

Introdução à framework .NET com C# e ASP

Bruno Oliveira

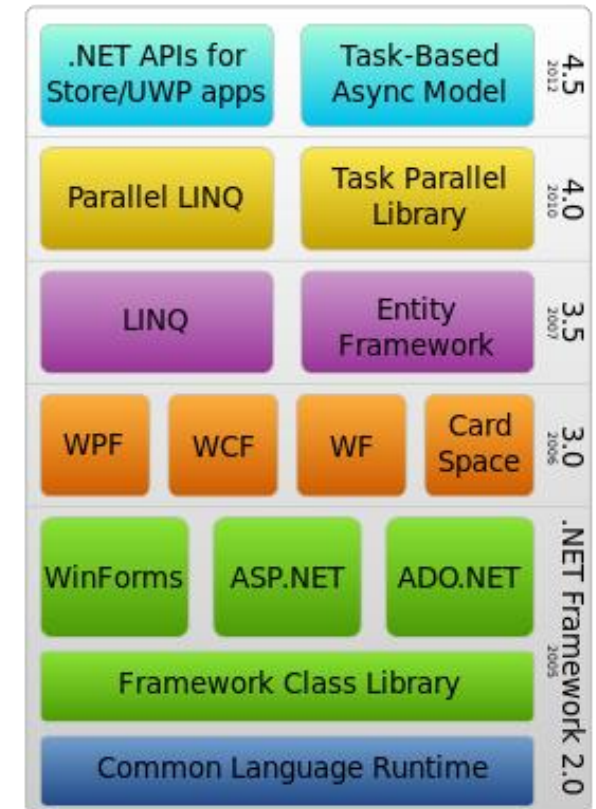
Tópicos

- Introdução à framework .NET;
- Desenvolvimento WEB utilizando ASP.NET;
 - Introdução a SQL Server;
 - Inserção e leitura de dados;

Framework .NET

- .NET Framework é um ambiente de execução que fornece uma **biblioteca** de classes abrangente;
- permite aos programadores desenvolver aplicações robustas com código confiável para todas as principais áreas de desenvolvimento;

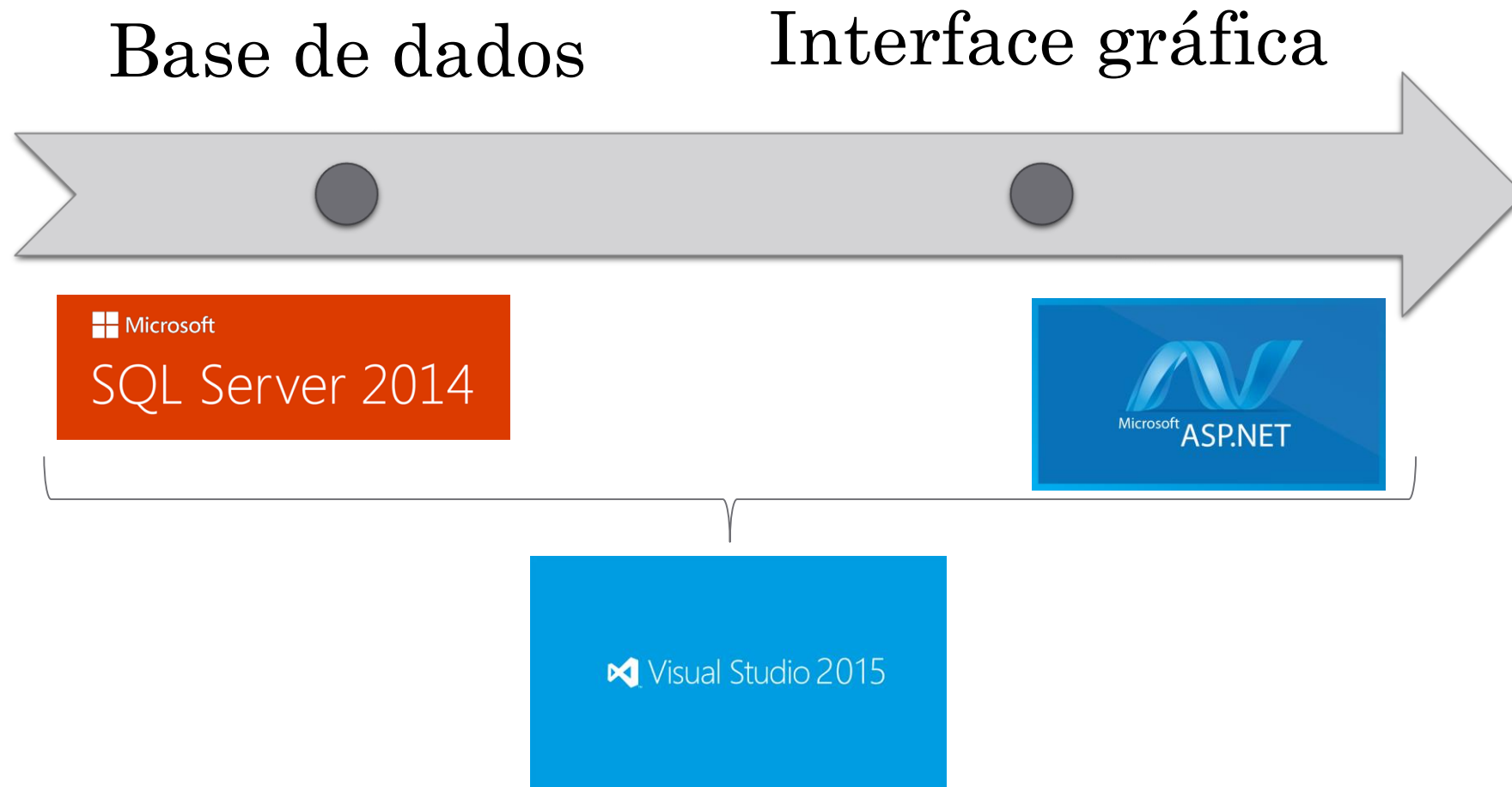
Download: <https://www.microsoft.com/pt-pt/download/details.aspx?id=30653>



SQL Server – C# – ASP.NET

- O **SQL Server** é um SGBD da Microsoft, muito utilizado a nível empresarial que permite a gestão de dados de suporte a toda a organização;
- O **C#** ("C sharp") é uma linguagem de programação criada para o desenvolvimento de aplicações que executam sobre a Framework .NET.
- C# é uma linguagem poderosa, tipada e orientada a objetos;
- **ASP.NET** é uma framework para a construção de aplicações web e serviços;
- Com **ASP.NET** é possível criar web sites baseados em **HTML5**, **CSS** e **JavaScript** de forma simples e rápida.

Principais passos no desenvolvimento da aplicação

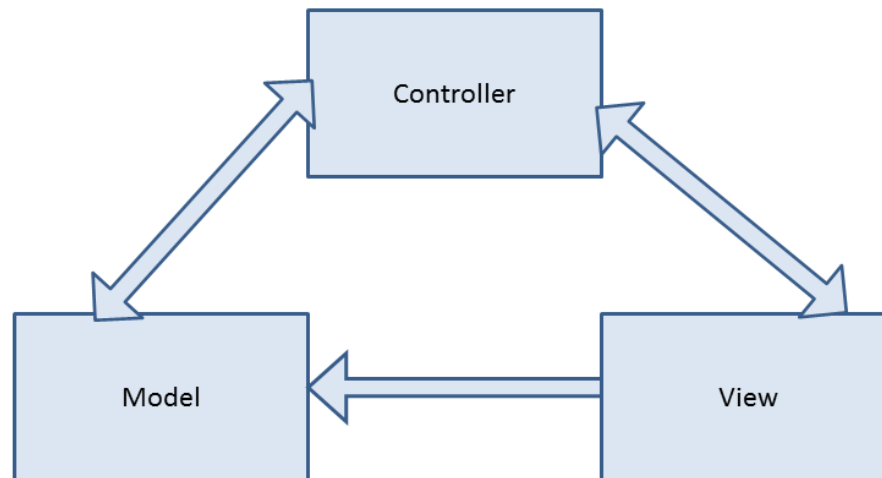


ASP.NET

- Dois modelos de desenvolvimento:
 - **Web Forms**: Permite a construção de websites dinâmicos utilizando uma interface *drag-and-drop* baseado num modelo orientado a eventos a eventos; Permite a criação de aplicações de forma rápida e simples;
 - **MVC**: A framework **MVC** (MVC5) da Microsoft aplica o padrão MVC (Model-View-Controller) sobre o ASP.NET, permitindo o desenvolvimento de aplicações web que promovem a reutilização, organização e desempenho do código;

MVC – Model View Controller

- É um padrão de arquitetura que separa a aplicação em três áreas distintas:
 - **Model**: Representa os dados que a aplicação utiliza;
 - **View**: Representa a interface gráfica com que o utilizador interage;
 - **Controller**: Representa a lógica aplicacional que relaciona as três áreas distintas;



MVC – Model View Controller

- Algumas *práticas* devem ser seguidas para tirar partido das *vantagens* da utilização do padrão MVC:
 - O *model* deverá ser um objeto com propriedades de *escrita* e *leitura* para suportar uma única *view*;
 - A *lógica* da *view* deverá ser *restringida* à interação do utilizador e *não* deve incluir lógica de negócio;
 - Os *controllers* devem ser *independentes* em relação à forma como os dados do *model* são manipulados;
 - Os *controllers* devem ser independentes em relação à forma como os dados são *armazenados* para além do *model*;

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

MVC e .NET

- Os *Model Binders* gerem os dados desde que são introduzidos pelo utilizador (na *view*) e como são transmitidos ao *model*;
- Exemplo de um tipo simples:

```
public string AdicionarAtleta(string nome, DateTime dataNascimento, char genero) (...)
```

<http://localhost:49431/Atleta/AdicionarAtleta?nome=Bruno&dataNascimento=1987-08-15&genero=m>



- Exemplo tipo complexo:

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

Documentação adicional:

<http://www.tutorialsteacher.com/mvc/model-binding-in-asp.net-mvc>

MVC e .NET

- São também utilizados **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:

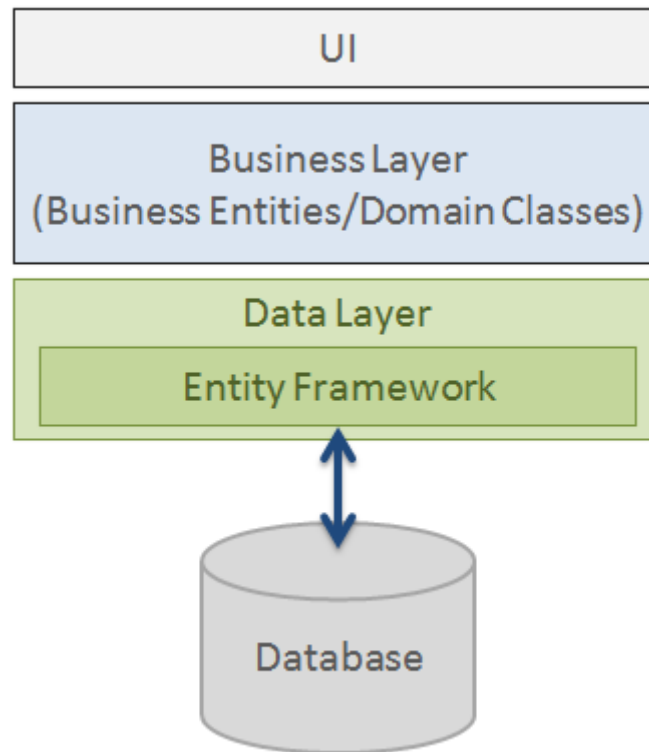
```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
</head>
<body>
  <div>
    @Html.ActionLink("adicionar Atleta", "Index", "Atleta" );
    @Html.ActionLink("ver atletas", "verAtletas", "Atleta");
  </div>
</body>
</html>
```

Documentação adicional:

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

Entity Framework

- A Entity Framework é um ORM suportado pela Microsoft que permite simplificar a interação dos programadores com a base de dados utilizando objetos .NET;




© EntityFrameworkTutorial.net

Caso de estudo

- Pré-requisitos:
 - SQL Server Express Edition 2014 ou superior com Management Studio (SSMS);
 - Visual Studio Community Edition 2015 ou superior com Web Tools (ASP.NET Core) e SSDT;
- Links para download:
 - <https://www.visualstudio.com/pt-br/vs/features/modern-web-tooling/>
 - <https://www.microsoft.com/pt-pt/sql-server/sql-server-downloads>
 - <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms>

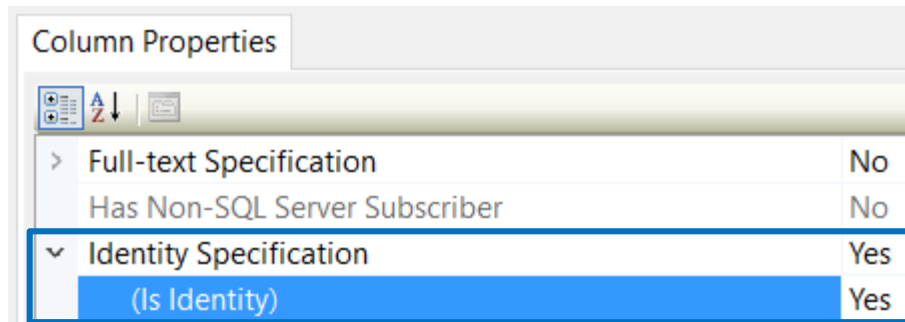
Base de dados

- Registrar **atletas** de uma equipa:
 - Número interno de registo
 - Nome completo
 - Data de nascimento
 - Genero (M ou F)
- Passo 1: iniciar o SQL server Management Studio e criar a base de dados com a tabela **Atleta**:

	Column Name	Data Type	Allow Nulls
	id_atleta	int	<input type="checkbox"/>
	nome	varchar(80)	<input checked="" type="checkbox"/>
	data_nascimento	date	<input checked="" type="checkbox"/>
	genero	char(1)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Base de dados

- Considerações:
 - `Id_atleta`: auto incremental (propriedade `identity`)



Base de datos

- Genero: check constraint:

Check Constraints

Selected Check Constraint:

generoValidation

Editing properties for existing check constraint.

(General)

Expression	genero IN ('M', 'F')
------------	----------------------

Identity

(Name)	generoValidation
Description	

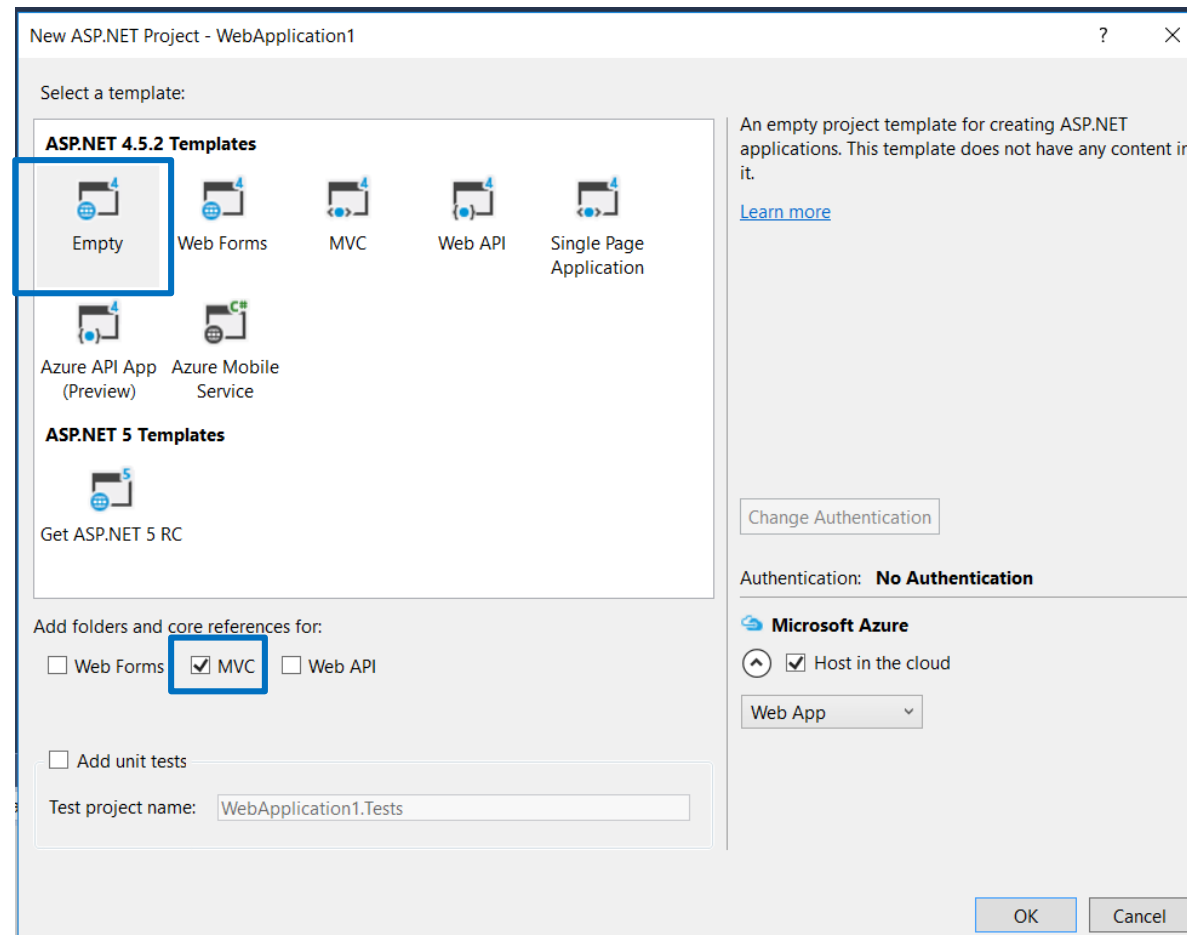
Table Designer

Check Existing Data On Cr	Yes
Enforce For INSERTs And L	Yes
Enforce For Replication	Yes

Add Delete Close

Projeto Visual Studio

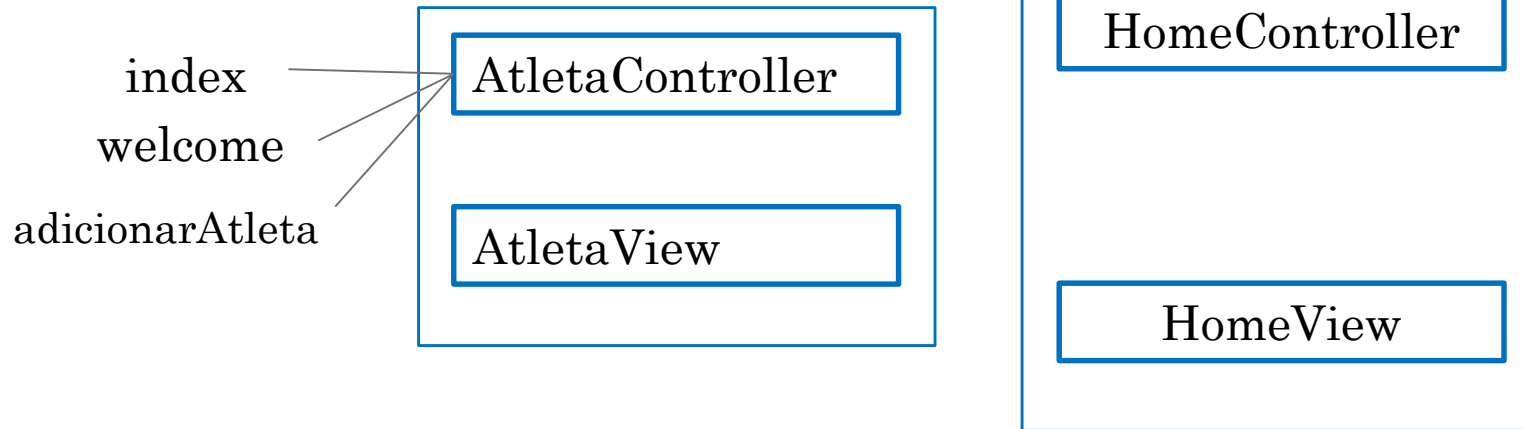
- Criar um **projeto** sem conteúdo (empty) baseado em MVC;



Contexto

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

- Objetivo 1:



Atleta Controller

- Criar um **controller** de forma a receber os **pedidos** do *browser*, devolver os dados do **model** e especificar a **view** que retorna a **resposta** para o *browser*;
- Criar: **AtletaController**;

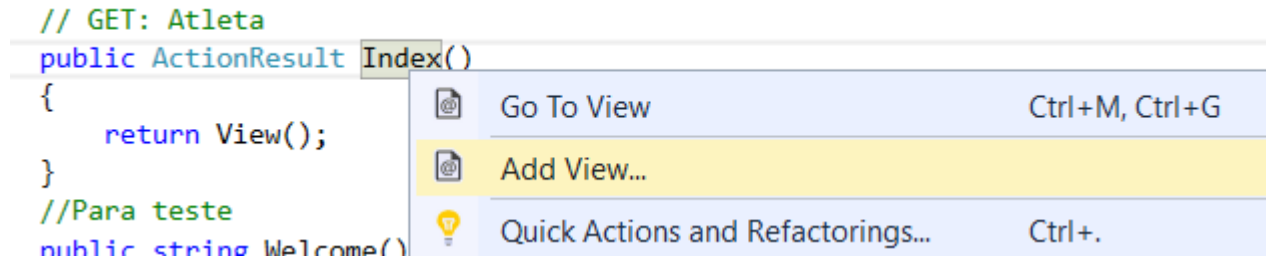
```
public class AtletaController : Controller
{
    // Método por defeito
    public ActionResult Index()
    {
        return View();
    }
    //Para teste
    public string Welcome()
    {
        return "This is the Welcome action method...";
    }
}
```

Documentação adicional:

<https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-controller>
<https://gist.github.com/brunobmo/c9998c82ea06f933f9f5d7418c264983> (GitGist)

Atleta View

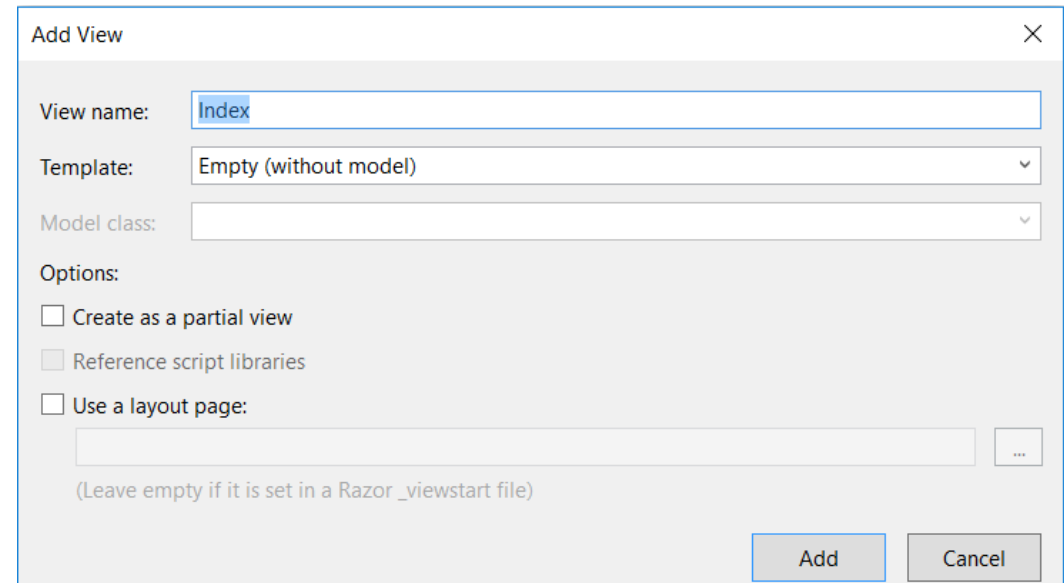
- Adicionar a **view**:



- Testar como:
- <http://localhost:49431/atleta/Index>

Ou

- <http://localhost:49431/atleta/>



Rotas

- Adicionar uma página que será apresentada na rota por defeito: /;
- Repetir o processo anterior para o **controller**: home;
- As rotas são definidas em: App_Start/RouteConfig:

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        );
    }
}
```

Controller Action

<http://localhost:xxxx/Atleta/Welcome>

Invocar o Atleta controller

- Passar parâmetros para o **controller** de forma a **adicionar** um novo Atleta!
- Por método **GET** utilizando **queryStrings**:

```
public string AdicionarAtleta(string nome, DateTime dataNascimento, char genero)
{
    return HttpUtility.HtmlEncode("Nome " + nome + ", dataNascimento: " + dataNascimento + ", genero: " + genero);
}
```

```
http://localhost:49431/Atleta/AdicionarAtleta?nome=Bruno&dataNascimento=1987-08-15&genero=m
```

- Utilizando **rotas**: (o terceiro parâmetro representa o parâmetro: ID)

```
public string AdicionarAtleta(string nome, DateTime dataNascimento, char género, int ID = 1)
{
    return HttpUtility.HtmlEncode("Nome " + nome + ", dataNascimento: " + dataNascimento + ", genero: " + genero);
}
```

```
http://localhost:49431/Atleta/AdicionarAtleta/1?nome=Bruno&dataNascimento=1987-08-15&genero=m
```

Objeto ViewBag

- De seguida, é necessário gerar uma resposta **dinâmica**, enviando dados do **controller** para a **view** para a geração da **resposta**;
- O objeto: **ViewBag** é utilizado para que o **controller** possa enviar dados para a **view**;
- O objeto **ViewBag** permite definir as **propriedades** desejadas para suportar o envio de dados;
- Modificar **HomeController**:

```
public ActionResult Index()  
{  
    ViewBag.title = "Equipa APP";  
    ViewBag.mensagem = "Página das equipas";  
    return View();  
}
```

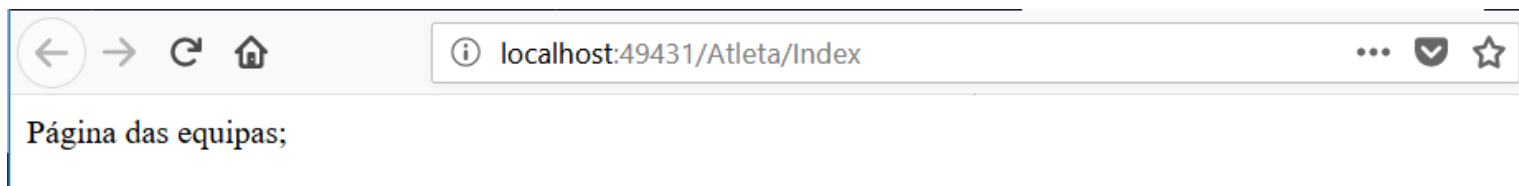
Recursos:

<https://gist.github.com/brunobmo/b2416e02317b0f7bd13b535191c0c961> (GitGist)

Objeto ViewBag

- Alterar a [view](#):

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>@ViewBag.Title</title>
</head>
<body>
  <div>
    @ViewBag.mensagem
  </div>
</body>
</html>
```



Recursos:

<https://gist.github.com/brunobmo/15200b1e2e989142f78294fe4c20b0c3> (GitGist)

Criar um formulário: View do Atleta

- Modificar a **view** associada ao índice do **controller** do **Atleta**:

HTML helper

```
<div>
  @using (Html.BeginForm("AdicionarAtleta", "Atleta", FormMethod.Post))
  {
    @Html.ValidationSummary(true)
    <fieldset>
      Nome: <input type="text" name="nome" required />
      Data nascimento: <input type="date" name="data_nascimento" required />
      <select name="genero">
        <option value="M">Masculino</option>
        <option value="F">Feminino</option>
      </select>
    </fieldset>
    <input type="submit" value="Submit" />
  }
</div>
```

Mostra uma lista das mensagens associadas à validação do formulário

Recursos:

<https://gist.github.com/brunobmo/de547cd86fc6a70700ee47579bf53f5a> (GitGist)

Criar um formulário: Controller do Atleta

- Modificar o `Controller` do `Atleta`, adicionando:

```
[HttpPost]
public string AdicionarAtleta(string nome, DateTime dataNascimento, char genero)
{
    return HttpUtility.HtmlEncode("Nome " + nome + ", dataNascimento: " + dataNascimento + ", genero: " +
genero);
}
```

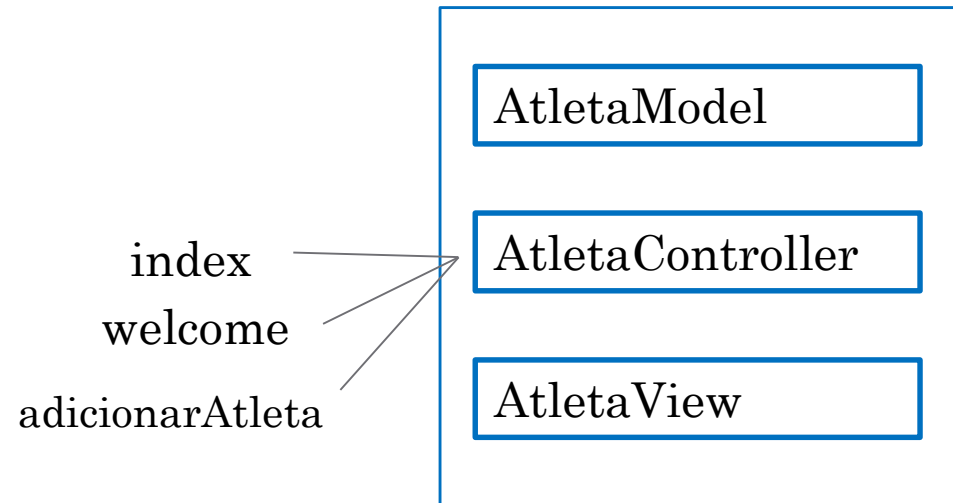
- Os seletores `[HttpGet]` e `[HttpPost]` definem o tipo de pedido que deverá ser aceite pela ação a executar;

Recursos:

<https://gist.github.com/brunobmo/1cf9b6c830b7ef175c774d3e54f39560> (GitGist)

Contexto

- Objetivo 2:



Model para o Atleta

- Adicionar um **model**:
 - Adicionar uma classe: **Atleta** na pasta **Models**;

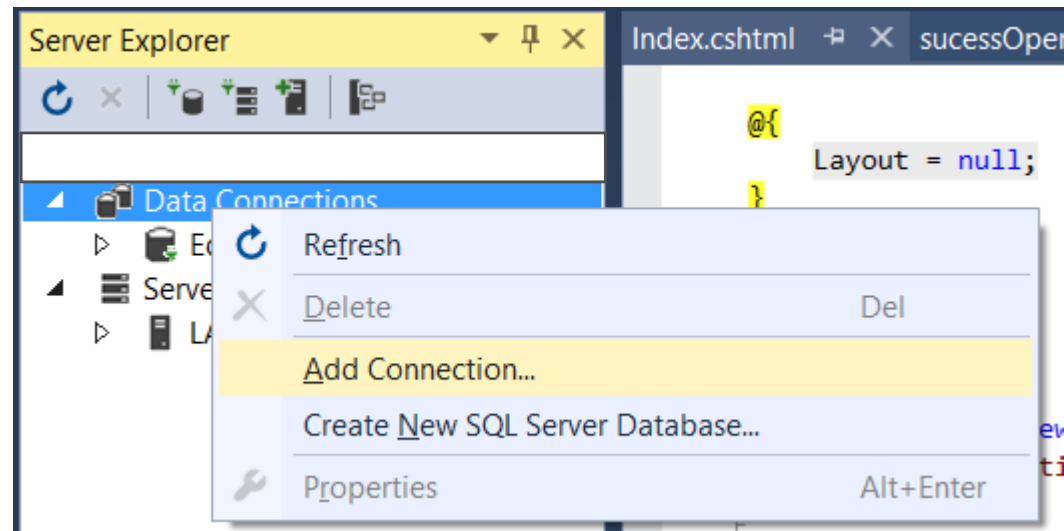
```
public class Atleta
{
    public string nome { get; set; }
    public DateTime data_nascimento { get; set; }
    public string genero { get; set; }
}
```

Recursos:

<https://gist.github.com/brunobmo/1f6c6e72711ebe040df4a7e4819dac75> (GitGist)

Ligação Base de Dados

1. (No Visual Studio) Adicionar a [ligação](#) para a base de dados (separador: [server explorer](#)):



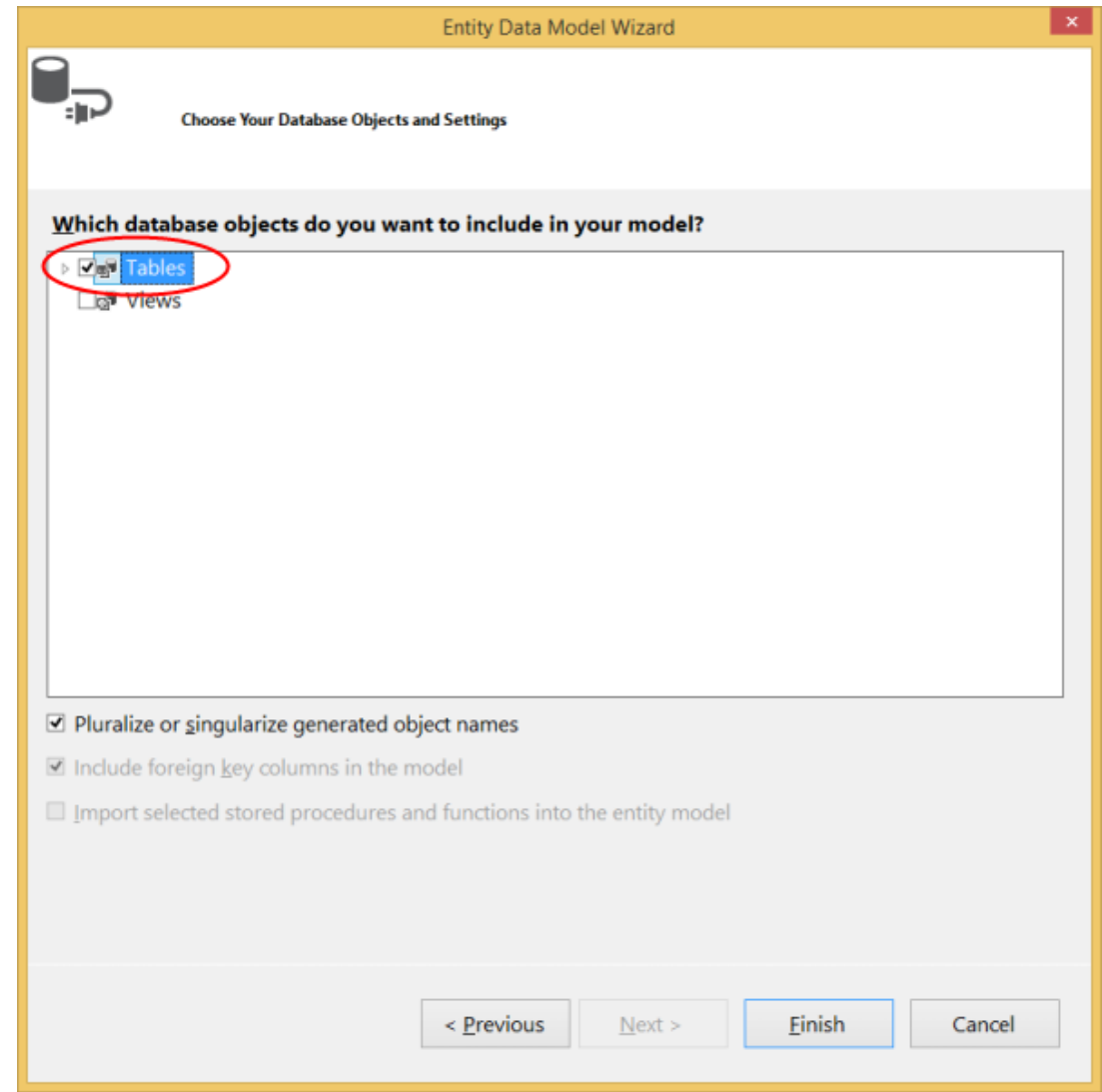
Ligação Base de Dados

- Utilizar a **Entity Framework** para comunicação com a base de dados:
 - Project -> Add New Item...
 - Selecionar: **ADO.NET Entity Data Model**
 - Introduza: **EquipaContext** como nome;
 - De seguida, seleccionar: **Code First from Database**;
 - Selecionar a ligação à base de dados

O ficheiro: **App.config** foi atualizada com os dados da ligação;

Mais Informação:

<https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/creating-a-connection-string>



Alterar o Model do Atleta

- Modificar o **Model: Atleta**, adicionando a class **EquipaContext**:

```
namespace ASPBlank.Models
{
    public class Atleta
    {
        public string nome { get; set; }
        public DateTime data_nascimento { get; set; }
        public string genero { get; set; }
    }

    public class EquipaContext : DbContext
    {
        public DbSet<Atleta> Atleta { get; set; }
    }
}
```

Recursos:

<https://gist.github.com/brunobmo/9dc15658ccf65822b93c0d1428baa10a> (GitGist)

Alterar o Controller do Atleta

- Modificar o método: `adicionarAtleta` do `Controller` Atleta

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

A framework ASP.NET MVC valida os dados enviados para o *controller*, povoando o objeto *ModelState* com os erros encontrados

```
public ActionResult sucessOperation()
{
    ViewBag.title = "Atleta adicionado com sucesso";
    ViewBag.mensagem = "Atleta inserido com sucesso";
    return View();
}
```

Recursos:

<https://gist.github.com/brunobmo/1ed4d05976ef20c3ec23212a7ddd5527> (GitGist)

<https://exceptionnotfound.net/asp-net-mvc-demystified-modelstate/>

Ligar as várias partes do website

- Criar a página `Index` da vista `Home` (e respetivo `Controller`):

```
<div>
    @Html.ActionLink("adicionar Atleta", "Index", "Atleta" );
    @Html.ActionLink("ver atletas", "verAtletas", "Atleta");
</div>
```

- Adicionar no `controller` do `atleta`:

```
public ActionResult verAtletas()
{
    var atletas = (from m in db.Atleta
                    select m);
    return View(atletas);
}
```

← LINQ query

Recursos:

<https://gist.github.com/brunobmo/0e0b27943231b39ba78028f6fb15aee3> (GitGist)
<https://gist.github.com/brunobmo/7523b4e8814c699d223cdac0d4093bd4> (GitGist)
<https://msdn.microsoft.com/en-us/library/bb308959.aspx>

Ligar as várias partes do website

- Adicionar a `view`: `verAtletas`

```
@model IEnumerable<ASPBlank.Atleta>
(...)
    <table class="table">
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.nome)
            </th>
            (...)
        </tr>
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.nome)
                </td>
                (...)
            </tr>
        }
    </table>
(...)
```

Recursos:

<https://gist.github.com/brunobmo/2d48274aa29a0ec80b3482e656611001> (GitGist)
[https://msdn.microsoft.com/en-us/library/system.collections.ienumerable\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.110).aspx)

Questões?

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

Introdução à framework .NET com C# e ASP

Bruno Oliveira