Full Stack Development with MERN

Project Documentation

1. Introduction

- **Project Title:** ShailMeet Video Conferencing Application
- **Team Members:** Shailesh Yadav (Full Stack Developer)

2. Project Overview

- **Purpose:**
  The purpose of ShailMeet is to provide a seamless and efficient platform for video conferencing, allowing users to connect and collaborate in real time with high-quality audio and video features.
- **Features:**
  - Real-time video and audio communication using WebRTC
  - Screen sharing
  - Chat functionality
  - Meeting scheduling
  - Participant management
  - Multi-user support

3. Architecture

- **Frontend:**
  The frontend of ShailMeet is built using React, with a focus on providing an intuitive and user-friendly interface. Components are reusable, and state management is handled via Redux.
- **Backend:**
  The backend is powered by Node.js and Express.js, serving REST APIs for managing users, meetings, and video calls. WebSockets are used for real-time communication.
- **Database:**
  MongoDB is used to store user data, meeting details, and chat logs. Mongoose is used for schema modeling and interaction with the database.

4. Setup Instructions

- **Prerequisites:**
  - Node.js

    - MongoDB
    - NPM/Yarn
    - Agora SDK
    - Socket.IO
- **Installation:**
    - Clone the repository from GitHub.
    - Install dependencies using `npm install` or `yarn`.
    - Set up environment variables (e.g., MongoDB connection string, JWT secret).

5. Folder Structure

Client Folder:

- **node_modules/:** Holds all the dependencies required for the React project, automatically generated after running `npm install`.
- **public/:** Contains static assets like `index.html`, which serves as the entry point of the React app.
- **src/:** The primary folder where the entire source code of the frontend resides.
    - **components/:** Contains modular and reusable UI components such as buttons, headers, video chat windows, etc.
    - **context/:** Manages the application's global state using React's Context API for features like user information or chat status.
    - **images/:** Stores images or other static assets used throughout the app.
    - **pages/:** Contains individual page components representing different screens in the app.
    - **protectedRoute/:** Ensures that only authenticated users can access certain routes within the application.
    - **styles/:** Includes the CSS or SCSS styles used across the components and pages.
    - **AgoraRTMSetup.js and AgoraSetup.js:** Configures the Agora SDK for handling video conferencing, real-time messaging, and media streaming features.
    - **App.js:** The root component that renders other components and handles navigation.
    - **Socket.js:** Manages real-time communication and data synchronization using Socket.IO.
    - **index.js:** The entry point file that renders the App component into the DOM.
    - **setupTests.js:** Used to configure tests for the app, ensuring its reliability.
    - **package.json:** Provides metadata about the project, such as its name, version, and scripts, as well as the list of dependencies used in the project.
    - **package-lock.json:** A lockfile to ensure the same versions of dependencies are installed across different environments.

    - **README.md:** Contains documentation for setting up and running the project.

Server Folder:

- **controllers/:** Contains the business logic for handling API requests.
    - **auth.js:** Manages user authentication tasks like login, registration, and token management.
- **middleware/:** Custom middleware for request/response processing.
    - **auth.js:** Verifies user authentication using JWT for secure API access.
- **models/:** Defines the database schema and models for MongoDB.
    - **Rooms.js:** Defines the structure of a "room" document for video conferencing room details.
    - **User.js:** Defines the structure of a "user" document storing user credentials, profile info, and authentication tokens.
- **routes/:** Defines the API routes for interacting with the backend.
    - **auth.js:** Manages routes related to user authentication, such as `/login`, `/signup`, or `/logout`, facilitating secure user sessions.
- **socket/:** Contains logic for managing real-time communication using Socket.IO.
    - **roomHandler.js:** Handles socket events related to video conferencing rooms, such as creating a new room, joining a room, or managing room participants.
- **.env:** The environment file holds sensitive configuration data like database connection URLs, API keys, and other credentials required by the backend.
- **index.js:** Entry point for the Node.js server. It initializes the server, connects to the database, sets up middleware, and listens for incoming requests.
- **package.json:** Contains metadata about the project, including the project name, version, and list of dependencies required for the backend.
- **package-lock.json:** A lockfile generated by npm to ensure consistent installation of dependencies.

6. Running the Application

- **To start the application locally:**
    - **Frontend:** Run `npm start` in the client directory.
    - **Backend:** Run `npm start` in the server directory or `node index.js`.

7. API Documentation

- **GET /api/meetings:** Fetch all meetings.
    - Parameters: None
    - Response: Array of meeting objects.
- **POST /api/meetings:** Create a new meeting.

    - Parameters: Meeting details (JSON)
    - Response: Created meeting object.
- **Authentication required:** JWT token for all API endpoints related to meetings and user data.

8. Authentication

- **JWT Tokens:** JSON Web Tokens (JWT) are used for stateless authentication. A token is issued upon login and is required for accessing protected routes.
- **Middleware:** The backend uses middleware to verify the validity of tokens in protected routes.
- **Sessions:** No session management is used, as the application relies on JWT for authentication.

9. User Interface

- **Screenshots of the user interface:**
    - The meeting dashboard
    - The video conferencing screen
    - The chat window

10. Testing

- **Jest** is used for unit testing React components.
- **Supertest** is used for testing the Express API endpoints.

11. Screenshots or Demo

- https://drive.google.com/file/d/1OfJ5lo7GrPZvbLCvt6dIv-etJbOKWq10/view?usp=sharing

12. Known Issues

- Users might experience minor delays in video syncing during large meetings.
- Some users report issues with screen sharing on older browsers.

13. Future Enhancements

- Adding recording functionality for meetings.
- Integration with third-party calendar services (e.g., Google Calendar).
- Enhancing the chat system to support file sharing.