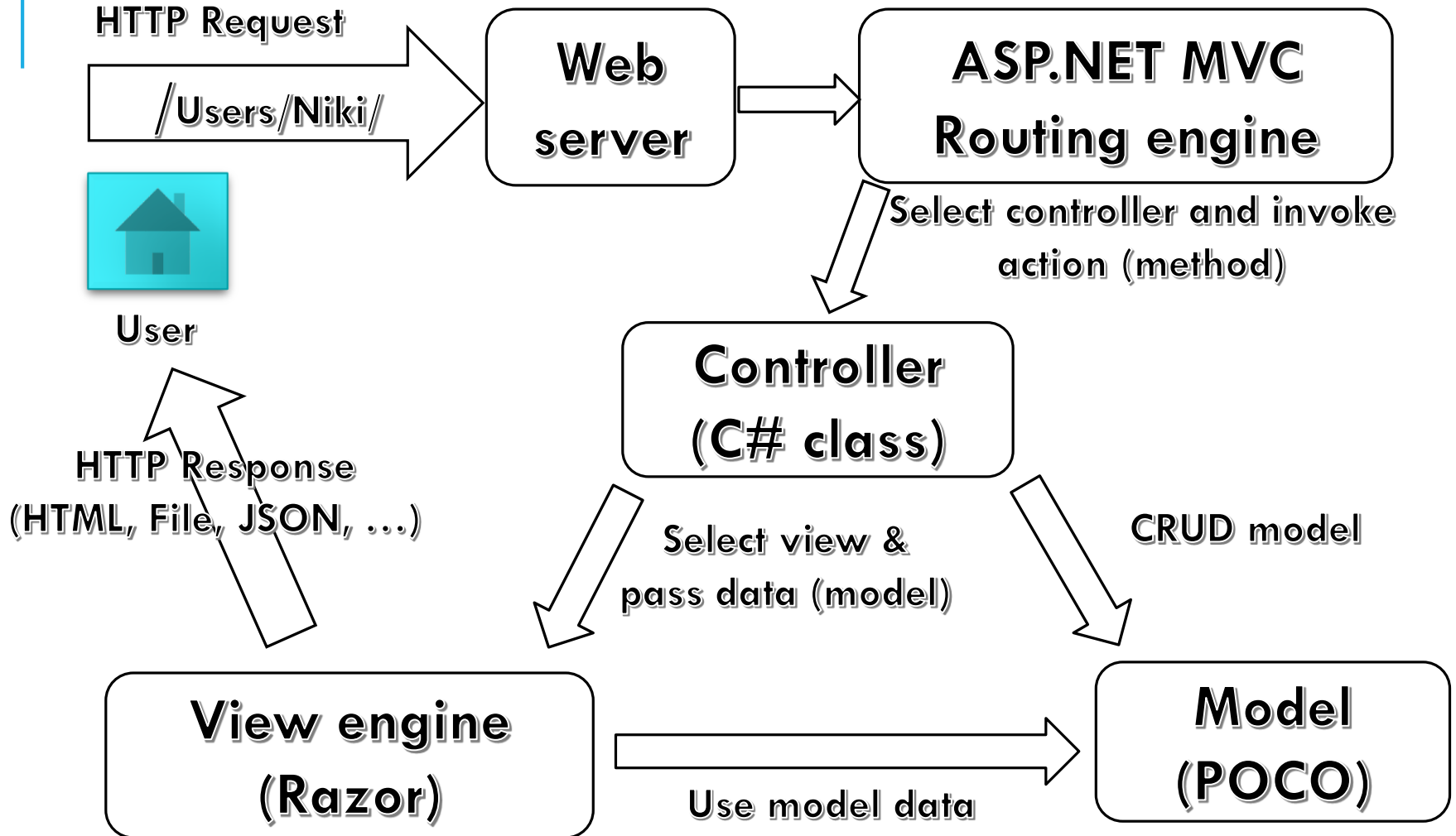


PROGRAMAÇÃO VISUAL

ASP.NET Core MVC - Complementos

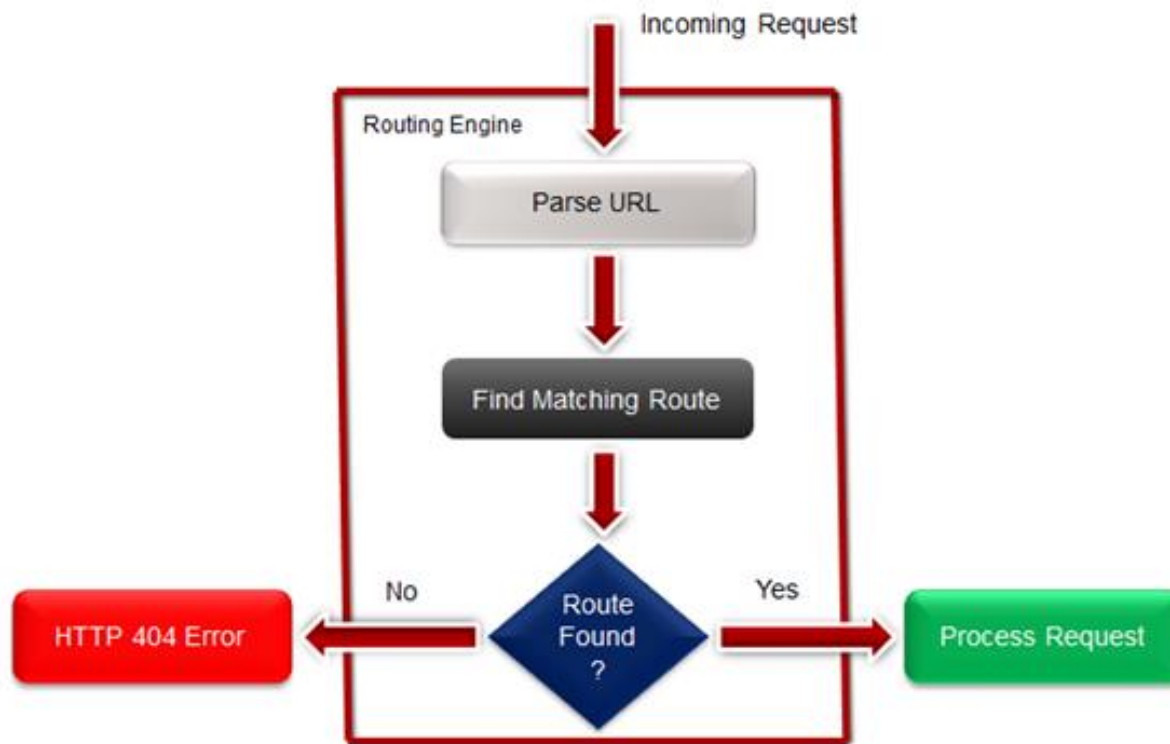


MVC – MODELO DE FUNCIONAMENTO



MVC – ENCAMINHAMENTO (ROUTING)

HOW ROUTING WORKS



MVC — ENCAMINHAMENTO (ROUTING)

Mapeamento entre padrões e uma combinação de **controlador + ação + parâmetros**;

Routes são definidas na aplicação ASP.NET e são configuradas no arranque.

Algoritmo

- O primeiro *match* ganha

MVC – ENCAMINHAMENTO (ROUTING)

Configurado em **Startup.cs** no método **Configure** da classe **Startup**

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    //...

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

Route name

Route pattern

Default parameters

MVC – EXEMPLOS DE ENCAMINHAMENTO

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

<http://localhost/Products/ById/3>



Controller: Products

Action: ById

Id: 3

MVC – EXEMPLOS DE ENCAMINHAMENTO

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

<http://localhost/Products/ById>

Controller: Products

Action: ById

Id: o (parâmetro opcional)

MVC – EXEMPLOS DE ENCAMINHAMENTO

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

<http://localhost/Products>

Controller: Products

Action: Index

Id: o (parâmetro opcional)

MVC – EXEMPLOS DE ENCAMINHAMENTO

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

<http://localhost/>



Controller: Home

Action: Index

Id: o (parâmetro opcional)

MVC — EXEMPLOS DE ENCAMINHAMENTO

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "users",
        template: "Users/{username}",
        defaults: new { controller = "Users", action = "ByUsername" });
});
```

<http://localhost/Users/Jose>



Controller: Users

Action: ByUserName

Id: Jose

MVC – EXEMPLOS DE ENCAMINHAMENTO

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "users",
        template: "Users/{username}",
        defaults: new {
            controller = "Users",
            action = "ByUsername",
            username = "DefaultValue"
        });
});
```

<http://localhost/Users>

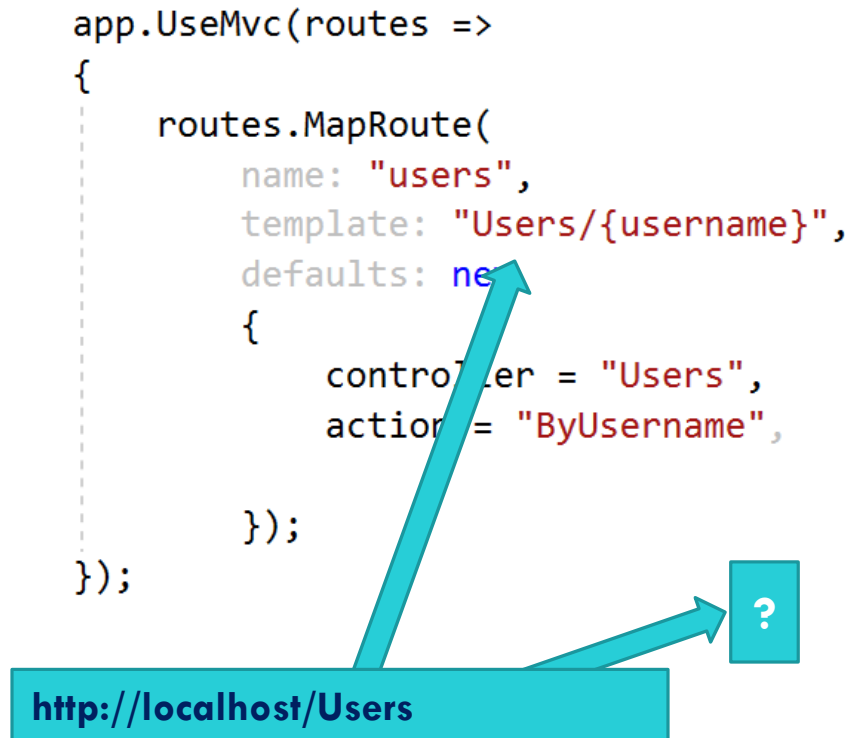


Controller: Users

Action: ByUserName

UserName: DefaultValue

MVC – EXEMPLOS DE ENCAMINHAMENTO



Result: 404 Not Found

MVC — RESTRIÇÕES DO ENCAMINHAMENTO

Restrições (**Constraints**) são regras sobre os *segmentos* de URL

As restrições podem ser fornecidas como expressões regulares (compatíveis com a classe **Regex**)

Definidas como um dos parâmetros de `routes.MapRoute(...)`

```
// 2013/01/29/Blog-title
routes.MapRoute(
    name: "Blog",
    url: "{year}/{month}/{day}",
    defaults: new { controller = "Blog", action = "ByDate" },
    constraints: new { year=@"\d{4}", month=@"\d{2}", day=@"\d{2}" }
);
```

)?

MVC - AÇÕES

Ações representam o destino final dos pedidos

- São métodos públicos do controlador
- Não estáticos
- Sem restrições ao valor de retorno

As ações retornam normalmente um **ActionResult**

```
public IActionResult Contact()  
{  
    ViewData["Message"] = "Your contact page.";  
  
    return View();  
}
```

MVC — RESULTADOS DAS AÇÕES

Respondem a um pedido do Browser;

Implementam a interface **IActionResult**;

São 5 os principais grupos de Action Results:

- Status Code
- Status Code com Object Results
- Redirecionamento
- Ficheiros
- Conteúdo

MVC — RESULTADOS DAS AÇÕES

Tipos de result:

Name	Framework Behavior	Producing Method
ContentResult	Returns a string literal by default, may return other types.	Content
EmptyResult	No response. For command operations like delete, update, create	
FileContentResult PhysicalFileResult FileStreamResult VirtualFileResult	Return the contents of a file. Derived from FileResult	File

MVC — RESULTADOS DAS AÇÕES

Name	Framework Behavior	Producing Method
ChallengeResult	Authentication credential not valid or not present. Returns an HTTP 401 status code	
UnauthorizedResult	Returns an HTTP 401 status code and nothing else	
RedirectResult RedirectToActionResult RedirectToRouteResult LocalRedirectResult	Redirects the client to a new URL. Derived from RedirectToActionResult	Redirect / RedirectPermanent

MVC – RESULTADOS DAS AÇÕES

Name	Framework Behavior	Producing Method
ForbiddenResult	To refuse a request to a particular resource. Returns an HTTP 403 status	
JsonResult	Returns data in JSON format	Json
ViewResult PartialViewResult	Response is the responsibility of a view engine- To render a view or part of it.	View / PartialView
ViewComponentResult	Returns html content for a view	
SignInResult SignOutResult	Sign in or sign out the user based on provided mechanism	

... → Existem mais tipos de retorno:

MVC — PARÂMETROS DAS AÇÕES

ASP.NET MVC mapeia a informação do pedido HTTP para os parâmetros das ações de diferentes formas:

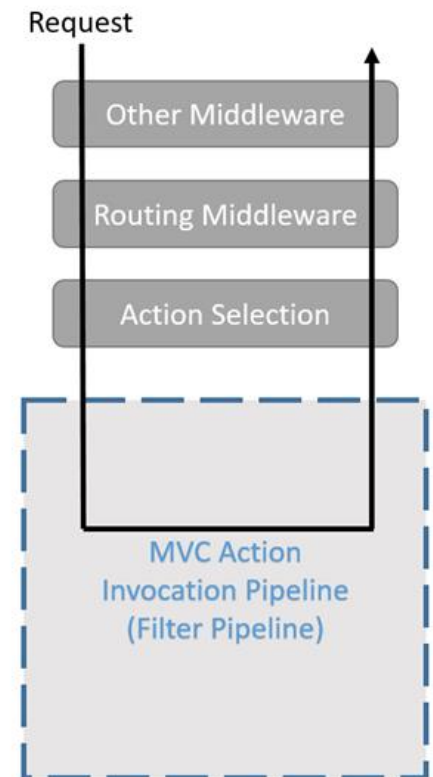
- Routing engine fornece os parâmetros das ações
 - `http://localhost/Users/NikolayIT`
 - Routing pattern: `Users/{username}`
- URL query string fornece os parâmetros
 - `/Users/ByUsername?username=NikolayIT`
- HTTP post pode igualmente fornecer os parâmetros

```
public IActionResult ByUsername(string username)
{
    return Content(username);
}
```

MVC - FILTERS

Aplicam a lógica de pré- e pós-processamento

Podem ser aplicados a ações e controladores



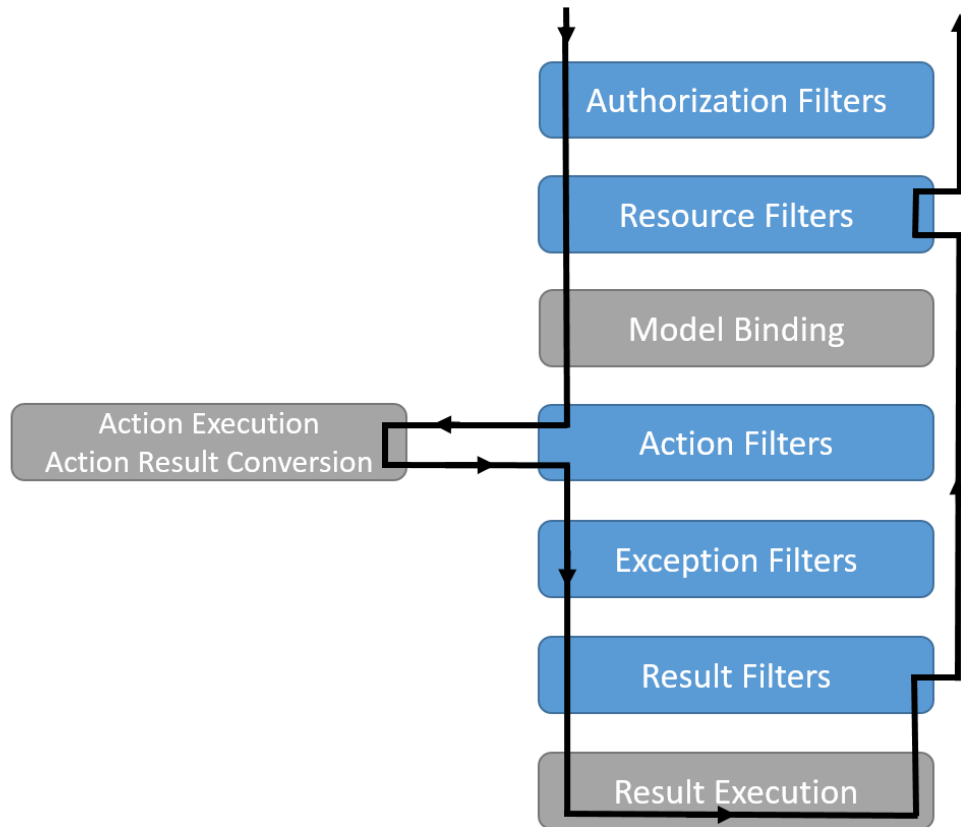
MVC - FILTERS

Tipos de Filtros:

- **Authorization filters** correm primeiro e verificam se o utilizador atual está autorizado a fazer este pedido.
- **Resource filters** correm a seguir aos filtros de autorização. Podem ser utilizados para implementar caches.
- **Action filters** correm código imediatamente antes e depois de um método ser chamado
- **Exception filters** usados para aplicarem políticas globais a exceções não tratadas que ocorrem antes de qualquer escrita no corpo da resposta.
- **Result filters** correm código imediatamente antes e depois da escrita dos resultados de uma ação e apenas quando o método foi executado com sucesso.

MVC - FILTERS

Tipos de Filtros:



MVC - ACTION SELECTORS

ActionName(string name)

NonAction

AcceptVerbs

- HttpPost
- HttpGet
- HttpDelete
- HttpOptions
- ...

RequireHttps

ChildActionOnly – Apenas para **Html.Action()**

```
public class UsersController : Controller
{
    [AcceptVerbs("UserLogin")]
    [HttpPost]
    [RequireHttps]
    public IActionResult Login(string pass)
    {
        return Content(pass);
    }
}
```

MVC — PASSAR DADOS PARA UMA VISTA

Através de **ViewData** (dictionary)

- `ViewData["message"] = "Hello World!";`
- `View: @ViewData["message"]`

Vistas **Strongly-typed**:

- Action: `return View(model);`
- View: `@model ModelDataType;`

Através de **ViewBag** (dynamic type):

- Action: `ViewBag.Message = "Hello World!";`
- View: `@ViewBag.Message`

MVC – PASSAGEM DE DADOS PARA A VISTA

ByUsername.cshtml

```
@model MyFirstMvcApplication.Models.UserModel
@{
    ViewBag.Title = Model.Username;
}
<h1>@ViewBag.Title</h1>
<p>@Model.FullName is @Model.Age years old</p>
```

HTML Output

NikolayIT

Nikolay Kostov is 22 years old

Template



Data



Generated Output

UsersController.cs

```
public IActionResult ByUsername(string username)
{
    var userModel = new UserModel()
    {
        Username = "NikolayIT",
        FullName = "Nikolay Kostov",
        Age = 22
    };
    return View(userModel);
}
```

UserModel.cs

```
public class UserModel
{
    public string Username { get; set; }
    public string FullName { get; set; }
    public int Age { get; set; }
}
```

MVC - VISTAS

Templates HTML da aplicação

Razor **view engine** disponível

- View engines executam o código e fornecem o HTML
- Disponibilizam vários *helpers* para a geração do HTML
- Em ASP.NET Core MVC usa-se o **Razor**

É possível passar informação para as vistas através de: **ViewBag**, **ViewData** e **Model** (strongly-typed views)

Views suportam *master pages* (layout views)

Outras vistas podem ser renderizadas (partial views)

VISTAS E LAYOUT

Define um modelo comum para o *site*;

Similar às ASP.NET *master pages* (mas melhor!);

Razor view engine renderiza o conteúdo de dentro para fora;

- Primeira a vista, depois o Layout

@RenderBody() –
indica o local para o
“preenchimento” que as vistas
baseadas neste *layout* têm de
preencher fornecendo o
conteúdo.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
  </head>
  <body>
    <nav>@* Menu *@</nav>
    <div id="body">
      @RenderBody()
    </div>
    <footer>@* Footer *@</footer>
  </body>
</html>
```

MVC — VISTAS E LAYOUTS

As vista não necessitam de especificar o *layout* uma vez que o valor por omissão é dado em `_ViewStart.cshtml`:

- `~/Views/_ViewStart.cshtml` (Código para todas as vistas)

Cada vista pode especificar páginas de layout particulares

```
@{  
    Layout = "~/Views/Shared/_UncommonLayout.cshtml";  
}
```

```
@{  
    Layout = null;  
}
```

MVC — VISTAS E SECÇÕES

É possível ter uma ou mais Secções (**sections**) (opcional)

As secções são definidas nas vistas:

```
@section SideBar {  
    <aside>  
        Some side information  
    </aside>  
}
```

Podem ser renderizadas em qualquer sitio do Layout usando o método **RenderSection()**

- **@RenderSection(string name, bool required)**
- Se a secção for obrigatória e não estiver definida na vista é lançada uma exceção (**IsSectionDefined()**)

MVC – UTILITÁRIOS DAS VISTAS (**VIEW HELPERS**)

A propriedade **Html** das vistas tem métodos que retornam uma *string* que pode ser usada na criação do HTML (**HTML Helpers**)

- Criar inputs
- Criar links
- Criar forms

```
@using (Html.BeginForm("Search", "Users",  
                        FormMethod.Post))  
{  
    @Html.TextBox("username")  
    <input type="submit" />  
}  
@Html.Raw(htmlContent)
```

Existem outras propriedades utilitárias para além da propriedade **HTML**

- **Ajax**, **Url**, custom helpers

MVC — HTML HELPERS X TAG HELPERS

Os **Html helpers** foram substituídos em ASP.NET Core MVC pelos **Tag Helpers**. No entanto, ainda são válidos e são utilizados por detrás dos **Tag Helpers**

Existem alguns **Html Helpers** que não foram substituídos por **Tag helpers**.

Recomenda-se a utilização de **Tag Helpers** dado que estão alinhados com a sintaxe do **Html** e são mais simples de compreender e utilizar.

MVC - HTML HELPERS

Method	Type	Description
<i>BeginForm, BeginRouteForm</i>	Form	Returns an internal object that represents an HTML form that the system uses to render the <code><form></code> tag
<i>EndForm</i>	Form	A void method, closes the pending <code></form></code> tag
<i>CheckBox, CheckBoxFor</i>	Input	Returns the HTML string for a check box input element
<i>Hidden, HiddenFor</i>	Input	Returns the HTML string for a hidden input element
<i>Password, PasswordFor</i>	Input	Returns the HTML string for a password input element
<i>RadioButton, RadioButtonFor</i>	Input	Returns the HTML string for a radio button input element
<i>TextBox, TextBoxFor</i>	Input	Returns the HTML string for a text input element
<i>Label, LabelFor</i>	Label	Returns the HTML string for an HTML label element

MVC - HTML HELPERS (2)

Method	Type	Description
<i>ActionLink, RouteLink</i>	Link	Returns the HTML string for an HTML link
<i>DropDownList, DropDownListFor</i>	List	Returns the HTML string for a drop-down list
<i>ListBox, ListBoxFor</i>	List	Returns the HTML string for a list box
<i>TextArea, TextAreaFor</i>	TextArea	Returns the HTML string for a text area
<i>Partial</i>	Partial	Returns the HTML string incorporated in the specified user control
<i>RenderPartial</i>	Partial	Writes the HTML string incorporated in the specified user control to the output stream
<i>ValidationMessage, ValidationMessageFor</i>	Validation	Returns the HTML string for a validation message
<i>ValidationSummary</i>	Validation	Returns the HTML string for a validation summary message

MVC — HTML HELPERS DEFINIDOS PELO UTILIZADOR

Escrita de *métodos de extensão* para a classe **HtmlHelper**

- Retornar **string** ou fazer **override** do método **ToString**
- **TagBuilder** faz a gestão do fecho dos “tags” e dos marcadores

```
public static class HtmlhelperExtensions
{
    public static TagBuilder Image(this HtmlHelper helper,
                                   string imageUrl, string alt)
    {
        TagBuilder imageTag = new TagBuilder("img");
        imageTag.MergeAttribute("src", imageUrl);
        imageTag.MergeAttribute("alt", alt);
        return imageTag;
    }
}
```

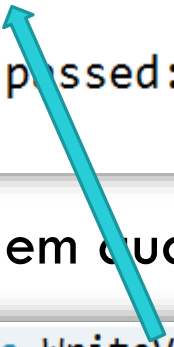
```
@Html.Image("image.jpg", "Just image");
```

MVC — HTML HELPERS DEFINIDOS PELO UTILIZADOR (2)

Outra forma de escrever os *Helpers*:

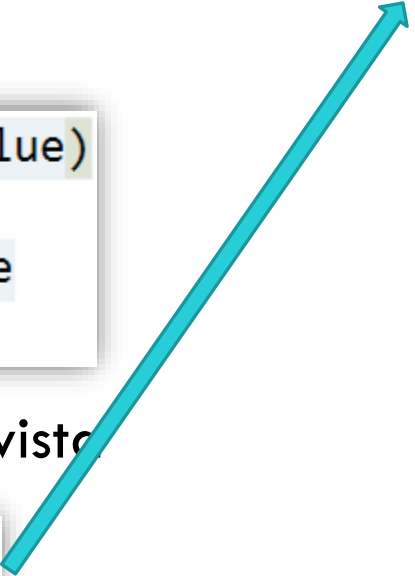
- Criar a pasta /**App_Code**/
- Criar uma vista dentro dessa pasta (por exemplo **Helpers.cshtml**)
- Escrever um helper usando **@helper**

```
@helper WriteValue(int value)
{
    @:Value passed: @value
}
```



Pode ser utilizado em qualquer vista

```
@Helpers.WriteValue(20)
```



Existe muito código numa vista? => criar helpers

MVC – VISTAS PARCIAIS

As vistas parciais permitem renderizar parte da página

- Reutilizam pedaços de vistas
- Html helpers – Partial, RenderPartial e Action

Sub-request

As vistas parciais Razor são igualmente ficheiros .cshtml

```
@using MyFirstMvcApplication.Models;  
@model IEnumerable<UserModel>
```

```
@foreach (var user in Model)  
{  
    @Html.Partial("_UserProfile", user);  
}
```

```
@Html.Partial("_UserProfile", user);
```

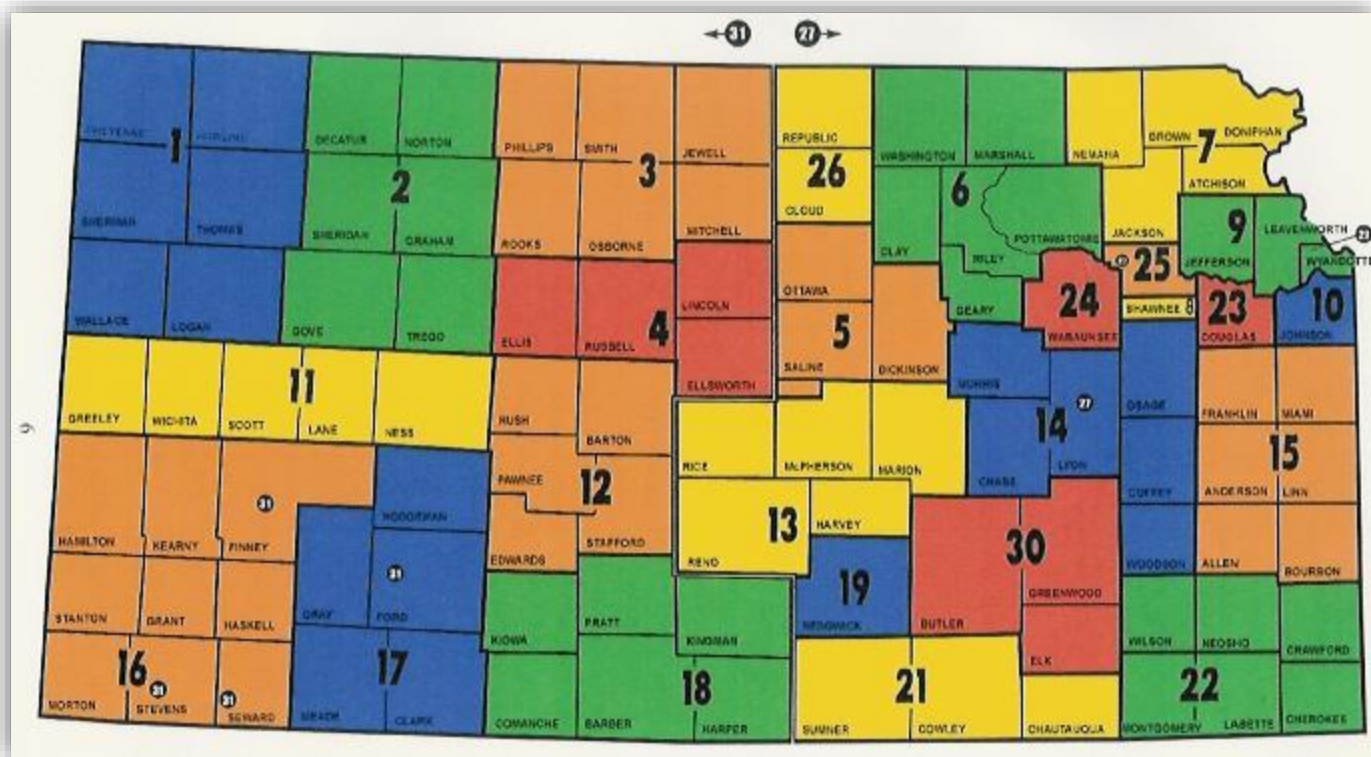
```
@Html.Partial("_UserProfile", user);
```

```
@using MyFirstMvcApplication.Models;  
@model UserModel
```

```
<h2>@ViewBag.Title</h2>  
<p>@Model.FullName is @Model.Age years old</p>
```

Na mesma pasta que as
outras vistas ou na pasta
Shared

MVC – AREAS



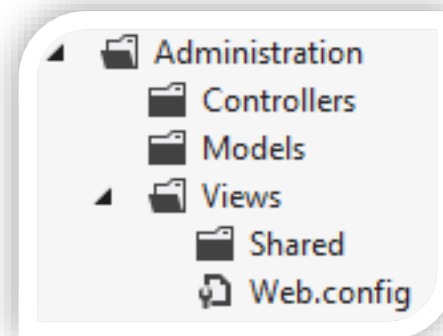
MVC - AREAS

Algumas aplicações podem ter um grande número de controladores
O ASP.NET MVC permite a partição das aplicações Web em unidades mais pequenas (**areas**)

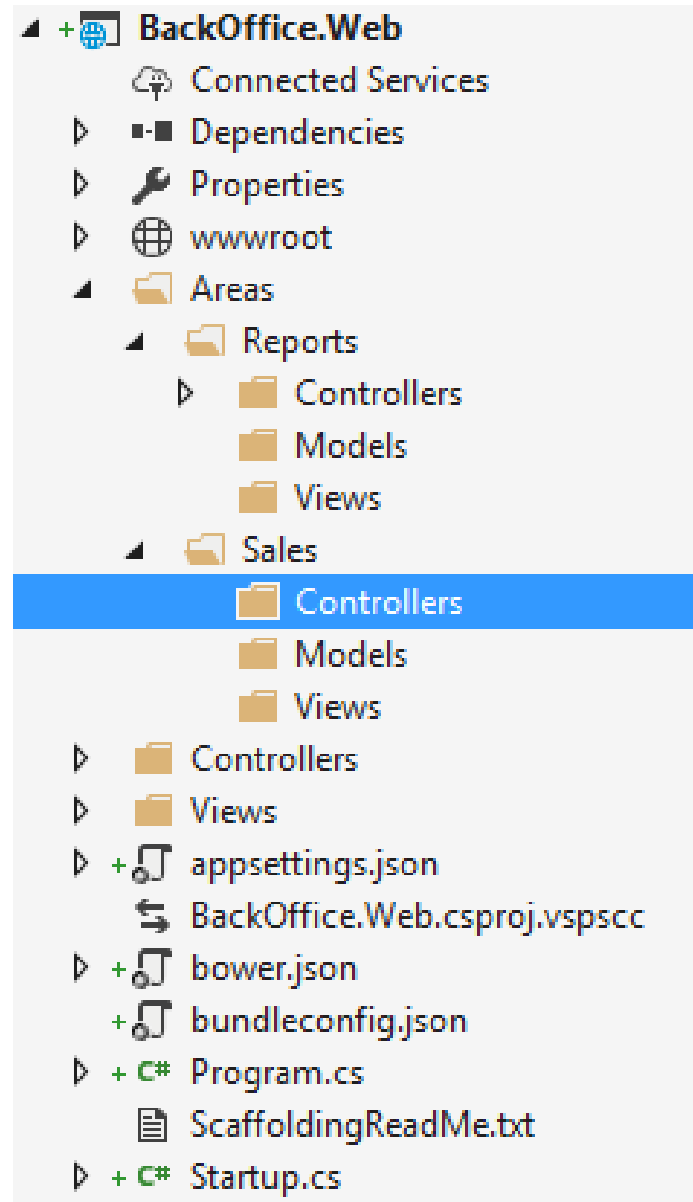
Uma **area** constitui, na prática, uma estrutura MVC dentro da aplicação

Exemplo: uma grande aplicação de e-commerce

- Loja principal, utilizadores
- Blog, fórum
- Administração



MVC - AREAS



REFERÊNCIAS

- ◆ Telerik Software Academy
 - ◆ academy.telerik.com

