MediESTecaComIdentityeTestes

1. **Testes unitários com Identity**

   i. Para a criação dos testes unitários foi acrescentado à solução o novo projeto **MediESTecaTest** usando o template **xUnit Test Project (.Net Core) C#**. Neste projeto foi definida a classe de teste **HomeControllerTest** para testar inicialmente o controlador **Home**.

      a) Nota: É necessário adicionar uma referência ao projeto que se está a testar para que seja reconhecido o código no projeto de teste.

      b) Nota: Uma vez que a classe **HomeController** necessita de um objeto que implementa a interface **ILogger<HomeController>** para ser instanciada, poderá ser usado um *Mock* desse objeto. Sendo assim, usando o Nuget Package Manager Console será necessário instalar:
      install-package Moq

   ii. Foi criada a classe **UtentesControllerTest** para testar o controlador dos **Utentes**. Para instanciar no teste este controlador é necessário ter um **contexto** e um **UserManager** que são recebidos por dependency injection no construtor. Para a definição do contexto foi usada uma base de dados em memória, para o UserManager foi definido um objeto Mock.

      a) Nota: No caso do **contexto**, não é fácil nem aconselhável utilizar um objeto Mock pelo que se repetiu a opção de usar uma base de dados em memória. Foi necessário instalar no projeto de testes:
      install-package Microsoft.EntityFrameworkCore.Sqlite

      b) Nota: Fazer o Mock do **UserManager** e permitir obter os dados dos utilizadores é um processo complicado. A Microsoft tem uma solução para a EntityFramework que não funciona com a versão Core: https://docs.microsoft.com/en-us/ef/ef6/fundamentals/testing/mocking

      c) Nota: A solução passou por criar um método de extensão que permitisse a utilização da propriedade Users como **IQueryable<T>** e simultaneamente como **IAsyncEnumerable<T>**

   iii. Foi criada a classe **GestaoUtentesControllerTest** para testar o controlador associado. Neste caso apenas se testou o método **Edit** onde se utiliza a estratégia usada anteriormente mas, neste caso, definido uma fixture onde se cria separadamente a base dados que pode depois ser reutilizada entre testes.

2. **Testes de Integração**

   i. Para a criação dos testes unitários foi acrescentado à solução o projeto **MediESTecaIntegrationTests** usando o template **xUnit Test Project**. Neste projeto foi definida a classe de teste **HomeControllerIntegrationTests** para testar inicialmente o controlador **Home.** Para os testes de integração é necessário criar um servidor de testes que funcione autonomamente. Neste caso, utilizou-se para os testes a classe

WebApplicationFactory tal como é descrito em: https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-6.0

a) Nota: para estes testes foi necessário instalar o pacote de testes: **install-package microsoft.aspnetcore.mvc.testing**

b) Nota: tal como é descrito no documento referido, foi necessário acrescentar a linha seguinte ao ficheiro **Program.cs** do projeto original: **public partial class Program { }**

ii. Para testar o controlador dos utentes foi criada a classe **UtentesControllerIntegrationTests.** Nesta classe, utilizou-se a mesma estratégia para criar o servidor de testes. Apenas se incluiu um teste ao acesso à ação **Index** deste controlador. Neste caso, num dos testes o utilizador não fez login, pelo que é redirecionado para a página de login. No outro, o utilizador faz login antes e já deve ter acesso à página de índex.

a) Nota. Para fazer o login do utilizador podem ser usadas várias estratégias, a solução encontrada foi fazer o login do utilizador acedendo à página de login e fornecendo os dados necessários. Para isso, foi criada a classe **UserAuthentication.** A solução usada está descrita em: https://www.dotnetcurry.com/aspnet-core/1420/integration-testing-aspnet-core