TI 2

.NET Core e ASP.NET Core

.NET - História

- A versão 1.0 da .NET Framework foi introduzida em 2002, estando disponível apenas para Windows.
- Projetos como o Mono tinham como objetivo trazer a .NET Framework a macOS e Linux (mas não **era** suportado oficialmente pela Microsoft).
- Em 2016, a Microsoft introduziu o **.NET Core**, uma versão **open-source** do .NET, com o objetivo de ser **multi-plataforma**.
- A versão 4.8 da .NET Framework é a última versão que vai receber novas funcionalidades (só vai receber bug fixes).

.NET Core

- .NET Core uma a versão multi-plataforma da .NET Framework da Microsoft.
- A vasta maioria das classes, funções e interfaces da .net Framework são iguais no .NET Core.
  - Exceções: Coisas específicas a Windows (e são cada vez menos com cada versão)
- Tal como a .NET Framework, as dependências são instaladas com o NuGet.
- Versão atual: 2.2
  - Versão 3 em *preview*, com melhorias ao nível de IoT, Machine Learning, e aplicações *desktop*.

.NET Core - Instalação

- Se se tem o Visual Studio >= 2017, não é preciso fazer nada (excepto certificar que está atualizado).
- Em Linux e macOS: Instalar o .NET Core SDK (ver instalações sobre como instalar no site)
  - Pode-se instalar também o .NET Core SDK em Windows, mesmo que não se tenha o Visual Studio instalado.
- Para Visual Studio Code, seguir este artigo, e opcionalmente ver este vídeo para fazer o set up.
- Para macOS, pode-se usar o Visual Studio for Mac.

ASP.NET Core

ASP.NET Core

- O ASP.NET Core é a framework da Microsoft para desenvolver aplicações web.
- 95% dos conceitos são iguais ao ASP.NET Framework (MVC):
  - Controllers, Views, Models, Entity Framework, Identity, Model Binding, Validação, … (Podem existir algumas diferenças nestes componentes, mas são poucas)
- A principal diferença é que foi concebido para ser multi-plataforma (com o .NET Core).

ASP.NET Core e ASP.NET Framework - Diferenças

Funcionalidade	**ASP.NET Core**	**ASP.NET Framework**
Web Forms (ASPX)	Ausente, substituído por Razor Web Pages	Presente, opcional.
Web.config	Substituído pelo ficheiro appsettings.json, entre outros	Obrigatório
Configuração no arranque	Main() e classe Startup	Global.asax, App_Start
Guardar ficheiros de forma privada	Tudo o que não está na pasta wwwroot é privado	Os ficheiros privados devem ir para o App_Data
Hosting e deployment	Opcional, pode-se usar Apache, nginx, ou o programa diretamente	Obrigatório usar o IIS

ASP.NET Core e ASP.NET Framework - Diferenças

Funcionalidade	**ASP.NET Core**	**ASP.NET Framework**
Controllers	Presente quando se usa MVC	Presente quando se usa MVC
API Controllers	Usam-se os mesmos controllers que os de MVC, unificados	É recomendado usar o ASP.NET Web API, mas é limitada
Ficheiros	Usam a interface IFormFile	Usam a classe HttpPostedFileBase
Uso de BDs e outros serviços	Recomenda-se usar Dependency Injection (DI)	DI limitada, objetos instanciados diretamente (new).
Sessões	Desligadas por defeito, têm que ser ligadas no Startup	Ligadas por defeito

Diferença principal (ou, dica para o Google)

- Se se está a fazer uma pesquisa específica ao .NET Core, ou a alguma das seguintes funcionalidades:
  - ASP.NET Core
  - Entity Framework Core (ou EF Core)
  - ASP.NET Identity Core
  - .NET Core
- Deve-se meter “Core” na pesquisa (como se vê na lista acima):
  - “ef core migrations”
  - “asp.net core configure identity”
- Caso contrário, poderão aparecer resultados para o ASP.NET Framework, que poderão ser incompatíveis.

ASP.NET Identity Core e Identity - Diferenças

Funcionalidade	**Identity Core**	**Identity**
Identity no template com autenticação	Escondida por defeito, mas pode-se fazer o scaffold dos ficheiros	Todas as classes e views estão disponíveis no projeto quando este é criado
Classe do User personalizada	Criar uma classe que extende IdentityUser e mudar a classe da base de dados para a referenciar. Poderão ser necessárias migrações para refletir as alterações.	A classe é criada no projeto por defeito.

EF Core vs. Entity Framework - Diferenças

Funcionalidade	**EF Core**	**Entity Framework**
Migrações	Apenas manuais, recomenda-se apagar a pasta das migrações e BD (se possível) e recriar a migração	Manuais e automáticas (não se recomenda usar migrações automáticas em produção)
Seed	Seed de dados possível com HasData, mas é limitado. Cenários mais complexos (ex: autenticação) ainda requerem classes dedicadas	Possível na classe de configurações das migrações.
Convenções (ex: ON DELETE CASCADE)	Tem que se desligar no método de configuração para cada relacionamento	Pode-se desligar removendo a convenção.

EF Core vs. Entity Framework - Diferenças

Funcionalidade	**EF Core**	**Entity Framework**
Carregar dados relacionados	Por defeito, é necessário usar o Include, caso contrário os valores das propriedades serão NULL ou arrays vazios. Alternativamente, configura-se para ser automático. Recomendo o Include se se sabe os dados necessários *a priori*, pode resultar em queries SQL mais otimizadas.	Feito automaticamente por defeito, com “classes proxy”.
Atualizar uma entidade (o SaveChanges é igual)	db.Tabela.Update(entidade);	db.Entry(entidade).State = EntityState.Modified;

EF Core vs. Entity Framework - Diferenças

Funcionalidade	**EF Core**	**Entity Framework**
Relacionamentos M-N	É obrigatório criar a classe do relacionamento, com as chaves primárias e forasteiras.	A classe do relacionamento é opcional, desde que não sejam necessários mais campos que as chaves primárias e forasteiras.
Chaves primárias compostas	É necessário usar o HasKey no método de configuração (ver segundo snippet da secção “Fluent API”).	Usam-se os atributos [Key] e [Column(Order = n)] nas propriedades afetadas.
Usar outras bases de dados (depois de instalar os NuGet packages necessários)	Configurado no Startup e appsettings.json (mudar a Connection String).	Configurado no Web.config.

Criação de projetos

Criação de projetos sem Visual Studio

- No .NET Core, é possível usar a linha de comandos para criar projetos.
- Para isso, usa-se o programa “dotnet”.
- O “dotnet” é usado também para:
  - Scaffolding
  - Migrações
  - Executar aplicações
  - Correr *scripts*

Ferramenta “dotnet” - coisas úteis

Comando	Descrição
`dotnet [comando] --help`	Mostra a ajuda do [comando].
`dotnet new -l`	Mostra os “templates” disponíveis (ex: mvc, console, …).
`dotnet new mvc -o ProjetoMvc`	Cria um projeto MVC na pasta ProjetoMvc (convém usar o “cd” para entrar na pasta do projeto).
`dotnet restore`	Restaura os NuGet packages do projeto da pasta atual.
`dotnet build`	Compila o projeto da pasta atual.
`dotnet run`	Executa o projeto da pasta atual.
`dotnet watch run`	Automaticamente reinicia o projeto quando existem alterações.

Ou seja, para criar e iniciar um projeto...

```
# Criar projeto com autenticação (dotnet new mvc --help para detalhes)
> dotnet new mvc --auth Individual -o ProjetoTeste

# Ir para a pasta do projeto
> cd ProjetoTeste

# Executar o projeto
> dotnet run

# Abrir o browser em http://localhost:5000 (poderá sair erro de https).
# Pode-se abrir o ficheiro .csproj no Visual Studio, ou a pasta no Visual Studio Code.
```

Ferramenta “dotnet” - (mais) coisas úteis

Comando	Descrição
`dotnet add package [nome]`	Instala o package NuGet com o nome [nome] (“Manage NuGet packages for solution” continua disponível)
`dotnet ef migrations add [Nome]`	Adiciona uma migração (“Add-Migration” e “Enable-Migrations” continuam disponíveis no Visual Studio).
`dotnet ef database update`	Aplica as migrações na base de dados. (“Update-Database” continua disponível no Visual Studio).
`dotnet ef migrations remove`	Desfaz a última migração criada e apaga-a.
`dotnet ef database update 0`	**Reverte todas as migrações.**
`dotnet publish -c Release -o [pasta]`	Compila a aplicação para ser colocada num servidor em produção. O output fica na pasta [pasta].