**WEEK 4 Handson**

**1. WebApi\_Handson**

**Objectives:**

**1.Explain the concept of RESTful web service, Web API & Microservice**

**RESTful Web Service**

* **REST:** *Representational State Transfer*.
* An **architectural style** for designing web services.
* Uses **HTTP** for communication.
* Resources are identified using **URIs** and transferred via **representations** (JSON, XML, etc.).

**Features of REST Architecture:**

**Key Features of REST:**

| **Feature** | **Description** |
| --- | --- |
| **Stateless** | No session info stored on server; every request has all info needed. |
| **Uniform Interface** | Standard operations (GET, POST, PUT, DELETE). |
| **Cacheable** | Responses can be cached. |
| **Client-Server** | Separation of concerns between client & server. |
| **Layered System** | Intermediary layers can exist. |

**Web API**

* Framework for building **HTTP-based services**.
* Supports **multiple formats** (JSON, XML).
* Used to expose business logic to different clients.
* Example: ASP.NET Core Web API.

**Microservice**

* Architectural style that structures an app as a **collection of loosely coupled services**.
* Each service:
  + Has its own database.
  + Runs independently.
  + Focuses on a specific business capability.
* Communicates using lightweight protocols (HTTP, gRPC, etc.).

**Difference Between WebService & WebAPI:**

| **Feature** | **Web Service** | **Web API** |
| --- | --- | --- |
| Protocol | SOAP | HTTP/HTTPS |
| Format | XML | JSON, XML, etc. |
| Speed | Slower due to SOAP overhead | Faster (lightweight) |
| Platform | Platform dependent | Platform independent |
| Usage | Enterprise-level applications | Web, Mobile, Desktop apps |

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

**HttpRequest**

* Sent by client to server.
* Contains:
  + HTTP Method (GET, POST, etc.)
  + Headers (User-Agent, Authorization)
  + URL
  + Body (for POST/PUT)

**HttpResponse**

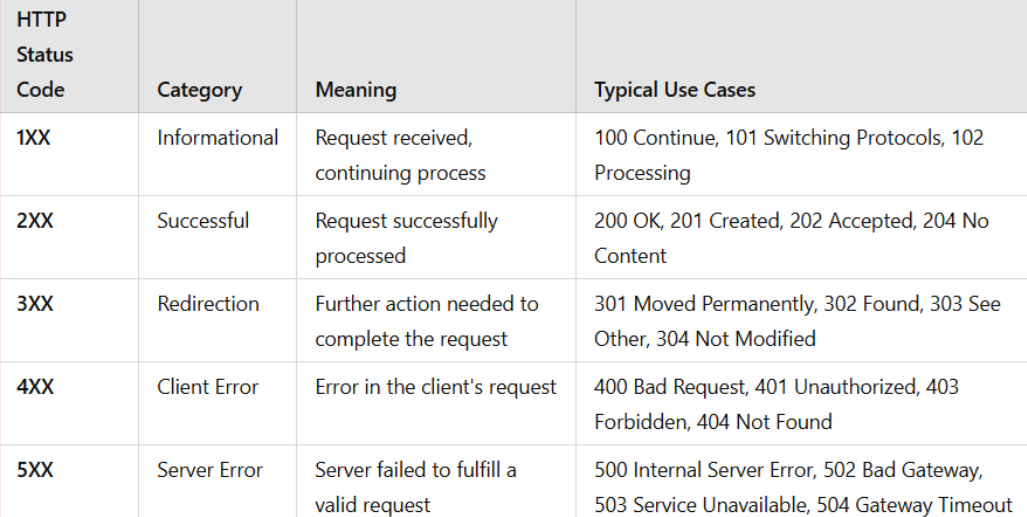
* Sent by server to client.
* Contains:
  + Status Code (200, 404, etc.)
  + Headers
  + Body (usually JSON/XML response)

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

* **Types of Action Verbs**

| **Verb** | **Meaning** | **Attribute in Web API** |
| --- | --- | --- |
| **GET** | Read data | [HttpGet] |
| **POST** | Create new resource | [HttpPost] |
| **PUT** | Update existing resource | [HttpPut] |
| **DELETE** | Remove resource | [HttpDelete] |

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



**Explain the types of 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.

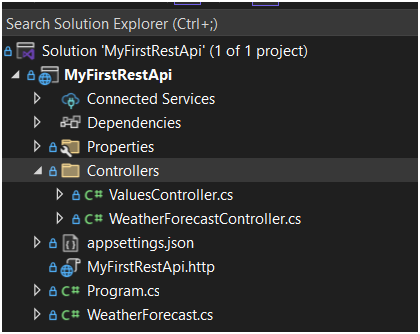
5.WebApi.config

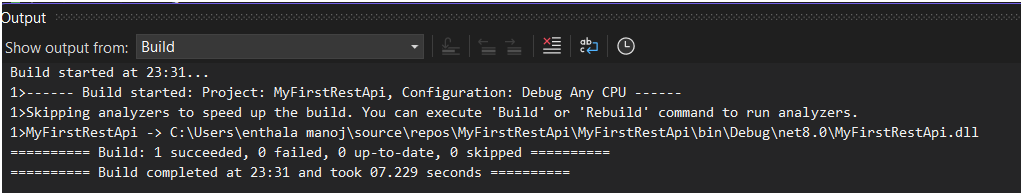
* Registers Web API routes and configures settings like response format (JSON, XML).
* Helps set up how requests are handled across the Web API application.

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

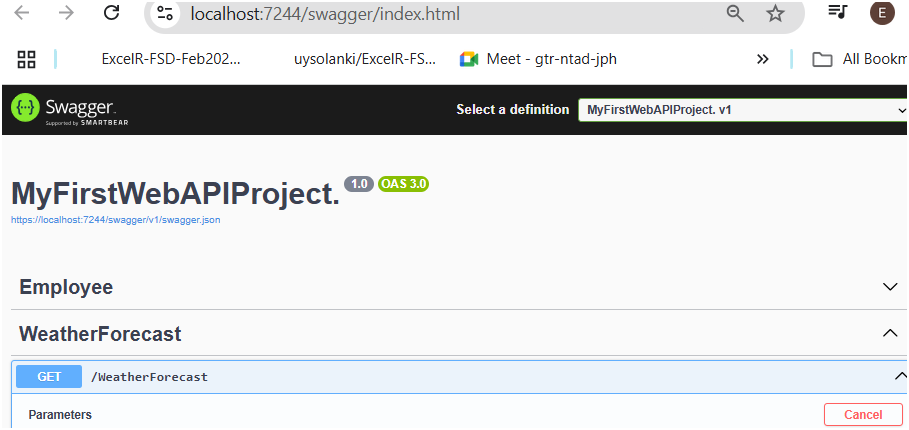
Create a .Net core web application with API template. Use the option to create controller with Read Write permissions. Notice the ValuesController creation with Action methods corresponding to the Action verbs.

On creation of the Web API, execute the application and check if the GET action method result is returned as expected.

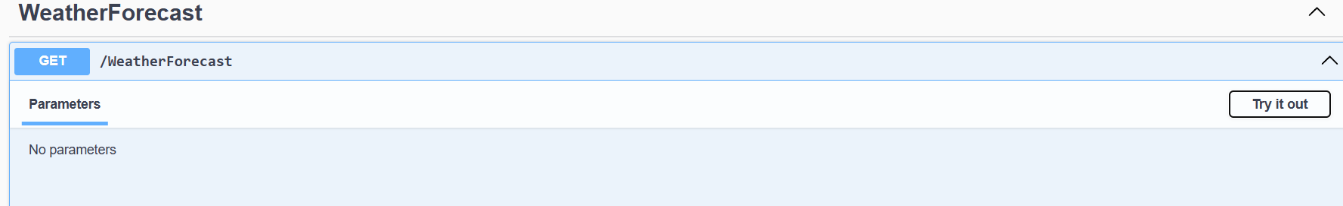




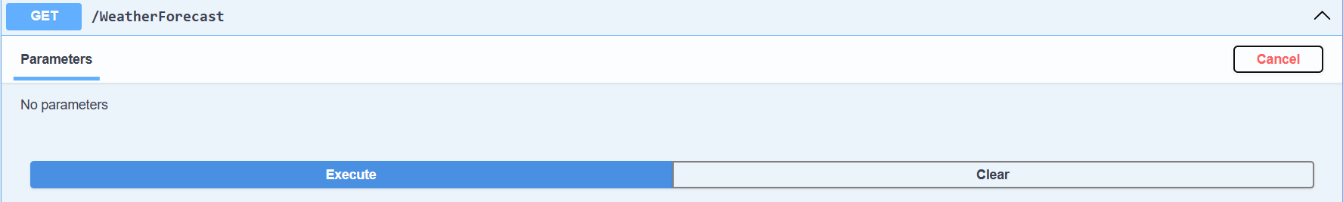
after run it will open browser and display like below



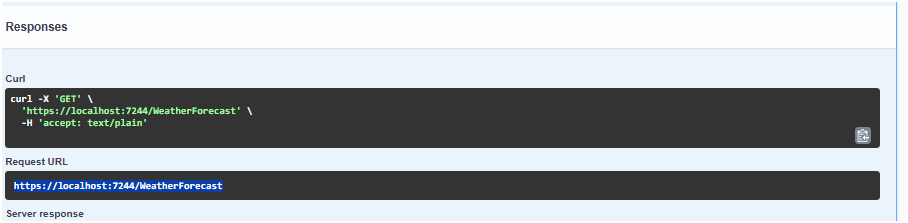
click try it out



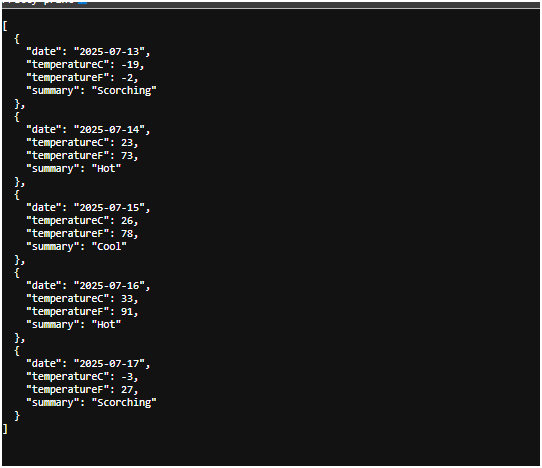
click excute



use the Request URL in browser

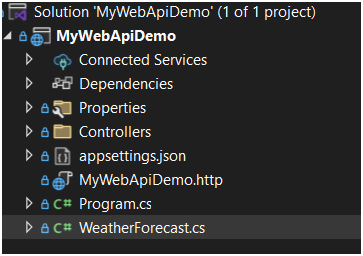


Output: call the Request URL in any of your browsers, and you will get the response as expected

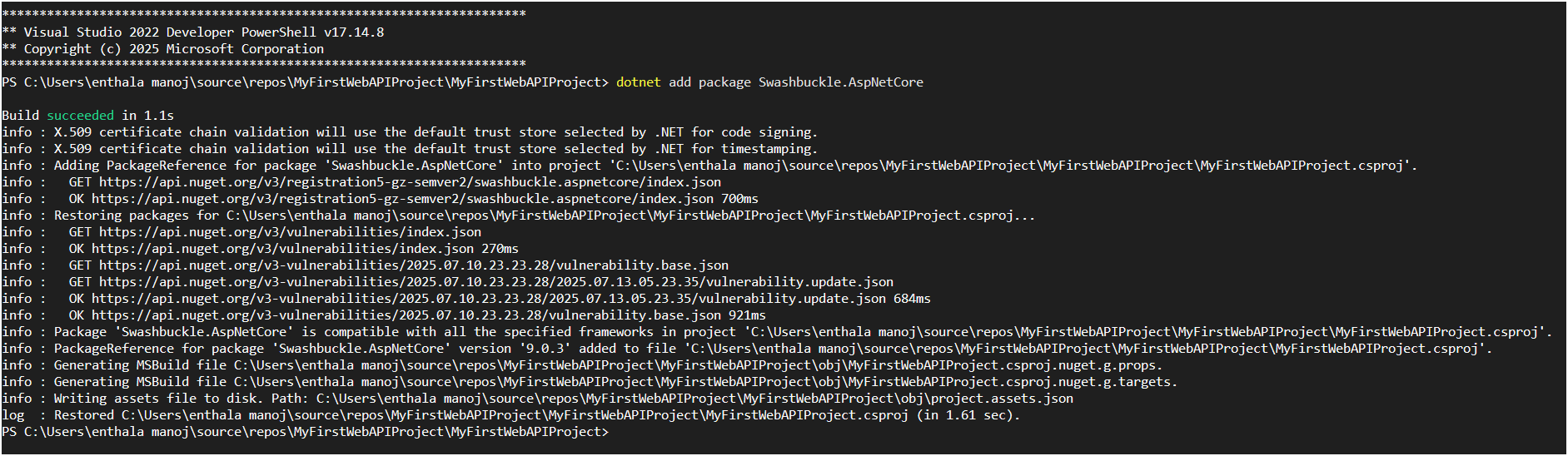


**Web Api using .Net core with Swagger**

Create a .Net core web application with API template. (Use existing application if created).



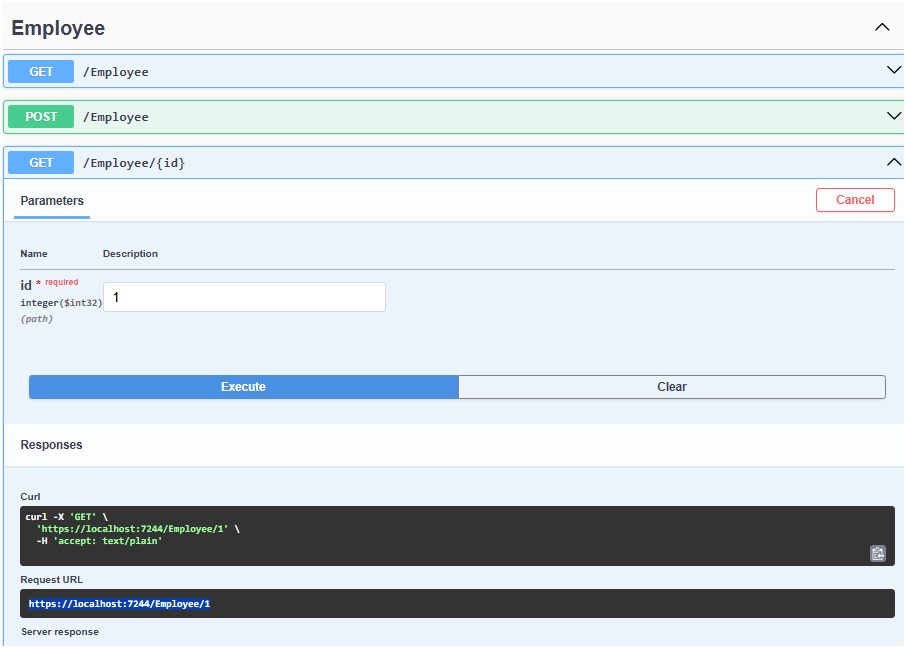
Install Swashbuckle.AspNetCore Nuget package.



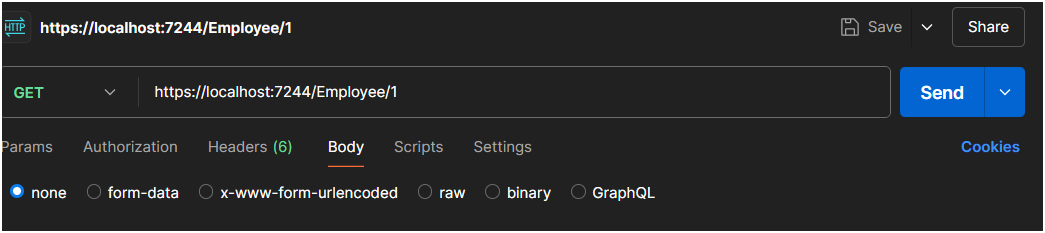
alter program.cs file’s “services.AddSwaggerGen” and “app.UseSwaggerUI” similar to below



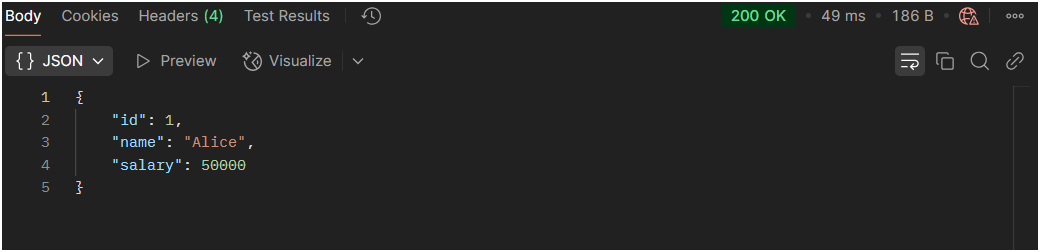
now run the project in “https” and give “try it out” to “execute” and give the value for id get the request url



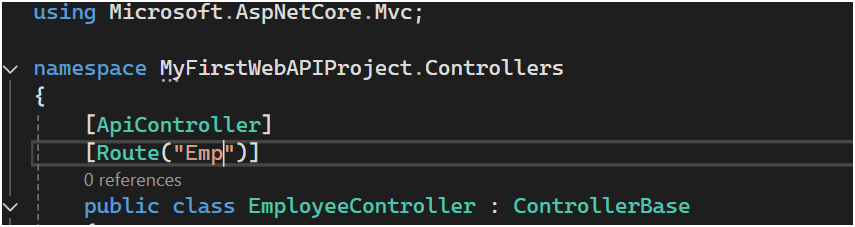
now give the url in postmen “GET”

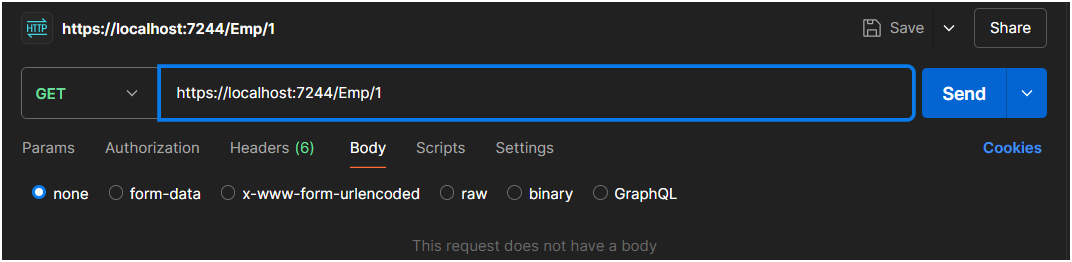


now check the value inside the response similar to the data expected



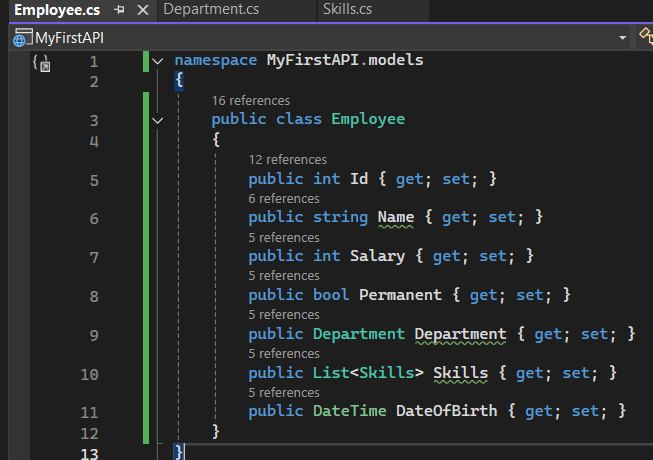
Modify the Controller name in the Route attribute of the Employee controller to ‘Emp’ and check its access thru POSTMAN

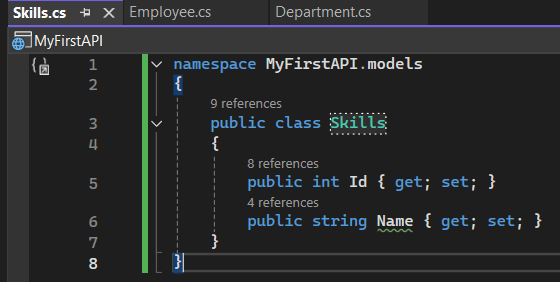


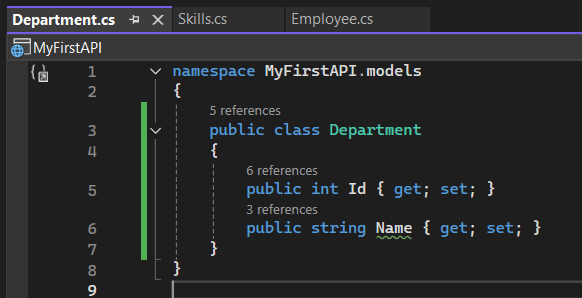


EX3: Web\_API Handson

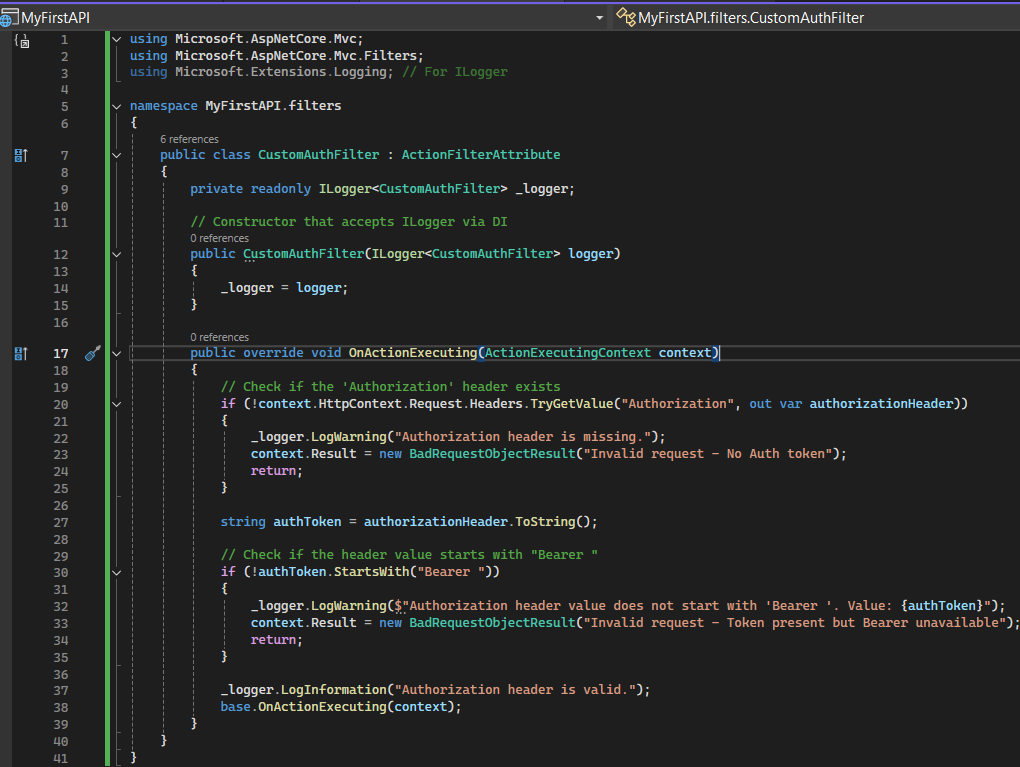
Step1: create Skills.cs, Department.cs and Employee.cs and write the codes given below







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);

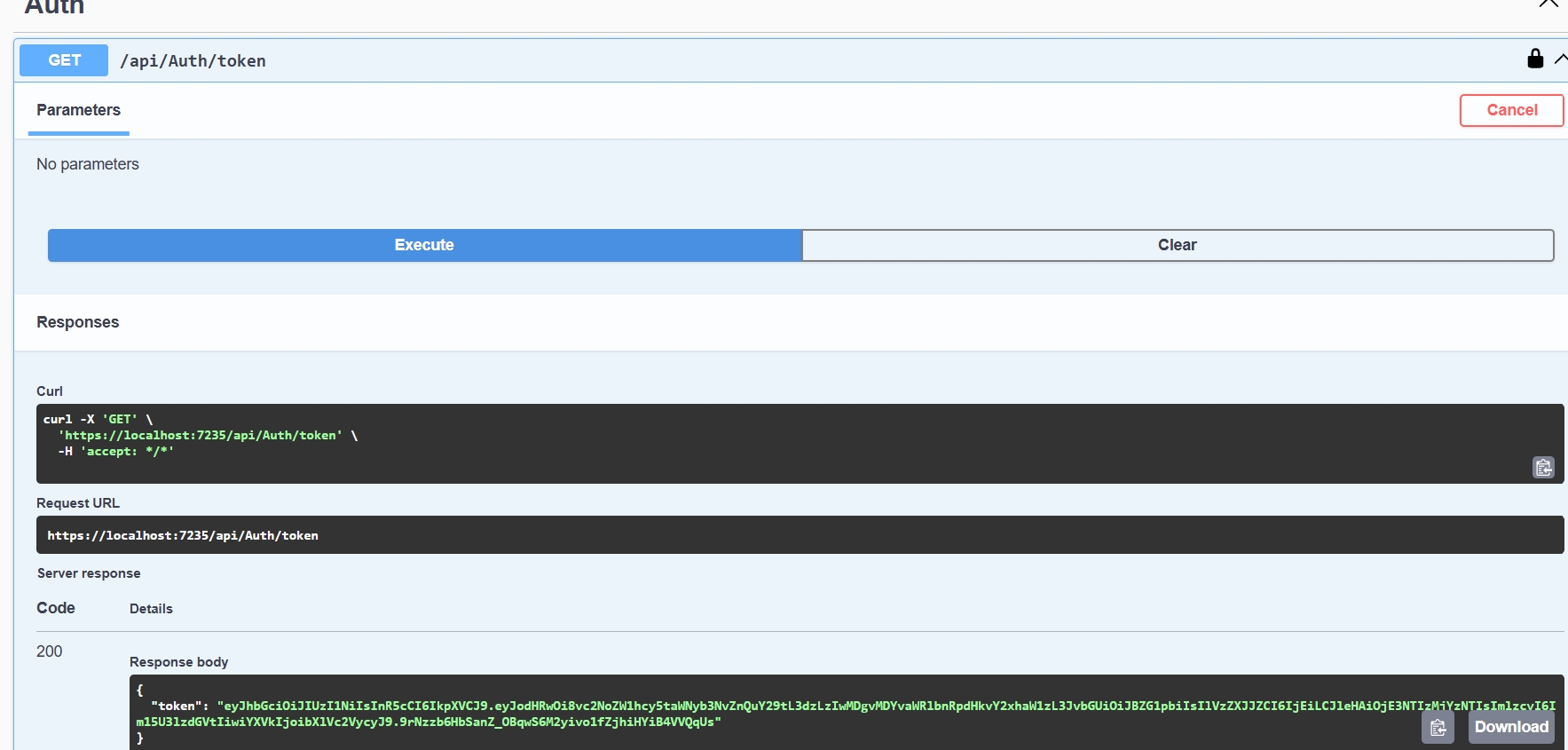
\_logger.LogInformation($"Employee with ID {id} deleted successfully.");

return NoContent();

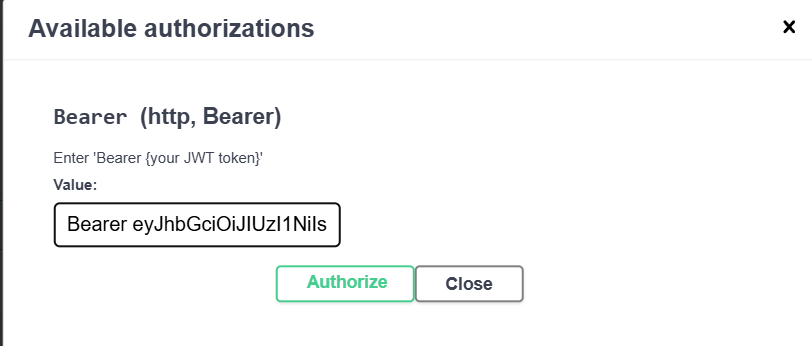
}

}

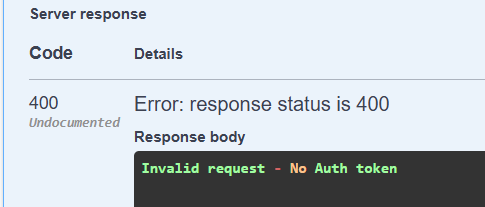
}

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>



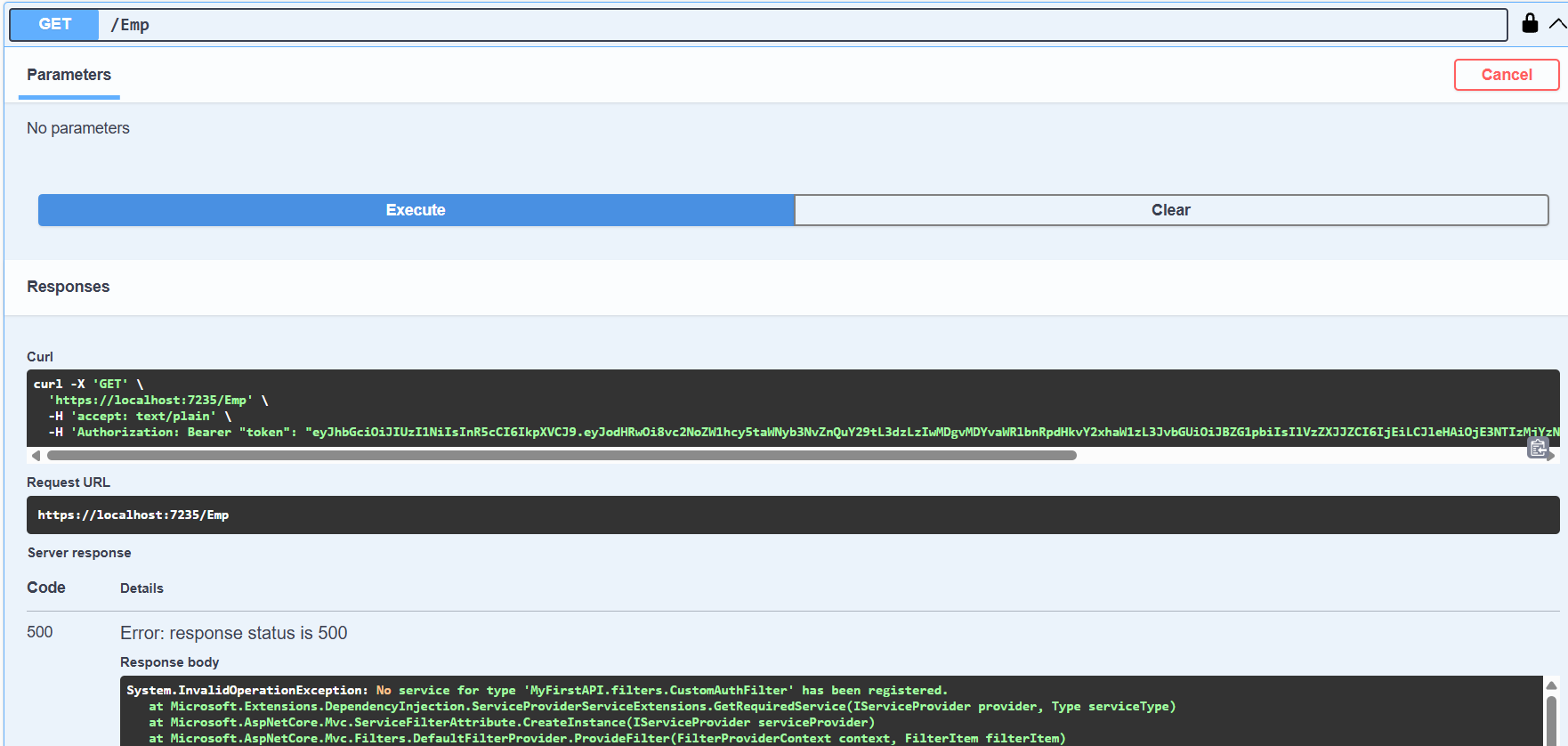
Output1: executing without Authorize will get code 400 with “Invalid requeset - No Auth token”



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("api/[controller]")]

[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.");

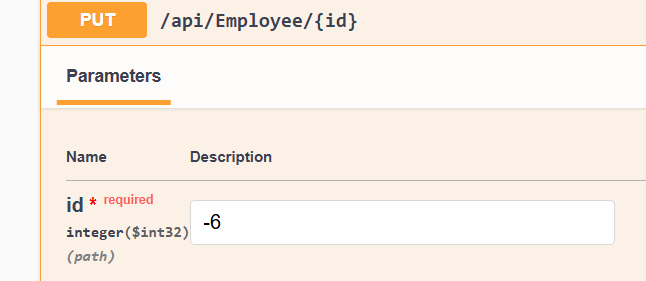
return NoContent();

}

}

}

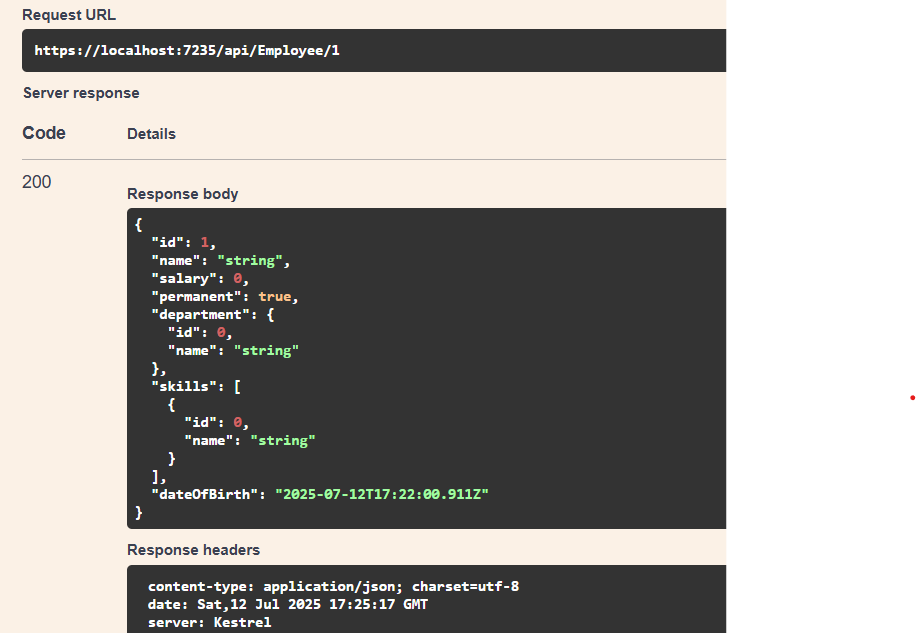
Step2: Now run the application and give input inside PUT as some invalid values and Execute to see a “BAD REQUEST” code 400



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("api/[controller]")]

[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

);

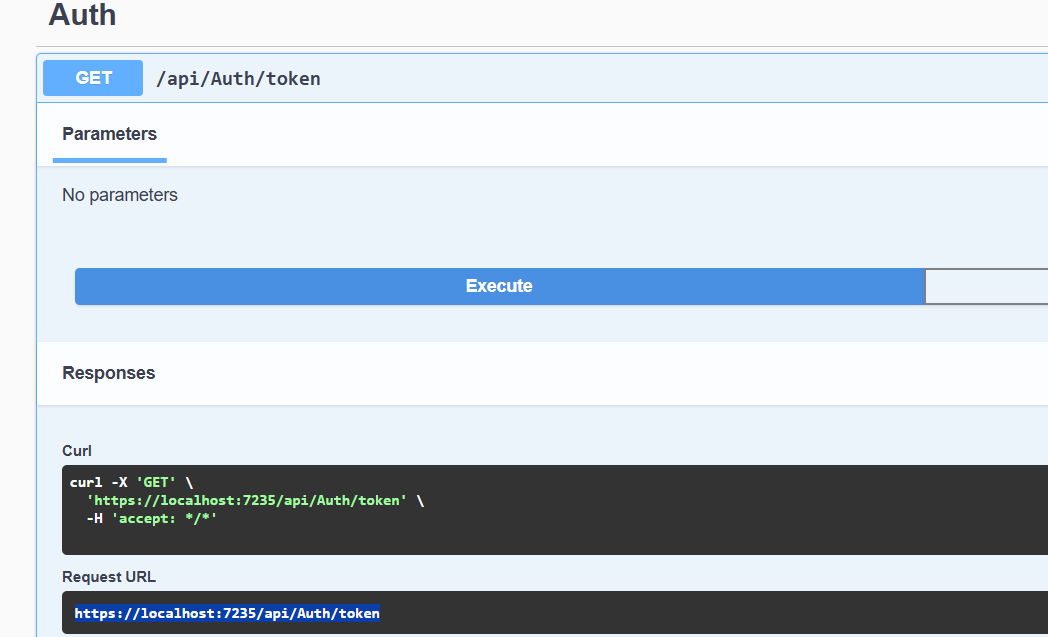
return new JwtSecurityTokenHandler().WriteToken(token);

}

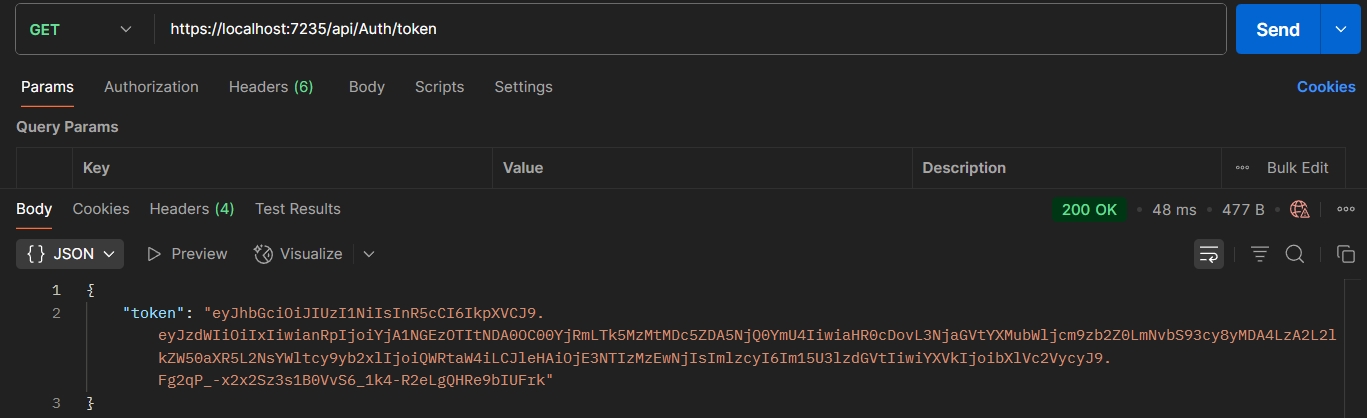
}

}

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,

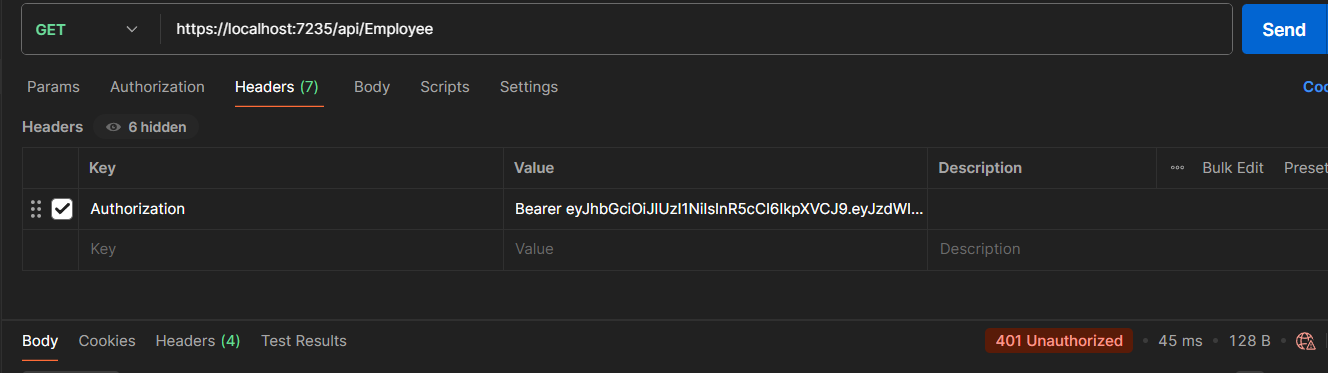
claims: claims,

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 Employee Get



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("api/[controller]")]

[ServiceFilter(typeof(CustomAuthFilter))] // Applies CustomAuthFilter to all actions in this controller

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("api/[controller]")]

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();

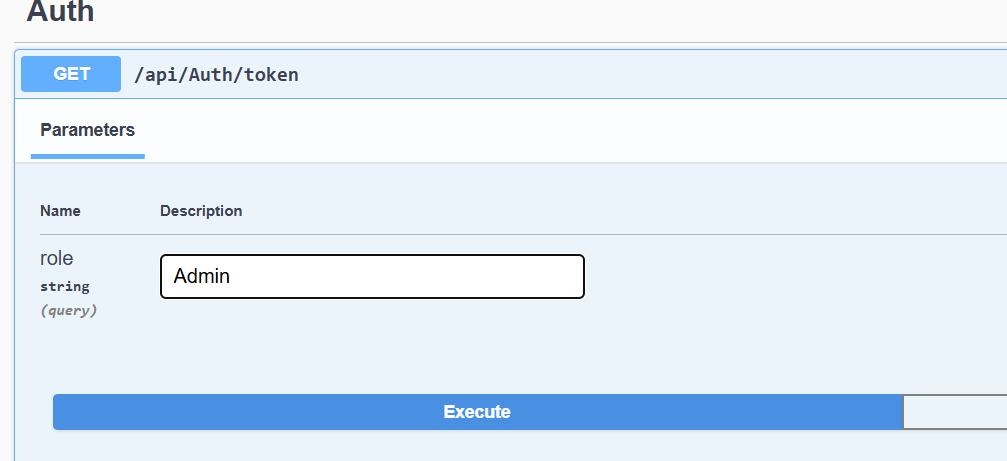
app.Run();

}

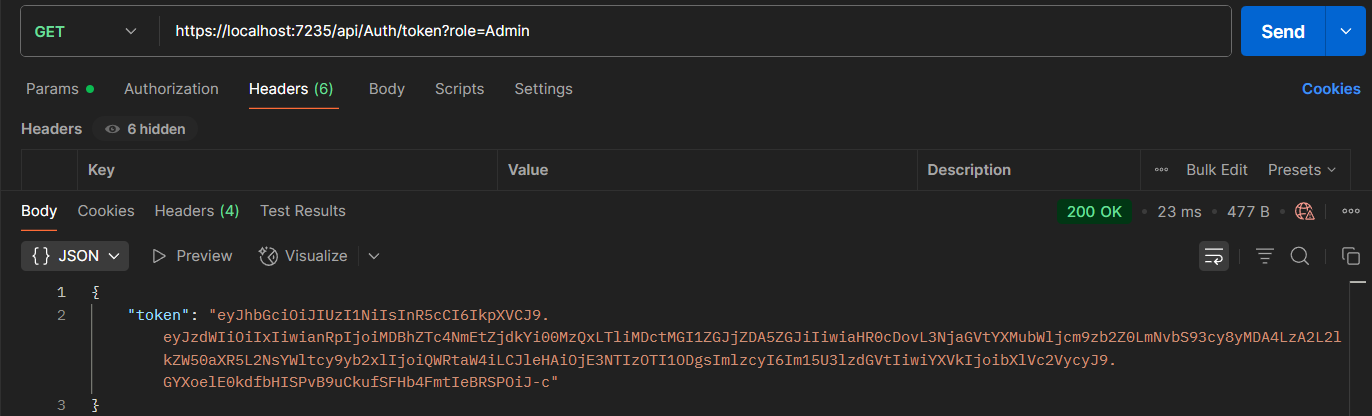
}

}

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



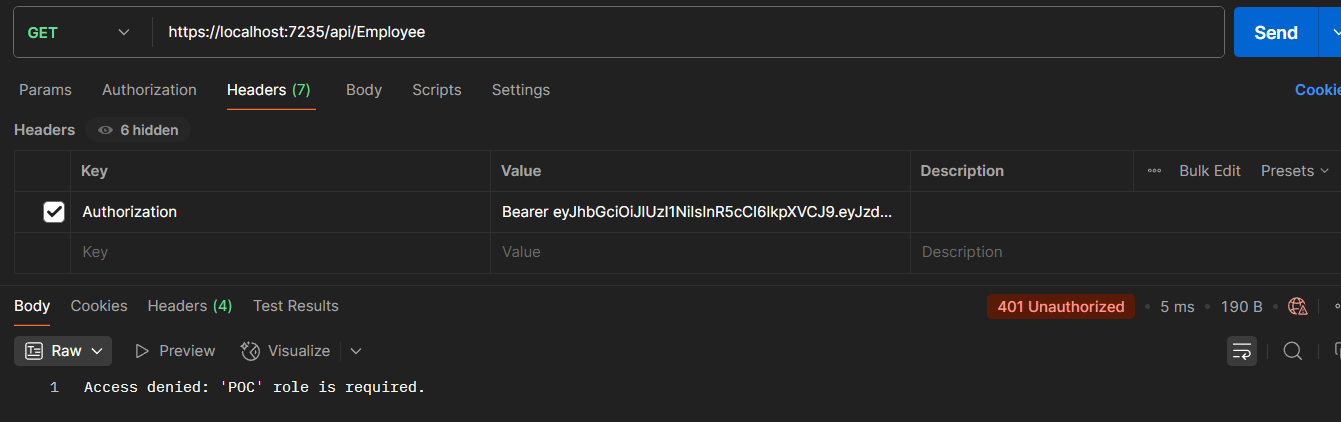
Paste the URL generated in Postman [**https://localhost:7235/api/Auth/token?role=Admin**](https://localhost:7235/api/Auth/token?role=Admin) and u can see a token generated in response

****

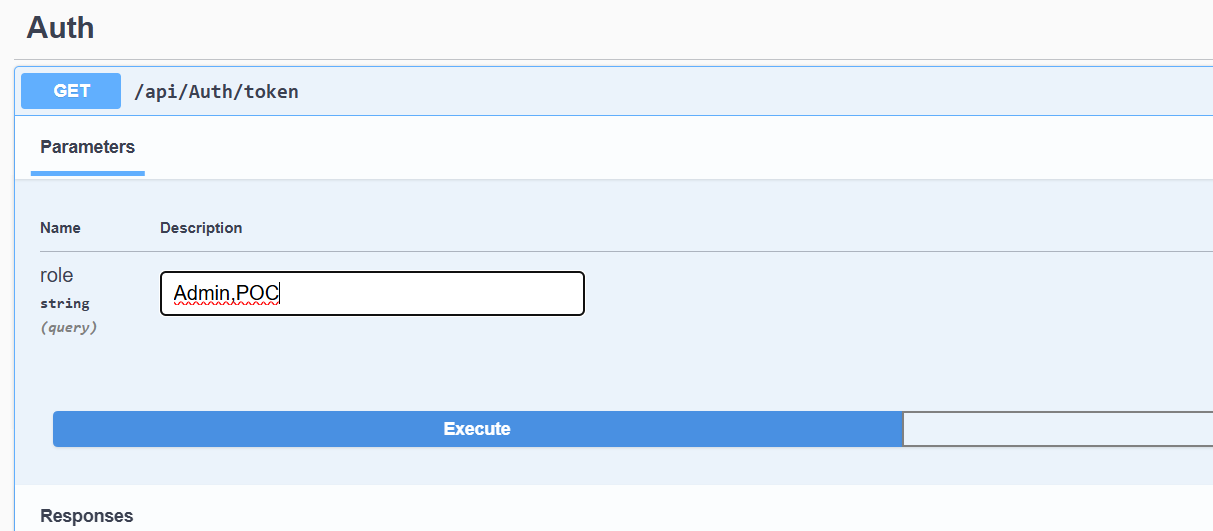
Now use in Token in Header Key-“Authorization” and Value as “Bearer <Token>”

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

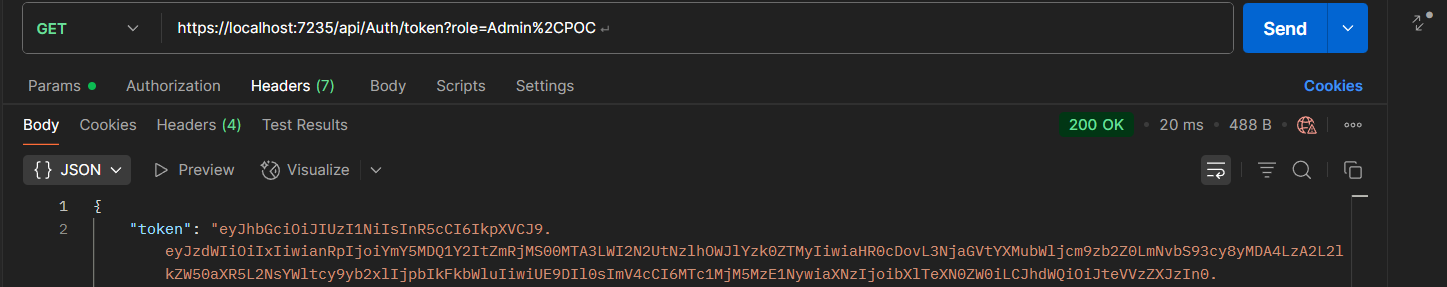
GET /api/Employee:



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 