



UNIVERSITY INSTITUTE *of*
COMPUTING
Asia's Fastest Growing University



A
Case Study
On
“Grocery Management Store”



MASTERS IN COMPUTER APPLICATION

Submitted By:

Name- Viraj Vikram Singh
UID- 24MCA20369
Branch- MCA
Section- 6-A

Submitted To:

Ms. Palwinder Kaur Mangat
Assistant Professor

Index

| S. No. | Title | Page No. |
|---------------|-------------------------------|-----------------|
| 1. | Objectives | 3 |
| 2. | Introduction | 3 |
| 3. | Problem Definition | 4 |
| 4. | Solution Approach | 4-5 |
| 5. | Software Requirements | 5 |
| 6. | Steps for creation of project | 5-9 |
| 7. | Results | 10-12 |
| 8. | Conclusion | 13 |
| 9. | References | 13 |

Objectives:

The primary objective of the **Grocery Management Store Python web application** is to provide an efficient, automated solution for managing a grocery store's operations, including inventory tracking, customer transactions, and order management. The application aims to streamline the overall workflow by offering store administrators tools to easily add, update, and manage product listings, including details such as product name, price, quantity, and categories. It reduces manual intervention by automatically updating inventory levels when items are purchased or restocked, ensuring accurate stock management.

The application also focuses on scalability and flexibility, providing features for managing multiple users, including customers and staff, while ensuring data security. In essence, the objective is to modernize traditional grocery store management, reducing human errors, improving operational efficiency, and delivering a better shopping experience for customers.

Introduction:

The **Grocery Management Store** is a Python-based web application designed to optimize the management and daily operations of a grocery store. It integrates various features like inventory management, sales tracking, customer interaction, and billing into a single, efficient system. This application is intended to minimize manual efforts, reduce errors, and enhance the user experience by offering a digital platform to manage the various aspects of a grocery store. Whether you are a store owner, employee, or customer, the system ensures smooth, reliable, and real-time interactions, making the overall process more organized and efficient.



Fig-1.1 (Manage Grocery Products on daily basis)

Problem Definition:

Managing a grocery store involves several complex, repetitive, and error-prone tasks. Some of the key challenges are:

1) Manual Inventory Tracking:

Traditional paper-based or spreadsheet methods to keep track of products can lead to inaccuracies, overstocking, or stockouts.

2) Billing and Invoicing Errors:

Manually calculating bills and invoices can result in discrepancies and delays in checkout.

3) Order Management:

Handling customer orders, either in-store or online, requires a system to track and fulfill these orders efficiently.

4) Time-consuming Reports:

Generating reports such as daily sales summaries, stock levels, and profit/loss calculations can be tedious when done manually.

Solution Approach:

To address these problems, the Grocery Management Store Web Application provides the following solutions:

1) Inventory Management System:

- Real-time tracking of stock levels.
- Simplified product addition and categorization (with details like product name, price, order date, etc.)

2) Order Management:

- Easy-to-use system for processing in-store orders.
- Track and fulfill customer orders in real-time.
- Provide customers with order histories.

3) Customer Management:

- Track customer purchase history, preferences, and feedback.
- Use customer data to offer personalized promotions and discounts.

4) Data Analytics & Reporting:

- Daily, weekly, and monthly sales reports for quick analysis of business performance.
- Stock level reports and product popularity insights.

5) User-Friendly Interface:

- Store owners can manage product listings, stock levels, and reports from a dashboard.
- Owner can also generate orders for customer as per their requirement.

Software Requirement:

1) Frontend (UI):

- **HTML, CSS, JavaScript:** For creating the web interface, ensuring responsiveness, and providing interactivity.
- **Bootstrap:** For styling and creating a mobile-friendly design.

2) Backend:

- **Python:** Core language for developing the application's logic.
- **Flask:** A lightweight Python web framework used to create the web application.

3) Database:

- **MySQL Workbench:** Database for storing product, customer, and transaction data.

In any web backend, there has to be a web server, and In this web application, we are using a Flask as a micro-framework for our web server.

There is another option i.e. Django but we will use Flask because it is light-weight.

Steps for creation of Grocery Store Web App:

To create the web application, where Owners can add products, view a list of products, and maintain order list, follow these general steps:

1) Set Up Your Development Environment:

- **Install Python and Flask:** Ensure we have Python installed, then install Flask (a lightweight web framework) for building the backend.
- **Install PyCharm:** Ensure we have an Integrated Development Environment (IDE) for easy flow and execution of our code.
- **Set Up Directories:** For smooth flow, we have to create a proper directory structure for the frontend files.

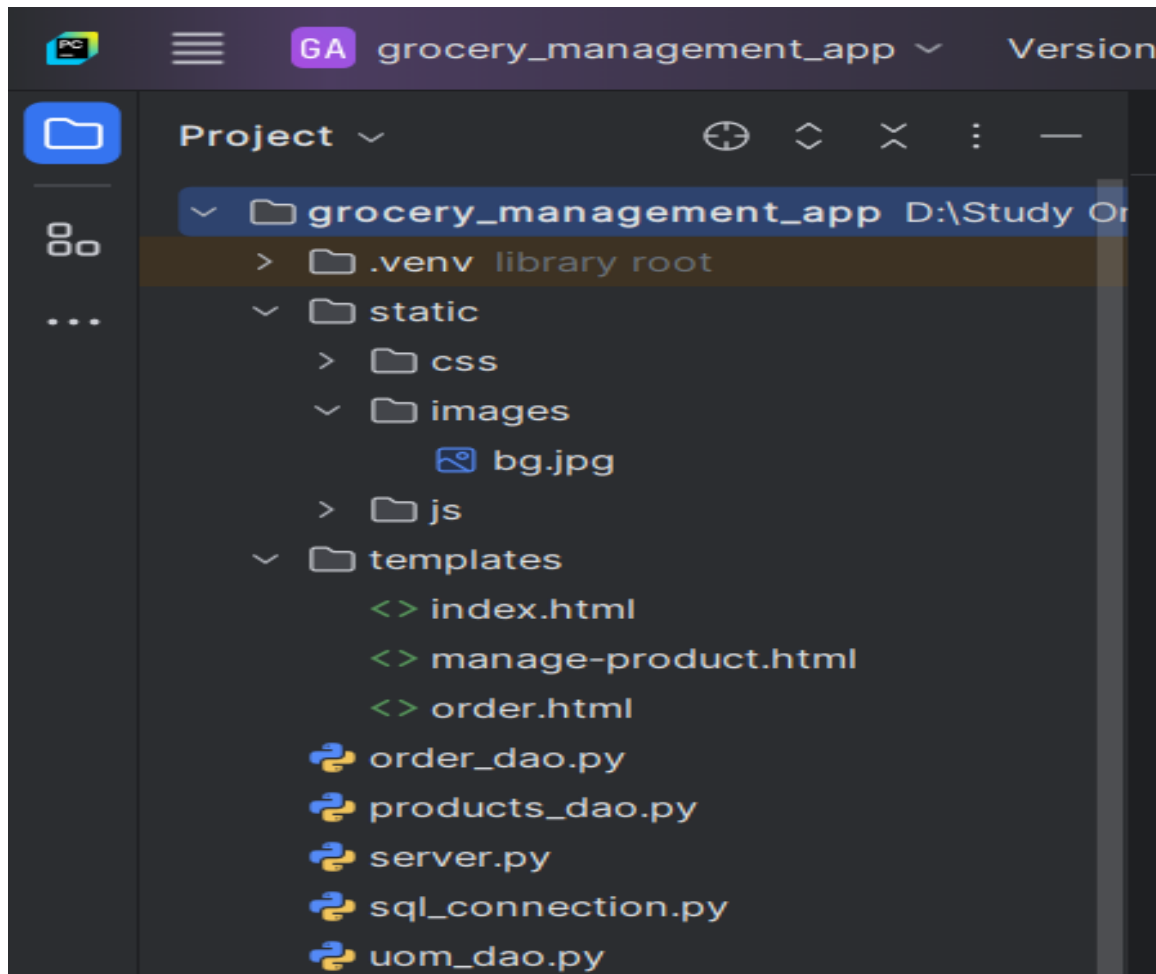


Fig-1.2 (Creation of Directories)

2) Create Flask Application (Backend):

- Inside server.py file (the main Flask app), This is the basic Boiler Plate to run the Flask is:

```

2  from flask import Flask, request, jsonify
3  from flask import render_template
4  import order_dao
5  from sql_connection import get_sql_connection
6  import products_dao
7  import uom_dao
8
9  app = Flask(__name__)
10
11  connection = get_sql_connection()
12
13  @app.route('/')
14  def index():
15      return render_template('index.html')
16
17  if __name__ == "__main__":
18      print("Starting Python Flask Server For Grocery Management App")
19      app.run(debug=True, port=5000)

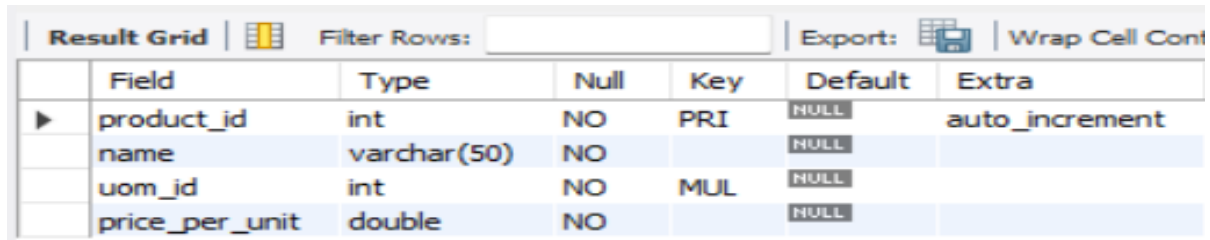
```

Fig-1.3 (Creation of Flask App)

3) Create Database Schemas:

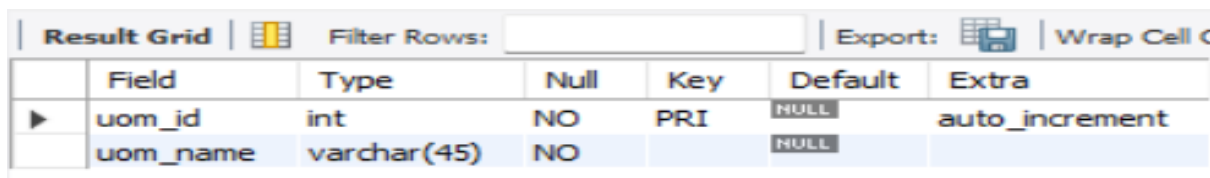
Firstly, we will have to create one schema as like 'grocery_store'. After that we have to create tables as like -

- **products table:** [product_id, name, uom_id, price_per_unit]
- **uom table:** [uom_id, uom_name]



| | Field | Type | Null | Key | Default | Extra |
|---|----------------|-------------|------|-----|---------|----------------|
| ▶ | product_id | int | NO | PRI | NULL | auto_increment |
| | name | varchar(50) | NO | | NULL | |
| | uom_id | int | NO | MUL | NULL | |
| | price_per_unit | double | NO | | NULL | |

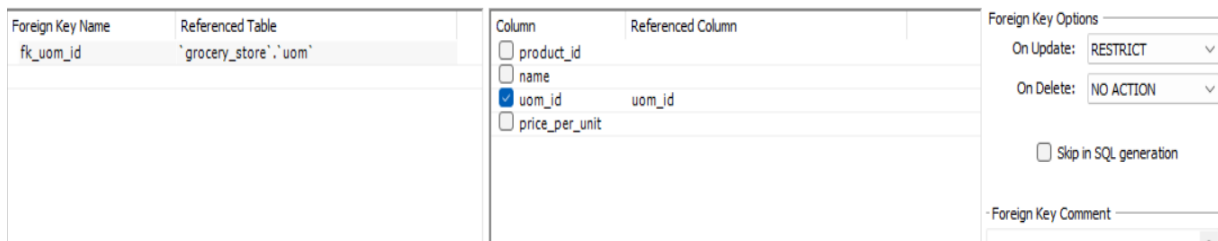
Fig-1.4 (product table)



| | Field | Type | Null | Key | Default | Extra |
|---|----------|-------------|------|-----|---------|----------------|
| ▶ | uom_id | int | NO | PRI | NULL | auto_increment |
| | uom_name | varchar(45) | NO | | NULL | |

Fig-1.5 (uom table)

Now set up relation between products table and uom table using foreign key concept to maintain the referential integrity, then it will link the two tables, and this process is done for validation.



| Foreign Key Name | Referenced Table |
|------------------|-----------------------|
| fk_uom_id | 'grocery_store`.`uom` |

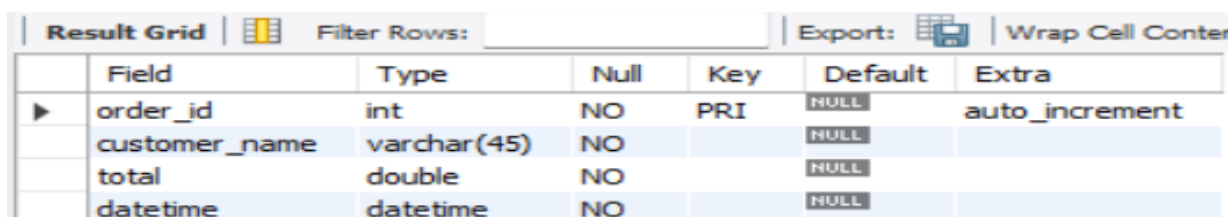
| Column | Referenced Column |
|--|-------------------|
| <input type="checkbox"/> product_id | |
| <input type="checkbox"/> name | |
| <input checked="" type="checkbox"/> uom_id | uom_id |
| <input type="checkbox"/> price_per_unit | |

Foreign Key Options
On Update: RESTRICT
On Delete: NO ACTION
☐ Skip in SQL generation
Foreign Key Comment

Fig-1.6 (linking of both tables as per uom_id)

After that we have to create two more tables as like –

- **orders table:** [order_id, customer_name, total, datetime]
- **order_details:** [order_id, product_id, quantity, total_price]



| | Field | Type | Null | Key | Default | Extra |
|---|---------------|-------------|------|-----|---------|----------------|
| ▶ | order_id | int | NO | PRI | NULL | auto_increment |
| | customer_name | varchar(45) | NO | | NULL | |
| | total | double | NO | | NULL | |
| | datetime | datetime | NO | | NULL | |

Fig-1.7 (orders table)

| Result Grid | | Filter Rows: | | Export: | | |
|-------------|-------------|--------------|------|---------|---------|-------|
| | Field | Type | Null | Key | Default | Extra |
| ▶ | order_id | int | NO | PRI | NULL | |
| | product_id | int | NO | MUL | NULL | |
| | quantity | double | NO | | NULL | |
| | total_price | double | NO | | NULL | |

Fig-1.8 (order_details table)

Now again set up relation between orders table and order_details table using foreign key concept to maintain the referential integrity, then it will link the two tables, and this process is done for validation.

| Foreign Key Name | Referenced Table | Column | Referenced Column | Foreign Key Options |
|------------------|-----------------------------|--|-------------------|---|
| fk_order_id | 'grocery_store', 'orders' | <input checked="" type="checkbox"/> order_id | order_id | On Update: RESTRICT On Delete: NO ACTION |
| fk_product_id | 'grocery_store', 'products' | <input type="checkbox"/> product_id <input type="checkbox"/> quantity <input type="checkbox"/> total_price | | <input type="checkbox"/> Skip in SQL generation |

Fig-1.9 (linking of order_details with orders table as per order_id)

| Foreign Key Name | Referenced Table | Column | Referenced Column | Foreign Key Options |
|------------------|-----------------------------|---|-------------------|--|
| fk_order_id | 'grocery_store', 'orders' | <input type="checkbox"/> order_id | | On Update: RESTRICT On Delete: RESTRICT |
| fk_product_id | 'grocery_store', 'products' | <input checked="" type="checkbox"/> product_id <input type="checkbox"/> quantity <input type="checkbox"/> total_price | product_id | |

Fig-2.0 (linking of order_details with orders table as per product_id)

4) Create sql-connection.py:

Now, In PyCharm we have to create one sql-connection.py file inside 'grocery_management_app'. This file contains the basic information about the database of user such as:

- user='root', password='12345', host='127.0.0.1' and database='grocery_store'
- We also have to add 'mysql-connector-python' library in our project to establish a database connection between database and our project.
- In this way, we will separately create sql_connection to make our code modular and return the connection. We can use the connection whenever we want, we don't need to write the code again and again.

```

2  from flask import Flask, request, jsonify
3  from flask import render_template
4  import order_dao
5  from sql_connection import get_sql_connection
6  import products_dao
7  import uom_dao
8
9  app = Flask(__name__)
10
11  connection = get_sql_connection()
12
13  @app.route('/')
14  def index():
15      return render_template('index.html')
16
17  if __name__ == "__main__":
18      print("Starting Python Flask Server For Grocery Management App")
19      app.run(debug=True, port=5000)

```


Fig-2.1 (creating sql-connection.py)

5) Create templates Directory:

Inside 'templates' directory, we have to create three html files for development of our frontend UI Interface which are as follows:

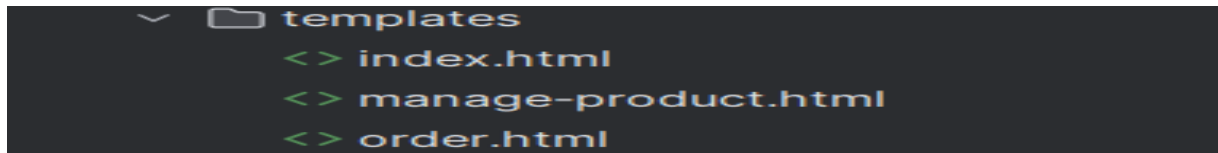


Fig-2.2 (creating html files inside templates directory)

6) Create static Directory:

Inside 'static' directory, we have to create three more directories -

- **css:** for storing css files.
- **images:** for storing images, which will render after our project deployment.
- **js:** for storing js files, in which we embedded the business logic.

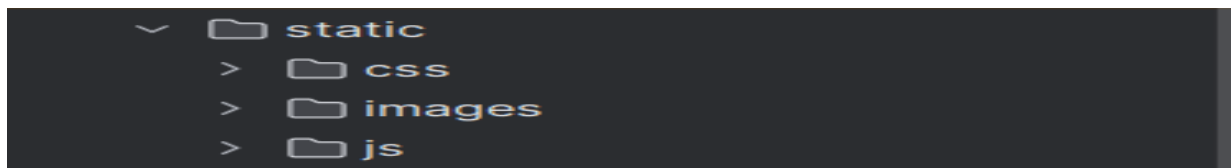


Fig-2.3 (creating static directory)

7) Create DAO files:

Now, we have to create various dao files [data access object] to perform different operations as per the requirement of the project.

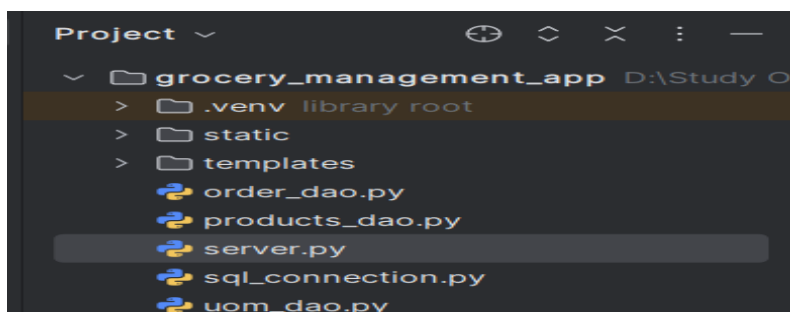
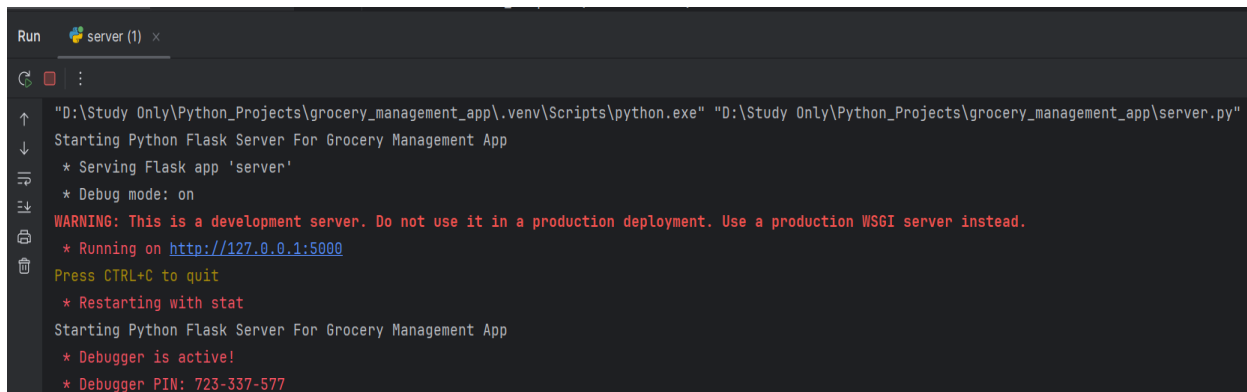


Fig-2.4 (creating dao files)

Results:

- Firstly, we have to start the Flask server to deploy our web application on the local server.



```
Run server (1) x
"D:\Study Only\Python_Projects\grocery_management_app\.venv\Scripts\python.exe" "D:\Study Only\Python_Projects\grocery_management_app\server.py"
Starting Python Flask Server For Grocery Management App
* Serving Flask app 'server'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
Starting Python Flask Server For Grocery Management App
* Debugger is active!
* Debugger PIN: 723-337-577
```

Fig-2.5 (Deployment of Project)

- Now we have to copy the URL and paste it on any web browser with ‘/’ because it is mapped in the index() function inside server.py to render or load the index page.

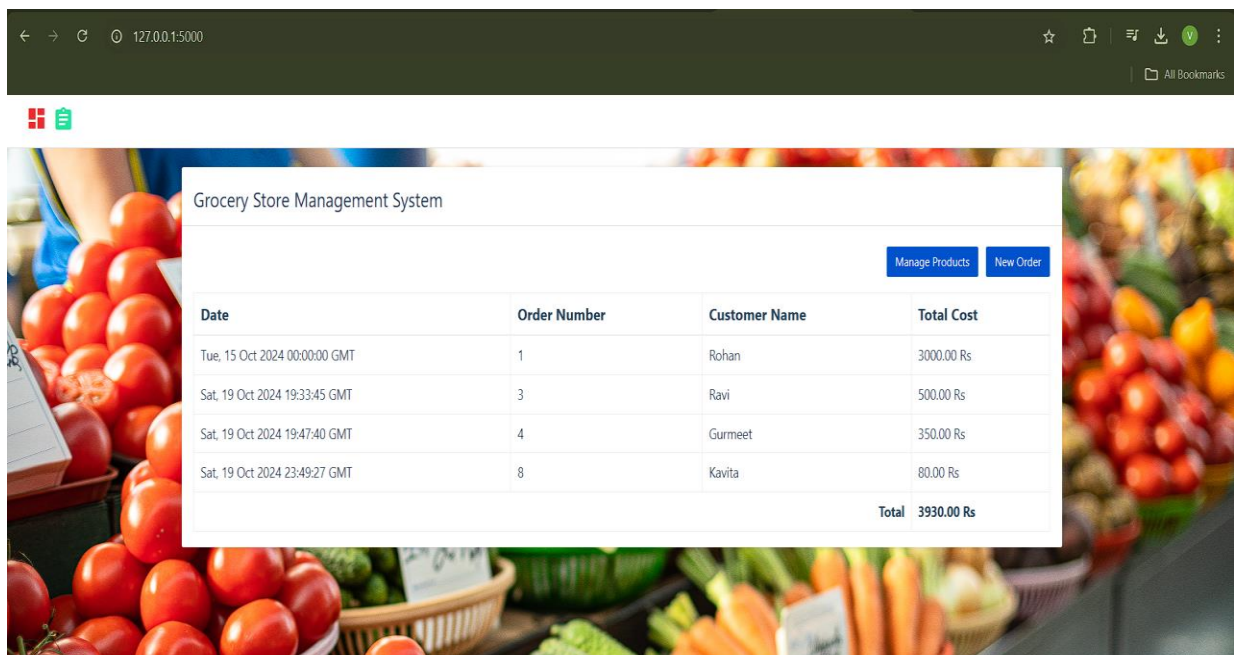


Fig-2.6 (Index Page)

- The Index page has a tabular format which contains the order of particular customers with Date, Order Number, Customer Name and Total Cost.

Grocery Store Management System

| | | | | Manage Products | New Order |
|-------------------------------|--------------|---------------|--------------|---------------------------------|---------------------------|
| Date | Order Number | Customer Name | Total Cost | | |
| Tue, 15 Oct 2024 00:00:00 GMT | 1 | Rohan | 3000.00 Rs | | |
| Sat, 19 Oct 2024 19:33:45 GMT | 3 | Ravi | 500.00 Rs | | |
| Sat, 19 Oct 2024 19:47:40 GMT | 4 | Gurmeet | 350.00 Rs | | |
| Sat, 19 Oct 2024 23:49:27 GMT | 8 | Kavita | 80.00 Rs | | |
| | | | Total | 3930.00 Rs | |

Fig-2.7 (Orders of different Customers)

- When we click on 'Manage Products' button then it will open a dialog box to add products in our grocery store.

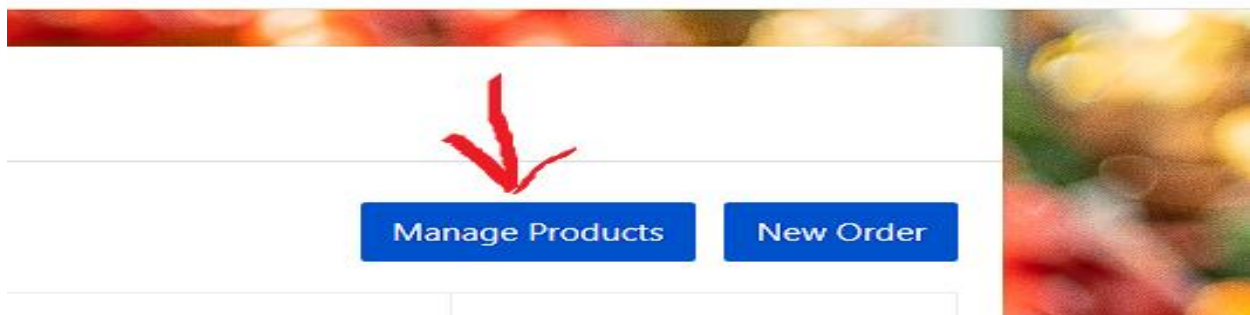


Fig-2.8 (Manage Products Button)

Add New Product

Name
Maggie

Unit
each

Price Per Unit
10

Close Save

Fig-2.9 (Add New Product dialog box)

- When we click on ‘New Order’ button then it will display a new html page in which we can create a new order for the customer.

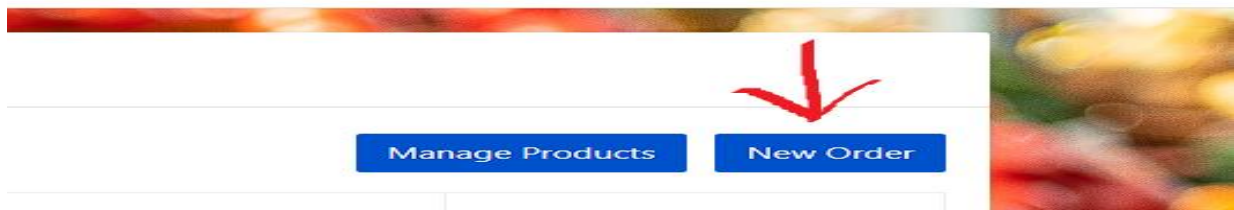


Fig-3.0 (New Order Button)

 A screenshot of the 'New Order' form. The form has a header with 'New Order' and a user name 'Shobhit'. Below the header is a table with columns: 'Product', 'Price', 'Quantity', and 'Total'. There is an 'Add More' button. The table contains three rows of items: 'toothpaste' (Price: 30, Quantity: 2, Total: 60.00), 'Maggie' (Price: 10, Quantity: 5, Total: 50.00), and 'Sugar' (Price: 45, Quantity: 2, Total: 90.00). Each row has a 'Remove' button. At the bottom, there is a 'Total' row showing '200.00' and a 'Save' button.

Fig-3.1 (Create a new order for Customer)

- When we click on the ‘Save’ button then it will automatically create the order, store the order_details in the database and reflect this new order in the index page as a newly added order.

 A screenshot of the 'Grocery Store Management System' index page. It shows a table with columns: 'Date', 'Order Number', 'Customer Name', and 'Total Cost'. The table lists five orders. A red arrow points to the last order. At the bottom right, there is a 'Total' row showing '4130.00 Rs'.

| Date | Order Number | Customer Name | Total Cost |
|-------------------------------|--------------|---------------|------------|
| Tue, 15 Oct 2024 00:00:00 GMT | 1 | Rohan | 3000.00 Rs |
| Sat, 19 Oct 2024 19:33:45 GMT | 3 | Ravi | 500.00 Rs |
| Sat, 19 Oct 2024 19:47:40 GMT | 4 | Gurmeet | 350.00 Rs |
| Sat, 19 Oct 2024 23:49:27 GMT | 8 | Kavita | 80.00 Rs |
| Mon, 21 Oct 2024 01:40:07 GMT | 48 | Shobhit | 200.00 Rs |
| Total | | | 4130.00 Rs |

Fig-3.2 (Create a new order for Customer)

Conclusion:

The Grocery Management Store Python web application serves as an efficient and user-friendly platform for managing a grocery store's day-to-day operations. Built using modern web development technologies like Python, Flask, and integrated databases, the application streamlines inventory management, customer transactions, and order tracking. It allows store administrators to easily add, update, and delete product details such as name, price, quantity, and category, ensuring real-time inventory control.

The application also implements essential features like order history tracking, stock alerts for low inventory, and create new orders for customers to help store managers make data-driven decisions. Moreover, the responsive design ensures the platform is accessible across different devices, enhancing user experience. By automating many of the manual tasks involved in grocery store management, this web application improves operational efficiency, reduces errors, and enhances customer satisfaction, making it a comprehensive solution for modern retail businesses.

References:

- Pallets Projects. (2023). *Flask Documentation*. Retrieved from <https://flask.palletsprojects.com>
- SQLAlchemy Project. (2023). *SQLAlchemy Documentation*. Retrieved from <https://docs.sqlalchemy.org>
- Bootstrap Team. (2023). *Bootstrap Documentation*. Retrieved from <https://getbootstrap.com/docs>
- SQLite Consortium. (2023). *SQLite Documentation*. Retrieved from <https://www.sqlite.org/docs.html>
- Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
- ChatGPT, "Grocery Management System Web Application: Conceptual Design and Features." *ChatGPT by OpenAI*. Consulted for generating ideas and understanding the key components required in developing a Grocery Management System web application.