POINT OF SALE SYSTEM

Campbell, Giarusso, Menke, Morisette

CS 2720 – SOFTWARE ENGINNERING Spring, 2016; Dr. J. Phillip East

Table of Contents

System Documentation

System-Level Overview

The Point-of-Sale System (POS) is a versatile, expandable, and powerful tool for retail businesses seeking to effectively manage their sales and inventory. It is a web-application powered by Python 3.5 and Flask, an extensive module that enables web-development through Python. It requires installation on a web server with an associated database and appropriate configuration, steps that should be handled by technical support, or very carefully by trained professionals after reading this guide.

The System Documentation section covers the documentation/API to each of the requisite files that power the POS, known issues with the system, design of the databases, and key design considerations relevant to future maintainers.

Terminology note: The following terms have a standardized meaning, which is held consistent throughout the document.

- *Item – Refers to an individual item itself. A single box of matches.*
- *Product – Refers to a type of item. Matches.*
- *Transaction – A single transaction with a customer, which may include one or many items.*
- *Item_Sold – Refers to an individual item that has been sold.*
- *Discount – A temporary change in price, commonly reflecting a special price of some kind.*

Component Functionality and Interfaces

We have made available pydoc-generated documentation for all source files, through the directory master/Documentation. These are html files that can be viewed through standard browsers. These cover general purposes and functionality, as well as interfaces, for all components.

Abstract discussions of functionality from a user's perspective can be found in the relevant sections of the Guide to Setup and Guide to User sections.

Known Problems

There are no known major errors with the Point-of-Sale System (POS) at this time. However, there are a few small items of note.

The POS database is a complex combination of multiple data tables. Therefore, some queries would cause permanent damage or crash the system entirely. Rather than accept these queries, POS recognizes them and simply drops them entirely. Future versions would convey appropriate information to correct the user's behavior. At the moment, however, it simply disregards the query and continues running.

Given the complexity of the database, instructions to setup and use must be followed. Without setting up each individual table appropriately by following the setup instructions included, the database will not have the necessary information to handle dependencies.

Database Design

The POS Database contains 7 tables for the various types of information, implemented through a SQL-Alchemy Wrapper with classes that inherit from the SQL-Alchemy Model class, all found in the file Models.py. Basic configuration settings direct SQL-Alchemy to the proper location and type for the production/local implementation. For more information on the proper configuration settings, see the configuration section.

All tables are user-accessible through the database dropdown menu to perform basic row additions, deletions, and edits. All fields in all tables are required ("nullable=false"). The complete framework is listed below. Each table column is listed along with its data type (listed *right)*. Fields in bold are user-entered, fields not in bold are automatically generated, most commonly from other data tables. Required columns from other data tables are underlined. Finally, columns' foreign keys are listed in italics. These prohibit row deletion if other rows in other tables are dependent or row addition if the appropriate dependency does not already exist. References to other tables are written as *other_table.column_in_other_table.* Capitalization is used for readability here; all source code variables and methods in the database are lowercase.

Data Types (Python)	
int	integer (primitive)
Float	floating point number
String-xx	string, *xx* specifies max number of characters
DATE	Instance of datetime.date (python module, see datetime documentation)

Items				
ID (int) *Primary Key*	**Product_ID (int)** ***Foreign Key: products.id***	**Inventory_Cost (Float)**	Expiration_Date (DATE) from products.id	Author_ID (int) *Foreign Key: users.id*

Users			
ID (int) *Primary Key*	**Name (String-20)**	**Password (String-400)**	**Permissions (int)**
Note: Permissions (int) is a categorical variable: **1 – Manager; 2 – Cashier; 3 - Stocker**			

Products						
ID (int) *Primary Key*	**Name (String-20)**	**Supplier_ID (int)** ***Foreign Key: suppliers.id***	Inventory_Count (int) from items	**Min_Inventory (int)**	**Shelf_Life (int)**	**Standard_Price (Float)**

Items_Sold					
ID (int) *Primary Key*	**Item_ID (int) from items.id**	Product_ID (int) from products.id	**Price_Sold (Float)**	Inventory_Cost (Float) from items.inventory_cost	**Transaction_ID (int) *Foreign Key: transactions.id***

Suppliers		
ID (int) *Primary Key*	**Name (String-20)**	**Email (String-40)**

Transactions				
ID (int) *Primary Key*	**Cust_Name (String-20)**	**Cust_Contact (String-40)**	**Payment_Type (int)**	**Date (DATE) (defaults to today)**
Note: Payment_type (int) is a categorical variable: 1 – Cash; 2 – Credit; 3 – Debit; 4 – Check; 5 – Other				

Discounts				
ID (int) *Primary Key*	**Product_ID (int) *Foreign Key: products.id***	**Start_Date (DATE)**	**End_Date (DATE)**	**Discount (Float)**
Note: Discount must be a number between 0 and 1 (inclusive), representing the "percent-off". For example, Discount = 0.3 would signify 30% off. Entering 0 would not change the price; 1 would make the item free.				

Configuration Settings

Each hosting service has different requirements for configuration settings with SQL-Alchemy. Reference their documentation for the specific settings to configure SQL-Alchemy to point correctly to your setup. Contact technical support for more information.

Design Considerations

The design of POS focuses around three general principles: simple, scalable, expandable.

Simple

Given the level of development time and experience, particularly a lack of experience with a web-application, we immediately recognized a need to keep a system within the grasp of beginner web developer. There are no complicated, flashy features. Rather the design is focused on a core, simple functionality.

Scalable

While our customer only intends to care around 50 products, we elected to design a system that could handle a vast expansion in both size and composition of inventory without huge performance issues. Functions are written to maximize code reuse and the effective utilization of existing, highly-optimized open-source modules. Furthermore, future versions could include the ability to remove large quantities of historical data at once, allowing the system to continue to provide fast functionality—even in large applications—for years to come despite concerns of disk size or hardware updates.

Expandable

Standing somewhat in contrast to our first principle of simplicity, we recognized the need of POS to change and adapt with the needs of the customer through changes in accounting practices, hardware, and the retail environment itself. In this line of thinking, we recognized that historical data not collected is lost forever. Therefore, we constructed a system that stores significant amounts of low-level data, withholding more advanced functionality or derived data for later versions. This design decision is most evident in the storage of a database entry for each individual item in inventory, down to each individual box of matches. While this is more data-intensive than most current applications, we agree with our customers when we note that we think this will soon be the industry standard.

Design in Practice

While we purport lofty design principles, we also have concrete examples of these principles in minute details of our design. These include the decision to stick to standardized web templates and rely heavily on their reuse as opposed to creating modified versions to optimize the interface for each individual page. Similarly, we worked to abstract our work far from details of deployment-specific hardware, relying extensively on python modules to provide the necessary interfaces.

Customer Report

Overview of Deliverable

The Point-Of-Sale System (POS) is a web-application that provides merchants with the ability to store and track inventory and sales in a robust and detailed custom database. Unlike most existing systems, POS stores individualized inventory information, eliminating complicated accounting issues in cost-of-sale calculations. It provides a detailed interface to stored data, helping to reduce regular technical maintenance requirements. Perhaps most importantly, it focuses around storing core data, allowing customers to easily add individualized functionality in future updates as required.

Terminology note: The following terms have a standardized meaning, which is held consistent throughout the document.

- *Item – Refers to an individual item itself. A single box of matches.*
- *Product – Refers to a type of item. Matches.*
- *Transaction – A single transaction with a customer, which may include one or many items.*
- *Item_Sold – Refers to an individual item that has been sold.*
- *Discount – A temporary change in price, commonly reflecting a special price of some kind.*

Comparison to customer request

POS meets and exceeds all customer requirements. Support for requirements not specifically addressed below can be found in referenced sections.

Core Requirements

- Handle a sale, looking up product prices, adding items to a receipt, and updating inventory values (*see Guide to Use: Cashier*)
- Generate need report when inventory for a product reaches a preset value (*see Guide to Setup and Guide to Use: Manager*)
- Record audit trail data (*see Guide to Use: Manager*)
- Receive incoming products (*see Guide to Use: Stocker*)
- Add or remove products from the inventory (*see Guide to Setup*)
- Update inventory values (see *Guide to Use: Manager*)
- Prepare reports, including sales-audit and inventory-worth (see *Guide to Use: Manager*)
- Handle temporary price changes (discounts) over a specified period of time (see *Guide to Use: Manager*)

Additional Requirements

- Log of transactions including: list of items sold, name of buyer, buyer's contact information, type of payment (see *Guide to Use: Manger*)
- Revenue and profit tracking for previous day, week, and month (see *Guide to Use: Manager*)
- Minimum capacity for approximately 50 different products (see *Guide to Setup*)
- Web interface (see *Guide to Use*)

Guide to Setup

Congratulations on your new Point-Of-Sale System (POS)!

By this point in the process, your system should be running on a web-server with both a web interface and a database, which you can verify by navigating to the URL provided by technical support when they installed your POS. Contact technical support if these steps are not completed. This guide will walk you through the following steps in system setup:

- User accounts
- Supplier information
- Product information
- Inventory input

User Accounts

When you first log on to POS, you will have to use the default username/password given to you by technical support. If technical support did not give you a default, use the manufacturer standard default. Username: admin Password: admin

Because default information is not secure, you'll first want to select Databases tab in the navbar and select Users from the dropdown. From here, you can create a new user by inputting the required information and selecting add. The first user you create should have manager permissions (by entering 1 in the permissions window). See *Guide to Use: Users* for more information regarding user permissions.

Next logout from the default account and login using your new username and password. Now you can delete the default user and add other users from the same window used to add the initial user.

Supplier Information

While logged in to an account with manager permissions, select Databases in the navbar and Suppliers from the dropdown. Here, you'll need to input a name and contact for suppliers of inventory you currently have or expect to input in the near future. An example entry may look like this: "Name: Acme Parts Co. Contact: suzyQ@acme.co"

Note: Contact information is saved as a string of plain text, so you can enter up to 40 characters of email, phone, or other information you would like to store.

Product Information

While logged in to an account with manager permissions, select Databases in the navbar and Products from the dropdown. Here, you'll need to input critical information for all products in your inventory (remember these are types, like matches, not individual items, like a box of matches). For each product type, enter the following information:

- Name: The product name, in 20 characters or less
- Supplier_ID: The ID number of that product's supplier, taken from the just-completed suppliers table
- Min_Inventory: A whole number of how many of this item you would like to have in stock at all times
- Shelf_Life: A whole number of days indicating how many days an item lasts on the shelf before it is expired and must be thrown out
- Standard_Price: A decimal number indicating the standard price you intend to charge for the product, $1.11

Inventory Input

While logged in to an account with manager or stocker permissions, select Inventory in the navbar. Here, you'll need to input your current inventory (you'll also use this window to add new inventory in the future). For each item (remember, an item is an individual item such as a box of matches), enter the following information:

- Product_ID: The ID number of that item's type of product, taken from the just-completed products table
- Inventory_Cost: A decimal number indicating the price paid for the item from the supplier, used for accounting/audit purposes.

After inventory input is complete, your POS is ready for use! Read on to learn how to handle day to day operations with you POS.

Guide to Use

Users

All employees need a user account, whether shared or unique, that reflects the level of system access required for their jobs. These accounts should be maintained by a manager, and updated regularly with employee turnover.

The POS supports three types of users, each with varying permissions. In general, they are:

- Manager (1) – Has full access to all windows, data tables, editing functionality, and the ability to create, edit, and delete other users.
- Cashier (2) – Has access to the cashier window and supporting data tables.
- Stocker (3) – Has access to the stocker window and supporting data tables.

Note – If a user has access to a given window, they have access to all features of that window.

Complete Permissions Guide (lack of access denoted by ~~strikethrough~~)		
Manager (1)	Cashier (2)	Stocker (3)

- Reports
- Cashier
- Stocker
- Databases
  - Items
  - Products
  - Transactions
  - Items Sold
  - Discounts
  - Suppliers
  - Users
	- ~~Reports~~
	- Cashier
	- ~~Stocker~~
	- Databases
	  - ~~Items~~
	  - ~~Products~~
	  - Transactions
	  - Items Sold
	  - ~~Discounts~~
	  - ~~Suppliers~~
	  - ~~Users~~
		- ~~Reports~~
		- ~~Cashier~~
		- Stocker
		- Databases
		  - Items
		  - Products
		  - ~~Transactions~~
		  - ~~Items Sold~~
		  - ~~Discounts~~
		  - ~~Suppliers~~
		  - ~~Users~~

If you would like modifications to these specifications, contact technical support.

Manager

As a manager, you have access to all windows and functionality. However, you'll probably spend most of your time in the Reports window, which features an accounting dashboard, allowing you to track pertinent stats automatically. In the left hand sidebar, you are able to update the dashboard to display these stats over a custom date window and to download copies of your data tables as comma-separated-value files for use in Microsoft Excel or other business analytics software.

In addition to downloading data tables, there are three custom reports that we think you'll find particularly useful: Inventory Worth, Revenue Audit, and Purchase Order.

The Inventory Worth report prepares a table of all items currently in your inventory. It first contains all information from the Items table and then adds all relevant information from the Products and Suppliers tables as well. This is available as a comma-separated-value file download, allowing you to quickly produce well-styled and informative custom reports through Microsoft Excel or other business analytics software. This is very useful in understanding the value of your inventory and how it breaks apart by different metrics such as by product type or supplier.

The Revenue Audit report prepares a table of all items you have sold and adds relevant transaction information. This is available as a comma-separated-value file download, allowing you to quickly produce well-styled and informative custom reports through Microsoft Excel or other business analytics software. This report is very useful for preparing documentation for auditing and for general accounting reports.

The Purchase Order report prepares a table of all products you need to reorder. As default settings, the POS checks the past week of sales for each product. If the difference between that product's current inventory and minimum inventory is less than the past week's sales, POS adds this product to those that it thinks you should order. POS then prepares a report for download as a comma-separated-value file that contains the relevant product information as well as name and contact information for the supplier. POS makes it simple to generate this report once a week and ensure that your inventory stays above minimum levels.

Under the Databases dropdown, you can access all data tables to add, remove, or edit information. Specific to your role as a manager, you may find yourself updating the products, suppliers, or users tables the same way you did in the guide to setup. Finally, the Discounts table allows you to add special pricing for a product over a certain date window. Simply add a new row to this table using the sidebar, specifying what product you would like the discount to apply to, when you would like the discount to begin and end, and by how much you would like to discount the product.
Note: Discount must be a number between 0 and 1 (inclusive), representing the "percent-off". For example, Discount = 0.3 would signify 30% off. Entering 0 would not change the price; 1 would make the item free.

Cashier

As a cashier, you have access to all the functionality necessary to check out a customer and to receive returns.

To check out a customer, head to the Cashier window. Here you'll see an empty table and the ability to add items by item id. Simply enter the item id, and the POS gets the appropriate pricing and information. When you have added all the customer's items, select "Enter Customer Info", which will prompt you to enter a customer's name, contact, and payment type from the dropdown. Then simply select "Finish Transaction". Note that there is an "other" payment type, useful for those transactions that just don't fit the mold. This allows for easy documentation of items taken at gunpoint.

Should you need to adjust a price, perhaps due to mislabeling, simply enter the appropriate row number and an updated price. Then select update and watch the changes happen.

To handle returns, you may have to access the Items Sold table under the Databases dropdown. Simply find the item id of the item being returned, and remove this from the items sold table. The relevant transaction information will update automatically.

Stocker

As a stocker, you have access to all the functionality necessary to receive shipments and add them to inventory.

To add items to inventory, head to the Stocker window. Here you'll see an empty table and a sidebar. To add items to inventory, enter the product id of what you are adding, the cost of that item from the supplier, and the quantity you are adding. All changes are stored locally, so any mistakes can be easily corrected, by using the row number to select a row and change incorrect values. Once all incoming items have been input into the table, select "Update Inventory" to send these changes to the POS Databases.

You may need to make specific adjustments to the Items or Products portions of the Databases, perhaps due to spoiled or broken items. Simply navigate to the appropriate table using the Databases dropdown. From there, select the relevant row using the id, and make any necessary changes.

Guide to Maintenance

Regular day-to-day operations should center around the Cashier, Inventory, Reports, and Database: Discounts windows. However, periodically, you may need to update other tables or correct erroneous entries. In general, reference the *Guide to Setup* for these revisions, employing the edit and delete functions of these windows in addition to the add function.

If user-maintenance causes errors or impairs functionality, reference the *System Documentation: Database Design* section for more information. If problems persist, contact technical support.

System Execution and Testing

For instructor evaluation and grading, we have deployed a production version of the Point-of-Sale System (POS) to http://sailingsales.pythonanywhere.com/. A description of this is under the Web-Based section below. Additionally, the more technically minded user may wish to experiment with deploying their own version of the system to a local host, which is detailed under the Local Host section. Such experimentation is for those familiar with Python source code and Flask-based applications only, as it is very technical in nature.

Web-Based

To demo POS for day-to-day operations, we have deployed a production version to http://sailingsales.pythonanywhere.com/. This simulates a client who has already completed the *Guide to Setup*. For evaluation and grading, we recommend re-following the *Guide to Setup* to observe adding and editing to different tables. Next, we recommend following the *Guide to Use* to observe the day-to-day functionality. Following these guides will cover all customer design requirements and fully exercise the system.

Local Host

Alternatively, POS may be deployed to a local host by the technically inclined user. However, we must emphasize that this is a complicated process recommended for individuals familiar with Python source code and Flask-based web applications. While POS is generally robust and resilient, it was designed for deployment to a web-server by technical support. We make no guarantees regarding the functionality of local host deployments of POS.

To deploy POS locally, first download all necessary source files from https://github.com/gioGats/CS2720-SE/archive/master.zip and unzip the file.

Next, prepare the local environment. POS requires Python 3.5 or higher and a list of additional modules. Ensure first that Python 3.5 is installed and functional, and then that each additional module listed in master/Flask-App/requirements.txt is installed and functional.

For those users new to Python, we recommend using the Pip module to install further modules. Most newer installs come with Pip by default, but the use of Pip from a command-line or terminal interface is operating system dependent.

- A Pip download is available at: https://pypi.python.org/pypi/pip
- Pip documentation is available at: https://pip.pypa.io/en/stable/

Once Pip is installed, enter the following command in the command-line/terminal while in the master/Flask-App directory

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

After all additional modules have been installed, you must initialize databases by running the python file master/Flask-App/db_create.py. This may be done through command-line/terminal input, the shell provided with Python, or other development environment.

Finally, run the python file master/Flask-app/app.py. This will deploy POS to the local host. Enter the IP address displayed in the python shell into a web browser (usually 127.0.0.1:5000). This will be a fully functioning local deployment of POS, with default “simulated” database entries as listed in master/Flask-App/db_create.py

Source Code Overview

The source code for the Point-of-Sale System (POS) is available for perusal and printing through the zip-file, at https://github.com/gioGats/CS2720-SE/archive/master.zip

Directory Structure (with file contents)

- DesignAndBacklog – Directory containing documents from design and product backlogs at various stages of the project.
- Documentation – Directory containing HTML versions of documentation/API for all files.
- Flask-App
  - app.py - This is the master file for the Flask application, contains logic for generating webpages, routing data, and managing the flow of the entire system.
  - db-create.py - Generates a sample database used for testing and demonstrations.
  - flask-bcrypt.py - A Flask extension providing bcrypt hashing and comparison facilities.
  - helper.py - This module provides user session management for Flask. It lets you log your users in and out in a database-independent manner.
  - manage.py – This module contains the configurations necessary for the production deployment. It is not used for local hosting.
  - models.py - Contains the SQL-Alchemy database model extension classes for each database table and the associated automatic functionality when working with calculated entries.
  - POS_database.py - Contains all database interaction functions for passing data between the user interface and databases.
  - POS_display.py - Module containing functions that get data input from the user interface, process it into appropriate formats, and route it accordingly.
  - POS_helpers.py - This is the file with helper functions for the POS Application.
  - POS_logic.py - Module that holds all class definitions, function definitions, and variables for displaying information to the WEBSITE tables (not to be confused with our db tables).
  - README.md – Readme file for display on GitHub.
  - requirements.txt – Text file listing all required modules, pre-formatted for pip.
- README.md – Readme file for display on GitHub.