

# **SEZG533/CSIZG533**

# **SERVICE ORIENTED**

# **COMPUTING**

Assignment 1

Siva Sankar A

2022MT93006

12-Nov-2023

## Problem Statement: E-Commerce application

E-commerce, short for "electronic commerce," is the buying and selling of goods and services over the internet. It has become a fundamental part of modern business, changing the way companies operate and how customers shop. E-commerce involves conducting financial transactions online. This can include purchasing products, services, or digital goods, as well as making payments for bills, subscriptions, etc.

## PART 1: Design

Basic capabilities of an e-commerce application are essential for facilitating online buying and selling. These capabilities are built around the use cases, which enables users to browse, select a product, purchase, and receive product updates. Few key capabilities are listed below,

- User Registration and Authentication:
  - Allow users to create an account and manage their profiles.
  - Implement secure authentication methods to protect user accounts and data.
- Product Catalog:
  - Maintain a catalog of products.
  - Display product information and price.
- Shopping Cart:
  - Allows users to add or remove items to the cart.
  - Calculate total price, product cost + shipping cost.
- Payment processing:
  - Collect payment and process invoice.
- History profile
  - Maintain history of shopping items and provide relevant preference.
- Product shipment updates:
  - Status of product shipment and delivery tracking.

## Entities and Models:

- User:

- Represents registered users of the e-commerce platform.
- Attributes: “userid”.
- Product:
  - Represents individual product.
  - Attributes: “id”
- Order:
  - Represents an order placed by the user.
  - Attributes:” productid”.
- Payment:
  - Represents payment transaction.
  - Attributes: “paymentid”, “transactionid”
- Shipment
  - Represents shipment details for the shipped product.
  - Attributes: “shipid”.

Few operations that can be performed for each capabilities listed above are,

- User Resource
  - GET /users/{userid}: Retrieve user information by ID.
  - PUT /users/{userid}: Update user information.
  - DELETE /user/{userid}: Delete a user account.
- Product Catalog:
  - GET /products: Retrieve a list of all available products.
  - GET /products/{productid}: Retrieve details about a specific product by ID.
  - POST /products: Add a new product to the catalog.
  - PUT /products/{productid}: Update product details.
  - DELETE /product/{productid}: Remove a product from the catalog.
- Shopping Cart:
  - GET /carts/{userid}: Retrieve the contents of users shopping cart.
  - POST /carts/{userid}/add: Add a product to the user’s shopping cart.
  - POST /carts/{userid}/remove: Remove a product from the user’s shopping cart.

- Payment Processing:
  - GET / payments/methods: Retrieve list of all available payment methods.
  - GET / payments/methods/{paymentid}: Retrieve detailed information about specific payment method.
  - POST /payments/transaction: Initiate a payment transaction for an order.
  - GET /payments/transaction{transactionid}:
- Shipment
  - GET / shipmentdetail/{shipid}: Retrieve details about shipment.
  - GET / tracklocation/{shipid}: Retrieve and track the shipment location.

## URI, HTTP methods and Headers

### *User Resource*

- Retrieve user information by ID:
  - URI: 'GET /users/{userid}'
  - HTTP Method: GET
  - Header: Authorization: Bearer <Token>
- Update user information
  - URI: 'PUT /users/{userid}'
  - HTTP Method: PUT
  - Header: Content-Type: application/json, Authorization: Bearer <Token>
- Delete user account.
  - URI: 'DELETE /users/{userid}'
  - HTTP Method: DELETE
  - Header: Authorization: Bearer <Token>

### *Product Catalog*

- Retrieve all available products.
  - URI: 'GET /products'
  - HTTP Method: GET
  - Header: Authorization: Bearer <Token>

- Retrieve specific product detail by ID
  - URI: 'GET /products/{productid}'
  - HTTP Method: GET
  - Header: Authorization: Bearer <Token>
- Add New product to catalog.
  - URI: 'POST /products'
  - HTTP Method: POST
  - Header: Content-Type: application/json, Authorization: Bearer <Token>
- Update Product details
  - URI: 'PUT /products/{productid}'
  - Header: Content-Type: application/json, Authorization: Bearer <Token>
- Remove a product from catalog.
  - URI: 'DELETE /products/{productid}'
  - HTTP Method: DELETE
  - Header: Authorization: Bearer <Token>

### *Shopping Cart*

- Retrieve the contents of user's shopping cart.
  - URI: 'GET /carts/{userid}'
  - HTTP Method: GET
  - Header: Authorization: Bearer <Token>
- Add a product to shopping cart.
  - URI: 'POST /carts/{userid}/add'
  - HTTP Method: POST
  - Header: Content-Type: application/json, Authorization: Bearer <Token>
- Remove product from shopping cart
  - URI: 'DELETE /carts/{userid}/remove'
  - HTTP Method: DELETE
  - Header: Authorization: Bearer <Token>

Similarly, we could extend it for Payment processing and Shipment.

Design representations accepted by and served to the client.

The representation(s) accepted by the client and the representation(s) served to the client in the RESTful API are typically specified through the content negotiation that allows clients and server to communicate about the types and formats they support for request and response.

Common types include JSON, XML, HTML or even sometimes plain text or stream of bytes.

For the chosen use case e-commerce application Figure 1 shows sample responses to client from server with content negotiation

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 123

{
  "productid": 123,
  "productname": "Product123",
  "productprice": 100
}
```

*Figure 1: Client Response from Server with Content negotiation*

Resource category based on type.

Mostly all the listed API endpoints falls under **core resource category** as they are primary resource in this application and represents main entities that include for example users, products, orders, payment etc

However, if there are related components or sub-entities within the core resources, then we shall categorize them to sub process.

E.g.: We can create sub resource for user profile, preference, and orders, where separate data artefacts shall be used to perform data analytics using machine learning

For special actions which doesn't fit the standard Create, Read, Update and Delete model, we shall create action resource.

E.g.: Action to reset user's account password

## PART 2: Implementation of service end points

Implementation of user registration and authentication for the e-commerce application in Python.

Attached code serviceendpoints.py is implemented with service endpoints using Flask.

```
# User registration endpoint
@app.route('/users/register', methods=['POST'])
def register_user():
    data = request.get_json()
    if 'username' in data and 'password' in data:
        username = data['username']
        password = data['password']

        if username in users_db:
            return jsonify({'message': 'User already exists'}), 400
        else:
            hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
            users_db[username] = hashed_password
            return jsonify({'message': 'User registered successfully'}), 201
    else:
        return jsonify({'message': 'Invalid registration data'}), 400
```

Figure 2: Python def user registration end point

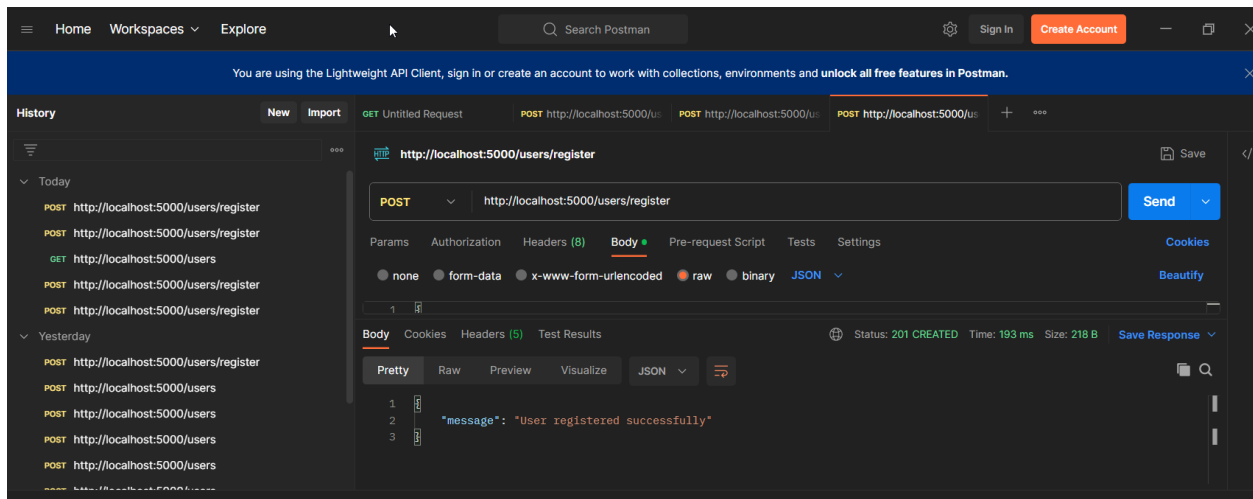


Figure 3: Postman - POST Method for user account registration

Note: REST API shall be also tested with curl (client URL)

```
C:\Users\sau2cob>curl -X POST -H "Content-Type: application/json" -d '{"username": "john", "password": "password123"}' http://localhost:5000/users/register
{"message": "User registered successfully"}
```

Figure 4: Curl command

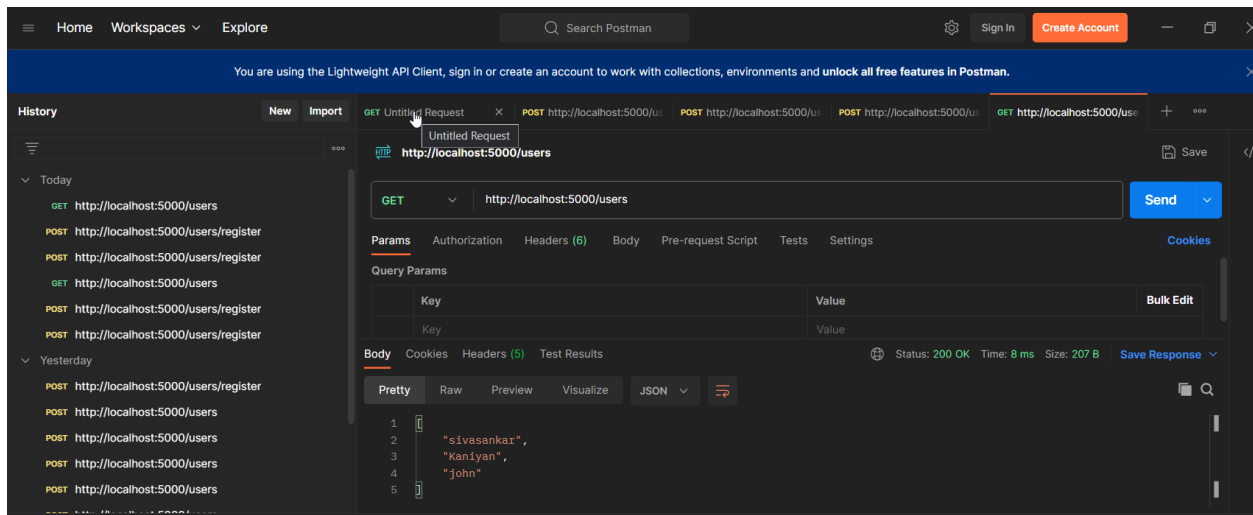


Figure 5: Postman - GET Method to retrieve all registered users.

To authenticate a user for `login_user` below method is implemented in python.

```

# User login endpoint
@app.route('/users/login', methods=['POST'])
def login_user():
    data = request.get_json()
    if 'username' in data and 'password' in data:
        username = data['username']
        password = data['password']

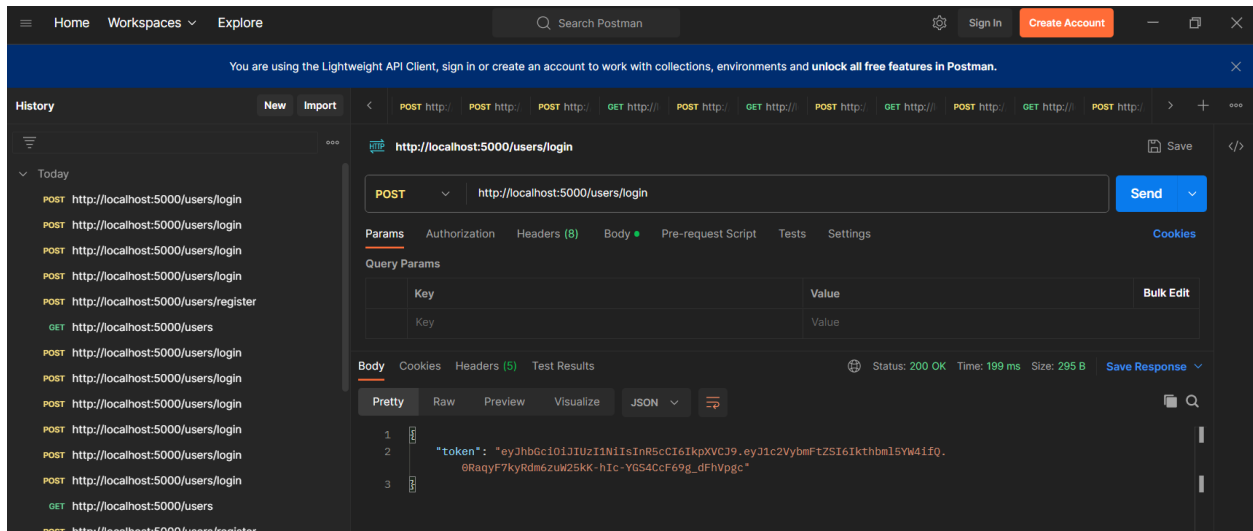
        if username in users_db and bcrypt.checkpw(password.encode('utf-8'), users_db[username]):
            token = jwt.encode({'username': username}, app.config['SECRET_KEY'], algorithm='HS256')
            return jsonify({'token': token})
        else:
            return jsonify({'message': 'Invalid credentials'}), 401
    else:
        return jsonify({'message': 'Invalid login data'}), 400

```

Figure 6: Python def login\_user

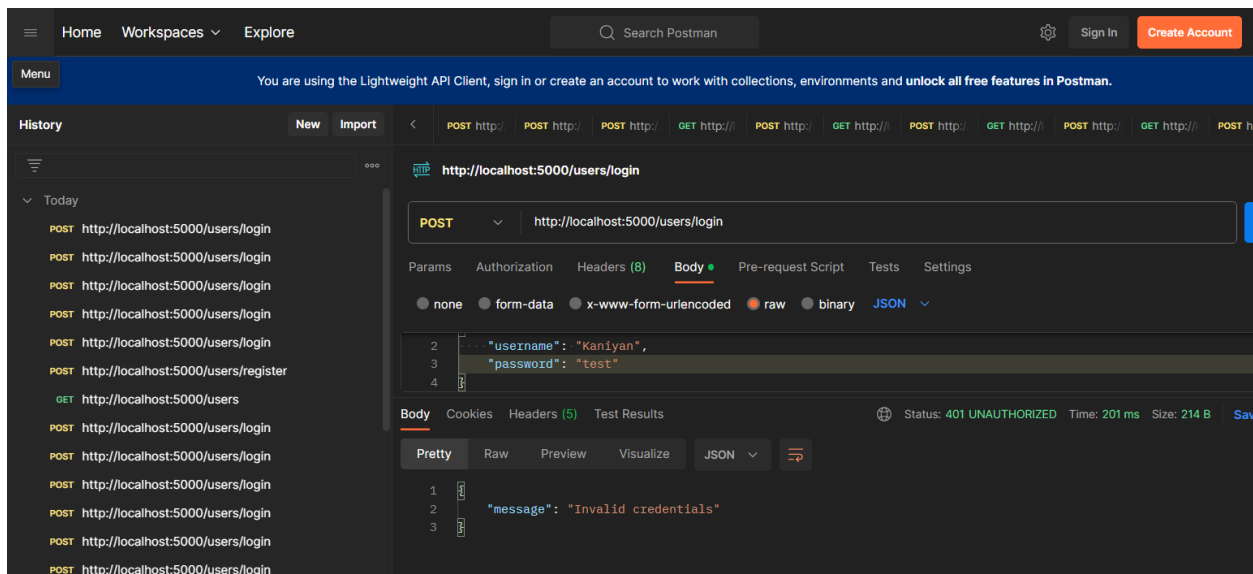


On Post man tool POST method is used to authenticate the user by providing the user name and password in the body. On successful login valid token is issued as response.



*Figure 7: POSTMAN - user login with password*

Below Figure 8 is the response for incorrect password



*Figure 8: Postman- Invalid credentials or incorrect password*

Implementation of service end point for managing product catalog, add, modify, remove product.

To register or add new product to the products db the method is shown in Figure 9

```
# Product registration endpoint
@app.route('/products/register', methods=['POST'])
def register_Product():
    data = request.get_json()
    if 'productid' in data and 'productname' in data:
        productid = data['productid']
        productname = data['productname']
        productprice = data['productprice']
        productdescription = data['productdescription']

        if productid in products_db:
            return jsonify({'message': 'Product already exists'}), 400
        else:
            products_db[productid] = data
            return jsonify({'message': 'Product registered successfully'}), 201
    else:
        return jsonify({'message': 'Invalid registration data'}), 400
```

Figure 9: Python def for product registration

Using Postman, POST method to add new product to products db is shown in Figure 10

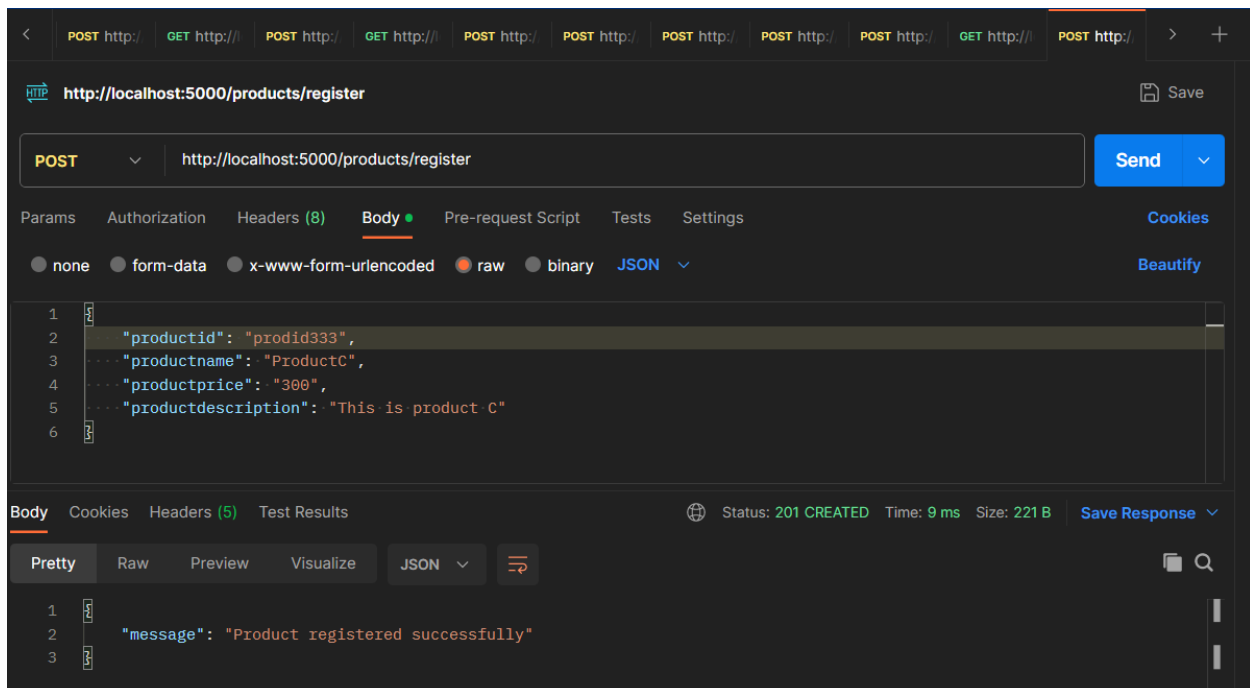
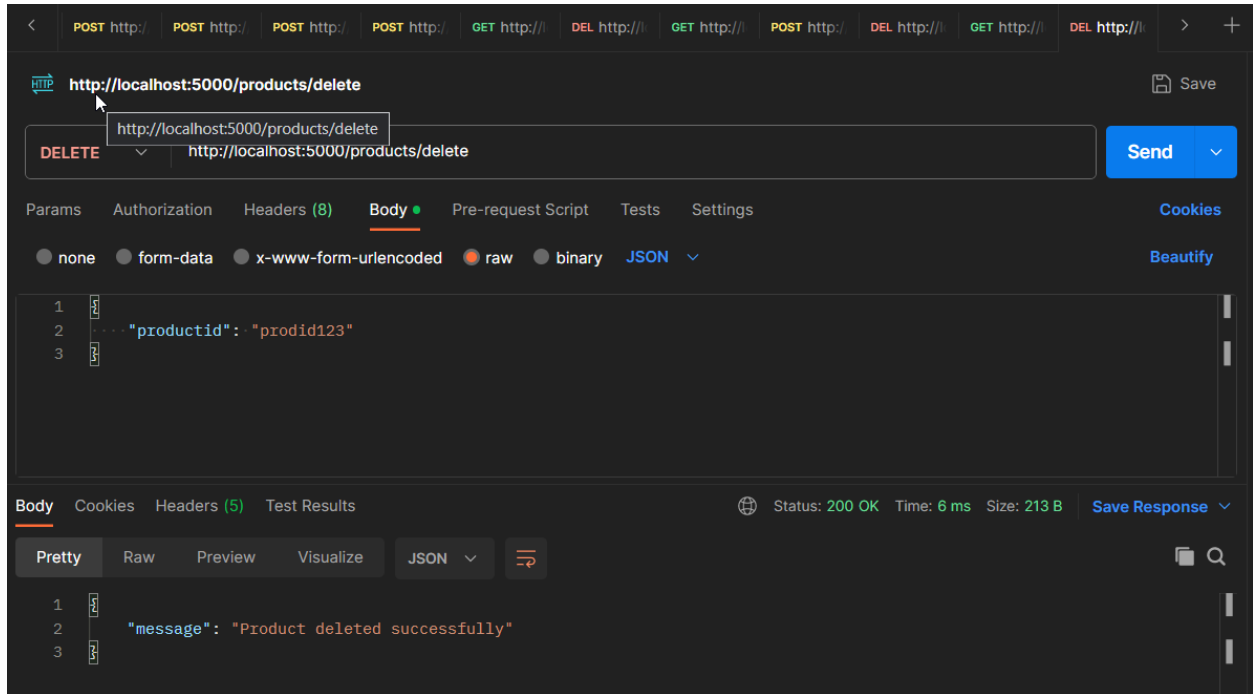


Figure 10: Postman - To register new product.

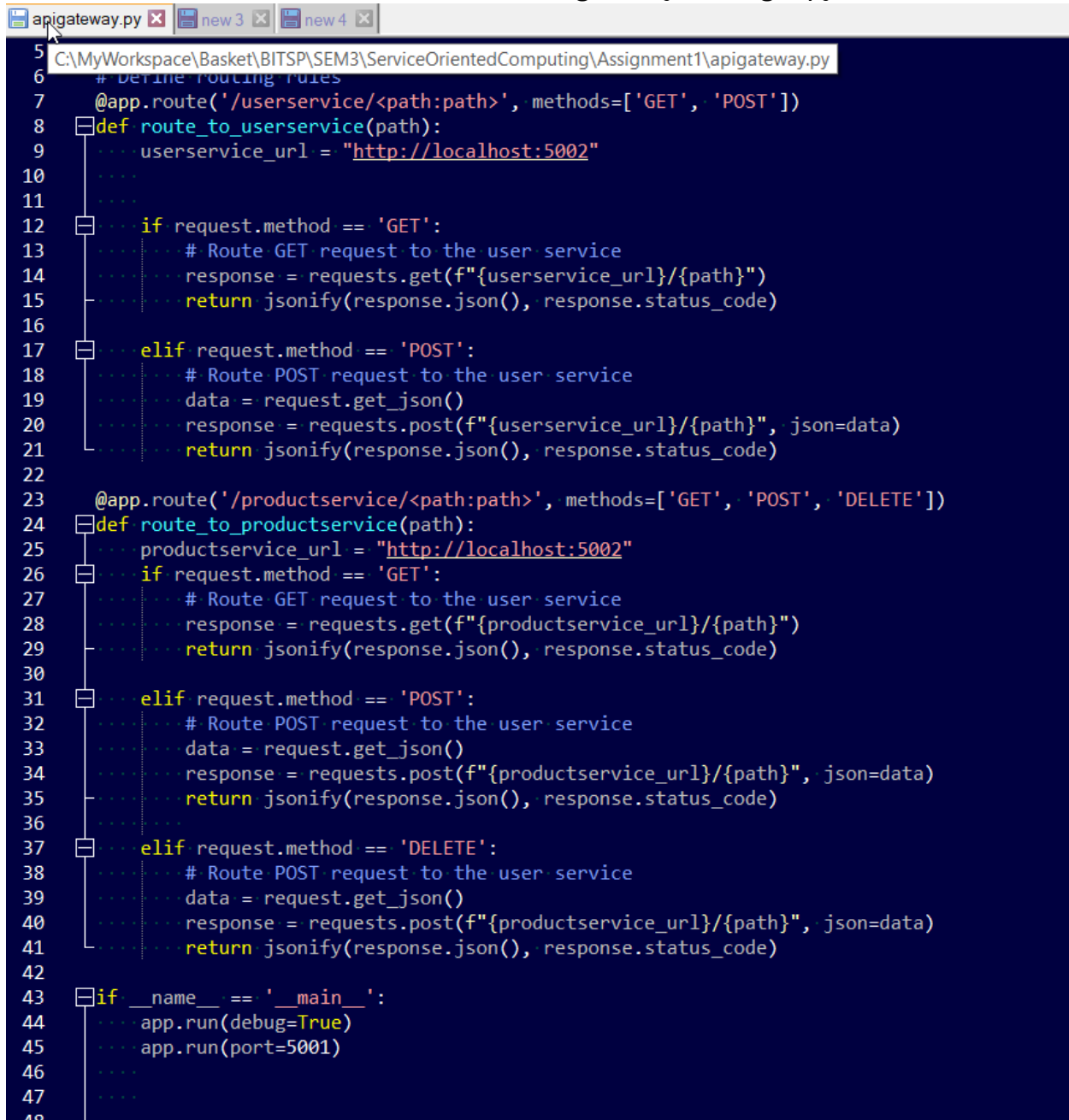
To Delete a product based on the productid the method is shown below Figure 11



*Figure 11: Postman - DELETE a product based on product id.*

## PART 2.2 API Gateway

Designing and developing an API Gateway is a crucial aspect of building a service-oriented application. The API Gateway serves as a single-entry point for all client requests and helps route those requests to the appropriate services. In the below section we see how to build API gateway using python and Flask.



```
5 C:\MyWorkspace\Basket\BITSP\SEM3\ServiceOrientedComputing\Assignment1\apigateway.py
6 # Define Routing Rules
7 @app.route('/userservice/<path:path>', methods=['GET', 'POST'])
8 def route_to_userservice(path):
9     userservice_url = "http://localhost:5002"
10     ...
11     ...
12     if request.method == 'GET':
13         # Route GET request to the user service
14         response = requests.get(f"{userservice_url}/{path}")
15         return jsonify(response.json(), response.status_code)
16     ...
17     elif request.method == 'POST':
18         # Route POST request to the user service
19         data = request.get_json()
20         response = requests.post(f"{userservice_url}/{path}", json=data)
21         return jsonify(response.json(), response.status_code)
22
23 @app.route('/productservice/<path:path>', methods=['GET', 'POST', 'DELETE'])
24 def route_to_productservice(path):
25     productservice_url = "http://localhost:5002"
26     if request.method == 'GET':
27         # Route GET request to the user service
28         response = requests.get(f"{productservice_url}/{path}")
29         return jsonify(response.json(), response.status_code)
30     ...
31     elif request.method == 'POST':
32         # Route POST request to the user service
33         data = request.get_json()
34         response = requests.post(f"{productservice_url}/{path}", json=data)
35         return jsonify(response.json(), response.status_code)
36     ...
37     elif request.method == 'DELETE':
38         # Route POST request to the user service
39         data = request.get_json()
40         response = requests.post(f"{productservice_url}/{path}", json=data)
41         return jsonify(response.json(), response.status_code)
42
43 if __name__ == '__main__':
44     app.run(debug=True)
45     app.run(port=5001)
46     ...
47     ...
48
```

Figure 12: Python apigateway.py with service routing

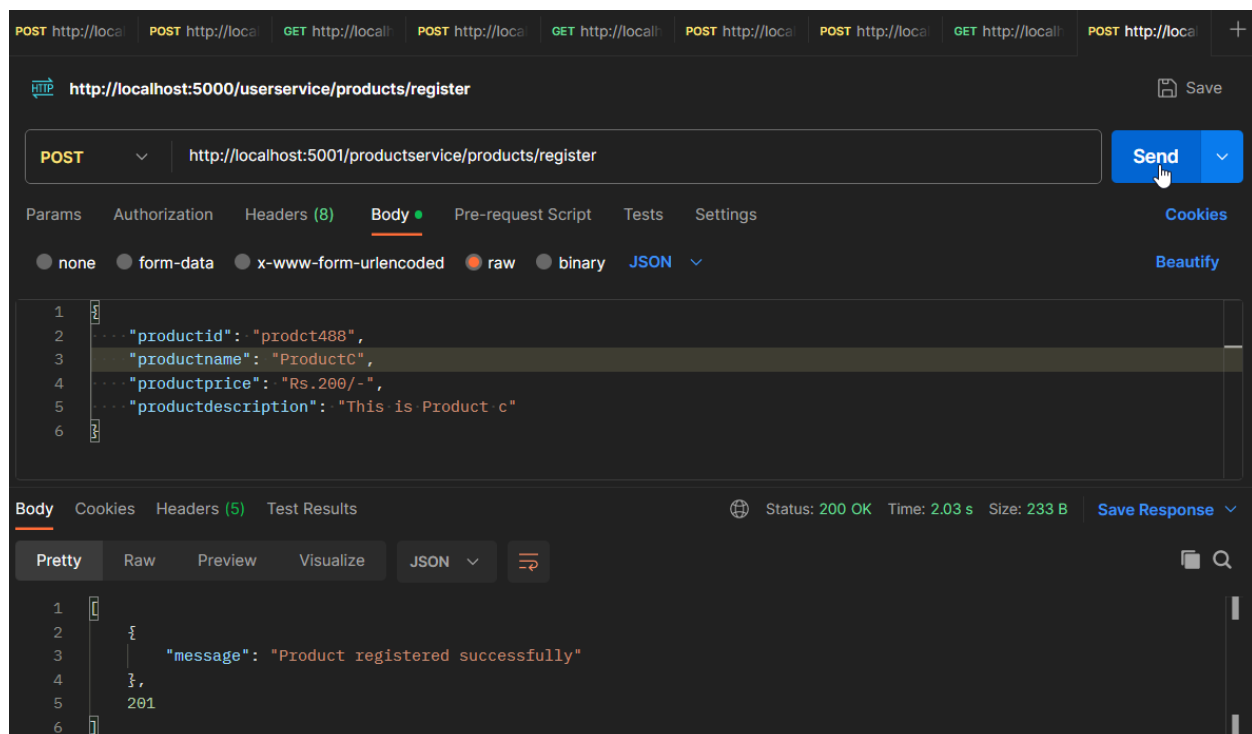
Python code apigateway.py is attached.

Here there are two API gateways created to handle user related service and product related service. These service request from client is routed to the service end points based on the request method.

To run this application, Execution Instruction

- The serviceendpoint.py is started and listens on port 5002.
- This API gateway application is configured with the correct url.
  - Here in this case local host
- Then apigateway.py is started and listens on port 5001, which routes the incoming API requests to the required service.

Below Figure 13 is a test done on postman using API gateway to invoke the APIs



*Figure 13: Postman - API call using API gateway.*

## Request flow diagram

Below Figure 14, shows the Client request flow diagram.

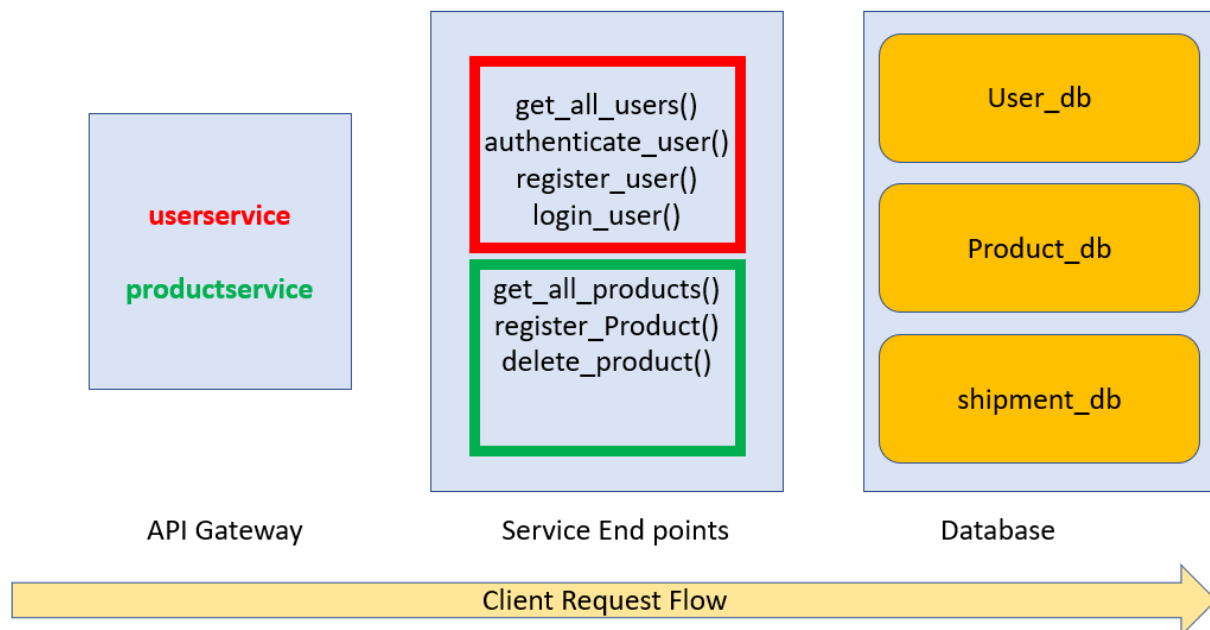


Figure 14: Client Request flow diagram

## Data base Schema

For this application e-commerce we shall list tables like *user*, *product*, *shipment*, *orders* etc. For user table we shall have attributes like *user\_id*, *username*, *password*. There can be relation between *user* table and *orders* table indicating users shall place orders and the records are maintained in the *orders* table. **Error! Reference source not found.**, show sample data base schema with 2 tables and their relationship.

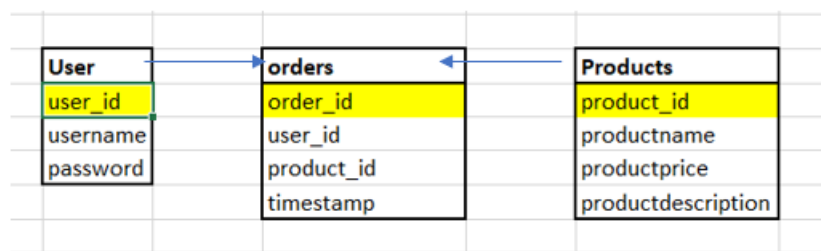


Figure 15: Database schema - table and relation

## Example commands

Figure 16, shows example commands.

```
Example commands

#To Add/register new user
→ http://localhost:5001/userservice/users/register →
→

#To Add/register new product
→ http://localhost:5001/productservice/products/register

JSON format

# product
{
  .... "productid": "prodct488",
  .... "productname": "ProductC",
  .... "productprice": "Rs.200/-",
  .... "productdescription": "This is Product c"
}

# user account
{
  .... "username": "sivasankar",
  .... "password": "siva123"
}
```

*Figure 16: Example Commands*

## Enclosure

- GIT hub – Source code is updated in the below link
  - [https://github.com/sivasankarbitasp/SOC\\_Assignment.git](https://github.com/sivasankarbitasp/SOC_Assignment.git)
- Video demo updated in the below link
  - [https://drive.google.com/file/d/1KdPRhQxVgpEV6Fg\\_KnLc92ABCVAg9uzD/view?usp=drive\\_link](https://drive.google.com/file/d/1KdPRhQxVgpEV6Fg_KnLc92ABCVAg9uzD/view?usp=drive_link)