PWEB – Resumo (+/- simplificado)

[View – Ficha 1 1](#_Toc1726805086)

[Model – Ficha 1 1](#_Toc1898231890)

[Controller – Ficha 1 1](#_Toc1314185502)

[Get – Ficha 3 1](#_Toc313799219)

[Post – Ficha 3 2](#_Toc2015394245)

[DataBase – Ficha 2 2](#_Toc1745939586)

[Migration 2](#_Toc1154872613)

[Update 2](#_Toc994620320)

[ViewModel – Ficha 3 2](#_Toc1535718442)

[ViewData 3](#_Toc2138327587)

[ViewBag 3](#_Toc1954963318)

[TempData 3](#_Toc1258518379)

[IdentityUser [ApplicationUser] - Ficha 5 3](#_Toc724185699)

[Roles – Ficha 6 3](#_Toc2054215666)

[Relações entre models – Ficha 3 4](#_Toc886192126)

[1 para N 4](#_Toc543415266)

[1 para 1 4](#_Toc179148043)

[OnModelCreating (1-to-1 relation) - Ficha x (ainda não encontrada ou se calhar não será feita) 4](#_Toc2007106838)

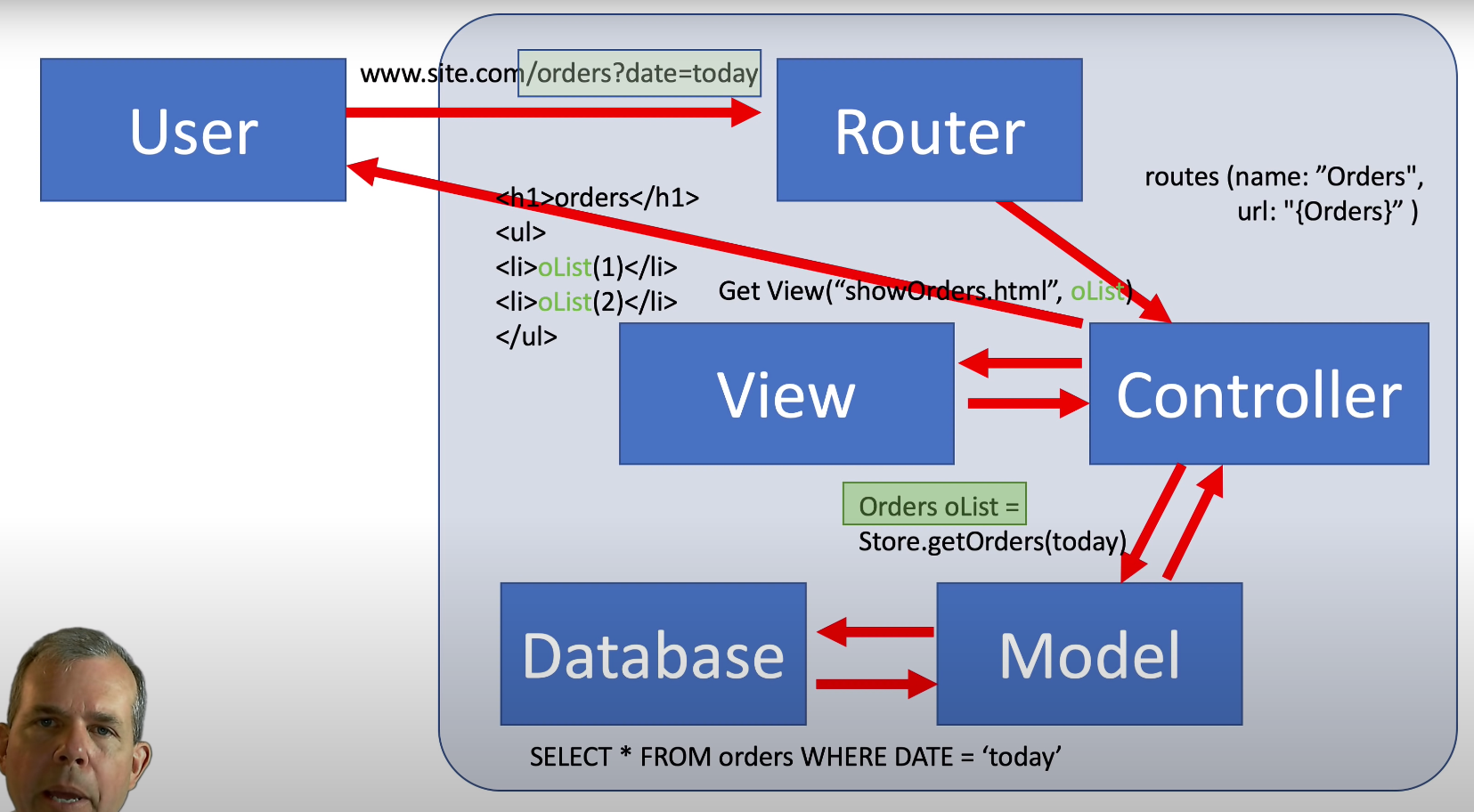
[Visual Studio 5](#_Toc1354206855)

[Criação de View 5](#_Toc323153037)

[Criação do Controller 5](#_Toc1394369619)

[Visualização da base de dados 5](#_Toc214025652)

[Links uteis 5](#_Toc1928769018)



# View – Ficha 1

Ficheiro *cshtml* (html com script/funções do C#) onde o programa vai mostar no browser. Nela podes importar class’s Model ou ViewModel para mostar as informações da base de dados.

Funciona como um html normal e o css funciona igual (uso recomendado das fichas o boostrap para um melhor estilo em pouco tempo)

# Model – Ficha 1

Class’s C# com o propósito de criar as tabelas SQL para a base de dados, com isso não têm funçoes internas e todos os seus atributos são *public* e com a função *get* e *set* caso seja indicas à frente dos atributos.

A class cria uma tabela SQL em que cada coluna corresponde ao atributos da class.

Caso o tipo do atributo tiver um ‘?’ depois quer dizer que o valor pode ser omitido (*nullable*), com isso caso já exita registo na base de dados nessa tabela/class/model não é preciso fazer *Drop-DataBase* e perder todos registo para poder atualizar as colunas.

Ex.: public int Id { get; set; }

# Controller – Ficha 1

Class principal de comunicação e gestão entre as páginas e as suas informações.

## Get – Ficha 3

Função que vai ter a funcionalidade de mostrar a *view* que é dedicada.

Usada, normalmente, para organizar os dados que

## Post – Ficha 3

Função que vai ter a funcinualide de recolher os dados de um formulário ou input vindo do browser para poder fazer a sua lógica. A forma para destiguir a função *post* com *get* é com a tag **[HttpPost].** Os paramentros de entrada é a class desejada com a função Bind antes par construir a class de entrada com os atributos recebido pelo forms.

Usada para registar algo na base de dados ou até mesmo para filtrar para puder ser usar para mostrar.

Ex.:

Pesquisa – no input é inserido o texto que o utilizador deseja ver e o a função recebe a informação para saber qual itens/dados deve mostrar.

Registar – no form é inserido as informações que o forms pede para assim a função receber e criar o novo registo desse objeto, assim não sendo preciso abrir a base de dados para adicionar registo manualmente. Algum dos paramentos podem ser atributos automaticamente, logo facilitando ainda mais a criação do registo.

# DataBase – Ficha 2

Ficheiro SQL criado fora do projeto na pasta do Utilizador do sistema operativo.

Os atributos dos modelos, com o nome certo ou com um tipo de palvra nela, ficam os atributos automaticos.

Ex.:   
Model ‘Cliente’  
 ClienteId → Id → Colulna Key da tabela

CursoId → Id → Coluna Foregin Key da tabela com a tabela ‘Curso’

## Migration

Ficheiros C# onde fizem qual informações que a base de dados vai ter.

A cada alteração dos modelos, deve ser feito o comando *Add-Migration <nome da migração>* assim criado o ficheiro onde atualizará a base de dados. O contido, analisado, pode verificar que tem código SQL de criação e destruição de tabelas, assim com as *Keys* e *ForeginKeys* das tabelas.

Caso o conteúdo esteja “vazio”, significa que não foi feito alguma alteração ou não foi identificado as alterações.

Com esta mecânica, facilita a criação e atualização da base dados consoante o desenvolvimento do aplicativo

## Update

Quando a(s) migração(ões) existem, o comando *Update-Database* “pega” nos ficheiros de migração e cria a base dados no ficheiro gerado automaticamente.

# ViewModel – Ficha 3

Similar aos modelos, mas com a designação de listar os modelos que quer ser mostrado. A principal propriedade é a lista de modelos que a ViewModel é designada, assim podendo agrupar todos os dados/modelos numa zona e assim a view que vai “pegar” nesse ViewModel ter só os modelos desejado pelo utilizador.

Estas classes são usadas nos forms assim podendo ajudar/facilitar o input dos dados de filtragem.

E também contem os valore que foram usados para fazer essa filtração.

Ex.: Pesquisa – percorrer os dados todos com o atributo de pesquisa desejado e adicionar à lista do ViewModel os dados que vai ser mostrado.

## ViewData

* ViewData é um objeto ViewDataDictionary acessível por meio de chaves do tipo string.
* Os dados do tipo string podem ser armazenados e usados diretamente sem a necessidade de serem convertidos.
* Outro tipo de dados/objetos necessitam de conversão para o tipo especifico antes de serem utilizados.

## ViewBag

* ViewBag é um Objeto que fornece acesso dinâmico aos objetos armazenados no ViewData.
* Pode ser mais conveniente para trabalhar, já que não requer casting.

## TempData

* Similar ao ViewData mas com as seguintes importantes diferenças:
  + ViewData torna-se nulo se ocorreu um redirecionamento.
  + TempData é usado para passar dados entre duas solicitações consecutivas.
  + ViewData persiste apenas durante a solicitação atual
  + TempData persiste na a solicitação atual e numa subsequente

# IdentityUser [ApplicationUser] - Ficha 5

Class principal do registo de utilizadores.

Nela existe o parâmetro de username, email, password e entre outros.

Para criar uma class de Utilizador ou *ApplicationUser*, como é feito nas aulas/fichas, é usado a class base *IdentityUser* para ter os dados principais e as propriedades adicionar desejadas.

Nota: o Id desta class não é um *int* mas sim uma *string*.

# Roles – Ficha 6

A tag **[Authorize]** diz respeito ao tipo de role do user. Caso a tag estaja no formato simples ([Authorize]) simplesmente verifica se o utilizador está logado, se sim executa a função, caso contrário mostra a pagina de login.

Para identificar o tipo de role tem de por [**Authorize(Roles = "NOME\_DA\_ROLE")]** assim podendo ser identificada. Existe 3 tipos de “configurações” desta tag

* Pertence **à** role:

[**Authorize(Roles = "NOME\_DA\_ROLE")]**

* Pertence a **uma** das roles (ou):

[**Authorize(Roles = "NOME\_DA\_ROLE1, NOME\_DA\_ROLE1")]**

* Pertence a **todas** as rolas (e):

[**Authorize(Roles = "NOME\_DA\_ROLE1")]**

[**Authorize(Roles = "NOME\_DA\_ROLE2")]**

Uma melhor explicação de uso deste mecanimos, recomando fazer ou ler a ficha 6 onde ela explica como usar isto.

# Relações entre models – Ficha 3

## 1 para N

Relação base entre os models. Tens 4 tipos de conveção:

1. Incluir na entidade dependente uma propriedade de navegação (referência) para a entidade principal
2. Incluir na entidade principal uma propriedade de navegação (coleção)
3. Incluir na entidade dependente uma propriedade de navegação (referência) para a entidade principal e incluir na entidade principal uma propriedade de navegação (coleção)
4. Definir a relação na sua totalidade em ambas as entidades, indicando, além as propriedades de navegação, a chave estrangeira na entidade dependente

## 1 para 1

É parecido como a 4ª conveção com a diferença que os dois modelos ficam o Id e a propriedade do outro modelo dentro dela. Mas com isto, a criação/atualização da base de dados fica confusa com as foreginKeys.

# OnModelCreating (1-to-1 relation) - Ficha x (ainda não encontrada ou se calhar não será feita)

Função do *ApplicationDbContext* onde podemos identificar as configurações das tabelas quando são para ser criadas/atualizadas.

Ex.:

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<**MODEL\_COM\_RELAÇÃO\_1\_PARA\_1**>()

.HasOne(c => c.**PROPRIEDADE\_COM\_OUTRO\_MODEL**)

.WithOne(a => a.**PROPRIEDADE\_COM\_ESTE\_MODEL**)

.HasForeignKey<**MODEL\_COM\_RELAÇÃO\_1\_PARA\_1** >(c => c.**ID\_DO\_OUTRO\_MODELO**);

base.OnModelCreating(modelBuilder);

}

# Visual Studio

## Criação de View

Com o Visual Studio, a criação das views é mais rápido e fácil. Simplesmente clica com o botão direito e clica na opção de view. Podes indicar o tipo de view (uma vazia ou uma com template/formato pré-definido).

Nos templates escolhes o tipo de template desejado e o model desejado, assim criando a view com todos as propriedades do model.

## Criação do Controller

Com o Visual Studio, a criação dos controllers é mais rápido e fácil. Simplesmente clica com o botão direito e clica na opção de controller. Podes indicar o tipo de controller (um vazio, um com leitura e escrita e um com criação automática das view, no sistema no **CRUD** – Create, Read, Update and Delete).

Na janela do **CRUD** selecionas o modelo desejado assim criado todas as views necessárias e o código base para o funcionamento.

Nota: Caso o modelo contenha uma relação com outro model, o registo fica invalido pois não é especifica na leitura dos inputs do utilizador.

## Visualização da base de dados

Com o Visual Studio, a visualização e edição da base de dados (na parte de registo) é mais fácil. Abre a feramenta de Explorador de Servidor e abres o ficheiro SQL que existe na pasta, como foi mencionado anteriormente. Nela, abre a pasta de Tables e com o botão direito na tabela que deseja ver seleciona a opção *Show Table*.

# Links uteis

[Crash Course básico e base da matéria](https://www.youtube.com/watch?v=BfEjDD8mWYg&t=1111s)

[ASP.NET Docs](https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0)