Chat Management App - Documentation

Project Overview

The Chat Management App is a multi-user communication platform designed for HR teams to manage job proposal interactions across multiple platforms like Upwork. The system provides a unified chat dashboard (UniBox), allowing HRs to:

* Monitor all conversations across different platforms.
* Filter chats based on job type, follow-up dates, and pending replies.
* Communicate with job applicants in real-time.

Technologies Used

* Frontend: React.js
* Backend: Node.js, Express.js
* Database: PostgreSQL
* Real-time Communication: WebSockets
* Styling: CSS

Project Setup & Installation

1Install & Run the Backend (Node.js + Express)

cd backend

npm install # Install dependencies

npm start # Start the backend server (default port: 5000)

2.Install & Run the Frontend (React.js)

cd frontend

npm install # Install dependencies

npm start # Start the frontend server (default port: 3000)

3.Set Up PostgreSQL Database

1. Open pgAdmin and create a new database named chat\_management.
2. Import the provided database backup using:
   * pgAdmin: Right-click on the database > Restore > Select chat\_management\_backup.sql

4. Environment Variables (.env)

.env file in the backend folder and add:

DB\_USER=postgres

DB\_HOST=localhost

DB\_NAME=chat\_management

DB\_PASSWORD=yourpassword

DB\_PORT=5432

JWT\_SECRET=your\_jwt\_secret\_key

How the Chat System Works

🔹 HR Dashboard (UniBox)

* Displays all chat conversations from multiple platforms.
* Includes filters (Pending Replies, Follow-up Date, Status, Job Type).
* HR can click on a user chat to open the ChatBox and respond.

ChatBox (Real-Time Messaging)

* HR & User can send instant messages via WebSockets.
* Messages are stored in the database (messages table).
* Previous chat history is visible when opening a conversation.
* File attachments can be sent & downloaded.

Database Structure

1.Users Table (users)

Stores registered users (HRs & Candidates).

CREATE TABLE users (

id SERIAL PRIMARY KEY,

username VARCHAR(255) UNIQUE NOT NULL,

password TEXT NOT NULL,

role VARCHAR(50) CHECK (role IN ('HR', 'User')) NOT NULL

);

2. Chats Table (chats)

Stores chat details.

CREATE TABLE chats (

id SERIAL PRIMARY KEY,

account\_username VARCHAR(255) NOT NULL,

job\_post\_id VARCHAR(255),

job\_type VARCHAR(255),

follow\_up\_date DATE,

status VARCHAR(50),

platform VARCHAR(100),

notes TEXT

);

3.Messages Table (messages)

Stores chat messages.

CREATE TABLE messages (

id SERIAL PRIMARY KEY,

chat\_id INT REFERENCES chats(id) ON DELETE CASCADE,

sender VARCHAR(255) NOT NULL,

message TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT NOW()

);

API Endpoints

Authentication Routes

* POST /register - User Registration
* POST /login - User Login

Chat Routes

* GET /chats - Get all chats (with filters)
* GET /chats/:chatId - Get messages of a specific chat
* POST /messages - Send a new message

Testing the Real-Time Chat

How to Test Real-Time Updates

1. Open the app in two browser windows (One for HR, one for User).( <http://localhost:3000/>)
2. For user(<http://localhost:3000/chat/user1>). Didn’t build no ui for user chat due to lack of time.
3. HR & User should see messages appear instantly when sent.
4. Messages should also be stored in the database (messages table).
5. Refresh the page and ensure previous chat history is still visible.