CodeSync - Technical Design Document

Table of Contents

Introduction and System Overview

Introduction

CodeSync is an advanced web application designed to facilitate seamless virtual collaboration and coding interactions. Inspired by the functionalities of Google Meet, CodeSync integrates essential video conferencing features with the additional power of an online compiler, enabling users to connect, communicate, and collaborate efficiently.
In the modern era of remote work and online learning, CodeSync aims to provide an all-in-one solution that fosters real-time engagement, coding collaboration, and interactive communication. Its user-friendly interface and robust functionality make it suitable for both professional and educational use cases.

System Objectives

CodeSync is built with the goal of making remote collaboration and coding simple, efficient, and secure. Here's what we aim to achieve:

1. **Better Teamwork**: Allow people to connect, share ideas, and work together seamlessly through video calls, chat, and shared tools.
2. **Integrated Coding**: Provide a shared, real-time coding environment so teams can work on code together without needing multiple apps.
3. **Ease of Use**: Create an interface that’s intuitive and accessible, so anyone—from beginners to experienced users—can use it comfortably.
4. **Support for Growth**: Design a system that can handle more users and rooms as it grows.
5. **Data Safety**: Prioritize user security and privacy with strong authentication and secure communication.

Features

- **User Authentication**: Secure login/logout functionality to protect user data.
- **Room Management**: Users can create or join virtual rooms tailored for group interactions.
- **Video Sharing**: High-quality video streaming for effective communication.
- **Live Chat**: Instant messaging within rooms to enhance collaboration.
- **Participant Management**: A dynamic list of participants visible in every room.
- **Shared Compiler**: Real-time code editing and execution, ideal for pair programming or collaborative coding sessions.

Architecture

High-Level Architecture

CodeSync is designed to seamlessly integrate video conferencing, real-time chat, and a shared online compiler into a single platform. The architecture follows the MERN stack (MongoDB, Express.js, React.js, Node.js) and leverages scalable, efficient technologies to provide a smooth user experience.

The system is divided into three main layers:

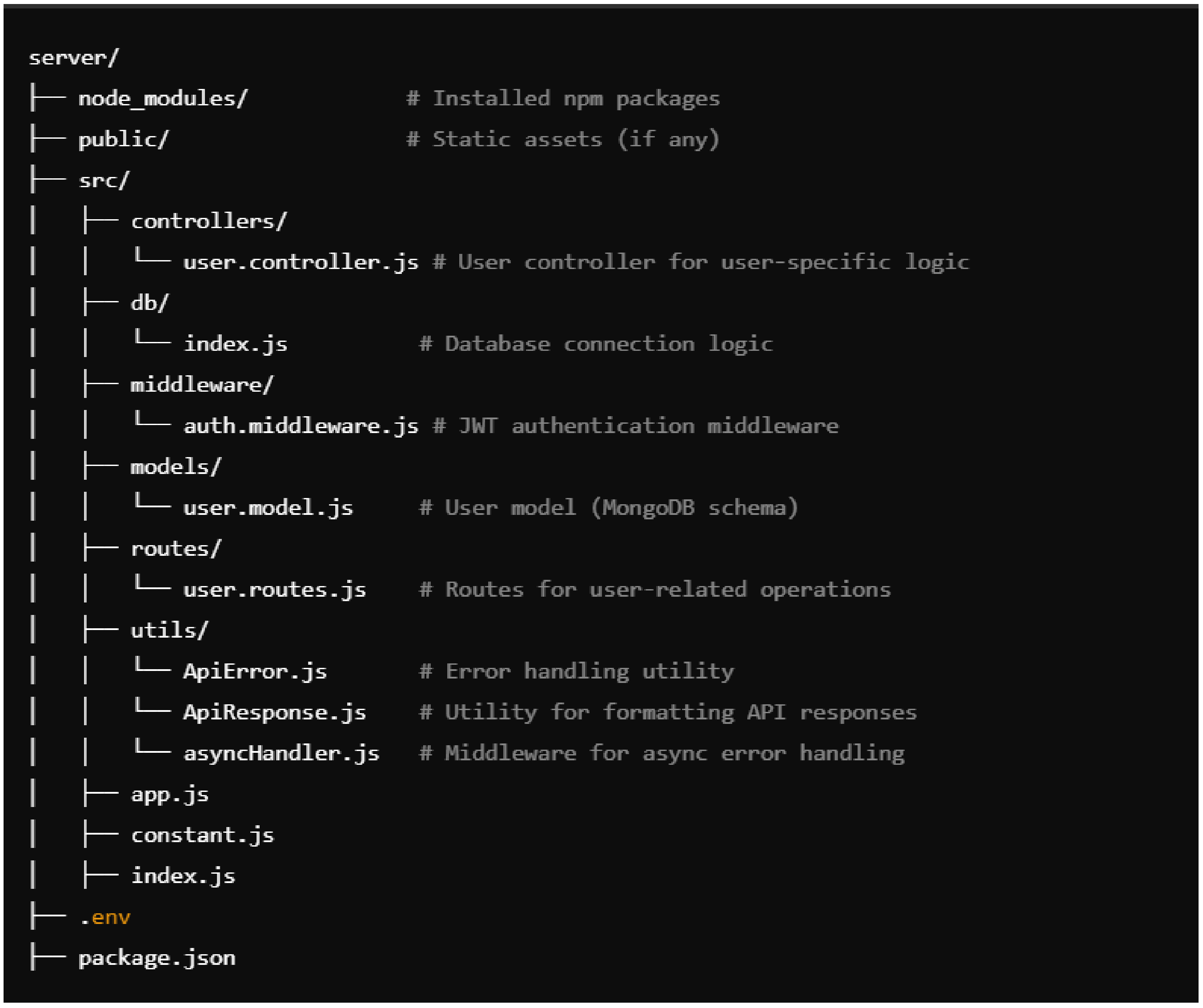
1. Frontend: Provides the user interface using React, ensuring a dynamic and responsive experience.
2. Backend: Manages the business logic, APIs, and database interactions through Node.js and Express.
3. Database: Stores persistent data like user information, chat history, and room details using MongoDB.

Backend Implementation

Core Technologies:

- **Node.js:** As the runtime environment for your backend.
- **Express.js:** Web application framework for building APIs.
- **MongoDB:** Database for storing user data, room details, etc.
- **JWT:** For handling authentication.
- **Socket.IO:** For real-time communication, like video sharing, chat, and collaboration.
- **Bcrypt.js:** For password hashing.

Directory Layout



```
server/
├── node_modules/            # Installed npm packages
├── public/                  # Static assets (if any)
├── src/
│   ├── controllers/
│   │   └── user.controller.js # User controller for user-specific logic
│   ├── db/
│   │   └── index.js          # Database connection logic
│   ├── middleware/
│   │   └── auth.middleware.js # JWT authentication middleware
│   ├── models/
│   │   └── user.model.js     # User model (MongoDB schema)
│   ├── routes/
│   │   └── user.routes.js    # Routes for user-related operations
│   ├── utils/
│   │   └── ApiError.js       # Error handling utility
│   │   └── ApiResponse.js    # Utility for formatting API responses
│   │   └── asyncHandler.js   # Middleware for async error handling
│   ├── app.js
│   ├── constant.js
│   ├── index.js
├── .env
├── package.json
```

Backend Design - API, Database Design and Middlewares

API Endpoints

The backend exposes RESTful API endpoints categorized under:

Authentication

- **POST** /user/register: Register a new user.
- **POST** /user/login: User login.
- **POST** /user/logout: Logout the user.

Database Design

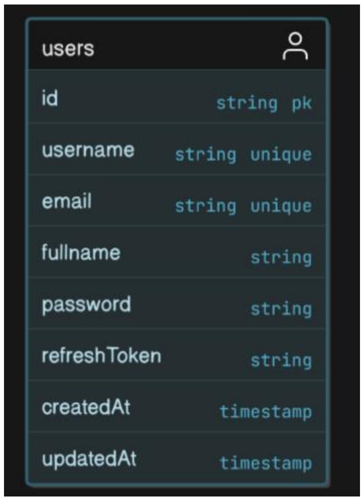
The Database Design section outlines the structure of the data and how it is organized within the database, ensuring it meets the requirements for the application's functionality.

1. **Database Choice:**
   We are using MongoDB as our database for this project, which is a NoSQL, document-based database that offers flexibility in handling various data structures. MongoDB allows us to store data in JSON-like documents, which is ideal for applications with rapidly changing data models.

2. **Collections:**
   - Users Collection:
     This collection will store the details of users who interact with the application, including their credentials and other essential information.

     Fields:
     - name: The user's full name.
     - email: A unique identifier for the user, used for login.
     - password: The user's encrypted password.
     - Fullname: Full name of the user
     - refreshToken: For auth purpose.

Schema for User:



users	
id	string pk
username	string unique
email	string unique
fullname	string
password	string
refreshToken	string
createdAt	timestamp
updatedAt	timestamp

Middleware’s

Authentication Middleware (auth.middleware.js)

This middleware verifies whether the user is authenticated before accessing certain protected routes. It checks if a valid JWT token is present in the request headers, ensuring that only authorized users can proceed.

Middleware functions are key to enhancing the functionality of the server, allowing us to handle tasks like authentication, error handling, logging, and more. Below are the key middleware functions implemented in the system.

JWT Authentication Middleware (verifyJWT)

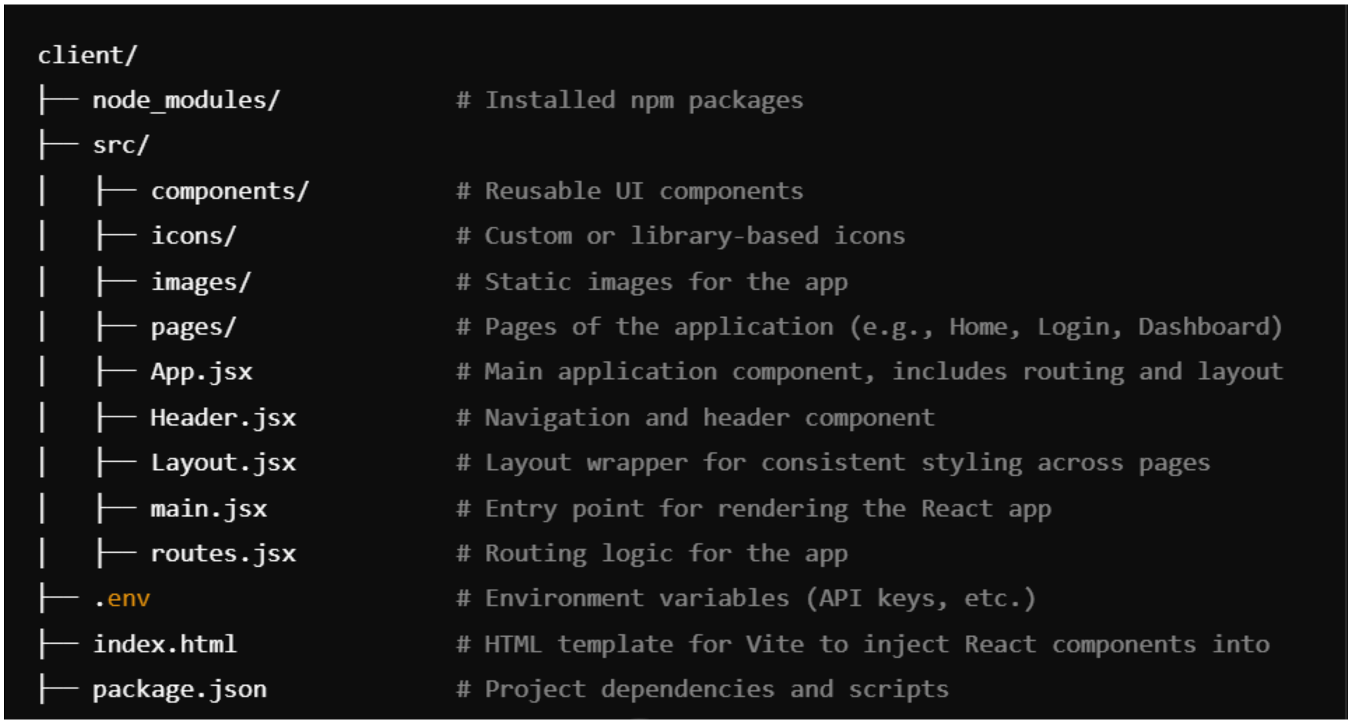
This middleware is used to authenticate users based on a JSON Web Token (JWT). It ensures that only authorized users can access protected routes. It checks for the presence of an accessToken in the request (either in the cookies or the Authorization header) and verifies it.

Frontend Design

Frameworks and Tools

- **Vite:** A modern build tool for fast development and efficient bundling.
- **React & React DOM:** For building and rendering dynamic UI components.
- **React Router DOM:** Handles routing and navigation.
- **UUID:** Generates unique identifiers for features like room IDs.
- **Environment Variables:** .env file for securely managing API endpoints and secrets.

Project Structure



```
client/
├── node_modules/          # Installed npm packages
├── src/
│   ├── components/        # Reusable UI components
│   ├── icons/             # Custom or library-based icons
│   ├── images/            # Static images for the app
│   ├── pages/             # Pages of the application (e.g., Home, Login, Dashboard)
│   ├── App.jsx            # Main application component, includes routing and layout
│   ├── Header.jsx         # Navigation and header component
│   ├── Layout.jsx         # Layout wrapper for consistent styling across pages
│   ├── main.jsx           # Entry point for rendering the React app
│   ├── routes.jsx         # Routing logic for the app
├── .env                   # Environment variables (API keys, etc.)
├── index.html             # HTML template for Vite to inject React components into
├── package.json           # Project dependencies and scripts
```

Frontend Design – Routing flow, State Handling and Key Components

Routing

Implemented using React Router:

1. **/ (Home):** The home page, which renders the User component. This is the first child route under the main layout.
2. **/meet-room:** This route leads to the JoinRoom page, where users can join a meeting by selecting or creating a room.
3. **/meet-room/:roomid:** A dynamic route that includes a roomid parameter in the URL. This is where users join a specific meeting room. The MeetRoom component will be rendered here, and it can access the room ID via the roomid parameter.

State Handling

- **Local State**: Managed using *useState* and *useEffect* hooks.
- **Authentication State**:
  - Stored in *localStorage.*
  - Accessed via custom hooks or utility functions.
- **Data Fetching**:
  - Utilizes fetch API
  - Handles loading and error states.

Key Components

Layout:

- The Layout component serves as the wrapper for all pages. It includes a Header component and renders the child components via the Outlet component from react-router-dom. The Outlet will render the appropriate page based on the active route, making Layout the common container for all pages.

Header

- The Header component provides the top navigation for the app. It includes:
  - A logo section with a link to the home page.
  - A Meet ID display (for the meeting room identifier).
  - Navigation buttons for the "Lobby" and "Create Room" options.
  - A mobile navigation button for toggling a menu on smaller screens (hamburger icon).

User

- The User component manages the user authentication flow, allowing users to either sign up or log in. It uses a *useState* hook to toggle between the *Signup* and *Login* components based on the *flagSignup* state.
    - The *Register* function toggles the view to the *Signup* form.
    - The *Signin* function toggles the view to the *Login* form.

Signup and Login

- These are authentication components that handle user registration and login. They are conditionally rendered based on the *flagSignup* state in the User component.

MeetChatBox

- **Purpose**: Displays and handles messages in a meeting room.
- **Features**: Shows a list of messages, allows users to send new ones with a timestamp.
- **State**: *messages* (list of messages), *newMessage* (new message being typed).

MeetParticipants

- **Purpose**: Displays a list of meeting participants with their online status.
- **Features**: Shows participant's names and status (online/offline).
- **State**: *participants* (list of participants).

MeetVideo

- **Purpose**: Displays meeting participants' video feeds.
- **Features**: Shows participants' status and can be expanded to show video.
- **State**: *participants* (list of participants).

Security Strategies

Authentication

- **JWT Tokens**:
  - Securely generated and signed with a secret key.
  - Stored in the client's *localStorage.*
- **Password Security**:
  - Passwords hashed using *bcrypt* before storing in the database.
  - Plain passwords are never stored or logged.

Authorization

- **Protected Routes**:
  - Backend routes require valid JWT tokens.
  - Frontend routes use higher-order components to restrict access.
- **Input Validation**:
  - Sanitization of inputs to prevent injection attacks.
  - Validation rules applied on both client and server sides.

CORS Configuration

CORS (Cross-Origin Resource Sharing) is configured using the cors package to allow cross-origin requests from trusted domains.

Environment Variable:

- The allowed origin is managed by the CORS_ORIGIN variable, which can be set to * (all origins) or a specific domain.

Security:

- In production, only trusted domains should be allowed to access the API to prevent security risks.

Deployment Plans

Environment Setup

- **Backend Environment Variables**:
    - PORT: Port number for the server.
    - DB_CONNECTION_STRING: MongoDB connection URI.
    - JWT_SECRET: Secret key for signing JWTs.
- **Frontend Environment Variables**:
    - VITE_API_URL: Base URL for the backend API.

Deployment Steps

1. **Backend Deployment**:
    - Host on platforms like Heroku, AWS EC2, or DigitalOcean.
    - Ensure environment variables are securely set.
    - Use process managers like PM2 for process management.
2. **Frontend Deployment**:
    - Build the React application using npm run build.
    - Host static files on services like Netlify, Vercel, or serve through NGINX.
3. **Domain and SSL**:
    - Configure a custom domain.
    - Set up SSL certificates for secure HTTPS communication.
4. **Database Hosting**:
    - Use managed MongoDB services like MongoDB Atlas.
    - Configure IP whitelisting and security measures.
5. **Continuous Deployment**:
    - Set up CD pipelines to automatically deploy on code changes.
    - Use GitHub Actions or other CI/CD tools.

Testing

Frontend Testing (Localhost):

- Purpose: Test user interactions and UI behavior during development.
- Approach: Test the interactions between components on browser
- Tools:
  - React Testing Library: For testing React components.
  - Localhost: Run the app locally for manual testing and debugging.
- Debugging: Utilize console.log() to log values, state changes, and errors during development.

Backend Testing (Postman):

- Purpose: Test API endpoints to ensure they are functioning correctly.
- Approach: Use Postman to send requests and verify API responses, status codes, and data accuracy.
- Tools:
  - Postman: For manual API testing and verifying request/response flow.
- Debugging: Use console.log() to print out API responses, error messages, and check for issues during development.

Future Enhancements

---

1. Frontend:

- **Code Splitting**: React's lazy loading and dynamic imports to load only necessary components, improving performance.
- **Responsive Design**: Ensures the app is optimized across devices using CSS frameworks like TailwindCSS.
- **State Management**: Efficient state handling through React hooks to minimize unnecessary re-renders.

2. Backend:

- **API Rate Limiting**: Implements throttling to prevent overloading the server with requests, ensuring fair use.
- **Database Optimization**: Uses indexing and optimized queries to ensure fast data retrieval.
- **Scalable Infrastructure**: Hosted on cloud services like AWS or similar, allowing for horizontal scaling based on user load.

3. General:

- **Caching**: Leveraging caching mechanisms (e.g., Redis) to store frequently accessed data and reduce database load.
- **Load Balancing**: Distributes incoming traffic across multiple servers to avoid bottlenecks and ensure high availability.

Conclusion

CodeSync is a MERN stack-based online meeting platform with features similar to Google Meet. It includes frontend components built with React, such as user authentication (login/signup), room management, video streaming, messaging, and participant management. The backend is API-driven, tested using Postman. The project leverages **Vite** for faster builds and **React Router** for navigation.

Key features:

- **User Page**: Handles login and signup.
- **MeetChatbox**: Real-time messaging.
- **MeetParticipants**: Displays participants with status.
- **MeetVideo**: Video streaming for meetings.
- **CORS Configuration**: Handles cross-origin requests with credentials.

Testing is done via **Postman** for the backend and **console.log()** for debugging during development. The application aims to provide a seamless real-time collaboration experience.