Bylexa Project: Voice-Controlled Automation

# Brief of Idea

Bylexa (G.O.A.T. - Greatest Of Automated Tasks) is a voice-controlled automation system that enables users to perform complex actions on their local machines via voice commands. It simplifies interactions with applications, websites, and IoT devices by allowing the user to send commands through a frontend app or mobile device. The backend interprets these commands using a language model (Gemini API) and executes the tasks locally on the user's PC through a Python-based client agent.

# Deliverables of the Solution

- A Python client agent installed on the user's machine to execute OS-level commands.  
- WebSocket-based real-time communication between the frontend and the backend server.  
- User authentication and token management using JWT for secure communication.  
- Interpretation of natural language commands using Gemini API.  
- A frontend app for users to log in, send commands, and view execution results.  
- Ability to open local applications and perform specified tasks (e.g., opening a browser and navigating to a URL).  
- CLI-based Python module for users to easily set up and configure the system.

# Tech Stack Used

Frontend:  
- React (JavaScript)  
- Tailwind CSS (for styling)  
Backend:  
- Node.js (Express.js)  
- WebSocket server (ws)  
Python:  
- websockets for real-time communication  
- PyJWT for token management  
- Requests for making API calls  
- subprocess for executing OS-level commands

# Architecture of the Project

The Bylexa project follows a server-client architecture. The frontend app or mobile device sends voice commands to the backend server, which interprets the commands using the Gemini language model. The interpreted commands are sent to the user's machine via WebSocket connection. The Python client agent running on the user's machine receives the commands and executes them locally using subprocess.

# Workflow and Flow of Data

1. The user logs in to the frontend app, which authenticates the user using JWT.  
2. The user sends a voice command through the app (e.g., 'open Spotify').  
3. The backend server receives the command and uses the Gemini API to interpret the action, application, and task.  
4. The server sends the interpreted command to the Python client agent running on the user's machine.  
5. The Python client agent executes the command (e.g., opening Spotify) and reports the result back to the server.  
6. The server sends the result back to the frontend app, where the user can see the execution status.

# Future Scopes of the Project

- Integration with additional IoT devices and platforms.  
- Extending support for more natural language models beyond Gemini.  
- Enabling real-time monitoring and analytics for executed tasks.  
- Shared computing power to distribute processing tasks across multiple systems.  
- Simultaneous remote communication with multiple backend servers to allow for parallel task execution.