**WEBAPI 05**

**Aim:**

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.

**Department.cs:**

namespace EmployeeJwtApi.Models

{

    public class Department

    {

        public int Id { get; set; }

        public string? Name { get; set; }

        public static implicit operator Department(string v)

        {

            throw new NotImplementedException();

        }

    }

}

**Skills.cs:**

namespace EmployeeJwtApi.Models

{

    public class Skill

    {

        public int Id { get; set; }

        public string? Name { get; set; }

    }

}

**Employee.cs:**

using System;

using System.Collections.Generic;

namespace EmployeeJwtApi.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 string? Department { get; set; }

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

        public DateTime DateOfBirth { get; set; }

    }

}

**EmployeeControllers.cs:**

using EmployeeJwtApi.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace EmployeeJwtApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class EmployeeController : ControllerBase

    {

        // Hardcoded employee list

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

        {

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

            new Employee { Id = 2, Name = "Bob", Department = "IT", Salary = 70000 },

            new Employee { Id = 3, Name = "Charlie", Department = "Finance", Salary = 65000 }

        };

        // GET all employees

        [HttpGet]

        [AllowAnonymous]

        public ActionResult<IEnumerable<Employee>> GetEmployees()

        {

            return Ok(employees);

        }

        // GET employee by ID

        [HttpGet("{id}")]

        [AllowAnonymous]

        public ActionResult<Employee> GetEmployeeById(int id)

        {

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

            if (emp == null)

                return NotFound($"Employee with ID {id} not found");

            return Ok(emp);

        }

        // POST - Add employee

        [HttpPost]

        [AllowAnonymous]

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

        {

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

            employees.Add(newEmployee);

            return CreatedAtAction(nameof(GetEmployeeById), new { id = newEmployee.Id }, newEmployee);

        }

        // PUT - Update employee

        [HttpPut("{id}")]

        [AllowAnonymous]

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

        {

            if (id <= 0)

                return BadRequest("Invalid employee id");

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

            if (emp == null)

                return BadRequest("Invalid employee id");

            emp.Name = updatedEmployee.Name;

            emp.Department = updatedEmployee.Department;

            emp.Salary = updatedEmployee.Salary;

            return Ok(emp);

        }

        // DELETE - Remove employee

        [HttpDelete("{id}")]

        [AllowAnonymous]

        public ActionResult DeleteEmployee(int id)

        {

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

            if (emp == null)

                return NotFound("Employee not found");

            employees.Remove(emp);

            return Ok($"Employee {id} deleted");

        }

    }

}

**AuthControllers.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace EmployeeJwtApi.Controllers

{

    [ApiController]

    [AllowAnonymous]

    [Route("api/[controller]")]

    public class AuthController : ControllerBase

    {

        // Keep the same key/issuer/audience as in Startup.cs

        private const string SecurityKey = "mysuperdupersecret";

        private const string Issuer = "mySystem";

        private const string Audience = "myUsers";

        // GET: api/Auth/GenerateToken?userId=1&role=Admin&minutes=10

        [HttpGet("GenerateToken")]

        public IActionResult GenerateToken(int userId = 1, string role = "Admin", int minutes = 10)

        {

            var token = GenerateJSONWebToken(userId, role, minutes);

            return Ok(new { token });

        }

        private string GenerateJSONWebToken(int userId, string userRole, int minutesValid)

        {

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

            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: Issuer,

                audience: Audience,

                claims: claims,

                expires: DateTime.UtcNow.AddMinutes(minutesValid),

                signingCredentials: credentials);

            return new JwtSecurityTokenHandler().WriteToken(token);

        }

    }

    internal class AllowAnonymousAttribute : Attribute

    {

    }

}

**Startup.cs:**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System;

using System.Text;

using Microsoft.AspNetCore.Authentication.JwtBearer;

namespace EmployeeJwtApi

{

    public class Startup

    {

        // NOTE: keep this key in sync with AuthController GenerateJSONWebToken

        private const string SecurityKey = "mysuperdupersecret";

        public void ConfigureServices(IServiceCollection services)

        {

            services.AddControllers();

            // JWT setup

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

            services.AddAuthentication(x =>

            {

                x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

                x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

            })

         .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, options =>

            {

                options.TokenValidationParameters = new TokenValidationParameters

                {

                    ValidateIssuer = true,

                    ValidateAudience = true,

                    ValidateLifetime = true,

                    ValidateIssuerSigningKey = true,

                    ValidIssuer = "mySystem",

                    ValidAudience = "myUsers",

                    IssuerSigningKey = symmetricSecurityKey,

                    // remove default clock skew so expiry checks are strict

                    ClockSkew = TimeSpan.Zero

                };

            });

            services.AddAuthorization();

            // Swagger with JWT auth UI

            services.AddSwaggerGen(c =>

            {

                c.SwaggerDoc("v1", new OpenApiInfo { Title = "Employee JWT API", Version = "v1" });

                // JWT Bearer security definition for Swagger

                var securityScheme = new OpenApiSecurityScheme

                {

                    Name = "Authorization",

                    Description = "Enter 'Bearer {token}'",

                    In = ParameterLocation.Header,

                    Type = SecuritySchemeType.Http,

                    Scheme = "bearer",

                    BearerFormat = "JWT",

                    Reference = new OpenApiReference { Type = ReferenceType.SecurityScheme, Id = "Bearer" }

                };

                c.AddSecurityDefinition("Bearer", securityScheme);

                c.AddSecurityRequirement(new OpenApiSecurityRequirement

                {

                    { securityScheme, Array.Empty<string>() }

                });

            });

        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

        {

            if (env.IsDevelopment())

            {

                app.UseDeveloperExceptionPage();

            }

            // Serve swagger UI at app root (http://localhost:xxxx/)

            app.UseSwagger();

            app.UseSwaggerUI(c =>

            {

                c.SwaggerEndpoint("/swagger/v1/swagger.json", "Employee JWT API v1");

                c.RoutePrefix = string.Empty;

            });

            app.UseRouting();

            // Authentication must come BEFORE Authorization

            app.UseAuthentication();

            app.UseAuthorization();

            app.UseEndpoints(endpoints => { endpoints.MapControllers(); });

        }

    }

}

**Program.cs:**

var builder = WebApplication.CreateBuilder(args);

// Add services to container

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Enable Swagger in Development mode

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI();

}

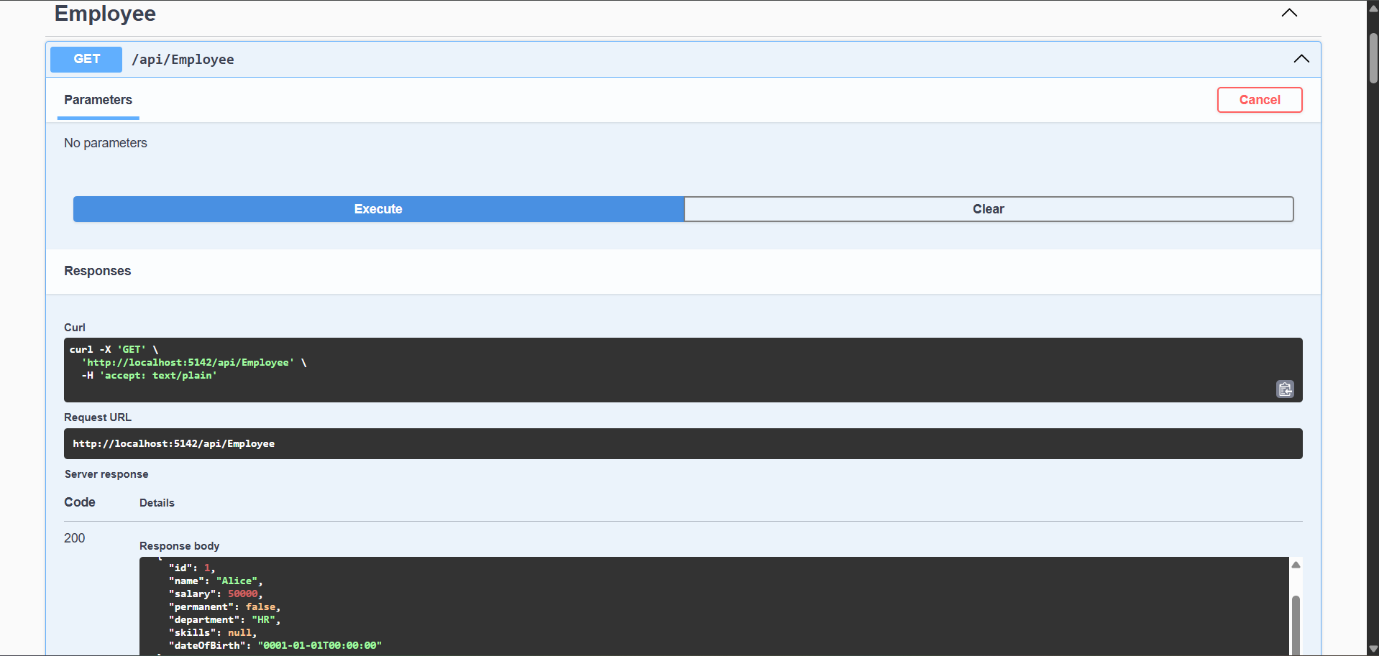
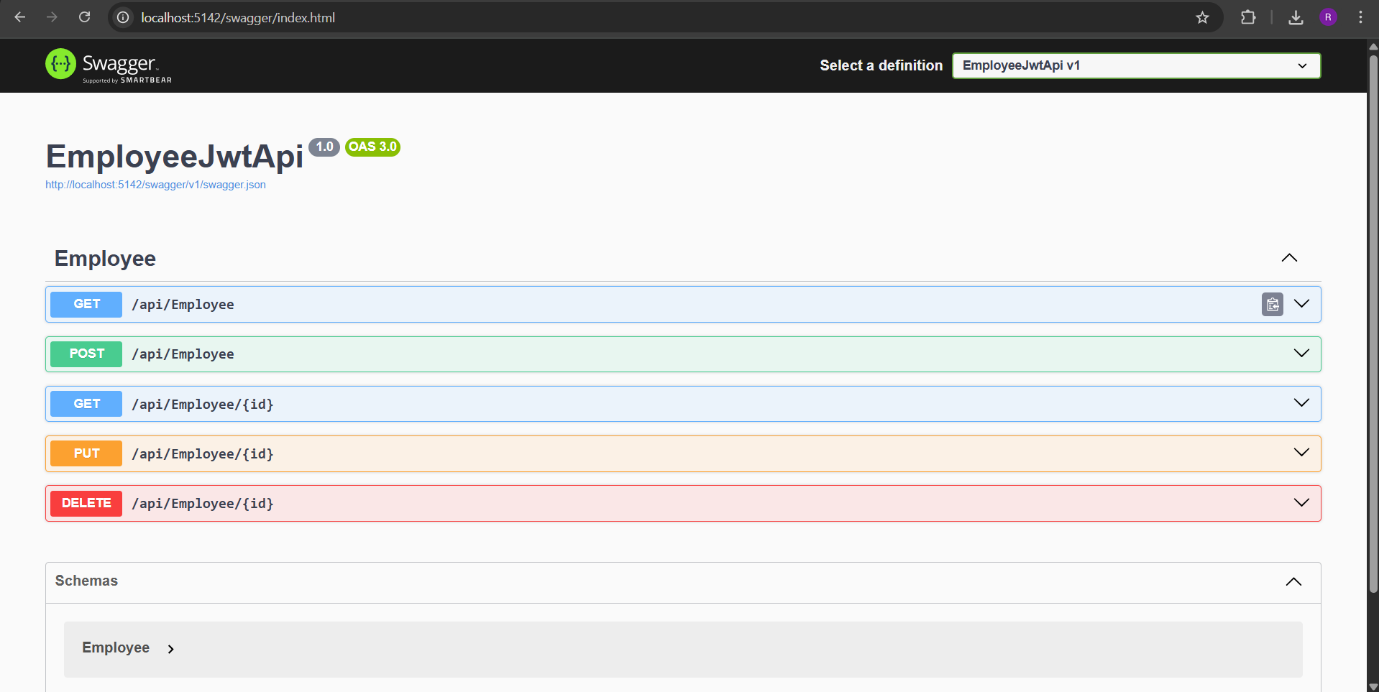
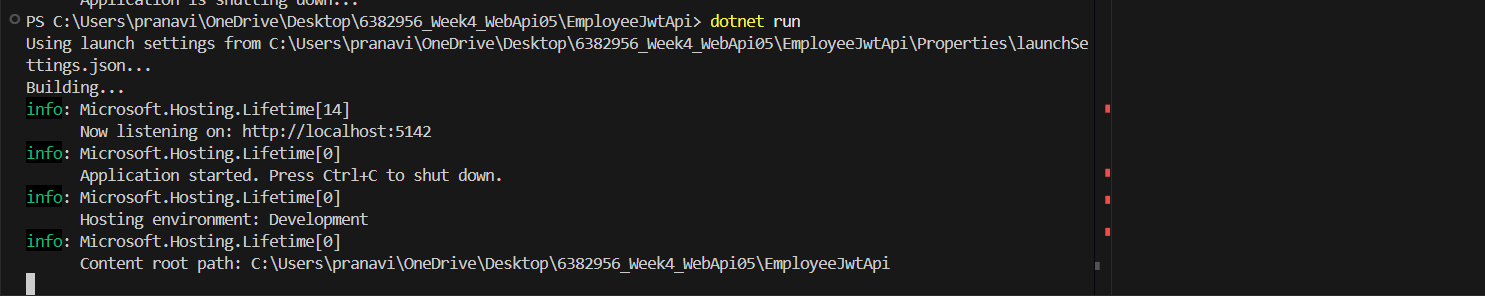
app.UseHttpsRedirection();

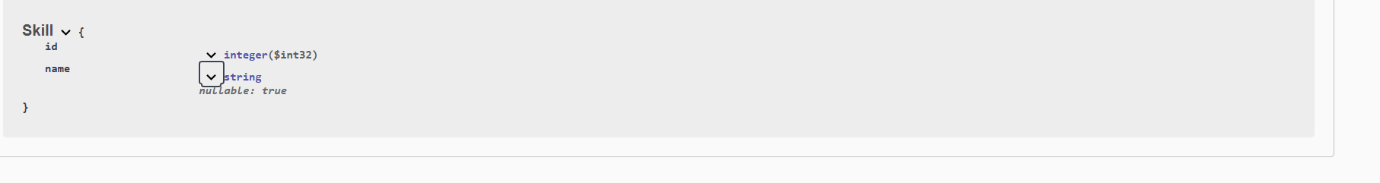
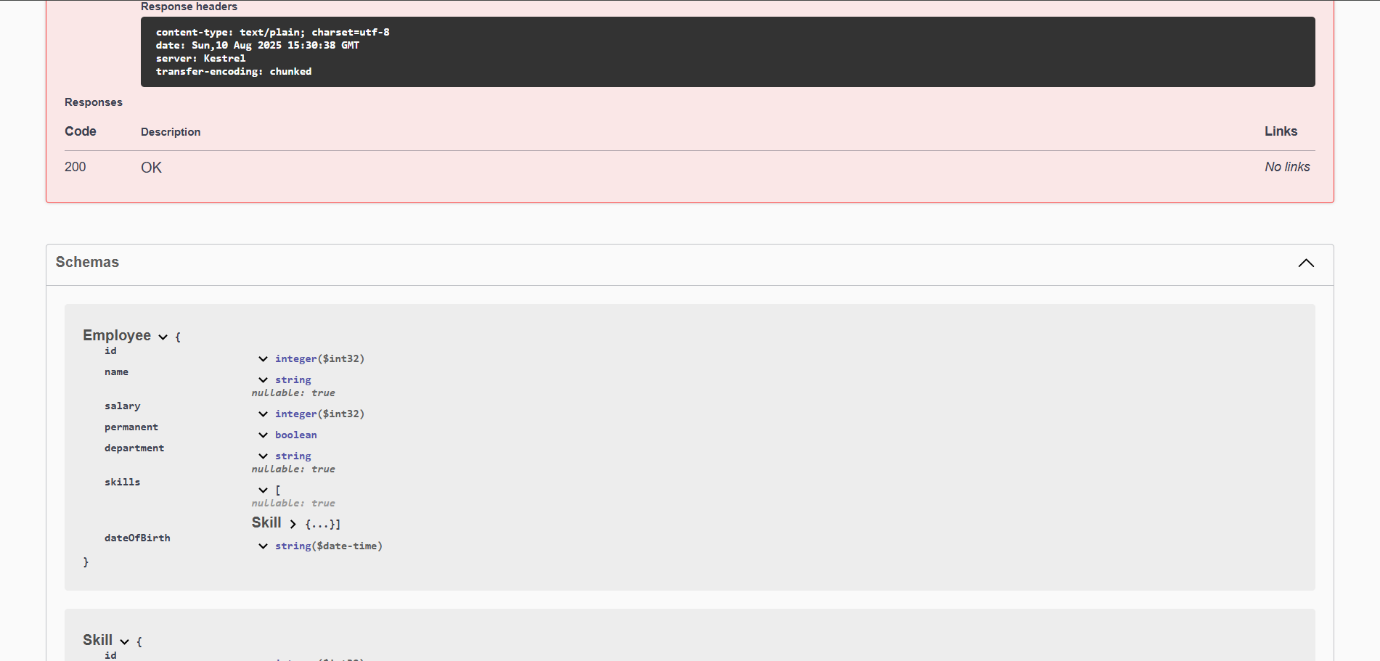
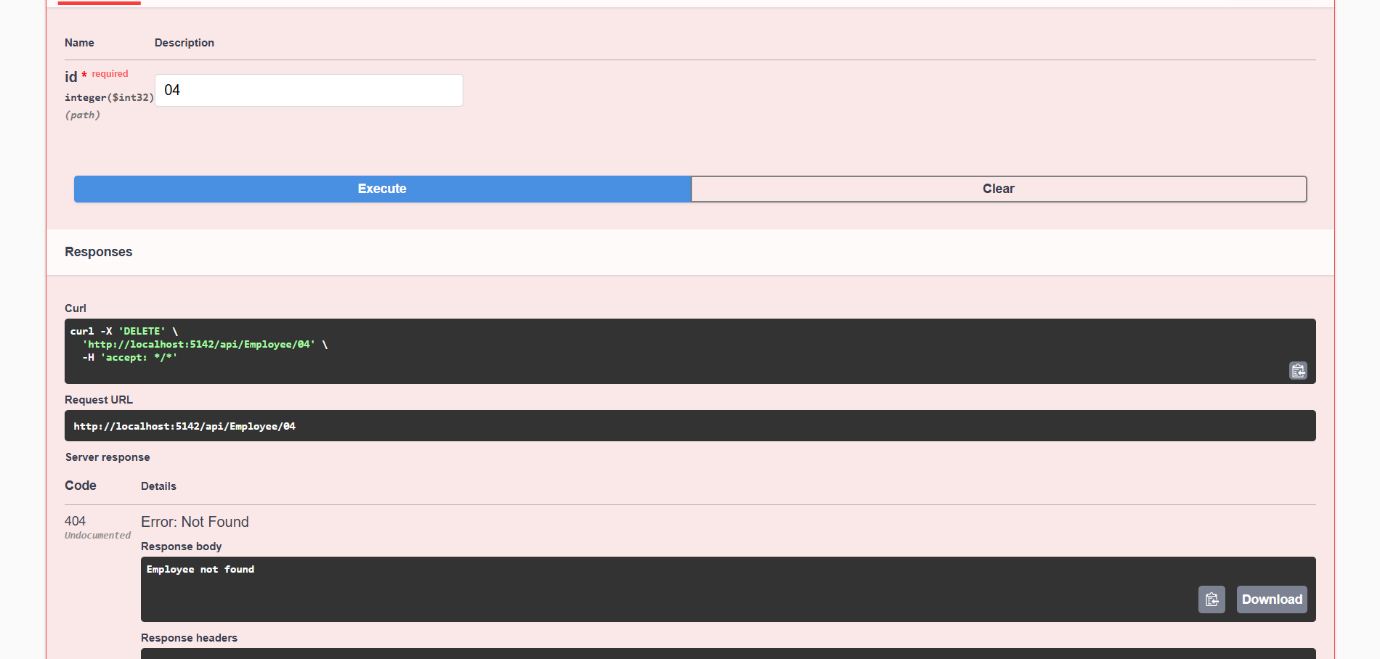
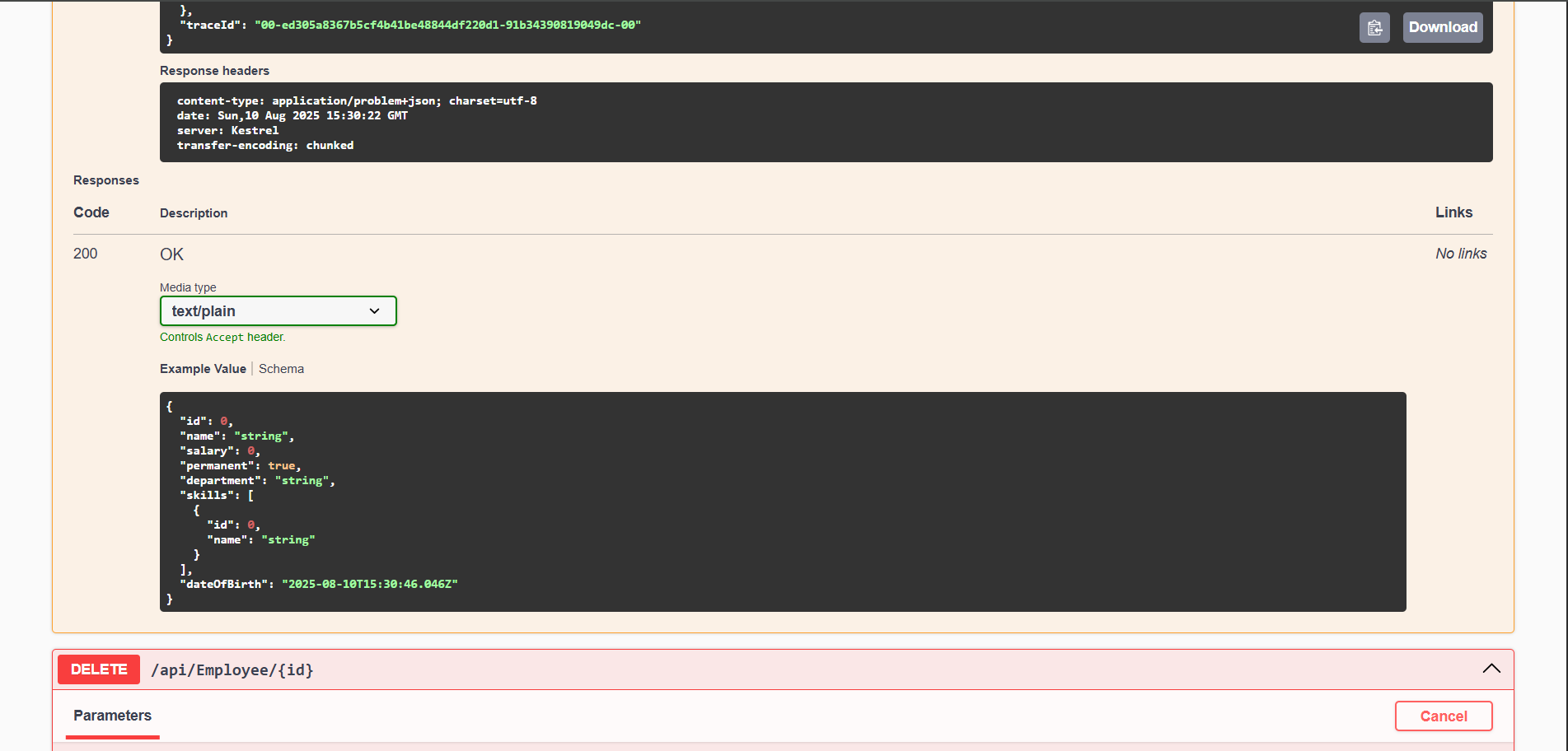
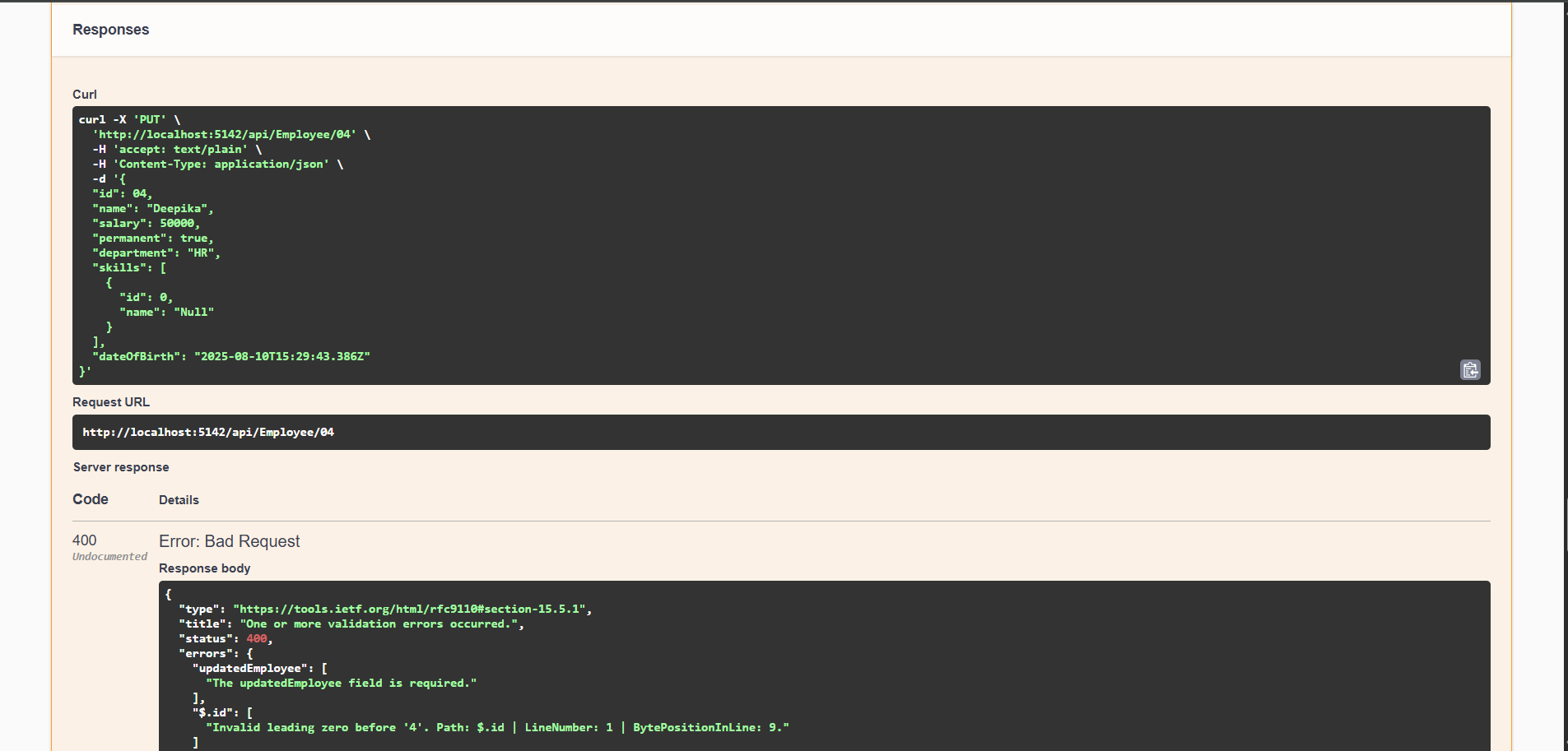
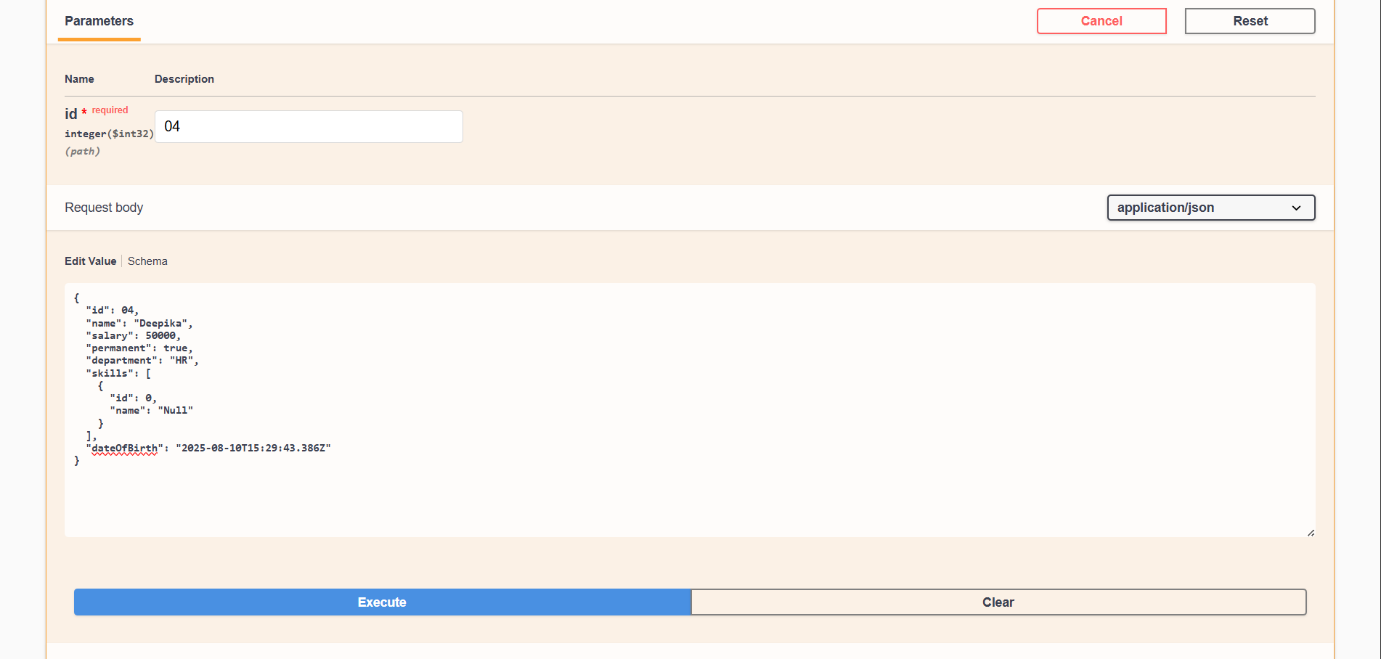
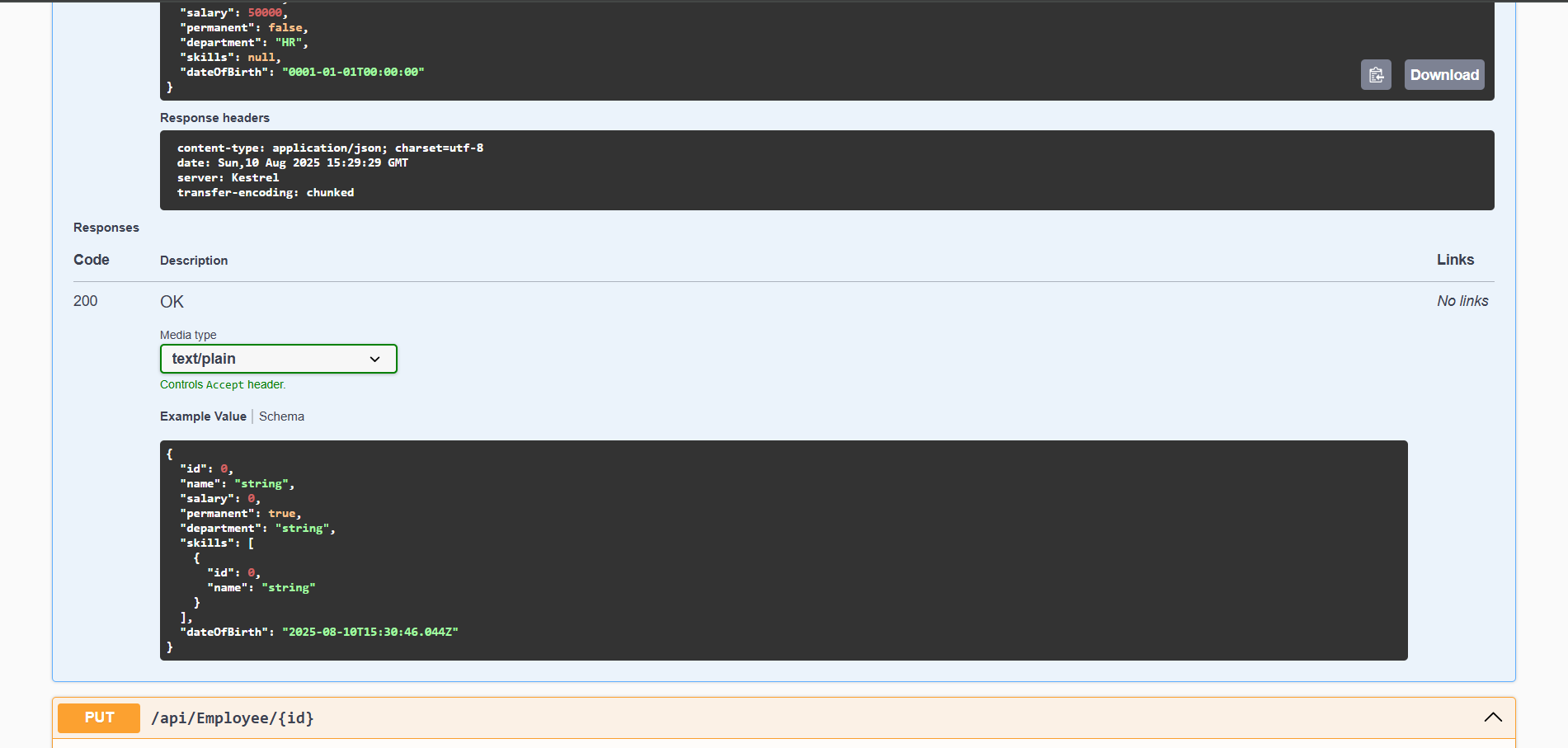
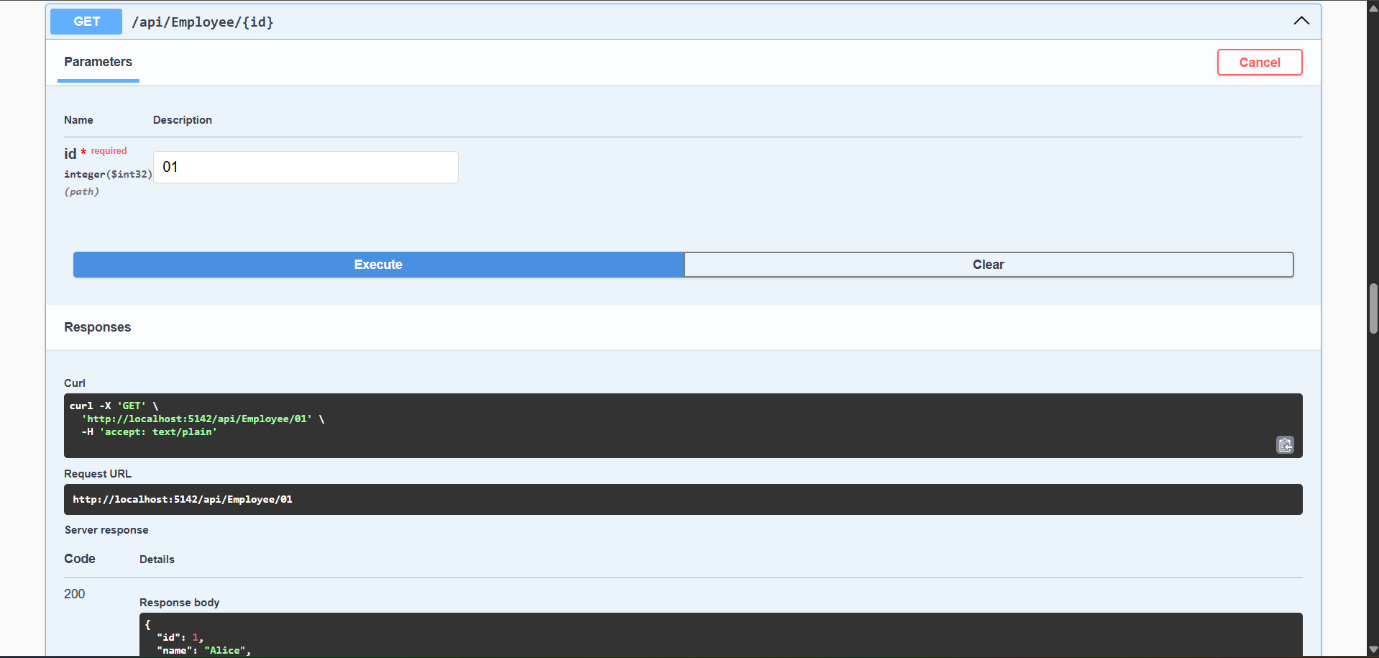
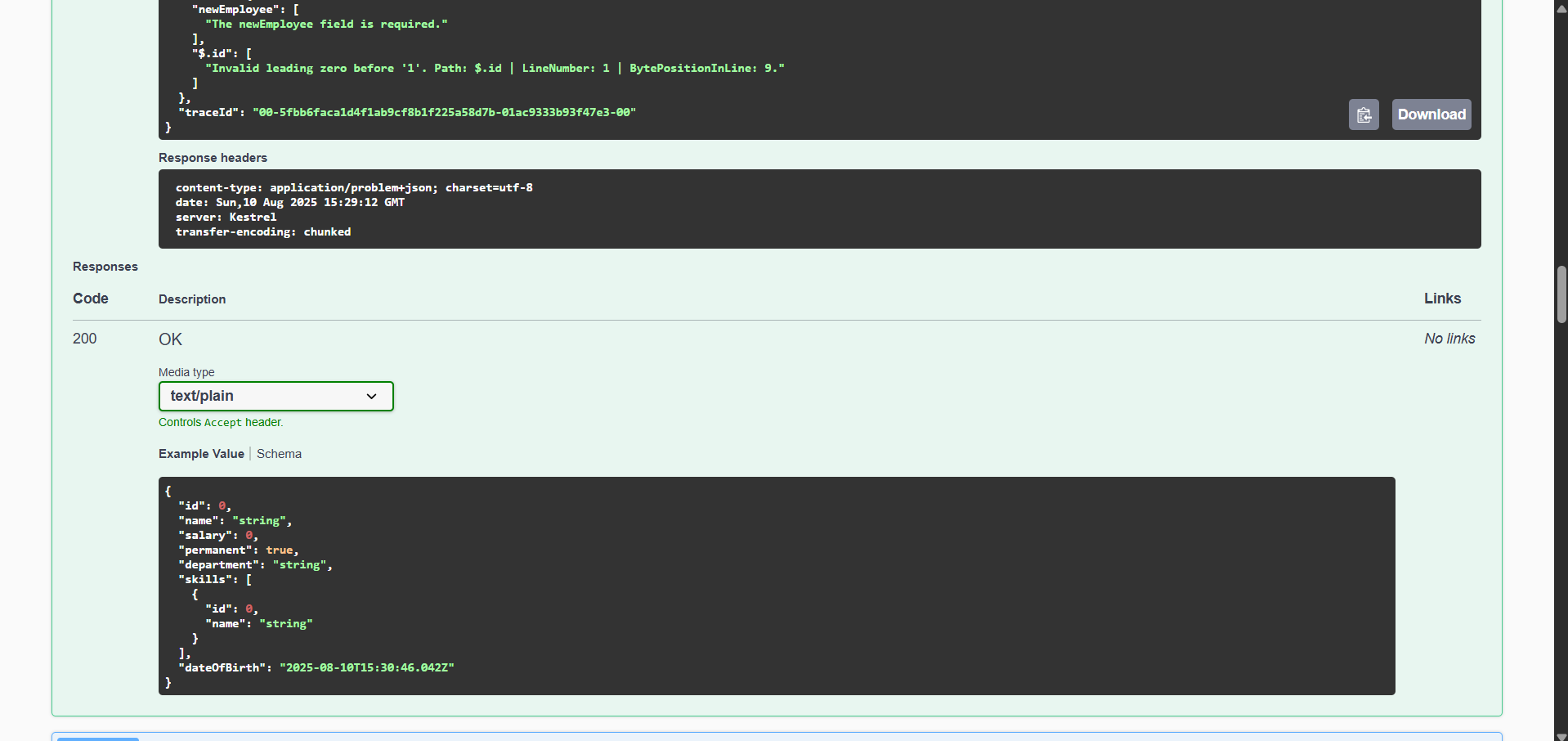
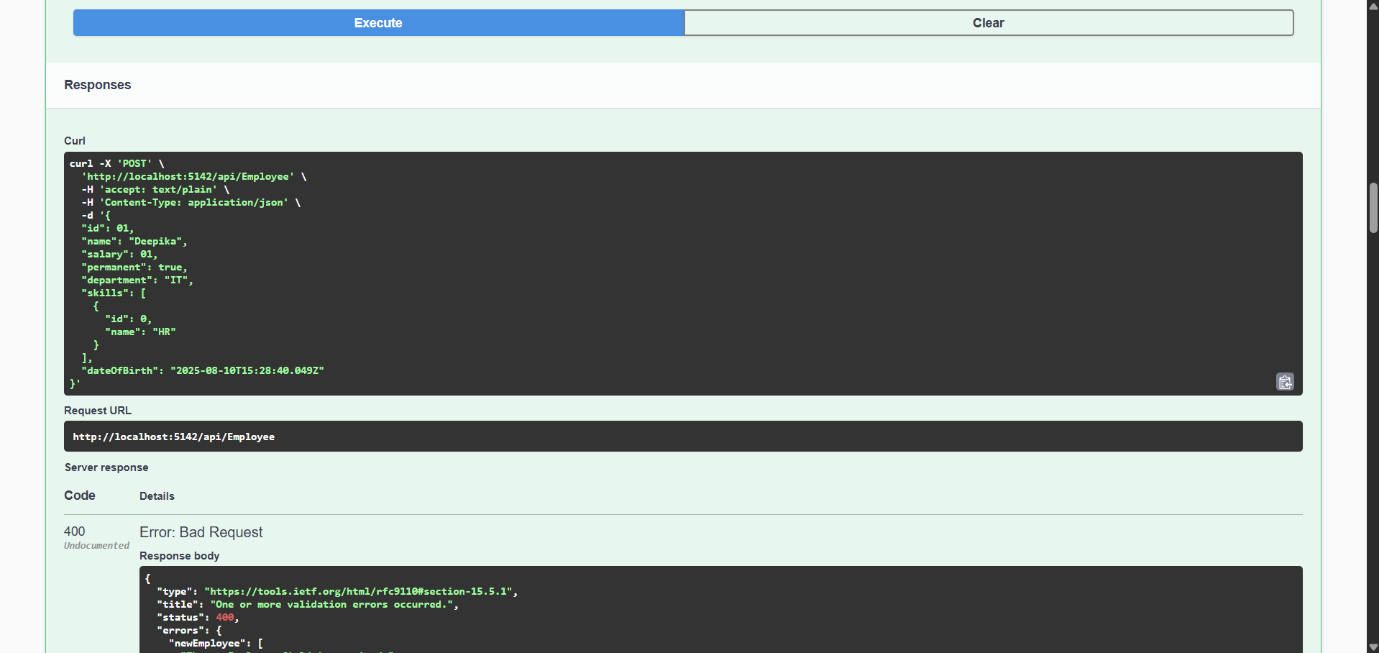
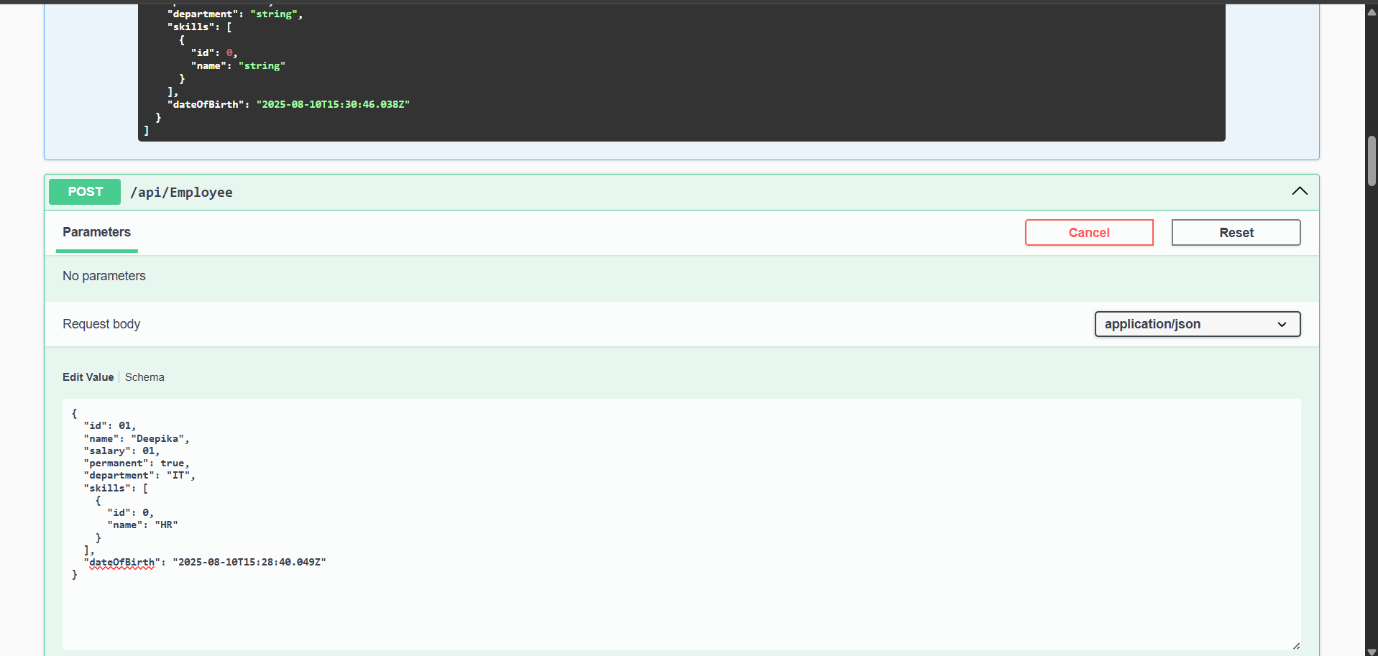
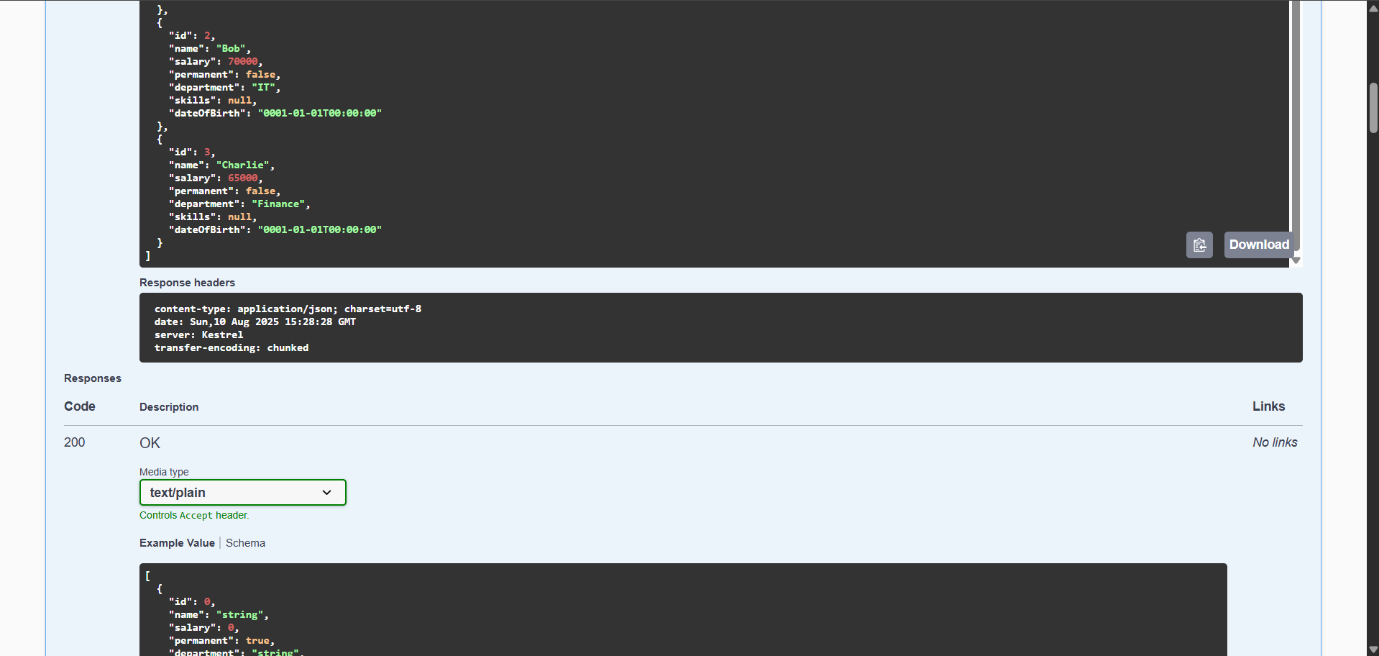
app.UseAuthorization();

app.MapControllers();

app.Run();

**Output:**

****

****