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

Create a .Net core web application with API template. Use the option to create controller with Read Write permissions. Notice the ValuesController creation with Action methods corresponding to the Action verbs.

On creation of the Web API, execute the application and check if the GET action method result is returned as expected.

ValuesController.cs

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace FirstWebAPI.Controllers

{

[ApiController]

[Route("[controller]")]

public class ValuesController : ControllerBase

{

// Static list to hold values in memory

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

// GET /values

[HttpGet]

public IActionResult Get()

{

return Ok(values);

}

// GET /values/{id}

[HttpGet("{id}")]

public IActionResult Get(int id)

{

if (id < 0 || id >= values.Count)

return NotFound();

return Ok(values[id]);

}

// POST /values

[HttpPost]

public IActionResult Post([FromBody] string value)

{

if (string.IsNullOrWhiteSpace(value))

return BadRequest("Value cannot be empty");

values.Add(value);

return Ok(values);

}

// PUT /values/{id}

[HttpPut("{id}")]

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

{

if (id < 0 || id >= values.Count)

return BadRequest("Invalid index");

values[id] = value;

return Ok(values);

}

// DELETE /values/{id}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

if (id < 0 || id >= values.Count)

return BadRequest("Invalid index");

values.RemoveAt(id);

return Ok(values);

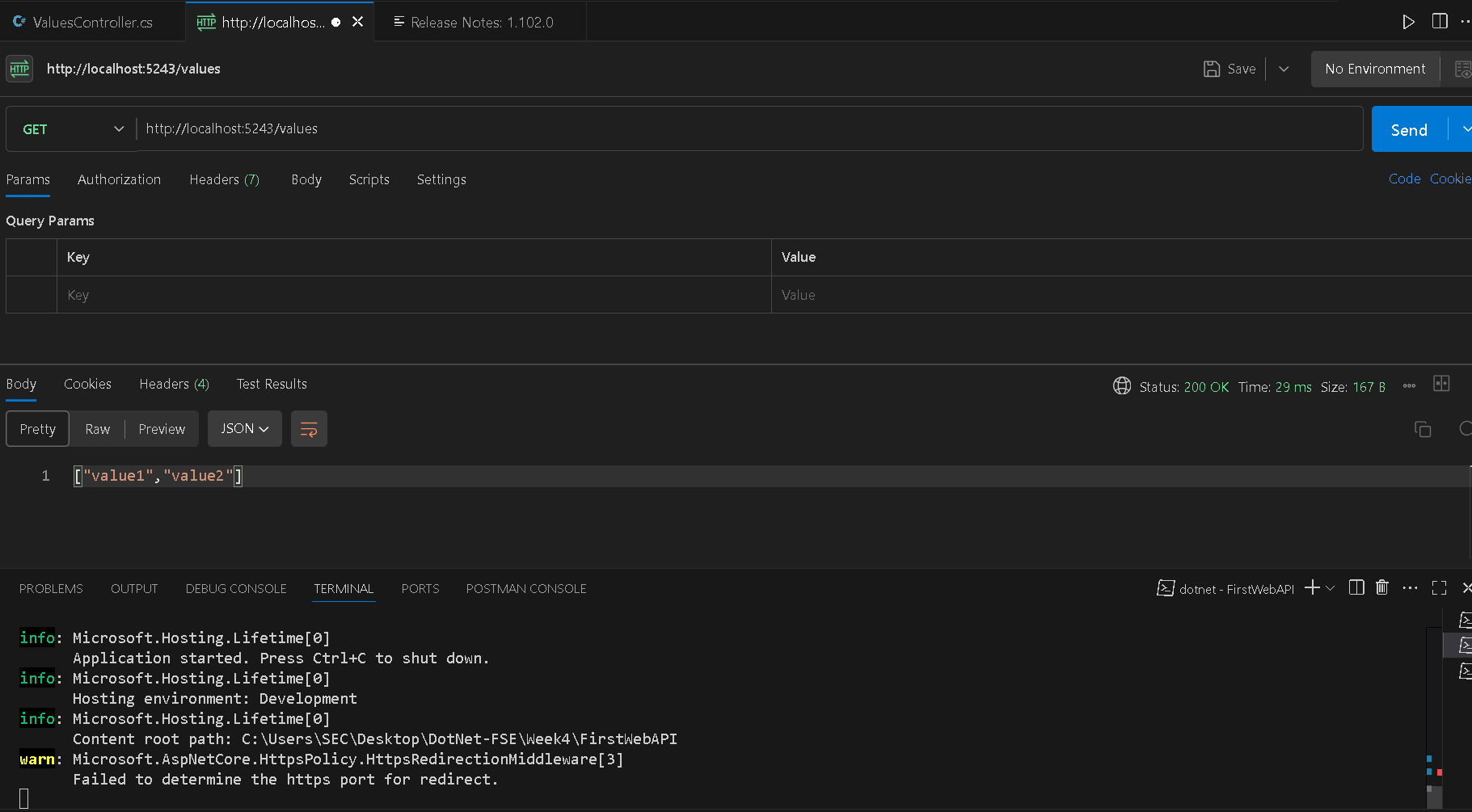
}

}

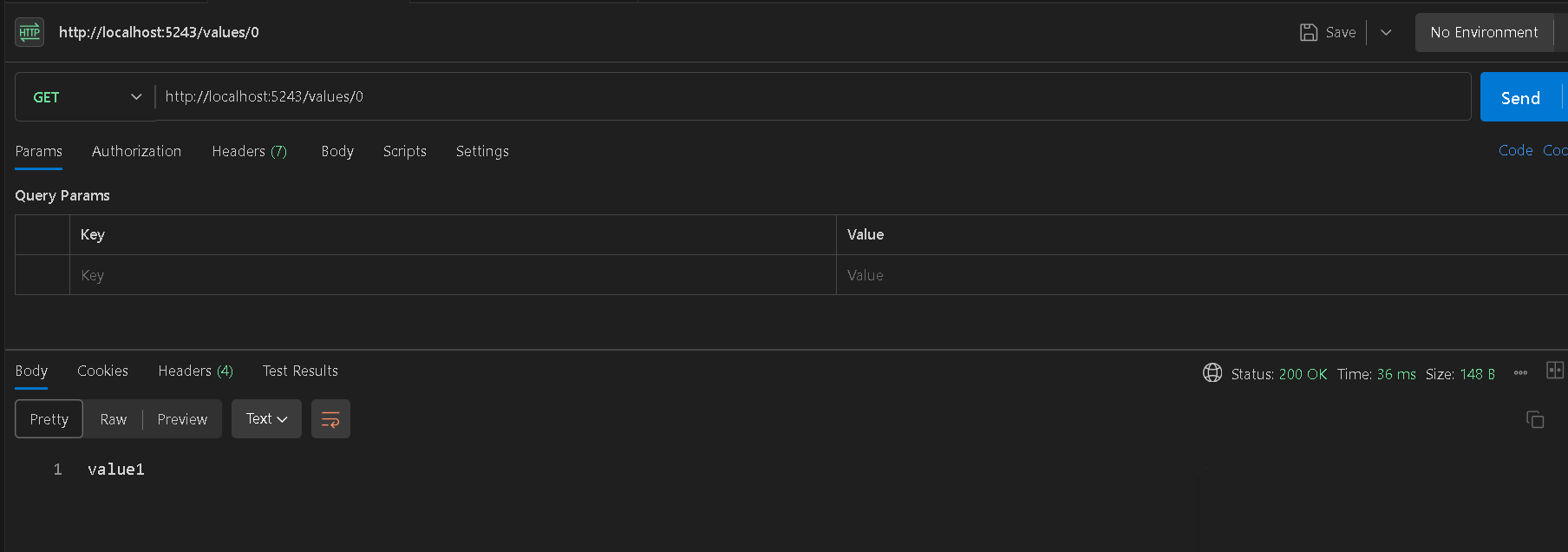
}

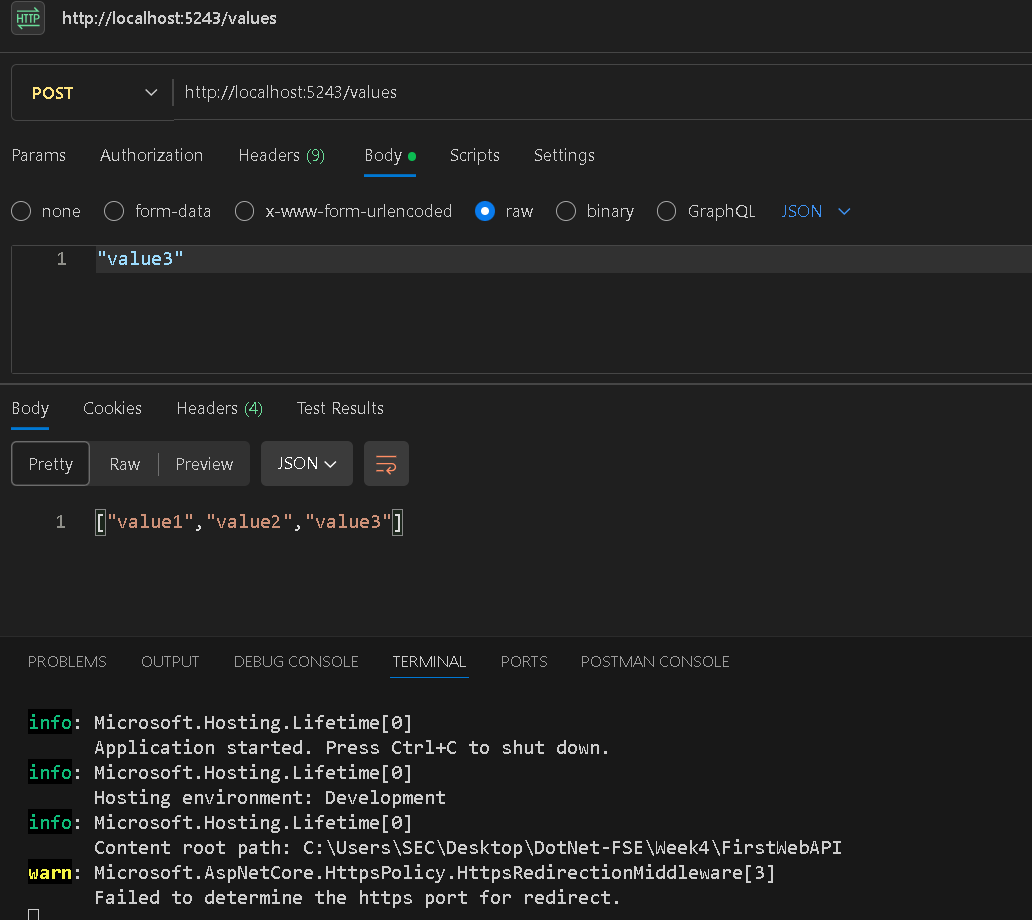
OUTPUT:

GET /values

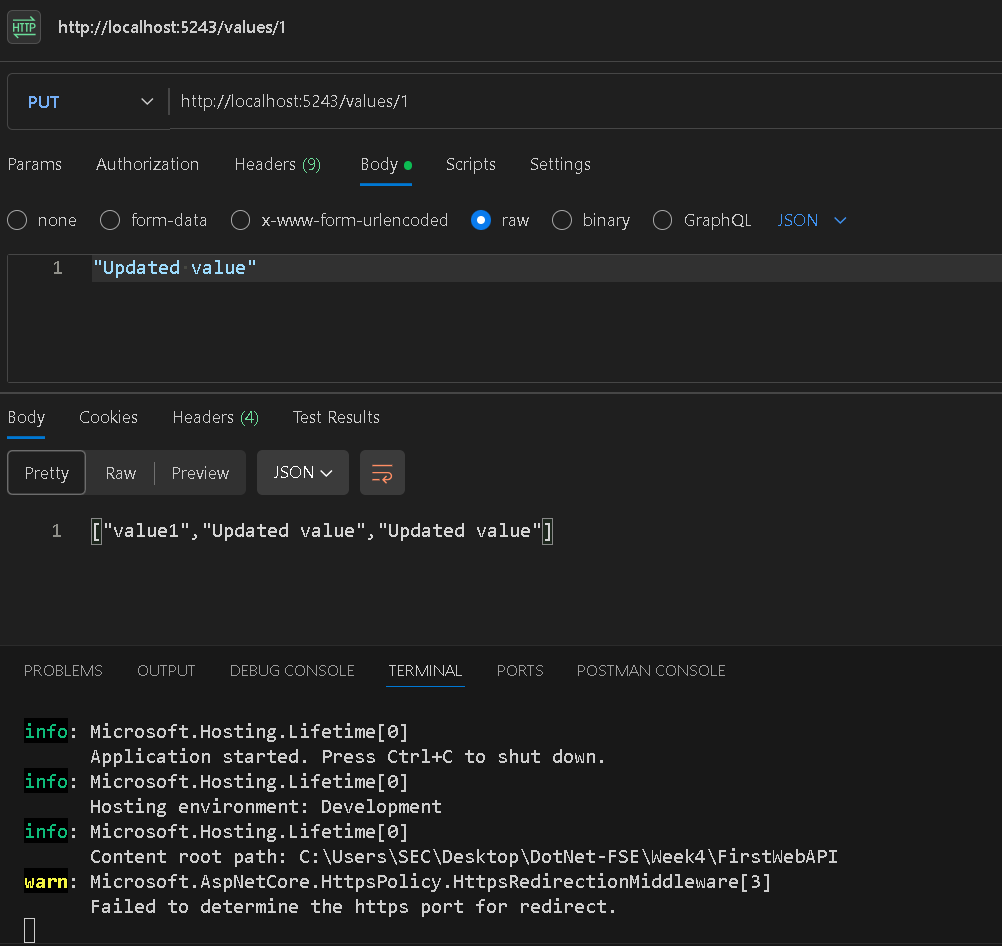


GET /values/{id}

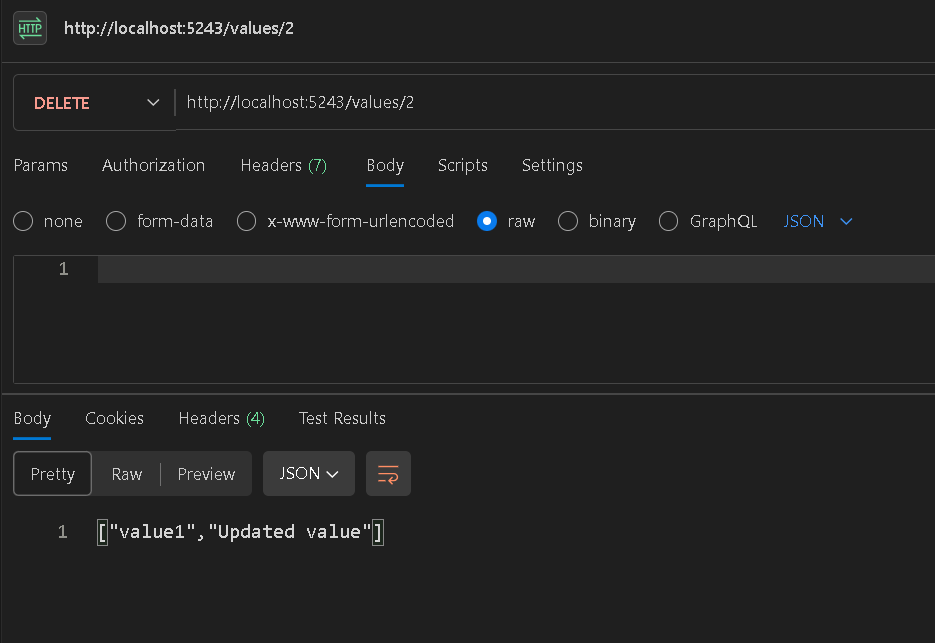


POST

PUT



DELETE



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](about:blank) 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).

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

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://example.com/terms"),

Contact = new OpenApiContact

{

Name = "John Doe",

Email = "john@xyzmail.com",

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

},

License = new OpenApiLicense

{

Name = "License Terms",

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

}

});

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

}

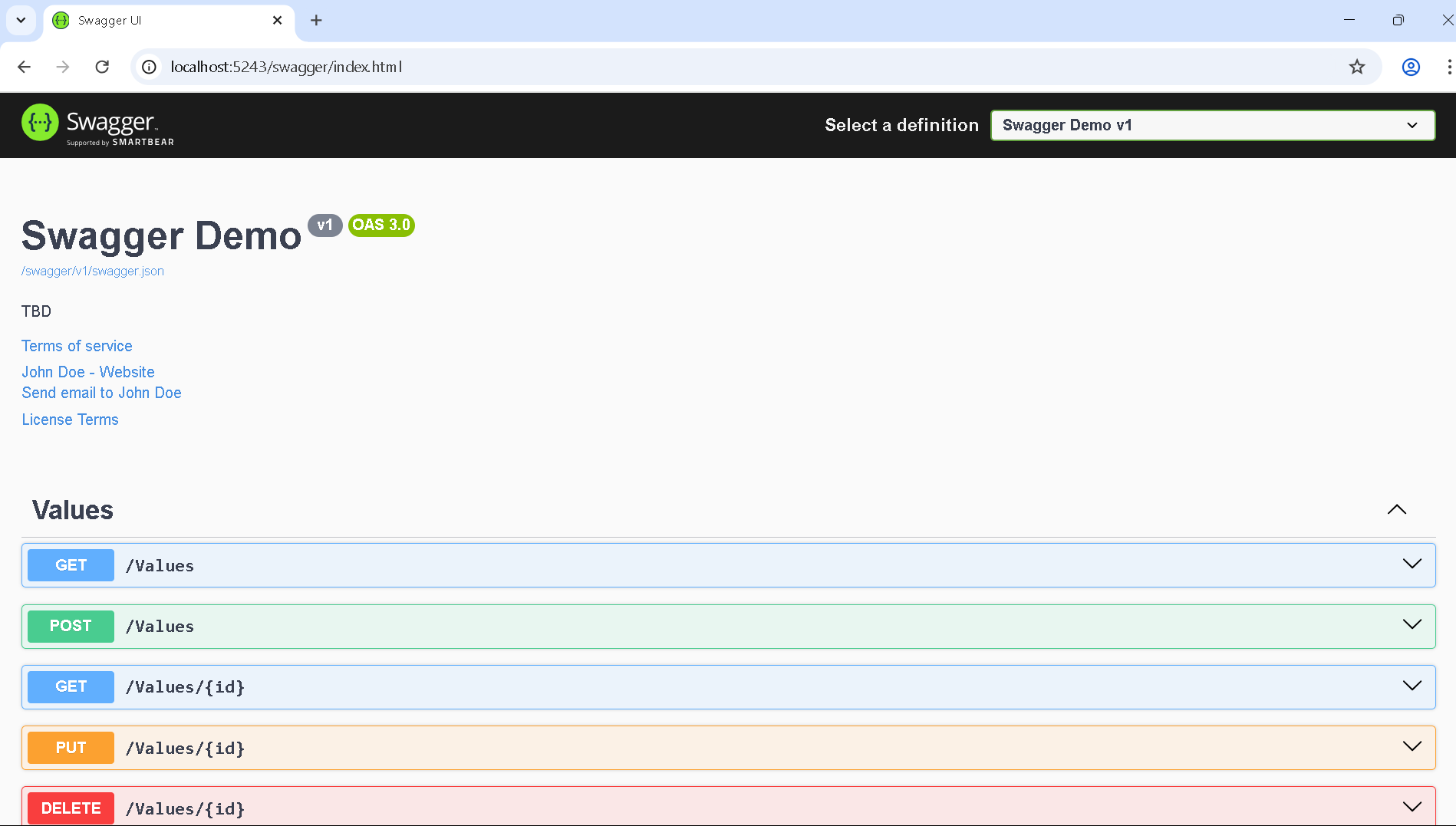
app.UseHttpsRedirection();

app.UseAuthorization();

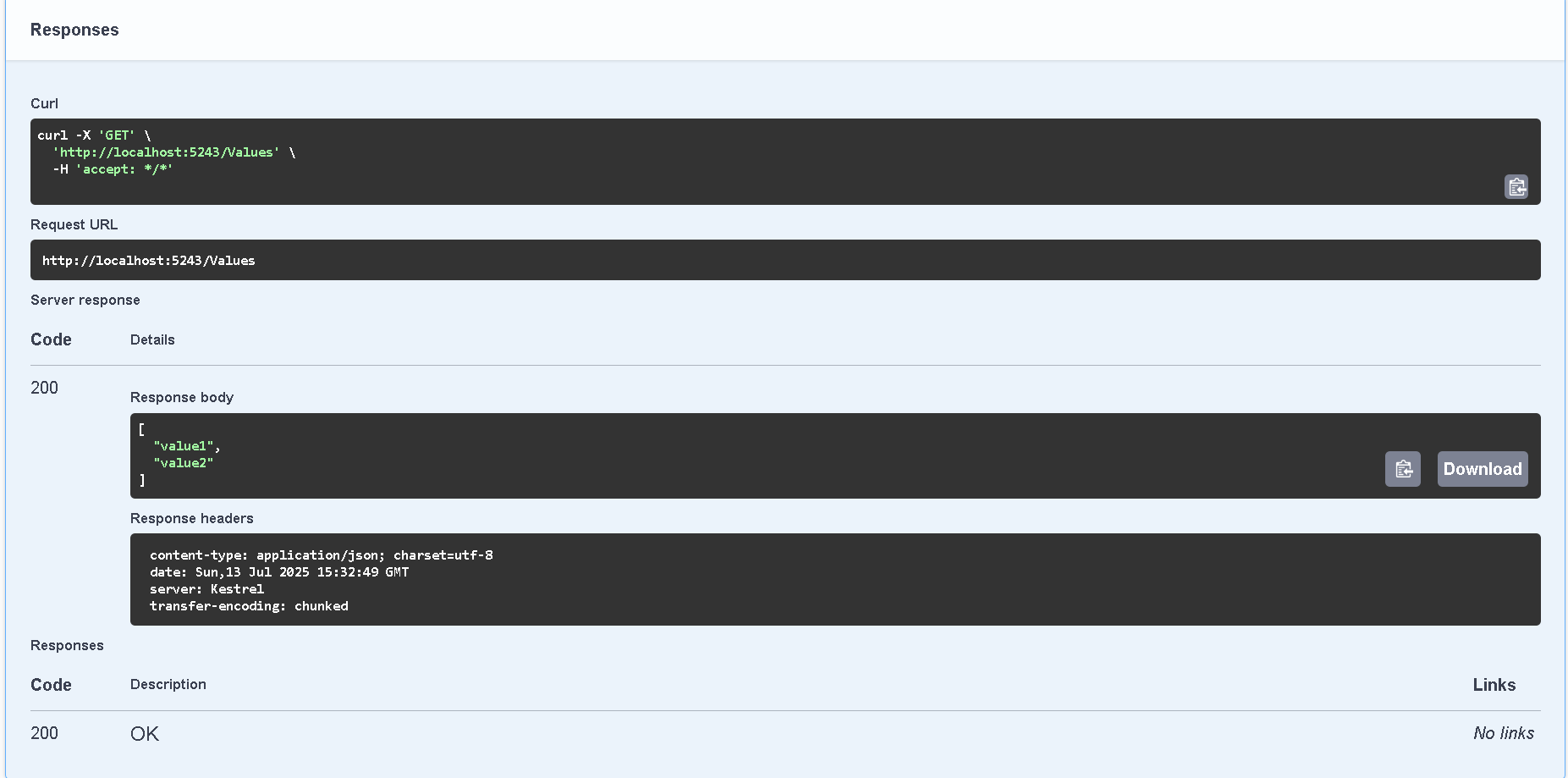
app.MapControllers();

app.Run();

OUTPUT:



GET



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.

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace FirstWebAPI.Controllers

{

[ApiController]

[Route("Emp")]

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

public class EmployeeController : ControllerBase

{

private static List<string> employees = new List<string> { "Alice", "Bob", "Charlie" };

[HttpGet]

public IActionResult Get()

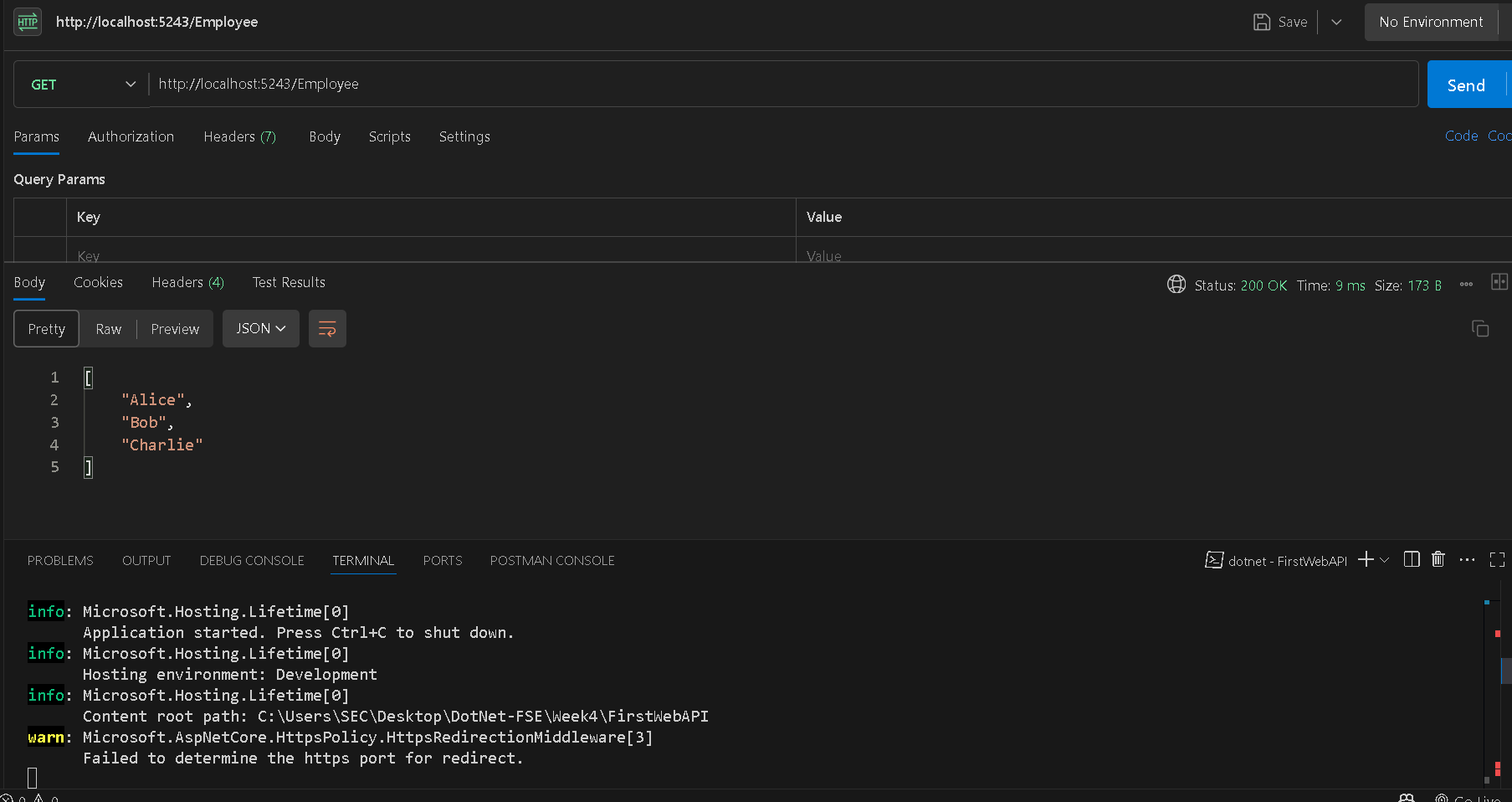
{

return Ok(employees);

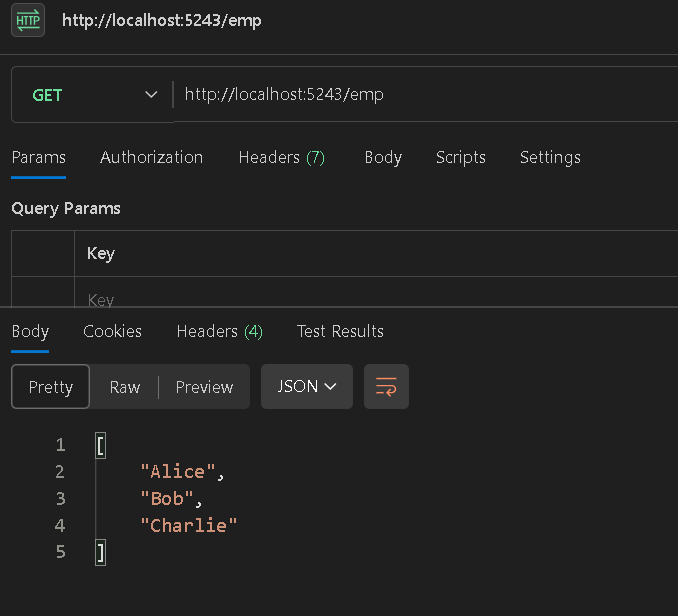
}

}

}



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



3. **Web Api using custom model class**

Create a Custom class ‘Employee’ of the below defined structure

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; }

}

Create a new controller - EmployeeController with Read Write actions

Constructor: Create few records, HTTPGet, HTTPPost/HTTPPut

Create a Private method GetStandardEmployeeList that returns a List of Employee class. Invoke this method in the Get action method of the EmployeeController that was created in the previous step.

Public ActionResult<Employee> GetStandrad()

Modify the return type of the Get action method(without parameter) to return List of Employee class object

Add ProducesResponseType to the GET action method for Status code 200

Check the Swagger description for the GET method for success status code

**Create a Custom action filter for Authorization.**

The requirement is to intercept incoming requests and check if there is a key ‘Authorization’ in the request header or not. If it is there, then to check if it contains a value ‘Bearer’ or not.

Create a folder ‘Filters’ in the application solution. Create a class ‘**CustomAuthFilter**’ to filter requests. Inherit ActionFilterAttribute. Override OnActionExecuting method to check if the request object has Header ‘Authorization’ or not. If not, throw BadRequestResult with the message

Invalid request - No Auth token

If the header is present, then check if the value contains the word ‘Bearer’. If not, throw BadRequestResult with the message

Invalid request - Token present but Bearer unavailable

Add an attribute **CustomAuthFilter** to the Employee controller to filter any request to check for the Authorization token in the request header.

**Custom Exception filter**

Create a class ‘CustomExceptionFilter’ to catch the exceptions occuring the application. Implement IExceptionFilter thru the OnException method  
  
Use the exception context to fetch the exception detail. Capture that and write it to a File in the system.  
  
Set the Result property of the exception context to ExceptionResult.  
  
Throw an exception in GET action method.  
Ensure that the GET action method has ProducesResponseType for 500 - Internal server error  
  
Use Swagger to test the exception and message being thrown.  
  
Note: This needs WebApiCompatShim NuGet package installation

**SOLUTION:**

Program.cs

using Microsoft.AspNetCore.Builder;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using FirstWebAPI.Filters;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers(options =>

{

options.Filters.Add<CustomExceptionFilter>(); // Register exception filter globally

});

builder.Services.AddScoped<CustomExceptionFilter>(); // Register for DI

// Add Swagger/OpenAPI support

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure middleware

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

Models

Employee.cs

using System;

using System.Collections.Generic;

namespace FirstWebAPI.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 FirstWebAPI.Models

{

public class Skill

{

public int Id { get; set; }

public string Name { get; set; }

}

}

Department.cs

namespace FirstWebAPI.Models

{

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

}

}

CustomAuthFilter.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace FirstWebAPI.Filters

{

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

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

if (!hasAuthHeader)

{

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;

using System;

using System.IO;

namespace FirstWebAPI.Filters

{

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

var exception = context.Exception;

var logPath = "exceptions.log";

File.AppendAllText(logPath, $"{DateTime.Now}: {exception.Message}{Environment.NewLine}{exception.StackTrace}{Environment.NewLine}");

var result = new ObjectResult(new

{

Message = "An unexpected error occurred.",

Detail = exception.Message

})

{

StatusCode = 500

};

context.Result = result;

context.ExceptionHandled = true;

}

}

}

EmployeeController.cs

using Microsoft.AspNetCore.Mvc;

using FirstWebAPI.Filters;

using FirstWebAPI.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace FirstWebAPI.Controllers

{

[ApiController]

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

[CustomAuthFilter] // Apply custom auth filter here

public class EmployeeController : ControllerBase

{

// Static employee list to persist data during app lifetime

private static List<Employee> \_employees = GetStandardEmployeeList();

private static 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 = "Recruitment" }

},

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, 21)

}

};

}

// GET: api/Employee

[HttpGet]

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

public ActionResult<List<Employee>> GetStandard()

{

return Ok(\_employees);

}

// GET api/Employee/exception -- to test exception filter

[HttpGet("exception")]

[ProducesResponseType(500)]

public IActionResult ThrowException()

{

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

}

// POST: api/Employee

[HttpPost]

public ActionResult<Employee> CreateEmployee([FromBody] Employee employee)

{

// Simple ID generation for demo purposes

if (\_employees.Any())

employee.Id = \_employees.Max(e => e.Id) + 1;

else

employee.Id = 1;

\_employees.Add(employee);

return CreatedAtAction(nameof(GetStandard), new { id = employee.Id }, employee);

}

// PUT: api/Employee/{id}

[HttpPut("{id}")]

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

{

var existing = \_employees.Find(e => e.Id == id);

if (existing == null)

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

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 NoContent();

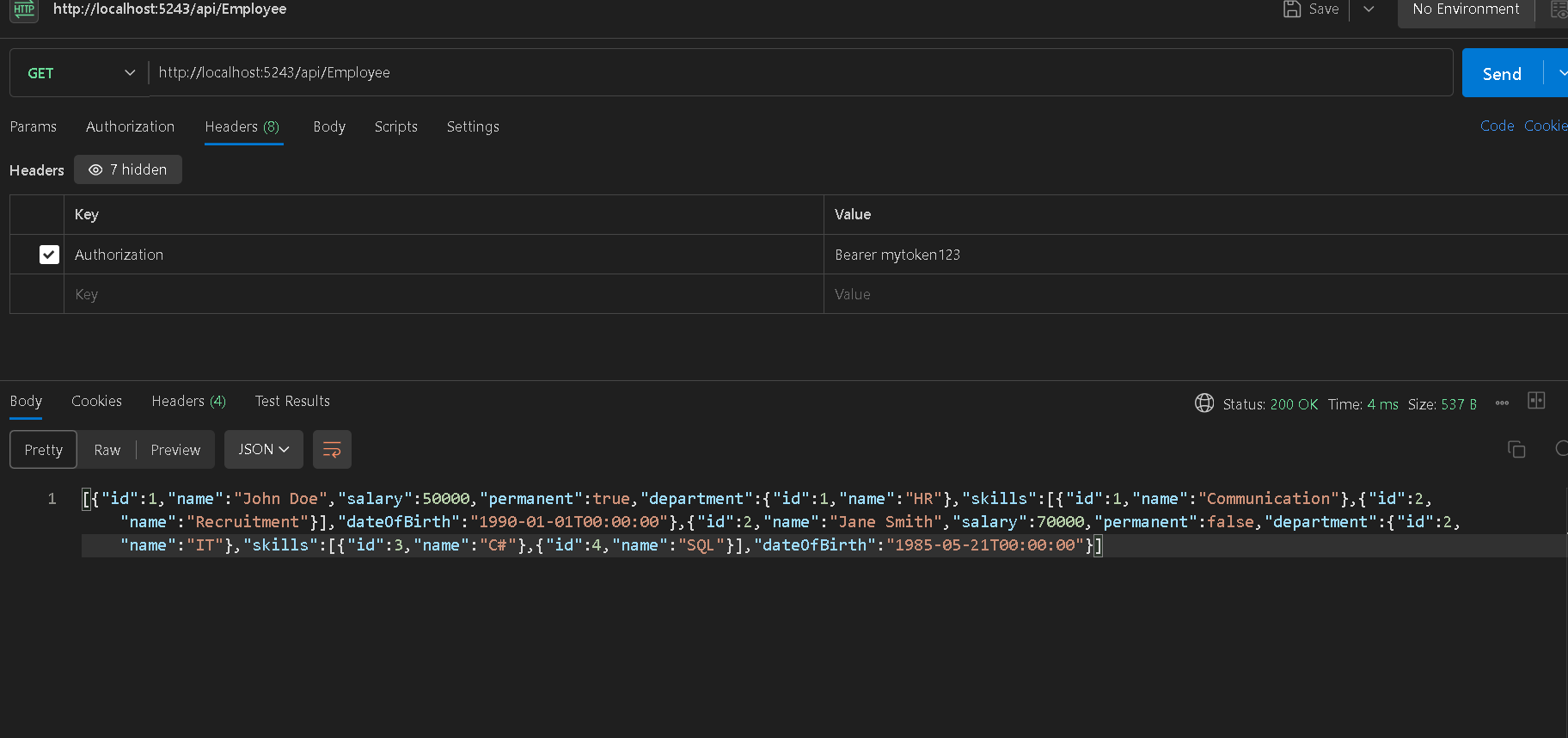
}

}

}

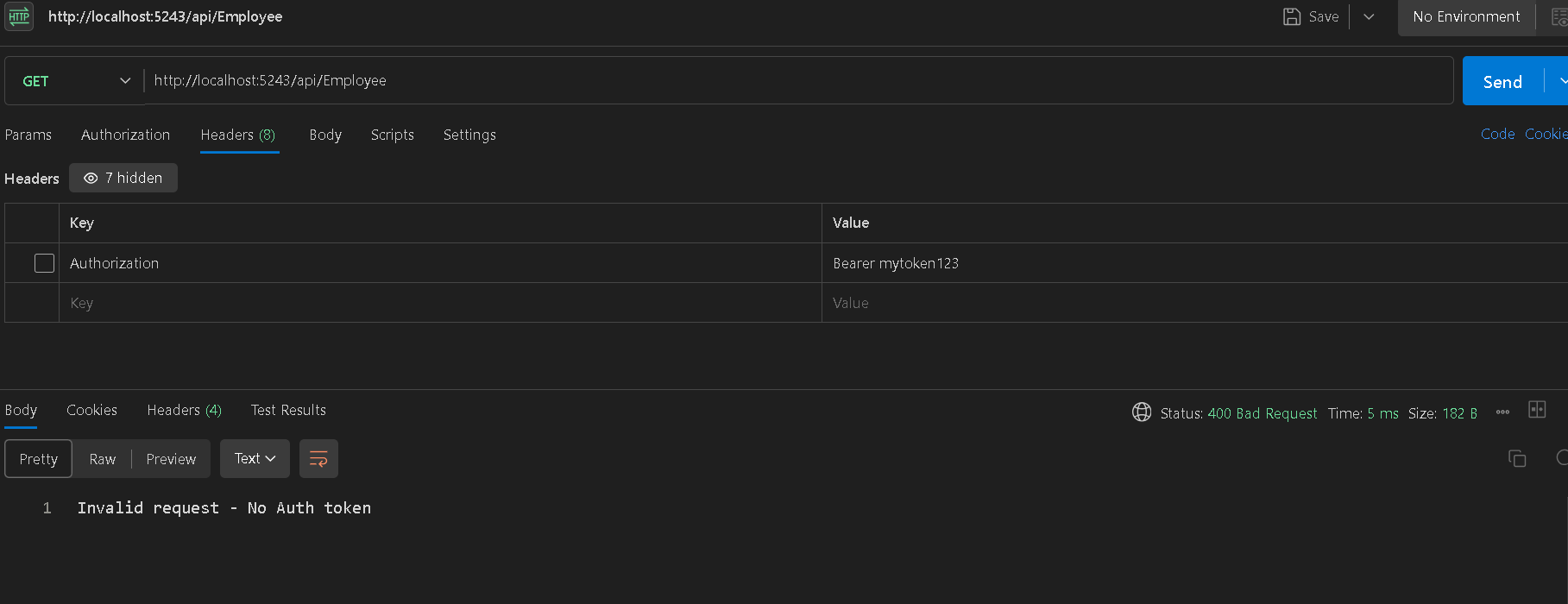
OUTPUT

With Authorization token



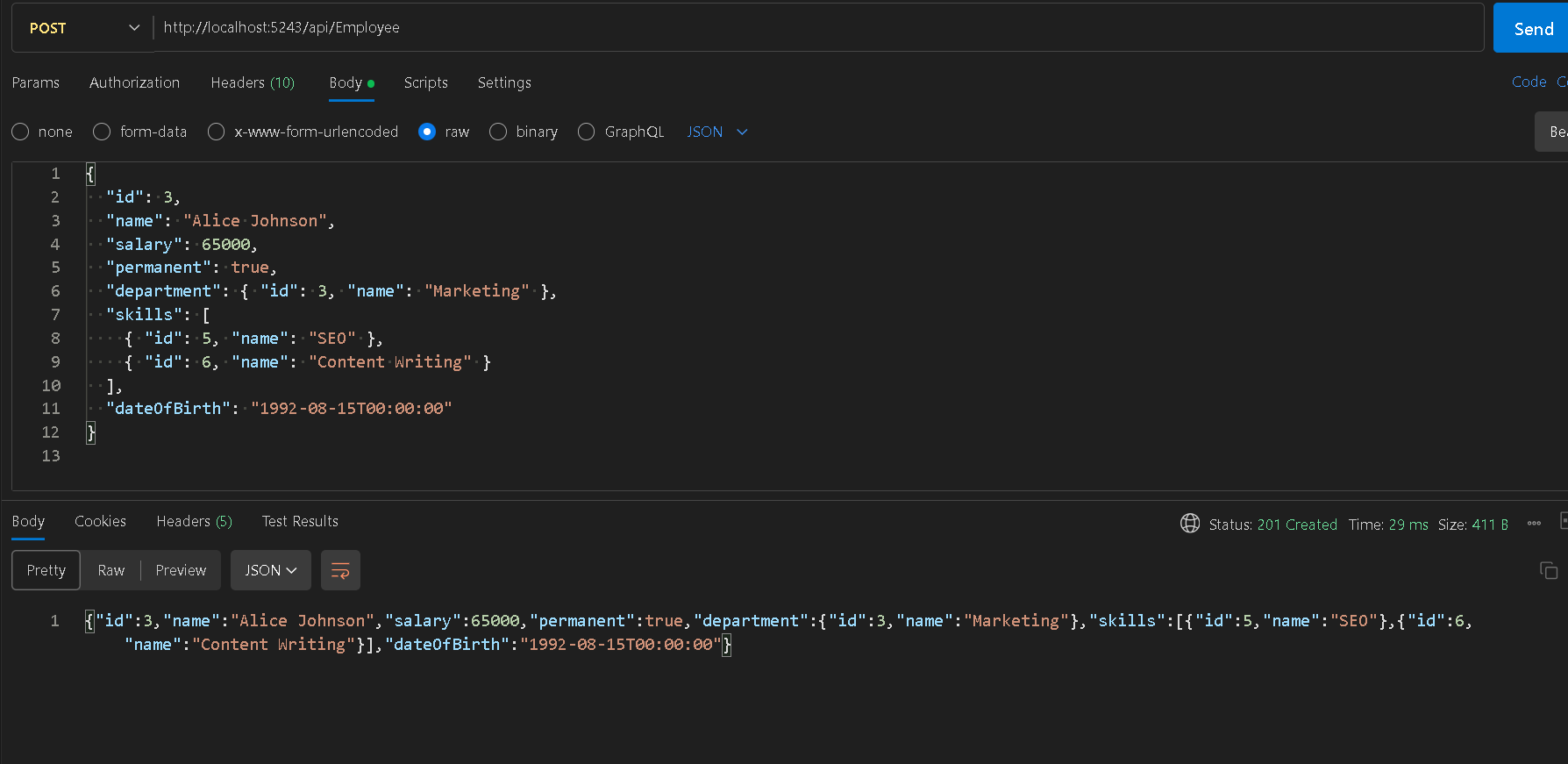
Without Authorization token

GET method

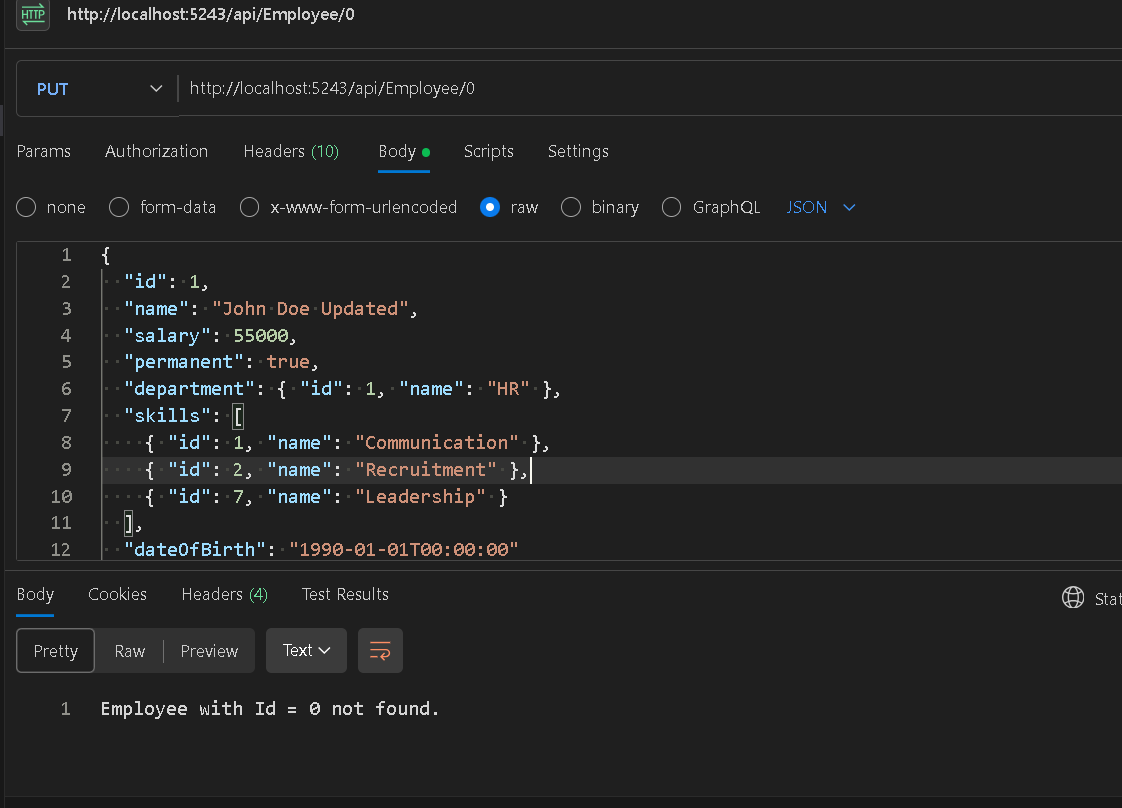


POST method

Status 201 Created



PUT



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.

EmployeeController.cs

using Microsoft.AspNetCore.Mvc;

using FirstWebAPI.Filters;

using FirstWebAPI.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace FirstWebAPI.Controllers

{

[ApiController]

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

[CustomAuthFilter] // Apply custom auth filter here

public class EmployeeController : ControllerBase

{

// Static employee list (data persists during app lifetime)

private static List<Employee> \_employees = GetStandardEmployeeList();

// Initialize with hardcoded employee list

private static 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 = "Recruitment" }

},

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, 21)

}

};

}

// GET: api/Employee

[HttpGet]

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

public ActionResult<List<Employee>> GetStandard()

{

return Ok(\_employees);

}

// GET: api/Employee/exception (test exception filter)

[HttpGet("exception")]

[ProducesResponseType(500)]

public IActionResult ThrowException()

{

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

}

// POST: api/Employee

[HttpPost]

[ProducesResponseType(typeof(Employee), 201)]

public ActionResult<Employee> CreateEmployee([FromBody] Employee employee)

{

// Auto-generate ID

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

\_employees.Add(employee);

return CreatedAtAction(nameof(GetStandard), new { id = employee.Id }, employee);

}

// PUT: api/Employee/{id}

[HttpPut("{id}")]

[ProducesResponseType(typeof(Employee), 200)]

[ProducesResponseType(400)]

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

{

// Step 1: Validate ID

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

// Step 2: Find existing employee

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

if (existing == null)

{

return BadRequest("Invalid employee id");

}

// Step 3: Update fields

existing.Name = employee.Name;

existing.Salary = employee.Salary;

existing.Permanent = employee.Permanent;

existing.Department = employee.Department;

existing.Skills = employee.Skills;

existing.DateOfBirth = employee.DateOfBirth;

// Step 4: Return updated object

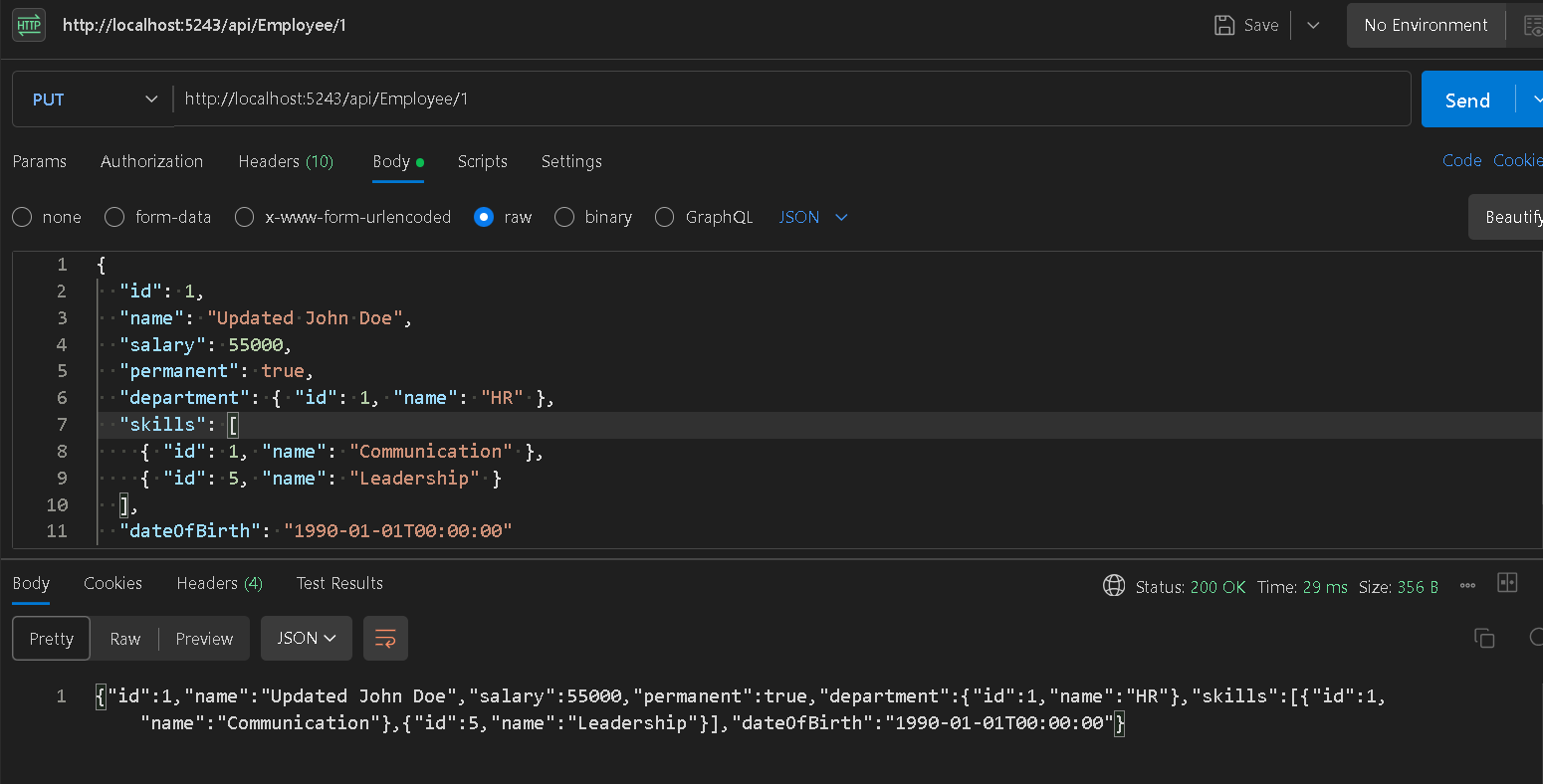
return Ok(existing);

}

}

}

OUTPUT:



5. **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.

**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.

4. **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

Program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

var jwtSettings = builder.Configuration.GetSection("JwtSettings");

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtSettings["Issuer"],

ValidAudience = jwtSettings["Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings["SecretKey"])),

ClockSkew = TimeSpan.Zero

};

});

builder.Services.AddControllers();

var app = builder.Build();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

AuthController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace FirstWebAPI.Controllers

{

[ApiController]

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

public class AuthController : ControllerBase

{

private readonly IConfiguration \_configuration;

public AuthController(IConfiguration configuration)

{

\_configuration = configuration;

}

public class LoginRequest

{

public string Username { get; set; }

public string Password { get; set; }

}

[HttpPost("login")]

public IActionResult Login([FromBody] LoginRequest loginRequest)

{

// For demo: simple username/password validation (replace with your own)

if (loginRequest.Username != "admin" || loginRequest.Password != "password")

return Unauthorized("Invalid credentials");

var jwtSettings = \_configuration.GetSection("JwtSettings");

var secretKey = jwtSettings["SecretKey"];

var issuer = jwtSettings["Issuer"];

var audience = jwtSettings["Audience"];

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var claims = new[]

{

new Claim(ClaimTypes.Name, loginRequest.Username)

};

var token = new JwtSecurityToken(

issuer: issuer,

audience: audience,

claims: claims,

expires: DateTime.UtcNow.AddHours(1),

signingCredentials: creds);

var tokenString = new JwtSecurityTokenHandler().WriteToken(token);

return Ok(new { Token = tokenString });

}

}

}

OUTPUT:

