**ASP.NET Core 8.0 Web API**

Suman Sourabh

6363866

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.

**Answer:**

**RESTful Web Service, Web API & Microservice**

A RESTful Web Service follows the REST architectural style, which stands for *Representational State Transfer*. REST uses HTTP for communication between client and server. It treats every object as a resource, accessible via unique URIs (Uniform Resource Identifiers). REST is stateless, meaning the server does not store any session; each request must contain all necessary information. It allows multiple formats for data transfer like JSON, XML, HTML, but in modern applications, JSON is commonly used because it is lightweight.

A Web API is a framework for building HTTP-based services. It implements REST principles and allows applications to expose data to clients such as browsers, mobile apps, or other systems. In .NET, Web API is built using controllers derived from ApiController (in older versions) or ControllerBase (in .NET Core).

A Microservice is a small, independent service focused on doing one specific business function. Each microservice is deployed separately and communicates via HTTP or messaging protocols. Unlike monolithic apps, microservices enable scalability, flexibility, and faster development.

The difference between Web Service and Web API is that Web Services often use SOAP and XML, while Web APIs typically use REST and JSON. Web APIs are lightweight, faster, and more suited for web and mobile applications. Also, Web APIs are not restricted to XML; they can handle JSON, plain text, and more.

**HTTP Request & HTTP Response**

An **HttpRequest** is sent by the client to the server. It includes details like the **HTTP method** (GET, POST, etc.), the **URL**, optional **headers** (like authorization), and a **body** (for POST/PUT).

An **HttpResponse** is the server’s reply to the client. It contains a **status code** (such as 200 for success or 400 for bad request), response **headers**, and the **response body** (usually JSON or XML with the data or result).

**Action Verbs in Web API**

**Action Verbs** represent the operation the API is performing. In RESTful APIs, the most common verbs are:

* **GET** – Retrieves data.
* **POST** – Creates a new resource.
* **PUT** – Updates or replaces an existing resource.
* **DELETE** – Removes a resource.

In .NET Web API, these verbs are implemented using attributes like [HttpGet], [HttpPost], [HttpPut], and [HttpDelete] on action methods inside controll

**HTTP Status Codes in Web API**

When the API processes a request, it returns a **status code** to inform the client about the result. Common status codes include:

* **200 OK** – Request was successful.
* **400 Bad Request** – The client sent invalid data.
* **401 Unauthorized** – Authentication failed.
* **500 Internal Server Error** – Something went wrong on the server side.

In Web API, you typically return these using helper methods like Ok(), BadRequest(), Unauthorized(), or StatusCode(500).

**Create a Simple Web API (Read/Write Actions)**

A Web API project usually consists of:

* Model Classes – Represent data (e.g., Product.cs).
* Controller Classes – Handle HTTP requests (e.g., ProductController.cs).

For example, you can create a Product model with Id and Name. In the controller, use [HttpGet] to return the list of products and [HttpPost] to add a new product. The controller inherits from ControllerBase or ApiController.

**Configuration Files in Web API**

**In .NET Core Web API, the main configuration files are:**

* **Startup.cs – Registers services, middleware, and routing. It’s used for Dependency Injection and setting up things like CORS, Swagger, etc.**
* **appsettings.json – Stores configuration data such as connection strings, API keys, etc.**
* **launchSettings.json – Configures profiles for debugging, like ports and environment variables.**

**In .NET Framework (4.5) Web API, the configuration is done in:**

* **WebApiConfig.cs – Defines API routes using MapHttpRoute.**
* **RouteConfig.cs – Used for MVC routing, not Web API.**
* **Web.config – Stores application settings, connection strings, etc.**

**CODE:**

ValuesController.cs:

using Microsoft.AspNetCore.Mvc;

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

[Route("api/emp")]

[ApiController]

public class ValuesController : ControllerBase

{

[HttpGet]

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

[HttpGet("{id}")]

public string Get(int id) => "value";

[HttpPost]

public void Post([FromBody] string value) { }

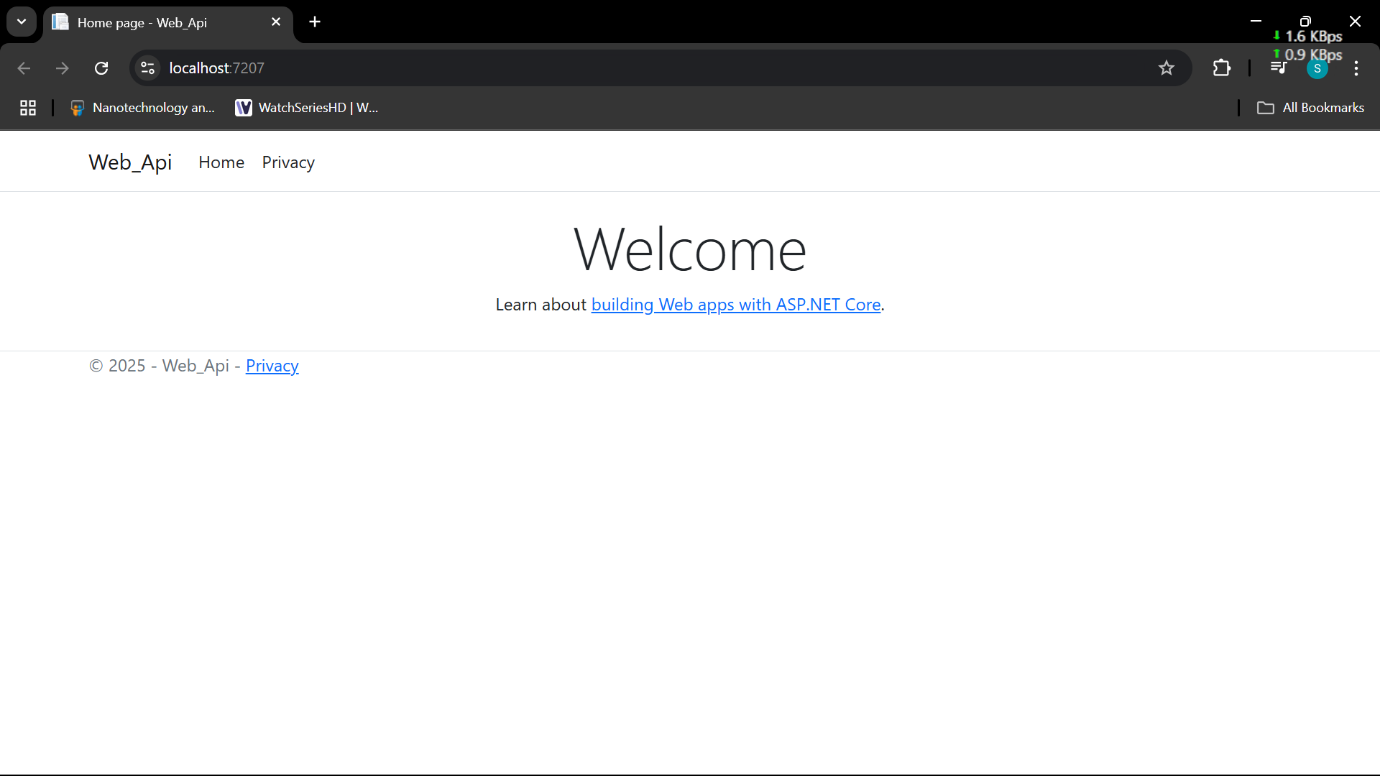
[HttpPut("{id}")]

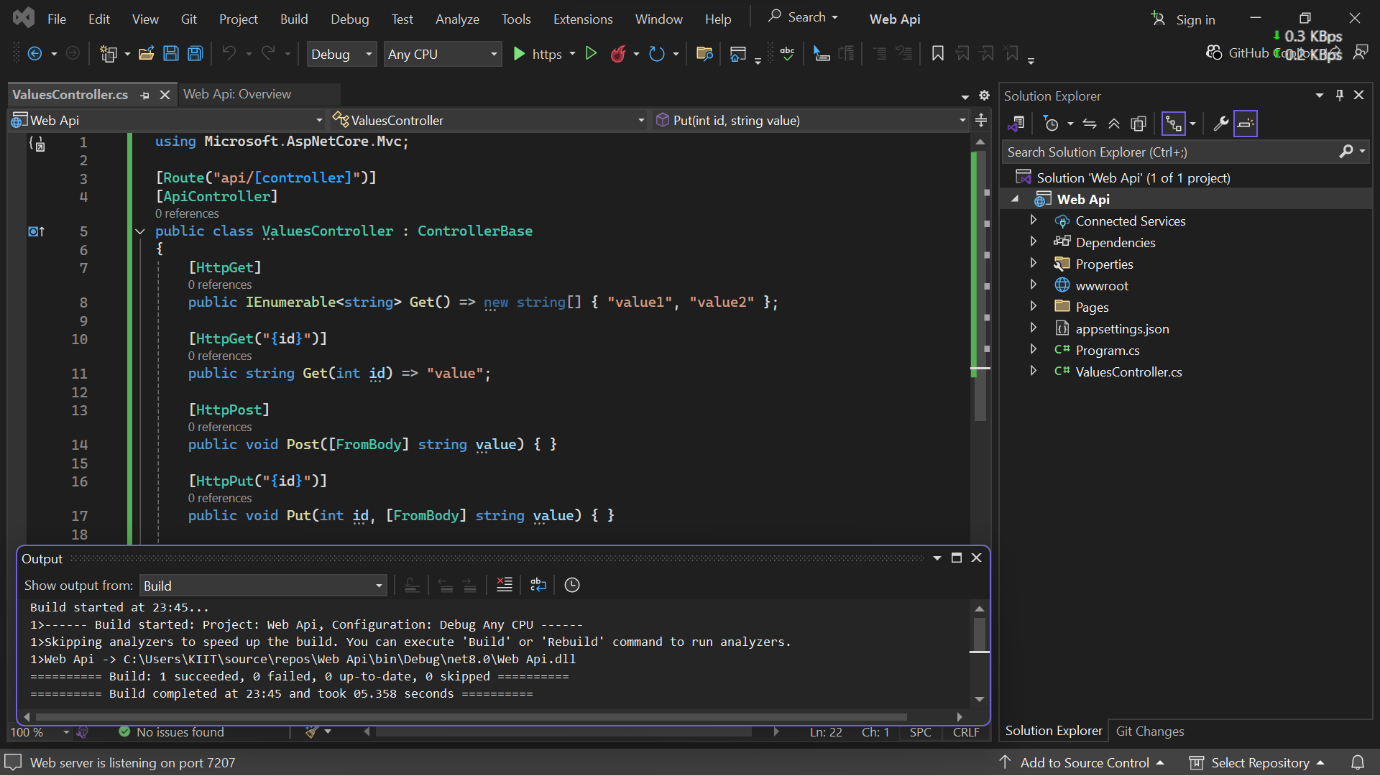
public void Put(int id, [FromBody] string value) { }

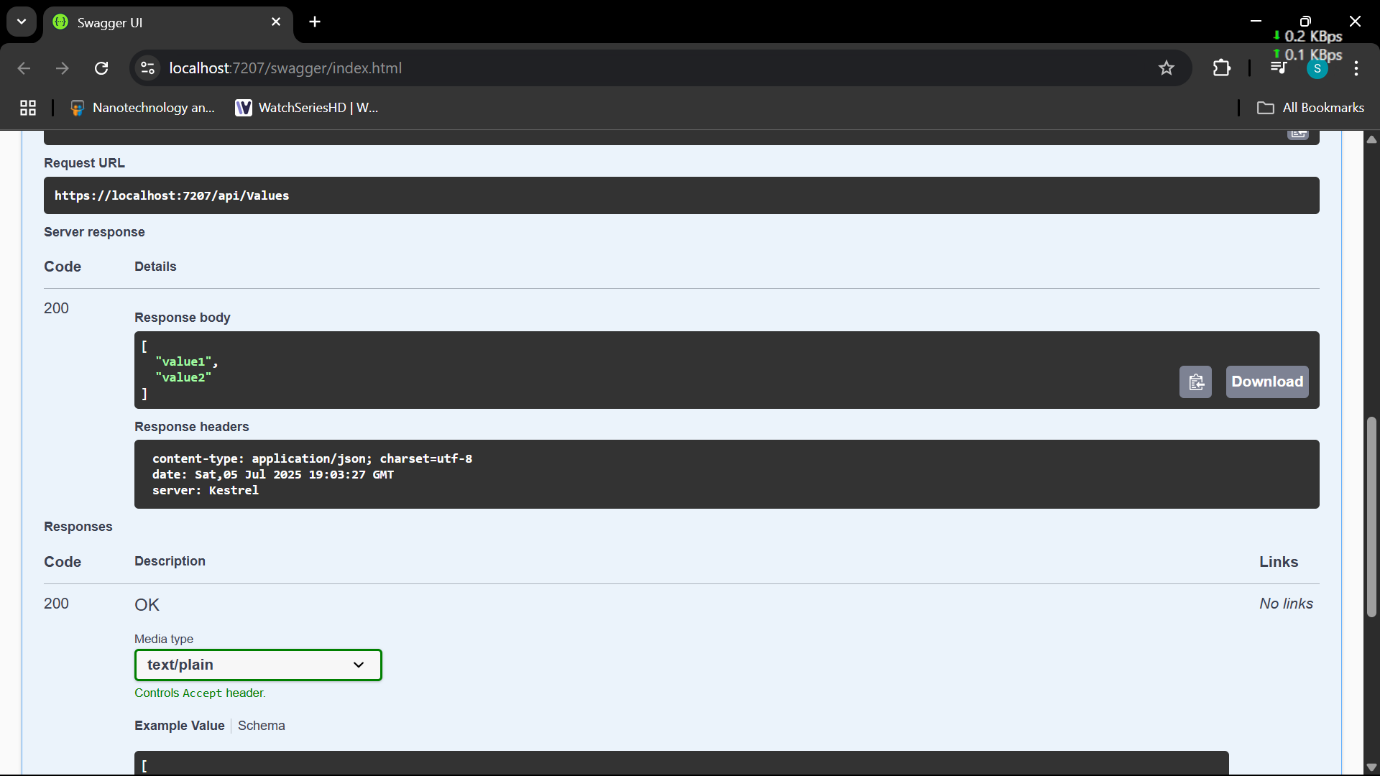
[HttpDelete("{id}")]

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 verbDemonstrate 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. Post this do the following steps in Startup.cs

* In ConfigureServices method, add the code provided below.

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Info

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = "None",

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

License = new License() { Name = "License Terms", Url = "www.example.com" }

});

});

* In Configure method, add the code provided below.

app.UseSwagger();

app.UseSwaggerUI(c =>

{

// specifying the Swagger JSON endpoint.

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

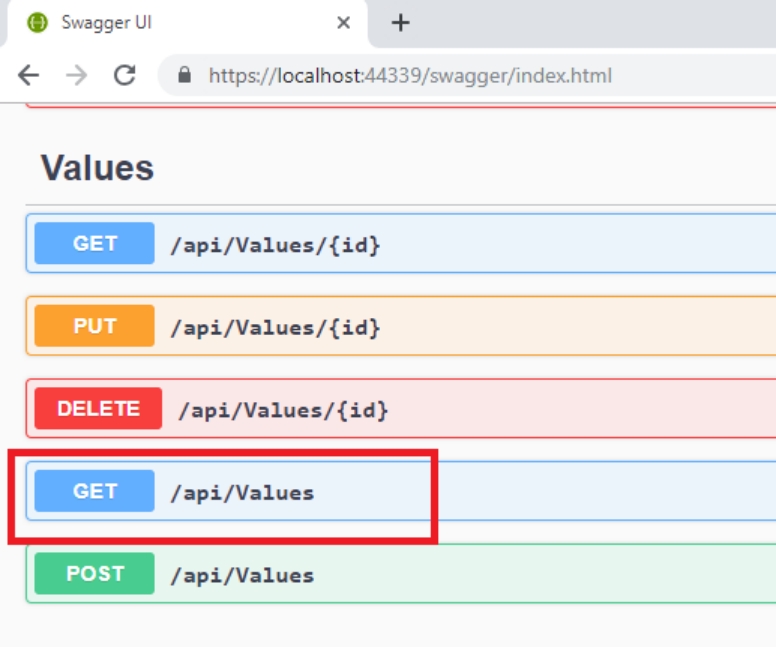
});

Execute the application which will load the default ‘Values’ controller(Settings as per launchSettings.json) GET action method. Change the url to <https://localhost:[port> number]/swagger

Notice the Title, Version, Contact detail provided shown on the top of the page

Notice the Values controller HttpVerb action methods getting listed.

Click the ‘GET’ action verb method(Without the parameter).



It opens a panel which has ‘Try it out’ button. Click that and Click ‘Execute’ button.

1. Use POSTMAN tool, to point to the local Web API that was created with Employee controller. Test the GET action method using POSTMAN.

Verify the output if the List of employees are listed in the ‘Body’ part of the GET window on POSTMAN tool.

Verify the Status on the right side of the output pane on POSTMAN tool.

1. Modify the Controller name in the Route attribute of the Employee controller to ‘Emp’ and check its access thru POSTMAN

**Answer:**

Install-Package Swashbuckle.AspNetCore

**Program.cs**

var builder = WebApplication.CreateBuilder(args);

// Register Swagger services

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Microsoft.OpenApi.Models.OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = new Uri("https://example.com/terms"),

Contact = new Microsoft.OpenApi.Models.OpenApiContact

{

Name = "John Doe",

Email = "john@xyzmail.com",

Url = new Uri("https://www.example.com")

},

License = new Microsoft.OpenApi.Models.OpenApiLicense

{

Name = "License Terms",

Url = new Uri("https://www.example.com")

}

});

});

var app = builder.Build();

// Enable Swagger middleware

app.UseSwagger();

app.UseSwaggerUI(c =>

{

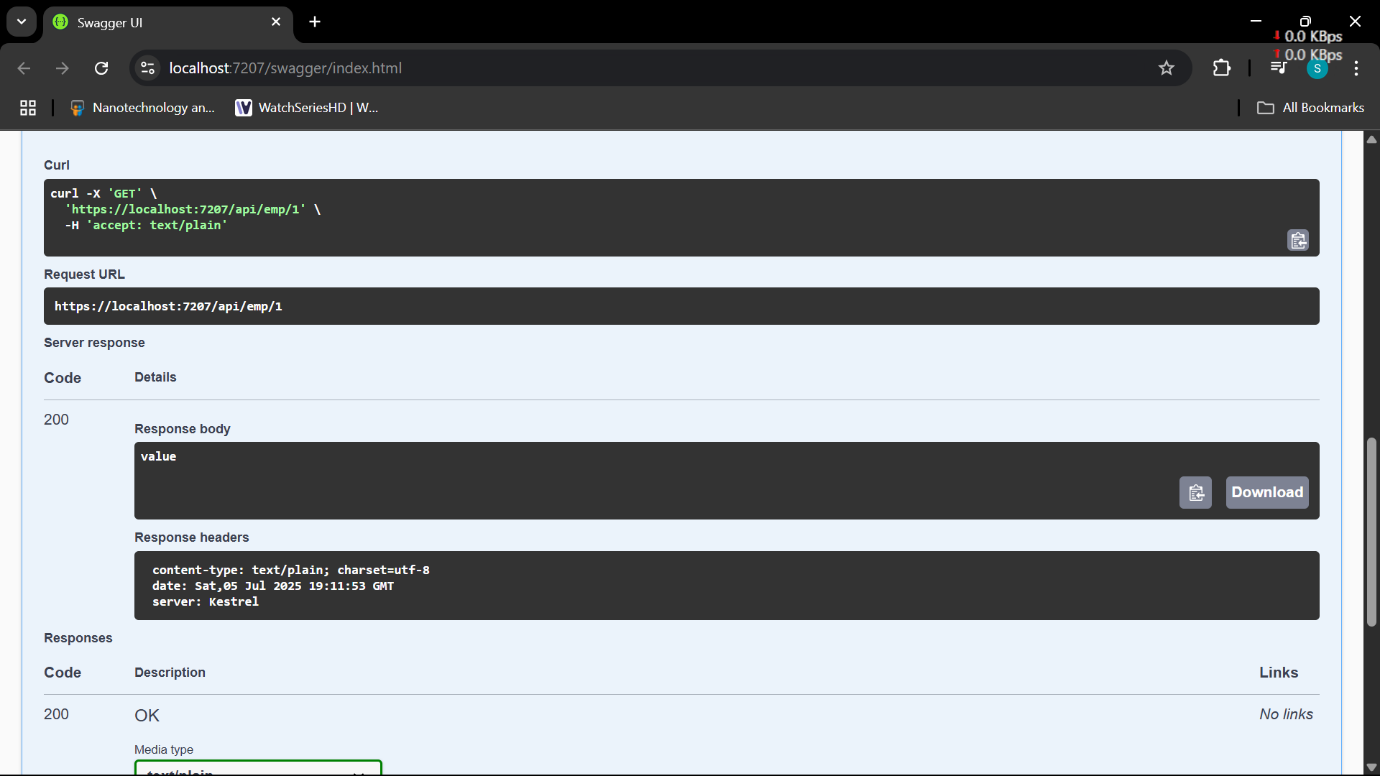
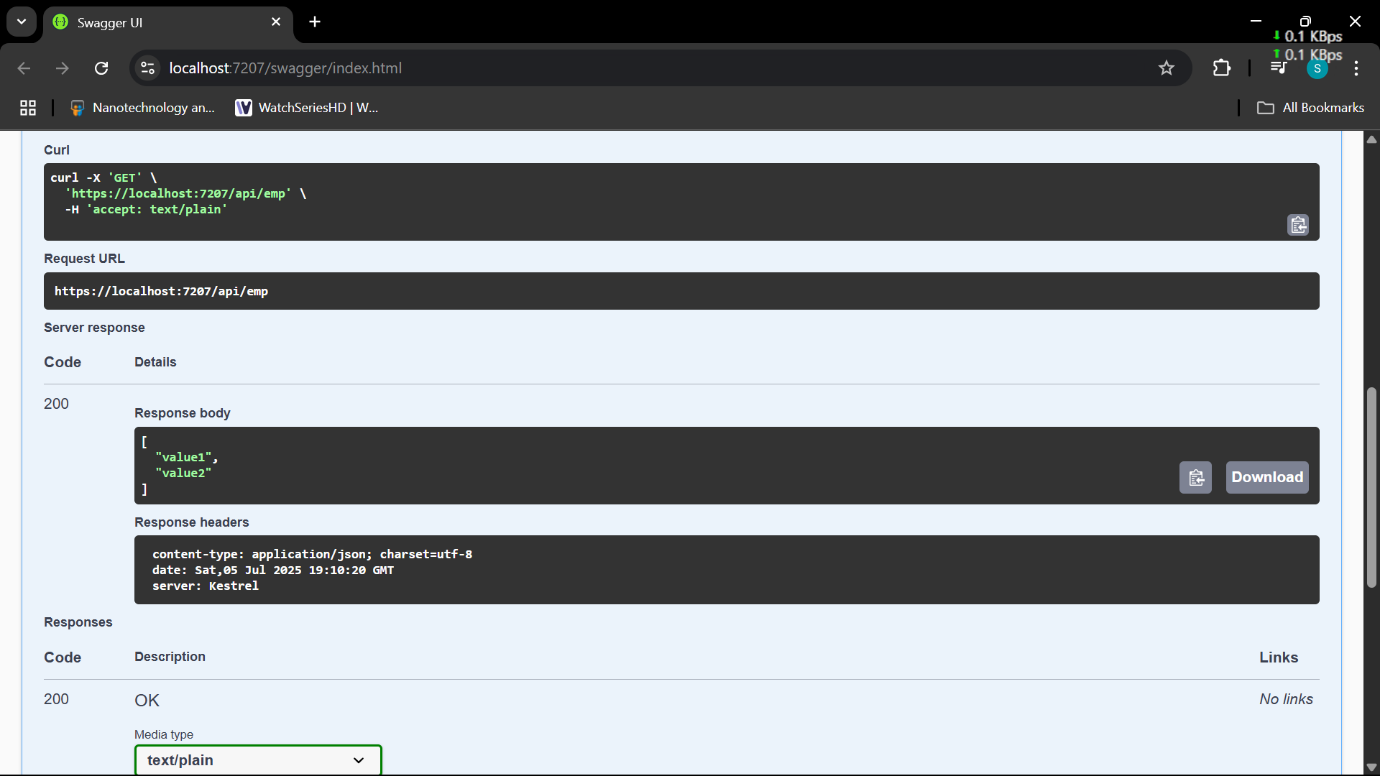
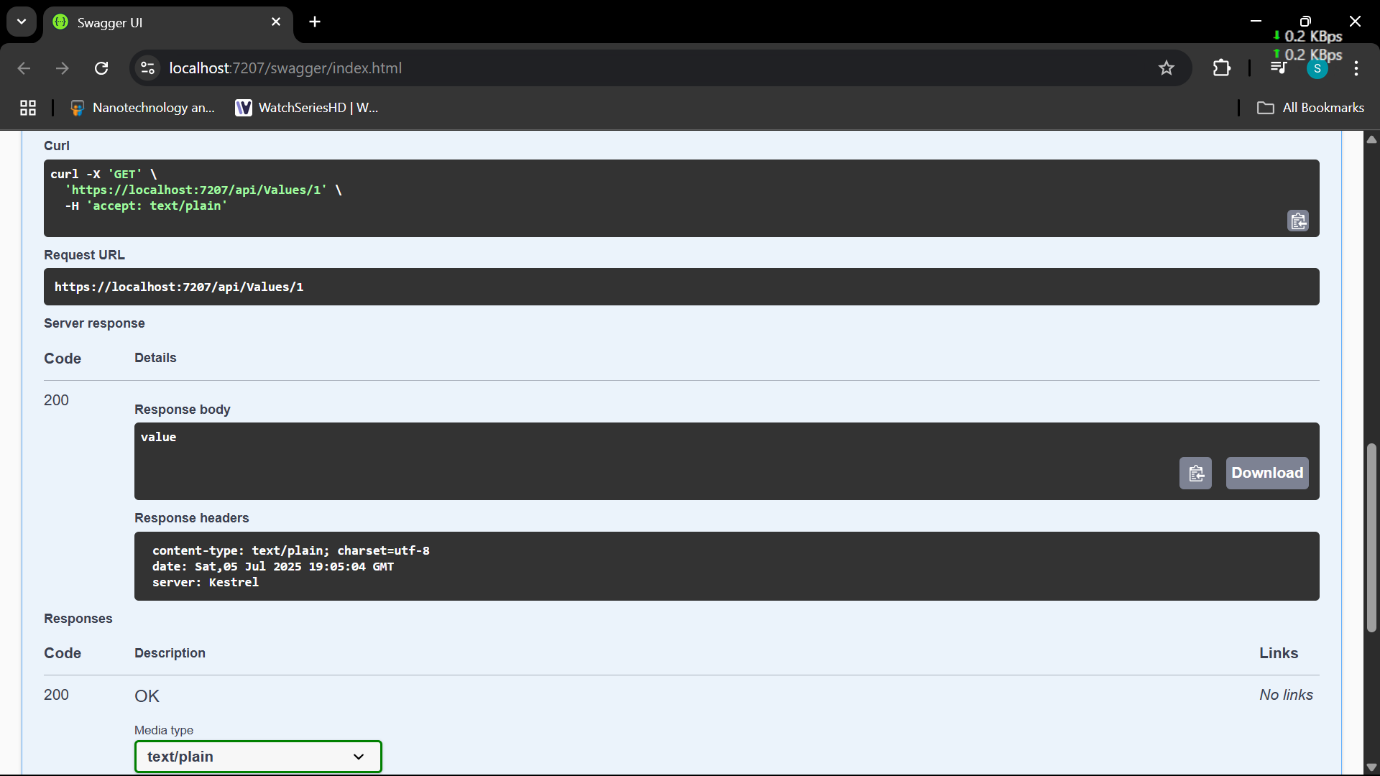
c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

});

app.UseAuthorization();

app.MapControllers();

app.Run();



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<Employee> 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

1. **Create a Custom action filter for Authorization.**

The requirement is to intercept incoming requests and check if there is a key ‘Authorization’ in the request header or not. If it is there, then to check if it contains a value ‘Bearer’ or not.

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.

1. **Custom Exception filter**

Create a class ‘CustomExceptionFilter’ to catch the exceptions occuring the application. Implement IExceptionFilter thru 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.  
  
Note: This needs WebApiCompatShim NuGet package installation

**Answer:**

**Department.cs**

namespace Web\_Api.Models

{

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**Skill.cs**

namespace Web\_Api.Models

{

public class Skill

{

public int Id { get; set; }

public string SkillName { get; set; }

}

}

**Employee.cs**

using System;

using System.Collections.Generic;

namespace Web\_Api.Models

{

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; }

}

}

**EmployeeController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Web\_Api.Models;

using Web\_Api.Filters;

namespace Web\_Api.Controllers

{

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

[ApiController]

[Route("[controller]")]

//[ServiceFilter(typeof(CustomAuthFilter))] // Custom Authorization filter

public class EmployeeController : ControllerBase

{

private static List<Employee> employees = new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

Salary = 50000,

Permanent = true,

Department = new Department { Id = 101, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 1, SkillName = "C#" },

new Skill { Id = 2, SkillName = "SQL" }

},

DateOfBirth = new DateTime(1990, 5, 21)

}

};

[HttpGet("GetStandard")]

[AllowAnonymous]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<List<Employee>> GetStandard()

{

// To test exception filter uncomment this line

//throw new Exception("Test Exception");

return Ok(employees);

}

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> Post([FromBody] Employee emp)

{

if (emp.Id <= 0)

{

return BadRequest("Invalid employee id");

}

if (employees.Any(e => e.Id == emp.Id))

{

return BadRequest("Employee with same id already exists");

}

employees.Add(emp);

return CreatedAtAction(nameof(GetStandard), new { id = emp.Id }, emp);

}

[HttpPut("UpdateEmployee")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> UpdateEmployee([FromBody] Employee inputEmp)

{

if (inputEmp.Id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (emp == null)

{

return BadRequest("Invalid employee id");

}

// Only update fields that are NOT default in input

if (!string.IsNullOrEmpty(inputEmp.Name))

emp.Name = inputEmp.Name;

if (inputEmp.Salary > 0)

emp.Salary = inputEmp.Salary;

emp.Permanent = inputEmp.Permanent;

if (inputEmp.Department != null)

emp.Department = inputEmp.Department;

if (inputEmp.Skills != null && inputEmp.Skills.Count > 0)

emp.Skills = inputEmp.Skills;

if (inputEmp.DateOfBirth != DateTime.MinValue)

emp.DateOfBirth = inputEmp.DateOfBirth;

return Ok(emp);

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public IActionResult Delete(int id)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (emp == null)

{

return BadRequest("Employee not found");

}

employees.Remove(emp);

return Ok($"Employee with id {id} deleted successfully");

}

}

}

**CustomAuthFilter.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace Web\_Api.Filters

{

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

var hasAuthHeader = context.HttpContext.Request.Headers.TryGetValue("Authorization", out var token);

if (!hasAuthHeader)

{

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

return;

}

if (!token.ToString().Contains("Bearer"))

{

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

return;

}

base.OnActionExecuting(context);

}

}

}

**CustomExceptionFilter.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using System.IO;

namespace Web\_Api.Filters

{

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

var exception = context.Exception;

string logPath = "C:\\Logs\\ExceptionLog.txt";

if (!Directory.Exists("C:\\Logs"))

{

Directory.CreateDirectory("C:\\Logs");

}

File.AppendAllText(logPath, $"{DateTime.Now}: {exception.Message}\n{exception.StackTrace}\n\n");

context.Result = new ObjectResult("An error occurred, check the logs.")

{

StatusCode = 500

};

}

}

}

**Program.cs**

using Web\_Api.Filters;

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers(options =>

{

options.Filters.Add(typeof(CustomExceptionFilter)); // Global Exception Filter

});

builder.Services.AddScoped<CustomAuthFilter>(); // Register Custom Auth Filter

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "Demo API with Custom Auth and Exception Filters",

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")

}

});

// Add Bearer Token Authorization to Swagger

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Name = "Authorization",

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer",

BearerFormat = "JWT",

In = ParameterLocation.Header,

Description = "Enter 'Bearer' followed by space and token.\nExample: Bearer xyz123"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] { }

}

});

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

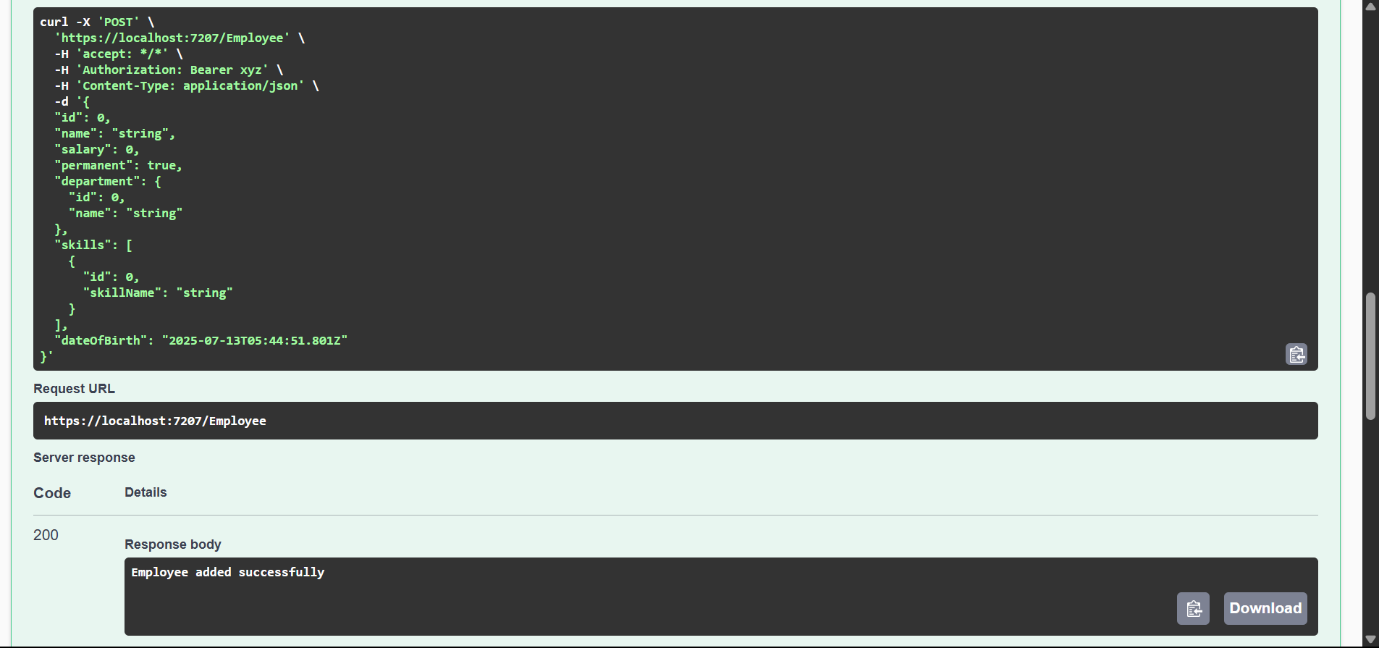
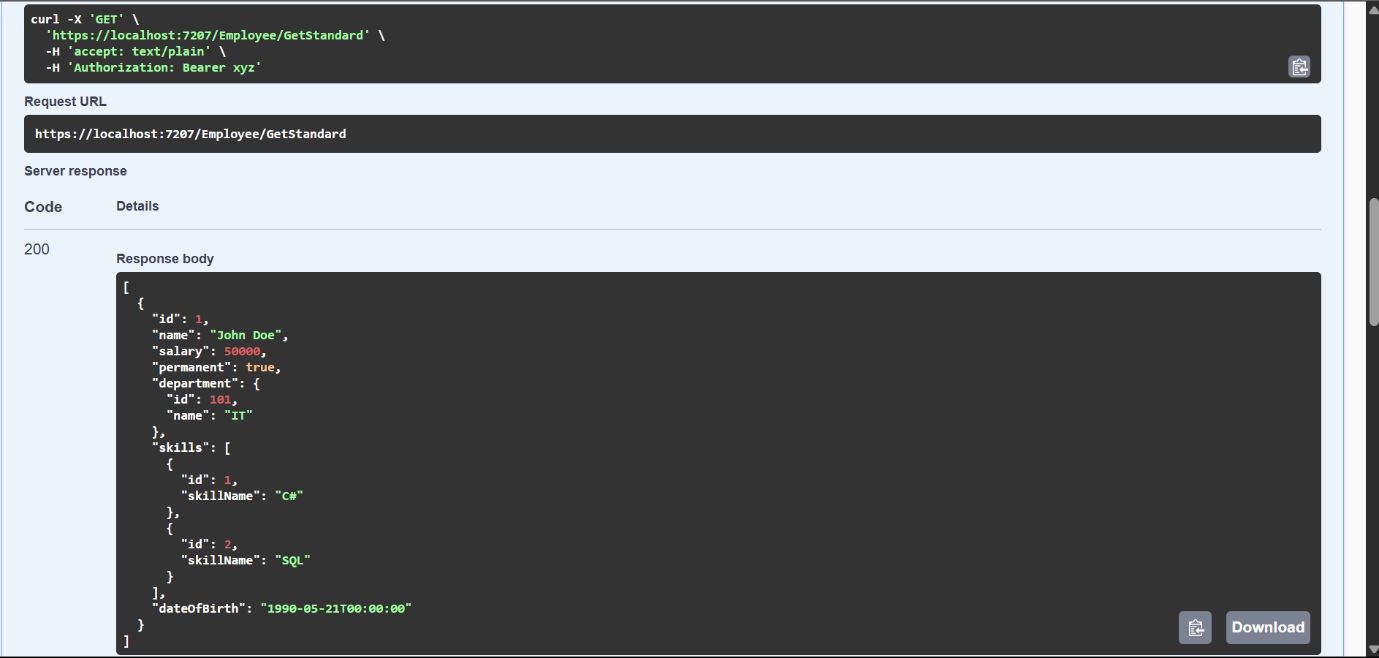
app.UseSwaggerUI();

}

app.UseAuthorization();

app.MapControllers();

app.Run();



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.

**Answer:**

**EmployeeController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Web\_Api.Models;

using Web\_Api.Filters;

namespace Web\_Api.Controllers

{

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

[ApiController]

[Route("[controller]")]

//[ServiceFilter(typeof(CustomAuthFilter))] // Custom Authorization filter

public class EmployeeController : ControllerBase

{

private static List<Employee> employees = new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

Salary = 50000,

Permanent = true,

Department = new Department { Id = 101, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 1, SkillName = "C#" },

new Skill { Id = 2, SkillName = "SQL" }

},

DateOfBirth = new DateTime(1990, 5, 21)

}

};

[HttpGet("GetStandard")]

[AllowAnonymous]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<List<Employee>> GetStandard()

{

// To test exception filter uncomment this line

//throw new Exception("Test Exception");

return Ok(employees);

}

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> Post([FromBody] Employee emp)

{

if (emp.Id <= 0)

{

return BadRequest("Invalid employee id");

}

if (employees.Any(e => e.Id == emp.Id))

{

return BadRequest("Employee with same id already exists");

}

employees.Add(emp);

return CreatedAtAction(nameof(GetStandard), new { id = emp.Id }, emp);

}

[HttpPut("UpdateEmployee")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> UpdateEmployee([FromBody] Employee inputEmp)

{

if (inputEmp.Id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (emp == null)

{

return BadRequest("Invalid employee id");

}

// Only update fields that are NOT default in input

if (!string.IsNullOrEmpty(inputEmp.Name))

emp.Name = inputEmp.Name;

if (inputEmp.Salary > 0)

emp.Salary = inputEmp.Salary;

emp.Permanent = inputEmp.Permanent;

if (inputEmp.Department != null)

emp.Department = inputEmp.Department;

if (inputEmp.Skills != null && inputEmp.Skills.Count > 0)

emp.Skills = inputEmp.Skills;

if (inputEmp.DateOfBirth != DateTime.MinValue)

emp.DateOfBirth = inputEmp.DateOfBirth;

return Ok(emp);

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public IActionResult Delete(int id)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (emp == null)

{

return BadRequest("Employee not found");

}

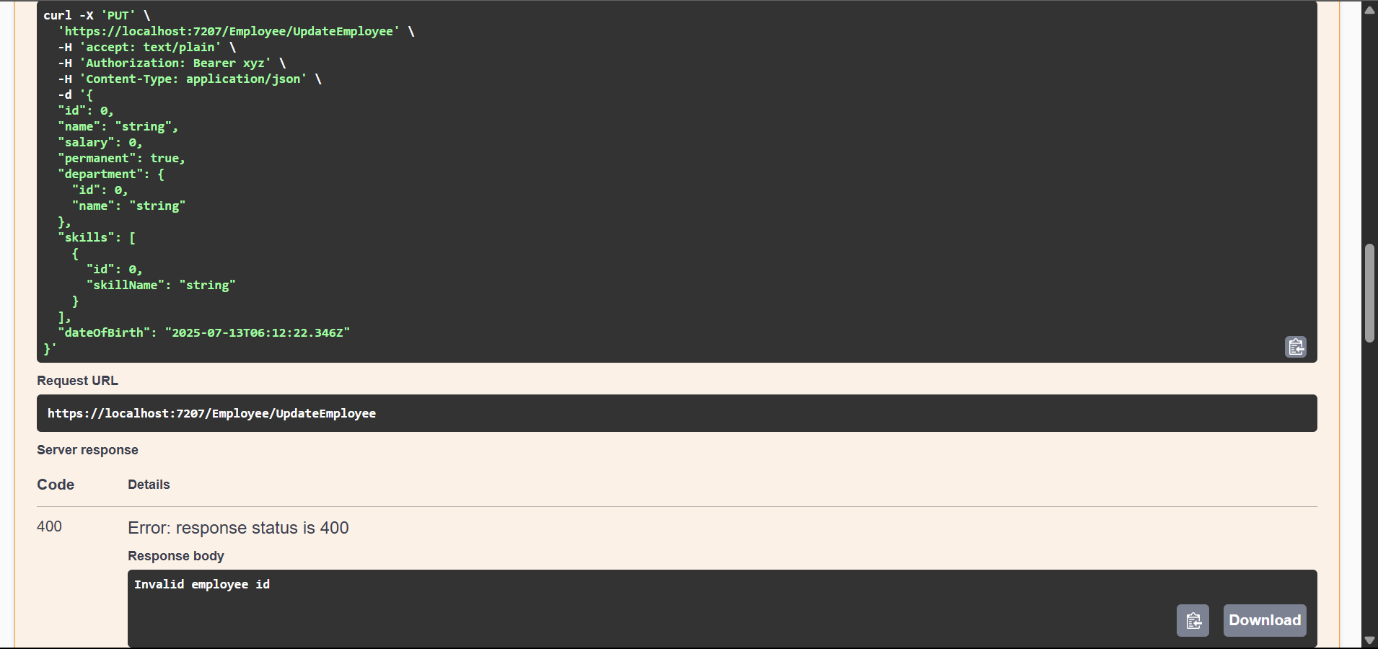
employees.Remove(emp);

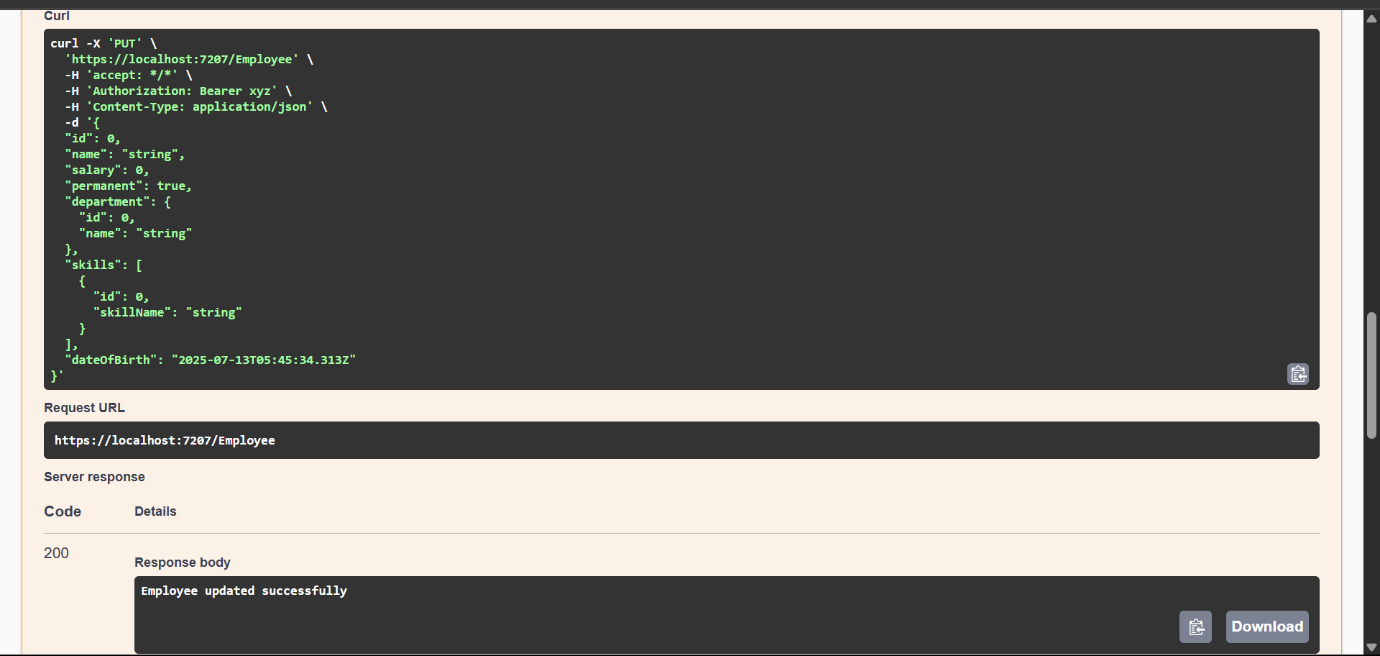
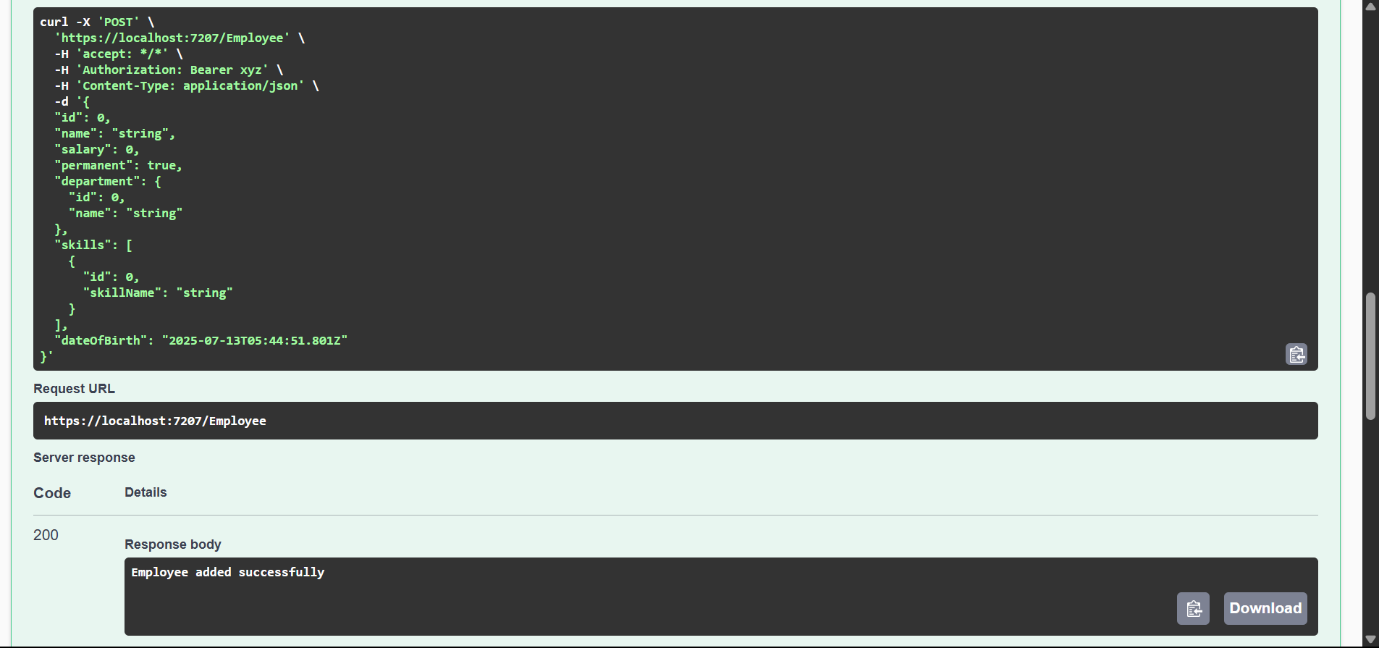
return Ok($"Employee with id {id} deleted successfully");

}

}

}





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**

There are various modes of authenticating a request. Json Web Token(JWT) authentication is one among them. It is a methodology of passing a token in the Authorization header value in the request so that it can be checked at the WebAPI and validated. If not there, then ‘**Unauthorized**’ status message with status code 401 should be thrown.

Use the below code in Startup.cs

* In ConfigureServices method

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

{

//what to validate

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

//setup validate data

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey

};

});

* In Configure method

app.UseAuthentication();

This is to enable the JWT authentication in .Net core

Create a new controller ‘AuthController’ in the Web API application. Add **AllowAnonymous** attribute to the controller. Create a private method GenerateJSONWebToken as shown thru the code below.

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);

}

Note that the issuer, audience and the securitykey defined in the Startup.cs code and method code shown above is the same and should match.

Invoke the GenerateJSONWebToken in the GET action method by sending some value for user id and ‘Admin’ for the user role. This is to set Claims information to check the user role

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

Remove the ‘CustomAuthFilter’ that is currently mapped to the Employee controller(if already done)

Use **Authorize** attribute in the Employee controller to check if the request header contains ‘Authorization’ key with ‘Bearer’ token or not. If the token is unavailable, then ‘Unauthorized’ status message whose code 401 will be thrown. Use POSTMAN to hit a GET action method call. Note the ‘Status’ attribute in the ‘Headers’ section in the output window.

Use the AuthController to generate the JWT. Use that in the GET action method call thru POSTMAN if the request is authenticated or not.

Modify the token value in the POSTMAN tool and check if ‘Unauthorized’ status message is thrown. Note the ‘Status’ attribute in the ‘Headers’ section in the output window.

1. **Check for JWT expiration**

A JWT token has an attribute which can be set to determine how long the token is valid to be used.

In the GenerateJSONWebToken method in AuthController, the ‘expires’ attribute of the JwtSecurityToken object denotes the time in minutes for which the token would be valid.

Modify the duration for ‘expires’ attribute to 2 minutes. Check the POSTMAN request for GET call **AFTER** 2 minutes of generation of the JWT, which should yield ‘Unauthorized’ message with Http status code 401.

1. **Add the roles to be authorized in the Authorize attribute.**

The **Authorize** attribute supports the roles to be used to filter the controller action method access.

The GenerateJSONWebToken method of AuthController uses the role ‘Admin’ set in the claims.

Include the role ‘**POC**’ in the Authorize attribute in the Employee controller. Hit the GET action method of the Employee controller thru POSTMAN. Verify if the response status is ‘Unauthorized’ with status code 401

Include the role ‘**Admin**’ along with ‘POC’ in the Authorize attribute in the Employee controller. Hit the GET action method of the Employee controller thru POSTMAN. Verify if the response status is OK with status code 200

**Answer:**

CORS (Cross-Origin Resource Sharing) allows a web application running at one origin (domain) to access resources from a different origin.

**CODE:**

**Program.cs:**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

using Web\_Api.Filters;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers(options =>

{

options.Filters.Add(typeof(CustomExceptionFilter)); // Global Exception Filter

});

// CORS Policy for local frontend (http://localhost:3000)

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowLocalhost3000", builder =>

{

builder.WithOrigins("http://localhost:3000")

.AllowAnyHeader()

.AllowAnyMethod();

});

});

// Register CustomAuthFilter (commented out since it's no longer needed)

// builder.Services.AddScoped<CustomAuthFilter>();

builder.Services.AddEndpointsApiExplorer();

// Swagger JWT Bearer Token Setup

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "Demo API with JWT Authentication and Exception Filters",

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")

}

});

// Bearer Token Definition for Swagger UI

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Name = "Authorization",

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer",

BearerFormat = "JWT",

In = ParameterLocation.Header,

Description = "Enter 'Bearer' followed by space and token.\nExample: Bearer xyz123"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] { }

}

});

});

// JWT Authentication Configuration

var securityKey = "mysuperdupersecret";

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

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey

};

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseCors("AllowLocalhost3000");

app.UseAuthentication(); // Enable JWT Authentication

app.UseAuthorization();

app.MapControllers();

app.Run();

**AuthController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace Web\_Api.Controllers

{

[ApiController]

[Route("[controller]")]

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet("GenerateToken")]

public IActionResult GenerateToken(int userId, string role)

{

var token = GenerateJSONWebToken(userId, role);

return Ok(new { Token = token });

}

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecretkey\_that\_is\_long\_enough\_123!"));

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), // set to 2 for expiration check

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**EmployeeController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Web\_Api.Models;

using Web\_Api.Filters;

namespace Web\_Api.Controllers

{

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

[ApiController]

[Route("[controller]")]

//[ServiceFilter(typeof(CustomAuthFilter))] // Custom Authorization filter

public class EmployeeController : ControllerBase

{

private static List<Employee> employees = new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

Salary = 50000,

Permanent = true,

Department = new Department { Id = 101, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 1, SkillName = "C#" },

new Skill { Id = 2, SkillName = "SQL" }

},

DateOfBirth = new DateTime(1990, 5, 21)

}

};

[HttpGet("GetStandard")]

[AllowAnonymous]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<List<Employee>> GetStandard()

{

// To test exception filter uncomment this line

//throw new Exception("Test Exception");

return Ok(employees);

}

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> Post([FromBody] Employee emp)

{

if (emp.Id <= 0)

{

return BadRequest("Invalid employee id");

}

if (employees.Any(e => e.Id == emp.Id))

{

return BadRequest("Employee with same id already exists");

}

employees.Add(emp);

return CreatedAtAction(nameof(GetStandard), new { id = emp.Id }, emp);

}

[HttpPut("UpdateEmployee")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> UpdateEmployee([FromBody] Employee inputEmp)

{

if (inputEmp.Id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (emp == null)

{

return BadRequest("Invalid employee id");

}

// Only update fields that are NOT default in input

if (!string.IsNullOrEmpty(inputEmp.Name))

emp.Name = inputEmp.Name;

if (inputEmp.Salary > 0)

emp.Salary = inputEmp.Salary;

emp.Permanent = inputEmp.Permanent;

if (inputEmp.Department != null)

emp.Department = inputEmp.Department;

if (inputEmp.Skills != null && inputEmp.Skills.Count > 0)

emp.Skills = inputEmp.Skills;

if (inputEmp.DateOfBirth != DateTime.MinValue)

emp.DateOfBirth = inputEmp.DateOfBirth;

return Ok(emp);

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public IActionResult Delete(int id)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (emp == null)

{

return BadRequest("Employee not found");

}

employees.Remove(emp);

return Ok($"Employee with id {id} deleted successfully");

}

}

}

