# Web API Assignment (WEEK 4)

## 1. Explain the concept of RESTful Web Service, Web API & Microservice

RESTful Web Service: An architecture style using standard HTTP methods. It emphasizes stateless communication, resource identification through URIs, and the use of standard verbs (GET, POST, PUT, DELETE).

Web API: A framework that allows access to services via HTTP. ASP.NET Web API enables creation of HTTP services.

Microservice: A small, independently deployable service, built around specific business capabilities and running in its own process.

## 2. Features of REST Architecture

Stateless

Uniform Interface

Cacheable

Layered System

Code on Demand (optional)

Client-Server architecture

Supports JSON and XML (not limited to XML)

Difference: WebService vs WebAPI:

- WebService is SOAP-based and returns XML

- WebAPI is RESTful and supports multiple formats

## 3. Explain HttpRequest & HttpResponse

HttpRequest: Represents the incoming request, contains data like headers, body, method, URL.

HttpResponse: Sent from server to client, includes status code, headers, and content.

## 4. List and Explain Action Verbs

HttpGet: Fetch data

HttpPost: Create data

HttpPut: Update data

HttpDelete: Remove data

Declared using attributes: [HttpGet], [HttpPost] etc.

## 5. Http Status Codes

200 OK

400 BadRequest

401 Unauthorized

500 InternalServerError

## 6. Create a Simple Web API (Read, Write)

[ApiController]

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

public class ValuesController : ControllerBase

{

[HttpGet]

public IActionResult Get() => Ok(new string[] { "value1", "value2" });

[HttpPost]

public IActionResult Post([FromBody] string value) => Ok("Value Received: " + value);

}

## 7. Configuration Files in Web API

Startup.cs: Contains ConfigureServices and Configure method for DI, Middleware

appsettings.json: App-specific settings

launchSettings.json: Controls environment config for launch

Route.config / WebAPI.config (in .NET Framework)

## 8. Add Swagger to Web API

// In ConfigureServices

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new() {

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

Contact = new() { Name = "John Doe", Email = "john@xyzmail.com" },

License = new() { Name = "License Terms" }

});

});

// In Configure

app.UseSwagger();

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

});

## 9. Postman Tool Demonstration

Set method type (GET, POST, PUT, DELETE)

Use Authorization header for tokens

Body tab for sending JSON data

## 10. Routing with ActionName & Name Attributes

[HttpGet("getEmp"), ActionName("GetEmployee")]

public IActionResult GetEmployee() => Ok();

## 11. Employee Model Creation

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public int Salary { get; set; }

public bool Permanent { get; set; }

public Department Department { get; set; }

public List<Skill> Skills { get; set; }

public DateTime DateOfBirth { get; set; }

}

## 12. EmployeeController with Read/Write

[ApiController]

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

public class EmployeeController : ControllerBase

{

private List<Employee> employees = new();

public EmployeeController()

{

employees = GetStandardEmployeeList();

}

private List<Employee> GetStandardEmployeeList() => new() {

new Employee { Id = 1, Name = "John", Salary = 5000 }

};

[HttpGet]

[ProducesResponseType(200)]

public ActionResult<List<Employee>> GetStandard() => Ok(employees);

[HttpPost]

public IActionResult Post([FromBody] Employee emp) => Ok(emp);

}

## 13. Custom Authorization Filter

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

var headers = context.HttpContext.Request.Headers;

if (!headers.ContainsKey("Authorization"))

context.Result = new BadRequestObjectResult("Invalid request - No Auth token");

else if (!headers["Authorization"].ToString().Contains("Bearer"))

context.Result = new BadRequestObjectResult("Invalid request - Token present but Bearer unavailable");

}

}

## 14. Custom Exception Filter

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

File.WriteAllText("error.txt", context.Exception.ToString());

context.Result = new ObjectResult("An error occurred") { StatusCode = 500 };

}

}

## 15. Web API CRUD Operations - PUT

[HttpPut("update")]

public IActionResult UpdateEmployee([FromBody] Employee emp)

{

if (emp.Id <= 0)

return BadRequest("Invalid employee id");

var existing = employees.FirstOrDefault(e => e.Id == emp.Id);

if (existing == null)

return BadRequest("Invalid employee id");

existing.Name = emp.Name;

existing.Salary = emp.Salary;

return Ok(existing);

}

## 16. Enable CORS

// In ConfigureServices

services.AddCors(options =>

{

options.AddDefaultPolicy(policy => policy.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

});

// In Configure

app.UseCors();

## 17. JWT Authentication - Startup.cs + Token Generation

// Startup.cs

var key = Encoding.UTF8.GetBytes("mysuperdupersecret");

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = new SymmetricSecurityKey(key)

};

});

app.UseAuthentication();

// Token Generator

private string GenerateJSONWebToken(int userId, string role)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret"));

var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

var claims = new List<Claim>

{

new Claim(ClaimTypes.Role, role),

new Claim("UserId", userId.ToString())

};

var token = new JwtSecurityToken("mySystem", "myUsers", claims, expires: DateTime.Now.AddMinutes(10), signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}