Project Report: Smart Inventory and Billing System

1. GitHub Repository

Project Source Code:
https://github.com/AdiChakote/ds-billingsystem

This repository contains the complete Python implementation of the **Smart Inventory and Billing System**, including inventory management, billing logic, and role-based access for customers and shopkeepers.

2. Introduction

The **Smart Inventory and Billing System** is a console-based Python application designed to simulate a basic **retail store environment**.
The system supports two roles:

- **Customer** – for purchasing items and generating bills
- **Shopkeeper** – for managing stock and adding new items

This project demonstrates real-world concepts such as **inventory management, billing logic, discounts, and role-based access** using Python.

---

3. Objective

The main objectives of this project are:

- To manage store inventory using Python dictionaries
- To allow customers to purchase items and generate bills
- To provide discount logic based on total purchase amount
- To allow shopkeepers to view and update stock
- To demonstrate modular programming using functions

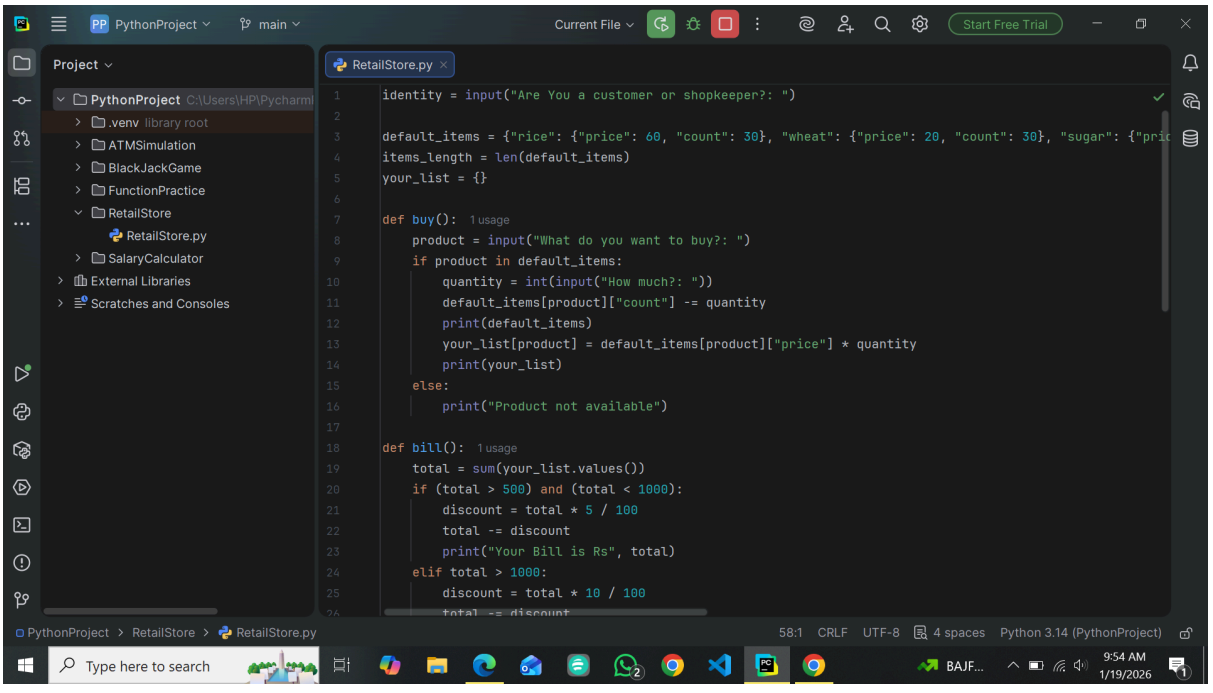
---

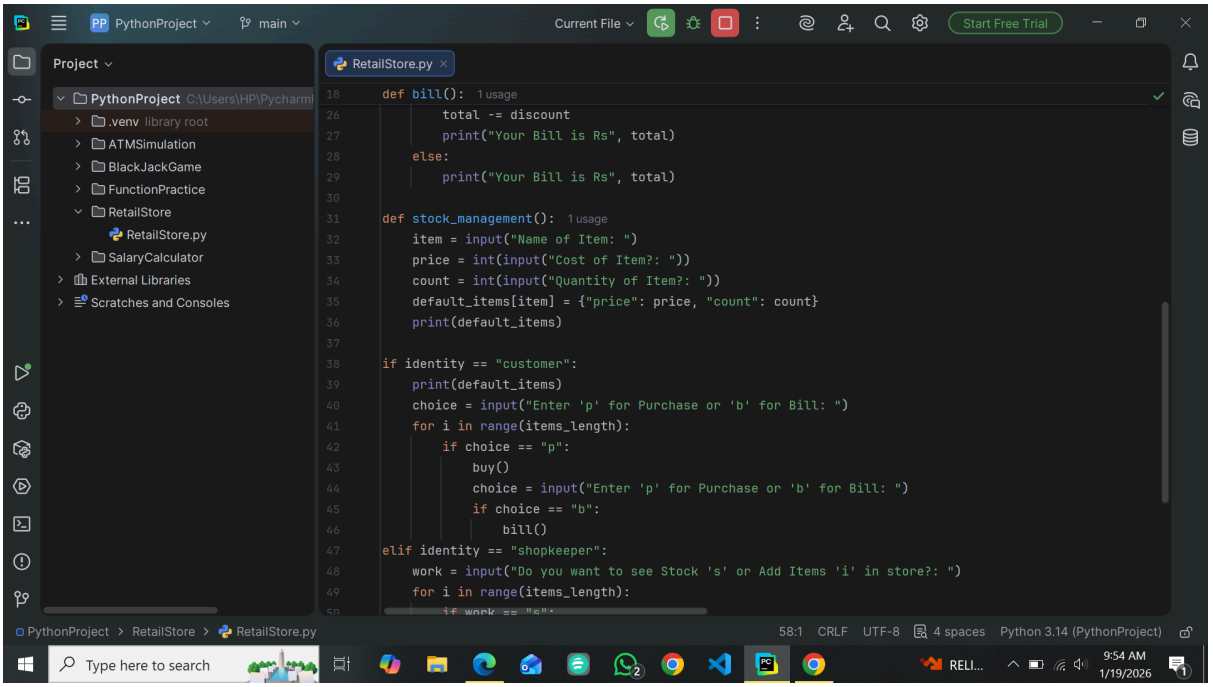
4. Technologies Used

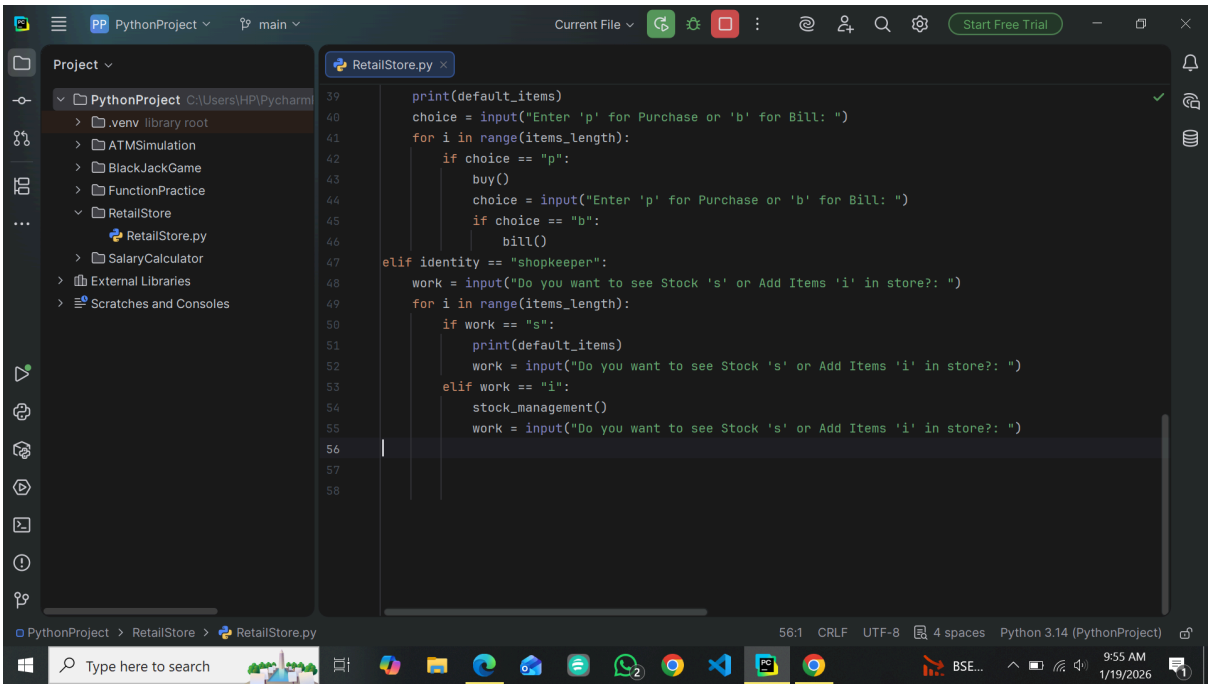
- **Programming Language:** Python
- **Concepts Applied:**
    - Dictionaries (nested dictionaries)
    - Functions
    - Conditional statements
    - Loops
    - User input/output
- **Interface:** Command Line (CLI)

---

5. Source Code







📸 **Screenshot: Smart Inventory and Billing System Code**

---

6. Code Overview

```
identity = input("Are You a customer or shopkeeper?: ")
```

```
default_items = {
    "rice": {"price": 60, "count": 30},
    "wheat": {"price": 20, "count": 30},
    "sugar": {"price": 40, "count": 70},
    "ghee": {"price": 600, "count": 40},
    "soap": {"price": 30, "count": 100},
    "toothpaste": {"price": 50, "count": 80}
}

your_list = {}
```

---

7. Data Structure Explanation

7.1 Inventory Dictionary (`default_items`)

- Stores product name as key
- Each product has:
    - `price`
    - `count` (available stock)

Example:

```
"rice": {"price": 60, "count": 30}
```

7.2 Cart Dictionary (`your_list`)

- Stores purchased items and their total cost
- Used for bill calculation

---

8. Function-wise Explanation

8.1 buy() Function

```
def buy():
    product = input("What do you want to buy?: ")
    if product in default_items:
        quantity = int(input("How much?: "))
        default_items[product]["count"] -= quantity
        your_list[product] = default_items[product]["price"] *
quantity
    else:
        print("Product not available")
```

Purpose:

- Allows customer to purchase products

Logic:

- Checks if product exists
- Reduces stock count
- Adds item to customer cart

8.2 bill() Function

```
def bill():
    total = sum(your_list.values())
    if (total > 500) and (total < 1000):
        discount = total * 5 / 100
        total -= discount
    elif total > 1000:
        discount = total * 10 / 100
        total -= discount
    print("Your Bill is Rs", total)
```

Purpose:

- Calculates final bill amount

Discount Rules:

- ₹500 – ₹999 → **5% discount**
- Above ₹1000 → **10% discount**
- Below ₹500 → No discount

8.3 stock_management() Function

```
def stock_management():
    item = input("Name of Item: ")
    price = int(input("Cost of Item?: "))
    count = int(input("Quantity of Item?: "))
    default_items[item] = {"price": price, "count": count}
```

Purpose:

- Allows shopkeeper to add or update items in inventory

---

9. Role-Based Logic

9.1 Customer Flow

```
if identity == "customer":
```

Customer can:

- View available items
- Purchase products
- Generate bill

9.2 Shopkeeper Flow

```
elif identity == "shopkeeper":
```

Shopkeeper can:

- View stock
- Add new items
- Update inventory

---

10. Sample Execution

Example:

```
Are You a customer or shopkeeper?: customer

What do you want to buy?: rice

How much?: 5

Your Bill is Rs 300
```

---

11. Features of the System

- Role-based access (Customer / Shopkeeper)
- Real-time stock update
- Discount-based billing system
- Modular and reusable functions
- Dictionary-based inventory storage

---

12. Limitations

- No validation for negative or excess quantity
- No loop exit option
- Stock can go negative
- No persistent storage (data resets on restart)
- No authentication for shopkeeper

---

13. Future Scope / Enhancements

The system can be enhanced by:

1. **Input Validation**

- Prevent negative values
- Check stock availability before purchase

2. **Persistent Storage**
    - Save data using files or databases
3. **Multiple Customers**
    - Support multiple users and sessions
4. **Authentication System**
    - Login system for shopkeeper
5. **Advanced Billing**
    - GST, tax, invoice generation
6. **GUI / Web Application**
    - Convert to Tkinter / Flask / Django app

---

14. Conclusion

The Smart Inventory and Billing System effectively demonstrates real-world retail logic using Python. It applies core programming concepts such as dictionaries, functions, and conditional logic in a practical scenario. This project serves as a strong foundation for building more advanced inventory and billing systems.