SOCIAL MEDIA

**A Project Report**

Submitted in partial fulfillment of the Requirements for the award of the Degree of

### BACHELOR OF COMPUTER APPLICATION)

**By**

Nitinkumar das

Seat Number : 91355

**Under the esteemed guidance of**

**Mrs. Deepa Mishra**

**Assistant Professor, Department of Information Technology**

****

**DEPARTMENT OF INFORMATION TECHNOLOGY**

### Sahyog Pratishthan’s

***(Affiliated to Kavikulaguru Kalidas Sanskrit University)***

**THANE, 400601 MAHARASHTRA**

**202-24**

### SAHYOG COLLEGE

***(Affiliated to Kavikulaguru Kalidas Sanskrit University)***

**THANE-MAHARASHTRA-400601**

### DEPARTMENT OF INFORMATION TECHNOLOGY (14 BOLD, CAPS)

****

**CERTIFICATE**

This is to certify that the project entitled, **"Social Media"**, is bonafied work of **NITINKUMAR DAS** bearing Seat.No: 91355 submitted in partial fulfillment of the requirements for the award of degree of BACHELOR’S OF COMPUTER APPLICATION Kavikulguru kalidas Sanskrit University

**Internal Guide Coordinator**

**External Examiner**

**Date: College Seal**

# Abstract

This social media application offers a multifaceted platform for user interaction and community building. Its key features include: Personalization: User accounts facilitate registration and sign-in, allowing individuals to establish their online presence. Content Engagement: Users can actively respond to others' content through likes, dislikes, and comments. Content Management: Individuals have control over their own content, with the ability to upload, edit, and delete posts, enabling self-expression and information sharing. Profile Management: Users can personalize their profiles with profile pictures, potentially adding details to introduce themselves to the community. Social Connections: The application empowers users to forge social connections by following and unfollowing other users, building personalized networks. Profile Discovery: Users can explore and access other users' profiles and potential connection building.

# ACKNOWLEDGEMENT

**We project members:**

Nitinkumar : 2021018100091192

Krish jain :

Vishnu kudikala :

Manas Dubey :

Hereby declare that this project work entitled “Social Media” was carried out by us under the supervision of Prof. Deepa Mishra, Department of IT, Kavikulguru Kalidas Sanskrit University. This project work is submitted to Department of Information Technology during the academic year 2023-24.

# DECLARATION

I here by declare that the project entitled, “**Social Media**” done at College Campus, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of

**BACHELOR OF COMPUTER APPLICATION** to be submitted as final semester project as part of our curriculum.

**Name and Signature of the Student**

# TABLE OF CONTENTS (20bold, caps, centered)

Chapter 1: Introduction----------------------------------------------------------------------1

* 1. Background------------------------------------------------------------------------------1.1

### Objectives--------------------------------------------------------------------------------1.2

* 1. Purpose and Scope----------------------------------------------------------------------1.3
     1. Purpose-----------------------------------------------------------------------------1.2.1
     2. Scope-------------------------------------------------------------------------------1.2.2

### Chapter 2: System Analysis-----------------------------------------------------------------2

### Proposed System--------------------------------------------------------------------------2.1

* 1. Requirement Analysis--------------------------------------------------------------------2.2

### Hardware Requirements------------------------------------------------------------------2.3

* 1. Software Requirements-------------------------------------------------------------------2.4

### Justification of selection of Technology------------------------------------------------2.5

Chapter 3: System Design--------------------------------------------------------------------3

### Module Division---------------------------------------------------------------------------3.1

* 1. Data Dictionary----------------------------------------------------------------------------3.2

### ER Diagrams-------------------------------------------------------------------------------3.3

* 1. DFD/UML Diagrams----------------------------------------------------------------------3.4

### Chapter 4: Implementation and Testing-----------------------------------------------------4

* 1. Code (Place Core segments)--------------------------------------------------------------4.1

### Testing Approach--------------------------------------------------------------------------4.2

* + 1. Unit Testing (Test cases and Test Results)--------------------------------4.2.1

### Integration System (Test cases and Test Results)-----------------------4.2.2

Chapter 5: Results and Discussions

Chapter 6: Conclusion and Future Work

Chapter 7: Reference

# Chapter 1

**Introduction**

In today's interconnected world, social media platforms have become an integral part of our daily lives, facilitating communication, collaboration, and information sharing on a global scale. Our MERN (MongoDB, Express.js, React.js, Node.js) Social Media Project aims to provide a dynamic and interactive platform for users to connect, share content, and engage with one another in real-time.

Key Features:

1. User Authentication and Profiles: Users can create accounts, customize their profiles, and manage their personal information securely. Authentication mechanisms ensure that only authorized users can access their accounts and interact with the platform's features.

2. Post Creation and Sharing: Users can Create, Delete, Like/Dislike Comments on posts.

3. Social Interactions: Users can engage with posts by liking, commenting, them.

4. Friend and Follower System: The platform facilitates connections between users through a friend and follower system. Users can follow other users to stay updated on their activities, and manage their network of connections.

5. Explore and Discover: A functionality allow users to explore a diverse range of content based on their interests, preferences, and trending topics

Technology Stack:

Our project leverages the MERN stack, comprising:

- MongoDB: A scalable and flexible NoSQL database for storing user data, posts, and other content.

- Express.js: A lightweight and minimalist web framework for building robust and scalable server-side applications.

- React.js: A powerful JavaScript library for building dynamic and interactive user interfaces.

- Node.js: A server-side JavaScript runtime environment that enables the execution of JavaScript code outside a web browser, facilitating server-side logic and API integration.

# Background

# The decision to develop a social media application using the MERN (MongoDB, Express.js, React.js, Node.js) stack is influenced by several factors. Below is a background analysis outlining the rationale behind choosing this technology stack for building a social media app:

# 1. Scalability: Social media platforms typically experience rapid user growth and need to handle large volumes of data and concurrent user interactions. The MERN stack, with its ability to horizontally scale both frontend and backend components, provides a scalable foundation for accommodating increasing user demands.

# 2. Real-time Updates: Social media apps require real-time updates to display new posts, comments, likes, and notifications as they occur. The event-driven architecture of Node.js and the reactive nature of React.js make it well-suited for implementing real-time features, ensuring a dynamic and engaging user experience.

# 3. Rich User Interfaces: Modern social media users expect visually appealing and responsive user interfaces that provide seamless navigation and interactivity. React.js, with its component-based architecture and virtual DOM, facilitates the creation of rich and interactive UI components, enabling developers to deliver a highly engaging user experience.

# 4. Data Modeling and Flexibility: Social media platforms involve complex data models with relationships between users, posts, comments, likes, and other entities. MongoDB, a NoSQL database used in the MERN stack, offers flexibility in data modeling with its document-based storage and support for nested structures, making it suitable for managing diverse and evolving data schemas common in social media apps.

# 5. Developer Productivity: The MERN stack promotes developer productivity by enabling full-stack JavaScript development. Developers can use JavaScript for both frontend and backend development, reducing context switching and allowing for code reuse across different layers of the application. Additionally, the extensive ecosystem of libraries, frameworks, and tools available for the MERN stack streamlines development tasks and accelerates time-to-market.

# 6. Community Support and Resources: The MERN stack benefits from a large and active community of developers, contributors, and resources. Developers can leverage community-maintained packages, documentation, tutorials, and forums to overcome challenges, seek guidance, and stay updated on best practices and emerging trends in web development.

# 7. Cost-Effectiveness: Using open-source technologies like the MERN stack can result in cost savings compared to proprietary solutions. The availability of free and open-source tools, frameworks, and libraries reduces licensing fees and infrastructure costs, making it an attractive choice for startups and businesses with limited budgets.

# Objectives

**Core Functionality:**

* **User Accounts:** Allow users to register, login, and manage their profiles.
* **Content Creation:** Enable users to create and share text posts, images, or even videos.
* **Feed & Engagement:** Display a dynamic feed of content from followed users, with the ability to like and comment on posts.
* **Search & Discovery:** Implement search functionality to find users and content, and potentially explore features for content discovery.

**Technical Objectives:**

* **Authentication & Authorization:** Implement secure user authentication (e.g., email/password or social login) and manage user roles/permissions.
* **Database Management:** Effectively utilize MongoDB to store user data, posts, comments, and other app-specific information.
* **API Design:** Create well-structured APIs for communication between the React frontend and the Node.js backend.

**Beyond the Basics:**

* **Focus on User Experience:** Prioritize a user-friendly and intuitive interface that fosters engagement and community building.
* **Monetization Strategy:** Explore potential monetization strategies like advertising or premium features (if applicable).

# Purpose And Scope

# 1.2.1 Purpose

# The purpose of the MERN (MongoDB, Express.js, React.js, Node.js) social media app example is multifaceted, aiming to fulfill various user needs and business objectives. Below are the primary purposes of developing such an application:

# 1. Facilitating Social Connections : The foremost purpose of the MERN social media app is to provide a platform for users to connect with friends, family, colleagues, and like-minded individuals. It serves as a digital space where users can establish and nurture social relationships through communication, interaction, and shared interests.

# 2. Content Sharing and Discovery : The app enables users to share diverse types of content, including text, images, videos, links, and more. By facilitating content creation and dissemination, the app encourages users to express themselves, share experiences, and discover new ideas, trends, and perspectives from their social network and beyond.

# 3. Enhancing Communication : A key purpose of the social media app is to facilitate communication and collaboration among users. Through features such as comments, likes, shares, and direct messaging, users can engage in conversations, exchange information, provide feedback, and support each other in a dynamic and interactive environment.

# 4. Building Communities : The app aims to foster the formation of communities based on shared interests, hobbies, professions, or geographical locations. Users can join groups, participate in discussions, attend events, and connect with others who share similar passions or affiliations, creating a sense of belonging and camaraderie.

# 5. Entertainment and Engagement : Providing entertainment and engagement opportunities is another purpose of the social media app. Users can consume entertaining content, discover viral trends, participate in challenges, quizzes, and polls, and engage with interactive features that captivate their attention and sustain their interest in the platform.

# 6. Personal Branding and Networking : For individuals, professionals, and businesses, the app serves as a platform for personal branding, networking, and self-promotion. Users can showcase their skills, achievements, and expertise, build professional networks, and explore career opportunities by leveraging the visibility and reach offered by the platform.

# 7. Data Collection and Analysis : From a business perspective, the social media app serves as a valuable source of user data and insights. By analyzing user behavior, preferences, interactions, and demographics, the app generates valuable insights that can inform marketing strategies, product development decisions, and content optimization efforts.

# 8. Generating Revenue : Finally, the social media app may have a revenue generation component, whether through advertising, sponsored content, premium subscriptions, e-commerce integrations, or other monetization strategies. By creating value for users and attracting a large and engaged audience, the app can generate revenue streams to sustain its operations and fuel further growth and innovation.

# Scope

# 1. User Authentication and Profiles :

# - User registration and login functionality.

# - User profile creation and management, including profile pictures, bios, and contact information.

# - Account settings and preferences customization.

# 2. Content Management :

# - Creation, editing, and deletion of posts comprising text, images, videos, links, and other multimedia content.

# - Post categorization, tagging, and privacy settings (public, private, friends-only).

# - Content discovery through news feeds, trending topics, and personalized recommendations.

# 3. Social Interactions :

# - Liking, commenting, sharing, and bookmarking posts.

# - Following and unfollowing other users.

# - Receiving notifications for likes, comments, mentions, and new followers.

# 4. Messaging and Communication :

# - Direct messaging between users.

# - Group messaging functionality.

# - Notifications for new messages and message requests.

# 5. Friend and Follower System:

# - Sending and accepting friend requests.

# - Managing friend lists and followers.

# - Filtering and sorting content based on friend/follower activity.

# 6. Search and Discovery:

# - Search functionality for users, posts, hashtags, and topics.

# - Advanced search filters based on criteria such as location, date, and content type.

# - Explore section featuring trending posts, popular users, and suggested content.

# 7. Analytics and Insights :

# - Tracking user engagement metrics, such as likes, comments, shares, and views.

# - Providing insights into user behavior, demographics, and content preferences.

# - Generating reports and visualizations to aid decision-making and content optimization

# 8. Security and Privacy:

# - Implementation of secure authentication mechanisms, such as OAuth or JWT.

# - Data encryption for sensitive user information.

# - Role-based access control to manage user permissions and privileges.

# 9. Moderation and Content Control:

# - Reporting and flagging of inappropriate or abusive content.

# - Automated and manual content moderation tools.

# - Compliance with community guidelines and legal regulations regarding content moderation and user safety.

# 10. Cross-Platform Compatibility:

# - Development of responsive web design for seamless user experience across desktop and mobile devices.

# - Consideration for native mobile app development or progressive web app (PWA) implementation for enhanced performance and accessibility.

# 11. Monetization Strategies:

# - Integration of advertising networks for displaying ads within the app.

# - Offering premium features or subscription plans for enhanced user experience.

# - Partnering with brands for sponsored content or influencer marketing campaigns.

# 12. Scalability and Performance:

# - Designing a scalable architecture capable of handling increased user traffic and data volume.

# - Optimizing performance through caching, load balancing, and database indexing.

# - Conducting stress testing and performance monitoring to identify and address bottlenecks.

# 13. Documentation and Support:

# - Creation of comprehensive documentation covering app functionality, APIs, and deployment instructions.

# - Providing user support through FAQs, tutorials, and responsive customer service channels.

# Chapter 2

**System Analysis**

* 1. **Proposed System**

1. Enhanced User Experience:

- Intuitive Interface: Design a user-friendly interface with intuitive navigation and visually appealing design elements to enhance the user experience.

- Responsive Design: Ensure that the application is responsive and accessible across various devices, including desktops, tablets, and smartphones.

- Personalized Feeds: Implement algorithms to curate personalized content feeds based on user preferences, interests, and social connections.

2. Robust User Management:

- Seamless Registration and Authentication: Streamline the registration process and provide multiple authentication options (e.g., email, social media login) for user convenience.

- Profile Customization: Allow users to customize their profiles with profile pictures, and other personal information.

3. Engaging Social Interactions:

- Interactive Features: Enable users to like, comment on, and share content to foster engagement and interaction within the community.

- Notifications and Activity Feeds: Notify users about relevant activities, such as new likes, comments, messages,to keep them engaged and informed.

6. Enhanced Security and Privacy:

- Secure Data Encryption: Implement robust encryption mechanisms to protect sensitive user data, both in transit and at rest.

- Regular Security Audits: Conduct regular security audits and vulnerability assessments to identify and address potential security risks proactively.

7. Scalability and Performance Optimization:

- Scalable Architecture: Design the system with scalability in mind to accommodate growing user bases and increasing data volumes.

- Performance Optimization: Optimize code, database queries, and network requests to minimize latency and ensure smooth performance even under high load conditions.

8. Analytics and Insights:

- Comprehensive Analytics Dashboard: Provide administrators with access to a comprehensive analytics dashboard to monitor user engagement, content performance, and platform usage metrics.

- Data-driven Decision Making: Utilize analytics insights to make data-driven decisions for feature enhancements, content moderation, and community management.

9. Continuous Improvement and Innovation:

- Agile Development Methodology: Adopt an agile development approach to iterate quickly, gather feedback from users, and continuously improve the platform.

- Regular Feature Updates: Roll out regular feature updates and enhancements based on user feedback, industry trends, and emerging technologies.

- Community Engagement: Foster a sense of community and collaboration among users by actively soliciting feedback, hosting events, and recognizing user contributions.

10. Documentation and Support:

- Comprehensive Documentation: Provide detailed documentation for developers, administrators, and end-users to facilitate onboarding, troubleshooting, and platform customization.

- Dedicated Support Channels: Offer responsive customer support channels, including email support, live chat, and community forums, to address user inquiries and issues promptly.

* 1. **Requirement Analysis**

1. **Functional Requirements**:
   * **User Authentication and Authorization**:
     + Implement user registration and login.
     + Ensure secure access to user-specific features.
   * **User Profile Management**:
     + Allow users to create and edit profiles.
     + Upload profile pictures.
     + Specify personal information (bio, interests).
   * **Posts and Feeds**:
     + Users can create, and delete posts.
     + Display a feed with posts from followed users.
     + Implement real-time updates for new posts.
   * **Likes and Comments**:
     + Users can like and comment on posts.
     + Track the number of likes per post.
     + Enable threaded comments.
   * **Followers and Following**:
     + Users can follow/unfollow other users.
     + Show followers and following counts.
   * **Notifications**:
     + Notify users of new likes, comments, or followers.
     + Implement real-time notifications.
   * **Multimedia Content**:
     + Support image uploads.
     + Optimize media storage and retrieval.
   * **Reporting and Moderation**:
     + Implement reporting mechanisms for inappropriate content.
     + Have moderation tools to handle violations.
2. **Non-Functional Requirements**:
   * **Performance and Scalability**:
     + Optimize database queries for efficient data retrieval.
     + Handle concurrent users and large datasets.
   * **Security**:
     + Protect against common vulnerabilities.
     + Validate user inputs.
   * **Usability and User Experience**:
     + Design an intuitive and responsive user interface.
     + Ensure smooth navigation and interactions.
   * **Compatibility**:
     + Test across different browsers and devices.
     + Consider mobile responsiveness.
   * **Maintainability and Extensibility**:
     + Write clean, modular code.
     + Document APIs and components.
     + Plan for future enhancements.
3. **Technology Stack**:
   * **Backend**:
     + Use **Express.js** for API development.
     + Set up **MongoDB** for data storage.
     + Handle file uploads using **Cloudinary**.
   * **Frontend**:
     + Build the user interface with **React**.
     + Use **Fetch** for API communication.
     + Implement routing with **React Router**.
4. **Testing and Quality Assurance**:
   * Develop unit tests for backend and frontend components.
   * Conduct integration testing.
   * Ensure error handling and graceful degradation.
   1. **Hardware Requirements**

* Processor AMD Ryzen or Intel based
* Storage SSD Storage for faster read/write operations improved overall performance
* RAM 8gb
  1. **Software Requirements**
* Frontend : HTML ,CSS ,JavaScript ,React.js ,Bootstrap
* Backend : Node.js, Express.js ,MongoDB
* IDE : Microsoft Visual Studio Code
* Tool : PostMan ,ThunderClient
* Additional Libraries/Frameworks : Mongoose , bcrypt , React-toastify.
* Operating System : Window 10 or Higher

# Chapter 3

**System Design (20 bold, centered)**

* 1. **Module Division**

3.1.1 Authentication Module:

* Responsible for user registration, login, and authentication.
* Includes features such as Google Authentication, password reset, and social media authentication.
* Manages user sessions and access control.

3.1.2 User Profile Module:

* Handles user profile creation, management, and customization.
* Allows users to edit profile information, upload profile pictures, and set privacy settings.
* Manages connections between users (e.g., friend requests, followers, following).

3.1.3 Social Interaction Module:

* Enables social interactions between users, such as liking, commenting, and sharing content.
* Implements real-time notifications for user interactions (e.g., new likes, comments, mentions).

3.1.4 UI/UX Module:

* Responsible for designing and implementing the user interface (UI) and user experience (UX) of the application.
* Includes frontend components, layouts, stylesheets, and interactive elements.
* Ensures consistency, accessibility, and responsiveness across different devices and screen sizes.
  1. **Data Dictionary**

1. User:

-Attributes:

- UserID (Primary Key): Unique identifier for each user.

- Username: User's chosen username for identification.

- Email: User's email address for communication and authentication.

- Password: Hashed password for user authentication.

- FullName: User's full name or display name.

- Bio: User's biography or profile description.

- ProfilePicture: URL or reference to the user's profile picture.

- CreatedAt: Timestamp indicating when the user account was created.

- Relationships:

- One-to-Many with Posts: Each user can create multiple posts.

- One-to-Many with Comments: Each user can post multiple comments.

- Many-to-Many with Followers/Following: Users can follow/unfollow other users.

2. Post:

- Attributes:

- PostID (Primary Key): Unique identifier for each post.

- UserID (Foreign Key): ID of the user who created the post.

- Content: Text content of the post.

- Media: media attached to the post

- LikesCount: Number of likes received on the post.

- CommentsCount: Number of comments posted on the post.

- CreatedAt: Timestamp indicating when the post was created.

- Relationships:

- Many-to-One with User: Each post is created by one user.

3. Comment:

- Attributes:

- CommentID (Primary Key): Unique identifier for each comment.

- PostID (Foreign Key): ID of the post to which the comment belongs.

- UserID (Foreign Key): ID of the user who posted the comment.

- Content: Text content of the comment.

- CreatedAt: Timestamp indicating when the comment was posted.

- Relationships:

- Many-to-One with Post: Each comment belongs to one post.

- Many-to-One with User: Each comment is posted by one user.

4. Followers/Following:

- Attributes:

- FollowerID (Primary Key): ID of the user following another user.

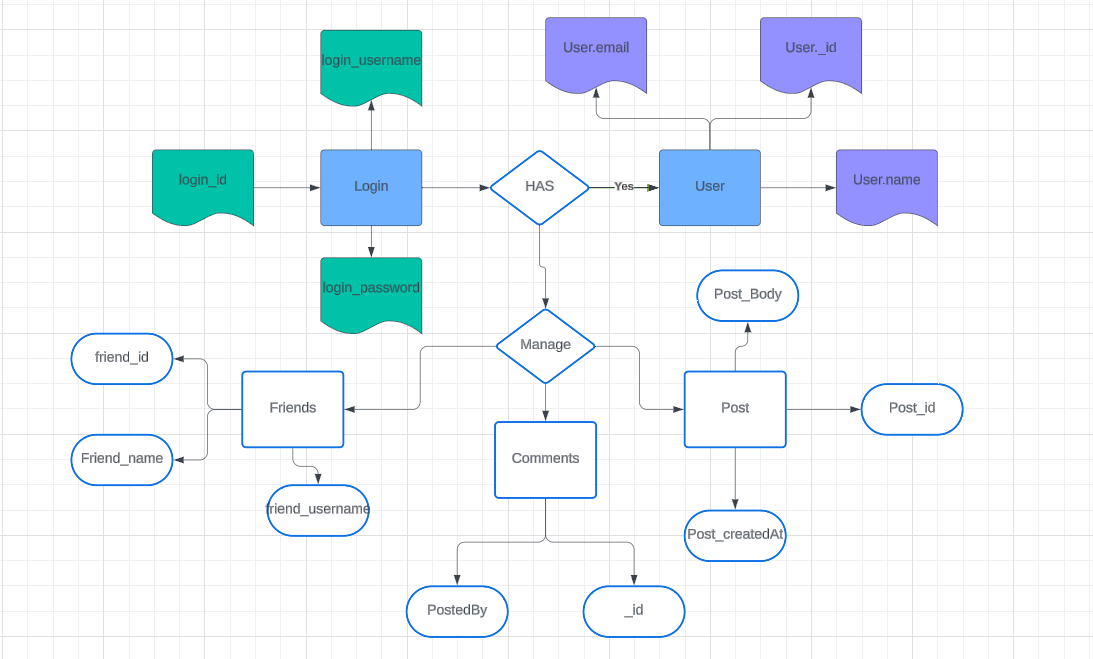
- FollowingID (Primary Key): ID of the user being followed.

- CreatedAt: Timestamp indicating when the follow relationship was established.

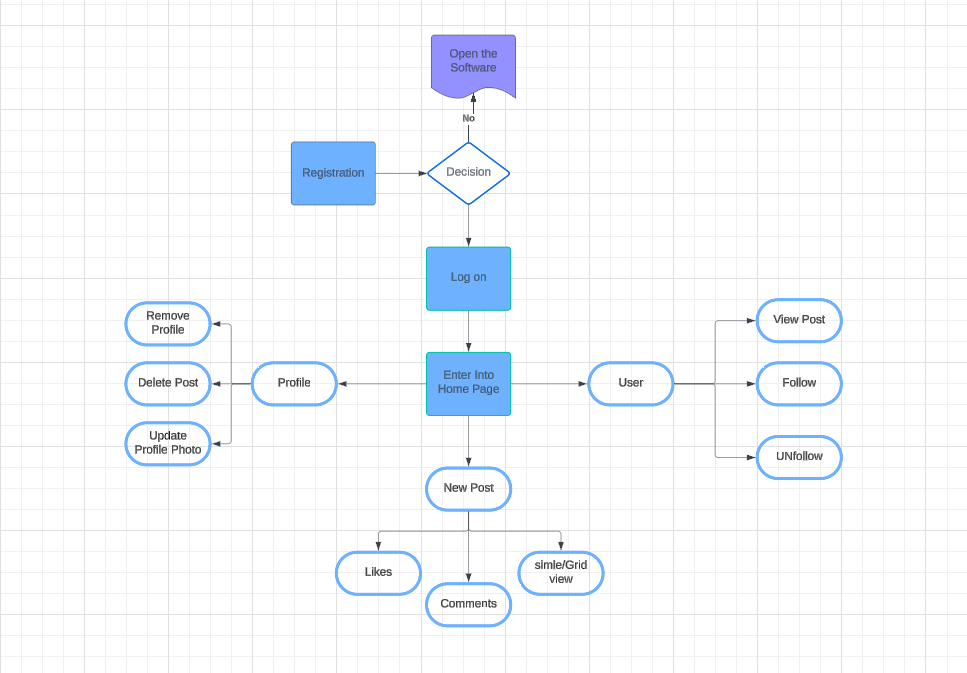
- Relationships:

- Many-to-Many with User: Users can follow/unfollow other users.

* 1. **E-R Diagrams**

****

* 1. **Data Flow Diagrams / UML**

****

**Data Flow**

User actions such as posting content, commenting, or liking trigger requests from the frontend application to the backend server.

The backend server processes these requests, performs necessary operations (e.g., creating posts, updating user profiles), and interacts with the database.

Data retrieved from the database is sent back to the frontend as a response, updating the user interface with the latest information.

updates from other users trigger real-time updates to the user interface

# Chapter 4

**Implementation and Testing (20 bold, centered)**

### Code

### App.js file

import { createContext, useEffect, useState } from "react";

import "./App.css";

import "./style/Navbar.css";

import Navbar from "./components/Navbar";

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Home from "./components/Home";

import SignUp from "./components/SignUp";

import SignIn from "./components/SignIn";

import Profile from "./components/Profile";

import { ToastContainer, toast } from "react-toastify";

import "react-toastify/dist/ReactToastify.css";

import Createpost from "./components/Createpost";

import { LoginContext } from "./context/LoginContext";

import Model from "./components/Model";

import UserProfile from "./components/UserProfile";

import MyFollowingPost from "./components/MyFollowingPost";

import { GoogleOAuthProvider } from "@react-oauth/google";

import Loader from "./components/Loader";

// import "bootstrap/dist/css/bootstrap.min.css";

function App() {

  const [userLogin, setUserLogin] = useState(false);

  const [model, setModel] = useState(false);

  const [loading, setLoading] = useState(true);

  useEffect(() => {

    const timeout = setTimeout(() => {

      setLoading(false);

    }, 2000); // Adjust as needed

    return () => clearTimeout(timeout);

  }, []);

  // useEffect

  return (

    <BrowserRouter>

      <div className="App">

        <Loader loading={loading} />

        <GoogleOAuthProvider clientId="404752276702-b2ns027c0fin75hlb2e409h82vmh7fj8.apps.googleusercontent.com">

          <LoginContext.Provider value={{ setUserLogin, setModel }}>

            <Routes>

              <Route path="/" element={<Home />}></Route>

              <Route path="/signup" element={<SignUp />}></Route>

              <Route path="/signin" element={<SignIn />}></Route>

              <Route exact path="/profile" element={<Profile />}></Route>

              <Route path="/createpost" element={<Createpost />}></Route>

              <Route path="/profile/:userId" element={<UserProfile />}></Route>

              <Route

                path="/followingpost"

                element={<MyFollowingPost />}

              ></Route>

            </Routes>

            <ToastContainer theme="dark" position="bottom-right" />

            {/\* <Model></Model> \*/}

            {model && <Model setmodel={setModel}></Model>}

            {window.location.pathname !== "/signin" &&

              window.location.pathname !== "/signup" && (

                <Navbar login={userLogin} />

              )}

          </LoginContext.Provider>

        </GoogleOAuthProvider>

      </div>

    </BrowserRouter>

  );

}

export default App;

### Home.jsx

import React, { useEffect, useState } from "react";

import "../style/Home.css";

import { Link, useNavigate } from "react-router-dom";

import { ToastContainer, toast } from "react-toastify";

// import { IoClose } from "react-icons/io5";

import { IoMdClose } from "react-icons/io";

// import { FaRegComment } from "react-icons/fa";

import { MdOutlineModeComment } from "react-icons/md";

import { FaRegBookmark } from "react-icons/fa";

import { RxGrid } from "react-icons/rx";

// import { SiCodesandbox } from "react-icons/si";

import { MdCheckBoxOutlineBlank } from "react-icons/md";

const Home = () => {

  let picLink = "https://cdn-icons-png.flaticon.com/128/847/847969.png";

  const navigate = useNavigate();

  const [data, setData] = useState([]);

  const [comment, setComment] = useState([]);

  const [show, setShow] = useState(false);

  const [item, setItem] = useState([]);

  const [gridView, setGridView] = useState(true);

  const notifyA = (err) => toast.error(err);

  const notifyB = (err) => toast.success(err);

  useEffect(() => {

    const token = localStorage.getItem("jwt");

    if (!token) {

      navigate("./signin");

    }

    fetch("http://localhost:5000/allposts", {

      headers: {

        Authorization: "Bearer" + localStorage.getItem("jwt"),

      },

    })

      .then((res) => res.json())

      .then((result) => {

        console.log(result);

        setData(result);

      })

      .catch((err) => console.log(err));

  }, []);

  const likePost = (id) => {

    fetch("http://localhost:5000/like", {

      method: "put",

      headers: {

        "Content-Type": "application/json",

        Authorization: "Bearer" + localStorage.getItem("jwt"),

      },

      body: JSON.stringify({

        postId: id,

      }),

    })

      .then((res) => res.json())

      .then((result) => {

        setData((prevData) =>

          prevData.map((post) => (post.\_id === result.\_id ? result : post))

        );

        console.log(result);

      });

  };

  const unlikePost = (id) => {

    fetch("http://localhost:5000/unlike", {

      method: "put",

      headers: {

        "Content-Type": "application/json",

        Authorization: "Bearer" + localStorage.getItem("jwt"),

      },

      body: JSON.stringify({

        postId: id,

      }),

    })

      .then((res) => res.json())

      .then((result) => {

        setData((prevData) =>

          prevData.map((post) => (post.\_id === result.\_id ? result : post))

        );

        console.log(result);

      });

  };

  const makeComment = (text, id) => {

    console.log(comment);

    fetch("http://localhost:5000/comment", {

      method: "put",

      headers: {

        "Content-Type": "application/json",

        Authorization: "Bearer" + localStorage.getItem("jwt"),

      },

      body: JSON.stringify({

        text: text,

        postId: id,

      }),

    })

      .then((res) => res.json())

      .then((result) => {

        const newData = data.map((posts) => {

          if (posts.\_id == result.\_id) {

            return result;

          } else {

            return posts;

          }

        });

        setData(newData);

        setComment("");

        notifyB("commented successfully");

        console.log(result);

      });

  };

  const viewComment = (posts) => {

    if (show) {

      setShow(false);

    } else {

      setShow(true);

      setItem(posts);

      console.log(item);

    }

  };

  const toggleView = () => {

    setGridView(!gridView);

  };

  return (

    <div className={gridView ? "home grid-view" : "home"}>

      <button className="grid-btn" onClick={toggleView}>

        {gridView ? <RxGrid /> : <MdCheckBoxOutlineBlank />}

      </button>

      {data.map((posts) => {

        return (

          <div className={gridView ? "card" : "card grid-card"}>

            <div className="scroll-effect">

              <div className="card-image">

                <img

                  src={posts.photo}

                  alt=""

                  onClick={() => {

                    viewComment(posts);

                  }}

                />

              </div>

              <div className="card-header">

                <div className="like-comment">

                  <div className="card-pic">

                    <img

                      src={

                        posts.postedBy.Photo ? posts.postedBy.Photo : picLink

                      }

                      // onDoubleClick={() => {

                      //   likePost(posts.\_id);

                      // }}

                      alt=""

                    />

                    <h5>

                      <Link

                        to={`/profile/${posts.postedBy.\_id}`}

                        style={{ color: "#000" }}

                      >

                        {" "}

                        {posts.postedBy.name}

                      </Link>

                    </h5>

                  </div>

                  <div className="card-content">

                    <p id="like">

                      <span> {posts.likes.length}</span> <span>likes</span>

                    </p>

                    {posts.likes.includes(

                      JSON.parse(localStorage.getItem("user")).\_id

                    ) ? (

                      <span

                        className="material-symbols-outlined material-symbols-outlined-red"

                        onClick={() => {

                          unlikePost(posts.\_id);

                        }}

                      >

                        favorite

                      </span>

                    ) : (

                      <span

                        className="material-symbols-outlined"

                        id="fav"

                        onClick={() => {

                          likePost(posts.\_id);

                        }}

                      >

                        favorite

                      </span>

                    )}

                    <MdOutlineModeComment

                      id="message"

                      onClick={() => {

                        viewComment(posts);

                      }}

                    />

                    {/\* <FaRegBookmark id="bookMark" /> \*/}

                    {/\* <p

                      style={{ fontWeight: "bolder", cursor: "pointer" }}

                      onClick={() => {

                        viewComment(posts);

                      }}

                    >

                      {" "}

                      view all comments

                    </p> \*/}

                  </div>

                </div>

              </div>

              <div className="add-comment">

                <span className="material-symbols-outlined">mood</span>

                <input

                  type="text"

                  placeholder="Add a comment"

                  value={comment}

                  onChange={(e) => {

                    setComment(e.target.value);

                  }}

                />

                <button

                  type="button"

                  className="btn btn-primary post-comment"

                  onClick={() => {

                    if (comment.trim() !== "") {

                      // Check if comment is not empty or only contains whitespace

                      makeComment(comment, posts.\_id);

                      setComment(""); // Clear the input field after posting

                    } else {

                      notifyA("Please enter a comment before posting."); // Show an alert if comment is empty

                    }

                  }}

                >

                  Post

                </button>

              </div>

            </div>

          </div>

        );

      })}

      {show && (

        <div className="showComment">

          <div className="container">

            <div className="postPic">

              <img src={item.photo} alt="" />

            </div>

            <div className="details">

              <div className="card-header">

                <div className="card-pic">

                  <img

                    src={item.postedBy.Photo ? item.postedBy.Photo : picLink}

                    alt=""

                  />

                </div>

                <h5 style={{ color: "#000" }}>

                  {" "}

                  <Link

                    to={`/profile/${item.postedBy.\_id}`}

                    style={{ color: "#000" }}

                  >

                    {" "}

                    {item.postedBy.name}

                  </Link>

                </h5>

              </div>

              <div className="comment-section">

                {item.comments.length === 0 ? (

                  <p

                    style={{

                      display: "flex",

                      alignItems: "center",

                      justifyContent: "center",

                      height: "75%",

                      fontWeight: "bolder",

                      fontSize: "2rem",

                    }}

                  >

                    No comments yet.

                  </p>

                ) : (

                  item.comments.map((comment) => (

                    <p className="add-comment" key={comment.\_id}>

                      <div className="card-pic">

                        <img

                          src={

                            comment.postedBy.Photo

                              ? comment.postedBy.Photo

                              : picLink

                          }

                          alt=""

                        />

                      </div>

                      <Link

                        to={`/profile/${comment.postedBy.\_id}`}

                        className="user-comment"

                        style={{

                          fontWeight: "bolder",

                          margin: "5px",

                          color: "#000",

                        }}

                      >

                        {comment.postedBy.name}

                      </Link>

                      <span

                        className="user-comment-text"

                        style={{ color: "#000" }}

                      >

                        {comment.comment}

                      </span>

                    </p>

                  ))

                )}

              </div>

              <div className="card-content">

                <p id="like">{item.likes.length} likes</p>

                <p id="caption">{item.body}</p>

              </div>

              <div className="add-comment">

                <span className="material-symbols-outlined">mood</span>

                <input

                  type="text"

                  placeholder="Add a comment"

                  value={comment}

                  onChange={(e) => {

                    setComment(e.target.value);

                  }}

                />

                <button

                  type="button"

                  className="btn btn-primary post-comment"

                  onClick={() => {

                    if (comment.trim() !== "") {

                      // Check if comment is not empty or only contains whitespace

                      makeComment(comment, posts.\_id);

                      setComment(""); // Clear the input field after posting

                    } else {

                      notifyA("Please enter a comment before posting."); // Show an alert if comment is empty

                    }

                  }}

                >

                  Post

                </button>

              </div>

            </div>

          </div>

          <div className="close-comment">

            {/\* <IoClose className="c" /> \*/}

            <IoMdClose className="c" onClick={() => setShow(false)} />

            {/\* alternate {()=>viewComment()} \*/}

          </div>

        </div>

      )}

    </div>

  );

};

export default Home;

### Backend app.js

const express = require("express");

const app = express();

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

const mongoose = require("mongoose");

const { mongoUrl } = require("./keys");

const cors = require("cors");

app.use(cors());

require("./models/model");

require("./models/post");

app.use(express.json());

app.use(require("./routes/auth"));

app.use(require("./routes/createpost"));

app.use(require("./routes/user"));

mongoose.connect(mongoUrl);

mongoose.connection.on("connected", () => {

  console.log("successfully connected");

});

mongoose.connection.on("error", () => {

  console.log("not connected");

});

app.listen(PORT, () => {

  console.log("server is running on" + PORT);

});

### Testing Approach

### Unit Testing

-Backend (Node.js/Express.js):

Routes: Test each route with different inputs, including valid and invalid data, to ensure proper handling of requests.

Controllers/Handlers: Test each controller or handler to ensure they behave as expected.

Models: Test database operations like CRUD operations ensuring that data is being stored and retrieved correctly.

-Frontend (React):

Components: Test each React component in isolation using tools like Jest and Enzyme. Verify that components render correctly, handle user interactions, and update state as expected.

API Calls: API calls to ensure that they are dispatched correctly and update state appropriately.

Authentication/Authorization:

Test authentication and authorization flows, ensuring that only authenticated users can access certain routes or perform specific actions.

* + 1. **Integration Testing**

-Testing Database Operations:

Use a testing database or a separate test environment to perform integration tests against the actual database.

Test CRUD (Create, Read, Update, Delete) operations on your database models, ensuring that data is stored and retrieved correctly.

Verify that relationships between different entities (e.g., users, posts, comments) are maintained correctly.

-Testing Authentication and Authorization:

Test the authentication and authorization flows by making requests to protected endpoints with valid and invalid authentication tokens.

Ensure that only authenticated users can access protected resources and that unauthorized users receive appropriate error responses.

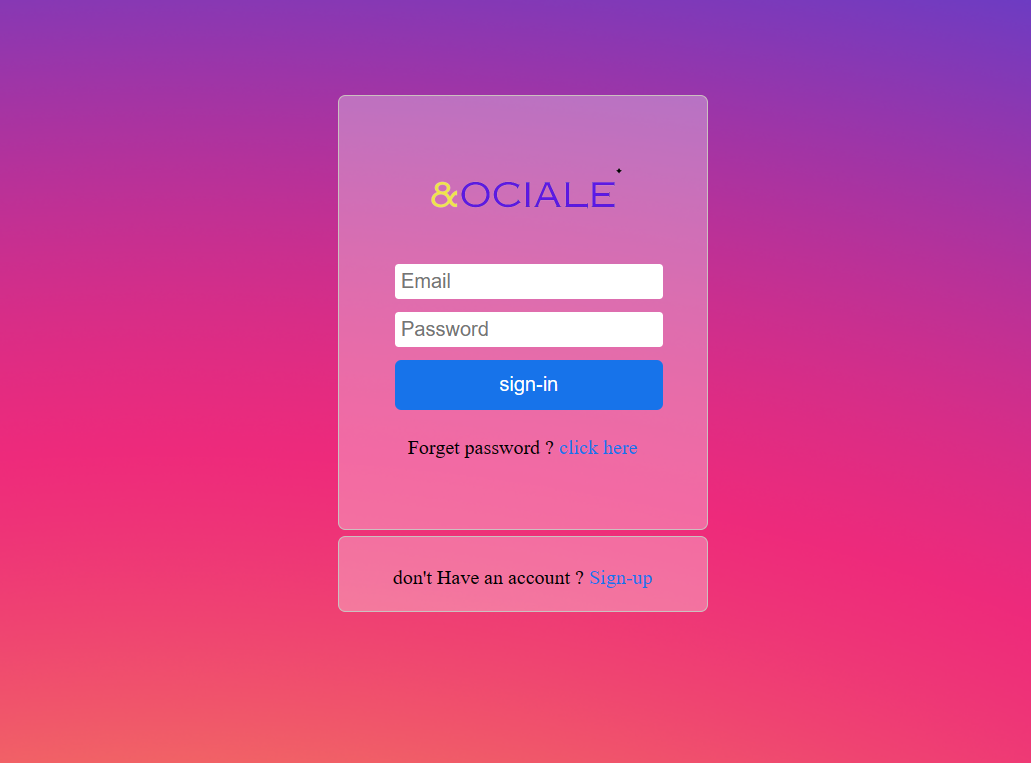
-Testing Error Handling:

Ensure that error messages are displayed correctly to users and that the application gracefully handles unexpected situations.

# Chapter 5

**Results and Discussions**

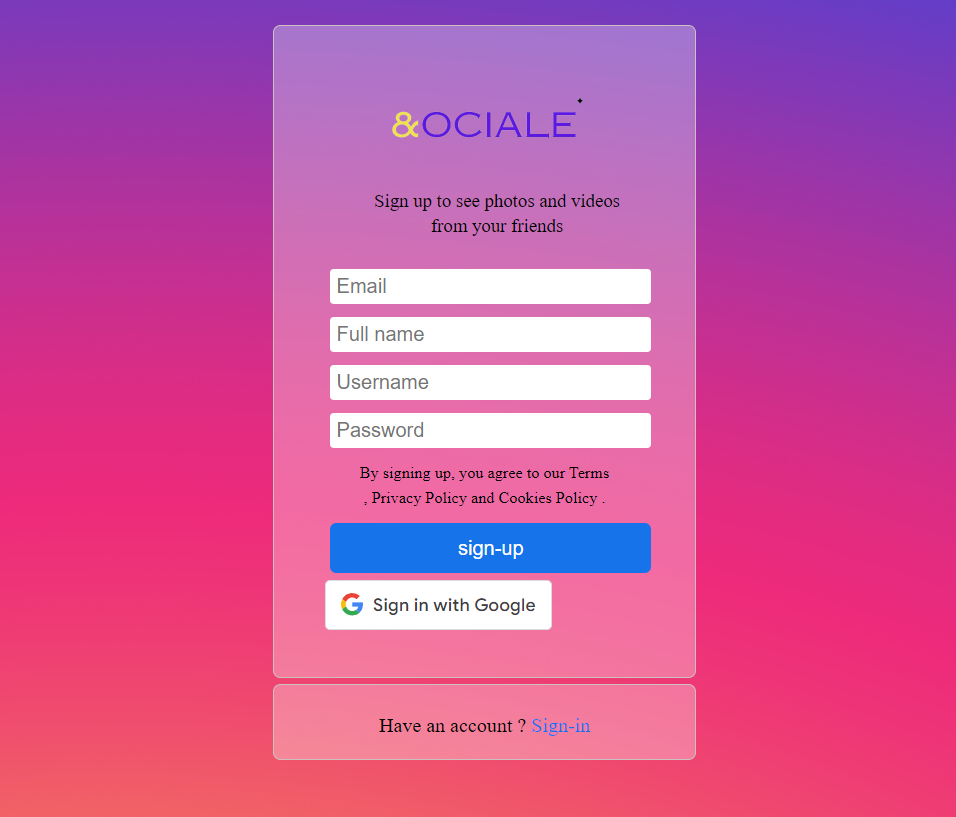
Signin Page



The above Image is the signin page where the user can be able to signin to their account with their Authenticated Email id and password

When user successfully sigin it will get redirected to home page where all post,likes and comments will be display to Authenticated user

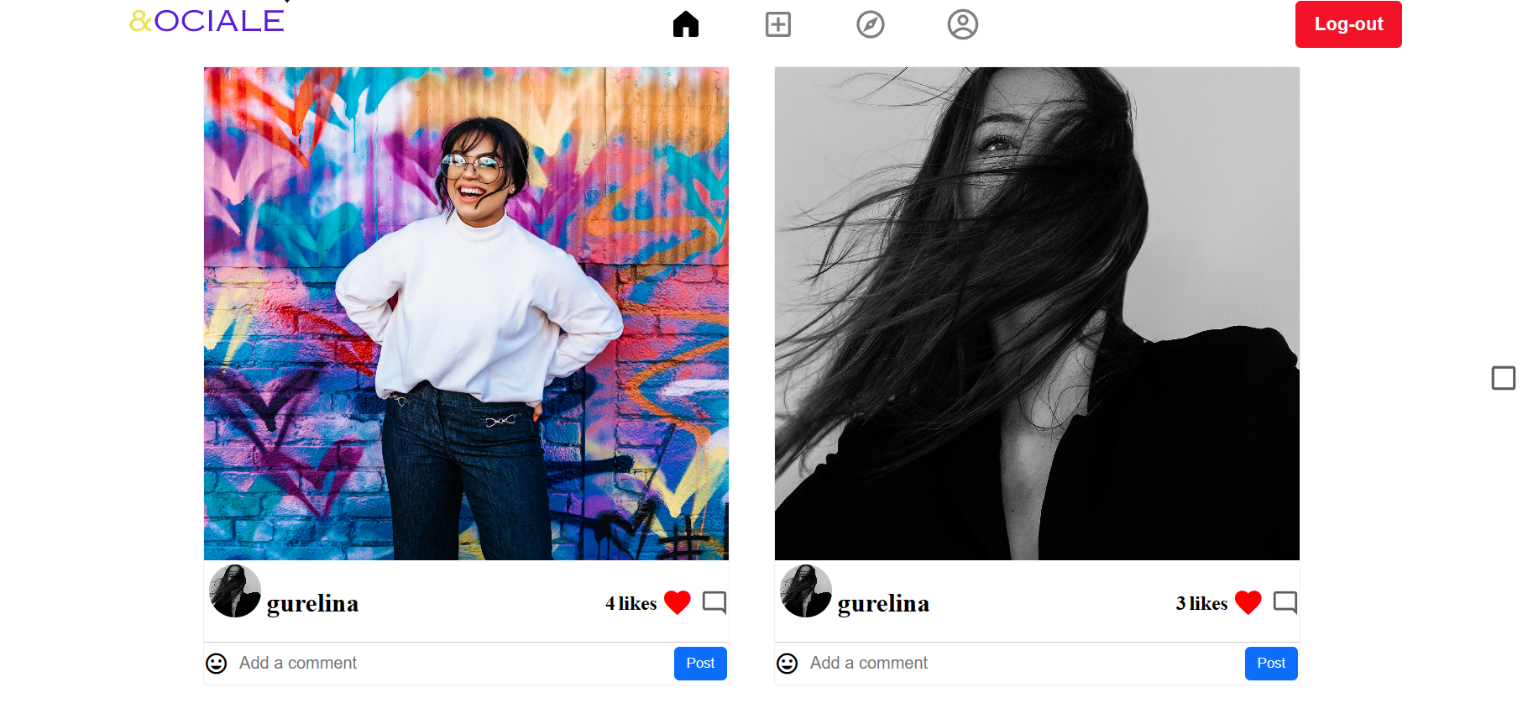
Signup Page

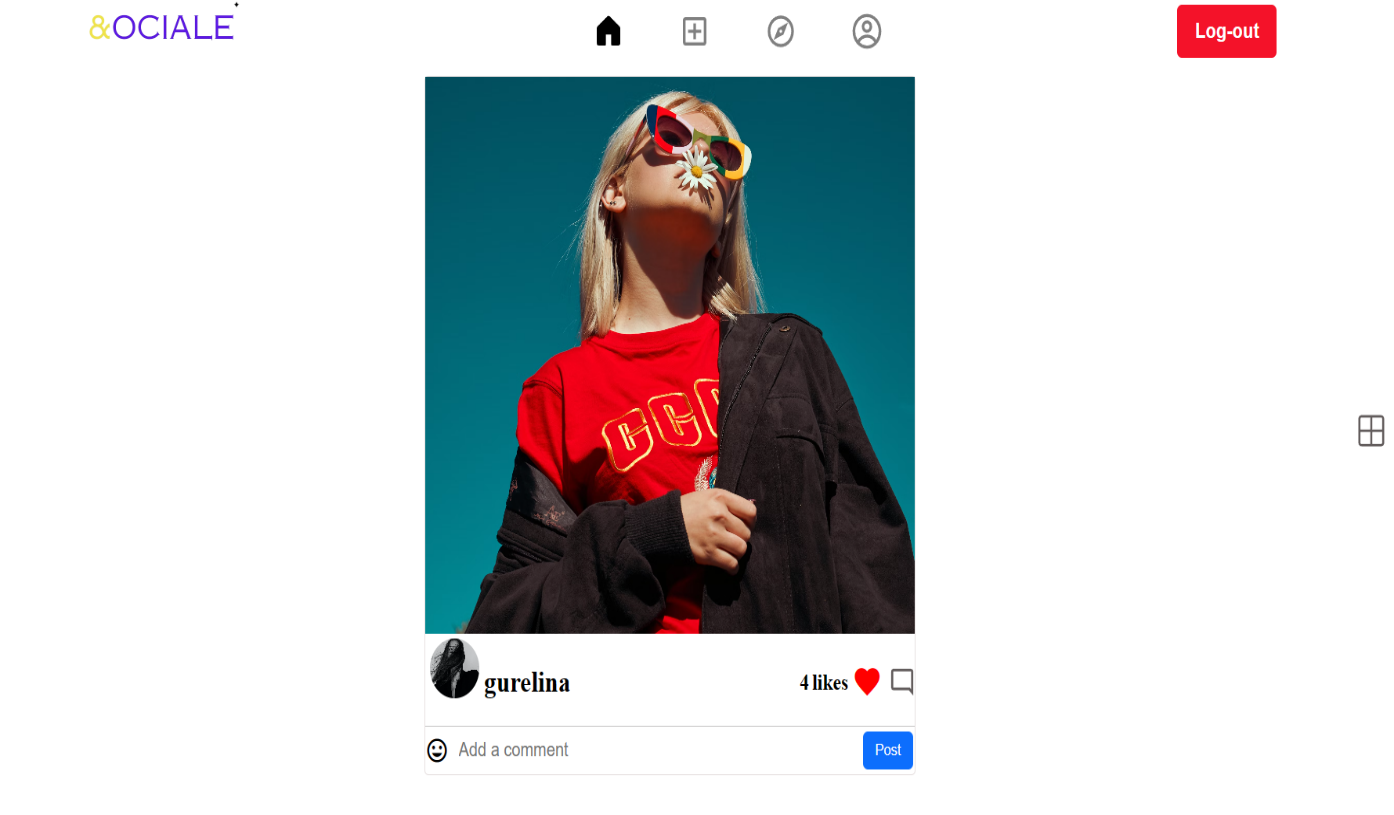


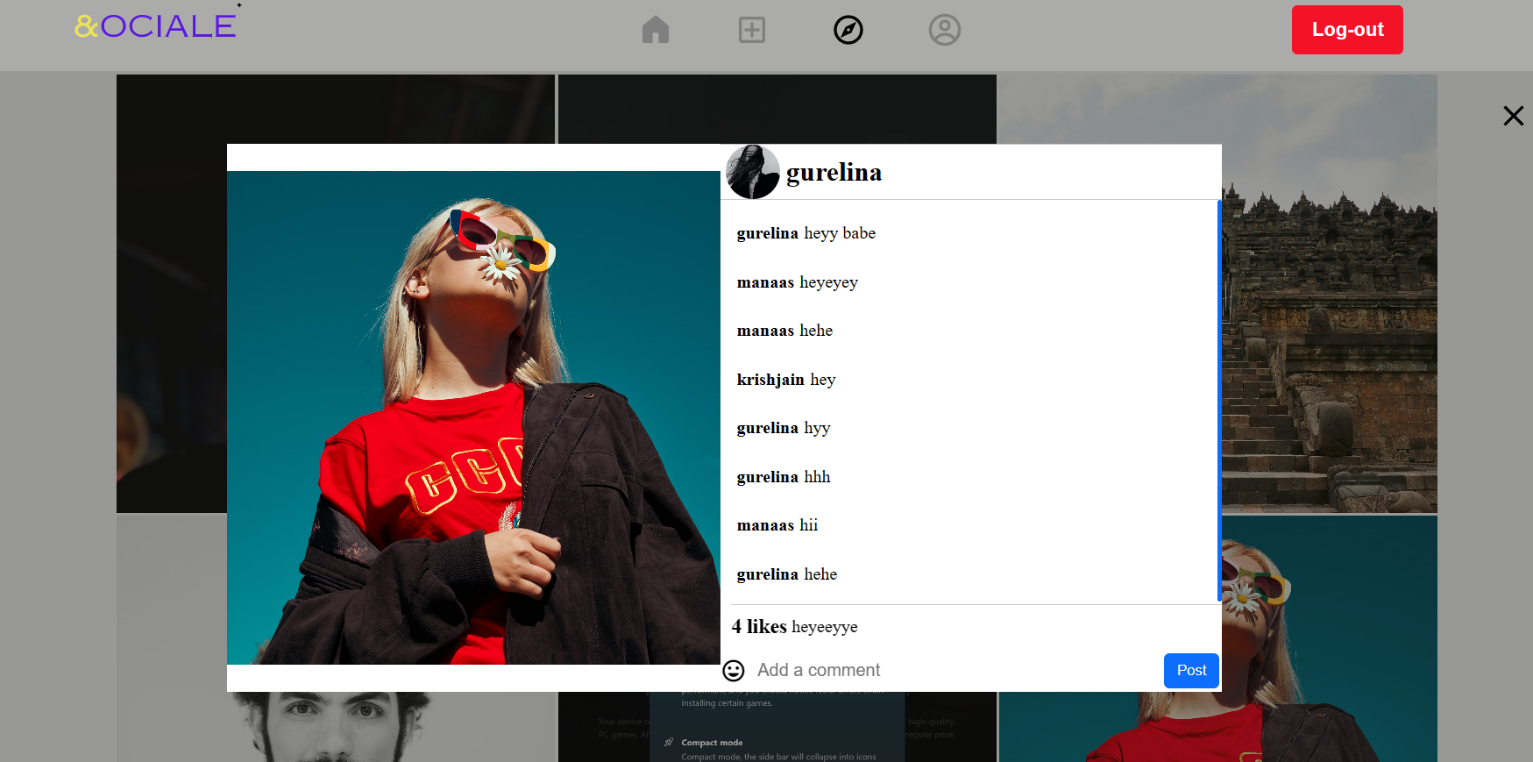
The above image is the signup page where the user can create their account if user successfully signup to this page it will redirected to signin page for authentication.

While user can also create their account using their google account where the details of user will taken down from google account like name , profile photo etc and it will get redirected to home page After successfully sigup using google account.

Home Page

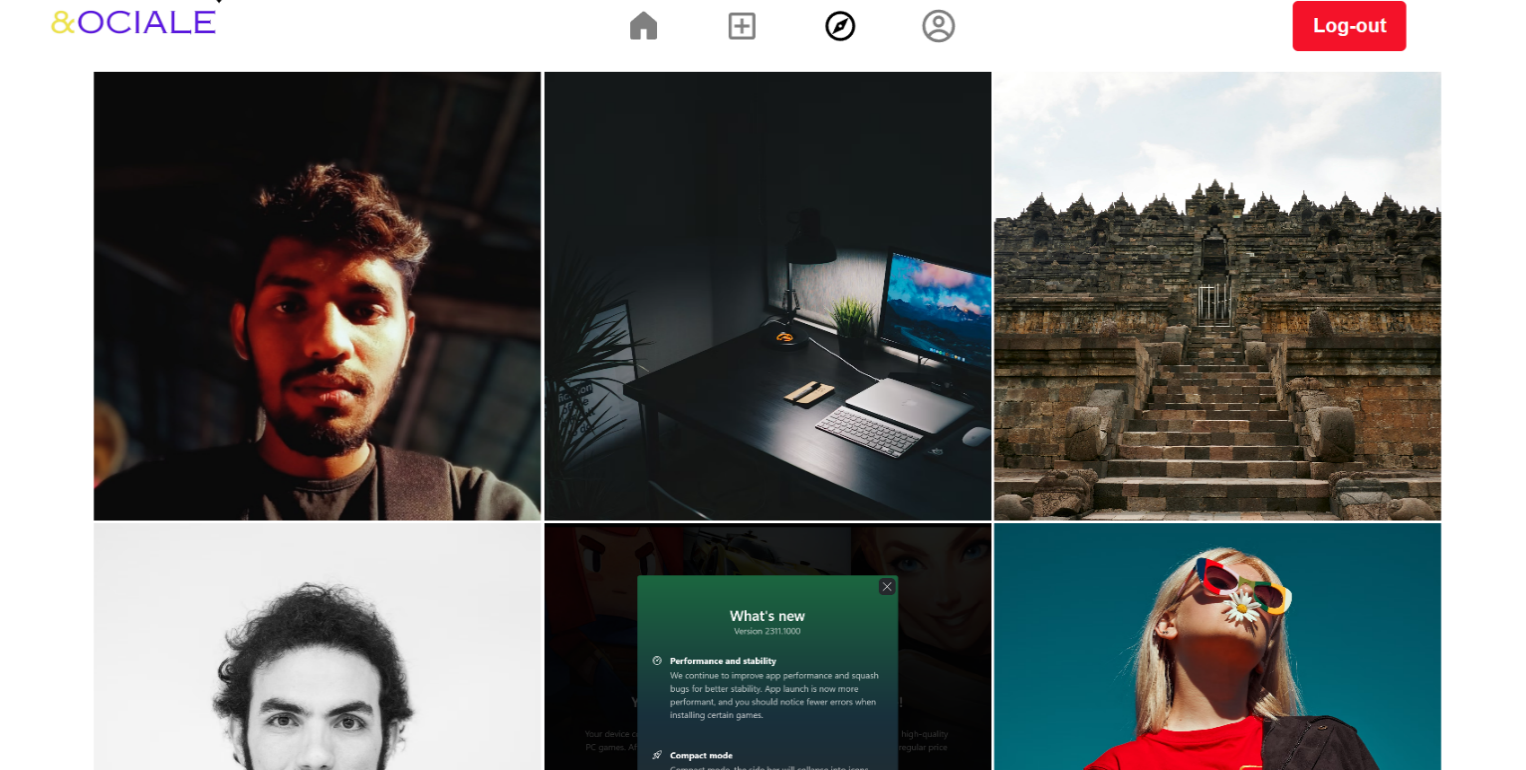


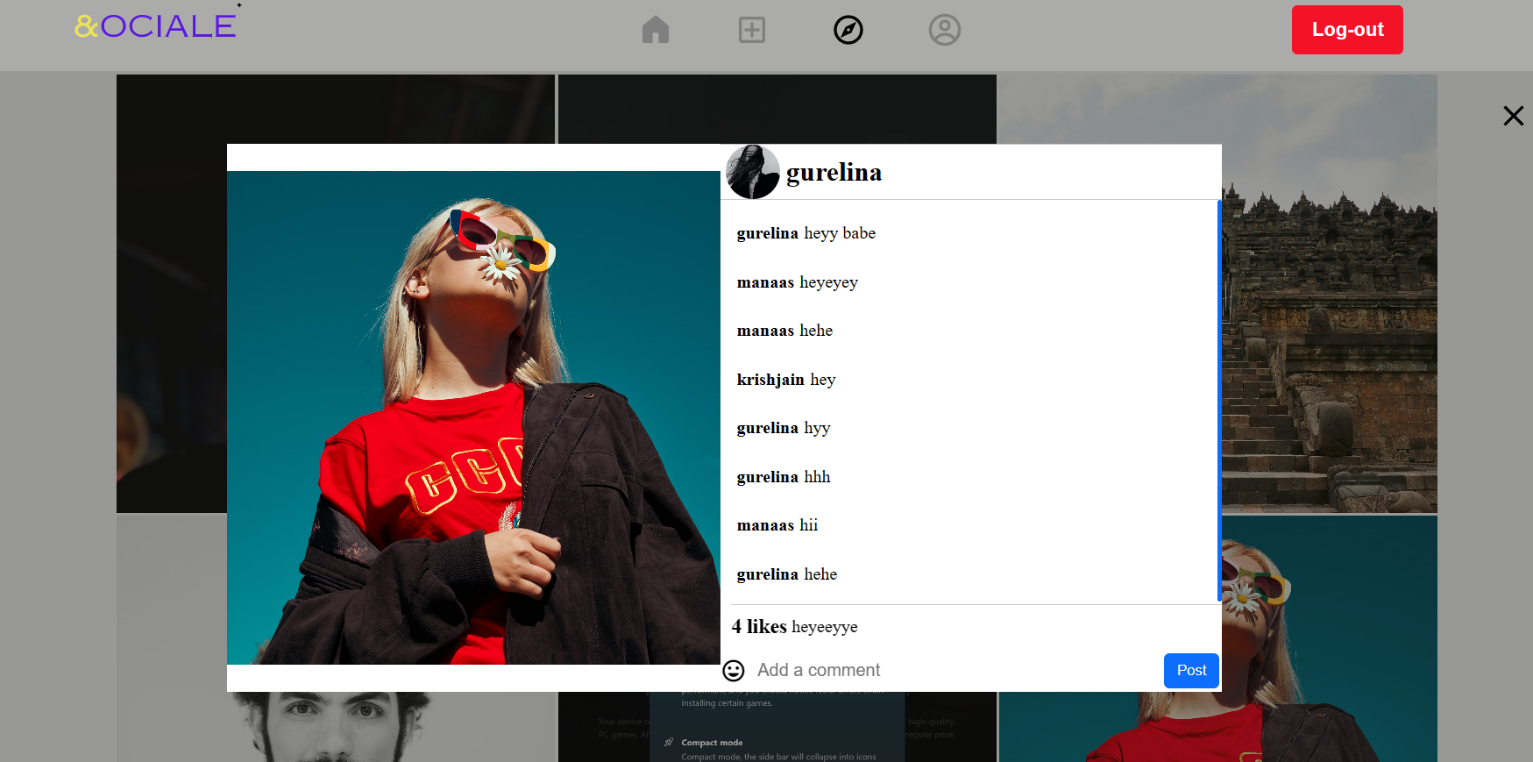




In this home.jsx file where the user will be able to see the latest Post share by the different User. user are able to like/unlike ,comments on another users post. User can customize the home page like it can view post in Normal view and Grid View And it is responsive in every device

Explore Page





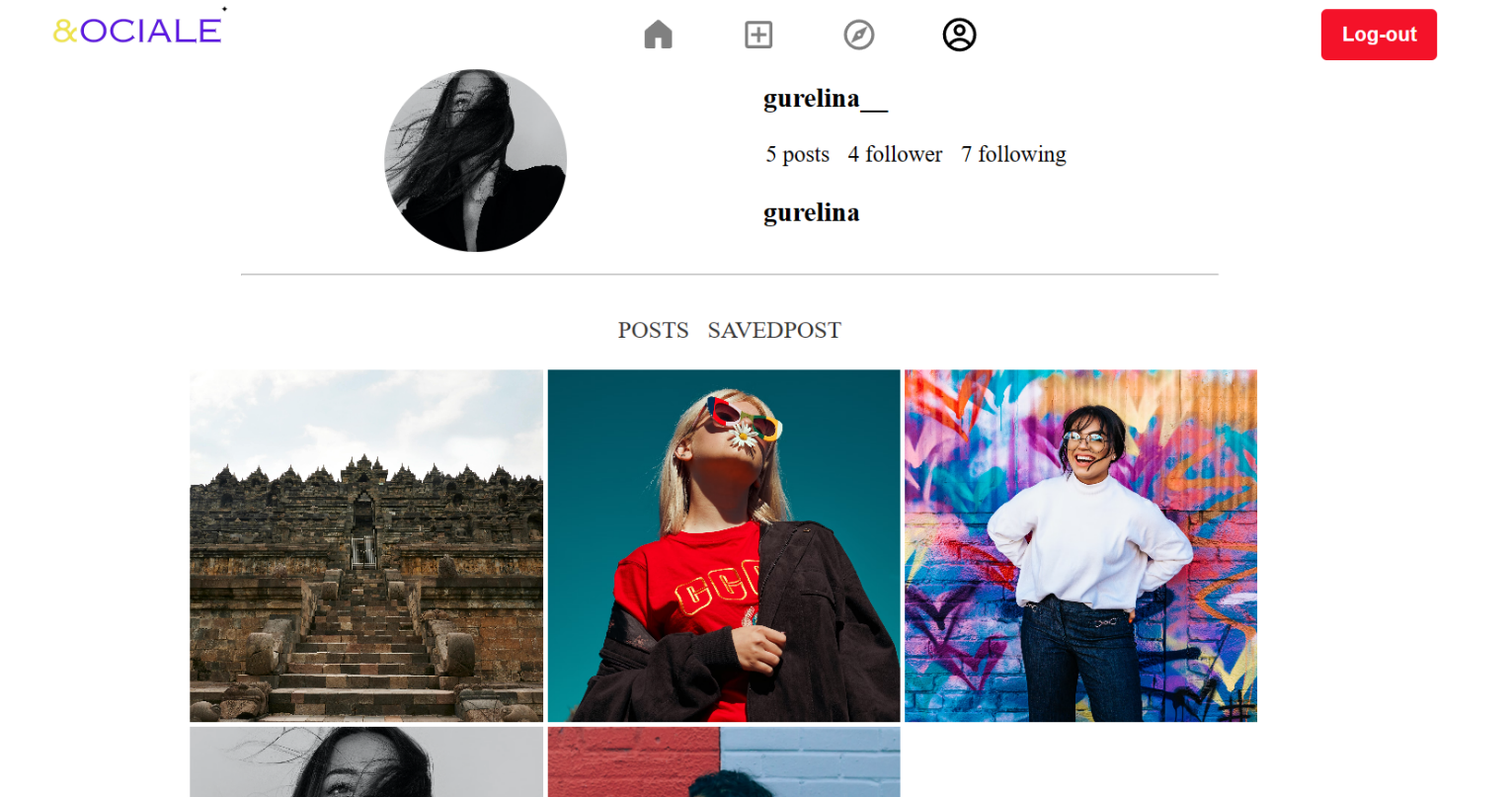
In this Explore page it will display POST whom he/she has follow to another user and user can like/unlike comment to their POST

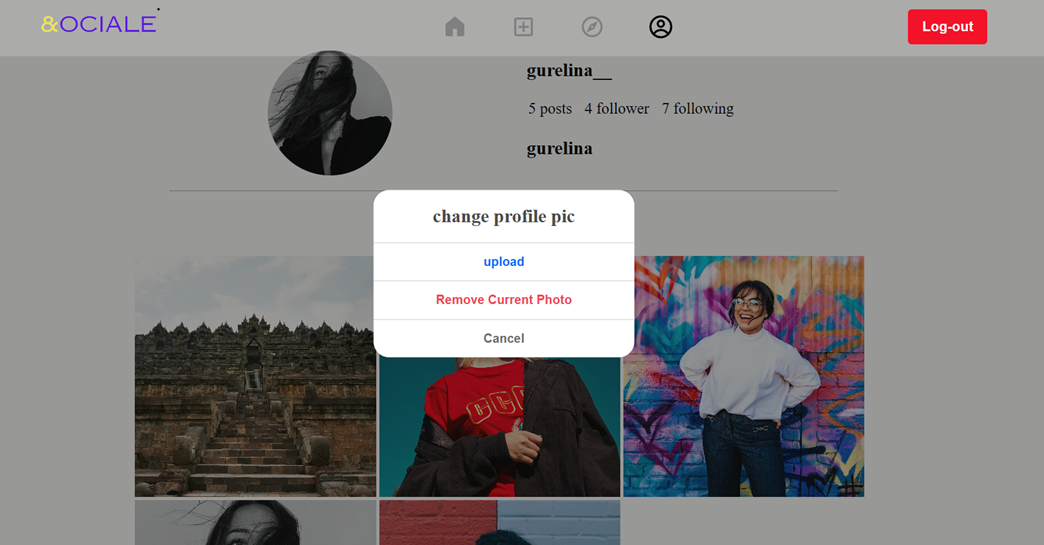
Create Post



In create Post user can upload their Post and caption to Sociale where it will be display to all user

Profile Page





In Profile page of sociale user will be able to view their Profile page in which user name, follower , following , username, profile photo and all the post uploaded by the user.

User can upload , remove their profile photo and also delete their profile Photo according to their choice

# Chapter 6

**Conclusion**

Integration testing stands as a cornerstone in the development process of MERN (MongoDB, Express.js, React, Node.js) social media applications. Through a systematic approach encompassing API endpoint validation, database operation scrutiny, and frontend-backend interaction assessment, developers guarantee the seamless functionality and robustness of their applications.

In testing API endpoints, tools like Supertest facilitate the examination of various request scenarios, encompassing valid inputs, error handling, and authentication procedures. Concurrently, database operation testing ensures data integrity and consistency, vital for the reliable storage and retrieval of user-generated content.

Authentication and authorization mechanisms are subject to rigorous evaluation to safeguard user data and application security. Through integration testing, developers confirm that only authenticated users gain access to designated resources, bolstering the application's resilience against unauthorized access attempts.

Frontend-backend interaction, a pivotal aspect of user experience, is meticulously scrutinized. By simulating user interactions with frontend components and assessing data fetching, form submissions, and real-time functionality, developers ensure a seamless user experience across the application.

Error handling and edge case scenarios are meticulously examined to fortify the application against unforeseen circumstances. Robust error handling mechanisms, tested through integration testing, ensure that the application gracefully navigates through network failures, unexpected API responses, and database errors, upholding user trust and satisfaction.

In summation, integration testing serves as the bedrock for the stability, security, and functionality of MERN social media applications. By adhering to best practices and leveraging appropriate tools, developers can confidently deliver applications that meet user expectations and withstand the rigors of real-world usage.

**Limitations**

-Scalability Challenges: MongoDB, while suitable for many use cases, may face scalability challenges with massive amounts of data or high traffic volumes. Horizontal scaling (sharding) can mitigate this, but it adds complexity to the application architecture.

-Real-Time Communication Complexity: Implementing real-time features like live chat or notifications can be complex in a MERN stack. While tools like Socket.IO facilitate real-time communication, managing socket connections and ensuring scalability can be challenging.

-SEO Limitations: Single-page applications (SPAs) built with React may face challenges with search engine optimization (SEO) due to their client-side rendering approach. Ensuring proper server-side rendering (SSR) or implementing techniques like prerendering can mitigate this limitation.

-Complex State Management: As applications grow in complexity, managing state in React can become challenging. While Redux or Context API can help manage global state, improper state management can lead to performance issues and difficult debugging.

-Security Concerns: As with any web application, security is paramount. MERN applications may face common security vulnerabilities such as cross-site scripting (XSS), cross-site request forgery (CSRF), and injection attacks. Robust security practices, including input validation, authentication, and authorization, are essential to mitigate these risks.

-Learning Curve: Learning and mastering the entire MERN stack requires time and effort, especially for developers new to JavaScript and full-stack development. Additionally, keeping up with the rapid pace of change in the JavaScript ecosystem can be challenging.

-Dependency Management: Managing dependencies and ensuring compatibility between different packages, libraries, and versions can be challenging in a MERN stack. Dependency conflicts or outdated packages may lead to compatibility issues and security vulnerabilities.

-Testing Complexity: While integration testing is crucial, setting up and maintaining a comprehensive test suite for a MERN application, including both frontend and backend components, can be complex and time-consuming.

**Future Scope**

**-AI-powered Features:** Integration of Artificial Intelligence (AI) can personalize content feeds, recommend connections, and even moderate content for a safer user experience.

**-Immersive Experiences:** Technologies like Virtual Reality (VR) and Augmented Reality (AR) can create new avenues for social interaction and content creation.

**-Decentralized Social Media:** Blockchain technology could enable the development of MERN apps that prioritize user data ownership and privacy.

**-Focus on Micro-communities:** MERN apps can cater to niche interests by fostering smaller, focused communities around specific passions.

**-Live Features:** Real-time features like live streaming and interactive events can further enhance user engagement.

**-Integration with IoT:** Connecting with smart devices can allow users to share experiences and data in innovative ways through their social network.

# Chapter 7

**References**

* Head First C# by Andrew Stellman and Jennifer Greene
* HTML5 and CSS3 All-in-One For Dummies by Andy Harris
* "React Redux Book: Learn React and Redux in One Day" by André Castro:
* React Up & Running: Building Web Applications" by Stoyan Stefanov

**Websites Used**

[Image and Video Upload, Storage, Optimization and CDN (cloudinary.com)](https://cloudinary.com/)

[bcrypt - npm (npmjs.com)](https://www.npmjs.com/package/bcrypt)

[React Icons (react-icons.github.io)](https://react-icons.github.io/react-icons/)

[javascript - Regular Expression for password validation - Stack Overflow](https://stackoverflow.com/questions/2370015/regular-expression-for-password-validation)

[Browse Fonts - Google Fonts](https://fonts.google.com/)

[Beautiful Free Images & Pictures | Unsplash](https://unsplash.com/)

[MongoDB Compass | MongoDB](https://www.mongodb.com/products/tools/compass)

[Pinterest](https://in.pinterest.com/)