

**BAHAUDDIN ZAKARIYA UNIVERSITY MULTAN, SUB
CAMPUS LODHRAN, PAKISTAN**



A Project Report On:

“Threads Clone”

(Social Media Web Application)

*Submitted in the partial fulfilment of the requirements for the award of the
Degree of **Bachelor of Science in Information Technology***

Submitted by

MUHAMMAD ZEESHAN MUKHTAR

LDBTT-20-09

Under the support and guidance of

Mr. Muzammil Mahboob,
Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

Session (2020-24)

FINAL APPROVAL

It is hereby certified that we have carefully reviewed the report titled '**Threads Clone Social Media Web App**' authored by **Muhammad Zeeshan Mukhtar**. In our evaluation, we find this report to meet the necessary standards for acceptance by the Department of Information Technology, Bahauddin Zakariya University Multan, Sub Campus Lodhran, in partial fulfillment of the requirements for the Bachelor of Science in Information Technology.

Approved on_____

By

Committee:

1. External Examiner

Examiner Name
Designation
Organization

2. Supervisor

Mr. Muzammil Mahboob
Assistant Professor,
Department of Information Technology

3. Head of Department

Mr. Muzammil Mahboob
Assistant Professor,
Department of Information Technology

DEDICATION

To our parents, teachers, and all those who offered their prayers for our success. I extend a heartfelt expression of gratitude to my beloved parents. Additionally, I dedicate this dissertation to my cherished friends and supportive family members, whose unwavering encouragement has been a source of strength throughout this journey.

ACKNOWLEDGMENTS

With the blessings of the Almighty and the unwavering support of our parents, I have embarked on this journey to attain the goal of completing my bachelor's degree. Despite the inherent complexity and intricacies of this project, I dedicated my utmost effort to achieve this milestone. I am profoundly grateful to our esteemed educators, and I extend a special expression of gratitude to our final year project supervisor, Mr. Kamran Qadir. His invaluable guidance, insightful suggestions, and continuous encouragement served as a constant wellspring of inspiration throughout the entire project.

Lastly, I want to convey my deepest gratitude to our parents, friends, and family members for their unwavering prayers, unending support, and relentless encouragement during the long and arduous journey to complete this work on schedule.

Muhammad Zeeshan Mukhtar

ABSTRACT

The "Threads Clone" social media web application represents a comprehensive and user-centric platform designed to facilitate seamless user interactions and real-time communication. This detailed documentation explores the project's multifaceted features, demonstrating its capabilities in providing a unified social media experience. "Threads Clone" integrates advanced technologies, including AI-driven content generation and real-time chat functionality, making it a valuable tool for social engagement and communication.

Focusing on user-friendliness, the application boasts an intuitive design, customizable themes, and a responsive interface suitable for various devices. Robust authentication mechanisms, such as JWT and Bcrypt, ensure secure user login and data protection. The platform supports core social media functionalities, including post creation, image attachments, likes, replies, user profiles, and follow/unfollow interactions. Additionally, real-time updates and notifications enhance user engagement and connectivity.

The project's scalability, coupled with its continuous development and enhancement potential, underscores its commitment to delivering an exceptional social media experience. "Threads Clone" not only meets current social interaction needs but also sets the stage for future innovations and improvements in the social media landscape.

PROJECT BRIEF	
Project Name	Threads Clone
Developed By	Zeeshan Mukhtar
Supervised By	Mr. Muzammil Mahboob
Starting Date	February 05, 2024
Completion Date	June 05, 2024
Computer Used	Lenovo K14, 16GB, 256GB
Operating System	Windows 10 Enterprise
Tech Used	React 18, Node.js, Express.js
DBMS Used	MongoDB (NOSQL)
Tools	VS Code

Contents

FINAL APPROVAL	ii
DEDICATION	iii
ACKNOWLEDGMENTS	iv
ABSTRACT	v
PROJECT BRIEF	vi
Background:	2
Purpose and Scope:	2
Problem Statement:	2
Project Goals and Objectives:	3
Overview of Social Media Platforms:	5
Challenges and Limitations:	5
Technology Stack:	7
Backend Technologies:	7
Functional Requirements:	8
Non-Functional Requirements:	10
Advantages of Proposed System:	10
Overview of the Architectural Design	13
Use Case Diagram: Core Platform	13
Use Case Diagram: User interaction features	14
Use Case Diagram: Ai features	15
Use Case Diagram: Chat functionality	16
Use Case Diagram: Complete overview	17
Threads-Full-Stack: Activity diagram	18
UML Diagram:	19
Project Structure	19
Frontend Design	20
Sequence Diagram:	20
Backend Design	21
Database Design	22
UX/UI Design	23
Responsive Design	25
Frontend Development:	28
Component Development:	29
State Management:	29
Building the Home Page:	31

Design and Layout:	32
Home page design:	34
AI Post creation feature:	35
Backend Development:	41
Project Setup	41
1. Initialize the Backend Project:	41
2. Project Structure:	43
3. Configuration:	43
4. Starting the Server:	44
Database Models	44
Middleware	46
Client-side Form Validation	49
Client-side Validation Process	50
Handling Validation Errors	50
Server-side Error Handling	51
Server-side Validation Process	52
Error Notification	52
Deployment Process	55
Hosting Providers	57
Visual representation of how we add environment variables:	58
Summary	59
Key Challenges Faced	61
Solutions Implemented	62
Planned Future Features and Enhancements	64
Summary of Project Achievements	67
Contributions to the Field	68
External Libraries, Frameworks, and Resources Used	72
External Libraries:	72
Dev Dependencies:	73
Backend Libraries:	73
Backend Dev Dependencies:	74
Resources:	74

CHAPTER 1

INTRODUCTION

Background:

Social media has become an integral part of modern life, transforming how people communicate, share information, and engage with content. Platforms like Facebook, Twitter, and Instagram have set the standard for online interactions, providing features that allow users to create posts, share images, and interact in real time. However, the rapid evolution of technology continues to demand innovative solutions that enhance user experience and offer new functionalities. The "Threads Clone" project aims to develop a robust social media web application that integrates these features with AI capabilities, real time chatting, and modern dark theme, offering users a seamless and engaging platform for social interaction.

Purpose and Scope:

The primary purpose of the "Threads Clone" project is to create a social media platform that replicates the core functionalities of popular social networks while introducing unique features powered by AI. This includes post creation, real-time chat, image attachments, and user profile management including account deactivation feature. The scope of the project encompasses both the development of the front-end and back-end components, ensuring a smooth and interactive user experience across various devices. Additionally, the project focuses on implementing secure authentication mechanisms and scalable architecture to handle user growth and interaction demands.

Problem Statement:

Despite the availability of numerous social media platforms, there remains a need for a user-friendly, secure, and feature-rich application that leverages the latest technologies to enhance user interaction and content creation. Existing platforms often suffer from limitations in customization, real-time engagement, and AI integration. "Threads Clone" addresses these gaps by providing a comprehensive solution that integrates advanced AI models for content generation, complete optimized source-code according to industry standards and real-time chat functionalities, ensuring users have access to a dynamic and interactive social media experience.

Project Goals and Objectives:

The primary goals and objectives of the "Threads Clone" project are as follows:

1. **Develop a User-Friendly Interface:** Create an intuitive and responsive design that supports seamless navigation and interaction across different devices.
2. **A complete industry standard optimized and maintainable source code.**
3. **Implement Core Social Media Features:** Enable users to create posts, attach images, like and reply to posts, and manage their profiles.
4. **Integrate AI Capabilities:** Utilize AI models for generating text replies, posts, and other interactive content to enhance user engagement.
5. **Ensure Real-Time Communication:** Develop a robust chat system that supports text and image messages, read receipts, and online status indicators.
6. **Provide Secure Authentication:** Implement secure login mechanisms using JWT and Bcrypt to protect user data and privacy.
7. **Scalability and Performance:** Design the application architecture to be scalable and capable of handling high volumes of user interactions and data.
8. **Customization and Theming:** Allow users to personalize their experience with customizable themes, including dark and light modes.

By achieving these objectives, the "Threads Clone" project aims to offer a comprehensive social media platform that meets the evolving needs of users and sets a new standard for online social interaction.

CHAPTER 2

LITERATURE REVIEW

Overview of Social Media Platforms:

Social media platforms have drastically altered the landscape of digital communication and interaction, becoming essential tools for personal and professional engagement. They facilitate the sharing of information, experiences, and media among users globally. "Threads Clone" aims to replicate and enhance the functionalities found in leading social media platforms by integrating advanced technologies and user-centric features.

Social media platforms typically offer a range of core features such as user profiles, posting and sharing content, real-time interactions, and various engagement mechanisms like comments and likes. "Threads Clone" is designed to encompass these fundamental aspects while introducing innovative elements to improve user experience and functionality.

Challenges and Limitations:

During the development of "Threads Clone," we encountered several challenges and limitations that impacted its implementation and overall success. These are discussed in detail below.

Technical Challenges:

- **Integration Complexity:** Combining multiple technologies and frameworks posed significant integration challenges. Ensuring seamless compatibility and functionality among various components required meticulous planning and troubleshooting.
- **Performance Optimization:** As the application grew in complexity, performance issues emerged, particularly when handling large datasets and complex algorithms. Optimizing the application to ensure smooth and responsive user experience was critical.
- **Real-Time Functionality:** Implementing real-time updates and communication features like live chat and notifications required robust and scalable solutions, adding complexity to the development process.

Time Constraints:

- **Limited Timeframe:** The project had a strict deadline, limiting the time available for development and testing. This necessitated prioritizing tasks and efficient time management to meet project milestones.

- **Learning Curve:** as some technologies were new to me which are used in the project, requiring additional time for learning and adaptation, which impacted the overall development timeline.

User Experience and Usability:

- **User Interface Design:** Designing an intuitive and user-friendly interface required a deep understanding of user expectations and preferences. Iterative feedback and testing were essential to refine the user interface and improve usability.
- **Cross-Browser Compatibility:** Ensuring compatibility across different web browsers and devices added complexity to the development process. Extensive testing and debugging were necessary to address compatibility issues and provide a consistent user experience.

Ethical and Legal Considerations:

- **Data Privacy and Security:** Handling sensitive user data necessitated stringent privacy and security measures to protect user information. Ensuring compliance with data protection regulations was a critical aspect of the development process.
- **Bias and Fairness:** AI models used in the application needed careful monitoring to avoid biases that could result in unfair or discriminatory outcomes. Ensuring the ethical use of AI and addressing potential biases were significant considerations.

Security and Authentication:

- **JWT (JSON Web Token):** Utilized for secure user authentication and authorization, ensuring that user data is protected and only accessible to authorized users.
- **Bcrypt:** A password hashing function used to securely store user passwords, enhancing the security of user accounts.

AI Integration:

Gemini: An AI model integrated to provide features like post generation and AI-Powered replies, enhancing user engagement with intelligent content suggestions.

Technology Stack:

Frontend Technologies:

The "Threads clone social media" web application leverages a modern and efficient frontend technology stack to deliver a seamless user experience. Key technologies and frameworks employed on the frontend include:

- **React:** Utilized for building a dynamic and responsive user interface. React's component-based architecture facilitates the development of complex and interactive UIs.
- **SaaS:** A pre-processor for CSS, essentially adding superpowers to plain CSS.
- **Chakra UI:** A modern UI component library for React, providing a set of accessible and customizable components that ensure a consistent design language and improved user experience.
- **Recoil:** Recoil is a state management library developed by Facebook specifically for React. It helps manage the application's state in a predictable and efficient manner.
- **React Icons:** A wrapper that simplifies importing icons from different libraries like Font Awesome, Material Design, and more.
- **Socket.IO:** Employed for real-time communication, enabling instant updates and interactions such as live chat and notifications

Backend Technologies:

Behind the scenes, the web application utilizes a robust backend technology stack to manage data, authentication, and various server-side operations:

- **Node.js:** A JavaScript runtime used for building the server-side of the application. Node.js provides the scalability and performance necessary for handling concurrent user requests.
- **Express.js:** A web application framework for Node.js, facilitating the development of APIs and server-side logic.
- **MongoDB:** A NoSQL database used for storing application data, offering flexibility and scalability for handling large volumes of user-generated content.

- **Mongoose:** An object data modeling (ODM) library for MongoDB and Node.js, simplifying data validation, casting, and business logic.
- **Cloudinary:** The company provides cloud-based image and video management services. It enables users to upload, store, manage, manipulate, and deliver images and video for websites and apps.

In summary, the "Threads clone" web application combines a powerful frontend stack with an array of backend technologies to deliver a feature-rich and user-friendly experience. This technical stack ensures optimal performance, security, and scalability while harnessing the capabilities of advanced AI tools to empower users in various technical endeavors.

Functional Requirements:

User Registration and Authentication:

- Users should be able to register and create an account.
- Users should be able to log in and authenticate their credentials.
- All validation checks should be performed on both client and server side immediately.
- User should be see proper error and success messages in both client and server.
- The user credentials should be saved securely in the browser so user can automatically logged-in in next visit.
- The preferred theme should be stored in the browser for personalized theming experience.

User Profiles and Account Management:

- Users should be able to update their profile information.
- Users should be able to follow and unfollow other users.
- Account management functionalities like freezing/unfreezing accounts should be available.

- User should see some other nearby users in suggested user sections.
- From suggestions user can directly follow users without visiting their profile.
- All users should visit other user profiles and copy their profile link easily with a specific copy link button having a proper success message.
- The source code should be fully optimized.
- A proper not found component should be displayed instead of a blank screen if not found case occurs for any section like posts not found or users not found.
- The source code should be industry standard and maintainable for future use.
- The account UI should be persisted even when we log out the account, prevent accidental page disappearance.
- User should share posts to other platform.
- User should copy post URL.
- User can report improper context to administrator.
- While searching, the system should search the user perform string matching to display similar user with similar names.
- Displaying sponsored posts to fresh users until they follow other users.

Real-Time Communication:

- Users should be able to send and receive text and image messages.
- Users should be able to see the online status of other users.
- Read receipts should be implemented.
- Notification tones for new messages should be implemented.

Customization and Theming:

- Users should be able to switch between dark and light themes for a personalized experience from the application header.

Non-Functional Requirements:

- **Performance:** The application should be responsive and provide quick response times for generating AI models and processing user requests.
- **Scalability:** The system should handle a growing number of users and data without significant performance degradation.
- **Security:** Robust security measures should be implemented to protect user data, including encryption and secure authentication.
- **Reliability:** The system should be reliable and available, with minimal downtime.
- **Usability:** The user interface should be intuitive and user-friendly.
- **Compatibility:** The application should be compatible with different web browsers and devices.
- **Maintainability:** The codebase should be well-structured and maintainable, allowing for future updates and enhancements.
- **Documentation:** Comprehensive documentation should be provided, including installation instructions, user guides, and API documentation.
- **Integration:** The application should integrate with external services or APIs to enhance its functionality.
- **Performance Monitoring:** The system should have monitoring capabilities to track performance metrics and optimize resource usage.

Advantages of Proposed System:

- **Enhanced Efficiency:** Automates various tasks such as post generation and real-time communication, improving overall efficiency.
- **Increased Productivity:** AI-powered tools enable users to quickly create and engage with content, saving time and effort.
- **Improved User Experience:** A user-friendly interface and intuitive design enhance user satisfaction and engagement.

- **Cost-Effective Solution:** The application offers free and premium tiers, allowing users to choose the level of service that meets their needs.
- **Scalability and Customization:** The system is designed to scale with user growth and offers customizable features to meet individual user preferences.

Reliable and Secure: Robust security measures ensure the protection of user data and provide a safe environment for interaction.

CHAPTER 3

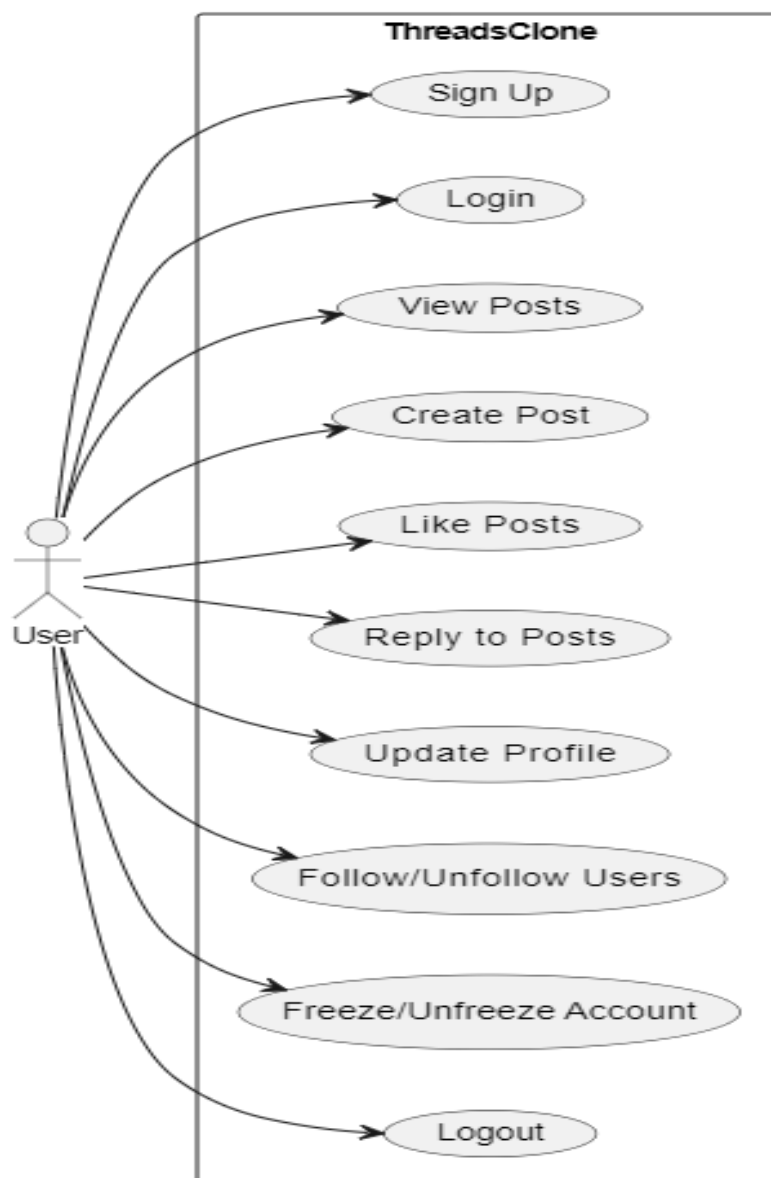
ARCHITECTURE AND DESIGN

Overview of the Architectural Design

"Threads Clone" is a comprehensive full-stack web application built using the MERN stack, which includes MongoDB, Express.js, React, and Node.js. This combination of technologies ensures a robust, scalable, and maintainable solution for developing a social media platform with integrated real-time chat capabilities and AI-driven features.

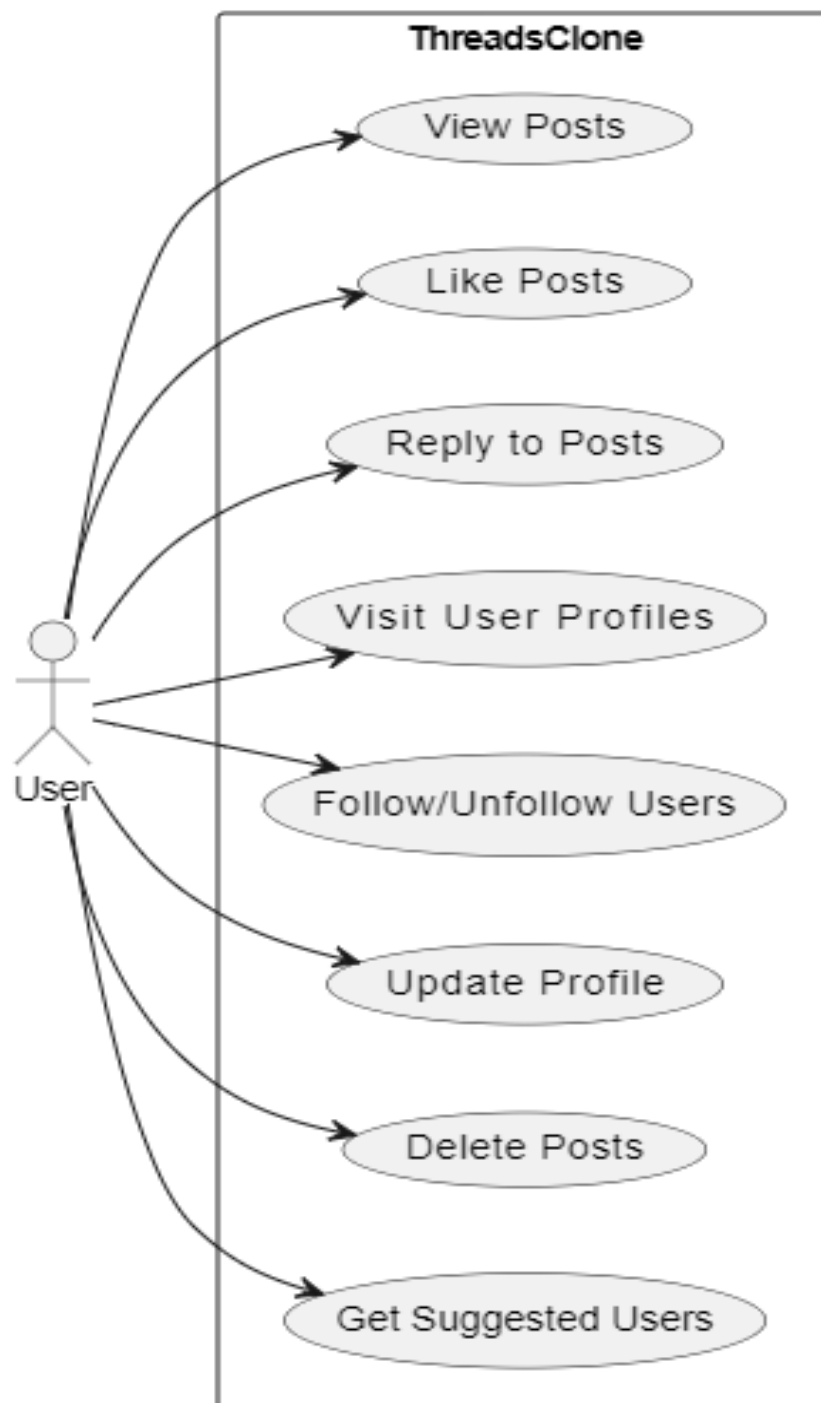
Use Case Diagram: Core Platform

The Core Platform Use Case Diagram outlines essential functionalities of Threads Clone. Users can securely sign up and log into the platform, view posts from others, create their own posts, interact by liking and replying to posts, update their profiles, manage connections through follow/unfollow actions, freeze/unfreeze their accounts, and securely log out.



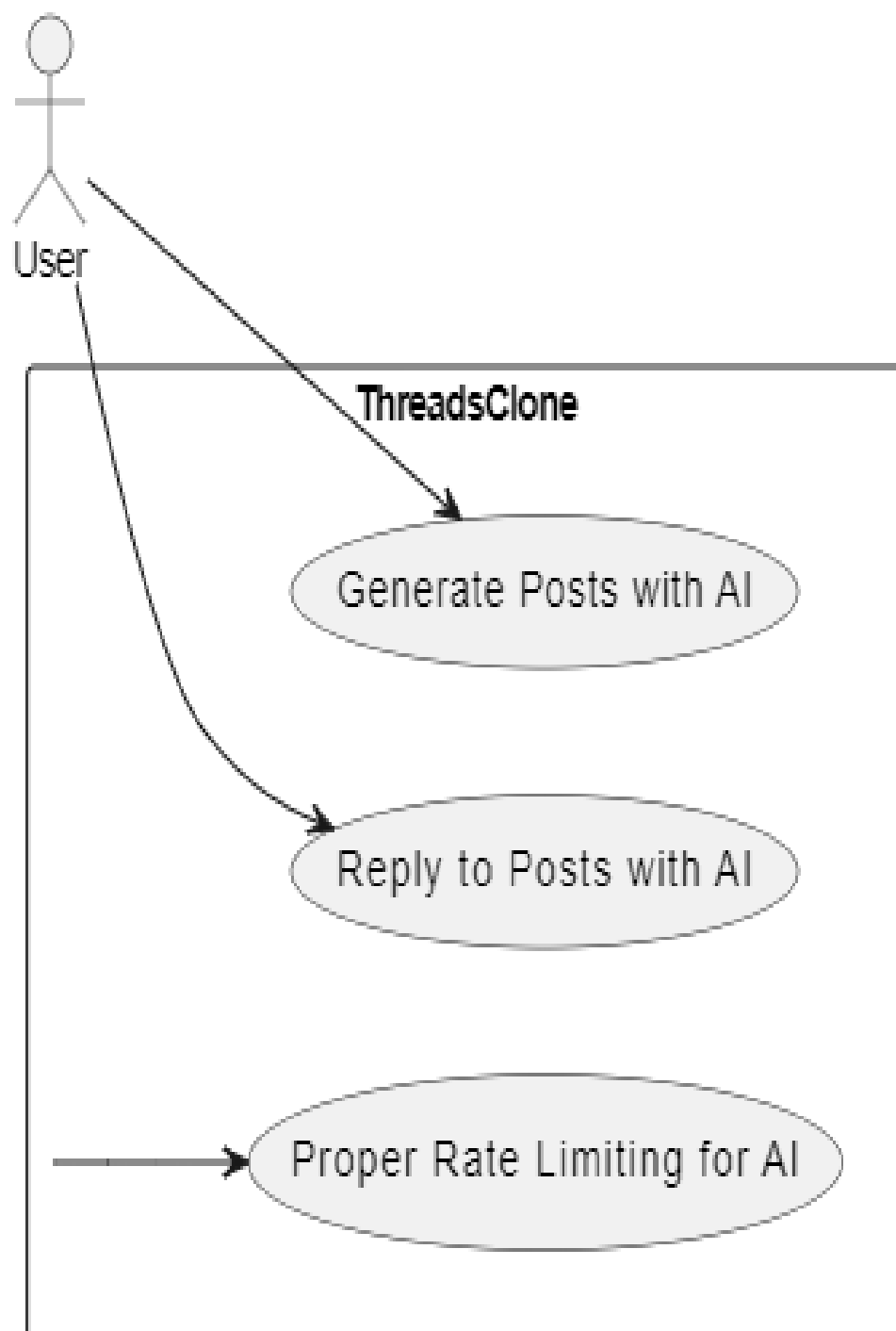
Use Case Diagram: User interaction features

In the User Interaction Use Case Diagram, users engage with Threads Clone by viewing posts from others, liking posts to express appreciation, and interacting through comments and replies. They can visit user profiles to explore more about others, manage their own profiles, delete their posts, and discover new connections by following suggested users.



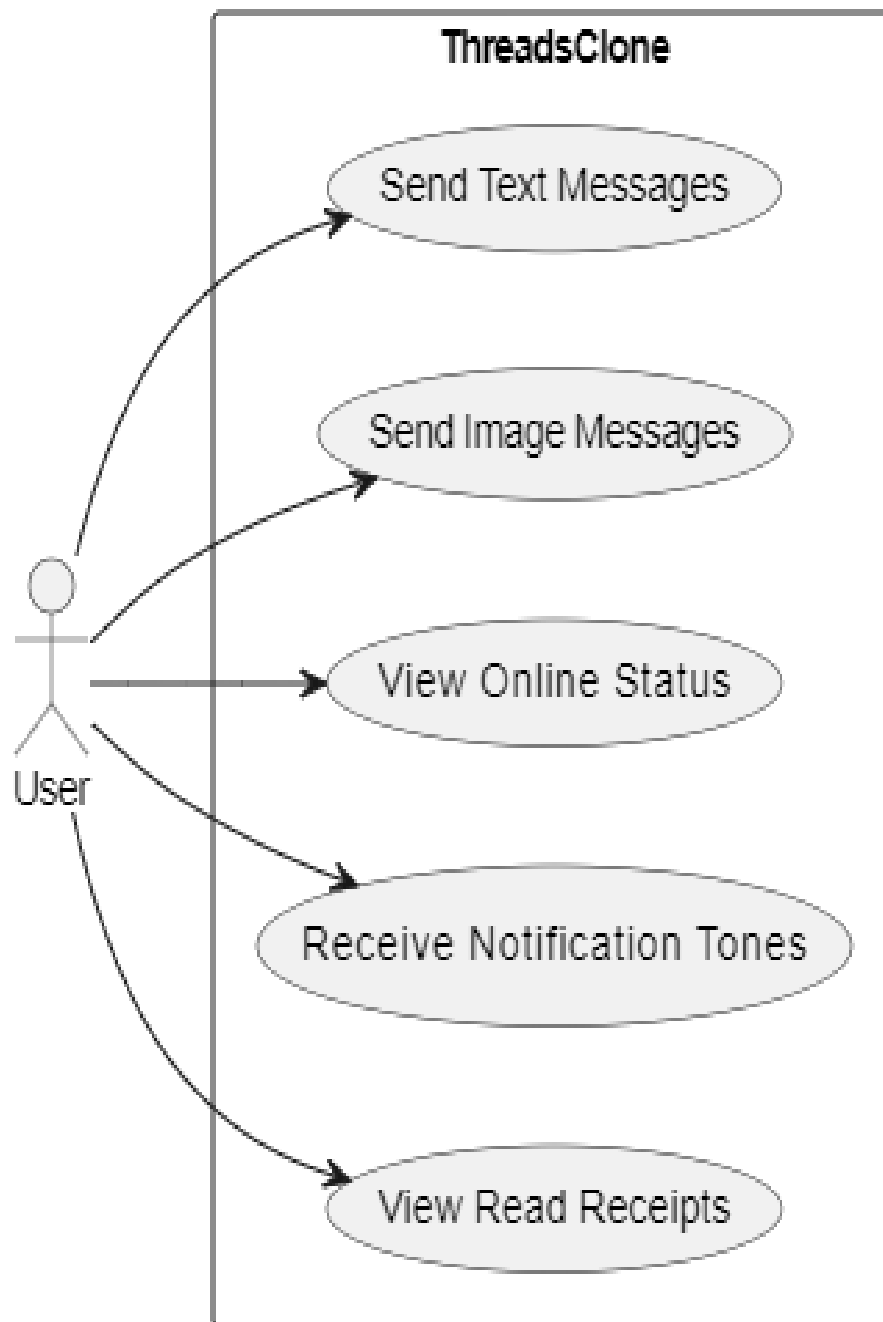
Use Case Diagram: Ai features

The AI Integration Use Case Diagram demonstrates how AI enhances user interactions in Threads Clone. Users can generate posts using AI capabilities, reply to posts with AI-generated responses, and ensure proper rate limiting to manage AI usage effectively, preventing abuse and maintaining platform stability.

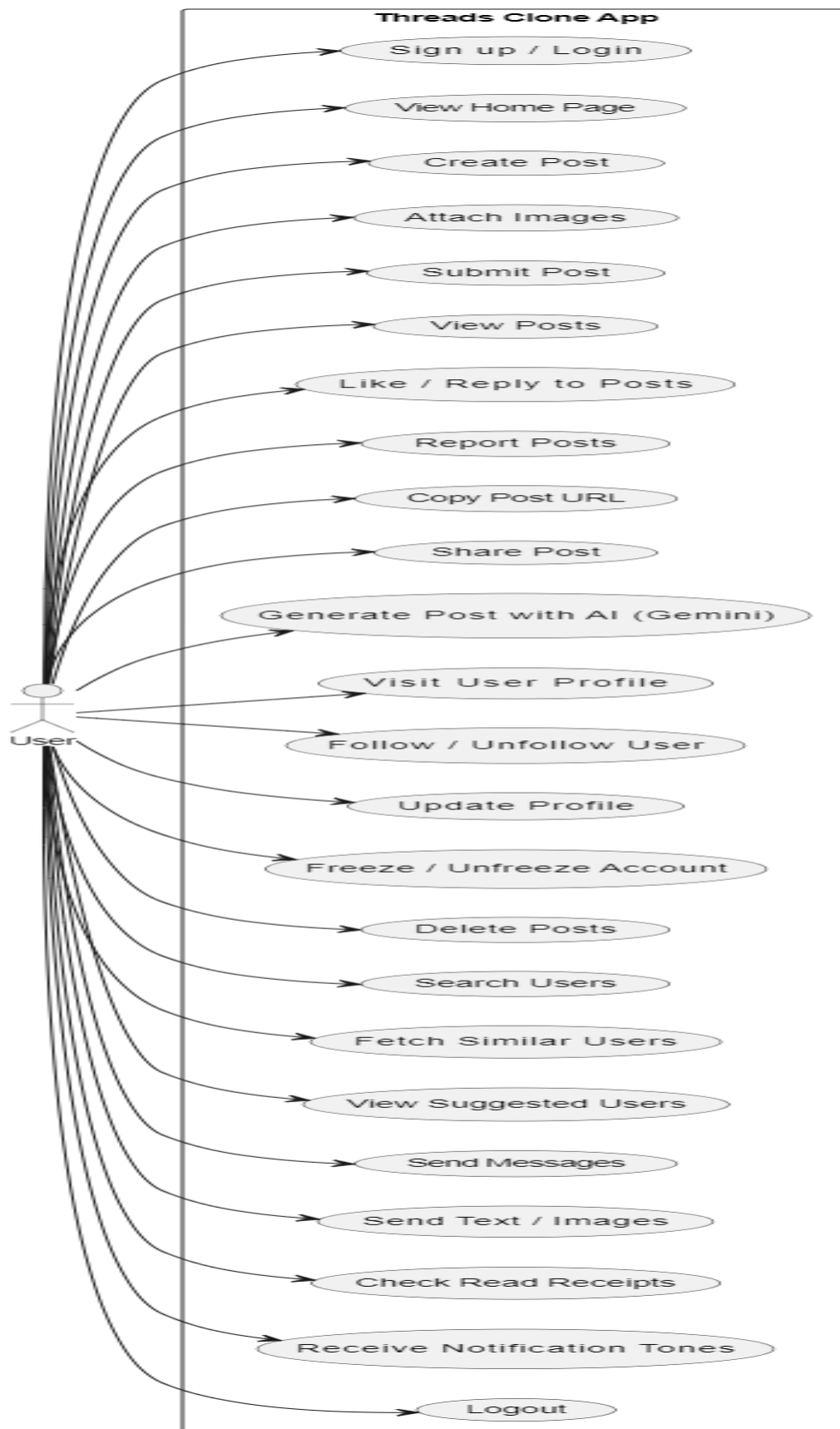


Use Case Diagram: Chat functionality

The Chat Feature Use Case Diagram illustrates how users communicate in real-time within Threads Clone. They can send text and image messages to each other, check the online status of their contacts, receive notification tones for new messages when the chat is out of focus, and verify if their messages have been seen by recipients through read receipts.

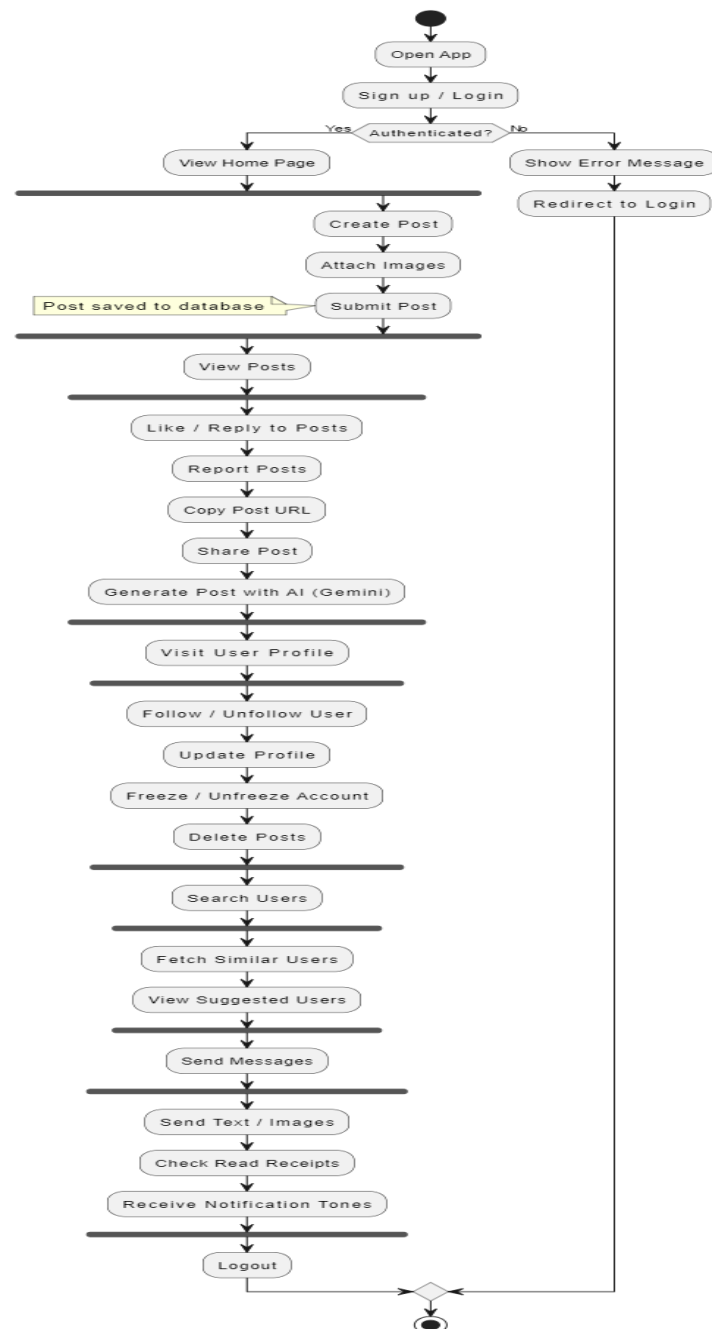


Use Case Diagram: Complete overview



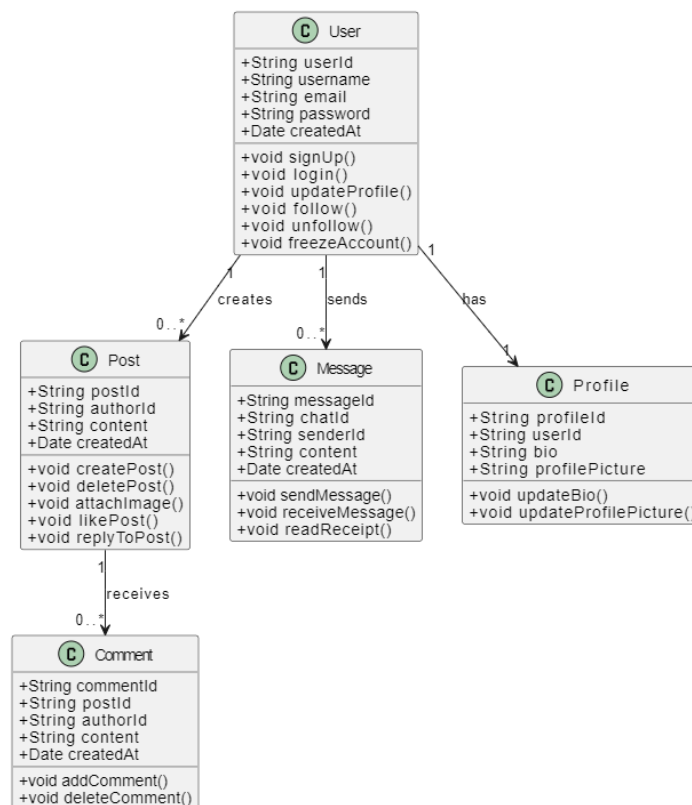
Threads-Full-Stack: Activity diagram

The User Activity Diagram for Threads Clone captures the dynamic interactions and workflows users engage in while using the platform. It illustrates users signing up and logging into the application securely, navigating through various sections to view posts from others, create their own content with images, and interacting by liking, replying, and deleting posts. Users manage their profiles by updating information, follow/unfollow other users, receive and respond to real-time chat messages, and utilize AI features to generate and reply to posts. The diagram highlights the seamless integration of features like dark/light themes, error handling with informative messages, and efficient loading with visual spinners, ensuring a user-friendly experience across different devices and interaction scenarios.



UML Diagram:

The UML Diagram for Threads Clone provides a structured overview of its architecture and interactions. It includes essential classes like User, Post, Comment, and Message, detailing their attributes such as `user_id` and `username`, along with methods for managing data. The diagram also illustrates sequence diagrams depicting user interactions such as login, post creation, and messaging flows, ensuring a clear visualization of the application's behavior. Additionally, an activity diagram outlines user workflows from signing up and logging in to navigating posts, interacting with comments, and managing profiles, facilitating efficient development and maintenance processes.



Project Structure

Backend

- `server.js`: Entry point of the backend application where Express.js is configured.
- `controllers`: Contains the business logic for various functionalities, such as user authentication, posts, and chat.
- `models`: Defines the database schemas using Mongoose for collections like users, posts, and messages.

- routes: Defines the API endpoints and maps them to controller functions.
- middleware: Middleware functions for handling requests, authentication, and error handling.
- DB: Code for the database connection used to connect to database when server starts.
- utils: Utility functions and helper modules for common functionalities.

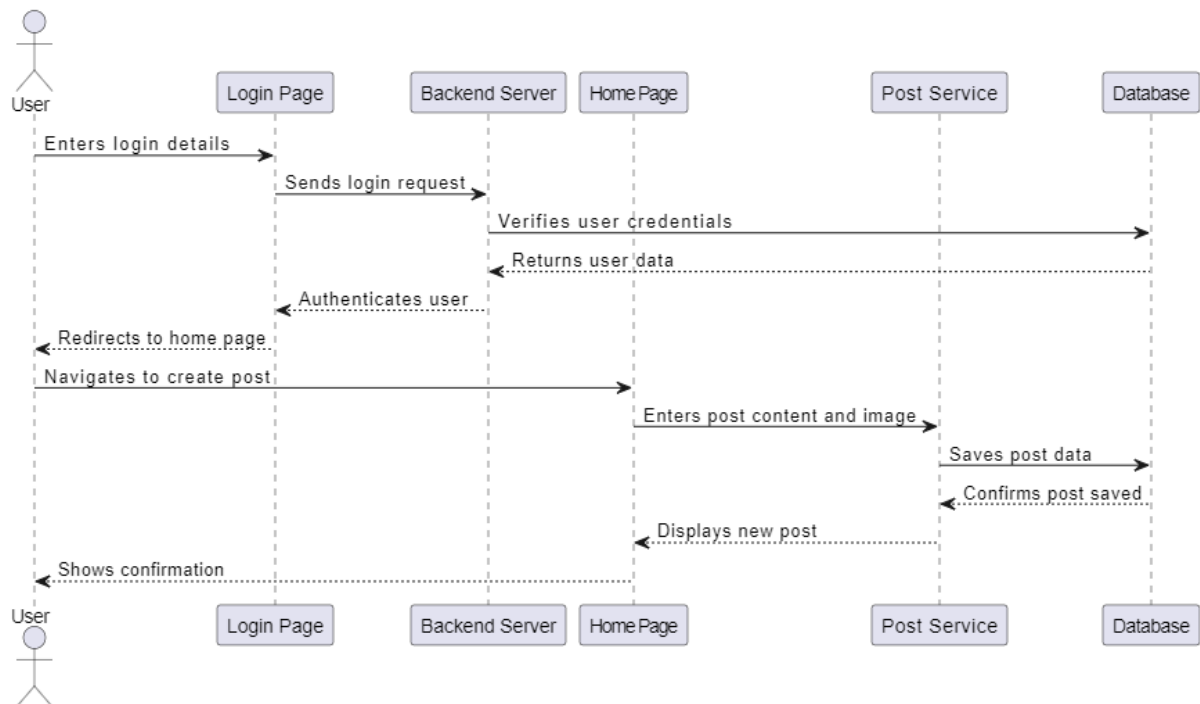
Frontend

- components: Reusable UI components such as Post, Comment, Header, and Suggested User.
- pages: Different pages of the application (e.g., Home, Profile, Chat).
- hooks: Custom React hooks to manage state and side effects.
- assets: Static assets like images and sound notifications audio clip.
- styles: Global and component-specific CSS styles.

Frontend Design

The frontend design focuses on creating an intuitive and visually appealing user interface to provide a seamless user experience while showcasing the power of AI.

Sequence Diagram:



Backend Design

Technology Stack

- Node.js: Server-side JavaScript runtime.
- Express.js: Web application framework for Node.js.
- Mongoose: ODM for MongoDB.
- Mongo DB: NOSQL database.

Architecture

- Model-View-Controller (MVC): Separates the application into three main components:
- Model: Represents the data and business logic.
- View: Handles the presentation logic (managed by the frontend).
- Controller: Handles the request/response flow and business logic.

API Endpoints

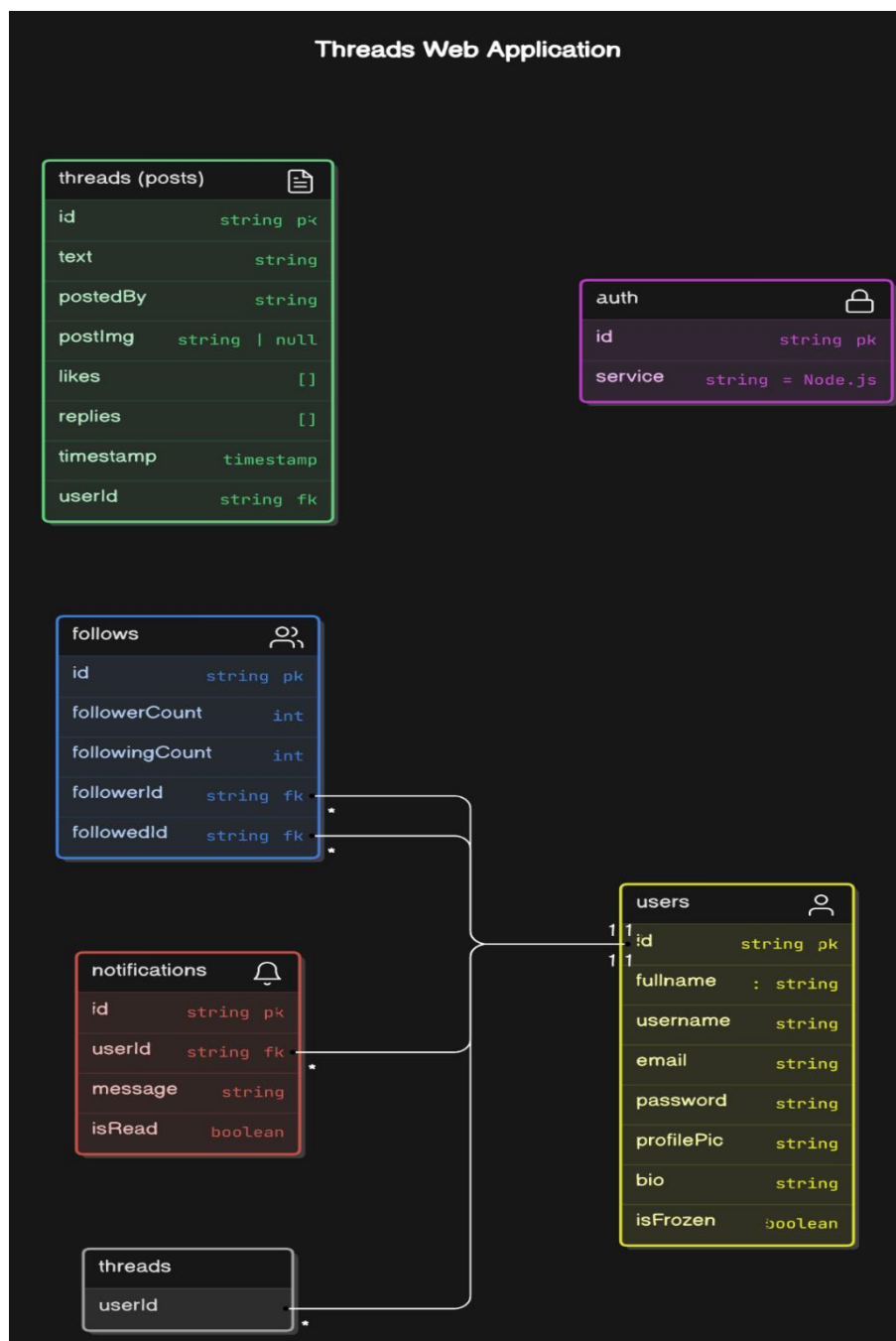
- The backend exposes various API endpoints for user authentication, posts management, chat functionality, etc.
- Endpoints are defined using Express.js routes and controllers.
- Authentication and Authorization
- User authentication and authorization are implemented using JWT (JSON Web Tokens).
- Middleware functions handle user authentication and session management.

Database Management

- MongoDB: Database management system for storing and retrieving data.
- Mongoose: Provides a schema-based solution to model data.

Database Design

The database design includes collections for users, posts, comments, and chat messages.



User Collection

- `_id`: Unique identifier for each user.
- `username`: String representing the user's name.
- `email`: String representing the user's email address.
- `password`: Hashed string representing the user's password.
- `createdAt`: Timestamp indicating when the user was created.

Post Collection

- `_id`: Unique identifier for each post.
- `authorId`: Reference to the user who created the post.
- `content`: String representing the post's content.
- `createdAt`: Timestamp indicating when the post was created.

Comment Collection

- `_id`: Unique identifier for each comment.
- `postId`: Reference to the post being commented on.
- `authorId`: Reference to the user who created the comment.
- `content`: String representing the comment's content.
- `createdAt`: Timestamp indicating when the comment was created.

Chat Message Collection

- `_id`: Unique identifier for each message.
- `chatId`: Reference to the chat thread the message belongs to.
- `senderId`: Reference to the user who sent the message.
- `content`: String representing the message content.
- `createdAt`: Timestamp indicating when the message was created.

UX/UI Design

Navigation

The navigation bar and the header component include a logo and links to different sections of the app. It is designed to be responsive and easily accessible from any page within the application.

Home Page

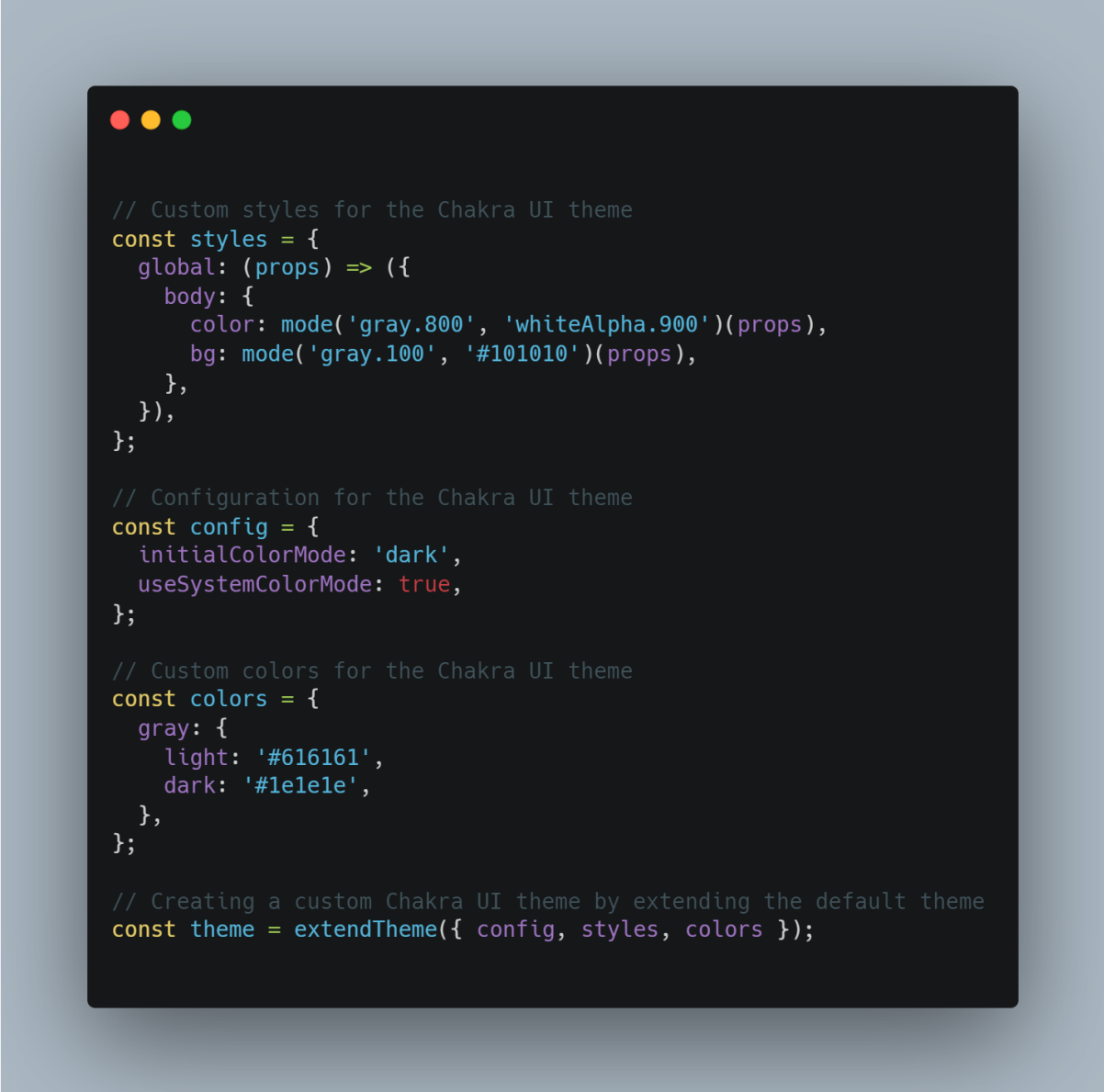
The home page features a Login and signup a new account option, it has different links that directly redirect user to different sections like login or signup.

Typography

The application combines different font sizes and weights to create visual hierarchy and improve readability. Typography is consistent throughout the app, ensuring a cohesive look and feel.

Color Scheme

Current project has Two color schemes one for light mode and second d for dark lovers. Both themes are configured in root of the client-side app in react project.



```
// Custom styles for the Chakra UI theme
const styles = {
  global: (props) => ({
    body: {
      color: mode('gray.800', 'whiteAlpha.900')(props),
      bg: mode('gray.100', '#101010')(props),
    },
  }),
};

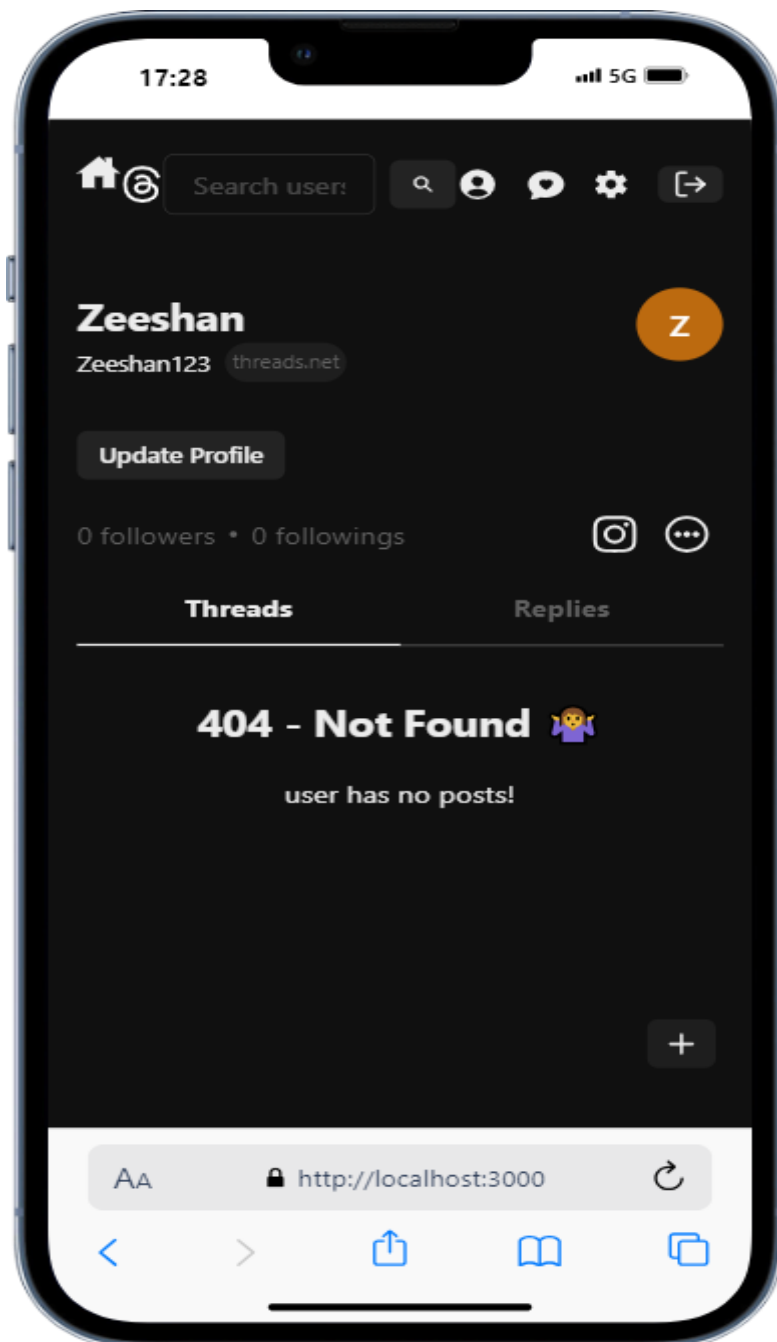
// Configuration for the Chakra UI theme
const config = {
  initialColorMode: 'dark',
  useSystemColorMode: true,
};

// Custom colors for the Chakra UI theme
const colors = {
  gray: {
    light: '#616161',
    dark: '#1e1e1e',
  },
};

// Creating a custom Chakra UI theme by extending the default theme
const theme = extendTheme({ config, styles, colors });
```


Responsive Design

The application ensures a consistent and optimal user experience across different devices by adjusting typography and layout accordingly. Media queries and flexible grid layouts are used to adapt the UI to various screen sizes and orientations.



User Interactions

Call-to-action buttons and interactive elements are prominently placed and visually distinct to guide user actions. Hover effects and animations enhance the user experience by providing feedback and improving engagement.

Accessibility

The application is designed with accessibility in mind, ensuring that all users, including those with disabilities, can navigate and interact with the platform. ARIA (Accessible Rich Internet Applications) attributes and semantic HTML are used to improve screen reader support.

CHAPTER 4

IMPLEMENTATION

Frontend Development:

Frontend development focuses on building the user interface and experience of the application. Using the MERN stack, particularly React.js, this involves structuring, designing, and coding the visual elements and interactivity that users interact with.

Project Setup:

First, you need to have Node.js installed on this machine. You can download it from the official Node.js website.

Once you have Node.js installed, open this terminal or command prompt and navigate to the directory where you want to create this project.

To create a new Next.js project, run the following command in this terminal:

```
npm create vite@latest
```

This command will create a new react project with default boilerplate.

After the project is created, navigate into the project directory using the following command:

```
cd project-name
```

Now, you can start the development server by running the following command:

```
npm run dev
```

This will start the Next.js development server and you can access this project in this browser at <http://localhost:3000>.

At this point, you have a basic React.js project set up. You can start building this frontend components and pages. Feel free to use any file folder structure because it is a library not a framework.

To add additional dependencies to this project, you can use the npm install command followed by the package name. For example, if you want to install a UI library like Chakra-UI, you can run:

```
npm i @chakra-ui/react @emotion/react @emotion/styled framer-motion
```

Component Development:

Install required libraries:

We can easily install different libraries that help us to make our development faster and more scalable.

Accessing Components:

All the components can be found in the component's directory in the client directory. The components are reusable pieces of code in the project.

Customizing Components:

We can easily personalize various components provided my chakra UI according to our project needs.

Creating Custom Components:

We can also build the custom components to make the code reusable and maintainable. This project also has a bunch of custom components like Actions, post and comment.

Styling Components:

I added different props to different components provided my chakra Ui to meet the project needs.

State Management:

In this project, state management is an important aspect to consider managing and share data across different components of the application. State management allows us to keep track of the application's state and make it accessible to different parts of the application.

There are several state management libraries and patterns available in the React ecosystem, such as Redux, MobX, and Context API. In this project, the Recoil library is used for state management.

Recoil is a state management library developed by Facebook that provides a simple and efficient way to manage state in React applications. It leverages React's built-in context API and hooks to provide a predictable and efficient way to manage and share state.

Let's take a closer look at how state management is implemented in this SaaS project using Recoil:

State Atoms:

Atoms are the smallest units of state in Recoil. They represent individual pieces of state that can be read from and written to. In this project, atoms are used to define and store the application's state. For example, there might be an atom called user Atom that stores information about the currently logged-in user.

Selectors:

Selectors are derived state values that are computed based on one or more atoms. They allow us to derive new state values from existing state values. Selectors can be used to perform calculations, filtering, or any other data manipulation. For example, there might be a selector called filteredDataSelector that filters a list of data based on certain criteria.

Recoil Root:

The Recoil Root component is used to provide the Recoil state to the entire application. It wraps the root component of the application and makes the Recoil state accessible to all the components within the application.

useRecoilState and useRecoilValue:

These are hooks provided by Recoil that allow components to read and write to atoms and selectors. The useRecoilState hook returns a tuple containing the current value of an atom and a setter function to update the value. The useRecoilValue hook returns the current value of an atom or selector.

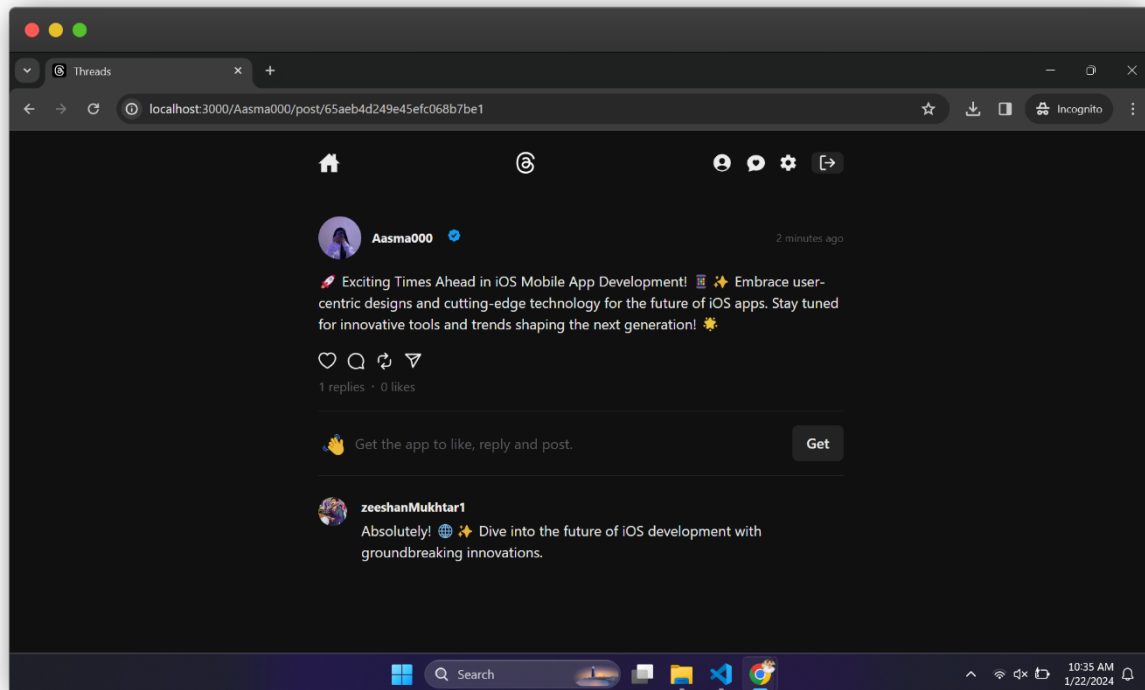
By using Recoil for state management, the application can easily share and update state across different components. Components can subscribe to changes in state and automatically re-render when the state changes. This helps in keeping the application's UI in sync with the underlying data.

Building the Home Page:

In this web app, I started by explaining how to build the landing page of a social media web app using MERN tech stack. Here are the steps I follow:

- Create a new file called Home Page .jsx inside the pages folder.
- Inside the HomePage.jsx file, create a functional component called Homepage.
- Add the necessary imports for React and any other components or styles you may need.
- Define the JSX structure of the home page, including any headings, images, buttons, or other elements you want to include.
- Export the Homepage component as the default export of the file.
- Import the home page in the main app where the entire app will be rendered for example main. Jsx.
- Save the changes and run the application to see the landing page.

Also, some pages like update profile page are protected, meaning that only the logged in users can access the and update their own profile information.

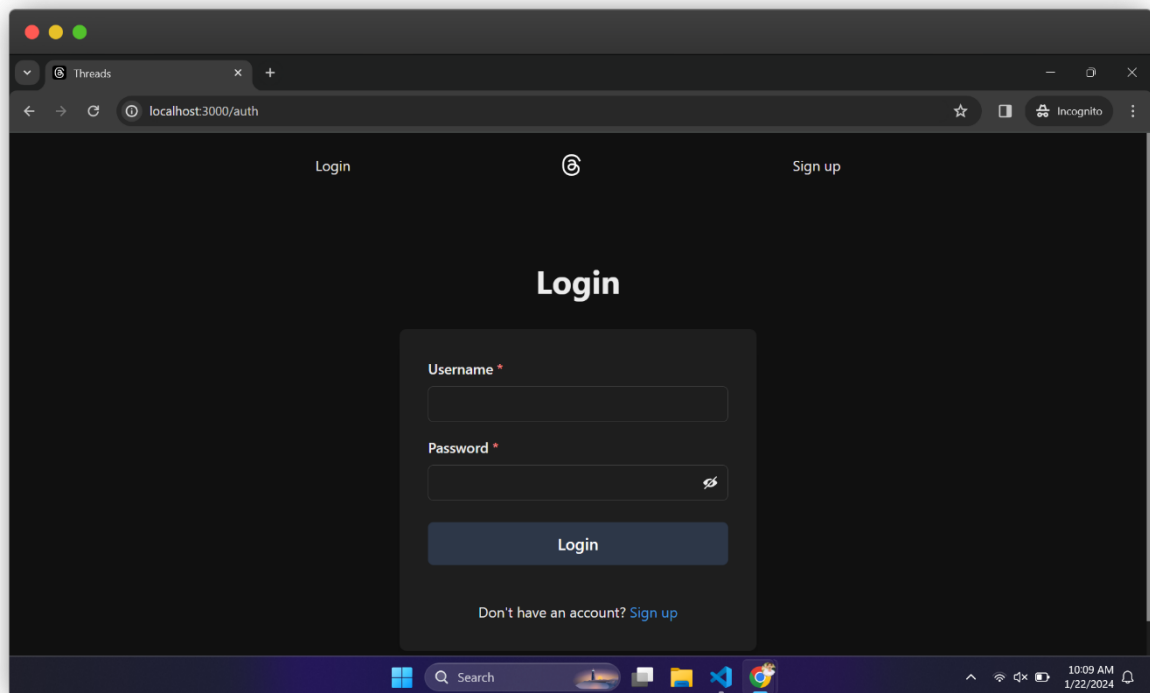


Overall, the process involves creating a new file for the home page, defining its structure using JSX, and then integrating it into the application by importing and using the component in the main index. Jsx file or main. Jsx.

Design and Layout:

Auth Page:

The Auth page play a key role in the client-side web application that enables users to gain unauthorized access to the private pages like update profile and settings page.



```
// Component for handling authentication page
const AuthPage = () => {
  // Accessing the authentication screen state using Recoil
  const authScreenState = useRecoilValue(authScreenAtom);

  return (
    // Conditional rendering based on the authentication screen state
    <{authScreenState === 'login' ? <LoginCard /> : <SignupCard />}</>
  );
};

// Exporting the AuthPage component as the default export
export default AuthPage;
```

Home page design:

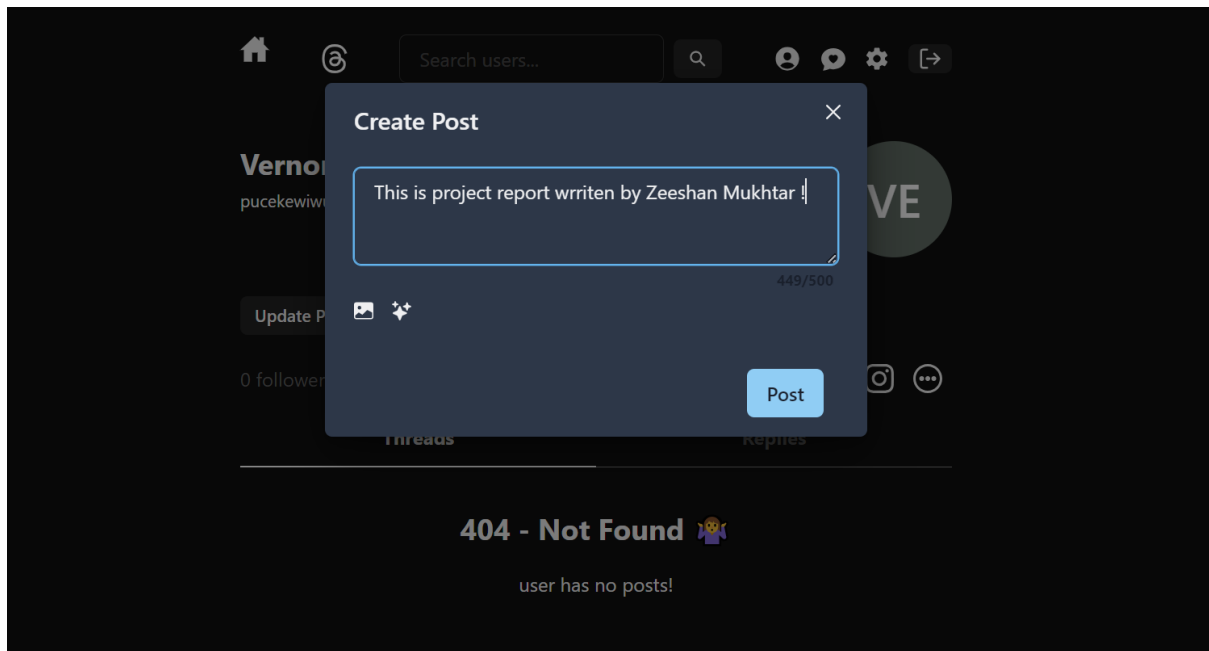
The home page design can show all the posts of those people which u followed, for error handling instead of a blank screen this page also handles the case when u don't yet follow. The home page design can show all the posts of those people which u followed, for error handling instead of a blank screen this page also handles the case when u don't yet follow anyone yet, in this case this page informs the user by saying "Your feed is waiting for you! Follow users to see their latest posts!".



Post page:

On this page use have an option to create a new post, the post may have different media like:

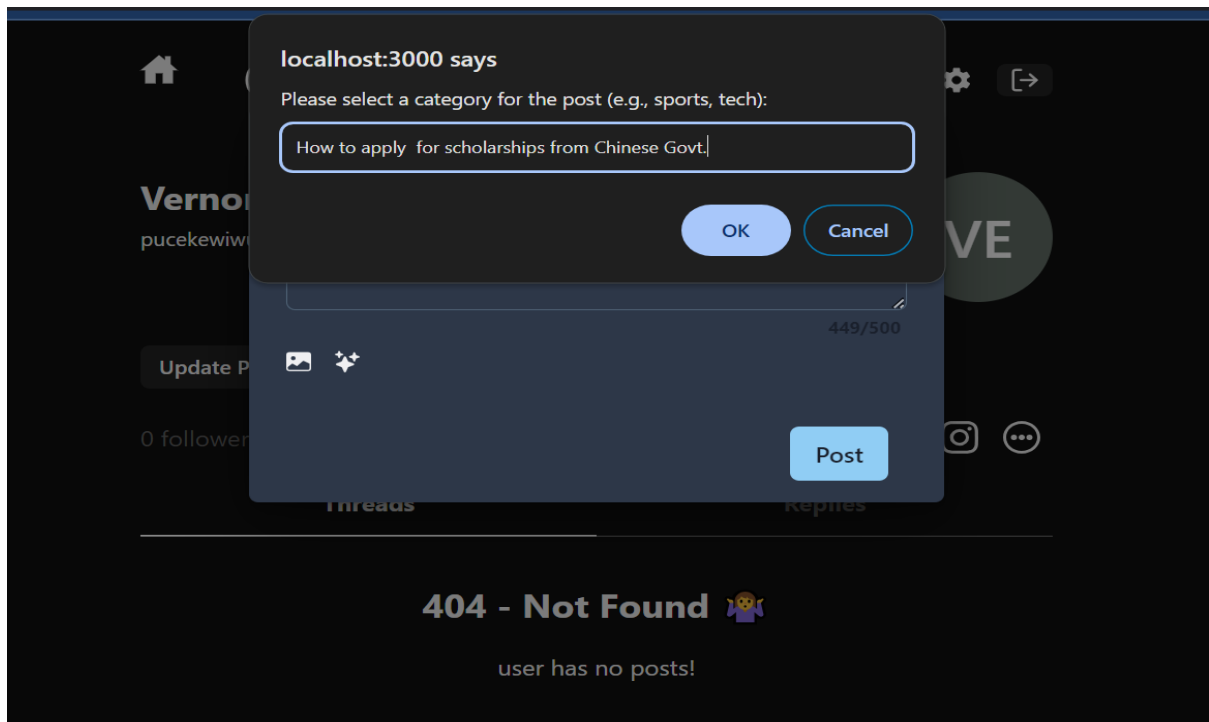
- Text post
- Image post



AI Post creation feature:

When user click on “sparkles” icon the app asks the user to confirm the post topic to generate an AI powered post.

We are using “Gemini API for fetching posts, also we have configured “Open-Ai” Api but currently it is limited to quota. So far, we have two options already configured in the code part to generate post from Ai.

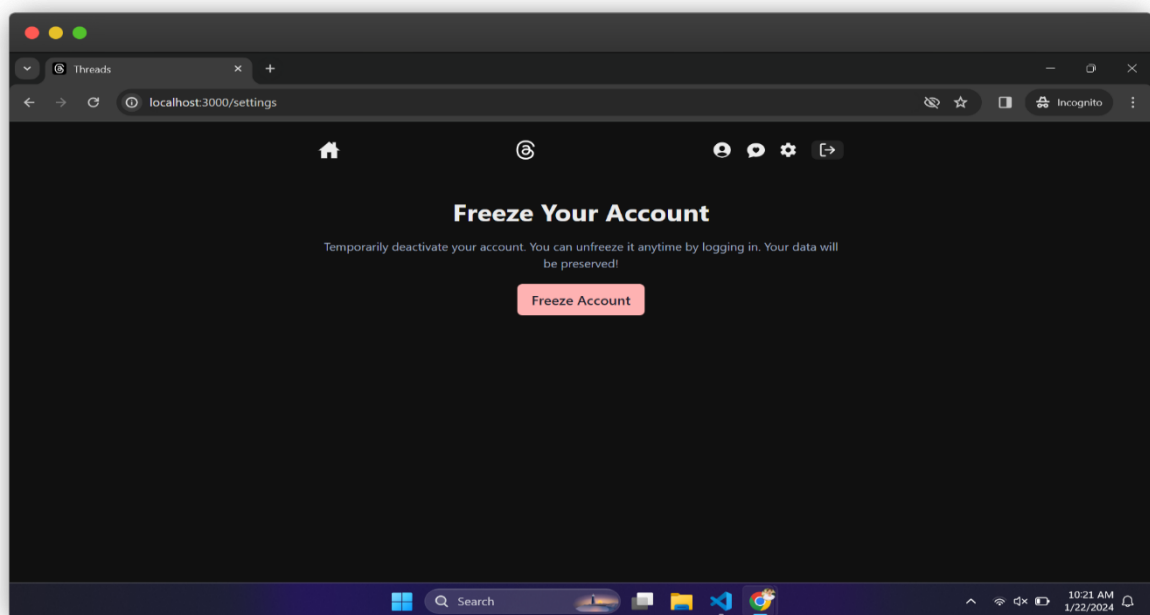


Settings Pages:

The project includes following:

Freeze account feature

- A proper confirmation dialogue to prevent accidental account suspension
- A comprehensive message on what happens on account froze.
- Logout account feature



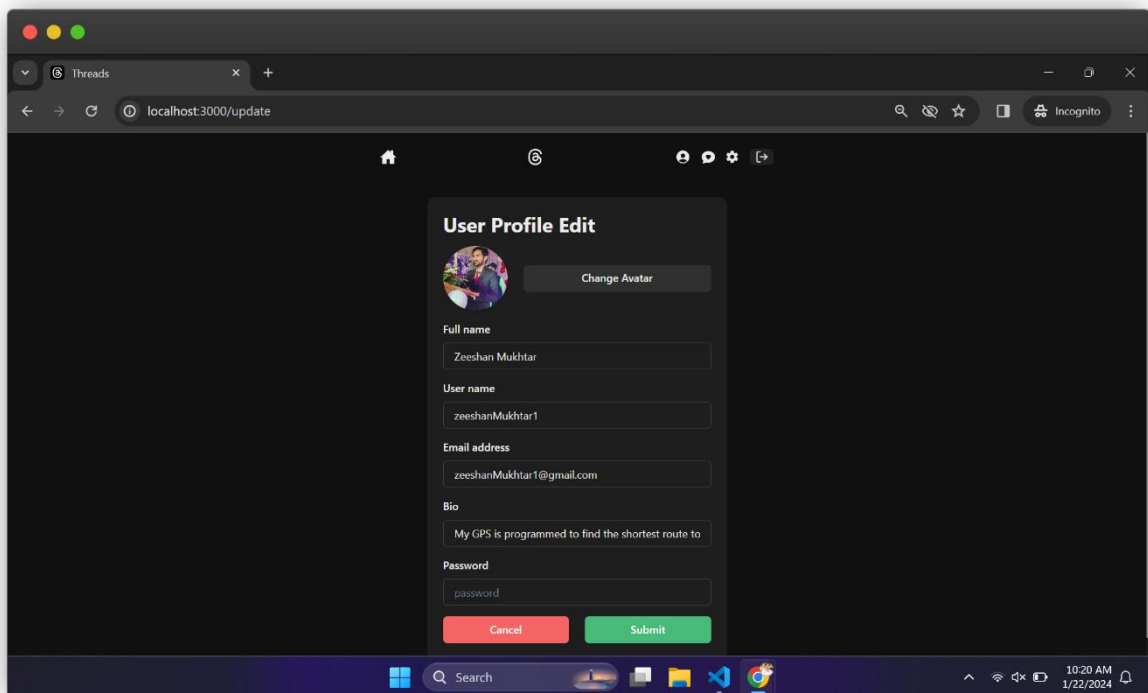
Update profile page:

The update profile page had the following abilities:

- User can change profile Avatar
- Update username
- Update full name
- Update password
- Update email
- Update optional Biography

Security Concern:

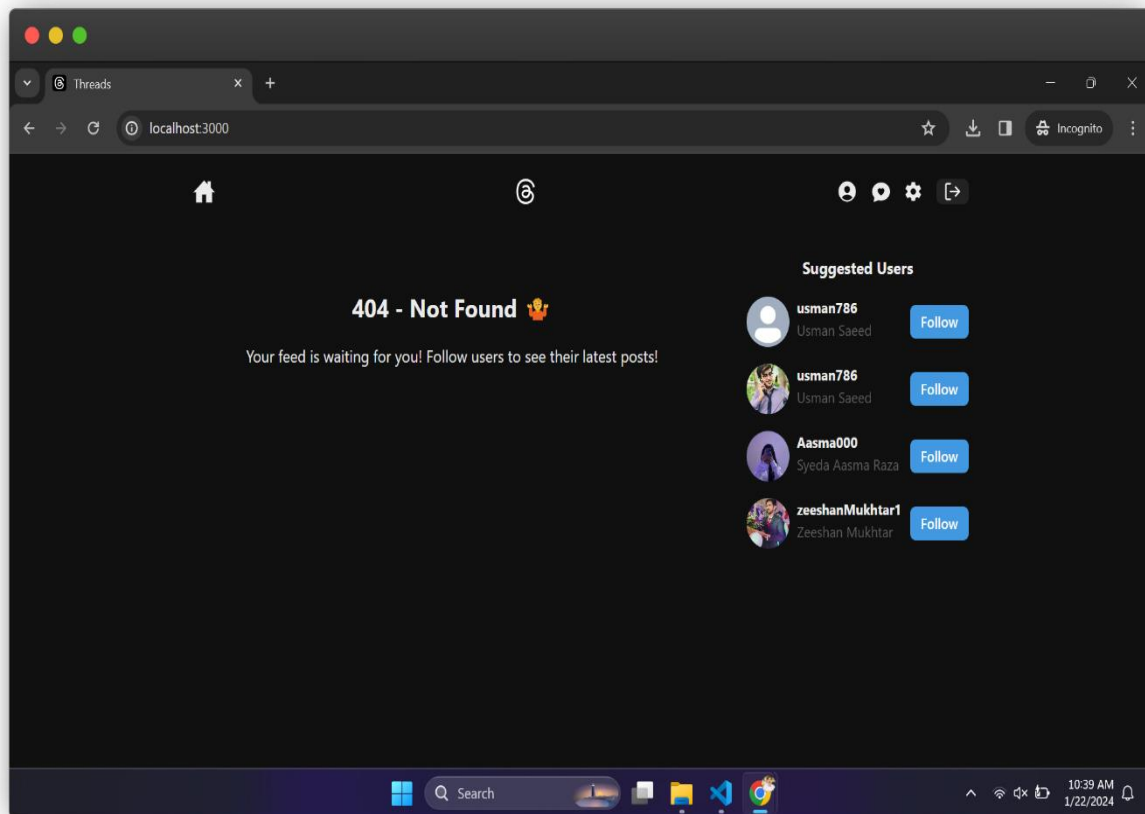
By default, the password fields will be empty to prevent unauthorized password reveal.



Suggested user:

The suggested user page had the following abilities:

- Logged in user can directly follow the use from suggestions
- Logged in suer can directly unfollow the user from suggestions
- Logged in user can visit the profile of other users from suggestions
- Logged in user can copy the profile link after visiting user from suggestions.



Custom hooks:

Current project uses various types of hooks like:

- Default hooks provided by react itself like useState and useEffect
- The chakra Ui provided hooks like switching to dark theme
- My custom hooks for displaying user custom error and success messages.

Custom Hooks list:

- Follow / Unfollow hook
- Get user profile hook
- User logout hook
- Preview image hook
- Toast message hook (most used hook here)



```
// Importing the useToast hook from Chakra UI
import { useToast } from '@chakra-ui/toast';

// Importing the useCallback hook from React
import { useCallback } from 'react';

// Custom hook for displaying toasts using Chakra UI
const useShowToast = () => {
  // Initializing the useToast hook
  const toast = useToast();

  // Function to show a toast with the specified title, description, and status
  const showToast = useCallback(
    (title, description, status) => {
      // Calling the toast function with the specified parameters
      toast({
        title,
        description,
        status,
        duration: 3000,
        isClosable: true,
      });
    },
    [toast] // Dependency array to ensure stable reference to the toast function
  );

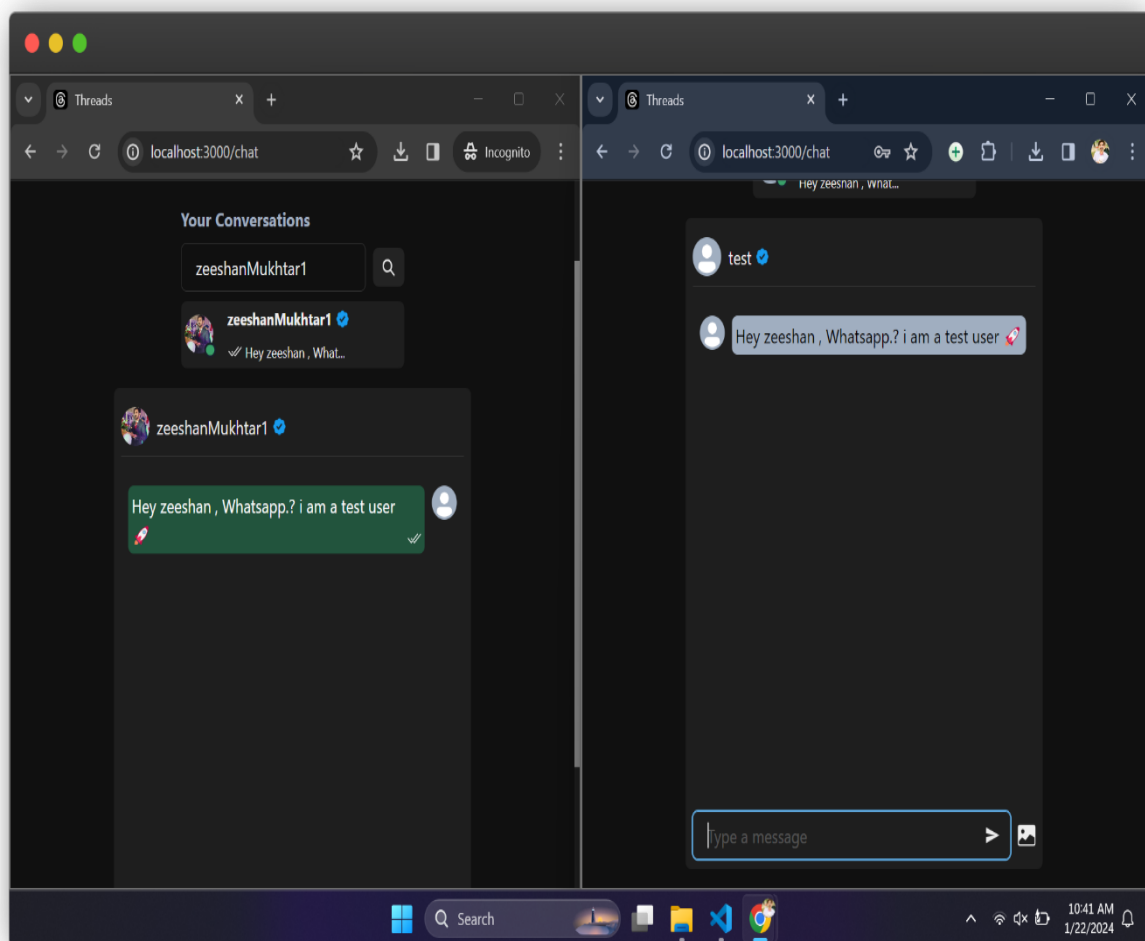
  // Returning the showToast function for component usage
  return showToast;
};

// Exporting the custom hook for use in other components
export default useShowToast;
```

Real Time chat functionality:

This route has the powerful section and have following capabilities.

- User can chat with other users
- User can search other users.
- User can see proper error messages when searched user can't be found
- User cannot search and chat with their self, if he tries to do so he will be informed by a toast info message.
- User will get a notification tone on new incoming message if he is away from the chat but logged in.



Backend Development:

Backend Development

Backend development involves creating and managing the server-side logic, database, and application programming interface (API) endpoints that power the application. For this project, we are using the MERN stack, specifically Node.js and Express.js for the server, and MongoDB for the database.

Project Setup

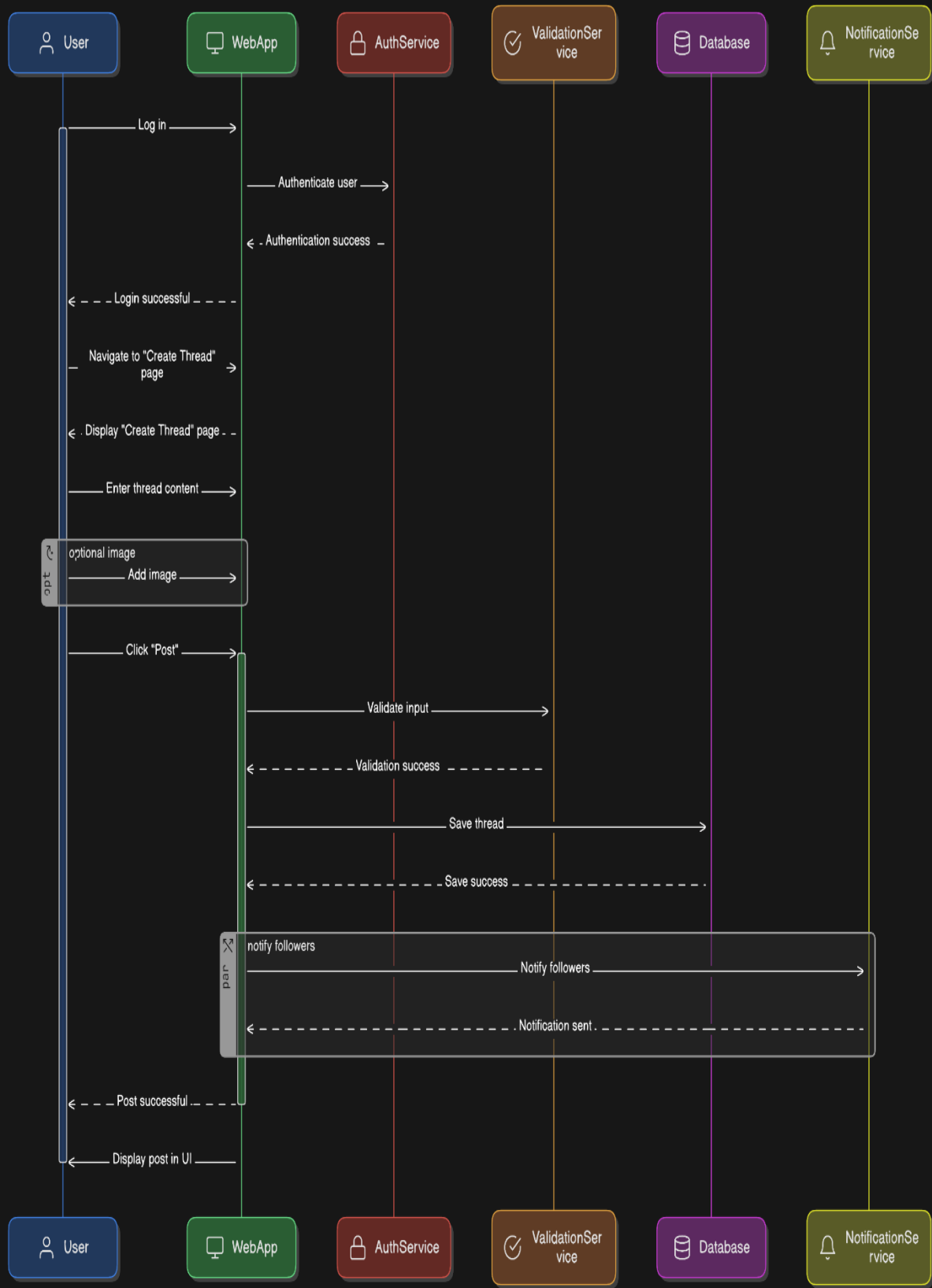
1. Initialize the Backend Project:

- Ensure Node.js is installed on your machine. You can download it from the official Node.js website.
- Open the terminal or command prompt and navigate to the directory where you want to create the backend project.
- Initialize a new Node.js project by running the following command:

```
npm init -y
```

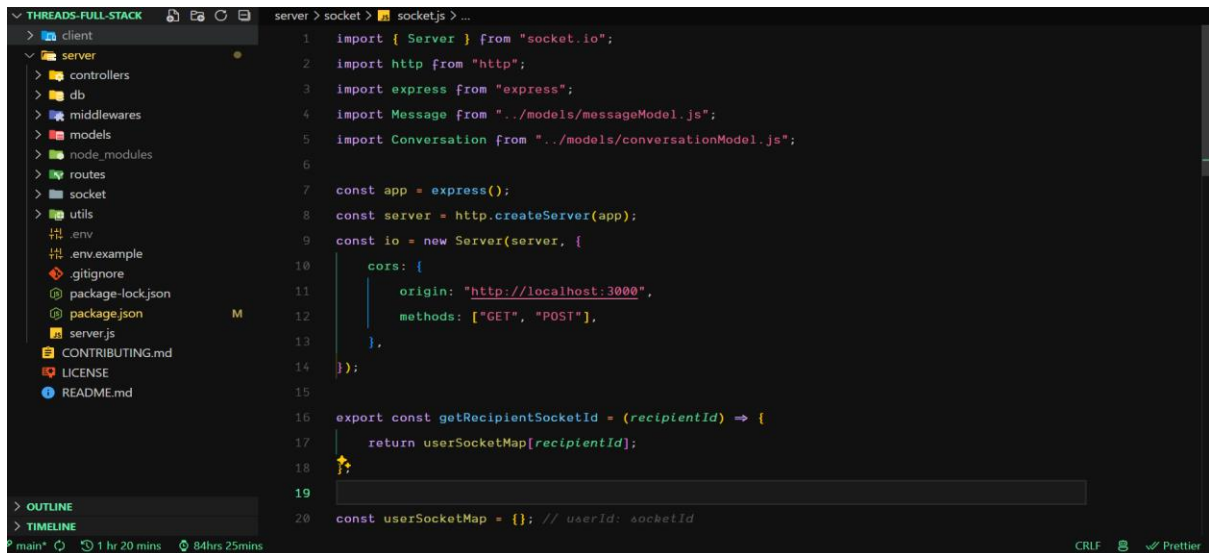
Install necessary dependencies such as Express, Mongoose, and other required libraries:
`npm install express mongoose cors dotenv bcryptjs jsonwebtoken cloudinary socket.io`

User Posting a Thread in "Threads"



2. Project Structure:

Following is my project folder structure for better organization:



3. Configuration:

Create a `.env` file in the root directory to store environment variables:



4. Starting the Server:

Create a `server.js` file to set up the Express server and connect to MongoDB:

```
import express from "express";
import dotenv from "dotenv";
import connectDB from "../db/connectDB.js";
import cookieParser from "cookie-parser";
import userRoutes from "../routes/userRoutes.js";
import postRoutes from "../routes/postRoutes.js";
import messageRoutes from "../routes/messageRoutes.js";
import { v2 as cloudinary } from "cloudinary";
import { app, server } from "../socket/socket.js";

dotenv.config();

connectDB();

const PORT = process.env.PORT || 5000;

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

// Middlewares
app.use(express.json({ limit: "50mb" })); // To parse JSON data in the req.body
app.use(express.urlencoded({ extended: true })); // To parse form data in the
app.use(cookieParser());

// Routes
app.use("/api/users", userRoutes);
app.use("/api/posts", postRoutes);
app.use("/api/messages", messageRoutes);

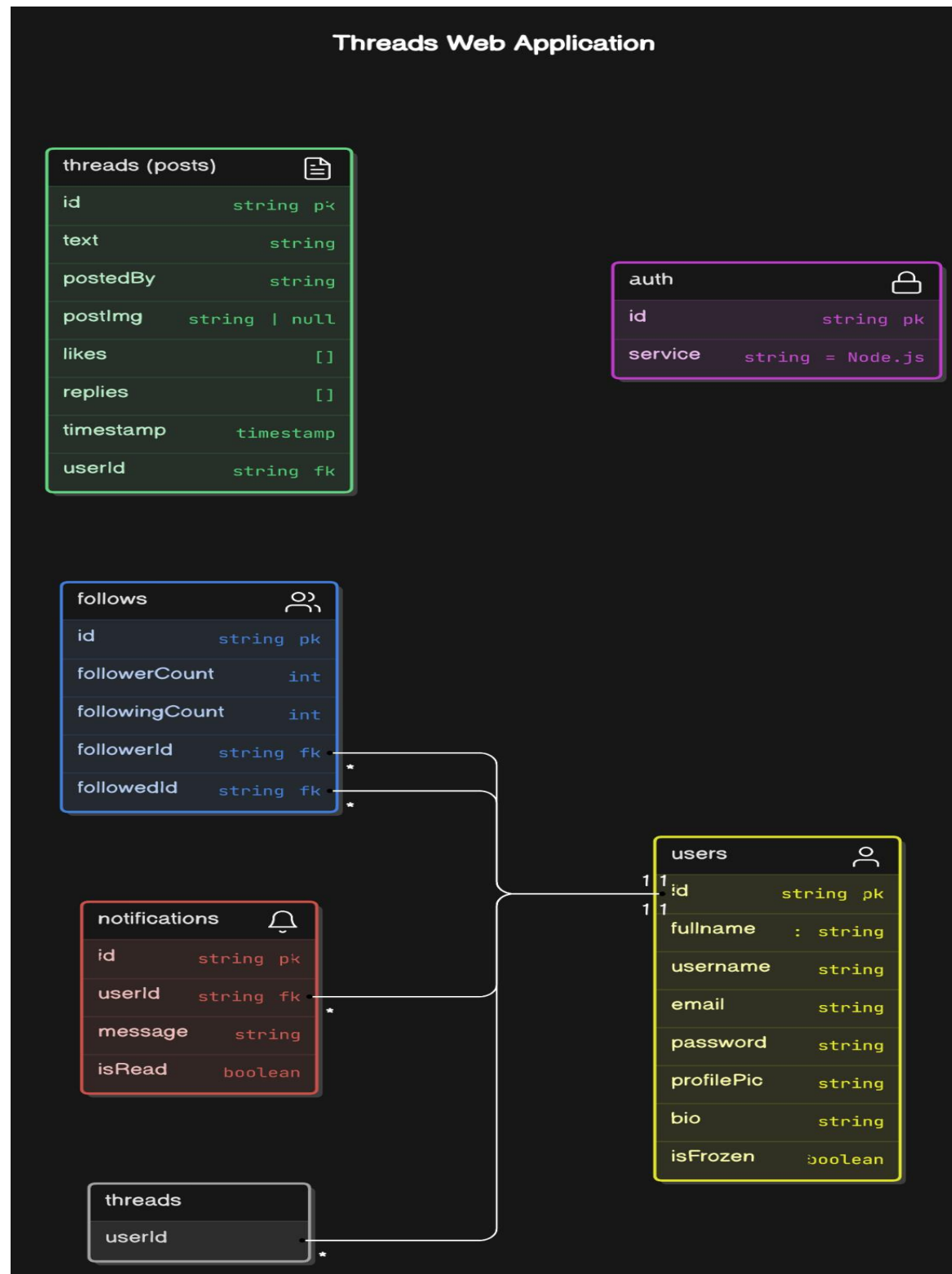
server.listen(PORT, () => console.log(`Server started at http://localhost:${PORT}`));
```

Database Models

Since current project has a full stack, we app so we have a database to store stuff, that's why there are some databases schemas implemented in the project like:

- User Model
- Post model
- Message model
- Conversation model

The user model has the following scheme to follow and vice versa for others.



localhost:27017 > My-Threads-Db > users

Documents 7 Aggregations Schema Indexes 3 Validation

Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find </> Options

ADD DATA EXPORT DATA UPDATE DELETE 1 - 7 of 7

	ObjectId	name String	username String	email String	password	
1	ectId('662522effa0fc5c...	"one"	"oneone1"	"one@gmail.com"	"\$2a\$10\$Dl	
2	ectId('66252309fa0fc5c...	"two"	"twotwo2"	"two@gmail.com"	"\$2a\$10\$Ml	
3	ectId('6625435c6c309f2...	"three"	"threethree3"	"three@gmail.com"	"\$2a\$10\$vl	
4	ectId('662770891357f5b...	"Zeeshan "	"qwhajajajaj7"	"zeshanmukhtar878@gmail.c...	"\$2a\$10\$P"	
5	ectId('6655c3338a9336d...	"Vernon Edwards"	"pucekewiwu"	"dumehe@mailinator.com"	"\$2a\$10\$SH	
6	ectId('66643a9d4e67f1a...	"Lee Watson"	"nujedas1"	"pufowynop@mailinator.com"	"\$2a\$10\$9v	
7	ectId('66644e83996b196...	"Zeeshan"	"Zeeshan123"	"zeshanmukhtar8708@gmail...	"\$2a\$10\$T!	

Middleware

1. Authentication Middleware:

Create a `middleware/ protectRoute.js` file to handle user authentication and protect routes:

```
import User from "../models/userModel.js";
import jwt from "jsonwebtoken";

const protectRoute = async (req, res, next) => {
  try {
    const token = req.cookies.jwt;

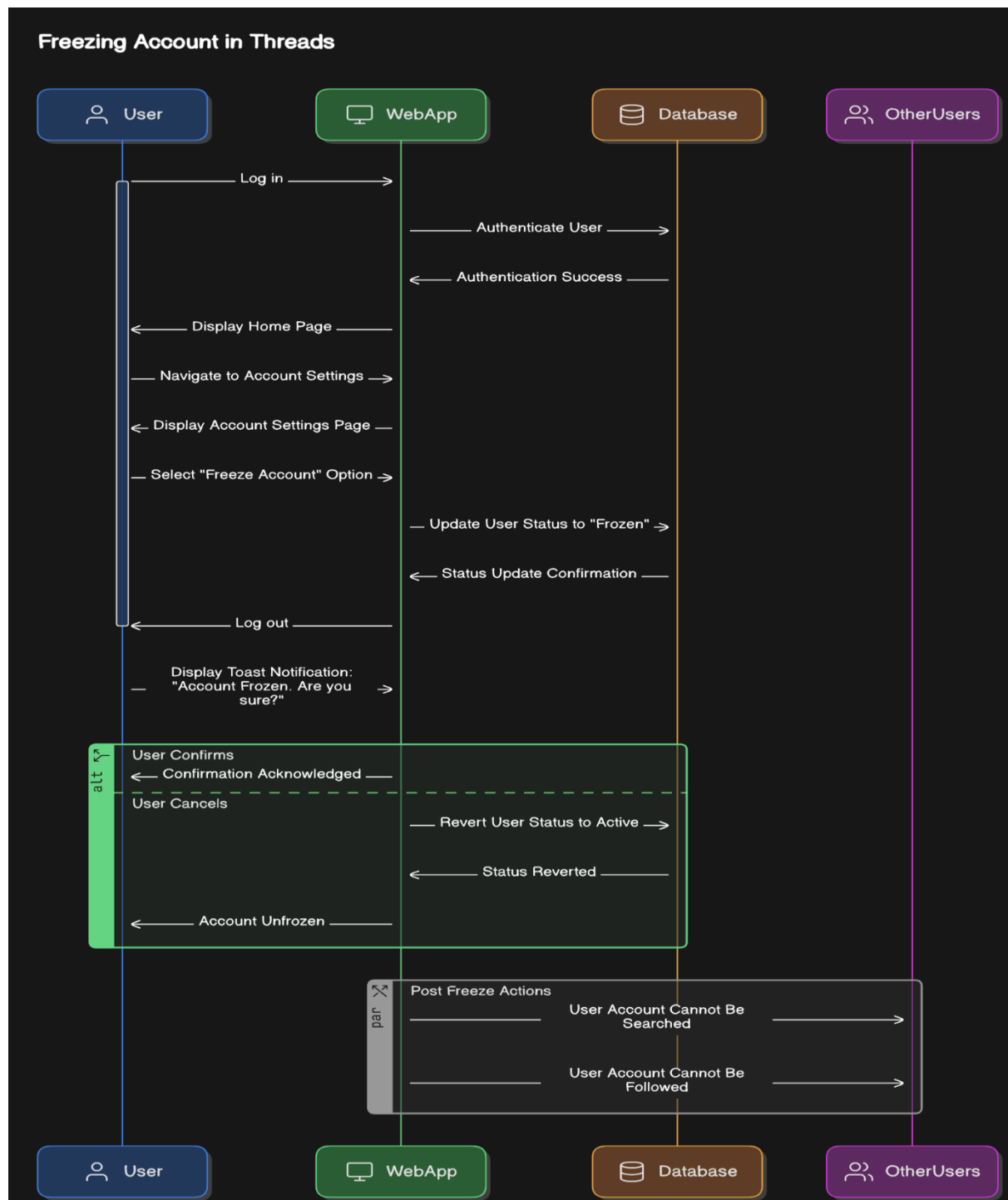
    if (!token) return res.status(401).json({ message: "Unauthorized" });

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    const user = await User.findById(decoded.userId).select("-password");
    req.user = user;

    next();
  } catch (err) {
    res.status(500).json({ message: err.message });
    console.log("Error in signupUser: ", err.message);
  }
};

export default protectRoute;
```



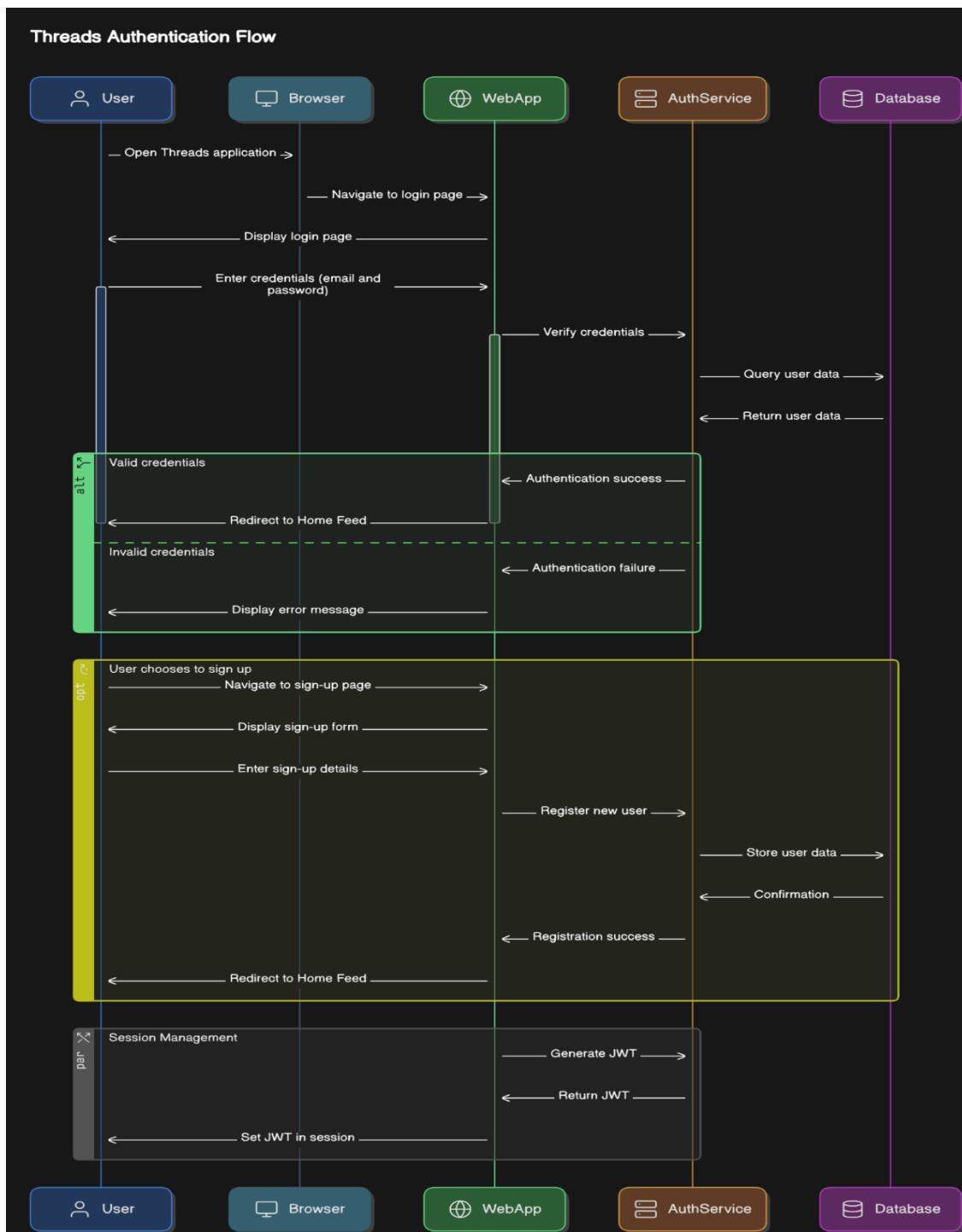
By implementing the above structure and logic, you have created a robust backend for your social media web application. This backend can handle user authentication, post creation, and various other functionalities required by the frontend to operate smoothly.

CHAPTER 5

TESTING

Client-side Form Validation

In the Threads project, we implemented client-side form validation using plain JavaScript. This approach ensures that we provide immediate feedback to users, enhancing the overall user experience by preventing form submission until all fields meet the specified criteria.



Client-side Validation Process

1. Name Validation:

- The name field must be at least 3 characters long.
- It must not contain any special characters.

2. Username Validation:

- The username must be at least 5 characters long.
- It should contain at least one letter and one number.
- Only alphanumeric characters and underscores are allowed.

3. Email Validation:

- The email must follow a standard email format (e.g., user@example.com).

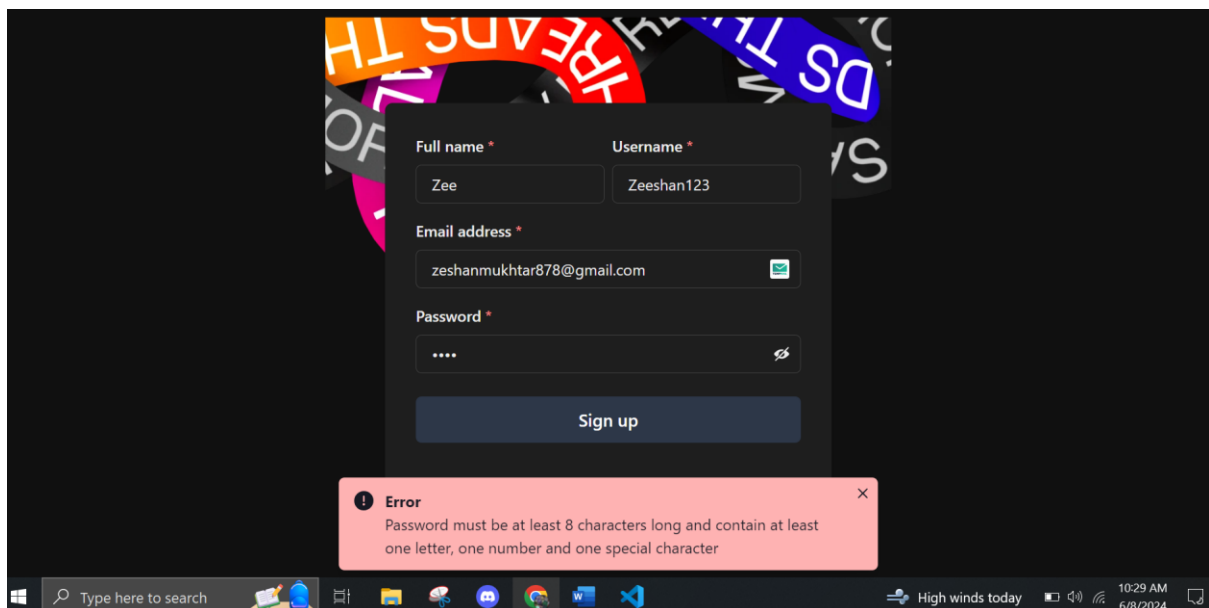
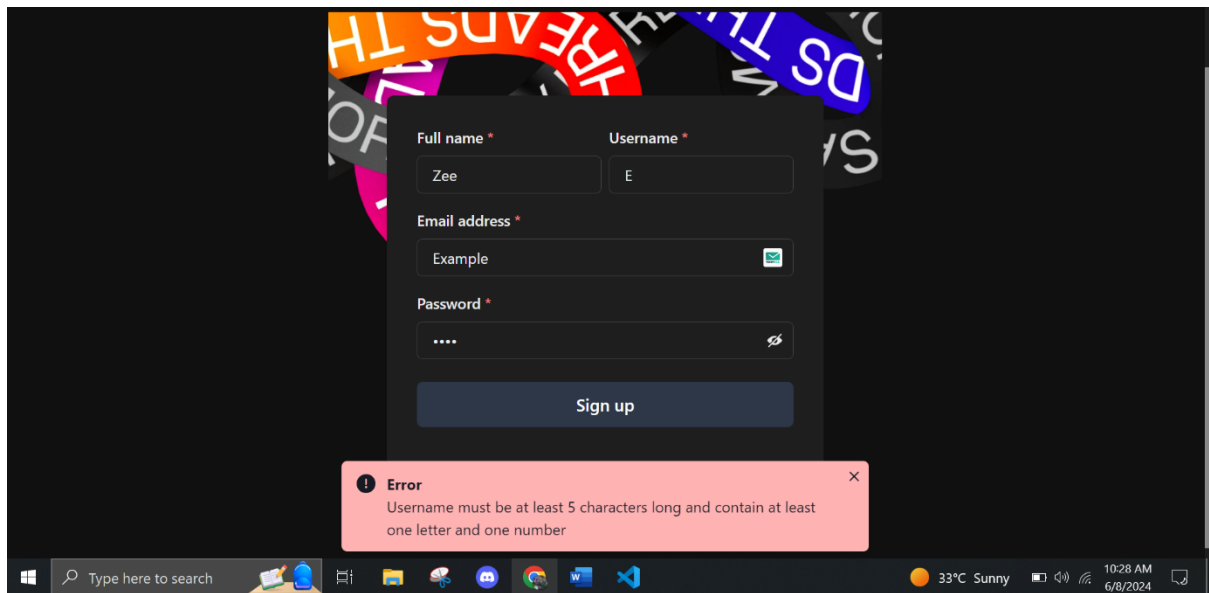
4. Password Validation:

- The password must be at least 8 characters long.
- It must contain at least one letter, one number, and one special character.

Handling Validation Errors

To handle validation errors on the client side, we display error messages next to the corresponding form fields. This is achieved by:

- Attaching event listeners to form fields to validate input as the user types.
- Displaying error messages dynamically if the input does not meet the validation criteria.
- Using `react-hot-toast` to show error messages in a more user-friendly manner.



Server-side Error Handling

For server-side error handling, we ensure robust validation to maintain data integrity and security. This is crucial as it provides a second layer of validation to catch any invalid data that might bypass client-side checks.

Server-side Validation Process

1. Name Validation:

- The server checks if the name is at least 3 characters long.
- It ensures the name does not contain any special characters.

2. Username Validation:

- The server verifies that the username is at least 5 characters long and contains at least one letter and one number.
- It also checks for the presence of only allowed characters (alphanumeric and underscores).

3. Email Validation:

- The server validates the email against a standard email format to ensure it is correctly structured.

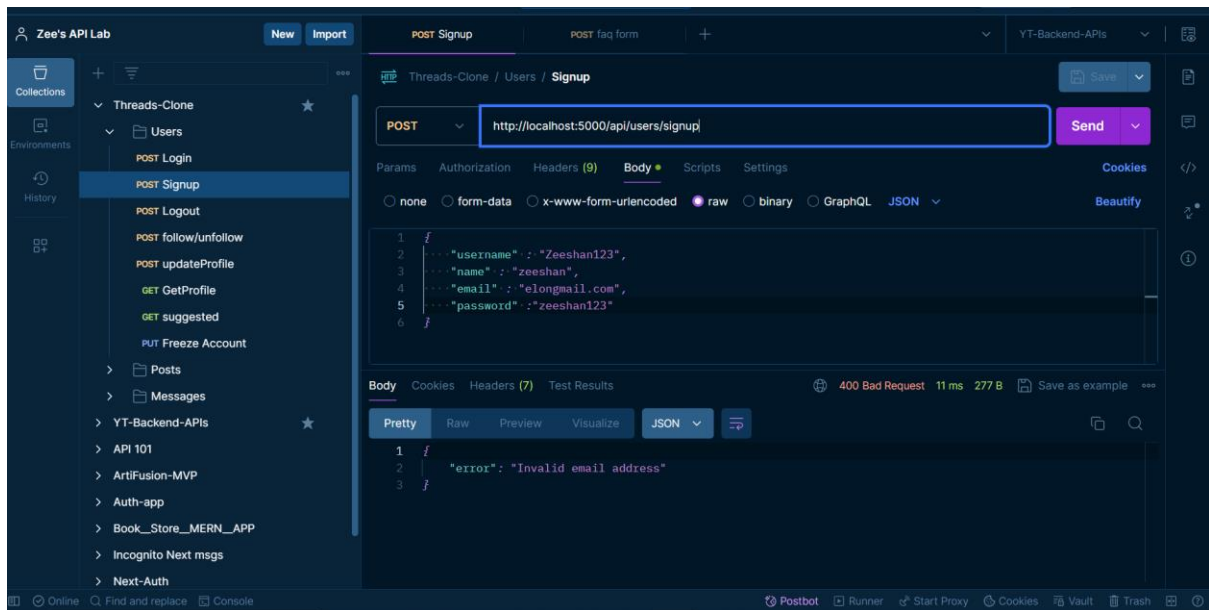
4. Password Validation:

- The server ensures that the password is at least 8 characters long.
- It checks that the password includes at least one letter, one number, and one special character.

Error Notification

On the server side, when validation errors are detected, the server responds with appropriate error messages which are then displayed to the user. This process involves:

- Sending a descriptive error message back to the client if the input data does not meet the validation criteria.
- Using ``react-hot-toast`` on the client side to notify users of any validation errors that occur during server-side processing.



Summary

By implementing comprehensive form validation on both the client and server sides, we ensure that user input is thoroughly checked and validated. This dual-layer approach enhances the security and integrity of the data in the Threads project, providing a reliable and user-friendly experience.

CHAPTER 6

DEPLOYMENT AND HOSTING

Deployment Process

For the Threads project, the deployment process is crucial to ensure that our web application is accessible to users in a stable and scalable manner. However, due to the nature of our project involving web sockets, we faced challenges with free hosting providers. Currently, we are conducting local testing, but we plan to deploy using a cloud provider once the project scales up. Here is a typical deployment process we intend to follow:

1. Sign Up for a Hosting Account:

- Choose a suitable hosting provider like AWS, Google Cloud, Azure, or others that support web sockets.
- Sign up for an account if you do not have one already.

2. Install Required CLI Tools:

- Depending on the hosting provider, install the necessary CLI tools for deployment. For example, for Vercel, you would run:
- `npm install -g vercel`

3. Build the Web App:

- Ensure the Threads application is production ready. For a React or Vite application, use:
- `npm run build`
- This command will create a production-ready build of the application.

4. Configure the Project for Deployment:

- Navigate to the root directory of your project in the terminal.
- If using Vercel, log in using:
- `vercel login`

5. Deploy the Application:

Follow the provider-specific steps to deploy your application. For Vercel, you would use:

- `vercel`
- Provide the necessary details when prompted, such as the project directory and build commands.

6. Configure Environment Variables:

If your application requires environment variables, set them up in the hosting provider's dashboard. For Vercel, navigate to the project settings and add the necessary variables under the "Environment Variables" section.

7. Set Up Custom Domains (Optional):

If you have a custom domain, configure it through the hosting provider's dashboard. For instance, in Vercel, go to the "Domains" section and follow the instructions to add your domain.

8. Continuous Deployment (Optional):

Enable continuous deployment to automatically deploy changes whenever you push updates to your repository. Connect your GitHub, GitLab, or Bitbucket repository to the hosting provider for this feature.

Hosting Providers

Given the constraints and the specific needs of the Threads project, especially regarding web sockets, we can use some potential hosting providers we may consider once we move beyond local testing:

1. AWS (Amazon Web Services):

AWS offers a comprehensive suite of services, including EC2 for virtual servers, S3 for storage, and RDS for databases. It's a robust option for scalable and reliable hosting.

2. Google Cloud Platform:

Google Cloud provides similar services to AWS, including Compute Engine for virtual machines, Cloud Storage for storage, and Cloud SQL for databases. It's known for its high performance and integration with other Google services.

3. Microsoft Azure:

Azure offers various hosting services such as Virtual Machines, Blob Storage, and Azure SQL Database. It's a good choice for those already using Microsoft technologies.

4. Heroku:

Heroku is a PaaS that simplifies the deployment and management of web applications. It supports various programming languages and frameworks, making it a versatile option.

5. Digital Ocean:

Digital Ocean provides virtual private servers (Droplets) for hosting web applications. It's known for its simplicity and developer-friendly interface.

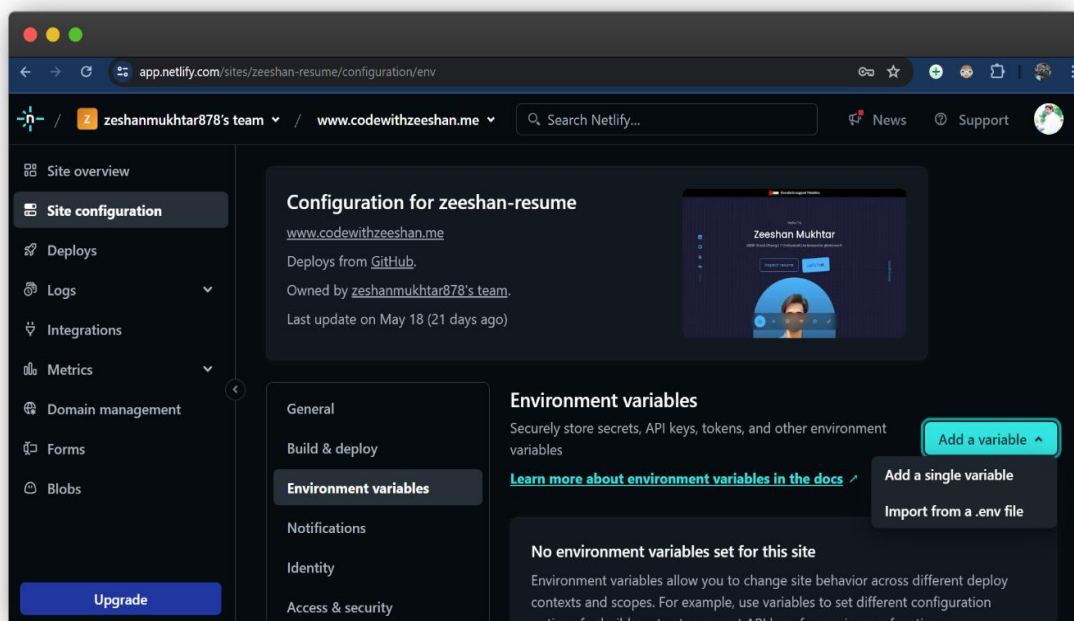
6. Netlify:

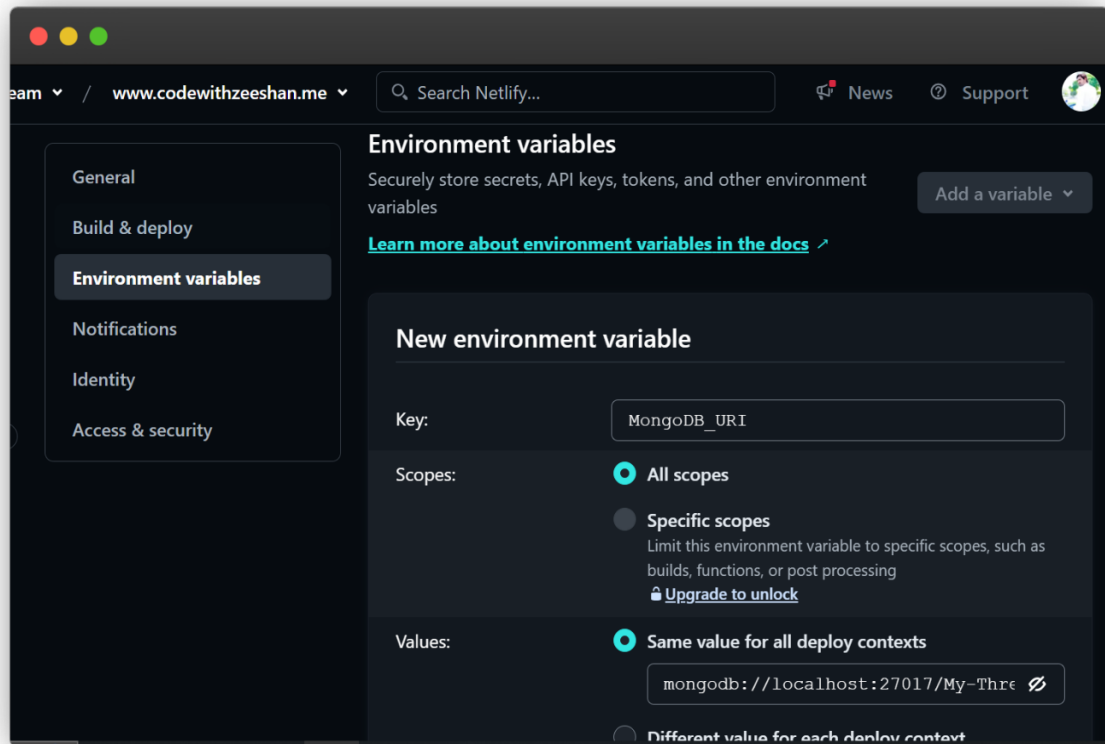
Netlify is designed for static websites and JAM stack applications, offering features like continuous deployment, CDN, and serverless functions. It's a good option for simpler projects.

7. Vercel:

Vercel specializes in static websites and serverless functions and integrates well with Next.js. It's popular for deploying Next.js applications and offers features like automatic scaling and serverless functions.

Visual representation of how we add environment variables:





Summary

While the Threads project currently undergoes local testing due to the limitations of free hosting providers supporting web sockets, we plan to transition to a suitable cloud provider as the project scales. This will ensure that our application remains robust, scalable, and accessible to our users.

HAPTER 7

CHALLENGES AND SOLUTIONS

Key Challenges Faced

During the development of the Threads project, several key challenges were encountered. These challenges included:

1. Robust Authentication:

Implementing a secure and reliable authentication system was essential. Ensuring that user data was protected and that only authorized users could access certain features required a robust solution.

2. Modern UI Design:

Designing a user interface that was both modern Mobile responsive and user-friendly was a significant challenge. Creating a visually appealing design that also provided a seamless user experience required careful consideration.

3. User Engagement Concerns:

There was a concern about whether users would actively use the application. This led to the integration of AI-powered post creation to enhance user engagement and provide unique features.

4. Cost Issues for Deployment:

Deploying the application with web sockets functionality was challenging due to the costs associated with hosting. Free hosting providers often do not support web sockets, necessitating a local testing environment until the project scales up.

5. API Abuse:

Some users could potentially misuse the AI features by constantly hitting the API for fun, which would strain the resources and exceed the usage limits.

6.Ongoing Bug Fixes:

Like any software project, ongoing testing and bug fixing were necessary to ensure the application worked correctly and provided a good user experience.

Solutions Implemented

1. Core JS for Authentication:

To address the need for robust authentication, Core JS was used to implement secure client-side and server-side form validations. This ensured that user data was properly validated and protected against common security threats.

2. Black and White Theme for Modern UI:

A modern and minimalist black and white theme was chosen for the UI design. This provided a sleek and professional look while maintaining a user-friendly interface. Feedback from users was continuously incorporated to refine the design and improve usability.

3. AI-Powered Post Creation:

To enhance user engagement and provide value, an AI-powered post creation feature was integrated into the application. This allowed users to generate creative and interesting posts, encouraging more active use of the app.

4. Deployment Strategy:

Due to the costs associated with hosting web sockets, the initial deployment was conducted on a local testing environment. Plans are in place to transition to a suitable hosting provider once the project scales up, ensuring scalability and stability.

5.API Rate Limiting:

To prevent API abuse, rate limiting was implemented. This involved setting limits on the number of API requests a user could make within a certain time. Middleware or custom logic was used to track and enforce these limits, ensuring fair usage and protecting against abuse.

6. Ongoing Bug Fixes and Testing:

Regular testing and debugging processes were established to identify and fix bugs promptly. User feedback was actively sought to identify issues and improve the overall user experience. This ongoing effort ensures the application remains reliable and user-friendly.

By addressing these challenges with the above solutions, the Threads project aims to provide a secure, engaging, and user-friendly platform for its users. Continuous improvements and scalability plans will further enhance the application's performance and reach.

CHAPTER 8

FUTURE ENHANCEMENTS

Planned Future Features and Enhancements

Here are some planned future features and enhancements for the Threads app:

Real-Time Collaboration:

Implement a real-time collaboration feature that allows multiple users to work on the same post simultaneously. This will enhance teamwork and facilitate more dynamic content creation.

Advanced Customization Options Provide more advanced customization options for posts, such as custom fonts, themes, React Rich Text Editor and layouts. This will give users more control over the appearance of their content.

Enhanced AI Features:

Improve the AI-powered post creation feature to create Ai image posts as well.

Mobile Application:

Develop a mobile version of the Threads app in React Native to enable users to create and manage their posts on the go. This will increase accessibility and convenience for users.

Integration with Social Media Platforms:

Implement integration with popular social media platforms such as Twitter, Facebook, and Instagram. This will allow users to share their Threads posts directly to their social media accounts.

User Analytics and Insight:

Provide users with analytics and insights about their posts, such as engagement metrics, audience demographics, and post-performance. This will help users understand their audience better and optimize their content.

Multi-Language Support:

Add support for multiple languages to cater to a global audience. This will allow users from different regions to create and interact with content in their preferred language.

Advanced Search and Filtering:

Implement advanced search and filtering options to help users find specific posts or threads quickly and efficiently. This will enhance the overall user experience and make content discovery easier.

Improved Security and Privacy Features:

Continuously enhance the security and privacy features of the app to protect user data and ensure compliance with data protection regulations.

Subscription Plans: Introduce subscription plans with premium features and additional benefits. This will provide a revenue stream for the app while offering users more value.

Community and Forum Features:

Create a community section where users can discuss topics, share ideas, and provide feedback. This will foster a sense of community and encourage user engagement.

Automated Moderation Tools:

Develop automated moderation tools to detect and handle inappropriate content, spam, and abusive behavior. This will help maintain a positive and safe environment for all users.

By implementing these future enhancements, the Threads app aims to provide a richer and more engaging experience for its users. These features will help the app grow, attract more users, and provide greater value to its community.

CHAPTER 9

CONCLUSION

Summary of Project Achievements

The development of the "Threads" web application marks a significant milestone in the realm of real-time collaborative platforms. This section encapsulates the key accomplishments and milestones achieved throughout the project's development lifecycle.

1. Robust Authentication System:

Implemented a secure and efficient authentication system using Core JS, ensuring that user data and privacy are always protected. This robust authentication framework provides a seamless and secure user experience.

2. Modern User Interface:

Designed a sleek, modern, and minimalist user interface featuring a black and white theme. This design choice enhances readability and user focus, providing a clean and professional appearance.

3. AI-Powered Content Creation:

Integrated AI capabilities for post creation, allowing users to generate content effortlessly. This feature addresses the concern of low user engagement by providing intelligent content suggestions and automation.

4. Scalability and Cost-Effective Deployment:

Due to the cost constraints associated with deploying web sockets, the project is designed to scale up when necessary. Future deployment plans include leveraging cost-effective solutions to handle increased traffic and user demands.

5. API Rate Limiting:

Implemented API rate limiting to prevent misuse and ensure fair usage of resources. This feature protects the platform from potential abuse by users who might excessively hit the API for non-productive purposes.

6. Continuous Bug Fixing and Improvements:

A commitment to ongoing bug fixing and feature enhancements ensures that the application remains stable, secure, and user-friendly. Regular updates and maintenance are prioritized to address user feedback and improve overall performance.

7. Realization of Project Objectives:

The project has successfully met its core objectives, providing a versatile, user-friendly, and scalable platform for real-time collaboration. The achievement of these goals underscores the team's dedication to delivering a high-quality product.

8. User Suggestions and Following:

Developed a dynamic sidebar that displays suggested users for the current user to follow or unfollow. This list is intelligently shuffled and excludes the current user, enhancing the social networking experience by promoting interaction and connectivity.

9.Account Freeze Feature:

Introduced an account freeze feature that allows users to temporarily freeze their accounts. Frozen accounts can still be searched and followed by other users but cannot be accessed or unfrozen by anyone except the original user, ensuring security and control.

In conclusion, the "Threads" web application stands as a testament to innovation and user-centric design in real-time collaborative platforms. By integrating advanced technologies and maintaining a focus on usability and security, the project has laid a strong foundation for future growth and enhancements. The dedication and technical expertise of the development team have culminated in a powerful tool that promises to evolve and adapt to the needs of its users.

Contributions to the Field

The "Threads" web application has made significant contributions to the fields of web development, real-time collaboration, and user experience. This section highlights the impactful contributions and advancements brought forth by the project.

1. Advancement in Real-Time Collaboration:

"Threads" has pushed the boundaries of real-time collaborative technology, providing a platform where users can interact and work together seamlessly. This advancement enhances productivity and fosters a more connected digital workspace.

2.User-Centric Design and Intuitive Interface:

The project's emphasis on a modern, minimalist design contributes to the field by demonstrating the importance of simplicity and user focus in UI/UX design. The intuitive interface ensures a positive user experience and sets a standard for future applications.

3. Secure and Efficient Authentication:

The implementation of a robust authentication system using Core JS highlights best practices in security and user management. This contribution reinforces the importance of safeguarding user data in web applications.

4. AI Integration for Enhanced User Engagement:

By incorporating AI-powered content creation, "Threads" showcases the potential of AI in boosting user engagement and productivity. This innovative feature provides valuable insights and automation capabilities for content generation.

5. Scalable Architecture and Cost-Effective Solutions:

The project's scalable architecture and strategic approach to deployment underscore the importance of planning for growth while managing costs effectively. This contribution serves as a model for other developers facing similar challenges.

6. API Rate Limiting for Fair Resource Usage:

Implementing API rate limiting demonstrates a proactive approach to managing resources and preventing abuse. This contribution highlights the need for balanced resource allocation in web applications.

7. Ongoing Maintenance and Improvement:

The project's commitment to continuous improvement and regular updates sets a precedent for maintaining software quality and user satisfaction. This dedication to ongoing development ensures that the application remains relevant and effective.

8. Interdisciplinary Application:

"Threads" bridges various technical domains, from real-time collaboration to AI integration and user authentication. This interdisciplinary approach enriches the field by demonstrating the synergies between different technological advancements.

9. Dynamic User Suggestions:

The introduction of a dynamic user suggestion feature enriches the social networking aspect of the platform. By providing a constantly updating list of users to follow, this feature encourages interaction and community building.

10. Account Freeze Functionality:

The innovative account freeze functionality offers users greater control over their privacy and account activity. This feature ensures that users can maintain their presence on the platform without active participation until they choose to unfreeze their account

In Summary, the "Threads" web application has made noteworthy contributions to web development and user experience. Its innovative features, user-centric design, and commitment to security and scalability serve as valuable examples for future projects. The application's success and impact underscore the transformative power of technology when applied with a focus on usability, security, and continuous improvement.

CHAPTER 10

REFERENCES

External Libraries, Frameworks, and Resources Used

External Libraries:

✓ **React:**

Official website. Available at: [<https://reactjs.org/>] (<https://reactjs.org/>)

✓ **JavaScript:**

Official website. Available at: (<https://www.javascript.com/>)

✓ **Chakra-Ui:**

Official website. Available at: (<https://v2.chakra-ui.com/getting-started>)

✓ **@chakra-ui/icons:**

Official website. Available at: (<https://chakra-ui.com/docs/components/icon>)

✓ **Bard API Node:**

Documentation and resources. Available at: [<https://www.npmjs.com/package/bard-api-node>] (<https://www.npmjs.com/package/bard-api-node>)

✓ **Date-fns:**

Official website. Available at: [<https://date-fns.org/>] (<https://date-fns.org/>)

✓ **React-icons:**

Official website. Available at: [<https://react-icons.github.io/react-icons/>] (<https://react-icons.github.io/react-icons/>)

✓ **React-router-Dom:**

Official website. Available at: [<https://reactrouter.com/>] (<https://reactrouter.com/>)

✓ **Recoil:**

Official website. Available at: [<https://recoiljs.org/>] (<https://recoiljs.org/>)

✓ **Sass:**

Official website. Available at: [<https://sass-lang.com/>] (<https://sass-lang.com/>)

✓ **Socket.io-client:**

Official website. Available at: [<https://socket.io/docs/v4/client-api/>] (<https://socket.io/docs/v4/client-api/>)

Dev Dependencies:

✓ @types/react:

Official website. Available at: [<https://www.npmjs.com/package/@types/react>]
(<https://www.npmjs.com/package/@types/react>)

✓ @types/react-dom:

Official website. Available at: [<https://www.npmjs.com/package/@types/react-dom>]
(<https://www.npmjs.com/package/@types/react-dom>)

✓ @vitejs/plugin-react:

Documentation and resources. Available at: [<https://vitejs.dev/plugin/vite-plugin-react.html>]
(<https://vitejs.dev/plugin/vite-plugin-react.html>)

✓ ESLint:

Official website. Available at: [<https://eslint.org/>] (<https://eslint.org/>)

✓ ESLint-plugin-react:

Official website. Available at: [<https://www.npmjs.com/package/eslint-plugin-react>]
(<https://www.npmjs.com/package/eslint-plugin-react>)

✓ ESLint-plugin-react-hooks:

Documentation and resources. Available at: [<https://www.npmjs.com/package/eslint-plugin-react-hooks>]
(<https://www.npmjs.com/package/eslint-plugin-react-hooks>)

✓ ESLint-plugin-react-refresh:

Documentation and resources. Available at: [<https://www.npmjs.com/package/eslint-plugin-react-refresh>]
(<https://www.npmjs.com/package/eslint-plugin-react-refresh>)

✓ Vite:

Official website. Available at: [<https://vitejs.dev/>] (<https://vitejs.dev/>)

Backend Libraries:

✓ bcryptjs:

Official website. Available at: [<https://www.npmjs.com/package/bcryptjs>]
(<https://www.npmjs.com/package/bcryptjs>)

✓ **Cloudinary:**

Official website. Available at: [<https://cloudinary.com/documentation>] (<https://cloudinary.com/documentation>)

✓ **cookie-parser:**

Documentation and resources. Available at: [<https://www.npmjs.com/package/cookie-parser>] (<https://www.npmjs.com/package/cookie-parser>)

✓ **Dotenv:**

Documentation and resources. Available at: [<https://www.npmjs.com/package/dotenv>] (<https://www.npmjs.com/package/dotenv>)

✓ **Express.js:**

Official website. Available at: [<https://expressjs.com/>] (<https://expressjs.com/>)

✓ **jsonwebtoken:**

Official website. Available at: [<https://www.npmjs.com/package/jsonwebtoken>] (<https://www.npmjs.com/package/jsonwebtoken>)

✓ **Mongoose:**

Official website. Available at: [<https://mongoosejs.com/>] (<https://mongoosejs.com/>)

✓ **Socket.io:**

Official website. Available at: [<https://socket.io/>] (<https://socket.io/>)

Backend Dev Dependencies:

✓ **Nodemon:**

Official website. Available at: [<https://www.npmjs.com/package/nodemon>] (<https://www.npmjs.com/package/nodemon>)

Resources:

✓ **As a Programmer YT channel:**

Tutorial videos on building a full-stack Threads web app. Available at: (<https://youtu.be/YR5IUtAPwhg?si=Rq2G7MpWPyCx0bjf>)

✓ **OpenAI API:**

Documentation for integrating AI functionalities. Available at:
[<https://beta.openai.com/docs/>] (<https://beta.openai.com/docs/>)

✓ **Socket.io:**

Official website and documentation for WebSocket implementation. Available at:
[<https://socket.io/>] (<https://socket.io/>)

These references collectively represent the essential libraries, frameworks, and resources that were utilized during the development of the "Threads" web application. Each resource played a crucial role in shaping the functionality, design, and performance of the final product.