Verificação e Validação de Software
2021/2022
Mestrado em Engenharia Informática

Assignment 2
WebApp Demo Testing

Tomás Dias nº57582

Estrutura do projeto

Na diretoria *src/test/java* encontram-se os packages e respetivas classes:

- **webapp**
  - *WebAppNarrativesTest*: classe que contém os testes referentes aos casos de uso da aplicação.
  - *PopulatedDatabaseTest*: classe que contém os testes que trabalham com uma porção de dados inserida na base de dados.
  - *MocksBasedTest*: classe que contém os testes alvos de *mocking*.
- **webapp.utils**
  - *DBSetupUtils*: classe responsável pela configuração da base de dados e das suas operações.

Na diretoria *src/main/java*, foi adicionado o package **webapp.refactoring** com as classes da aplicação que foram sujeitas a refactoring do código fonte.

De salientar também que a aplicação foi executada num servidor *Wildfly*.

Desenvolvimento

Testes dos *use cases* da aplicação utilizando *web scraping*

Para a implementação dos testes que tinham como objetivo testar as diferentes narrativas da aplicação, utilizou-se o processo de *web scraping* através da ferramenta *HtmlUnit*.

Com o HtmlUnit, foi possível simular um *web browser* através da classe *WebClient*, fundamental para o scraping das páginas da aplicação.

```
final WebClient webClient = new WebClient(BrowserVersion.getDefault())
```

Porção do código fonte referente à inicialização da classe WebClient.

Foram feitas algumas configurações ao *web client* de prevenção a falhas dos testes bem como verificações do seu funcionamento (ex: se obtinha uma resposta de uma página).

O fluxo de acontecimentos de cada teste realizado encontra-se descrito de seguida.

2.a) *insertCustomerAddressTest()*

O método responsável por testar a inserção de um endereço de um cliente tem as seguintes etapas:

1) Verificação da existência do cliente ao qual vai ser associado o endereço no sistema.
2) Navegação até à página do cliente de modo a obter o número atual de endereços. Esta navegação é feita através de um pedido *HTTP* à página de informação do cliente, utilizando a classe *WebRequest*.

```
// Check customer related data
req = new WebRequest(new java.net.URL(APPLICATION_URL+"GetCustomerPageController"), HttpMethod.GET);
req.setRequestParameters(asList(new NameValuePair("vat", NPC)));
```

Porção do código fonte referente à definição do request a ser feito à página de informação do cliente.

```
try (final WebClient webClient = new WebClient(BrowserVersion.getDefault())) {
    nextPage = (HtmlPage) webClient.getPage(req);
}
```

Porção do código fonte onde é obtida a página resultante do request.

```
// Xpath for table
xpath = "/html/body//table";

// Get customer addresses table size
tableSize = 0;
if(nextPage.getByXPath(xpath).size() != 0) {
    table = (HtmlTable) nextPage.getByXPath(xpath).get(0);
    tableSize = table.getRowCount() - 1; // without header
}
```

Porção do código fonte referente ao modo de como é obtido o número de endereços atual de um cliente.

3) Preenchimento do formulário com os campos do endereço e envio para o sistema (através de um pedido *HTTP*).
4) Verificação se o endereço foi adicionado (atualização da tabela de endereços na página do cliente).

```
assertTrue(textNextPage.contains(ADDRESS));
assertTrue(textNextPage.contains(DOOR));
assertTrue(textNextPage.contains(POSTAL_CODE));
assertTrue(textNextPage.contains(LOCALITY));

// Get new customer addresses table size
table = (HtmlTable) nextPage.getByXPath(xpath).get(0);
newTableSize = table.getRowCount() - 1;

assertEquals(tableSize + 1, newTableSize);
```

Porção do código fonte referente aos asserts que verificam se o novo endereço persiste no sistema.

2.b) *newSaleListedAsOpenTest()*

O método responsável por testar se uma nova venda é listada como aberta tem as seguintes etapas:

1) Verificação da existência do cliente ao qual vai ser associado o endereço no sistema.
2) Navegação até à página de adição de uma venda.
3) Preenchimento e submissão do formulário.
4) Navegação até à página onde se encontram listadas as vendas de cada cliente e verificação do estado da venda adicionada mais recente.

```
// Check new sale status
xpath = "/html/body//table";
table = (HtmlTable) nextPage.getByXPath(xpath).get(0);
row = table.getRow(table.getRowCount() - 1);
status = row.getCell(3).asText();

assertEquals("0", status);
```

Porção do código fonte onde é verificado o estado da venda adicionada mais recente.

2.c) *saleListedAsClosedTest()*

O método responsável por testar se após o fecho de uma venda esta aparece listada como fechada tem as seguintes etapas:

1) Verificação da existência do cliente ao qual vai ser associado o endereço no sistema.

2) Navegação até à página de adição de uma venda.
3) Preenchimento do formulário para adição de uma venda e envio para o sistema (através de um pedido *HTTP*).
4) Obtenção do identificador da venda (através da página obtida após o pedido).

```
// Get new sale id
xpath = "/html/body//table";
table = (HtmlTable) nextPage.getByXPath(xpath).get(0);
row = table.getRow(table.getRowCount() - 1);
id = row.getCell(0).asText();
```

Porção do código fonte onde é salvaguardado o id da venda adicionada.

5) Envio do pedido com o identificador da venda adicionado para o controlador responsável pela atualização do estado da venda.
6) Verificação se o estado da venda foi atualizado.

```
// Check new sale updated status
table = (HtmlTable) nextPage.getByXPath(xpath).get(0);
row = table.getRow(table.getRowCount() - 1);
status = row.getCell(3).asText();

assertEquals("C", status);
```

Porção do código fonte onde é verificado se o estado da venda foi atualizado.

2.d) *insertSaleDeliveryTest()*

O método responsável por testar a inserção de uma entrega tem as seguintes etapas:

1) Envio do pedido com a informação para a adição de um cliente para o controlador responsável. É verificado se a informação do cliente está presente na página obtida após o pedido.
2) Envio do pedido com a informação para a adição de um endereço para o controlador responsável. É verificado se a informação do endereço está presente na página obtida após o pedido.
3) Envio do pedido com a informação para a adição de uma venda para o controlador responsável. É verificado se a informação da venda está presente na página obtida após o pedido.
4) Envio de pedido para o controlador responsável pela adição de uma entrega de modo a obter os identificadores da venda e do endereço do cliente.

```
// Get last address id
xpath = "/html/body//table";
table = (HtmlTable) nextPage.getByXPath(xpath).get(0);
row = table.getRow(table.getRowCount() - 1);
addr_id = row.getCell(0).asText();

// Get last sale id
table = (HtmlTable) nextPage.getByXPath(xpath).get(1);
row = table.getRow(table.getRowCount() - 1);
sale_id = row.getCell(0).asText();
```

Porção do código fonte onde são salvaguardados os ids do endereço do cliente e da venda adicionada.

5) Envio de pedido para o controlador responsável pela adição de uma entrega com a informação do endereço e da venda.
6) Verificação se a entrega foi adicionada.

Testes com manipulação da base de dados

Os dados a serem utilizados nos testes foram organizados com o auxílio da ferramenta *DBSetup*. Toda a configuração relacionada com a base de dados encontra-se na classe *DBSetupUtils*.

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

Porção do código fonte onde são definidos os dados relacionados com os clientes a serem inseridos na base de dados para testes.

```
Insert insertSales =
    insertInto("SALE")
    .columns("ID",                          "DATE", "TOTAL", "STATUS", "CUSTOMER_VAT")
    .values(   1,  new GregorianCalendar(2018,01,02),     0.0,      'O',      197672337)
    .values(   2,  new GregorianCalendar(2017,03,25),     0.0,      'O',      197672337)
    .build();
```

Porção do código fonte onde são definidos os dados relacionados com as vendas a serem inseridos na base de dados para testes.

```
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();
```

Porção do código fonte onde são definidos os dados relacionados com os endereços dos clientes a serem inseridos na base de dados para testes.

```
Insert insertDeliveries =
        insertInto("SALEDELIVERY")
        .columns("ID",                                "SALE_ID", "CUSTOMER_VAT", "ADDRESS_ID")
        .values(1, 1, 197672337, 1)
        .build();
```

Porção do código fonte onde são definidos os dados relacionados com as entregas dos clientes a serem inseridos na base de dados para testes.

```
INSERT_CUSTOMER_ALL_DATA = sequenceOf(insertCustomers, insertAddresses, insertSales, insertDeliveries);
```

Porção do código fonte onde são combinados todos os dados a serem inseridos na base de dados para testes.

Os testes implementados foram os seguintes:

- *addNewClientWithAnExistingVATTest()*: testa se o sistema não permite a inserção de um cliente com um VAT existente. (3.a)
- *updateClientContactTest()*: testa se a informação de um contacto de um cliente persiste no sistema quando atualizada. (3.b)
- *deleteAllClientsTest()*: testa se quando são eliminados todos os clientes do sistema, a lista de clientes fica vazia. (3.c)
- *addNewSaleTest()*: testa se quando é adicionada uma nova venda, o número de vendas total é incrementado em um. (3.d)
- *retrieveAllSalesTest()*: testa se todas as vendas são obtidas. (3.e)
- *updateSaleTest()*: testa se a informação sobre o estado de uma venda de um cliente persiste no sistema quando atualizada. (3.f)
- *retrieveAllSalesDeliveriesByCustomerTest()*: testa se todas as entregas de um cliente são obtidas. (3.g)
- *addNewSaleDeliveryTest():* testa se quando é adicionada uma nova entrega, o número de entregas total do cliente é incrementado em um. (3.h)

Testes baseados em *mocks* (*mocking*)

Após terem sido estudado os diferentes módulos da camada de negócio da aplicação, optou-se por testar o serviço *CustomerService*.

Tomando como exemplo o método *addCustomerAddress()*. Este método, à semelhança dos restantes métodos do serviço, tem como dependência uma classe *row data gateway* de acesso à base de dados. Por isso, a abordagem tomada foi a de criar um mock para a classe *AddressRowDataGateway*.

De modo a ser possível injetar o mock para o serviço alvo de teste, foi necessário realizar um refactoring no código fonte.

Alterou-se o tipo do *CustomerService* de *enum* para *class* e adicionou-se um construtor que permite a injeção do *mock object*.

```
private CustomerRowDataGateway customer;
private CustomerFinder finder;
private AddressRowDataGateway address;

public NewCustomerService(
        CustomerRowDataGateway customer,
        CustomerFinder finder,
        AddressRowDataGateway address) {
    this.customer = customer;
    this.finder = finder;
    this.address = address;
}
```

Porção do código fonte referente ao construtor da classe NewCustomerService (classe baseada no serviço CustomerService utilizada apenas para os testes baseados em mocks).

Os restantes testes da classe alvo de *mocking NewCustomerService* podem ser encontrados na classe *MocksBasedTest*.

Falhas encontradas no sistema e possíveis soluções

Durante o desenvolvimento deste trabalho foram encontrados problemas que metem em causa a fiabilidade, consistência e bom funcionamento do sistema. Todas essas falhas foram devidamente reportadas no projeto partilhado em https://57855.backlog.com/.

Uma maioria significativa das falhas estão relacionadas com a falta de verificação por parte dos serviços da aplicação no tipo de dados que é passado para a base de dados.

Exemplificando, quando pretendemos adicionar um cliente, é possível adicionar um número de telemóvel que não segue as normas de composição dos números de telemóvel. Outro exemplo é quando pretendemos adicionar um endereço a um cliente, os campos podem ser preenchidos com valores nulos.
Outras falhas recorrentes estão relacionadas com associações erradas entre os principais conceitos da aplicação (cliente, endereços, vendas e entregas).

Exemplo disso, é a possibilidade de adicionar uma venda com um VAT de um cliente que não existe no sistema ou adicionar um entrega de uma venda de um outro cliente que não nós para um endereço que não persiste no sistema.

Isto são apenas alguns exemplos das muitas falhas expostas no projeto de *bugtracking*.

A solução para a maioria destes *bugs* passaria pela implementação ao nível da camada de negócio, ou seja, dos serviços da aplicação, de verificações e validações do tipo de dados introduzidos pelos utilizadores. Garantir também, através da comunicação com a camada de persistência, que os dados recebidos não alteram a consistência do sistema nem causam problemas ao nível da segurança (ex: não permitir associações de endereços, vendas ou entregas a clientes que não persistem no sistema).