
Apostila Java Web



Sumário

1	Introdução.....	3
2	Estrutura básica de uma aplicação Java Web.....	4
3	Fundamentos do Java EE.....	7
4	Tecnologia Java Servlets.....	9
5	JavaServer Pages (JSP).....	14
6	Model View Controller (MVC).....	18
7	JavaServer Faces 2.0 (JSF).....	19
8	Introdução ao PrimeFaces.....	21
9	Desenvolvendo um CRUD com PrimeFaces.....	31
10	Conclusão.....	51

1 Introdução

O que é um aplicativo Web Java?

Um aplicativo Web Java gera páginas Web interativas, que contêm vários tipos de linguagem de marcação (HTML, XML, etc.) e conteúdo dinâmico. Normalmente é composto por componentes Web, como JavaServer Pages (JSP), servlets e JavaBeans para modificar e armazenar dados temporariamente, interagir com bancos de dados e serviços Web e processar o conteúdo como resposta às requisições do cliente.

Como a maioria das tarefas envolvidas no desenvolvimento de aplicativos da Web, pode ser repetitiva ou exigir um excedente de código padrão, os frameworks Web podem ser aplicados para aliviar a sobrecarga associada às atividades comuns. Muitos frameworks, como JavaServer Faces, fornecem, por exemplo, bibliotecas para páginas de modelo e gerenciamento de sessão, e geralmente fomentam a reutilização do código.

O que é Java EE?

O Java EE (Enterprise Edition) é uma plataforma amplamente usada que contém um conjunto de tecnologias coordenadas que reduz significativamente o custo e a complexidade do desenvolvimento, implantação e gerenciamento de aplicativos de várias camadas centrados no servidor. O Java EE é construído sobre a plataforma Java SE e oferece um conjunto de APIs (interfaces de programação de aplicativos) para desenvolvimento e execução de aplicativos portáteis, robustos, escaláveis, confiáveis e seguros no lado do servidor.

Alguns dos componentes fundamentais do Java EE são:

- O Enterprise JavaBeans (EJB): uma arquitetura gerenciada de componente do lado do servidor utilizada para encapsular a lógica corporativa de um aplicativo. A tecnologia EJB permite o desenvolvimento rápido e simplificado de aplicativos distribuídos, transacionais, seguros e portáteis baseados na tecnologia Java.
- O Java Persistence API (JPA): uma estrutura que permite aos desenvolvedores gerenciar os dados utilizando o mapeamento objeto-relacional (ORM) em aplicativos construídos na plataforma Java.

Desenvolvimento em JavaScript e Ajax

JavaScript é uma linguagem de script orientada a objetos utilizada principalmente em interfaces no lado do cliente para aplicativos da Web. Ajax (Asynchronous JavaScript and XML) é uma técnica Web 2.0 que permite que sejam feitas alterações nas páginas Web sem que seja necessário atualizar a página. O kit de ferramentas JavaScript pode ser aproveitado para implementar funcionalidades e componentes habilitados para o Ajax em páginas Web.

Atenção

Apesar de tanta popularidade no ambiente Web, o desenvolvimento com Java não é trivial: é necessário conhecer com certa profundidade as APIs de servlets e de JSP, mesmo que você venha utilizar frameworks como Struts, VRaptor ou JSF. Conceitos de HTTP, session e cookies também são mandatórios para poder enxergar gargalos e problemas que sua aplicação enfrentará.

Referências

Informações retiradas de:

Treinamento NetBeans

http://netbeans.org/kb/trails/java-ee_pt_BR.html?utm_source=netbeans&utm_campaign=welcomepage

Apostila FJ-21 da Caelum

www.caelum.com.br/apostilas

2 Estrutura básica de uma aplicação Java Web

O objetivo deste capítulo é compreender a estrutura básica e obrigatória de um aplicativo Java Web. Para tal, vamos utilizar a IDE NetBeans para nos auxiliar nesta tarefa. Siga os passos abaixo para criar um projeto Java Web no NetBeans:

1. Acesse o menu 'Arquivos->Novo Projeto';
2. Em Escolher Projeto selecione 'Categorias = JavaWeb' e em 'Projeto = Aplicação Web', como na imagem 1. Então clique em 'Próximo'.

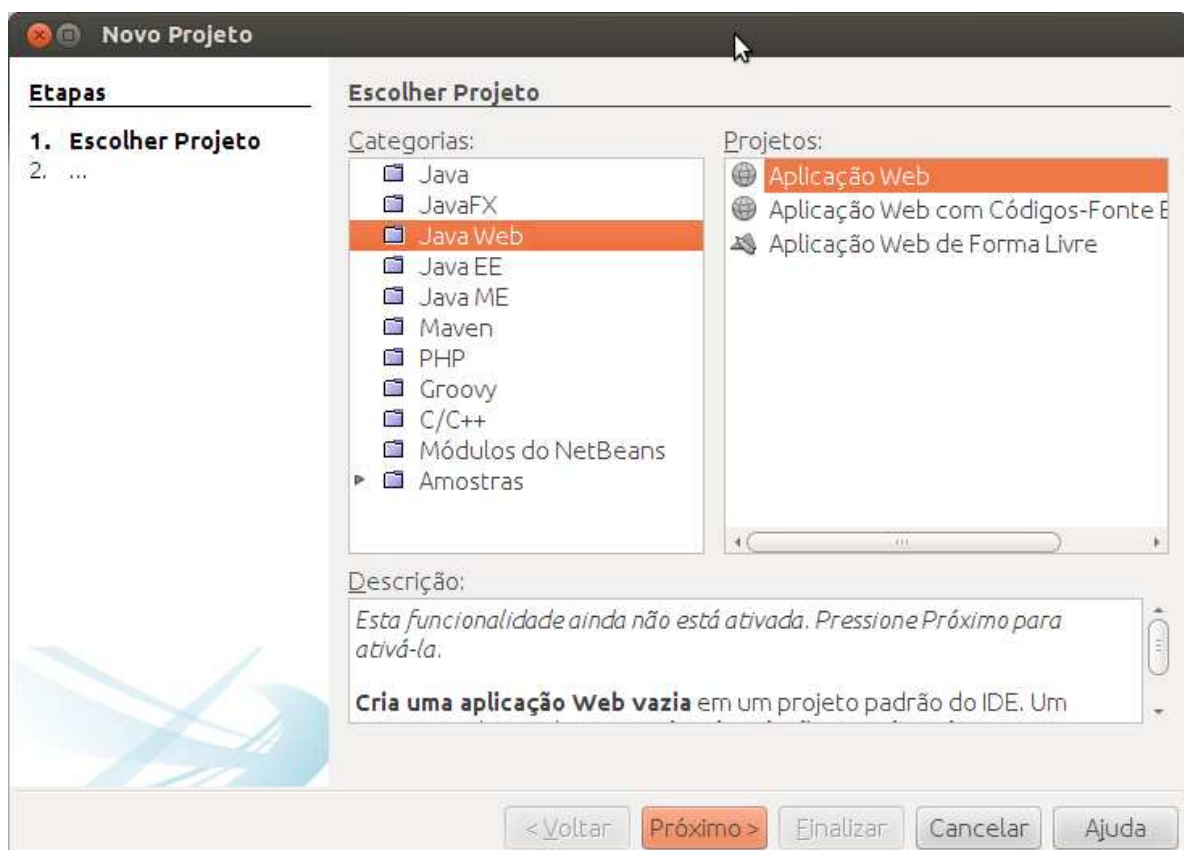


Ilustração 1: Criar novo projeto Web

3. Em 'Nome e Localização' defina o 'Nome do Projeto = OlaMundoWeb' e os demais campos deixe como padrão.

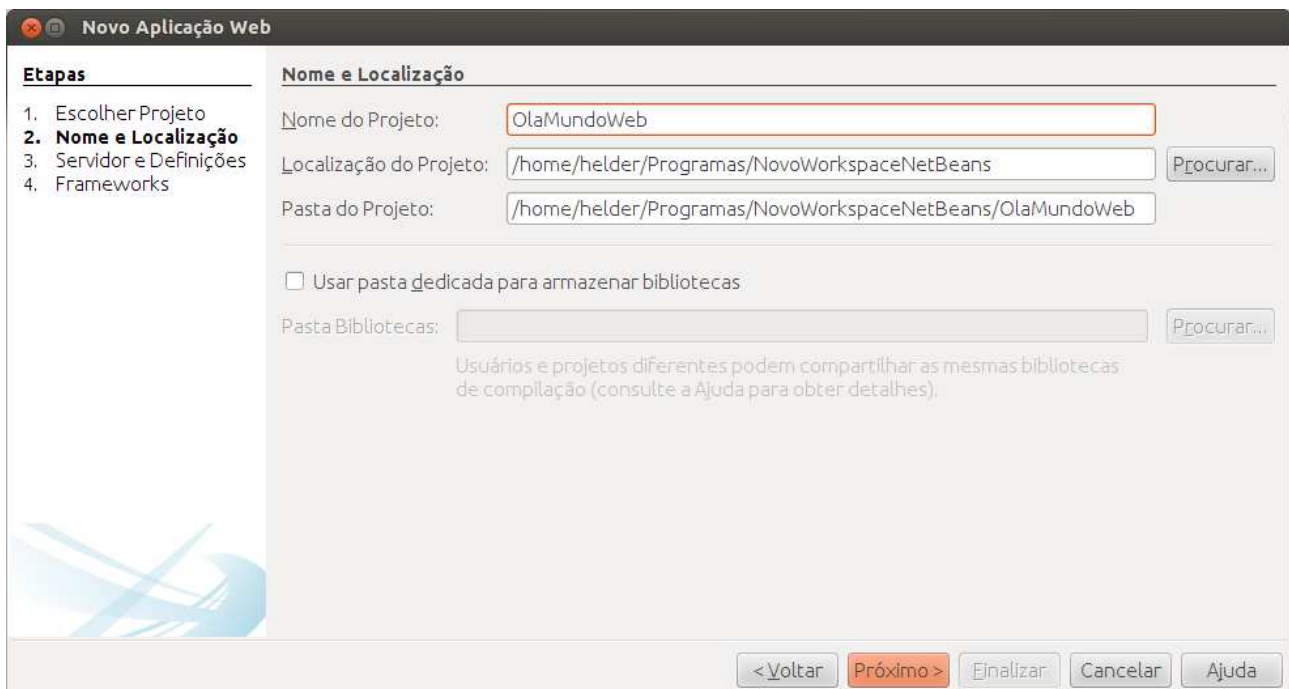


Ilustração 2: Criar novo projeto Web - Nome e localização

4. Em 'Servidor e Definições' mantenha o preenchimento padrão e clique em 'Finalizar', como na figura 3. Observe que o servidor que irá executar a aplicação Web é o Tomcat e a versão do JavaEE é a 6. Também é importante observar que o caminho de contexto é '/OlaMundoWeb' que implica que para acessar a aplicação deverá informar 'http:<endereço do servidor>:<porta do servidor>/OlaMundoWeb'.

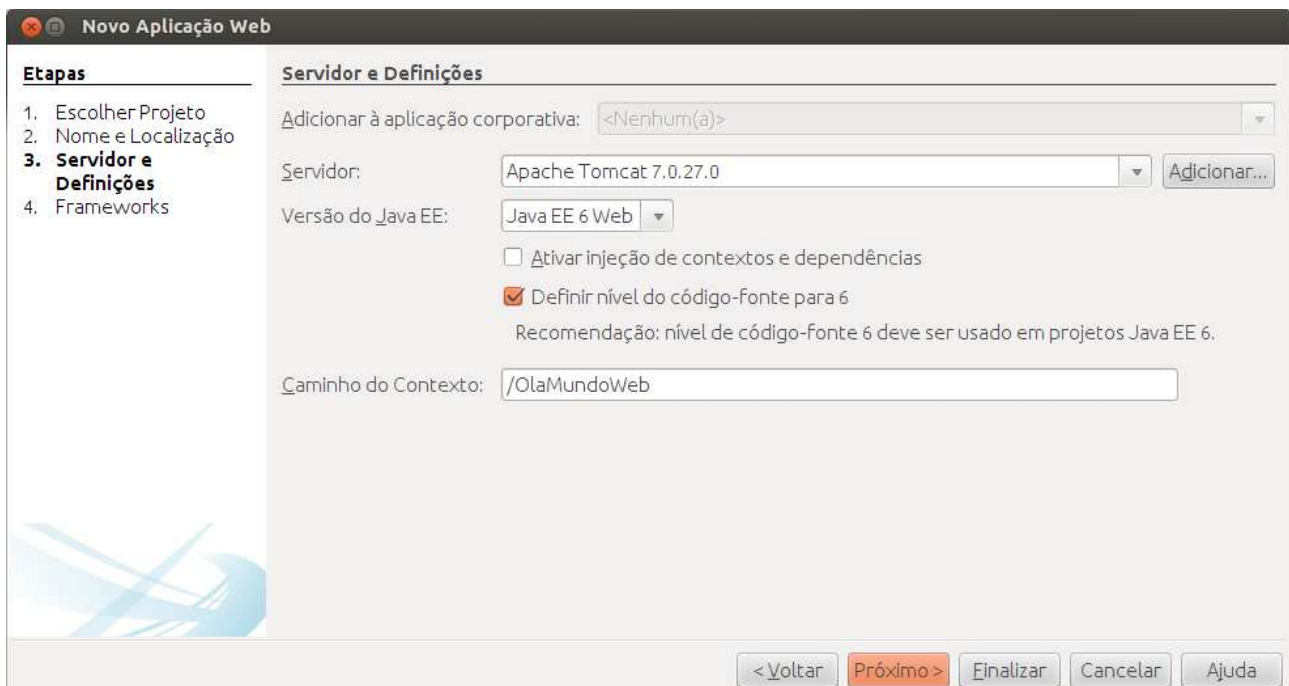


Ilustração 3: Criar novo projeto Web - Servidor

- Após estes passos o projeto 'OlaMundoWeb' foi criado. Observe a estrutura de diretórios criada acessando a aba 'Arquivos'. Toda aplicação Java Web precisa ter uma pasta para os arquivos Web (html, jsp, css e etc) e outra para os arquivos de classes Java. Neste cenário a pasta 'web' armazena os Web e a pasta src/java será a pasta que terá os arquivos Java. Veja na figura 4:

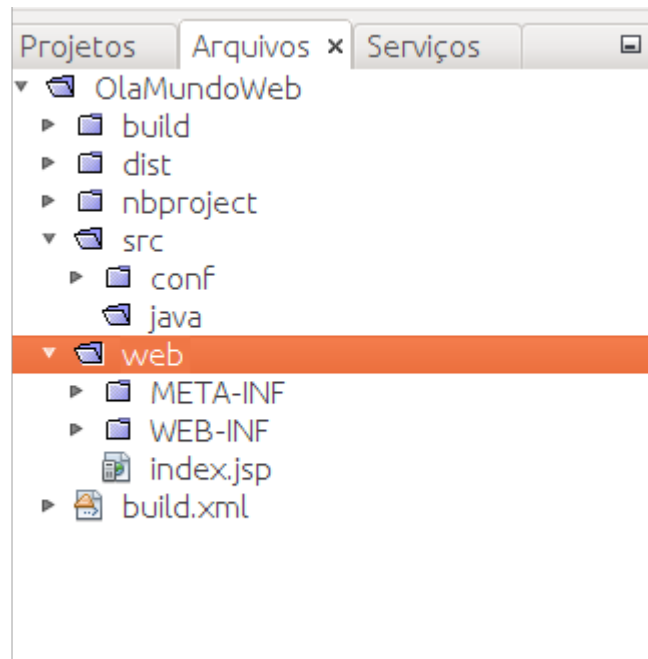


Ilustração 4: Estrutura diretórios projeto Java Web

- Observe que o único arquivo Web criado automaticamente pelo NetBeans foi o 'index.jsp' (a tecnologia JSP será estudada mais à frente), trata-se de uma página HTML comum que será a inicial porque está definida com o nome index.

```
1  <!--
2      Document   : index
3      Created on  : 07/08/2012, 11:18:41
4      Author      : helder
5  -->
6
7  <%@page contentType="text/html" pageEncoding="UTF-8"%>
8  <!DOCTYPE html>
9  <html>
10     <head>
11         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12         <title>JSP Page</title>
13     </head>
14     <body>
15         <h1>Hello World!</h1>
16     </body>
17 </html>
```

Ilustração 5: index.jsp

- Agora execute o projeto. Espera-se que exiba a tela da figura 6 que é acessada através da URL: <http://localhost:8084/OlaMundoWeb>



Ilustração 6: Tela Olá Mundo

8. Faça o seguinte teste: acesse a aba 'Projetos' e clique com o botão direito do mouse no nome do projeto, selecione a opção 'Limpar e Construir'. Vá na aba 'Arquivos' e veja que foi criado o arquivo 'OlaMundoWeb.war' na pasta 'dist'. Afinal de contas o que é este arquivo '.war'? Este é o arquivo de distribuição de aplicativos Java EE, quando você for implantar a aplicação que você desenvolveu basta colocar na pasta especificada pelo servidor Java Web. Aqui estamos testando no Tomcat, mas, este arquivo irá funcionar em qualquer servidor Java Web. A extensão 'war' é um acrônimo para 'Web application Archive'.

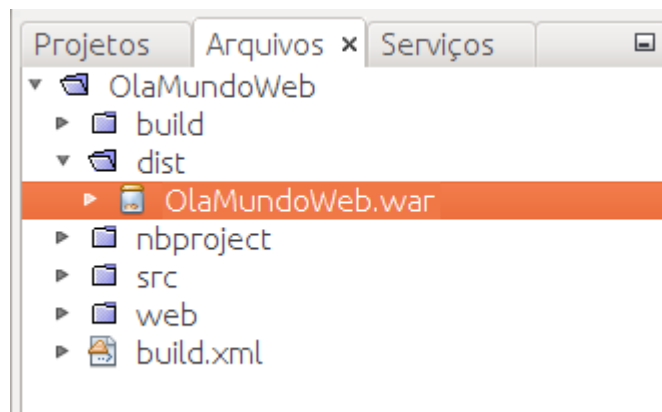


Ilustração 7: Arquivo de distribuição

3 Fundamentos do Java EE

Após sentir o gosto de ver uma aplicação Java Web funcionando é hora de fixar alguns conceitos fundamentais desta tecnologia.

Como o Java EE pode te ajudar

As aplicações Web de hoje em dia já possuem regras de negócio bastante complicadas. Codificar essas muitas regras já representam um grande trabalho. Além dessas regras, conhecidas como requisitos funcionais de uma aplicação, existem outros requisitos que precisam ser atingidos através da nossa infraestrutura: persistência em banco de dados, transação, acesso remoto, web services, gerenciamento de threads, gerenciamento de conexões HTTP, cache de objetos, gerenciamento da sessão web, balanceamento de carga, entre outros. São chamados de **requisitos não-funcionais**.

Se formos também os responsáveis por escrever código que trate desses outros requisitos, teríamos muito mais trabalho a fazer. Tendo isso em vista, a Sun criou uma série de especificações que, quando implementadas, podem ser usadas por desenvolvedores para tirar proveito e reutilizar toda essa infraestrutura já pronta.

O Java EE (Java Enterprise Edition) consiste de uma série de especificações bem detalhadas, dando uma receita de como deve ser implementado um software que faz cada um desses serviços de infraestrutura.

Algumas APIS especificadas no Java EE

- JavaServer Pages (JSP), Java Servlets, Java Server Faces (JSF) (trabalhar para a Web)
- Enterprise Javabeans Components (EJB) e Java Persistence API (JPA). (objetos distribuídos, clusters, acesso remoto a objetos etc)
- Java API for XML Web Services (JAX-WS), Java API for XML Binding (JAX-B) (trabalhar com arquivos xml e webservices)
- Java Authentication and Authorization Service (JAAS) (API padrão do Java para segurança)
- Java Transaction API (JTA) (controle de transação no contêiner)
- Java Message Service (JMS) (troca de mensagens assíncronas)
- Java Naming and Directory Interface (JNDI) (espaço de nomes e objetos)
- Java Management Extensions (JMX) (administração da sua aplicação e estatísticas sobre a mesma)

Servidor de Aplicação Java EE

É o nome dado a um servidor que implementa as especificações do Java EE. Existem diversos servidores de aplicação famosos compatíveis com a especificação Java EE. O JBoss é um dos líderes do mercado e tem a vantagem de ser gratuito e *open source*. Alguns softwares implementam apenas uma parte dessas especificações do Java EE, como o Apache Tomcat, que só implementa JSP e Servlets (como dissemos, duas das principais especificações), portanto não é totalmente correto chamá-lo de servidor de aplicação. A partir do Java EE 6, existe o termo “application server web profile”, para poder se referenciar a servidores que não oferecem tudo, mas um grupo menor de especificações, consideradas essenciais para o desenvolvimento web.

Alguns servidores de aplicação conhecidos no mercado:

- RedHat, JBoss Application Server, gratuito;
- Sun (Oracle), GlassFish, gratuito;
- Apache, Apache Geronimo, gratuito;
- Oracle/BEA, WebLogic Application Server;
- IBM, IBM Websphere Application Server;
- Sun (Oracle), Sun Java System Application Server (baseado no GlassFish);
- SAP, SAP Application Serve.

Servlet Container

O Java EE possui várias especificações, entre elas, algumas específicas para lidar com o desenvolvimento de uma aplicação Web:

- JSP
- Servlets
- JSTL
- JSF

Um Servlet Container é um servidor que suporta essas funcionalidades, mas não necessariamente o Java EE completo. É indicado a quem não precisa de tudo do Java EE e está interessado apenas na parte web (boa parte das aplicações de médio porte se encaixa nessa categoria).

Há alguns Servlet Containers famosos no mercado. O mais famoso é o Apache Tomcat, mas há outros como o Jetty.

Referências

Informações retiradas de:

Apostila FJ-21 da Caelum

www.caelum.com.br/apostilas

4 Tecnologia Java Servlets

Quando a Web surgiu, seu objetivo era a troca de conteúdos através, principalmente, de páginas HTML estáticas. Eram arquivos escritos no formato HTML e disponibilizados em servidores para serem acessados nos navegadores. Imagens, animações e outros conteúdos também eram disponibilizados.

Mas logo se viu que a Web tinha um enorme potencial de comunicação e interação além da exibição de simples conteúdos. Para atingir esse novo objetivo, porém, páginas estáticas não seriam suficientes. Era preciso servir páginas HTML geradas dinamicamente baseadas nas requisições dos usuários.

Na plataforma Java, a primeira e principal tecnologia capaz de gerar páginas dinâmicas são as Servlets, que surgiram no ano de 1997. Atualmente o Servlets estão na versão 3 e permite o uso de anotações para definir as configurações.

Como funciona as Servlets

Na tecnologia Servlets se usa a própria linguagem Java para criar páginas web dinâmicas, criando uma classe que terá capacidade de gerar conteúdo HTML. O nome “servlet” vem da ideia de um

pequeno servidor (servidorzinho, em inglês) cujo objetivo é receber chamadas HTTP, processá-las e devolver uma resposta ao cliente.

Uma primeira ideia da servlet seria que cada uma delas é responsável por uma página, sendo que ela lê dados da requisição do cliente e responde com outros dados (uma página HTML, uma imagem GIF etc). Como no Java tentamos sempre que possível trabalhar orientado a objetos, nada mais natural que uma servlet seja representada como um objeto a partir de uma classe Java.

Cada servlet é, portanto, um objeto Java que recebe tais requisições (request) e produz algo (response), como uma página HTML dinamicamente gerada.

O diagrama abaixo mostra três clientes acessando o mesmo servidor através do protocolo HTTP:

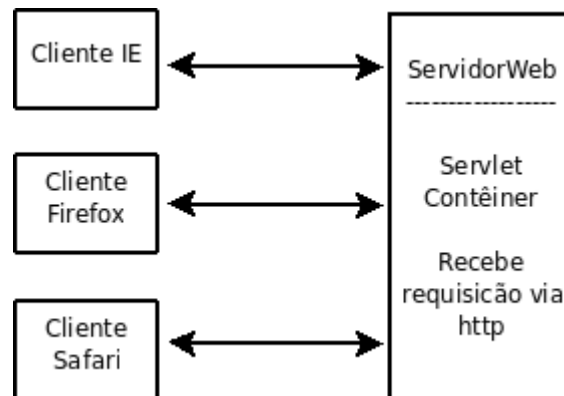


Ilustração 8: Servlets

Criando o primeiro Servlets

Vamos considerar o projeto 'OlaMundoWeb' feito no capítulo 2. Siga os passos a seguir para criar o Servlets que tem o objetivo de exibir uma mensagem na tela de um navegador da Internet.

1. Clique com o botão direito do mouse em 'Pacotes de Códigos-Fonte' e selecione 'Novo->Outros';
2. Em 'Escolher Tipo de Arquivo' defina 'Categorias = Web' e 'Tipos de Arquivos = Servlet' e então clique no botão 'Próximo', como na figura 9:

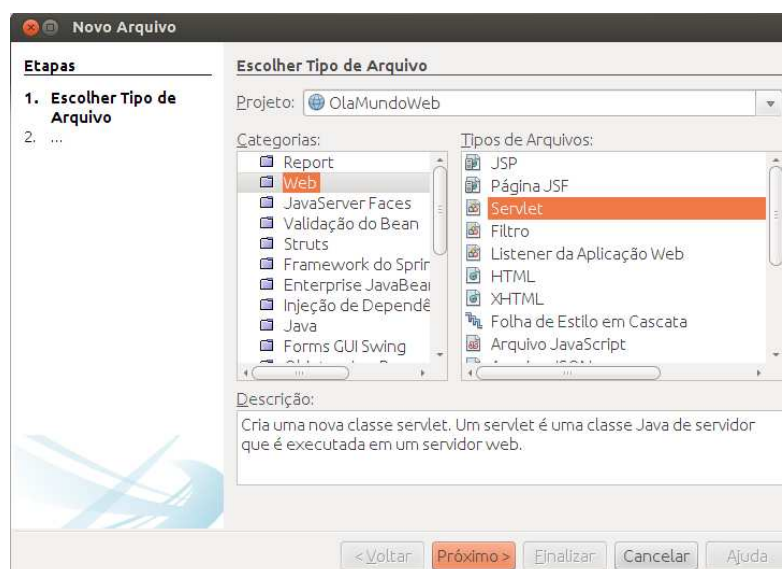


Ilustração 9: Criando Servlet

3. Em 'Nome e Localização' defina os campos conforme a figura 10:

The screenshot shows the 'New Servlet' dialog box with the 'Nome e Localização' tab selected. The 'Etapas' (Steps) list on the left shows three steps: 1. Escolher Tipo de Arquivo, 2. Nome e Localização (highlighted), and 3. Configurar Implantação do Servlet. The main area contains the following fields:

- Nome da Classe:
- Projeto:
- Localização:
- Pacote:
- Arquivo Criado:

At the bottom, there are five buttons: '< Voltar', 'Próximo >', 'Finalizar', 'Cancelar', and 'Ajuda'.

Ilustração 10: Criando Servlet

4. Em 'Configurar Implantação do Servlet' mantenha os valores padrões do formulário e então clique em finalizar, como na figura 11:

The screenshot shows the 'New Servlet' dialog box with the 'Configurar Implantação do Servlet' tab selected. The 'Etapas' (Steps) list on the left shows three steps: 1. Escolher Tipo de Arquivo, 2. Nome e Localização, and 3. Configurar Implantação do Servlet (highlighted). The main area contains the following fields and options:

- Register the Servlet in the application by assigning an internal name to the Servlet (Servlet Name). Then specify the patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.
- ☐ Adicionar informações ao descritor de implantação (web.xml)
- Nome da Classe:
- Nome do Servlet:
- Padrão(ões) de URL:
- Parâmetros de Inicialização:

Below the 'Parâmetros de Inicialização' label, there is a table with two columns: 'Nome' and 'Valor'. To the right of the table are three buttons: 'Novo', 'Editar...', and 'Deletar'.

At the bottom, there are five buttons: '< Voltar', 'Próximo >', 'Finalizar', 'Cancelar', and 'Ajuda'.

Ilustração 11: Criando Servlet

5. Perceba que o NetBeans criou a classe 'OlaMundoServlet' e já definiu alguns métodos. Vamos entender agora o que esta classe implementa.
6. Na imagem 12 vemos a declaração da classe 'OlaMundoServlet' e percebemos que ela herda de 'HttpServlet'. É importante compreender que para uma classe ser um **Servlet** ela precisa herdar de 'HttpServlet'. Na linha 19 a anotação 'WebServlet' define o nome do Servlet e o padrão de URL para acioná-lo.

```
18  /
19  @WebServlet(name = "OlaMundoServlet", urlPatterns = {"/OlaMundoServlet"})
20  public class OlaMundoServlet extends HttpServlet {
21
```

Ilustração 12: Classe OlaMundoServlet

7. Na classe 'HttpServlet' é definido os métodos 'doGet' e 'doPost' que são acionados de acordo com a requisição HTTP (method get e post). No exemplo os dois métodos fazem a mesma coisa, apenas chamam o método 'processRequest'.
8. Acrescente uma linha no método 'processRequest' para que fique como na imagem 13:

```
32  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33      throws ServletException, IOException {
34      response.setContentType("text/html;charset=UTF-8");
35      PrintWriter out = response.getWriter();
36      try {
37          /* TODO output your page here. You may use following sample code. */
38          out.println("<html>");
39          out.println("<head>");
40          out.println("<title>Servlet OlaMundoServlet</title>");
41          out.println("</head>");
42          out.println("<body>");
43          out.println("<h1>Servlet OlaMundoServlet at " + request.getContextPath() + "</h1>");
44          out.println("Página HTML gerada através de um Java Servlet");
45          out.println("</body>");
46          out.println("</html>");
47      } finally {
48          out.close();
49      }
50  }
```

Ilustração 13: Método processRequest

Compreendendo o código do método 'processRequest'

Na linha 32 é definida a assinatura do método, nele são definidos os parâmetros 'request' e 'response' dos tipos 'HttpServletRequest' e 'HttpServletResponse' respectivamente. O primeiro trata-se da requisição HTTP de onde podemos extrair os parâmetros, por exemplo, e o segundo trata-se de um objeto que irá processar a resposta HTTP.

Na linha 35 é definido padrão 'UTF-8' para codificação da resposta.

Na linha 35 é obtido o objeto 'out' do tipo 'PrintWriter' através do método 'getWriter' do 'response', com isso, tudo que for impresso fará parte da resposta.

Nas linhas 38 até 46 é definido o conteúdo de uma página HTML.

9. Execute o projeto. Não exibiu a página definida no servlet? Isso é porque você ainda não acessou a URL correta. Para acessar acrescente na URL 'OlaMundoServlet', como na imagem 14:



Ilustração 14: Página HTML gerada pelo Servlet

10. Se apareceu a tela acima, parabéns, o seu primeiro Servlet foi criado com sucesso.

Exercício

Implemente um novo servlet que exiba na tela o nome de uma pessoa e se ela é maior ou menor de idade. O objetivo é que o nome e a idade sejam passados por parâmetro para o servlet e ele irá gerar uma página HTML exibindo as informações desejadas. Os parâmetros podem ser passados via 'get' ou 'post', como você preferir.

Dica: para recuperar o parâmetro no Servlet utilize o método 'getParameter' definido na classe 'HttpServletRequest'. Por exemplo, para recuperar o parâmetro chamado 'idade' utilize o método da seguinte forma: 'getParameter("idade")'.

Veja os protótipos abaixo, que foram chamados via get, pois, foram passados através da URL da página:



Ilustração 15: Exemplo maior de idade



Ilustração 16: Exemplo menor de idade

Referências

Informações retiradas de:

Apostila FJ-21 da Caelum

www.caelum.com.br/apostilas

5 JavaServer Pages (JSP)

Até agora, vimos que podemos escrever conteúdo dinâmico através de Servlets. No entanto, se toda hora criarmos Servlets para fazermos esse trabalho, teremos muitos problemas na manutenção das nossas páginas e também na legibilidade do nosso código, pois sempre aparece código Java misturado com código HTML. Imagine todo um sistema criado com Servlets fazendo a geração do HTML.

Para não termos que criar todos os nossos conteúdos dinâmicos dentro de classes, misturando fortemente HTML com código Java, precisamos usar uma tecnologia que podemos usar o HTML de forma direta, e que também vá possibilitar a utilização do Java. Algo similar ao ASP e PHP. Essa tecnologia é o JavaServer Pages (JSP) .

Se na tecnologia Servlets você gerava páginas HTML através de código Java, no JSP você escreve código no meio de uma página HTML. Isto é possível através do recurso chamado **Scriptlet** que é o código Java escrito entre `<% e %>` . Esse nome é composto da palavra script (pedaço de código em linguagem de script) com o sufixo let, que indica algo pequeno.

Podemos avançar mais um pouco e utilizar uma das variáveis já implícitas no JSP: todo arquivo JSP já possui uma variável chamada out (do tipo JspWriter) que permite imprimir para o response através do método println:

```
<% out.println(nome); %>
```

A variável out é um objeto implícito na nossa página JSP e existem outras de acordo com a especificação. Repare também que sua funcionalidade é semelhante ao out que utilizávamos nas Servlets mas sem precisarmos declará-lo antes.

Existem ainda outras possibilidades para imprimir o conteúdo da nossa variável: podemos utilizar um atalho (muito parecido, ou igual, a outras linguagens de script para a Web):

```
<%= nome %><br>
```

Verificador de Maioridade em JSP

Para praticar vamos implementar o exercício que verifica se uma pessoa é maior de idade. Só que agora será implementado via página JSP. Siga os passos, considere o projeto 'OlaMundoWeb':

1. Acesse a aba 'Projetos' e clique com o botão direito em 'Páginas Web' e selecione 'Novo->Outros';
2. Em 'Escolher Tipo de Arquivo' defina 'Categorias = Web' e 'Tipos de Arquivos = JSP' e então clique em 'Próximo';
3. Em 'Nome e Localização' defina o 'Nome do Arquivo = maioridade', mantenha os demais campos com o preenchimento padrão e então clique em 'Finalizar', como na figura seguir:

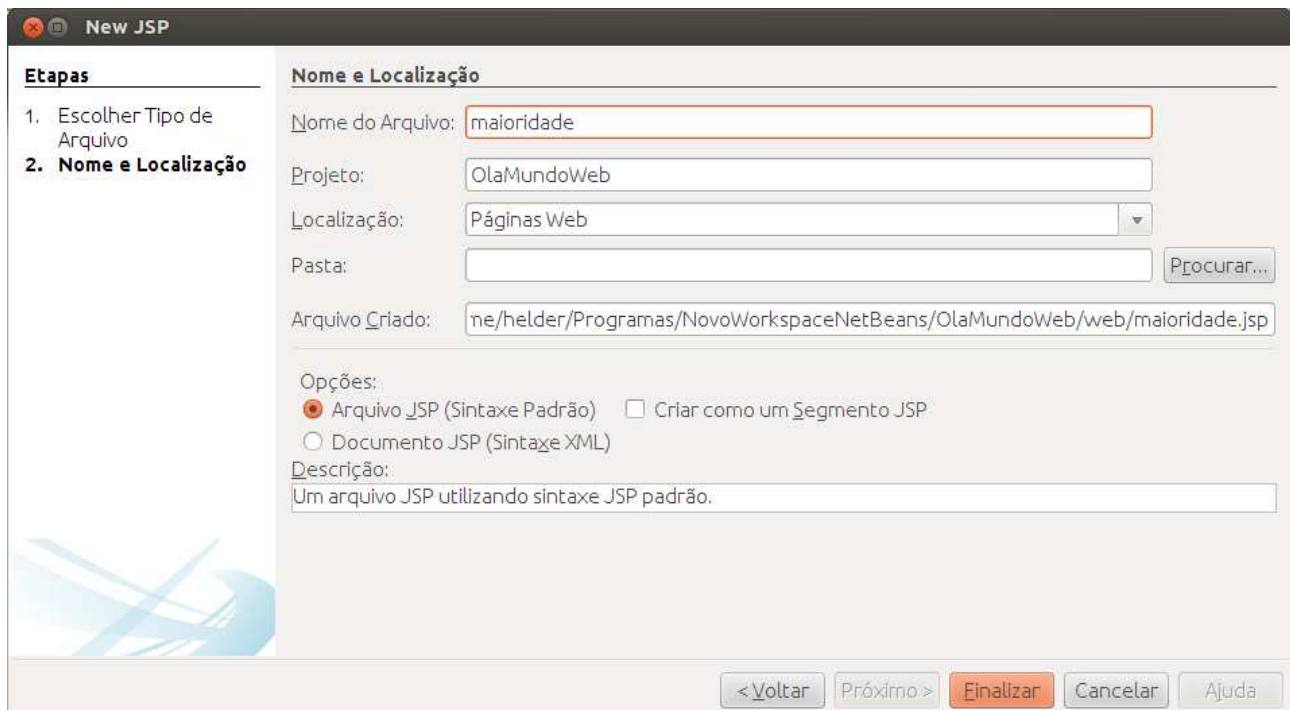


Ilustração 17: Criando arquivo JSP

4. Defina o código para o arquivo 'maioridade.jsp' como na figura 18.

Entendendo o arquivo maioridade.jsp

Este é um arquivo HTML, logo, percebe-se várias tags HTML.

Nas linhas 16 até a linha 31 foi definido um scriptlet que produz um conteúdo dinâmico para informar se uma pessoa é maior de idade ou não.

Nas linhas 18 e 19 se obtém os parâmetros nome e idade através do objeto implícito

'request'.

Nas linhas a seguir são impressos na tela as informações necessárias.

```
7 <%@page contentType="text/html" pageEncoding="UTF-8"%>
8 <!DOCTYPE html>
9 <html>
10 <head>
11 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12 <title>Verificado de Maioridade - JSP</title>
13 </head>
14 <body>
15 <h1>Verificado de Maioridade - JSP</h1>
16 <%
17 //Obtém parâmetros
18 String nome = request.getParameter("nome");
19 String idadeParametro = request.getParameter("idade");
20 int idade = Integer.parseInt(idadeParametro);
21
22 //Escreve o nome da pessoa
23 out.print("Olá "+nome+",<br/>");
24
25 //Verifica e escreve se a pessoa é maior de idade
26 if(idade<18){
27     out.print("Você é menor de idade!");
28 }else{
29     out.print("Você é maior de idade!");
30 }
31 %>
32 </body>
33 </html>
```

Ilustração 18: Arquivo 'maioridade.jsp'

5. Execute o projeto e acesse a URL '<http://localhost:8084/OlaMundoWeb/maioridade.jsp?nome=Josefina&idade=44>' e então verifique se resultou na imagem abaixo:



Ilustração 19: Tela gerada pelo arquivo JSP

6. Teste com outras idades e veja os resultados.

7. Se deu tudo como esperado sua primeira JSP está 100%.

Exercício

Implemente uma página JSP que receba 5 números inteiros como parâmetro e exiba na tela qual o maior número.

Dica: caso você deseje criar um método a parte para encontrar o maior número é necessário implementá-lo em um scriptlet diferenciado que deve ser circundado com '`<%! %>`'. Por exemplo:

```
<%!  
String concatenarNome(String nome, String sobrenome){  
    return nome + sobrenome;  
}  
%>
```

Verifique o protótipo abaixo para este exercício:



Ilustração 20: Protótipo do exercício

Aprofundado mais em JSP

O objetivo das nossas aulas é apenas ter uma noção básica do funcionamento da tecnologia JSP, porém, caso deseje conhecer mais sobre o desenvolvimento de páginas JSP complexas é necessário realizar um estudo aprofundado sobre Expression Language, Taglibs e JSTL.

Referências

Informações retiradas de:

Apostila FJ-21 da Caelum

www.caelum.com.br/apostilas

6 Model View Controller (MVC)

Servlet ou JSP?

Colocar todo HTML dentro de uma Servlet realmente não nos parece a melhor ideia. O que acontece quando precisamos mudar o design da página? O designer não vai saber Java para editar a Servlet, recompilá-la e colocá-la no servidor.

Imagine usar apenas JSP. Ficaríamos com muito scriptlet, que é muito difícil de dar manutenção.

Uma ideia mais interessante é usar o que é bom de cada um dos dois.

O JSP foi feito apenas para apresentar o resultado, e ele não deveria fazer acessos ao banco de dados e nem fazer a instanciação de objetos. Isso deveria estar em código Java, na Servlet.

O ideal então é que a Servlet faça o trabalho árduo, a tal da lógica de negócio. E o JSP apenas apresente visualmente os resultados gerados pela Servlet. A Servlet ficaria então com a lógica de negócios (ou regras de negócio) e o JSP tem a lógica de apresentação.

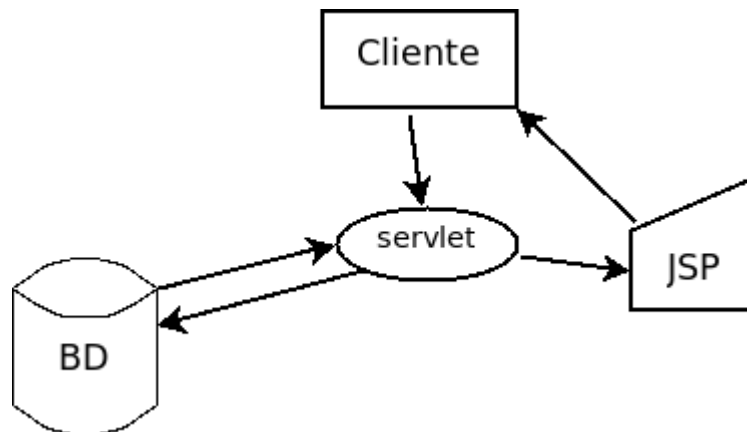


Ilustração 21: Arquitetura MVC

O padrão arquitetural Model View Controller (MVC) foi criado com o objetivo de separar os códigos de tela dos códigos que representam as regras de negócio do sistema em ambiente Web.

Dando nomes a cada uma das partes da arquitetura MVC dizemos que quem é responsável por apresentar os resultados na página web é chamado de Apresentação (View).

A servlet (e auxiliares) que faz os despachos para quem deve executar determinada tarefa é chamada de Controladora (Controller).

As classes que representam suas entidades e as que te ajudam a armazenar e buscar os dados são chamadas de Modelo (Model).

Esses três formam um padrão arquitetural chamado de MVC, ou Model View Controller. Ele pode sofrer variações de diversas maneiras. O que o MVC garante é a separação de tarefas, facilitando assim a reescrita de alguma parte, e a manutenção do código.

Para auxiliar a implementação do padrão MVC vários frameworks Java MVC foram desenvolvidos:

- JSF (especificação padrão do Java EE);
- Struts;
- Vraptor;

- Stripes;
- Jboss Seam;
- Spring MVC;
- Wicket.

Referências

Informações retiradas de:

Apostila FJ-21 da Caelum

www.caelum.com.br/apostilas

7 JavaServer Faces 2.0 (JSF)

Entre os vários frameworks MVC Java, o JavaServer Faces (JSF) é o de maior destaque simplesmente por fazer parte da especificação do Java EE.

O JSF consiste em uma API para representação de componentes e gerência do seu estado, manipulação de eventos, validação no lado do servidor, e conversão de dados; definição de navegação de páginas e suporte a internacionalização. Para isso, ele é baseado no uso de bibliotecas de tags que produzem um ambiente web rico.

O JSF é apenas uma especificação, assim como a máquina virtual Java. Ou seja, nós encontramos no mercado diversas implementações do JSF. Abaixo as implementações mais famosas:

- MyFaces
- ICEfaces
- RichFaces
- Oracle ADF
- PrimeFaces
- OpenFaces

A imagem 22 ilustra o funcionamento básico do JSF. O cliente faz um requisição HTTP ao Web Container que irá processar as regras de negócio em código Java e produzir uma resposta em HTML.

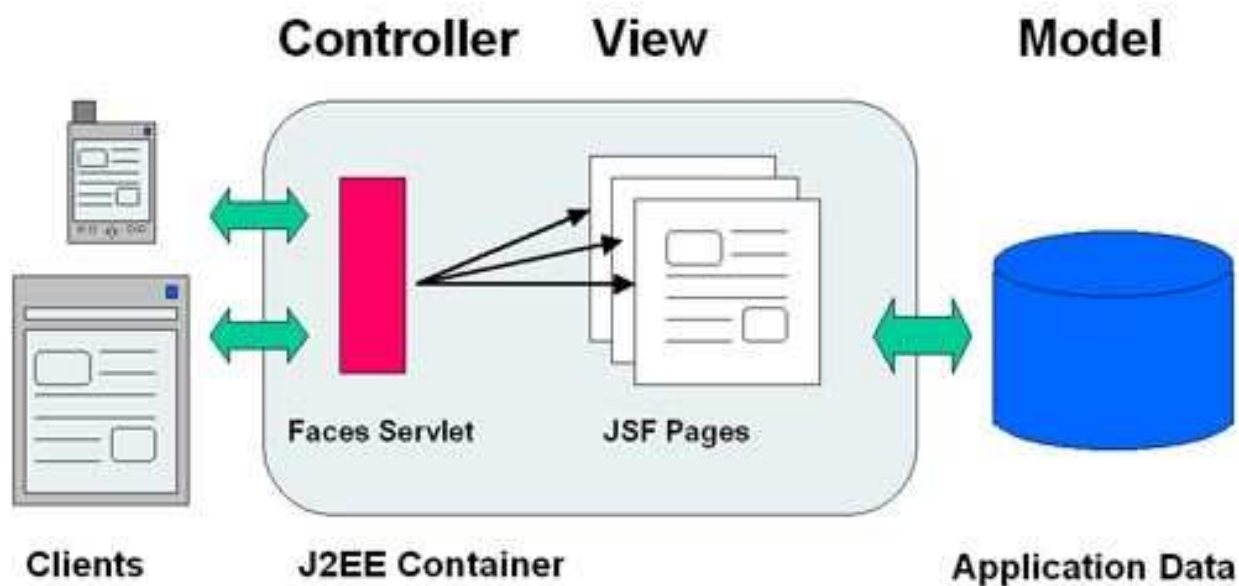


Ilustração 22: Fluxo do JSF

O ciclo de vida de uma requisição JSF é ilustrado na figura 23:

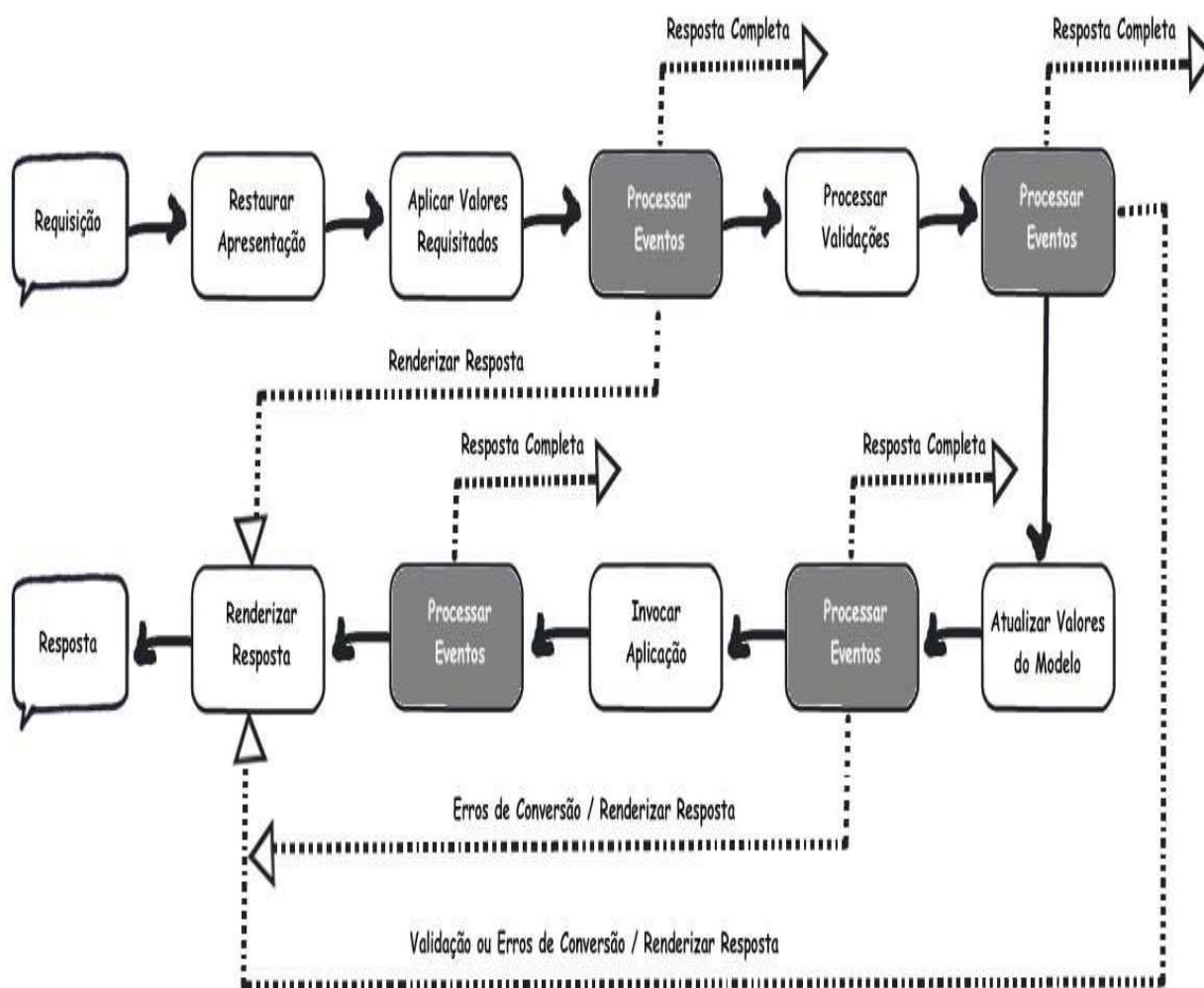


Ilustração 23: Ciclo de vida do JSF

Referências

Informações retiradas de:

The Java EE Tutorial

<http://docs.oracle.com/javaee/6/tutorial/doc/>

8 Introdução ao PrimeFaces

A implementação JSF que será adotada nesta apostila será a PrimeFaces por apresentar um rico conjunto de componentes, Ajax nativo e por ter uma curva de aprendizado mais suave que outras implementações JSF.

A documentação do PrimeFaces pode ser encontrada em:
<http://www.primefaces.org/documentation.html>

No site a seguir é possível visualizar vários componentes do PrimeFaces e sua implementação, acesse: <http://www.primefaces.org/showcase-labs/ui/home.jsf>

Integrando o JSF numa aplicação Web

O NetBeans já vem com suporte ao JSF e PrimeFaces 3.2, portanto, para integrá-los ao nosso projeto atual, 'OlaMundoWeb', será muito simples.

Vamos considerar uma prática dirigida que faça a mesma verificação de maioria que foi feito com Servlet e JSP, só que agora com o JSF. Siga os passos:

1. Acesse a aba 'Projetos', clique com o botão direito no nome do projeto e então selecione 'Propriedades';
2. Em 'Categorias' selecione 'Frameworks' e então clique no botão 'Adicionar';



Ilustração 24: Adicionando Framework

3. Na tela que se abre selecione JavaServer Faces e clique em ok;

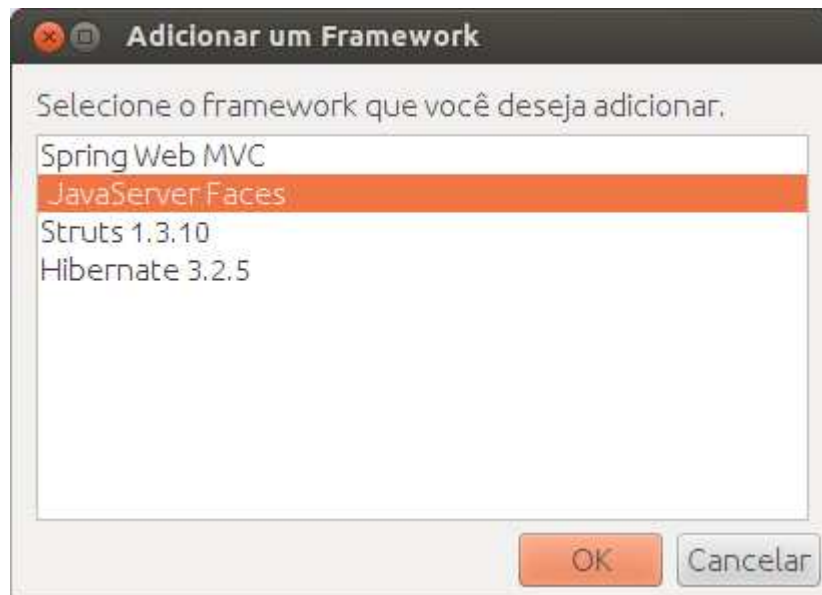


Ilustração 25: Adicionando framework JSF

4. Na aba 'Componentes' selecione o 'PrimeFaces' e clique em 'Ok'.

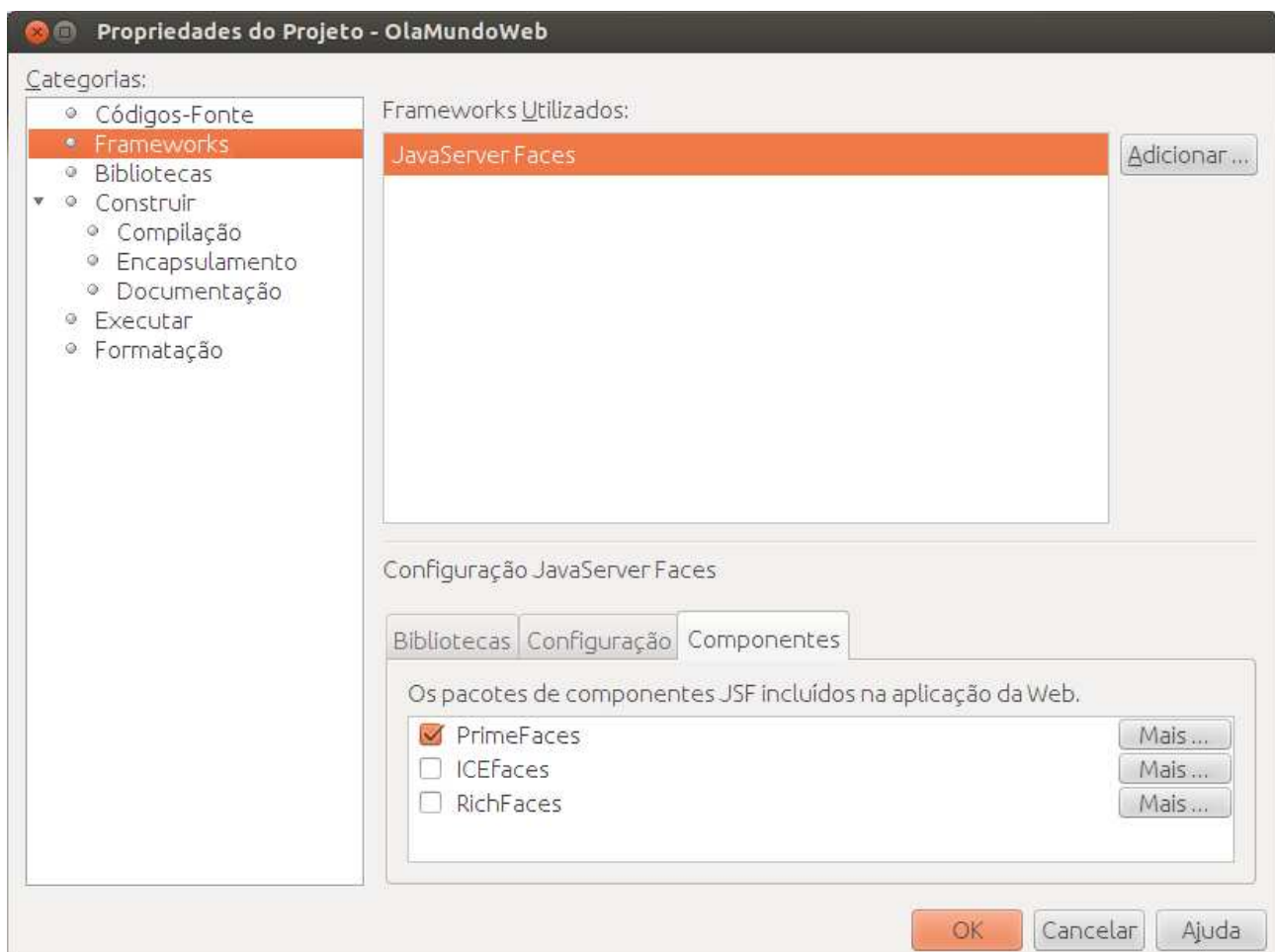


Ilustração 26: Definindo framework PrimeFaces

5. Perceba que foram criados os '/web/index.xhtml', '/web/welcomePrimeFaces.xhtml' e '/web/WEB-INF/web.xml'. Os dois primeiros são exemplos do JSF e PrimeFaces, eles não serão úteis para nós, se deseja pode apagar. Já o '/web/WEB-INF/web.xml' arquivo é de extrema importância e será explicado mais a frente.
6. Vamos criar nosso formulário para entrada de dados, então, clique com o botão direito do mouse em 'Páginas Web' e selecione 'Novo->Outros';
7. Em 'Escolher Tipo de Arquivo' selecione 'Categorias = JavaServer Faces' e 'Tipos de Arquivos = Página JSF', então clique em 'Próximo';

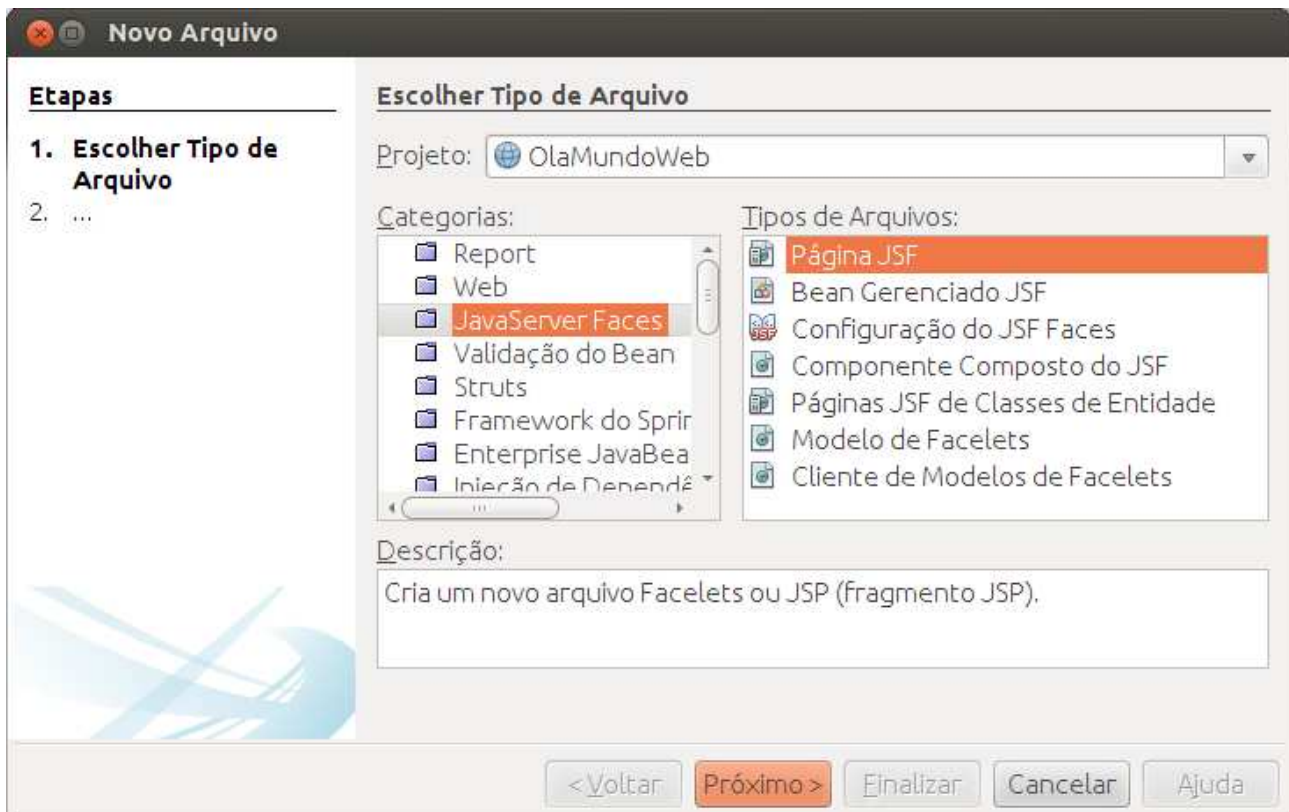


Ilustração 27: Criando página JSF

8. Em 'Nome e Localização' defina o 'Nome do Arquivo = maioridadeJSF', mantenha os demais campos com preenchimento padrão e clique em 'Finalizar';

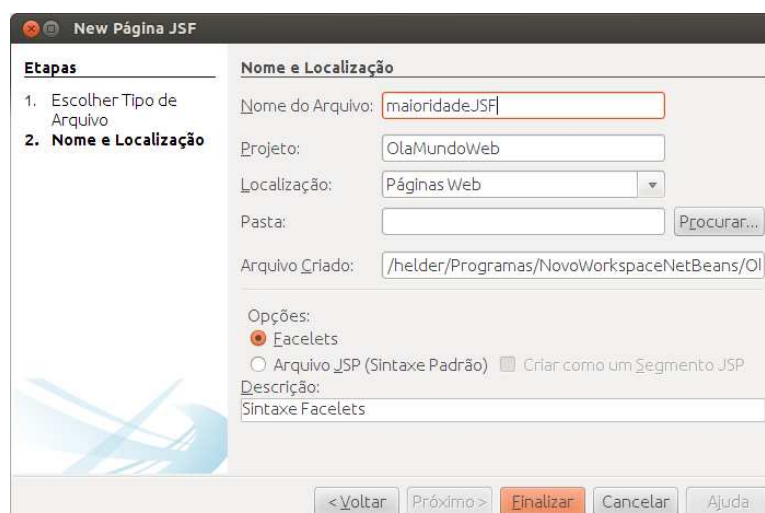


Ilustração 28: Definindo nome do arquivo JSF

9. Perceba que foi criado o arquivo 'maioridadeJSF.xhtml' na pasta 'web'.

```
3 <html xmlns="http://www.w3.org/1999/xhtml"
4       xmlns:h="http://java.sun.com/jsf/html">
5   <h:head>
6       <title>Facelet Title</title>
7   </h:head>
8   <h:body>
9       Hello from Facelets
10  </h:body>
11 </html>
```

Ilustração 29: Arquivo maioridadeJSF.xhtml

Entendendo o arquivo

- Trata-se de um arquivo no formato XHTML, pois, o JSF exige que as páginas estejam neste formato. Para conhecer mais sobre o padrão XHTML acesse: http://www.w3schools.com/html/html_xhtml.asp;
 - Na linha 4 é definido a taglib básica do JSF. O uso desta taglib é percebida em '<h:head>' e '<h:body>';
 - Os arquivos XHTML do JSF segue a estrutura normal de uma página HTML, porém, o conteúdo dinâmico será implementado com taglibs do JSF.
10. Antes de modificarmos o arquivo 'maioridadeJSF.xhtml' precisamos informar que esta será a página inicial do nosso sistema, para isso, faça as modificações abaixo no arquivo 'web.xml';

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:
3   <context-param>
4       <param-name>javax.faces.PROJECT_STAGE</param-name>
5       <param-value>Development</param-value>
6   </context-param>
7   <servlet>
8       <servlet-name>Faces Servlet</servlet-name>
9       <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
10      <load-on-startup>1</load-on-startup>
11  </servlet>
12  <servlet-mapping>
13      <servlet-name>Faces Servlet</servlet-name>
14      <url-pattern>/faces/*</url-pattern>
15  </servlet-mapping>
16  <session-config>
17      <session-timeout>
18          30
19      </session-timeout>
20  </session-config>
21  <welcome-file-list>
22      <welcome-file>faces/maioridadeJSF.xhtml</welcome-file>
23  </welcome-file-list>
24 </web-app>
```

Ilustração 30: Arquivo web.xml

Entendendo o arquivo 'web.xml'

- Este é o arquivo de configuração de uma aplicação Java Web;
- Nas linhas 3 – 6 é definido um parâmetro do sistema que indica que o projeto está no estágio de desenvolvimento;
- Nas linhas 7 – 15 é definido o servlet do JSF e que o acesso às páginas do JSF será feito através do padrão de URL '/faces/*';
- Nas linhas 16 – 20 é configurado o tempo de sessão, neste caso está definido 30 minutos;
- A linha 22 é onde você precisa modificar, ela indica qual é página inicial do sistema.

11. Vamos implementar um formulário que solicita que o usuário informe o nome e idade para verificar se a pessoa possui maioridade. Faça as modificações da acordo com a imagem abaixo:

```
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:h="http://java.sun.com/jsf/html"
5     xmlns:p="http://primefaces.org/ui">
6 <h:head>
7     <title>Verificador Maioridade - JSF</title>
8 </h:head>
9 <h:body>
10 <h:form>
11     <p:panel id="pnlFormulario" header="Verificador Maioridade - JSF">
12         Nome: <p:inputText id="txtNome" label="Nome"/>
13         <br/><br/>
14         Idade: <p:inputText id="txtIdade" label="Idade"/>
15         <br/><br/>
16         <p:commandButton id="btnVerificar" value="Verificar"/>
17     </p:panel>
18 </h:form>
19 </h:body>
20 </html>
```

Ilustração 31: Modificando o arquivo maioridadeJSF.xhtml

Entendendo o arquivo

- Na linha 5 foi adicionado a taglib para os componentes do 'PrimeFaces' representados pelo prefixo 'p';
- Na linha 10 foi definido um 'form' que deve abranger toda a área que onde há requisição;
- Na linha 11 é definido um painel que será renderizado como uma 'div'. Perceba que no atributo 'header' se define um cabeçalho para o painel;
- Nas linhas 12 e 14 são definidos os campos de entrada de texto;
- Na linha 16 é definido um botão.

12. Execute o projeto e veja o resultado. Até agora temos um formulário que ainda não faz nada.

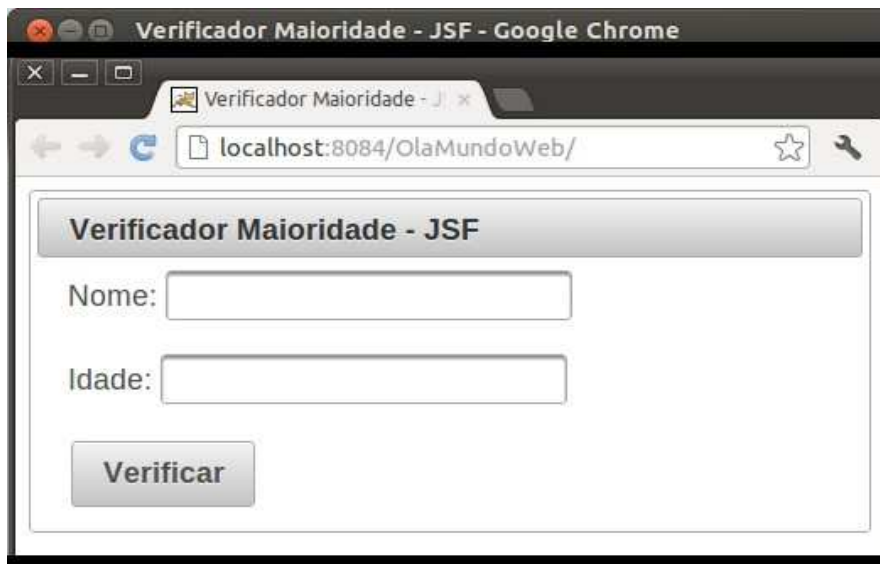


Ilustração 32: Página HTML gerada pelo PrimeFaces

13. Vamos dar vida ao nosso formulário, para isso, precisamos definir o 'Controller' da nossa tela. Observação, o JSF nomeia o objetos que representam o 'Controller' como 'ManagedBeand'.
14. Acesse a aba 'Projetos', clique com o botão direito do mouse no pacote 'olamundoweb' e selecione 'Novo->Outro';
15. Em 'Escolher Tipo de Arquivo' selecione 'Categorias = JavaServer Faces' e 'Tipo de Arquivos = Bean Gerenciado JSF' e clique em 'Próximo';

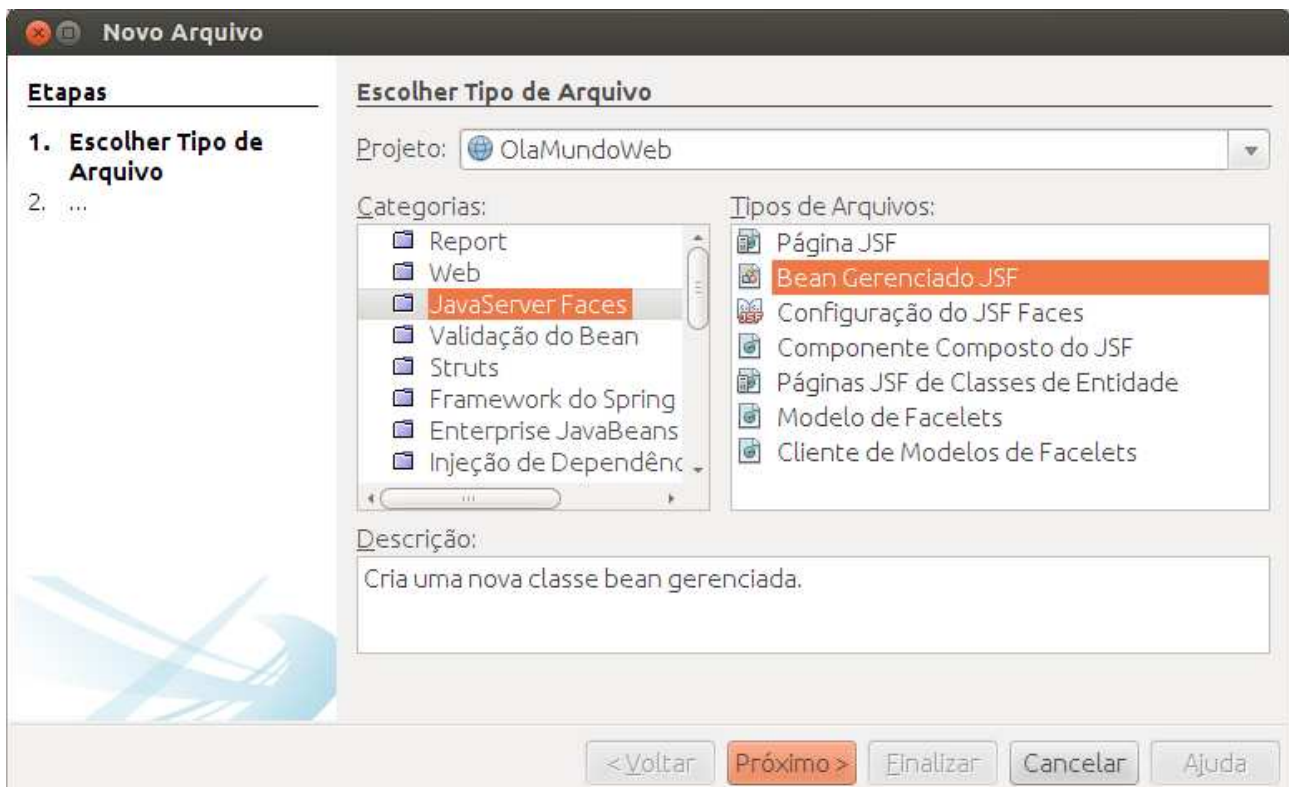


Ilustração 33: Criando classe Controller

16. Em 'Nome e Localização' defina o 'Nome da Classe = MaioridadeController', 'Escopo = session', mantenha os demais campos com o preenchimento padrão e clique em 'Finalizar'.

New Bean Gerenciado JSF

Etapas

1. Escolher Tipo de Arquivo
2. **Nome e Localização**

Nome e Localização

Nome da Classe:

Projeto:

Localização:

Pacote:

Arquivo Criado:

☐ Adicionar dados ao arquivo de configuração

Arquivo de Configuração:

Nome:

Escopo:

Descrição do Bean:

< Voltar Próximo > **Finalizar** Cancelar Ajuda

Ilustração 34: Criando classe Controller

17. O arquivo 'MaioridadeController.java' foi criado, modifique para que fique como na figura 35.

Entendendo a classe MaioridadeController

- Na linha 14 a anotação '[@ManagedBean](#)' define que a classe em questão é um 'Controller' do JSF;
- Na linha 15 a anotação '[@SessionScoped](#)' define que os objetos deste 'Controller' serão mantidos no escopo de sessão;
- Nas linhas 19 e 20 são definidos os atributos que irão mapear os valores dos campos do formulários;
- Nas linhas 22 e 23 são definidos atributos que irão auxiliar na exibição do resultado da verificação;
- Na linha 30 é definido, no construtor da classe, que inicialmente não é para ser exibido o resultado;
- Nas linhas 36 – 44 é disparado o evento que contém a lógica de negócio.

```

13  //
14  @ManagedBean
15  @SessionScoped
16  public class MaioridadeController {
17
18      // Campos do formulário
19      private String nome;
20      private Integer idade;
21
22      // Atributos auxiliares para produzir o resultado
23      private Boolean maior;
24      private Boolean exibirResultado;
25
26      /**
27       * Método construtor
28       */
29      public MaioridadeController() {
30          this.exibirResultado = false;
31      }
32
33      /**
34       * Método acionado pelo botão 'Verificar'
35       */
36      public void verificar(){
37          if(this.idade<18){
38              maior = false;
39          }else{
40              maior=true;
41          }
42
43          this.exibirResultado = true;
44      }
45
46      //Get e set ...

```

Ilustração 35: Classe MaioridadeController

18. O arquivo 'maioridadeJSF.xhtml' também precisa ser modificado para dar vida a tela. Faça as modificações conforme a figura 36.

Entendendo as modificações no arquivo

- Nas linhas 13 e 15 são definidos os atributos 'value' para os campos nome e idade. Agora o valor dos campos estão amarrados com os atributos do 'Controller';
- Nas linhas 17 e 18 foi definido a ação que será executada quando o usuário clicar no botão, através do atributo 'actionListener' que está vinculado ao método 'verificar' do 'Controller'. O atributo 'update' define qual parte da tela que será atualizada após o clique no botão, neste caso o painel 'pnlResultado';
- Nas linhas 20 – 26 contém o painel que irá exibir o resultado da verificação da idade;
- A taglib '<c:if >' é um utilitário para customizar o conteúdo que será exibido na tela;

- Nas linhas 23 e 24 se vê o atributo 'rendered' que está presente em todos os componentes visuais do JSF, trata-se apenas de um teste booleano para verificar se o componente deve ser renderizado ou não.

```

10 </h:form>
11 <h:form>
12   <p:panel id="pnlFormulario" header="Verificador Maioridade - JSF">
13     Nome: <p:inputText id="txtNome" label="Nome" value="#{maioridadeController.nome}" />
14     <br/><br/>
15     Idade: <p:inputText id="txtIdade" label="Idade" value="#{maioridadeController.idade}" />
16     <br/><br/>
17     <p:commandButton id="btnVerificar" value="Verificar" |
18       actionListener="#{maioridadeController.verificar}" update="pnlResultado"/>
19   </p:panel>
20   <p:panel id="pnlResultado" header="Resultado da Verificação">
21     <c:if test="#{maioridadeController.exibirResultado}">
22       Olá #{maioridadeController.nome},<br/>
23       <h:outputLabel value="Você é maior de idade!" rendered="#{maioridadeController.maior}" />
24       <h:outputLabel value="Você é menor de idade!" rendered="#{!maioridadeController.maior}" />
25     </c:if>
26   </p:panel>
27 </h:form>

```

Ilustração 36: Modificando o arquivo maioridadeJSF.xhtml

19. Execute o projeto e teste;

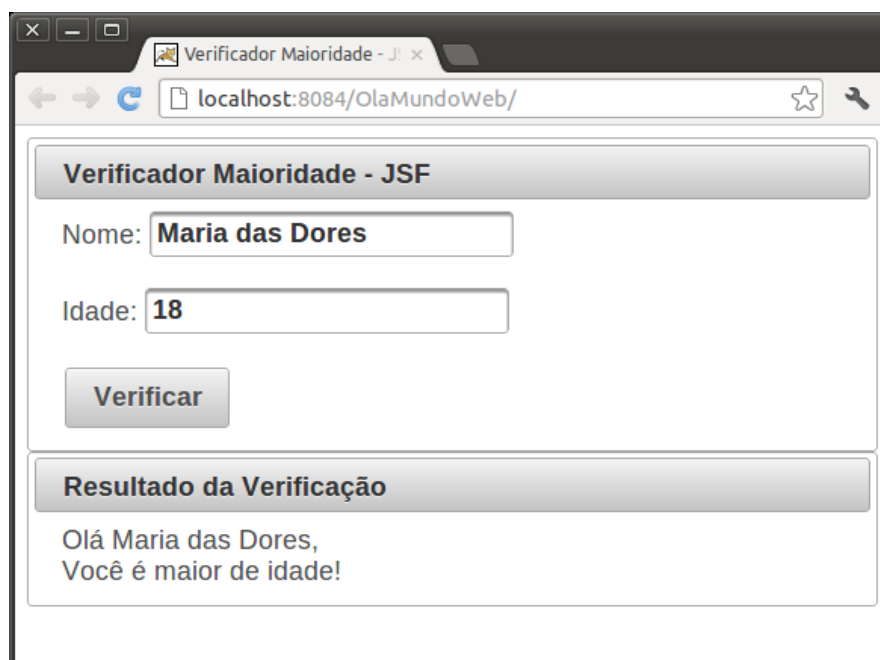


Ilustração 37: Testando implementação

20. Muito bom, muito legal e funcional. Mas, os campos nome e idade deveriam ser obrigatórios. Como fazer isso através do JSF? Faça as modificações destacadas em vermelho no arquivo 'maioridadeJSF.xhtml', conforme a imagem 38.

```
12 <p:panel id="pnlFormulario" header="Verificador Maioridade - JSF">
13   <p:messages autoUpdate="true"/>
14   Nome: <p:inputText id="txtNome" label="Nome" value="#{maioridadeController.nome}"
15         <u>required="true"/>
16   <br/><br/>
17   Idade: <p:inputText id="txtIdade" label="Idade" value="#{maioridadeController.idade}"
18         <u>required="true"/>
19   <br/><br/>
20   <p:commandButton id="btnVerificar" value="Verificar"
21         actionListener="#{maioridadeController.verificar}" update="pnlResultado"/>
22 </p:panel>
23 <p:panel id="pnlResultado" header="Resultado da Verificação">
24   <c:if test="#{maioridadeController.exibirResultado}">
25     Olá #{maioridadeController.nome}, <br/>
26     <h:outputLabel value="Você é maior de idade!" rendered="#{maioridadeController.maior}"/>
27     <h:outputLabel value="Você é menor de idade!" rendered="#{!maioridadeController.maior}"/>
28   </c:if>
29 </p:panel>
```

Ilustração 38: Modificando o arquivo maioridadeJSF.xhtml

Entendendo as modificações no arquivo


- Na linha 13 é definido um componente de mensagem que terá seu conteúdo atualizado automaticamente;
- Nas linhas 15 e 18 são definidos que os campos nome e idade são requeridos.

21. Teste e veja os resultados.



Ilustração 39: Testando a validação dos campos

22. Outra validação que você pode testar é quanto a idade aceitar somente inteiros. Automaticamente o JSF faz a conversão da String do campo do formulário para o tipo do atributo que está vinculado. Tente colocar uma String para idade e veja que já existe uma validação padrão.



Verificador Maioridade - JSF

Idade: 'abc' deve ser um número formado por um ou mais dígitos.

Nome: Marieta

Idade: abc

Verificar

Resultado da Verificação

Ilustração 40: Testando validação de conversão de valores

23. Chegamos ao fim desta primeira prática do JSF. O que achou? Difícil! Sim, mas é muito poderoso para o desenvolvimento de sistemas Web. Acostume, só estamos começando.

9 Desenvolvendo um CRUD com PrimeFaces

Este capítulo tem o objetivo de ser um guia dirigido para a construção de uma aplicação complexa com o JSF PrimeFaces além de praticar a integração de uma aplicação Java EE com o banco de dados via JPA. Nesta prática dirigida será considerado a boa prática da separação das camadas de negócio e persistência, além de colocar em prática o padrão arquitetural MVC.

Deseja-se um sistema que faça o cadastro de clientes de uma empresa. Os dados do cliente que se deseja persistir são nome, cpf, sexo, data de nascimento, cidade, salário, telefone e email. Apenas os campos salário, telefone e e-mail não são obrigatórios. Deve-se considerar que não se cadastra clientes menores de idade.

Vale ressaltar que a empresa em questão possui lojas apenas nas cidades: Januária-MG, Montes

Claros-MG, Volta Redonda-RJ, Rio de Janeiro-RJ, São Caetano-SP e Campinas-SP; os clientes estão vinculados somente a estas cidades.

Veja os protótipos a seguir para se orientar na construção do sistema:



Ilustração 41: Tela inicial do sistema

Ilustração 42: Tela de cadastro de clientes

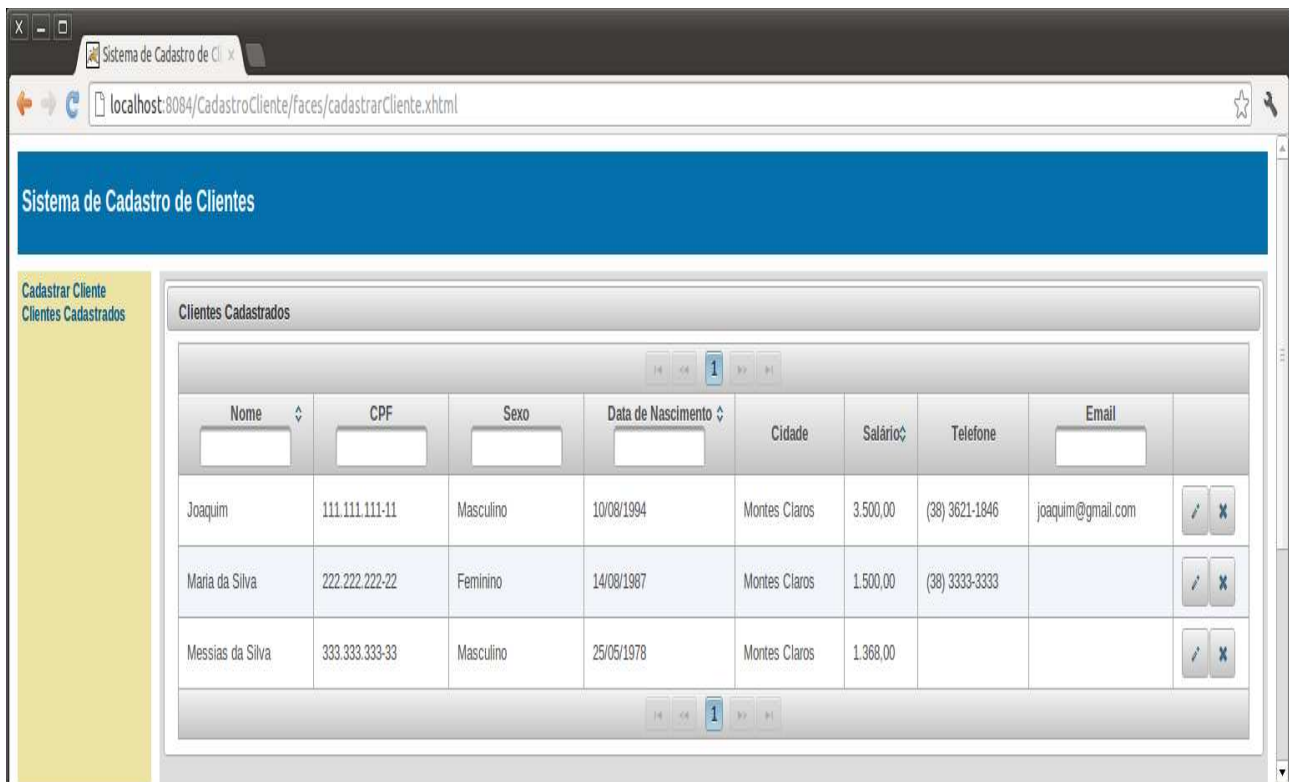


Ilustração 43: Tela de clientes cadastrados

Siga os passos:

- 1 Crie um novo Projeto 'JavaWeb' com suporte ao framework JSF PrimeFaces. Neste guia será considerado um projeto com nome 'CadastroCliente'. Na tela 'Servidor e Definições' marque a opção 'Ativar injeção de contextos e dependências';

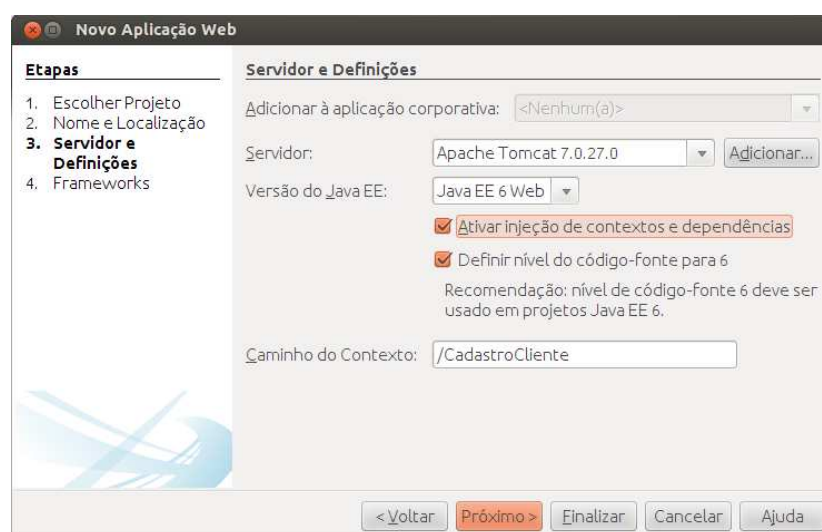


Ilustração 44: Criando projeto Web

- 2 Após criar o projeto, os arquivos 'index.xhtml' e 'welcomePrimfaces.xhtml' podem ser apagados;
- 3 Crie um banco de dados. Neste guia será considerando o banco de dados Derby. A conexão ficou 'jdbc:derby://localhost:1527/CADASTRO_CLIENTE' com usuário 'root' e senha 'root';
- 4 Adicione o driver JDBC na pasta '/web/WEB-INF/lib';

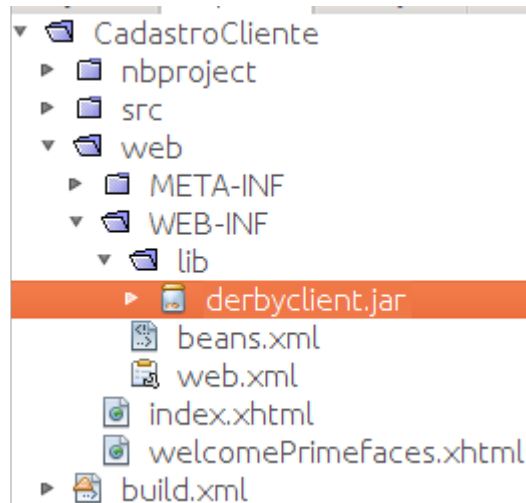


Ilustração 45: Driver JDBC na pasta web/WEB-INF/lib

- 5 Crie uma unidade de persistência para o banco de dados criado;
- 6 Crie os seguintes pacotes para separar os códigos das camadas do sistema:

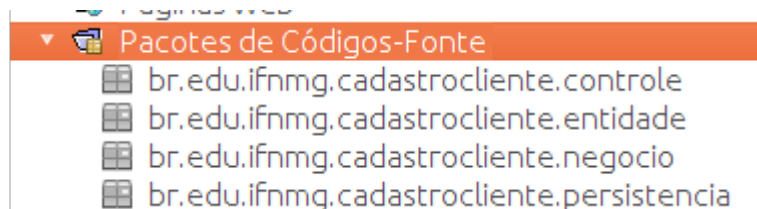
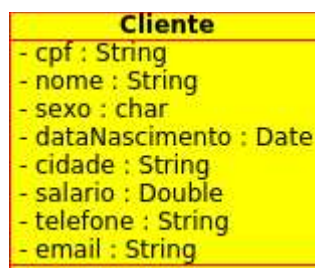


Ilustração 46: Estrutura de diretórios

- 7 Faça uma análise do domínio da aplicação, ou seja, as classes que representam as entidades do sistema. Quais classes você identificou? No meu caso identifiquei somente a classe Cliente ilustrada a seguir:



*Ilustração 47:
Diagrama de classe do projeto*

8 Implemente a classe de entidade 'Cliente';

```
18  @Entity
19  public class Cliente implements Serializable {
20
21      @Id
22      @Column(length=14)
23      private String cpf;
24      @Column(nullable=false)
25      private String nome;
26      @Column(nullable=false)
27      private Character sexo;
28      @Temporal(javax.persistence.TemporalType.DATE)
29      @Column(nullable=false)
30      private Date dataNascimento;
31      @Column(nullable=false)
32      private String cidade;
33      @Column(nullable=true)
34      private Double salario;
35      @Column(nullable=true)
36      private String telefone;
37      @Column(nullable=true)
38      private String email;
```

Ilustração 48: Classe de entidade Cliente

9 **[CRIANDO TEMPLATE]** Vamos criar um template padrão para todas as telas do sistema, para isso, existe o conceito de Facelets no JSF. Desejamos um template em que tenha um cabeçalho, painel lateral esquerdo e um painel para os conteúdos na área central. Siga os passos:

- 9.1 Na aba 'Projetos' clique com o botão direito do mouse no nome do projeto, e selecione 'Novo->Outros';
- 9.2 Em 'Escolher Tipo de Arquivo' selecione 'Categorias = JavaServer Faces' e 'Tipos de Arquivos = Modelo de Facelets', e então clique em 'Próximo';
- 9.3 Em 'Nome e Localização' defina 'Nome do Arquivo = template' e o 'Estilo de Layout' correspondente, conforme imagem 49:

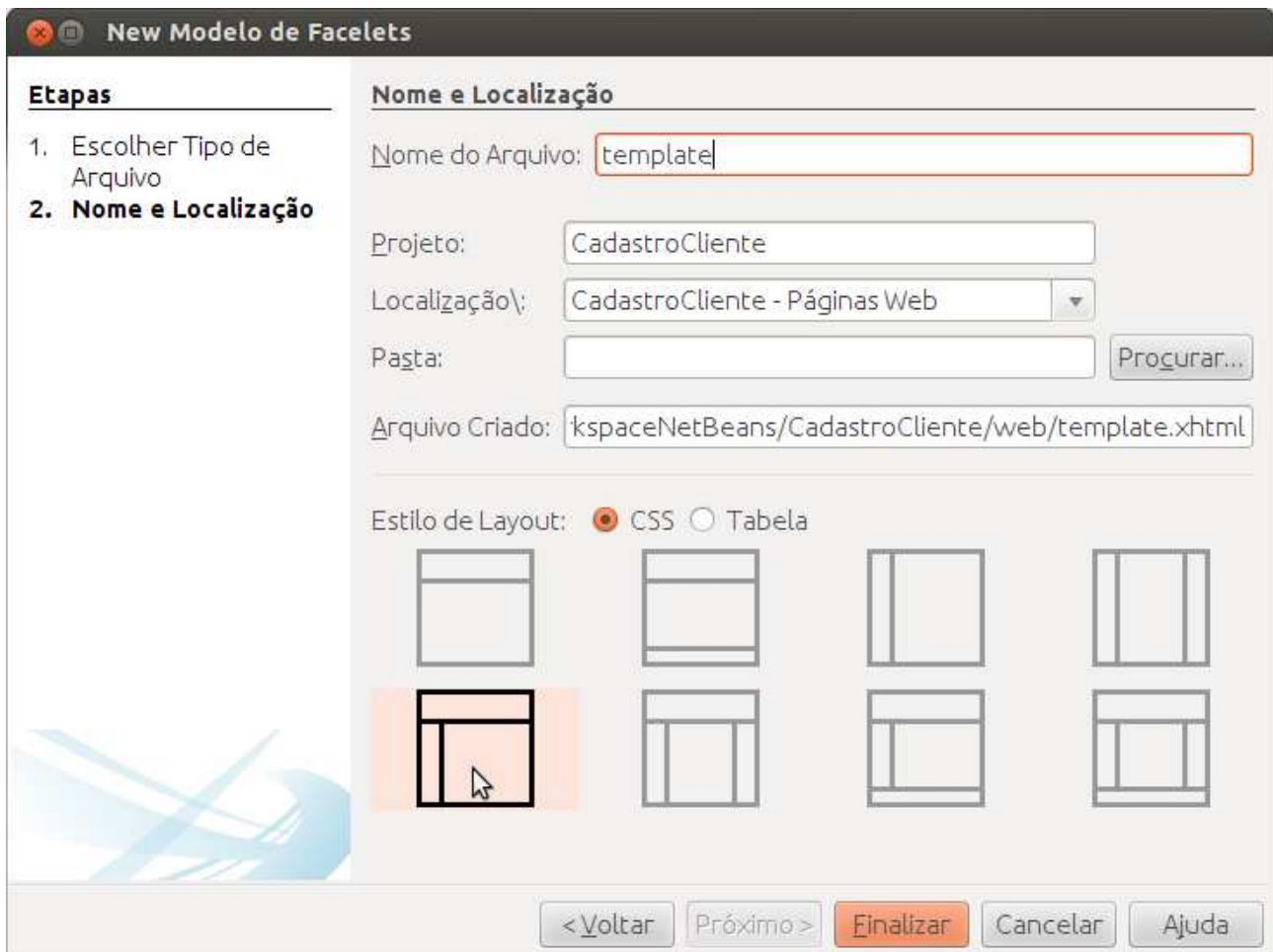


Ilustração 49: Criando template de telas

9.4 Foi criado o arquivo 'template.xhtml'. Faça as modificações conforme imagem 50.

Entendendo o arquivo 'template.xhtml'

- Este arquivo será o modelo para todas as outras tela do sistema;
- Nas linhas 8 – 13 é definido o cabeçalho da página. Veja que foram criados arquivos CSS padrões que você pode modificar.
- Nas linhas 16 – 18 é definido o espaço reservado para a parte superior da tela. Observe que a tag '<ui:insert name="top">' define um módulo que pode ser utilizado pelas telas que implementarem o template.
- Nas linhas 22 – 27 é definido a barra lateral esquerda do template. A tag '<p:commandLink >' define um Link que redireciona para outras páginas.
- Nas linhas 29 – 31 é definido o espaço reservado para o conteúdo das páginas.

```

3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:ui="http://java.sun.com/jsf/facelets"
5     xmlns:h="http://java.sun.com/jsf/html"
6     xmlns:p="http://primefaces.org/ui"
7     xmlns:f="http://java.sun.com/jsf/core">
8 <h:head>
9     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
10    <link href="./resources/css/default.css" rel="stylesheet" type="text/css" />
11    <link href="./resources/css/cssLayout.css" rel="stylesheet" type="text/css" />
12    <title>Sistema de Cadastro de Clientes</title>
13 </h:head>
14 <h:body>
15     <div id="top" class="top">
16         <ui:insert name="top">
17             <h2>Sistema de Cadastro de Clientes</h2>
18         </ui:insert>
19     </div>
20     <div>
21         <div id="left" style="height: 500px;">
22             <ui:insert name="left">
23                 <h:form id="frmMenu">
24                     <p:commandLink value="Cadastrar Cliente" /><br/>
25                     <p:commandLink value="Clientes Cadastrados" />
26                 </h:form>
27             </ui:insert>
28         </div>
29         <div id="content" class="left_content" style="height: 500px;" >
30             <ui:insert name="content">Content</ui:insert>
31         </div>
32     </div>
33 </h:body>
34 </html>

```

Ilustração 50: Template das telas do projeto

- 10 **[CRIANDO TELA INICIAL]** Agora que temos o template criado vamos criar nossa primeira página que implementa o template. O objetivo é criar a tela inicial do sistema, a tela index. Para isso siga os passos:
 - 10.1 Acesse a aba 'Projetos' e clique com o botão direito do mouse no nome do projeto e selecione 'Novo->Outros';
 - 10.2 Em 'Escolher Tipo de Arquivo' selecione 'Categorias = JavaServer Faces' e 'Tipos de Arquivos = Cliente de Modelos de Facelets', e então clique em 'Próximo';

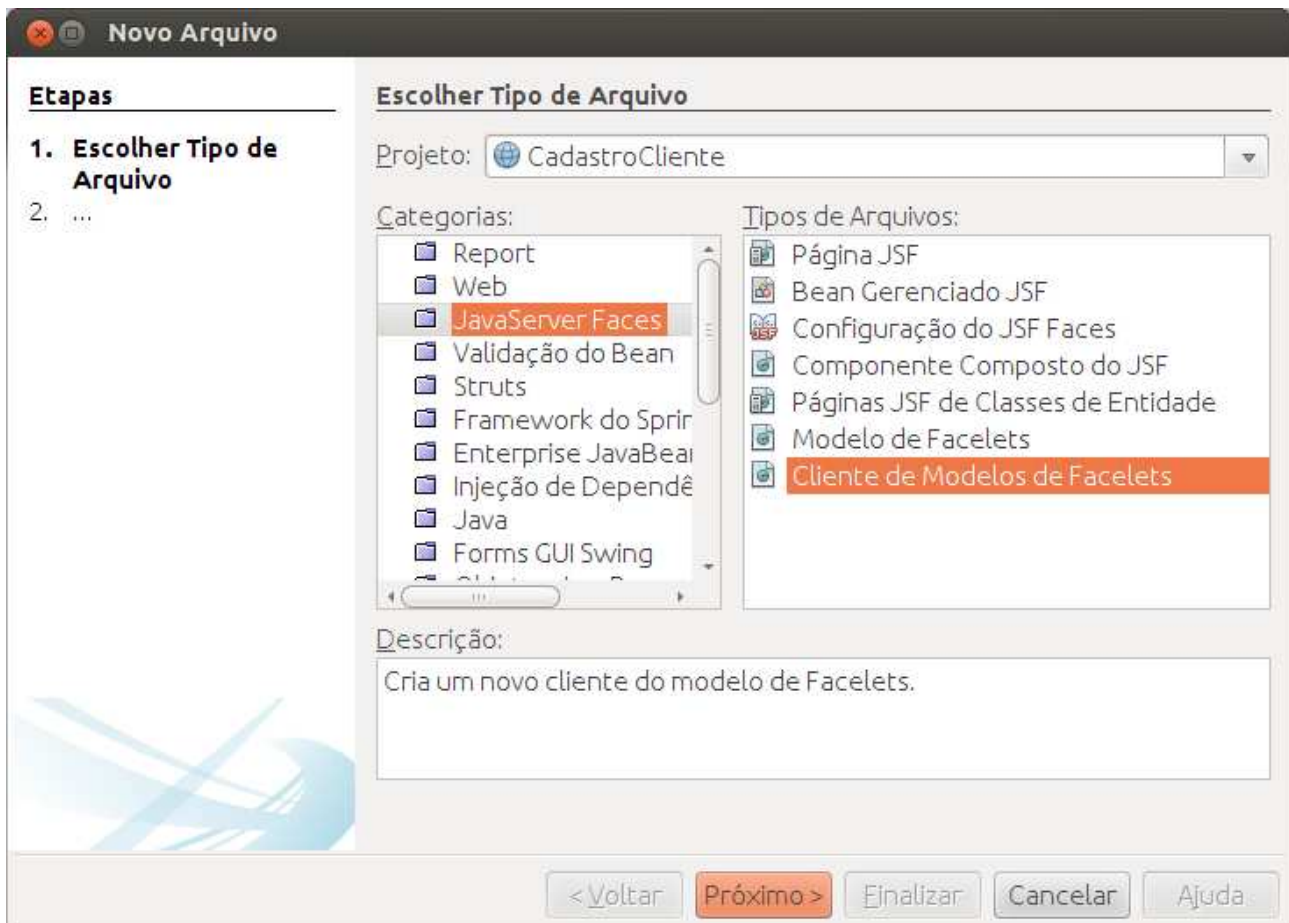


Ilustração 51: Criando tela que utiliza o template

- 10.3 Em 'Nome e Localização' defina 'Nome do Arquivo = index', em 'Modelo' escolha o arquivo 'template.xhtml' e então clique em 'Finalizar';



Ilustração 52: Criando index.xhtml

10.4 Modifique o arquivo 'index.xhtml' conforme a imagem abaixo:

```
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:ui="http://java.sun.com/jsf/facelets"
5     xmlns:h="http://java.sun.com/jsf/html"
6     xmlns:p="http://primefaces.org/ui"
7     xmlns:f="http://java.sun.com/jsf/core">
8     <body>
9         <ui:composition template="./template.xhtml">
10             <ui:define name="content">
11                 Seja bem vindo ao sistema de cadastro de clientes.<br/>
12                 Acesse as funcionalidade através do menu ao lado.
13             </ui:define>
14         </ui:composition>
15     </body>
16 </html>
```

Ilustração 53: Arquivo index.xhtml

Entendendo o arquivo 'index.xhtml'

- Este arquivo será a página inicial do sistema, ele aproveita o modelo do arquivo 'template.xhtml' e só é definido o conteúdo específico;
- Na linha 9 é definido que este arquivo tem como referência o 'template.xhtml';
- Na linha 10 é definido o conteúdo que irá ocupar a área 'content', ou seja, a área que fica o conteúdo da página.

10.5 Execute e veja o resultado:

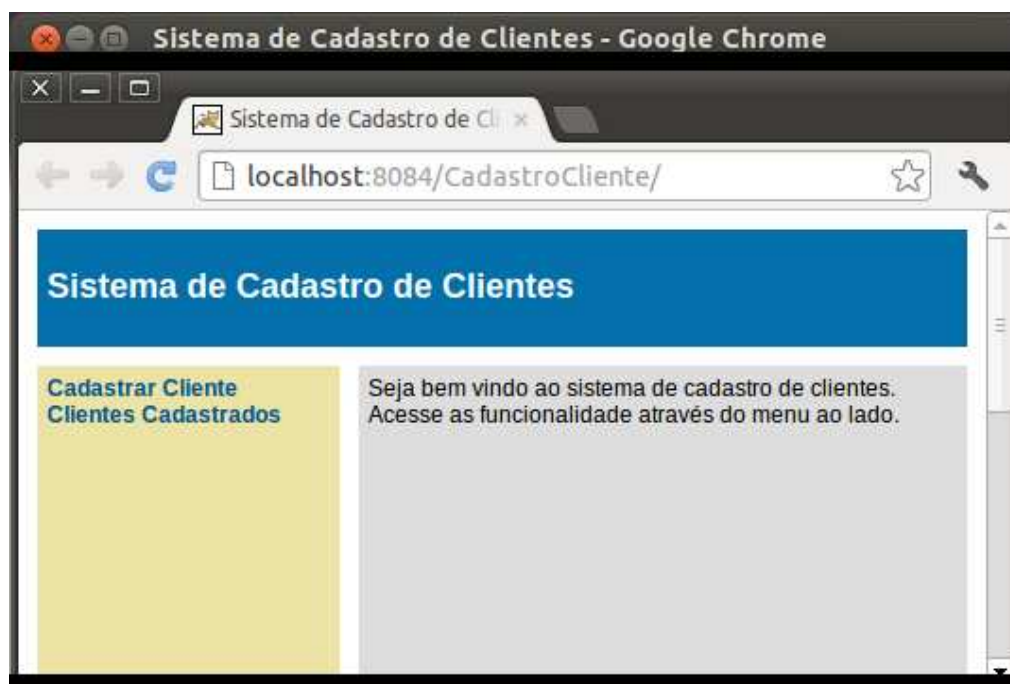


Ilustração 54: Tela inicial do projeto

11 **[TELA CADASTRO CLIENTES]** Esta será a tela que terá a entrada dos dados do cliente. Para isso crie uma nova tela como visto na criação da tela 'index.xhtml', crie com o nome 'cadastroCliente.xhtml';

11.1 Crie também a classe Java que será o 'Controller' desta tela, crie com o nome 'CadastroClienteController.java' no pacote 'br.edu.ifnmg.cadastrocliente.controle';

12 **[Campo Nome]** Este guia será feito em pequenos passos. O ideal é que a cada campo criado o projeto seja executado e que se verifique e compreenda o resultado da modificação. Portanto, faça as modificações a seguir:

12.1 Modifique a classe 'Controller' conforme imagem 55.

```
20 @ManagedBean
21 @SessionScoped
22 public class CadastroClienteController implements Serializable {
23
24     // Objeto que será usado para mapear os campos da tela
25     private Cliente cliente;
26
27     /**
28      * Método executado para iniciar a tela;
29      * Basicamente instancia um objeto de cliente para limpar os campos.
30      */
31     public String iniciarTela() {
32         this.cliente = new Cliente();
33         return "cadastrarCliente";
34     }
35
36     /**
37      * Método chamado pelo botão salvar.
38      */
39     public void salvar() {
40         // Apenas exibe uma mensagem de teste
41         FacesContext facesContext = FacesContext.getCurrentInstance();
42         facesContext.addMessage(null, new FacesMessage("Teste... Passou pelo controller!"));
43     }
44
45     /**
46      * Método usado para validar campos com espaço em branco.
47      */
48     public void validarEspacoBranco(FacesContext contexto, UIComponent componente, Object valor) {
49         String valorString = (String) valor;
50         if (valorString.trim().equals("")) {
51             ((UIInput) componente).setValid(false);
52             String mensagem = componente.getAttributes().get("label")
53                 + ": Valor inválido, preencha com caracteres diferentes de espaço.";
54             FacesMessage facesMessage = new FacesMessage(FacesMessage.SEVERITY_ERROR, mensagem, mensagem);
55             contexto.addMessage(componente.getClientId(contexto), facesMessage);
56         }
57     }
58
59     //Get e set
60 }
```

Ilustração 55: Classe CadastroClienteController

Entendendo a classe CadastroClienteController

- Este é o 'Controller' para a tela 'cadastroCliente.xhtml';
- Na linha 25 é declarado um objeto da classe 'Cliente' que irá se vincular com os campos da tela;
- Nas linhas 31 – 34 é definido o método que irá carregar a tela e conduzir a tela de cadastro de clientes. A 'String' retornada se refere página que deve ser carregada;
- Nas linhas 39 – 43 é definido o método 'salvar' que será invocado com o clique no botão 'Salvar'. Esta ainda é uma implementação somente para teste, para se verificar se a aplicação está funcionando;
- Nas linhas 48 – 57 é definido um método para validar se o campo foi preenchido sem espaços em branco.

12.2 Modifique o arquivo 'cadastroCliente.xhtml' como na figura 56:

```
3 <html xmlns="http://www.w3.org/1999/xhtml"
4       xmlns:ui="http://java.sun.com/jsf/facelets"
5       xmlns:p="http://primefaces.org/ui"
6       xmlns:h="http://java.sun.com/jsf/html"
7       xmlns:f="http://java.sun.com/jsf/core">
8 <body>
9     <ui:composition template="./template.xhtml">
10         <ui:define name="content">
11             <h:form id="frmCadastroCliente">
12                 <p:panel id="pnlFormulario" header="Cadastro de Cliente">
13                     <p:messages autoUpdate="true"/>
14
15                     Nome*:
16                     <p:inputText value="#{cadastroClienteController.cliente.nome}" required="true" label="Nome"
17                                 style="width: 400px" validator="#{cadastroClienteController.validarEspacoBranco}" />
18                 </p:inputText>
19                 <br/>
20
21                 <br/>
22                 <p:panel id="pnlRodape">
23                     <p:commandButton value="Salvar" action="#{cadastroClienteController.salvar}" />
24                     <p:commandButton value="Limpar" type="reset" />
25                 </p:panel>
26             </h:form>
27         </ui:define>
28     </ui:composition>
29 </body>
30 </html>
```

Ilustração 56: Arquivo cadastroCliente.xhtml

Entendendo o arquivo 'cadastroCliente.xhtml'

- Na linha 15 – 18 é definido o campo 'Nome' do tipo '<p:inputText >' foi definido como 'required' e com a validação de espaço em branco 'validator'.
- Nas linhas 23 e 24 são definidos os botões para ação de salvar e limpar a tela.

12.3 Modificações no arquivo 'template.xhtml'. Vamos precisar modificar como o menu chama a tela de cadastro de clientes, para isso, faça as modificações destacadas na imagem 57.

```
<div id="left" style="height: 500px;">
  <ui:insert name="left">
    <h:form id="frmMenu">
      <p:commandLink value="Cadastrar Cliente" ajax="false"
        action="#{cadastroClienteController.iniciarTela}" /><br/>
      <p:commandLink value="Clientes Cadastrados" />
    </h:form>
  </ui:insert>
</div>
```

Ilustração 57: Modificando o arquivo template.xhtml

Entendendo a modificação

- Modifique apenas o 'commandLink' que aciona a tela de cadastro de clientes. É importante definir a mudança de tela como 'ajax=false' para evitar problemas de navegação futura. Neste caso o método 'iniciarTela' do 'Controller' será acionado.

12.4 Execute o projeto e faça testes:

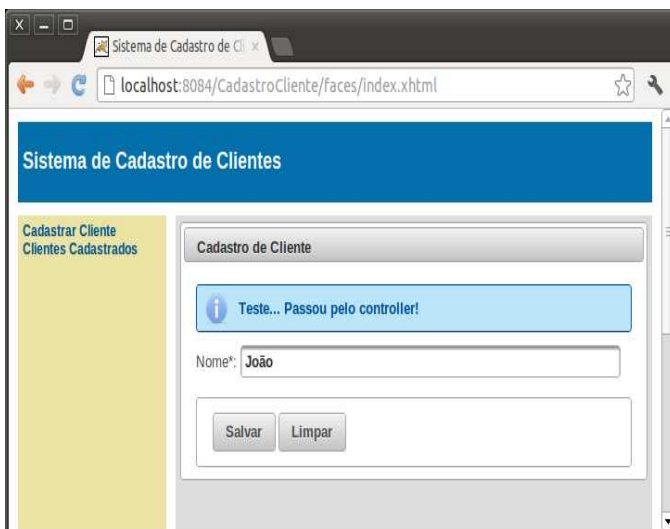


Ilustração 59: Teste com sucesso

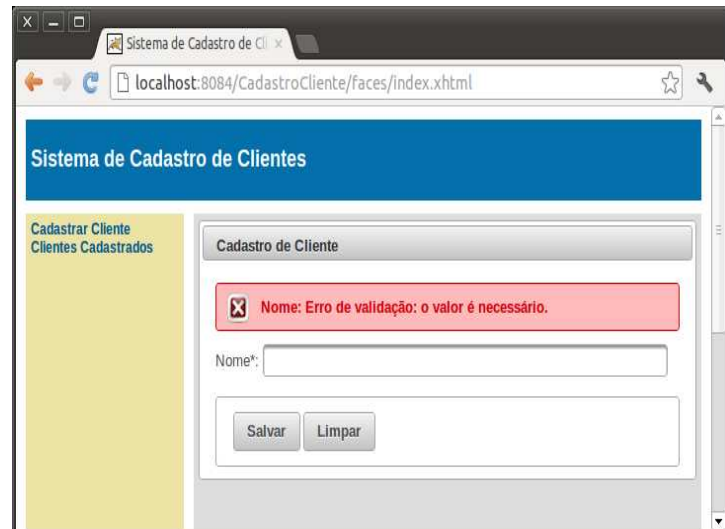


Ilustração 58: Teste com campo não preenchido

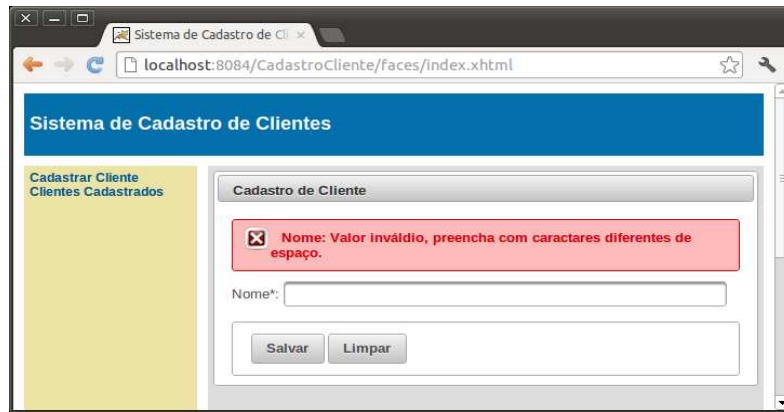


Ilustração 60: Teste de campo preenchido com sucesso

- 13 **[Campo CPF]** O campo CPF é especial, pois, precisa de máscara. Mas, o JSF facilita nossa vida, veja como é simples definir a máscara. Acrescente o trecho de código abaixo campo 'Nome' no arquivo 'cadastroCliente.xhtml':

```
CPF*:
<p:inputMask value="#{cadastroClienteController.cliente.cpf}" label="CPF"
style="200px" required="true" mask="999.999.999-99" />
<br/><br/>
```

Ilustração 61: Código do campo CPF

Entendendo a mudança:

- A tag 'inputMask' é semelhante ao 'inputText' a diferença fundamental é que é definido a máscara no atributo 'mask'.

13.1 Execute o projeto e veja se a máscara funciona corretamente.

- 14 **[Campo Sexo]** Neste campo será utilizado o componente do tipo 'Radio Button'. Adicione o trecho de código abaixo do campo CPF.

```
Sexo*:
<p:selectOneRadio value="#{cadastroClienteController.cliente.sexo}" required="true" label="Sexo" >
<f:selectItem itemLabel="Feminino" itemValue="F" />
<f:selectItem itemLabel="Masculino" itemValue="M" />
</p:selectOneRadio>
<br/><br/>
```

Ilustração 62: Código campo Sexo

14.1 Até agora nossa tela de cadastro de cliente tem o seguinte visual. Teste e veja:

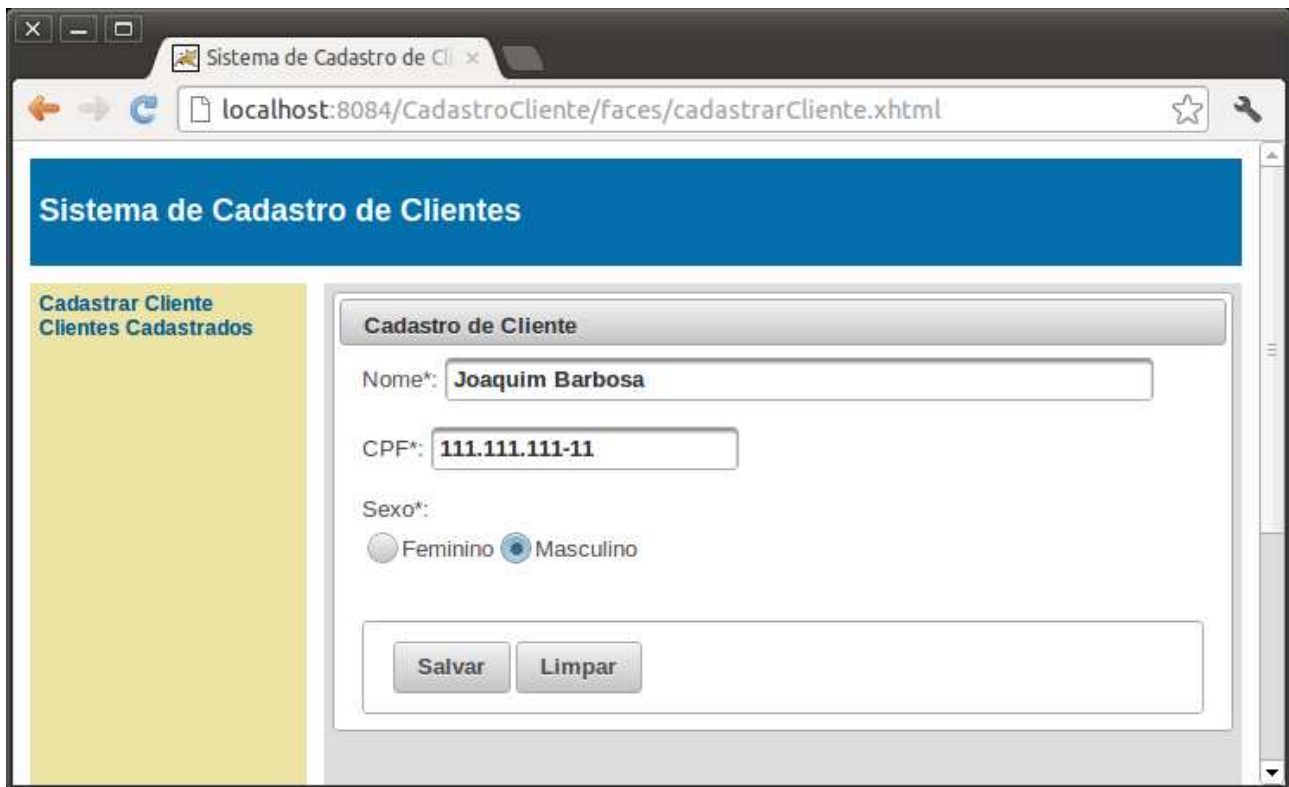


Ilustração 63: Tela parcial de cadastro de cliente

- 15 **[Campo Data de Nascimento]** Neste campo será exibido um calendário para que o usuário selecione a data ou então a digite, para isto utilize o componente '<p:calendar >' como no trecho de código que deve ser colocado após o campo 'Sexo':

```
Data de Nascimento*:
<p:calendar value="#{cadastroClienteController.cliente.dataNascimento}" required="true"
            label="Data de Nascimento" pattern="dd/MM/yyyy" locale="pt_BR" />
<br/><br/>
```

Ilustração 64: Código campo Data de Nascimento

- 16 **[Campo Cidade]** Neste campo será exibido uma combo com as cidades. Acrescente o código abaixo do campo 'Data de Nascimento':

```

Cidade*:<br/>
<p:<selectOneMenu value="#{cadastroClienteController.cliente.cidade}" label="Cidade"
    required="true">
    <f:selectItem itemLabel="Selecione..." itemValue="" />
    <f:selectItem itemLabel="Januária" itemValue="01"/>
    <f:selectItem itemLabel="Montes Claros" itemValue="02"/>
    <f:selectItem itemLabel="Volta Redonda" itemValue="03"/>
    <f:selectItem itemLabel="Rio de Janeiro" itemValue="04"/>
    <f:selectItem itemLabel="São Caetano" itemValue="05"/>
    <f:selectItem itemLabel="Campinas" itemValue="06"/>
</p:<selectOneMenu>
<br/><br/>

```

Ilustração 65: Código campo Cidade

- 17 **[Campo Salário]** Neste campo o valor será tratado pelo padrão '#,###.00'. Acrescente o trecho de código abaixo do campo 'Data Nascimento':

```

Salário:
<p:<inputText value="#{cadastroClienteController.cliente.salario}" label="Salário">
    <f:convertNumber pattern="#,###.00" />
</p:<inputText>
<br/><br/>

```

Ilustração 66: Código campo Salário

- 18 **[Campo Telefone]** Neste campo é utilizado uma máscara para o telefone, adicione o trecho de código abaixo do campo 'Salário':

```

Telefone:
<p:<inputMask value="#{cadastroClienteController.cliente.telefone}" label="Telefone"
    style="width: 200px" mask="(99) 9999-9999"/>
<br/><br/>

```

Ilustração 67: Código campo Telefone

- 19 **[Campo Email]** Este campo será um 'inputText' normal, ele será validado na camada de negócio. Adicione o trecho de código após o campo 'Telefone':

Email:

```
<p:inputText value="#{cadastrroClienteController.cliente.email}" label="Email" style="width: 400px"/>
```

```
<br/><br/>
```

(*) Campos de preenchimento obrigatório.

```
<br/><br/>
```

```
<p:panel id="pnlRodape">
```

```
    <p:commandButton value="Salvar" update="pnlFormulario" action="#{cadastrroClienteController.salvar}" />
```

```
    <p:commandButton value="Limpar" type="reset" />
```

```
</p:panel>
```

Ilustração 68: Código campo Email

20

Após seguir todos estes passos temos a tela vista na figura 69.

Sistema de Cadastro de Clientes

Cadastrar Cliente
Clientes Cadastrados

Cadastro de Cliente

Nome*:

CPF*:

Sexo*:
☐ Feminino ☐ Masculino

Data de Nascimento*:

Cidade*:
Selecione...

Salário:

Telefone:

Email:

(*) Campos de preenchimento obrigatório.

Salvar Limpar

Ilustração 69: Tela de cadastro de cliente validando todos os campos obrigatórios

21 **[Camada de Persistência]** Vamos agora implementar os códigos necessários para persistir os dados do cliente no banco de dados.

21.1 Crie a classe 'GerenciadoEntidade' que será um utilitário para obter a conexão com o banco de dados:

```
public class GerenciadorEntidade {  
    private static EntityManager gerenciadoEntidade = null;  
  
    public static EntityManager getGerenciadorEntidade(){  
        if(gerenciadoEntidade == null || !gerenciadoEntidade.isOpen()){  
            gerenciadoEntidade = Persistence.createEntityManagerFactory("CadastroClientePU").createEntityManager();  
        }  
        return gerenciadoEntidade;  
    }  
}
```

Ilustração 70: Classe GerenciadorEntidade

21.2 Crie a classe 'ClienteDAO' que será a responsável por persistir os dados da classe 'Cliente'.

```
10 public class ClienteDAO {  
11     public void persistir(Cliente cliente){  
12         EntityManager em = GerenciadorEntidade.getGerenciadorEntidade();  
13         em.getTransaction().begin();  
14         em.persist(cliente);  
15         em.getTransaction().commit();  
16     }  
17 }
```

Ilustração 71: Classe ClienteDAO

22 **[Camada de Negócio]** Esta é camada responsável por garantir a consistência das regras de negócio do seu sistema. Neste cenário as seguintes regras de negócio serão validadas para salvar um cliente:

- O CPF do cliente deve ser válido;
- Não pode cadastrar clientes com CPF iguais;
- O sexo somente pode ser 'F' para feminino e 'M' para masculino;
- O cliente deve ser maior de idade;
- O salário, se informado, não pode ser negativo;

22.1 Para validar o CPF vamos utilizar uma biblioteca chamada 'Stella' que serve para fazer algumas validações de CPF, CNPJ e outros. Ela pode ser baixada em: <https://github.com/downloads/caelum/caelum-stella/caelum-stella-core-2.0-beta1.jar>. Baixe a biblioteca e adicione na pasta /web/WEB-INF/lib e também adicione ao classpath do projeto. Para mais informações acesse a wiki do projeto:

<https://github.com/caelum/caelum-stella/wiki/Validadores-core>.

- 22.2 Modifique a classe 'ClienteDAO', adicione o método 'buscar' na classe, conforme imagem abaixo:

```
public Cliente buscar(String cpf){
    EntityManager em = GerenciadorEntidade.getGerenciadorEntidade();
    Cliente cliente = em.find(Cliente.class, cpf);
    return cliente;
}
```

Ilustração 72: Método Buscar em ClienteDAO

- 22.3 Crie a classe 'ClienteBO' e crie o método 'salvar' que tem o propósito de salvar um cliente observando as regras de negócio:

```
13 public class ClienteBO {
14     public void salvar(Cliente cliente) {
15         ClienteDAO clienteDAO = new ClienteDAO();
16
17         //Verifica se o CPF é válido
18         try {
19             CPFValidator cpfValidator = new CPFValidator(true);
20             cpfValidator.assertValid(cliente.getCpf());
21         } catch (InvalidStateException e) {
22             throw new RuntimeException("CPF inválido.");
23         }
24
25         //Verifica se já existe um cliente cadastrado com o mesmo CPF
26         Cliente clienteExistente = clienteDAO.buscar(cliente.getCpf());
27         if(clienteExistente!=null && !clienteExistente.equals(cliente)){
28             throw new RuntimeException(
29                 "Já existe outro cliente cadastrado com o mesmo CPF.");
30         }
31
32         //Verifica se os sexos informados são válidos
33         if(!cliente.getSexo().equals('F') && !cliente.getSexo().equals('M')) {
34             throw new RuntimeException("Sexo inválido");
35         }
36
37         //Verifica se o cliente é maior de idade
38         GregorianCalendar data18AnosAtras = new GregorianCalendar();
39         data18AnosAtras.add(GregorianCalendar.YEAR, -18);
40         if(cliente.getDataNascimento().after(data18AnosAtras.getTime())){
41             throw new RuntimeException("O cliente deve ser maior de idade.");
42         }
43
44         //Verifica se o salário não é negativo
45         if(cliente.getSalario()!=null && cliente.getSalario()<0.0){
46             throw new RuntimeException("O salário não pode ser negativo.");
47         }
48
49         //Caso todas as regras sejam atendidas, persiste o cliente no BD
50         clienteDAO.persistir(cliente);
51     }
52 }
```

Ilustração 73: Classe ClienteBO

22.4 Modifique o método 'Salvar' do 'Controller':

```
public void salvar() {  
    // Obtém o contexto do JSF  
    FacesContext facesContext = FacesContext.getCurrentInstance();  
    try {  
        // Aciona a camda de negócio para salvar o cliente  
        ClienteBO clienteBO = new ClienteBO();  
        clienteBO.salvar(this.cliente);  
        // Exibe mensagem de sucesso  
        facesContext.addMessage(null, new FacesMessage("Cliente: "+this.cliente.getNome()+" cadastrado com sucesso!"));  
        // Instancie um novo objeto de cliente para iniciar um novo cadastro  
        this.cliente = new Cliente();  
    } catch (Exception e) {  
        // Caso haja algum erro exibe mensagem de falha  
        FacesMessage facesMessage = new FacesMessage(FacesMessage.SEVERITY_ERROR, e.getMessage(), e.getMessage());  
        facesContext.addMessage(null, facesMessage);  
    }  
}
```

Ilustração 74: Modificações no método salvar do Controller

22.5 Teste e veja os resultados:

Sistema de Cadastro de Clientes

Cadastrar Cliente
Clientes Cadastrados

Cadastro de Cliente

Cliente: Hélder Seixas Lima cadastrado com sucesso!

Nome*:

CPF*:

Sexo*: ☐ Feminino ☐ Masculino

Data de Nascimento*:

Cidade*:

Salário:

Telefone:

Email:

(*) Campos de preenchimento obrigatório.

Salvar Limpar

Ilustração 75: Tela de cadastro de cliente já em estado funcional

23 Muito legal, mas, seria interessante o usuário ter um retorno enquanto se processa a requisição Ajax. Para isso vamos adicionar uma tela de 'Carregando'.

23.1 Adicione o arquivo 'loadgin.gif' na pasta '/web/resources/imagem'. O arquivo pode ser baixado em: <http://logd.tw.rpi.edu/files/loading.gif>

23.2 Adicione o trecho de código no arquivo 'template.xhtml' logo no início da tag '<h:body >':

```
<p:ajaxStatus onStart="statusDialog.show();" onSuccess="statusDialog.hide();" />
<p:dialog modal="true" widgetVar="statusDialog" header="Carregando..."
    draggable="false" closable="false">
    <p:graphicImage value="./resources/imagem/loading.gif" />
</p:dialog>
```

Ilustração 76: Modificação no arquivo template.xhtml para adicionar imagem de carregando

Entendendo a modificação

- A tag '<p:ajaxStatus >' define qual ação será executada quando se inicia e finaliza uma requisição ajax.

23.3 A tag '<p:dilog >' define uma tela no estilo 'modal' que impede que o usuário modifique a tela principal e exibe o arquivo '.gif'.

23.4 Teste e veja os resultados.

24 **[Tela de Clientes Cadastrados]** Após criar a tela de cadastro de clientes, você já tem condições de evoluir sozinho. Construa você mesmo a tela de visualização de clientes cadastrados, conforme a imagem 77. Para implementar uma tabela utilize o componente '<p:datatable >', para conhecer exemplos de tabela acesse: <http://www.primefaces.org/showcase-labs/ui/datatableHome.jsf>.

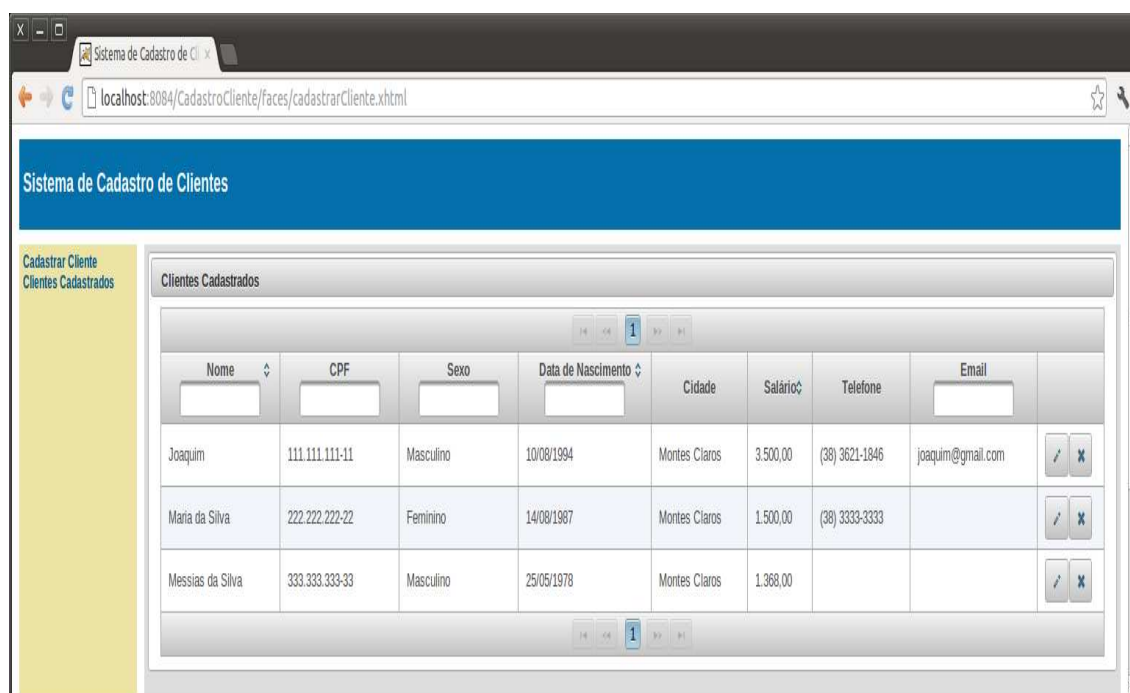


Ilustração 77: Tela de clientes cadastrados

10 Conclusão

Nesta apostila você aprendeu os conceitos básicos de uma aplicação Java Web (Servlets e JSP), foi apresentado ao desenvolvimento seguindo o padrão arquitetural MVC e a utilização do framework JSF baseado na implementação PrimeFaces com AJAX integrado. Também colocou em prática o desenvolvimento de uma aplicação Web com acesso ao banco de dados via framework JPA.

Foi possível compreender que o desenvolvimento Web é mais complexo que uma aplicação desktop convencional, por acrescentar uma série de fatores que o programador deve ter conhecimento.

Pratique e conheça mais as possibilidades do desenvolvimento de aplicações ricas em Java utilizando frameworks como o JSF.