

Laboratório – Trabalhando Com Tabelas e Consultas Segundo o Padrão MVC, com JSP, Servlets e EJBs.

O modelo pelo qual está sendo desenvolvido este projeto é apresentado na Figura 1. Até o momento, foram usados arquivos JSP e Servlets para tratar as requisições dos usuários, e como regra de controle para cadastrar dados na base de dados.

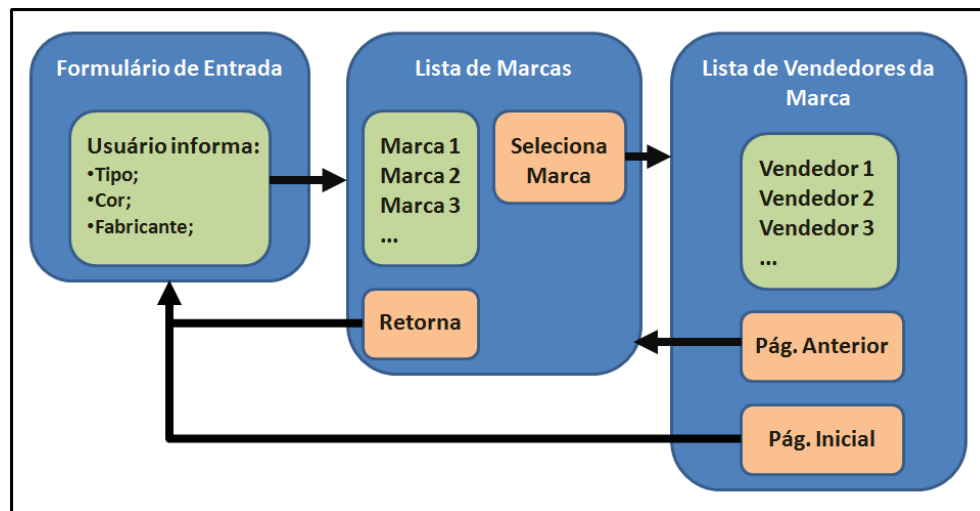


Figura 1 – Fluxo de Páginas da Funcionalidade

De acordo com a Figura 1, a última etapa do aplicativo é a apresentação de uma página com a lista de marcas que atendem às características que o usuário informou. Nessa tela, o usuário pode selecionar um item da lista para identificar os vendedores que trabalham com a marca, ou retornar para a página anterior. Por último, o sistema apresenta uma lista dos vendedores que possuem a marca para vender e as apresenta ao usuário. Nessa tela, o usuário pode retornar para a página inicial, ou para a página anterior.

Apesar dos Servlets serem boas ferramentas para consistir os dados que são informados pelo usuário, eles não são muito úteis para montar páginas HTML que precisem mostrar resultados na forma de listas, pois esse gerenciamento é mais trabalhoso, se feito dentro de um Servlet. A forma mais simples e eficiente de fazer esse trabalho é usando os JavaBeans, que são componentes de software escritos na linguagem de programação Java. Na Prática são simples classes Java escritas de acordo com uma convenção em particular, que encapsulam muitos objetos em um único objeto (o bean), assim eles podem ser transmitidos como se fossem um único objeto, em vez de vários objetos individuais. Na forma de classe, o JavaBean é um construtor nulo e permite acesso às suas propriedades através de métodos getter e setter. Um código de JavaBeans simple é apresentado na Figura 1.

```

public class JavaBeanPessoa implements java.io.Serializable {

    private String nome;
    private int idade;

    public JavaBeanPessoa() {
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public String getNome() {
        return this.nome;
    }

    public int getIdade() {
        return this.idade;
    }
}

```

Figura 1 – Exemplo de código-fonte de um JavaBean

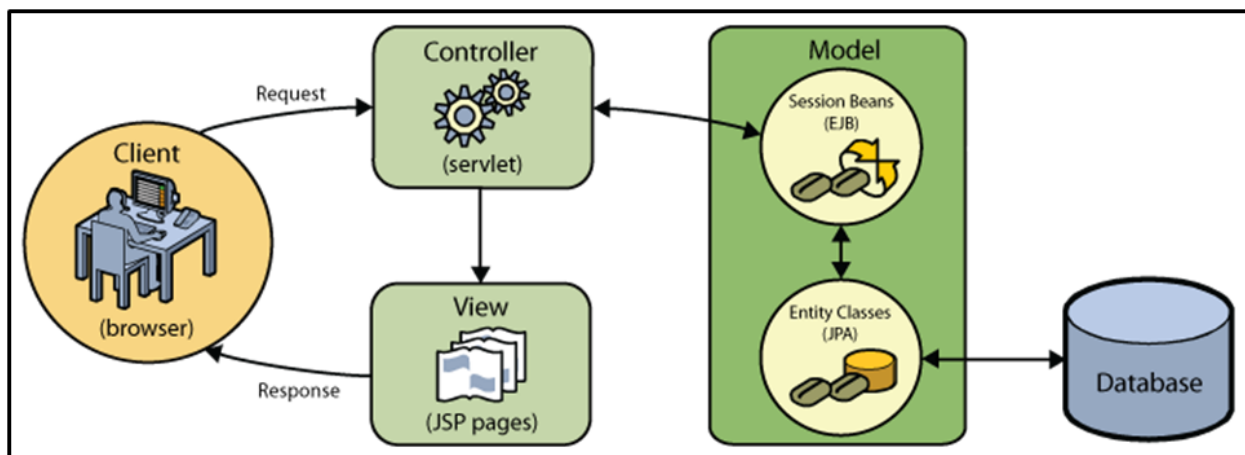


Figura 2 – Implementação do MVC com arquivos JSP, Servlets e EJBs

O Que Já Existe

A partir do exemplo feito no último roteiro, o aplicativo já contém as seguintes classes de entidade Vendedor e Marca, e os EJBs AbstractFacade, MarcaFacade e VendedorFacade, que já realizam as tarefas de persistência de dados na base de dados. Ou seja, toda essa parte já resolve as regras de manipulação da base de dados.


A partir do modelo MVC apresentado no último roteiro, o software terá os JavaBeans inseridos no mesmo nível que os Servlets, segundo a arquitetura apresentada na Figura 2. Os JavaBeans serão responsáveis por conectar as páginas escritas em JSP com os EJBs que controlam o acesso à base de dados, e também farão parte da camada web, de modo análogo aos Servlets. Ou seja, somente adicionaremos novas partes de código-fonte ao projeto, sem interferir no que já existe. Isso demonstra a característica de reusabilidade do software, uma das vantagens do modelo MVC.


O Que Deve Ser Feito

Na prática, o que se pretende, é apresentar ao usuário uma página similar à da Figura 3, que apresenta uma lista com todos os dados gravados na base.

Página de Exemplo - Listagem de Vendedores	
Id	Nome cadastrado
0	CERPA
1	Murphys
2	1795
Voltar	

Figura 3 – O resultado a ser exibido ao usuário final

Ou seja, o conteúdo da lista tem que ser dinamicamente gerado, uma vez que mais dados podem ser cadastrados enquanto o visitante está manipulando os dados, para fazer isso, é necessário adicionar uma página HTML ao projeto web, identificado pelo ícone  BeerChooser.

Para adicionar uma nova página, clique com o botão direito do mouse em  BeerChooser, e no menu que abrir, seleciona a opção HTML. Isso fará com que apareça a tela similar à tela apresentada na Figura 4. Nessa tela, de um nome para a página que lista todos os vendedores, e clique no botão “Finalizar”.

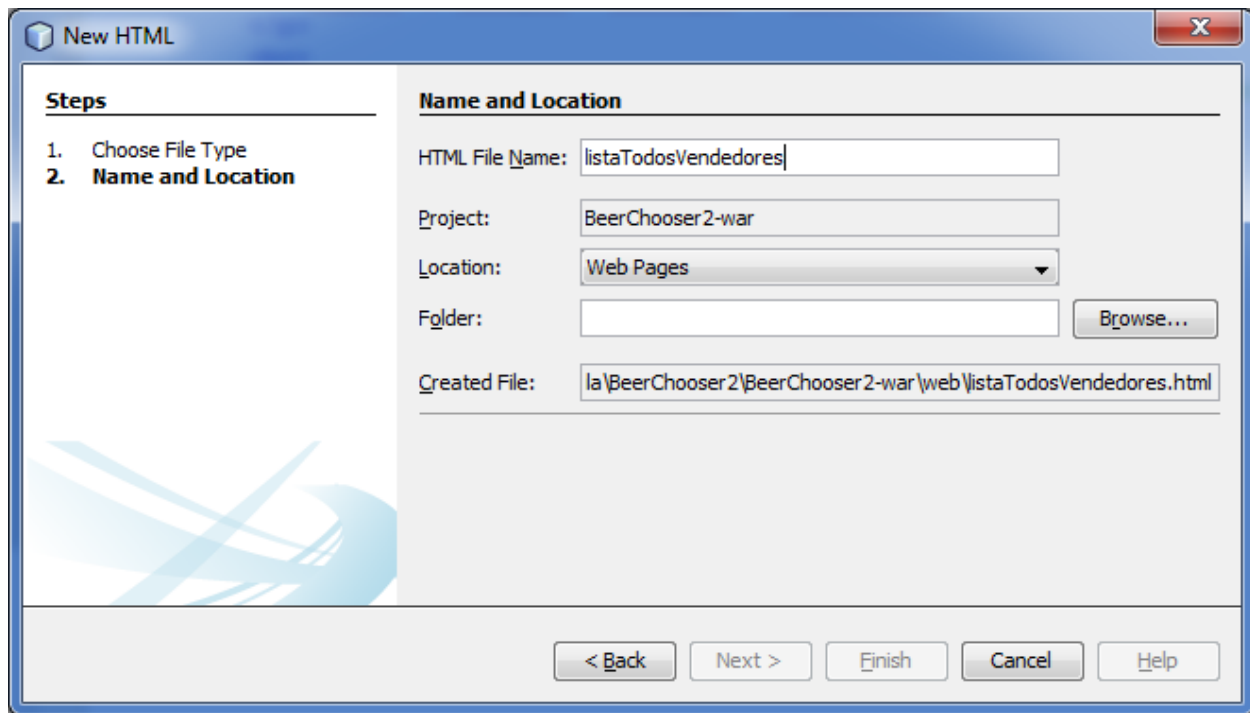


Figura 4 – Tela de configuração de nova página HTML

Isso fará com que o projeto web no NetBeans tenha a aparência da Figura 5, com a nova página HTML adicionada a ele.

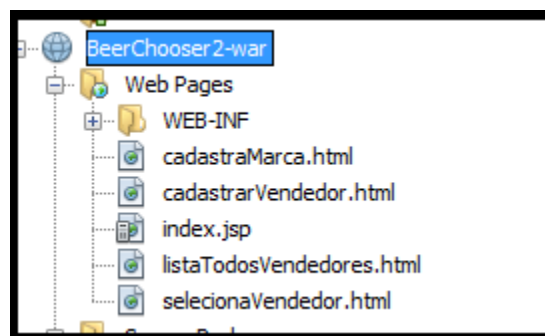


Figura 5 – Projeto web após inclusão de nova página HTML

Agora, basta clicar duas vezes no arquivo HTML recém-criado, para que seja possível editá-lo no NetBeans, e alterado o código HTML para apresentar a listagem, que no caso da página web, é uma tabela com várias linhas (uma para o título e as demais para os nomes cadastrados na base de dados) e duas colunas (ID e Nome do vendedor). O código-fonte da página HTML que faz isso é apresentado na Figura 6, que deverá ter o efeito apresentado na Figura 7.

```

<html>
  <head>
    <title>ListaTodosVendedores</title>
  </head>

  <body>
    <p>
      Página de Exemplo - Listagem de Vendedores
    </p>
    <br>
    <table style="text-align: left; width: 378px; height: 80px;" border="0"
      cellpadding="2" cellspacing="2">
      <tbody>
        <tr>
          <td>Id</td>
          <td>Nome cadastrado</td>
        </tr>
        <tr>
          <td></td>
          <td></td>
        </tr>
      </tbody>
    </table>
    <br>
    <a href="index.jsp">Voltar</a>

  </body>
</html>

```

Figura 6 – Código-fonte da página HTML que lista os resultados do bando

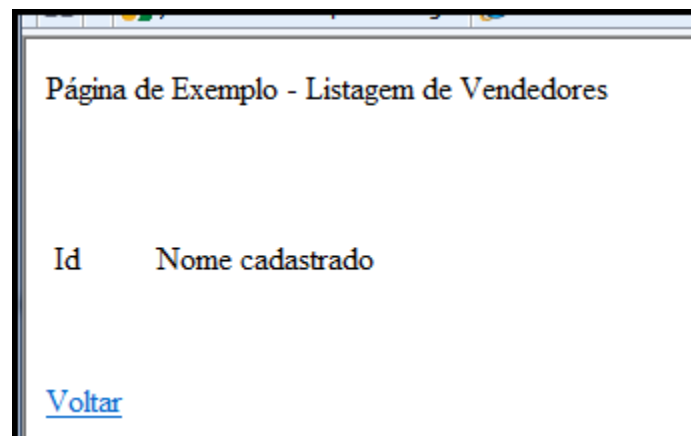



Figura 7 – Resultado visualizado no browser

Neste momento, o primeiro passo é alterar o tipo de arquivo de HTML para JSP, para que seja possível fazer uma página programada usando Java. Isso é necessário, pois a lista de vendedores cadastrados é gerada dinamicamente a partir dos dados armazenados na base de dados, e ao fazer com que a página programada seja gerada no arquivo de extensão JSP, isso fará com que seja possível integrar o desenho do site, escrito em HTML, com a geração dinâmica de conteúdo, que é controlado por scripts Java, inseridos no meio da codificação HTML. No caso deste exemplo, será gerada uma linha de tabela para cada item armazenado no banco de dados, e será apresentado ao usuário o Id e o Nome do vendedor cadastrado.

Para realizar essa alteração, clique com o botão direito do mouse sobre o nome do arquivo “listaTodosVendedores.html” (Figura 5), e no menu que abrir, clique na opção “Propriedade” para abrir a tela apresentada na Figura 8. Depois, no campo “Extensão”, clique no botão  para alterar o tipo de arquivo, conforme Figuras de 9 a 12.

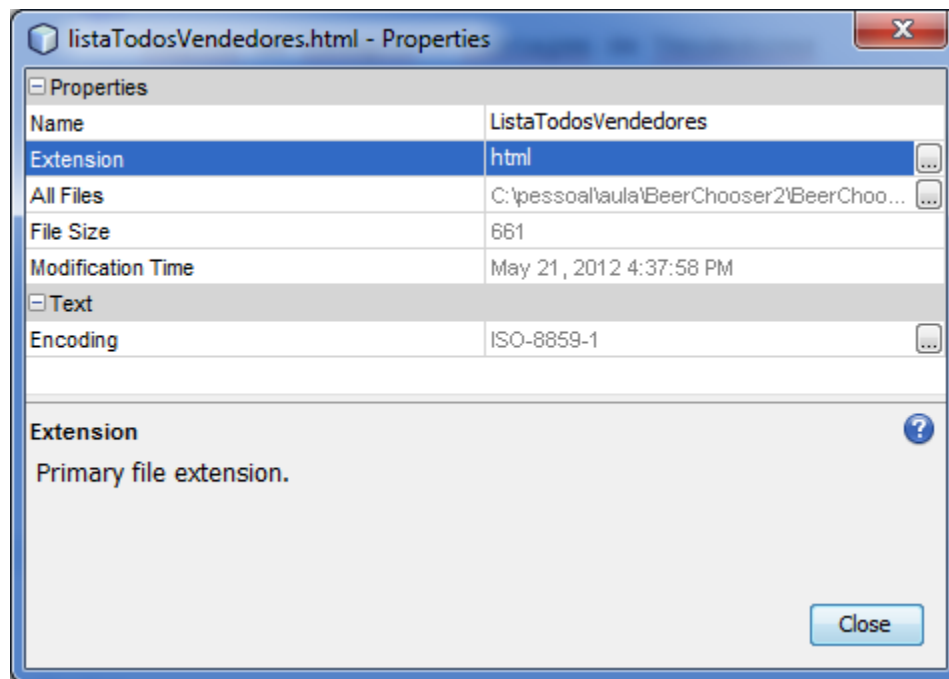


Figura 8 – Tela de Propriedades de Arquivo

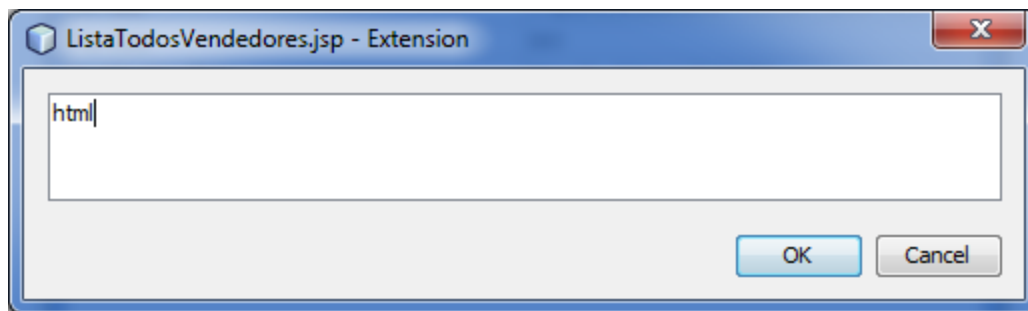


Figura 9 – Tela de Edição de Extensão de Arquivo

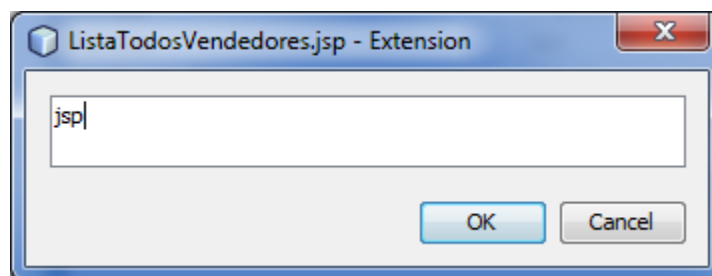


Figura 10 – Tela de Edição de Extensão de Arquivo Após Modificação

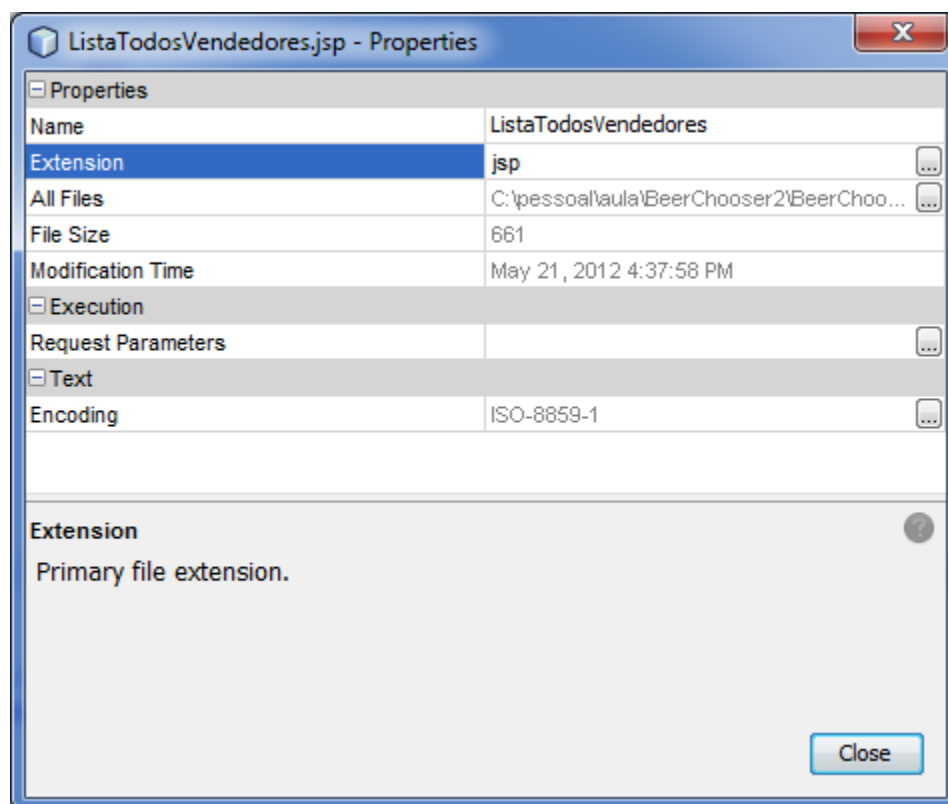


Figura 11 – Resultado da Modificação nas Propriedades do Arquivo

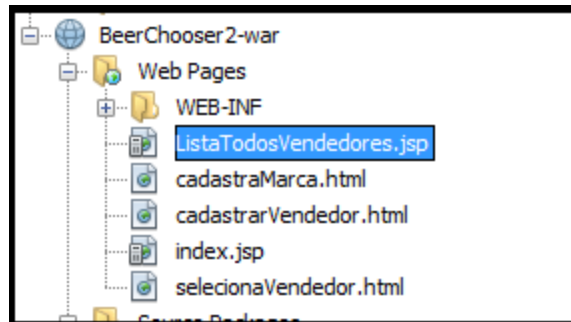


Figura 12 – Resultado da Modificação na Exibição do Projeto

Codificação Para Acessar os EJBs a Partir das Páginas JSP

Por uma questão de implementação, não é possível adicionar o endereçamento de um EJB a uma página JSP, de modo análogo ao que ocorre com os Servlets quando se usa a anotação `@EJB`, que usa o próprio endereçamento do servidor de aplicações para essa tarefa. Uma razão para que isso aconteça é que ao usar o modelo MVC, as páginas JSP são usadas principalmente para tratar de aspectos relacionados ao desenho da interface de interação com o usuário, e toda processamento de controle deve ser realizado através de uma classe escrita especialmente para esse fim, no caso deste exemplo, através de um Servlet, como pode ser observado na Figura 2.

Uma vez que a página JSP não pode interagir com o Servlet para obter os valores que ela precisa, é necessário usar um “bean” que conecte a página JSP ao EJB. Esse “bean” é um “bean de sessão” de tipo “Singleton” que foi usado em outros exemplos de laboratório. Isso fará com que o modelo do aplicativo seja de acordo com a Figura 13, onde o Bean Singleton foi adicionado ao modelo.

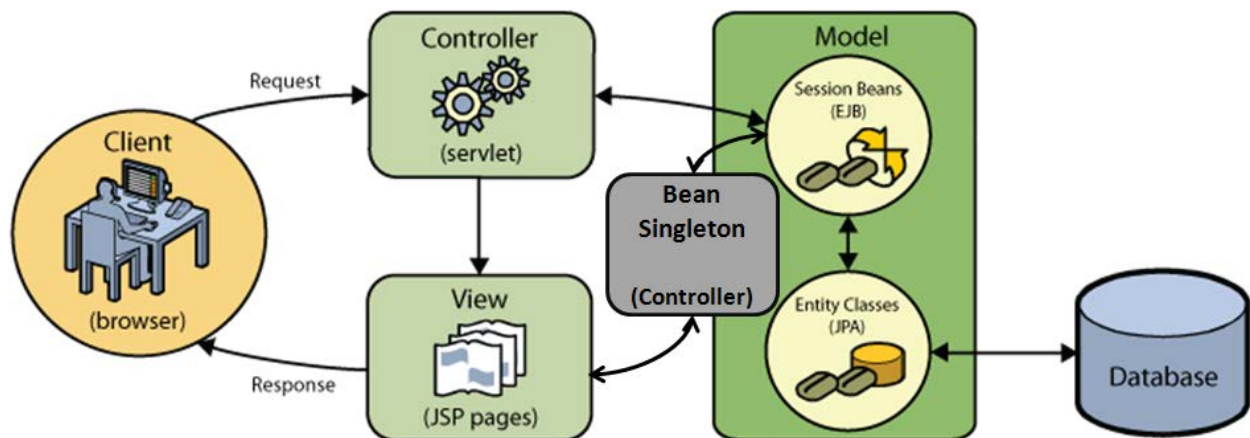



Figura 13 – Modelo MVC após adição do Bean Singleton

Como pode ser observado na Figura 13, o Bean Singleton será capaz de manipular os EJBs que estão na camada Modelo, mas ainda assim, o Bean Singleton fará parte da camada Controle.

Para adicionar o Bean Singleton, clique com o botão direito do mouse em  BeerChooser, e no menu que abrir, selecione as opções Novo -> Bean de Sessão. Isso fará com que apareça a tela da Figura 14.

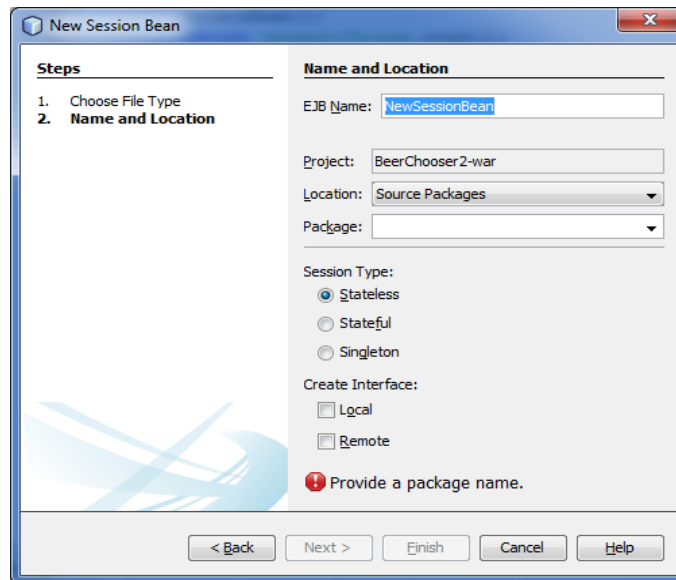


Figura 14 – Tela para configurar novos Beans de Sessão

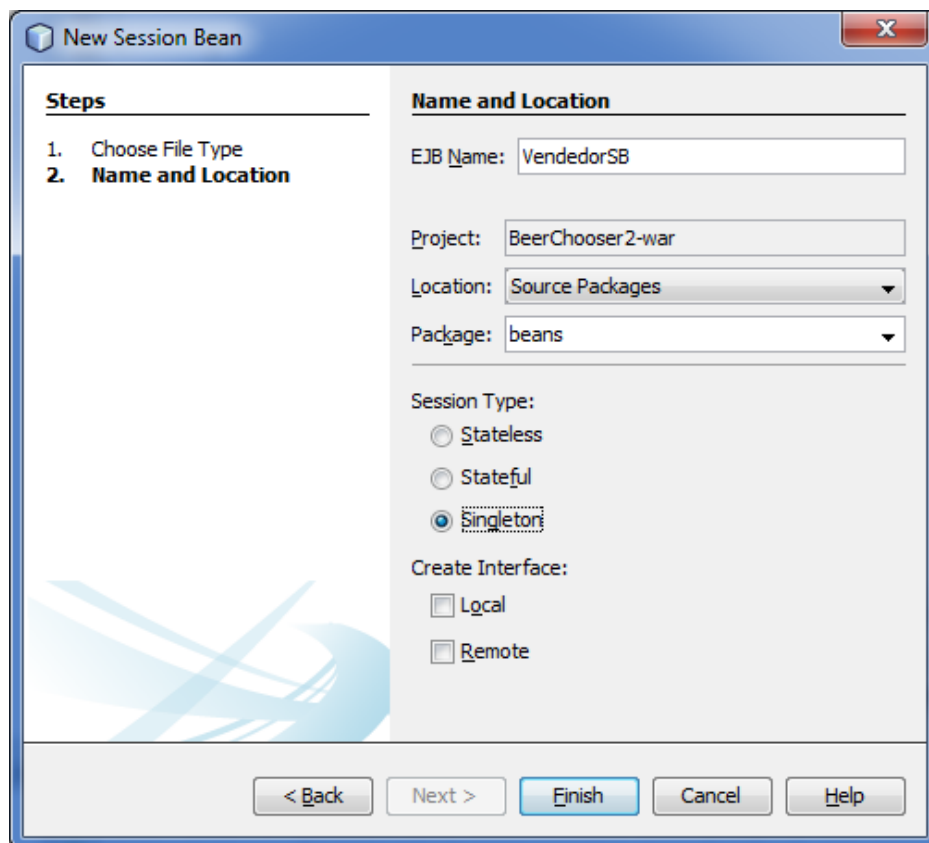


Figura 15 – Configuração do novo Bean Singleton

A Figura 15 apresenta a configuração usada neste exemplo. Não esqueça de definir o tipo de sessão como “Singleton”, que é o valor que definirá o bean como sendo do tipo Singleton e clique no botão “Finalizar”. Caso o NetBeans exija que seja adicionada uma interface para o bean, selecione a opção Local, e apague o arquivo depois que o bean for adicionado ao projeto. O resultado destes passos deve ser semelhante ao ambiente que aparece na Figura 16.

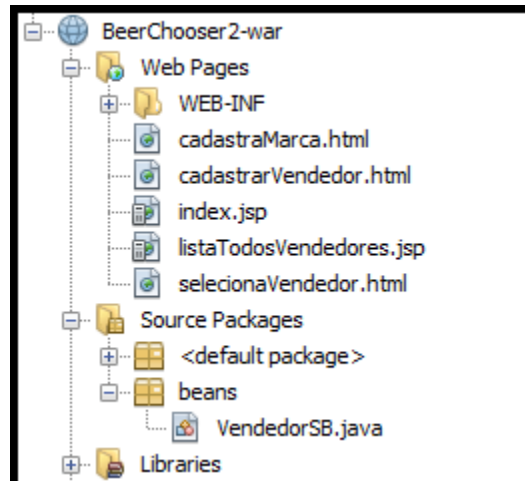


Figura 16 – Estrutura do módulo web após adição do Bean Singleton

O código do EJB final deve ficar igual ao que é apresentado na Figura 20. O primeiro passo é feito através da adição do código que enderça o EJB que está sendo usado. No caso deste exemplo, é o VendedorFacade, o que é feito através do método *locateBean()*, declarado de acordo com a Figura 17.

```
private void locateBean() {  
    try {  
        InitialContext ctx = new InitialContext();  
        Object ref = ctx.lookup("java:global/BeerChooser2/BeerChooser2-ejb/VendedorFacade");  
        vendedorFacade = (VendedorFacade) PortableRemoteObject.narrow(ref, VendedorFacade.class);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Figura 17 – Implementação do método *locateBean()*

Neste código, é necessário criar um contexto de execução do bean, para que seja possível usá-lo, uma vez que ele é ativado por esta aplicação, e sai da memória quando não é mais usado. Isso é feito pelo comando a seguir:

```
InitialContext ctx = new InitialContext();
```

Depois, é necessário usar o nome que o Bean está registrado no servidor de aplicações, e verificar se ele existe, ou se está disponível, o que é feito pelo comando lookup, como demonstrado pela linha abaixo:

```
Object ref = ctx.lookup("java:global/BeerChooser2/BeerChooser2-ejb/VendedorFacade");
```

Esse nome pode ser observado pelo log de inicialização do GlassFish, ou quando o módulo EJB é instalado no servidor de aplicações. Nesse caso, é gerada uma mensagem de acordo com esta no log do servidor (Figura 18).

```
INFO: EJB5181:Portable JNDI names for EJB VendedorFacade:
```

```
[java:global/BeerChooser2/BeerChooser2-ejb/VendedorFacade,
```

```
java:global/BeerChooser2/BeerChooser2-ejb/VendedorFacade!entidades.VendedorFacade]
```

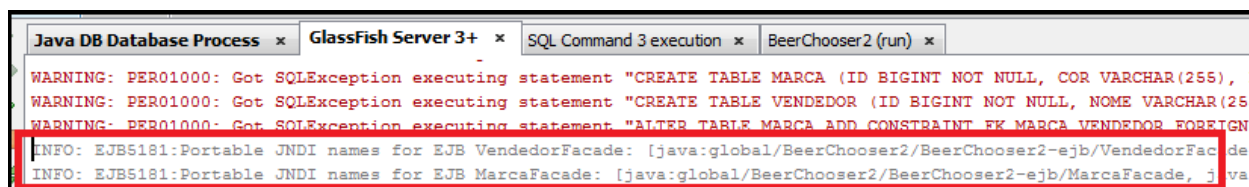


Figura 18 –Vista parcial do Log do GlassFish, em Vermelho, detalha do nome dos EJBs sendo instalados

Na última linha, a criação da ligação com o EJB:

```
vendedorFacade = (VendedorFacade)PortableRemoteObject.narrow(ref, VendedorFacade.class);
```

E qualquer erro que ocorra nesse processo, será capturado pelo comando catch, na estrutura abaixo:

```
} catch (Exception e) {  
  
    e.printStackTrace();  
  
}
```

Depois, a criação da propriedade getCount e do método findAll, conforme a figuras 19. Em ambos os casos, antes de chamar um dos métodos existentes no EJB da camada Modelo, é necessário executar a chamada ao método locateBean(), que montará a conexão com o EJB que queremos usar.

```

29 public int getCount() {
30     locateBean();
31     return vendedorFacade.count();
32 }
33
34
35 public List findAll() {
36     locateBean();
37     return vendedorFacade.findAll();
38 }

```

Figura 19 – Definição das propriedades getCount e do método findAll

```

1 package beans;
2
3 import entidades.VendedorFacade;
4 import javax.ejb.Singleton;
5 import javax.ejb.LocalBean;
6 import javax.ejb.EJB;
7 import javax.naming.InitialContext;
8 import javax.rmi.PortableRemoteObject;
9 import java.util.List;
10
11 @Singleton
12 @LocalBean
13 public class VendedorSB {
14     @EJB
15     private VendedorFacade vendedorFacade;
16
17     private void locateBean() {
18         try {
19             InitialContext ctx = new InitialContext();
20             Object ref = ctx.lookup("java:global/BeerChooser2/BeerChooser2-ejb/VendedorFacade");
21             vendedorFacade = (VendedorFacade) PortableRemoteObject.narrow(ref, VendedorFacade.class);
22         } catch (Exception e) {
23             e.printStackTrace();
24         }
25     }
26
27
28
29     public int getCount() {
30         locateBean();
31         return vendedorFacade.count();
32     }
33
34
35     public List findAll() {
36         locateBean();
37         return vendedorFacade.findAll();
38     }
39 }
40

```

Figura 20 – Listagem Completa do Bean Singleton

O passo seguinte, é modificar a página jsp. Para isso, clique duas vezes sobre o arquivo listarTodosVendedores.jsp para editar o código, e altere-o para que fique igual ao da Figura 21.

```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <%@ page import="java.util.List, entidades.Vendedor;"%>
3 <%@ page import="java.util.Iterator;"%>
4 <jsp:useBean id="vendFac" scope="page" class="beans.VendedorSB"/>
5 <html>
6 <head>
7 <title>Lista Todos Vendedores</title>
8 </head>
9 <body>
10 <p>
11 <u>Página de Exemplo - Listagem de Vendedores</u>
12 </p>
13 <br>
14 <table style="text-align: left; width: 378px; height: 80px;" border="0"
15 <cellpadding="2" cellspacing="2">
16 <tbody>
17 <tr>
18 <td>Id</td>
19 <td>Nome cadastrado</td>
20 </tr>
21 <%
22 List vendedores = vendFac.findAll();
23 for (Iterator i = vendedores.iterator(); i.hasNext();) {
24 Vendedor vend = (Vendedor) i.next();
25 %>
26 <tr>
27 <td><%= vend.getId() %></td>
28 <td><%= vend.getNome() %></td>
29 </tr>
30 <%
31 }
32 %>
33 </tbody>
34 </table>
35 <br>
36 <p>
37 <u>Número total de vendedores:</u>
38 <jsp:getProperty name="vendFac" property="count"/>
39 </p>
40 <a href="index.jsp">Voltar</a>
41
42 </body>
43 </html>
```

Figura 21 – Página JSP exibindo todos os dados cadastrados no banco de dados



Observando o código-fonte, as tags de JSP “<%@ page import” indicam as classes que precisam ser declaradas para que o programa funcione corretamente, isso é o mesmo que ocorre nas demais classes Java.

Depois, há a tag da linha 4, <jsp:useBean id="vendFac" scope="page" class="beans.VendedorSB"/>, que indica a declaração do bean que está sendo usado pela página JSP. O comando *id* indica o nome da variável que eu declarei para usar o bean e o comando *scope* indica que ele somente será usado enquanto a página está sendo formatada, depois disso, os dados são apagados da memória. Isso seria equivalente a declarar numa classe Java:

```
beans.Vendedor vendFac;
```

Depois, entre as linhas 21 e 25, há a declaração da interação que será feita para incluir todas as linhas obtidas a partir da consulta ao banco de dados. Na linha 21, é declarada a variável *vendedores* como sendo do tipo *List*, que receberá todo o conteúdo passado pelo método *findAll()*. Como não é possível imprimir toda a lista de uma vez, é necessário pegar linha a linha e imprimí-lo na tela. Isso é feito através do comando *for* declarado na linha 23, que recebe da variável *vendedores* se há resultado a ser tratado, e se houver, é montado um objeto do tipo *Vendedor* na linha 24.

Nas linhas 27 e 28, são chamados os métodos *getId()* e *getNome()* para imprimir os valores na tela. Na linha 31, há a chave que fecha o *for*, iniciado na linha 23, e entre eles a tag HTML <tr> na linha 26. Isso é necessário ficar desse modo, pois esse comando *for* é que controla a impressão das linhas da tabela, de acordo com os dados lidos do banco de dados.

Assim, esta parte do projeto já está pronta para ser executada. Para isso, clique com o botão direito do Mouse no nome do projeto web ( BeerChooser) e selecione a opção “Limpar e Construir”. Quando isso acabar, clique com o botão direito do mouse no nome do projeto corporativo ( BeerChooser), e selecione as opções “Limpar e Construir” e ao término da mesma, clique em Executar. O resultado deve ser similar ao das Figuras 22 a 23.

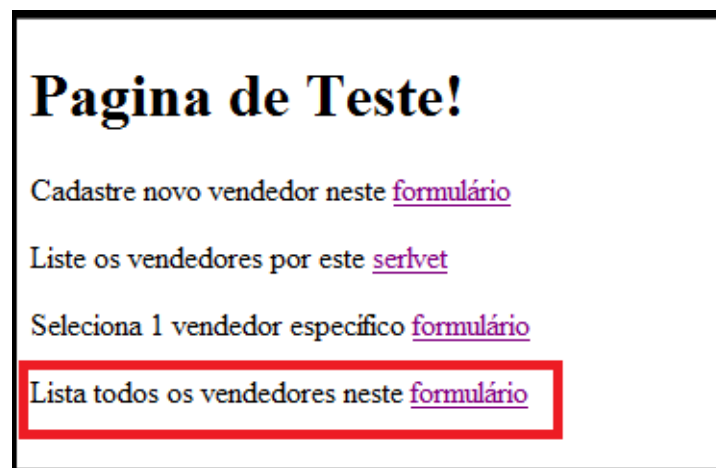


Figura 22 – Página de entrada – Área em vermelho é o link a ser testado

Página de Exemplo - Listagem de Vendedores	
Id	Nome cadastrado
1	Adega Praiano
2	JMC Bebidas
3	Praiano Bebidas
4	Empório Santa Inês
5	2
6	,bnnnb,
7	J Martins
8	Antunes Bebidas
9	MM Produtos de Festa
10	De Maison
11	Maison des Caves
12	Metre DAvis
Número total de vendedores: 12	
Voltar	

Figura 23 – Tela com o resultado da Listagem

Finalizando o Aplicativo

Até o momento, há as telas de cadastramento, usando uma mistura de formulários escritos em HTML (Camada View) que chamam Servlets (Camada Controle), que coletam os dados informados pelos formulários e os repassam para o EJB (Camada Modelo) que manipula a base de dados para executar as operações de persistência.

Depois, foi incluída uma página que lista todos os itens cadastrados em banco de dados. O desenho dessa lista está numa página JSP que contém a estrutura de desenho da tela, formatado em HTML, e misturado nesse código, uma parte de código Java que executa um bean de tipo Singleton para buscar os dados no banco de dados. Nesse caso, o arquivo JSP forma a Camada View, o Bean Singleton forma a Camada Controle que usam a Camada Modelo, representada através do EJB.

Para finalizar esse aplicativo, basta excluir os vendedores que não desejamos mais. Para isso, é possível montar um link para cada item da lista, que executa um Servlet que acessa o EJB para excluir o item solicitado do banco de dados, e para facilitar, ao realizar essa tarefa, o Servlet redireciona a chamada

para a tela contendo a lista de vendedores, que será mostrada atualizada, após a exclusão de algum item, conforme fluxo representado na Figura 24.

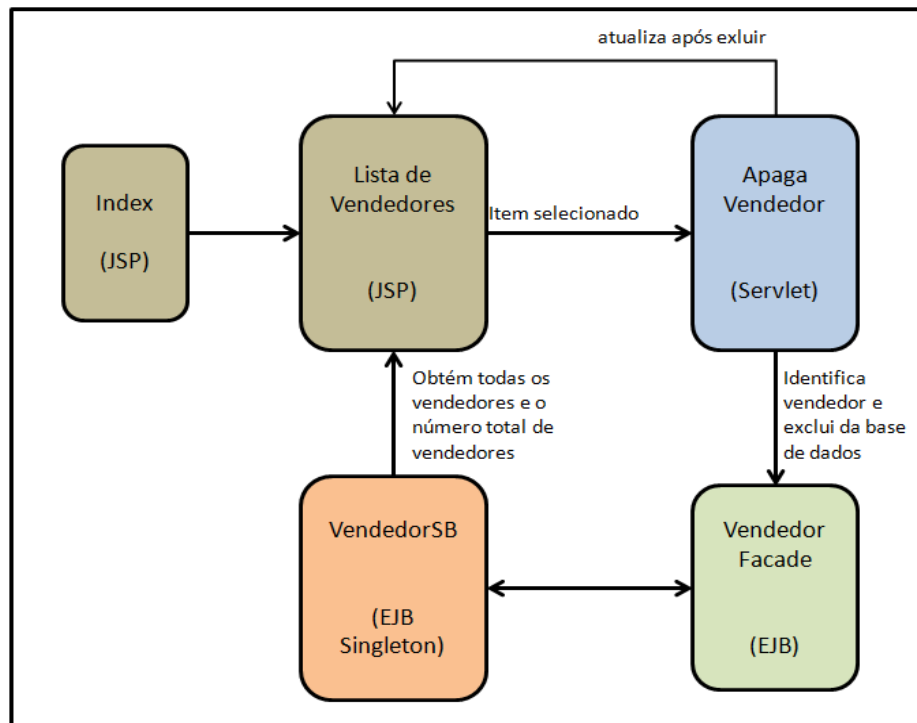


Figura 26 – Fluxo de páginas do aplicativo

Neste ponto, vale comentar um aspecto sobre os Servlets. Quando um formulário envia parâmetros para um Servlet, isso ocorre através da propriedade *action* da tag HTML *form*. Tomando como exemplo o código HTML da Figura 27, a propriedade *action* aponta para um link chamado *ValidaCliente*. No momento em que o usuário clicar no botão para enviar o formulário, o browser submeterá os dados através do link “*ValidaCliente?nome=‘valor digitado pelo usuário’*”.

```
<form enctype="application/x-www-form-urlencoded" method="get"
  action="ValidaCliente" name="cadastrarCliente">

  Nome: <input name="nome"><br>
  <br>
  <button name="enviar"></button><br>
</form>
```

Figura 27 – Exemplo de formulário

Nesse link, os valores após o “?” são a lista de parâmetros enviados pelo formulário, e que devem ser lidos pelo Servlet através do comando `request.getParameter(“nome do parâmetro”)`. Exemplo disso é o Servlet codificado para cadastrar os vendedores sendo adicionados ao sistema.

Assim para montar uma lista que apresenta os vendedores a serem excluídos do sistema, é necessário montar o mesmo tipo de link que o formulário monta, quando o usuário clica no botão para enviar as informações. Um jeito de fazer isso, é montar a tag “`<a href=`” de HTML dinamicamente. Para isso, basta adicionar mais uma coluna à tabela definida no arquivo “`listaTodosVendedores.jsp`” com as tags de HTML `<td>` e a montagem do link através do tag HTML `<a href`, conforme a Figura 28.

```
26 <tr>
27 <td><%= vend.getId() %></td>
28 <td><%= vend.getNome() %></td>
29 <td><a href="ApagaVendedor?id=<%=vend.getId().toString() %>">excluir</a></td>
30 </tr>
```

Figura 28 – Alteração da tabela para montar um link criado dinamicamente

Como pode ser observado na Figura 28, é adicionado à tabela um link que chama um Servlet chamado `ApagaVendedor`, e que passa o parâmetro `id`, adicionado ao link pela tag `<%=vend.getId().toString() %>`, enquanto aparecerá escrito para o usuário excluir.

```
26 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
27     throws ServletException, IOException {
28     response.setContentType("text/html;charset=UTF-8");
29     PrintWriter out = response.getWriter();
30     try {
31
32         Vendedor vend = new Vendedor();
33         vend.setId(Long.parseLong(request.getParameter("id")));
34         vend = vendedorFacade.find(Long.parseLong(request.getParameter("id")));
35         vendedorFacade.remove(vend);
36
37         request.getRequestDispatcher("listaTodosVendedores.jsp").forward(request, response);
38
39     } catch (Exception e) {
40         e.printStackTrace();
41     } finally {
42         out.close();
43     }
44 }
```

Figura 29 – Método `processRequest` do Servlet `ApagaVendedor`

O próximo passo é criar o Servlet de controle chamado *ApagarVendedor*, que receberá o *id* como parâmetro, e que fará com que o item selecionado pelo usuário seja apagado do banco e dados. O processo de criação do Servlet é o mesmo usado para criar os outros Servlets usados neste aplicativo. Depois de criado, o método *processRequest* do Servlet deve ser alterado para que fique igual ao da Figura 29. Outra alteração que não pode ser esquecida nesse Servlet, é o endereçamento do EJB *VendedorFacade* (Figura 30).

```
11
12  @WebServlet(name = "ApagaVendedor", urlPatterns = {"/ApagaVendedor"})
13  public class ApagaVendedor extends HttpServlet {
14      @EJB
15      private VendedorFacade vendedorFacade;
16  }
```


Figura 30 - Declaração do EJB *VendedorFacade* no Servlet *ApagaVendedor*


Como pode ser observado na linha 32 da Figura 29, é criado o objeto *vend*, que é do tipo da classe de entidade *Vendedor*. Na linha 33, o comando *request.getParameter("id")* lê o parâmetro *id*, que é informado pelo link, depois, ele é convertido para um número do tipo *Long* pelo comando *Long.parseLong*, e por último, o resultado de tudo isso é atribuído ao objeto *vend* pelo método *setId*.

Quando se trata parâmetros de links HTML, é sempre necessário convertê-los para número, pois os links HTML somente trafegam valores de tipo texto. Então, caso a aplicação use algum valor numérico, como é o caso do atributo *id* da classe *Vendedor*, tornar-se obrigatório fazer a conversão do mesmo. Caso contrário, o aplicativo apontará um erro do de tipo de dado incompatível.

Na linha 34, o método *find* do objeto *vendedorFacade* usa a JPA para encontrar os dados da tabela *vendedor*, e copia o valor lido da base de dados para o objeto *vend*. Depois, na linha 35, é solicitado para a JPA que apague esse resultado da tabela *vendedor*, ao ser usado o método *remove* do objeto *vendedorFacade*.

Por último, a linha 37 redireciona o aplicativo para chamar novamente a página que lista todos os vendedores existentes na base de dados, através do comando *request.getDispatcher("caminho da página").forward(request, response)*.

Para testar as modificações, clique com o botão direito do Mouse no nome do projeto web ( BeerChooser) e selecione a opção "Limpar e Construir". Quando isso acabar, clique com o botão

direito do mouse no nome do projeto corporativo ( BeerChooser), e selecione as opções "Limpar e Construir" e ao término da mesma, clique em Executar. O resultado deve ser similar ao das Figuras 22 a 23.

Página de Exemplo - Listagem de Vendedores

Id	Nome cadastrado	
3	Praiano Bebidas	excluir
4	Empório Santa Inês	excluir
8	Antunes Bebidas	excluir
9	MM Produtos de Festa	excluir
11	Maison des Caves	excluir
13	Quinta do Marquês	excluir

Número total de vendedores: 6

[Voltar](#)

Figura 31 – Tela com a lista de vendedores, após inclusão da opção excluir

Página de Exemplo - Listagem de Vendedores

Id	Nome cadastrado	
3	Praiano Bebidas	excluir
4	Empório Santa Inês	excluir
9	MM Produtos de Festa	excluir
11	Maison des Caves	excluir
13	Quinta do Marquês	excluir

Número total de vendedores: 5

[Voltar](#)

Figura 32 – Tela com a lista de vendedores, após excluir o ID 8

Exercícios

1. Repita o mesmo procedimento para cadastrar os dados da tabela Marca;
2. Aponte 2 vulnerabilidades do código-fonte deste exemplo;