System Design Document:

Overview

This document describes the design and architecture of a real-time chat application incorporating an AI chatbot powered by the Gemini-1.5 Flash API. The application leverages Socket.io for real-time communication, JWT for authentication, Zustand for state management, and MongoDB for the backend. The frontend is built with React and styled using ShadCN components and Tailwind CSS.

Table of Contents

---

1. Architecture Overview

The application follows a client-server architecture. The frontend communicates with the backend via RESTful API calls for non-real-time features and Socket.io for real-time messaging. MongoDB is used for database storage, while JWT ensures secure user authentication. The AI chatbot is integrated via external API calls to Gemini-1.5 Flash.

---

2. Frontend Design

Tech Stack

- **React: Core UI library for the frontend.**
- **ShadCN: UI component library used for building reusable and consistent components.**
- **Tailwind CSS: CSS utility-first framework for styling.**
- **Zustand: Lightweight state management library.**
- **Axios: Used for API requests and handling interactions with the backend.**

Component Structure

The frontend follows a modular component-based structure:

- **App.js: Root component.**
- **ChatContainer: Handles the chat interface.**
- **Sidebar: Displays contact and group chat lists.**
- **MessageInput: Input for typing messages with emoji support.**
- **FileUpload: Allows file and image uploads using Multer.**
- **Authentication: Login and registration components.**

State Management

- **Zustand is used to manage the global state, such as user authentication state, real-time messages, and contacts.**

API Integration

- **Axios is used for sending HTTP requests to the backend API. This includes authentication requests, fetching contacts, sending messages, and retrieving chat history.**

Responsiveness

- **The UI is designed with Tailwind CSS to ensure responsiveness across different devices (mobile, tablet, desktop).**

3. Backend Design

Tech Stack

- **Node.js: Runtime environment for server-side JavaScript.**
- **Express: Web framework for creating RESTful API routes.**
- **MongoDB: NoSQL database for storing user, message, and channel data.**
- **Mongoose: Object Data Modeling (ODM) library for MongoDB.**
- **Socket.io: For real-time communication between the server and clients.**
- **Multer: Middleware for file handling (e.g., file and image storage).**

Routes and Controllers

1. **Auth Routes (`/api/auth`)**
   - **POST /login: User authentication.**
   - **POST /register: User registration.**
2. **Contacts Routes (`/api/contacts`)**
   - **GET /contacts: Fetch user's contacts.**
   - **POST /add-contact: Add a new contact.**
3. **Messages Routes (`/api/messages`)**
   - **POST /send-message: Send a message.**
   - **GET /messages/**
     : Fetch chat messages.
4. **Channel Routes (`/api/channel`)**
   - **POST /create-channel: Create a group chat.**
   - **GET /channels: Fetch all channels.**

Controllers

Controllers handle the logic for each route. For example, the message controller would manage saving a message to the database and emitting a socket event to notify other users.

Middlewares

1. **JWT Authentication Middleware: Protects certain routes by validating tokens.**
2. **Multer Middleware: Handles file uploads for profile images and chat file sharing.**

Database Design

- User Model: Stores user information (e.g., username, hashed password, profile image URL).
- Message Model: Stores message content, sender, timestamp, and attachments.
- Channel Model: Stores group chat information, including participants.
- Contact Model: Stores user contacts and their relationships.

---

4. Real-Time Communication

Socket.io Integration

Socket.io is used for real-time features such as:

- Real-Time Messaging: Users can send and receive messages instantly.
- Typing Indicators: Shows when a user is typing in the chat.
- Online/Offline Status: Tracks user presence.

The `setupSocket.js` file establishes the connection between the server and clients. The server listens for events like `message`, `joinRoom`, and `disconnect`, and emits events back to the clients.

Sample code:

```
// socket.js

import { Server } from 'socket.io';



const setupSocket = (server) => {

  const io = new Server(server, { cors: { origin: process.env.ORIGIN }
});



  io.on('connection', (socket) => {

    console.log('User connected:', socket.id);



    socket.on('joinRoom', (roomId) => {
      socket.join(roomId);
    });

    socket.on('sendMessage', (message) => {
      io.to(message.roomId).emit('receiveMessage', message);
    });

    socket.on('disconnect', () => {
      console.log('User disconnected:', socket.id);
    });
  });
};

export default setupSocket;
```

---

5. AI Chatbot Integration

Gemini-1.5 Flash API

The AI chatbot is integrated via API calls to the Gemini-1.5 Flash platform. This API provides intelligent responses in chats, assisting users with various tasks, answering questions, or providing conversational interactions.

The chatbot's functionality is handled in the frontend by sending the user's messages to the Gemini API and receiving responses to display in the chat.

---

6. Authentication and Security

JWT Authentication

- JWT tokens are used for secure user authentication. Upon successful login, a JWT token is issued and stored in the user's cookies.
- The JWT Middleware verifies these tokens on every API request to ensure secure access to protected routes.

Password Hashing

- User passwords are hashed using bcrypt before being stored in the database.

---

7. File Handling and Storage

Multer Middleware

Multer is used for file handling, such as image uploads for user profiles and file sharing in chats. Files are stored locally in a dedicated `uploads` folder:

- /uploads/profiles: Stores profile pictures.
- /uploads/files: Stores shared files.

js

Sample code

```js
// multer setup

import multer from 'multer';

const storage = multer.diskStorage({

  destination: (req, file, cb) => {

    cb(null, 'uploads/files');

  },

  filename: (req, file, cb) => {

    cb(null, `${Date.now()}-${file.originalname}`);
  },
});
const upload = multer({ storage });
```

File Downloading

The backend serves files via static file routes, allowing users to download shared content from the chat directly.

---

8. System Flow

Flow Diagrams

Frontend to Backend Flow

```
    User-->Frontend[React Frontend]
    Frontend-->API[Axios API Calls]
    API-->Backend[Node.js & Express Backend]
    Backend-->MongoDB[MongoDB Database]
    Backend-->SocketIO[Socket.io]
    SocketIO-->Frontend
```

Authentication Flow

```
    User-->LoginRequest[POST /login]
    LoginRequest-->AuthController
```

```
AuthController-->JWT[JWT Token Creation]

JWT-->Client[JWT Token Stored in Cookie]

Client-->ProtectedRoute[Authenticated API Calls]
```

Real-Time Messaging Flow

```
User1-->Socket[Socket.io]

Socket-->Server

Server-->SocketRoom[Socket Room]

User2-->SocketRoom

Server-->User2[Message Sent]
```

---

9. Conclusion

This chat application integrates real-time messaging, AI chatbot functionality, and file-sharing capabilities within a responsive UI. By leveraging technologies like React, Socket.io, MongoDB, and JWT, it offers both scalability and secure communications.