* **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**

**Answer-->**

## ****RESTful Web Service****

## A **RESTful web service** is a way for two systems to communicate over HTTP using the principles of REST (Representational State Transfer). It provides interoperability between computer systems on the Internet.

**Example**: When you open a weather app and it shows you the current weather, it likely used a RESTful API to fetch that data from a remote server.

## ****Web API****

A **Web API (Application Programming Interface)** is a set of HTTP-based endpoints that allow external applications to interact with a system. A **RESTful Web API** is one that follows REST principles.

* Web APIs can be RESTful, SOAP-based, or use other architectures.
* RESTful APIs are the most common today due to their simplicity and performance.

## ****Microservice****

A **Microservice** is an architectural style where a large application is broken down into smaller, independent services. Each service handles a specific business function and communicates with other services via APIs (often RESTful).

**Example**: In an e-commerce site:

* Payment Service
* Inventory Service
* Order Service
* Each is a microservice.

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

### ****1.Representational State Transfer (REST)****

* REST is an architectural style, not a protocol.
* It uses **resources** (like users, orders) represented by **URIs** (e.g., /api/users/1).
* Actions on resources are performed using **HTTP method:** GET, POST, PUT, DELETE, etc.

### ****2.Stateless****

* Each request from the client to the server must contain all information needed to understand and process the request.
* The server does **not** store session state between requests.

### ****3.Messages****

* REST uses standard HTTP methods and formats (like **JSON** or **XML**) for communication.
* **Request/Response messages** are simple and human-readable.

### ****4.Not restricted to XML****

* Unlike SOAP, which typically uses XML, RESTful services can respond with:
* **JSON** (most common)
* **XML**
* **HTML**
* **Plain Text**
* **Others**, depending on the Content-Type

## ****Concept of Microservice (Detailed)****

* Microservices are **loosely coupled**, meaning each can be developed and deployed independently.
* Each service has its own **database** and handles its own domain logic.
* Communication is typically through **RESTful APIs** or **message brokers**.

| **Feature** | **Web Service** | **Web API** |
| --- | --- | --- |
| **Definition** | A service available over the web | Interface for communication between systems |
| **Protocol** | Typically uses **SOAP** or **REST** | Uses **HTTP/HTTPS** (REST, etc.) |
| **Data Format** | Mostly **XML** | Supports **JSON**, **XML**, **Text**, etc. |
| **Platform** | Platform-dependent in some cases | Platform-independent |
| **Speed** | Generally slower (SOAP has overhead) | Faster, especially with REST and JSON |
| **Use Case** | Enterprise-level integrations | Web apps, mobile apps, lightweight systems |

* **Explain what is HttpRequest & HttpResponse**

**Answer-->**

## ****HttpRequest****

An **HttpRequest** is a message sent by the **client** (like a web browser or mobile app) to the **server**, asking for a resource or to perform an action.

### ****Key Components of an HttpRequest:****

**1.HTTP Method**:

* Tells the server what action to perform.

**Examples:**

* + - GET – Retrieve data
    - POST – Submit data
    - PUT – Update data
    - DELETE – Remove data

**2.URL (Uniform Resource Locator)**:

* The address of the resource (e.g., /products/123).

**3.Headers**:

* Key-value pairs with extra information.

**Examples:**

* + - Content-Type: application/json
    - Authorization: Bearer <token>

**4.Body (optional)**:

* Data sent with the request (mainly in POST, PUT).
* Often in **JSON**, **XML**, or **form data** format.

## ****HttpResponse****

An **HttpResponse** is the message sent by the **server** back to the **client** in reply to an HttpRequest.

### ****Key Components of an HttpResponse:****

**1.Status Code**:

* A number indicating the result.

**Examples:**

* + - 200 OK – Success
    - 201 Created – Resource created
    - 404 Not Found – Resource not found
    - 500 Internal Server Error – Server error

**2.Headers**:

* Information like response type, cache settings, etc.
* Example: Content-Type: application/json

**3.Body**:

* The actual data sent back.
* Can be HTML, JSON, XML, or plain text.

Example:

{

"id": 123,

"name": "Product A"

}

* **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**

**Answer-->**

| **HTTP Verb** | **Purpose** | **Typical Use Case** |
| --- | --- | --- |
| HttpGet | Retrieve data | Get a list or single item |
| HttpPost | Create a new resource | Submit new data to the server |
| HttpPut | Update an existing resource | Modify all fields of an item |
| HttpDelete | Delete a resource | Remove an item from the server |
| HttpPatch (optional) | Partially update resource | Modify selected fields only |

## ****Meanings of Action Verbs****

### ****[HttpGet]****

* + Used to fetch data.
  + Safe and idempotent (doesn’t change anything on the server).

**Example:**

[HttpGet]

public IActionResult GetAllProducts()

{

// returns list of products

}

### ****[HttpPost]****

* + Used to send data to the server to **create** a new resource.

**Example:**

[HttpPost]

public IActionResult CreateProduct(Product product)

{

// logic to save the product

}

### ****[HttpPut]****

* + Used to **update an existing** resource entirely.

**Example:**

[HttpPut("{id}")]

public IActionResult UpdateProduct(int id, Product product)

{

// logic to update the product

}

### ****[HttpDelete]****

* + Used to **delete** a resource from the server.

**Example:**

[HttpDelete("{id}")]

public IActionResult DeleteProduct(int id)

{

// logic to delete product

}

* **List the types of HttpStatusCodes used in WebAPI**
  + **Ok, InternalServerError, Unauthorized, BadRequest - All thru the action result types**

**Answer-->**

| **Status Code** | **Name** | **Meaning** | **Returned Using** |
| --- | --- | --- | --- |
| 200 | **OK** | Request succeeded and returned data | return Ok(data); |
| 201 | **Created** | New resource created successfully | return Created(uri, data); |
| 400 | **Bad Request** | Invalid request format or parameters | return BadRequest("Invalid input"); |
| 401 | **Unauthorized** | No or invalid authentication credentials | return Unauthorized(); |
| 403 | **Forbidden** | Authenticated but not allowed to access | return Forbid(); |
| 404 | **Not Found** | Resource not found | return NotFound(); |
| 500 | **Internal Server Error** | Server-side error occurred | return StatusCode(500, "Error message"); |

## ****Details of Common ActionResult Types****

### ****200 OK****

Used when everything is successful.

[HttpGet]

public IActionResult GetProduct(int id)

{

var product = db.Products.Find(id);

if (product == null) return NotFound();

return Ok(product); // returns 200 OK with data

}

### ****400 Bad Request****

When input is invalid or malformed.

[HttpPost]public IActionResult CreateProduct(Product product)

{

if (!ModelState.IsValid)

return BadRequest(ModelState); // returns 400

// save product

return Ok(product);

}

### ****401 Unauthorized****

When no valid authentication is provided.

[HttpGet]

public IActionResult GetSecureData()

{

if (!User.Identity.IsAuthenticated)

return Unauthorized(); // returns 401

return Ok("Secure data");

}

### ****500 Internal Server Error****

Used when an unexpected error occurs on the server.

[HttpGet]

public IActionResult GetSomething()

{

try

{

// logic

}

catch (Exception ex)

{

return StatusCode(500, "An error occurred"); // returns 500

}

}

* **Demonstrate creation of a simple WebAPI - With Read, Write actions**
  + **Structure of a web api - Controller & its inheritance from ApiController, Action verbs, Action method**

**Answer-->**

## ****1. Project Structure of a Web API****

MyWebAPI/

│

├── Controllers/

│ └── ProductController.cs <-- Web API Controller

│

├── Models/

│ └── Product.cs <--Data Model

│

├── Program.cs <--App Entry Point

├── Startup.cs (if applicable)<-- Configuration

## ****2. Sample Model – Product.cs****

namespace MyWebAPI.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public double Price { get; set; }

}

}

## ****3. API Controller – ProductController.cs****

using Microsoft.AspNetCore.Mvc;using MyWebAPI.Models;using System.Collections.Generic;using System.Linq;

namespace MyWebAPI.Controllers

{

[ApiController] // Inherited from ApiController

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

public class ProductController : ControllerBase

{

// Simulated in-memory data store

private static List<Product> products = new List<Product>

{

new Product { Id = 1, Name = "Laptop", Price = 50000 },

new Product { Id = 2, Name = "Phone", Price = 25000 }

};

// READ: Get all products

[HttpGet]

public IActionResult GetAll()

{

return Ok(products); // returns 200 OK with product list

}

// READ: Get a single product by ID

[HttpGet("{id}")]

public IActionResult GetById(int id)

{

var product = products.FirstOrDefault(p => p.Id == id);

if (product == null)

return NotFound(); // 404

return Ok(product); // 200

}

// WRITE: Create a new product

[HttpPost]

public IActionResult Create(Product product)

{

product.Id = products.Max(p => p.Id) + 1;

products.Add(product);

return CreatedAtAction(nameof(GetById), new { id = product.Id }, product); // 201

}

}

}

## ****Example API Calls****

### GET → /api/product

Returns all products.

### GET → /api/product/1

Returns product with ID 1.

### POST → /api/product

Send JSON:

{

"name": "Tablet",

"price": 15000}

Returns 201 Created.

* **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**

**Answer-->**

## ****In ASP.NET Core Web API****

### 1. Startup.cs

This is the **main configuration file** for an ASP.NET Core Web API application.

#### Purpose:

* + Set up **services** and **middleware pipeline**
  + Handle **Dependency Injection (DI)**
  + Configure **routing, authentication, CORS, etc.**

#### Structure:

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers(); // Add Web API controllers

services.AddScoped<IProductService, ProductService>(); // Dependency Injection

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers(); // Map API endpoints

});

}

}

### 2. appsettings.json

This file holds **application-level configuration settings** in **JSON format**.

#### Purpose:

Store settings like:

* + Connection strings
  + Logging options
  + API keys
  + Custom configuration sections

#### Example:

#### {

"ConnectionStrings": {

"DefaultConnection": "Server=.;Database=MyDB;Trusted\_Connection=True;"

},

"AppSettings": {

"AppName": "My Web API"

}}

#### Access in Code:

var appName = Configuration["AppSettings:AppName"];

### 3.launchSettings.json

Found in the Properties folder. This file is used **only during development**.

#### Purpose:

* + Configure how the app launches during debugging
  + Set environment variables and ports

#### Example:

#### {

"profiles": {

"MyWebAPI": {

"commandName": "Project",

"launchBrowser": true,

"applicationUrl": "https://localhost:5001;http://localhost:5000",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}}

## ****In ASP.NET Web API (.NET Framework 4.5)****

### 4. Web.config

This is the **main configuration file** in .NET Framework applications.

* Purpose:
  + Define app settings, connection strings, authentication, error handling, etc.

#### Example:

<configuration>

<appSettings>

<add key="AppName" value="MyLegacyWebAPI"/>

</appSettings>

<connectionStrings>

<add name="DefaultConnection" connectionString="..." />

</connectionStrings></configuration>

### 5. RouteConfig.cs

Located in App\_Start. It configures **MVC/Web API routing** for traditional .NET Framework apps.

#### Example (MVC routing):

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

}

}

### 6. WebApiConfig.cs

Also located in App\_Start, specifically for **Web API routing** (not MVC).

#### Example (Web API routing):

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

**Screenshot**

