**Q1)**

**CODE1**

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();

**CODE 2**  
using Microsoft.AspNetCore.Mvc;

namespace FirstWebAPI.Controllers

{

[ApiController]

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

public class ValuesController : ControllerBase

{

// In-memory storage for demonstration

private static List<string> \_values = new List<string>

{

"Value1",

"Value2",

"Value3"

};

// GET: api/values

[HttpGet]

public IActionResult Get()

{

try

{

return Ok(new

{

message = "Success",

data = \_values,

count = \_values.Count

});

}

catch (Exception ex)

{

return StatusCode(500, new { message = "Internal server error", error = ex.Message });

}

}

// GET: api/values/5

[HttpGet("{id}")]

public IActionResult Get(int id)

{

try

{

if (id < 0 || id >= \_values.Count)

{

return NotFound(new { message = $"Value with id {id} not found" });

}

return Ok(new

{

message = "Success",

data = \_values[id],

index = id

});

}

catch (Exception ex)

{

return StatusCode(500, new { message = "Internal server error", error = ex.Message });

}

}

// POST: api/values

[HttpPost]

public IActionResult Post([FromBody] string value)

{

try

{

if (string.IsNullOrEmpty(value))

{

return BadRequest(new { message = "Value cannot be empty or null" });

}

\_values.Add(value);

var newIndex = \_values.Count - 1;

return Created($"api/values/{newIndex}", new

{

message = "Value created successfully",

data = value,

index = newIndex

});

}

catch (Exception ex)

{

return StatusCode(500, new { message = "Internal server error", error = ex.Message });

}

}

// PUT: api/values/5

[HttpPut("{id}")]

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

{

try

{

if (id < 0 || id >= \_values.Count)

{

return NotFound(new { message = $"Value with id {id} not found" });

}

if (string.IsNullOrEmpty(value))

{

return BadRequest(new { message = "Value cannot be empty or null" });

}

var oldValue = \_values[id];

\_values[id] = value;

return Ok(new

{

message = "Value updated successfully",

oldValue = oldValue,

newValue = value,

index = id

});

}

catch (Exception ex)

{

return StatusCode(500, new { message = "Internal server error", error = ex.Message });

}

}

// DELETE: api/values/5

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

try

{

if (id < 0 || id >= \_values.Count)

{

return NotFound(new { message = $"Value with id {id} not found" });

}

var deletedValue = \_values[id];

\_values.RemoveAt(id);

return Ok(new

{

message = "Value deleted successfully",

deletedValue = deletedValue,

remainingCount = \_values.Count

});

}

catch (Exception ex)

{

return StatusCode(500, new { message = "Internal server error", error = ex.Message });

}

}

// GET: api/values/count

[HttpGet("count")]

public IActionResult GetCount()

{

return Ok(new

{

message = "Total count retrieved",

count = \_values.Count

});

}

// DELETE: api/values (clear all)

[HttpDelete]

public IActionResult DeleteAll()

{

try

{

var count = \_values.Count;

\_values.Clear();

return Ok(new

{

message = "All values cleared",

deletedCount = count

});

}

catch (Exception ex)

{

return StatusCode(500, new { message = "Internal server error", error = ex.Message });

}

}

}

}

**CODE 3**

<?xml version="1.0" encoding="utf-8"?>

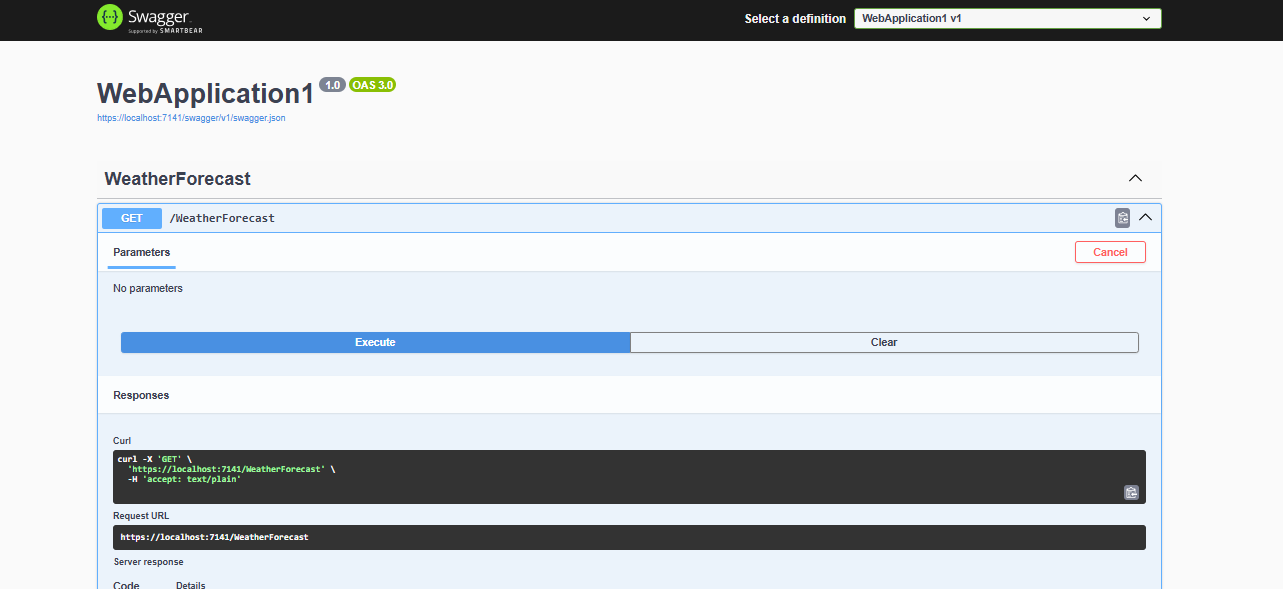
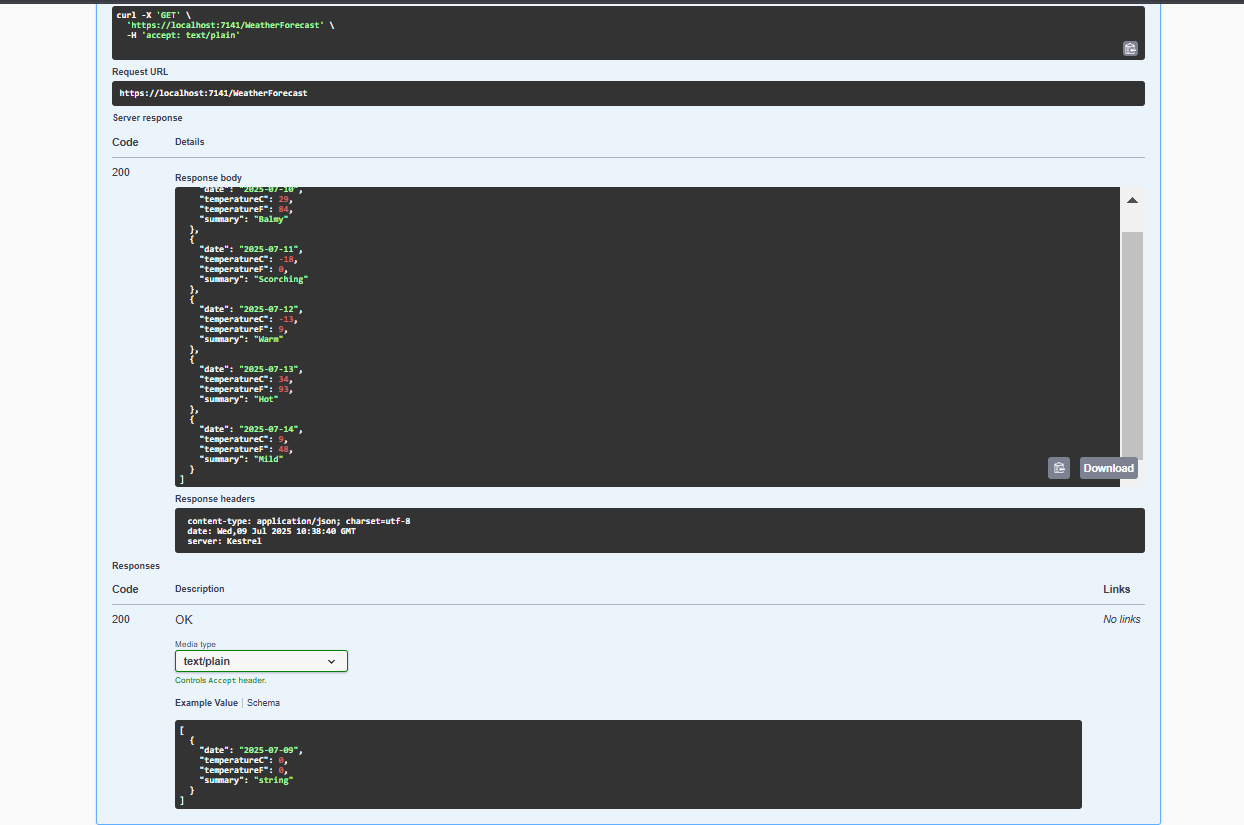
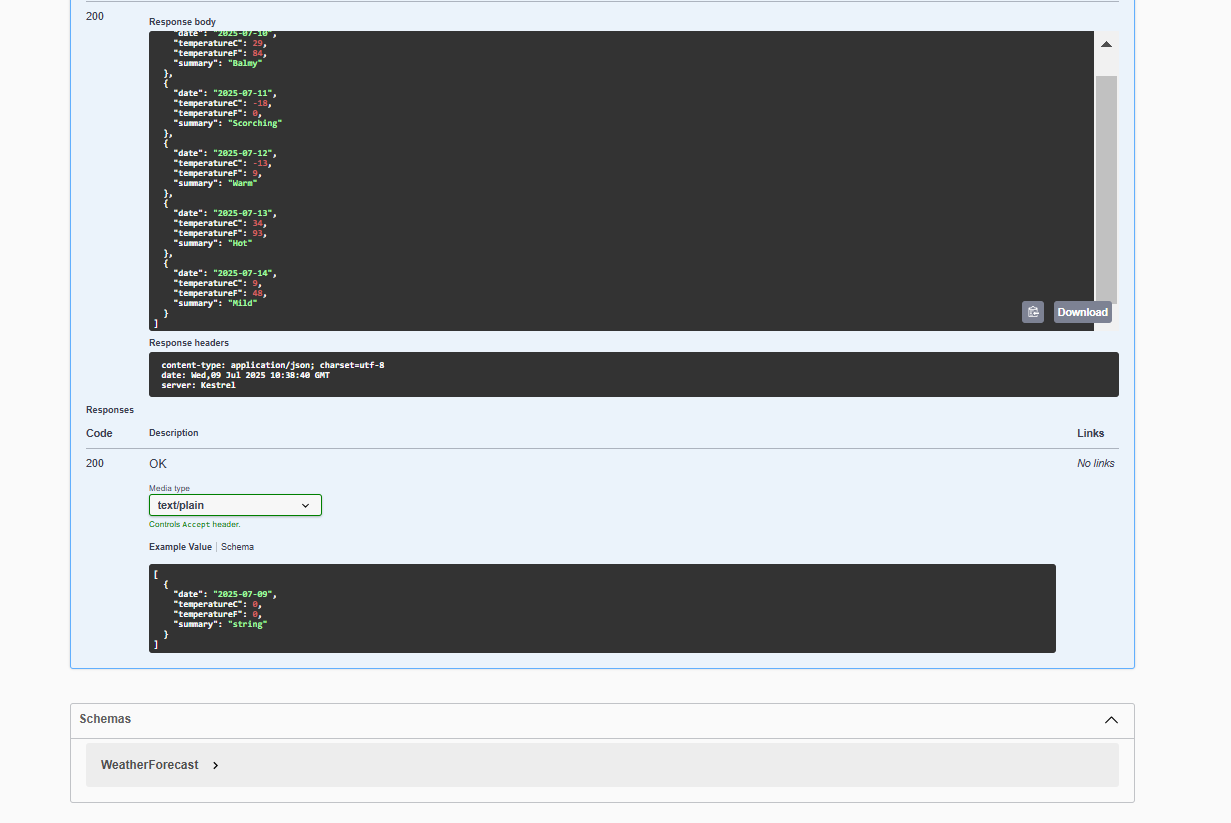
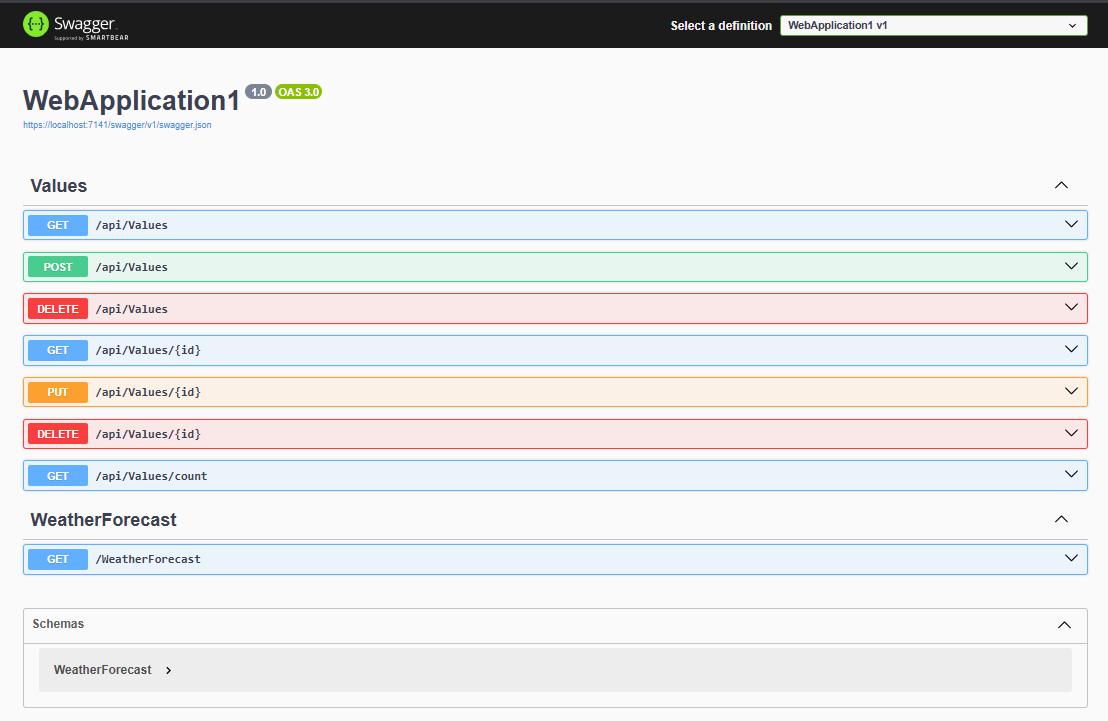
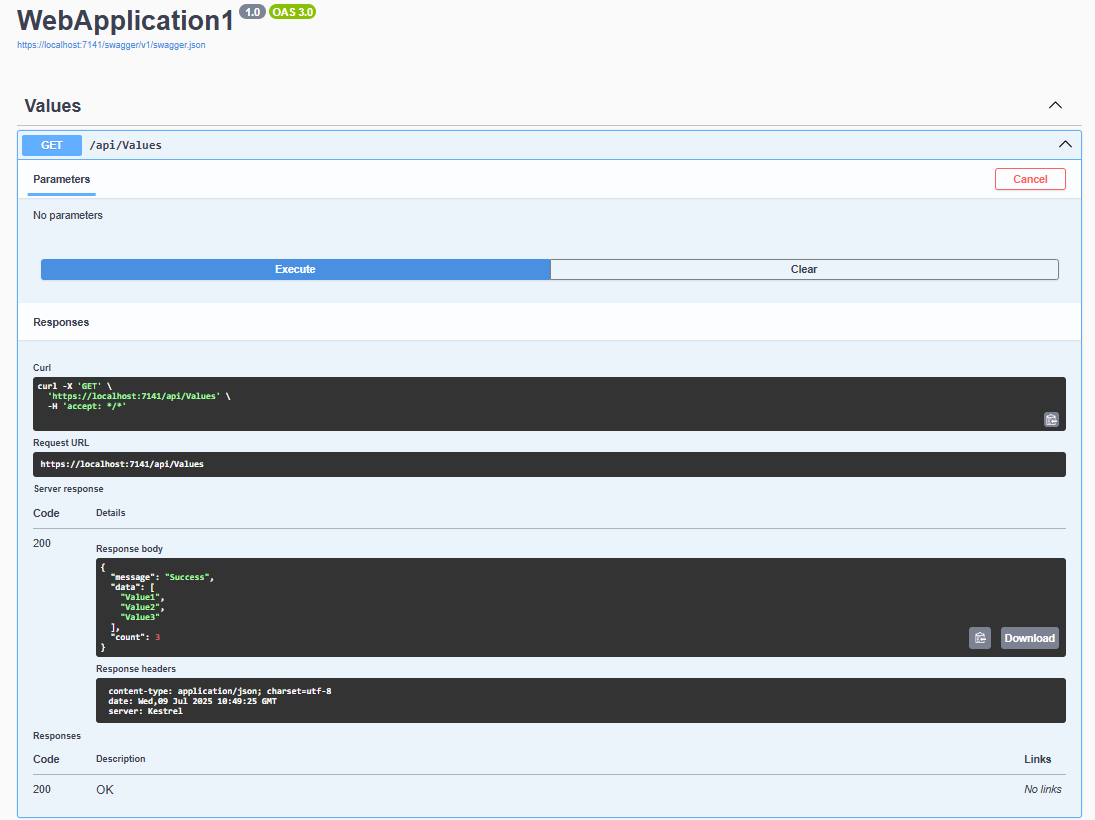
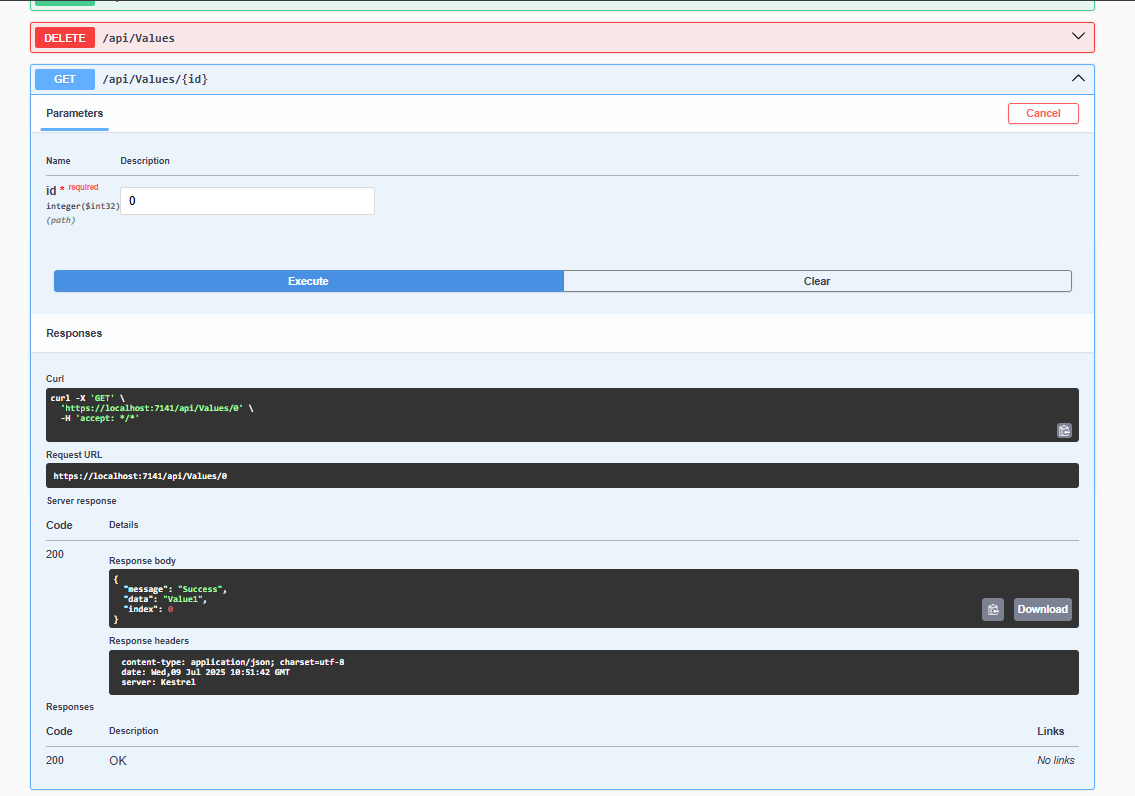
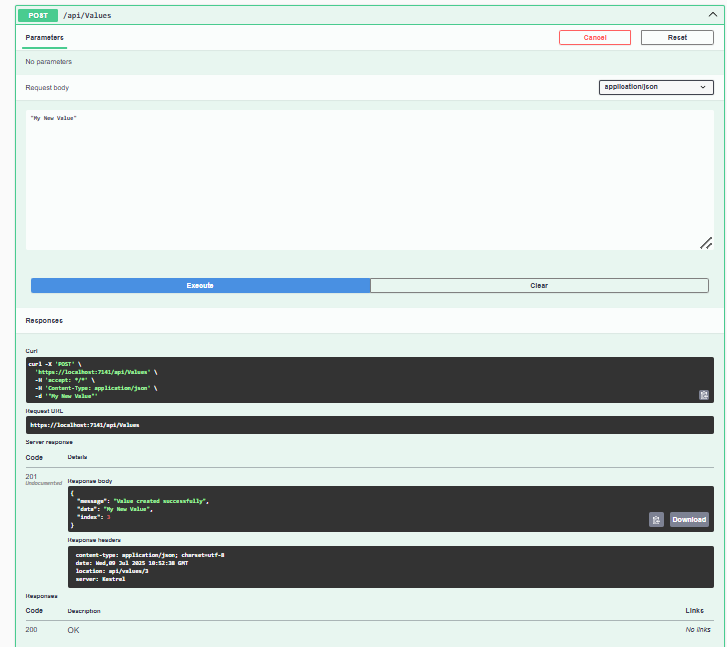
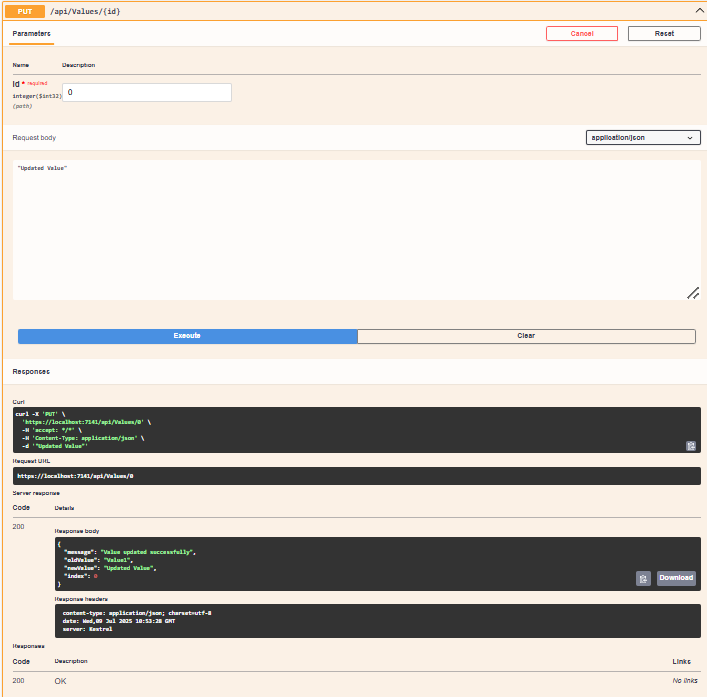
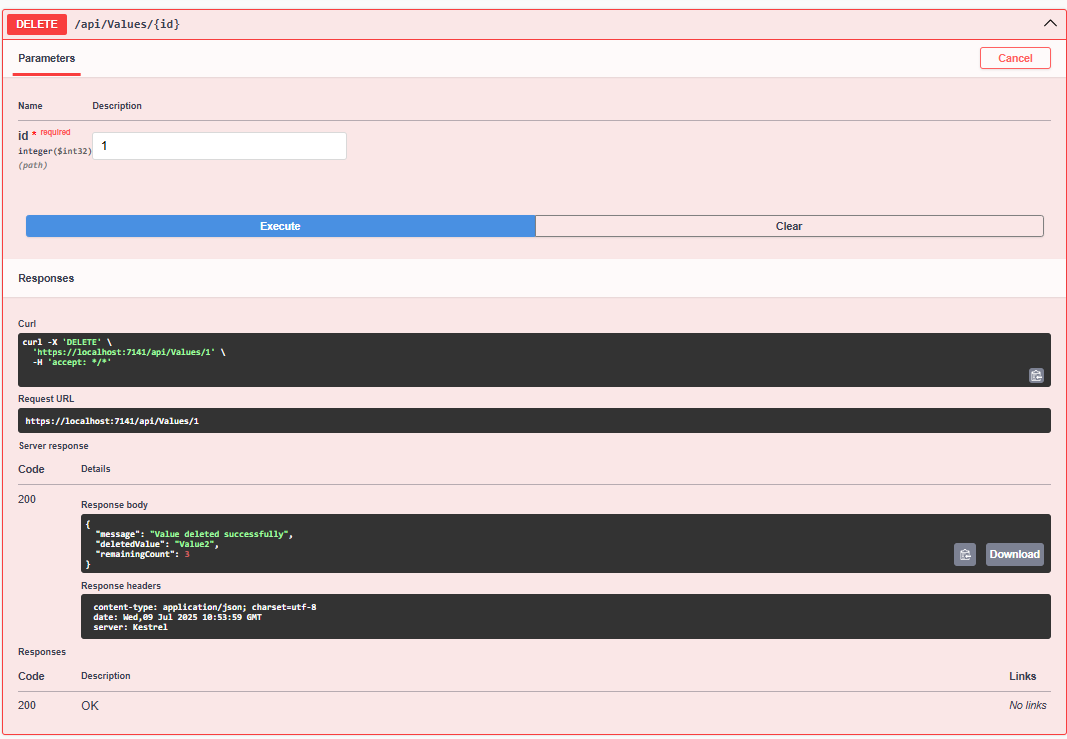
<Project ToolsVersion="Current" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<PropertyGroup>

<ActiveDebugProfile>https</ActiveDebugProfile>

</PropertyGroup>

</Project>

OUTPUTS  
  


Q2)

**CODE 1**

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

// Updated Swagger configuration as per Q2 requirements

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://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 =>

{

// specifying the Swagger JSON endpoint.

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

});

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

**CODE 2**

using Microsoft.AspNetCore.Mvc;

namespace Q2WebAPIDemo.Controllers

{

[ApiController]

[Route("api/Emp")]

public class EmployeeController : ControllerBase

{

// Sample employee data

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

{

new Employee { Id = 1, Name = "John Doe", Position = "Developer", Salary = 50000 },

new Employee { Id = 2, Name = "Jane Smith", Position = "Manager", Salary = 75000 },

new Employee { Id = 3, Name = "Bob Johnson", Position = "Analyst", Salary = 45000 }

};

[HttpGet]

[ProducesResponseType(typeof(List<Employee>), StatusCodes.Status200OK)]

public ActionResult<List<Employee>> GetAllEmployees()

{

return Ok(employees);

}

[HttpGet("{id}")]

[ProducesResponseType(typeof(Employee), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<Employee> GetEmployee(int id)

{

var employee = employees.Find(e => e.Id == id);

if (employee == null)

{

return NotFound();

}

return Ok(employee);

}

[HttpPost]

[ProducesResponseType(typeof(Employee), StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

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

{

if (employee == null)

{

return BadRequest();

}

employee.Id = employees.Count + 1;

employees.Add(employee);

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

}

[HttpPut("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

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

{

if (employee == null)

{

return BadRequest();

}

var existingEmployee = employees.Find(e => e.Id == id);

if (existingEmployee == null)

{

return NotFound();

}

existingEmployee.Name = employee.Name;

existingEmployee.Position = employee.Position;

existingEmployee.Salary = employee.Salary;

return Ok(existingEmployee);

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult DeleteEmployee(int id)

{

var employee = employees.Find(e => e.Id == id);

if (employee == null)

{

return NotFound();

}

employees.Remove(employee);

return Ok("Employee deleted successfully");

}

}

public class Employee

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

public string Position { get; set; } = string.Empty;

public decimal Salary { get; set; }

}

}

**CODE 3**

using Microsoft.AspNetCore.Mvc;

namespace Q2WebAPIDemo.Controllers

{

[ApiController]

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

public class ValuesController : ControllerBase

{

[HttpGet]

[ProducesResponseType(typeof(string[]), StatusCodes.Status200OK)]

public ActionResult<string[]> Get()

{

return Ok(new string[] { "value1", "value2", "value3" });

}

[HttpGet("{id}")]

[ProducesResponseType(typeof(string), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<string> Get(int id)

{

if (id < 1 || id > 3)

{

return NotFound();

}

return Ok($"value{id}");

}

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public IActionResult Post([FromBody] string value)

{

if (string.IsNullOrEmpty(value))

{

return BadRequest();

}

return CreatedAtAction(nameof(Get), new { id = 1 }, value);

}

[HttpPut("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

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

{

if (string.IsNullOrEmpty(value))

{

return BadRequest();

}

return Ok($"Updated value{id} with: {value}");

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

public IActionResult Delete(int id)

{

return Ok($"Deleted value{id}");

}

}

}

**CODE 4**

using Microsoft.AspNetCore.Mvc;

namespace Q2WebAPIDemo.Controllers

{

[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)]

})

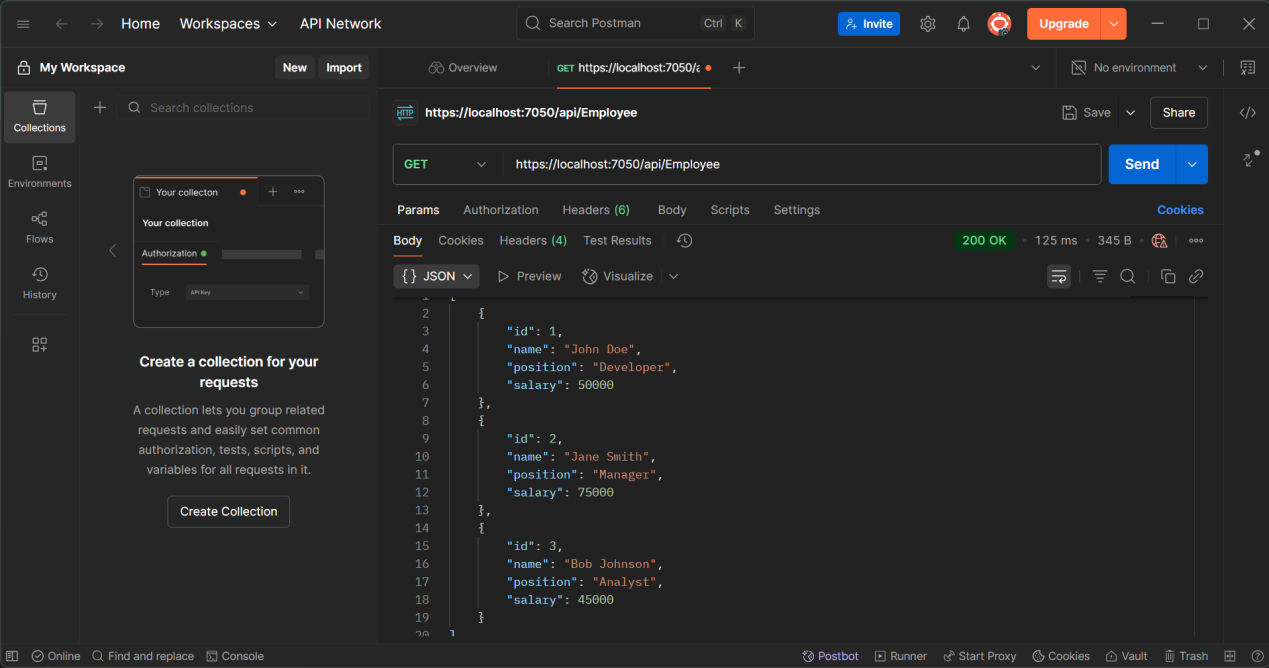
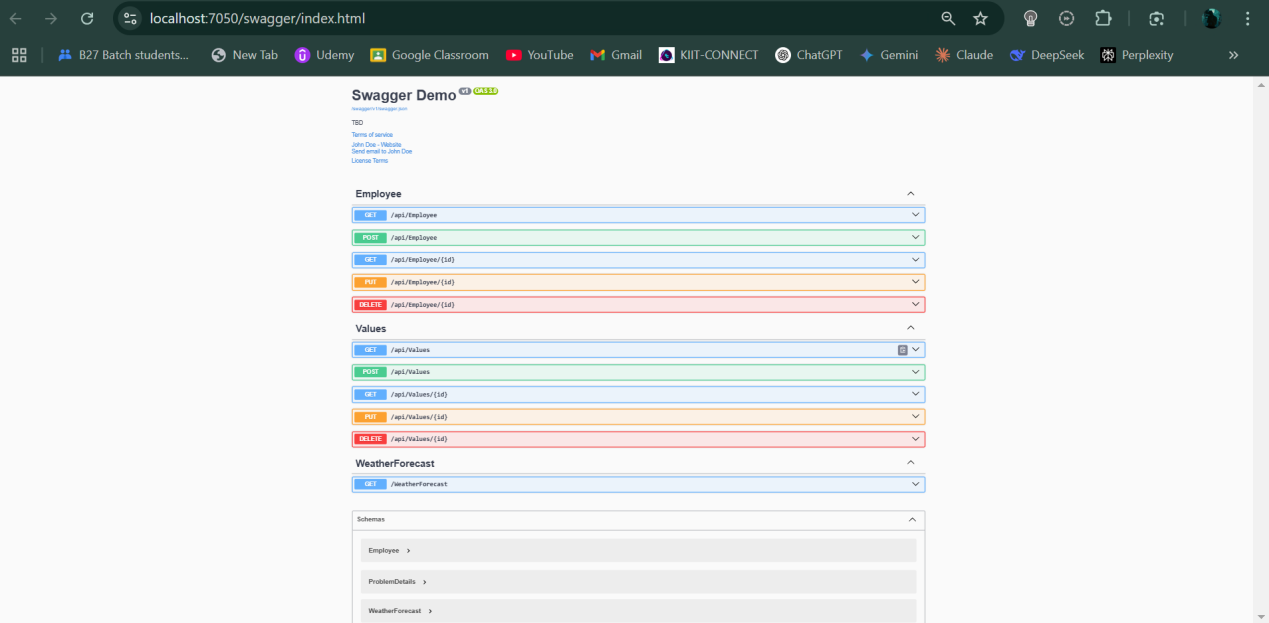
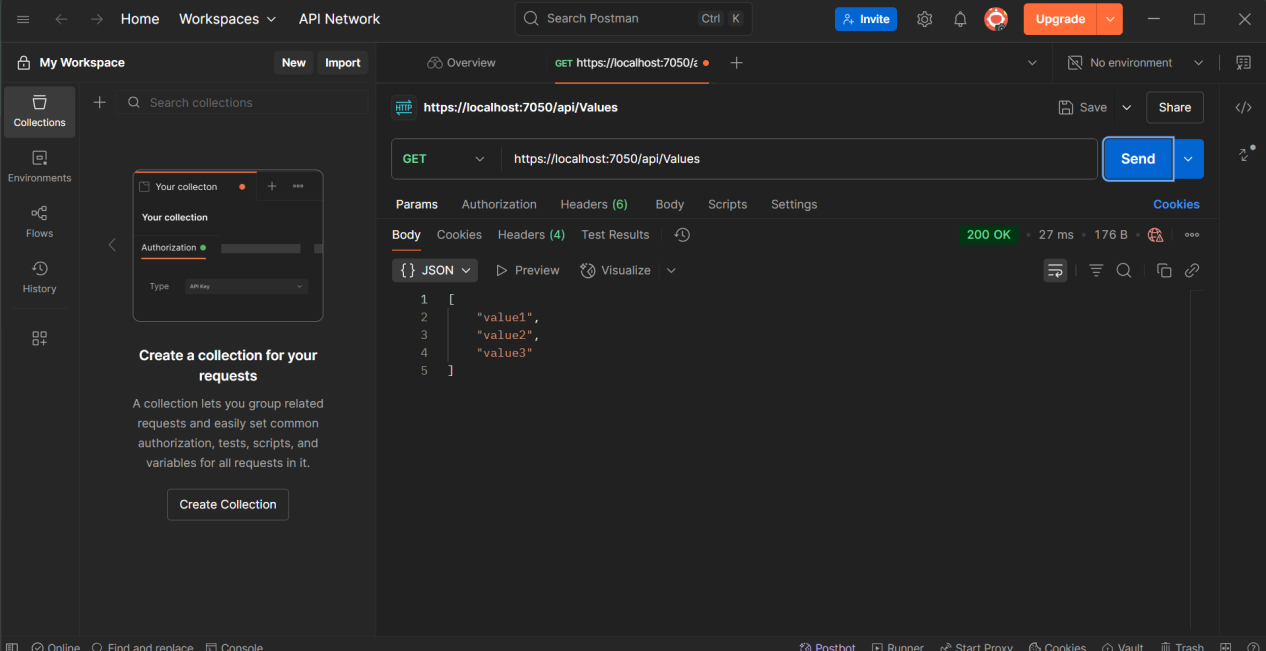
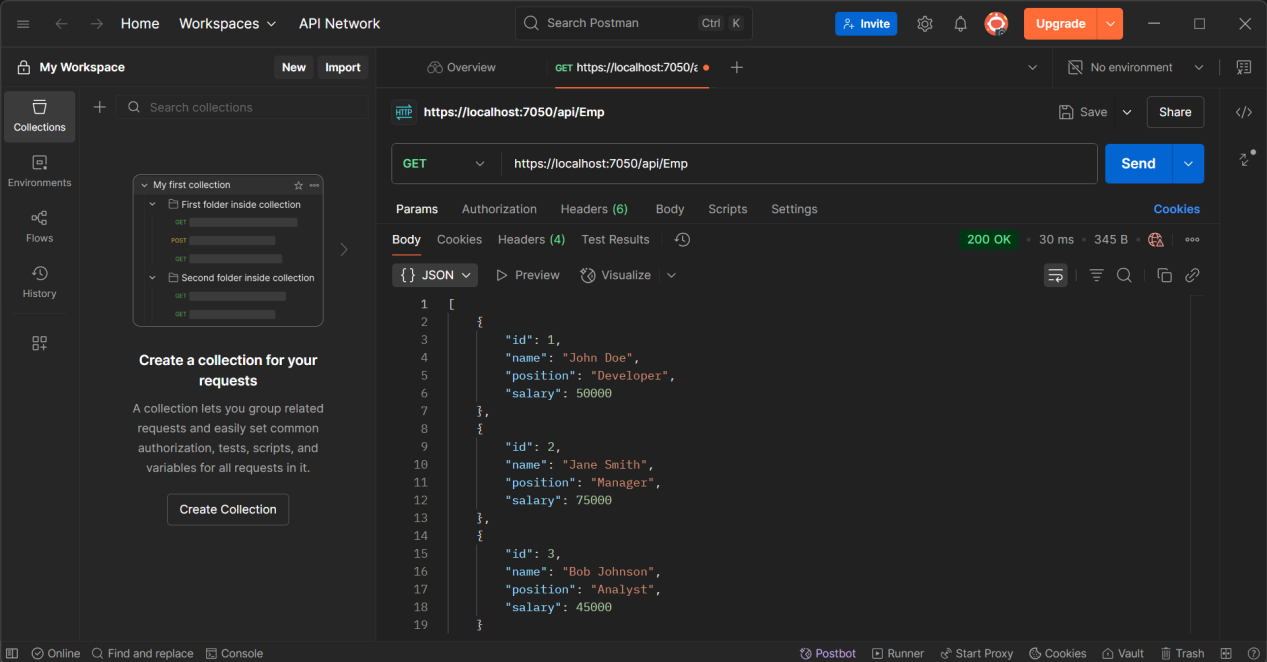
.ToArray();

}

}

}

**OUTPUTS**



**Q3)**

**CODE 1**

using Microsoft.OpenApi.Models;

using Q3WebAPIDemo.Filters;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers(options =>

{

// Add custom exception filter globally

options.Filters.Add<CustomExceptionFilter>();

});

builder.Services.AddEndpointsApiExplorer();

// Swagger configuration

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Q3 WebAPI Demo",

Version = "v1",

Description = "Custom Model and Filters Demo",

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", "Q3 WebAPI Demo");

});

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

**CODE 2**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Q3WebAPIDemo.Models;

using Q3WebAPIDemo.Filters;

namespace Q3WebAPIDemo.Controllers

{

[ApiController]

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

[CustomAuthFilter]

public class EmployeeController : ControllerBase

{

private static List<Employee> employees = new List<Employee>();

// Constructor to create sample data

public EmployeeController()

{

if (employees.Count == 0)

{

employees = GetStandardEmployeeList();

}

}

// Private method to create standard employee list

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

Salary = 50000,

Permanent = true,

DateOfBirth = new DateTime(1990, 5, 15),

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

Skills = new List<Skill>

{

new Skill { Id = 1, Name = "C#", Level = "Advanced" },

new Skill { Id = 2, Name = "ASP.NET", Level = "Intermediate" }

}

},

new Employee

{

Id = 2,

Name = "Jane Smith",

Salary = 75000,

Permanent = true,

DateOfBirth = new DateTime(1985, 8, 22),

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

Skills = new List<Skill>

{

new Skill { Id = 3, Name = "Leadership", Level = "Advanced" },

new Skill { Id = 4, Name = "Communication", Level = "Expert" }

}

},

new Employee

{

Id = 3,

Name = "Bob Johnson",

Salary = 45000,

Permanent = false,

DateOfBirth = new DateTime(1992, 12, 10),

Department = new Department { Id = 3, Name = "Finance", Location = "Texas" },

Skills = new List<Skill>

{

new Skill { Id = 5, Name = "Excel", Level = "Advanced" },

new Skill { Id = 6, Name = "Accounting", Level = "Intermediate" }

}

}

};

}

[HttpGet]

[AllowAnonymous]

[ProducesResponseType(typeof(List<Employee>), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public ActionResult<List<Employee>> GetStandard()

{

// Uncomment this line to test exception filter

// throw new Exception("Test exception for exception filter");

return Ok(employees);

}

[HttpGet("{id}")]

[AllowAnonymous]

[ProducesResponseType(typeof(Employee), StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public ActionResult<Employee> GetEmployee(int id)

{

var employee = employees.Find(e => e.Id == id);

if (employee == null)

{

return NotFound();

}

return Ok(employee);

}

[HttpPost]

[ProducesResponseType(typeof(Employee), StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

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

{

if (employee == null)

{

return BadRequest("Employee data is required");

}

employee.Id = employees.Count + 1;

employees.Add(employee);

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

}

[HttpPut("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

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

{

if (employee == null)

{

return BadRequest("Employee data is required");

}

var existingEmployee = employees.Find(e => e.Id == id);

if (existingEmployee == null)

{

return NotFound();

}

existingEmployee.Name = employee.Name;

existingEmployee.Salary = employee.Salary;

existingEmployee.Permanent = employee.Permanent;

existingEmployee.DateOfBirth = employee.DateOfBirth;

existingEmployee.Department = employee.Department;

existingEmployee.Skills = employee.Skills;

return Ok(existingEmployee);

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public IActionResult DeleteEmployee(int id)

{

var employee = employees.Find(e => e.Id == id);

if (employee == null)

{

return NotFound();

}

employees.Remove(employee);

return Ok("Employee deleted successfully");

}

}

}

**CODE 3**

using Microsoft.AspNetCore.Mvc;

namespace Q3WebAPIDemo.Controllers

{

[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)]

})

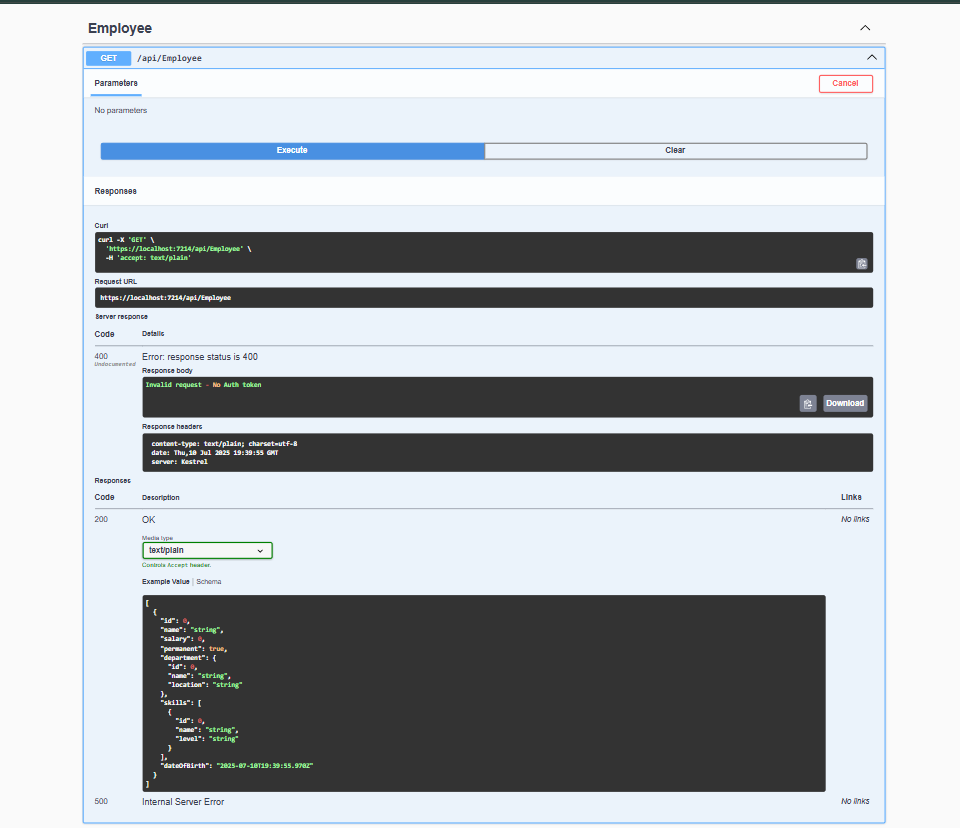
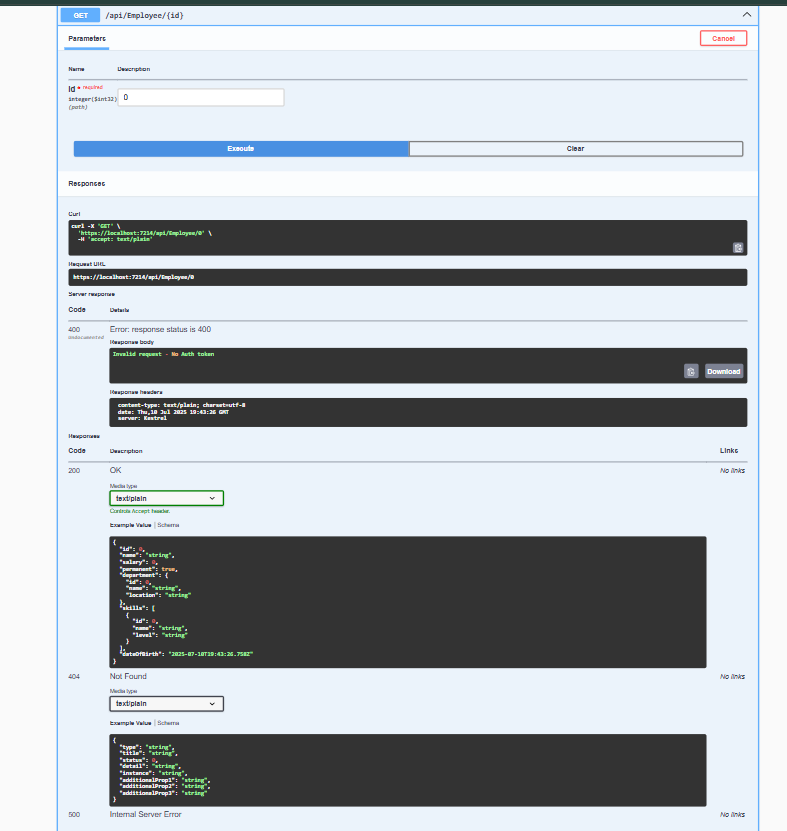
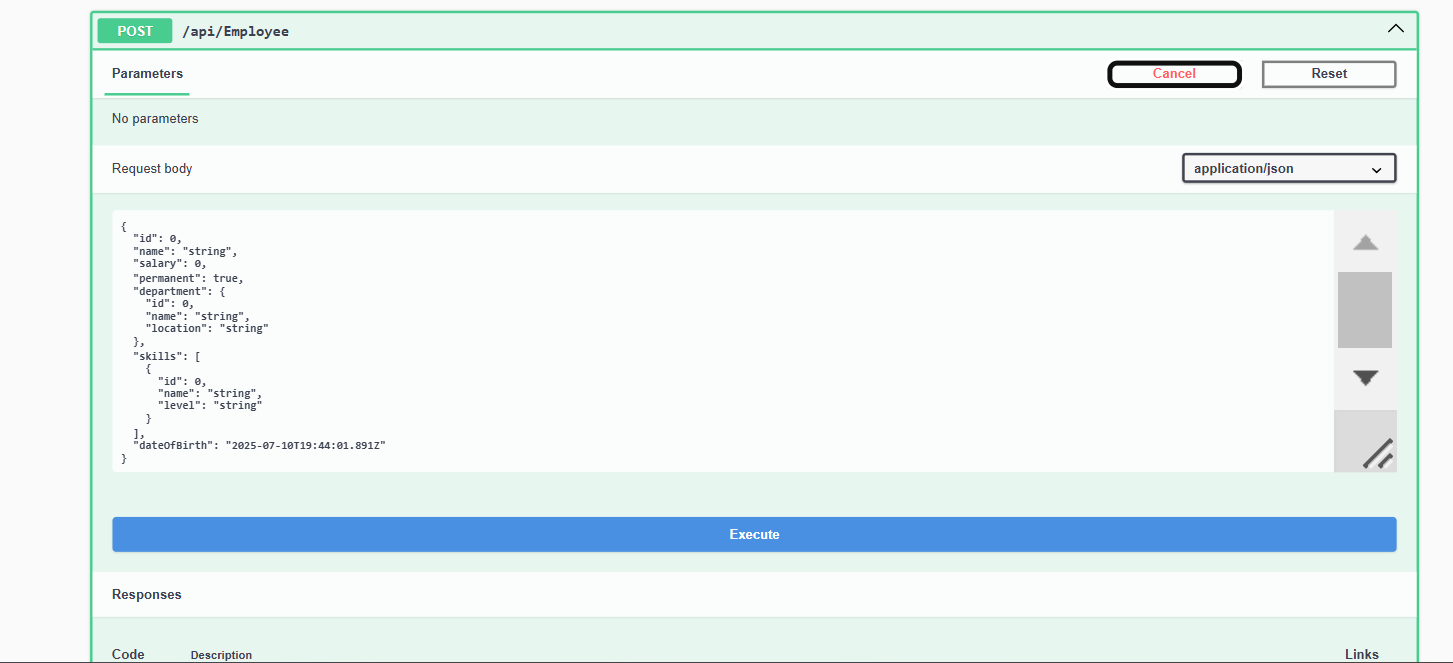
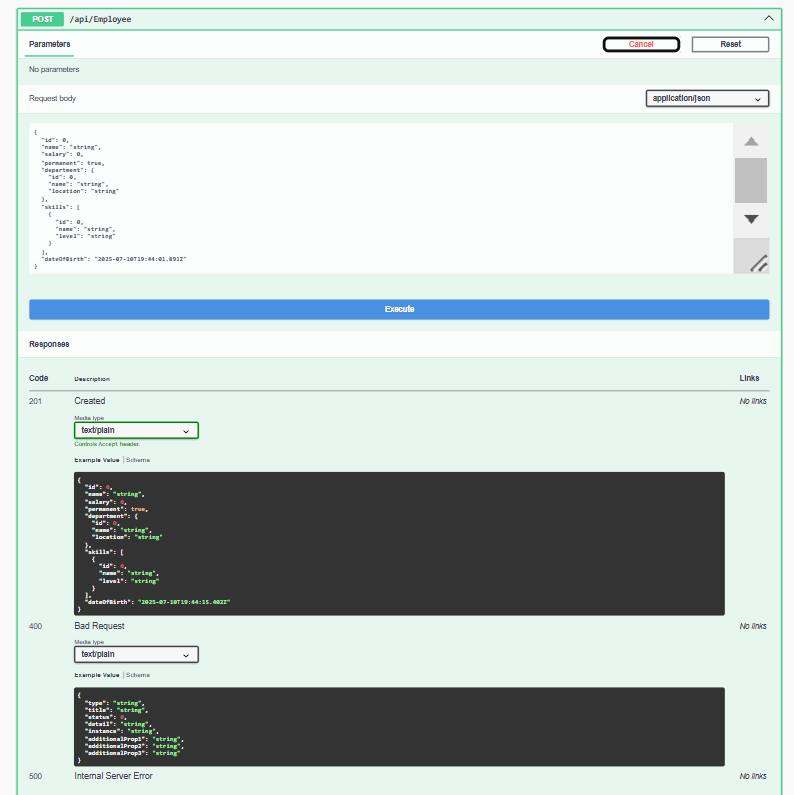
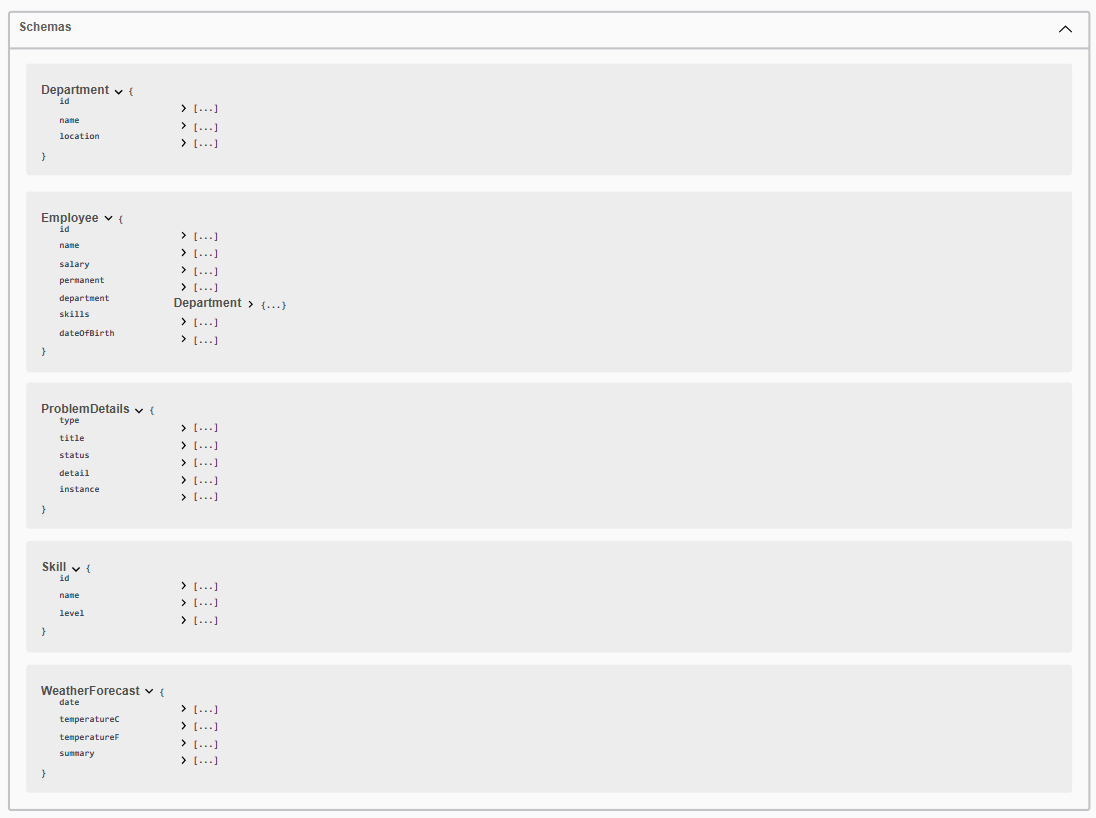
.ToArray();

}

}

}

**OUTPUTS**



**Q4)**

**CODE 1**

using Microsoft.OpenApi.Models;

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(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Employee Management API",

Version = "v1",

Description = "A simple Employee Management API with CRUD operations"

});

});

// Add CORS policy (optional - for frontend integration)

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowAll", builder =>

{

builder.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader();

});

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Employee Management API V1");

});

}

app.UseHttpsRedirection();

app.UseCors("AllowAll");

app.UseAuthorization();

app.MapControllers();

app.Run();

**CODE 2**

using Microsoft.AspNetCore.Mvc;

using EmployeeWebAPI.Models;

using System.ComponentModel.DataAnnotations;

namespace EmployeeWebAPI.Controllers

{

[ApiController]

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

public class EmployeeController : ControllerBase

{

// Hardcoded employee data for demonstration

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

{

new Employee

{

Id = 1,

FirstName = "John",

LastName = "Doe",

Email = "john.doe@company.com",

Department = "IT",

Salary = 50000,

DateOfJoining = new DateTime(2020, 1, 15),

IsActive = true

},

new Employee

{

Id = 2,

FirstName = "Jane",

LastName = "Smith",

Email = "jane.smith@company.com",

Department = "HR",

Salary = 45000,

DateOfJoining = new DateTime(2021, 3, 10),

IsActive = true

},

new Employee

{

Id = 3,

FirstName = "Mike",

LastName = "Johnson",

Email = "mike.johnson@company.com",

Department = "Finance",

Salary = 55000,

DateOfJoining = new DateTime(2019, 7, 20),

IsActive = true

},

new Employee

{

Id = 4,

FirstName = "Sarah",

LastName = "Williams",

Email = "sarah.williams@company.com",

Department = "Marketing",

Salary = 48000,

DateOfJoining = new DateTime(2022, 2, 5),

IsActive = true

}

};

// GET: api/Employee

[HttpGet]

public ActionResult<IEnumerable<Employee>> GetAllEmployees()

{

return Ok(employees);

}

// GET: api/Employee/5

[HttpGet("{id}")]

public ActionResult<Employee> GetEmployee(int id)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (employee == null)

{

return BadRequest("Invalid employee id");

}

return Ok(employee);

}

// POST: api/Employee

[HttpPost]

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

{

if (employee == null)

{

return BadRequest("Employee data is required");

}

// Validate model state

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

// Generate new ID

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

employees.Add(employee);

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

}

// PUT: api/Employee/5

[HttpPut("{id}")]

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

{

// Check if the id value is lesser than or equal to 0

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

// Check if employee exists in the hardcoded list

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

if (existingEmployee == null)

{

return BadRequest("Invalid employee id");

}

// Validate input data

if (employee == null)

{

return BadRequest("Employee data is required");

}

// Validate model state

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

// Update the existing employee with new data

existingEmployee.FirstName = employee.FirstName;

existingEmployee.LastName = employee.LastName;

existingEmployee.Email = employee.Email;

existingEmployee.Department = employee.Department;

existingEmployee.Salary = employee.Salary;

existingEmployee.DateOfJoining = employee.DateOfJoining;

existingEmployee.IsActive = employee.IsActive;

// Return the updated employee

return Ok(existingEmployee);

}

// DELETE: api/Employee/5

[HttpDelete("{id}")]

public ActionResult DeleteEmployee(int id)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (employee == null)

{

return BadRequest("Invalid employee id");

}

employees.Remove(employee);

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

}

// GET: api/Employee/department/{department}

[HttpGet("department/{department}")]

public ActionResult<IEnumerable<Employee>> GetEmployeesByDepartment(string department)

{

if (string.IsNullOrEmpty(department))

{

return BadRequest("Department name is required");

}

var departmentEmployees = employees.Where(e =>

e.Department.Equals(department, StringComparison.OrdinalIgnoreCase)).ToList();

if (!departmentEmployees.Any())

{

return NotFound($"No employees found in {department} department");

}

return Ok(departmentEmployees);

}

// GET: api/Employee/active

[HttpGet("active")]

public ActionResult<IEnumerable<Employee>> GetActiveEmployees()

{

var activeEmployees = employees.Where(e => e.IsActive).ToList();

return Ok(activeEmployees);

}

}

}

**CODE 3**

using Microsoft.AspNetCore.Mvc;

namespace EmployeeWebAPI.Controllers

{

[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)]

})

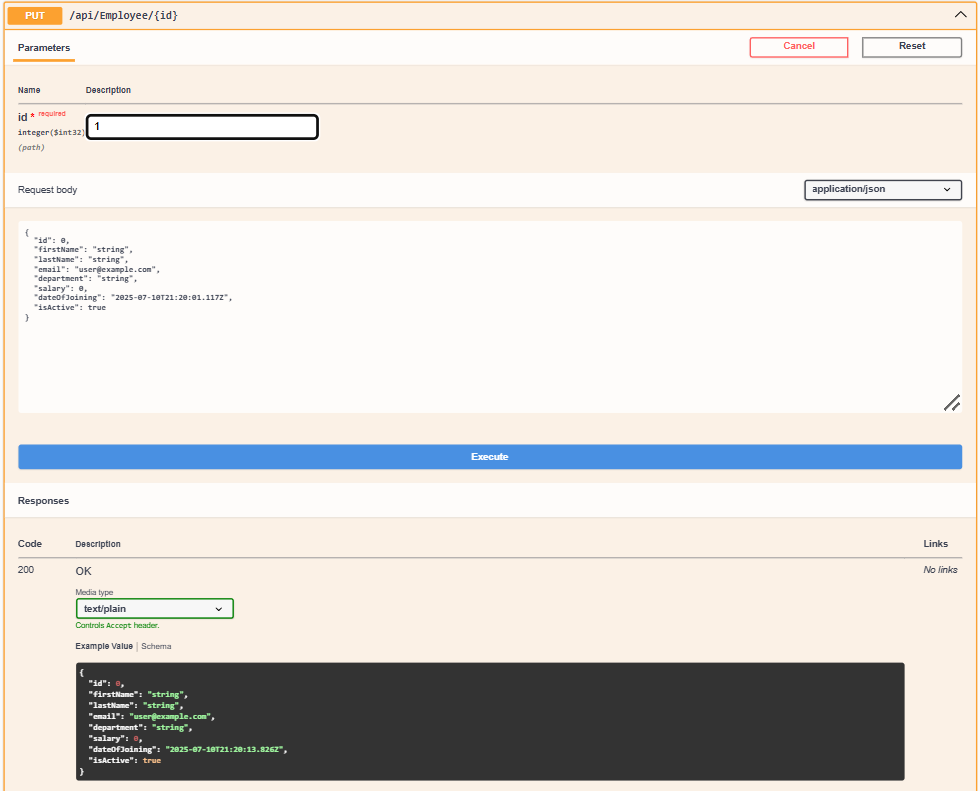
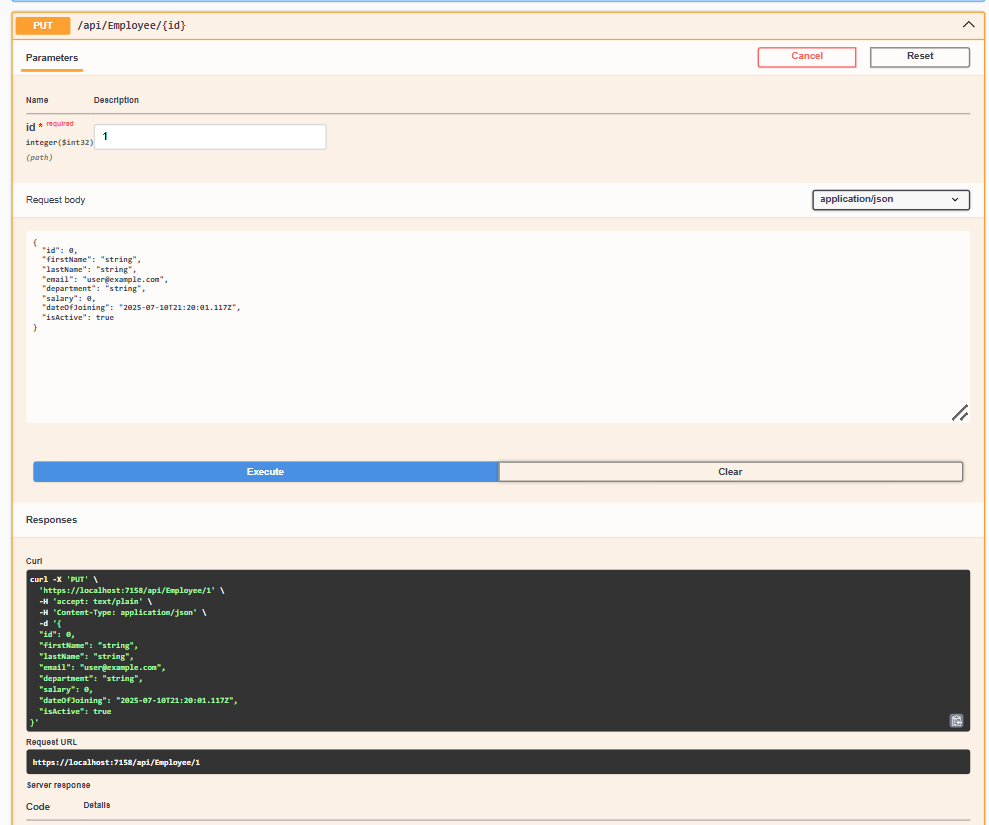
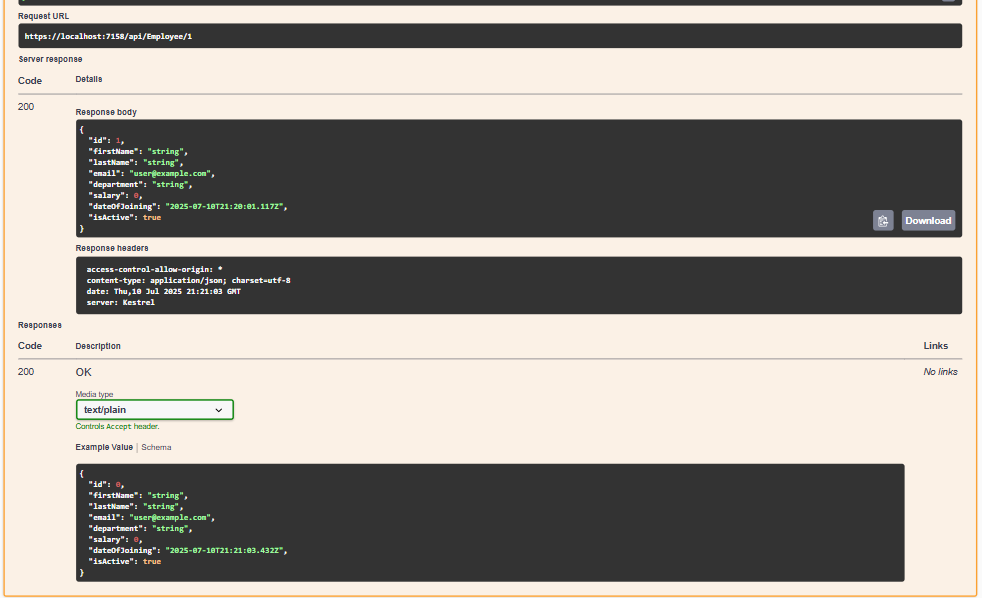
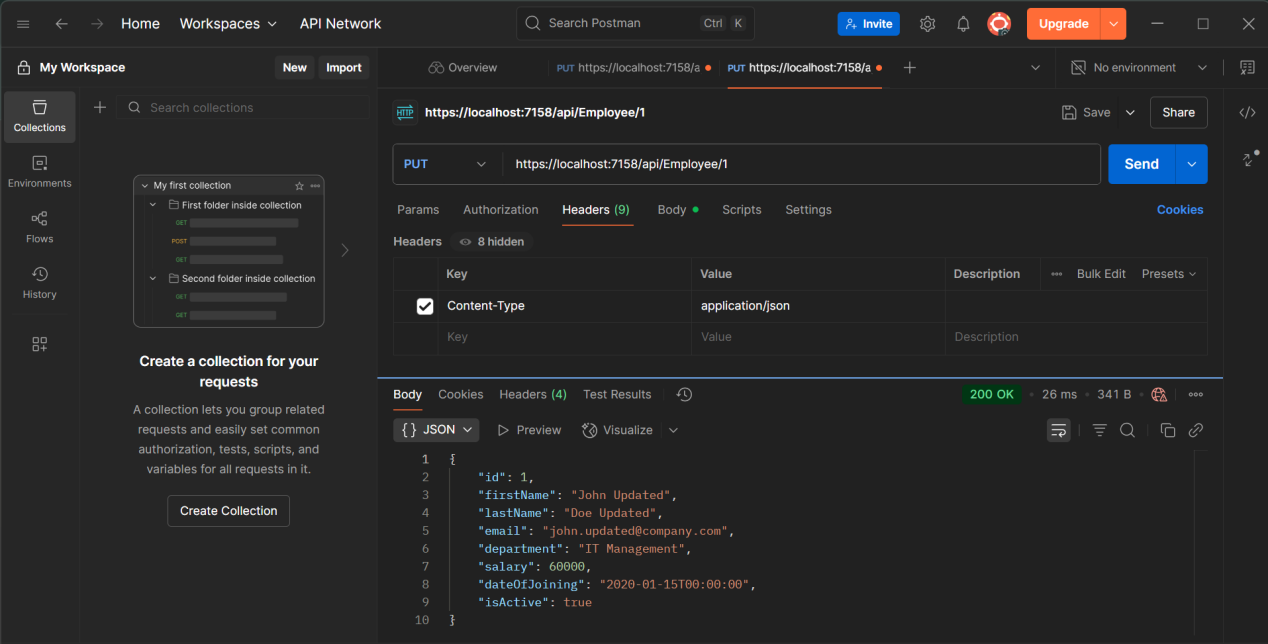
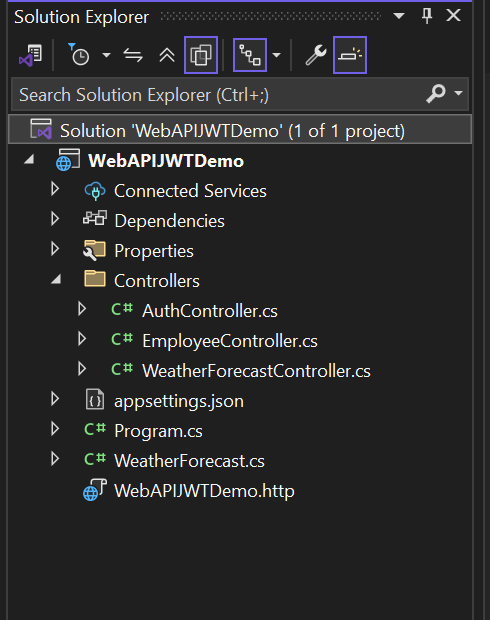
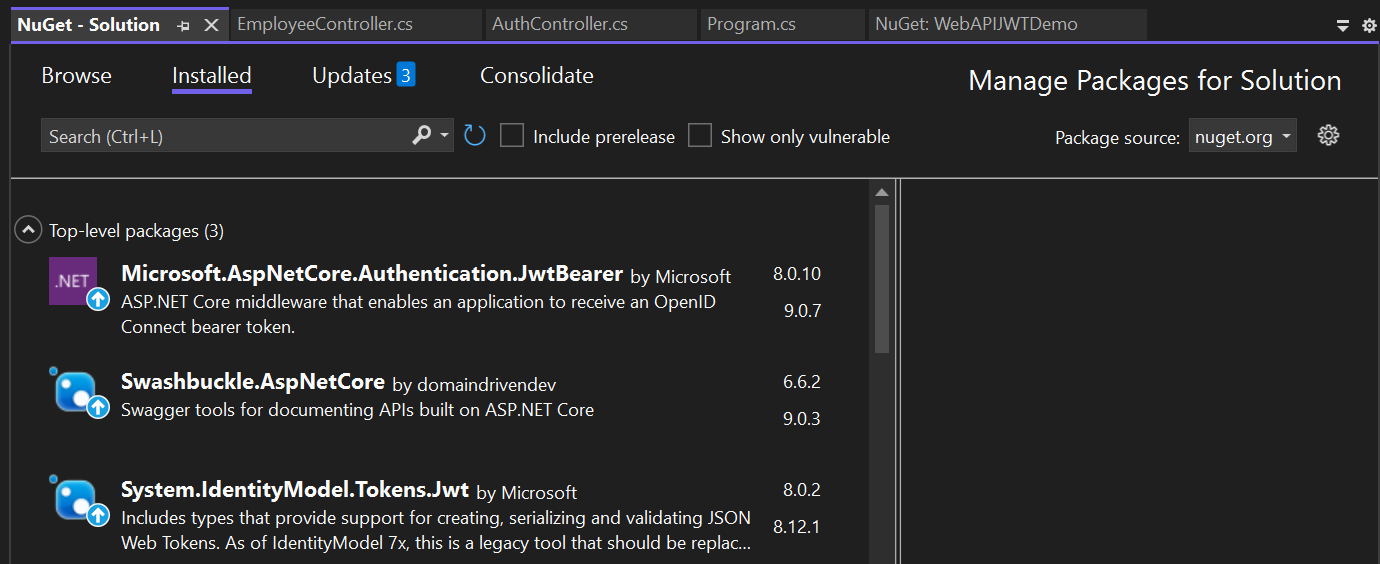
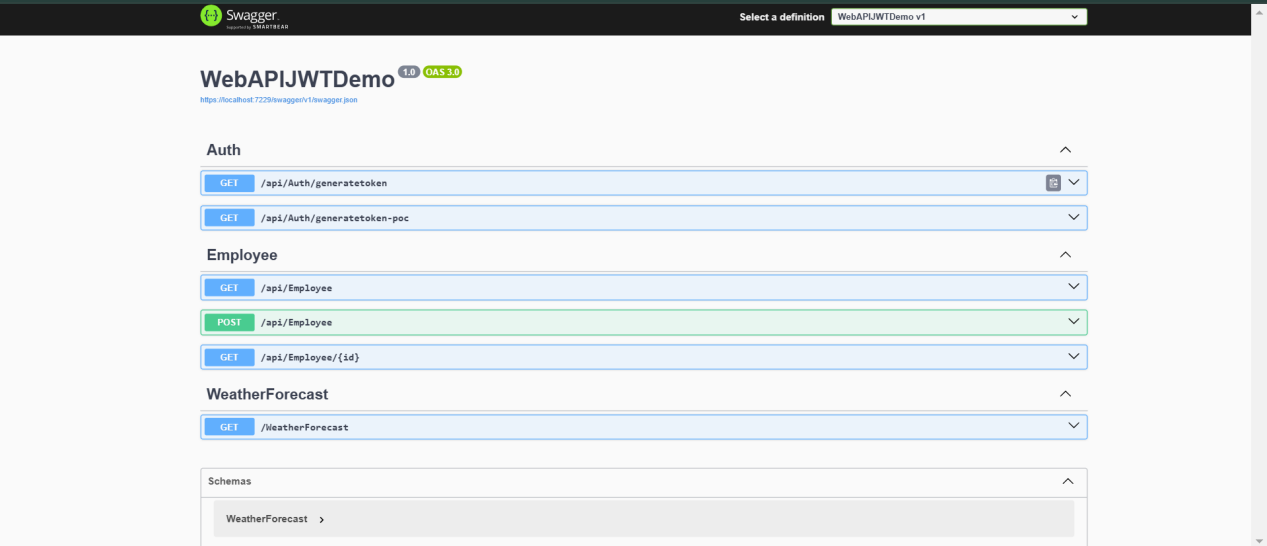
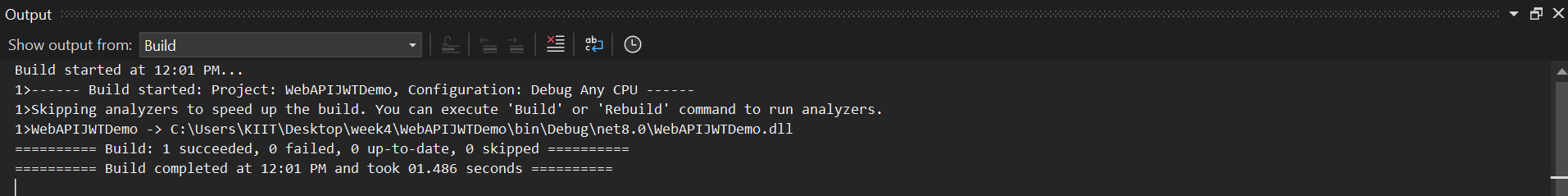
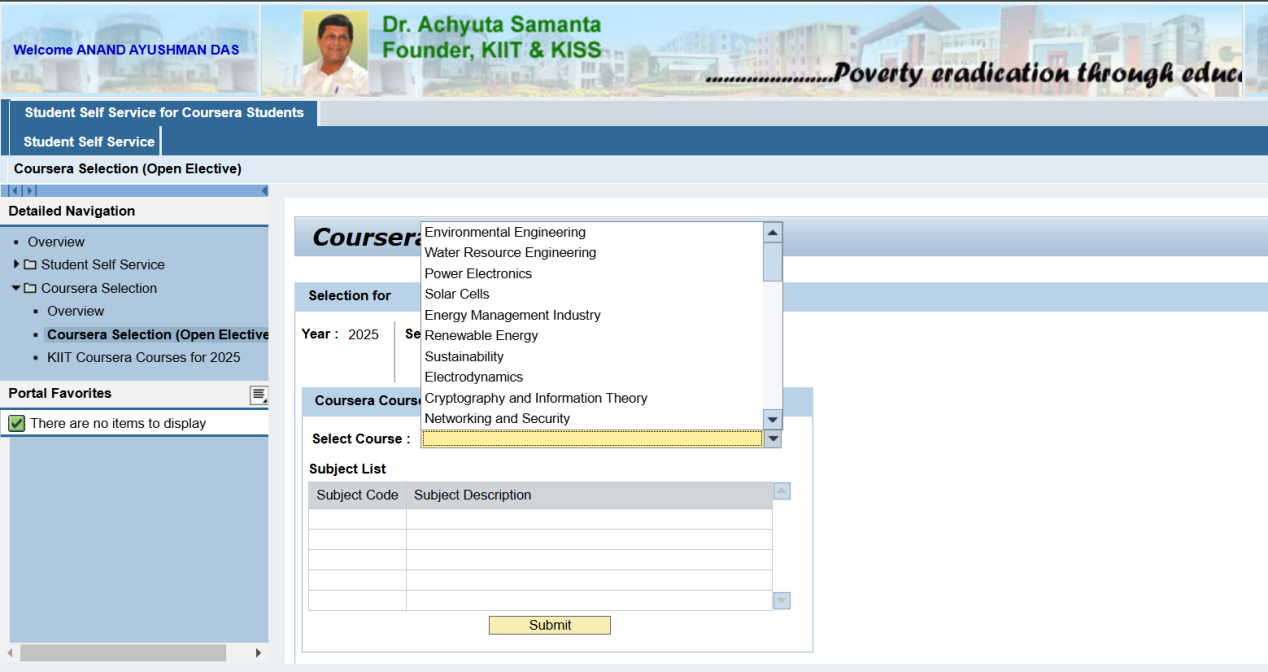
.ToArray();

}

}

}

**OUTPUTS**



**Q5)**

**CODE 1**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// CORS Configuration

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowAll",

policyBuilder =>

{

policyBuilder.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader();

});

});

// JWT Authentication Configuration

string securityKey = "mysuperdupersecretkeyforjwtauthentication2024";

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

builder.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

};

});

// 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();

// Enable CORS

app.UseCors("AllowAll");

// Enable Authentication

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

**CODE 2**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace WebAPIJWTDemo.Controllers

{

[ApiController]

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

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet("generatetoken")]

public IActionResult GenerateToken()

{

// Sample user data - in real app, this would come from login validation

int userId = 1;

string userRole = "Admin";

var token = GenerateJSONWebToken(userId, userRole);

return Ok(new { token = token });

}

[HttpGet("generatetoken-poc")]

public IActionResult GenerateTokenPOC()

{

// Generate token with POC role for testing

int userId = 2;

string userRole = "POC";

var token = GenerateJSONWebToken(userId, userRole);

return Ok(new { token = token });

}

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecretkeyforjwtauthentication2024"));

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), // Initially 10 minutes

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**CODE 3**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace WebAPIJWTDemo.Controllers

{

[ApiController]

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

[Authorize] // Basic authorization - any valid JWT token

public class EmployeeController : ControllerBase

{

[HttpGet]

public IActionResult GetEmployees()

{

var employees = new[]

{

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

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

new { Id = 3, Name = "Bob Johnson", Department = "Finance" }

};

return Ok(employees);

}

[HttpGet("{id}")]

public IActionResult GetEmployee(int id)

{

var employee = new { Id = id, Name = "Sample Employee", Department = "IT" };

return Ok(employee);

}

[HttpPost]

[Authorize(Roles = "Admin")] // Only Admin role can create employees

public IActionResult CreateEmployee([FromBody] dynamic employee)

{

return Ok(new { Message = "Employee created successfully", Data = employee });

}

}

}

**CODE 4**

namespace WebAPIJWTDemo

{

public class WeatherForecast

{

public DateOnly Date { get; set; }

public int TemperatureC { get; set; }

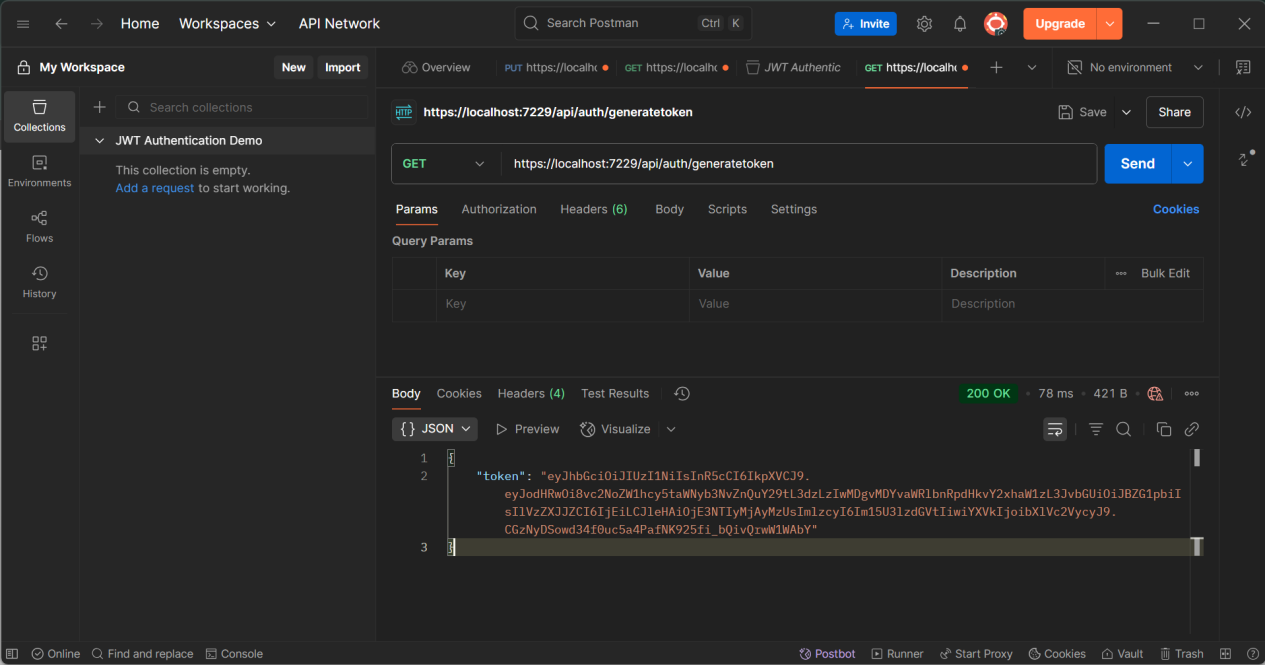
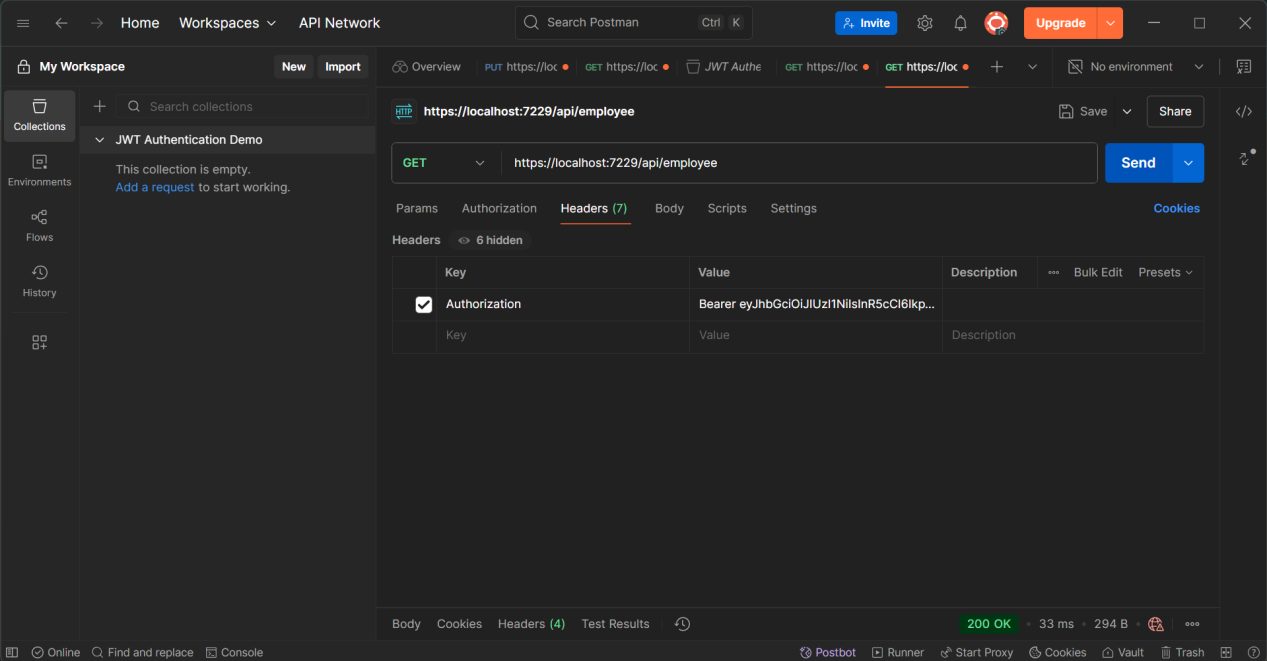
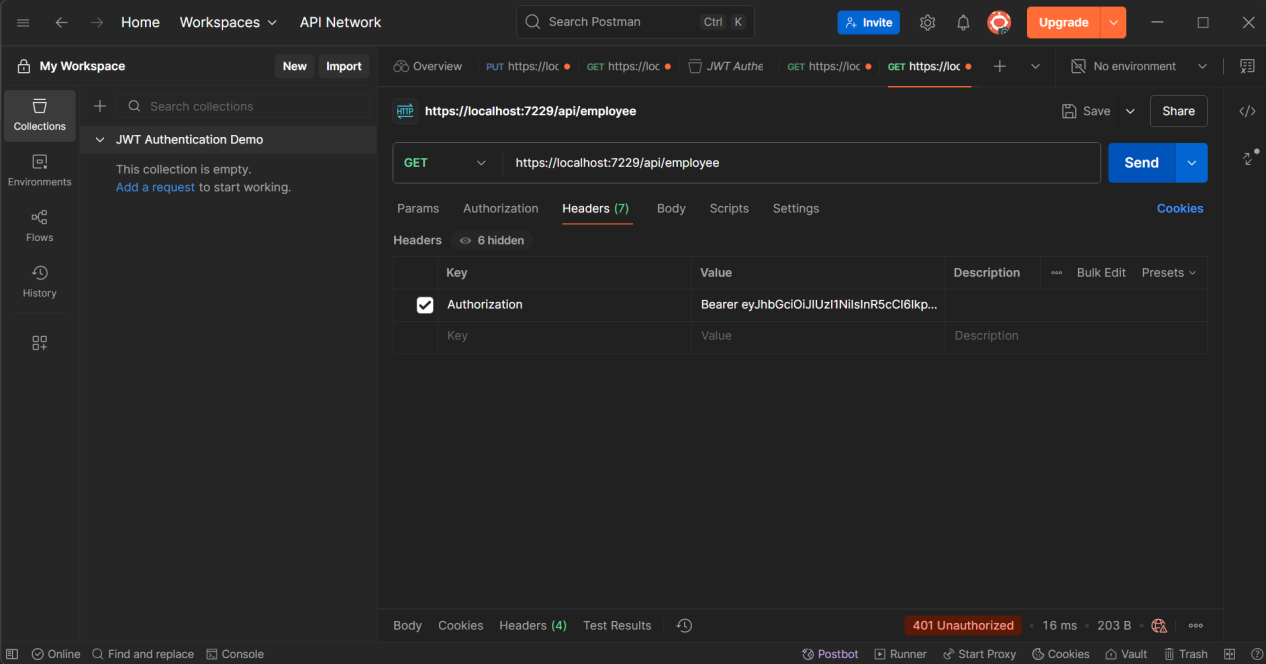
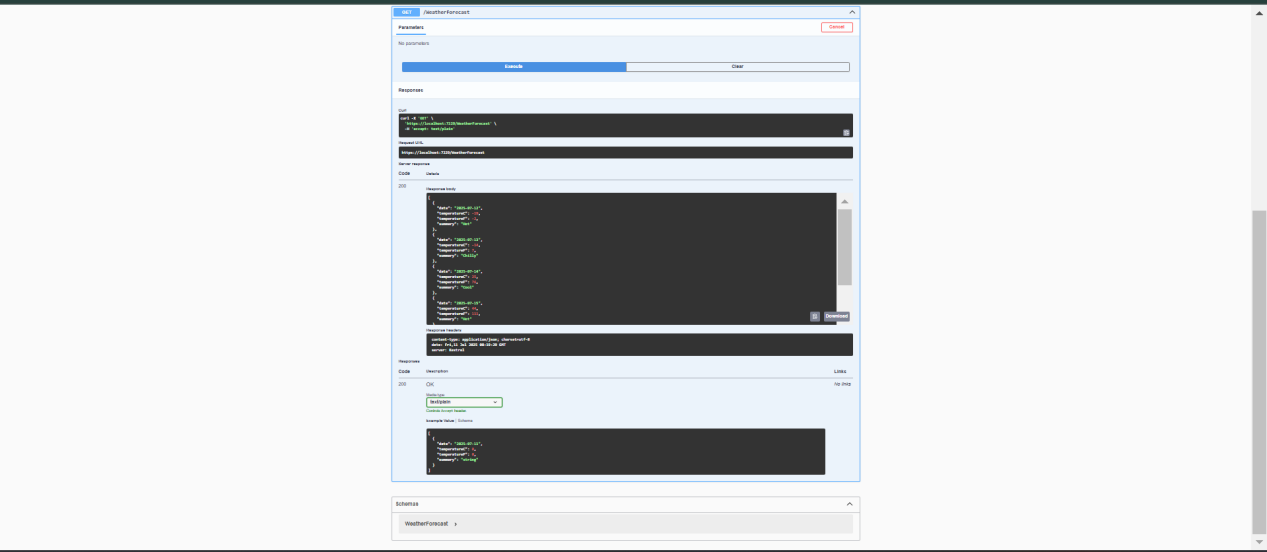
public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

public string? Summary { get; set; }

}

}

OUTPUTS



**Q6**

**CODE 1**

using System;

using Confluent.Kafka;

class Program

{

static void Main(string[] args)

{

var config = new ProducerConfig { BootstrapServers = "localhost:9092" };

using (var producer = new ProducerBuilder<Null, string>(config).Build())

{

while (true)

{

Console.Write("Enter message (or 'exit' to quit): ");

var value = Console.ReadLine();

if (value == "exit") break;

producer.Produce("test-topic", new Message<Null, string> { Value = value },

(deliveryReport) =>

{

if (deliveryReport.Error.IsError)

Console.WriteLine($"Delivery Error: {deliveryReport.Error.Reason}");

else

Console.WriteLine($"Delivered to: {deliveryReport.TopicPartitionOffset}");

});

}

}

}

}

**CODE 2**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net8.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Moonlay.Confluent.Kafka" Version="1.0.0" />

</ItemGroup>

</Project>

**OUTPUTS**

