Project Documentation

● Overview

This project demonstrates the development of a Sales Chatbot application with integrated product management. It uses a Flask backend to handle API requests and manage data through an SQLite database. The frontend interacts with the backend via JavaScript to fetch and display product data dynamically.

Objectives

- ◆ Create a Flask-based backend for managing product data.
- ◆ Develop a user-friendly frontend to interact with the chatbot and display product information.
- ◆ Implement database operations to store, retrieve, and display mock product data.

● Technologies Used

Backend: Python, Flask, Flask-SQLAlchemy

Frontend: HTML, CSS, JavaScript

Database: SQLite

Other Tools: Flask-CORS, Fetch API

● Project Setup

Prerequisites

Python (v3.9 or later)

Flask and Flask-SQLAlchemy libraries

Browser for testing the frontend

● Code Explanation

Backend

app.py

Configures the Flask app and database.

Defines the Product model with attributes: id, name, price, category, and description.

Sets up routes to fetch data from the database:

```
@app.route('/api/products', methods=['GET'])
def get_products():
    products = Product.query.all()
    return jsonify([product.__dict__ for product in products if '_sa_instance_state' not
in product.__dict__])
```

populate_db.py

Populates the database with mock product data.

Generates 100 sample entries with dynamic pricing and categories.

Frontend

index.html

Defines the chatbot interface and a container to display products dynamically.

Includes the script.js file for handling JavaScript interactions.

script.js

Fetches product data from the Flask backend using the Fetch API:

```
fetch("http://127.0.0.1:5000/api/products")
    .then(response => response.json())
    .then(data => displayProducts(data))
    .catch(error => console.error("Error fetching products:", error));
```

Dynamically renders product data inside the #product-container div.

● Learnings and Methodology

Objectives Achieved:

Successfully implemented a Flask backend to handle API requests and database operations.

Created a dynamic frontend to fetch and display data from the backend.

Designed a system to manage product data efficiently.

Challenges and Solutions:

Challenge: Populating the database with realistic data. **Solution**: Used a script to generate mock data dynamically.

Challenge: Fetching and displaying data dynamically. **Solution**: Utilized the Fetch API and DOM manipulation to render data in real time.

Future Enhancements:

Implement chatbot logic to handle customer queries intelligently.

Add authentication and user management.

Deploy the application using a production-grade WSGI server.
