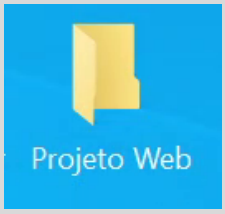
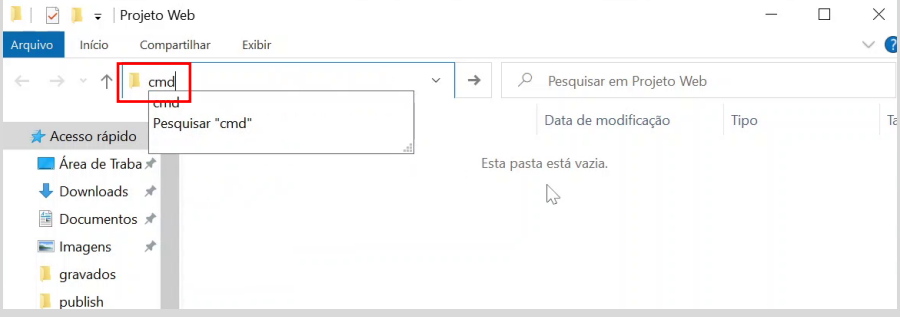
Projeto web

Criando o Projeto web

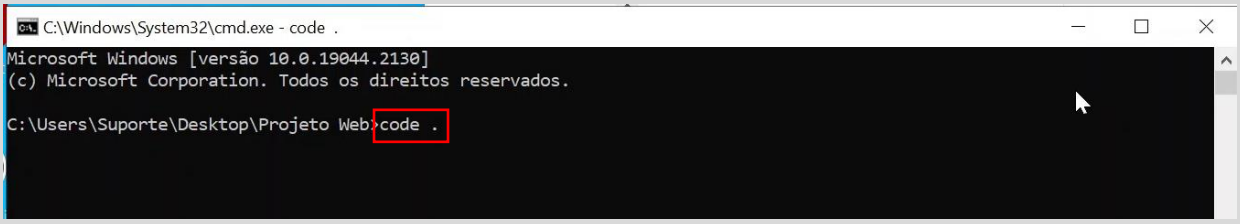
1. Crie uma pasta chamada **Projeto Web**.



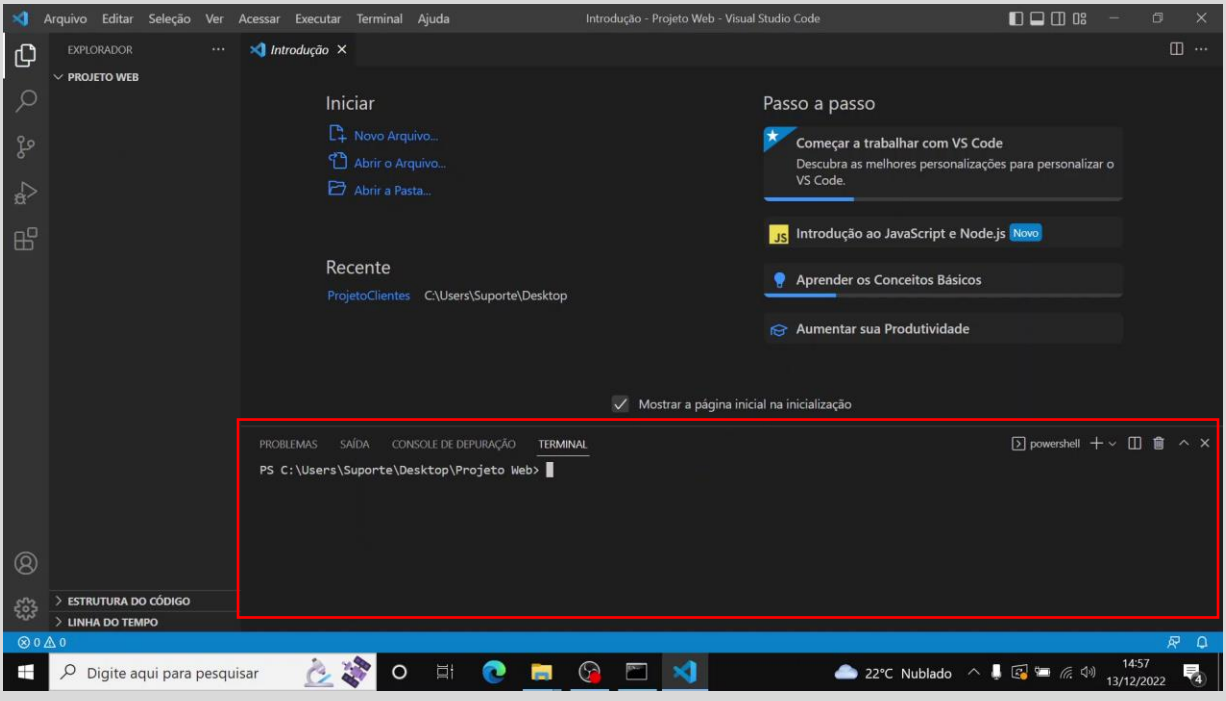
2. Abra essa pasta, selecione o texto na barra de endereços, digite **cmd** e pressione **Enter**, conforme indicado na imagem a seguir.



3. Na janela que abrir, digite o comando **code .**, conforme mostrado na imagem.
   Atenção: tudo em minúsculo e com **espaço** antes do ponto.



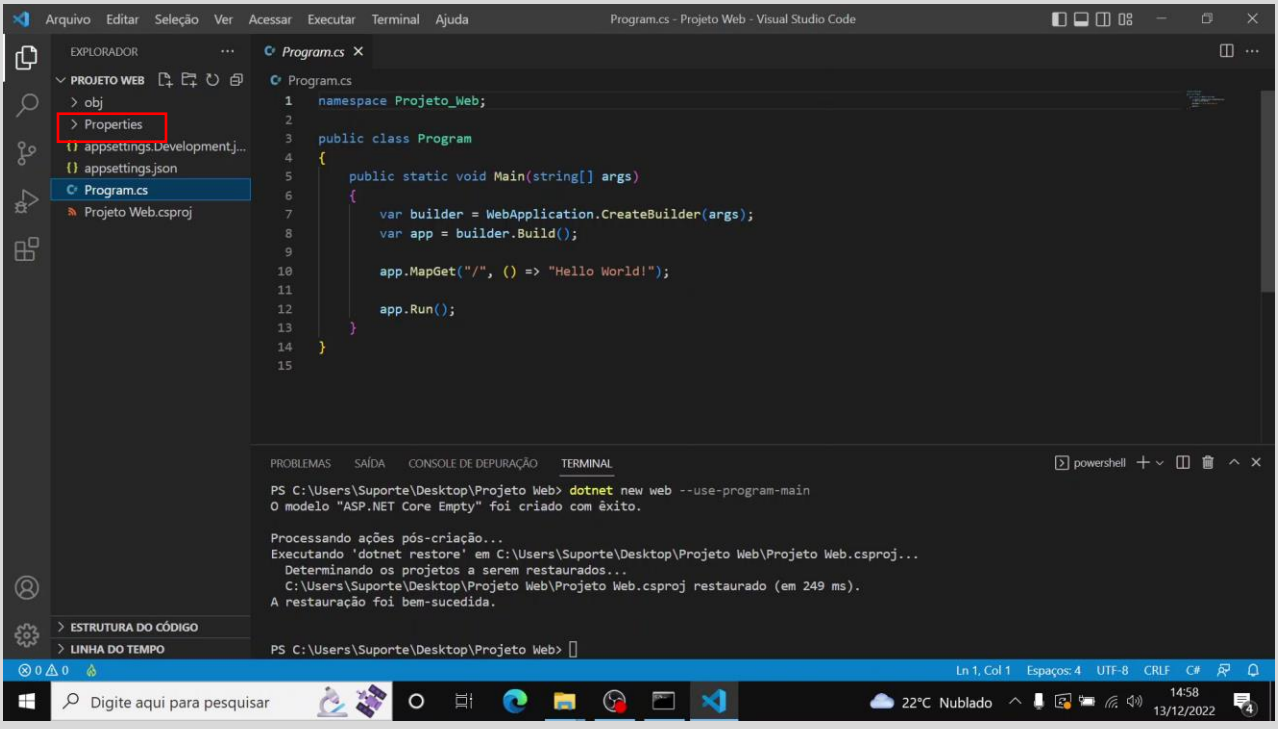
4. O VSCode iniciará com o arquivo **PROJETO WEB** aberto. Utilize o atalho **Ctrl + apóstrofo** para abrir o terminal na parte inferior do programa, como na imagem a seguir.



5. Digite no terminal o comando a seguir para criar o projeto web com as pastas-padrão. Isso permitirá que o projeto seja compilado e publicado na internet.

```
dotnet new web --use-program-main
```

6. Localize e abra o arquivo **Program.cs**, como indicado na imagem.



7. No terminal, digite o comando a seguir.

```
dotnet run
```

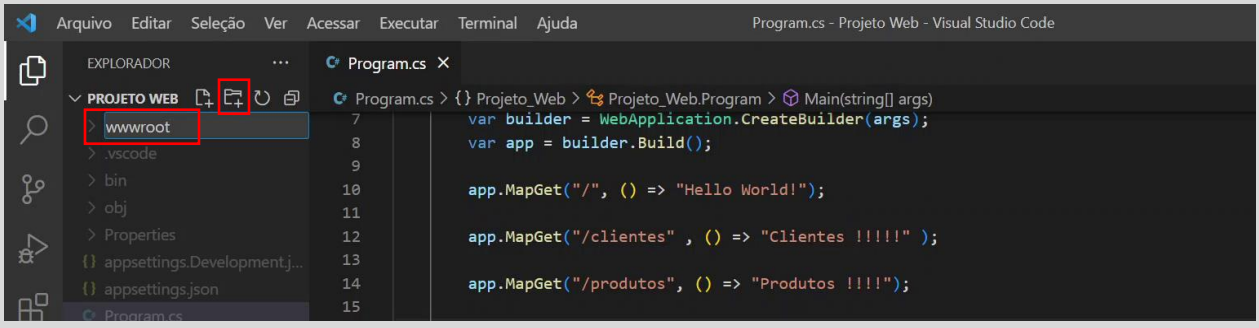
Publicando o projeto

No terminal, digite o comando:

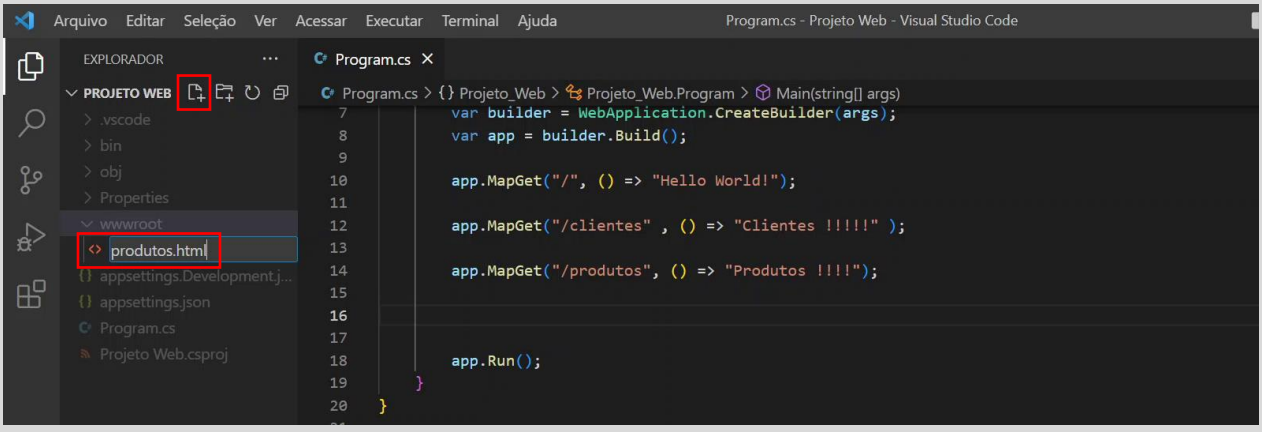
```
dotnet publish -c Release
```

Usando páginas estáticas

1. Crie uma nova pasta e nomeie-a de **wwwroot**, conforme a imagem.



2. Dentro da pasta recém-criada, crie o arquivo **produtos.html**.



3. Em **produtos.html**, digite o seguinte código:

```
<html>
<body>

    <H1> Página estática Produtos !!!!</H1>


</body>
</html>
```

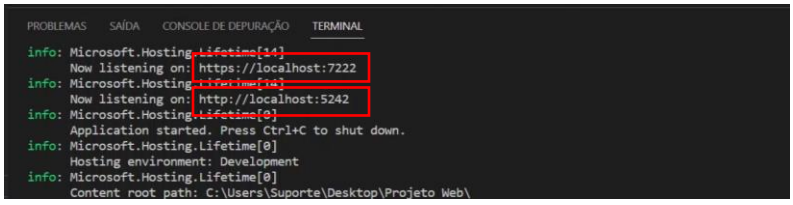
4. Abra o arquivo **Program.cs** e digite as linhas de código destacadas na imagem. Elas serão explicadas logo mais.

```
namespace Projeto_Web;

public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);
        var app = builder.Build();

        app.UseStaticFiles();

        app.MapGet("/", () => "Hello World!");

        app.MapGet("/cliente" , () => "Cliente !!!!!" );

        app.MapGet("/produtos" , () => "Produtos !!!!!" );

        app.Run();
```

5. Faça um teste no terminal usando o comando **dotnet run** e acesse o link da página local pelo navegador. Digite o caminho **/produtos.html**.

Você sabia?

Ao usar o comando **dotnet run** no terminal para rodar o programa, **são exibidos os links do projeto**. Assim, ao clicar nesses links com o Ctrl pressionado, o projeto é aberto no navegador.



6. Para resolver o problema de acentuação apresentado no navegador, volte ao arquivo **produtos.html** e digite o seguinte trecho de código.

```
<html>
        <head>
            <meta charset="utf-8">
        </head>
<body>

    <H1>Página estática Produtos !!!!</H1>


</body>
</html>
```

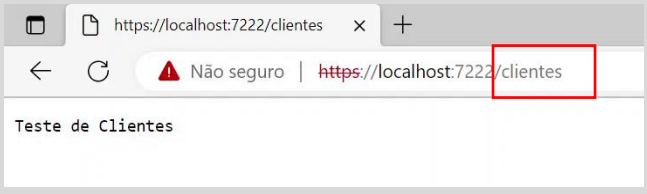
7. Faça o teste novamente com o comando **dotnet run** e acesse o link da página local pelo navegador (**/produtos.html**).

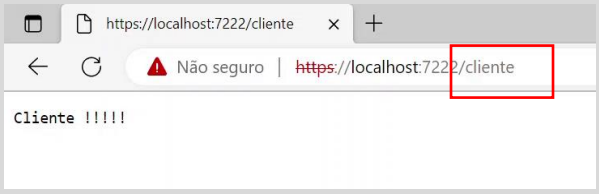
Mapping

1. No arquivo **Program.cs**, localize o trecho mostrado na imagem e insira a linha de código destacada.

```
app.MapGet("/", () => "Hello World!");

app.MapGet("/cliente" , () => "Cliente !!!!!" );

app.MapGet("/produtos" , () => "Produtos !!!!!" );

app.MapGet("/clientes" , () => "Teste de Clientes" );

app.Run();
```

2. Faça um novo teste com o comando **dotnet run** e acesse as páginas dos caminhos **/clientes** e **/cliente**.





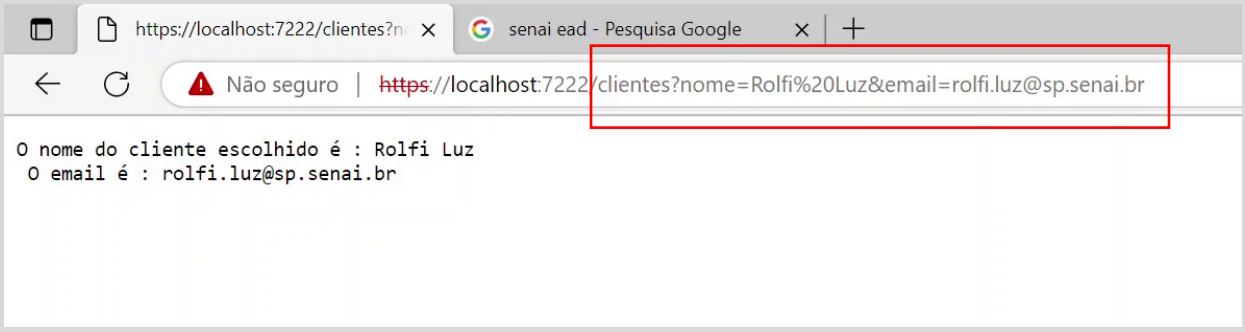
Importante!

Para que as alterações no código surtam efeito, salve o arquivo antes publicar o projeto. Para isso, use o atalho **CTRL + s**.

3. Na linha **MapGet** de **Clientes**, substitua o conteúdo pelo indicado na imagem.

```
        app.MapGet("/clientes", (string nome, string
email) => $"O nome do cliente escolhido é : { nome } \n O
email é : {email}" );
```

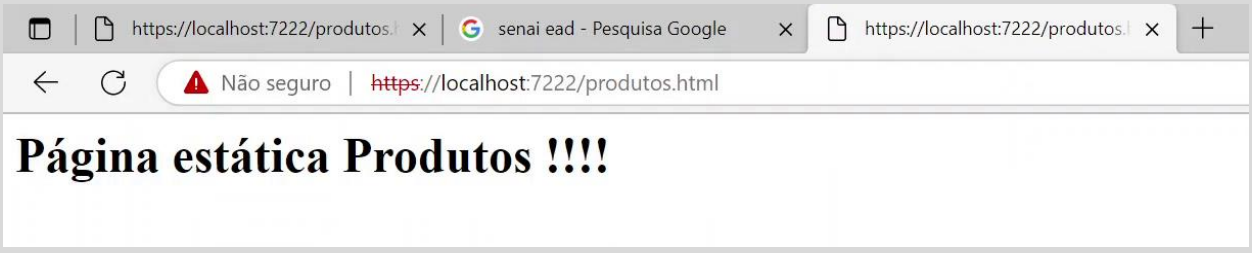
4. Faça mais um teste no terminal com o comando **dotnet run**. Acesse o link da página local e, em seguida, digite o caminho **/nome=Rolfi%20Luz&email=rolfi.luz@sp.senai.br**.



5. Agora, acesse o arquivo **produtos.html**. Vamos modificar o mapping de produtos para que seja retornada uma página em vez de uma mensagem. Para isso, pelo trecho destacado na imagem a seguir, localize a linha **app.MapGet("/produtos"**.

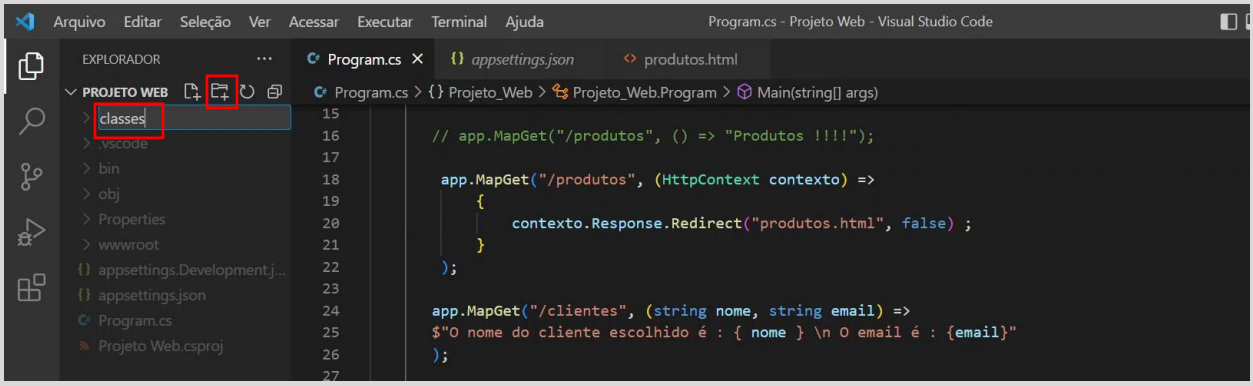
```
app.UseStaticFiles();

app.MapGet("/", () => "Hello World!");

app.MapGet("/cliente" , () => "Cliente !!!!!" );

// app.MapGet("/produtos", () => "Produtos !!!!");

app.MapGet("/produtos", (HttpContext contexto) =>
    {
        contexto.Response.Redirect("produtos.html", false) ;
    }
);
```

6. Mais uma vez, realize o teste no terminal com o comando **dotnet run** e acesse o link. No navegador, digite o caminho **/produtos.html**.

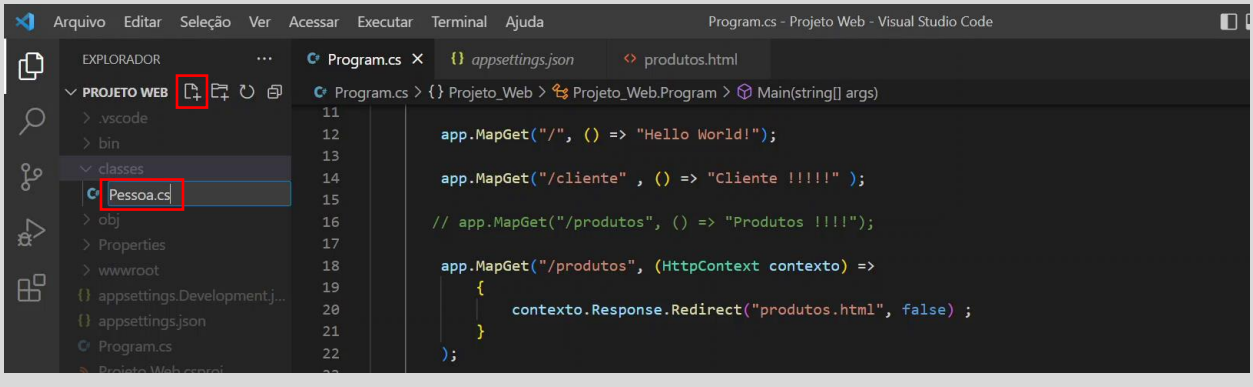


Criando classes no Back-end

1. Crie uma nova pasta e nomeie-a de **classes**.



2. Crie um novo arquivo dentro dessa pasta e nomeie-o de **Pessoa.cs**.



3. Abra o arquivo **Pessoa.cs** e digite o código indicado na imagem a seguir.

```
class Pessoa {

    public int id {set; get ;}

    public string? nome {set; get ;}

}
```

4. Em **Program.cs**, logo após o MapGet de clientes, insira a linha destaca na imagem a seguir.

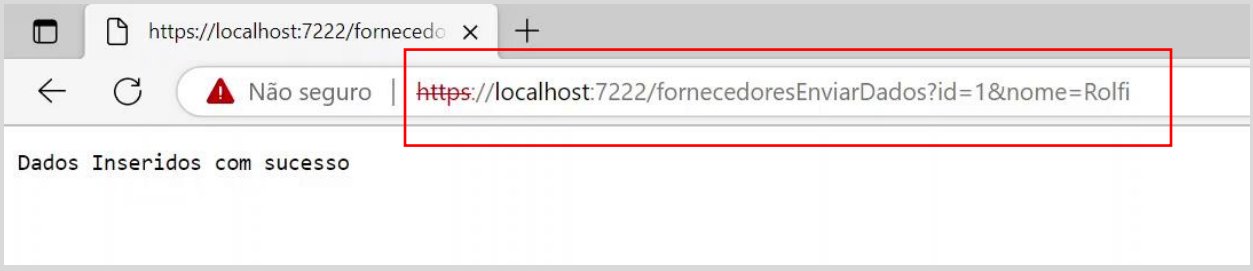
```
        app.MapGet("/clientes", (string nome, string email)
=>
        $"O nome do cliente escolhido é : { nome } \n O
email é : {email}"
        );

        Pessoa p1=new Pessoa() { id=1 ,  nome="Ana"};

        app.MapGet("/fornecedores", () =>
           $"O fornecedor é: {p1.id} - {p1.nome}"
        );
```

5. Faça um teste no terminal usando o comando **dotnet run** e acesse o link da página. No navegador, digite: **/fornecedores.html**.

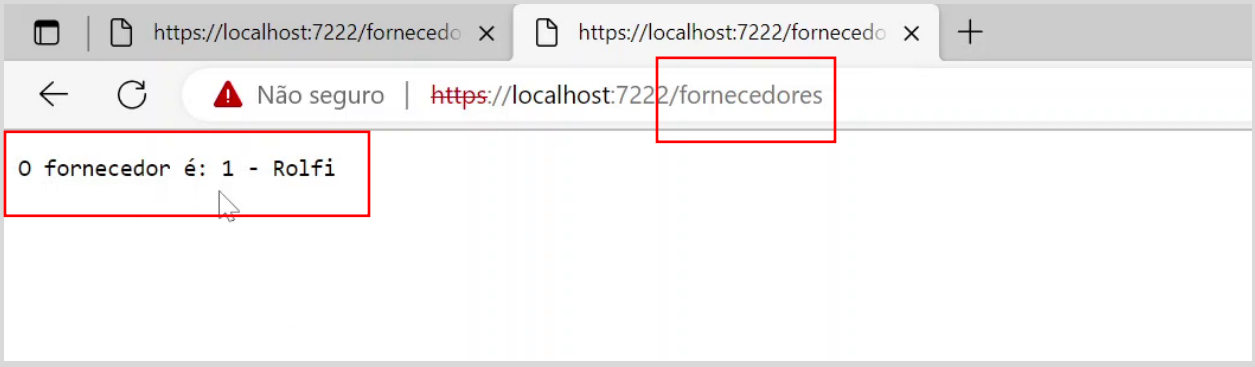
6. Também é possível enviar dados do Front para o Back-end. De volta ao **Program.cs**, localize o MapGet de **Fornecedores** e digite, logo após ele, estas linhas:

```
        app.MapGet("/fornecedoresEnviarDados", (int id,
string nome) =>
         {
                  p1.id = id;
                  p1.nome=nome;
                  return "Dados Inseridos com sucesso";
         } ) ;
```

7. Realize um teste no terminal usando o comando **dotnet run**. No navegador, digite **/fornecedoresEnviarDados?id=1&nome=Rolfi**.



8. Ainda no navegador, confira se o fornecedor foi alterado. Para isso, acesse o caminho: **/fornecedores**.



9. Agora, vamos imprimir a classe fornecedores com comandos em HTML. Para isso, retorne ao **Program.cs** e faça a substituição do código após o mapping **"/clientes"**, conforme destacado na imagem.

```
        app.MapGet("/clientes", (string nome, string email) =>
        $"O nome do cliente escolhido é : { nome } \n O email é :
{email}"
        );

        // text/plain
        //app.MapGet("/fornecedores", () =>
        //  $"O fornecedor é: {p1.id} - {p1.nome}"
        // );


        app.MapGet("/fornecedores", (HttpContext contexto) => {
            contexto.Response.WriteAsync("<h1> Fornecedores </h1>");

        } ) ;
```

10. Agora, faça a substituição do código no mapping **“/fornecedores”** conforme destacado na imagem.

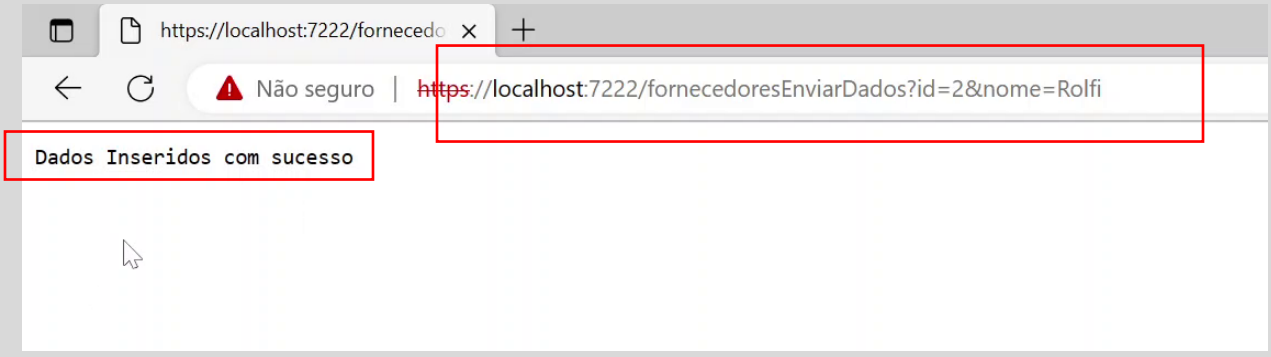
```
        app.MapGet("/clientes", (string nome, string email) =>
        $"O nome do cliente escolhido é : { nome } \n O email é :
{email}"
        );

         Pessoa p1=new Pessoa() { id=1 ,  nome="Ana"};

        // text/plain
        // app.MapGet("/fornecedores", () =>
        //   $"O fornecedor é: {p1.id} - {p1.nome}"
        // );

         app.MapGet("/fornecedores", (HttpContext contexto) => {
                contexto.Response.WriteAsync(“<h1> Forncedores
</h1>");
             });

         app.MapGet("/fornecedoresEnviarDados", (int id, string nome)
=>
         {
                p1.id = id;
                p1.nome=nome;
                return "Dados Inseridos com sucesso";

         }  ) ;
```

11. Faça um teste no terminal utilizando o comando **dotnet run**. No navegador, digite o caminho: **/fornecedores.html**.

12. Em **Program.cs**, faça a substituição do código no mapping **"/fornecedores"** conforme mostrado a seguir.

```
        // text/plain
        // app.MapGet("/fornecedores", () =>
        //   $"O fornecedor é: {p1.id} - {p1.nome}"
        // );

         app.MapGet("/fornecedores", (HttpContext contexto) =>

                string pagina = "<h1> Fornecedores </h1>";
                pagina = pagina + $" <h2> ID:{p1.id} – Nome
{p1.nome} </h2>";

                contexto.Response.WriteAsync(pagina);
            });
```

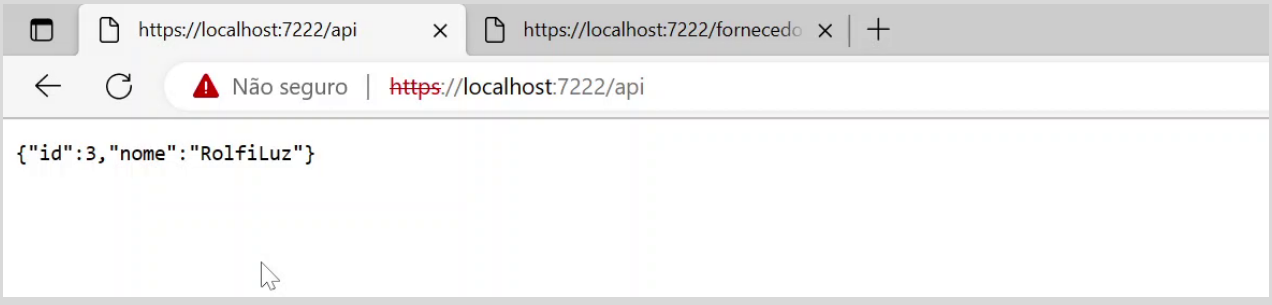
13. Faça um teste no terminal com o comando **dotnet run** acessando o link. No navegador, digite: **/fornecedoresEnviarDados?id=2&nome=Rolfi**



14. Agora, vamos incluir outro MapGet. Copie o trecho destacado a seguir antes da linha **app.Run( );**.

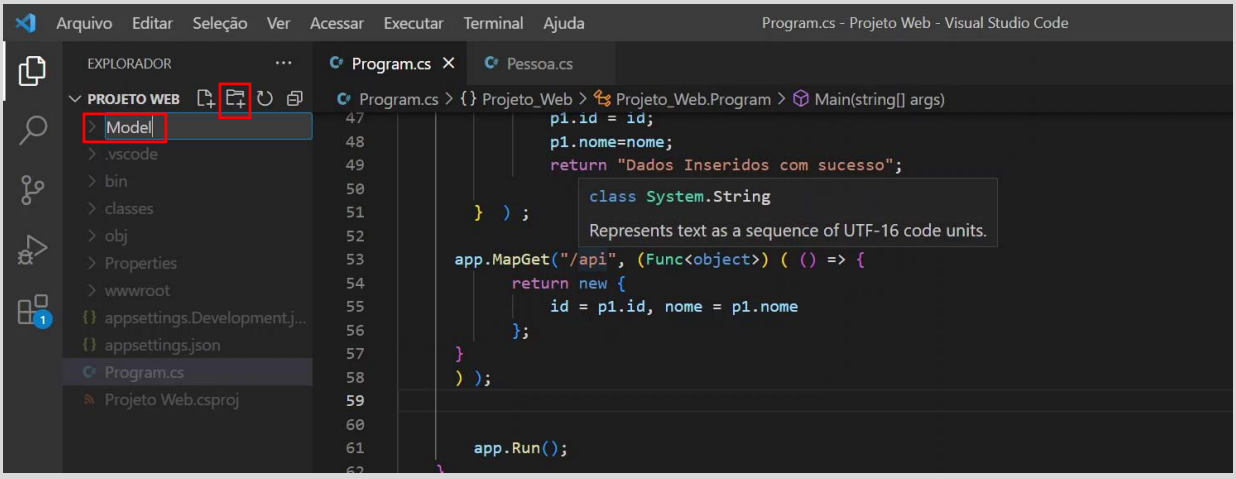
```
        app.MapGet("/api", (Func<object>) ( ()=> {
        return new {
            id = p1.id, nome = p1.nome
        };
            }
    ) ) ;

        app.Run();
    }
}
```

15. Após inserir esses dados, acesse o link da página local e o caminho **/api**.

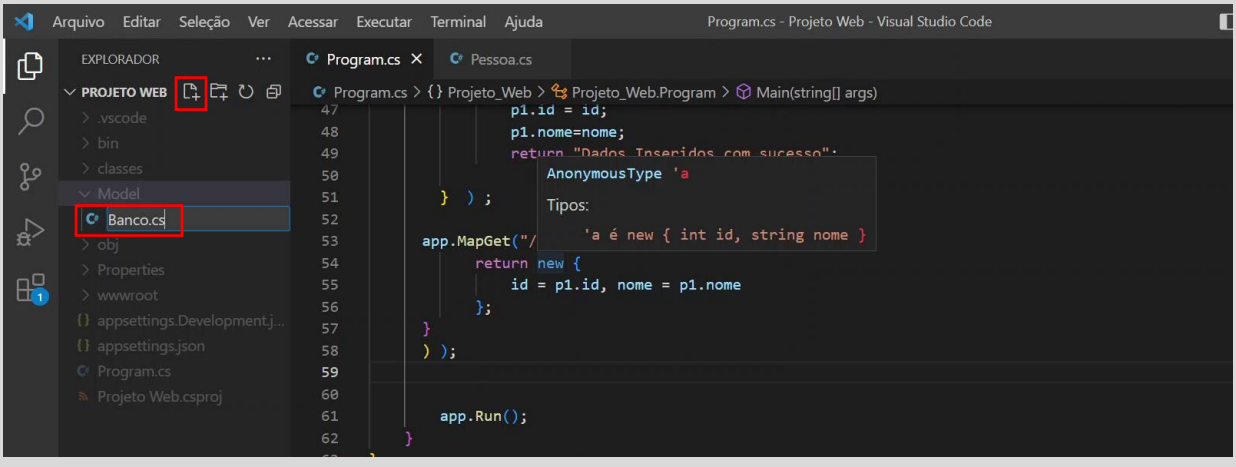


Banco de dados

1. Crie uma nova pasta e nomeie-a de **Model**.



2. Crie um novo arquivo dentro da pasta Model e nomeie-o de **Banco.cs**.



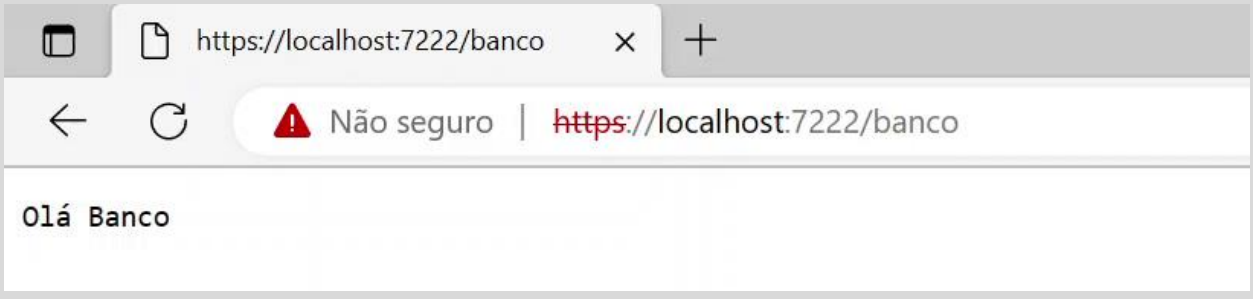
3. No arquivo **Banco.cs**, digite o seguinte código:

```
class Banco
{

public string mensagem = "Olá Banco";
```

4. Volte ao arquivo **Program.cs** e digite, na linha antes de **app.Run**, o seguinte trecho:

```
Banco dba=new Banco();
dba.carregarBanco();
app.MapGet("/banco", () => {

    return dba.mensagem;
} ) ;
```

5. Faça um teste no terminal utilizando o comando **dotnet run** e acessando o link. No navegador, digite: **/banco**.



6. No terminal, digite o comando a seguir e aperte **Enter**. Isso adicionará o conector do banco.

```
dotnet add package System.Data.SqlClient
```

7. Para atualizar o projeto com os pacotes novos, digite no terminal o comando a seguir e aperte **Enter**.

```
dotnet restore
```

8. Agora, vamos digitar os comandos relativos ao banco de dados. Logo na primeira linha do arquivo **Banco.cs**, insira o código:

```
using System.Data.SqlClient;
```

9. Após a primeira linha do arquivo **Banco.cs**, copie o trecho destacado e cole-o após a atribuição “public string mensagem”:

```
using System.Data.SqlClient;

 class Banco
    {  //chave de abertura

    public string mensagem = "Olá Banco";


    private List<Pessoa> lista=new List<Pessoa>();

    public List<Pessoa> GetLista()
    {
        return lista;
    }

 } //chave de fechamento
```

Importante!

Na linguagem C#, as chaves **{ }** definem o início e o final de um bloco de códigos. Assim, na classe Banco.cs, deve haver um bloco representado com as chaves de abertura **{** e, na última linha, de fechamento **}**.

É muito comum errar as chaves ao digitar os códigos de programação, por isso, atente-se para que para cada chave de abertura tenha o respectivo fechamento, tanto na função GetLista quanto nas demais que deverão estar no arquivo Banco.cs.

10. Antes da chave de fechamento do arquivo **Banco.cs**, cole o código a seguir.

```
    public void carregarBanco()
    {
        try
                {
                    SqlConnectionStringBuilder builder = new
SqlConnectionStringBuilder(
                    "User ID=sa;Password=12345;" +
                    "Server=localhost\\SQLEXPRESS;" +
                    "Database=projetoclientes;" +
                    "Trusted_Connection=False;"
                    );

                    using (SqlConnection conexao = new
SqlConnection(builder.ConnectionString))
                    {
                        String sql = "SELECT * FROM clientes";
                        using (SqlCommand comando = new SqlCommand(sql,
conexao ))
                        {
                            conexao.Open();
                            using (SqlDataReader tabela =
comando.ExecuteReader()) {

                                while(tabela.Read())
                                {
                                    System.Console,WriteLine(tabela["nome"]);

                                }
                            }
                        }
                    }
                }
        catch (Exception e)
                {
                    System.Console.WriteLine("Erro:"+e.ToString());
                }
    }


    } //chave de fechamento
```

Importante!

O login utilizado no banco de dados será o **login sa**, porque estamos em um ambiente de desenvolvimento, rodando de uma máquina que é o servidor de aplicação e de banco de dados.
No entanto, em aplicações comerciais ou para rodar em rede, é recomendado criar um login diferenciado com o mínimo de permissões possível.

11. Para criar a função de carregamento de dados na lista, localize a chave do **while** dentro do código e insira o trecho destacado na imagem a seguir.

```
while(tabela.Read())
{
    //System.Console,WriteLine(tabela["nome"]);
    //Pessoa p1=new Pessoa() { id=1 ,  nome="Ana"};

    lista.Add(new Pessoa(){
        id = Convert.ToInt32(tabela["id"]),
        nome = tabela["nome"].ToString(),

    });
}
```

Dica!

É possível imprimir esses dados e trazer as informações para o Front-end. Quando são transformados em lista, esses dados podem ser melhor manipulados.

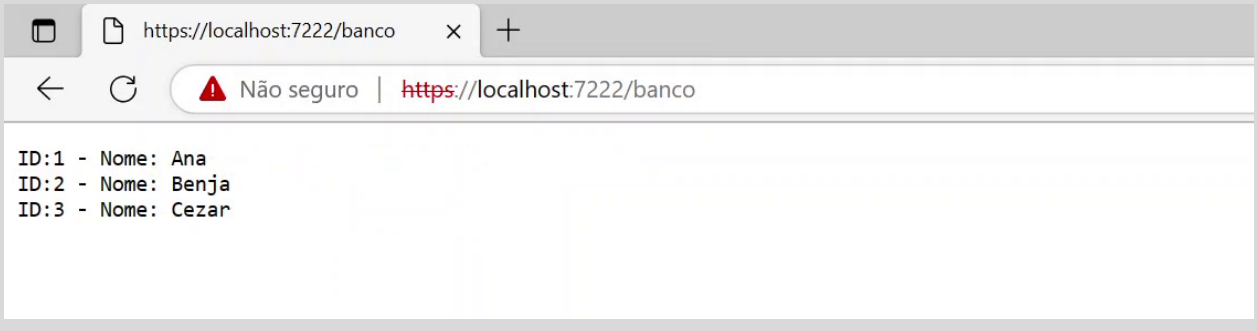
12. Retorne ao arquivo **Program.cs** e copie o trecho destacado na imagem. Nesse ponto, a impressão no Front é como texto puro.

```
    Banco dba=new Banco();
    dba.carregarBanco();
    app.MapGet("/banco", () => {

    var valoresdalista="";
    List<Pessoa> listaaux = dba.GetLista();
    foreach(Pessoa aux in listaaux)
    {
        valoresdalista = valoresdalista + $"ID:{aux.id} -
Nome:{aux.nome}\n";
    }
        return valoresdalista;
    });

        app.Run();
    }
}
```

13. Para impressão em html no front end, volte ao arquivo **Program.cs** e substitua o trecho destacado abaixo, na linha antes de **app.Run**.

```
    Banco dba=new Banco();
    dba.carregarBanco();
    app.MapGet("/banco", (HttpContext contexto) => {

    var valoresdalista="";
    List<Pessoa> listaaux = dba.GetLista();
    foreach(Pessoa aux in listaaux)
    {
        valoresdalista = valoresdalista + $"<b> ID: </b>{aux.id} - <b>
Nome:{aux.nome} </b> <br>";
    }
        //return valoresdalista;
        contexto.Response.WriteAsync(valoresdalista);

    });


        app.Run();
    }
}
```

14. Faça um teste no terminal utilizando o comando **dotnet run** e acessando o link. No navegador, confira o resultado em: **/banco**.



15. Ainda é possível melhorar o código. Confira o código completo de **Program.cs**:

```
namespace Projeto_Web;

public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);
        var app = builder.Build();

        app.UseStaticFiles();

        app.MapGet("/", () => "Hello World!");

        app.MapGet("/cliente" , () => "Cliente !!!!!" );

       // app.MapGet("/produtos", () => "Produtos !!!!");

        app.MapGet("/produtos", (HttpContext contexto) =>
            {
                contexto.Response.Redirect("produtos.html", false)
;
            }
        );

       app.MapGet("/clientes", (string nome, string email) =>
       $"O nome do cliente escolhido é : { nome } \n O email é :
{email}"
       );

        Pessoa p1=new Pessoa() { id=1 ,  nome="Ana"};

        // text/plain
        //app.MapGet("/fornecedores", () =>
        //  $"O fornecedor é: {p1.id} - {p1.nome}"
        // );
```

Continuação do código:

```
        app.MapGet("/fornecedores", (HttpContext contexto) => {

           string pagina = "<h1> Fornecedores </h1>";
           pagina = pagina + $"<h2> ID:{p1.id} - Nome : {p1.nome} </h2>";
            contexto.Response.WriteAsync(pagina);

        }
        );

        app.MapGet("/fornecedoresEnviarDados", (int id, string nome) =>
        {
                p1.id = id;
                p1.nome=nome;
                return "Dados Inseridos com sucesso";

        }  ) ;

        app.MapGet("/api", (Func<object>) ( ()=> {
        return new {
            id = p1.id, nome = p1.nome
        };
            }
    ));


    Banco dba=new Banco();
    dba.carregarBanco();
    app.MapGet("/banco", (HttpContext contexto) => {

    var valoresdalista="";
    List<Pessoa> listaaux = dba.GetLista();
    foreach(Pessoa aux in listaaux)
    {
        valoresdalista = valoresdalista + $"<b> ID: </b>{aux.id} - <b>
Nome:{aux.nome} </b> <br>";
    }
        //return valoresdalista;
        contexto.Response.WriteAsync(valoresdalista);
    });

        app.Run();
    }
}
```

16. Podemos melhorar o código do model **Banco.cs** também, confira:

```
using System.Data.SqlClient;

    class Banco
    {

    public string mensagem = "Olá Banco";
    private List<Pessoa> lista=new List<Pessoa>();

    public List<Pessoa> GetLista()
    {
        return lista;
    }

    public void carregarBanco()
    {
        try
                {
                    SqlConnectionStringBuilder builder = new
SqlConnectionStringBuilder(
                    "User ID=sa;Password=12345;" +
                    "Server=localhost\\SQLEXPRESS;" +
                    "Database=projetoclientes;" +
                    "Trusted_Connection=False;"
                    );

                    using (SqlConnection conexao = new
SqlConnection(builder.ConnectionString))
                    {
```

Continuação do código

```
                    {
                        String sql = "SELECT * FROM clientes";
                        using (SqlCommand comando = new SqlCommand(sql,
conexao ))
                        {
                            conexao.Open();
                            using (SqlDataReader tabela =
comando.ExecuteReader())
                            {

                                while(tabela.Read())
                                {
                                    lista.Add(new Pessoa()
                                    {
                                        id =
Convert.ToInt32(tabela["id"]),
                                        nome =
tabela["nome"].ToString(),

                                    });
                                }
                            }
                        }
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("Erro:"+e.ToString());
                    }
    }


    }
```