JavaServer Faces: A mais nova tecnologia Java para desenvolvimento WEB

Talita Pitanga

Conheça um pouco sobre a mais nova tecnologia para desenvolvimento de aplicações WEB: JavaServer Faces.

Introdução

As tecnologias voltadas para o desenvolvimento de aplicações WEB têm mudado constantemente. Como sabemos, inicialmente os sites possuíam apenas conteúdo estático, ou seja, o conteúdo de uma página não podia ser modificado em tempo de execução. Depois, os sites passaram a oferecer páginas com conteúdos dinâmicos e personalizados. Diversas tecnologias estão envolvidas no desenvolvimento das aplicações WEB como, por exemplo, CGI (Common Gateway Interface), Servlets e JSP (Java Server Pages).

A primeira tecnologia voltada para a construção de páginas dinâmicas foi a CGI. Os programas CGI podem ser escritos em qualquer linguagem de programação. Eles, porém, apresentam problemas de portabilidade e escalabilidade, além de mesclarem as regras de negócio com a visualização. Vale salientar que um servidor que usa este tipo de tecnologia pode ter seu desempenho comprometido, uma vez que cada solicitação recebida de uma CGI requer a criação de um novo processo.

Em seguida vieram os servlets. Similarmente às CGIs, servlets são pequenos programas feitos em Java que encapsulam alguma funcionalidade inerente à sua aplicação WEB. Diferentemente das CGIs, servlets são objetos Java que não precisam ser executados em outro processo: o processamento é executado dentro de uma thread do processo do servidor. No entanto, eles ainda não resolvem o problema da separação das regras de negócio da visualização, dificultando a manutenção.

Posteriormente surgiram as páginas JSP. Elas são facilmente codificadas e produzem conteúdos reutilizáveis. Assim como os servlets, as JSPs também não resolvem o problema da manutenção das aplicações.

Esse problema só foi resolvido quando começou a se aplicar os design patterns no desenvolvimento das páginas. No caso das tecnologias para desenvolvimento WEB usando Java, o design pattern utilizado é o MVC (Model-View-Controller). Se você não sabe o que é MVC, fique tranqüilo!! Falaremos um pouco sobre esse padrão mais adiante. Mas você pode encontrar algo sobre esse pattern bem aqui pertinho: http://www.guj.com.br/forum/viewtopic.php?t=7228&highlight=mvc.

Com a grande utilização dos patterns, principalmente no “mundo Java”, começaram a surgir diversos frameworks para auxiliar no desenvolvimento de aplicações WEB. Posso apostar que a maioria de vocês já ouviu falar em Struts ou WebWork.

Finalmente chegamos ao assunto que se propõe esse artigo. Se você for como eu, já deve ter feito a seguinte pergunta para si mesmo: “Beleza... mas e daí? Afinal, o que é JavaServer Faces?”

O que é JavaServer Faces?

JSF é uma tecnologia que incorpora características de um framework MVC para WEB e de um modelo de interfaces gráficas baseado em eventos. Por basear-se no padrão de projeto MVC, uma de suas melhores vantagens é a clara separação entre a visualização e regras de negócio (modelo).

A idéia do padrão MVC é dividir uma aplicação em três camadas: modelo, visualização e controle. O modelo é responsável por representar os objetos de negócio, manter o estado da aplicação e fornecer ao controlador o acesso aos dados. A visualização representa a interface com o usuário, sendo responsável por definir a forma como os dados serão apresentados e encaminhar as ações dos usuários para o controlador. Já a camada de controle é responsável por fazer a ligação entre o modelo e a visualização, além de interpretar as ações do usuário e as traduzir para uma operação sobre o modelo, onde são realizadas mudanças e, então, gerar uma visualização apropriada.

O Padrão MVC segundo JSF

No JSF, o controle é composto por um servlet denominado *FacesServlet*, por arquivos de configuração e por um conjunto de manipuladores de ações e observadores de eventos. O *FacesServlet* é responsável por receber requisições da WEB, redirecioná-las para o modelo e então remeter uma resposta. Os arquivos de configuração são responsáveis por realizar associações e mapeamentos de ações e pela definição de regras de navegação. Os manipuladores de eventos são responsáveis por receber os dados vindos da camada de visualização, acessar o modelo, e então devolver o resultado para o *FacesServlet*.

O modelo representa os objetos de negócio e executa uma lógica de negócio ao receber os dados vindos da camada de visualização. Finalmente, a visualização é composta por *component trees* (hierarquia de componentes UI), tornando possível unir um componente ao outro para formar interfaces mais complexas. A Figura 1 mostra a arquitetura do JavaServer Faces baseada no modelo MVC.

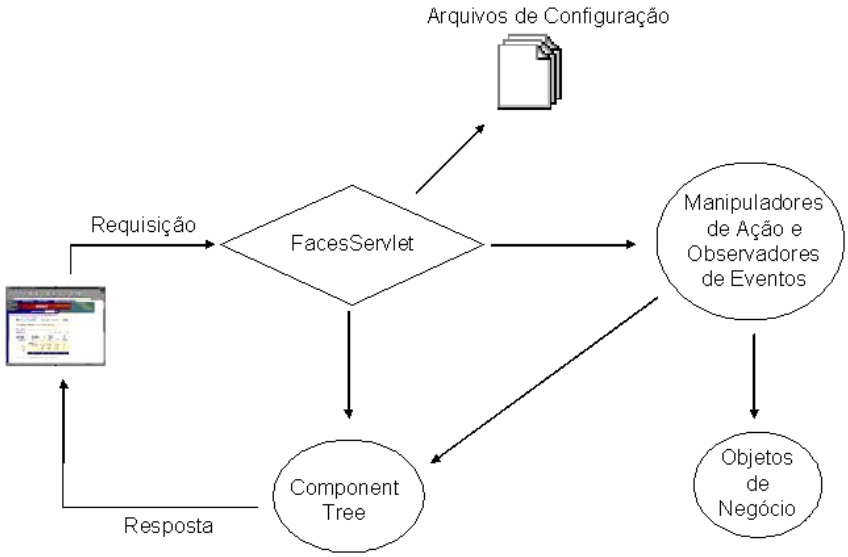

Figura 1 - Arquitetura JSF baseada no modelo MVC

Características e Vantagens

JavaServer Faces oferece ganhos no desenvolvimento de aplicações WEB por diversos motivos:

- Permite que o desenvolvedor crie UIs através de um conjunto de componentes UIs pré-definidos;
- Fornece um conjunto de tags JSP para acessar os componentes;
- Reusa componentes da página;
- Associa os eventos do lado cliente com os manipuladores dos eventos do lado servidor (os componentes de entrada possuem um valor local representando o estado no lado servidor);
- Fornece separação de funções que envolvem a construção de aplicações WEB.

Embora JavaServer Faces forneça tags JSP para representar os componentes em uma página, ele foi projetado para ser flexível, sem limitar-se a nenhuma linguagem markup em particular, nem a protocolos ou tipo de clientes. Ele também permite a criação de componentes próprios a partir de classes de componentes, conforme mencionado anteriormente.

JSF possui dois principais componentes: Java APIs para a representação de componentes UI e o gerenciamento de seus estados, manipulação/observação de eventos, validação de entrada, conversão de dados, internacionalização e acessibilidade; e taglibs JSP que expressam a interface JSF em uma página JSP e que realizam a conexão dos objetos no lado servidor.

É claro que existe muito mais a ser dito sobre JavaServer Faces. Esse artigo apenas fornece uma visão geral, mas espero ter criado uma certa curiosidade a respeito dessa nova tecnologia.

Download e Instalação

JavaServer Faces pode ser usado em conjunto com o Sun Java™ System Application Server - Plataform Edition 8, com o Java™ Web Services Developer Pack (Java WSDP) ou ainda com outro Container, tal como o Tomcat.

O download de JavaServer Faces pode ser obtido em http://java.sun.com/j2ee/javaserverfaces/download.html. Caso você ainda não tenha o Tomcat, você pode fazer o download em http://jakarta.apache.org/tomcat.

Instale o Tomcat e logo em seguida descompacte o arquivo do JavaServer Faces. Acrescente ao seu Classpath o diretório *lib* do JavaServer Faces. Copie qualquer arquivo .WAR do diretório *samples* do JavaServer Faces para o diretório *webapps* do seu servidor Tomcat. Se o seu servidor não estiver rodando, inicialize-o.

Agora só falta testar! Abra o browser e digite http://localhost:8080/jsf-cardemo (supondo que copiamos o arquivo jsf-cardemo.war). Se não tiver ocorrido nenhum erro, parabéns! Caso contrário, reveja os passos de instalação/configuração.

Vamos agora ao nosso exemplo.

Implementando um exemplo em JSF

Será uma aplicação bem simples para demonstrar o uso dessa nova tecnologia. O nosso exemplo consiste de uma página inicial contendo 2 links: um para a inserção de dados e outro para a busca.

A página de inserção consiste de um formulário onde o usuário entrará com o nome, endereço, cidade e telefone. Os dados serão guardados em um banco de dados (no meu caso, eu uso o PostgreSQL) para uma posterior consulta. Se o nome a ser inserido já existir no banco de dados, uma mensagem será exibida informando ao usuário que o nome já está cadastrado (no nosso exemplo, o nome é a chave primária da tabela). Caso contrário, uma mensagem de sucesso será exibida ao usuário.

A busca se dará pelo nome da pessoa. Se o nome a ser buscado estiver cadastrado no banco, então uma página com os dados relativos ao nome buscado serão exibidos. Caso contrário, será informado ao usuário que o nome buscado não existe no banco.

Para a criação da tabela da base de dados foi utilizado o script apresentado abaixo.

```
CREATE TABLE pessoa
(
    nome varchar(30) NOT NULL,
    endereco varchar(50),
    cidade varchar(20),
    telefone varchar(10),
    PRIMARY KEY (nome)
);
```

Vamos utilizar a IDE eclipse (http://www.eclipse.org/) para o desenvolvimento deste exemplo devido à existência de um plugin para JSF (http://www.exadel.com/products_jsfstudio.htm), tornando o desenvolvimento mais fácil.

No eclipse, escolha a opção New Project do menu File. Depois, escolha **JSF Project**. Nomeie seu projeto como exemplo-jsf. Com o seu projeto criado, vá até o diretório *WebContent*. As páginas JSP devem ficar armazenadas nesse diretório.

O código da página *index.jsp* ficará como mostrado abaixo.

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <head>
    <title>Exemplo JSF</title>
  </head>
  <body>
    <f:view>
      <h:form>
         <center>
           <h1>Agenda</h1>
           <br>
           <h3>
```

```
            <h:outputLink value="inserir.jsf">
               <f:verbatim>Inserir</f:verbatim>
             </h:outputLink>
             <br><br>
             <h:outputLink value="buscar.jsf">
               <f:verbatim>Buscar</f:verbatim>
             </h:outputLink>
           </h3>
         </center>
       </h:form>
    </f:view>
  </body>
</html>
```

Algumas tags aqui merecem ser comentadas:

- <h:form> gera um formulário.
- <h:outputLink> cria um link para a página definida pelo campo *value*. O texto que compõe o link é colocado utilizando-se a tag <f:verbatim>.

O usuário terá a opção de buscar ou inserir dados. Os códigos das páginas de busca e inserção são mostrados a seguir, respectivamente.

buscar.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form>
         <center><h2> Busca </h2></center>
         <br>
         Digite o nome:
         <h:inputText id="nome" value="#{agenda.nome}"/>
         <h:commandButton value="OK"     action="#{agenda.buscar}"/>
       </h:form>
       <br>
       <h:outputLink value="index.jsf">
         <f:verbatim>voltar</f:verbatim>
       </h:outputLink>
    </f:view>
  </body>
</html>
```

inserir.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form>
         <center><h2> Inserção </h2></center>
         <br>
         <h3>Entre com os dados abaixo</h3>
         <table>
           <tr>
            <td>Nome:</td>
            <td>
             <h:inputText value="#{agenda.nome}"/>
            </td>
           </tr>
           <tr>
            <td>Endereço:</td>
            <td>
              <h:inputText value="#{agenda.endereco}"/>
            </td>
           </tr>
           <tr>
```

```
            <td>Cidade:</td>
            <td>
             <h:inputText value="#{agenda.cidade}"/>
            </td>
           </tr>
           <tr>
            <td>Telefone:</td>
            <td>
              <h:inputText value="#{agenda.telefone}"/>
            </td>
           </tr>
       </table>
       <p>
         <h:commandButton value="Inserir" action="#{agenda.inserir}"/>
       </p>
      </h:form>
      <br>
      <h:outputLink value="index.jsf">
        <f:verbatim>voltar</f:verbatim>
      </h:outputLink>
    </f:view>
  </body>
</html>
```

A tag <h:inputText> cria uma caixa de texto onde o valor digitado é guardado em *value*. Para criar um botão, é utilizada a tag <h:commandButton>. O label do botão é colocado em *value*. *Action* determina qual a ação que o botão deve tomar. Na página buscar.jsp, ao clicar no botão OK, o método buscar() da classe AgendaDB é chamado.

O código da classe bean AgendaDB com os métodos getters e setters das variáveis nome, endereço, cidade e telefone e com os métodos inserir e buscar ficará conforme mostrado abaixo. É nesse arquivo onde a conexão com o banco é feita. Nas aplicações JSF, os beans são usados para que dados possam ser acessados através de uma página. O código java referente a essa classe deve estar localizado no diretório JavaSource. Já que estamos utilizando um Java bean, um *managed-bean* deverá ser criado no arquivo *faces-config.xml* que está presente no diretório *WEB-INF*.

```
import java.sql.*;

public class AgendaDB {
    private String nome = blank;
    private String endereco = blank;
    private String cidade = blank;
    private String telefone = blank;

    private String result_busca = blank;
    private String result_inserir = blank;

    public static final String BUSCA_INVALIDA = "failure";
    public static final String BUSCA_VALIDA = "success";
    public static final String SUCESSO_INSERCAO = "success";
    public static final String FALHA_INSERCAO = "failure";

    static Connection con = null;
    static Statement stm = null;
    static ResultSet rs;
    static private String blank = "";

    public AgendaDB() {
        if (con==null) {
            try {
                Class.forName("org.postgresql.Driver");
                con =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/talita","talita","tata");
            } catch (SQLException e) {
                System.err.println ("Erro: "+e);
                con = null;
            } catch (ClassNotFoundException e) {
                 System.out.println("ClassNotFound...");
```

```
                e.printStackTrace();
             }
         }
    }

    public String getNome() {
        return nome;
    }

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

    public String getCidade() {
         return cidade;
    }

    public void setCidade(String cidade) {
         this.cidade = cidade;
    }

    public String getEndereco() {
         return endereco;
    }

    public void setEndereco(String endereco) {
         this.endereco = endereco;
    }

    public String getTelefone() {
         return telefone;
    }

    public void setTelefone(String telefone) {
         this.telefone = telefone;
    }

    public String inserir() {
         String result_inserir = FALHA_INSERCAO;
        try {
           stm = con.createStatement();
           stm.execute("INSERT INTO pessoa(nome,endereco,cidade,telefone) VALUES ('" + nome + "','"+
endereco + "','" + cidade + "','"+ telefone + "')");
           stm.close();
           result_inserir = SUCESSO_INSERCAO;

       } catch (SQLException e) {
           System.err.println ("Erro: "+e);
          result_inserir = FALHA_INSERCAO;
       }
       return result_inserir;
    }

    public String buscar() throws SQLException {
         String result_busca = BUSCA_INVALIDA;
        try {
           stm = con.createStatement();
           rs = stm.executeQuery("SELECT * FROM pessoa WHERE nome = '" + nome + "'");
           if (rs.next()) {
             nome = rs.getString(1);
             endereco = rs.getString(2);
             cidade = rs.getString(3);
             telefone = rs.getString(4);
             result_busca = BUSCA_VALIDA;
           }
           else
             result_busca = BUSCA_INVALIDA;
           rs.close();
           stm.close();
        } catch (SQLException e) {
           System.err.println ("Erro: "+e);
        }
        return result_busca;
    }
}
```

Ainda existirão mais 4 páginas JSP em nosso exemplo:

- sucesso_busca.jsp: esta página informa ao usuário que a busca foi bem sucedida, apresentando os dados referentes ao nome procurado;

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form>
        <center><h2> Resultado da Busca </h2></center>
        <br>
        <table>
       <tr>
          <td>Nome:</td>
          <td>
            <h:outputText value="#{agenda.nome}"/>
          </td>
        </tr>
        <tr>
          <td>Endereço:</td>
          <td>
            <h:outputText value="#{agenda.endereco}"/>
          </td>
        </tr>
        <tr>
          <td>Cidade:</td>
          <td>
            <h:outputText value="#{agenda.cidade}"/>
          </td>
        </tr>
        <tr>
          <td>Telefone:</td>
          <td>
            <h:outputText value="#{agenda.telefone}"/>
          </td>
        </tr>
        </table>
      </h:form>
      <br>
      <h:outputLink value="index.jsf">
        <f:verbatim>voltar</f:verbatim>
      </h:outputLink>
    </f:view>
  </body>
</html>
```

- falha_busca.jsp: esta página informa ao usuário que o nome buscado não existe no banco de dados;

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form>
        <h3>
       <h:outputText value="#{agenda.nome}"/>
       não foi encontrado(a)!
        </h3>
      </h:form>
      <br>
      <h:outputLink value="buscar.jsf">
        <f:verbatim>voltar</f:verbatim>
      </h:outputLink>
    </f:view>
  </body>
</html>
```

- sucesso_insercao.jsp: informa que os dados foram inseridos com sucesso no banco;

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```
<html>
  <body>
    <f:view>
      <h:form>
        Dados Inseridos com Sucesso!
      </h:form>
      <br>
      <h:outputLink value="index.jsf">
        <f:verbatim>voltar</f:verbatim>
      </h:outputLink>
    </f:view>
  </body>
</html>
```

- falha_insercao.jsp: informa que os dados não foram inseridos porque já existe no banco um nome igual ao que está se tentando inserir.

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form>
        <h3>
       <h:outputText value="#{agenda.nome}"/>
       já está cadastrado!Entre com outro nome!
        </h3>
      </h:form>
      <br>
      <h:outputLink value="inserir.jsf">
        <f:verbatim>voltar</f:verbatim>
      </h:outputLink>
    </f:view>
  </body>
</html>
```

Arquivos de configuração

Finalmente faltam os arquivos de configuração *faces-config.xml* e *web.xml*. No arquivo *faces-config.xml* nós vamos definir as regras de navegação e o managed-bean relativo à nossa classe AgendaDB. O código está mostrado logo abaixo.

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
  <navigation-rule>
    <from-view-id>/buscar.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/sucesso_busca.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>failure</from-outcome>
      <to-view-id>/falha_busca.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <navigation-rule>
    <from-view-id>/inserir.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/sucesso_insercao.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>failure</from-outcome>
```

```
      <to-view-id>/falha_insercao.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <managed-bean>
    <managed-bean-name>agenda</managed-bean-name>
    <managed-bean-class>AgendaDB</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

O primeiro bloco contém uma regra de navegação para a página *buscar.jsp*. Caso o método buscar() da classe AgendaDB retorne “success”, então a página chamada será *sucesso_busca.jsp*. Caso a palavra retornada pelo método seja “failure”, então a página a ser exibida deverá ser *falha_busca.jsp*. O segundo bloco contém a mesma regra do primeiro, sendo que para a página *inserir.jsp*.

O terceiro bloco refere-se ao managed-bean. No parâmetro class, deve ser indicado o nome da classe Java bean. O nome fica a critério do desenvolvedor, podendo até mesmo ser o mesmo da classe.

O código do arquivo *web.xml* pode ser visto abaixo.

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <context-param>
   <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
   <param-value>client</param-value>
  </context-param>

  <context-param>
   <param-name>javax.faces.CONFIG_FILES</param-name>
   <param-value>/WEB-INF/faces-config.xml</param-value>
  </context-param>

  <listener>
   <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
  </listener>

  <!-- Faces Servlet -->
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup> 1 </load-on-startup>
  </servlet>

  <!-- Faces Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
</web-app>
```

A linha <url-pattern>*.jsf</url-pattern> presente no último bloco desse arquivo, indica que as páginas terão a extensão jsf. Por esse motivo que qualquer chamada de uma página dentro de outra a extensão não é jsp e sim jsf. Existem outras opções de configuração. Mas que não estão no escopo desse artigo.

Dentro do diretório *lib* ficam armazenados os arquivos .jar. Como o nosso exemplo acessa um banco de dados, devemos adicionar a esse diretório o driver relativo à versão do banco que se está utilizando. Para esse exemplo eu utilizei a versão 8.0 do PostgreSQL (http://www.postgresql.org). Você pode obter o driver em http://jdbc.postgresql.org/download.html.

Rodando o exemplo

Com todas as páginas, classe Java e arquivos de configuração criados, podemos finalmente rodar a nossa aplicação. Podemos fazer isso de duas maneiras. Primeiro, você pode utilizar as opções que o Eclipse oferece. Clique com o botão direito em cima do diretório exemplo-jsf. Em *JSF Studio* escolha *Register Web Context in Tomcat*. Em seguida, inicialize o servidor. Abra o browser e digite http://localhost:8080/exemplo-jsf/index.jsf.

Você pode, ao invés de registrar a sua aplicação no Tomcat, gerar o arquivo WAR. Para isso, vá até o diretório *WebContent* da sua aplicação e então digite

jar –cvf exemplo-jsf .

Em seguida, copie o arquivo WAR para o diretório *webapps* do Tomcat. Inicialize o servidor, abra o browser e digite o mesmo endereço já dito anteriormente.

Conclusão

JavaServer Faces é uma tecnologia bastante recente para o desenvolvimento de aplicações WEB e pouco tem se visto ainda sobre essa tecnologia. Como já mencionado, esse artigo fornece apenas uma simples introdução sobre o assunto. Existe muito mais a ser dito a respeito dessa tecnologia.

O exemplo aqui mostrado faz uso de pouquíssimos recursos oferecidos pelo JSF. Lembre-se que o JSF faz uso do tratamento de eventos (que não foi demonstrado nesse exemplo). Mas acredito que outros artigos virão e, quem sabe nos próximos poderemos tratar mais a fundo essa tecnologia

Talita Pitanga (*tpitanga@gmail.com*) é estudante do curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte – UFRN.