Relatório Para o Segundo Projeto de VVS

BRUNO LIU FC56297

HTMLUnit

Para esta parte do projeto foi utilizado o HTMLUnit para realizar os testes. Após a execução dos testes é garantido que o sistema volta ao estado que tinha antes de começar os testes.

Para isso foi necessário implementar 3 páginas extras:

- DeleteLastSaleController.java -> Apaga a última sale inserida no sistema a um certo customer
- DeleteAllSaleDeliveryController.java -> Apaga todas as sale deliveries inserida no sistema a um certo customer
- DeleteAllAddressController.java -> Apaga todas os endereços inseridos no sistema a um certo customer

Foi necessário implementar estas 3 funcionalidades pois o sistema não as possuía, estas funcionalidades são essenciais para garantir que o sistema volta ao seu estado base após a realização dos testes.

a) Insert two new addresses for an existing customer, then the table of addresses of that client includes those addresses and its total row size increases by two

Foi escolhido um cliente já existente na base de dados, o "JOSE FARIA" cujo o seu VAT é 197672337.

Antes da introdução dos novos endereços para o cliente escolhido foi obtida a informação da invariante do sistema, que neste caso é o número de endereços existentes para esse cliente, atualmente, antes da realização dos testes. Este valor será utilizado para verificar se os endereços foram bem introduzidos e para verificar se os endereços foram bem retirados depois de voltar a remover esses endereços. O número de endereços foi obtido através do acesso à página "Find Customer by vat number", onde são expostos os endereços do cliente cujo o vat lhe pertence.

```
String customerVAT = "197672337";

DomElement findCustomerByVatButton = page.getElementsById("botao2").get(3);
assertEquals("Find customer by vat number", findCustomerByVatButton.asText());
HtmlPage findCustomerByVatPage = findCustomerByVatButton.click();
HtmlForm findCustomerByVatForm = findCustomerByVatPage.getForms().get(0);
findCustomerByVatForm.getInputByName("vat").setValueAttribute(customerVAT);
HtmlSubmitInput submitButton2 = findCustomerByVatForm.getInputByValue("Get Customer");
HtmlPage clientInfoPage = submitButton2.click();
List<DomElement> rows = (List<DomElement>) clientInfoPage.getElementsByTagName("tr");
int actualNumberOfAddresses = rows.size()-1;
```

Após o número atual de endereços ser obtido foi feita a introdução de endereços através da página “Insert new Address to Customer”.

```
DomElement addAddressToCustomerButton = page.getElementsById("botao2").get(1);
assertEquals("Insert new Address to Customer", addAddressToCustomerButton.asText());
HtmlPage addAddressToCustomerPage = addAddressToCustomerButton.click();
HtmlForm form = addAddressToCustomerPage.getForms().get(0);

//First Address
String address1= "Rua 1";
String door1 = "Porta 1";
String postalCode1 = "1234-1";
String locality1 = "Lisboa";
form.getInputByName("vat").setValueAttribute(customerVAT);
form.getInputByName("address").setValueAttribute(address1);
form.getInputByName("door").setValueAttribute(door1);
form.getInputByName("postalCode").setValueAttribute(postalCode1);
form.getInputByName("locality").setValueAttribute(locality1);
HtmlSubmitInput submitButton = form.getInputByValue("Insert");
HtmlPage nextPage = submitButton.click();

//Second Address
String address2 = "Rua 2";
String door2 = "Porta 2";
String postalCode2 = "1234-2";
String locality2 = "Porto";
form.getInputByName("vat").setValueAttribute(customerVAT);
form.getInputByName("address").setValueAttribute(address2);
form.getInputByName("door").setValueAttribute(door2);
form.getInputByName("postalCode").setValueAttribute(postalCode2);
form.getInputByName("locality").setValueAttribute(locality2);
submitButton = form.getInputByValue("Insert");
nextPage = submitButton.click();
```

Após a introdução dos endereços ser feita, foi obtido o número total de endereços para esse certo cliente, mas agora após a realização do teste.

```
//VERIFICAR TABLES ADDRESS DESSE CLIENTE E VERIFICAR SE AUMENTOU O NUMERO DE ADDRESS EM 2
findCustomerByVatForm.getInputByName("vat").setValueAttribute(customerVAT);
submitButton2 = findCustomerByVatForm.getInputByValue("Get Customer");
clientInfoPage = submitButton2.click();

rows = (List<DomElement>) clientInfoPage.getElementsByTagName("tr");
int numberOfAddresses = rows.size()-1;

if (actualNumberOfAddresses == -1) { //BECAUSE IT DOESNT HAVE A TABLE AND WILL CREATE 1 ROW EXTRA FOR THE TABLE HEADER
    assertEquals(numberOfAddresses, actualNumberOfAddresses + 3);
} else {
    assertEquals(numberOfAddresses, actualNumberOfAddresses + 2);
}
```

Depois de obter o valor é feita a comparação do número de endereços antes e depois da introdução de novos. É possível reparar que há um condicional, esse condicional existe, pois, para obter o número de endereços é contada as linhas da tabela, e se caso um cliente não tenha nenhum endereço associado essa tabela não existirá, isso significa que quando introduzido o primeiro endereço a quantidade de linha irá aumentar em 2, pois será introduzido o cabeçalho da tabela, que conta também como uma linha.

A seguir dos valores serem comparados é feita a remoção dos endereços através do acesso à nova página criada por mim.

```
//DELETE DE TODOS OS ADDRESS DO CUSTOMER
DomElement deleteAllAddressFromCustomerButton = page.getElementsById("botao2").get(12);
HtmlPage deleteAllAddressFromCustomerPage = deleteAllAddressFromCustomerButton.click();
HtmlForm deleteAllAddressFromCustomerForm = deleteAllAddressFromCustomerPage.getForms().get(0);
deleteAllAddressFromCustomerForm.getInputByName("customerVat").setValueAttribute(customerVAT);
HtmlSubmitInput deleteAllAddressButton = deleteAllAddressFromCustomerForm.getInputByValue("Delete All Address");
deleteAllAddressButton.click();
```

No final são comparadas as linhas de novo para verificar que o estado do programa antes da execução do teste se mantém.

```
//FAZER UM ASSERT PARA A INVARIANTE DO SISTEMA
clientInfoPage = (HtmlPage) clientInfoPage.refresh();
rows = (List<DomElement>) clientInfoPage.getElementsByTagName("tr");
int finalNumberOfAddresses = rows.size()-1;

assertEquals(actualNumberOfAddresses, finalNumberOfAddresses);
```

b) Insert two new customers and check if all the information is properly listed in the List All Customers use case

Antes da introdução de 2 novos clientes foi obtida a informação da invariante do sistema, que neste caso é o número total de clientes existentes atualmente, antes da realização dos testes. Este valor foi obtido através da página "List All Customers" e apenas será utilizado mais tarde para verificar se os clientes foram corretamente removidos.

```
//OBTER NUMERO DE CUSTOMER INICIAL
DomElement listAllCustomerButton = page.getElementsById("botao2").get(5);
HtmlPage listAllCustomerPage = listAllCustomerButton.click();
String numberOfClientsText = listAllCustomerPage.getElementById("body").asText();
String prefix = "Number of Clients: ";
int startIndex = numberOfClientsText.indexOf(prefix);
int endIndex = numberOfClientsText.indexOf("\n", startIndex);
String numberOfCustomersInicial = numberOfClientsText.substring(startIndex + prefix.length(), endIndex).trim();
```

Foi utilizada a página "Insert new Customer" e ao preencher os campos foram introduzidos 2 novos clientes.

```
DomElement addCustomerButton = page.getElementsById("botao2").get(0);
assertEquals("Insert new Customer", addCustomerButton.asText());
HtmlPage addCustomerPage = addCustomerButton.click();
HtmlForm form = addCustomerPage.getForms().get(0);

//First Customer
String firstName = "Customer Teste 1";
String firstPhone = "961234578";
String firstVAT = "123456789";
form.getInputByName("vat").setValueAttribute(firstVAT);
form.getInputByName("designation").setValueAttribute(firstName);
form.getInputByName("phone").setValueAttribute(firstPhone);
HtmlSubmitInput submitButton = form.getInputByValue("Get Customer");
submitButton.click();

//Second Customer
String secondName = "Customer Teste 2";
String secondPhone = "213456789";
String secondVAT = "269065822";
form.getInputByName("vat").setValueAttribute(secondVAT);
form.getInputByName("designation").setValueAttribute(secondName);
form.getInputByName("phone").setValueAttribute(secondPhone);
submitButton = form.getInputByValue("Get Customer");
submitButton.click();
```

Após a introdução dos 2 novos clientes a verificação da existência deles no sistema foi feita da seguinte maneira, foi acedida a página "List All Customers" e foi obtido as últimas 2 linhas da tabela, que iram corresponder aos 2 clientes inseridos, com essas 2 linhas foi feita a comparação de cada campo com a informação dos 2 clientes inseridos.

```
//VERIFICAR TABLES ADDRESS DESSE CLIENTE
DomElement listAllCustomersButton = page.getElementsById("botao2").get(5);
HtmlPage listAllCustomersPage = listAllCustomersButton.click();

List<DomElement> rows = (List<DomElement>) listAllCustomersPage.getElementsByTagName("tr");

DomElement firstAddedRow =rows.get(rows.size()-2);
List<HtmlElement> firstCells = firstAddedRow.getElementsByTagName("td");
assertTrue(firstCells.get(0).getTextContent().equals(firstName));
assertTrue(firstCells.get(1).getTextContent().equals(firstPhone));
assertTrue(firstCells.get(2).getTextContent().equals(firstVAT));


DomElement secondAddedRow =rows.get(rows.size()-1);
List<HtmlElement> secondCells = secondAddedRow.getElementsByTagName("td");
assertTrue(secondCells.get(0).getTextContent().equals(secondName));
assertTrue(secondCells.get(1).getTextContent().equals(secondPhone));
assertTrue(secondCells.get(2).getTextContent().equals(secondVAT));
```

Foi utilizada a página “Remove Existing Customer” para remover os 2 clientes inseridos.

```
//Remover customers inseridos
DomElement removeCustomerButton = page.getElementsById("botao2").get(2);
HtmlPage removeCustomerPage = removeCustomerButton.click();
HtmlForm removeCustomerform = removeCustomerPage.getForms().get(0);
removeCustomerform.getInputByName("vat").setValueAttribute(firstVAT);
HtmlSubmitInput removeButton = removeCustomerform.getInputByValue("Remove");
removeButton.click();
removeCustomerform.getInputByName("vat").setValueAttribute(secondVAT);
removeButton.click();
```

Após a remoção foi obtido o número total de clientes após o teste e foi comparado com o valor obtido inicialmente e é verificado se são iguais.

```
//VERIFICAR A INVARIANTE DO SISTEMA

//OBTER NUMERO DE CUSTOMER FINAL
DomElement listAllCustomerButton2 = page.getElementsById("botao2").get(5);
HtmlPage listAllCustomerPage2 = listAllCustomerButton2.click();

String numberOfClientsText2 = listAllCustomerPage2.getElementById("body").asText();
int startIndex2 = numberOfClientsText2.indexOf(prefix);
int endIndex2 = numberOfClientsText2.indexOf("\n", startIndex2);
String numberOfCustomersFinal = numberOfClientsText2.substring(startIndex2 + prefix.length(), endIndex2).trim();

//VERIFICAR QUANTIDADE DE CUSTOMERS
assertEquals(numberOfCustomersInicial, numberOfCustomersFinal);
```

c) A new sale will be listed as an open sale for the respective customer

Antes da introdução de 1 nova sale foi obtida a informação da invariante do sistema, que neste caso é o número total de sales existentes para um certo cliente, atualmente, antes da realização dos testes. Este valor foi obtido através da página “Show Customer Sale’s” e apenas será utilizado mais tarde para verificar se a sale foi corretamente removida.

```
String customerVAT = "197672337";

//OBTER INVARIANTE DO SISTEMA
DomElement showCustomersSaleButton = page.getElementsById("botao2").get(8);
assertEquals("Show Customer Sale's", showCustomersSaleButton.asText());
HtmlPage showCustomersSalePage = showCustomersSaleButton.click();
HtmlForm showCustomersSaleform = showCustomersSalePage.getForms().get(0);
showCustomersSaleform.getInputByName("customerVat").setValueAttribute(customerVAT);
HtmlSubmitInput showCustomersSaleButtonForm = showCustomersSaleform.getInputByValue("Get Sales");
HtmlPage saleInfoPage = showCustomersSaleButtonForm.click();
List<DomElement> rows2 = (List<DomElement>) saleInfoPage.getElementsByTagName("tr");
int numberOfSalesForThatCustomer = rows2.size() - 1;
```

Foi criada uma nova sale para um cliente já existente na base de dados por default, através da página “Insert New Sale”.

```
DomElement newSaleButton = page.getElementsById("botao2").get(6);
assertEquals("Insert new Sale", newSaleButton.asText());
HtmlPage newSalePage = newSaleButton.click();
HtmlForm newSaleform = newSalePage.getForms().get(0);
newSaleform.getInputByName("customerVat").setValueAttribute(customerVAT);
HtmlSubmitInput addSaleButton = newSaleform.getInputByValue("Add Sale");
HtmlPage saleInfoPage2 = addSaleButton.click();
```

Após a sale ser criada sistema automaticamente reencaminha a página para uma pagina que contém todas as sales desse cliente, por essa pagina foi obtida a ultima linha, que corresponde à sale mais recente, e foi feita a comparação com os valores defaults que a sale fica quando é acabada de criar

```
// verificar sale inserido
Date todaysDate = new Date();
List<DomElement> rows = (List<DomElement>) saleInfoPage2.getElementsByTagName("tr");
int numberOfSalesForThatCustomerAfterTest = rows.size() - 1;
if (numberOfSalesForThatCustomer == -1) { //BECAUSE IT DOESNT HAVE A TABLE AND WILL CREATE 1 ROW EXTRA FOR THE TABLE HEADER
    assertEquals(numberOfSalesForThatCustomer+2, numberOfSalesForThatCustomerAfterTest);
} else {
    assertEquals(numberOfSalesForThatCustomer+1, numberOfSalesForThatCustomerAfterTest);
}

for (DomElement row : rows) {
    List<HtmlElement> cells = row.getElementsByTagName("td");
    for (int i = 0; i < cells.size(); i++) {
        HtmlElement cell = cells.get(i);
        if (i == 0) {
            assertTrue(cell.getTextContent().equals(1)); //JUST A COUNTER
        } else if (i == 1) {
            assertTrue(cell.getTextContent().equals((new java.sql.Date(todaysDate.getTime())).toString())); // DATA
                                                                                                            // DE
                                                                                                            // HOJE
        } else if (i == 2) {
            assertTrue(cell.getTextContent().equals("0.0"));
        } else if (i == 3) {
            assertTrue(cell.getTextContent().equals("0"));
        } else if (i == 4) {
            assertTrue(cell.getTextContent().equals(customerVAT));
        }

    }
}
```

Depois da comparação foi realizada a remoção da sale através da página “Delete Last Sale”, criada por mim.

```
// APAGAR SALE
DomElement deleteLastSaleMenuButton = page.getElementsById("botao2").get(11);
HtmlPage deleteLastSalePage = deleteLastSaleMenuButton.click();
HtmlForm deleteLastSaleform = deleteLastSalePage.getForms().get(0);
deleteLastSaleform.getInputByName("customerVat").setValueAttribute(customerVAT);
HtmlSubmitInput deleteLastSaleButton = deleteLastSaleform.getInputByValue("Delete Last Sale");
deleteLastSaleButton.click();
```

No final foi obtido outra vez o número total de sales para comparar com o valor inicial e certificar que não mudou.

```
showCustomersSaleButton = page.getElementsById("botao2").get(8);
assertEquals("Show Customer Sale's", showCustomersSaleButton.asText());
showCustomersSalePage = showCustomersSaleButton.click();
showCustomersSaleform = showCustomersSalePage.getForms().get(0);
showCustomersSaleform.getInputByName("customerVat").setValueAttribute(customerVAT);
showCustomersSaleButtonForm = showCustomersSaleform.getInputByValue("Get Sales");
saleInfoPage = showCustomersSaleButtonForm.click();
rows2 = (List<DomElement>) saleInfoPage.getElementsByTagName("tr");
int numberOfSalesForThatCustomerFinal = rows2.size() - 1;

assertEquals(numberOfSalesForThatCustomer, numberOfSalesForThatCustomerFinal);
```

d) After closing a sale, it will be listed as closed

Os passos para este teste são os mesmos até a inserção de uma sale nova, após essa inserção, é utilizada a página “Close Existing Sale” para fechar essa mesma sale.

```
//        //FECHAR A SALE
          DomElement closeSaleButton = page.getElementsById("botao2").get(7);
          assertEquals("Close Existing Sale", closeSaleButton.asText());
          HtmlPage closeSalePage = closeSaleButton.click();
          HtmlForm closeSaleform = closeSalePage.getForms().get(0);
          closeSaleform.getInputByName("id").setValueAttribute(saleID);
          HtmlSubmitInput closeSaleButtonForm = closeSaleform.getInputByValue("Close Sale");
          closeSaleButtonForm.click();
```

Com a sale fechada é verificada através da tabela presente na página “Show Customer Sale”, a última linha da tabela, que corresponde a essa sale e é possível verificar o campo status e perceber que a sale está fechada, i.e, “C”.

```
// CONFIRMAR QUE A SALE FOI INSERIDA
DomElement showCustomersSaleButton2 = page.getElementsById("botao2").get(8);
assertEquals("Show Customer Sale's", showCustomersSaleButton2.asText());
HtmlPage showCustomersSalePage2 = showCustomersSaleButton2.click();
HtmlForm showCustomersSaleform2 = showCustomersSalePage2.getForms().get(0);
showCustomersSaleform2.getInputByName("customerVat").setValueAttribute(customerVAT);
HtmlSubmitInput showCustomersSaleButtonForm2 = showCustomersSaleform2.getInputByValue("Get Sales");
saleInfoPage = showCustomersSaleButtonForm2.click();

List<DomElement> rows3 = (List<DomElement>) saleInfoPage.getElementsByTagName("tr");
List<HtmlElement> cells2 = rows3.get(rows.size() - 1).getElementsByTagName("td");
for (int i = 0; i < cells2.size(); i++) {
    HtmlElement cell2 = cells2.get(i);
    if (i == 0) {
        assertTrue(cell2.getTextContent().equals(saleID));
    } else if (i == 1) {
        assertTrue(cell2.getTextContent().equals((new java.sql.Date(todaysDate.getTime())).toString())); // DATA
                                                                                                         // DE
                                                                                                         // HOJE
    } else if (i == 2) {
        assertTrue(cell2.getTextContent().equals("0.0"));
    } else if (i == 3) {
        assertTrue(cell2.getTextContent().equals("C"));
    } else if (i == 4) {
        assertTrue(cell2.getTextContent().equals(customerVAT));
    }
}
```

Após isso os passos voltam a ser semelhantes aos passos presentes no teste anterior, é apagada a sale e é feita a verificação com o número de sales existentes.

e) Create a new customer, create a new sale for her, insert a delivery for that sale and then show the sale delivery. Check that all intermediate pages have the expected information.

Este teste é uma junção de todos os testes anteriores.

Um cliente é criado, é adicionado um endereço a esse cliente para depois ser utilizado no sale delivery. É criada uma sale para esse mesmo cliente e de em seguida é criada uma sale delivery com a id da sale criada e com o id do endereço que o cliente possui.

Entre estes passos todos é feita a verificação para perceber se os passos foram realizados com sucesso.

Depois de introduzir a sale delivery a mesma é verificada se foi bem introduzida:

```
HtmlForm addSaleDeliveryform = addSaleDeliveryPage.getForms().get(0);
addSaleDeliveryform.getInputByName("addr_id").setValueAttribute(addressID);
addSaleDeliveryform.getInputByName("sale_id").setValueAttribute(saleID);
HtmlSubmitInput insertButton = addSaleDeliveryform.getInputByValue("Insert");
HtmlPage customerSaleInfoPage = insertButton.click();

// verificar sale DELIVERY inserido
List<DomElement> rows3 = (List<DomElement>) customerSaleInfoPage.getElementsByTagName("tr");
int numberOfDeliveryForThatCustomer = rows3.size() - 1;
List<HtmlElement> cells3 = rows3.get(rows3.size() - 1).getElementsByTagName("td");
for (int i = 0; i < cells3.size(); i++) {
    cell = cells3.get(i);
    if (i == 0) {
        assertTrue(cell.getTextContent().equals(1)); //JUST AN ID COUNTER
    } else if (i == 1) { // SALE ID
        assertTrue(cell.getTextContent().equals(saleID));
    } else if (i == 2) { // ADDRESS ID
        assertTrue(cell.getTextContent().equals(addressID));
    }
}
```

Após a verificação da sale delivery o teste está completo por isso só resta apagar tudo o que foi introduzido (customer, address, sale e sale delivery) e verificar a invariante do sistema.

```
// DEPOIS DE VERIFICAR TUDO APAGAR DELIVERY SALE, SALE e CUSTOMER
// APAGAR SALE DELIVERY
DomElement deleteAllSaleDeliveryMenuButton = page.getElementsById("botao2").get(13);
HtmlPage deleteAllSaleDeliveryPage = deleteAllSaleDeliveryMenuButton.click();
HtmlForm deleteAllSaleDeliveryform = deleteAllSaleDeliveryPage.getForms().get(0);
deleteAllSaleDeliveryform.getInputByName("customerVat").setValueAttribute(VAT);
HtmlSubmitInput deleteAllSaleDeliveryButton = deleteAllSaleDeliveryform.getInputByValue("Delete All Sale Delivery");
deleteAllSaleDeliveryButton.click();

// APAGAR SALE
DomElement deleteLastSaleMenuButton = page.getElementsById("botao2").get(11);
HtmlPage deleteLastSalePage = deleteLastSaleMenuButton.click();
HtmlForm deleteLastSaleform = deleteLastSalePage.getForms().get(0);
deleteLastSaleform.getInputByName("customerVat").setValueAttribute(VAT);
HtmlSubmitInput deleteLastSaleButton = deleteLastSaleform.getInputByValue("Delete Last Sale");
deleteLastSaleButton.click();

//DELETE DE TODOS OS ADDRESS DO CUSTOMER
DomElement deleteAllAddressFromCustomerButton = page.getElementsById("botao2").get(12);
HtmlPage deleteAllAddressFromCustomerPage = deleteAllAddressFromCustomerButton.click();
HtmlForm deleteAllAddressFromCustomerForm = deleteAllAddressFromCustomerPage.getForms().get(0);
deleteAllAddressFromCustomerForm.getInputByName("customerVat").setValueAttribute(VAT);
HtmlSubmitInput deleteAllAddressButton = deleteAllAddressFromCustomerForm.getInputByValue("Delete All Address");
deleteAllAddressButton.click();

//Remover customer inserido
DomElement removeCustomerButton = page.getElementsById("botao2").get(2);
HtmlPage removeCustomerPage = removeCustomerButton.click();
HtmlForm removeCustomerform = removeCustomerPage.getForms().get(0);
removeCustomerform.getInputByName("vat").setValueAttribute(VAT);
HtmlSubmitInput removeButton = removeCustomerform.getInputByValue("Remove");
removeButton.click();
```

```
//VERIFICAR INVARIANTE DOS CUSTOMERS
clientInfoPage9 = (HtmlPage) clientInfoPage9.refresh();
rows9 = (List<DomElement>) clientInfoPage9.getElementsByTagName("tr");
int finalNumberOfAddresses = rows9.size()-1;

assertEquals(actualNumberOfAddresses, finalNumberOfAddresses);


//VERIFICAR INVARIANTE DOS SALES
showCustomersSaleButton8 = page.getElementsById("botao2").get(8);
assertEquals("Show Customer Sale's", showCustomersSaleButton8.asText());
showCustomersSalePage8 = showCustomersSaleButton8.click();
showCustomersSaleform8 = showCustomersSalePage8.getForms().get(0);
showCustomersSaleform8.getInputByName("customerVat").setValueAttribute(VAT);
showCustomersSaleButtonForm8 = showCustomersSaleform8.getInputByValue("Get Sales");
saleInfoPage8 = showCustomersSaleButtonForm8.click();
rows8 = (List<DomElement>) saleInfoPage8.getElementsByTagName("tr");
int numberOfSalesForThatCustomerFinal = rows8.size() - 1;
assertEquals(numberOfSalesForThatCustomer8, numberOfSalesForThatCustomerFinal);


//Numero de sale deliveries atual
showCustomersSaleDeliveriesButton7 = page.getElementsById("botao2").get(8);
showCustomersSaleDeliveriesPage7 = showCustomersSaleDeliveriesButton7.click();
showCustomersSaleDeliveriesform7 = showCustomersSaleDeliveriesPage7.getForms().get(0);
showCustomersSaleDeliveriesform7.getInputByName("customerVat").setValueAttribute(VAT);
showCustomersSaleDeliveriesButtonForm7 = showCustomersSaleDeliveriesform7.getInputByValue("Get Sales");
saleDeliveriesInfoPage7 = showCustomersSaleDeliveriesButtonForm7.click();
rows7 = (List<DomElement>) saleDeliveriesInfoPage7.getElementsByTagName("tr");
int numberOfSaleDeliveriesForThatCustomerFinal7 = rows7.size() - 1;

assertEquals(numberOfSaleDeliveriesForThatCustomer7, numberOfSaleDeliveriesForThatCustomerFinal7);
```

DBSetup

Primeira mente precisamos de popular a base de dados com alguns valores para conseguir testar o código, os valores inseridos na base de dados foram:
Para a tabela customer:

```
Insert insertCustomers =
    insertInto("CUSTOMER")
    .columns("ID", "DESIGNATION", "PHONENUMBER", "VATNUMBER")
    .values(   1,   "JOSE FARIA",     914276732,   197672337)
    .values(   2,  "LUIS SANTOS",     964294317,   168027852)
    .build();

NUM_INIT_CUSTOMERS = insertCustomers.getRowCount();
```

Para a tabela address:

```
Insert insertAddresses =
       insertInto("ADDRESS")
       .withGeneratedValue("ID", ValueGenerators.sequence().startingAt(100L).incrementingBy(1))
       .columns(                              "ADDRESS", "CUSTOMER_VAT")
       .values(           "FCUL, Campo Grande, Lisboa",      197672337)
       .values(          "R. 25 de Abril, 101A, Porto",      197672337)
       .values( "Av Neil Armstrong, Cratera Azul, Lua",      168027852)
       .build();

NUM_INIT_ADDRESSES = insertAddresses.getRowCount();
```

Para a tabela sale:

```
Insert insertSales =
       insertInto("SALE")
       .withGeneratedValue("ID", ValueGenerators.sequence().startingAt(1).incrementingBy(1))
       .columns("DATE", "TOTAL", "STATUS", "CUSTOMER_VAT")
       .values(new Date(118, 0, 2), 0.0, 'O', 197672337)
       .values(new Date(118, 0, 2), 0.0, 'O', 197672337)
       .build();


NUM_INIT_SALES = insertSales.getRowCount();
```

Para a tabela delivery:

```
Insert insertSaleDeliveries =
       insertInto("SALEDELIVERY")
       .withGeneratedValue("ID", ValueGenerators.sequence().startingAt(1).incrementingBy(1))
       .columns("SALE_ID", "CUSTOMER_VAT", "ADDRESS_ID")
       .values(1, 197672337, 1)
       .values(2, 197672337, 2)
       .build();


NUM_INIT_SALE_DEVIVERIES = insertSaleDeliveries.getRowCount();
```

a) the SUT does not allow to add a new client with an existing VAT

Para este testei foi feito um assertThrows sobre a adição de um novo Customer.

```
@Test
public void testDuplicateClient() throws ApplicationException {
    assertThrows(ApplicationException.class, () -> CustomerService.INSTANCE.addCustomer(197672337, "JOSE FARIA", 914276732));
}
```

b) after the update of a customer contact, that information should be properly saved

É utilizado o CustomerService para alterar o número de telefone de um customer, após a alteração ser feita uma comparação é realizada para certificar que o uptade foi bem realizado

(PRINT)

c) after deleting all costumers, the list of all customers should be empty

É feita a busca por todos os utilizadores, após obter a lista de utilizadores existentes é feita a remoção através do VAT 1 por 1. Após a remoção obtemos outra vez a lista de utilizadores e verificamos se essa lista está vazia

```
@Test
public void uptadeCustomerContact() throws ApplicationException {
    int newPhone = 961234567;
    CustomerService.INSTANCE.updateCustomerPhone(197672337, newPhone);
    CustomerDTO updatedCustomer = CustomerService.INSTANCE.getCustomerByVat(197672337);
    assertEquals(newPhone, updatedCustomer.phoneNumber);
}
```

d) after deleting a certain costumer, it's possible to add it back without lifting exceptions

Para este teste é apagado um customer que já existe na base de dados e depois esse customer é voltado a ser inserido. No final faz se um getCustomerByVat através do CustomerService e são verificados os campos do customer para certificar que ele foi bem inserido de novo.

```
@Test
public void testDeleteAllCustomers() throws ApplicationException {
    CustomersDTO allCustomers = CustomerService.INSTANCE.getAllCustomers();
    assertTrue(allCustomers.customers.size() > 0);
    for (CustomerDTO customer : allCustomers.customers) {
        CustomerService.INSTANCE.removeCustomer(customer.vat);
    }
    CustomersDTO allCustomersAfter = CustomerService.INSTANCE.getAllCustomers();
    allCustomersAfter.customers.size();
    assertTrue(allCustomersAfter.customers.size() == 0);
}
```

e) after deleting a certain costumer, its sales should be removed from the database

Para a realização deste teste foi necessário implementar a deleção das sales de um certo customer quando esse customer é apagado.

Para a implementação da deleção das sales foi adicionado o seguinte código ao CustomerRowDataGateway.java:

```
// ADICIONADO POR MIM PARA APAGAR OS SALES QUANDO APAGA O USER
private static final String REMOVE_SALES_BY_CUSTOMER_VAT = "DELETE FROM sale " + "WHERE customer_vat = ?";
```

```
// ADICIONADO POR MIM PARA APAGAR OS SALES QUANDO APAGA O USER
try (PreparedStatement statement = DataSource.INSTANCE.prepare(REMOVE_SALES_BY_CUSTOMER_VAT)) {
    statement.setInt(1, vat);
    statement.executeUpdate();
} catch (SQLException e) {
    throw new PersistenceException("Internal error updating customer " + id + ".", e);
}
```

Após a deleção ser implementada, no teste é apagado um customer e é verificado se o número de sale desse customer estão a 0.

```
@Test
public void testDeleteCertainCustomerAndItsSales() throws ApplicationException {
    CustomerService.INSTANCE.removeCustomer(197672337);
    SalesDTO sales2  = SaleService.INSTANCE.getSaleByCustomerVat(197672337);
    assertTrue(sales2.sales.size() == 0);
}
```

f) adding a new sale increases the total number of all sales by one

Para este teste foi utilizado o SaleService para adicionar uma nova sale e foi comparado se o número total de sales aumentou.

```
@Test
public void testAddSaleTotalSaleIncrease() throws ApplicationException {
    int totalNumSales = SaleService.INSTANCE.getAllSales().sales.size();
    SaleService.INSTANCE.addSale(197672337);
    assertEquals(totalNumSales + 1, SaleService.INSTANCE.getAllSales().sales.size());
}
```

g) 2 extra tests concerning the expected behavior of sales

1. It is not possible to close a sale that doesn't exist

Foi utilizado um assertThrows para apanhar e exceção lancada pelo programa

```
@Test
public void testExpectedBehaviourForSales1() throws ApplicationException {
    //It is not possible to close a sale that doesnt exist!
    assertThrows(ApplicationException.class, () -> SaleService.INSTANCE.updateSale(5));

}
```

2. When a sale is closed it should change the status

Mudamos o estado de uma sale já existente na base de dados e depois de mudarmos o estado da sale verificamos se o estado é igual a "C", que significa Closed.

```
@Test
public void testExpectedBehaviourForSales2() throws ApplicationException {
    //when a sale is closed it should change the status
    SaleService.INSTANCE.updateSale(1);
    SalesDTO sales  = SaleService.INSTANCE.getAllSales();
    assertTrue(sales.sales.get(0).statusId.equals("C"));

}
```

h) 2 extra tests concerning the expected behavior of sales deliveries

1. After adding a sale delivery the number of sale deliveries should increase

Foi adicionada uma sale delivery a um certo customer e no final do teste é verificado se o número de sale deliveries aumentou em 1.

```
@Test
public void testExpectedBehaviourForSaleDeliveries1() throws ApplicationException {
    //after adding a sale delivery the number of sale deliveries should increase
    int numSaleDeliveries = SaleService.INSTANCE.getSalesDeliveryByVat(197672337).sales_delivery.size();
    SaleService.INSTANCE.addSaleDelivery(1, 1);
    int finalNumSaleDeliveries = SaleService.INSTANCE.getSalesDeliveryByVat(197672337).sales_delivery.size();
    assertEquals(numSaleDeliveries+1, finalNumSaleDeliveries);
}
```

2. After deleting a customer its sale deliveries should be deleted too

Para a realização deste teste foi necessário implementar a deleção das sales deliveries de um certo customer quando esse customer é apagado.

Para a implementação das sales deliveries foi adicionado o seguinte código ao CustomerRowDataGateway:

```
// ADICIONADO POR MIM PARA APAGAR OS SALES DELIVERIES QUANDO APAGA O USER
private static final String REMOVE_SALE_DELIVERIES_BY_CUSTOMER_VAT = "DELETE FROM saledelivery "
        + "WHERE customer_vat = ?";
```

```
// ADICIONADO POR MIM PARA APAGAR OS SALE DELIVERIES QUANDO APAGA O USER
try (PreparedStatement statement = DataSource.INSTANCE.prepare(REMOVE_SALE_DELIVERIES_BY_CUSTOMER_VAT)) {
    statement.setInt(1, vat);
    statement.executeUpdate();
} catch (SQLException e) {
    throw new PersistenceException("Internal error updating customer " + id + ".", e);
}
```

Após a deleção ser implementada, no teste é apagado um customer e é verificado se o número de sale deliveries desse customer estão a 0.

```
@Test
public void testExpectedBehaviourForSaleDeliveries2() throws ApplicationException {
    //after deleting a customer its sale deliveries should be deleted too
    int numSaleDeliveries = SaleService.INSTANCE.getSalesDeliveryByVat(197672337).sales_delivery.size();
    assertTrue(numSaleDeliveries > 0);
    CustomerService.INSTANCE.removeCustomer(197672337);
    int finalNumSaleDeliveries = SaleService.INSTANCE.getSalesDeliveryByVat(197672337).sales_delivery.size();
    assertTrue(finalNumSaleDeliveries == 0);
}
```

Mockito

Neste projeto apenas temos 2 serviços e ambos são independentes uns dos outros. Mas existe uma dependência no que toca aos serviços e aos controllers, onde os controllers dependem do serviço para o seu funcionamento.

Logo numa primeira visão seria possível realizar um mock de um modulo da business layer para remover essa dependência entre serviços e controllers, mas da forma como o programa esta implementado as classes serviços, o CustomerService e o SaleService, são enumerados e em java um enumerado é implicitamente uma classe final, significando que não é possível realizar o mock de nenhum dos serviços existentes no projeto.

Para conseguir implementar o mocking no projeto seria necessário mudar o tipo da classe dos serviços para uma classe normal. Tendo os serviços como uma classe normal seria possível implementar o mock da seguinte forma:

```
CustomerService mock = mock(CustomerService.class);

LinkedList<CustomerDTO> customerList = new LinkedList<>(Arrays.asList(
        new CustomerDTO(0, 123456789, "teste1", 961234567),
        new CustomerDTO(0, 197672337, "teste2", 961234565),
        new CustomerDTO(0, 168027852, "teste3", 961234563),
        new CustomerDTO(1, 269065822, "teste4", 987654321)
));

when(mock.getAllCustomers()).thenReturn(new CustomersDTO(customerList));
```

Neste caso, o mock foi feito ao CustomerService e a único método onde foi feito o mock foi o getAllCustomers(), que é o método utilizado pelo GetAllCustomersPageController.java

Bugs

Os bugs que foram encontrados ao longo dos testes realizado com o projeto foram:

- Ao introduzir um address a um customer, o address fica guardado com um numero inesperado de espaços:
    - Correção: Alterar a query e realizar logo um trim ao address

```
private static final String GET_ADDRESS_BY_CUSTOMER_VAT_SQL =
        "SELECT *, TRIM(address) AS address " +
        "FROM address " +
        "WHERE customer_Vat = ?";
```

- Ao preencher os campos quando se introduz um novo endereço, se alguns dos campos, exceto o último for preenchido com pelo menos 1 “;”, o display da informação irá ficar mal formatado:
    - Correção: Adição desta condição ao programa, para detetar ‘;’.

```
if (isInt(ch, vat, "Invalid VAT number") || isInt(ash, vat, "Invalid VAT number")) {
    int vatNumber = intValue(vat);
    ch.fillWithCustomer(cs.getCustomerByVat(vatNumber));
    if(address != null) {
        if(!address.contains(";") && !postalCode.contains(";") && !door.contains(";")) {
            cs.addAddressToCustomer(vatNumber, (address + ";" + door + ";" + postalCode + ";" + locality));
        } else {
            ch.addMessage("Cannot include ';' in the inputs");
            request.getRequestDispatcher("CustomerError.jsp").forward(request, response);
        }

    }
```

- Ao apagar um Customer as suas sales permanecem no sistema:
    - Correção: Já foi explicado anteriormente no relatório como foi resolvido este bug
- Ao apagar um CUstomer as suas sale deliveries permanecem no sistema:
    - Correção: Já foi explicado anteriormente no relatório como foi resolvido este bug
- É possível atribuir um VAT que não exista ou que não esteja presente no sistema a uma nova sale
    - Correção: Foi necessário criar um método para verificar se um certo utilizador já existe, caso não exista, ao inserir uma sale lança um erro

```
public boolean existCustomer(int VAT) throws ApplicationException {
    try {
        return CustomerRowDataGateway.customerExists(VAT);
    } catch (PersistenceException e) {
            throw new ApplicationException ("Can't delete the last sale from that cutomer.", e);
    }
}
```

```
public static boolean customerExists(int vat) throws PersistenceException {
    try (PreparedStatement statement = DataSource.INSTANCE.prepare(CUSTOMER_EXISTS_QUERY)) {
        statement.setInt(1, vat);
        try (ResultSet resultSet = statement.executeQuery()) {
            return resultSet.next();
        }
    } catch (SQLException e) {
        throw new PersistenceException("Error checking if customer exists", e);
    }
}
```

```
public void addSale(int customerVat) throws ApplicationException {
    try {

        if (!CustomerService.INSTANCE.existCustomer(customerVat)) {
            throw new ApplicationException("Customer with VAT number " + customerVat + " does not exist.");
        }

        SaleRowDataGateway sale = new SaleRowDataGateway(customerVat, new Date());
        sale.insert();

    } catch (PersistenceException e) {
        throw new ApplicationException("Can't add customer with vat number " + customerVat + ".", e);
    }
}
```

- É possível iniciar uma sale delivery com um endereço que não exista
    - Correção: É necessário verifica se o id do address introduzido está presente na tabela
- É possível iniciar uma sale delivery com uma sale que está fechada. Neste caso não seria bem um erro, teríamos que verificar os requerimentos do projeto para saber se era suposto o sistema ter este comportamento
- Quando adicionado uma sale ou sale delivery nova, se dermos f5 na pagina é possível introduzir uma nova sale ou sale delivery com as mesmas informações. Neste caso não seria bem um erro, teríamos que verificar os requerimentos do projeto para saber se era suposto o sistema ter este comportamento
- É possível adicionar um customer sem nome
    - Correção: Foi adicionado esta condição no ficheiro, AddCustomerPageController.java

```
if (isInt(ch, vat, "Invalid VAT number") && isInt(ch, phone, "Invalid phone number") && !designation.equals("")) {
    int vatNumber = intValue(vat);
    int phoneNumber = intValue(phone);
    cs.addCustomer(vatNumber, designation, phoneNumber);
    ch.fillWithCustomer(cs.getCustomerByVat(vatNumber));
    request.getRequestDispatcher("CustomerInfo.jsp").forward(request, response);
}
```

- É possível atribuir um número de telefone negativo a um customer, quer a introduzir um novo customer, quer a atualizar o número de telefone de um customer.
    - Correção: Foi adicionado esta condição no ficheiro, AddCustomerPageController.java

```
if (isInt(ch, vat, "Invalid VAT number") && isInt(ch, phone, "Invalid phone number")
        && !designation.equals("") && Integer.parseInt(phone)>0) {
```

- É possível atribuir um endereço vazio a customer.
    - Correção: Foi adicionado esta condição no ficheiro, GetCustomerPageController.java

```
if(address != null) {
    if(!address.contains(";") && !postalCode.contains(";") && !door.contains(";") && !address.isEmpty()) {
```

- Falta de mensagens de erro quando introduzido um VAT invalido para remover um customer
    - Correção: Adicionar mensagens de erro
- Possivel visualizar as sales de um customer que não existe, mas com VAT válido
    - Correção: O mesmo processo feito para a adição de uma sale
- Erros gerais na introdução de um novo endereço a um customer, o sistema não faz verificação das informações a serem introduzidas, por isso podem ser introduzidas informações duplicadas ou mesmo informações totalmente incorretas

    - Correção: Fazer verificação dos dados introduzidos com os dados presentes na base de dados
- Uma sale delivery pode ser inicializada com um VAT válido, mas inexistente na base de dados
    - Correção: Fazer verificação para saber se o VAT existe na base de dados
- Possível aceder à página de sale deliveries de um utilizador, mesmo que ele não exista no sistema.
    - Correção: Fazer verificação para saber se o VAT existe na base de dados