**Hands-on 1:**

**Step 1:** Explain the concept of RESTful web service, Web API & Microservice  
  
*a. RESTful Web Service, Web API & Microservice*

* **RESTful Web Service**: Architectural style using HTTP methods (GET, POST, PUT, DELETE) to interact with resources.
* **Web API**: Framework for building HTTP services that can be accessed from any client.
* **Microservice**: Independent deployable services; loosely coupled; focused on specific business capabilities.

*b. Features of REST Architecture*

* **Stateless**: No client context stored on server between requests.
* **Messages**: Use standard HTTP methods and status codes.
* **Flexible format**: Not limited to XML; can use JSON, etc.
* **Microservice-compatible**: REST is suitable for microservice design.

*c. Difference: WebService vs WebAPI*

|  |  |
| --- | --- |
| **WebService** | **WebAPI** |
| SOAP-based | RESTful (typically) |
| Returns XML | Returns JSON/XML |
| Slower due to overhead | Lightweight, faster |
| Requires .NET client | Accessible from any client |

**Step 2**: Explain what is HttpRequest & HttpResponse

HttpRequest: Represents incoming HTTP request (contains method, headers, body, query string).

HttpResponse: Represents outgoing response (contains status code, headers, body).  
  
  
  
  
  
  
**Step 3**: List the types of Action Verbs

**HttpGet**: Retrieve resource.  
**HttpPost**: Create resource.  
**HttpPut**: Update resource.  
**HttpDelete**: Delete resource.

**Step 4:** List the types of HttpStatusCodes used in WebAPI

**200 OK** – Success  
**400 BadRequest** – Client error  
**401 Unauthorized** – Auth required  
**500 InternalServerError** – Server error  
  
  
**Step 5:** Demonstrate creation of a simple WebAPI - With Read, Write actions

**Controller**: Class that handles HTTP requests. Inherits from ControllerBase.  
**Action Methods**: Methods inside controller, decorated with [HttpGet], [HttpPost], etc.  
**Routing**: Set using [Route("controllerName")] and parameterized with {id} if needed.  
**Return Types**: Typically IActionResult or ActionResult<T> to return data with status codes.

Example:  
[ApiController]  
[Route("[controller]")]  
public class ValuesController : ControllerBase  
{  
 [HttpGet]  
 public IActionResult Get() => Ok(new[] { "val1", "val2" });  
}

**Step 6:** Explain the types of Configuration files of WebAPI

*a. Startup.cs*

* Configure services (e.g., DI, Swagger)
* Configure middleware (e.g., routing, auth)

*b. appsettings.json*

* App configuration (e.g., connection strings)

*c. launchSettings.json*

* Launch profile for debugging

*d. In .NET Framework (.NET 4.5)*

* Route.config: Route mapping
* WebApi.config: API specific settings

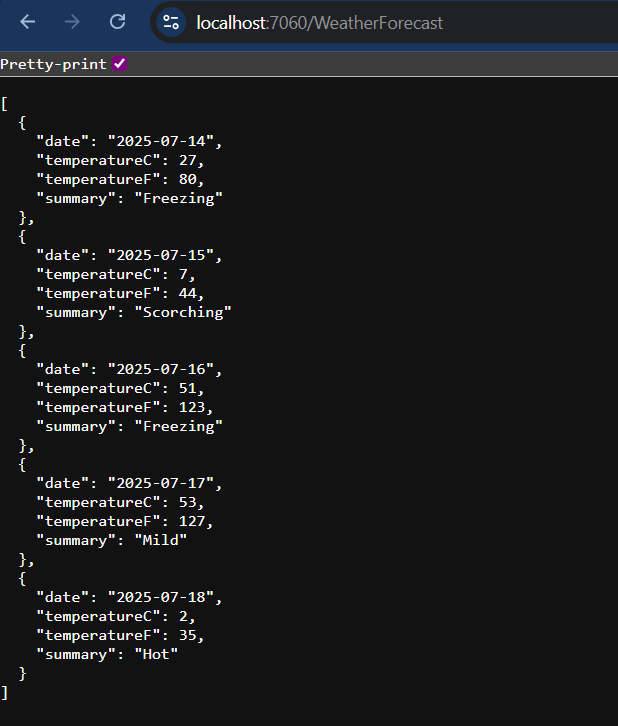
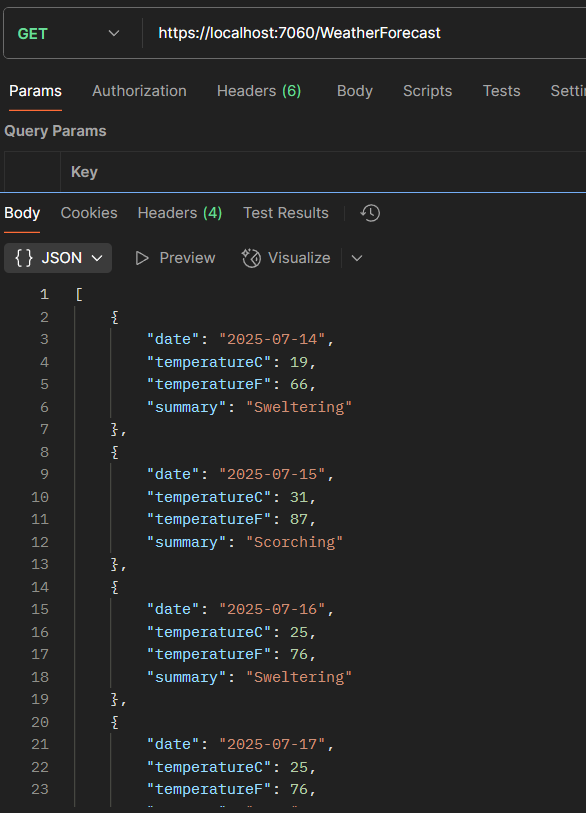
**Step 7:** First Web Api using .Net core  
Code:  
  
public class Program  
{  
 public static void Main(string[] args)  
 {  
 var builder = WebApplication.CreateBuilder(args);  
   
 // Add services to the container.  
   
 builder.Services.AddControllers();  
 // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle  
 builder.Services.AddEndpointsApiExplorer();  
 builder.Services.AddSwaggerGen();  
   
 var app = builder.Build();  
   
 // Configure the HTTP request pipeline.  
 if (app.Environment.IsDevelopment())  
 {  
 app.UseSwagger();  
 app.UseSwaggerUI();  
 }  
  
 app.UseHttpsRedirection();

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

public class WeatherForecast  
{  
 public DateOnly Date { get; set; }  
   
 public int TemperatureC { get; set; }  
   
 public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);  
   
 public string? Summary { get; set; }  
}

[ApiController]  
[Route("[controller]")]  
public class WeatherForecastController : ControllerBase  
{  
 private static readonly string[] Summaries = new[]  
 {  
 "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering",  
"Scorching"  
 };  
   
 private readonly ILogger<WeatherForecastController> \_logger;  
   
 public WeatherForecastController(ILogger<WeatherForecastController> logger)  
 {  
 \_logger = logger;  
 }  
   
 [HttpGet(Name = "GetWeatherForecast")]  
 public IEnumerable<WeatherForecast> Get()  
 {  
 return Enumerable.Range(1, 5).Select(index => new WeatherForecast  
 {  
 Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),  
 TemperatureC = Random.Shared.Next(-20, 55),  
 Summary = Summaries[Random.Shared.Next(Summaries.Length)]  
 })  
 .ToArray();  
 }  
}

Output:

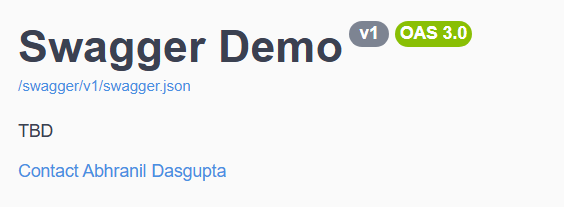
  
  
**Hands-on 2:**

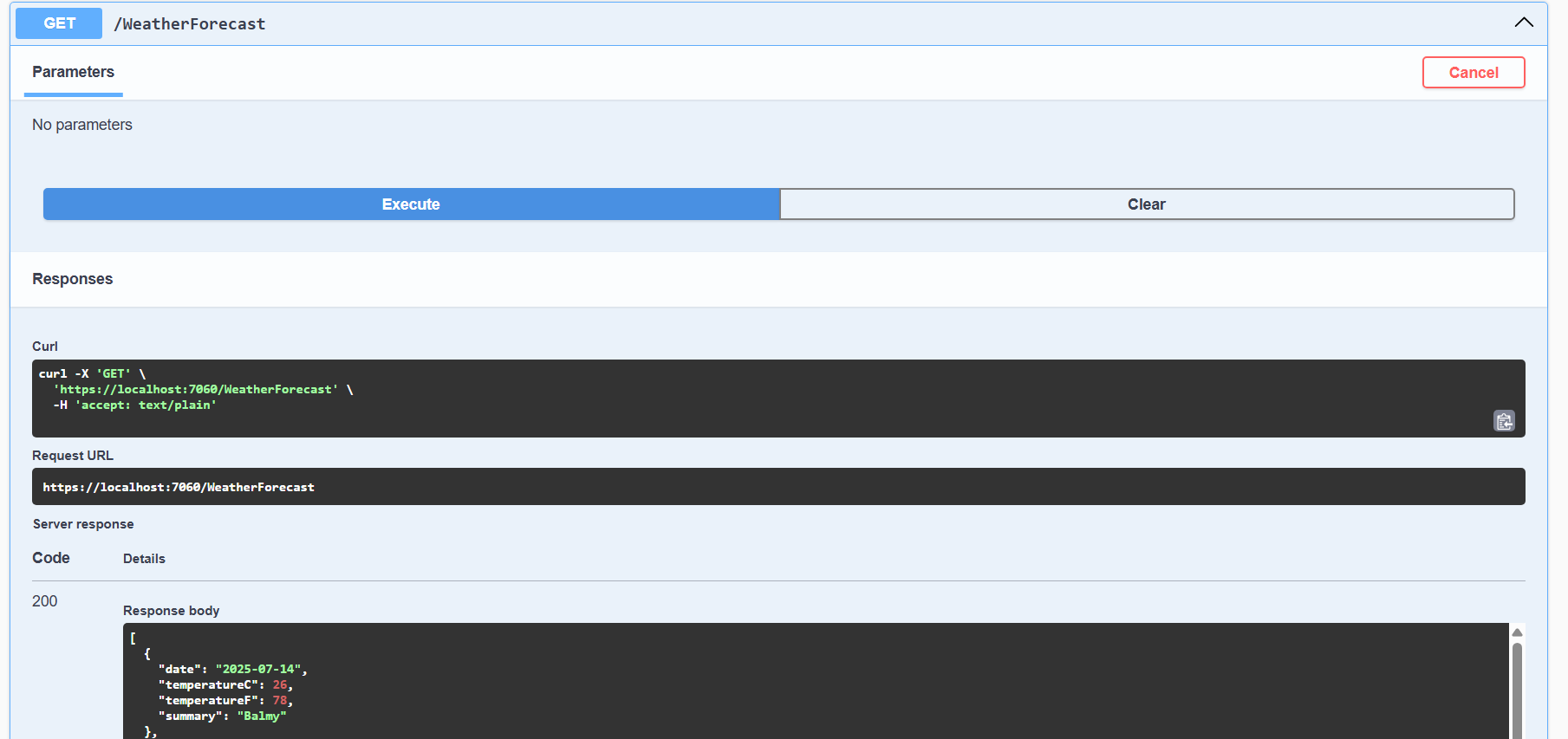
**Step 1:** Web Api using .Net core with Swagger

Code:

public class Program  
{  
 public static void Main(string[] args)  
 {  
 var builder = WebApplication.CreateBuilder(args);  
   
 // Add services to the container.  
   
 builder.Services.AddControllers();  
 builder.Services.AddEndpointsApiExplorer();  
 builder.Services.AddSwaggerGen(c =>  
 {  
 c.SwaggerDoc("v1", new()  
 {  
 Title = "Swagger Demo",  
 Version = "v1",  
 Description = "TBD",  
 Contact = new() { Name = "Abhranil Dasgupta", Email = "abhranilnxt@gmail.com" }  
 });  
 });  
   
 var app = builder.Build();  
   
 // Configure the HTTP request pipeline.  
 if (app.Environment.IsDevelopment())  
 {  
 app.UseSwagger();  
 app.UseSwaggerUI(c =>  
 {  
 c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");  
 });  
 }  
   
 app.UseHttpsRedirection();  
   
 app.UseAuthorization();  
   
 app.MapControllers();  
   
 app.Run();  
 }  
}

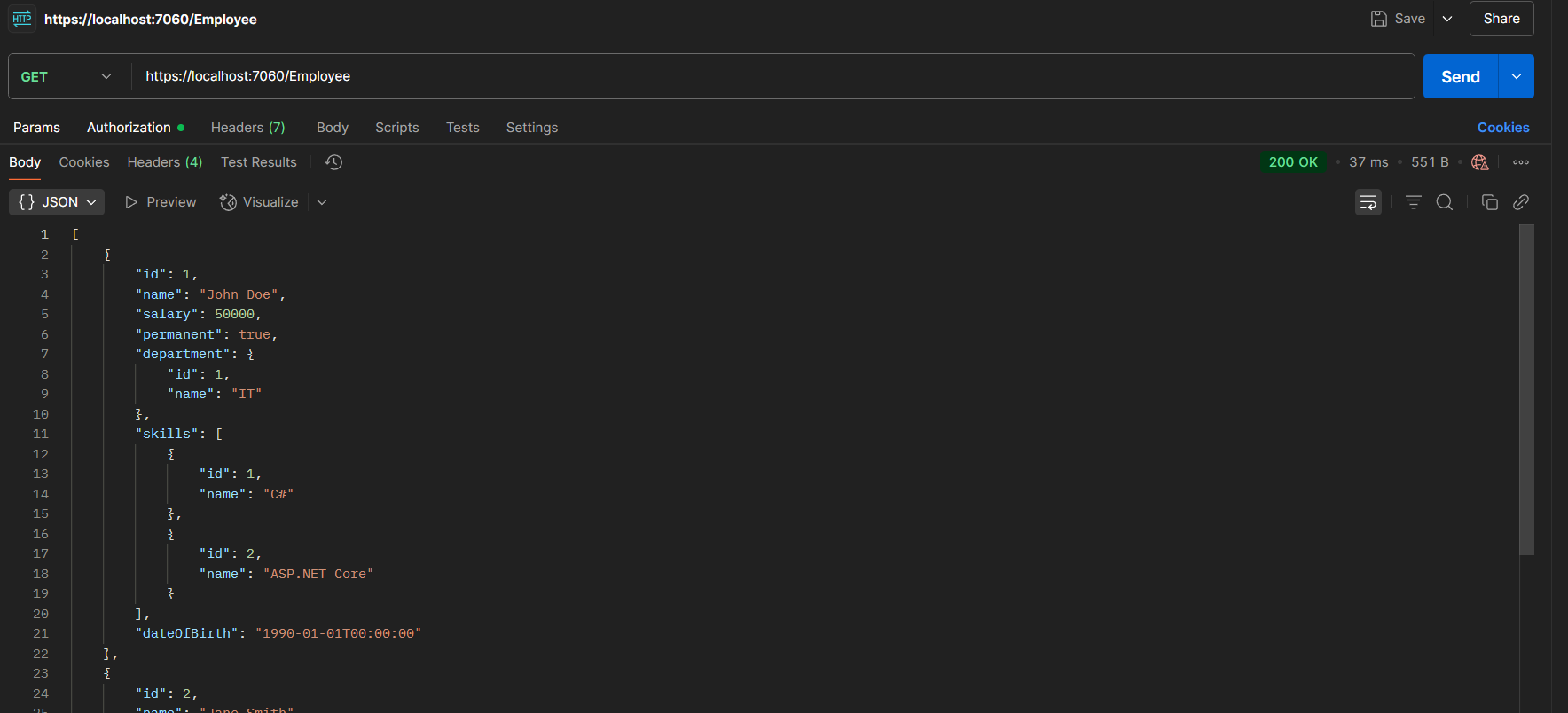
Output:





**Step 2:** Use POSTMAN tool, to point to the local Web API that was created with Employee controller

Output:

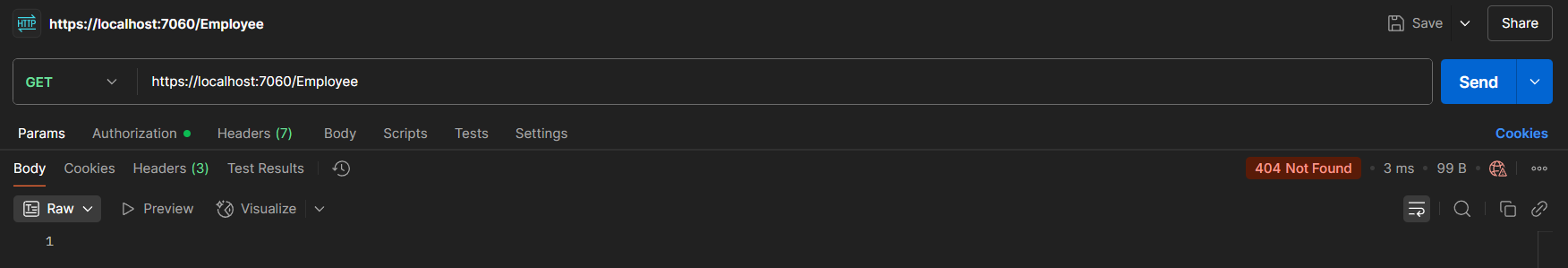


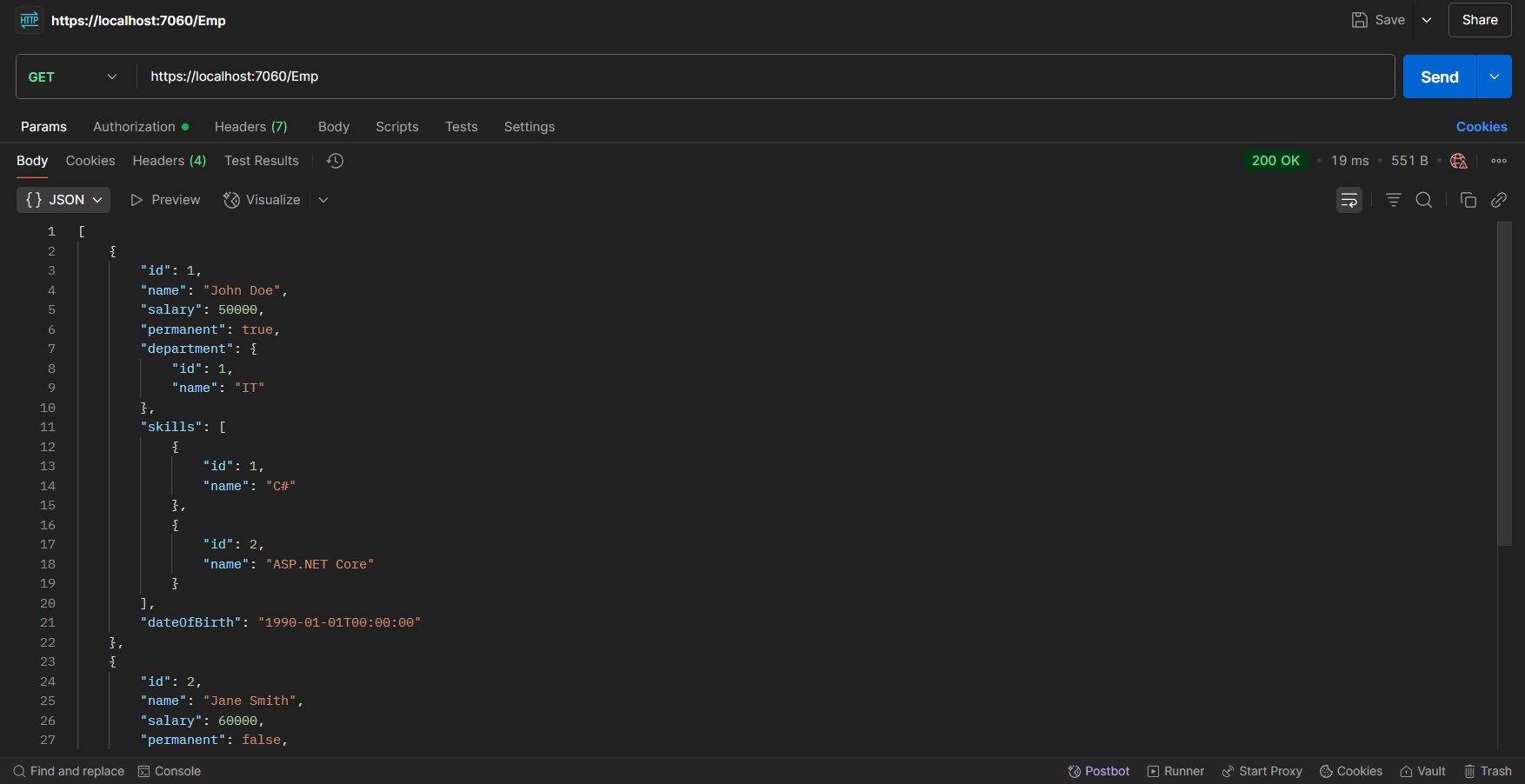
**Step 3:** Modify the Controller name in the Route attribute of the Employee controller to ‘Emp’ and check its access thru POSTMAN

Code:

[Route("Emp")]  
[ApiController]  
public class EmployeeController : ControllerBase  
{  
 //Rest of the boilerplate code  
}

Output:





**Hands-on 3:**

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

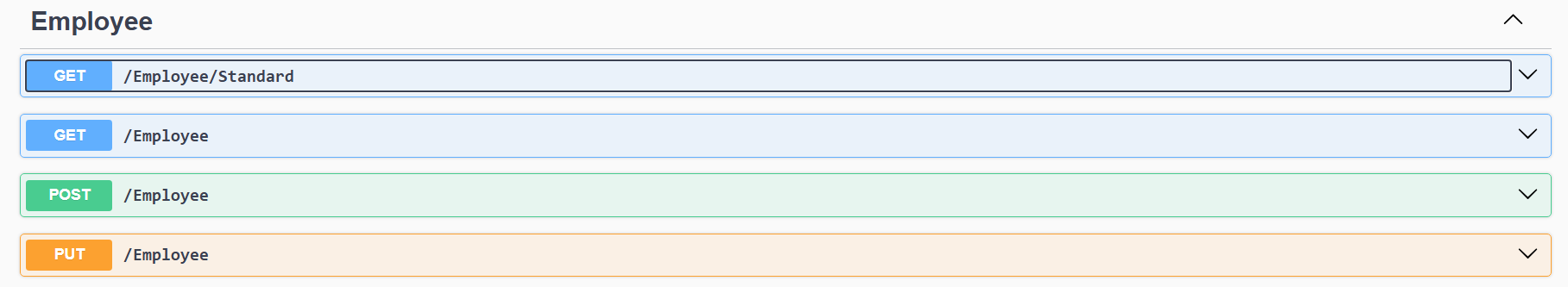
Code:

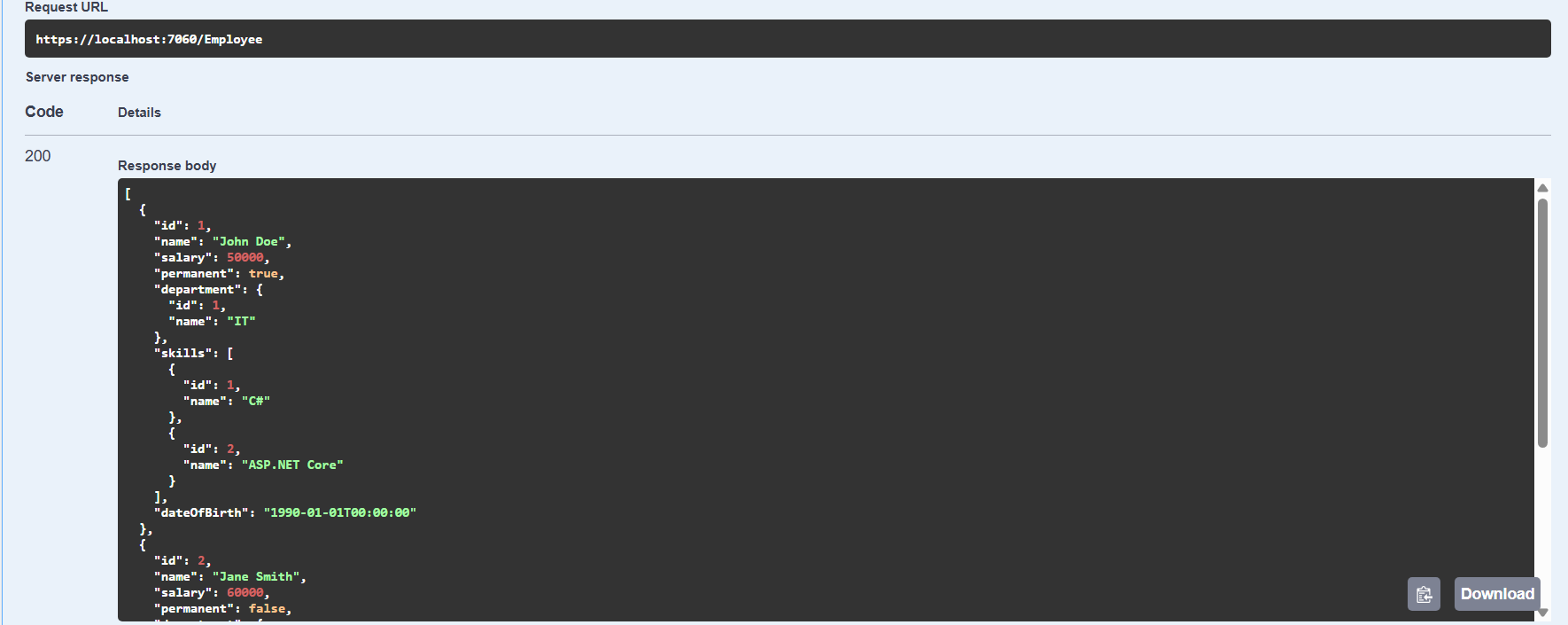
public class Employee  
{  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public int Salary { get; set; }  
 public bool Permanent { get; set; }  
 public Department Department { get; set; }  
 public List<Skill> Skills { get; set; }  
 public DateTime DateOfBirth { get; set; }  
}

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

public class Skill  
{  
 public int Id { get; set; }  
 public string Name { get; set; }  
}

[Route("[controller]")]  
[ApiController]  
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 = "IT" },  
 Skills = new List<Skill>  
 {  
 new Skill { Id = 1, Name = "C#" },  
 new Skill { Id = 2, Name = "ASP.NET Core" }  
 },  
 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 = "Recruitment" },  
 new Skill { Id = 4, Name = "Employee Relations" }  
 },  
 DateOfBirth = new DateTime(1985, 5, 15)  
 }  
 };  
 }  
   
 [HttpGet("Standard")]  
 public ActionResult<Employee> GetStandard() {  
 return Ok(\_employees.First());  
 }  
   
 [HttpGet]  
 [ProducesResponseType(200)]  
 public ActionResult<List<Employee>> GetAll()  
 {  
 return Ok(\_employees);  
 }  
   
 [HttpPost]  
 public IActionResult AddEmployee([FromBody] Employee emp)  
 {  
 \_employees.Add(emp);  
 return Ok(emp);  
 }  
   
 [HttpPut]  
 public IActionResult UpdateEmployee([FromBody] Employee emp)  
 {  
 var existing = \_employees.FirstOrDefault(e => e.Id == emp.Id);  
 if (existing == null)  
 return NotFound();  
   
 existing.Name = emp.Name;  
 existing.Salary = emp.Salary;  
 existing.Permanent = emp.Permanent;  
 existing.DateOfBirth = emp.DateOfBirth;  
 existing.Department = emp.Department;  
 existing.Skills = emp.Skills;  
   
 return Ok(existing);  
 }   
}

Output:  




**Step 2:** Create a Custom action filter for Authorization

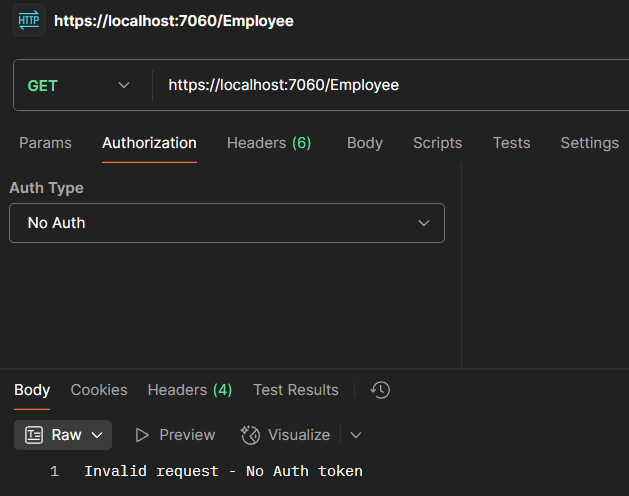
Code:

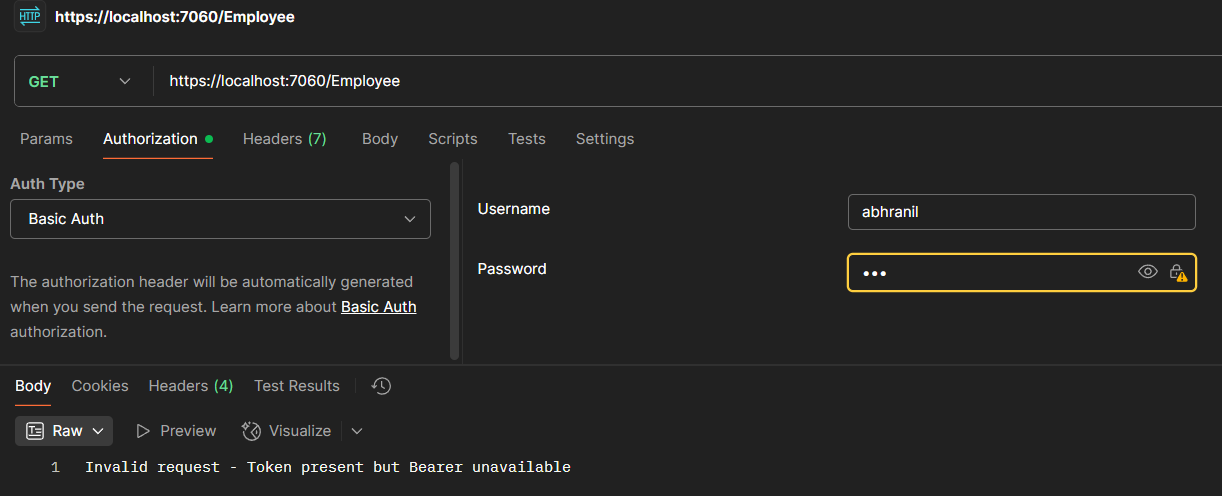
public class CustomAuthFilter : ActionFilterAttribute  
{  
 public override void OnActionExecuting(ActionExecutingContext context)  
 {  
 var headers = context.HttpContext.Request.Headers;  
   
 if (!headers.ContainsKey("Authorization"))  
 {  
 context.Result = new BadRequestObjectResult("Invalid request - No Auth token");  
 return;  
 }  
   
 var token = headers["Authorization"].ToString();  
   
 if (!token.Contains("Bearer"))  
 {  
 context.Result = new BadRequestObjectResult("Invalid request - Token present but  
Bearer unavailable");  
 }  
 }  
}

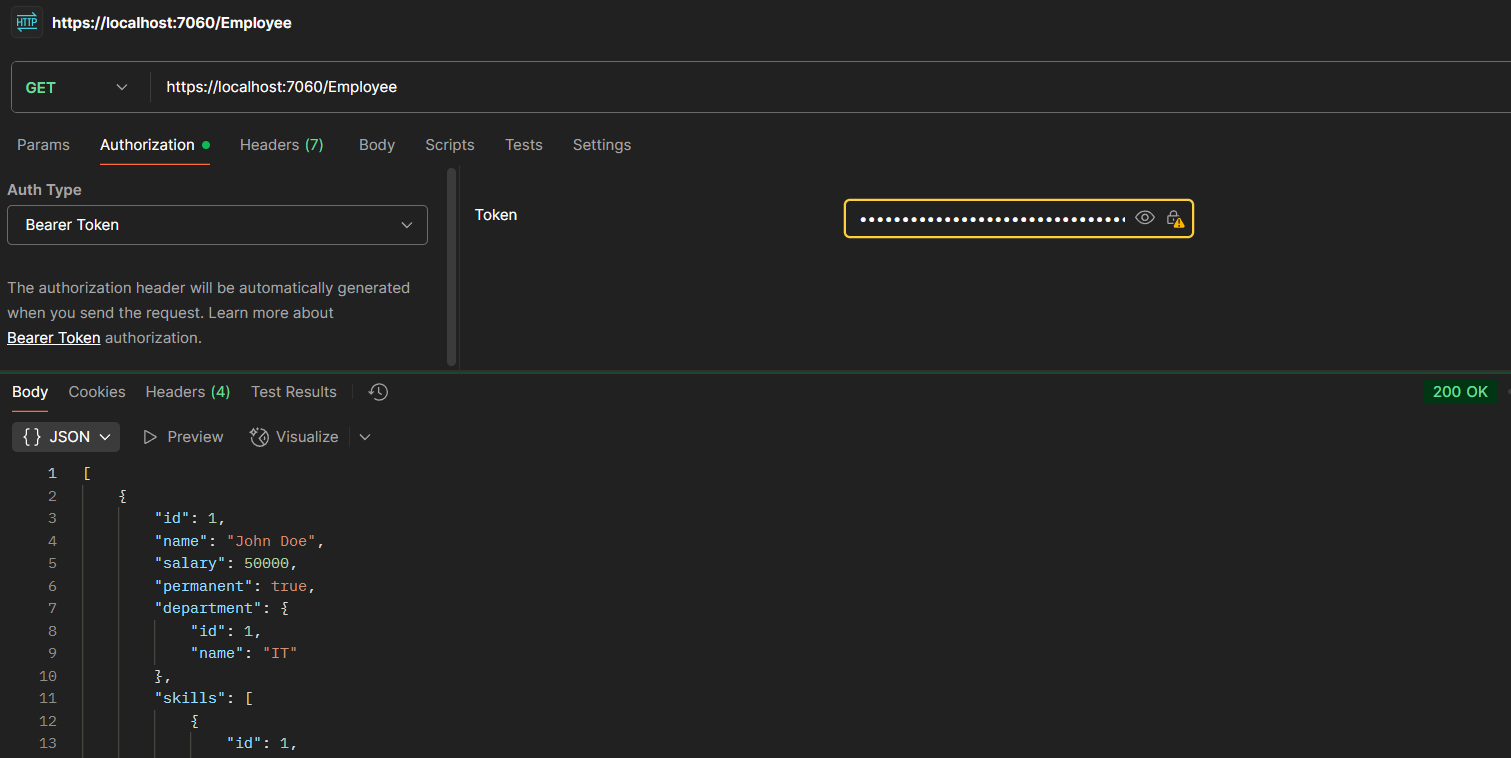
public class Program  
{  
 public static void Main(string[] args)  
 {  
 var builder = WebApplication.CreateBuilder(args);  
   
 // Add services to the container.  
   
 builder.Services.AddControllers();  
 builder.Services.AddEndpointsApiExplorer();  
 *builder.Services.AddScoped<CustomAuthFilter>();*  
  
//Rest of the boilerplate code  
  
 }  
}

[Route("[controller]")]  
[ApiController]  
*[ServiceFilter(typeof(CustomAuthFilter))]*  
public class EmployeeController : ControllerBase  
{  
 //Rest of the boilerplate code  
}

Output:







**Step 3:** Custom Exception filter

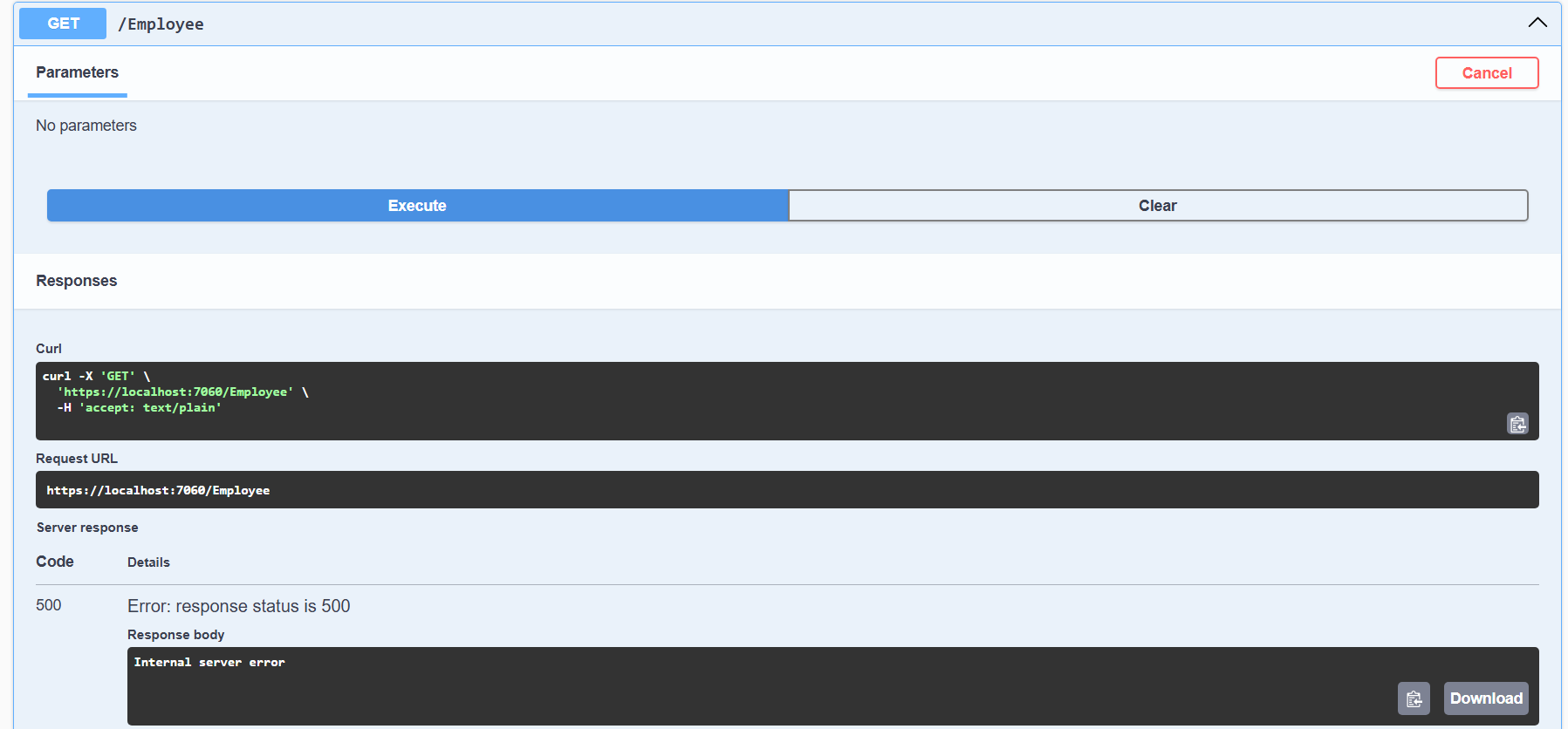
Code:

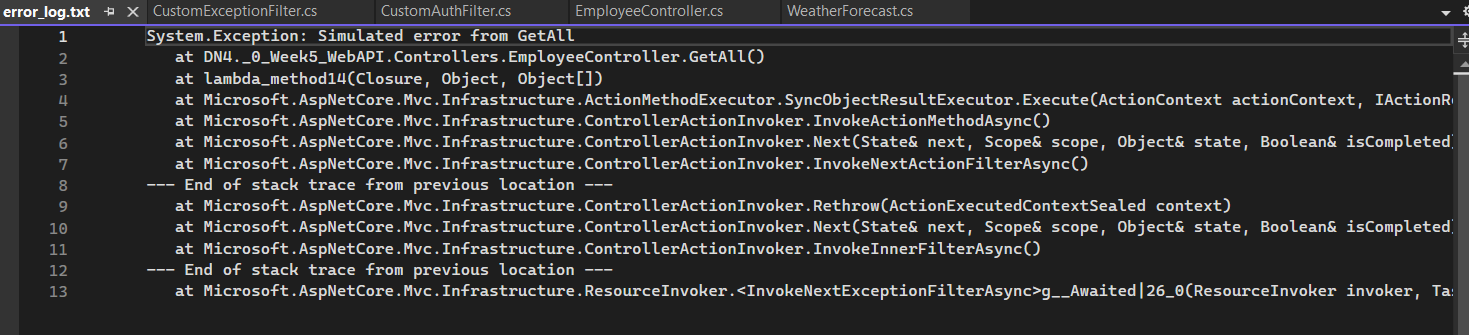
public class CustomExceptionFilter : IExceptionFilter  
{  
 public void OnException(ExceptionContext context)  
 {  
 string logPath = Path.Combine(Directory.GetCurrentDirectory(), "error\_log.txt");  
 File.WriteAllText(logPath, context.Exception.ToString());  
   
 context.Result = new ObjectResult("Internal server error")  
 {  
 StatusCode = 500  
 };  
 }  
}

public class Program  
{  
 public static void Main(string[] args)  
 {  
 var builder = WebApplication.CreateBuilder(args);  
   
 // Add services to the container.  
   
 builder.Services.AddControllers();  
 builder.Services.AddEndpointsApiExplorer();  
 builder.Services.AddScoped<CustomAuthFilter>();  
 *builder.Services.AddScoped<CustomExceptionFilter>();*  
  
//Rest of the boilerplate code  
  
 }  
}

[Route("[controller]")]  
[ApiController]  
[ServiceFilter(typeof(CustomAuthFilter))]  
*[TypeFilter(typeof(CustomExceptionFilter))]*  
public class EmployeeController : ControllerBase  
{  
 //Rest of the boilerplate code  
  
 [HttpGet]  
 [ProducesResponseType(200)]  
 [ProducesResponseType(500)]  
 public ActionResult<List<Employee>> GetAll()  
 {  
 throw new Exception("Simulated error from GetAll");  
 //return Ok(\_employees);  
 }  
  
}

Output:





**Hands-on 4:**

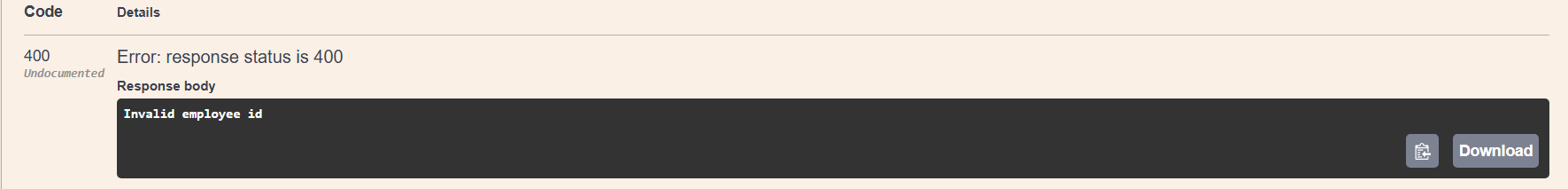
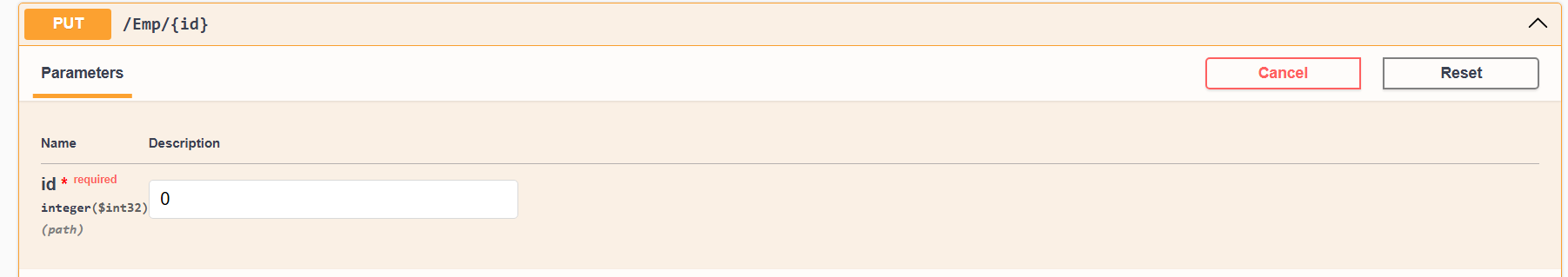
**Step 1:** Web Api CRUD operation

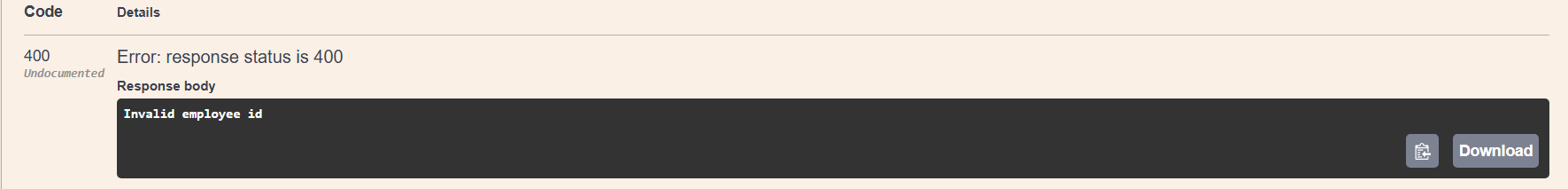
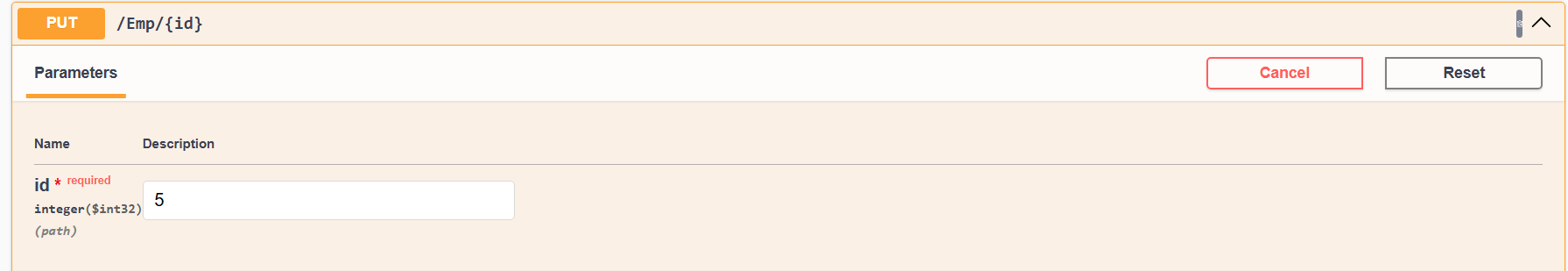
Code:

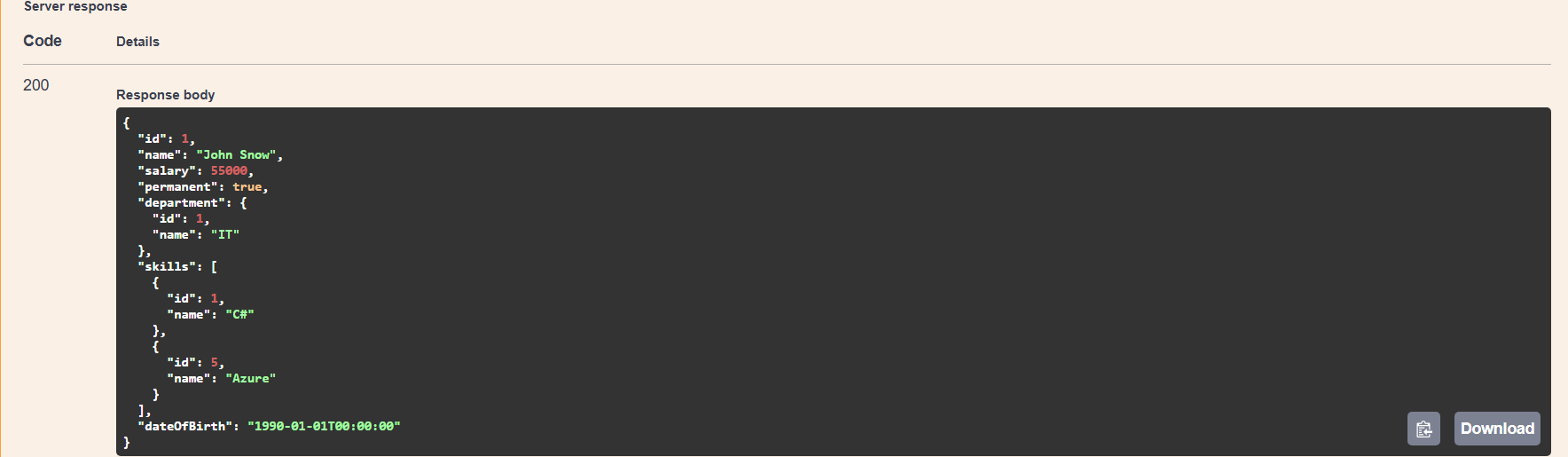
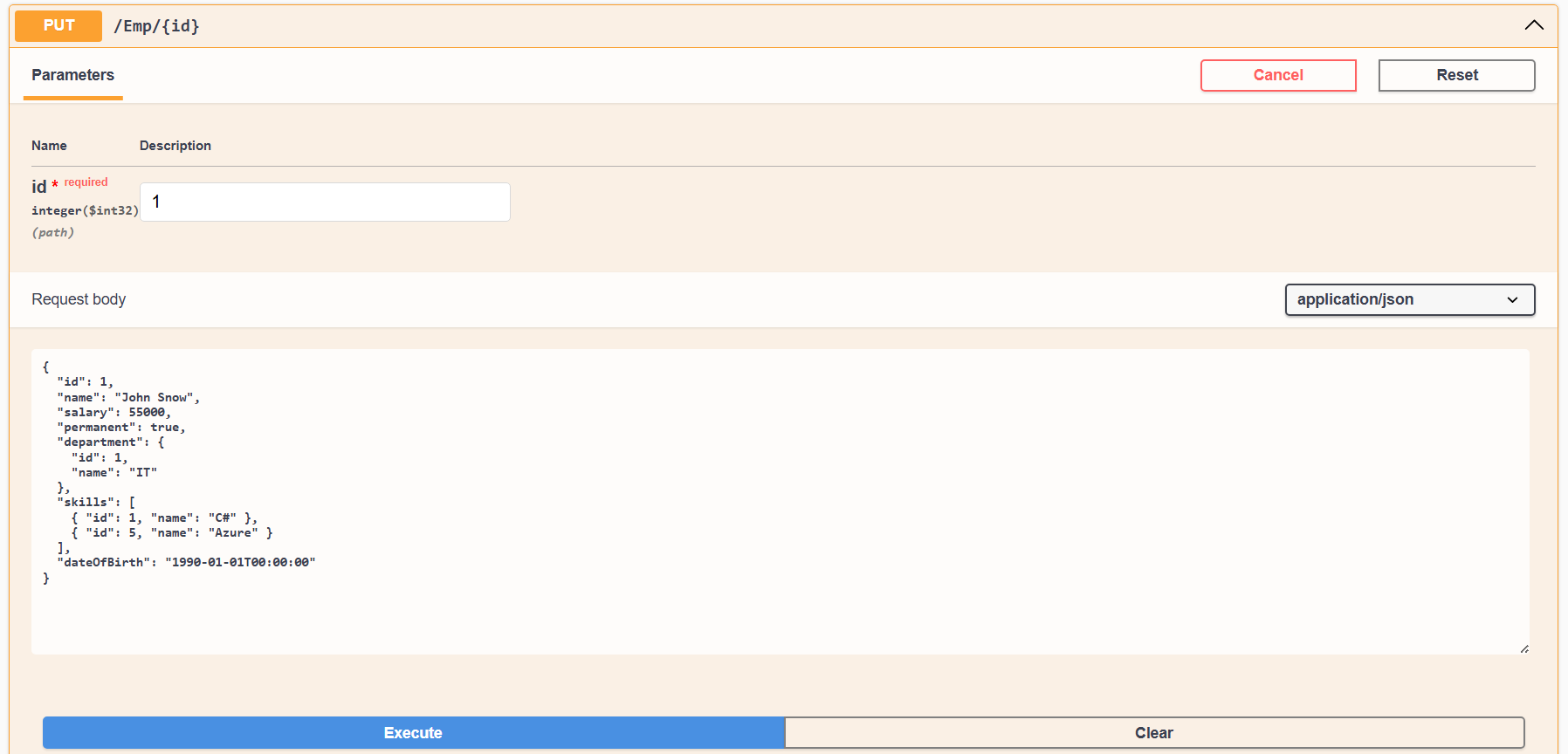
public interface IEmployeeRepository  
{  
 IEnumerable<Employee> GetAll();  
 Employee? GetById(int id);  
 void Add(Employee employee);  
 void Update(Employee employee);  
 void Delete(int id);  
 bool Exists(int id);  
}

public class EmployeeRepository : IEmployeeRepository  
{  
 private readonly List<Employee> \_employees;  
 public EmployeeRepository()  
 {  
 // Initialize with some sample data  
 \_employees = 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 Core" }  
 },  
 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 = "Recruitment" },  
 new Skill { Id = 4, Name = "Employee Relations" }  
 },  
 DateOfBirth = new DateTime(1985, 5, 15)  
 }  
 };  
 }  
 public IEnumerable<Employee> GetAll()  
 {  
 return \_employees;  
 }  
 public Employee? GetById(int id)  
 {  
 return \_employees.FirstOrDefault(e => e.Id == id);  
 }  
 public void Add(Employee employee)  
 {  
 employee.Id = \_employees.Max(e => e.Id) + 1;  
 \_employees.Add(employee);  
 }  
 public void Update(Employee employee)  
 {  
 var existingEmployee = GetById(employee.Id);  
 if (existingEmployee != null)  
 {  
 existingEmployee.Name = employee.Name;  
 existingEmployee.Salary = employee.Salary;  
 existingEmployee.Permanent = employee.Permanent;  
 existingEmployee.Department = employee.Department;  
 existingEmployee.Skills = employee.Skills;  
 existingEmployee.DateOfBirth = employee.DateOfBirth;  
 }  
 }  
 public void Delete(int id)  
 {  
 var employee = GetById(id);  
 if (employee != null)  
 {  
 \_employees.Remove(employee);  
 }  
 }  
 public bool Exists(int id)  
 {  
 return \_employees.Any(e => e.Id == id);  
 }  
}

[Route("Emp")]  
[ApiController]  
public class EmployeeController : ControllerBase  
{  
 private readonly IEmployeeRepository \_repository;  
 public EmployeeController(IEmployeeRepository repository)  
 {  
 \_repository = repository;  
 }  
   
 [HttpGet]  
 public ActionResult<IEnumerable<Employee>> GetAllEmployees()  
 {  
 var employees = \_repository.GetAll();  
 return Ok(employees);  
 }  
   
 [HttpGet("{id}")]  
 public ActionResult<Employee> GetEmployeeById(int id)  
 {  
 var employee = \_repository.GetById(id);  
 if (employee == null)  
 {  
 return NotFound();  
 }  
 return Ok(employee);  
 }  
   
 [HttpPost]  
 public ActionResult<Employee> CreateEmployee([FromBody] Employee employee)  
 {  
 if (!ModelState.IsValid)  
 {  
 return BadRequest(ModelState);  
 }  
 \_repository.Add(employee);  
 return CreatedAtAction(nameof(GetEmployeeById), new { id = employee.Id },  
employee);  
 }  
   
 [HttpPut("{id}")]  
 public IActionResult UpdateEmployee(int id, [FromBody] Employee employee)  
 {  
 if (id <= 0)  
 {  
 return BadRequest("Invalid employee id");  
 }  
   
 if (!\_repository.Exists(id))  
 {  
 return BadRequest("Invalid employee id");  
 }  
 if (id != employee.Id)  
 {  
 return BadRequest("Employee ID mismatch.");  
 }  
 \_repository.Update(employee);  
 return Ok(employee);  
 }  
   
 [HttpPatch("{id}")]  
 public IActionResult PatchEmployee(int id, [FromBody] Employee employee)  
 {  
 var existingEmployee = \_repository.GetById(id);  
 if (existingEmployee == null)  
 {  
 return NotFound();  
 }  
 // For simplicity, updating all fields. In real scenarios, use JSON Patch.  
 existingEmployee.Name = employee.Name ?? existingEmployee.Name;  
 existingEmployee.Salary = employee.Salary != 0 ? employee.Salary :  
existingEmployee.Salary;  
 existingEmployee.Permanent = employee.Permanent;  
 existingEmployee.Department = employee.Department ??  
existingEmployee.Department;  
 existingEmployee.Skills = employee.Skills ?? existingEmployee.Skills;  
 existingEmployee.DateOfBirth = employee.DateOfBirth != default ?  
employee.DateOfBirth : existingEmployee.DateOfBirth;  
   
 \_repository.Update(existingEmployee);  
 return Ok(employee);  
 }  
   
 [HttpDelete("{id}")]  
 public IActionResult DeleteEmployee(int id)  
 {  
 if (!\_repository.Exists(id))  
 {  
 return NotFound();  
 }  
 \_repository.Delete(id);  
 return NoContent();  
 }  
}

Output:  






**Hands-on 5:**

**Step 1:** Explain CORS enablement for Web API access for local application

CORS (Cross-Origin Resource Sharing) allows your Web API to be called from a different domain (e.g., frontend app).

Code:

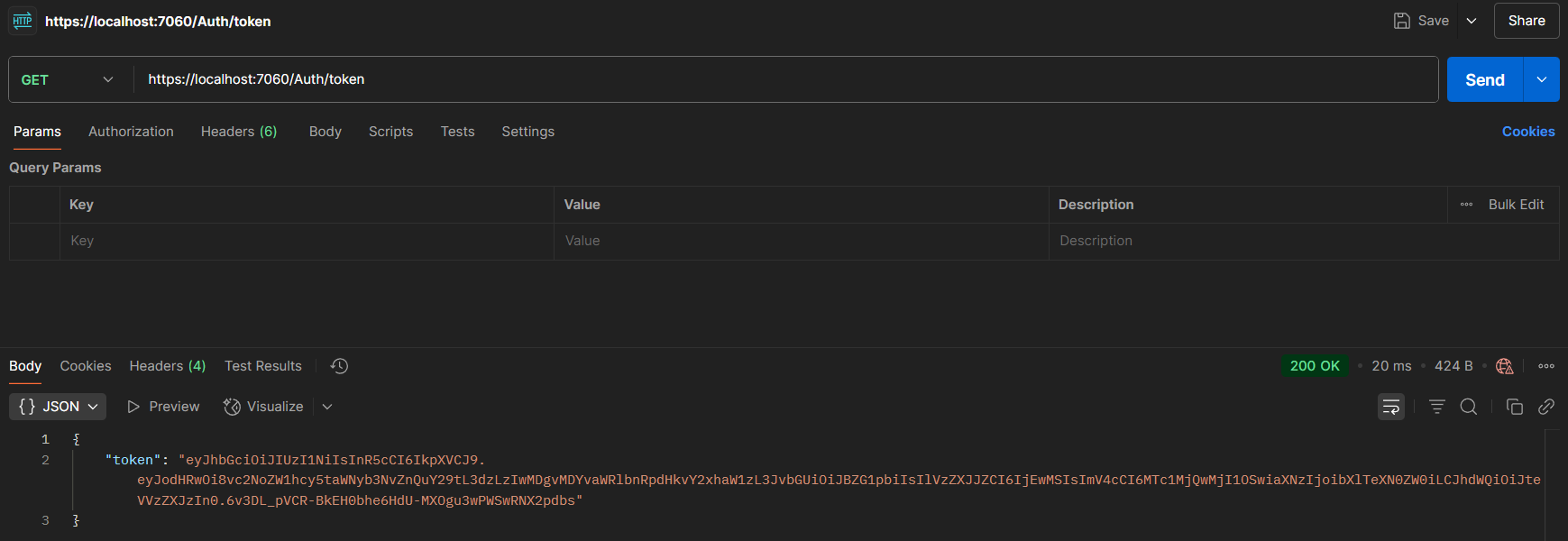
builder.Services.AddCors();  
  
app.UseCors(policy =>  
 policy.AllowAnyOrigin()  
 .AllowAnyMethod()  
 .AllowAnyHeader());

**Step 2:** JsonWebToken

Code:

public class Program  
{  
 public static void Main(string[] args)  
 {  
 var builder = WebApplication.CreateBuilder(args);  
   
 // Add services to the container.  
   
 builder.Services.AddControllers();  
 // Learn more about configuring Swagger/OpenAPI at  
https://aka.ms/aspnetcore/swashbuckle  
 builder.Services.AddEndpointsApiExplorer();  
 builder.Services.AddSingleton<IEmployeeRepository, EmployeeRepository>();  
 builder.Services.AddScoped<CustomAuthFilter>();  
 builder.Services.AddScoped<CustomExceptionFilter>();  
 builder.Services.AddCors();  
 builder.Services.AddSwaggerGen(c =>  
 {  
 c.SwaggerDoc("v1", new()  
 {  
 Title = "Swagger Demo",  
 Version = "v1",  
 Description = "TBD",  
 Contact = new() { Name = "Abhranil Dasgupta", Email = "abhranilnxt@gmail.com" }  
 });  
 });  
 var securityKey = "mysuperdupersecrettokenkey123456!";  
 var symmetricSecurityKey = new  
SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));  
   
 builder.Services.AddAuthentication(options =>  
 {  
 options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;  
 options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;  
 options.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;  
 })  
 .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme,options =>  
 {  
 options.TokenValidationParameters = new TokenValidationParameters  
 {  
 //what to validate   
   
 ValidateIssuer = true,  
   
 ValidateAudience = true,  
   
 ValidateLifetime = true,  
   
 ValidateIssuerSigningKey = true,  
   
 //setup validate data   
   
 ValidIssuer = "mySystem",  
   
 ValidAudience = "myUsers",  
   
 IssuerSigningKey = symmetricSecurityKey  
 };  
 });  
   
   
 var app = builder.Build();  
   
 // Configure the HTTP request pipeline.  
 if (app.Environment.IsDevelopment())  
 {  
 app.UseSwagger();  
 app.UseSwaggerUI(c =>  
 {  
 c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");  
 });  
 app.UseCors(policy =>  
 policy.AllowAnyOrigin()  
 .AllowAnyMethod()  
 .AllowAnyHeader());  
   
 }  
   
 app.UseAuthentication();  
   
 app.UseHttpsRedirection();  
   
 app.UseAuthorization();  
   
 app.MapControllers();  
   
 app.Run();  
 }  
}

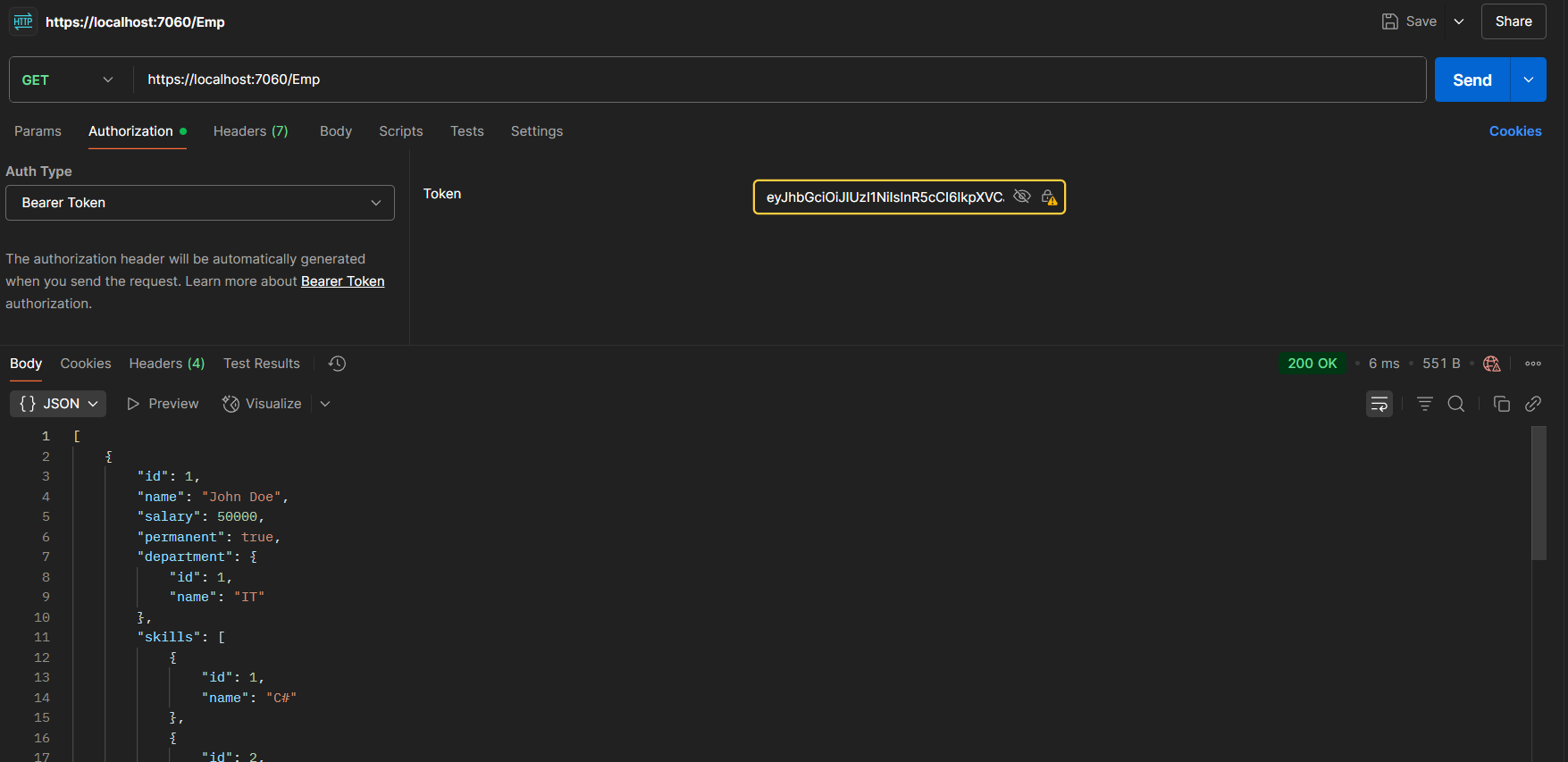
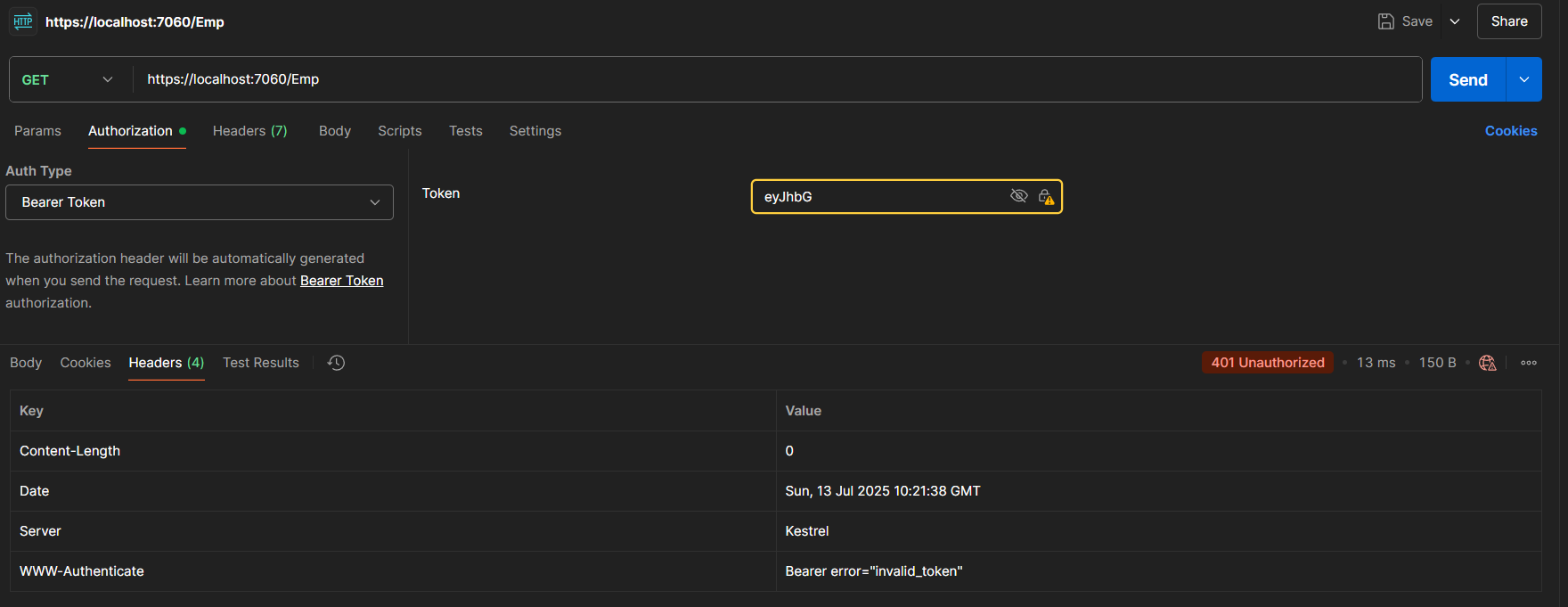
[ApiController]  
[Route("[controller]")]  
[AllowAnonymous]  
public class AuthController : ControllerBase  
{  
 [HttpGet("token")]  
 public IActionResult GetToken()  
 {  
 // Generate token for userId 101 and role Admin  
 string token = GenerateJSONWebToken(101, "Admin");  
 return Ok(new { token });  
 }  
   
 private string GenerateJSONWebToken(int userId, string userRole)  
 {  
 var securityKey = new  
SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecrettokenkey123456!")  
;  
 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);  
 }  
}

Output:  


**Step 3:** Use the JWT generated thru the AuthController to be used in POSTMAN request.

Code:

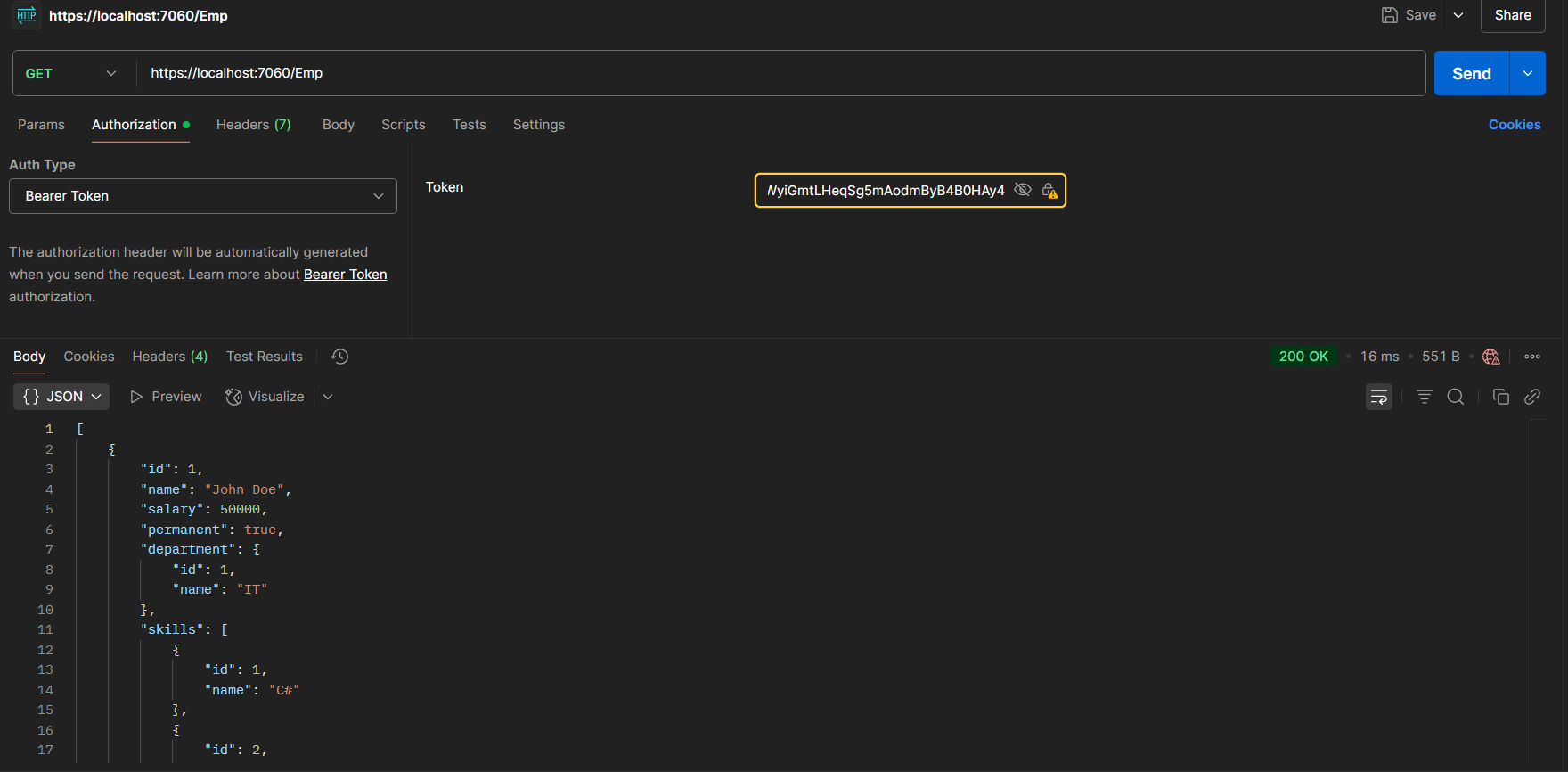
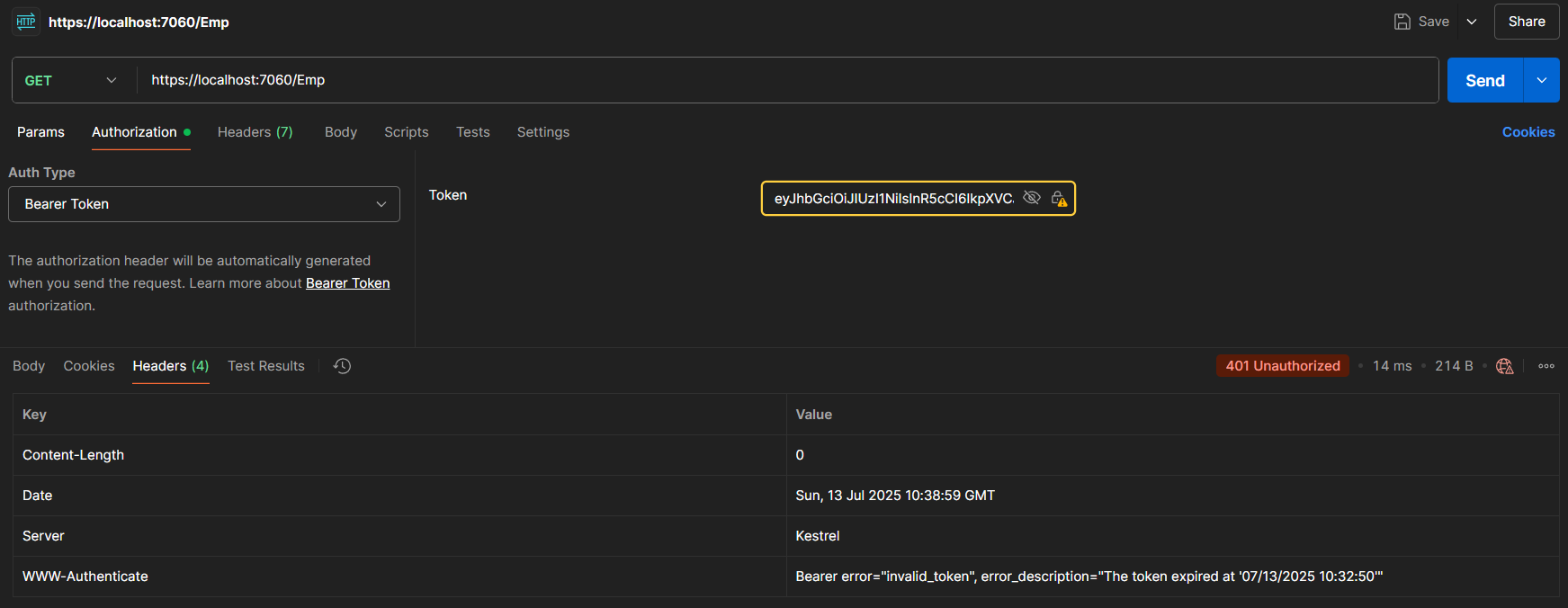
[Route("Emp")]  
[ApiController]  
[TypeFilter(typeof(CustomExceptionFilter))]  
//[ServiceFilter(typeof(CustomAuthFilter))]  
*[Authorize]*  
public class EmployeeController : ControllerBase  
{  
 //Rest of the boilerplate code  
}

Output:  
  


**Step 4:** Check for JWT expiration

Code:

[ApiController]  
[Route("[controller]")]  
[AllowAnonymous]  
public class AuthController : ControllerBase  
{  
 [HttpGet("token")]  
 public IActionResult GetToken()  
 {  
 // Generate token for userId 101 and role Admin  
 string token = GenerateJSONWebToken(101, "Admin");  
 return Ok(new { token });  
 }  
   
 private string GenerateJSONWebToken(int userId, string userRole)  
 {  
 var securityKey = new  
SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecrettokenkey123456!")  
;  
 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(2),  
 signingCredentials: credentials);  
   
 return new JwtSecurityTokenHandler().WriteToken(token);  
 }  
}

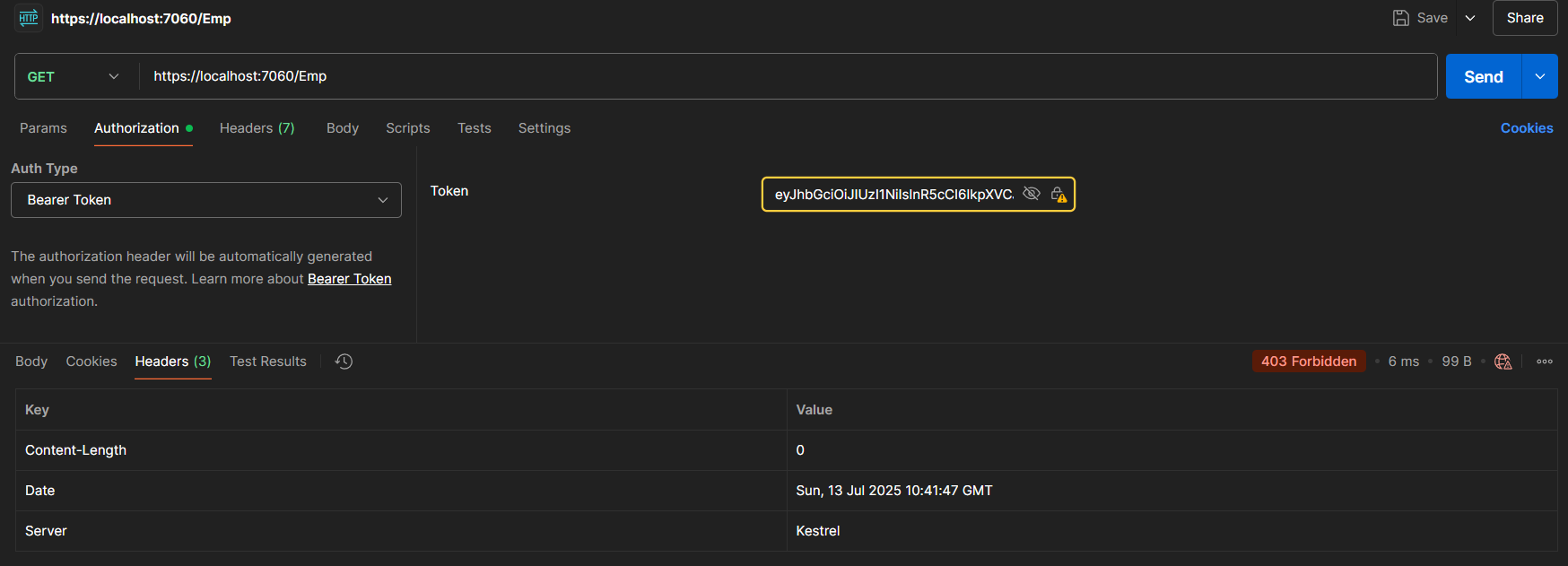
Output:  
  


**Step 5:** Add the roles to be authorized in the Authorize attribute.

Code:

[Route("Emp")]  
[ApiController]  
[TypeFilter(typeof(CustomExceptionFilter))]  
//[ServiceFilter(typeof(CustomAuthFilter))]  
*[Authorize (Roles = "POC")]*  
public class EmployeeController : ControllerBase  
{  
 //Rest of the boilerplate code  
}

Output:



Code:

[Route("Emp")]  
[ApiController]  
[TypeFilter(typeof(CustomExceptionFilter))]  
//[ServiceFilter(typeof(CustomAuthFilter))]  
//[Authorize(Roles = "POC")]  
*[Authorize (Roles = "Admin, POC")]*  
public class EmployeeController : ControllerBase  
{  
 //Rest of the boilerplate code  
}

Output:  
  
