****

**Department of Computer science and engineering**

Project report on

“**Grocery Shop Billing System**”

Submitted by

**Prashanth S**

Under the guidance of

**Mr Harsha Vardhan**

**Signature**

**Table of contents**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Content** | **Page.no** |
| 1. | Introduction | 3 |
| 2. | Objective | 4 |
| 3. | Scope | 4 |
| 4. | System design and Methodology | 5 |
| 5. | Features and modules | 6 |
| 6. | Technologies used | 6 |
| 7. | Results And discussion  And screenshots | 7-9 |
| 8. | Future enhancement | 10 |
| 9. | References | 10 |
| 10. | Code | 11-18 |
| 11. | Conclusion | 19 |

**Introduction:**

The Grocery Shop Billing System is a Python-based project to automate inventory and billing.  
It provides a web interface for managing products, prices, and stock levels.  
The backend is built with **FastAPI**, and the frontend uses **Jinja2** templates.  
Products can be added, updated, and listed in the inventory.  
The billing module supports cart management, discounts, and checkout.  
Data is stored persistently in a **JSON file (inventory.json)**.  
The system demonstrates CRUD operations with real-time stock updates.  
It is lightweight and suitable for small to medium-sized grocery shops.

**Objective**

- Automate inventory management.  
- Simplify billing operations.  
- Apply discounts dynamically.  
- Provide a user-friendly web interface.  
- Maintain persistent storage using JSON files.

**Scope**

The project focuses on simplifying billing and stock management in grocery shops. It is suitable for small to medium-sized businesses. The system supports inventory updates, dynamic billing, discounts, and persistent storage. It can be scaled further by integrating with databases and advanced reporting features.

**System Design and Methodology**

The system follows a modular design with the following key components:  
  
1. **models.py** – Contains Product, Inventory, and Bill classes.

2. **main.py** – Implements FastAPI routes for inventory, billing, and checkout.

3. **templates/** – HTML templates (base.html, index.html, inventory.html, billing.html).

4. **static/** – CSS styles. (kept as optional)

5. **inventory.json** – Stores product data persistently.

**Features and Modules**

1. Inventory Module – Add, update, and list products.  
2. Billing Module – Add products to cart, calculate totals.  
3. Discount Module – Apply discounts before checkout.  
4. Checkout Module – Finalize purchase and clear cart.

**Technologies Used**

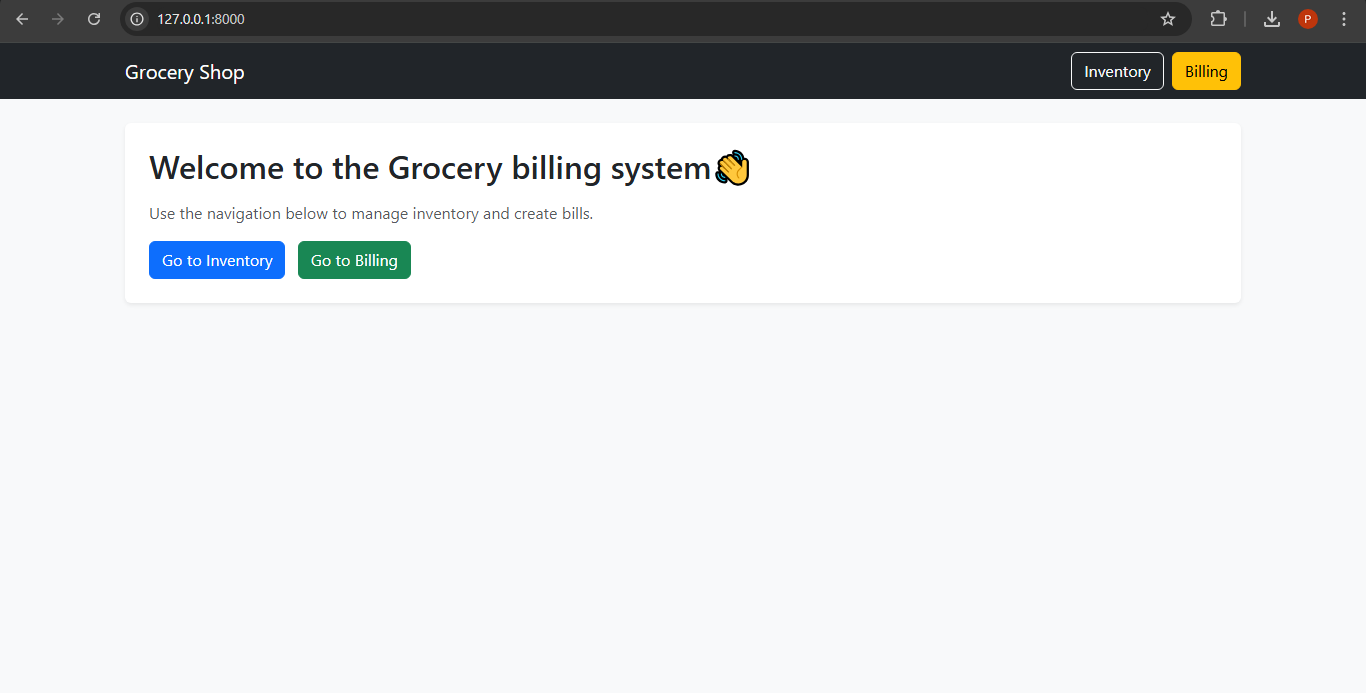
- Python 3  
- FastAPI (Backend)  
- Uvicorn (Server)  
- Jinja2 (Templating)  
- Bootstrap (UI)  
- JSON (Data storage)

**Results and Discussion**

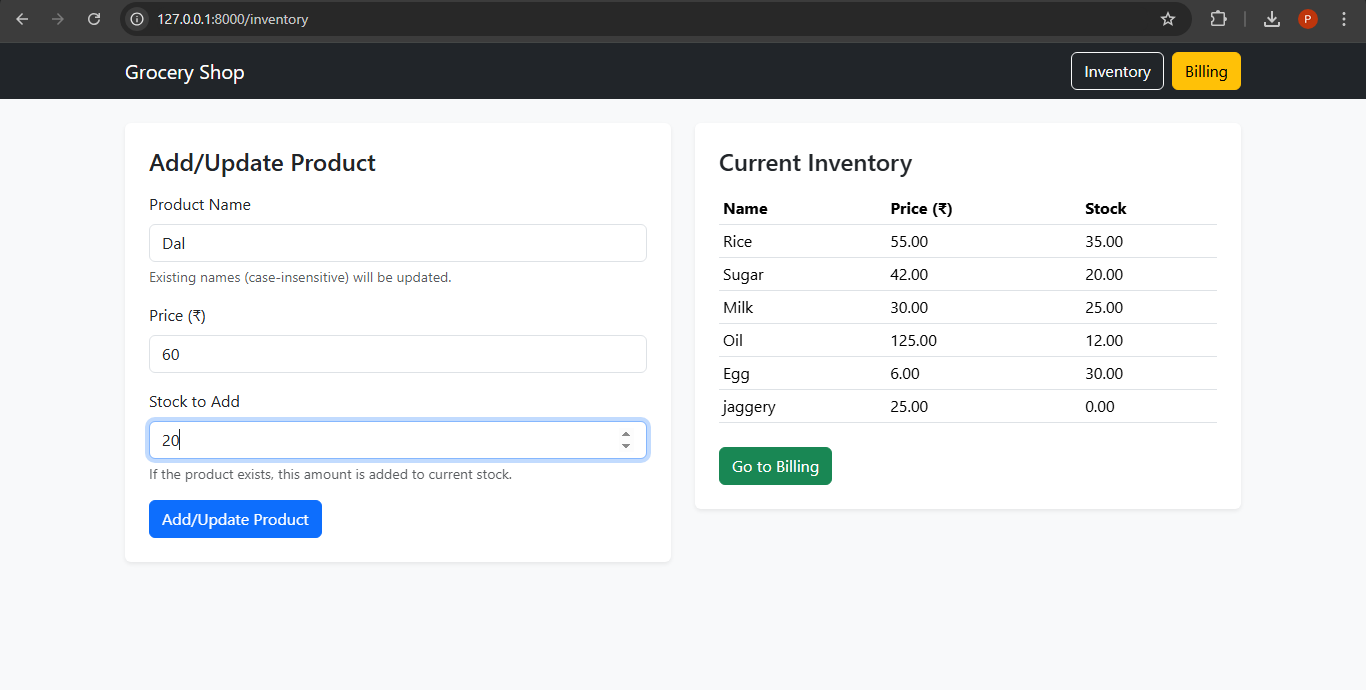
The system was tested successfully. Products were added and updated in inventory, reflected in inventory.json. Items were added to the cart, discounts applied, and final totals generated correctly. The FastAPI application rendered templates smoothly, providing an intuitive UI.

**Screenshots**

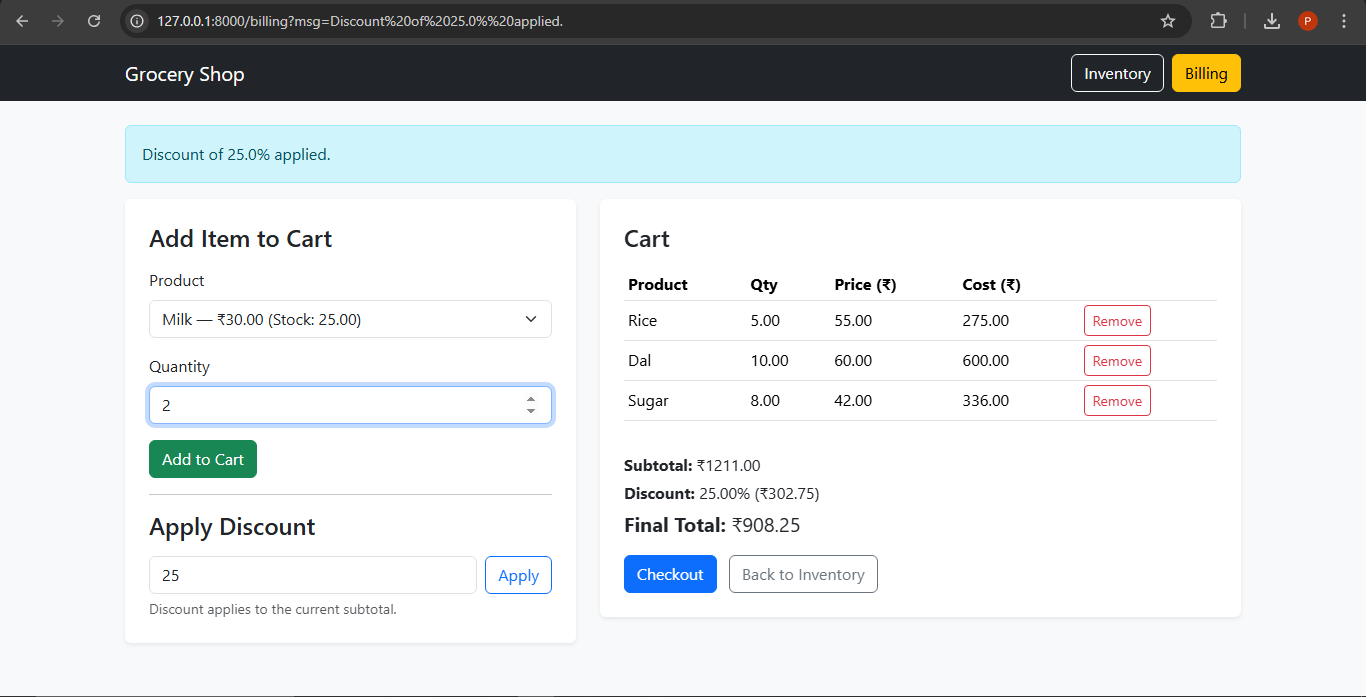
**Home page –**



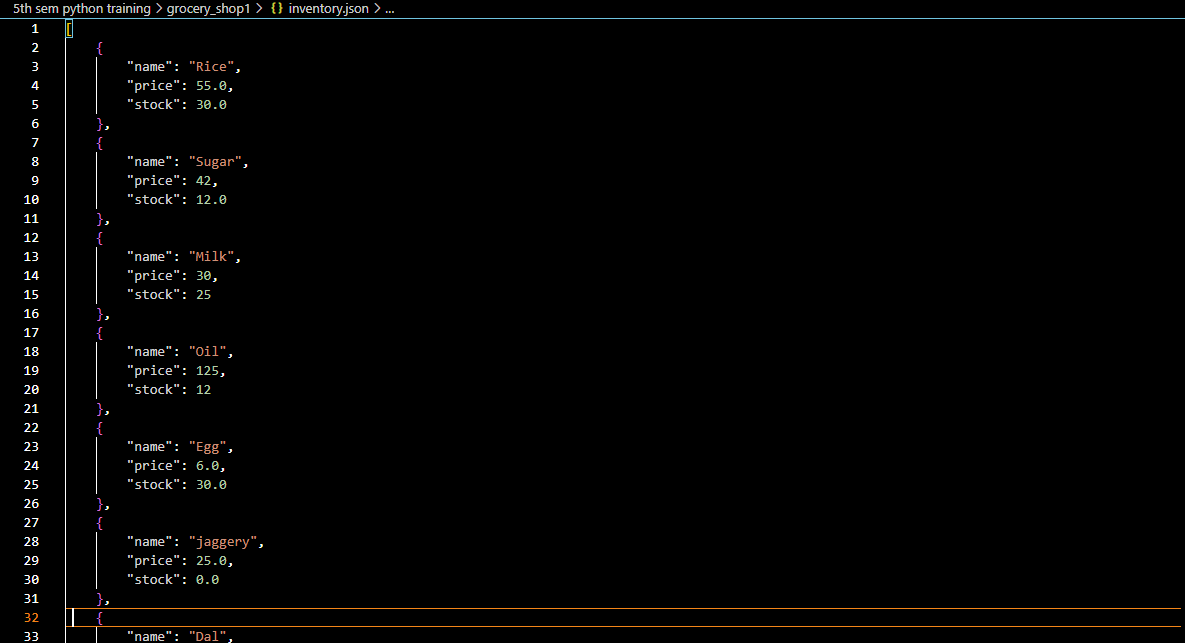
**Inventory page -**



**Billing page –**



***Inventory JSON File* –**



**Future Enhancements**

- Integration with SQL/NoSQL databases for scalability.  
- Authentication system (Admin/User roles).  
- Generating PDF bills.  
- Cloud deployment for accessibility.  
- Barcode scanner support for quick billing.

**References**

1. **Python Official Documentation**
2. **FastAPI Documentation**
3. **Jinja2 Documentation**

**Code**

**#models.py file**

**import json**

**from dataclasses import dataclass, asdict, field**

**from pathlib import Path**

**INVENTORY\_FILE = Path("inventory.json")**

**@dataclass**

**class Product:**

**name: str**

**price: float**

**stock: float**

**class Inventory:**

**def \_\_init\_\_(self):**

**self.\_products: dict[str, Product] = {}**

**self.load()  # Load inventory from JSON if exists**

**def \_key(self, name: str) -> str:**

**return name.strip().lower()**

**def add\_or\_update(self, name: str, price: float, stock\_delta: float) -> str:**

**k = self.\_key(name)**

**if k in self.\_products:**

**p = self.\_products[k]**

**p.stock += stock\_delta**

**p.price = price**

**self.save()**

**return f"Updated {p.name}: stock={p.stock}, price={p.price}"**

**else:**

**self.\_products[k] = Product(name=name.strip(), price=price, stock=stock\_delta)**

**self.save()**

**return f"Added new product {name.strip()} with stock={stock\_delta}, price={price}"**

**def find(self, name: str) -> Product | None:**

**return self.\_products.get(self.\_key(name))**

**def list\_all(self) -> list[Product]:**

**return list(self.\_products.values())**

**def reduce\_stock(self, name: str, qty: float) -> tuple[bool, str]:**

**p = self.find(name)**

**if p is None:**

**return False, f"{name} not found in inventory."**

**if qty <= 0:**

**return False, "Quantity must be > 0."**

**if p.stock < qty:**

**return False, f"Not enough stock for {p.name}. Available: {p.stock}"**

**p.stock -= qty**

**self.save()**

**return True, f"Reserved {qty} of {p.name}."**

**def restore\_stock(self, name: str, qty: float) -> None:**

**p = self.find(name)**

**if p and qty > 0:**

**p.stock += qty**

**self.save()**

**# ---------- JSON Persistence ----------**

**def save(self):**

**data = [asdict(p) for p in self.\_products.values()]**

**with open(INVENTORY\_FILE, "w") as f:**

**json.dump(data, f, indent=4)**

**def load(self):**

**if INVENTORY\_FILE.exists():**

**with open(INVENTORY\_FILE, "r") as f:**

**data = json.load(f)**

**for item in data:**

**self.\_products[self.\_key(item["name"])] = Product(\*\*item)**

**class Bill:**

**def \_\_init\_\_(self, inventory: Inventory):**

**self.inventory = inventory**

**self.cart: dict[str, dict] = {}**

**self.discount\_percent: float = 0.0**

**def \_key(self, name: str) -> str:**

**return name.strip().lower()**

**def add\_to\_cart(self, name: str, qty: float) -> tuple[bool, str]:**

**ok, msg = self.inventory.reduce\_stock(name, qty)**

**if not ok:**

**return False, msg**

**k = self.\_key(name)**

**prod = self.inventory.find(name)**

**price = prod.price if prod else 0.0**

**if k in self.cart:**

**self.cart[k]["qty"] += qty**

**self.cart[k]["price"] = price**

**else:**

**self.cart[k] = {"name": prod.name if prod else name.strip(), "price": price, "qty": qty}**

**return True, f"Added {qty} x {self.cart[k]['name']} to cart."**

**def remove\_from\_cart(self, name: str) -> tuple[bool, str]:**

**k = self.\_key(name)**

**if k not in self.cart:**

**return False, f"{name} not found in cart."**

**item = self.cart.pop(k)**

**self.inventory.restore\_stock(item["name"], item["qty"])**

**return True, f"Removed {item['name']} from cart."**

**def apply\_discount(self, percent: float) -> tuple[bool, str]:**

**if percent < 0 or percent > 100:**

**return False, "Discount must be between 0 and 100."**

**self.discount\_percent = percent**

**return True, f"Discount of {percent}% applied."**

**def clear\_discount(self):**

**self.discount\_percent = 0.0**

**def subtotal(self) -> float:**

**return sum(v["price"] \* v["qty"] for v in self.cart.values())**

**def totals(self) -> dict:**

**sub = self.subtotal()**

**discount\_amt = (self.discount\_percent / 100.0) \* sub if self.discount\_percent > 0 else 0.0**

**final\_total = sub - discount\_amt**

**return {**

**"subtotal": round(sub, 2),**

**"discount\_percent": self.discount\_percent,**

**"discount\_amount": round(discount\_amt, 2),**

**"final\_total": round(final\_total, 2),**

**}**

**def checkout(self) -> dict:**

**summary = {"items": list(self.cart.values()), "totals": self.totals()}**

**self.cart.clear()**

**self.clear\_discount()**

**return summary**

**#main.py file**

**from fastapi import FastAPI, Request, Form**

**from fastapi.responses import HTMLResponse, RedirectResponse**

**from fastapi.staticfiles import StaticFiles**

**from fastapi.templating import Jinja2Templates**

**from models import Inventory, Bill**

**app = FastAPI()**

**# Static files and templates**

**app.mount("/static", StaticFiles(directory="static"), name="static")**

**templates = Jinja2Templates(directory="templates")**

**# Global state (in-memory)**

**inventory = Inventory()**

**bill = Bill(inventory)**

**# Preload items if inventory is empty**

**if not inventory.list\_all():**

**inventory.add\_or\_update("Rice", 55, 30)**

**inventory.add\_or\_update("Sugar", 42, 20)**

**inventory.add\_or\_update("Milk", 30, 25)**

**inventory.add\_or\_update("Oil", 125, 12)**

**def redirect(url: str, msg: str | None = None):**

**if msg:**

**return RedirectResponse(url=f"{url}?msg={msg}", status\_code=303)**

**return RedirectResponse(url=url, status\_code=303)**

**@app.get("/", response\_class=HTMLResponse)**

**def home(request: Request, msg: str | None = None):**

**return templates.TemplateResponse("index.html", {"request": request, "msg": msg})**

**# ---------------- Inventory ----------------**

**@app.get("/inventory", response\_class=HTMLResponse)**

**def get\_inventory(request: Request, msg: str | None = None):**

**products = inventory.list\_all()**

**return templates.TemplateResponse(**

**"inventory.html", {"request": request, "products": products, "msg": msg}**

**)**

**@app.post("/inventory/add-update")**

**def add\_update\_product(name: str = Form(...), price: float = Form(...), stock: float = Form(...)):**

**message = inventory.add\_or\_update(name, price, stock)**

**return redirect("/inventory", message)**

**# ---------------- Billing / Cart ----------------**

**@app.get("/billing", response\_class=HTMLResponse)**

**def billing\_page(request: Request, msg: str | None = None):**

**products = inventory.list\_all()**

**cart\_items = list(bill.cart.values())**

**totals = bill.totals()**

**return templates.TemplateResponse(**

**"billing.html",**

**{**

**"request": request,**

**"products": products,**

**"cart\_items": cart\_items,**

**"totals": totals,**

**"msg": msg,**

**},**

**)**

**@app.post("/cart/add")**

**def add\_to\_cart(product\_name: str = Form(...), quantity: float = Form(...)):**

**ok, message = bill.add\_to\_cart(product\_name, quantity)**

**return redirect("/billing", message)**

**@app.post("/cart/remove")**

**def remove\_from\_cart(product\_name: str = Form(...)):**

**ok, message = bill.remove\_from\_cart(product\_name)**

**return redirect("/billing", message)**

**@app.post("/discount/apply")**

**def apply\_discount(percent: float = Form(...)):**

**ok, message = bill.apply\_discount(percent)**

**return redirect("/billing", message)**

**@app.post("/checkout")**

**def checkout():**

**\_ = bill.checkout()**

**return redirect("/", "Checkout complete. Cart cleared.")**

**# ---------------- CLI Inventory Display ----------------**

**def show\_inventory\_cli():**

**print("\n--- Current Inventory ---")**

**products = inventory.list\_all()**

**if not products:**

**print("Inventory is empty.")**

**else:**

**for product in products:**

**print(f"{product.name} - Price: ₹{product.price} - Stock: {product.stock}")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**while True:**

**print("\nCommands:")**

**print("1. Show Inventory")**

**print("2. Exit")**

**choice = input("Enter choice: ").strip()**

**if choice == "1":**

**show\_inventory\_cli()**

**elif choice == "2":**

**break**

**else:**

**print("Invalid choice.")**

**Conclusion**

The Grocery Shop Billing System demonstrates an efficient solution for inventory and billing management. It highlights the use of FastAPI for backend processing, Jinja2 for frontend rendering, and JSON for lightweight storage. The system fulfills the objectives of inventory updates, billing, discount management, and checkout.

**Signature**