**AlumniConnect Real-Time Chat Feature - Full System Plan (MERN + Firebase Auth)**

**🏠 Overview**

This document outlines the full architecture and development plan for implementing a scalable, real-time chat feature in the AlumniConnect web app. It uses the MERN stack (MongoDB, Express, React, Node.js) with Firebase Authentication (already in use) and MongoDB Atlas (cloud-hosted) for message storage.

**💡 Goals**

* Enable real-time 1-on-1 messaging between connected users
* Allow 24/7 usage of the chat system (hosted DB)
* Display message history (full or time-based)
* Use Firebase Auth for login, MongoDB for data
* Ensure system scalability to 20-30+ users

**📊 Tech Stack**

|  |  |  |
| --- | --- | --- |
| **Component** | **Tech** | **Reason** |
| Authentication | Firebase Auth | Already used; secure and simple |
| Realtime Messaging | Socket.IO | Lightweight WebSocket framework |
| Backend | Node.js + Express | Fast API + Socket server |
| Database | MongoDB Atlas (Cloud) | 24/7 accessible, scalable, no local limits |
| Frontend | React.js | Modern, SPA-ready web UI |
| Media Storage (opt) | Cloudinary | Image upload for messages (optional) |
| Hosting | Vercel (FE), Render/EC2 | Live deployment for users |

**📂 Database Models (MongoDB)**

**1. User**

{

firebaseUid: String, // from Firebase

name: String,

email: String,

profileImage: String,

connections: [firebaseUid]

}

**2. Conversation**

{

\_id: ObjectId,

members: [firebaseUid, firebaseUid],

lastMessage: {

text: String,

sender: String,

createdAt: Date

},

createdAt: Date

}

**3. Message**

{

\_id: ObjectId,

conversationId: ObjectId,

sender: String, // firebaseUid

text: String,

imageUrl?: String,

isDeleted?: Boolean,

createdAt: Date

}

**⚖️ Auth & Security**

* Firebase UID will be the main user identity
* On login, pass Firebase ID Token to backend via Socket.IO auth
* Verify token on backend using Firebase Admin SDK
* Prevent unauthorized access to messages/conversations

**✨ Features**

|  |  |  |
| --- | --- | --- |
| Feature | Status | Implementation Plan |
| Message Button on Profile | ✅ | Visible after connection between users |
| Real-time chat | ✅ | Use Socket.IO for bidirectional messaging |
| View conversations list | ✅ | List all chats where user is a member |
| Chat history (10+ days) | ✅ | Filter messages by createdAt timestamp |
| Pagination of messages | ✅ | Limit + sort on client scroll |
| Message seen/read status | ✅ | Add isSeen and seenBy tracking |
| Delete message | ✅ | Flag as isDeleted without removing |
| Typing indicator | ✅ | Emit typing event via socket |
| Image message (optional) | ✅ | Upload to Cloudinary, store URL in message |

**🔄 Routing Structure**

**Frontend**

* /chats → All conversations for logged-in user
* /chats/[uid] → Chat window with specific user

**Backend (Express API)**

* POST /api/conversations → Create/fetch 1:1 chat
* GET /api/conversations/:uid → Fetch all conversations for user
* POST /api/messages → Send a message
* GET /api/messages/:conversationId → Fetch messages (with pagination)

**🔄 Socket.IO Events**

* join-conversation → Join chat room
* send-message → Send + broadcast to recipient
* receive-message → Push to client UI
* typing → Show typing status
* seen-message → Mark message as read

**🚀 Hosting & Deployment Plan**

|  |  |  |
| --- | --- | --- |
| Component | Service | Notes |
| Frontend | Vercel | Deploy React with dynamic routes |
| Backend | Render/EC2 | Persistent Node.js Socket.IO app |
| MongoDB Atlas | MongoDB Cloud | 24/7 live database |
| Firebase Auth | Firebase | Already integrated |
| Cloudinary | (optional) | For image uploads |

**⚡ Storage and Performance**

* Free MongoDB Atlas (M0) offers 512MB → Enough for ~500k text messages
* Scale to M2 (~$9/mo) when needed
* Indexes on conversationId, createdAt, and sender fields
* Use cursor-based pagination for performance

**📅 Next Steps Checklist**

This document should be referenced during all chat-related implementation. If features are added, removed, or changed, update this doc accordingly.