Project Proposal

1. Project Title

ChatMon – A Real-Time Web Messaging Platform

---

2. Problem Statement

In today's fast-paced world, seamless digital communication is essential. However, most chat applications are either too heavy, dependent on third-party APIs, or lack real-time responsiveness.
ChatMon aims to provide a lightweight, responsive, and real-time chat experience using **WebSockets**, allowing users to communicate instantly through a clean and efficient interface built with **React and Express.js**.

---

3. System Architecture

Architecture Overview:

```
Frontend (React + Vite + TypeScript)
        ↓
Backend (Express.js + WebSocket Server)
        ↓
Database (MongoDB)
```

Flow Description:

1. The **frontend** is built with React (Vite) for fast, modular, and component-based UI rendering.

2. The **backend** (Express.js) handles API requests, user authentication, and manages WebSocket connections for real-time communication.

3. The **database** stores user profiles, chat messages, and conversation histories.

4. **JWT authentication** secures the communication between client and server.

5. Both frontend and backend are deployed on cloud platforms for high availability and scalability.

Hosting Setup:

- **Frontend:** Vercel
- **Backend:** Render or Railway
- **Database:** MongoDB Atlas or PostgreSQL (hosted on Aiven/ElephantSQL)

---

4. Key Features

Category	Features
Authentication & Authorization	User registration, login, and logout using JWT authentication.
Real-Time Messaging	Send and receive messages instantly through WebSocket connections.
User Presence	Show when users are online or offline in real time.
Chat Rooms	Users can create or join different chat rooms for group communication.
Message Persistence	Messages are stored in the database for history and reloads.
Frontend Routing	Pages: Login, Signup, Chat Dashboard, Chat Room.
Responsive Design	Optimized UI for both desktop and mobile users.
Hosting	Frontend and backend deployed on Vercel and Render for global access.

---

5. Tech Stack

Layer	Technologies Used
Frontend	React.js (with Vite), TypeScript, TailwindCSS
Backend	Node.js, Express.js, WebSocket (ws)

Database	MongoDB / PostgreSQL
Authentication	JWT-based login and signup
Hosting	Frontend: Vercel, Backend: Render/Railway, Database: MongoDB Atlas / Aiven

---

6. API Overview

Endpoint	Method	Description	Access
`/api/auth/signup`	POST	Register a new user	Public
`/api/auth/login`	POST	Authenticate user credentials	Public
`/api/users`	GET	Fetch user list	Authenticated
`/api/messages/:roomId`	GET	Retrieve messages for a specific chat room	Authenticated
`/api/messages`	POST	Send a message	Authenticated

WebSocket Events

Event	Direction	Description
`message:send`	Client → Server	Send message to room
`message:receive`	Server → Client	Broadcast new message
`user:online`	Server → Client	Notify users of new online members

---

7. Backend Functionalities

- **Authentication & Authorization:** Secure login/signup using JWT.
- **CRUD Operations:** Create, Read, Update, Delete user and message data.
- **Filtering & Pagination:** Retrieve chat messages efficiently.

- **Real-Time Communication:** Bidirectional socket connection for live chat.
- **Deployment:** Hosted on **Render** with live WebSocket endpoints.

---

8. Database Layer

- **Database Type:** Non-relational (MongoDB) or Relational (PostgreSQL)
- **Entities:**
    - **User:** { id, username, email, passwordHash, createdAt }
    - **Message:** { id, senderId, roomId, content, timestamp }
    - **Room:** { id, name, participants }
- **Hosting:** MongoDB Atlas / Aiven / ElephantSQL

---

9. Frontend Layer

- Built with **React + Vite + TypeScript** for optimized performance.
- **React Router** for navigation (Login → Signup → Chat Dashboard).
- **WebSocket integration** for real-time chat.
- **Dynamic fetching** of messages and user lists using Axios.
- Hosted on **Vercel** for production deployment.