

Superset ID: 6364957

Web API Solution with .NET 9

Exercise 1: First Web API Using .NET Core

Step 1: Creating a New Web API Project

```
cd C:\Users\KIIT\OneDrive\Desktop\Web-API
dotnet new webapi -n MyFirstWebAPI --use-controllers
cd MyFirstWebAPI
dotnet run
```

Step 2: Understanding the Default Structure

Key files created:

- Program.cs: Entry point and configuration.
- Controllers/WeatherForecastController.cs: Sample controller.
- WeatherForecast.cs: Model class.

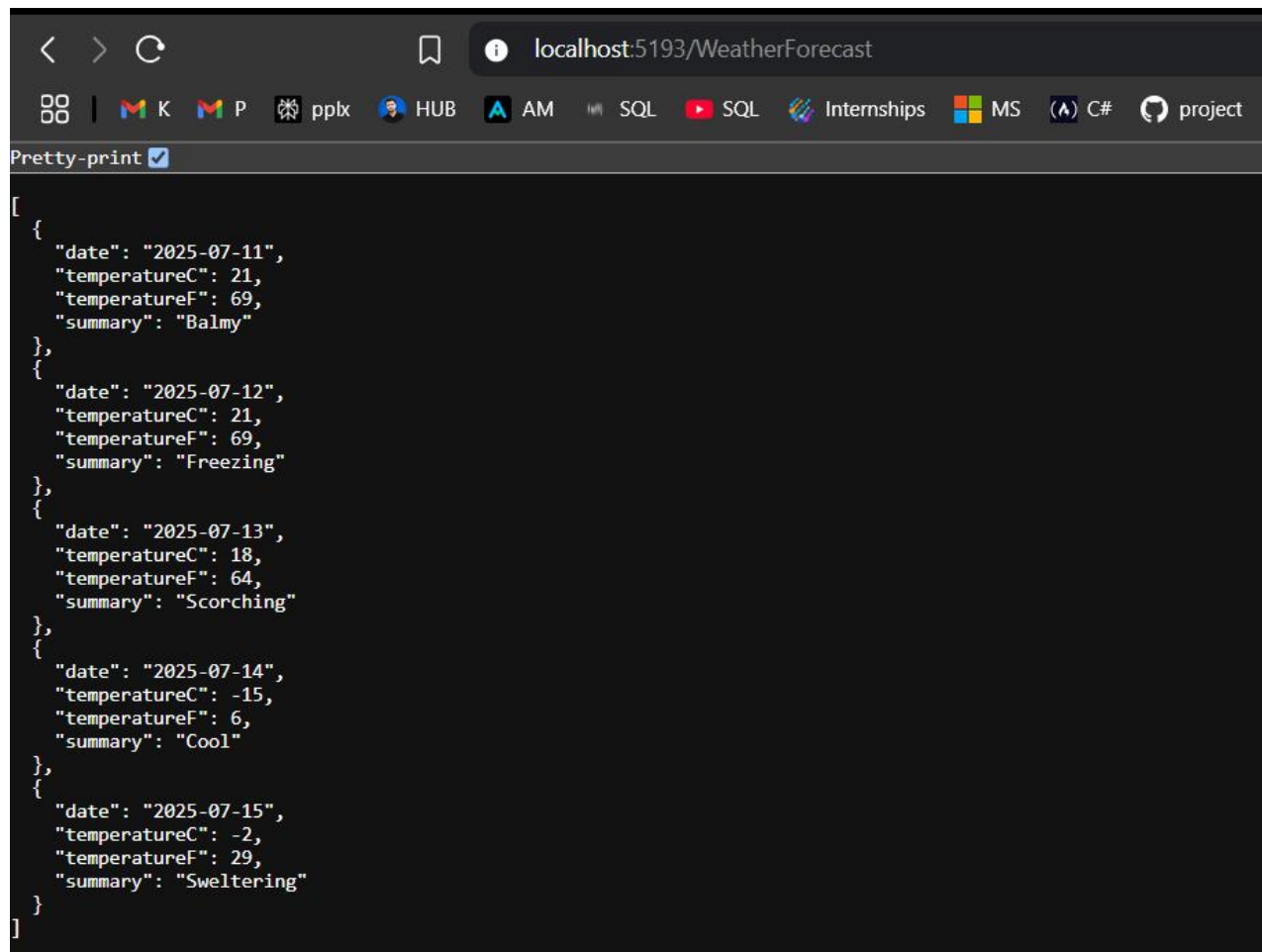
Step 3: Testing the Default API

1. Running dotnet run.

```
PS C:\Users\KIIT\OneDrive\Desktop\Web-API> cd .\MyFirstWebAPI\
PS C:\Users\KIIT\OneDrive\Desktop\Web-API\MyFirstWebAPI> dotnet run
Using launch settings from C:\Users\KIIT\OneDrive\Desktop\Web-API\MyFirstWebAPI\Properties\launchSettings.json...
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5193
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\KIIT\OneDrive\Desktop\Web-API\MyFirstWebAPI
```

2. Running `http://localhost:5193/WeatherForecast` (JSON response as output) :

Output:



The screenshot shows a web browser window with the address bar displaying 'localhost:5193/WeatherForecast'. The browser's taskbar at the top includes icons for various applications like Google, VS Code, and others. Below the address bar, there is a 'Pretty-print' button. The main content area displays a JSON array of five weather forecast objects, each containing a date, temperature in Celsius and Fahrenheit, and a summary.

```
[
  {
    "date": "2025-07-11",
    "temperatureC": 21,
    "temperatureF": 69,
    "summary": "Balmy"
  },
  {
    "date": "2025-07-12",
    "temperatureC": 21,
    "temperatureF": 69,
    "summary": "Freezing"
  },
  {
    "date": "2025-07-13",
    "temperatureC": 18,
    "temperatureF": 64,
    "summary": "Scorching"
  },
  {
    "date": "2025-07-14",
    "temperatureC": -15,
    "temperatureF": 6,
    "summary": "Cool"
  },
  {
    "date": "2025-07-15",
    "temperatureC": -2,
    "temperatureF": 29,
    "summary": "Sweltering"
  }
]
```

Exercise 2: Web API with Swagger

Step 1: Installing Swagger Package

```
dotnet add package Swashbuckle.AspNetCore
```

Step 2: Configuring Swagger in Program.cs

Replacing the content of Program.cs with:

```
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 = "Swagger Demo",
        Version = "v1",
        Description = "TBD",
        TermsOfService = new Uri("https://example.com/terms"),
        Contact = new OpenApiContact
        {
            Name = "Sachin Ray\\",
            Email = "sachin@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();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");
    });
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

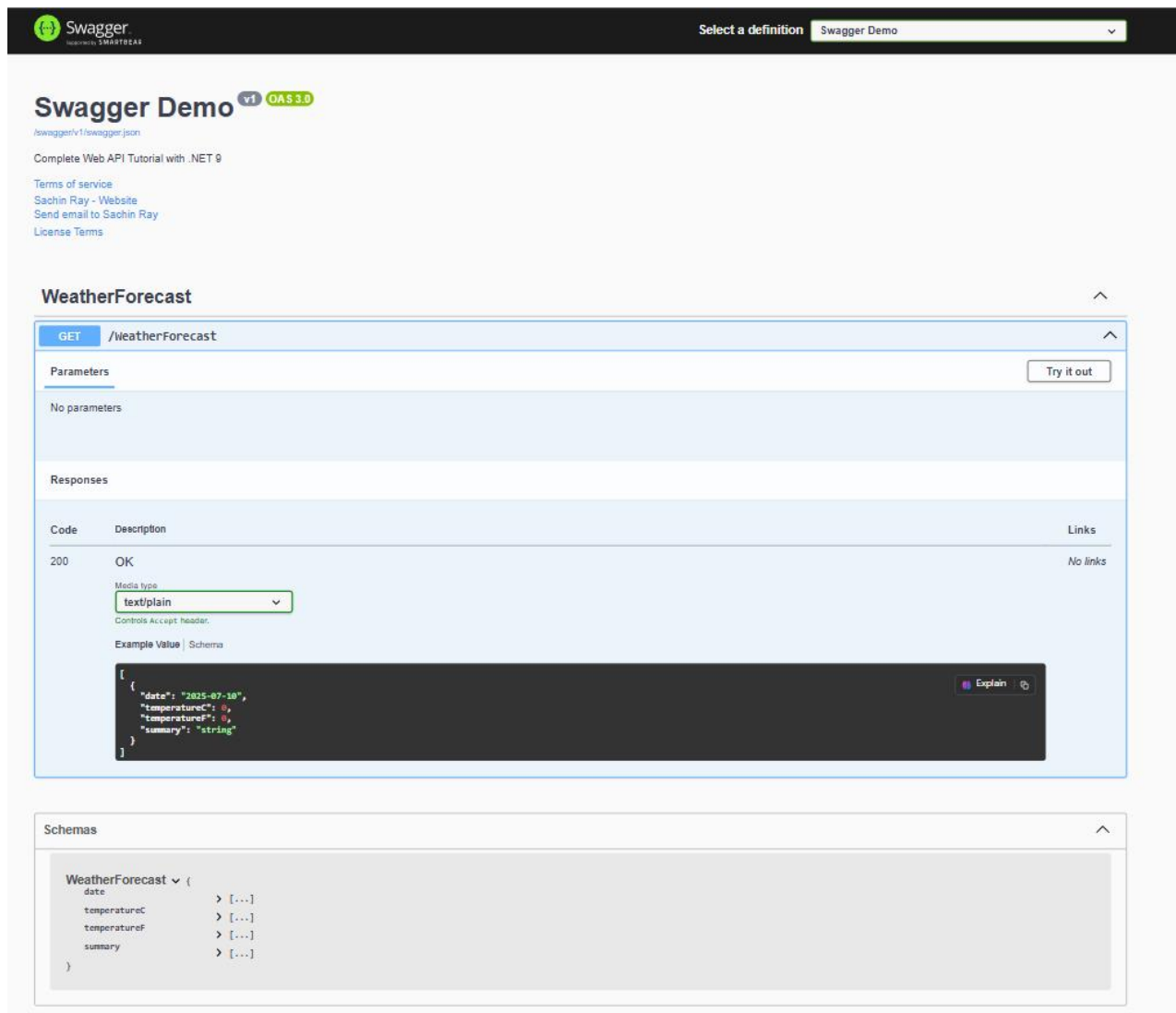
app.Run();

```

Step 3: Testing Swagger

- Run `dotnet run`.
- Visiting <http://localhost:5193/swagger> to view the Swagger UI with API documentation.

Output:



The image shows the Swagger UI for a service named "Swagger Demo". The top navigation bar includes the Swagger logo, the text "Powered by SMARTBEAR", and a dropdown menu labeled "Select a definition" with "Swagger Demo" selected. Below the header, the service name "Swagger Demo" is displayed with a version tag "v1 OAS 3.0" and the path "/swagger/v1/swagger.json". A description "Complete Web API Tutorial with .NET 9" is provided, along with links for "Terms of service", "Sachin Ray - Website", "Send email to Sachin Ray", and "License Terms".

The main section is titled "WeatherForecast" and shows the details for the GET endpoint "/weatherForecast". It includes a "Parameters" section with "No parameters" and a "Responses" section. The response table shows a 200 status code with the description "OK". A dropdown menu for "Media type" is set to "text/plain". Below this, an "Example Value" is displayed in a dark box with a light blue border, showing a JSON object:

```
{  "date": "2025-07-10",  "temperatureC": 0,  "temperatureF": 0,  "summary": "string"}
```

. A "Try it out" button is located in the top right corner of the response section.

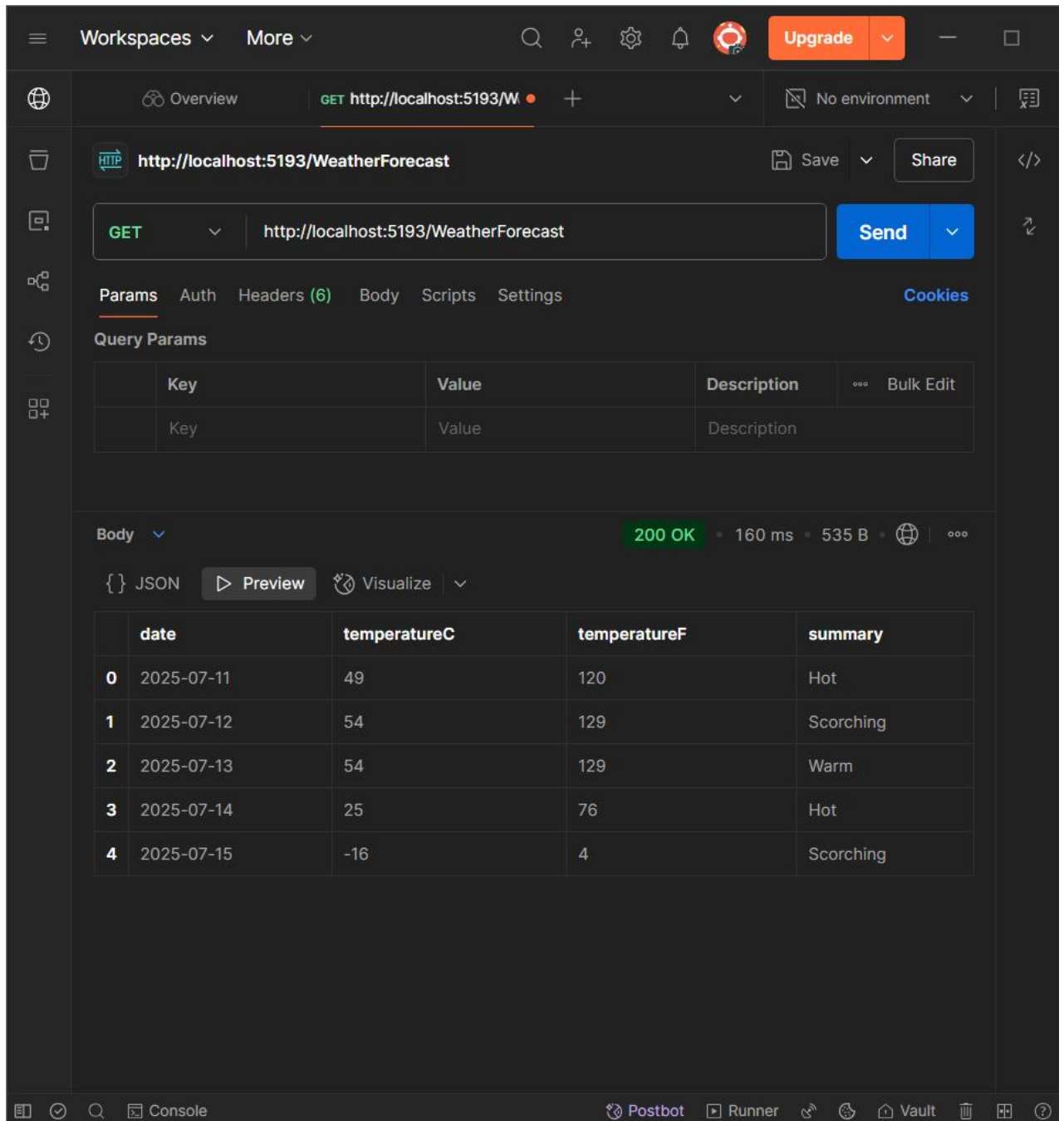
The "Schemas" section at the bottom shows the "WeatherForecast" schema, which is a JSON object with the following properties: "date" (array), "temperatureC" (array), "temperatureF" (array), and "summary" (array). Each property is followed by a link to its definition: "> [...]".

Step 4: Testing with Postman

- Create a new GET request to <http://localhost:5193/WeatherForecast>.

- Clicking Send;

Response:



The screenshot shows the Postman interface with a GET request to `http://localhost:5193/WeatherForecast` successfully executed. The response is a 200 OK status with a response time of 160 ms and a body size of 535 B. The response body is a JSON array of weather forecast objects.

	date	temperatureC	temperatureF	summary
0	2025-07-11	49	120	Hot
1	2025-07-12	54	129	Scorching
2	2025-07-13	54	129	Warm
3	2025-07-14	25	76	Hot
4	2025-07-15	-16	4	Scorching

Step 5: Modifying Controller Route

Creating Controllers/EmployeeController.cs:

```
using Microsoft.AspNetCore.Mvc;
```

```

namespace MyFirstWebAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class EmployeeController : ControllerBase
    {
        [HttpGet]
        public ActionResult<string> Get()
        {
            return Ok("Employee data retrieved successfully");
        }
    }
}

```

- Changing the route by modify the Route attribute to [Route("api/Emp")] and testing in Postman after running freshly.

Output:

The screenshot shows the Postman interface with a GET request to `http://localhost:5193/api/emp` successfully executed. The response is a 200 OK status with a body containing the text "Employee data retrieved successfully".

Query Params

Key	Value	Description
Key	Value	Description

Body

200 OK • 9 ms • 178 B • ...

Raw Preview Visualize

Employee data retrieved successfully

Exercise 3: Custom Model Class and Filters

1. Creating Models Folder and Classes

2.1 Creating the Models Folder

```
mkdir Models
```

2.2 Adding Employee.cs Model

Create Models/Employee.cs with the following content:

```
namespace MyFirstWebAPI.Models
{
    public class Employee
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
        public int Salary { get; set; }
        public bool Permanent { get; set; }
        public Department Department { get; set; } = new Department();
        public List<Skill> Skills { get; set; } = new List<Skill>();
        public DateTime DateOfBirth { get; set; }
    }

    public class Department
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
    }

    public class Skill
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
    }
}
```

3. Updating Employee Controller with Custom Models

3.1 Replace EmployeeController.cs

Overwriting Controllers/EmployeeController.cs with:

```
using Microsoft.AspNetCore.Mvc;
using MyFirstWebAPI.Models;

namespace MyFirstWebAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class EmployeeController : ControllerBase
    {
        private static List<Employee> _employees = new List<Employee>();

        public EmployeeController()
        {
            if (_employees.Count == 0)
            {
                _employees = GetStandardEmployeeList();
            }
        }

        [HttpGet]
        [ProducesResponseType(typeof(List<Employee>), 200)]
        public ActionResult<List<Employee>> Get()
        {
            return Ok(_employees);
        }

        [HttpGet("{id}")]
        [ProducesResponseType(typeof(Employee), 200)]
        [ProducesResponseType(404)]
        public ActionResult<Employee> Get(int id)
        {
            var employee = _employees.FirstOrDefault(e => e.Id == id);
            if (employee == null)
            {
                return NotFound($"Employee with ID {id} not found");
            }
            return Ok(employee);
        }

        [HttpPost]
        [ProducesResponseType(typeof(Employee), 201)]
        [ProducesResponseType(400)]
        public ActionResult<Employee> Post([FromBody] Employee employee)
```



```

{
    if (employee == null)
        return BadRequest("Employee data is required");

    employee.Id = _employees.Count > 0 ? _employees.Max(e => e.Id) + 1 : 1;
    _employees.Add(employee);
    return CreatedAtAction(nameof(Get), new { id = employee.Id }, employee);
}

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 = "IT" },
            Skills = new List<Skill> { new Skill { Id = 1, Name = "C#" }, new Skill { Id = 2, Name = "ASP.NET" } },
            DateOfBirth = new DateTime(1990, 1, 1)
        },
        new Employee
        {
            Id = 2,
            Name = "Jane Smith",
            Salary = 60000,
            Permanent = false,
            Department = new Department { Id = 2, Name = "HR" },
            Skills = new List<Skill> { new Skill { Id = 3, Name = "Management" }, new Skill { Id = 4, Name =
"Communication" } },
            DateOfBirth = new DateTime(1985, 5, 15)
        },
        new Employee
        {
            Id = 3,
            Name = "Bob Johnson",
            Salary = 55000,
            Permanent = true,
            Department = new Department { Id = 1, Name = "IT" },
            Skills = new List<Skill> { new Skill { Id = 5, Name = "JavaScript" }, new Skill { Id = 6, Name = "React" } },
            DateOfBirth = new DateTime(1988, 12, 10)
        }
    };
};

```

```
}  
}  
}
```

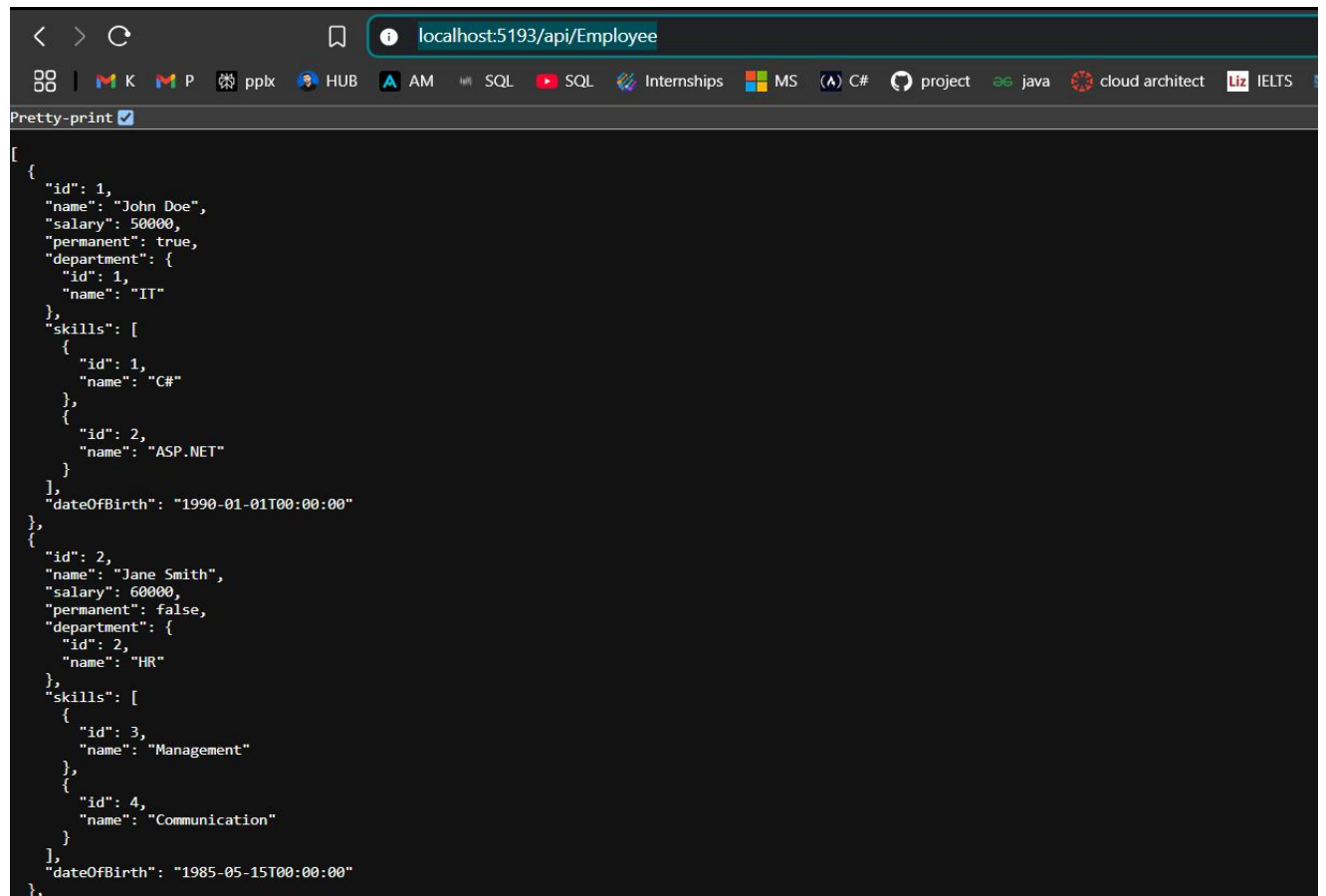
4. Test Custom Models

4.1 Build and Run

```
dotnet build  
dotnet run
```

4.2 Testing in Browser

3. Going to: <http://localhost:5193/api/Employee>
4. **Output:**

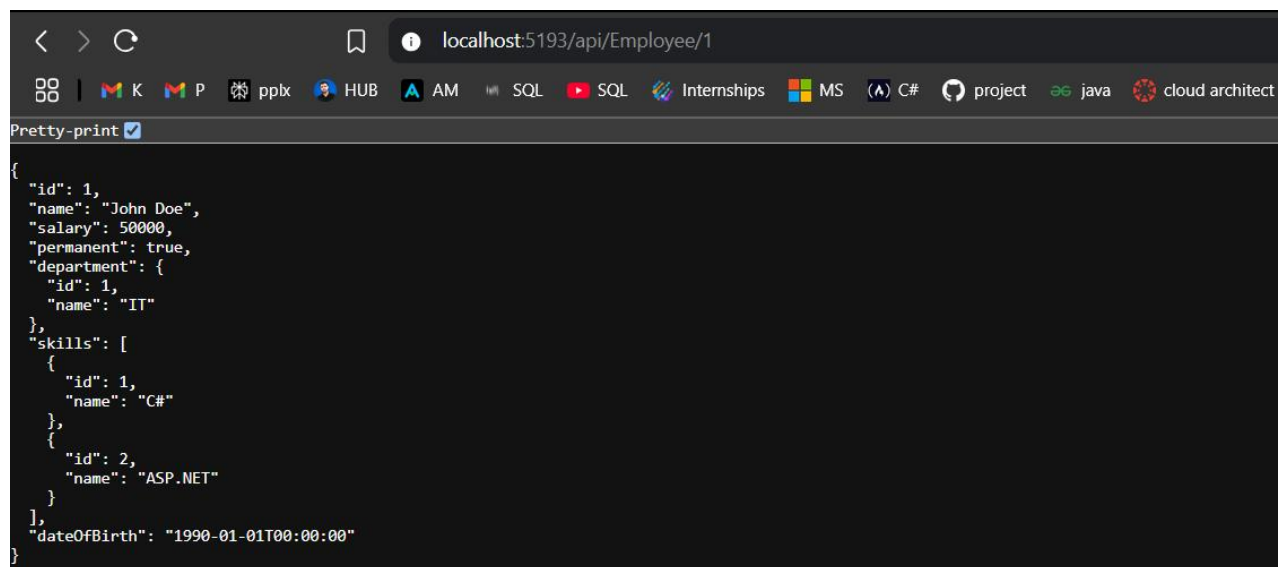


```
[  
  {  
    "id": 1,  
    "name": "John Doe",  
    "salary": 50000,  
    "permanent": true,  
    "department": {  
      "id": 1,  
      "name": "IT"  
    },  
    "skills": [  
      {  
        "id": 1,  
        "name": "C#"  
      },  
      {  
        "id": 2,  
        "name": "ASP.NET"  
      }  
    ],  
    "dateOfBirth": "1990-01-01T00:00:00"  
  },  
  {  
    "id": 2,  
    "name": "Jane Smith",  
    "salary": 60000,  
    "permanent": false,  
    "department": {  
      "id": 2,  
      "name": "HR"  
    },  
    "skills": [  
      {  
        "id": 3,  
        "name": "Management"  
      },  
      {  
        "id": 4,  
        "name": "Communication"  
      }  
    ],  
    "dateOfBirth": "1985-05-15T00:00:00"  
  }  
]
```

```
{
  "id": 3,
  "name": "Bob Johnson",
  "salary": 55000,
  "permanent": true,
  "department": {
    "id": 1,
    "name": "IT"
  },
  "skills": [
    {
      "id": 5,
      "name": "JavaScript"
    },
    {
      "id": 6,
      "name": "React"
    }
  ],
  "dateOfBirth": "1988-12-10T00:00:00"
}
```

4.3 Testing Single Employee

- Going to: `http://localhost:5193/api/Employee/1`
- **Output:**



```
Pretty-print
{
  "id": 1,
  "name": "John Doe",
  "salary": 50000,
  "permanent": true,
  "department": {
    "id": 1,
    "name": "IT"
  },
  "skills": [
    {
      "id": 1,
      "name": "C#"
    },
    {
      "id": 2,
      "name": "ASP.NET"
    }
  ],
  "dateOfBirth": "1990-01-01T00:00:00"
}
```

5. Create Custom Auth Filter

5.1 Creating Filters Folder

```
mkdir Filters
```

5.2 Adding CustomAuthFilter.cs

Creating Filters/CustomAuthFilter.cs:

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

namespace MyFirstWebAPI.Filters
{
    public class CustomAuthFilter : ActionFilterAttribute
    {
        public override void OnActionExecuting(ActionExecutingContext context)
        {
            var request = context.HttpContext.Request;

            if (!request.Headers.ContainsKey("Authorization"))
            {
                context.Result = new BadRequestObjectResult("Invalid request - No Auth token");
                return;
            }

            var authHeader = request.Headers["Authorization"].ToString();
            if (!authHeader.Contains("Bearer", StringComparison.OrdinalIgnoreCase))
            {
                context.Result = new BadRequestObjectResult("Invalid request - Token present but Bearer unavailable");
                return;
            }

            base.OnActionExecuting(context);
        }
    }
}

```

6. Applying Custom Auth Filter

6.1 Updating EmployeeController.cs

Final code:

```

using Microsoft.AspNetCore.Mvc;
using MyFirstWebAPI.Models;
using MyFirstWebAPI.Filters;

namespace MyFirstWebAPI.Controllers
{

```

```

[CustomAuthFilter]

[ApiController]
[Route("api/[controller]")]
public class EmployeeController : ControllerBase
{
    private static List<Employee> _employees = new List<Employee>();

    public EmployeeController()
    {
        if (_employees.Count == 0)
        {
            _employees = GetStandardEmployeeList();
        }
    }

    [HttpGet]
    [ProducesResponseType(typeof(List<Employee>), 200)]
    public ActionResult<List<Employee>> Get()
    {
        return Ok(_employees);
    }

    [HttpGet("{id}")]
    [ProducesResponseType(typeof(Employee), 200)]
    [ProducesResponseType(404)]
    public ActionResult<Employee> Get(int id)
    {
        var employee = _employees.FirstOrDefault(e => e.Id == id);
        if (employee == null)
        {
            return NotFound($"Employee with ID {id} not found");
        }
        return Ok(employee);
    }
}

```

```

[HttpPost]
[ProducesResponseType(typeof(Employee), 201)]
[ProducesResponseType(400)]
public ActionResult<Employee> Post([FromBody] Employee employee)
{
    if (employee == null)
        return BadRequest("Employee data is required");

    employee.Id = _employees.Count > 0 ? _employees.Max(e => e.Id) + 1 : 1;
    _employees.Add(employee);
    return CreatedAtAction(nameof(Get), new { id = employee.Id }, employee);
}

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 = "IT" },
            Skills = new List<Skill> { new Skill { Id = 1, Name = "C#" }, new Skill { Id = 2, Name = "ASP.NET" } },
            DateOfBirth = new DateTime(1990, 1, 1)
        },
        new Employee
        {
            Id = 2,
            Name = "Jane Smith",

```

```

        Salary = 60000,

        Permanent = false,

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

        Skills = new List<Skill> { new Skill { Id = 3, Name = "Management" }, new Skill { Id = 4, Name =
"Communication" } },

        DateOfBirth = new DateTime(1985, 5, 15)

    },

    new Employee

    {

        Id = 3,

        Name = "Bob Johnson",

        Salary = 55000,

        Permanent = true,

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

        Skills = new List<Skill> { new Skill { Id = 5, Name = "JavaScript" }, new Skill { Id = 6, Name = "React" } },

        DateOfBirth = new DateTime(1988, 12, 10)

    }

};

}

}

}

```

7. Testing Custom Auth Filter:

Once again building and running

```

dotnet build
dotnet run

```

```

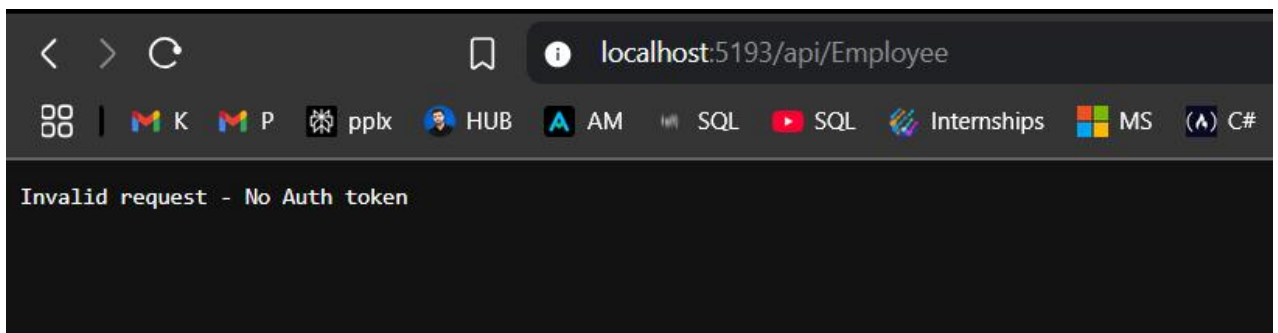
PS C:\Users\KIIT\OneDrive\Desktop\Web-API\MyFirstWebAPI> dotnet build
Restore complete (0.9s)
MyFirstWebAPI succeeded (5.6s) → bin\Debug\net9.0\MyFirstWebAPI.dll

Build succeeded in 7.6s
PS C:\Users\KIIT\OneDrive\Desktop\Web-API\MyFirstWebAPI> dotnet run
Using launch settings from C:\Users\KIIT\OneDrive\Desktop\Web-API\MyFirstWebAPI\Properties\launchSettings.json...
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5193
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\KIIT\OneDrive\Desktop\Web-API\MyFirstWebAPI

```

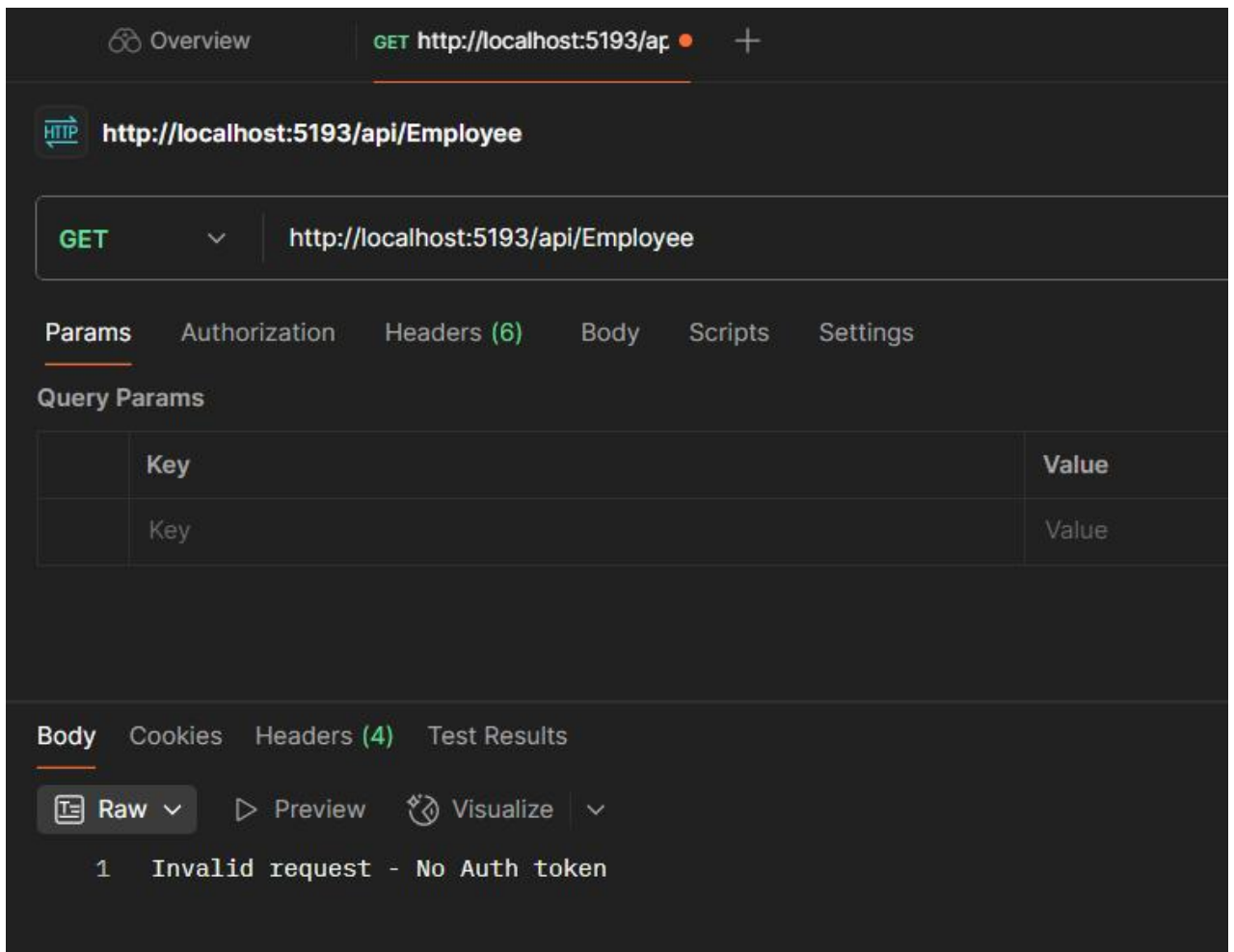
- **Without Authorization Header:**

- GET `http://localhost:5193/api/Employee`
- **Result:**

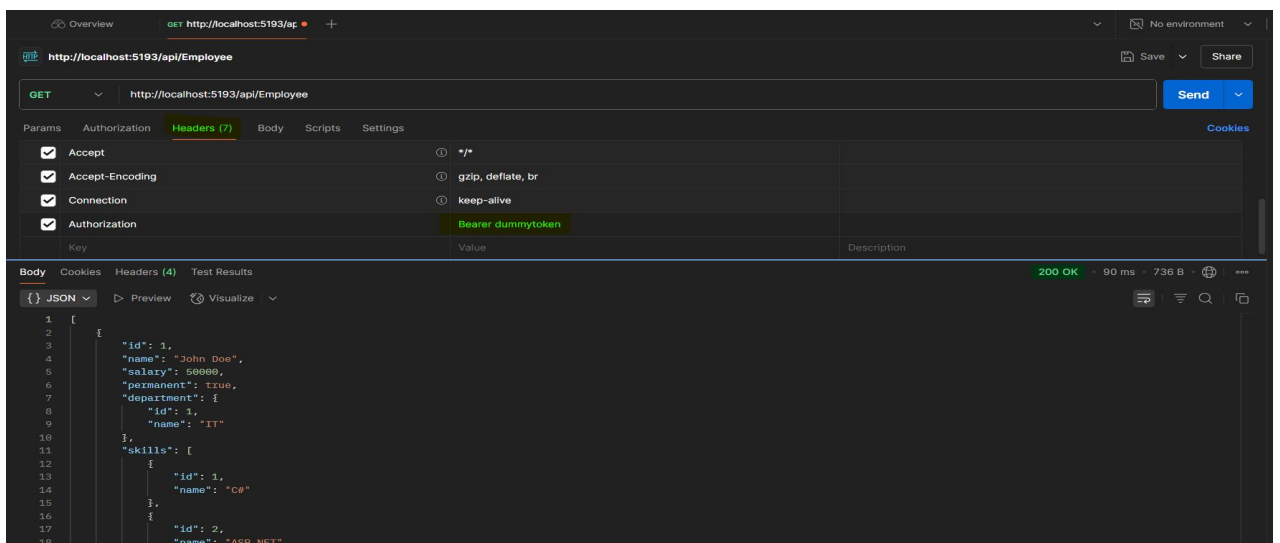


- **With Invalid Authorization Header in Postman:**

- Header: `Authorization: InvalidToken`
- **Result:**



- **With Valid Authorization Header:**
 - Header: Authorization: `Bearer dummytoken`
 - **Result:**



8. Creating Custom Exception Filter

8.1 Adding CustomExceptionFilter.cs

Creating Filters/CustomExceptionFilter.cs:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using System.Text.Json;

namespace MyFirstWebAPI.Filters
{
    public class CustomExceptionFilter : IExceptionHandler
    {
        public void OnException(ExceptionContext context)
        {
            var exception = context.Exception;

            var logEntry = new
            {
                TimeStamp = DateTime.Now,
                Message = exception.Message,
                StackTrace = exception.StackTrace,
                Source = exception.Source,
                InnerException = exception.InnerException?.Message
            };

            var logJson = JsonSerializer.Serialize(logEntry, new JsonSerializerOptions { WriteIndented = true });
            try
            {
                var logPath = Path.Combine(Directory.GetCurrentDirectory(), "exceptions.log");
                File.AppendAllText(logPath, logJson + Environment.NewLine + "---" + Environment.NewLine);
            }
            catch
            {
                // Ignore logging errors
            }

            context.Result = new ObjectResult(new
            {
                error = "An internal server error occurred",
                message = exception.Message,
                timestamp = DateTime.Now
            })
            {
                StatusCode = StatusCodes.Status500InternalServerError
            };
        }
    }
}
```

```

        StatusCode = 500
    };

    context.ExceptionHandled = true;
}
}
}

```

9. Registering Exception Filter

9.1 Updating Program.cs

- Adding at the top:

```
using MyFirstWebAPI.Filters;
```

- Replacing controller registration:

```
builder.Services.AddControllers(options =>
{
    options.Filters.Add<CustomExceptionFilter>();
});
```

Final Code of Program.cs :

```
using Microsoft.OpenApi.Models;
using MyFirstWebAPI.Filters;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers(options =>
{
    options.Filters.Add<CustomExceptionFilter>();
});

builder.Services.AddEndpointsApiExplorer();

// Add Swagger services with custom configuration

```

```

builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Swagger Demo",
        Version = "v1",
        Description = "Complete Web API Tutorial with .NET 9",
        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")
        }
    });
});

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");
        c.RoutePrefix = "swagger"; // Access swagger at /swagger
    });
}

```

```
});  
}  
  
app.UseHttpsRedirection();  
app.UseAuthorization();  
app.MapControllers();  
  
app.Run();
```

10. Testing Exception Filter

10.1 Adding Test Exception Endpoint

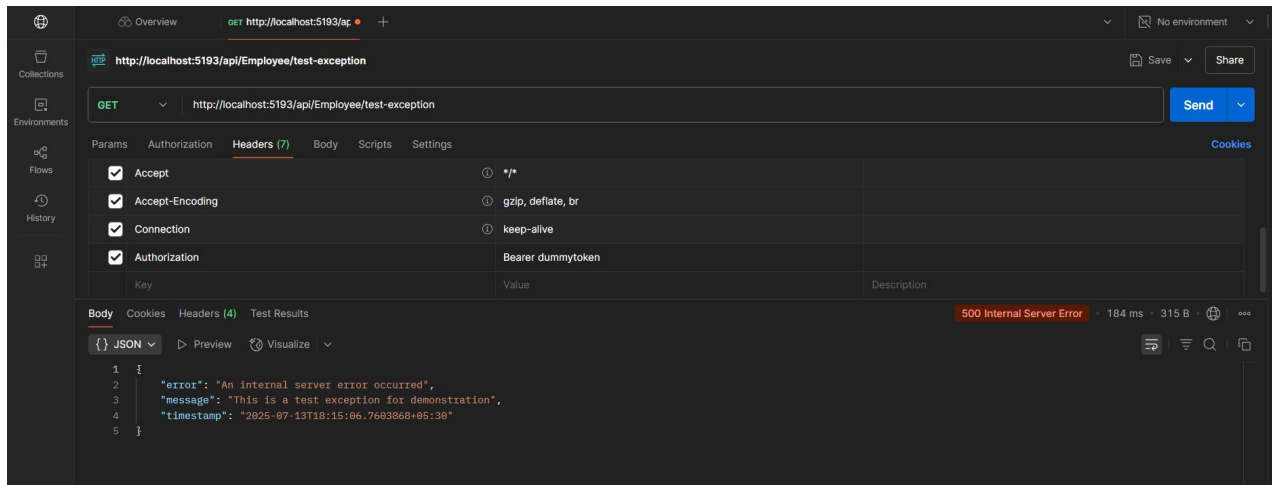
Add to EmployeeController.cs:

```
[HttpGet("test-exception")]  
[ProducesResponseType(500)]  
public ActionResult TestException()  
{  
    throw new Exception("This is a test exception for demonstration");  
}
```

10.2 Testing in Postman

1. GET <http://localhost:5193/api/Employee/test-exception>
2. Header: Authorization: Bearer dummytoken

3. Result:



Exercise 4: CRUD Operations

1. Implementing the PUT (Update) Method

1.1 Adding PUT Method to EmployeeController

Insert the following method after your POST method in EmployeeController.cs:

```
[HttpPut("{id}")]
[ProducesResponseType(typeof(Employee), 200)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public ActionResult<Employee> Put(int id, [FromBody] Employee employee)
{
    // Check if id is valid
    if (id <= 0)
        return BadRequest("Invalid employee id");

    // Check if employee data is provided
    if (employee == null)
        return BadRequest("Employee data is required");

    // Validate required fields
    if (string.IsNullOrEmpty(employee.Name))
        return BadRequest("Employee name is required");

    if (employee.Salary <= 0)
        return BadRequest("Employee salary must be greater than 0");
}
```

```

if (employee.Department == null || string.IsNullOrWhiteSpace(employee.Department.Name))
    return BadRequest("Employee department is required");

// Find existing employee
var existingEmployee = _employees.FirstOrDefault(e => e.Id == id);
if (existingEmployee == null)
    return BadRequest("Invalid employee id");

// Update employee data
existingEmployee.Name = employee.Name;
existingEmployee.Salary = employee.Salary;
existingEmployee.Permanent = employee.Permanent;
existingEmployee.Department = employee.Department;
existingEmployee.Skills = employee.Skills;
existingEmployee.DateOfBirth = employee.DateOfBirth;

return Ok(existingEmployee);
}

```

1.2 Testing the PUT Method

- Build and run your project:

```

dotnet build
dotnet run

```

- Test with Postman:
 - Method: PUT
 - URL: `https://localhost:7293/api/Employee/1`
 - Headers: Authorization: Bearer dummytoken
 - Body (JSON):

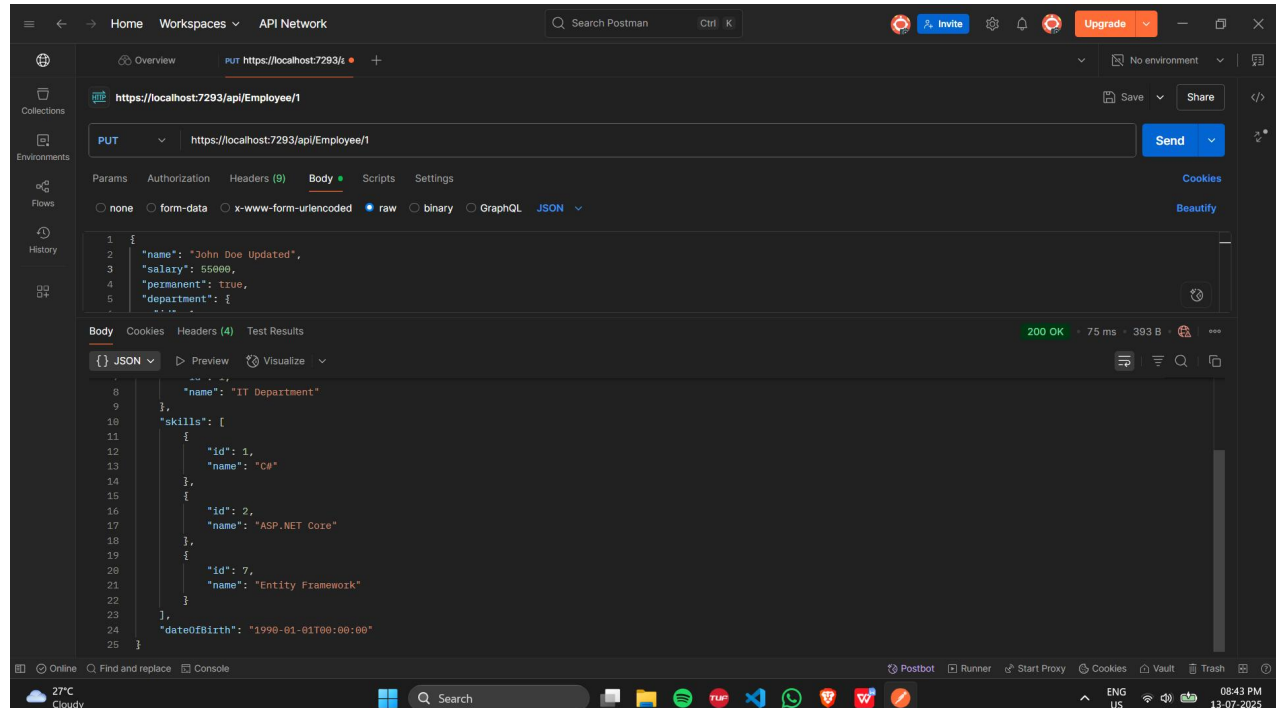
```

{
  "name": "John Doe Updated",
  "salary": 55000,
  "permanent": true,
  "department": { "id": 1, "name": "IT Department" },
  "skills": [
    { "id": 1, "name": "C#" },
    { "id": 2, "name": "ASP.NET Core" },
    { "id": 7, "name": "Entity Framework" }
  ],
}

```

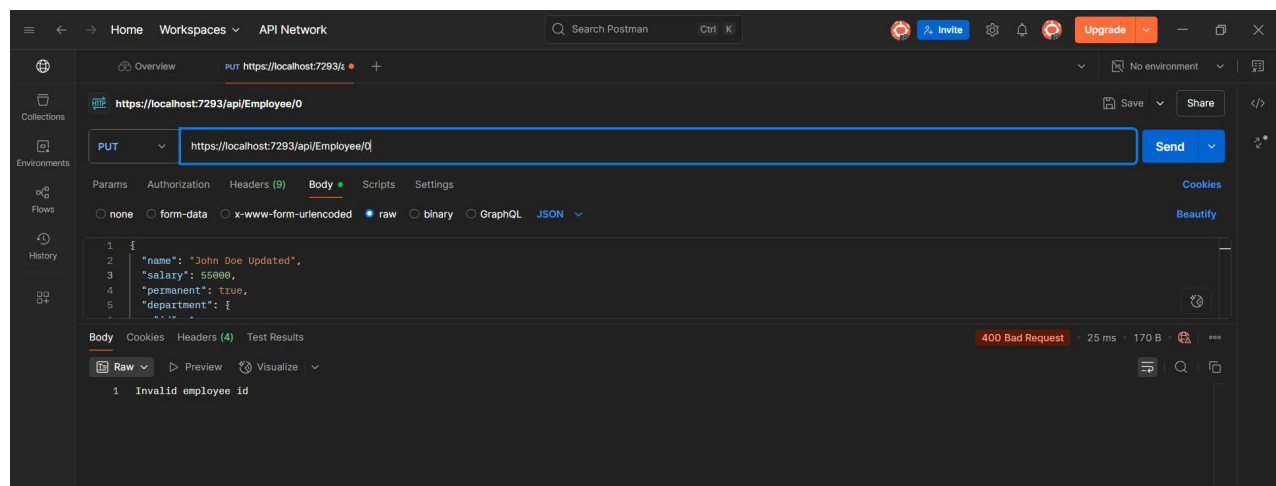
```
"dateOfBirth": "1990-01-01T00:00:00"
}
```

- Response:

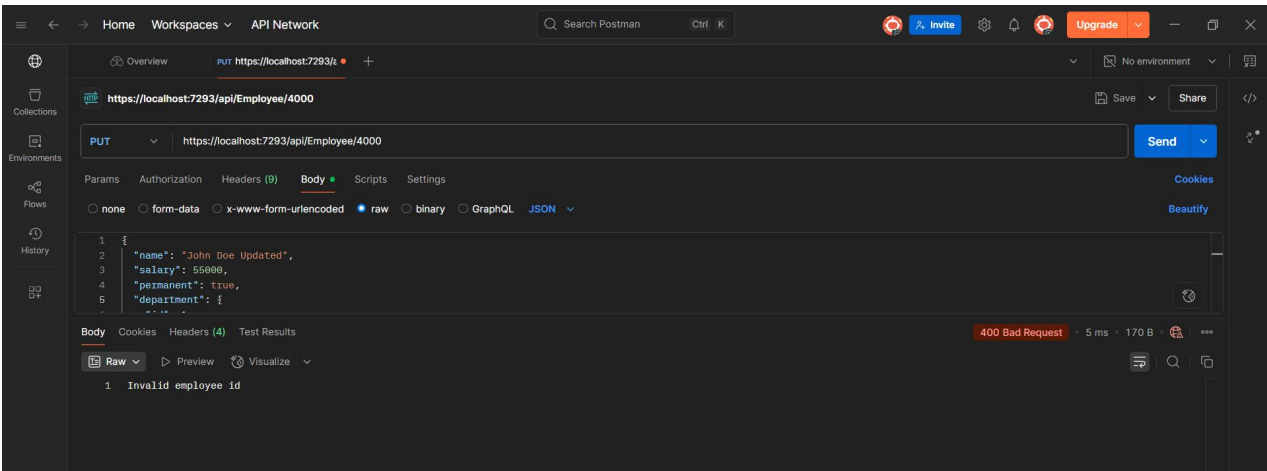


Error Cases Tested:

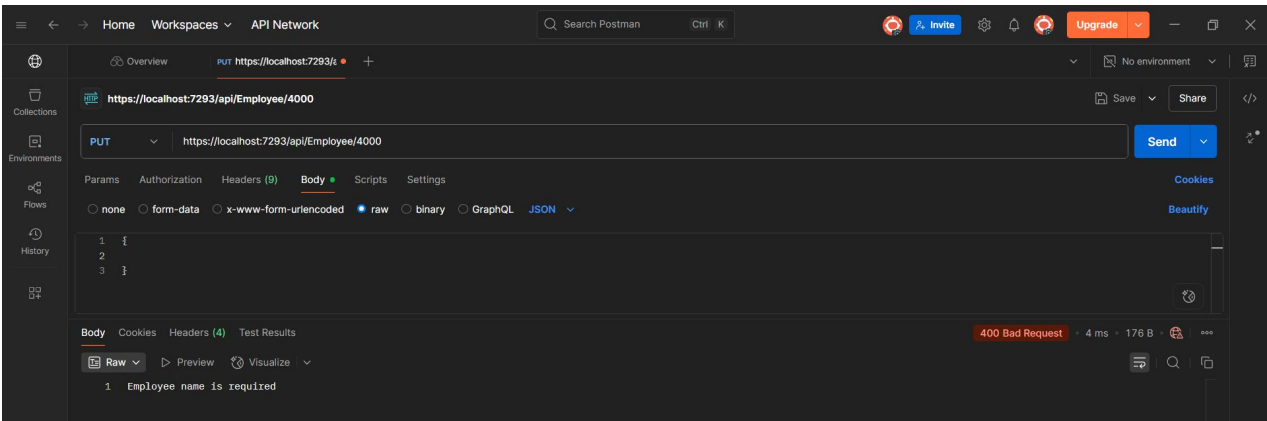
- Invalid ID (0 or negative):



- Non-existent ID:



- No body data:



2. Implementing the DELETE Method

2.1 Adding DELETE Method to EmployeeController

Adding this method after the PUT method:

```
[HttpDelete("{id}")]
[ProducesResponseType(200)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public ActionResult Delete(int id)
{
    // Check if id is valid
    if (id <= 0)
        return BadRequest("Invalid employee id");

    // Find existing employee
    var existingEmployee = _employees.FirstOrDefault(e => e.Id == id);
```

```

if (existingEmployee == null)
    return BadRequest("Invalid employee id");

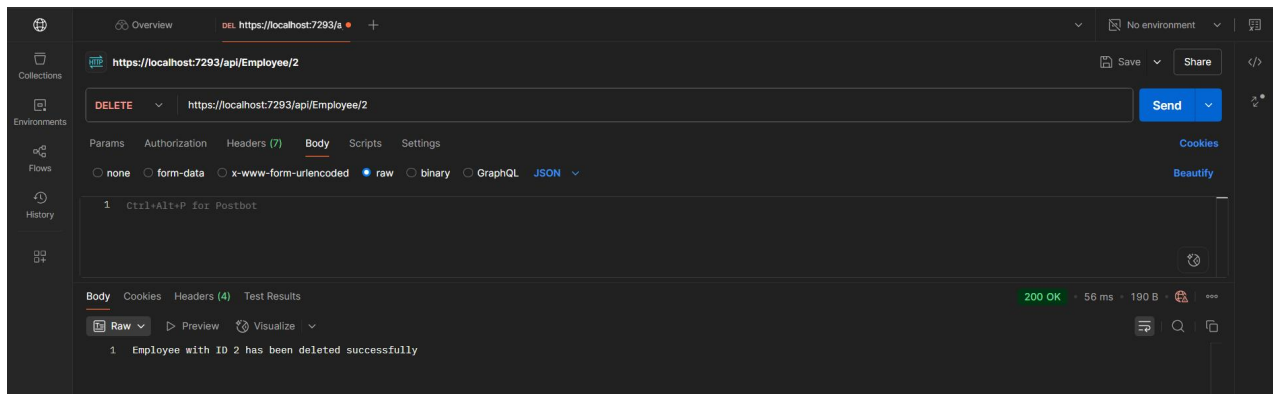
// Remove employee
_employees.Remove(existingEmployee);

return Ok($"Employee with ID {id} has been deleted successfully");
}

```

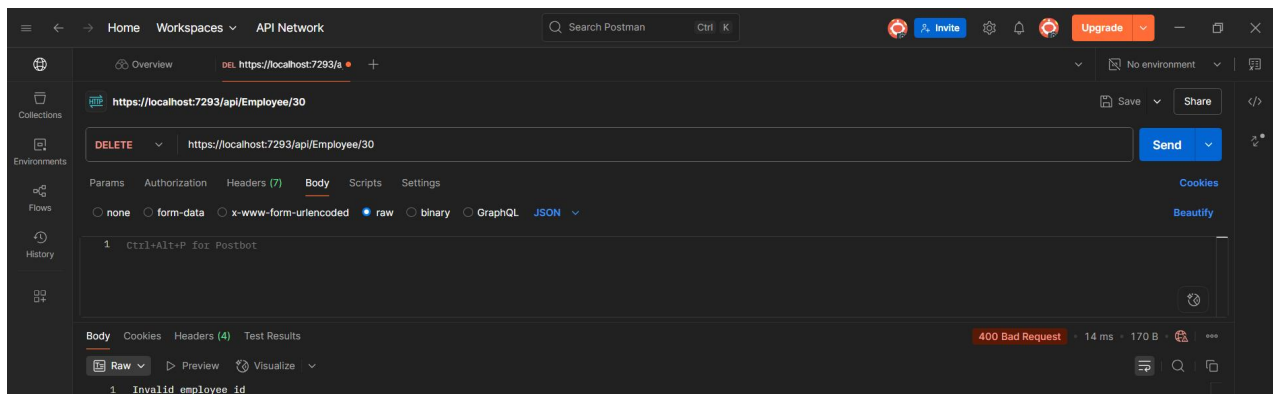
2.2 Testing the DELETE Method

- Method: DELETE
- URL: `https://localhost:7293/api/Employee/2`
- Headers: Authorization: Bearer dummytoken
- Response:

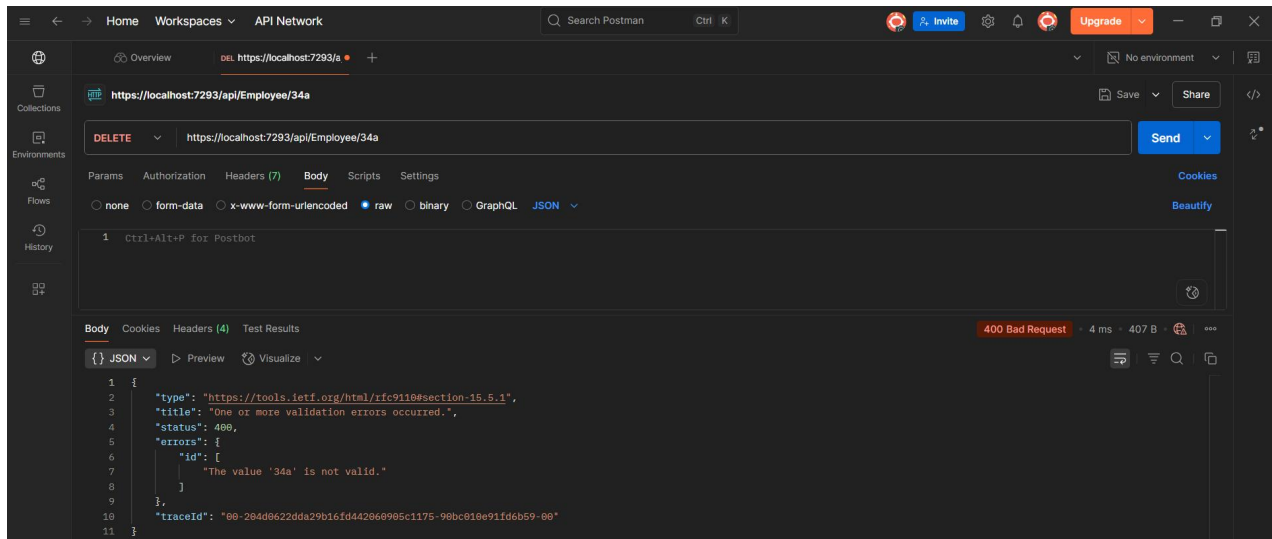


Error Cases Tested:

- Invalid ID:



- Non-existent ID:



3. Complete CRUD Testing

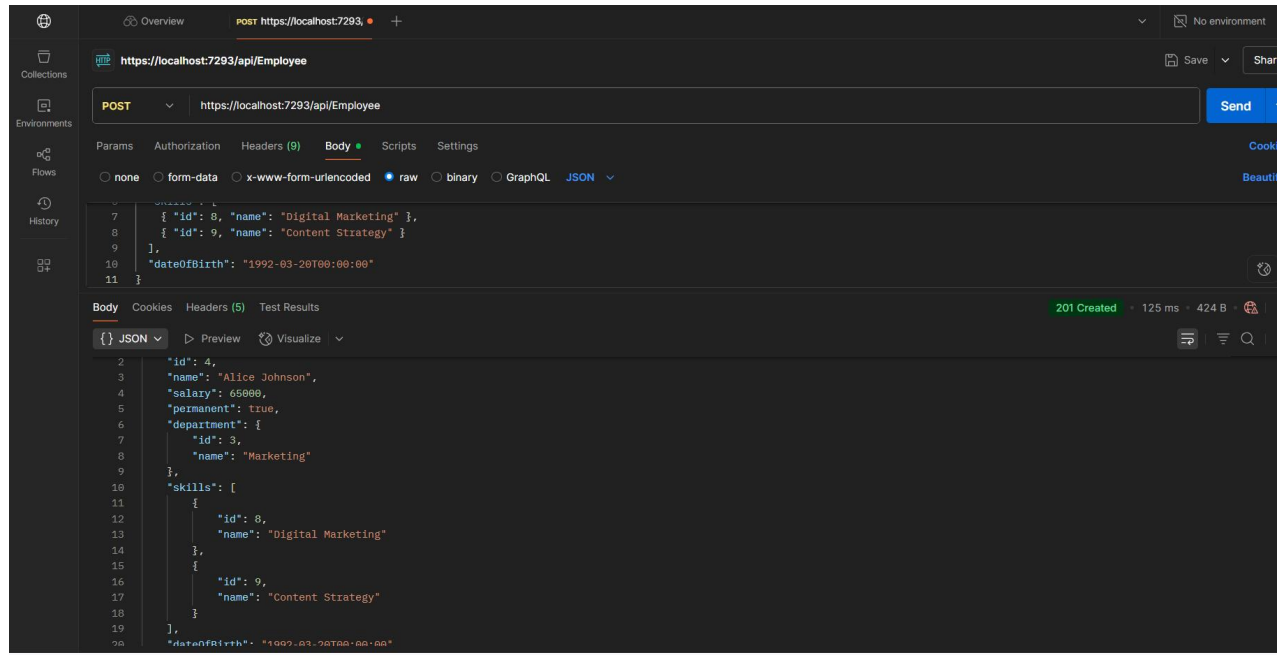
3.1 Test Full CRUD Flow

1. CREATE (POST)

- URL: https://localhost:7293/api/Employee
- Body:

```
{
  "name": "Alice Johnson",
  "salary": 65000,
  "permanent": true,
  "department": { "id": 3, "name": "Marketing" },
  "skills": [
    { "id": 8, "name": "Digital Marketing" },
    { "id": 9, "name": "Content Strategy" }
  ],
  "dateOfBirth": "1992-03-20T00:00:00"
}
```

RESULT:



2. READ (GET) – All employees

- URL: <https://localhost:7293/api/Employee>

RESULT:

```
[  {    "id": 1,    "name": "John Doe",    "salary": 50000,    "permanent": true,    "department": {      "id": 1,      "name": "IT"    },    "skills": [      {        "id": 1,        "name": "C#"      },      {        "id": 2,        "name": "ASP.NET"      }    ]  }
```

```
    ],
    "dateOfBirth": "1990-01-01T00:00:00"
  },
  {
    "id": 3,
    "name": "Bob Johnson",
    "salary": 55000,
    "permanent": true,
    "department": {
      "id": 1,
      "name": "IT"
    },
    "skills": [
      {
        "id": 5,
        "name": "JavaScript"
      },
      {
        "id": 6,
        "name": "React"
      }
    ],
    "dateOfBirth": "1988-12-10T00:00:00"
  },
  {
    "id": 4,
    "name": "Alice Johnson",
    "salary": 65000,
    "permanent": true,
    "department": {
      "id": 3,
      "name": "Marketing"
    },
    "skills": [
      {
        "id": 8,
        "name": "Digital Marketing"
      },
      {
```

```

        "id": 9,

        "name": "Content Strategy"

    }

],

    "dateOfBirth": "1992-03-20T00:00:00"

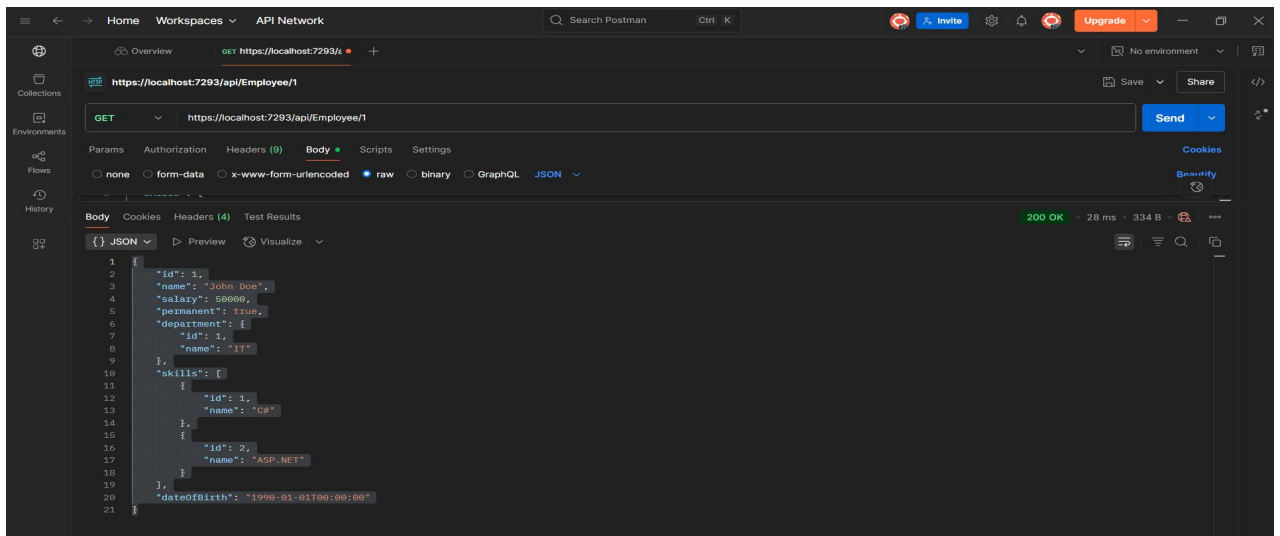
}
]

```

3. READ (GET) – Single employee

URL:

<https://localhost:7293/api/Employee/1>



4. UPDATE (PUT)

○ URL: <https://localhost:7293/api/Employee/4>

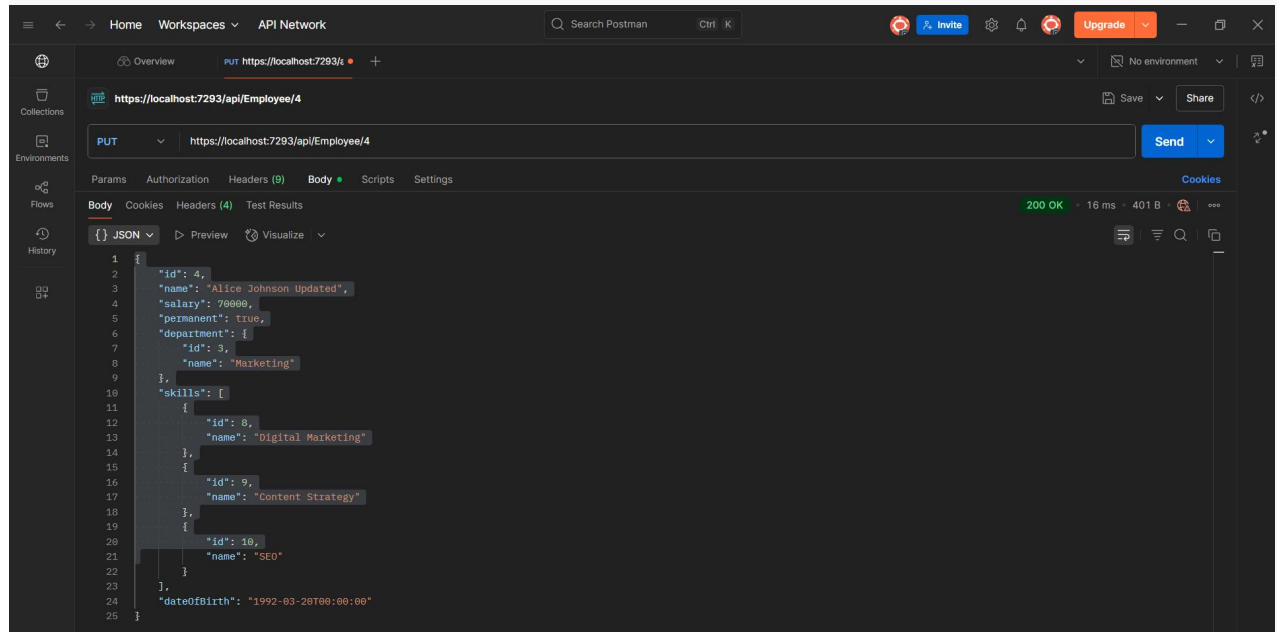
○ Body:

```

{
  "name": "Alice Johnson Updated",
  "salary": 70000,
  "permanent": true,
  "department": { "id": 3, "name": "Marketing" },
  "skills": [
    { "id": 8, "name": "Digital Marketing" },
    { "id": 9, "name": "Content Strategy" },
    { "id": 10, "name": "SEO" }
  ],
  "dateOfBirth": "1992-03-20T00:00:00"
}

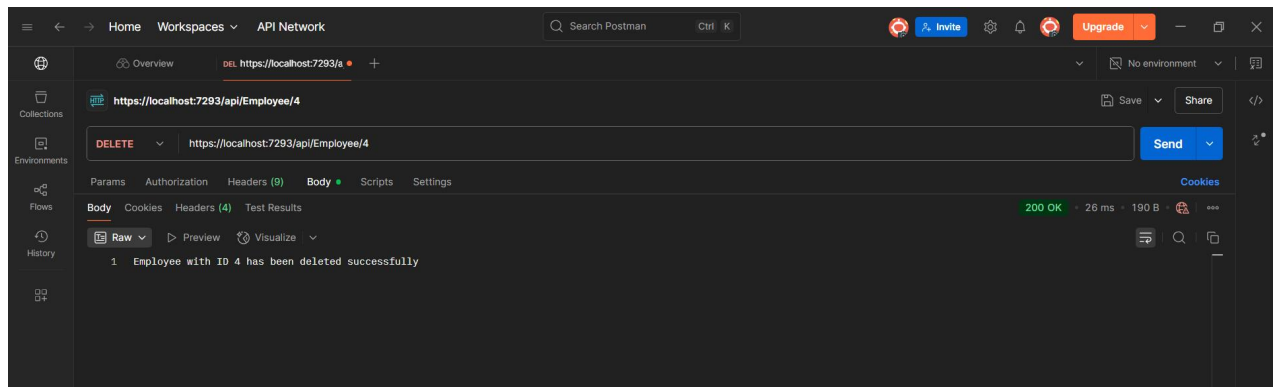
```

RESULT:



5. DELETE

URL: <https://localhost:7293/api/Employee/4>



Exercise 5: JWT Authentication

1. Installing Required Packages

Running the following commands in the project root to add JWT support:

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
dotnet add package System.IdentityModel.Tokens.Jwt
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell - MyFirstWebAPI + - [ ] [ ] ... ^ X
info : OK https://api.nuget.org/v3/vulnerabilities/index.json 286ms
info : GET https://api.nuget.org/v3-vulnerabilities/2025.07.10.23.28/vulnerability.base.json
info : GET https://api.nuget.org/v3-vulnerabilities/2025.07.10.23.28/2025.07.13.11.23.36/vulnerability.update.json
info : OK https://api.nuget.org/v3-vulnerabilities/2025.07.10.23.28/2025.07.13.11.23.36/vulnerability.update.json 288ms
info : OK https://api.nuget.org/v3-vulnerabilities/2025.07.10.23.28/vulnerability.base.json 290ms
info : Package 'Microsoft.AspNetCore.Authentication.JwtBearer' is compatible with all the specified frameworks in project 'C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\MyFirstWebAPI.csproj'.
info : PackageReference for package 'Microsoft.AspNetCore.Authentication.JwtBearer' version '9.0.7' added to file 'C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\MyFirstWebAPI.csproj'.
info : Writing assets file to disk. Path: C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\obj\project.assets.json
log : Restored C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\MyFirstWebAPI.csproj (in 5.04 sec).
PS C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI>
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell - MyFirstWebAPI + - [ ] [ ] ... ^ X
stractions\8.0.0 with content hash arDBqTgFCyS0EVRV703MzturChstw500J0y9bDJVAcMEPJm0FFPyJjU/3LVyStNGGey081DvnQYIncNXSJJGA==.
info : CACHE https://api.nuget.org/v3/vulnerabilities/index.json
info : CACHE https://api.nuget.org/v3-vulnerabilities/2025.07.10.23.28/vulnerability.base.json
info : CACHE https://api.nuget.org/v3-vulnerabilities/2025.07.10.23.28/2025.07.13.11.23.36/vulnerability.update.json
info : Package 'System.IdentityModel.Tokens.Jwt' is compatible with all the specified frameworks in project 'C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\MyFirstWebAPI.csproj'.
info : PackageReference for package 'System.IdentityModel.Tokens.Jwt' version '8.12.1' added to file 'C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\MyFirstWebAPI.csproj'.
info : Generating MSBuild file C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\obj\MyFirstWebAPI.csproj.nuget.g.targets.
info : Writing assets file to disk. Path: C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\obj\project.assets.json
log : Restored C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI\MyFirstWebAPI.csproj (in 5.1 sec).
PS C:\Users\KIIT\OneDrive\Desktop\Web-Api\MyFirstWebAPI>
```

2. Updating Program.cs for JWT Configuration

Final Program.cs:

```
using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using MyFirstWebAPI.Filters;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// JWT Configuration

string securityKey = "mysuperdupersecret_that_is_long_enough_for_security";

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

// Add Authentication

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,
        ClockSkew = TimeSpan.Zero
    };
});

// Add services to the container, including your custom exception filter
builder.Services.AddControllers(options =>
{
    options.Filters.Add<CustomExceptionFilter>();
});

builder.Services.AddEndpointsApiExplorer();

// Add Swagger services with custom configuration and JWT support
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Swagger Demo with JWT",
        Version = "v1",
        Description = "Complete Web API Tutorial with .NET 9 and JWT Authentication",
        TermsOfService = new Uri("https://example.com/terms"),
    });
});

```

```

Contact = new OpenApiContact
{
    Name = "Sachin Ray",
    Email = "sachin@xyzmail.com",
    Url = new Uri("https://www.example.com")
},
License = new OpenApiLicense
{
    Name = "License Terms",
    Url = new Uri("https://www.example.com")
}
});

// Add JWT Authentication to Swagger
c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
{
    Description = "JWT Authorization header using the Bearer scheme. Example: \"Authorization: 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();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo with JWT");
        c.RoutePrefix = "swagger"; // Access swagger at /swagger
    });
}

app.UseHttpsRedirection();

// Add Authentication and Authorization middleware (ORDER MATTERS!)
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();

```

3. Adding JWT Models

Appending these classes to Models/Employee.cs:

```
public class LoginRequest
{
    public string Username { get; set; } = string.Empty;
    public string Password { get; set; } = string.Empty;
}

public class LoginResponse
{
    public string Token { get; set; } = string.Empty;
    public string Username { get; set; } = string.Empty;
    public string Role { get; set; } = string.Empty;
    public DateTime ExpiresAt { get; set; }
}
```

4. Creating AuthController

Creating Controllers/AuthController.cs:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using MyFirstWebAPI.Models;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace MyFirstWebAPI.Controllers
{
    [AllowAnonymous]
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        [HttpGet("generate-token")]
        public ActionResult<LoginResponse> GenerateToken()
        {
            var token = GenerateJSONWebToken(1, "Admin");
            return Ok(new LoginResponse
            {
                Token = token,
                Username = "admin",
                Role = "Admin",
            });
        }
    }
}
```

```

        ExpiresAt = DateTime.Now.AddMinutes(10)
    });
}

[HttpGet("generate-token-short")]
public ActionResult<LoginResponse> GenerateTokenShort()
{
    var token = GenerateJSONWebToken(1, "Admin", 2);
    return Ok(new LoginResponse
    {
        Token = token,
        Username = "admin",
        Role = "Admin",
        ExpiresAt = DateTime.Now.AddMinutes(2)
    });
}

[HttpPost("login")]
public ActionResult<LoginResponse> Login([FromBody] LoginRequest request)
{
    if (request.Username == "admin" && request.Password == "admin123")
    {
        var token = GenerateJSONWebToken(1, "Admin");
        return Ok(new LoginResponse
        {
            Token = token,
            Username = "admin",
            Role = "Admin",
            ExpiresAt = DateTime.Now.AddMinutes(10)
        });
    }
    return Unauthorized("Invalid username or password");
}

[HttpGet("generate-token-poc")]
public ActionResult<LoginResponse> GenerateTokenPOC()
{
    var token = GenerateJSONWebToken(2, "POC");
    return Ok(new LoginResponse
    {
        Token = token,
        Username = "poc_user",
        Role = "POC",
        ExpiresAt = DateTime.Now.AddMinutes(10)
    });
}

```

```

    }

    private string GenerateJSONWebToken(int userId, string userRole, int expirationMinutes = 10)
    {
        var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret_that_is_long_enough_for_security"));
        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(expirationMinutes),
            signingCredentials: credentials
        );

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

```

5. Update EmployeeController for Authorization

Replace [CustomAuthFilter] with [Authorize]:

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using MyFirstWebAPI.Models;

namespace MyFirstWebAPI.Controllers
{
    [Authorize]
    [ApiController]
    [Route("api/[controller]")]
    public class EmployeeController : ControllerBase
    {
        // ... existing methods
    }
}

```

```
}
```

For role-based access, use:

```
[Authorize(Roles = "POC")] // Only POC role
```

```
// or
```

```
[Authorize(Roles = "Admin,POC")] // Both Admin and POC
```

6. Testings Done:

- **Without Token:**

```
GET /api/Employee
```

Result:

The screenshot shows the Postman interface with a GET request to `https://localhost:7293/api/Employee`. The Headers tab is active, displaying the following headers:

Key	Value	Description
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.44.1	
<input checked="" type="checkbox"/> Accept	*/*	
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection	keep-alive	
<input type="checkbox"/> Authorization	Bearer dummytoken	

The response status is **401 Unauthorized** with a response time of 39 ms and a body size of 128 B. The response body is shown in the Raw tab.

- **Get Token:**

```
GET /api/Auth/generate-token
```

Using the received token for subsequent requests.

Result:

The screenshot shows a REST client interface with the following details:

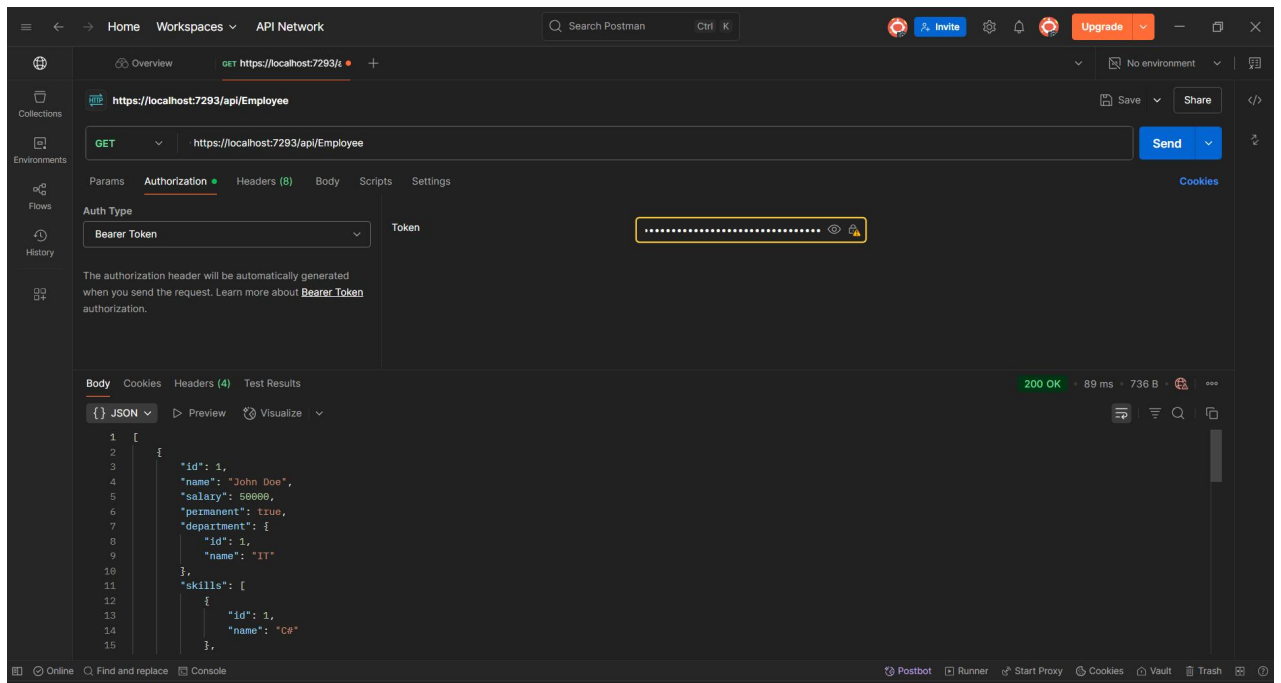
- URL:** `https://localhost:7293/api/Auth/generate-token`
- Method:** `GET`
- Status:** `200 OK` (167 ms, 503 B)
- Response Body (JSON):**

```
{  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy5taWNYb3NvZnQuY29tL3dzLzIwMDgvdjYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJBZG1pb1IsIlVzZXJJZCI6IjEiLCJleHAiOiJE3NTI0MjU5ODAsIm1zcyI6Im15U3lzdGVtIiwiaXVkiOiJoibXlvc2VycyJ9.VL1qPawYjQK10CD_4RnOT4TU9pCgu2DIdJKFXDlK-Rs",  "username": "admin",  "role": "Admin",  "expiresAt": "2025-07-13T22:29:40.4337545+05:30"}
```

- **With Token:**

`GET /api/Employee`

RESULT:

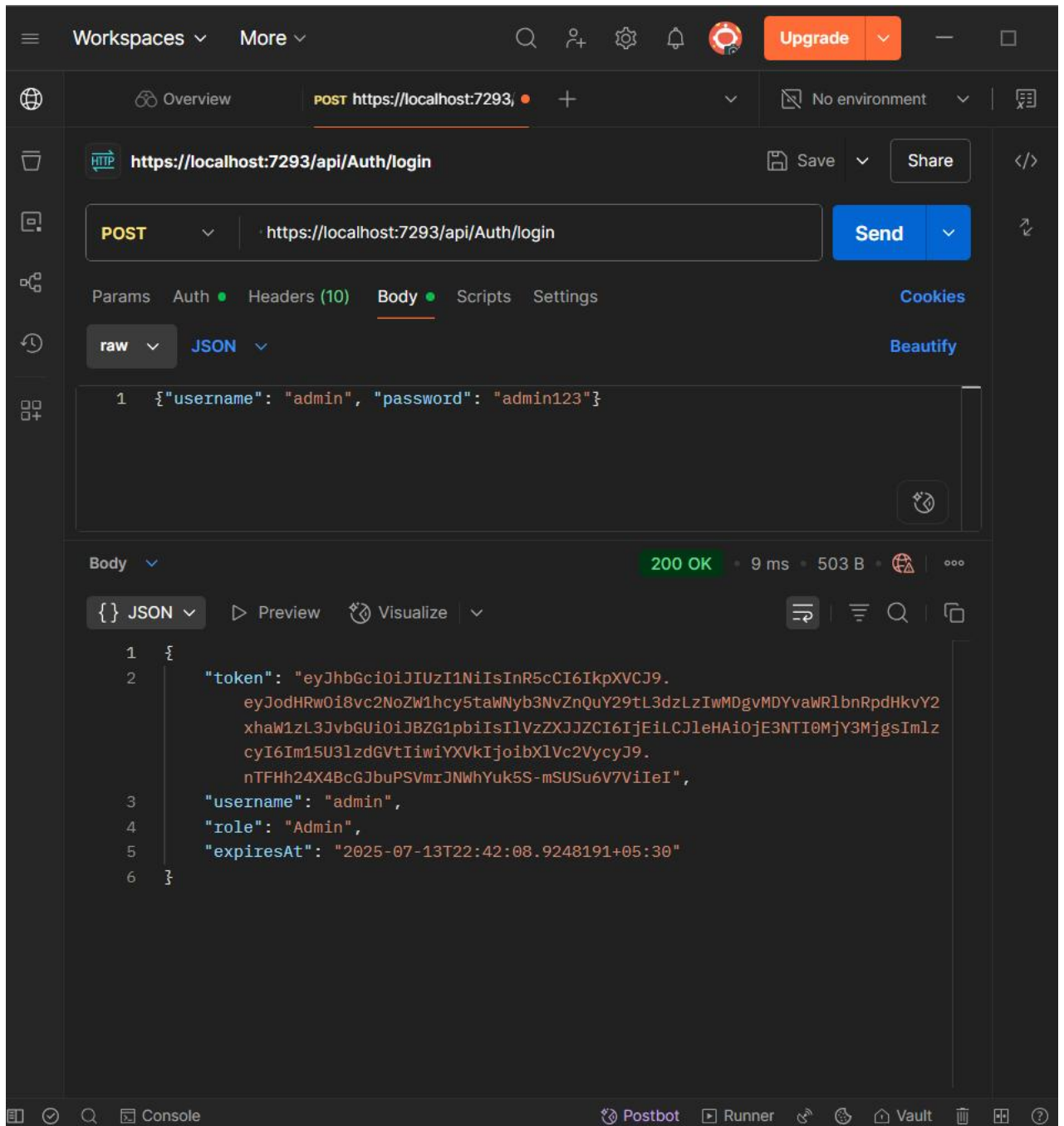


- **Login:**

POST `/api/Auth/login`

Body: `{"username": "admin", "password": "admin123"}`

RESULT:



- **Token Expiration:**

GET /api/Auth/generate-token-short

RESULT: Waited 2+ minutes, then tried `/api/Employee` with expired token. The result is here

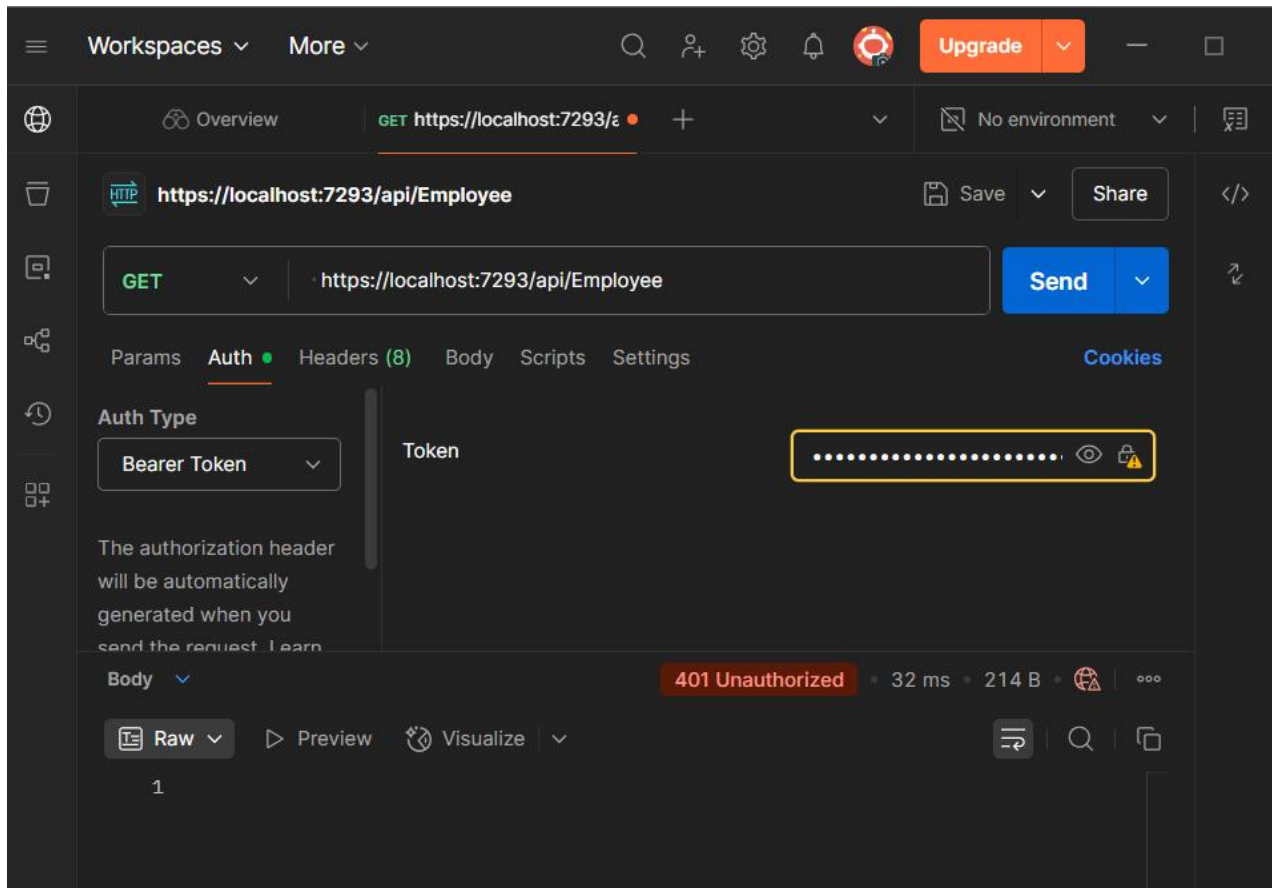
The screenshot displays the Postman interface for a REST client. The top bar shows 'Workspaces' and 'More' menus, along with search, user, settings, and notification icons. An 'Upgrade' button is visible on the right. The main interface is divided into several sections:

- Overview:** Shows the current request URL: `https://localhost:7293/`.
- Request:** The method is `GET` and the URL is `https://localhost:7293/api/Auth/generate-token-short`. A 'Send' button is present.
- Auth:** The 'Auth Type' is set to 'Bearer Token'. A 'Token' field is visible, containing a long alphanumeric string (partially obscured by a yellow box). Below this, a note states: 'The authorization header will be automatically generated when you send the request. Learn more.'
- Response:** The status is `200 OK` with a response time of `4 ms` and a body size of `503 B`. The response is displayed in JSON format:

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy5taWwNyY3NvZnQuY29tL3dzLzIwMDgvMDYvawRlbnRpdHkvY2xhaW1zL3JvbGUiOiJBZG1pb2IiLCJleHAiOiJlcyI6Im15U3lzdGVtIiwiaXVkiOiBjbXV2VycyJ9.L_gzY5zBQMkzUCGgBndgh5rjTKmPweKAwzQRAicdV1w",
3   "username": "admin",
4   "role": "Admin",
5   "expiresAt": "2025-07-13T22:43:53.0217396+05:30"
6 }
```

The bottom status bar includes icons for 'Console', 'Postbot', 'Runner', 'Vault', and other utility functions.

After 2 minutes:



- **Role-Based Authorization:**

- Changed [Authorize(Roles = "POC")] in EmployeeController

```
166 // }
167 // }
168
169 //Exercise-5
170
171 using Microsoft.AspNetCore.Authorization;
172 using Microsoft.AspNetCore.Mvc;
173 using MyFirstWebAPI.Models;
174 // using MyFirstWebAPI.Filters; // No longer needed for JWT
175
176 namespace MyFirstWebAPI.Controllers
177 {
178
179     // [Authorize] // Any authenticated user
180
181     [Authorize(Roles = "POC")] // Only users with "POC" role
182
183     // [Authorize(Roles = "Admin,POC")] // Users with "Admin" OR "POC" roles
184
185
186     [ApiController]
187     [Route("api/[controller]")]
188     public class EmployeeController : ControllerBase
```

- Use /api/Auth/generate-token-poc for a POC token

Result:

- POC-Token:

The screenshot displays a web client interface with a dark theme. At the top, there are tabs for 'Workspaces' and 'More'. The main header shows the URL 'https://localhost:7293/api/Auth/generate-token-poc' with a 'GET' method. Below this, the 'Auth' tab is selected, showing 'No Auth' as the authentication type. The 'Send' button is visible. The response section shows a '200 OK' status with a response time of 210 ms and a body size of 502 B. The response body is displayed in JSON format, containing a long token string, a username of 'poc_user', a role of 'POC', and an expiration time.

Workspaces ▾ More ▾

Overview GET https://localhost:7293/api/Auth/generate-token-poc

Save ▾ Share

GET https://localhost:7293/api/Auth/generate-token-poc Send ▾

Params Auth Headers (7) Body Scripts Settings Cookies

Auth Type

No Auth ▾

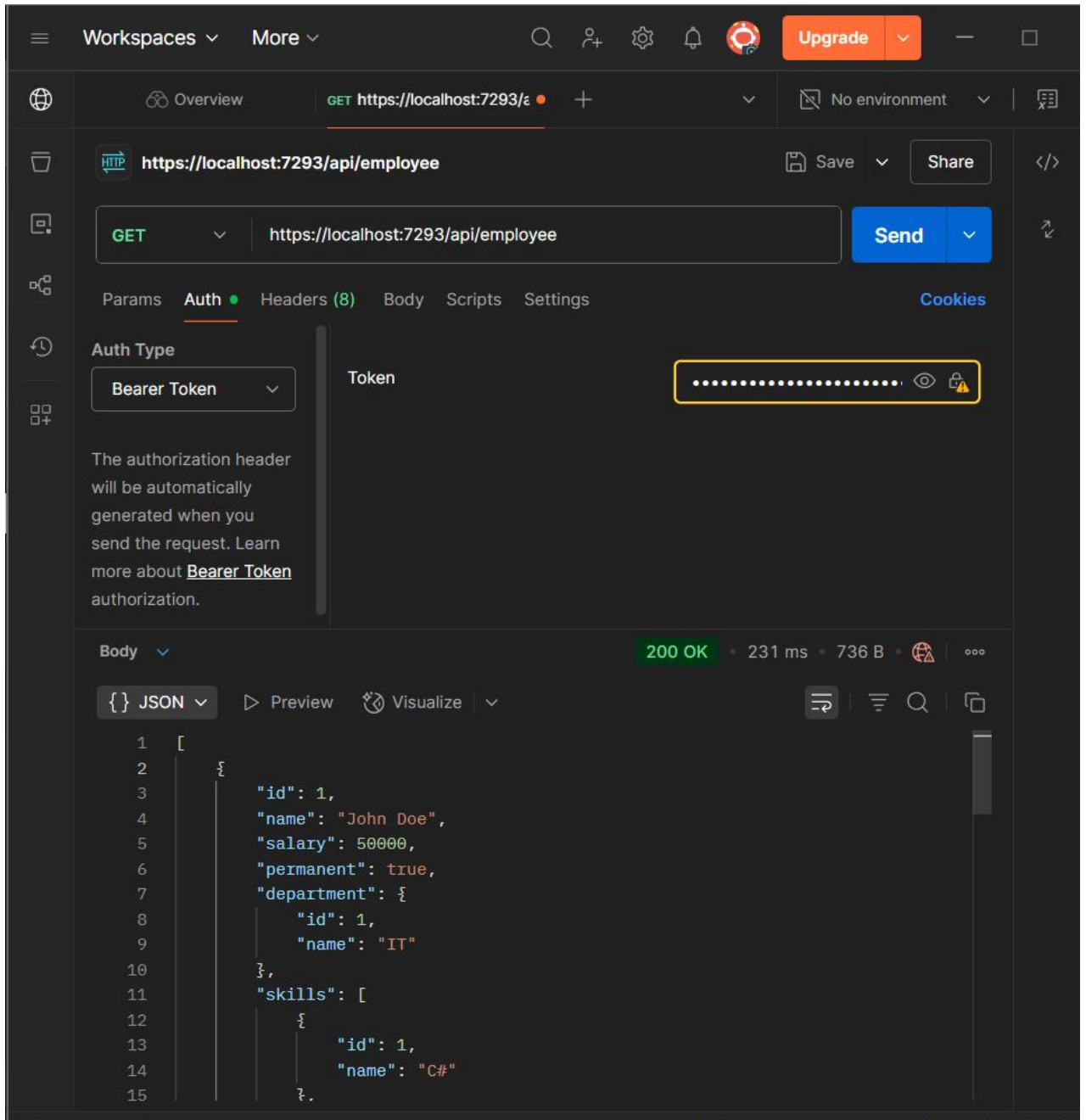
No Auth

This request does not use any authorization. ⓘ

Body ▾ 200 OK • 210 ms • 502 B • 🌐 ⋮

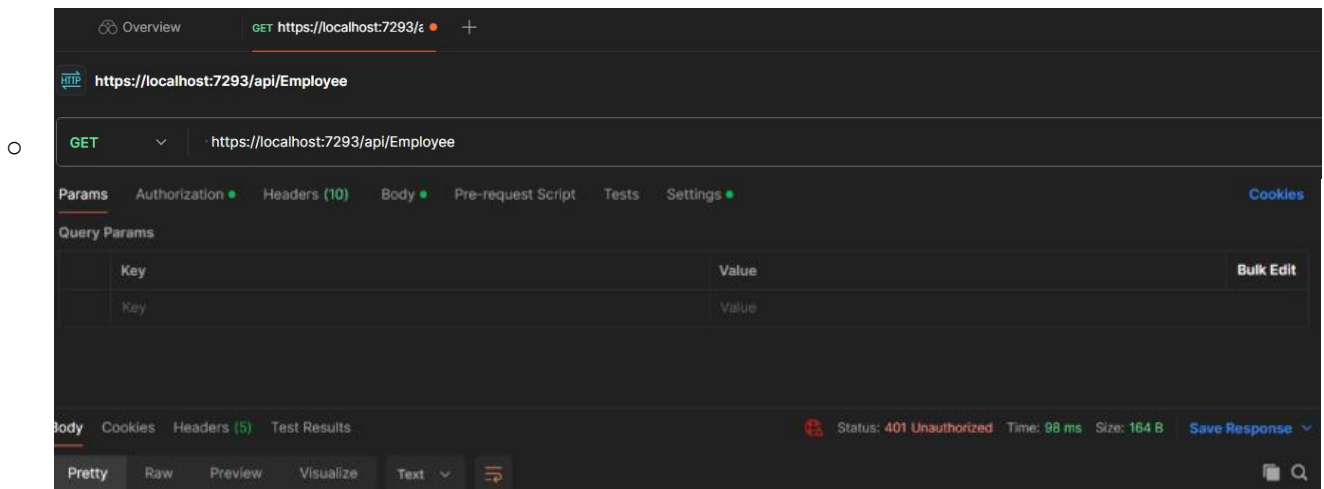
{ } JSON ▾ ▶ Preview 🔄 Visualize ▾

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy5taWw3NjZnZnQuY29tL3dzLzIwMDgvdjYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJQT0MiLCJvc2VySWQiOiIyIiwiaXNjaXhwIjozNzUyNDI4MTQ5LCJpc3MiOiJ0JteVN5c3RlbnR5ImF1ZCI6Im15VXNlcnMifQ.usNf4tLeyrFgMMSXx8iDM7KruGI5dI8RecubG3yCGRM",
3   "username": "poc_user",
4   "role": "POC",
5   "expiresAt": "2025-07-13T23:05:49.6385411+05:30"
6 }
```



- **Admin token:**

It gives 401 Unauthorized error.



Points Found

- **JWT tokens** expire in 10 minutes (2 minutes for short token)
- Used Bearer [token] in the Authorization header generated automatically by keeping bearer token with token in the box.
- Role-based access via [Authorize(Roles = "RoleName")]
- The **security key** must be identical in both Program.cs and AuthController.