



## Programação para Servidores-II

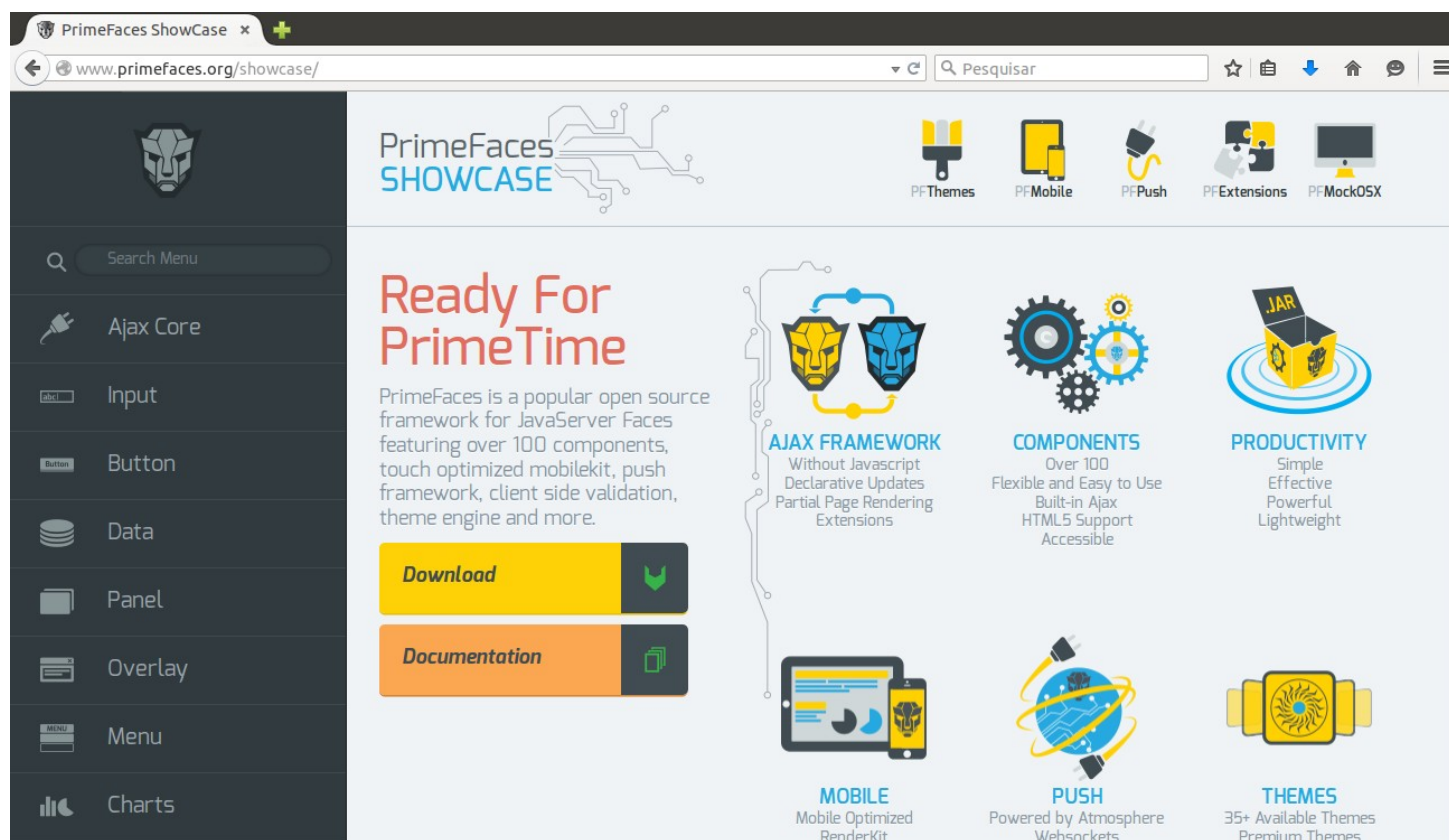
# PrimeFaces

Neste tópico abordaremos o PrimeFaces, um framework de composição tela.

*Prof. Ciro Cirne Trindade*

- PrimeFaces é um biblioteca de componentes JSF open source com várias extensões, entre elas:
  - Mais de 100 componentes de interface
    - HtmlEditor, Dialog, AutoComplete, Charts, etc.
  - Suporte a Ajax baseado no padrão JSF 2.2 Ajax APIs
- Versão otimizada para dispositivos móveis (PrimeMobile) como iPhone, Android, etc.

- <http://primefaces.org>
- Showcase muito interessante



The screenshot shows the PrimeFaces Showcase website. The browser address bar displays [www.primefaces.org/showcase/](http://www.primefaces.org/showcase/). The website features a dark sidebar on the left with a search menu and a list of categories: Ajax Core, Input, Button, Data, Panel, Overlay, Menu, and Charts. The main content area is titled "PrimeFaces SHOWCASE" and includes a navigation bar with links to PFThemes, PFMobile, PFPush, PFExtensions, and PFMockOSX. The central section is titled "Ready For PrimeTime" and describes PrimeFaces as a popular open source framework for JavaServer Faces. It lists features such as over 100 components, touch optimized mobilekit, push framework, client side validation, theme engine, and more. Below this, there are two buttons: "Download" and "Documentation". The bottom section highlights six key features: AJAX FRAMEWORK (Without Javascript, Declarative Updates, Partial Page Rendering, Extensions), COMPONENTS (Over 100, Flexible and Easy to Use, Built-in Ajax, HTML5 Support, Accessible), PRODUCTIVITY (Simple, Effective, Powerful, Lightweight), MOBILE (Mobile Optimized, RenderKit), PUSH (Powered by Atmosphere, Websockets), and THEMES (35+ Available Themes, Premium Themes).

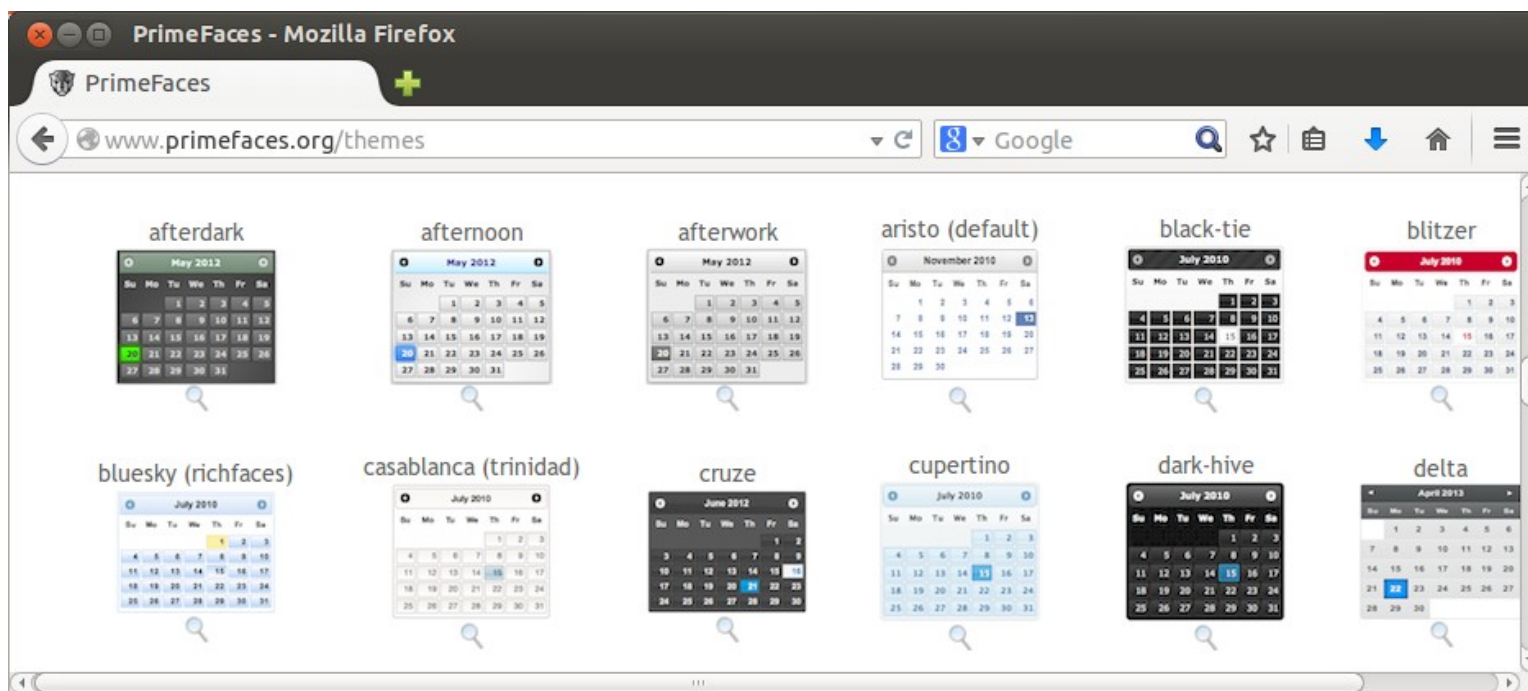
- É necessário uma única biblioteca
  - primefaces-{versão}.jar
  - Versão atual 5.2

- Referenciando a tag library

```
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:h="http://xmlns.jcp.org/jsf/html"  
      xmlns:p="http://primefaces.org/ui">
```

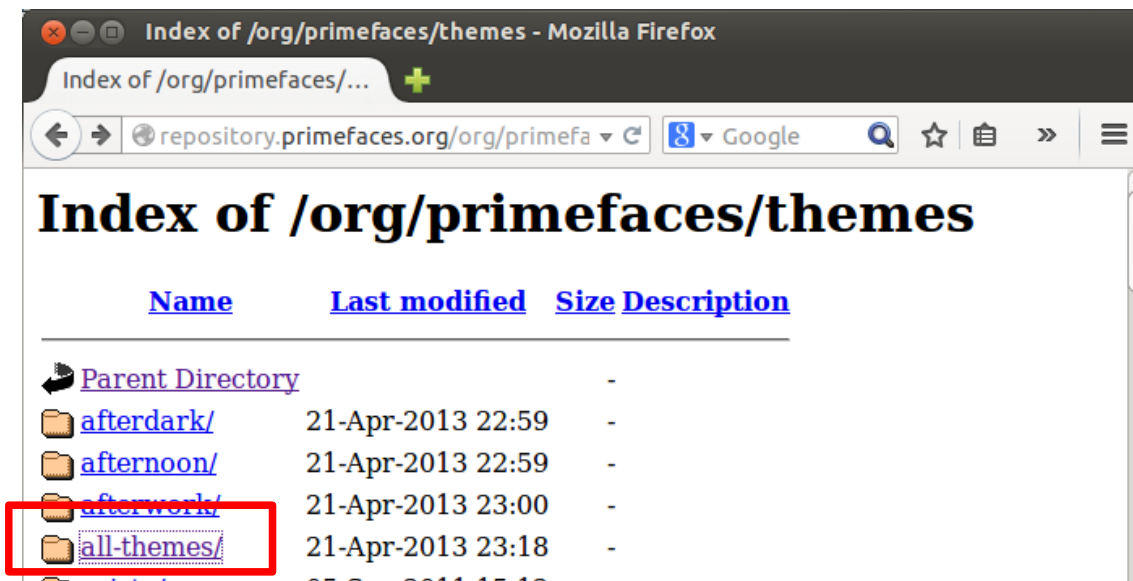
# Temas (1/4)

- O PrimeFaces trabalha com um mecanismo de temas
- [primefaces.org/themes](http://primefaces.org/themes)



# Temas (2/4)

- Para baixar o .jar dos temas, clique no link [PrimeFaces Repository](http://repository.primefaces.org/org/primefaces/themes/)  
(<http://repository.primefaces.org/org/primefaces/themes/>)
- Para baixar todos os temas, acesse o diretório all-themes



- O tema default do PrimeFaces é o *aristo*
- Para mudar o tema, basta acrescentar essa informação do web.xml
- Por exemplo, para usar o tema *bootstrap*

```
<context-param>  
    <param-name>primefaces.THEME</param-name>  
    <param-value>bootstrap</param-value>  
</context-param>
```

- Também é possível mudar o tema em tempo de execução através da tag `<p:themeSwitcher>`



# DataTable

- `<p:dataTable>` é uma versão melhorada de `<h:dataTable>` do JSF
- Provê soluções prontas para paginação, ordenação, filtro, seleção, etc.
- É possível definir o cabeçalho de colunas através do atributo `headerText` da tag `<p:column>`

# Usando um DataTable simples para listar as contas

```
<p:dataTable
    value="#{contaBackbean.contas}"
    var="c">
    <p:column headerText="Id">
        #{c.id}
    </p:column>
    <p:column headerText="Titular">
        #{c.titular}
    </p:column>
    <p:column headerText="Banco">
        <h:outputText value="#{c.banco}"/>
    </p:column>
    <p:column headerText="Agência">
        <h:outputText value="#{c.agencia}"/>
    </p:column>
    <p:column headerText="Número">
        <h:outputText value="#{c.numero}"/>
    </p:column>
</p:dataTable>
```

# Paginação (1/2)

- `<p:dataTable>` possuir um paginador baseado em Ajax que é habilitado definindo o valor do atributo **paginator** como `true`
- O `<p:dataTable>` deve estar dentro de um `<h:form>`

`<h:form>`

```
<p:dataTable  
    value="#{contaBackbean.contas}"  
    var="c" rows="5" paginator="true">  
    // colunas  
</p:dataTable>
```

Define o número  
linhas por página

## ■ Outros atributos de paginação:

- `paginatorPosition="both|top|bottom"`
- `PaginatorTemplate="{CurrentPageReport}{FirstPageLink} {PreviousPageLink}{PageLinks} {NextPageLink} {LastPageLink}{RowsPerPageDropdown}"`
- `rowPerPageTemplate="<int>, <int>, ..."`

## ■ Exemplo:

```
<p:dataTable value="#{contaBackBean.contas}" var="c"
paginator="true" rows="3" rowPerPageTemplate="3,5,10"
paginatorPosition="top" paginatorTemplate=
"{FirstPageLink} {PreviousPageLink}
{CurrentPageReport} {NextPageLink}
{LastPageLink} {RowsPerPageDropdown}">
```

# Ordenação

- O atributo **sortBy** de uma coluna habilita a ordenação via Ajax por uma coluna em particular

```
<p:column headerText="Banco"  
           sortBy="#{c.banco}">
```

- O atributo `sortMode="multiple"` do `<p:dataTable>` habilita a ordenação por mais de uma coluna (default é `single`)

- De forma semelhante a ordenação, o filtro baseado em Ajax é habilitado ao nível de coluna definindo o atributo `filterBy` e provendo uma lista para manter os registros filtrados
- Esta lista deve ser definida através do atributo `filteredValue` do `<p:dataTable>`

## ■ Exemplo:

```
<p:dataTable
    value="#{contaBackbean.contas}"
    var="c" paginator="true" rows="5"
    filteredValue="#{contaBackbean.filtradas}" ...>
    ...
    <p:column headerText="Titular"
        filterBy="#{c.titular}">
        ...
    </p:column>
```

```
@ManagedBean
@SessionScoped
public class ContaBackbean implements Serializable {
    private List<Conta> filtradas;
```

- Outros atributos de filtros:
  - `filterMatchMode="startsWith|endsWith|contains|exact..."`
  - `filterFunction="#{bean.método}"`
    - `filterFunction` deve ser um método com 3 parametros: valor da coluna, value do filtro e locale
    - O valor de retorno é booleano, `true` aceita o valor e `false` rejeita-o



## ■ Exemplo de método de filtro

```
public boolean consultarPorNome(Object value,
                                Object filter, Locale locale) {
    String filterText = (filter == null) ? null :
                        filter.toString().trim();
    String valueText = (value == null) ? null :
                        value.toString();
    if(filterText == null || filterText.equals("")) {
        return true;
    }
    if(valueText == null) {
        return false;
    }
    return valueText.matches("(?i).*" + filterText
                              + ".*");
}
```

- `p:cellEditor` é usado para definir a edição de célula para uma coluna
- Há dois tipos de edição, linha e célula
- O modo *default* é a edição de linha e é usada ao adicionar um componente `p:rowEditor` a uma coluna do `p:dataTable`
- Para definir a edição de célula, é necessário definir o valor do atributo `editMode` do `p:dataTable` como `"cell"`

- A tag `p:cellEditor` deve possuir duas tags `f:facet` no seu corpo, uma com o valor do atributo `name` igual a `"output"` e a outra com o valor do atributo `name` igual a `"input"`
  - As tags `f:facet` devem possuir componentes de saída e entrada
- Para habilitar a edição é necessário definir o atributo `editable` do `p:dataTable` como `"true"`

## ■ Exemplo:

```
<p:dataTable
    value="#{contaBackbean.contas}" var="c"
    ...
    editable="true">
    ...
    <p:column headerText="Titular"
        filterBy="#{c.titular}"
        <p:cellEditor>
            <f:facet name="output">
                <h:outputText value="#{c.titular}"/>
            </f:facet>
            <f:facet name="input">
                <h:inputText value="#{c.titular}"/>
            </f:facet>
        </p:cellEditor>
    </p:column>
    <p:column>
        <p:rowEditor />
    </p:column>
    ...
```

- Definindo as ações no *managed bean* através da tag `p:ajax` para responder aos eventos `rowEdit` e `rowEditCancel` gerados pelo `p:rowEdit`
- Para a edição de célula, devemos responder ao evento `cellEdit`

## ■ Exemplo:

```
<h:form id="form">
  <p:growl id="messages" showDetail="true"/>
  ...
  <p:dataTable
    value="#{contaBackbean.contas}"
    var="c" ... editable="true">
    <p:ajax event="rowEdit"
      listener="#{contaBackbean.onEdit}"
      update=":form:messages" />
    <p:ajax event="rowEditCancel"
      listener="#{contaBackbean.onCancel}"
      update=":form:messages" />
    ...
  </p:dataTable>
</h:form>
```

Componente para exibir mensagens semelhante a `h:messages`, mas baseado nas notificações do Mac

## ■ Métodos no *managed bean*

```
public void onEdit(RowEditEvent event) {  
    Conta c =  
        (Conta) event.getObject();  
    DAO<Conta> dao =  
        new DAO(Conta.class);  
    dao.alterar(c);  
    FacesMessage msg = new FacesMessage(  
        "Conta atualizada",  
        c.getTitular());  
    FacesContext.getCurrentInstance().  
        addMessage(null, msg);  
}  
  
public void onCancel(RowEditEvent event) {  
    ...  
}
```

# Remoção (1/2)

```
<p:column headerText="Remover">  
    <p:commandButton title="Remover"  
        update="@form" ajax="true"  
        icon="ui-icon-trash"  
        action="#{contaBackbean.onDelete(c)}">  
    </p:commandButton>  
</p:column>
```



# Remoção (2/2)

```
public void onDelete(Conta c) {  
    DAO<Conta> dao = new DAO(Conta.class);  
    dao.excluir(c.getId());  
    contas.remove(c);  
    FacesMessage msg = new FacesMessage(  
        "Conta removida", c.getTitular());  
  
    FacesContext.getCurrentInstance().  
        addMessage(null, msg);  
}
```

# Referências

- ÇIVICI, Çagatay. *PrimeFaces User's Guide 5.2*. Disponível em:  
<[http://www.primefaces.org/docs/guide/primefaces\\_user\\_guide\\_5\\_2.pdf](http://www.primefaces.org/docs/guide/primefaces_user_guide_5_2.pdf)>.