FullStack Development with MERN

Project Documentation

1. Introduction

- **ProjectTitle:**

CONNECTIFY: YOUR PERSONALISED SOCIAL NETWORK

- **TeamMembers:**
  SHAIK SHABANA(ROLE: FRONTEND,IMPLEMENTATION & EXECUTION)
  SHAIK FUZAIL JANI (ROLE: FRONTEND, BACKEND IMPLEMENTATION)
  POGIRI JYOTHI (ROLE: BACKEND)
  SHAIK PARVIN (ROLE: BACKEND)
  TUNGALA DHARANI (ROLE: FRONTEND)

2. ProjectOverview

- **Purpose:** The main objective of the SocialeX project is to develop a highly interactive and secure social networking platform that enhances user engagement and connectivity.
- **Features:**

  - **Real-time Updates:** Stay up to date with the latest activities and posts from your connections. Receive instant notifications for likes, comments, and mentions, ensuring you never miss out on important interactions.

  - **Explore & Discover:** Explore a vast world of content and discover new ideas, trends, and communities. Engage with trending posts, discover new accounts, and connect with like-minded individuals.

  - **Messaging and Chat:** Engage in private conversations and group chats with friends and followers. Share messages, emojis, photos, and videos, fostering real-time communication and connection.

  - **Interactive Features:** Interact with posts through likes, comments, and shares. Express your thoughts, provide feedback, and engage in lively discussions with your network.

  - **Follow and Connect:** Follow your favourite accounts and connect with influencers, brands, and individuals who inspire you. Build a vibrant network of connections and discover new opportunities.

  - **Data privacy and Security:** We prioritize the protection of your personal information and data. Our app employs robust security measures, ensuring that your interactions, posts, and personal details remain secure and confidential.

3. Architecture

Frontend:
- **Technology Stack:** React.js for building a dynamic and responsive user interface, Redux for state management, and SCSS for styling.
- **User Interface Design:** Wireframes and prototypes were created to ensure a user-friendly experience. Components were built to be reusable, responsive, and accessible.
- **Routing:** Implemented using React Router to handle navigation between pages, ensuring smooth transitions and a single-page application feel.
- **API Integration:** Axios was used for making HTTP requests to the backend, handling tasks such as user authentication, data retrieval, and content submission.

Backend:
- **Technology Stack:** Node.js with Express.js for server-side development.
- **Database:** MongoDB for storing user data, posts, comments, and other dynamic content, with Mongoose as the ORM (Object Relational Mapping) tool.
- **Authentication:** JWT (JSON Web Tokens) for secure and stateless user sessions.
- **API Endpoints:** RESTful API endpoints were developed to manage CRUD (Create, Read, Update, Delete) operations for users, posts, comments, and likes.
- **Real-Time Features:** Socket.io was integrated to enable real-time notifications and updates, enhancing user engagement.

Database:
- **Hosting:** The frontend was deployed on Vercel and the backend on Heroku, both chosen for their scalability and ease of use.
- **CI/CD:** GitHub Actions were used to automate testing and deployment processes, ensuring that new updates were seamlessly integrated and deployed.
- **Environment Management:** .env files were used to manage environment-specific variables securely.

4.Setup Instructions:

Prerequisities:

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, React.js:

☐ Node.js and npm:

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side.It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine,as they are required to run JavaScript on the server-side.

Download:https://nodejs.org/en/download/

Installationinstructions:https://nodejs.org/en/download/package-manager/

npm init

☐ Express.js:

Express.jsisafastandminimalistwebapplicationframeworkforNode.js.   It simplifiestheprocessofcreatingrobustAPIsandwebapplications,offering   features   like routing, middleware support, and modular architecture.

InstallExpress.js,awebapplicationframeworkforNode.js,whichhandlesserver-   side   routing, middleware, and API development.

Installation:Open your command prompt or terminal and run the following command:

npm install express

☐ MongoDB:

MongoDB is a flexible and scalableNoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data. Download:https://www.mongodb.com/try/download/community

Installationinstructions:https://docs.mongodb.com/manual/installation/

⬜ React.js:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components,making it easier to build dynamic and responsive web applications.
Install React.js,aJava Script library for building user interfaces.

Follow the installation guide:https://reactjs.org/docs/create-a-new-react-app.html

⬜ HTML,CSS,andJavaScript:

o Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

⬜ Socket.io:

Socket.io is a real-time bidirectional communication library that enables seamless communication between the server and clients. It allows for real-time data exchange, event-based messaging, and facilitates the development of real-time applications such as chat, collaboration, and gaming platforms. Install Socket.io, a real-time bidirectional communication library for web applications.

Installation:

• Open your command prompt or terminal of server and run the following command: npm install socket.io

• Open your command prompt or terminal of client and run the following command: npm install socket.io-client

⬜ DatabaseConnectivity:

UseaMongoDBdriveroranObject-DocumentMapping (ODM) library likeMongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations. To Connect the Database with Node JS go through the below provided link: https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/

⬜ Front-end Framework:

Utilize Reactjs to build the user-facing part of the application,includingenteringbookingroom,statusofthebooking,anduser interfaces for the admin dashboard.FormakingbetterUIwehavealsousedsomelibrarieslikematerialUIand boostrap.

⬜ Version Control:

Use Git for version control, enabling collaboration and tracking changesthroughoutthedevelopmentprocess.PlatformslikeGitHuborBitbucket can host your repository.

Git:Downloadandinstallationinstructionscanbefoundat: https://git- scm.com/download

☐ **Development Environment**:

ChooseacodeeditororIntegratedDevelopment Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- VisualStudioCode:Download from https://code.visualstudio.com/download

☐ **Installation:**

gitclone: https://github.com/Muskan-0000s/SocialeX-App

Install Dependencies:

- Navigate into the cloned repository directory:
  cd socialex
- Install the required dependencies by running the following commands:
cd client
npm install
cd../server
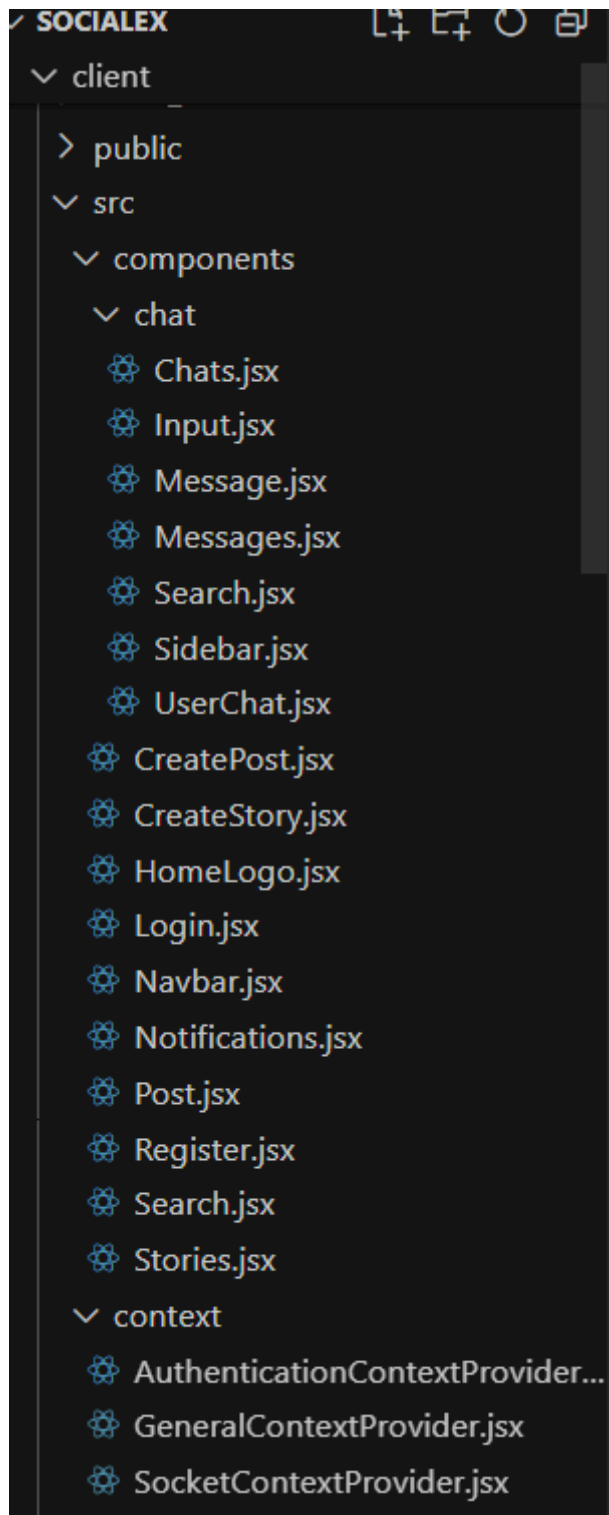npm install

Start the Development Server:

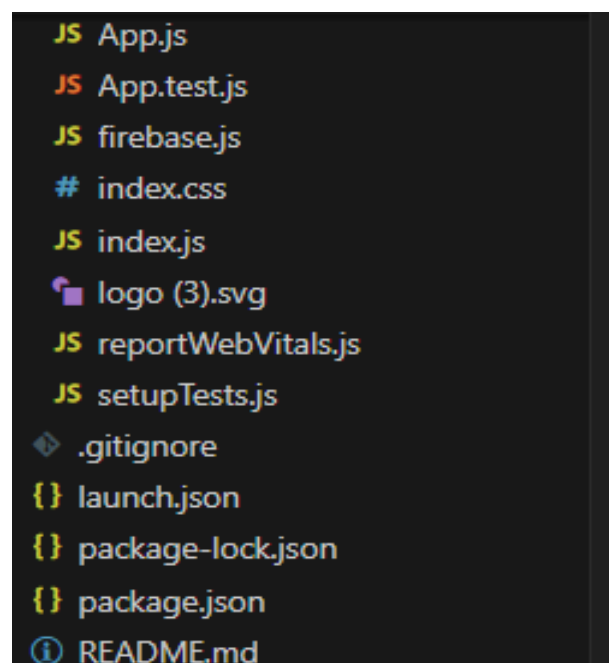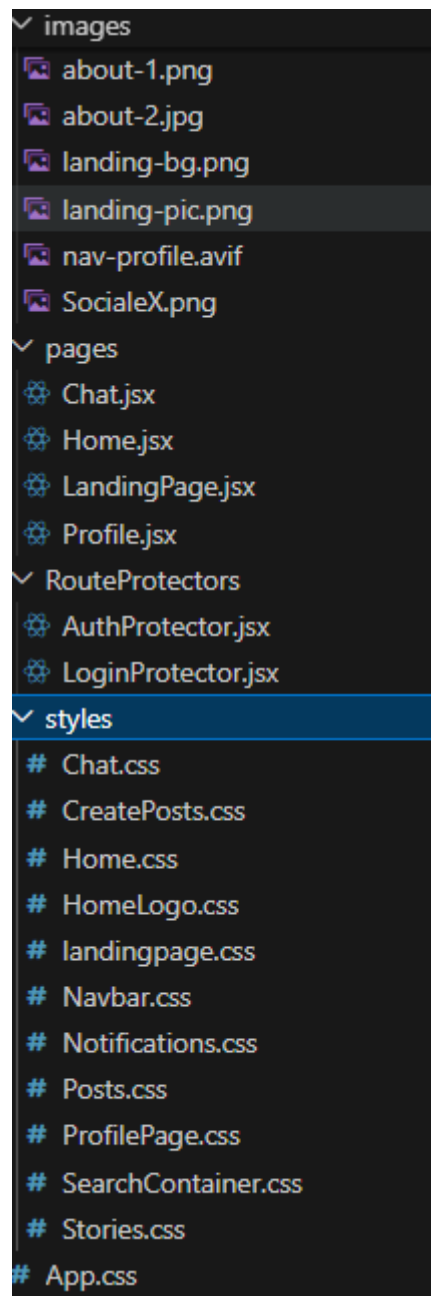Open your web browser and navigate to http://localhost:3000
    • You should see the video conference app's homepage, indicating that the installation and setup were successful.
You have successfully installed and set up the e-commerce app on your local machine. You can now proceed with further customization, development, and testing as needed.

5.Project Structure:

- **Client:**

```
> images
  about-1.png
  about-2.jpg
  landing-bg.png
  landing-pic.png
  nav-profile.avif
  SocialeX.png
> pages
  Chat.jsx
  Home.jsx
  LandingPage.jsx
  Profile.jsx
> RouteProtectors
  AuthProtector.jsx
  LoginProtector.jsx
> styles
  # Chat.css
  # CreatePosts.css
  # Home.css
  # HomeLogo.css
  # landingpage.css
  # Navbar.css
  # Notifications.css
  # Posts.css
  # ProfilePage.css
  # SearchContainer.css
  # Stories.css
  # App.css
```

```
JS App.js
JS App.test.js
JS firebase.js
 # index.css
JS index.js
 logo (3).svg
JS reportWebVitals.js
JS setupTests.js
 .gitignore
{} launch.json
{} package-lock.json
{} package.json
 README.md
```

The above directory structure represents the directories and files in the client folder (front end) where, react Js is used along with Api's such as socket.io.

☐ **Server:**
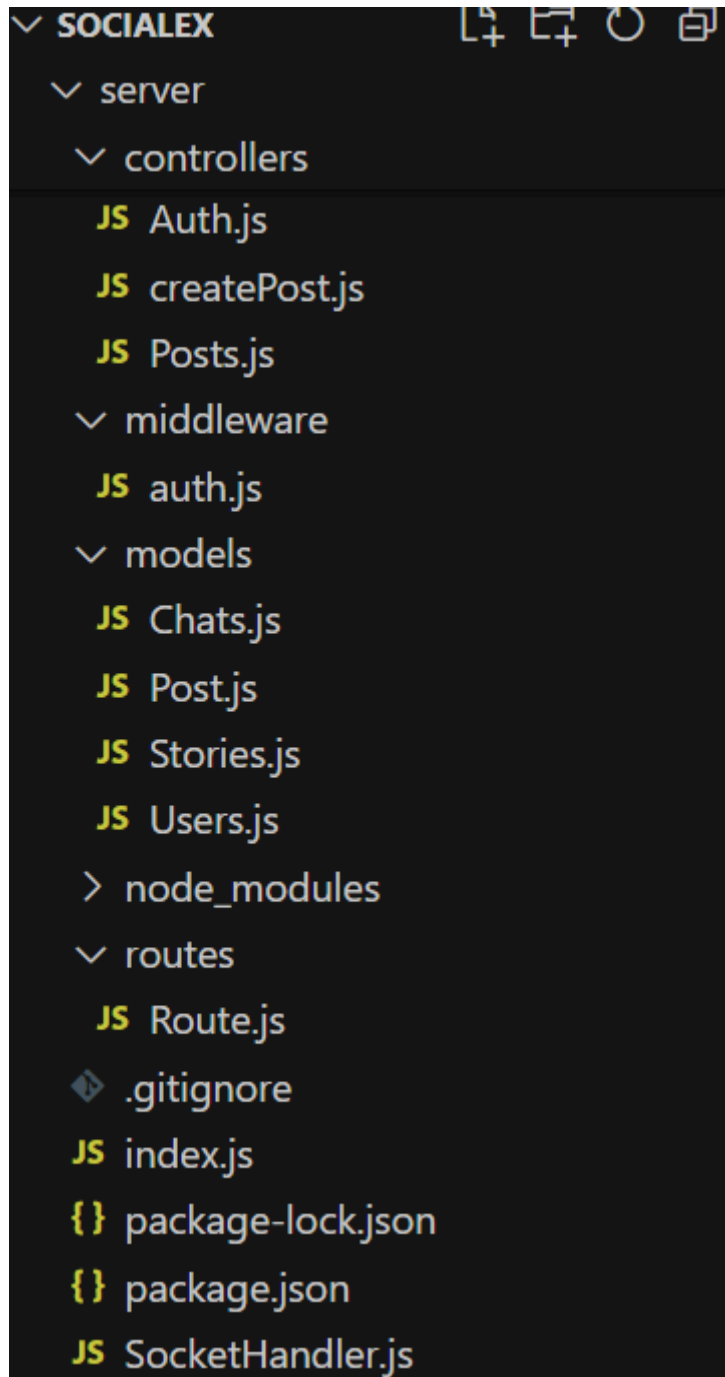
The below above structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with socket.io.

Running the Application:

- Provide commands to start the frontend and backend servers locally.

    1. **Frontend:** `npm start` in the client directory.
    2. **Backend:** `npm start` in the server directory.

6.API Documentation:

Here's an API documentation outline for our social media app backend,detailing common endpoints and their functionality.Each endpoint includes HTTP method,route,requestparameters, example responses, and any relevant error codes.

1.Authentication Endpoints:

☐ *AuthenticationContextProvider.jsx:*

import React, { createContext, useState } from 'react';

import axios from "axios";

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

export constAuthenticationContext = createContext();

constAuthenticationContextProvider = ({children}) => {

const [username, setUsername] = useState('');

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

constprofilePic = 'https://images.unsplash.com/photo-1593085512500-5d55148d6f0d?ixlib=rb4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w=580&q=80';

const inputs = {username: username, email: email, password: password, profilePic: profilePic};

const navigate = useNavigate();

const login = async () =>{

  try{

constloginInputs = {email: email, password: password}

```
await axios.post('http://localhost:6001/login', loginInputs)
    .then( async (res)=>{
        console.log("holaads",res);
localStorage.setItem('userToken', res.data.token);

localStorage.setItem('userId', res.data.user._id);

localStorage.setItem('username', res.data.user.username);

localStorage.setItem('email', res.data.user.email);

localStorage.setItem('profilePic', res.data.user.profilePic);

localStorage.setItem('posts', res.data.user.posts);

localStorage.setItem('followers', res.data.user.followers);

localStorage.setItem('following', res.data.user.following);

        navigate('/');
    }).catch((err) =>{
        console.log(err);
    });
  }catch(err){
    console.log(err);
  }
 }
const register = async () =>{
  try{
    await axios.post('http://localhost:6001/register', inputs)


.then( async (res)=>{
localStorage.setItem('userToken', res.data.token);

localStorage.setItem('userId', res.data.user._id);

localStorage.setItem('username', res.data.user.username);

localStorage.setItem('email', res.data.user.email);
```

```
            localStorage.setItem('profilePic', res.data.user.profilePic);

            localStorage.setItem('posts', res.data.user.posts);

            localStorage.setItem('followers', res.data.user.followers);

            localStorage.setItem('following', res.data.user.following);

                navigate('/');

            }).catch((err) =>{

                console.log(err);

            });

        }catch(err){

            console.log(err);

        }

    }

    const logout = async () =>{

        for (let key in localStorage) {

            if (localStorage.hasOwnProperty(key)) {

localStorage.removeItem(key);

            }

        }


    navigate('/landing');

    }

    return (

    <AuthenticationContext.Provider value={{login, register, logout, username, setUsername,
    email, setEmail, password, setPassword }} >{children}</AuthenticationContext.Provider>

    )

    }
export default AuthenticationContextProvider
```

2.UserProfileEndpoints:

- *Profile.jsx:*

```
import React, { useContext, useEffect, useState } from 'react'
import '../styles/ProfilePage.css'
import '../styles/Posts.css';
import { AiOutlineHeart, AiTwotoneHeart } from "react-icons/ai";
import { BiCommentDetail } from "react-icons/bi";
import { FaGlobeAmericas } from "react-icons/fa";
//import {IoIosPersonAdd} from 'react-icons/io'
import HomeLogo from '../components/HomeLogo'
import Navbar from '../components/Navbar'
import { AuthenticationContext } from '../context/AuthenticationContextProvider'
import { GeneralContext } from '../context/GeneralContextProvider'
import {useParams} from 'react-router-dom';
import axios from 'axios';

const Profile = () => {

  const {logout} = useContext(AuthenticationContext);

  const {socket} = useContext(GeneralContext);

  const {id} = useParams();
  const userId = localStorage.getItem("userId");

  const [userProfile, setUserProfile] = useState([]);

const [updateProfilePic, setUpdateProfilePic] = useState('');
const [updateProfileUsername, setUpdateProfileUsername] = useState('');
const [updateProfileAbout, setUpdateProfileAbout] = useState('');

const [isUpdating, setIsUpdating] = useState(false);


  useEffect(()=>{

   socket.emit("fetch-profile", {_id: id});

   socket.on("profile-fetched", async({profile})=>{
    setUserProfile(profile);
    setUpdateProfilePic(profile.profilePic);
    setUpdateProfileUsername(profile.username);
    setUpdateProfileAbout(profile.about);
   })


  },)


  const handleUpdate = async () =>{
      socket.emit('updateProfile', {userId: userProfile._id, profilePic: updateProfilePic,
username: updateProfileUsername, about: updateProfileAbout});
    setIsUpdating(false);
  }
```

```javascript
  const [posts, setPosts] = useState([]);
  useEffect(() => {
    fetchPosts();
  }, []);

  const fetchPosts = async () => {
    try {
      const response = await axios.get('http://localhost:6001/fetchAllPosts');
      const fetchedPosts = response.data;
      setPosts(fetchedPosts);
    } catch (error) {
      console.error(error);
    }
  };

  const handleLike = (userId, postId) =>{
    socket.emit('postLiked', {userId, postId});

}

const handleUnLike = (userId, postId) =>{
    socket.emit('postUnLiked', {userId, postId});

}

const handleFollow = async (userId) =>{
    socket.emit('followUser', {ownId: localStorage.getItem('userId'), followingUserId:
userId});
}
const handleUnFollow = async (userId) =>{
    socket.emit('unFollowUser', {ownId: localStorage.getItem('userId'), followingUserId:
userId});
}

useEffect(()=>{
  socket.on('userFollowed', ({following})=>{
    localStorage.setItem('following', following);
})

socket.on('userUnFollowed', ({following})=>{
  localStorage.setItem('following', following);
})
})


//const [followDisplayType, setFollowDisplayType] = useState('followers');

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

  const handleComment = (postId, username)=>{
    socket.emit('makeComment', {postId, username, comment});
    setComment('');
  }

const handleDeletePost = async (postId) =>{
  await socket.emit('delete-post', {postId});
}
```

```
useEffect(()=>{

  socket.on('post-deleted', async ({posts})=>{

   setPosts(posts)
  })

},[socket])


 return (
   <div className='profilePage'>
    <HomeLogo />
    <Navbar />

                <div className="profileCard" style={isUpdating ? {display:'none'}:
{display:"flex"}}>

        <img src={userProfile.profilePic} alt="" />

        <h4>{userProfile.username}</h4>
        <p>{userProfile.about} </p>


        <div className="profileDetailCounts">

         <div className="followersCount">
          <p>Followers</p>
          <p>{userProfile.followers ? userProfile.followers.length : 0}</p>
         </div>
         <div className="followingCounts">
          <p>Following</p>
          <p>{userProfile.following ? userProfile.following.length : 0}</p>
         </div>
        </div>

        <div className="profileControls">
        {
         userProfile._id === userId ?

         <div className="profileControlBtns">

          <button onClick={async () => {await logout()}}>Logout</button>

                             <button type="button" className="btn btn-primary"
onClick={()=>setIsUpdating(true)}>Edit</button>


         </div>

         :
         <div className="profileControlBtns">

         {
          localStorage.getItem('following').includes(userProfile._id) ?
          <>
```

```jsx
                                <button   onClick={()=>handleUnFollow(userProfile._id)}
style={{backgroundColor: 'rgb(224, 42, 42)'}}>Unfollow</button>
        <button >Message</button>
        </>
        :

        <button onClick={()=>handleFollow(userProfile._id)}>Follow</button>

      }

    </div>
    }
    </div>

  </div>


            <div   className='profileEditCard'style={!isUpdating  ?  {display:'none'}:
{display:"flex"}}>
      <div class="mb-3">
       <label for="exampleInputEmail1" class="form-label">Profile Image</label>
                <input   type="text"   class="form-control"   id="exampleInputEmail1"
onChange={(e)=> setUpdateProfilePic(e.target.value)} value={updateProfilePic} />
      </div>
      <div class="mb-3">
       <label for="exampleInputPassword1" class="form-label">Username</label>
                <input   type="text"   class="form-control"   id="exampleInputPassword1"
onChange={(e)=>                      setUpdateProfileUsername(e.target.value)}
value={updateProfileUsername}/>
      </div>
      <div class="mb-3">
       <label for="editAbout" class="form-label">About</label>
            <input   type="text"   class="form-control"   id="editAbout"   onChange={(e)=>
setUpdateProfileAbout(e.target.value)} value={updateProfileAbout}/>
      </div>
      <button className='btn btn-primary' onClick={handleUpdate}>Update</button>
    </div>




    <div className="profilePostsContainer">

    {posts.filter(post => post.userId === userProfile._id).map((post) => {


    return(

    <div className="Post" key={post._id}>

    <div className="postTop">
      <div className="postTopDetails">
        <img src={post.userPic} alt="" className="userpic" />
        <h3 className="usernameTop">{post.userName}</h3>
```

```jsx
                    </div>
                                    <button className='btn btn-danger deletePost' onClick={()=>
handleDeletePost(post._id)}>Delete</button>
        </div>

        { post.fileType === 'photo'?

            <img src={post.file} className='postimg' alt="" />

            :

            <video id="videoPlayer" className='postimg' controls autoPlay muted>
              <source src={post.file} />
            </video>

            }

        <div className="postReact">
          <div className="supliconcol">

            {
              post.likes.includes(localStorage.getItem('userId')) ?

                        <AiTwotoneHeart className='support reactbtn'  onClick={() =>
handleUnLike(localStorage.getItem('userId'), post._id)}/>

                :

                        <AiOutlineHeart className='support reactbtn'  onClick={() =>
handleLike(localStorage.getItem('userId'), post._id)}/>
            }


                                                <label  htmlFor="support"
className='supportCount'>{post.likes.length}</label>
          </div>
          <BiCommentDetail className='comment reactbtn' />
           {/* <FiSend className='share reactbtn' onClick={()=> {handleShare(post)}} />
*/}
          <div className="placeiconcol">
            <FaGlobeAmericas className='placeicon reactbtn' name='place' />
            <label htmlFor="place" className='place'>{post.location}</label>
          </div>
        </div>


        <div className="detail">
          <div className='descdataWithBtn'>
            <label htmlFor='username' className="desc labeldata" id='desc'>
              <span style={{{fontWeight: 'bold'}}}>
                {post.userName}
              </span>
                   {post.description}
            </label>
          </div>
        </div>
```

```jsx
        <div className="commentsContainer">
            <div className="makeComment">
                            <input type="text" placeholder='type something...' onChange={(e)=>setComment(e.target.value)}/>
                {comment.length === 0 ?
                    <button className='btn btn-primary' disabled>comment</button>
                :
                            <button className='btn btn-primary' onClick={()=>handleComment(post._id,           localStorage.getItem('username'))}
                >comment</button>
                }
            </div>
            <div className="commentsBody">
                <div className="comments">
                    {post.comments.map((comment)=>{
                        return(

                            <p><b>{comment[0]}</b> {comment[1]}</p>
                        )
                    })}
                </div>
            </div>
        </div>

        </div>
        )

        })}


    </div>


    </div>
 )
}

export default Profile
```

Error :
No Code Errors

3.Authentication

1. **User Registration:**
   - A new user signs up by providing their username, email, and password.

   - This information is sent to the backend via a POST request to the /api/v1/register endpoint.

   - The password is hashed using a library like bcrypt before being stored in the MongoDB database for security.

2. **UserLogin:**
   - A user attempts to log in by providing their username/email and password.
   - This information is sent to the backend via a POST request to the /api/v1/login endpoint.
   - The backend verifies the user's credentials by checking the database and comparing the provided password using bcrypt.
   - If the credentials are valid, a JWT (JSON Web Token) is generated and returned to the client.

3. **StoringtheToken:**

   - The client stores the JWT in local storage or cookies.This token is used to authenticate future requests to protected routes.

4. **TokenExpiration:**

   - JWTs typically have an expiration time. After expiration, the user must log in again to receive a new token.

5. **Middlewarefor Authentication:**

   - On the backend,middleware is used to protect routes.This middleware checks the validity of the JWT in incoming requests.
   - If the token is valid,there quest proceeds;otherwise,a 401 Unauthorized response is sent.

4.Authorization:

- **Role-basedAccess Control:**

  - Users can have different roles(e.g.,regularuser,admin).This is defined when the user is created or modified.
  - When a user logs in, their role is included in the JWT.

Route Protection:

- Protected routes(like creating or editing posts)check the user's role.For example, only admins may have access to user management features.
- Middleware checks the user's role before allowing access to certain routes.

  FrontendAuthorization:

- On the client side, the application can use the role information from the JWT to conditionally render components.For instance,only admins might see options to manage users.

  HandlingUnauthorized Actions:

- If a user attempts to access a resource they're not authorized for, the backend responds with a 403 Forbidden status,and the frontend can display an appropriate message.

- Detailsabout tokens, sessions, or anyother methods used:

5.Sessions:

  WhatareSessions?

o Sessions are stored on the server and maintained via sessionIDs.When a user logs in, the server creates a session and associates it with the user.

  How Sessions Could Work:

o **Session Creation**: Upon login, the server creates a session object and stores it in memory or a store like Redis. A unique session ID is generated and sent to the client.
o **ClientStorage**:The client stores the sessionID in a cookie.
o **Authorization**:For subsequent requests,the client sends the sessionID cookie, allowing the server to retrieve the corresponding session data.
o **SessionManagement**:Sessions can have a configureable expiration time,after which they become invalid.

  Benefitsof UsingSessions:

o Simpleto implement for small applications.
o Easy to manage user state across requests.

  Choosing BetweenTokens and Sessions

Considerations:

- **Scalability**:Forlargerapplications,JWTsarepreferreddduetotheirstatelessnature, which avoids server-side session storage.
- **Security**:Bothmethodscanbesecureifimplementedcorrectly,butsessionsrequire careful management of session storage and expiration.
- **UseCases**:If youneedtosupportmobileappsorthird-partyintegrations,JWTs might be more suitable due to their flexibility.
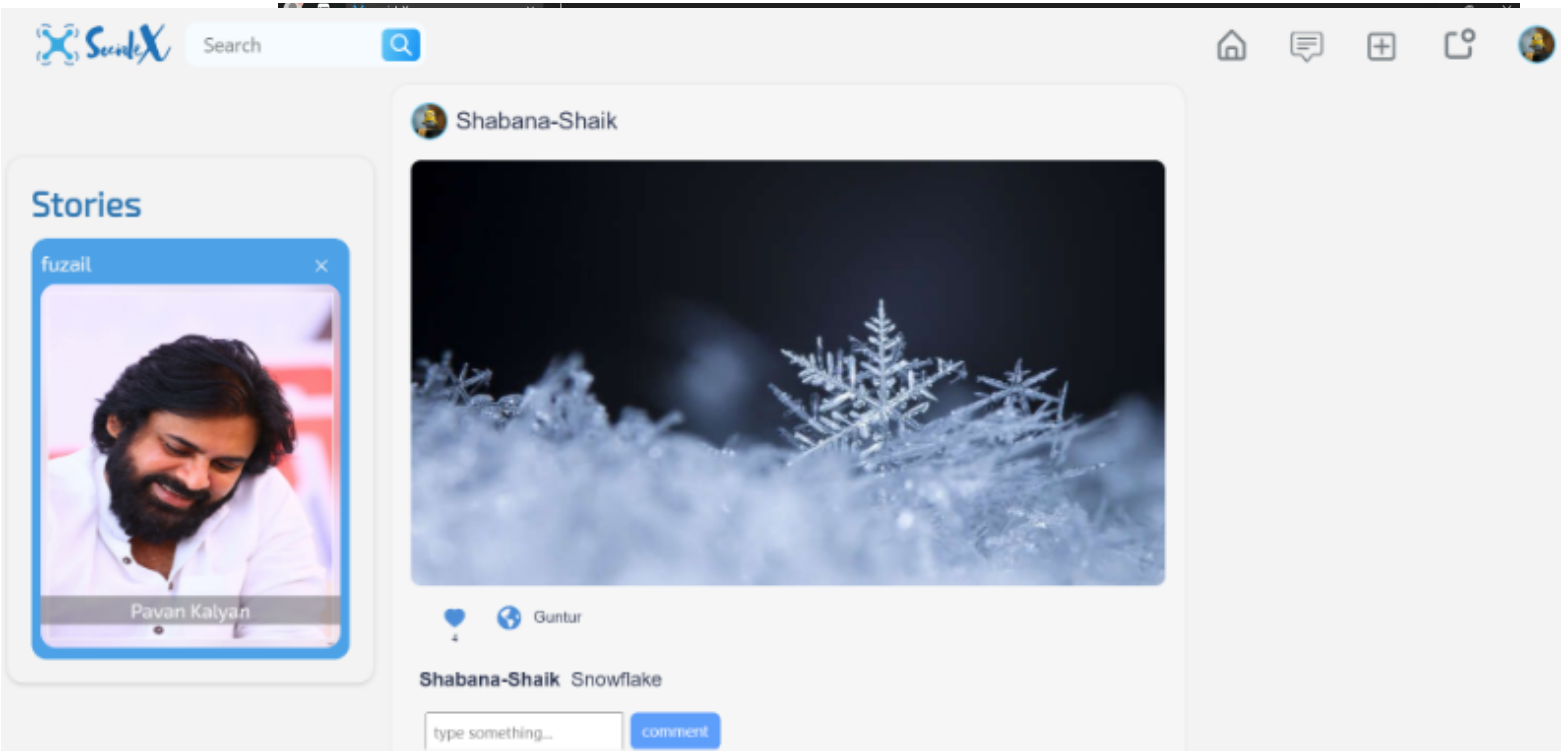
9.User Interface:

☐ Screenshots or showcasing different UI features:

☐ **Landing Page:**
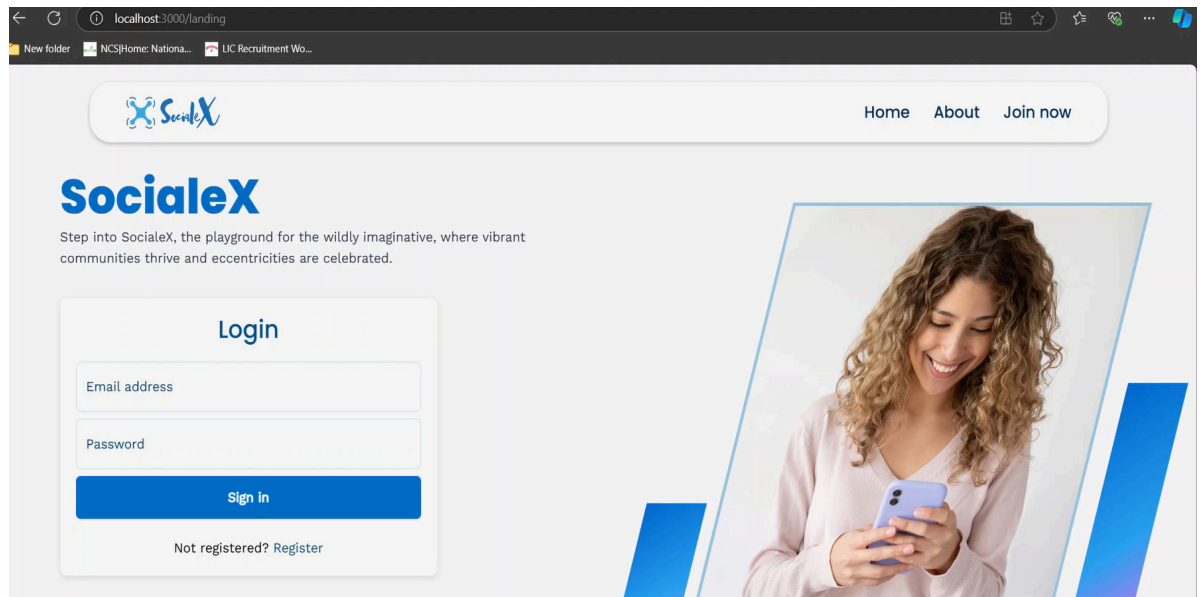
On completing the development part, we then run the application one last time to verify all the functionalities and look for any bugs in it. The user interface of the application looks a bit like the one's provided below.

Home Page:

- Discover, Connect, and Share! SocialeX is your new digital hub where stories come to life.
- Whether you're here to share your moments, connect with friends, or explore new content, we've got you covered.
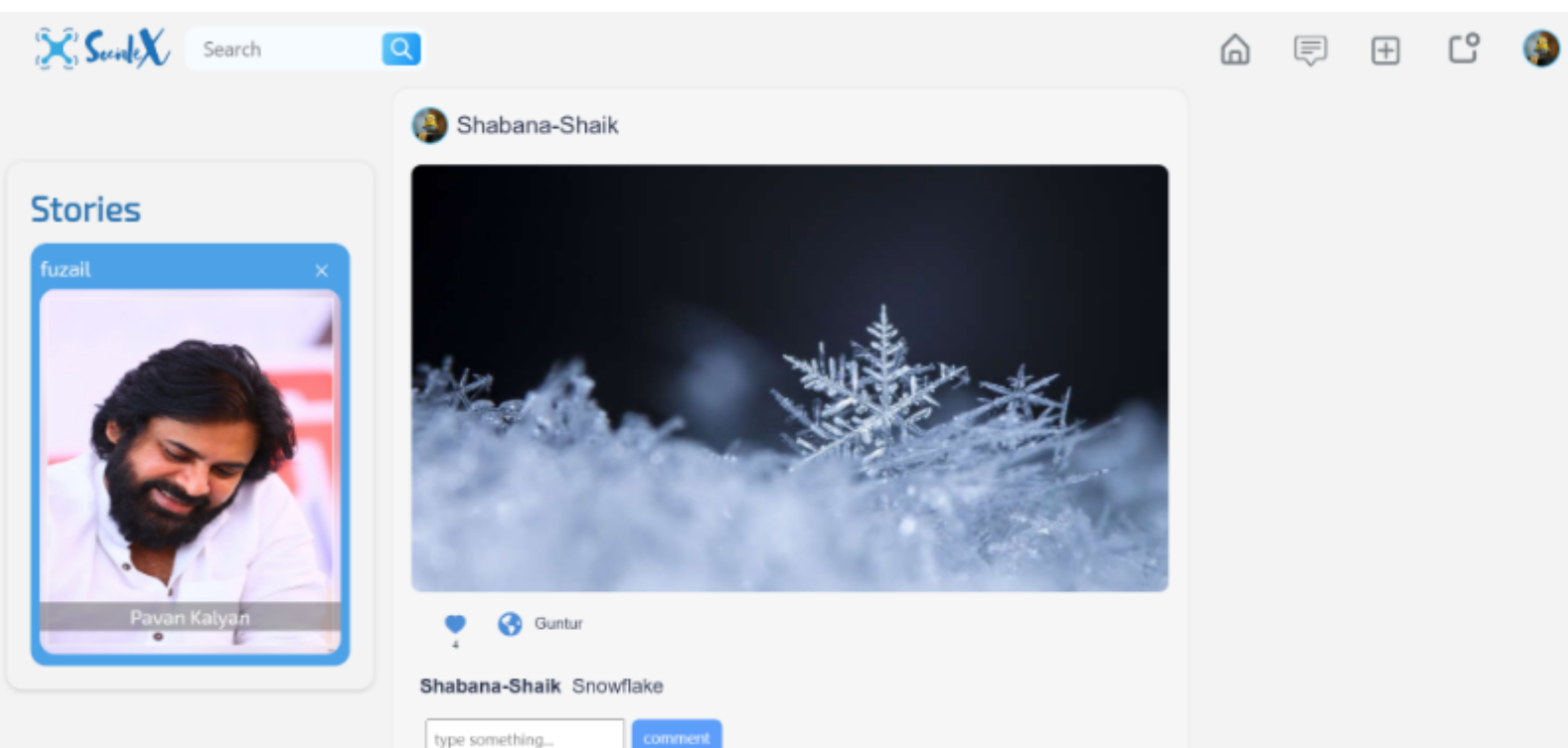
Create Posts:
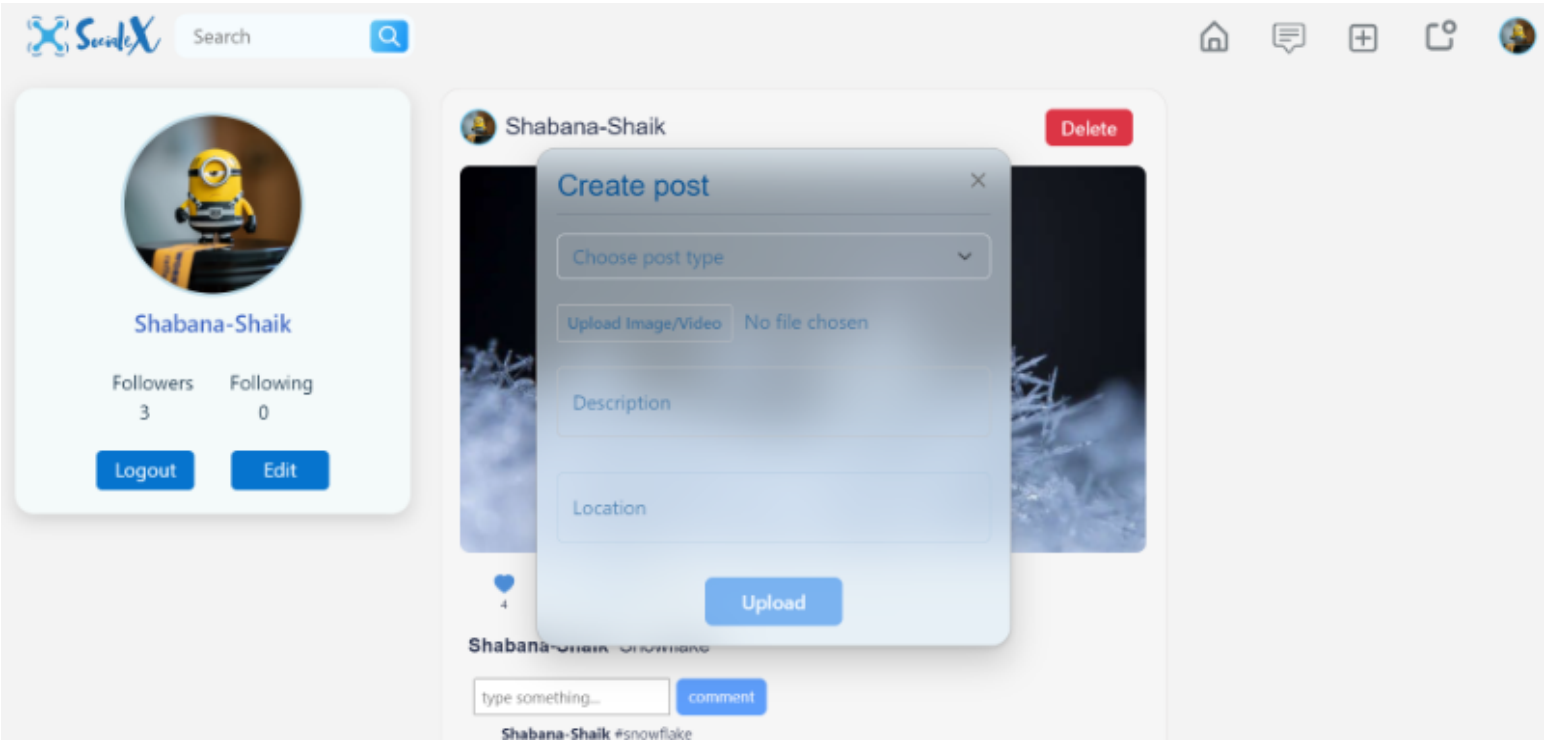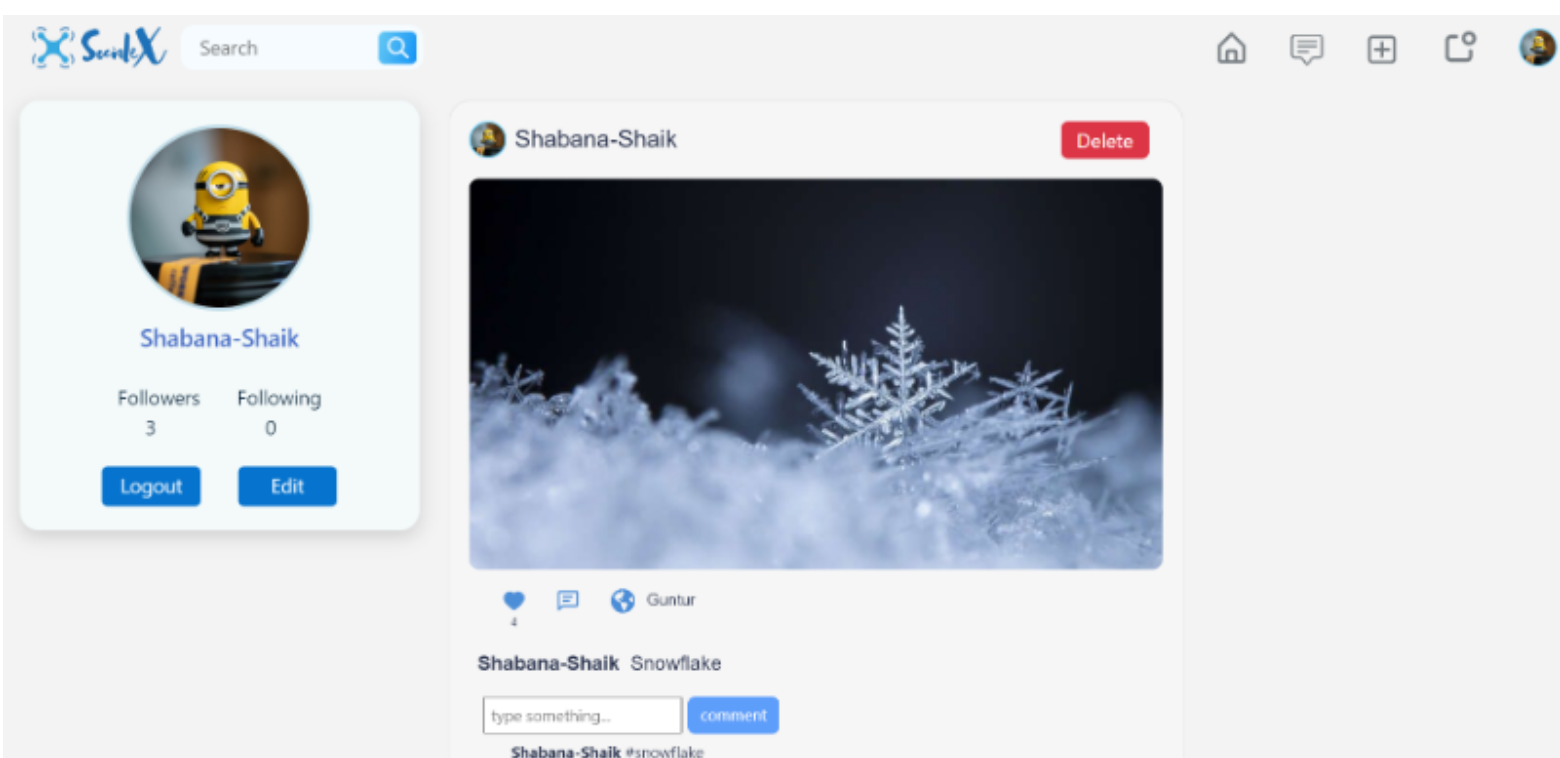
 Express yourself! Share your thoughts, experiences, and moments with your friends and followers. Whether it's a special memory, an inspiring quote, or just what's happening today, your story matters.
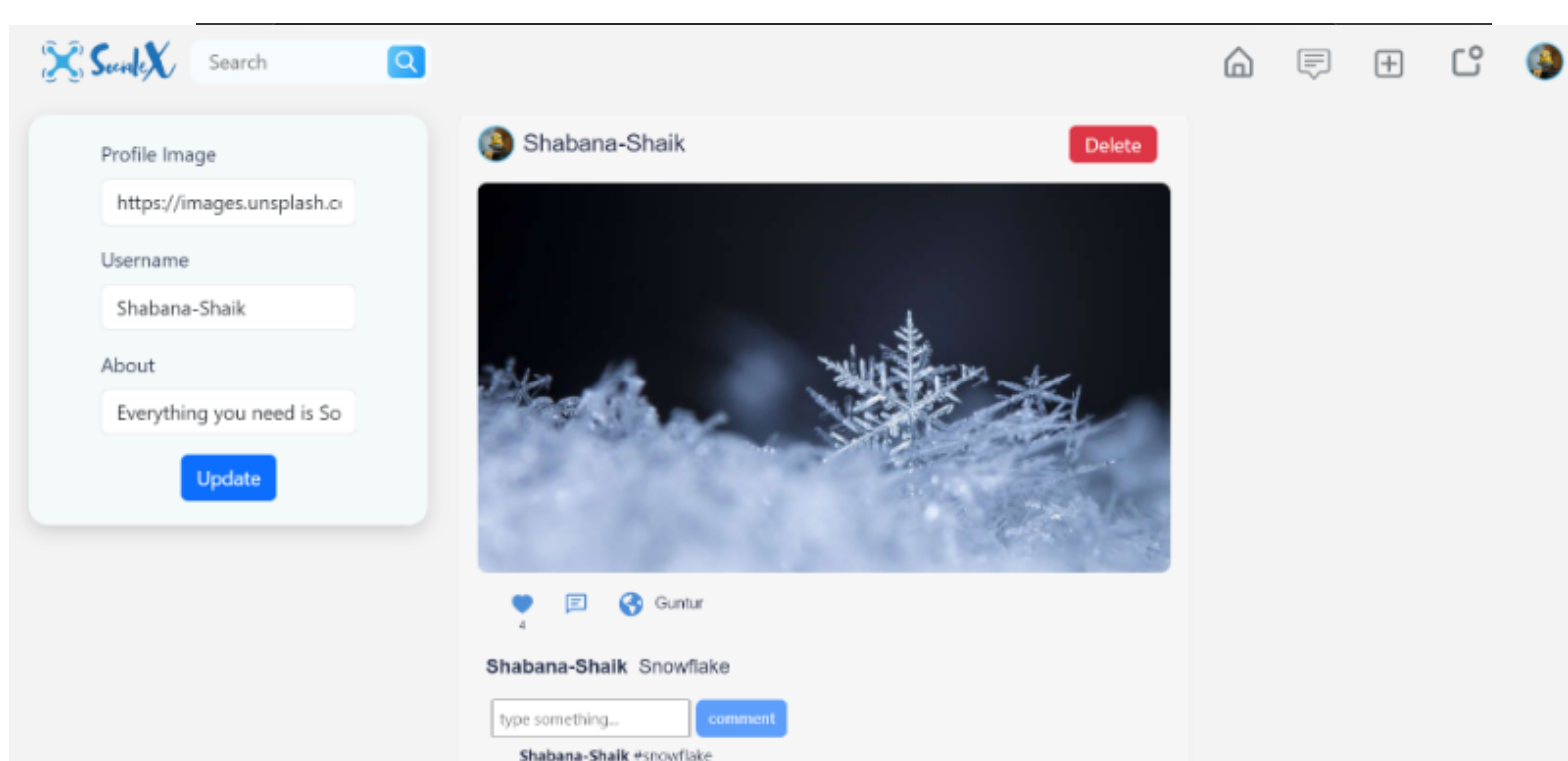
Profile:

Welcome to Your Personal Space!

Showcase Yourself:

Profile Picture: Upload a picture to put a face to your name. It's the first thing people see, so make it count!
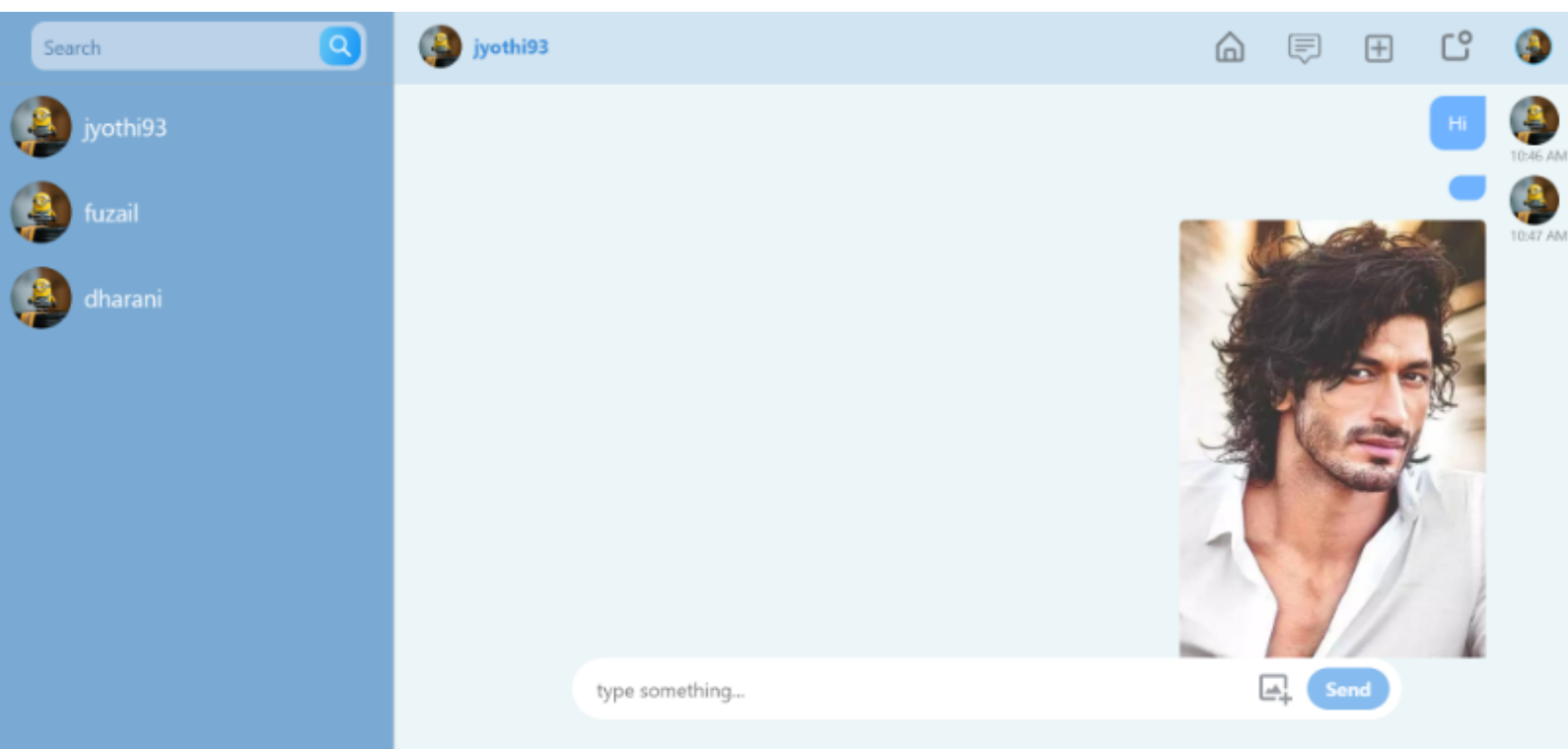
Bio: Write a short bio to let others know who you are. Share your interests, passions, and a bit about your journey.

Update Profile:

☐ **Update Your Profile:**
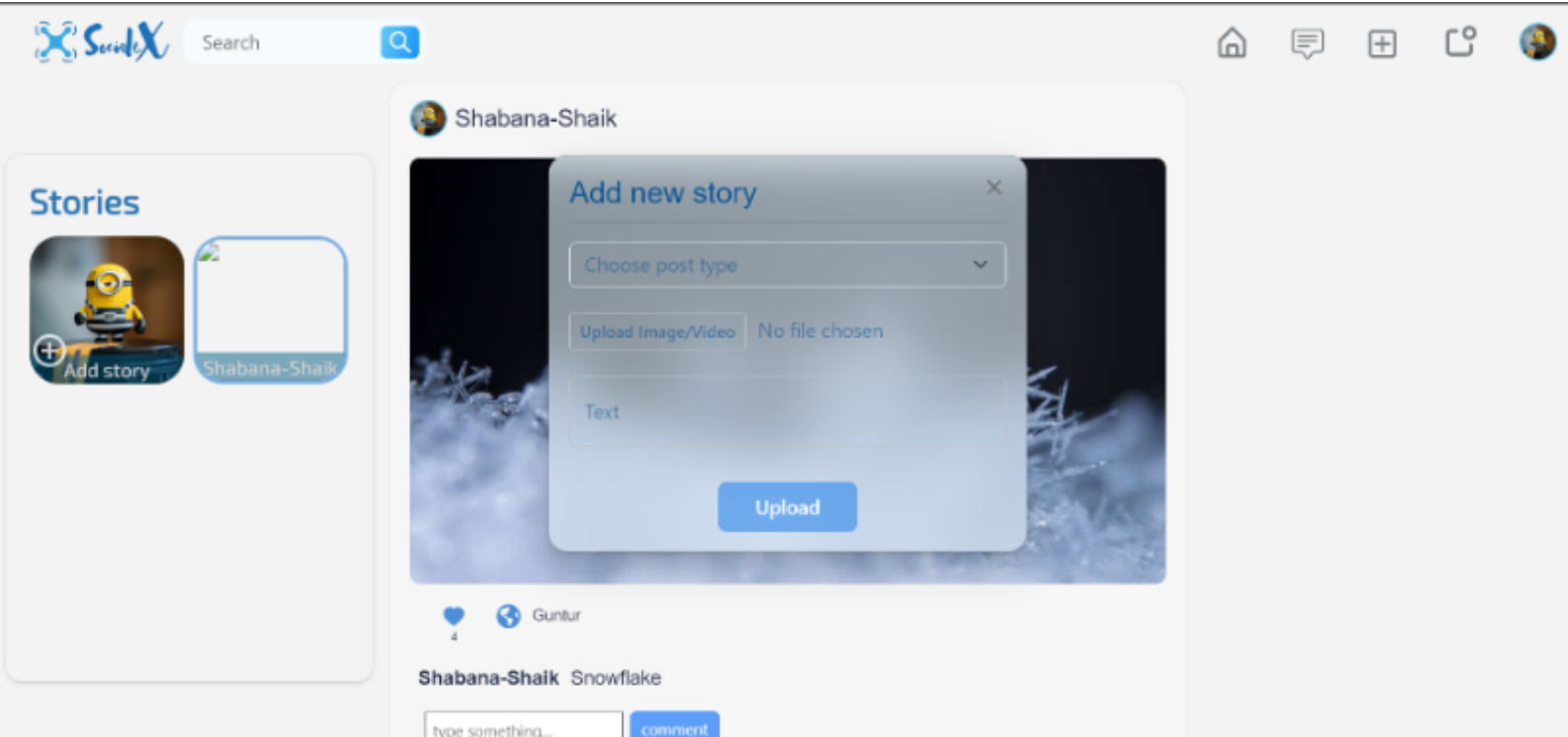
☐ Refresh Your Look

☐ Edit Your About
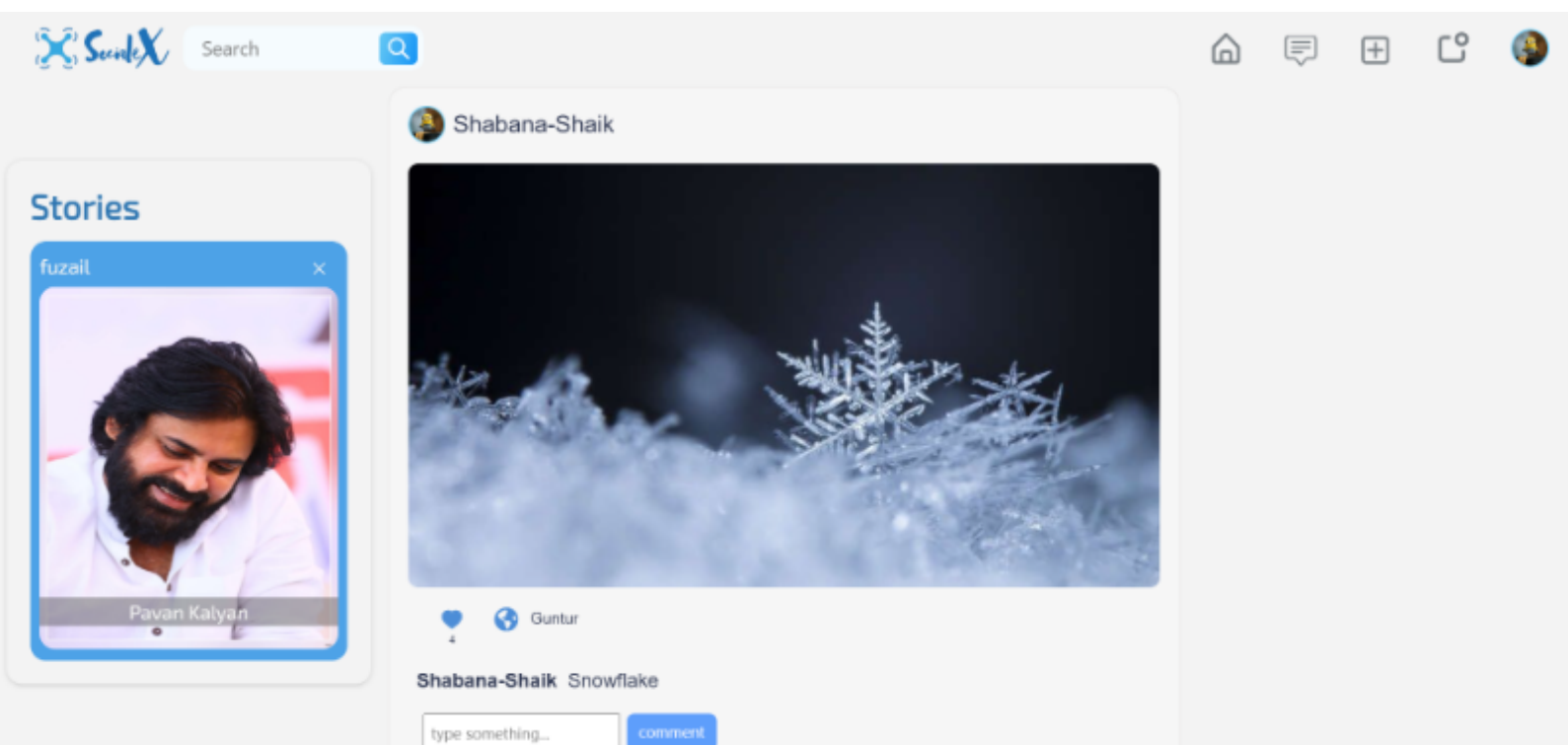
☐ Edit Your Posts

Chats:

Your Chats

- Stay Connected
- Chat History
- Media Sharing
- Privacy and Security

Create Stories:

 Share your day with quick, ephemeral posts that disappear after 24 hours. Snap a photo, record a video, or type a short message to let your friends know what you're up to.

10. Testing

Testing

Strategy:

1. Functional Testing:

- **Description:** Evaluates the system's functionality against the defined requirements.
- **Example:** Verifying user login, data processing, and report generation functionalities.

2. Performance Testing:

- **Description:** Assesses the system's performance under various conditions, including load, stress, and scalability.
- **Example:** Testing the system's response time with a high number of concurrent users.

3. Security Testing:

- **Description:** Identifies vulnerabilities and ensures the system is protected against threats.
- Example: Conducting penetration testing to identify potential security breaches.

4. Usability Testing:

- **Description:** Evaluates the user interface and user experience, ensuring the system is intuitive and user-friendly.
- **Example:** Testing the ease of navigation and the clarity of error messages.

5. Compatibility Testing

- **Description:** Ensures the system works across different browsers, devices, and operating systems.
- **Example:** Verifying that the system functions correctly on both iOS and Android devices.
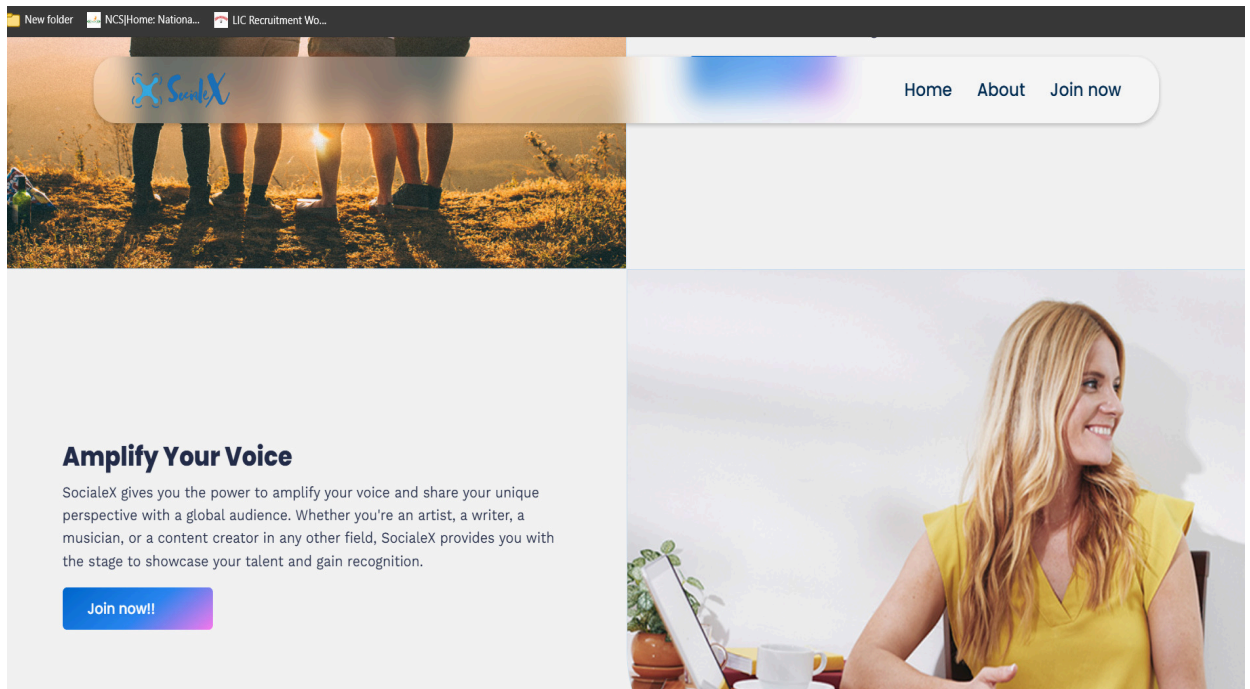
6. Regression Testing:

- **Description:** Ensures that new code changes do not adversely affect existing functionalities.
- **Example:** Running tests on the entire system after a new feature is added.

11.Screenshots or Demo:

- Screenshots or a link to a demo(if available)to showcase the application:

- **Landing Page:**
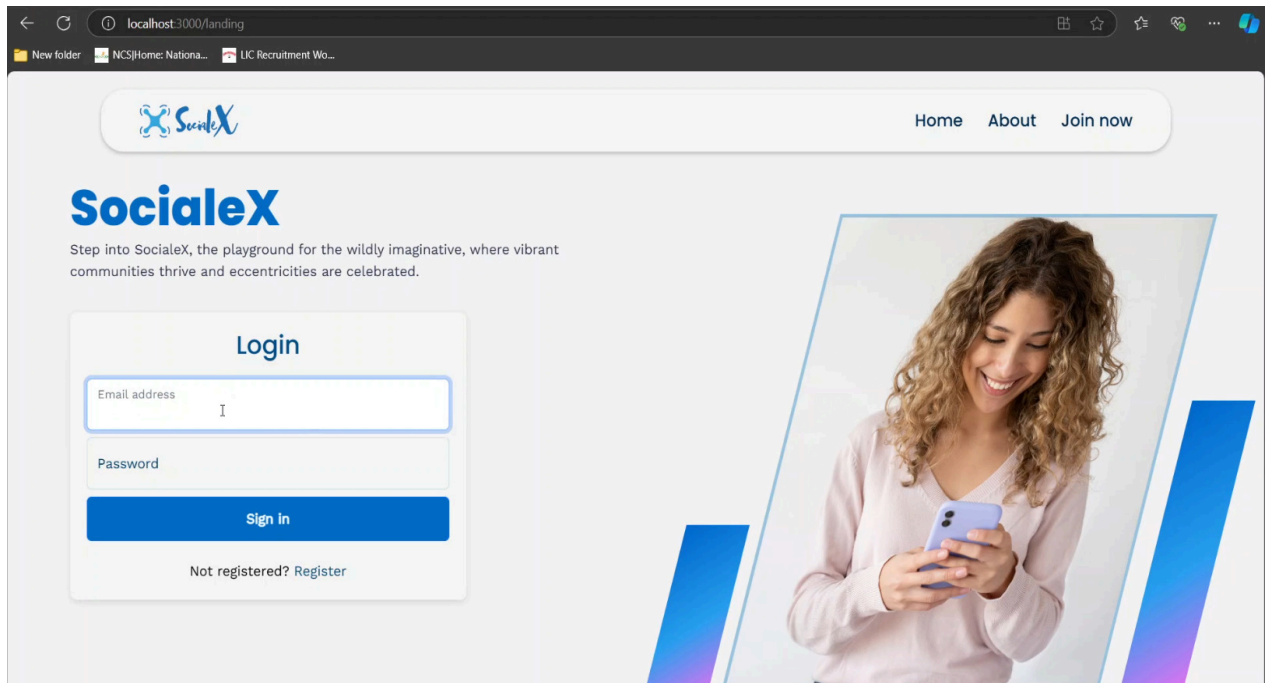
- Demo   Video   Link: https://drive.google.com/file/d/1P5c5zMuf3wmtax_14Ls74Y0JB9OIm1dE/view?usp=drivesdk

- Code Link:

   https://github.com/Muskan-0000s/SocialeX-App

12. Known Issues

- Here are some potential issues that might be encountered:

User Interface/Experience Issues

Navigation Difficulties:
- **Problem:** Users may find navigation between different sections (like the dashboard, posts, and settings) confusing.
- **Solutions:**
  1. **Improve UI Design:** Simplify and streamline the navigation menu. Use clear labels and icons.
  2. **User Feedback:** Collect feedback from users to identify confusing areas and make necessary adjustments.

Mobile Responsiveness:
- **Problem:** Some users might experience layout or functionality issues when accessing the platform on mobile devices.
- **Solutions:**
  1. **Responsive Design:** Ensure that the app uses responsive design principles to adapt to various screen sizes.

2. **Testing:** Perform thorough testing on different devices and screen sizes to identify and fix issues.

Performance Problems

Slow Loading Times:
- **Problem:** Posts with heavy media content may lead to longer loading times, affecting user engagement.
- **Solutions:**
    1. **Optimize Media:** Compress images and videos before uploading to reduce their size.
    2. **Caching:** Implement caching strategies to speed up the loading of frequently accessed content.

Server Downtime:
- **Problem:** Occasional outages or slow server responses can disrupt the user experience.
- **Solutions:**
    1. **Server Monitoring:** Implement server monitoring tools to detect and resolve issues quickly.
    2. **Load Balancing:** Use load balancing to distribute traffic evenly across multiple servers.

Content Management Challenges

Draft Recovery:
- **Problem:** Users might face issues recovering unsaved drafts or auto-saves not functioning properly.
- **Solutions:**
    1. **Auto-Save Feature:** Ensure that the auto-save feature works reliably and frequently saves drafts.
    2. **Draft Management:** Provide users with easy access to their saved drafts.

Version Control:
- **Problem:** Lack of effective version control for posts can lead to confusion, especially in multi-author environments.
- **Solutions:**
    1. **Version History:** Implement a version history feature that allows users to see and revert to previous versions of posts.
    2. **Collaboration Tools:** Provide tools for authors to collaborate and manage versions effectively.

Search and Discovery Limitations

Ineffective Search Functionality:
- **Problem:** Users may struggle to find specific posts or topics due to inadequate search filters or algorithms.
- **Solutions:**
    1. **Advanced Search Filters:** Implement advanced search filters to help users narrow down their search results.
    2. **Search Optimization:** Improve the search algorithm to provide more relevant results.

Limited Tagging and Categorization:
- **Problem:** A poor tagging system may hinder content discovery and organization.
- **Solutions:**
    1. **Enhanced Tagging:** Improve the tagging system to allow for more detailed and specific tagging.
    2. **Content Categories:** Introduce a comprehensive categorization system to help users find content more easily.

Commenting and Community Engagement Issues

Spam and Moderation:
- **Problem:** Users may report spam comments and a lack of effective moderation tools.
- **Solutions:**

1. **Moderation Tools:** Provide moderators with robust tools to manage and filter spam comments.
2. **User Reporting:** Allow users to report spam and inappropriate comments easily.

Notification Delays:
- **Problem:** Notifications for comments or interactions might be slow or inconsistent, affecting community engagement.
- **Solutions:**
    1. **Real-Time Notifications:** Implement real-time notifications to keep users updated promptly.
    2. **Notification Settings:** Allow users to customize their notification settings to ensure they receive timely alerts.

13. Future Enhancements:

1. Through a graphical web user interface, users may interact with each other through chats while taking use of a secure login and authorisation approach.

2. Users can share their own own stories which can stay active for 24 hours

3. Adding the option of commenting on a particular post.

4. Adding 'related posts' section. This section will have all the posts related to the original post by location. To identify the location we will use either the title of the post or the tags provided by the user. Location refers to the country / continent.

5. Providing a separate and better 'compose' page.

6. The ability for users to add videos in posts.

7. The integration of NLP to group posts with similar hashtags.

8. The ability to integrate graph algorithms for friend recommendation systems similar to what Facebook has achieved.

9. The functionality of having an in app feature to take photos and videos with filters.

Providing third-party authentication.