Chat Application Documentation

This documentation provides an overview of the chat application, its architecture, and how its components interact. The application consists of a frontend (React-based) and a backend (Node.js with Express and Socket.IO). It allows users to log in, manage contacts, and exchange real-time messages.

Table of Contents

Features

- User Login: Users log in using their phone numbers.
- Contacts Management: Users can view and add contacts.
- Real-Time Messaging: Messages are sent and received in real-time using Socket.IO.
- Chat History: Chat history is fetched from the backend.
- Message Status: Messages can be marked as seen.

Architecture

The application follows a client-server architecture:

1. Frontend: A React-based single-page application (SPA) that provides the user interface.

2. Backend: A Node.js server using Express for REST APIs and Socket.IO for real-time communication.
3. Data Storage: User and contact data are stored in a users.json file, while messages are stored in memory.

Frontend

The frontend is built using React and styled with Bootstrap. It communicates with the backend via REST APIs and Socket.IO.

Components

1. App.js: The main component that manages the application state and renders the login screen, contacts list, and chat window.
2. Login.js: Handles user login by emitting a login event to the backend.
3. ContactsList.js: Displays the user's contacts and allows selecting a contact to chat with.
4. ChatWindow.js: Displays the chat messages and provides an input field to send messages.
5. NewChatModal.js: A modal for starting a new chat by adding a contact.

Key Files

- App.js: Main application logic.
- ChatWindow.js: Handles chat messages and real-time updates.
- NewChatModal.js: Allows adding new contacts.
- App.css: Styles for the application.

Backend

The backend is built using Node.js, Express, and Socket.IO. It handles user authentication, contact management, and real-time messaging.

Endpoints

1. GET /users: Returns a list of all registered users.
2. GET /contacts?phone={phone}: Returns the contacts of a user.

3. GET /chats/:contact?phone={phone}: Returns the chat history between a user and a contact.
4. POST /messages: Sends a message via REST.
5. POST /messages/:id/seen: Marks a message as seen.
6. POST /chats: Starts a new chat between two users.

Socket.IO Events

1. login: Joins the user to their room and emits their contacts.
2. sendMessage: Sends a message to a contact and emits it to the recipient's room.
3. receiveMessage: Listens for incoming messages.
4. messageSeen: Notifies the sender when a message is marked as seen.

Setup and Running

Prerequisites

- Node.js installed on your system.
- A package manager like npm or yarn.

Steps to Run

1. Backend:
   - Navigate to the backend directory.
   - Install dependencies: npm install.
   - Start the server: npm start.
2. Frontend:
   - Navigate to the frontend directory.
   - Install dependencies: npm install.
   - Start the React app: npm start.
3. Access the Application:
   - Open the browser and navigate to http://localhost:3000.

How It Works

1. **Login:**
    - **The user enters their phone number.**
    - **The frontend emits a login event to the backend.**
    - **The backend verifies the user and returns their contacts.**
2. **Contacts:**
    - **Contacts are displayed in the sidebar.**
    - **Users can add new contacts via the NewChatModal.**
3. **Messaging:**
    - **Messages are sent via Socket.IO (sendMessage event).**
    - **The backend broadcasts the message to the recipient's room.**
    - **Chat history is fetched via the /chats/:contact endpoint.**
4. **Real-Time Updates:**
    - **The backend uses Socket.IO to push real-time updates (e.g., new messages) to connected clients.**