COGNIZANT DN - 4.0 DEEP SKILLING

HANDS ON WEEK-4

ASP.NET Core 8.0 Web API

1. **Web API hands on(1)**
2. **Explain the concept of RESTful web service, Web API & Microservice.**

**Answer:**

1. RESTful Web Service

REST (Representational State Transfer) is an architectural style for designing networked applications. A RESTful Web Service allows systems to communicate over HTTP by using standard HTTP methods (GET, POST, PUT, DELETE, etc.).

2. Features of REST Architecture

| **Feature** | **Explanation** |
| --- | --- |
| **Stateless** | Each request from client to server must contain all the information to understand and process it. Server does not store client context between requests. |
| **Client-Server** | The client and server are independent. The interface (usually HTTP) separates the client (frontend) from the server (backend). |
| **Cacheable** | Responses should define themselves as cacheable or not to improve performance. |
| **Uniform Interface** | Standard HTTP methods (GET, POST, PUT, DELETE) are used consistently across resources. |
| **Representation** | Resources are represented in various formats (like JSON, XML, HTML). Clients interact with representations of resources. |
| **Messages** | Communication between client and server happens using HTTP requests and responses, often with JSON or XML as the message body. |

## **3. Web API**

* **Web API (Application Programming Interface)** is a **set of HTTP-based endpoints** that allow different software systems to interact.
* It is **not limited to REST**, but REST is a common implementation.
* A Web API can return data in formats like **JSON, XML, or plain text,** depending on the client's needs.

## **4. Microservices**

**Microservice Architecture** is a method of developing software systems where each component or **service** is small, independently deployable, and responsible for a specific business function.

5.Web Service and Web API

| **Feature** | **Web Service** | **Web API** |
| --- | --- | --- |
| **Definition** | A standardized way of communication over a network using protocols like SOAP or REST | A broader term for any interface over HTTP |
| **Protocol** | Often uses **SOAP** or **REST** | Primarily uses **HTTP** with REST-style access |
| **Data Format** | Often uses **XML** | Uses **JSON**, **XML**, or other formats |
| **Platform Dependency** | Platform-independent | Platform-independent |
| **Complexity** | More formal and rigid (especially SOAP) | Lightweight and easy to use |

1. **Explain what is HttpRequest & HttpResponse**.

**Answer:**

**HttpRequest**

An **HttpRequest** is the message sent by a **client (such as a web browser, mobile app, or any HTTP client)** to a **web server**, asking it to perform a specific action or retrieve a resource. This is the first step in any communication over the web using the HTTP protocol.

HttpResponse

An **HttpResponse** is the server's reply to an HttpRequest. Once the server receives and processes the client's request, it generates an HttpResponse message and sends it back to the client. This response contains information about whether the request was successful and, if so, includes the data requested or the result of the action performed.

1. **List the types of Action Verbs.**

**Answers:**

These are the main **types of HTTP action verbs** used in Web APIs:

1. **HttpGet**

* **Purpose**: Used to **retrieve data** from the server.
* **No request body** is required.
* It is **safe and idempotent**, meaning it doesn't change anything on the server.

Example:

[HttpGet]

public IActionResult GetAllUsers() {

// Returns a list of users

}

2. **HttpPost**

* **Purpose**: Used to **send data to the server** to create a new resource.
* The data is usually sent in the **request body**, often as JSON.
* Not idempotent multiple identical requests may create multiple records.

Example:

[HttpPost]

public IActionResult CreateUser([FromBody] User user) {

// Adds a new user

}

3. **HttpDelete**

* **Purpose**: Used to **delete a resource** on the server.
* It is also **idempotent** deleting the same resource multiple times has the same outcome.

Example:

[HttpDelete("{id}")]

public IActionResult DeleteUser(int id) {

// Deletes user with the given ID

}

4. **HttpPut**

* **Purpose**: Used to **update an existing resource** (usually replacing it entirely).
* The updated data is sent in the request body.
* It is **idempotent** sending the same request multiple times will have the same effect.

Example:

[HttpPut("{id}")]

public IActionResult UpdateUser(int id, [FromBody] User user) {

// Updates user with the given ID

}

1. **List the types of HttpStatusCodes used in WebAPI.**

**Answer:**

**1. 200 OK**

* **Meaning**: The request was successful.
* **Used When**: You want to return a successful result along with data.
* **ActionResult**: return Ok(data);

**2. 400 Bad Request**

* **Meaning**: The server could not understand the request due to invalid syntax or missing data.
* **Used When**: Input validation fails.
* **ActionResult**: return BadRequest("Invalid input");

**3. 500 Internal Server Error**

* **Meaning**: A generic server error occurred.
* **Used When**: An unhandled exception or failure occurs in processing.
* **ActionResult**: return StatusCode(500, "Server error occurred");

**4. 401 Unauthorized**

* **Meaning**: The user is not authenticated.
* **Used When**: Authentication is required but not provided or failed.
* **ActionResult**: return Unauthorized();

1. **Demonstrate creation of a simple WebAPI - With Read, Write actions.**

**Code:**

using System.Collections.Generic;

using System.Web.Http;

using YourNamespace.Models;

public class ProductController : ApiController

{

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

{

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

new Product { Id = 2, Name = "Mouse", Price = 700 }

};

[HttpGet]

public IHttpActionResult GetAllProducts()

{

return Ok(products);

}

[HttpGet]

public IHttpActionResult GetProduct(int id)

{

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

if (product == null)

return NotFound();

return Ok(product);

}

[HttpPost]

public IHttpActionResult AddProduct(Product newProduct)

{

if (!ModelState.IsValid)

return BadRequest("Invalid data");

products.Add(newProduct);

return Created(Request.RequestUri + "/" + newProduct.Id, newProduct);

}

}

1. **Explain the types of Configuration files of WebAPI.**

**Answers:**

* 1. In ASP.NET Core Web API

1. Startup.cs

* **Purpose**: Core configuration file that sets up services, middleware, routing, and dependency injection.
* **Contains**:
  + ConfigureServices() → Adds services like DB context, authentication, CORS.
  + Configure() → Defines middleware pipeline (routing, authorization, etc.)

2. appsettings.json

* **Purpose**: Stores configuration data like connection strings, API keys, JWT secrets, etc.
* **Structure**: JSON format, easily readable and hierarchical.

3. launchSettings.json

* **Purpose**: Development-time configuration that sets how the app launches.
* **Includes**:
  + Ports
  + Environment (Development, Production)
  + Whether to open browser

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

1. Web.config

* **Purpose**: Primary configuration file for ASP.NET apps.
* **Stores**:
  + App settings
  + Connection strings
  + Authentication
  + Custom errors
  + HTTP handlers/module

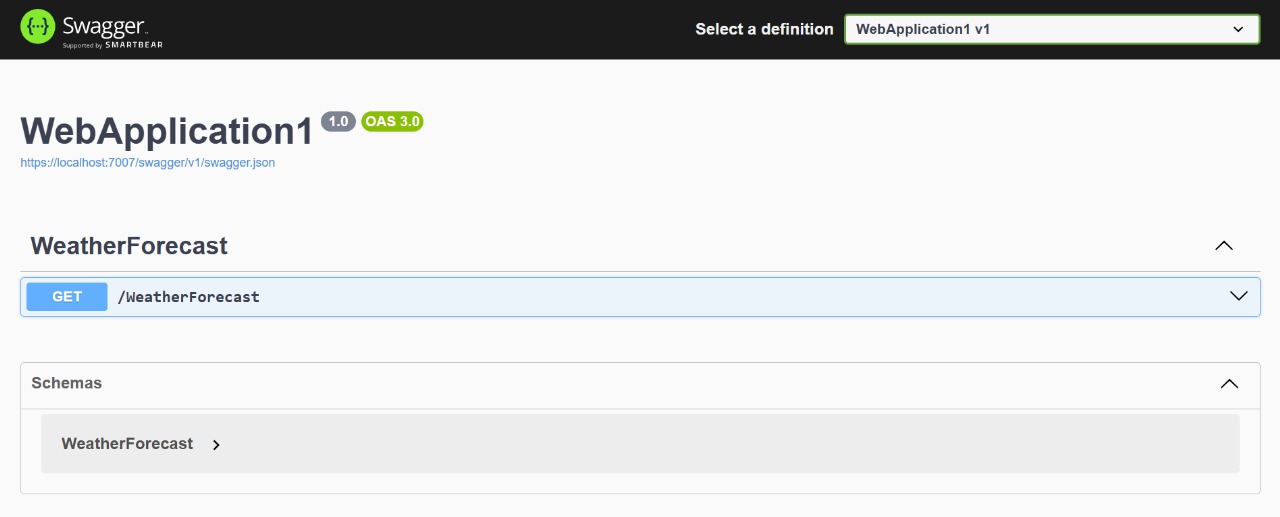
2. WebApiConfig.cs

* **Purpose**: Defines routes and configuration for Web API.
* **Location**: Usually under App\_Start/

3. RouteConfig.cs (For MVC apps only)

* Used for setting up MVC routes, not necessary for Web API unless MVC is integrated

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

****

1. **Web API hands on(2)**
   * + 1. **Web Api using .Net core with Swagger**

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

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

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")

}

});

});

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI(c =>

{

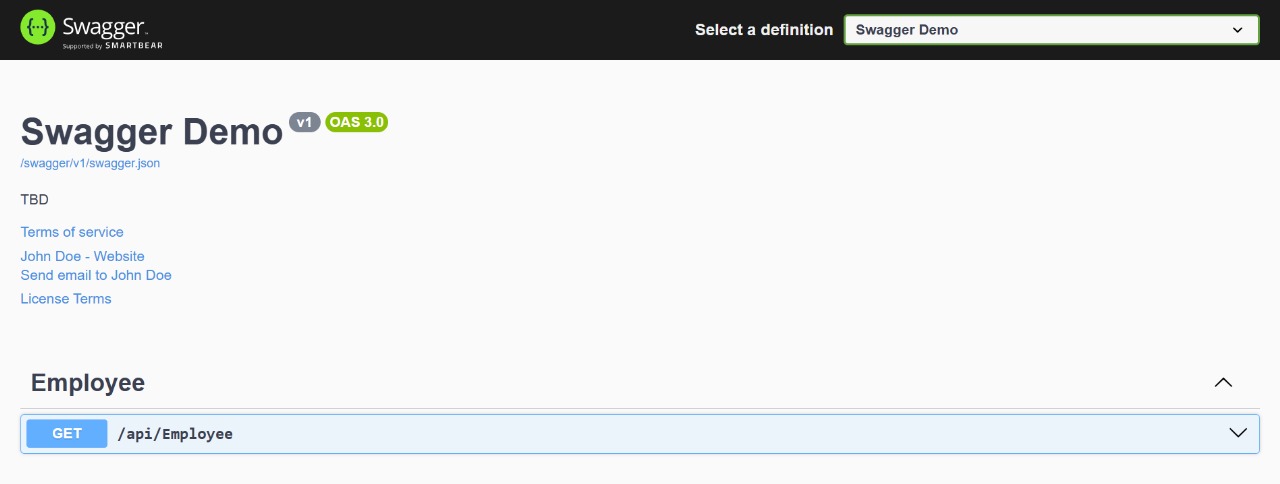
c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

});

app.UseAuthorization();

app.MapControllers();

app.Run();



**2.Use POSTMAN tool, to point to the local Web API that was created with Employee controller.**

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

public class EmployeeController : ControllerBase

{

[HttpGet]

public IActionResult Get()

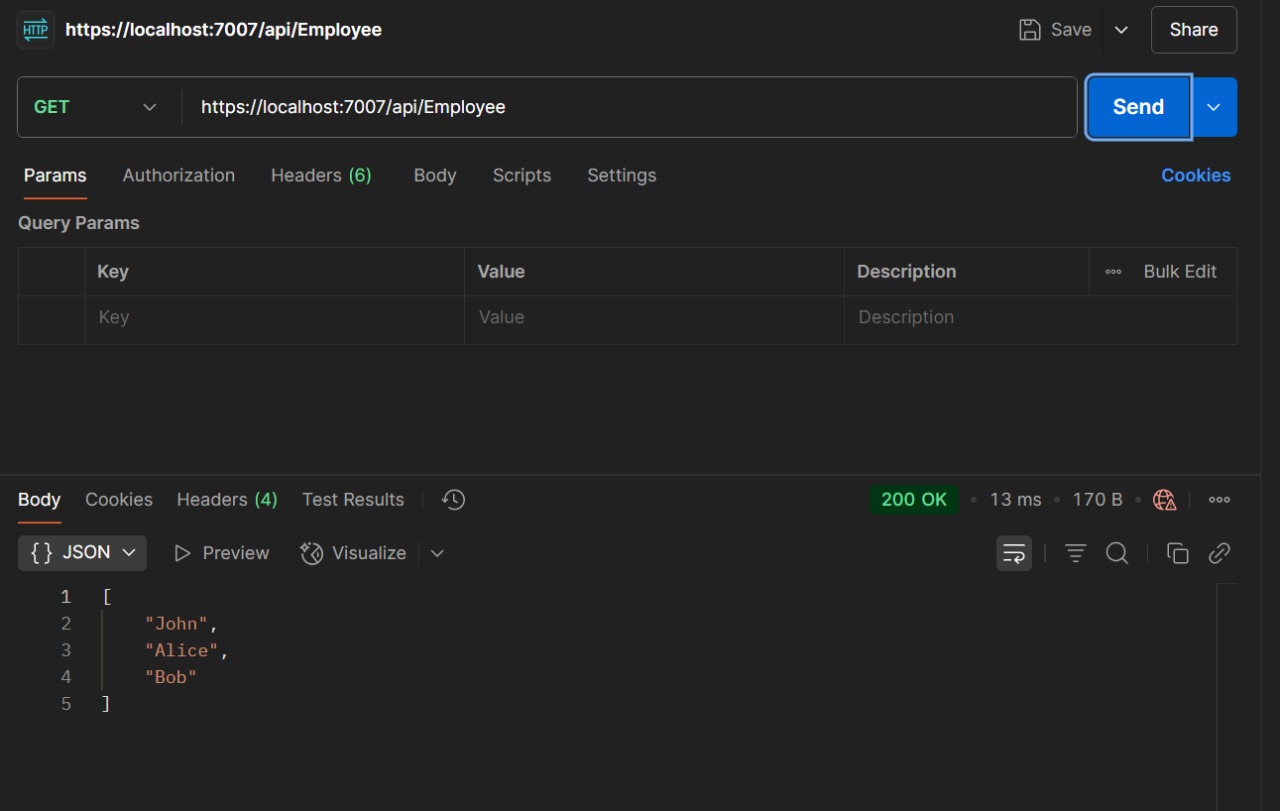
{

var employees = new List<string> { "John", "Alice", "Bob" };

return Ok(employees);

}

}



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

using Microsoft.AspNetCore.Mvc;

[ApiController]

[Route("api/emp")]

public class EmployeeController : ControllerBase

{

[HttpGet]

public IActionResult Get()

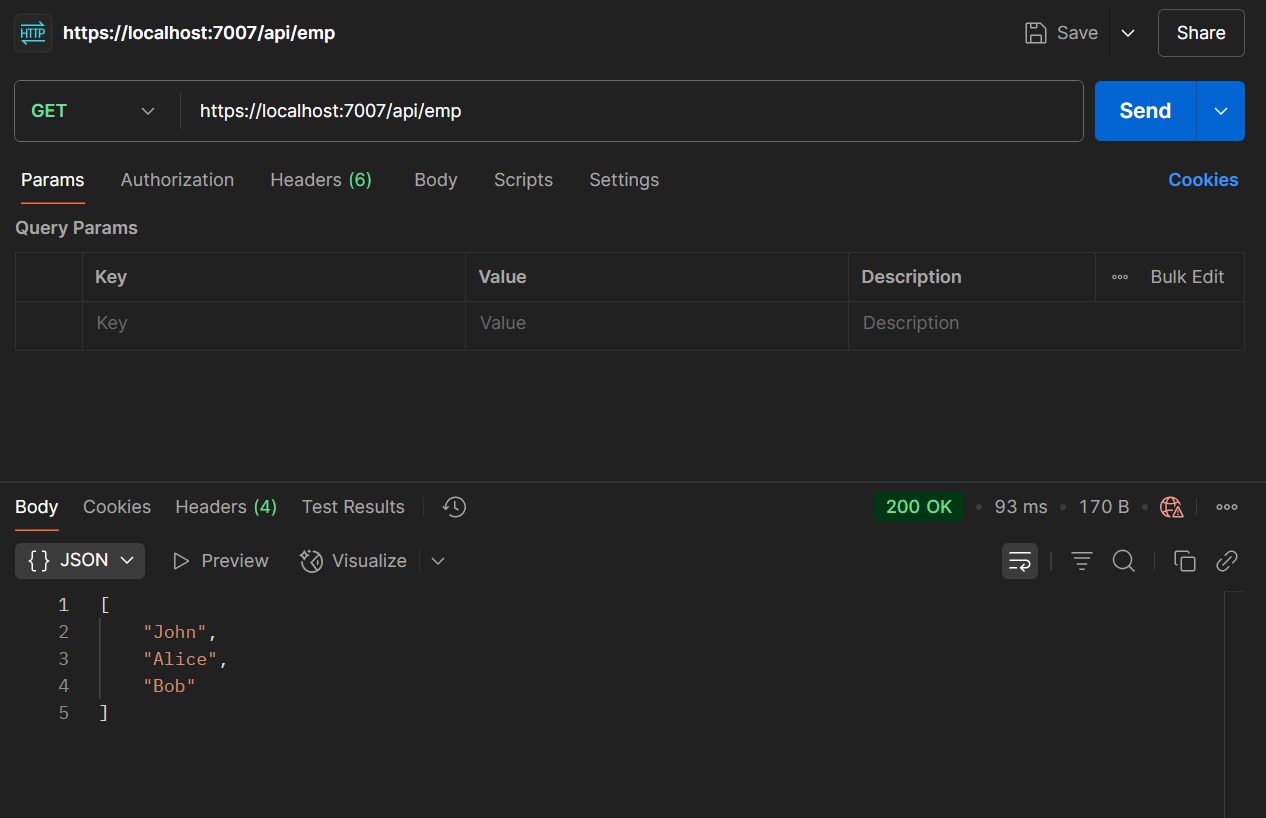
{

var employees = new List<string> { "John", "Alice", "Bob" };

return Ok(employees);

}

}

****

1. **Web API hands on(3)**
   * + 1. **Web Api using custom model class**

using Microsoft.AspNetCore.Mvc;

using WebApplication1.Models;

using WebApplication1.Models;

namespace WebApplication1.Controllers

{

[ApiController]

[Route("api/emp")]

public class EmployeeController : ControllerBase

{

private readonly List<Employee> \_employees;

public EmployeeController()

{

\_employees = GetStandardEmployeeList();

}

[HttpGet]

[ProducesResponseType(typeof(List<Employee>), StatusCodes.Status200OK)]

public ActionResult<List<Employee>> Get()

{

return Ok(\_employees);

}

[HttpGet("standard")]

public ActionResult<Employee> GetStandard()

{

return Ok(\_employees.FirstOrDefault());

}

[HttpPost]

public ActionResult Create([FromBody] Employee employee)

{

\_employees.Add(employee);

return Ok(employee);

}

[HttpPut("{id}")]

public ActionResult Update(int id, [FromBody] Employee employee)

{

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

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

emp.Name = employee.Name;

emp.Salary = employee.Salary;

emp.Permanent = employee.Permanent;

emp.Department = employee.Department;

emp.Skills = employee.Skills;

emp.DateOfBirth = employee.DateOfBirth;

return Ok(emp);

}

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "John",

Salary = 50000,

Permanent = true,

DateOfBirth = new DateTime(1990, 1, 1),

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

Skills = new List<Skill>

{

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

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

}

}

};

}

}

}

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

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace WebApplication1.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;

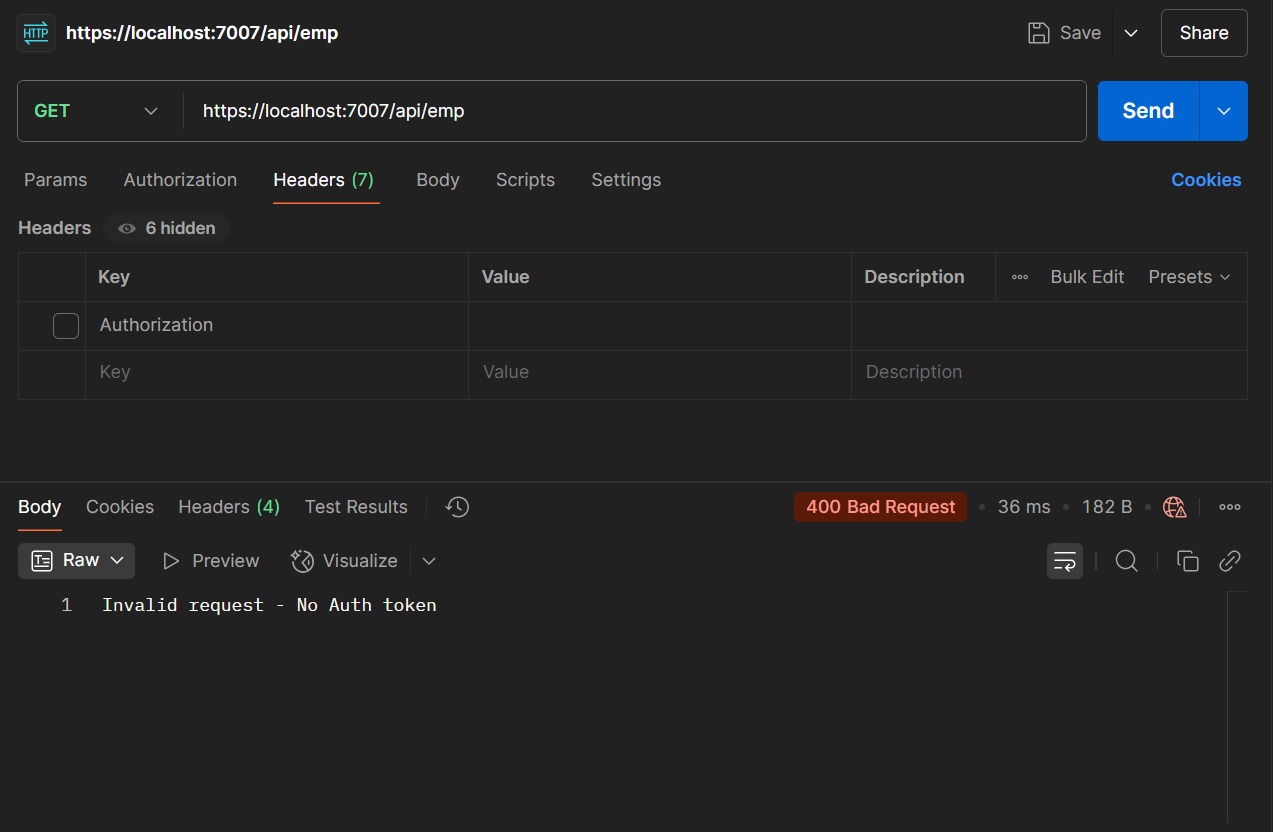
}

base.OnActionExecuting(context);

}

}

}



* + - 1. **Custom Exception filter**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace WebApplication1

.Filters

{

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

var exception = context.Exception;

var errorMessage = $"[{DateTime.Now}] {exception.Message}\n{exception.StackTrace}\n\n";

File.AppendAllText("exception\_log.txt", errorMessage);

context.Result = new ObjectResult("An unexpected error occurred.")

{

StatusCode = 500

};

context.ExceptionHandled = true;

}

}

}

1. **Web API hands on(4)**
   * + 1. **Web Api CRUD operation**

using Microsoft.AspNetCore.Mvc;

using WebApplication1.Filters;

using WebApplication1.Models;

namespace YourProject.Controllers

{

[ApiController]

[Route("api/emp")]

[CustomAuthFilter]

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees = new List<Employee>

{

new Employee

{

Id = 1,

Name = "John",

Salary = 50000,

Permanent = true,

DateOfBirth = new DateTime(1990, 1, 1),

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

Skills = new List<Skill>

{

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

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

}

},

new Employee

{

Id = 2,

Name = "Alice",

Salary = 60000,

Permanent = false,

DateOfBirth = new DateTime(1992, 5, 15),

Department = new Department { Id = 102, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 3, Name = "Java" },

new Skill { Id = 4, Name = "React" }

}

}

};

[HttpGet]

[ProducesResponseType(typeof(List<Employee>), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public ActionResult<List<Employee>> Get()

{

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

return Ok(\_employees);

}

[HttpGet("standard")]

public ActionResult<Employee> GetStandard()

{

return Ok(\_employees.FirstOrDefault());

}

[HttpPost]

[ProducesResponseType(typeof(Employee), StatusCodes.Status200OK)]

public ActionResult<Employee> Create([FromBody] Employee employee)

{

employee.Id = \_employees.Max(e => e.Id) + 1;

\_employees.Add(employee);

return Ok(employee);

}

[HttpPut("{id}")]

[ProducesResponseType(typeof(Employee), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> Update(int id, [FromBody] Employee updatedEmployee)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

var existingEmployee = \_employees.FirstOrDefault(e => e.Id == id);

if (existingEmployee == null)

{

return BadRequest("Invalid employee id");

}

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills;

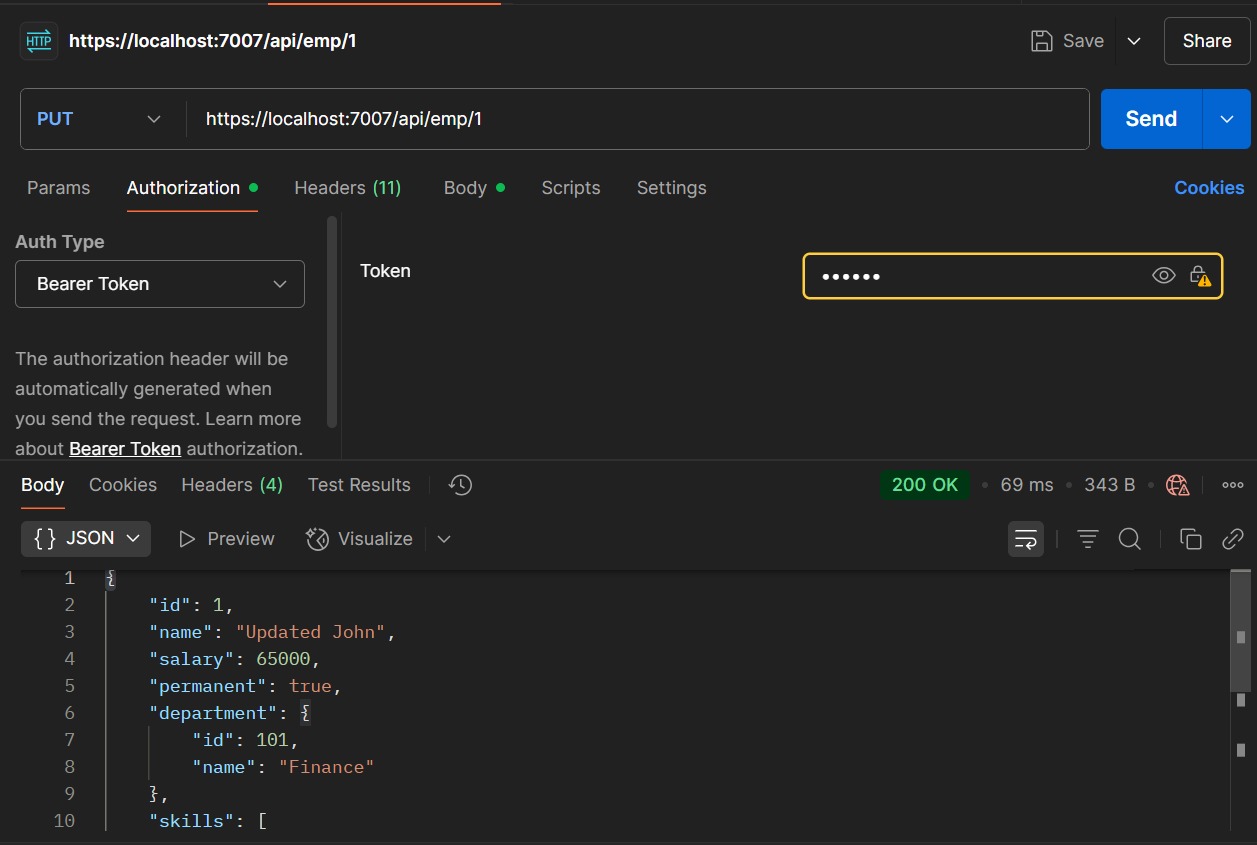
existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

return Ok(existingEmployee);

}

}

}

****

1. **Web API hands on(5)**
2. **JsonWebToken**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

string securityKey = "mysuperdupersecret";

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

builder.Services.AddControllers();

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

};

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

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

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 YourProject.Controllers

{

[ApiController]

[Route("api/auth")]

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet("token")]

public IActionResult GetToken()

{

var token = GenerateJSONWebToken(1, "Admin");

return Ok(token);

}

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);

}

}

}

1. **Check for JWT expiration**

var token = new JwtSecurityToken(

issuer: "mySystem",

audience: "myUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(2),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

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

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using WebApplication1.Models;

using WebApplication1.Models;

namespace YourProject.Controllers

{

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

[ApiController]

[Route("api/emp")]

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees = new List<Employee>

{

new Employee

{

Id = 1,

Name = "John",

Salary = 50000,

Permanent = true,

DateOfBirth = new DateTime(1990, 1, 1),

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

Skills = new List<Skill>

{

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

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

}

},

new Employee

{

Id = 2,

Name = "Alice",

Salary = 60000,

Permanent = false,

DateOfBirth = new DateTime(1992, 5, 15),

Department = new Department { Id = 102, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 3, Name = "Java" },

new Skill { Id = 4, Name = "React" }

}

}

};

[HttpGet]

[ProducesResponseType(typeof(List<Employee>), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status401Unauthorized)]

public ActionResult<List<Employee>> Get()

{

return Ok(\_employees);

}

[HttpGet("standard")]

[ProducesResponseType(typeof(Employee), StatusCodes.Status200OK)]

public ActionResult<Employee> GetStandard()

{

return Ok(\_employees.FirstOrDefault());

}

[HttpPost]

[ProducesResponseType(typeof(Employee), StatusCodes.Status201Created)]

public ActionResult<Employee> Create([FromBody] Employee employee)

{

employee.Id = \_employees.Max(e => e.Id) + 1;

\_employees.Add(employee);

return CreatedAtAction(nameof(Get), new { id = employee.Id }, employee);

}

[HttpPut("{id}")]

[ProducesResponseType(typeof(Employee), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> Update(int id, [FromBody] Employee updatedEmployee)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

var existingEmployee = \_employees.FirstOrDefault(e => e.Id == id);

if (existingEmployee == null)

{

return BadRequest("Invalid employee id");

}

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

existingEmployee.Department = updatedEmployee.Department;

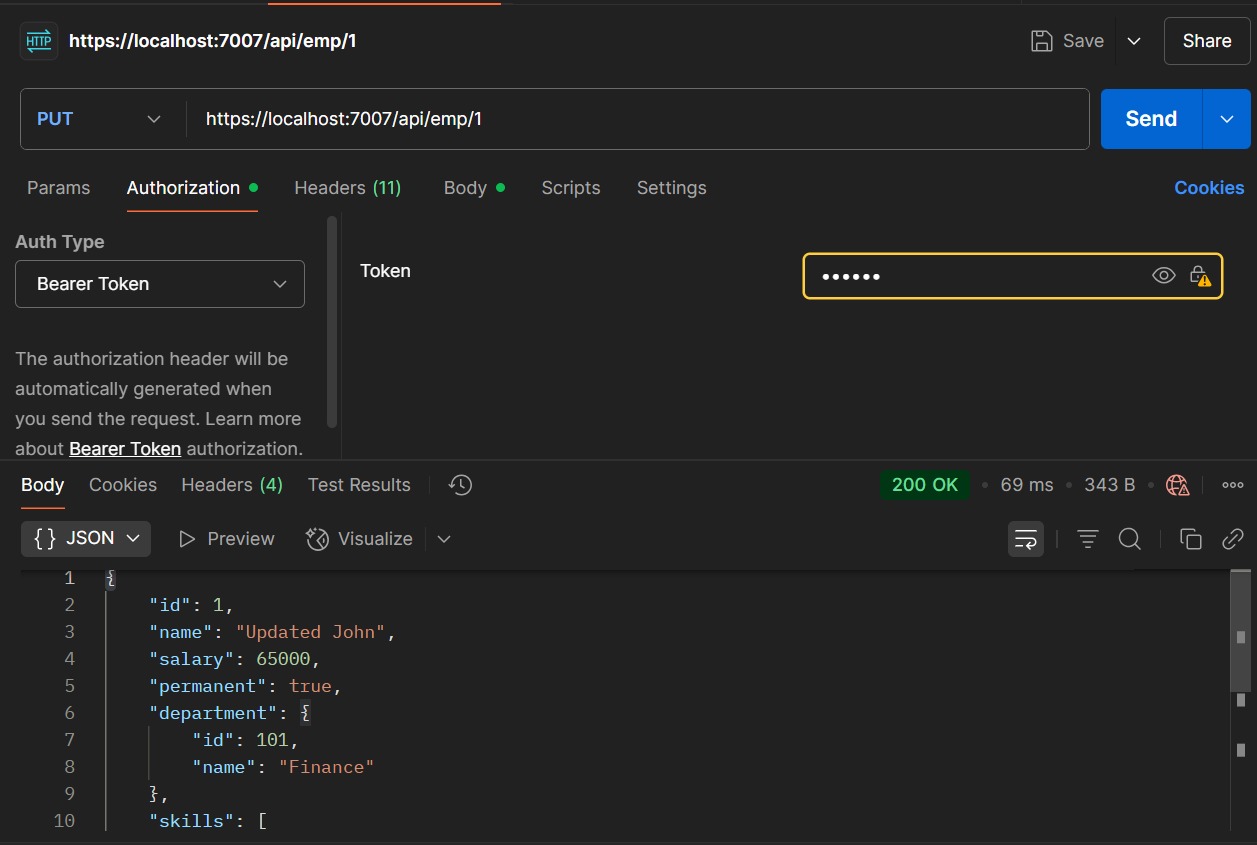
existingEmployee.Skills = updatedEmployee.Skills;

return Ok(existingEmployee);

}

}

}

****