Lab Manual for Cloud Computing

Lab No. 10

Consuming Micro services

LAB 10: CONSUMING MICRO SERVICES

1. INTRODUCTION:

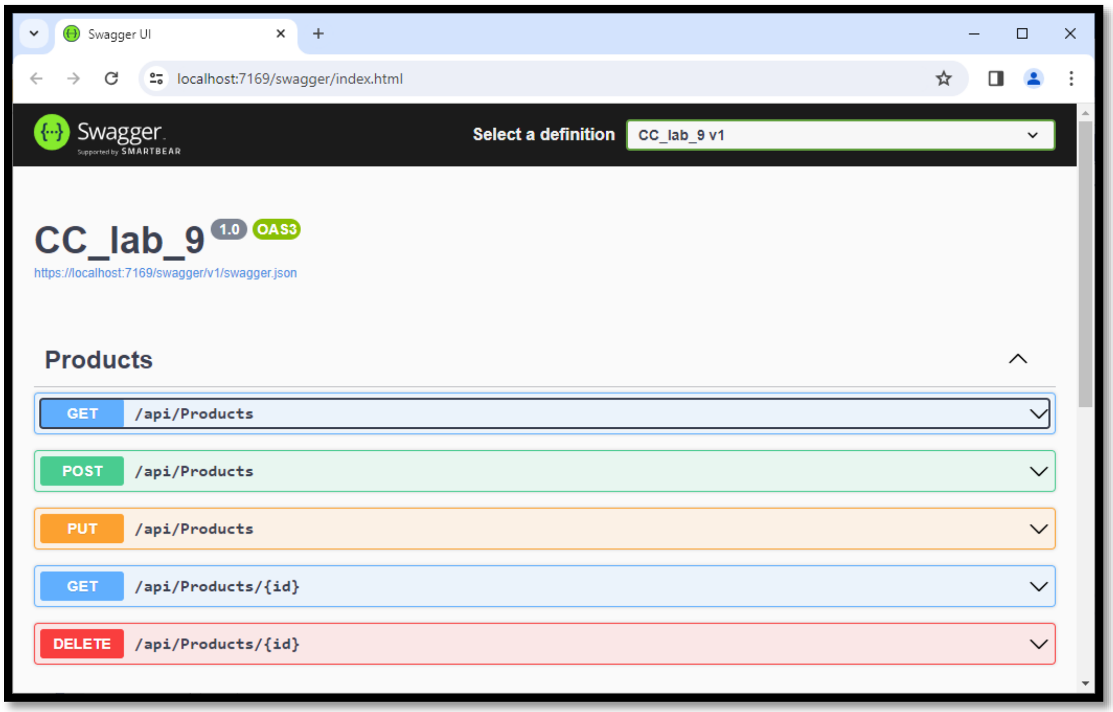
In today's dynamic software landscape, microservices have emerged as a pivotal architectural pattern, facilitating the development of scalable, modular, and maintainable applications. Microservices architecture decomposes complex systems into smaller, loosely coupled services, each responsible for a specific business function. This approach offers numerous advantages, including agility, scalability, and resilience.

Consuming microservices involves interacting with these distributed services to access their functionalities and data. Whether it's retrieving user information, processing payments, or fetching product details, consuming microservices requires seamless communication between different components of the application ecosystem.

In this lab, we'll delve into the process of consuming microservices using ASP.NET Core Web API, Swagger for API documentation and exploration and Postman for API testing. We'll explore how these tools and technologies can streamline the consumption of microservices, empowering developers to build robust and efficient applications.

Consuming Microservices with ASP.NET Core Web API (Swagger):

ASP.NET Core Web API provides a powerful framework for building HTTP-based APIs, making it an ideal choice for implementing microservices. Leveraging ASP.NET Core's flexibility and performance, developers can create lightweight and scalable APIs to expose the functionality of individual microservices. Swagger, also known as OpenAPI, simplifies the process of documenting and exploring APIs. With Swagger, developers can generate interactive API documentation, making it easier for consumers to understand the available endpoints, request/response formats, and authentication mechanisms. Additionally, Swagger's UI allows developers to interactively test API endpoints, facilitating rapid development and debugging.



Consuming Microservices with Postman:

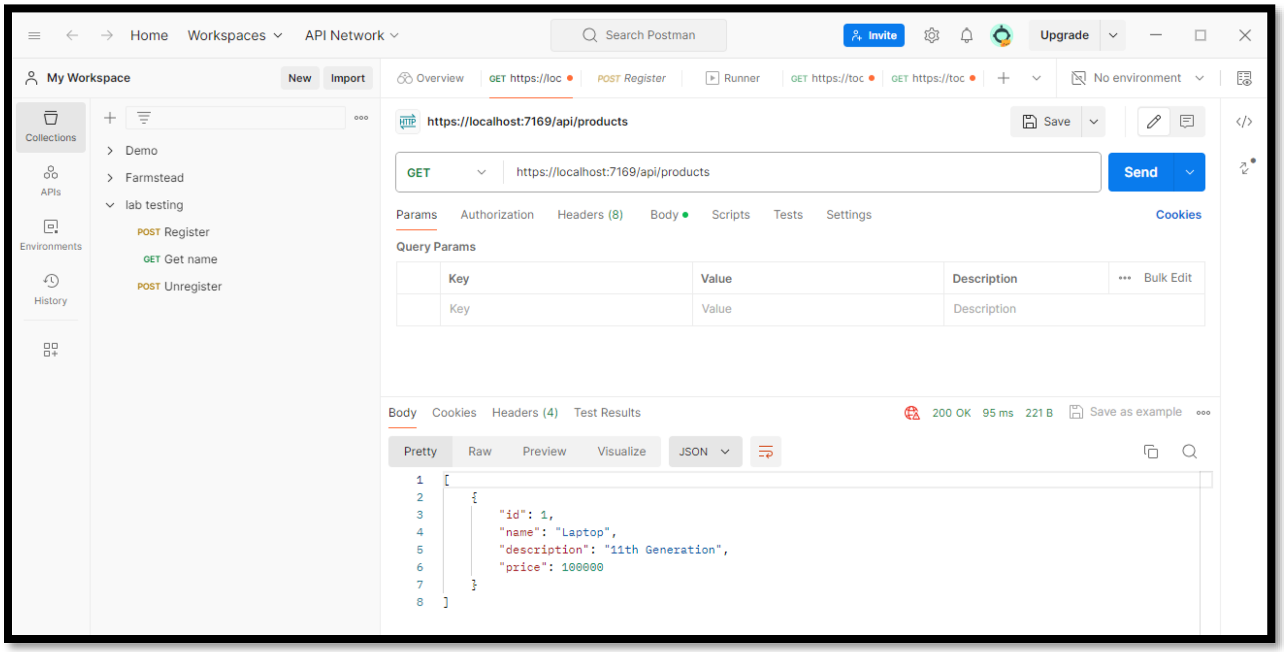
Postman is a popular tool for API testing and collaboration. With its intuitive interface and comprehensive feature set, Postman enables developers to create and execute HTTP requests, automate testing workflows, and share collections of APIs with team members. By leveraging Postman's capabilities, developers can ensure the reliability and performance of microservices during development and deployment.

The desktop postman can be downloaded using the link: https://www.postman.com/downloads/.

Meanwhile the APIs can be tested using the postman like this:

GET:

In Postman, making a GET request involves selecting the GET method from the dropdown menu and entering the URL of the endpoint you want to retrieve data from. Optionally, you can add query parameters or headers if required. Once configured, simply send the request and observe the response displayed below, including status codes, headers, and the response body. GET requests are commonly used to fetch data from servers without modifying them, making them ideal for retrieving information from RESTful APIs or accessing resources from a server.



GET https://localhost:7169/api/products

200 OK 95 ms 221 B

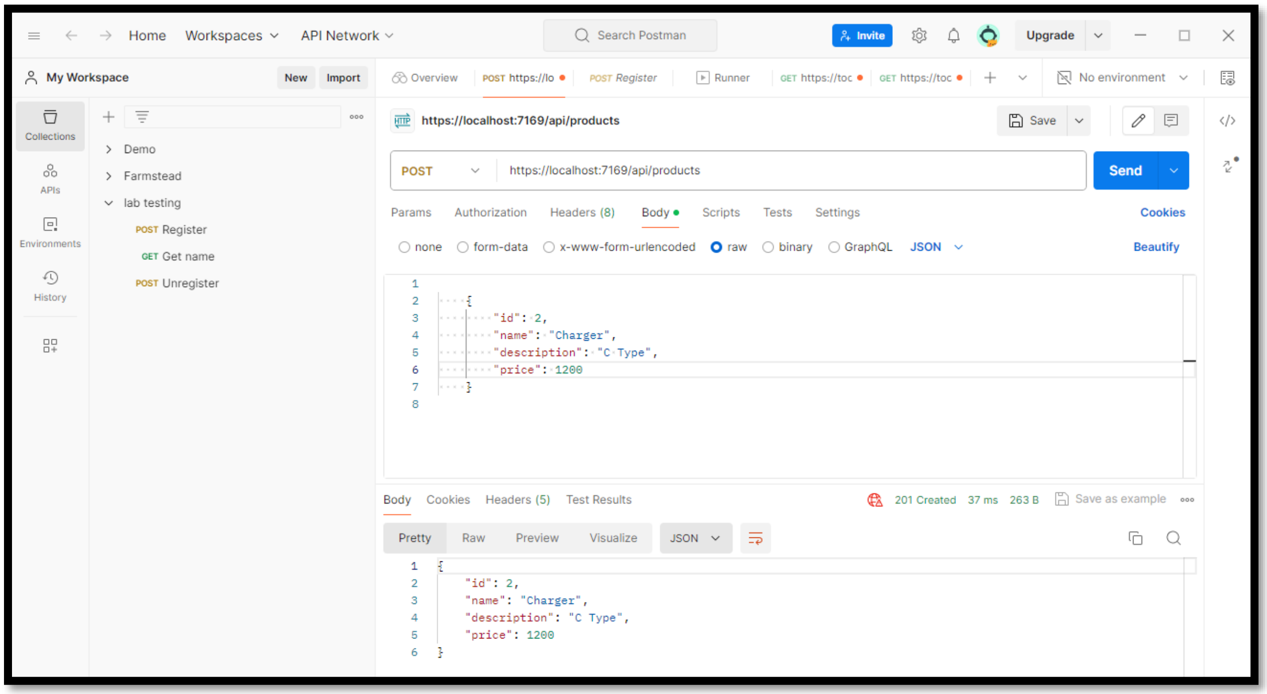
```
[
    {
        "id": 1,
        "name": "Laptop",
        "description": "11th Generation",
        "price": 100000
    }
]
```

POST:

In Postman, executing a POST request involves selecting the "POST" method from the dropdown menu and specifying the URL endpoint where you intend to send data.

Following this, you input the necessary data into the request body using the "Body" tab, utilizing formats such as JSON, form-data, or raw text. Optionally, you can include headers via the "Headers" tab if required by the server.

Once the request is configured, simply click the "Send" button to dispatch it. Postman then presents the response, encompassing status codes, headers, and the response body. POST requests are commonly employed for actions that modify the server's state, such as creating new records or submitting form data.



https://localhost:7169/api/products

POST https://localhost:7169/api/products

Body: none · form-data · x-www-form-urlencoded · [✓ raw] · binary · GraphQL · JSON

```
{
    "id": 2,
    "name": "Charger",
    "description": "C Type",
    "price": 1200
}
```

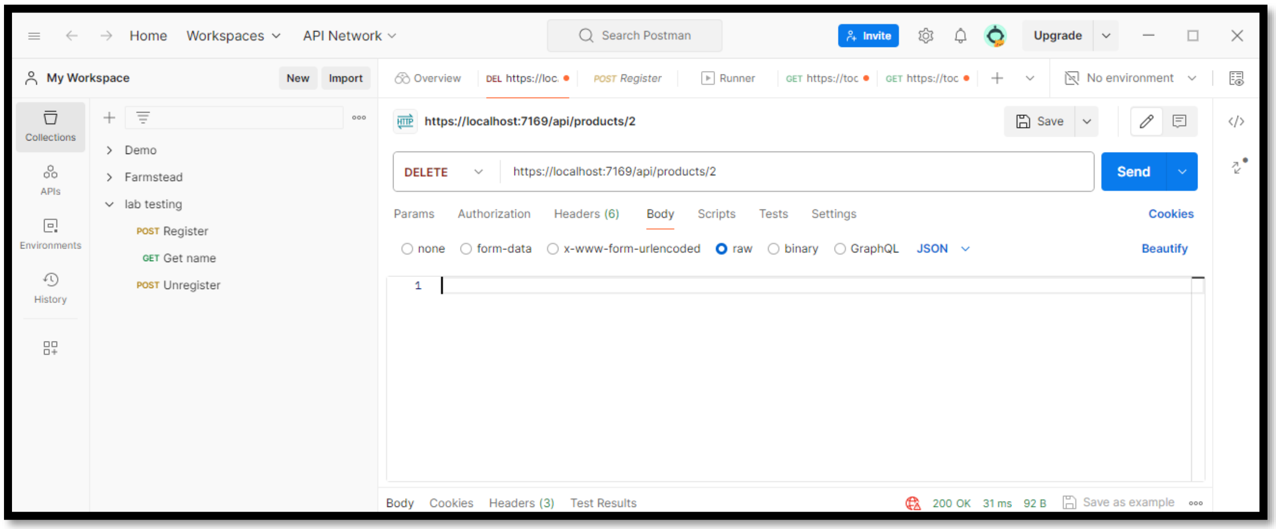
Body · Cookies · Headers (5) · Test Results — 201 Created 37 ms 263 B

```
{
    "id": 2,
    "name": "Charger",
    "description": "C Type",
    "price": 1200
}
```

DELETE:

In Postman, utilizing the DELETE method is straightforward. You begin by selecting "DELETE" from the method dropdown menu in the request pane. Following this, input the URL of the resource you intend to delete. Unlike other methods like POST or PUT, DELETE requests typically do not include a request body, as they are designed to remove resources from the server.

Optionally, you can add headers if necessary. Once the request is configured, simply click the "Send" button to execute it. Postman then presents the response, which includes status codes, headers, and any response body. DELETE requests are commonly used for removing specific records or resources from a server, such as deleting entries from a database or removing files from storage.



https://localhost:7169/api/products/2

DELETE https://localhost:7169/api/products/2

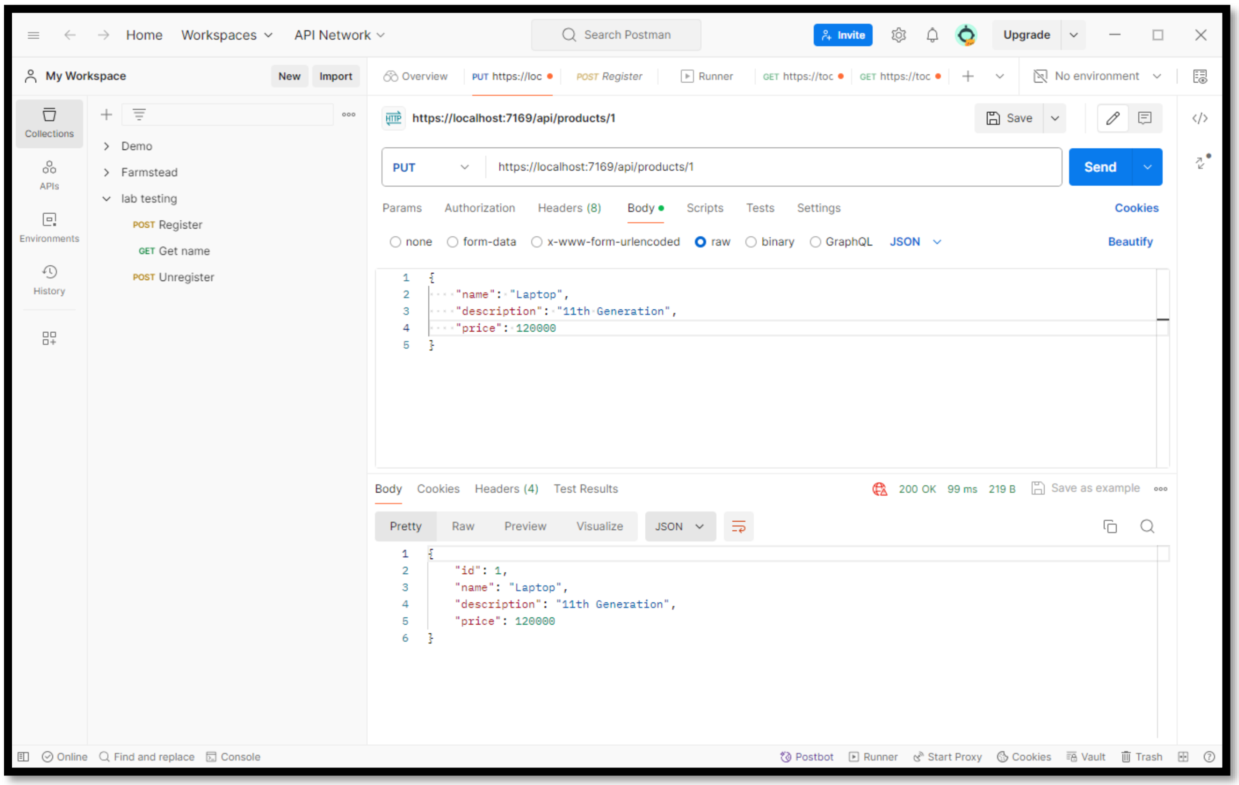
Body: none · form-data · x-www-form-urlencoded · [✓ raw] · binary · GraphQL · JSON

Body · Cookies · Headers (3) · Test Results — 200 OK 31 ms 92 B

PUT:

In Postman, using the PUT method involves selecting "PUT" from the method dropdown menu in the request pane. You then input the URL of the resource you want to update. Unlike a POST request, which typically creates new resources, a PUT request is commonly used to update existing resources.

In the "Body" tab, you can enter the updated data in formats such as JSON, form-data, or raw text. Additionally, you may include headers if necessary via the "Headers" tab. Once the request is configured, click the "Send" button to dispatch it. Postman displays the response, including status codes, headers, and any response body. PUT requests are valuable for modifying existing records or resources on the server, such as updating user information or changing settings.



PUT https://localhost:7169/api/products/1

```
{
    "name": "Laptop",
    "description": "11th Generation",
    "price": 120000
}
```

200 OK 99 ms 219 B

```
{
    "id": 1,
    "name": "Laptop",
    "description": "11th Generation",
    "price": 120000
}
```

2. Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

3. Objectives

After completing this lab the student should be able to:

a. *Recognize the importance of consuming microservices and their essential function in enabling modular and scalable system architectures.*

b. *Develop a microservice consumer application utilizing ASP.NET Core Web API and test its functionality with Postman.*

4. Lab Tasks/Practical Work

1. Implement the microservice for user profile management, incorporating registration, login, and profile updates using ASP.NET Core Identity. After implementation, test the microservice endpoints using Postman to ensure functionality and seamless interaction.