Gemini Powered Pizza Chatbot - PizzaBahn

Group Members: Srihari Ananthan (100001648), Srinivasalu Medeboyina, Berke Keskinler, Anmol Sharma

Introduction

This project implements an AI-powered conversational agent to take pizza orders. The bot collects customer preferences, handles special dietary requests (halal, vegan, allergies), and captures a delivery address. The final order is structured in a machine-readable format (JSON). By leveraging the Gemini API, the bot understands natural language to provide a seamless and personalized customer experience, moving beyond traditional, static ordering systems.

Stack Used

- **Generative AI:** Google Gemini, using the gemini-2.5-flash model for conversational capabilities.
- **Backend:** Python 3.x with the flask framework, creating a RESTful API.
- **Frontend:** Standard web technologies, including HTML (index.html), CSS (style.css), and JavaScript (script.js).
- **Libraries and Dependencies:** Key Python libraries include Flask, Flask-CORS, and google-generativeai. The frontend uses window.SpeechRecognition for speech-to-text.
- **System Design:** The project's conversational flow and state management were planned using a Mermaid flowchart and a PDF flowchart.
- **Data Storage:** A sqlite3 database is used to store order data, which is saved as JSON. The menu data is parsed from a docx file.

Methodology

The chatbot's architecture is based on a client-server model. The frontend, built with HTML, CSS, and JavaScript, provides the user interface. It sends user messages to the Python backend via API calls and displays the chatbot's responses. The script.js file handles all client-side logic, including conversation history and speech-to-text functionality.

The Python backend, a Flask application (main2.py), is the core of the system. It uses an OrderState object for each user session to maintain the conversational flow as defined in the Mermaid flowchart. When a user sends a message, the chatbot.process_conversation function determines the current step in the OrderState and the user's input to decide the next action. For example, if the user specifies a dietary need, the system transitions to the dietary_needs state, and the menu is filtered accordingly. The menu data, parsed from the AI_Pizza_Menu_With_Allergy_Halal_Updated.docx file, is used to present relevant options. Confirmed orders are saved as JSON objects in a sqlite3 database. The backend also includes API endpoints for tasks like menu retrieval and session resets, ensuring a robust interaction with the frontend.

Results

The developed application provides a functional and conversational pizza ordering experience. The chatbot can:

- **Greet users** and initiate the ordering process.
- **Identify and apply dietary preferences** like Vegetarian, Vegan, or Halal.
- **Filter and display menu items** from the provided AI_Pizza_Menu_With_Allergy_Halal_Updated.docx file based on these dietary needs.
- **Guide the user** through selecting pizzas, extras, and providing delivery details.
- **Provide a clear order summary** for confirmation.
- **Save the final order** in a structured JSON format.

A sample dialogue demonstrates the flow:

- **User:** "I'd like to order a pizza."
- **Chatbot:** "Hello! What are your dietary needs? For example, are you looking for Halal, Vegan, or Vegetarian options?" (Reflecting the transition from A to B in the flowchart).
- **User:** "I'm looking for a vegan pizza."
- **Chatbot:** (Shows the Vegan menu options, which corresponds to the D state) "We have some great vegan options: Vegan Delight, Spicy Vegan Inferno, Vegan Pesto Paradise, and BBQ Jackfruit."

Conclusion

The Gemini-powered pizza chatbot project successfully integrates modern conversational AI with a practical e-commerce application. The system design, guided by a clear flowchart, enables a stateful and intuitive user experience. The use of the Gemini API allows the chatbot to handle natural language queries, making the interaction feel more human-like. This project demonstrates a viable and engaging alternative to traditional form-based online ordering.

Known Issues:

1. **Hallucinations:** The Gemini API may occasionally generate incorrect or unexpected responses, especially with ambiguous user input.
2. **Speech-to-Text Accuracy:** The window.SpeechRecognition API's accuracy can be affected by factors like background noise and accents, potentially leading to misinterpretations.
3. **Scalability:** The current in-memory session management is not suitable for high-traffic environments. A more robust solution would require a distributed cache or database for session states.
4. **Error Handling:** While some error handling exists, a more comprehensive approach would be needed to manage potential failures like API downtime or database connection issues gracefully.