Full Stack Development with MERN

Project Documentation

1. Introduction

- **Project Title:** Smart Meet -Video Conferencing Application
- **Team Members:**

Hridhima Dabhade [22BCE3958]	Fullstack Lead-Oversee the entire project, ensure all components are integrated seamlessly, develop and manage server-side controllers, routes, and middleware, implement WebSocket communication, and ensure database integration and security.
Steffi Infancia. A [21MID0193]	Frontend Specialist -Develop and manage React components, integrate Agora SDK for video/audio, and ensure real-time updates with SocketContext.
Sujith. Y [21BBS0243]	Backend Developer -Develop and manage MongoDB models, create authentication middleware, handle room operations with roomhandler.js, and manage Socket.IO server integration.
Grandhi Naga Sri Santhosha Abiram [22BCE9412]	Documentation and testing- Write detailed project documentation, ensure proper commenting, manage testing setup, and maintain project documentation (README, setup guides, API documentation).

2. Project Overview

- **Purpose:** The purpose of the Smart Meet project is to create a fully-fledged video conferencing web application that facilitates seamless communication and collaboration.
- **Features:** Key features include secure user authentication, allowing users to sign up and log in securely. The dashboard serves as the central hub, enabling users to join or create meetings, view upcoming meetings, and access user settings. The core functionality lies in the meeting room, equipped with video and audio controls, real-time chat, screen sharing, and participant management. Users can switch between different layout options, such as grid or speaker view, to enhance their experience. The application is built with a responsive design, ensuring a consistent user experience across desktops, tablets, and mobile devices. Additionally, customizable settings and support resources are provided to cater to user preferences and troubleshooting needs, making Smart Meet an effective solution for virtual meetings.

3. Architecture

- **Frontend:** The front end of the Smart Meet project is architected using React.js, leveraging a component-based structure to ensure modularity, reusability, and

maintainability. Here's an overview of the architecture:

Key Components:

- ***App Component:*** This is the root component that initializes the application and sets up the routing using React Router. It defines the main navigation and routes to various pages like the Landing Page, Sign-Up, Login, Dashboard, and Meeting Room.
- ***Landing Page:*** Serves as the entry point for users, featuring navigation to sign-up and login pages. It likely includes components such as headers, footers, and promotional sections.
- ***Authentication Components:***
  Sign-Up Component: Handles user registration with input fields for user details, validation, and submission logic.
  Login Component: Manages user login with similar input fields and authentication processes.
- ***Dashboard Component:*** Post-login interface where users manage meetings. It includes components to display upcoming meetings, user profile settings, and options to create or join meetings.
- ***Meeting Room Component:*** The central component for video conferencing, incorporating several sub-components for enhanced functionality:
- ***Video/Audio Controls:*** Toggles for video and audio streams.
- ***Chat Component:*** Real-time messaging interface for participants.
- ***Participant Management:*** Displays a list of participants and allows for management actions.
- ***Screen Sharing Component:*** Enables users to share their screens.
- ***Layout Options:*** Provides different viewing modes such as grid view and speaker view.

State Management:

- ***Redux:*** Used for managing the global state across the application, ensuring that data flows seamlessly between components and the application remains performant.

Styling:

- ***CSS:*** Utilized for styling the components, ensuring a consistent and modern look throughout the application.

Routing and Navigation:

- ***React Router:*** Implements client-side routing to navigate between different pages and components without reloading the page, providing a seamless user experience.

- **Backend:** The backend architecture of the 'Smart Meet' video conferencing web application is built using Node.js and Express.js. Node.js provides the runtime environment for executing JavaScript on the server-side, while Express.js is used to create the server and handle routing. Middleware functions handle requests, parse JSON, manage cookies, and sessions, and CORS configuration allows communication between the frontend and backend hosted on different domains.
  - ***Routing using API Endpoints:*** The server features various API endpoints, including authentication routes for user registration, login, and logout, user routes for managing profiles and settings, meeting routes to create, join, and manage video conferences, and chat routes for real-time messaging within meetings. These endpoints facilitate CRUD operations, interacting with a MongoDB database via Mongoose schemas and models.
  - ***Real-time communication:*** Real-time communication is enabled through Socket.IO, which allows bidirectional communication between the server and clients. Socket events handle users joining or leaving meetings and

sending messages, with broadcasting to notify all participants of changes.

- ***Security:*** Security measures include JWT-based authentication and authorization, data validation and sanitization, and encryption of sensitive data like passwords. Error handling is managed through centralized middleware, with logging for debugging and monitoring.
- ***Error handling:*** Centralized error handling middleware to catch and respond to errors consistently. Logging errors and important events for debugging and monitoring purposes.
- ***Deployment and Scalability:*** For deployment and scalability, environment variables manage configurations for different environments, and deployment instructions are provided for platforms like AWS, or DigitalOcean. Scalability considerations include using load balancers and clustering to handle increased load.

- **Database:** The 'Smart Meet' video conferencing application employs MongoDB as its database, using Mongoose for object data modeling and interaction. The database is structured into two main collections: users and rooms.

  - The users collection manages user information, with a schema that includes fields for username, email, and password. Each user document ensures the email field is unique and mandatory, facilitating user authentication and preventing duplicate entries. Passwords are stored securely by hashing before saving them to the database, enhancing security.
  - The rooms collection handles the details of video conference rooms. Its schema comprises fields such as roomName, host, meetType, meetDate, meetTime, participants, and currentParticipants. The roomName identifies each room, while the host field records the creator of the room. The meetType, meetDate, and meetTime fields organize the meeting's specifics. The participants array lists all invited users, and currentParticipants tracks those currently in the meeting, allowing real-time updates and management of meeting attendance.
  - Interactions with MongoDB are facilitated by Mongoose, which provides an elegant API for performing CRUD operations. Creating a user involves adding a new document to the users collection with the necessary details. When scheduling a meeting, a new document is inserted into the rooms collection, capturing all pertinent information. Updating a room, such as adding current participants, involves modifying the relevant document in the rooms collection. These interactions ensure efficient data management and retrieval, supporting the application's functionality and real-time capabilities.

1. Collection: users
- Schema:
```
username:{
type: String,
require: true
},
email:{
type: String,
require: true,
unique: true
},
password:{
type: String,
require: true
```

}
```

2. Collection: Rooms
- Schema:
```
roomName:{
type: String
},
host: {
type: String,
require: true
},
meetType:{
type: String,
},
meetDate:{
type: String,
},
meetTime:{
type: String,
},
participants: {
type: Array
},
currentParticipants: {
type: Array
}
```

## 4. Setup Instructions

- **Prerequisites:**
  - Node.js: Required for running the JavaScript runtime and package management.
  - MongoDB: Required for the database.Install from MongoDB official website.
  - Git: Required for cloning the repository.
  - Agora RTC: For real time audio and video communications
  - Socket.io: To enable seamless connectivity between servers and clients

- **Installation:**
  1. Clone the repository:
     git clone https://github.com/harsha-vardhan-reddy-07/smart-meet.git
     cd smart-meet
  2. Install dependencies:
     cd client
     npm install
     cd ../server
     npm install
```

3. Set up environment variables
   PORT=4002
   NODE_ENV=development
   MONGODB_URL=<mongodb_url>
   JWT_SECRET=<jwt_secret_key>
4. Start the development servers
   cd ../server
   npm start
   cd ../client
   npm start
5. Access the app:
   http://localhost:4002 .

5. Folder Structure

- **Client:** The client side of the Smart Meet project, which is built using React.js, follows a typical React project structure:

  - ***src:*** Contains all the source code for the front end.

  - ***components:*** Houses reusable components like buttons, forms, headers, footers, and other UI elements.

  - ***pages:*** Includes the main pages of the application, such as the Landing Page, Sign-Up, Login, Dashboard, and Meeting Room.

  - ***redux:*** Contains Redux-related files, including actions, reducers, and store configuration.

  - ***styles:*** Holds CSS/SCSS files for styling the components and pages.

  - ***App.js:*** The root component that sets up routing and initializes the application.

  - ***index.js:*** The entry point of the React application, rendering the App component and integrating Redux and other providers.

- **Server:** The node.js backend of Smart Meet Video conferencing application's project is organized into several key directories and files, each serving a specific purpose to ensure clarity, modularity, and maintainability of the codebase.
  - At the root level, we use a package.json file, which serves as a central configuration hub for the project. It includes metadata about the project, such as dependencies, scripts for running the application, and other relevant information necessary for managing the Node.js application.
  - Inside the project, the config/ directory typically contains configuration files that define settings for the environment, database connections, and other configuration parameters. These settings are crucial for ensuring the application behaves correctly across different environments, such as development, testing, and production.
  - The controllers/ directory houses the business logic of the application. Each controller file, such as userController.js or meetingController.js, contains

functions that handle incoming requests from the API routes. These functions interact with services to perform operations such as fetching data from the database, processing data, and forming responses back to the client.

- Models are defined in the models/ directory, often using an ORM (Object-Relational Mapping) library like Mongoose for MongoDB for SQL databases. Model files such as User.js or Meeting.js define the structure of data entities stored in the database. They include schemas that specify the fields and their types, as well as methods for querying and manipulating data in the database.
- Routes are defined in the routes/ directory. These files, like userRoutes.js or meetingRoutes.js, define the endpoints of the API and map them to the corresponding controller functions. Routes provide a clear and organized way to manage incoming requests, separating concerns between different parts of the application.
- The services/ directory contains modules that encapsulate the business logic and interact with the models. Services are used by controllers to perform operations that involve more complex business logic, such as authentication, data validation, and other application-specific tasks.
- Middleware functions, found in the middleware/ directory, intercept incoming requests or outgoing responses. They provide a way to execute cross-cutting concerns like authentication, logging, error handling, and data parsing. Middleware functions can be applied globally to all routes
- Utility functions and helper methods are stored in the utils/ directory. These functions are reusable across different parts of the application, providing common functionalities such as data formatting, validation, error handling utilities, and other helper functions that streamline development tasks and ensure consistency throughout the codebase.
- The tests/ directory contains unit tests and integration tests for the application. These tests verify that individual units of code (functions, modules) work correctly and that different parts of the application work together as expected. Testing is essential for ensuring the reliability, correctness, and maintainability of the application over time.
- Finally, the entry point of the application, typically app.js initializes the server, sets up middleware, defines routes, connects to databases, and starts the server listening for incoming requests. It integrates all the components of the backend together, providing a cohesive structure that allows the application to function as a whole.

6. Running the Application

- Commands to start the frontend and backend servers locally.
  - **Frontend:** `npm start` in the client directory.
  - **Backend:** `npm start` in the server directory.

7. API Documentation

- User Authentication Endpoints

  - ***POST /api/user/register:*** Accepts POST requests to register a new user. Requires parameters in JSON format including username, email, and password. Example response confirms successful registration.

  - ***POST /api/user/login:*** Handles POST requests for user login, expecting email and password in JSON format. Returns a JWT token upon successful

authentication.

- User Management and Room Creation

  - ***GET /api/user/:*** Retrieves user information based on the provided user ID via a GET request. The response includes the user's ID, username, and email.

  - ***PUT /api/user/:*** Updates user information using a PUT request to the specified user ID. Parameters such as username and email can be updated in the JSON request body. The response confirms the successful update.

  - ***GET /api/rooms:*** Retrieves all meeting rooms with a GET request to /api/rooms, returning an array of room objects containing IDs, names, and descriptions.

  - ***POST /api/rooms:*** Allows creation of new meeting rooms via a POST request to /api/rooms. Requires parameters like name and description in the JSON request body. Example responses include the newly created room's ID, name, and description.

  These endpoints facilitate secure user authentication, user management functionalities, and the creation and retrieval of meeting rooms. Error handling and validation are implemented to ensure data integrity and security throughout the backend operations.

8. Authentication

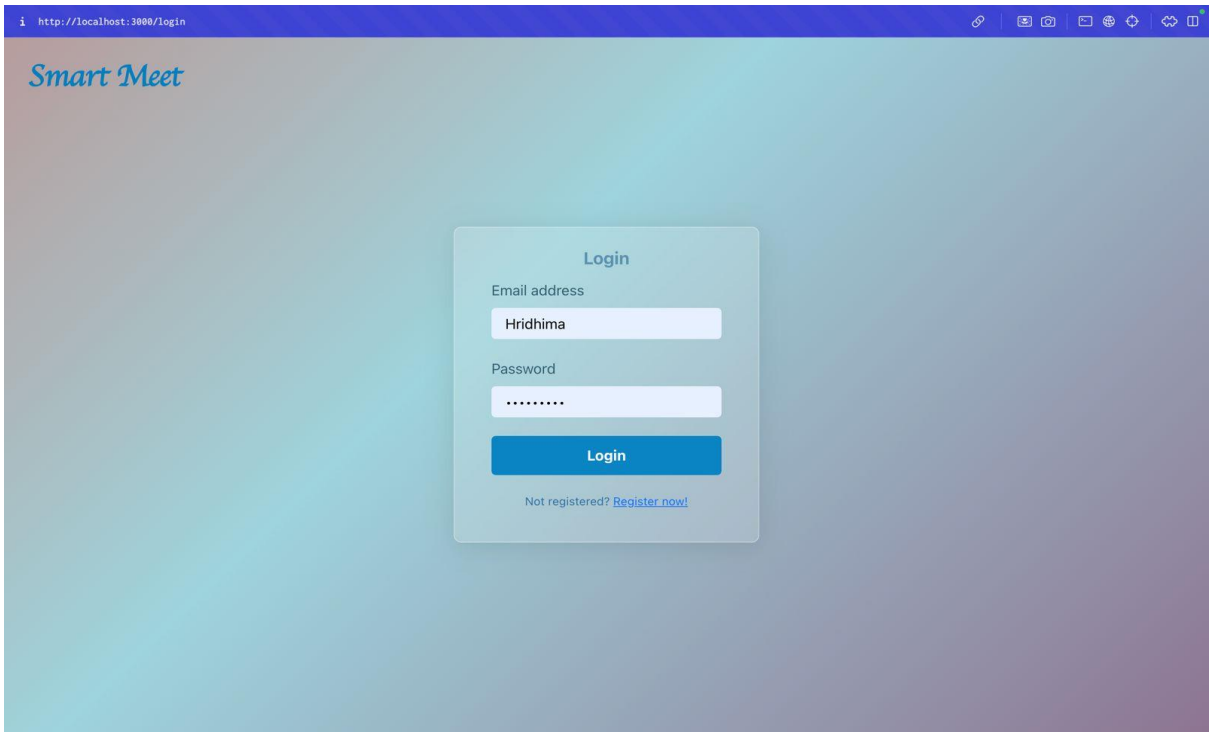
- Authentication in the Smart Meet project is managed using secure token-based authentication. When a user successfully logs in, a JWT (JSON Web Token) is generated and sent back to the client. This token is then included in subsequent requests' headers to authenticate and authorize access to protected endpoints.

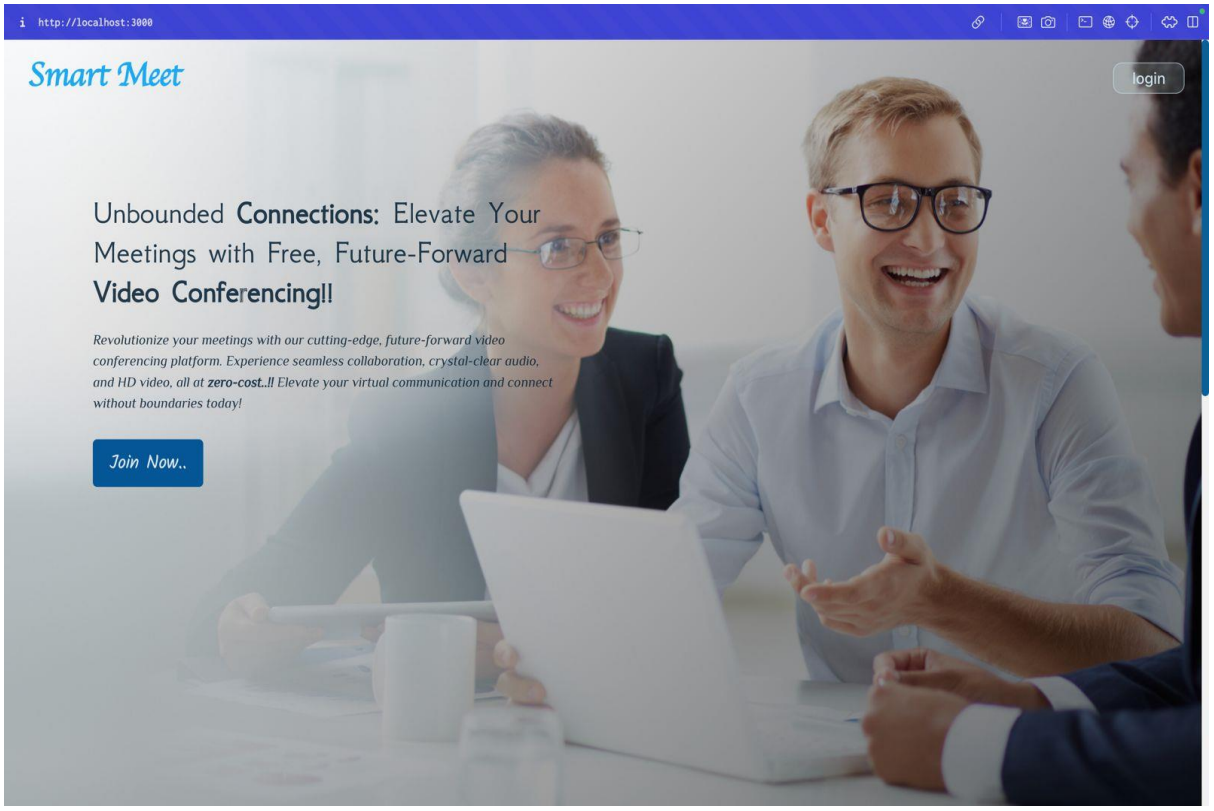
  - JWT tokens are utilized to maintain user sessions securely. They contain encoded information about the user's identity and permissions, signed with a secret key to ensure authenticity and integrity.

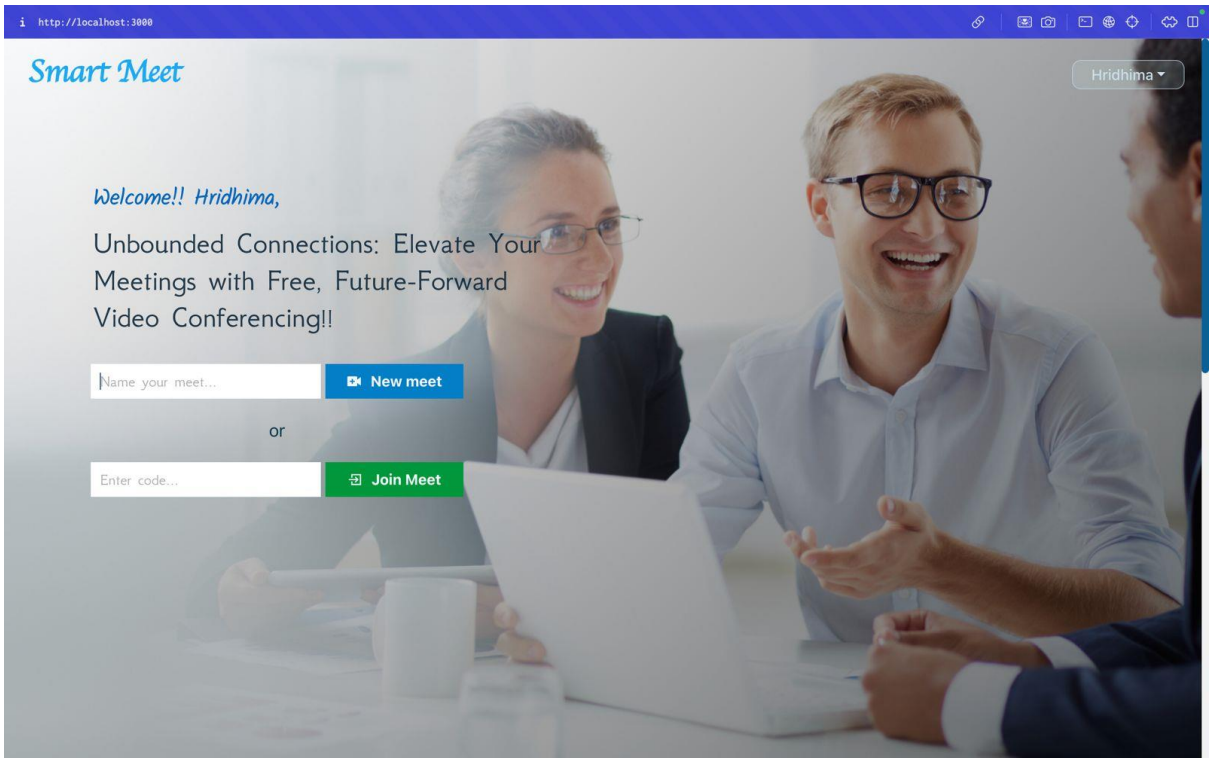
- Secure token-based authentication ensures that only authenticated users can access protected resources and endpoints within the Smart Meet application. This approach mitigates common security risks such as session hijacking and CSRF (Cross-Site Request Forgery) attacks by securely managing user sessions without storing session data on the server.

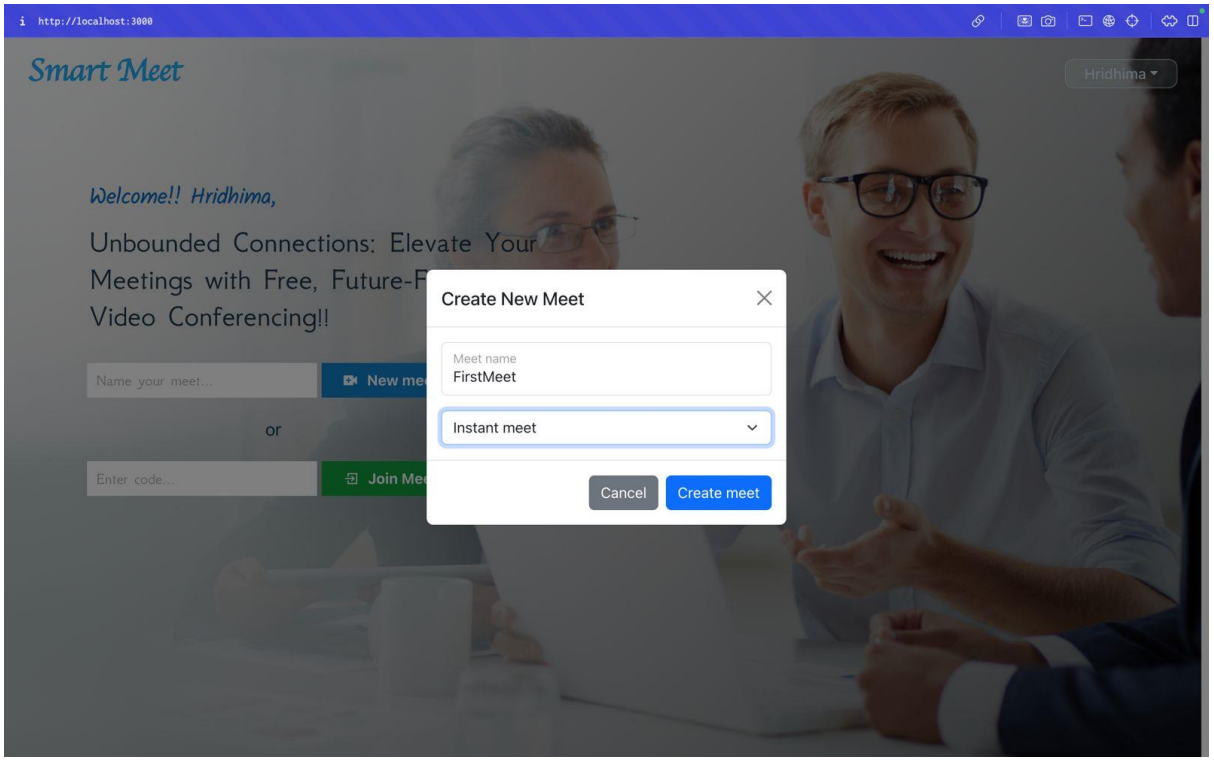
- Smart Meet prioritizes data security by employing encryption techniques both at rest and in transit. Sensitive data, such as user credentials and potentially sensitive meeting details, are encrypted before being stored in the database. In transit, HTTPS (HTTP Secure) is used to encrypt data exchanged between the client and the server, preventing eavesdropping and tampering during transmission.

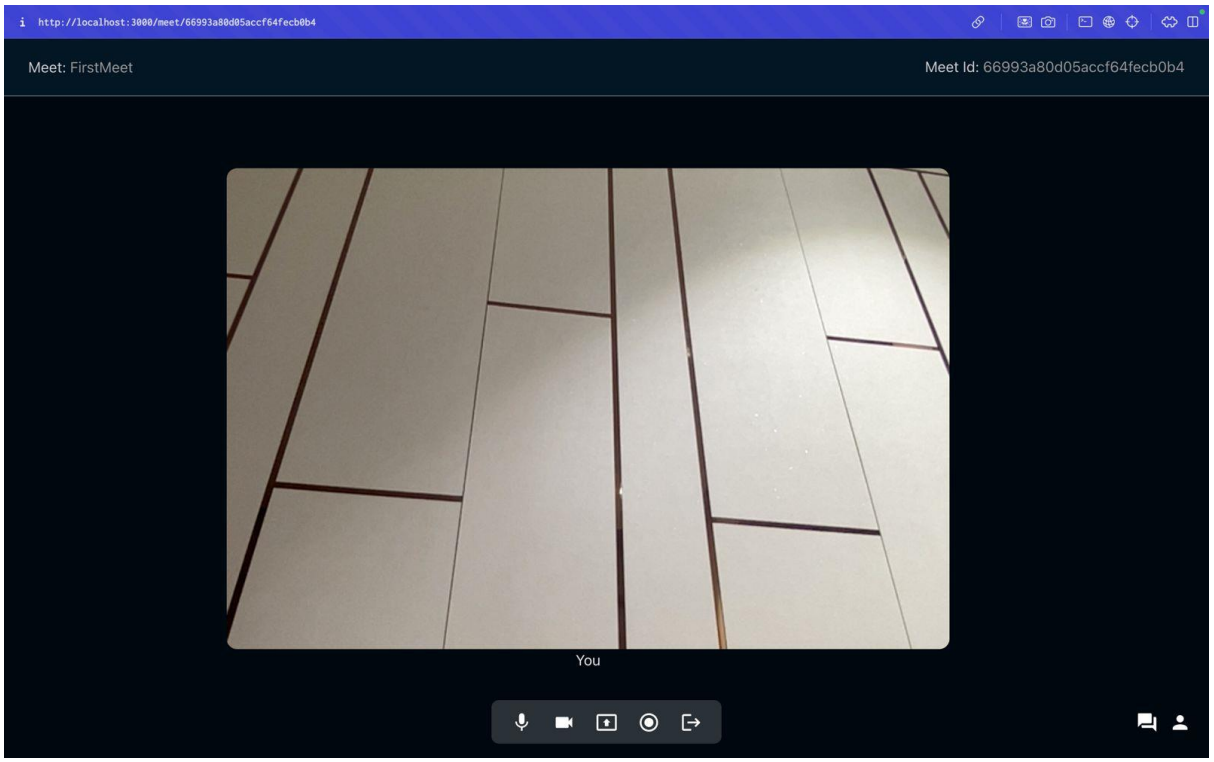
9. User Interface:

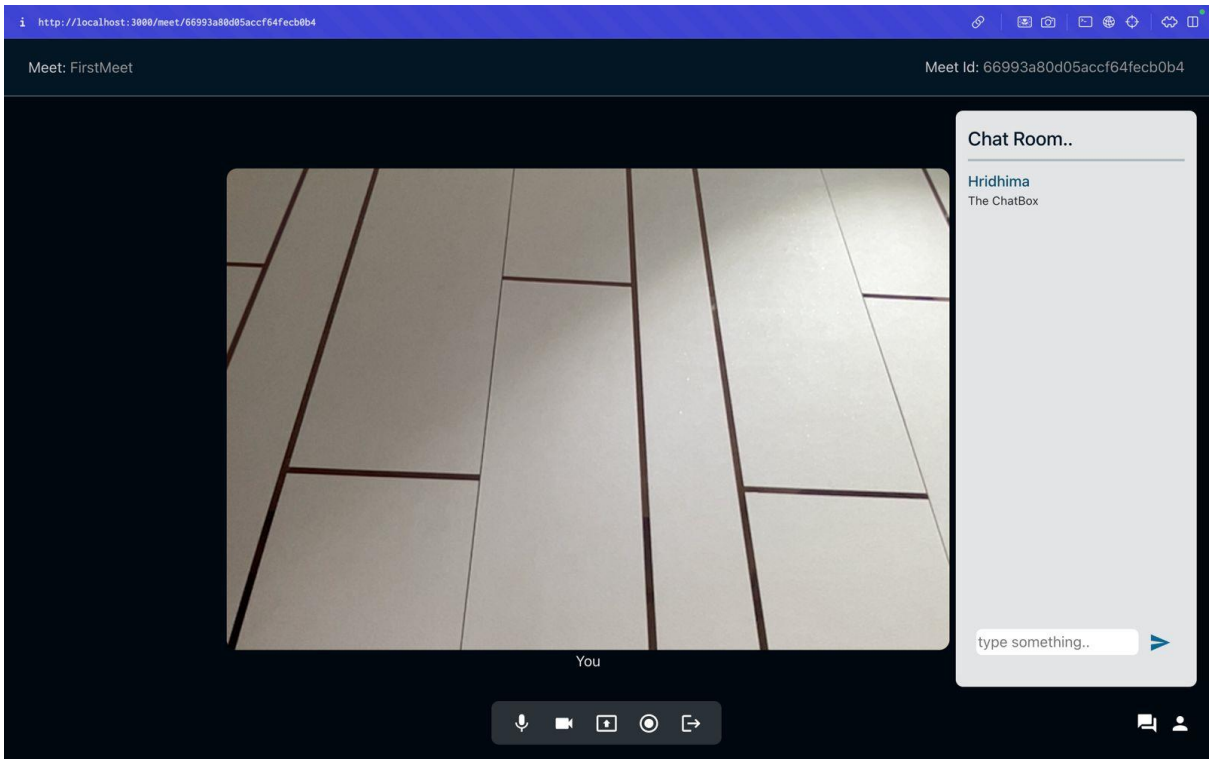

http://localhost:3000/login
Smart Meet
Login
Email address
Hridhima
Password
Login
Not registered? Register now!


http://localhost:3000
Smart Meet
login
Unbounded Connections: Elevate Your Meetings with Free, Future-Forward Video Conferencing!!
Revolutionize your meetings with our cutting-edge, future-forward video conferencing platform. Experience seamless collaboration, crystal-clear audio, and HD video, all at zero-cost..!! Elevate your virtual communication and connect without boundaries today!
Join Now..


http://localhost:3000
Smart Meet
Hridhima
Welcome!! Hridhima,
Unbounded Connections: Elevate Your Meetings with Free, Future-Forward Video Conferencing!!
Name your meet...
New meet
or
Enter code...
Join Meet


http://localhost:3000
Smart Meet
Hridhima
Welcome!! Hridhima,
Unbounded Connections: Elevate Your
Meetings with Free, Future-F
Video Conferencing!!
Name your meet...
or
Enter code...
Create New Meet
Meet name
FirstMeet
Instant meet
Cancel
Create meet


http://localhost:3000/meet/66993a80d05accf64fecb0b4
Meet: FirstMeet
Meet Id: 66993a80d05accf64fecb0b4
You


http://localhost:3000/meet/66993a80d05accf64fecb0b4
Meet: FirstMeet
Meet Id: 66993a80d05accf64fecb0b4
Chat Room..
Hridhima
The ChatBox
type something..
You

10. Testing

- **Test Strategy:**
    1. Test Planning
    2. Test Design
    3. Test Execution
    4. Bug Reporting
    5. Test Automation
    6. Performance testing
    7. User Acceptance Testing
    8. Reporting and Metrics
- **Tools Used:** Jira, VS Code, Git and Github

11. Screenshots or Demo

- https://drive.google.com/drive/folders/11JbcCwR200ucWqSKoQcG2oyNUCG9oOGj

12. Known Issues

- There are currently no bugs or issues that users or developers should be aware of.

13. Future Enhancements

- Potential Future Features could include integrating a time zone API to retrieve the time zones of participants, simplifying scheduling meetings, and integrating calendar services to create and add meetings directly from users' calendars. Other enhancements might include advanced analytics for meeting performance, AI-driven features for automatic note-taking and transcription, and more robust security measures to ensure privacy and data protection. Additionally, features like virtual backgrounds, improved multi-language support could significantly enhance the user experience. These improvements would further enhance the application's functionality, making it even more user-friendly and versatile for a broader range of users.