FullStack ChatBOT Documentation

Overview:
The FullStack ChatBOT is a web-based application that allows users to register, login, and interact with a chatbot in real-time. It provides a seamless user experience with a React-based frontend and an Express.js backend. (*Assuming it uses AWS services for hosting, scalability, and reliability*)

Features:
- User registration and login (Token-based authentication for secure communication).
- Real-time chatbot interaction.
- Scalable architecture for handling varying traffic loads.

Prerequisites:
1. Node.js and npm installed on your machine.
2. PostgreSQL database installed and running locally.
3. Git installed for cloning the project repository.

Installation and Setup:
Here are the deployment instructions for running the FullStack ChatBOT application on localhost:

1. Clone the project repository from GitHub

2. Run the Backend Server:

```
cd backend
npm run server
```

   This will start the Express.js server on port 8080 by default.

3. Run the Frontend React App:

```
cd frontend
npm start
```

4. Database Setup:

   Create a PostgreSQL database named chatbot_db.
   Run the database migrations to create the necessary tables:

```
npm run migrate
```

Or :

Create new table using

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  password VARCHAR(255) NOT NULL,
  tokens INT NOT NULL DEFAULT 1000)
```

5. Testing

For Backend -

cd backend
npm test

For frontend -

cd frontend/src
npm test

APIs Documentation

Register User
Registers a new user with the system.

- URL: /register
- Method: POST
- Request Body:
  - username (string, required): The username of the user.
  - password (string, required): The password of the user.
- Response:
  - Status: 201 Created
  - Body: "User registered"
- Error Responses:
  - 400 Bad Request: If the request body is missing or invalid.
  - 500 Internal Server Error: If there's a server-side error.

Login User
Logs in an existing user.

- URL: /login
- Method: POST
- Request Body:
  - username (string, required): The username of the user.
  - password (string, required): The password of the user.
- Response:
  - Status: 200 OK
  - Body: JWT token for authentication
- Error Responses:
  - 401 Unauthorized: If the credentials are invalid.
  - 500 Internal Server Error: If there's a server-side error.

Send Message
Sends a message to the chatbot and receives a response.

- URL: /messages
- Method: POST
- Authentication: Required (JWT token)
- Request Body:
  - text (string, required): The message text to send to the chatbot.
- Response:

Status: 200 OK
Body:json

```
{
  "text": *Response text from the chatbot*,
  "usage": {
    "inputTokens": 10,
    "outputTokens": 5,
    "availableTokens": 985
  }
}
```

- Error Responses:
  - 401 Unauthorized: If the JWT token is missing or invalid.
  - 403 Forbidden: If the user's token limit is reached.
  - 500 Internal Server Error: If there's a server-side error.

WebSocket Endpoint
WebSocket endpoint for real-time chat functionality.

- Authentication: Required (JWT token)
- Protocol:
  Client sends messages in JSON format:json

```
{
  "type": "chat",
  "text": "Message text"
}
```

  Server responds with messages in JSON format:json

```
{
  "type": "chat",
  "text": "Response text from the server"
}
```

Authentication:

Some endpoints like /messages and for web socket require authentication using JWT tokens. Include the JWT token in the Authorization header of your requests in the format: Bearer <token>