**Cognizant Digital Nurture 4.0 – DotNet FSE Deep Skilling Program**

Pranjal Yadav(22052918)

Week 4 – ASP.NET Core 8.0 Web API Labs

# Lab 1 - First Web API using .NET Core 8.0

## Objective

- Explain the concept of RESTful Web Service, Web API & Microservice.  
- Describe HTTP Request, HTTP Response, Action Verbs, and HttpStatusCodes.  
- Demonstrate creation of a simple Web API using .NET 8.0 Core Web API template.  
- Explore project structure: appsettings.json, launchSettings.json, Program.cs, and Controller.

## Step 1 - Creating the Web API Project

Command used:

dotnet new webapi -n Lab1\_WebApiProject -f net8.0

This creates a new ASP.NET Core 8.0 Web API with a controller for Read/Write operations.

## Step 2 - Running the Web API

Command used:

dotnet run

Application hosted at: http://localhost:5046/swagger

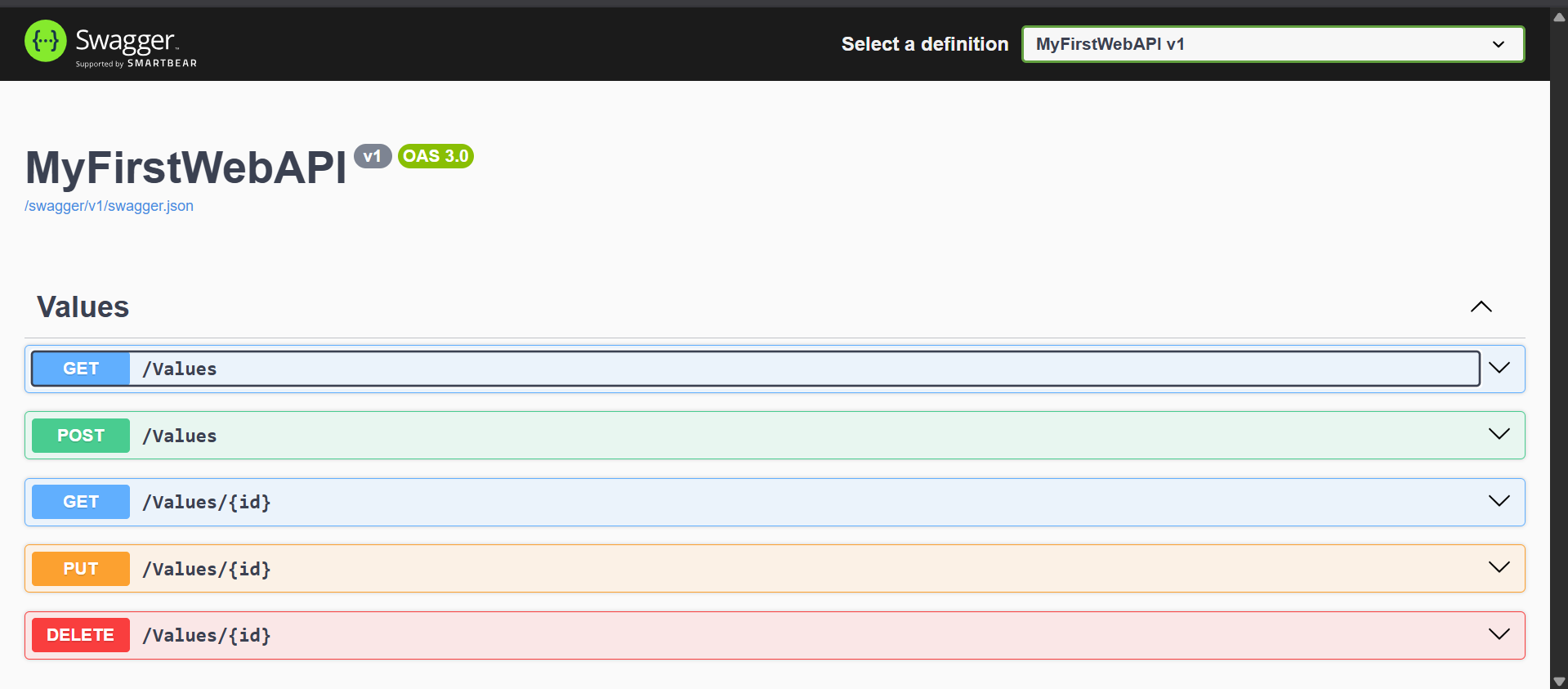
## Step 3 - Modified Program.cs

Swagger UI enabled for all environments and given a custom title:

using Microsoft.OpenApi.Models;  
  
var builder = WebApplication.CreateBuilder(args);  
  
builder.Services.AddControllers();  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen(c =>  
{  
 c.SwaggerDoc("v1", new OpenApiInfo { Title = "MyFirstWebAPI", Version = "v1" });  
});  
  
var app = builder.Build();  
  
app.UseSwagger();  
app.UseSwaggerUI(c =>  
{  
 c.SwaggerEndpoint("/swagger/v1/swagger.json", "MyFirstWebAPI v1");  
});  
  
app.UseAuthorization();  
app.MapControllers();  
app.Run();

## Step 4 - Swagger UI Result

The following image shows that the Web API is running successfully with GET, POST, PUT, and DELETE endpoints:



# Lab 2: Testing Web API with Postman

## Objective

- Learn how to test ASP.NET Core 8.0 Web API using Postman.  
- Understand the structure of requests (GET, POST, PUT, DELETE).  
- Validate JSON request and response formats.

## Step 1: Project Setup

Used the existing Lab1\_WebApiProject or created a similar API project exposing CRUD operations.

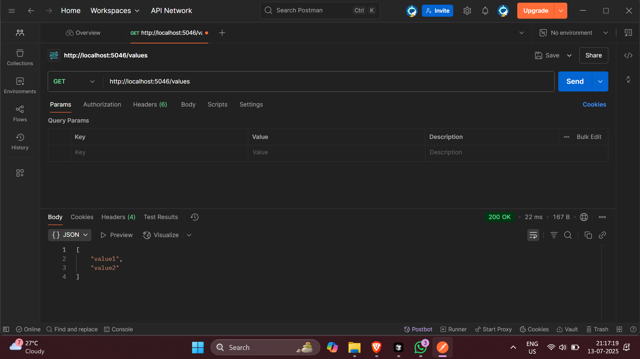
## Step 2: Endpoint Testing Using Postman

We tested the following endpoints using Postman:  
- GET /weatherforecast – Returns a list of weather forecasts.  
- POST /employee – Adds a new employee with a JSON body.  
- PUT /employee/{id} – Updates an existing employee.  
- DELETE /employee/{id} – Deletes an employee.

## Step 3: Sample Code for Controller (EmployeeController.cs)

using Microsoft.AspNetCore.Mvc;  
using System.Collections.Generic;  
using System.Linq;  
  
namespace Lab1\_WebApiProject.Controllers  
{  
 [ApiController]  
 [Route("[controller]")]  
 public class EmployeeController : ControllerBase  
 {  
 private static List<Employee> employees = new List<Employee>  
 {  
 new Employee { Id = 1, Name = "Alice", Role = "Developer" },  
 new Employee { Id = 2, Name = "Bob", Role = "Tester" }  
 };  
  
 [HttpGet]  
 public IActionResult GetAll() => Ok(employees);  
  
 [HttpGet("{id}")]  
 public IActionResult GetById(int id)  
 {  
 var emp = employees.FirstOrDefault(e => e.Id == id);  
 if (emp == null) return NotFound();  
 return Ok(emp);  
 }  
  
 [HttpPost]  
 public IActionResult Create(Employee emp)  
 {  
 emp.Id = employees.Count + 1;  
 employees.Add(emp);  
 return CreatedAtAction(nameof(GetById), new { id = emp.Id }, emp);  
 }  
  
 [HttpPut("{id}")]  
 public IActionResult Update(int id, Employee emp)  
 {  
 var existing = employees.FirstOrDefault(e => e.Id == id);  
 if (existing == null) return NotFound();  
  
 existing.Name = emp.Name;  
 existing.Role = emp.Role;  
 return NoContent();  
 }  
  
 [HttpDelete("{id}")]  
 public IActionResult Delete(int id)  
 {  
 var emp = employees.FirstOrDefault(e => e.Id == id);  
 if (emp == null) return NotFound();  
  
 employees.Remove(emp);  
 return NoContent();  
 }  
 }  
  
 public class Employee  
 {  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public string Role { get; set; }  
 }  
}

## Step 4: Postman Screenshot (GET)



# Lab 3: Custom Authorization and Exception Filters in ASP.NET Core 8 Web API

In this lab, we implemented a Web API using a custom model class `Employee`, with nested objects `Department` and `Skill`. We also created:  
- A custom authorization filter using `ActionFilterAttribute`  
- A custom exception filter using `IExceptionFilter`  
- Swagger to verify the API endpoints

Program.cs:

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Add Swagger/OpenAPI support

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "ASP.NET Core 8 Web API for Cognizant Digital Nurture 4.0 – DotNet FSE Deep Skilling Program by Pranjal Yadav",

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

Contact = new OpenApiContact

{

Name = "Pranjal Yadav",

Email = "pranjal@example.com",

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

},

License = new OpenApiLicense

{

Name = "Use under LICX",

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

}

});

});

var app = builder.Build();

// Enable middleware for Swagger

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

c.RoutePrefix = string.Empty; // Open Swagger UI at root (http://localhost:<port>/)

});

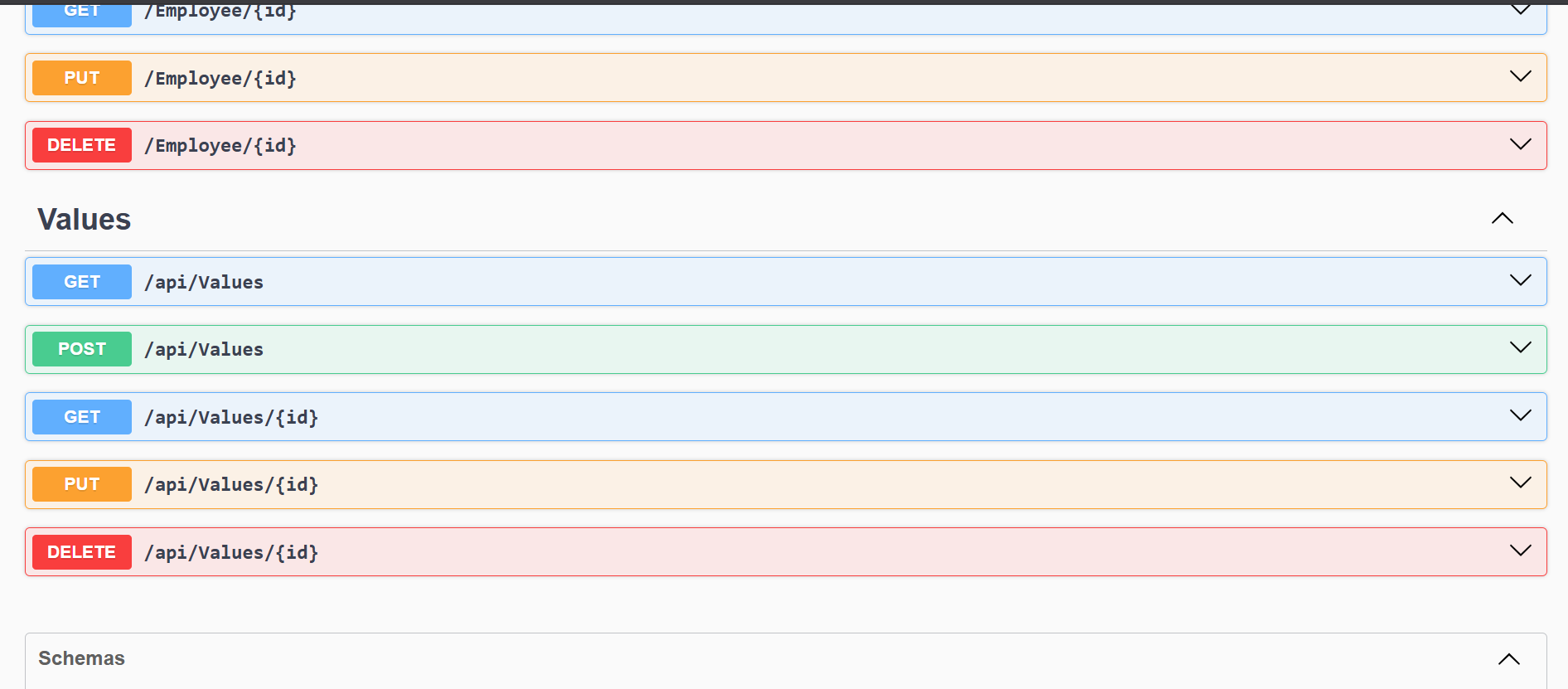
}

app.UseAuthorization();

app.MapControllers();

app.Run();

## Swagger UI Output



**Lab 4: Update Employee with PUT Method**

#### ****Objective****

* Demonstrate creation of an Action method to perform data update operation.
* Use [FromBody] attribute and extract JSON data into custom model.
* Use hardcoded data to validate and update employee list.
* Test the endpoint using Swagger and Postman.

#### ****Step 1: PUT Endpoint with Validation****

Update employee information using Web API PUT action method. The method performs the following validations:

* If id <= 0: returns BadRequest("Invalid employee id")
* If id > 0 but not found in list: returns BadRequest("Invalid employee id")
* If valid, updates employee and returns updated object

#### ****Step 2: Code Snippet****

Paste this inside your EmployeeController.cs:

csharp

CopyEdit

[HttpPut("{id}")]

public ActionResult<Employee> UpdateEmployee(int id, [FromBody] Employee input)

{

if (id <= 0)

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

if (existing == null)

return BadRequest("Invalid employee id");

existing.Name = input.Name;

existing.Role = input.Role;

return Ok(existing);

}

#### ****Step 3: Swagger Screenshot:****

#### return BadRequest("Invalid employee id");

## Lab 5 – Securing Web API using JWT, Roles & CORS

### 🎯 ****Objective****

* Understand and enable **CORS** for local access to Web API
* Implement **JWT authentication** using AuthController
* Use [Authorize] with **roles** like Admin and POC to protect endpoints
* Test role-based access and token expiry using **Swagger** and **Postman**

Step 1: Enable CORS and JWT in Program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "ASP.NET Core 8 Web API for Cognizant Digital Nurture 4.0 – DotNet FSE Deep Skilling Program by Pranjal Yadav",

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

Contact = new OpenApiContact

{

Name = "Pranjal Yadav",

Email = "pranjal@example.com",

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

},

License = new OpenApiLicense

{

Name = "Use under LICX",

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

}

});

});

// Enable CORS

builder.Services.AddCors(options =>

{

options.AddDefaultPolicy(policy =>

{

policy.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod();

});

});

// JWT setup

string securityKey = "mysuperdupersecret\_key\_which\_is\_long\_enough\_to\_be\_secure\_2025";

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

builder.Services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(x =>

{

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey

};

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

c.RoutePrefix = string.Empty;

});

}

app.UseCors();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

Step 2: Create AuthController.cs to Generate Token

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace Lab5\_WebApiProject.Controllers

{

[ApiController]

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

public class AuthController : ControllerBase

{

[AllowAnonymous]

[HttpGet]

public IActionResult GetToken()

{

var token = GenerateJSONWebToken(1, "Admin"); // use "POC" to test role access

return Ok(new { token });

}

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret\_key\_which\_is\_long\_enough\_to\_be\_secure\_2025"));

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), // Use 2 mins to test expiration

signingCredentials: credentials

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

Step 3: Update EmployeeController.cs to Require JWT + Role

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Linq;

namespace Lab5\_WebApiProject.Controllers

{

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

[ApiController]

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

public class EmployeeController : ControllerBase

{

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

{

new Employee { Id = 1, Name = "Alice", Role = "Developer" },

new Employee { Id = 2, Name = "Bob", Role = "Tester" }

};

[HttpGet]

public IActionResult GetAll()

{

return Ok(employees);

}

}

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

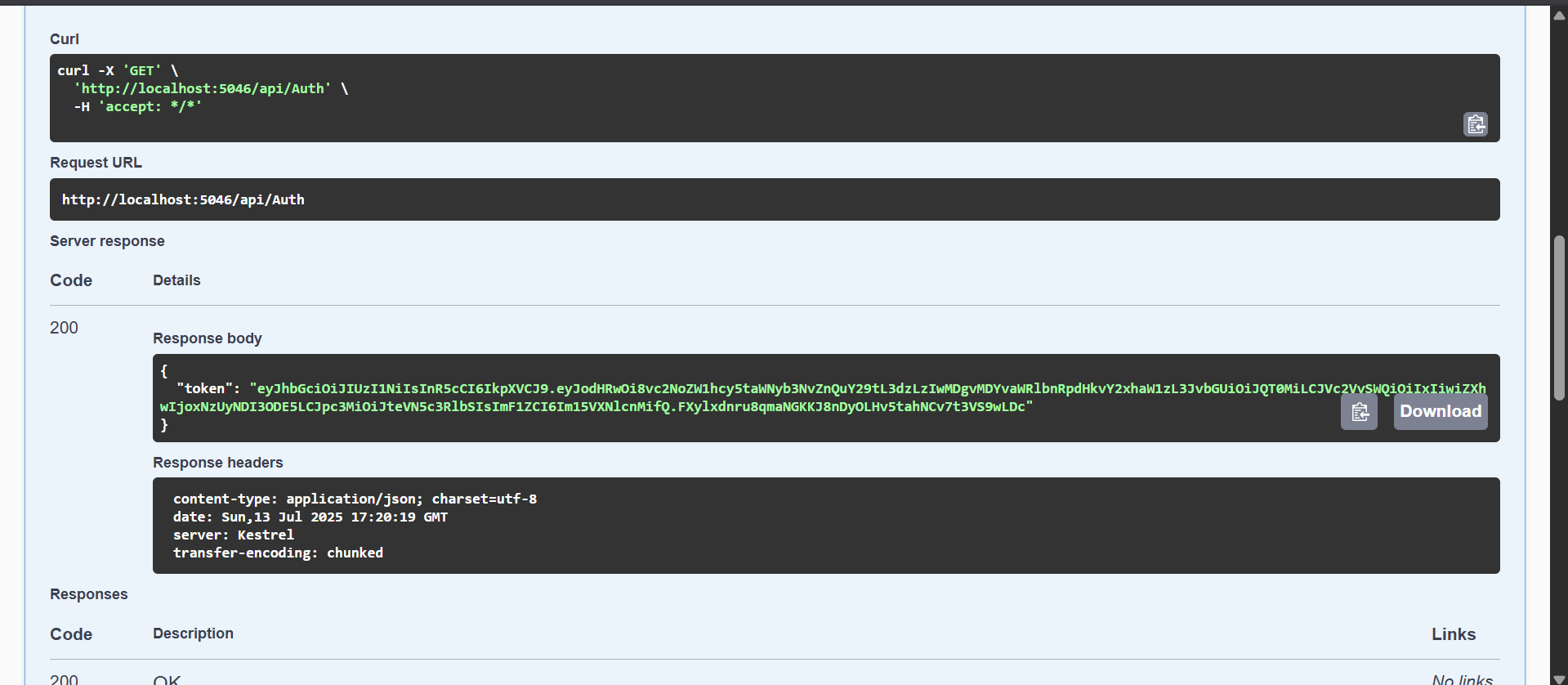
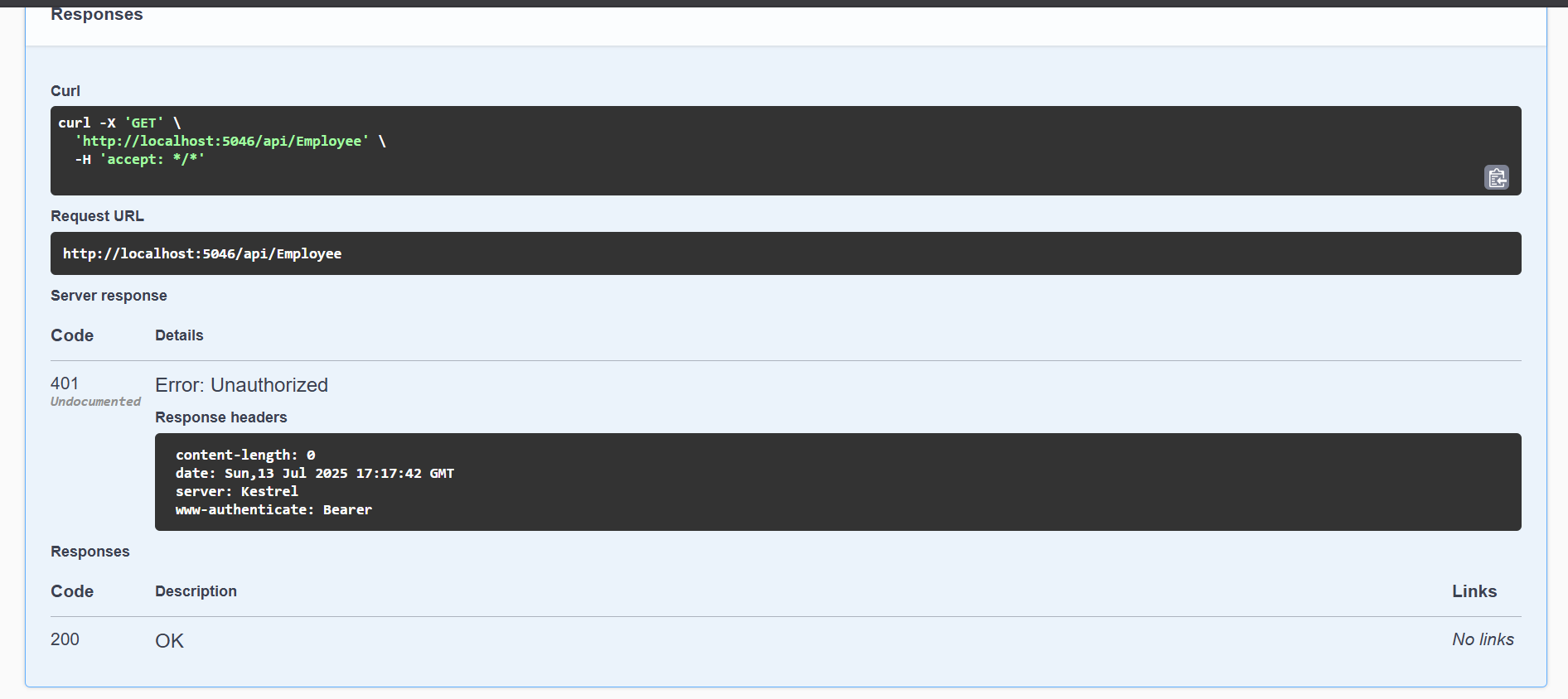
public string Role { get; set; }

}

}

### Step 4: Testing via Swagger & Postman

Unauthorized (No Token) :

Token Generation :

Role Mismatch :

