Capa e index e introdução e conclusão, dizer que na primeira secção fei feita uma análise ao codigoporque foi o primeiro exercício de testes, logo foi durante a execução das tarefas dessa secção que se estudou o código do SUT.

# Teste de narrativas com HtmlUnit

Neste tipo de testing, são usadas as funcionalidades disponíveis nas views (html) da aplicação, como links e formulários.

Os testes para cada uma das narrativas pedidas está no package htmlunit\_test

Foi criada uma HtmlUnitUtils classe com métodos para as operações repetidas necessárias para as diferentes narrativas pedidas. Exemplos destas operações que se repetem nos testes são: criar um customer, remover um customer, criar address, remover um address, criar uma sale, remover uma sale, e remover uma delivery.

Como está-se a usar a base de dados da aplicação para fazer testes, é necessário reverter as operações feitas nos testes, que alteraram o estado da base de dados. As alterações que envolvem alterar o estado da base dados, que são necessárias para os testes, são as operações de insert de customers, addresses, sales e deliveries. Por isto, foi necessário implementar na web\_app, casos de uso para remoção de addresses, sales e deliveries (remoção de clientes já estava implementado). No ponto 1.6. é explicada a implementação destes casos de uso extra.

Para cada narrativa no enunciado, foram feitos testes manuais de forma a se perceber o fluxo da troca de páginas, e o conteúdo de cada página. Também foram descobertos alguns bugs neste processo, preenchendo os formulários com valores não adequados e lendo o código do SUT, sendo estes explorados em testes, devidamente documentados. Com isto surgiram sequências de operações para as quais não é possível escrever um teste, pois não é sabido o comportamento esperado, sendo então apenas documentados neste relatório.

É muito importante destacar que **para correr estes testes é necessário ter o sistema a correr**.

## Narrativa a)

Testes para a narrativa definida em 2.a) estão na classe InsertTwoAddressesToCustomerNarrativeTest.

Para além de testar a narrativa preenchendo os formulários com valores adequados, também se testou manualmente se seria possível criar addresses com campos vazios, quer seja da morada, porta, código postal ou localidade. Foi verificado que é possível criar addresses para clientes com algum ou todos estes campos vazios Bug ou era preciso os requirements?.

Mas como isto são detalhes que têm de ser esclarecidos com os requerimentos do cliente do SUT, não é possível escrever testes para estes casos, dai apenas haver um teste para o “happy path”.

## Narrativa b)

Testes para a narrativa definida em 2.b) estão na classe InsertTwoNewCustomersNarrativeTest.

Com os testes relativos a esta narrativa, foi descoberto o seguinte bug:

1. É possível criar números de telefone com números negativos.

Para resolver este bug, foi acrescentada uma verificação para o número de telefone dado, no método addCustomer da classe CustomerService. Caso seja dado um número negativo, é lançada uma exceção.

Para além deste bug em concreto, foram descobertas funcionalidades que dependendo dos requisitos do cliente do SUT, podem ser determinados como bugs ou não:

* É possível criar números de telefone com qualquer e combinação de dígitos, desde que sejam positivos, e.g., 0 ou 200
* É possível criar customers com designação vazia

## Narrativa c)

Testes para a narrativa definida em 2.c) estão na classe CreateSaleNarrativeTest.

Com os testes relativos a esta narrativa foi descoberto o seguinte bug:

1. É possível criar uma sale com um customer vat, com formato válido, mas para o qual não existe nenhum cliente associado a ele.

Para corrigir este bug é feita uma procura por um customer com o vat dado, antes de inserir a sale, na classe AddSalePageController. Caso a pesquisa do customer pelo vat falhe, a inserção da sale não é feita, pois é lançada uma exceção que interrompe a execução do caso de uso de inserção de uma sale.

## Narrativa d)

Testes para a narrativa definida em 2.d) estão na classe CloseSaleNarrativeTest.

Este caso de uso é bastante simples, sendo apenas necessário preencher o id da sale que se pretende fechar. Passar ids inválidos (negativos) ou que não estejam associados a nenhuma sale, são tratados no SUT, pois é feita uma operação de update na base de dados, portanto esses valores errados não existiram na base de dados, logo o um update com valores inadequados não mudaria estado nenhum na base dados, não deixando o sistema num estado erróneo. Portanto na classe de testes para esta narrativa apenas está um teste, para o “happy path”.

## Narrativa e)

Testes para a narrativa definida em 2.e) estão na classe CreateDeliveryNarrativeTest.

Com os testes relativos a esta narrativa, foram descobertos os seguintes bugs:

1. É possível criar Deliveries com id’s de Addresses que não existem, e.g., addr\_id = -1.

Para resolver este bug foi criado um método de procura de address por id numa nova class AddressFinder. No método addSaleDelivery, em SaleService, é chamado o método de procura para o addr\_id dado, para garantir que ele existe. Caso não existir, este método vai lançar uma exceção, interrompendo o fluxo da função de inserção de delivery, em vez de deixar inserir deliveries com address\_id’s que não existem.

1. Construtor de AddressRowDataGateWay que recebe um ResultSet tinha nome da coluna onde está o customer vat com o nome errado

No ficheiro onde é feito o setup da base de dados (createDDLHSQLDB.sql) é possível ver o nome correto das colunas. Natabela Address o nome da coluna que guarda o vat do customer é “customer\_vat”, e na classe AddressRowDataGateway, no construtor que recebe um ResultSet, estava-se a obter o vat numa coluna “customerVat”, o que está errado pois esta coluna não existe. Portanto, substituindo o nome errado (customerVat) pelo correto (customer\_vat), resolve o problema.

1. Criar SaleDelivery com saleId de uma sale fechada

Para além destes bugs, foram encontradas algumas funcionalidades, durante testes manuais da applicação, que dependendo dos requisitos do cliente do software, podem ser determinados como sendo o comportamento esperado, ou então um bug:

* É possível um customer criar uma delivery com um address id, de um Address criado por outro customer
* É possível um customer criar uma delivery com um sale id, de uma Sale criada por outro customer.

## Reverter os inserts feitos nos testes para as narrativas

Modificou-se o SUT, para ter a capacidade de remover Adresses, Sales e Deliveries de Customers. Nesta subsecção detalha-se como é que foram implementados casos de uso extra para haver a capacidade de “limpar” os efeitos secundários produzidos nos testes.

Para se poder remover Addresses acrescentou-se um link no index.html para se ser direcionado a um form de remoção de Address, parecido com o de inserção de um novo Address. A diferença é que este form faz um pedido para um outro PageController, criado por mim, o RemoveAddressPageController. Este controller recebe os dados necessários para a remoção de um Address de um Customer. Para seguir a logica de “separation of concerns” seguida pelos developers da app, foi criado o método removeAddressOfCustomer, na class CustomerService, método este que é então chamado no controller, abstraindo-o dos detalhes da remoção de um Address. Nesta função no CustomerService, é primeiro verificado que o customer identidicado com o vat dado tem o Address a remover, e no fim foi usada de novo a mesma lógica dos developers, sendo por isso usada AddressRowDataGateway, para tratar da lógica da camada dos dados.

Para se poder remover Sales, acrescentou-se um link no index.html para se ser direcionado para um form, que recebe o vat do customer, que criou a sale a apagar, e o id da sale a apagar. O pedido submetido por este form é tratado pelo RemoveSalePageController. Este controller lê os campos do pedido e usa o SalesService para se abstrair da lógica da remoção de uma sale, chamando o método removeSale com os valores do vat e id da sale lidos no pedido. No método removeSale, em SaleService, é verificado que o customer com o vat dado foi quem criou a sale, e é usada a class SaleRowDataGateway para se tratar da lógica de remoção de uma sale da base de dados.

Para se poder remover Deliveries acrescentou-se um link no index.html, para se ser direcionado para um form, que recebe o vat do customer, que criou a delivery, e o id da delivery a remover. Ao submeter um form é feito um pedido, que é tratado pelo controller RemoveDeliveryPageController. Neste controller são lidos os campos do pedido, e é usado o SaleService para abstrair o controller da remoção de uma delivery, com o método removeDelivery. Neste método, é primeiro verificado que o customer com o vat dado foi quem criou a Delivery , e é usado SaleDeliveryRowDataGateway para se tratar da lógica de remoção de Deliveries na base de dados.

Neste processo todo, como foram usadas várias funcionalidades implementadas pelos developers, foi possível ter um melhor entendimento sobre os detalhes da implementação, o que tornou possível descobrir os seguintes bugs:

1. Strings dos Addresses que são retornados pela BD têm vários espaços em branco, no fim o que pode causar erros quando são feitas comparações da string dos addresses vindos da bd.

Sempre que se faz uma query para obter Addresses da base de dados, feito um trim() nas strings que representam o address em texto.

# DBSetup testing

Neste tipo de testing, é feita uma ligação direta á base de dados do sistema, de forma a testar a validade das operações feitas pelo sistema, que envolvem aceder à base de dados.

Testes para cada um dos requisitos pedidos no ponto 3. estão no package dbsetup\_tests. Neste package estão classes em que são feitos testes relativos a certas entidades da base de dados, e.g., na classe CustomerDBTest estão os testes relativos a operações na base de dados sobre registos de Customer. Os requisitos para este tipo de testing são relativos a Customers, Sales e SaleDevlieries, portanto há as classes CustomerDBTest, SaleDBTest e SaleDeliveryDBTest.

Para a execução desta tarefa, foi usada a classe DBSetupUtils, apresentada nas aulas de VVS. Nesta classe já está implementada a ligação à base de dados e também estão implementadas uma série de operações para a base de dados. Também foi usado o mesmo esquema apresentado nas aulas, em ter um método setupClass, anotado com @BeforeAll, que trata da ligação com a base de dados, e um método setup, anotado com @BeforeEach, que trata de usar operações expostas por DBSetupUtils, de forma a limpar a base de dados e inserir os dados pretendidos.

## Testes sobre Customers

Testes sobre Customers estão implementados na classe CustomerDBTest.

Nesta classe, foram tratados os requisitos a), b), c), d), e). Os testes estão documentados de forma que seja direto perceber qual o requisito que está a ser tratado.

Todos os testes escritos para os requisitos especificados acima correram sem erros, exceto o teste removingCustomerRemovesHisSalesTest, que cobre o requisito e), que a sua primeira execução falhou, indicando um possível bug no sistema. Analisando o código responsável por processar a remoção de um cliente, percebe-se que não há qualquer tratamento sobre as sales do cliente a romver. Também no ficheiro onde está a criação da base de dados, não é configurada nenhuma estratégia de “cascade” e de “foreign key” sobre a coluna “customer\_vat”, da tabela Sale. Para se resolver o bug adicionou-se no método removeCustomer na classe CustomerService um ciclo que percorre a lista de Sales do Customer a remover, e essas sales vão sendo “deleted”. Esta solução resulta, mas uma solução possivelmente mais elegante seria, na configuração da base de dados, ter a “cusomter\_vat”, da tabela sale, como chave estrangeira que referência o vat da tabela customer e que haveria um mecanismo de cascade em que caso um customer fosse removido, todos os registos da tabela salem, que têm uma referência à sua coluna “vat”, também seriam removidos automaticamente.

Para facilitar a implementação dos testes foi implementado um método privado para verificar se existe um cliente com um determinado vat.

## Testes sobre Sales

Testes sobre Sales estão implementados na classe SaleDBTest.

Nesta classe, foi tratado o requisito f) e há também dois testes originais extra, como pedido. Os testes estão documentados de forma que seja direto perceber qual o requisito que está a ser tratado.

Como identificar no código os testes extra?

O teste para o requisito f) e os testes originais extra passaram sem problemas. Os testes originais extra desenvolvidos focam-se na operação de atualizar o estado de uma Sale, por outras palavras, fechar uma Sale. O teste addSaleToCustomerWithoutOpenSalesThenCloseItResultsInNoOpenSalesTest, explora o caso de abrir uma Sale a um Customer, sem Sales abertas, verificar que este tem uma sale aberta e verificar que quando se fecha essa mesma Sale o Customer volta a ficar sem Sales abertas. O teste closeNonExistentSaleTest, explora o caso de chamar a função updateSale com um sale id que não existe no sistema, que deve resultar no lançamento de uma exceção.

Para facilitar a implementação dos testes foi implementado um método privado para verificar se um cliente com um determinado vat tem Sales abertas.

## Testes sobre SaleDeliveries

Testes sobre SaleDeliveries estão implementados na classe SaleDeliveryDBTest.

Nesta classe, há apenas dois testes originais extra, como pedido. Os testes estão documentados de forma que seja direto perceber qual o caso a ser testado. O teste addSaleDeliveryWithNonExistentSale, explora o caso de criar uma SaleDelivery com um sale id que não existe, que deve resultar no lançamento de uma exceção. O teste addSaleDeliveryToCustomerWithNoSaleDeliveries, explor o caso de cruar uma SaleDelivery num Customer, que não tem SaleDeliveries, e verificar que depois disto o Customer tem SaleDeliveires.

Para facilitar a implementação dos testes foi implementado um método privado para verificar se um cliente com um determinado vat tem Addresses criados e outro para verificar se tem SaleDeliveries.

Ponto 4. Mcokar um dos services e testar o controller em que esse service é usado