

**FIAP GRADUAÇÃO**

# **DIGITAL BUSINESS ENABLEMENT**

*Prof. THIAGO T. I. YAMAMOTO*

#08 - JSF INTRODUÇÃO



- ♦ Tipos de Framework Web
- ♦ Java Server Faces
- ♦ Componentes Visuais do JSF
- ♦ Escopo do Managed Bean
- ♦ Javascript e CSS no JSF

## Request-based

- Trabalha perto do protocolo HTTP
- Dificuldade no reuso de código de interface
- Vasta documentação e casos de sucesso
- Exemplos: Struts, Spring e WebWork

Struts<sup>2</sup>

spring

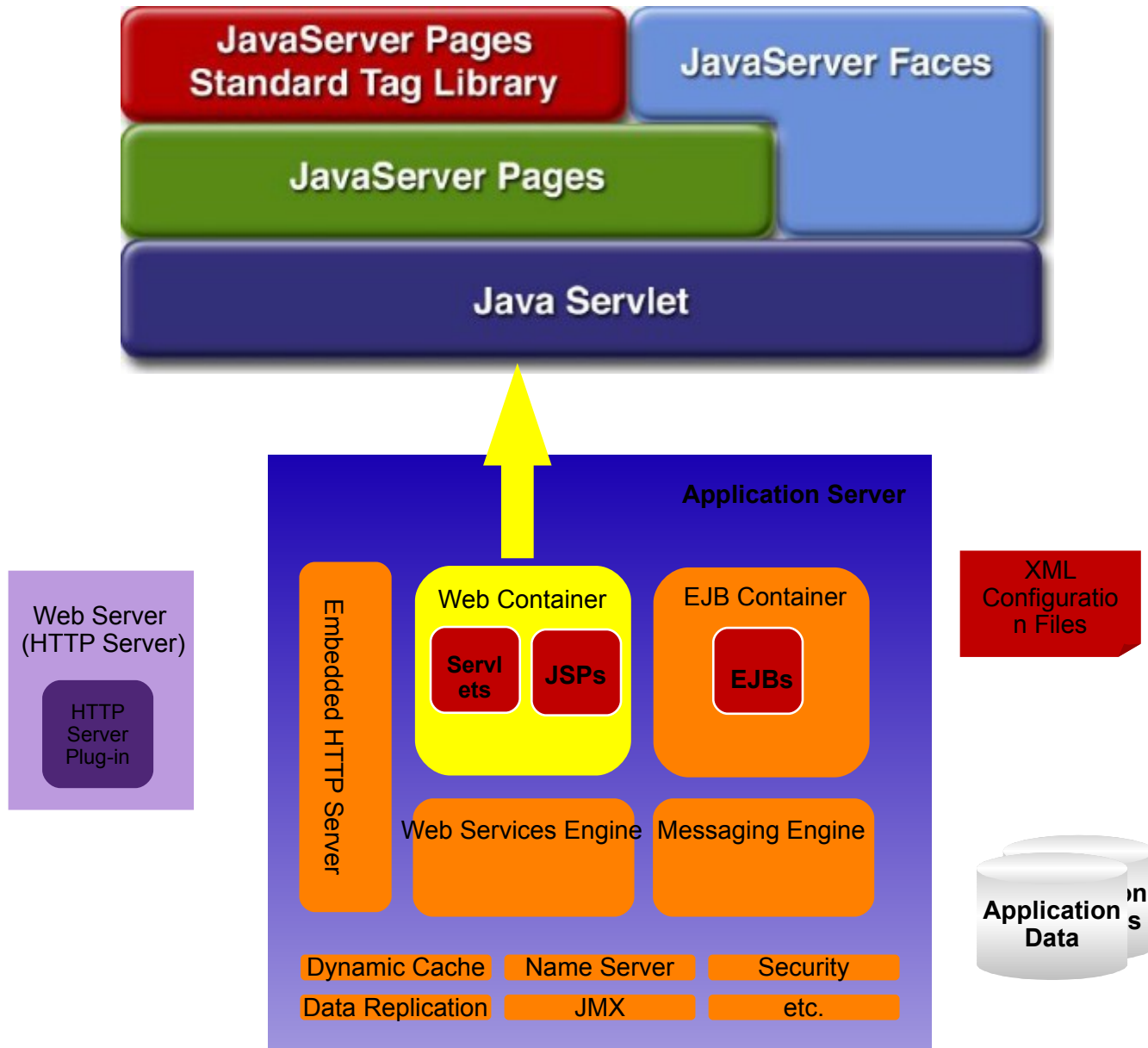
## Component-based

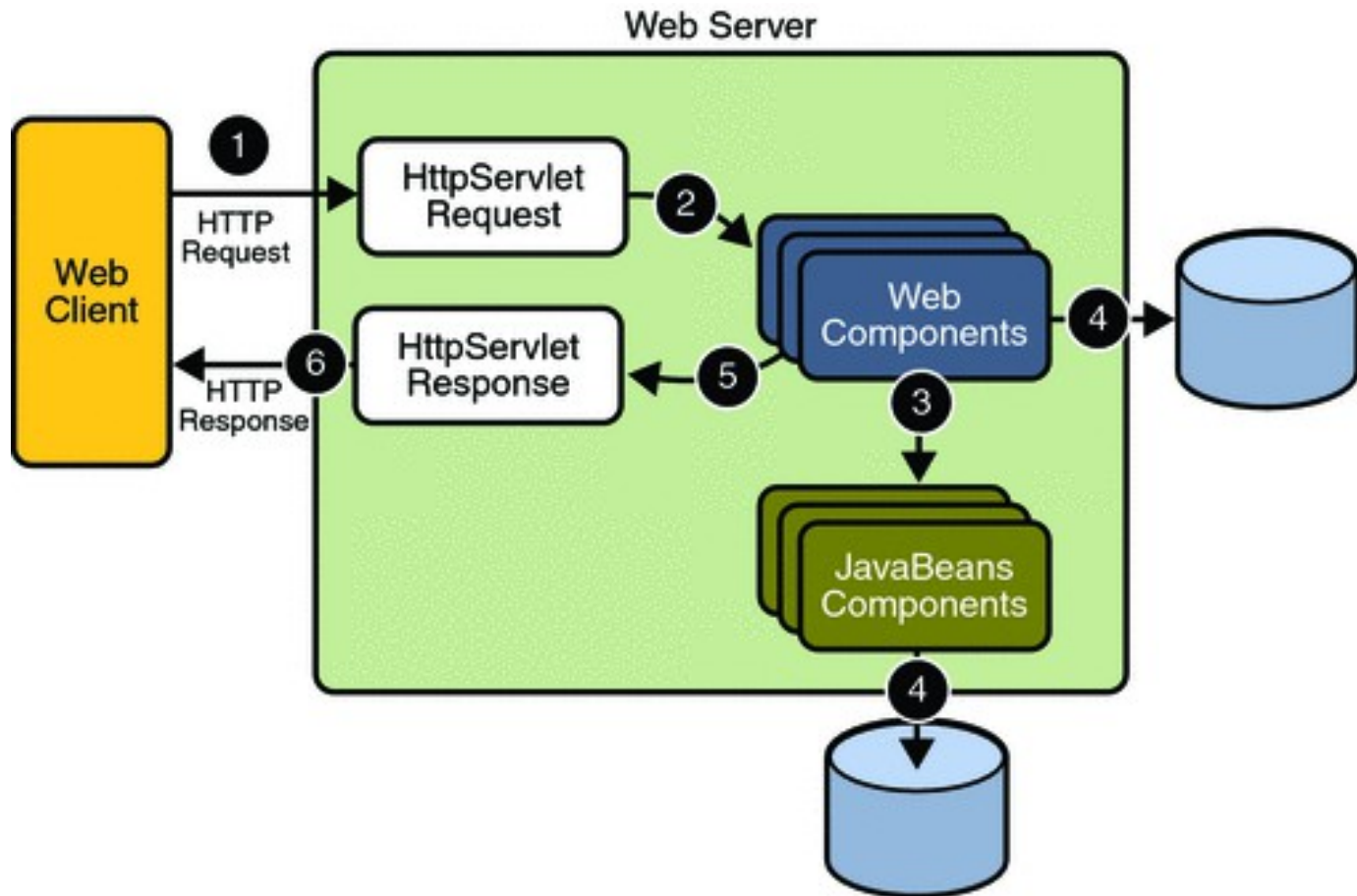
- Modelo de desenvolvimento parecido com desktop
- Gera componentes a partir da necessidade da tela
- Crescimento acelerado como padrão de interface Web
- Exemplos: JSF, Tapestry e GWT

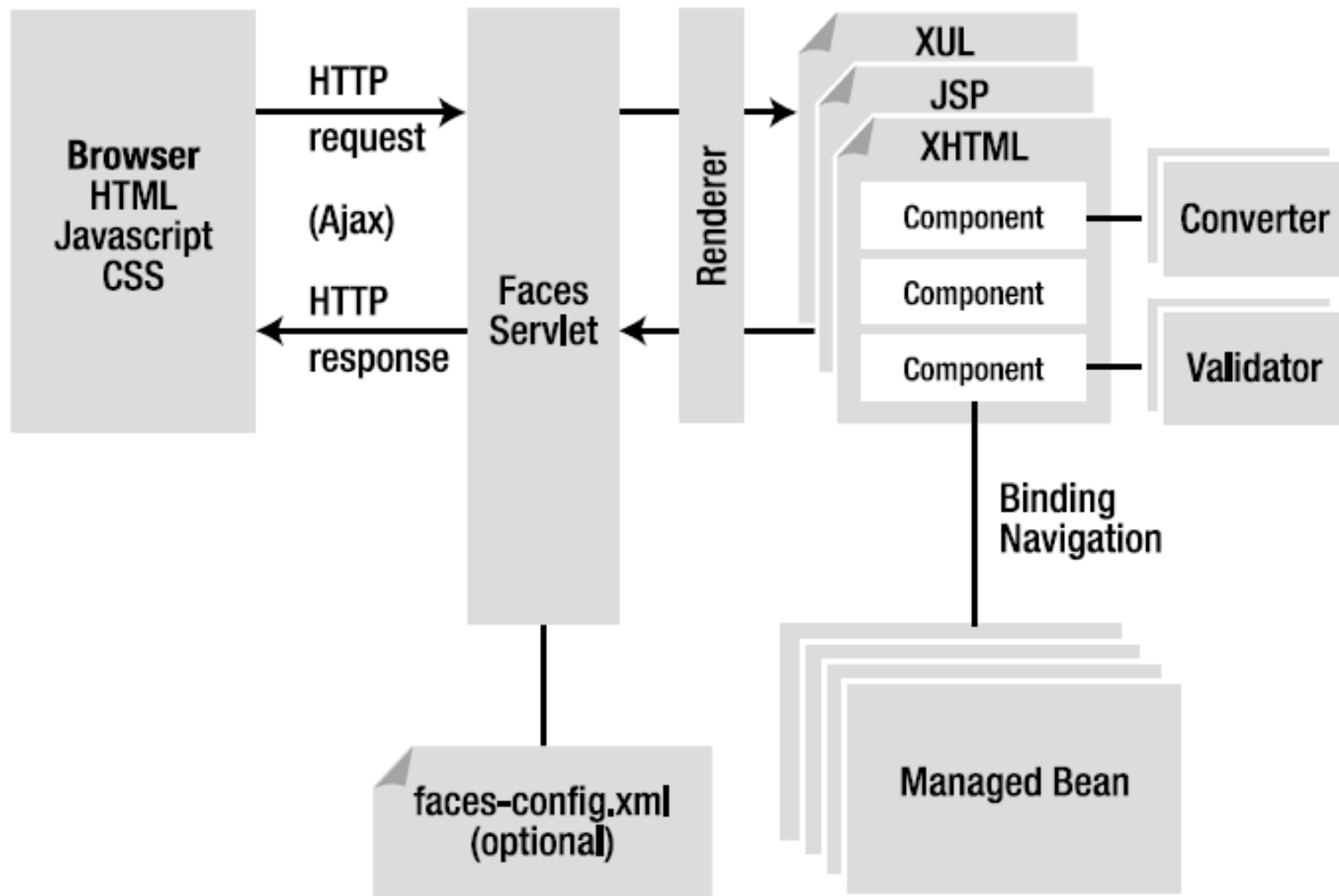


# JAVA SERVER FACES

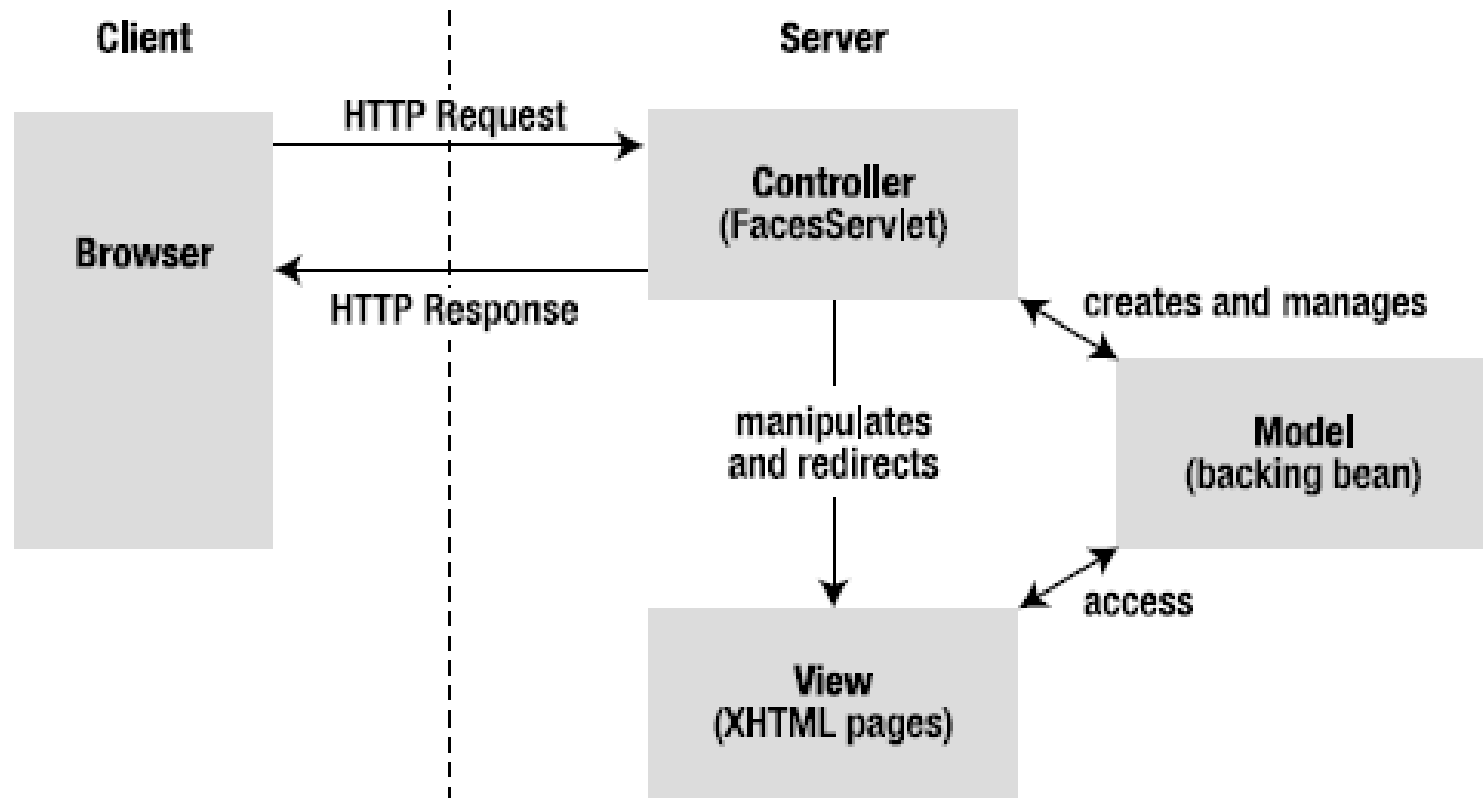
# JAVA EE API - WEB CONTAINER



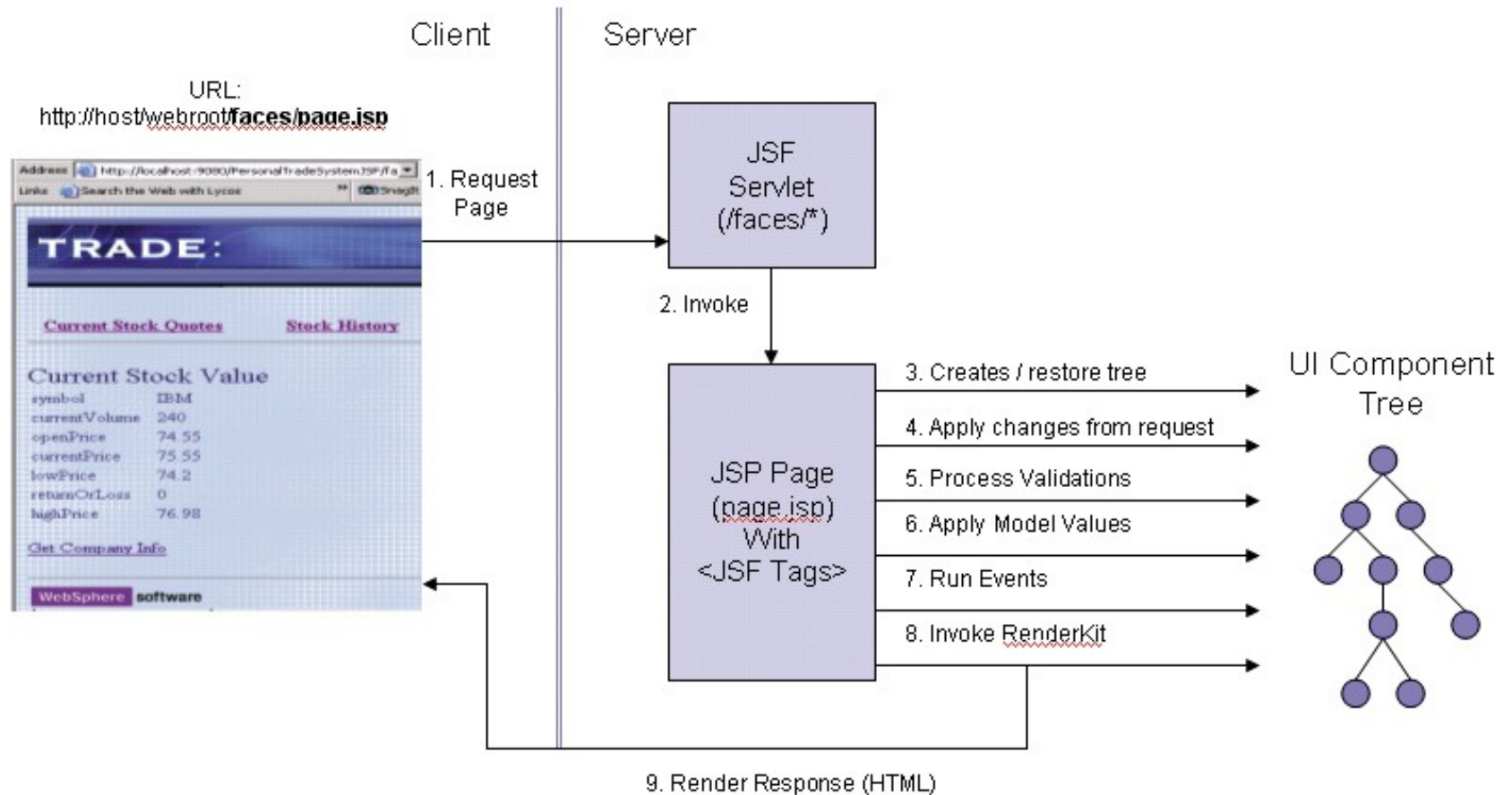




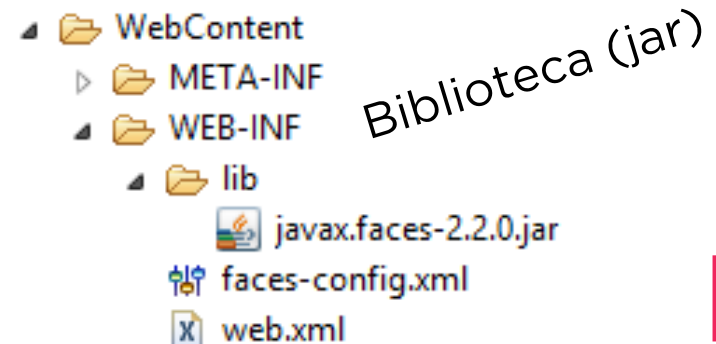


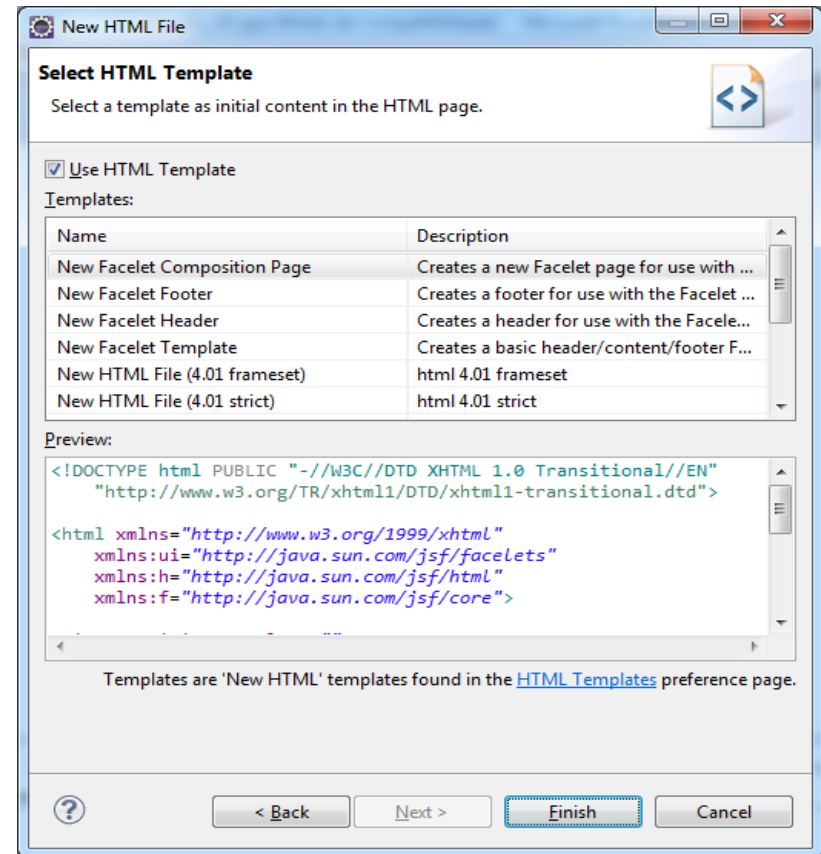
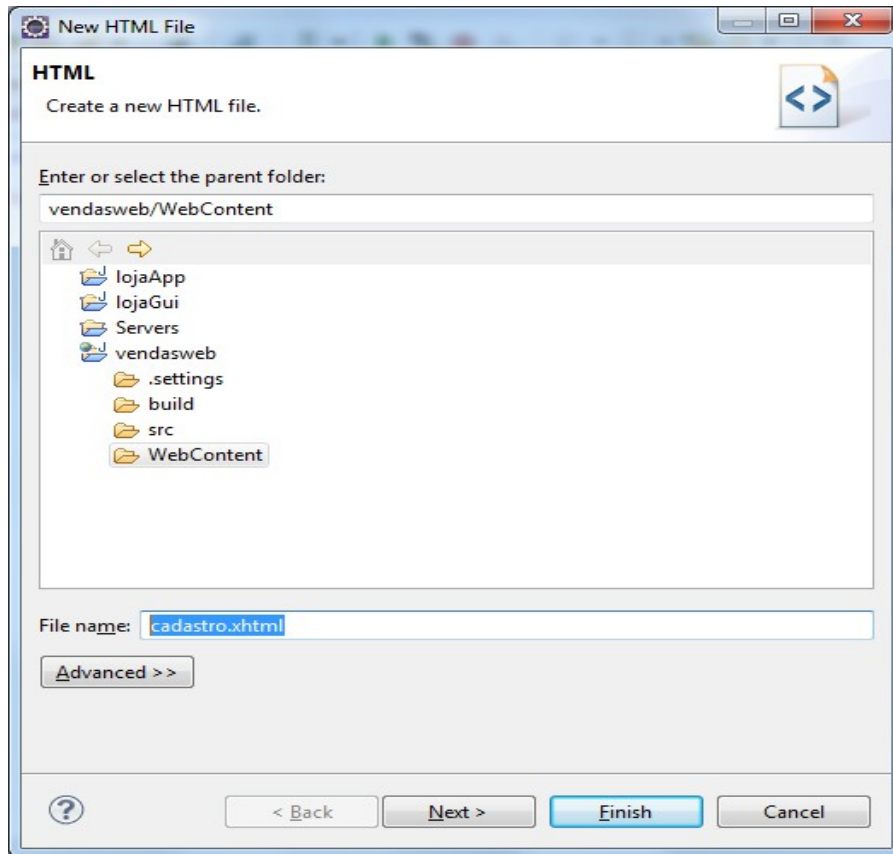


# JSF - CICLO DE VIDA



- A **especificação** Java define como JSF deve ser implementado.
- Fornecedores produzem o framework a partir da especificação
- JSF utiliza técnica de CoC (Convention of Configuration) e anotações para o desenvolvimento do código
- Implementações mais conhecidas:
  - **Oracle Mojarra Java Server Faces - Reference Implementation**
  - Apache MyFaces
  - ADF Faces (Oracle) doada para Apache
  - IBM Faces





```
<h:form id="frmCadastro">
```

```
<h:inputText id="txtUsuario" value="#{userBean.name}" required="true"/>
```

```
<h:inputSecret id="txtSenha" value="#{userBean.password}" required="true">
```

```
<f:validateLength minimum="6"/>
```

```
</h:inputSecret>
```

```
<h:commandButton id="btnConfirmar" action="#{userBean.validate}">
```

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

- Padronizar nome de página sempre com minúsculo. Ex: cadastro.xhtml, reposicao-estoque.xhtml). Não utilizar acentos ou espaços em branco em nomes de página
- As páginas comunicam-se com as classes Java por meio de **Expression Language (EL)**. Elas são formadas por #{nome-managed-bean.nome-propriedade}.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
<h:head> <title>Login</title> </h:head>
<h:body>
<h:form>
    Nome: <h:inputText value="#{cadastroBean.login}" />
    <h:commandButton value="Confirmar" action="#{cadastroBean.logar}" />
</h:form>
</h:body>
</html>
```

- Captura as ações e valores preenchidos na página Web
- Usar padrão de nomenclatura de classe Java para criar Managed Bean
- Inserir sufixo Bean por convenção (Ex: LoginBean, CadastroClienteBean)
- Colocar nomes que estejam relacionados com a página

@ManagedBean

```
public class LoginBean {  
    private String login;  
    private String senha;  
  
    public void logar() {  
        if ("thiago".equals(getLogin()) && "password".equals(getSenha())) {  
            System.out.println("Usuario logado: " + getLogin());  
        } else {  
            System.out.println("Usuario com senha incorreta: " + getLogin());  
        }  
    }  
}
```

```
<!-- Configura a servlet do JSF. -->
```

```
<servlet>
```

```
  <servlet-name>Faces Servlet</servlet-name>
```

```
  <servlet-class> javax.faces.webapp.FacesServlet</servlet-class>
```

```
  <load-on-startup>1</load-on-startup>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
  <servlet-name>Faces Servlet</servlet-name>
```

```
  <url-pattern>/faces/*</url-pattern>
```

```
</servlet-mapping>
```

# I EXERCÍCIO 1

Crie um projeto JSF para validação de Login e Senha.



- Criar um projeto Java Web Application com o nome "**vendasweb**" com configuration de JSF 2.2
- Montar uma página que deverá chamar-se cadastroProduto.xhtml
- O cliente pesquisará informações de um produto a partir do argumento de pesquisa "Código do Produto";
- Utilizar lógica de negócio EstoqueBO utilizada na aula pwai1c\_Framework (Exercício 1);
- Olhe o view-source do código HTML para ver a página gerada pelo browser
- Para a realização do exercício você deve utilizar os componentes JSF abaixo:
  - JSF HTML: Form (prefixo de padrão de nome de instância: frm)
  - JSF HTML: OutputText
  - JSF HTML: InputText (prefixo de padrão de nome de instância : txt)
  - JSF HTML: Command Button (prefixo de padrão de nome de instância : btn)
  - Managed Bean
  - Notação de Expression Language JSF ( #{ } )
  - JSF HTML: Panel Grid

# JSF – COMPONENTES VISUAIS

# I OBJETOS VISUAIS DE PÁGINAS JSF FIAP

- No JSF o componente responsável por gerar o HTML final chama-se RenderKit
- No desenvolvimento de formulários nós podemos misturar objetos de diferentes bibliotecas. Estas bibliotecas ficam organizadas em subgrupos (também conhecidas como taglibs).
- As 2 taglibs mais básicas que temos no desenvolvimento com JSF são:
  - JSF Core
  - JSF HTML
- Os objetos em JSF possuem uma uniformidade no modo de interação com o Managed Bean
- Para entender o comportamento específico do objeto basta entender a interação dele com as suas respectivas propriedades
- Utilizar a **propriedade ID** dos objetos de página JSF é uma boa prática para facilitar a depuração do html gerado

- Componentes JSF são elementos como `inputText`, `commandButton`, `outputText`, `dataTable` que são usados para criar interface com o usuário (UI – User Interface) da aplicação JSF, ou seja, estes objetos que realizam a interação com o usuário.
- Existem 2 categorias de componentes JSF:
  - **Simple Components (Componentes Simples)**, por exemplo: `inputText`, `outputText` e `commandButton`;
  - **Compound Components (Componentes Compostos)**, por exemplo: `dataTable` e `panelGrid`.
- **Simple Components:** são componentes individuais, considerados simples;
- **Compound Components:** são componentes compostos por outros componentes dentro dele, por exemplo uma tabela.

- `<h:inputText>`
  - Campo de texto.
- `<h:inputTextarea>`
  - Campo de área de texto.
- `<h:inputSecret>`
  - Campo de senha.

## Exemplo:

### XHTML:

```
<h:inputText value="#{loginBean.user}"/>
```

### JAVA:

```
@ManagedBean  
public class LoginBean {  
  
    private String user;  
  
    //get e set  
}
```

- `<h:outputLabel>`
  - Cria um elemento de label

## Exemplo:

```
<h:outputLabel value="Login" for="user"/>  
<h:inputText value="#{loginBean.user}" id="user"/>
```

- `<h:selectBooleanCheckbox>`
  - Define um campo de checkbox;
  - Esse campo é associado a uma propriedade do tipo boolean no managed bean.

## Exemplo:

### XHTML:

```
<h:selectBooleanCheckbox value="#{cadastroBean.termo}" id="idConcordo"/>  
<h:outputLabel value="Concordo com os termos" for="idConcordo"/>
```

### JAVA:

```
@ManagedBean  
public class CadastroBean{  
  
    private boolean termo;  
  
    //get e set  
}
```

# CAIXAS DE SELEÇÃO - CHECKBOX

- `<h:selectManyCheckbox>`
  - Define um conjunto de campos de checkbox
  - As opções podem ser definidas de forma estática e/ou dinamica:
    - `<f:selectItem>`
      - Define cada opção (Estático)
    - `<f:selectItems>`
      - Define dinamicamente as opções, através de uma lista.

## Exemplo com itens estáticos:

### XHTML:

```
<h:selectManyCheckbox value="#{cadastroBean.listaUsuariosSelecionados}">
  <f:selectItem itemLabel="Thiago" itemValue="1"/>
  <f:selectItem itemLabel="Leandro" itemValue="2"/>
</h:selectManyCheckbox>
```

### JAVA:

```
@ManagedBean
public class CadastroBean{

    private List<String> listaUsuariosSelecionados;
    //get e set

}
```



# CAIXAS DE SELEÇÃO - CHECKBOX FIAP

## Exemplo com itens dinamicos: (valores do banco)

### XHTML:

```
<h:selectManyCheckbox value="#{cadastroBean.listaUsuariosSelecionados}">
  <f:selectItems value="#{cadastroBean.todosUsuarios()}"
    var="usuario"
    itemLabel="#{usuario.nome}"
    itemValue="#{usuario.id}"/>
</h:selectManyCheckbox>
```

### JAVA:

```
@ManagedBean
public class CadastroBean{

    private List<String> listaUsuariosSelecionados;

    public List<Usuario> todosUsuarios(){
        UsuarioBO bo = new UsuarioBO();
        return bo.getAll();
    }
    //get e set
}
```

- **value** - lista de itens
- **var** - define uma variável para ser utilizado no itemLabel e itemValue
- **itemLabel** - valor que será exibido na tela
- **itemValue** - valor que será enviado para o servidor se a opção for escolhida

- **<h:selectOneRadio>**  
Define um conjunto de campos de radio
- **<h:selectOneMenu>**  
Define um select list
- **<h:selectOneListbox>**  
Define um select list, mas exibe todas as opções

Os campos são parecidos, permitem a escolha de uma única opção. Se diferem na forma que são exibidos na tela.

<h:selectOneRadio>

## Exemplo com itens dinamicos: (valores do banco)

### XHTML:

```
<h:selectOneRadio value="#{cadastroBean.usuarioSelecionado}">
  <f:selectItems value="#{cadastroBean.todosUsuarios()}"
    var="usuario"
    itemLabel="#{usuario.nome}"
    itemValue="#{usuario.id}"/>
</h:selectOneRadio>
```

### JAVA:

@ManagedBean

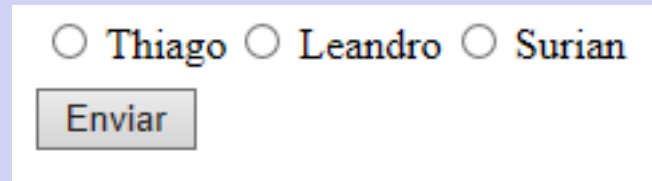
```
public class CadastroBean{
```

```
    private String usuarioSelecionado;
```

```
    public List<Usuario> todosUsuarios(){
        UsuarioBO bo = new UsuarioBO();
        return bo.getAll();
    }
```

```
    //get e set
```

```
}
```



☐ Thiago ☐ Leandro ☐ Surian

<h:selectOneMenu>

## Exemplo com itens dinamicos: (valores do banco)

### XHTML:

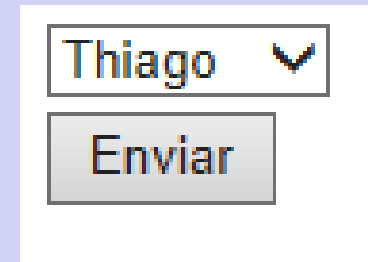
```
<h:selectOneMenu value="#{cadastroBean.usuarioSelecionado}">
  <f:selectItems value="#{cadastroBean.todosUsuarios()}"
    var="usuario"
    itemLabel="#{usuario.nome}"
    itemValue="#{usuario.id}"/>
</h:selectOneMenu>
```

### JAVA:

```
@ManagedBean
public class CadastroBean{

    private String usuarioSelecionado;

    public List<Usuario> todosUsuarios(){
        UsuarioBO bo = new UsuarioBO();
        return bo.getAll();
    }
    //get e set
}
```

A screenshot of a web form. It features a dropdown menu with the text 'Thiago' and a blue downward arrow. Below the dropdown is a button with the text 'Enviar'.

<h:selectOneListBox>

## Exemplo com itens dinamicos: (valores do banco)

### XHTML:

```
<h:selectOneListbox value="#{cadastroBean.usuarioSelecionado}">
  <f:selectItems value="#{cadastroBean.todosUsuarios()}"
    var="usuario"
    itemLabel="#{usuario.nome}"
    itemValue="#{usuario.id}"/>
</h:selectOneListbox>
```

### JAVA:

```
@ManagedBean
public class CadastroBean{

    private String usuarioSelecionado;

    public List<Usuario> todosUsuarios(){
        UsuarioBO bo = new UsuarioBO();
        return bo.getAll();
    }
    //get e set
}
```

Thiago  
Leandro  
Surian

Enviar

- `<h:commandButton>`
- `<h:commandLink>`

Enviam os dados de um formulárioHTML para o servidor através do método POST do HTTP, diferem na forma que é mostrado na tela.

## Exemplo:

### XHTML:

```
<h:commandButton actionListener="#{loginBean.logar}" value="Login" />
```

### JAVA:

```
@ManagedBean  
public class LoginBean{  
  
    public void logar() {  
        //Lógica de logar  
    }  
  
}
```

## Exemplo de link e link com imagem:

### XHTML:

#### Link com texto:

```
<h:commandLink actionListener="#{loginBean.logar}" value="Login" />
```

#### Link com imagem:

```
<h:commandLink actionListener="#{loginBean.logar}">  
    <h:graphicImage value="/enviar.jpg"/>  
</h:commandLink>
```

### JAVA:

```
@ManagedBean  
public class LoginBean{  
  
    public void logar(){  
        //Lógica de logar  
    }  
  
}
```

- **readonly**
  - Campo somente de leitura
- **rendered**
  - true: o campo será exibido
  - false: campo ficará oculto

## Exemplo:

```
<h:inputText value="#{cadastroProdutoBean.produto.codProduto}"
              readonly="true"
              rendered="#{cadastroProdutoBean.produto.codProduto != 0}">
</h:inputText>

<h:commandButton
    rendered="#{!empty cadastroCliente.acao}"
    value="Pesquisa Cliente"
    action="#{cadastroCliente.btnPesquisaCliente}"/>
```



- Podemos exibir mensagens na tela;
- Muito utilizado com validação.

## Exemplo:

### **JAVA:**

```
FacesMessage mensagem = new FacesMessage( " Usuário cadastrado" );  
FacesContext.getCurrentInstance().addMessage( null , mensagem);
```

### **XHTML:**

```
< h : messages />
```

- `<h:dataTable>`
  - Utilizado na criação de tabelas.
- `<h:column>`
  - Coluna de uma tabela.
- `<f:facet>`
  - Utilizado para definir cabeçalho, rodapés e o título de cada coluna.

## Exemplo:

```
<h:dataTable value="#{cursosBean.cursos}" var="curso">
```



Lista de elementos



Variável que recebe cada item da lista.

## Exemplo:

```
<h:dataTable value="#{cadastroBean.produtos}" var="prod" >
  <h:column>
    <f:facet name="header">Código</f:facet>
    #{prod.codigo}
  </h:column>
  <h:column>
    <f:facet name="header">Descrição</f:facet>
    #{prod.descricao}
  </h:column>
  <h:column>
    <f:facet name="header">Preço</f:facet>
    #{prod.preco}
  </h:column>
  <h:column>
    <f:facet name="header">Quantidade</f:facet>
    #{prod.qtd}
  </h:column>
</h:dataTable>
```

```
<h:outputText value="#{p.quantidade}">
    <f:convertNumber pattern= "#,###" />
</h:outputText>
<h:outputText value="#{p.preco}">
    <f:convertNumber currencyCode="BRL" type="currency"/>
</h:outputText>
```

*Mais informações: <http://www.roseindia.net/jsf/convertNumber.shtml>*

```
<h:outputText value="#{p.dataHoje.time}">
    <f:convertDateTime timeZone="America/Sao_Paulo" />
</h:outputText>
```

## Observação:

A propriedade **.time** para data é usada somente para objetos do tipo **java.util.Calendar**. Para **java.util.Date** coloque “p.dataHoje”.

*Mais informações: <http://www.roseindia.net/jsf/convertDateTime.shtml>*

# ESCOPO: MANAGED BEAN

```
@ManagedBean (name="login")
```

```
@ViewScoped
```

```
public class LoginBean implements Serializable{
```

```
    private String login;
```

```
    private String senha;
```

```
    /* get e set's */
```

```
    public void logar() {
```

```
        if ("thiago".equals(this.login) &&  
            "password".equals(this.senha)) {
```

```
            System.out.println("Usuario logado: " + getLogin() );
```

```
        } else {
```

```
            System.out.println("Usuario com senha errada: " + getLogin() );
```

```
        }
```

```
    }
```

```
}
```

O Managed Bean pode ter 5 escopos:

- **Request:** conteúdo do Managed Bean visível somente no fluxo request-response
- **View:** conteúdo do Managed Bean visível enquanto a página estiver em processamento. Escopo maior que Request e menor que Session.
- **Session:** conteúdo do Managed Bean visível durante toda a sessão do usuário
- **Application:** conteúdo do Managed Bean para todas as sessões do servidor
- **Custom:** possibilidade de se customizar um escopo de acordo com a necessidade

Para manipular estes valores diretamente é necessário interagir com o objeto Faces Context

## JSF:

Um projeto JSF pode trabalhar como WebService **Provider** ou WebService **Requester**. Tudo depende de lógica da funcionalidade.

O WebServices no JSF segue as mesmas regras que foram utilizadas com outros tipos de canais

## JPA:

Um projeto JSF pode utilizar JPA para acesso ao banco de dados. Para isso, as bibliotecas do JPA/Hibernate devem estar dentro da pasta “**WEB-INF/lib**” e o persistence.xml dentro da pasta “**Java Resources/src/META-INF**”

O JPA no JSF segue as mesmas regras, ou seja é recomendado utilizar os padrões DAO, Singleton e etc..

É possível gerar o conteúdo misturando elementos **dinâmicos** e elementos **estáticos** através das tags:

- `<h:outputScript>`
- `<h:outputStylesheet>`

Por convenção, recursos estáticos devem ficar no diretório **resources**. Exemplos:

- `WebContent/resources/css`
- `WebContent/resources/imagens`

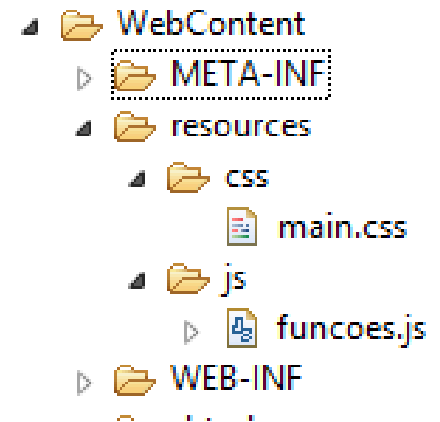
Para aplicações dinâmicas na Web é recomendável habilitar a meta tag para desativar o cache:

```
<meta http-equiv="Pragma" content="no-cache" />
```



```
<html>
<h:head>
  <title>Banco FIAP - Internet Banking </title>
  <META HTTP-EQUIV="Pragma" CONTENT="no-cache">
  <h:outputScript library="js" name="funcoes.js" />
  <h:outputStylesheet library="css" name="main.css" />
</h:head>
<h:body>
  <h:form>
    <div ><div id="container">
      <h:graphicImage library="imagens" name="logo.png" id="logo"/>
    </div></div>
    Preencher Conta Corrente: <h:inputText styleClass="form" />

    <h:commandButton styleClass="button" value="Salvar"/> <br/>
    Saldo: <h:outputText styleClass="txtBlue-10" value="99999"/>
  </h:form>
</h:body>
</html>
```



Copyright © 2013 - 2017 Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

*“Aquele que não luta pelo futuro que quer, deve  
aceitar o futuro que vier”*