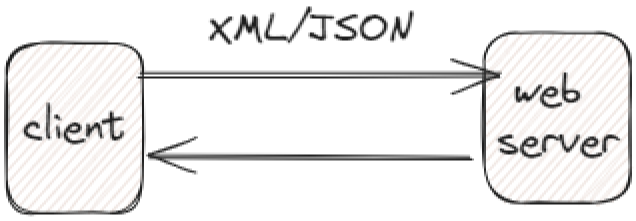
Asp.net Core Web Api Notes

API clients communicate with the server over HTTP, and the two exchange information by using a data format such as JSON or XML



Web service APIs that adhere to REST are called RESTful APIs. They're defined through:

- A base URI.
- HTTP methods, such as `GET`, `POST`, `PUT`, `PATCH`, or `DELETE`.
- A media type for the data, such as JavaScript Object Notation (JSON) or XML.

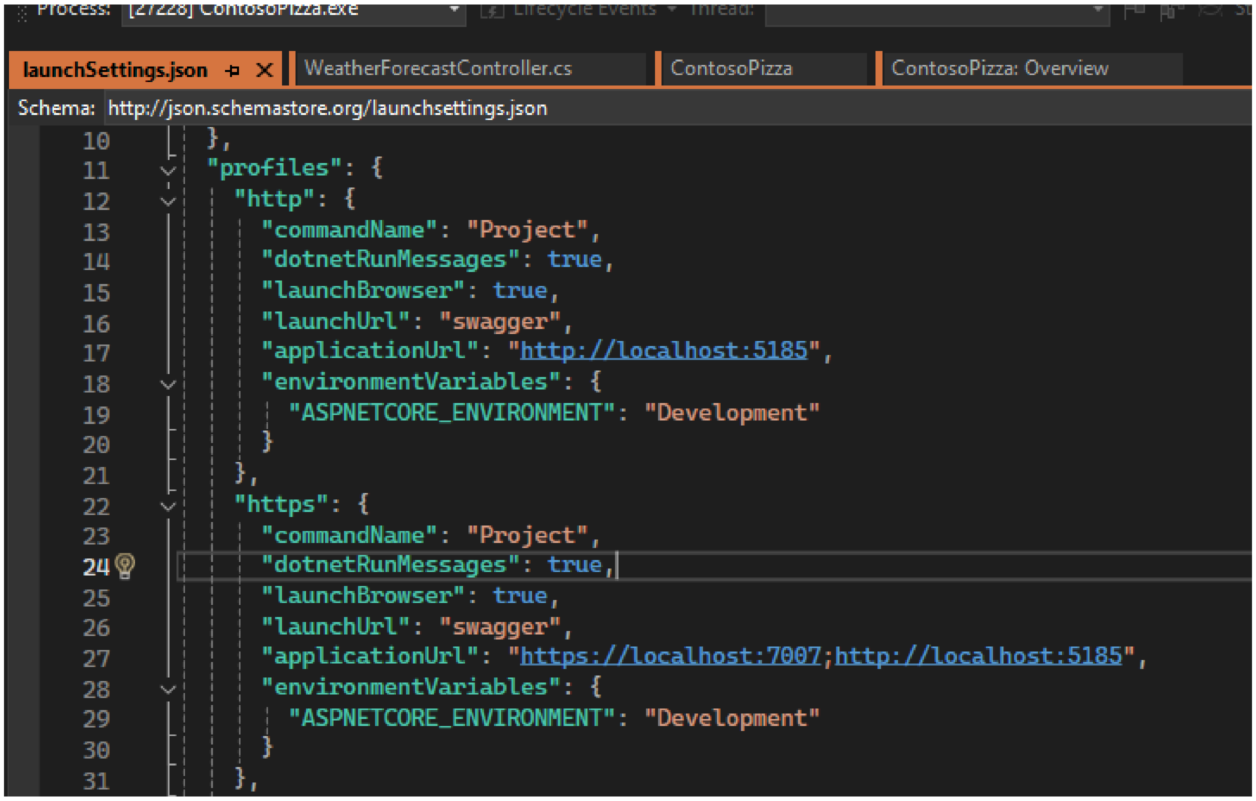
One of the easiest options for exploring and interacting with web APIs is the .NET HTTP REPL. REPL stands for *read-evaluate-print loop*. It's a simple and popular way to build interactive command-line environments. In the next unit, you create a simple web API and then interact with it by using the .NET HTTP REPL.

Step 1: Create a new Asp.net Web Api projcet with a name

- While creating project, Use .NET SDK 8.
- Don’t forgot to click the “use controllers” checkbox.
- After creating, Run the project

When you build and run the applicatin for the first time, A port from 5000 to 5300 is selected for HTTP, and from 7000 to 7300 for HTTPS, when the project is created. You can easily change the ports that you use during development by editing the project's *launchSettings.json* file. This module uses the secure `localhost` URL that begins with `https`. You can find the launchsettings.json file inside properties folder.

Copt the https url showing in launchsettings.json and open in browser.

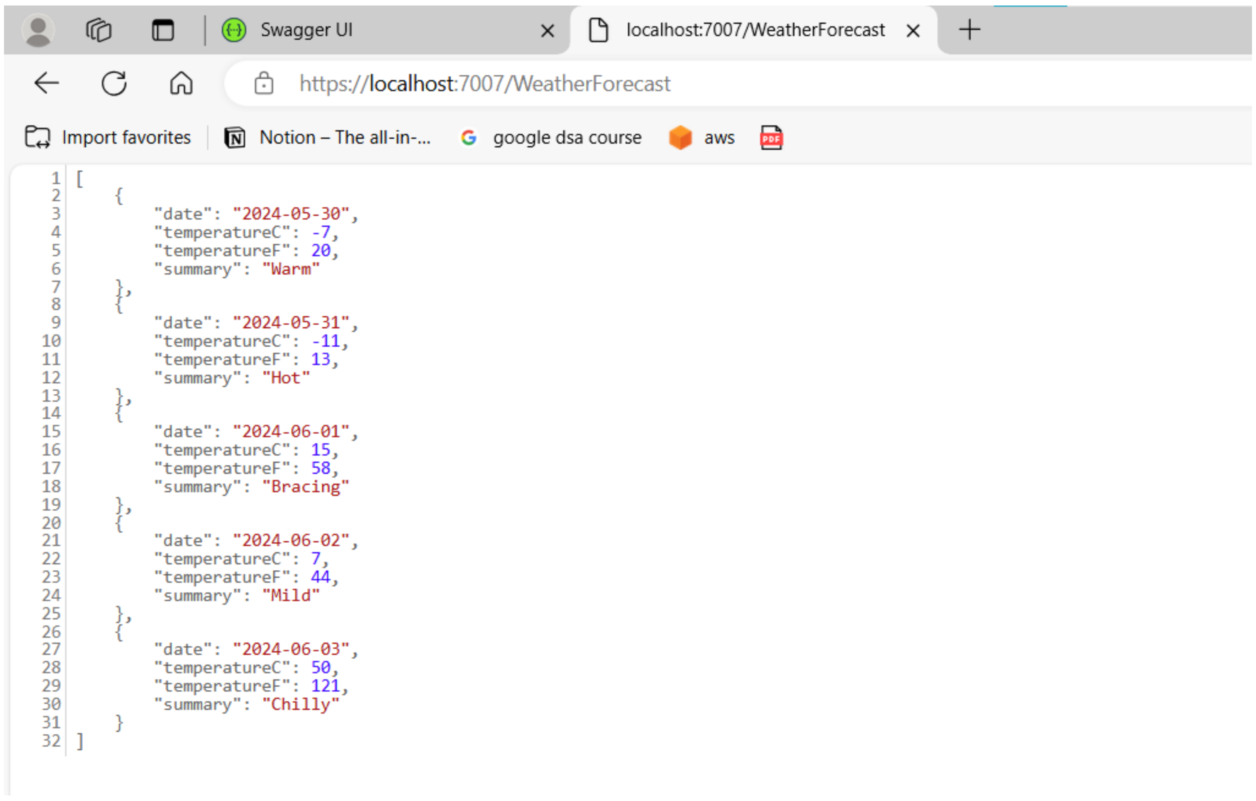


Schema: http://json.schemastore.org/launchsettings.json

```
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5185",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:7007;http://localhost:5185",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
```

For me, Port 7007 is allocated to run this application using https.

You will set a web page like below.



```
[
    {
        "date": "2024-05-30",
        "temperatureC": -7,
        "temperatureF": 20,
        "summary": "Warm"
    },
    {
        "date": "2024-05-31",
        "temperatureC": -11,
        "temperatureF": 13,
        "summary": "Hot"
    },
    {
        "date": "2024-06-01",
        "temperatureC": 15,
        "temperatureF": 58,
        "summary": "Bracing"
    },
    {
        "date": "2024-06-02",
        "temperatureC": 7,
        "temperatureF": 44,
        "summary": "Mild"
    },
    {
        "date": "2024-06-03",
        "temperatureC": 50,
        "temperatureF": 121,
        "summary": "Chilly"
    }
]
```

There are two other ways to test the api.

Using HTTPREPL

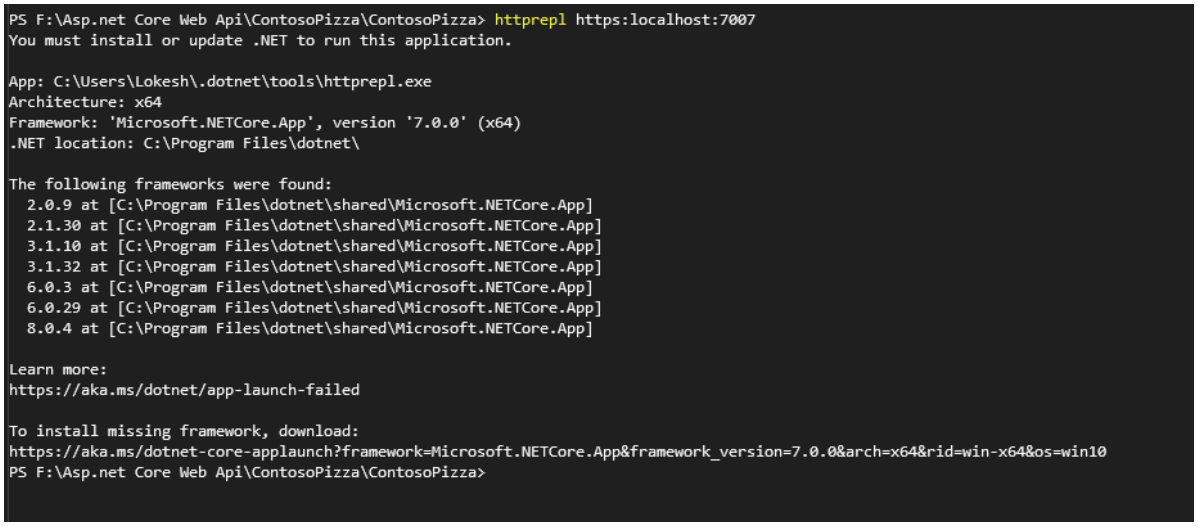
- Download httprepl using the below command

```
dotnet tool install -g Microsoft.dotnet-httprepl
```

- After installing,

  Run the following command (Replace the port if needed).

```
httprepl https:localhost:7007
```



```
PS F:\Asp.net Core Web Api\ContosoPizza\ContosoPizza> httprepl https:localhost:7007
You must install or update .NET to run this application.

App: C:\Users\Lokesh\.dotnet\tools\httprepl.exe
Architecture: x64
Framework: 'Microsoft.NETCore.App', version '7.0.0' (x64)
.NET location: C:\Program Files\dotnet\

The following frameworks were found:
  2.0.9 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  2.1.30 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  3.1.10 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  3.1.32 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  6.0.3 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  6.0.29 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]
  8.0.4 at [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]

Learn more:
https://aka.ms/dotnet/app-launch-failed

To install missing framework, download:
https://aka.ms/dotnet-core-applaunch?framework=Microsoft.NETCore.App&framework_version=7.0.0&arch=x64&rid=win-x64&os=win10
PS F:\Asp.net Core Web Api\ContosoPizza\ContosoPizza>
```

A controller is a public class with one or more public methods known as *actions*. By convention, a controller is placed in the project root's *Controllers* directory. The actions are exposed as HTTP endpoints via routing. So an HTTP `GET` request to `https://localhost:{PORT}/weatherforecast` causes the `Get()` method of the `WeatherForecastController` class to be executed.

😡 **Note: Don't create a web API controller by deriving from the `Controller` class. `Controller` derives from `ControllerBase` and adds support for views, so it's for handling webpages, not web API requests.**

API controller class attributes

Two important attributes are applied to `WeatherForecastController`, as shown in the following code:

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
```

`[ApiController]` enables opinionated behaviors that make it easier to build web APIs. Some behaviors include *parameter source inference*, *attribute routing as a requirement*, and *model validation error-handling*enhancements*.

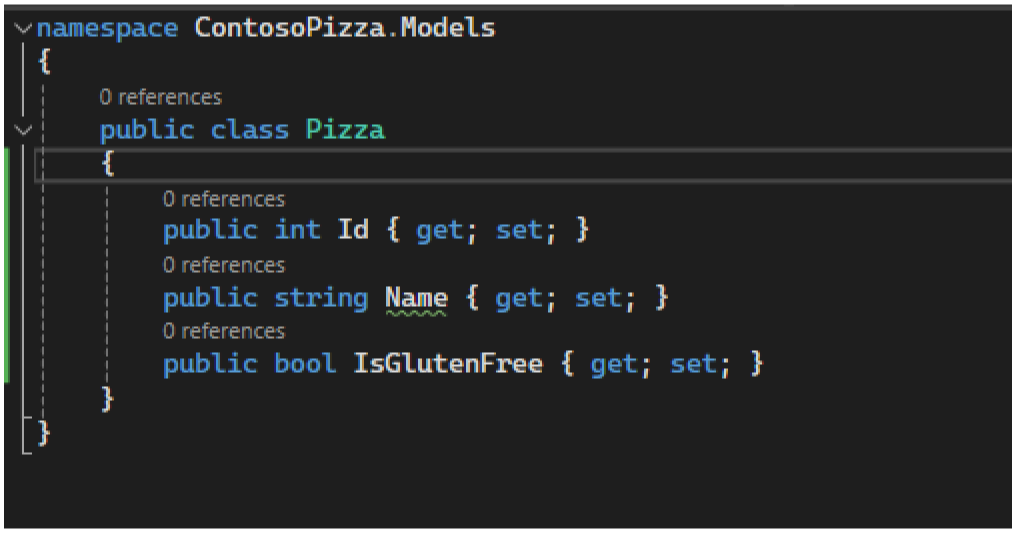
`[Route]` defines the routing pattern `[controller]`. The controller's name (case-insensitive, without the *Controller* suffix) replaces the `[controller]` token. This controller handles requests to `https://localhost:{PORT}/weatherforecast`.

The route might contain static strings, as in `api/[controller]`. In this example, this controller would handle a request to `https://localhost:{PORT}/api/weatherforecast`.

Step 2: Add Model

Create a folder named **Models.** Inside models, create a class "Pizza".

The Pizza.cs file should look like this:



```
namespace ContosoPizza.Models
{
    0 references
    public class Pizza
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Name { get; set; }
        0 references
        public bool IsGlutenFree { get; set; }
    }
}
```

Step 3: Create a folder Services for data services.

💡 Note: We are using local-cache.

Here is the code of PizzaService class.

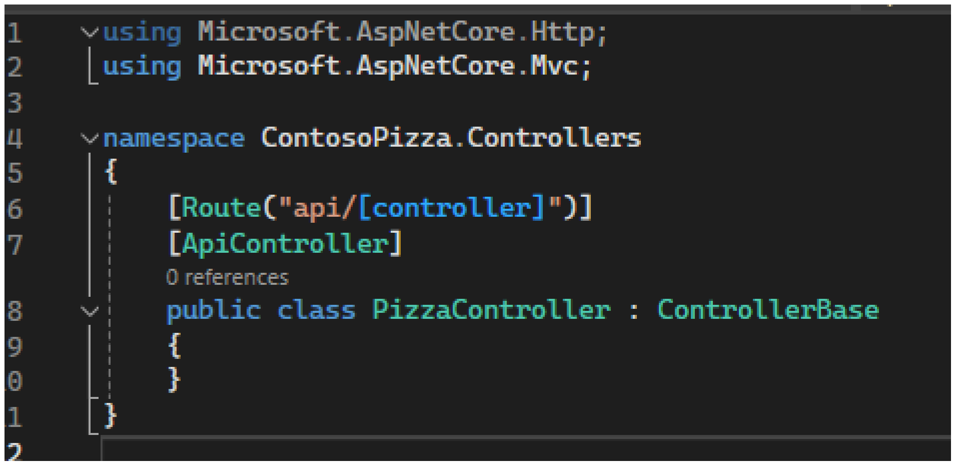
```
using ContosoPizza.Models;

namespace ContosoPizza.Services
{
    public static class PizzaService
    {
        static List<Pizza> Pizzas { get; set; }
        static int nextId = 3;

        // Constructor
        static PizzaService()
        {
            Pizzas = new List<Pizza>()
            {
                new Pizza()
                {
                    Id = 1,
                    Name = "Classic Italian",
                    IsGlutenFree = false
                },
                new Pizza()
                {
                    Id = 2,
                    Name = "Veggie",
                    IsGlutenFree = true
                }
            };
        }

        //Get All
        public static List<Pizza> GetAll()
        {

            return Pizzas;
        }

        //Get by Id
        public static Pizza? Get(int id)
        {
            return Pizzas.FirstOrDefault(x => x.Id == id);
        }

        //Delete
        public static void Delete(int id)
        {
            Pizza item = Pizzas.FirstOrDefault(x => x.Id == id)
            if(item is null)
            {
                return;
            }
            Pizzas.Remove(item);
        }

        //Update
        public static void Update(int id,  Pizza pizza)
        {
            //// Get the pizza using id,
            //Pizza item = Pizzas.Find(x => x.Id == id);
            //// update  the properties of pizza,
            //item.Name = pizza.Name;
            //item.IsGlutenFree = pizza.IsGlutenFree;

            //or

            // find the index of pizza you want to update,
            // replace old one with new one.
            var index = Pizzas.FindIndex(x => x.Id == pizza.Id)
            if(index == -1)
            {
                return;
            }
            Pizzas[index] = pizza;
        }

    }
}
```

This service provides a simple in-memory data caching service with two pizzas by default. Our web API uses that service for demo purposes. When you stop and start the web API, the in-memory data cache is reset to the two default pizzas from the constructor of `PizzaService`.

Step 3: Create PizzaController

Inside **Contollers** folder, Create a new controller "PizzaController".

💡 Note: Controller type is : ApiController



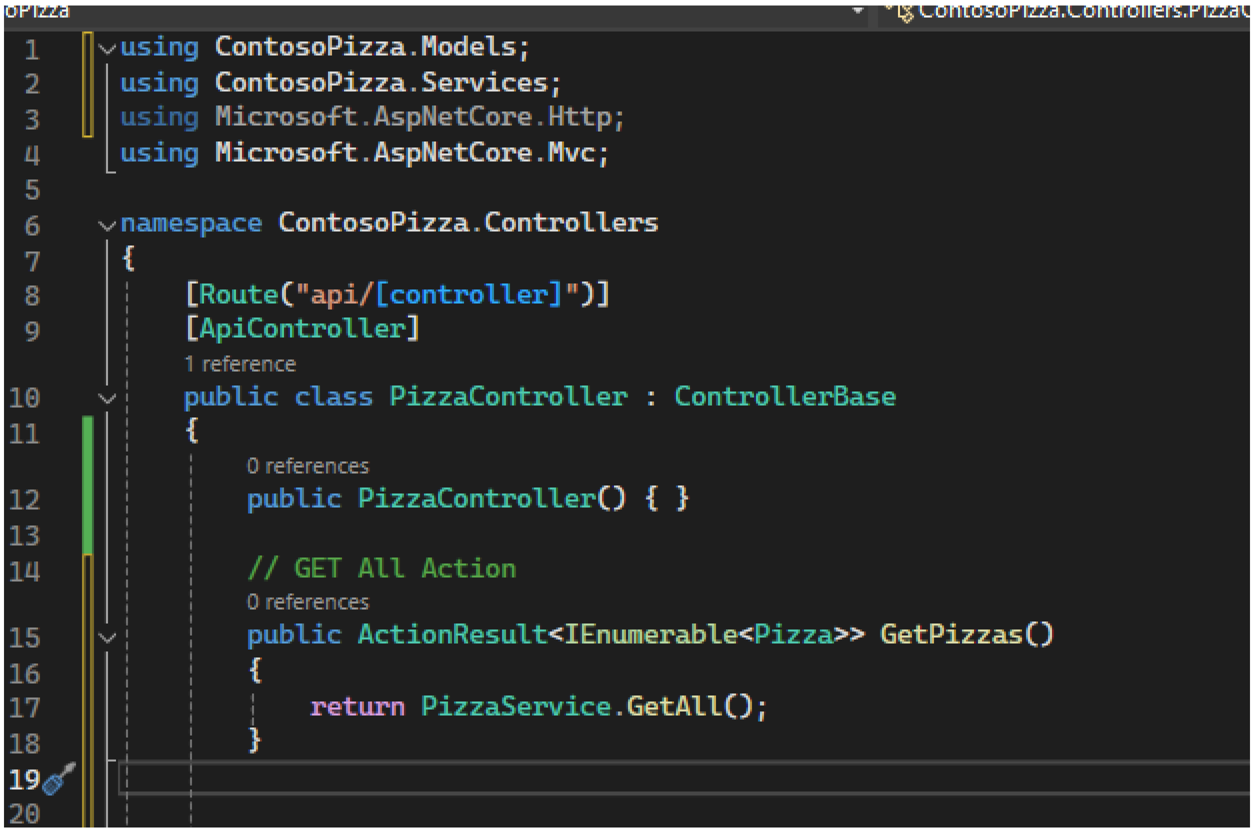
```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace ContosoPizza.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    0 references
    public class PizzaController : ControllerBase
    {
    }
}
```

💡 You can change the route attribute as needed

```
[Route("api/[controller]")]
//For this, url will be like: https:localhost/7007/api/{Cont

[Route("[controller]")]
//For this, url will be like: https:localhost/7007/{Controll
```

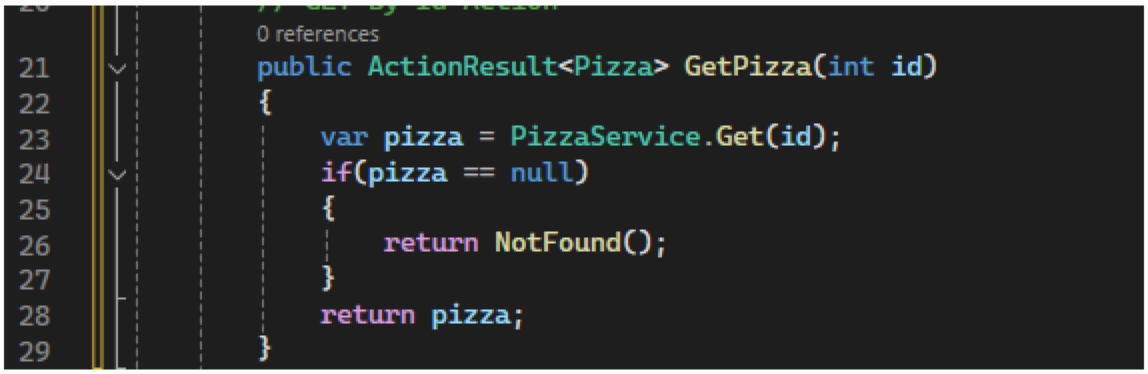
`[Route]` attribute defines a mapping to the `[controller]` token. Because this controller class is named `PizzaController`, this controller handles requests to `https://localhost:{PORT}/pizza .`

Note: You can replace the [controller] token with any name. This name will be reflected in url.



```
using ContosoPizza.Models;
using ContosoPizza.Services;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace ContosoPizza.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class PizzaController : ControllerBase
    {
        public PizzaController() { }

        // GET All Action
        public ActionResult<IEnumerable<Pizza>> GetPizzas()
        {
            return PizzaService.GetAll();
        }

```

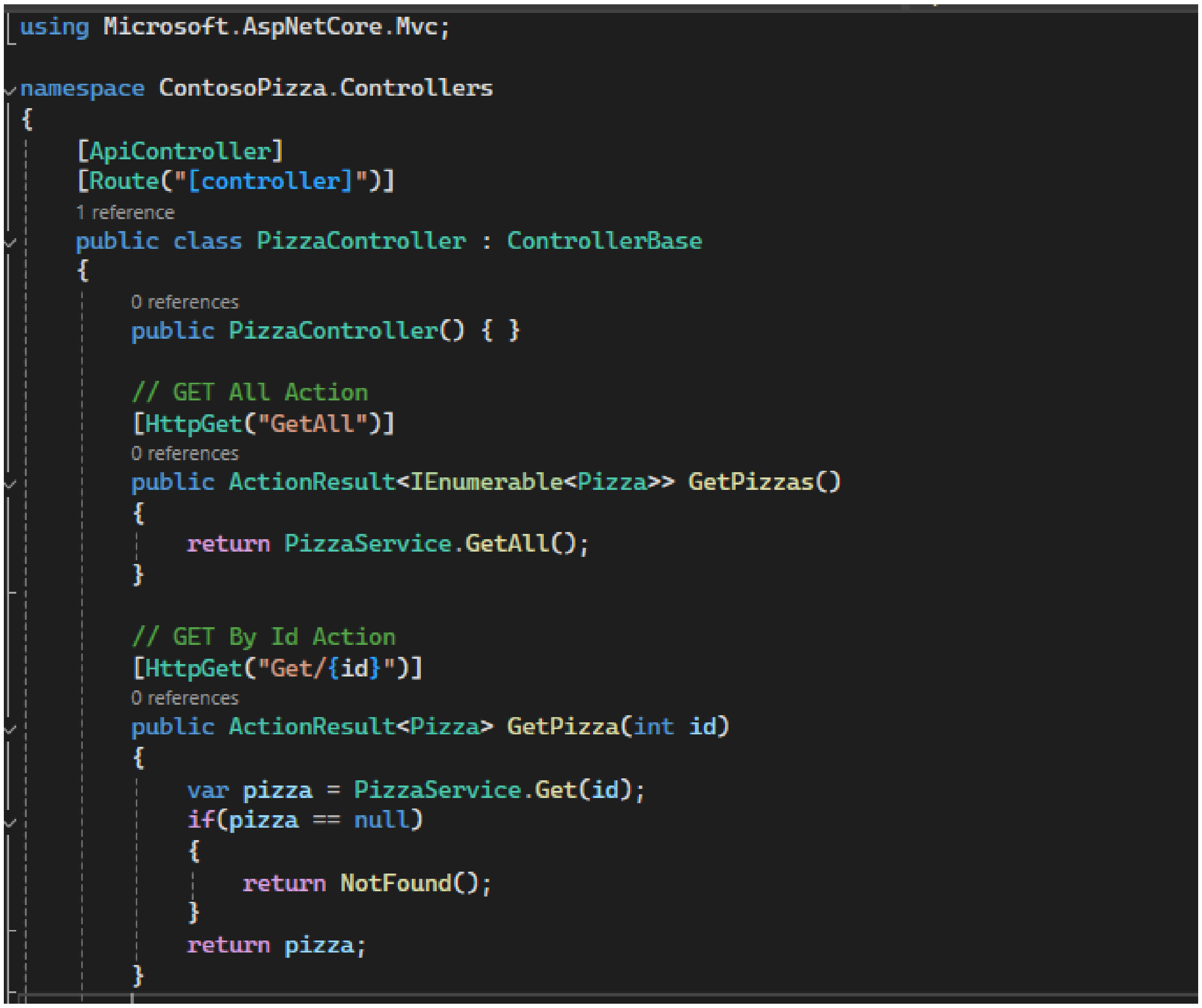
- Returns an `ActionResult` instance of type `List<Pizza>`. The `ActionResult` type is the base class for all action results in ASP.NET Core.



```
        // GET By Id Action
        public ActionResult<Pizza> GetPizza(int id)
        {
            var pizza = PizzaService.Get(id);
            if(pizza == null)
            {
                return NotFound();
            }
            return pizza;
        }
```

- Requires that the `id` parameter's value is included in the URL segment after `pizza/`. Remember, the controller-level `[Route]` attribute defined the `/pizza` pattern.

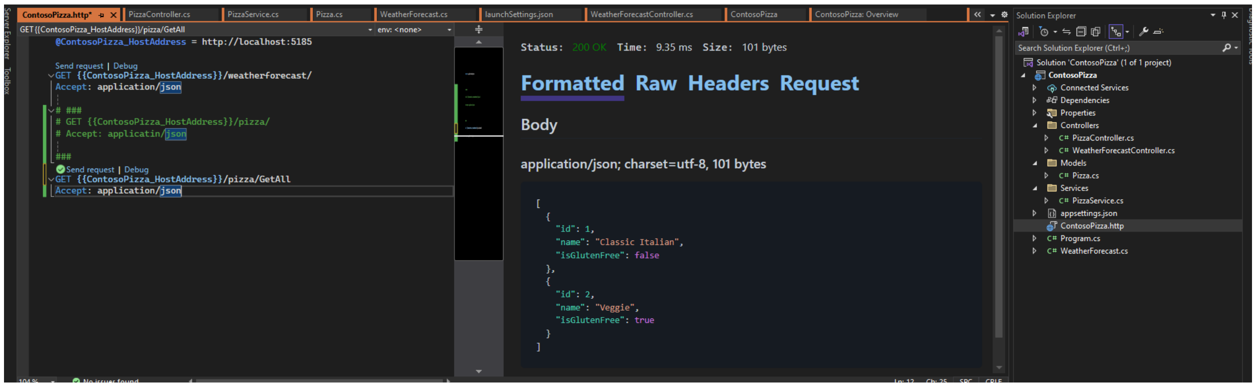
The code after writing these endpoints should look like this:



```
using Microsoft.AspNetCore.Mvc;

namespace ContosoPizza.Controllers
{
    [ApiController]
    [Route("[controller]")]
    1 reference
    public class PizzaController : ControllerBase
    {
        0 references
        public PizzaController() { }

        // GET All Action
        [HttpGet("GetAll")]
        0 references
        public ActionResult<IEnumerable<Pizza>> GetPizzas()
        {
            return PizzaService.GetAll();
        }

        // GET By Id Action
        [HttpGet("Get/{id}")]
        0 references
        public ActionResult<Pizza> GetPizza(int id)
        {
            var pizza = PizzaService.Get(id);
            if(pizza == null)
            {
                return NotFound();
            }
            return pizza;
        }
```

💡 Note: Add HTTP arrtibutes and Route. else will get error. swagger also give 500 error.

Now Run the application and check if the two HTTP endpoints are working.



```
@ContosoPizza_HostAddress = http://localhost:5185

Send request | Debug
GET {{ContosoPizza_HostAddress}}/weatherforecast/
Accept: application/json

# ###
# GET {{ContosoPizza_HostAddress}}/pizza/
# Accept: applicatin/json

###
Send request | Debug
GET {{ContosoPizza_HostAddress}}/pizza/GetAll
Accept: application/json
```

Status: 200 OK Time: 9.35 ms Size: 101 bytes

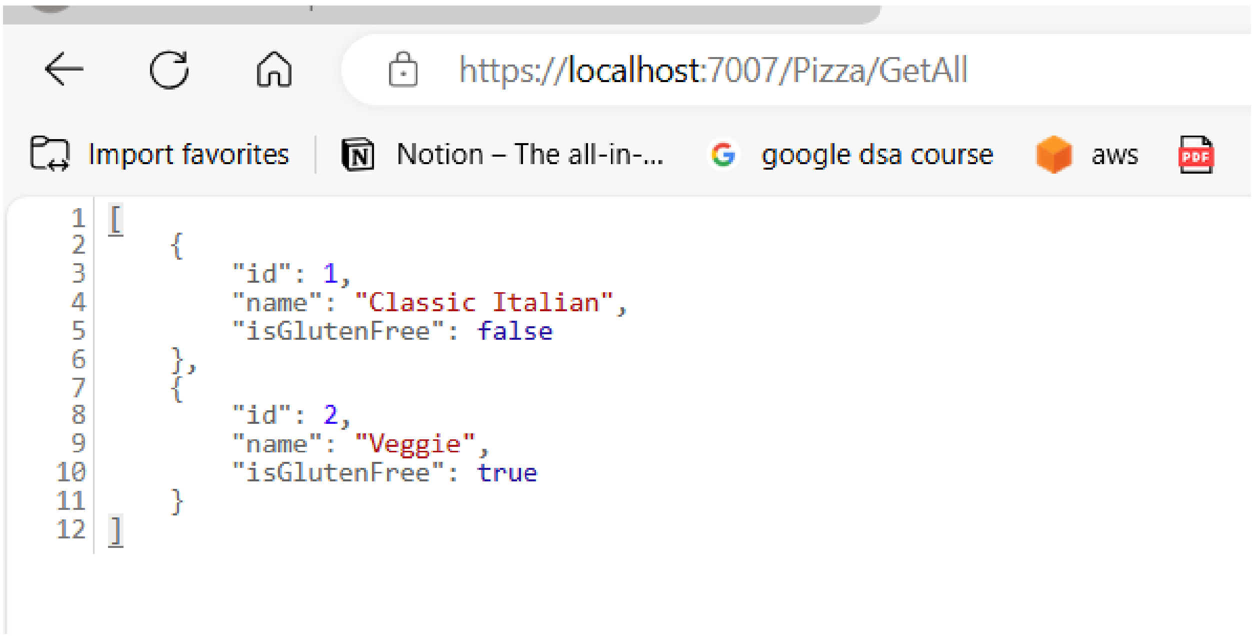
Formatted Raw Headers Request

Body

application/json; charset=utf-8, 101 bytes

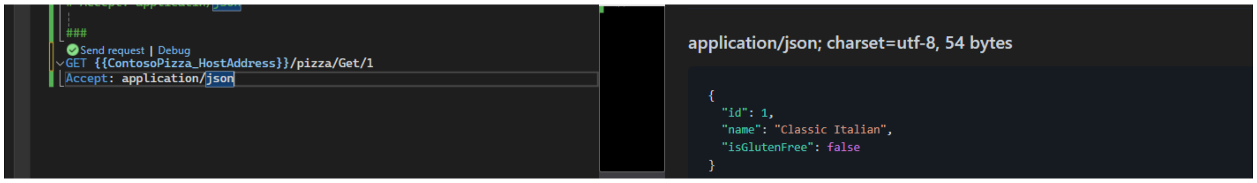
```
[
  {
    "id": 1,
    "name": "Classic Italian",
    "isGlutenFree": false
  },
  {
    "id": 2,
    "name": "Veggie",
    "isGlutenFree": true
  }
]
```

In browser, you will get an output like this.



https://localhost:7007/Pizza/GetAll

```
[
    {
        "id": 1,
        "name": "Classic Italian",
        "isGlutenFree": false
    },
    {
        "id": 2,
        "name": "Veggie",
        "isGlutenFree": true
    }
]
```



```
###
Send request | Debug
GET {{ContosoPizza_HostAddress}}/pizza/Get/1
Accept: application/json
```

application/json; charset=utf-8, 54 bytes

```
{
  "id": 1,
  "name": "Classic Italian",
  "isGlutenFree": false
}
```

If you do not get the data, Please make sure to enter the Action after Controller name in the url.

Since we didn’t set the default Controller and Action method, It will give response only when we give the correct URL

POST Endpoint:

```
[HttpPost]
public IActionResult Create(Pizza pizza)
{
    // This code will save the pizza and return a result
}
```

`IActionResult` lets the client know if the request succeeded and provides the ID of the newly created pizza. `IActionResult` uses standard HTTP status codes, so it can easily integrate with clients regardless of the language or platform they're running on.

Complete Code of PizzaController:

```
using ContosoPizza.Models;
using ContosoPizza.Services;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace ContosoPizza.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PizzaController : ControllerBase
    {
        public PizzaController() { }

        // GET All Action
        [HttpGet("GetAll")]
        public ActionResult<IEnumerable<Pizza>> GetPizzas()
        {
            return PizzaService.GetAll();
        }

        // GET By Id Action
        [HttpGet("Get/{id}")]
        public ActionResult<Pizza> Get(int id)
        {
            var pizza = PizzaService.Get(id);
            if(pizza == null)
            {
                return NotFound();
            }
            return pizza;
        }

        // POST Action
        [HttpPost("Create")]
        public IActionResult Create(Pizza pizza)
        {
            PizzaService.Add(pizza);
            return CreatedAtAction(nameof(Get), new { id = pizza
        }

        // PUT Action
        [HttpPut("{id}")]
        public IActionResult Update(int id, Pizza pizza)
        {
            if (id != pizza.Id) return BadRequest();

            var existingPizza = PizzaService.Get(id);

            if(existingPizza == null)
            {
                return NotFound();
            }
            PizzaService.Update(id, pizza);
            return NoContent(); // return 204. means the pizza i

        }

        // DELETE Action
        [HttpDelete]
        public IActionResult Delete(int id)
        {
            var pizza = PizzaService.Get(id);

            if (pizza == null) return NotFound();

            PizzaService.Delete(id);

            return NoContent(); // 204, means pizza is deleted.
        }
    }
}
```

Test using Swagger or postman.