**1 RESTful Web Service**

REST stands for **Representational State Transfer**, a lightweight, stateless, and cacheable protocol used to create scalable web services.

**Key Features:**

* **Stateless:** Each request from client to server must contain all the information.
* **Uniform Interface:** Standard methods like GET, POST, PUT, DELETE.
* **Representation:** Data is typically sent as JSON (not limited to XML).
* **Client-Server Architecture:** Separation of concerns.
* **Cacheable:** Responses must define themselves as cacheable or not.

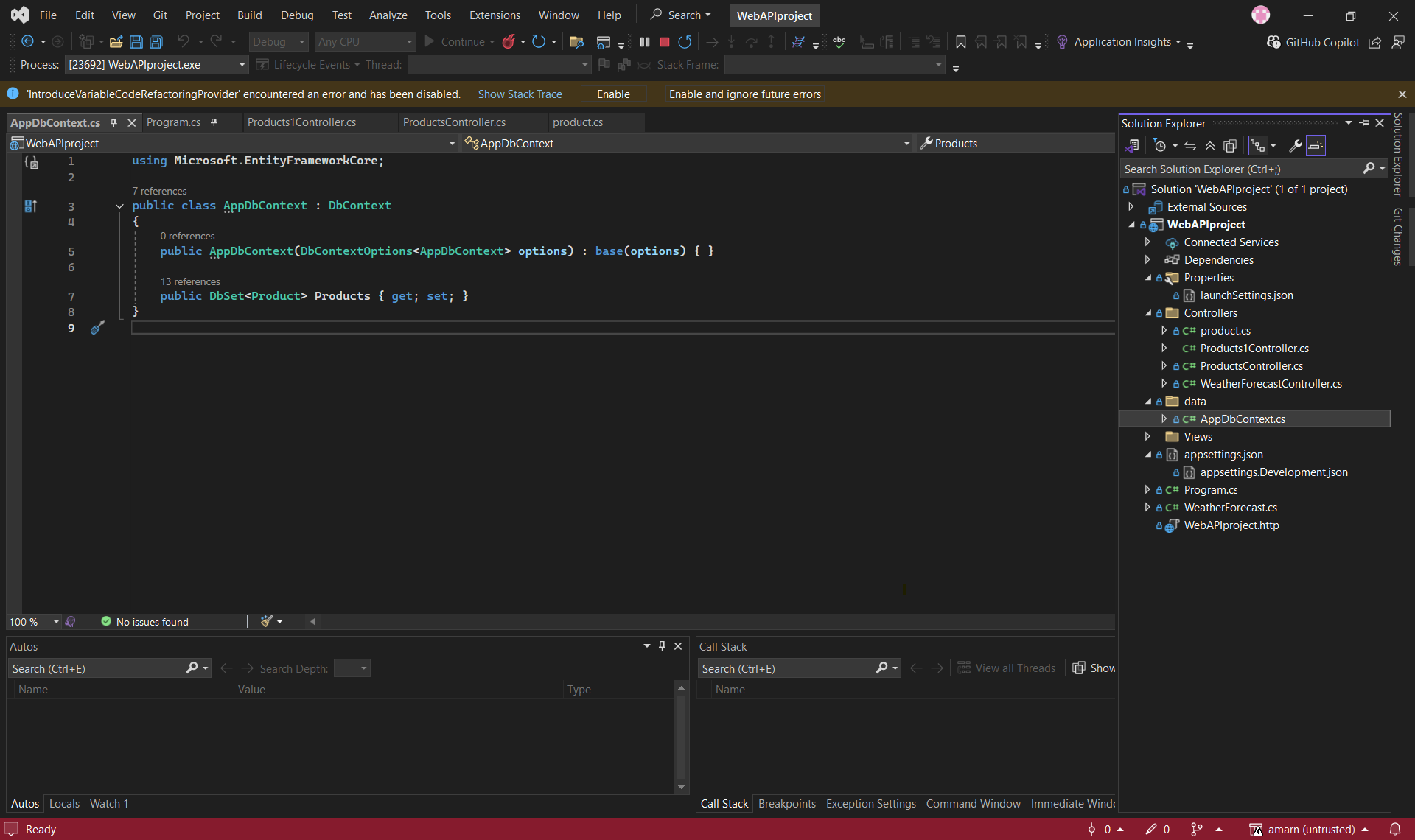
**2. HTTPRequest & HTTPResponse**

* **HttpRequest:** Represents a request from a client to the server (contains method, URL, headers, and body).
* **HttpResponse:** Server’s response to the client (contains status code, body, and headers).

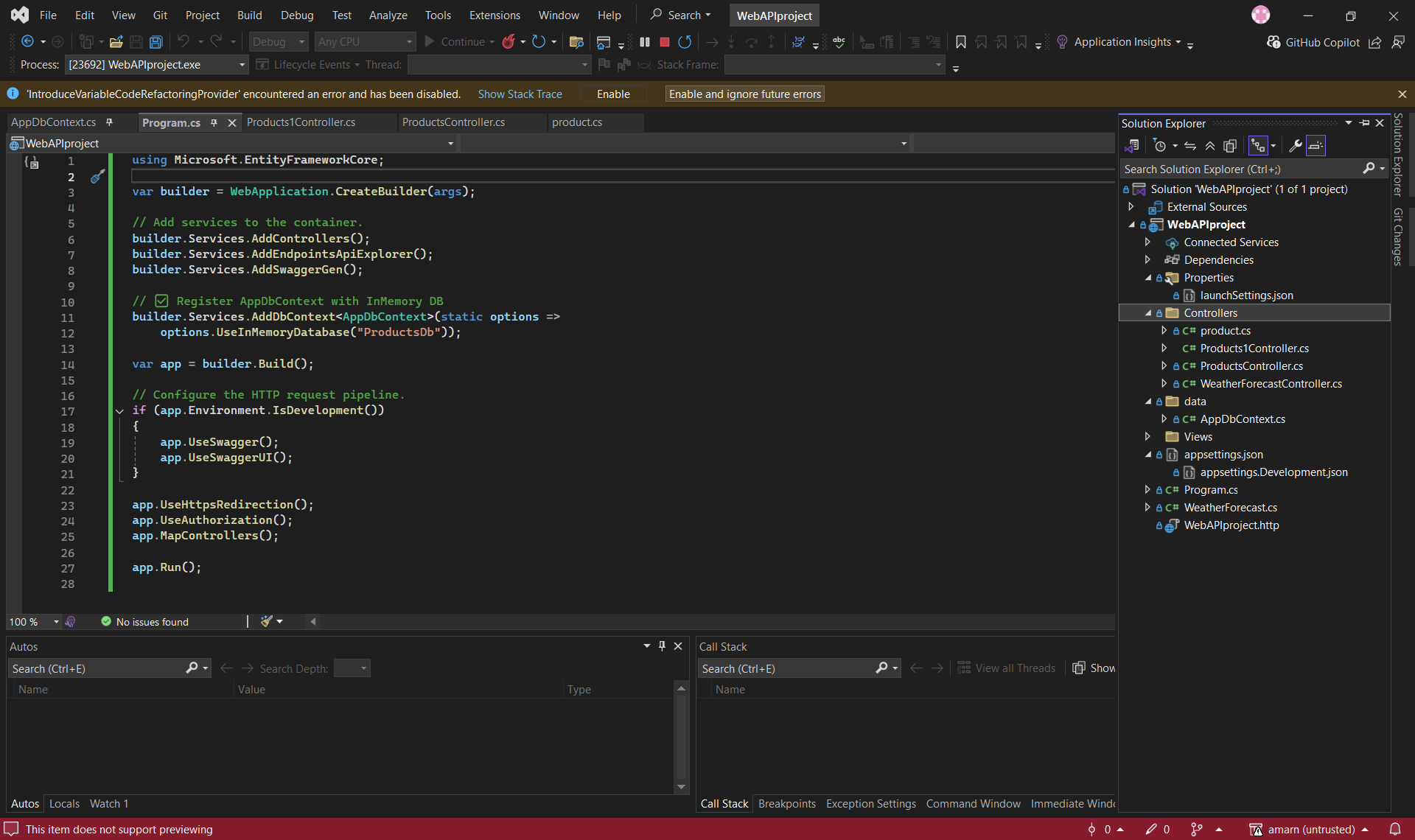
1. **First Web Api using .Net core**

**Create a new controller.**

**Using the option to create controller with Read Write permissions.**

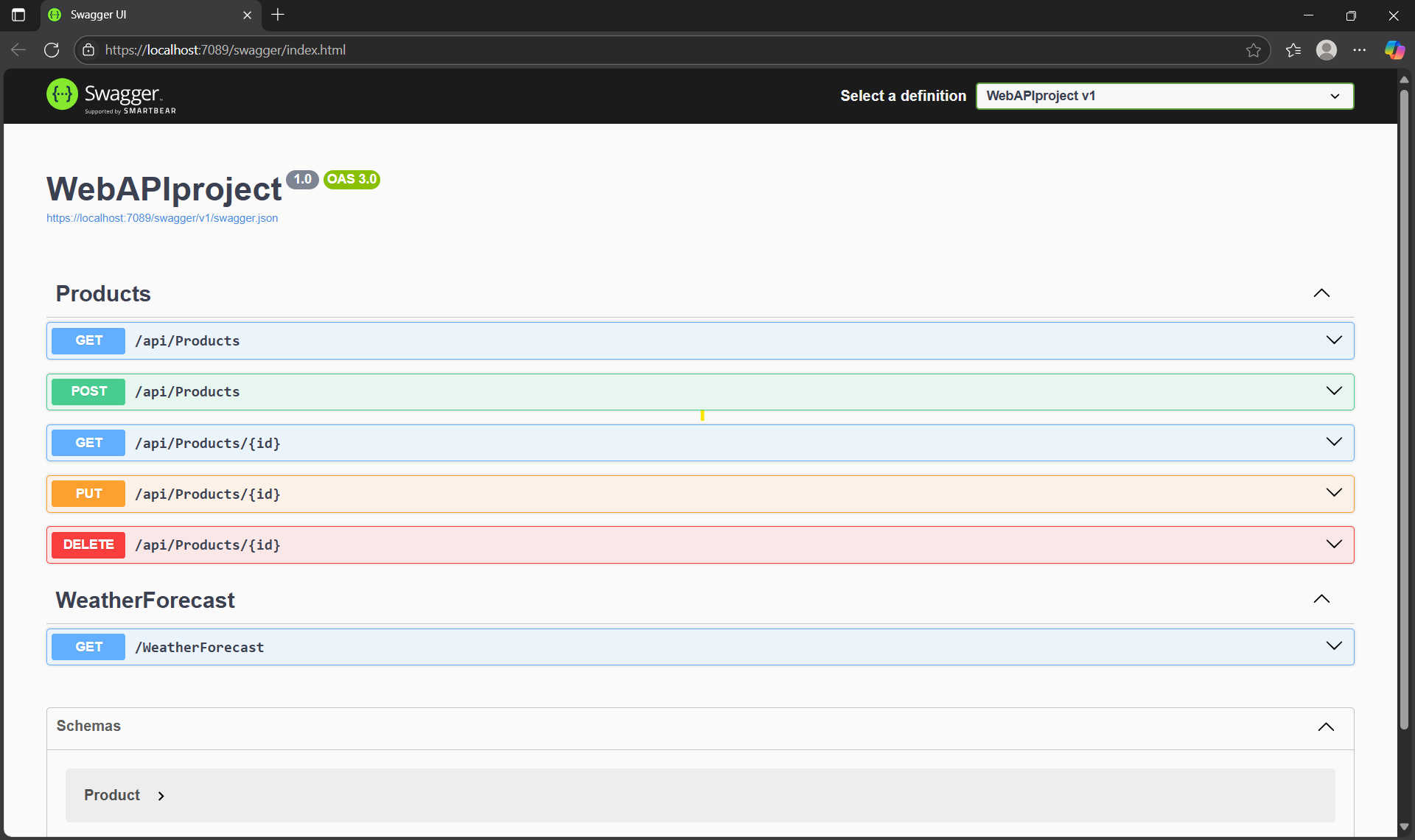


Creating appcontext.db

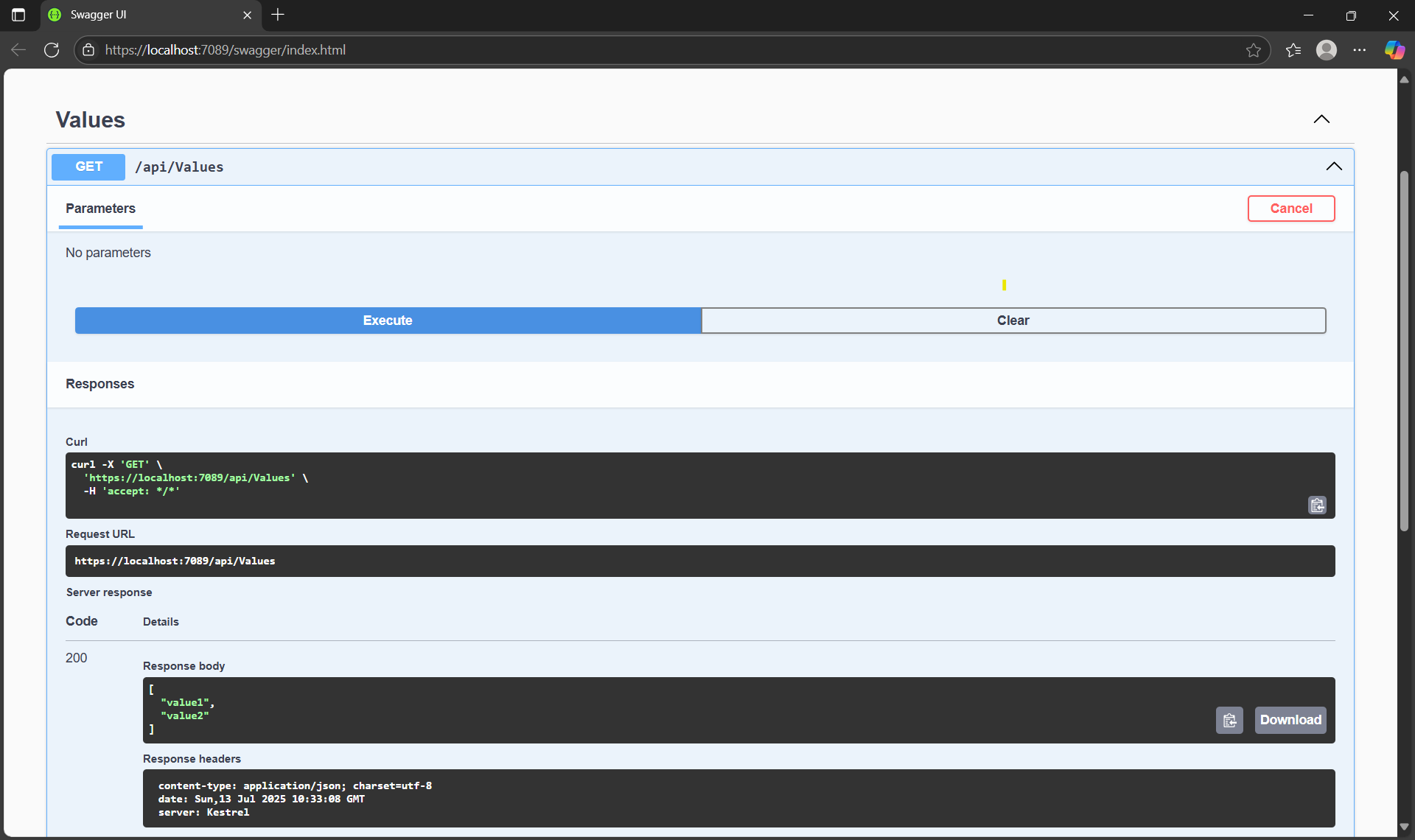


WEB API project :

After running the project we can see this page redirected from swagger>.



Open a browser or Postman and navigate to:



Exercise: 2

Demonstrating Swagger Installation and Usage with WebAPI:

Swagger (OpenAPI) provides a standardized, language-agnostic interface to REST APIs, allowing both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or network traffic inspection. Swashbuckle.AspNetCore is a popular library for integrating Swagger into ASP.NET Core Web APIs.

**1. Nuget Package to Download: Swashbuckle.AspNetCore**

Steps:

1. Create a new ASP.NET Core Web API project:

o Open Visual Studio.

o Select "Create a new project."

o Search for and select "ASP.NET Core Web API."

o Click "Next," provide a name for your project (e.g., MyWebApiSwaggerDemo), and click "Create."

1. **Install the Swashbuckle.AspNetCore NuGet Package:**

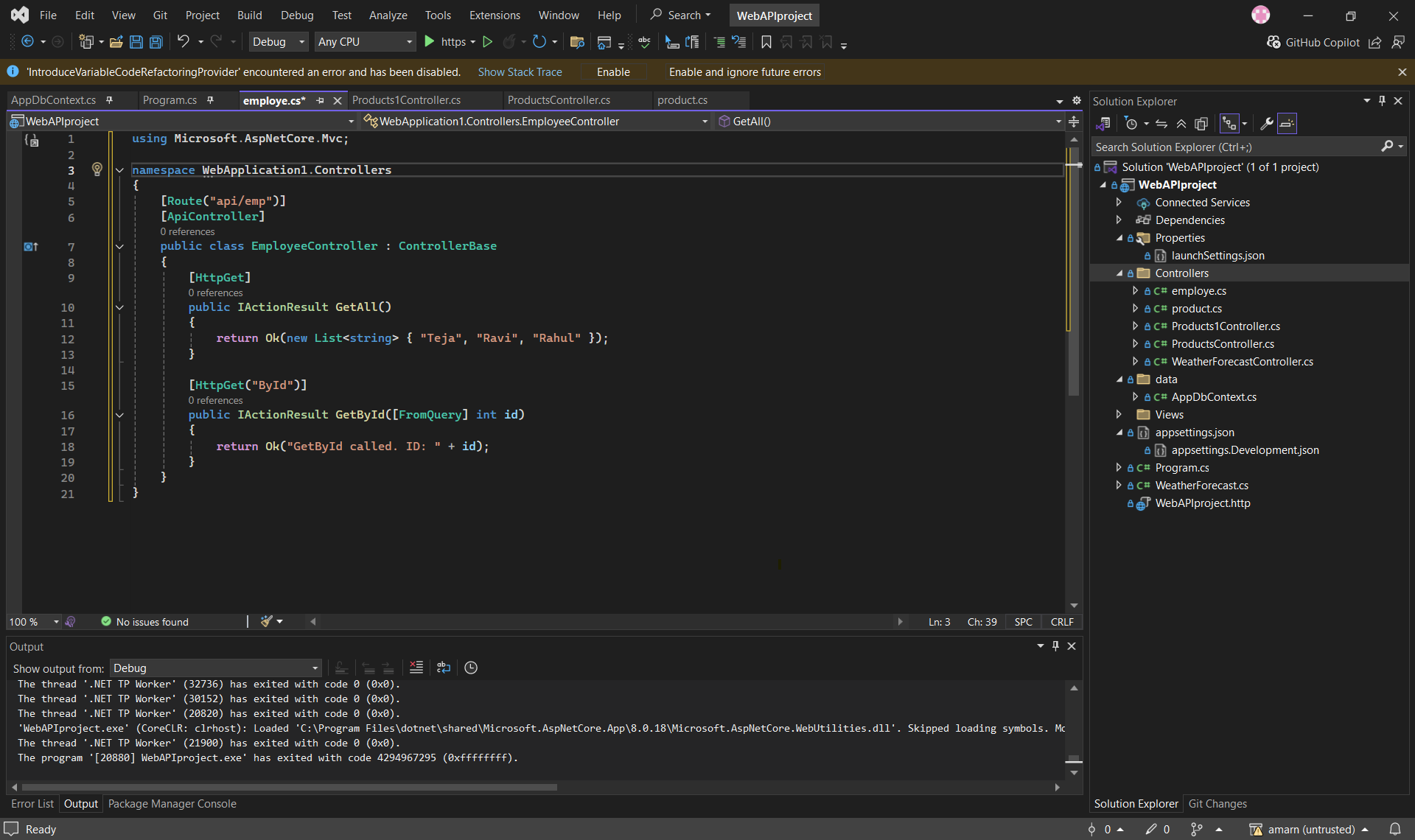
o In Visual Studio, right-click on your project in Solution Explorer.

o Select "Manage NuGet Packages..."

o Go to the "Browse" tab.

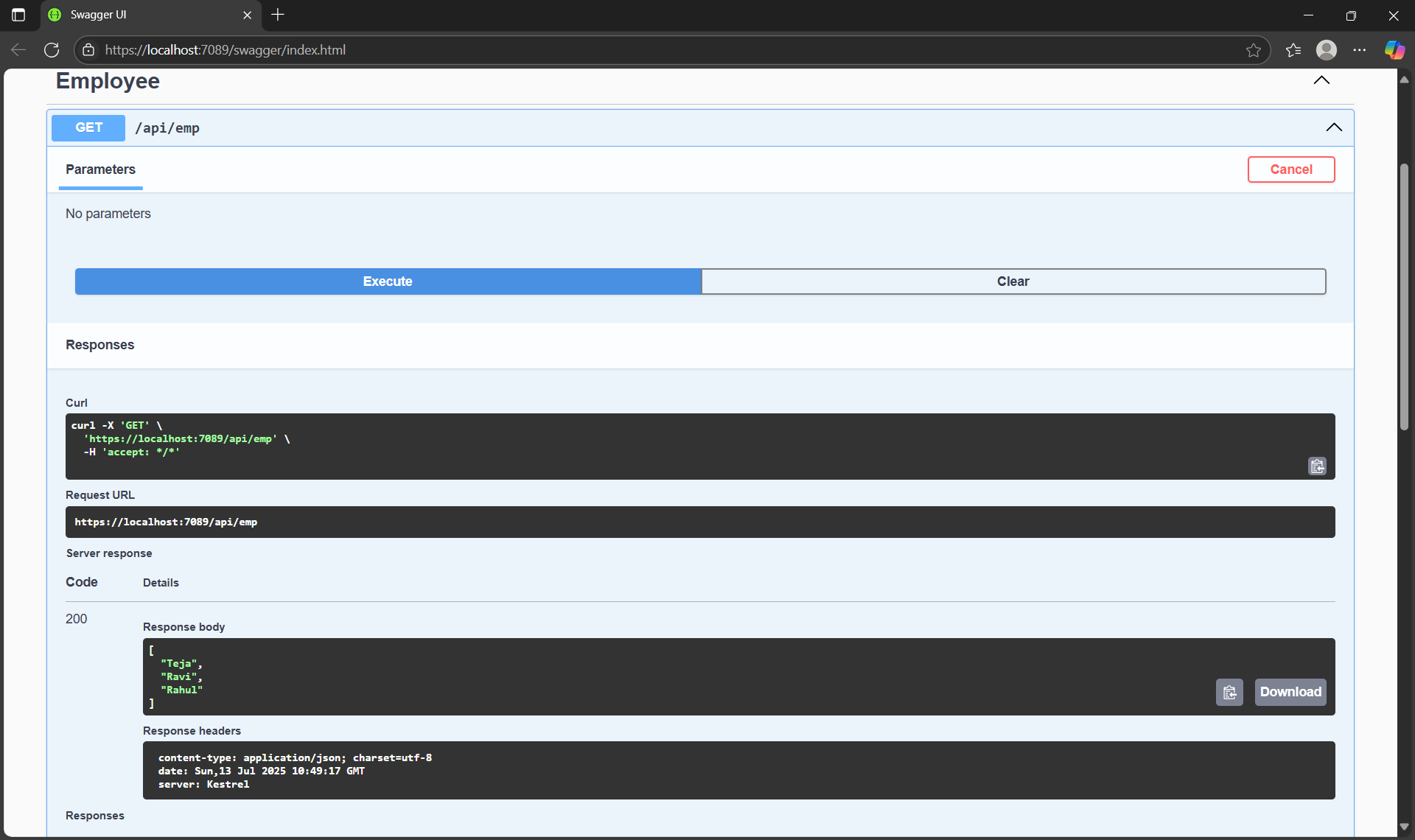
o Search for Swashbuckle.AspNetCore.

o Select the latest stable version and click "Install."



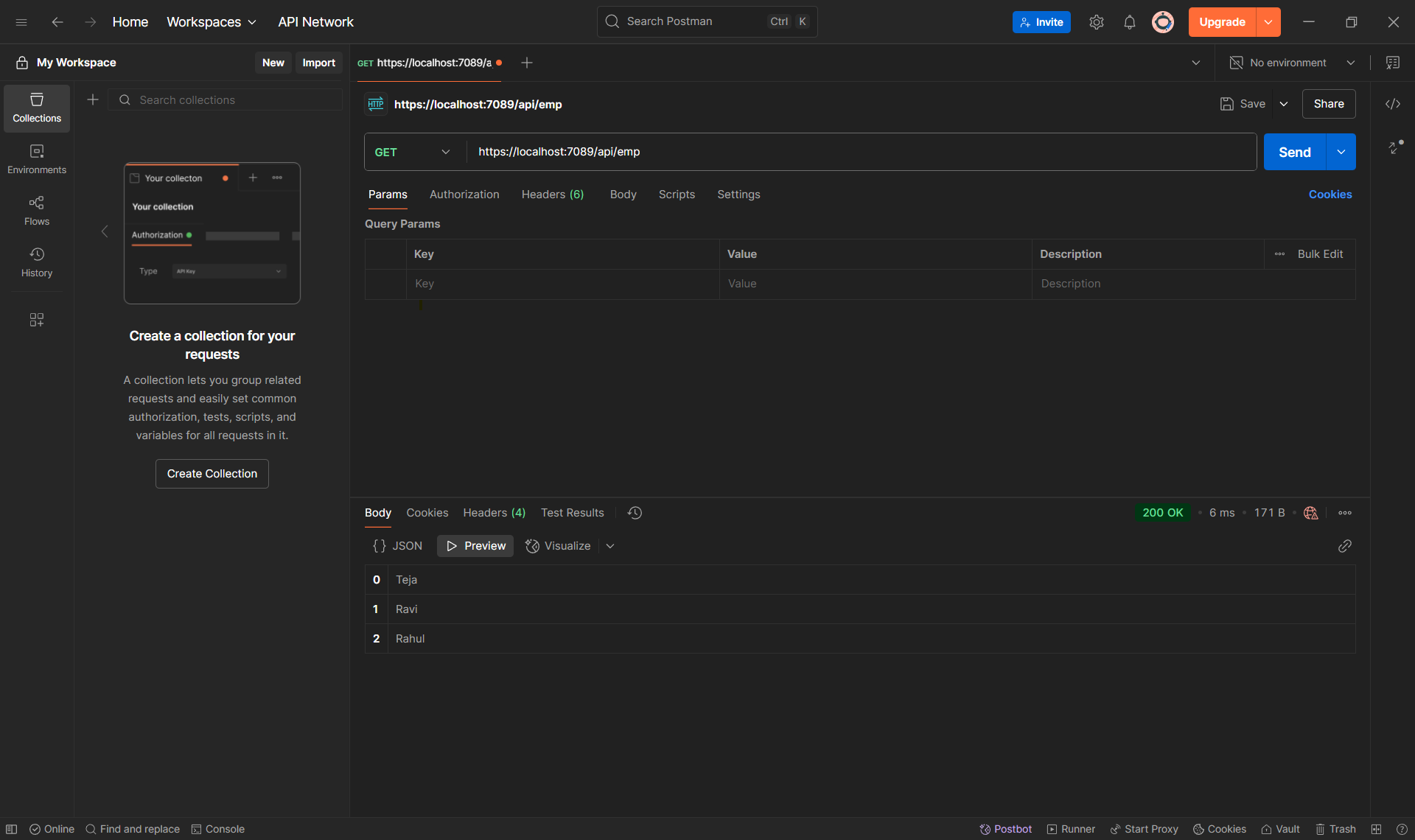
Swager UI :

Output

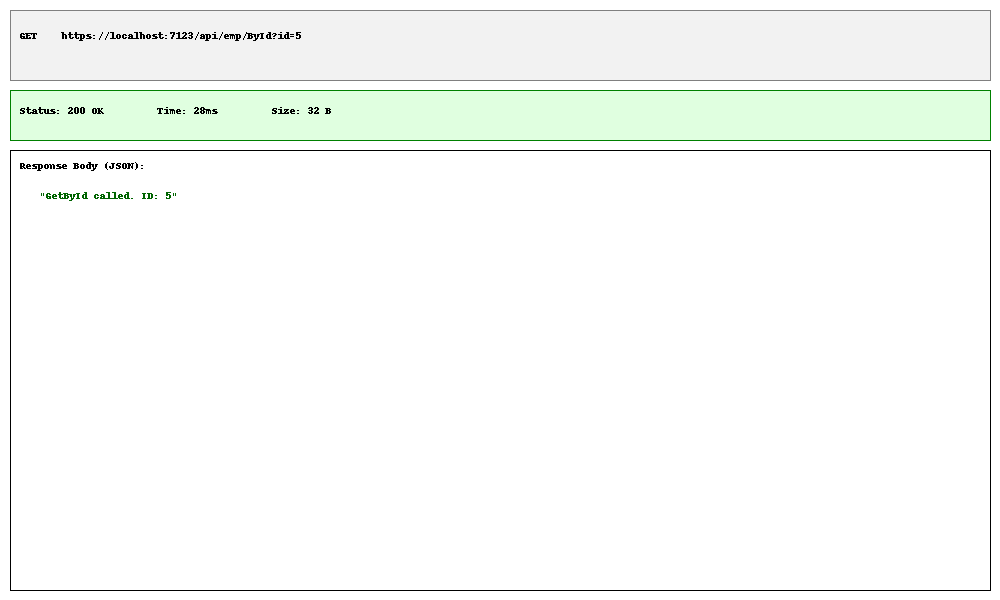


**https://localhost:7089/api/emp**

preview:

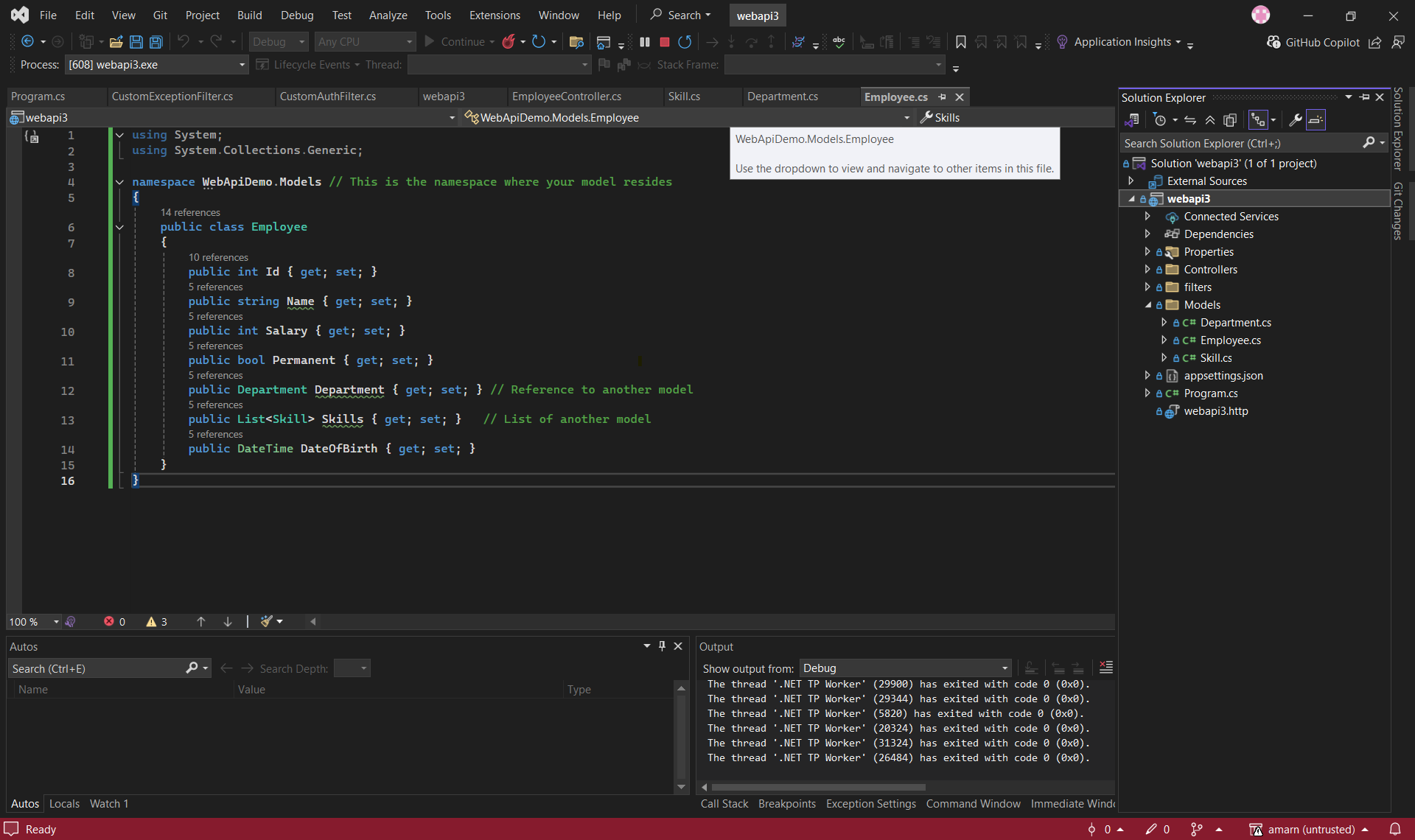


Postman output after verifying:



**Exercise: 3**

**Creating model classes:**

****

**Skill.cs**

namespace WebApiDemo.Models

{

public class Skill

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**Department.cs:**

namespace WebApiDemo.Models

{

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**Step 2: adding filters:**

**Filters/CustomAuthFilter.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using System.Linq;

namespace WebApiDemo.Filters

{

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

// Check if the AllowAnonymous attribute is present on the action or controller

var allowAnonymous = context.ActionDescriptor.EndpointMetadata.OfType<AllowAnonymousAttribute>().Any();

if (allowAnonymous)

{

base.OnActionExecuting(context);

return; // Skip authentication for anonymous actions

}

if (!context.HttpContext.Request.Headers.ContainsKey("Authorization"))

{

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

return;

}

string authorizationHeader = context.HttpContext.Request.Headers["Authorization"];

if (!authorizationHeader.Contains("Bearer", System.StringComparison.OrdinalIgnoreCase))

{

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

return;

}

// If everything is fine, continue with the action execution

base.OnActionExecuting(context);

}

}

}

**3. Custom Exception filter:**

**Filters/CustomExceptionFilter.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.AspNetCore.Mvc.WebApiCompatShim; // Required for ExceptionResult

using System;

using System.IO;

namespace WebApiDemo.Filters

{

public class CustomExceptionFilter : ExceptionFilterAttribute

{

private readonly string \_logFilePath = "exception\_log.txt"; // Path to log file

public override void OnException(ExceptionContext context)

{

// Fetch the exception detail

var exception = context.Exception;

var exceptionDetails = $"Timestamp: {DateTime.UtcNow}\n" +

$"Request Path: {context.HttpContext.Request.Path}\n" +

$"Exception Type: {exception.GetType().Name}\n" +

$"Message: {exception.Message}\n" +

$"Stack Trace: {exception.StackTrace}\n" +

"---------------------------------------------------\n";

// Capture and write it to a File

try

{

File.AppendAllText(\_logFilePath, exceptionDetails);

}

catch (Exception logEx)

{

// Handle cases where logging itself fails (e.g., file permissions)

Console.WriteLine($"Error logging exception: {logEx.Message}");

}

var problemDetails = new ProblemDetails

{

Status = 500,

Title = "An error occurred while processing your request.",

Detail = "Please try again later. If the issue persists, contact support.",

Instance = context.HttpContext.Request.Path

};

// Modern approach:

context.Result = new ObjectResult(problemDetails)

{

StatusCode = 500

};

context.ExceptionHandled = true;

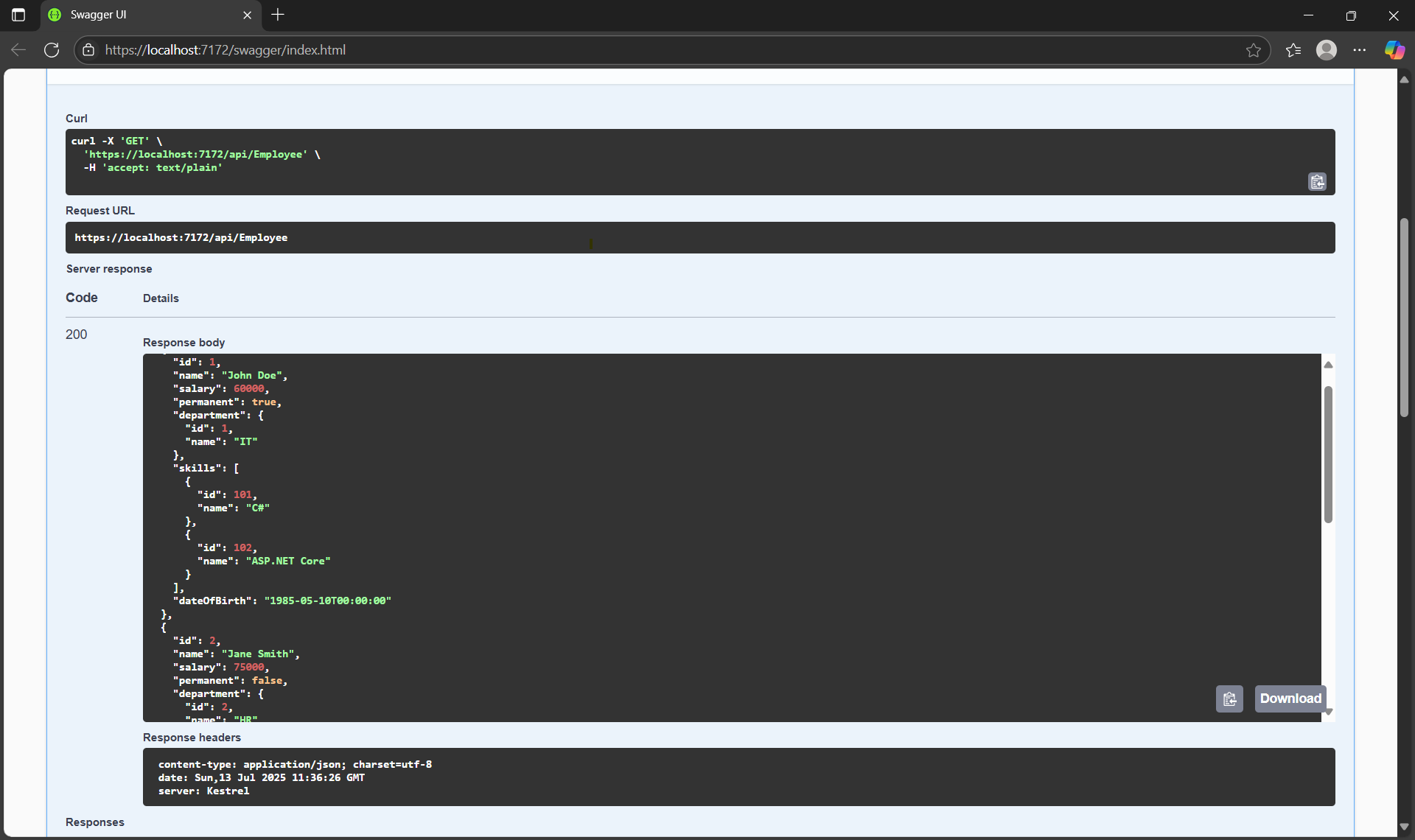
base.OnException(context);

}

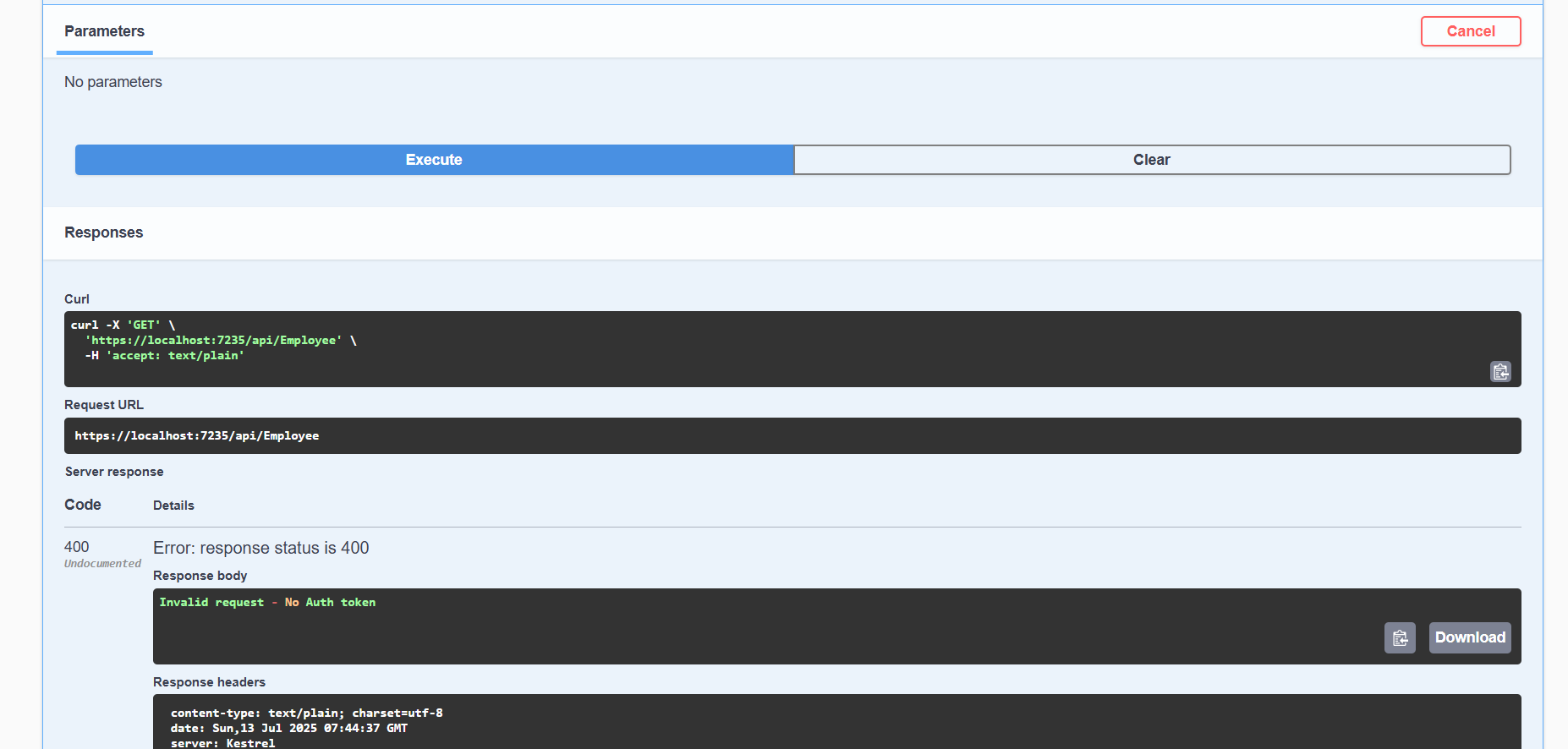
}

}

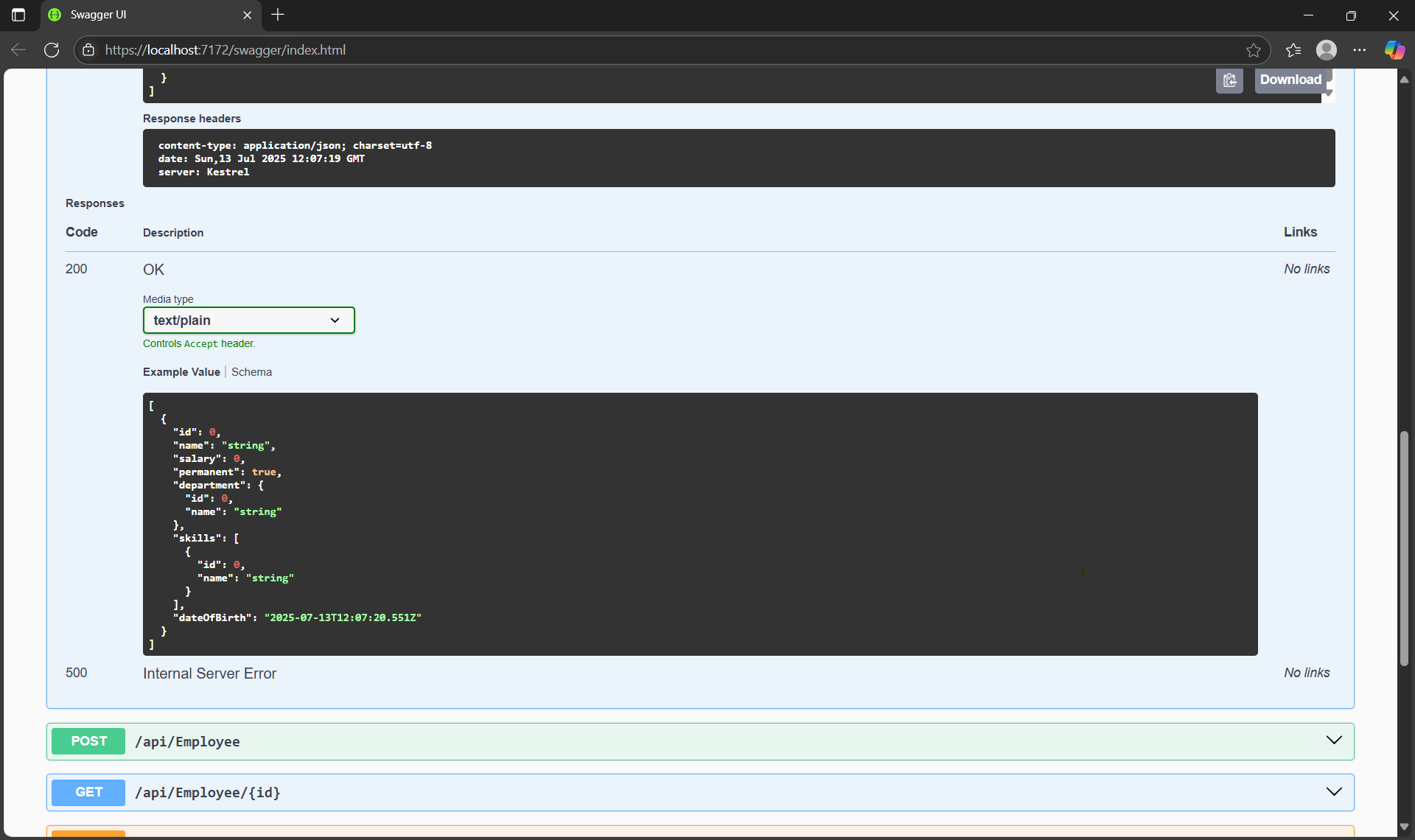
**List of employes: using swagger UI,**

****

**Objective 2:**



**Custom Exception filter:**

****

**Exercise : 4**

**Updating Controllers/EmployeeController.cs:**

**Code:**using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

using WebApiDemo.Filters; // Assuming your filter is in this namespace

using WebApiDemo.Models;

namespace WebApiDemo.Controllers

{

[ApiController]

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

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees;

static EmployeeController() // Static constructor to initialize data once

{

\_employees = GetStandardEmployeeList();

}

// Private method to generate initial hardcoded employee data

private static List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

Salary = 60000,

Permanent = true,

Department = new Department { Id = 1, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 101, Name = "C#" },

new Skill { Id = 102, Name = "ASP.NET Core" }

},

DateOfBirth = new DateTime(1985, 5, 10)

},

new Employee

{

Id = 2,

Name = "Jane Smith",

Salary = 75000,

Permanent = false,

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

Skills = new List<Skill>

{

new Skill { Id = 201, Name = "Recruitment" },

new Skill { Id = 202, Name = "Payroll" }

},

DateOfBirth = new DateTime(1990, 8, 15)

},

new Employee

{

Id = 3,

Name = "Peter Jones",

Salary = 50000,

Permanent = true,

Department = new Department { Id = 1, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 103, Name = "JavaScript" },

new Skill { Id = 104, Name = "React" }

},

DateOfBirth = new DateTime(1992, 2, 20)

}

};

}

[HttpGet("GetStandard")]

[AllowAnonymous]

[ProducesResponseType(typeof(List<Employee>), 200)]

[ProducesResponseType(500)]

public ActionResult<List<Employee>> GetStandard()

{

return \_employees;

}

// GET: api/Employee/{id}

[HttpGet("{id}")]

[ProducesResponseType(typeof(Employee), 200)]

[ProducesResponseType(400)] // For invalid ID scenarios (though our current impl only does 404)

[ProducesResponseType(404)]

public ActionResult<Employee> Get(int id)

{

if (id <= 0)

{

return BadRequest("Invalid employee ID.");

}

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

if (employee == null)

{

return NotFound($"Employee with ID {id} not found.");

}

return employee;

}

[HttpPost]

[ProducesResponseType(typeof(Employee), 201)] // 201 Created

[ProducesResponseType(400)] // 400 Bad Request

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

{

if (newEmployee == null)

{

return BadRequest("Employee data is null.");

}

// Basic validation

if (string.IsNullOrWhiteSpace(newEmployee.Name) || newEmployee.Salary <= 0)

{

return BadRequest("Employee name and salary are required.");

}

// Assign a new ID (simulate auto-increment from a database)

newEmployee.Id = \_employees.Any() ? \_employees.Max(e => e.Id) + 1 : 1;

\_employees.Add(newEmployee);

// Return 201 Created with the newly created employee and a Location header

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

}

[HttpPut("{id}")]

[ProducesResponseType(typeof(Employee), 200)] // 200 OK (Returning updated employee)

[ProducesResponseType(400)] // 400 Bad Request

[ProducesResponseType(404)] // 404 Not Found (Though our logic maps to 400 for consistency)

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

{

// Step 1: Check if the id value is lesser than or equal to 0. If true, throw BadRequest.

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

// Ensure the ID in the URL matches the ID in the request body (if provided)

if (updatedEmployee == null || updatedEmployee.Id != id)

{

// You might want to be more specific here, e.g., if (updatedEmployee.Id != id) return BadRequest("ID mismatch");

return BadRequest("Invalid employee data or ID mismatch.");

}

// Step 2: Check if the employee with the given ID exists in the hardcoded list.

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

if (existingEmployee == null)

{

// If not available in the list, throw BadRequest action result.

return BadRequest($"Invalid employee id: Employee with ID {id} not found.");

}

// Step 3: If the id value is valid and employee exists, update the hardcoded list.

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills;

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

// Step 4: Filter the employee list data for the input id and return that as the output.

return Ok(existingEmployee); // Return 200 OK with the updated employee object.

}

// DELETE: api/Employee/{id}

// Deletes an Employee record

[HttpDelete("{id}")]

[ProducesResponseType(204)] // 204 No Content

[ProducesResponseType(400)] // 400 Bad Request

[ProducesResponseType(404)] // 404 Not Found

public IActionResult Delete(int id)

{

if (id <= 0)

{

return BadRequest("Invalid employee id.");

}

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

if (employeeToRemove == null)

{

return NotFound($"Employee with ID {id} not found.");

}

\_employees.Remove(employeeToRemove);

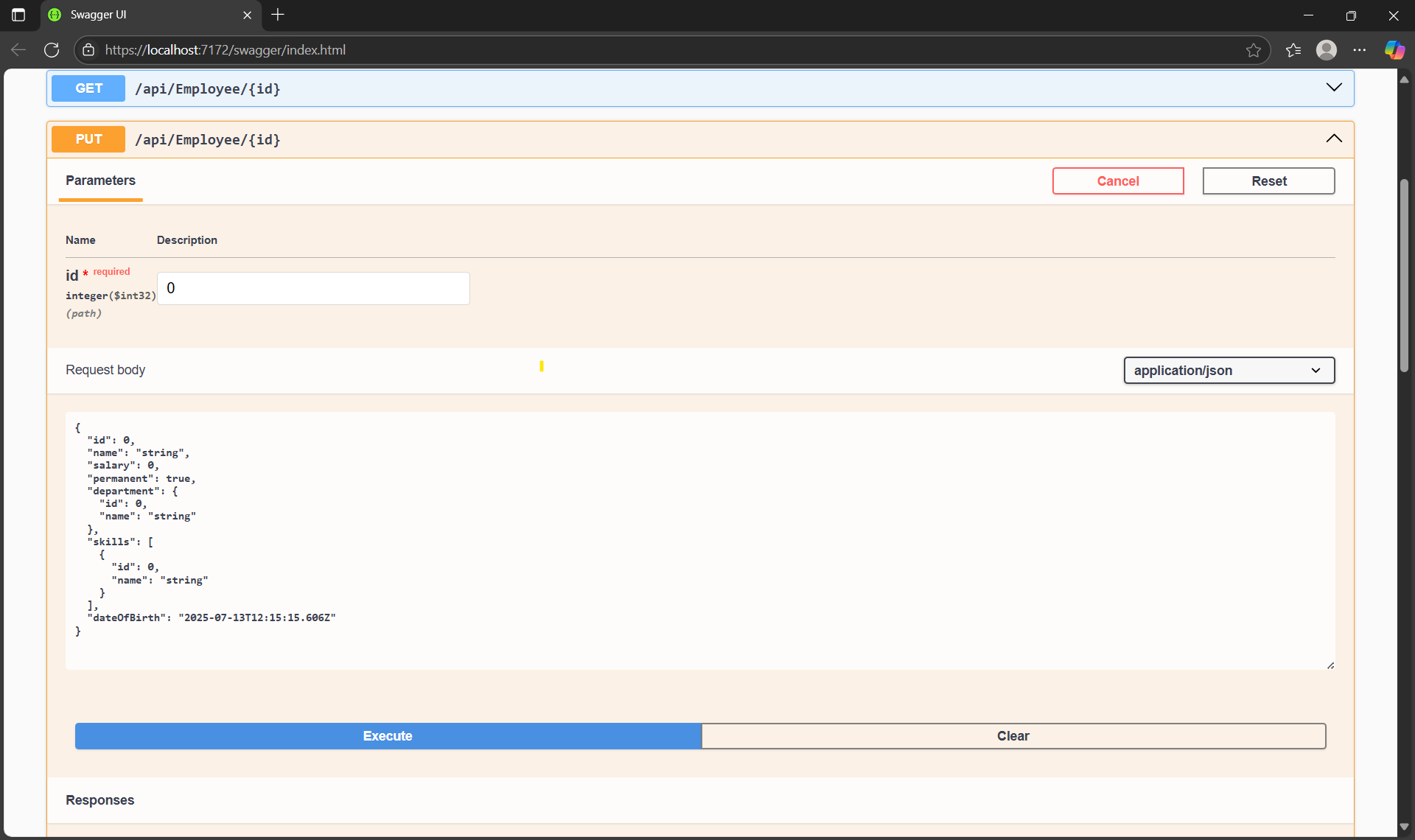
return NoContent(); // 204 No Content indicates successful deletion with no response body.

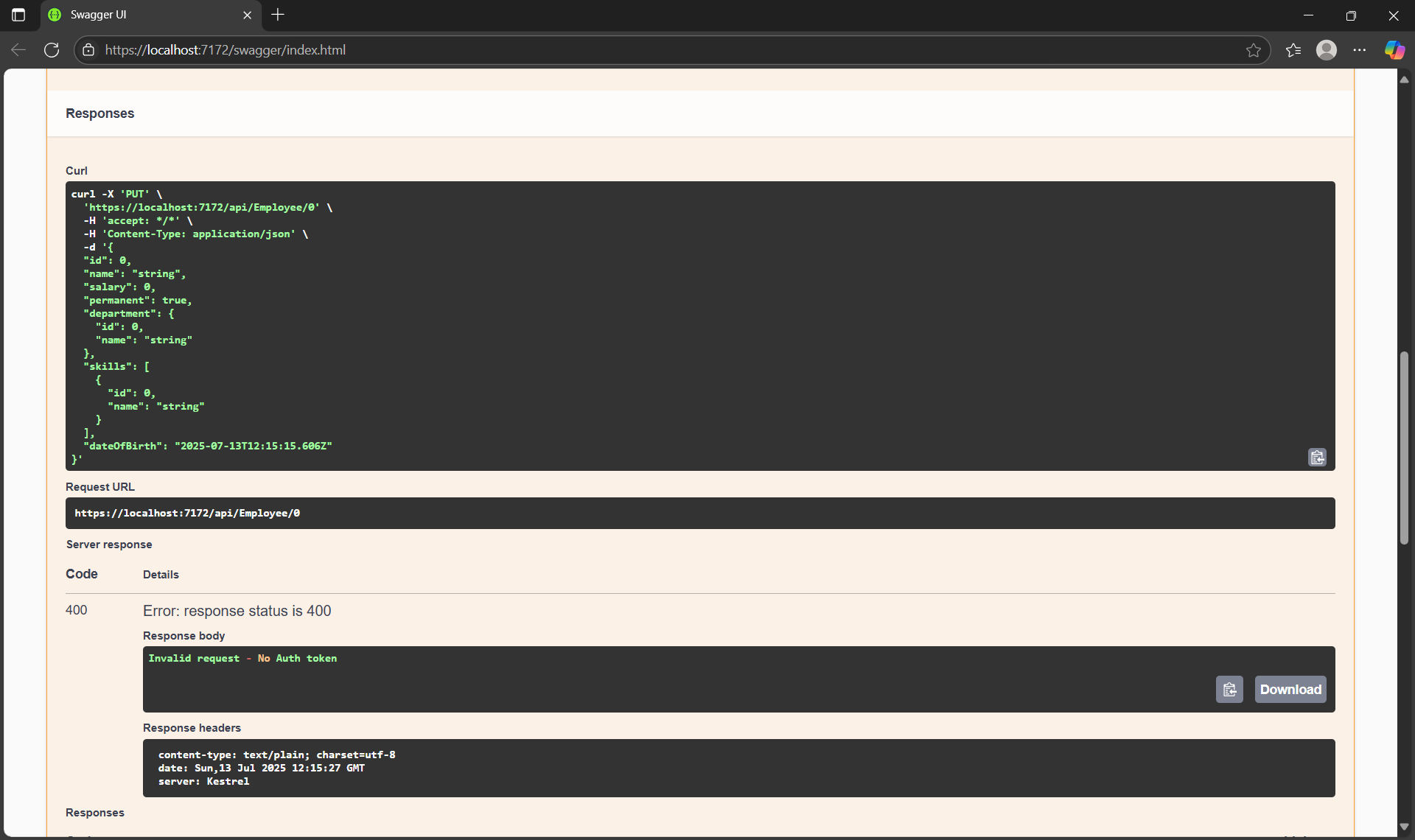
}

}

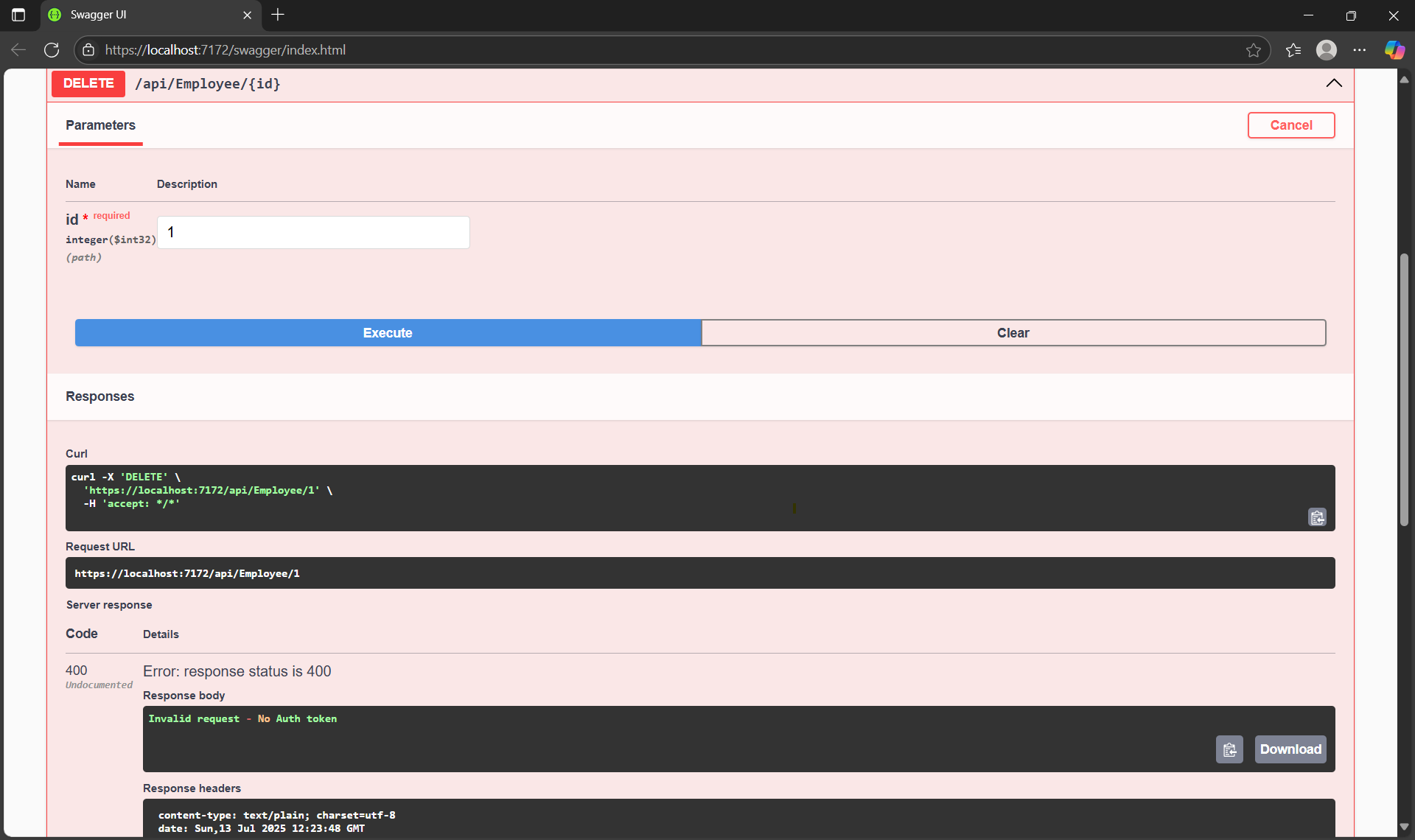
**}**

**Using swagger test the CRUD operations:**

****

****

**Delete :**

****

**By using the swagger performing CRUD operations.**

**Exercise : 5**

**What is CORS?**

CORS (Cross-Origin Resource Sharing) is a security feature implemented by web browsers. It restricts web pages from making requests to a domain different from the one that served the original web page. This is part of the Same-Origin Policy (SOP), a fundamental browser security measure designed to prevent malicious scripts on one website from accessing sensitive data on another.

When your front-end application (e.g., running on http://localhost:3000 with React/Angular/Vue) tries to communicate with your Web API (e.g., running on https://localhost:7000), the browser will block these requests by default because the origins (protocol, domain, and port) are different. CORS provides a standardized way for the server (your Web API) to tell the browser that it is permissible to allow requests from specific foreign origins.

How to enable CORS:

a. Install CORS NuGet Package: The Microsoft.AspNetCore.Cors package is usually included by default in modern ASP.NET Core Web API templates. If not, you can install it via NuGet Package Manager or the .NET CLI:

**Step:1**

**Creating JSON web token:**

**In ConfigureServices method**

**string securityKey = "mysuperdupersecret";**

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

**services.AddAuthentication(x =>**

**{**

**x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;**

**x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;**

**x.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;**

**})**

**.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>**

**{**

**x.TokenValidationParameters = new TokenValidationParameters**

**{**

**//what to validate**

**ValidateIssuer = true,**

**ValidateAudience = true,**

**ValidateLifetime = true,**

**ValidateIssuerSigningKey = true,**

**//setup validate data**

**ValidIssuer = "mySystem",**

**ValidAudience = "myUsers",**

**IssuerSigningKey = symmetricSecurityKey**

**};**

**});**

* **In Configure method**

**app.UseAuthentication();**

**This is to enable the JWT authentication in .Net core**

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

**private string GenerateJSONWebToken(int userId, string userRole)**

**{**

**var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret"));**

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

**var claims = new List<Claim>**

**{**

**new Claim(ClaimTypes.Role, userRole),**

**new Claim("UserId", userId.ToString())**

**};**

**var token = new JwtSecurityToken(**

**issuer: "mySystem",**

**audience: "myUsers",**

**claims: claims,**

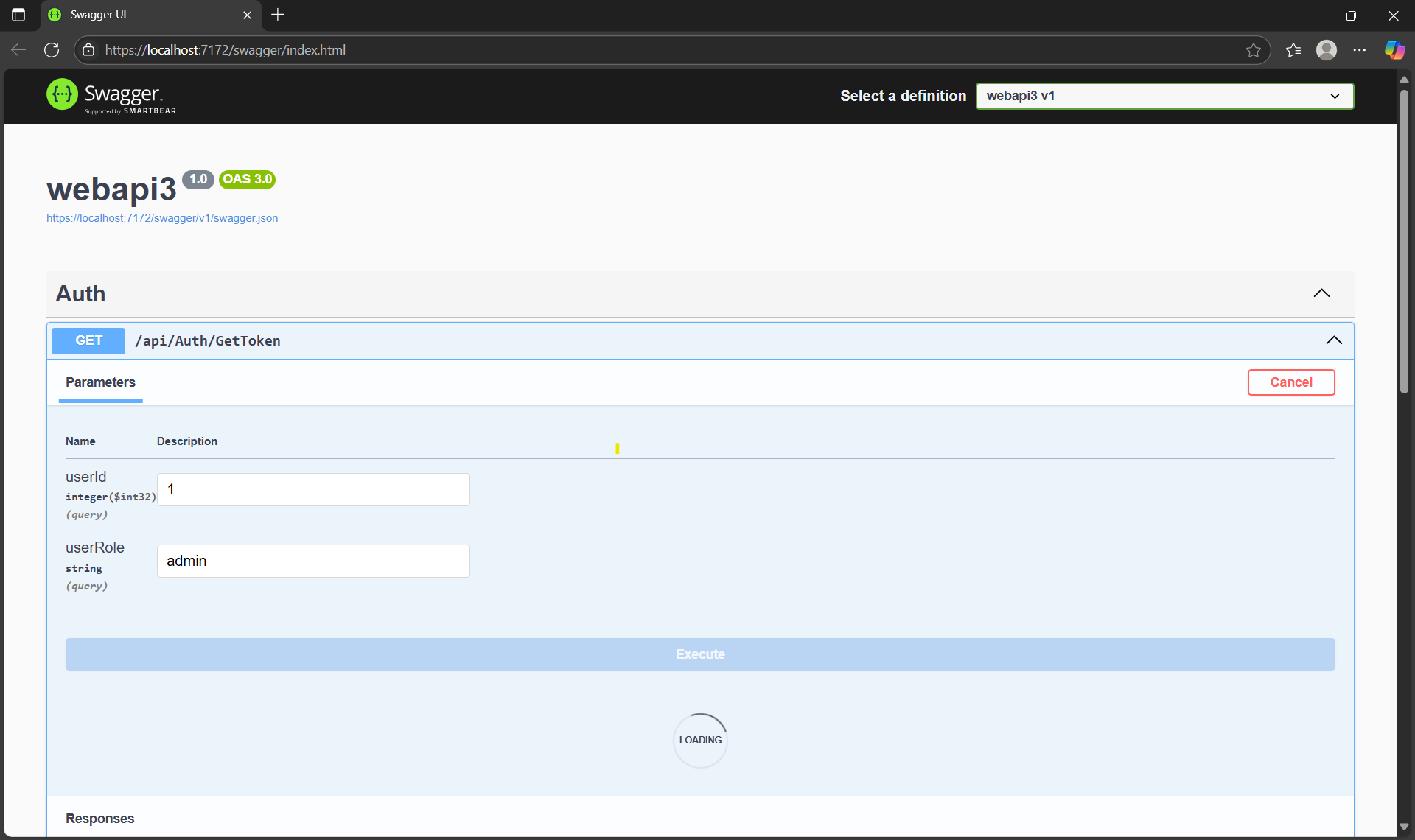
**expires: DateTime.Now.AddMinutes(10),**

**signingCredentials: credentials);**

**return new JwtSecurityTokenHandler().WriteToken(token);**

**}**

**Execution in swagger :**

****

**After JWT generation Authorize .**

**Modify [Authorize] attribute in EmployeeController:**

// ...

[ApiController]

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

[Authorize(Roles = "POC")] // <--- Only users with the "POC" role can access this controller

public class EmployeeController : ControllerBase

{

// ...

}

**Check for JWT Expiration :**

// ...

private string GenerateJSONWebToken(int userId, string userRole)

{

// ... other token creation code ...

var token = new JwtSecurityToken(

// ... issuer, audience, claims ...

expires: DateTime.Now.AddMinutes(2), // <--- Changed to 2 minutes for testing expiration

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

**Scenario 2: Require 'Admin' or 'POC' Role:**

// Controllers/EmployeeController.cs

[ApiController]

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

[Authorize(Roles = "Admin,POC")] // <--- Users with EITHER "Admin" OR "POC" roles can access

public class EmployeeController : ControllerBase

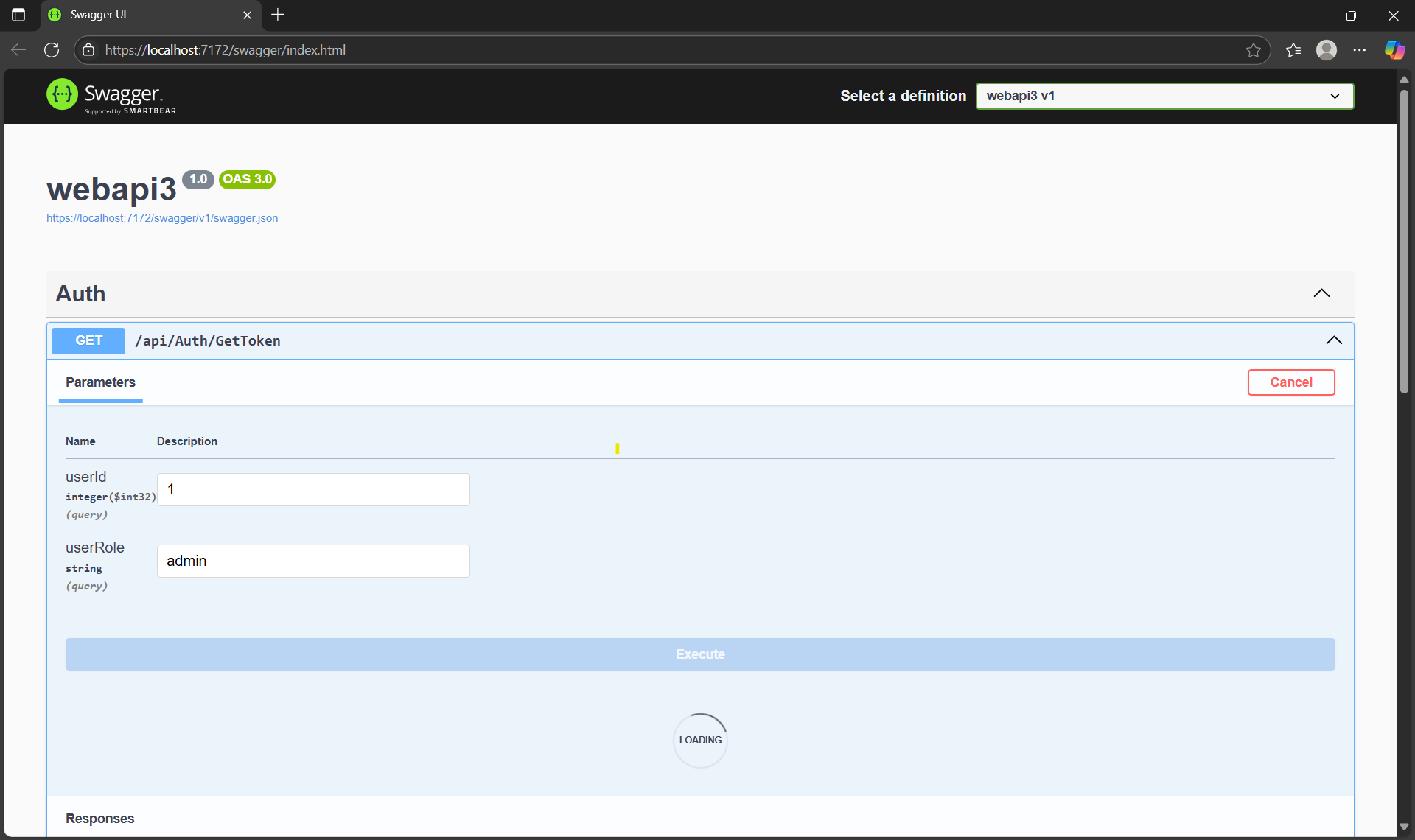
{

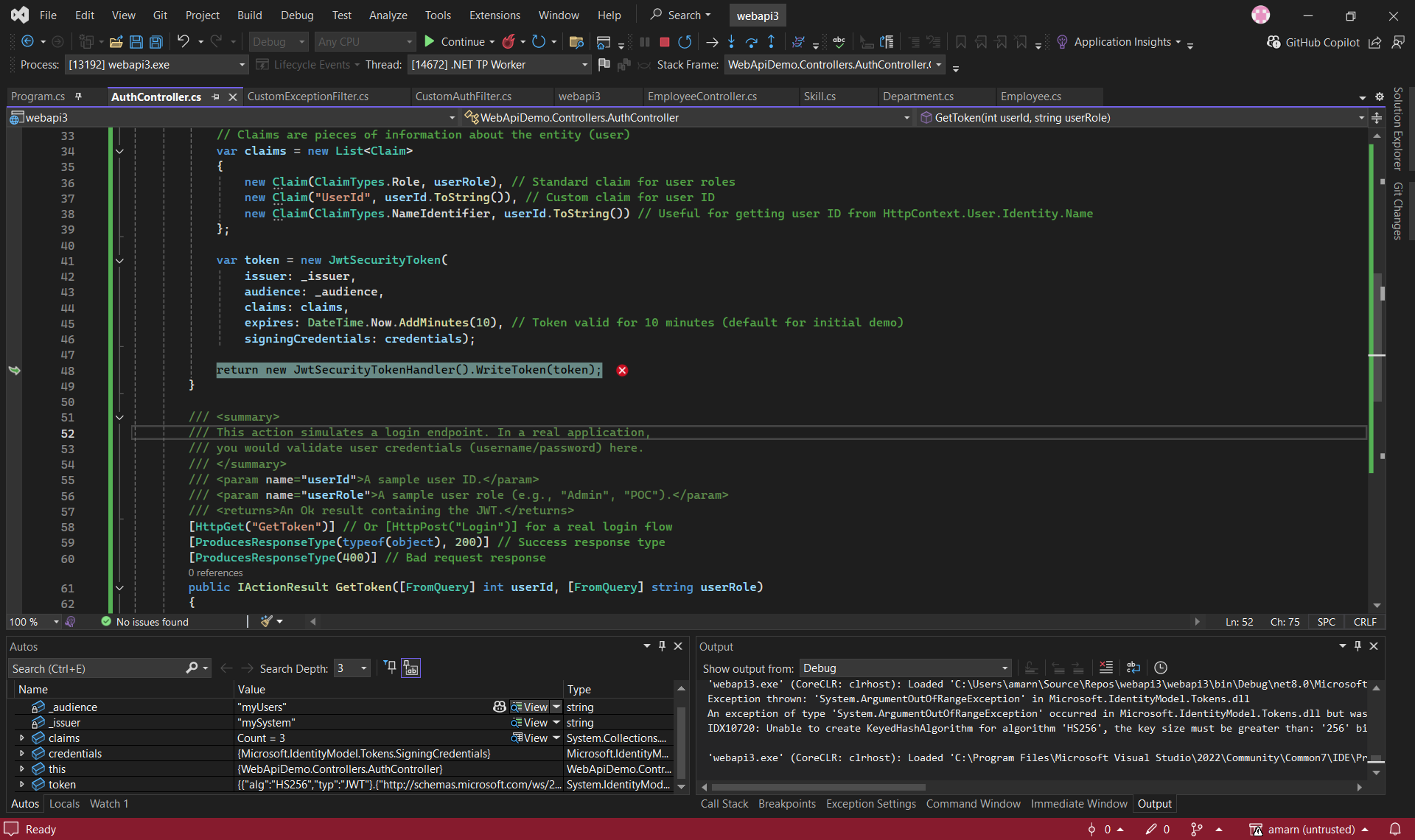
// ...

}

**Final Test with Postman:**

Restart API.

****

****