**WebAPI1:**

A service built on the foundations of Representational State Transfer (REST) is known as a RESTful Web Service. It operates on resources, which are usually represented in XML or JSON, using standard HTTP methods. REST places a strong emphasis on stateless communication, which means that the server does not save client context in between requests and that each request includes all the information required to process it.

Web API: An Application Programming Interface that can be accessed via HTTP on the web. Web APIs can have different architectural styles or be RESTful. They make endpoints accessible so that clients can engage with the functionality and data of applications.

An application built using the microservice architectural style consists of discrete, small services that are each dedicated to a particular business function. Because each microservice can have its own database and RESTful API, it can be deployed and scaled independently.

Features of REST architecture :

* Clients use standard HTTP methods to interact with resources that are represented by URLs.
* Statelessness: Every client-server request must include all the data required to comprehend and handle the request.
* Messages: HTTP requests and responses, frequently utilizing JSON or XML, are used for communication.
* Microservice Concept: To enable autonomous development and deployment, microservices are frequently connected via RESTful APIs.

WebAPI vs. WebService:

Any service that is accessible via the internet (such as SOAP-based) is referred to as a webservice.

RESTful APIs, which are lighter and use HTTP directly, are frequently referred to as WebAPI.

Not Limited to XML: RESTful APIs can return data in a number of formats, including plain text, HTML, XML, and most frequently JSON.

HTTP Request and HTTP Response:

The incoming request from a client to the server is represented by the HttpRequest object. It includes details like the URL, headers, HTTP method, and body (client-sent data).

The server's answer to the client's request is represented by the HttpResponse. It consists of the body (data sent back to the client), headers, and the status code.

Action verbs in Web API:

|  |  |  |
| --- | --- | --- |
| Verb | Meaning | Attribute in Web API |
| GET | Retrieve data from the server | [HttpGet] |
| POST | Create new data on the server | [HttpPost] |
| PUT | Update existing data on the server | [HttpPut] |
| DELETE | Remove data from the server | [HttpDelete] |

Typical HTTP Status Codes for Web APIs:

|  |  |  |
| --- | --- | --- |
| Status | Meaning | ActionResult Type |
| 200 OK | Request Succeeded | Ok() |
| 400 Bad Request | Client sent invalid data | BadRequest() |
| 401 Unauthorized | Authentication required/failed | Unauthorized() |
| 500 Internal Server Error | Server encountered an error | InternalServerError() |

Structure of a web api:

* Controller:A basic web API controller's structure is derived from ApiController (in.NET Core, from ControllerBase).
* Action Verbs: An HTTP verb attribute is used to adorn each method.
* Action methods: are those that deal with incoming HTTP requests.

Configuration files of WebAPI:

Startup.cs: Manages dependency injection, middleware configuration, and application startup.

appSettings.json:Application settings, including connection strings and custom configuration, are stored in appSettings.json.

launchSettings.json: Contains environment variables and profiles that control how the application is launched.

Route.config & WebAPI.config (in .NET 4.5):In older.NET Framework projects, the files Route.config and WebAPI.config (in.NET 4.5) are used for routing and general Web API configuration.

Q1)

ValuesController.cs:

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

public class ValuesController : ControllerBase

{

[HttpGet]

public IEnumerable<string> Get()

{

return new string[] { "value1", "value2" };

}

[HttpPost]

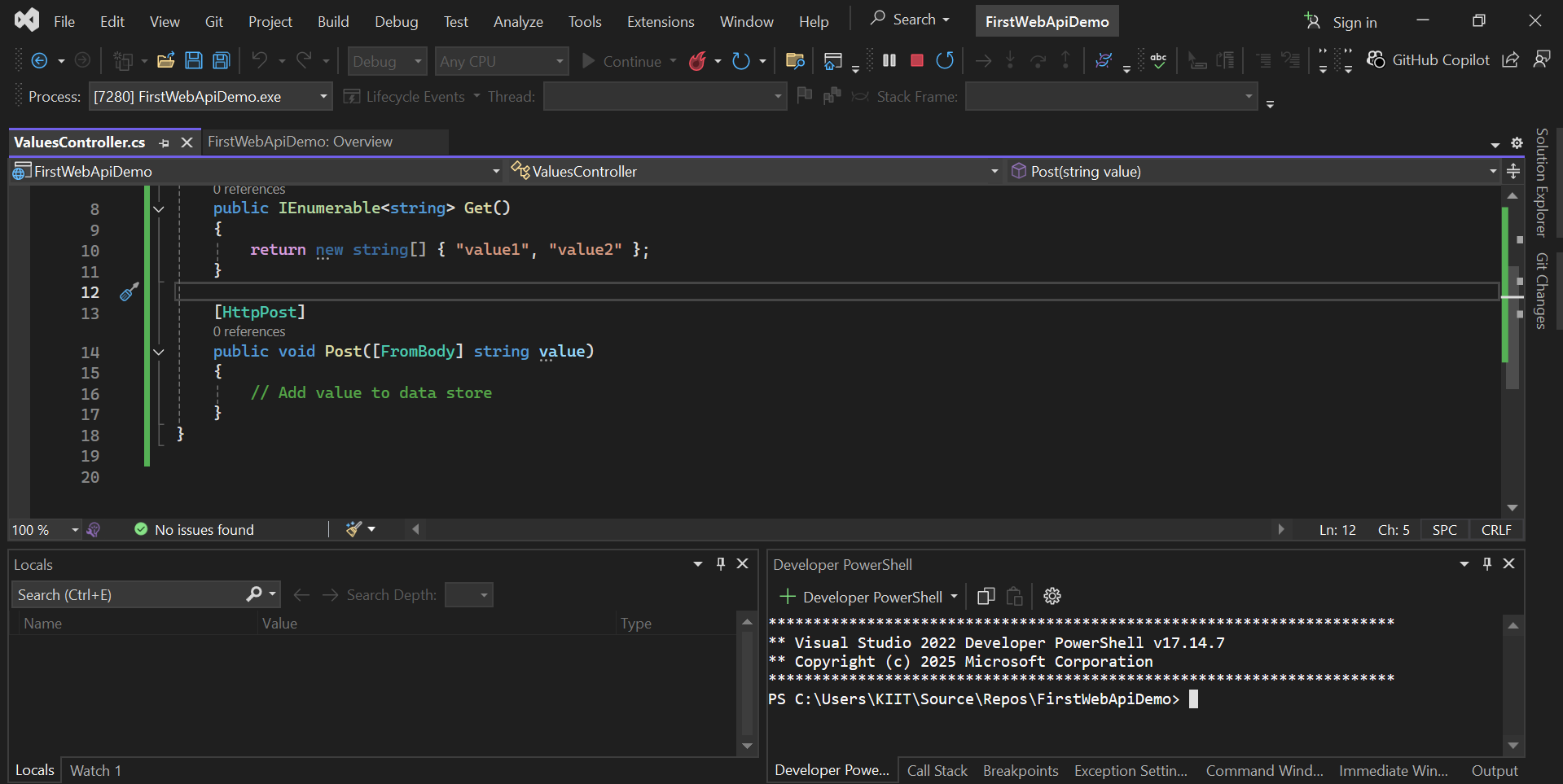
public void Post([FromBody] string value)

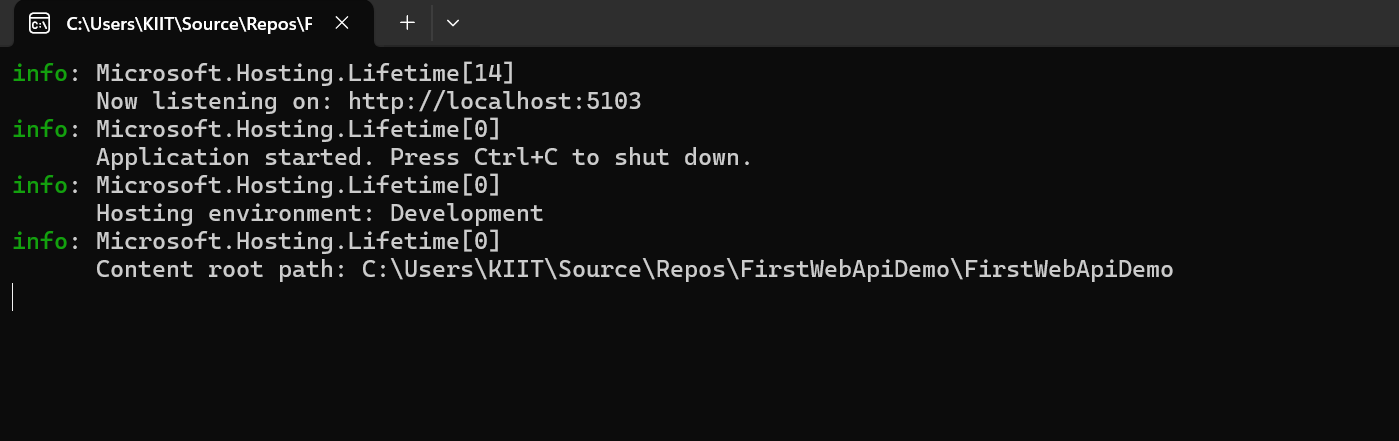
{

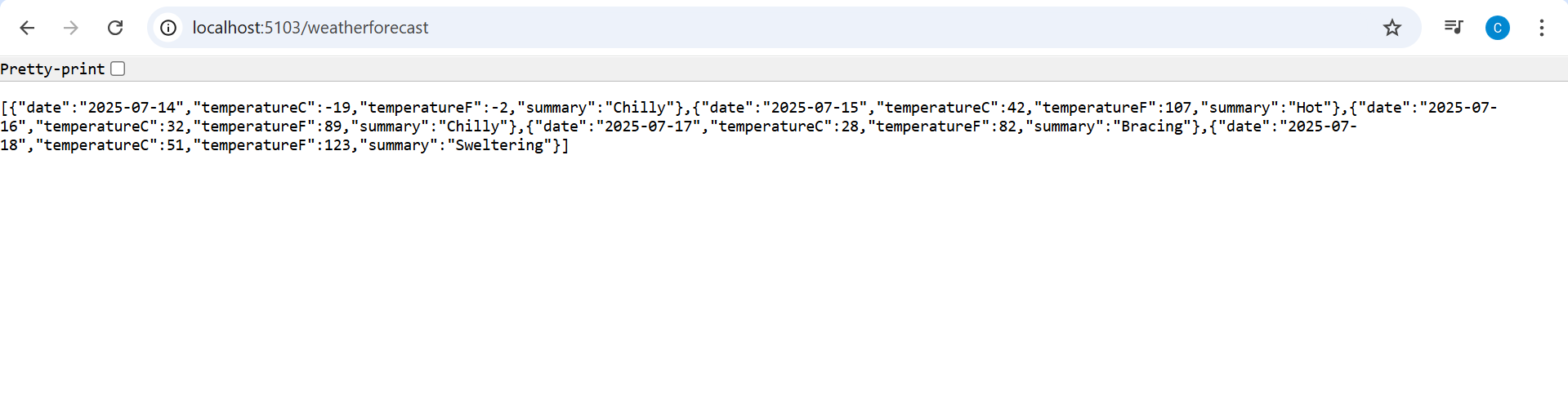
// Add value to data store

}

}









**WebAPI2:**

Program.cs:

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddOpenApi();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Microsoft.OpenApi.Models.OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = new Uri("https://example.com/terms"),

Contact = new Microsoft.OpenApi.Models.OpenApiContact

{

Name = "John Doe",

Email = "john@xyzmail.com",

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

},

License = new Microsoft.OpenApi.Models.OpenApiLicense

{

Name = "License Terms",

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

}

});

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

app.MapOpenApi();

}

app.UseAuthorization();

app.MapControllers();

app.Run();

WeatherForecast.cs:

namespace WebApplication2

{

public class WeatherForecast

{

public DateOnly Date { get; set; }

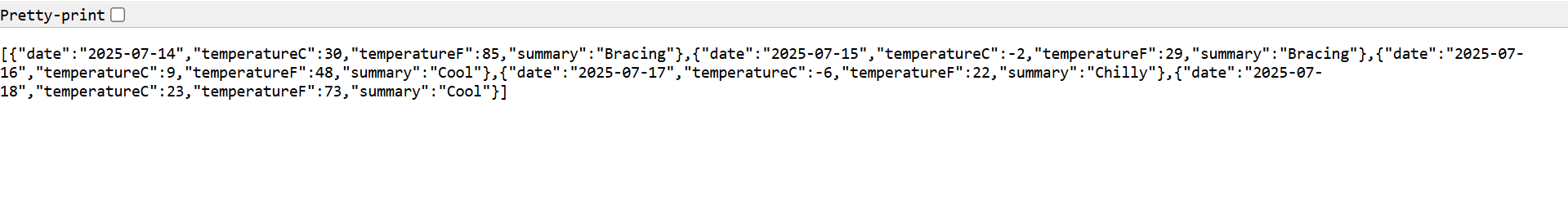
public int TemperatureC { get; set; }

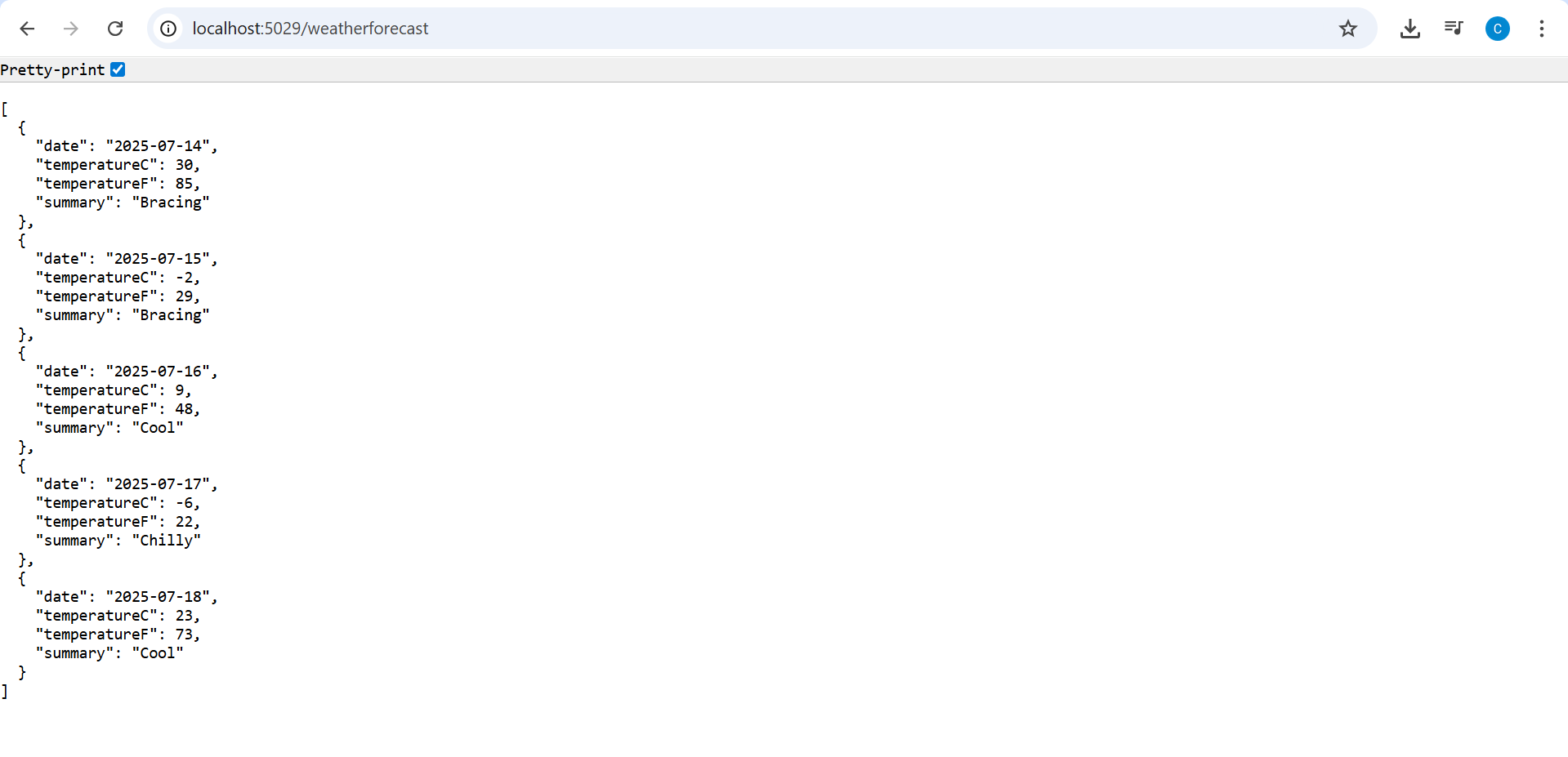
public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

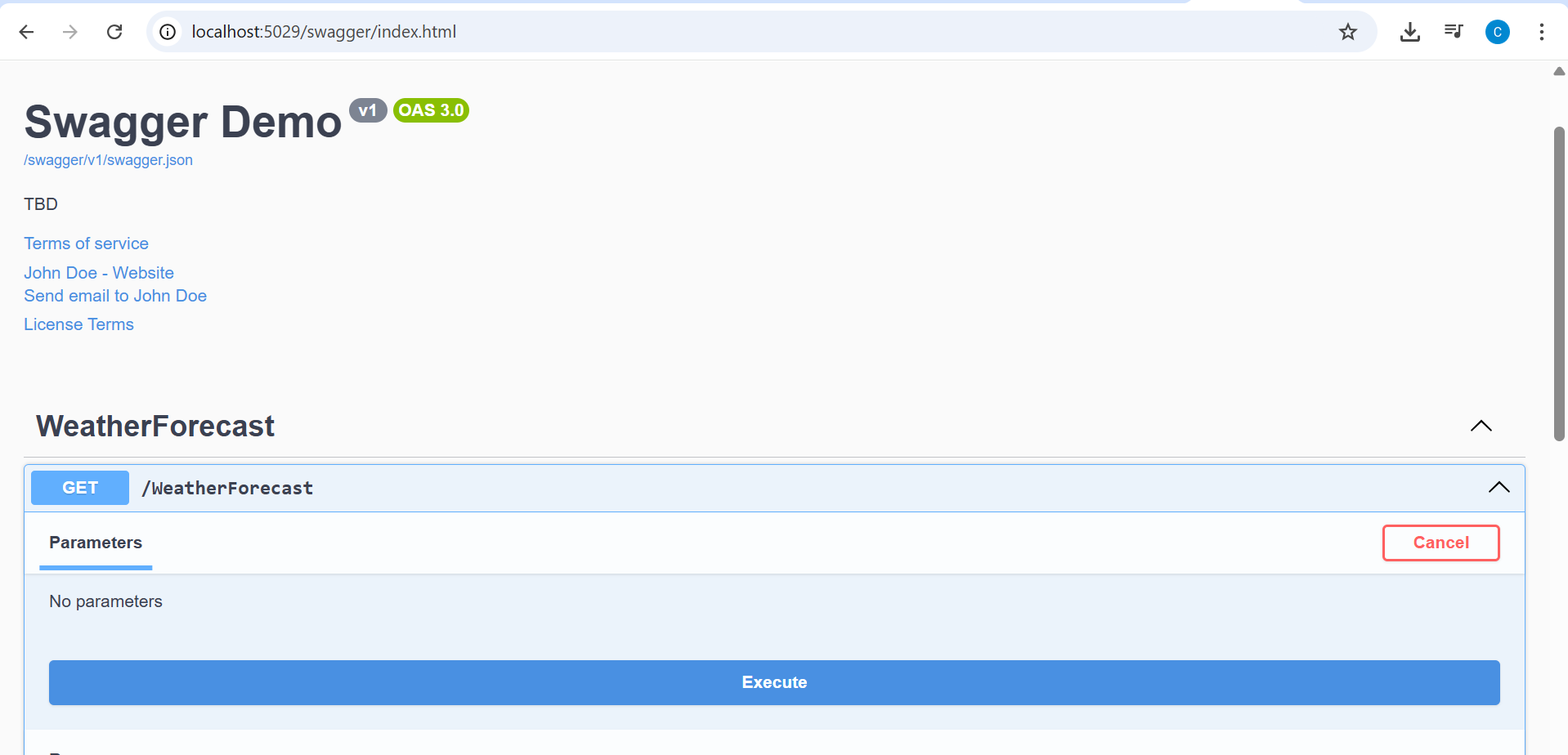
public string? Summary { get; set; }

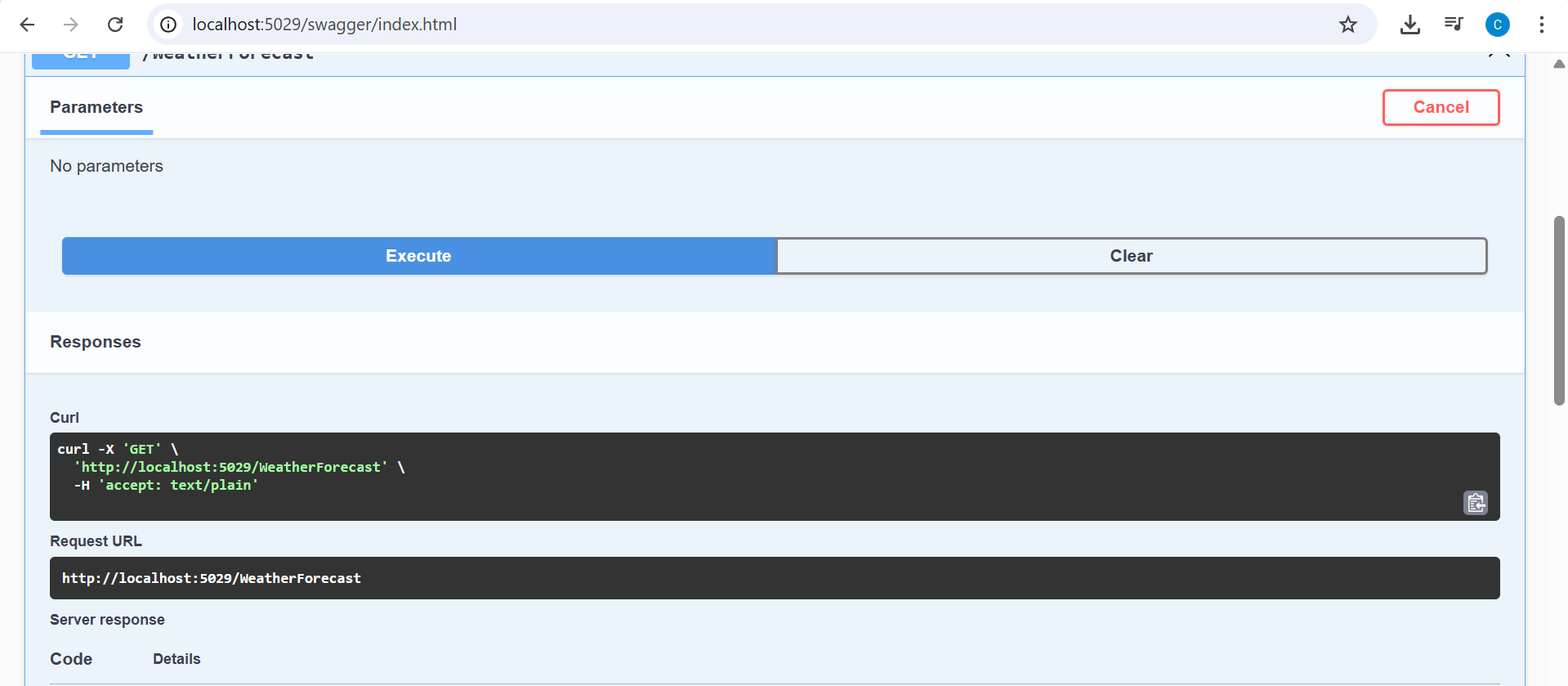
}

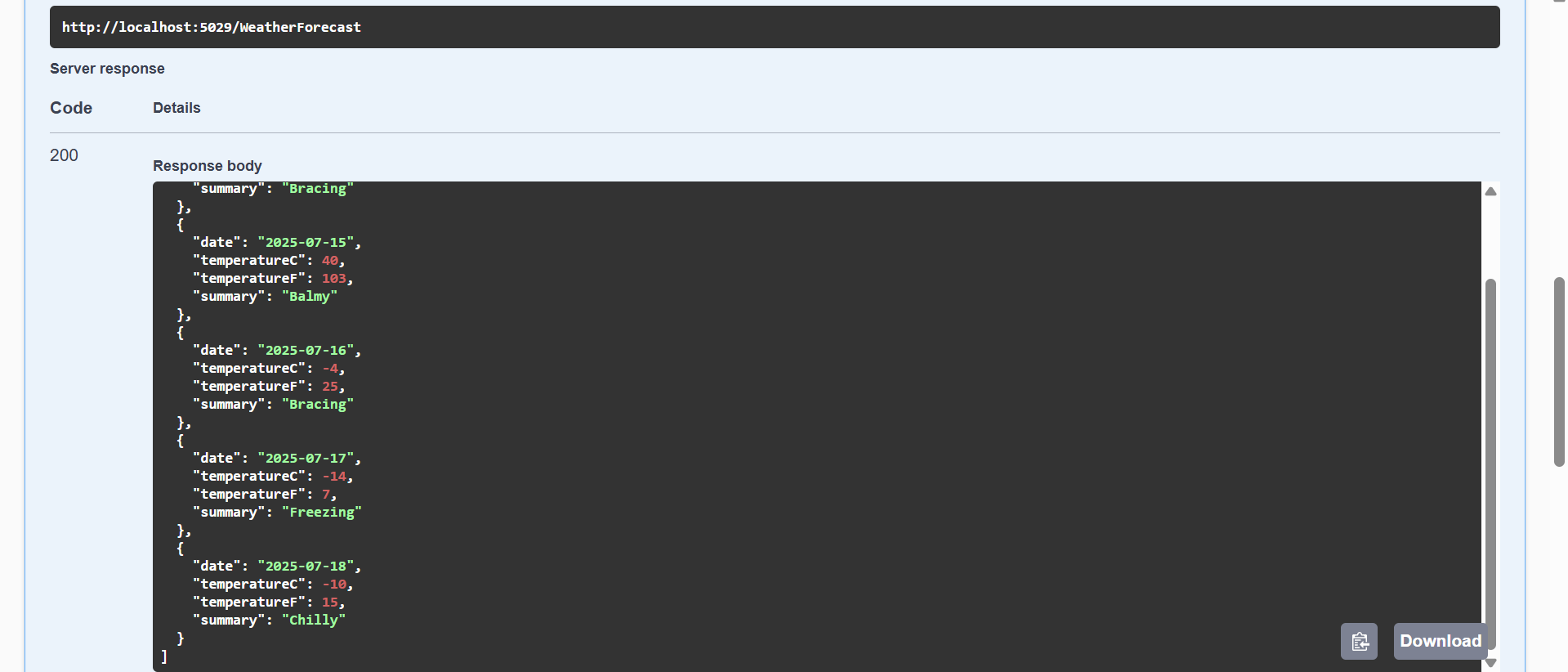
}

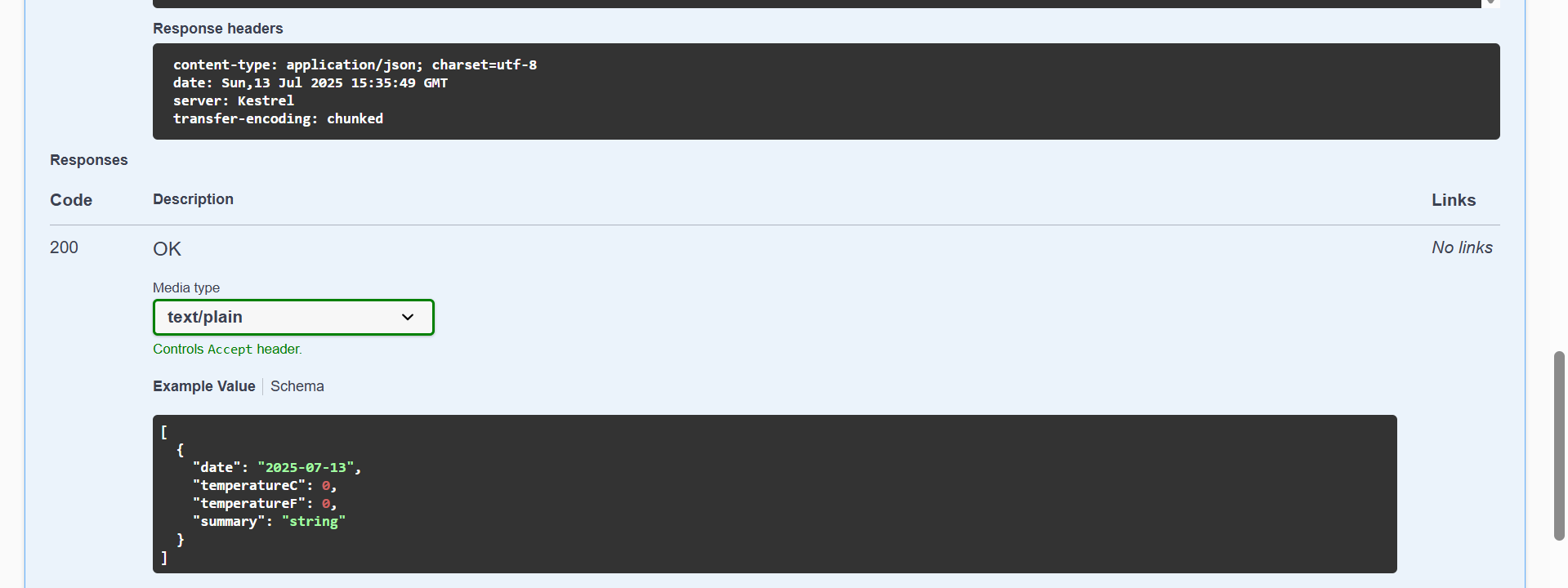




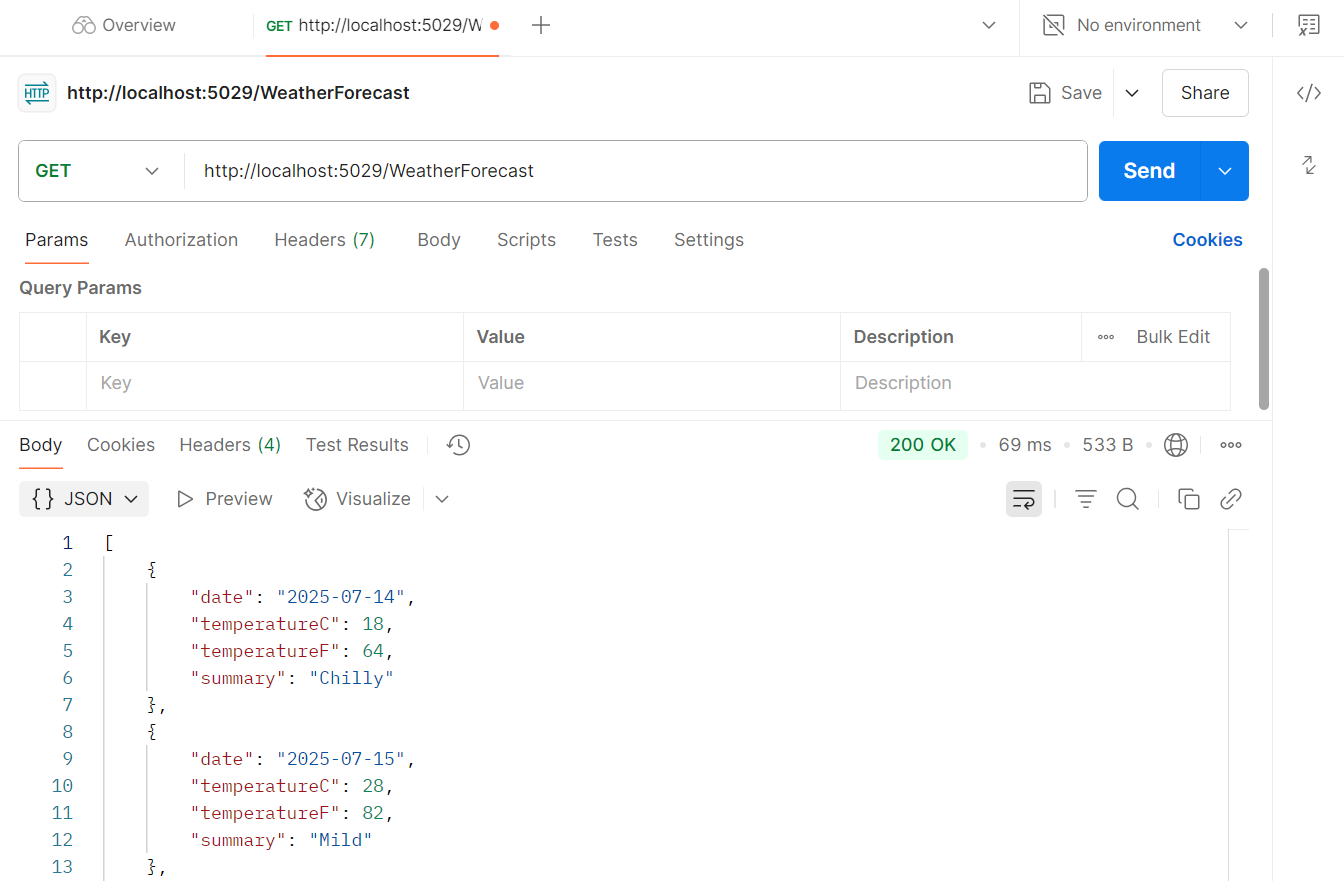








POSTMAN:



{}JSON:

[

    {

        "date": "2025-07-14",

        "temperatureC": 43,

        "temperatureF": 109,

        "summary": "Hot"

    },

    {

        "date": "2025-07-15",

        "temperatureC": 9,

        "temperatureF": 48,

        "summary": "Hot"

    },

    {

        "date": "2025-07-16",

        "temperatureC": 44,

        "temperatureF": 111,

        "summary": "Sweltering"

    },

    {

        "date": "2025-07-17",

        "temperatureC": 34,

        "temperatureF": 93,

        "summary": "Bracing"

    },

    {

        "date": "2025-07-18",

        "temperatureC": 31,

        "temperatureF": 87,

        "summary": "Freezing"

    }

]

**Q3) WebAPI\_3:**

EmployeeController.cs:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using EmployeeApi.Models;

[ApiController]

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

public class EmployeeController : ControllerBase

{

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice",

Salary = 50000,

Permanent = true,

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

Skills = new List<Skill> { new Skill { Id = 1, Name = "Communication" } },

DateOfBirth = new DateTime(1990, 1, 1)

}

// Add more sample employees as needed

};

}

[AllowAnonymous]

[HttpGet]

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

public ActionResult<List<Employee>> Get()

{

return Ok(GetStandardEmployeeList());

}

[HttpPost]

public IActionResult Post([FromBody] Employee employee)

{

// Add logic to save employee

return Ok(employee);

}

[HttpPut]

public IActionResult Put([FromBody] Employee employee)

{

// Add logic to update employee

return Ok(employee);

}

}

Employee.cs:

namespace EmployeeApi.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; }

}

}

Department.cs:

namespace EmployeeApi.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 EmployeeApi.Models

{

public class Skill

{

public int Id { get; set; }

public string Name { get; set; }

}

}

Program.cs:

using Microsoft.Extensions.DependencyInjection;

using Swashbuckle.AspNetCore.SwaggerGen;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi

builder.Services.AddOpenApi();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.MapOpenApi();

}

app.UseAuthorization();

app.MapControllers();

if (app.Environment.IsDevelopment())

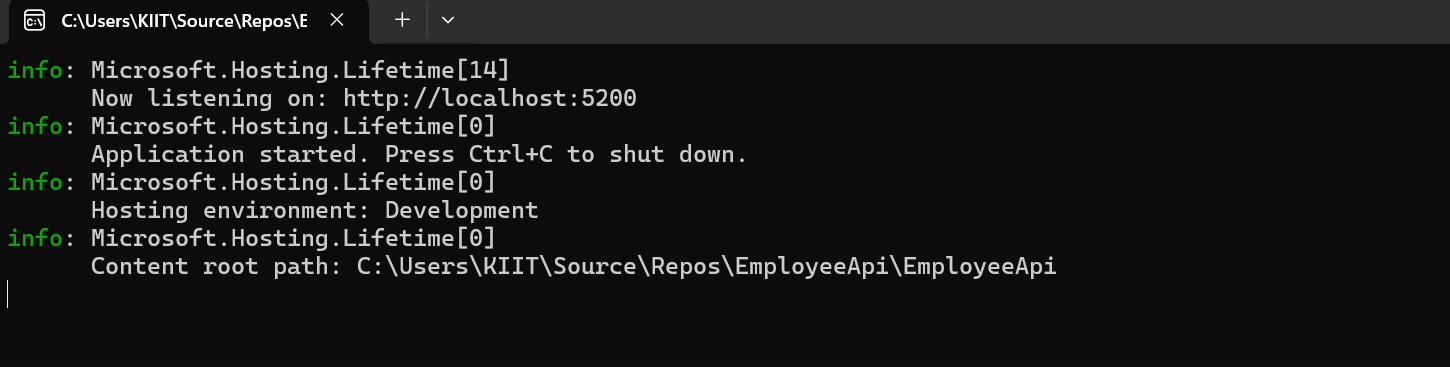
{

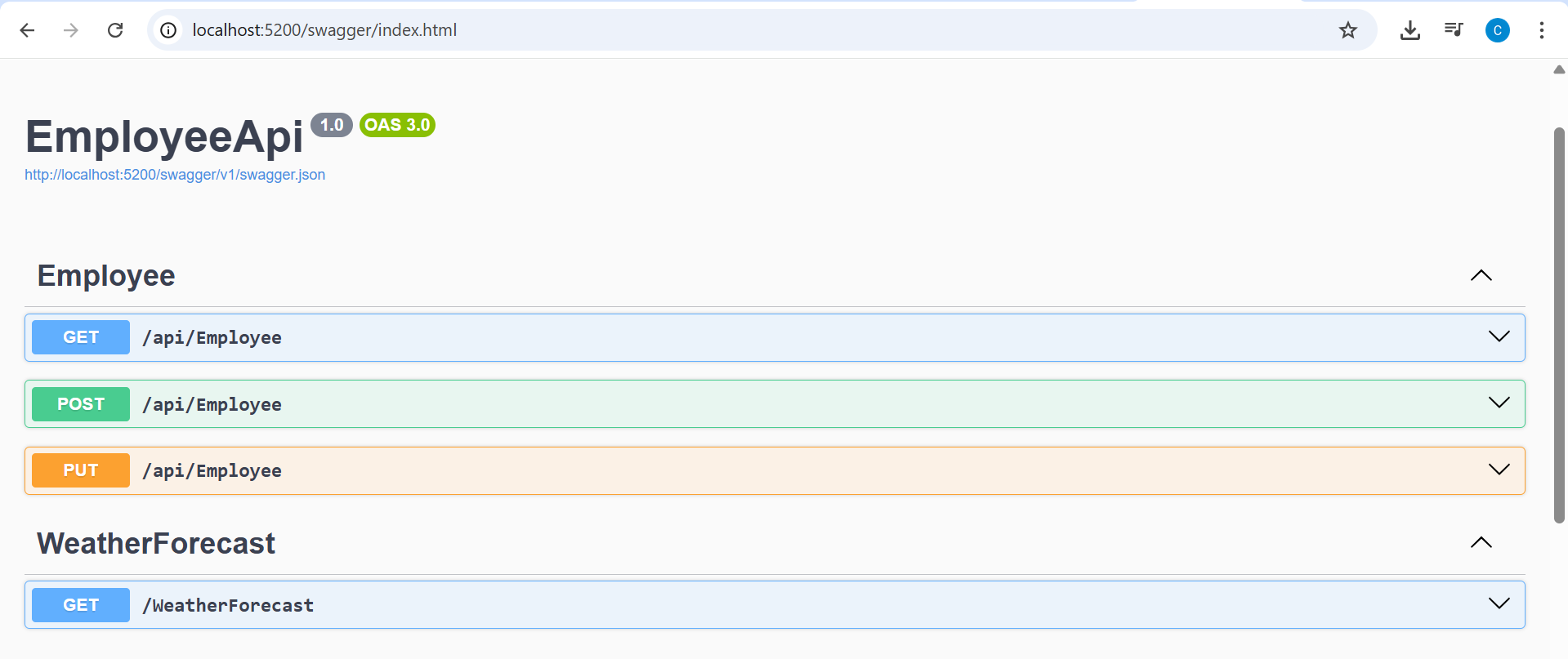
app.UseSwagger();

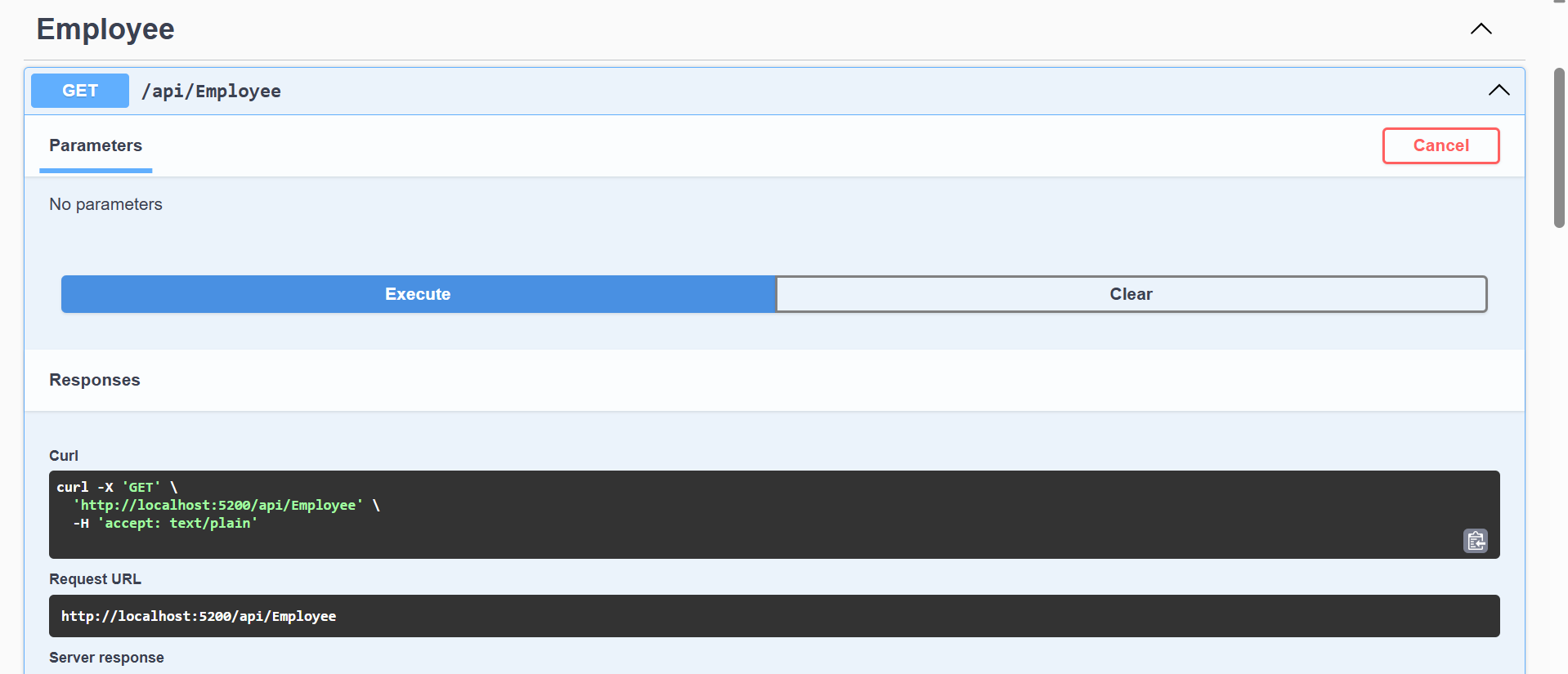
app.UseSwaggerUI();

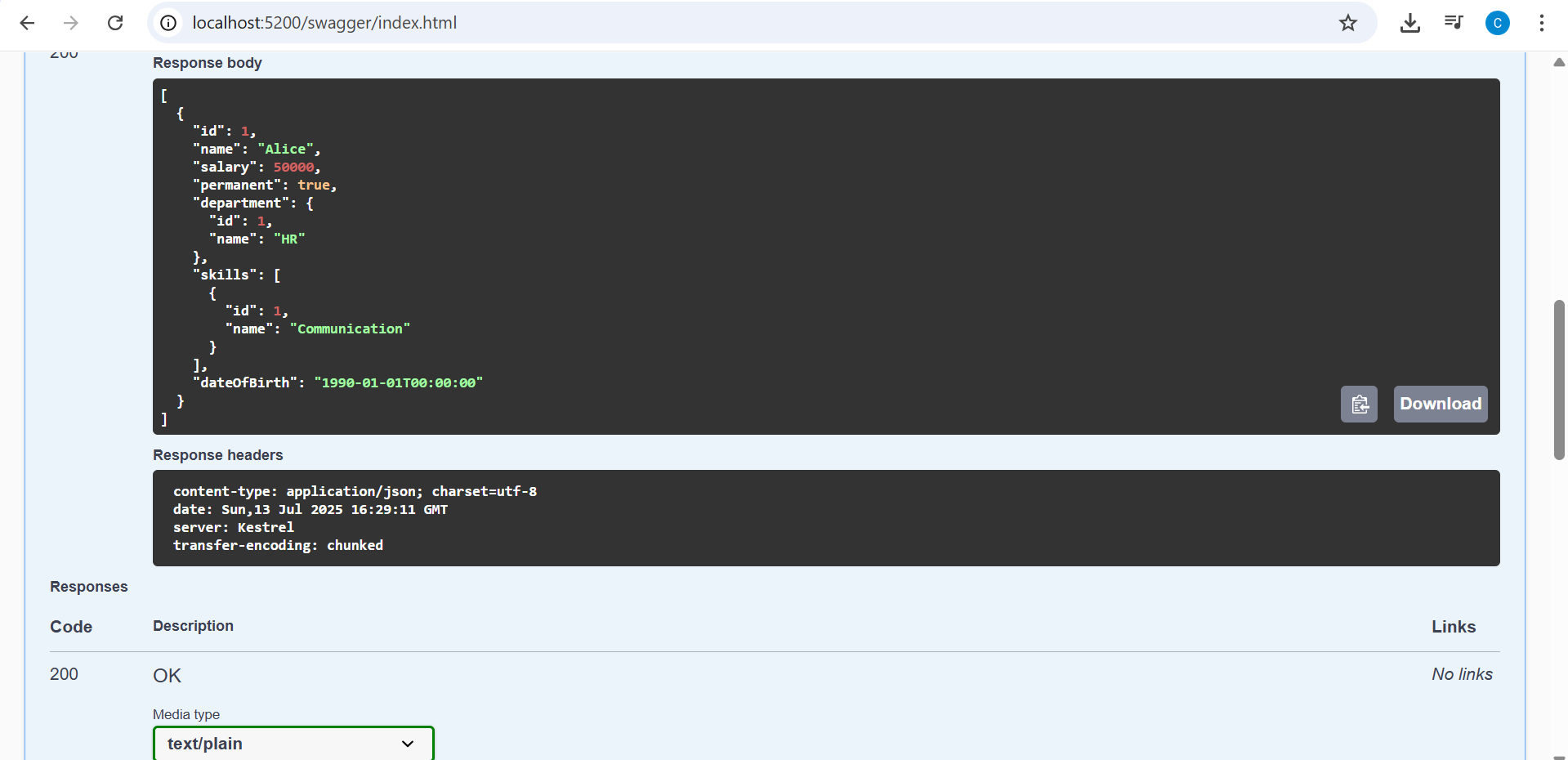
}

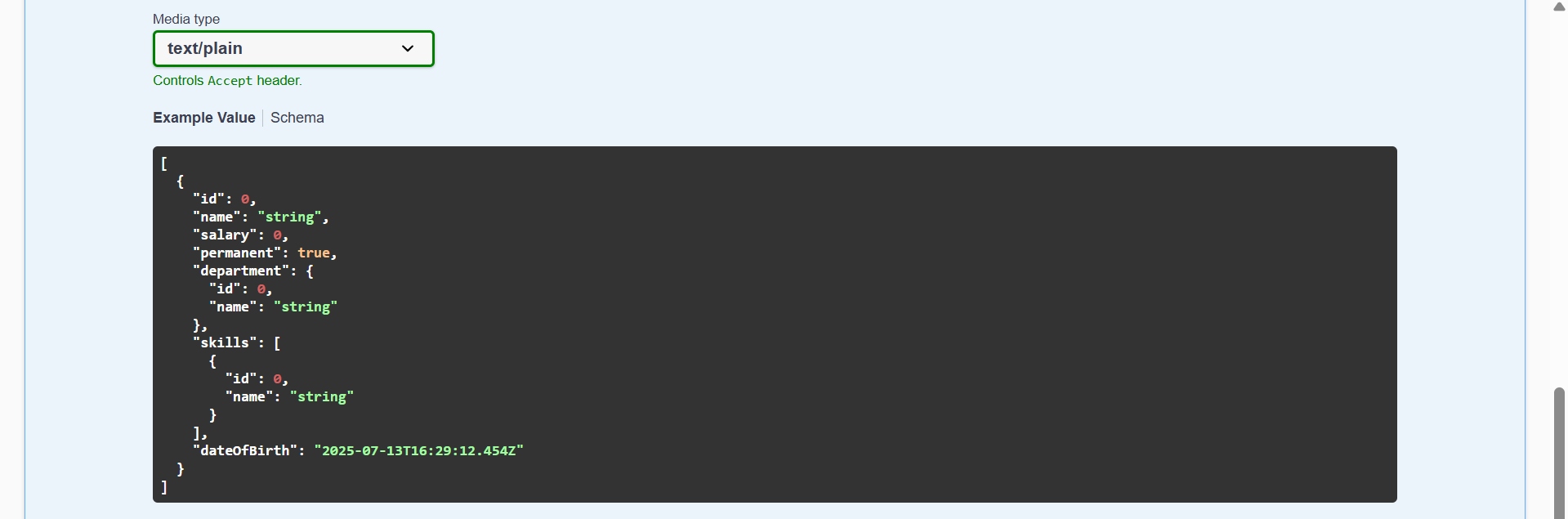
app.Run();











**WebAPI4:**

EmployeeController.cs:

using EmployeeApi2.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Linq;

[ApiController]

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

public class EmployeesController : ControllerBase

{

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

{

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

new Employee { Id = 2, Name = "Jane", Department = "IT", Salary = 60000 },

new Employee { Id = 3, Name = "Bob", Department = "Finance", Salary = 55000 }

};

[HttpPut("{id}")]

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

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (employee == null)

{

return BadRequest("Invalid employee id");

}

// Update the employee details

employee.Name = updatedEmployee.Name;

employee.Department = updatedEmployee.Department;

employee.Salary = updatedEmployee.Salary;

return Ok(employee);

}

}

Program.cs:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddOpenApi();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.MapOpenApi();

}

app.UseAuthorization();

app.MapControllers();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(options =>

{

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

// options.RoutePrefix = string.Empty; // Optional: Swagger UI at root

});

}

app.Run();

Employee.cs:

namespace EmployeeApi2.Models

{

public class Employee

{

public int Id { get; set; }

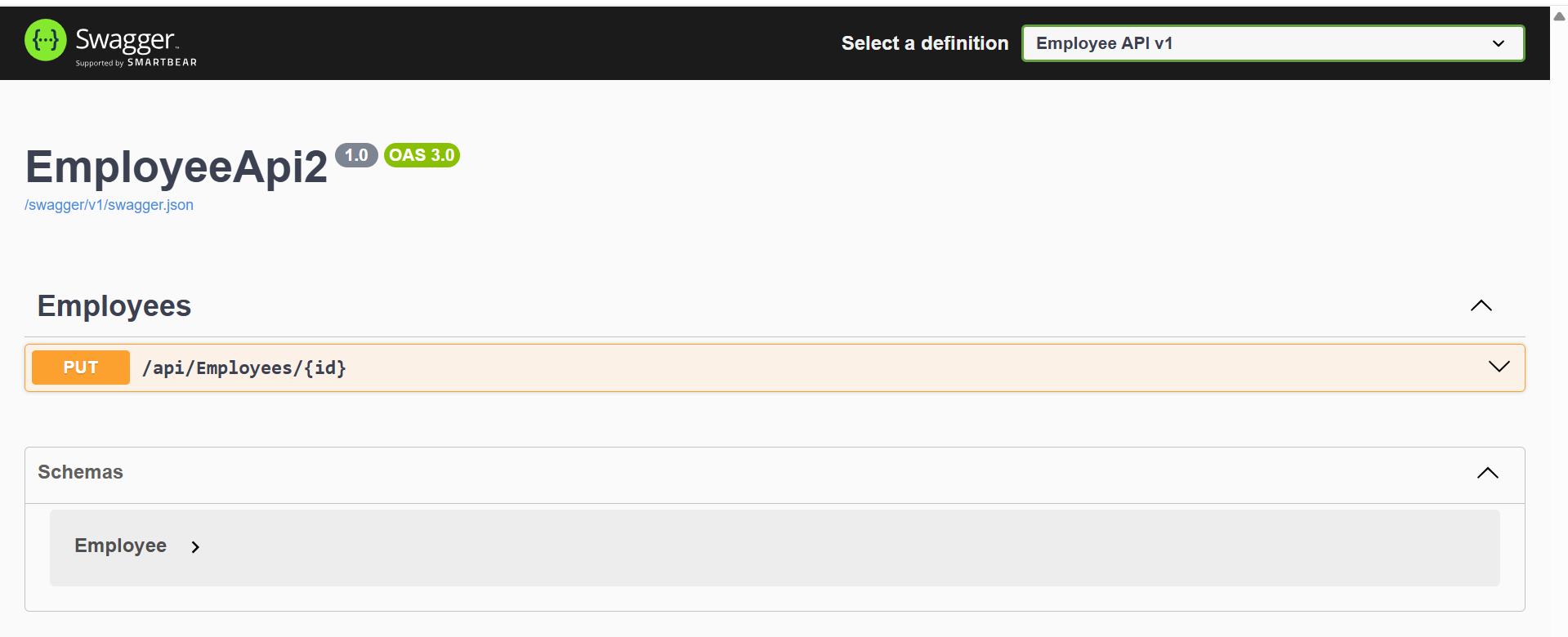
public string Name { get; set; }

public string Department { get; set; }

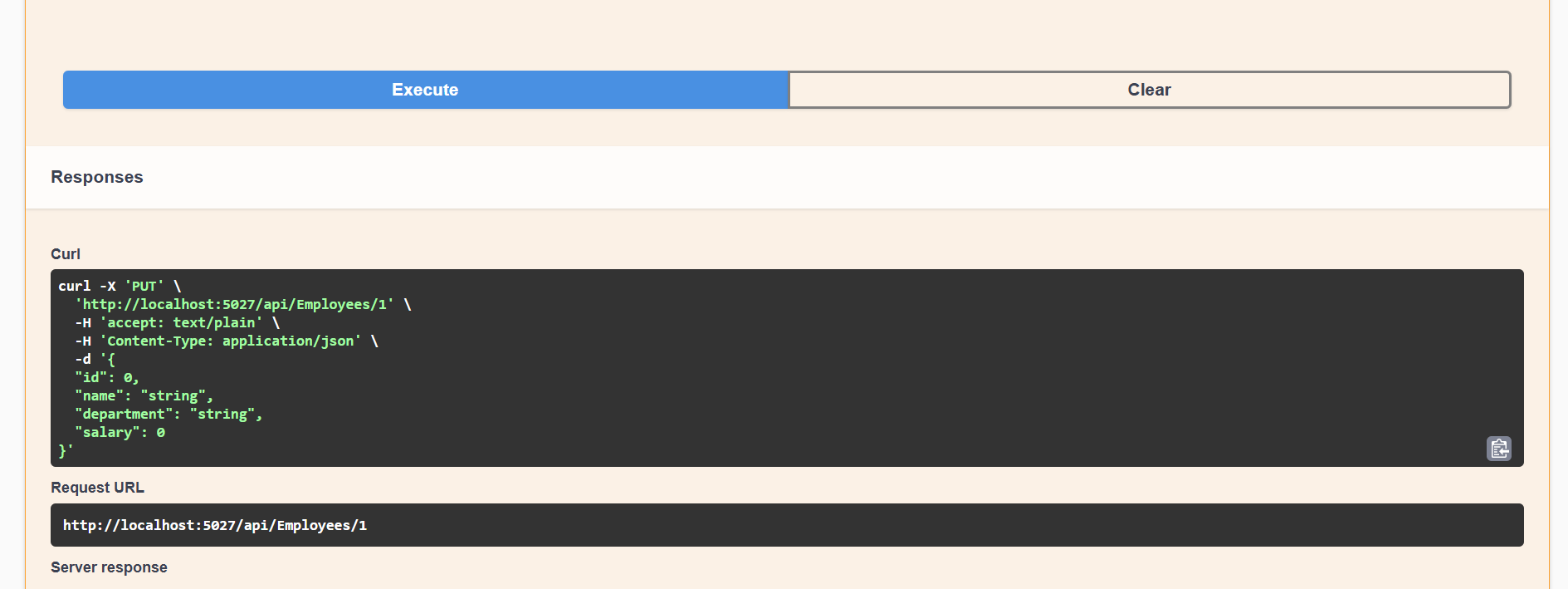
public double Salary { get; set; }

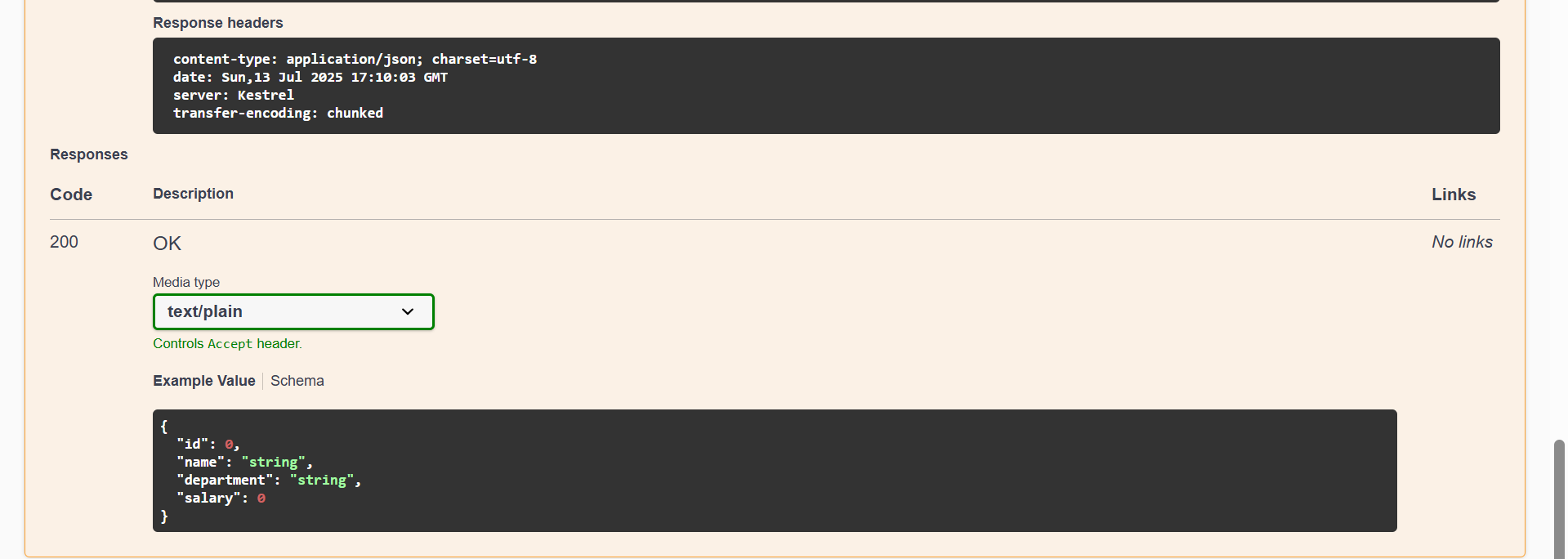
}

}









**WebAPI5:**

What is CORS?

Browsers use a security feature called CORS (Cross-Origin Resource Sharing) to prevent web apps that are operating at one origin from interacting with resources from another origin unless the server specifically permits it.

When your API runs on localhost:5000 and your frontend (such as React, Angular, or plain HTML/JS) runs on localhost:3000, it is imperative.

How to enable CORS thru Startup.cs, Install Cors nuget package to Web API application:

Launch the NuGet Package Manager, then install:

Microsoft.AspNetCore.Cors

Program.cs:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer(); // Required for Swagger UI

builder.Services.AddSwaggerGen(); // Required for Swagger UI

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowLocalhost",

policy =>

{

policy.WithOrigins("http://localhost:3000")

.AllowAnyHeader()

.AllowAnyMethod();

});

});

var app = builder.Build();

// Configure the HTTP request pipeline

if (app.Environment.IsDevelopment())

{

app.UseSwagger(); // Enables Swagger JSON endpoint

app.UseSwaggerUI(); // Enables Swagger UI at /swagger

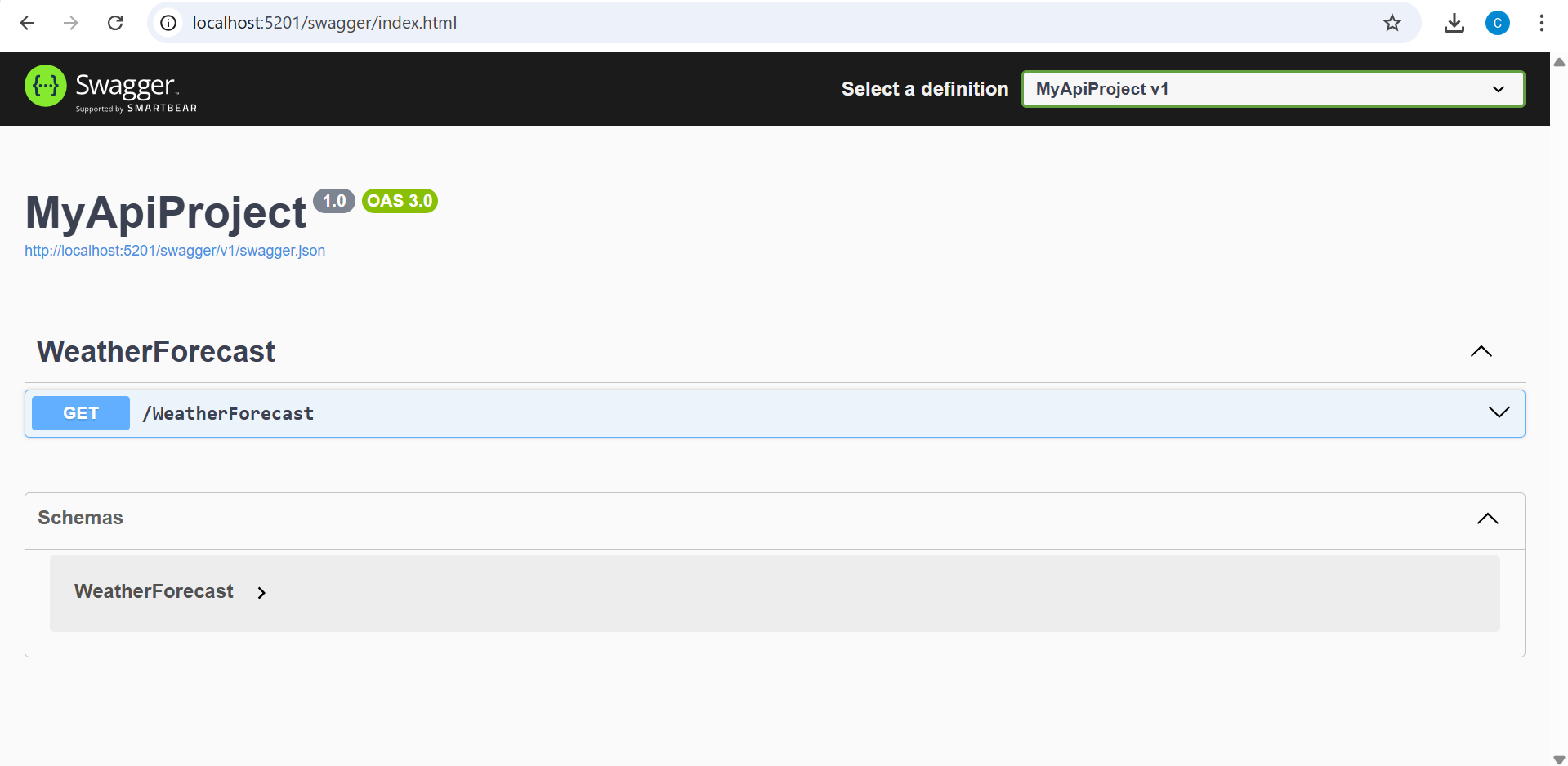
}

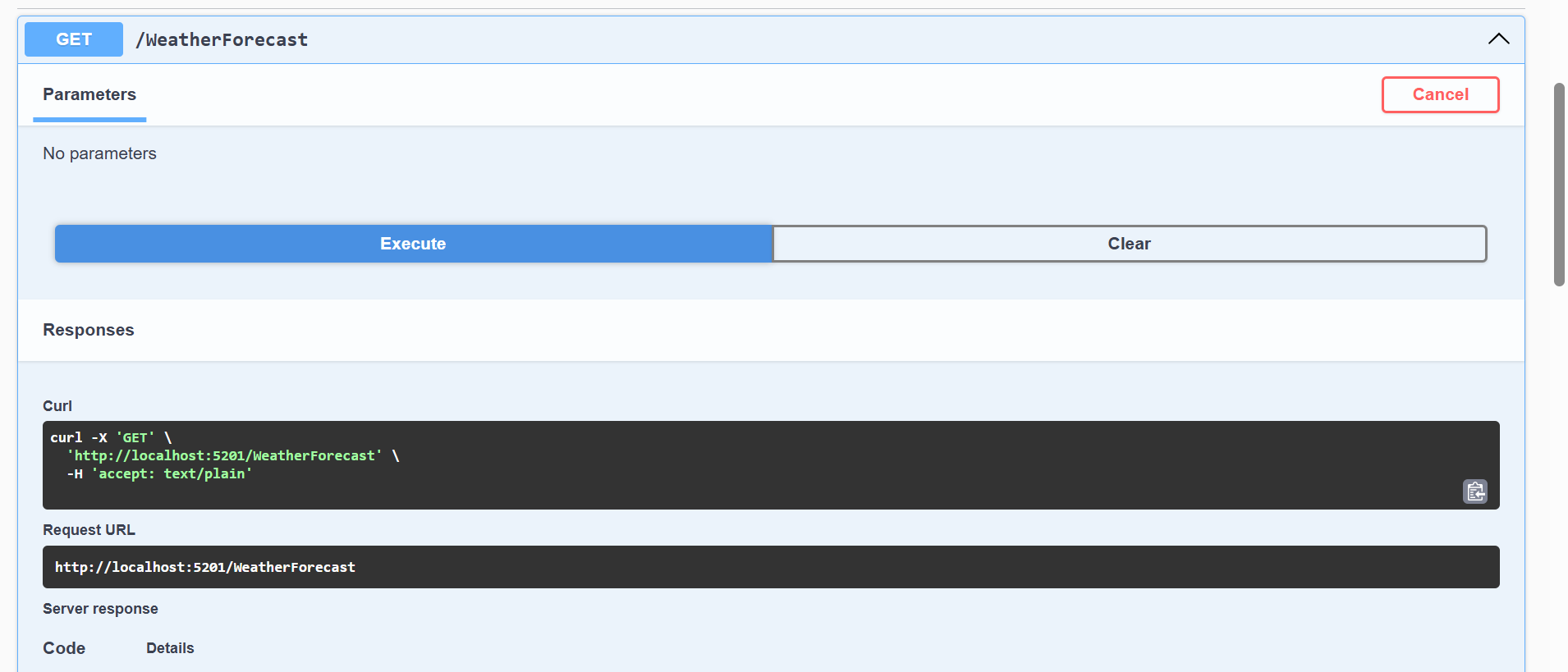
app.UseCors("AllowLocalhost");

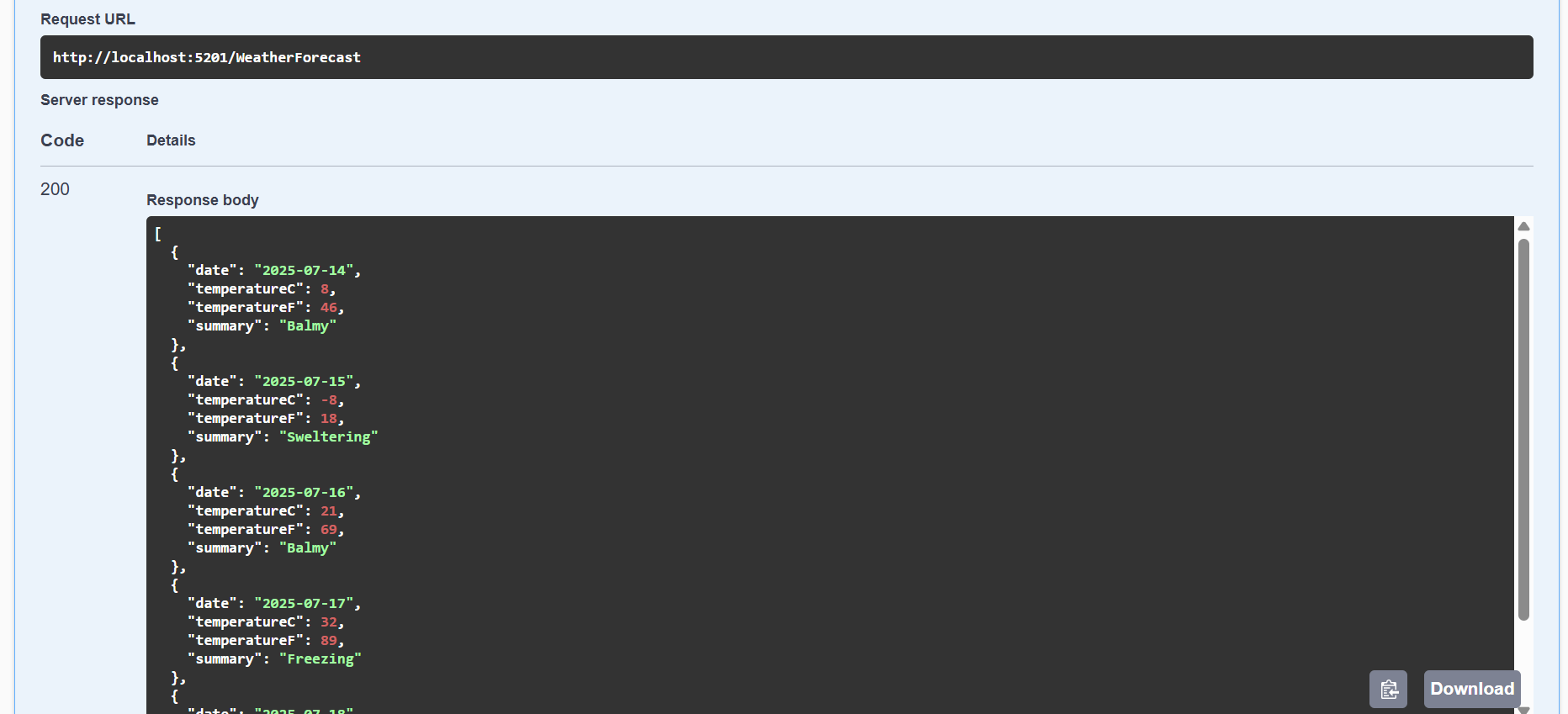
app.UseAuthorization();

app.MapControllers();

app.Run();

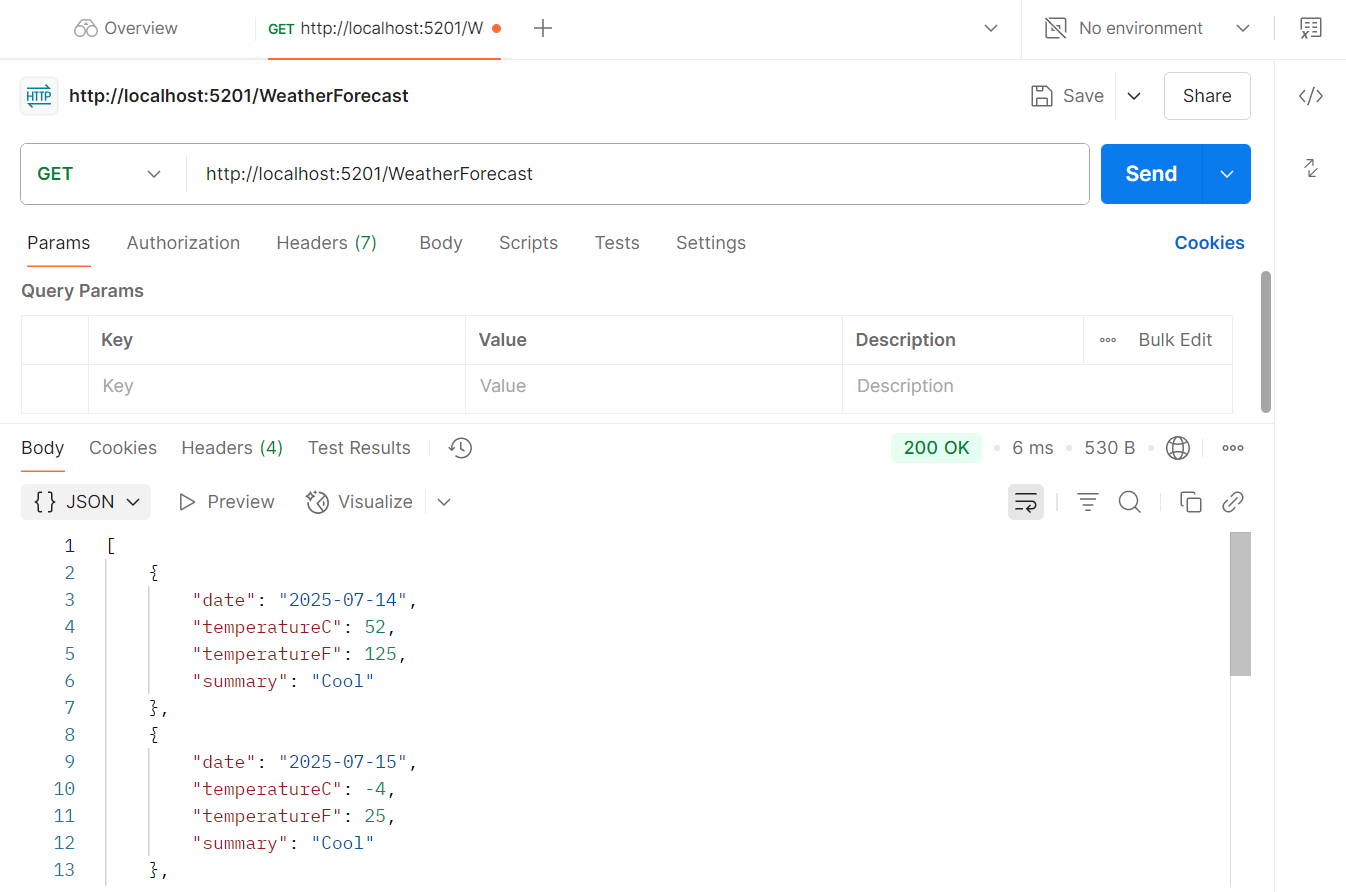








POSTMAN:



{}JSON:

[

    {

        "date": "2025-07-14",

        "temperatureC": 52,

        "temperatureF": 125,

        "summary": "Cool"

    },

    {

        "date": "2025-07-15",

        "temperatureC": -4,

        "temperatureF": 25,

        "summary": "Cool"

    },

    {

        "date": "2025-07-16",

        "temperatureC": 13,

        "temperatureF": 55,

        "summary": "Balmy"

    },

    {

        "date": "2025-07-17",

        "temperatureC": 5,

        "temperatureF": 40,

        "summary": "Cool"

    },

    {

        "date": "2025-07-18",

        "temperatureC": 52,

        "temperatureF": 125,

        "summary": "Freezing"

    }

]