Lesson 05 Demo 02

Building a Grocery App with GitHub Copilot and Flask

Objective: To build starter code for an e-commerce application using GitHub Copilot, Flask, and MySQL to construct REST APIs, and Postman for comprehensive testing and interaction

Tools required: Visual Studio Code, GitHub Copilot, MySQL, and Postman

Prerequisites: Lesson 05 demo 01

Steps to be followed:

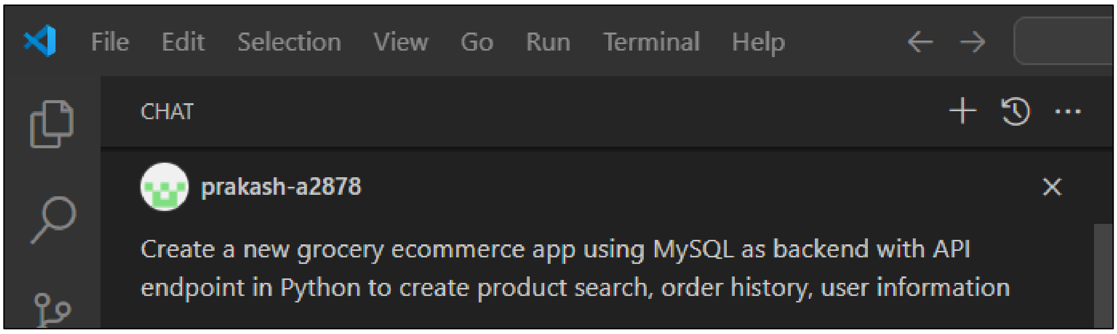
1. Utilize GitHub Copilot to generate the REST APIs for the grocery e-commerce application
2. Insert data into MySQL using REST APIs through Postman
3. Fetch and display MySQL data in the browser

Note: Generative AI tool used in this exercise can produce varied outputs even when presented with similar prompts. Thus, you may get different output for the same prompt

Step 1: Utilize GitHub Copilot to generate the REST APIs for the grocery e-commerce application

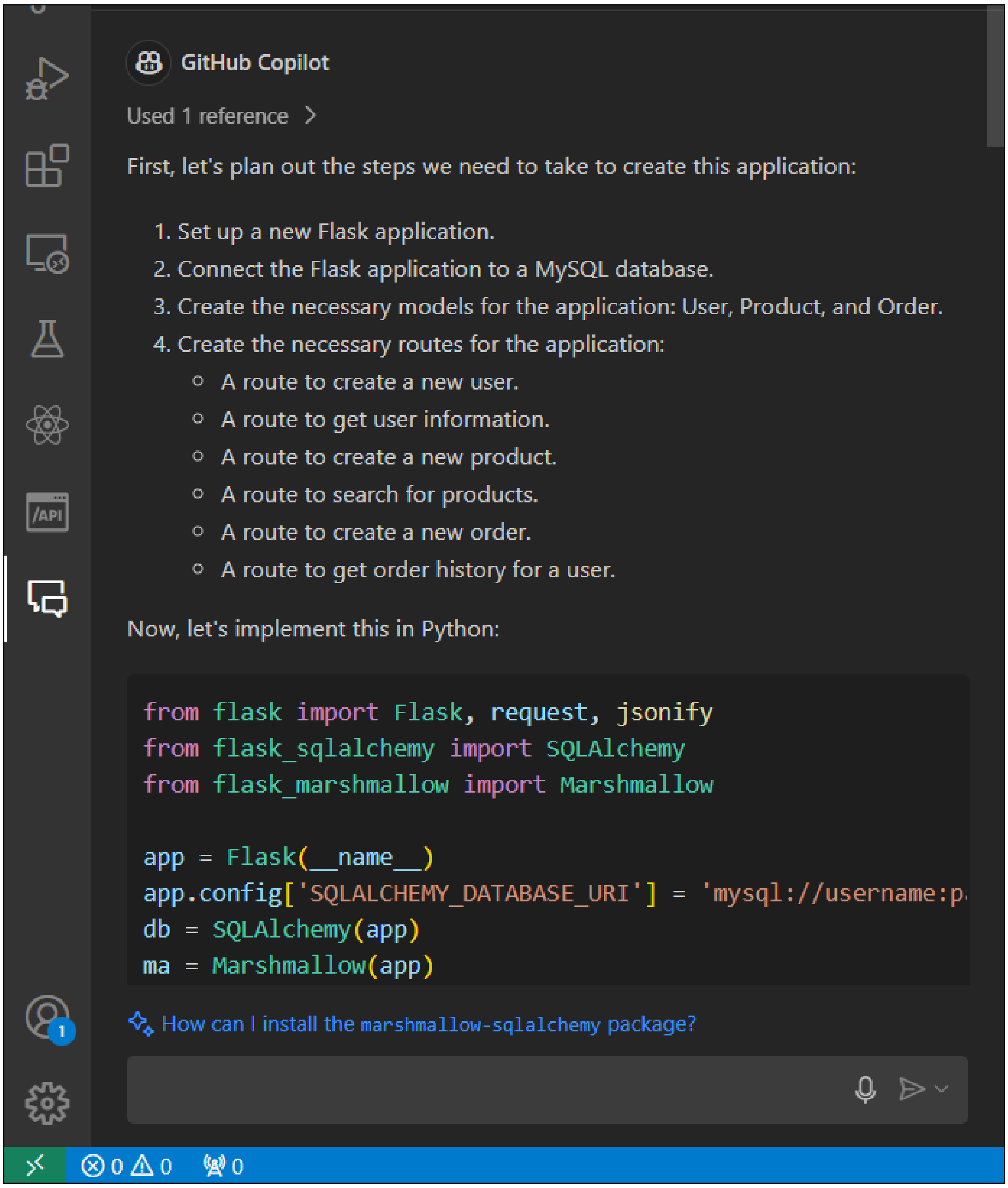
1.1 Open Visual Studio Code and create a new Python file. In this case, the Python file is named **myGrocery.py**. Open **GitHub Copilot** in the sidebar of Visual Studio Code and provide the following prompt to generate the REST APIs for the grocery e-commerce app.

Create a new grocery ecommerce app using MySQL as backend with API endpoint in Python to create product search, order history, user information



Create a new grocery ecommerce app using MySQL as backend with API endpoint in Python to create product search, order history, user information

It generates the following Python code:



GitHub Copilot

Used 1 reference >

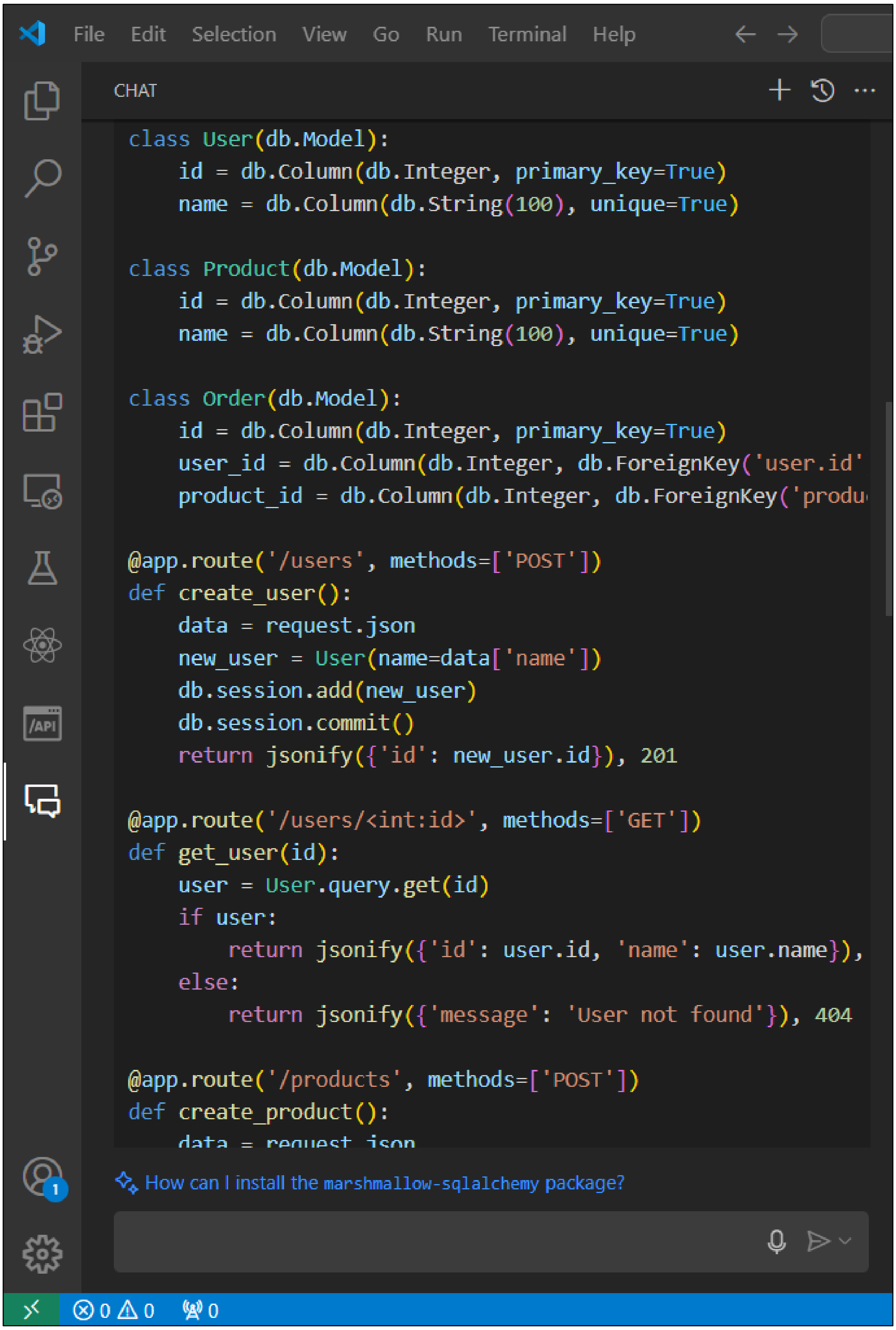
First, let's plan out the steps we need to take to create this application:

1. Set up a new Flask application.
2. Connect the Flask application to a MySQL database.
3. Create the necessary models for the application: User, Product, and Order.
4. Create the necessary routes for the application:
    - A route to create a new user.
    - A route to get user information.
    - A route to create a new product.
    - A route to search for products.
    - A route to create a new order.
    - A route to get order history for a user.

Now, let's implement this in Python:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://username:p
db = SQLAlchemy(app)
ma = Marshmallow(app)
```

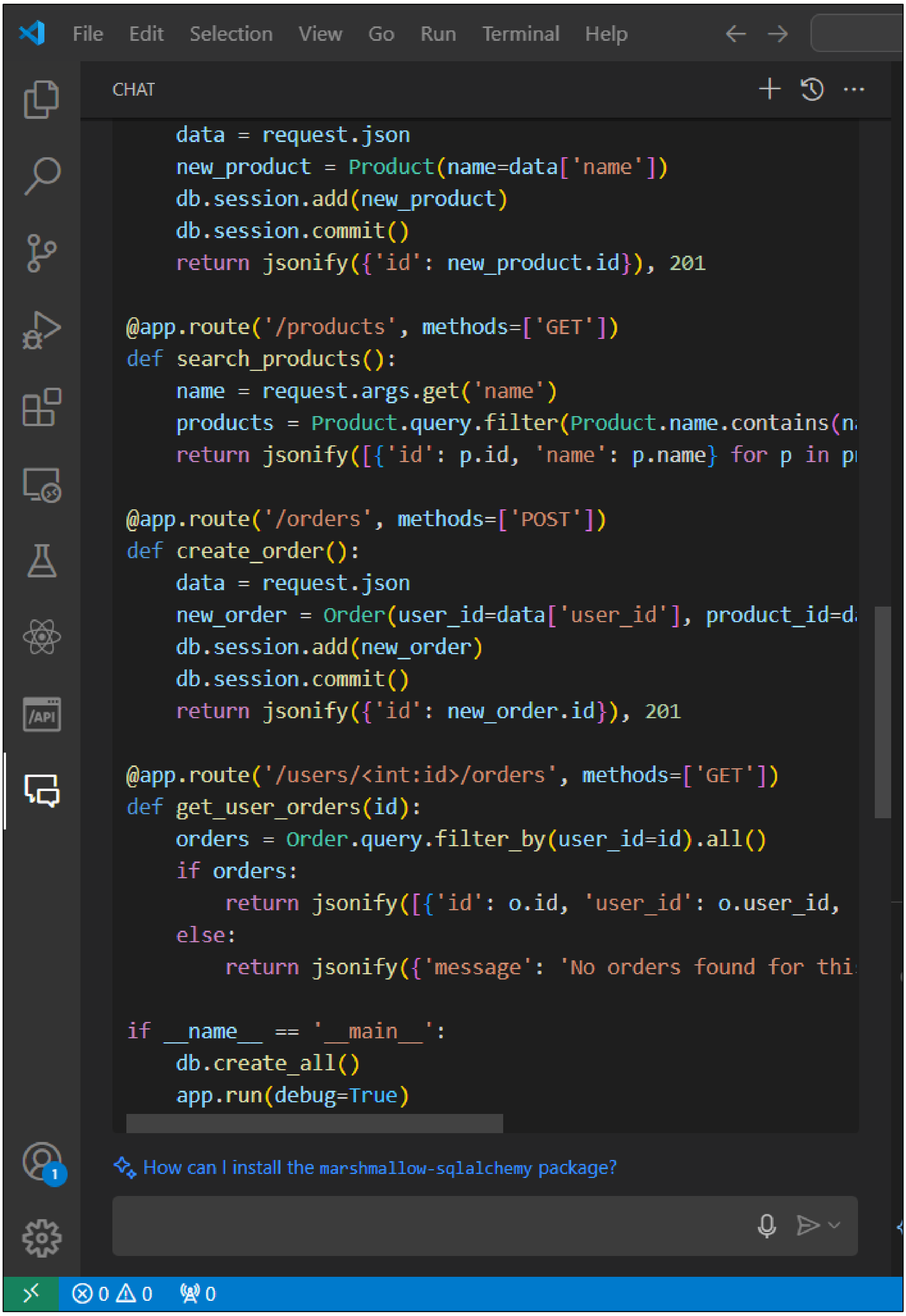
How can I install the `marshmallow-sqlalchemy` package?



```python
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True)

class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True)

class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'
    product_id = db.Column(db.Integer, db.ForeignKey('produ

@app.route('/users', methods=['POST'])
def create_user():
    data = request.json
    new_user = User(name=data['name'])
    db.session.add(new_user)
    db.session.commit()
    return jsonify({'id': new_user.id}), 201

@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    user = User.query.get(id)
    if user:
        return jsonify({'id': user.id, 'name': user.name}),
    else:
        return jsonify({'message': 'User not found'}), 404

@app.route('/products', methods=['POST'])
def create_product():
    data = request.json
```

How can I install the `marshmallow-sqlalchemy` package?

⊗ 0 ⚠ 0 0



```
    data = request.json
    new_product = Product(name=data['name'])
    db.session.add(new_product)
    db.session.commit()
    return jsonify({'id': new_product.id}), 201

@app.route('/products', methods=['GET'])
def search_products():
    name = request.args.get('name')
    products = Product.query.filter(Product.name.contains(n
    return jsonify([{'id': p.id, 'name': p.name} for p in p

@app.route('/orders', methods=['POST'])
def create_order():
    data = request.json
    new_order = Order(user_id=data['user_id'], product_id=d
    db.session.add(new_order)
    db.session.commit()
    return jsonify({'id': new_order.id}), 201

@app.route('/users/<int:id>/orders', methods=['GET'])
def get_user_orders(id):
    orders = Order.query.filter_by(user_id=id).all()
    if orders:
        return jsonify([{'id': o.id, 'user_id': o.user_id,
    else:
        return jsonify({'message': 'No orders found for thi

if __name__ == '__main__':
    db.create_all()
    app.run(debug=True)
```

How can I install the `marshmallow-sqlalchemy` package?

The generated code is as follows:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql://username:password@localhost/database'
db = SQLAlchemy(app)
ma = Marshmallow(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True)

class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True)

class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    product_id = db.Column(db.Integer, db.ForeignKey('product.id'))

@app.route('/users', methods=['POST'])
def create_user():
    data = request.json
    new_user = User(name=data['name'])
    db.session.add(new_user)
    db.session.commit()
    return jsonify({'id': new_user.id}), 201

@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    user = User.query.get(id)
    if user:
        return jsonify({'id': user.id, 'name': user.name})
    else:
        return jsonify({'message': 'User not found'}), 404

@app.route('/products', methods=['POST'])
def create_product():
    data = request.json
    new_product = Product(name=data['name'])
    db.session.add(new_product)
    db.session.commit()
    return jsonify({'id': new_product.id}), 201

@app.route('/products', methods=['GET'])
def search_products():
    name = request.args.get('name')
    products = Product.query.filter(Product.name.contains(name))
    return jsonify([{'id': p.id, 'name': p.name} for p in products])

@app.route('/orders', methods=['POST'])
def create_order():
    data = request.json
    new_order = Order(user_id=data['user_id'], product_id=data['product_id'])
    db.session.add(new_order)
    db.session.commit()
    return jsonify({'id': new_order.id}), 201

@app.route('/users/<int:id>/orders', methods=['GET'])
def get_user_orders(id):
    orders = Order.query.filter_by(user_id=id).all()
    if orders:
        return jsonify([{'id': o.id, 'user_id': o.user_id, 'product_id': o.product_id} for o
in orders])
    else:
        return jsonify({'message': 'No orders found for this user'}), 404

if __name__ == '__main__':
    db.create_all()
    app.run(debug=True)
```

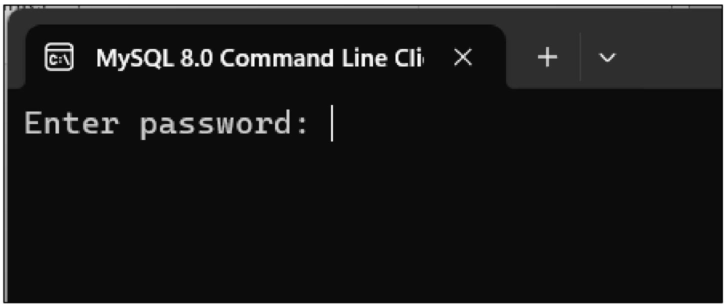
1.2 Make the changes in the code per the requirement and save it in the file **myGrocery.py**. The modified code is:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow
from urllib.parse import quote

app = Flask(__name__)
password = quote("pass123")  # URL encode the password
app.config['SQLALCHEMY_DATABASE_URI'] =
f'mysql+pymysql://root:{password}@localhost/grocery_ecommerce'
db = SQLAlchemy(app)
ma = Marshmallow(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True)

class Product(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True)

class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    product_id = db.Column(db.Integer, db.ForeignKey('product.id'))

@app.route('/', methods=['GET'])
def home():
    return "Welcome to the Grocery Ecommerce API!", 200

@app.route('/users', methods=['POST'])
def create_users():
    data = request.get_json()
    print(data)
    if isinstance(data, list):  # Check if the data is a list of users
        new_users = []
        for item in data:
            new_user = User(name=item['name'])
            db.session.add(new_user)
            new_users.append(new_user)
        db.session.commit()
        return jsonify([{'id': user.id, 'name': user.name} for user in new_users]), 201
    else:
        return jsonify({'message': 'Invalid input format, expected a list of users.'}), 400

@app.route('/users', methods=['GET'])
def get_users():
    users = User.query.all()
    table = '<table border="1"><tr><th>ID</th><th>Name</th></tr>'
    for user in users:
        table += f'<tr><td>{user.id}</td><td>{user.name}</td></tr>'
    table += '</table>'
    return table

@app.route('/products', methods=['POST'])
def create_products():
    data = request.get_json()
    if isinstance(data, list):  # Check if the data is a list of products
        new_products = []
        for item in data:
            new_product = Product(name=item['name'])
            db.session.add(new_product)
            new_products.append(new_product)
        db.session.commit()
        return jsonify([{'id': product.id, 'name': product.name} for product in
new_products]), 201
    else:
        return jsonify({'message': 'Invalid input format, expected a list of products.'}),
400

@app.route('/products', methods=['GET'])
def get_products():
    products = Product.query.all()
    table = '<table border="1"><tr><th>ID</th><th>Name</th></tr>'
    for product in products:
        table += f'<tr><td>{product.id}</td><td>{product.name}</td></tr>'
    table += '</table>'
    return table

@app.route('/orders', methods=['POST'])
def create_orders():
    data = request.get_json()
    if isinstance(data, list):  # Check if the data is a list of orders
        new_orders = []
        for item in data:
            new_order = Order(user_id=item['user_id'], product_id=item['product_id'])
            db.session.add(new_order)
            new_orders.append(new_order)
        db.session.commit()
        return jsonify([{'id': order.id, 'user_id': order.user_id, 'product_id':
order.product_id} for order in new_orders]), 201
    else:
        return jsonify({'message': 'Invalid input format, expected a list of orders.'}),
400

@app.route('/orders', methods=['GET'])
def get_orders():
    orders = Order.query.all()
    table = '<table border="1"><tr><th>ID</th><th>User ID</th><th>Product
ID</th></tr>'
    for order in orders:
        table +=
f'<tr><td>{order.id}</td><td>{order.user_id}</td><td>{order.product_id}</td></tr
>'
    table += '</table>'
    return table
```

```
if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

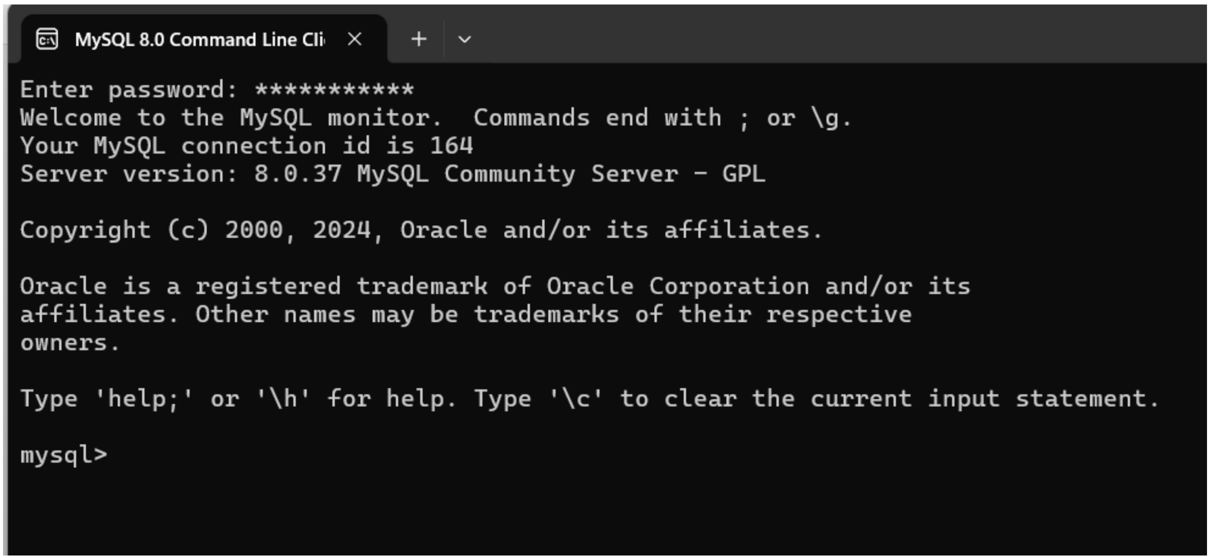
Note: This Flask application provides REST APIs with endpoints **/users** (POST, GET), **/products** (POST, GET), and **/orders** (POST, GET) to create and retrieve Users, Products, and Orders respectively, in a grocery e-commerce platform. It handles JSON data for **POST** methods and returns HTML tables for **GET** methods using SQLAlchemy and Marshmallow.

1.3 Open a browser in a separate window, download MySQL from the below link, and install it. Create the password for MySQL during installation:
https://www.mysql.com/downloads/



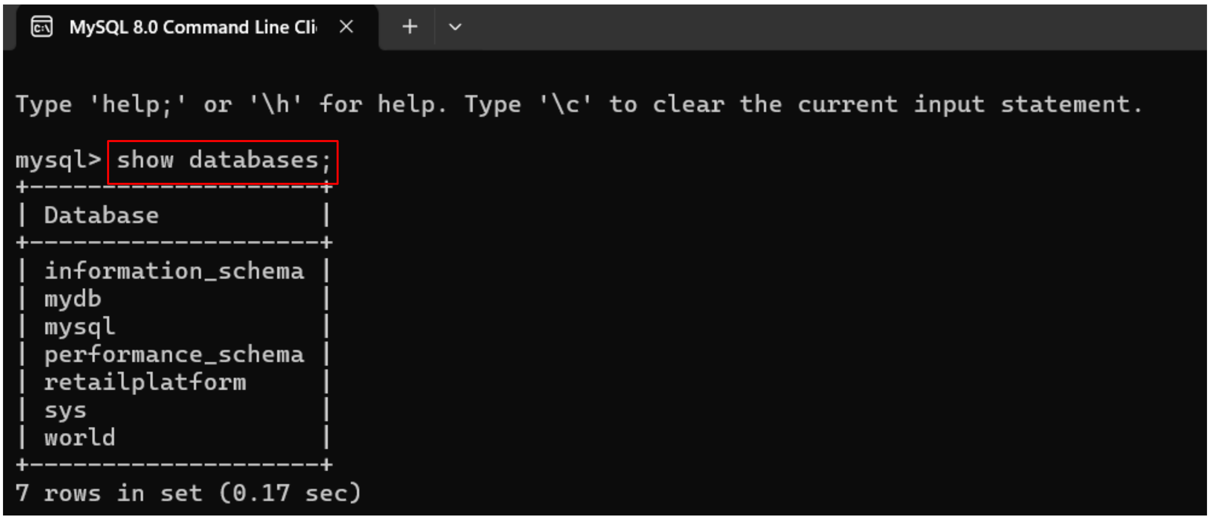
```
Enter password: 
```

Provide the password when prompted and press the Enter key



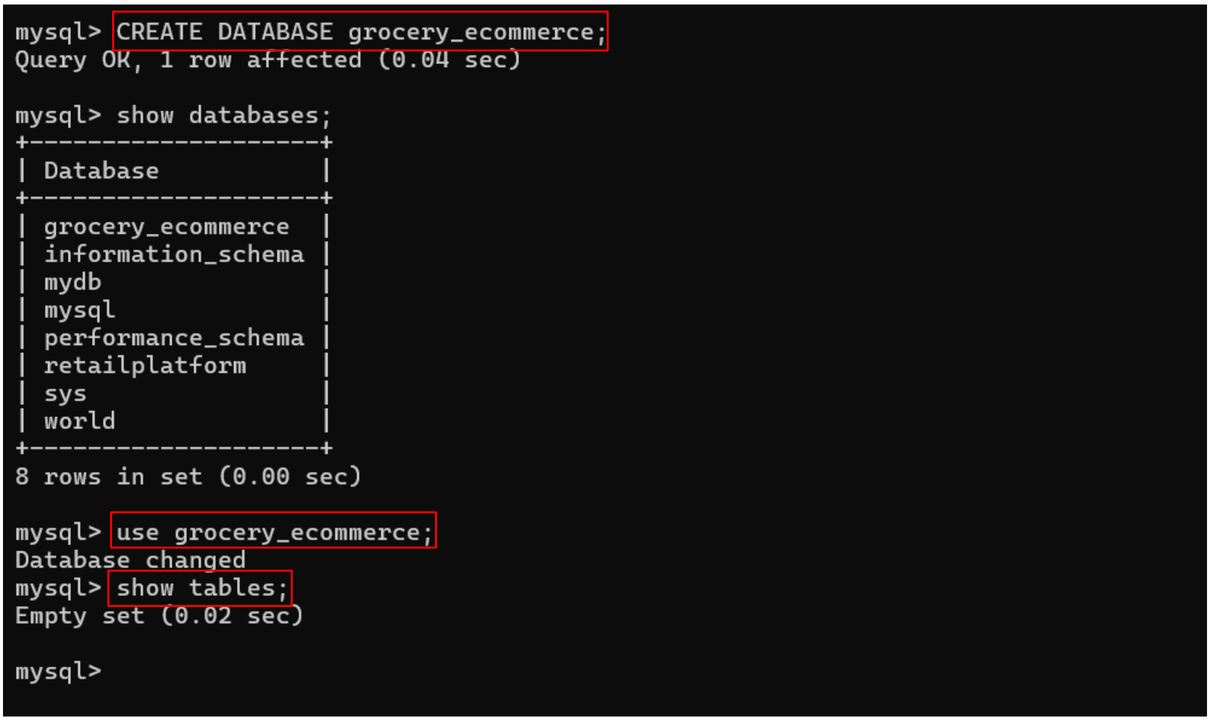
```
Enter password: ***********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 164
Server version: 8.0.37 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

1.4 Open MySQL and list the existing databases using the following command:
show databases;



```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb               |
| mysql              |
| performance_schema |
| retailplatform     |
| sys                |
| world              |
+--------------------+
7 rows in set (0.17 sec)
```

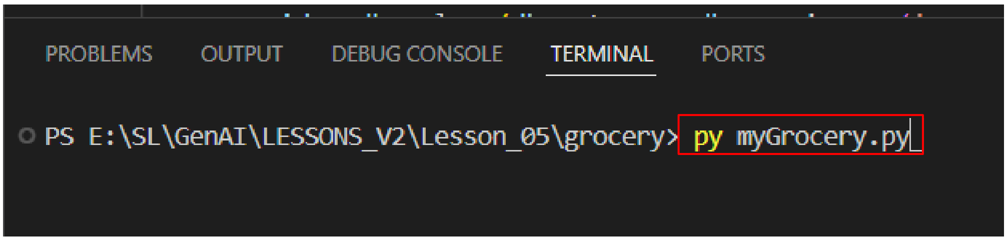
1.5 Create a new database called **grocery_ecommerce**. Initially, the tables in this database are empty using the following commands:
CREATE DATABASE grocery_ecommerce;
show databases;



```
mysql> CREATE DATABASE grocery_ecommerce;
Query OK, 1 row affected (0.04 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| grocery_ecommerce  |
| information_schema |
| mydb               |
| mysql              |
| performance_schema |
| retailplatform     |
| sys                |
| world              |
+--------------------+
8 rows in set (0.00 sec)

mysql> use grocery_ecommerce;
Database changed
mysql> show tables;
Empty set (0.02 sec)

mysql>
```

1.6 Navigate to the Visual Studio and execute the Python code using the following command:

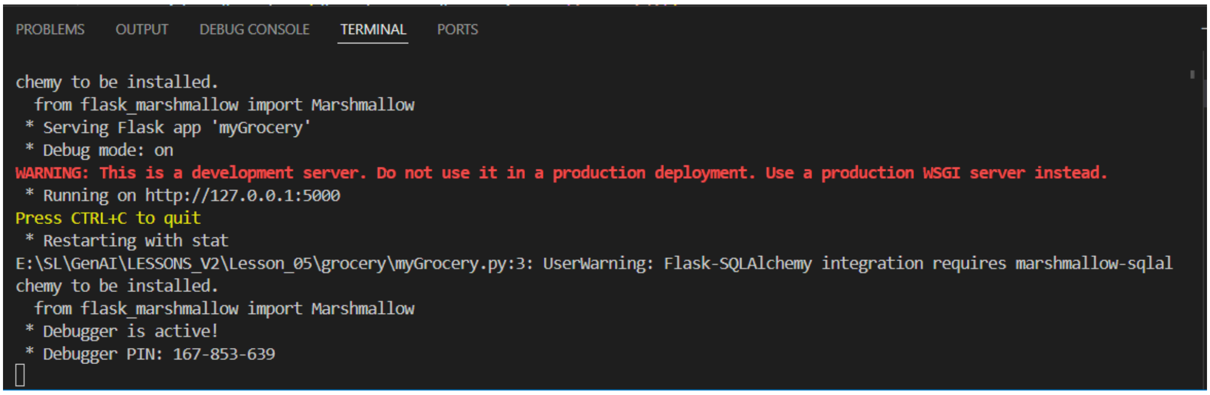
py myGrocery.py



```
PS E:\SL\GenAI\LESSONS_V2\Lesson_05\grocery> py myGrocery.py
```

Note: Create the database **grocery_commerce** before executing the Python code mentioned in **myGrocery.py**

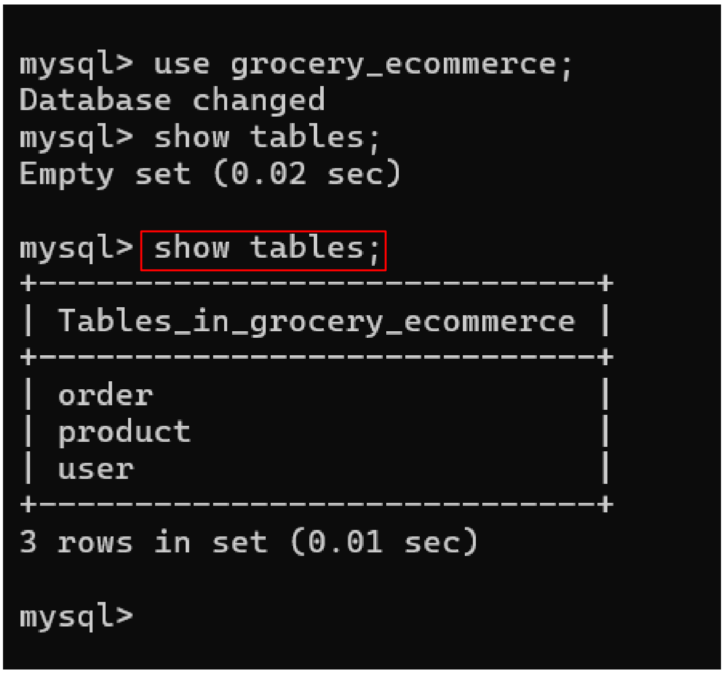
It shows the successful execution of the code:



```
chemy to be installed.
  from flask_marshmallow import Marshmallow
 * Serving Flask app 'myGrocery'
 * 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
E:\SL\GenAI\LESSONS_V2\Lesson_05\grocery\myGrocery.py:3: UserWarning: Flask-SQLAlchemy integration requires marshmallow-sqlal
chemy to be installed.
  from flask_marshmallow import Marshmallow
 * Debugger is active!
 * Debugger PIN: 167-853-639
```

1.7 Navigate to MySQL and list the tables using the following command:

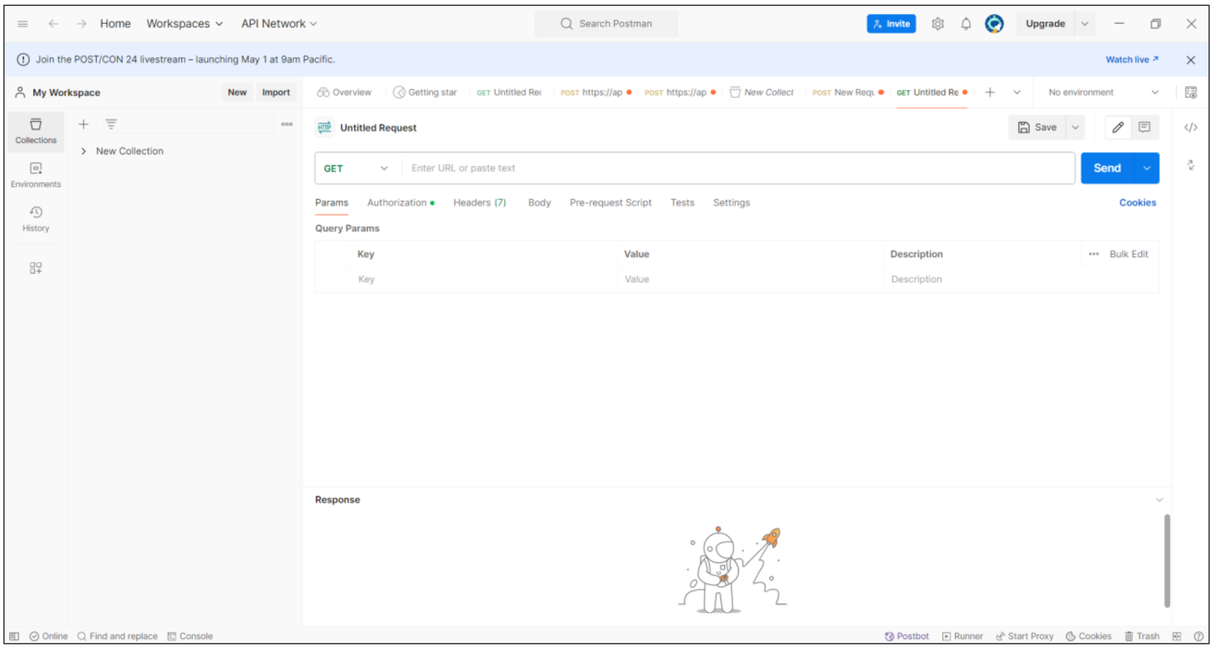
show tables;



```
mysql> use grocery_ecommerce;
Database changed
mysql> show tables;
Empty set (0.02 sec)

mysql> show tables;
+-----------------------------+
| Tables_in_grocery_ecommerce |
+-----------------------------+
| order                       |
| product                     |
| user                        |
+-----------------------------+
3 rows in set (0.01 sec)

mysql>
```

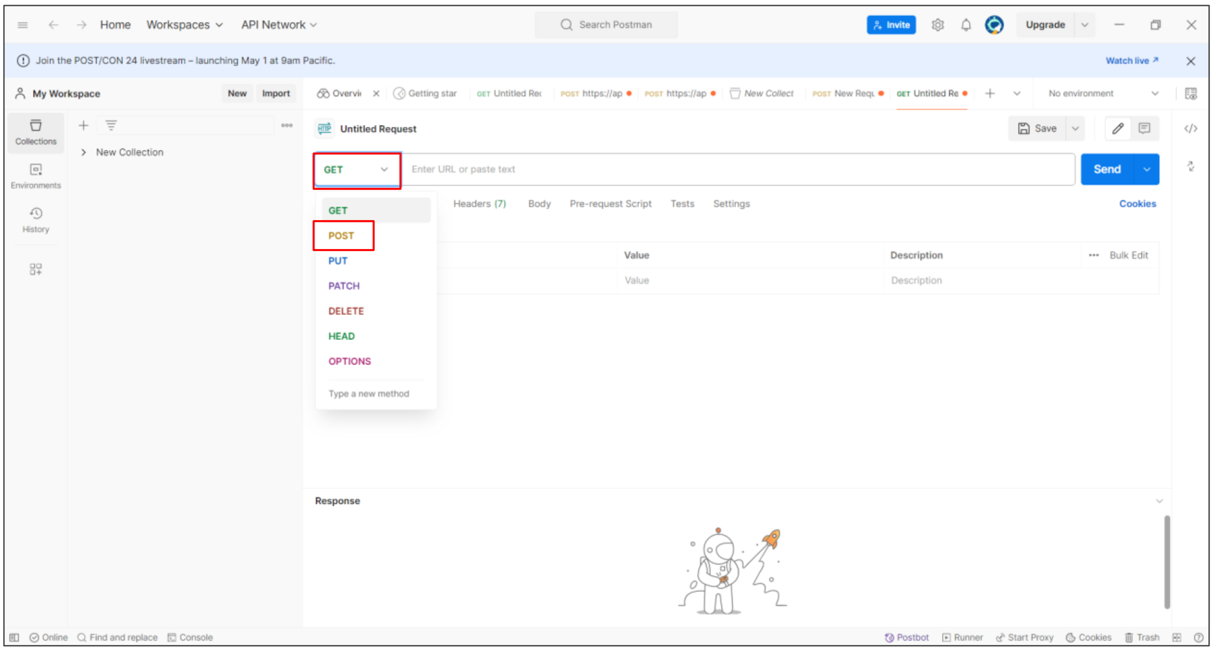
Step 2: Insert data into MySQL using REST APIs through Postman

2.1 Open the Postman application

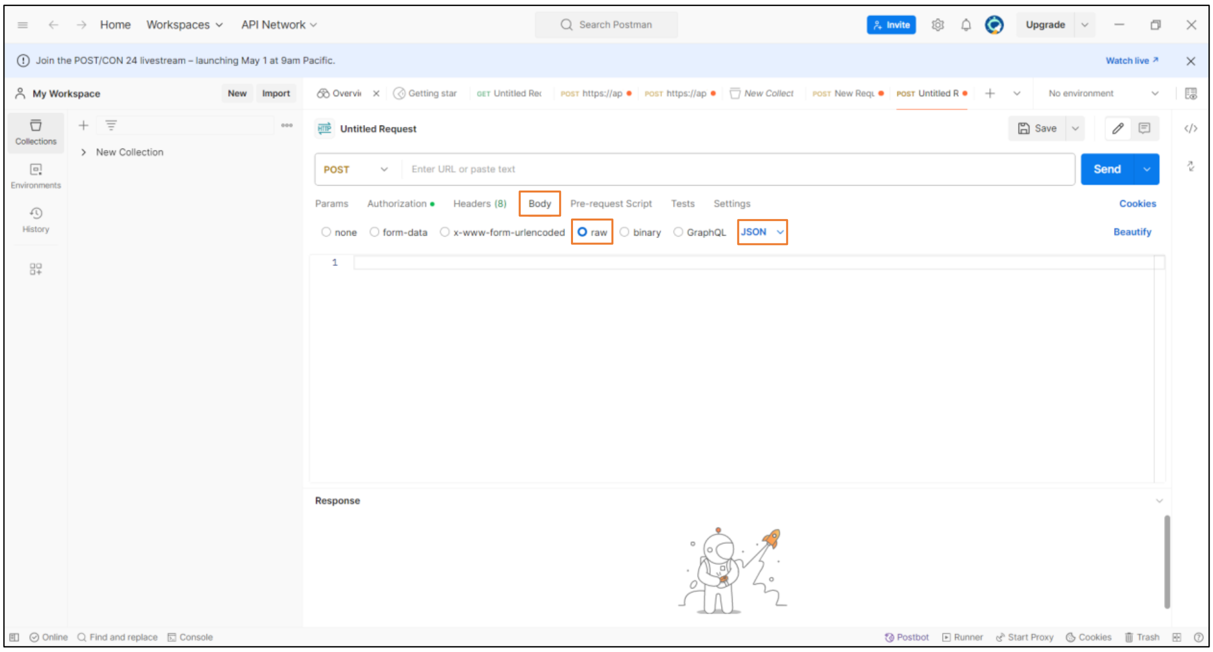


Note: The Postman application can be accessed online via their website at **https://www.postman.com/**, or it can be used offline by downloading and installing it.

2.2 Select the **POST** method from the dropdown menu



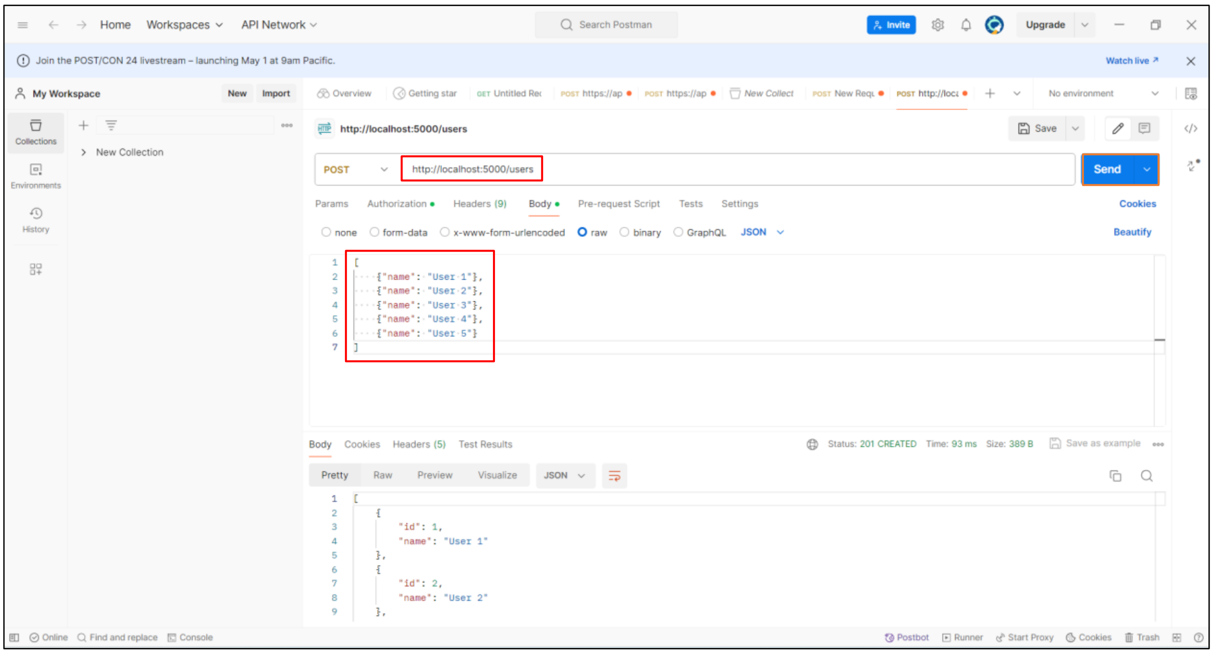
2.3 In the **Body** tab, choose **raw** and ensure the format is set to **JSON**



Note: Ensure that the Python code in Visual Studio is running while inserting data through Postman

2.4 Provide the following URL in the **POST** along with the **JSON** input in the **raw** format. Then, click on **Send** to insert records into the user table in MySQL
http://localhost:5000/users

```
[
        {"name": "User1"},
        {"name": "User2"},
        {"name": "User3"},
        {"name": "User4"},
        {"name": "User5"}
]
```



http://localhost:5000/users

POST http://localhost:5000/users

Send

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
[
    {"name": "User 1"},
    {"name": "User 2"},
    {"name": "User 3"},
    {"name": "User 4"},
    {"name": "User 5"}
]
```

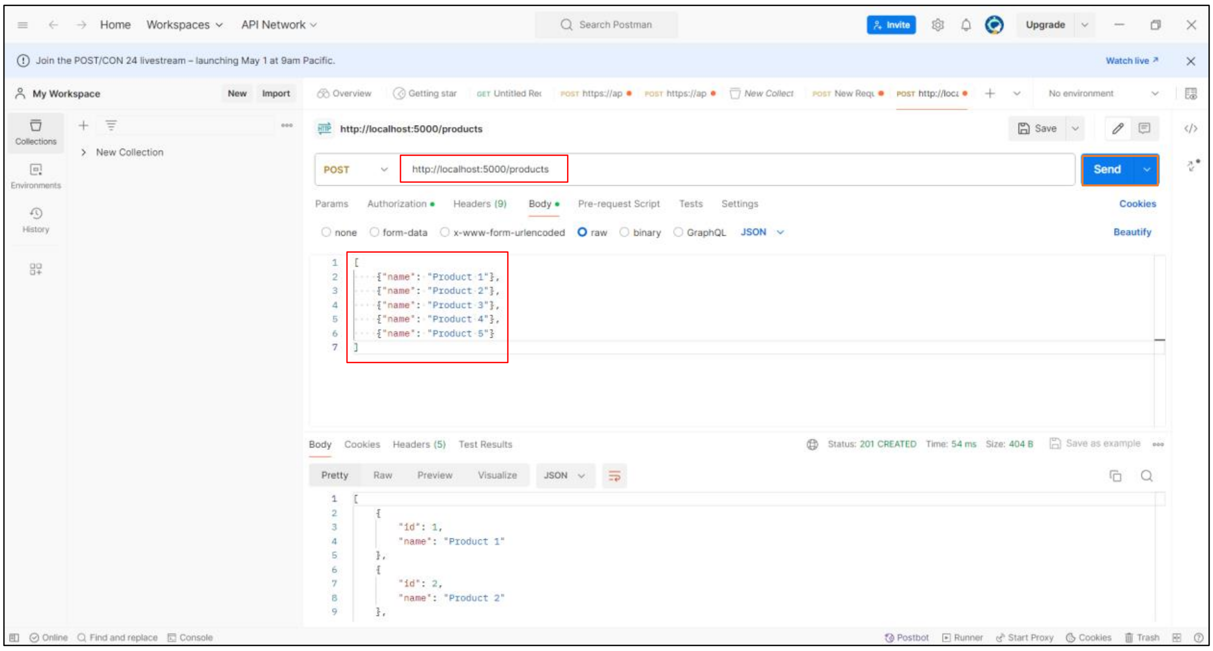
Status: 201 CREATED Time: 93 ms Size: 389 B

```
[
    {
        "id": 1,
        "name": "User 1"
    },
    {
        "id": 2,
        "name": "User 2"
    },
```

2.5 Insert the following records into the product table using Postman POST:

http://localhost:5000/products

```
[
        {"name": "Product1"},
        {"name": "Product2"},
        {"name": "Product3"},
        {"name": "Product4"},
       {"name": "Product5"}
]
```



POST http://localhost:5000/products

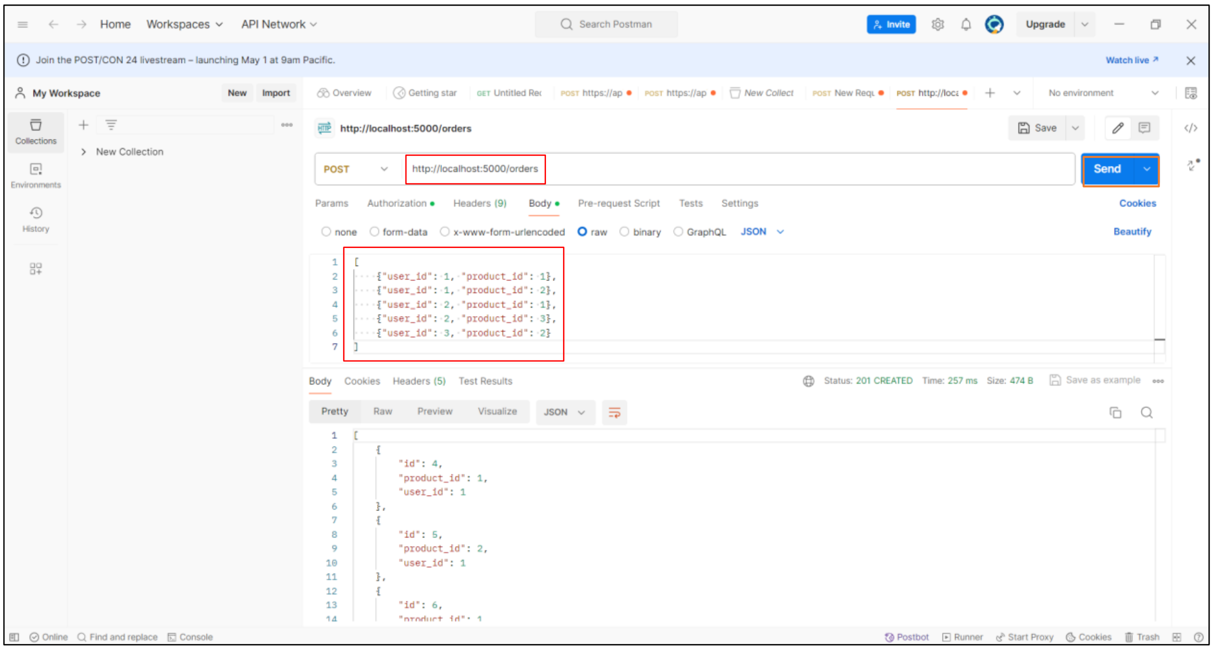
```
[
    {"name": "Product 1"},
    {"name": "Product 2"},
    {"name": "Product 3"},
    {"name": "Product 4"},
    {"name": "Product 5"}
]
```

Status: 201 CREATED Time: 54 ms Size: 404 B

```
[
    {
        "id": 1,
        "name": "Product 1"
    },
    {
        "id": 2,
        "name": "Product 2"
    },
```

2.6 Insert the following records into the order table using Postman POST: **http://localhost:5000/products**

```
[
	{"user_id": 1, "product_id": 1},
	{"user_id": 2, "product_id": 2},
	{"user_id": 3, "product_id": 3},
	{"user_id": 4, "product_id": 4},
	{"user_id": 5, "product_id": 5}
]
```



http://localhost:5000/orders

POST http://localhost:5000/orders

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

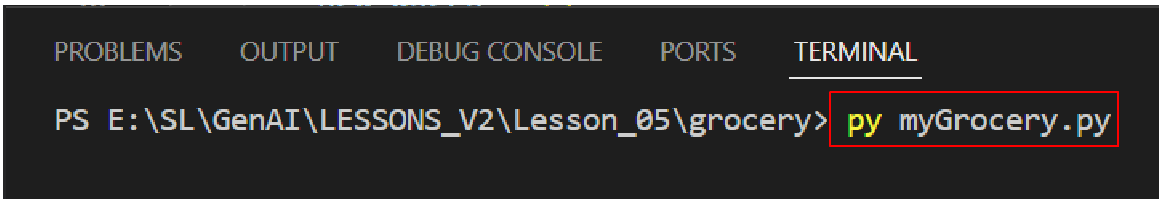
```
[
    {"user_id": 1, "product_id": 1},
    {"user_id": 1, "product_id": 2},
    {"user_id": 2, "product_id": 1},
    {"user_id": 2, "product_id": 3},
    {"user_id": 3, "product_id": 2}
]
```

Status: 201 CREATED Time: 257 ms Size: 474 B

```
[
    {
        "id": 4,
        "product_id": 1,
        "user_id": 1
    },
    {
        "id": 5,
        "product_id": 2,
        "user_id": 1
    },
    {
        "id": 6,
        "product_id": 1
```

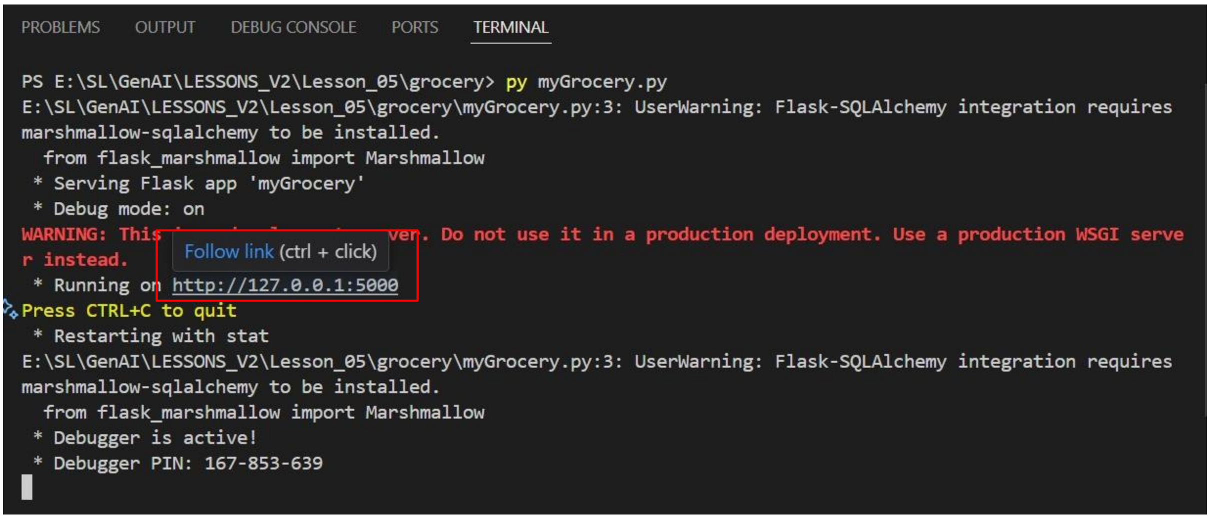
Step 3: Fetch and display MySQL data in the browser

3.1 Navigate to the Visual Studio terminal where the code is running, press **CTRL + C,** and execute the code again using the following command:
py myGrocery.py



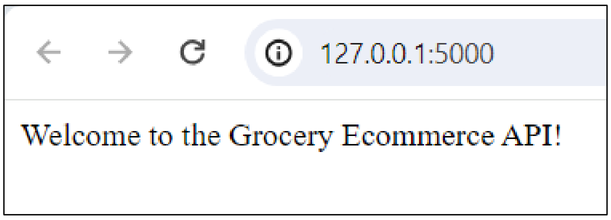
```
PS E:\SL\GenAI\LESSONS_V2\Lesson_05\grocery> py myGrocery.py
```

3.2 Select **http://127.0.0.1:5000** and then click on **Follow link**



```
PS E:\SL\GenAI\LESSONS_V2\Lesson_05\grocery> py myGrocery.py
E:\SL\GenAI\LESSONS_V2\Lesson_05\grocery\myGrocery.py:3: UserWarning: Flask-SQLAlchemy integration requires
marshmallow-sqlalchemy to be installed.
  from flask_marshmallow import Marshmallow
 * Serving Flask app 'myGrocery'
 * Debug mode: on
WARNING: This                        ver. Do not use it in a production deployment. Use a production WSGI serve
r instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
E:\SL\GenAI\LESSONS_V2\Lesson_05\grocery\myGrocery.py:3: UserWarning: Flask-SQLAlchemy integration requires
marshmallow-sqlalchemy to be installed.
  from flask_marshmallow import Marshmallow
 * Debugger is active!
 * Debugger PIN: 167-853-639
```

When the link is clicked, the welcome page appears as follows:

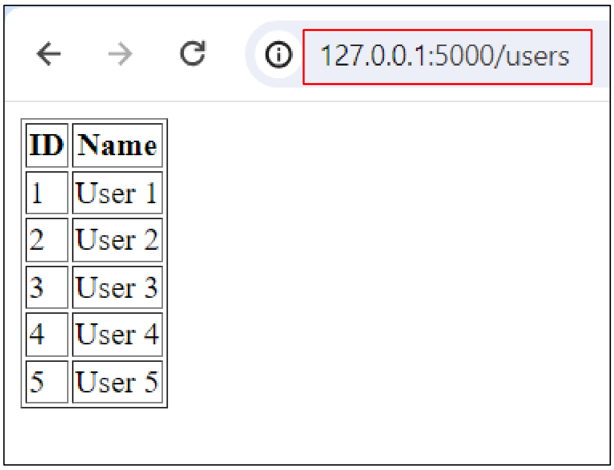


Welcome to the Grocery Ecommerce API!

3.3 Provide table names after the link to fetch the inserted data from the MySQL

The **users** table represents the registered customers or users of the platform. It contains an **ID**, which serves as a unique identifier for each customer, and a user name.

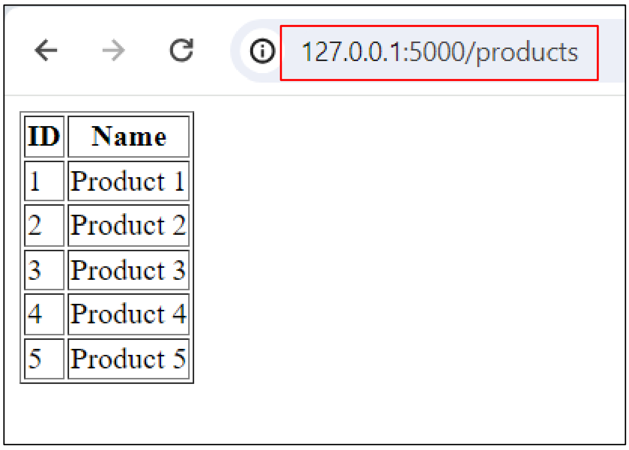
127.0.0.1:5000/users



ID	Name
1	User 1
2	User 2
3	User 3
4	User 4
5	User 5

The **products** table holds information about the various grocery items or products available for sale on the platform. It includes an **ID**, which uniquely identifies each product, and a description of the product.

127.0.0.1:5000/products

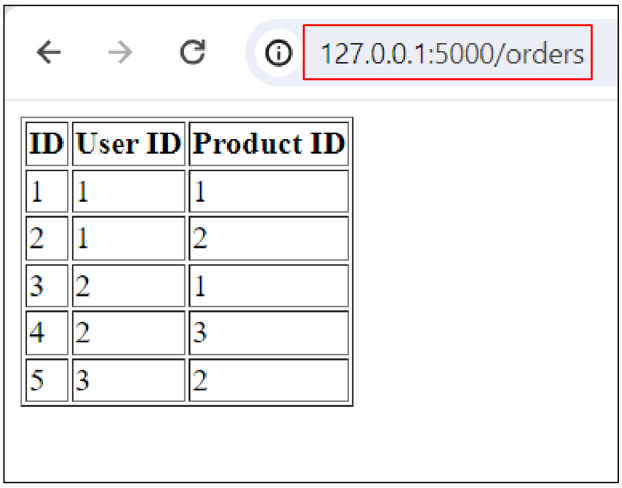


127.0.0.1:5000/products

ID	Name
1	Product 1
2	Product 2
3	Product 3
4	Product 4
5	Product 5

The **orders** table represents the orders placed by customers for specific grocery products. It includes an **ID**, which uniquely identifies each order. The **user_id** column is a foreign key that links an order to the customer who placed it, and the **product_id** column is a foreign key that associates an order with the specific product being ordered.

127.0.0.1:5000/orders



127.0.0.1:5000/orders

ID	User ID	Product ID
1	1	1
2	1	2
3	2	1
4	2	3
5	3	2

By following the outlined steps, you have successfully utilized GitHub Copilot to develop starter code for a grocery e-commerce application, integrating Flask and MySQL to create robust REST APIs. Additionally, you have employed Postman to effectively test and interact with these APIs, demonstrating a practical application of these technologies in building a dynamic e-commerce platform.