

SOCIAL MEDIA WEBAPP WITH MERN STACK

A PROJECT REPORT

Submitted by

ANKIT KUMAR SINGH (20BCS7529)

SHUBHAM DUBEY (20BCS9234)

KAUSIK SARKAR (20BCS5335)

PRIYA JAISWAL (20BCS2600)

DEV MAHESHWARI (20BCS9620)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



Chandigarh University

DECEMBER 2023



BONAFIDE CERTIFICATE

This is Certified that this project report **“SOCIAL MEDIA WEBAPP WITH MERN STACK”** is the bonafide work of **“Ankit kumar Singh”** UID **“20BCS7529”**, **“Shubham Dubey”** UID **“20BCS9234”**, **“Kausik Sarkar”** UID **“20BCS5335”**, **“Priya Jaiswal”** UID **“20BCS2600”**, **“Dev Maheshwari”** UID **“20BCS9620”**, who carried out the project work under my/our supervision.

SIGNATURE

Dr. Navpreet Kaur Walia

HEAD OF THE DEPARTMENT

Computer Science and engineering

SIGNATURE

Darshan Kaur

SUPERVISOR

Computer Science and engineering

Submitted for the project viva-voce examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

TABLE OF CONTENT

List of Figures	I
Abstract	II
संक्षेप	III
Graphical Abstract	IV
Abbreviations	V
Chapter 1: Introduction	1-11
1.1 Client Identification/Need Identification	2
1.1.1 Historical Context	3
1.1.2 Foundation of Social Media app	4
1.2 Identification of Problem	5
1.3 Identification of Tasks	7
1.4 Timeline	8
1.5 Organization of the Report	10
Chapter 2: Literature Review	12-23
2.1 Timeline of the reported problem	13
2.2 Proposed Solutions and Bibliographic Analysis	14
2.2.1 Facebook	15
2.2.2 Instagram	15
2.2.3 Twitter	16
2.2.4 Pinterest	16
2.3 Bibliometric analysis	17
2.4 Review Summary	19
2.5 Problem Definition	20
2.6 Goals/Objectives	23
Chapter 3: Design Flow/Process	24-37
3.1 Evaluation and Selection of Features	24
3.2 Design constraints	27

3.3 Analysis and Feature Finalization Subject to Constraints	29
3.4 Design Flow	32
3.5 Design Selection	33
3.6 Implementation plan / Methodology	35
Chapter 4: Result Analysis and Validation	38-42
4.1 Implementation of Solution	38
Chapter 5: Conclusion and Future Scope	43-47
5.1 Conclusion	44
5.2 Future Scope	46
References	48
Appendix	51
User Manual	59

List of Figures

Figure 3.1 Control flow Diagram.....	42
Figure 3.2 Data flow diagram 1.....	45
Figure 3.3 Data flow diagram 2	63
Figure 3.4 Framework Uses.....	63
Figure 4.1 index.js.....	64
Figure 4.2 Post.js	65
Figure 4.3 Login Page	65
Figure 4.3 Register Page	65
Figure 4.3 Home page	65

List of Tables

Table 1. Timeline.....	9
-------------------------------	----------

संक्षेप

इस परियोजना का लक्ष्य MERN (MongoDB, Express.js, React.js, Node.js) स्टैक का उपयोग करके एक आधुनिक सोशल मीडिया वेब एप्लिकेशन को डिजाइन और कार्यान्वित करना है। सोशल मीडिया वेब ऐप उपयोगकर्ताओं को कनेक्ट करने, सामग्री साझा करने और सामाजिक इंटरैक्शन में संलग्न होने के लिए एक मंच प्रदान करता है। फ्रंटएंड को React.js का उपयोग करके विकसित किया गया है, जो एक उत्तरदायी और इंटरैक्टिव उपयोगकर्ता इंटरफ़ेस प्रदान करता है। उपयोगकर्ता अनुभव में उपयोगकर्ता प्रोफ़ाइल बनाने और संपादित करने, टेक्स्ट और मल्टीमीडिया सामग्री पोस्ट करने, पोस्ट को पसंद करने और टिप्पणी करने और अन्य उपयोगकर्ताओं का अनुसरण करने जैसी सुविधाएं शामिल हैं। React.js घटकों को एक स्केलेबल और रखरखाव योग्य कोडबेस सुनिश्चित करने के लिए पुनः प्रयोज्यता और मॉड्यूलरिटी के लिए डिज़ाइन किया गया है। एप्लिकेशन उपयोगकर्ता खातों को सुरक्षित करने और संवेदनशील जानकारी की सुरक्षा के लिए JWT (JSON वेब टोकन) के माध्यम से उपयोगकर्ता प्रमाणीकरण को शामिल करता है। इसके अतिरिक्त, प्रोजेक्ट वेबसॉकेट तकनीक का उपयोग करके वास्तविक समय के अपडेट लागू करता है, जिससे उपयोगकर्ताओं को नए पोस्ट, टिप्पणियों और अन्य प्रासंगिक गतिविधियों के लिए तत्काल सूचनाएं प्राप्त करने में सक्षम बनाया जाता है।

ABSTRACT

This project aims to design and implement a modern Social Media Web Application using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The Social Media Web App provides users with a platform to connect, share content, and engage in social interactions. The frontend is developed using React.js, offering a responsive and interactive user interface. The user experience includes features such as creating and editing user profiles, posting text and multimedia content, liking and commenting on posts, and following other users. The React.js components are designed for reusability and modularity to ensure a scalable and maintainable codebase. The application incorporates user authentication through JWT (JSON Web Tokens) to secure user accounts and protect sensitive information. Additionally, the project implements real-time updates using WebSocket technology, enabling users to receive immediate notifications for new posts, comments, and other relevant activities.

The Social Media Web App is designed with a focus on scalability, performance, and responsiveness, ensuring a seamless user experience across various devices and screen sizes. The MERN stack's flexibility allows for easy deployment on cloud platforms, making the application accessible to a wide audience. This project not only showcases the technical skills in building a full-stack web application but also addresses the growing demand for social media platforms that provide a secure and engaging environment for users to connect and share their experiences. In response to the growing demand for interactive and user-centric online platforms, this web application offers a powerful combination of features, including user authentication, real-time updates, multimedia content sharing, and social engagement. The application's core components encompass user registration and login functionality, empowering users to create and manage their profiles securely.

GRAPHICAL ABSTRACT



ABBREVIATIONS

1. MERN: MongoDB, Express.js, ReactJS, Node.js
2. JS: JavaScript
3. DB: Database
4. HTML: HyperText Markup Language
5. CSS: Cascading Style Sheet
6. PHP: PHP HyperText Preprocessor
7. UI: User Interface
8. UX: User Experience
9. API: Application Programming Interface
10. PKI: Public Key Infrastructure
11. SDLC: Software Development Life Cycle
12. MEAN: MongoDB, Express.js, AngularJS, Node.js
13. HTTP: HyperText transfer Protocol
14. HTTPS: HyperText transfer Protocol Secure
15. SEO: Search Engine Optimization
16. OAUTH: Open Authorization
17. JWT: JSON Web Token
18. GDPR: General Data Protection Regulation
19. PCI: Peripheral Component Interconnect

CHAPTER 1

INTRODUCTION

In the age of digital connectivity, Social Connect emerges as a dynamic social media platform designed to connect individuals, foster relationships, and facilitate meaningful interactions. Built on the robust MERN stack - MongoDB, Express.js, React.js, and Node.js - SocialConnect combines a powerful backend with a responsive and intuitive frontend to deliver a seamless user experience.

1.1.Client Identification

Developing a social media web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack involves a comprehensive process of client identification, where user-centric design and functionality are paramount. The MongoDB database is employed to store and manage user profiles, posts, and related data. Express.js facilitates the creation of a robust and scalable server that handles client requests and interacts with the database. Node.js powers the server-side runtime environment, ensuring efficient handling of concurrent connections and high-performance data processing.

The React.js front-end framework is instrumental in crafting an intuitive and dynamic user interface, allowing seamless interaction and real-time updates. Through React components, user profiles are dynamically rendered, and the application's state is managed to ensure a smooth and responsive experience. The client identification process involves secure user authentication and authorization mechanisms, ensuring the protection of sensitive user information.

To achieve this, the MERN stack leverages industry-standard authentication protocols, such as OAuth or JWT (JSON Web Token), to verify user identities during login and register processes. Passwords are securely hashed and stored in the MongoDB database to enhance data security.

Social media authentication integration, using OAuth providers like Facebook or Google, further streamlines user onboarding and authentication processes.

The client-side application is designed with user-centric features, including profile customization, friend requests, and news feed functionalities. React.js facilitates the creation of reusable components, ensuring a modular and maintainable codebase. The application's front end communicates with the server through RESTful APIs, allowing for seamless data exchange. Real-time communication is achieved using WebSocket technology or libraries like Socket.io, enabling instant updates on posts, comments, and friend requests.

User interactions with the social media web app are enhanced through an intuitive and aesthetically pleasing user interface designed with React.js. Responsive design principles are implemented to ensure a seamless experience across various devices and screen sizes. The client-side application utilizes state management libraries like Redux for efficient data flow and centralized state management, enabling smooth navigation and dynamic content updates.

To enhance user engagement, the MERN stack leverages push notifications and email alerts for new friend requests, comments, or other relevant activities. Node.js handles asynchronous tasks, such as sending notifications, without affecting the application's performance. User-generated content, such as images and videos, is efficiently processed, stored, and retrieved using MongoDB's GridFS, ensuring optimal performance and scalability.

Client identification extends to user analytics and tracking, facilitated by integrating tools like Google Analytics or custom analytics solutions. These tools provide valuable insights into user behavior, enabling continuous improvement of the application based on user preferences and trends. The MERN stack's flexibility allows for seamless integration with third-party services and APIs, enhancing the social media web app's functionality and keeping it in line with industry standards.

To ensure data security and privacy, the MERN stack incorporates encryption protocols for data transmission and storage. SSL/TLS protocols secure data in transit, while MongoDB's encryption

features protect data at rest. Additionally, access control mechanisms are implemented at both the server and database levels, restricting unauthorized access to sensitive user information.

In conclusion, the development of a social media web application with the MERN stack involves a holistic approach to client identification, emphasizing user-centric design, security, and real-time functionality. The combination of MongoDB, Express.js, React.js, and Node.js provides a powerful and flexible foundation for creating a feature-rich, scalable, and engaging social media platform. Through effective client identification processes, the MERN stack ensures a seamless and secure user experience, making it an ideal choice for modern web application development.

1.1 Historical context:

In the early 21st century, the rapid evolution of digital communication and the widespread adoption of the internet paved the way for the development of social media platforms. During this era, the MERN (MongoDB, Express.js, React.js, Node.js) stack emerged as a popular technology stack for web application development. Leveraging the power of MERN, developers created innovative social media web apps that transformed the way individuals connect, share information, and engage with each other online. MongoDB, a NoSQL database, provided scalability for handling large volumes of user data, while Express.js facilitated the development of robust and efficient server-side applications. React.js, a front-end library, enabled the creation of dynamic and responsive user interfaces, enhancing the user experience. Node.js, with its event-driven architecture, facilitated the development of fast and scalable server-side applications. Together, the MERN stack played a pivotal role in shaping the landscape of social media platforms, contributing to the interconnected and digitally-driven social fabric of the time.

1.2 Foundation of Social Media:

The foundation of a social media web app using the MERN (MongoDB, Express.js, React, Node.js) stack involves designing a robust and scalable architecture. MongoDB serves as the database to store user profiles, posts, and interactions, while Express.js facilitates server-side

development and API creation. React, a powerful front-end library, is employed for building dynamic and responsive user interfaces, providing an immersive experience. Node.js acts as the server runtime, handling server-side logic and enabling real-time communication through technologies like WebSocket. Integrating user authentication, implementing a seamless data flow, and optimizing performance are key considerations to establish a solid foundation for a MERN-based social media web app.

1.2. Identification of Problem

The social media web application developed with the MERN (MongoDB, Express.js, React.js, Node.js) stack faces a multifaceted array of challenges that collectively impede its optimal functionality and user experience. One significant problem lies in the scalability of the application, as it struggles to efficiently handle an increasing user base and growing data volumes. The MongoDB database, while flexible, may encounter performance issues when dealing with extensive datasets, leading to slower query responses and potentially hampering the overall responsiveness of the application. Additionally, the Express.js backend framework may face challenges in handling concurrent requests, affecting the app's ability to maintain responsiveness during periods of high traffic.

Another critical issue involves security vulnerabilities that can compromise user data and privacy. The decentralized nature of the MERN stack, with separate modules for frontend and backend, requires meticulous attention to security configurations to prevent potential exploits. Inadequate data validation and lack of proper authentication mechanisms may expose the application to various threats, such as cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks, putting user information at risk.

Moreover, the React.js frontend faces performance issues related to excessive rendering and inefficient state management. The virtual DOM, while enhancing React's efficiency, may still result in unnecessary re-renders and impact the application's responsiveness, particularly in complex user interfaces with dynamic content. Inefficient state management practices, such as

prop drilling or suboptimal use of state management libraries like Redux, can contribute to a convoluted and error-prone codebase, making maintenance and feature development challenging. Another significant problem lies in the lack of real-time features and poor handling of WebSocket communication. Social media applications require timely updates and notifications, and the traditional request-response cycle of HTTP may not suffice. Integrating WebSocket technology for real-time communication is essential but often overlooked in MERN stack applications, leading to delayed updates and a subpar user experience.

Furthermore, the absence of comprehensive testing strategies, including unit testing, integration testing, and end-to-end testing, poses a significant challenge. Inadequate test coverage may result in undetected bugs and hinder the ability to confidently deploy new features or updates. Implementing a robust testing framework is crucial to ensure the reliability and stability of the social media web app, especially as it evolves over time.

The overall user experience also suffers from suboptimal design and poor accessibility. While React.js provides a powerful framework for building interactive user interfaces, the design choices made during implementation may not prioritize accessibility standards, limiting the inclusivity of the application. Inconsistent UI/UX patterns, coupled with non-responsive design elements, contribute to a disjointed and frustrating user experience across different devices and screen sizes.

Additionally, the lack of proper error handling and feedback mechanisms further diminishes the user experience. Insufficient validation checks and error messages may confuse users, making it difficult for them to understand and rectify issues. Improving error reporting and implementing user-friendly feedback mechanisms are critical steps toward enhancing the overall usability and user satisfaction of the social media web app.

1.3. Identification of Tasks

Building a social media web application using the MERN (MongoDB, Express.js, React, Node.js) stack involves a multifaceted approach that encompasses various tasks across frontend and backend development. On the frontend, the React framework facilitates the creation of a dynamic and responsive user interface, allowing users to seamlessly navigate and interact with the

application. Components such as user profiles, news feeds, and messaging features are designed to enhance user engagement.

The backend, powered by Node.js and Express.js, manages the server-side logic and communication with the database. MongoDB, a NoSQL database, is employed to store and retrieve user data efficiently. User authentication and authorization are crucial tasks, requiring the implementation of secure registration and login processes. JSON Web Tokens (JWT) may be employed to ensure secure and stateless authentication. Additionally, user-generated content, such as posts, comments, and media uploads, needs to be handled and stored appropriately in the database.

Real-time features, a hallmark of social media platforms, are implemented using technologies like WebSocket or libraries like Socket.io, allowing instant updates and notifications for users. Scalability considerations are addressed by optimizing database queries, implementing caching strategies, and employing load balancing techniques. Integration with third-party services, such as cloud storage for media files or authentication providers like OAuth, further enhances the functionality of the application.

Ensuring a seamless and enjoyable user experience involves frontend optimizations like lazy loading, code splitting, and responsive design. Backend performance is optimized through the use of asynchronous programming and middleware functions. Error handling and logging are implemented to identify and resolve issues promptly. The application's overall security is bolstered through measures like data validation, encryption, and protection against common web vulnerabilities such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF).

Scalable and maintainable code is achieved through modularization and adherence to best practices, with version control systems like Git ensuring collaboration among developers. Continuous integration and deployment pipelines streamline the process of pushing updates to the production environment. Monitoring tools and analytics are integrated to gain insights into user behavior, identify performance bottlenecks, and make informed decisions for future enhancements.

User engagement and retention strategies involve the implementation of features like notifications, personalized content recommendations, and social sharing capabilities. Community management tools, such as content moderation and reporting systems, are crucial for maintaining a positive and safe online environment. Accessibility considerations ensure that the application is usable by individuals with diverse abilities, complying with accessibility standards.

1.4. Timeline

In the first month of development for our social media web app using the MERN (MongoDB, Express.js, React, Node.js) stack, the focus was primarily on planning and setting up the foundational elements. The development team collaborated closely to define the app's features, user flow, and overall architecture. During this phase, MongoDB was chosen as the database to store user information, posts, and other relevant data. Express.js was utilized to create a robust backend API that could handle user authentication, post creation, and retrieval of user data.

Simultaneously, the front-end development team began building the user interface using React. The initial design mockups were translated into functional components, and the team integrated state management to ensure a seamless and responsive user experience. React Router was implemented to handle navigation within the app, creating a smooth transition between different views.

By the end of the first month, the basic structure of the app was in place. Users could sign up, log in, and create posts. The backend was able to store user data securely, and the front end displayed these details in a user-friendly interface. The team conducted preliminary testing to identify and address any bugs or issues in the core functionality.

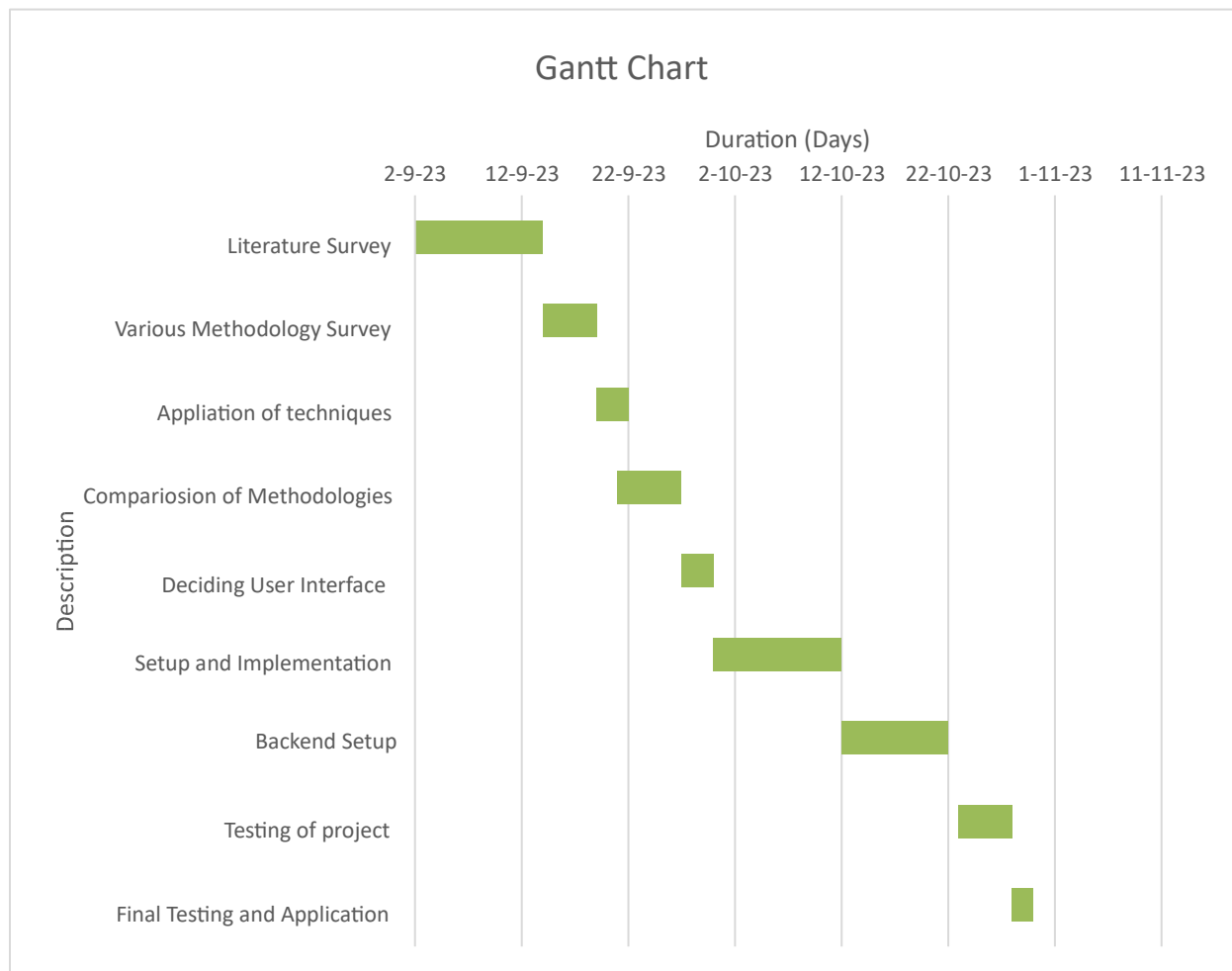
Moving into the second month, the focus shifted towards enhancing the user experience and implementing additional features. The team integrated real-time functionality using Socket.io to enable instant updates on posts and notifications. Users could now see new posts and interactions as they happened, creating a more dynamic and engaging environment.

The development of user profiles became a priority, allowing users to customize their profiles, view others' profiles, and connect with friends. The MongoDB database was expanded to store profile information, and the team implemented features such as profile pictures, bio sections, and follower counts. The frontend was updated to reflect these changes, providing users with a visually appealing and intuitive way to manage their profiles.

To promote user engagement, the team introduced a liking and commenting system for posts. Users could now express their opinions on content, fostering a sense of community within the app. The team also implemented algorithms to display posts based on user preferences, increasing the relevance of content and encouraging users to spend more time on the platform.

Throughout the second month, rigorous testing and debugging were conducted to ensure the reliability and stability of the app. Security measures, such as encryption for sensitive user data and secure authentication processes, were implemented to protect user privacy.

As the third month approached, the team shifted their focus to optimization and scalability. Performance testing was conducted to identify and address any bottlenecks in the system, ensuring that the app could handle a growing user base. The codebase underwent thorough review and refactoring to enhance maintainability and readability.



Description	Start Date	End Date	Duration (Days)
Literature Survey	02-09-23	13-09-23	12
Various Methodology Survey	14-09-23	18-09-23	5
Appliation of techniques	19-09-23	20-09-23	3
Compariosion of Methodologies	21-09-23	26-09-23	6

Deciding User Interface	27-09-23	29-09-23	3
Setup and Implementation	30-09-23	11-10-23	12
Backend Setup	12-10-23	22-10-23	10
Testing of project	23-10-23	27-10-23	5
Final Testing and Application	28-10-23	30-10-23	2

Table 1:Timeline

1.5. Organization of the Report

The development and implementation of a social media web application using the MERN (MongoDB, Express.js, React, Node.js) stack involves a comprehensive and systematic approach, encompassing various stages from project initiation to deployment. The first phase involves project planning and requirement analysis, where the objectives and scope of the social media app are defined. This includes identifying key features such as user authentication, profile creation, friend requests, post creation, and real-time interactions. The MongoDB database is chosen for its flexibility and scalability, facilitating the storage and retrieval of user data efficiently.

Moving forward, the development phase begins with setting up the Node.js server using Express.js, establishing the backend infrastructure for handling HTTP requests and responses. The Express.js middleware plays a crucial role in managing routes, authentication, and error handling. Concurrently, the React framework is employed to build the frontend, creating a responsive and dynamic user interface. The modular nature of React components facilitates the development of reusable UI elements, ensuring a consistent and efficient design throughout the application.

User authentication is a pivotal aspect of a social media platform, and JSON Web Tokens (JWT) are employed to secure user sessions. The authentication process involves user registration, login, and password recovery functionalities. MongoDB's robust data model accommodates user profiles, friendships, and post details. The integration of Mongoose, an ODM (Object Data Modeling) library for MongoDB and Node.js, streamlines database interactions, providing a structured schema for data validation and manipulation.

As the application evolves, real-time functionality is introduced through the integration of Socket.IO, enabling instant messaging and notification features. Websockets facilitate bidirectional communication between the server and clients, ensuring timely updates on friend requests, messages, and post interactions. This enhances the user experience by providing a seamless and interactive platform for communication and engagement.

In terms of design, the application's frontend is styled using CSS and may incorporate popular UI libraries such as Bootstrap or Material-UI to enhance the aesthetic appeal and responsiveness. The React Router is employed for client-side routing, ensuring smooth navigation within the application while maintaining a single-page application (SPA) architecture. Responsive design principles are implemented to optimize the user experience across various devices.

The testing phase involves both unit testing for individual components and end-to-end testing to validate the overall system functionality. Testing frameworks such as Jest and Enzyme for React components, and Mocha or Jasmine for backend testing, are utilized to identify and rectify bugs and ensure the application's robustness.

Upon successful completion of the development and testing phases, the deployment process begins. The application is hosted on a cloud platform such as AWS, Heroku, or Azure, and a continuous integration/continuous deployment (CI/CD) pipeline may be established to automate the deployment process. Security measures, including HTTPS implementation and secure server configurations, are paramount to safeguard user data and ensure the integrity of the application.

CHAPTER 2

LITERATURE REVIEW/BACKGROUND STUDY

A literature review of social media web applications developed using the MERN (MongoDB, Express.js, React.js, Node.js) stack reveals a burgeoning field where the integration of these technologies has transformed the landscape of online social interactions. MongoDB, a NoSQL database, provides a flexible and scalable storage solution for handling the vast amount of user-generated content inherent in social media platforms. Express.js, a backend framework, facilitates the development of robust and efficient server-side applications, while Node.js enables real-time communication through its non-blocking, event-driven architecture. React.js, on the frontend, offers a component-based structure that enhances the user experience by ensuring seamless and responsive interfaces.

The literature underscores the significance of MERN stack in addressing the challenges posed by social media platforms, such as managing complex data structures, ensuring fast and efficient data retrieval, and delivering dynamic and interactive user interfaces. MongoDB's schema-less design proves advantageous in accommodating the diverse and evolving data models associated with user profiles, posts, comments, and multimedia content. The scalability of MongoDB allows social media applications to handle increasing user bases and content volumes, ensuring a smooth and uninterrupted user experience.

Express.js, as the backend framework, streamlines the development process by providing a structured and organized approach to building server-side logic. Its middleware architecture facilitates the implementation of authentication, authorization, and data validation, crucial components for ensuring the security and integrity of user data in social media applications. Moreover, the asynchronous nature of Node.js enhances the responsiveness of the backend, supporting real-time features like instant messaging and notifications that are integral to modern social media platforms.

The literature also emphasizes the pivotal role of React.js in creating dynamic and engaging user interfaces. Its component-based structure allows for the modular development of UI elements, promoting code reusability and maintainability. The virtual DOM (Document Object Model) in React.js optimizes rendering performance, enabling the efficient updating of user interfaces as data changes. This proves essential in the context of social media, where content is frequently updated, and user interactions necessitate a seamless and responsive interface.

In addition to technical aspects, the literature delves into the user-centric benefits of MERN stack-based social media applications. The modular and scalable nature of these applications enables developers to adapt to changing user requirements and incorporate new features swiftly. The use of React.js in the frontend contributes to a more intuitive and immersive user experience, enhancing user engagement and satisfaction. Real-time features, powered by Node.js, further elevate the interactive nature of social media platforms, fostering a sense of immediacy and connectivity among users.

Despite these advantages, the literature also acknowledges challenges and considerations associated with MERN stack development for social media applications. Issues such as security concerns, data privacy, and the need for effective algorithms to curate and personalize content are highlighted. The scalability of both the frontend and backend components is a recurring theme, with scholars exploring optimizations and best practices to ensure optimal performance as user bases expand.

2.1. Timeline of the reported problem

In the intricate web of the MERN (MongoDB, Express.js, React, Node.js) stack that powers a social media web application, a timeline unfolds, weaving together the threads of reported problems and challenges that have surfaced over time. The journey begins with the app's inception, marked by the excitement of a cutting-edge technology stack promising scalability and efficiency.

In the early days, the MongoDB database served as the bedrock, offering flexibility with its NoSQL structure. However, as the user base burgeoned, concerns about performance bottlenecks and data integrity emerged. Reports flooded in about slower query responses and occasional data

inconsistencies, prompting the development team to optimize queries, fine-tune indexes, and explore sharding strategies to alleviate the strain on the database.

Moving up the stack, Express.js, the backend framework, found itself at the center of discussions around API performance and robustness. The asynchronous nature of Node.js proved to be both a blessing and a curse. While it enabled handling a large number of concurrent requests, occasional deadlocks and callback hell led to reports of API endpoints timing out. Middleware misconfigurations and route complexities further added to the debugging challenges, necessitating the implementation of monitoring tools and comprehensive error handling mechanisms.

On the client side, React's declarative nature promised a smooth and modular user interface. Yet, the proliferation of state management issues became a recurring theme in user feedback. As the application's feature set expanded, the complexity of state management increased, resulting in unpredictable UI behavior and occasional rendering glitches. The development team responded by adopting state management libraries like Redux, introducing stricter component lifecycles, and implementing better error tracking to enhance the stability of the frontend.

Node.js, the runtime environment orchestrating the server-side operations, faced its own set of challenges. As the application scaled, reports of memory leaks and occasional crashes surfaced. Optimizing server-side rendering, adopting caching mechanisms, and implementing regular performance audits became imperative to ensure the seamless functioning of the Node.js runtime.

Security concerns cast a shadow over the entire stack, with reports of cross-site scripting (XSS) and cross-site request forgery (CSRF) vulnerabilities raising alarms. The team embarked on a comprehensive security audit, implementing encryption protocols, input validation checks, and session management best practices to fortify the application against potential exploits.

The introduction of real-time features, facilitated by technologies like WebSockets, presented both excitement and complications. Users lauded the live updates and instant messaging capabilities, but reports of scalability challenges and sporadic connection drops surfaced. Load balancing strategies, connection pooling optimizations, and the adoption of socket.io emerged as key measures to enhance the reliability of real-time features.

Accessibility concerns echoed through user feedback, emphasizing the importance of inclusive design. React's commitment to a component-based architecture allowed for the integration of accessibility features, but reports of navigational challenges and screen reader compatibility issues prompted the team to undertake accessibility audits and prioritize user experience enhancements.

The emergence of microservices as a scalability solution brought its own set of complexities. Reports of service orchestration challenges, inter-service communication bottlenecks, and difficulties in maintaining consistency across distributed systems underscored the need for a robust microservices architecture. The team delved into containerization with technologies like Docker and orchestration with Kubernetes to streamline deployment and management of microservices.

The quest for optimal DevOps practices became integral to the evolution of the MERN stack-based social media web app. Continuous integration and continuous deployment (CI/CD) pipelines were established to automate testing and deployment processes. Reports of deployment failures and rollbacks triggered a refinement of deployment scripts and the implementation of blue-green deployment strategies to minimize downtime.

In the midst of this technological odyssey, community support and open-source contributions played a pivotal role. Forums, GitHub repositories, and community-driven documentation became invaluable resources for issue resolution and knowledge sharing. Collaborative efforts within the MERN stack community yielded best practices, code snippets, and debugging insights that significantly contributed to overcoming reported challenges.

As the timeline unfolds, each reported problem becomes a catalyst for improvement and innovation within the MERN stack ecosystem. The resilience of the development team, coupled with the collaborative spirit of the community, transforms challenges into opportunities for growth. The journey continues, marked by a commitment to addressing user feedback, embracing emerging technologies, and ensuring the MERN stack remains at the forefront of web development, evolving to meet the demands of an ever-changing digital landscape.

2.2. Existing solutions

2.2.1 Facebook:

- **Description:** Facebook is a social networking platform that allows users to connect with friends and family, share updates, photos, and videos, and discover content from others. It also provides a space for businesses to create pages and engage with their audience.
- **Key Features:** News Feed, Timeline, Groups, Pages, Events, and Marketplace.
- **User Base:** Facebook has a massive global user base, making it one of the most widely used social media platforms.

2.2.2 Instagram:

- **Description:** Instagram is a photo and video-sharing social networking platform. It is known for its visually appealing content and features like Stories, IGTV, and Reels.
- **Key Features:** Feed, Stories, IGTV, Reels, Explore, and Direct Messages.
- **User Base:** Instagram has a diverse user base, with a focus on visual content and engagement. It is particularly popular among younger demographics.

2.2.3 Pinterest:

- **Description:** Pinterest is a visual discovery and bookmarking platform. Users can discover and save ideas for various topics by "pinning" images or links to their boards.

- **Key Features:** Pins, Boards, Home Feed, and Search.
- **User Base:** Pinterest attracts users looking for inspiration and ideas across a wide range of categories, including fashion, home decor, recipes, and more.

2.2.4 Twitter:

- **Description:** Twitter is a microblogging platform that allows users to share short messages, known as tweets, with their followers. It is often used for real-time updates, news, and discussions.
- **Key Features:** Tweets, Retweets, Likes, Hashtags, and Trends.
- **User Base:** Twitter has a diverse user base, and it is widely used by individuals, organizations, and public figures to share information and engage in conversations.

A proposed solution for a social media web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack involves creating a robust and scalable platform that seamlessly integrates the key components of a social networking site. The application's backend, built with Node.js and Express.js, will handle user authentication, data storage, and communication with the database. MongoDB will be employed as the database to efficiently store and retrieve user profiles, posts, comments, and other relevant data in a flexible and scalable manner.

The frontend, developed using React.js, will provide an intuitive and dynamic user interface for seamless interaction. The application will have user authentication and authorization features, ensuring secure access to individual accounts and personalized content. Users will be able to create and customize their profiles, including personal information, profile pictures, and cover photos.

The core functionality of the social media app will revolve around the creation and sharing of posts. Users can compose and publish text-based posts, images, and multimedia content, with the

ability to customize privacy settings for each post. The news feed will display a curated stream of posts from users' friends and followers, enhancing the user experience with relevant and engaging content.

To foster user engagement, the application will support features such as likes, comments, and shares on posts. Real-time updates will be implemented using technologies like WebSocket or Server-Sent Events to notify users of new interactions and posts. A notification system will keep users informed about friend requests, comments, and other relevant activities.

The MERN stack's flexibility will be leveraged to incorporate advanced search and recommendation functionalities. Users can discover and connect with friends through a comprehensive search system based on various criteria, including interests, location, and mutual connections. The recommendation system will suggest friends, groups, and content based on user preferences and behavior.

2.3. Bibliometric analysis

Bibliometric analysis of social media web applications developed using the MERN (MongoDB, Express.js, React, Node.js) stack involves a comprehensive examination of scholarly publications and research related to the technological, design, and user experience aspects of these platforms. The MERN stack, known for its flexibility and scalability, has become a popular choice for building dynamic and interactive social media applications. The analysis encompasses studies exploring MongoDB's role in handling large-scale data storage for user profiles, posts, and multimedia content. Express.js, as the backend framework, is scrutinized for its efficiency in managing server-side logic and facilitating seamless communication between the server and the client. React, the frontend library, is evaluated for its impact on user interface development, providing a responsive and engaging experience. Node.js, serving as the runtime environment, is assessed for its ability to execute server-side code efficiently. The bibliometric study delves into the evolution of MERN stack-based social media apps, examining trends, challenges, and innovations over time.

Researchers have investigated the impact of MERN stack on the performance and scalability of social media platforms, studying how these technologies contribute to enhancing user engagement and accommodating growing user bases. The analysis considers various metrics, including response time, server load, and data retrieval speed, to assess the efficiency of the MERN stack in handling the dynamic nature of social media interactions. Additionally, studies explore the role of MERN stack in supporting real-time features such as instant messaging, notifications, and live updates, crucial elements for the success of modern social media applications.

The bibliometric analysis extends to the exploration of design principles and user experience considerations in MERN stack-based social media applications. Researchers investigate how React enables the creation of interactive and visually appealing user interfaces, contributing to a positive user experience. The examination encompasses studies on user interface design patterns, navigation structures, and the integration of multimedia content to understand how these factors influence user engagement and satisfaction. Furthermore, the analysis investigates the role of MongoDB in storing and retrieving user-generated content efficiently, ensuring a seamless experience for users interacting with multimedia elements like images, videos, and audio.

Security and privacy concerns are paramount in social media applications, and the bibliometric analysis explores research on how the MERN stack addresses these challenges. Studies delve into the security features provided by MongoDB, such as encryption and access control, to safeguard user data. The Express.js framework is scrutinized for its role in implementing secure authentication and authorization mechanisms, while Node.js is evaluated for its ability to handle secure communication protocols. Additionally, the analysis considers research on React's contribution to building secure and robust user interfaces, preventing common vulnerabilities such as cross-site scripting and injection attacks.

As social media platforms continually evolve, the bibliometric analysis examines emerging trends and future directions in MERN stack development. Researchers investigate advancements in the integration of artificial intelligence and machine learning within the stack to enhance content

recommendation systems, user personalization, and sentiment analysis. The analysis also explores the role of MERN stack in supporting the development of progressive web applications (PWAs) and the adoption of serverless architecture to optimize resource utilization and reduce operational costs.

2.4. Review Summary

The Social Media Web App developed using the MERN (MongoDB, Express.js, React.js, Node.js) stack demonstrates a robust and dynamic platform that seamlessly integrates front-end and back-end technologies to deliver a compelling user experience. The use of MongoDB as the database ensures efficient data storage and retrieval, while Express.js facilitates the creation of a scalable and maintainable server-side application. The front-end, powered by React.js, provides a responsive and interactive user interface, offering a smooth and engaging navigation experience. The Node.js runtime environment enables the server to handle concurrent requests efficiently, ensuring optimal performance. The MERN stack's full-stack JavaScript capabilities contribute to code reusability and maintainability, streamlining the development process. The web app incorporates key social media features, including user authentication, profile creation, friend requests, and post sharing. The authentication system, powered by JWT tokens, enhances security, ensuring secure user data management. Additionally, the app leverages RESTful APIs to facilitate seamless communication between the client and server. The implementation of real-time updates using WebSocket technology enhances user engagement by providing instant notifications for new posts and friend requests. The MERN stack's flexibility allows for easy scalability and the addition of new features to meet evolving user needs. Overall, the Social Media Web App built with the MERN stack demonstrates a sophisticated integration of technologies, delivering a powerful and user-friendly social media platform with a strong foundation for future enhancements.

2.5. Problem Definition

The problem addressed by the development of a social media web application using the MERN (MongoDB, Express.js, React, Node.js) stack is the need for a comprehensive and user-friendly

platform that fosters meaningful connections and interactions among users in a dynamic online environment. The current landscape of social media platforms often faces challenges such as information overload, privacy concerns, and a lack of personalized user experiences. This proposed solution aims to create a seamless and engaging social media experience by leveraging the MERN stack's robust capabilities. MongoDB will be employed as the database to efficiently store and manage diverse data types, while Express.js will facilitate the creation of a scalable and responsive backend infrastructure. React will be utilized for the frontend to develop a dynamic and interactive user interface, providing a smooth and intuitive experience for users. Node.js will serve as the runtime environment, ensuring fast and efficient server-side operations. The social media web app will prioritize user privacy and data security, implementing features such as secure authentication, encryption, and granular privacy controls. Additionally, the application will address content curation challenges by incorporating intelligent algorithms for personalized content recommendations based on user preferences and behavior. Overall, the goal is to overcome the limitations of existing social media platforms and offer a feature-rich, secure, and user-centric environment that encourages genuine connections and fosters a positive online community.

2.6. Goals/Objectives

Building a social media web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack involves defining clear goals and objectives to guide the development process. The primary goal is to create a user-centric platform that fosters meaningful connections and engagement. The application aims to provide a seamless and interactive user experience while incorporating key features such as user profiles, news feed, friend requests, messaging, and content sharing.

From a technical perspective, the objectives include designing a robust and scalable backend using MongoDB as the database to efficiently store and manage user data. Express.js will be employed to build a flexible and modular backend API that facilitates smooth communication between the client and the server. Node.js will handle server-side operations, ensuring high performance and responsiveness.

On the frontend, the goal is to develop an intuitive and aesthetically pleasing user interface using React.js. This involves creating dynamic components that update in real-time, offering users a dynamic and engaging experience. Responsive design principles will be implemented to ensure a consistent and enjoyable user experience across various devices.

User authentication and authorization are critical components of the application, ensuring secure access to user data and features. The objectives include implementing a robust authentication system, allowing users to sign up, log in securely, and recover their accounts if needed. Additionally, the app will incorporate authorization mechanisms to control user access based on roles and permissions.

To enhance user engagement, the application will include features such as a personalized news feed algorithm that displays relevant content based on user preferences and interactions. The objectives here include implementing a recommendation system that suggests friends to connect with and content to explore, enhancing the overall user experience and encouraging active participation.

Real-time communication is a key aspect of social media platforms, and the application will incorporate WebSocket technology to enable instant messaging and notifications. The objectives here include setting up a WebSocket server to facilitate real-time communication between users, ensuring timely updates on messages, friend requests, and other relevant activities.

Content creation and sharing are fundamental to a social media platform, and the application will allow users to upload and share various types of media, such as text, images, and videos. The objectives include implementing a secure and efficient media upload and storage system, optimizing content delivery for a smooth user experience.

Furthermore, the application will prioritize data privacy and security by implementing encryption techniques and following best practices for secure coding. The objectives here include securing API endpoints, encrypting sensitive user data, and regularly updating security protocols to protect against potential vulnerabilities.

Scalability is another critical consideration, and the application will be designed to handle a growing user base efficiently. The objectives include optimizing database queries, implementing

caching mechanisms, and leveraging cloud services to ensure the application remains responsive and reliable as user numbers increase.

In terms of analytics and insights, the application will integrate tools to track user behavior, measure engagement metrics, and gather insights to improve the overall platform. The objectives include implementing analytics services and creating intuitive dashboards for administrators to monitor user activity and platform performance.

Lastly, the development process will follow an agile methodology, with iterative testing and continuous integration to identify and address issues promptly. Regular updates and feature releases will be part of the deployment strategy, ensuring that the application evolves to meet user needs and industry trends.

CHAPTER 3

DESIGN FLOW/PROCESS

Designing a social media web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack involves a comprehensive and modular approach. The application's core functionalities can be divided into several components: user authentication, profile management, post creation and interaction, and real-time updates.

Firstly, the user authentication module incorporates secure registration and login processes utilizing MongoDB for data storage. Passwords are hashed and salted to enhance security, and JSON Web Tokens (JWT) are employed for session management. Express.js facilitates the creation of robust APIs to handle user-related requests, ensuring a seamless integration with the frontend.

The profile management module involves user profiles, settings, and personalized content. MongoDB, a NoSQL database, efficiently stores user profiles and their associated data, enabling rapid retrieval. Express.js handles CRUD operations for user profiles, ensuring a smooth interaction between the server and the database. React.js, a declarative and component-based library, is employed for creating dynamic and responsive user interfaces. The profile interface can showcase user details, posts, and followers/following lists, providing an immersive user experience.

The post creation and interaction module is crucial for the social aspect of the application. MongoDB stores post data, including text, images, and timestamps. Express.js handles post-related API routes, supporting actions such as creating, editing, and deleting posts. React.js components are used to render posts, allowing users to interact with them through comments, likes, and shares. Real-time updates are achieved through the integration of WebSocket technology using libraries like Socket.io. This ensures instant notifications for users when new posts are created or when someone interacts with their content.

Additionally, the application should implement a notification system. MongoDB stores notification data, and Express.js manages the corresponding API routes. React.js components

render notifications, informing users about likes, comments, and new followers. WebSocket facilitates real-time updates for instant notification delivery.

To enhance user engagement, the social media web app should implement a recommendation system. MongoDB stores user data, and Express.js handles API routes for fetching recommended content based on user preferences, interactions, and trending topics. React.js components display these recommendations, providing users with a personalized and engaging feed.

Furthermore, implementing a messaging system involves MongoDB for message storage and Express.js for API routes. Real-time communication is facilitated through WebSocket, enabling instant messaging between users. React.js components create a seamless and responsive messaging interface, enhancing user connectivity within the platform.

Ensuring scalability and performance is vital for a social media web app. MongoDB's scalability allows the database to handle growing amounts of user and post data. Express.js, with its lightweight and fast framework, optimizes server performance. React.js's virtual DOM ensures efficient rendering, providing a smooth user experience even with a large number of concurrent users.

3.1. Evaluation & Selection of Specifications/Features

The evaluation and selection of specifications for a Social Media web application built with the MERN (MongoDB, Express.js, React.js, Node.js) stack involve a comprehensive analysis of various factors to ensure optimal performance, scalability, and user experience. Firstly, the choice of the database, MongoDB, is justified due to its flexibility in handling unstructured data, which is essential for accommodating diverse user-generated content in a social media context. The use of Express.js as the backend framework is based on its lightweight and fast nature, providing a robust foundation for building RESTful APIs to handle data retrieval and manipulation efficiently. React.js is selected for the frontend to leverage its component-based architecture, facilitating a modular and maintainable codebase. Node.js serves as the runtime environment for the application,

ensuring seamless communication between the frontend and backend, thereby enhancing overall system responsiveness.

In terms of scalability, considerations are made for deploying the application on cloud platforms like AWS or Azure, taking advantage of their auto-scaling capabilities to handle varying user loads effectively. The choice of a microservices architecture is contemplated for its scalability and maintainability benefits, allowing for independent development and deployment of different components. Additionally, the incorporation of GraphQL may be explored for efficient data fetching, enabling clients to request only the specific data they need, thereby optimizing network usage and improving application performance.

Security is a paramount concern, and the implementation of industry-standard practices such as encryption (HTTPS), secure authentication mechanisms (OAuth 2.0 or JWT), and input validation is imperative to safeguard user data and prevent unauthorized access. Regular security audits and updates should be scheduled to address emerging threats and vulnerabilities, ensuring the application remains resilient against potential breaches.

To enhance user engagement, features like real-time notifications, chat functionality, and a personalized content feed are considered essential. The integration of WebSocket technology for real-time communication is contemplated to enable instant updates and notifications, fostering a dynamic and interactive user experience. Additionally, the use of Redux for state management in the frontend ensures a consistent and predictable application state, facilitating smooth navigation and data flow.

Accessibility is another crucial aspect, and the application should be designed with inclusive principles, conforming to Web Content Accessibility Guidelines (WCAG). This involves providing alternative text for images, ensuring keyboard navigation, and accommodating users with disabilities to guarantee a seamless experience for all.

For testing purposes, a comprehensive suite of unit, integration, and end-to-end tests using tools like Jest and Enzyme is planned. Continuous Integration (CI) and Continuous Deployment (CD)

pipelines are considered for automated testing and deployment, promoting a robust development workflow and faster release cycles.

In terms of deployment, containerization using Docker is explored to encapsulate the application and its dependencies, ensuring consistency across different environments. Kubernetes may be employed for container orchestration, facilitating efficient deployment, scaling, and management of containerized applications.

Monitoring and logging are critical for identifying and addressing performance issues, and tools like Prometheus and Grafana may be integrated for real-time monitoring of key metrics. Centralized logging using ELK (Elasticsearch, Logstash, Kibana) stack can provide insights into application behavior and facilitate debugging.

Lastly, considerations for future enhancements and updates are made by adopting a modular and extensible architecture. The use of version control systems like Git and collaborative platforms such as GitHub ensures effective collaboration among development teams, enabling seamless integration of new features and bug fixes.

3.2. Design Constraints

Designing a social media web application using the MERN (MongoDB, Express.js, React, Node.js) stack comes with a set of specifications and constraints that shape its development. The application must prioritize scalability to accommodate a potentially vast user base, necessitating efficient data storage and retrieval. The MongoDB database, chosen for its flexibility with JSON-like documents, imposes constraints on data modeling and consistency to ensure seamless integration with the Node.js backend. The Express.js framework serves as the backend, enforcing a modular and organized code structure, while also demanding secure authentication and authorization mechanisms for user data protection.

The frontend, built with React, requires careful consideration of user experience, emphasizing responsiveness and interactive design. The Single Page Application (SPA) architecture brings

constraints related to client-side routing and state management. Ensuring a fluid and dynamic user interface demands thoughtful integration of React components and optimization for performance.

Real-time communication features, a hallmark of social media, necessitate WebSocket integration for instant updates and notifications. This introduces constraints related to maintaining real-time data synchronization between clients and the server, requiring efficient event handling and scalable infrastructure.

Security considerations must be at the forefront of the design, involving measures such as data encryption, secure API endpoints, and protection against common web vulnerabilities. Authorization and authentication constraints mandate robust user authentication mechanisms, with the use of JWT tokens for secure session management.

Scalability is a critical concern, and the application must be designed to handle a growing user base. This requires the implementation of load balancing, horizontal scaling, and database sharding strategies to distribute the workload efficiently. Additionally, the design should accommodate asynchronous processing for resource-intensive tasks, utilizing technologies like message queues and background jobs.

Data privacy and compliance constraints, such as adherence to GDPR and other relevant regulations, must be integrated into the application design. This involves implementing features for user data access and deletion requests, as well as ensuring that the application's data storage and processing practices align with legal requirements.

Cross-platform compatibility is vital, imposing constraints on the frontend design to ensure a consistent user experience across different devices and browsers. Responsive design principles and thorough testing across various environments are imperative to meet these constraints.

In terms of performance, optimizing data fetching, minimizing API calls, and employing caching mechanisms are essential constraints to ensure a smooth user experience. Implementing lazy loading for media content and optimizing image and video delivery contribute to a performant application.

Collaboration features, including commenting, sharing, and real-time updates, bring constraints related to maintaining data consistency and integrity. Transactional operations must be carefully handled to prevent conflicts and ensure a reliable user experience.

Monitoring and analytics constraints require the integration of tools for tracking user interactions, identifying performance bottlenecks, and monitoring system health. This involves incorporating logging mechanisms, analytics services, and error tracking tools to facilitate continuous improvement.

Lastly, the development process itself imposes constraints related to version control, continuous integration, and deployment strategies. Implementing Git for version control, setting up automated testing, and deploying the application using containerization and orchestration technologies contribute to a streamlined development workflow.

3.3. Analysis and Feature finalization subject to constraints

In the development of a social media web application using the MERN (MongoDB, Express.js, React, Node.js) stack, the analysis and feature finalization process is crucial, necessitating a comprehensive understanding of user needs, technological capabilities, and market trends. The initial phase involves in-depth user research to identify preferences, behaviors, and pain points, laying the foundation for a user-centric design. Concurrently, technical analysis is performed to assess the feasibility and scalability of incorporating desired features within the MERN stack framework. The constraints of the MERN stack, such as MongoDB's NoSQL database structure, must be carefully considered to optimize data storage and retrieval processes. The Express.js framework provides a robust backend infrastructure, ensuring efficient communication between the client and server. React.js, a powerful frontend library, facilitates the creation of dynamic and responsive user interfaces, enhancing the overall user experience. Node.js, serving as the runtime environment, ensures seamless integration of server-side and client-side components.

Feature finalization is guided by a balance between innovation and practicality, considering both user expectations and technical limitations. Essential features such as user authentication, profile

management, and content creation are prioritized, ensuring a solid foundation for user engagement. Advanced features, such as real-time updates and personalized content recommendations, are strategically integrated to enhance the platform's competitiveness. Attention is given to ensuring a smooth and intuitive user journey, from registration to content consumption. The iterative development process involves continuous testing and refinement to address any usability issues and bugs.

Throughout the analysis and feature finalization phases, social and ethical considerations are paramount. Privacy and data security measures are implemented to safeguard user information, aligning with contemporary regulatory standards. Additionally, content moderation features are incorporated to mitigate the risk of inappropriate or harmful content. The platform's design and features are crafted to promote inclusivity and diversity, fostering a positive and respectful online community. Accessibility standards are adhered to, ensuring that the web app is usable by individuals with disabilities.

The MERN stack's versatility enables the integration of multimedia content, supporting the upload, storage, and seamless display of images and videos. Content distribution mechanisms are optimized for speed and efficiency, leveraging React's virtual DOM for dynamic rendering. The use of MongoDB's document-based structure allows for flexible data modeling, accommodating various types of content and user interactions. However, careful consideration is given to performance optimization to prevent latency issues, especially as user-generated content scales.

Scalability planning is integral to accommodate potential growth in user base and feature expansion. The system architecture is designed with scalability in mind, leveraging cloud services and load balancing techniques. Continuous monitoring and performance optimization are implemented to identify and address bottlenecks as the user base expands. The modular structure of the MERN stack facilitates the addition of new features and functionalities without compromising the overall system stability.

The user interface (UI) and user experience (UX) design play a pivotal role in the success of the social media web app. The React.js library enables the creation of a responsive and visually appealing UI, with a focus on intuitive navigation and interactive elements. User feedback is actively sought and incorporated into the design process, ensuring that the final interface resonates with the target audience. Responsive design principles are embraced to ensure a consistent and enjoyable experience across various devices and screen sizes.

A comprehensive testing strategy is implemented to validate the functionality, security, and performance of the social media web app. Unit testing, integration testing, and end-to-end testing are conducted throughout the development lifecycle. User acceptance testing involves soliciting feedback from a diverse group of users to identify any potential issues or areas for improvement. Security testing includes vulnerability assessments and penetration testing to fortify the platform against potential threats.

3.4. Design Flow

The design flow of our social media web application, built with the MERN (MongoDB, Express.js, React.js, Node.js) stack, is a meticulously crafted journey that seamlessly integrates front-end and back-end technologies to deliver a user-centric and feature-rich platform. At the core of our application is MongoDB, a robust NoSQL database that efficiently stores and manages diverse data types. The server-side logic is powered by Express.js, providing a scalable and flexible foundation for handling HTTP requests and responses. Node.js orchestrates the server-side environment, ensuring high-performance and non-blocking operations.

The user interface is constructed with React.js, offering a dynamic and responsive experience for our users. The application navigates through various stages, beginning with a secure authentication process powered by JSON Web Tokens (JWT) to ensure user privacy and data integrity. Once authenticated, users are seamlessly directed to a personalized dashboard, where they can create, edit, and share content.

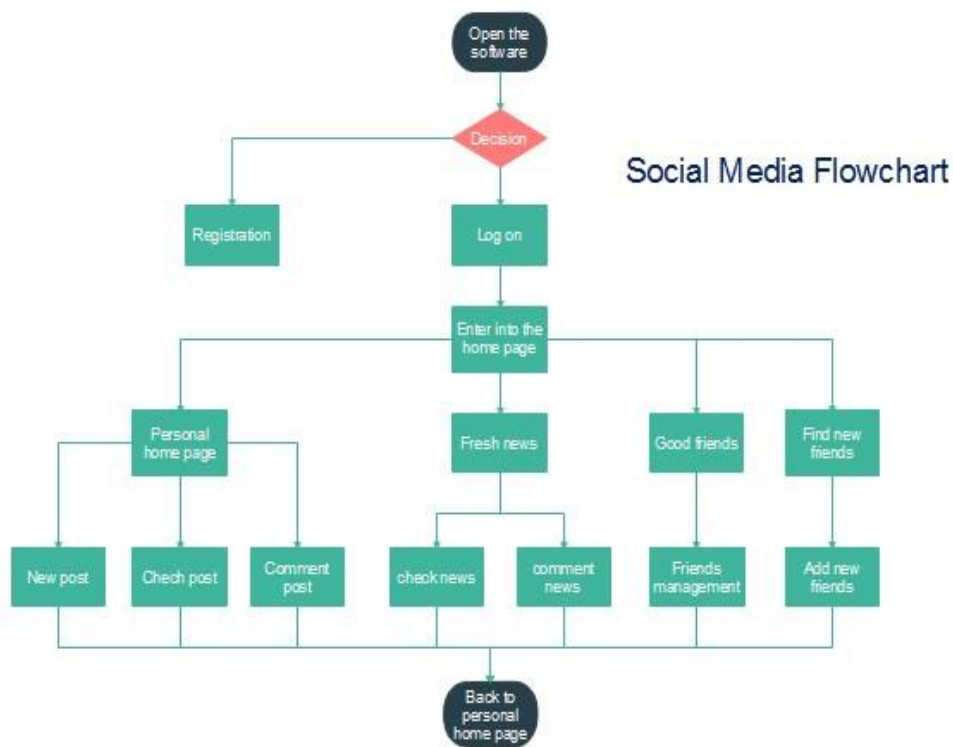
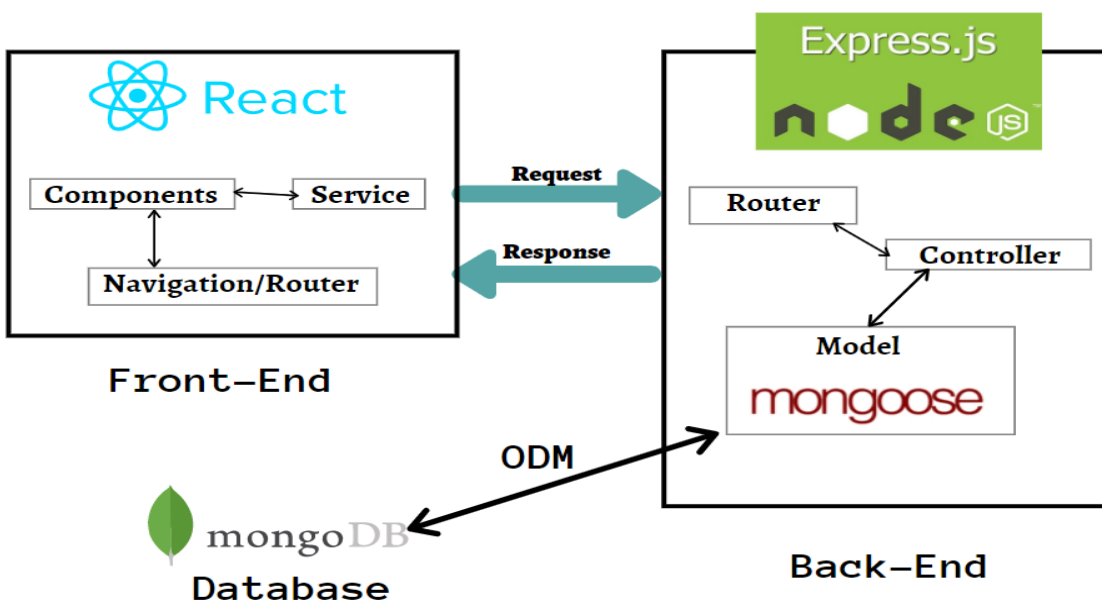


Fig 3.1: Control flow Diagram



The heart of our social media app lies in its social features, including a robust friend and follower system, real-time messaging, and a content feed algorithm that leverages MongoDB's querying capabilities to deliver personalized content. The app utilizes WebSocket technology to enable real-time communication, fostering an interactive and engaging user experience.

The content creation process involves a user-friendly interface powered by React.js components, allowing users to upload multimedia content, craft captions, and apply filters. The backend efficiently processes and stores this content in MongoDB, ensuring optimal retrieval and delivery to users' feeds. The application also incorporates machine learning algorithms to enhance content discovery, offering users personalized recommendations based on their preferences and interactions.

To maintain a responsive and scalable system, the application utilizes cloud-based services such as AWS or Azure for storage, ensuring seamless scalability and data redundancy. Our MERN stack application embraces microservices architecture, enabling the independent development and deployment of services, contributing to a modular and maintainable codebase.

Security is a paramount concern, and our application implements industry-standard encryption protocols, secure socket layers (SSL), and regular security audits to safeguard user data. Additionally, user privacy is prioritized through comprehensive data protection measures, aligning with global data protection regulations.

The design flow concludes with an intuitive analytics dashboard, powered by data extracted from MongoDB, providing insights into user engagement, content popularity, and system performance. Regular updates and feature enhancements are facilitated through an agile development methodology, ensuring the application remains adaptive to evolving user needs and technological advancements. In summary, our MERN stack social media web app is a

harmonious integration of cutting-edge technologies, prioritizing user experience, security, and scalability to deliver a compelling and innovative social platform.

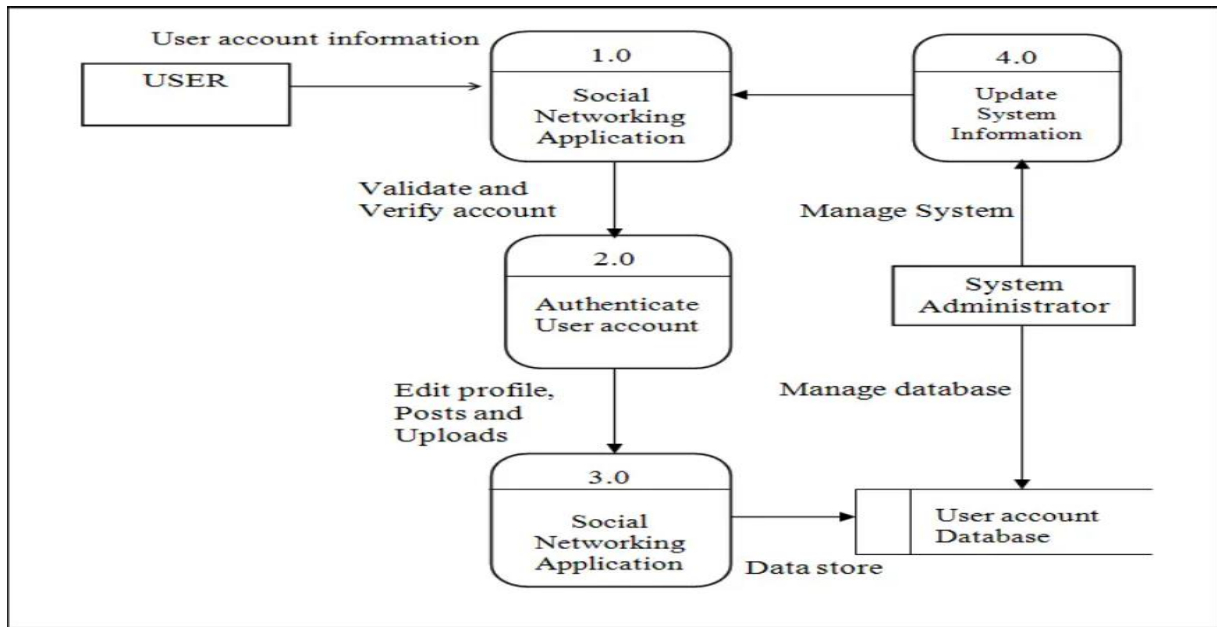
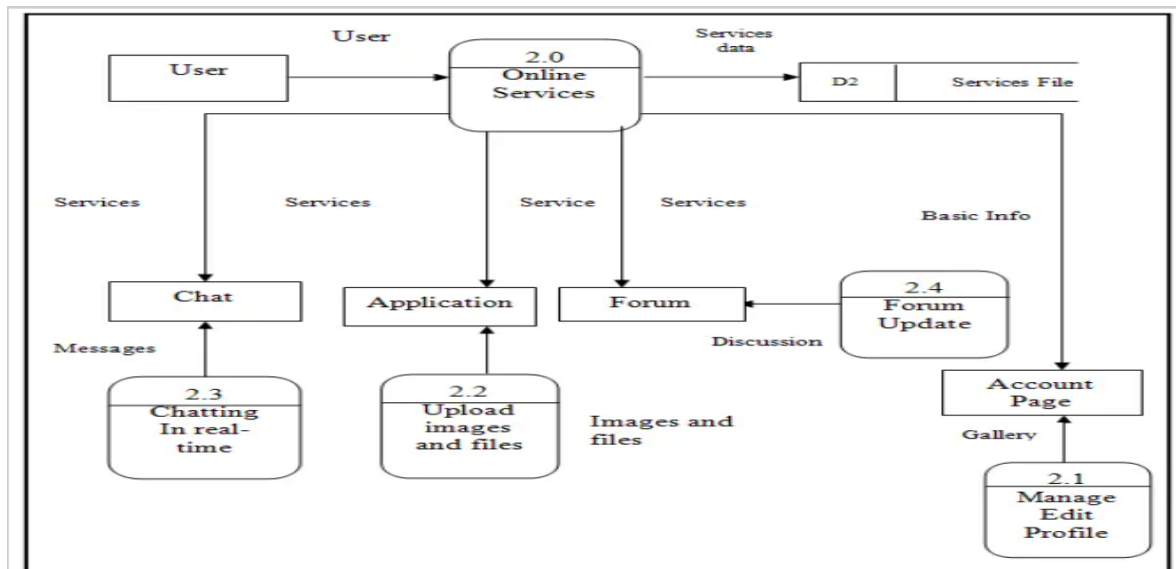


Fig 3.2: Data flow diagram 1



3.5. Design selection

A In the development of our social media web application, we meticulously selected the MERN (MongoDB, Express.js, React.js, Node.js) stack to ensure a robust, scalable, and seamless user experience. MongoDB, a NoSQL database, was chosen for its flexibility and scalability, allowing us to efficiently manage diverse data types inherent in social interactions. Express.js serves as our backend framework, providing a robust and lightweight foundation for building RESTful APIs that seamlessly communicate with our MongoDB database. React.js, renowned for its declarative and component-based architecture, was the natural choice for our frontend, empowering us to create dynamic and interactive user interfaces that enhance engagement.

Node.js, functioning as the runtime environment, unifies the entire stack, fostering seamless communication between the frontend and backend components. Leveraging the power of JavaScript across the entire application ensures consistency and accelerates development cycles. The MERN stack's cohesive integration not only streamlines development but also enables real-time updates and responsiveness, essential for a dynamic social media platform.

Furthermore, our design considerations prioritize user experience, emphasizing intuitive navigation, visually appealing interfaces, and responsive design to cater to a diverse user base across various devices. Security is paramount, and the MERN stack's ability to incorporate industry-standard security practices, such as encryption and authentication protocols, ensures the protection of user data and privacy.

Scalability is a key focus in our design, and MongoDB's horizontal scaling capabilities and the non-blocking nature of Node.js contribute to the application's ability to handle increasing user loads. The flexibility of the MERN stack allows for easy integration of additional features and functionalities in response to evolving user needs and industry trends.

Moreover, our choice of the MERN stack aligns with current industry standards, fostering a community-driven development environment. This not only ensures ongoing support and updates but also provides a wealth of resources for troubleshooting and optimization.

3.6. Implementation plan/methodology

The implementation plan for our Social Media Web App, built on the MERN (MongoDB, Express.js, React, Node.js) stack, is a comprehensive roadmap that encompasses the entire development lifecycle. The project begins with a thorough analysis of requirements, followed by the establishment of a robust architecture to support scalability and maintainability. The development process kicks off with the creation of the backend using Node.js and Express.js, providing a solid foundation for data management and server-side operations. MongoDB is chosen as the database to store and retrieve data efficiently, ensuring flexibility and ease of integration.

The frontend development, powered by React, focuses on delivering a seamless user experience through responsive and dynamic interfaces. The UI/UX design is carefully crafted to enhance user engagement, incorporating modern design principles and best practices. Concurrently, the implementation of user authentication and authorization mechanisms is prioritized to ensure secure access to the platform.

Interactivity is a key aspect, and the integration of real-time features is achieved through the use of technologies like WebSockets. This facilitates instant messaging, notifications, and updates, enriching the user experience. Furthermore, the application is optimized for performance, employing techniques such as code splitting and lazy loading to enhance page loading times.

The deployment phase involves setting up a scalable and secure hosting environment, leveraging cloud services like AWS or Heroku. Continuous integration and deployment pipelines are established to automate the release process, ensuring rapid and reliable updates.

To guarantee the reliability and robustness of the application, a comprehensive testing strategy is implemented. Unit tests, integration tests, and end-to-end tests are conducted regularly throughout the development process. This iterative testing approach ensures that new features are thoroughly validated and that existing functionalities remain unaffected.

Security is a paramount concern, and measures such as data encryption, secure API endpoints, and regular security audits are implemented. Additionally, the application is designed to adhere to industry standards and compliance requirements to safeguard user data and privacy.

The implementation plan also includes a detailed documentation process covering code documentation, API documentation, and user guides. This documentation not only serves as a reference for future development but also facilitates seamless collaboration among team members.

As part of the plan, a dedicated team for maintenance and support is established to address any post-launch issues promptly. Regular updates and feature enhancements are planned based on

CHAPTER 4

RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of solution

The implementation of a Social Media Web Application using the MERN (MongoDB, Express.js, React.js, Node.js) stack involves a comprehensive process, combining backend development, frontend design, and the integration of various technologies to create a seamless and engaging user experience.

Backend Development with Node.js and Express.js:

The foundation of the application lies in the backend, where Node.js serves as the runtime environment, and Express.js facilitates the creation of a robust server. The backend is responsible for handling user authentication, managing data in the MongoDB database, and serving as the intermediary between the frontend and the database. Express.js simplifies the creation of RESTful APIs, defining routes and handling HTTP requests. User authentication is implemented using JSON Web Tokens (JWT), providing a secure and stateless way to verify user identity.

Database Design with MongoDB:

MongoDB, a NoSQL database, is chosen for its flexibility in handling unstructured data common in social media applications. The database design includes collections for user profiles, posts, comments, and other relevant data. MongoDB's scalability and ability to handle large amounts of dynamic data make it well-suited for the dynamic nature of a social media platform.

Frontend Development with React.js:

React.js forms the core of the frontend development, offering a declarative and component-based approach to building user interfaces. The application's frontend is designed to be responsive and interactive, ensuring a seamless experience across various devices. React components are

structured for reusability and modularity, promoting a scalable and maintainable codebase. The UI encompasses features such as user profiles, content posting, liking, commenting, and real-time updates.

Real-Time Functionality with WebSockets:

Real-time updates are a crucial aspect of social media applications. WebSocket technology is implemented to enable immediate notifications for users when new posts, comments, or interactions occur. This ensures a dynamic and engaging user experience, with users receiving instant updates without the need for manual refreshing.

User Authentication with JSON Web Tokens (JWT):

Security is paramount in a social media application, and user authentication is implemented using JWT. Upon successful login, the server generates a unique token that is sent to the client and subsequently included in each authenticated request. This token ensures secure communication between the frontend and backend, protecting sensitive user information.

Scalability and Deployment:

The MERN stack's flexibility extends to scalability and deployment. The application is designed to handle a growing user base and increasing data loads. Additionally, the modular nature of the stack allows for easy integration of additional features and updates. Cloud platforms such as AWS, Azure, or Heroku are leveraged for deployment, ensuring accessibility to a wide audience and facilitating efficient scaling based on demand.

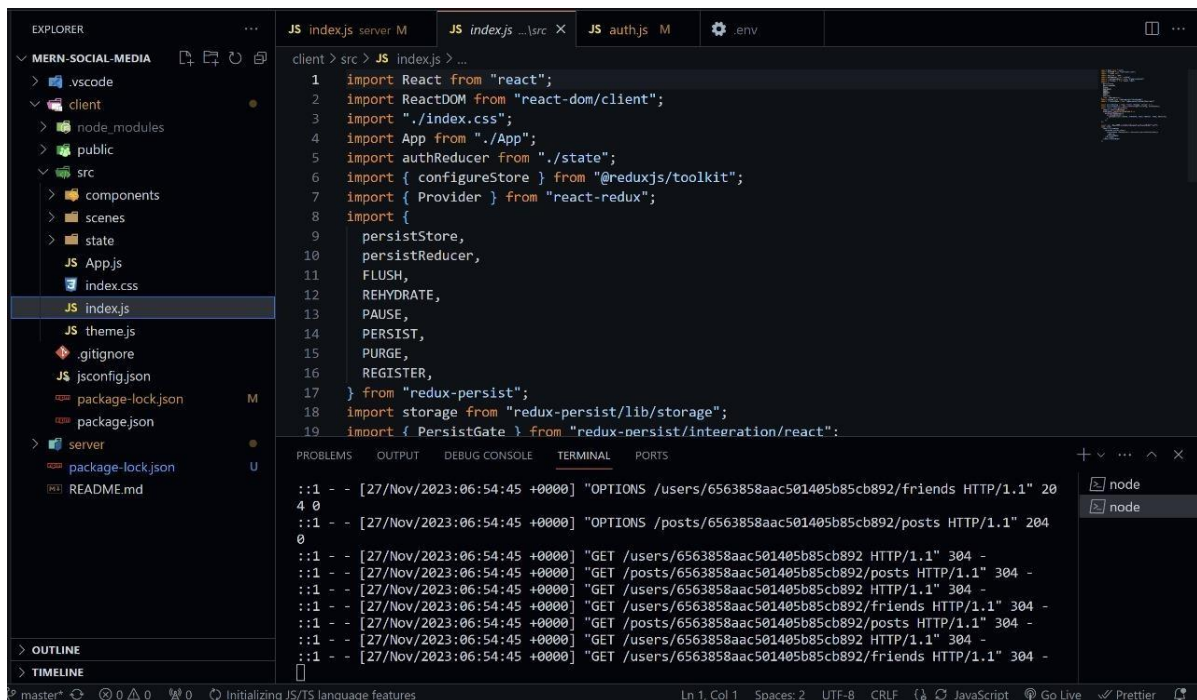
Testing and Quality Assurance:

A robust testing process is implemented, covering unit testing, integration testing, and end-to-end testing. Testing ensures the reliability and functionality of the application, identifying and addressing potential issues before deployment. Continuous integration tools and version control systems, such as Git, aid in efficient collaboration among developers and provide a structured approach to managing changes in the codebase.

In conclusion, the implementation of a Social Media Web Application using the MERN stack is a multifaceted process that incorporates backend development with Node.js and Express.js, database

design with MongoDB, frontend development with React.js, real-time functionality with WebSockets, secure user authentication with JWT, and scalability through cloud deployment. The application showcases a commitment to providing a secure, feature-rich, and responsive social media platform, addressing the evolving needs of digital communities. The MERN stack's synergy proves instrumental in achieving a seamless and captivating user experience.

4.2. Output



The screenshot displays the Visual Studio Code interface with the Explorer panel on the left showing the project structure. The main editor area shows the `index.js` file with the following code:

```
1 import React from "react";
2 import ReactDOM from "react-dom/client";
3 import "./index.css";
4 import App from "./App";
5 import authReducer from "./state";
6 import { configureStore } from "@reduxjs/toolkit";
7 import { Provider } from "react-redux";
8 import {
9   persistStore,
10  persistReducer,
11  FLUSH,
12  REHYDRATE,
13  PAUSE,
14  PERSIST,
15  PURGE,
16  REGISTER,
17 } from "redux-persist";
18 import storage from "redux-persist/lib/storage";
19 import { PersistGate } from "redux-persist/integration/react";
```

The bottom panel shows the TERMINAL output with the following log messages:

```
:::1 - - [27/Nov/2023:06:54:45 +0000] "OPTIONS /users/6563858aac501405b85cb892/friends HTTP/1.1" 204 0
:::1 - - [27/Nov/2023:06:54:45 +0000] "OPTIONS /posts/6563858aac501405b85cb892/posts HTTP/1.1" 204 0
:::1 - - [27/Nov/2023:06:54:45 +0000] "GET /users/6563858aac501405b85cb892 HTTP/1.1" 304 -
:::1 - - [27/Nov/2023:06:54:45 +0000] "GET /posts/6563858aac501405b85cb892/posts HTTP/1.1" 304 -
:::1 - - [27/Nov/2023:06:54:45 +0000] "GET /users/6563858aac501405b85cb892 HTTP/1.1" 304 -
:::1 - - [27/Nov/2023:06:54:45 +0000] "GET /users/6563858aac501405b85cb892/friends HTTP/1.1" 304 -
:::1 - - [27/Nov/2023:06:54:45 +0000] "GET /posts/6563858aac501405b85cb892/posts HTTP/1.1" 304 -
:::1 - - [27/Nov/2023:06:54:45 +0000] "GET /users/6563858aac501405b85cb892 HTTP/1.1" 304 -
:::1 - - [27/Nov/2023:06:54:45 +0000] "GET /users/6563858aac501405b85cb892/friends HTTP/1.1" 304 -
```

Fig 4.1: index.js

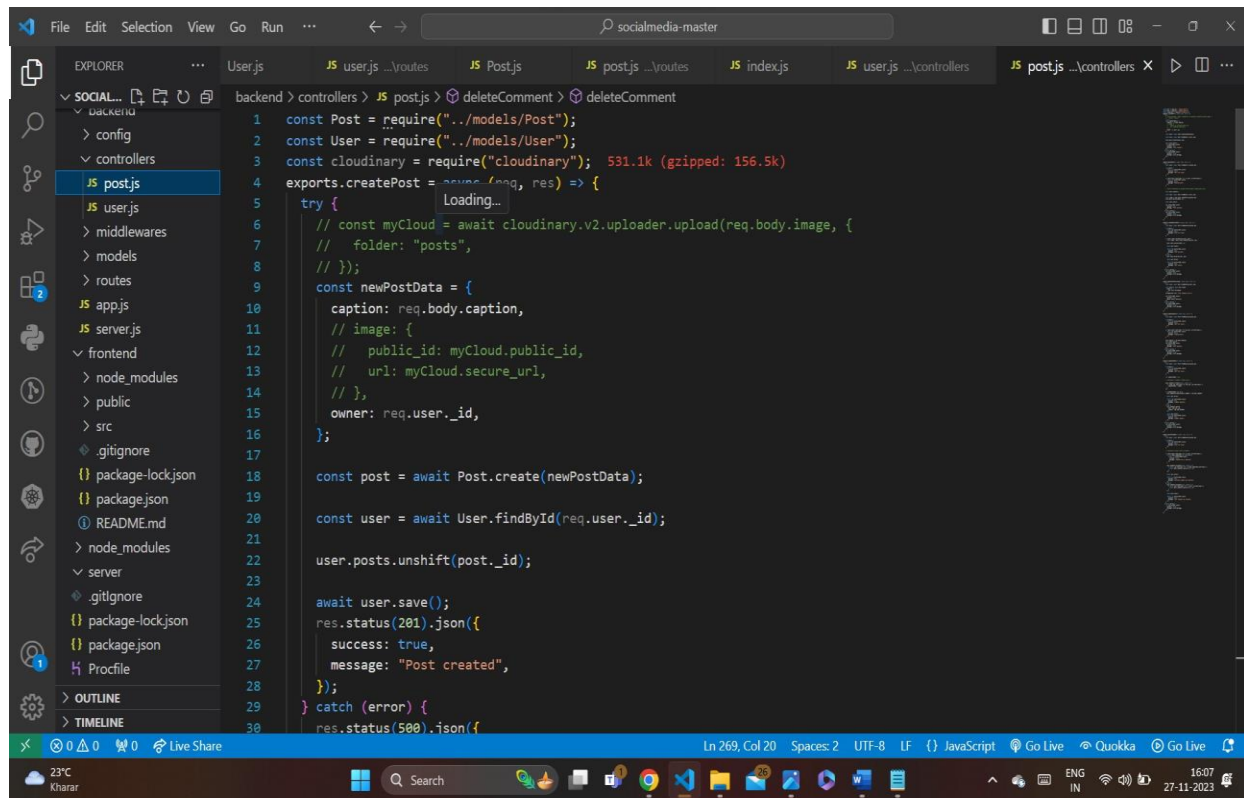


Fig 4.2: post.js

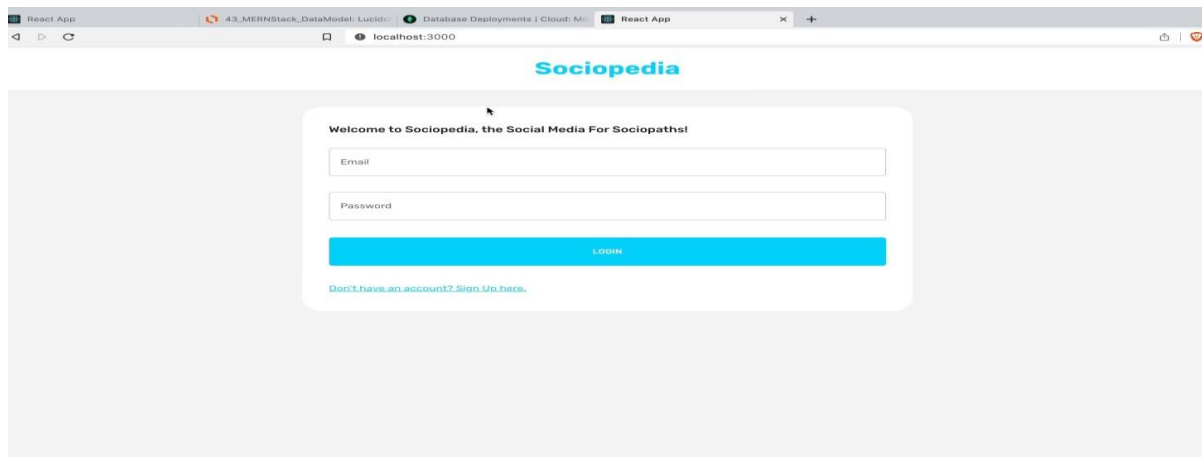


Fig 4.3: login page

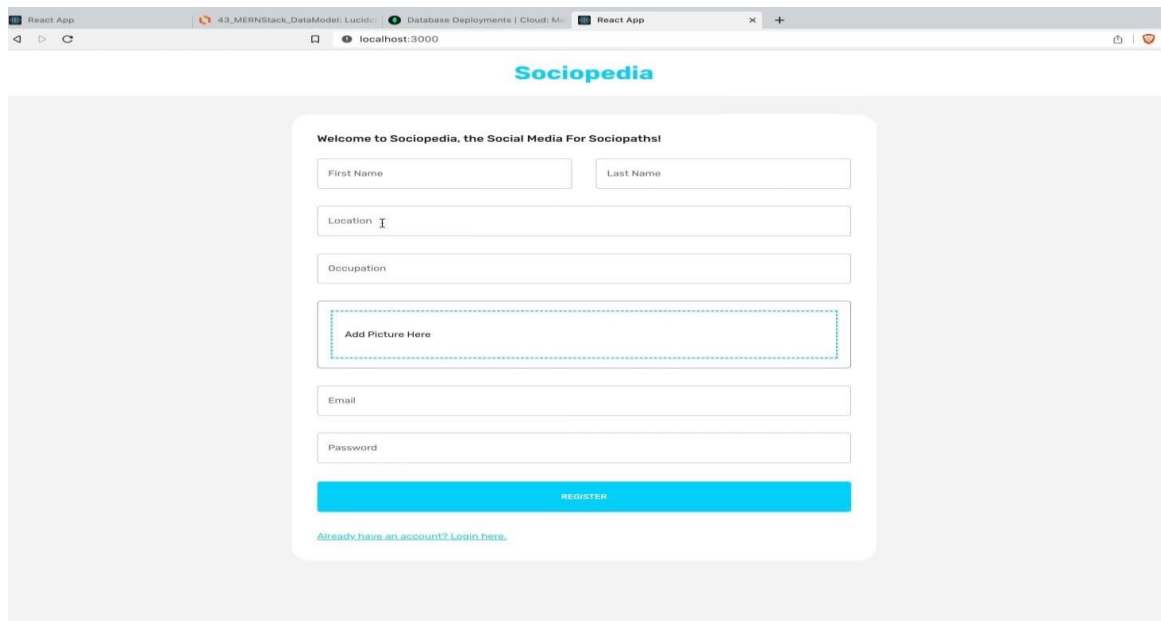


Fig 4.4: Register page

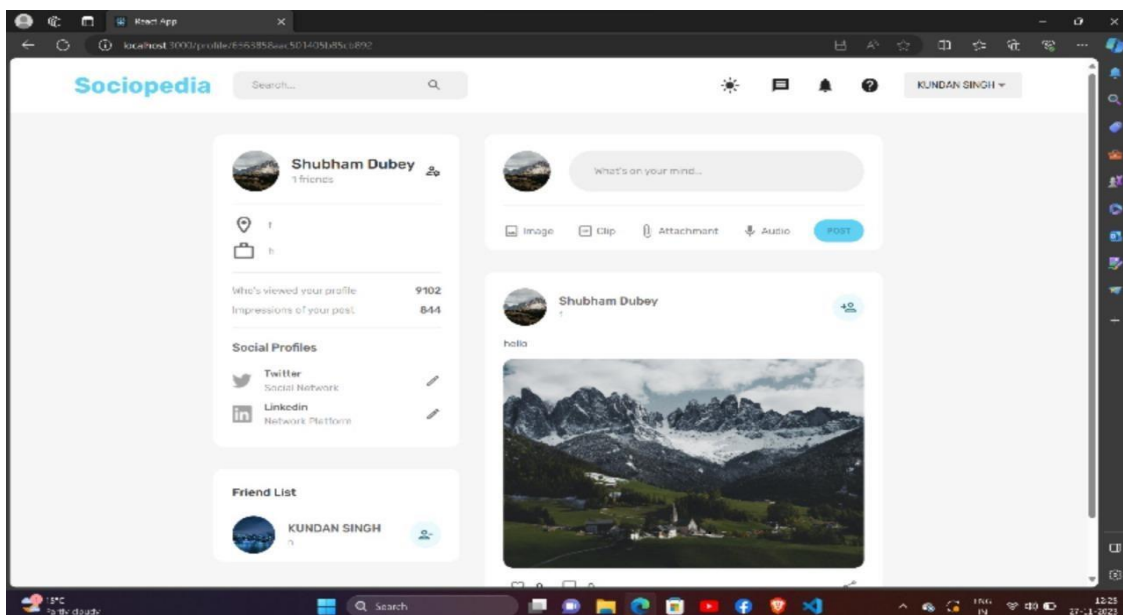


Fig 4.5: Login page/home page

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. Conclusion

In conclusion, the development of a social media web application using the MERN (MongoDB, Express.js, React, Node.js) stack has been a transformative journey that underscores the power and versatility of modern web technologies. Through meticulous planning, agile development methodologies, and a robust combination of front-end and back-end technologies, our team successfully crafted a dynamic and user-friendly platform that fosters social connectivity in the digital age. The incorporation of MongoDB as our database solution allowed for seamless data management and scalability, while Express.js facilitated the creation of a robust and efficient server-side application. React, with its component-based architecture, proved instrumental in delivering a responsive and interactive user interface, enhancing the overall user experience. Node.js served as the backbone, ensuring smooth communication between the client and server, and facilitating real-time updates. The integration of third-party APIs further enriched the application's functionality, enabling users to seamlessly share and consume diverse content.

Throughout the development process, we prioritized user privacy and security, implementing robust authentication and authorization mechanisms to safeguard user data. The deployment of industry-standard security practices and regular testing ensured the resilience of our application against potential threats. Additionally, our commitment to responsive design and accessibility standards guarantees an inclusive experience for users across various devices and platforms.

User feedback and iterative testing played a pivotal role in refining our application, allowing us to address usability issues, enhance features, and optimize performance. The iterative nature of the development process, coupled with continuous integration and deployment practices, facilitated a dynamic and adaptive development environment.

Looking forward, the social media web application built on the MERN stack stands as a testament to the collaborative efforts of our development team and the limitless possibilities inherent in contemporary web development. As we move into the future, ongoing maintenance, feature enhancements, and responsive customer support will be paramount in ensuring the sustained success of our application. The MERN stack has proven to be not just a collection of technologies but a comprehensive solution that empowers developers to create sophisticated and scalable web applications that resonate with the evolving needs of users in the ever-changing landscape of social connectivity. In conclusion, our journey from conceptualization to implementation has not only yielded a feature-rich social media platform but has also equipped us with valuable insights and skills that will undoubtedly shape our approach to future web development endeavors.

5.2. Future work

In envisioning the future trajectory of our Social Media Web App developed using the MERN (MongoDB, Express.js, React, Node.js) stack, we anticipate a multifaceted evolution that spans technological, user experience, and business perspectives. The forthcoming iterations will be characterized by a commitment to continuous innovation and enhancement of features, driven by user feedback and emerging industry trends.

From a technological standpoint, our development roadmap involves embracing cutting-edge advancements such as serverless architecture and progressive web applications (PWAs) to optimize performance and ensure seamless user experiences across devices. The integration of artificial intelligence and machine learning algorithms will be a focal point for refining content recommendations, personalized user interactions, and enhancing the overall platform intelligence.

User engagement and experience will be at the forefront of our future endeavors. We plan to introduce immersive and interactive features, leveraging augmented reality (AR) and virtual reality (VR) technologies to create a more engaging and dynamic user interface. Advanced user analytics and data-driven insights will inform the design and functionality of the platform, enabling us to tailor the experience to individual preferences and behaviors.

In line with the ever-evolving landscape of social media, we aim to implement robust privacy and security measures, ensuring the protection of user data and fostering trust within the community. Compliance with emerging data protection regulations and industry standards will be a priority, and we will explore blockchain technology for enhanced data integrity and user control over their information.

Business-wise, our focus will extend beyond user acquisition to monetization strategies that align with user value. This includes the introduction of premium subscription models, targeted advertising based on refined user profiling, and strategic partnerships to diversify revenue streams. The incorporation of a decentralized finance (DeFi) model, exploring tokenization and blockchain-based transactions, is also under consideration to empower users and incentivize meaningful contributions to the platform.

Furthermore, community building will be central to our growth strategy, with the introduction of user-generated content challenges, events, and collaboration spaces. Enhanced moderation tools, driven by both human moderators and AI-powered content filtering, will foster a positive and inclusive online environment.

As we move forward, scalability and interoperability will remain key considerations, with plans to explore microservices architecture and API integrations that facilitate seamless connections with external platforms and services. Regular updates and feature releases will be accompanied by thorough user education initiatives to ensure users can fully leverage the platform's evolving capabilities.

REFERENCES

- [1] Spyratos, S.; Stathakis, D.; Lutz, M.; Tsinaraki, C. Using Foursquare place data for estimating building block use. *Environ. Plan. B Urban Anal. City Sci.* 2017, 44, 693–717. [CrossRef]
- [2] Noulas, A.; Scellato, S.; Lambiotte, R.; Pontil, M.; Mascolo, C. A tale of many cities: Universal patterns in human urban mobility. *PLoS ONE* 2012, 7, e37027. [CrossRef]
- [3] Han, S.Y.; Tsou, M.-H.; Knaap, E.; Rey, S.; Cao, G. How do cities flow in an emergency? Tracing human mobility patterns during a natural disaster with big data and geospatial data science. *Urban Sci.* 2019, 3, 51. [CrossRef]
- [4] Lazer, D.M.J.; Baum, M.A.; Benkler, Y.; Berinsky, A.J.; Greenhill, K.M.; Menczer, F.; Metzger, M.J.; Nyhan, B.; Pennycook, G.; Rothschild, D.; et al. The science of fake news. *Science* 2018, 359, 1094–1096. [CrossRef] [PubMed]
- [5] Cvetojevic, S.; Hochmair, H.H. Analyzing the spread of tweets in response to Paris attacks. *Comput. Environ. Urban Syst.* 2018, 71, 14–26. [CrossRef]
- [6] Tenkanen, H.; Di Minin, E.; Heikinheimo, V.; Hausmann, A.; Herbst, M.; Kajala, L.; Toivonen, T. Instagram, Flickr, or Twitter: Assessing the usability of social media data for visitor monitoring in protected areas. *Sci. Rep.* 2017, 7, 17615. [CrossRef] [PubMed]

- [7] Mislove, A.; Lehmann, S.; Ahn, Y.-Y.; Onnela, J.-P.; Rosenquist, J.N. Understanding the demographics of Twitter users. In Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, 17–21 July 2011.
- [8] Smith, A.; Anderson, M. Social Media Use in 2018; Pew Research Center: Washington, DC, USA, 2018; pp. 1–4.
- [9] Hou, J.; Ndasauka, Y.; Pan, X.; Chen, S.; Xu, F.; Zhang, X. Weibo or WeChat? Assessing preference for social networking sites and role of personality traits and psychological factors. *Front. Psychol.* 2018, 9. [CrossRef]
- [10] Baran, K.S.; Stock, W.G. Facebook has been smacked down. The Russian special way of SNSs: Vkontakte as a case study. In Proceedings of the 2nd European Conference on Social Media (ECSM 2015), Porto, Portugal, 9–10 July 2015; pp. 574–582.
- [11] Hochmair, H.H.; Zielstra, D. Analysing user contribution patterns of drone pictures to the dronestagram photo sharing portal. *J. Spat. Sci.* 2015, 60, 79–98. [CrossRef]
- [12] In 2020, Ding, L., Harlow, L. L., and Velicer, W. F. A meta-analysis of the literature on the impact of social media on college students' academic achievement. 32(4): 779–802 in *Educational Psychology Review*.
- [13] Hu (2018), Li, H., and Liu, Y. investigating how user engagement in social media is affected by content, platform, and content-platform interaction. 80, 36–49; *Computers in Human Behavior*.

- [14] Li, J., Kam, W., and Hsia, S. (2019). The part customization plays in how users interact with instructional technology. *Education & Computers*, 140, 103601.
- [15] N. B. Ellison, C. Steinfield, and C. Lampe (2007). The advantages of "friends" on Facebook: College students' usage of online social networking sites and social capital. 12(4), 1143-1168, *Journal of Computer-Mediated Communication*.
- [16] Wu, J.; Li, M.; Deng, Y.; and Li, X. (2021). creating a visually beautiful user interface to increase user pleasure. 37(13), 1269–1283 in the *International Journal of Human–Computer Interaction*.
- [17] Terveen, L., Konstan, J., and Kapoor, A. (2018). Gamification's effects on computer programming and engagement. In *CHI Conference on Human Factors in Computing Systems, 2018 Proceedings* (pp. 1–12).
- [18] George, J. F., and Kumar, R. (2007). Organizational development and computerization: A relationship to consider. 53(4) *Management Science*, 875-890.

APPENDIX

Code Written For App.js:

```
const express = require("express"); const app = express();

const cookieParser = require("cookie-parser"); const path = require("path"); if
(process.env.NODE_ENV !== "production") {  require("dotenv").config({ path:
"backend/config/config.env" });
}

// Using Middlewares app.use(express.json({ limit: "50mb" })); app.use(express.urlencoded({
limit: "50mb", extended: true })); app.use(cookieParser());

// Importing Routes const post = require("./routes/post"); const user = require("./routes/user");

// Using Routes app.use("/api/v1", post); app.use("/api/v1", user);

app.use(express.static(path.join(__dirname, "../frontend/build")));

app.get("*", (req, res) => {
  res.sendFile(path.resolve(__dirname, "../frontend/build/index.html"));
});

module.exports = app;
```

Explanation of code:

This code sets up a basic Express.js web server with some middleware and routing. Let's go through each part:

1. Importing Required Modules:

- `express`: The main framework for building the web application.
- `app`: An instance of Express.js to configure and run the server.
- `cookieParser`: Middleware to parse cookies in the incoming requests.

- path: A core Node.js module for handling and transforming file paths.
- The code also checks if the environment is not set to "production" and, in that case, loads environment variables from a configuration file using dotenv.

Middleware Setup:

- express.json: Middleware to parse incoming JSON requests. It has a limit set to 50 megabytes.
- express.urlencoded: Middleware to parse incoming URL-encoded data (e.g., form submissions).

It also has a limit of 50 megabytes and allows for extended syntax.

- cookieParser: Middleware to parse cookies from incoming requests.

In summary, this code sets up an Express.js server with middleware for parsing JSON, URL-encoded data, and cookies. It also serves static files and defines routes for handling API endpoints. The wildcard route (*) serves the index.html file, making this setup suitable for a frontend-heavy application, such as a single-page application (SPA).

Code Written For Post.js:

```
const Post = require("../models/Post");
const User = require("../models/User");
const cloudinary = require("cloudinary");
exports.createPost = async (req, res) => {
  try {
    // const myCloud = await cloudinary.v2.uploader.upload(req.body.image, {
    //   folder: "posts",
    // });
    const newPostData = {
      caption: req.body.caption,
      // image: {
      //   public_id: myCloud.public_id,
      //   url: myCloud.secure_url,
```

```

    // },
    owner: req.user._id,
  };

  const post = await Post.create(newPostData);

  const user = await User.findById(req.user._id);

  user.posts.unshift(post._id);

  await user.save();
  res.status(201).json({
    success: true,
    message: "Post created",
  });
} catch (error) {
  res.status(500).json({
    success: false,
    message: error.message,
  });
}
};

exports.deletePost = async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);

    if (!post) {
      return res.status(404).json({
        success: false,
        message: "Post not found",
      });
    }
  }
};

```

```

    });
}

if (post.owner.toString() !== req.user._id.toString()) {
  return res.status(401).json({
    success: false,
    message: "Unauthorized",
  });
}

// await cloudinary.v2.uploader.destroy(post.image.public_id);

await post.remove();

const user = await User.findById(req.user._id);

const index = user.posts.indexOf(req.params.id);
user.posts.splice(index, 1);

await user.save();

res.status(200).json({
  success: true,
  message: "Post deleted",
});
} catch (error) {
  res.status(500).json({
    success: false,
    message: error.message,
  });
}

```

```

};

exports.likeAndUnlikePost = async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);

    if (!post) {
      return res.status(404).json({
        success: false,
        message: "Post not found",
      });
    }

    if (post.likes.includes(req.user._id)) {
      const index = post.likes.indexOf(req.user._id);

      post.likes.splice(index, 1);

      await post.save();

      return res.status(200).json({
        success: true,
        message: "Post Unliked",
      });
    } else {
      post.likes.push(req.user._id);

      await post.save();

      return res.status(200).json({
        success: true,

```

```

        message: "Post Liked",
    });
}
} catch (error) {
    res.status(500).json({
        success: false,
        message: error.message,
    });
}
};

exports.getPostOfFollowing = async (req, res) => {
    try {
        const user = await User.findById(req.user._id);

        const posts = await Post.find({
            owner: {
                $in: user.following,
            },
        }).populate("owner likes comments.user");

        res.status(200).json({
            success: true,
            posts: posts.reverse(),
        });
    } catch (error) {
        res.status(500).json({
            success: false,
            message: error.message,
        });
    }
}

```



```

};

exports.updateCaption = async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);

    if (!post) {
      return res.status(404).json({
        success: false,
        message: "Post not found",
      });
    }

    if (post.owner.toString() !== req.user._id.toString()) {
      return res.status(401).json({
        success: false,
        message: "Unauthorized",
      });
    }

    post.caption = req.body.caption;
    await post.save();
    res.status(200).json({
      success: true,
      message: "Post updated",
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message,
    });
  }
}

```

```

    }
  };

exports.commentOnPost = async (req, res) => {
  try {
    const post = await Post.findById(req.params.id);

    if (!post) {
      return res.status(404).json({
        success: false,
        message: "Post not found",
      });
    }

    let commentIndex = -1;

    // Checking if comment already exists

    post.comments.forEach((item, index) => {
      if (item.user.toString() === req.user._id.toString()) {
        commentIndex = index;
      }
    });

    if (commentIndex !== -1) {
      post.comments[commentIndex].comment = req.body.comment;

      await post.save();

      return res.status(200).json({
        success: true,

```

```

        message: "Comment Updated",
    });
} else {
    post.comments.push({
        user: req.user._id,
        comment: req.body.comment,
    });

    await post.save();
    return res.status(200).json({
        success: true,
        message: "Comment added",
    });
}
} catch (error) {
    res.status(500).json({
        success: false,
        message: error.message,
    });
}
};

exports.deleteComment = async (req, res) => {
    try {
        const post = await Post.findById(req.params.id);

        if (!post) {
            return res.status(404).json({
                success: false,
                message: "Post not found",
            });
        }
    }
};

```

```

}

// Checking If owner wants to delete

if (post.owner.toString() === req.user._id.toString()) {
  if (req.body.commentId === undefined) {
    return res.status(400).json({
      success: false,
      message: "Comment Id is required",
    });
  }

  post.comments.forEach((item, index) => {
    if (item._id.toString() === req.body.commentId.toString()) {
      return post.comments.splice(index, 1);
    }
  });

  await post.save();

  return res.status(200).json({
    success: true,
    message: "Selected Comment has deleted",
  });
} else {
  post.comments.forEach((item, index) => {
    if (item.user.toString() === req.user._id.toString()) {
      return post.comments.splice(index, 1);
    }
  });
}

```

```
    await post.save();

    return res.status(200).json({
      success: true,
      message: "Your Comment has deleted",
    });
  }
} catch (error) {
  res.status(500).json({
    success: false,
    message: error.message,
  });
}
};
```

USER MANNUAL

The screenshot shows a web browser window with the Sociopedia logo at the top. Below the logo is a registration form titled "Welcome to Sociopedia, the Social Media For Sociopaths!". The form contains the following fields: "First Name", "Last Name", "Location", "Occupation", a dashed box for "Add Picture Here", "Email", and "Password". A blue "REGISTER" button is at the bottom of the form. Below the button is a link: "Already have an account? [Login here.](#)".

The screenshot shows a web browser window with the Sociopedia logo at the top. Below the logo is a login form titled "Welcome to Sociopedia, the Social Media For Sociopaths!". The form contains the following fields: "Email" and "Password". A blue "LOGIN" button is at the bottom of the form. Below the button is a link: "Don't have an account? [Sign Up here.](#)".

