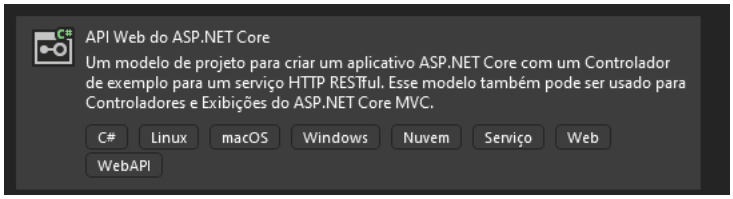
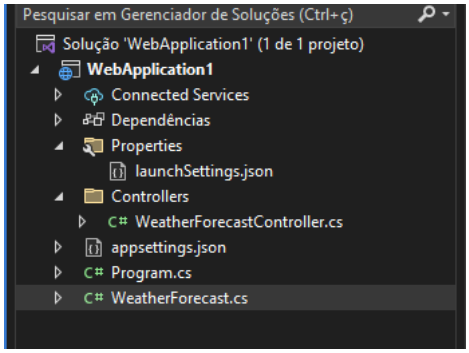
API ASP.Net Core

Criando projeto focado em uma api:

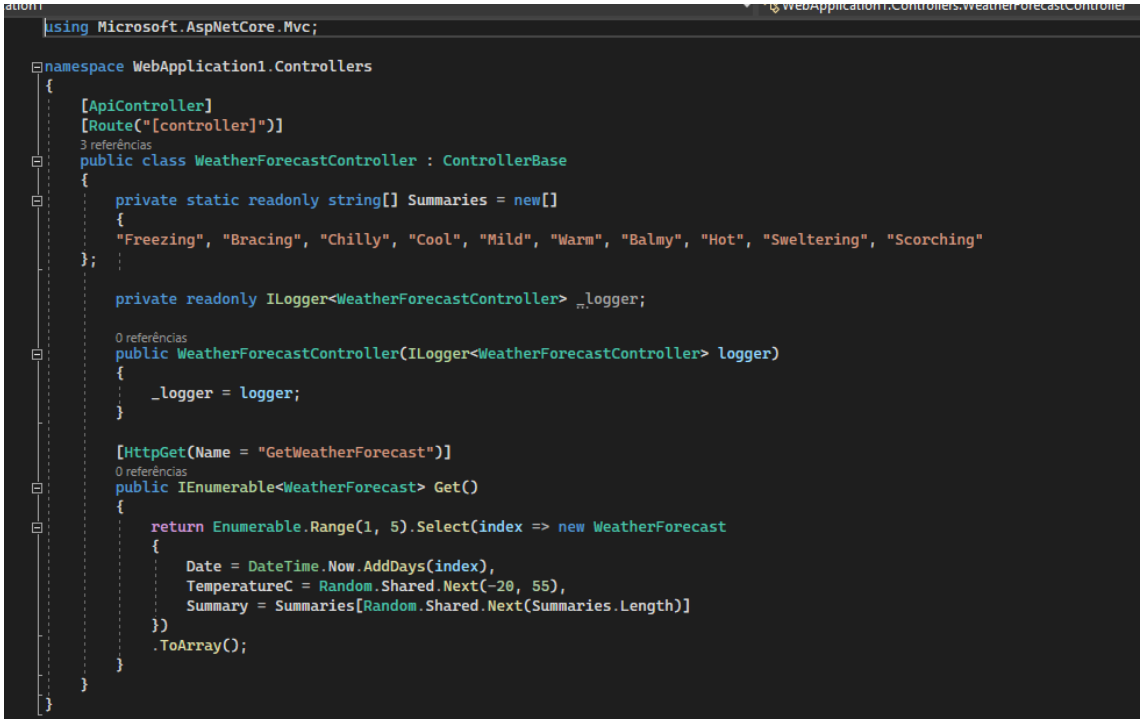
Para criarmos um projeto focado em uma API do ASP.NET Core, devemos criar uma solução e em seguida adicionar o seguinte projeto:



Esse projeto sera focado no uso de uma api, com isso temos a seguinte estrutura de arquivos:



Analisando a estrutura percebemos que se trata de um projeto ASP.NET Core extremamente simples onde temos uma classe WeatherForecast usada apenas como exemplo de uma classe model e temos o seu Controller onde é implementada a api referente a ela:



```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication1.Controllers
{
    [ApiController]
    [Route("[controller]")]
    3 referências
    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;

        0 referências
        public WeatherForecastController(ILogger<WeatherForecastController> logger)
        {
            _logger = logger;
        }

        [HttpGet(Name = "GetWeatherForecast")]
        0 referências
        public IEnumerable<WeatherForecast> Get()
        {
            return Enumerable.Range(1, 5).Select(index => new WeatherForecast
            {
                Date = DateTime.Now.AddDays(index),
                TemperatureC = Random.Shared.Next(-20, 55),
                Summary = Summaries[Random.Shared.Next(Summaries.Length)]
            })
            .ToArray();
        }
    }
}
```

Analisando a classe controller da api podemos perceber a estrutura que temos dela, em suma para termos um controller de uma api no ASP.NET Core so precisamos utilizar das anotações **ApiController, Route** e **HttpTipoApiRest** para os métodos.

Utilizando de Querys nos métodos:

Para utilizarmos as Querys nos métodos devemos apenas acrescentar a anotação **FromQuery** Antes da variavel de entrada:

```
[ApiController]
[Route("Students")]
public class StudentController : ControllerBase
{
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Student>>> GetStudentsAsync([FromQuery]
string firstName)
    {
        // Aqui você pode usar o valor do parâmetro firstName para filtrar os resultados
        // ...
    }
}
```

Entendendo as Rotas:

O que determina as rotas dos métodos são as Anotação [Route] na classe e nos métodos exemplo:

```
[ApiController]
[Route("api/[controller]")]
public class WeatherForecastController : ControllerBase
{
    [HttpGet]
    [Route("Login")]
    public async Task<ActionResult> Login()
    {
        // ...
    }
}
```

O método Login nesse exemplo tera a sua rota como “api/WeatherForecast/Login”

A anotação [controller] no Route indica que ali tera o nome do Controller que é determinada pela String escrita no nome da classe Que antecede a Palavra Controller.

Parametros de Rotas:

```
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
    [HttpGet("{id}/{name}")]
    public async Task<ActionResult<User>> GetUserAsync(int id, string name)
    {
        // Aqui você pode usar os valores dos parâmetros id e name para buscar o usuário
        // ...
    }
}
```

Parametros de Body:

Para termos paramêtros retirados do Body do método podemos utilizar a anotação **[FromBody]** antes do parametro:

```
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
    [HttpPost]
    public async Task<ActionResult<User>> CreateUserAsync([FromBody] User user)
    {
        // Aqui você pode usar o valor do parâmetro user para criar um novo usuário
        // ...
    }
}
```

Retornos ActionResult:

No ASP.NET nos temos os tipos IActionResult e ActionResult<t> que são como o ResponseEntity do java e serve para manipularmos os retornos dos métodos:

```
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
    [HttpGet("{id}")]
    public async Task<ActionResult<User>> GetUserAsync(int id)
    {
        // Aqui você pode buscar o usuário com o id especificado
        User user = ...;

        if (user == null)
        {
            // Se o usuário não for encontrado, retorne um código de status 404 (Not Found)
            return NotFound();
        }
        else
        {
            // Se o usuário for encontrado, retorne um código de status 200 (OK) com o
usuário no corpo da resposta
            return Ok(user);
        }
    }
}
```