Documentation for the Messaging and Video Calling App – Hangout

1. App Overview

Hangout is a messaging and video calling web application designed using Next.js. It integrates robust third-party services like Stream SDK for chat, Yoom (Stream) for video calling, Clerk for authentication, and Gemini for AI chatbot functionalities. The app provides real-time chat, video calls, group meetings, and various interactive features such as typing indicators, reactions, and file uploads.

2. System Architecture Overview

2.1 Frontend

- **Framework**: Next.js is used to build the frontend, leveraging its SSR (Server-Side Rendering) capabilities for fast performance.
- **UI Components**: The UI design uses Shadcn UI components for a consistent and professional look.
- **Stream SDK Components**: Pre-built Stream SDK components are utilised for chat and video calling interfaces, providing efficient and real-time communication features.

2.2 Backend & API

- **Next.js Server Actions**: Server actions handle API requests for communicating with external services, like authentication, chat, and video calling APIs.
- Rest API: REST API is created to use the Gemini API

2.3 Authentication

- **Clerk**: The Clerk authentication service is integrated to manage user registration and login. After successful authentication, the Clerk sends the user's username to Stream, which generates a token to create and maintain the user's session for messaging and video calls.

2.4 Chat System (Powered by Stream SDK)

- **Stream**: All messaging data, including chat messages, reactions, typing indicators, and group chats, is handled by Stream SDK. Stream's cloud service stores and manages the messages, ensuring scalability and real-time performance.
- **No Database Required**: As Stream stores all data internally, no external database is needed for chat or video calling data.

Chat Features:

1. **Typing Indicators**: Real-time notifications when someone is typing.
2. **Reply to Specific Message**: Allows users to quote and reply directly to a specific message in a conversation.
3. **Reactions**: Users can react to messages with emojis.

4. **File Uploads:** Supports sharing photos, files, and documents in chat.
5. **Group Chats**: Supports conversations between multiple users in real-time.
6. **Seen & Online Indicators**: Displays if a message is seen and shows the user's online/offline status.

2.5 Video Calling System (Powered by Yoom from Stream SDK)

- **Yoom (Real-time Video Calls)**: Based on Yoom (Stream's video calling service), providing Zoom-like features.

Key Features:

1. **Schedule Meetings:** Users can schedule and join meetings.
2. **Recordings:** Ability to record meetings for future reference.
3. **Screen Sharing**: Users can share their screens during a call.
4. **Change Layout:** Users can toggle between different viewing layouts (e.g., grid, speaker view).
5. **Reactions**: Users can send reactions (e.g., thumbs up, claps) during a call.
6. **Chat Integration:** In-call chat for messaging during video calls.

2.6 AI Chatbot (Powered by Gemini)

- **Gemini Integration:** The AI chatbot Gemini is integrated into the chat system, providing automated responses, interactive conversations, and assistance to users within the chat interface.

2.7 Deployment

- **Platform**: The entire app is deployed on Vercel, which uses serverless functions (AWS Lambda) to handle requests. Vercel also ensures continuous deployment and fast performance.

3. System Design Architecture

3.1 Authentication Flow:

- The user initiates login/register via Clerk.
- Upon successful authentication, the Clerk sends the username to Stream SDK.
- Stream generates a unique token for the user to access chat and video call services.

3.2 Messaging System:

- Client (Next.js) interacts with the Stream SDK for all chat-related actions.
- Stream manages and stores:
- Chat rooms, messages, files, and reactions.
- Real-time updates like typing indicators, seen indicators, and online status.
- Gemini (AI chatbot) is embedded in the chat UI to assist users.

3.3 Video Calling System:

- Users initiate and join video calls via Yoom (Stream SDK).

- All video and audio data is streamed via Yoom, which handles features like screen sharing, meeting scheduling, and recordings.
- Stream stores video call metadata like meeting times, participants, and reactions.

3.4 Frontend and Backend Interaction:

- Next.js handles the UI, using server-side API calls to integrate with Clerk and Stream SDK.
- The app's logic resides in the serverless actions, which send requests to the appropriate services like Clerk for authentication and Stream for messaging or video calls.

4. Features Breakdown

4.1 Chat Features:

- **Typing Indicators:** Show when a user is typing.
- **Replying to a Message**: Users can select and reply to a specific message.
- **Reactions**: Users can add emoji reactions to messages.
- **File Upload**: Users can upload photos, videos, and other documents.
- **Group Chats**: Supports multiple users in a single conversation.
- **Seen and Online Indicators**: Shows if a user has seen the message or is online.

4.2 Video Calling Features:

- **Schedule Meetings**: Users can create scheduled meetings.
- **Recordings**: Meetings can be recorded for later viewing.
- **Screen Sharing**: Participants can share their screens.
- **Layout Change**: Users can toggle between various view modes (e.g., gallery view, speaker view).
- **In-Call Chat & Reactions**: Chat and reactions within the call.

5. Tech Stack

- **Frontend**: Next.js (with Server Actions for API calls)
- **UI**: Shadcn UI components
- **Authentication**: Clerk
- **Chat & Video**: Stream SDK (Chat and Yoom for video calling)
- **AI Chatbot**: Gemini
- **Deployment**: Vercel (AWS Lambda)

6. Setting Up and Running the App

6.1 Dependencies

Ensure the following dependencies are installed:

- **Next.js**
- **Stream SDK for chat and video calling**

- **Clerk for authentication**
- **Shadcn UI for frontend components**
- **Gemini for AI chatbot integration**

6.2 Running the App

1. **Clone the repository from GitHub.**
Chat app (hangout) : (https://github.com/Ishuboi07/chat-app.git)
Video calling app (yoom): (https://github.com/Ishuboi07/zoom-clone.git)

2. **Install dependencies using:**

```
npm install
```

3. **Set up environment variables for Clerk, Gemini and Stream SDK API keys.**

https://clerk.com/
https://getstream.io/
https://ai.google.dev/
you can check out .env.example file for reference

4. **Clone the video calling app repository**:

```
git clone https://github.com/Ishuboi07/zoom-clone.git
```

5. **Navigate to the project directory**:

```
cd zoom-clone
```

6. **Install the dependencies**:

```
npm install
```

7. **Set up environment variables**:

Create a `.env.local` file in the root of the project and add the necessary API keys for **Clerk** and **Stream SDK** you can check out .env.example file for reference

8. **Run both the application on local servers:**

```
npm run dev
```

9. **Access the app via the local development server (typically http://localhost:3000).**

Checkout the deployed website here: https://hangout.ishaanagarwal.xyz/

You can log in with
Username: tester

Password: tester@imbesideyou
Check how the website looks with a lot of users. (There is a surprise also 🙂)

7. Future scope of this app:

This project is a learning experience that extends beyond its initial scope. I am committed to continuously developing and enhancing **Hangout** in the following areas:

1. **User Experience (UX) Improvements**:
   - I recognise the current app's UX has room for enhancement. I plan to focus on the form validation process, especially when adding a channel. For example, if a user tries to create a channel without entering a username, the app will provide clear and user-friendly validation prompts.
2. **Seamless Video and Chat Integration**:
   - One of my goals is to create a more cohesive experience by integrating video calling and chat features. This will allow users to join video calls directly from the chat interface without logging in separately to the video calling system, streamlining the overall user experience.
3. **Metamask Authentication & Web3 Integration**:
   - As a Web3 enthusiast, I have already enabled **Metamask** authentication within the app. In the future, I aim to explore how **Hangout** can evolve into a Web3 marketplace, integrating blockchain technologies to provide decentralised services and transactions.
4. **End-to-End Encryption**:
   - Security is a crucial focus for future development. I plan to implement **end-to-end encryption** to ensure that user data, particularly chat messages and video call content, is securely encrypted and protected against unauthorised access.