Project: Messaging Service Prototype

GitHub link: https://github.com/botgirish/Realtime_Chat_Application

Name: Girish Sai Krishna Akula

Roll: 2201EE06

Dept.: Electrical and Electronics Engineering

University: IIT Patna

Project Overview

This project is a messaging API designed for both group and individual chats, real-time communication using WebSockets, and secure authentication using JWT tokens and cookies. The application is built using the MERN stack (MongoDB, Express, React, Node.js), and it supports scalable, high-performance messaging functionalities for large user bases.

System Components

1. **Frontend: React.js**
   - **Purpose**: Provides the user interface for individual and group chat, login, and signup functionalities.
   - **Key Libraries**:
     - React: For building reusable UI components.
     - Tailwind CSS: For styling and responsive layouts.
     - Axios: For making HTTP requests to the backend.
     - Socket.io-client: For handling WebSocket-based real-time communication with the server.
   - **Why React?**: React is chosen due to its efficiency in rendering, component-based structure, and ease of state management.
2. **Backend: Node.js & Express.js**

- **Purpose**: The backend handles API requests, authentication, and real-time messaging using WebSockets.
- **Key Libraries**:
    - Express: A minimal and flexible Node.js framework for handling HTTP requests and routing.
    - Socket.io: To provide real-time, bi-directional communication between clients and the server.
    - JWT (jsonwebtoken): For securing user authentication and authorization via tokens.
    - bcrypt: For password hashing to ensure secure storage of user passwords.
- **Why Node.js & Express?**: This combination provides non-blocking, asynchronous, and event-driven architecture, ideal for a real-time messaging system.

3. **Database: MongoDB**

- **Purpose**: Stores user data, chat messages, group details, and session information.
- **Key Libraries**:
    - Mongoose: An Object Data Modeling (ODM) library that allows easy interaction with MongoDB from Node.js.
- **Why MongoDB?**: MongoDB's schema-less design makes it ideal for rapidly changing, unstructured data like chat messages and user details.

4. **Authentication & Security**

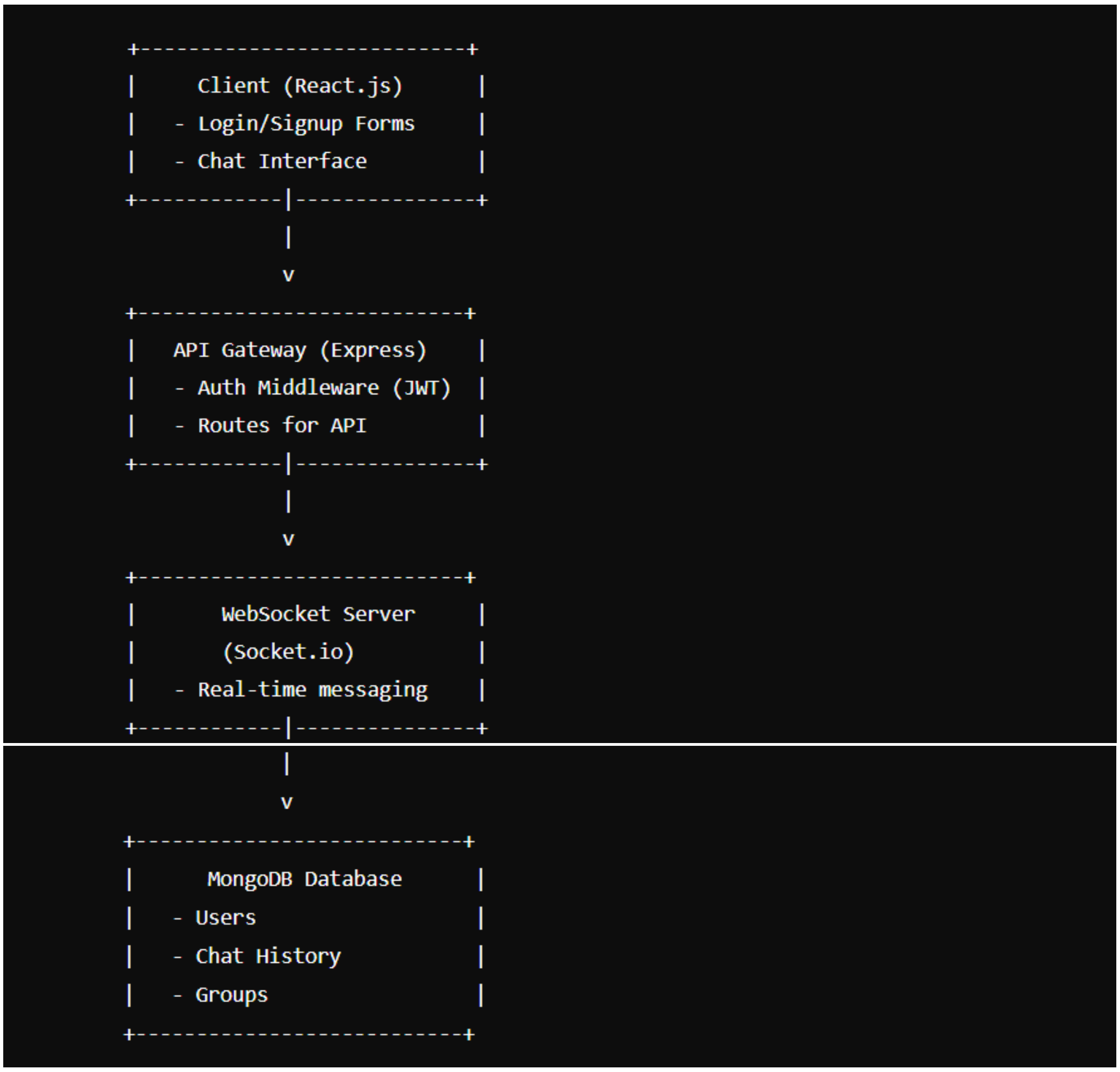
- **JWT (JSON Web Token)**: Used to securely transmit information between parties as a JSON object. JWT ensures stateless authentication, making it scalable and secure.
- **bcrypt**: Used for hashing passwords to ensure that even if the database is compromised, passwords are securely stored.
- **Cookies**: Utilized for storing session tokens and maintaining user login state.

5. **Real-Time Communication: WebSockets (Socket.io)**
    - **Purpose**: Enables real-time messaging between users.
    - **Why WebSockets?**: Unlike HTTP, WebSocket provides full-duplex communication, allowing messages to be pushed from the server to the client without polling.

Reasoning Behind Libraries & Frameworks

1. **MERN Stack (MongoDB, Express, React, Node.js)**:
    - Provides a full-stack JavaScript solution for both frontend and backend.
    - MongoDB's schema-less structure allows flexibility for chat applications where messages may vary in size and content.
    - React's component-based design makes building responsive UI easier and faster.
    - Node.js and Express offer scalability and speed for handling API requests and real-time messaging.
2. **WebSockets (Socket.io)**:
    - WebSockets provide low-latency, real-time communication, ideal for chat-based applications where instant message delivery is critical.
3. **JWT & bcrypt**:
    - JWT ensures that authentication is stateless, allowing scalability.
    - bcrypt securely hashes passwords, protecting users from potential breaches.

System Design Diagram :



```
+--------------------------+
|     Client (React.js)     |
|   - Login/Signup Forms    |
|   - Chat Interface        |
+------------|--------------+
             |
             v
+--------------------------+
|  API Gateway (Express)    |
|   - Auth Middleware (JWT) |
|   - Routes for API        |
+------------|--------------+
             |
             v
+--------------------------+
|      WebSocket Server     |
|      (Socket.io)          |
|   - Real-time messaging   |
+------------|--------------+
             |
             v
+--------------------------+
|     MongoDB Database      |
|   - Users                 |
|   - Chat History          |
|   - Groups                |
+--------------------------+
```

Libraries and Dependencies

Frontend

- React: Provides efficient component rendering for UI.
- Tailwind CSS: For fast and responsive UI styling.
- Axios: Handles API requests to the backend.
- Socket.io-client: For real-time WebSocket communication with the server.

Backend

- Node.js: Handles asynchronous, non-blocking server-side code.

- Express.js: For setting up an API with efficient routing and middleware support.
- Socket.io: Enables real-time communication between the client and server.
- JWT (jsonwebtoken): Manages stateless authentication with token-based security.
- bcrypt: Hashes passwords securely before storing them in MongoDB.
- Mongoose: Provides a schema-based solution to model MongoDB data.

Database

- MongoDB: A NoSQL database suited for unstructured data like chat messages.

Testing WebSocket Functionality

- Open two browser windows and log into different accounts or join a group chat. Messages should appear in real-time.

Setup and Installation Instructions

1. Clone the Project Repository

First, clone the repository to your local machine using Git.

```
git clone <repository_url>
cd <repository_directory>
```

2. Install Node.js and npm

Ensure that you have Node.js and npm installed. You can install them using the following commands:

For Ubuntu:

```
sudo apt update
sudo apt install nodejs npm
```

For Windows:

- Download and install Node.js from nodejs.org.

Check the versions:

```
node -v
npm -v
```

3. Install Dependencies

You'll need to install dependencies separately for the API and client directories.

API:

Navigate to the API folder:

```
cd api
```

Install the dependencies:

```
npm install
```

Client:

Then navigate to the client folder:

```
cd ../client
npm install
```

4. Install cors and Other Required Packages

Ensure you have installed cors and any other missing dependencies for cross-origin requests.

In the api folder, install cors:

```
cd api
npm install cors
```

5. Set Up Environment Variables

In the api folder, create a .env file and add your environment variables for both MongoDB and JWT.

```
PORT=5000
MONGO_URI=<your_mongo_db_connection_string>
JWT_SECRET=<your_jwt_secret_key>
```

6. Run the Application

Start the API:

Navigate to the API directory:

```
cd api
```

Run the API using:

```
node index.js
```

or, if using nodemon:

```
nodemon index.js
```

Start the Client:

In another terminal, navigate to the client directory:

```
cd client
```

Run the client using:

```
npm run dev
```

This will start the development server and open the client side in your browser.

7. Testing the API and Client

Once both the API and client are running, you can test your application by navigating to http://localhost:3000 (for the client) and accessing your API endpoints at http://localhost:5000.

For testing the API, use Postman or similar tools to send requests to the following endpoints:

- **User Registration**:
  POST /api/v1/users/register
- **User Login**:
  POST /api/v1/users/login
- **Send Message**:
  POST /api/v1/messages/send
- **Create Group**:
  POST /api/v1/groups/create

This setup will allow you to run both the API and client simultaneously, using node index.js for the API and npm run dev for the client.

Future Considerations

- **Scaling**: Introduce a distributed database (e.g., MongoDB shards) and horizontally scale the WebSocket server using a message broker (e.g., Redis).
- **Security**: Implement rate-limiting, SSL for encrypted connections, and additional security policies such as Content Security Policy (CSP).
- **File Attachments**: Implement a file storage system (e.g., AWS S3) to handle attachments in chats.
- **AI powered chatbot** for user.
- **Video calling or audio calling** feature.
- Deployment into **Docker**.