Grocery Management Store

(Minor Project)

On

Python Programming

Course Code- 24CAH-606

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

Aim:

Grocery Management Store (Minor Project)

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.

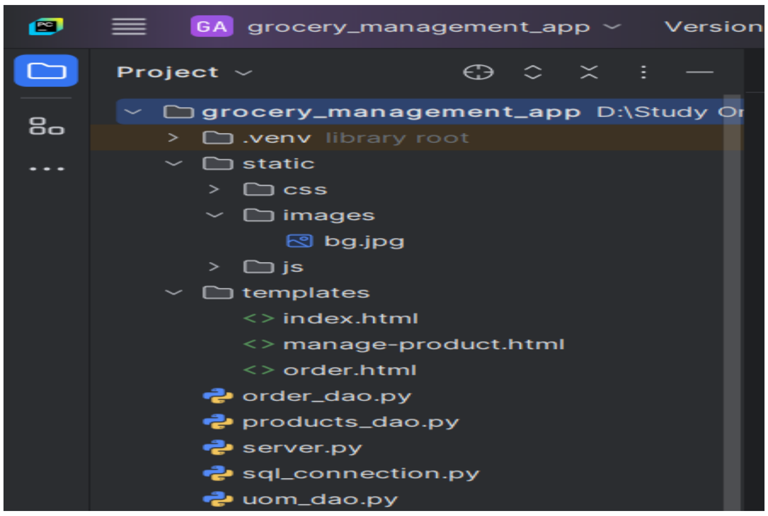
The application also focuses on scalability and flexibility, providing features for managing multiple users, including customers and staff, while ensuring data security. In essence, the objective is to modernize traditional grocery store management, reducing human errors, improving operational efficiency,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

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.

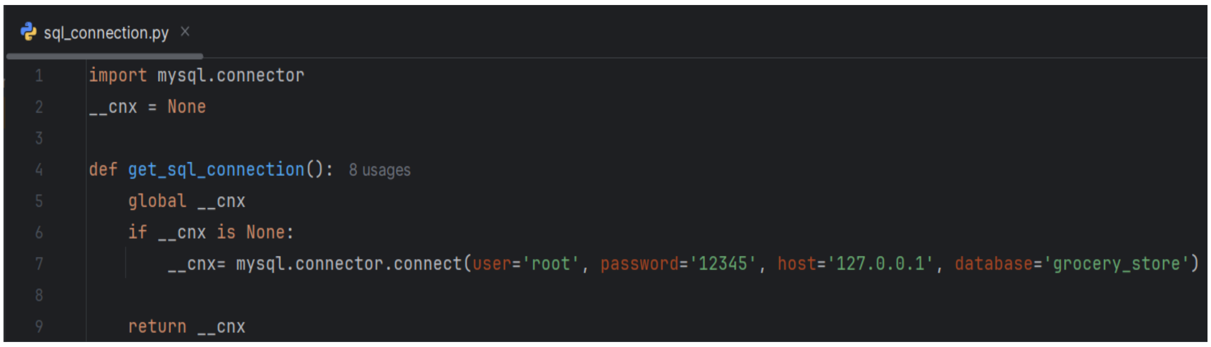


```
grocery_management_app D:\Study O
  .venv library root
  static
    css
    images
      bg.jpg
    js
  templates
    index.html
    manage-product.html
    order.html
  order_dao.py
  products_dao.py
  server.py
  sql_connection.py
  uom_dao.py
```

Fig-1.1 (Creation of Directories)

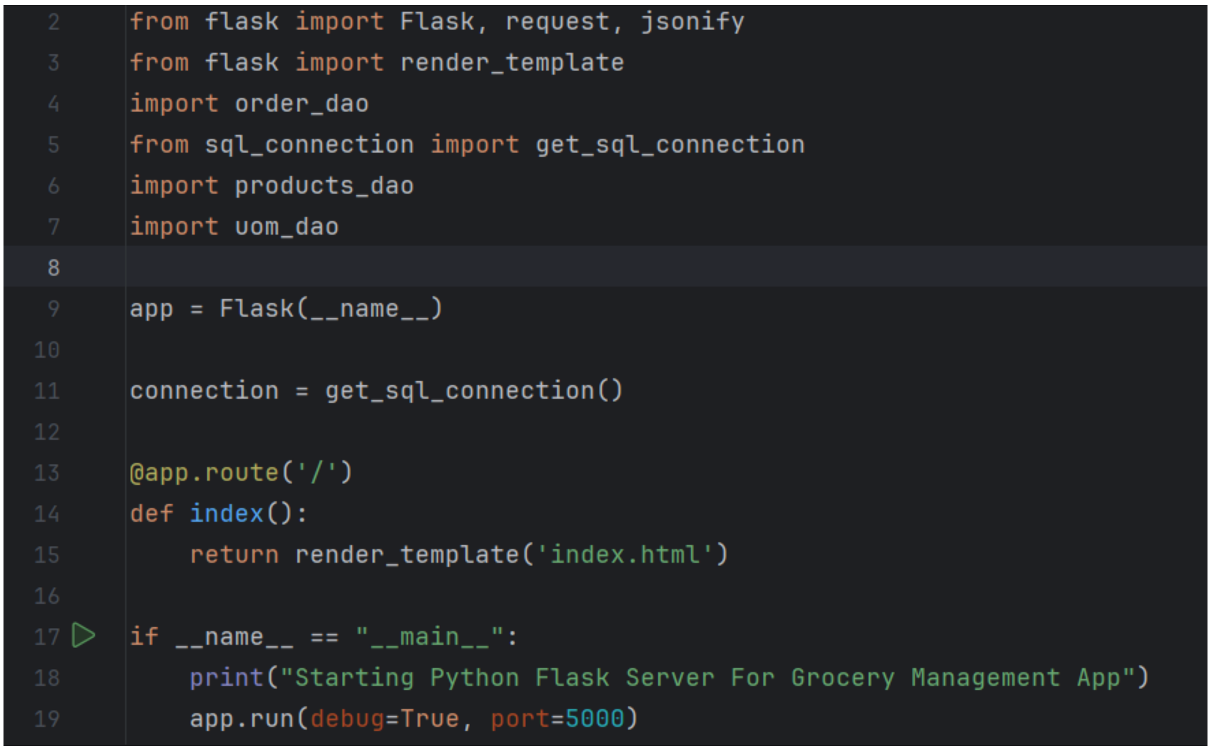
2) **Create Flask Application (Backend):**

- Create SQL Connection:



```python
import mysql.connector
__cnx = None

def get_sql_connection():
    global __cnx
    if __cnx is None:
        __cnx= mysql.connector.connect(user='root', password='12345', host='127.0.0.1', database='grocery_store')

    return __cnx
```

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



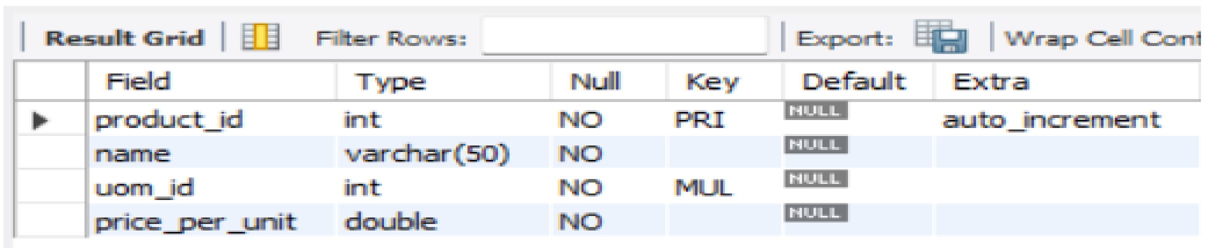
```python
from flask import Flask, request, jsonify
from flask import render_template
import order_dao
from sql_connection import get_sql_connection
import products_dao
import uom_dao

app = Flask(__name__)

connection = get_sql_connection()

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == "__main__":
    print("Starting Python Flask Server For Grocery Management App")
    app.run(debug=True, port=5000)
```

Fig-1.2 (Creation of Flask App)

3) **Create Database Schemas:**

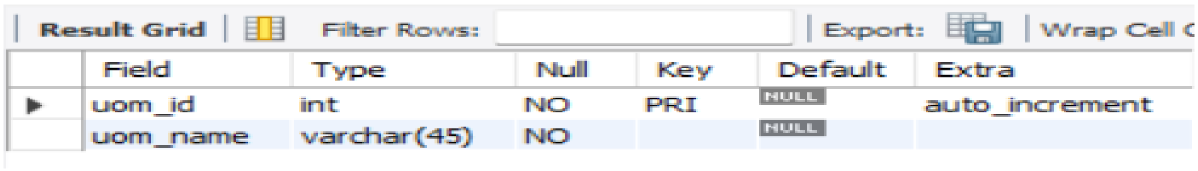
Firstly, we will have to create one schema as like ‘grocery_store’. After that we have to create tables as like -

- **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	

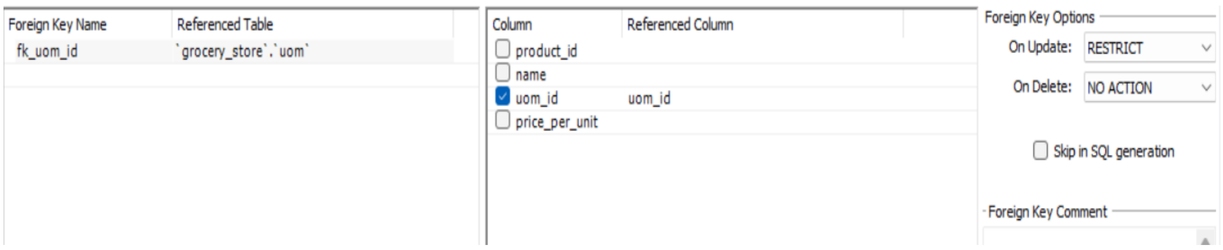
Fig-1.3 (product table)



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

Fig-1.4 (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\`

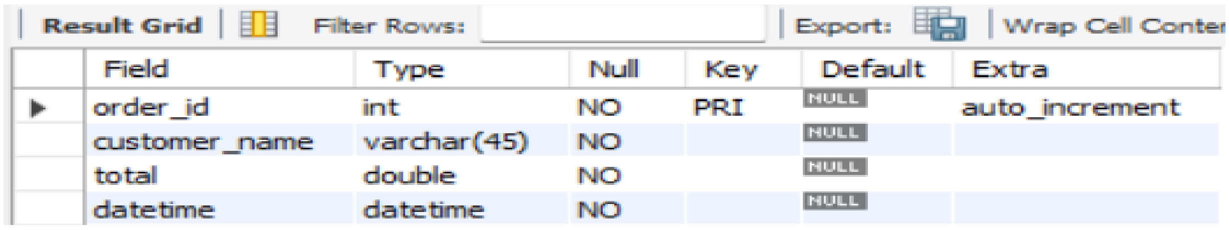
Column	Referenced Column
☐ product_id	
☐ name	
☑ uom_id	uom_id
☐ price_per_unit	

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

Fig-1.5 (linking of both tables as per uom_id)

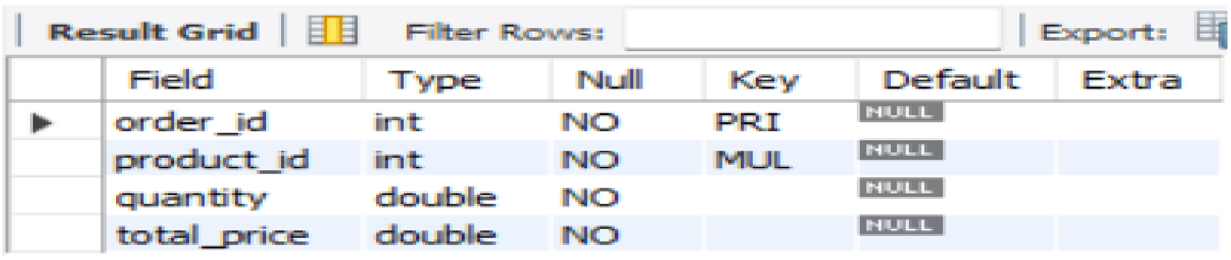
After that we have to create two more tables as like –

- 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	

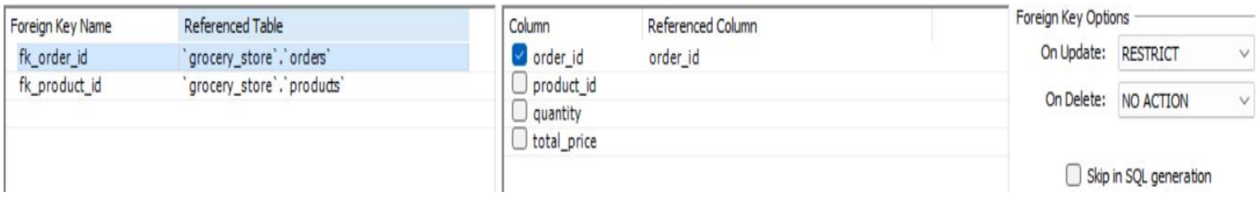
Fig-1.6 (orders table)



	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-1.7 (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.

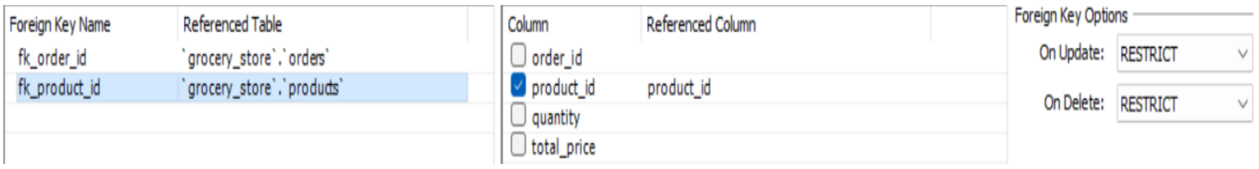


Foreign Key Name	Referenced Table
fk_order_id	`grocery_store`.`orders`
fk_product_id	`grocery_store`.`products`

Column	Referenced Column
✓ order_id	order_id
☐ product_id	
☐ quantity	
☐ total_price	

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

Fig-1.8 (linking of order_details with orders table as per order_id)



Foreign Key Name	Referenced Table
fk_order_id	`grocery_store`.`orders`
fk_product_id	`grocery_store`.`products`

Column	Referenced Column
☐ order_id	
✓ product_id	product_id
☐ quantity	
☐ total_price	

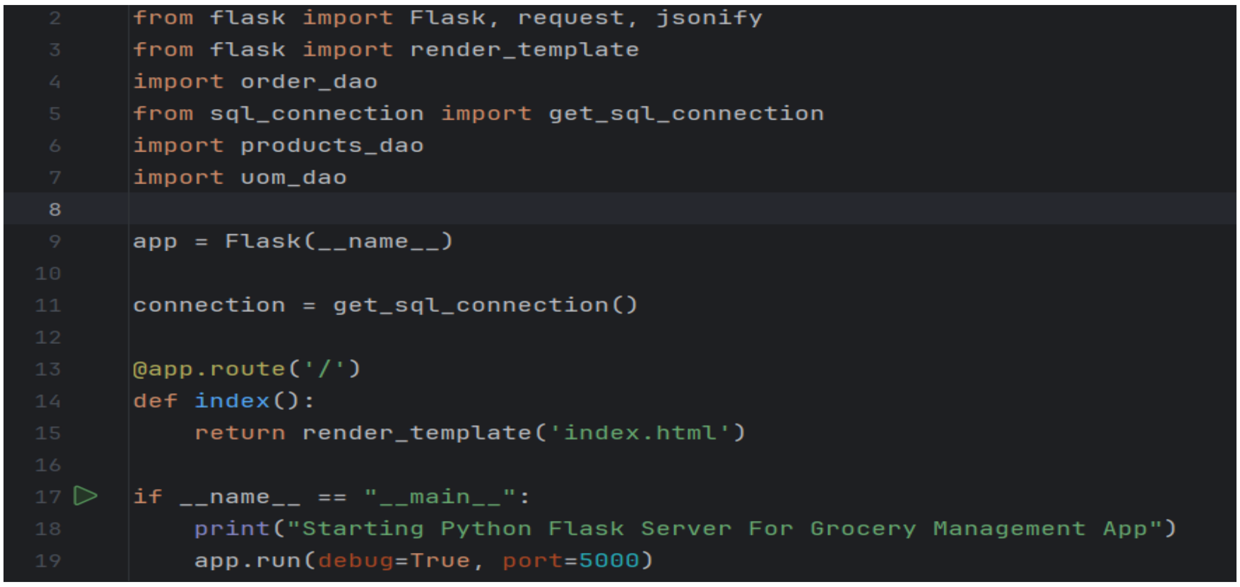
Foreign Key Options
On Update: RESTRICT
On Delete: RESTRICT

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

4) **Create sql-connection.py:**

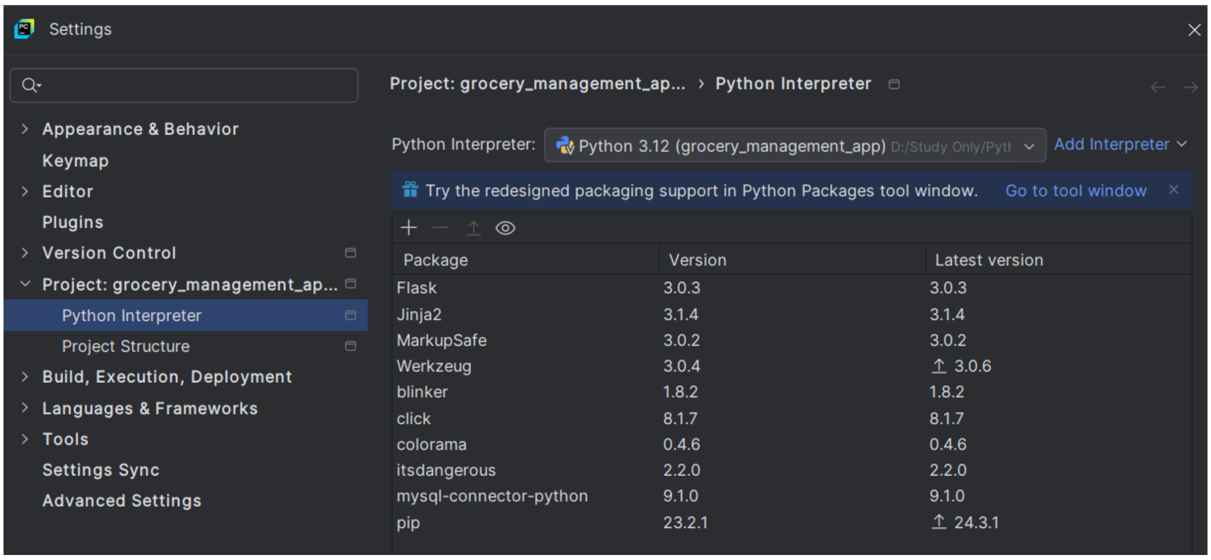
Now, In PyCharm we have to create one sql-connection.py file inside 'grocery_management_app'. This file contains the basic information about the database of user such as:

- user='root', password='12345', host='127.0.0.1' and database='grocery_store'
- We also have to add 'mysql-connector-python' and 'flask' 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.



```python
from flask import Flask, request, jsonify
from flask import render_template
import order_dao
from sql_connection import get_sql_connection
import products_dao
import uom_dao

app = Flask(__name__)

connection = get_sql_connection()

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == "__main__":
    print("Starting Python Flask Server For Grocery Management App")
    app.run(debug=True, port=5000)
```

Fig-2.0 (creating sql-connection.py)



Python Interpreter: Python 3.12 (grocery_management_app) D:/Study Only/Pyth

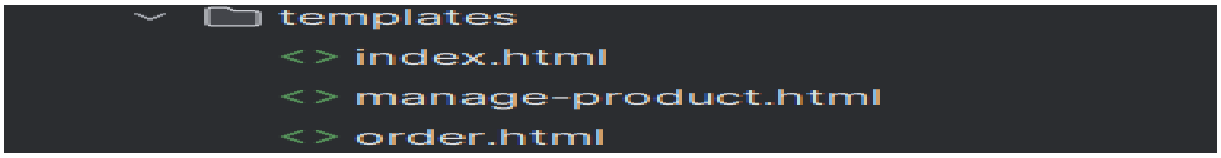
Try the redesigned packaging support in Python Packages tool window.

Package	Version	Latest version
Flask	3.0.3	3.0.3
Jinja2	3.1.4	3.1.4
MarkupSafe	3.0.2	3.0.2
Werkzeug	3.0.4	↥ 3.0.6
blinker	1.8.2	1.8.2
click	8.1.7	8.1.7
colorama	0.4.6	0.4.6
itsdangerous	2.2.0	2.2.0
mysql-connector-python	9.1.0	9.1.0
pip	23.2.1	↥ 24.3.1

Fig-2.1 (adding necessary libraries)

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:



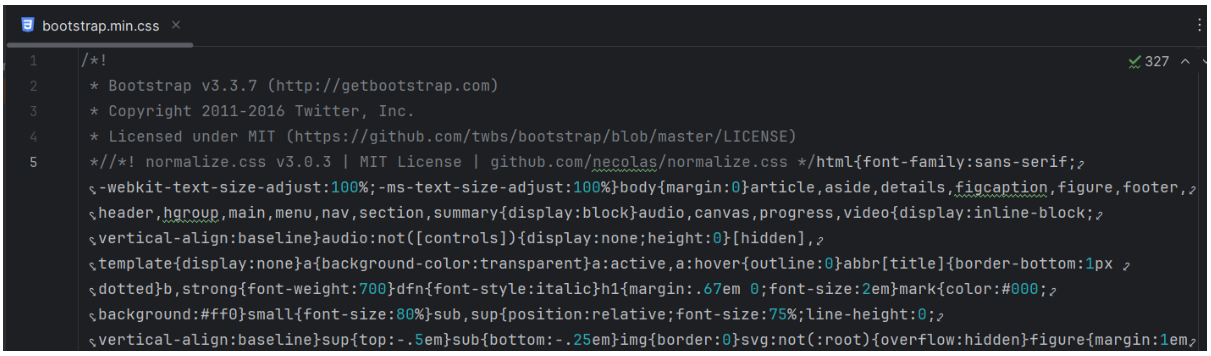
```
templates
    index.html
    manage-product.html
    order.html
```

Fig-2.2 (creating html files inside templates directory)

6) **Create static Directory:**

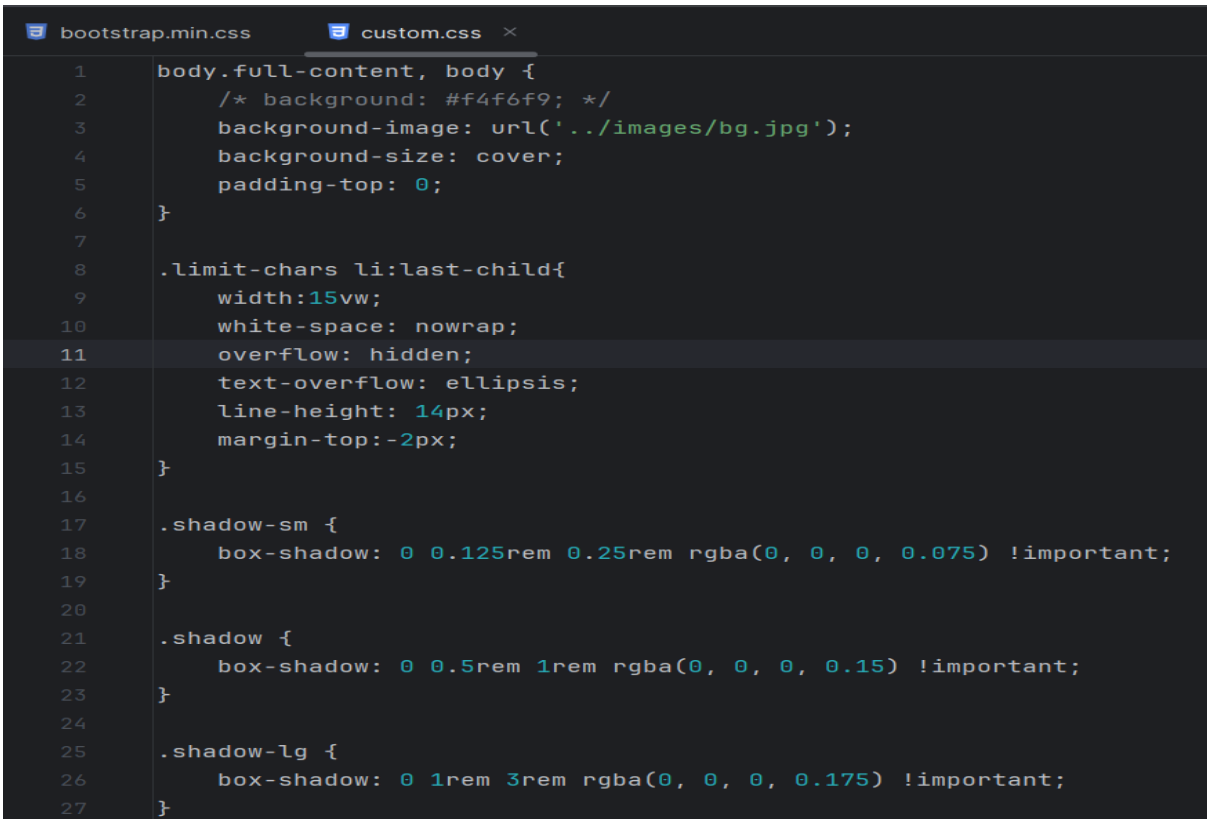
Inside ‘static’ directory, we have to create three more directories -

- **css**: for storing css files, there are various css files inside css directory:



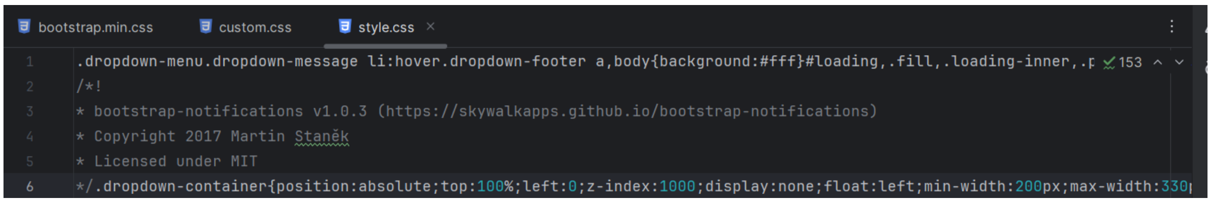
```
/*!
 * Bootstrap v3.3.7 (http://getbootstrap.com)
 * Copyright 2011-2016 Twitter, Inc.
 * Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/LICENSE)
 *//*! normalize.css v3.0.3 | MIT License | github.com/necolas/normalize.css */html{font-family:sans-serif;
-webkit-text-size-adjust:100%;-ms-text-size-adjust:100%}body{margin:0}article,aside,details,figcaption,figure,footer,
header,hgroup,main,menu,nav,section,summary{display:block}audio,canvas,progress,video{display:inline-block;
vertical-align:baseline}audio:not([controls]){display:none;height:0}[hidden],
template{display:none}a{background-color:transparent}a:active,a:hover{outline:0}abbr[title]{border-bottom:1px
dotted}b,strong{font-weight:700}dfn{font-style:italic}h1{margin:.67em 0;font-size:2em}mark{color:#000;
background:#ff0}small{font-size:80%}sub,sup{position:relative;font-size:75%;line-height:0;
vertical-align:baseline}sup{top:-.5em}sub{bottom:-.25em}img{border:0}svg:not(:root){overflow:hidden}figure{margin:1em
```

Fig-2.3 (bootstrap.min.css file)



```
body.full-content, body {
    /* background: #f4f6f9; */
    background-image: url('../images/bg.jpg');
    background-size: cover;
    padding-top: 0;
}

.limit-chars li:last-child{
    width:15vw;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    line-height: 14px;
    margin-top:-2px;
}

.shadow-sm {
    box-shadow: 0 0.125rem 0.25rem rgba(0, 0, 0, 0.075) !important;
}

.shadow {
    box-shadow: 0 0.5rem 1rem rgba(0, 0, 0, 0.15) !important;
}

.shadow-lg {
    box-shadow: 0 1rem 3rem rgba(0, 0, 0, 0.175) !important;
}
```

Fig-2.4 (custom.css file)



```
.dropdown-menu.dropdown-message li:hover.dropdown-footer a,body{background:#fff}#loading,.fill,.loading-inner,.p
/*!
* bootstrap-notifications v1.0.3 (https://skywalkapps.github.io/bootstrap-notifications)
* Copyright 2017 Martin Staněk
* Licensed under MIT
*/.dropdown-container{position:absolute;top:100%;left:0;z-index:1000;display:none;float:left;min-width:200px;max-width:330
```

Fig-2.5 (style.css file)

This is a segment of a CSS (Cascading Style Sheets) file. It's a combination of *Bootstrap v3.3.7* and *Normalize.css v3.0.3*, two popular libraries for styling HTML documents.

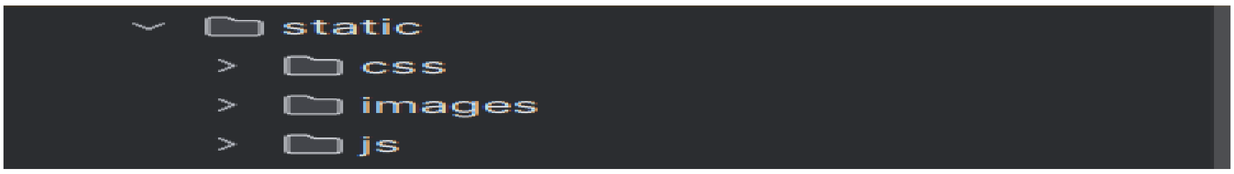
a) Bootstrap is a front-end framework that helps with responsive design and provides a standardized look and feel for web applications.
b) Normalize.css is a small CSS file that makes browsers render elements more consistently and in line with modern standards. It helps "normalize" the differences in styling defaults between browsers.

We can access the full code for both Bootstrap and Normalize.css on their official websites or GitHub repositories.

- **GitHub**: The source code is available on GitHub at Bootstrap GitHub Repository.
- **GitHub**: We can download Normalize.css v3.0.3 directly from its GitHub Repository.

➢ **images**: for storing images, which will render after our project deployment.

➢ **js**: for storing js files, in which we embedded the business logic.

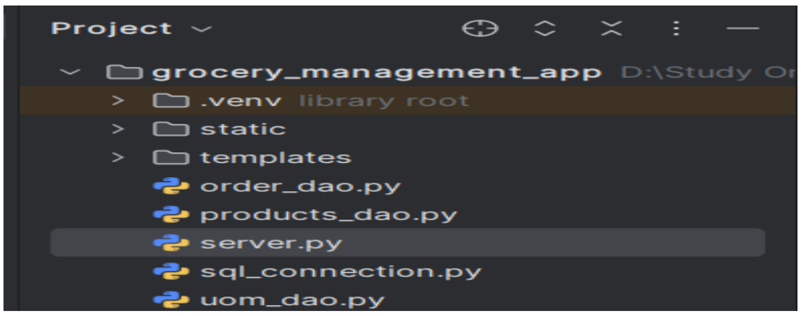


```
static
    css
    images
    js
```

Fig-2.6 (creating static directory)

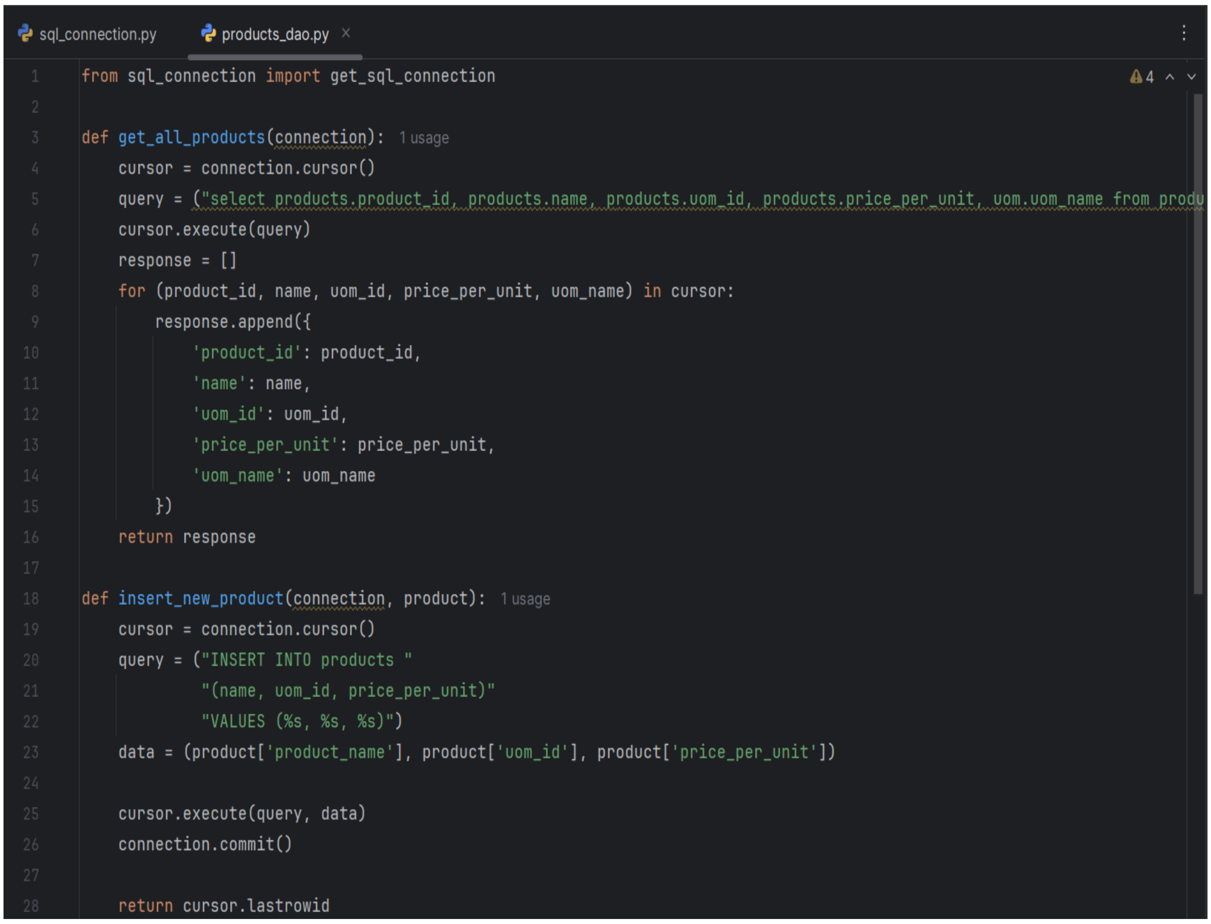
7) **Create DAO files:**

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

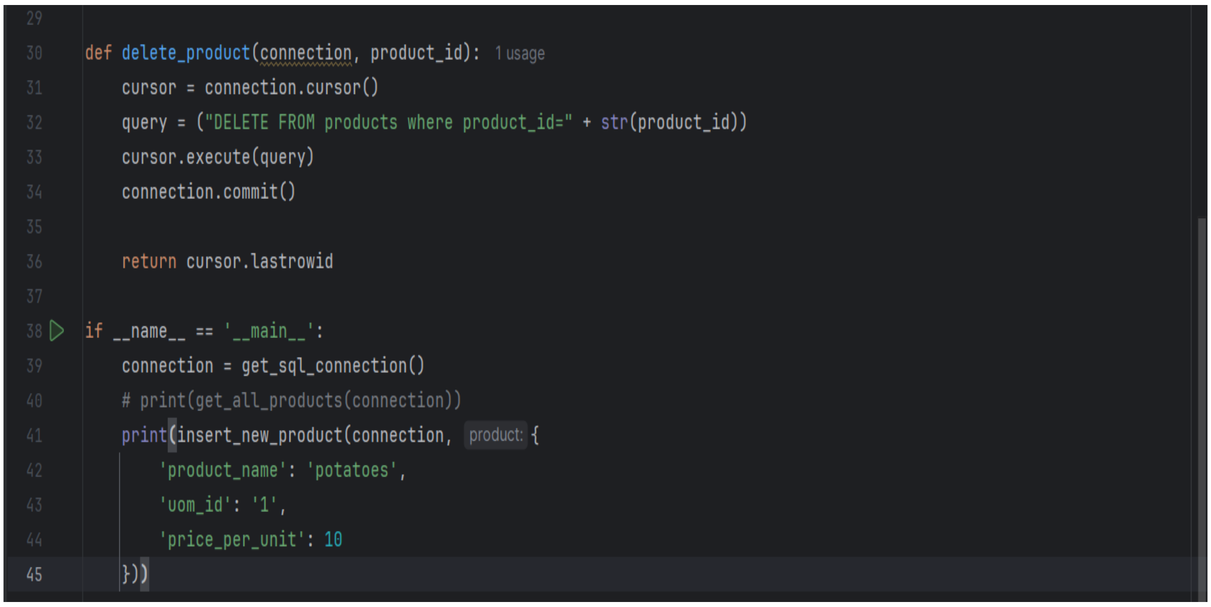


```
Project
grocery_management_app  D:\Study Or
    .venv  library root
    static
    templates
    order_dao.py
    products_dao.py
    server.py
    sql_connection.py
    uom_dao.py
```

Fig-2.7 (creating dao files)

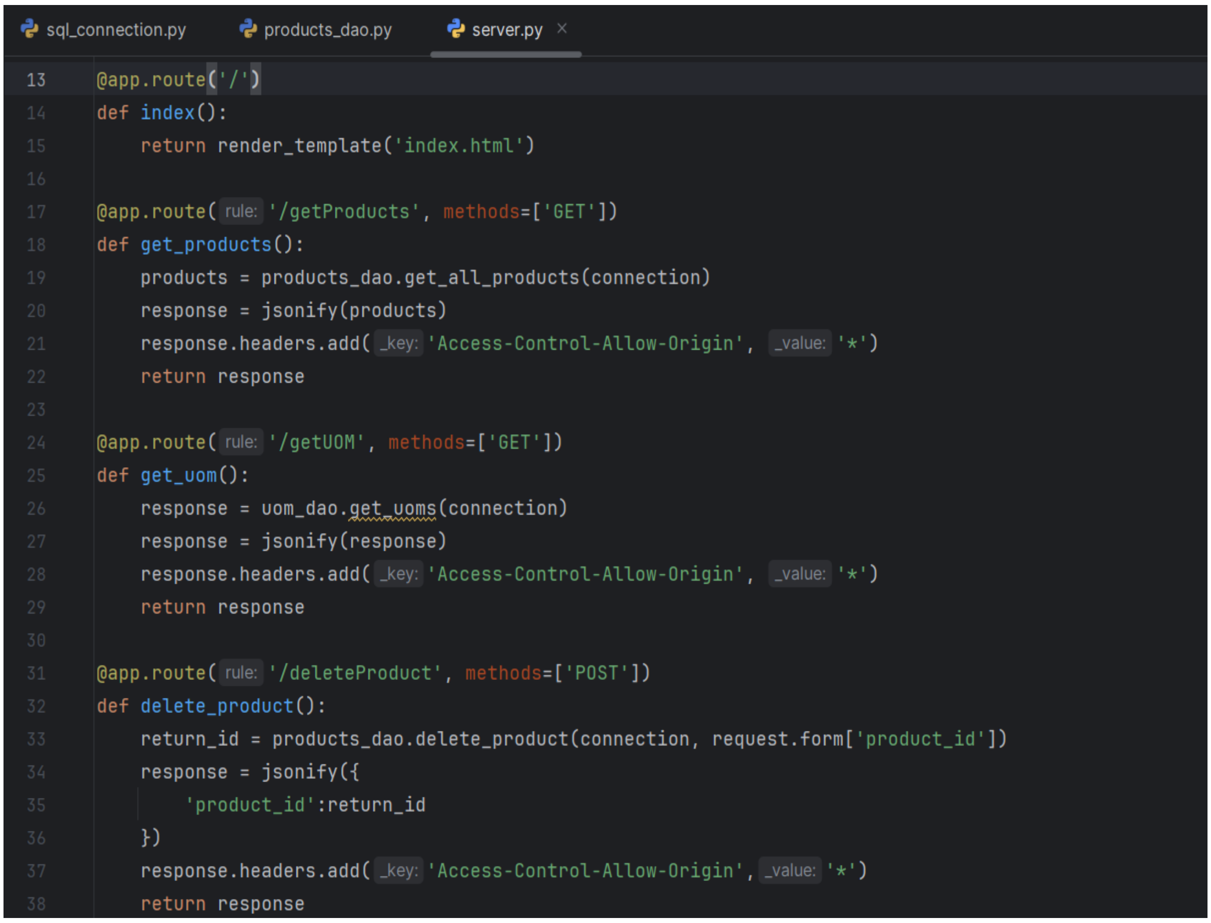


```python
from sql_connection import get_sql_connection

def get_all_products(connection):  1 usage
    cursor = connection.cursor()
    query = ("select products.product_id, products.name, products.uom_id, products.price_per_unit, uom.uom_name from produ
    cursor.execute(query)
    response = []
    for (product_id, name, uom_id, price_per_unit, uom_name) in cursor:
        response.append({
            'product_id': product_id,
            'name': name,
            'uom_id': uom_id,
            'price_per_unit': price_per_unit,
            'uom_name': uom_name
        })
    return response

def insert_new_product(connection, product):  1 usage
    cursor = connection.cursor()
    query = ("INSERT INTO products "
             "(name, uom_id, price_per_unit)"
             "VALUES (%s, %s, %s)")
    data = (product['product_name'], product['uom_id'], product['price_per_unit'])

    cursor.execute(query, data)
    connection.commit()

    return cursor.lastrowid
```

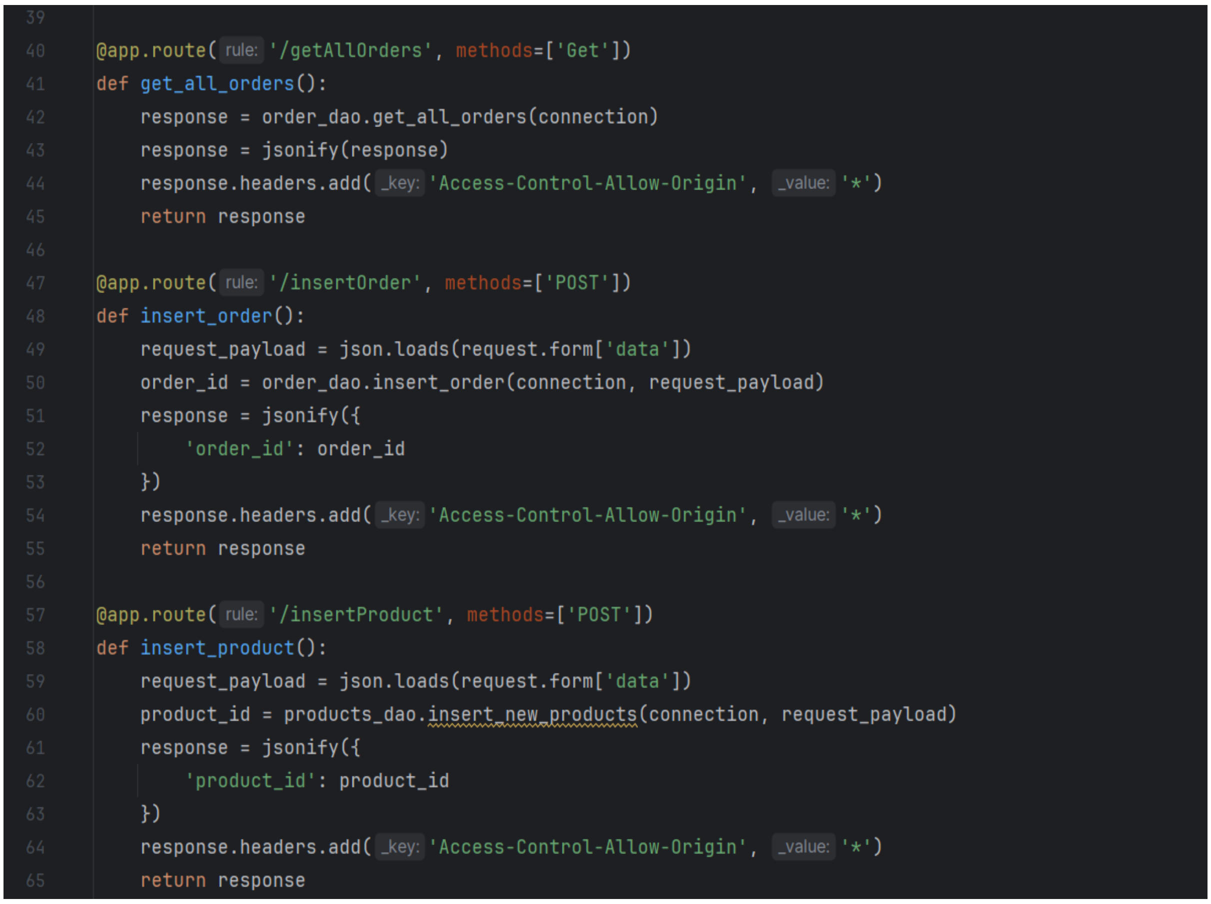


```python

def delete_product(connection, product_id):  1 usage
    cursor = connection.cursor()
    query = ("DELETE FROM products where product_id=" + str(product_id))
    cursor.execute(query)
    connection.commit()

    return cursor.lastrowid

if __name__ == '__main__':
    connection = get_sql_connection()
    # print(get_all_products(connection))
    print(insert_new_product(connection, product: {
        'product_name': 'potatoes',
        'uom_id': '1',
        'price_per_unit': 10
    }))
```

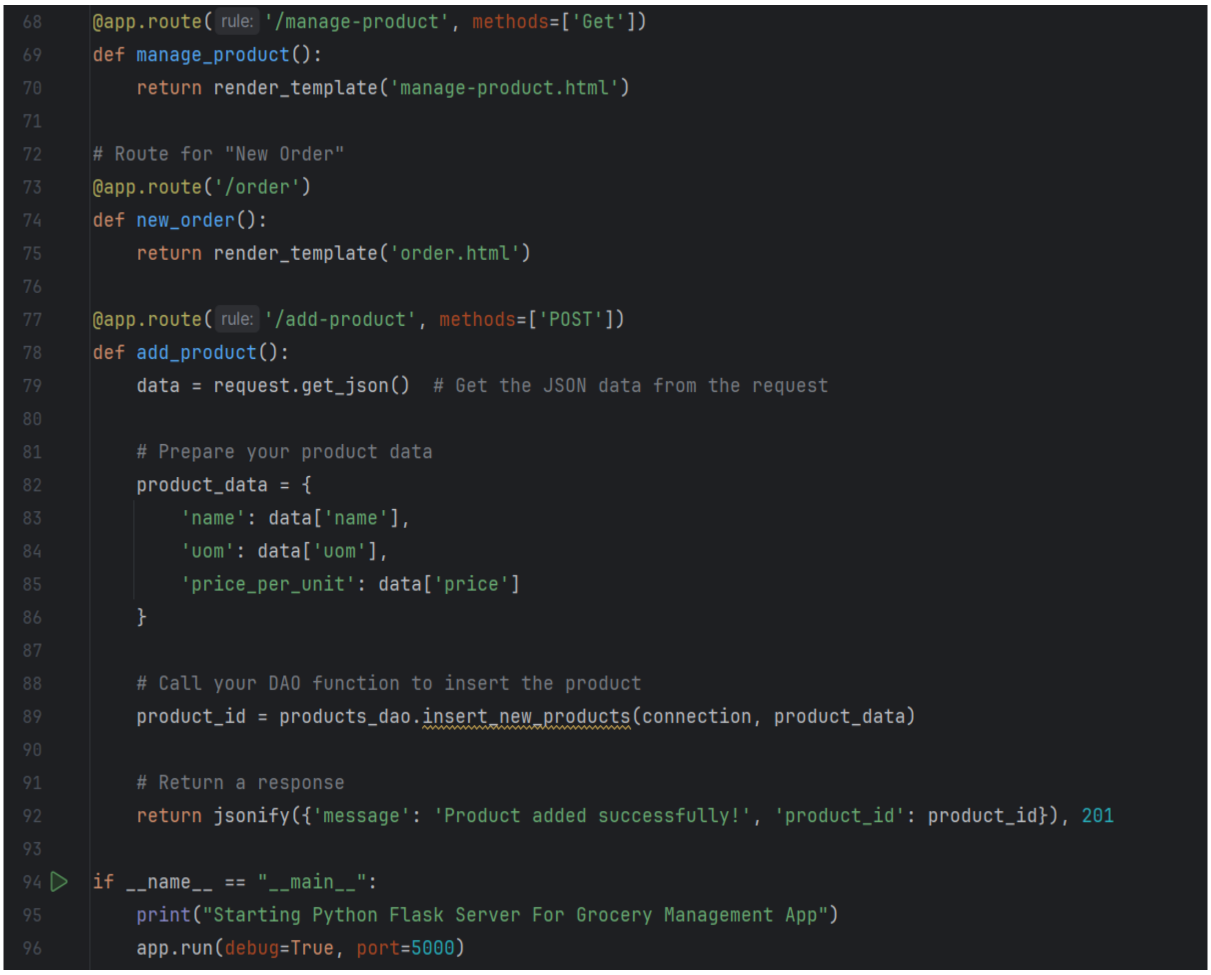
Fig-2.8 (products_dao file)


sql_connection.py   products_dao.py   server.py ×

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route( rule: '/getProducts', methods=['GET'])
def get_products():
    products = products_dao.get_all_products(connection)
    response = jsonify(products)
    response.headers.add( _key: 'Access-Control-Allow-Origin',  _value: '*')
    return response

@app.route( rule: '/getUOM', methods=['GET'])
def get_uom():
    response = uom_dao.get_uoms(connection)
    response = jsonify(response)
    response.headers.add( _key: 'Access-Control-Allow-Origin',  _value: '*')
    return response

@app.route( rule: '/deleteProduct', methods=['POST'])
def delete_product():
    return_id = products_dao.delete_product(connection, request.form['product_id'])
    response = jsonify({
        'product_id':return_id
    })
    response.headers.add( _key: 'Access-Control-Allow-Origin', _value: '*')
    return response
```



```

@app.route( rule: '/getAllOrders', methods=['Get'])
def get_all_orders():
    response = order_dao.get_all_orders(connection)
    response = jsonify(response)
    response.headers.add( _key: 'Access-Control-Allow-Origin',  _value: '*')
    return response

@app.route( rule: '/insertOrder', methods=['POST'])
def insert_order():
    request_payload = json.loads(request.form['data'])
    order_id = order_dao.insert_order(connection, request_payload)
    response = jsonify({
        'order_id': order_id
    })
    response.headers.add( _key: 'Access-Control-Allow-Origin',  _value: '*')
    return response

@app.route( rule: '/insertProduct', methods=['POST'])
def insert_product():
    request_payload = json.loads(request.form['data'])
    product_id = products_dao.insert_new_products(connection, request_payload)
    response = jsonify({
        'product_id': product_id
    })
    response.headers.add( _key: 'Access-Control-Allow-Origin',  _value: '*')
    return response
```

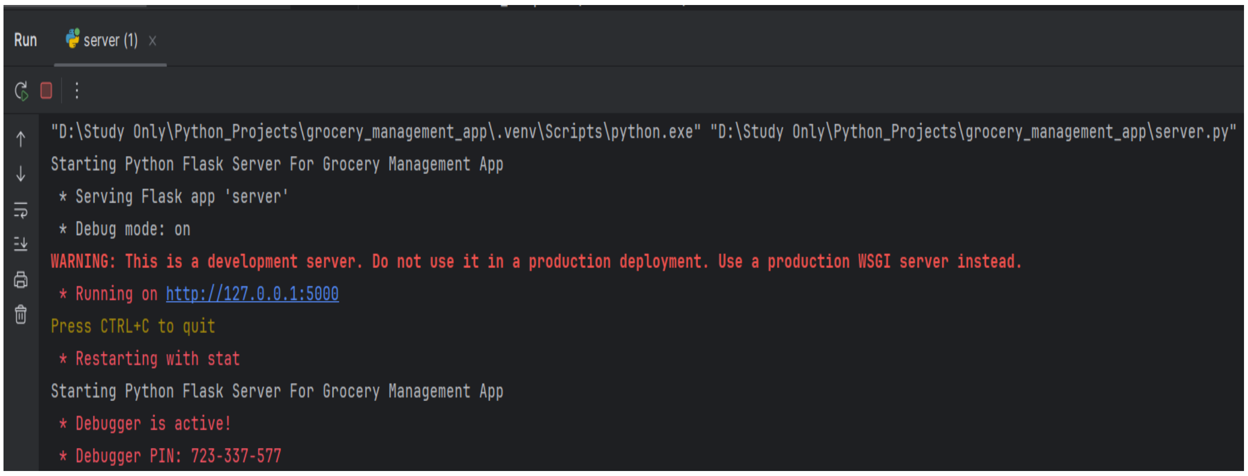


```python
@app.route( rule: '/manage-product', methods=['Get'])
def manage_product():
    return render_template('manage-product.html')

# Route for "New Order"
@app.route('/order')
def new_order():
    return render_template('order.html')

@app.route( rule: '/add-product', methods=['POST'])
def add_product():
    data = request.get_json()  # Get the JSON data from the request

    # Prepare your product data
    product_data = {
        'name': data['name'],
        'uom': data['uom'],
        'price_per_unit': data['price']
    }

    # Call your DAO function to insert the product
    product_id = products_dao.insert_new_products(connection, product_data)

    # Return a response
    return jsonify({'message': 'Product added successfully!', 'product_id': product_id}), 201

if __name__ == "__main__":
    print("Starting Python Flask Server For Grocery Management App")
    app.run(debug=True, port=5000)
```

Fig-2.9 (server.py file)

Results:

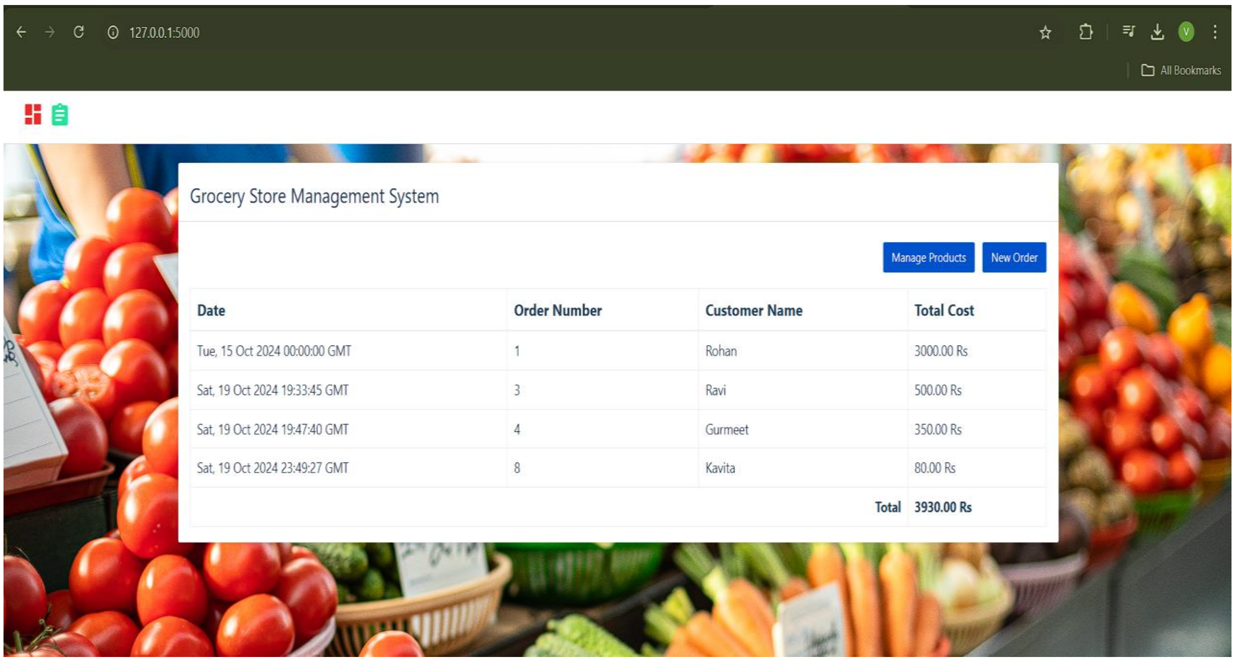
➢ Firstly, we have to start the Flask server to deploy our web application on the local server.



```
"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-3.0 (Deployment of Project)

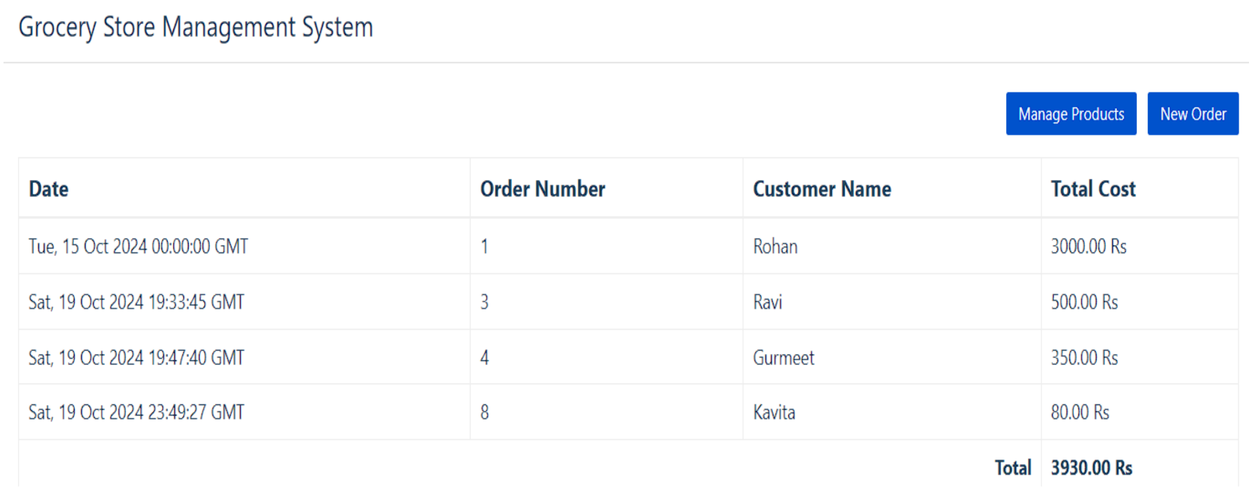
➢ Now we have to copy the URL and paste it on any web browser with '/' because it is mapped in the index() function inside server.py to render or load the index page.



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-3.1 (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.



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-3.2 (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.

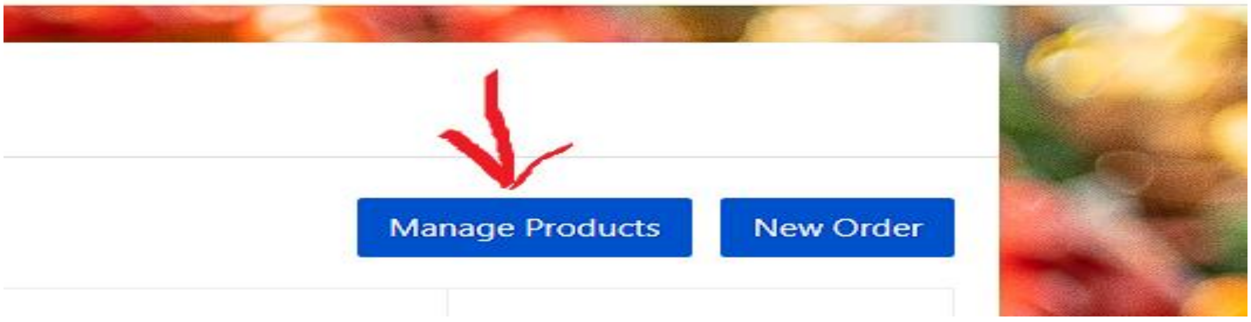
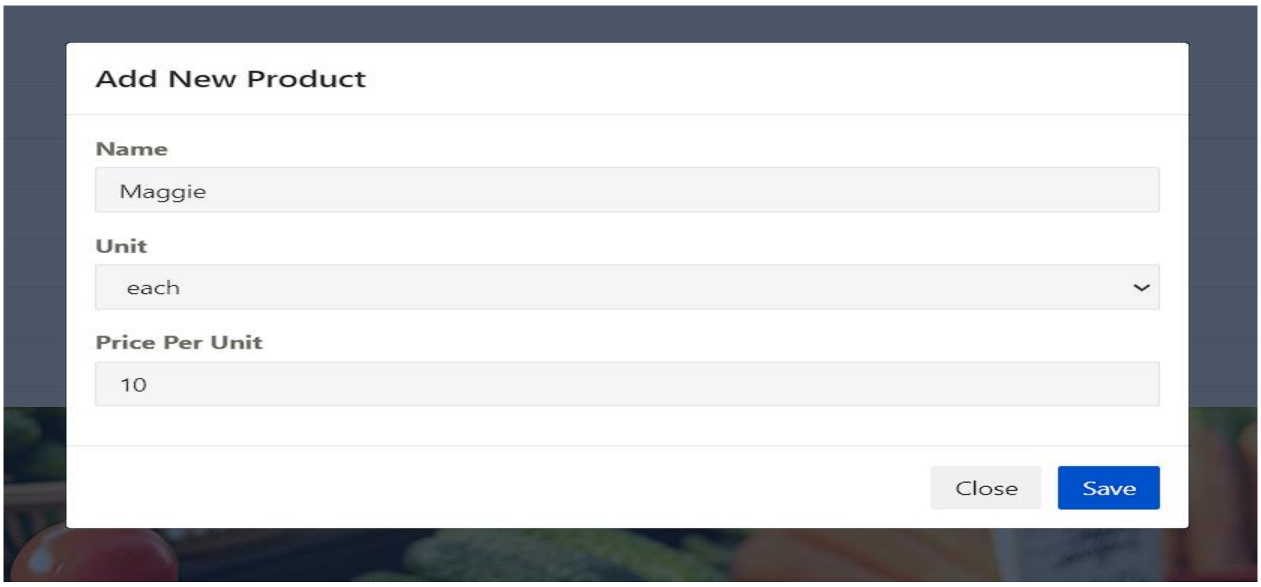


Fig-3.3 (Manage Products Button)


Add New Product

Name

Maggie

Unit

each

Price Per Unit

10

Fig-3.4 (Add New Product dialog box)

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

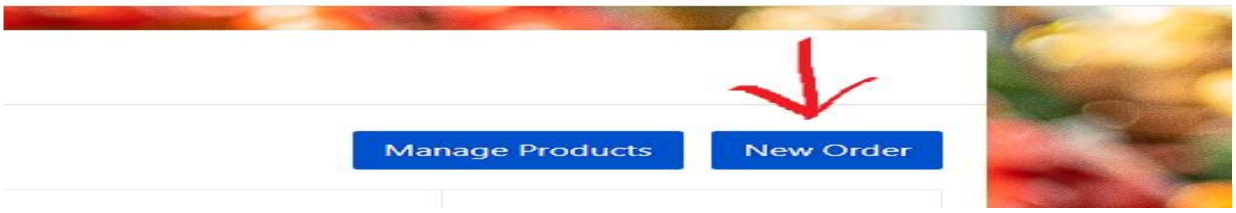
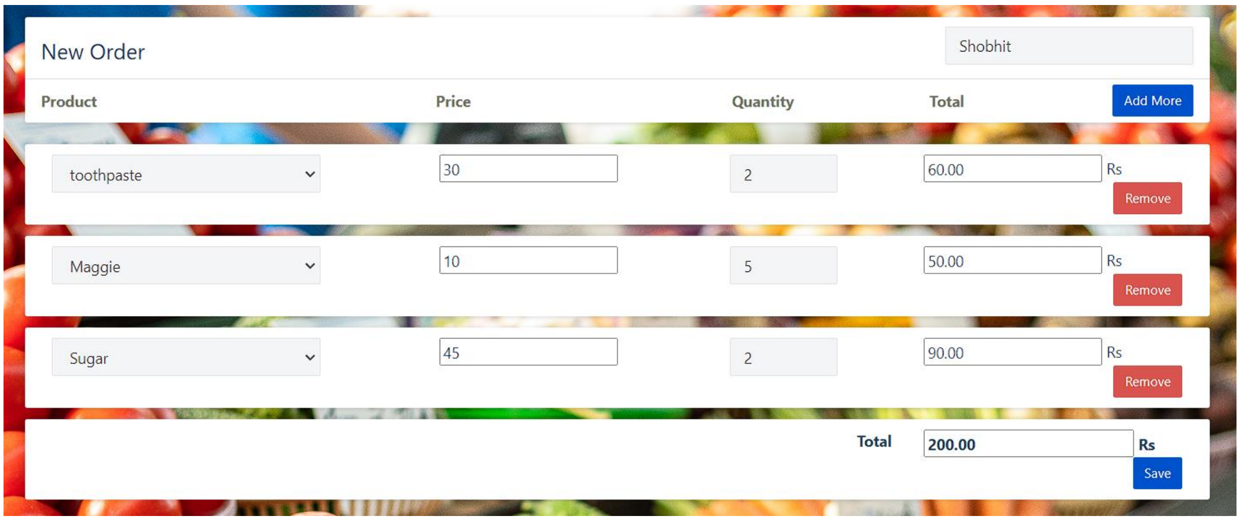


Fig-3.5 (New Order Button)



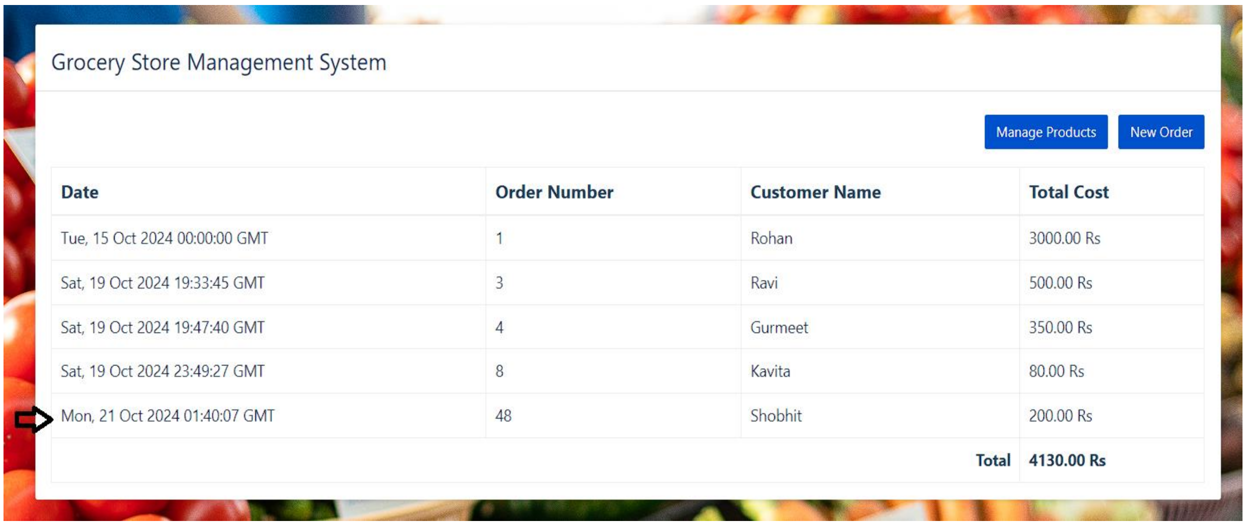
New Order

Shobhit

Product	Price	Quantity	Total
toothpaste	30	2	60.00 Rs
Maggie	10	5	50.00 Rs
Sugar	45	2	90.00 Rs
		Total	200.00 Rs

Fig-3.6 (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.



Grocery Store Management System

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-3.7 (Create a new order for Customer)