CMPE 280 Web UI and Design and Development

Final Project

Title: VDOChat (Video Chat Application)

Team Name: J2D

Team Members:

Dhanush Babu (15923882)
Janmejay Bhavsar (015931344)
Jay Patel (015216357)

Description:

VdoChat is a cutting-edge video chat application designed to make it easy for people to connect and communicate with each other. The goal of developing this app is to create a platform that enables users to interact with each other in real-time through video chat, regardless of their location or time zone.

One of the primary motivations behind developing this app is to provide people with a more immersive and engaging communication experience than what traditional text-based chat apps can offer. With VdoChat, users can see and hear each other in real-time, which helps to build stronger and more meaningful connections.

Another motivation behind developing this app is to provide a platform that is easy to use and accessible to everyone. The app has a simple and intuitive interface, which makes it easy for users to navigate and use all of its features. Additionally, VdoChat is designed to work on a wide range of devices, including smartphones, tablets, and desktop computers, so users can connect with each other from virtually anywhere.

Furthermore, a unique feature of this application is the ability to chat with a Chatbot running on OpenAI's cutting edge GPT-3.5-Turbo model. You can communicate with the Chatbot like with any other user on the chat window. This chatbot will provide help and support with the application and virtually anything you may ask it with the help of GPT models.

Overall, VdoChat is designed to be a user-friendly, versatile, and reliable video chat app that brings people closer together, no matter where they are in the world.

Requirement:

To build VdoChat as a web application, the following requirements would be necessary:

· User Authentication: Users need to be able to create accounts, log in and log out of the application securely. Therefore, user authentication and authorization mechanisms such as OAuth, JSON Web Tokens (JWT), or similar solutions should be integrated into the application.

· Video and Audio Streaming: The application must be able to transmit video and audio data between participants in real-time. Therefore, a media server such as Kurento, Janus, or Jitsi should be integrated to handle the streaming.

· Peer-to-Peer Connection: To ensure the privacy and security of the participants, the application should support peer-to-peer connections in addition to the media server. This allows participants to communicate directly with each other, reducing latency and enhancing the quality of the video and audio transmission.

· User Interface: A user-friendly interface that allows users to connect with each other quickly and easily should be developed. The interface should include features such as chat, video and audio controls, and the ability to mute or remove participants.

· Cross-browser Compatibility: The application should be compatible with major web browsers such as Chrome, Firefox, Safari, and Edge, as well as mobile browsers.

· Scalability: The application should be designed to handle many participants simultaneously. Therefore, horizontal scaling should be considered using technologies such as load balancers and containerization.

· Security: Security measures such as data encryption, secure data storage, and protection against unauthorized access should be implemented to ensure the privacy and security of the participants.

· Testing: Rigorous testing should be carried out to ensure that the application works as expected across a wide range of devices and network conditions.

· Deployment: The application should be deployed on a reliable and scalable hosting platform such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure. Deployment automation tools such as Docker and Kubernetes should be used to simplify the process.

By fulfilling these requirements, VdoChat can be developed as a robust, scalable, and user-friendly video chat application for the web.

VOCs:

1. Ease of Use: Customers want the video chat web application to be easy to use and navigate. They want the interface to be intuitive and user-friendly, so they can quickly start and join video calls without any hassle.

2. Video and Audio Quality: Customers expect the video and audio quality to be clear and reliable. They don't want to experience any lag or buffering during the call, and they want the audio to be crystal clear so they can communicate effectively.

3. Security and Privacy: Customers want their video calls to be secure and private. Theywant to be sure that their conversations are not being recorded or monitored without their permission.

4. Customization and Personalization: Customers may want to customize the video chat application to their liking. They may want to choose their own backgrounds or avatars, have the ability to adjust video and audio settings to fit their preferences.

5. Integration with Other Tools: Customers may want the video chat application to integrate with other tools they use, such as calendars, email, or project management software.This can help streamline their workflow and make it easier to schedule and join video calls.

6. Customer Support: Customers expect good customer support from the video chat application provider. They want quick and effective solutions to any issues they may encounter, and they want to be able to easily get in touch with customer support if they have any questions or concerns.

Personas:

● Frank (Business Professional): This user persona may be a busy professional who needs to conduct meetings and presentations online. They want a video chat web application that is reliable, easy to use, and can accommodate multiple participants.

● Ravi (Remote Worker): This user persona may be a remote worker who needs to stay connected with his team and clients. He wants a video chat web application that is secure, user-friendly, and can be used on a variety of devices.

● Ganesh (Student): This user persona may be a student who needs to attend virtual classes and group projects. They want a video chat web application that is simple, easy to use, and has features like screen sharing and virtual whiteboards.

● Lucy (Social User): This user persona may be someone who wants to connect with friends and family online. She wants a video chat web application that is easy to use,has fun features like virtual backgrounds and filters, and is accessible on different devices.

● Richard (Healthcare Professional): This user persona may be a healthcare professional who needs to conduct telehealth appointments with patients. He wants a video chat web application that is secure, HIPAA-compliant, and easy to use for both the healthcare provider and patient.

● Tracy (Teacher): This user persona may be a teacher who needs to conduct online classes and meetings with students. She wants a video chat web application that is easy to use, can accommodate multiple participants, and has features like screen sharing and virtual whiteboards to enhance the learning experience.

Job shadowing:

Job shadowing customers of a web-based video chat application can provide valuable insights that can help improve the user experience and make the application more successful in meeting the needs of its users. Here are some key takeaways:

1. Video and Audio Quality: One of the most important aspects of a video chat application is the quality of the video and audio. Improvements in this area can help users better understand each other and reduce misunderstandings that may arise due to poor quality.

2. Ease of Use: A video chat application should be intuitive and easy to use. Improvements In this area can reduce frustration for users and make it more likely that they will use the application again in the future.

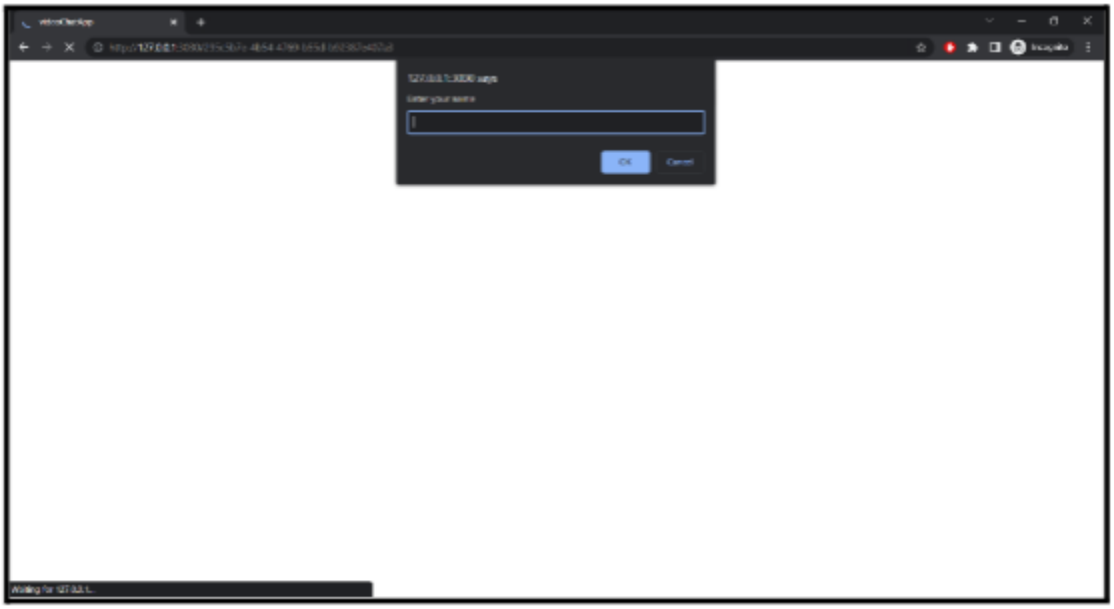
3. Connectivity and Stability: A video chat application should be stable and work well even with low-bandwidth or unstable connections. Improvements in this area can reduce dropouts or other interruptions in communication.

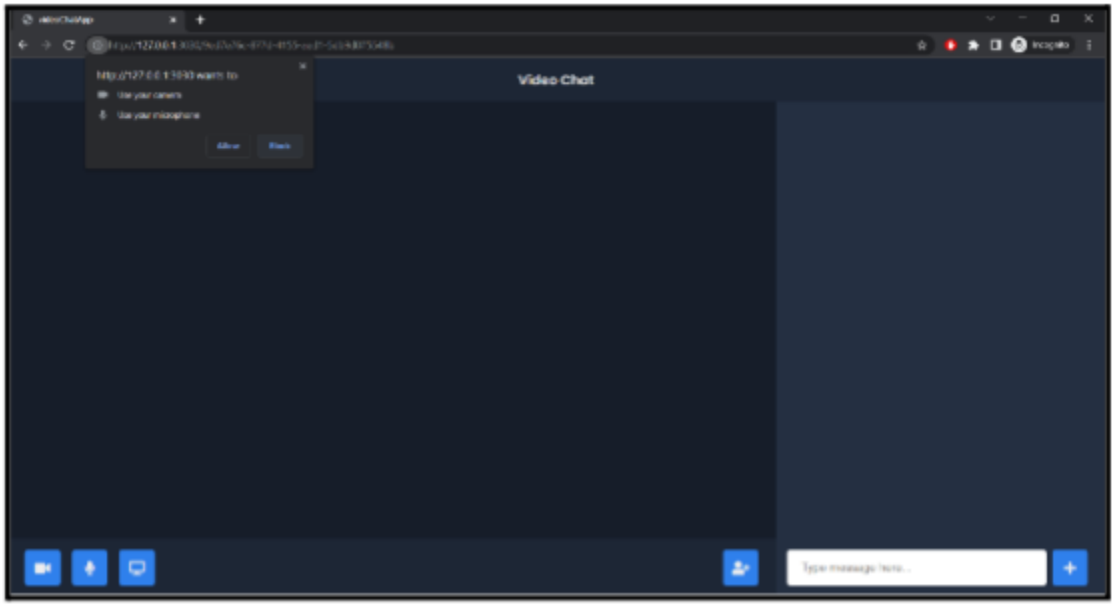
4. Security and Privacy: Video chat applications should prioritize security and privacy features to protect users' personal and sensitive information. Improvements in this area can increase trust in the application and reduce the risk of data breaches or other security-related incidents.

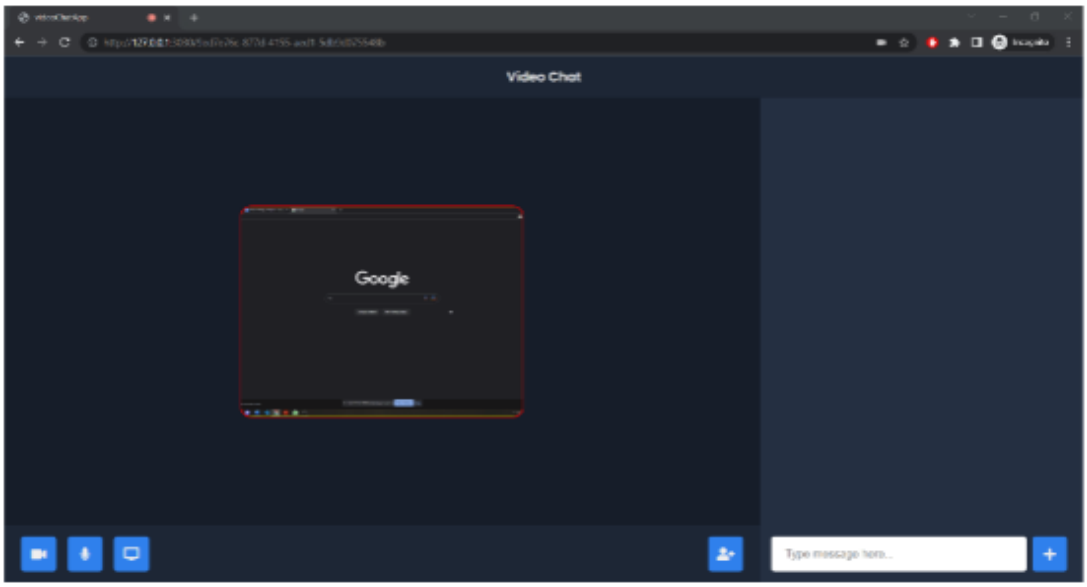
5. Collaboration Features: Many users of video chat applications use them for collaboration, such as virtual meetings or online classes. Improvements in collaboration features, such as the ability to share screens or collaborate on virtual whiteboards, canmake these sessions more effective and productive.

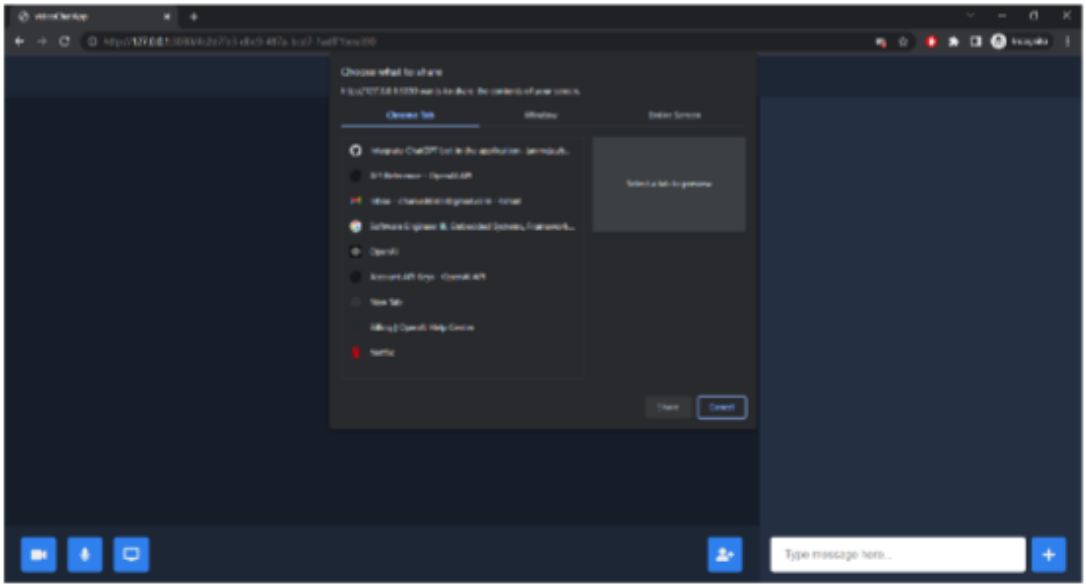
6. Customization and Personalization: Some users may have specific needs or preferences when it comes to using a video chat application. Improvements in customization and personalization, such as the ability to choose different camera or microphone settings or apply virtual backgrounds, can increase user satisfaction.Overall, improvements in these areas can make a video chat application more effective,user-friendly, and enjoyable to use.

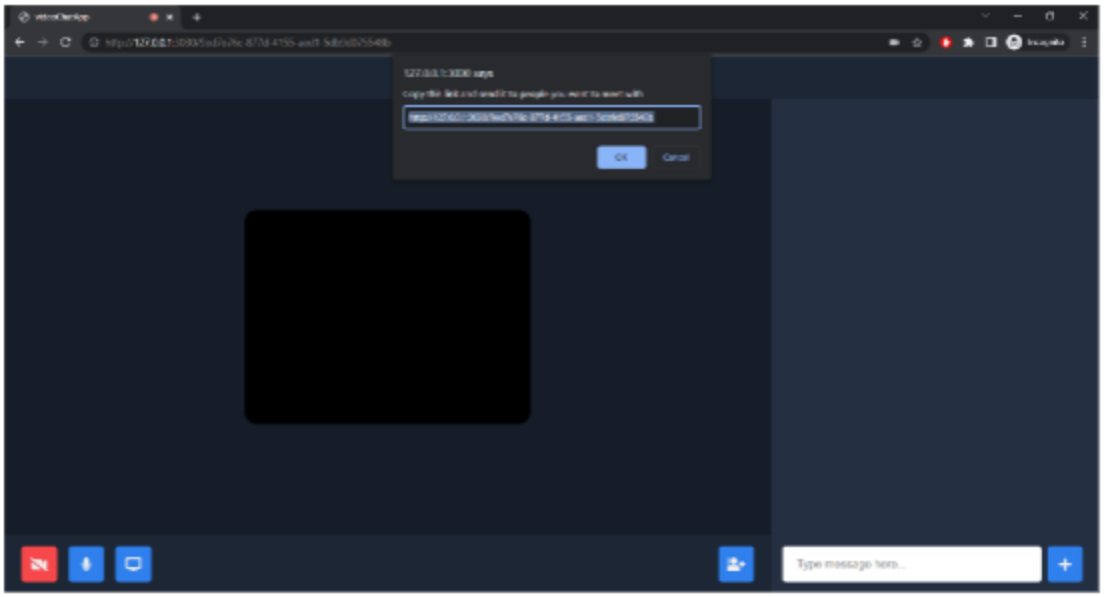
Wireframes:

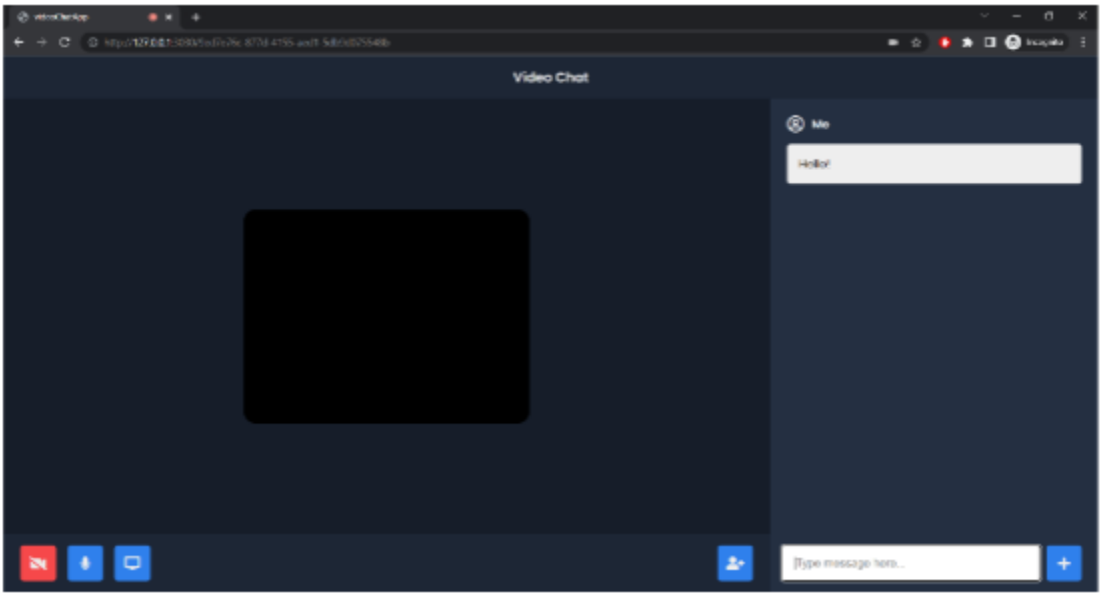




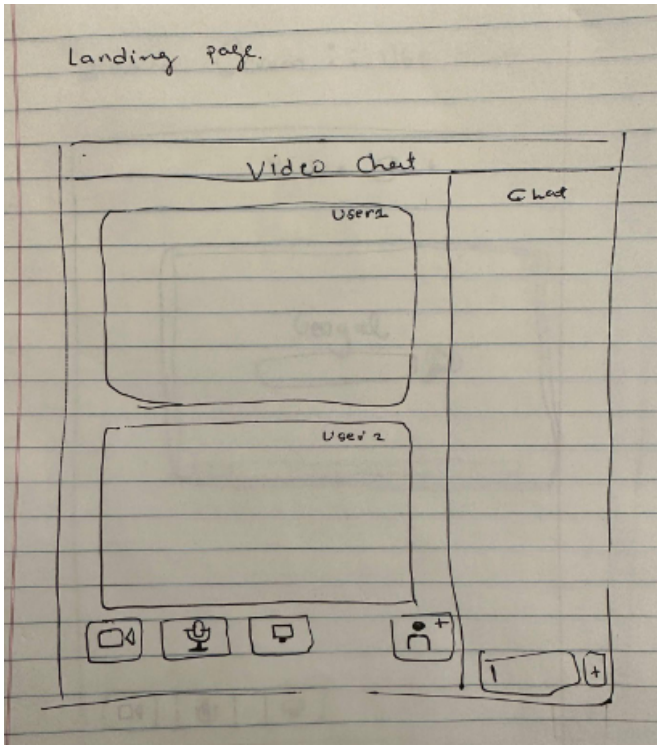

Video Chat
Google
Type message here...

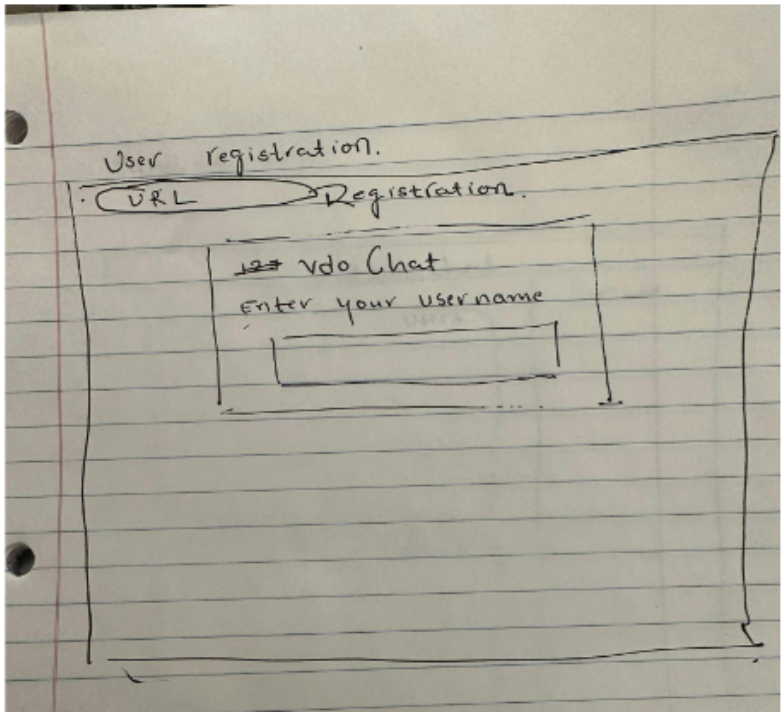

Choose what to share
Type message here...

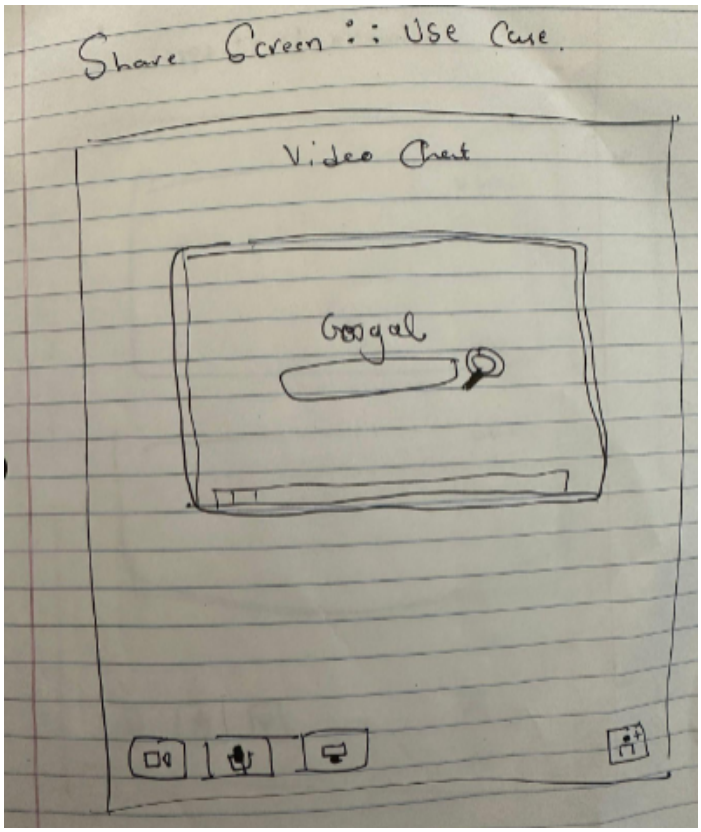

OK
Cancel
Type message here...

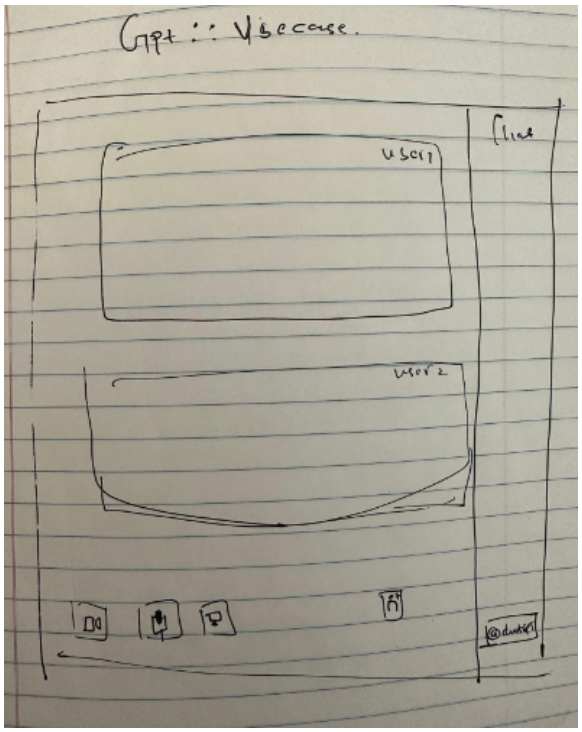

Video Chat
Me
Hello!
Type message here...

Storyboard:

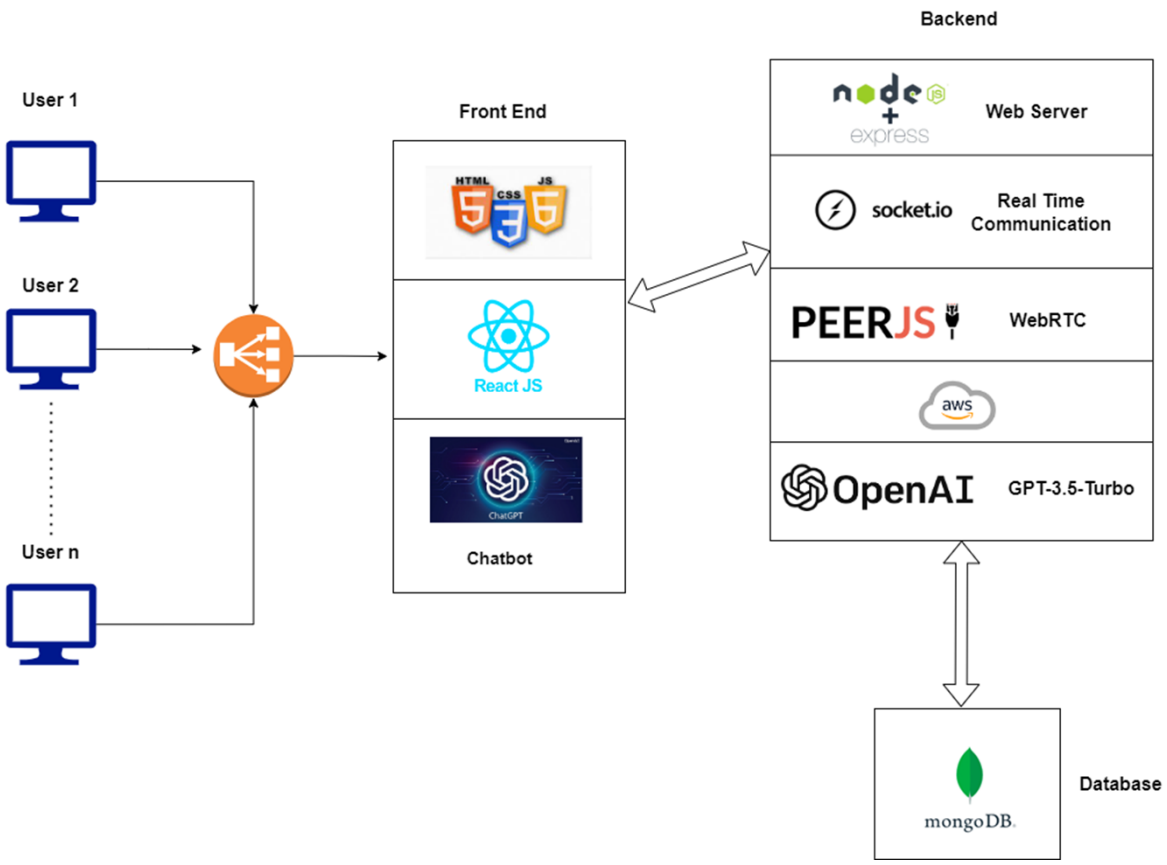

Landing page.
Video Chat
Chat
User1
User 2


User registration.
URL
Registration.
Vdo Chat
Enter your username

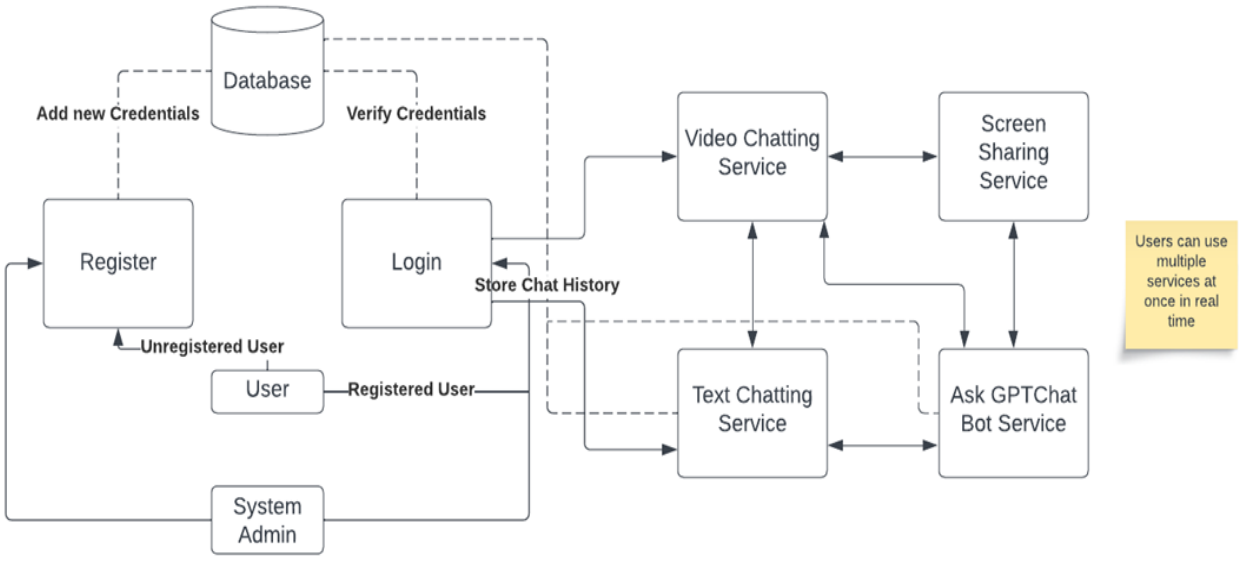

Share Screen :: Use Case.
Video Chat
Google


Gipt :: Usecase.
User1
User 2

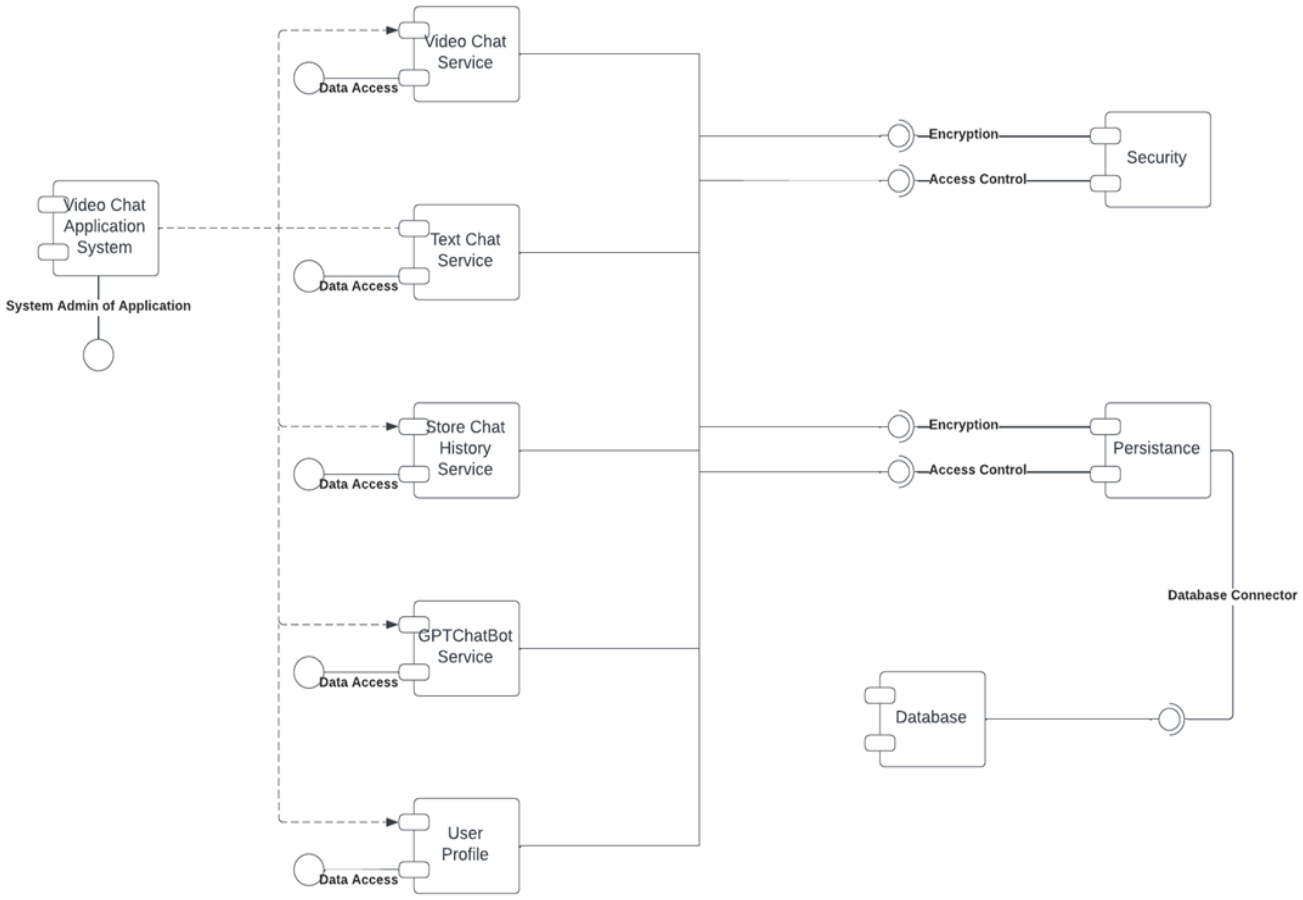
High-Level Architecture Design:



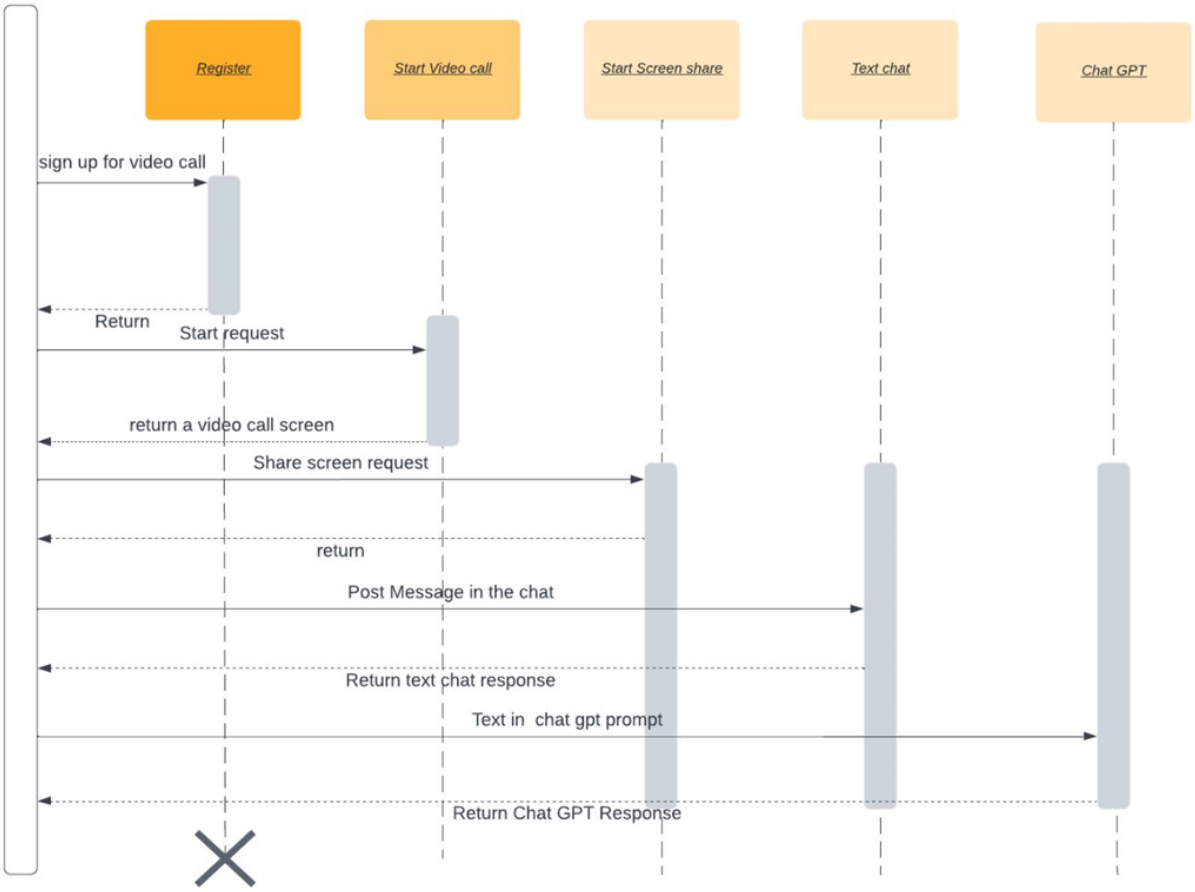
Data Flow Diagram:



Component Level Design:



Sequence or Workflow Diagram:



HTML5 Features Used:

1) Headers and Footers
2) Audio and Video
3) Vector Graphics
4) Nav Tag
5) Placeholder Attribute
6) Email Attribute
7) Storage
8) Ease of Use

Interfaces - RESTFUL and Server-Side Design:

The server side design involves the following components:

- Express.js Setup: The code sets up an Express.js server using the express module. It creates an instance of the server and sets the view engine to EJS (app.set("view engine", "ejs")). It also includes middleware such as body-parser to parse request bodies (app.use(bodyParser.urlencoded({ extended: true }))) and serves static files from the "public" directory (app.use(express.static("public"))).
- Database Connection: The code establishes a connection to a MongoDB database using Mongoose (mongoose.connect(uri)). It uses the provided MongoDB connection URI (uri) and handles the connection success and error callbacks.
- OpenAI GPT-3.5 Chatbot Integration: The code includes the necessary configurations and dependencies to integrate with the OpenAI GPT-3.5 chatbot. It uses the openai instance to send messages to the chatbot and receive responses. The sendMessageToChatbot function takes a message, processes it, and sends it to the chatbot using the OpenAI API.
- Routing and Request Handling: The code defines various routes and request handlers using Express.js. Here are the main routes and their purposes:GET "/" route: Renders the registration page or redirects the user to an existing chat room if they have a valid invite link.POST "/register" route: Handles user registration. It updates or creates a user document in the MongoDB database, generates an invite link, and redirects the user to the chat room with the generated invite link.GET "/:room" route: Renders the chat room page using the specified room ID. The room ID is typically obtained from the invite link.
- Socket.IO Integration: The code integrates Socket.IO to enable real-time communication between the server and clients. It listens for various events such as "connection", "join-room", "disconnect", and "message" to handle user interactions and emit messages or notifications to connected clients. It also includes logic to send messages to the chatbot and broadcast the chatbot's response to all users in a room.
- Peer.js Integration: The code includes Peer.js integration using the /peerjs route to establish peer-to-peer connections between clients for video/audio communication.

- Error Handling and Logging: The code includes error handling for database connections and logs relevant information to the console.
- Server Initialization: The code starts the server by calling server.listen(3030) to listen on port 3030.

Client-Side Design:

The client side design involves the following components:

- HTML: The code assumes the existence of HTML elements that are referenced in the JavaScript code. These elements include a video grid container ( ), buttons for muting audio, stopping video, screen sharing, and ending the call, an input field for entering chat messages, a button for sending chat messages, a container for displaying chat messages ( ), and various other elements for user interface interactions.
- JavaScript: The JavaScript code handles the logic for establishing connections, managing video and audio streams, handling chat messages, and controlling user interface elements. It utilizes libraries such as Socket.IO and Peer.js for real-time communication and WebRTC capabilities.
- CSS: The code assumes the presence of CSS styles to define the layout and appearance of the HTML elements. It references classes such as header__back, main__left, main__right, message, background__red, etc. These classes are used to style the elements and control their visibility, flex properties, colors, etc.

Open AI GPT Models:

In the project we used the OPEN AI api to query the GPT Model-3 to retrieve the answers to any queries the user might type as a message by prepending the message with @ChatGPT.

Testing:

Jest Unit Test(Server side):

To test the server side we used the Jest Unit Testing Framework to individually check each of the components of our server by sending HTTP requests to the Express.js server and make assertions on the responses. We're also using Jest's mocking capabilities to mock the User model and the OpenAIApi class to isolate the tests and control their behavior.

Selenium Automation Tests(Client side):
We cover all the functionalities and interactions with the client-side code by using Selenium WebDriver which provides various methods for interacting with elements, performing actions, and asserting the expected behavior.

Cross Browser Compatibility:
We tested cross browser compatibility using Selenium by replacing the name of the browser to use any browser which we like.

```
driver = await new Builder().forBrowser('<example(chrome,firefox,
safari)>').build();
```

Javascript Libraries:

1) React
2) Bootstrap
3) Modernizr
4) Webpack
5) jQuery
6) Babel

Design Patterns Used:

The **MVC** pattern is a widely adopted architectural pattern for building web applications. It helps in separating the concerns of data storage and retrieval (Model), user interface and presentation logic (View), and request handling and business logic (Controller). Here's how is applied to this application:

Model:

- In the context of Express.js and MongoDB, the Model represents the data layer and handles the interaction with the database.

- It includes components such as data models, schema definitions, and database operations.
- You would define MongoDB schemas and models using a library like Mongoose, which provides an elegant way to interact with MongoDB.

View:

- In Express.js, the View corresponds to the user interface, typically implemented using template engines like EJS, Handlebars, or Pug.
- Views handle the presentation logic, rendering dynamic HTML pages that are sent to the client.
- The views can include forms, chat interfaces, video player components, or any other elements required for the user interface.

Controller:

- The Controller in Express.js acts as the intermediary between the Model and the View.
- It handles incoming requests, retrieves data from the Model, and prepares the appropriate response to be sent back to the client.
- The Controller contains the business logic for processing and manipulating data before it is rendered in the View.
- It defines routes, middleware, and request handlers to handle various HTTP methods and endpoints.
- The MVC pattern allows for a clear separation of concerns, making your codebase more organized, maintainable, and testable. Express.js provides a lightweight framework that is flexible enough to implement the MVC pattern effectively. MongoDB serves as the database for storing and retrieving application data.

Pagination:

Our application is not applicable for this feature as it does not contain lists of data instead we have scroll functionality present. We have the scroll set to auto enable whenever the chat message window starts to fill up. The user can scroll up and down in order to view recent and old messages.
One other alternative which we have is the expand chat window button which provides a better view to the user.

Localization:

The app consists of very accessible button icons from which users can easily understand the functionality. Additionally, We have integrated chat-gpt in the text chat to help users with any information including the translation of the text into their local language.

SEO:

We added meta and title tags in order to enable search engine optimization.

```
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>VDO Chat</title>
```

Profiling:

We used the in browser profiling using the Chrome browser. Below is the screenshot: