FULL STACK RESPONSIVE SOCIAL MEDIA APPLICATION USING THE MERN STACK

Project report submitted in partial fulfilment of the requirement for the degree

of Bachelor of Technology

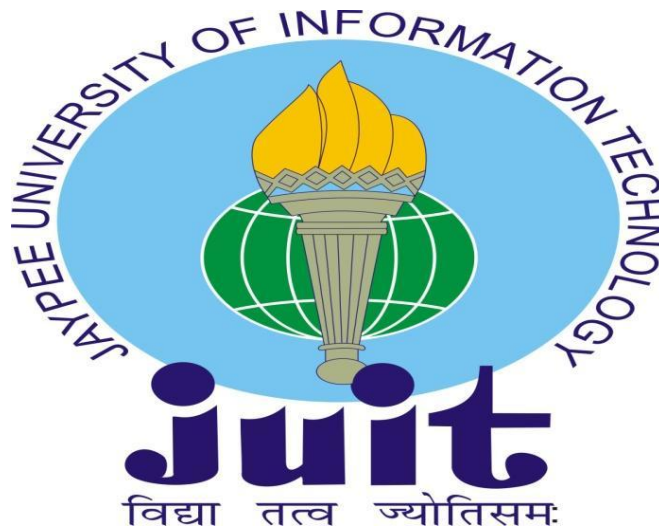in

Computer Science and Engineering

By

Priyansh Khatri (191268)

Under the supervision of

Dr. Amit Srivastava and Dr. Vipul Sharma

to

Department of Computer Science & Engineering and Information Technology

**Jaypee University of Information Technology, Waknaghat,
Solan-173234, Himachal Pradesh**

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Full Stack Responsive Social Media Application Using The MERN Stack"** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering submitted to the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from January 2023 to May 2023 under the supervision of Dr. Amit Srivastava (Associate Professor and Head, Department of Humanities and Social Sciences) and Dr. Vipul Sharma (Assistant Professor (SG), Department of Computer Science & Engineering and Information Technology).

I also authenticate that I have carried out the above-mentioned project work under the proficiency stream Data Science. The matter embodied in this report has not been submitted for the award of any other degree or diploma.

Priyansh Khatri

191268

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Amit Srivastava   Dr. Vipul Sharma

Associate Professor and HoD   Assistant Professor

Department of HSS   Department of CS and IT

Dated: Dated:

Plagiarism Certificate

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): | PhD Thesis | | M.Tech Dissertation/ Report | | B.Tech Project Report | | Paper |

Name: _____ __Department: _____ Enrolment No _____

Contact No. _____E-mail. _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:
- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor)   **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by

Name & Signature   **Librarian**

....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ii

iii

Acknowledgment

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing in making it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to supervisors Dr. Amit Srivastava, Associate Professor, and Head, Department of Humanities and Social Sciences, and Dr. Vipul Sharma, Assistant Professor (SG), Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat. The deep knowledge & keen interest of my supervisors in the field of "Web Development" to carry out this project has helped me out a lot. Their endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to Dr. Amit Srivastava and Dr. Vipul Sharma for their kind help to finish my project. I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I would like to express my gratitude to all of the staff members, instructional and non-instructional, who have made my task easier by providing handy assistance.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Priyansh Khatri

191268

Table of Contents

List of Abbreviations

1. MERN - MongoDB, Express.Js, React, Node.Js
2. MEAN - MongoDB, Express.Js, Angular, Node.Js
3. HTML - HyperText Markup Language
4. CSS - Cascading Style Sheets
5. JS - Javascript
6. NPM - Node Package Manager
7. HTTP - HyperText Transfer Protocol
8. API - Application Programming Interface
9. JSON - Javascript Object Notation
10. JWTs - JSON Web Tokens
11. LAMP - Linux, Apache, MySQL, PHP
12. XML - Extensible Markup Language
13. JSX - Javascript XML
14. UI - User Interface
15. UX - User Experience
16. IBEF - India Brand Equity Foundation
17. AR - Augmented Reality
18. VR - Virtual Reality
19. VS Code - Visual Studio Code

List of Figures

List of Graphs

Abstract

Over the past decade, social media has fundamentally altered how people connect and communicate with one another and has become an integral part of our daily lives. Approximately 692 million people in India use social media, with platforms like WhatsApp, YouTube, Instagram, LinkedIn, Snapchat and Facebook dominating usage. At the beginning of 2023, there were approximately 398 million people in India who were 18 years of age or older who were active on social media, which is equal to 40.2 percent of the country's entire population of adults. In general, 67.5 percent of all internet users in India in January 2023 (regardless of age) used at least one social networking platform. [1]

Social media has had an impact on more than just interpersonal communication; it has also changed how individuals engage in politics, business, and even social activism. Social media platforms have given people and businesses the chance to rapidly interact with a larger audience and spread their messages. According to a research from the IBEF in 2020, social media is a potent tool for reaching people all throughout the country because India has the second-highest number of social media users worldwide, after China. In 2023, a study published by Hootsuite reported that there are over 4.74 billion social media users worldwide, making social media a powerful tool for reaching people across the globe. [2]

The Social Media application that we have developed uses the MERN stack, which consists of MongoDB, Express.js, React, and Node.js. This Full stack Responsive Social Media App with Authentication and Authorization, Posts, Likes and Comments, and Dark Mode is an excellent example of how social media applications can benefit from modern development technologies.

This application has a responsive layout that changes according to the size of the screen. Users may sign up, log in, and log out securely thanks to the application's user authentication and authorization features. One of its distinguishing characteristics is the option for dark mode, which users may toggle on and off. The application also has a search feature that lets users look for specific postings by title or body content.

Overall, the Full stack Responsive Social Media App with Auth, Likes, Dark Mode is a fully functional web application that demonstrates the power and flexibility of the MERN stack. The inclusion of user authentication and authorization, as well as the ability to like and comment on posts, makes it a fully featured social media platform.

Chapter 01: INTRODUCTION

1.1 Introduction

Over the last fifteen years, Social media has grown quickly and has assimilated into our daily lives. The increasing use of social media has altered how we interact with one another, do business, and even perceive our surroundings. People now frequently share their ideas, viewpoints, and experiences with others on sites like Facebook, Twitter, Instagram, Youtube, and LinkedIn. According to Statista, in 2023, there were 4.74 billion active social media users worldwide, with this number expected to grow to 6 billion by 2027. [3]

The world has greatly benefited from social media. The internet has made it possible for people to communicate with others all over the world, erasing geographical boundaries and fostering a sense of global community. Businesses may now reach out to potential clients and customers and better target their marketing campaigns thanks to social media. The use of social media in political campaigns, activism, and social movements is proof of the influence it has in influencing public opinion.

In conclusion, social media has developed quickly and taken over our lives. The ability for individuals to interact and communicate with one another, as well as the facilitation of the expansion of businesses and economies, has had a huge impact on the world. The advantages of social media outweigh the disadvantages, notwithstanding the spread of bogus news and detrimental effects on mental health. It is crucial that individuals and society as a whole comprehend social media's potential and take action to lessen its negative consequences as it continues to develop.

Let us look at some statistics that depict how social media has grown over the last two decades -
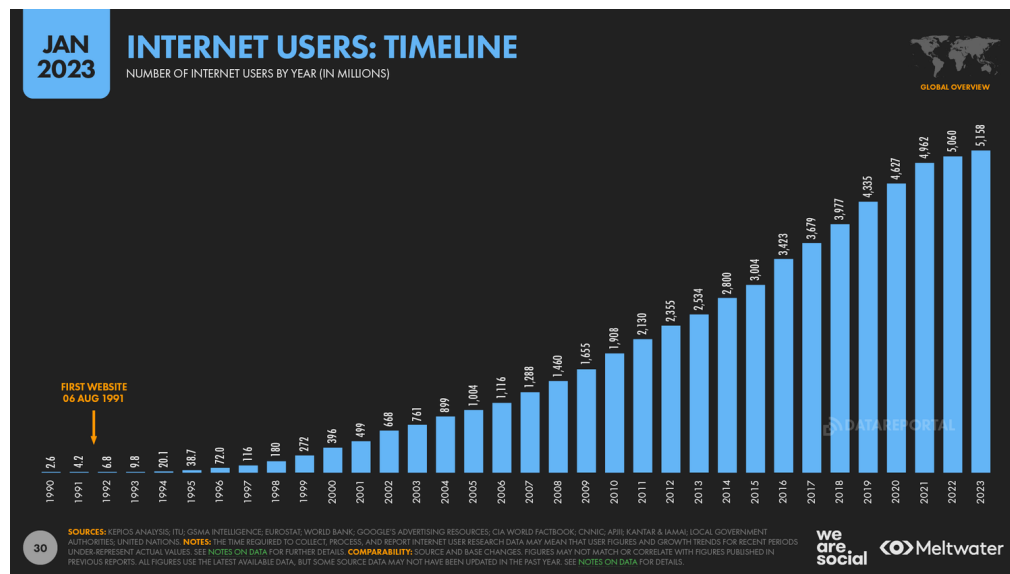
Graph 1 : The growth of social media users in the last two decades. [4]
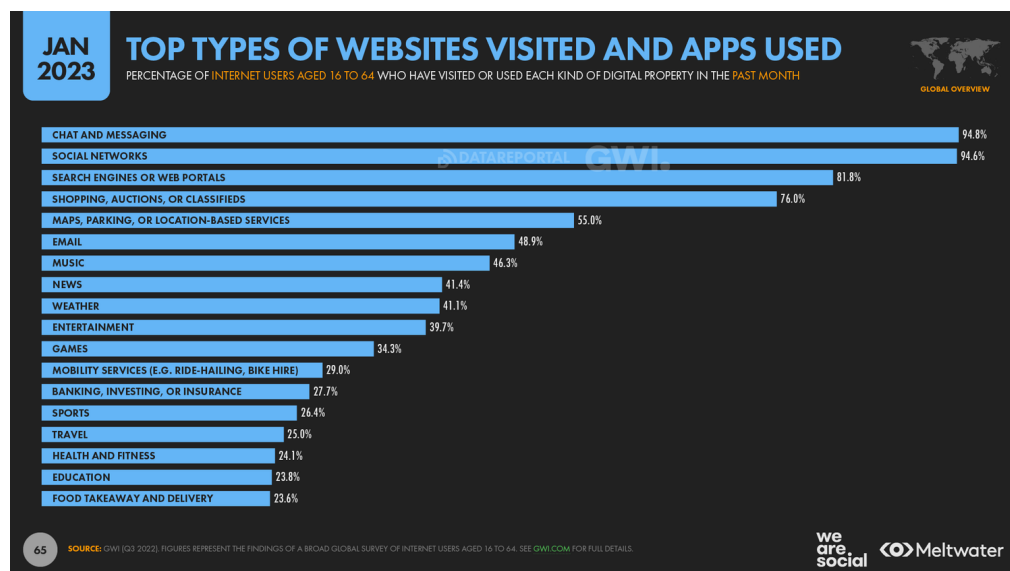
Graph 2 : Top types of websites visited and web applications used. [4]
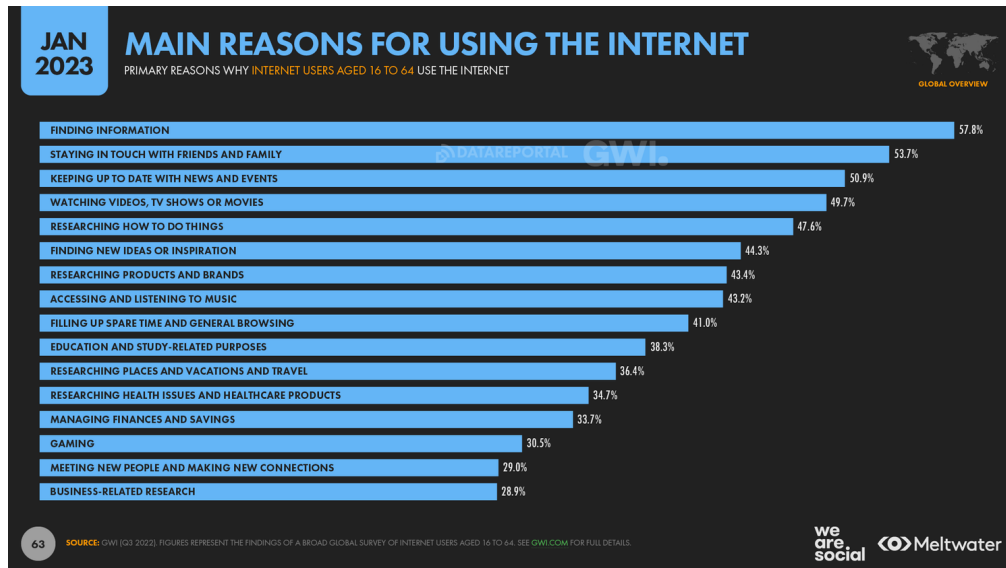
Graph 3 : Main reasons for using the internet. [4]
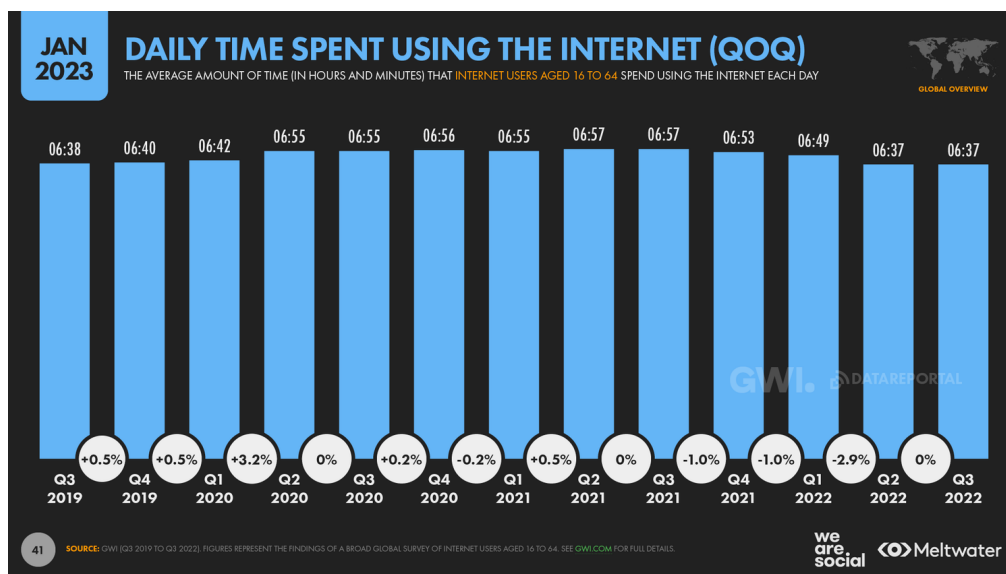
Graph 4 : Avg. amount of time spent by users aged 16-64 on the internet [4]

The problem our Fullstack Responsive Social Media App aims to solve is to provide a modern and responsive web application that allows users to create and share content with other users.

Users of the application can sign up for an account, log in, and post content that other users can like. Users can search for specific information. Furthermore, according to their preferences, users can switch between light and dark themes using the application's dark mode feature.

The need and desire for an exciting, innovative and responsive social media web application that enables users to create and share content in an easy-to-use manner is the problem that our project seeks to address. The application offers a number of features that enhance the UX in an effort to draw new users and retain the old ones who are looking for an engaging and dynamic platform to share their thoughts and content.

1.2 Problem Statement

Our app aims to address the need for a social media platform that allows users to create and share posts, like and comment on other users' posts, and authenticate users for security purposes. Additionally, the app includes a dark mode feature for better user experience in low light conditions. Overall, the app seems to aim at providing a robust and user-friendly social media platform for users.

1.3 Objectives

Our project's goal is to create a fully functional online application that leverages the MERN stack and provides users with a safe and secure platform to contribute their opinions, information, and other types of content. The online application we are building aims to create a platform that -

- Allows users to interact and engage with each other in a safe and secure environment. Our application will ensure that user information is protected from unauthorized access.

- Allows users to contribute their ideas and opinions without worrying about criticism or abuse. They will be able to connect with one another and engage in constructive criticism on the site, promoting a sense of wholesomeness in the community.

1.4 Methodology

The Fullstack Responsive Social Media MERN App was developed using a methodology that adheres to the conventional software development lifecycle, which includes requirement analysis, design, development, integration, and testing. We employed agile development approaches to make sure the project was finished on time and to the required quality standards.

- Requirement Analysis - The study of the project requirements was done during the first phase. It involved selecting the features and functionalities that we wished to offer, as well as the target market and website objective. To understand the needs, preferences, and behaviors of our users, we must define our target audience and develop user personas. Using this data, we need to design a website that appeals to our target audience.

- Design - During the design stage, a wireframe and application design layout were created. A user-friendly interface for the programme was intended, and the design was based on the criteria acquired in the initial stage. For the purpose of building responsive and resizable UI components for React-based web applications, we utilized the well-known open-source design framework Material UI. It provides a selection of pre-made components, such as buttons, forms, icons, typography, and many others, that may be used to create aesthetically beautiful user interfaces. A social networking platform's user profiles, posts, comments, notifications, and more may all be created and customized using the Material UI framework.

- Development - The development stage involved the actual coding of the application. We used the MERN stack, which includes Node.js, Express.js, React.js, and MongoDB, to develop the application. The three steps of the development process were database development, front-end or client-side development, and back-end or server-side development. During the front-end development stage, React is used to create the website's UI and client-side functionality. Making the pages, elements, and interactive features of the website falls under this category. Using Node.js and Express.js, the website's server-side functionality and API are developed during the back-end development stage. This entails developing new features like user identification and authorisation, friend addition and removal and toggling between dark mode and light mode. MongoDB is used to design and build the queries, data models, and database structure required for the website during the database development stage.

- Integration & Testing - During the integration phase, the front-end, back-end, and database components are integrated. The testing phase involved testing the application's functionality and ensuring that it met the requirements defined in the first stage.

1.5 Motivation

There are many social media applications available with various uses for various communities. One such application, Facebook, allows you to connect with friends, family, and other acquaintances, send messages, exchange images and videos, and get updates. On YouTube, users may watch the movies and songs they like, add original content, and share it all with their friends, family, and other users. We attempted to create an application where users may share their views and anything new happening in their lives after being inspired by the services of the aforementioned social media apps.

While developing your own social media application, you can tailor the features and functionality of your application to the particular requirements of your target market. You can develop a distinctive and interesting user experience with the help of this level of personalization. Additionally, you have total control over the platform's security when you create your own social network application. User loyalty and trust can be increased if you have strong security policies in place and make sure that user data is shielded from unauthorized access.

1.6 Organization

The rest of the report is organized as follows. Chapter 02 reviews the related work in development of social media applications using different tech stacks & privacy and security of websites. Chapter 03 discusses design of the project and several web development technologies, and system development including schema development of the database while chapter 04 carefully discusses the implementation of our project followed by the applications and limitations of the project and the conclusion in chapter 05.

Chapter 02: LITERATURE SURVEY

In the last few years, significant improvements have been made in social media. The development of technology like machine learning, artificial intelligence, and big data has made it possible for social media platforms to provide their customers more individualized services and targeted marketing. Additionally, new channels for social media participation have been made possible through the use of AR and VR technologies [5]. Furthermore, the integration of social media with other technologies such as blockchain is also being explored [6].

The two most popular tech stacks in the field of web development are MEAN (MongoDB, Express.js, Angular, and Node.js) and MERN (MongoDB, Express.js, React, and Node.js). According to a number of criteria, including performance, scalability, ease of development, and community support, the two stacks are evaluated in [7]. The writers carefully examine each technology in both stacks before comparing them based on a variety of different criteria. They also shed light on which stack would be more appropriate for a certain kind of web app. The study does, however, have a number of shortcomings. It doesn't go into great detail regarding the implementation of the two stacks or how to use them to build specific types of web apps. Additionally, the research only compares the two stacks without thoroughly analyzing any of their potential flaws or limitations. For developers who are debating which stack to employ for their web application, "Comparative analysis of MEAN stack and MERN stack" is a useful reference. [7] offers a thorough comparison of the two stacks and can assist developers in making a choice based on their own needs and specifications.

Since our stack of choice is the MERN stack, we'll try to learn more about it. MERN stack is a popular choice for developing web applications, including social media applications. The use of MongoDB, Express, React, and Node.js offers advantages such as scalability, flexibility, and faster development times.

The ability to use a single language (JavaScript) across the stack also simplifies development [8]. The "A Comprehensive Study of MERN Stack Architecture for Full Stack Web Development" paper offers a thorough examination of the MERN stack and its elements. The introduction to the MERN stack and its benefits over existing web development stacks comes first in the paper. After that, the authors go into each MERN stack element, including MongoDB, Express, React, and Node.js. They go into great length to explain how each of these parts functions and how they relate to one another. The MERN stack's numerous tools and libraries are also covered in the article in order to improve its usefulness. The MERN stack is also contrasted with other web development stacks, such MEAN and LAMP. [9]

The MERN (MongoDB, Express, React, Node.js) stack is proposed in the paper in [12] as a real-time social networking application. User registration, user authentication, profile creation, post creation, post likes and comments, and real-time notifications are just a few of the features available in the app. The architecture and development process of the application were also covered in the paper. Finally, the authors concluded that the MERN stack is a suitable option for building scalable and real-time web applications.

In [10], Jacob Wylie provides a detailed walkthrough of building a full-stack web application using the MERN stack, including the integration of Redux Toolkit. The article begins by setting up the backend using Node.js, Express, and MongoDB, then moves on to building the frontend with React and Redux Toolkit. Throughout the process, Wylie provides detailed explanations of the key concepts and techniques used, including the creation of Redux slices, the use of asynchronous actions, and the implementation of CRUD operations.

Users of social media have serious concerns about their privacy and security. The collection and analysis of user data on social media sites has become simpler thanks to technological advancements. Concerns over data privacy and security breaches have grown as a result of this. Platforms like social media are putting safeguards in place to allay these worries, like privacy controls and data encryption [11].

One of the essential components of developing a secure web application is the implementation of strong authentication and authorization mechanisms. It aids in protecting user data security and preventing unauthorized access. Authentication is the process of verifying a user's identity before granting access to the platform's resources. It helps ensure that only authorized users can access the platform's features and services. Social media sites utilize a variety of authentication methods, including password-based authentication, two-factor authentication, and biometric authentication, to confirm users' identities. Social networking platforms most frequently use password-based authentication. Users are required to enter a special username and password combination, which is compared to the database's credentials. In our application, password-based authentication is used. Authentication can be integrated with other technologies, such as blockchain, to further boost security [12]. Social media networks that employ blockchain may make sure that user authentication data is securely saved and cannot be altered without permission.

Chapter 03: SYSTEM DEVELOPMENT

3.1 Design of the Project

Fullstack Responsive Social Media App is a web application built using the MERN stack, which stands for MongoDB, Express.Js, React, and Node.js. It includes a range of features such as user authentication, posts, likes & comments, and dark mode, and is designed to be responsive and accessible on different devices and screen sizes. In this section, we will explore the design of the project in more detail -

- Backend Design: The application's backend is created with Node.js and Express.Js. It has a number of controllers and routes that deal with client-side application requests. A MongoDB database is also a part of the backend, and it houses users and likes data. JSON Web Tokens (JWTs) are also utilized by the backend to authorize and authenticate users.

- Frontend Design: The frontend of the application is built using React and includes several components that make up the UI. The application is designed to be responsive and accessible, and includes support for different screen sizes and device types. The application also includes dark mode functionality, which can be toggled on and off by the user.

- User Authentication: The application's user authentication feature is crucial, and JWTs are used to implement it. The backend creates a JWT token at login or registration, which is then saved in the user's local storage. The user is then verified using the token in order to make more requests to the backend. Additionally, the application has a logout feature that eliminates the JWT from the user's local storage.

- Likes Functionality: Users can like and dislike posts using the likes functionality. A request to add the like data to the MongoDB database is sent from the frontend to the backend whenever a user likes a post.

- Dark Mode Functionality: The dark mode functionality is implemented using Material UI, CSS variables and JavaScript. When the user toggles on the dark mode, the frontend sets the CSS variables for the application's colors and backgrounds to their dark mode equivalents. The application then stores the user's preference in their local storage, so that it is persisted across sessions.

3.2 Functional and Non-Functional Requirements

Here are some of the functional requirements of the application:

- User authentication: The application should allow users to create an account, login, and logout. User authentication should be secure and protect sensitive user data.

- Post creation: The application should allow users to create, edit, and delete posts. Posts can include text as well as images. It does not currently support addition of other media.

- Post interaction: The application should allow users to like and comment on posts created by other users.

- Dark mode: The application should have a dark mode option that changes the color scheme of the application.

- Responsive design: The application should be designed to work on different screen sizes and devices, including desktops, tablets, and mobile phones.

- Image upload: The application should allow users to upload images to their posts, which should be stored securely on the server.

- Search functionality: The application should allow users to search for other users and posts by keyword or tag.

These functional requirements are crucial to the success of the application and should be thoroughly tested to ensure that they meet the needs of the users.

Non-functional requirements of a software application refer to the aspects of the system that are related to its performance, usability, reliability, security, and other quality attributes. Here are some non-functional requirements of our app:

- Performance: The application should be fast and responsive, with quick page load times and minimal lag when interacting with the server.

- Scalability: The application should be able to handle a growing number of users and data without compromising its performance or functionality.

- Usability: The application should be easy to use, with intuitive navigation, clear instructions, and a visually appealing design.

- Reliability: The application should be stable and dependable, with minimal downtime and error-free operation.

- Security: The application should be secure, with measures in place to protect user data, prevent unauthorized access, and defend against cyber attacks.

- Compatibility: The application should work across different devices, with a responsive design that adapts to different screen sizes.

- Maintainability: The application should be easy to maintain and update, with well-organized code and clear documentation.

Overall, these non-functional requirements are crucial for making sure the application is efficient and successful at meeting user needs while also being effective, dependable, and user-friendly.

3.3 Technologies Used

● Languages: HTML, CSS, Javascript

HTML, CSS, and JavaScript are used for social media app development because they are the cornerstones of modern web development.

1) HTML is used to organize a webpage's content. It explains the organization and semantic significance of various elements on a web page, including headings, paragraphs, images, and links. Because of this, it is an essential component in creating any website, including social media apps.

2) The content of a webpage produced with HTML is styled with CSS. It is used to specify a webpage's visual design, color scheme, typography, and other aesthetic components. Developers can use CSS to add visual appeal, engagement, and interactivity to their web sites. This is crucial for social media apps, since they can aid to enhance user engagement with an aesthetically appealing design.

3) A web page can have interactive features and dynamic behavior by using JavaScript. Input validation is managed, input responses are handled, and asynchronous calls are made to web servers using it to build user interfaces that respond to user activities.

● Package Manager: npm

A package manager for Node.js applications is called NPM. Developers can use it as a command-line tool to install, manage, and share packages (sometimes referred to as modules or libraries) used in Node.js applications. Use of package managers, such as NPM, makes managing dependencies in software projects easier. In a software project, dependencies are third-party files or libraries that are used. These packages can be used to provide features like database connectivity, user authentication, and image manipulation.

Developers would have to manually download, install, and manage each dependency without a package management, which may be a laborious and error-prone procedure.

- Frameworks: React, React Router Dom, Node.js, Express.js, Mongoose

1) React: A well-liked JavaScript library for creating user interfaces is called React. React can be used to build reusable user interface (UI) components for a variety of social networking application functions, including messaging, news feeds, and user profiles.

2) React Router Dom: A package called React Router Dom gives React applications the ability to route data. It is a specialized package that can only be used in web-browser-based application development. It allows programmers to specify the application's navigation structure, allowing users to traverse between pages without refreshing the page.

3) Node.Js: A web framework built on JS that is used to build scalable and effective backends for online applications. It can be used to manage HTTP requests, and other server-side capabilities for social networking apps which also includes management of the database that is in use.

4) Express.js: It is a Node.js online application framework that incorporates a variety of features such as middleware and routing, making it simple to construct scalable and successful web apps.

5) Mongoose: A Node.js and MongoDB object data modeling (ODM) library is called Mongoose. It offers a schema-based method to modeling application data. In order to facilitate database interaction, Mongoose can be used to develop data models for users, posts, comments, and other relevant items in a social networking application.

● Libraries: Axios, Bcrypt, React-Redux, @reduxjs/toolkit, Redux-Persist, React-dropzone, Dotenv, Formik, Yup, @mui/material, @emotion/react, @emotion/styled, @mui/icons-material, JWT, Multer, Helmet, Morgan.

1) Axios: JS library Axios has been used in our application to send HTTP queries to the server-side API. It allows developers to fetch data from their own or a third-party server.

2) Bcrypt: This Javascript library is used to hash passwords. It provides a secure way to save passwords by hashing them before storing them in a database.

3) Dotenv: It is a JS library that loads environment variables into process.env without configuring and maintaining dependencies. It is employed in our project to load environment variables like the JWT_SECRET (Secret Key for hashing), PORT Number and MONGO_URL (For database connection).

4) Jsonwebtoken: Jsonwebtoken is a JavaScript package for user authentication. It provides a technique for creating JWTs, which can be used to validate users' identities in a web application. In our MERN application, it was used to construct and validate JWTs for user authentication.

5) Cors: The well known and widely used JS package Cors is used to enable Cross-Origin Resource Sharing (CORS) in a web application. It provides a technique for customizing CORS settings in order to accept requests from multiple domains.

6) React-toastify: It provides a simple and user-friendly API for displaying success, warning, and error messages. In our app, we use React-toastify to display notifications to the user.

7) Helmet: Helmet. js is an open source JavaScript library that assists you in securing your Node. js application by configuring a number of HTTP headers. It serves as a middleware for Express and other comparable technologies, automatically adding and removing HTTP headers to ensure compliance with online security standards.

8) Morgan: Morgan is yet another Node. js HTTP request logger middleware. It makes logging requests to your application easier.

9) Formik: Formik is a tiny collection of React components and hooks used to create forms in React and React Native. It helps with the three most vexing aspects: Changing the form state of values. Validation and error messages are provided.

10) Yup: Yup is a schema builder for runtime value parsing and validation. It offers a quick and easy approach to construct validation schemas and validate form inputs. In our application, it was used in combination with Formik.

11) Redux Toolkit: Redux Toolkit is a collection of tools that aid in the building of Redux applications. It comes with utilities for constructing and managing Redux stores, as well as developing Redux actions and reducers. It is used in our project to maintain the global state of the application, which includes user authentication status and dark mode status.

12) React Dropzone: React Dropzone is a common library used in React apps to handle file uploads.

13) Multer: Multer and Express have been used on the server side to manage file uploads.

14) MaterialUI:Material UI is a free and open-source React component library based on Google's Material Design. It comes with a large number of prebuilt components that are ready for use in production right away. It offers a collection of pre-made elements, like buttons, forms, icons, typography, and many others, that may be used to create a UI that is aesthetically pleasing.

- Database : MongoDB

MongoDB is a non-relational document database that can store data in JSON format. The MongoDB database includes a flexible data model that allows you to store unstructured data, as well as comprehensive indexing and replication support via rich and intuitive APIs.

- REST Client: Postman

Postman is an API Platform that allows developers to create, test, and iterate on their APIs. Postman claims to have over 25 million registered users and 75,000 open APIs as of February 2023, making it the world's largest public API hub. In addition to supporting a range of HTTP methods such as GET, POST, PUT, DELETE, and others, POSTMAN allows users to add custom headers, authentication, and other parameters to their requests.

- Text Editor : VS Code

Visual Studio Code is a simplified code editor that supports development tasks such as debugging, task execution, and version control. It seeks to give only the tools required for a speedy code-build-debug cycle, leaving more complex workflows to full-featured IDEs like Visual Studio IDE. It has a variety of tools and capabilities to assist developers in writing and debugging code more successfully. Consider code autocompletion, debugging capabilities, third-party extensions, and so on.

● Developer Tools: Google Chrome

Google Chrome Developer Tools is a collection of online developer tools integrated into the Google Chrome browser that allow developers to analyze, debug, and optimize web applications. You can use developer tools of other browsers such as Mozilla Firefox as well.

3.4 Folder Structure

Our app's folder structure is organized in a way that separates client-side code (React) from server-side code (Node.js) The primary folders and their contents are broken down as follows:

● client: Contains all the client-side code for the React application, including components, stylesheets, and other assets.

```
client
  node_modules
  public
  src
    components
    scenes
    state
    App.js
    index.css
    index.js
    theme.js
  .gitignore
  jsconfig.json
  package-lock.json
  package.json
  README.md
```
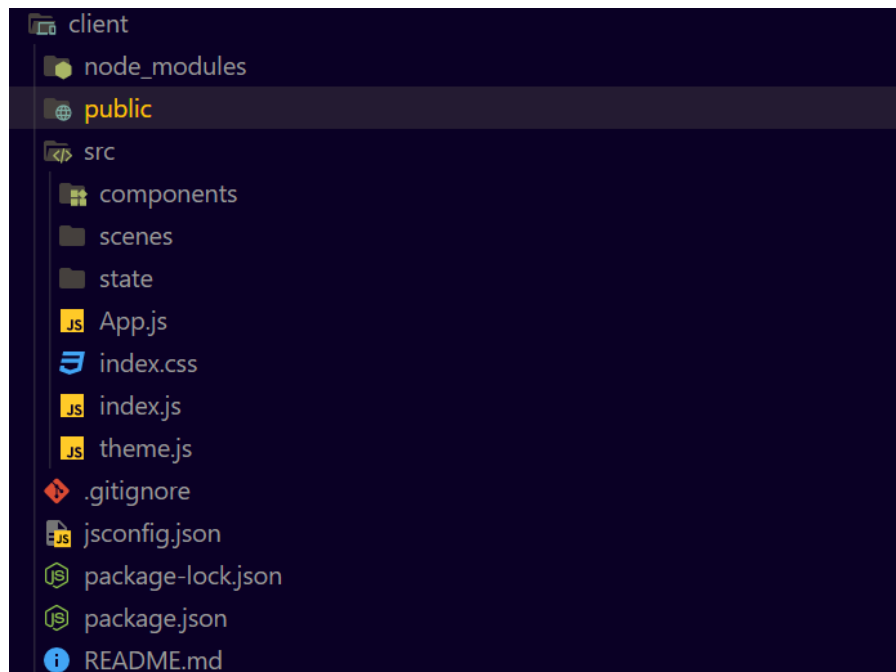
Fig 1: Folder Structure of Client-side code

- node_modules: The node_modules folder might be compared to a cache for the external modules that your project needs. They are downloaded from the internet and placed in the node_modules folder when you install them, and Node.js is programmed to search for them there when you import them (without a specified path). Because the node_modules folder can always be completely regenerated from scratch by just reinstalling all the dependent modules, I refer to it as a cache.

- public: Contains the index.html file and other static assets, such as images and fonts. It also contains the public assets which have PNG images of twitter and linkedIn icons.

- src: Contains the source code for the React application, organized into different folders for components, pages, styles, and other files.

- components: Contains reusable UI components used throughout the application. These components are written in JSX.

- scenes: Contains the main pages of the application, such as the login and home pages.

- state: Contains the Redux store configuration, including reducers and actions.

- App.js: The main React component that renders the application.

- index.js: The entry point for the React application.

- jsconfig: Contains configuration files for the Node.js server, including environment variables and database configuration.

- theme.js: It is used to define the global theme and styling variables for the application, such as colors, fonts, and spacing using Material UI.

- Server: Contains all the server-side code for the application, including models, routes, controllers, middlewares and other assets.
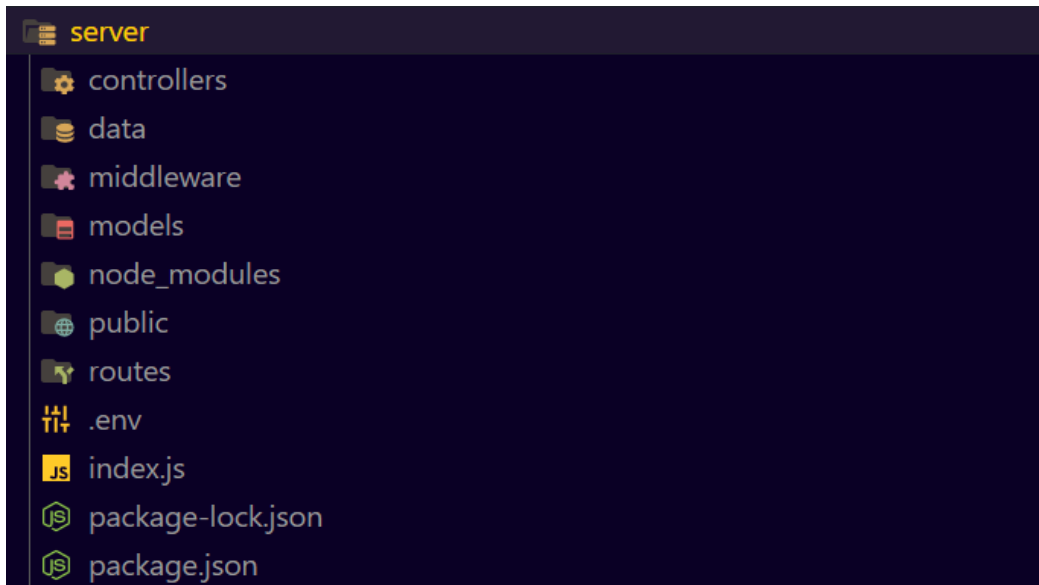
Fig 2: Folder structure of server side code of our application

- models: Contains the Mongoose schemas for the MongoDB database.

- routes: Contains the routes for the Node.js server, organized into different files for authentication, posts, and other features.

- data: Contains uploaded files, such as images.

- server.js: The entry point for the Node.js server.

- package.json: The NPM package file that contains metadata about the application and its dependencies. It is present in the client folder as well as the server folder.

```json
{
  "name": "server",
  "version": "1.0.0",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "gridfs-stream": "^1.1.1",
    "helmet": "^6.1.5",
    "jsonwebtoken": "^9.0.0",
    "mongoose": "^7.1.0",
    "morgan": "^1.10.0",
    "multer": "^1.4.4",
    "multer-gridfs-storage": "^5.0.2"
  },
  "devDependencies": {},
  "description": ""
}
```

Fig 3: Package.Json file Of Server Side code

```json
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.10.8",
    "@emotion/styled": "^11.10.8",
    "@mui/icons-material": "^5.11.16",
    "@mui/material": "^5.12.3",
    "@reduxjs/toolkit": "^1.9.5",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "dotenv": "^16.0.3",
    "formik": "^2.2.9",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-dropzone": "^14.2.3",
    "react-redux": "^8.0.5",
    "react-router-dom": "^6.11.0",
    "react-scripts": "5.0.1",
    "redux-persist": "^6.0.0",
    "web-vitals": "^2.1.4",
    "yup": "^1.1.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
```

Fig 4: Package.Json file Of Client Side code

- package-lock.json: The NPM lock file that ensures that dependencies are installed at the exact version specified in package.json.

- .env: The environment file that contains sensitive configuration data, such as database credentials and API keys.

```
#URL
MONGO_URL = 'mongodb+srv://priyansh:priyansh@cluster0.pn6wuot.mongodb.net/?retryWrites=true&w=majority'

#KEYS
JWT_SECRET = 'eyJhbGciOiJIUzM4NCJ9.
eyJSb2xlIjoiQWRtaW4iLCJJc3N1ZXIiOiJJc3N1ZXIiLCJVc2VybmFtZSI6IkphdmFJblVzZSIsImV4cCI6MTY4MzA5OTcyNCwiaWF0IjoxNjgzMDk5NzI0fQ.
oUh0nNEBUZydn59ewHEfJhAi9nQHaRpwhohQbenZ6Me-GV0SgIoNR-1640HUHwtI'

#PORT
PORT = 3001
```

Fig 5: Dotenv file for our application

Overall, the folder structure is organized in a logical and modular way that separates client-side and server-side code, making it easy to maintain and scale the application.

3.5 ER diagram

Entity-Relationship diagrams, also known as ER diagrams, are a sort of visual representation that illustrates the connections between entities in a database. Designing and modeling databases is a frequent practice in the software development industry, and it can be useful in ensuring that the database schema is effective, efficient, and fits the requirements of the application.
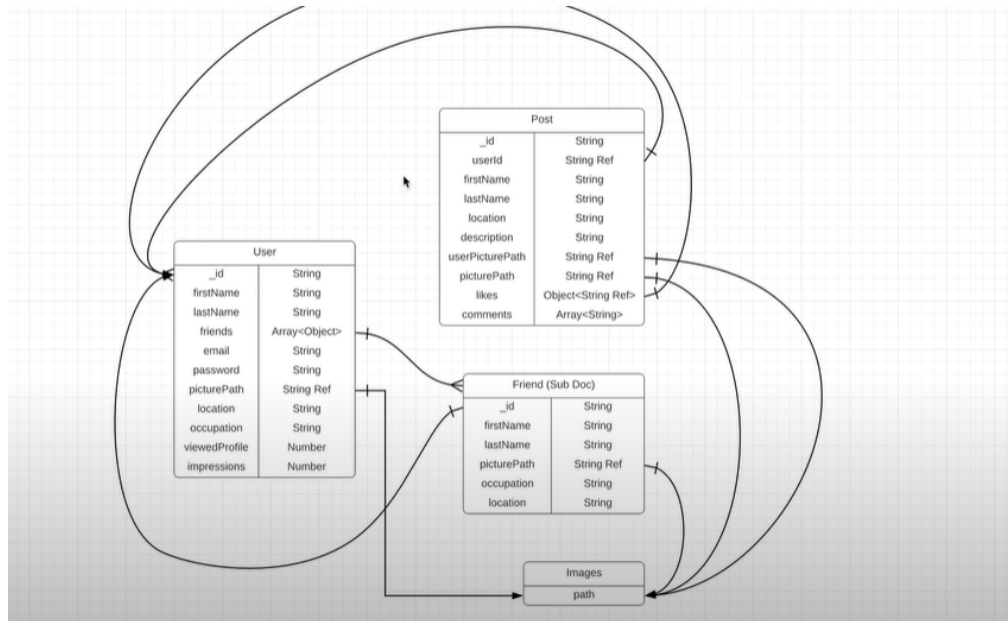

Fig 6: ER diagram of how data is modeled in our application

In our application, the ER diagram would show the relationships between different entities in the MongoDB database, such as users, friends, posts, likes, and comments. Here is a rough outline of what the ER diagram might look like:

- Users: This entity would represent the users of the application, and would likely contain fields such as firstName, lastName, friends, email, password, occupation and picture.

- Posts: This entity would represent the posts created by users, and would likely contain fields such as title, content, image, createdAt, likes, and comments.

```javascript
1   import mongoose from "mongoose";
2
3   const PostSchema = new mongoose.Schema(
4     {
5       userId: {
6         type: String,
7         required: true,
8       },
9       firstName: {
10        type: String,
11        required: true,
12      },
13      lastName: {
14        type: String,
15        required: true,
16      },
17      location: String,
18      description: String,
19      picturePath: String,
20      userPicturePath: String,
21      likes: {
22        type: Map,
23        of: Boolean,
24      },
25      comments: {
26        type: Array,
27        default: [],
28      },
29    },
30    { timestamps: true }
31  );
32
33  const Post = mongoose.model("Post", PostSchema);
34
35  export default Post;
36
```

Fig 7: Code Snippet for Post data Schema

```
 1   import mongoose from "mongoose";
 2
 3   const UserSchema = new mongoose.Schema(
 4     {
 5       firstName: {
 6         type: String,
 7         required: [true, "First Name is a required field"],
 8         min: 2,
 9         max: 50,
10         trim: true,
11       },
12       lastName: {
13         type: String,
14         required: [true, "Last Name is a required field"],
15         min: 1,
16         max: 50,
17         trim: true,
18       },
19       email: {
20         type: String,
21         required: [true, "Email is a required field"],
22         max: 100,
23         unique: true,
24         trim: true,
25         lowercase: true,
26       },
27       password: {
28         type: String,
29         required: [true, "Password is a required field"],
30         min: 8,
31       },
32       picturePath: {
33         type: String,
34         default: "",
35       },
```

```
36       friends: {
37         type: Array,
38         default: [],
39       },
40       location: String,
41       occupation: String,
42       viewedProfile: Number,
43       impressions: Number,
44     },
45     { timestamps: true }
46     //Will give timestamps when created-updated
47   );
48
49   const User = mongoose.model("User", UserSchema);
50   export default User;
51
```

Fig 8,9: Code Snippets for User data Schema

26

3.6 Pseudo Code

Client-side:

- Define a React component for the login page with a form to accept user credentials.

- Define a React component for the registration page with a form to accept user information.

- Define a React component for the home page with a form to create and submit new posts, and a section to display existing posts.

- Define a React component for the post page to display the details of a single post, including its likes and comments.

- Implement Redux Toolkit to manage application state, including user authentication status, current user information, and post data.

- Use Axios to make HTTP requests to the server-side API for CRUD (create, read, update, delete) operations on posts and user authentication.

- Use React Router DOM to manage client-side routing between pages.

- Implement Formik and Yup for form validation and handling.

- Use react-dropzone to allow users to upload images with their posts.

- Use Multer for handling file uploads on the server-side.

- Use Material UI to create responsive UI components and implement light and dark mode themes.

Server-side:

- Define API routes using ExpressJS and Mongoose for CRUD operations on posts and user authentication.

- Use Helmet to secure the ExpressJS app with various HTTP headers.

- Use Morgan for logging HTTP requests to the console.

- Implement Bcrypt to hash user passwords before storing them in the database.

- Use JSON Web Tokens (JWT) to authenticate and authorize user requests to protected routes.

- Use dotenv to load environment variables for server configuration.

- Implement CORS to allow cross-origin resource sharing between the client-side and server-side.

- Connect to a MongoDB database using Mongoose to store and retrieve post and user data.

3.7 Flow of the Application
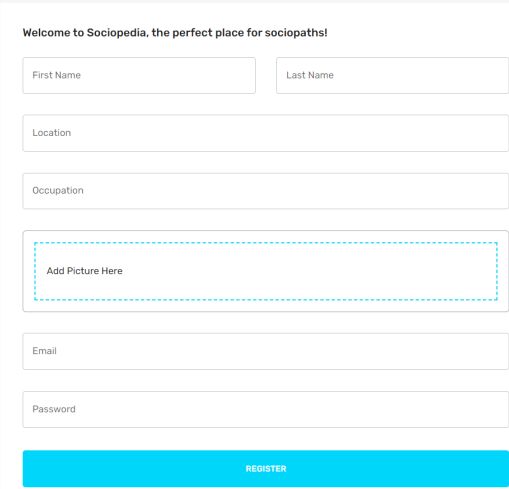
The flow our app can be summarized as follows:

- Authentication Flow: A user has the option to sign up or log in when they access the application. The user must first sign up by providing their email address and password, which are hashed and saved in the MongoDB database. The user must first input their email address and password, which are then verified against the database in order to log in. The user is routed to the application's home page after a successful authentication.

- Posting Flow: A user can create a new post by providing a title and content after logging in. The post is subsequently saved in the MongoDB database and shown on the application's home page. In addition, users can contribute photographs to go with their posts.

- Like Flow: When a user clicks the "like" button on a post, the server receives a POST request. The client then modifies the UI in accordance with the revised data, which is subsequently sent back to the server,

which updates the likes count for the article in the database.

- Comment Flow: By typing their comments in the comment box and hitting the submit button, users can leave comments on posts. The remark is then shown beneath the post and kept in the MongoDB database.

- Dark Mode Flow: By tapping the moon icon in the application's header, users can switch between light and dark mode. The React app experiences a state change as a result, and the CSS styles are updated appropriately.

- Responsive Flow: The UI layout varies based on the device being used because the application is responsive to multiple screen sizes. Media queries in the CSS and React components are used to do this.

3.8 Snapshots

UI Snapshots -

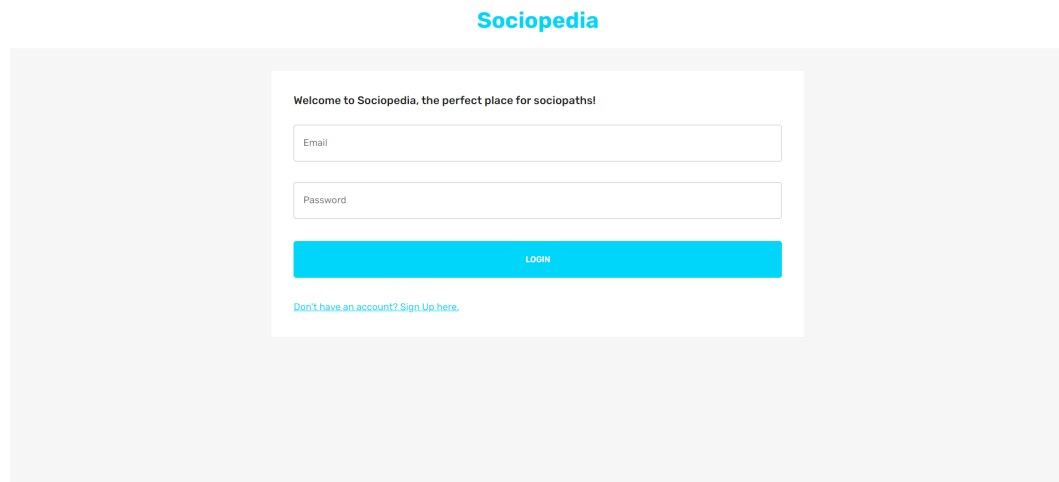**Fig 10: Registration Page for a new user**

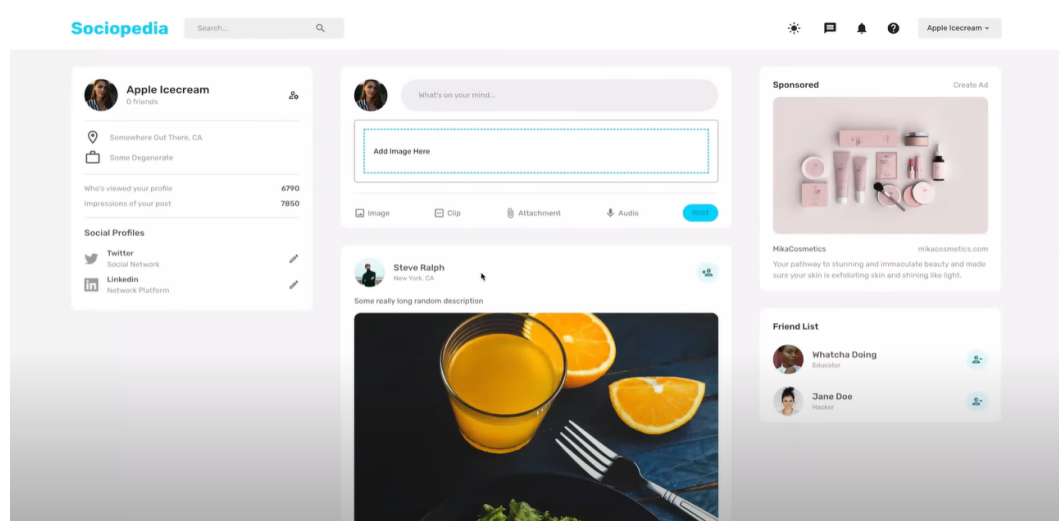**Fig 11: Login Page for an existing user**

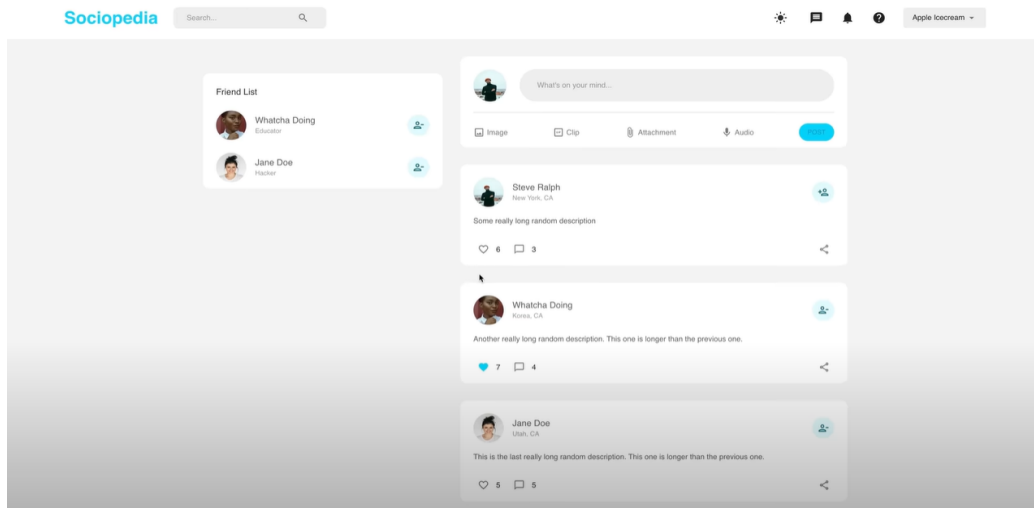**Fig 12: Home Page for a user**

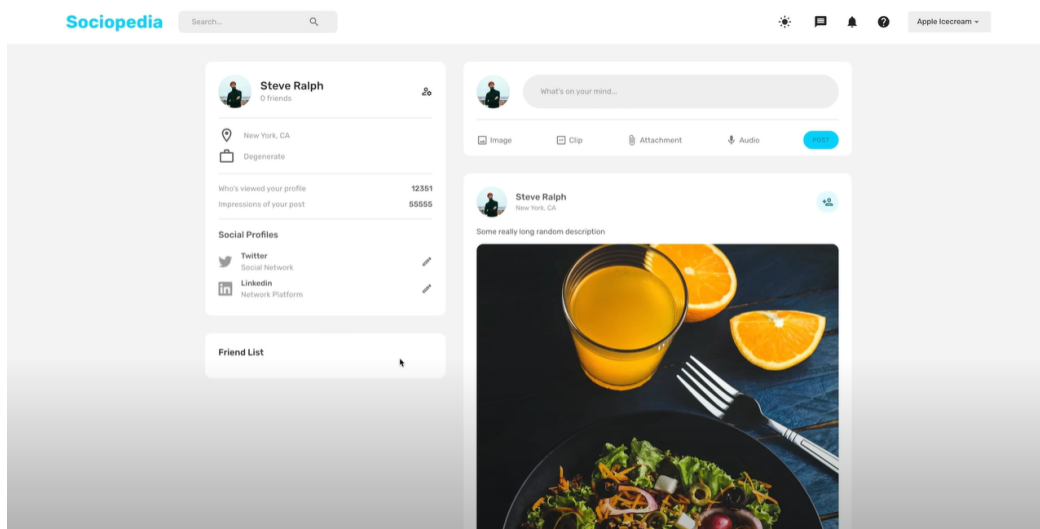**Fig 13: Feed for a user for all the posts**
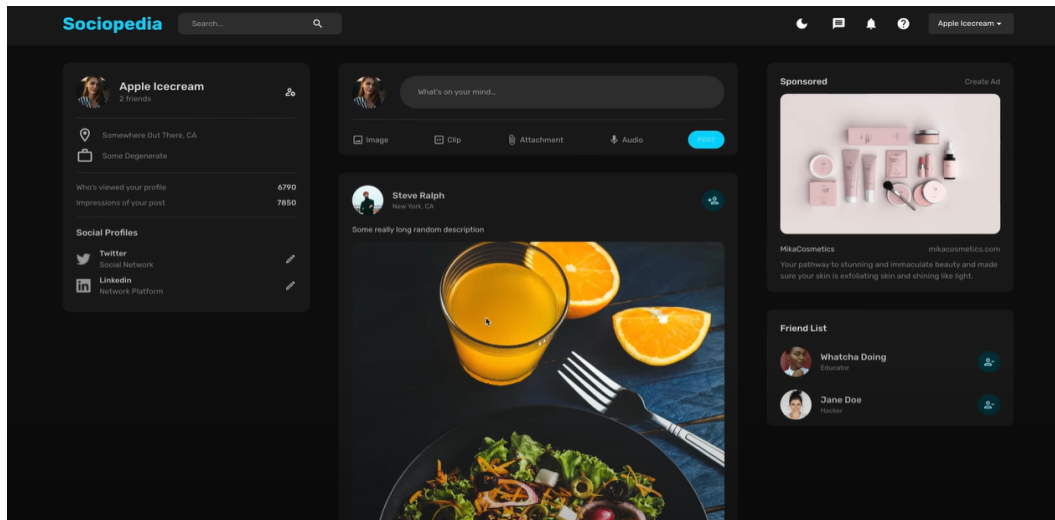

Fig 14: User Profile Page

Fig 15: Dark mode in our application

Code Snippets -


```
1   import { Box, useMediaQuery } from "@mui/material";
2   import { useSelector } from "react-redux";
3   import Navbar from "scenes/navbar";
4   import UserWidget from "scenes/widgets/UserWidget";
5   import MyPostWidget from "scenes/widgets/MyPostWidget";
6   import PostsWidget from "scenes/widgets/PostsWidget";
7   import AdvertWidget from "scenes/widgets/AdvertWidget";
8   import FriendListWidget from "scenes/widgets/FriendListWidget";
9
10  const HomePage = () => {
11    const isNonMobileScreens = useMediaQuery("(min-width:1000px)");
12    const { _id, picturePath } = useSelector((state) => state.user);
13
14    return (
15      <Box>
16        <Navbar />
17        <Box
18          width="100%"
19          padding="2rem 6%"
20          display={isNonMobileScreens ? "flex" : "block"}
21          gap="0.5rem"
22          justifyContent="space-between"
23        >
24          <Box flexBasis={isNonMobileScreens ? "26%" : undefined}>
25            <UserWidget userId={_id} picturePath={picturePath} />
26          </Box>
27          <Box
28            flexBasis={isNonMobileScreens ? "42%" : undefined}
29            mt={isNonMobileScreens ? undefined : "2rem"}
30          >
31            <MyPostWidget picturePath={picturePath} />
32            <PostsWidget userId={_id} />
33          </Box>
34          {isNonMobileScreens && (
35            <Box flexBasis="26%">
36              <AdvertWidget />
37              <Box m="2rem 0" />
38              <FriendListWidget userId={_id} />
39            </Box>
40          )}
41        </Box>
42      </Box>
43    );
44  };
45
46  export default HomePage;
```

Fig 16: Code Snippet of Homepage

```jsx
1   import { Box, useMediaQuery } from "@mui/material";
2   import { useEffect, useState } from "react";
3   import { useSelector } from "react-redux";
4   import { useParams } from "react-router-dom";
5   import Navbar from "scenes/navbar";
6   import FriendListWidget from "scenes/widgets/FriendListWidget";
7   import MyPostWidget from "scenes/widgets/MyPostWidget";
8   import PostsWidget from "scenes/widgets/PostsWidget";
9   import UserWidget from "scenes/widgets/UserWidget";
10
11  const ProfilePage = () => {
12    const [user, setUser] = useState(null);
13    const { userId } = useParams();
14    const token = useSelector((state) => state.token);
15    const isNonMobileScreens = useMediaQuery("(min-width:1000px)");
16
17    const getUser = async () => {
18      const response = await fetch(`http://localhost:3001/users/${userId}`, {
19        method: "GET",
20        headers: { Authorization: `Bearer ${token}` },
21      });
22      const data = await response.json();
23      setUser(data);
24    };
25
26    useEffect(() => {
27      getUser();
28    }, []); // eslint-disable-line react-hooks/exhaustive-deps
29
30    if (!user) return null;
31
32    return (
33      <Box>
34        <Navbar />
35        <Box
36          width="100%"
37          padding="2rem 6%"
38          display={isNonMobileScreens ? "flex" : "block"}
39          gap="2rem"
40          justifyContent="center"
41        >
42          <Box flexBasis={isNonMobileScreens ? "26%" : undefined}>
43            <UserWidget userId={userId} picturePath={user.picturePath} />
44            <Box m="2rem 0" />
45            <FriendListWidget userId={userId} />
46          </Box>
47          <Box
48            flexBasis={isNonMobileScreens ? "42%" : undefined}
49            mt={isNonMobileScreens ? undefined : "2rem"}
50          >
51            <MyPostWidget picturePath={user.picturePath} />
52            <Box m="2rem 0" />
53            <PostsWidget userId={userId} isProfile />
54          </Box>
55        </Box>
56      </Box>
57    );
58  };
59
60  export default ProfilePage;
```

Fig 17, 18: Code Snippet of User Profile Page

```javascript
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
  mode: "light",
  user: null,
  token: null,
  posts: [],
};

export const authSlice = createSlice({
  name: "auth",
  initialState,
  reducers: {
    setMode: (state) => {
      state.mode = state.mode === "light" ? "dark" : "light";
    },
    setLogin: (state, action) => {
      state.user = action.payload.user;
      state.token = action.payload.token;
    },
    setLogout: (state) => {
      state.user = null;
      state.token = null;
    },
    setFriends: (state, action) => {
      if (state.user) {
        state.user.friends = action.payload.friends;
      } else {
        console.error("User friends non-existent :(");
      }
    },
    setPosts: (state, action) => {
      state.posts = action.payload.posts;
    },
    setPost: (state, action) => {
      const updatedPosts = state.posts.map((post) => {
        if (post._id === action.payload.post_id) return action.payload.post;
        return post;
      });
      state.posts = updatedPosts;
    },
  },
});

export const { setMode, setLogin, setLogout, setFriends, setPosts, setPost } =
  authSlice.actions;
export default authSlice.reducer;
```

Fig 19: Code Snippet for Redux toolkit configuration

```
1   //for password encryption
2   import bcrypt from "bcrypt";
3
4   //Sends a web token to the user that they can use for authorization
5   import jwt from "jsonwebtoken";
6   import User from "../models/User.js";
7
8   /* REGISTER USER */
9   export const register = async (req, res) => {
10    //callback function
11    //req -> request from the frontend
12    //res -> response from the backend
13    try {
14      const {
15        firstName,
16        lastName,
17        email,
18        password,
19        picturePath,
20        friends,
21        location,
22        occupation,
23      } = req.body; //destructuring parameters from req.body
24
25      const salt = await bcrypt.genSalt(); //Used to encrypt our password
26      const passwordHash = await bcrypt.hash(password, salt);
27
28      //The flow will be - > We will encrypt the password and then we'll save it. When user tries to log in,
29      //they'll provide the password and we'll make sure it is the correct one and then we'll give them a json web token.
30      const newUser = new User({
31        firstName,
32        lastName,
33        email,
34        password: passwordHash,
35        picturePath,
36        friends,
37        location,
38        occupation,
39        viewedProfile: Math.floor(Math.random() * 1000),
40        impressions: Math.floor(Math.random() * 1000),
41      });
42      const savedUser = await newUser.save();
43      res.status(201).json(savedUser);
44      //We will the send the user a status of 201 that indicates the creation of a resource
45      //It is the reponsibility of the bakcend engineer to ensure that correct data is being sent as response
46    } catch (err) {
47      res.status(500).json({ error: err.message });
48    }
49  };
50
```

```
51    //Logging in
52    export const login = async (req, res) => {
53      try {
54        const { email, password } = req.body;
55        const user = await User.findOne({ email: email });
56        if (!user) return res.status(400).json({ msg: "User does not exist. " });
57
58        //Compares the hashes of sent password and stored password
59        const isMatch = await bcrypt.compare(password, user.password);
60        if (!isMatch) return res.status(400).json({ msg: "Invalid Credentials. " });
61
62        const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);
63        //JWT_SECRET must be Some Super hard string to guess
64        delete user.password;
65        res.status(200).json({ token, user });
66      } catch (error) {
67        res.status(500).json({ error: err.message });
68      }
69    };
70
```

Fig 20, 21: Code Snippet for password encryption

```
1    import jwt from "jsonwebtoken";
2
3    export const verifyToken = async (req, res, next) => {
4      try {
5        let token = req.header("Authorization");
6
7        if (!token) {
8          return res.status(403).send("Access Denied");
9        }
10
11       if (token.startsWith("Bearer")) {
12         token = token.slice(7, token.length).trimleft();
13       }
14
15       const verified = jwt.verify(token, process.env.JWT_SECRET);
16       req.user = verified;
17
18       next();
19     } catch (error) {
20       res.status(500).json({ error: error.message });
21     }
22   };
23
```

Fig 22: Code for middleware for Auth

```
1    import express from "express";
2    import {
3      getUser,
4      getUserFriends,
5      addRemoveFriend,
6    } from "../controllers/users.js";
7    import { verifyToken } from "../middleware/auth.js";
8
9    const router = express.Router();
10
11   //Read routes
12   router.get("/:id", verifyToken, getUser);
13   router.get(":/id/friends", verifyToken, getUserFriends);
14
15   //Update
16   router.patch("/:id/:friendId", verifyToken, addRemoveFriend);
17
18   export default router;
19
```

```
1    import express from "express";
2    import { login } from "../controllers/auth.js";
3
4    //Setup a router -> allows express to identify that these routes
5    // will all be configured and allows us to keep them in separate files
6    const router = express.Router();
7
8    router.post("/login", login);
9
10   export default router;
11
```

```
1    import express from "express";
2    import { getFeedPosts, getUserPosts, likePost } from "../controllers/posts.js";
3    import { verifyToken } from "../middleware/auth.js";
4
5    const router = express.Router();
6
7    //Read
8    //Sends all the feeds
9    router.get("/", verifyToken, getFeedPosts);
10   //Get specific users posts
11   router.get("/:userId/posts", verifyToken, getUserPosts);
12
13   //Update
14   router.patch("/:id/like", verifyToken, likePost);
15
16   export default router;
17
```

Fig 23, 24, 25: Code Snippets for routes of Users, Posts and auth

Database screenshots-

\+ Create Database

🔍 Search Namespaces

▾ **test**

| **posts**

users

test.posts

STORAGE SIZE: 20KB   LOGICAL DATA SIZE: 3.58KB   TOTAL DOCUMENTS: 6   INDEXES TOTAL SIZE: 20KB

Find   Indexes   Schema Anti-Patterns ⓪   Aggregation   Search Indexes   Charts ●

INSERT DOCUMENT

Filter ⬀   Type a query: { field: 'value' }   Reset   **Apply**   More Options ▸

QUERY RESULTS: **1-6 OF 6**

```
_id: ObjectId('645a34135e0abfd14d139b1d')
userId: "645a34135e0abfd14d139b16"
firstName: "Steve"
lastName: "Ralph"
location: "New York, CA"
description: "Some really long random description"
picturePath: "post1.jpeg"
userPicturePath: "p3.jpeg"
▸ likes: Object
▸ comments: Array
__v: 0
createdAt: 2023-05-09T11:52:56.894+00:00
updatedAt: 2023-05-09T11:52:56.894+00:00
```
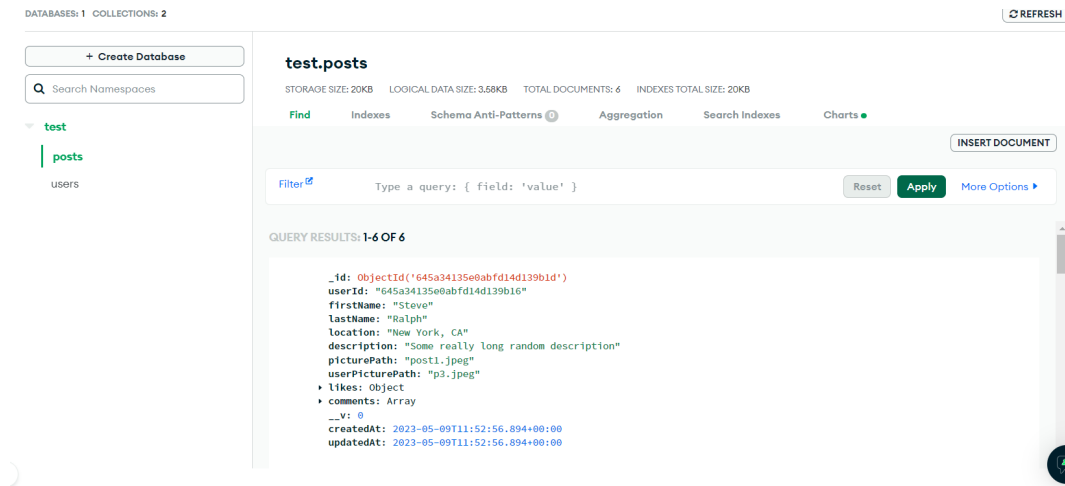
Fig 26, 27: MongoDB Collection for different posts and users

Chapter 04: PERFORMANCE ANALYSIS

Performance analysis is the measuring and analysis of various elements influencing how quickly, responsively, and dependably a website operates. Developers can identify and fix potential problems that might be degrading the website's usability and user experience by conducting a performance analysis.

To increase the website's speed and responsiveness, developers can use a variety of performance optimization strategies in addition to monitoring and load testing tools. One such strategy is to optimize the website's photos and other media assets to make them smaller and load more quickly.

To raise the website's efficiency and responsiveness, developers might also enhance its source code and database queries. This can be achieved by decreased HTTP requests, database queries, and website CSS and JavaScript file optimisations.

To give users the greatest experience possible, developers may use a variety of performance monitoring and load testing tools as well as speed optimisation approaches. Additionally, it's crucial to do continuing performance optimization and analysis to solve any potential problems that may develop over time and guarantee the website continues to function at its best.

Conducting load testing to assess how the website works under high traffic is one of the first steps in performance analysis. Several simulated people can access a website at once while it is being tested under load, and the server's capacity, resource utilization, and response time are all being tracked. Automating this process allows for the generation of comprehensive performance data using load testing tools like JMeter and LoadRunner. Another important aspect of performance analysis is database optimization. To make procedures for data retrieval and manipulation efficient and effective, this entails optimizing the database structure, queries, and indexing. Use software to analyze and improve database performance, such as MongoDB Compass.

Optimizing the performance of social media apps is crucial for several reasons:

- User satisfaction: Social media platforms have become an integral part of people's daily lives, and users expect them to be fast and responsive. Slow-loading pages or unresponsive features can lead to frustration and a poor user experience, leading users to switch to other platforms.

- Retention and engagement: Slow or poorly performing apps can impact user retention and engagement. Users are more likely to return to and spend more time on an app that performs effectively, loads quickly, and provides a seamless user experience.

- Growth of the Business: A terribly optimized social networking app might stifle the growth of the company that created it. A slow or unreliable software can result in unfavorable reviews, user loss, and, eventually, income loss.

Response Time of the Website:

Perf_hooks which is a Node.js built-in module, allows you to monitor the performance of your Node.js applications. It's used to calculate how long it takes for the code's functions to get executed.

Developers can detect bottlenecks and speed up their code by monitoring the amount of time a given block of code or function takes to finish.

```
router.get("/performance/getPerformance", (req,res)=>{
    const t1 = performance.now();
    const responseTime = t1 - t0;
    res.status(201).json({message:responseTime});
});
```
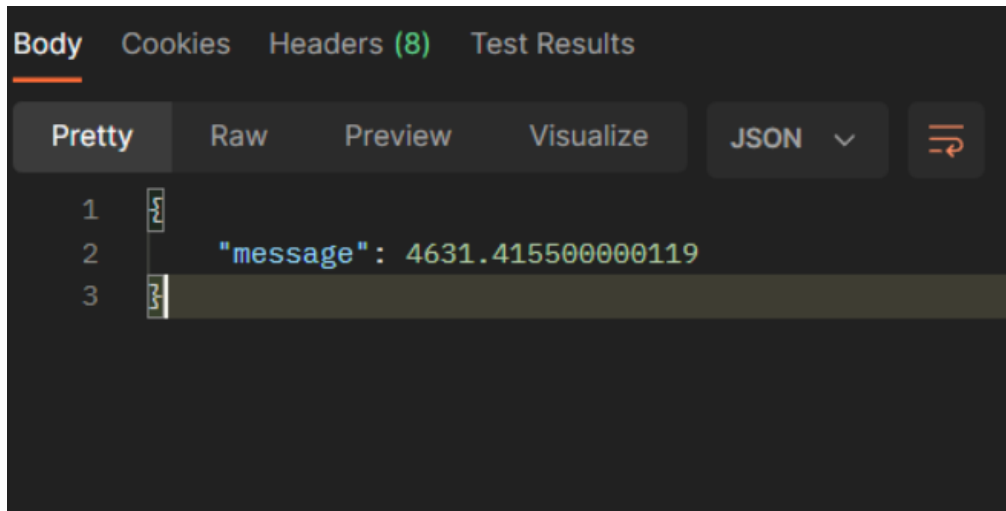
Fig 28, 29: Load Time of the website

React Hooks:

React Hooks are basic JavaScript functions that can be used to separate reusable components from functional components. Hooks can be declarative and manage side effects. UseState(), UseEffect() and UseContext() are some of the most commonly used hooks.

Hooks in the project:

The following hooks were utilized in our application:

- useState(): useState is a React Hook that adds state to a functional component. It returns an array with two values: the current state and an update function. It is used in the project to govern user authentication status, profile information, and the app's dark mode.

- useMemo(): useMemo is a hook in the React functional component that returns a memoized value.It is used in the project to save user profile data and reduce unnecessary component re-rendering.

- useDispatch(): Actions are dispatched to the Redux store using this hook. The project uses it to dispatch user authentication, profile data, and dark mode state operations.

- useNavigate(): This hook is applied to the app to enable programmatic navigation. It is used in the project to switch between the app's many pages, including the login, signup, and user profile pages.

These hooks are used to manage state and avoid needless component re-rendering in order to improve the efficiency of the app. These hooks enable the programme to efficiently update and provide data to the user, improving user experience. Additionally, by centralizing state management and lowering the amount of API calls necessary, the implementation of Redux in the MERN stack architecture contributes to performance optimization. Overall, by offering a stable and expandable foundation for creating high-performance web applications, the MERN stack contributes to optimisation.

Chapter 05: CONCLUSION

After everything is finished, we have a full-stack web application that enables users to create accounts, log in, post, like, and comment on content while switching between light and dark themes. The MERN stack, which consists of MongoDB, Express.js, React, and Node.js, is used to build the application. It also includes a number of third-party libraries and frameworks, including Redux Toolkit, Formik, Yup, Multer, and React Dropzone.

When a user creates an account, their data is encrypted with bcrypt and saved safely in the MongoDB database. Following that, the user can write posts, which are made up of a title, text, and optional image. The application maintains track of the number of likes and comments received for each post and lets other users like or remark on it. The Material UI library is used to switch between a light and dark look for users of the application.

5.1 Applications

Here are some possible applications of this type of app:

- Social Networking: The application is ideal for social networking applications since it enables users to upload content and engage with other users through likes and comments. Using this kind of app, a social network for a certain interest or community might be established.

- Blogging:The programme works well with blogging platforms since it lets users write, read, and update entries. Using this kind of app, a platform for bloggers or authors to contribute their content might be created.

- E-commerce: The application is ideal for e-commerce applications because it lets users upload both text and photos. An online marketplace for goods might be built using this kind of app, where users could post products and communicate with one another through likes and comments.

- Collaboration: The application is helpful for collaborative apps because it enables users to write, read, and update posts. A platform for teams or organizations to work together on projects, discuss ideas, and interact with one another might be created using this kind of app.

Overall, our App has a wide range of potential applications, and could be customized to meet the needs of a variety of different types of users and businesses.

5.2 Limitations

There are several potential limitations of our app that should be considered:

- Scalability: Although the application is somewhat scalable, there might be restrictions on how well it can manage high volumes of traffic or users. The application might need more infrastructure or optimisation if user activity were to significantly grow and the application were to become very popular.

- Security:The application offers a number of security protections, like password hashing and JWT authentication, but security flaws or assaults are always a possibility. It is crucial to maintain the application up to date with the most recent security patches and best practises, as well as to carry out routine security audits, in order to reduce these risks.

- Complexity: The application uses a variety of different features and technologies, which could make developing and maintaining it more difficult. This intricacy may cause problems with continuous development, maintenance, and bug fixing.

- Compatibility: The application was created using a particular set of technologies and versions, which might not work with all situations or systems. For instance, a user may be unable to utilize the application if their machine cannot run the necessary versions of Node.js or MongoDB.

- User Experience: The application is intended to be responsive and user-friendly, but there may be restrictions on how effectively it can accommodate different users' needs and preferences. Some users could, for instance, prefer different color schemes or layouts from what the application offers.

Overall, while our application is a well-designed and functional application, it is important to consider these potential limitations in order to ensure that it meets the needs and expectations of its users.

5.3 Future Work

Here are several potential ways to improve our MERN App:

- Add more features: Authentication, likes, and dark mode are some of the application's basic features right now, but there are a lot more that might be included to enhance both its usability and user experience. The programme might be made more interesting and practical for users by adding features like the ability to comment on posts, search for topics or individuals, or integrate with other social media sites.

- Optimize performance: Despite the fact that the programme is designed to be scalable, there may be ways to increase its efficiency and reduce load times. Installing cache, for example, could help to reduce server strain and speed up user page loads.

- Enhance security: Although the programme includes a variety of security security measures, extra steps may be done to strengthen its security and protect user data.

REFERENCES

[1] Kemp, S. (2023) Digital 2023: India - DataReportal – global digital insights, DataReportal. Available at : https://datareportal.com/reports/digital-2023-india

[2] Hootsuite. (2023). Available at: https://www.hootsuite.com/resources/digital-2023

[3] "Social media worldwide - statistics & facts." Statista, https://www.statista.com/topics/1164/social-networks/

[4] S. Kemp, "Digital 2023: India," *DataReportal – Global Digital Insights*, 13-Feb-2023. [Online]. Available: https://datareportal.com/reports/digital-2023-india.

[5] M. Trier and A. Richter, "The deepening of digitalization: An introduction to the special issue on digitalization in the context of social media," Journal of Business Research, vol. 122, pp. 10–13, 2021.

[6] B. Xu, X. Du, J. Shao, L. Yu, and Y. Yang, "Blockchain-based decentralized social media platform: Architecture design and performance analysis," Journal of Network and Computer Applications, vol. 171, 2020.

[7] S. Agarwal and J. Verma, "Comparative analysis of MEAN stack and MERN stack," International Journal of Recent Research Aspects, vol. 7, no. 3, pp. 10–14, 2020.

[8] Shah, M. (2021). MERN Stack – A Complete Guide. Available at: https://www.geeksforgeeks.org/mern-stack-a-complete-guide/.

[9] S. Patil and S. Chavan, "A Comprehensive Study of MERN Stack Architecture for Full Stack Web Development," International Journal of Advanced Research in Computer Science, vol. 11, no. 5, pp. 97–101, 2020.

[10]   "Medium,"   Medium.   [Online].   Available:
https://javascript.plainenglish.io/full-stack-web-application-with-mern-stack-and
-redux-toolkit-a4a4aee4d301.

[11] G. Bello-Orgaz, J. J. Jung, and D. Camacho, "Social big data: Recent achievements and new challenges," Inf. Fusion, vol. 28, pp. 45–59, 2016.

[12] F. Yang, Q. Tang, Z. Xu, and J. Dai, "A blockchain-based biometric authentication scheme for social media platforms," Journal of Ambient Intelligence and Humanized Computing, vol. 12, no. 4, pp. 4611–4621, 2021.

[13] N. S. Ebere, "How to build a full-stack authentication app with react, Express, MongoDB, heroku, and netlify," freecodecamp.org, 05-Jul-2022. [Online].   Available:
https://www.freecodecamp.org/news/how-to-build-a-fullstack-authentication-sys
tem-with-react-express-mongodb-heroku-and-netlify/.