PULP - Interactive Ordering System using Python Language

Programming Languages LAB
BSCS - J4A

Members:
Agustin, Jerome Loyd
Bermundo, Nicole
Depoo, Zynnon Kyle
Torres, Junell

Instructor:
Grachel Eliza Ching

November 2025

Version	Date	Description of Change
1.0 - PULP	November 20, 2025	Initial draft / Backend Coding
1.1 - PULP	November 21, 2025	Added frontend and Finalization

Executive Summary

PULP is an interactive e-commerce ordering system that helps customers browse products and manage orders easily. It is designed for small businesses like cafes or online shops that need a simple digital ordering tool. Customers can view items with images, names, and prices. They can add products to their order, change quantities, or remove items. The system keeps track of the total cost automatically.

The main improvement from the midterm version is the separation of backend and frontend. Previously, the system was a single block of code, which was hard to manage. Now, the backend handles order management, calculations, and data logic, while the frontend focuses on user interface and interaction. This separation improves maintainability and allows easier updates or feature additions. It also makes the system more flexible.

The project also emphasizes clean and modular code. Helper functions, backend modules, and GUI logic are separated for better structure. The system shows professional practices like version control, teamwork, and error handling. It handles edge cases and missing data more gracefully. Overall, PULP now delivers a complete, usable, and well-structured ordering system that demonstrates good programming practices.

System Overview

Background of the project idea

The project aims to digitize the ordering process for small businesses. Before, orders were handled manually or with a simple program, which was inefficient and prone to errors. PULP provides an interactive digital system that shows products, prices, and images. It allows customers to add items to an order, modify quantities, and check out easily. The final project version separates backend and frontend to improve structure and maintainability.

Problem Addressed

Small businesses often face challenges with order management using paper or basic programs. Mistakes can happen when calculating totals or keeping track of items. The previous midterm version had all code in a single file, which made updates and enhancements difficult. PULP solves this by splitting the system into backend logic and frontend interface. This ensures a smoother and more reliable ordering experience.

Objectives:

- Separate backend and frontend for cleaner code and easier maintenance.
- Improve usability, error handling, and performance.
- Demonstrate professional coding practices, including modularity and teamwork.
- Provide a system that can be extended with new features easily.

Development environment and tools used.

- **Programming Language:** Python 3.10+ – Used to develop both backend logic and frontend GUI.
- **IDE:** Visual Studio Code – Provides code editing, debugging, and project management tools.
- **Version Control:** Git and GitHub Classroom – Used for team collaboration, code tracking, and version history.
- **GUI Library:** Tkinter – Handles the creation of windows, buttons, labels, and other interactive elements.
- **Image Handling Library:** Pillow (PIL) – Used to load, resize, and display product images in the GUI.
- **Operating System:** Windows 10/11 – Platform where the system was developed and tested.

Software design & Architecture

Diagram of program structure or module relationships.

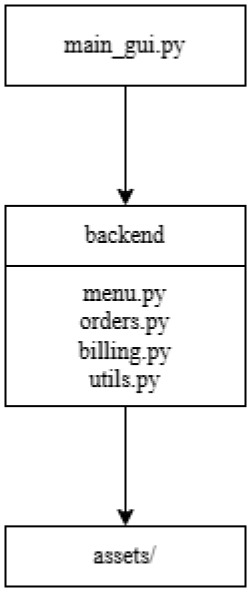


Figure 1: Diagram of program structure or module relationships.

On figure 1 the program is divided into two main parts: frontend and backend. The frontend is the main_gui.py file, which is the graphical user interface where users can see the menu, add items to their order, and checkout. The backend contains all the logic and data handling, including menu.py for menu items, orders.py for creating and validating orders, billing.py for calculating totals and printing receipts, and utils.py for helper functions like input validation. The frontend communicates with the backend to get menu data and update orders. All images and icons are stored in the assets folder, which the frontend uses to display items.

Explanation of major functions/subprograms.

Backend Functions

- menu.py
    - display_menu(): Shows the full menu in a clean format.
    - add_item(category, item_id, name, price, description): Adds a new item to the menu.

    - remove_item(item_id): Removes an item from the menu by its ID.

- orders.py
    - create_order(): Lets users choose menu items and creates an order list.
    - get_all_valid_ids(): Returns all valid menu item IDs for input validation.
- billing.py
    - calculate_total(order): Calculates the total price of the order.
    - print_receipt(order): Prints a formatted receipt showing items, categories, and total.
- utils.py
    - validate_input(value, valid_options): Checks if user input is valid.
    - get_valid_input(prompt, valid_options): Keeps asking for input until a valid option is entered.

Frontend Functions

- main_gui.py
    - asset_path_for_item(item): Finds the image file path for a menu item.
    - load_original_image(path): Loads an image from the assets folder.
    - get_resized_photo(path, w, h): Resizes an image to fit a GUI card.
    - add_to_order(item), increase_item(iid), decrease_item(iid), remove_item(iid): Manage items in the current order.
    - refresh_order_panel(): Updates the order panel in the GUI to reflect changes.

  - on_checkout(): Shows a receipt and clears the order when the user checks out.

  - rebuild_menu_grid(): Dynamically creates the menu cards in the GUI depending on window size.

  - card_on_click(frame, item, selected): Handles the user clicking on a menu item card to add it to the order.

Flowchart or pseudocode of main processes.

1. Main Program (main.py / GUI flow)

```
START
Display Welcome Message        # Show restaurant name and slogan

LOOP until user quits
   DISPLAY main options (1: Place Order, 2: View Menu, q: Quit)  #
Main menu options
   GET user input              # Read what the user wants to do

   IF input == 'q'             # User wants to exit
      EXIT loop

   ELSE IF input == '2'        # User wants to view the menu
      CALL display_menu()       # Show all menu items with categories

   ELSE IF input == '1'        # User wants to place an order
      CALL display_menu()       # Show menu before ordering
      CALL create_order()       # Start order creation process
      IF order is not empty     # Check if user added any items
         CALL print_receipt(order)  # Print the order receipt
      ELSE
         DISPLAY "No items ordered"  # Inform user they did not order
anything

   ELSE
      DISPLAY "Invalid choice"    # Handle invalid input

END LOOP
END
```

2. Order Creating (orders.py)

```
FUNCTION create_order()
    INIT empty order list           # Prepare an empty list for the order
    GET all valid menu IDs          # From backend menu
    LOOP until user enters 'q'      # Keep asking until user finishes
        ASK user to input item number
        IF input is valid ID        # Check if the number is in the menu
            ADD item to order list
            DISPLAY "Item added"    # Confirm addition to the user
        ELSE
            DISPLAY "Invalid item number"  # Show error for wrong input
    RETURN order                    # Send completed order back to main
END FUNCTION
```

3. Checkout / Billing (billing.py)

```
FUNCTION create_order()
    INIT empty order list           # Prepare an empty list for the order
    GET all valid menu IDs          # From backend menu
    LOOP until user enters 'q'      # Keep asking until user finishes
        ASK user to input item number
        IF input is valid ID        # Check if the number is in the menu
            ADD item to order list
            DISPLAY "Item added"    # Confirm addition to the user
        ELSE
            DISPLAY "Invalid item number"  # Show error for wrong input
    RETURN order                    # Send completed order back to main
END FUNCTION
```

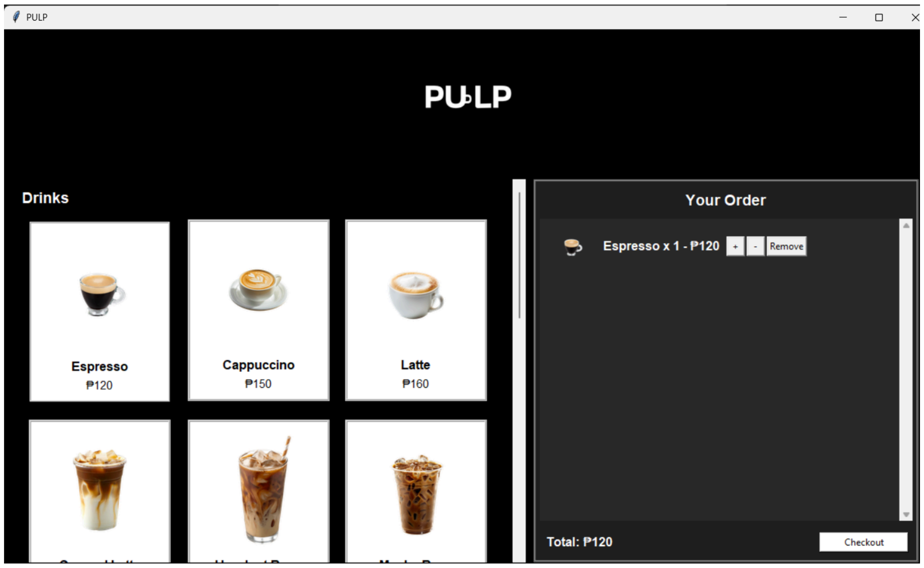
The program starts by showing a welcome message with the restaurant name and slogan. It then enters a loop where the user can view the menu, place an order, or quit. Viewing the menu displays all items by category. Placing an order lets the user select items, after which a receipt is generated with item details and the total. Invalid inputs are handled with error messages, and the loop repeats until the user quits, ensuring a smooth ordering process.

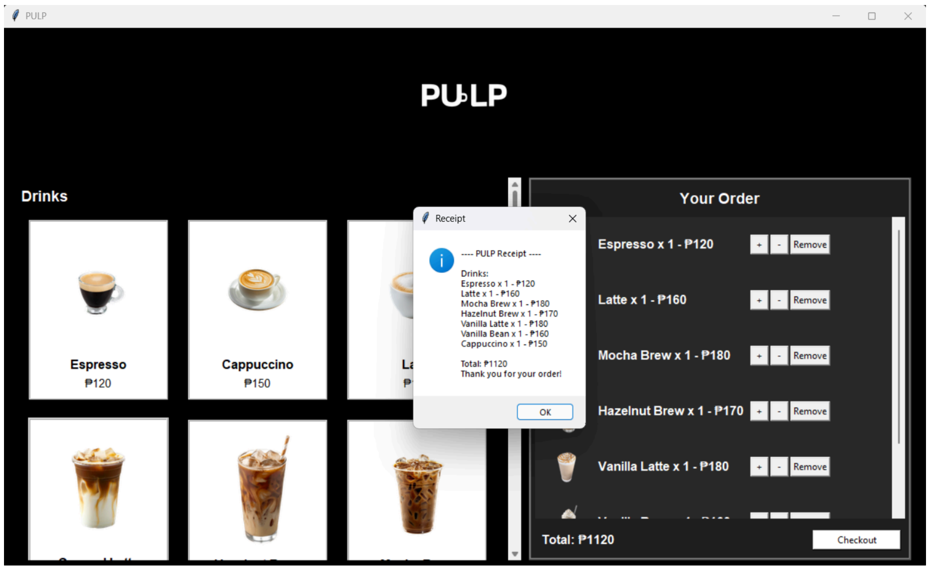
Concurrency / Multi-Threading Model Explanation

The system currently runs as a single-threaded application. Concurrency is not implemented because user interactions (menu browsing, ordering, and checkout) are handled sequentially in the GUI or terminal. For the GUI frontend, the Tkinter mainloop ensures smooth UI updates without blocking, so image loading and order updates appear responsive. If needed in the future, features like simultaneous order processing or background data saving could use Python’s threading or asyncio to prevent the interface from freezing. This design keeps the program simple, stable, and easy to maintain while maintaining a responsive user experience.

Implementation

Sample Input/Output:





```
---- PULP Receipt ----

Drinks:
Espresso x 1 - ₱120
Latte x 1 - ₱160
Mocha Brew x 1 - ₱180
Hazelnut Brew x 1 - ₱170
Vanilla Latte x 1 - ₱180
Vanilla Bean x 1 - ₱160
Cappuccino x 1 - ₱150

Total: ₱1120
Thank you for your order!
```

Highlight sections of code that demonstrate

Control Flow

- In main_gui.py, clicking a menu item triggers card_on_click(), which adds the item to the order and updates the panel.

- In main.py (backend), the main loop asks the user to view menu, place order, or quit. Each choice runs a different function.

Function Modularity

- Backend is split into modules: menu.py for menu data, orders.py for order creation, billing.py for total calculation and receipts, and utils.py for input validation.

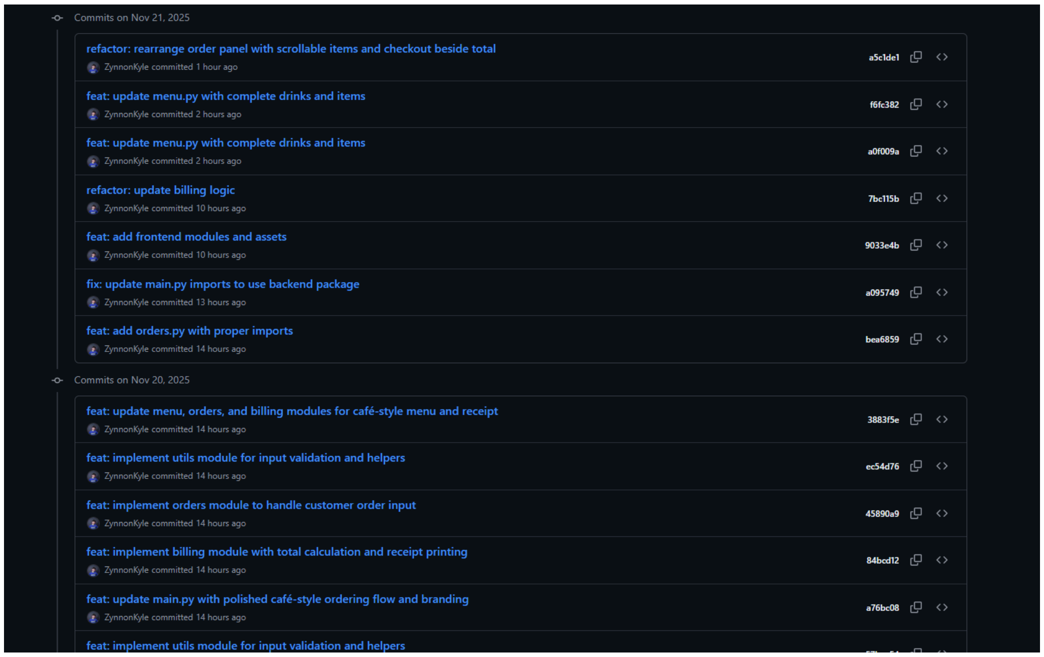
- Frontend handles GUI rendering, image handling, and order panel updates in separate helper functions.

Concurrency

The system currently runs single-threaded. The GUI uses Tkinter's mainloop, which keeps the interface responsive. Concurrency can be added later for handling multiple tasks simultaneously.

Explanation of how Git commits reflect teamwork (show commit logs or history summary).

- Each member worked on separate features and modules, reflected by multiple logical commits.
- Commit messages follow the Conventional Commits style (feat:, fix:, refactor:).
- Example:
  feat: add scrollable order panel in GUI
  refactor: separate menu functions into backend module
  fix: correct image loading in frontend


Commits on Nov 21, 2025

refactor: rearrange order panel with scrollable items and checkout beside total
ZynnonKyle committed 1 hour ago — a5c1de1

feat: update menu.py with complete drinks and items
ZynnonKyle committed 2 hours ago — f6fc382

feat: update menu.py with complete drinks and items
ZynnonKyle committed 2 hours ago — a0f009a

refactor: update billing logic
ZynnonKyle committed 10 hours ago — 7bc115b

feat: add frontend modules and assets
ZynnonKyle committed 10 hours ago — 9033e4b

fix: update main.py imports to use backend package
ZynnonKyle committed 13 hours ago — a095749

feat: add orders.py with proper imports
ZynnonKyle committed 14 hours ago — bea6859

Commits on Nov 20, 2025

feat: update menu, orders, and billing modules for café-style menu and receipt
ZynnonKyle committed 14 hours ago — 3883f5e

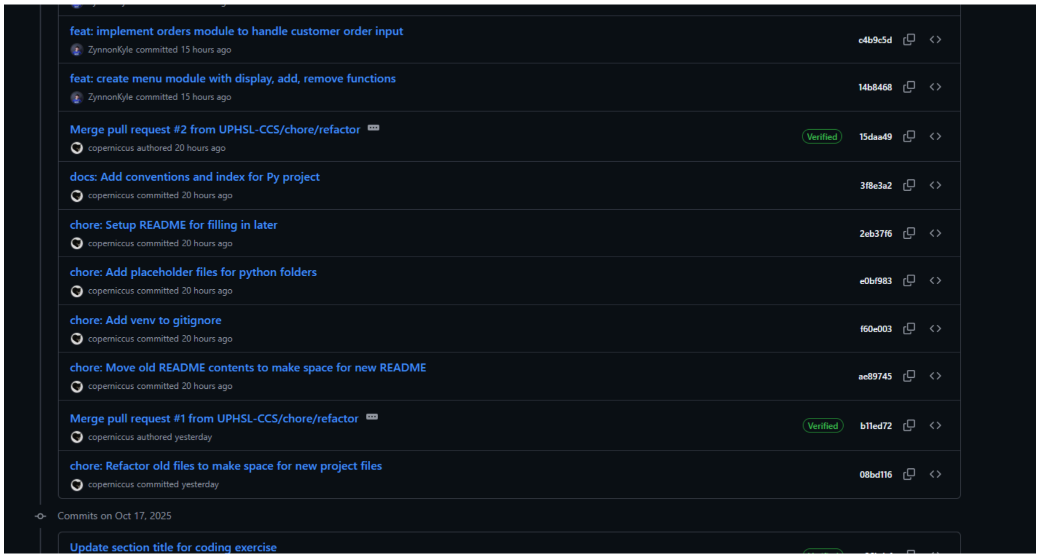
feat: implement utils module for input validation and helpers
ZynnonKyle committed 14 hours ago — ec54d76

feat: implement orders module to handle customer order input
ZynnonKyle committed 14 hours ago — 45890a9

feat: implement billing module with total calculation and receipt printing
ZynnonKyle committed 14 hours ago — 84bcd12

feat: update main.py with polished café-style ordering flow and branding
ZynnonKyle committed 14 hours ago — a76bc08

feat: implement utils module for input validation and helpers


feat: implement orders module to handle customer order input
ZynnonKyle committed 15 hours ago — c4b9c5d

feat: create menu module with display, add, remove functions
ZynnonKyle committed 15 hours ago — 14b8468

Merge pull request #2 from UPHSL-CCS/chore/refactor
copernicus authored 20 hours ago — Verified — 15daa49

docs: Add conventions and index for Py project
copernicus committed 20 hours ago — 3f8e3a2

chore: Setup README for filling in later
copernicus committed 20 hours ago — 2eb37f6

chore: Add placeholder files for python folders
copernicus committed 20 hours ago — e0bf983

chore: Add venv to gitignore
copernicus committed 20 hours ago — f60e003

chore: Move old README contents to make space for new README
copernicus committed 20 hours ago — ae89745

Merge pull request #1 from UPHSL-CCS/chore/refactor
copernicus authored yesterday — Verified — b11ed72

chore: Refactor old files to make space for new project files
copernicus committed yesterday — 08bd116

Commits on Oct 17, 2025

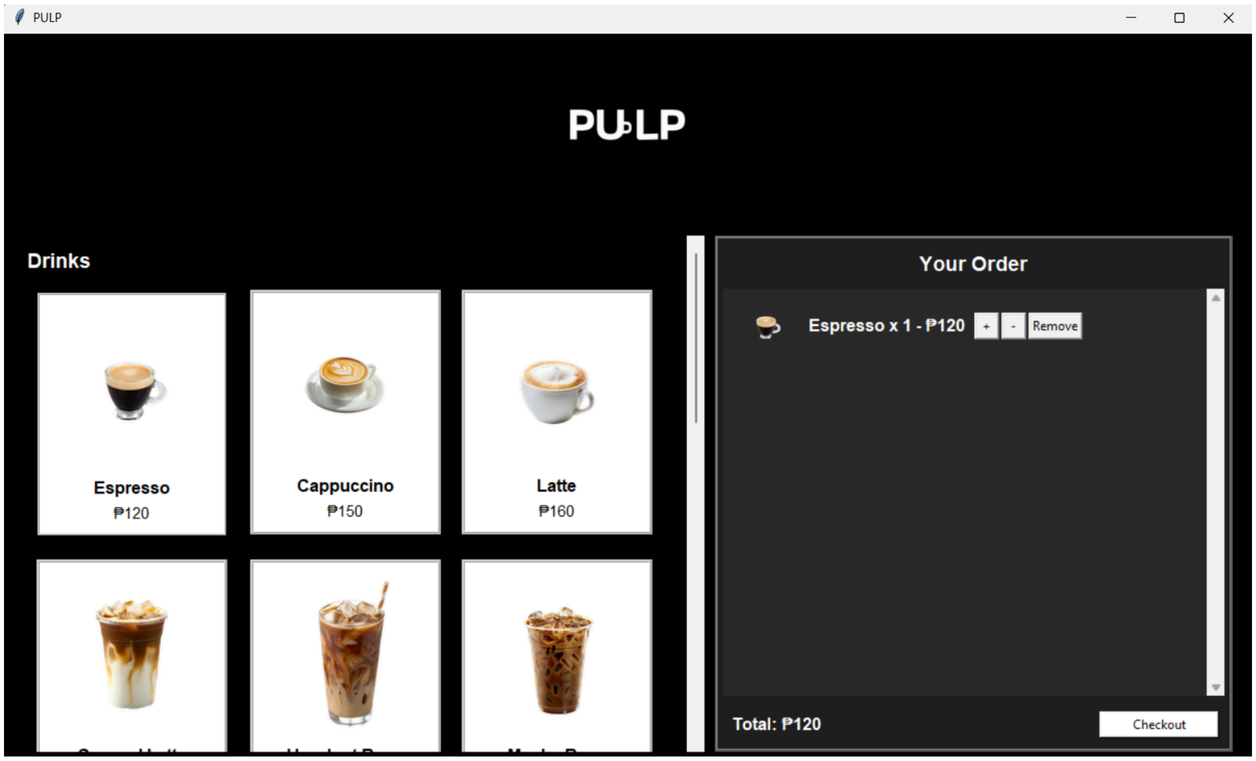
Update section title for coding exercise

Testing Evaluation

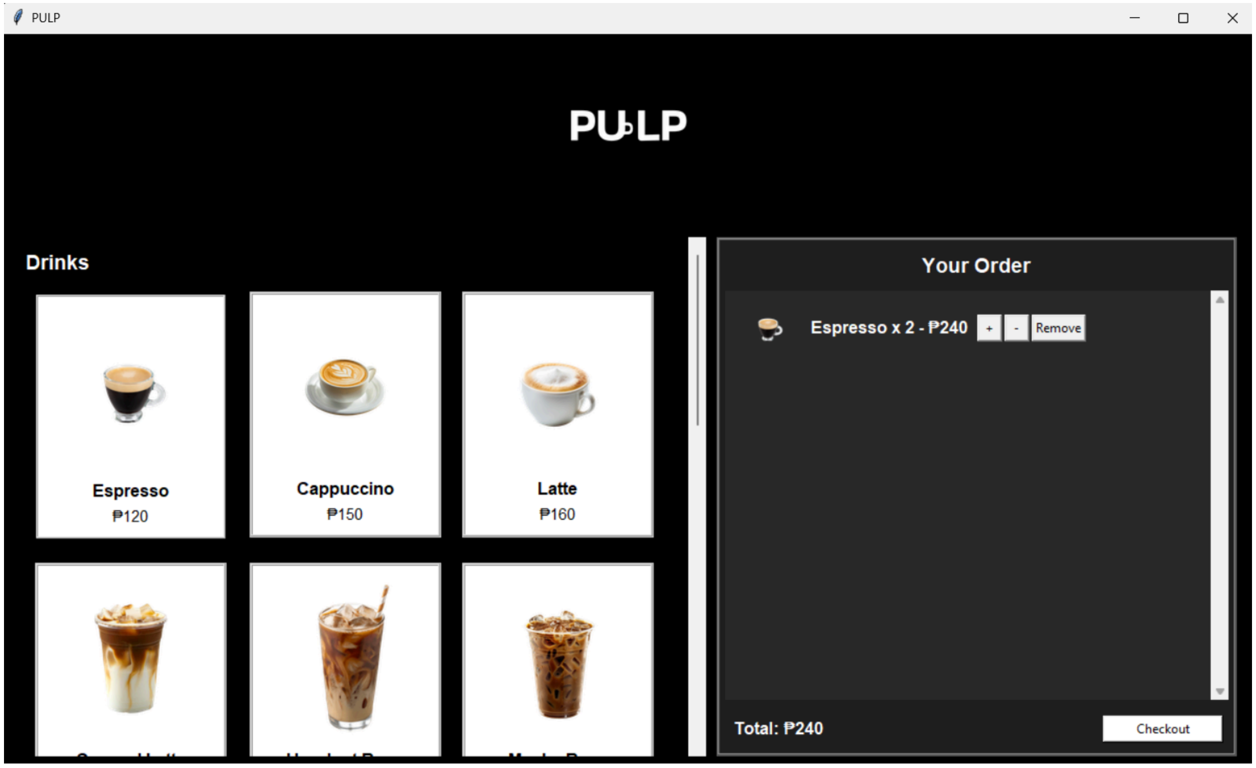
Test cases used (inputs, expected outputs, actual outputs).

Test Case	**Input**	**Expected Output**	**Actual Output**	**Result**
Placing Order	Click “Espresso”	Order panel shows Espresso x1, total 120 Pesos	Same as expected	Pass
Increase Quantity	Click “+” on Espresso	Quantity updates to x2, total 240 pesos	Same as expected	Pass
Remove Item	Click “Remove” on Espresso	Item disappears, total 0 Peso	Same as expected	Pass
Checkout empty	Click “Checkout” with no items	Message “No items in order”	Same as expected	Pass
Multiple Items	Add Espresso and Croissant	Order panel shows both items, correct subtotal	Same as expected	Pass

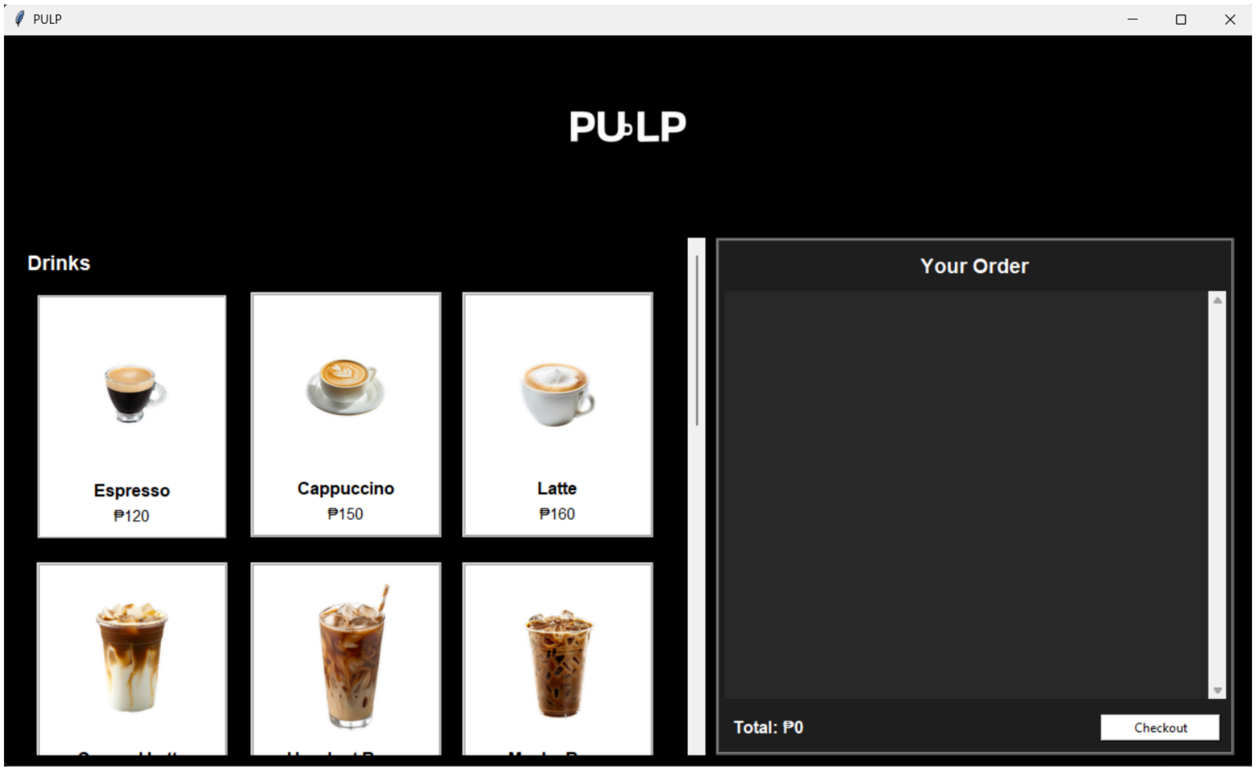
Place single order



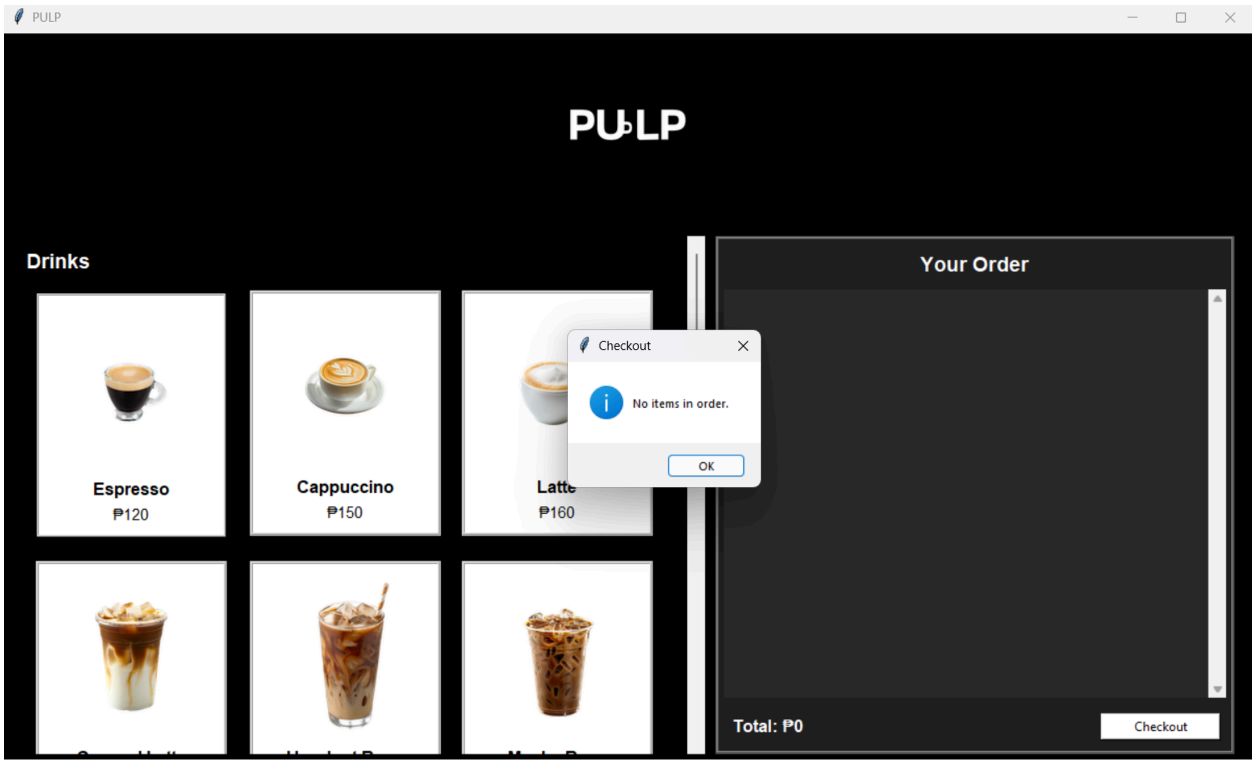
Increase quantity



Remove Item

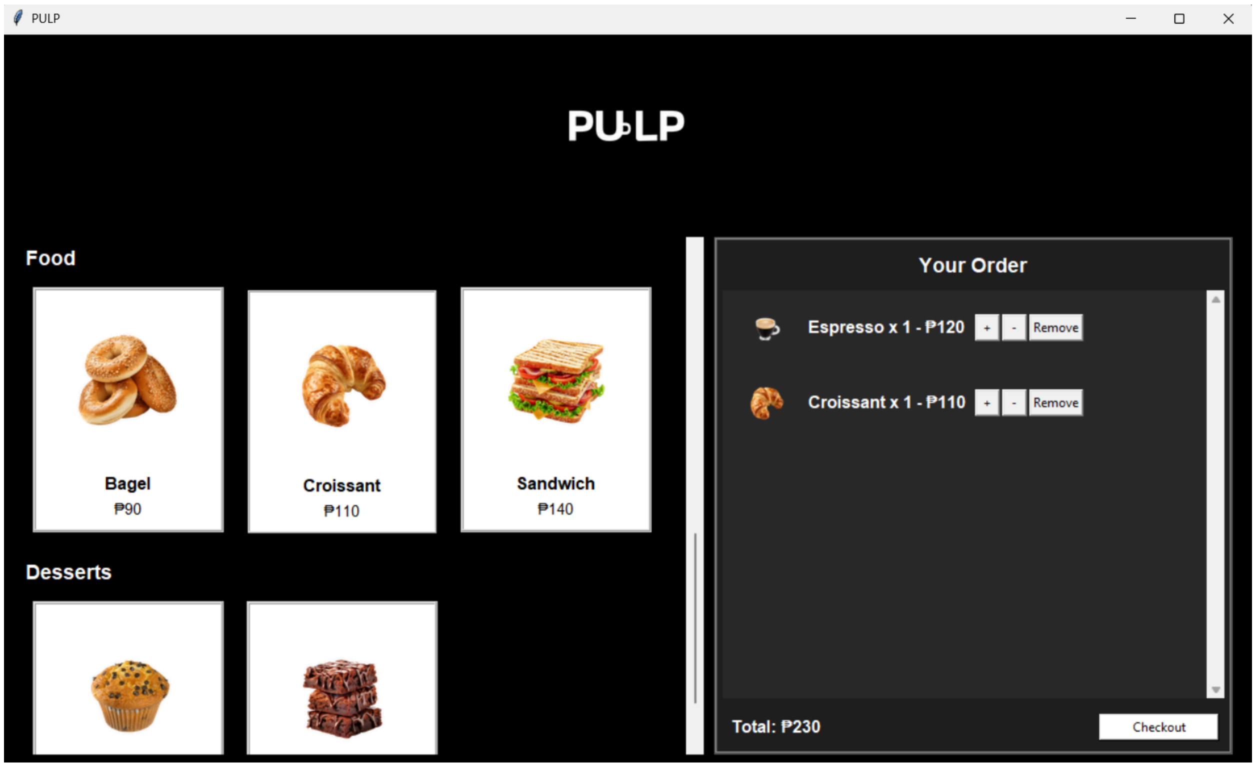


Checkout empty



No items in order.

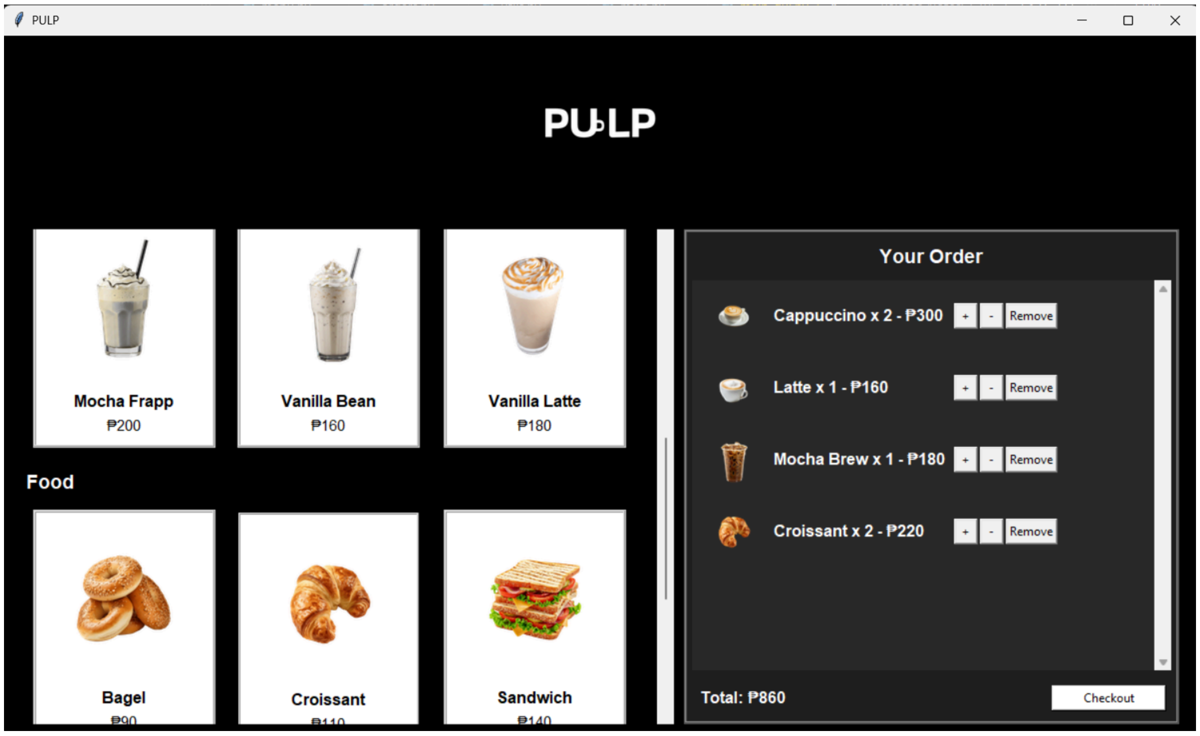
Multiple Items



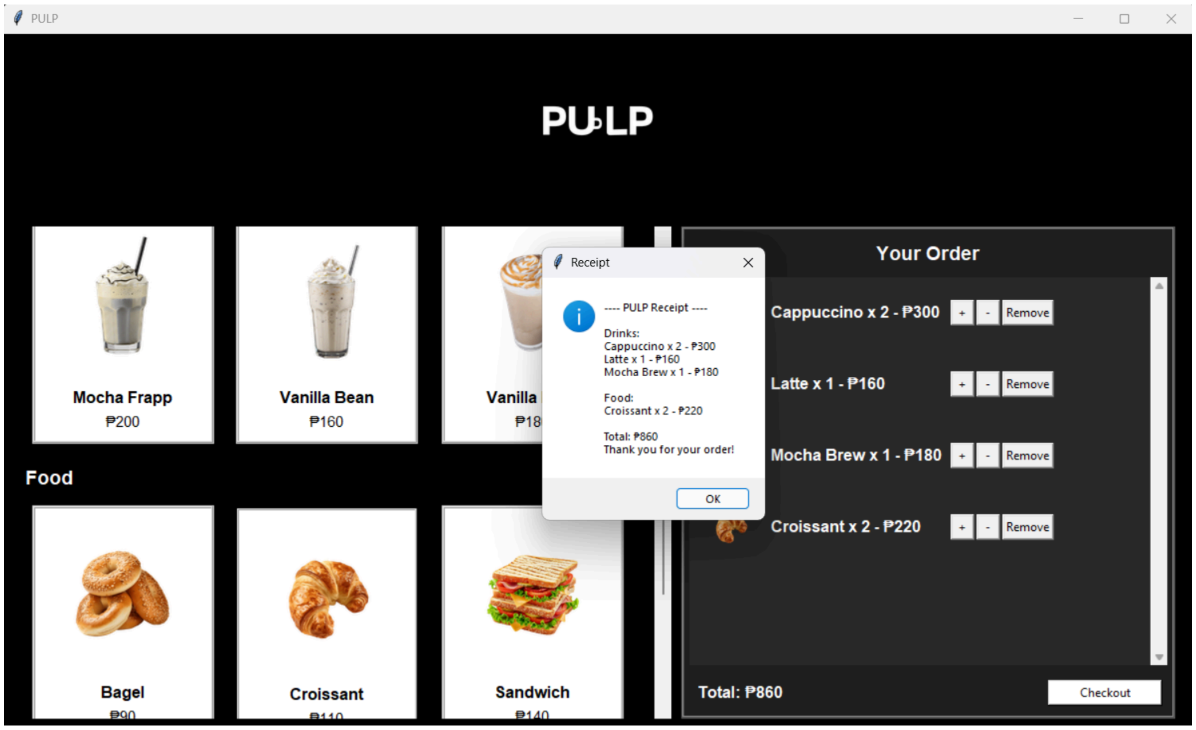
Discussion of results, issues, or limitations.

- The system successfully handles adding, removing, and updating items in real time.
- Input validation prevents incorrect or invalid selections in the backend ordering system.
- Limitations: No multi-user concurrency; GUI is single-threaded.
- Images must exist in the assets folder; missing images are replaced with a placeholder.
- Checkout prints receipt correctly, but currently does not save it externally.

Screenshot of working outputs.



```
Cappuccino x 2 - ₱300
Latte x 1 - ₱160
Mocha Brew x 1 - ₱180
Croissant x 2 - ₱220
Total: ₱860
```



```
---- PULP Receipt ----

Drinks:
Cappuccino x 2 - ₱300
Latte x 1 - ₱160
Mocha Brew x 1 - ₱180

Food:
Croissant x 2 - ₱220

Total: ₱860
Thank you for your order!
```

```
Cappuccino x 2 - ₱300
Latte x 1 - ₱160
Mocha Brew x 1 - ₱180
Croissant x 2 - ₱220
Total: ₱860
```

Ethical and Professional Reflection

- **How did your team ensure ethical collaboration (no plagiarism, fair contribution)?**
  Depoo: With the help of AI (I'm not that good at writing this code all by myself), I understood the tasks better and ensured ethical collaboration by writing all my code myself, contributing fairly, and coordinating with my team to avoid overlap or copying.
- **How does your system ensure data privacy (if applicable) and responsible programming?**
  Depoo: I ensured data privacy by handling only non-sensitive information and following safe, responsible coding practices to prevent errors or data loss.
- **What lessons can you apply from professional practice and version control ethics?**
  Depoo: I learned the importance of clear communication, proper Git usage, and documenting all changes to maintain professionalism and transparency.

Independent Learning Component

- What new concept or tool they independently learned (e.g., Git branching, threading, or modular architecture).
  Depoo: I learned how to implement modular architecture in Python by separating the backend and frontend into different files and modules. I also explored Git branching to manage my work without affecting the main project. These tools helped me understand how to structure a program professionally and track changes effectively.

Agustin: A new concept I independently learned was Git branching. In order to separate features and avoid conflicts in collaborative development, I investigated the creation, switching, merging, and management of branches. My workflow was enhanced by learning branching since it made it possible for me to safely test changes without affecting the main codebase.

- How it improved their contribution.

Depoo: Applying modular architecture allowed me to add new features like the order panel and menu system efficiently. Using Git branching helped me work safely on updates and collaborate better with my team. Overall, it made my contributions cleaner, more organized, and easier for the team to integrate.

Agustin: By enabling me to independently develop new features without affecting with the main codebase, learning Git branching enhanced my contribution. Faster and more dependable development cycles resulted from my ability to experiment safely, handle merge conflicts effectively, and work more easily with the team.

References

GeeksforGeeks. (n.d.). *Python Tkinter tutorial*. Retrieved November 21, 2025, from https://www.geeksforgeeks.org/python-gui-tkinter/

Real Python. (n.d.). *Python modules and packages – An introduction*. Retrieved November 21, 2025, from https://realpython.com/python-modules-packages/

GitHub Docs. (n.d.). *Git branching strategies*. Retrieved November 21, 2025, from https://docs.github.com/en/get-started/using-git/about-branches

Python Software Foundation. (n.d.). *Python 3 documentation*. Retrieved November 21, 2025, from https://docs.python.org/3/

Stack Overflow. (n.d.). *Tkinter scrollable frame*. Retrieved November 21, 2025, from https://stackoverflow.com/questions/16188420/tkinter-scrollbar-for-frame