

# Projeto de Banco de Dados e OO .NET

CRUD e Webservice REST com ASP.NET

# Objeto Relational Mapping



A vertical stack of four blue rectangular boxes, each preceded by a white circle. A thin blue line connects the top-left of the first circle to the top-left of the second, and so on, ending at the bottom-left of the fourth. The circles are positioned to the left of the boxes, and the line starts from the top-left of the first circle and ends at the bottom-left of the fourth.

Classes

Repositório

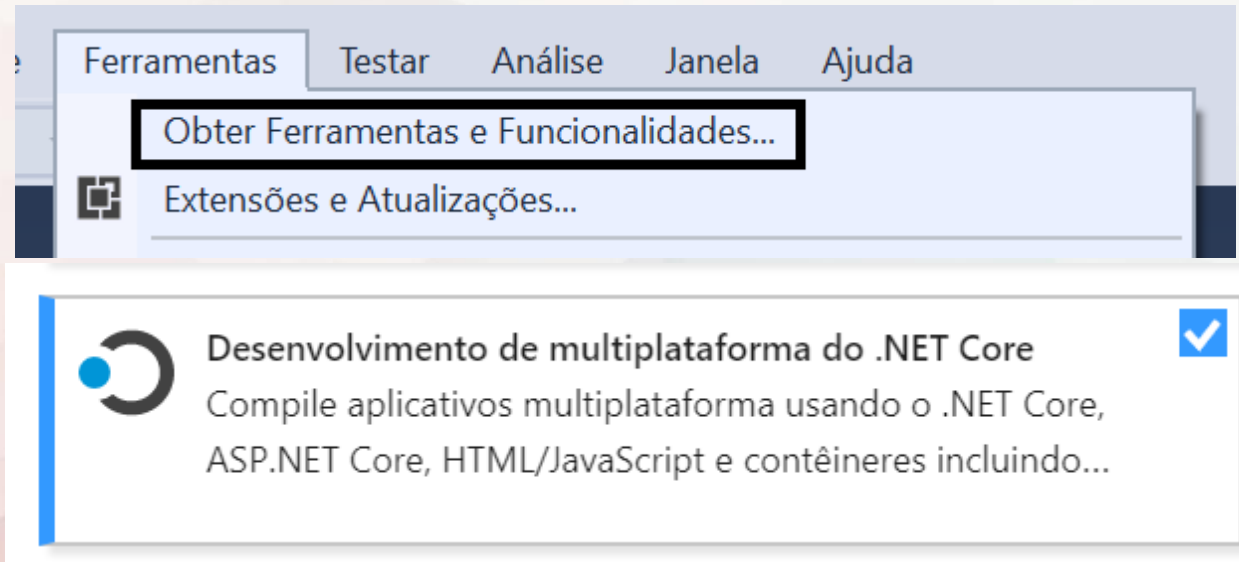
ORM (Ex.: **Entity Framework**)

Drivers de Acesso ao BD

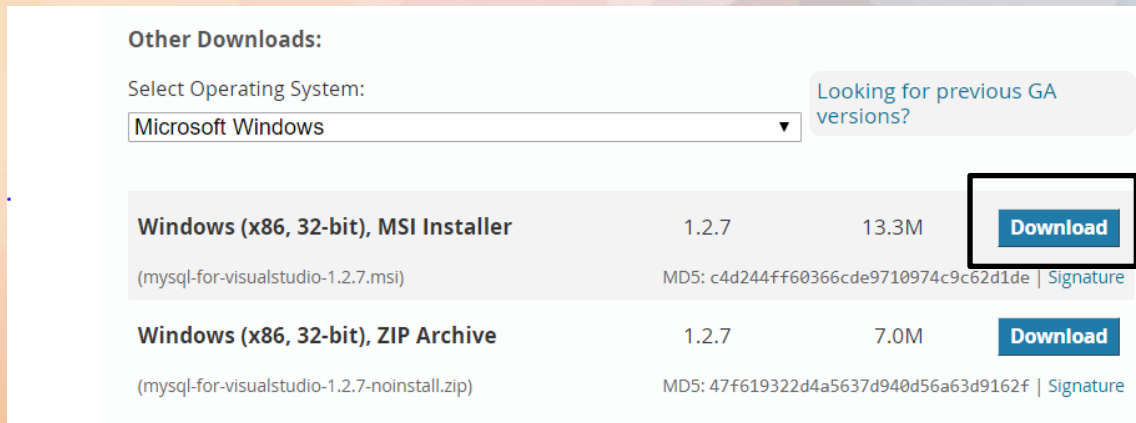
# Módulos Necessários

Instalar Módulo do Visual Studio

Desenvolvimento de multiplataforma do .NET Core



<https://dev.mysql.com/downloads/windows/visualstudio/>

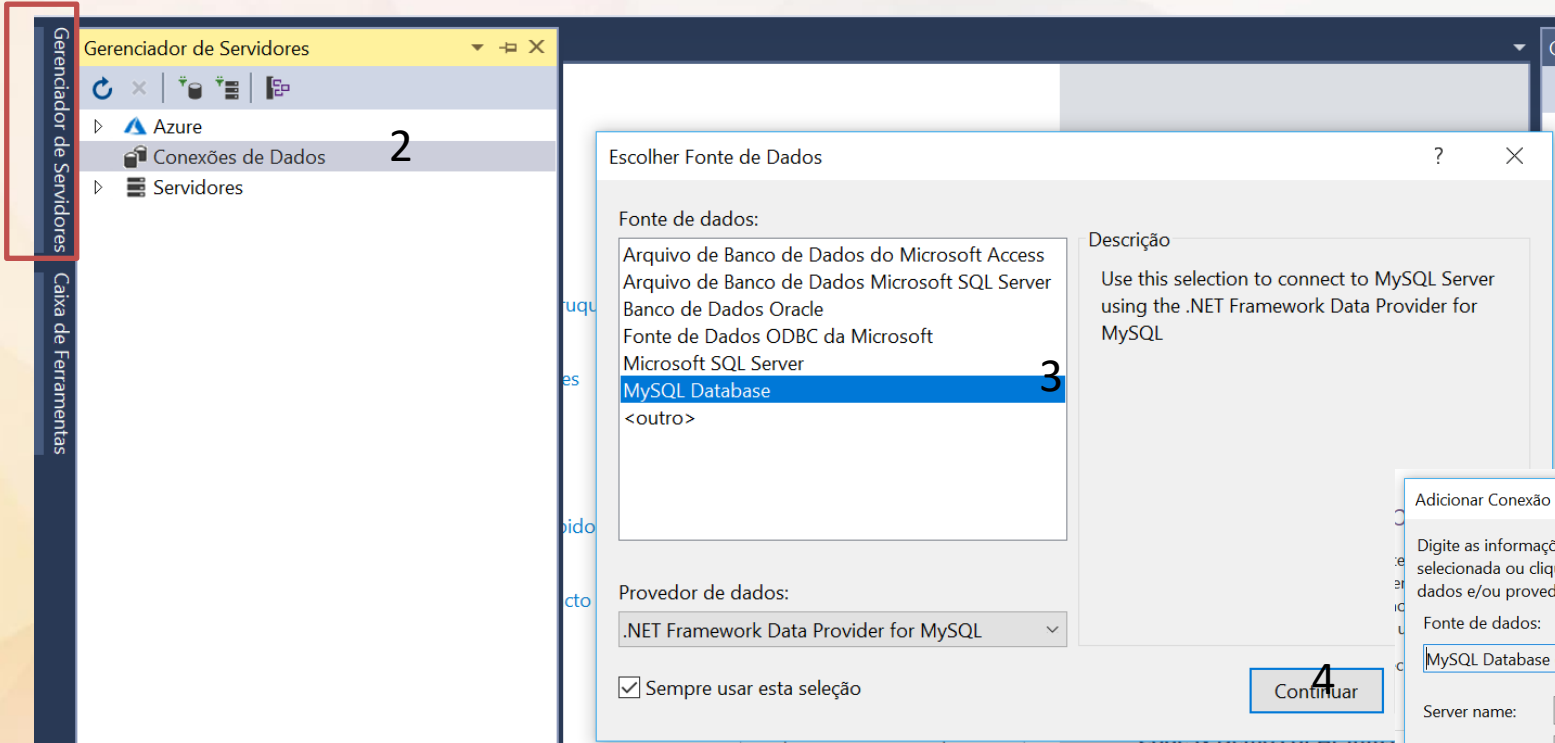


Instalar MySQL for Visual Studio

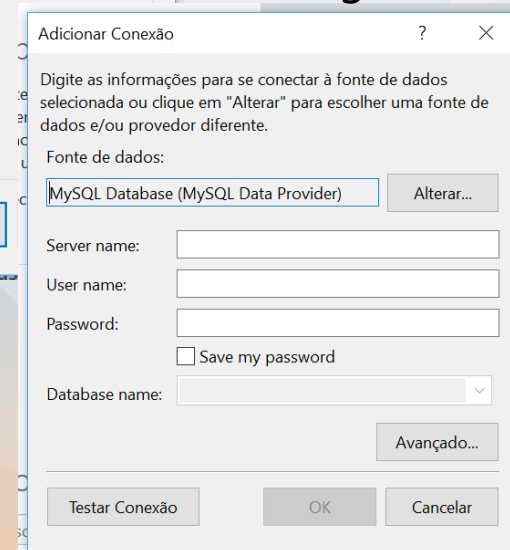
Página oficial do MySQL

# Conectando o Banco de Dados

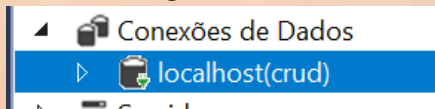
1



5



6



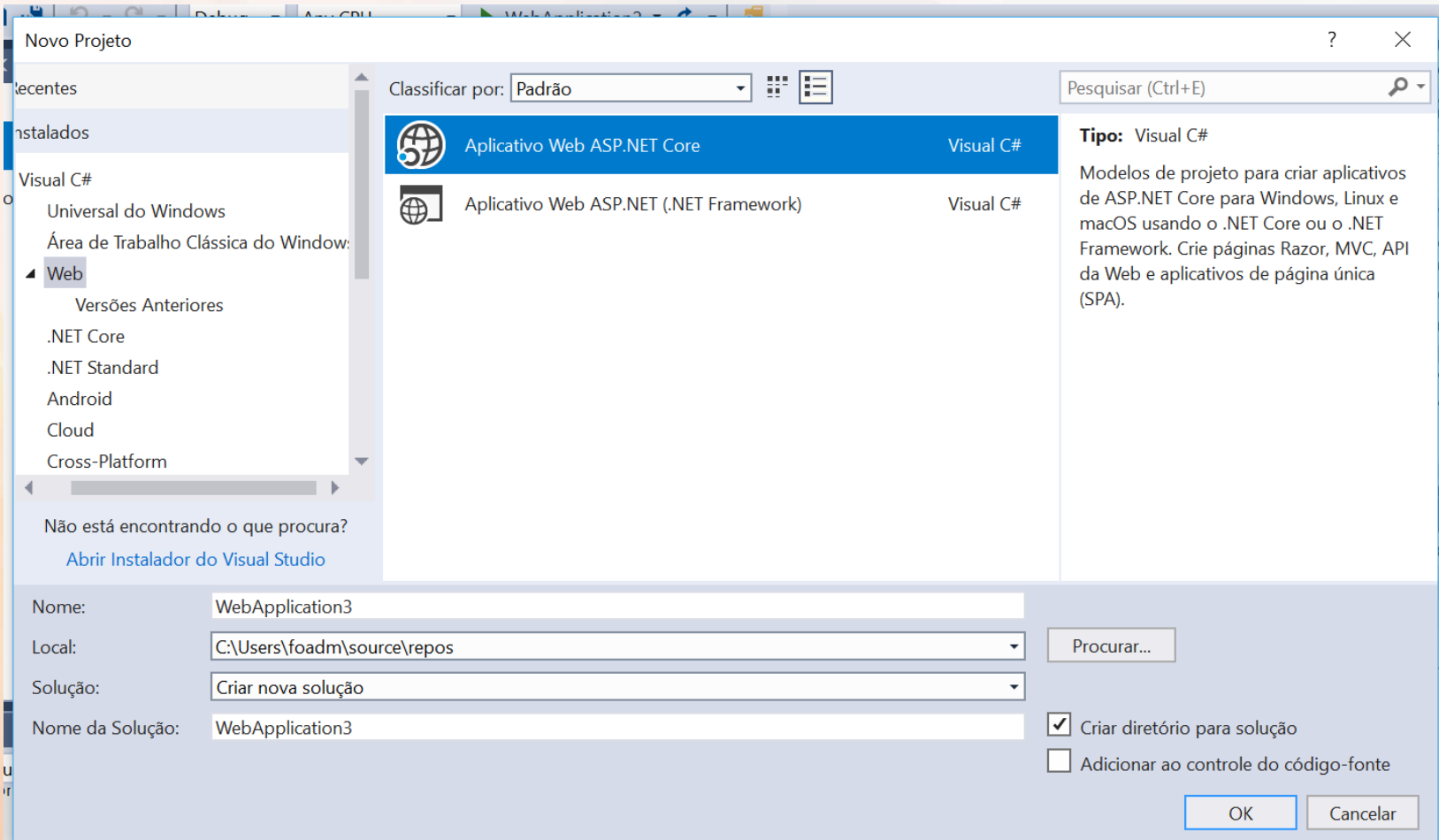
Server name: localhost

User name: root

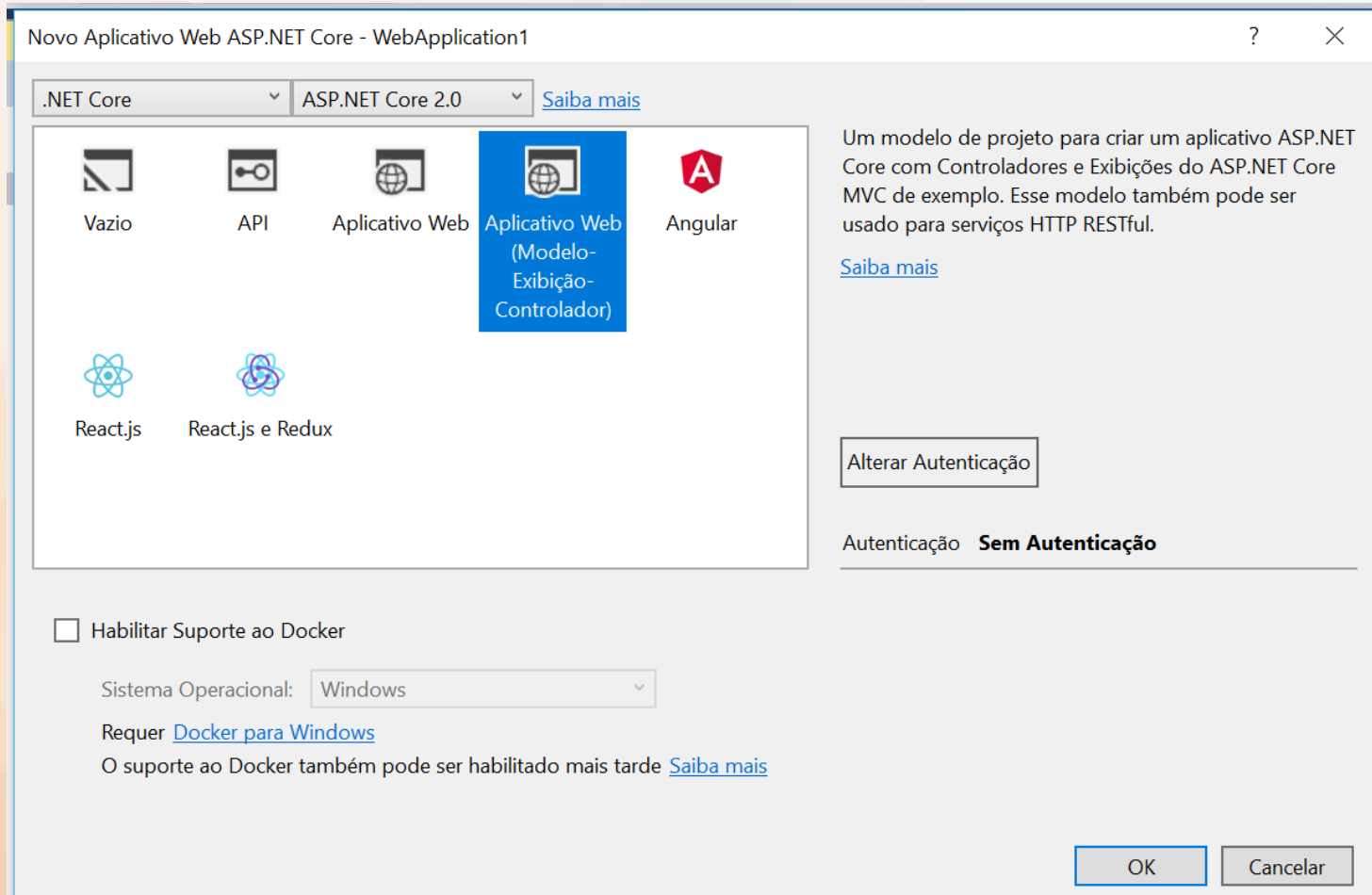
Password: (sua senha)

Database: (Crie um novo e escolha aqui)

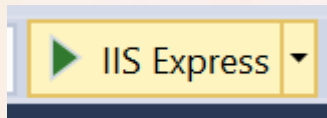
# Criando um Projeto WEB



# Aplicativo Web MVC



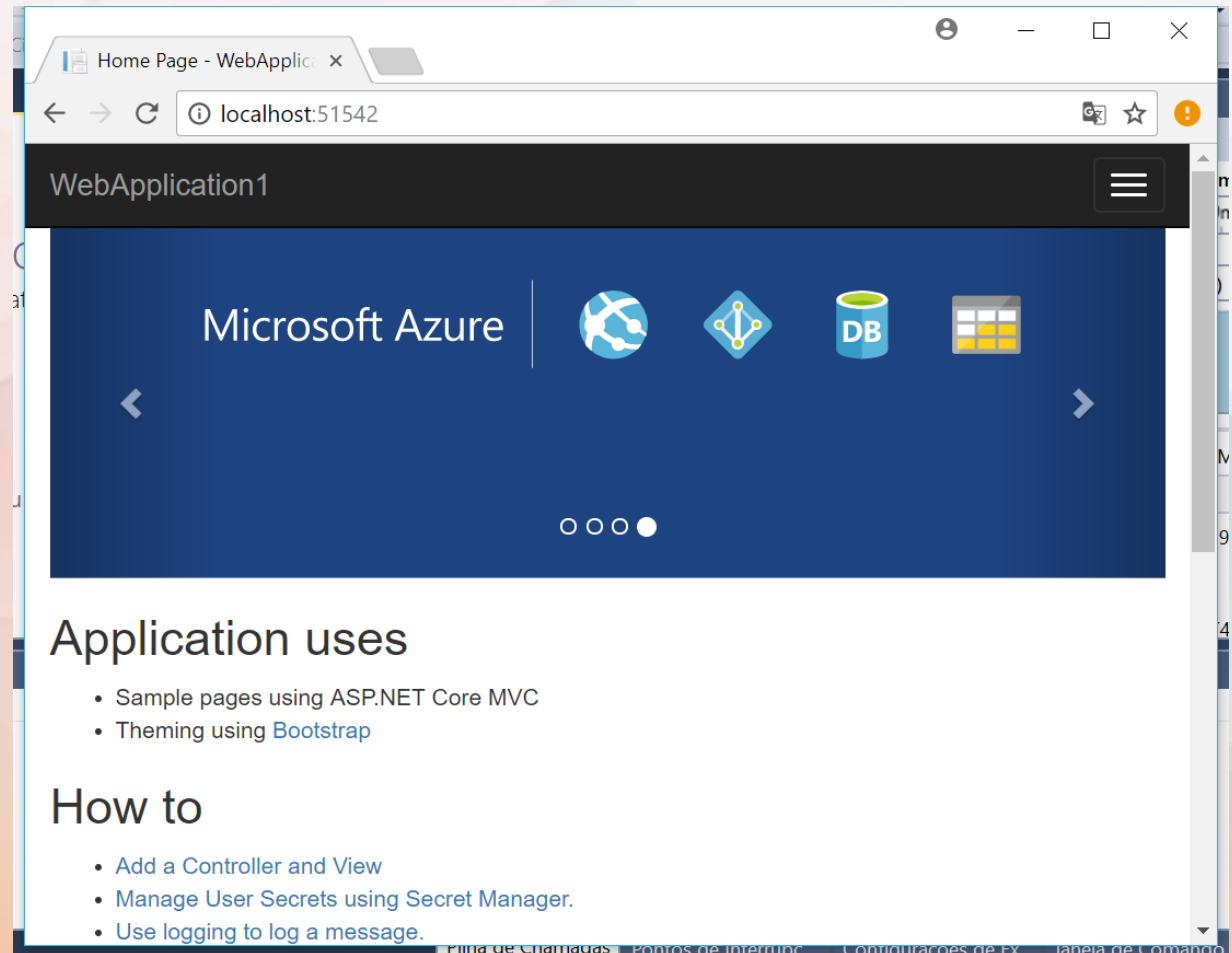
# Aplicação Padrão



Esse é o template padrão do ASP.NET.

Ele possui estilização utilizando Bootstrap

Para saber mais:  
<https://getbootstrap.com/>



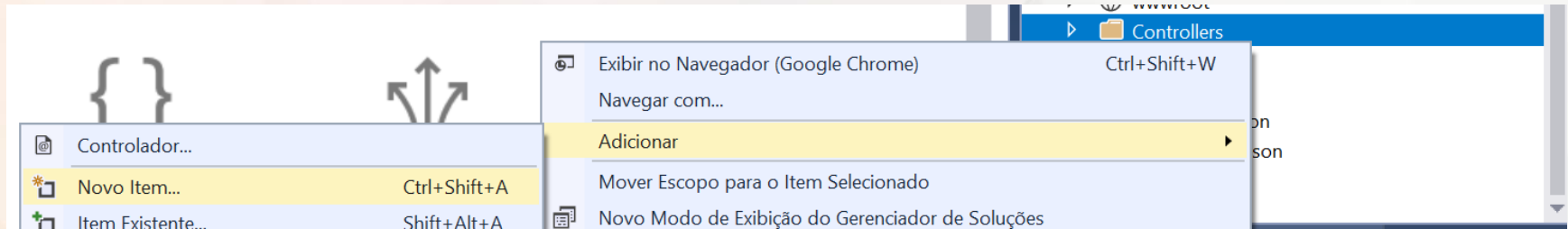
# MVC - *model-view-controller*

- **Design Pattern** que faz a separação entre os dados (models), interfaces do usuário (view) e controle (controller)
  - **M**odels: Classes que representam os dados da aplicação. No model também ficam as regras de validação de dados.
  - **V**iews: Template de apresentação dos dados
  - **C**ontrollers: Classe que processa as requisições de dados, obtenção de dados e retorno das requisição.

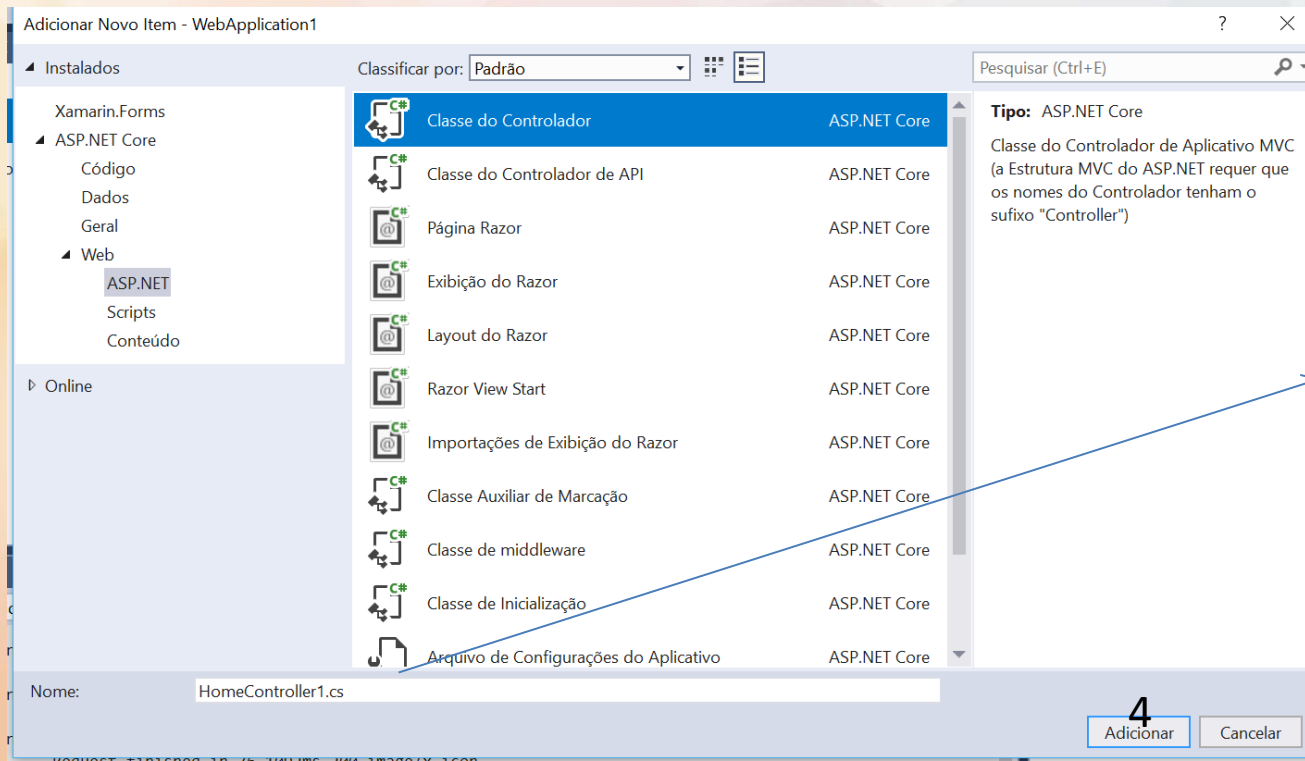


# Adicionando Controllers

1. Clique na pasta Controllers no lado direito da tela no Gerenciador de Soluções (Se ele não estiver aparecendo aperte Ctrl + Alt + L) e escolha Adicionar -> Novo Item



## 2. Escolha Classe do Controlador



3. Altere o nome para HelloWorldController.cs

# Criando Actions para o Controlador

Esse é o controlador padrão criado

```
namespace WebApp.Controllers
{
    0 referências
    public class HelloWorldController : Controller
    {
        // GET: /<controller>/
        0 referências | 0 solicitações | 0 exceções
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

0 referências | 0 exceções

```
public string Index()
{
    return "Essa é a ação padrão";
}
```

Note que é string agora

1. Altere o método Index padrão pelo código acima

0 referências | 0 exceções

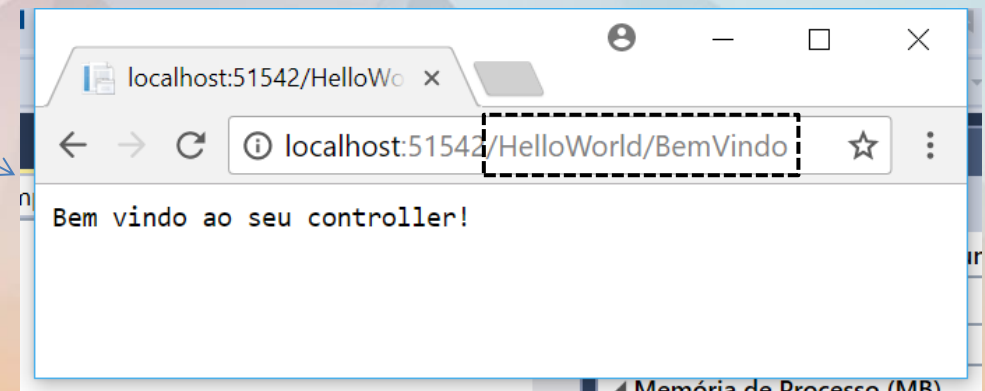
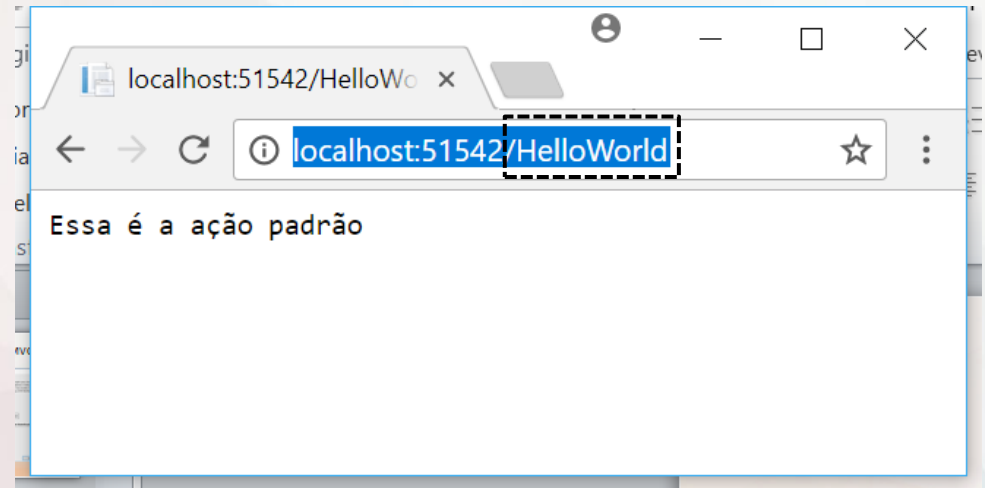
```
public string BemVindo()
{
    return "Bem vindo ao seu controller!";
}
```

2. Crie um novo método BemVindo conforme acima

# Testando Nosso Controlador

```
// GET: /HelloWorld/  
0 referências | 0 exceções  
public string Index()  
{  
    return "Essa é a ação padrão";  
}
```

```
//  
// GET: /HelloWorld/BemVindo/  
0 referências | 0 exceções  
public string BemVindo()  
{  
    return "Bem vindo ao seu controller!";  
}
```



# Controlador com Parâmetros

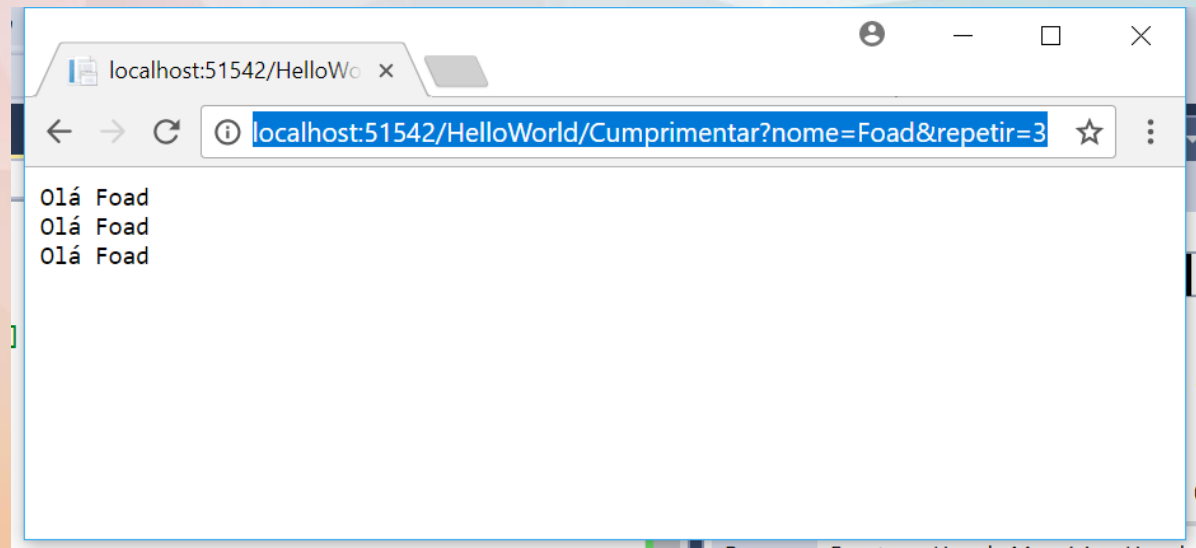
```
// GET: /HelloWorld/Cumprimentar?nome=[NOME]&repetir=[NUM]
0 referências | 0 exceções
public string Cumprimentar(string nome, int repetir = 1)
{
    string ret = "";
    for(int i=0; i<repetir; i++)
    {
        ret += $"Olá {nome} \n";
    }
    return ret;
}
```

1. Crie mais um método chamado Cumprimentar no HelloWorldController.cs conforme ao Lado

2. Teste seu novo método (action) conforme ao lado.

Veja que os parâmetros são passados após o ? e separados por &

?parametro=valor&parametro2=valor2

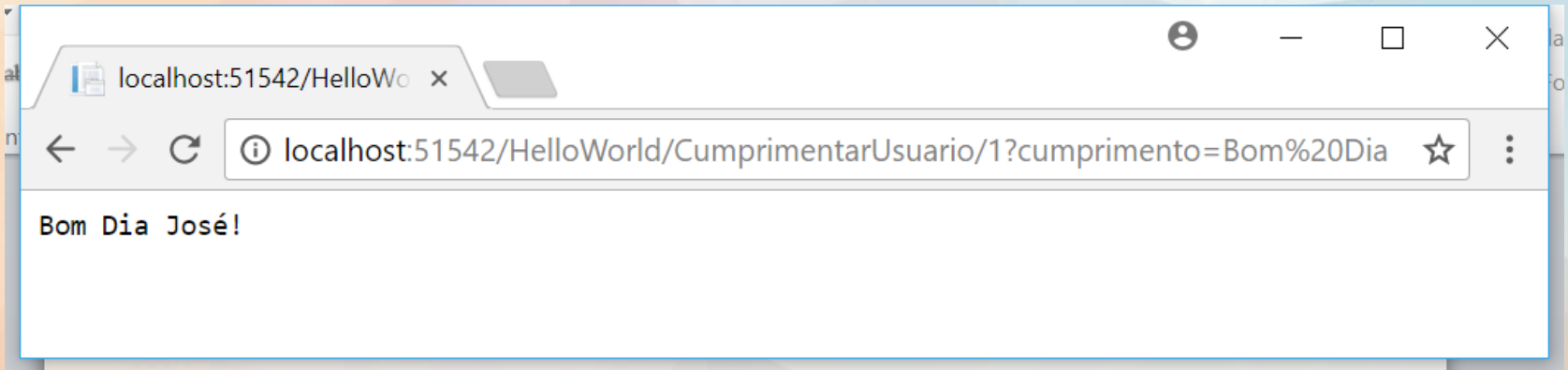


# Controlador com ID

```
// GET: /HelloWorld/CumprimentarUsuario/[ID]?cumprimento=[Cumprimento]
0 referências | 0 exceções
public string CumprimentarUsuario(int ID, string cumprimento)
{
    string[] nomes = { "Foad", "José", "Maria", "Pedro" };

    return $"{cumprimento} {nomes[ID]}!";
}
```

1. Crie mais um método chamado CumprimentarUsuario no HelloWorldController.cs conforme ao Lado



2. Teste seu novo método (action) conforme url acima. Note que o ID no caso com valor igual a 1 aparece logo após o / sem necessidade de usar ?ID=1

# Rota (URL) Padrão Startup.cs

```
0 referências | 0 exceções
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseBrowserLink();
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

Veja a que a rota segue o padrão CONTROLLER/ACTION/ID

Ex.: HelloWorld/CumprimentarUsuario/1?cumprimento=oi

Controller

Action

ID (opcional)

Outros Parâmetros

# Criando Rotas Personalizadas

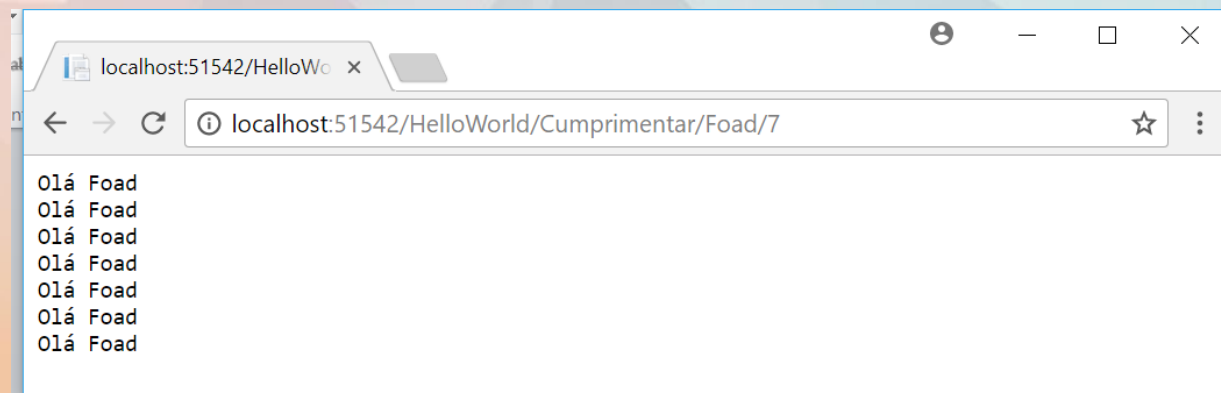
```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");

    routes.MapRoute(
        name: "HelloWorld#Cumprimentar",
        template: "{controller}/{action}/{nome}/{repetir}"
    );
});
```

1. Adicione essa nova rota ao arquivo Startup.cs

Nova Rota => HelloWorld/Cumprimentar/Parâmetro Nome/Parâmetro Repetir

2. Teste sua nova rota personalizada

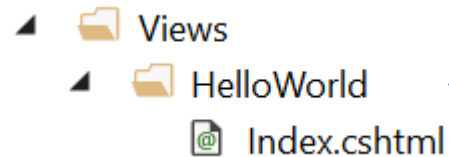


# Criando Views

1. Mude no arquivo HelloWorldController.cs o método Index() conforme ao lado

```
// GET: /HelloWorld/  
0 referências | 0 solicitações | 0 exceções  
public ActionResult Index()  
{  
    return View();  
}
```

2. Crie uma pasta HelloWorld no Views



```
Views  
└─ HelloWorld  
    └─ Index.cshtml
```

3. Clique na Pasta HelloWorld e escolha Adicionar -> Novo Item e crie um arquivo com nome Index.cshtml



Exibição do Razor

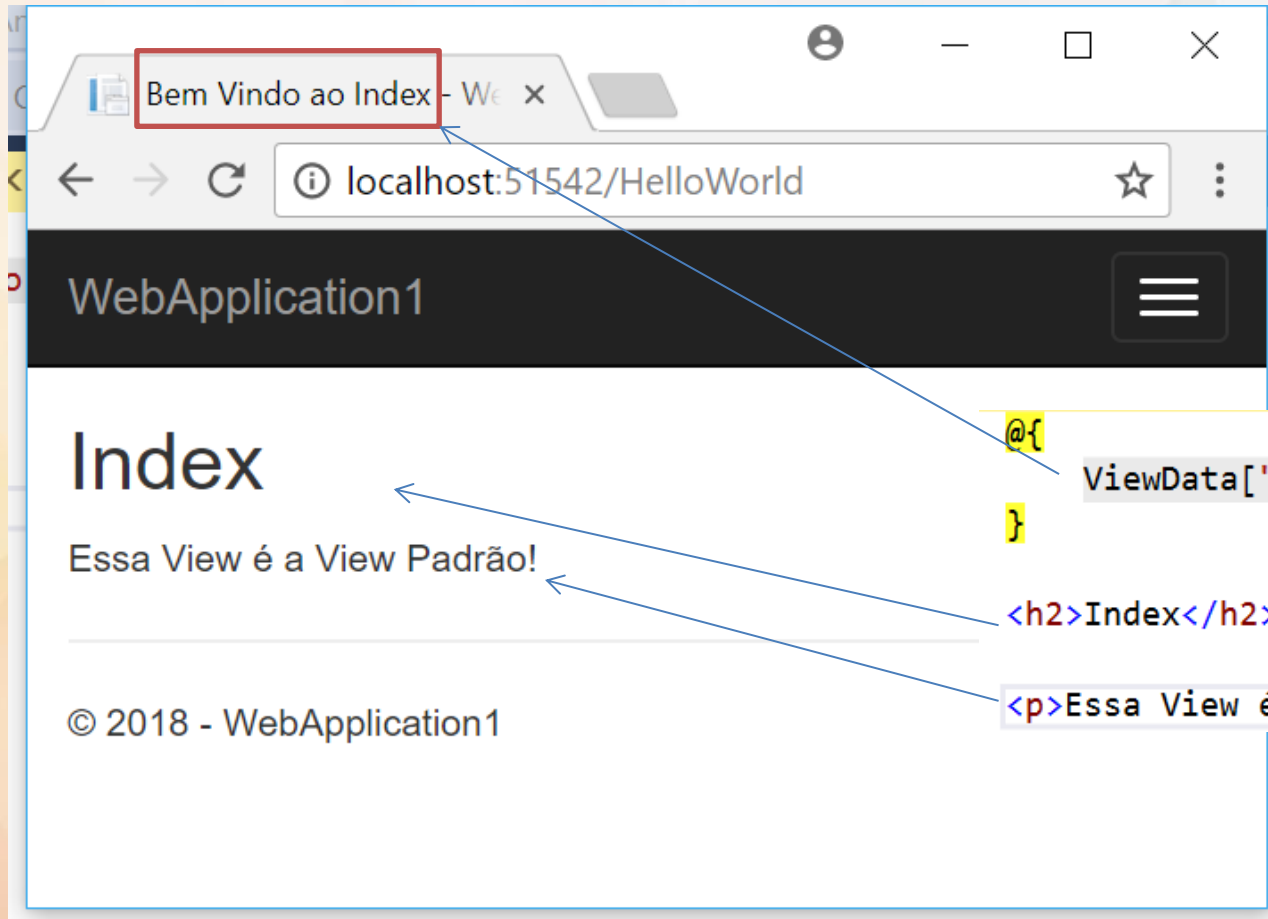
ASP.NET Core

4. Edite o arquivo Index.cshtml conforme ao lado

```
@{  
    ViewData["Title"] = "Bem Vindo ao Index";  
}  
  
<h2>Index</h2>  
  
<p>Essa View é a View Padrão!</p>
```



# Testando a View Recém Criada



```
@{
```

```
ViewData["Title"] = "Bem Vindo ao Index";
```

```
}
```

```
<h2>Index</h2>
```

```
<p>Essa View é a View Padrão!</p>
```

# Detalhes Layout Padrão das Views

## \_Layout.cshtml

```
_Layout.cshtml Program.cs Index.cshtml Startup.cs _ValidationScriptsPartial.cshtml
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>@ViewData["Title"] WebApplication1</title>
7
8   <environment include="Development">
9     <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
10    <link rel="stylesheet" href="~/css/site.css" />
11  </environment>
12  <environment exclude="Development">
13    <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min
14      asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
15      asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test
16    <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
17  </environment>
18 </head>
19 <body>
20   <nav class="navbar navbar-inverse navbar-fixed-top">
21     <div class="container">
22       <div class="navbar-header">
23         <button type="button" class="navb
24           <span class="sr-only">Toggle
25           <span class="icon-bar"></span>
26
27   @{}
28   ViewData["Title"] = "Bem Vindo ao Index";
29   {}
30
31   <h2>Index</h2>
32
33   <p>Essa View é a View Padrão!</p>
```

# Layout Padrão das Views

## \_Layout.cshtml

```
_Layout.cshtml Program.cs Index.cshtml Startup.cs
28         </button>
29         <a asp-area="" asp-controller="Home" asp-action="Index" class="navbar-brand">WebApplic
30     </div>
31     <div class="navbar-collapse collapse">
32         <ul class="nav navbar-nav">
33             <li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>
34             <li><a asp-area="" asp-controller="Home" asp-action="About">About</a></li>
35             <li><a asp-area="" asp-controller="Home" asp-action="Contact">Contact</a></li>
36         </ul>
37     </div>
38 </div>
39 </nav>
40 <div class="container body-content">
41     @RenderBody()
42     <hr />
43     <footer>
44         <p>&copy; 2018 - WebApplication1</p>
45     </footer>
46 </div>
47
48 <environment include="Development">
49     <script src="~/lib/jquery/dist/jquery.js">
50     <script src="~/lib/bootstrap/dist/js/boot
51     <script src="~/js/site.js" asp-append-ver
52 </environment>
```

ViewData["Title"] = "Bem Vindo ao Index";

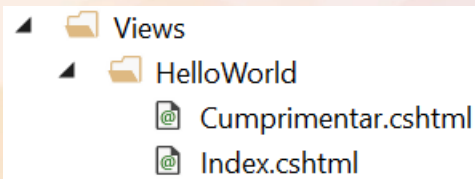
```
<h2>Index</h2>
<p>Essa View é a View Padrão!</p>
```

# Passando dados para View

```
// GET: /HelloWorld/Cumprimentar?nome=[NOME]&repetir=[NUM]
0 referências | 0 solicitações | 0 exceções
public ActionResult Cumprimentar(string nome, int repetir = 1)
{
    ViewData["Mensagem"] = "Olá " + nome + "!";
    ViewData["Repetir"] = repetir;

    return View();
}
```

1. Mude no arquivo HelloWorldController.cs o método Cumprimentar() conforme ao lado



2. Clique na Pasta HelloWorld e escolha Adicionar -> Novo Item e crie um arquivo com nome Cumprimentar.cshtml



Exibição do Razor

ASP.NET Core

```
@{
    ViewData["Title"] = "Cumprimentar";
}

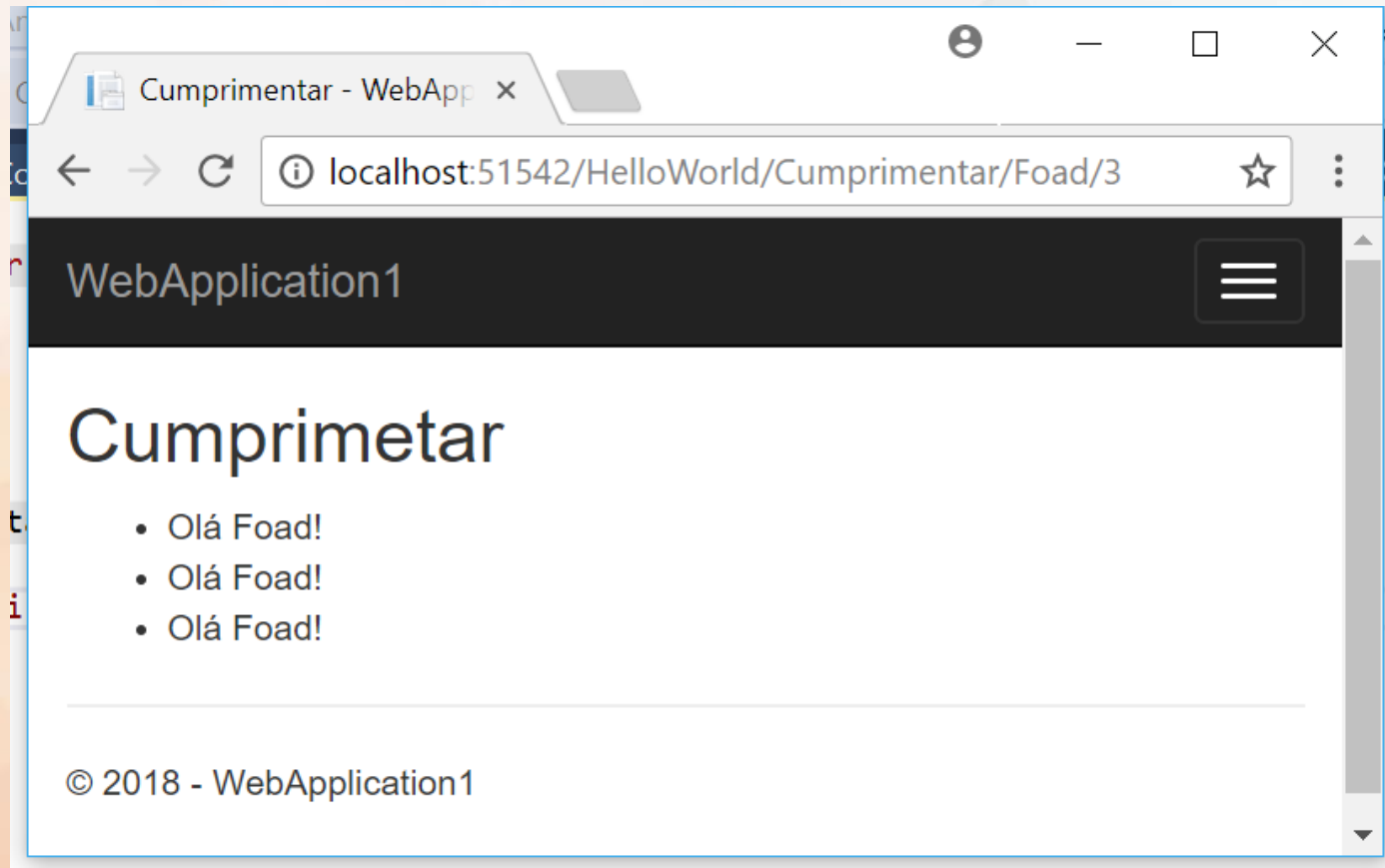
<h2>Cumprimentar</h2>

<ul>
    @for (int i = 0; i < (int)ViewData["Repetir"]; i++)
    {
        <li>@ViewData["Mensagem"]</li>
    }
</ul>
```

3. Altere o arquivo Cumprimentar.cshtml conforme ao lado

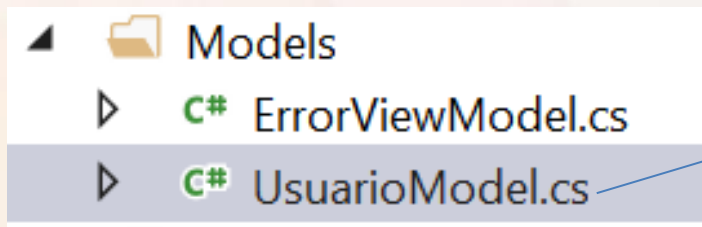
Note que quando usamos um código em C# na View usamos o @

# Resultado



# Criando um Model

## Opção sem Criptografia na senha



1. Criar um nova Classe UsuarioModel na pasta Models. (Classe Simples)

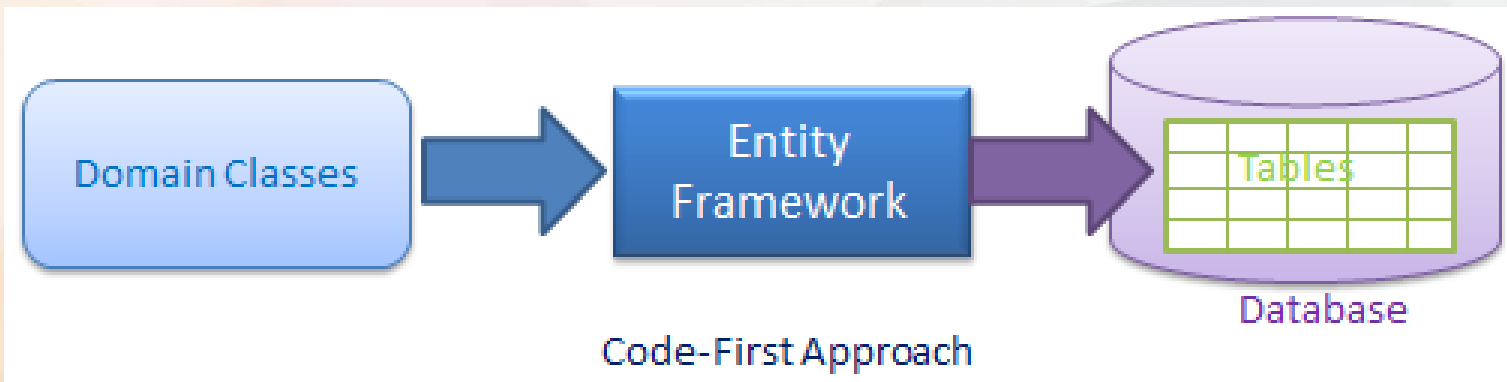
```
namespace WebApplication1.Models
{
    6 referências
    public class UsuarioModel
    {
        12 referências | 0 exceções
        public int ID { get; set; }
        0 referências | 0 exceções
        public string UserName { get; set; }
        0 referências | 0 exceções
        public string Nome { get; set; }
        0 referências | 0 exceções
        public string Senha { get; set; }
        0 referências | 0 exceções
        public string CPF { get; set; }
        0 referências | 0 exceções
        public DateTime Aniversario { get; set; }
    }
}
```

Coloque os parâmetros que você deseja adicionar ao model. Esses parâmetros serão refletidos futuramente no banco de dados. Essa metodologia é chamada Code First.

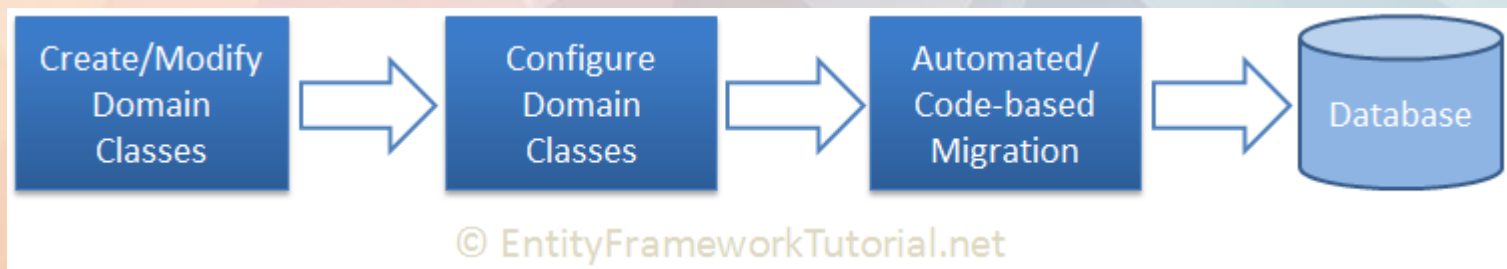
# Code First

Nessa metodologia, o foco do desenvolvedor é o código, sendo que o banco de dados é uma consequência do código.

As classes são mapeadas pelo Entity Framework no banco de dados.

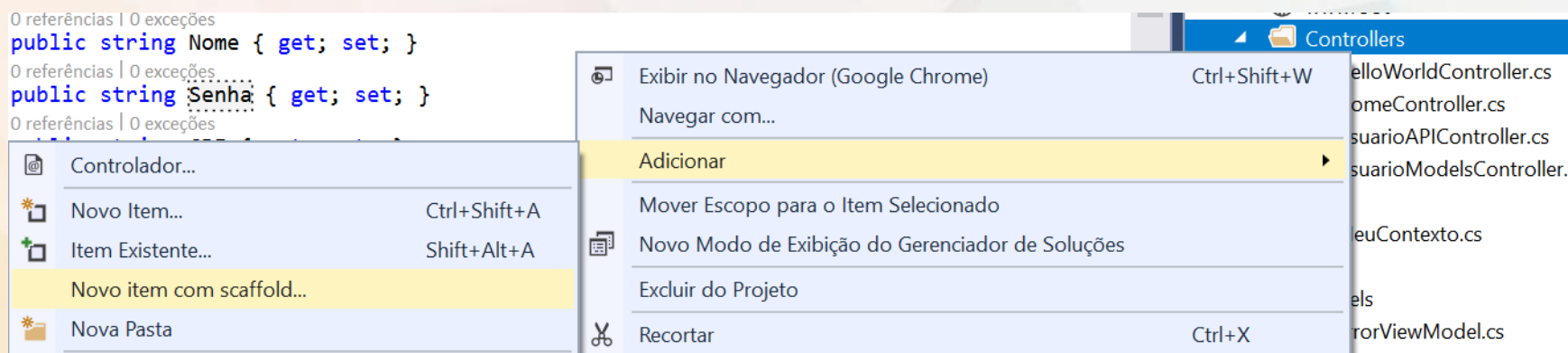


Os arquivos de migração são criados automaticamente a partir do código, e esses arquivos criam/alteram as tabelas do banco de dados



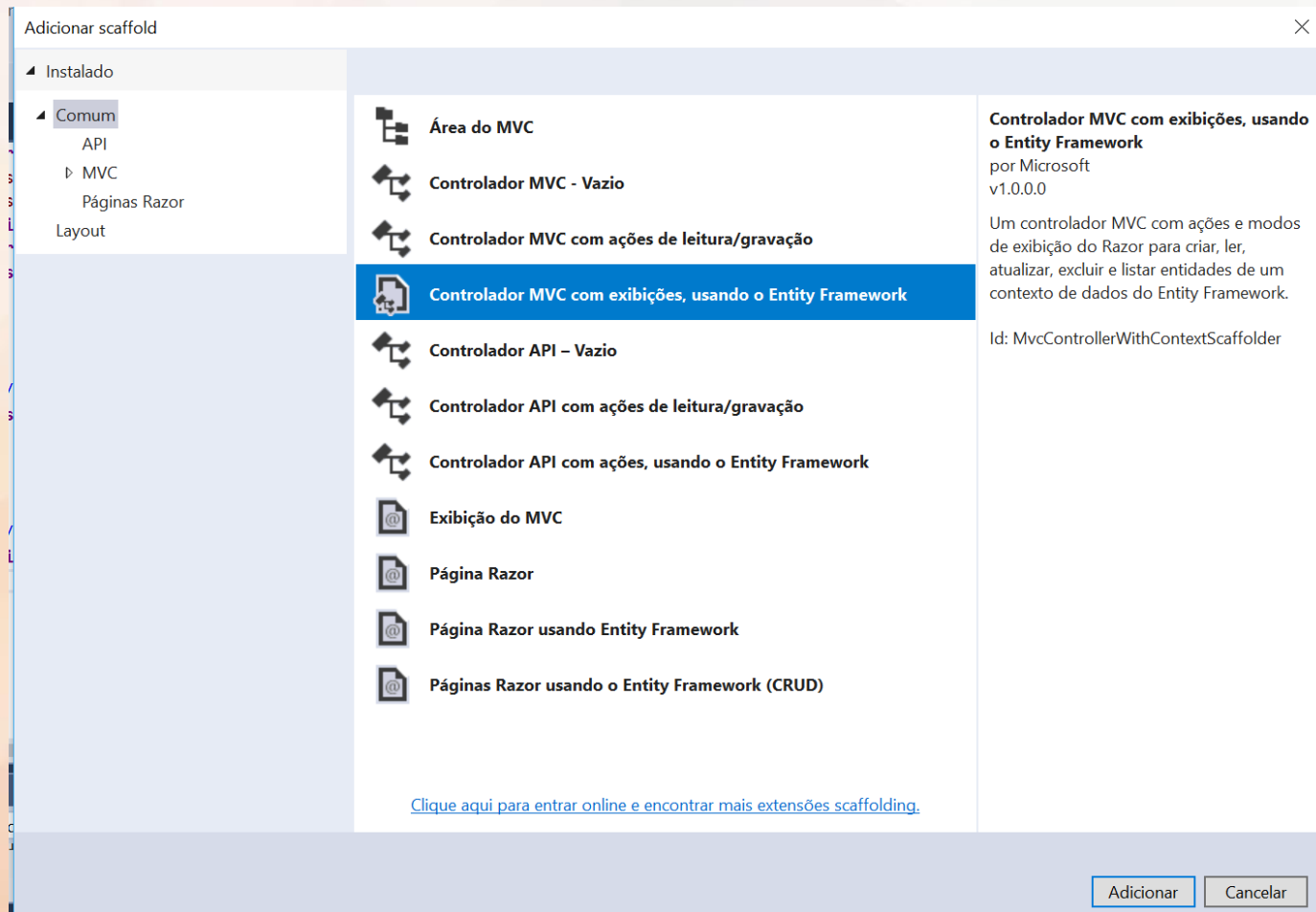
# Criando um CRUD com Scaffold

## 1. Clique na pasta Controllers e escolha Novo Item com scaffold





# Escolha Controlador MVC usando Entity Framework



# Criando um Scaffold de Controlador

Adicionar Controlador MVC com exibições, usando o Entity Framework

Classe do modelo:

Classe de contexto de dados:  +

Modos de exibição:

- ☒ Gerar modos de exibição
- ☒ Bibliotecas de scripts de referência
- ☒ Use uma página de layout:

...

(Deixe em branco se ele estiver definido em um arquivo Razor \_viewstart)

Nome do controlador:

Adicionar Cancelar

Esse é o nome do controlador que será criado, vamos manter o nome.

Crie um Contexto com nome MeuContext

Adicionar Contexto de Dados

Novo tipo de contexto de dados:

Adicionar Cancelar

# Setar o nosso projeto para utilizar o MySQL como banco de dados

No Arquivo Startup.cs altere o texto UseSqlServer para UseMySQL

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddDbContext<WebApplication2Context>(options =>
        options.UseSqlServer(Configuration.GetConnectionString
    )
}
```

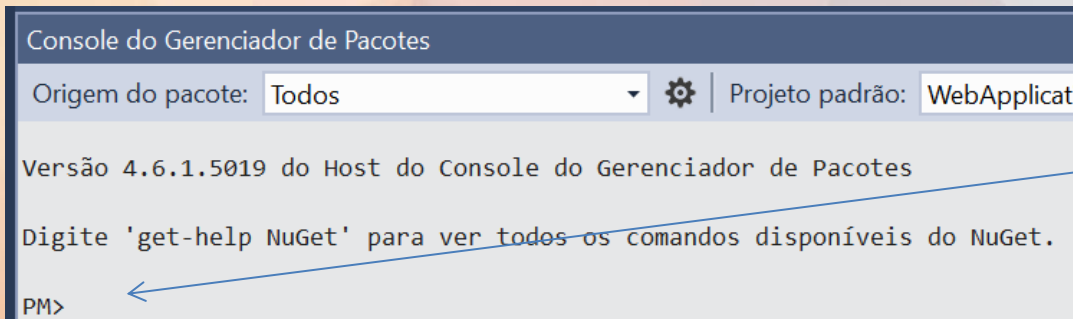
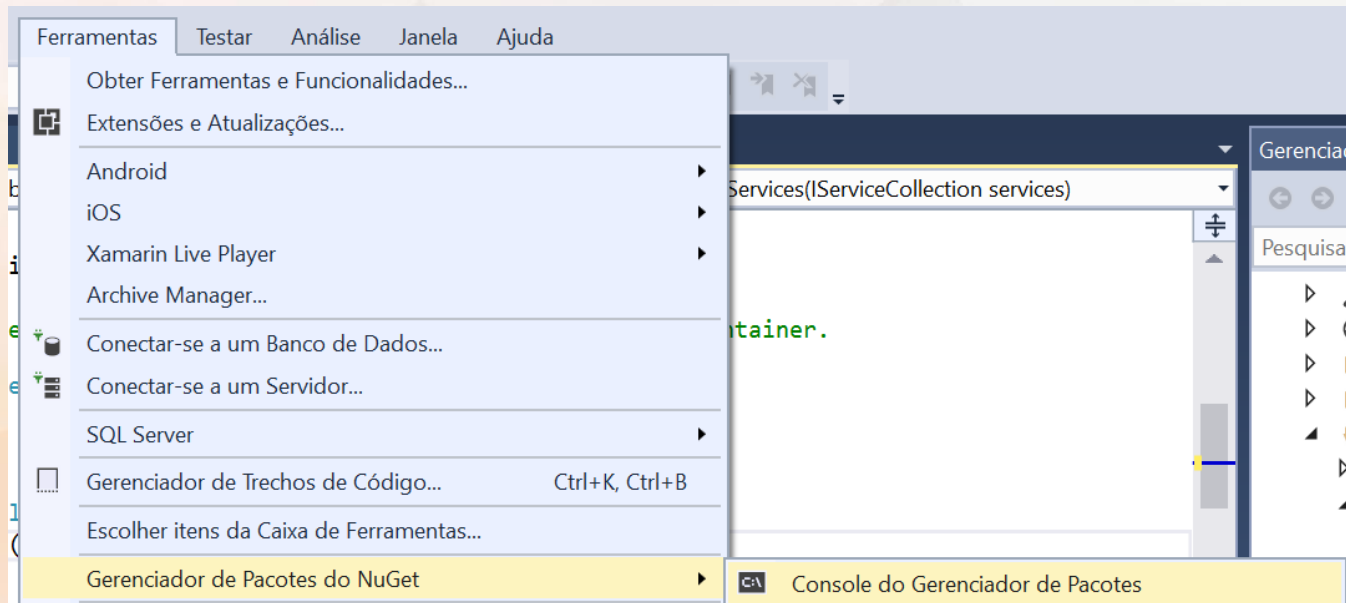
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddDbContext<MeuContexto>(options =>
        options.UseMySQL(Configuration.GetConnectionString
    )
}
```

1. Provavelmente aparecerá um erro, já que ainda não instalamos os pacotes do MySQL
2. Certifique que a grafia do UseMySQL esteja correta com SQL todo em maiusculo.

# Instale os pacotes necessários usando o NuGet

Acesse o console do NuGet



Execute os códigos abaixo na linha de comando do NuGet

```
Install-Package MySql.Data  
Install-Package MySql.Data.Entity  
Install-Package MySql.Data.EntityFrameworkCore  
Install-Package Microsoft.EntityFrameworkCore.Tools
```

# Setar o Connection String

O connection string define os parâmetros da conexão com o banco de dados. Ela é definida no arquivo appsettings.json

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "ConnectionStrings": {
    "MeuContext": "Server=(localdb)\\mssqllocaldb;Database=WebApplication2Conte...;MultipleActiveResultSets=true"
  }
}
```

Altere o connection string para “server=localhost;database=[DATABASE QUE VOCE IRA USAR];userid=[USUARIO, ex root];password=[SUA SENHA]”

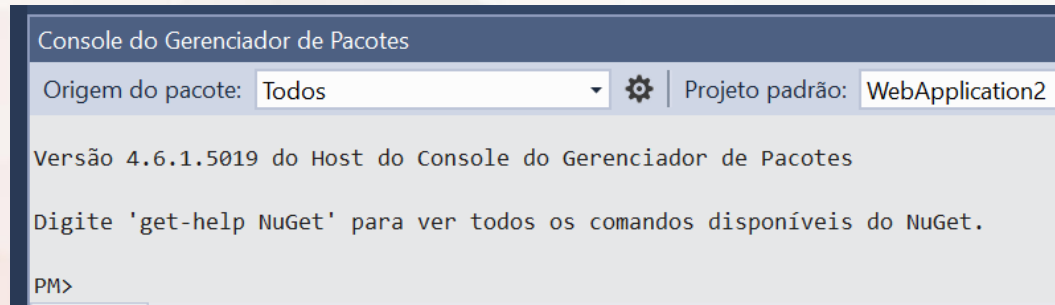
Exemplo de string:

```
"server=localhost;database=crud;userid=root;password=foad"
```

Use sua senha

# Criando o Arquivo de Migração

No console do NuGet ->



Execute os seguintes comandos:   Add-Migration Initial  
  Update-Database

Atenção! Em caso do erro abaixo no comando Update-Database:

```
at Microsoft.EntityFrameworkCore.Design.Operati  
Table 'crud.__efmigrationshistory' doesn't exist  
PM> |
```

Entre no MySQL, selecione o banco de dados que você irá utilizar e crie a tabela \_\_efmigrationshistory:





```
CREATE TABLE __efmigrationshistory (  
MigrationId VARCHAR(150) NOT NULL,  
ProductVersion VARCHAR(32) NOT NULL,  
PRIMARY KEY (MigrationId)  
)
```


# Tabela Criada no Banco

Após o comando Update-Database, uma nova tabela é criada automaticamente no banco de dados:

Nome:	usuariomodel
Comentário:	

Colunas:	 Adicionar	 Remover	 Mover para cima	 Mover para baixo
----------	---	---	---	--

#	Nome	Tipo de dados	Tamanho/Itens	Permitir...	Zerofill	Padrão	Unsign...	Comentário
 1	ID	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREMENT	<input type="checkbox"/>	
2	Aniversario	DATETIME		<input type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão	<input type="checkbox"/>	
3	CPF	TEXT		<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL	<input type="checkbox"/>	
4	Nome	TEXT		<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL	<input type="checkbox"/>	
5	Senha	TEXT		<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL	<input type="checkbox"/>	
6	UserName	TEXT		<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL	<input type="checkbox"/>	

# Testando o CRUD

Execute o servidor, e entre no endereço abaixo. O CRUD agora está 100% operacional.

WebApplication1 Home About Contact

## Index

[Create New](#)

UserName	Nome	Senha	CPF	Aniversario	
foadmk	Foad Mobini Kesheh	ec6a6536ca304edf844d1d248a4f08dc	033.334.232- 23	31/01/1984 00:00:00	<a href="#">Edit</a> <a href="#">Details</a> <a href="#">Delete</a>

© 2018 - WebApplication1

Criar novo

Editar

Ver

Excluir



# Arrumando Data e Textos (Opcional)

No seu model, você pode fazer algumas alterações para deixar o CRUD mais amigável:

```
[Display(Name = "CPF")]  
0 referências | 0 exceções  
public string CPF { get; set; }
```

Nome da variável que  
aparecerá no CRUD

```
[Display(Name = "Aniversário")]  
[DataType(DataType.Date)]  
0 referências | 0 exceções  
public DateTime Aniversario { get; set; }
```

Usar apenas a Data no aniversário

Define o tamanho do campo, na prática se for feita uma nova migration, o campo muda de TEXT para VARCHAR

```
private string senha { get; set; }  
  
[Display(Name = "Senha")]  
[Required(ErrorMessage = "A senha é obrigatória")]  
[StringLength(255, ErrorMessage = "A senha deve ter pelo menos 5 caracteres", MinimumLength = 5)]  
[DataType(DataType.Password)]  
0 referências | 0 exceções  
public string Senha  
{
```

Transformar o campo senha em obrigatório

Opcional, define um número mínimo  
obrigatório de caracteres.

Altera o password para parecer oculto \*\*\*\*

# Criptografar a senha (Opcional)

2 referências | 0 exceções

```
private string senha { get; set; }
```

```
[Display(Name = "Senha")]
```

```
[Required(ErrorMessage = "A senha é obrigatória")]
```

```
[StringLength(255, ErrorMessage = "A senha deve ter pelo menos 5 caracteres", MinimumLength = 5)]
```

```
[DataType(DataType.Password)]
```

0 referências | 0 exceções

```
public string Senha
```

```
{
```

```
    get
```

```
    {
```

```
        return senha;
```

```
    }
```

```
    set
```

```
    {
```

```
        byte[] data = MD5.Create().ComputeHash(Encoding.UTF8.GetBytes(value));
```

```
        StringBuilder sBuilder = new StringBuilder();
```

```
        for (int i = 0; i < data.Length; i++)
```

```
        {
```

```
            sBuilder.Append(data[i].ToString("x2"));
```

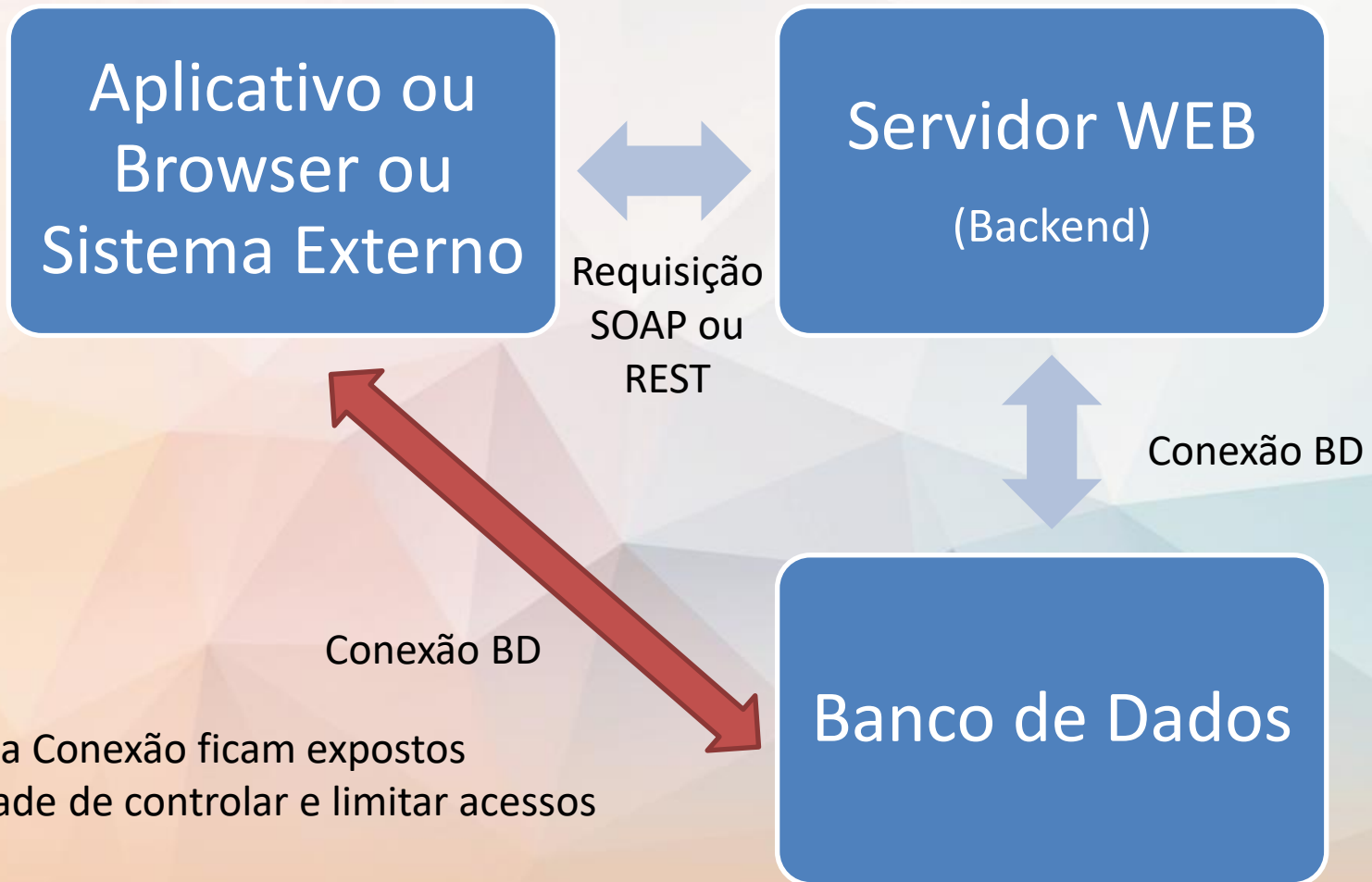
```
        }
```

```
        senha = sBuilder.ToString();
```

```
    }
```

```
}
```

# Diagrama de Serviço WEB



- Dados da Conexão ficam expostos
- Dificuldade de controlar e limitar acessos

# Serviços WEB

- Integração entre diferentes sistemas
- Troca de dados de forma mais segura
- Principais Formatos
  - SOAP / WSDL
    - XML
  - RESTful
    - JSON ou XML

# WSDL

- A Web Services Description Language (**WSDL**) é uma linguagem baseada em XML utilizada para descrever Web Services funcionando como um contrato do serviço. Trata-se de um documento escrito em XML que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis.
- <https://www.w3.org/TR/soap/>

## Example 1: SOAP message containing a SOAP header block and a SOAP body

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

# RESTful

- **RE**presentational **S**tate **T**ransfer
- Resource-based != Action-based -> coisas (REST) vs ações (WSDL)
- Identificado por URI
- Formato mais comum é em JSON, mas pode ser em XML
- Exemplo:
  - Recurso: pessoa (Jão)
  - Serviço: informações de contato (GET)
  - Representação: nome, endereço, telefone
- Verbos HTTP (GET, PUT, POST, DELETE) – Ação
- URIs (endereço do recurso)
- Resposta HTTP (status, body)
- Stateless:
  - Servidor não contém o estado do cliente
  - Mensagens auto descritivas

# XML vs JSON

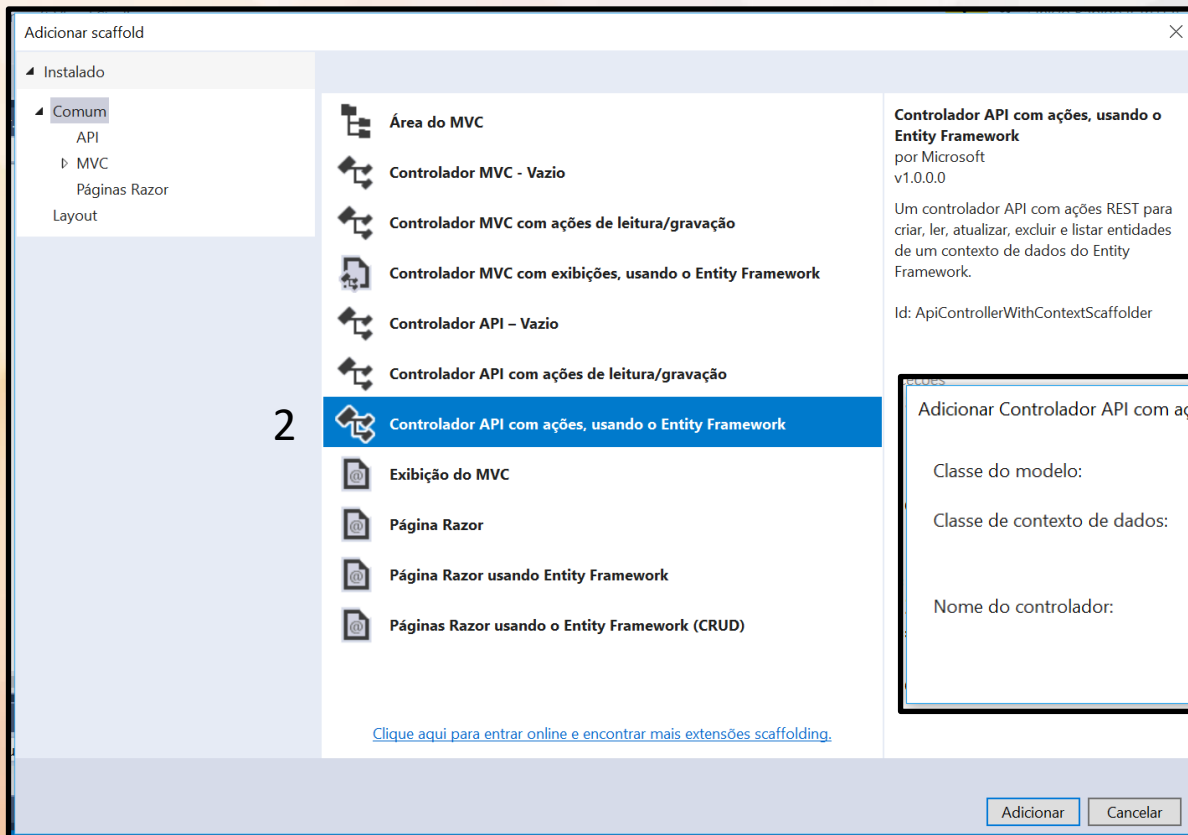
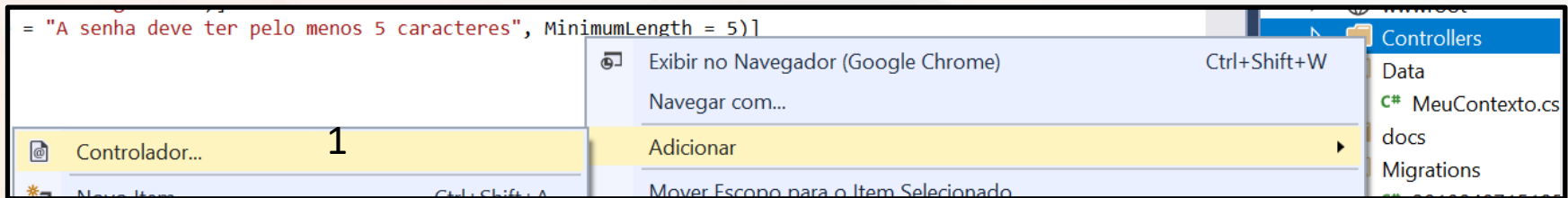
## XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

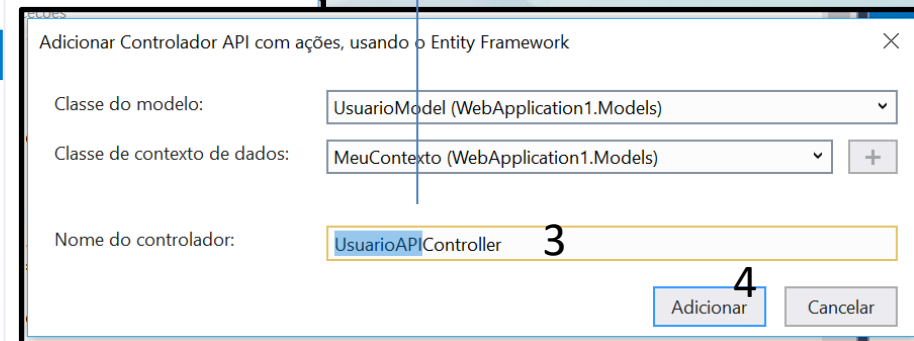
## JSON

```
{ "empinfo" :
  {
    "employees" : [
      {
        "name" : "James Kirk",
        "age" : 40,
      },
      {
        "name" : "Jean-Luc Picard",
        "age" : 45,
      },
      {
        "name" : "Wesley Crusher",
        "age" : 27,
      }
    ]
  }
}
```

# Adicionando um Controlador para API

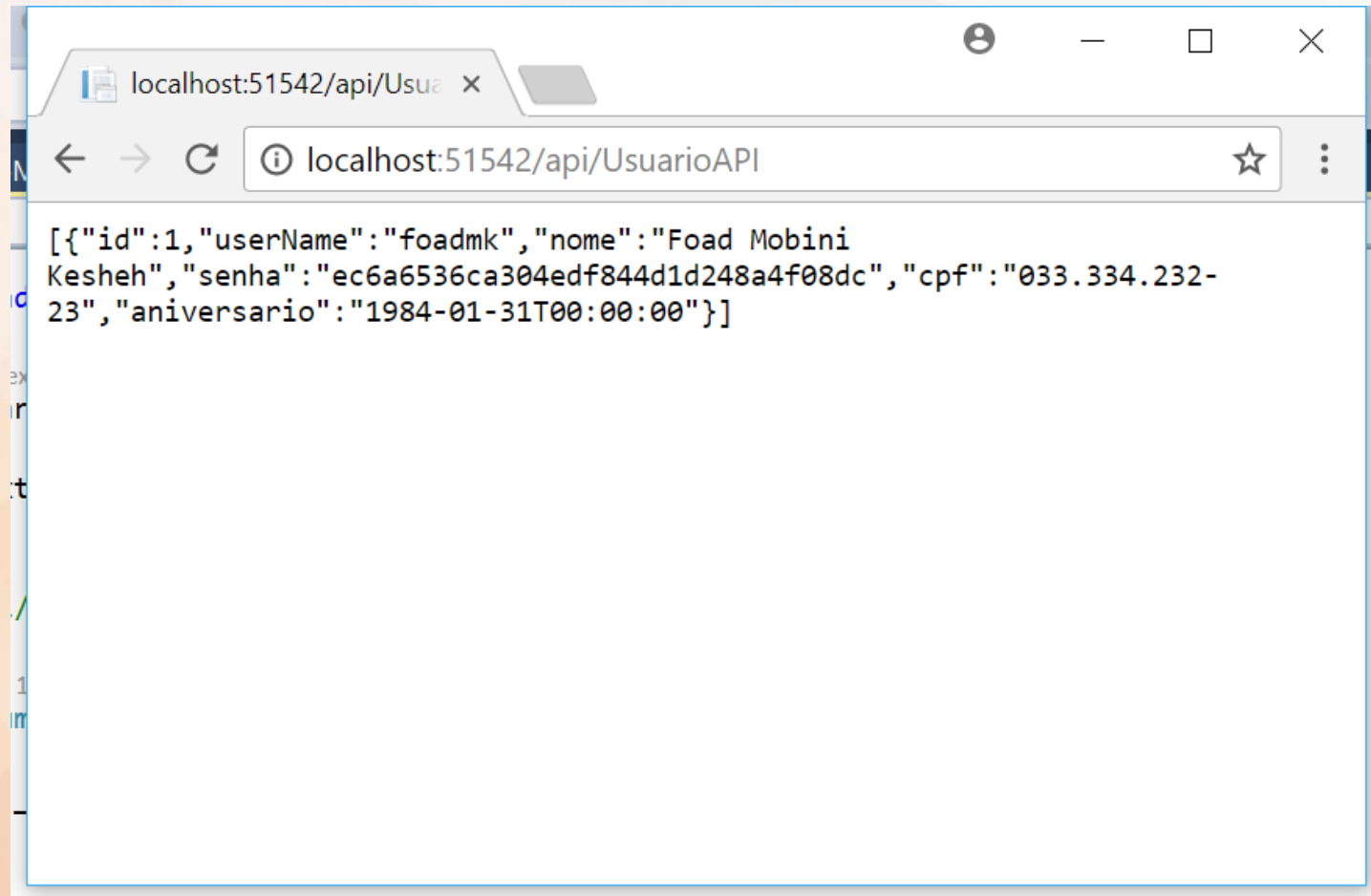


Escolha o nome do controlador:  
UsuarioApiController





# Testando a API



# Referências e Saiba Mais

- <https://docs.microsoft.com/pt-br/aspnet>