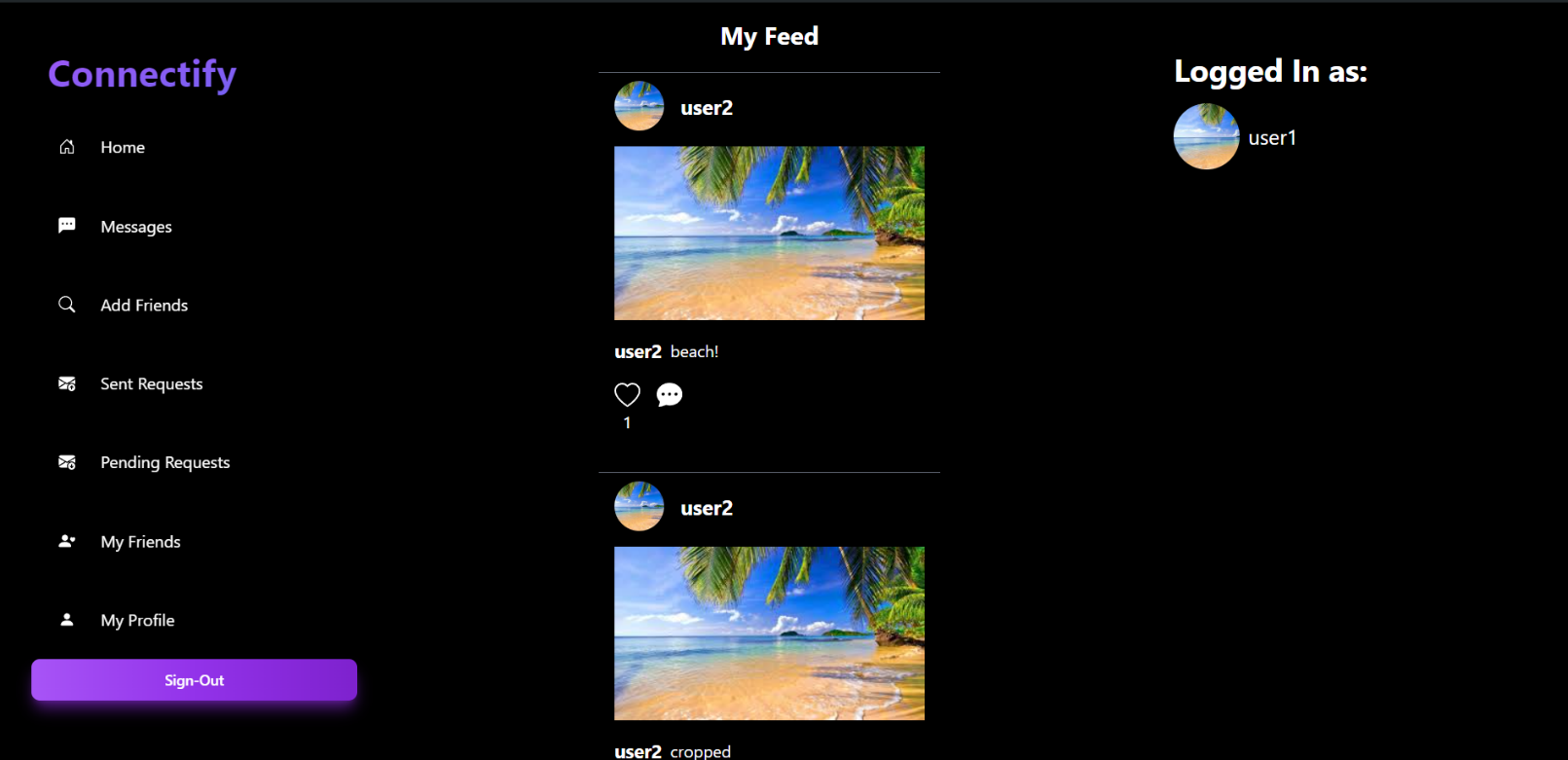
* JSON Web Tokens are used to handle authentication. A user signs up by providing a username, email, and password. The password is securely stored in the database. Upon successful registration, the server generates a JWT and sends it back to the client. A user logs in by providing their email and password.The server compares the provided password with the password in the database. If the credentials are valid, the server generates a JWT and sends it to the client. The JWT contains a payload with user-specific information It is signed with a secret key to ensure its integrity and prevent tampering. The token is sent to the client and can be used for subsequent requests to authenticate the user. On the client side, the JWT is typically stored in local storage or cookies. An authentication middleware is used to verify the JWT provided in the Authorization header of requests. The middleware checks if the token is valid and extracts the user’s information from the token. If the token is valid, the request proceeds; otherwise, the server responds with an unauthorized error.

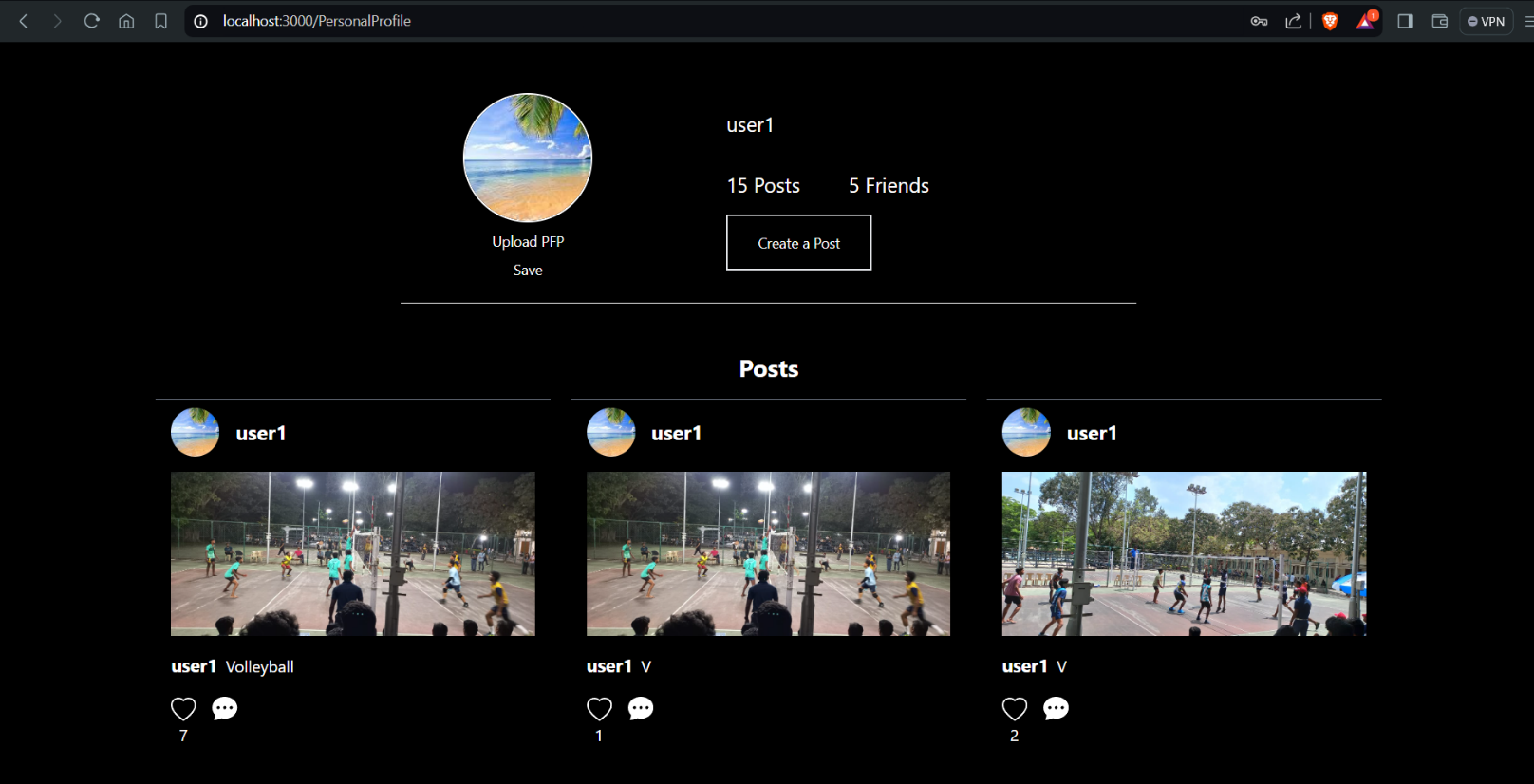
**9. User Interface**

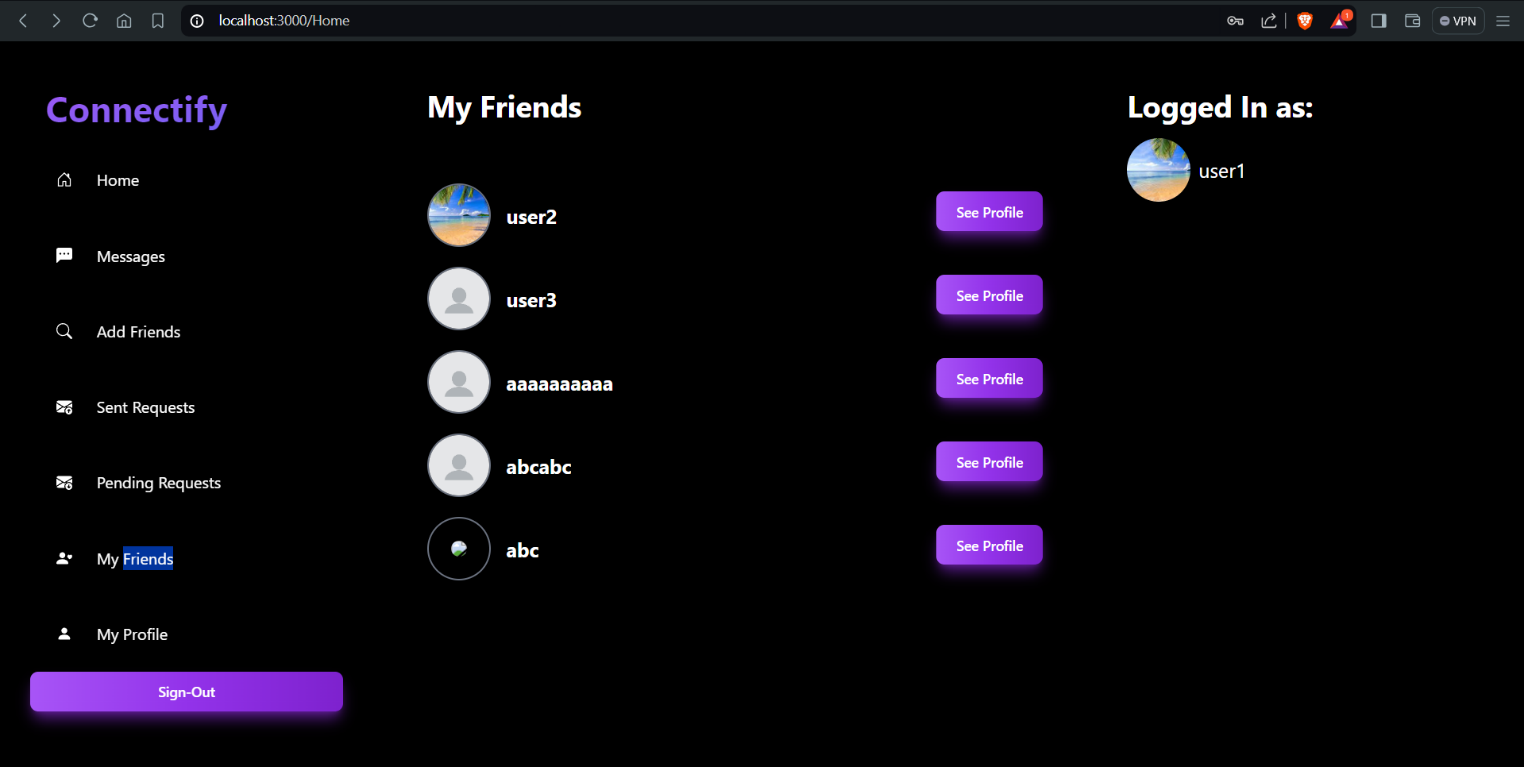


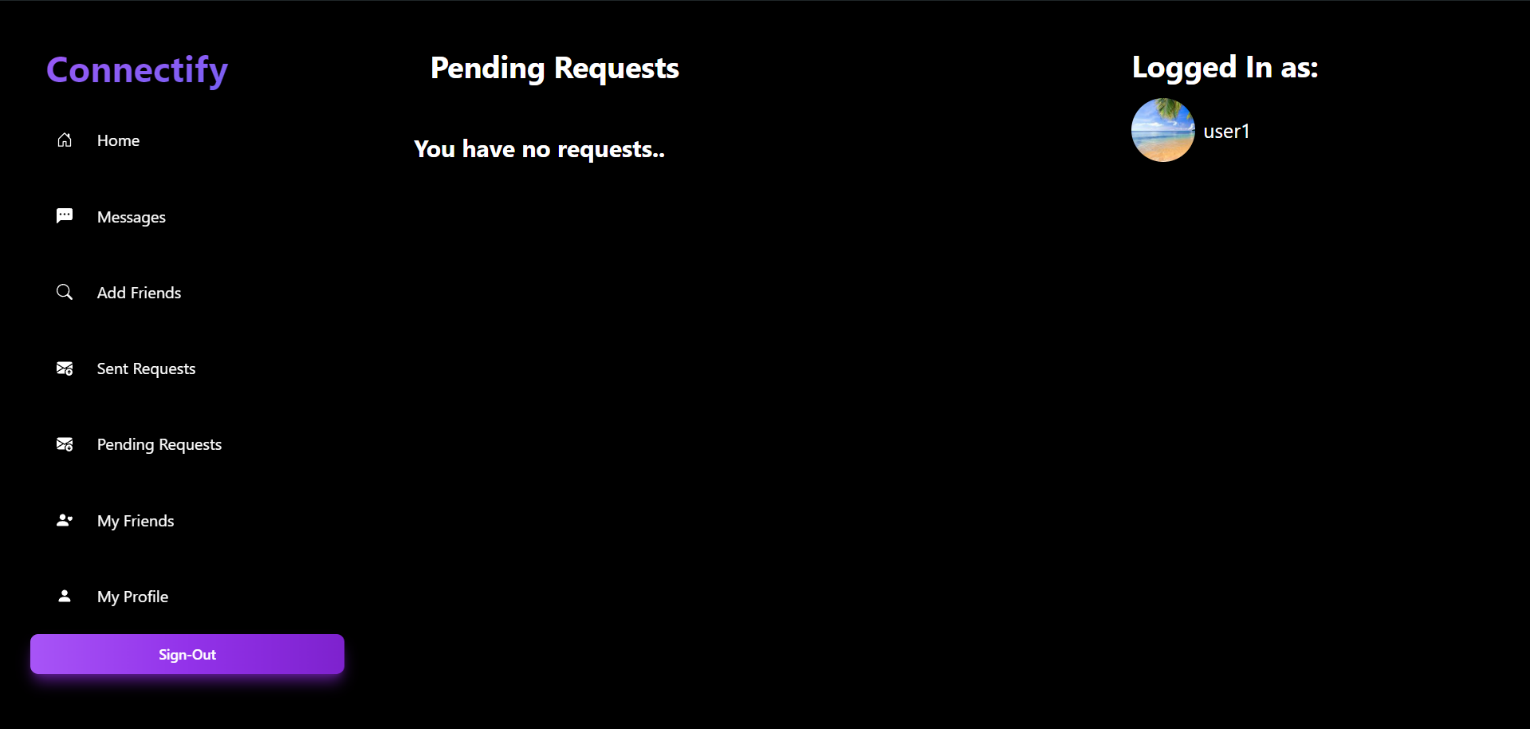


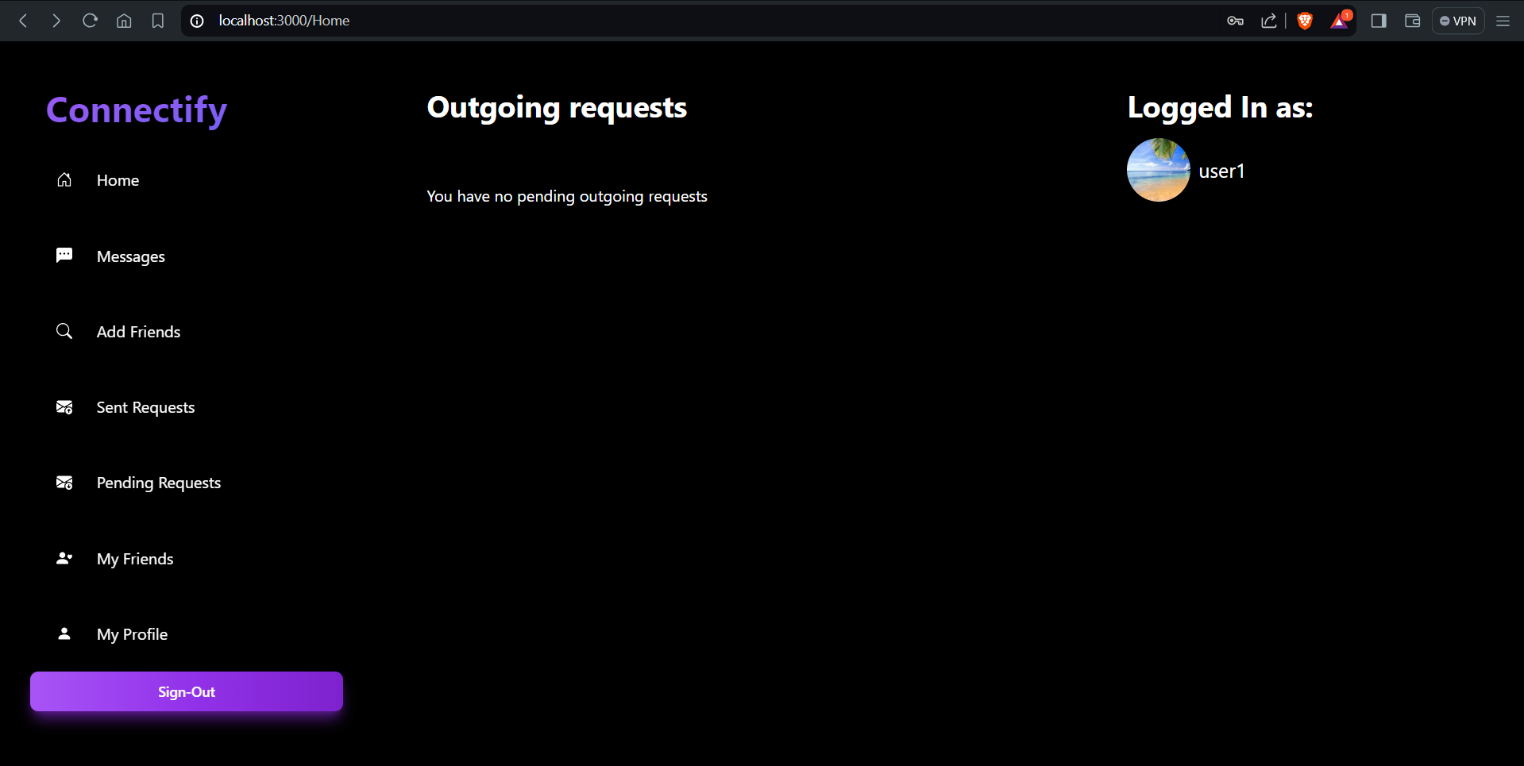


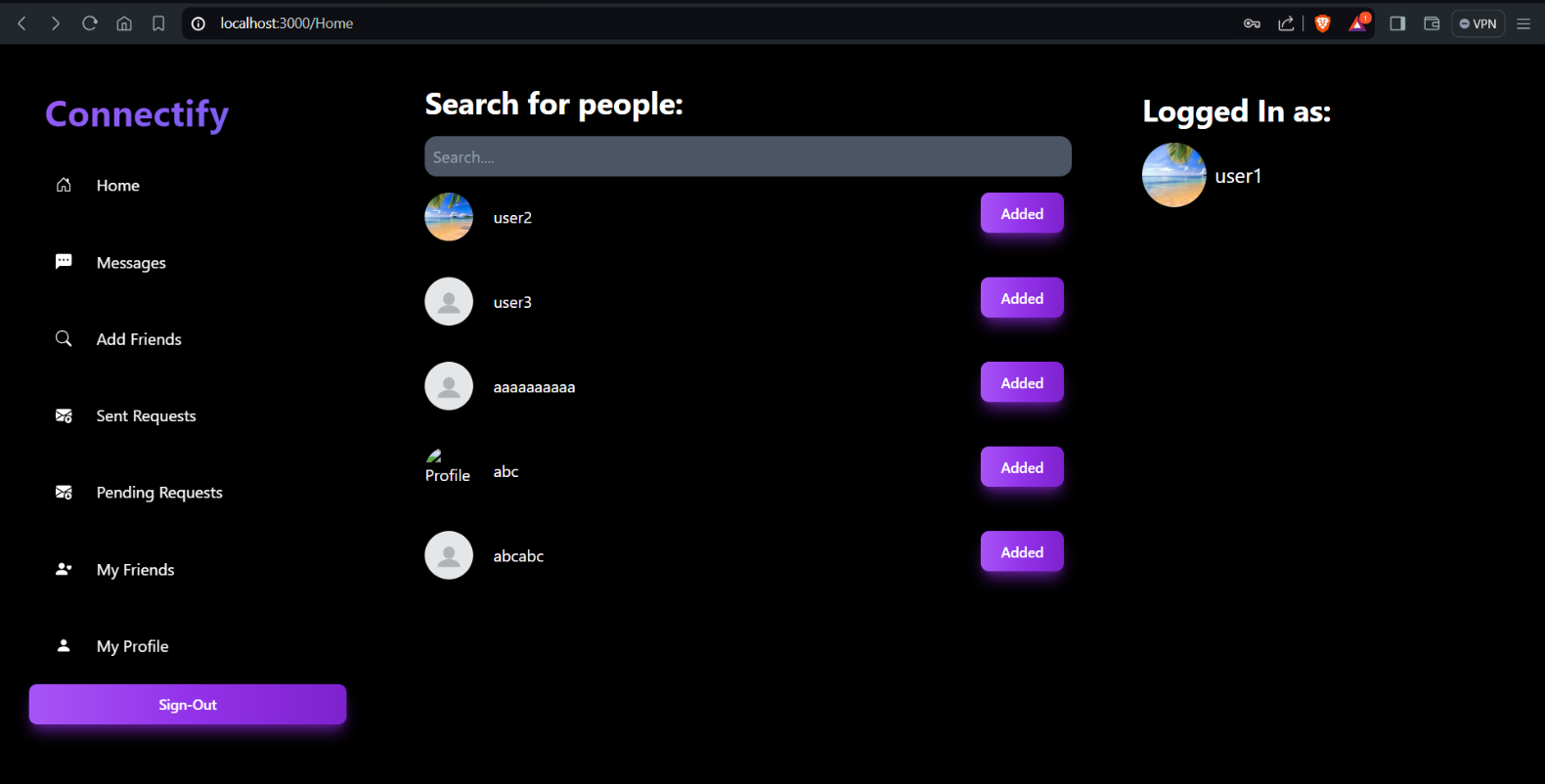


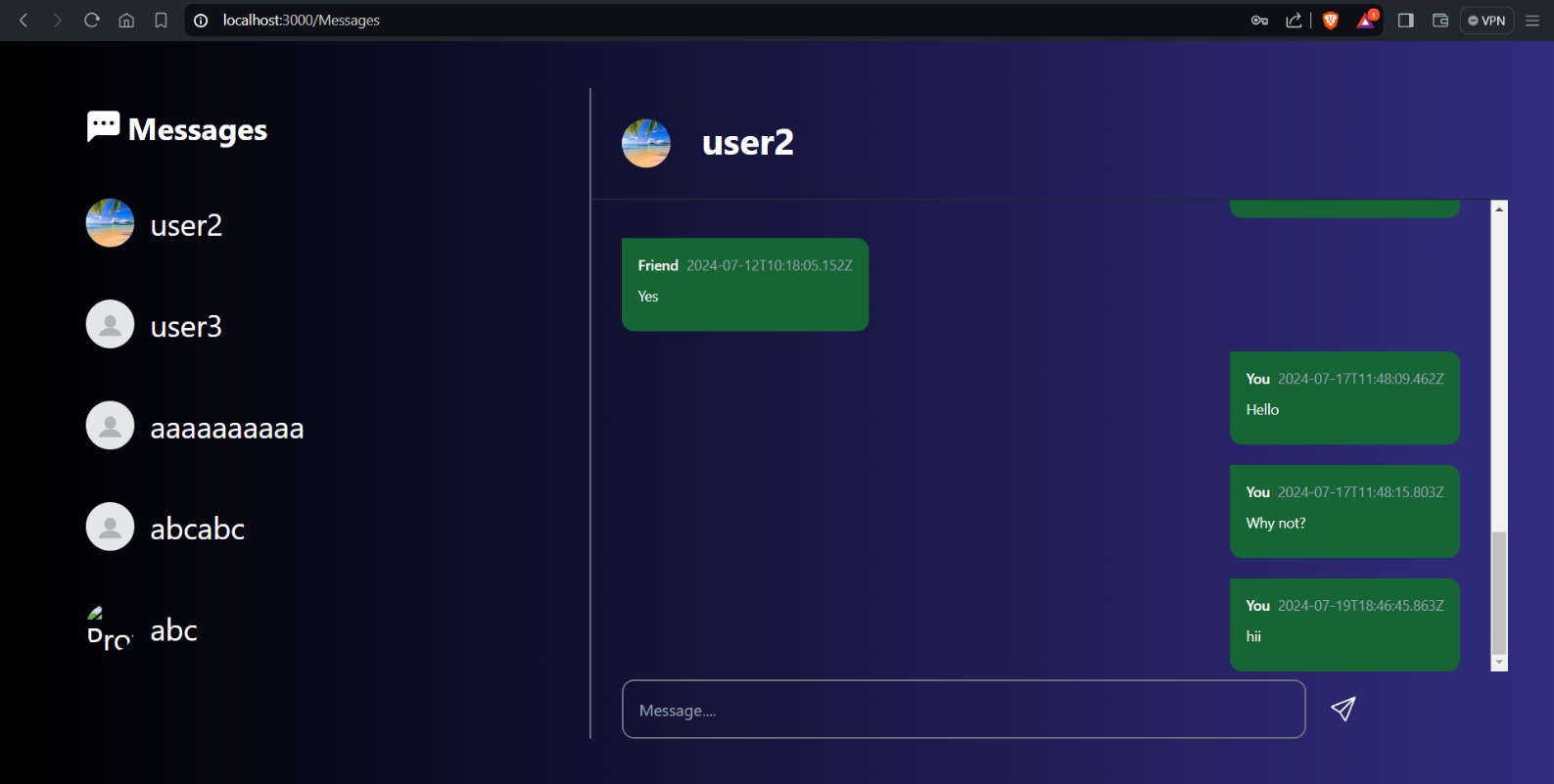












**10. Testing**

**Unit Testing:** To test individual components or functions in isolation, ensuring they perform as expected.

**Integration Testing:** To test the interaction between different parts of the application, such as API endpoints interacting with the database.

**End-to-End (E2E) Testing:** To test the application as a whole, simulating real user scenarios.

**Performance Testing:** To evaluate the performance of the application under various conditions and loads.

**Security Testing:** To identify and fix security vulnerabilities in the application.

**Usability Testing:** To ensure that the application is user-friendly and meets the needs of its users.

**11. Screenshots or Demo**

* https://drive.google.com/file/d/1QUaWiGP7UsbHi1ArIO2uBFh\_O\_ooRSWZ/view?usp=sharing

**12. Known Issues**

Although the bugs faced in the past were resolved, these were the issues which were consistent across different browsers:

* User authentication: Failure to handle token expiration led to user session unexpectedly terminating.
* Sometimes the messages were not being sent and it was consistent along different browsers.

**13. Future Enhancements**

* Outline potential future features or improvements that could be made to the project.

**Group chats and forums** - Allow users to create their own private communities.

**User Notifications**: Implement a notification system to alert users about new likes, comments, and friend requests

**Comment Replies**: Add the ability for users to reply to comments, creating threaded conversations.

**Content Moderation**: Add basic content moderation tools to report and remove inappropriate content.

**Multilingual Support**: Introduce support for multiple languages to make the app accessible to a wider audience.

**Basic Analytics**: Provide users with basic analytics to see how their posts are performing in terms of likes and comments.

.