

FIA/P GRADUAÇÃO

# ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#03 - ASP.NET CORE - ROTAS E CONTROLLERS

# TRAJETÓRIA

---



Plataforma .NET



Linguagem C# e Orientação a Objetos



ASP.NET Core – Rotas e Controller

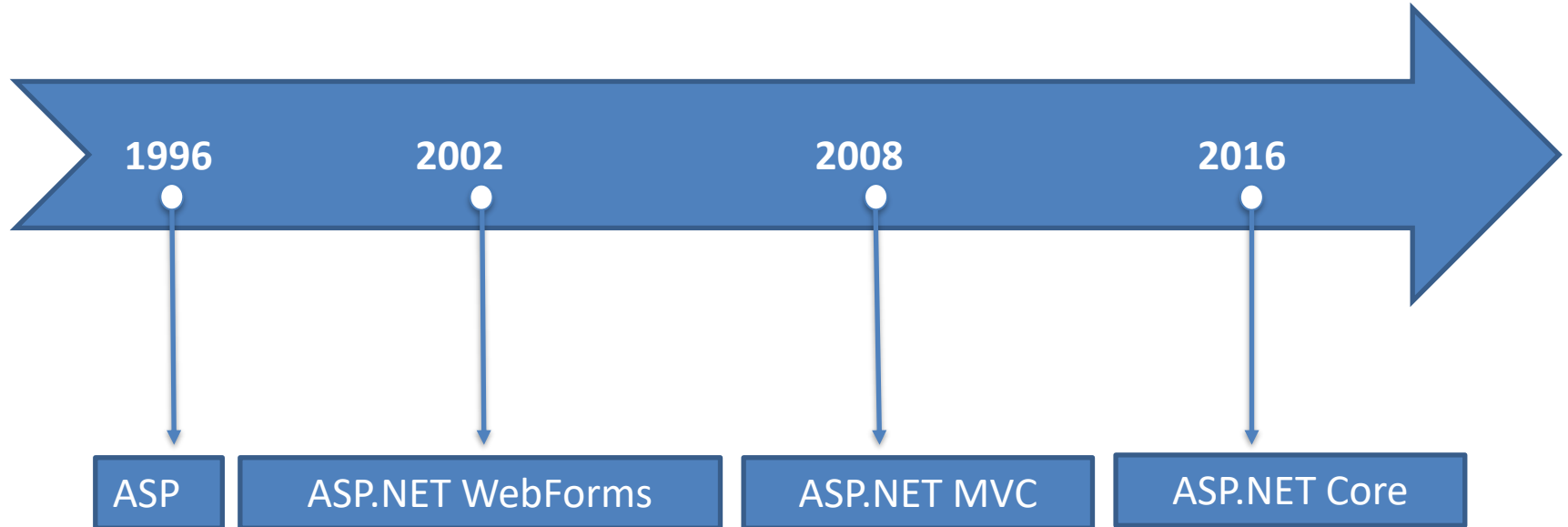
## #03 - AGENDA

---

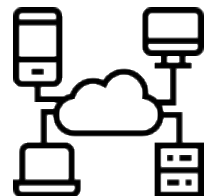


- ASP.NET
- ASP.NET Core MVC
- Rotas e Controller
- Action e tipos de retorno
- Forward x Redirect
- Recebendo parâmetros da view
- Enviando valores para a view

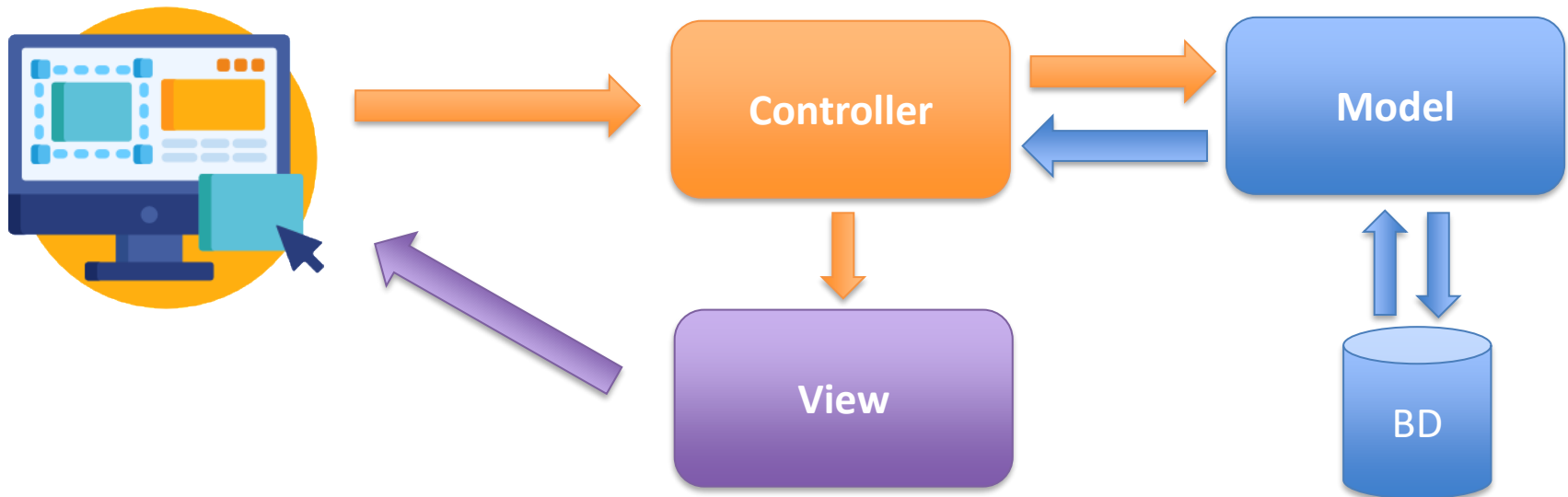
- **ASP.NET** plataforma da Microsoft para o desenvolvimento de aplicações Web e Web Services;



- **ASP.NET Core** é um **framework web** de alta performance, 100% open-source e cross-plataform, desenvolvido pela Microsoft e pela comunidade;
- Desenvolvido para **nuvem** e de forma **modular**;
- Disponível no github:
  - <https://github.com/aspnet/AspNetCore>
- Documentação:
  - <https://docs.microsoft.com/pt-br/aspnet/core>



- Baseado nos padrões **MVC**:
  - **Model**: lógica da aplicação e modelo de dados;
  - **View**: exibe as informações para o usuário (Interface);
  - **Controller**: recebe as requisições do usuário e encaminha o model e/ou view;



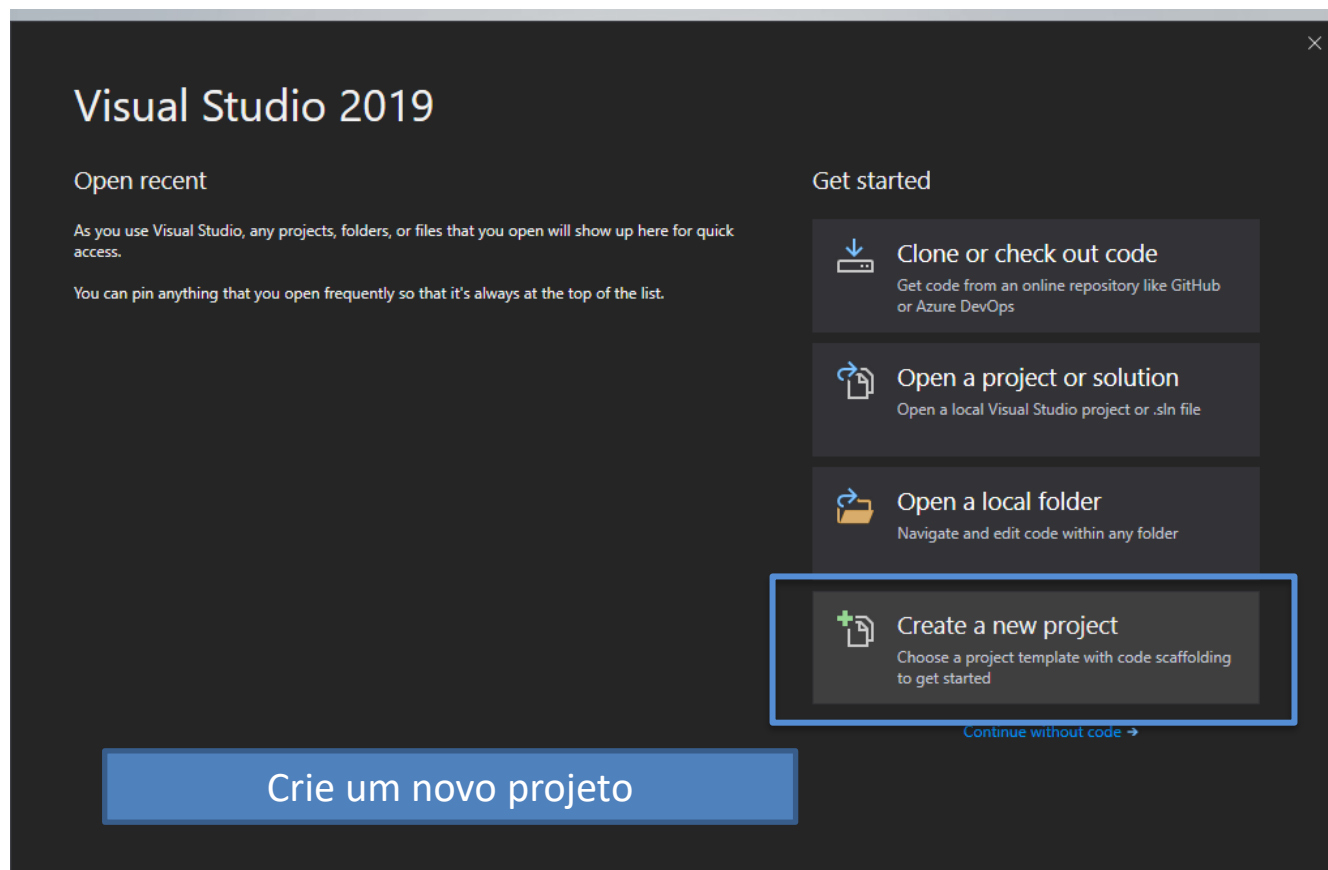
- Para desenvolver a aplicação **ASP.NET Core** vamos utilizar o **Visual Studio** como IDE, **SQL Server** para o banco de dados e o **ISS Express** para servidor:
- O **Visual Studio** possui um banco de dados **SQL Server Local** e o Servidor **ISS Express**, dessa forma, só é necessário instalar o Visual Studio para o desenvolvimento;

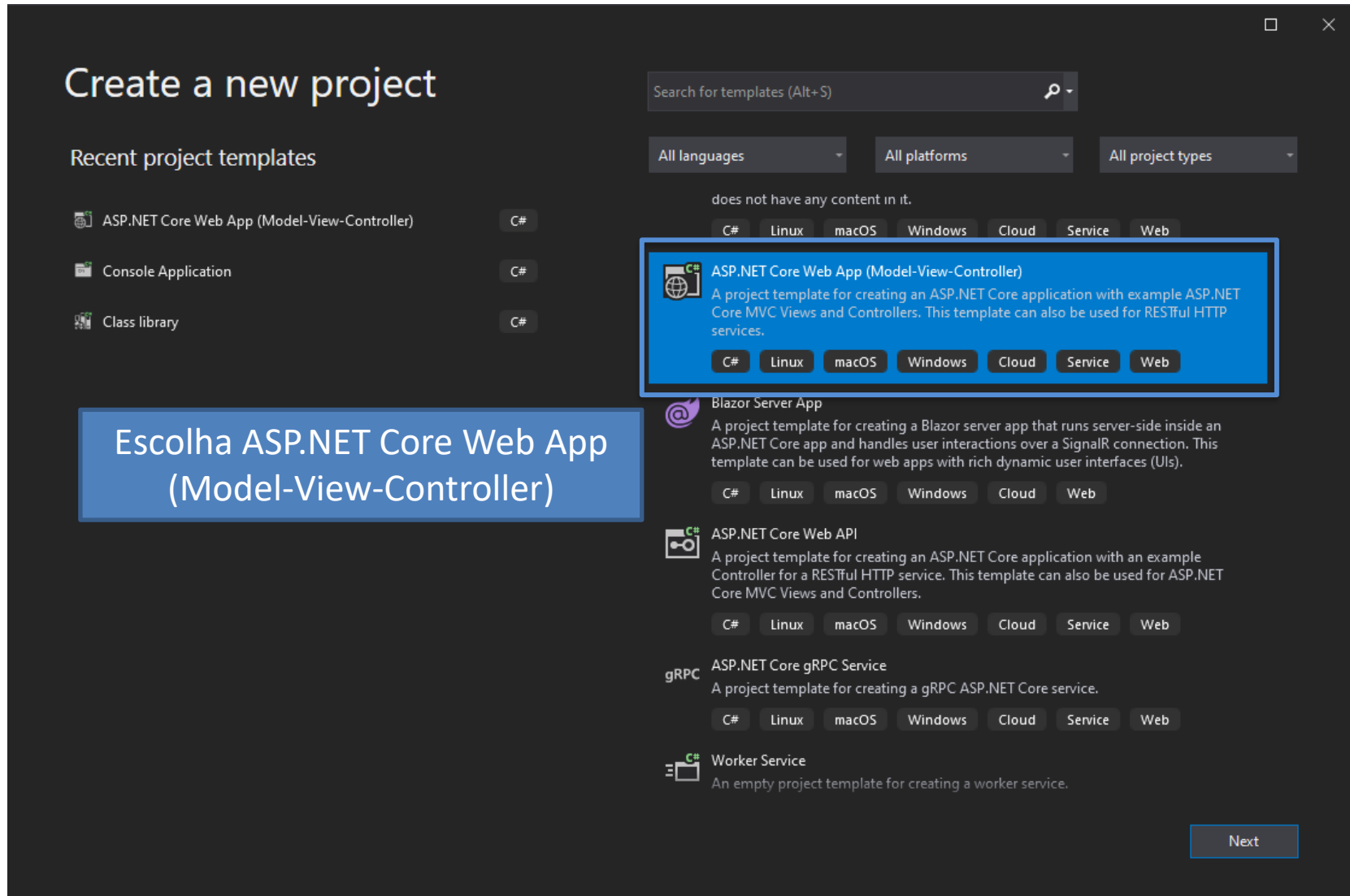






# CRIANDO O PROJETO ASP.NET Core





The screenshot shows the 'Configure your new project' window in Visual Studio. The title is 'Configurar seu novo projeto'. Below the title, it says 'Aplicativo Web do ASP.NET Core (Model-View-Controller)'. There are several tabs: 'C#', 'Linux', 'macOS', 'Windows', 'Nuvem', 'Serviço', and 'Web'. The 'Web' tab is selected. A blue box highlights the configuration fields: 'Nome do projeto' (Project name) with the value 'Fiap.Web.App', 'Local' (Location) with the value 'C:\Users\thiagoyama\source\repos', and 'Nome da solução' (Solution name) with the value 'Fiap.Web.App'. There is also a checkbox labeled 'Colocar a solução e o projeto no mesmo diretório' (Put the solution and the project in the same directory) which is currently unchecked. At the bottom, there is a blue box with the text: 'Configure o nome do projeto e da solução. Escolha o local para salvar os arquivos.' (Configure the project and solution name. Choose the location to save the files.). There are 'Voltar' (Back) and 'Próximo' (Next) buttons at the bottom right.

Configurar seu novo projeto

Aplicativo Web do ASP.NET Core (Model-View-Controller)

C# Linux macOS Windows Nuvem Serviço Web

Nome do projeto

Fiap.Web.App

Local

C:\Users\thiagoyama\source\repos

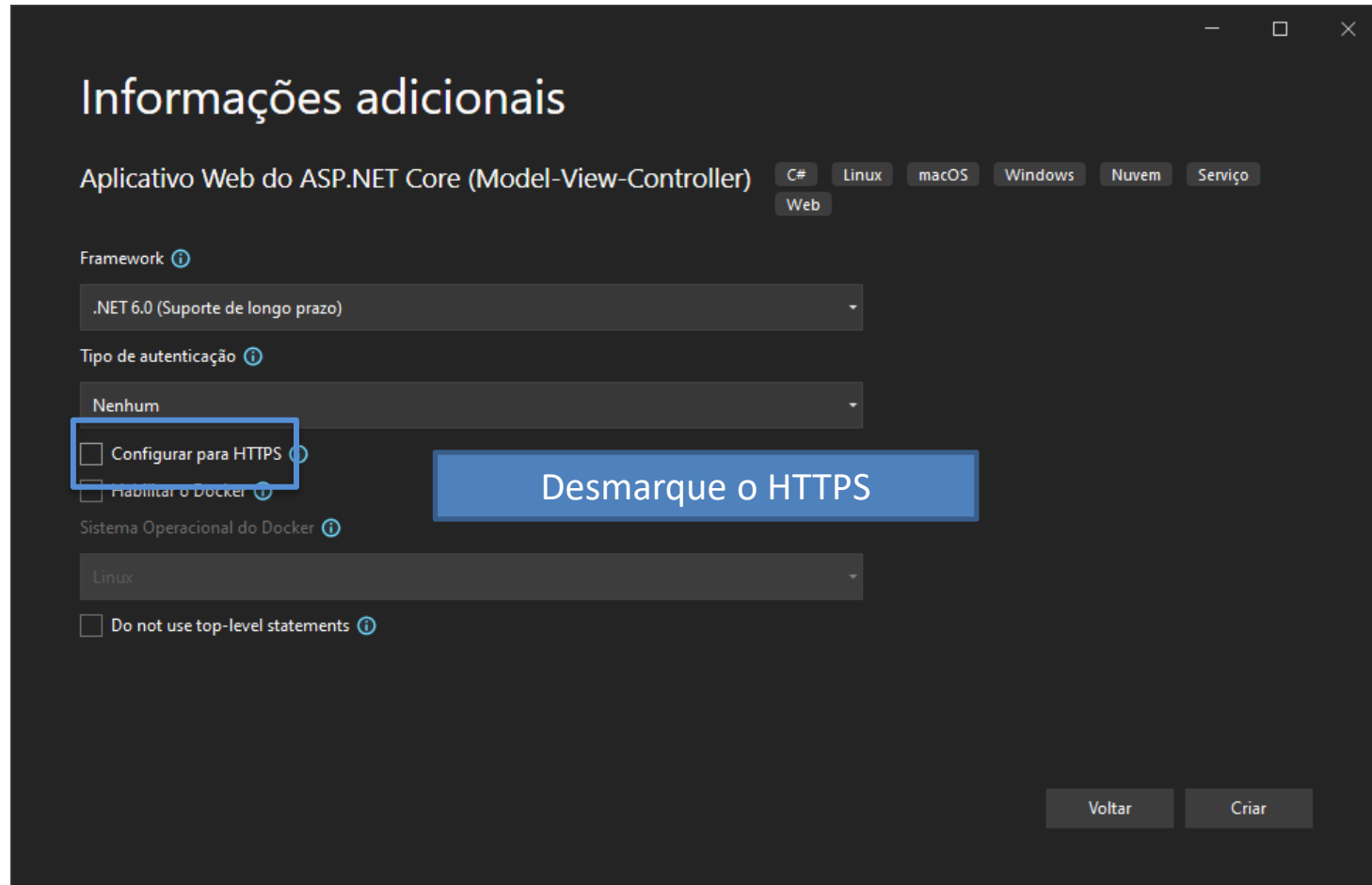
Nome da solução ⓘ

Fiap.Web.App

☐ Colocar a solução e o projeto no mesmo diretório

Configure o nome do projeto e da solução. Escolha o local para salvar os arquivos.

Voltar Próximo



Informações adicionais

Aplicativo Web do ASP.NET Core (Model-View-Controller)

C# Linux macOS Windows Nuvem Serviço Web

Framework ⓘ

.NET 6.0 (Suporte de longo prazo)

Tipo de autenticação ⓘ

Nenhum

☐ Configurar para HTTPS ⓘ

☐ Habilitar o Docker ⓘ

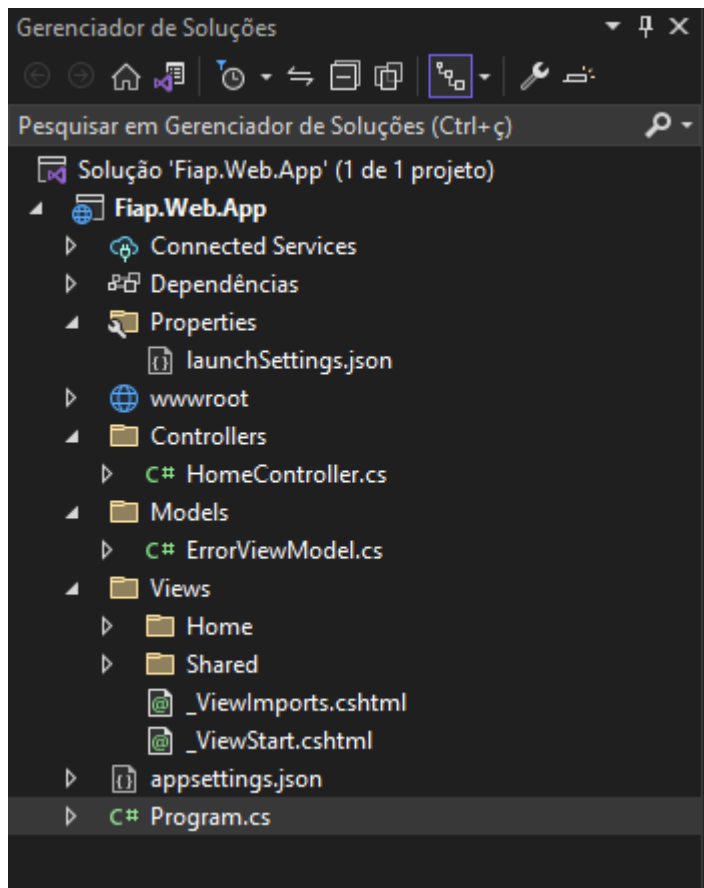
Sistema Operacional do Docker ⓘ

Linux

☐ Do not use top-level statements ⓘ

Desmarque o HTTPS

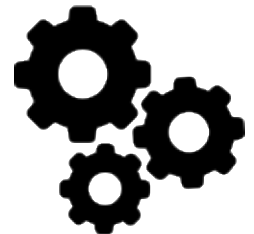
Voltar Criar



- **wwwroot** – diretório para arquivos estáticos, como HTML, CSS, Javascript, imagens e etc;
- Diretórios **Controllers**, **Models** e **Views** para a arquitetura MVC;
- **appsettings.json** – arquivo de configuração do projeto, como o banco de dados, por exemplo;
- Classes **Program.cs** é utilizado para configurar o comportamento da aplicação ASP.NET.

- A classe **Program.cs** é responsável por inicializar a aplicação, configurar os serviços que serão utilizados na aplicação e configura o pipeline de solicitações do ASP.NET Core, um conjunto de middleware para manipular solicitações HTTP;

```
Program.cs
1  var builder = WebApplication.CreateBuilder(args);
2
3  // Add services to the container.
4  builder.Services.AddControllersWithViews();
5
6  var app = builder.Build();
7
8  // Configure the HTTP request pipeline.
9  if (!app.Environment.IsDevelopment())
10 {
11     app.UseExceptionHandler("/Home/Error");
12 }
13 app.UseStaticFiles();
14
15 app.UseRouting();
16
17 app.UseAuthorization();
18
19 app.MapControllerRoute(
20     name: "default",
21     pattern: "{controller=Home}/{action=Index}/{id?}");
22
23 app.Run();
24
```

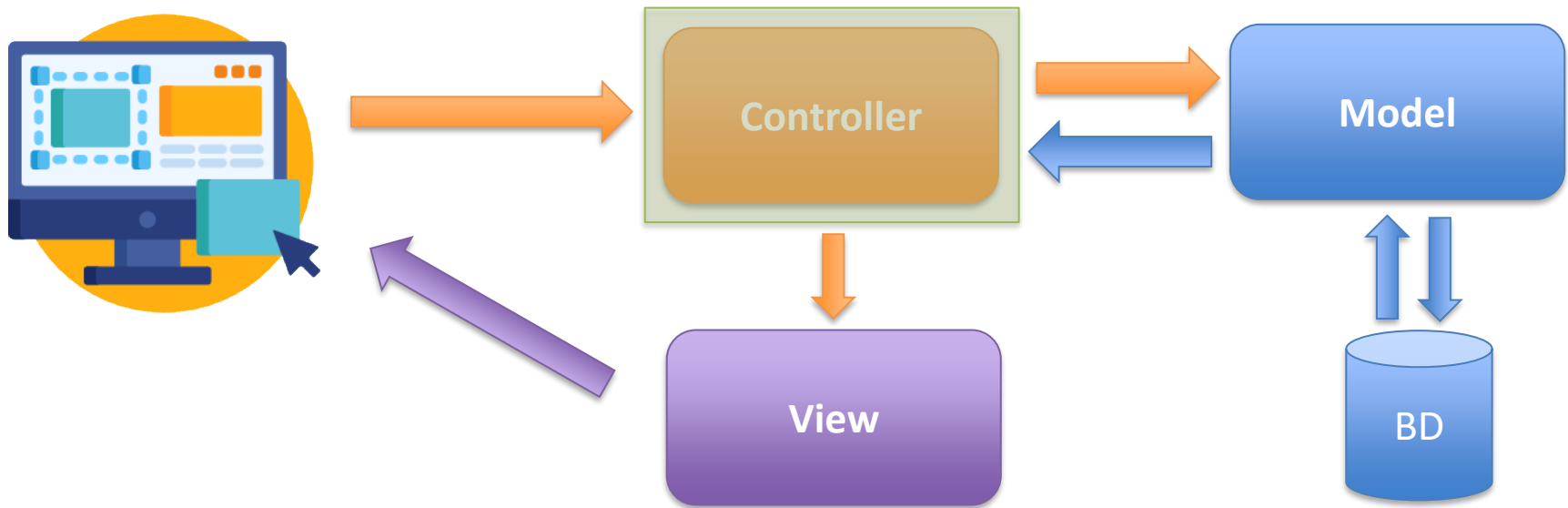




# ROTAS E CONTROLLER



- Responsabilidades básicas do **controller**:
  - » Recuperar dados enviados pelo usuário;
  - » Interagir com a camada model;
  - » Acionar a camada de apresentação para enviar a resposta ao usuário;



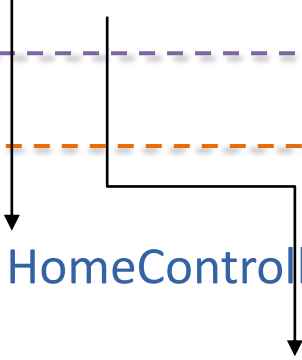
- Para implementar um **controller**:
  - » Nome da classe deve ter o sufixo “*Controller*”;
  - » Pode estender da classe *Microsoft.AspNetCore.Mvc.Controller*;
- As **URLs** são mapeadas para **métodos** (actions) da classe controller;
- A aplicação **ASP.NET Core** possui uma **configuração padrão** de rotas:
  - Após o protocolo, host e porta a primeira informação é o **nome da classe controller** e a segunda é o nome da **action**, ou seja, o nome do método desta classe;
  - É possível configurar **outras rotas** na aplicação;

<http://localhost:40392/home/index>

- Na configuração padrão de rotas, a URL `/home/index` envia a requisição para o método `Index()` da classe `HomeController`:

`http://localhost:40392/home/index`

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        //Código...
    }
}
```



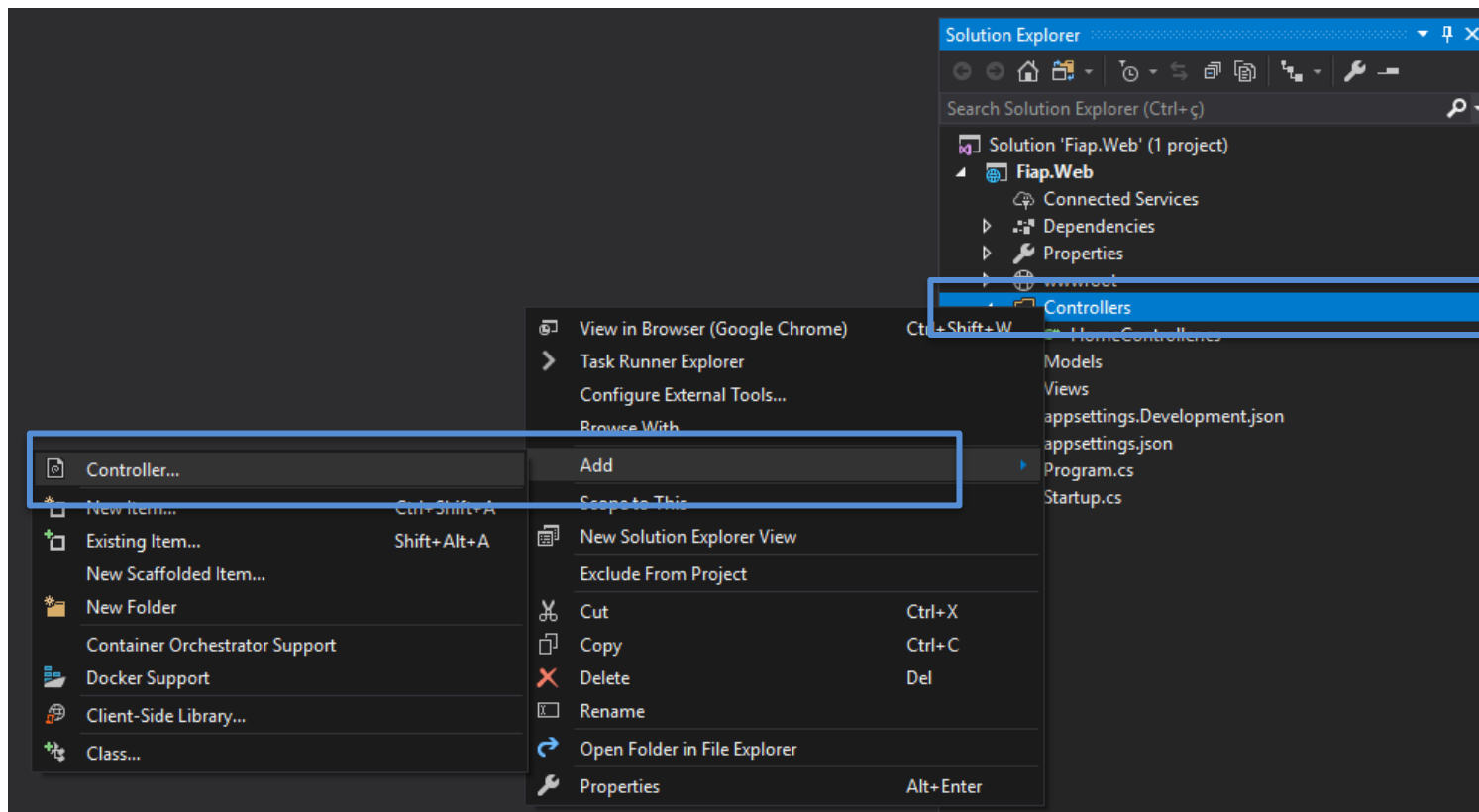
- As **configurações** ficam dentro da classe **Program.cs**, inclusive a configuração padrão de rotas.
- Podemos adicionar mais configurações de rotas;
- **MapControllerRoute**:
  - **name**: nome da rota;
  - **pattern**: definição da url;

Valores padrões

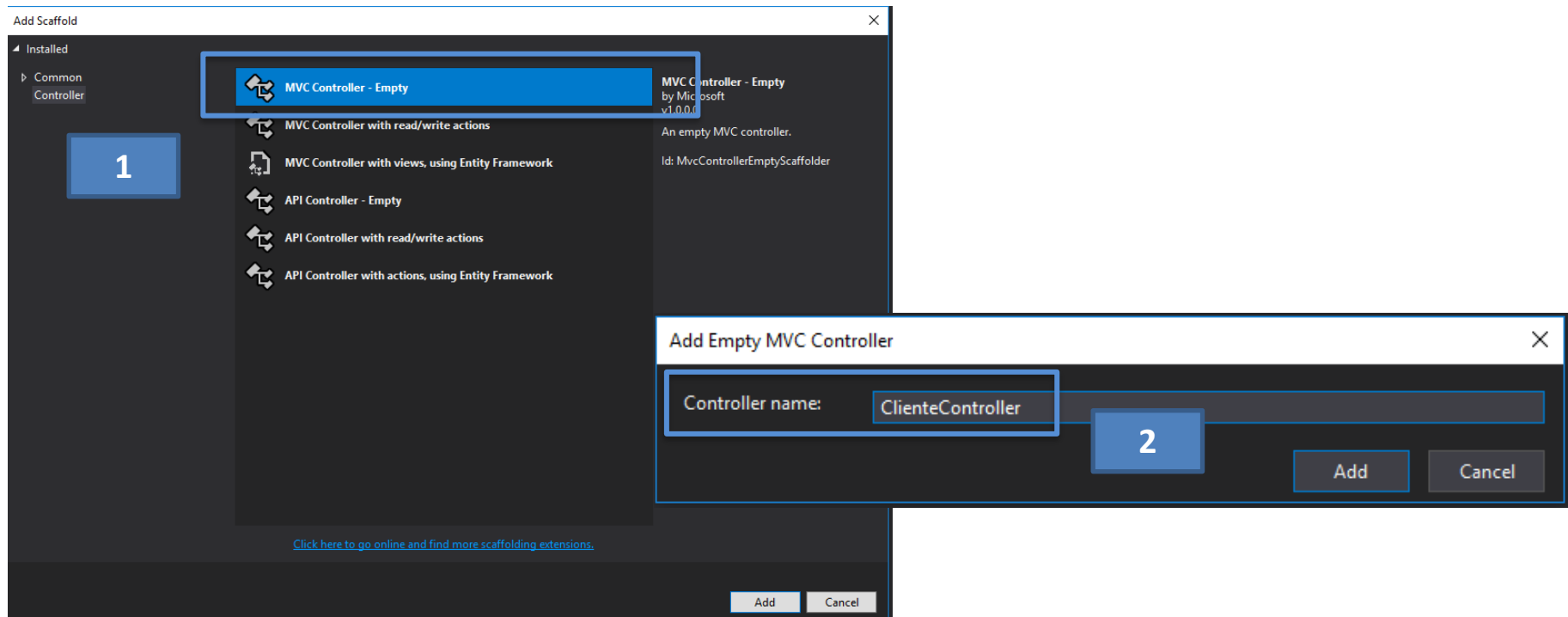
Parâmetro na URL  
opcional (?)

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

- Clique com o botão direito do mouse na pasta **Controllers** e escolha: **Add > Controller..**



- Escolha o template MVC Controller - Empty;
- Defina o nome da classe, lembre-se que deve **terminar** com a palavra **Controller**;



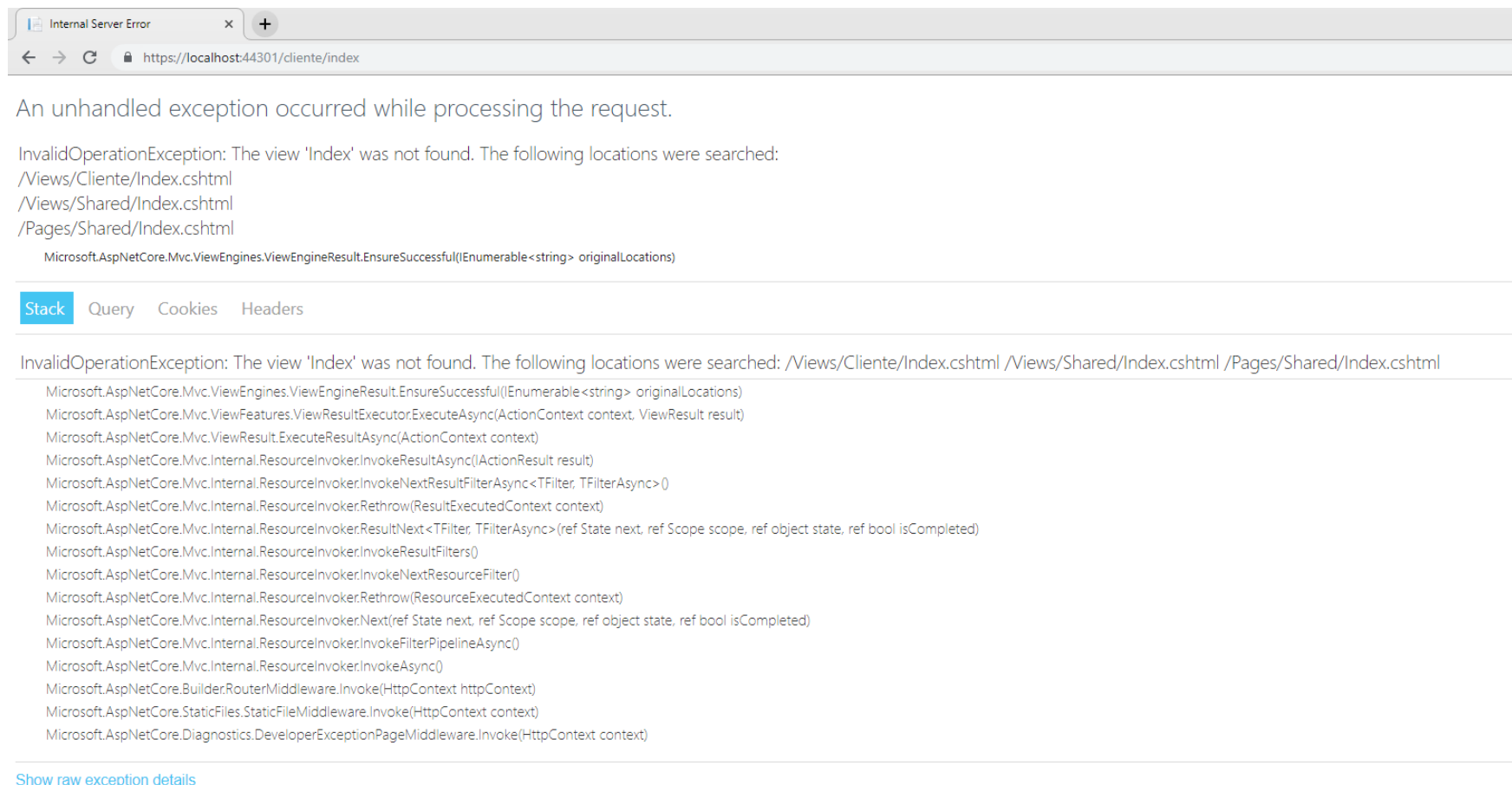
- O usuário interage com a página do sistema para enviar uma **requisição** para um **método da classe controller**, este executa um processamento e depois deve retornar um objeto do tipo **ActionResult** indicando o que deve ser executado depois da ação;

```
public class ClienteController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

- O método invoca o método **View()** para criar um objeto do tipo **ViewResult**, dessa forma o **ASP.NET Core MVC** deve retornar uma **view** para o usuário;
- Por padrão, o framework irá retornar uma **view (página)** com um **nome específico** que está em um diretório dentro de views;
  - Cada **controller** possui um diretório dentro de **Views** com o mesmo nome da classe;
  - O arquivo da **página** tem o mesmo **nome do método** (action);
  - Existe um **diretório** que as páginas podem ser **compartilhadas** entre todos os controllers (Shared);



- Se o servidor não encontrar a **view** correspondente, um erro 500 será exibido no browser:



Internal Server Error

https://localhost:44301/cliente/index

An unhandled exception occurred while processing the request.

InvalidOperationException: The view 'Index' was not found. The following locations were searched:  
/Views/Cliente/Index.cshtml  
/Views/Shared/Index.cshtml  
/Pages/Shared/Index.cshtml

Microsoft.AspNetCore.Mvc.ViewEngines.ViewEngineResult.EnsureSuccessful(IEnumerable<string> originalLocations)

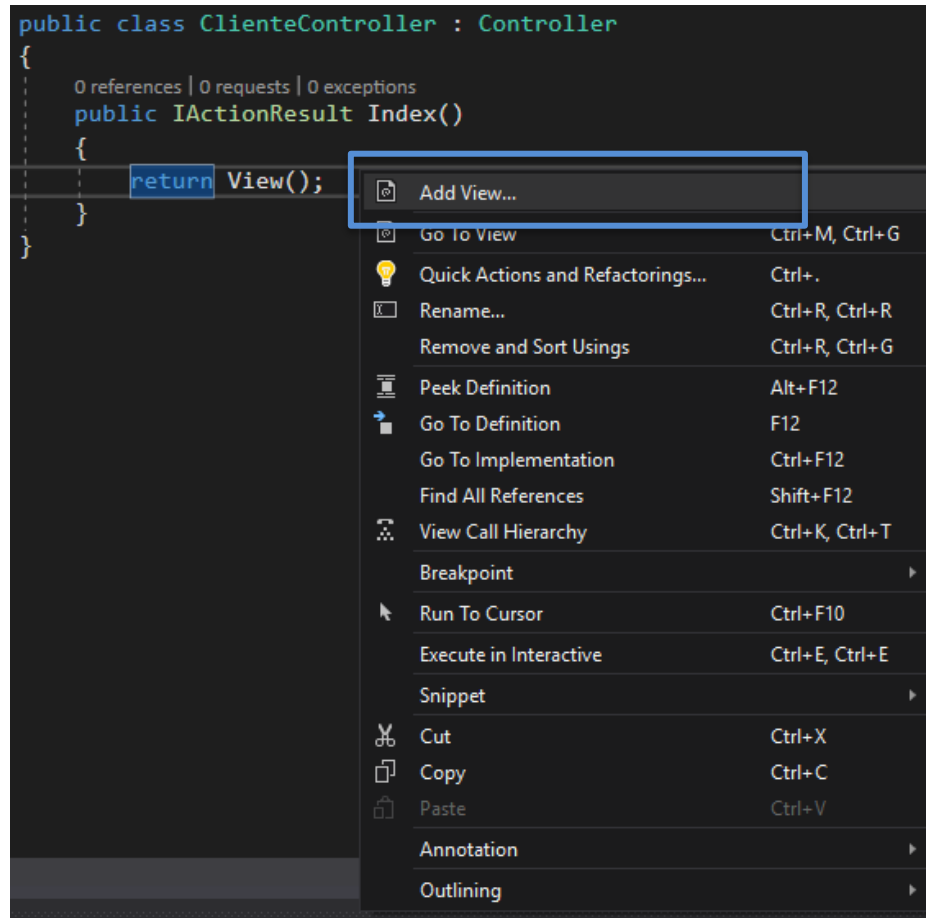
Stack Query Cookies Headers

InvalidOperationException: The view 'Index' was not found. The following locations were searched: /Views/Cliente/Index.cshtml /Views/Shared/Index.cshtml /Pages/Shared/Index.cshtml

Microsoft.AspNetCore.Mvc.ViewEngines.ViewEngineResult.EnsureSuccessful(IEnumerable<string> originalLocations)  
Microsoft.AspNetCore.Mvc.ViewFeatures.ViewResultExecutor.ExecuteAsync(ActionContext context, ViewResult result)  
Microsoft.AspNetCore.Mvc.ViewResult.ExecuteResultAsync(ActionContext context)  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.InvokeResultAsync(IActionResult result)  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.InvokeNextResultFilterAsync<TFilter, TFilterAsync>()  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.Rethrow(ResultExecutedContext context)  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.ResultNext<TFilter, TFilterAsync>(ref State next, ref Scope scope, ref object state, ref bool isCompleted)  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.InvokeResultFilters()  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.InvokeNextResourceFilter()  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.Rethrow(ResourceExecutedContext context)  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.Next(ref State next, ref Scope scope, ref object state, ref bool isCompleted)  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.InvokeFilterPipelineAsync()  
Microsoft.AspNetCore.Mvc.Internal.ResourceInvoker.InvokeAsync()  
Microsoft.AspNetCore.Builder.RouterMiddleware.Invoke(HttpContext httpContext)  
Microsoft.AspNetCore.StaticFiles.StaticFileMiddleware.Invoke(HttpContext context)  
Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware.Invoke(HttpContext context)

[Show raw exception details](#)

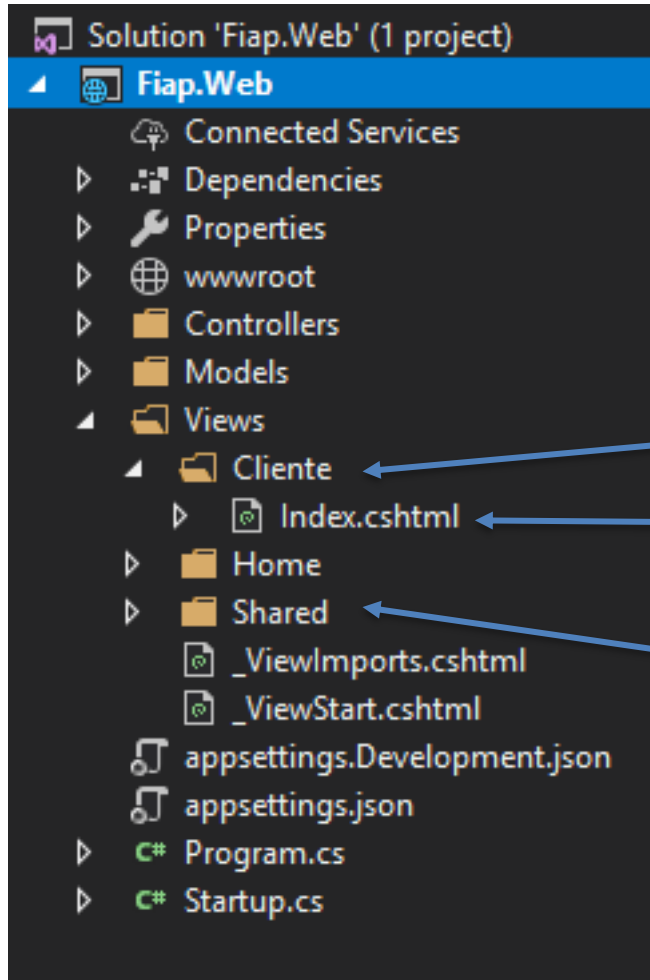
- Para adicionar a view de uma Ação (Método), clique com o botão direito dentro do método e escolha Add View...



- O nome da **View** já vem preenchida de acordo com o nome da action;
- Clique em **Add**;

The screenshot shows the 'Add MVC View' dialog box. The 'View name' field contains the text 'Index'. The 'Template' dropdown menu is set to 'Empty (without model)'. The 'Model class' dropdown menu is empty. Under the 'Options' section, there are three checkboxes: 'Create as a partial view' (unchecked), 'Reference script libraries' (unchecked), and 'Use a layout page:' (checked). Below the 'Use a layout page:' checkbox, there is a text input field and a button with three dots. At the bottom of the dialog, there is a note: '(Leave empty if it is set in a Razor \_viewstart file)'. The 'Add' button is highlighted with a red rectangle, and the 'Cancel' button is also visible.

# RETORNO DO MÉTODO - VIEWS



Diretório para as views do  
**ClienteController**

Página com o **mesmo nome** do  
**método** do controller

Diretório para as views compartilhadas  
entre os controllers

- Nem sempre precisamos devolver uma **View** (Página) como resposta para o usuário..
- É possível redirecionar o usuário para uma URL específica, direciona-lo para uma outra **Action**, etc..



Action Result	Helper Method	Description
<a href="#">ViewResult</a>	<a href="#">View</a>	Renderiza a view para usuário
<a href="#">PartialViewResult</a>	<a href="#">PartialView</a>	Determina o arquivo que deve ser utilizado para construir a página parcial de resposta
<a href="#">RedirectToResult</a>	<a href="#">Redirect</a>	Redireciona para uma URL específica;
<a href="#">RedirectToRouteResult</a>	<a href="#">RedirectToAction</a> <a href="#">RedirectToRoute</a>	Redireciona para outra ação da camada de controle
<a href="#">ContentResult</a>	<a href="#">Content</a>	Devolve um tipo definido pelo desenvolvedor, por exemplo texto
<a href="#">JsonResult</a>	<a href="#">Json</a>	Devolve um objeto no formato JSON.
<a href="#">FilePathResult</a>	<a href="#">File</a>	Devolve valores binários para escrever no response
<a href="#">EmptyResult</a>	(None)	Devolve uma resposta vazia

```
public IActionResult Index()  
{  
    return Content("Olá Mundo");  
}
```

Retorna o texto: **Olá Mundo**

```
public IActionResult Index()  
{  
    return View("Home");  
}
```

Retorna a página **Home**

```
public IActionResult Index()  
{  
    return RedirectToAction("Listar");  
}
```

Redireciona para a action **Listar**

- Um **redirect** é uma nova requisição que o cliente (browser) faz a pedido da aplicação web, logo ele fica ciente sobre como está ocorrendo a navegação e para onde ele está sendo redirecionado;
- Um **forward** pode executar várias requisições no lado servidor sem o conhecimento do cliente;
- Um **forward** mantém os atributos e parâmetros do request original, já um redirect não;

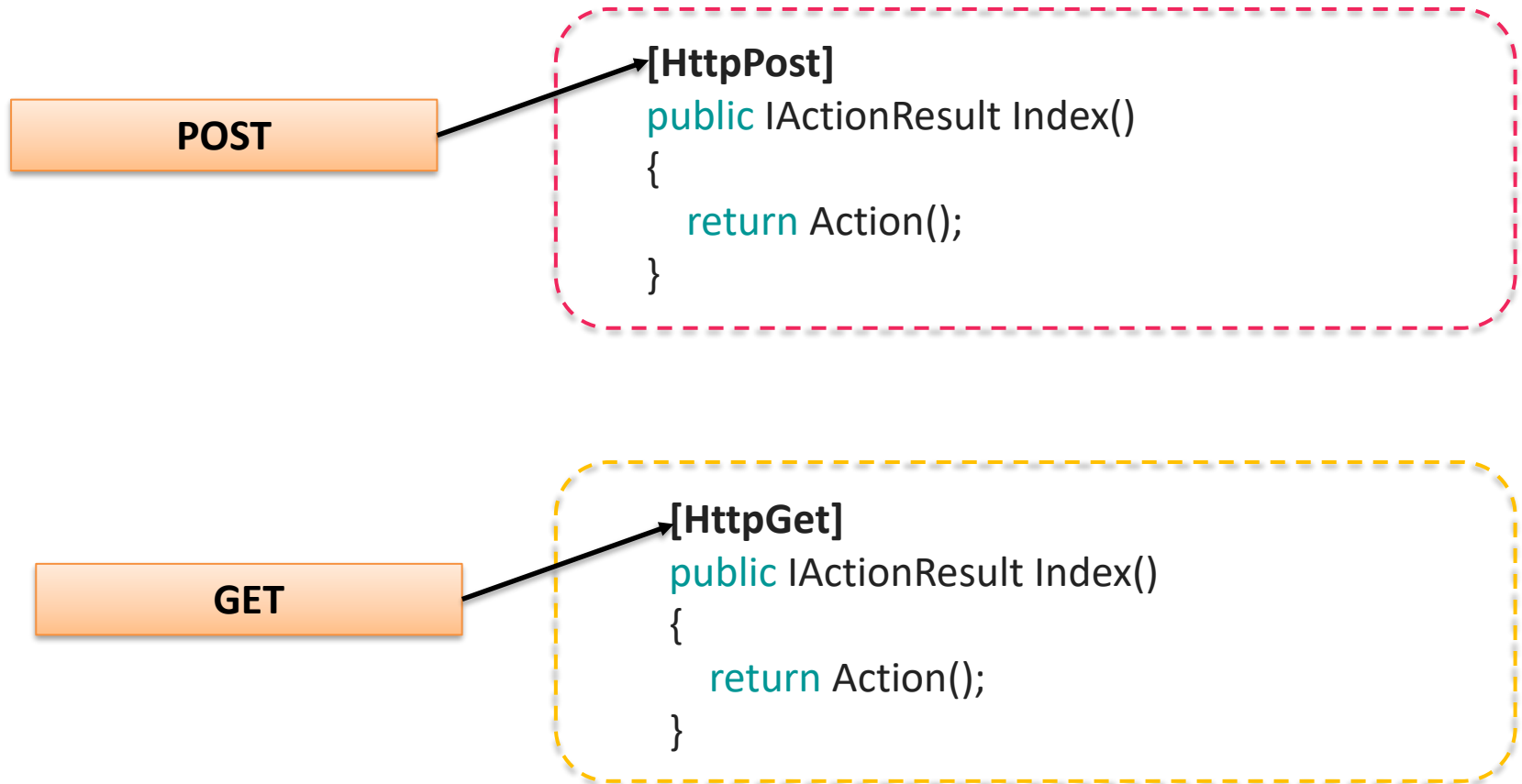
Com **forward**, após um cadastro, caso o usuário atualize a página (F5) a requisição original será processada novamente, ou seja, o cadastro será realizado novamente!

Então, sempre após um **POST** devemos realizar um **redirect**!





- Podemos definir qual **tipo de requisição** a action irá processar:





# RECEBENDO PARÂMETROS

- Quando o usuário **clica em um link** ou **envia um formulário** uma **requisição** é enviada para o servidor com **parâmetros** (valores);
- O primeiro passo para desenvolver um sistema web é compreender como **recuperar esses valores no Controller**;



- O formulário envia para a URL “Home/Cadastrar” (action) dois parâmetros para o Controller: nome e sobrenome;

## Formulário HTML

```
<form action = "/Home/Cadastrar" >  
  <label>Nome</label>  
  <input type="text" name="nome" />  
  
  <label>Sobrenome </label>  
  <input type="text" name="sobrenome" />  
  
  <input type = "submit" value="Enviar" />  
</form>
```

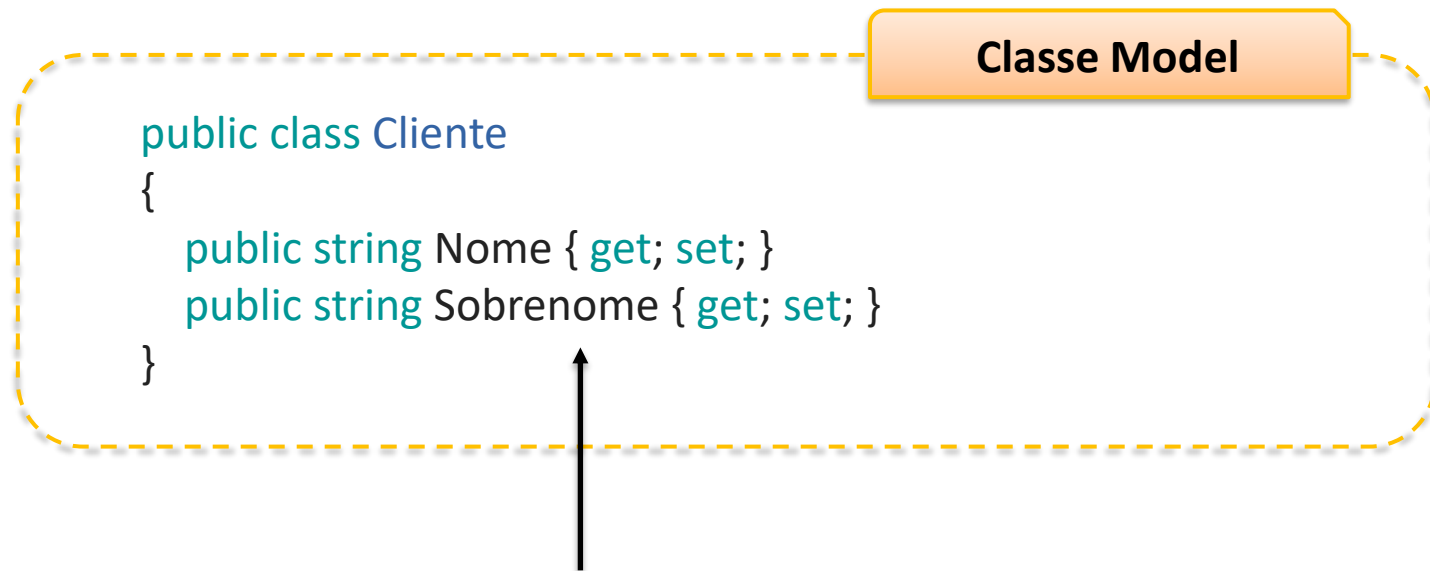
- Podemos receber os valores como **parâmetro do método**;
- Basta definir o **nome dos parâmetros** do método com os **mesmos nomes** dos campos do formulário HTML;

Controller

```
public IActionResult Cadastrar(string nome, string sobrenome)
{
    return View();
}
```

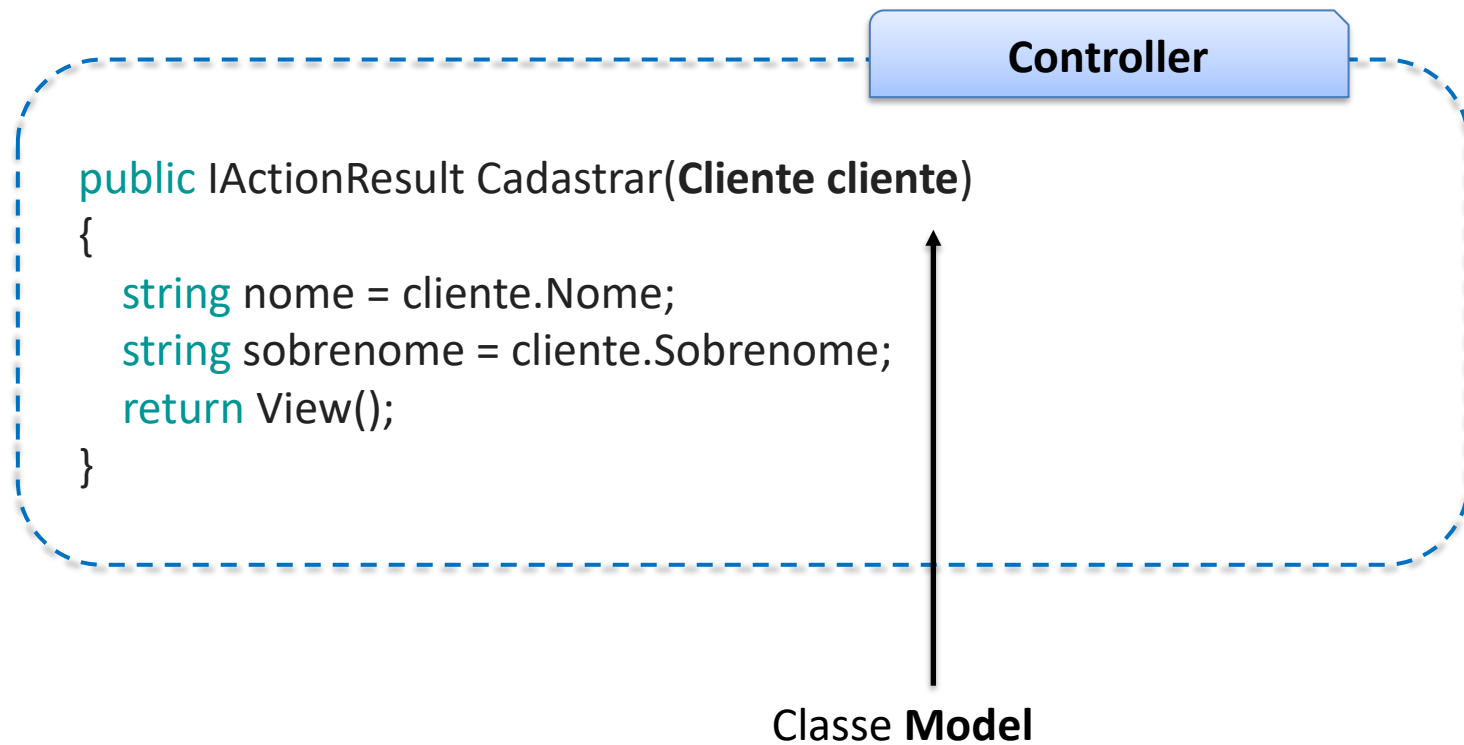
Mesmo **nome** dos campos do formulário HTML

- Podemos utilizar também uma **classe** como **parâmetro**, assim o **ASP.NET Core MVC** irá criar o objeto com os valores dos campos do formulário HTML;
- A classe deve ter as **propriedades** iguais aos **nomes** dos campos do formulário;



Mesmo **nome** dos campos do formulário HTML

- O método (action) recebe a classe como **parâmetro**:

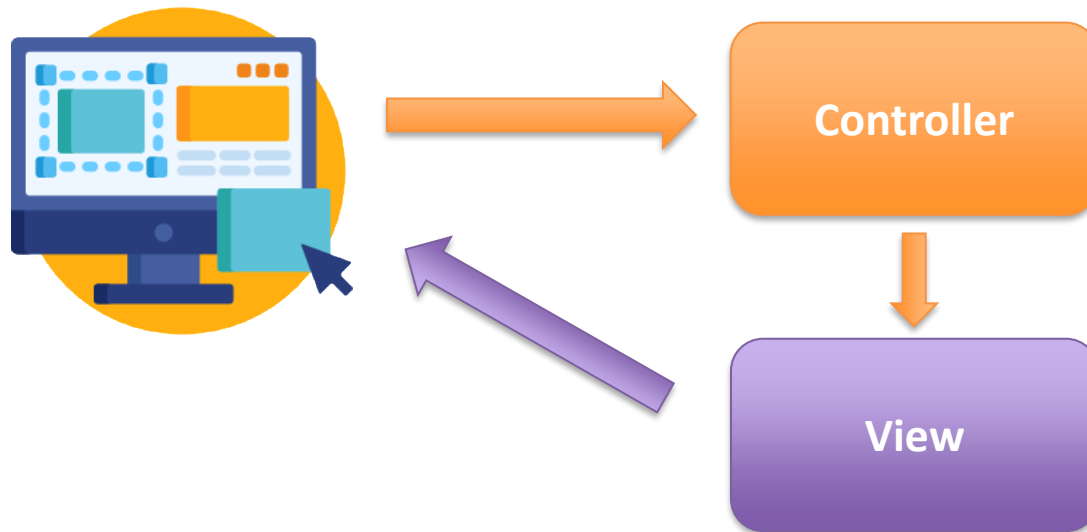


# ENVIANDO VALORES PARA A VIEW

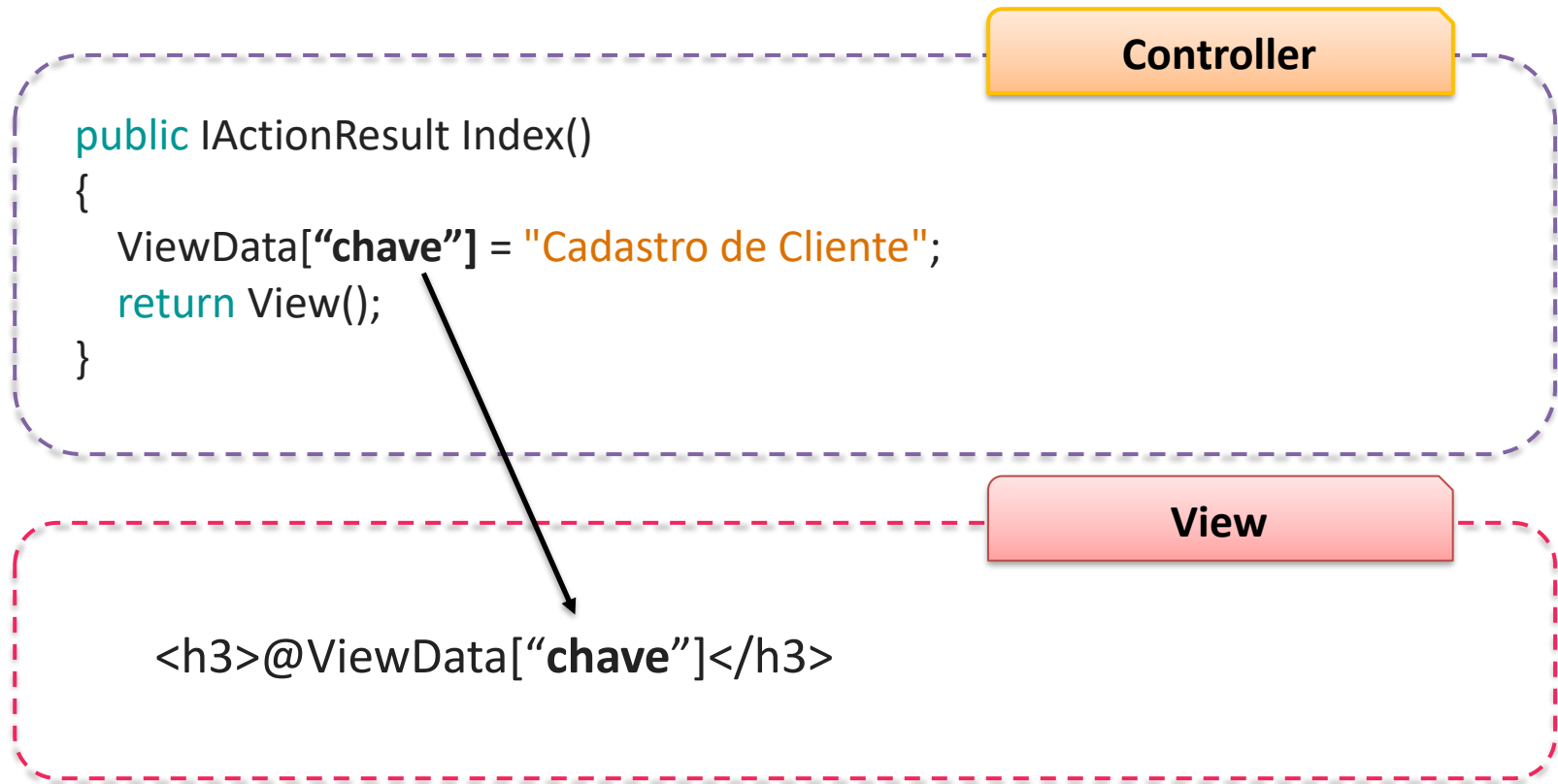




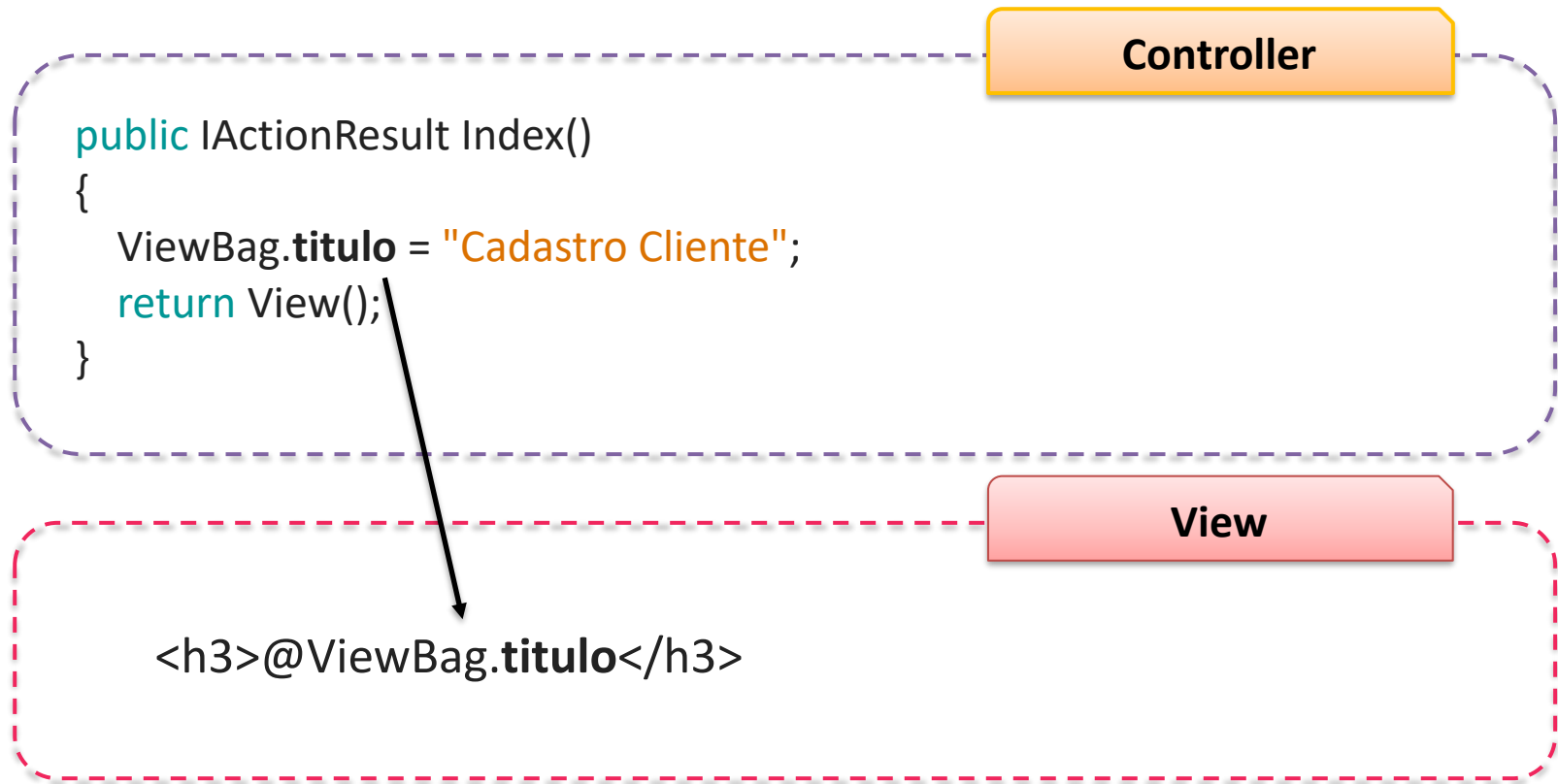
- Depois de **recuperar e processar** as informações, o **Controller** deve **retornar um resultado** para o usuário;
- Para isso, o **Controller** deve **enviar informações para a view** ser construída e exibida para o usuário;



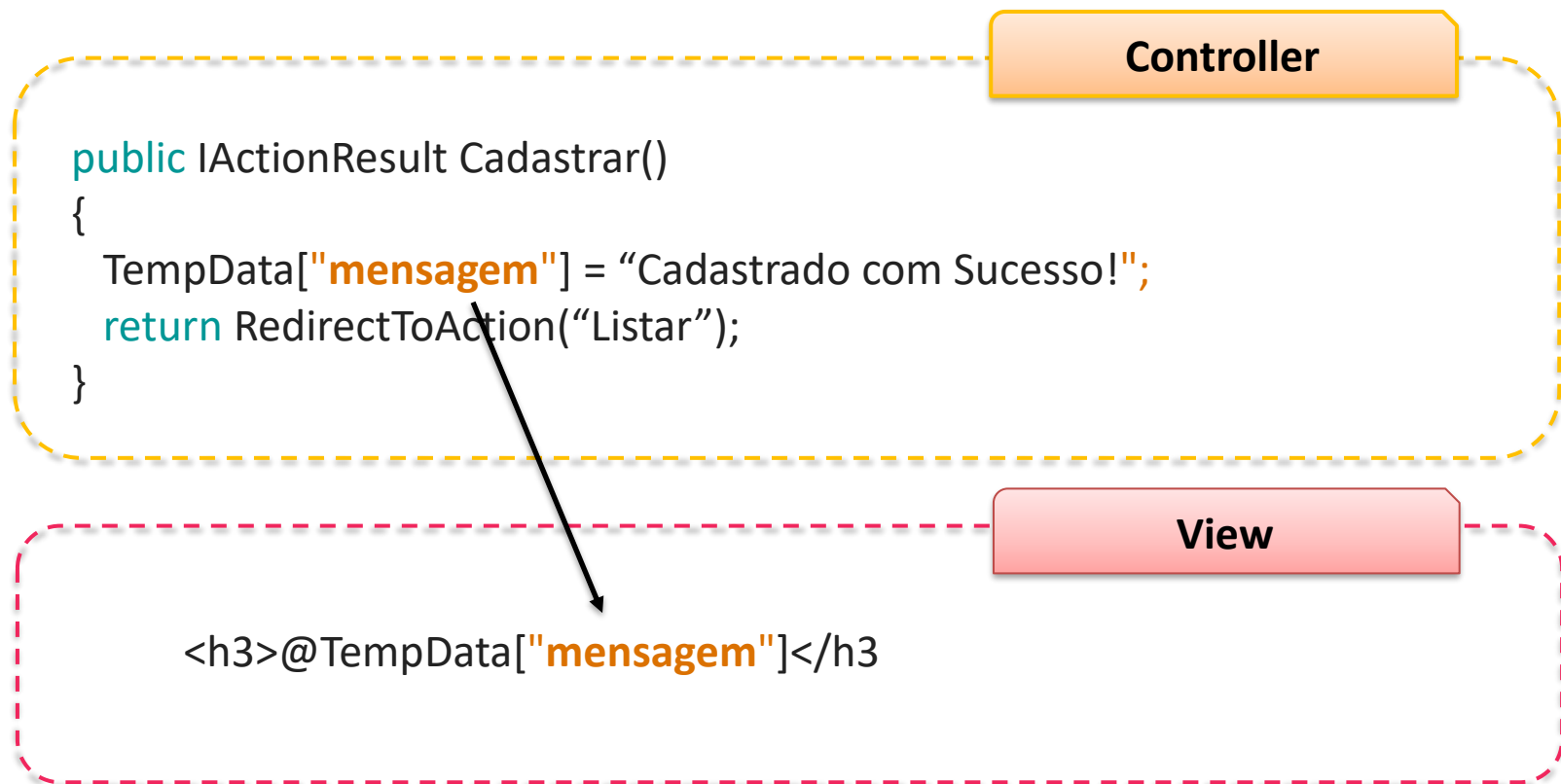
- Trabalha com pares **chave/valor**;



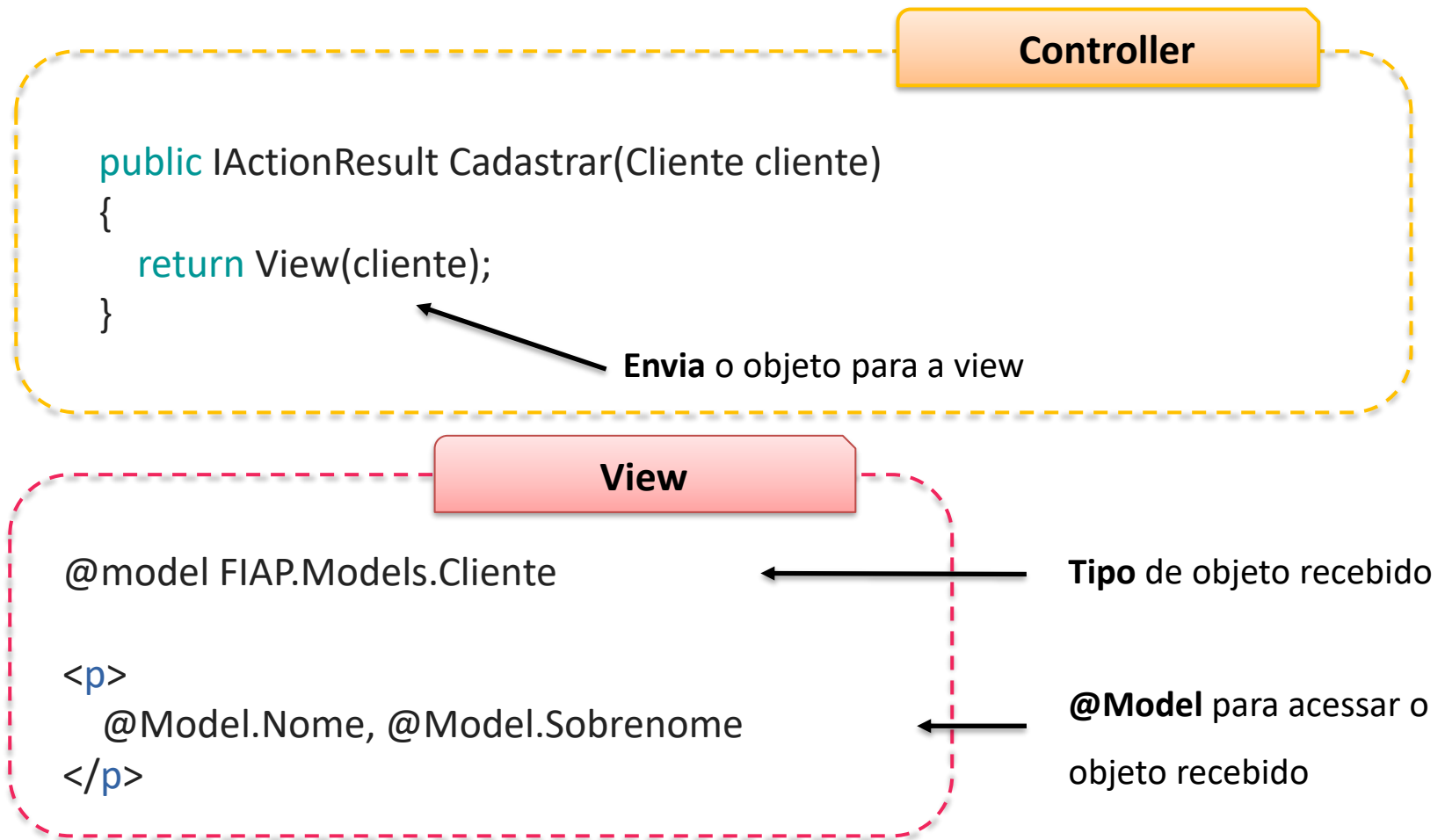
- Parecido com o ViewData, porém **não** precisa de casting ou validação de objetos nulos;



- TempData funciona de forma parecida com os outros dois, porém consegue **manter** as informações após um **redirect**:

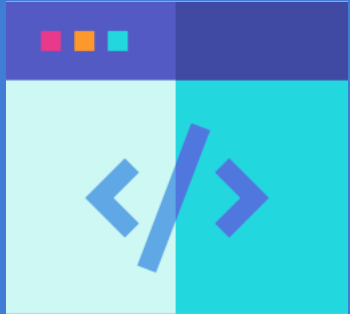


- Podemos enviar um **objeto** para a **view** passando o objeto como parâmetro do **retorno da action**:



# CODAR!

---



- Crie um **formulário** para receber informações de um usuário:
  - Nome
  - Data Nascimento
  - E-mail
- Depois de enviar as informações, o usuário deve receber uma **mensagem de sucesso** e as informações do usuário enviadas.

# VOCÊ APRENDEU..

---



- Sobre o ASP.NET e sua evolução;
- O sistema de **rotas** do **ASP.NET Core MVC**;
- As responsabilidades do **controller** e os tipos de **retorno** de uma action;
- A diferença entre **forward** e **redirect**;
- Recuperar os **parâmetros** enviados para o controller;
- Enviar **dados** do **controller** para a camada de **view**;

# Copyright © 2013 – 2023

## Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

*“Descobri que quanto mais eu estudo, mais sorte eu pareço ter nas provas”*