Normal 23/24

1. a) Indique, justificando detalhadamente, as partes programáticas constituintes de um website desenvolvido em ASP. Net Core 6.0, nomeadamente os Models, as Views e os Controllers, referindo a necessidade de cada um deles e o modo com interagem entre si.

> **Cada um deles é importante para a criação de um website em ASP.Net Core 6.0. De acordo com o padrão MVC (Model-View-Controller)**
>
> **As Views são interfaces de apresentação de dados ao utilizador, ou seja, a UI da aplicação. As Views nunca devem executar qualquer tipo de lógica ou interagir diretamente com a base de dados devendo apenas trabalhar com os dados que são fornecidos/carregados pelo seu próprio Controller.**
> **Os ficheiros das Views são divididos por pastas com o nome dos respetivos Controllers, ou seja, se tivermos um**
> ***AccountController.cs*****, iremos ter uma pasta de Views chamada Account em que dentro dessa pasta iremos ter as diferentes Views (Create, Delete, Details, Edit e Index) - estes são criados automaticamente.**
> **A nível de código/ficheiros, as Views, ficheiros do tipo *.cshtml,* normalmente são escritas em sintaxe Razor e permitem incorporar instruções C# diretamente em Tag Helpers.**
>
> **Os Controllers são os intermediários entre o utilizador e**

o código da aplicação propriamente dito. O Controller traduz os pedidos/ações do utilizador feitos na View em operações correspondentes de código. Embora a View tenha apenas o seu Controller específico, o Controller não é limitado apenas a uma View. Quando o utilizador pede algum tipo de dado, o Controller irá ao modelo de dados buscar e tratar a informação pedida. Temos um Controller para cada tipo de dados existente no Model. Nos Controllers, podemos definir Roles de Autorização de acesso e isto faz com que todo o acesso a esse Controller seja feito a quem tem direito (por exemplo: utilizador logado e com role de administrador). Dentro de cada Controller, temos definido os dados que queremos enviar para a View dependendo do pedido do utilizador (GET, POST, etc...). Esses dados são enviados para a View em forma de ViewModels, ViewData e ViewBag.

Os Models representam a estrutura dos dados da aplicação. Definem como os dados são organizados, são usados para interagirem com a base de dados da aplicação onde nele estão definidas as entidades e relações da base de dados. Os dados são requeridos pelos Controllers.

Um exemplo do fluxo de interatividade desta arquitetura pode ser:
Quando clicamos no botão "Editar" na página Habitações, a View que irá ser carregada será a respetiva à função Editar do seu Controller

> **(Habitações), que irá buscar os dados necessários ao Model.**

1. b) É indiferente a utilização do MVC ou de Razor Pages na construção de sites usando ASP.Net Core 6.0? Quando é que preferencialmente utilizaria um ou outro? Justifique, detalhadamente, a resposta que der!

> **A utilização do MVC ou de Razor Pages é indiferente porque com cada uma delas. conseguimos produzir o mesmo resultado, no entanto de maneiras e com objetivos diferentes.**
>
> **Devemos usar MVC quando queremos criar um website com alguma dimensão e complexidade isto porque, ao contrário das Razor Pages, com o MVC podemos separar a lógica e todos os diferentes tipos de Controllers enquanto que nas Razor Pages um ficheiro lida com a interface e com a sua respetiva lógica. Isto é vantajoso no caso de termos um website em que o utilizador manipula diferentes e interligadas informações. Enquanto que ao usarmos MVC, em apenas num Controller podemos definir vários tipos de ações, se usássemos Razor Pages teríamos que ter um ficheiro para cada ação.**
>
> **O facto de escolhermos usar Razor Pages poderia ser adequado para a criação de páginas estáticas que não necessitem necessariamente de uma estrutura de dados complexa e bastante relacionada entre si, como criar páginas de aviso ou de conteúdo que não seja**

> **propriamente dinâmico como texto e/ou imagens por exemplo.**

1. c) Qual o interesse e a necessidade de se utilizar ViewModels? Dê um exemplo de uma sua eventual utilização. Justifique, detalhadamente, a resposta que der!

> **Os ViewModels servem para quando queremos mostrar mais que um modelo/tabela de dados numa página ou, que tenham dados complexos e com muitas relações.**
> **Este modelo de dados é de tipagem forte, o que quer dizer que cada variável e constante presente no mesmo tem um tipo de dados explicitamente definido.**
> **Com este tipo de dados, podemos fazer representar na View apenas os dados necessários de um Model especifico (não apresentando diversos campos, por exemplo) e garantem uma maior segurança na atualização dos dados, isto porque os tipos de dados são verificados em tempo de compilação, garantindo a coerência entre o tipo de dados que o utilizador quer enviar corresponder aos que pode receber.**
> **Uma possível utilização seria: quero uma lista de habitações onde mostra todos os detalhes da mesma, o vendedor associado a ela, o inquilino atual e as suas avaliações. O facto de querer aceder a diferentes Models (Habitação, Cliente, Avaliações) torna esta a estrutura melhor para o fazer.**

2. a) Refira, justificando, o porquê de nalgumas Views se utilizar por convenção no nome das mesmas o caracter _ como prefixo? Justifique, detalhadamente, a resposta que der!

> **Quando algumas Views usam o caracter _ como prefixo do seu nome estamos perante o uso de Partial Views. Estas Views podem ser partilhadas, ou seja, pode ser usada através da pasta Shared.**
> **Esta View permite especificar o código Razor e Tag Helpers ou HTML Helpers num ficheiro para que depois possa ser reutilizado em várias vistas. Pode ser usado simplesmente para simplificar uma vista simples.**
> **Estas Partial Views, quando partilhadas, dependem sempre de alguma coisa, como por exemplo, numa barra de navegação, o botão de "Login" ou "Logout" depende do estado da autenticação. Outro exemplo, seria uma barra de tarefas ou algo semelhante que não faz sentido por si só ser renderizado sozinho. Para isto ser usado, devemos ao HTML.Partial (que é um HTML Helper) passando o nome da vista.**

2. b) Qual a função do Razor View Engine? Qual a sua relação com os Tag Helpers? Justifique, detalhadamente, a resposta que der!

> **O Razor View Engine é o mecanismo que renderiza as Views gerando HTML. Suporta a sintaxe Razor, que esta permite incorporar código C# com elementos que gerem código HTML. Estas Tag Helpers são um recurso com uma sintaxe semelhante ao HTML que após renderizadas pelo Razor no servidor, converte-as para HTML. Estas Tag's permitem que integremos instruções do tipo *asp* como por exemplo:** `<input type="text" asp-for="Nome"/>` **que resultaria em código HTML semelhante a:** `<input id="nome" name="nome" type="text" value="" />` **.**

2. c) Quando se define o MER para a implementação de um site usando o ASP.Net Core 6.0 pode-se implementar para uma mesma entidade mais do que um controller? E caso se vá utilizar Razor Pages para essa entidade, pode-se criar mais do que uma Razor Page? Justifique, detalhadamente, a resposta que der!

> **Quando se define o MER em com recurso ao MVC, por definição, cada entidade faz-se representar por 1 Model, 1 View e 1 Controller. Embora seja possível haver multiplos Controllers, não é o que é pretendido com a arquitetura MVC.**
> **Caso se utilize Razor Pages, uma entidade pode ter múltipas Razor Pages de forma a que cada uma possa ter funcionalidades bastante diferentes entre si. Da forma que existe uma ação descrita num Controller no modelo MVC, com Razor Pages, existe uma Razor Pages, Pages porque é constituída sempre por duas páginas (.cshtml e .cs), ou seja, uma ação em MVC representa uma Razor Pages.**

3. a) Quando se utiliza o Entity Framework Core 6.0, na construção de sites usando o ASP. Net Core 6.0, porque é que é necessário especificar uma classe, por exemplo com o nome TesteDBContext, que deriva da classe DbContext? Qual o conteúdo que se tem de incluir nessa classe? Justifique, detalhadamente, a resposta que der!

> **A Classe TesteDBContext irá ser criada com o propósito de definir a mesma como o contexto que representa o nosso modelo (tabela, por exemplo). Essa classe herda da DbContext porque queremos "importar" os métodos da classe herdada. Esta nova classe vai ter os DBSets e**

> **os relacionamentos entre modelos (1-1, 1-N, N-N). Posteriormente, para que estas relações de Bases de Dados funcionem, precisamos de registar o DBContext como um serviço, é preciso para que quando fazemos Add-Migration, crie um snapshot dos modelos existentes para que, no final, Update-Database, crie a base de dados (se esta não existir) e atualiza as suas respetivas tabelas (ou criá-las). Também podemos reverter as migrações, tendo em conta que pode resultar em perda de dados.**

3. b) Relativamente ao referido na alínea anterior, quando é que a classe em vez de derivar da classe DbContext deriva da classe IdentityDbContext? Indique, justificando detalhadamente, o porquê dessa derivação! Justifique, detalhadamente, a resposta que der!

> **Quando derivamos da classe IdentityDbContext, temos acesso aos métodos da classe Identity, métodos estes que servem para lidar com a autenticação e permissões dos utilizadores. Serve o propósito derivar desta classe para servir as características do ASP.NET Core Identity, que serve para a criação de diferentes tipos de Roles de Autorização de Acesso e Autenticação e isto faz com que todo o acesso seja feito a quem tem direito (por exemplo: utilizador logado e com role de administrador → `[Authorize(Roles="Admin, Gestor")]` / `[AllowAnonymous]` / `[Authorize(Users="joao")]`).**
> **Assim, temos uma classe com métodos de Base de Dados e de Identity.**

3. c) Ainda no que se refere à utilização do Entity Framework Core 6.0, na construção de sites usando o ASP.Net Core 6.0 e com uma base de dados em SQL Server, descreva o que é necessário fazer-se para a partir do MER especificado se poder utilizar a base de dados correspondente? Justifique, detalhadamente, a resposta que der!

   > **A pergunta refere-se a um dos Workflows possíveis de utilização de base de dados que é o Database-First. Este Workflow aplica-se muito a casos em que já exista uma base de dados minimamente complexa ou grande, em que este iria criar os modelos relativos à base de dados existente. Para isso, teríamos que primeiramente usar o comando `Scaffold-DbContext` (é o processo para criar modelos de uma base de dados já existente). Neste caso, as criações de migrações, `Add-Migration <nome>`, apenas são necessárias quando alteramos o modelo de dados e por consequente, teremos que usar usar o `Update-Database`.**

4. Comente, justificando e contextualizando, a seguinte afirmação: "Para que um site desenvolvido em ASP .Net Core 6.0, quer utilizando-se a abordagem clássica quer a abordagem Blazor, seja executado correctamente, é imprescindível que, quer no computador servidor quer no computador cliente onde é executado o browser que acede a esse site, esteja instalado exactamente o mesmo software de bases de dados bem como o mesmo software necessário para executar o C#!"

   > **A afirmação está errada. O cliente apenas necessita de um browser minimamente recente que interprete páginas HTML + CSS + JS se utilizando a abordagem clássica o código e WASH usando a abordagem Blazor . O servidor, apesar de dispor páginas com código C#, este é compilado do lado do servidor e, quando do lado**

> **do cliente, como os browsers não interpretam C# porque este também já fora pré-renderizado, é recebido como se fosse um executável.**