Chatbot using WebSockets

Description

This project focuses on creating an interactive web application that allows real-time communication between users and a chatbot. Users can sign up, log in and engage in conversations with an AI-powered chatbot. The application's goal is to practice and consolidate JavaScript, DOM manipulation, WebSockets and client-side storage, building a clean, responsive, and user-oriented chat interface.

Goal

- Using WebSockets for real-time bidirectional communication between client and server.
- Using the Google Gemini API (https://ai.google.dev/gemini-api/docs) to generate intelligent chatbot responses.
- Consolidating JavaScript skills by implementing authentication, dynamic DOM updates, and chat functionality inspired by modern messaging applications.
- Developing a web application focused on interactivity, performance and best practices.

Minimum Features

- Sign up and log in pages – user authentication with data stored in localStorage (username, email, password etc.).
- Real-time Chat Interface – a clean, scrollable conversation display featuring aligned messages and instant, live communication powered by WebSockets.
- Client-specific conversation history – stores each user's messages per conversation on the server for context-aware replies; server history clears on disconnect, but localStorage preserves it across sessions.
- Start new conversation functionality – prompts the user to enter a conversation name and creates a new conversation, allowing messages to be sent and tracked separately from existing conversations.
- User profile page – displays editable username, total messages, total conversations, and allows viewing conversations saved in localStorage.

Other Features (Optional)

- Message history – display conversation history with timestamps for each message.
- Search messages – allow users to search for specific messages within a conversation.
- Export/Download chat – allow exporting a conversation as a text or JSON file.
- Installable PWA – allow the app to be added to a mobile device like a native app.

Software Principles

- Separation of concerns – WebSocket logic, DOM manipulation, and styles should be separated into different modules.
- Reusability & modularity – functions for sending, receiving, and displaying messages should be reusable.

- Error handling – properly handle WebSocket errors, failed API requests, and invalid user input.
- Best Practices – apply ES6+, async/await, const/let, arrow functions, and clean, readable code.

Useful tips – creating a Node.js server for WebSockets

1. Install Node.js v18+ – required for the Google Gemini SDK. Download it from https://nodejs.org/.
2. Initialize project with the command ***npm init -y*** (it creates a package.json file to manage project dependencies)
3. Install required packages with ***npm install ws @google/genai***
    - ws: WebSocket server for real-time communication
    - @google/genai: SDK to interact with Google Gemini API
4. Write your server: create server.js to handle WebSocket connections, client messages, and AI responses.
5. Start the server using ***node server.js.*** This launches the WebSocket server, ready to accept client connections.
6. Connect the client: browser connects via WebSocket to send messages and receive AI-generated replies in real-time.