**Ashutosh Dash  
Superset ID – 6365404  
WebAPI**

**OBJECTIVES**

**1 . RESTful Web Service:**

* REST stands for *Representational State Transfer*.
* It's an architecture style for building scalable web services using HTTP.
* It uses standard HTTP verbs like GET, POST, PUT, DELETE.
* It is stateless: Every request from client to server must contain all the information needed.
* Data is typically sent as JSON (but can also be XML or other formats).

**Key Features of REST:**

* **Stateless**: Server does not store client state.
* **Client-Server Architecture**: Client and server work independently.
* **Uniform Interface**: Resource-based access using URLs.
* **Cacheable**: Responses can be cached to improve performance.

**Web API:**

* ASP.NET Web API is a framework to build RESTful services on the .NET platform.
* It is lightweight, open-source, and can return JSON, XML, or other formats.
* It is a part of ASP.NET Core, which supports cross-platform development.

**Microservices:**

* A microservice is a small, independent service that performs one task.
* It can be deployed and scaled independently.
* RESTful Web APIs are typically used for communication between microservices.

**Difference Between WebService and WebAPI:**

| **Feature** | **Web Service (.asmx)** | **Web API (ASP.NET Core)** |
| --- | --- | --- |
| Protocol | SOAP only | HTTP / REST |
| Data Format | XML only | JSON, XML, or any custom format |
| Platform | Windows only (.NET Framework) | Cross-platform (.NET Core) |

**2. Explain What is HttpRequest & HttpResponse**

**HttpRequest:**

* It is the message sent from client to server.
* Contains:
  + Method (GET, POST, etc.)
  + Headers (Content-Type, Auth)
  + Body (data being sent)

**HttpResponse:**

* It is the message sent from server to client.
* Contains:
  + Status Code (200, 400, 500, etc.)
  + Headers
  + Response body (e.g., JSON data)

**3. List the Types of Action Verbs**

These are mapped to HTTP methods and define the purpose of the API action:

| **HTTP Verb** | **C# Attribute** | **Purpose** |
| --- | --- | --- |
| GET | [HttpGet] | Retrieve data |
| POST | [HttpPost] | Insert new record |
| PUT | [HttpPut] | Update entire record |
| DELETE | [HttpDelete] | Remove a record |

In ASP.NET Core Web API, you declare these with attributes above action methods in your controller.

**4. List the Types of HttpStatusCodes Used in WebAPI**

These indicate the result of the HTTP request:

| **Status Code** | **Name** | **Usage** |
| --- | --- | --- |
| 200 | OK | Successful request |
| 400 | BadRequest | Client sent invalid data |
| 401 | Unauthorized | Auth required or failed |
| 500 | InternalServerError | Unexpected server-side failure |

**5. Demonstrate Creation of a Simple Web API – With Read, Write Actions**

**ValuesController.cs**

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace SimpleWebApi.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class ValuesController : ControllerBase

{

private static List<string> values = new List<string> { "Value1", "Value2" };

[HttpGet]

public IActionResult Get()

{

return Ok(values);

}

[HttpPost]

public IActionResult Post([FromBody] string newValue)

{

values.Add(newValue);

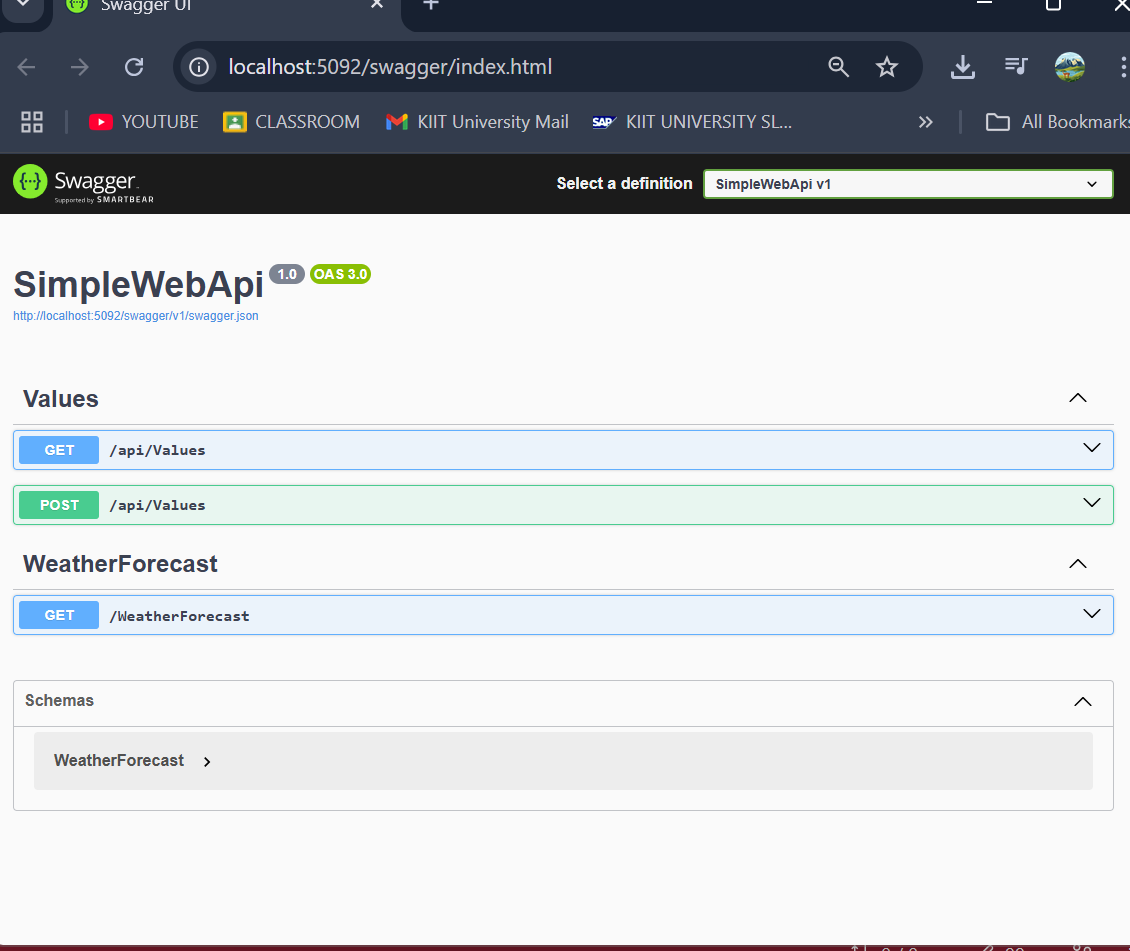
return Ok($"Added: {newValue}");

}

}

}

**OUTPUT**



**6. Explain the types of Configuration files of WebAPI**

In ASP.NET Core Web API, configuration files define how the application behaves. Startup.cs (or Program.cs in .NET 6+) is used to configure services and the request pipeline. appsettings.json stores application settings like connection strings and API keys. launchSettings.json sets local development settings such as the port number and environment. In older .NET Framework (4.5), RouteConfig.cs was used for MVC routing, and WebApiConfig.cs for setting up API routes. These are replaced by simpler configurations in ASP.NET Core.

**QUESTION**

1. **First Web Api using .Net core**

Create a .Net core web application with API template. Use the option to create controller with Read Write permissions. Notice the ValuesController creation with Action methods corresponding to the Action verbs.

On creation of the Web API, execute the application and check if the GET action method result is returned as expected.

**WeatherForecastController.cs**

using Microsoft.AspNetCore.Mvc;

namespace FirstWebApi.Controllers

{

[ApiController]

[Route("[controller]")]

public class WeatherForecastController : ControllerBase

{

private static readonly string[] Summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

private static List<string> \_customValues = new List<string>();

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

[HttpGet(Name = "GetWeatherForecast")]

public IEnumerable<WeatherForecast> Get()

{

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

{

Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

})

.ToArray();

}

[HttpPost]

public IActionResult Post([FromBody] string value)

{

\_customValues.Add(value);

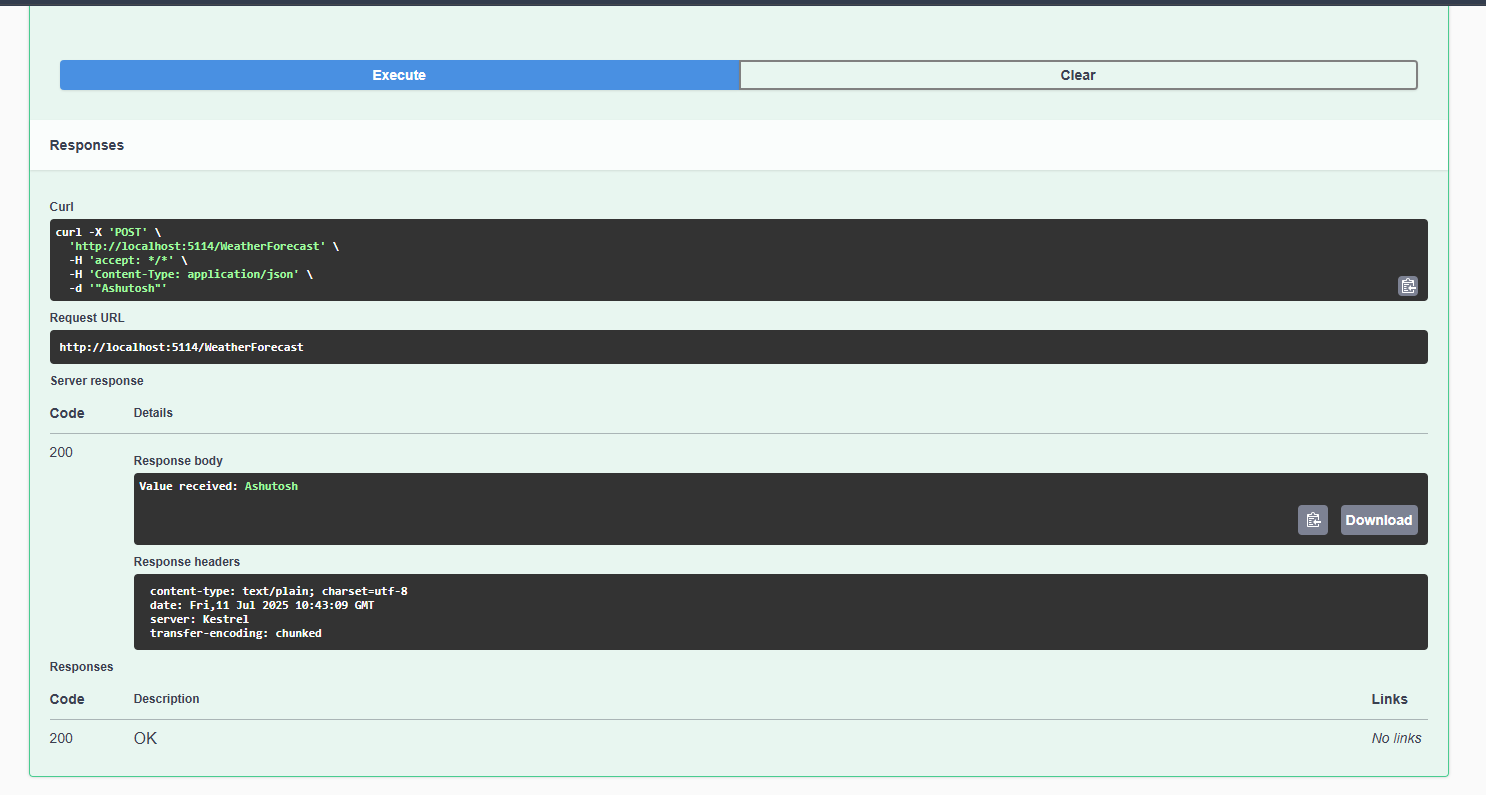
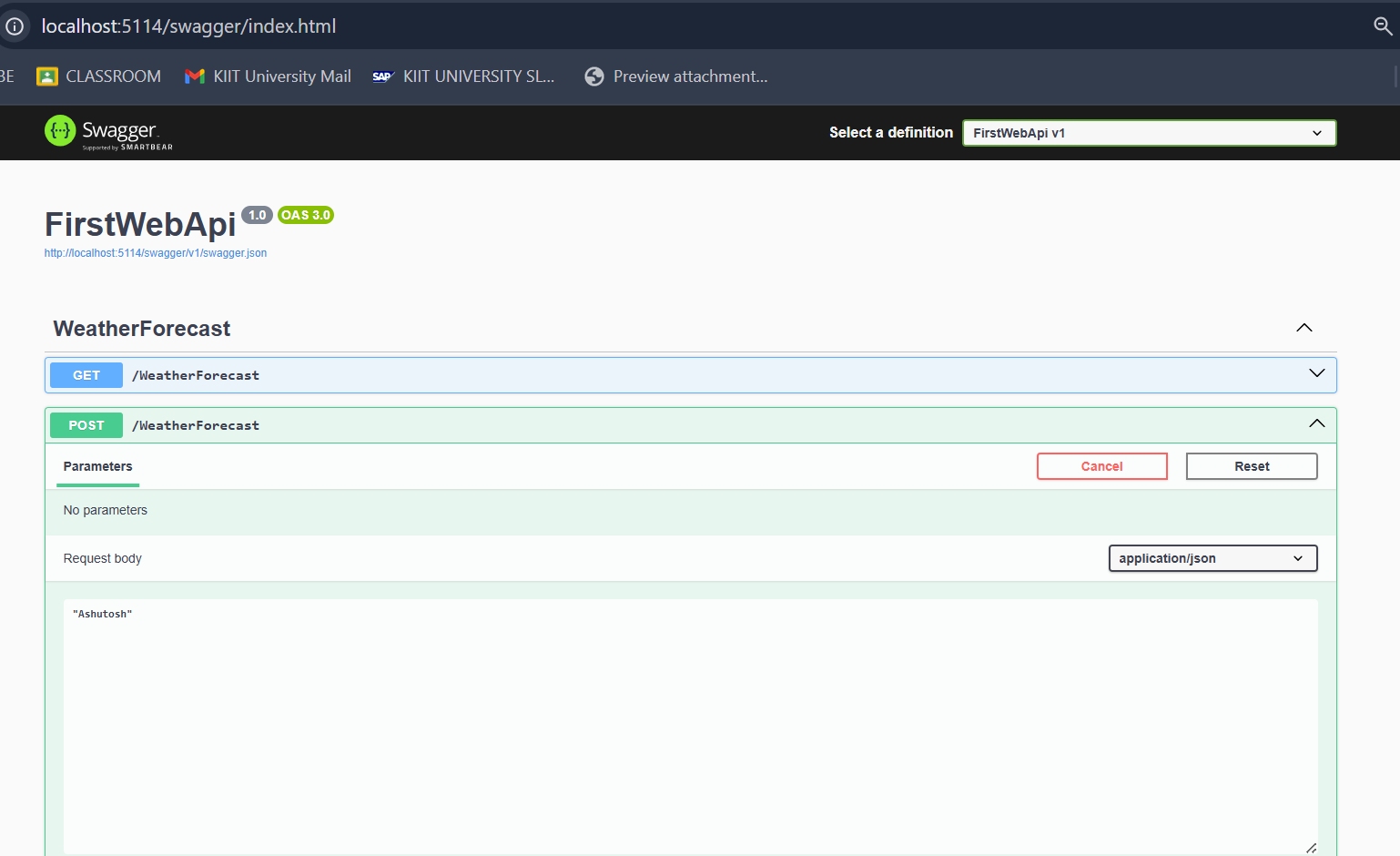
return Ok($"Value received: {value}");

}

}

}

**OUTPUT**

****