

## DECLARATION

I hereby declare that the Minor Project Report entitled "Socialize : A real-time social media platform" is an authentic record of work completed as a requirement of the Minor Project during the period from 09/10/2024 to 18/11/2024 in University School of Automation and Robotics under the supervision of Mr. Chiranjit Pal.



(Signature of student)

Ashish Prasad

10419011921

Date: 19/11/2024



(Signature of Supervisor)

Mr. Chiranjit Pal

Date: 19/11/2024

# Minor Project Report

on

SOCIALIZE : A Real Time Social Media Platform

Submitted in partial fulfillment of the  
requirement for the completion of minor project[ARP 455]

**Name: Ashish Prasad**

**Enrollment Number: 10419011921**

**Under the supervision of**

**Mr. Chiranjit Pal**

Assistant Professor

USAR, GGSIPU



**UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS  
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY  
EAST DELHI CAMPUS, SURAJMAL VIHAR, DELHI- 110032**

## **DECLARATION**

I hereby declare that the Minor Project Report entitled “Socialize : A real-time social media platform” is an authentic record of work completed as a requirement of the Minor Project during the period from 09/10/2024 to 18/11/2024 in University School of Automation and Robotics under the supervision of Mr. Chiranjit Pal.

**(Signature of student)**

**Ashish Prasad**

**10419011921**

**Date:** \_\_\_\_\_

**(Signature of Supervisor)**

**Mr. Chiranjit Pal**

**Date:** \_\_\_\_\_

## ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my supervisor, **Mr. Chiranjit Pal** (Assistant Professor), for his invaluable guidance, unwavering support, and insightful feedback throughout the development of this project, “**Socialize: A Real-Time Social Media Platform.**” His expertise and encouragement were instrumental in shaping the direction of my work and enhancing the quality of this project.

I am also grateful for his constant motivation, which inspired me to tackle challenges effectively and explore innovative solutions. His constructive criticism and suggestions were pivotal in helping me refine my ideas and achieve the project’s objectives.

Ashish Prasad

10419011921

AI-DS B-2

## TABLE OF CONTENT

S.no		Page No.
1.	Abstract	1
2.	Introduction	2
3.	Problem statement	3
4.	Major Modules of project	4
5.	Methodology	6
6.	Hardware and Software Requirements	17
7.	Snapshots of IDE/ WEB PORTAL	19
8.	Results	21
9.	References	23

# 1. ABSTRACT

This project is a modern social networking platform inspired by Facebook, designed to facilitate seamless user interactions and foster connections. The platform allows users to engage with each other through core functionalities such as liking posts, commenting, and sending friend requests. With an intuitive interface and responsive design, it offers a user-friendly experience across devices, ensuring accessibility for diverse audiences.

The platform emphasises social engagement by enabling users to create and share posts, including text and media. The “like” and “comment” features provide avenues for interaction, encouraging meaningful conversations and fostering a sense of community. Additionally, the friend request functionality helps users expand their network, forming connections with others based on shared interests.

A personalised user experience is central to the platform’s design. Each user has a customizable profile where they can update their bio, upload profile and cover photos, and manage privacy settings to control the visibility of their posts and information. A dynamic news feed curates updates from connections, ensuring relevant content is always accessible.

Built using modern web technologies, the project integrates a robust frontend developed with HTML, CSS, and JavaScript, complemented by a secure and scalable backend architecture. Data management practices prioritise efficiency and security, ensuring user information is protected while maintaining high performance.

Overall, the platform provides a comprehensive social networking experience, combining usability, interactivity, and data security to create a reliable and engaging environment for users to connect and communicate.

## 2. INTRODUCTION

Social networking has transformed how individuals connect, communicate, and share information in the digital age. This project aims to create a lightweight, scalable, and secure social media platform designed to enhance user interaction and engagement. By incorporating features such as user authentication, content engagement, and real-time interaction, the platform provides a dynamic and immersive social networking experience.

The platform's foundation lies in ensuring a seamless and secure user experience. User authentication is implemented to protect accounts and maintain privacy, enabling secure login and account management processes. Once logged in, users can actively engage with the content through core features like posting updates, liking, and commenting, fostering meaningful interactions. Real-time interaction capabilities, such as live notifications and activity updates, further enhance engagement by ensuring instant connectivity and communication.

A key focus of the project is scalability, ensuring that the platform can handle a growing number of users without compromising performance. Efficient backend architecture supports data-intensive operations while maintaining speed and reliability. At the same time, lightweight frontend development ensures responsiveness and accessibility across various devices. Built using full-stack web development technologies, the project combines robust frameworks and tools to deliver an intuitive and feature-rich interface. Modern design principles ensure usability, making navigation straightforward for users of all technical backgrounds.

This project reflects a comprehensive approach to modern social networking, blending usability, security, and scalability to create a platform that caters to a diverse user base while fostering active interaction and engagement.

### 3. PROBLEM STATEMENT

The central challenge of this project lies in creating a social networking platform that seamlessly combines user interactivity and engagement with robust technical capabilities. To achieve this, the system must effectively address key aspects such as secure data management, efficient handling of interactions, real-time communication, and user-friendly design. One of the primary requirements is ensuring the **secure storage of user data and credentials**. Protecting sensitive information is critical to maintaining user trust and privacy. This necessitates the implementation of encryption mechanisms and secure storage solutions to safeguard user accounts and personal details from unauthorized access.

Another essential component is the **efficient and scalable management of interactions** such as likes, comments, and friend requests. The platform must be designed to handle high volumes of activity without compromising performance. Scalability ensures that as the user base expands, the system continues to operate smoothly, providing a consistent experience.

Additionally, **real-time communication** is vital for fostering user engagement. Features such as instant notifications, live updates, and real-time chat enhance the sense of connection and immediacy. Achieving this requires integrating reliable, low-latency communication protocols to deliver a dynamic and interactive experience for users.

Finally, a **user-friendly interface and responsive design** are crucial to the platform's success. The system must be intuitive and accessible across devices, catering to users of varying technical expertise. By prioritizing usability and responsiveness, the platform ensures that users can easily navigate and interact with its features. Addressing these challenges is fundamental to building a robust, scalable, and engaging social networking platform.



## 4. MAJOR MODULES OF PROJECT

The project consists of several key modules, each designed to handle specific functionalities and ensure seamless operation. These modules work together to provide a secure, interactive, and user-friendly platform:

### 4.1. User Authentication Module

This module ensures secure access to the platform by implementing robust authentication mechanisms.

- **Features:**
  - Login and registration using **JSON Web Tokens (JWT)** for session management.
  - Password hashing with **bcrypt** to protect sensitive user information.
  - Role-based access control to restrict unauthorized actions.

### 4.2. User Profile Management Module

This module allows users to create, update, and manage their profiles while enabling social connectivity.

- **Features:**
  - Profile creation with personal details, profile pictures, and bio.
  - Friend request system for sending, accepting, and rejecting connection requests.
  - Dynamic profile updates visible to other users.

### 4.3. Content Interaction Module

Focused on enabling users to engage with each other through posts and interactions.

- **Features:**
  - Post creation with support for text, images, and multimedia.
  - Interactions like likes and comments to promote engagement.
  - Efficient storage and retrieval of content in the database.

### 4.4. Real-Time Communication Module

Handles instant notifications and updates to enhance user interaction.

- **Features:**
  - **WebSocket** integration for real-time updates, such as new friend requests or comments.
  - Low-latency and reliable communication for notifications and messages.

#### 4.4. Frontend Design Module

Ensures a responsive and visually appealing user interface.

- **Features:**
  - Developed using **HTML** and **SCSS** for a clean and modern design.
  - Responsive layouts to adapt to devices of various screen sizes.
  - Seamless integration with backend APIs for dynamic content updates.

#### 4.5. Database Management Module

Responsible for storing, organizing, and managing application data.

- **Features:**
  - Utilizes **MongoDB** for scalable and flexible storage of user data, posts, and interactions.
  - Indexing for optimized search and retrieval operations.
  - Secure storage of sensitive information, such as hashed passwords.

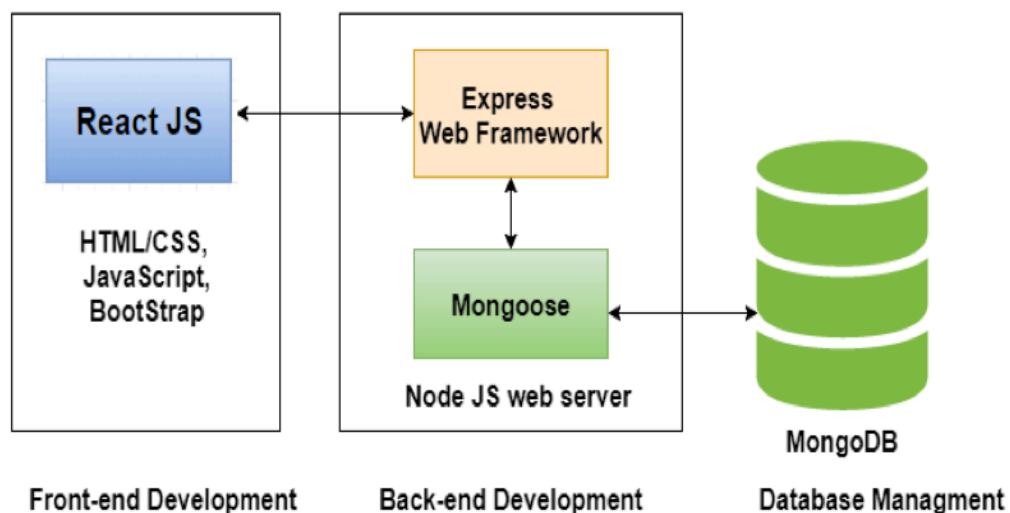
#### 4.6. Notification and Alert Module

Enhances user engagement by keeping them informed of activities.

- **Features:**
  - Push notifications for events like friend requests and comments.
  - Integration with the real-time communication system for instant updates.

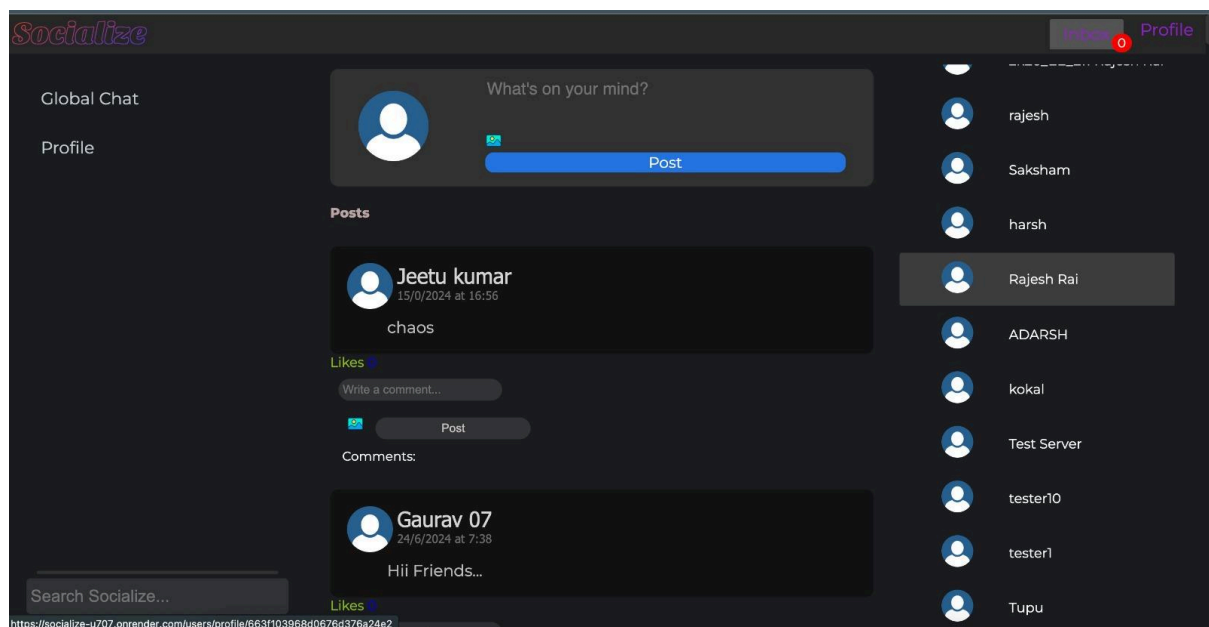
## 5. METHODOLOGY ADOPTED

The methodology of the project follows a modular and systematic approach, focusing on key aspects like security, user interaction, and responsive design. For **user authentication**, the project utilizes **JWT (JSON Web Tokens)** to securely handle login and session management. This method ensures stateless authentication, where the server doesn't need to store user session data, enhancing scalability and performance. User passwords are securely stored in MongoDB, with encryption techniques like bcrypt to protect sensitive information. **User profile management** allows individuals to create, update, and personalize their profiles, featuring functionalities like profile picture uploads and a friend request system, enabling users to connect and interact with others. The **content interaction module** supports users in posting content and



## 5.1 Frontend Development

Frontend development is focused on creating an intuitive, responsive, and visually appealing user interface that ensures a seamless user experience across various devices. In this project, the frontend is developed using **HTML** and **SCSS**. HTML serves as the foundation for structuring the web pages, ensuring semantic layout and accessibility. SCSS, a CSS preprocessor, enhances the styling process by offering advanced features like variables, nesting, and mixins, which help organise and maintain complex stylesheets. The goal is to make the platform responsive, meaning it adapts to different screen sizes (desktop, tablet, mobile), ensuring users can access the platform from any device without losing usability. This is achieved through **media queries** and **flexbox** or **grid layouts** to adjust page elements based on the screen size. The frontend also integrates with the backend via API calls to retrieve user data, posts, comments, and notifications. User interactions such as form submissions, post creations, and comments are handled dynamically, improving engagement and creating a smooth experience.



**Figure 1** : User Interface built using HTML ,SCSS

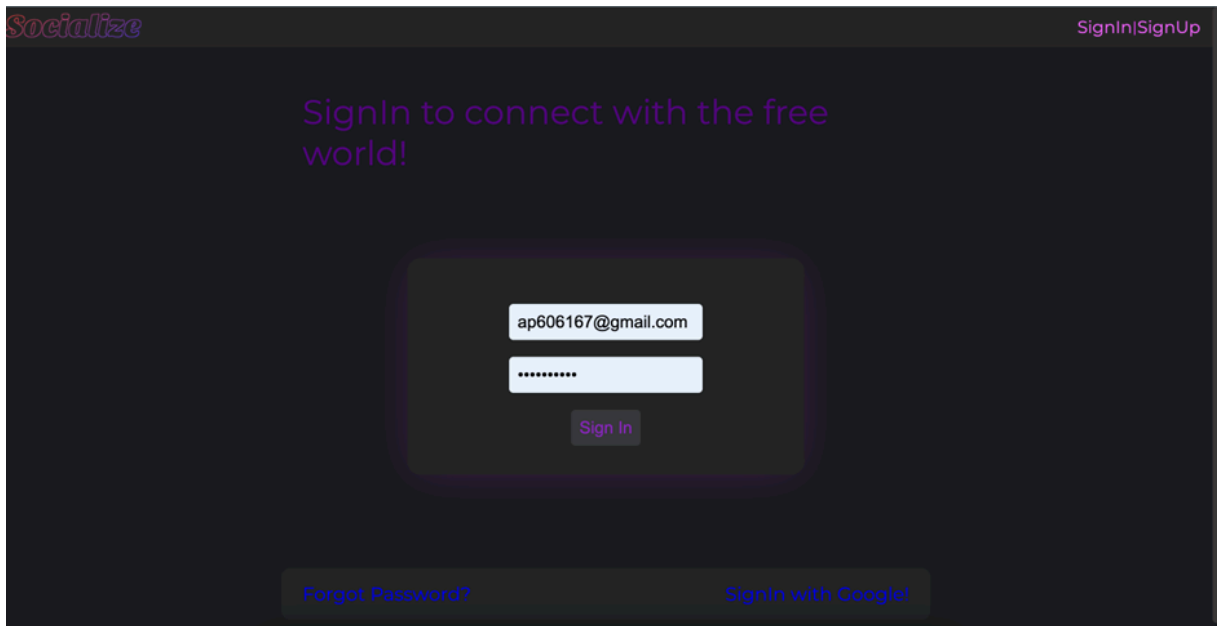


Figure 2

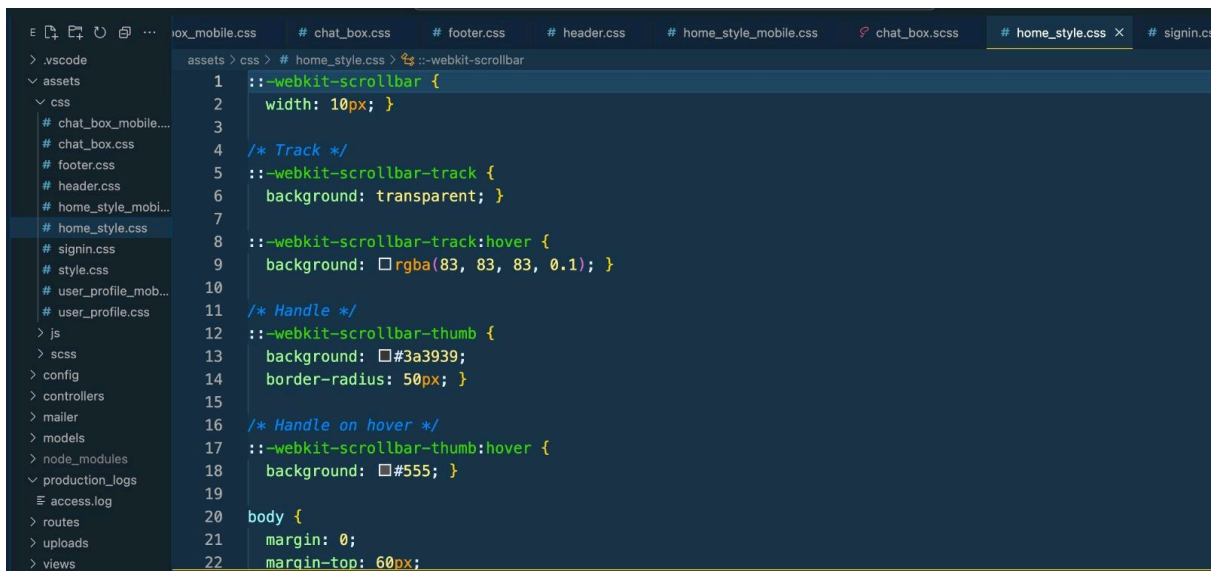
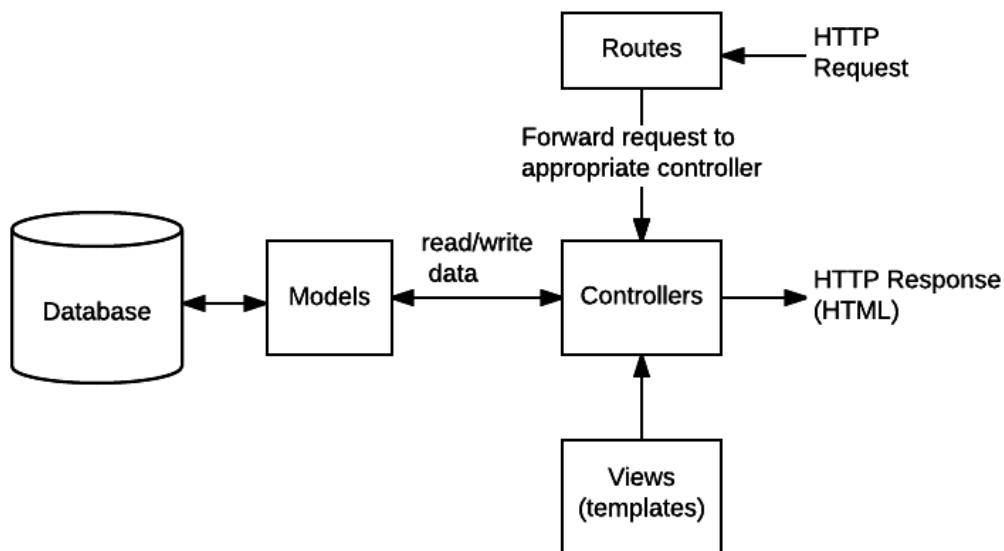


Figure 3

## 5.2 Backend Development

Backend development involves the creation of the server, application logic, and database interactions. In this project, the backend is responsible for handling user requests, processing data, and sending responses. The server is typically built with Node.js and Express, ensuring an efficient runtime environment and routing framework. The backend handles crucial tasks like managing user authentication, processing friend requests, storing and retrieving posts, likes, and comments, and pushing notifications. It processes incoming requests (e.g., login, register, post creation) and interacts with the database to fetch, update, or delete data as necessary. It also ensures the security and integrity of data by enforcing validation and implementing appropriate access control mechanisms.

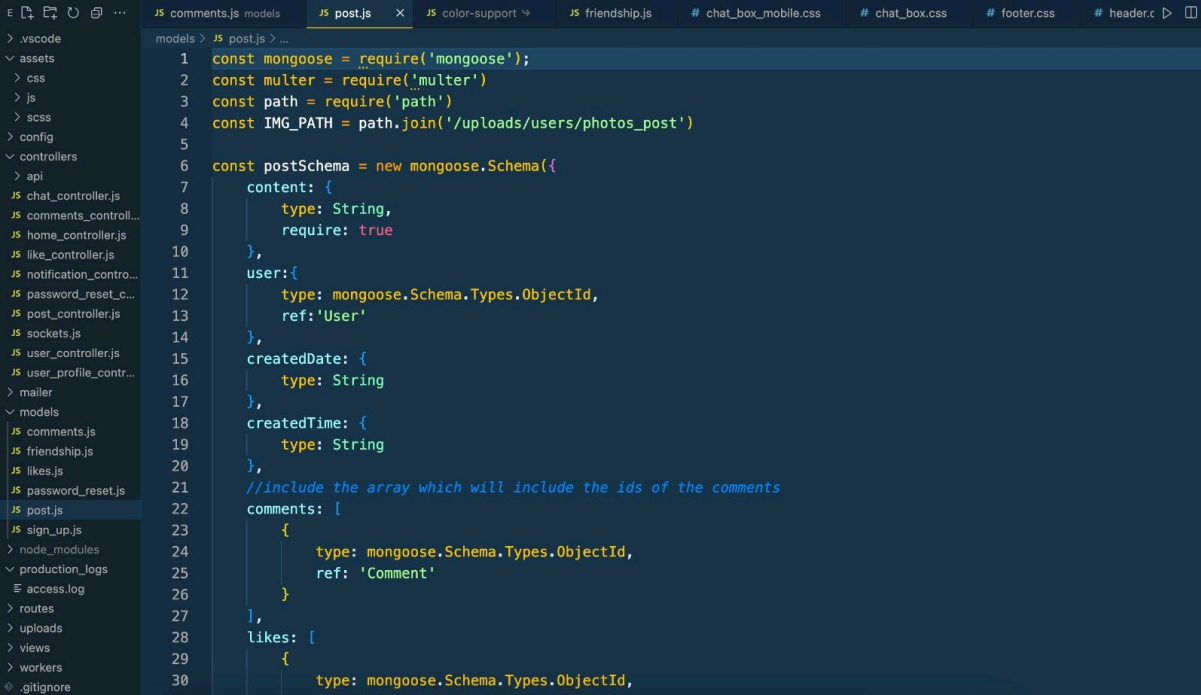


**Figure 4** : Backend Architecture



## 5.3 Database

The database serves as the central repository for storing all the application data. In this project, MongoDB, a NoSQL database, is used due to its flexibility in handling unstructured data and its scalability. MongoDB stores user information, including profile data, posts, comments, friend requests, and interactions. Since MongoDB uses a document-oriented structure (JSON-like format), it allows for the easy storage of diverse data types, such as text, images, or multimedia files. The database schema is designed to support efficient queries and fast data retrieval, even with a large volume of users and interactions. Indexing is also implemented to optimize search queries and ensure smooth performance as the user base grows.



```
1 const mongoose = require('mongoose');
2 const multer = require('multer')
3 const path = require('path')
4 const IMG_PATH = path.join('/uploads/users/photos_post')
5
6 const postSchema = new mongoose.Schema({
7   content: {
8     type: String,
9     require: true
10  },
11   user: {
12     type: mongoose.Schema.Types.ObjectId,
13     ref: 'User'
14  },
15   createdAt: {
16     type: String
17  },
18   createTime: {
19     type: String
20  },
21   //include the array which will include the ids of the comments
22   comments: [
23     {
24       type: mongoose.Schema.Types.ObjectId,
25       ref: 'Comment'
26     }
27  ],
28   likes: [
29     {
30       type: mongoose.Schema.Types.ObjectId,
```

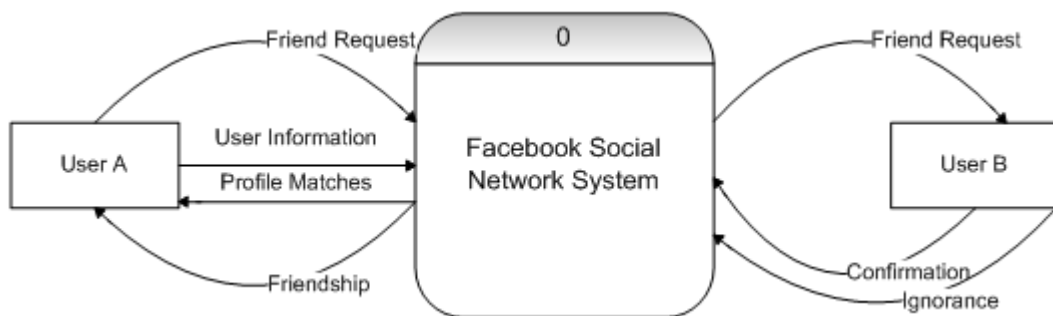


```
models > JS sign_up.js > ...
1  const mongoose = require('mongoose');
2  const multer = require('multer');
3  const path = require('path');
4  const AVATAR_PATH = path.join('/uploads/users/avatars')
5
6  const UserSchema = new mongoose.Schema({
7    email:{
8      type:String,
9      required:true,
10     unique:true
11   },
12   password:{
13     type:String,
14     required:true,
15   },
16   name:{
17     type:String,
18     required:true,
19   },
20   avatar:{
21     type:String
22   },
23   pendingFR: [
24     {
25       type: mongoose.Schema.Types.ObjectId,
26       ref: 'User'
27     }
28   ],
29   sentFR: [
30     {
31       type: mongoose.Schema.Types.ObjectId
```

```
models > JS likes.js > ...
1  const mongoose = require('mongoose');
2  const LikeSchema = new mongoose.Schema({
3    user:{
4      type: mongoose.Schema.ObjectId,
5      ref: 'User'
6    },
7    // what post or comment is liked, its id is saved in likeable
8    likeable : {
9      type: mongoose.Schema.ObjectId,
10     required: true,
11     refPath: 'onModel'
12   },
13   //to choose between whether a post is liked or a comment
14   onModel: {
15     type: String,
16     required: true,
17     enum: ['Post', 'Comment']
18   }
19 }, {timestamps:true})
20
21 const Like = mongoose.model('Like', LikeSchema)
22
23 module.exports = Like
```

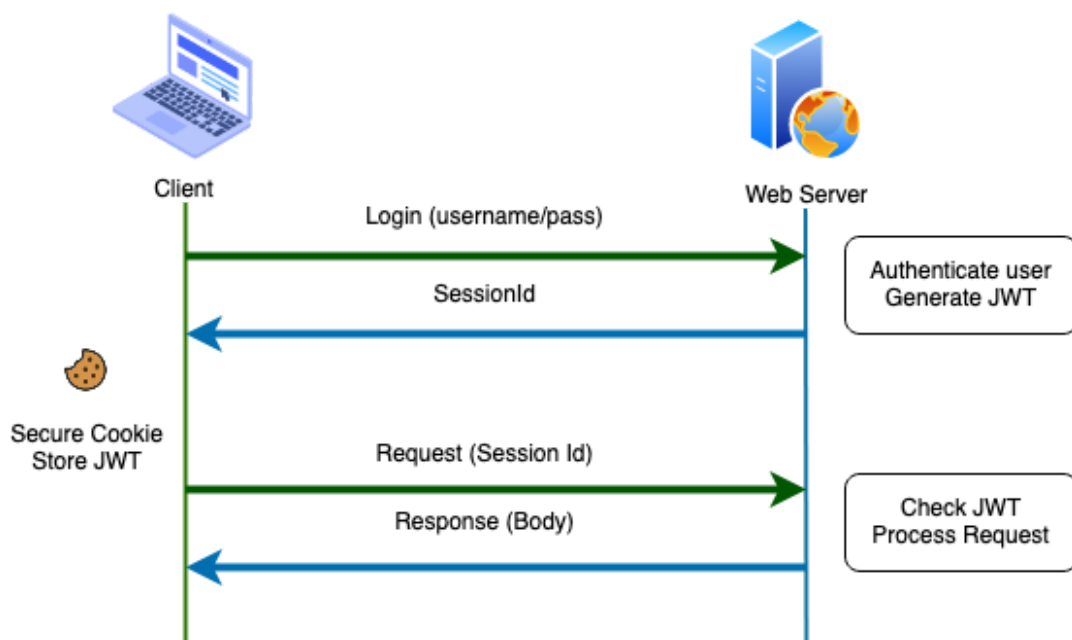
## 5.4 Data Flow and Communication

Data flow refers to how data moves between the user interface (frontend), the server (backend), and the database. The frontend makes API requests to the backend, which processes the data and interacts with the database. For example, when a user logs in, the frontend sends the credentials to the backend, which validates them and responds with a JWT. Similarly, when a user posts content or comments, the frontend sends the post data to the backend, which then stores it in the database and returns a success response. The backend is responsible for ensuring data consistency and integrity, as well as processing logic like authentication, validation, and error handling. The communication between the frontend and backend occurs via HTTP/HTTPS requests and responses, usually following RESTful principles.



## 5.5. User Authentication with JWT Token

- **JWT (JSON Web Token)** authentication ensures secure access to the platform by verifying the user's identity. Upon login, the user is issued a JWT token, which is then used to authenticate all subsequent requests to the server. This ensures that only authenticated users can access features, protecting sensitive data and maintaining security.
- The JWT token is stored on the client side and sent with each request to confirm the user's identity, enabling secure and seamless session management.
- The platform uses **JWT** for **secure user authentication** and **session management**. This prevents unauthorized access to features and protects user data.
- **Token expiration and renewal** mechanisms can be employed to ensure that user sessions remain secure over time, requiring re-authentication after a set period. This adds an extra layer of security to the platform, ensuring that user data and interactions are always protected



**Figure 8 :** JWT authentication flow diagram

```
JS passport-google-oauth2-strategy.js JS passport-jwt-strategy.js X JS passport-local-strategy.js JS middleware.js JS mongoose.js JS nodemailer.js
> .vscode
> assets
> css
> js
> scss
> config
JS environment.js
JS kue.js
JS middleware.js
JS mongoose.js
JS nodemailer.js
JS passport-google-o...
JS passport-jwt-strat...
JS passport-local-str...
> controllers
> mailer
JS comments_mailer.js
JS password_reset_m...
> models
JS comments.js
JS friendship.js
JS likes.js
JS password_reset.js
JS post.js
JS sign_up.js
> node_modules
> production_logs
JS access.log
> routes
> api
JS comments.js
JS index.js
JS posts.js
JS userProfile.js

config > JS passport-jwt-strategy.js > ...
1 const passport = require('passport')
2 const JWTStrategy = require('passport-jwt').Strategy
3 const ExtractJwt = require('passport-jwt').ExtractJwt
4 const User = require('../models/sign_up');
5 const env = require('../environment')
6
7 const opts = {
8   jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
9   secretOrKey: env.jwtSecretKey
10 }
11
12
13 // when the user log in to ther page, it creates a JWT which will expire in 60 seconds(set up in the users_api.js)
14 // then when we are going to delete a post, we are first authenticating [passport.authenticate('jwt', {session: false}
15 // below is the function which will act as a middleware, checking if the user exists in the token.
16 // Now is the token is expired we will get message "Unauthorized".
17 passport.use(new JWTStrategy(opts, function(jwtPayload, done){
18
19   User.findById(jwtPayload._id, function(err, user){
20     if(err){
21       console.log(err); return;
22     }
23     if(user){
24       return done(null, user);
25     }
26     else{
27       return done(null, false);
28     }
29   })
30 })})
```

```
JS passport-google-oauth2-strategy.js JS passport-jwt-strategy.js JS passport-local-strategy.js X JS middleware.js JS mongoose.js JS nodemailer.js
> .vscode
> assets
> css
> js
> scss
> config
JS environment.js
JS kue.js
JS middleware.js
JS mongoose.js
JS nodemailer.js
JS passport-google-o...
JS passport-jwt-strat...
JS passport-local-str...
> controllers
> mailer
JS comments_mailer.js
JS password_reset_m...
> models
JS comments.js
JS friendship.js
JS likes.js
JS password_reset.js
JS post.js
JS sign_up.js
> node_modules
> production_logs
JS access.log
> routes
> api
JS comments.js
JS index.js
JS posts.js
JS userProfile.js

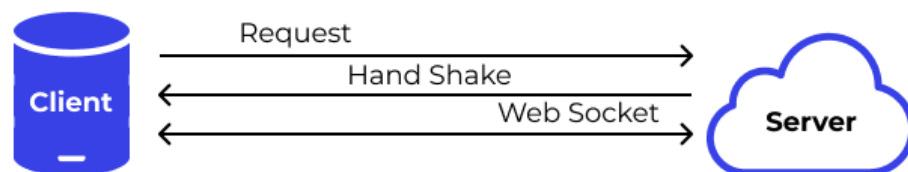
config > JS passport-local-strategy.js > ...
1 const req = require('express/lib/request');
2 const passport = require('passport');
3 const LocalStrategy = require('passport-local').Strategy;
4 const user = require('../controllers/user_controller');
5 const User = require('../models/sign_up');
6
7 //authentication using passport
8 passport.use(new LocalStrategy(
9   { usernameField: 'email',
10     passReqToCallback: true
11   },
12   function (req, email, password, done) {
13     // find a user and establish an identity
14     User.findOne({ email: email }, function (err, user) {
15       if (err) {
16         req.flash('error', err);
17         return done(err);
18       }
19       if (!user || user.password !== password) {
20         req.flash('success', 'Invalid Username or Password!');
21         return done(null, false);
22       }
23       return done(null, user);
24     })
25   })
26 )
27
28 passport.serializeUser(function (user, done) {
29   // stores user.id in encrypted format to cookie
30   done(null, user.id);
```

## 5.6. WebSocket Connection

Real-time communication is crucial in modern web applications, especially in social platforms where users expect immediate feedback, such as notifications for friend requests, likes, and comments. **WebSocket** is used to enable full-duplex, low-latency communication between the server and the client. Unlike traditional HTTP requests, WebSocket maintains an open connection between the client and server, allowing the server to push data to the client instantly without the need for the client to repeatedly request updates. When a user receives a new friend request or a comment on their post, the server sends a notification through the WebSocket connection, and the client updates the user interface in real time. This method ensures high performance, reduces network traffic, and provides an interactive experience for users. WebSocket is ideal for applications where instantaneous communication is required, as it maintains an open connection and eliminates the need for constant polling, which can be resource-intensive. The use of WebSocket ensures that notifications are delivered reliably and efficiently across the platform, contributing to a smooth and dynamic user experience.

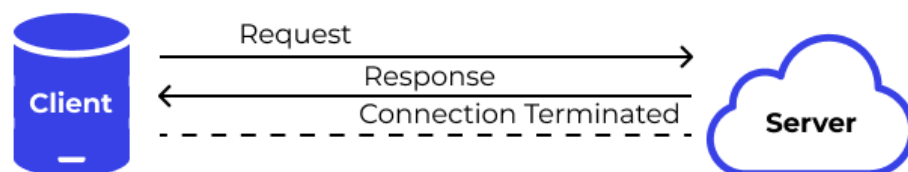


### WebSocket Connection



VS

### HTTP Connection



## **6. HARDWARE REQUIREMENTS AND SOFTWARE REQUIREMENTS**

### **6.1 HARDWARE REQUIREMENTS**

#### **1. Development Machine:**

- Processor Multi-core processor (e.g., Intel i5 or AMD Ryzen 5 and above)
- RAM: Minimum 8 GB (16 GB or more recommended for smoother multitasking)
- Storage: SSD with at least 256 GB (512 GB or more preferred)
- Graphics: Integrated graphics sufficient; discrete GPU optional for UI rendering performance
- Network: Stable internet connection for real-time API interactions

#### **2. Server/Hosting Environment:**

- Processor: Quad-core or higher (e.g., Intel Xeon or AMD EPYC)
- RAM: Minimum 16 GB (32 GB or higher for handling concurrent users)
- Storage: SSD storage for fast read/write operations (minimum 500 GB)
- Network Bandwidth: High bandwidth for handling multiple API requests and client-server interactions

## **6.2 SOFTWARE REQUIREMENTS**

### **1. Frontend Development:**

- Frameworks/Libraries: ReactJS, HTML5, CSS3
- Build Tools: Node.js (latest LTS version), npm or yarn for package management
- Version Control: Git for source code management

### **2. Backend Development:**

- Programming Language: Node.js (with Express or a similar framework)
- Database: MongoDB, PostgreSQL, or other suitable databases depending on project requirements
- Server Runtime: Node.js environment

### **3. Code Analysis Tools:**

- Integrated Libraries: Custom modules for syntax checking, optimization, and complexity analysis
- APIs: Third-party APIs for additional code insights or enhancements

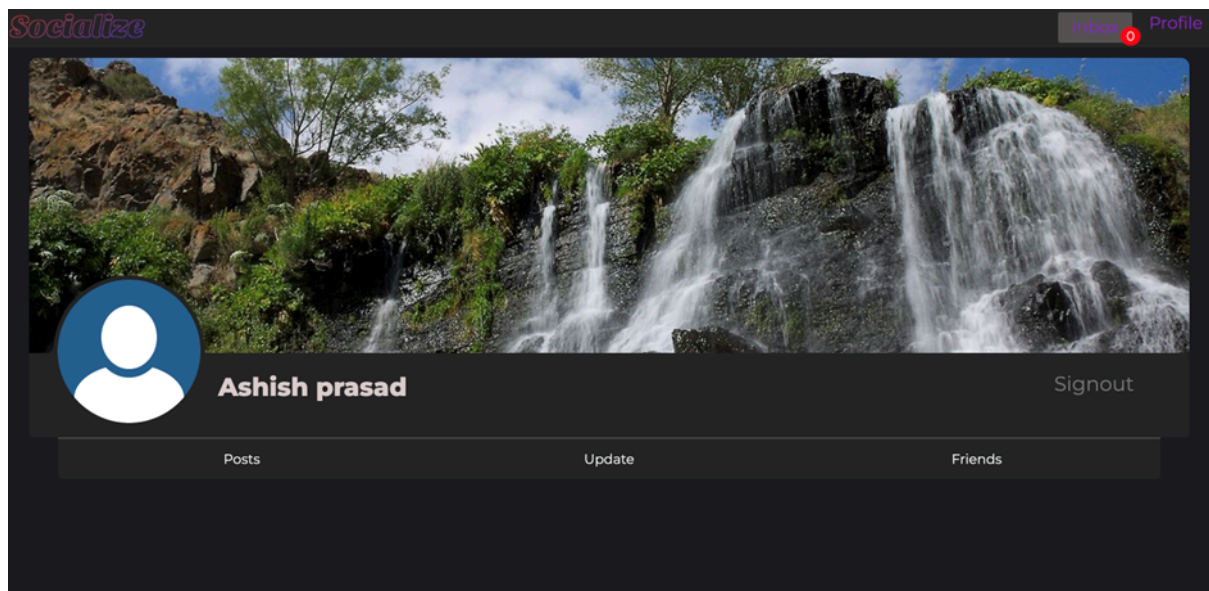
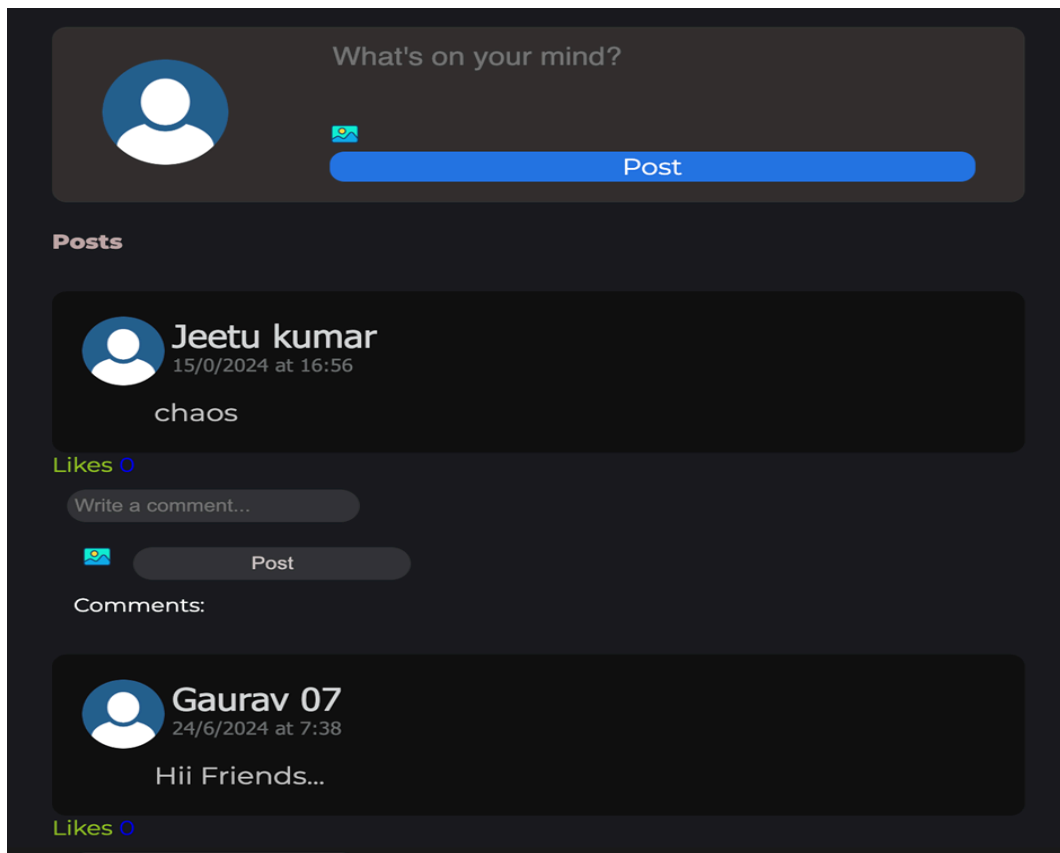
### **5. Development Environment:**

- IDE/Code Editor: Visual Studio Code, WebStorm, or any preferred code editor
- Debugging Tools: Chrome Developer Tools, React Developer Tools extension

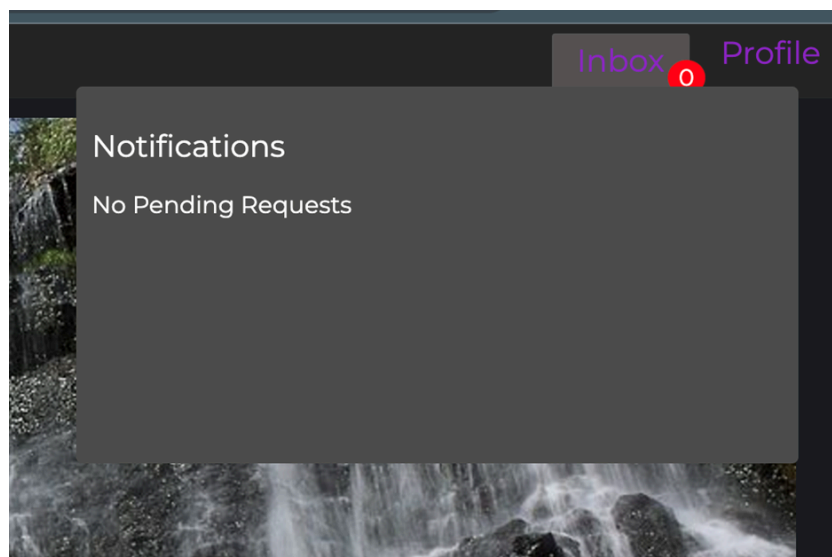
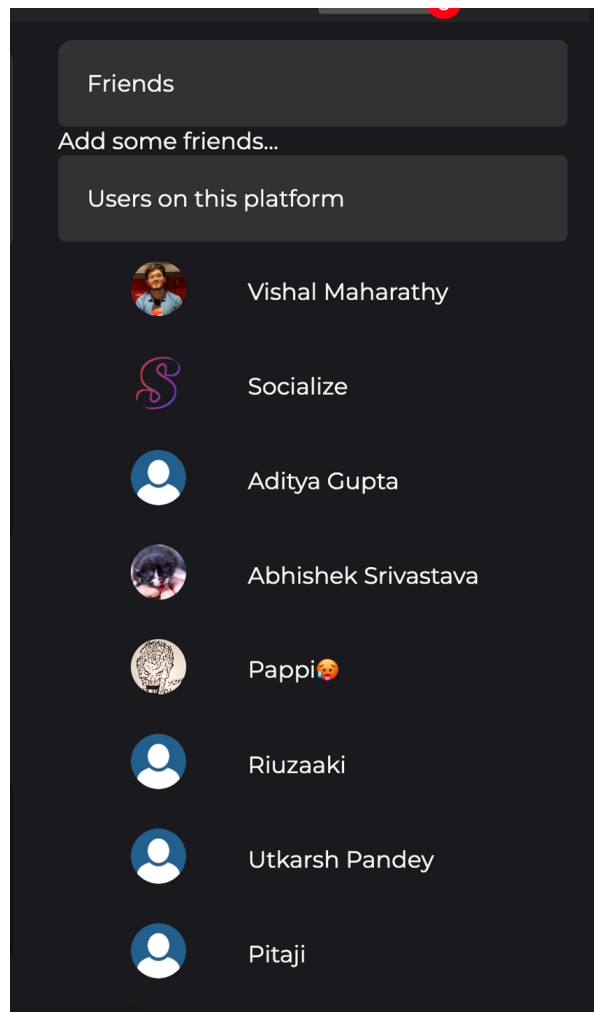
### **7. Security:**

- SSL/TLS: To secure client-server communications
- Authentication: Implementation of JWT-based user authentication

## 8. SCREENSHOTS OF WEB PORTAL







## 8. RESULTS

The project delivered successful outcomes across all modules, fulfilling the objective of creating a secure, interactive, and user-friendly platform. Each module contributed significantly to the platform's functionality, ensuring a seamless experience for users.

The **User Authentication Module** ensured robust security while maintaining ease of use. The implementation of **JSON Web Tokens (JWT)** provided stateless authentication, reducing the server's memory load and enabling scalability. Users experienced a smooth login and registration process, with sensitive credentials securely managed. Passwords were hashed using **bcrypt**, safeguarding user data even in the event of a database breach. Additionally, role-based access control successfully restricted unauthorized actions, ensuring only verified users could perform protected operations. This module offered a secure foundation for all other functionalities.

The **User Profile Management Module** was instrumental in enhancing personalization and connectivity. Users could create and update profiles with ease, including adding profile pictures, bios, and other personal details. The dynamic friend request system allowed users to send, accept, or reject connection requests, fostering community building and engagement. Changes to user profiles were reflected instantly, providing real-time updates for others to view. This module significantly improved user interaction and contributed to the overall satisfaction of the platform's users.

The **Content Interaction Module** facilitated seamless sharing and engagement among users. The ability to create posts with text, images, or multimedia enabled users to express themselves effectively. Features for liking and commenting on posts were well-received, encouraging interaction and creating a sense of community. The backend efficiently managed the storage and retrieval of content, ensuring a smooth and scalable user experience even as the platform grew. These features helped foster meaningful connections and drive consistent user engagement.

The **Real-Time Communication Module** was a highlight of the platform, providing instant notifications and updates. By integrating **WebSocket** technology, the system enabled low-latency, reliable communication. Users received immediate notifications for friend requests, comments, and other interactions without refreshing the page. This real-time

functionality significantly enhanced the interactivity and responsiveness of the platform, aligning with modern user expectations for instant feedback.

The **Frontend Design Module** delivered a visually appealing and responsive user interface. Built using **HTML** and **SCSS**, the design adapted seamlessly across devices, ensuring accessibility for users on desktops, tablets, and smartphones. The clean, intuitive design simplified navigation and improved the overall user experience. Real-time data integration with the backend ensured dynamic updates, making the interface interactive and engaging.

Finally, the **Database Management Module** efficiently handled the platform's data storage needs. Using **MongoDB**, the system managed a vast amount of user information, posts, and interactions. The database was designed with scalability in mind, supporting fast and reliable queries even under heavy user loads. The integration of indexing techniques further optimized data retrieval, contributing to the platform's high performance.

In conclusion, the project successfully achieved its goals by delivering a secure, engaging, and user-centric platform. The combination of well-implemented modules ensured scalability, responsiveness, and user satisfaction, setting a strong foundation for future growth and enhancements.

## 9. REFERENCES

### 1. **React Documentation**

React, a JavaScript library for building user interfaces, was used for the frontend development of the social platform. Best practices and examples were followed from the official React documentation.

<https://reactjs.org/docs/getting-started.html>

### 2. **Material UI Documentation**

Material UI, a React component library, was utilized to create visually appealing and responsive UI components for the platform. Guidance and components were referenced from its official documentation.

<https://mui.com/material-ui/getting-started/overview/>

### 3. **HTML5 Documentation**

The HTML5 standard was used to structure the platform's frontend. Semantic and accessible markup was ensured by referring to the official HTML5 guide.

<https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5>

### 4. **CSS Reference**

Custom CSS was implemented to enhance styling and responsiveness. The W3Schools CSS reference was used for syntax and best practices.

<https://www.w3schools.com/css/>

### 5. **Node.js Documentation**

Node.js was used to build the server-side application. Its official documentation provided insights into runtime features and module management.

<https://nodejs.org/en/docs/>

### 6. **Express Documentation**

Express.js, a Node.js framework, was employed for creating the backend API. Routes, middleware, and best practices were implemented as per the official guide.

<https://expressjs.com/en/starter/installing.html>

## 7. **MongoDB Documentation**

MongoDB served as the database solution for the platform. The MongoDB documentation was referred to for setting up collections, queries, and efficient indexing.

<https://www.mongodb.com/docs/manual/>

## 8. **JWT Documentation**

JSON Web Tokens (JWT) were used for authentication and secure data transmission. Official documentation guided the implementation of token-based security.

<https://jwt.io/introduction>

## 9. **WebSocket Documentation**

WebSocket was integrated for real-time communication and instant notifications. Its official documentation provided examples for establishing persistent connections.

<https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>

## 10. **Bcrypt Documentation**

Bcrypt was used for hashing user passwords securely. The official library documentation guided the integration and implementation of secure password storage.

<https://www.npmjs.com/package/bcrypt>

## 11. **React Router Documentation**

React Router was utilized for navigation between pages in the platform. Official documentation provided guidance on setting up routes and handling URL paths.

<https://reactrouter.com/en/main>

## 12. **React Query Documentation**

React Query was implemented for efficient data fetching and caching, simplifying API integration. Its official documentation was used for best practices.

<https://react-query-v3.tanstack.com/>

### 13. **Socket.IO Documentation**

Socket.IO, a library for real-time web applications, was used to extend WebSocket capabilities. Its documentation provided solutions for event-based communication.

<https://socket.io/docs/v4/>

### 14. **SCSS Documentation**

SCSS was employed to enhance the CSS styling process. Features like nesting, variables, and mixins were implemented following the official SCSS guide.

<https://sass-lang.com/documentation>

### 15. **OAuth Documentation**

OAuth 2.0 was used to integrate third-party authentication, such as Google or Facebook login. The official OAuth documentation provided a foundation for secure and standardized integration.

<https://oauth.net/2/>