Chat App Documentation

This document covers both the backend (Node.js + Express + Socket.IO) and frontend (React + Bootstrap + Socket.IO Client) of your MERN-style real-time chat demo. It assumes in-memory storage (no database) and phone-number–based “login.”

Table of Contents

Features

- **Login** with phone number
- **Global user registry**: every login registers a user
- **Contacts list** per user
- **Real-time messaging** via WebSockets

- **RESTful endpoints** for contacts, chat history, new chat, send/mark-seen
- **“Start New Chat”** modal with Bootstrap validation
- **WhatsApp-style UI**: sidebar, chat bubbles

Prerequisites

- **Node.js** v14+ and **npm**
- **Create React App** (for frontend)
- Local development on **localhost** (backend: port 5000, frontend: port 3000)

Project Structure

```
root/
├── backend/
│   ├── server.js
│   └── package.json
└── frontend/
    ├── src/
    │   ├── components/
    │   │   ├── Login.js
    │   │   ├── ContactsList.js
    │   │   ├─── ChatWindow.js
    │   │   └─── NewChatModal.js
    │   ├─── context/
    │   │   └─── UserContext.js
    │   ├─── App.js
    │   ├─── App.css
    │   └─── index.js
    └─── package.json
```

Backend

Backend Setup & Run

Initialize & install

```
 cd backend
npm init -y
npm install express socket.io uuid
npm install --save-dev nodemon
```

1.

Scripts in package.json

```
{
 "scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
 }
}
```

2.

3. **Run**

   - Development: npm run dev
   - Production: npm start

In-Memory Data Models

- **users: { [phone]: { phone, contacts: Set<phone> } }**
- **messages: { [id]: { id, from, to, text, timestamp, seen } }**

Helper:

```
function ensureUser(phone) {
  if (!users[phone]) users[phone] = { phone, contacts: new Set() };
  return users[phone];
}
```

REST API Endpoints

Method	**Path**	**Body / Query Params**	**Description**
GET	/users	—	List all registered phone numbers
GET	/contacts?phone={}	phone (query)	Retrieve contacts for a user
GET	/chats/{contact}?phone={}	phone (query), :contact (path)	Fetch chat history between two users

POST	/messages	{ from, to, text }	Send a message (also emits via WebSocket)
POST	/messages/{id}/seen	{ phone }	Mark message as seen (recipient only)
POST	/chats	{ phone, contact }	Start a new chat (validates users, returns updated contacts + history)

WebSocket Events

- **Client → Server**
  - login { phone }
  - sendMessage { from, to, text }
- **Server → Client**
  - loginSuccess { contacts: string[] }
  - receiveMessage Message
  - messageSent Message
  - messageSeen { id, by }

- usersUpdated string[] *(optional real-time user list)*

Frontend

Frontend Setup & Run

Create & install

```
cd frontend
npx create-react-app .
npm install axios socket.io-client bootstrap
```

1.

Import Bootstrap in src/index.js

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

2.

Run

```
npm start
```

3.

App Structure & Components

- **UserContext**
  - Provides phone & setPhone, persists to localStorage
- **Login**
  - Single input for phone number → calls onLogin

- **ContactsList**
    - Scrollable sidebar showing contacts
- **ChatWindow**
    - Fetches /chats/:contact, displays bubbles, emits sendMessage
- **NewChatModal**
    - Bootstrap modal
    - Fetches global /users or listens to usersUpdated
    - Calls POST /chats → returns { contacts, chat }

User Context & State

```
// src/context/UserContext.js
export const UserContext = createContext();
function UserProvider({ children }) {
  const [phone, setPhone] = useState(null);
  useEffect(() => { // load
    const p = localStorage.getItem('phone');
    if (p) setPhone(p);
  }, []);
```

```
 useEffect(() => { // persist
  if (phone) localStorage.setItem('phone', phone);
  else localStorage.removeItem('phone');
 }, [phone]);
 return (
  <UserContext.Provider value={{ phone, setPhone }}>
   {children}
  </UserContext.Provider>
 );
}
```

- **App gating**: if phone === null, render <Login>; otherwise render main UI.
- **Pass phone** into all API/socket calls.

Styling

- **App.css** defines flex layout, sidebar, chat bubbles
- **Bootstrap** used for modal, buttons, form controls

Workflow

1. **Login** (any tab/incognito) with phone → registers in backend → emits usersUpdated

2. **Open “New Chat”** → fetch /users, validate, POST /chats → add to contacts → open window

3. **Send Message** → via WebSocket or POST /messages → backend stores & emits → frontend updates

4. **Mark as Seen** → POST /messages/{id}/seen → backend updates & emits messageSeen

---

Extending & Production Notes

- **Persistence**: swap in MongoDB or other DB instead of in-memory users & messages.

- **Authentication**: integrate SMS OTP + JWT instead of trusting phone input.

- **Scaling Socket.IO**: use Redis adapter when running multiple server instances.

- **Security**: sanitize inputs, rate-limit endpoints, enforce HTTPS, CORS.

- **Pagination**: add limit & offset to /chats/:contact for large histories.

- **Group chats**: extend data model to track room memberships.