Project Title: AI Agent for SQL Query Execution

---

1. Project Overview

The “AI Agent for SQL Query Execution” is a cutting-edge solution that bridges the gap between natural language and SQL. This intelligent assistant empowers non-technical users to interact with a MySQL database effortlessly—by simply asking questions in plain English. It converts these queries into optimized, error-free SQL statements, executes them against a live database, and presents the results in an easy-to-read format.

---

2. Objective

The primary objective of this project is to democratize data access by:

- Allowing users to retrieve insights without prior SQL knowledge.
- Automating query generation using advanced AI and natural language processing.
- Providing a robust backend that executes SQL queries and returns results efficiently.

---

3. Technologies and Tools

- **Programming Language:** Python 3.8+
- **Backend Framework:** FastAPI
- **Database:** MySQL (configured via MySQL Workbench)
- **NLP and AI:** Gemini API and LangChain for natural language processing and SQL generation.
- **Frontend:** HTML, CSS, and Bootstrap (rendered via FastAPI templates)
- **Configuration Management:** dotenv (.env files)
- **Additional Services:**
    - **ai_service.py:** Handles communication with the Gemini API.
    - **sql_service.py:** Manages SQL cursor operations and query execution.

---

4. System Requirements

- **Operating System:** Windows, Linux, or macOS
- **Software:** Python 3.8 or above, MySQL Server, MySQL Workbench
- **Network:** Active internet connection (for accessing the Gemini API)
- **Tools:** Git for version control, Virtual Environment for dependency isolation

---

5. Setup Instructions

a. Environment Setup

1. **Clone the Repository:**

```
git clone https://github.com/yourusername/SQL_AI_AGENT.git
cd SQL_AI_AGENT
```

2. **Create and Activate Virtual Environment:**

```
python -m venv venv
venv\Scripts\activate
```

3. **Install Dependencies:**

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

4. **Configure Environment Variables:**

   - Create a `.env` file in the project root with:

```
DB_USER=root
DB_PASSWORD=root@123
DB_HOST=localhost
DB_PORT=3306
DB_NAME=sales_schema
GEMINI_API_KEY=your_gemini_api_key
```

b. Database Setup

Run the following SQL commands using MySQL Workbench to set up the database:

```
CREATE DATABASE sales_schema;
USE sales_schema;

CREATE TABLE customers (
    customer_id INT PRIMARY KEY NOT NULL,
    customer_name VARCHAR(100),
    gender CHAR(1),
    age INT,
    city VARCHAR(100),
    join_date DATE
);

CREATE TABLE products (
    product_id INT PRIMARY KEY NOT NULL,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10,2)
);

CREATE TABLE sales (
    sale_id INT PRIMARY KEY NOT NULL,
    customer_id INT,
```

```
    product_id INT,
    sale_date DATE,
    sale_amount DECIMAL(10,2),
    quantity_sold INT,
    region VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

Populate the tables with sample data as needed.

---

6. Detailed Project Structure

```
SQL AI AGENT/
├── app/
│   ├── services/
│   │   ├── ai_service.py       # Handles communication with the AI (Gemini) API
│   │   └── sql_service.py      # Manages SQL cursor operations and query
execution
│   ├── database.py             # Establishes and manages database connections
│   ├── main.py                 # Entry point for running the FastAPI application
│   └── models.py               # Contains data models (Pydantic or ORM models)
├── static/
│   ├── css/
│   │   └── styles.css          # Styles for the frontend
│   └── js/
│       └── script.js           # Client-side JavaScript for dynamic behavior
├── templates/
│   └── index.html              # Main HTML template for rendering the UI
├── .env                        # Environment variables (e.g., DB credentials, API
keys)
├── readme.md                   # Basic project documentation
└── requirements.txt            # Python dependencies required for the project
```

---

7. Core Components

- **Natural Language Processing Module:**
  Utilizes Gemini API to convert plain English queries into valid SQL. The module (`ai_service.py`) handles communication with the external AI API to ensure accurate translation.

- **SQL Execution Engine:**
  The `sql_service.py` safely executes SQL commands using MySQL connectors, managing cursors and ensuring efficient resource handling.

- **Database Connectivity:**
  Handled by the `database.py` module, which establishes a secure and efficient connection with the MySQL database using credentials from the `.env` file.

- **API and UI Integration:**
  The FastAPI application (`main.py`) receives user inputs via a web UI (`index.html` template), processes them through backend services, and returns results for display.

---

8. Usage Guide

a. Running the Backend:

- Start the FastAPI server:

```
python -m main
```

- The server will run at `http://localhost:8000`.

b. Interaction Workflow:

1. **User Query Input:** Enter your natural language question in the UI.
2. **API Processing:** The backend receives and processes the query, converting it into SQL.
3. **SQL Execution:** The SQL is executed against the MySQL database.
4. **Result Presentation:** Results are returned to the UI in a structured, tabular format.

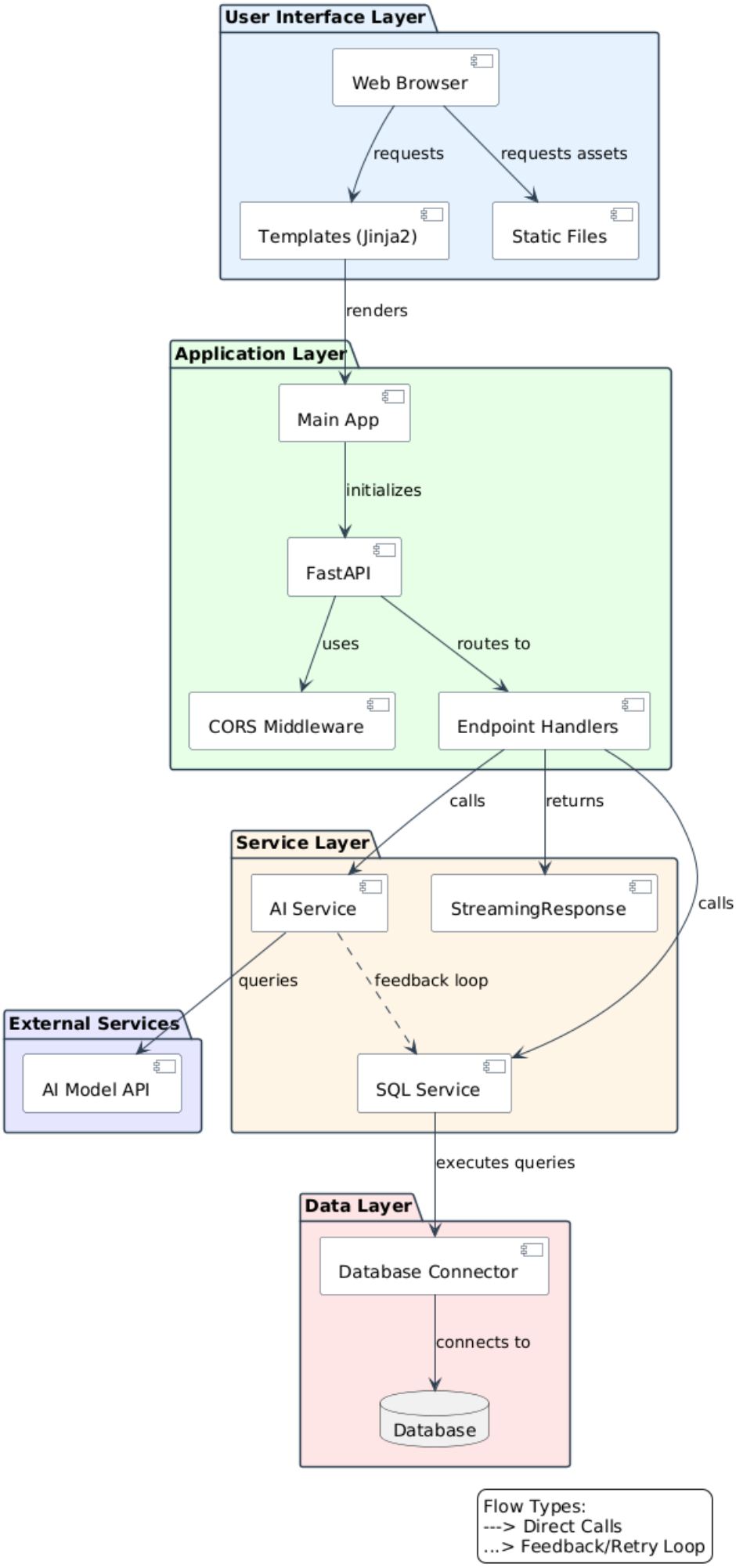
---

9. Sample Natural Language Queries

- List total sales per product.
- Find customers who purchased in more than one category.
- Show all sales recorded in January 2023.
- Retrieve names of customers who bought Electronics.
- What are the top 3 cities by number of customers?
- Identify products never sold.
- List customers who joined in 2022 and made at least one purchase.

---

10. Architecture Diagram


User Interface Layer
Web Browser
requests
requests assets
Templates (Jinja2)
Static Files
renders
Application Layer
Main App
initializes
FastAPI
uses
routes to
CORS Middleware
Endpoint Handlers
calls
returns
calls
Service Layer
AI Service
StreamingResponse
queries
feedback loop
External Services
AI Model API
SQL Service
executes queries
Data Layer
Database Connector
connects to
Database
Flow Types:
---> Direct Calls
...> Feedback/Retry Loop

- **Layered Architecture:** The diagram represents a multi-tiered architecture separating concerns into distinct layers such as UI, Application, Service, and Data layers, ensuring modularity and scalability.
- **User Interface Layer:** This layer handles all user interactions through a web browser, rendering views with Jinja2 templates and serving static assets like CSS and JavaScript.
- **Application Layer:** Powered by FastAPI, this layer manages API endpoints and middleware (e.g., CORS) to process incoming requests and route them to appropriate services.
- **Service Layer:** The core business logic resides here with dedicated services for AI operations (communicating with the external AI Model API) and SQL query execution, facilitating seamless interaction between user queries and database operations.
- **Data Layer:** Responsible for database connectivity, this layer uses a dedicated connector to manage query execution and ensure data integrity.
- **External Integration:** The system integrates with external services (like the AI Model API) to enhance its natural language processing capabilities, making it a powerful tool for converting natural language to SQL.
- **Feedback Loop:** The diagram also indicates a feedback mechanism between the AI and SQL services, which helps refine query generation and execution, ensuring accuracy and reliability in results.