

# JSF e Primefaces

Namom Alves Alencar



# Introdução

- Durante muitos anos, os usuários se habituaram com aplicações Desktop. Este tipo de aplicação é instalada no computador local e acessa diretamente um banco de dados ou gerenciador de arquivos. As tecnologias típicas para criar uma aplicação Desktop são Delphi, VB (Visual Basic) ou, no mundo Java, Swing(não vimos, pois está em desuso).

# Introdução

- Para o desenvolvedor, a aplicação Desktop é construída com uma série de componentes que a plataforma de desenvolvimento oferece para cada sistema operacional. Esses componentes ricos e muitas vezes sofisticados estão associados a eventos ou procedimentos que executam lógicas de negócio.

# Introdução

- Problemas de validação de dados são indicados na própria tela sem que qualquer informação do formulário seja perdida. De uma forma natural, esses componentes lembram-se dos dados do usuário, inclusive entre telas e ações diferentes.
- Nesse tipo de desenvolvimento são utilizados diversos componentes ricos, como por exemplo, calendários, menus diversos ou componentes drag and drop (arrastar e soltar). Eles ficam associados a eventos, ou ações, e guardam automaticamente seu estado, já que mantêm os valores digitados pelo usuário.

# Introdução

## Aplicação Desktop?

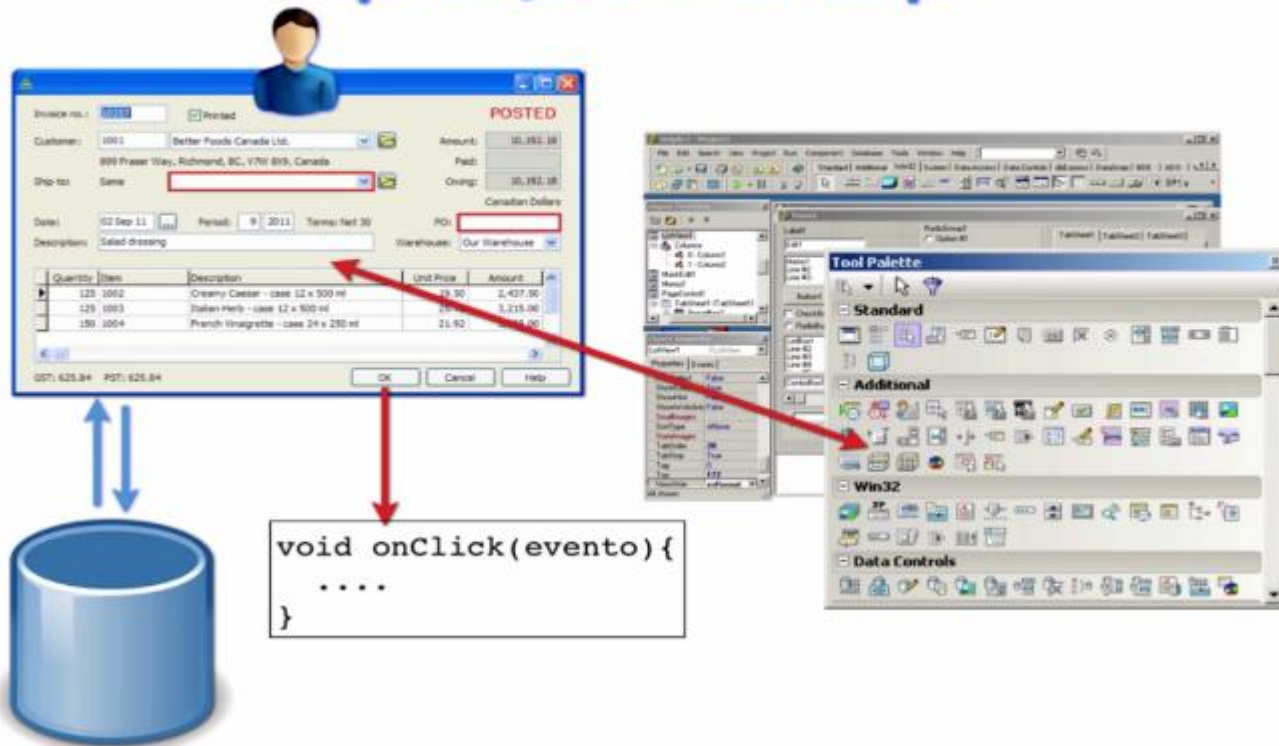


# Introdução

- Esses componentes não estão, contudo, associados exclusivamente ao desenvolvimento de aplicações Desktop. Podemos criar a mesma sensação confortável para o cliente em uma aplicação web, também usando componentes ricos e reaproveitáveis.

# Introdução

## Aplicação Desktop?



# O desenvolvimento Web e o protocolo HTTP

- Para resolver problemas como esse, surgiram as aplicações baseadas na web. Nessa abordagem há um servidor central onde a aplicação é executada e processada e todos os usuários podem acessá-la através de um cliente simples e do protocolo HTTP.
- Um navegador web, como Firefox ou Chrome, que fará o papel da aplicação cliente, interpretando HTML, CSS e JavaScript -- que são as tecnologias que ele entende.



# O desenvolvimento Web e o protocolo HTTP

- Enquanto o usuário usa o sistema, o navegador envia requisições (requests) para o lado do servidor (server side), que responde para o computador do cliente (client side). Em nenhum momento a aplicação está salva no cliente: todas as regras da aplicação estão no lado do servidor. Por isso, essa abordagem também foi chamada de cliente magro (thin client).

# O desenvolvimento Web e o protocolo HTTP



# O desenvolvimento Web e o protocolo HTTP

- Isso facilita bastante a manutenção e a gerenciabilidade, pois temos um lugar central e acessível onde a aplicação é executada. Contudo, note que será preciso conhecer HTML, CSS e JavaScript, para fazer a interface com o usuário, e o protocolo HTTP para entender a comunicação pela web. E, mais importante ainda, não há mais eventos, mas sim um modelo bem diferente orientado a requisições e respostas. Toda essa base precisará ser conhecida pelo desenvolvedor.

# O desenvolvimento Web e o protocolo HTTP

- Comparando as duas abordagens, podemos ver vantagens e desvantagens em ambas. No lado da aplicação puramente Desktop, temos um estilo de desenvolvimento orientado a eventos, usando componentes ricos, porém com problemas de manutenção e gerenciamento. Do outro lado, as aplicações web são mais fáceis de gerenciar e manter, mas precisamos lidar com HTML, conhecer o protocolo HTTP e seguir o modelo requisição/resposta.

# Características do JSF

- JSF é uma tecnologia que nos permite criar aplicações Java para Web utilizando componentes visuais pré-prontos, de forma que o desenvolvedor não se preocupe com Javascript e HTML. Basta adicionarmos os componentes (calendários, tabelas, formulários) e eles serão renderizados e exibidos em formato html.

# Características do JSF

- **Guarda o estado dos componentes**
- Além disso o estado dos componentes é sempre guardado automaticamente (como veremos mais à frente), criando a característica Stateful. Isso nos permite, por exemplo, criar formulários de várias páginas e navegar nos vários passos dele com o estado das telas sendo mantidos.

# Características do JSF

- **Separa as camadas**
- Outra característica marcante na arquitetura do JSF é a separação que fazemos entre as camadas de apresentação e de aplicação. Pensando no modelo MVC, o JSF possui uma camada de visualização bem separada do conjunto de classes de modelo.

# Características do JSF

- **Especificação: várias implementações**
- O JSF ainda tem a vantagem de ser uma especificação do Java EE, isto é, todo servidor de aplicações Java tem que vir com uma implementação dela e há diversas outras disponíveis.

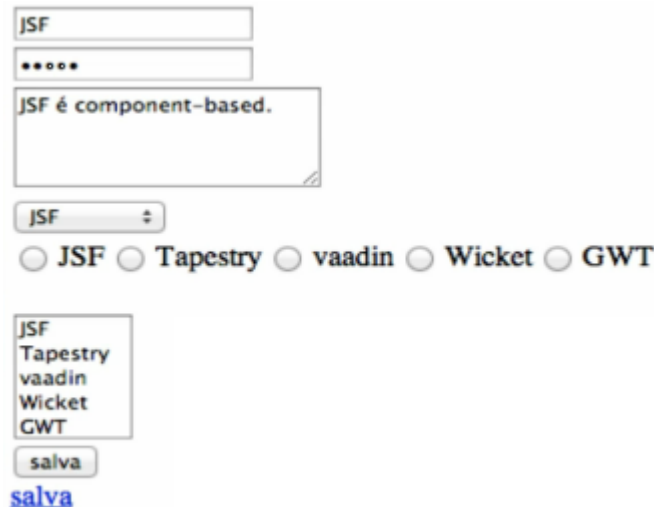


# Características do JSF

- A implementação mais famosa do JSF e também a implementação de referência, é a Oracle Mojarra disponível em <http://javaserverfaces.java.net/>. Outra implementação famosa é a MyFaces da Apache Software Foundation em <http://myfaces.apache.org/>.

# Primeiros passos com JSF

- Nosso projeto utilizará a implementação Mojarra do JSF. Ela já define o modelo de desenvolvimento e oferece alguns componentes bem básicos. Nada além de inputs, botões e ComboBoxes simples.



A screenshot of a web form demonstrating basic JSF components. The form includes a text input field with the value 'JSF', a password input field with five dots, a text area containing the text 'JSF é component-based.', a dropdown menu currently showing 'JSF', a group of radio buttons for selecting a framework (JSF, Tapestry, vaadin, Wicket, GWT), a list box showing the same five options, a 'salva' button, and a blue hyperlink labeled 'salva'.

# Primeiros passos com JSF

- Não há componentes sofisticados dentro da especificação e isso é proposital: uma especificação tem que ser estável e as possibilidades das interfaces com o usuário crescem muito rapidamente. A especificação trata do que é fundamental, mas outros projetos suprem o que falta.
- Para atender a demanda dos desenvolvedores por componentes mais sofisticados, há várias extensões do JSF que seguem o mesmo ciclo e modelo da especificação. Exemplos dessas bibliotecas são PrimeFaces, RichFaces e IceFaces. Todas elas definem componentes JSF que vão muito além da especificação.

# Primeiros passos com JSF



- Cada biblioteca oferece *ShowCases* na web para mostrar seus componentes e suas funcionalidades. Você pode ver o *showcase* do **PrimeFaces** no endereço <http://www.primefaces.org>.

# Primeiros passos com JSF

- Na sua *demo online*, podemos ver uma lista de componentes disponíveis, como inputs, painéis, botões diversos, menus, gráficos e componentes *drag & drop*, que vão muito além das especificações, ainda mantendo a facilidade de uso (vamos entrar no site).
- Para a definição da interface do projeto usaremos Oracle Mojarra com PrimeFaces, uma combinação muito comum no mercado.

# Preparação do ambiente

- Nossa aplicação precisa de uma interface web. Para isso vamos preparar uma aplicação web comum que roda dentro de um Servlet Container. Qualquer implementação de servlet container seria válida e, no curso, usaremos o Apache Tomcat 7+. Existem outros servidores como o Jetty, Jboss e o WebLogic

# Configuração do controlador do JSF

- O JSF segue o padrão arquitetural MVC (Model-View-Controller) e faz o papel do Controller da aplicação. Para começar a usá-lo, é preciso configurar a servlet do JSF no web.xml da aplicação. Esse Servlet é responsável por receber as requisições e delegá-las ao JSF. Para configurá-lo basta adicionar as seguintes configurações no web.xml (Criado automaticamente)

# Faces-config: o arquivo de configuração do mundo JSF

- Além disso, há um segundo XML que é o arquivo de configuração relacionado com o mundo JSF, o faces-config.xml.
- Como o JSF na versão dois encoraja o uso de anotações em vez de configurações no XML, este arquivo torna-se pouco usado. Ele era muito mais importante na primeira versão do JSF.



# Exercicio

- 1 – Instalar o TomCat 8
  - Acesse o site do tomcat
  - Baixe o arquivo zip
  - Crie um projeto do tipo server
  - Associe a pasta ao Tomcat 8
- 2 – Criar um projeto JSF 2.2
  - Vamos utilizar o Mojarra
  - Iremos baixar automaticamente as lib, pelo eclipse.
- 3 – Instalar o Jboss Tools

# A primeira página com JSF

- Como configuramos, na criação do projeto, que o JSF será responsável por responder às requisições com extensão .xhtml. Dessa forma, trabalharemos com arquivos xhtml no restante do curso.
- Vale lembrar uma diferença fundamental entre as duas formas de desenvolvimento para a web. A abordagem action based MVC, focam seu funcionamento nas classes que contêm as lógicas. A view é meramente uma camada de apresentação do que foi processado no modelo.

# A primeira página com JSF

- Enquanto isso, o pensamento component based adotado pelo JSF leva a view como a peça mais importante -- é a partir das necessidades apontadas pelos componentes da view que o modelo é chamado e populado com dados.
- As tags que representam os componentes do JSF estão em duas taglibs principais (bibliotecas de tags): a core e a html.

# A primeira página com JSF

- A taglib html contém os componentes necessários para montarmos nossa tela gerando o HTML adequado. Já a core possui diversos componentes não visuais, como tratadores de eventos ou validadores. Por ora, usaremos apenas os componentes da h:html

# A primeira página com JSF

- Importando as tags em nossa página
- Diferente da forma importação de taglibs em JSPs que vimos no curso de Java para a web (FJ-21), para importar as tags no JSF basta declararmos seus namespaces no arquivo .xhtml. Dessa forma, teremos:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
    <!-- aqui usaremos as tags do JSF -->
</html>
```

# Definindo a interface da aplicação

- Como qualquer outro aprendizado de tecnologia, vamos começar a explorar o JSF criando nossa primeira tela com uma mensagem de boas vindas para o usuário.
- Como todo arquivo HTML, todo o cabeçalho deve estar dentro da tag head e o que será renderizado no navegador deve ficar dentro da tag body. Uma página padrão para nós seria algo como:

```
<html ...>
  <head>
    <!-- cabeçalho aqui -->
  </head>
  <body>
    <!-- informações a serem mostradas -->
  </body>
</html>
```

# Definindo a interface da aplicação

- Quando estamos lidando com o JSF, no entanto, precisamos nos lembrar de utilizar preferencialmente as tags do próprio framework, já que, à medida que utilizarmos componentes mais avançados, o JSF precisará gerenciar os próprios body e head para, por exemplo, adicionar CSS e javascript que um componente requisitar.

# Definindo a interface da aplicação

- Assim, usando JSF preferiremos utilizar as tags estruturais do HTML que vêm da taglib `http://java.sun.com/jsf/html`, nosso html vai ficar mais parecido com esse:

```
<html ...>  
  <h:head>  
    <!-- cabeçalho aqui -->  
  </h:head>  
  <h:body>  
    <!-- informações a serem mostradas -->  
  </h:body>  
</html>
```



# Mostrando informações com h:outputText

- Como queremos mostrar uma saudação para o visitante da nossa página, podemos usar a tag h:outputText. É através do seu atributo value que definimos o texto que será apresentado na página.
- Juntando tudo, nosso primeiro exemplo é uma tela simples com um texto:

`<h:outputText value = "Olá JSF!" />`

# Managed Beans

- O `h:outputText` é uma tag com um propósito aparentemente muito bobo e, no exemplo acima, é exatamente equivalente a simplesmente escrevermos "Olá JSF!" diretamente. E, de fato, para textos fixos, não há problema em escrevê-lo diretamente!
- Contudo, se um pedaço de texto tiver que interagir com o modelo, uma lógica ou mesmo com outros componentes visuais, será necessário que ele também esteja guardado em um componente.

# Managed Beans

- Estes, são apenas classezinhas simples que com as quais o JSF consegue interagir através do acesso a seus métodos. Nada mais são do que POJOs anotados com @ManagedBean.
- *POJO (Plain Old Java Object)*
- *POJO é um termo criado por Martin Fowler, Rebecca Parsons e Josh Mackenzie que serve para definir um objeto simples. Segundo eles, o termo foi criado pois ninguém usaria objetos simples nos seus projetos pois não existia um nome extravagante para ele.*

# Managed Beans

- Se quisermos, por exemplo, mostrar quando foi o acesso do usuário a essa página, podemos criar a seguinte classe:

```
@ManagedBean
```

```
public class OlaMundoBean {
```

```
    public String getHorario() {
```

```
        SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss");
```

```
        return "Atualizado em " + sdf.format(new Date());
```

```
    }
```

```
}
```

# Managed Beans

- E, bem semelhantemente à forma padrão nas JSPs da ultima aula, acessaremos o getter através da Expression Language. Existe apenas uma pequena diferença: para chamar os métodos no JSF, em vez do cifrão (\$), usaremos a cerquilha (#).

```
<h:outputText value="#{olaMundoBean.horario}" />
```

# Managed Beans

- Ao fazer colocar o código acima, estamos dizendo que há uma classe gerenciada pelo JSF chamada OlaMundoBean que tem um método getHorario -- e que o retorno desse método será mostrado na página. É uma forma extremamente simples e elegante de ligar a view a métodos do model.

# Exercicio

- 1 – Crie uma página chamada horario e mostre o horario atual na primeira linha e o seu nome na segunda linha.
  - *Obs: seu nome deve ser manipulavel no futuro, isto é, ele deve ser uma variavel.*

# RECEBENDO INFORMAÇÕES DO USUÁRIO

- Agora que já sabemos conectar a página à camada de modelo, fica fácil obter dados do usuário! Por nossa vivência com aplicações web, até mesmo como usuários, sabemos que a forma mais comum de trazer tais dados para dentro da aplicação é através de formulários.
- A boa notícia é que no JSF não será muito diferente! Se para mostrar dados na página usamos a tag `h:outputText`, para trazer dados do usuário para dentro da aplicação, usaremos a tag `h:inputText`. Ela fará a ligação entre o atributo do seu bean e o valor digitado no campo.



# RECEBENDO INFORMAÇÕES DO USUÁRIO

- Note que a ideia é a mesma de antes: como o JSF precisará interagir com os dados desse componente, não podemos usar a tag HTML que faria o mesmo trabalho. Em vez disso, usaremos a taglib de HTML provida pelo próprio JSF, indicando como a informação digitada será guardada no bean.

```
<h:outputLabel value="Digite seu nome:"/>
```

```
<h:inputText value="#{olaMundoBean.nome}"/>
```

# BOTÃO E O FORMULÁRIO EM JSF

- Como dito antes, no entanto, o JSF tem a proposta de abstrair todo o protocolo HTTP, o JavaScript e o CSS. Para ter uma estrutura em que o formulário é marcado apenas como um agregador de campos e cada um dos botões internos pode ter funções diferentes, a estratégia do JSF foi a de deixar seu form como uma tag simples e adicionar a configuração da ação ao próprio botão.

# BOTÃO E O FORMULÁRIO EM JSF

```
<h:form>  
  <h:outputLabel for="nome" value="Digite seu nome:"/>  
  <h:inputText id="nome" value="#{olaMundoBean.nome}"/>  
  <h:commandButton value="Ok" action="#{olaMundoBean.digaOi}"/>  
</h:form>
```

- Quando o usuário clica no botão Ok, o JSF chama o setter do atributo nome do OlaMundoBean e, logo em seguida, chama o método digaOi. Repare que esta ordem é importante: o método provavelmente dependerá dos dados inseridos pelo usuário.

# BOTÃO E O FORMULÁRIO EM JSF

- Note, também, que teremos um novo método no managed bean chamado digaOi. Os botões sempre estão atrelados a métodos porque, na maior parte dos casos, realmente queremos executar alguma ação além da chamada do setter. Essa ação pode ser a de disparar um processo interno, salvar no banco ou qualquer outra necessidade.

# LEGAL...MAS E SE...

- Se o atributo ainda não tiver informação o que acontece? Aparece nulo? É renderizado?
- Sabendo que, antes de chamar o método correspondente à ação do botão, o JSF preenche os atributos através dos setters, sabemos que teremos a informação a ser mostrada para o usuário.

## LEGAL...MAS E SE...

- No entanto, muitas vezes não gostaríamos de mostrar um campo enquanto ele não estiver preenchido e, felizmente, o JSF tem uma forma bastante simples de só mostrar um `h:outputText` na tela apenas se a informação estiver preenchida! Basta usar o atributo `rendered`:

```
<h:outputText value="Oi #{olaMundoBean.nome}"  
    rendered="#{not empty olaMundoBean.nome}"/>
```

# EXERCICIOS

- 1 – Acesse a sua página de horario e verifique o código fonte gerado pela página. Repare que ele não é nada mais que simples HTML. Para isso, na maior parte dos navegadores, use ctrl + U.
- 2 – Crie uma nova pagina xhtml
  - Adicione um formulario que em seu input recebe um nome.
  - Imprima no console a chamado de botão e mostre o nome que foi digitado.

# COMPONENTES DO PRIMEFACES

- Onde entra o primefaces?
- Porque usar?
- Como usar?



# SHOWCASES NO SITE DO PRIMEFACES

- Vamos acessar o site do primefaces e visualizar alguns componentes interessantes.
- Perceba que não precisamos nos preocupar com CSS e JavaScript.

# COMO UTILIZAR OS COMPONENTES

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:p="http://primefaces.org/ui">
```

```
<h:head></h:head>
```

```
<h:body>
```

```
<p:meucomponente />
```

```
</h:body>
```

```
</html>
```

# LISTANDO INFORMAÇÕES

- Agora que já aprendemos o básico do JSF, nosso objetivo é listar em uma página de contatos. Nessa listagem, queremos mostrar as informações dos contatos carregados, isto é, queremos uma forma de mostrar nome, email, endereço e data de nascimento. E a forma mais natural de apresentar dados desse tipo é, certamente, uma tabela.

# LISTANDO INFORMAÇÕES

- Até poderíamos usar a tabela que vem na taglib padrão do JSF, mas ela é bastante limitada e não tem pré-definições de estilo. Isto é, usando a taglib padrão, teremos sim uma tabela no HTML, mas ela será mostrada da forma mais feia e simples possível.
- Já falamos, contudo, que a proposta do JSF é abstrair toda a complexidade relativa à web -- e isso inclui CSS, formatações, JavaScript e tudo o mais. Então, em apoio às tags básicas, algumas bibliotecas mais sofisticadas surgiram. As mais conhecidas delas são PrimeFaces, RichFaces e IceFaces.

# LISTANDO INFORMAÇÕES

- Taglibs como essas oferecem um visual mais bacana já pré-pronto e, também, diversas outras facilidades. Por exemplo, uma tabela que utilize as tags do Primefaces já vem com um estilo bonito, possibilidade de colocar cabeçalhos nas colunas e até recursos mais avançados como paginação dos registros.
- O componente responsável por produzir uma tabela baseada em um modelo se chama dataTable. Ele funciona de forma bem semelhante ao for do Java 5 ou o forEach da JSTL: itera em uma lista de elementos atribuindo cada item na variável definida.

# LISTANDO INFORMAÇÕES

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:p="http://primefaces.org/ui">
  <h:head>
    <title>Argentum</title>
  </h:head>
  <h:body>
    <p:dataTable var="contato" value="#{contatoBean.contatos}">

    </p:dataTable>
  </h:body>
</html>
```

# LISTANDO INFORMAÇÕES

- O código acima chamará o método `getContatos` da classe `contatoBean` e iterará pela lista devolvida atribuindo o objeto à variável `contato`. Então, para cada coluna que quisermos mostrar, será necessário apenas manipular o contato do momento.
- E, intuitivamente o bastante, cada coluna da tabela será representada pela tag `p:column`. Para mostrar o valor, você pode usar a tag que já vimos antes, o `h:outputText`. Note que as tags do Primefaces se integram perfeitamente com as básicas do JSF.

# LISTANDO INFORMAÇÕES

```
<p:dataTable var="contato" value="#{contatoBean.contatos}">  
  <p:column headerText="Preço">  
    <h:outputText value="#{contato.nome}"/>  
  </p:column>  
  ... outras colunas  
</p:dataTable>
```



# LISTANDO INFORMAÇÕES

- Falta ainda implementar a classe que cuidará de devolver essa lista de negociações. O código acima sugere que tenhamos uma classe chamada `ArgentumBean`, gerenciada pelo JSF, que tenha um getter de negociações que pode, por exemplo, trazer essa lista direto do `ClienteWebService` que fizemos anteriormente:

```
@ManagedBean
public class ContatoBean {

    public List<Contato> getContatos() {
        ..Vamos Criar alguns contatos e retornar uma lista
    }
}
```

# EXERCICIOS

- 1 – Vamos criar o nosso modelo Contato
  - Um contato deve possuir
    - Nome
    - E-mail
    - Endereço
    - DataDeNascimento
- 2 – Crie um Manage Bean ContatoBean.
  - Esse Bean será responsável por retornar uma lista de contatos

# EXERCICIOS

- 3 – Crie uma página xhtml chamada listadecontatos.
  - Você deve chamar o método getContatos do seu Bean e listar todos os contatos em uma dataTable utilizando o primefaces.

# FORMATAÇÃO DE DATA COM JSF

- A tabela já é funcional, mas com a data mal formatada. O componente não sabe como gostaríamos de formatar a data e chama por de baixo dos panos o método `toString` da data para receber uma apresentação como `String`.

# FORMATAÇÃO DE DATA COM JSF

- A forma clássica de resolver esse problema seria através de um getter que traria a data formatada por um SimpleDateFormat. Mas, JSF também tem uma tag para formatar valores, números e, claro, datas. Essas tags e muitas outras, são parte da biblioteca fundamental de tags lógicas do JSF e, para usá-las, será necessário importar uma taglib.

# FORMATAÇÃO DE DATA COM JSF

- Assim como as bibliotecas de tags de HTML e do Primefaces, para utilizar essas será necessário declará-las no namespace da sua página.
- Daí, podemos facilmente mudar a forma padrão de exibição usando o componente de formatação `f:convertDateTime` que define um pattern para a data. É importante lembrar que, internamente, o `f:convertDateTime` acaba fazendo uma chamada ao `SimpleDateFormat` e, assim, só podemos formatar objetos do tipo `java.util.Date` com ele. Por essa razão, chamaremos o método `getTime` que devolve a representação em `Date` do Calendar em questão. Mais uma vez podemos omitir a palavra "get" com expression language. Segue a tabela completa:

# FORMATAÇÃO DE DATA COM JSF

```
<p:column headerText="Data de nascimento">  
  <h:outputText value="#{contato.datadenascimento.time}">  
    <f:convertDateTime pattern="dd/MM/yyyy"/>  
  </h:outputText>  
</p:column>
```

# EXERCICIOS

1 – Mostre a data formatada na tabela do ultimo exercício.



# PAGINAÇÃO E ORDENAÇÃO

- O componente `p:dataTable` sabe listar items, mas não pára por aí. Ele já vem com várias outras funcionalidades frequentemente necessárias em tabelas já prontas e fáceis de usar.

## Muitos dados

- Por exemplo, quando um programa traz uma quantidade muito grande de dados, isso pode causar uma página pesada demais para o usuário que provavelmente nem olhará com atenção todos esses dados.

# PAGINAÇÃO E ORDENAÇÃO

- Para habilitar a paginação automática, basta adicionar o atributo `paginator="true"` à sua `p:dataTable` e definir a quantidade de linhas por página pelo atributo `rows`. A definição da tabela de negociações para paginação de 15 em 15 resultados ficará assim:

```
<p:dataTable var="contato" value="#{contatoBean.contatos}"  
    paginator="true" rows="15">
```

```
<!-- colunas omitidas -->
```

```
</p:dataTable>
```

# PAGINAÇÃO E ORDENAÇÃO

- Essa pequena mudança já traz uma visualização mais legal para o usuário, mas estamos causando um problema silencioso no servidor. A cada vez que você chama uma página de resultados, a cada requisição, o ContatoBean é recriado e perdemos a lista anterior. Assim, na criação da nova instância de ContatoBean, seu construtor é chamado e criamos novamente toda a lista.
- Como recebemos a lista completa do método, podíamos aproveitar a mesma lista para todas as páginas de resultado e, felizmente, isso também é bastante simples.

# PAGINAÇÃO E ORDENAÇÃO

- O comportamento padrão de um ManagedBean é durar apenas uma requisição. Em outras palavras, o escopo padrão de um ManagedBean é de request. Com apenas uma anotação podemos alterar essa duração. Os três principais escopos do JSF são:

# PAGINAÇÃO E ORDENAÇÃO

- **RequestScoped**: é o escopo padrão. A cada requisição um novo objeto do bean será criado;
- **ViewScoped**: escopo da página. Enquanto o usuário estiver na mesma página, o bean é mantido. Ele só é recriado quando acontece uma navegação em sí, isto é, um botão abre uma página diferente ou ainda quando acessamos novamente a página atual.
- **SessionScoped**: escopo de sessão. Enquanto a sessão com o servidor não expirar, o mesmo objeto do ContatoBean atenderá o mesmo cliente. Esse escopo é bastante usado, por exemplo, para manter o usuário logado em aplicações.

# PAGINAÇÃO E ORDENAÇÃO

- No nosso caso, o escopo da página resolve plenamente o problema: enquanto o usuário não recarregar a página usaremos a mesma listagem. Para utilizá-lo, basta adicionar ao bean a anotação `@ViewScoped`. No exemplo do Contato:

```
@ManagedBean  
@ViewScoped  
public class ContatoBean {  
    ...  
}
```

# PAGINAÇÃO E ORDENAÇÃO

- Sempre que um ManagedBean possuir o escopo maior que o escopo de requisição, ele deverá implementar a interface Serializable:

@ManagedBean

@ViewScoped

public class ContatoBean implements Serializable {

...

# Tirando informações mais facilmente

- Outra situação clássica que aparece quando lidamos com diversos dados é precisarmos vê-los de diferentes formas em situações diversas.
- Se quisermos encontrar um contato específico em nossa tabela, é melhor que ela esteja ordenada pelo nome. Mas caso precisemos pegar os contatos de todas as pessoas de uma região, é melhor que a tabela esteja ordenada, por exemplo, pelo DDD.



# Tirando informações mais facilmente

- Essa ideia de ordenação é extremamente útil e muito presente em aplicações. Como tal, essa funcionalidade também está disponível para tabelas do Primefaces. Apenas, como podemos tornar diversas colunas ordenáveis, essa configuração fica na tag da coluna.
- Para tornar uma coluna ordenável, é preciso adicionar um simples atributo sortBy à tag h:column correspondente. Esse atributo torna o cabeçalho dessa coluna em um elemento clicável e, quando clicarmos nele, chamará a ordenação.

# Tirando informações mais facilmente

- Contudo, exatamente pela presença de elementos clicáveis, será necessário colocar a tabela dentro de uma estrutura que comporte botões em HTML: um formulário. E, como quem configurará o que cada clique vai disparar é o JSF, será necessário usar o formulário da taglib de HTML dele. Resumidamente, precisamos colocar a tabela inteira dentro do componente `h:form`.

# Tirando informações mais facilmente

- Se quiséssemos tornar ordenáveis as colunas da tabela de contatos, o resultado final seria algo como:

```
<h:form id="listaContato">
  <p:dataTable var="contato" value="#{contatoBean.contatos}">

    <p:column sortBy="#{contato.nome}" headerText="Nome" >
      <h:outputText value="#{contato.nome}" />
    </p:column>

    <!-- outras colunas omitidas -->
  </p:dataTable>
</h:form>
```

# Tirando informações mais facilmente

- Note que não foi necessário adicionar código algum à classe ContatoBean! Note também que é até possível usar ambas as funcionalidades na mesma tabela. E essas são apenas algumas das muitas facilidades que o p:dataTable oferece. Vale a pena verificar o showcase e documentação no site do Primefaces.

# Exercicios

- 1 – Vamos tornar nossa tabela ordenável por todas as colunas.
- 2 – Vamos adicionar paginação em nossa tabela

# Bons Estudos

Namom Alves Alencar

