Huddle Chat

-Bringing Us Closer

Huddle Chat System Design Document

1. Overview

Huddle Chat is a real-time messaging service designed to bring people together. It supports user registration, authentication, private and group chat, and real-time message updates. The app is built with **React.js** for the frontend and a **Python-based backend** with REST APIs. The system is designed to be scalable, efficient and user-friendly.

---

2. Architecture Overview

Huddle Chat follows a **client-server architecture**. The frontend is responsible for handling the user interface and communication with the backend, which processes data, manages user sessions, and stores chat information.

High-Level Architecture

1. **Frontend (React.js)**:
    - User interface
    - Handles authentication, chat rooms, and real-time message display
    - Communicates with the backend via REST APIs
2. **Backend (Python)**:
    - Handles user authentication, message storage, and API responses
    - Serves as the central hub for managing real-time messaging
3. **Database**:
    - **NoSQL database** stores user data, message history, and chat metadata.
    - For real-time features, a caching mechanism like **(MongoDB)**can be used.
4. **APIs**:
    - REST APIs for handling user registration, login, and chat management.
    - APIs are used for CRUD(Create,Read,Update,Delete) operations on chat messages, users.Each of these operations is typically mapped to HTTP methods like:
        - **POST** (for Create),
        - **GET** (for Read),
        - **PUT** or **PATCH** (for Update),
        - **DELETE** (for Delete)..
5. **WebSocket**:

    - For real-time chat updates, WebSocket connection mechanism is used to notify the client when a new message is received.
    - Using WebSockets would allow you to create a smoother, faster real-time messaging experience, as new messages can be instantly delivered without waiting for the client to request them

---

3. System Components

Frontend Components (React.js)

- `App.js`:
    - Root component of the application that manages routing and authentication status.
- **Login and Signup Pages**:
    - Handle user authentication and registration. Use secure forms to send login/signup data to the backend.
- **Chat Window**:
    - Displays chat history and allows users to send/receive messages.
    - Communicates with the backend to fetch messages and send new messages in real time.
- **Context API**:
    - In our Huddle Chat app, the **Context API** in React can be used to manage the **authentication state** globally, meaning it helps ensure that the user's authentication status (whether they are logged in, their user info, etc.) is easily accessible from any component in your app.

Backend Components (Python)

- **Authentication Service**:
    - Manages user sessions, registration, and login/logout.
    - Stores user credentials securely using password hashing (e.g., bcrypt).
- **Chat Service**:
    - Manages chat room creation, sending, and receiving messages.
    - Stores message history and allows message retrieval.
- **API Endpoints**:
    - `POST /register`: Creates a new user.
    - `POST /login`: Authenticates a user and returns a session token.
    - `GET /messages`: Fetches chat history for a given user or group.
    - `POST /messages`: Sends a new message in a chat.

Database

- **User Table**:
    - Stores user information such as email, password hash, and profile data.

- **Chat Table**:
    - Stores message data, including message content, sender, timestamp, and chat room ID.
    - In MongoDB, instead of a table, We'll have a **collection** for users.
    - Each user would be stored as a document (in JSON format) inside the `users` collection.

---

4. Data Flow

1. **User Registration**:
    - The user enters their email and password on the frontend.
    - The frontend sends a `POST` request to the backend's `/register` endpoint.
    - The backend hashes the password and stores user details in the database.
2. **User Login**:
    - The user submits login credentials on the frontend.
    - The frontend sends a `POST` request to `/login`.
    - The backend verifies credentials and returns a session token or error.
3. **Real-Time Messaging**:
    - When a user sends a message, the frontend sends a `POST` request to `/messages`.
    - The backend stores the message and notifies other users in the chat room using WebSocket or long-polling.
    - The frontend updates the chat window in real time as new messages arrive.

---

5. Technology Stack

5.1 Frontend:

- **React**: A component-based library used to build the user interface. Chosen for its flexibility, reusability of components, and community support.
- **Context API**: Manages global state like authentication status.
- **CSS Modules**: Used for scoped styling to avoid global CSS conflicts.

5.2 Backend:

- **Node.js**: The runtime environment for JavaScript on the server. Chosen for its non-blocking I/O model and ability to handle concurrent requests efficiently.
- **Express.js**: A minimalist web framework for Node.js. It simplifies the creation of REST APIs.
- **MongoDB**: A NoSQL database used to store user information, messages, and group chats. It's chosen for scalability and flexibility in handling unstructured data.

- **JWT (JSON Web Token)**: Used for user authentication. Chosen for its stateless nature, allowing secure transfer of information between parties.

5.3 Libraries and Dependencies:

- **bcryptjs**: Used for hashing passwords securely before saving them to the database.
- **jsonwebtoken**: For generating and verifying JWT tokens used for authenticating users.
- **mongoose**: An ORM (Object-Relational Mapping) library for MongoDB that simplifies interactions with the database.
- **dotenv**: Used to load environment variables from a `.env` file.
- **nodemon**: A development tool that automatically restarts the server on code changes.
- **cors**: Enables Cross-Origin Resource Sharing, which is necessary when the frontend and backend are hosted on different servers.

---

6. Dependencies and Libraries

Frontend (React.js)

- `next`: The core library for server-side rendered React applications.
- `react`: To build interactive UIs.
- `axios` **or** `fetch`: For making HTTP requests to the backend.
- `WebSocket`: For real-time messaging functionality.
- `styled-components` **or** `CSS Modules`: For styling the components.

Backend (Python)

- `bcrypt`: For secure password hashing.
- `Mongoose`: An ODM (Object Data Modeling) library for managing database interactions with MongoDB.
- `JWT (PyJWT)`: For handling token-based authentication
- .`API` is a **RESTful API** built using **Node.js** with **Express.js** as the web framework.

Why these Dependencies Were Chosen:

- **React.js**: Chosen for its server-side rendering capabilities, which improves performance and SEO for a chat app.
- **Flask/FastAPI**: These Python frameworks are lightweight, easy to use, and scalable for RESTful APIs, making them ideal for your backend.
- **MongoDB :** Offers flexibility with dynamic schemas, scales easily to handle large user traffic, provides high performance for real-time messaging, and stores data in a document-oriented format that's ideal for complex chat structures. Additionally, it integrates well with REST APIs and ensures high availability.

- **bcrypt**: A widely used library for securely hashing and managing passwords, ensuring user security.
- **WebSocket**: Necessary for providing real-time message updates, a core feature of chat applications.

---

7. Setup Instructions

Setup and Run Instructions for Huddle Chat Application

1. Frontend Setup (React.js + Next.js)

Clone the Repository:

On the target device, follow these steps to clone the frontend project:

```
git clone <frontend-repo-url>
```

```
cd <frontend-directory>
```

Install Dependencies:

Install the necessary dependencies using `npm`, which are listed in the `package.json` file:

```
npm install
npm i
```

Set Up Environment Variables:

Create a `.env` file in the root directory of the frontend project and add the necessary environment variables, such as the backend API URL and WebSocket URL. Example:

```
NEXT_PUBLIC_API_URL=http://localhost:3000/api
```

Run the Frontend:

To start the frontend in development mode, use:

```
npm start
```

The application should now be accessible at `http://localhost:3000`.

---

2. Backend Setup (Python + MongoDB)

Clone the Repository:

On the target device, follow these steps to clone the backend repository:

```
git clone <backend-repo-url>
```

```
cd <backend-directory>
```

Install Dependencies:

Install the backend dependencies listed in the `requirements.txt` file:

```
pip install -r requirements.txt
```

Set Up Environment Variables:

Create a `.env` file in the backend project's root directory and define variables like the MongoDB URI, secret keys, and other necessary settings.

Example:

```
MONGO_URI=mongodb://localhost:27017/chatapp
SECRET_KEY=your_secret_key
```

Set Up MongoDB:

Ensure MongoDB is installed and running on the target device. You can download it from MongoDB's official site.

Start the MongoDB service:

```
mongod
```

Run the Backend:

Start the backend service in backend directory

```
npm run dev
```

By default, the backend should be accessible at `http://localhost:5000`.

---

3. Final Configuration (Frontend and Backend)

Once both the frontend and backend are running on their respective ports (`localhost:3000` for the frontend and `localhost:5000` for the backend), ensure the frontend's API calls are properly configured to point to the backend in the `.env` file.

---

4. Testing the Application

Access the App:

Open a browser on the target device and navigate to `http://localhost:3000` to access the frontend.

Try Logging In and Sending Messages:

Test the following functionalities to ensure everything works as expected:

- User registration
- User login
- Sending and receiving messages
- Real-time chat functionalities

---

5. Troubleshooting Tips

Check MongoDB Connection:

If the backend fails to connect to MongoDB, verify that:

- The MongoDB service is running.
- The MongoDB URI in the `.env` file is correct.

API Communication:

Ensure that the frontend's `.env` file has the correct backend API URL (`NEXT_PUBLIC_API_URL`). Also, verify that CORS (Cross-Origin Resource Sharing) is correctly configured on the backend, especially if you are deploying across different domains.

8. Future Considerations

- **Security:** Implement end-to-end encryption and robust authentication; perform regular security audits.
- **User Experience:** Gather feedback, personalise experiences, and ensure a clean interface.
- **Features:** Add file sharing, voice/video calls, and app integrations.
- **Analytics:** Track user behaviour and feature usage for improvements.
- **Maintenance:** Regularly update the app and provide customer support.