Candidatura Multicert

Elói José de Almeida Geria

914187063

eloidealmeida@gmail.com

1. FORMATO DA ENTREGA

O Código do exercício é entregue através do sistema de controlo de versões Github, que pode ser acedido pelo seguinte URL:

https://github.com/eloialmeida/repo.git

O Serviço tem o Soap Binding no seguinte URL:

http://eloidns.cloudapp.net:8080/app-inbound-adapters-soap/

Finalmente o seu WSDL encontra-se no seguinte endereço:

http://eloidns.cloudapp.net:8080/MULTICERT_WSDL/service.wsdl

Nota: Não usar o WSDL que se encontra em *http://eloidns.cloudapp.net:8080/app-inbound-adapters-soap/?wsdl*, pois possui os bindings incorretos.

2. RESUMO DA SOLUÇÃO

A solução desenvolvida é composta pelos seguintes módulos:

2.1. Multicert Model: Modelação das entidades do serviço (Cliente.java).
2.2. Multicert Core: Camada de Negócio da Aplicação e das suas interfaces. Incorpora o Bean do serviço GestaoLocalBean, e a camada de acesso a dados é servida pelo Bean MulticertDao.

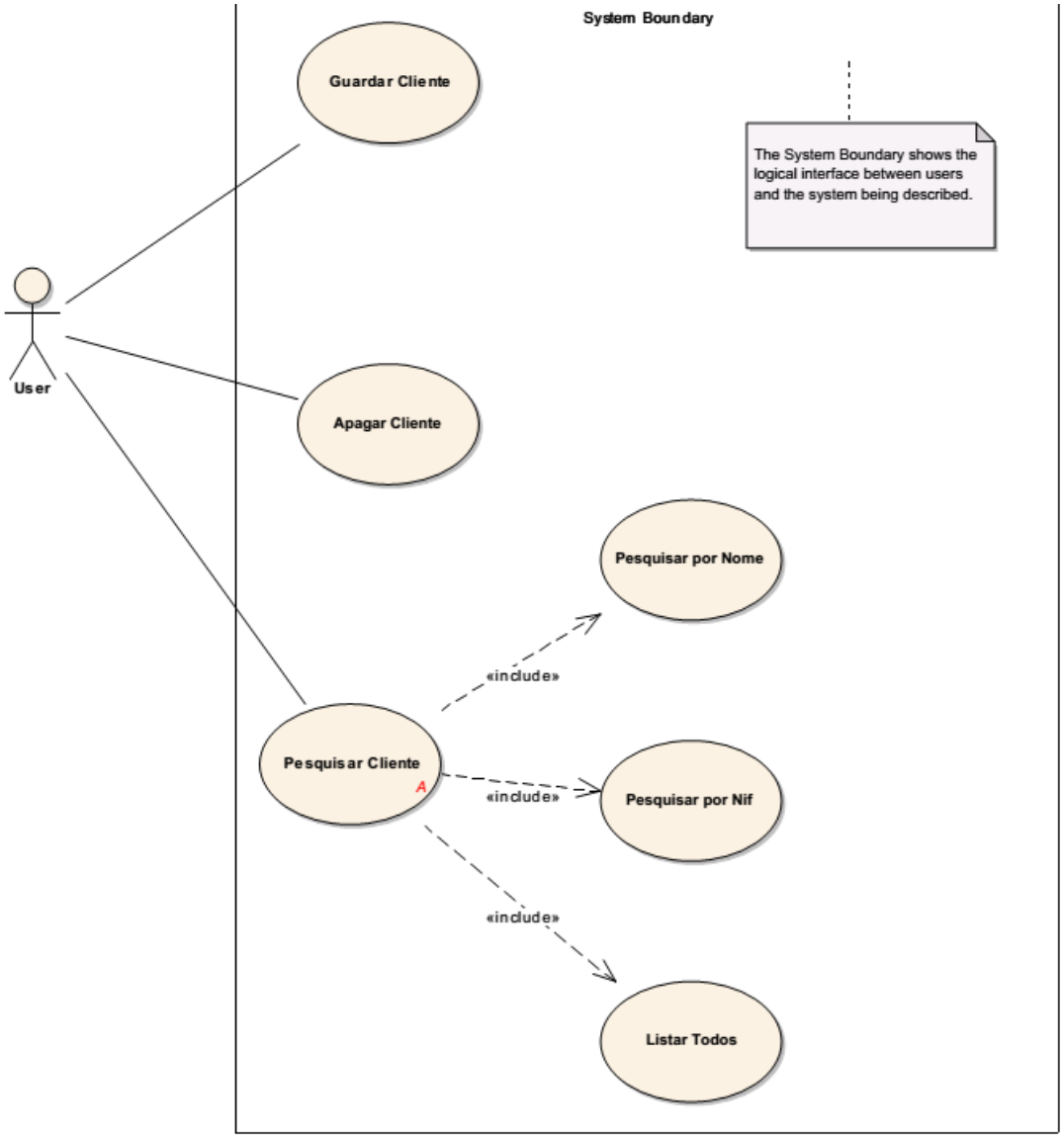
A camada de dados apresenta a seguinte stack:

MulticertDao
JPA, Provided by Hibernate
MySQL

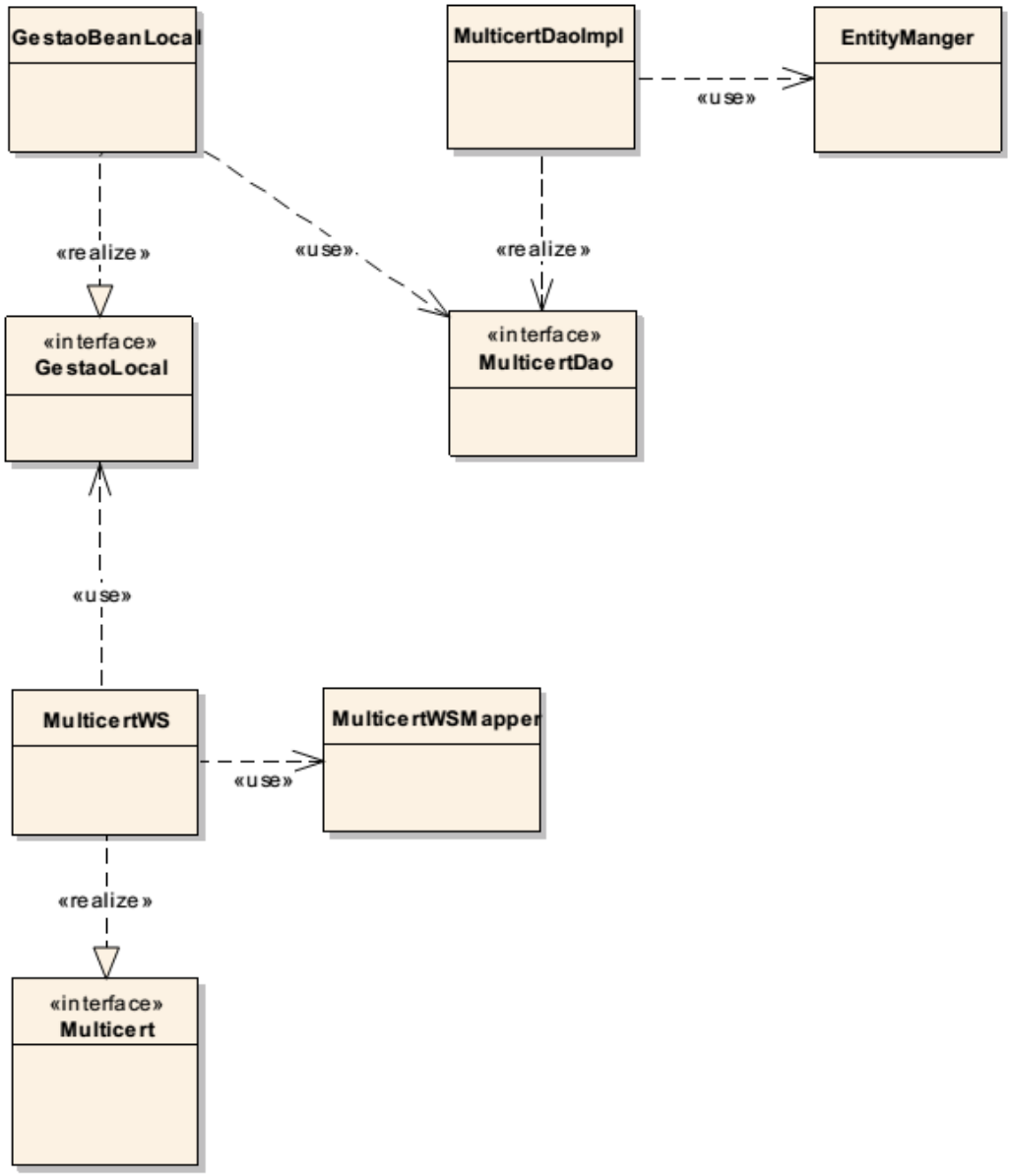
2.3. Multicert Network Adaptars: Camada da definição do end point do serviço para o exterior, via SOAP. Possui todos os artefactos gerados via Apache CXF a partir da definição do WSDL como contracto. Esta camada é cliente da camada Multicert Core.
2.4. Módulo EAR: Enterprise Archive que agrega todos os módulos.
2.5. Módulo APP: Raiz da estrutura Maven.

3. DIAGRAMAS UML

3.1. DIAGRAMA DE CASOS DE USO

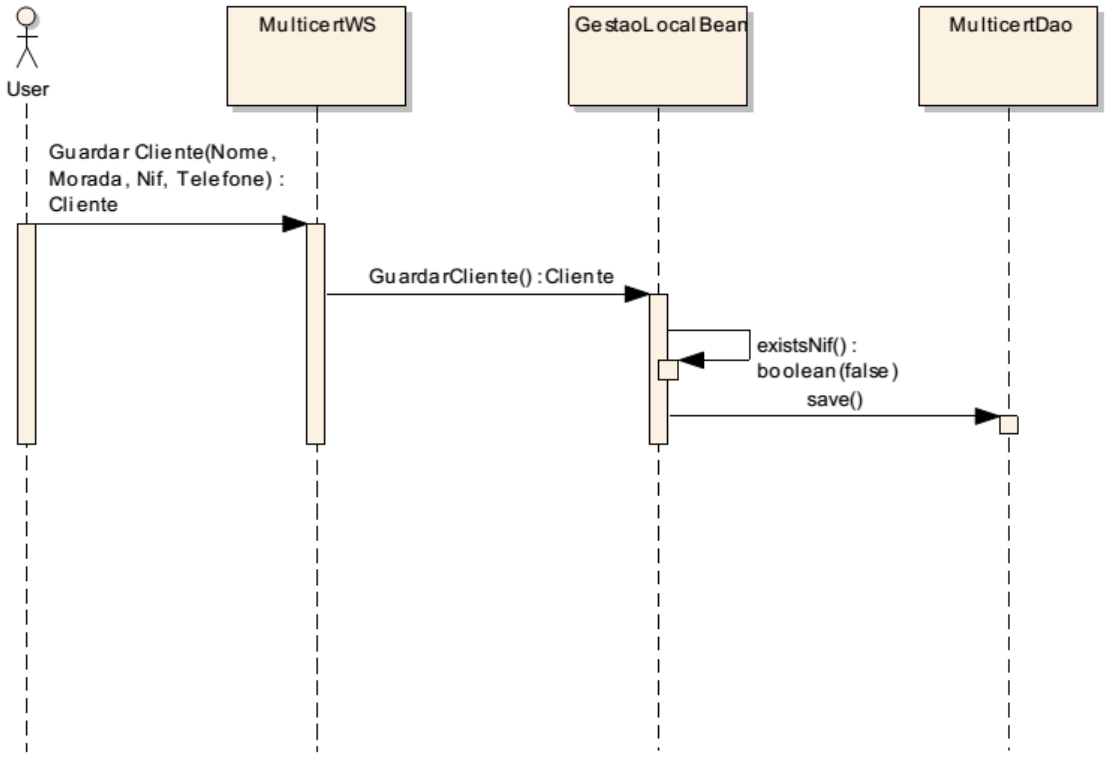


3.2. DIAGRAMA DE CLASSES

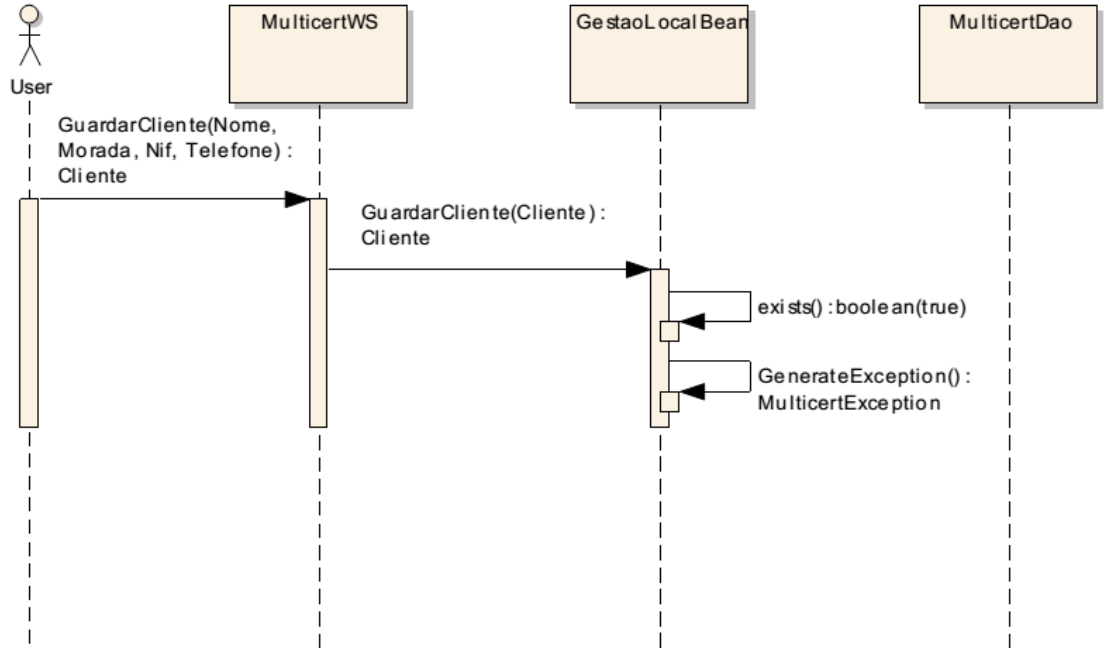


3.3. DIAGRAMAS DE SEQUENCIA

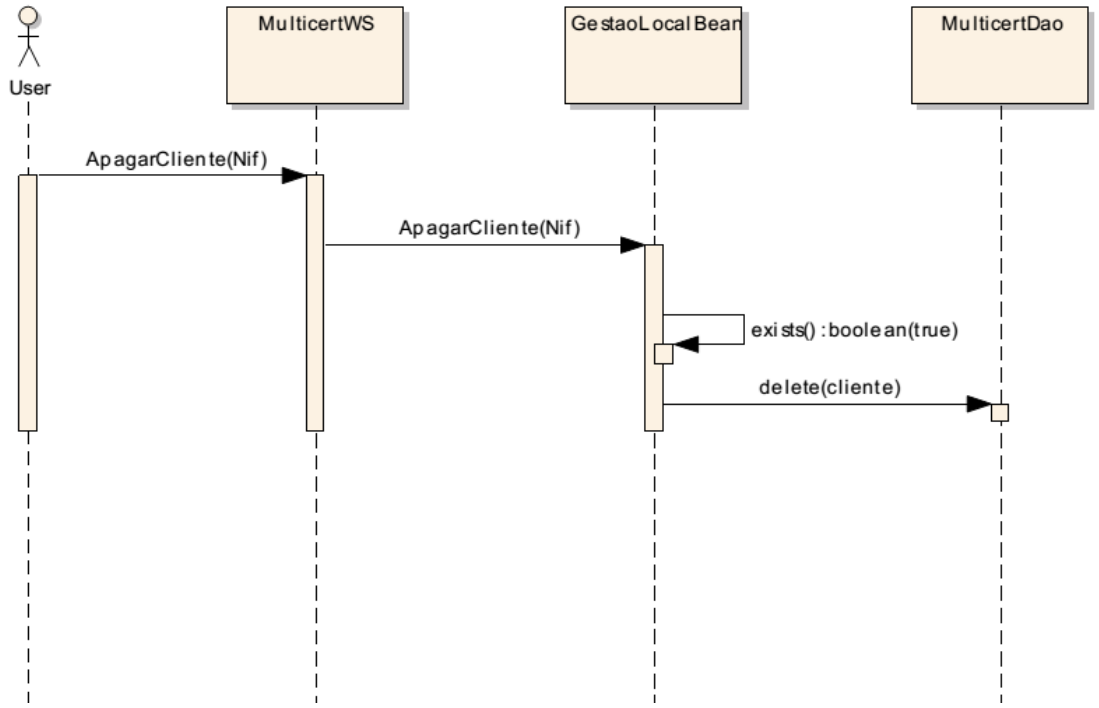
3.3.1. Guardar Cliente



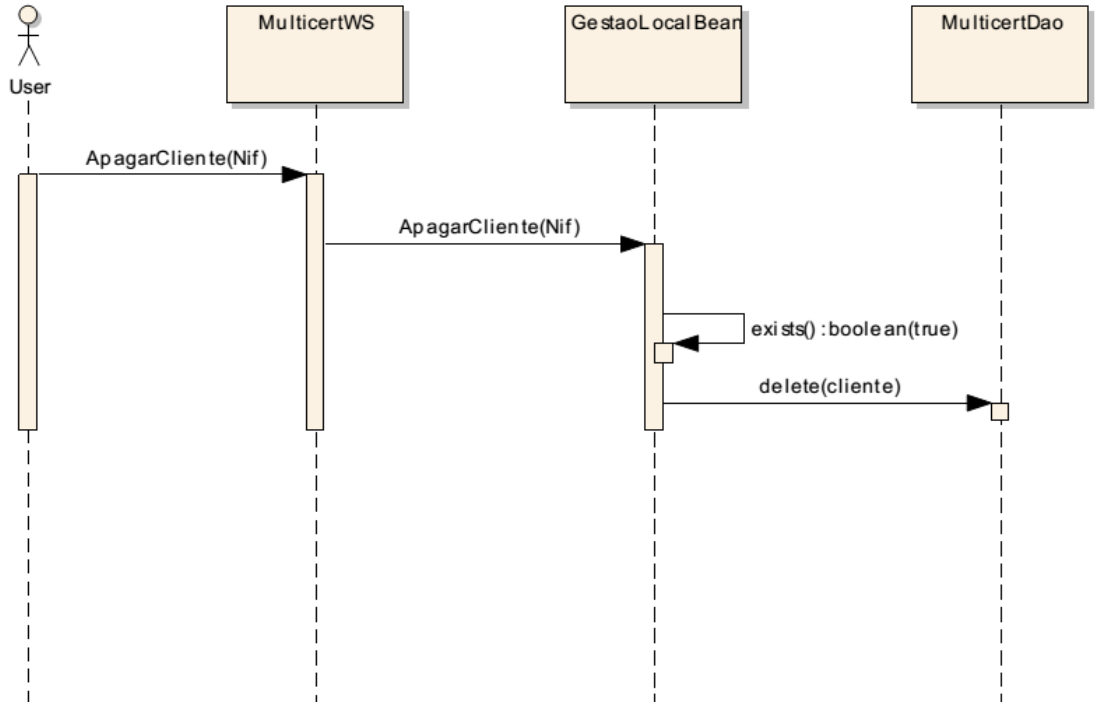
3.3.2. Guardar Cliente Com Nif Já Existente



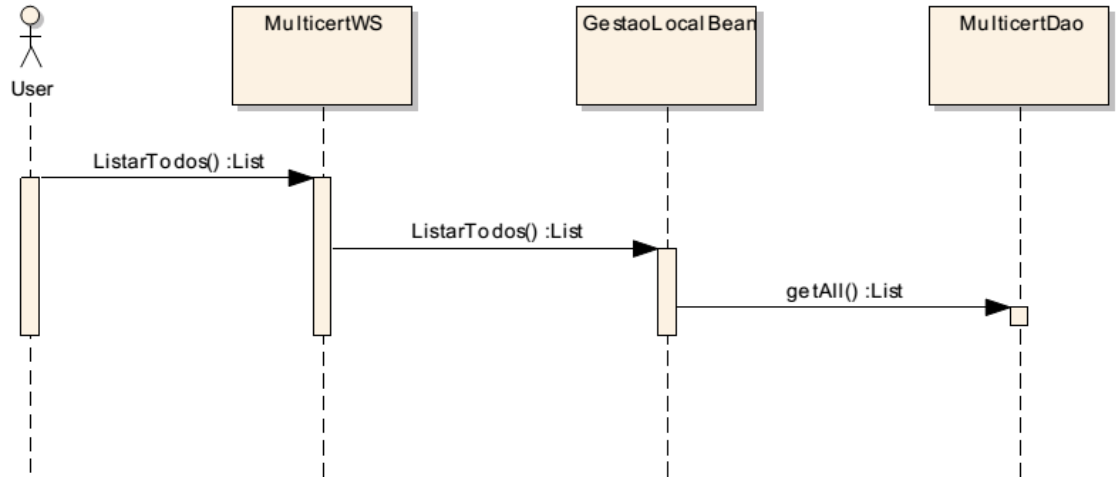
3.3.3. Apagar Cliente



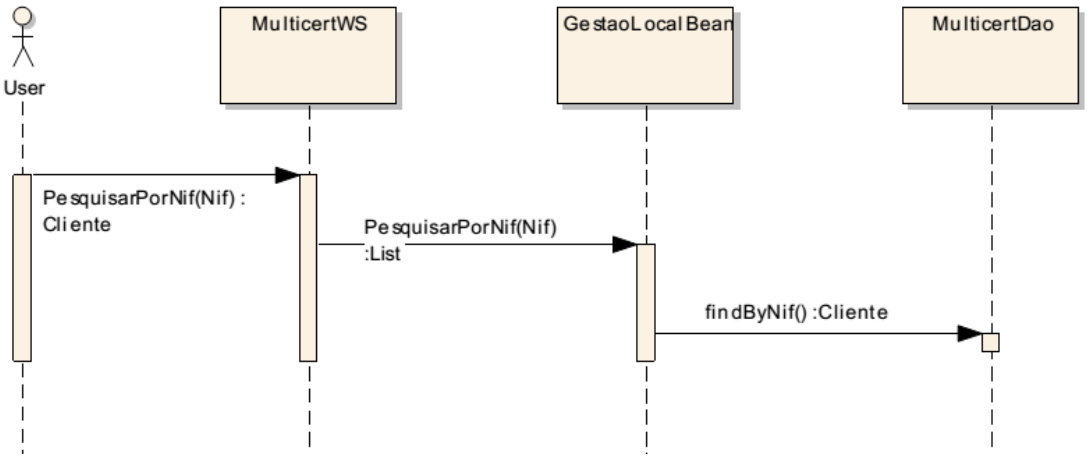
3.3.4. Apagar Cliente Inexistente



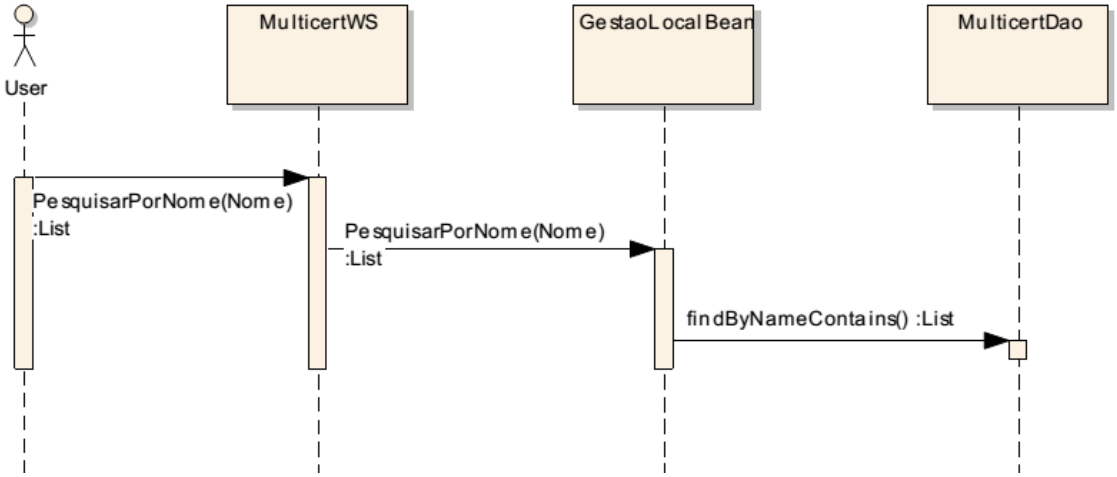
3.3.5. Listar Todos



3.3.6. Listar Por Nif



3.3.7. Listar Por Nome Parcial



4. TESTES UNITÁRIOS

Para assegurar a consistência da camada de negócio (GestãoLocalBean), fizeram-se testes unitários que tivessem uma cobertura de todas as suas operações. Para isolar o foco na entidade a testar, simulou-se o comportamento do bean MulticertDao com recurso à framework Mockito. Garantiu-se que os resultados das operações são consistentes, e que em caso de erro do utilizador, é gerada a respetiva mensagem de erro.

A classe de testes unitários é a `GestaoLocalBeanTest.java` e os seus testes são:

`testCreateAndRead`

`testCreateWithSameNifTwice`

`testCreateDeleteReadSameObject`

`testDeleteTwice`

5. TESTES FUNCIONAIS INTEGRADOS

Para testar a solução numa perspetiva mais macro num fluxo end-to-end, realizaram-se testes integrados. O objectivo é testar todos os use cases do serviço como um todo, utilizando somente um cliente Web da interface SOAP. Todos os aspetos da consistência de resultados e das mensagens de erro geradas foram testados.

Os testes estão definidos pela classe `FunctionalIT.java`, e os testes são:

```
        * Teste 1
        *  - Adicionar Cliente A
        *  - Listar Cliente A, verificar que foi retornado
        *  - Apagar Cliente A
        *  - Listar Por Nif o nif do Cliente A e verificar mensagem de erro
que não existe



        * Teste 2
        *  - Adicionar Cliente A
        *  - Adicionar Cliente B, com o mesmo nif
        *  - Verificar mensagem de erro
        *

        * Teste 3
        *  - Adicionar Cliente A
        *  - Apagar Cliente A
        *  - Apagar Novamente Cliente A
        *  - Verificar mensagem de erro
```

```
        * Teste 4
        *  - Adicionar Cliente A
        *  - Adicionar Cliente B (nome de B é parcialmente comum a A)
        *  - Adicionar Cliente C
        *  - Listar os 3 clientes
        *  - Listar Clientes com nomes parcialmente comuns, listar com nomes
singulares
```

6. TESTES DE CARGA

Para os testes de carga testou-se a escalabilidade do sistema em função do número de pedidos em paralelo. Populou-se a base de dados com 1000 registos, e estudou-se a performance do sistema a responder ao pedido "listarClientes()" em paralelo.

Utilizou-se a ferramenta SOAPUI para a simulação dos pedidos.

Dados da mesa de testes:

Servidor Windows Azure: - Windows Server 2015 - Single Core (Velocidade Desconhecida) -2Gb de RAM	Cliente: -Windows 10 -i7QuadCore - 8Gb Ram -Net: Meo 3g

Cada Thread do SOAPUI executou sequencialmente o pedido "listarClientes" durante um intervalo de tempo de 60 segundos. Fez-se variar o numero de threads de 5 até 720.