Team: Database Demons
Members: Julian Robinson, Aleesha Exantus

Relational Schema

Customer (<u>CustomerID</u>, Name, Email)
Vendor(<u>VendorID</u>, Name, ProfileInfo)
Market(<u>MarketID,</u> Date, Location, WeatherCondition, isSpecialEvent)
Product(<u>ProductID</u>, VendorID (FK), Name,Current_Price)
Order(<u>OrderID</u>, CustomerID (FK), MarketID (FK), OrderDate, Status)
OrderItem(<u>ItemNo</u>, OrderID (FK), ProductID (FK), PurchasePrice, Qty)
VendorMarket(<u>VendorID (FK)</u>, <u>MarketID (FK)</u>)

Platform, Database, and Languages

Database: PostgreSQL

- Ensures data integrity for complex relational structures (Customers, Vendors, Orders)
- Efficiently handles multi table joins and aggregate queries required for sales reporting
- Provides seamless integration with Python based data analysis tools

Backend: Python with Flask

- Offers a lightweight, flexible architecture that speeds up development
- Provides a native environment to execute the predictive model (advanced function)
- Simplifies the creation of an API to connect the database to the user interface

Frontend: HTML/CSS and JavaScript (Bootstrap)

- Enables a responsive, mobile friendly design for use at physical market locations.
- Utilizes prebuilt components to ensure a professional look without over complicating the codebase
- Facilitates real time interaction for searching and filtering product listings

Advanced Function Logic: Pandas and Scikit-learn

- Pandas: Efficiently cleans and structures historical sales data for analysis
- Scikit-learn: Implements the linear regression model used to generate demand forecasts
- Provides the technical depth required to transform raw data into actionable vendor insights

Implementing Advanced Function

Data Aggregation via SQL:

- Develop complex queries to extract historical sales data by joining the OrderItems, Orders, and Products tables
- Group sales by MarketID and Category to identify volume trends over specific dates.

Feature Engineering:

- Incorporate external variables into the dataset, such as WeatherCondition (e.g., sunny vs. rainy) and IsSpecialEvent (e.g., holiday weekends) from the Market table
- Create a "Seasonality Index" to account for the peak harvest times of specific produce.

Model Selection and Training:

- Utilize the Scikit-learn library to implement a Linear Regression model
- Train the model using the synthetic historical data to establish a baseline relationship between market conditions and sales volume

Prediction Pipeline:

- Build a Python script within the Flask backend that passes "upcoming market" parameters (date, predicted weather) into the trained model
- Generate a PredictedDemand value for each product in the vendor's inventory

Vendor Dashboard Integration:

- Design a "Smart Restock" Ul component using Bootstrap that displays the predicted demand alongside current stock levels
- Implement a logic gate that flags items as "High Demand" or "Potential Overstock" to assist vendors in physical inventory preparation.

Optimization and Validation:

- Use Pandas to calculate the error margin between predicted demand and actual sales once a market concludes
- Refine the model coefficients throughout the project lifecycle to improve forecasting accuracy for the final demo.

Data Acquisition

• Web + Public Lists: We will gather real product names, categories, and pricing from local farmers market websites and the USDA food database.
• Manual Creation: For historical sales data, we'll develop mock records that mirror real world shopping patterns to ensure our algorithm works and generate many restock recommendations

Labor Division:

Backend & Database:

Responsibilities include

- setting up the PostgreSQL database
- writing the SQL queries for basic CRUD functions
- developing the Python API for the predictive forecasting model

Frontend & System Integration.

Responsibilities include

- building the web interface (Vendor Dashboard and Customer Storefront)
- handling form submissions
- ensuring the frontend correctly displays the smart restock recommendations.

Timeline

The project is divided into four main sprints:

Milestone 1 (Week 1-2): Database Foundation
• Build the PostgreSQL schema.
• Seed the database with USDA product data and synthetic sales history.

Milestone 2 (Week 3-4): Core Functionality
• Implement user login and product management (CRUD).
• Build the "Search and Filter" feature for customers.

Milestone 3 (Week 5-6): The Advanced Function
• Develop the Python script to analyze sales trends and weather.
• Integrate the "Smart Restock" assistant into the vendor dashboard.

Milestone 4 (Week 7): Final Testing and Demo Recording
• Perform E2E testing (Ordering,Sales, Forecasting)
• Record the 20 minute demo and
• finalize a 5-10 page report.