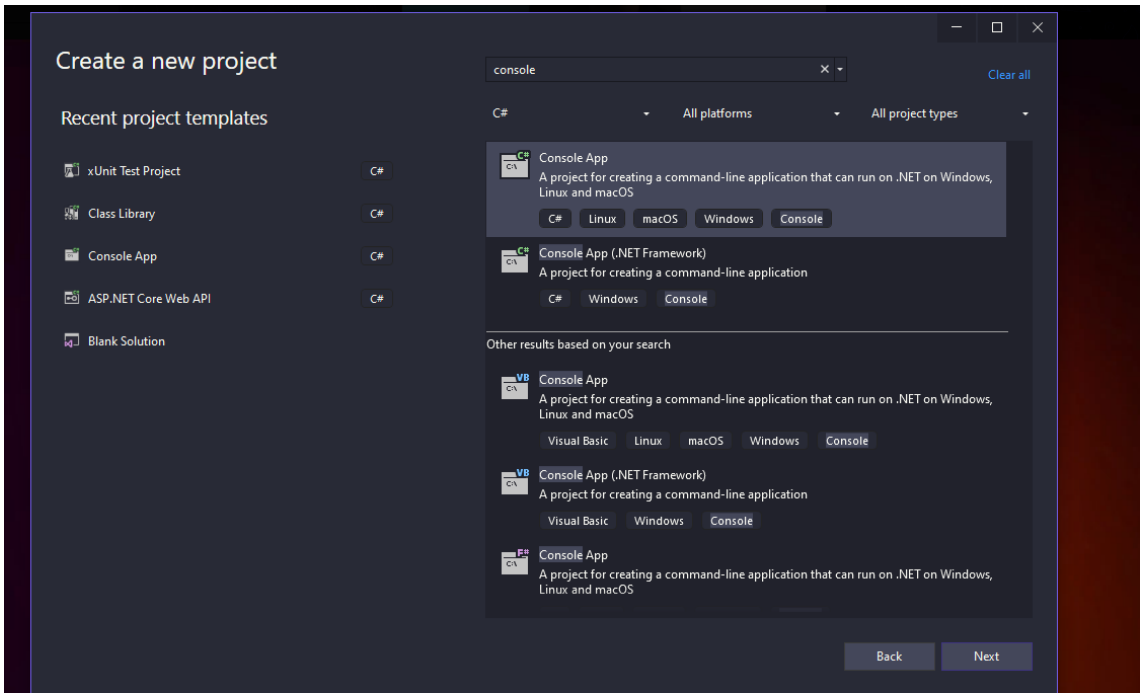
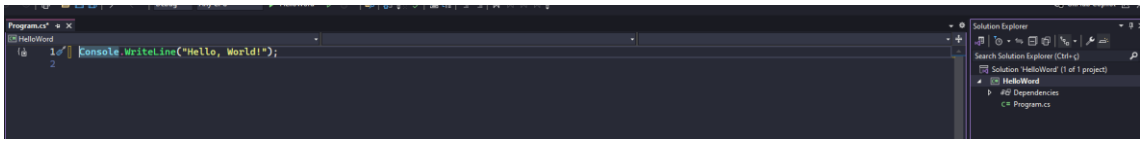
Módulo 01

#01 criando o primeiro projeto



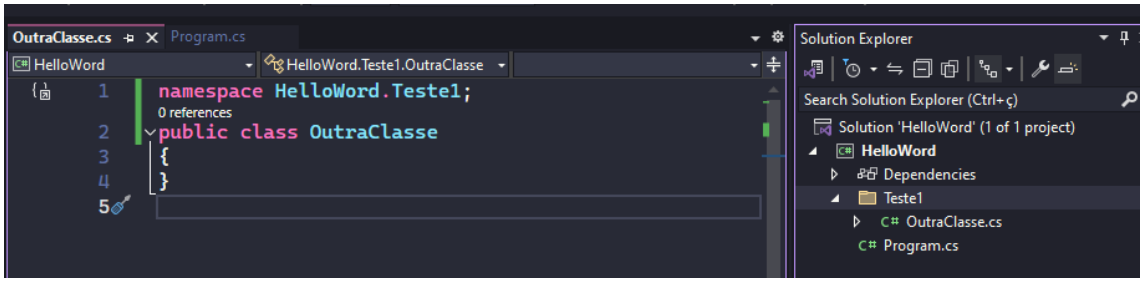
E da next e create

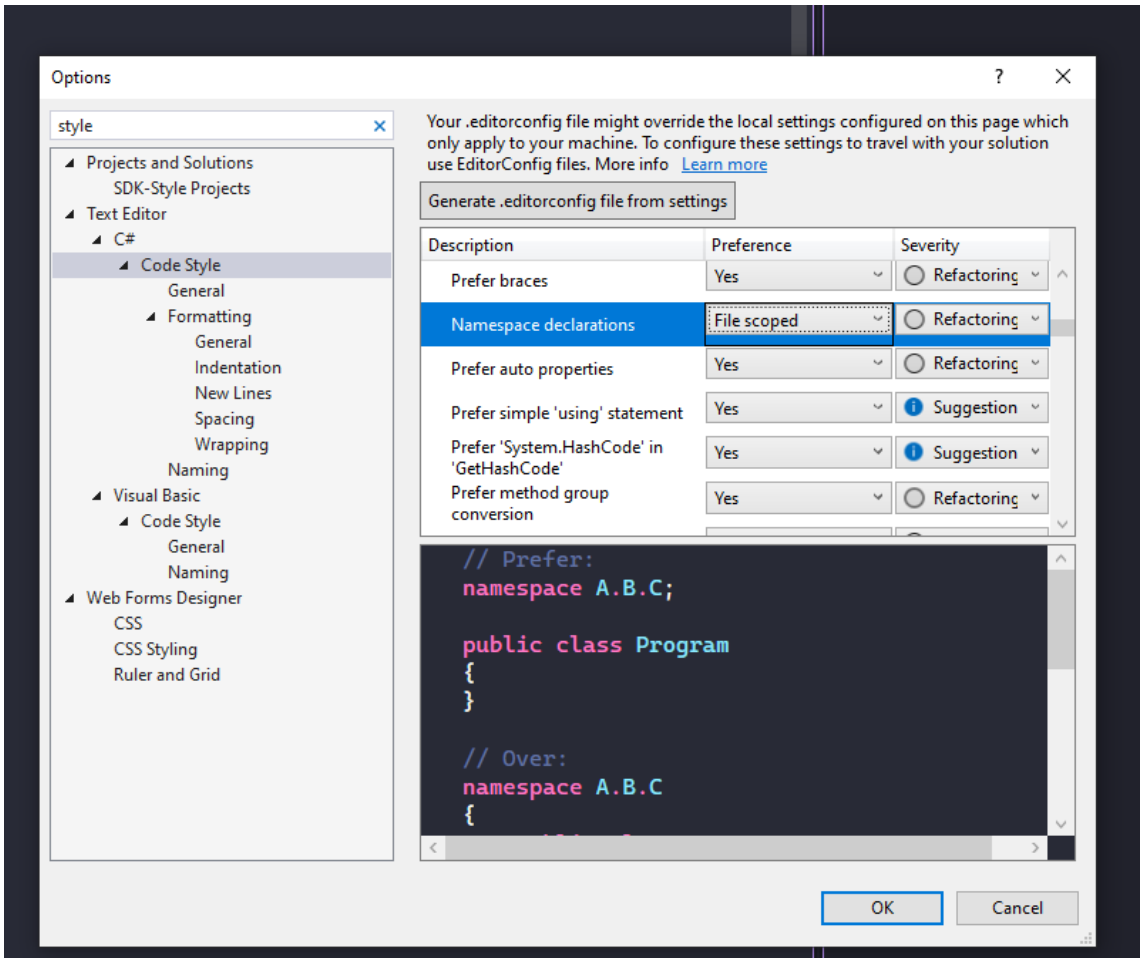


Projeto é onde implementa o código, solusions é um container onde agrupa projetos relacionados.

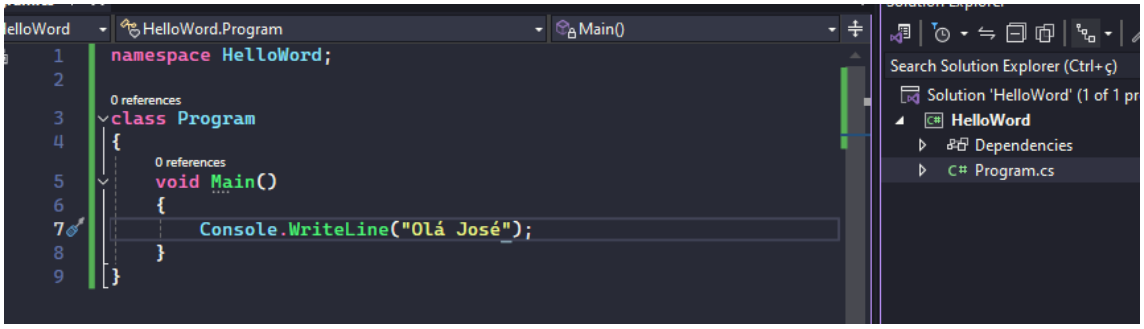
#02 O que é namespace

É a forma de organizar as classes dentro do projeto

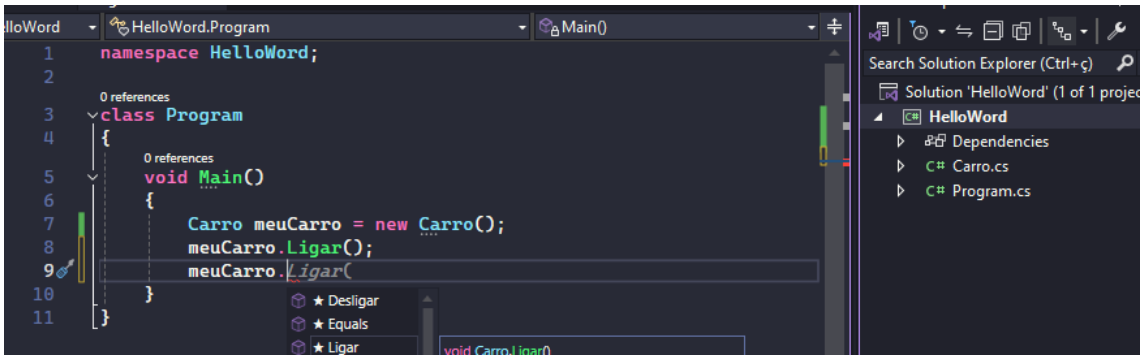




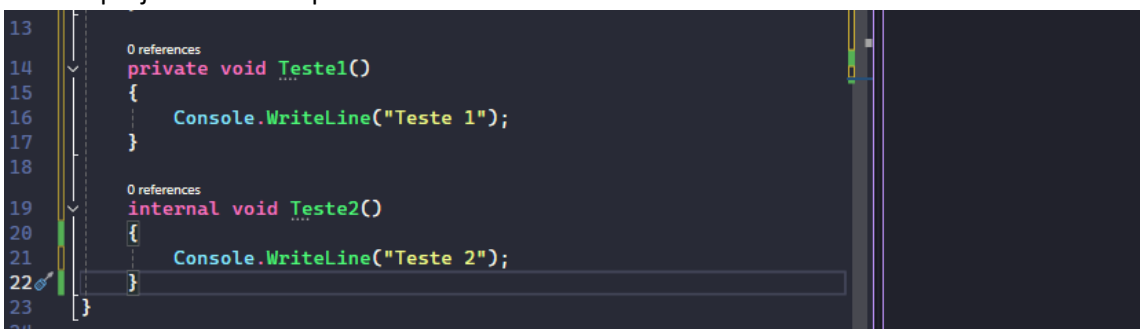
#03 O que são funções é algo que executa um bloco de código



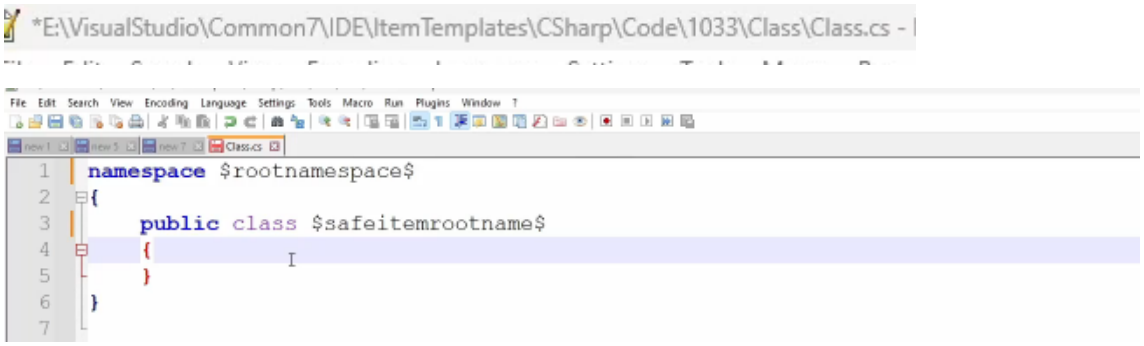
#04 Utilizando outras classes, cria um classe carro com as função desligar e ligar



#05 Modificadores de acesso, public qualquer classe pode acessa, private somente a classe pode usar, protected somente quem instância pode usar, internal somente que ta dentro do mesmo projeto ou namespace



#06 classes públicas sem usings entra onde vc instalou o visual em code 1033 e class e depois abra no notepad e salve



#07 tipos numéricos

#08 tipos booleano aceita apenas valores verdadeiro e falso

#09 tipos de texto

```
    static void Main()
    {
        char letra = 'a'; // apernas 1 caracteres
        string texto = "hello"; // textos ou frases
        char primeiraLetra = texto[0]; // pega o h do texto

        string removeSpaco = "    dawd ".Trim(); // remove os espaço inicio e fim
        bool comecaCom = texto.StartsWith("h"); // verifica se começa com
        bool terminaCom = texto.EndsWith("h"); // verifica se termina com

        string troca = texto.Replace('e', 'u'); // troca o e por u

        bool exite = texto.Contains("e"); // verifica se tem "e" dentro do texto
        bool exite2 = texto.Equals("e"); // verifica se é igual o texto
```

#10 operações com texto ,são concatenação com strings

#11 datas e horas

```
0 references
class Program
{
    //        d: 6/15/2008
    //        D: Sunday, June 15, 2008
    //        f: Sunday, June 15, 2008 9:15 PM
    //        F: Sunday, June 15, 2008 9:15:07 PM
    //        g: 6/15/2008 9:15 PM
    //        G: 6/15/2008 9:15:07 PM
    //        m: June 15
    //        o: 2008-06-15T21:15:07.0000000
    //        R: Sun, 15 Jun 2008 21:15:07 GMT
    //        s: 2008-06-15T21:15:07
    //        t: 9:15 PM
    //        T: 9:15:07 PM
    //        u: 2008-06-15 21:15:07Z
    //        U: Monday, June 16, 2008 4:15:07 AM
    //        y: June, 2008
    //
    //        'h:mm:ss.ff t': 9:15:07.00 P
    //        'd MMM yyyy': 15 Jun 2008
    //        'HH:mm:ss.f': 21:15:07.0
    //        'dd MMM HH:mm:ss': 15 Jun 21:15:07
    //        '\Mon\t\h\: M': Month: 6
    //        'HH:mm:ss.ffffzzz': 21:15:07.0000-07:00

    0 references
    static void Main()
    {   // apenas data
        DateOnly dia = new DateOnly(2024, 12, 1); // 01-janeiro-01

        string diaEmTexto = dia.ToString("dd/MM/yyyy", new CultureInfo("pt-BR")); //
          01/12/2024

        DateTime dia1 = new DateTime(2024, 12, 1, 20, 00, 01); // 2024/12/01 20:00:01

        DateTime diaAgora = DateTime.Now; // pega data agora data e hora
        DateTime diaAgora2 = DateTime.Today; // pega data agora data
        DateTime diaAgora3 = DateTime.UtcNow; // pega data Mundial
        DateTime nvData = diaAgora.AddDays(1); // adiciona um dia
    }
}
```

#12 enums

```
class Program
{
    2 references
    enum NivelDeDificuldade
    {
        baixo = 0,
        medio = 5,
        alto = 10,
    }

    0 references
    static void Main()
    {
        NivelDeDificuldade nivel = NivelDeDificuldade.baixo; // baixo

        int nivelInteiro = (int)nivel; // 0

    }
}
```

#13 var pode receber qualquer tipo para a variável, o compilador escolhe o tipo para você

```
    0 references
    static void Main()
    {
        var teste = "bom dia"; // bom dia

    }
}
```

#14 objects é do tipo obeto, usa para não definir um objeto

```
static void Main()
{
    object meutexto = "jose romario"; // jose romario
```

#15 null é um valor nullo

```
0 references
static void Main()
{
    int? idade = null; // aceita valores do tipo null e int
```

#16 arrays armazena coleção de valores

```
0 references
static void Main()
{
    int[] inteiro = new int[3]; // tem 3 tamanho o array [0,0,0]
    inteiro[1] = 20; // [0,20,0]

    int[] inteiro2 = [1,2,3]; // [1,2,3]

    int[,] duasDimencao = new int[2, 2]; // [0,0] [0,0]
    duasDimencao[0, 1] = 2; // [0,2] [0,0]
    duasDimencao[1, 0] = 4; // [0,2] [4,0]
}
```

#17 listas

```
{
    List<int> inteiro = new List<int>(); // usa o Count para contar quantos tem
    inteiro.Add(1); // 1 tamanho pode acessar pelo indicie tbm inteiro[0] = 1
    // inteiro.Remove(1) remove o elemento e indice passa o elemento
    // inteiro.First() pega o primeiro elemento
    // inteiro.RemoveAt(0) remove pelo indice
    // inteiro.Clear() remove todos elementos
}
```

#18 um pouco mais sobre listas

```
static void Main()
{
    List<object> obj = new List<object>();

    obj.Add("carro");
    obj.Add(1);
    obj.Add(false);

    List<string> list = new List<string>();

    list.Add("Hello");
    list.Add("World");

    string result = string.Join(", ", list); // O Join junta = hello, world
}
}
```

#19 dicionário

```
0 references
static void Main()
{
    //chave valor
    Dictionary<int, string> dicionario = new Dictionary<int, string>();

    dicionario.Add(1, "maria");
    dicionario.Add(2, "pedro");
    dicionario.Add(3, "julia");

    string value = dicionario[3]; // julia
    bool value2 = dicionario.ContainsKey(2); // true
}
}
```

#20 hashset

```
0 references
static void Main()
{
    //os valores tem q ser único
    HashSet<int> hashset = new HashSet<int>();

    hashset.Add(1);
    hashset.Add(2);
    hashset.Add(3);
    hashset.Add(3);
}
}
```

#21 funções com parâmetros

```
namespace HelloWord;
1 reference
public class Operacoes
{
    1 reference
    public void Adicionar(int v1, int v2)
    {
        var resultado = v1 + v2;
    }
}
```

```
Search Solution Explorer (Ctrl+ç)
Solution 'HelloWord' (1 of 1 project)
HelloWord
Dependencies
Operacoes.cs
Program.cs
```

```
0 references
class Program
{

    0 references
    static void Main()
    {
        var matematica = new Operacoes();

        matematica.Adicionar(v1: 5, v2: 5); // 10
    }
}
```

#22 funções devolvendo valores

```
1 reference
public class Operacoes
{
    1 reference
    public int Adicionar(int v1, int v2)
    {
        var resultado = v1 + v2;

        return resultado;
    }
}
```

```
    0 references
    static void Main()
    {
        var matematica = new Operacoes();

        var resultado = matematica.Adicionar(v1: 5, v2: 5); // 10

        Console.WriteLine(resultado);
    }
}
```

Ou pode fazer assim

```
{
    1 reference
    public int Adicionar(int v1, int v2) => v1 + v2;
}
```

Retornar 2 valores

```
1 reference
public class Operacoes
{
    1 reference
    public (int, string) Adicionar(int v1, int v2)
    {
        var resultado = v1 + v2;

        return (resultado, "teste");
    }
}
```

```
0 references
static void Main()
{
    var matematica = new Operacoes();

    var resultado = matematica.Adicionar(v1: 5, v2: 5); // 10

    Console.WriteLine(resultado.Item1); // int valor 1
    Console.WriteLine(resultado.Item2); // string valor 2
}
```

Pode criar 2 variável também

```
0 references
static void Main()
{
    var matematica = new OperacoesMatematicas();

    (int resultado, string nome) = matematica.Adicionar(7, 3);

    Console.WriteLine(resultado);
    Console.WriteLine(nome);
}
```

#23 parâmetros opcionais somente os últimos valores pode ser opcional

```
        return (resultado, "teste");
    }

    0 references
    public void Teste(int valor1, int valo2 = 7) // transforma o valor 2 em opcional
    {
        Console.WriteLine(valor1 + valo2);
    }
}
```

#24 criando classes com valores

Cor.cs | Carro.cs | Program.cs

HelloWord | HelloWord.Cor

```
namespace HelloWord;
1 reference
public enum Cor
{
    Vermelho,
    Azul,
    Amarelo
}
```

```
namespace HelloWord;
2 references
public class Carro
{
    4 references
    public required string Modelo { get; set; } // o required obriga a preencher o valores
    1 reference
    public DateOnly LancadoEm { get; set; }
    1 reference
    public Cor Cor { get; set; }

    1 reference
    public Carro(string modelo) // um construtor da classe
    {
        Modelo = modelo;
    }

    1 reference
    public void NomeDoModelo() => Console.WriteLine(Modelo);
}
```

```
0 references
class Program
{

    0 references
    static void Main()
    {
        var carro = new Carro("Teste") { Modelo = "dawd" };

        carro.Modelo = "Palio"; // adiconar valores as propriedades
        carro.Cor = Cor.Vermelho;
        carro.LancadoEm = new DateOnly(2020, 01, 01);
        carro.NomeDoModelo();
    }
}
```

#25 o que é static usa o static assim não precisa instanciar a classe para poder usar a função

```
    {
        Console.WriteLine(valor1 + valo2);
    }

    0 references
    public static int Adicionar2(int valor1, int valor2) => valor1 + valor2;
}
```

E usa ela assim com o nome da classe e a função

```
    0 references
    static void Main()
    {
        var resultado = Operacoes.Adicionar2(1, 20);
    }
}
```

```
1 reference
public class Carro
{
    2 references
    public static string Modelo { get; set; } // é compartilhado com todos por ser static
    0 references
    public DateOnly LancadoEm { get; set; }
```

Pode fazer uma classe static também, porém uma classe static não pode ser instaciada

```
1 reference
public static class Operacoes
{
    0 references
```

#26 oque é e como funciona o debug clica no canto da linha e ficará um bolinha ai roda o programa ai ele para ali e aperta f10 para ir próxima linha e entrar numa função é f11

```
3   class Program
4   {
        0 references
5       static void Main()
6       {
7           int numero = 10;
8
9           var meuCarro = new Carro("Porsche")
```

#27 if else e else if parte 1

```
int numero = 10;
double saldo = 100.50;
bool ativo = true;
string autor = "Welisson";
List<int> lista = new List<int> { 1, 7 };
Cores cor = Cores.Azul;

/*
    > Maior
    < Menor
    >= Maior ou igual
    <= Menor ou igual
    != diferente
 */

if (numero > 0)
{
    Console.WriteLine("Este numero é positivo");
}
}
```

```
if (numero > 0)
{
    Console.WriteLine("Este numero é POSITIVO");
}
else
{
    Console.WriteLine("Este numero é NEGATIVO");
}
}
```

```
 */

if (numero > 0)
{
    Console.WriteLine("Este numero é POSITIVO");
}
else if(numero == 0)
{
    Console.WriteLine("Este numero é NEUTRO");
}
else
{
    Console.WriteLine("Este numero é NEGATIVO");
}
}
```

Para string usa o Equals

```
if (autor.Equals("Edilaine"))
{
    Console.WriteLine("ENTROUUU");
}
}
```

#28 if else e else if parte 2

```
if (cor == Cores.Azul || numero >= 0)
{
    Console.WriteLine("ENTROUUU");
}
```

```
string mensagemDeErro = null;

if (mensagemDeErro is not null)
{
    Console.WriteLine("ENTROUUU");
}
```

```
if ((numero == 0 && saldo > 100.0) || ativo)
{
    Console.WriteLine("ENTROUUU");
}
```

#29 condicional ternário

```
static void Main()
{
    int numero = 7;

    string autor = numero == 7 ? "welisson" : "willian";
```

#30 switch

#31

#32

#33

#34

#35

#36

#37

#38

#39

#40

#41

#42

#43

#44

#45

#46

#47

#48

#49

#50

#51

#52

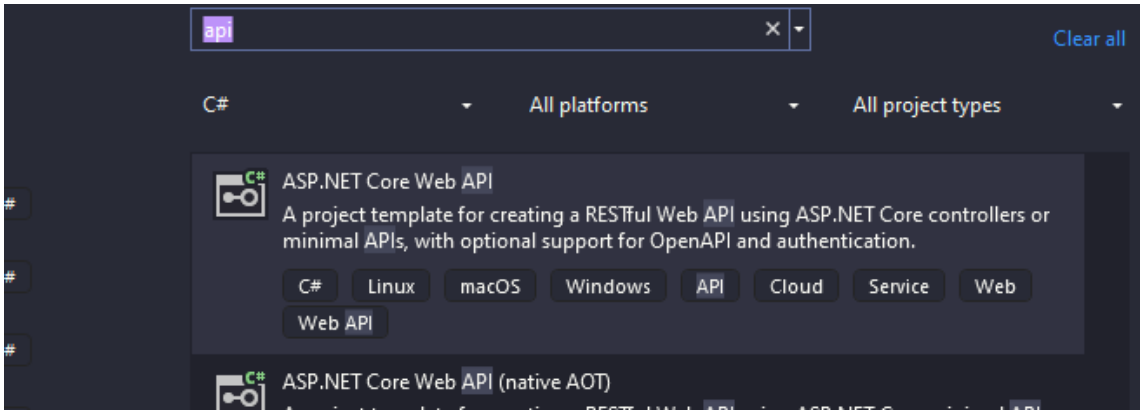
#53

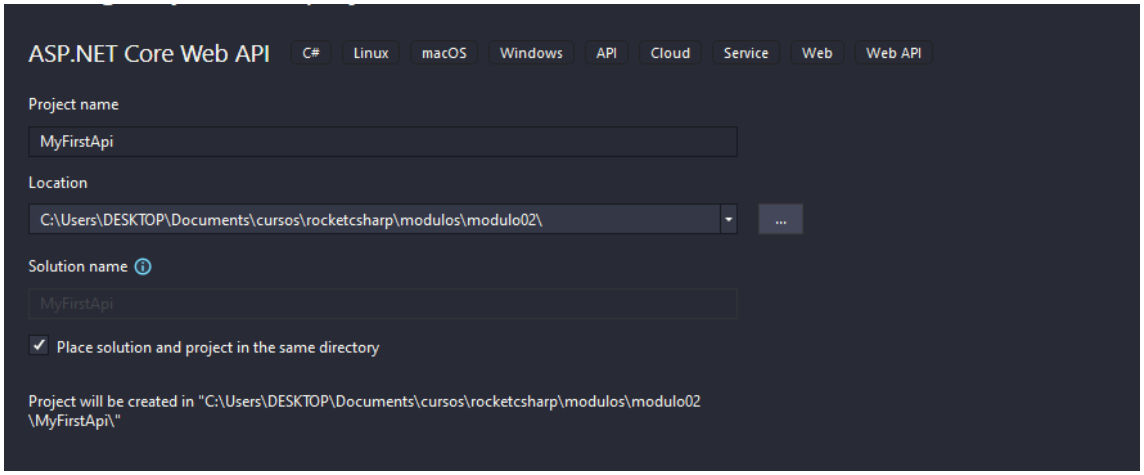
#54

#55

#56

#25 Criando uma api





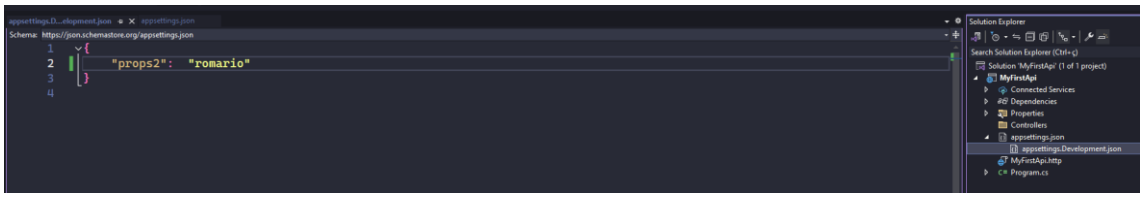
F5 roda o projeto

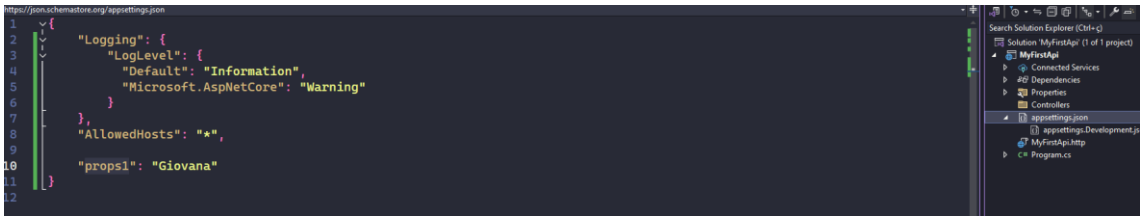
#26 Properties

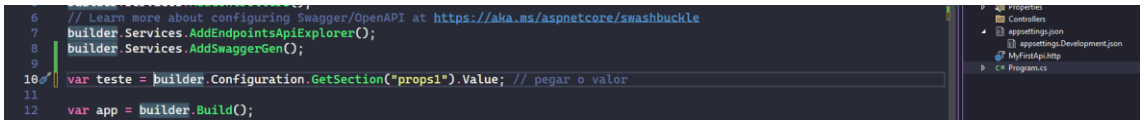
#27 O arquivo program.cs

#28 função appsetting.json

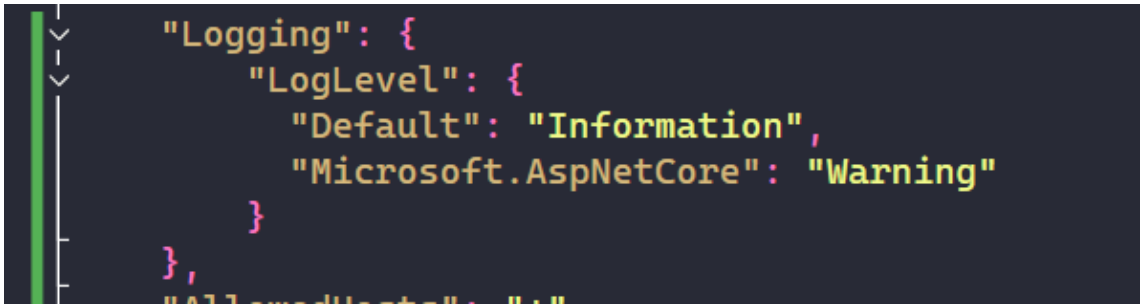
#29 Lendo appsettings.json

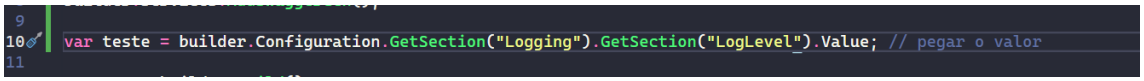




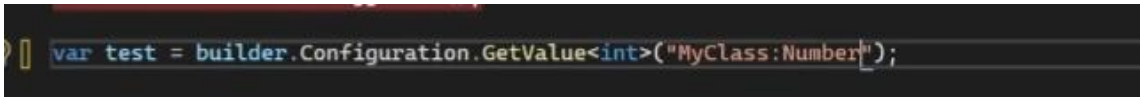


pegar valor dentro de um objeto, sempre da prioridade para o development caso tenha obj com o nomes igual



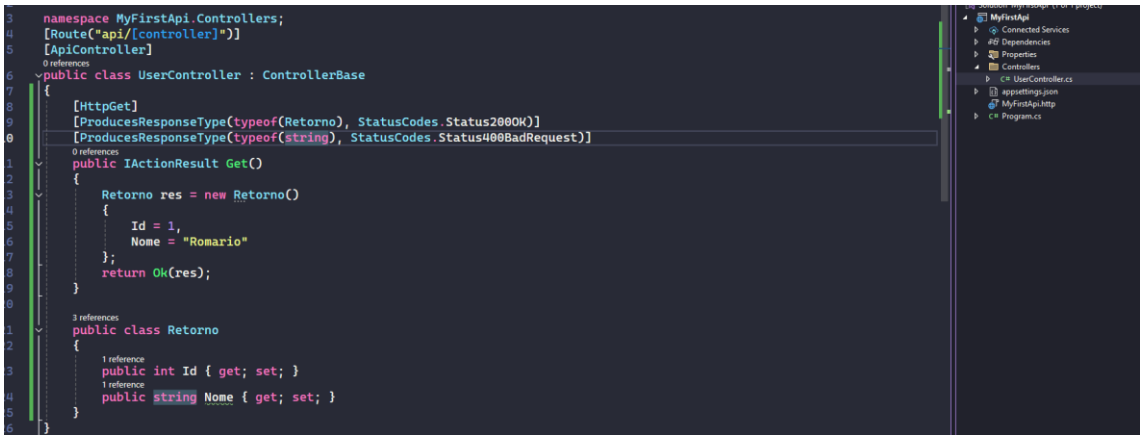


Pegar o valor direto



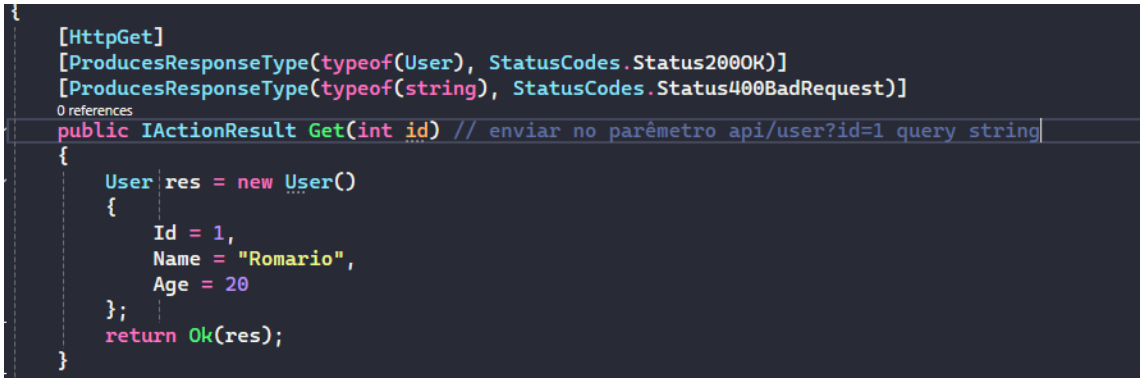
#30 oque é um controler

#31 criando o primeiro endpoint



```
namespace MyFirstApi.Controllers;
[Route("api/[controller]")]
[ApiController]
0 references
public class UserController : ControllerBase
{
    [HttpGet]
    [ProducesResponseType(typeof(Retorno), StatusCodes.Status200OK)]
    [ProducesResponseType(typeof(string), StatusCodes.Status400BadRequest)]
    0 references
    public IActionResult Get()
    {
        Retorno res = new Retorno()
        {
            Id = 1,
            Nome = "Romario"
        };
        return Ok(res);
    }

    3 references
    public class Retorno
    {
        1 reference
        public int Id { get; set; }
        1 reference
        public string Nome { get; set; }
    }
}
```

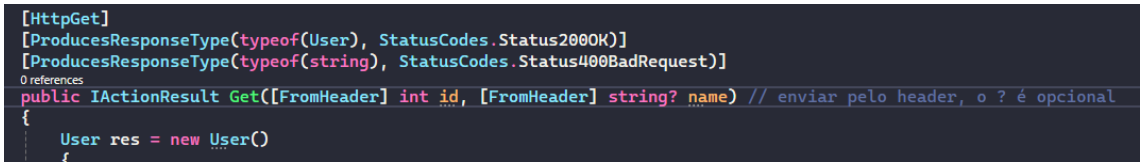
#32 como passar valores pela rota, GET só aceita passar nas rotas e no header



```
{
    [HttpGet]
    [ProducesResponseType(typeof(User), StatusCodes.Status200OK)]
    [ProducesResponseType(typeof(string), StatusCodes.Status400BadRequest)]
    0 references
    public IActionResult Get(int id) // enviar no parêmetro api/user?id=1 query string
    {
        User res = new User()
        {
            Id = 1,
            Name = "Romario",
            Age = 20
        };
        return Ok(res);
    }
```

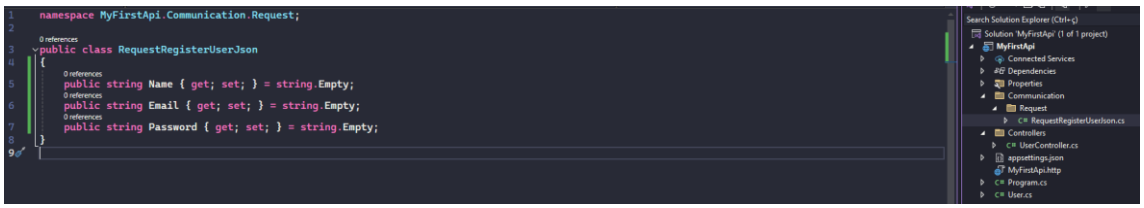
```
0 references
public class UserController : ControllerBase
{
    [HttpGet]
    [Route("{id}/teste/{name}")] // aqui fica assim api/user/10/teste/maria
    [ProducesResponseType(typeof(User), StatusCodes.Status200OK)]
    [ProducesResponseType(typeof(string), StatusCodes.Status400BadRequest)]
    0 references
    public IActionResult Get(int id, string name) // enviar no parêmetro api/user?id=1 query string
    {
        User res = new User()
        {
            Id = 1,
```

#33 passando valores pelo headers



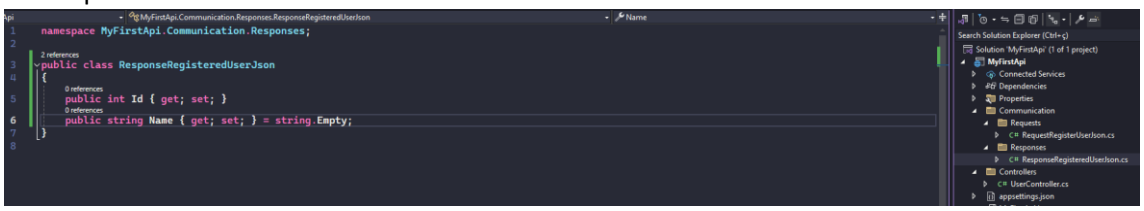
```
[HttpGet]
[ProducesResponseType(typeof(User), StatusCodes.Status200OK)]
[ProducesResponseType(typeof(string), StatusCodes.Status400BadRequest)]
0 references
public IActionResult Get([FromHeader] int id, [FromHeader] string? name) // enviar pelo header, o ? é opcional
{
    User res = new User()
    {
```

#34 criando um endpoint post quando for enviar um dados para criar cria um comunication e um request



```
namespace MyFirstApi.Communication.Request;

0 references
public class RequestRegisterUserJson
{
    0 references
    public string Name { get; set; } = string.Empty;
    0 references
    public string Email { get; set; } = string.Empty;
    0 references
    public string Password { get; set; } = string.Empty;
}
```

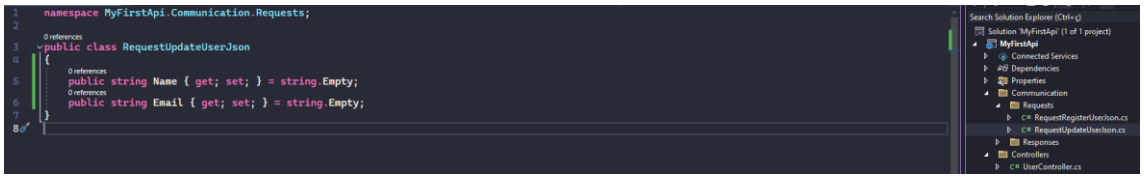
E a resposta



```
namespace MyFirstApi.Communication.Responses;

public class ResponseRegisteredUserJson
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
}
```

```
    }

    [HttpPost]
    [ProducesResponseType(typeof(ResponseRegisteredUserJson), StatusCodes.Status201Created)]
    0 references
    public IActionResult Create([FromBody] RequestRegisterUserJson request)
    {
        var response = new ResponseRegisteredUserJson()
        {
            Id = 1,
            Name = "teste"
        };

        return Created(string.Empty, response);
    }
}
```

#35 criando um endpoint put



```
namespace MyFirstApi.Communication.Requests;

public class RequestUpdateUserJson
{
    public string Name { get; set; } = string.Empty;
    public string Email { get; set; } = string.Empty;
}
```

```
    [HttpPut]
    [Route("{id}")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    0 references
    public IActionResult Update([FromRoute] int id, [FromBody] RequestUpdateUserJson request)
    {
        return NoContent();
    }
}
```

#36 criando um endpoint delete

```
[HttpDelete]
[ProducesResponseType(StatusCodes.Status204NoContent)]
0 references
public IActionResult Delete()
{
    return NoContent();
}
```

#37 endpoint com nomes iguais evitar colisões de endpoint dando nome as rotas

```
    [HttpPut("change-password")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    0 references
    public IActionResult ChangePassword([FromBody] RequestChangePasswordJson request)
    {
        return NoContent();
    }
```

#38

#39

#40

#41

#42

#43

#44

#45

#46

#47

#48

#49

#50

#51

#52

#53

#54

#55

#56