System Design Document:

Overview

LinkUp is a real-time chatting application with integrated AI chatbot support using the Gemini-1.5 Flash API. It is built with modern web technologies like React for the frontend, Node.js with Express for the backend, and Socket.io for real-time messaging. The application includes features like group chats, personal messages, emoji support, file sharing, file downloading, and Docker images for deployment.

Table of Contents

---

1. Architecture Overview

The application follows a client-server architecture where the frontend and backend communicate via RESTful APIs and Socket.io for real-time messaging. The AI chatbot is powered by Google's Gemini-1.5 Flash API, and user data and messages are stored in MongoDB.

Key Technologies:

- **Frontend: React, ShadCN UI, Tailwind CSS, Zustand for state management, Axios for API calls.**
- **Backend: Node.js, Express, JWT for authentication, Socket.io for real-time messaging, MongoDB.**
- **AI Chatbot: Google's Gemini-1.5 Flash API.**

---

2. Frontend Design

Tech Stack

- **React: For building dynamic and modular UI components.**
- **ShadCN: Component library for building consistent UI components.**
- **Tailwind CSS: For styling and responsive design.**
- **Zustand: Lightweight state management for global state (e.g., user authentication, chat state).**
- **Axios: For sending HTTP requests to the backend.**

Component Structure

- **App.js: Root component that manages routing and the main layout.**
- **ChatContainer: Displays the chat interface, integrates messages and emoji support.**
- **Sidebar: Contains the list of contacts and group chats.**
- **MessageInput: Component for sending messages, including emoji support.**
- **FileUpload: Component for file and image sharing via Multer.**
- **Authentication: Login and registration screens for user authentication.**

State Management

- **Zustand is used to manage the global state, including user authentication, active chats, and message flow. This ensures seamless synchronization across different components.**

API Integration

- **Axios handles API requests to the backend, including:**
  - **User authentication.**
  - **Sending and retrieving messages.**
  - **Managing contacts and groups.**

Responsiveness

- **The UI is designed with Tailwind CSS and ShadCN, ensuring it adapts smoothly across mobile, tablet, and desktop screen sizes.**

---

3. Backend Design

Tech Stack

- **Node.js: The runtime environment for server-side code.**
- **Express: A web framework used for defining API routes and handling requests.**
- **MongoDB: The NoSQL database used to store users, messages, and groups.**
- **Socket.io: For real-time, bi-directional communication.**
- **Multer: Middleware for handling file uploads.**

Routes and Controllers

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

Controllers

Controllers handle business logic for different API requests (e.g., processing and storing messages, sending them to other users via Socket.io, handling file uploads).

Middlewares

- **JWT Middleware: Protects routes that require authentication.**
- **Multer Middleware: Handles file uploads for profile images and file sharing.**

Database Design

- **User Model: Stores user details (username, email, hashed password, etc.).**
- **Message Model: Stores message content, sender, timestamp, and file references.**
- **Channel Model: Stores group chat information.**
- **Contact Model: Stores contact relations between users.**

---

4. Real-Time Communication

Socket.io Integration

- **Socket.io enables real-time messaging, online presence tracking, and typing indicators. Each user is connected to specific "rooms" for their active chats.**

The `setupSocket.js` file manages WebSocket connections and message broadcasting between users.

sample code

```
import { Server } from 'socket.io';


const setupSocket = (server) => {

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


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

    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');
    });
  });
};

export default setupSocket;
```

---

5. AI Chatbot Integration

The chatbot functionality is powered by the Gemini-1.5 Flash API, which allows users to have conversational interactions or receive AI-powered responses to queries.

Steps to Get Google-AI API Key for Gemini-1.5 Flash

To integrate the AI chatbot into your project, you need an API key from Google Cloud. Below are the detailed steps to get the key:

Step 1: Create a Google Cloud Account

- **Sign up at Google Cloud.**
- **Set up your billing information (you get some free credits when signing up).**

Step 2: Create a New Project

1. Go to the Google Cloud Console Dashboard.
2. Click the project dropdown in the top-left corner.
3. Select New Project.
4. Enter a project name, for example, LinkUp AI Chatbot.
5. Click Create.

Step 3: Enable the Gemini-1.5 Flash API

1. In the Google Cloud Console, go to APIs & Services > Library.
2. Search for Gemini-1.5 Flash API.
3. Select the API and click Enable.

Step 4: Create API Credentials

1. Go to APIs & Services > Credentials.
2. Click Create Credentials and select API Key.
3. A new API key will be generated. Copy the key.

Step 5: Restrict the API Key (Optional)

1. In the credentials page, click the Edit (pencil) icon next to the API key.
2. Under Application Restrictions, you can limit the key to certain IPs, HTTP referrers (websites), or apps.
3. Under API Restrictions, select Gemini-1.5 Flash API.
4. Save the changes.

Step 6: Add the API Key to Frontend

In your frontend's `.env` file, add the API key:

```
VITE_API_KEY="Your-Google-AI-API-KEY"
```

Step 7: Test Your Chatbot Integration

Once the API key is added, restart your frontend server:

```
cd ./client
```

```
npm run dev
```

The chatbot should now respond to user queries!

---

6. Authentication and Security

JWT Authentication

- **After login, a JWT token is issued and stored in cookies for subsequent authenticated requests.**
- **JWT middleware ensures that protected routes can only be accessed by authenticated users.**

Password Hashing

- **User passwords are hashed using bcrypt before they are stored in the MongoDB database, ensuring user credentials are secure.**

---

7. File Handling and Storage

Multer Middleware

- **Multer is used to handle file uploads (e.g., images, documents) from users. Uploaded files are stored in designated directories (`uploads/files` or `uploads/profiles`).**

Sample code

```
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

- **Files shared in the chat can be downloaded directly from a URL. The backend serves static files from the `/uploads` directory using `express.static()`.**

---

8. Installation Guide

Step 1: Clone the Repository

```
git clone https://github.com/Pranavsai0407/LinkUp.git
```

```
cd LinkUp
```

Step 2: Install Backend Dependencies

```
cd ./server
```

```
npm install
```

Step 3: Install Frontend Dependencies

```
cd ./client
```

```
npm install
```

Step 4: Set Up Environment Variables

Backend `.env`:

```
PORT=5000

JWT_KEY="your-jwt-key"

ORIGIN="http://localhost:5173"

DATABSE_URL="your-mongodDb-url"
```

Frontend `.env`:

```
VITE_SERVER_URL="http://localhost:5000"

VITE_SOCKET_URL="http://localhost:3001"

VITE_API_KEY="Your-Google-AI-API-KEY"
```

Step 5: Start Backend Server

```
cd ./server

nodemon index.js
```

Step 6: Start Frontend Server

```
cd ./client

npm run dev
```

---

8. System Flow

Overall System Architecture

```
Frontend[React Frontend]-->Backend[Express Backend]

Backend-->Database[MongoDB]

Backend-->SocketIO[Socket.io Server]

Frontend-->SocketIO

AIChatbot-->GeminiAPI[Google Gemini-1.5 Flash API]

Frontend-->AIChatbot[Chatbot Requests]
```

Backend Routes Flow

```
API[Express Routes]-->AuthController[Auth Controller]

API[Express Routes]-->MessageController[Message Controller]

API[Express Routes]-->ContactController[Contact Controller]

API[Express Routes]-->FileUpload[Multer for File Upload]

MessageController-->MongoDB

ContactController-->MongoDB
```

---

10. Conclusion

The LinkUp application provides a powerful, scalable chat platform with real-time messaging, file sharing, AI chatbot integration, and more. With a robust architecture and clean separation of concerns between the frontend and backend, the application is maintainable and easily extendable for future features.