**WEEK4 HANDSON**

**MODULE6-ASP.NET core web API**

**Ex1. WebApi\_Handson:**

**1. Explain the Concept of RESTful Web Service, Web API & Microservice**

* RESTful Web Service: REST (Representational State Transfer) is an architectural style for building web services over HTTP. A RESTful web service follows the principles of REST such as stateless communication, client-server architecture, and resource-based URIs.
* Web API: A Web API (Application Programming Interface) in ASP.NET Core is a framework for building HTTP services that can be consumed by clients like browsers, mobile apps, or other servers. It provides easy implementation of RESTful services.
* Microservice: Architectural style that structures an application as a collection of small, independent services that communicate over HTTP APIs. Each microservice focuses on a specific functionality and can be developed, deployed, and scaled independently.

**Features of REST Architecture:**

* **Representational State Transfer (REST):** REST is a design pattern used to exchange data in a distributed setup. It works by connecting a **client** and a **server** that communicate over the internet. The client and server can be built using **different platforms** like Java, .NET, or PHP. They don’t need to know each other’s internal code—just how to talk using REST. This makes it easy to build flexible and platform-independent systems.
* **Stateless:** The stateless constraint specifies that client-server communication must be stateless between requests. That means the server should not store any information, i.e., session state related to the client on the server. Each request from the client to the server must contain all the necessary information so that the server can identify the client and process that request.
* **Messages:** Communication is done using standard HTTP messages: GET, POST, PUT, DELETE.
* **Not restricted to XML**: REST APIs support multiple data formats like JSON, XML, plain text, etc. (JSON is widely used).
* **Microservice**: RESTful APIs are foundational in building microservices, where each service exposes its functionality via HTTP endpoints.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**Difference Between Web Service and Web API:**

| **Feature** | **Web Service** | **Web API** |
| --- | --- | --- |
| Protocol | SOAP-based, mostly uses HTTP, XML | RESTful, uses HTTP with JSON/XML |
| Platform Dependency | Platform-dependent (mostly .NET) | Platform-independent |
| Format | XML | JSON, XML, Text, HTML |
| Lightweight | Less (SOAP overhead) | More lightweight and fast |
| REST Support | Not designed for REST | Fully supports REST architecture |

**2.Explain What is HttpRequest & HttpResponse:**

* **HttpRequest:**  
  Represents an incoming HTTP request. It includes information like headers, method (GET, POST), body, URL, and query parameters.
* **HttpResponse:**  
  Represents the response sent back to the client. It includes status codes (200 OK, 404 Not Found), headers, and response body (e.g., JSON).

**3. List the types of Action Verbs:**

* [HttpGet] – Used to read or retrieve data.

Example: public IActionResult Get()

* [HttpPost] – Used to create new data.

Example: public IActionResult Post(Data d)

* [HttpPut] – Used to update existing data.

Example: public IActionResult Put(int id)

* [HttpDelete] – Used to delete existing data.

Example: public IActionResult Delete(int id)

**4.** **Types of HttpStatusCodes used in WebAPI:**

* 200 OK – Indicates a successful response.

Example: return Ok(data);

* 400 BadRequest – Indicates invalid input or a bad request from the client.

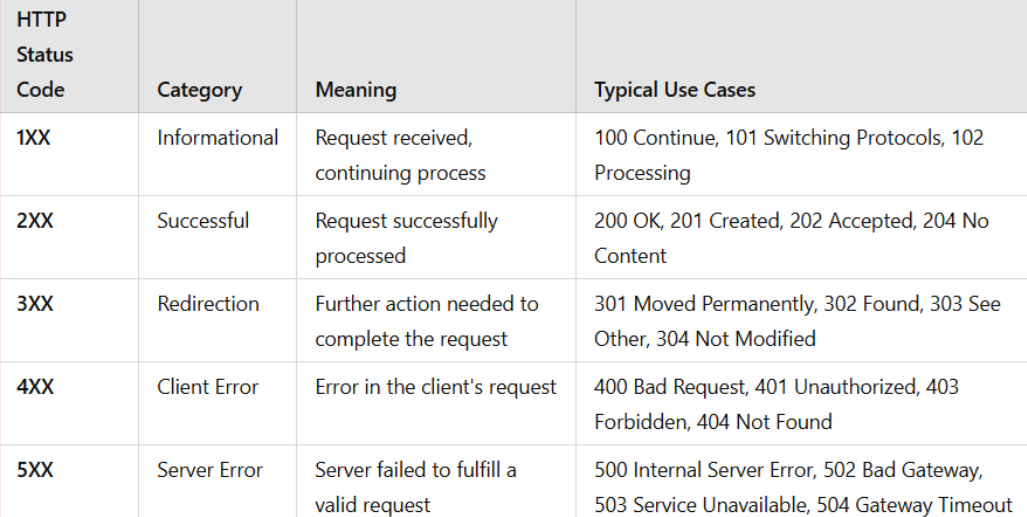
Example: return BadRequest("Invalid");

* 401 Unauthorized – Indicates that authentication has failed or the user is not authorized.

Example: return Unauthorized();

* 500 InternalServerError – Indicates a server-side error or unexpected failure.

Example: return StatusCode(500);



**5. Configuration files of WebAPI:**

1. Startup.cs
   * Configures services such as dependency injection.
   * Defines the middleware pipeline, which handles how requests and responses are processed.
2. appsettings.json
   * Stores application settings such as connection strings and API keys.
   * These settings can be accessed throughout the application using the built-in configuration system.
3. launchSettings.json
   * Contains environment-specific settings for development.
   * Defines how the application launches, the URL it uses, and any environment variables like development mode.

4.Route.config

* Used to define custom routing patterns for Web API endpoints.
* Helps map URLs to corresponding controller actions.

5.WebApi.config

* Registers Web API routes and configures settings like response format (JSON, XML).
* Helps set up how requests are handled across the Web API application.

**First Web Api using .Net core**

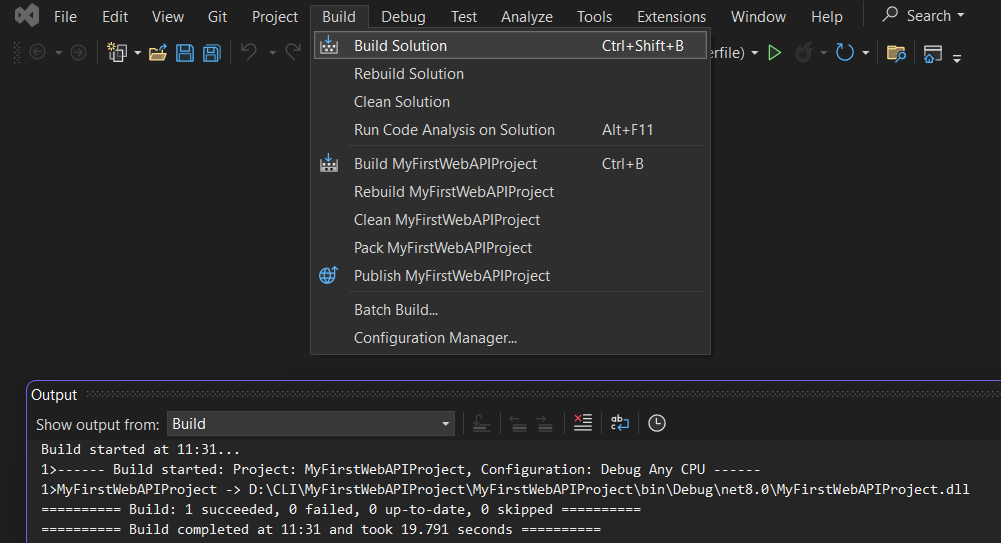
**Step1: open visual studio 2022, create new project**

**Step2: select the “ASP.NET Core Web API” template**

**Step3:** name the project (**MyFirstWebAPIProject**) and the location where you want to create the project.

Step4: create

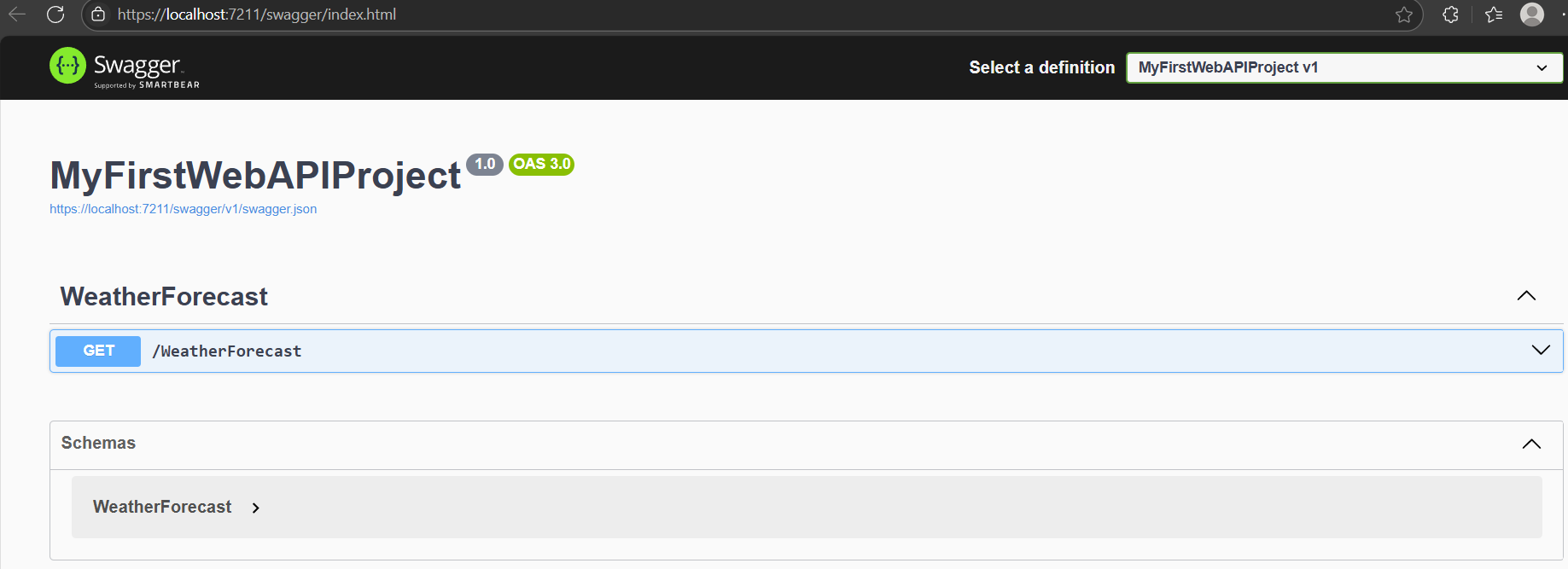
Step5: go to build, select build solution

****

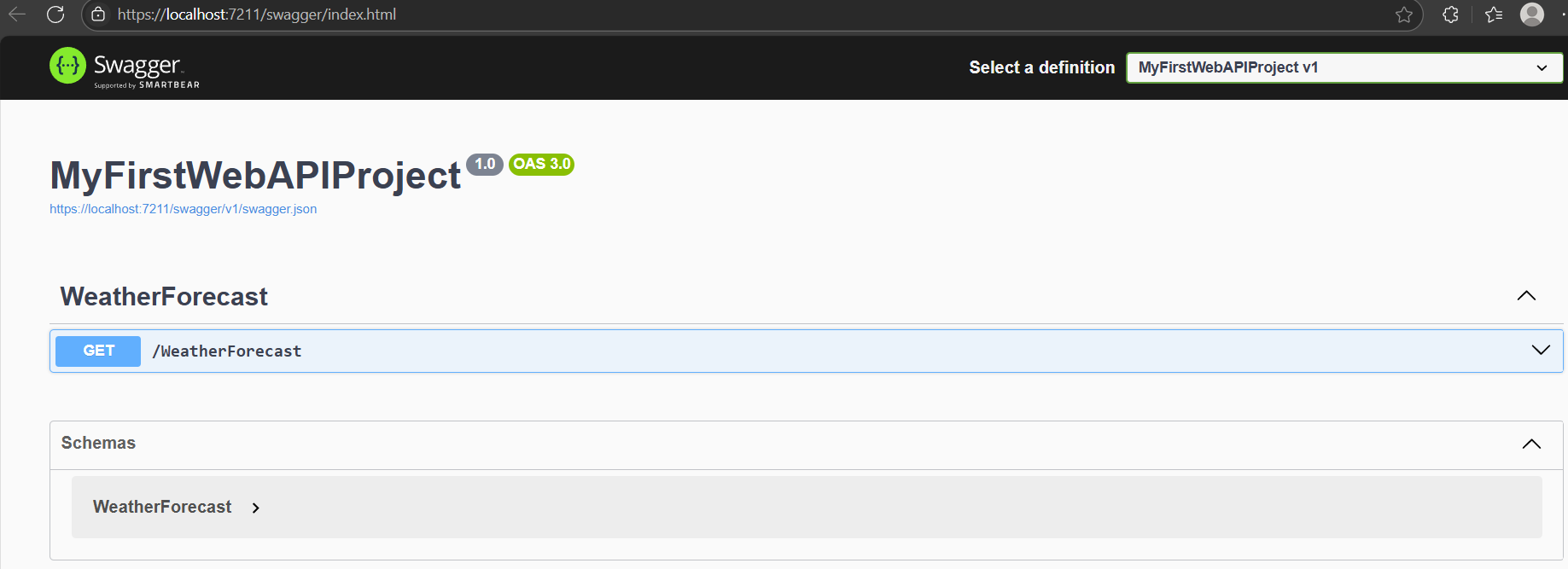
Step6: run the application in “http”



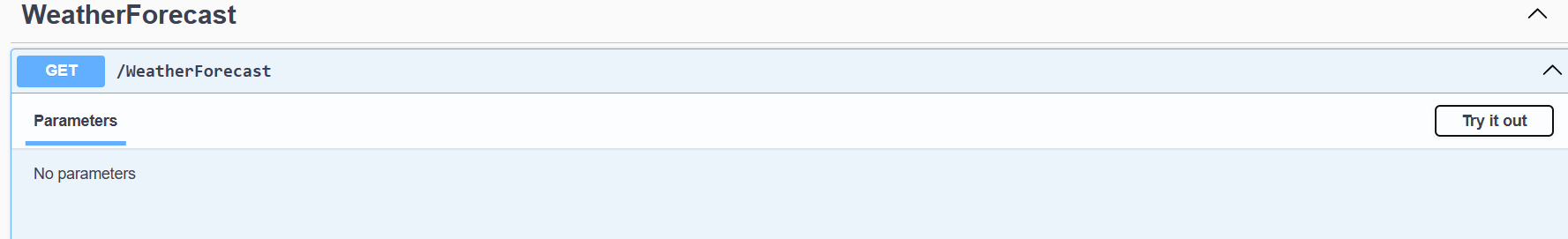
Step7:after run it will open browser and display like below



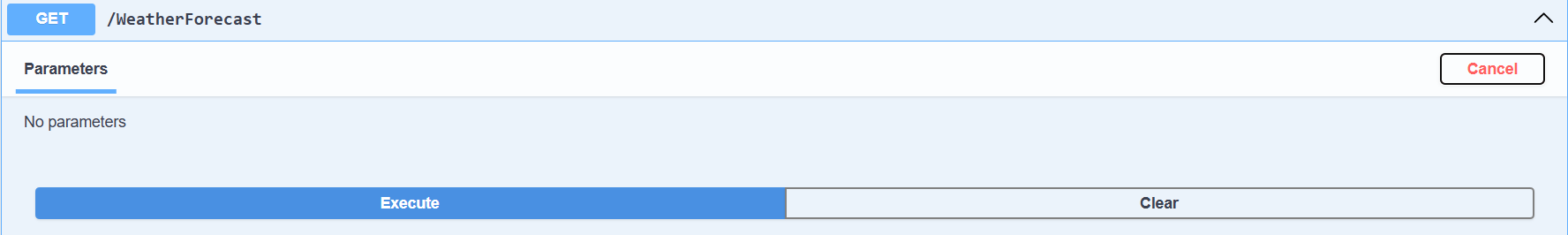
Step8: click the below mentioned area



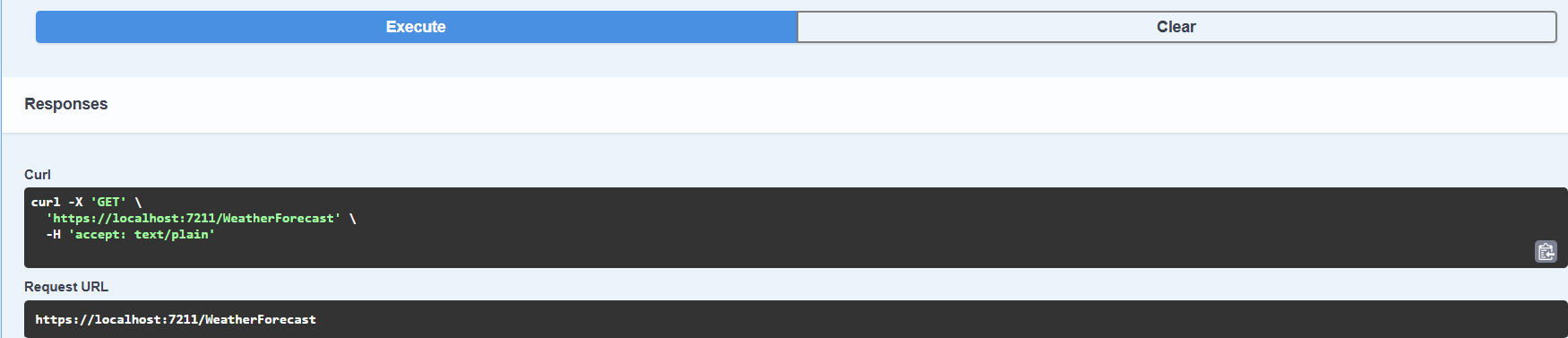
Step9: click try it out

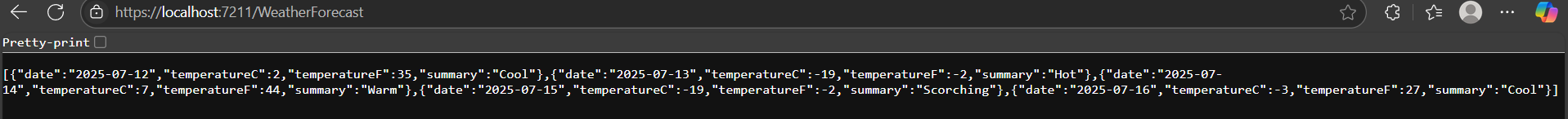


Step10: click excute



Step11: use the Request URL in browser

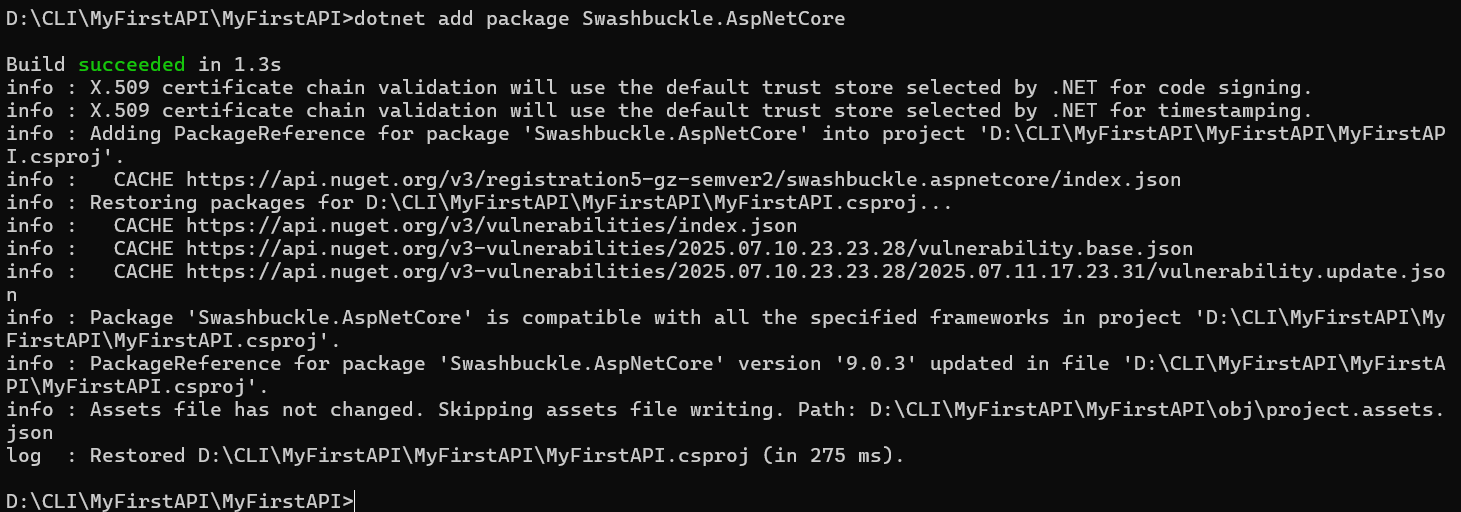


Output: call the Request URL in any of your browsers, and you will get the response as expected

**EX2. WebApi\_Handson**

Step1:install Swashbuckle.AspNetCore Nuget package by executing the below command in project file’s cmd

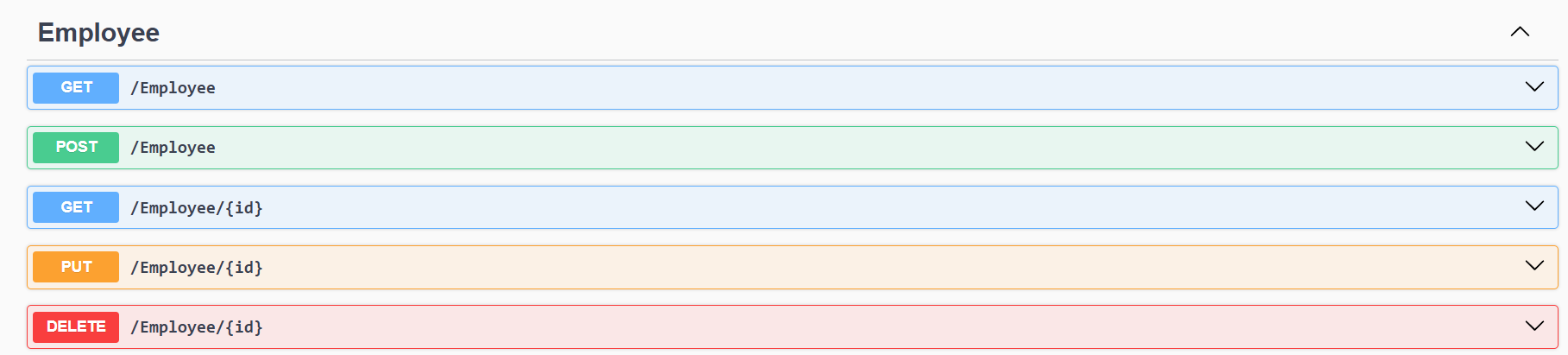
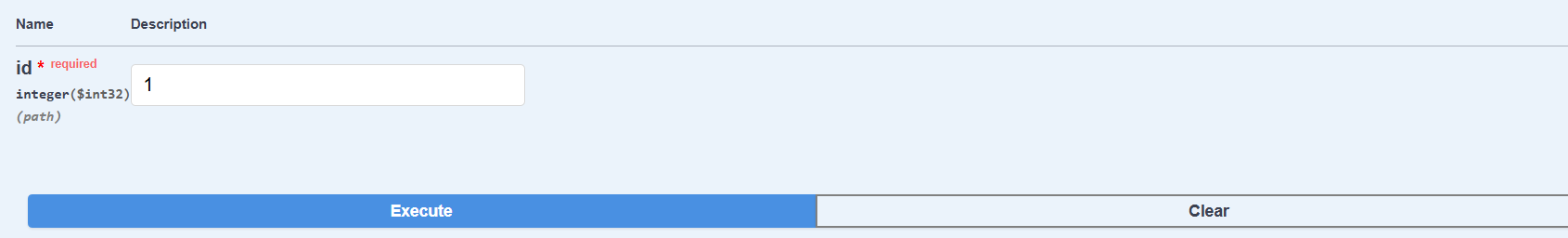
“dotnet add package Swashbuckle.AspNetCore”

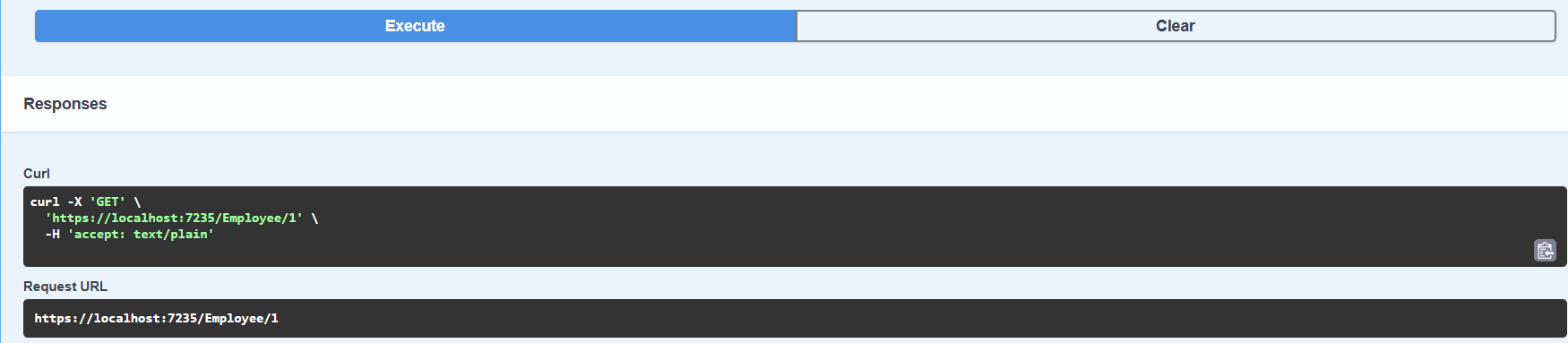


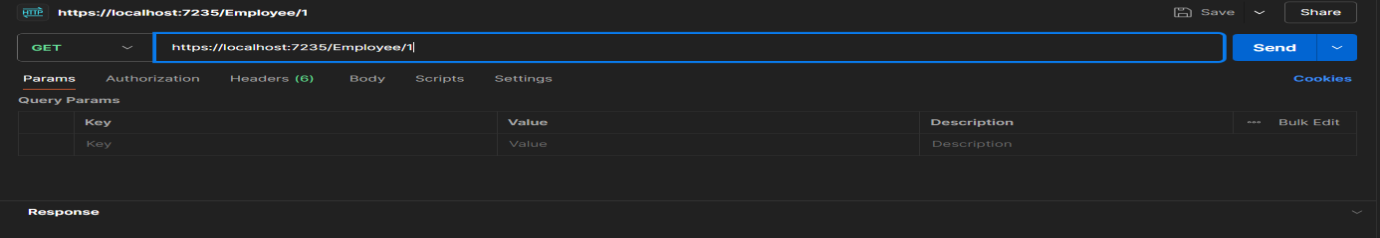
Step2: alter program.cs file’s “services.AddSwaggerGen” and “app.UseSwaggerUI” similar to below



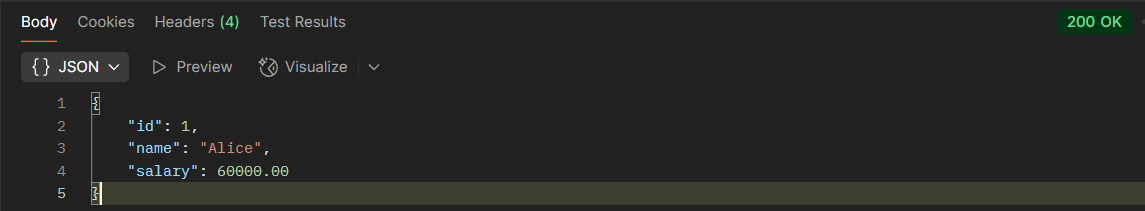
Step3: now run the project in “https” and give “try it out” to “execute” and give the value for id get the request url



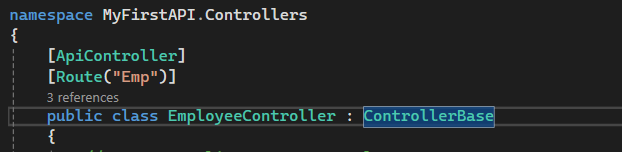


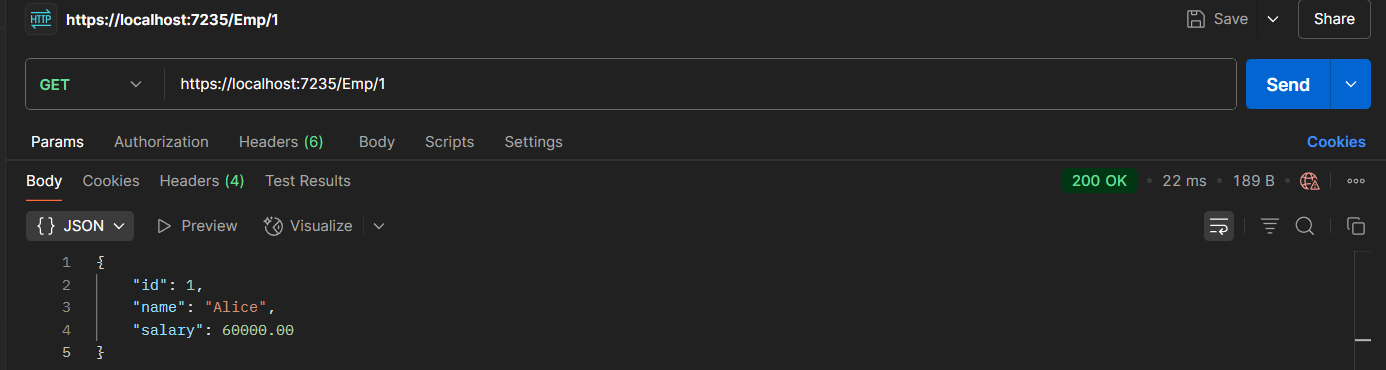
Step4: now give the url in postmen “GET”

Output: now check the value inside the response similar to the data expected

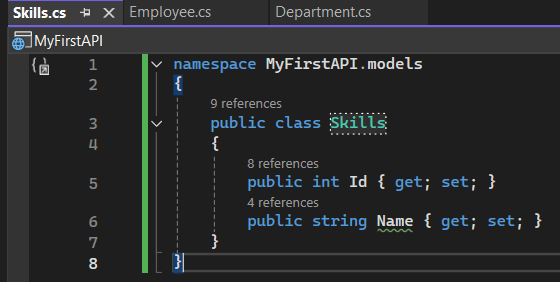


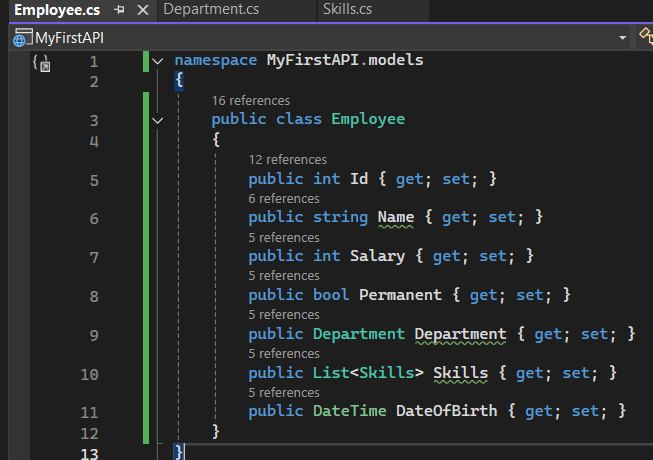
Modify the Controller name in the Route attribute of the Employee controller to ‘Emp’ and check its access thru POSTMAN

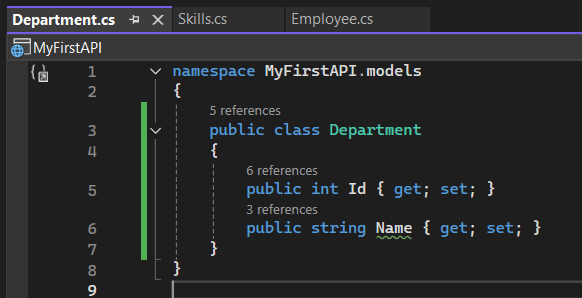




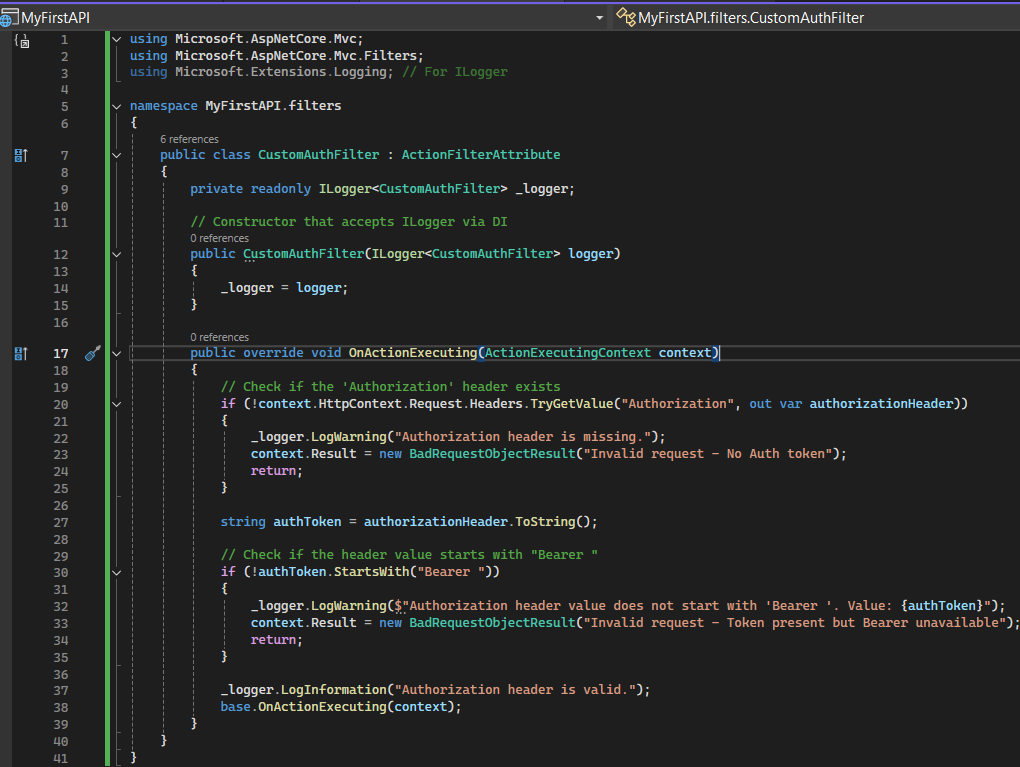
EX3: Web\_API Handson

Step1: create Skills.cs, Department.cs and Employee.cs and write the codes given below





Step2: create “filters” folder and add CustomAuthFilter.cs and CustomExceptionFilter.cs





Step3: update the EmployeeController.cs similar to below provided code

using Microsoft.AspNetCore.Mvc;

using MyFirstAPI.filters;

using MyFirstAPI.models;

using System;

using System.Collections.Generic;

using System.Linq;

// Assuming Department and Skill classes are in the root namespace or a Models folder.

// If in a Models folder, you'd need: using MyFirstAPI.Models;

namespace MyFirstAPI.Controllers

{

[ApiController]

[Route("Emp")]

[ServiceFilter(typeof(CustomAuthFilter))]

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees = new List<Employee>();

private readonly ILogger<EmployeeController> \_logger;

public EmployeeController(ILogger<EmployeeController> logger)

{

\_logger = logger;

\_employees = GetStandardEmployeeList(); // Data is initialized here

}

private List<Employee> GetStandardEmployeeList()

{

var departments = new List<Department>

{

new Department { Id = 1, Name = "Human Resources" },

new Department { Id = 2, Name = "Engineering" },

new Department { Id = 3, Name = "Sales" }

};

var skills = new List<Skills> // Assuming Skill class is singular

{

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

new Skills { Id = 2, Name = "JavaScript" },

new Skills { Id = 3, Name = "SQL" },

new Skills { Id = 4, Name = "Agile" }

};

var employeeList = new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice Smith",

Salary = 60000,

Permanent = true,

Department = departments.FirstOrDefault(d => d.Id == 2),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 1), skills.FirstOrDefault(s => s.Id == 4) },

DateOfBirth = new DateTime(1990, 5, 15)

},

new Employee

{

Id = 2,

Name = "Bob Johnson",

Salary = 75000,

Permanent = true,

Department = departments.FirstOrDefault(d => d.Id == 3),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 2) },

DateOfBirth = new DateTime(1988, 11, 22)

},

new Employee

{

Id = 3,

Name = "Charlie Brown",

Salary = 55000,

Permanent = false,

Department = departments.FirstOrDefault(d => d.Id == 1),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 3) },

DateOfBirth = new DateTime(1995, 1, 10)

}

};

\_logger.LogInformation("Initialized standard employee list.");

return employeeList;

}

// GET Emp

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(IEnumerable<Employee>))]

// Add ProducesResponseType for 500 Internal Server Error

[ProducesResponseType(StatusCodes.Status500InternalServerError, Type = typeof(ProblemDetails))]

public ActionResult<IEnumerable<Employee>> Get()

{

\_logger.LogInformation("Getting all employees.");

// Simulate an error to test the exception filter

throw new InvalidOperationException("Simulated error: Could not fetch employee data!");

// This line will not be reached due to the exception

// return Ok(\_employees);

}

// GET Emp/{id}

[HttpGet("{id}")]

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

[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<Employee> Get(int id)

{

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

if (employee == null)

{

\_logger.LogWarning($"Employee with ID {id} not found.");

return NotFound();

}

\_logger.LogInformation($"Getting employee with ID {id}.");

return Ok(employee);

}

// GET Emp/Standard

[HttpGet("Standard")]

public ActionResult<Employee> GetStandrad()

{

if (\_employees.Any())

{

\_logger.LogInformation("Getting a standard employee.");

return Ok(\_employees.First());

}

else

{

\_logger.LogWarning("No employees found to return a standard employee.");

return NotFound("No employee data available.");

}

}

// POST Emp

[HttpPost]

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

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status409Conflict)]

public ActionResult<Employee> Post(Employee newEmployee)

{

if (!ModelState.IsValid)

{

\_logger.LogError("Invalid model state for creating employee.");

return BadRequest(ModelState);

}

if (\_employees.Any(e => e.Id == newEmployee.Id))

{

\_logger.LogWarning($"Employee with ID {newEmployee.Id} already exists.");

return Conflict($"Employee with ID {newEmployee.Id} already exists.");

}

\_employees.Add(newEmployee);

\_logger.LogInformation($"Employee '{newEmployee.Name}' created with ID {newEmployee.Id}.");

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

}

// PUT Emp/{id}

[HttpPut("{id}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult Put(int id, Employee updatedEmployee)

{

if (!ModelState.IsValid)

{

\_logger.LogError($"Invalid model state for updating employee with ID {id}.");

return BadRequest(ModelState);

}

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

if (existingEmployee == null)

{

\_logger.LogWarning($"Employee with ID {id} not found for update.");

return NotFound();

}

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills;

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

\_logger.LogInformation($"Employee with ID {id} updated successfully.");

return NoContent();

}

// DELETE Emp/{id}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult Delete(int id)

{

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

if (employeeToDelete == null)

{

\_logger.LogWarning($"Employee with ID {id} not found for deletion.");

return NotFound();

}

\_employees.Remove(employeeToDelete);

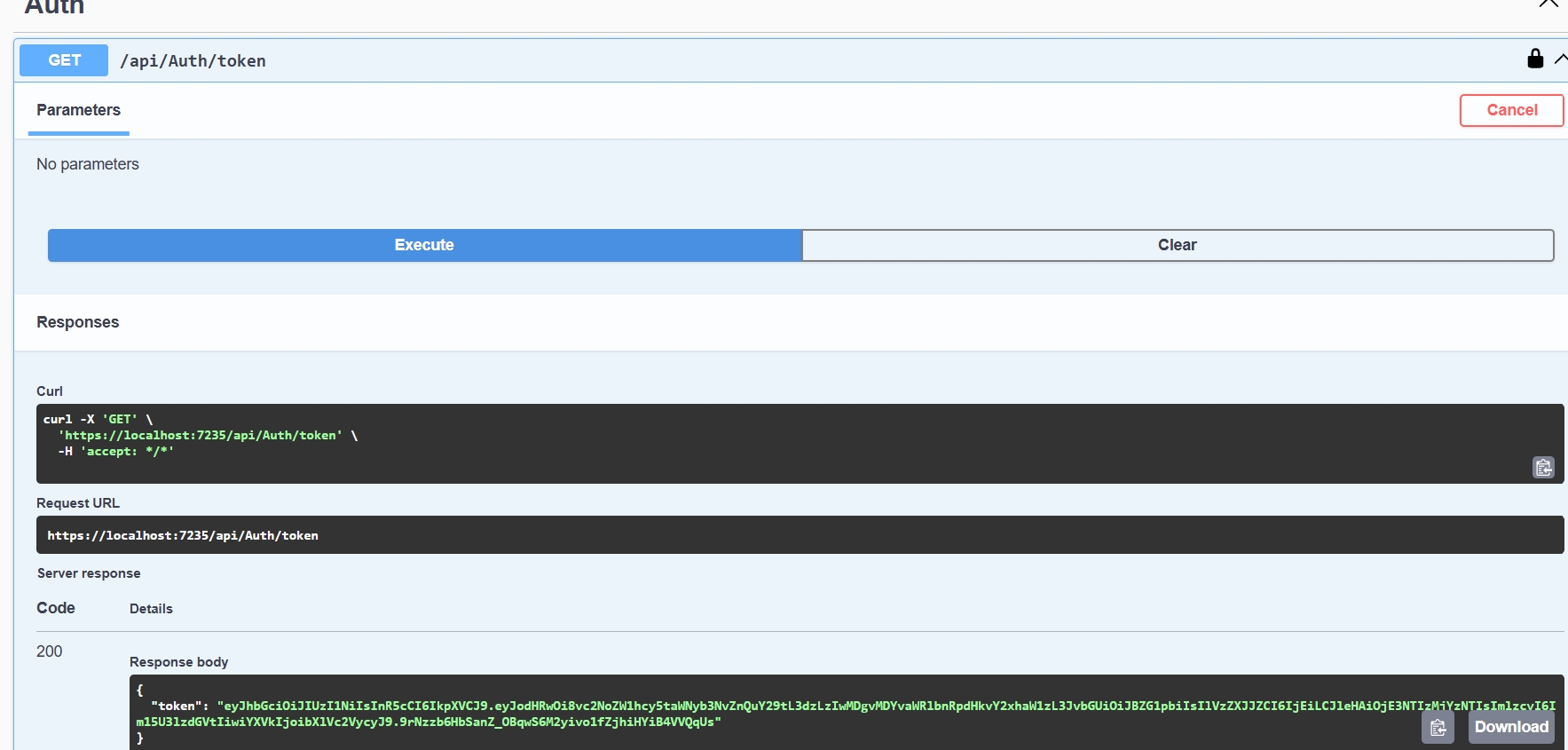
\_logger.LogInformation($"Employee with ID {id} deleted successfully.");

return NoContent();

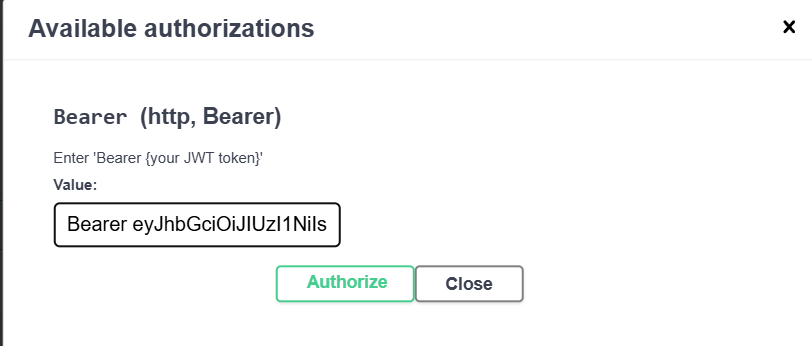
}

}

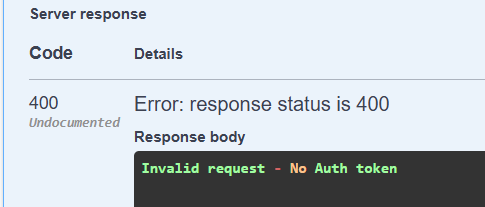
}

Step4: now execute “Auth GET/api/auth/token” and “try it out” and copy token value

Step5: use the token in “Authorize” section as Bearer <key>



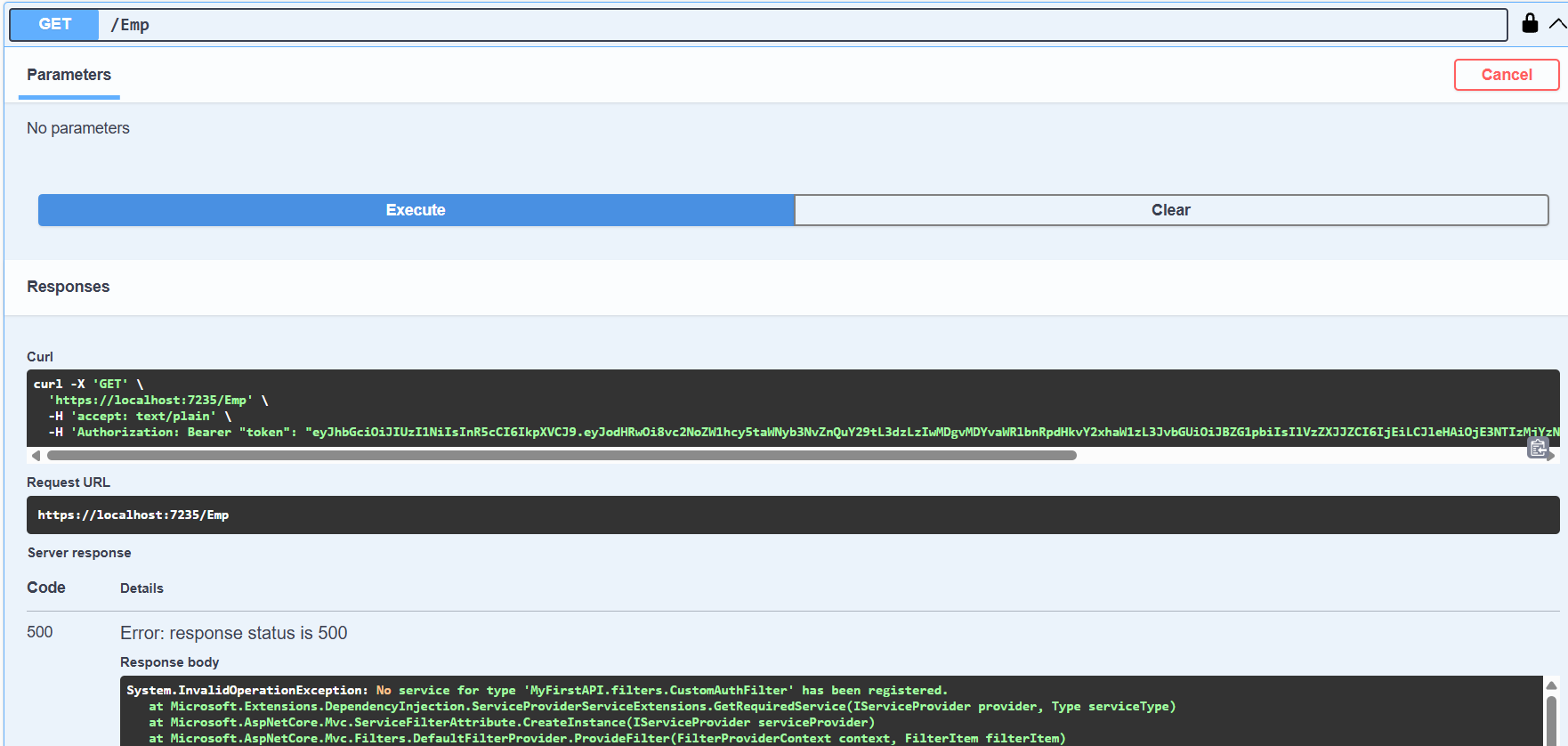
Output1: executing without Authorize will get code 400 with “Invalid requeset - No Auth token”



Output2: improper Authorization i.e entering Token without “Bearer ”



Output3: Authorize properly by “Bearer <token>” execute to get code 500 Exception



Ex4: Web\_API Handson

Step1: update EmployeeController similar to below to hndle invalid input values to be handled in “PUT” such as empId 0,negative numbers and number other than existing

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.Extensions.Logging;

using MyFirstAPI.filters;

using MyFirstAPI.models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace MyFirstAPI.Controllers

{

[ApiController]

[Route("Emp")]

[ServiceFilter(typeof(CustomAuthFilter))]

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees = new List<Employee>();

private readonly ILogger<EmployeeController> \_logger;

public EmployeeController(ILogger<EmployeeController> logger)

{

\_logger = logger;

if (\_employees.Count == 0)

{

\_employees = GetStandardEmployeeList();

}

}

private List<Employee> GetStandardEmployeeList()

{

var departments = new List<Department>

{

new Department { Id = 1, Name = "Human Resources" },

new Department { Id = 2, Name = "Engineering" },

new Department { Id = 3, Name = "Sales" }

};

var skills = new List<Skills>

{

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

new Skills { Id = 2, Name = "JavaScript" },

new Skills { Id = 3, Name = "SQL" },

new Skills { Id = 4, Name = "Agile" }

};

var employeeList = new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice Smith",

Salary = 60000,

Permanent = true,

Department = departments.FirstOrDefault(d => d.Id == 2),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 1), skills.FirstOrDefault(s => s.Id == 4) },

DateOfBirth = new DateTime(1990, 5, 15)

},

new Employee

{

Id = 2,

Name = "Bob Johnson",

Salary = 75000,

Permanent = true,

Department = departments.FirstOrDefault(d => d.Id == 3),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 2) },

DateOfBirth = new DateTime(1988, 11, 22)

},

new Employee

{

Id = 3,

Name = "Charlie Brown",

Salary = 55000,

Permanent = false,

Department = departments.FirstOrDefault(d => d.Id == 1),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 3) },

DateOfBirth = new DateTime(1995, 1, 10)

}

};

\_logger.LogInformation("Initialized standard employee list.");

return employeeList;

}

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(IEnumerable<Employee>))]

[ProducesResponseType(StatusCodes.Status500InternalServerError, Type = typeof(ProblemDetails))]

public ActionResult<IEnumerable<Employee>> Get()

{

\_logger.LogInformation("Getting all employees.");

return Ok(\_employees);

}

[HttpGet("{id}")]

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

[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<Employee> Get(int id)

{

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

if (employee == null)

{

\_logger.LogWarning($"Employee with ID {id} not found.");

return NotFound();

}

\_logger.LogInformation($"Getting employee with ID {id}.");

return Ok(employee);

}

[HttpPost]

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

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status409Conflict)]

public ActionResult<Employee> Post(Employee newEmployee)

{

if (!ModelState.IsValid)

{

\_logger.LogError("Invalid model state for creating employee.");

return BadRequest(ModelState);

}

if (\_employees.Any(e => e.Id == newEmployee.Id))

{

\_logger.LogWarning($"Employee with ID {newEmployee.Id} already exists.");

return Conflict($"Employee with ID {newEmployee.Id} already exists.");

}

\_employees.Add(newEmployee);

\_logger.LogInformation($"Employee '{newEmployee.Name}' created with ID {newEmployee.Id}.");

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

}

[HttpPut("{id}")]

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

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> Put(int id, Employee updatedEmployee)

{

if (id <= 0)

{

\_logger.LogWarning($"Attempted to update employee with invalid ID: {id}");

return BadRequest("Invalid employee id");

}

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

if (existingEmployee == null)

{

\_logger.LogWarning($"Employee with ID {id} not found for update.");

return BadRequest("Invalid employee id");

}

if (!ModelState.IsValid)

{

\_logger.LogError($"Invalid model state for updating employee with ID {id}.");

return BadRequest(ModelState);

}

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills ?? new List<Skills>();

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

\_logger.LogInformation($"Employee with ID {id} updated successfully.");

return Ok(existingEmployee);

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult Delete(int id)

{

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

if (employeeToDelete == null)

{

\_logger.LogWarning($"Employee with ID {id} not found for deletion.");

return NotFound();

}

\_employees.Remove(employeeToDelete);

\_logger.LogInformation($"Employee with ID {id} deleted successfully.");

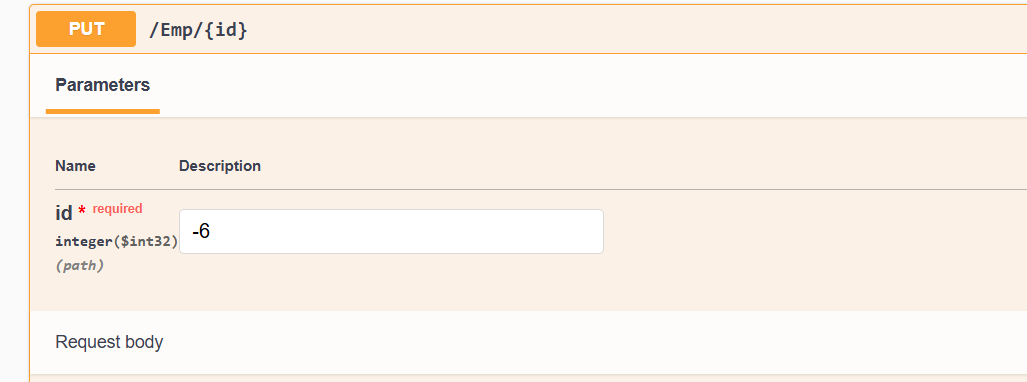
return NoContent();

}

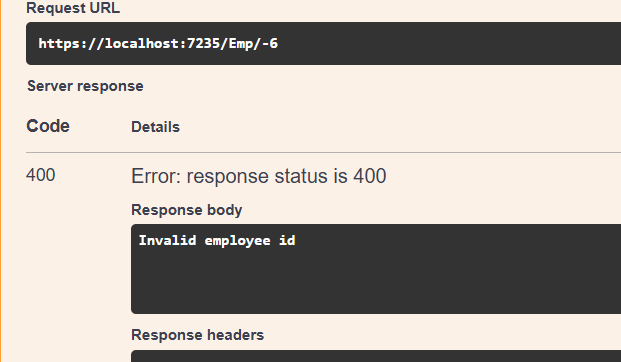
}

}

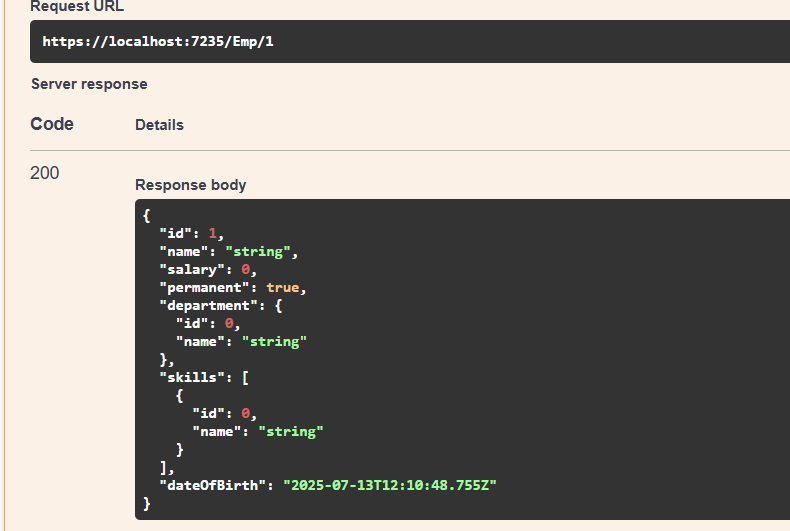
Step2: Now run the application and give input inside PUT as some invalid values and Execute to see a “BAD REQUEST” code 400



Output: while input id is negative (or) not from list



Output: while input id is valid i.e from available list hence got code 200



EX5: Web\_API Handson

Step1:Updated Program.cs with JWT Configuration similar to below

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using MyFirstAPI.filters;

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Linq;

using System.Text;

namespace MyFirstAPI

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers(options =>

{

options.Filters.Add<CustomExceptionFilter>();

});

builder.Services.AddScoped<CustomAuthFilter>();

builder.Services.AddScoped<CustomExceptionFilter>();

var jwtSettings = builder.Configuration.GetSection("Jwt");

var securityKeyBytes = Encoding.UTF8.GetBytes(jwtSettings["Key"]);

var symmetricSecurityKey = new SymmetricSecurityKey(securityKeyBytes);

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, options =>

{

options.RequireHttpsMetadata = false;

options.SaveToken = true;

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidIssuer = jwtSettings["Issuer"],

ValidateAudience = true,

ValidAudience = jwtSettings["Audience"],

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

IssuerSigningKey = symmetricSecurityKey,

ClockSkew = TimeSpan.Zero

};

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "MyFirstAPI",

Version = "v1",

Description = "API for Employees and JWT Authentication.",

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")

}

});

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Description = "JWT Authorization header using the Bearer scheme. \r\n Enter 'Bearer [your token]' below.",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new List<string>()

}

});

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

}

}

}

Step2: 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.

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.AspNetCore.Authorization;

using Microsoft.Extensions.Configuration;

namespace MyFirstAPI.Controllers

{

[ApiController]

[Route("Emp")]

[AllowAnonymous]

public class AuthController : ControllerBase

{

private readonly IConfiguration \_configuration;

public AuthController(IConfiguration configuration)

{

\_configuration = configuration;

}

[HttpGet("token")]

public IActionResult GetToken()

{

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

return Ok(new { token });

}

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKeyBytes = Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]);

var issuer = \_configuration["Jwt:Issuer"];

var audience = \_configuration["Jwt:Audience"];

var securityKey = new SymmetricSecurityKey(securityKeyBytes);

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

var claims = new List<Claim>

{

new Claim(JwtRegisteredClaimNames.Sub, userId.ToString()),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),

new Claim(ClaimTypes.Role, userRole),

};

var token = new JwtSecurityToken(

issuer: issuer,

audience: audience,

claims: claims,

expires: DateTime.Now.AddMinutes(30),

signingCredentials: credentials

);

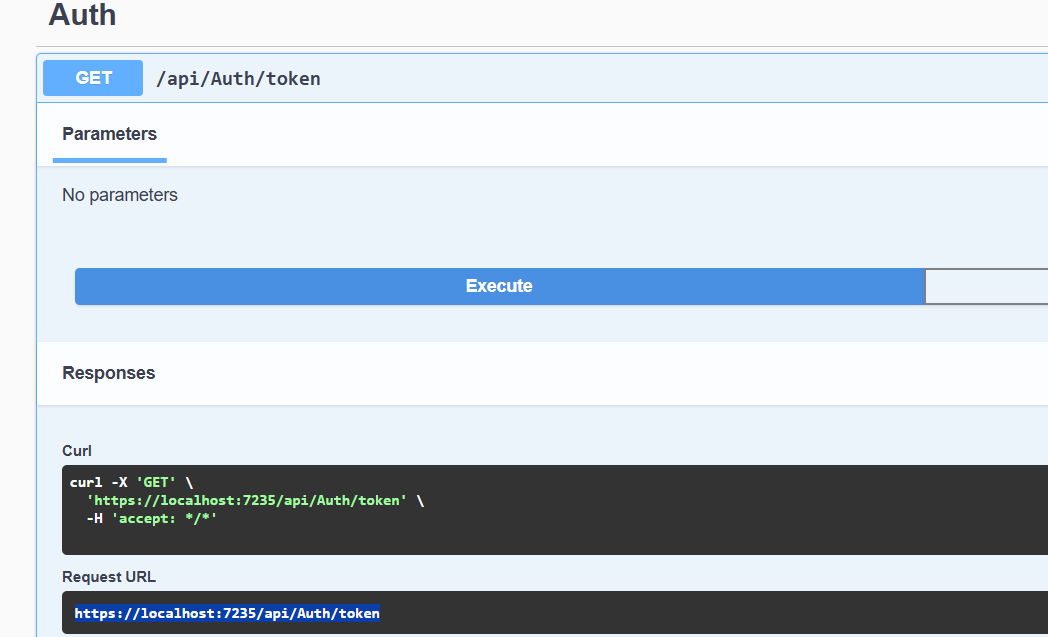
return new JwtSecurityTokenHandler().WriteToken(token);

}

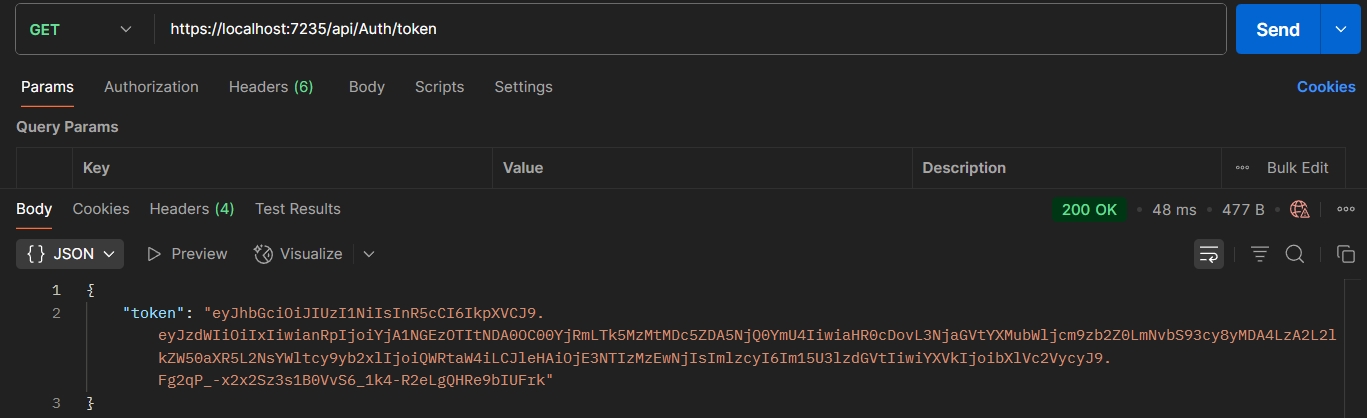
}

}

Step3:Now run the application and get the URL from Auth “GET api/auth/Token”



Step4:now use the url in POSTMAN “GET” to get the token as Expected



Step5: Modify the duration for ‘expires’ attribute to 2 minutes

var token = new JwtSecurityToken(

issuer: issuer,

audience: audience,

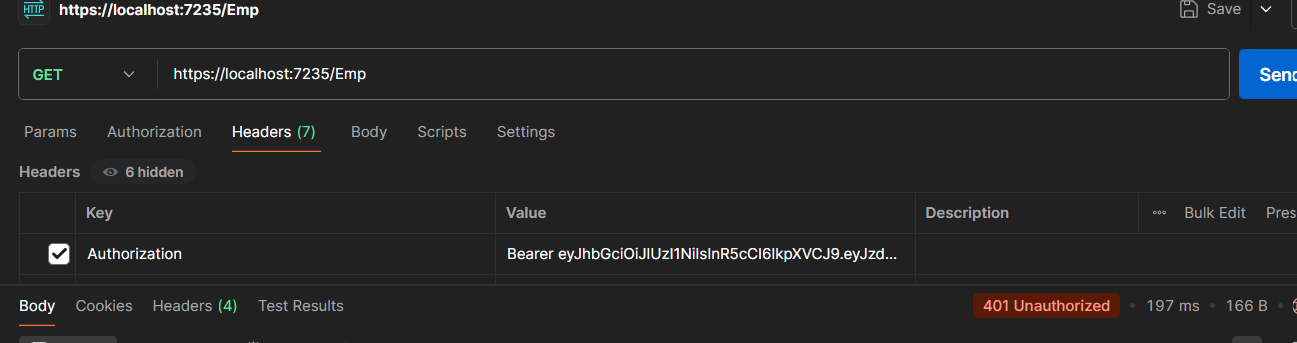
claims: claims,

expires: DateTime.Now.AddMinutes(2),

signingCredentials: credentials

);

Output: use the token in header of postman with key-Authentication and value-Bearer <Token> after 2 min to get “code 401 Unauthorized” by giving the URL from EmpGet



Step6: **Add the roles to be authorized in the Authorize attribute. By updating code in program.cs,CustomAuth.cs,EmployeeController.cs,AuthController.cs**

* **Change code in CustomAuth.cs by below**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace MyFirstAPI.filters

{

public class CustomAuthFilter : ActionFilterAttribute

{

private readonly ILogger<CustomAuthFilter> \_logger;

private readonly IConfiguration \_configuration;

public CustomAuthFilter(ILogger<CustomAuthFilter> logger, IConfiguration configuration)

{

\_logger = logger;

\_configuration = configuration;

}

public override void OnActionExecuting(ActionExecutingContext context)

{

if (!context.HttpContext.Request.Headers.TryGetValue("Authorization", out var authorizationHeader))

{

\_logger.LogWarning("Authorization header is missing.");

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

return;

}

string authorizationHeaderValue = authorizationHeader.ToString();

if (!authorizationHeaderValue.StartsWith("Bearer "))

{

\_logger.LogWarning($"Authorization header value does not start with 'Bearer '. Value: {authorizationHeaderValue}");

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

return;

}

string token = authorizationHeaderValue.Substring("Bearer ".Length);

var tokenHandler = new JwtSecurityTokenHandler();

var validationParameters = GetTokenValidationParameters();

try

{

SecurityToken validatedToken;

var principal = tokenHandler.ValidateToken(token, validationParameters, out validatedToken);

var roles = principal.Claims.Where(c => c.Type == ClaimTypes.Role).Select(c => c.Value).ToList();

bool hasAdminRole = roles.Contains("Admin");

bool hasPocRole = roles.Contains("POC");

var isEmployeeGetRequest = context.ActionDescriptor.RouteValues["controller"] == "Employee" &&

context.ActionDescriptor.RouteValues["action"] == "Get";

if (isEmployeeGetRequest)

{

if (!hasPocRole)

{

\_logger.LogWarning("Access denied for Employee.Get(): 'POC' role is required but not present in the token.");

context.Result = new UnauthorizedObjectResult("Access denied: 'POC' role is required.");

return;

}

if (hasPocRole && hasAdminRole)

{

\_logger.LogInformation("Access granted for Employee.Get(): Required roles 'Admin' and 'POC' are present.");

}

else if (hasPocRole && !hasAdminRole)

{

\_logger.LogWarning("Access denied for Employee.Get(): 'Admin' role is required along with 'POC' but not present in the token.");

context.Result = new UnauthorizedObjectResult("Access denied: 'Admin' role is required.");

return;

}

}

}

catch (SecurityTokenValidationException ex)

{

\_logger.LogError($"Token validation failed: {ex.Message}");

context.Result = new UnauthorizedObjectResult("Invalid token.");

return;

}

catch (Exception ex)

{

\_logger.LogError($"An error occurred during token processing: {ex.Message}");

context.Result = new StatusCodeResult(StatusCodes.Status500InternalServerError);

return;

}

base.OnActionExecuting(context);

}

private TokenValidationParameters GetTokenValidationParameters()

{

var securityKeyBytes = Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]);

var issuer = \_configuration["Jwt:Issuer"];

var audience = \_configuration["Jwt:Audience"];

return new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(securityKeyBytes),

ValidateIssuer = true,

ValidIssuer = issuer,

ValidateAudience = true,

ValidAudience = audience,

ClockSkew = TimeSpan.FromMinutes(5)

};

}

}

}

* Change code in EmployeeController.cs by below

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters; // Needed for ServiceFilter

using Microsoft.Extensions.Logging;

using MyFirstAPI.filters; // Assuming CustomAuthFilter is in this namespace

using MyFirstAPI.models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace MyFirstAPI.Controllers

{

[ApiController]

[Route("Emp")]

[ServiceFilter(typeof(CustomAuthFilter))]

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees = new List<Employee>();

private readonly ILogger<EmployeeController> \_logger;

public EmployeeController(ILogger<EmployeeController> logger)

{

\_logger = logger;

if (\_employees.Count == 0)

{

\_employees = GetStandardEmployeeList();

}

}

private List<Employee> GetStandardEmployeeList()

{

var departments = new List<Department>

{

new Department { Id = 1, Name = "Human Resources" },

new Department { Id = 2, Name = "Engineering" },

new Department { Id = 3, Name = "Sales" }

};

var skills = new List<Skills>

{

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

new Skills { Id = 2, Name = "JavaScript" },

new Skills { Id = 3, Name = "SQL" },

new Skills { Id = 4, Name = "Agile" }

};

var employeeList = new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice Smith",

Salary = 60000,

Permanent = true,

Department = departments.FirstOrDefault(d => d.Id == 2),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 1), skills.FirstOrDefault(s => s.Id == 4) },

DateOfBirth = new DateTime(1990, 5, 15)

},

new Employee

{

Id = 2,

Name = "Bob Johnson",

Salary = 75000,

Permanent = true,

Department = departments.FirstOrDefault(d => d.Id == 3),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 2) },

DateOfBirth = new DateTime(1988, 11, 22)

},

new Employee

{

Id = 3,

Name = "Charlie Brown",

Salary = 55000,

Permanent = false,

Department = departments.FirstOrDefault(d => d.Id == 1),

Skills = new List<Skills> { skills.FirstOrDefault(s => s.Id == 3) },

DateOfBirth = new DateTime(1995, 1, 10)

}

};

\_logger.LogInformation("Initialized standard employee list.");

return employeeList;

}

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(IEnumerable<Employee>))]

[ProducesResponseType(StatusCodes.Status500InternalServerError, Type = typeof(ProblemDetails))]

public ActionResult<IEnumerable<Employee>> Get()

{

\_logger.LogInformation("Getting all employees.");

return Ok(\_employees);

}

[HttpGet("{id}")]

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

[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<Employee> Get(int id)

{

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

if (employee == null)

{

\_logger.LogWarning($"Employee with ID {id} not found.");

return NotFound();

}

\_logger.LogInformation($"Getting employee with ID {id}.");

return Ok(employee);

}

[HttpPost]

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

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status409Conflict)]

public ActionResult<Employee> Post(Employee newEmployee)

{

if (!ModelState.IsValid)

{

\_logger.LogError("Invalid model state for creating employee.");

return BadRequest(ModelState);

}

if (\_employees.Any(e => e.Id == newEmployee.Id))

{

\_logger.LogWarning($"Employee with ID {newEmployee.Id} already exists.");

return Conflict($"Employee with ID {newEmployee.Id} already exists.");

}

\_employees.Add(newEmployee);

\_logger.LogInformation($"Employee '{newEmployee.Name}' created with ID {newEmployee.Id}.");

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

}

[HttpPut("{id}")]

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

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> Put(int id, Employee updatedEmployee)

{

if (id <= 0)

{

\_logger.LogWarning($"Attempted to update employee with invalid ID: {id}");

return BadRequest("Invalid employee id");

}

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

if (existingEmployee == null)

{

\_logger.LogWarning($"Employee with ID {id} not found for update.");

return BadRequest("Invalid employee id");

}

if (!ModelState.IsValid)

{

\_logger.LogError($"Invalid model state for updating employee with ID {id}.");

return BadRequest(ModelState);

}

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills ?? new List<Skills>();

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

\_logger.LogInformation($"Employee with ID {id} updated successfully.");

return Ok(existingEmployee);

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult Delete(int id)

{

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

if (employeeToDelete == null)

{

\_logger.LogWarning($"Employee with ID {id} not found for deletion.");

return NotFound();

}

\_employees.Remove(employeeToDelete);

\_logger.LogInformation($"Employee with ID {id} deleted successfully.");

return NoContent();

}

}

}

* Change code in AuthController.cs by below

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.Extensions.Configuration;

namespace MyFirstAPI.Controllers

{

[ApiController]

[Route("Emp")]

public class AuthController : ControllerBase

{

private readonly IConfiguration \_configuration;

public AuthController(IConfiguration configuration)

{

\_configuration = configuration;

}

[HttpGet("token")]

public IActionResult GetToken(string role = "User")

{

var roles = new List<string>();

if (!string.IsNullOrEmpty(role))

{

if (role.Contains(','))

{

roles.AddRange(role.Split(',', StringSplitOptions.RemoveEmptyEntries).Select(r => r.Trim()));

}

else

{

roles.Add(role.Trim());

}

}

else

{

roles.Add("User");

}

var token = GenerateJSONWebToken(1, roles);

return Ok(new { token });

}

private string GenerateJSONWebToken(int userId, List<string> userRoles)

{

var securityKeyBytes = Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]);

var issuer = \_configuration["Jwt:Issuer"];

var audience = \_configuration["Jwt:Audience"];

var securityKey = new SymmetricSecurityKey(securityKeyBytes);

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

var claims = new List<Claim>

{

new Claim(JwtRegisteredClaimNames.Sub, userId.ToString()),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),

};

foreach (var role in userRoles)

{

claims.Add(new Claim(ClaimTypes.Role, role));

}

var token = new JwtSecurityToken(

issuer: issuer,

audience: audience,

claims: claims,

expires: DateTime.Now.AddMinutes(20),

signingCredentials: credentials

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

* Change code in Program.cs by below

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using MyFirstAPI.filters;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace MyFirstAPI

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers(options =>

{

options.Filters.Add<CustomExceptionFilter>();

});

builder.Services.AddScoped<CustomAuthFilter>();

builder.Services.AddScoped<CustomExceptionFilter>();

var jwtSettings = builder.Configuration.GetSection("Jwt");

var keyBytes = Encoding.UTF8.GetBytes(jwtSettings["Key"]);

var securityKey = new SymmetricSecurityKey(keyBytes);

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, options =>

{

options.RequireHttpsMetadata = false;

options.SaveToken = true;

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidIssuer = jwtSettings["Issuer"],

ValidateAudience = true,

ValidAudience = jwtSettings["Audience"],

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

IssuerSigningKey = securityKey,

ClockSkew = TimeSpan.Zero

};

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "MyFirstAPI",

Version = "v1",

Description = "API for Employees and JWT Authentication.",

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

Contact = new OpenApiContact

{

Name = "John Doe",

Email = "john@xyz.com",

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

},

License = new OpenApiLicense

{

Name = "License Terms",

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

}

});

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Description = "JWT Authorization header using the Bearer scheme. \r\n Enter 'Bearer [your token]' below.",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new List<string>()

}

});

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

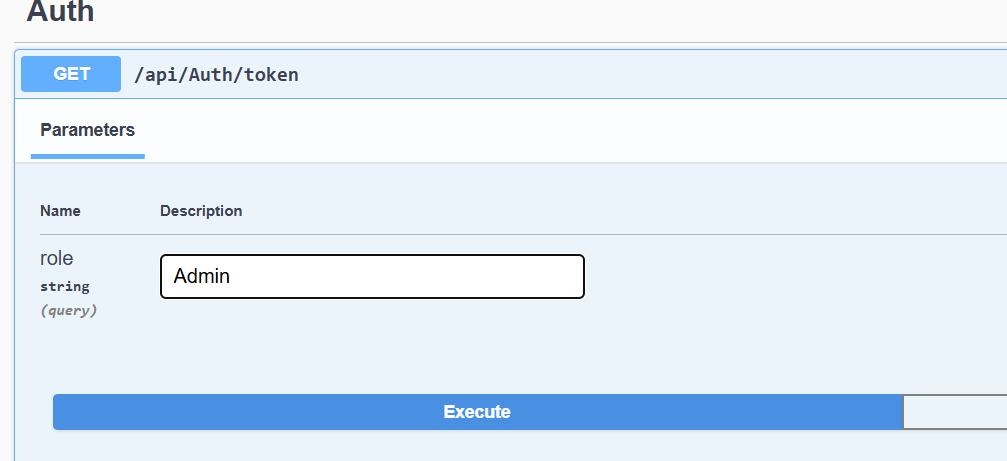
app.Run();

}

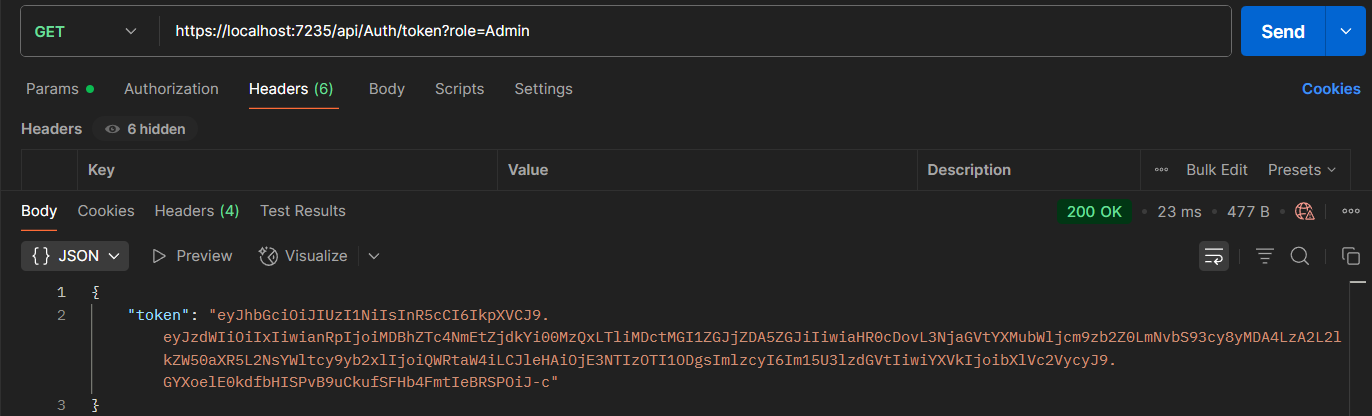
}

}

Step7: Use Admin as role to get token in Auth GET?api/Auth/Token



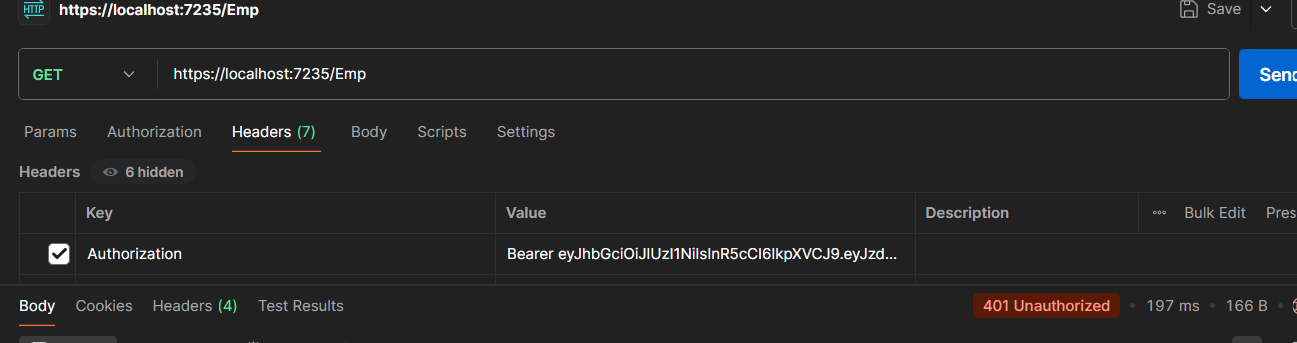
Paste the URL generated in Postman [**https://localhost:7235/api/Auth/token?role=Admin**](https://localhost:7235/api/Auth/token?role=Admin) and u can see a token generated in response

****

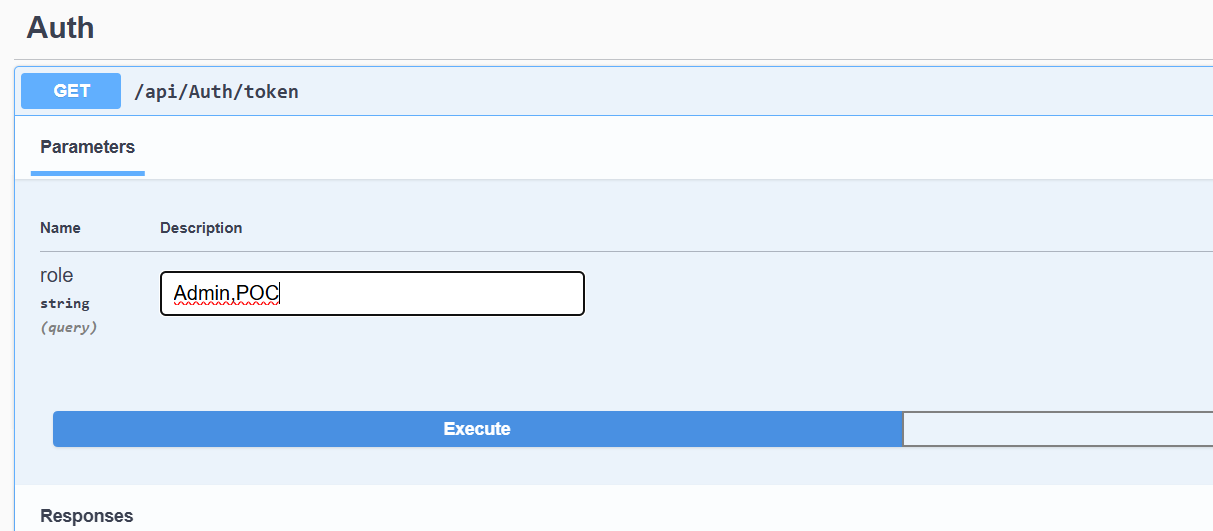
Now use in Token in Header Key-“Authorization” and Value as “Bearer <Token>”

Attempt to Access Employee Data without the 'POC' role by using URL from

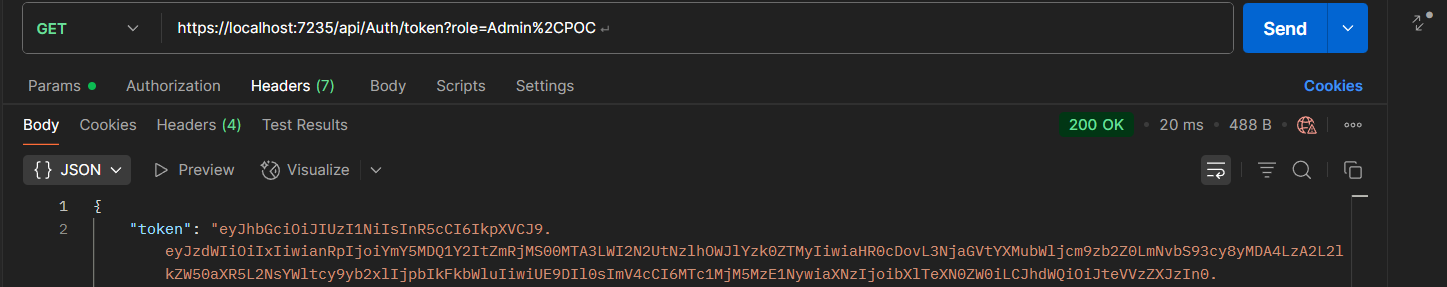
GET /Emp:



Now generate the Auth token by role both Admin,POC and use the token to access



Use the URL [**https://localhost:7235/api/Auth/token?role=Admin%2CPOC**](https://localhost:7235/api/Auth/token?role=Admin%2CPOC)in postman to get the token and use the token in Header Key-“Authorization” and Value as “Bearer <Token>”

****

Now Attempt to Access Employee Data 