**HANDSON 1:**

RESTful Web Service, Web API & Microservice

# 1. RESTful Web Service, Web API & Microservice

RESTful Web Service:  
• REST stands for Representational State Transfer — an architectural style for building services over HTTP.  
• REST uses stateless communication — each request is independent; the server does not store client context.  
• REST resources are identified by URIs.  
• Commonly uses JSON or XML to represent data, but is not limited to XML.  
• Uses standard HTTP methods: GET, POST, PUT, DELETE.

Web API:  
• A Web API is an application programming interface accessible over the web using HTTP.  
• In .NET, a Web API is usually a RESTful service.  
• It lets clients (web apps, mobile apps, other services) call endpoints to read/write data.

Microservice:  
• A Microservice is an independently deployable, small, focused piece of an application.  
• Each microservice has its own data & logic.  
• Microservices often expose a RESTful Web API.  
• Many microservices together make a larger system — easier to scale & maintain.

Key REST Features:  
• Representational State Transfer: Data (resources) is transferred in a standard format (JSON/XML).  
• Stateless: No session state stored on server between requests.  
• Messages: Requests/responses are standard HTTP messages.  
• Flexible Response: REST can return JSON, XML, plain text, HTML — not restricted to XML.  
• Difference — WebService vs WebAPI:  
 - WebService (older SOAP) mostly uses XML & WSDL.  
 - Web API (REST) is lighter, simpler — uses HTTP verbs, mostly JSON, easy to test & scale.

# 2. HttpRequest & HttpResponse

HttpRequest:  
• Represents everything sent by the client.  
• Contains: URL, headers, body, query string, cookies.  
• Example: a GET request asking for /api/values.

HttpResponse:  
• Represents everything sent back to the client.  
• Contains: status code, headers, body.  
• Example: the JSON { "id": 1, "name": "GPT" } with status 200 OK.

# 3. Types of Action Verbs

• HttpGet: Fetch resource(s).  
[HttpGet]  
public IActionResult Get() { ... }  
  
• HttpPost: Add new resource.  
[HttpPost]  
public IActionResult Post([FromBody] Data data) { ... }  
  
• HttpPut: Update existing resource.  
[HttpPut("{id}")]  
public IActionResult Put(int id, [FromBody] Data data) { ... }  
  
• HttpDelete: Delete resource.  
[HttpDelete("{id}")]  
public IActionResult Delete(int id) { ... }

# 4. Common HttpStatusCodes in WebAPI

• 200 OK: Request succeeded.  
• 400 BadRequest: Bad input data.  
• 401 Unauthorized: Auth required or failed.  
• 500 InternalServerError: Server failed to process request.  
Example:  
return Ok(data); // 200  
return BadRequest(); // 400  
return Unauthorized(); // 401  
return StatusCode(500); // 500

# 5. Simple WebAPI: Structure

Structure:  
• Program.cs or Startup.cs: Registers services.  
• Controllers/: Contains controllers.  
Example Controller:  
[ApiController]  
[Route("api/[controller]")]  
public class ValuesController : ControllerBase  
{  
 [HttpGet]  
 public IActionResult Get() => Ok(new[] { "value1", "value2" });  
  
 [HttpPost]  
 public IActionResult Post([FromBody] string value) => Ok(value);  
}  
✔️ Inherits ApiController → makes it a RESTful controller.  
✔️ Has action methods → mapped by [HttpGet], [HttpPost].

# 6. Configuration Files in WebAPI

• Startup.cs / Program.cs:  
 - Registers services, middleware (routing, DI).  
 - Example: builder.Services.AddControllers(); app.MapControllers();  
  
• appsettings.json:  
 - Stores config data — connection strings, keys.  
 - Example:  
 {  
 "ConnectionStrings": {  
 "DefaultConnection": "..."   
 },  
 "Logging": {  
 "LogLevel": {  
 "Default": "Information"  
 }  
 }  
 }  
  
• launchSettings.json:  
 - Dev-only file.  
 - Sets environment, application URL, profiles.  
 - Example: {  
 "profiles": {  
 "http": {  
 "applicationUrl": "http://localhost:5230"  
 }  
 }  
 }  
  
• .NET Framework 4.5: Web.config & Route.config:  
 - Web.config: Old .NET’s main config file. Stores connection strings, app keys.  
 - Route.config: Defines routes for classic ASP.NET MVC/WebAPI.  
 - Example:  
 routes.MapHttpRoute(  
 name: "DefaultApi",  
 routeTemplate: "api/{controller}/{id}",  
 defaults: new { id = RouteParameter.Optional }  
 );

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

**CODE:**

**ValueController.cs:**

using Microsoft.AspNetCore.Mvc;

namespace FirstWebApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class ValuesController : ControllerBase

    {

        [HttpGet]

        public IActionResult Get()

        {

            return Ok(new string[] { "value1", "value2" });

        }

        [HttpGet("{id}")]

        public IActionResult Get(int id)

        {

            return Ok($"value {id}");

        }

        [HttpPost]

        public IActionResult Post([FromBody] string value)

        {

            return Ok($"Posted: {value}");

        }

        [HttpPut("{id}")]

        public IActionResult Put(int id, [FromBody] string value)

        {

            return Ok($"Updated id {id} with {value}");

        }

        [HttpDelete("{id}")]

        public IActionResult Delete(int id)

        {

            return Ok($"Deleted id {id}");

        }

    }

}

**WeatherForecastController.cs:**

using Microsoft.AspNetCore.Mvc;

namespace FirstWebApi.Controllers

{

    [ApiController]

    [Route("[controller]")]

    public class WeatherForecastController : ControllerBase

    {

        private static readonly string[] Summaries = new[]

        {

            "Freezing", "Bracing", "Chilly", "Cool", "Mild",

            "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

        };

        [HttpGet]

        public IEnumerable<WeatherForecast> Get()

        {

            return Enumerable.Range(1, 5).Select(index => new WeatherForecast

            {

                Date = DateTime.Now.AddDays(index),

                TemperatureC = Random.Shared.Next(-20, 55),

                Summary = Summaries[Random.Shared.Next(Summaries.Length)]

            })

            .ToArray();

        }

    }

    public class WeatherForecast

    {

        public DateTime Date { get; set; }

        public int TemperatureC { get; set; }

        public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

        public string? Summary { get; set; }

    }

}

**WeatherForecast.cs:**

namespace FirstWebApi

{

    public class WeatherForecast

    {

        public DateTime Date { get; set; }

        public int TemperatureC { get; set; }

        public string? Summary { get; set; }

    }

}

**Program.cs:**

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI();

}

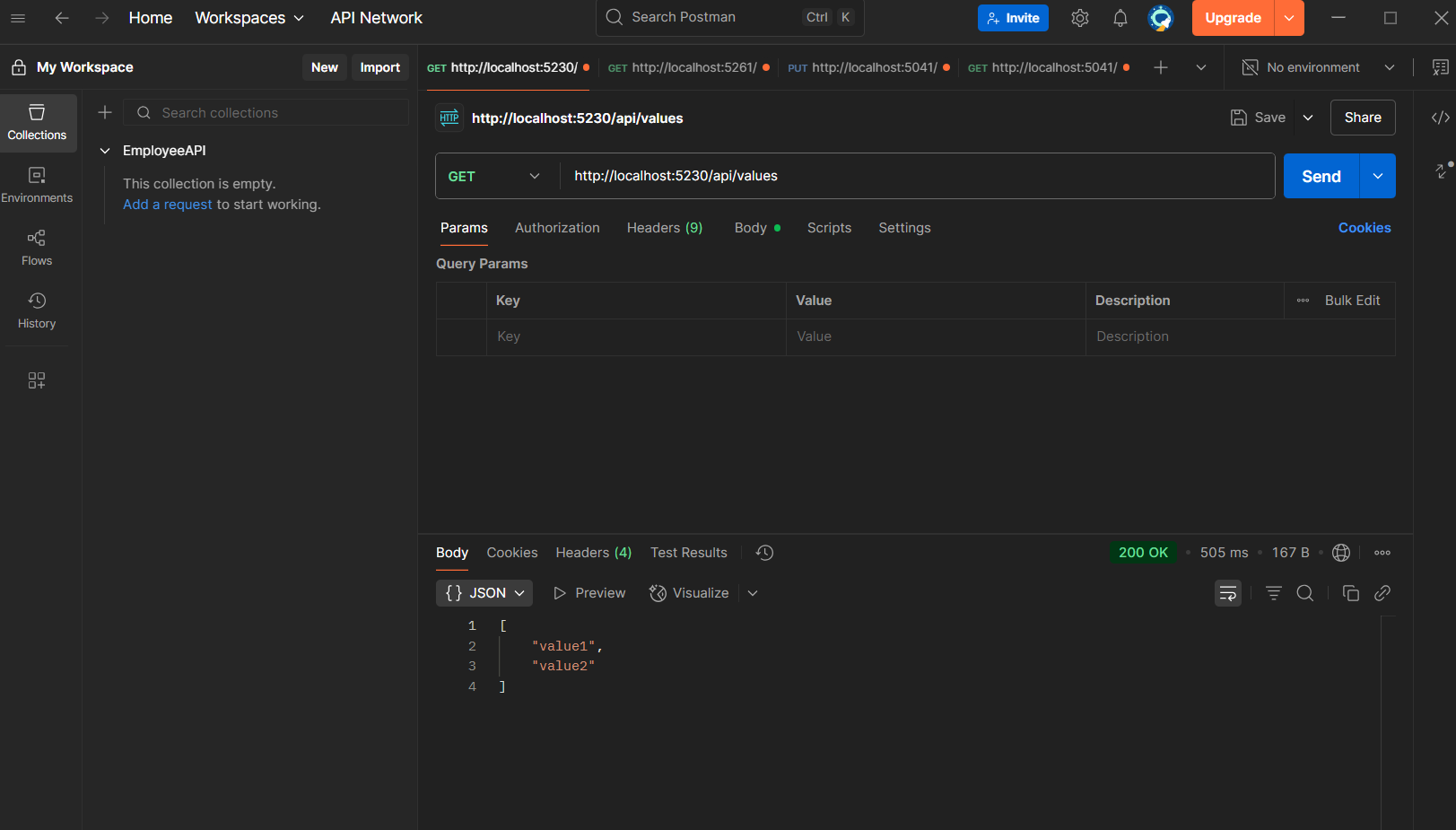
app.UseAuthorization();

app.MapControllers(); //

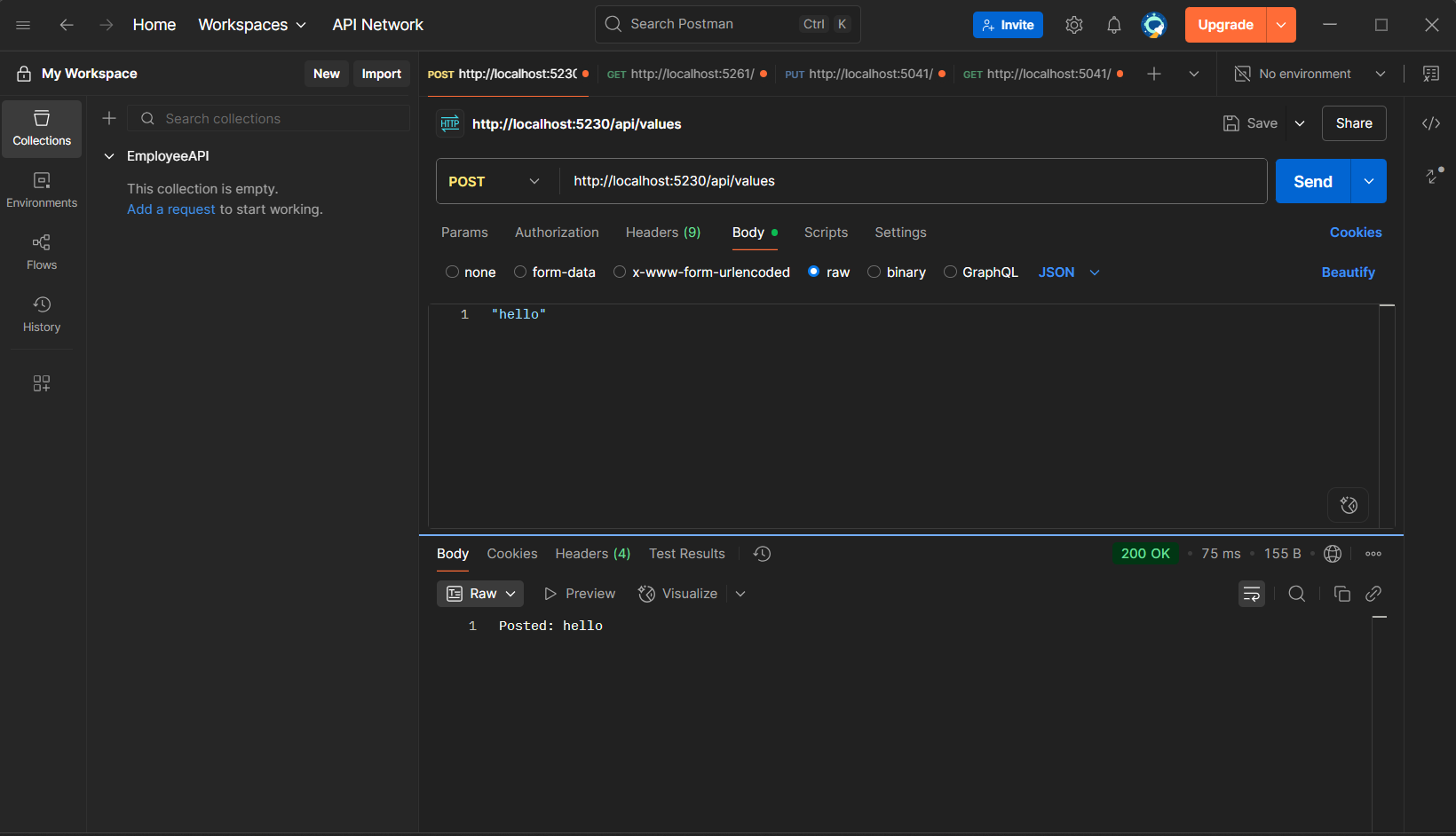
app.Run();

**OUTPUTS:**

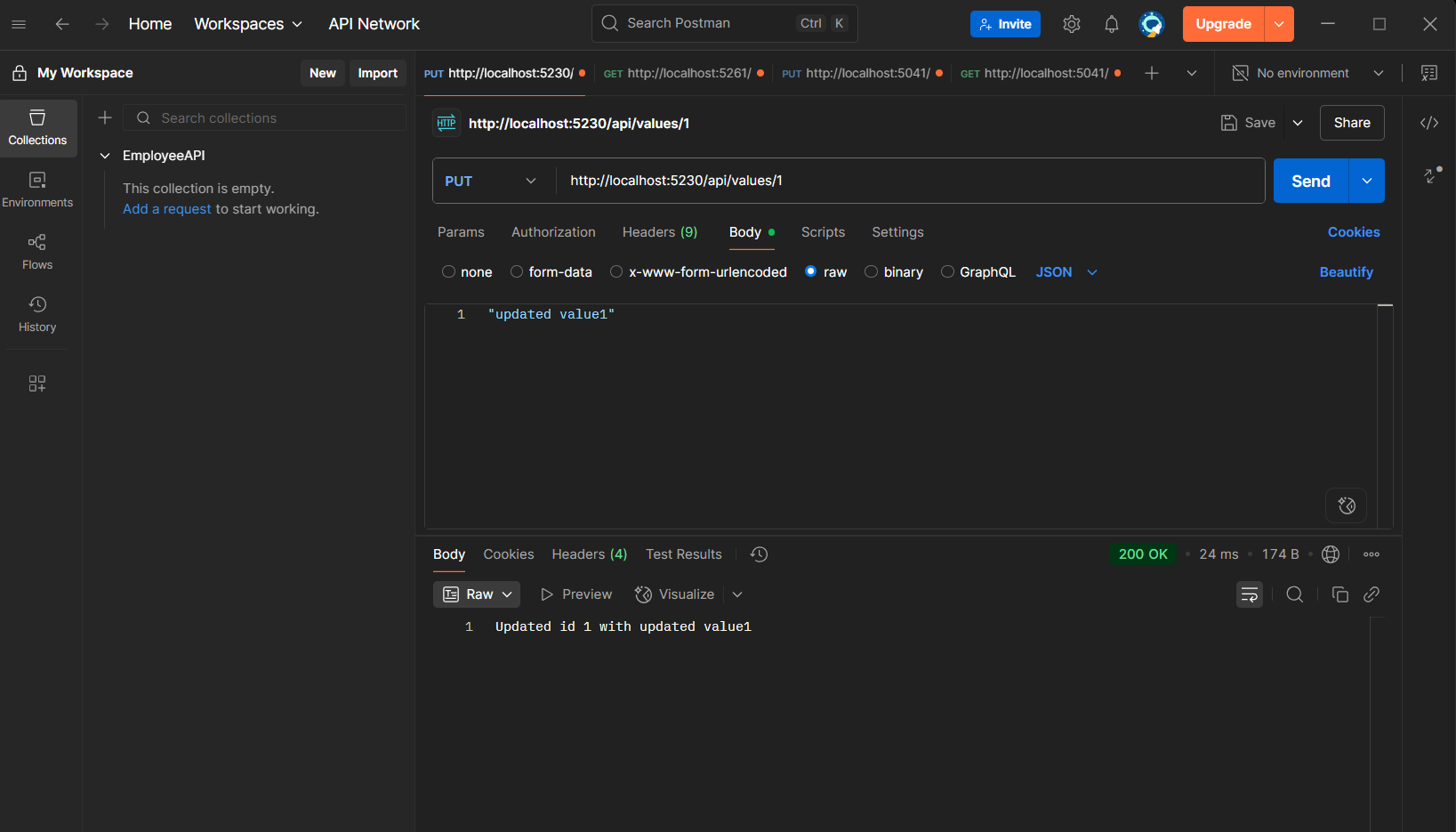
**GET:**

****

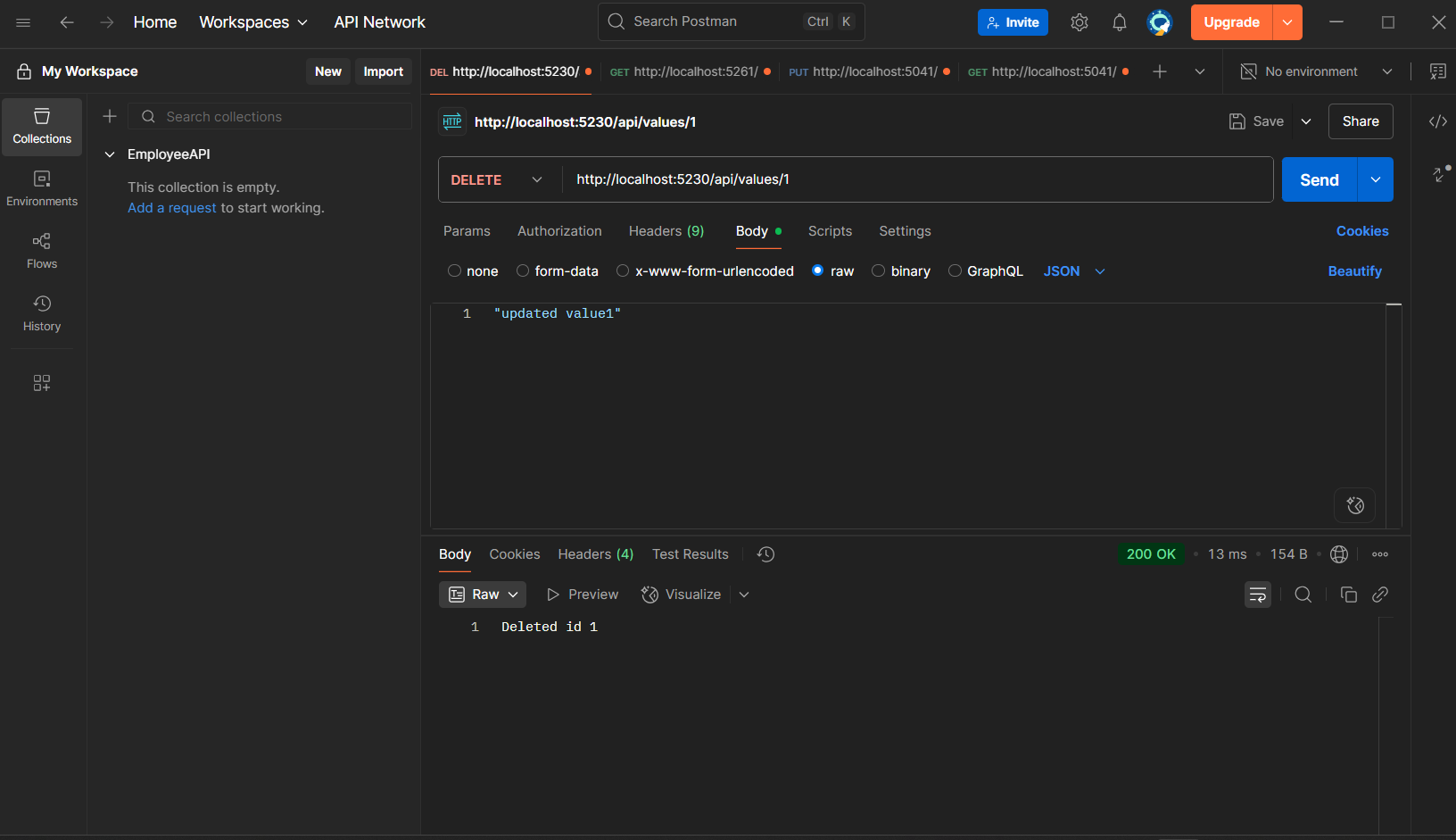
**POST:**

****

**PUT:**



**DELETE:**

****

**HANDSON 2:**

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

**EmployeeController.cs:**

using Microsoft.AspNetCore.Mvc;

namespace SwaggerDemo.Controllers

{

    //[Route("api/[controller]")]

    [Route("api/emp")]

    [ApiController]

    public class EmployeeController : ControllerBase

    {

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

        {

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

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

        };

        [HttpGet]

        [ProducesResponseType(StatusCodes.Status200OK)]

        public ActionResult<IEnumerable<Employee>> Get()

        {

            return Ok(employees);

        }

        [HttpGet("{id}", Name = "GetEmployeeById")]

        [ProducesResponseType(StatusCodes.Status200OK)]

        [ProducesResponseType(StatusCodes.Status404NotFound)]

        public ActionResult<Employee> GetById(int id)

        {

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

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

            return Ok(emp);

        }

        [HttpPost]

        [ProducesResponseType(StatusCodes.Status201Created)]

        public ActionResult<Employee> Post(Employee emp)

        {

            emp.Id = employees.Max(e => e.Id) + 1;

            employees.Add(emp);

            return CreatedAtRoute("GetEmployeeById", new { id = emp.Id }, emp);

        }

    }

    public class Employee

    {

        public int Id { get; set; }

        public string Name { get; set; }

    }

}

**Program.cs:**

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

    c.SwaggerDoc("v1", new OpenApiInfo

    {

        Title = "Swagger Demo",

        Version = "v1",

        Description = "TBD",

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

        Contact = new OpenApiContact

        {

            Name = "John Doe",

            Email = "john@xyzmail.com",

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

        },

        License = new OpenApiLicense

        {

            Name = "License Terms",

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

        }

    });

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI(c =>

    {

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

    });

}

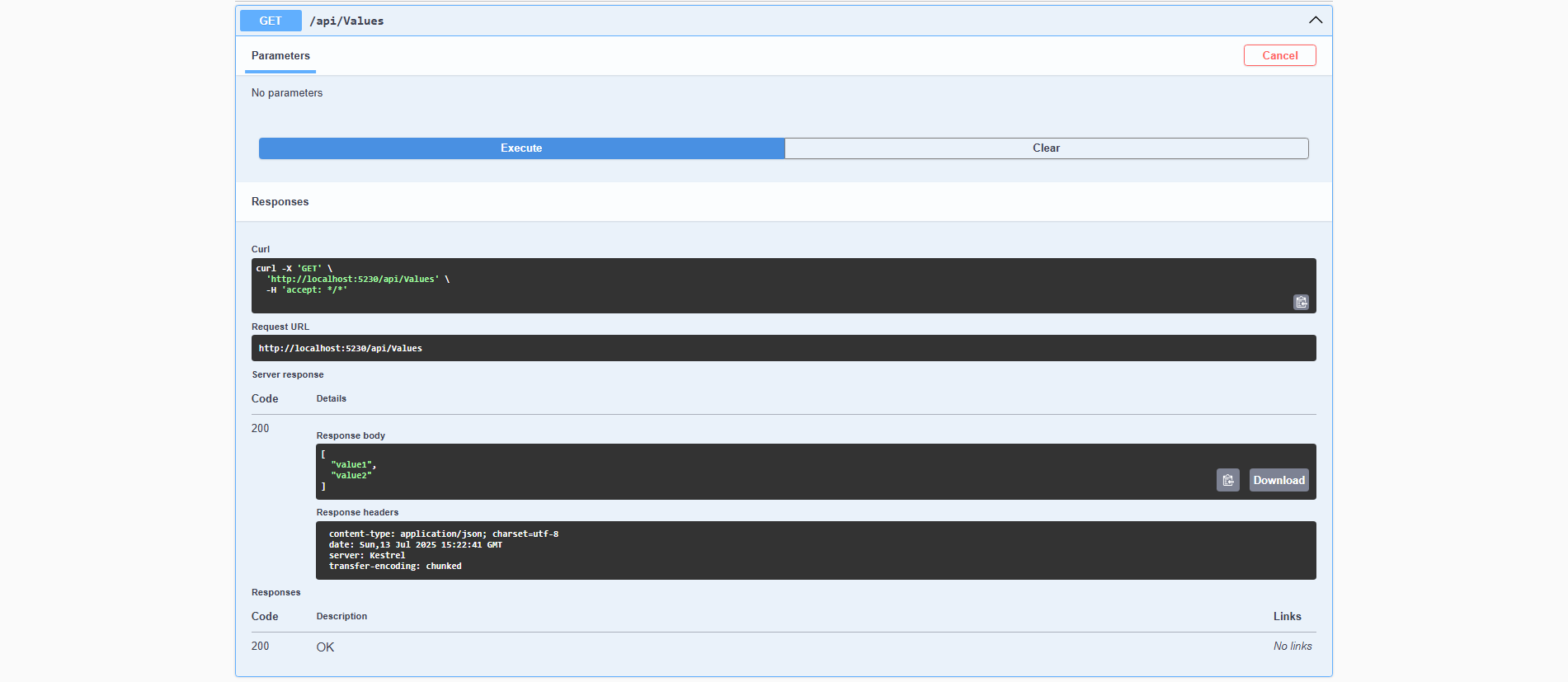
app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

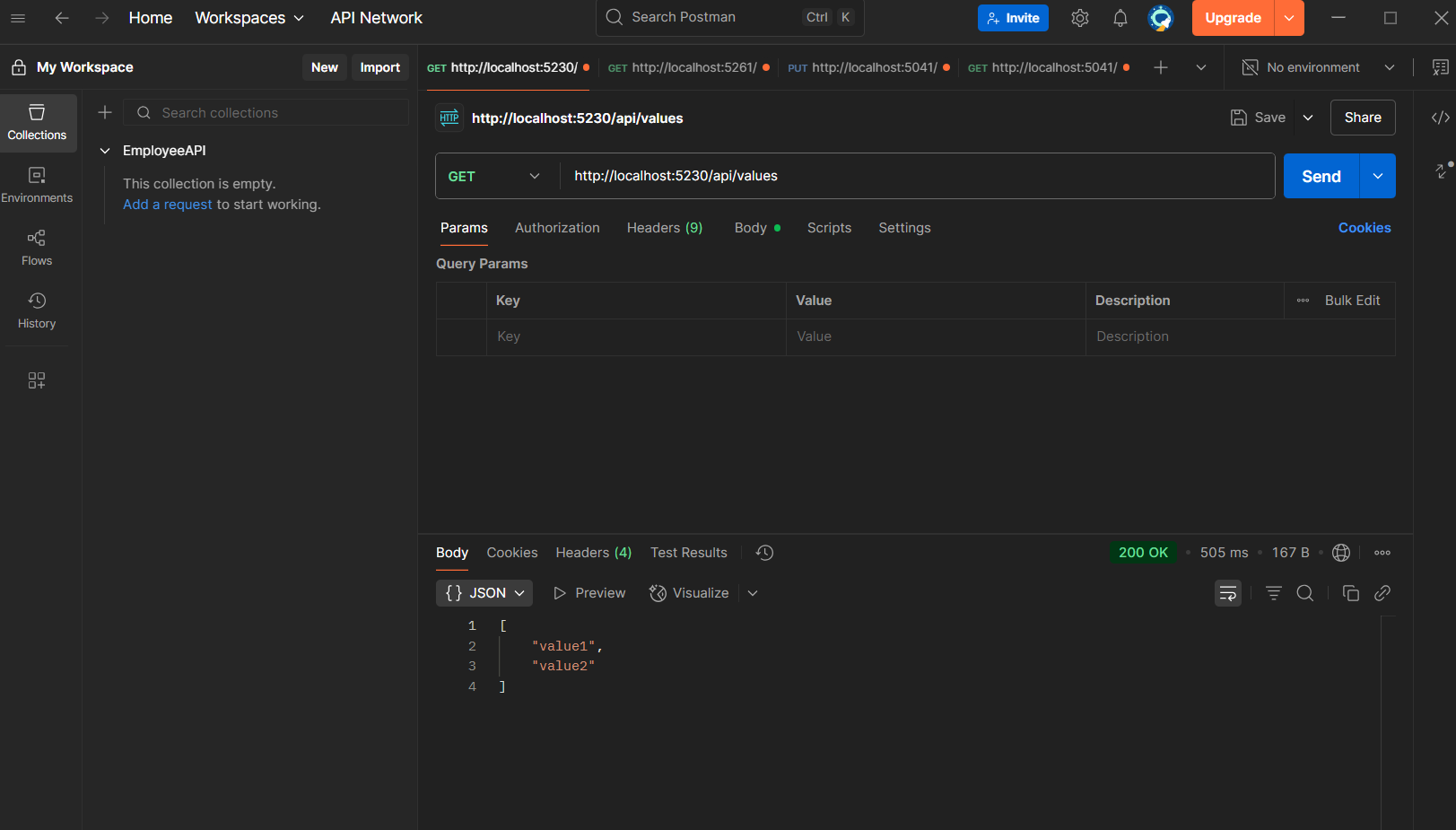
app.Run();

**OUTPUTS:**



**POSTMAN:**

**After modification:**

****

**HANDSON 3:**

1. **Web Api using custom model class:**

**EmployeeController.cs:**

using Microsoft.AspNetCore.Mvc;

using EmployeeApiDemo.Models;

using EmployeeApiDemo.Filters;

using Microsoft.AspNetCore.Authorization;

namespace EmployeeApiDemo.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    [ServiceFilter(typeof(CustomAuthFilter))]

    [ServiceFilter(typeof(CustomExceptionFilter))]

    public class EmployeeController : ControllerBase

    {

        private static List<Employee> employees;

        public EmployeeController()

        {

            if (employees == null)

                employees = GetStandardEmployeeList();

        }

        private List<Employee> GetStandardEmployeeList()

        {

            return new List<Employee>

            {

                new Employee

                {

                    Id = 1,

                    Name = "Alice",

                    Salary = 50000,

                    Permanent = true,

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

                    Skills = new List<Skill>

                    {

                        new Skill { Id = 1, Name = "Communication" },

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

                    },

                    DateOfBirth = new DateTime(1990, 5, 12)

                }

            };

        }

        [HttpGet]

        [AllowAnonymous] // ✅ This allows Swagger GET without Authorization header

        [ProducesResponseType(StatusCodes.Status200OK)]

        [ProducesResponseType(StatusCodes.Status500InternalServerError)]

        public ActionResult<IEnumerable<Employee>> Get()

        {

            // Uncomment this to test the Exception Filter:

            //throw new Exception("Test exception from GET.");

            return Ok(employees);

        }

        [HttpPost]

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

        {

            emp.Id = employees.Max(e => e.Id) + 1;

            employees.Add(emp);

            return CreatedAtAction(nameof(Get), emp);

        }

        // [HttpPut("{id}")]

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

        // {

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

        //     if (existing == null)

        //     {

        //         return NotFound();

        //     }

        //     existing.Name = emp.Name;

        //     existing.Salary = emp.Salary;

        //     existing.Permanent = emp.Permanent;

        //     existing.Department = emp.Department;

        //     existing.Skills = emp.Skills;

        //     existing.DateOfBirth = emp.DateOfBirth;

        //     return Ok(existing);

        // }

        [HttpPut("{id}")]

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

{

    if (id <= 0)

    {

        return BadRequest("Invalid employee id");

    }

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

    if (existing == null)

    {

        return BadRequest("Invalid employee id");

    }

    existing.Name = emp.Name;

    existing.Salary = emp.Salary;

    existing.Permanent = emp.Permanent;

    existing.Department = emp.Department;

    existing.Skills = emp.Skills;

    existing.DateOfBirth = emp.DateOfBirth;

    return Ok(existing);

}

    }

}

**CustomAuthFilter.cs:**

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

namespace EmployeeApiDemo.Filters

{

    public class CustomAuthFilter : ActionFilterAttribute

    {

        public override void OnActionExecuting(ActionExecutingContext context)

        {

            // ✅ Check if action has [AllowAnonymous]

            var isAllowAnonymous = context.ActionDescriptor.EndpointMetadata

                .Any(meta => meta is AllowAnonymousAttribute);

            if (isAllowAnonymous)

            {

                // Skip the filter

                base.OnActionExecuting(context);

                return;

            }

            var hasAuth = context.HttpContext.Request.Headers.ContainsKey("Authorization");

            if (!hasAuth)

            {

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

                return;

            }

            var token = context.HttpContext.Request.Headers["Authorization"].ToString();

            if (!token.Contains("Bearer"))

            {

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

                return;

            }

            base.OnActionExecuting(context);

        }

    }

}

**CustomExceptionFilter.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace EmployeeApiDemo.Filters

{

    public class CustomExceptionFilter : IExceptionFilter

    {

        public void OnException(ExceptionContext context)

        {

            var exception = context.Exception;

            var logPath = Path.Combine(Directory.GetCurrentDirectory(), "errorlog.txt");

            File.AppendAllText(logPath, $"[{DateTime.Now}] {exception.Message}\n");

            context.Result = new ObjectResult("Something went wrong.")

            {

                StatusCode = 500

            };

        }

    }

}

**Employee.cs:**

namespace EmployeeApiDemo.Models

{

    public class Employee

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public int Salary { get; set; }

        public bool Permanent { get; set; }

        public Department Department { get; set; }

        public List<Skill> Skills { get; set; }

        public DateTime DateOfBirth { get; set; }

    }

    public class Department

    {

        public int Id { get; set; }

        public string Name { get; set; }

    }

    public class Skill

    {

        public int Id { get; set; }

        public string Name { get; set; }

    }

}

**Program.cs:**

using Microsoft.OpenApi.Models;

using EmployeeApiDemo.Filters;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddScoped<CustomAuthFilter>();

builder.Services.AddScoped<CustomExceptionFilter>();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

    c.SwaggerDoc("v1", new OpenApiInfo

    {

        Title = "Employee API Demo",

        Version = "v1",

        Description = "Demo with custom filters",

        Contact = new OpenApiContact

        {

            Name = "Your Name",

            Email = "your@email.com",

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

        }

    });

    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

    {

        Description = "JWT Authorization header using the Bearer scheme. Example: \"Bearer {token}\"",

        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 string[] {}

        }

    });

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI();

}

app.UseHttpsRedirection();

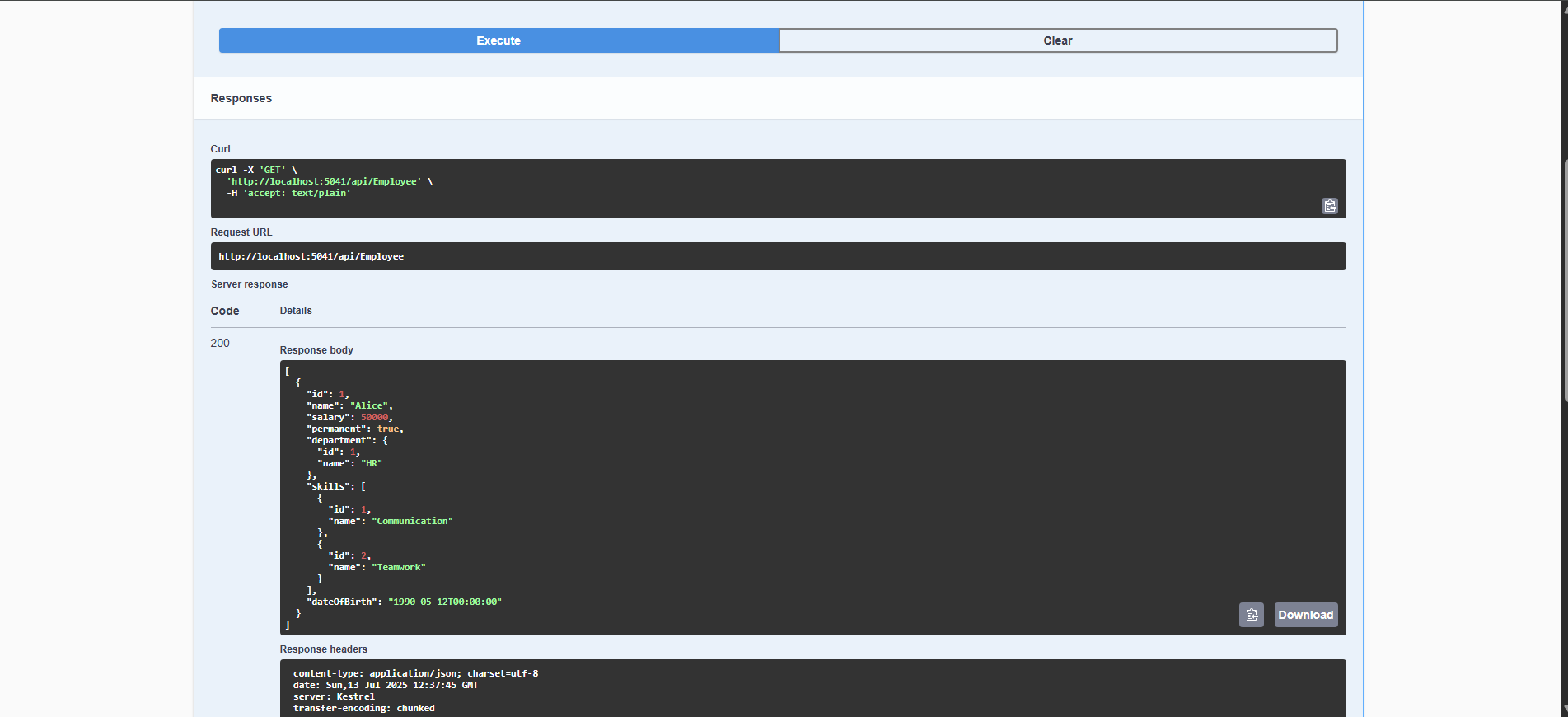
app.UseAuthorization();

app.MapControllers();

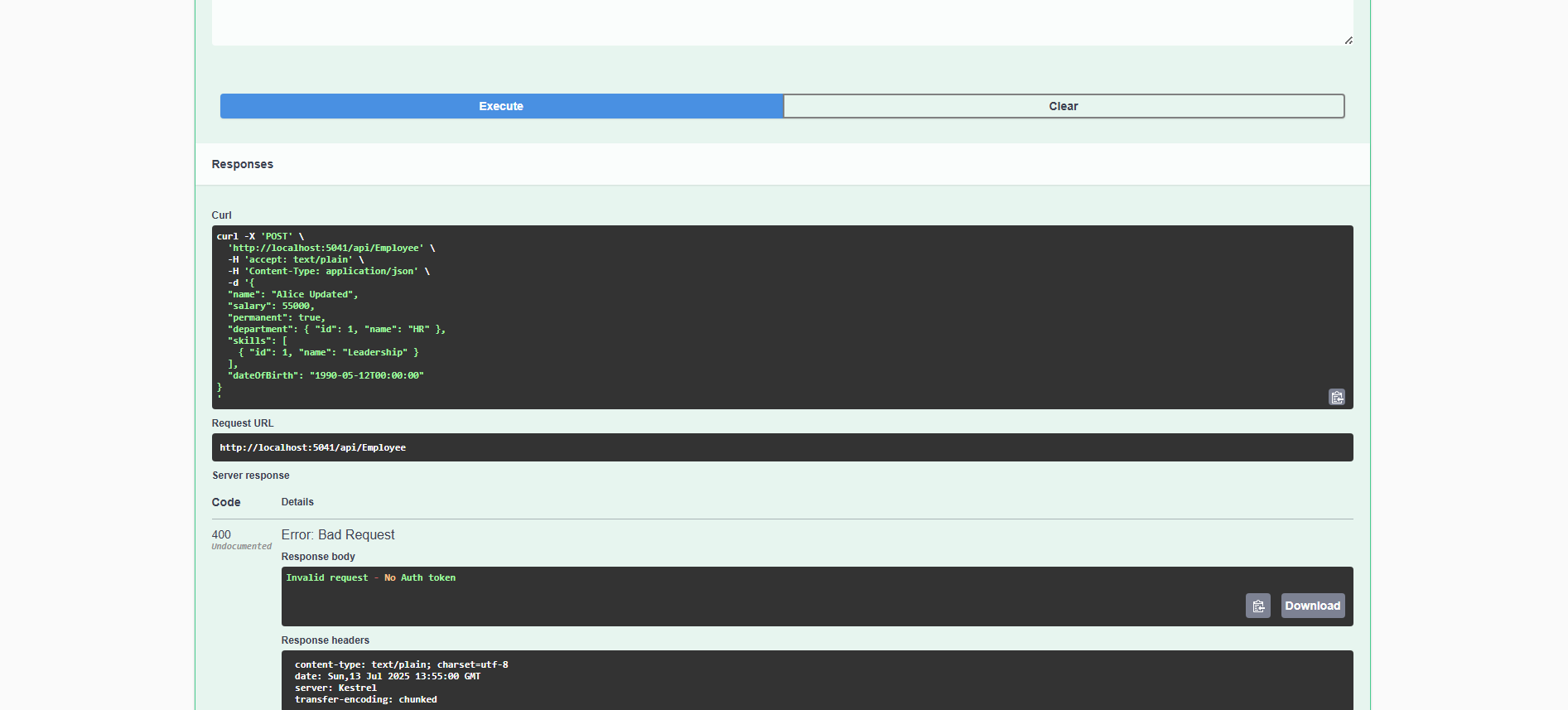
app.Run();

**OUTPUT:**

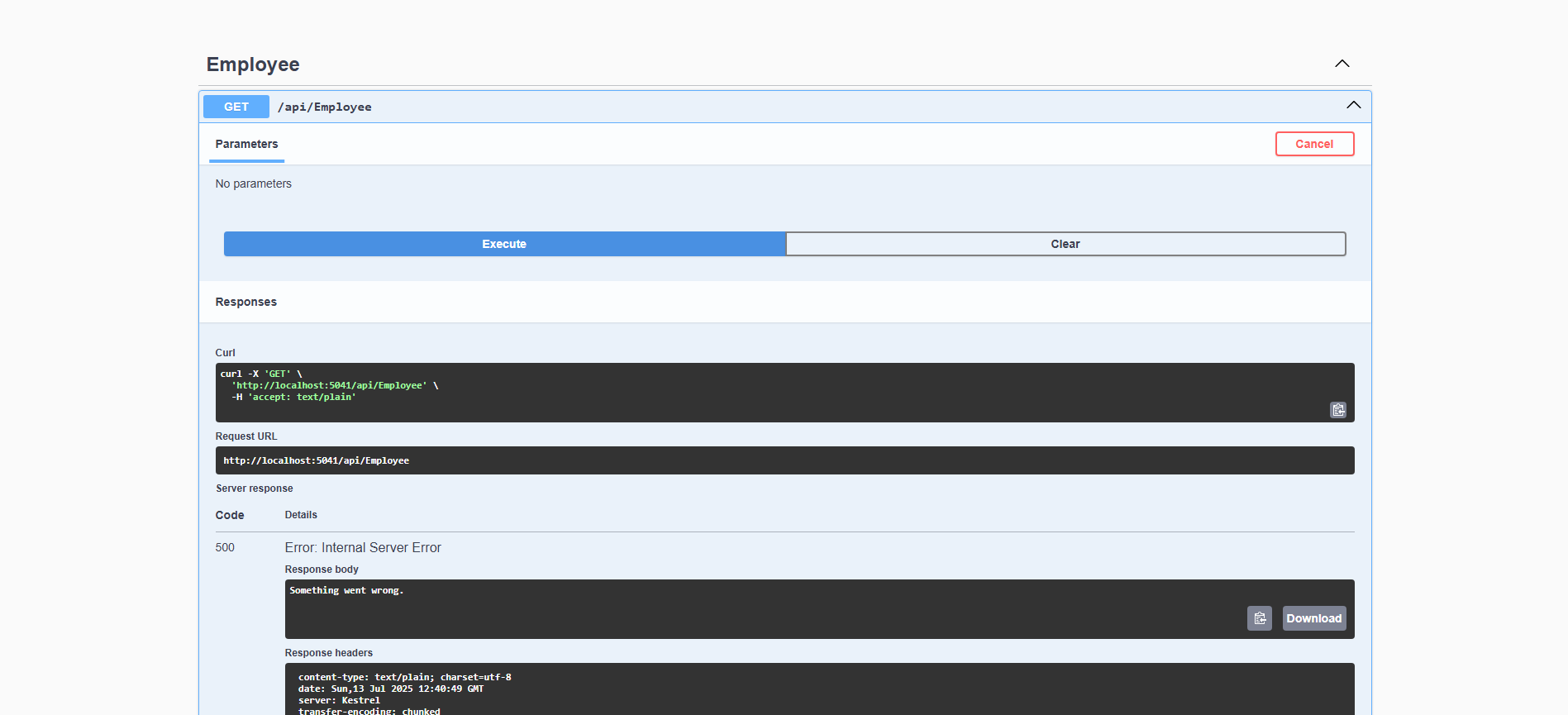
**200 OK:**

****

**400:**



**500:**

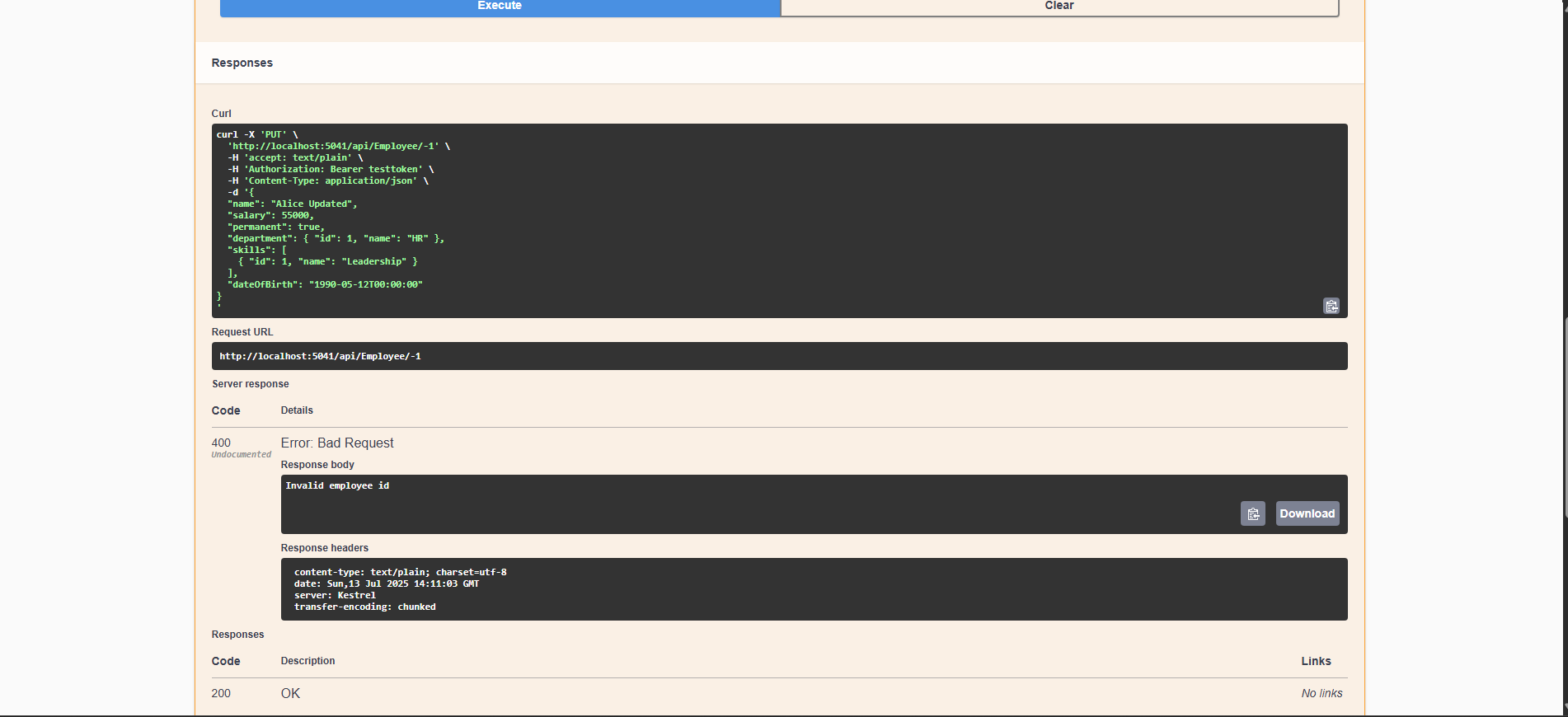
****

**HANDSON 4:**

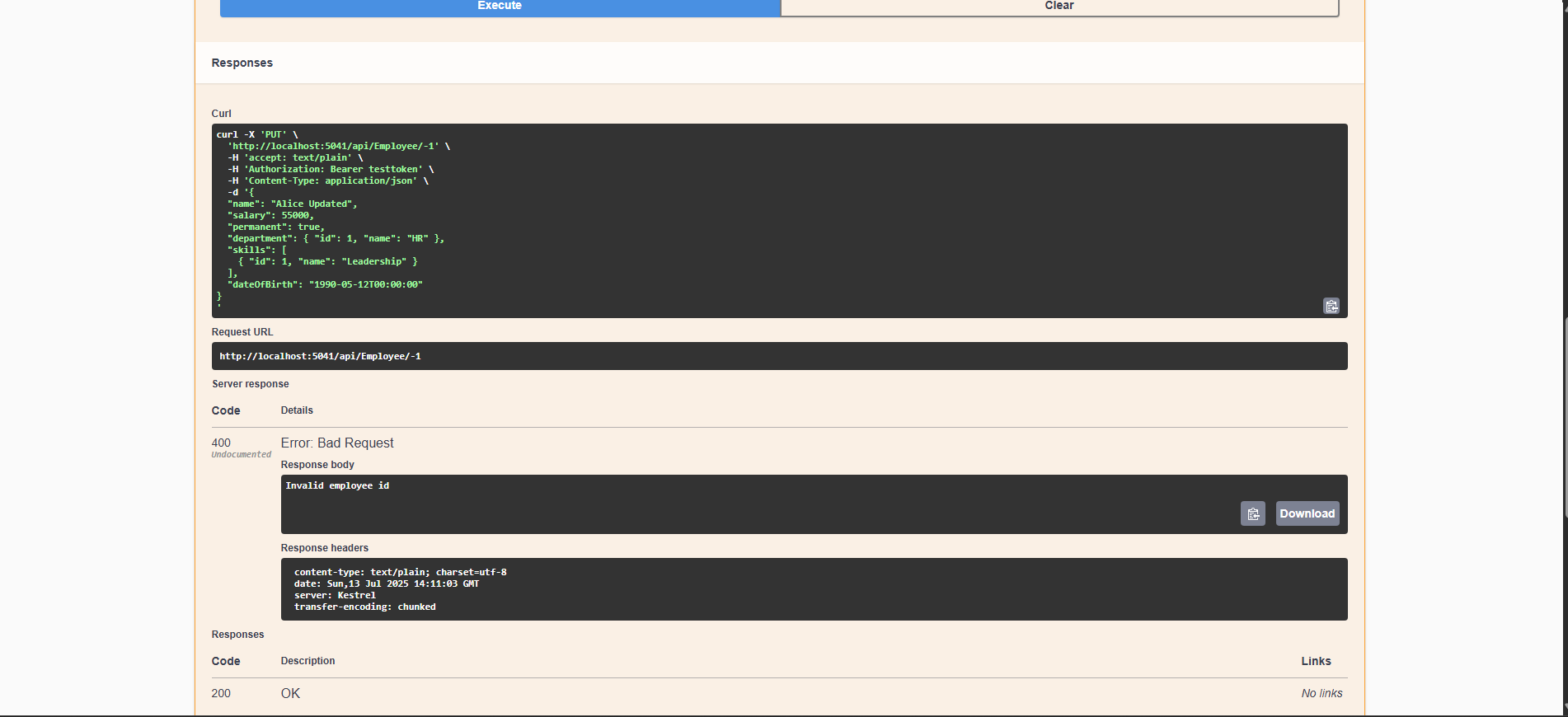
**CODE IS SAME AS ABOVE**

**OUTPUT:**

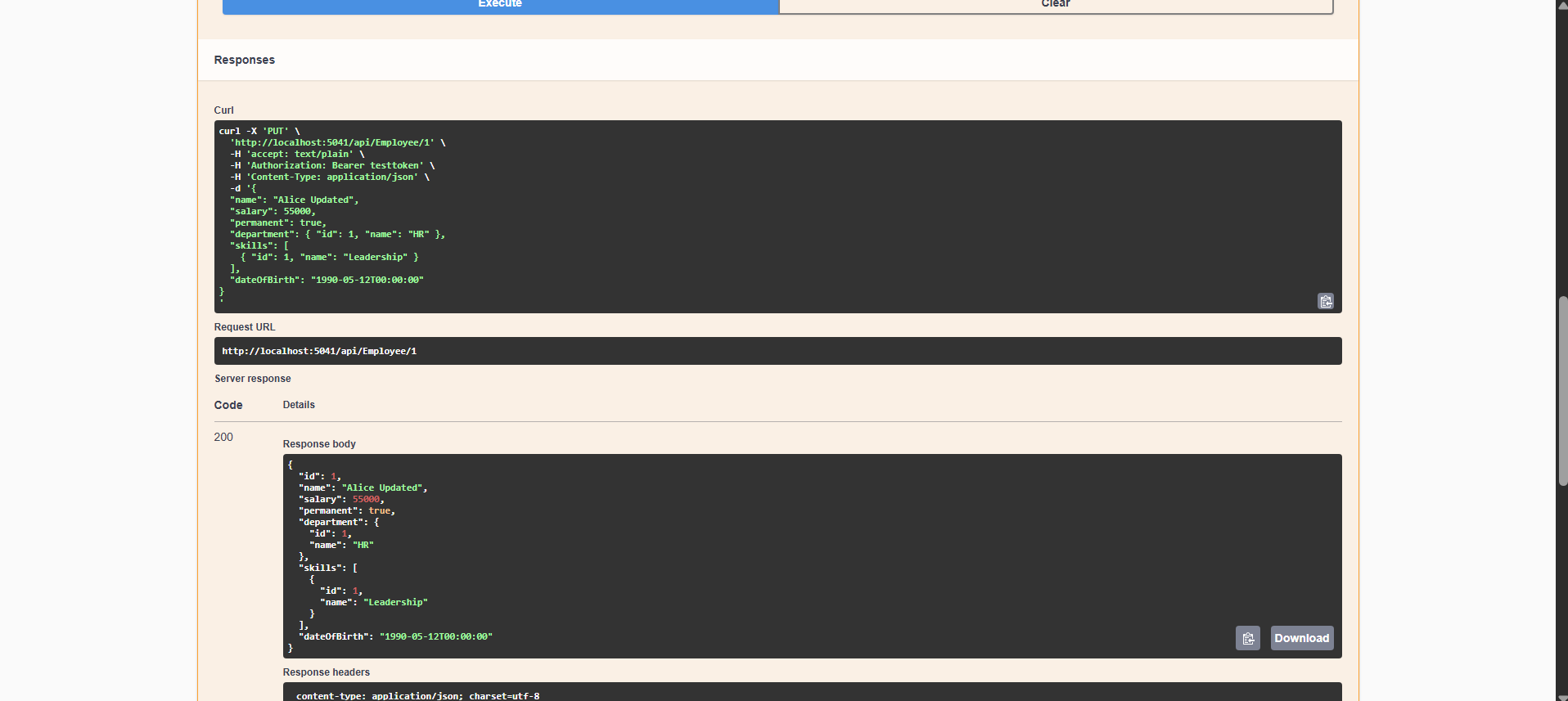
**IF LESS OR EQUAL TO 0:**

****

**IF GREATER THAN 0 BUT NOT PRESENT IN DATA:**

****

**IF ID IS PRESENT:**

****

**HANDSON 5:**

CORS and Security in Web API

# 1. What is CORS?

• CORS stands for Cross-Origin Resource Sharing.  
• It is a browser security feature that blocks requests if the frontend and backend run on different domains, ports, or protocols.  
• Example: Frontend runs on http://localhost:3000, API runs on http://localhost:5000 — this is cross-origin.  
• CORS allows you to specify which origins are allowed to access your API.

# 2. Enable CORS in ASP.NET Core

1) Install the CORS NuGet package if needed:  
 dotnet add package Microsoft.AspNetCore.Cors  
  
2) Configure CORS in Program.cs or Startup.cs:  
 builder.Services.AddCors(options =>  
 {  
 options.AddPolicy("AllowLocalhost",  
 policy => policy.WithOrigins("http://localhost:3000")  
 .AllowAnyMethod()  
 .AllowAnyHeader());  
 });  
  
3) Enable CORS middleware:  
 app.UseCors("AllowLocalhost");

# 3. Security in WebAPI

Modern APIs protect endpoints using Bearer Tokens, mainly JWT (JSON Web Token).  
  
How JWT works:  
• User logs in → server validates credentials.  
• Server generates a JWT → sends it to the client.  
• Client stores token → sends it in Authorization header for every API request.  
• Server checks token → verifies identity & roles.

# 4. Add JWT Authentication

1) Install the JwtBearer package:  
 dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer  
  
2) Configure JWT in Program.cs or Startup.cs:  
 builder.Services.AddAuthentication("Bearer")  
 .AddJwtBearer("Bearer", options =>  
 {  
 options.TokenValidationParameters = new TokenValidationParameters  
 {  
 ValidateIssuer = true,  
 ValidateAudience = true,  
 ValidateLifetime = true,  
 ValidateIssuerSigningKey = true,  
 ValidIssuer = "yourIssuer",  
 ValidAudience = "yourAudience",  
 IssuerSigningKey = new SymmetricSecurityKey(  
 Encoding.UTF8.GetBytes("your\_secret\_key\_here"))  
 };  
 });  
  
Add middleware:  
 app.UseAuthentication();  
 app.UseAuthorization();

# 5. Authorize Attribute & Roles

[Authorize(Roles = "Admin")]  
[HttpGet]  
public IActionResult GetSecretData()  
{  
 return Ok("Protected data for Admins.");  
}

# 6. AllowAnonymous for Login

[AllowAnonymous]  
[HttpPost("login")]  
public IActionResult Login([FromBody] LoginModel model)  
{  
 // Validate user and generate JWT  
}

# 7. Example Claims

When creating a JWT, add claims:  
var claims = new[]  
{  
 new Claim(ClaimTypes.Name, user.Username),  
 new Claim(ClaimTypes.Role, "Admin")  
};

# 8. Claims Theory

• A claim is a piece of information about the user, like name, role, or email.  
• Identity is built from one or more claims.  
• Claims are stored inside the JWT token.  
• Server checks claims to authorize user actions.

**AuthController.cs:**

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 EmployeeApiDemo.Controllers

{

    [ApiController] // ✅ This makes it appear in Swagger

    [Route("api/[controller]")] // ✅ This means URL = /api/Auth

    [AllowAnonymous] // ✅ So you can access without token

    public class AuthController : ControllerBase

    {

        [HttpGet]

        public ActionResult<string> GetToken()

        {

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

            return Ok(token);

        }

        private string GenerateJSONWebToken(int userId, string userRole)

        {

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

            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);

        }

    }

}

**EmployeeController.cs:**

using Microsoft.AspNetCore.Mvc;

using EmployeeApiDemo.Models;

using Microsoft.AspNetCore.Authorization;

namespace EmployeeApiDemo.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    [Authorize(Roles = "Admin,POC")] // ✅ Accept Admin or POC role

    public class EmployeeController : ControllerBase

    {

        private static List<Employee> employees;

        public EmployeeController()

        {

            if (employees == null)

                employees = GetStandardEmployeeList();

        }

        private List<Employee> GetStandardEmployeeList()

        {

            return new List<Employee>

            {

                new Employee

                {

                    Id = 1,

                    Name = "Alice",

                    Salary = 50000,

                    Permanent = true,

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

                    Skills = new List<Skill>

                    {

                        new Skill { Id = 1, Name = "Communication" },

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

                    },

                    DateOfBirth = new DateTime(1990, 5, 12)

                }

            };

        }

        [HttpGet]

        public ActionResult<IEnumerable<Employee>> Get()

        {

            return Ok(employees);

        }

    }

}

**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 = "Employee API Demo", Version = "v1" });

    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

    {

        Description = "JWT Authorization header using the Bearer scheme. Example: \"Bearer {token}\"",

        Name = "Authorization",

        In = ParameterLocation.Header,

        Type = SecuritySchemeType.ApiKey,

        Scheme = "Bearer"

    });

    c.AddSecurityRequirement(new OpenApiSecurityRequirement()

    {

        {

            new OpenApiSecurityScheme

            {

                Reference = new OpenApiReference

                {

                    Type = ReferenceType.SecurityScheme,

                    Id = "Bearer"

                }

            },

            Array.Empty<string>()

        }

    });

});

builder.Services.AddCors(options =>

{

    options.AddPolicy("AllowAll",

        builder => builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

});

string securityKey = "mysuperdupersecretmysuperdupersecret";

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

builder.Services.AddAuthentication(options =>

{

    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, options =>

{

    options.TokenValidationParameters = new TokenValidationParameters

    {

        ValidateIssuer = true,

        ValidateAudience = true,

        ValidateLifetime = true,

        ValidateIssuerSigningKey = true,

        ValidIssuer = "mySystem",

        ValidAudience = "myUsers",

        IssuerSigningKey = symmetricSecurityKey

    };

});

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

app.UseHttpsRedirection();

// Use CORS if you defined it

app.UseCors("AllowAll");

// Enable Authentication + Authorization

app.UseAuthentication();

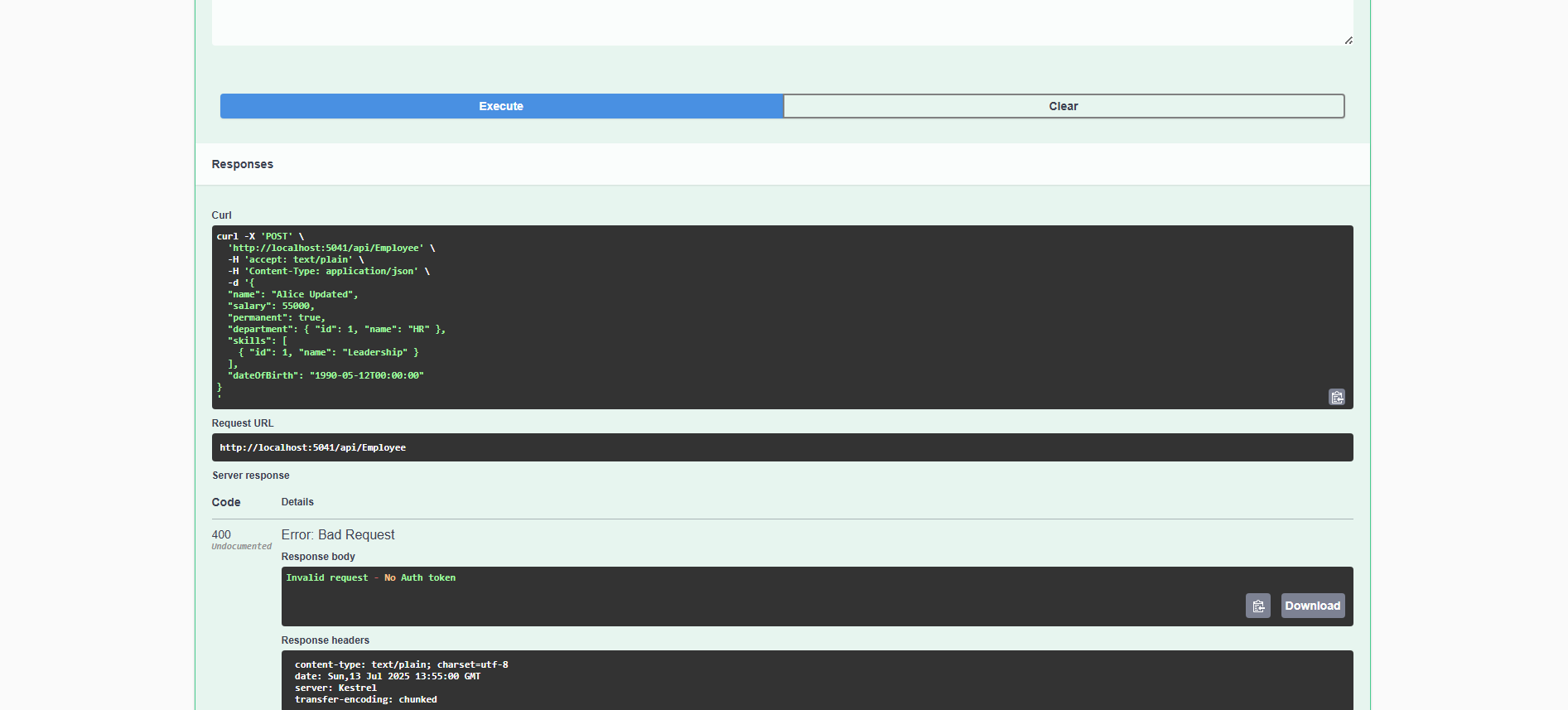
app.UseAuthorization();

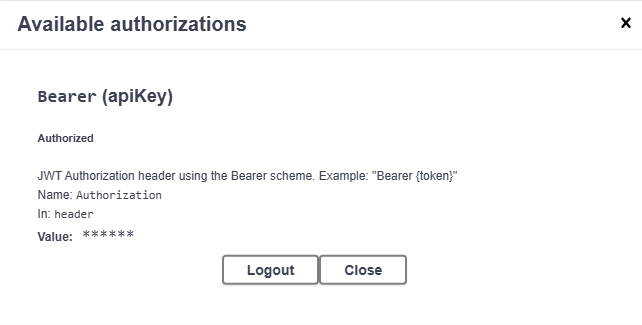
app.MapControllers();

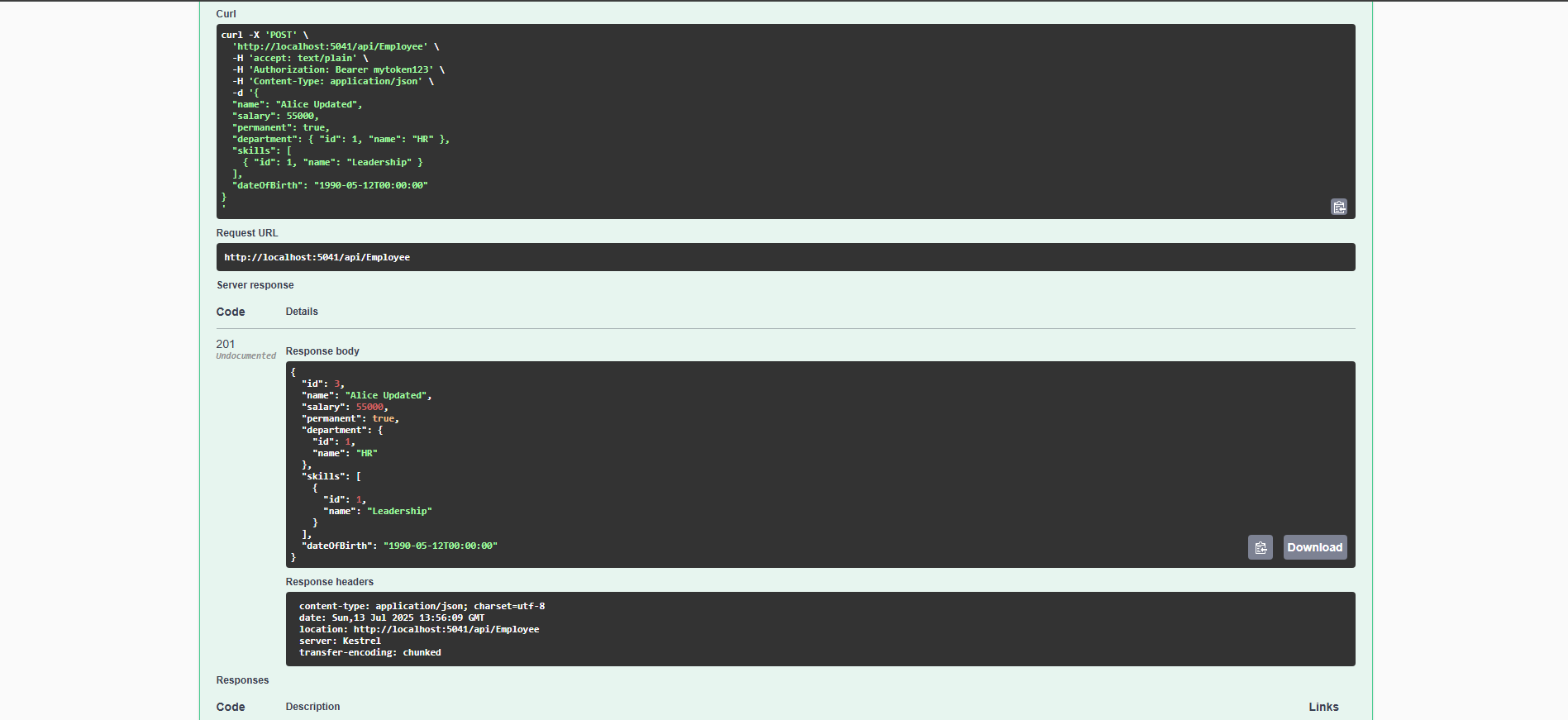
app.Run();

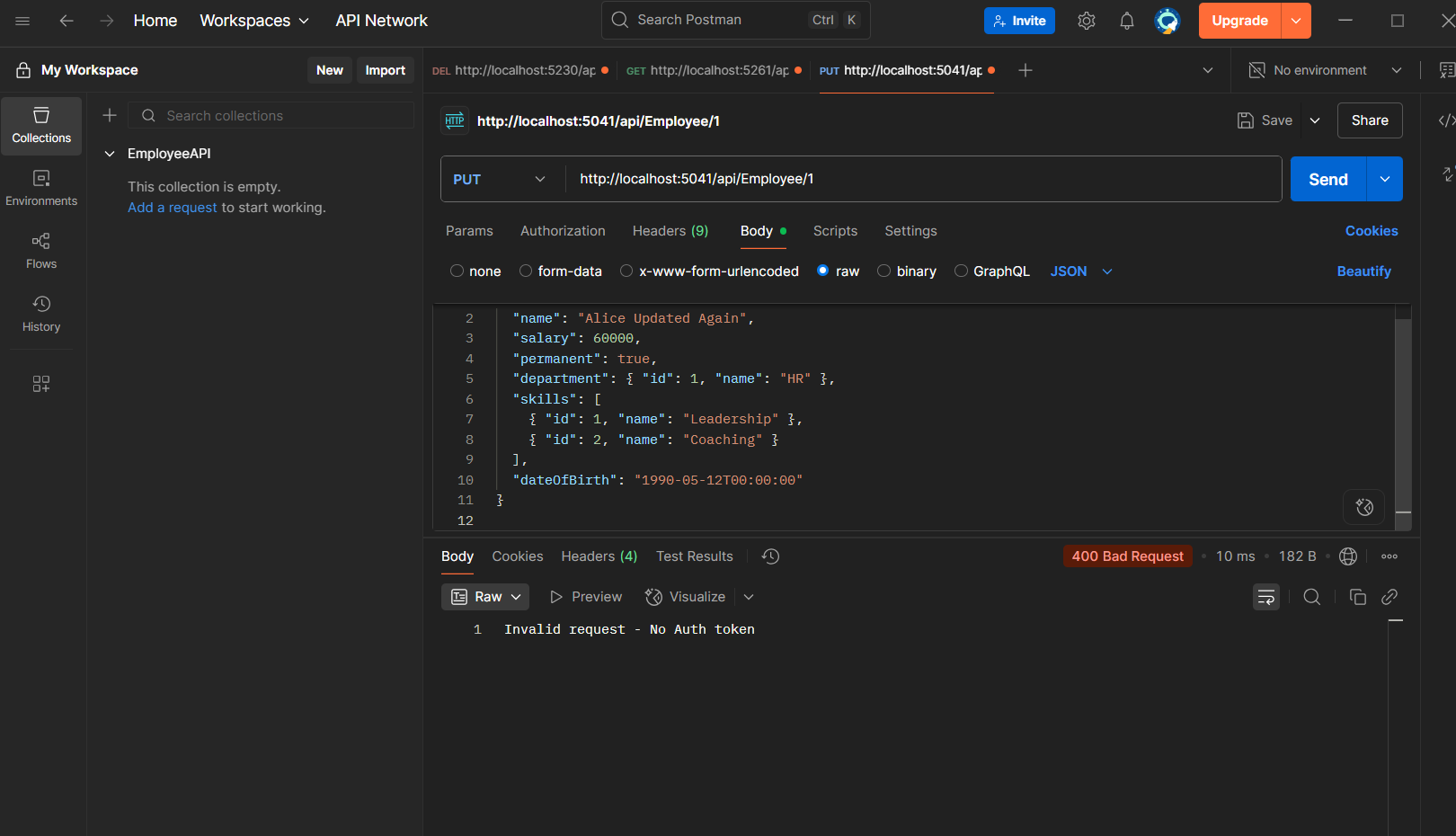
**Employee.cs and CustomerExceptionFilter.cs is same as previous**

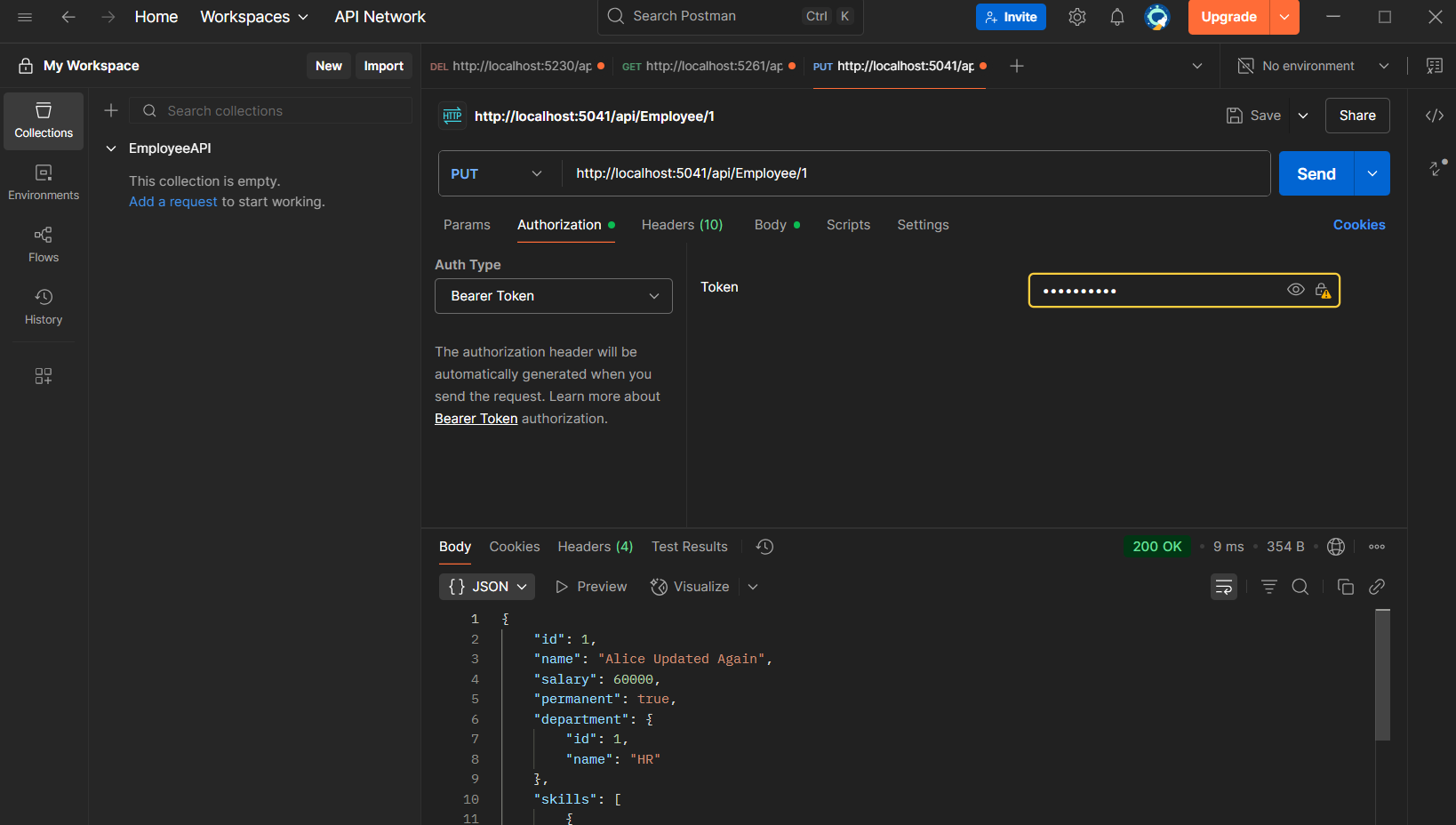
**OUTPUT:**

****

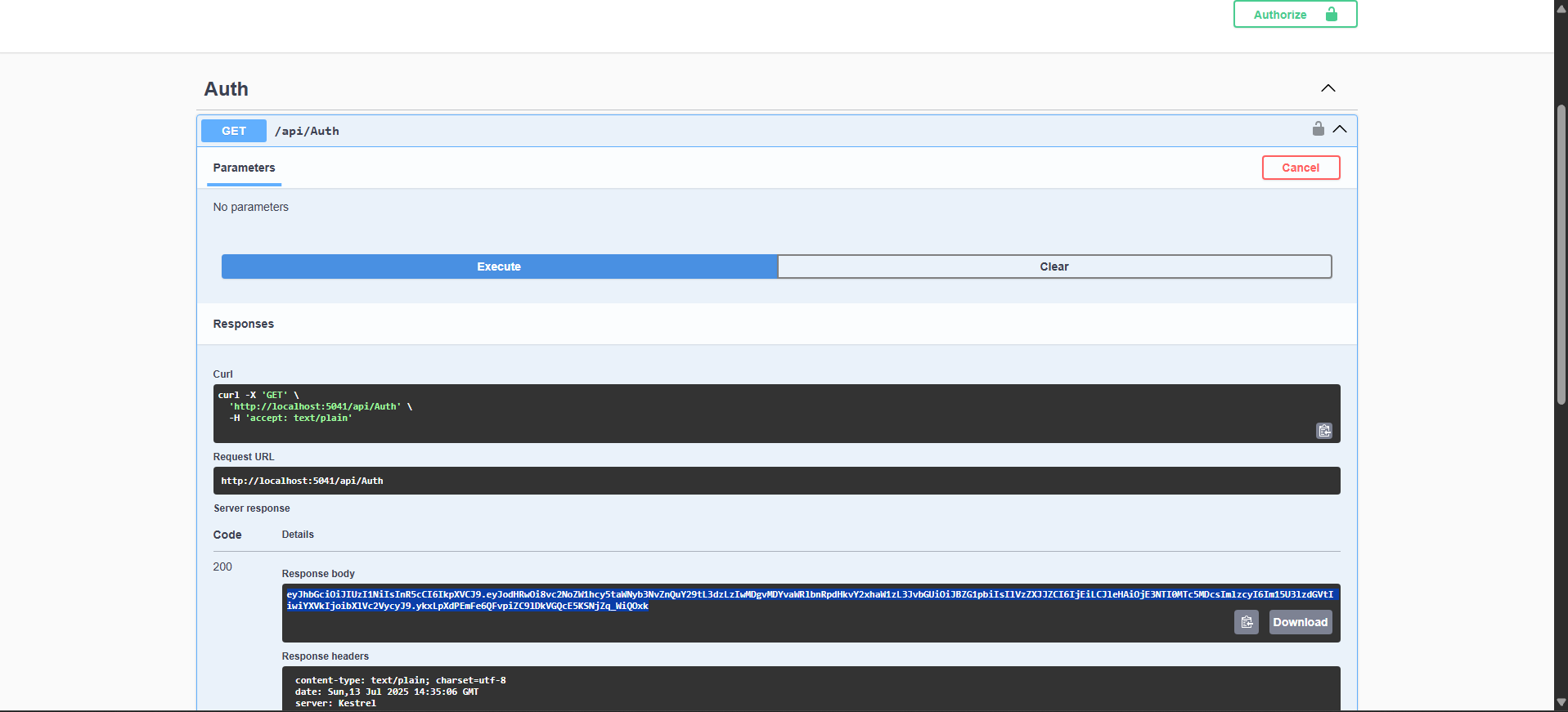
****

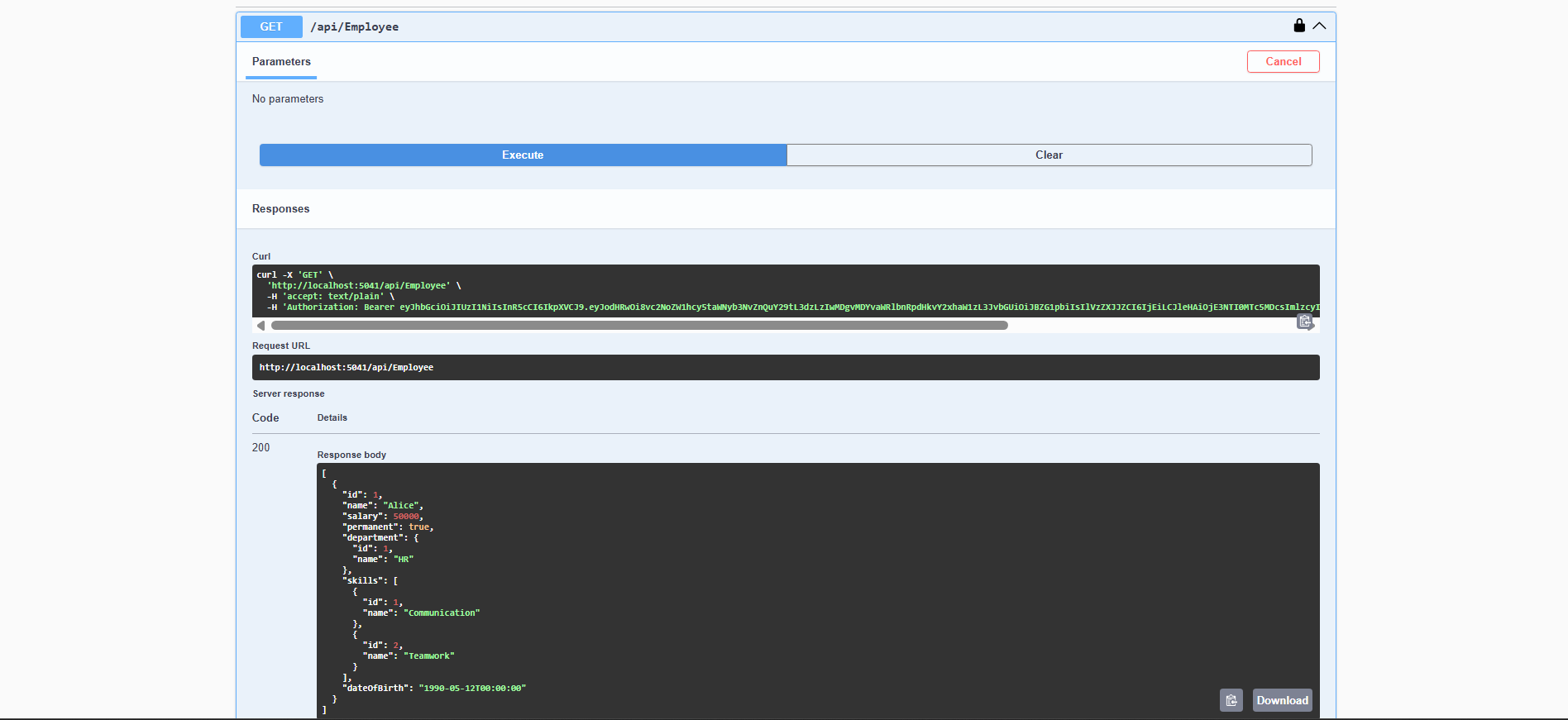
****

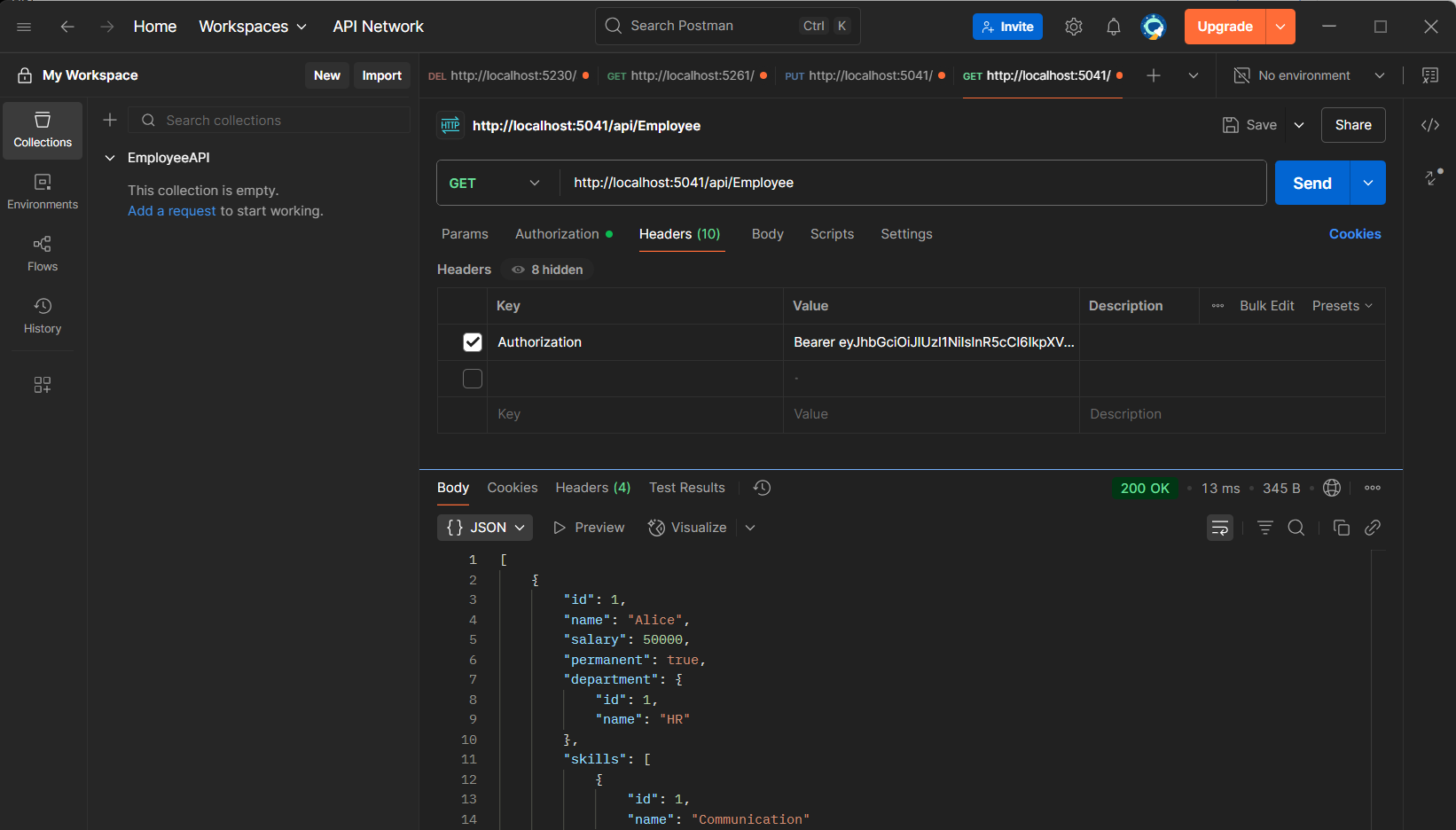
****

****

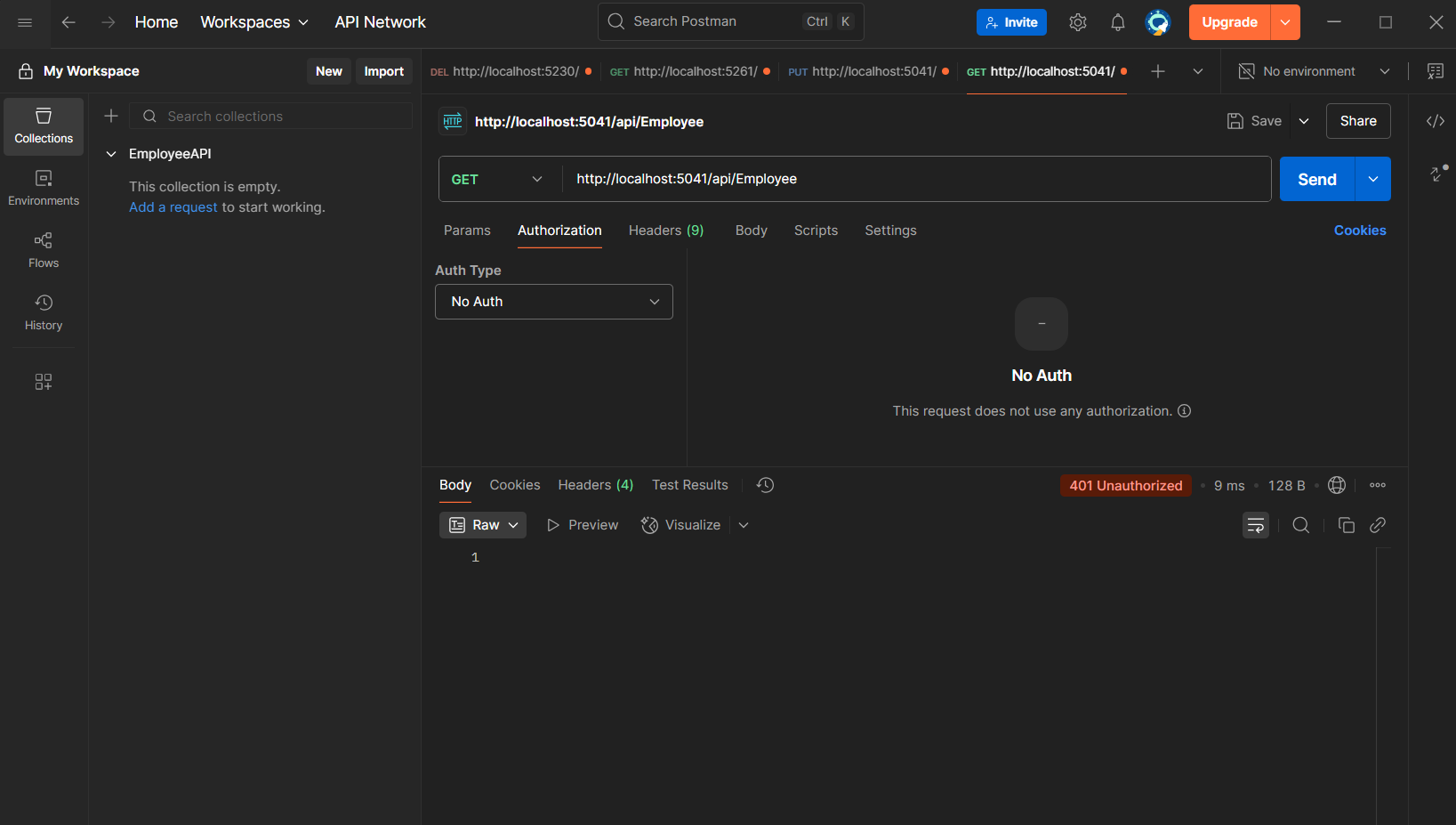
**JWT:**

****

****

****

**401:**

****