Mykola Doskoch
md1596

Inventory Management System Using Python and SQL

Introduction

Inventory management is the backbone of any retail or eCommerce business. It ensures that businesses are able to stock the right products in the right quantities to meet customer demand while keeping holding costs minimal. More often than not, small businesses struggle with effective inventory tracking due to limitations in resources and reliance on manual methods like spreadsheets. Such approaches are error-prone, time-intensive, and result in inefficiencies and lost revenue.

This project has been prepared with the objective of designing and implementing an easy-to-use yet effective Inventory Management System (IMS) for small businesses. Built on top of Python and SQL, this system aims at solving the common problems of inventory such as tracking stock levels, processing orders, and keeping inventory updated. The system was developed bearing in mind user friendliness, so even less technically savvy users can navigate the features easily. Additionally, the system has been enhanced to accept real-world orders the way they are exported from platforms such as Shopify, allowing seamless integration with online stores to manage inventory efficiently and streamline order processing.

This report provides an overview of the project's design, implementation, and evaluation, along with the challenges encountered and potential future improvements.

Project Scope and Design

The IMS is designed to serve a single business with inventory and order data modeled on formats used by popular eCommerce platforms like Shopify. The system allows users to upload initial inventory data in a CSV format and process orders in the same format they are exported from Shopify. This ensures seamless integration into existing workflows and eliminates the need for manual data entry.

Database Schema

The database comprises two main tables:

1. **Products Table**:
    - **Columns**: `product_id`, `name`, `price`, and `stock_level`.
    - **Primary Key**: `product_id`.
    - This table tracks the details of each product, including its current stock level.
2. **Orders Table**:
    - **Columns**: `order_id`, `order_date`, `customer_email`, `total_amount`, `status`, and `line_items`.
    - **Primary Key**: `order_id`.
    - **Line Items**: Stored as JSON, representing individual product details in each order.

The schema enforces data consistency, ensuring all orders reference valid products. Stock levels are automatically updated based on processed orders, maintaining real-time accuracy.

Data Flow

1. **Initial Inventory Upload**:
    - The user uploads a CSV file containing the details of the inventory. The system checks the format before inserting the records into the database. If the data is in the wrong format or lacks required fields, the system outputs an error message specifying what went wrong so that users can correct and re-upload the file.
2. **Order Processing**:
    - Users upload orders in a CSV format matching Shopify exports. The system parses these files, processes the line items, and updates stock levels accordingly.

The IMS follows a predefined data structure to ensure compatibility with existing eCommerce workflows.

System Implementation

Backend Development

The backend was built using Python with Flask for routing and SQLite for data storage. Flask is lightweight and makes it easy to handle routing and backend logic without the extra complexity of a larger framework like Django. SQLite was chosen for its simplicity and the convenience of using a local database instead of setting up a server-based system like MySQL or PostgreSQL.

Key Features:

- **Add Products**: Allows users to add new products to the inventory via a form or CSV upload.
- **Update and Delete Products**: Provides options to modify product details or remove products from the inventory.

- **Process Orders**: Handles bulk uploads of orders, parsing line items, and adjusting stock levels in real time.

Challenges: The major challenge was managing different data formats for orders. Shopify exports line items as JSON-like strings that need careful parsing and validation. This was solved by using Python's json library and adding strict data validation. Another challenge was designing the table to display different colors based on stock levels. I also added a sorting function to make it easier to organize the table and included a search bar for quickly finding specific items.

Frontend Design

Frontend: The user interface, built using HTML with Bootstrap, emphasizes ease of use. It contains:

- **Inventory Table**: Displays products in a scrollable, sortable table with real-time search functionality.
- **Order Table**: Lists uploaded orders with detailed views of individual line items.
- **Interactive Forms**: For adding, updating, and removing products.

To improve usability, the system includes validation messages, and file upload guidelines. This helps the user understand errors and fix them in real time.

Testing and Evaluation

Testing Approach

The system was rigorously tested using sample data files generated by chat GPT to verify:

1. **Inventory Management**:
    - Adding, updating, and deleting products.
    - Validating CSV uploads and ensuring format compliance.
2. **Order Processing**:
    - Parsing and storing order data accurately.
    - Updating stock levels based on order quantities.

Evaluation Metrics

1. **Accuracy**: Stock levels were verified after processing multiple orders, ensuring no discrepancies.
2. **Usability**: Feedback from my friends indicated the UI was intuitive and easy to navigate.
3. **Error Handling**: Uploading invalid files triggered appropriate error messages, preventing data corruption.

The system consistently met these goals, proving to be reliable and practical.

Challenges and Solutions

Technical Challenges

1. **Data Parsing**:
    - Shopify's export orders contain nested JSON structures for line items. Parsing these structures to extract relevant details was complex.
    - **Solution**: Implemented robust error handling and fallback logic to manage parsing errors.
2. **UI Improvements**:
    - The initial UI was minimal and functional but lacked user-friendliness.
    - **Solution**: Continuously improved the interface by adding features such as live search, sorting, and pop-ups.

Time Constraints

Since I worked on this project alone, I couldn't include advanced features like sales prediction models because of the limited time available. I also wasn't able to add an export feature to export the inventory data for bookkeeping.

Future Enhancements

1. **Sales Prediction Model**:
    - Add a machine learning feature to predict future sales using past data, helping with better stock planning and lowering storage costs.
2. **Multi-Store Support**:
    - Expand the system to manage inventory for multiple locations or online stores.
3. **Real-Time Synchronization**:
    - Enable real-time updates between the IMS and eCommerce platforms for smoother inventory management.
4. **Enhanced Data Visualization**:
    - Include charts and graphs to show inventory trends and sales performance.
5. **Export Feature**
    - Include an export button for downloading current inventory data for bookkeeping purposes.