

# Introdução a ASP .NET CORE 2

Bruno Oliveira – [bmo@estg.ipp.pt](mailto:bmo@estg.ipp.pt)

# No workshop anterior

- Desenvolvimento de uma Web API com os seguintes endpoints:
  - Retornar todos os utilizadores
    - </api/user>
  - Retornar um utilizador específico
    - </api/user/1>
  - Eliminar um utilizador
    - </api/user?codigo=1>
  - Adicionar um utilizador
    - </api/user> (por POST)
  - Adicionar uma tarefa:
    - </api/user?id=2>
  - Retornar todas as tarefas de um utilizador
    - </api/user/gettasks/1>
  - Retornar uma tarefa específica
    - </api/task/4>

# Tópicos

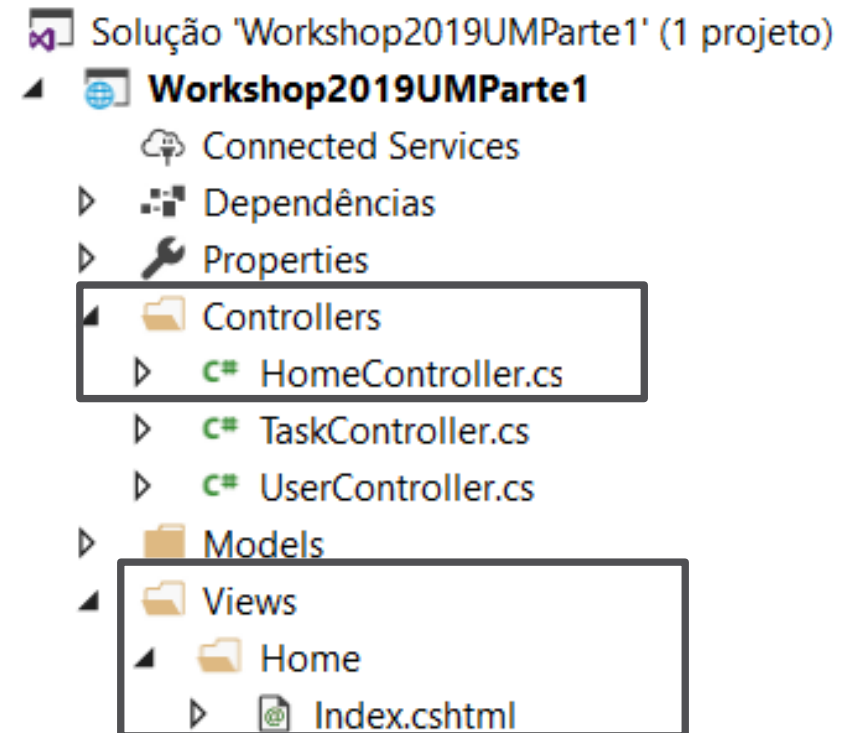
- Desenvolvimento de um front-end: Views e Razor
- Autenticação com autenticação por cookies

Página GitHub do Workshop:

<https://github.com/brunobmo/ASP.NET-Training/tree/master/2019>

# Home Controller

- Vamos criar um **HomeController** para apresentar a **página inicial** do website;
- Criar uma pasta **Views** o mesmo nível da pasta **Controllers**;
- Para o controller, criamos uma pasta dentro da pasta **Views** tendo por base a convenção:  
**HomeController -> Home**;
- De seguida, adicionar uma nova **MVC View Page: Index**:



# Home View

- Substituir o conteúdo da View por:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Hello World</title>
</head>
<body>
    <h1>Hello World</h1>
</body>
</html>
```

- Ao testar, será apresentada uma página com a frase: **Hello World** (o **HomeController** é a rota por **defeito** definida pelo método **app.UseMvcWithDefaultRoute()**; no ficheiro **startup.cs**)

# MVC e .NET

- *View engines* tornam o desenvolvimento de *views* mais simples;
- O *Razor* é o motor por defeito do ASP.NET MVC, fornecendo uma sintaxe que permite o desenvolvimento orientado ao modelo (*template-driven approach*);
- Com o *Razor* podemos “embutir” código *server-side* em páginas web através de uma *linguagem de marcação*, permitindo assim o desenvolvimento de páginas web *dinâmicas*;
- Exemplo:

```
<ul>  
@for(int i = 0; i < 10; i++) {  
    <li>@i</li>  
}  
</ul>
```

Documentação adicional:

[https://www.w3schools.com/asp/razor\\_intro.asp](https://www.w3schools.com/asp/razor_intro.asp)

<https://msdn.microsoft.com/pt-br/library/gg675215.aspx>

# MVC e .NET

- São também utilizados (Razor) **HTML helpers** que representam uma ponte entre o modelo: *drag-and-drop* e a escrita de HTML;
- Os **HTML helpers** disponibilizam métodos simples que **simplificam** a o desenvolvimento de documentos HTML;
- Exemplo:

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
</head>
<body>
  <div>
    @Html.ActionLink("adicionar Atleta", "Index", "Atleta" );
    @Html.ActionLink("ver atletas", "verAtletas", "Atleta");
  </div>
</body>
</html>
```

Documentação adicional:

<http://www.tutorialsteacher.com/mvc/html-helpers>

# Funcionalidades específicas da view

- Dois novos components são utilizados na manipulação de views:
  - Tag Helpers:
    - Em comparação com os helpers Razor, as Tag Helpers são mais parecidas com os elementos HTML, sendo necessário estar a alternar entre a syntax Razor e HTML;
    - Instalar o package:  
**Microsoft.AspNetCore.Mvc.TagHelpers**



# Funcionalidades específicas da view

- Para utilizar o Tag Helper é necessário configurar o .NET core de onde o pode encontrar;
- Criar na pasta views um ficheiro com o nome: [\\_ViewImports.cshtml](#)
- Adicionar a seguinte definição no ficheiro [\\_ViewImports.cshtml](#):

```
@addTagHelper "*", Microsoft.AspNetCore.Mvc.TagHelpers"
```

# Funcionalidades especificas da view

- Alterar a vista: Home -> Index:

```
(...)  
<body>  
    <h1>Hello World</h1>  
    <a asp-controller="UserView" asp-action="getUsers">Ver utilizadores</a>  
  
</body>  
(...)
```

# UserViewController

- Criar um novo controller UserViewController para ligar com a lógica relacionada com a apresentação de views;

```
[Route("[controller]/[action]")]
public class UserViewController : Controller
{
    public IActionResult getUsers()
    {
        return View();
    }
}
```

# UserViewController

- Adicionar uma view: `getUsers`;
  - Clique (botão direito) sobre o nome do método e selecionar: `Add View` (automaticamente é criada a estrutura de pastas e ficheiros necessários);
- Este novo controller será responsável por suportar a `gestão de views`, enquanto que o `UserController` é utilizado para suportar a web api;

# UserController

- Alterar o UserController para tirar partida da classe: **UserHandling**;

```
public class UserController : Controller
{
    private UserHandling userHandling;
    public UserController(UserContext context)
    {
        userHandling = new UserHandling(context);
    }

    [HttpGet]
    public User[] Get()
    {
        return userHandling.getUsers();
    }
}
```

(...)

# UserViewController

- Alterar o UserViewController para tirar partida da classe: **UserHandling**;

```
public class UserViewController : Controller
{
    private UserHandling userHandling;
    public UserViewController(UserContext context)
    {
        userHandling = new UserHandling(context);
    }
    public IActionResult getUsers()
    {
        User[] users = userHandling.getUsers();
        return View(users);
    }
}
```

Documentação adicional:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-2.2>

# Layouts

- Uma aplicação pode conter partes comuns entre as várias `views`; Por exemplo: Cabeçalho, rodapé ou Menu;
- Podemos assim definir **templates** que podem ser **reutilizados**, reduzindo a quantidade de código que é necessário reproduzir;
- No projeto criado, adicionar o ficheiro: **`_Layout.cshtml`** que contém uma estrutura base de exemplo;

Documentação adicional:

<http://www.tutorialsteacher.com/mvc/layout-view-in-asp.net-mvc>

# \_Layout.cshtml

- Git Gist com o código:
  - <https://gist.github.com/brunobmo/a76f3823f2a4975639909689f929433d>
- Inclusão das folhas de estilo e do bootstrap (framework para o desenvolvimento de interfaces - <https://getbootstrap.com/>)

(...)

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"]</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<link rel="stylesheet" href="~/css/site.css" />

</head>

(...)



# \_Layout.cshtml

```
(...)  
<div class="container">  
  <main role="main" class="pb-3">  
    @RenderBody()  
  </main>  
</div>  
(...)
```

Placeholder que indica o local onde será renderizado o conteúdo do elemento <body>

# \_ViewStart.cshtml

- O ficheiro `_ViewStart.cshtml` define o `layout` para as várias `views` da pasta em que está contido:

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

- Para aplicar o estilo podemos também incluir a propriedade `layout` do Razor em cada página:

Documentação adicional:

<http://www.tutorialsteacher.com/mvc/layout-view-in-asp.net-mvc>

# Layouts

- Alterar a view `index` do controller `Home`:

```
@{  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
(...)  
<body>  
    <div class="mycontainer">  
        <ul>  
            <li>@Html.ActionLink("adicionar Atleta", "Index", "Atleta")</li>  
            <li> @Html.ActionLink("ver atletas", "verAtletas", "Atleta")</li>  
            <li> @Html.ActionLink("Login", "index", "Login")</li>  
        </ul>  
    </div>  
</body>  
</html>
```

# Scripts

```
<script src="/lib/jquery/dist/jquery.js"></script>  
<script src="/lib/bootstrap/dist/js/bootstrap.bundle.js"></script>
```

- Podem instalar as frameworks como bootstrap ou jquery utilizando o [Manage NuGet packages](#);
- Podem também criar uma pasta com o nome [wwwroot](#) (que é uma convenção para colocação de static files) e copiar os ficheiros disponibilizados no repositório do github: css, js, lib, favicon.icon;

# Static Files

- Por defeito, um projeto ASP.NET Core não tem a capacidade de server static files como folhas de estilo ou scripts em JavaScript;
- Para habilitar essa funcionalidade temos de ativar um middleware específico;
- Instalar o package: [Microsoft.AspNetCore.App](#);
- Acrescentar o seguinte código no ficheiro startup.cs -> método Configure:

```
app.UseStaticFiles();
```

# Home -> index.cshtml

- Alterar o ficheiro index.cshtml do controller Home para:

```
<div class="text-center">  
  <h1 class="display-4">Home Page</h1>  
  <a asp-controller="UserView" asp-action="getUsers">Ver utilizadores</a>  
</div>
```

Documentação adicional:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-2.2>

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-2.2>

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-2.2>

# Autenticação

- O ASP.NET Core disponibiliza diversas formas de implementar o processo de **autenticação** na aplicação web:
  - ASP.NET Core **Identity** é um sistema bastante completo de associação que permite a criação de contas armazenadas pela **Identity** ou utilizando um **login provider** (como o facebook e a google).
  - **Cookie Authentication** permite criar uma autenticação personalizada baseada em **cookies**, podendo ser utilizada sem a ASP.NET Core Identity;

# Cookie Authentication

- Neste modo de autenticação, cada vez que é realizado um pedido ao website, o browser vai incluir os **cookies** (que representa uma associação entre uma chave e um valor);
- O **servidor** é responsável por **verificar** os cookies;
- No servidor é possível **configurar** diversos aspetos relacionados com os cookies como a **data de validade** ou modo de **encriptação**;
- Um cookie encriptado é conhecido como **cookie assinado**, permitindo manter a segurança de comunicação entre cliente e servidor;
- O browser armazena os **cookies** definidos pelo servidor associado ao **domínio** ao qual foram definidos

Documentação adicional:

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-2.2>



# Formulário de registo

- Adicionar o seguinte método ao UserController:

```
[HttpGet]  
public IActionResult RegisterUser()  
{  
    return View();  
}
```

- Criar uma view: RegisterUser;

# Formulário de registo

- Alterar a view: RegisterUser:

```
@model Workshop2019UMParte1.Models.User
@{
    ViewData["Title"] = "RegisterUser";
}
```

```
<h4>New User</h4>
<hr />
(...)
```

Modelo a utilizar

**ViewData** é um **dicionário** de objetos com uma key. Do lado dos servidor (no controller) Podemos definir esta instrução e tornar este tipo de informação mais dinâmica.

# View: RegisterUser

- Alterar a view: RegisterUser:

```
(...)  
<div class="row">  
  <div class="col-md-4">  
    @if (TempData["Success"] != null)  
    {  
      <p class="alert alert-success">@TempData["Success"]  
      <a asp-action="UserLogin">Click here to login</a></p>  
    }  
    @if (TempData["Fail"] != null)  
    {  
      <p class="alert alert-danger">@TempData["Fail"]</p>  
    }  
  }  
(...)
```

`TempData` pode ser utilizada para armazenar informação temporária que pode ser utilizada em pedidos subsequentes

Documentação adicional:

<https://www.tutorialsteacher.com/mvc/tempdata-in-asp.net-mvc>

# View: RegisterUser

- Alterar a view: RegisterUser: Adicionar o formulário:

(...)

`<form asp-action="RegisterUser">` Ação em que os dados dos formulário vão ser enviados

`<div asp-validation-summary="ModelOnly" class="text-danger"></div>`

`<div class="form-group">`

`<label asp-for="nome" class="control-label"></label>`

`<input asp-for="nome" class="form-control" />`

`<span asp-validation-for="nome" class="text-danger"></span>`

`</div>`

(...)

`</form>`

A mensagem de validação da propriedade é colocada num element span. Também adiciona a validação do domínio dos atributos através do jQuery

Não permite a submissão do formulário, baseando-se nas restrições do modelo

Documentação/Código completo da View:

<https://gist.github.com/brunobmo/12d119321625fa19d33d1bbdfd5a5f35>

<https://docs.microsoft.com/en-us/aspnet/web-pages/overview/ui-layouts-and-themes/4-working-with-forms#creating-a-simple-html-form>

# RegisterUser – validação cliente-side

- A validação client-side evita pedidos desnecessários de dados para o servidor quando o formulário possui erros desnecessários;
- A validação **não intrusiva** é suportada pela Microsoft através de jQuery;
- Para isso, devemos incluir na secção de scripts (por exemplo no ficheiro: **\_Layout.cshtml**):

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.17.0/jquery.validate.min.js"></script>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-unobtrusive/3.2.11/jquery.validate.unobtrusive.min.js"></script>
```

Código completo da View:

# RegisterUser - Resultado

- Incluir uma hiperligação para a nova página na view Index do HomeController:

```
<div class="text-center">
  <h1 class="display-4">Home Page</h1>
  <p>
    <a asp-controller="UserView" asp-action="getUsers">Ver utilizadores</a></p>
  <p>
    <a asp-controller="UserView" asp-action="RegisterUser">Registo</a></p>
  <p><a asp-controller="UserView" asp-action="UserLogin">Login</a></p>
</div>
```

# Armazenar o utilizador

- No controller `UserViewController` (por questões de simplicidade, apenas será abordada a gestão de view em detrimento da web API), adicionar a seguinte ação:

```
[HttpPost]
public IActionResult RegisterUser([Bind] User user){
    if (ModelState.IsValid){
        bool RegistrationStatus = this.userHandling.RegisterUser(user);
        if (RegistrationStatus){
            ModelState.Clear();
            TempData["Success"] = "Registration Successful!";
        }else{
            TempData["Fail"] = "This User ID already exists. Registration Failed.";
        }
    }
    return View();
}
```

Documentação adicional:

<https://exceptionnotfound.net/asp-net-core-demystified-model-binding-in-mvc/>

# Armazenar o utilizador

- O atributo: `Bind attribute` informa a framework que pretendemos associar as propriedades do objeto com os dados recebidos pelo servidor:

```
public IActionResult RegisterUser([Bind] User user)
```

- O `ModelState` representa os erros ocorridos pelo model binding ou model validation:

```
if (ModelState.IsValid)
```

Documentação adicional:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-2.2>



# UserHandling

- Para que o código anteriormente apresentado é necessário armazenar o utilizador na base de dados:

```
public bool registerUser(User user)
{
    user.password = MyHelpers.HashPassword(user.password);
    _context.user.Add(user);
    _context.SaveChanges();
    return true;
}
```

# Encriptar!

- De forma a aumentar a segurança do website, vamos **encriptar** a password armazenada na base de dados;
- Adicionar os seguintes métodos numa nova classe: **MyHelpers**:
- Método gerar uma Hash Md5:

```
public static string GetMd5Hash(MD5 md5Hash, string input)
{
    byte[] data = md5Hash.ComputeHash(Encoding.UTF8.GetBytes(input));
    StringBuilder sBuilder = new StringBuilder();
    for (int i = 0; i < data.Length; i++){
        sBuilder.Append(data[i].ToString("x2"));
    }
    return sBuilder.ToString();
}
```

# Encriptar!

- Método para encriptar a password:

```
public static String HashPassword(string password)
{
    using (MD5 md5Hash = MD5.Create())
    {
        string hash = GetMd5Hash(md5Hash, password);
        return hash;
    }
}
```

Documentação adicional:

<https://gist.github.com/brunobmo/061b4724ca2d629d686a59366aa631b0> (GitGist)

<https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-2.1>

# Encriptar!

- Método para comparar:

```
public static bool VerifyMd5Hash(MD5 md5Hash, string input, string hash)
{
    string hashOfInput = GetMd5Hash(md5Hash, input);
    StringComparer comparer = StringComparer.OrdinalIgnoreCase;
    if (0 == comparer.Compare(hashOfInput, hash)){
        return true;
    }
    else{
        return false;
    }
}
```

Documentação adicional:

<https://gist.github.com/brunobmo/061b4724ca2d629d686a59366aa631b0> (GitGist)

<https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-2.1>

# Encriptar!

- Resultado:

5	Bruno2	bruno.oliveira3...	bmo	81dc9bdb52d04dc20036dbd8313ed055
---	--------	--------------------	-----	----------------------------------

Documentação adicional:

<https://gist.github.com/brunobmo/54794cd03031dcf472f427642d5c82ba> (GitGist)

# Login

- Utilizando um processo semelhante ao registo:
  - Adicionar uma `action` para apresentação da View de `Login` (UserController):

```
[HttpGet]  
public IActionResult UserLogin(){  
    return View();  
}
```

# Login

- Utilizando um processo semelhante ao registo:

- Adicionar a view:

```
@model Workshop2019UMParte1.Models.User
(...)
<div class="row">
    <div class="col-md-4">
        @if (TempData["UserLoginFailed"] != null)
        {
            <p class="alert alert-danger">@TempData["UserLoginFailed"]</p>
        }
        <form asp-action="UserLogin">
    (...)
</div>
```

Documentação adicional:

<https://gist.github.com/brunobmo/a433ace5c8d9d466c5f2aacf6d06110b> (GitGist)

# Action para receber dados do login

- Adicionar ao UserController:

Utilizado por questões de segurança: escreve um valor único no cookie HTTP-only, prevenindo ataques: **Cross-site request forgery**

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> UserLogin([Bind] User user){
    ModelState.Remove("nome");
    ModelState.Remove("email");
    if (ModelState.IsValid){
        var LoginStatus = this.userHandling.validateUser(user);
    }
    (...)
```

Documentação adicional:

<https://gist.github.com/brunobmo/00b972edb018b2f5f8926996cdc7f8b6> (GitGist)



# Action para receber dados do login

- Adicionar ao UserController:

```
if (LoginStatus){  
    var claims = new List<Claim>{  
        new Claim(ClaimTypes.Name, user.username)  
    };  
    ClaimsIdentity userIdentity = new ClaimsIdentity(claims, "login");  
    ClaimsPrincipal principal = new ClaimsPrincipal(userIdentity);  
    await HttpContext.SignInAsync(principal);  
    return RedirectToAction("UserView", "getUsers");  
}  
(...)
```

Cria um cookie armazenando a informação de utilizador. A informação do utilizador é serializada e armazenada no cookie.

Documentação adicional:

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-2.2>

# Verificar o login

- Adicionar o método validateUser:

```
public bool validateUser(User user){  
    user.password = MyHelpers.HashPassword(user.password);  
    var returnedUser = _context.user.Where(b => b.username ==  
user.username && b.password == user.password).FirstOrDefault();  
  
    if (returnedUser == null){  
        return false;  
    }  
    return true;  
}
```

# Logout

- Criar a action no controller: UserController:

```
[HttpGet]
public async Task<IActionResult> Logout(){
    await HttpContext.SignOutAsync();
    return RedirectToAction("index", "Home");
}
```

# Configurar middleware

- Ficheiro startUp.cs

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
    {
        options.LoginPath = "/UserView/UserLogin/";
    });
```

# Alterar as views

- Alterar o ficheiro: [\\_Layout.cshtml](#) (ou criar um [\\_LoginPartial.cshtml](#)) com informação sobre o estado de autenticação do utilizador:

```
<ul class="navbar-nav">
  <li class="nav-item">
    @if (User.Identity.IsAuthenticated){
      <a asp-controller="UserView" asp-action="Logout">Hello @User.Identity.Name
(Sign Out)</a>
    }else{
      <a asp-action="UserLogin">Login</a>
    }
  </li>
</ul>
```

# Limitar o acesso

- No `UserViewController`, colocar na action `getUsers` a seguinte anotação:

```
[Authorize]  
public IActionResult getUsers()
```

- Desta forma os utilizadores apenas podem aceder ao conteúdo da action se estiverem `autenticados`.

# Questões?

- Projeto completo na página:
  - <https://github.com/brunobmo/ASP.NET-Training/tree/master/2019/Workshop1>

# Introdução à framework .NET com C# e ASP

Bruno Oliveira