**Full Stack Development with MERN**

**1. Introduction**

**• Project Title:** Social Media Web App

**• Team Members:**

Vanshika Jain: Project Manager

Shreya Sania: Lead Frontend Developer

Komal Baid: Backend Developer

Shreeya Joshi: Database Administrator

**2. Project Overview**

**• Purpose:**

The purpose of the **Sphere App** is to provide users with an interactive and immersive social media platform similar to Instagram. It allows users to post content, follow others, send real-time messages, and engage in meaningful interactions — all within a secure, scalable, and user-friendly environment. The app is designed to encourage creative sharing and community building.

**• Features:**

* User Registration and Login with JWT Authentication
* Real-time Chat with Typing Indicator and Message Status
* Create, Like, Comment, and Save Posts
* Explore Feed and Follow/Unfollow Users
* Profile Customization and Image Upload
* Post and Chat History
* Secure Password Encryption and Role-based Access
* Mobile-responsive U

**3. Architecture**

• Frontend:

The frontend is developed using **React.js**, offering a dynamic and responsive UI. It includes:

* **Redux** for global state management
* **React Router** for seamless navigation between routes
* **Axios** for making API requests to the backend
* **Tailwind CSS** (or your styling framework) for component styling

• Backend:

The backend is built with **Node.js** and **Express.js**, structured with RESTful API endpoints. Key functionalities include:

* User registration, login, and profile management
* Post creation, editing, deleting, and saving
* Real-time messaging using **Socket.io**
* API security via **JWT tokens**

• Database:

The app uses **MongoDB** as the database to store structured and relational data.

* **Mongoose ODM** is used for modeling schemas
* Collections include **Users**, **Posts**, **Messages**, **Comments**, and **SavedPosts**
* Media files (images, profile pics) can be stored using **Firebase Storage** or **AWS S3**

**4. Setup Instructions**

• Prerequisites:

* **Node.js** (v14 or above)
* **MongoDB** (v4 or above) or **MongoDB Atlas**
* **Git**
* **Socket.io**

• Installation:

git clone https://github.com/SmartBridge/sphere-app.git

Navigate to the project directory: cd sphere-app

Install frontend dependencies: cd client && npm install

Install backend dependencies: cd ../server && npm install

Set up environment variables: Create a .env file in the server directory and add the required variables.

**5. Folder Structure**

• Client:

The React frontend is organized as follows:

client/

├── public/

├── src/

├──index.html

├── vite.config.js

├── package.json

├── package-lock.json

• Server:

The Node.js backend is organized as follows:

backend/

├── models/

├── routes/

├── .env/

├── db.js

├── server.js

├── index.js

├── server.js

├── testpassword.js

├── testserver.js

├── package.json

├── package-lock.json

* README.md

**6. Running the Application**

• To start the frontend server:

Bash:

cd client

npm start

• To start the backend server:

Bash:

cd server

npm start

**7. API Documentation**

**User Management**

* **POST /api/auth/register –** Registers a new user.
* Parameters:json

{ "username": "ABC", "password": "my\_password" }

* Response:json

{ "message": "User registered successfully" }

**Transactions**

* **POST /api/auth/login** – Authenticates user and returns a JWT
* **POST /api/withdraw** – Authenticates user and returns a JWT
* **Response**.json

{

"token": "jwt-token-here",

"user": {

"username": "ABC",

"email": "ABC@example.com"

}

}

* **GET /api/user/:id** – Retrieves user profile by ID.
* **PUT /api/user/:id** – Updates user profile info such as bio or profile image

**Post and Interaction**

* **POST /api/posts –** Creates a new post.
* **Request.**json :

{

"userId": "12345@ac",

"caption": "Beautiful sunset!",

"image": "image-url"

}

* **GET /api/posts/:id – Fetch a specific post by ID.**
* **PUT /api/posts/like/:id – Like or unlike a post.**
* **DELETE /api/posts/:id – Delete a post.**
* **POST /api/posts/comment/:id – Add a comment to a post.**

**Chat and Messaging**

* **POST /api/messages** – Sends a message between users.
* **GET /api/messages/:chatId** – Retrieves chat history between users.
* **GET /api/chats/:userId** – Fetches all chat conversations for a user.

**Follow System**

* **PUT /api/users/follow/:id** – Follow a user.
* **PUT /api/users/unfollow/:id** – Unfollow a user.
* **GET /api/users/followers/:id** – Get followers of a user.
* **GET /api/users/following/:id** – Get users followed by the user.

.

**8. Authentication**

• The app uses **JWT (JSON Web Tokens)** for stateless authentication.

• After a successful login, the client stores the token and sends it in the Authorization header for protected routes.

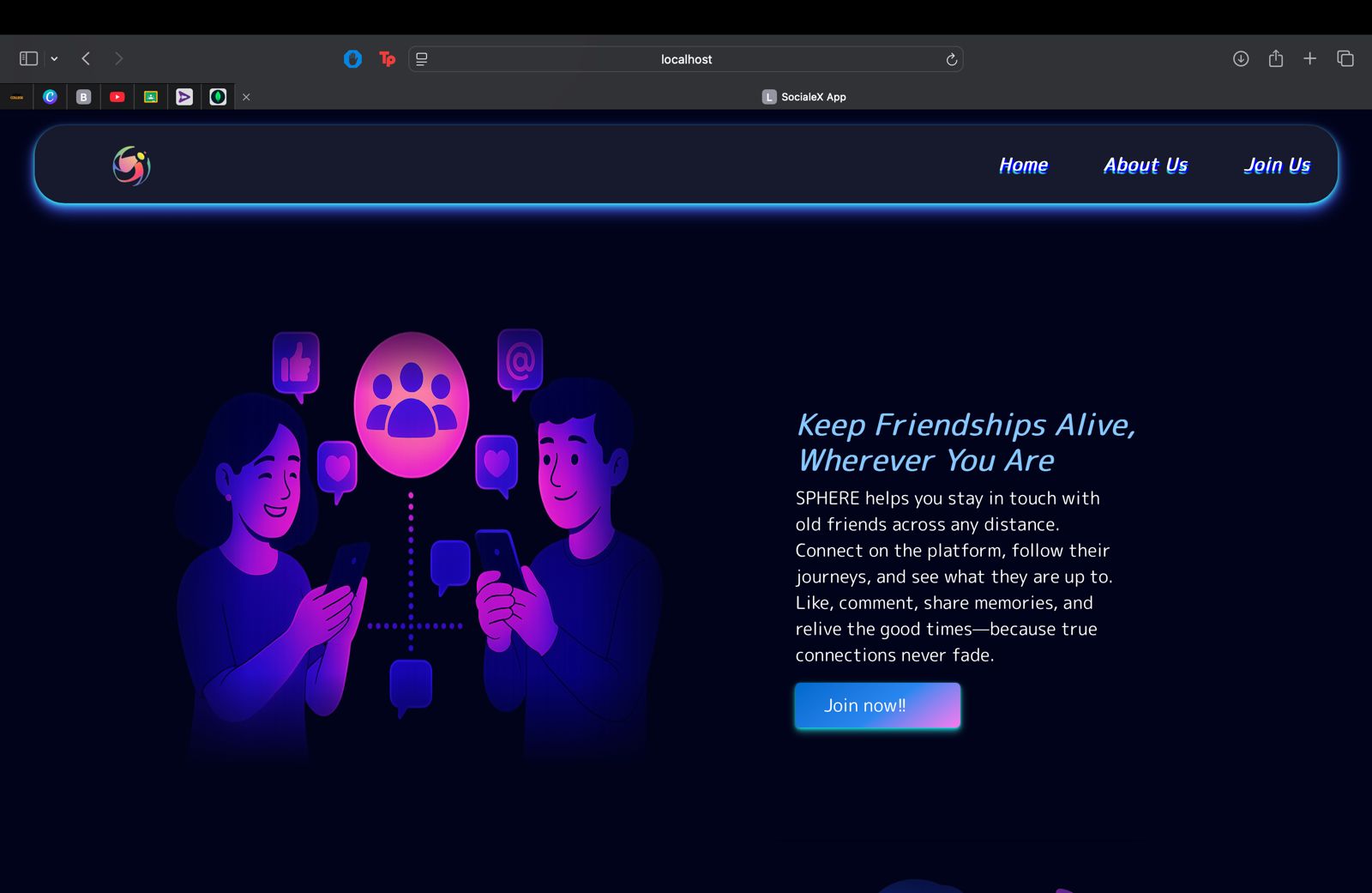
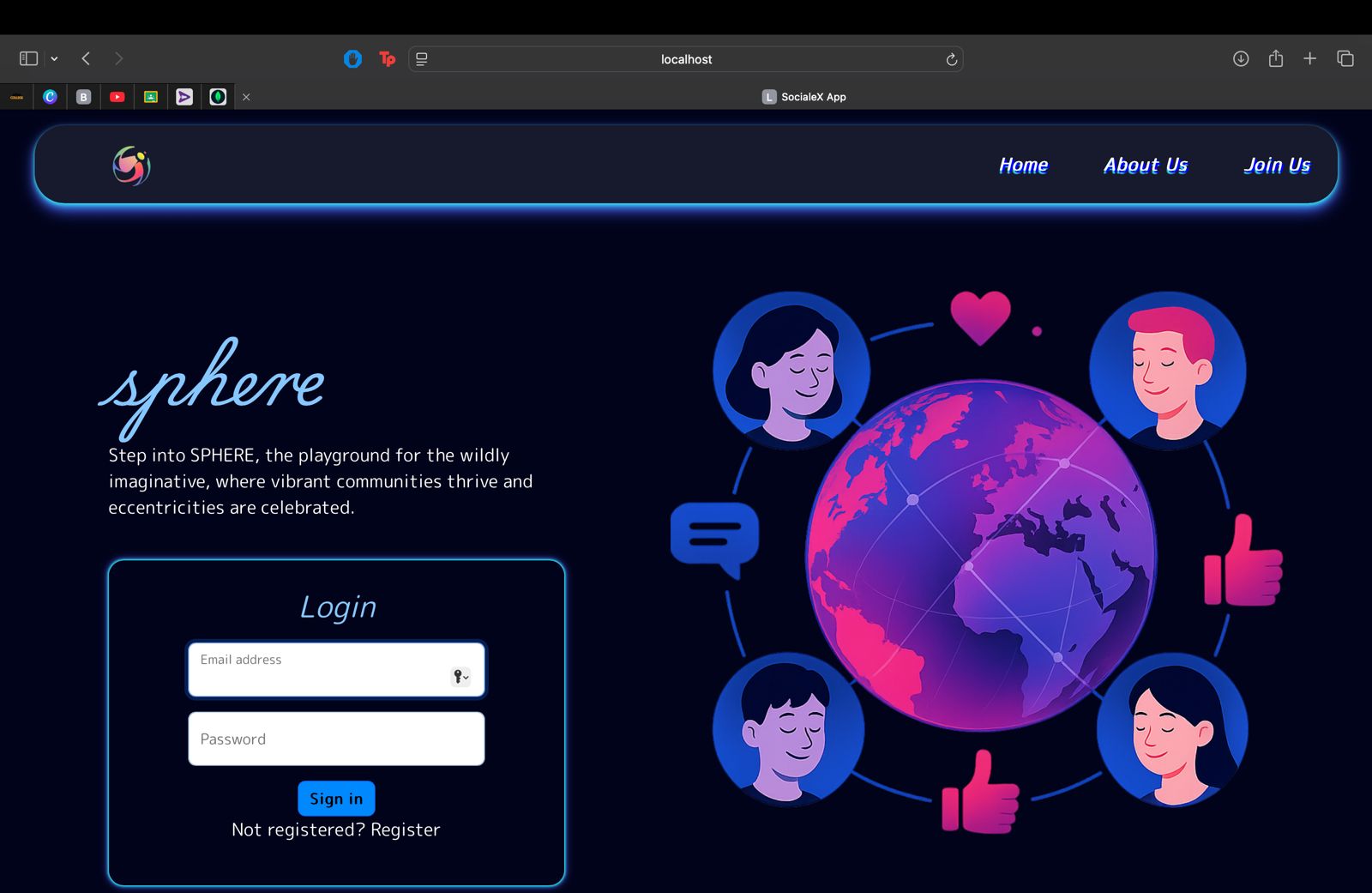
• Sessions are not used; all authentication is stateless.

**9. User Interface**

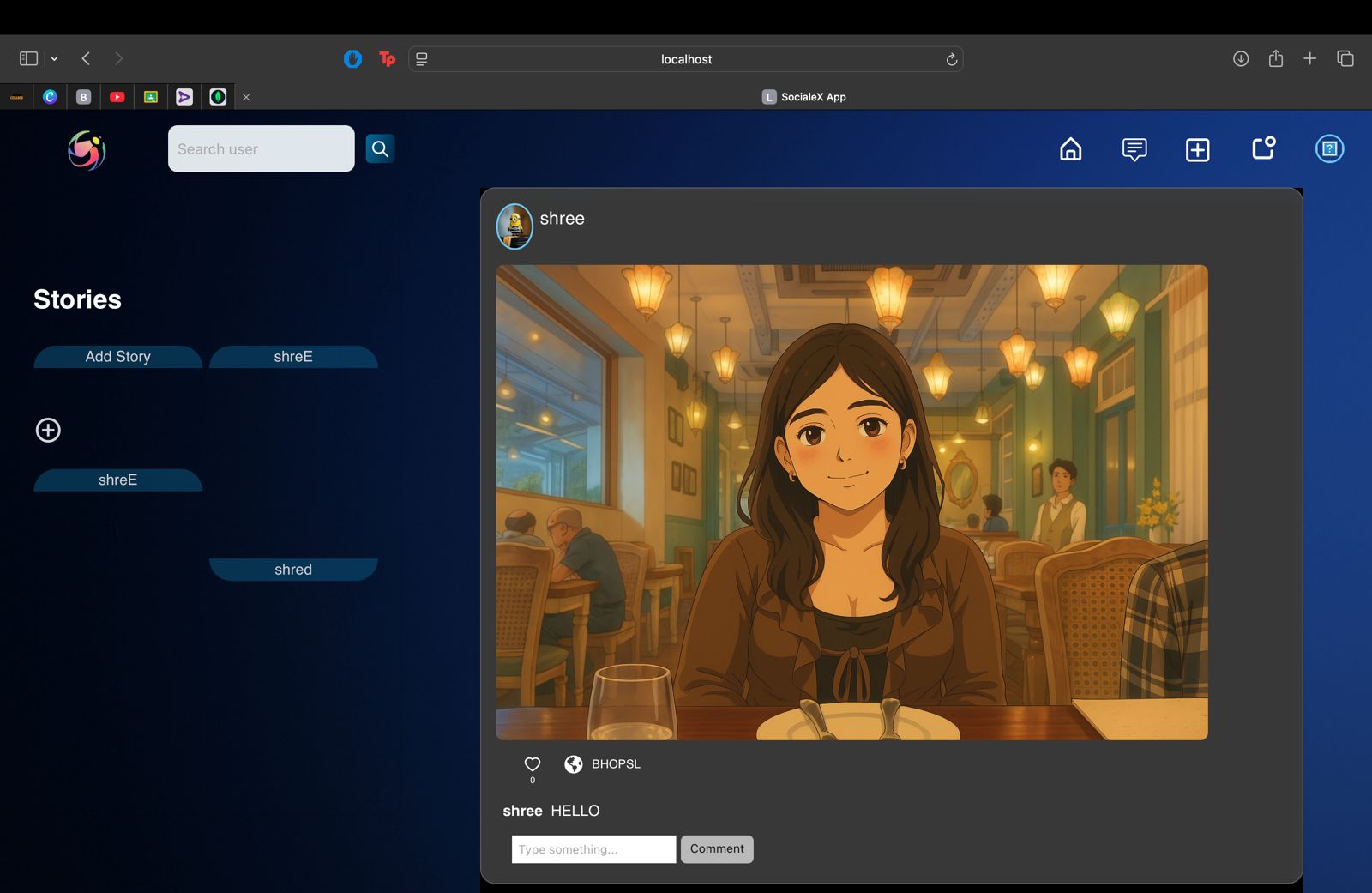
• The UI of **Sphere App** is built using **React.js** and is optimized for responsiveness and smooth navigation. The design is modern, minimal, and optimized for both desktop and mobile users.

**Key Screens Include**:

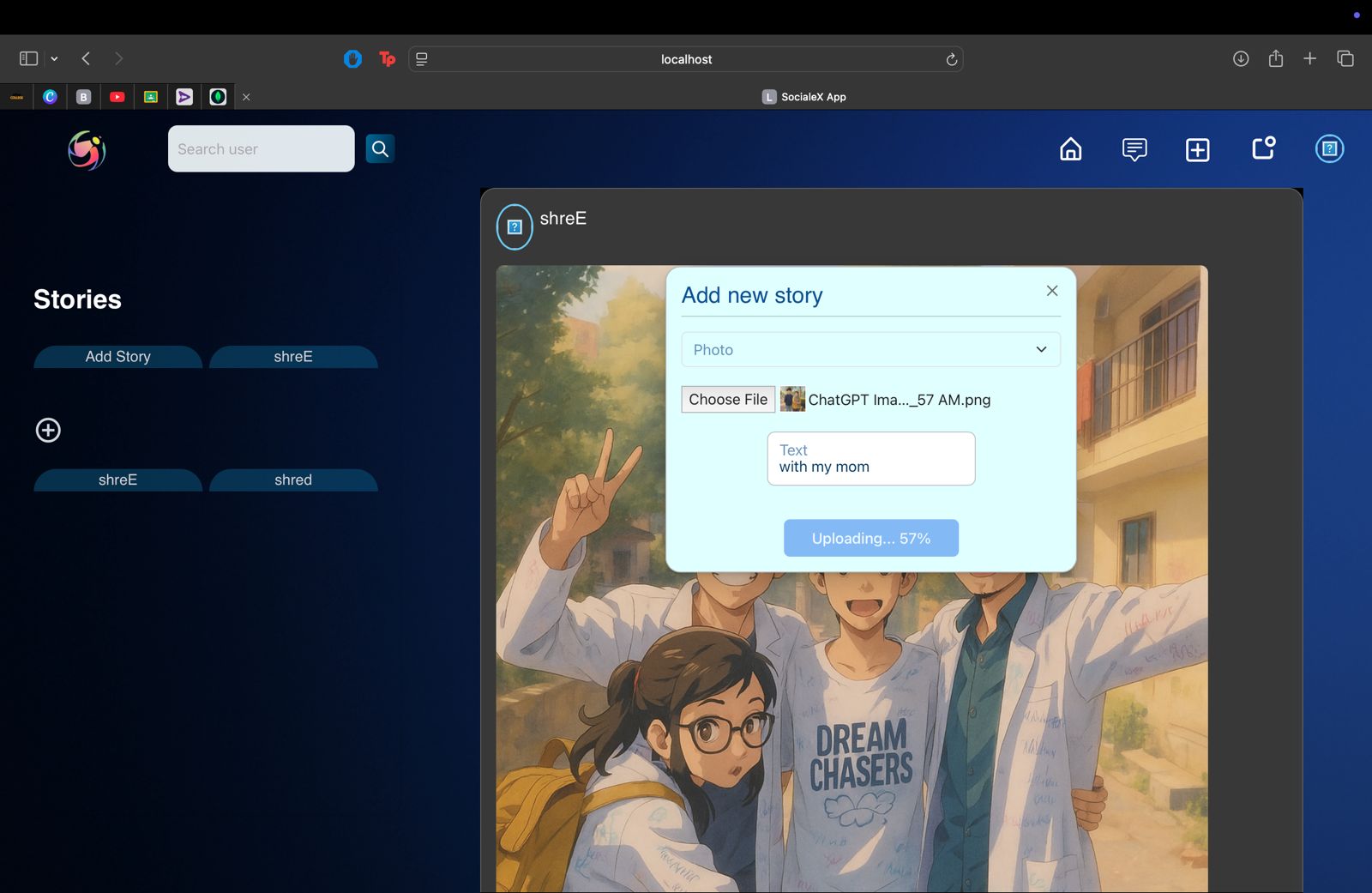
* Login / Registration Page
* Feed Page (Explore posts from all users)
* Profile Page with User Info and Posts
* Post Upload Form with Preview
* Real-time Chat Window with Online Status
* Notifications Panel
* Saved Posts and Follow Suggestions



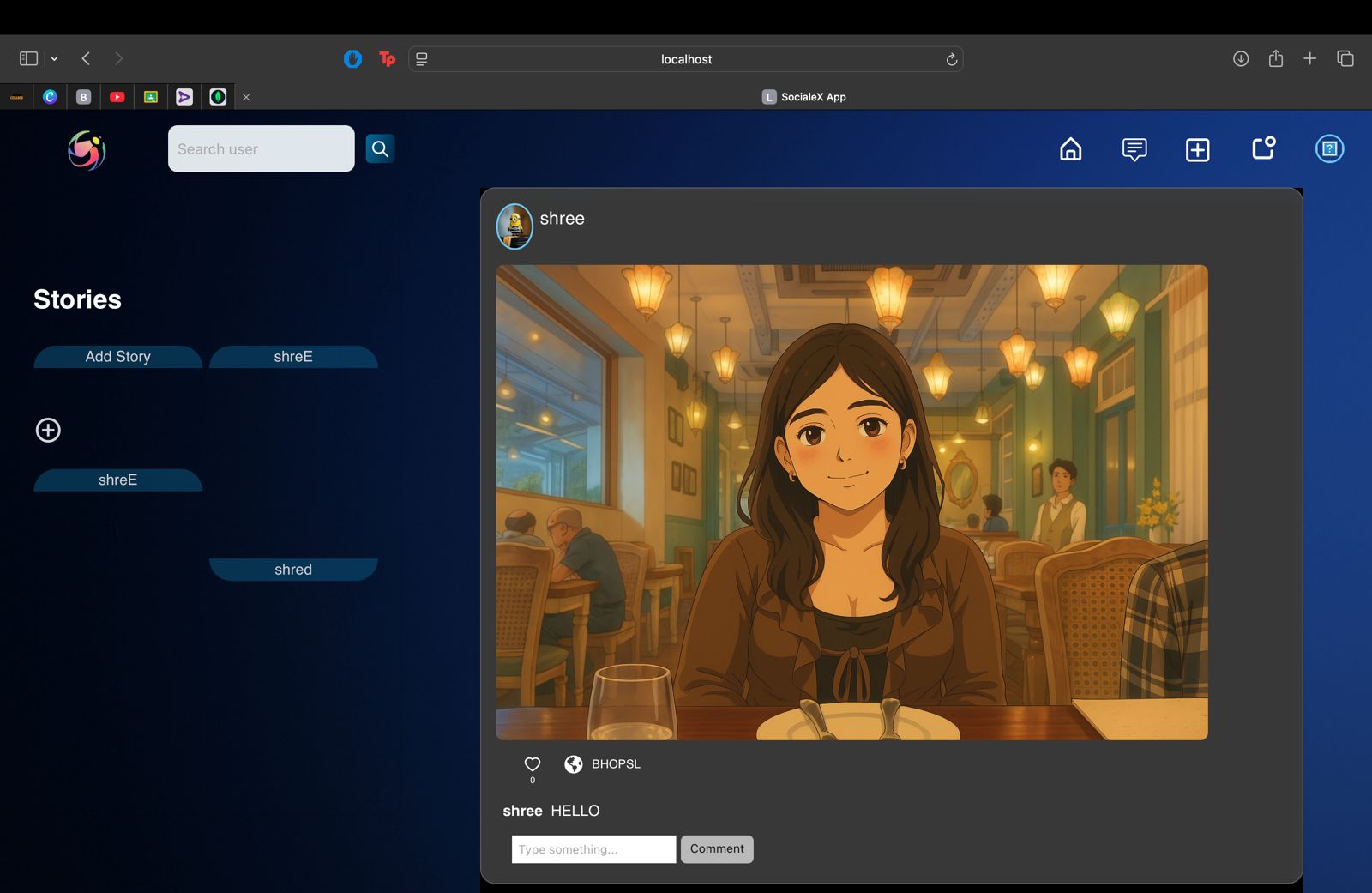
* Feed Page (Explore posts from all users)



* Profile Page with User Info and Posts



* Post Upload Form with Preview



**10. Testing**

* The testing strategy for **Sphere App** is designed to ensure robust performance, functional correctness, and high security across the application stack.
* It follows a **multi-layered testing approach** that includes the following levels:

**Unit Testing**

* Focuses on testing individual components such as authentication, post creation, and messaging logic.
* Tools: **Jest**, **React Testing Library** for frontend, **Mocha/Chai** for backend.

**Integration Testing**

* Verifies the interaction between modules, such as user authentication with JWT and database updates for likes/comments.
* Ensures Mongoose models work correctly with Express routes and API responses are as expected.

**End-to-End (E2E) Testing**

* Simulates real user workflows: register ➝ login ➝ post ➝ like ➝ comment ➝ follow ➝ chat.
* Tools: **Cypress** or **Puppeteer** for browser-based testing of critical user paths.

**Performance Testing**

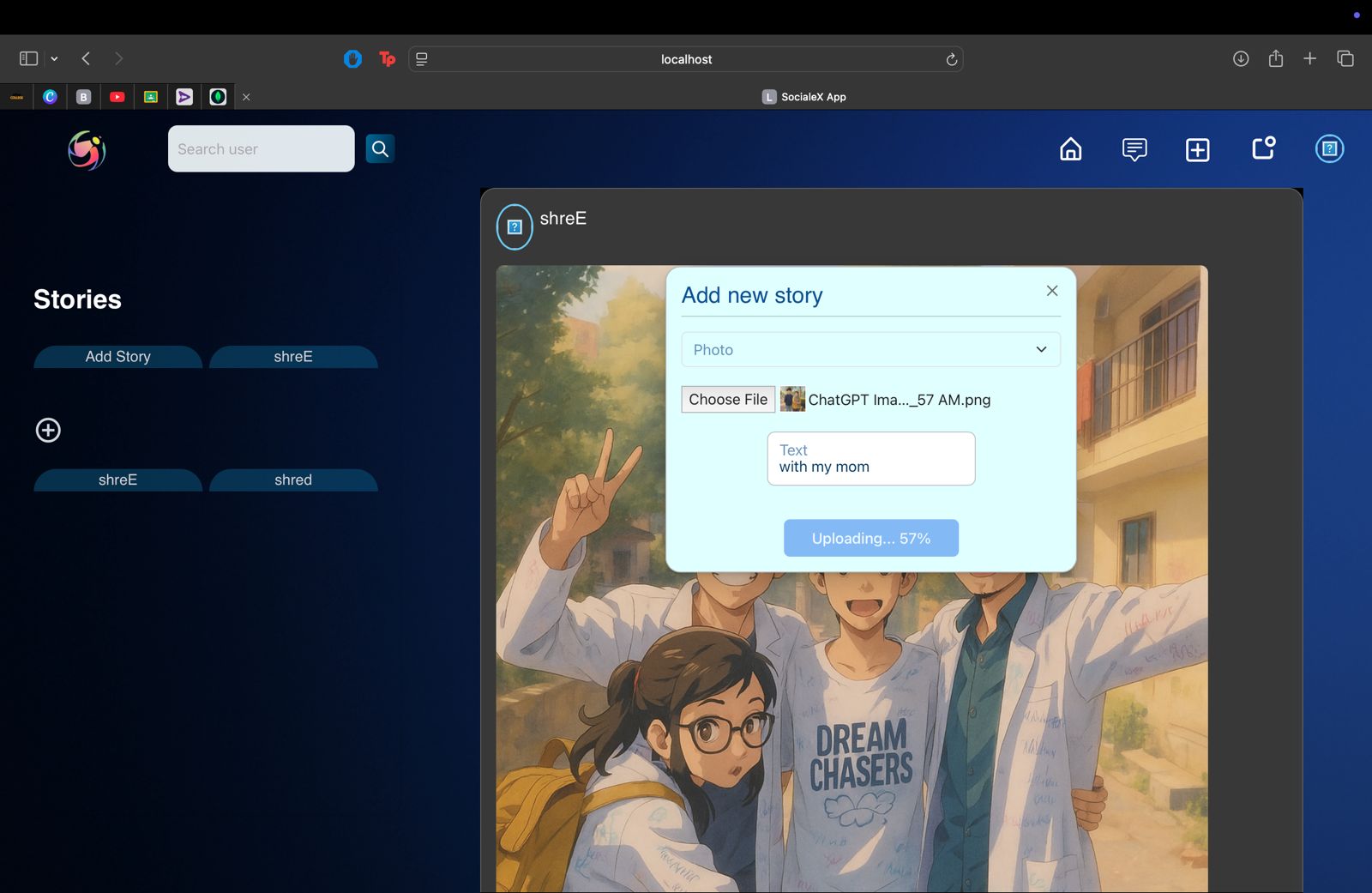
* Evaluates how the app behaves under high traffic conditions (multiple users posting, messaging, and exploring).
* Uses stress test tools like **Artillery** or **JMeter** to simulate concurrent users.

**Security Testing**

* Ensures all sensitive routes are protected using JWT and access control.
* Tests for vulnerabilities such as XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), and SQL/NoSQL injection.

**11. Screenshots or Demo**

**• Additional screenshots:**



**12. Known Issues**

* **Real-time Messaging Delay:** In some scenarios, real-time messages may experience a slight delay due to Socket.io reconnections. Optimization of event handling and connection stability is in progress.
* **Media Upload Time:** Large image or video uploads may take longer depending on the user's network. Improvements to file compression and upload speed are planned.
* **Mobile Responsiveness:** While the app is responsive, certain UI elements may appear misaligned on smaller screen devices. UI enhancements for mobile experience are underway.
* **Notification Sync Issues:** Occasionally, notifications may not sync instantly across devices. Sync reliability is being improved.

**13. Future Enhancements**

* **Mobile App Development:** Build native versions of the app for iOS and Android using React Native or Flutter.
* **AI-Powered Recommendations:** Integrate content recommendation based on user behavior and interest.
* **End-to-End Encrypted Messaging:** Implement stronger message privacy using client-side encryption.
* **Advanced Analytics Dashboard:** Provide users with detailed insights on engagement (likes, followers, reach).
* **Live Streaming Feature:** Enable users to go live, interact with followers in real time.
* **Multi-Language Support:** Add support for multiple languages to reach a broader audience.
* **Group Messaging and Channels:** Expand chat functionality to support group chats or broadcast channels like Telegram.