Tópicos Especiais em Análise e Desenvolvimento de Sistemas

Prof. Rafael Odon (rafael.alencar@prof.una.br)

Java Server Faces

Roteiro de Aula 04

Esse Roteiro se baseia no uso dos softwares:

- Netbeans 7.4 para JAVA EE
- GlassFish 4
- Java SDK 1.7
- Java EE 6
- Java DB (Apache Derby)

Fase 1: Aplicações Ricas para Internet

Um conceito amplamente difundido no desenvolvimento web são as Rich Internet Applications (RIA), ou seja, aplicações web ricas. Em outras palavras, busca-se levar para a janela do navegador as mesmas possibilidades de interação com o usuário das aplicações desktop.

Existem diversas tecnologias que permitem perseguir tal objetivo, dentre eles o Adobe Flex, Google Web Toolkit (GWT), JavaFX, dentre outros. No universo do JSF, quando queremos aproximar nossas aplicações dos princípios do RIA, basicamente estamos falando de duas coisas:

1. Utilização de componentes ricos que imitem o comportamento de componentes Desktop através do uso intensivo de HTML, JavaScript e CSS.

2. Projeto de telas que usem e abusem de Ajax para melhorar a experiência do usuário.

Uma das vantagens de se utilizar o JSF é que há a possibilidade de criar ou utilizar bibliotecas de componentes de terceiros, fazendo com que o framework não se limite a um conjunto específico de recursos. Sendo assim, uma biblioteca que se destaca ao facilitar a implementação dos dois aspectos acima mencionados no RIA é o PrimeFaces (http://primefaces.org/).

Outras bibliotecas existem, tais como o RichFaces e o IceFaces. Dentre as principais vantagens de se escolher o PrimeFaces é o seu amplo suporte ao Ajax e o fato de poder ser personalizado através de temas (http://www.primefaces.org/themes.html), além de possuir um excelente showcase onde é possível ver exemplos de uso e código de cada componente oferecido (http://www.primefaces.org/showcase/ui/home.jsf). Ainda, dentre as três bibliotecas mencionadas, no brasil o PrimeFaces é a mais utilizada com JSF 2.

Fase 1: Adicionando o PrimeFaces ao seu projeto

1. Acesse o site do PrimeFaces, na seção de downloads (http://primefaces.org/downloads.html), baixe a versão 3.5 Community:

   http://repository.primefaces.org/org/primefaces/primefaces/3.5/primefaces-3.5.jar

2. Copie o **arquivo baixado, primefaces-3.5.jar**, para a **pasta lib** do **seu projeto**. Se essa pasta não existir, crie-a primeiro. Ex:

/home/rafael/NetBeansProject/**SysFrotaWeb**/**lib**/**primefaces-3.5.jar**

3. No Netbeans, adicione o JAR do primefaces ao seu projeto: No Projeto, botão direito no item **Biblioteca** > **Adicionar JAR/Pasta** > Selecionar o **arquivo primefaces-3.5.jar** previamente adicionado na pasta lib no passo anterior.

Fase 2: Alterando o template para utilizar componentes do PrimeFaces

1. Modifique o seu arquivo **template.xhtml** para contemplar o uso de algumas tags já **do namespace do primefaces.**

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title>SysFrota</title>
    </h:head>
    <h:body>
        <p:panel>
            <h1>SysFrota</h1>
            <h:form>
                <p:menubar>
                    <p:menuitem action="/index.xhtml" value="Inicio" ajax="false"/>
                    <p:menuitem action="/fabricanteList.xhtml" value="Manter Fabricante" ajax="false"/>
                    <p:menuitem action="/modeloList.xhtml" value="Manter Modelo" ajax="false"/>
                    <p:submenu label="Ajuda">
                        <p:menuitem action="/sobre.xhtml" value="Sobre" ajax="false"/>
                    </p:submenu>
                </p:menubar>
            </h:form>
            <br/>
            <p:messages autoUpdate="true" globalOnly="true" />
            <ui:insert name="corpo"></ui:insert>
        </p:panel>
        <br/>
        <div class="rodape" style="text-align:center">
            SysFrotaWeb com PrimeFaces - Junho/2013<br/>
        </div>
    </h:body>
</html>
```

2. Observe que no código acima, adicionamos um **<p:panel />**. Na prática ele será renderizado como uma com bordas arredondadas que irá embrulhar o conteúdo do sistema além de adicionar alguns estilos aos elementos de texto que ocorrerem dentro dele.

3. Também foi utilizado um componente bem interessante, **o <p:menubar />, que será renderizado como uma barra de menu horizontal** bem atraente. Cada **item de menu deve ser representado por um <p:menuitem />**, que se tornará um botão que ao ser clicado disparará uma ação, semelhante ao conhecido <h:commandButton />. Para isso, perceba que **foi necessário um <h:form /> ao redor do menu.**
Pode-se optar também por adição de **submenus, através das tags <p:submenu />**, que podem conter outros <p:menuitem /> permitindo a criação de hierarquias.
Atenção! No PrimeFaces, tudo é feito via Ajax por padrão. Para evitar que os cliques do menu façam uma navegação Ajax, o que não é desejado ainda, **os itens de menu foram configurados com ajax=”false”**.

Não se assuste com a quantidade de nomes novos!
Para saber mais sobre um componente, por exemplo a barra de menu, **acesse seu showcase** no site do PrimeFaces (http://www.primefaces.org/showcase/ui/menubar.jsf) onde é possível ver exemplos e códigos de inspiração.
Na dúvida recorra sempre à **documentação do PrimeFaces** para saber o que cada propriedade dos componentes faz e como utilizá-lo (http://primefaces.org/documentation.html).

4. **Importante!** Lembre-se de adequar a barra de menu acima aos links disponíveis e nomes de páginas da sua aplicação.

5. Um outro componente que traz um grande atrativo visual é o **<p:messages />, que nesse caso, substitui o antigo <h:messages />, recebendo todas as mensagens de sucesso/erro/aviso enviadas pela aplicação**, porém já estilizadas com cores e ícones atraentes.
   Observe que ele foi configurado com a **opção autoUpdate="true"** para que mensagens oriundas de cliques Ajax sejam automaticamente apresentadas, tendo em vista que a página não será recarregada. Já a configuração **globalOnly="true"** já era feita no <h:messages /> e faz com que apenas as mensagens globais, ou seja, destinadas a nenhum componente específico, apareçam nesse local. Na tela de edição que faremos abordaremos novamente esse assunto.

6. **Importante!** Além das novidades introduzidas, é importante observar que esse template cria **um ponto de inserção chamado "corpo"**. Certifique que esse nome é compatível com o ponto de inserção esperado pelas páginas da sua aplicação. Se necessário faça adaptações.

7. Limpe e construa a aplicação e implante-a no GlassFish para verificar as novidades apresentadas.

Fase 3: Melhorando suas listagens com o DataTable do PrimeFaces

1. O componente <p:dataTable /> é o correspondente do <h:datatable /> do PrimeFaces. Dentre outras coisas, ele permite introduzir paginação, filtro e ordenação por coluna, tudo por Ajax. Modifique a tabela do seu XHTML de listagem de fabricantes inspirado no exemplo abaixo.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core" xmlns:p="http://primefaces.org/ui" template="template.xhtml" >
    <ui:define name="corpo">
        <h:form>
            <h1>Lista de Fabricantes</h1>
            <h:commandButton action="#{fabricanteMB.criar()}" value="Criar Novo"/>
            <p:dataTable value="#{fabricanteMB.lista}"  var="f"
                         paginator="true" rows="5" first="#{fabricanteMB.primeiro}" >
                <p:column headerText="Id" sortBy="#{f.id}">
                    <h:outputText value="#{f.id}" />
                </p:column>
                <p:column headerText="Nome" sortBy="#{f.nome}" filterBy="#{f.nome}">
                    <h:outputText value="#{f.nome}" />
                </p:column>
                <p:column headerText="Ações" style="width:230px">
                    <h:commandButton value="Editar" action="#{fabricanteMB.editar(f)}"/>
                    <h:commandButton value="Remover" action="#{fabricanteMB.remover(f)}"/>
                </p:column>
            </p:dataTable>
        </h:form>
    </ui:define>
</ui:composition>
```

2. Observe que o XHTML acima declara novamente **o namespace do PrimeFaces**, e faz uso do **template apresentado na fase 2.** Adapte o que for necessário para a sua aplicação.

3. Para fazer a listagem, introduzimos a **utilização do componente <p:dataTable />** que é muito similar a do <h:dataTable /> que já conhecemos para construir tabelas. Note também que cada **coluna é representada por um <p:column />** ao invés de um <h:column />, e, como antes, dentro de cada column podemos manipular a *variável f* definida.

4. Observe que a tag <p:dataTable /> no exemplo indicou que **será usada paginação (paginator="true") com 5 registros por página (rows="5").** Essa paginação fornecida pelo PrimeFaces por padrão é feita com Ajax. Para que ela funcione corretamente é preciso também informar para o componente **uma propriedade no ManagedBean para manter em memória o primeiro resultado da página corrente (first="#{fabricanteMB.primeiro}")**. Sendo assim, adicione também no seu ManagedBean de Fabricante **a propriede chamada "primeiro":**

```
...
public class FabricanteMB {
    ...
    private Long primeiro;
    ...
    public Long getPrimeiro() {
        return primeiro;
    }

    public void setPrimeiro(Long primeiro) {
        this.primeiro = primeiro;
    }
    ...
}
```

5. Ainda no XHTML, perceba que o antigo <f:facet /> do cabeçalho foi abandonado e **cada <p:column /> define o texto do cabeçalho através da propriedade headerText.**

6. É possível também definir para cada coluna, se for desejado, **uma propriedade da *variável f* para promover ordenação (ex: sortBy="#{f.nome}")**, e se também for desejado, pode-se informar **o campo que deve ser usado para filtro textual dos registros (ex: filterBy="#{f.nome}")**. Ambos os recursos também usam e abusam do Ajax.

7. Pare o Glassfish. Limpe e construa a aplicação e implante-a no GlassFish novamente para verificar as novidades apresentadas.

Fase 4: Embelezando suas páginas com alguns outros componentes do PrimeFaces

1. Vamos adicionar alguns outros detalhes a essa página de listagem criada para torná-la ainda mais atraente. Acrescente algumas novidades no XHTML da listagem de fabricante, como segue:

```
        ...
        <h:form>
            <p:fieldset legend="Lista de Fabricantes">
                <h:panelGroup layout="block" style="text-align: right">
                    <p:commandButton action="#{fabricanteMB.criar()}" value="Criar Novo"
                                     icon="ui-icon-document" ajax="false"/>
                </h:panelGroup>
                <p:dataTable value="#{fabricanteMB.lista}"  var="f"
                    ...
                    <p:column headerText="Ações" style="width:230px">
                        <p:commandButton value="Editar" action="#{fabricanteMB.editar(f)}"
                                         icon="ui-icon-pencil" ajax="false"/>
                        <p:commandButton value="Remover" action="#{fabricanteMB.remover(f)}"
                                         icon="ui-icon-trash" ajax="false"/>
                    </p:column>
                </p:dataTable>
            </p:fieldset>
        </h:form>
        ...
```

2. Inspirado no código XHTML acima, envolva todo o conteúdo da sua listagem em um **<p:fieldset /> para adicionar uma borda com legenda** que ajuda a delimitar os grupos de componentes visualmente relacionados.

3. Converta seus <h:commandButton /> em **<p:commandButton />**. Atente para o fato de que os commandButton e commandLink do PrimeFaces por padrão são Ajax. Como esse comportamento não é desejado ainda, **a propriedade ajax deve ser configurada para false.**

4. Botões do PrimeFaces permitem **adicionar um ícone** do mapa de ícones do Jquery, bastando para isso configurar **a propriedade icon com o respectivo nome do ícone**. Esses nomes podem ser facilmente encontrados na internet (Procure por *jquery icons,* Ex: http://jquery-ui.googlecode.com/svn/tags/1.6rc5/tests/static/icons.html).

5. Afim de dispor corretamente o botão de *criar fabricante* na tela, **foi colocado um <h:panelGroup /> ao seu redor** com um estilo que alinha o conteúdo à direita. Perceba que mesmo utilizando PrimeFaces, podemos e precisamos conviver com vários componentes nativos do JSF (*namespace h*).

6. Você também pode aplicar algumas melhorias no XHTML de edição de fabricante. Segue um exemplo de inspiração:

```
<?xml version="1.0" encoding="UTF-8"?>
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core" xmlns:p="http://primefaces.org/ui" template="template.xhtml" >
    <ui:define name="corpo">
        <h:form>
            <p:fieldset legend="Editar Fabricante">
                <h:panelGrid columns="3">
                    <h:outputLabel value="Id:"/>
                    <h:outputText value="#{fabricanteMB.fabricante.id}" />
                    <h:outputText />

                    <p:outputLabel value="Nome:" for="idNomeFabricante"/>
                    <p:inputText id="idNomeFabricante" value="#{fabricanteMB.fabricante.nome}"
                                 required="true" label="Nome" />
                    <p:message for="idNomeFabricante" />
                </h:panelGrid>

                <h:panelGroup layout="block" style="text-align: right" >
                    <p:commandButton value="Cancelar" action="#{fabricanteMB.cancelar()}" immediate="true"
                                     icon="ui-icon-arrowreturn-1-w" ajax="false" />
                    <p:commandButton value="Salvar" action="#{fabricanteMB.salvar()}"
                                     icon="ui-icon-disk" ajax="false" />
                </h:panelGroup>
            </p:fieldset>
        </h:form>
    </ui:define>
</ui:composition>
```

7. A novidade que o XHTML de edição acima apresenta é a utilização de campos de entrada do PrimeFaces. No caso acima, o <h:inputText /> foi substituído pelo **<p:inputText />**. Para acompanhá-lo, também foi usado um **<p:outputLabel />** que **exige que seja informado o Id do input que cujo o label faz referência**.

8. Além disso, perceba que **colocamos um <p:message />** (não confundir com <p:messages /> no plural), que é responsável por receber todas as **mensagens de erro relativas apenas ao campo Nome.**

9. Pare o Glassfish. Limpe e construa a aplicação e implante-a no GlassFish novamente para verificar as novidades apresentadas.

Fase 5: Fazendo um CRUD 100% Ajax com o PrimeFaces

1. Vamos exemplificar aqui como seria um crud 100% ajax, onde a edição e a listagem ocorrem na mesma página e não há refresh entre as ações. Tenha em mente que o código que será mostrado é muito próximo do código já construído até aqui, precisando apenas de pequenos retoques.

2. Modifique seu ManagedBean de fabricante **substituindo a anotação @SessionScoped pela @ViewScoped**. Ao invés desse ManagedBean sobreviver na memória durante toda a sessão do usuário, ele será mantido apenas durante uma "visualização", ou seja, enquanto o usuário permanecer naquela página específica.

```
...
@ManagedBean
@SessionScoped
@ViewScoped
public class FabricanteMB {
    ...
}
```

3. Além disso, faça com que **TODOS** os seus **métodos que eram ações de botões e que antes retornavam String sejam void, deixando de retornar páginas de listagem ou edição.** Não haverá navegação no nosso exemplo, tudo será feito em uma única página. Veja o exemplo com o método da ação editar:

```
...
@ManagedBean
@ViewScoped
public class FabricanteMB {
    ...
    public String editar(Fabricante f){
        this.fabricante = fabricanteDAO.carregarPeloId(f.getId());
        return PAGINA_DE_EDICAO;
    }
    public void editar(Fabricante f){
        this.fabricante = fabricanteDAO.carregarPeloId(f.getId());
    }
    ..
}
```

4. Agora, vamos construir um XHTML que é tanto de listagem, quanto de edição. A ideia é que, ao clicar em editar em algum fabricante da tabela, ele é editado no formulário acima da tabela. O Ajax será utilizado para carregar o fabricante editado no formulário e para atualizar a tabela após uma edição/remoção de um fabricante, sem precisar de refresh de página. Acompanhe...

```
<?xml version="1.0" encoding="UTF-8"?>
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:p="http://primefaces.org/ui"
                template="template.xhtml" >
    <ui:define name="corpo">
        <h:form id="form">
            <p:fieldset legend="Editar Fabricante">
                <h:panelGroup id="camposEditarFabricante">
                    <h:panelGrid columns="3">
                        <h:outputLabel value="Id:"/>
                        <h:outputText value="#{fabricanteMB.fabricante.id}" />
                        <h:outputText />

                        <p:outputLabel value="Nome:" for="idNomeFabricante"/>
                        <p:inputText id="idNomeFabricante" value="#{fabricanteMB.fabricante.nome}"
                                     required="true" label="Nome" />
                        <p:message for="idNomeFabricante" />
                    </h:panelGrid>
                </h:panelGroup>
```

```
                <h:panelGroup layout="block" style="text-align: right" >
                    <p:commandButton value="Cancelar" immediate="true" icon="ui-icon-arrowreturn-1-w"
                                     update=":form:camposEditarFabricante"/>
                    <p:commandButton value="Salvar" action="#{fabricanteMB.salvar()}" icon="ui-icon-disk"
                                     update=":form:camposEditarFabricante,:form:fabricanteDataTable" />
                </h:panelGroup>
            </p:fieldset>
            <br/>
            <p:fieldset legend="Lista de Fabricantes">
                <p:dataTable id="fabricanteDataTable" value="#{fabricanteMB.lista}"  var="f" paginator="true"
                             first="#{fabricanteMB.primeiro}" rows="5">
                    <p:column headerText="Id" sortBy="#{f.id}">
                        <h:outputText value="#{f.id}" />
                    </p:column>
                    <p:column headerText="Nome" sortBy="#{f.nome}" filterBy="#{f.nome}">
                        <h:outputText value="#{f.nome}" />
                    </p:column>
                    <p:column headerText="Ações" style="width:230px">
                        <p:commandButton value="Editar" action="#{fabricanteMB.editar(f)}"
                                         icon="ui-icon-pencil"
                                         process="@this" update=":form:camposEditarFabricante" />
                        <p:commandButton value="Remover" action="#{fabricanteMB.remover(f)}"
                                         icon="ui-icon-trash"
                                         process="@this" update=":form:fabricanteDataTable"/>
                    </p:column>
                </p:dataTable>
            </p:fieldset>
        </h:form>
    </ui:define>
</ui:composition>
```

5. De cima para baixo, vamos entender cada detalhe do XHTML acima.

   I. Em primeiro lugar, como já estamos acostumados, ele usa o nosso template e os namespaces já conhecidos. Não há novidade nessa parte.

   II. Uma novidade é que precisamos **colocar um id para o <h:form />**. No JSF, o id de um componente geralmente é composto pelo id do formulário e o seu próprio id. Como usaremos Ajax, precisamos conhecer os ids das coisas e por isso precisamos controlá-los, daí a necessidade de definir o id do nosso formulário. Você entenderá adiante...

   III. Visualmente, dividimos o layout nossa página de CRUD em **um *fieldset* (borda com legenda) para edição e um fieldset para listagem**. Isso nada tem a ver com Ajax ainda, é apenas o desenho da página.

   IV. No *fieldset* de edição, foi preciso **criar um <h:panelGroup /> para abranger os campos do formulário**, bem como definir um id para esse grupo de componentes. Isso é necessário pois ações Ajax precisarão atualizar essa porção da página sem atualizar a página por inteiro, e para isso, ela precisa de um id que a identifique.

   V. Perceba que ainda dentro do *fieldset* de edição, abaixo dos campos do formulário, temos os nossos 2 botões, o de Salvar e o de Cancelar.

   O botão de Salvar não possui mais a opção **ajax=”false”**, e sendo assim, ele passa a ser Ajax por padrão. Por ser Ajax, **é preciso indicar, além da ação executada no ManagedBean, que componentes na página serão atualizados ao término da execução da ação.** Isso é feito referenciando o id completo dos elementos que devem ser atualizados. Perceba que referenciamos o conjunto de campos agrupados no passo anterior.

   O botão Cancelar também é Ajax, e perceba que ele ainda é **immediate=”true”**, ou seja, não requisita validação e atualização do modelo. Essa regra continua valendo para o Ajax. O interessante é que **o cancelar pede para que seja atualizado na página tanto os campos de edição quanto a tabela**, separando por virgula o id desses dois conjuntos de componentes.

Esclarecimento

No JSF, o Id de um componente é construído da seguinte forma: id_do_pai:id_do_componente. Alguns componentes, mas não todos, interferem nessa construção de Id, fazendo papel de pai. O mais comum deles é o <h:form />. O id do form é atribuído a todos os componentes descritos dentro dele. Exemplo: meu_form:meu_input. Se você não determina o id nem do form nem do componente, o JSF irá gerar um id muito difícil de lidar e que pode variar. Ex: j_id34:j_id89.

Na atualização Ajax, uma dica geral é começar a expressão de id referenciado com dois pontos para indicar o caminho absoluto. Ex: meu_form:meu_campo → **:**meu_form:meu_campo.

VI. **A tabela também recebeu um id,** e você já deve imaginar que isso foi necessário para que ela fosse atualizada por chamadas Ajax como foi o caso do botão Salvar.

VII. O botão de Editar da tabela também é Ajax e **atualiza os campos do formulário.** Mas além disso, ele também **configura que componentes são processados pela requisição Ajax. Nesse caso, ele foi configurado para processar apenas a sí próprio**, o que significa que ele não enviará os dados do formulário por exemplo para o Managed Bean. É o que chamamos de requisição parcial. Para saber mais sobre isso veja o exemplo em http://www.primefaces.org/showcase/ui/pprPartialTree.jsf.

VIII. Da mesmo forma, o botão Remover **atualiza a tabela** para refletir a remoção e também **pede para que aconteça uma requisição parcial**, apenas enviando quem será removido, sem enviar os campos do formulário.

6. Pare o Glassfish. Limpe e construa a aplicação e implante-a no GlassFish novamente para verificar as novidades apresentadas.

7. Alguns erros comuns ao usar Ajax:

   I. Esquecer do **immediate="true"** ou do **process="@this"** ou **process="@none"** para evitar que todo formulário seja validado e enviado para o ManagedBean.

   II. Esquecer que o seu **<p:messages />** no template deve estar configurado com **autoUpdate="true"** para receber mensagens enviadas pela apliacação sem precisar ser atualizado explícitamente via Ajax.

   III. Tentar atualizar campos diretamente, um a um. Prefira criar agrupamentos identificados utilizando <h:panelGroup />.

   IV. Mencionar o id de forma errada na atualização de campos. Inspecione no navegador o campo para descobrir seu id real.

Fase 6: Escolhendo um tema do PrimeFaces para sua aplicação

1. O PrimeFaces permite criar temas ou mesmo escolher um tema pronto da sua galeria para modificar o aspecto visual da sua aplicação.

2. Navegue em http://primefaces.org/themes.html escolhendo um tema e verificando no showcase a aparência que ele gera. Guarde o nome do tema escolhido. Exemplo: ***flick***

3. Faça o download do **arquivo JAR** da última versão do tema no repositório do PrimeFaces em http://repository.primefaces.org/org/primefaces/themes/ e coloque **na pasta lib do seu projeto**, assim como você fez com o JAR do PrimeFaces. Exemplo:

```
/home/rafael/NetBeansProject/SysFrotaWeb/lib/flick-1.0.10.jar
```

4. Adicione o JAR do tema às bibliotecas do seu Projeto pelo NetBeans, assim como você fez anteriormente com o PrimeFaces.

5. Adicione no seu arquivo **web.xml** o **parâmetro de contexto** que informa ao PrimeFaces **nome do tema escolhido por você.** Exemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <welcome-file-list>
        <welcome-file>index.jsf</welcome-file>
    </welcome-file-list>

    <context-param>
            <param-name>primefaces.THEME</param-name>
            <param-value>flick</param-value>
    </context-param>

</web-app>
```

6. Pare o Glassfish. Limpe e construa a aplicação e implante-a no GlassFish novamente para verificar as novidades apresentadas.

Fase 7: Incrementando o restante da aplicação

1. Se desejar, aplique as evoluções apresentadas nas demais funcionalidades da sua aplicação.

2. Se desejar ainda, estude o componente **<p:dialog />** e faça com que a edição/criação de Fabricantes seja feita dentro de uma modal (uma sub-janela que bloqueia o uso da aplicação). Lembre-se! Além de fazer sumir e aparecer a modal ao editar/criar/salvar/cancelar, é preciso cuidar do Ajax para atualizar os componentes da tela adequadamente. Veja o efeito visual esperado: