**WEBAPI**

**1.Explain the concept of RESTful web service, Web API & Microservice.Features of REST architecture - Representational State Transfer, Stateless, Messages, Concept of Microservice, Difference between WebService & WebAPI, Not restricted to send XML as response**

**SOLUTION:**

**RESTful Web Service**

REST (Representational State Transfer) is an architectural style for designing networked applications. A RESTful Web Service adheres to REST's principles, treating everything as a resource identified by a unique URI. Clients interact with these resources using standard HTTP methods (GET, POST, PUT, DELETE) to transfer representations of the resource's state (e.g., JSON, XML). Key features include being stateless (each request is independent, improving scalability) and having a uniform interface (simplifying client-server interaction).

**Web API**

A Web API (Application Programming Interface) is a broad term for any API accessible over the web via HTTP. While it's a general concept, in modern development, "Web API" is almost synonymous with a RESTful API, emphasizing lightweight communication and ease of consumption. Unlike traditional "Web Services" (often associated with SOAP and XML-heavy communication), modern Web APIs are not restricted to XML and commonly use JSON for data exchange due to its simplicity and efficiency.

**Microservice**

Microservices represent an architectural style where an application is built as a collection of small, loosely coupled, and independently deployable services. Each microservice focuses on a single business capability. They communicate primarily through lightweight mechanisms, most commonly RESTful Web APIs, and can be developed and scaled independently. This approach enhances agility, resilience, and scalability compared to monolithic architectures.

In essence:

REST is the blueprint for designing efficient web communication.

A Web API is the interface that exposes application functionality over the web, often following the REST blueprint.

Microservices are an architectural strategy that leverages these RESTful Web APIs to build large, distributed systems from smaller, autonomous components.

**CODE:**

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace MyFirstWebAPI.Controllers

{

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

[ApiController]

public class ValuesController : ControllerBase

{

static List<string> data = new List<string> { "value1", "value2" };

// GET api/values

[HttpGet]

public ActionResult<IEnumerable<string>> Get()

{

return Ok(data); // 200 OK

}

// GET api/values/1

[HttpGet("{id}")]

public ActionResult<string> Get(int id)

{

if (id >= data.Count)

return NotFound(); // 404 Not Found

return Ok(data[id]);

}

// POST api/values

[HttpPost]

public IActionResult Post([FromBody] string value)

{

data.Add(value);

return Ok(data);

}

// PUT api/values/1

[HttpPut("{id}")]

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

{

if (id >= data.Count)

return NotFound();

data[id] = value;

return Ok(data);

}

// DELETE api/values/1

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

if (id >= data.Count)

return NotFound();

data.RemoveAt(id);

return Ok(data);

}

}

}

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

2.

**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. Post this do the following steps in Startup.cs**

* **In ConfigureServices method, add the code provided below.**

**services.AddSwaggerGen(c =>**

**{**

**c.SwaggerDoc("v1", new Info**

**{**

**Title = "Swagger Demo",**

**Version = "v1",**

**Description = "TBD",**

**TermsOfService = "None",**

**Contact = new Contact() { Name = "John Doe", Email = "john@xyzmail.com", Url = "www.example.com" },**

**License = new License() { Name = "License Terms", Url = "www.example.com" }**

**});**

**});**

* **In Configure method, add the code provided below.**

**app.UseSwagger();**

**app.UseSwaggerUI(c =>**

**{**

**// specifying the Swagger JSON endpoint.**

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

**});**

**Execute the application which will load the default ‘Values’ controller(Settings as per launchSettings.json) GET action method. Change the url to** [**https://localhost:[port**](https://localhost:[port) **number]/swagger**

**Notice the Title, Version, Contact detail provided shown on the top of the page**

**Notice the Values controller HttpVerb action methods getting listed.**

**Click the ‘GET’ action verb method(Without the parameter).**

**A screenshot of a computer

AI-generated content may be incorrect.**

**It opens a panel which has ‘Try it out’ button. Click that and Click ‘Execute’ button.**

1. **Use POSTMAN tool, to point to the local Web API that was created with Employee controller. Test the GET action method using POSTMAN.**

**Verify the output if the List of employees are listed in the ‘Body’ part of the GET window on POSTMAN tool.**

**Verify the Status on the right side of the output pane on POSTMAN tool.**

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

**SOLUTION**

**CODE:**

**Program.cs**

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Microsoft.OpenApi.Models.OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

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

Contact = new Microsoft.OpenApi.Models.OpenApiContact

{

Name = "John Doe",

Email = "john@xyzmail.com",

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

},

License = new Microsoft.OpenApi.Models.OpenApiLicense

{

Name = "License Terms",

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

}

});

});

var app = builder.Build();

// Configure the HTTP request pipeline.

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

**EmployeeController.cs**

using Microsoft.AspNetCore.Mvc;

namespace SwaggerDemo.Controllers

{

[ApiController]

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

public class EmployeeController : ControllerBase

{

[HttpGet]

public IActionResult Get()

{

var employees = new[]

{

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

new { Id = 2, Name = "Bob", Department = "IT" },

new { Id = 3, Name = "Charlie", Department = "Finance" }

};

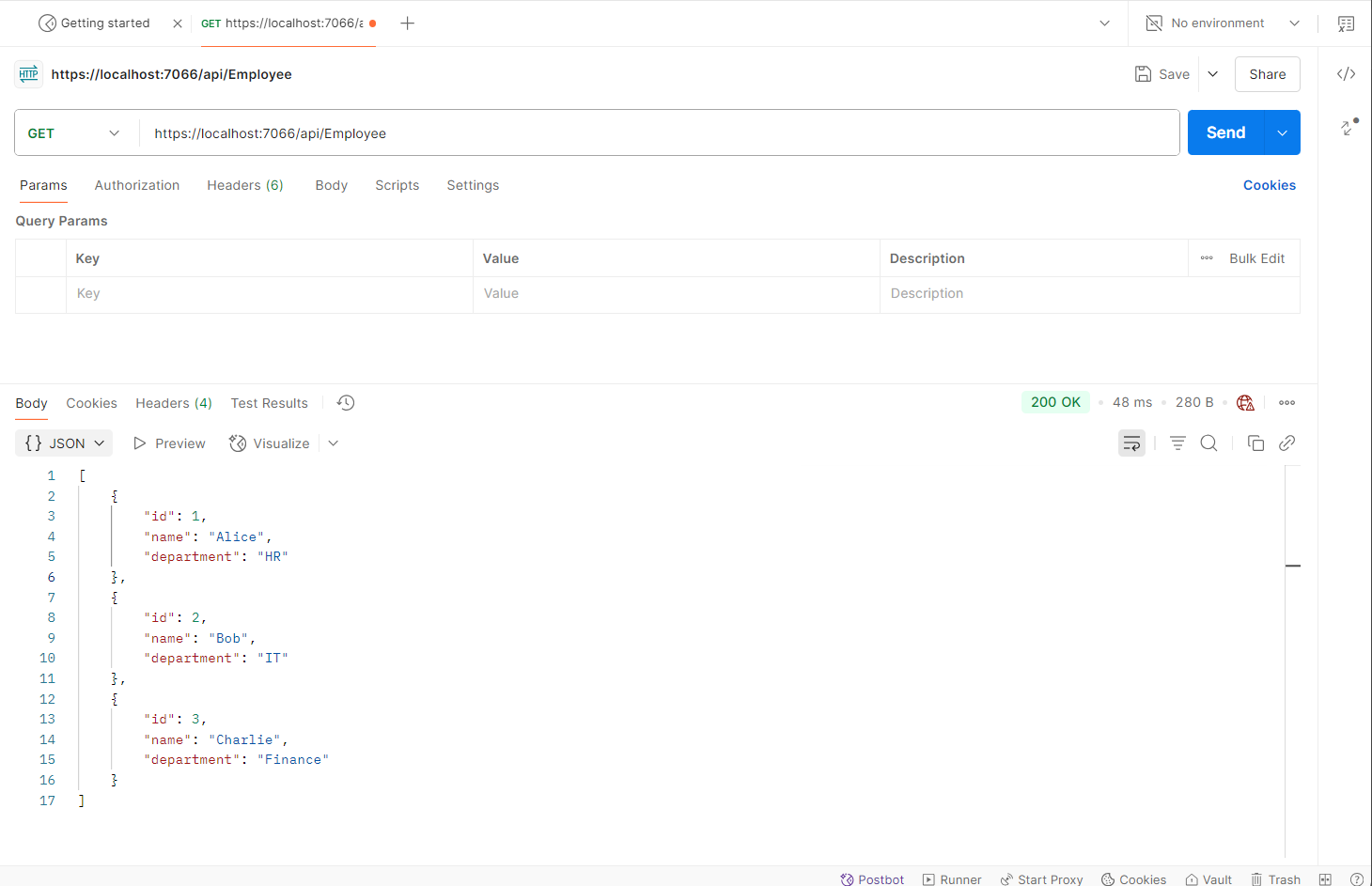
return Ok(employees);

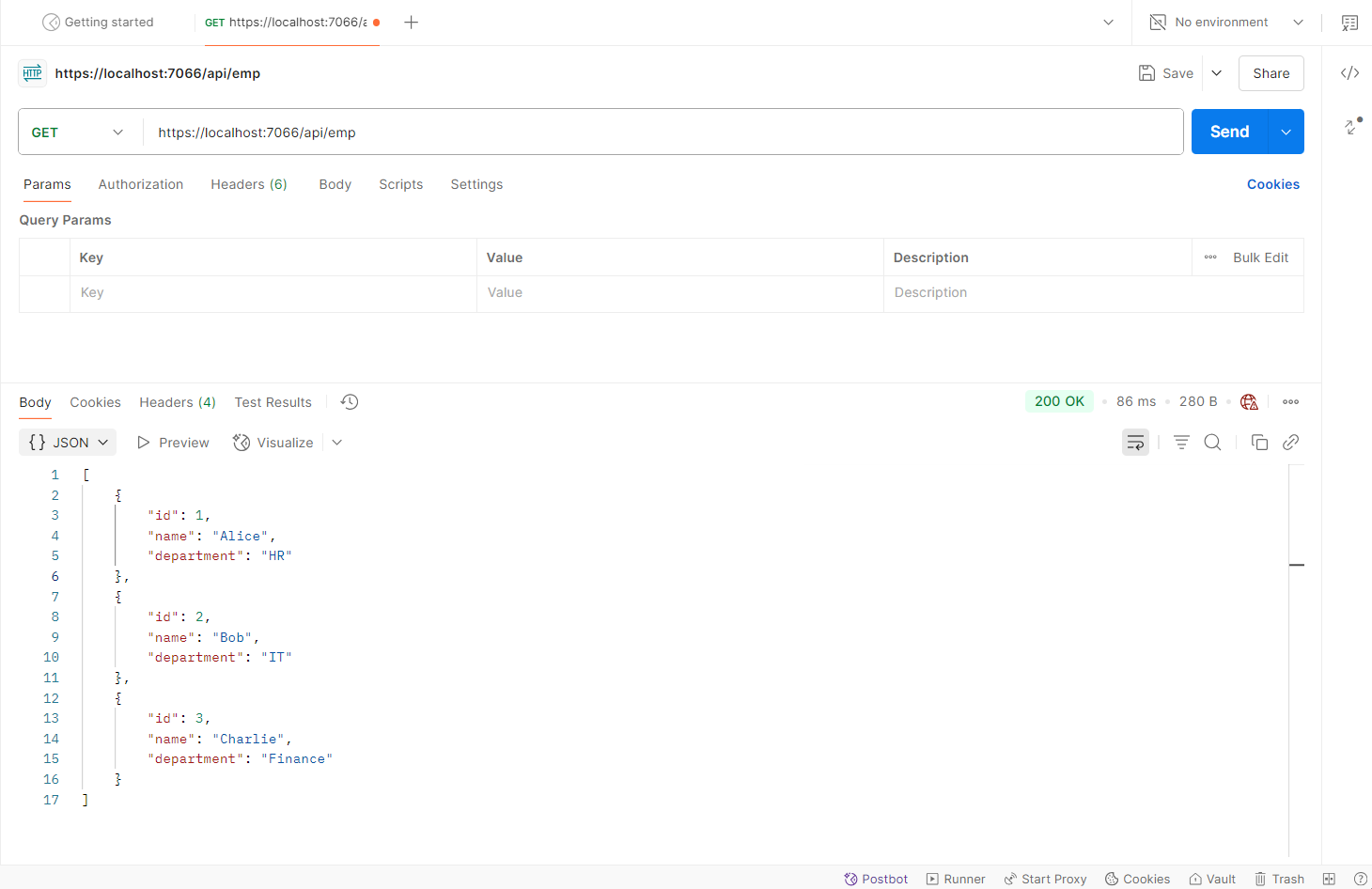
}

}

}

**OUTPUT**

****

****

3.**Explain the usage of FromBody attribute**

**Solution**:

The [FromBody] attribute tells ASP.NET Core to **bind a parameter** in your controller method **from the request body**, instead of the query string, route, or headers.

**CODE:**

Department.cs

namespace WebApi.Models

{

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

}

}

Employee.cs

using System;

namespace WebApi.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; }

}

}

Skill.cs

namespace WebApi.Models

{

public class Skill

{

public int Id { get; set; }

public string Name { get; set; }

}

}

EmployeeController.cs

using Microsoft.AspNetCore.Mvc;

using WebApi.Models;

//using WebApi.Filters;

namespace WebApi.Controllers

{

[ApiController]

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

//[ServiceFilter(typeof(CustomAuthFilter))] // Attach custom auth filter

public class EmployeeController : ControllerBase

{

private List<Employee> \_employees;

public EmployeeController()

{

\_employees = GetStandardEmployeeList();

}

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

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 = "Leadership" }

},

DateOfBirth = new DateTime(1990, 1, 1)

},

new Employee

{

Id = 2,

Name = "Jane Smith",

Salary = 70000,

Permanent = false,

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

Skills = new List<Skill>

{

new Skill { Id = 3, Name = "C#" },

new Skill { Id = 4, Name = "SQL" }

},

DateOfBirth = new DateTime(1985, 5, 15)

}

};

}

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public ActionResult<List<Employee>> GetStandard()

{

// Uncomment for exception testing:

// throw new Exception("Demo error in GET");

return Ok(\_employees);

}

[HttpPost]

public IActionResult AddEmployee([FromBody] Employee emp)

{

\_employees.Add(emp);

return Ok(emp);

}

[HttpPut("{id}")]

public IActionResult UpdateEmployee(int id, [FromBody] Employee emp)

{

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

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

employee.Name = emp.Name;

employee.Salary = emp.Salary;

employee.Permanent = emp.Permanent;

employee.Department = emp.Department;

employee.Skills = emp.Skills;

employee.DateOfBirth = emp.DateOfBirth;

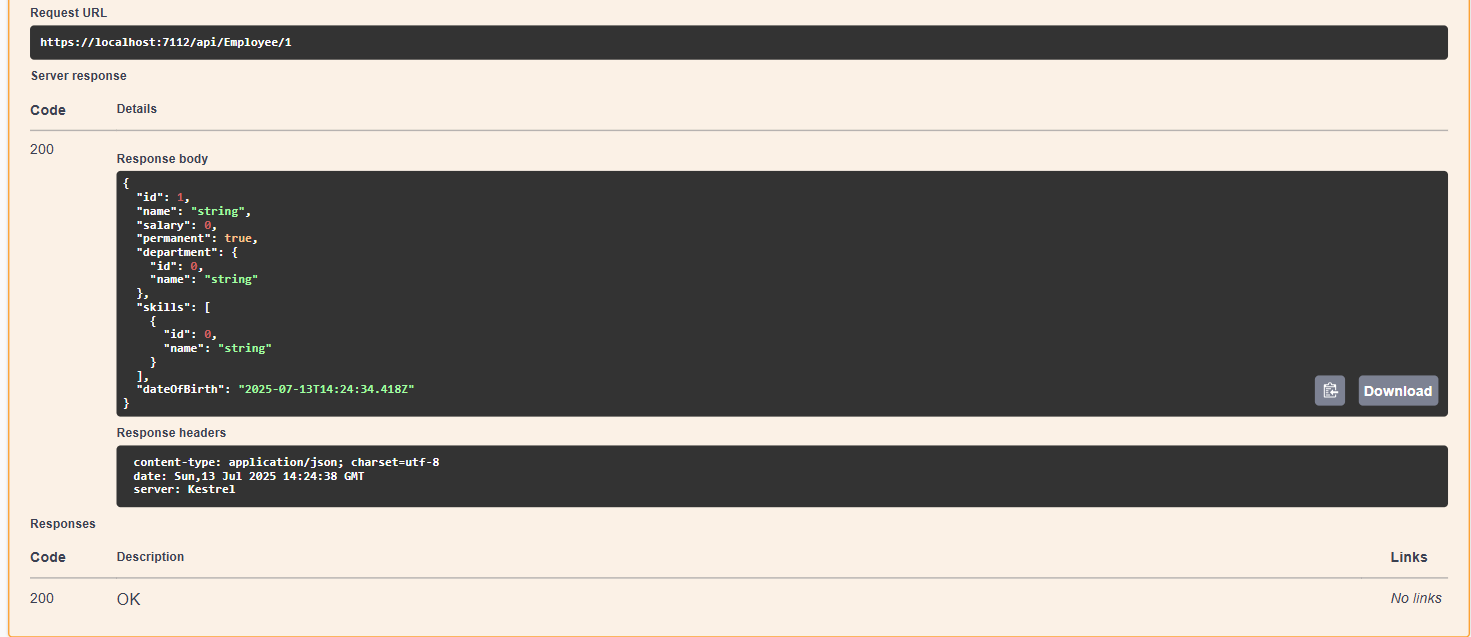
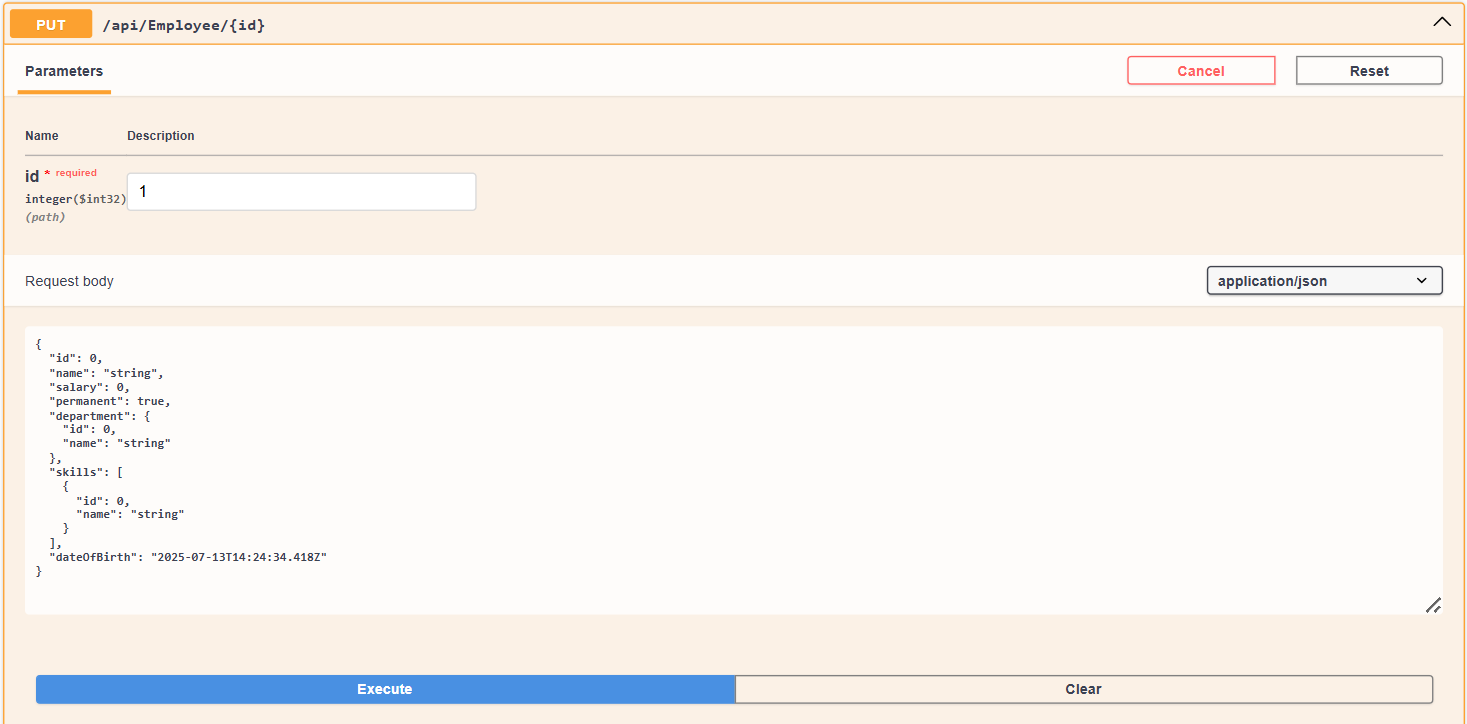
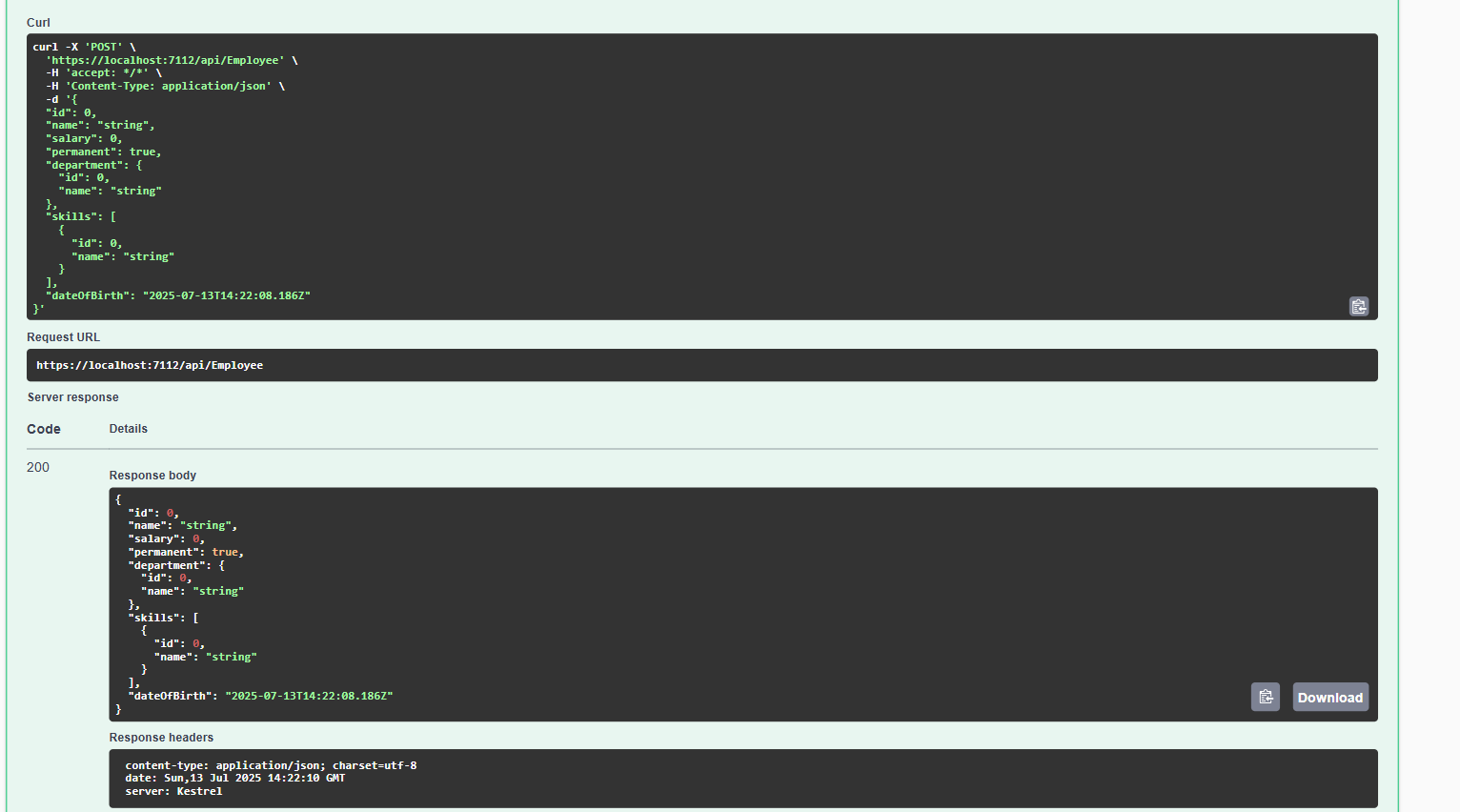
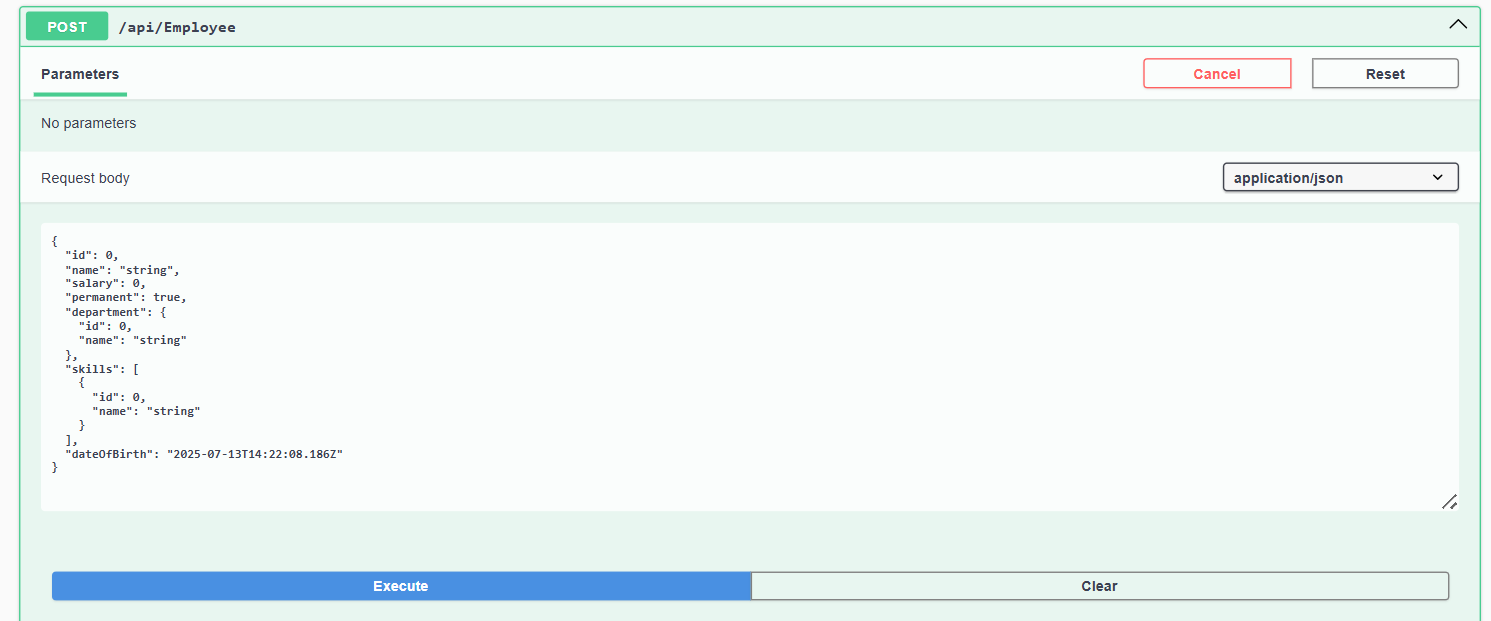
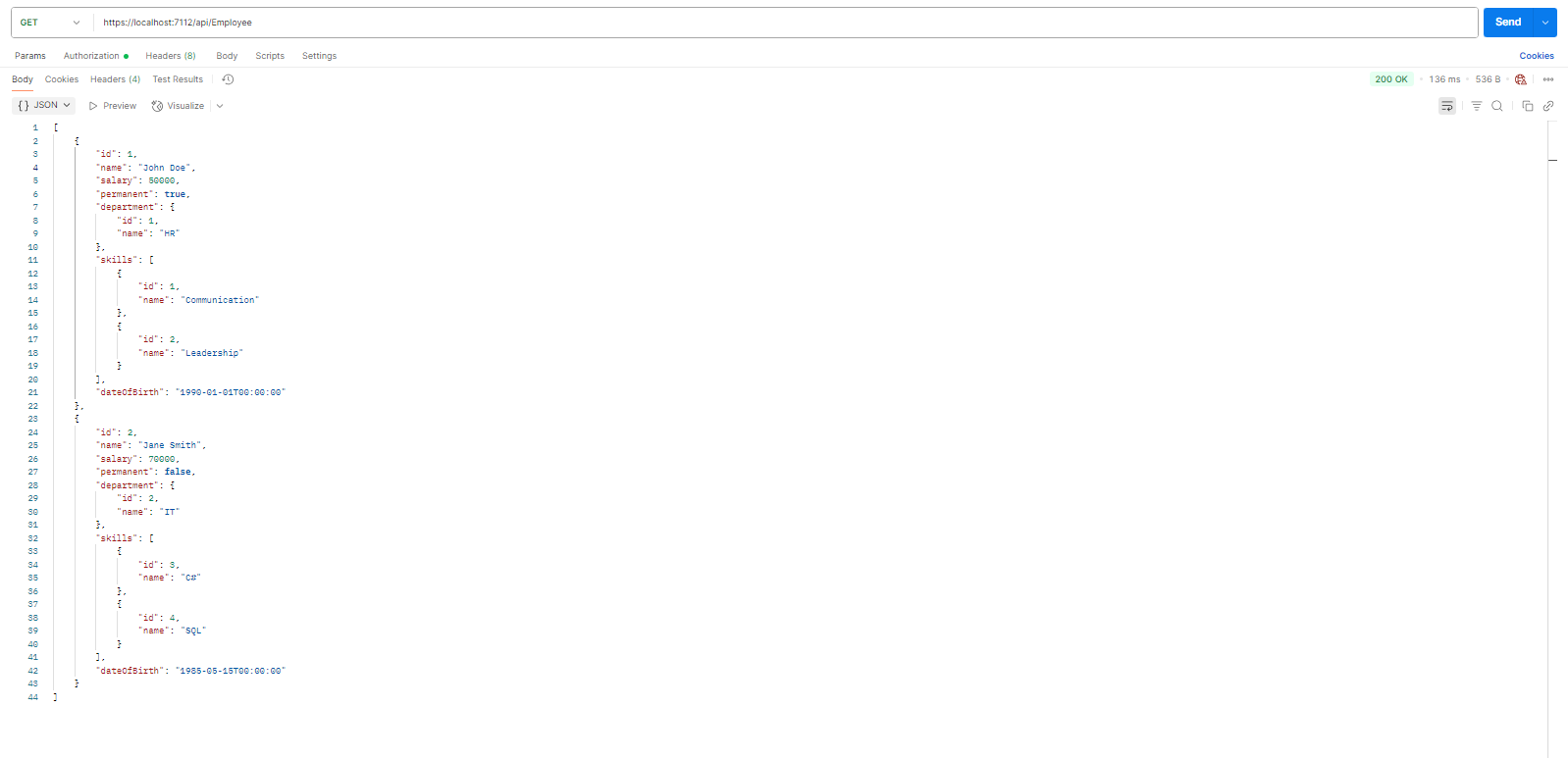
return Ok(employee);

}

}

}

**OUTPUT TEST1:**

****

**Code:**

**CustomAuthFilter.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace WebApi.Filters

{

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

var hasAuth = context.HttpContext.Request.Headers.TryGetValue("Authorization", out var token);

if (!hasAuth)

{

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

return;

}

if (!token.ToString().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 WebApi.Filters

{

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

var filePath = Path.Combine(Directory.GetCurrentDirectory(), "Logs", "errors.txt");

Directory.CreateDirectory(Path.GetDirectoryName(filePath)!);

File.AppendAllText(filePath,

$"[{DateTime.Now}] {context.Exception.Message}{Environment.NewLine}");

context.Result = new ObjectResult("An error occurred. Check logs.")

{

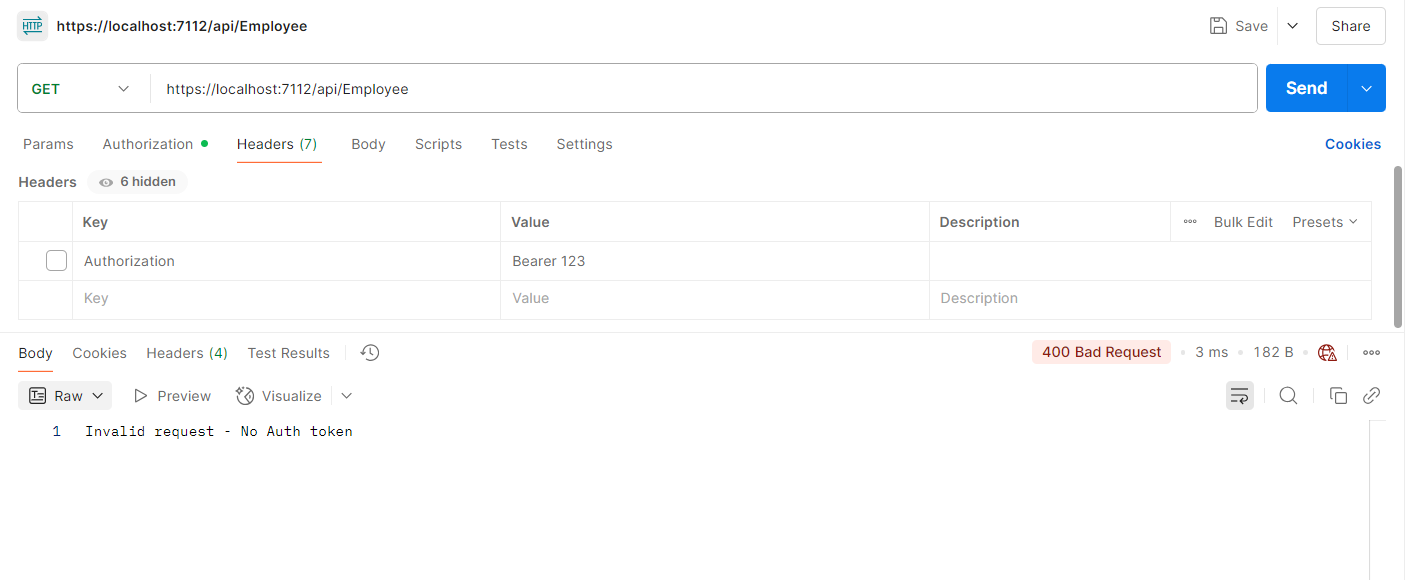
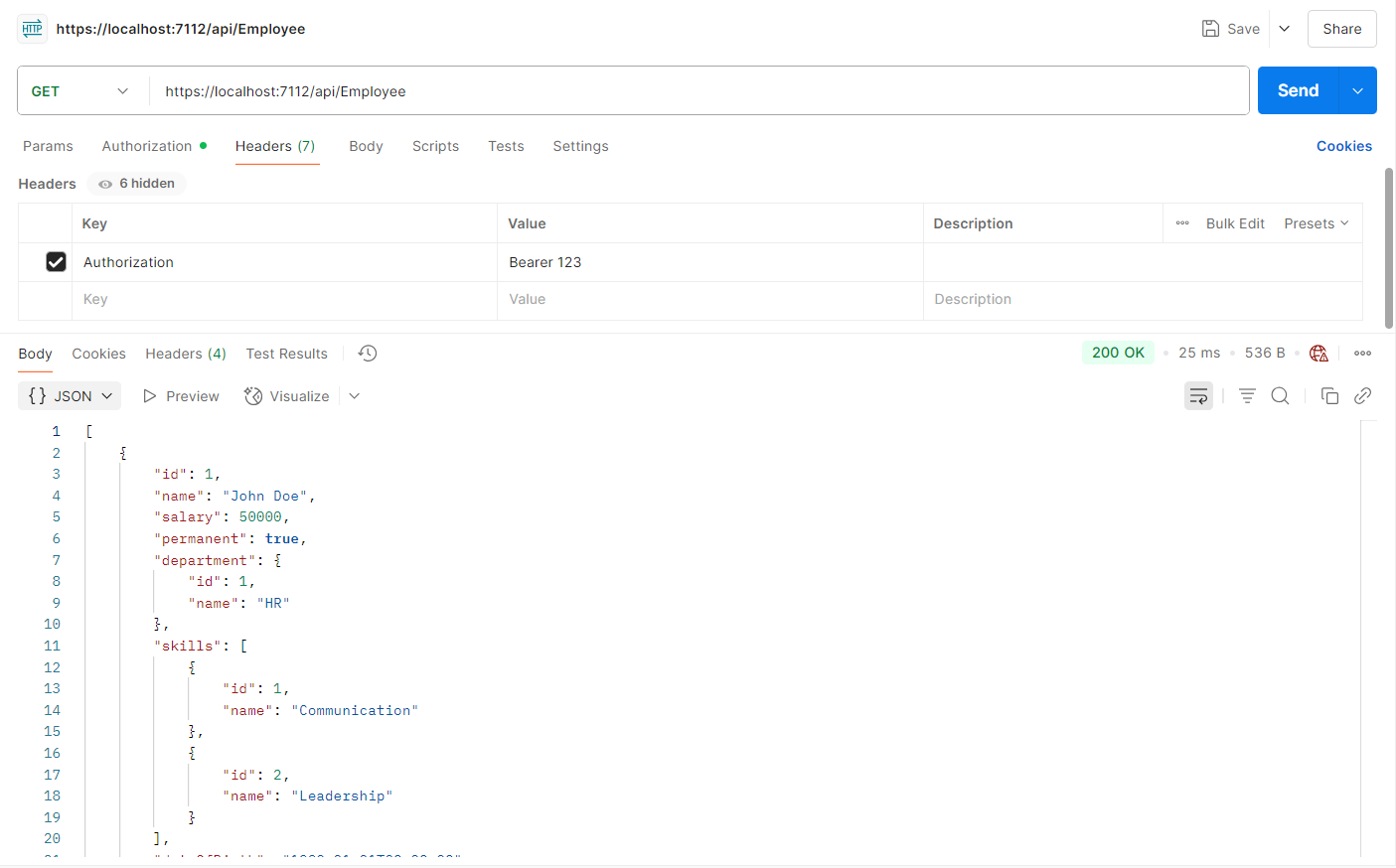
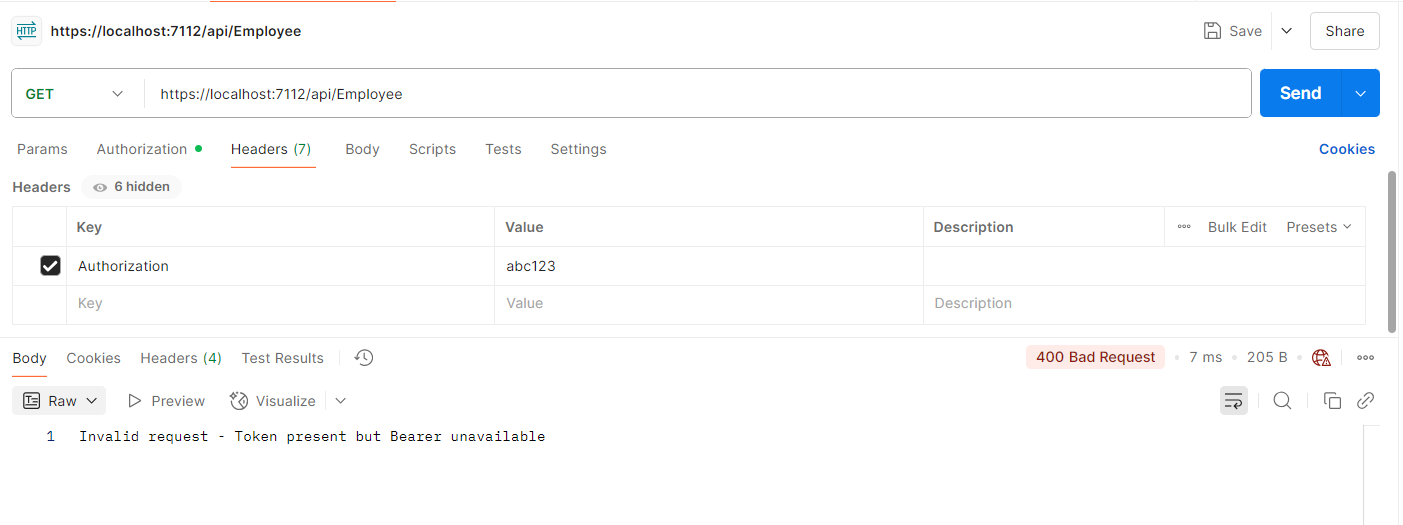
StatusCode = StatusCodes.Status500InternalServerError

};

}

}

}

**OUTPUT TEST2**

**CODE**

**EmployeeController.cs**

using Microsoft.AspNetCore.Mvc;

using WebApi.Models;

using WebApi.Filters;

namespace WebApi.Controllers

{

[ApiController]

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

[ServiceFilter(typeof(CustomAuthFilter))] // Attach custom auth filter

public class EmployeeController : ControllerBase

{

private List<Employee> \_employees;

public EmployeeController()

{

\_employees = GetStandardEmployeeList();

}

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

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 = "Leadership" }

},

DateOfBirth = new DateTime(1990, 1, 1)

},

new Employee

{

Id = 2,

Name = "Jane Smith",

Salary = 70000,

Permanent = false,

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

Skills = new List<Skill>

{

new Skill { Id = 3, Name = "C#" },

new Skill { Id = 4, Name = "SQL" }

},

DateOfBirth = new DateTime(1985, 5, 15)

}

};

}

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public ActionResult<List<Employee>> GetStandard()

{

throw new Exception("Demo error in GET"); // Uncomment this line

// return Ok(\_employees); -- Comment out the normal return

}

[HttpPost]

public IActionResult AddEmployee([FromBody] Employee emp)

{

\_employees.Add(emp);

return Ok(emp);

}

[HttpPut("{id}")]

public IActionResult UpdateEmployee(int id, [FromBody] Employee emp)

{

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

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

employee.Name = emp.Name;

employee.Salary = emp.Salary;

employee.Permanent = emp.Permanent;

employee.Department = emp.Department;

employee.Skills = emp.Skills;

employee.DateOfBirth = emp.DateOfBirth;

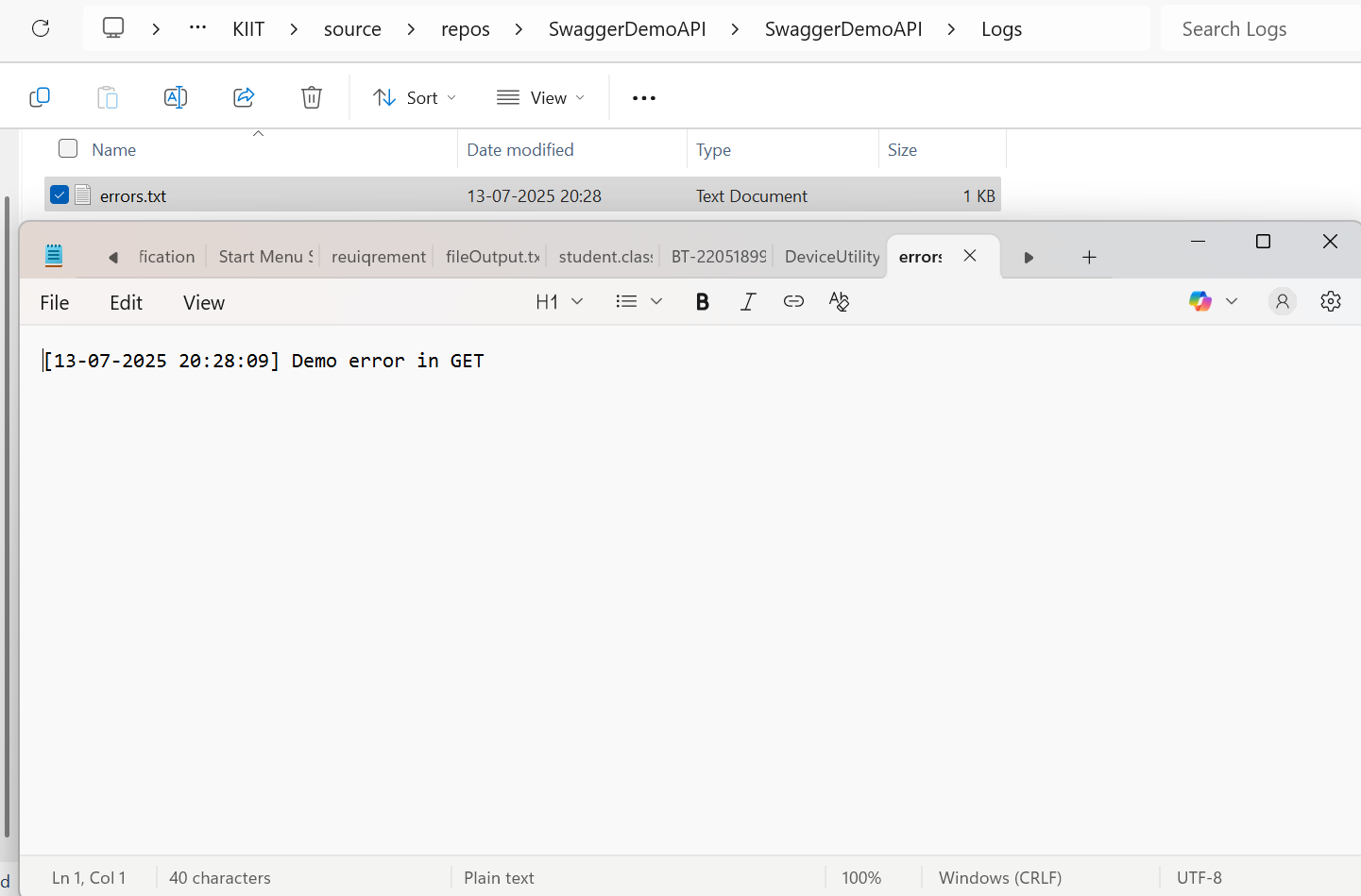
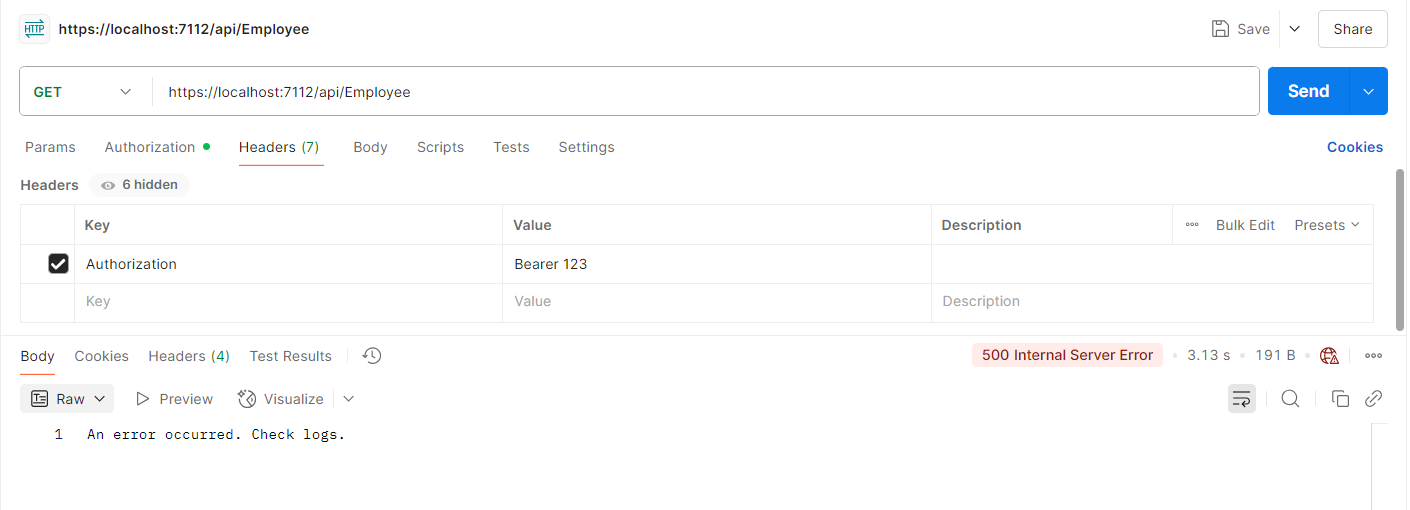
return Ok(employee);

}

}

}

**OUTPUT TEST3**

****

4. **Web Api CRUD operation**

**Update Employee data as per the input thru Web API PUT action method call**

**Employee information has to be updated based on the user input. Use Swagger tool to invoke the action method mapped with Http PUT action verb to update an employee data.**

**Modify the action method to return Employee data thru ActionResult.**

**Check if the id value is lesser than or equal to 0. If true, throw BadRequest action result with the message ‘Invalid employee id’**

**If the value is greater than 0 but not available in the list of employee ids that is there in the hardcoded list of employees, throw BadRequest action result with the same message as stated above.**

**If the id value is valid, use the JSON data from the input body and update the hardcoded list. Filter the employee list data for the input id and return that as the output.**

**CODE:**

**Employee.cs**

namespace HandsOn4.Model

{

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public string Department { get; set; }

public double Salary { get; set; }

}

}

**EmployeeController.cs**

namespace HandsOn4.Controllers

{

using HandsOn4.Model;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Linq;

[ApiController]

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

public class EmployeeController : ControllerBase

{

// Static hardcoded list simulating a DB

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

{

new Employee { Id = 1, Name = "John", Department = "HR", Salary = 50000 },

new Employee { Id = 2, Name = "Alice", Department = "IT", Salary = 60000 },

new Employee { Id = 3, Name = "Bob", Department = "Finance", Salary = 55000 }

};

// PUT: api/employee/2

[HttpPut("{id}")]

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

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (existingEmp == null)

{

return BadRequest("Invalid employee id");

}

// Update employee details

existingEmp.Name = updatedEmp.Name;

existingEmp.Department = updatedEmp.Department;

existingEmp.Salary = updatedEmp.Salary;

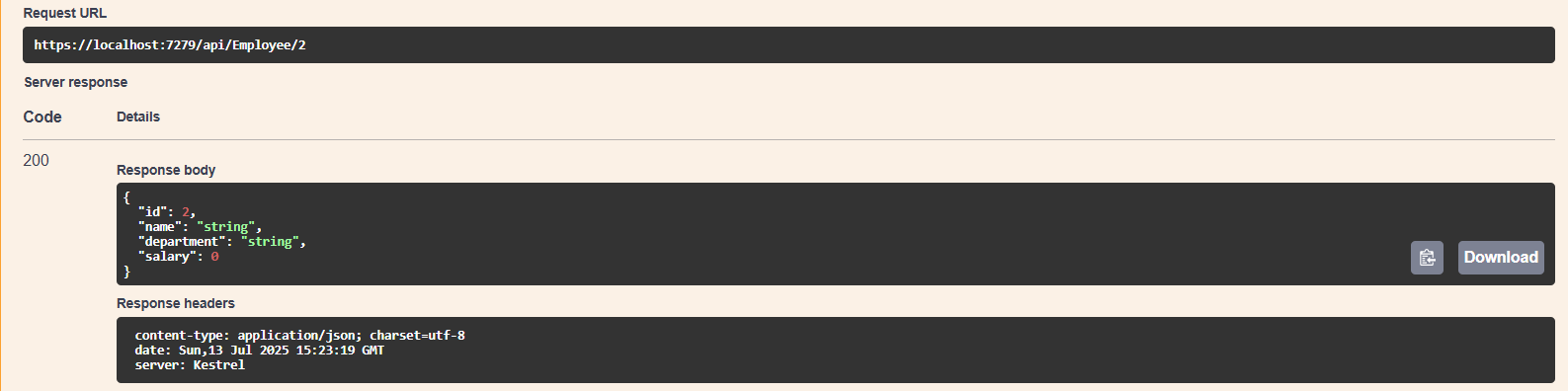
return Ok(existingEmp);

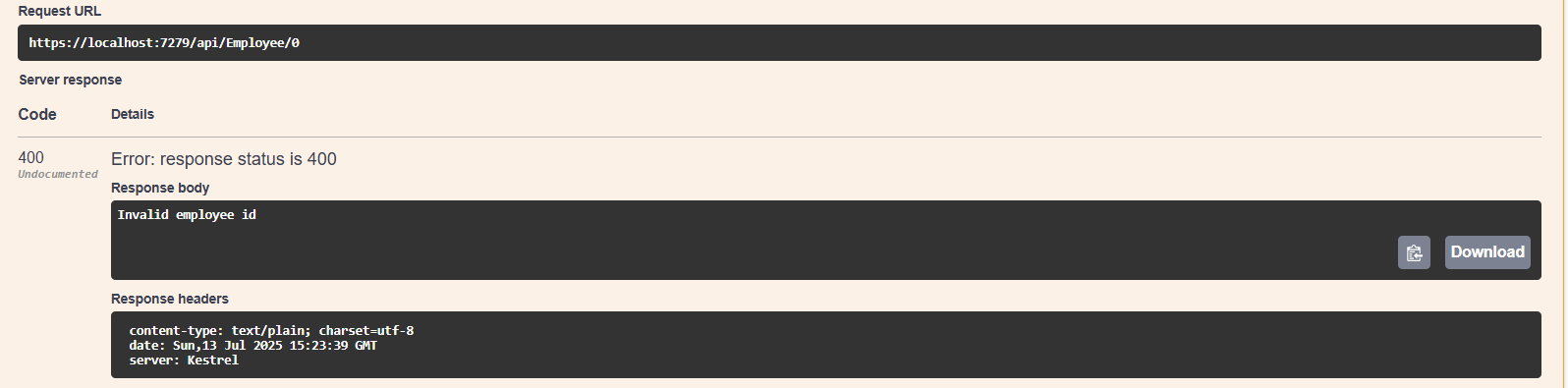
}

}

}

**OUTPUT:**





5.

**WebApi\_Handson**

**JsonWebToken**

There are various modes of authenticating a request. Json Web Token(JWT) authentication is one among them. It is a methodology of passing a token in the Authorization header value in the request so that it can be checked at the WebAPI and validated. If not there, then ‘**Unauthorized**’ status message with status code 401 should be thrown.

Use the below code in Startup.cs

* 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),

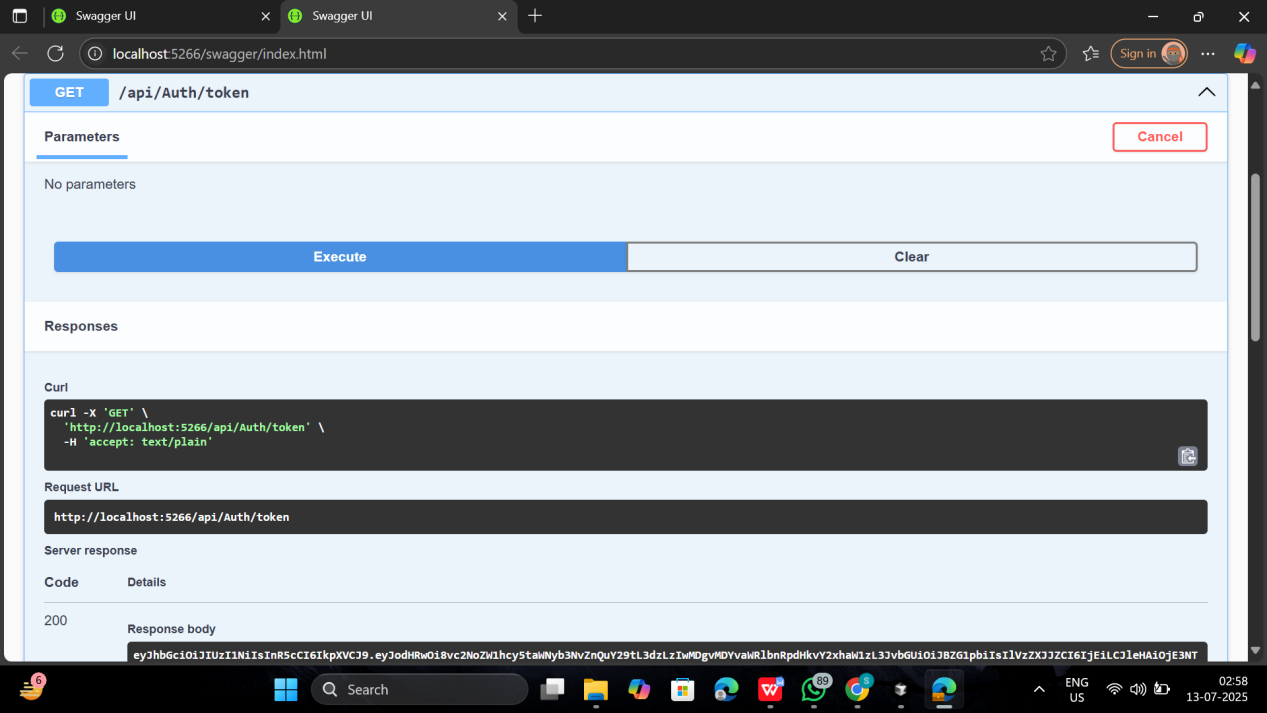
signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

Note that the issuer, audience and the securitykey defined in the Startup.cs code and method code shown above is the same and should match.

Invoke the GenerateJSONWebToken in the GET action method by sending some value for user id and ‘Admin’ for the user role. This is to set Claims information to check the user role



**Use the JWT generated thru the AuthController to be used in POSTMAN request.**

Remove the ‘CustomAuthFilter’ that is currently mapped to the Employee controller(if already done)

Use **Authorize** attribute in the Employee controller to check if the request header contains ‘Authorization’ key with ‘Bearer’ token or not. If the token is unavailable, then ‘Unauthorized’ status message whose code 401 will be thrown. Use POSTMAN to hit a GET action method call. Note the ‘Status’ attribute in the ‘Headers’ section in the output window.

Use the AuthController to generate the JWT. Use that in the GET action method call thru POSTMAN if the request is authenticated or not.

Modify the token value in the POSTMAN tool and check if ‘Unauthorized’ status message is thrown. Note the ‘Status’ attribute in the ‘Headers’ section in the output window.

**CODE:**

using Microsoft.AspNetCore.Mvc;

using EmployeeWebApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace EmployeeWebApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class EmployeeController : ControllerBase

    {

        private static List<Employee> \_employees = new List<Employee>

        {

            new Employee

            {

                Id = 1,

                Name = "John Doe",

                Salary = 50000,

                Permanent = true,

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

                Skills = new List<Skill>

                {

    new Skill { Id = 1, Name = "C#", Description = "Programming" }

},

                DateOfBirth = new DateTime(1990, 1, 1)

            },

            new Employee

            {

                Id = 2,

                Name = "Jane Smith",

                Salary = 60000,

                Permanent = false,

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

                Skills = new List<Skill>

                {

    new Skill { Id = 2, Name = "Recruitment", Description = "HR

Skill" }

},

                DateOfBirth = new DateTime(1992, 2, 2)

            }

        };

        [HttpPut("{id}")]

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

        {

            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 = employee.Name;

            existing.Salary = employee.Salary;

            existing.Permanent = employee.Permanent;

            existing.Department = employee.Department;

            existing.Skills = employee.Skills;

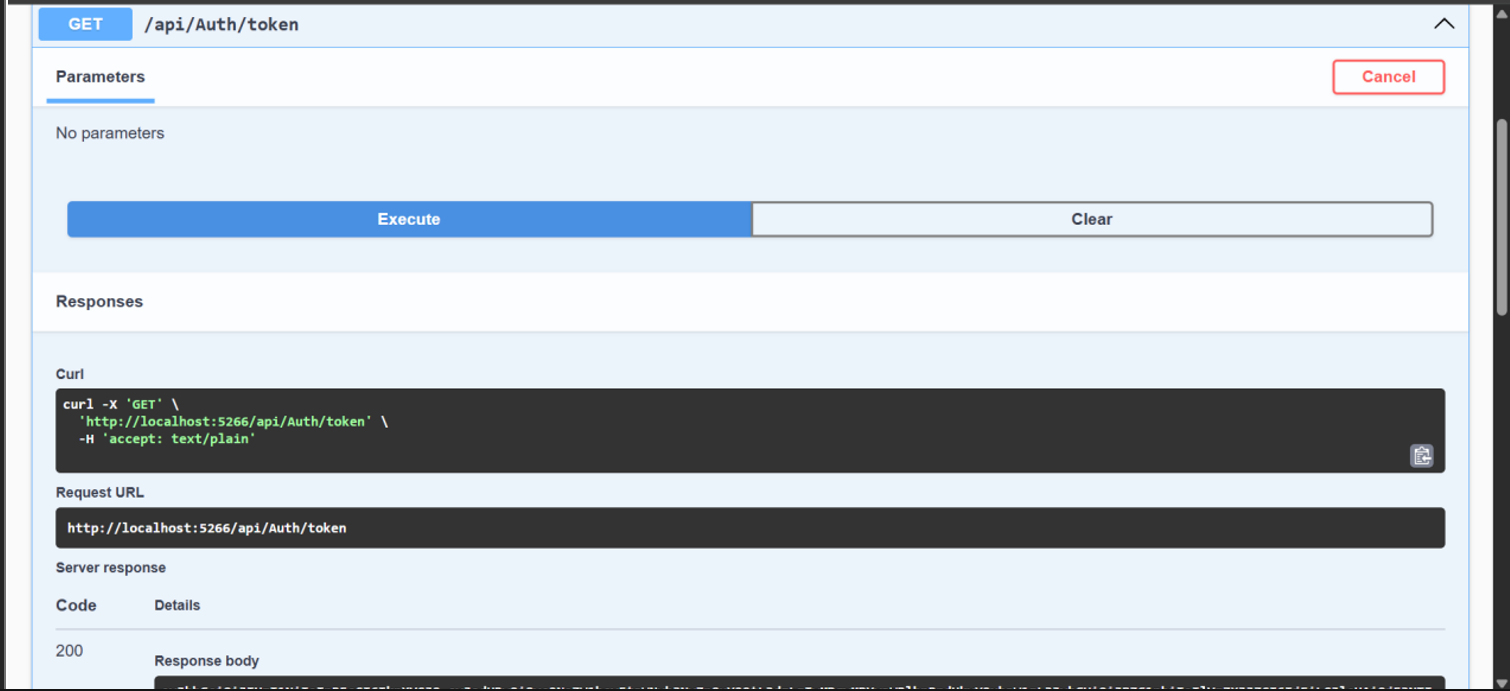
            existing.DateOfBirth = employee.DateOfBirth;

            return Ok(existing);

        }

    }

}

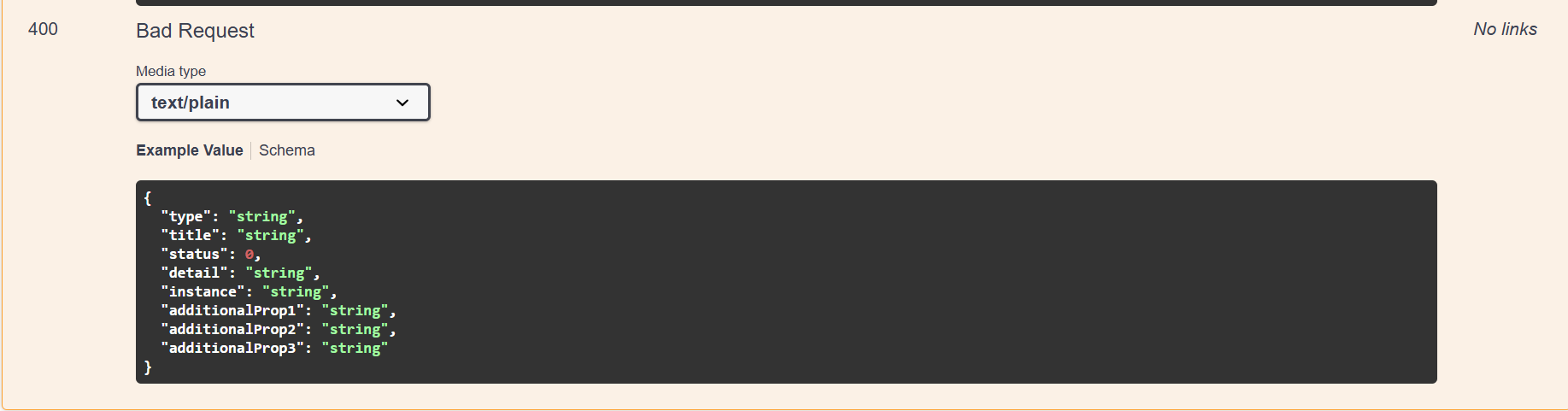


**Check for JWT expiration**

A JWT token has an attribute which can be set to determine how long the token is valid to be used.

In the GenerateJSONWebToken method in AuthController, the ‘expires’ attribute of the JwtSecurityToken object denotes the time in minutes for which the token would be valid.

Modify the duration for ‘expires’ attribute to 2 minutes. Check the POSTMAN request for GET call **AFTER** 2 minutes of generation of the JWT, which should yield ‘Unauthorized’ message with Http status code 401.



**Add the roles to be authorized in the Authorize attribute.**

The **Authorize** attribute supports the roles to be used to filter the controller action method access.

The GenerateJSONWebToken method of AuthController uses the role ‘Admin’ set in the claims.

Include the role ‘**POC**’ in the Authorize attribute in the Employee controller. Hit the GET action method of the Employee controller thru POSTMAN. Verify if the response status is ‘Unauthorized’ with status code 401

Include the role ‘**Admin**’ along with ‘POC’ in the Authorize attribute in the Employee controller. Hit the GET action method of the Employee controller thru POSTMAN. Verify if the response status is OK with status code 200

using Microsoft.AspNetCore.Mvc;

using EmployeeWebApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace EmployeeWebApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class EmployeeController : ControllerBase

    {

        private static List<Employee> \_employees = new List<Employee>

        {

            new Employee

            {

                Id = 1,

                Name = "John Doe",

                Salary = 50000,

                Permanent = true,

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

                Skills = new List<Skill>

                {

    new Skill { Id = 1, Name = "C#", Description = "Programming" }

},

                DateOfBirth = new DateTime(1990, 1, 1)

            },

            new Employee

            {

                Id = 2,

                Name = "Jane Smith",

                Salary = 60000,

                Permanent = false,

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

                Skills = new List<Skill>

                {

    new Skill { Id = 2, Name = "Recruitment", Description = "HR

Skill" }

},

                DateOfBirth = new DateTime(1992, 2, 2)

            }

        };

        [HttpPut("{id}")]

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

        {

            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 = employee.Name;

            existing.Salary = employee.Salary;

            existing.Permanent = employee.Permanent;

            existing.Department = employee.Department;

            existing.Skills = employee.Skills;

            existing.DateOfBirth = employee.DateOfBirth;

            return Ok(existing);

        }

    }}