Introdução à framework .NET com C# e ASP

Bruno Oliveira

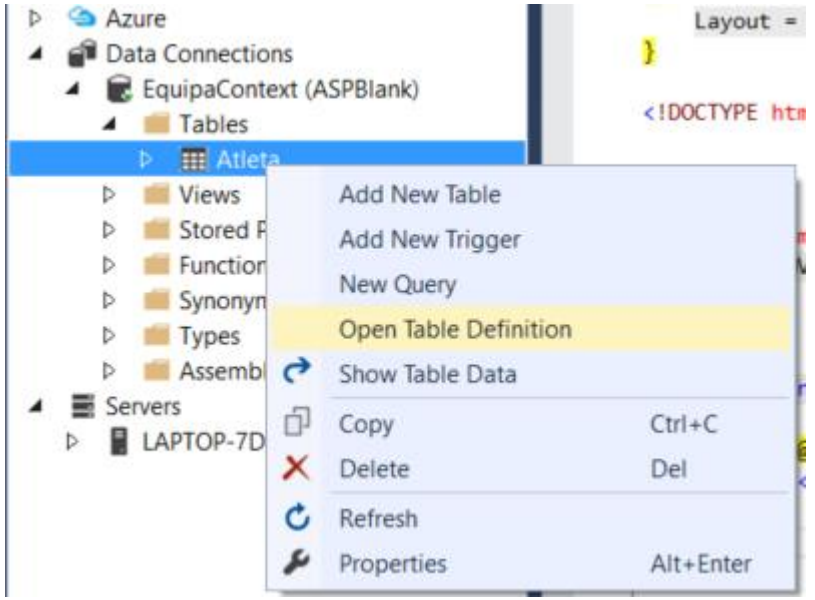
Tópicos

- Desenvolvimento WEB utilizando ASP.NET;
  - Autenticação e Autorização;
  - Layouts;

Página GitHub do Workshop:
https://github.com/brunobmo/ASP.NET-Training

Autenticação

- Antes de avançar para a autenticação, é necessário realizar algumas alterações no projeto desenvolvido anteriormente;
- Vamos alterar a estrutura da tabela Atleta:



- E acrescentar:
  - `username varchar(20)`
  - `password varchar(100)`

	Name	Data Type	Allow Nulls	Default
🔑	id_atleta	int	☐	
	nome	varchar(80)	☑	
	data_nascimento	date	☑	
	genero	char(1)	☑	
	username	varchar(20)	☑	
	password	varchar(100)	☑	
			☐	

Autenticação

- Vamos agora analisar o `model` Atleta: Experimente alterar o nome de alguma propriedade;
- A aplicação continua a funcionar normalmente!
- Este comportamento é normal, uma vez que o `model` Atleta criado anteriormente não está a ser utilizado;
- Se reparar, uma classe Atleta foi criada na raiz do projeto: É esta a classe utilizada pela Entity Framework (EF);
- Vamos alterar o `model` e classes correspondentes de forma a que o `model` implementado anteriormente seja utilizado pela EF;

Autenticação

- Alterar o `model` Atleta para:

```
[Table("Atleta")]
public class Atleta
{
    [Key]
    public int id_atleta { get; set; }

    [StringLength(80)]
    public string nome { get; set; }

    [Column(TypeName = "date")]
    public DateTime? data_nascimento { get; set; }

    [StringLength(1)]
    public string genero { get; set; }
    (...)
}
```

Annotations são utilizadas para validação dos modelos. São também interpretadas por outras aplicações .Net como é o caso da EF

Documentação adicional:

https://gist.github.com/brunobmo/a807f62ac6c56ff413edc208ae6421e0 (GitGist)

https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/database-first-development/enhancing-data-validation

https://msdn.microsoft.com/en-us/library/jj591583%28v=vs.113%29.aspx?f=255&MSPPError=-2147217396

Autenticação

- Elimine a classe Atleta da raiz do projeto e altere a classe: EquipaContext, importando o `model` criado anteriormente:

```
using Models;
```

- Na `view` verAtletas, adicione os campos anteriormente adicionados:

```
<th>
    @Html.DisplayNameFor(model => model.username)
</th>
<th>
    @Html.DisplayNameFor(model => model.password)
</th>
(...)
```

```
(...)
<td>
    @Html.DisplayFor(modelItem => item.username)
</td>
<td>
    @Html.DisplayFor(modelItem => item.password)
</td>
```

- Na `view` verAtletas.cshtml, alterar o objeto @model para:

```
@model IEnumerable<ASPBlank.Models.Atleta>
```

- Atualizar a `view` dos dados para o username e password

Documentação adicional:

https://gist.github.com/brunobmo/6147b51c2cde64ded754a256d3b115fe
https://gist.github.com/brunobmo/ee2e714f2353752d93695316ce2932cd

Autenticação

- Altere o `controller` Atleta de forma a incluir os campos adicionados:

```
public ActionResult AdicionarAtleta([Bind(Include = "nome,data_nascimento,genero, username, password")] Atleta atleta)
```

- Adicionar um novo `Controller` com o nome Login e uma vista para a `action` Index;

```
public class LoginController : Controller
{
        private EquipaContext db = new EquipaContext();

        public ActionResult Index()
        {
            return View();
        }
}
```

Documentação adicional:

https://gist.github.com/brunobmo/29d25ef26de690cef26eab377eedcd37 (GitGist)

Autenticação

- Adicionar um formulário de Login na `View`

```
@using (Html.BeginForm("Login", "Login", FormMethod.Post))
{
      @Html.ValidationSummary(true)
      <fieldset>
          Username: <input type="text" name="username" required />
          Password: <input type="text" name="password" required />
       </fieldset>
        <input type="submit" value="Submit" />
}
```

Documentação adicional:

https://gist.github.com/brunobmo/fe4efbc5228b9d1643e01bb8038a15b6 (GitGist)

Autenticação

- Adicione também um novo link na `view` correspondente ao `controller` Home -> Index Action;
- Adicionar uma `action`: Login ao `Controller` Login:

```
[HttpPost]
public void Login(string username, string password){
  if (ModelState.IsValid){
      var atletas = (from m in db.Atleta
                     where m.username == username && m.password == password
                     select m);
      if (atletas.ToList<Atleta>().Count > 0){
            Atleta atleta = atletas.ToList<Atleta>().ElementAt<Atleta>(0);
            FormsAuthentication.SetAuthCookie(atleta.username, false);
       }else{
            ModelState.AddModelError("", "Login data is incorrect!");
       }
  }
  (...)
}
```

O ASP.NET suporta diversos modos de autenticação. O modo *forms* permite a submissão de credenciais utilizando formulários

Permite adicionar erros ao dicionário de erros do modelo

Documentação adicional:

https://www.syncfusion.com/faq/1207/which-are-the-different-asp-net-authentication-modes

https://gist.github.com/brunobmo/6ff957b684a527c77a995302b7735995 (GitGist)

Autenticação

- Alterar a `view` (`LoginController`) para apresentar o formulário ou para apresentar uma mensagem de boas-vindas:

```
@if (!Request.IsAuthenticated)
{
    using (Html.BeginForm("Login", "Login", FormMethod.Post))
    {
        @Html.ValidationSummary(true);
        <fieldset>
            Username: <input type="text" name="username" required />
            Password: <input type="text" name="password" required />
         </fieldset>
          <input type="submit" value="Submit" />
    }
}
else{
    <span>Hello</span> <strong>@Html.Encode(User.Identity.Name)</strong>
    @Html.ActionLink("Sign Out", "Logout", "Login")
}
```

Documentação adicional:

https://gist.github.com/brunobmo/fd9844862eaa1425fc825eeafe0c55e2 (GitGist)

Autenticação

- Adicione a action para o logout:

```
public ActionResult Logout()
{
    FormsAuthentication.SignOut();
    return RedirectToAction("Index", "Login");
}
```

- De seguida, é necessário configurar a aplicação para utilizar a autenticação por formulários. Editar o ficheiro na raiz do projeto com o nome web.config e acrescentar:

```
(...)
<system.web>
    <compilation debug="true" targetFramework="4.5.2" />
    <httpRuntime targetFramework="4.5.2" />
    <authentication mode="Forms" />
 </system.web>
(...)
```

Documentação adicional:

https://gist.github.com/brunobmo/a9c8964c63c1cf7957d0212091e6727b (GitGist)
https://gist.github.com/brunobmo/b48b06e5d60ca4d5b43aca72a5e3e9cf (GitGist)

Autenticação

- Adicionar uma `action` para sinalizar o sucesso do login:

```
public ActionResult loginSucess()
{
    return RedirectToAction("Index", "Login");
}
```

- Alterar o `controller` Login na `action` Login de forma a redirecionar após sucesso no login:

```
return RedirectToAction("Index", "Login");
```

Documentação adicional:

https://gist.github.com/brunobmo/bfd1bca7e0426a5e0bf0a31d4fa4c31e (GitGist)

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;
    }
}
```

- 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!

- Modificar o método AdicionarAtleta do `Controller` Atleta:

```
[HttpPost]
public ActionResult AdicionarAtleta([Bind(Include = "nome,data_nascimento,genero, username, password")]
Atleta atleta)
{
    if (ModelState.IsValid)
    {
      atleta.password = MyHelpers.HashPassword(atleta.password);
      db.Atleta.Add(atleta);
      db.SaveChanges();
    }
    return RedirectToAction("sucessOperation");
}
```

Documentação adicional:

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

Encriptar!

- Modificar a `action` Login para que a password inserida seja encriptada e comparada:

```
(...)
Atleta atleta = atletas.ToList<Atleta>().ElementAt<Atleta>(0);
using (MD5 md5Hash = MD5.Create()){
    if (MyHelpers.VerifyMd5Hash(md5Hash, password, atleta.password)){
         FormsAuthentication.SetAuthCookie(atleta.username, false);
    }else{
          ModelState.AddModelError("", "Password incorreta!");
    }
}
(...)
```

- Alterar a query:

```
var atletas = (from m in db.Atleta
               where m.username == username
               select m);
```

Documentação adicional:

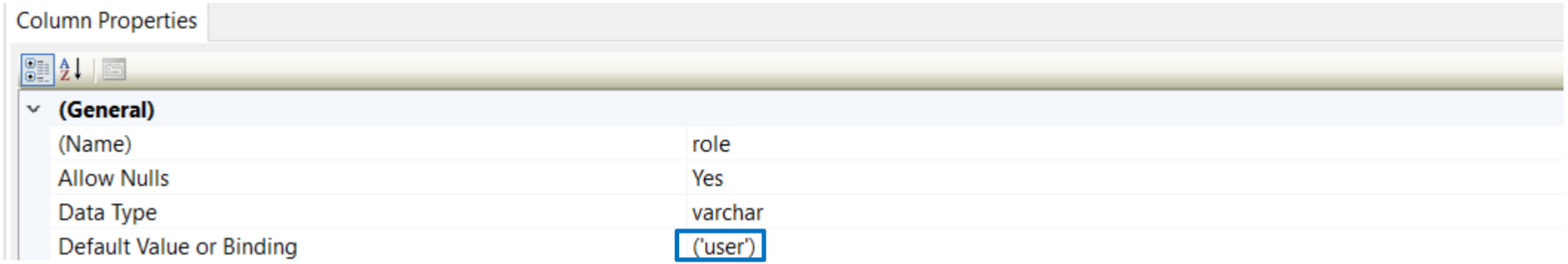
https://gist.github.com/brunobmo/75f2fd6854814b7d69a3875ae0e275c2 (GitGist)

https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396#Anchor_7

Autorização baseada no perfil

- Modificar a tabela Atleta para adicionar um campo: role

12	12	Bruno	2018-04-25	M	bmo	202cb962ac59075b964b07152d234b70	user
13	13	Bruno	2018-04-24	M	bmo2	202cb962ac59075b964b07152d234b70	user,admin



- Modificar o `model` para incluir o novo campo:

```
[StringLength(60)]
public string role { get; set; }
```

Autorização baseada no perfil

- Este tipo de autorização é declarativo, permitindo embutir Código nos `controllers` que determinam que perfis de utilizadores podem aceder a um recurso específico;
- Criar um novo `controller`: Admin:

```
[Authorize(Roles = "admin")]
public class AdminController : Controller
{
    public string Index()
    {
        return "Administrador";
    }
}
```

Documentação adicional:

https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-2.1

https://gist.github.com/brunobmo/9aa0c27dd75603b05f5db347a5fb49d2 (GitGist)

Autorização baseada no perfil

- Criar um novo método na classe MyHelpers:

```
public static HttpCookie CreateAuthorizeTicket(string id_Atleta, string roles)
{
    FormsAuthenticationTicket authTicket = new FormsAuthenticationTicket(
        1, // versão
        id_Atleta,  // nome
        DateTime.Now, // data de criação
        DateTime.Now.AddMinutes(30),  // validade
        false,   // se o cookie fica persistente ou não
        roles); //dados do utilizador

        HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsCookieName,
FormsAuthentication.Encrypt(authTicket));
        return cookie;
}
```

Esta classe é utilizada para criar um objeto que representa um ticket de autenticação que é utilizado pela autenticação baseada em formulários para identificar e autenticar o utilizador

Documentação adicional:

https://gist.github.com/brunobmo/031fff125c2b2ad5557062aa2e2acbda (GistGist)

https://msdn.microsoft.com/en-us/library/system.web.security.formsauthenticationticket%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396

https://support.microsoft.com/en-us/help/910443/understanding-the-forms-authentication-ticket-and-cookie

Autorização baseada no perfil

- Alterar no `Controller`: LoginController, a linha:

```
FormsAuthentication.SetAuthCookie(atleta.username, false);
```

- Para:

```
HttpCookie cookie = MyHelpers.CreateAuthorizeTicket(atleta.id_atleta.ToString(), atleta.role);
Response.Cookies.Add(cookie);
```

Autorização baseada no perfil

- Acrescentar o seguinte método no ficheiro: Global.asax

```
protected void Application_AuthenticateRequest(Object sender, EventArgs e){
    if (HttpContext.Current.User == null) return;
        if (!HttpContext.Current.User.Identity.IsAuthenticated) return;
            if (HttpContext.Current.User.Identity is FormsIdentity)
            {
                var id = (FormsIdentity)HttpContext.Current.User.Identity;
                FormsAuthenticationTicket ticket = id.Ticket;
                string userData = ticket.UserData;
                string[] roles = userData.Split(',');
                HttpContext.Current.User = new GenericPrincipal(id, roles);
            }
        }
    }
}
```

Documentação adicional:

https://gist.github.com/brunobmo/0431b65f7733912559efb4e845eb4d4b (GitGist)

Acesso à base de dados com base no perfil do utilizador

- Adicionar um novo utilizador:

```
CREATE LOGIN Bruno WITH PASSWORD = '123456';
CREATE USER BrunoUser FOR LOGIN Bruno;
GRANT SELECT ON BDEquipa.dbo.Atleta TO BrunoUser;
```

- No visual studio criar uma nova ligação em que o tipo de autenticação é: SQL Server authentication;
- Indicar os dados do novo tipo de utilizador criado;
- Adicionar uma nova Data Connection com o novo utilizador;
- Criar um novo context (UtilizadorLimitadoContext) para a acesso à base de dados utilizando a EF.
  - Clique direito sobre o projeto -> add -> New Item -> Data -> ADO.NET Entity Data Model -> Code First from database -> não selecionar nenhum objeto na janela seguinte;

Acesso à base de dados com base no perfil do utilizador

- Acrescentar o seguinte código ao novo context criado:

```
public virtual DbSet<Atleta> Atleta { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
        modelBuilder.Entity<Atleta>()
        .Property(e => e.nome)
        .IsUnicode(false);

         modelBuilder.Entity<Atleta>()
         .Property(e => e.genero)
         .IsFixedLength()
         .IsUnicode(false);
}
```

Documentação adicional:

https://gist.github.com/brunobmo/a8e821ea791b84821ccd640bcdeed65b GitGist

Acesso à base de dados com base no perfil do utilizador

- Para testar, replicar o adicionar atleta do `controller` Atleta e a respetiva `view`;
- Utilizar diferentes contextos para acesso à base de dados;

```
[Authorize(Roles = "admin")]
    public class AdminController : Controller
    {
        EquipaContext db = new EquipaContext();
        public ActionResult Index()
        {
            return View();
        }

(...)
```

```
 public class AtletaController : Controller{
     utilizadorLimitado db = new utilizadorLimitado();
     public ActionResult Index(){
         return View();
     }

(...)
```

- Testar e controlar os erros:

```
try{
    db.SaveChanges();
}
catch (DbUpdateException ex) {
    Console.Write(ex.ToString());
}
```

Layouts

- Para utilizar `views` em diferentes `controllers` é necessário colocar a `view` na pasta: shared;
- Criar uma `view`: _sucessView na pasta shared com o mesmo conteúdo da `view` criada anteriormente para sinalizar o sucesso do Login;
- Eliminar as `views`: sucessOperation;
- Substituir a linha: `return View();` por `return View("_sucessView");` na `action` sucessOperation do `controllers`: Atleta;
- Adicionar a view: _sucessView, aplicando-a para a sinalização de sucesso de inserção de atleta e também para o Login;

```
@ViewBag.mensagem
@{string returnControler = @ViewBag.controller;
@using (Html.BeginForm("Index", returnControler, FormMethod.Post)){
   <button> Voltar </button>
}
```

Layouts

- Testar reutilizando a `view`:
  - Criar uma `action` de sucesso no `controller` Login:

```
public ActionResult sucessAction(){
    ViewBag.title = "Sucesso";
    ViewBag.mensagem = "Login realizado com sucesso";
    ViewBag.controller = “Home";
    return View("_sucessView");
}
```

  - Na `action` Login, modificar a parte final do método:

```
return RedirectToAction("sucessAction");
```

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 (GitGist na parte inferior do slide);
- O ficheiro _ViewStart.cshtml define o layout para as várias `views` da pasta em que está contido;
- Para aplicar o estilo podemos também modificar a propriedade layout do Razor incluída em cada página:

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

Documentação adicional:

https://gist.github.com/brunobmo/e245e44014aa81306729c32cc4fedcac (GitGist)

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>
```

- Caso não tenho os pacotes de Jquery, Modernizr e bootstrap (Twitter bootstrap) adicionados ao projeto: Botão direito do rato-> Manage NuGet packages e na nova janela instalar as duas bibliotecas;

Layouts

- Adicionar o ficheiro: Site.css na pasta content:

```css
body {
    padding-top: 50px;
    padding-bottom: 20px;
}

/* Set padding to keep content from hitting the edges */
.body-content {
    padding-left: 15px;
    padding-right: 15px;
}

/* Set width on the form input elements since they're 100% wide by default */
input,
select,
textarea {
    max-width: 280px;
}
```

Documentação adicional:

https://gist.github.com/brunobmo/22dc10460bdad95868a29af94601ef14 (GitGist)

Layouts

- Alterar a folha de estilos (CSS): Site.css na pasta Content:

```
.mycontainer{
    margin:20px;
}
```

- Incluir a classe para a `action` Index do `Controller` Home:

```
<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>
```

Documentação adicional:

https://gist.github.com/brunobmo/bdb529ea4295a39ae8f0d868bd6a6620

Layouts

- Alterar a folha de estilos (CSS): Site.css na pasta Content:

```
.mycontainer{
    margin:20px;
}
```

- Incluir a class para a `action` Index do `Controller` Atleta:

```
<div class="mycontainer addForm">
        @using (Html.BeginForm("AdicionarAtleta", "Atleta", FormMethod.Post))
        {
            @Html.ValidationSummary(true)
            <fieldset>
                <label for="nome">Nome:</label>
                <input type="text" name="nome" id="nome" required />
                <label for="data_nascimento"> Data nascimento:</label>
                <input type="date" name="data_nascimento" id="data_nascimento" required />
                (...)
        }
</div>
```

Documentação adicional:

https://gist.github.com/brunobmo/ef61abb0607def79a63c5d776024ba88

Layouts

- Acrescentar no CSS:

```
.addForm label {
    display:block;
}

.addForm form{
    margin:auto;
    width:30%;
}
.addForm input, select {
    margin-top:10px;
    width:100%;
```

- Testar com um dispositivo móvel e adicionar ao CSS:

```
@media only screen and (max-width: 600px) {
    .addForm form{
       width:100%;
    }
}
```

Documentação adicional:

https://gist.github.com/brunobmo/56cd05b7136d2e32db9058b30b0b0aa0 (GitGist)

Layouts

- Utilizar Bootstrap:

```
<input type="submit" value="Submit" class="btn btn-primary btn-lg"/>
```

Questões?

- Projeto completo na página:
  - https://github.com/brunobmo/ASP.NET-Training

Introdução à framework .NET com C# e ASP

Bruno Oliveira