**Web API Assignment - Questions and Answers**

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

**RESTful Web Service:**

REST (Representational State Transfer) is an architectural style for designing web services that uses standard HTTP methods (GET, POST, PUT, DELETE) to perform operations[[1]](#fn1). Resources are identified by URLs and follow stateless communication principles, meaning each request contains all information needed to process it[[2]](#fn2). RESTful services are designed for HTTP requests and each request must be independent of others to be processed successfully[[2]](#fn2).

**Web API:**

A Web API is an Application Programming Interface that allows different software applications to communicate over the web, enabling data exchange between client and server using HTTP protocol[[3]](#fn3). Web APIs can return data in various formats (JSON, XML) and provide platform-independent communication[[3]](#fn3). Unlike traditional web services, APIs can function online or offline and are not restricted to specific protocols[[3]](#fn3).

**Microservice:**

Microservices represent an architectural approach where applications are built as small, independent services that each handle a specific business function[[1]](#fn1)[[4]](#fn4). Each microservice contains all the code required for a particular application function and can be developed, deployed, and scaled independently[[4]](#fn4). Microservices communicate with each other through APIs and allow for technology diversity, where different services can use different programming languages and frameworks[[1]](#fn1)[[5]](#fn5).

**Question 2: Features of REST architecture**

**Key Features:**

**Representational State Transfer:** Resources are represented in different formats (JSON, XML) and can be accessed through standardized URLs[[1]](#fn1)[[2]](#fn2).

**Stateless:** The server doesn't store client context between requests, making each request independent and containing all necessary information[[2]](#fn2).

**Messages:** Communication occurs through HTTP messages with headers and body content[[3]](#fn3).

**Uniform Interface:** Provides a consistent way to interact with resources using standard HTTP methods[[3]](#fn3).

**Not restricted to XML:** Can send responses in JSON, XML, or other formats, unlike SOAP which is limited to XML[[3]](#fn3).

**Concept of Microservice:**

Microservices are small, focused services that perform specific business functions independently[[6]](#fn6). They enable independent deployment and scaling, allowing different services to use different technologies and programming languages[[1]](#fn1)[[5]](#fn5).

**Difference between WebService & WebAPI:**

|  |  |  |
| --- | --- | --- |
| Aspect | WebService | WebAPI |
| Protocol | SOAP-based, XML-only | RESTful, HTTP-based |
| Complexity | More complex | Lightweight |
| Data Format | Restricted to XML | Supports multiple formats (JSON, XML) |
| Network Requirement | Requires network | Can work offline or online |

**Question 3: Explain HttpRequest & HttpResponse**

**HttpRequest:**

An HttpRequest is a message sent from client to server that contains the HTTP method, URL, headers, and body (for POST/PUT operations)[[3]](#fn3). For example, a GET request retrieves data from the server without sending a body, while a POST request includes data in the request body to create new resources.

**HttpResponse:**

An HttpResponse is a message sent from server back to client that contains a status code, headers, and response body[[3]](#fn3). For example, a successful request returns "200 OK" with JSON data, while an error might return "404 Not Found" or "500 Internal Server Error".

**Question 4: Types of Action Verbs**

**HTTP Action Verbs:**

**GET:** Retrieve data from server without modifying anything  
**POST:** Create new resource on the server  
**PUT:** Update existing resource completely (full replacement)  
**DELETE:** Remove resource from the server  
**PATCH:** Partial update of existing resource

**Declaration as Attributes:**

[HttpGet]  
public IActionResult GetData()   
{  
 // Retrieve data logic  
}  
  
[HttpPost]  
public IActionResult CreateData([FromBody] DataModel data)   
{  
 // Create new data logic  
}  
  
[HttpPut("{id}")]  
public IActionResult UpdateData(int id, [FromBody] DataModel data)   
{  
 // Update existing data logic  
}  
  
[HttpDelete("{id}")]  
public IActionResult DeleteData(int id)   
{  
 // Delete data logic  
}

**Question 5: Types of HttpStatusCodes used in WebAPI**

**Common Status Codes:**

**200 OK:** Successful request with data returned  
**201 Created:** Resource successfully created  
**400 BadRequest:** Invalid request data or malformed request  
**401 Unauthorized:** Authentication required or failed  
**404 NotFound:** Requested resource doesn't exist  
**500 InternalServerError:** Server encountered an error

**Action Result Types:**

return Ok(data); // 200  
return Created("location", newData); // 201  
return BadRequest("Invalid data"); // 400  
return Unauthorized(); // 401  
return NotFound(); // 404  
return StatusCode(500, "Server Error"); // 500

**Question 6: Structure of a Web API**

**Components:**

**Controller:** Inherits from ControllerBase or ApiController and contains action methods that handle HTTP requests.

**Action Methods:** Individual methods that handle specific HTTP requests and return appropriate responses.

**Action Verbs:** Attributes that specify which HTTP method each action responds to.

**Routing:** Maps URLs to controller actions using route templates.

**Example Structure:**

[ApiController]  
[Route("api/[controller]")]  
public class ValuesController : ControllerBase  
{  
 [HttpGet]  
 public IActionResult Get()   
 {  
 // Return all values  
 return Ok(values);  
 }  
   
 [HttpGet("{id}")]  
 public IActionResult Get(int id)   
 {  
 // Return specific value  
 return Ok(values[id]);  
 }  
   
 [HttpPost]  
 public IActionResult Post([FromBody] string value)   
 {  
 // Create new value  
 return CreatedAtAction(nameof(Get), new { id = newId }, value);  
 }  
}

**Question 7: Types of Configuration Files**

**.NET Core Configuration:**

**Program.cs:** Entry point that configures and starts the web application, sets up dependency injection, and configures middleware pipeline.

**appsettings.json:** Contains application settings, connection strings, logging configuration, and environment-specific settings.

**launchSettings.json:** Development environment settings including port numbers, SSL settings, and environment variables.

**.NET Framework 4.5:**

**RouteConfig.cs:** Defines routing rules that map URLs to controller actions.

**WebApiConfig.cs:** Web API specific configurations including formatters, filters, and routing conventions.