Projeto 2 - Verificação e Validação de Software

Diogo Fernandes - Fc54458

Junho 2023

1 Implementação Geral de Testes

Diversos testes, narrativas e ferramentas foram utilizadas neste projeto.

Na primeira parte do projeto foi utilizado o HtmlUnit, uma ferramenta que permite a interação com páginas Html via código java. Seguiu-se a ferramenta DbSetup que permite a população da base de dados para a execução de testes unitários. Por fim, a ferramenta Mockito permite o *mock* de certos módulos de maneira a atingir um certo isolamento para a execução de testes *junit*.

1.1 Testes HtmlUnit

TestHtmlUnit é a classe java encarregue de conduzir todos os testes unitários que fazem uso desta ferramenta. Nesta classe podemos encontrar, em concreto, quatros testes - *testAddAddress*, *testInsertSale*, *testCloseExistingSale* e *testNewCustSaleDelivery*. Todos estes testes seguem uma das narrativas referidas no enunciado do trabalho.

Ao longo desta classe, ver-se-á métodos da ferramenta HtmlUnit fundamentais para a interação com as páginas da aplicação bem como métodos que permitem obter elementos Html como tabelas ou títulos.

1.1.1 testAddAddress

O teste *testAddAddress* tem a função de testar a condição de que, uma vez adicionada uma nova morada a um cliente existente, a tabela referente às moradas iria ser incrementada em uma linha.

O teste começa por obter o número de linhas original de modo a, no futuro, seja capaz de comparar o número de linhas anterior com o número atual. Este número de linhas foi obtido introduzindo o número VAT na página de informações do cliente. Esta página apresenta a tabela de moradas associada a tal cliente, pelo que basta realizar um *getRowCount* para descobrir o número de linhas da tabela.

De seguida, o teste realiza a introdução da nova morada no sistema, colocando nos devidos campos os valores necessários como a morada, a porta, o número postal e a localidade (valores estes declarados como variáveis finais no inicio do método). Uma vez a morada adicionada, o sistema retorna à página

de informações do cliente e realiza a verificação final. Para além disso, o teste conduz uma comparação de célula a célula, garantindo que a linha adiciona conta com, de facto, a nova morada.

1.1.2 testInsertSale

O teste *testInsertSale* vai testar se, uma vez adicionada uma nova venda a um cliente, esta será listada como aberta.

Assim, o teste começa adicionando a nova venda, introduzindo todos os dados necessários para a mesma. Uma vez garantida que esta foi adicionada corretamente, o teste procura a célula referente ao estado da mesma (neste caso a célula de *index* 3), fazendo uma comparação entre o estado atual e o estado esperado (neste caso, o carácter "O").

1.1.3 testCloseExistingSale

O teste *testCloseExistingSale* irá testar se, uma vez a venda fechada, esta será listada como tal.

A lógica assemelha-se bastante ao *testInsertSale* - primeiro invoca o teste anterior para criar uma venda específica (privilegia-se a reutilização de código) , de seguida fecha-se a mesma e verifica-se o estado da venda. Neste caso, diferente do que acontece com o teste anterior, a comparação é feita esperando o estado fechado (neste caso, o carácter "C").

De referir ainda duas coisas - o identificador considerado é o 1 pois deduz-se que esta venda será a primeira a ser associada e, portanto, terá de ter o id inicial; a linha a ser considerada é a de *index* 2 pois a aplicação, quando cria a tabela, reserva a primeira linha para o cabeçalho da tabela e a segunda linha como uma linha vazia (possível bug de design). Assim, e para considerar a primeira venda temos de procurar na linha 2.

1.1.4 testNewCustSaleDelivery

O teste *testNewCustSaleDelivery*, de entre todos os testes anteriormente e referidos é, talvez, o mais complexo. Este teste testa a narrativa da criação de um novo cliente, de uma nova venda e de uma nova entrega da venda.

O teste unitário começa criando um novo cliente, introduzindo nos diversos campos, as informações fundamentais para a criação do cliente como a designação, o número VAT ou o número de telefone. Uma vez confirmado a criação do cliente, é adicionada uma nova venda, utilizando a lógica do teste *testInsertSale*. A adição deste venda é verificada bem como o estado da mesma, que se espera ser aberto (carácter "O").

Uma vez adicionada a venda, e como se sabe que a venda será única, o id obtido será o número 1. Com este identificador, o teste encaminha-se para a página *Add SaleDelivery*, de modo a criar uma nova entrega para a venda criada.

O teste, por fim, verifica se a entrega foi criada corretamente através da comparação do número de linhas e dos dados presentes nas células da tabela (mais especificamente, o identificador da entrega e a morada da mesma).

1.2 Testes DbSetup

Nos testes referentes à base de dados, mais concretamente à ferramenta DbSetup, foram utilizadas duas classes java - *DBSetupUtils* e *TestDB*.

A primeira classe é responsável pela população das diferentes tabelas da base de dados como a tabela dos clientes, das moradas, entre outras. Esta classe conta, ainda, com operações importantes para a interação com as tabelas como a operação DELETE_ALL que permite a remoção de todas as entradas das diversas tabelas consideradas.

Existem três tabelas a considerar - *CUSTOMERS*, *SALES* e *ADDRESSES*. Na própria classe, existem constantes que representam o número de entradas têm cada uma das tabelas. Estas constantes vão ser importantes nos testes pois permitem comparar o número inicial de entradas com o número de entradas existentes depois de terem sido feitas alterações à base de dados.

Por fim, é importante falar da operação *INSERT_CUSTOMER_SALE_DATA*. Esta operação permite a construção da base de dados como um todo e é sempre invocada antes da execução de um teste unitário.

A classe *TestDB* conta com todos os testes referentes à base de dados. Ainda nesta classe existe dois métodos fundamentais para a execução correta dos testes - *setupClass* (que faz a configuração inicial da classe, fazendo a ligação à DB) e *setup* (que faz a configuração inicial para cada teste, eliminando informações anteriormente mantidas pela DB e reiniciando-a para o teste em questão).

1.2.1 testAddClientWithExistingVAT

Este teste consiste em adicionar um novo cliente à base de dados, fornecendo um VAT já existente na mesma. O teste, de maneira a comprovar a correção da aplicação, espera que seja lançada uma *ApplicationException*, provando, assim, a impossibilidade de se introduzir clientes com VATs duplicados. É utilizado o método *addCustomer* para a verificação de comportamento.

1.2.2 testUpdateCostumerContact

Este teste atualiza o contacto de um determinado cliente e verifica se, uma vez atualizado, essa informação é mantida de forma correta na base de dados.São utilizados os métodos *updateCustomerPhone* (para atualizar o contacto) e *getCustomerByVat* para obter o objecto referente ao cliente e, assim, verificar o seu contacto atual.

1.2.3 testEmptyDataBase

Este teste segue a narrativa de que, se for executada uma remoção geral das entradas da base de dados, então a DB deve apresentar-se vazia. Neste teste, a tabela *Customer* é limpa e, de seguida, a conexão à base de dados é reiniciada. No final é verificado que o número de clientes existentes é, de facto, zero.

1.2.4 testAddSale

Este teste verifica se, uma vez adicionada uma venda à tabela, esta é guardada de maneira correta. Assim, utiliza a função *getAllSales* do *SaleService* para comparar o números de vendas iniciais com o número de vendas atuais. Este teste conta ainda com uma constante - *NUM_INIT_SALES* que guarda o número de entradas iniciais da tabela *SALES* de modo a que seja possível verificar a incrementação da tabela.

1.2.5 testUpdateSale

Este teste vai verificar se, uma vez atualizada a venda, esta apresenta-se da maneira esperada.

Primeiramente, o teste começa por introduzir a venda associada a um VAT e verifica se objecto de venda criada não é *null*. De seguida, obtém o identificador da venda em questão e atualiza-a. Por fim, compara o estado da venda com o estado esperado (que neste caso se espera ser fechado).

1.2.6 testGetAllSalesFromVAT

Este teste adiciona uma número de vendas a um determinado cliente com o número VAT especificado (através de um *loop* simples que cria essas vendas com um identificador incremental) e verifica se, depois de adicionadas, o número de vendas corresponde ao número esperado.

1.2.7 testAddInexistentSaleDelivery

Este teste adiciona uma entrega a uma venda inexistente esperando, em retorno, por uma exceção por parte da aplicação, provando que este tipo de comportamento não é aceite pelo sistema.

1.2.8 testGetAllSalesDeliveriesFromVAT

Este teste, em termos lógicos, assemelha-se ao *testGetAllSalesFromVAT*. Neste caso, adiciona-se tanto um número determinado de vendas como as respetivas entregas (tudo isto acontece dentro de um *loop*). No final, compara-se o número de entregas presentes na tabela com aqueles que eram esperados depois da execução do *loop*.

1.3 Mockito

A ferramenta *Mockito* permite a isolação de certos módulos de modo a que possamos testar código no ambiente mais controlado e previsível. No entanto, esta ferramenta conta com algumas limitações. A ferramenta não consegue, por exemplo, fazer *mock* de classes finais, de enums ou de classes anónimas.

No meu entendimento, a resposta a esta pergunta é não, por diversos motivos. Primeiramente, as classes Service são classes enum, sendo impossível

de fazer *mock*. Por outro lado, os métodos privados não podem ser *stubbed* e os métodos públicos contam com tipos primitivos como argumentos, sendo impossível utilizar objetos *mocked* para testar a execução dos mesmos.

As classes restantes, mais concretamente as classes DTO, são classes que permitem a criação de objetos que interagem com a base de dados. Estas classes contam apenas com parâmetros finais e não contam com nenhum tipo de método, nem mesmo simples *gets*. O mesmo acontece com a classe *ApplicationException*.

Assim, e para conseguir utilizar a ferramenta, é necessário modificar o código de diversas formas.De seguida, enumero uma delas.

Consideremos a classe *SaleService*, mais especificamente, o método *getSaleByCustomerVAT*. Neste método, o argumento utilizado é um tipo primitivo (um *int*) que consiste no número VAT a ser utilizado na pesquisa de vendas associadas ao mesmo. Se, ao invés de ser utilizado o número VAT, fosse utilizado um objeto *CustomerDTO* (o que faz sentido já que se procura por vendas associadas a um cliente, uma entidade), seria possível fazer *mock* do cliente e avaliar a dependência da classe ao objeto cliente. Eventualmente, poderia ser feito, num estágio final, uma *assertion* que compare informações esperadas com as obtidas.

Importante ainda referir que, como o objeto *CustomerDTO* conta com parâmetros finais, não é possível utilizar a ferramenta *Mockito* no código original. Assim, é ainda necessário fazer algumas mudanças na classe DTO, como alterar a designação final dos parâmetros ou introduzir métodos *gets* para os mesmos.

2 BugTracking

2.1 Customer Address insert allows one to insert invalid locality name

Este erro refere-se ao facto de que, na aplicação, foi detetada a possibilidade de se inserir, no campo destinado à localidade, números e que esta informação é aceite pela aplicação. Como é óbvio, um nome de uma localidade não deve ser um número visto ser um nome representativo da mesma e não um código de localidade.

A reprodução deste erro é 10/10 pelo que é bastante simples de o verificar. Tal como está enunciado no *backlog*, basta o individuo adicionar uma nova morada com um número como localidade e observar o comportamento da aplicação.

2.2 Shouldn't be possible to add a sale with an invalid VAT number or with a inexistent VAT number

Este *bug* refere-se ao facto de ser possível adicionar uma venda a um número VAT inválido ou inexistente na base de dados. No momento em que este número é introduzido, a aplicação, de facto, lança um erro realçando o problema quanto

ao VAT inserido. No entanto, e observando a tabela de vendas existentes podemos constatar que a venda é apresentada e permite a sua manipulação de estado.

A reprodução é, novamente, 10/10. Para verificar este erro, basta o individuo introduzir um número VAT inexsitente ou inválido e observar se a venda foi, de facto, adicionada à tabela de vendas.