1. **WebApi\_Handson**

**Objectives:**

* 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
* Explain what is HttpRequest & HttpResponse
* 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
* List the types of HttpStatusCodes used in WebAPI
  + Ok, InternalServerError, Unauthorized, BadRequest - All thru the action result types
* Demonstrate creation of a simple WebAPI - With Read, Write actions
  + Structure of a web api - Controller & its inheritance from ApiController, Action verbs, Action method
* 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

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.

### **Answers:**

#### **Objectives:**

* RESTful web services are stateless architectural styles for APIs using HTTP.
* Web API is a framework for building HTTP services in .NET. It allows creation of RESTful services.
* Microservices are small, independent services that work together. They improve scalability and deployment.
* Features of REST:
  + Stateless: Server doesn’t store client context.
  + Messages: Communication via HTTP.
  + Supports multiple formats like XML, JSON.
  + Lightweight.
  + Each resource is accessible via URI.
* Web API vs WebService:
  + WebService uses SOAP and returns XML.
  + Web API is light and returns JSON/XML.
  + WebService is slower, Web API is faster and flexible.
* HttpRequest: Represents the incoming request.
* HttpResponse: Represents the response returned.
* Action Verbs:
  + HttpGet: Fetch data.
  + HttpPost: Create data.
  + HttpPut: Update data.
  + HttpDelete: Delete data.
  + Decorated using [HttpGet], [HttpPost] etc.
* HttpStatusCodes:
  + Ok (200): Successful.
  + BadRequest (400): Invalid request.
  + Unauthorized (401): No authorization.
  + InternalServerError (500): Server error.
* Simple Web API:
  + Controller inherits from ApiController.
  + Action methods decorated with HTTP verbs.
  + Handles requests and returns responses.
* Configuration Files:
  + Startup.cs: ConfigureServices and Configure methods, DI setup.
  + appsettings.json: Configuration values.
  + launchSettings.json: Profiles for launching the app.
  + Route.config, WebAPI.config: Used in .NET Framework (not Core) for routing and config.

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

using Microsoft.AspNetCore.Mvc;  
using System.Collections.Generic;  
  
namespace WebApiDemo.Controllers  
{  
 [Route("api/[controller]")]  
 [ApiController]  
 public class ValuesController : ControllerBase  
 {  
 [HttpGet]  
 public IEnumerable<string> Get()  
 {  
 return new string[] { "value1", "value2" };  
 }  
  
 [HttpGet("{id}")]  
 public string Get(int id)  
 {  
 return "value" + id;  
 }  
  
 [HttpPost]  
 public void Post([FromBody] string value)  
 {  
 }  
  
 [HttpPut("{id}")]  
 public void Put(int id, [FromBody] string value)  
 {  
 }  
  
 [HttpDelete("{id}")]  
 public void Delete(int id)  
 {  
 }  
 }  
}

**2. WebApi\_Handson**

### **Objectives:**

* Demonstrate Swagger installation to WebAPI and WebAPI listing on browser
  + Nuget package to download Swashbuckle.AspNetCore, Usage of ProducesResponseType to Web API method, AddSwaggerGen, UseSwaggerUI
* Demonstrate the usage of Postman tool to hit WebAPI methods
  + Structure in Postman tool, Headers with Authorization, Body as JSON, Option to choose the type of request, Request collection and how to add a new request in the collection, Tabs in the center pane that corresponds to the request
* Demonstrate the usage of Route and Explain Name attribute in Http requests
  + Importance of user friendly name to action method, Explain the usage of ActionName to have more than 1 method with the same Action verb. Demonstrate creation of a simple WebAPI - With Read, Write actions

**1. Web Api using .Net core with Swagger**  
Create a .Net core web application with API template. (Use existing application if created). Install Swashbuckle.AspNetCore Nuget package. Add SwaggerGen in ConfigureServices and UseSwagger in Configure method. Open the Swagger UI, check GET method, click “Try it out” and execute.

**2. Use POSTMAN tool**  
Point to the local Web API with Employee controller. Test the GET action method using POSTMAN. Check if employee list is shown in Body and status is correct.

**3. Modify the Controller name**  
Modify the Route attribute of Employee controller to ‘Emp’ and verify accessibility via POSTMAN.

### **Answers:**

#### **Objectives:**

* Swagger provides auto-generated API documentation and UI for testing endpoints.
* Postman is used to test API endpoints using various HTTP verbs with headers and body.
* Routing and ActionName attributes allow clean URL and multiple methods with same HTTP verb.

#### **1. Web Api using .Net core with Swagger**

// Startup.cs  
public void ConfigureServices(IServiceCollection services)  
{  
 services.AddControllers();  
 services.AddSwaggerGen(c =>  
 {  
 c.SwaggerDoc("v1", new OpenApiInfo  
 {  
 Title = "Swagger Demo",  
 Version = "v1",  
 Description = "TBD",  
 TermsOfService = new Uri("https://example.com/terms"),  
 Contact = new OpenApiContact { Name = "John Doe", Email = "john@xyzmail.com", Url = new Uri("https://www.example.com") },  
 License = new OpenApiLicense { Name = "License Terms", Url = new Uri("https://www.example.com") }  
 });  
 });  
}  
  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
 app.UseSwagger();  
 app.UseSwaggerUI(c =>  
 {  
 c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");  
 });  
 app.UseRouting();  
 app.UseEndpoints(endpoints =>  
 {  
 endpoints.MapControllers();  
 });  
}

#### **2. Use POSTMAN tool**

// GET request in POSTMAN  
GET http://localhost:{port}/api/Employee

// Sample Output (in Body tab)  
[  
 {  
 "id": 1,  
 "name": "Alice",  
 "salary": 50000,  
 "permanent": true  
 },  
 {  
 "id": 2,  
 "name": "Bob",  
 "salary": 60000,  
 "permanent": false  
 }  
]

#### **3. Modify the Controller name**

// EmployeeController.cs  
[Route("api/[controller]")]  
[ApiController]  
public class EmpController : ControllerBase  
{  
 [HttpGet]  
 public IEnumerable<Employee> Get()  
 {  
 return GetEmployeeList();  
 }  
}

**3. WebApi\_Handson**

### **Objectives:**

* Demonstrate creation of an Action method to return list of custom class entity
* Model class creation, Use AllowAnonymous attribute, Use HttpGet action method
* Explain the usage of FromBody attribute
* Read the model object from request, other than the query string parameter
* Demonstrate Custom filter
* Usage of ActionFilterAttribute, OnActionExecuting method to intercept the request, Create filter for Custom exception - Need to install Microsoft.AspNetCore.Mvc.WebApiCompatShim package

**1. Web Api using custom model class**  
Create a Custom class ‘Employee’ of the below defined structure

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; }  
}

Create a new controller - EmployeeController with Read Write actions Constructor: Create few records, HTTPGet, HTTPPost/HTTPPut Create a Private method GetStandardEmployeeList that returns a List of Employee class. Invoke this method in the Get action method of the EmployeeController that was created in the previous step. Public ActionResult GetStandrad() Modify the return type of the Get action method(without parameter) to return List of Employee class object Add ProducesResponseType to the GET action method for Status code 200 Check the Swagger description for the GET method for success status code

**2. Create a Custom action filter for Authorization.**  
Create a folder ‘Filters’ in the application solution. Create a class ‘CustomAuthFilter’ to filter requests. Inherit ActionFilterAttribute. Override OnActionExecuting method to check if the request object has Header ‘Authorization’ or not. If not, throw BadRequestResult with the message  
Invalid request - No Auth token  
If the header is present, then check if the value contains the word ‘Bearer’. If not, throw BadRequestResult with the message  
Invalid request - Token present but Bearer unavailable  
Add an attribute CustomAuthFilter to the Employee controller to filter any request to check for the Authorization token in the request header.

**3. Custom Exception filter**  
Create a class ‘CustomExceptionFilter’ to catch the exceptions occurring the application. Implement IExceptionFilter through the OnException method  
Use the exception context to fetch the exception detail. Capture that and write it to a File in the system.  
Set the Result property of the exception context to ExceptionResult.  
Throw an exception in GET action method.  
Ensure that the GET action method has ProducesResponseType for 500 - Internal server error  
Use Swagger to test the exception and message being thrown.

### **Answers:**

#### **Objectives:**

* Custom class entities help structure complex response data.
* AllowAnonymous lets public access to specific API methods.
* FromBody attribute is used to extract complex objects from request body.
* ActionFilterAttribute allows pre-processing of incoming requests (e.g., validation, authorization).

#### **1. Web Api using custom model class**

public class Department  
{  
 public int Id { get; set; }  
 public string Name { get; set; }  
}  
  
public class Skill  
{  
 public int Id { get; set; }  
 public string Name { get; set; }  
}  
  
[Route("api/[controller]")]  
[ApiController]  
public class EmployeeController : ControllerBase  
{  
 private List<Employee> employeeList;  
  
 public EmployeeController()  
 {  
 employeeList = GetStandardEmployeeList();  
 }  
  
 private List<Employee> GetStandardEmployeeList()  
 {  
 return new List<Employee>  
 {  
 new Employee  
 {  
 Id = 1,  
 Name = "John",  
 Salary = 50000,  
 Permanent = true,  
 Department = new Department { Id = 1, Name = "HR" },  
 Skills = new List<Skill> {  
 new Skill { Id = 1, Name = "C#" },  
 new Skill { Id = 2, Name = "SQL" }  
 },  
 DateOfBirth = new DateTime(1990, 01, 01)  
 }  
 };  
 }  
  
 [HttpGet("GetStandard")]  
 [ProducesResponseType(StatusCodes.Status200OK)]  
 [AllowAnonymous]  
 public ActionResult<List<Employee>> GetStandard()  
 {  
 return employeeList;  
 }  
}

#### **2. Create a Custom action filter for Authorization**

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");  
 return;  
 }  
  
 var token = headers["Authorization"].ToString();  
 if (!token.Contains("Bearer"))  
 {  
 context.Result = new BadRequestObjectResult("Invalid request - Token present but Bearer unavailable");  
 }  
 }  
}  
  
[ServiceFilter(typeof(CustomAuthFilter))]  
public class EmployeeController : ControllerBase  
{  
 // controller logic  
}

#### **3. Custom Exception filter**

public class CustomExceptionFilter : IExceptionFilter  
{  
 public void OnException(ExceptionContext context)  
 {  
 var exception = context.Exception;  
 File.WriteAllText("log.txt", exception.ToString());  
  
 context.Result = new ObjectResult("Internal Server Error")  
 {  
 StatusCode = StatusCodes.Status500InternalServerError  
 };  
 }  
}  
  
[ProducesResponseType(StatusCodes.Status500InternalServerError)]  
[ServiceFilter(typeof(CustomExceptionFilter))]  
[HttpGet("ThrowError")]  
public IActionResult ThrowError()  
{  
 throw new Exception("Something went wrong");  
}

**4. WebApi\_Handson**

### **Objectives:**

* Demonstrate creation of an Action method to perform data create, update & delete operation
* Use FromBody attribute, extract data to custom model class using FromBody attribute, use hardcoded data to update & delete data, Use Swagger and POSTMAN to test

**1. Web Api CRUD operation** Update Employee data as per the input thru Web API PUT action method call

Employee information has to be updated based on the user input. Use Swagger tool to invoke the action method mapped with Http PUT action verb to update an employee data. Modify the action method to return Employee data thru ActionResult. Check if the id value is lesser than or equal to 0. If true, throw BadRequest action result with the message ‘Invalid employee id’

If the value is greater than 0 but not available in the list of employee ids that is there in the hardcoded list of employees, throw BadRequest action result with the same message as stated above.

If the id value is valid, use the JSON data from the input body and update the hardcoded list. Filter the employee list data for the input id and return that as the output.

### **Answers:**

#### **Objectives:**

* CRUD operations form the foundation of Web API interactions.
* FromBody helps bind complex data types from the request body.
* Hardcoded data is often used for demo/testing.
* Swagger and Postman are essential tools for testing Web API endpoints.

#### **1. Web Api CRUD operation**

[HttpPut("UpdateEmployee")]  
public ActionResult<Employee> UpdateEmployee([FromBody] Employee emp)  
{  
 if (emp.Id <= 0)  
 {  
 return BadRequest("Invalid employee id");  
 }  
  
 var existingEmp = employeeList.FirstOrDefault(e => e.Id == emp.Id);  
 if (existingEmp == null)  
 {  
 return BadRequest("Invalid employee id");  
 }  
  
 existingEmp.Name = emp.Name;  
 existingEmp.Salary = emp.Salary;  
 existingEmp.Permanent = emp.Permanent;  
 existingEmp.Department = emp.Department;  
 existingEmp.Skills = emp.Skills;  
 existingEmp.DateOfBirth = emp.DateOfBirth;  
  
 return Ok(existingEmp);  
}

**5. WebApi\_Handson**

### **Objectives:**

* Explain CORS enablement for Web API access for local application
  + What is CORS?, How to enable CORS thru Startup.cs, Install Cors nuget package to Web API application
* Demonstrate security in WebAPI
  + Bearer and Jwt token authentication, Use Authorize attribute & send roles in Jwt token, Setting in Startup.cs for AddAuthentication and AddJwtBearer with validation attributes, UseAuthentication, AllowAnonymous to AuthController to generate token, Claims

**1. JsonWebToken**  
Use JWT authentication by configuring Startup.cs. Create AuthController to generate token with claims. Use the token in GET action request header for authorization.

**2. Use the JWT generated thru the AuthController to be used in POSTMAN request.**  
Use Authorize attribute in EmployeeController. Remove CustomAuthFilter if any. Use generated JWT token in POSTMAN to test GET call. Modify token and verify Unauthorized status.

**3. Check for JWT expiration**  
Modify ‘expires’ attribute in AuthController to 2 minutes. Test POSTMAN request after expiry to see Unauthorized message with status 401.

**4. Add the roles to be authorized in the Authorize attribute.**  
Use role ‘POC’ and check Unauthorized status. Use both ‘Admin’ and ‘POC’ roles to validate and get 200 OK status.

### **Answers:**

#### **Objectives:**

* CORS enables cross-origin access from browser-based apps.
* JWT provides secure stateless authentication using tokens.
* Claims help pass metadata such as roles and user info.
* Startup.cs contains configuration for authentication and CORS policy.

#### **1. JsonWebToken**

// Startup.cs  
string securityKey = "mysuperdupersecret";  
var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));  
services.AddAuthentication(x =>  
{  
 x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;  
 x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;  
 x.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;  
})  
.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>  
{  
 x.TokenValidationParameters = new TokenValidationParameters  
 {  
 ValidateIssuer = true,  
 ValidateAudience = true,  
 ValidateLifetime = true,  
 ValidateIssuerSigningKey = true,  
 ValidIssuer = "mySystem",  
 ValidAudience = "myUsers",  
 IssuerSigningKey = symmetricSecurityKey  
 };  
});  
  
app.UseAuthentication();

// AuthController.cs  
[AllowAnonymous]  
[Route("api/[controller]")]  
public class AuthController : ControllerBase  
{  
 [HttpGet("GenerateToken")]  
 public string GetToken()  
 {  
 return GenerateJSONWebToken(123, "Admin");  
 }  
  
 private string GenerateJSONWebToken(int userId, string userRole)  
 {  
 var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret"));  
 var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);  
  
 var claims = new List<Claim>  
 {  
 new Claim(ClaimTypes.Role, userRole),  
 new Claim("UserId", userId.ToString())  
 };  
  
 var token = new JwtSecurityToken(  
 issuer: "mySystem",  
 audience: "myUsers",  
 claims: claims,  
 expires: DateTime.Now.AddMinutes(10),  
 signingCredentials: credentials);  
  
 return new JwtSecurityTokenHandler().WriteToken(token);  
 }  
}

#### **2. Use the JWT generated thru the AuthController to be used in POSTMAN request**

// EmployeeController.cs  
[Authorize]  
[Route("api/[controller]")]  
public class EmployeeController : ControllerBase  
{  
 [HttpGet("GetEmployee")]  
 public ActionResult<string> Get()  
 {  
 return Ok("Authorized access");  
 }  
}

#### **3. Check for JWT expiration**

// In AuthController.cs change token expiration  
var token = new JwtSecurityToken(  
 issuer: "mySystem",  
 audience: "myUsers",  
 claims: claims,  
 expires: DateTime.Now.AddMinutes(2),  
 signingCredentials: credentials);

#### **4. Add the roles to be authorized in the Authorize attribute**

[Authorize(Roles = "POC")]

[Authorize(Roles = "Admin,POC")]