AutoDine Official Documentation

Table of Contents

Overview

AutoDine is a chatbot-driven food ordering system that enables customers to place orders using speech. The system supports real-time order processing, checks for item availability, and handles complex customizations through conversational AI.

Key features:

- Voice-based ordering system
- AI-driven natural language processing for order extraction
- Inventory check and order placement backend
- Customizable menu and order history
- Speech recognition and text-to-speech (TTS) integration for conversational interaction

Architecture

The project consists of a **frontend** for interacting with customers and a **backend** for handling the core logic such as checking item feasibility, placing orders, and retrieving the menu. The frontend integrates a language model API and handles user interaction via speech, while the backend processes orders and communicates with the frontend.

System Components:

1. **Frontend**: Responsible for capturing user speech, generating requests to the language model, and interacting with the backend.
2. **Backend**: Manages the menu, handles feasibility checks, processes orders, and stores order history.

High-Level Architecture:

```
Frontend (Speech -> Text -> API Requests)
  |
```

```
  |----> Language Model API (Processes orders from text)
  |
Backend (Handles order feasibility and processing)
```

Tech Stack

- **Frontend**: Speech-to-Text and Text-to-Speech APIs, Python, OpenAI API and django for interaction
- **Backend**: Django (Python), OpenAI API for language model processing, `requests` library for API communication
- **Database**: SQLite3

Installation Guide

Frontend Installation

1. **Clone the frontend repository**:

   ```
   git clone https://github.com/yourusername/autodine-frontend.git
   cd autodine-frontend
   ```

2. **Install dependencies**:

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

3. **Run the frontend application**:

   ```
   python main.py
   ```

4. **Configure OpenAI API key** in the `llm.py` file:

   ```
   client = openai.OpenAI(api_key="your_openai_api_key")
   ```

5. **Configure the backend server URL** in the `llm.py` file:

   ```
   SERVER_URL = "http://127.0.0.1:8000"
   ```

Backend Installation

1. **Clone the backend repository**:

   ```
   git clone https://github.com/yourusername/autodine-backend.git
   cd autodine-backend
   ```

2. **Install dependencies**:

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

3. **Run the backend server**:

   ```
   python app.py
   ```

The backend will be available at `http://127.0.0.1:8000`.

API Documentation

Backend Endpoints

1. **Check Feasibility**
    - **URL**: `/check_feasible_items/`
    - **Method**: `POST`
    - **Description**: Checks if requested items are available in the inventory.
    - **Request Body**:

```
{
  "items": {
    "Burger": {},
    "Fries": {}
  }
}
```

    - **Response**:

```
{
  "feasible": true,
  "unavailable_items": []
}
```

2. **Place Order**
    - **URL**: `/order_items/`
    - **Method**: `POST`
    - **Description**: Places an order and updates the inventory.
    - **Request Body**:

```
{
  "items": {
    "Burger": {},
    "Fries": {}
  },
  "tip": 3.50
}
```

    - **Response**:

```
{
  "status": "success",
  "order_id": 1
}
```

3. **Retrieve Menu**

- **URL**: `/menu`
- **Method**: `GET`
- **Description**: Retrieves the available menu.
- **Response**:

```
{
  "Burger": 9,
  "Fries": 19
}
```

4. **Order History**
   - **URL**: `/order_history`
   - **Method**: `GET`
   - **Description**: Retrieves the order history.
   - **Response**:

```
[
  {
    "items": {
      "Burger": {},
      "Fries": {}
    },
    "tip": 3.50,
    "status": "completed"
  }
]
```

Frontend-Backend Communication

The frontend communicates with the backend via POST and GET requests, sending structured JSON objects as described in the API documentation. The backend responds with order feasibility, processing confirmations, and inventory information.

Speech Recognition and Language Model Integration

AutoDine uses **speech-to-text (STT)** and **text-to-speech (TTS)** capabilities for a seamless customer experience. The following modules are used:

- **Speech-to-Text**: Captures user orders and converts them into text for processing by the language model.
- **Text-to-Speech**: Converts the response from the language model into speech for the customer.

Real-time Speech Handling:

- **AudioToTextRecorder**: Captures user speech and converts it to text.
- **Pyttsx3**: Handles text-to-speech for responses.

Language Model (OpenAI):

The system interacts with the OpenAI API to process and structure customer orders based on predefined prompts. The model generates responses in JSON format, which the backend processes.

Usage Instructions

1. **Start the backend server**:

   ```
   python app.py
   ```

2. **Start the frontend speech interaction**:

   ```
   python main.py
   ```

3. The system will display the menu to the user. The user speaks their order, which is processed through the language model, checked for feasibility, and placed.

Testing

Testing the Backend

1. **Unit Tests**:
   - Write unit tests for each backend endpoint (e.g., `/check_feasible_items/`, `/order_items/`).
   - Use the `unittest` library to mock API requests and responses.
2. **Manual Testing**:
   - Use a tool like **Postman** or **cURL** to manually test backend endpoints.

Testing the Frontend

1. **Speech Interaction Testing**:
   - Simulate speech inputs to test the functionality of the speech-to-text and text-to-speech modules.
   - Ensure that the language model properly handles the flow from user speech to structured order.

Contributing

We welcome contributions! Please submit a pull request for any new features, bug fixes, or documentation improvements.

Future Enhancements

- **User Authentication**: Add user accounts and authentication to keep track of individual order history.
- **Real-time Updates**: Use websockets for real-time inventory and order updates.
- **Admin Dashboard**: Develop an admin interface to manage orders, inventory, and customer feedback.

- **Multilingual Support**: Add support for multiple languages for a more diverse customer base.
