****

****

Brey&Brew Management System

By:

Sofhia Aubrey M. Asilo

ADVISER

Ms. Jenny Rose Mendoza

IT 211: Database Management System

­­­­­­­

December, 2025

**PROJECT PROPOSAL**

**PROJECT DESCRIPTION**

Some business relied heavily on manual processes for order handling and sales monitoring. These traditional methods often led to delays, inconsistencies, and difficulties in maintaining organized daily operations. Staff experienced challenges in keeping accurate records, while management lacked quick access to reliable data needed for informed decision-making. These observations emphasized the need for a more efficient, accurate, and integrated digital solution to support smoother and more consistent business processes.

To address these issues, the Brey&Brew Management System was developed as a comprehensive desktop application using Python and SQLite. It serves as a centralized point of sale (POS) tool for a coffee shop. Designed to streamline essential store operations, the system integrates order processing, sales tracking, and data-driven reporting into an accessible and user-friendly interface. With this digital database solution, manual workload is significantly reduced, errors are minimized, and staff can perform tasks more efficiently, ensuring data accuracy and faster service. Furthermore, management now benefits from real-time data and insights, supporting better decision-making, enhanced productivity, and sustainable operational growth.

**PROJECT OBJECTIVES**

The Brey&Brew Management System is envisioned as an integrated digital platform designed to automate and streamline businesses daily operations by replacing manual, paper-based processes with a faster, more accurate, and organized workflow. The system centralizes essential functions such as order processing, sales monitoring, kitchen status tracking, and sales history viewing, allowing staff to perform tasks more efficiently while reducing errors and operational delays. Prioritizing usability, accessibility, and real-time data visibility, the system supports better decision-making and ensures smooth, consistent business operations. Ultimately, this project aims to enhance productivity, improve customer service, and contribute to the long-term growth and competitiveness of businesses such as Brey&Brew.

**SPECIFIC OBJECTIVES**

* **Database Integration:** To implement a persistent storage solution using SQLite that maintains data integrity across sessions.
* **Demonstrate Database Mastery:** To design and implement a relational database that adheres to normalization principles (3NF) to minimize data redundancy.
* **Relational Design:** To utilize a normalized database structure, specifically using an OrderItem bridge table to link Order and Product, ensuring a one-to-many relationship is handled correctly.
* **Full CRUD Functionality:** To provide a user-friendly interface for Creating, Reading, Updating, and Deleting product records and order data, allowing staff to interact with the database without needing technical SQL knowledge.
* **Operational Efficiency:** To simulate real-world operations including a *Kitchen Monitor* for tracking order status (Pending and Complete) and a *Sales History* for revenue analysis, replacing manual ticketing with digital automation.

**TOOLS AND TECHNOLOGIES**

1. **Programming Language**

* **Python:** The core logic of the application is written in Python.

1. **Graphical User Interface (GUI)**

* **Tkinter:** This is the standard Python interface for creating the GUI windows, buttons, labels, and input fields.
* **Tkinter.ttk:** Used for modern-styled widgets like the Notebook (tabs), Treeview (tables), and Style configuration.

1. **Database Management**

* **SQLite3:** A database engine used to store all application locally in a file named Brey&Brew.db
* **SQL:** Used for defining the database structure and performing CRUD (Create, Read, Update, Delete) operations.

1. **Image Processing**

* **Pillow (PIL):** The Python Imaging Library is used to open, resize, and display images within the Tkinter GUI.

1. **System Utilities**

* **OS Module**: Used to handle file paths and directory navigation, ensuring the application can correctly locate image files and the database regardless of where the project folder is stored.

**DATABASE DESIGN**

**PROPOSED DATABASE STRUCTURE**

Database Schema: Brey&Brew.db

**TABLE 1: User**

|  |  |
| --- | --- |
| **ATTRIBUTES** | **DATA DECLARATIONS** |
| user\_id | INTEGER PRIMARY KEY AUTOINCREMENT |
| username | TEXT UNIQUE NOT NULL |
| password | TEXT NOT NULL |
| role | TEXT DEFAULT 'staff' |

**TABLE 2: Product**

|  |  |
| --- | --- |
| **ATTRIBUTES** | **DATA DECLARATIONS** |
| product\_id | INTEGER PRIMARY KEY AUTOINCREMENT |
| name | TEXT NOT NULL UNIQUE |
| description | TEXT |
| price | REAL NOT NULL |
| image\_path | TEXT |

**TABLE 3: Order**

|  |  |
| --- | --- |
| **ATTRIBUTES** | **DATA DECLARATIONS** |
| order\_id | INTEGER PRIMARY KEY AUTOINCREMENT |
| user\_id | FOREIGN KEY RELATED TO users |
| status | TEXT DEFAULT 'Pending' |
| order\_date | TIMESTAMP DEFAULT CURRENT\_TIMESTAMP |

**TABLE 4: OrderItem**

|  |  |
| --- | --- |
| **ATTRIBUTES** | **DATA DECLARATIONS** |
| item\_id | INTEGER PRIMARY KEY AUTOINCREMENT |
| order\_id | FOREIGN KEY RELATED TO Order |
| product\_id | FOREIGN KEY RELATED TO Product |
| quantity | INTEGER |
| unit\_price | REAL |

**SAMPLE DATA**

|  |
| --- |
| INSERT OR IGNORE INTO User (user\_id, username, password, role) VALUES      (1, 'Sofhia', 'sofhia123', 'Manager');  INSERT OR IGNORE INTO Product (product\_id, name, description, price, image\_path) VALUES      (1, 'Hot Chocolate Deluxe', 'Marshmallow topped hot chocolate.', 120, 'images/Hot Chocolate.png'),      (2, 'Caramel Macchiato', 'Layered caramel macchiato with drizzle.', 110, 'images/Caramel Macchiato.png'),      (3, 'Chai Latte', 'Frothy spiced chai latte.', 110, 'images/Chai Latte.png'),      (4, 'Iced Americano', 'Refreshing iced black coffee.', 100, 'images/Iced Americano.png'),      (5, 'Iced Latte', 'Layered iced milk and espresso.', 110, 'images/Iced Latte.png'),      (6, 'Matcha Latte', 'Vibrant green tea latte.', 130, 'images/Matcha Latte.png'),      (7, 'Irish Coffee', 'Coffee with thick cream topping.', 110, 'images/Irish Coffee.png'),      (8, 'Hot Mocha', 'Whipped cream topped hot mocha.', 110, 'images/Hot Mocha.png'); |

**ENTITY-RELATIONSHIP DIAGRAM**

The Entity- Relationship Diagram represents the data structure for the "Brey&Brew Management System. It consists of four entities connected by specific relationships to ensure data integrity and efficient processing.

The Entity- Relationship Diagram represents the data structure for the Brey&Brew Management System. It consists of four entities connected by specific relationships to ensure data integrity, eliminate redundancy, and support efficient transaction processing.

|  |
| --- |
| **Order** |
| **PK**: order\_id  **FK**: user\_id  status  order\_date |

|  |
| --- |
| **User** |
| **PK**: user\_id  username  password  role |

has

processes

has

contains

|  |
| --- |
| **Product** |
| **PK**: product\_id  name  description  price  image\_path |

|  |
| --- |
| **OrderItem** |
| **PK**: item\_­id  **FK**: order\_id  **FK**: product\_id  quantity  unit\_price |

listed in

has

**ENTITIES**

* **User Table:**
* Manages the authentication and authorization details for staff members operating the system.
* The Primary Key*user\_id* uniquely identifies each staff member.
* **Order Table:**
  + Acts as the header for a transaction. It records *when* a sale happened and *who* processed it.
  + The Primary Key*order\_id* uniquely identifies the transaction receipt.
  + The Foreign Key*user\_id* links the order to the User who processed it.
* **Product Table:**
  + Stores the current menu items, including their standard prices and descriptions.
  + The Primary Key*product\_id* uniquely identifies each item on the menu.
* **OrderItems Table:**
  + This is an Associative Entity that links Orders and Products It details exactly which items were purchased in a specific transaction.
  + The Primary Key*item\_id* uniquely identifies a specific line on a receipt.
  + The Foreign Keys *order\_id* links to the specific receipt and *product\_id* links to the specific menu item.

**RELATIONSHIPS**

* **User & Order:**
  + A One-to-Many relationship (Optional).
  + A single User processes the transaction. While a User can process multiple Orders, they are not required to have any sales history (Optional Many). However, every Order must be linked to exactly one User (Mandatory One).
* **Orders & OrderItems:**
  + A One-to-Many relationship (Mandatory).
  + An Order is composed of OrderItems. An Order must contain at least one item to exist (Mandatory Many), and OrderItem *must* belong to exactly one Order (Mandatory One).
* **Products & OrderItems:**
  + A One-to-Many relationship (Optional).
  + One product can appear in many different OrderItems (sales). A Product can exist in the database without ever being sold (Optional Many). However, every sold OrderItem *must* reference a valid Product (Mandatory One).

**NORMALIZATION ANALYSIS**

The database schema has been normalized to the Third Normal Form (3NF) to optimize storage and integrity.

**First Normal Form**

All records are unique, and columns contain only atomic (single) values. To handle transactions where a customer purchases multiple different coffees, a dedicated OrderItem table was created. This allows a single Order ID to be associated with an infinite number of line items, satisfying 1NF requirements.

**Second Normal Form**

The tables utilize single-column integer Primary Keys, making partial dependency impossible. All non-key attributes are fully functionally dependent on the specific line-item entry defined by the Primary Key.

**Third Normal Form**

The Order table references the staff member via user\_id (Foreign Key) rather than duplicating the username. When staff details are needed for display, the system utilizes an SQL JOIN operation, preserving 3NF compliance. It also stores transaction metadata but intentionally excludes a ‘Total\_Amount’ column. Storing a total would create a transitive dependency on the sum of OrderItem rows. Instead, the application calculates totals dynamically using SQL aggregation whenever required, ensuring data consistency:

file: *database.py*

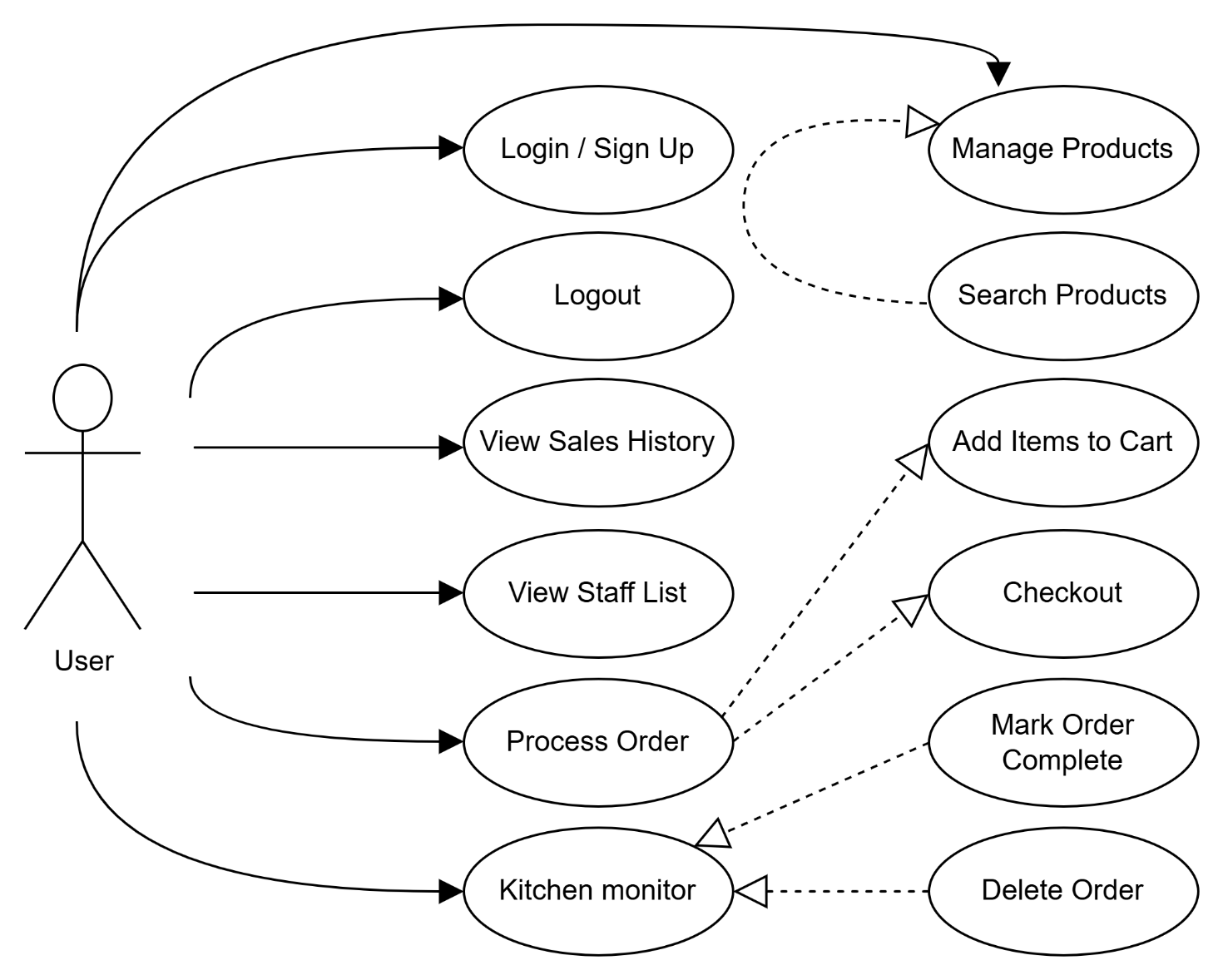
cursor.execute("SELECT SUM(quantity \* unit\_price) FROM OrderItem")

**Strategic Database Architecture**

A critical design feature is the implementation of *Snapshot Pricing*, it is the act of recording the current price of the product when an order is placed, ensuring that price remains fixed for that specific transaction, even if the product price changes later. While the Product table stores the current catalog price of an item, the OrderItem table also includes a unit\_price column.

If the management changes a product's price in the Product table, past sales records in OrderItem retain the price at the moment of sale. This ensures that financial reports and sales history remain accurate regardless of future price adjustments.

**Use Case Diagram**

****

<<extend>>

<<extend>>

<<extend>>

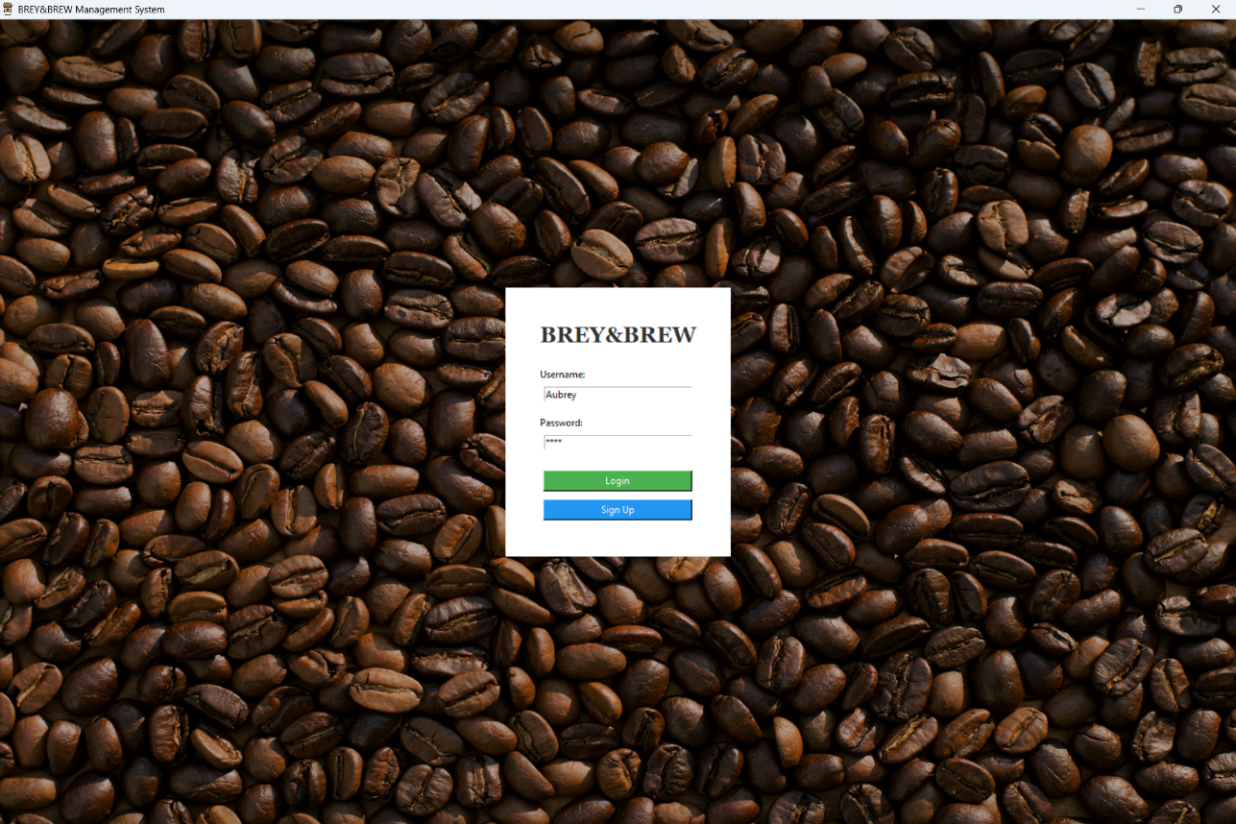
<<include>>

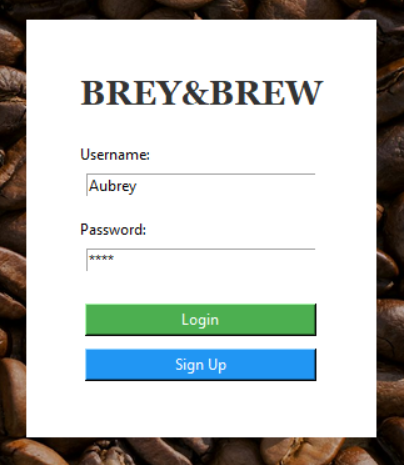
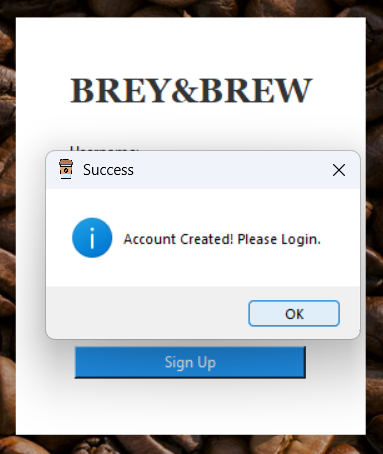
<<include>>

**GRAPHICAL USER INTERFACE**

**USER MANUAL**

**Login & Registration**

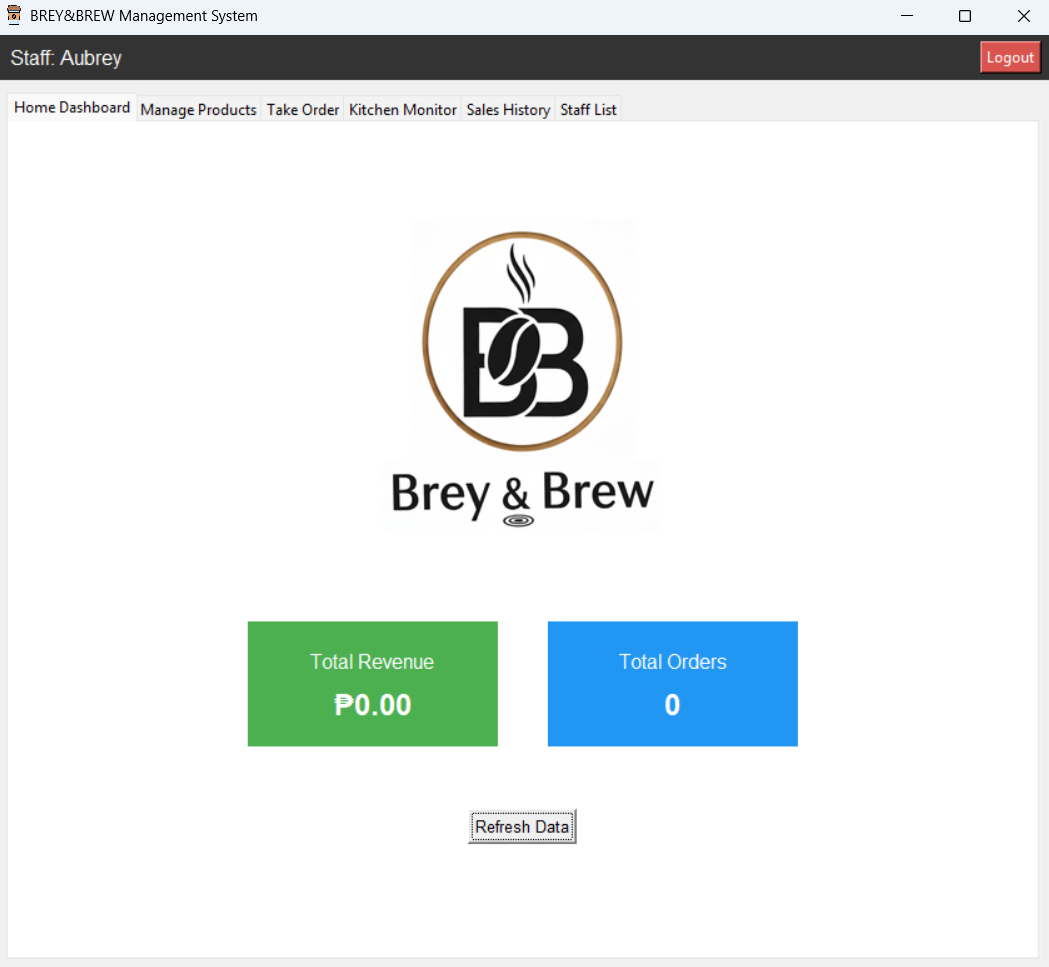
* **Log In:** Enter your **Username** and **Password** in the center card and press **Enter** or click the green **Login** button.
* **Sign Up:** If you are a new staff member, enter your desired credentials and click the blue Sign Up button. You will see a success message, after which you can log in.

****

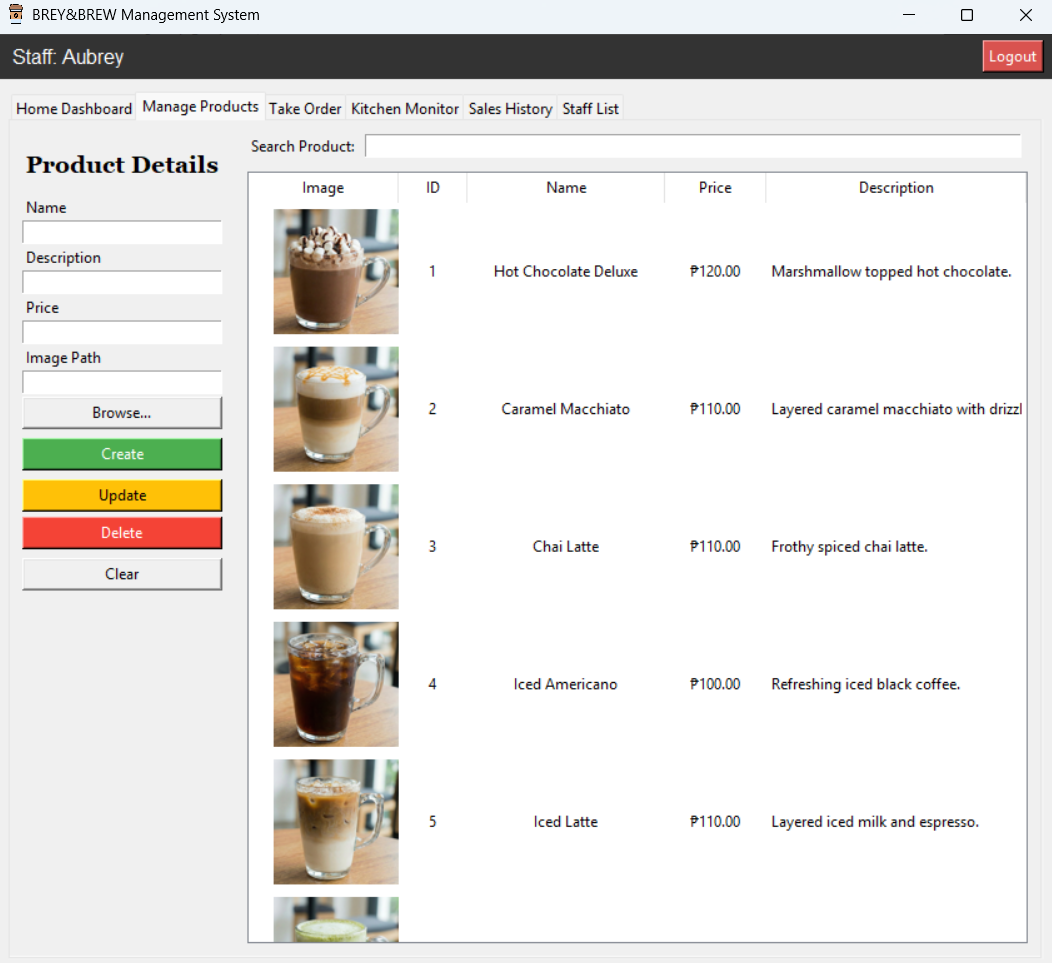
**Home Dashboard**

Once logged in, you will land on the **Home Dashboard**. This gives you an immediate snapshot of the business performance.

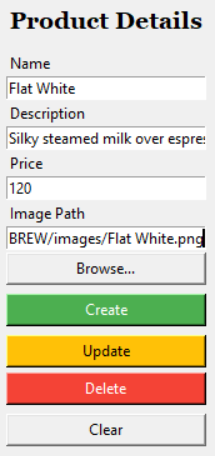
* **Total Revenue:** The green card displays the total sales generated from *Completed* orders.
* **Total Orders:** The blue card displays the total count of orders processed.
* **Refresh Data:** Click this button to update the statistics if new orders have been processed since you logged in.



**Managing Products**

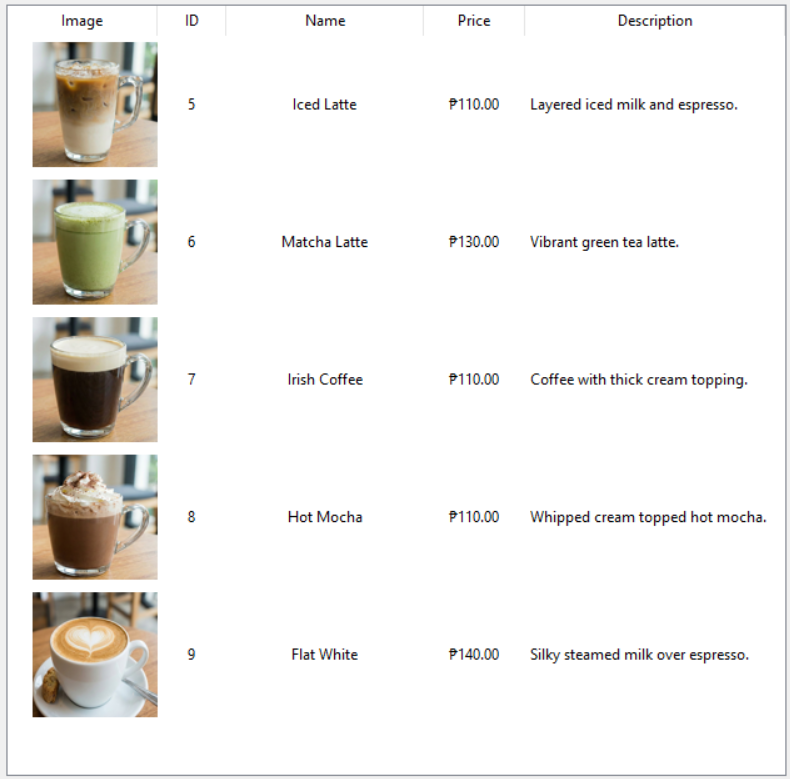
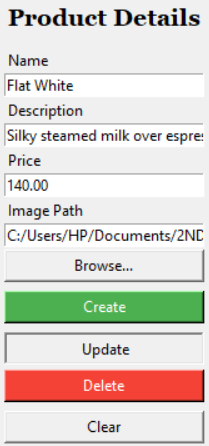
Navigate to the **Manage Products** tab to add, edit, or remove items from the menu.

**A. Adding a New Product**

1. **Name:** Enter the product name (e.g., "Flat White").
2. **Description:** Enter a short description (e.g., "Silky steamed milk over espresso.").
3. **Price:** Enter the numeric price (e.g., 120). *Do not include the "₱" symbol.*
4. **Image Path:** Click ‘*Browse...’* to select a .jpg or .png image from your computer.
5. Click the green **Create** button. The product will appear in the list on the right.

Product Added

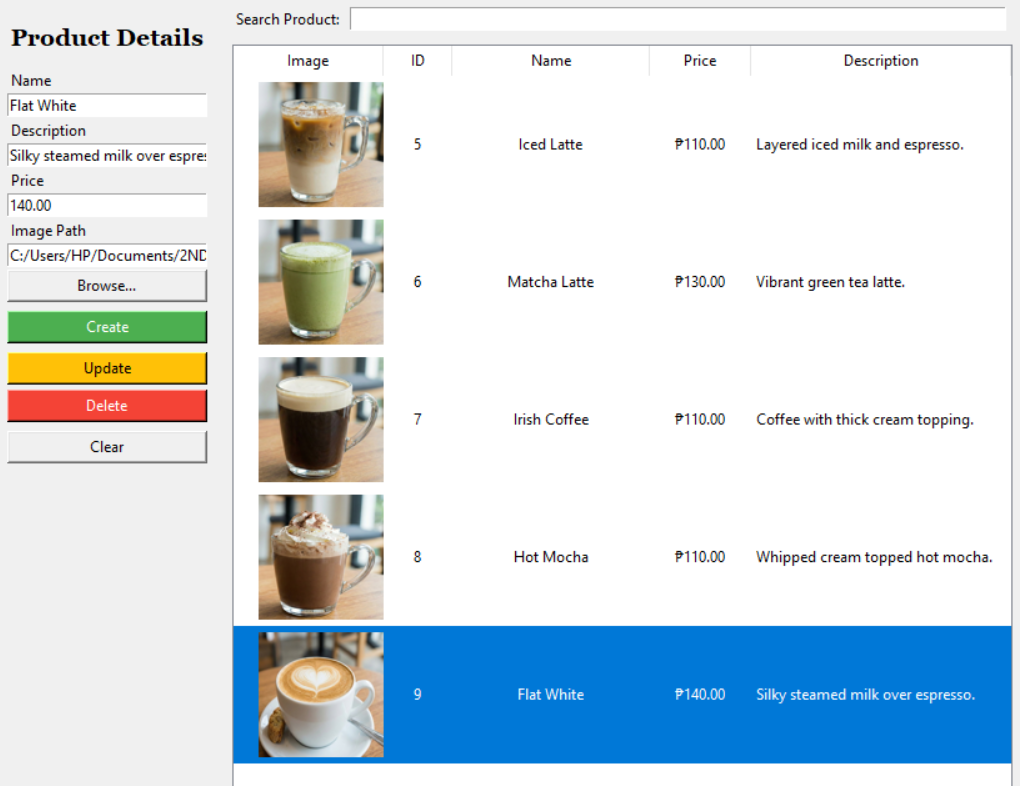
**B. Updating a Product**

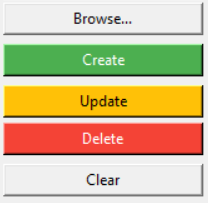
1. **Select** a product from the list on the right. The details will auto-fill into the form on the left.
2. Change the desired fields (e.g., update the Price).
3. ****Click the yellow **Update** button.

Price Updated

**C. Deleting a Product**

1. **Select** the product from the list.
2. Click the red **Delete** button. The item will be permanently removed from the database.



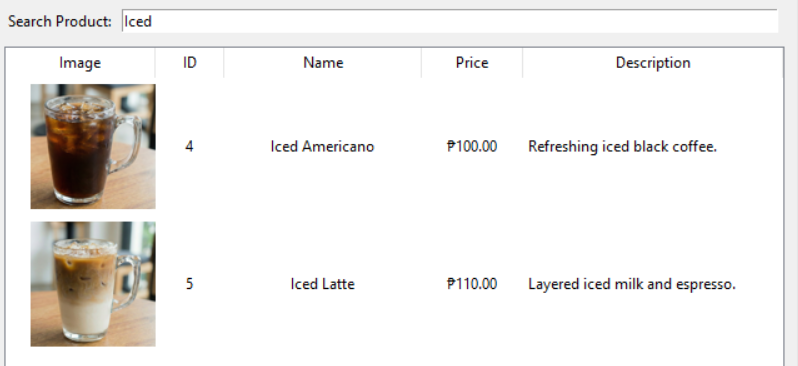
****

Click to Delete

**D. Search**

Use the **Search Product** bar at the top right to filter the list by name. Press **Enter** to clear the search.

Type something to search





**Taking Orders (POS)**

Navigate to the **Take Order** tab to process customer purchases.

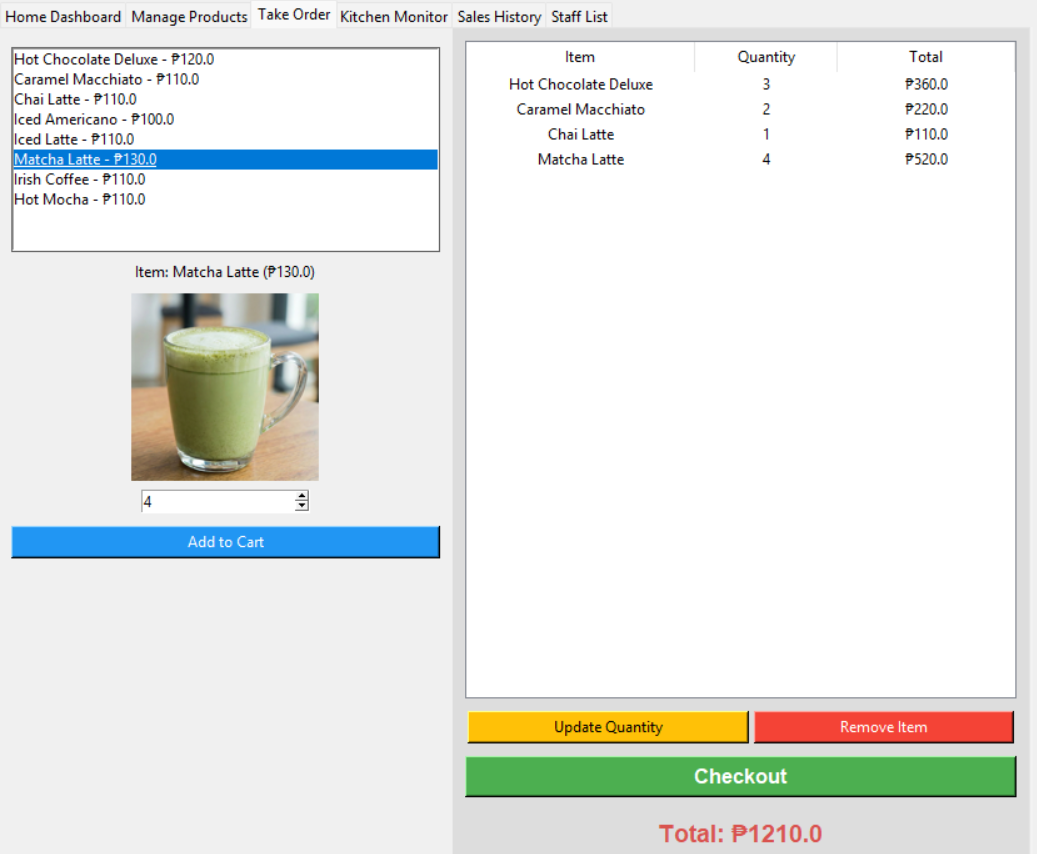
**A. How to Process an Order**

1. **Select Item:** Click a product from the list on the **Left**. You will see a preview image and price.
2. **Set Quantity:** Use the spinner to choose the quantity (default is 1).
3. **Add to Cart:** Click the blue **Add to Cart** button. The item will move to the *Cart* table on the **Right**.
4. **Checkout:** Once the customer is ready, verify the **Total** at the bottom right and click the green **Checkout** button.

* *Result:* The order is saved to the database with a status of "Pending" and sent to the Kitchen Monitor.

Cart

Product Selection



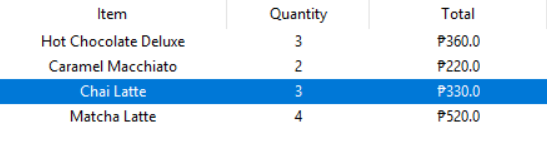
The Product Selection list automatically *updates* when products are created, edited, or deleted in the *Manage Product Tab.*

**B. Modifying the Cart**

* **Update Quantity:** Click an item in the *Cart*, change the number in the quantity box on the left, and click the yellow **Update Quantity** button.
* **Remove Item:** Click an item in the cart and click the red **Remove Item** button.



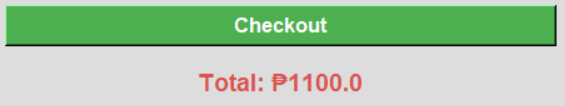
**Quantity Updated:**

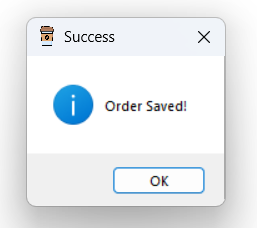


**Product Removed:**



**Order Checkout:**



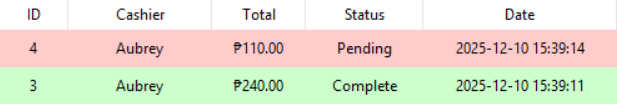


**Kitchen Monitor**

Navigate to the **Kitchen Monitor** tab. This is where baristas or kitchen staff view incoming orders.

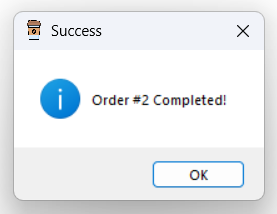
* ******View Details:** Click on any order in the *Incoming Orders* list (Left) to see the specific items and quantities in the *Order Details* pane (Right).

**Pink Rows:** Pending Orders.

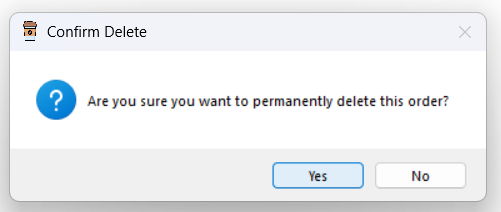
**Green Rows:** Completed Orders.

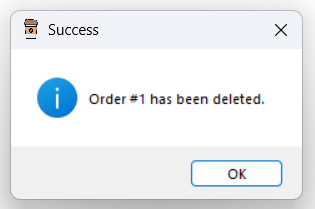


* **Complete Order:** When the drinks are ready, select the order and click **Mark Selected as Complete**. This updates the status and adds the revenue to the sales history.



* **Delete Order:** If an order was made in mistake, select **Delete Order** and confirm.



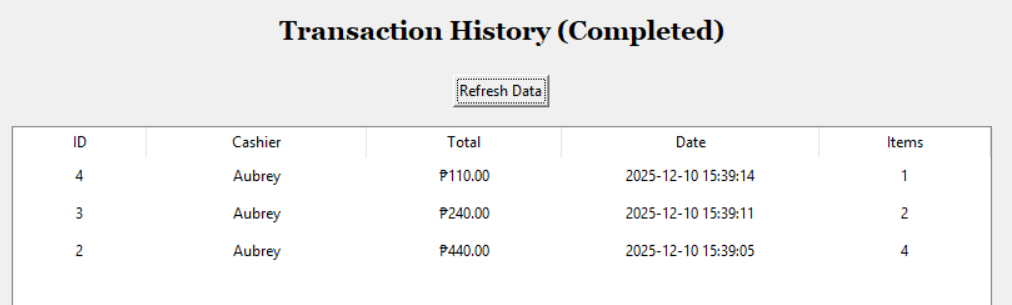


**Sales History & Staff List**

**A. Sales History Tab**

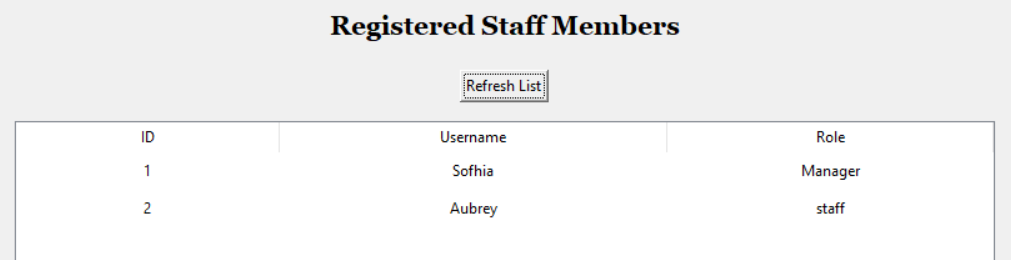
This tab serves as a ledger for completed transactions.

* View a read-only list of all successful sales, including the User ID, Total Amount, and Date.
* Use Refresh Data to see the latest transactions.

****

**B. Staff List Tab**

This tab displays all registered users in the system (ID, Username, and Role).

****

**REFLECTION AND CONCLUSION**

**CONCLUSION**

The development of the Brey&Brew Management System successfully achieved its primary objective of transitioning the coffee shop from a manual, paper-based workflow to an integrated digital solution. By leveraging Python for core logic and SQLite for data management, the system provides a robust platform that centralizes order processing, product management, and sales monitoring.

The project successfully addressed operational inefficiencies by automating sales tracking and minimizing human error. A significant technical achievement was the implementation of a Third Normal Form (3NF) database structure. By utilizing an associative entity (OrderItem) to bridge the relationship between Orders and Products, the system eliminates data redundancy while maintaining strict referential integrity.

The system ensures financial accuracy through "Snapshot Pricing," a critical design feature that records the unit price at the moment of sale. This ensures that historical sales data remains accurate even if menu prices are updated in the future. Ultimately, the Brey&Brew Management System demonstrates that a well-architected desktop application can significantly enhance productivity, provide real-time business insights, and serve as a scalable foundation for future growth.

**REFLECTION**

While the initial goal was simply to digitize a coffee shop's workflow, the process of building the system revealed the critical importance of rigorous database design and software architecture. Working on the Brey&Brew project was a significant learning experience that bridged the gap between theoretical programming concepts and real-world application development. Through the construction of this Point of Sale (POS) system, several key insights were gained:

* **Database Normalization**

One of the most significant technical challenges was correctly implementing the Third Normal Form (3NF). In the initial planning stages, it was tempting to simply list products directly within an order record. However, implementing the Relationship Schema taught me that a Many-to-Many relationship between Orders and Products cannot exist directly in a relational database. By creating the associative entity OrderItem, I successfully bridged these tables. This not only eliminated data redundancy but also allowed for infinite scalability, a single order can now contain any number of unique items without breaking the database structure.

* **Data Integrity and Snapshot Pricing**

I realized that simply linking to the Product table was insufficient because menu prices change over time. If a product's price was updated in the master list, all historical sales records would effectively become incorrect. To solve this, I implemented "Snapshot Pricing" by including a unit\_price column in the OrderItem table. This ensures that the price is recorded at the moment of the transaction, preserving the integrity of historical financial data regardless of future menu updates. This reinforced the lesson that data integrity is not just about structure, but about the context of time.

* **GUI logic vs. Backend Logic**

Integrating Tkinter with SQLite3 required careful management of data flow. Learning how to capture user input from the UI, sanitize it, and execute accurate CRUD (Create, Read, Update, Delete) operations provided a deeper understanding of full-stack desktop development.

* **User Experience Considerations:**

Developing the interface taught me that functionality must be matched with usability. Features like the dynamic Cart View, visual product previews using Pillow (PIL), and the dedicated Kitchen Status tab were designed specifically to make the staff's workflow intuitive and efficient.

In summary, this project not only honed technical skills in Python and SQL, it was a simulation of real-world software engineering. It demonstrated that a successful application requires a solid foundation in database theory. The Brey&Brew Management System stands as proof that well-planned architecture, strict adherence to normalization principles, and user-centric design are the keys to building robust digital solutions