VIT®

BHOPAL

Mall Billing System

Mini Project Report

Submitted By: Shubh Gupta

Reg. No.: 25BAI10562

Under The Guidance Of:

Dr. SHAHANA GAJALA QURESHI

Index

1. Problem Statement

- In small shops it is very difficult to manage all the things like managing the inventory and where the product is located , how much the product cost and how much discount will be applied to that product.
- On the other side the project not only works on the customers side it also help the owner to manage it shops well like it provide the owner the option to add the products in the store , also display the inventory of the shop, and the best thing it shows the item needed to be restocked in the shop.
- Not only this there is also the text file attached which shows all the activity like if customer taken something form the shop and the bill is generated then in the .txt file it will save the date time the product purchase and then reduce form the owner stock this make the flow of the purchasing and selling very smooth

Goal: Billing and inventory management in a small-scale retail environment using a lightweight Python application.

2. Objectives

The main objectives of the **Mall Billing System** project are:

1. **Automate Billing Process**
    - To reduce the manual error that make the owner benefits of not mistaking and the customer can fastly do the billing
2. **Efficient Inventory Management**
    - To allow the owner to see the inventory , if wanted to add some product and want to check the items needed to be restocked
3. **Monitor Stock Levels**
    - To provide alerts for low-stock items so that owners can replenish inventory in time and avoid stockouts.
4. **Maintain Records**
    - To store all inventory updates, owner actions, and customer bills in text files for future reference and reporting.
5. **User-Friendly Interface**
    - To provide a terminal interface that can be used by both store owners and customers with minimal technical knowledge.
6. **Facilitate Customer Interaction**
    - To allow customers to check prices, locate products, and generate bills efficiently with discounts applied automatically.

4. Working & Explanation

The **Mall Billing System** is a console-based Python application designed to handle both **owner and customer operations** efficiently. It uses **text files** for data storage, ensuring all transactions and inventory updates are saved.

1. System Startup

- When the program starts, it checks if the data folder and necessary files (products.txt, customers.txt, owner_logs.txt) exist.
- If not, it automatically creates the folder and files to store inventory, bills, and owner logs.

2. Owner Module

- Accessible from the main menu by selecting **Owner**.
- **Features:**

1. **Add New Product:**
    - Owner can input product name, price, initial stock, discount percentage, and storage location.
    - Data is saved in products.txt and logged in owner_logs.txt.
2. **Replenish Stock:**
    - Owner can update stock quantity for existing products.
    - Updates are saved in products.txt and logged.

3. **Show Inventory:**
    - Displays all products with price, quantity, discount, and location.
    - Fetches data from products.txt.
4. **Check Low Stock:**
    - Alerts the owner for products with stock ≤ 5 units to prevent running out of items.

3. Customer Module

- Accessible from the main menu by selecting **Customer**.
- **Features:**
    1. **Buy Products / Generate Bill:**
        - Customer selects products and quantity.
        - Program checks available stock and applies discount automatically.
        - Generates a bill and saves it in customers.txt with date, time, payment method, and customer name.
    2. **Check Product Price:**
        - Customer can view the price and discount of any product.
        - Information is retrieved from products.txt.

3. **Locate Product:**
    - Customer can find the storage location (aisle) of any product in the store.

4. Data Management

- **Inventory (products.txt)**
    - **Stores product details in the format:**ProductName|Price|Quantity|Discount|Location
    - **Customer Bills (customers.txt)**
        - Stores detailed bills with timestamp, products purchased, quantities, discounts, total amount, and payment method.
    - **Owner Logs (owner_logs.txt)**
        - Records all actions performed by the owner for accountability.
- **Customer Bills (customers.txt)**
    - Stores detailed bills with timestamp, products purchased, quantities, discounts, total amount, and payment method.
- **Owner Logs (owner_logs.txt)**
    - Records all actions performed by the owner.

5. Program Flow

1. User selects **Owner** or **Customer** from the main menu.

2. Owner performs inventory management operations, or customer makes purchases.

3. All updates and bills are immediately saved to their respective text files.

4. The program loops back to the main menu until the user chooses to exit.

6. Key Benefits

- Reduces manual errors in billing and stock management.

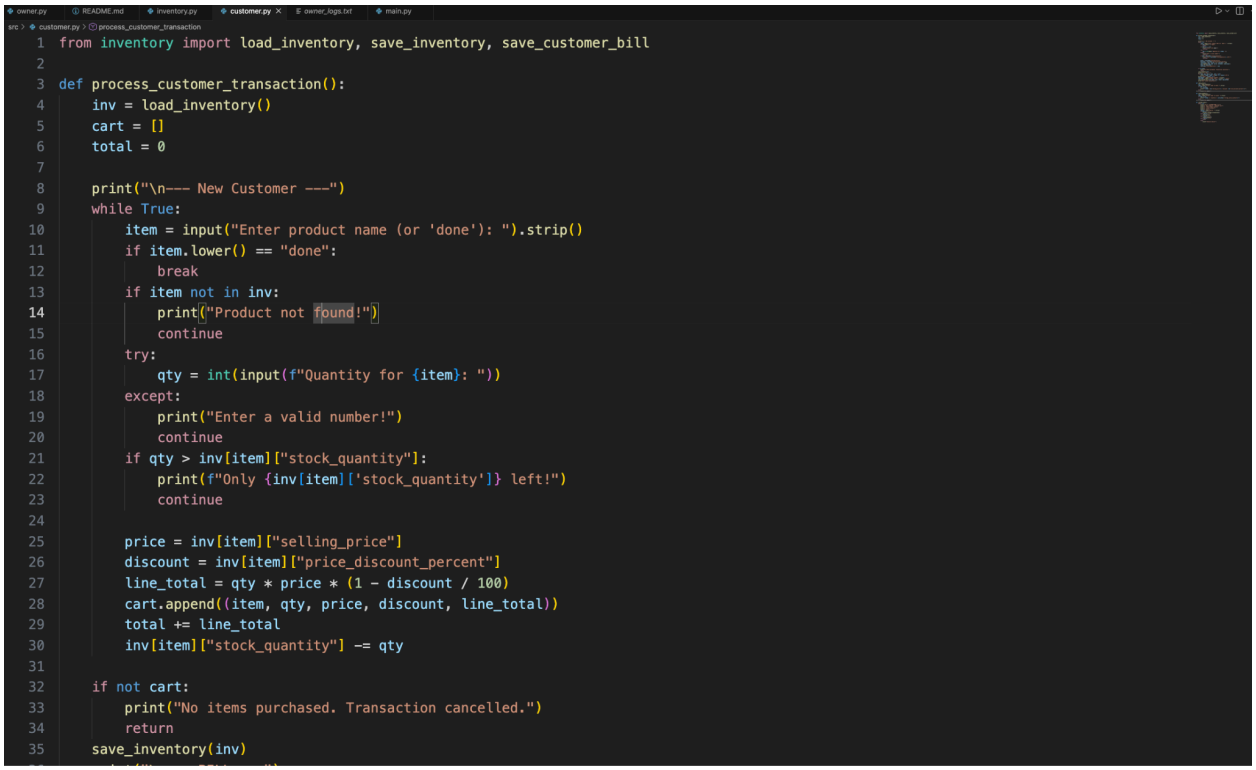
- Provides real-time stock updates and alerts.

- Maintains permanent records of transactions and inventory.

- Simple console interface suitable for small retail environments.

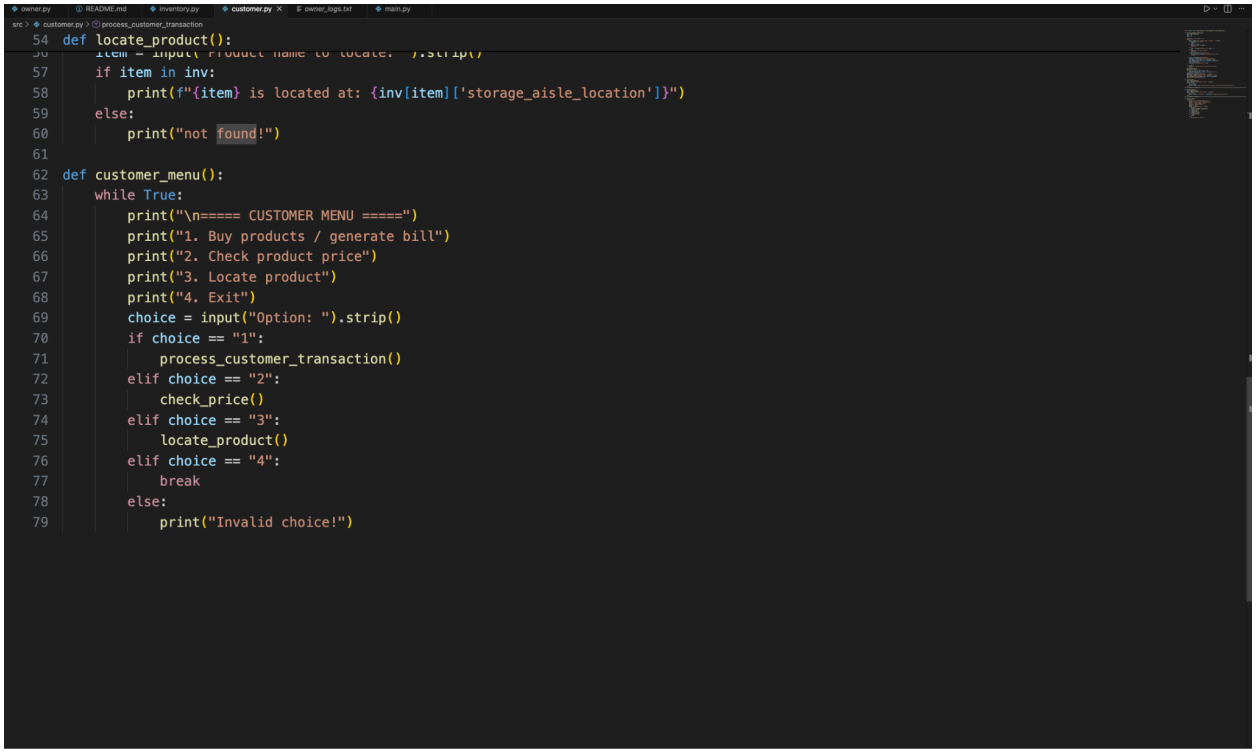
5. Full Source Code

- **Customer.py**



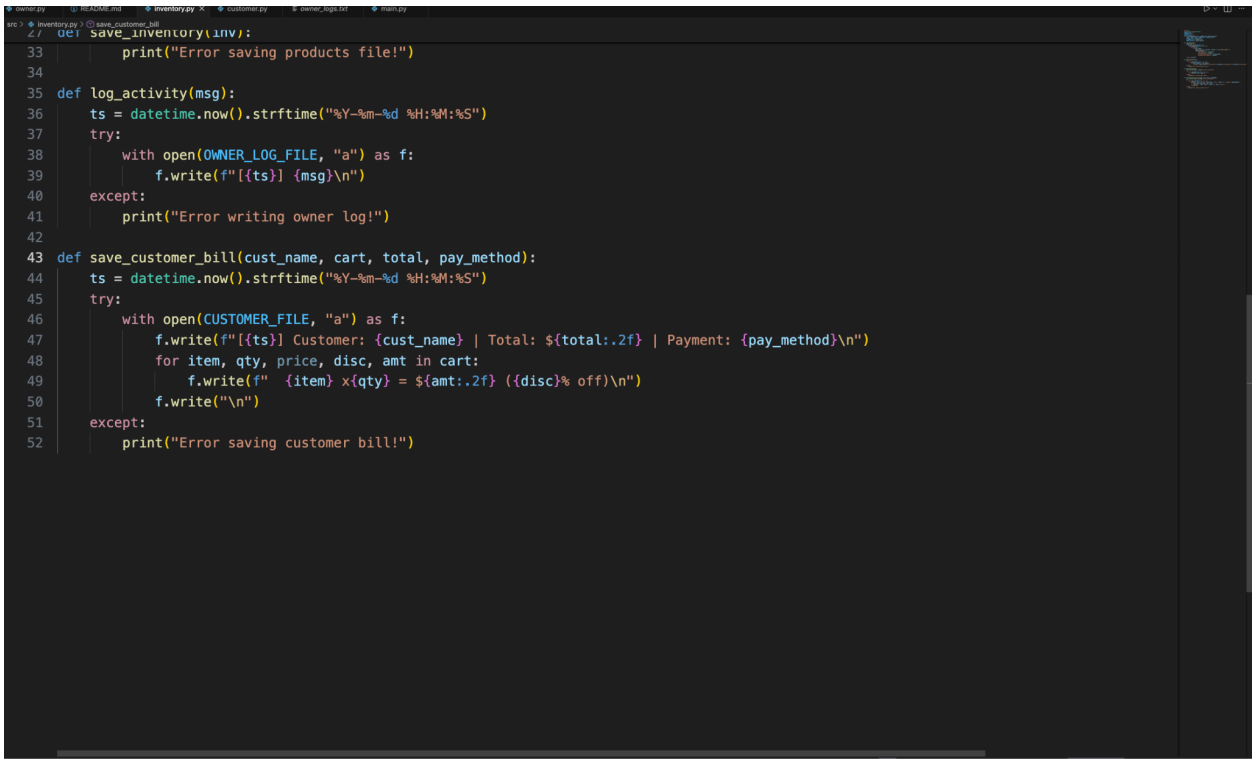
```python
from inventory import load_inventory, save_inventory, save_customer_bill

def process_customer_transaction():
    inv = load_inventory()
    cart = []
    total = 0

    print("\n--- New Customer ---")
    while True:
        item = input("Enter product name (or 'done'): ").strip()
        if item.lower() == "done":
            break
        if item not in inv:
            print("Product not found!")
            continue
        try:
            qty = int(input(f"Quantity for {item}: "))
        except:
            print("Enter a valid number!")
            continue
        if qty > inv[item]["stock_quantity"]:
            print(f"Only {inv[item]['stock_quantity']} left!")
            continue

        price = inv[item]["selling_price"]
        discount = inv[item]["price_discount_percent"]
        line_total = qty * price * (1 - discount / 100)
        cart.append((item, qty, price, discount, line_total))
        total += line_total
        inv[item]["stock_quantity"] -= qty

    if not cart:
        print("No items purchased. Transaction cancelled.")
        return
    save_inventory(inv)
```

```python
def process_customer_transaction():
        print("No items purchased. Transaction cancelled.")
        return
    save_inventory(inv)
    print("\n--- BILL ---")
    for name, qty, price, disc, amt in cart:
        print(f"{name} x{qty} = ${amt:.2f} ({disc}% off)")
    print(f"TOTAL: ${total:.2f}")
    pay_method = input("Payment method: ").strip()
    cust_name = input("Customer name: ").strip() or "Guest"
    save_customer_bill(cust_name, cart, total, pay_method)
    print("Bill saved successfully")

def check_price():
    inv = load_inventory()
    item = input("Product name to check: ").strip()
    if item in inv:
        d = inv[item]
        print(f"{item}: ${d['selling_price']}, Discount: {d['price_discount_percent']}%")
    else:
        print("not found!")

def locate_product():
    inv = load_inventory()
    item = input("Product name to locate: ").strip()
    if item in inv:
        print(f"{item} is located at: {inv[item]['storage_aisle_location']}")
    else:
        print("not found!")

def customer_menu():
    while True:
        print("\n===== CUSTOMER MENU =====")
        print("1. Buy products / generate bill")
        print("2. Check product price")
        print("3. Locate product")
```


owner.py
README.md
inventory.py
customer.py
owner_logs.txt
main.py
src > customer.py > process_customer_transaction
def locate_product():
item = input("Product name to locate: ").strip()
if item in inv:
print(f"{item} is located at: {inv[item]['storage_aisle_location']}")
else:
print("not found!")
def customer_menu():
while True:
print("\n===== CUSTOMER MENU =====")
print("1. Buy products / generate bill")
print("2. Check product price")
print("3. Locate product")
print("4. Exit")
choice = input("Option: ").strip()
if choice == "1":
process_customer_transaction()
elif choice == "2":
check_price()
elif choice == "3":
locate_product()
elif choice == "4":
break
else:
print("Invalid choice!")

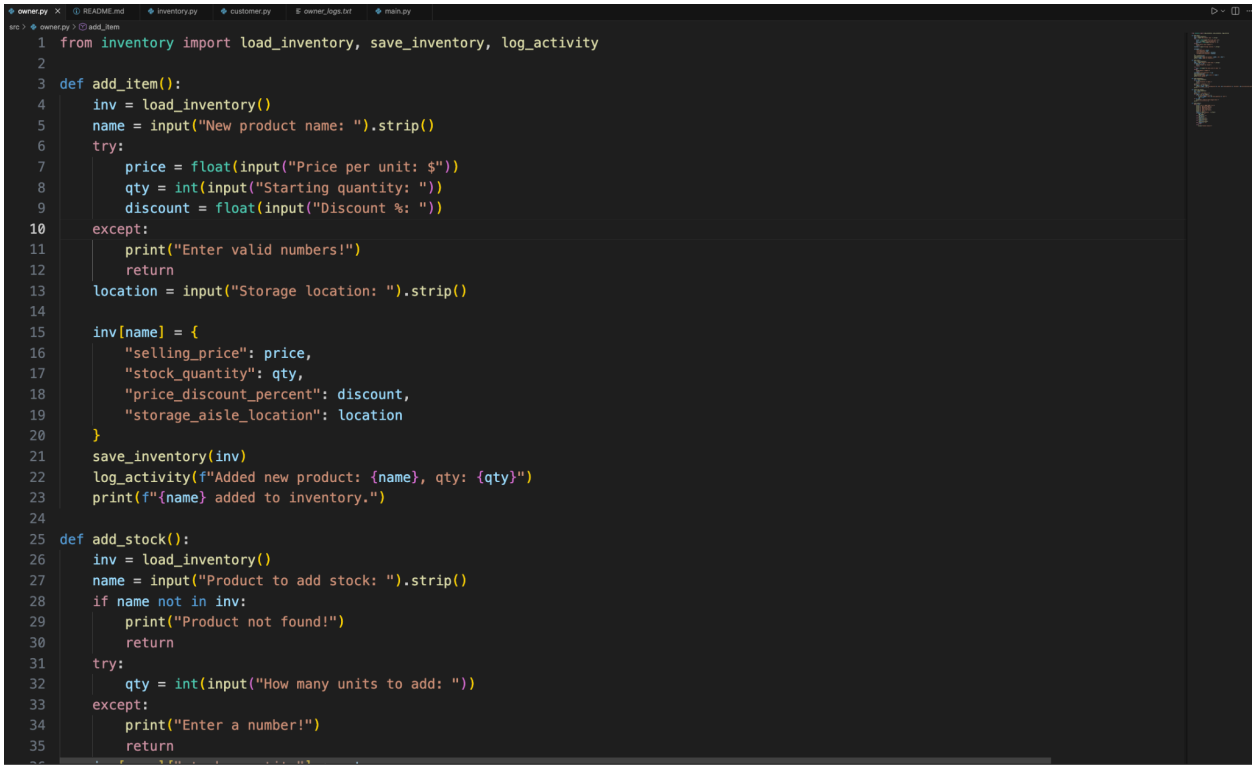
- **Inventory.py**

```python
import os
from datetime import datetime
(constant) CUSTOMER_FILE: Literal['']
CUSTOMER_FILE = ""
OWNER_LOG_FILE = ""
def set_paths(product_file, customer_file, owner_log_file):
    global PRODUCT_FILE, CUSTOMER_FILE, OWNER_LOG_FILE
    PRODUCT_FILE = product_file
    CUSTOMER_FILE = customer_file
    OWNER_LOG_FILE = owner_log_file

def load_inventory():
    inventory = {}
    if os.path.exists(PRODUCT_FILE):
        with open(PRODUCT_FILE, "r") as f:
            for line in f:
                if line.strip():
                    name, price, qty, discount, location = line.strip().split("|")
                    inventory[name] = {
                        "selling_price": float(price),
                        "stock_quantity": int(qty),
                        "price_discount_percent": float(discount),
                        "storage_aisle_location": location
                    }
    return inventory

def save_inventory(inv):
    try:
        with open(PRODUCT_FILE, "w") as f:
            for name, details in inv.items():
                f.write(f"{name}|{details['selling_price']}|{details['stock_quantity']}|{details['price_discount_percent']}|{details['stor
    except:
        print("Error saving products file!")

def log_activity(msg):
```

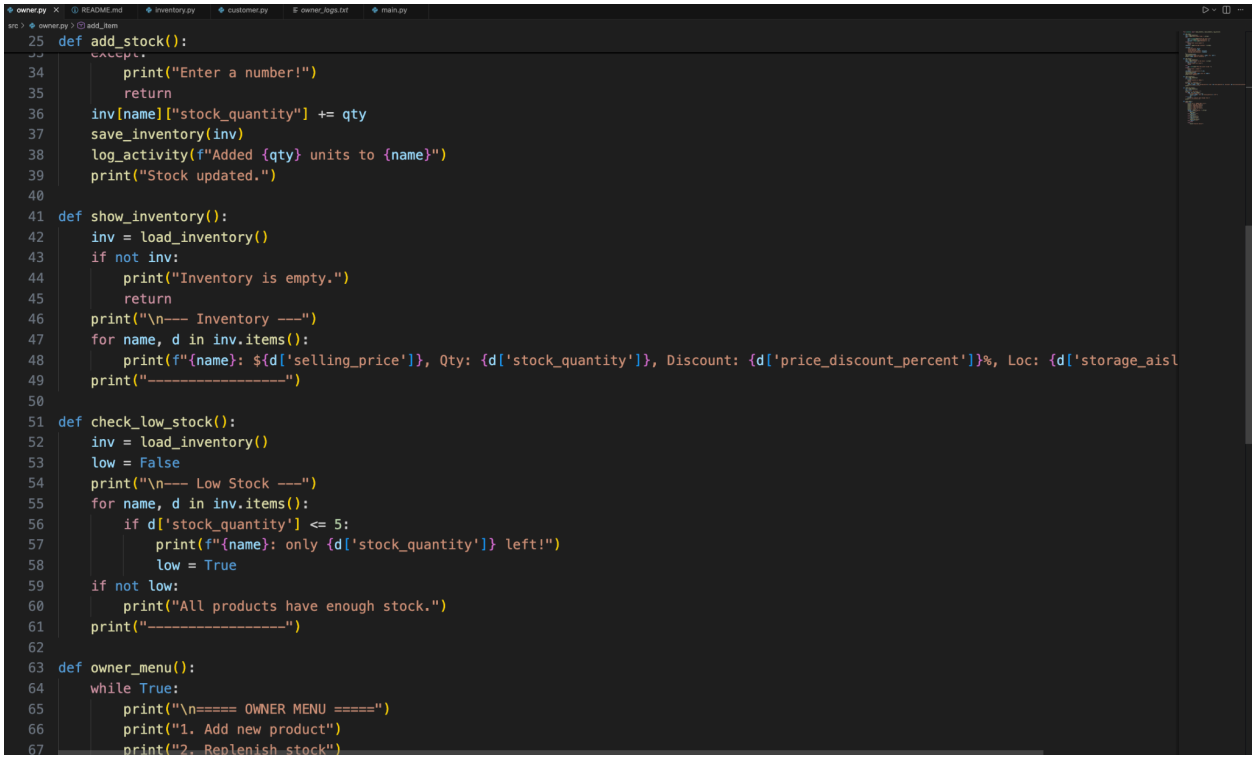


```python
def save_inventory(inv):
        print("Error saving products file!")

def log_activity(msg):
    ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    try:
        with open(OWNER_LOG_FILE, "a") as f:
            f.write(f"[{ts}] {msg}\n")
    except:
        print("Error writing owner log!")

def save_customer_bill(cust_name, cart, total, pay_method):
    ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    try:
        with open(CUSTOMER_FILE, "a") as f:
            f.write(f"[{ts}] Customer: {cust_name} | Total: ${total:.2f} | Payment: {pay_method}\n")
            for item, qty, price, disc, amt in cart:
                f.write(f"  {item} x{qty} = ${amt:.2f} ({disc}% off)\n")
            f.write("\n")
    except:
        print("Error saving customer bill!")
```

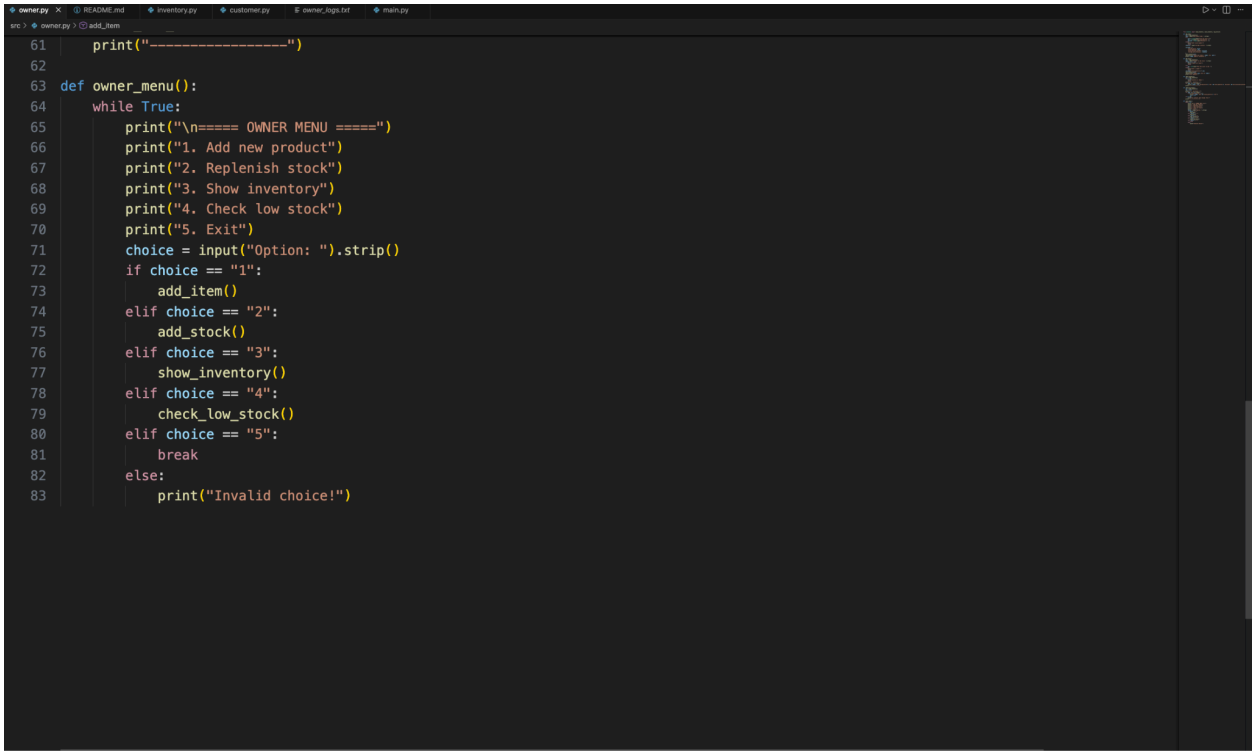
- **Owner.py**



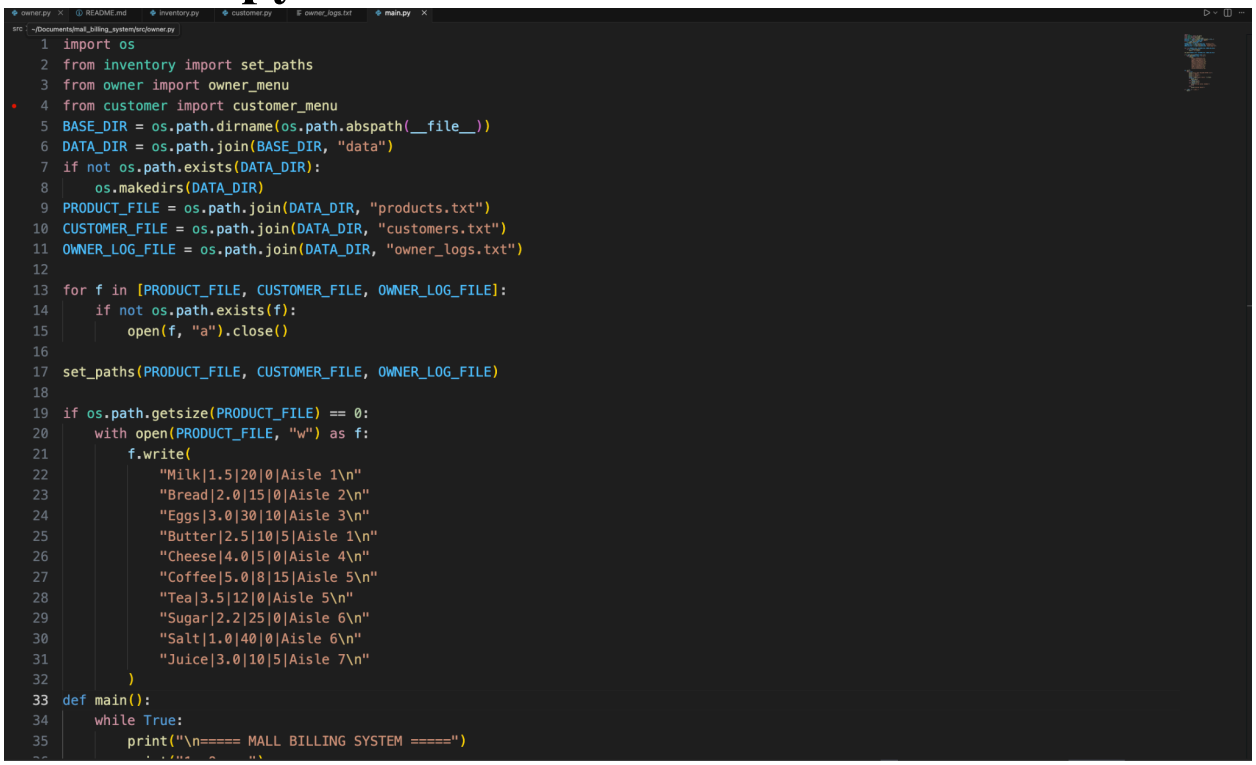
```python
from inventory import load_inventory, save_inventory, log_activity

def add_item():
    inv = load_inventory()
    name = input("New product name: ").strip()
    try:
        price = float(input("Price per unit: $"))
        qty = int(input("Starting quantity: "))
        discount = float(input("Discount %: "))
    except:
        print("Enter valid numbers!")
        return
    location = input("Storage location: ").strip()

    inv[name] = {
        "selling_price": price,
        "stock_quantity": qty,
        "price_discount_percent": discount,
        "storage_aisle_location": location
    }
    save_inventory(inv)
    log_activity(f"Added new product: {name}, qty: {qty}")
    print(f"{name} added to inventory.")

def add_stock():
    inv = load_inventory()
    name = input("Product to add stock: ").strip()
    if name not in inv:
        print("Product not found!")
        return
    try:
        qty = int(input("How many units to add: "))
    except:
        print("Enter a number!")
        return
```



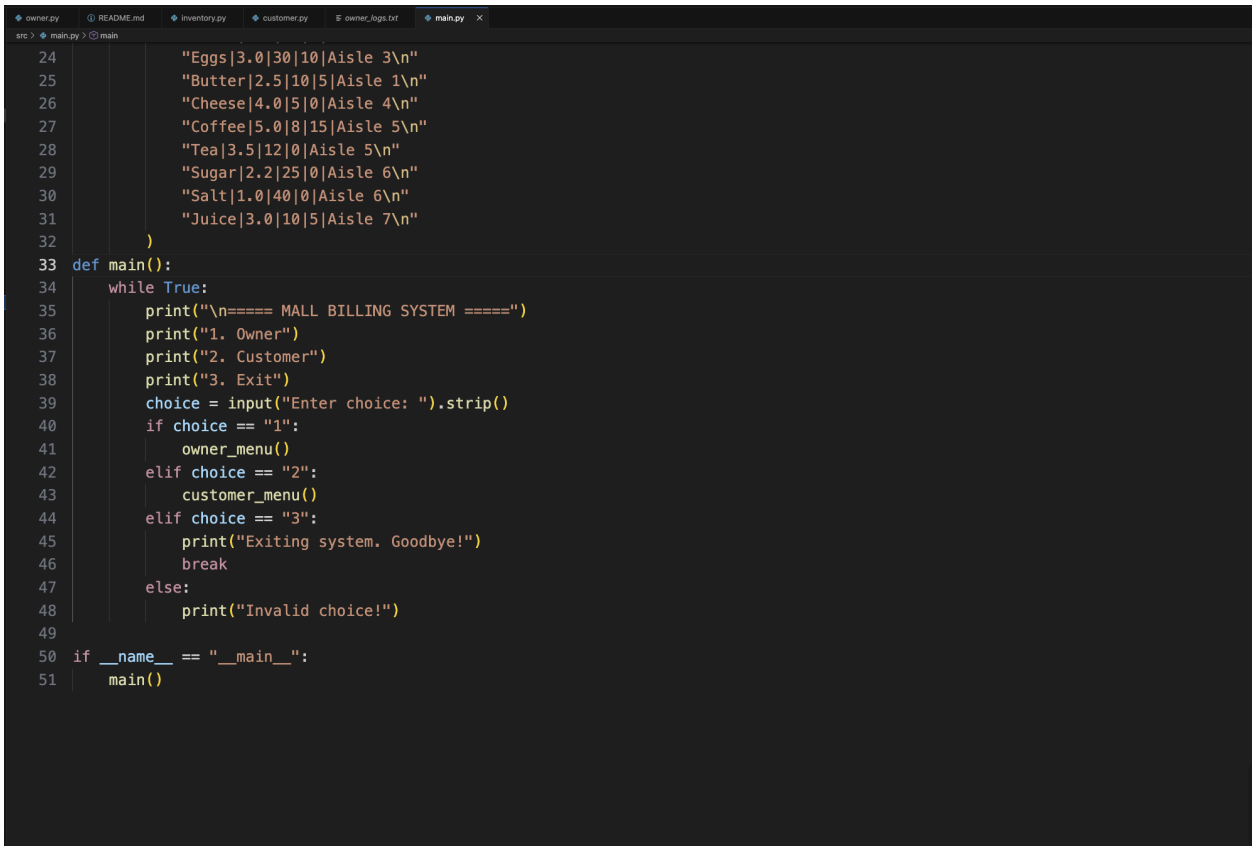
```python
def add_stock():
        print("Enter a number!")
        return
    inv[name]["stock_quantity"] += qty
    save_inventory(inv)
    log_activity(f"Added {qty} units to {name}")
    print("Stock updated.")

def show_inventory():
    inv = load_inventory()
    if not inv:
        print("Inventory is empty.")
        return
    print("\n--- Inventory ---")
    for name, d in inv.items():
        print(f"{name}: ${d['selling_price']}, Qty: {d['stock_quantity']}, Discount: {d['price_discount_percent']}%, Loc: {d['storage_aisl
    print("-----------------")

def check_low_stock():
    inv = load_inventory()
    low = False
    print("\n--- Low Stock ---")
    for name, d in inv.items():
        if d['stock_quantity'] <= 5:
            print(f"{name}: only {d['stock_quantity']} left!")
            low = True
    if not low:
        print("All products have enough stock.")
    print("-----------------")

def owner_menu():
    while True:
        print("\n===== OWNER MENU =====")
        print("1. Add new product")
        print("2. Replenish stock")
```


owner.py
README.md
inventory.py
customer.py
owner_logs.txt
main.py
src > owner.py > add_item
61 print("——————————————————")
62
63 def owner_menu():
64 while True:
65 print("\n===== OWNER MENU =====")
66 print("1. Add new product")
67 print("2. Replenish stock")
68 print("3. Show inventory")
69 print("4. Check low stock")
70 print("5. Exit")
71 choice = input("Option: ").strip()
72 if choice == "1":
73 add_item()
74 elif choice == "2":
75 add_stock()
76 elif choice == "3":
77 show_inventory()
78 elif choice == "4":
79 check_low_stock()
80 elif choice == "5":
81 break
82 else:
83 print("Invalid choice!")

- **Main.py**



```python
import os
from inventory import set_paths
from owner import owner_menu
from customer import customer_menu
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_DIR = os.path.join(BASE_DIR, "data")
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
PRODUCT_FILE = os.path.join(DATA_DIR, "products.txt")
CUSTOMER_FILE = os.path.join(DATA_DIR, "customers.txt")
OWNER_LOG_FILE = os.path.join(DATA_DIR, "owner_logs.txt")

for f in [PRODUCT_FILE, CUSTOMER_FILE, OWNER_LOG_FILE]:
    if not os.path.exists(f):
        open(f, "a").close()

set_paths(PRODUCT_FILE, CUSTOMER_FILE, OWNER_LOG_FILE)

if os.path.getsize(PRODUCT_FILE) == 0:
    with open(PRODUCT_FILE, "w") as f:
        f.write(
            "Milk|1.5|20|0|Aisle 1\n"
            "Bread|2.0|15|0|Aisle 2\n"
            "Eggs|3.0|30|10|Aisle 3\n"
            "Butter|2.5|10|5|Aisle 1\n"
            "Cheese|4.0|5|0|Aisle 4\n"
            "Coffee|5.0|8|15|Aisle 5\n"
            "Tea|3.5|12|0|Aisle 5\n"
            "Sugar|2.2|25|0|Aisle 6\n"
            "Salt|1.0|40|0|Aisle 6\n"
            "Juice|3.0|10|5|Aisle 7\n"
        )
def main():
    while True:
        print("\n===== MALL BILLING SYSTEM =====")
```



```python
            "Eggs|3.0|30|10|Aisle 3\n"
            "Butter|2.5|10|5|Aisle 1\n"
            "Cheese|4.0|5|0|Aisle 4\n"
            "Coffee|5.0|8|15|Aisle 5\n"
            "Tea|3.5|12|0|Aisle 5\n"
            "Sugar|2.2|25|0|Aisle 6\n"
            "Salt|1.0|40|0|Aisle 6\n"
            "Juice|3.0|10|5|Aisle 7\n"
        )
def main():
    while True:
        print("\n===== MALL BILLING SYSTEM =====")
        print("1. Owner")
        print("2. Customer")
        print("3. Exit")
        choice = input("Enter choice: ").strip()
        if choice == "1":
            owner_menu()
        elif choice == "2":
            customer_menu()
        elif choice == "3":
            print("Exiting system. Goodbye!")
            break
        else:
            print("Invalid choice!")

if __name__ == "__main__":
    main()
```

7. Conclusion

The **Mall Billing System** is a simple yet effective console-based application designed to **automate billing and inventory management** for small retail stores or malls.

Through this project:

- Store owners can **easily manage inventory**, monitor stock levels, and log all actions.
- Customers can **purchase products efficiently**, check prices, and locate items in the store.
- All **transactions, inventory updates, and billing details** are saved persistently in text files for future reference.

This project demonstrates **fundamental concepts of Python programming**, including **file handling, modular coding, loops, conditionals, and user input handling**.

The system is **scalable and can be enhanced further** with features like a GUI, category-wise inventory, search functionality, and multi-customer billing.

Overall, the project **reduces manual errors, saves time, and provides a reliable record-keeping system**, making it a practical solution for small retail businesses.

8. References

Python Official Documentation – https://docs.python.org/3/
Used for understanding file handling, datetime, and basic Python syntax.

GeeksforGeeks – Python File Handling ://www.geeksforgeeks.org/file-handling-python/
Reference for reading, writing, and appending data to .txt files.

W3Schools – Python Tutorial – https://www.w3schools.com/python/
For learning Python loops, conditionals, and user input.

Stack Overflow – https://stackoverflow.com/
Used for troubleshooting errors and learning best practices for Python projects.

YouTube Tutorials – Various Python project tutorials
Helpful for understanding modular programming and project structure.

B.Tech Project Reports of Previous Years
For reference on formatting, documentation style, and report structure.