WeShare Social Media App - Project Documentation

Project Title: WeShare Social Media App

Project Description

WeShare is a social media application built using the MERN stack (MongoDB, Express.js, React, Node.js). The app allows users to create profiles, follow others, share posts with images, send messages, and manage events. This socially engaging platform enhances user connectivity through features like a posts feed, real-time messaging, and community events, providing a seamless and interactive experience for users.

Table of Contents

1 Overview

WeShare aims to:

- Enable users to connect through profiles, posts, and messaging.
- Provide a platform for sharing and discovering events.
- Foster community engagement with features like pages and groups.
- Offer a user-friendly interface with accessible navigation.

2 Features

User Features:

- Register/Login with profile creation
- Follow/unfollow users and view followers/following lists
- Create posts with text and images, like, comment, and share
- Send and receive messages in Primary, General, and Requests tabs
- View and join upcoming events
- Join communities like UI/UX Community, Web Designer, etc.

Admin Features:

- Manage user accounts (future enhancement)
- Moderate posts and events (future enhancement)

3 Tech Stack

Frontend:

- React.js
- Tailwind CSS
- React Router, React Toastify

Backend:

- Node.js + Express.js
- MongoDB Atlas + Mongoose
- JWT Authentication

Hosting:

- Vercel or Netlify (frontend)
- Render or Heroku (backend)
- MongoDB Atlas (database)

4 Project Structure

```
we-share/
  client/                  # React frontend
     src/
        components/       # Reusable React components
        pages/            # Page components (Feed, Messages, Events)
        App.js            # Main App component
        index.js          # Entry point
  server/                  # Node.js/Express backend
     models/              # MongoDB schemas (User, Post, Message, Event)
     routes/              # API routes
     controllers/         # Route handlers
     server.js            # Main server file
  package.json             # Project dependencies
```

5 Implementation Plan

5.1 Backend Setup (Node.js/Express/MongoDB)

Database Models:

- User: { username, email, password, followers, following, profilePic }
- Post: { userId, content, image, likes, comments, shares, createdAt }
- Message: { senderId, receiverId, content, createdAt }
- Event: { title, description, date, location }

API Routes:

- /api/users: Register, login, get user profile, follow/unfollow
- /api/posts: Create, get feed, like, comment, share
- /api/messages: Send message, get conversations
- /api/events: Get events, join event

Setup:

- Install dependencies: express, mongoose, dotenv, cors
- Connect to MongoDB using Mongoose
- Create routes and controllers for each feature

5.2 Frontend Setup (React)

Components:

- Sidebar: Navigation links (Feed, Friends, Events, etc.)
- ProfileHeader: Display user info (followers, following, profile pic)
- PostCard: Display individual posts with like/comment/share options
- Messages: Tabs for Primary, General, Requests with conversation list
- EventsList: Display upcoming events

Pages:

- FeedPage: Main feed with posts and post creation form
- MessagesPage: Messaging interface
- EventsPage: Events listing

Setup:

- Install dependencies: react, react-router-dom, axios, tailwindcss
- Use React Router for navigation between pages
- Fetch data from backend APIs using Axios

5.3 Styling with Tailwind CSS

- Use Tailwind classes for responsive design
- Style components to match the UI (e.g., rounded avatars, card layouts, sidebar)

5.4 Key Features Implementation

User Authentication:

- Register/Login using JWT for authentication
- Store user token in localStorage for persistent login

Posts Feed:

- Fetch posts from /api/posts and display in a scrollable feed
- Allow users to create posts with text and image uploads
- Implement like, comment, and share functionality

Messaging:

- Fetch conversations from /api/messages
- Display messages in tabs (Primary, General, Requests)
- Allow sending new messages

Events:

- Fetch events from /api/events
- Display event cards with title, date, and location

6 Deployment

- **Backend**: Deploy on a platform like Render or Heroku
- **Frontend**: Deploy on Netlify or Vercel
- **Database**: Use MongoDB Atlas for cloud-hosted MongoDB

7 Next Steps

- Add real-time messaging using WebSockets
- Implement notifications for likes, comments, and messages
- Add image upload functionality using Multer and Cloudinary