

### Aula 06 – 10/jun.

#### RIA – Rich Internet Applications – Aplicações ricas para Internet.

Aplicações de Internet Rica são Aplicações Web que tem características e funcionalidades de softwares tradicionais do tipo Desktop. RIA típicos transferem todo o processamento da interface para o navegador da internet, porém mantém a maior parte dos dados (como por exemplo, o estado do programa, dados do banco) no servidor de aplicação.

Benefícios:

- Riqueza: É possível oferecer à interface do usuário características que não podem ser obtidos utilizando apenas o HTML disponível no navegador para aplicações Web padrão. Esta capacidade de poder incluir qualquer coisa no lado do cliente, incluindo o arraste e solte, utilizar uma barra para alterar dados, cálculos efetuados apenas pelo cliente e que não precisam ser enviados volta para o servidor, como por exemplo, uma calculadora básica de quatro operações.
- Melhor resposta: A interface é mais reativa a ações do usuário do que em aplicações Web padrão que necessitam de uma constante interação com um servidor remoto. Com isto os RIAs levam o usuário a ter a sensação de estarem utilizando uma aplicação desktop.
- Equilíbrio entre Cliente/Servidor: A carga de processamento entre o Cliente e Servidor torna-se mais equilibrada, visto que o servidor web não necessita realizar todo o processamento e enviar para o cliente, permitindo que o mesmo servidor possa lidar com mais sessões de clientes concomitantemente.
- Comunicação assíncrona: O *client engine* pode interagir com o servidor de forma *assíncrona* -- desta forma, uma ação na interface realizada pelo usuário como o clique em um botão ou link não necessita esperar por uma resposta do servidor, fazendo com que o usuário espere pelo processamento. Talvez o mais comum é que estas aplicações, a partir de uma solicitação, antecipe uma futura necessidade de alguns dados, e estes são carregados no cliente antes de o usuário solicite-os, de modo a acelerar uma posterior resposta. O site Google Maps utiliza esta técnica para, quando o usuário move o mapa, os segmentos adjacentes são carregados no cliente antes mesmo que o usuário mova a tela para estas áreas.
- Otimização da rede: O fluxo de dados na rede também pode ser significativamente reduzida, porque um *client engine* pode ter uma inteligência imbutida maior do que um navegador da Web padrão quando decidir quais os dados que precisam ser trocados com os servidores. Isto pode acelerar a solicitações individuais ou reduzir as respostas, porque muitos dos dados só são transferidos quando é realmente necessário, e a carga global da rede é reduzida. Entretanto, o uso destas técnicas podem neutralizar, ou

mesmo reverter o potencial desse benefício. Isto porque o código não pode prever exatamente o que cada usuário irá fazer em seguida, sendo comum que tais técnicas baixar dados extras, para muitos ou todos os clientes, cause um tráfego desnecessário.

As deficiências e restrições associadas aos RIAs são:

- **Sandbox:** Os RIAs são executado dentro de um *Sandbox*, que restringe o acesso a recursos do sistema. Se as configurações de acesso aos recursos estiverem incorretas, os RIAs podem falhar ou não funcionar corretamente.
- **Scripts desabilitados:** JavaScripts ou outros scripts são muitas vezes utilizados. Se o usuário desativar a execução de scripts em seu navegador, o RIA poderá não funcionar corretamente, na maior parte das vezes.
- **Velocidade de processamento no cliente:** Para que as aplicações do lado do cliente tenha independência de plataforma, o lado do cliente muitas vezes são escritos em linguagens interpretadas, como JavaScript, que provocam uma sensível redução de desempenho. Isto não é problema para linguagens como Java, que tem seu desempenho comparado a linguagens compiladas tradicionais, ou com o Flash, em que a maior parte das operações são executadas pelo código nativo do próprio Flash.
- **Tempo de carregamento da aplicação:** Embora as aplicações não necessitem de serem *instaladas*, toda a inteligência do lado cliente (ou *client engine*) deve ser baixada do servidor para o cliente. Se estiver utilizando um web cache, esta carga deve ser realizada pelo menos uma vez. Dependendo do tamanho ou do tipo de solicitação, o carregamento do script pode ser demasiado longo. Desenvolvedores RIA podem reduzir este impacto através de uma compactação dos scripts, e fazer um carregamento progressivo das páginas, conforme elas forem sendo necessárias.
- **Perda de Integridade:** Se a aplicação-base é X/HTML, surgem conflitos entre o objetivo de uma aplicação (que naturalmente deseja estar no controle de toda a aplicação) e os objetivos do X/HTML (que naturalmente não mantém o estado da aplicação). A interface DOM torna possível a criação de RIAs, mas ao fazê-lo torna impossível garantir o seu funcionamento de forma correta. Isto porque um cliente RIA pode modificar a estrutura básica, sobrescrevendo-a, o que leva a uma modificação do comportamento da aplicação, causando uma falha irreversível ou *crash* no lado do cliente. Eventualmente, este problema pode ser resolvido através de mecanismos que garantam uma aplicação do lado cliente com restrições e limitar o acesso do usuário para somente os recursos que façam parte do escopo da aplicação. (Programas que executam de forma nativa não tem este problema, porque, por definição, automaticamente possui todos os direitos de todos os recursos alocados).
- **Perda de visibilidade por Sites de Busca:** Sites de busca podem não ser capazes de indexar os textos de um RIA.

- Dependência de uma conexão com a Internet: Enquanto numa aplicação desktop ideal permite que os seus usuários fiquem *ocasionalmente conectados*, passando de uma rede para outra, hoje (em 2007), um típico RIA requer que a aplicação fique permanentemente conectada à rede.

(Fonte: Wikipédia)

### Frameworks RIA para JSF 2.0 – PrimeFaces

PrimeFaces é um dos melhores frameworks RIA para JSF. Todos os seus componentes, quando renderizados, são estruturas HTML, definidas esteticamente por CSS, com controle de eventos e troca de mensagens em JavaScript. Ou seja, encapsulam a complexidade de usar CSS com HTML e JavaScript para construir certos componentes. Além de prover o reuso dos mesmos.

Para utilizar o Primefaces é preciso baixar a biblioteca "[primefaces-2.2.1.jar](#)" e criar um projeto NetBeans de acordo com a aula 05.

Antes de começar, vamos criar um beans que utilizaremos a seguir:

```
@Named
@RequestScoped
public class TableBean{

    private String nome;
    private String descricao;
    private String telefone;
    private String cpf;
    private String observacao;
    private Date dataCadastro;
    private String senha;

    public void testar(){
        FacesContext context = FacesContext.getCurrentInstance();
        context.addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_INFO, "Aviso", "Testado com sucesso!"));
    }

    public String getCpf() {...}
    public void setCpf(String cpf) {...}
    public Date getDataCadastro() {...}
    public void setDataCadastro(Date dataCadastro) {...}
    public String getDescricao() {...}
    public void setDescricao(String descricao) {...}
    public String getNome() {...}
    public void setNome(String nome) {...}
    public String getObservacao() {...}
    public void setObservacao(String observacao) {...}
    public String getSenha() {...}
    public void setSenha(String senha) {...}
    public String getTelefone() {...}
    public void setTelefone(String telefone) {...}
}
```

O método testar será utilizado para criar mensagens de erro ou confirmação como veremos a seguir.

Não podemos esquecer-nos de adicionar a biblioteca na página.jsf e de colocar uma tag chamada view, que corrige alguns problemas de compatibilidade, principalmente no Google Chrome. Veja como deverá ficar sua página index.

```
<?xml version='1.0' encoding='UTF-8' ?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.prime.com.tr/ui"
      xmlns:f="http://java.sun.com/jsf/core">
  <f:view contentType="text/html">
    <h:head>
      <title>Facelet Title</title>
    </h:head>
    <h:body>

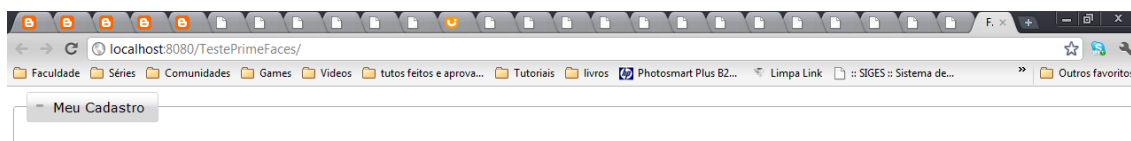
      </h:body>
    </f:view>
  </html>
```

### Botões e mensagens

Começaremos mostrando quatro componentes interessantes. O primeiro é o FieldSet, uma borda que circula formulário e deixa o visual interessante.

```
<p:fieldset legend="Meu Cadastro" toggleable="true">
</p:fieldset>
```

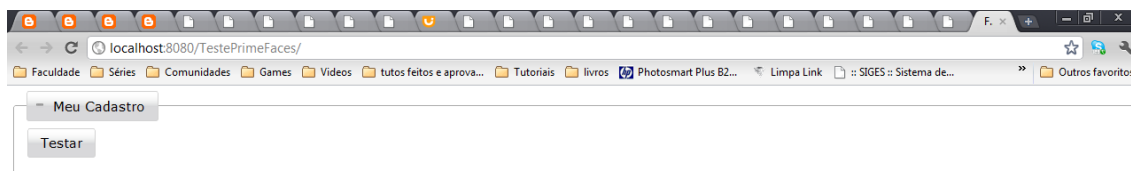
Legend é o título e toggleable indica se o usuário poderá esconder o conteúdo dentro do fieldset ou não. Veja o resultado:



O segundo componente é muito importante, trata-se de um botão commandButton.

```
<p:commandButton value="Testar" actionListener="#{testeBean.testar}"
/>
```

Veja o resultado:



O atributo actionListener indica ao botão qual método/comando no beans ele deverá fazer quando pressionado. Você pode utilizar action para se referir as páginas jsf. Mas não esqueça de sempre utilizar o atributo: ajax="false" quando o fizer, por exemplo:

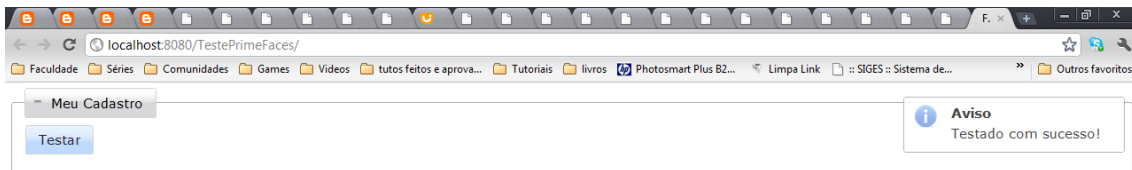
```
<p:commandButton value="Painéis" action="testepainels.jsf" ajax="false"/>
```

```
<p:commandButton value="Dock" action="testedock.jsf" ajax="false"/>
```

Agora vamos fazer o botão mostrar uma mensagem quando for clicado. O PrimeFaces oferece dois tipos de mensagens, embutidas na páginas, e uma que aparece numa pequena janelinha e desaparece com o tempo. Essa ultima é usada da seguinte forma:

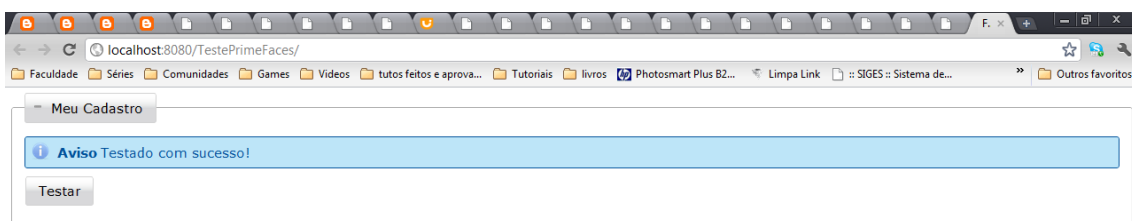
```
<p:growl id="aviso" showDetail="true" life="3000" />
```

O atributo life é para controlar, em milissegundos o tempo que a mensagem será mostrada. Veja o resultado:



Outra forma é usando:

```
<p:messages id="mensagens" showDetail="true" />
```



Para que o botão execute ação é preciso adicionar o seguinte atributo:

```
update="aviso,mensagens"
```

Ambos os métodos não precisam ser utilizados ao mesmo tempo, mas podem caso queira. Nossa página ficou assim no final:

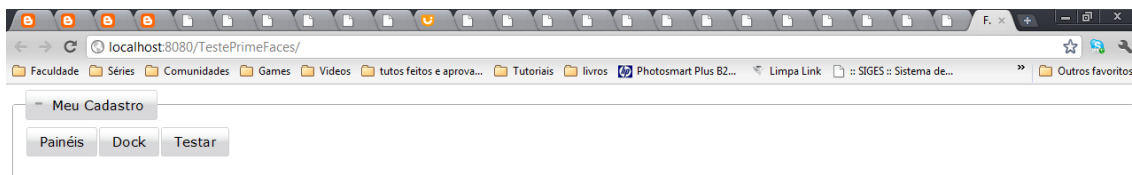
```
<?xml version='1.0' encoding='UTF-8' ?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.prime.com.tr/ui"
      xmlns:f="http://java.sun.com/jsf/core">
  <f:view contentType="text/html">
    <h:head>
      <title>Facelet Title</title>
    </h:head>
    <h:body>
      <p:fieldset legend="Meu Cadastro" toggleable="true">
        <h:form>

          <p:growl id="aviso" showDetail="true" life="3000" />
          <p:messages id="mensagens" showDetail="true" />

          <p:commandButton value="Painéis" action="testepainels.jsf" ajax="false"/>
          <p:commandButton value="Dock" action="testedock.jsf" ajax="false"/>

          <p:commandButton value="Testar" actionListener="#{tableBean.testar}"
                        update="aviso,mensagens" />

        </h:form>
      </p:fieldset>
    </h:body>
  </f:view>
</html>
```



Os botões Dock e Testar nos levarão para as páginas que serão criadas nos próximos tópicos.

### Formulários

Formulários estão presentes em praticamente todos os sites. São compostos com campos de texto, senhas, datas, entre outros. O exemplo a seguir mostra um exemplo de uso de formulários.

```
<p:fieldset legend="Meu Cadastro" toggleable="true">
  <h:form>
    <h:panelGrid columns="2">
      <h:outputText value="Nome:" />
      <p:inputText id="nome" value="#{tableBean.nome}" />

      <h:outputText value="Observações:" />
      <p:inputTextarea value="#{tableBean.observacao}" />

      <h:outputText value="Data cadastro:" />
      <p:calendar value="#{tableBean.dataCadastro}" />

      <h:outputText value="Telefone:" />
      <p:inputMask mask="(999)9999-9999" value="#{tableBean.telefone}" />
      <h:outputText value="CPF:" />
      <p:inputMask mask="999.999.999-99" value="#{tableBean.cpf}" />

      <h:outputText value="Descrição:" />
      <p:keyboard layout="qwertyBasic" value="#{tableBean.descricao}" />

      <h:outputText value="Senha:" />
      <p:keyboard
        password="true"
        keypadOnly="true"
        value="#{tableBean.descricao}" />
    </h:panelGrid>
    <p:separator style="width: 80%; height: 5px" />
  </h:form>
</p:fieldset>
```

```
<h:outputText value="Data cadastro:" />
<p:calendar value="#{tableBean.dataCadastro}" />
```

```
<h:outputText value="Telefone:" />
<p:inputMask mask="(999)9999-9999" value="#{tableBean.telefone}" />
```

```
<h:outputText value="CPF:" />
<p:inputMask mask="999.999.999-99" value="#{tableBean.cpf}" />
```

```
<h:outputText value="Descrição:" />
<p:keyboard layout="qwertyBasic" value="#{tableBean.descricao}" />
```

```
<h:outputText value="Senha:" />
<p:keyboard password="true" keypadOnly="true"
value="#{tableBean.descricao}" />
```

Senha:

Testar

1	2	3	Close
4	5	6	Clear
7	8	9	Back
0			

### Menus

Menus são muito utilizados nas aplicações Desktop e Web, servem para facilitar a navegação pelo site. Na mesma página, criamos um novo fieldset e colocamos o componente toolbar, ele cria uma barra de ferramentas que serve como um menu. Dentro dele estipulamos os toolbarGroup que são o conjunto de elementos (botões) dispostos na barra de ferramentas. Podemos utilizar o componente divider, para separar os botões. Veja o código e seu resultado:



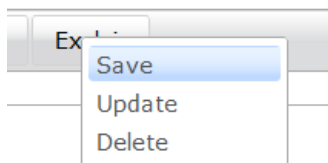
```
<p:fieldset legend="Menus" toggleable="true">
  <h:form>
    <p:toolbar>
      <p:toolbarGroup align="left">
        <p:commandButton value="Buscar"/>
        <p:divider/>
        <p:commandButton value="Novo"/>
        <p:commandButton value="Salvar"/>
        <p:commandButton value="Excluir"/>
      </p:toolbarGroup>
      <p:toolbarGroup align="right">
        <p:commandButton value="Sair"/>
      </p:toolbarGroup>
    </p:toolbar>
  </h:form>
</p:fieldset>
```

O componente contextMenu serve para criar menus que aparecem ao clicar no botão direito do mouse.

```
<h:form>
  <p:contextMenu>
    <p:menuitem value="Save" actionListener="..."/>
    <p:menuitem value="Update" actionListener="..."/>
    <p:menuitem value="Delete" actionListener="..."/>
  </p:contextMenu>
```

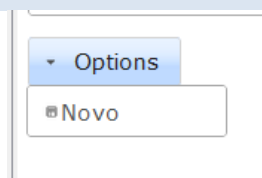


```
</h:form>
```



MenuBar é um componente que cria um botão-menu, ao ser clicado, ele estende um menu criado. No exemplo a seguir, percebe-se que podemos colocar uma pequena imagem no menu.

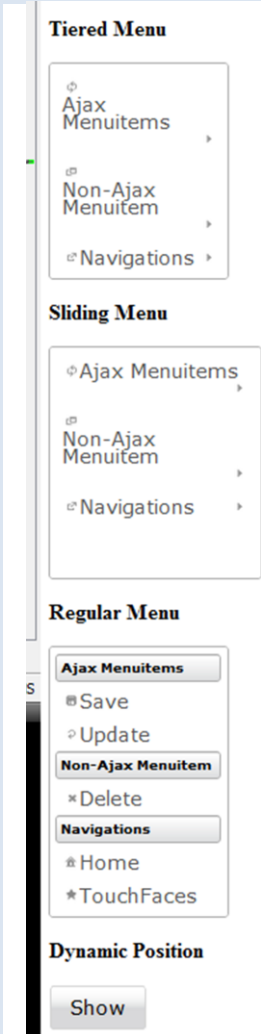
```
<h:form>
    <p:menuButton value="Options">
        <p:menuItem value="Novo" action="novoCadastro.jsf" ajax="false" icon="ui-icon
ui-icon-disk"></p:menuItem>
    </p:menuButton>
</h:form>
```



Temos outros menus oferecidos pelo prime-faces.

```
<h:form>
    <p:growl id="messages"/>
    <h3>Tiered Menu</h3>
    <p:menu type="tiered">
        <p:submenu label="Ajax Menuitems" icon="ui-icon ui-icon-refresh">
            <p:menuItem value="Save" actionListener="#{buttonBean.save}" update="messages"
icon="ui-icon ui-icon-disk" />
            <p:menuItem value="Update" actionListener="#{buttonBean.update}" update="messages"
icon="ui-icon ui-icon-arrowrefresh-1-w" />
        </p:submenu>
        <p:submenu label="Non-Ajax Menuitem" icon="ui-icon ui-icon-newwin">
            <p:menuItem value="Delete" actionListener="#{buttonBean.delete}" update="messages"
ajax="false" icon="ui-icon ui-icon-close" />
        </p:submenu>
        <p:submenu label="Navigations" icon="ui-icon ui-icon-extlink">
            <p:submenu label="Prime Links">
                <p:menuItem value="Prime" url="http://www.prime.com.tr" />
                <p:menuItem value="PrimeFaces" url="http://www.primefaces.org" />
            </p:submenu>
            <p:menuItem value="TouchFaces" url="#{request.contextPath}/touch" />
        </p:submenu>
    </p:menu>
```

```
<h3>Sliding Menu</h3>
<p:menu type="sliding">
  <p:submenu label="Ajax Menuitems" icon="ui-icon ui-icon-
refresh">
    <p:menuitem value="Save"
actionListener="#{buttonBean.save}" update="messages" icon="ui-icon ui-
icon-disk" />
    <p:menuitem value="Update"
actionListener="#{buttonBean.update}" update="messages" icon="ui-icon
ui-icon-arrowrefresh-1-w"/>
  </p:submenu>
  <p:submenu label="Non-Ajax Menuitem" icon="ui-icon ui-icon-
newwin">
    <p:menuitem value="Delete"
actionListener="#{buttonBean.delete}" update="messages" ajax="false"
icon="ui-icon ui-icon-close"/>
  </p:submenu>
  <p:submenu label="Navigations" icon="ui-icon ui-icon-extlink">
    <p:submenu label="Prime Links">
      <p:menuitem value="Prime" url="http://www.prime.com.tr"
/>
    <p:menuitem value="PrimeFaces"
url="http://www.primefaces.org" />
    </p:submenu>
    <p:menuitem value="TouchFaces"
url="#{request.contextPath}/touch" />
  </p:submenu>
</p:menu>
<h3>Regular Menu</h3>
<p:menu>
  <p:submenu label="Ajax Menuitems">
    <p:menuitem value="Save" actionListener="#{buttonBean.save}" update="messages"
icon="ui-icon ui-icon-disk"/>
    <p:menuitem value="Update" actionListener="#{buttonBean.update}" update="messages"
icon="ui-icon ui-icon-arrowrefresh-1-w"/>
  </p:submenu>
  <p:submenu label="Non-Ajax Menuitem">
    <p:menuitem value="Delete" actionListener="#{buttonBean.delete}" update="messages"
ajax="false" icon="ui-icon ui-icon-close"/>
  </p:submenu>
  <p:submenu label="Navigations">
    <p:menuitem value="Home" url="http://www.primefaces.org" icon="ui-icon ui-icon-
home"/>
    <p:menuitem value="TouchFaces" url="#{request.contextPath}/touch" icon="ui-icon ui-
icon-star"/>
  </p:submenu>
</p:menu>
```



The image displays three different menu styles: **Tiered Menu**, **Sliding Menu**, and **Regular Menu**. The **Tiered Menu** shows a hierarchical structure with 'Ajax Menuitems' and 'Non-Ajax Menuitem' as main categories, each with sub-items. The **Sliding Menu** is a vertical list of items that can slide out. The **Regular Menu** is a standard list of items with icons. Below these, there is a **Dynamic Position** section with a 'Show' button.

```
<h3>Dynamic Position</h3>
<p:commandButton id="dynaButton" value="Show" type="button"/>
<p:menu position="dynamic" trigger="dynaButton" my="left top" at="left bottom">
  <p:submenu label="Ajax Menuitems">
    <p:menuitem value="Save" actionListener="#{buttonBean.save}" update="messages"
icon="ui-icon ui-icon-disk"/>
    <p:menuitem value="Update" actionListener="#{buttonBean.update}" update="messages"
icon="ui-icon ui-icon-arrowrefresh-1-w"/>
  </p:submenu>
  <p:submenu label="Non-Ajax Menuitem">
    <p:menuitem value="Delete" actionListener="#{buttonBean.delete}" update="messages"
ajax="false" icon="ui-icon ui-icon-close"/>
  </p:submenu>
  <p:submenu label="Navigations">
    <p:menuitem value="Home" url="http://www.primefaces.org" icon="ui-icon ui-icon-
home"/>
    <p:menuitem value="TouchFaces" url="#{request.contextPath}/touch" icon="ui-icon ui-
icon-star"/>
  </p:submenu>
</p:menu>
</h:form>
```

Dock Menu - é um menu especial que imita o menu dos computadores Mac.



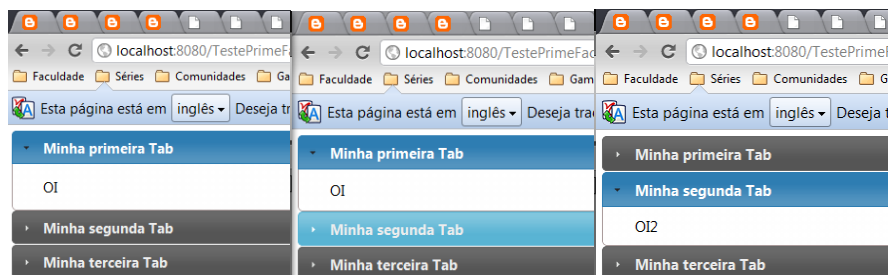
```
<p:dock position="top">
  <p:menuitem value="Home" icon="/images/dock/home.png" url="#" />
  <p:menuitem value="Music" icon="/images/dock/music.png" url="#" />
  <p:menuitem value="Video" icon="/images/dock/video.png" url="#" />
  <p:menuitem value="Email" icon="/images/dock/email.png" url="#" />
  <p:menuitem value="Portfolio" icon="/images/dock/portfolio.png" url="#" />
  <p:menuitem value="Link" icon="/images/dock/link.png" url="#" />
  <p:menuitem value="RSS" icon="/images/dock/rss.png" url="#" />
  <p:menuitem value="History" icon="/images/dock/history.png" url="#" />
</p:dock>
```

### Painéis e Efeitos

Nessa parte iremos ver sobre painéis e efeitos de transição. Para isso criaremos uma nova página chamada testepainels. Onde terá nossos painéis.

AccordionPanel é um painel especial que contem uma lista de Titulos, ao clicar no titulo, abre-se o painel com um conteúdo. Podendo ter qualquer coisa dentro.

```
<p:accordionPanel>
  <p:tab title="Minha primeira Tab">
    <h:outputText value="OI"/>
  </p:tab>
  <p:tab title="Minha segunda Tab">
    <h:outputText value="OI2"/>
  </p:tab>
  <p:tab title="Minha terceira Tab">
    <h:outputText value="OI3"/>
  </p:tab>
</p:accordionPanel>
```



Efeitos em painéis – os painéis podem ter efeitos diversos. Use o código a seguir e confira.



```
<p:fieldset legend="Meu teste!!!">
  <h:panelGrid columns="4">

    <p:panel header="Blind">
      <h:outputText value="click" />
      <p:effect type="blind" event="click">
        <f:param name="direction" value="horizontal" />
      </p:effect>
    </p:panel>
```

```
<p:panel header="Clip">
  <h:outputText value="click" />
  <p:effect type="clip" event="click" />
</p:panel>

<p:panel header="Drop">
  <h:outputText value="click" />
  <p:effect type="drop" event="click" />
</p:panel>

<p:panel header="Explode">
  <h:outputText value="click" />
  <p:effect type="explode" event="click" />
</p:panel>

<p:panel header="Fold">
  <h:outputText value="doubleclick" />
  <p:effect type="fold" event="dblclick" />
</p:panel>

<p:panel header="Puff">
  <h:outputText value="doubleclick" />
  <p:effect type="puff" event="dblclick" />
</p:panel>

<p:panel header="Slide">
  <h:outputText value="doubleclick" />
  <p:effect type="slide" event="dblclick" />
</p:panel>

<p:panel header="Scale">
  <h:outputText value="doubleclick" />
  <p:effect type="scale" event="dblclick">
    <f:param name="percent" value="90" />
  </p:effect>
</p:panel>

<p:panel header="Bounce">
  <h:outputText value="click" />
  <p:effect type="bounce" event="click" />
</p:panel>
```

```
<p:panel header="Pulsate">
  <h:outputText value="click" />
  <p:effect type="pulsate" event="click" />
</p:panel>

<p:panel header="Shake">
  <h:outputText value="click" />
  <p:effect type="shake" event="click" />
</p:panel>

<p:panel header="Size">
  <h:outputText value="click" />
  <p:effect type="size" event="click">
    <f:param name="to" value="{width: 200,height: 60}" />
  </p:effect>
</p:panel>

</h:panelGrid>
</p:fieldset>
```

### Tabelas

Vimos no decorrer do curso que podemos criar tabelas para mostrar dados do banco. O primeFaces cria mais rapidamente essas tabelas, e com controles e funções diversas. Vamos modificar nosso beans para criar uma lista de animais e criar uma tabela com base nela.

```
private List<String> animais;

public List<String> getAnimais() {
  animais = new ArrayList<String>();
  animais.add("Girafa");
  animais.add("Pato");
  animais.add("Leopardo");
  animais.add("Elefante");
  animais.add("Zebra");
  return animais;
}

public void setAnimais(List<String> animais) {
  this.animais = animais;
}
```

Iremos criar um componente dataTable, que pode, inclusive, ser página e muitas outras funções.

```
<h:form>
  <p:dataTable value="#{tableBean.animais}" var="a">
```

```
<p:column headerText="Nome dos animais">
    <h:outputText value="#{a}" />
</p:column>
<p:column headerText="Nome dos animais 2">
    <h:outputText value="#{a}" />
</p:column>
</p:dataTable>
</h:form>
```

Nome dos animais	Nome dos animais 2
Girafa	Girafa
Pato	Pato
Leopardo	Leopardo
Elefante	Elefante
Zebra	Zebra
Girafa	Girafa
Pato	Pato
Leopardo	Leopardo
Elefante	Elefante
Zebra	Zebra
Girafa	Girafa
Pato	Pato
Leopardo	Leopardo
Elefante	Elefante
Zebra	Zebra

Adicionando uma paginação com até 4 linhas:

```
paginator="true" rows="4"
```

Temos:

Nome dos animais	Nome dos animais 2
Girafa	Girafa
Pato	Pato
Leopardo	Leopardo
Elefante	Elefante

## Temas

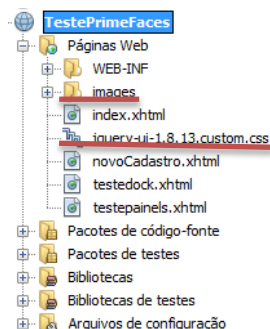
O primeFaces tem diversos temas prontos, e a possibilidade de personalizar e criar novos temas. O site <http://jqueryui.com/themeroller/> oferece todas as ferramentas necessárias para criação de temas para primeFaces basta utilizar o menu do lado esquerdo.



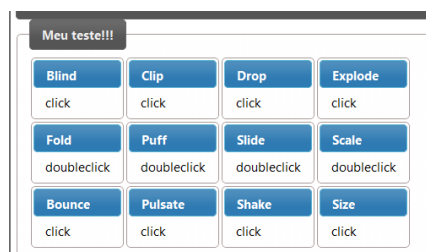
# Mini Curso para Formação Desenvolvedor Web 2011

## - Módulo Java Web

Para usar os temas criados, você deverá baixar o tema pronto do site e colocar a pasta de imagens e o arquivo .css da pasta gerada em seu projeto.



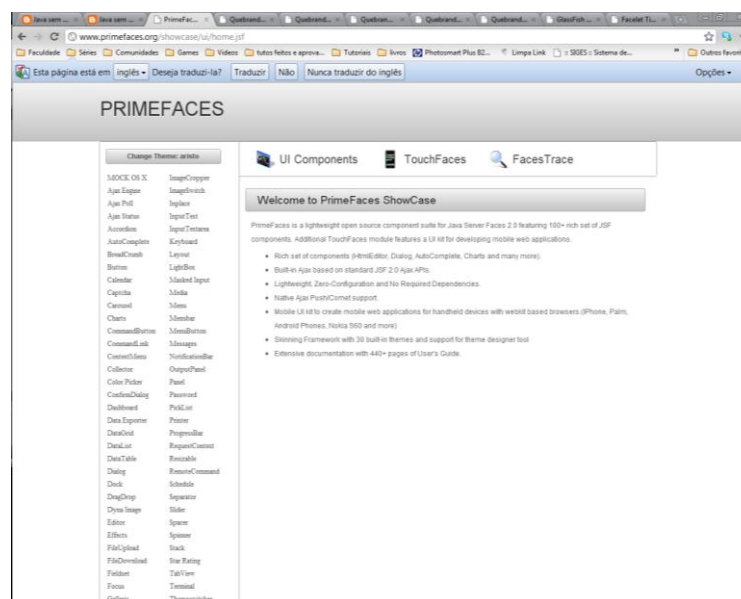
Veja um exemplo que eu crie (a pagina de painéis usada nesse tutorial):



### Considerações finais

A lista de componentes do PrimeFaces é enorme, tem muitos componentes, variações e atributos. Explore a biblioteca visual que esta a disposição no site:

<http://www.primefaces.org/showcase/ui/home.jsf>



### Exercícios Complementares de Fixação

1) Refaça os exercícios feitos em aula.



- 2) Refaça os exercícios complementares das outras aulas aplicando componentes primeFaces. Crie seus temas e desafie a imaginação.

### **Exercícios de Pesquisa:**

- 1) Pesquise sobre RIA e sua importância.
- 2) Pesquise sobre outros frameworks RIA.
- 3) Pesquise sobre ferramentas RIA que podem ser utilizadas com Java como o Flex da adobe.