# **1.INTRODUCTION**

# PROJECT TITLE: SIMPLIFYING VIDEO CONFERENCING

# TEAM MEMBERS:

1. B.BHAVANA-218X1A0406
2. CH.SUSHMA-218X1A0409
3. K.SIVA SANKARA VARA PRASAD-228X5A0406
4. B.VENKATA ARJUN-218X1A0408

5.K.HARISH-218X1A0461

# 2.PROJECT OVERVIEW:

## PURPOSE:

The purpose of the video conferencing app project is to create a platform that enables users to communicate and collaborate remotely through high-quality video and audio calls. The goal is to provide a user-friendly, reliable, and secure solution for virtual meetings, whether for personal, educational, or business purposes. Key goals include ensuring smooth video streaming, supporting large participant numbers, integrating features like screen sharing and messaging, and maintaining high security and data privacy standards.

## FEATURES:

A feasibility study for developing a video conferencing app would involve several key areas to assess its potential in the market. Below is an outline for such a project:

### 1. Market Assessment:

**Target Audience:** The primary users would be individuals, businesses, and educational institutions looking for effective communication tools. This could include remote workers, corporate teams, schools, and freelancers.

**Growth Trends:** With the rise in remote work, online education, and virtual meetings, the demand for video conferencing solutions has surged. Platforms like Zoom, Microsoft Teams, and Google Meet have seen massive growth, particularly post-pandemic.

**Geographic Reach:** Target global users with localized features, including multiple language support and regional data compliance (such as GDPR for the EU).

### 2. Competitor Analysis:

**Zoom:** Offers a range of features such as breakout rooms, screen sharing, and webinar support. Its ease of use is a major plus.

**Microsoft Teams:** Popular for its integration with Microsoft 365 products, it's used primarily by businesses and enterprises.

**Google Meet:** Integrated with Google's ecosystem (Gmail, Calendar), it’s favored for quick, simple meetings.

**Webex:** Focused more on enterprise-level features, with a focus on security and large-scale events.

**Skype:** Still widely used for smaller, casual meetings but less feature-rich than newer platforms.

### 3. Feature Gaps and Opportunities:

**Customization:** Many existing platforms offer a generic experience. There's room to build a platform that allows users to customize their interface, branding, and meeting settings.

**User Experience (UX):** Focus on a simplified, intuitive interface. Several competitors have complex features that can be overwhelming for new users.

**AI Integration:** The potential to leverage artificial intelligence for meeting transcription, translation, and even participant engagement (e.g., automated meeting summaries).

**Collaboration Tools:** Enhanced collaboration features like whiteboards, file sharing, and real-time note-taking could improve the experience.

**Security:** Offering end-to-end encryption and stricter data privacy controls could differentiate a new platform from existing ones.

**Scalability:** Ensure the app can handle a range of needs from small group chats to large webinars or virtual events.

### 4. Monetization Strategies:

**Subscription** **Models**: Offer free basic services and premium features through tiered subscription plans.

**Pay-per-Use:** A model for occasional users who might only need large-scale meetings or events a few times a year.

**Freemium Model:** Basic features for free with additional premium features for businesses and enterprises.

### 5. Technical Considerations:

**Platform Compatibility:** Ensure the app is cross-platform, supporting iOS, Android, Windows, and macOS.

**Low Latency & High-Quality Video:** Invest in optimizing the app for different bandwidths, ensuring smooth video and audio quality.

**Cloud Integration:** Cloud storage for meeting recordings, file sharing, and easy access to materials post-meeting.

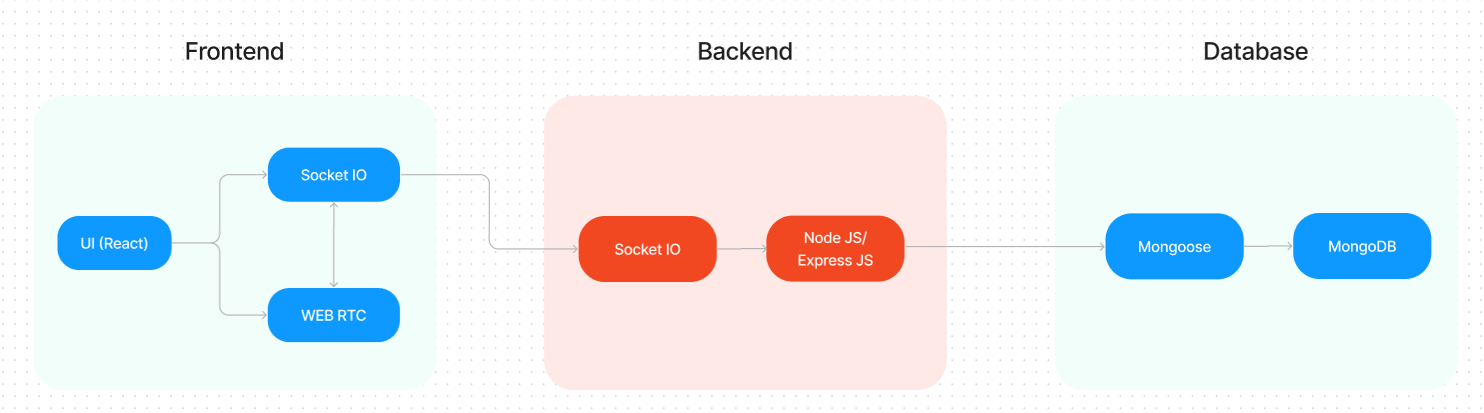
### 6. Challenges and Risks:

**Market Saturation:** The video conferencing space is already highly competitive. Differentiating the new platform will require unique value propositions.

**User Adoption:** Convincing users to switch from well-established platforms may be challenging.

Data Privacy and Compliance: Adhering to strict data protection regulations and maintaining trust will be critical for success.

# 3.ARCHITECTURE:



The technical architecture of our video conference app follows a client-server model, where the frontend serves as the client and the backend acts as the server. The frontend encompasses not only the user interface and presentation but also incorporates the socket.io-client and WebRTC API.

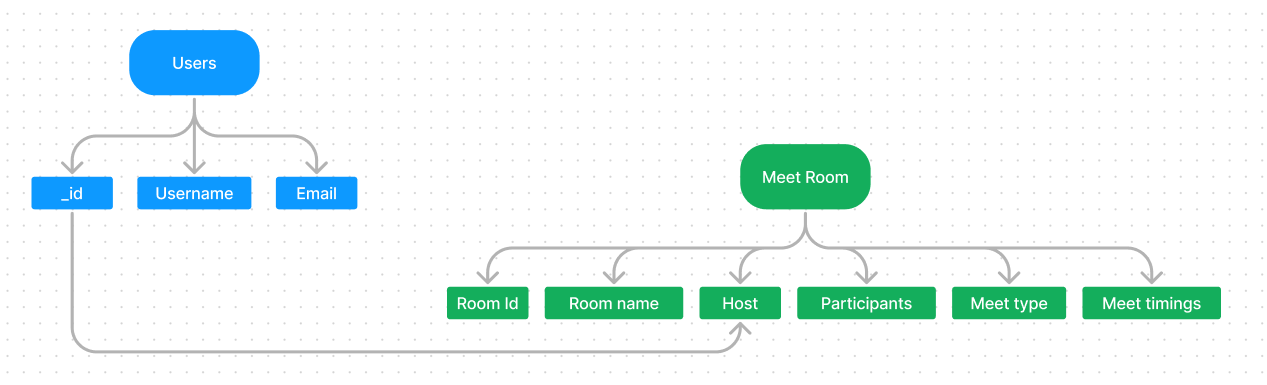
The frontend utilizes the socket.io-client to establish real-time bidirectional communication with the backend server. This enables seamless and instant exchange of audio, video, and chat data between participants during the video conference. Additionally, the WebRTC API plays a crucial role in facilitating peer-to-peer communication, enabling direct audio and video streaming between users without the need for intermediate servers.

On the backend side, we employ socket.io and Express.js frameworks to handle the server-side logic and communication. Socket.io allows for real-time event-based communication between the server and clients, enabling efficient synchronization of video conference data and seamless updates across all participants.

For data storage and retrieval, our backend relies on MongoDB. MongoDB allows for efficient and scalable storage of user data, including user profiles, meeting schedules. It ensures reliable and quick access to the necessary information during video conferences.

Together, the frontend and backend components, along with socket.io, Express.js, WebRTC API, and MongoDB, form a comprehensive technical architecture for our video conference app. This architecture enables real-time communication, efficient data exchange, and seamless integration, ensuring a smooth and immersive video conferencing experience for all users.

**ER Diagram:**



In our video conference app, the ER diagram showcases entities such as users, meetings, and scheduling. It illustrates how these entities relate to each other, helping us understand the underlying database structure and the flow of information within the app.

Furthermore, the ER diagram represents the relationship between meetings and scheduling, enabling users to plan and organize video conferences efficiently. This aspect highlights how meeting details and scheduling information interact within the system.

# SETUP INSTRUCTIONS:

**PRE-REQUISITES**:

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, React.js, Socket.io, Agora RTC, and Agora RTM:

* **Node.js and npm**:

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

* Download: <https://nodejs.org/en/download/>
* Installation instructions: <https://nodejs.org/en/download/package-manager/>
* **Express.js**:

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.

Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

Installation: Open your command prompt or terminal and run the following command:

**npm install express**

* **MongoDB**:

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

* Download: <https://www.mongodb.com/try/download/community>
* Installation instructions: <https://docs.mongodb.com/manual/installation/>
* **React.js**:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

* **Socket.io**:

Socket.io is a real-time bidirectional communication library that enables seamless communication between the server and clients. It allows for real-time data exchange, event-based messaging, and facilitates the development of real-time applications such as chat, collaboration, and gaming platforms.

Install Socket.io, a real-time bidirectional communication library for web applications.

Installation:

* Open your command prompt or terminal of server and run the following command:

**npm install socket.io**

* Open your command prompt or terminal of client and run the following command:

**npm install socket.io-client**

* **Agora RTC**:

Agora RTC (Real-Time Communication): Agora RTC provides a platform for real-time audio and video communication. It offers a range of features like video conferencing, live streaming, and interactive broadcasting, enabling developers to create immersive and interactive communication experiences.

* Sign up for an Agora developer account to access their RTC platform.
* Integration guide and documentation: <https://docs.agora.io/en/>
* **Agora RTM**:

Agora RTM enables real-time messaging and data synchronization between users. It provides reliable and low-latency messaging capabilities, allowing developers to build chat applications, collaborative tools, and real-time notification systems.

* **HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.
* **Database Connectivity**: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations. To Connect the Database with Node JS go through the below provided link:
* • [https://www.section.io/engineering-education/nodejs- mongoosejs-mongodb/](https://www.section.io/engineering-education/nodejs-%20mongoosejs-mongodb/) **Front-end Framework**: Utilize Angular to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.
* **Version Control**: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

• Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

* **Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from <https://code.visualstudio.com/download>

• Sublime Text: Download from <https://www.sublimetext.com/download>

• WebStorm: Download from [https://www.jetbrains.com/webstorm/download](https://www.jetbrains.com/webstorm/download%20)

To run the existing Video Conference App project downloaded from Google drive:

Follow below steps:

* **Download the code:**

• Download the code from the below link:

<https://drive.google.com/drive/folders/1c1SBTyz8HLUh6g_TfPdNrojvMl3KIK0g?usp=sharing>

* **Install Dependencies:**

• Navigate into the cloned repository directory:

**cd smart-meet**

• Install the required dependencies by running the following commands:

**cd client**

**npm install**

**cd ../server**

**npm install**

* **Start the Development Server**:

• To start the development server, execute the following command:

**npm start**

• The video conference app will be accessible at http://localhost:3000

* **Access the App:**

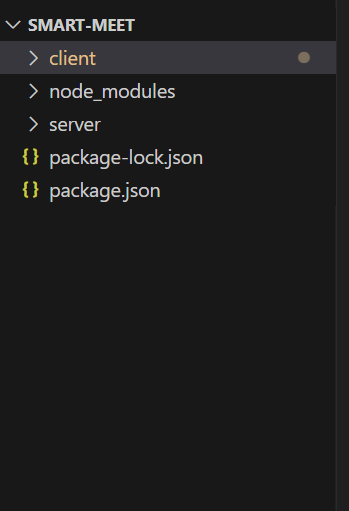
• Open your web browser and navigate to [http://localhost:3000](http://localhost:3000/).

• You should see the video conference app's homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the e-commerce app on your local machine. You can now proceed with further customization, development, and testing as needed.

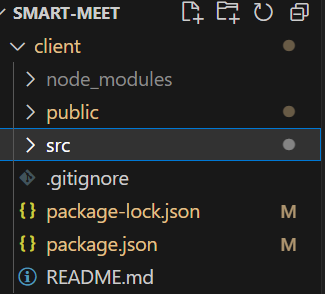
# FOLDER STRUCTURE:

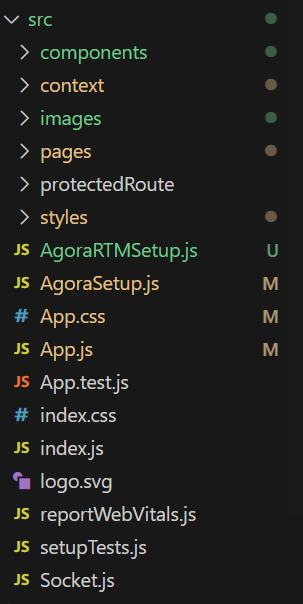
Inside the smart-meet (video conference app) directory, we have the following folders

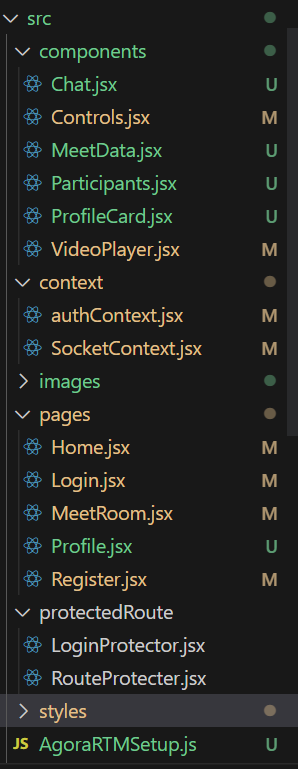


* **Client directory:**

The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api’s such as socket.io and agora.

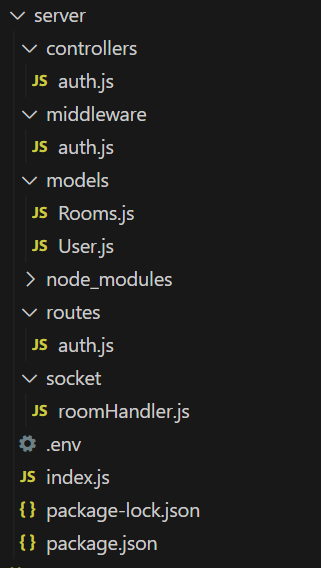






* **Server directory:**

The below directory structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with socket.io Api.



# APPLICATION FLOW:

* **Host:**
* Initiate and manage the video conferences.
* Schedule and send invitations for upcoming meetings.
* Facilitate smooth communication and collaboration among participants.
* Ensure the overall organization and flow of the video conference.
* **Participant:**
* Join video conferences hosted by the host.
* Engage in active communication and contribute to discussions.
* Share screens, if necessary, to present content or collaborate on projects.
* Utilize in-meet chat to ask questions, share information, or provide feedback.
* Respect meeting etiquette and follow guidelines set by the host.
* Actively participate and contribute to a productive and collaborative meeting environment.

**Project Flow:**

**Milestone 1: Project setup and configuration.**

* **Folder setup**:

To start the project from scratch, firstly create frontend and backend folders to install essential libraries and write code.

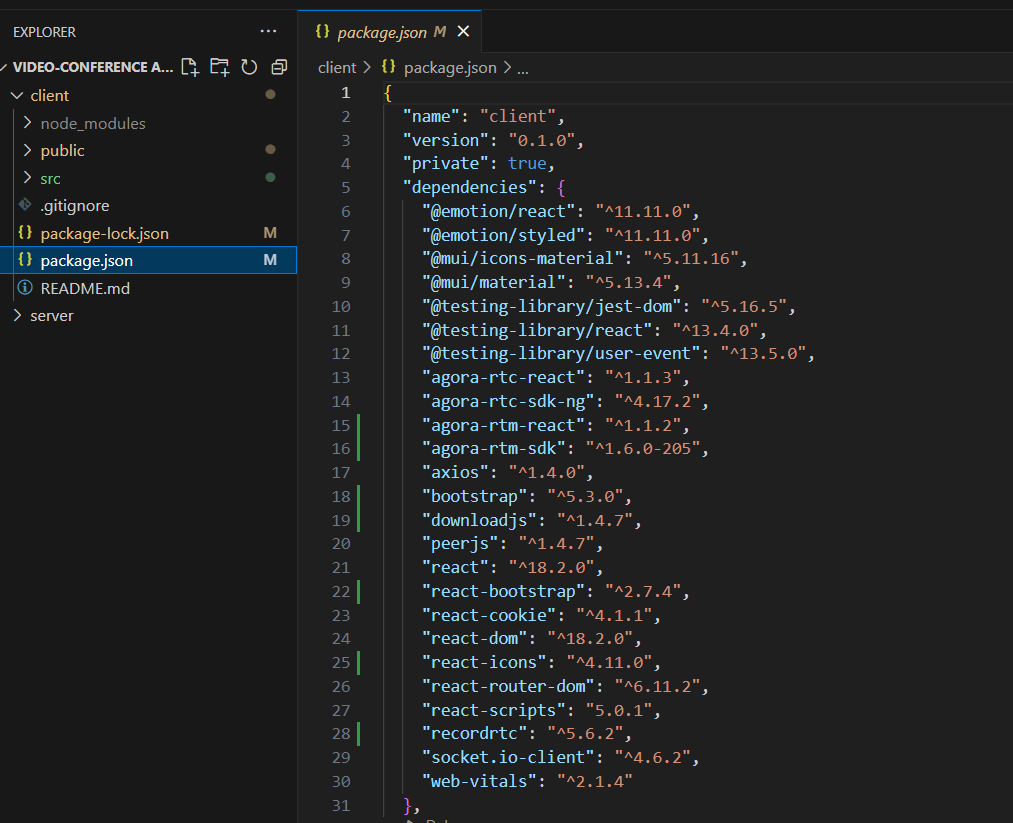
* + - client
    - server
* **Installation of required tools**:

Now, open the frontend folder to install all the necessary tools we use.

For frontend, we use:

* React Js
* Socket.io-client
* Bootstrap
* Material UI
* Axios
* recordrtc
* downloadjs
* agora-rtc-react
* agora-rtc-sdk-ng

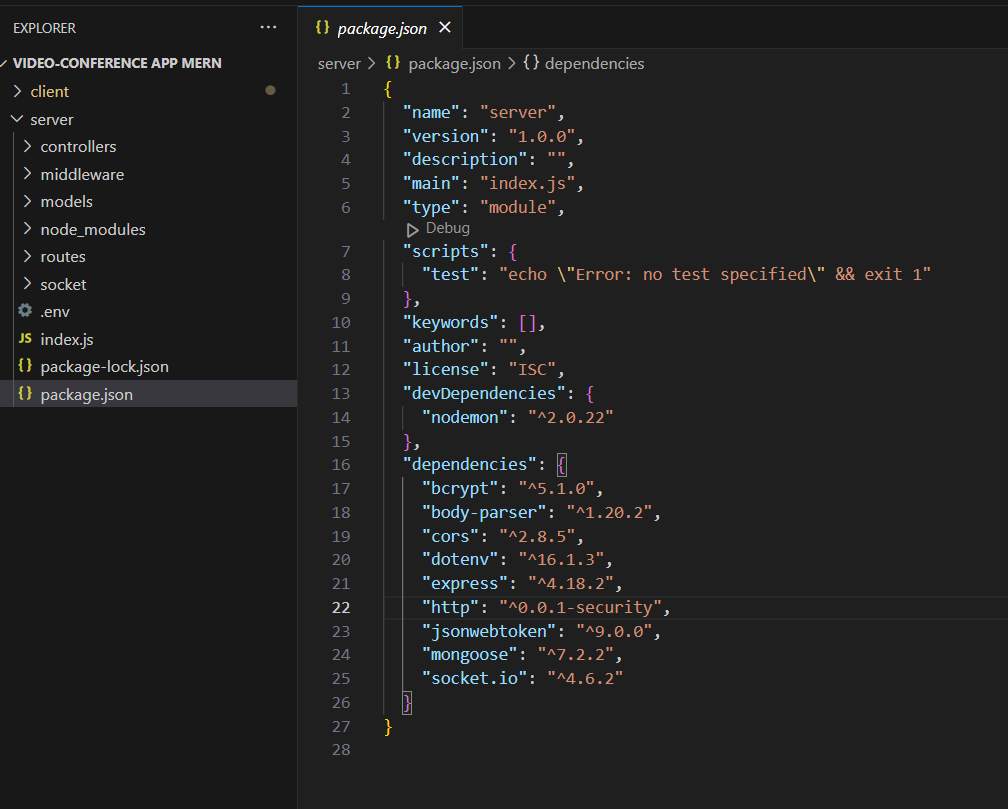
After installing all the required libraries, we’ll be seeing the package.json file similar to the one below.



Now, open the backend folder to install all the necessary tools that we use in the backend.

For backend, we use:

* bcrypt
* body-parser
* cors
* dotenv
* express
* http
* jsonwebtoken
* mongoose
* socket.io

After installing all the required libraries, we’ll be seeing the package.json file similar to the one below.

**Milestone 2: Backend Development**

**Setup express server**

1. Create index.js file in the server (backend folder).
2. Create a .env file and define port number to access it globally.
3. Configure the server by adding cors, body-parser.

**Create socket.io connection**

* 1. Import socket.io in the index.js file.
  2. Create server using the socket.io.
  3. Start the server.

**Configure MongoDB**

* 1. Import mongoose.
  2. Add Database URL to the .env file.
  3. Connect the database to the server.
  4. Create a ‘models’ folder in the server to store all the DB models.

**Add authentication**

* 1. Create the “User” model for the MongoDB.
  2. Create auth controller file to control the authentication actions.
  3. Import “bcrypt” – used to hash(encode) the password to make it secure.
  4. Import “JWT” to create authentication tokens.
  5. Define registration & login activities.
  6. Configure frontend & backend for authentication and store authenticated data in Context API in frontend.

**Create rooms**

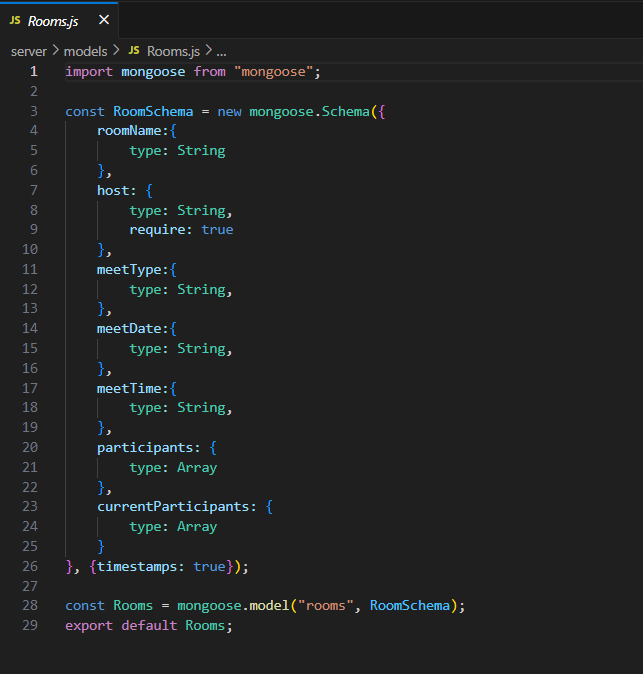
* 1. Create “Rooms” model to store data of rooms (meet rooms) in Database.
  2. Configure socket.io-client to make client-server communication simpler.
  3. Send message to server if client requested to create a room.
  4. Create a new room and add user to it.
  5. Allow other users to join the room.
  6. Manage user leaving the room.
  7. Use the calendar input, if the user wants to schedule meet for later.

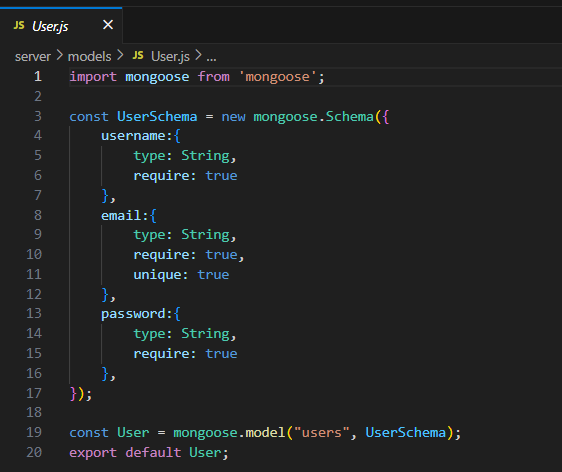
**Milestone 3: Database development**

* **Configure schema**

Firstly, configure the Schemas for MongoDB database, to store the data in such pattern. Use the data from the ER diagrams to create the schemas.

The Schemas for this application look alike to the one provided below.





* **Connect database to backend**

Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.



**Milestone 4: Frontend development**

* **Configure API**
  1. Choose an API for media streaming. (You are free to use any WebRTC based API. But in tutorial, we’ve used Agora API).
  2. Create a setup file in client folder (frontend).
  3. Create a new project and copy the token & app id in the API.
* **Access media streams**
  1. Access the camera and microphone.
  2. Capture the media streams and access them.
  3. Also access the display video for screen sharing functionality.
* **Add meet (conference) in room**
  1. Add the users in room to the meet channel.
  2. Publish the media stream of user to others in the meet.
  3. Display the video of all participants in the meet.
* **Add meet (media) controls**
  1. Create a component to add the controlling features.
  2. Add icons to represent the controls.
  3. Create functionalities to mute audio/video and to leave the meet.
* **Add screen sharing functionality**
  1. Access the screen display media when user tap on screen share control option.
  2. Unpublish the camera track of the user and publish the screen track.
  3. Undo the actions when user closed screen share.
* **Add Recording functionality**
  1. Use any Js/react Js libraries to record the meet screen.
  2. Here, we use” recordrtc” to record the meet and “downloadjs” to download the recorded meet.
  3. Access this functionality using the meet controls.
* **Add in-meet chat facility**
  1. Add the chat feature to seamless communication in the room.
  2. Use any API’s like AgoraRTM or simply Socket.io
  3. In case of socket.io, if any user shared the message (text/file), broadcast the message to all other participants in the meet.
* **Add user profile section** 
  1. Create a file for profile page.
  2. Display the user details in the page and allow user to edit them.
  3. Use a different page or use the profile page to display the past & upcoming meet details.
  4. Allow user to edit the upcoming meet details.
* **Milestone 5: Project Implementation**
* Finally, after finishing coding the projects we run the whole project to test it’s working process and look for bugs. Now, let’s have a final look at the working of our video conference application

# AUTHENTICATION:

**Authentication Process**

Authentication ensures that the user is who they claim to be. Here’s how it works in the project:

**User Login (Authentication)**

* **User credentials** (email and password) are provided when attempting to log in.
* The backend verifies the credentials against the database. If the credentials are valid, the server generates a **JWT token**.

**JWT Token**

* The **JWT** is a stateless token that is signed by the server and includes claims, such as user ID, email, and user roles.
* Once the JWT token is issued, it is sent to the client, which can then use this token for subsequent requests to authenticate the user.

**How the JWT Token works:**

* **Token Format**: The JWT consists of three parts:
  + **Header**: Contains information about how the token is signed.
  + **Payload**: Contains the claims about the user (e.g., user ID, email, role).
  + **Signature**: A signature created using the secret key to verify the token’s authenticity.
* The client stores the JWT token (usually in local storage or a cookie) and sends it in the Authorization header of requests.

**Authorization Process**

Authorization controls what an authenticated user can do. This process ensures that a user has the appropriate permissions to perform an action.

**Role-based Authorization**

Once a user is authenticated, their **role** is used to determine what they are allowed to access or modify. Common roles might include:

* admin: Full access to all resources.
* user: Limited access to personal or public resources.
* moderator: Can moderate content or users.

Roles are typically included as claims in the JWT token. These roles are used to protect specific endpoints and actions in the API.

For example:

* **Admin users** can access /admin routes.
* **Regular users** can only access /user routes.

**Access Control in Endpoints**

The backend checks the user’s **JWT token** to determine their identity and role before allowing access to specific resources. Here’s an example of how role-based access control (RBAC) works in the API:

**Example:**

* **Admin-only routes**: If the request is made to an endpoint requiring the admin role, the server will check the token for the role claim. If the role is admin, the request will be allowed to proceed; otherwise, the server responds with an error (403 Forbidden).
* The server extracts the JWT from the Authorization header and decodes it. It checks if the role claim in the payload is admin.
* If the role is not admin, the server will respond with a **403 Forbidden** status.

**Token Expiry and Refresh:**

**Expiration of JWT Token**

JWT tokens have an expiration date (exp claim) after which they become invalid. When a JWT token expires, the user will need to log in again to obtain a new token.

**Refresh Tokens**

To avoid forcing users to log in frequently, the project may implement a **refresh token** mechanism.

* **Refresh tokens** are long-lived tokens that are used to request new access tokens when the current access token expires.
* The refresh token is securely stored by the client and sent to the server to exchange for a new access token.
* The refresh\_token is typically valid for a longer period (e.g., a few weeks), while the access\_token is short-lived (e.g., 1 hour).

**Sessions and Cookies (Optional)**

For some web applications, authentication can also be handled via **sessions** and **cookies**. In this model:

* **Session tokens** are stored on the server and linked to the user's session ID.
* A **cookie** is sent with each request, allowing the server to identify the user by their session.

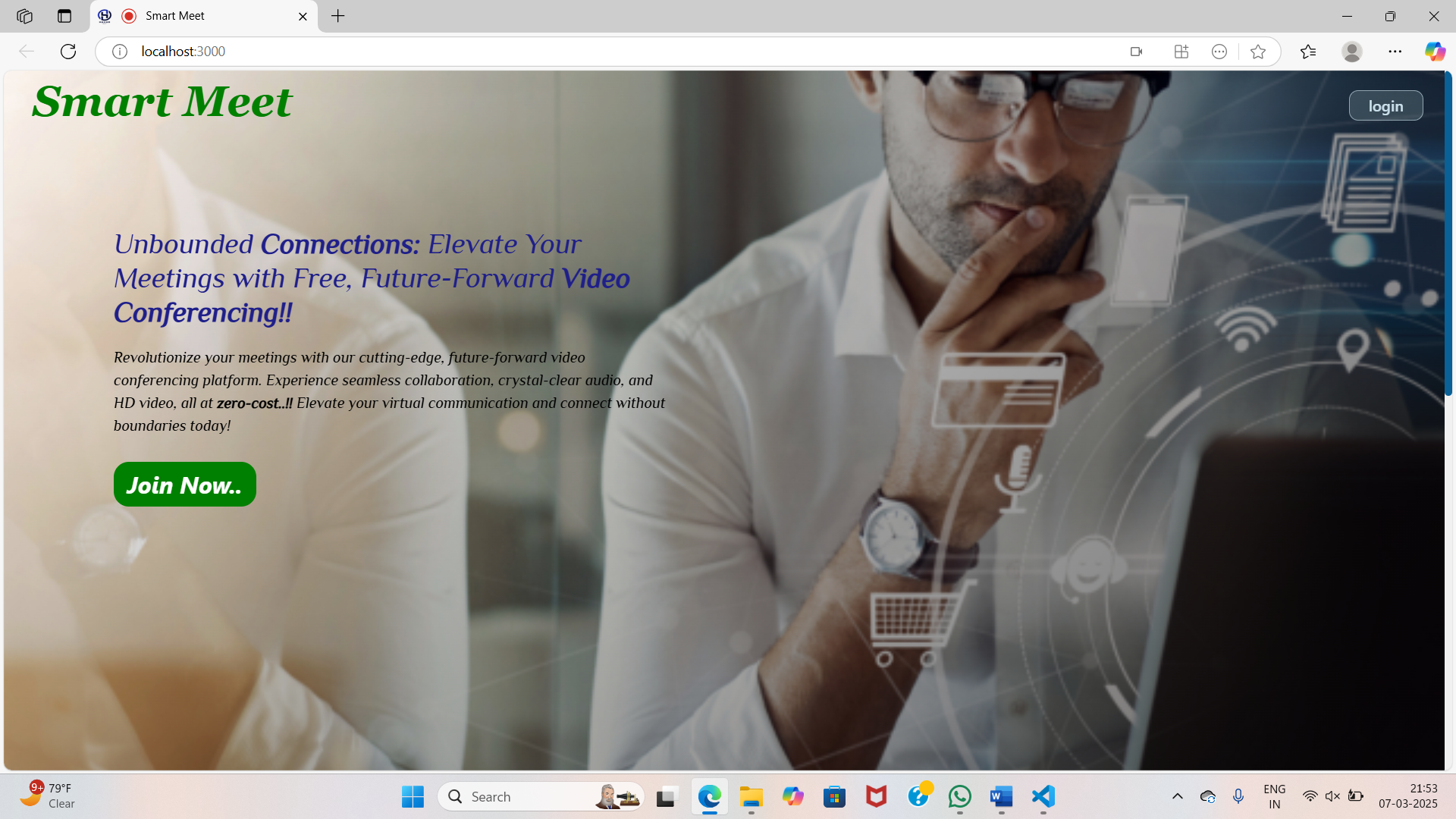
For this method, after successful login, the server generates a session ID, stores it on the server, and sends a cookie with that ID to the client.

**Security Measures**

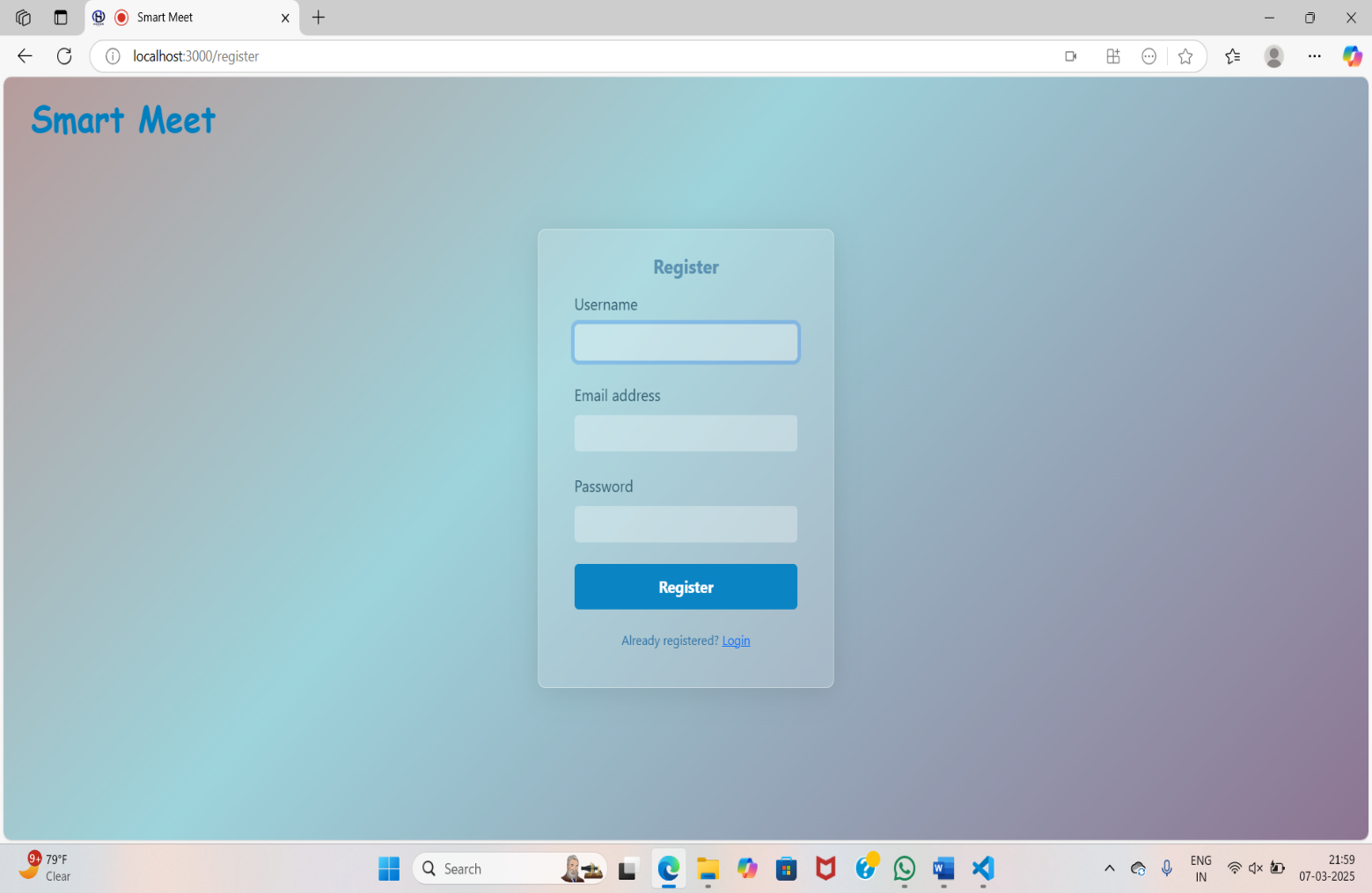
To ensure security in the authentication and authorization processes:

* **JWT Signing**: JWT tokens are signed using a secret key or private/public key pair to prevent tampering.
* **Secure Cookies**: If using session-based authentication, ensure cookies are marked as HttpOnly, Secure, and SameSite=Strict to mitigate risks like cross-site scripting (XSS) and cross-site request forgery (CSRF).
* **Rate Limiting**: To prevent brute force attacks, rate limiting is implemented for authentication endpoints.

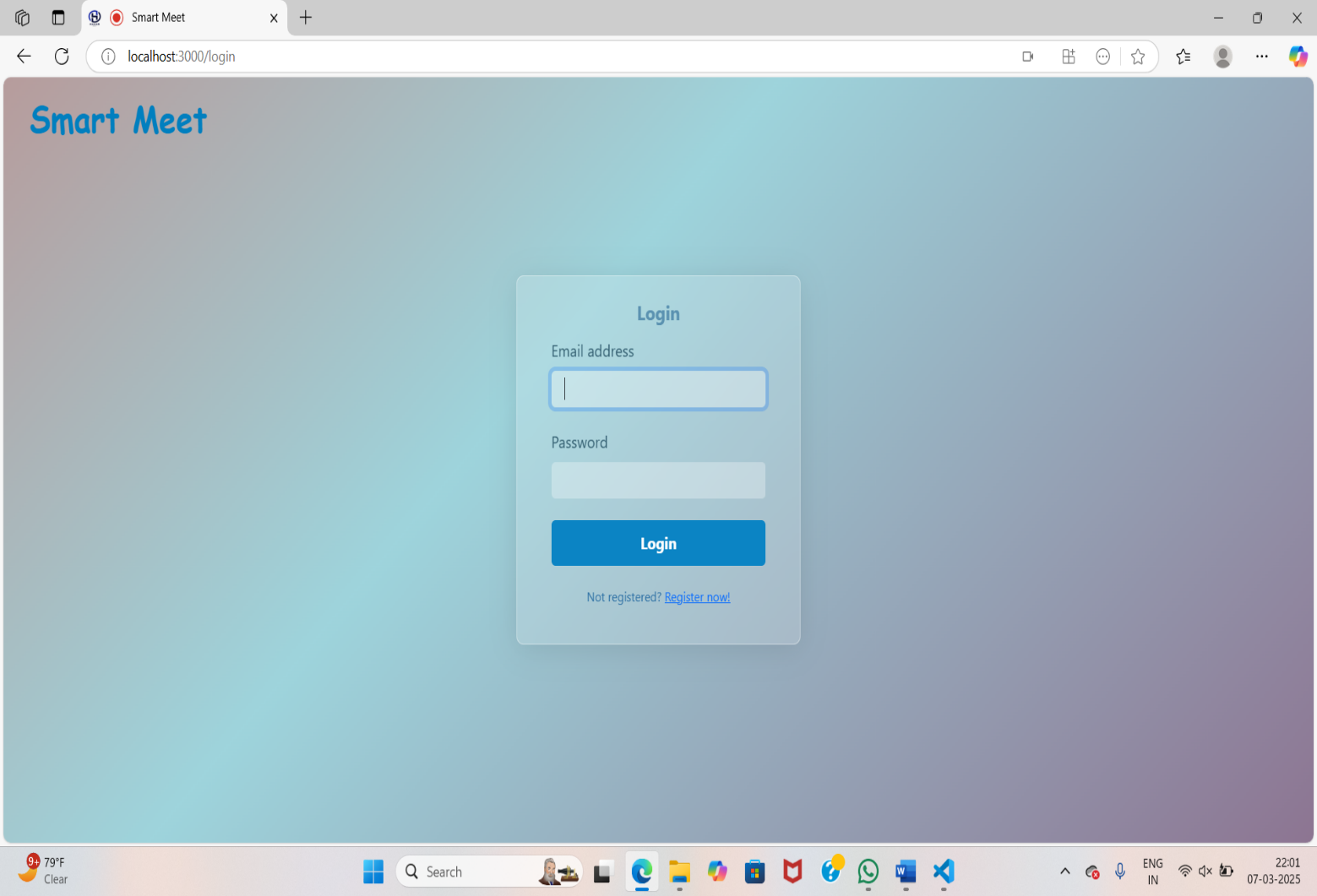
Landing page UI:

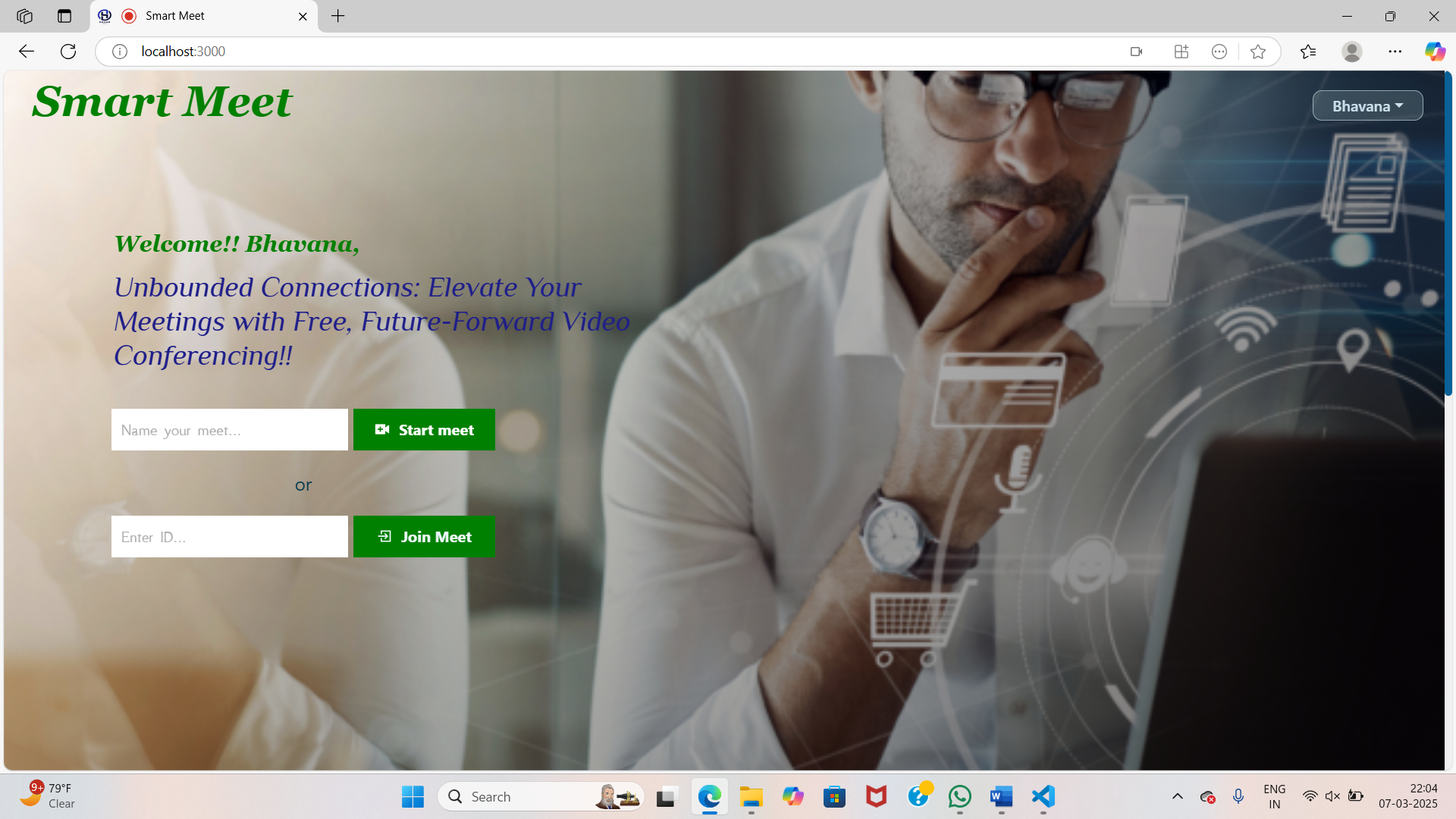


Registration page:

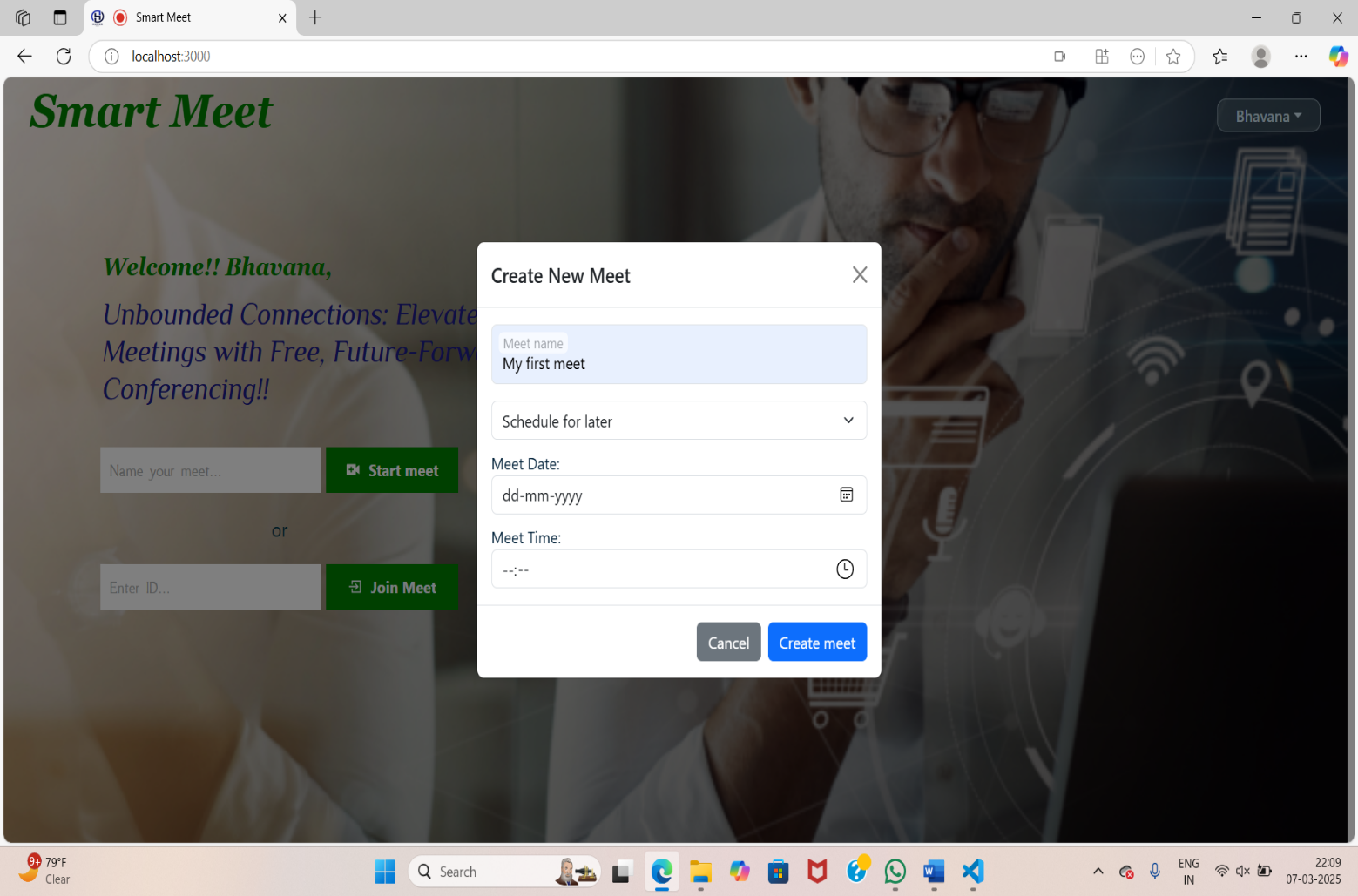


Login Page:

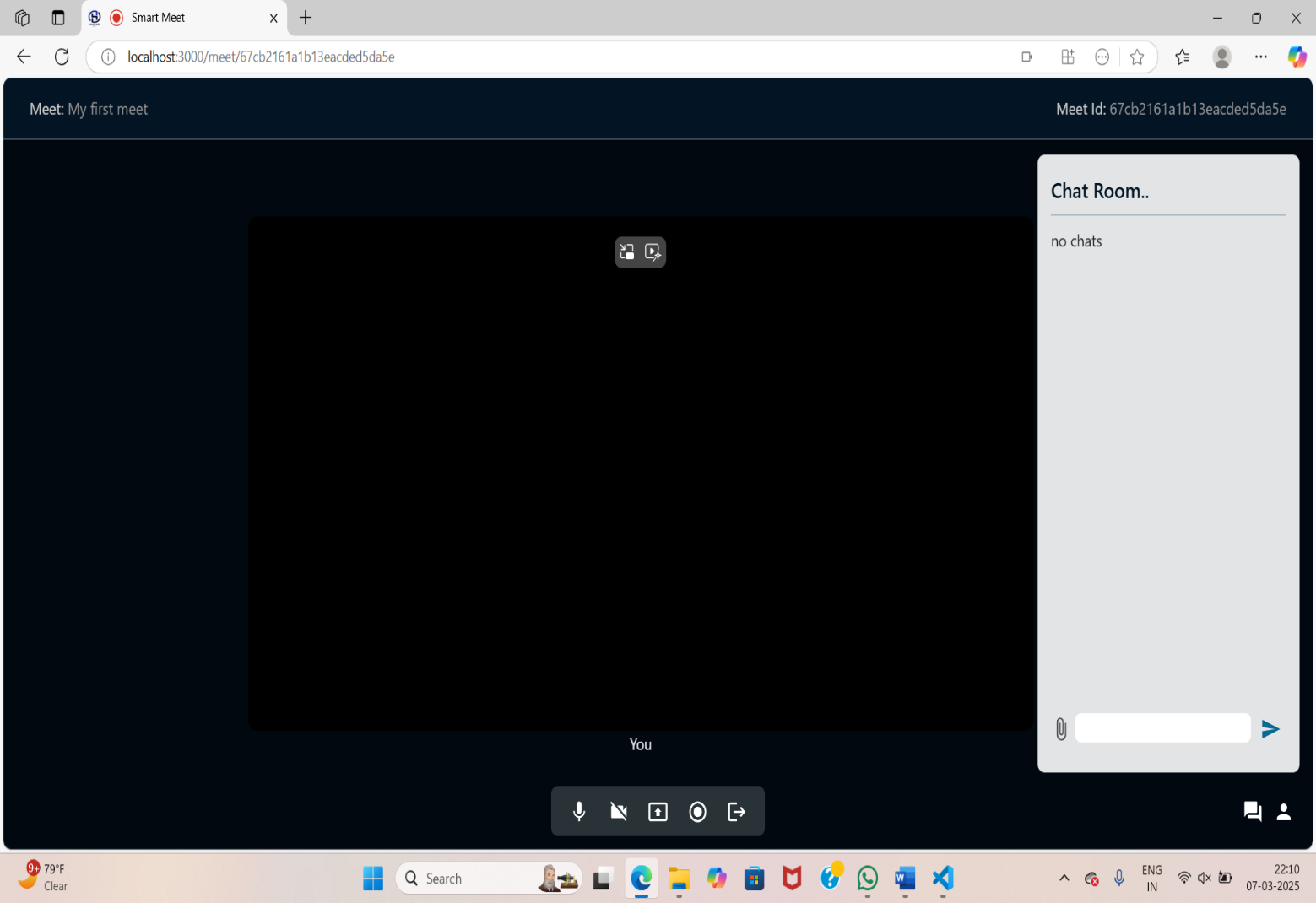


Home Page:

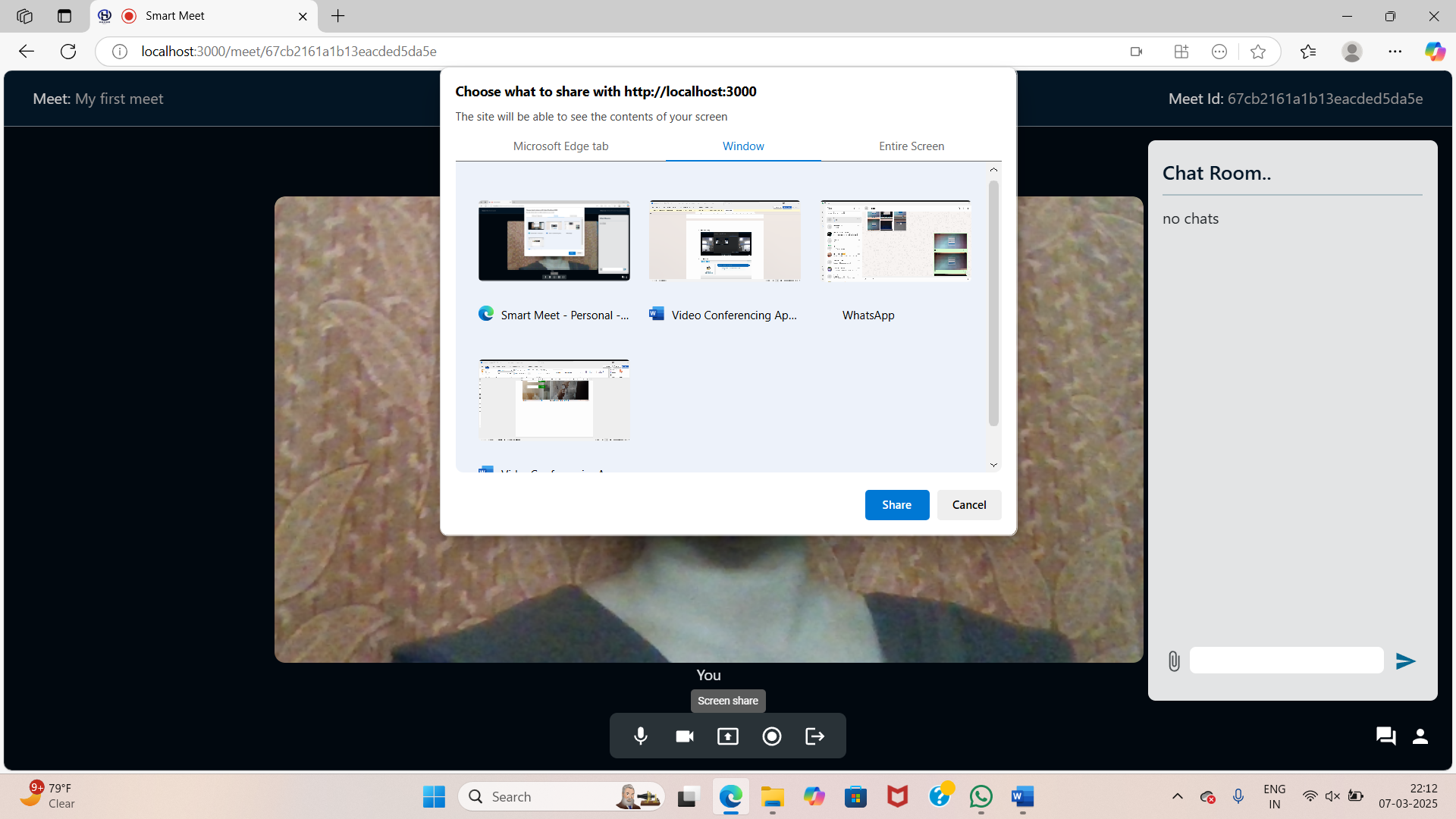
Create New Meet:



Meet Page:



Screen Sharing:



Profile Page:

