Exame de P. Web da E.R. de 2020/2021

1. Considerando a utilização de Delegates em C#:
   a) Selecione a ou as opções seguintes que estejam corretas! (7,5%)
      - São tipos que permitem fazer referência a métodos
      - ~~São uma melhor opção às funções~~
      - Podem ser usados para passar métodos a outros métodos
      - ~~Consoante os seus parâmetros, um delegate pode ser do tipo referência ou do tipo valor~~
      - Pode ser usado para encapsular um método com nome ou anónimo
      - ~~A sua opção é facultativa pois nem sempre se pode usar~~

   b) Justifique a escolha da ou das opções que selecionou! (10%)
      - São tipos que permitem fazer referência a métodos
        R: Um delegate é um tipo que permite fazer referência a métodos cuja utilização é feita através da adição de um parâmetro do tipo delegate no protótipo do método que a recebe e posterior passagem de um argumento do mesmo tipo. Dentro do método que a recebe o método para a qual o delegate referencia pode ser invocado através do método Delegate.Invoke()
      - Podem ser usados para passar métodos a outros métodos
        R: Como referido acima, é só isso que um delegate é, um tipo que referencia um método e que pode ser passado a um método que assim o permita (através da declaração de um parâmetro do tipo delegate no protótipo do método).
      - Pode ser usado para encapsular um método com nome ou anónimo
        R: Um delegate pode referenciar um método com nome, simplesmente usando uma variante do seguinte excerto de código:

```
public delegate void Del(string message);
public static void ExampleMethod(string message)
{
    // …
}
Del handler = ExampleMethod(string message);
```

        Para contrastar a situação mostrada acima, um delegate também pode referenciar um método anónimo, simplesmente usando uma variante do seguinte excerto de código:

```
Func<int, int> func = delegate (int a) { return a; };
```

2. "Para que um site desenvolvido em ASP.Net MVC 5.0 e que utilize bases de dados 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 SGBD bem como seja possível executar em ambos o C# e o Razor!" Comente esta afirmação, justificando o que referir! (10%)
   R: Esta afirmação é totalmente incorreta, ASP.Net MVC 5.0 é uma framework que tradicionalmente opera sob a forma de cliente-servidor, o que isto significa é que existem

recursos e ferramentas que só precisam de existir e correr no servidor. Entre estas ferramentas estão o SGBD (Sistema de Gestão de Bases de Dados), porque a base de dados da aplicação é um recurso que só deve existir no servidor (se existisse no cliente, deixaria de fazer sentido existir um servidor – também porque um Website é algo partilhado por vários utilizadores e os dados de cada utilizador só devem estar expostos ao próprio utilizador e ao servidor de modo a permitir que sejam feitas operações sobre eles através de pedidos do cliente), o C# também é uma ferramenta (linguagem de programação) que não necessita de poder correr nativamente no computador do cliente porque é vocacionado para o backend (é utilizado na construção do modelo de dados, controladores e afins do servidor), quando existe algum código que originalmente é escrito em C# e que precisa de correr no PC do cliente (como o browser não interpreta C# mas interpreta HTML, CSS, JavaScript) este é transformado em HTML e ocasionalmente JavaScript, o mesmo acontece para páginas Razor que são páginas escritas num Markup alterado (HTML misturado com C#) que para apresentação ao utilizador são (em runtime) transformados em HTML.

3. Considerando o desenvolvimento de sites utilizando-se ASP.Net MVC 5.0 e considerando que existem HTML Helpers sem "for" e com "for" refira o seguinte:
   a) Que principais diferenças existem entre estes dois "tipos" de HTML Helpers e quando se deve utilizar um ou outro e porquê? Detalhe a sua resposta e justifique o que referir. (10%)
   R: Os HTML Helpers com for servem para "automaticamente" construir HTML com base, por exemplo, num modelo (@model declarado no cimo da página), enquanto que os HTML Helpers sem for servem para "manualmente" (tecnicamente também automaticamente porque é para isso que os HTML Helpers servem – mas sem que seja feito com a inferência dos requisitos com base num modelo) construir HTML com base nos argumentos passados ao Helper.
   b) Descreva detalhada e justificadamente a relação existente entre as Views do modelo MVC e as páginas de sites desenvolvidos com a framework ASP.Net MVC 5.0. (10%)
   R: As Views do modelo MVC são páginas desenvolvidas em Razor cujo "controlo" é feito por Controladores do mesmo modelo, contêm código que não pode ser nativamente corrido por browsers modernos e por isso tem que ser traduzido para linguagens que assim o possam (HTML, CSS e JavaScript). Assim, o produto final (páginas de sites desenvolvidos com a framework ASP.Net MVC 5.0) são páginas "renderizadas" a partir das Views desenvolvidas em Razor (mistura de HTML e C#).
   c) "Sem HTML Helpers e o Razor, não seria possível desenvolver sites utilizando-se o framework ASP.Net MVC 5.0!" Comente esta afirmação, detalhando a sua resposta e justificando o que referir (7,5%)
   R: Esta afirmação não é correta, porque apesar de HTML Helpers e o Razor facilitarem (muito) a realização de algumas tarefas, é possível desenvolver sites utilizando ASP.Net MVC 5.0 sem a utilização das mesmas, simplesmente escrevendo HTML e C# puro.
4. Considerando a arquitetura mostrada na figura, responda às perguntas seguintes.
   a) Descreva, detalhando e justificando a sua resposta, a arquitectura mostrada na figura! (10%)
   R: A arquitetura mostrada na figura explica (em pouco pormenor) a relação entre a Aplicação, o LINQ e o servidor da base de dados. De forma sucinta, quando um programador pretende aceder a objetos da base de dados utilizando a framework Entity Core, pode, no seu código utilizar código LINQ (Language-Integrated Query – deve ser feito no controlador,

NUNCA na vista, e o programador pode optar por utilizar query syntax, que é um síntaxe mais aproximado de SQL, ou extension methods (método preferido atualmente) que aproxima o acesso a objetos da base de dados de programação orientada a objetos), independentemente do tipo de síntaxe LINQ que utilizar, a framework irá traduzir o código LINQ em SQL puro, aceder à base de dados, obter as o resultado do query (objetos da base de dados), transformá-la em uma coleção de instâncias do tipo de classes do modelo de dados e devolvê-la à aplicação, permitindo ainda operar sob esta coleção com LINQ porque a ferramenta funciona sob qualquer instância da classe "Enumerable" ou da interface "IQueryable" e a coleção devolvida será do tipo "IQueryable".

b) Considerando a utilização de LINQ, é indiferente utilizar um ou outro dos chamados Query Syntax ou Extension Methods? (10%)
R: Na maior parte dos casos, é possível obter os mesmos resultados finais utilizando os dois tipos de síntaxe mas isso não significa que seja indiferente utilizar um ou o outro, o Query Syntax foi o primeiro a surgir e como tal, é o mais desatualizado, desde que surgiram os Extension Methods, a Microsoft tem-se focado em lhes adicionar cada vez mais funcionalidades, mas não tem atualizado (muito) o Query Syntax. Como tal, os Extension Methods permitem fazer mais coisas, são mais flexíveis. Também são o método preferido dos developers a trabalhar com Entity porque nunca obrigam o programador a sair do mundo da programação orientada a objetos, enquanto que o Query Syntax obriga o developer a pensar mais proximamente ao SQL.

5. Considerando o seguinte contexto e a última instrução do seguinte trecho de código, responda às alíneas seguintes:

```
Aluno[] alunos = {
        new Aluno() { Numero = 1, Nome= "Maria Travassos", Idade= 18 },
        new Aluno() { Numero = 2, Nome = "Patrícia Augusta", Idade = 22 },
        new Aluno() { Numero = 3, Nome = "Diógenes Trancoso", Idade = 25 },
        new Aluno() { Numero = 4, Nome = "Teodoro Campos" , Idade = 20 },
        new Aluno() { Numero = 5, Nome = "Sandra Perestrelo" , Idade = 31 },
        new Aluno() { Numero = 6, Nome = "Tiago Loureiro", Idade = 17 },
        new Aluno() { Numero = 7, Nome = "Renato Neves", Idade = 19 },
};

var alunosF = alunos.Where( a => a.Nome.Split(' ').Last().StartsWith("T"));
```

a) Descreva detalhadamente o que essa instrução faz e descreva também cada uma das partes que a constituem. Justifique a sua resposta! (7,5%)
R: A instrução que contém a atribuição da variável alunosF dita que alunosF irá ser a coleção de alunos cujo último nome começa com T (Maria Travassos e Diógeneses Trancoso). Isto é porque na query dos alunos (através do extension method Where) procuramos por todos os alunos cujo último nome de entre os nomes que se obtêm dividindo o nome inteiro nos sítios onde se encontrem espaços começe com um T.

b) Qual é o tipo de alunosF presente na instrução? Quando é que este tipo deve ser usado? Da sua utilização decorrem algumas vantagens, quais e porquê? (10%)
R: O tipo de alunosF presente na instrução é IEnumerable<Aluno>, é o tipo-base de todas as coleções em C# e deve ser utilizado sempre que se queira guardar dados numa coleção às

quais mais tarde se possam querer aplicar queries LINQ. A principal vantagem da interface é exatamente esta, existem outras como por exemplo a possibilidade de enumerar a coleção com o forEach, entre outros.

c) O que significa e qual o papel de "=>" nessa instrução? Em que contexto é que pode ser utilizada? Justifique a sua resposta. (7,5%)
R: O "=>" é o chamado operador "Lambda" em C# e serve para construir "Lambda Expressions", um tipo de expressão anónima que permite fazer operações tal como se fosse um método mas com um síntaxe mais compacto e legível. Pode ser utilizado em todos os contexto em que se tem que passar um "Delegate" a um outro método e também em que se queira utilizar um método anónimo sem que se tenha que definir exatamente o tipo do mesmo. Para ser mais específico sobre o tipo de método que o .Where() do LINQ espera, é um tipo "Predicate<T>", um tipo de delegate no qual o resultado é um valor booleano (true/false) e o parâmetro corresponde ao tipo T da coleção Collection<T> à qual se aplique o .Where().