



Escola de Ciências e Tecnologias

Curso: Engenharia Informática

Disciplina: Sistemas Operativos II

Professor: José Saias

Viagens Aéreas do Alentejo

Segundo Trabalho Prático

Trabalho Elaborado Por:

Luís Polha, Nº 20464

Marlene Oliveira, Nº 25999

Ano Lectivo 2011/2012

Introdução

No âmbito do segundo trabalho prático da disciplina de Sistemas Operativos II foi-nos solicitada a elaboração de um sistema de venda de viagens aéreas. O sistema a ser elaborado irá armazenar a informação relativa aos voos de cada companhia numa base de dados relacional e deverá permitir o acesso às diversas funcionalidades disponibilizadas (pesquisa de voos por destino, compra de lugares num voo, consultar a lista de passageiros de um voo e consultar os registos das vendas efectuadas) através de uma interface web.

Para que tal seja possível decidimos fazer o nosso trabalho como uma aplicação web usando *servlets*. As *servlets* são programas Java compilados e que são executados do lado do servidor e que adicionam funcionalidades, permitindo a criação dinâmica de conteúdo web, a um servidor que aceita pedidos. As *servlets* funcionam num modelo de pedido/resposta, como muitos outros objectos em utilização na web. O cliente efectua um pedido, normalmente usando o protocolo http, que invoca a servlet no servidor e que depois recebe como retorno o resultado da sua execução.

Para que um serviço seja seguro têm de ter algum tipo de segurança entre as mensagens enviadas entre o cliente e o servidor. Para isso usamos as chaves e algoritmos criptográficos que o java fornece. A criptografia tem 4 objectivos, garantir a confidencialidade da mensagem, isto é, só o destinatário deve ser capaz de extrair o conteúdo da mensagem, deve ter integridade da mensagem, ou seja, o destinatário deve ser capaz de determinar se a mensagem foi alterada durante a transmissão, deve ser possível ao destinatário identificar o remetente e verificar que foi ele mesmo quem enviou a mensagem e finalmente o não-repúdio do emissor, que significa que não é possível ao emissor negar a autoria da mensagem. Foram utilizados elementos de criptografia na elaboração do extra deste trabalho.

As classes elaboradas para que o trabalho pudesse funcionar correctamente foram as seguintes:

- **Crypto;**
- **CryptoKS;**
- **LojaImpl;**
- **Replicacao;**
- **ViagensClient;**
- **ViagensServer.**

Para o *backend* do nosso trabalho foi ainda elaborada a interface Loja.

Para o *frontend* do nosso trabalho foram elaboradas as seguintes classes que implementam as servlets:

- **MyLoja;**
- **MyLojaPesq;**
- **MyLojaCompra;**
- **MyLojaPLP;**
- **MyLojaRegistos.**

Classe Crypto

Esta classe contém a implementação dos métodos que encriptam e desencriptam ficheiros. Estes métodos são utilizados durante a implementação do extra deste trabalho prático, que requer que sejam gerados ficheiros encriptados que contenham a informação da venda e que posteriormente terão de ser desencriptados para que a sua informação seja consultada. Esta classe possui os seguintes métodos:

- **cifrar:** Este método recebe como argumentos uma *SecretKey* e o nome do ficheiro a encriptar. O método cifrar começa por ler os dados do ficheiro desencriptado e por criar uma instância da cifra a ser utilizada para encriptar o referido ficheiro utilizando a chave secreta fornecida como argumento. Neste caso, utilizámos o algoritmo de encriptação AES (Advanced Encryption Standard) para efectuar a encriptação de ficheiros. De seguida, o método irá encriptar os dados do ficheiro cujo nome foi fornecido como argumento e irá escrever os dados encriptados para um novo ficheiro. Finalmente, elimina-se o ficheiro desencriptado, permanecendo apenas no repositório o ficheiro encriptado.
- **decifrar:** Este método recebe como argumentos um nome de um ficheiro encriptado e uma chave secreta. Tal como o nome indica, o método decifrar efectua a operação contrária à efectuada pelo método cifrar. A única diferença entre estes métodos, para além da função que desempenham, está relacionada com o retorno dos mesmos. Ao invés de não retornar nada como acontece com o método cifrar, o método decifrar retorna a string com os dados desencriptados.

Classe CryptoKS

Esta classe permite a criação do KeyStore onde irá ser guardada a chave secreta utilizada para encriptar e descriptar ficheiros. Inicialmente, começamos por ler de um ficheiro a password que pretendemos atribuir ao KeyStore. De seguida é criada uma instância de um KeyStore do tipo JCEKS (Java Cryptography Extension KeyStore). Seguidamente é criado um KeyGenerator que irá gerar a chave secreta que pretendemos guardar no KeyStore. Finalmente, é criada a entrada da chave secreta no KeyStore e é efectuado o load do KeyStore, que irá ser gravado no ficheiro KStore.

Classe ViagensServer

Os servidores do backend nosso trabalho utilizam RMI. Assim, reutilizou-se a implementação do servidor RMI do primeiro trabalho prático desta disciplina. Recapitulando o funcionamento desta classe: inicialmente é criado um objecto remoto, que será depois exportado (permitindo que o mesmo receba pedidos) e registado no serviço de nomes do sistema e, finalmente, é efectuado o bind do referido objecto.

Classe ViagensClient

Inicialmente esta classe era usada apenas para que fosse possível testar o funcionamento das servlets sem que o código que garante a coerência dos dados tivesse sido ainda implementado. Porém, após a implementação do referido código, verificou-se que as servlets ignoravam completamente as chamadas do mesmo. Assim, arranjámos uma solução alternativa para que pudéssemos demonstrar o funcionamento das mesmas. Esta classe faz parte da implementação da solução utilizada. Assim, aqui fica uma descrição dos métodos da mesma:

- **assocObjRem:** efectua a invocação do objecto remoto.
- **pesquisar:** recebe como argumento uma string correspondente ao destino para o qual se pretendem encontrar voos. Este método invoca o método remoto e efectua a pesquisa, armazenando os resultados num vector, que posteriormente será retornado.
- **compra:** recebe como argumentos o ID do voo para o qual se pretendem adquirir lugares, uma string com os nomes dos passageiros e um inteiro com o número de lugares a adquirir. Este método invoca o método remoto que efectua a compra e armazena a confirmação da mesma numa string. A informação recebida posteriormente será retornada.
- **consultaListaP:** recebe como argumento o ID do voo cuja lista de passageiros se pretende consultar. Este método invoca o método remoto que efectua a consulta

da lista de passageiros do voo indicado. A informação recebida será armazenada num vector temporário, que posteriormente será retornado.

- **consultaRegistos:** recebe como argumento o ID da venda cujos dados se pretende consultar. Este método invoca o método remoto que efectua a descriptação do ficheiro correspondente ao ID da venda e devolve uma string com a informação pretendida. A informação recebida será armazenada numa string que posteriormente será retornada.

Classe LojaImpl

A classe LojaImpl implementa a interface remota Loja. Esta classe possui a implementação de todos os métodos remotos, bem como alguns métodos extra que servem apenas de complemento aos métodos remotos. Fazem parte desta classe os métodos seguintes:

- **connectBD:** permite efectuar a ligação à base de dados. Este método lê a informação necessária (host, nome da base de dados, username e password) de um ficheiro e utiliza essa informação para se ligar ao SGBDR. Se a ligação for bem-sucedida, é retornada uma string que assim o indica. Caso contrário, é retornada uma mensagem de erro.
- **logoutBD:** este método fecha a ligação à BD, permitindo assim fazer o logout da mesma.
- **pesquisa:** recebe como argumento uma string correspondente ao destino para o qual se pretende pesquisar voos. Este método liga-se à base de dados utilizando o método connectBD e efectua a pesquisa utilizando uma query SQL. No final, é efectuado o logout da base de dados e é retornado o resultado da pesquisa. Se algo correr mal, é retornado o vector que contém as mensagens de erro fornecidas pelo programa.
- **consultaLP:** recebe como argumentos um inteiro correspondente ao ID do voo cuja lista de passageiros se pretende consultar. Este método começa por se ligar à base de dados, utilizando o método connectBD. Seguidamente, é utilizada uma query SQL para pesquisar na base de dados e obter a lista de passageiros pretendida. Finalmente, é efectuado o logout da BD retornada a referida lista de passageiros. Caso ocorra algum erro, é retornado o vector que contém as mensagens de erro fornecidas pelo programa.
- **comprar:** recebe como argumentos o ID do voo no qual se pretende adquirir lugares, um array de Strings com os nomes dos passageiros e um inteiro correspondente ao número de lugares a adquirir. Tal como acontece nos métodos anteriores, o programa liga-se à base de dados com o método connectBD. De seguida obtém-se a informação do voo indicado (destino, número de lugares vagos e data e hora). Verifica-se se ainda pode ser adquirido neste voo um número de lugares igual ao indicado. Em caso afirmativo, são criados os inserts que irão armazenar a informação do cliente na base de dados e, após a informação ter sido inserida na base de dados, é actualizado o número de lugares vazios no voo.

Finalmente, é armazenada a informação da venda num ficheiro encriptado. Caso o número de lugares disponíveis não seja suficiente para satisfazer o pedido, a compra não é efectuada e é retornada uma mensagem de erro.

- **obterSecretKey**: este método permite obter a chave secreta que se encontra no KeyStore e que foi usada para encriptar os ficheiros que contêm a informação das vendas. Para que tal seja possível, começamos por criar um KeyStore vazio, que mais tarde irá armazenar a informação do KeyStore guardado no ficheiro. De seguida, obtemos a password do KeyStore do ficheiro e fazemos o load do mesmo para o KeyStore vazio. Finalmente, obtemos a chave secreta pretendida e retornamos a mesma.
- **saveVenda**: recebe como argumento o nome para o ficheiro com que pretendemos armazenar a informação e uma string correspondente à informação da venda. Este método começa por escrever a informação para um ficheiro descriptado, que será posteriormente encriptado com recurso ao método cifrar da classe Crypto e eliminado. O repositório armazenará apenas os ficheiros encriptados.
- **consultaRegistos**: recebe como argumento o ID da venda cujos dados se pretendem consultar. Este método descripta a informação constante no ficheiro pretendido e retorna uma string com a informação descriptada.

Classe Replicacao

Esta classe foi elaborada com o objectivo de garantir que a informação retornada quando são efectuadas pesquisas é coerente. Para tal, foram elaborados os seguintes métodos:

- **checkOperacionais**: este método começa por ler a informação relativa aos servidores dos backend e armazena a mesma. De seguida são criados os clientes necessários para a comunicação e são armazenados os mesmos num vector criado para o efeito.
- **fazPedidoPesquisa**: recebe como argumento a string correspondente ao destino para o qual se pretendem pesquisar voos. Este método começa por efectuar a pesquisa pretendida e armazenando as respostas. De seguidas as respostas são comparadas entre si, sendo o número de vezes que cada resposta aparece repetida armazenado num array de inteiros. O programa irá então verificar qual foi a resposta mais comum. Seguidamente, é efectuada a pesquisa e retornado o vector que corresponde à resposta mais comum (que é tida como correcta).
- **fazPedidoConsultaLP**: recebe como argumento um inteiro correspondente ao ID do voo cuja lista de passageiros pretendemos consultar. Este método funciona de modo semelhante ao modo que efectua a pesquisa e utiliza o mesmo método para retornar a resposta certa (mais comum).

NOTA: Como as servlets estavam a ignorar a instância desta classe lá efectuada, mantivemos o main desta classe na mesma de modo a que se possa testar o funcionamento desta. O comando do makefile que permite testar esta classe é

Servlets do Nosso Trabalho

O frontend do nosso trabalho é constituído por uma webapp que utiliza as servlets elaboradas para fornecer uma interface mais amigável e transparente ao utilizador. De seguida procedemos à descrição do funcionamento das servlets elaboradas.



Figura 1 – Homepage do nosso sistema.

Servlet MyLoja

Esta servlet serve como menu do nosso sistema de vendas. É uma servlet muito simples que contém apenas os botões que encaminham o utilizador para as diversas funções que o sistema oferece.

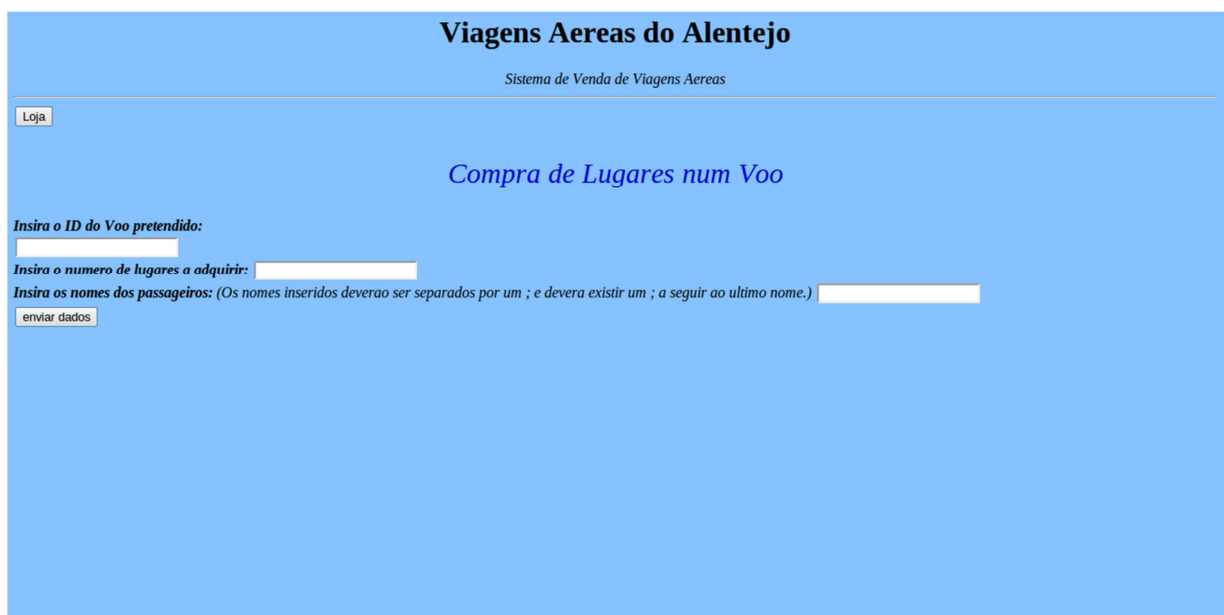


The screenshot shows a web page titled "Viagens Aereas do Alentejo" with a subtitle "Sistema de Venda de Viagens Aereas". A "Home" button is in the top left. The main content area is titled "Escolha a operacao:" and lists four options, each with a "Go" button: "Pesquisar Voos", "Comprar Lugares", "Consultar Lista Passageiros", and "Consultar Registos de Vendas".

Figura 2 – Página gerada pela servlet MyLoja.

Servlet MyLojaCompra

Esta servlet permite ao utilizador adquirir lugares num voo. Para tal apenas terá de inserir os dados solicitados nos campos indicados.



The screenshot shows a web page titled "Viagens Aereas do Alentejo" with a subtitle "Sistema de Venda de Viagens Aereas". A "Loja" button is in the top left. The main content area is titled "Compra de Lugares num Voo". It contains three input fields with labels: "Insira o ID do Voo pretendido:", "Insira o numero de lugares a adquirir:", and "Insira os nomes dos passageiros: (Os nomes inseridos deverao ser separados por um ; e devera existir um ; a seguir ao ultimo nome.)". A "enviar dados" button is at the bottom left.

Figura 3 – Página gerada pela servlet MyLojaCompra.

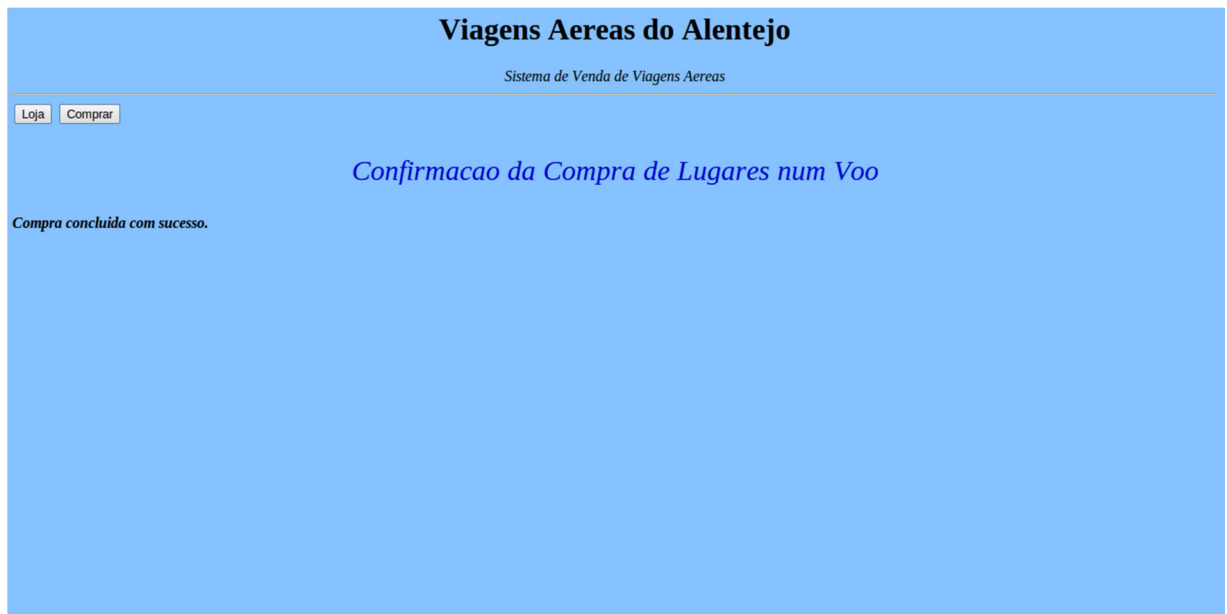


Figura 4 – Página gerada pela servlet MyLojaCompra quando é retornada a confirmação da compra.

Servlet MyLojaPesq

Esta servlet permite ao utilizador pesquisar voos por destino. Caso existam voos para o destino indicado, o utilizador poderá comprar lugares nos voos retornados pela pesquisa ou consultar a lista de passageiros dos mesmos.

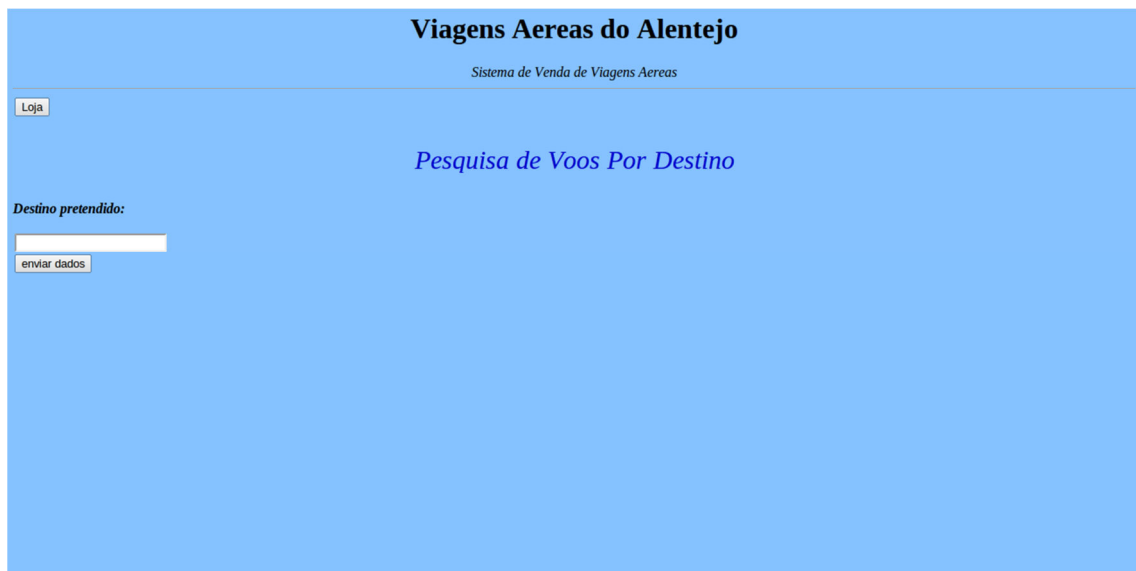


Figura 5 – Página gerada pela servlet MyLojaPesq.

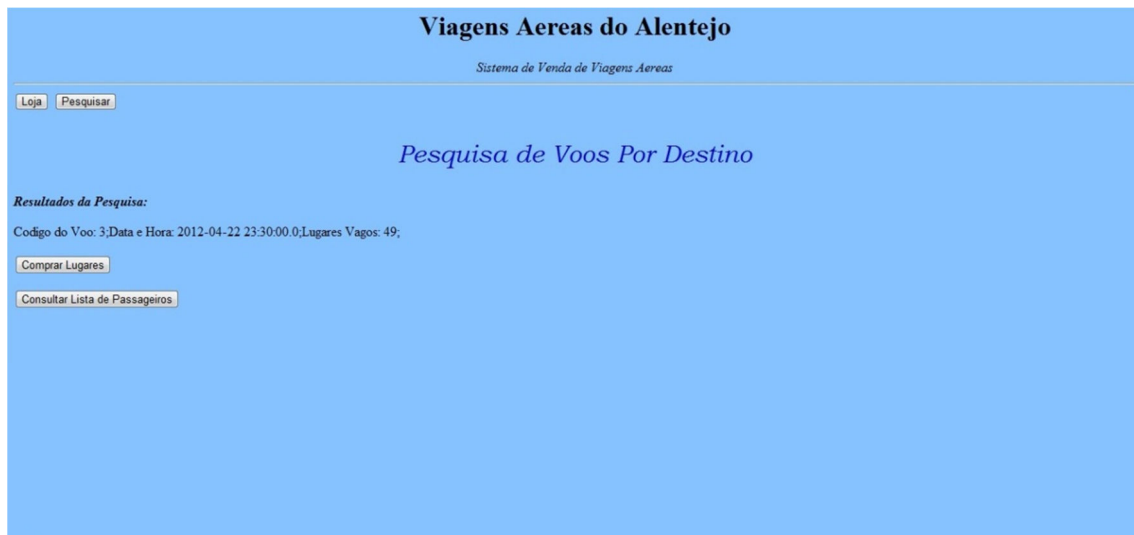


Figura 6 – Página gerada quando são obtidos resultados na pesquisa.

Servlet MyLojaPLP

Servlet que permite ao utilizador consultar a lista de passageiros de um voo. Para tal, o utilizador apenas terá de fornecer o ID do voo cuja lista de passageiros pretende consultar. Caso o ID do voo inserido seja inválido, na página que indicará que não existem voos com aquele ID será disponibilizada a opção que permite a pesquisa de voos.

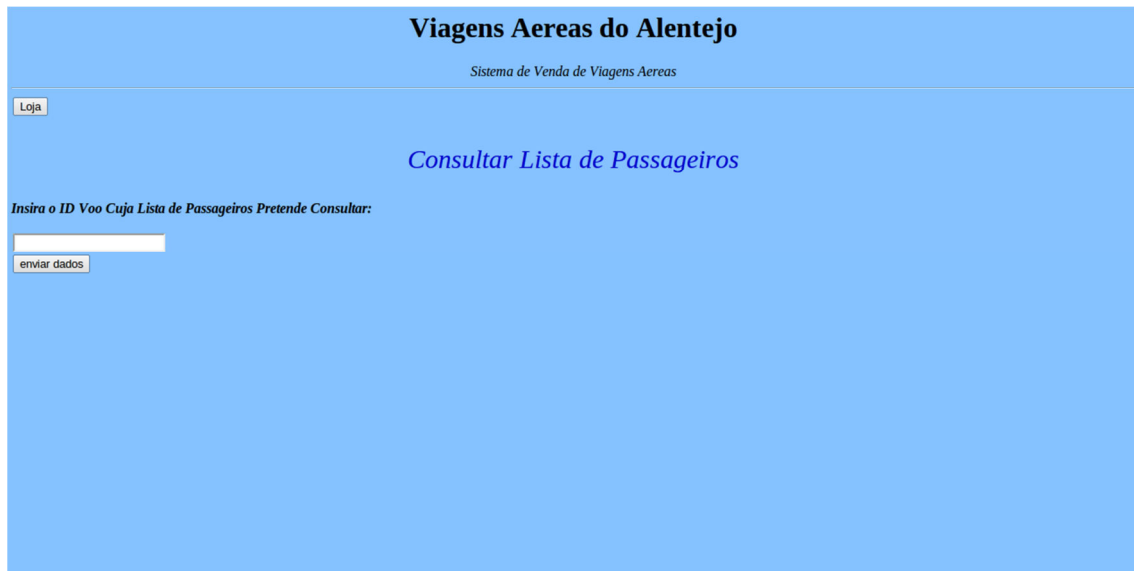


Figura 7 – Página gerada pela servlet MyLojaPLP.

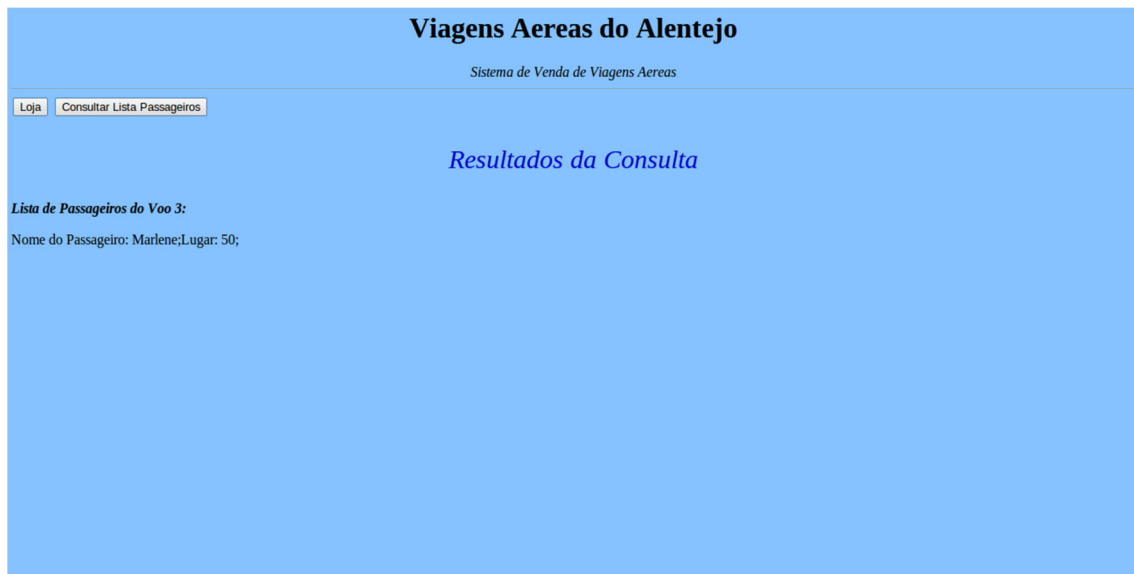


Figura 8 – Página gerada quando existem resultados na consulta.

Servlet MyLojaRegistos

Esta servlet faz parte do extra deste trabalho prático e permite ao utilizador consultar registos de vendas. Para efectuar a consulta, o utilizador apenas tem de fornecer o ID da venda cuja informação pretende consultar.

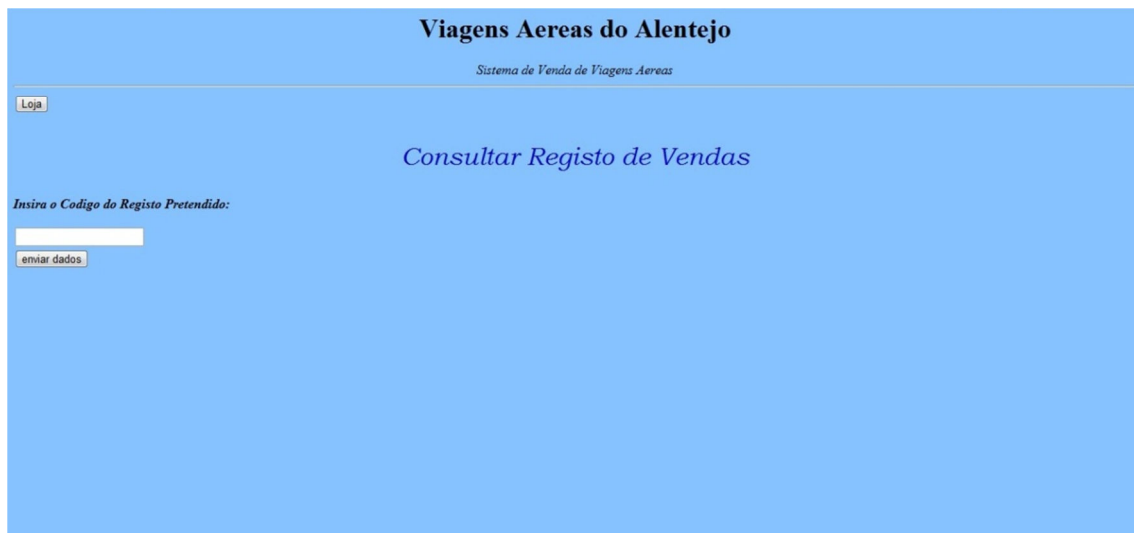


Figura 9 – Página gerada pela servlet MyLojaRegistos.

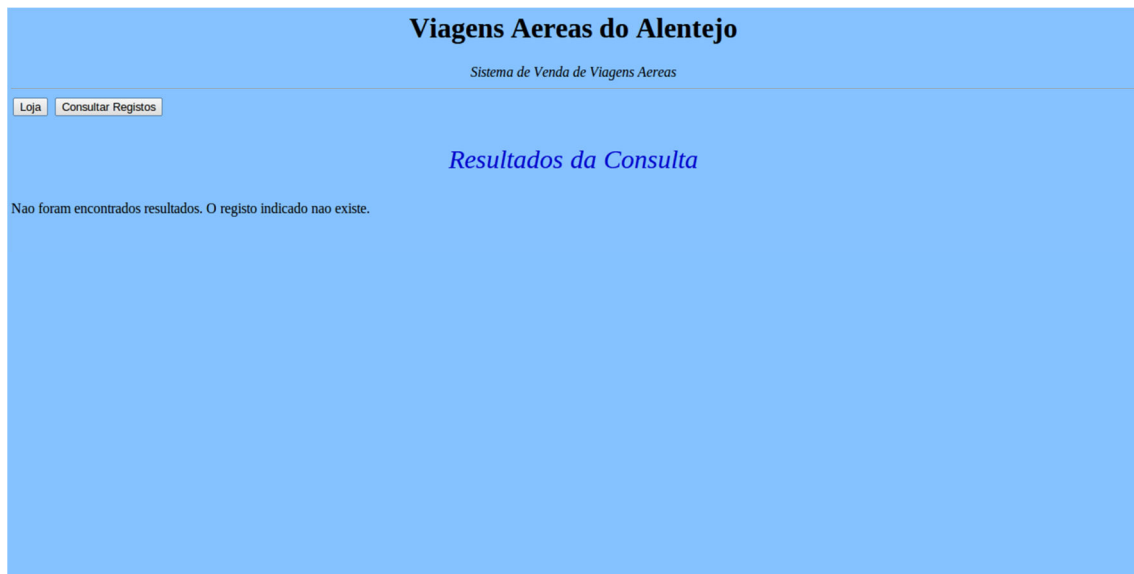


Figura 10 – Página gerada quando não existem resultados na consulta.

NOTA: Como as servlets MyLojaPesq e MyLojaPLP estavam a ignorar a instanciação da classe Replicacao que iria efectuar a pesquisa e garantir que os resultados eram coerentes, arranjámos a seguinte alternativa: instanciar um objecto que funciona como um cliente RMI que, sempre que a servlet ignora os métodos da instância da classe Replicacao, se liga a um servidor RMI de backup (que contém, em teoria, a informação actualizada) e efectua a operação pretendida. Note-se que os métodos da classe Replicacao funcionam correctamente (tal foi verificado quando estes foram testados fora das servlets).

Anexos

SQL que Permite Criar a Base de Dados do Nosso Trabalho

```
CREATE TABLE Voo( id_voo INTEGER NOT NULL, destino VARCHAR(40) NOT NULL, data_hora  
TIMESTAMP NOT NULL, lugares_vagos INTEGER NOT NULL, PRIMARY KEY(id_voo));
```

```
CREATE TABLE Cliente(id_passageiro INTEGER NOT NULL, nome VARCHAR(150) NOT NULL,  
PRIMARY KEY(id_passageiro));
```

```
CREATE TABLE ListaPass(id_voo INTEGER NOT NULL, id_passageiro INTEGER NOT NULL, lugar  
INTEGER NOT NULL, PRIMARY KEY(id_voo,id_passageiro,lugar), foreign key(id_voo)  
REFERENCES Voo(id_voo), foreign key(id_passageiro) REFERENCES Cliente(id_passageiro));
```

```
CREATE TABLE Vendas(cod_venda INTEGER NOT NULL, id_passageiro INTEGER NOT NULL,  
id_voo INTEGER NOT NULL, PRIMARY KEY(cod_venda,id_passageiro,id_voo), foreign  
key(id_passageiro) REFERENCES Cliente(id_passageiro), foreign key(id_voo) REFERENCES  
Voo(id_voo));
```

SQL que Para a Inserção dos Voos disponíveis na Base de Dados do Trabalho

```
INSERT INTO Voo VALUES('1','Seychelles',to_timestamp('12-04-2012 21:30','DD-MM-YYYY  
HH24:MI'),'50');
```

```
INSERT INTO Voo VALUES('2','Nova Iorque',to_timestamp('12-04-2012 11:30','DD-MM-YYYY  
HH24:MI'),'50');
```

```
INSERT INTO Voo VALUES('3','Roma',to_timestamp('22-04-2012 23:30','DD-MM-YYYY  
HH24:MI'),'50');
```

```
INSERT INTO Voo VALUES('4','Moscovo',to_timestamp('22-04-2012 23:30','DD-MM-YYYY  
HH24:MI'),'5');
```

```
INSERT INTO Voo VALUES('5','Sydney',to_timestamp('25-04-2012 13:30','DD-MM-YYYY  
HH24:MI'),'50');
```

```
INSERT INTO Voo VALUES('6','Rio de Janeiro',to_timestamp('27-04-2012 16:30','DD-MM-YYYY  
HH24:MI'),'50');
```

```
INSERT INTO Voo VALUES('7','Lisboa',to_timestamp('30-04-2012 22:30','DD-MM-YYYY  
HH24:MI'),'50');
```

```
INSERT INTO Voo VALUES('8','Lisboa',to_timestamp('02-04-2012 10:00','DD-MM-YYYY  
HH24:MI'),'50');
```

```
INSERT INTO Voo VALUES('9','Pequim',to_timestamp('02-07-2012 15:00','DD-MM-YYYY  
HH24:MI'),'50');
```

```
INSERT INTO Voo VALUES('10','Tunisia',to_timestamp('12-07-2012 18:00','DD-MM-YYYY  
HH24:MI'),'50');
```