**Objectives:**

**1.Explain the concept of RESTful web service, Web API & Microservice**

**Features of REST architecture - Representational State Transfer, Stateless, Messages, Concept of Microservice, Difference between WebService & WebAPI, Not restricted to send XML as response**

Ans:

### RESTful Web Service

REST (Representational State Transfer) is an **architectural style** for designing networked applications. It uses **HTTP protocol** for communication.

#### **Features of REST Architecture:**

#### **Representational State Transfer:** Data is represented in formats like JSON, XML, etc.

* **Stateless:** Each HTTP request is independent. Server does not store session information.
* **Messages:** Communication via standard HTTP methods (GET, POST, PUT, DELETE).
* **Cacheable:** Clients can cache responses.
* **Client-Server:** Separation of concerns between client UI and server backend.

### **Web API**

* **Web API** is a framework that allows you to **expose HTTP services**.
* It supports content negotiation: can return data in **JSON, XML, or any other format**.

### **Microservices**

* A **microservice** is a small, independent service that communicates with others via APIs.
* Each microservice handles a specific business functionality.

| **Feature** | **WebService (.asmx)** | **Web API** |
| --- | --- | --- |
| Protocol | SOAP | HTTP/HTTPS (REST) |
| Data Format | XML only | JSON, XML, etc. |
| Lightweight | No | Yes |
| Platform Dependent | Mostly Windows | Cross-platform |

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

ANS:

**HttpRequest:**

Contains data sent by the client (e.g., browser) to the server.

Example: headers, query strings, route data, body, etc.

**HttpResponse**:

Data sent back by the server to the client.

Example: status code, body, content type, etc.

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

**HttpGet, HttpPost, HttpPut, HttpDelete - Meaning of action verbs and how that should be declared as attributes for Web API**

Ans:

| **Verb** | **Use** | **Example** |
| --- | --- | --- |
| HttpGet | Fetch data | [HttpGet] |
| HttpPost | Submit data | [HttpPost] |
| HttpPut | Update data | [HttpPut] |
| HttpDelete | Remove data | [HttpDelete] |

**4.List the types of HttpStatusCodes used in WebAPI**

**Ok, InternalServerError, Unauthorized, BadRequest - All thru the action result types**

Ans:

## ****Common HTTP Status Codes in Web API****

| **Status** | **Meaning** | **Return Type** |
| --- | --- | --- |
| 200 OK | Request successful | return Ok(data); |
| 400 BadRequest | Invalid client request | return BadRequest("Invalid data"); |
| 401 Unauthorized | Authentication required | return Unauthorized(); |
| 500 InternalServerError | Server error | return StatusCode(500, "Error"); |

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

**Startup.cs with depdency injection, appSettings.json, launchSettings.json, Explain Route.config & WebAPI.config in .Net 4.5 framework**

Ans:

## ****Configuration Files in .NET Core Web API****

| **File** | **Purpose** |
| --- | --- |
| Startup.cs (older versions) or Program.cs (in .NET 6/7) | Application startup logic: Middleware, services, DI |
| appsettings.json | App configurations (e.g., connection strings, logging) |
| launchSettings.json | Development profile & launch config (e.g., port, browser) |
| RouteConfig.cs | Defines routing rules |
| WebApiConfig.cs | Registers routes specific to Web API |
| Web.config | XML file for app settings, DB connections, etc. |

**Q.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.**

**CODE:**

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace MyFirstWebAPI.Controllers

{

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

[ApiController]

public class ValuesController : ControllerBase

{

static List<string> data = new List<string> { "value1", "value2" };

// GET api/values

[HttpGet]

public ActionResult<IEnumerable<string>> Get()

{

return Ok(data); // 200 OK

}

// GET api/values/1

[HttpGet("{id}")]

public ActionResult<string> Get(int id)

{

if (id >= data.Count)

return NotFound(); // 404 Not Found

return Ok(data[id]);

}

// POST api/values

[HttpPost]

public IActionResult Post([FromBody] string value)

{

data.Add(value);

return Ok(data);

}

// PUT api/values/1

[HttpPut("{id}")]

public IActionResult Put(int id, [FromBody] string value)

{

if (id >= data.Count)

return NotFound();

data[id] = value;

return Ok(data);

}

// DELETE api/values/1

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

if (id >= data.Count)

return NotFound();

data.RemoveAt(id);

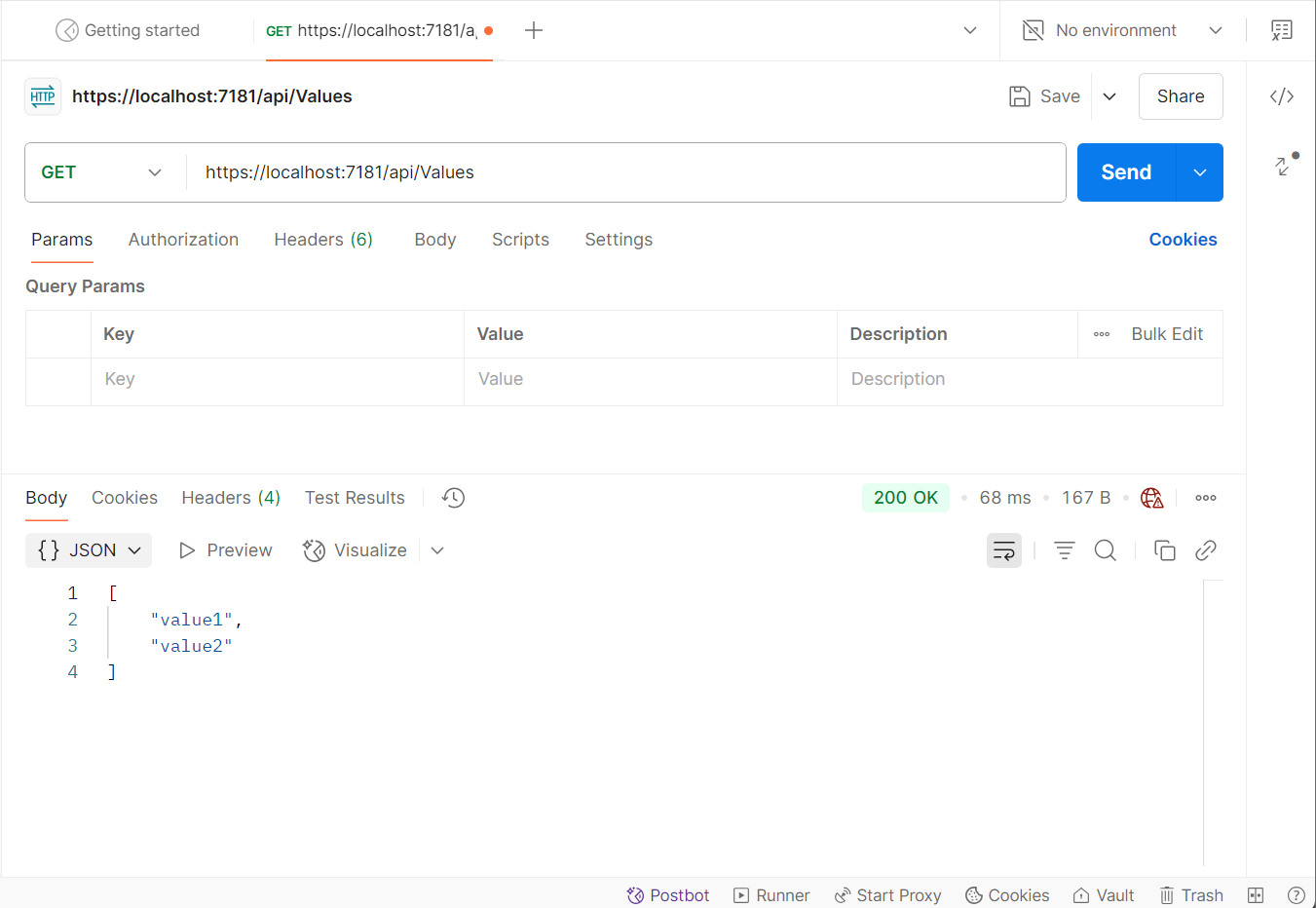
return Ok(data);

}

}

}

**OUTPUT:**

****