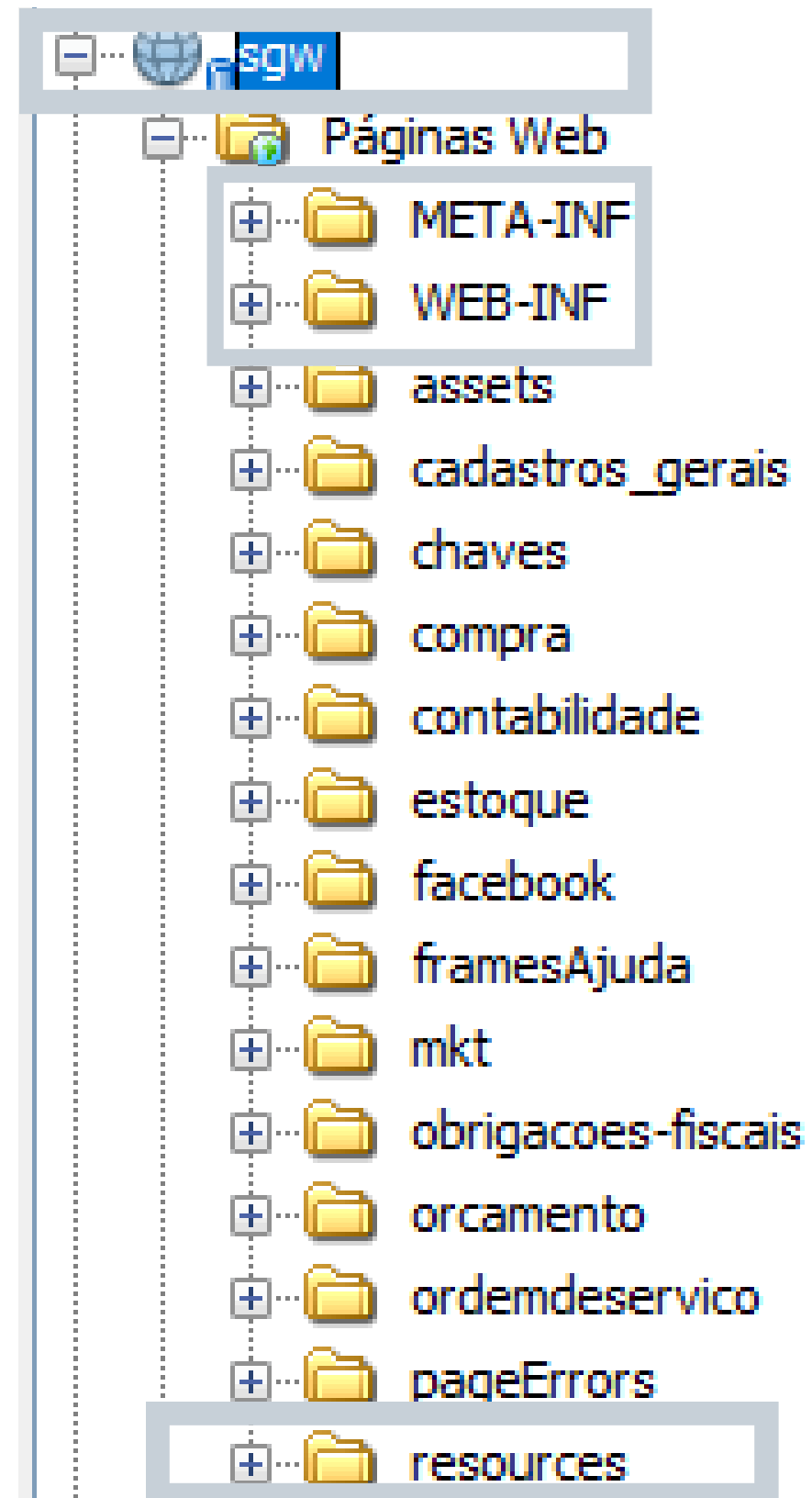


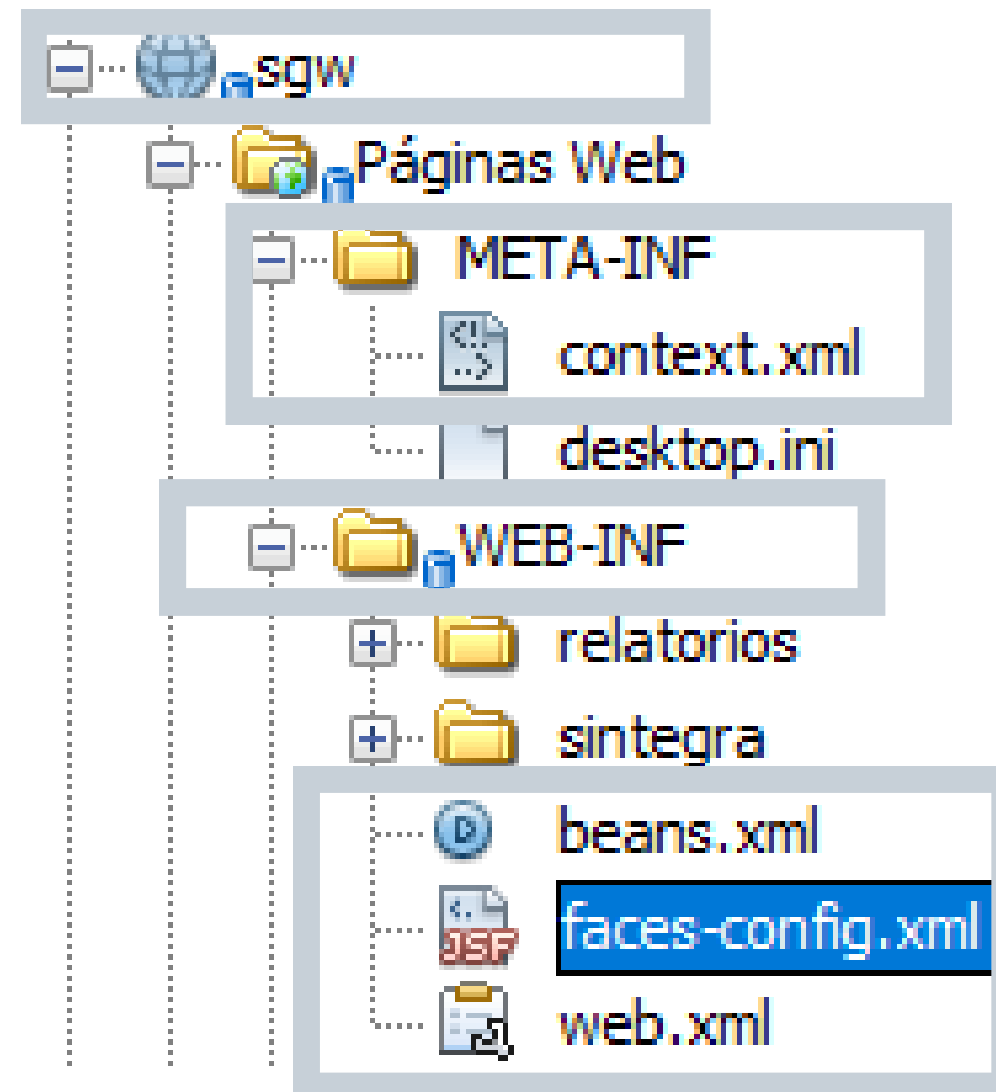
FUNCIONAMENTO JSF



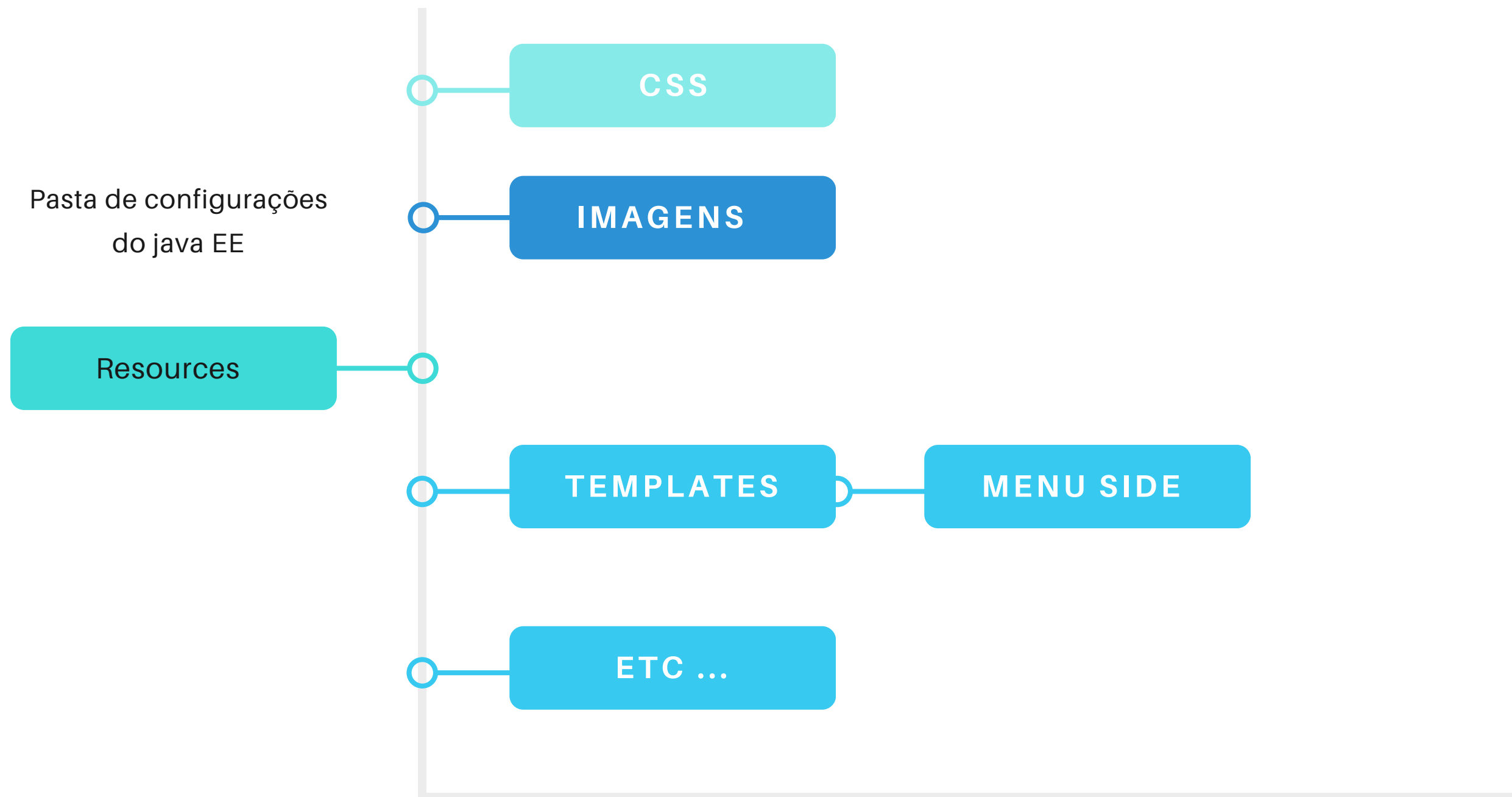
FUNCIONAMENTO JSF



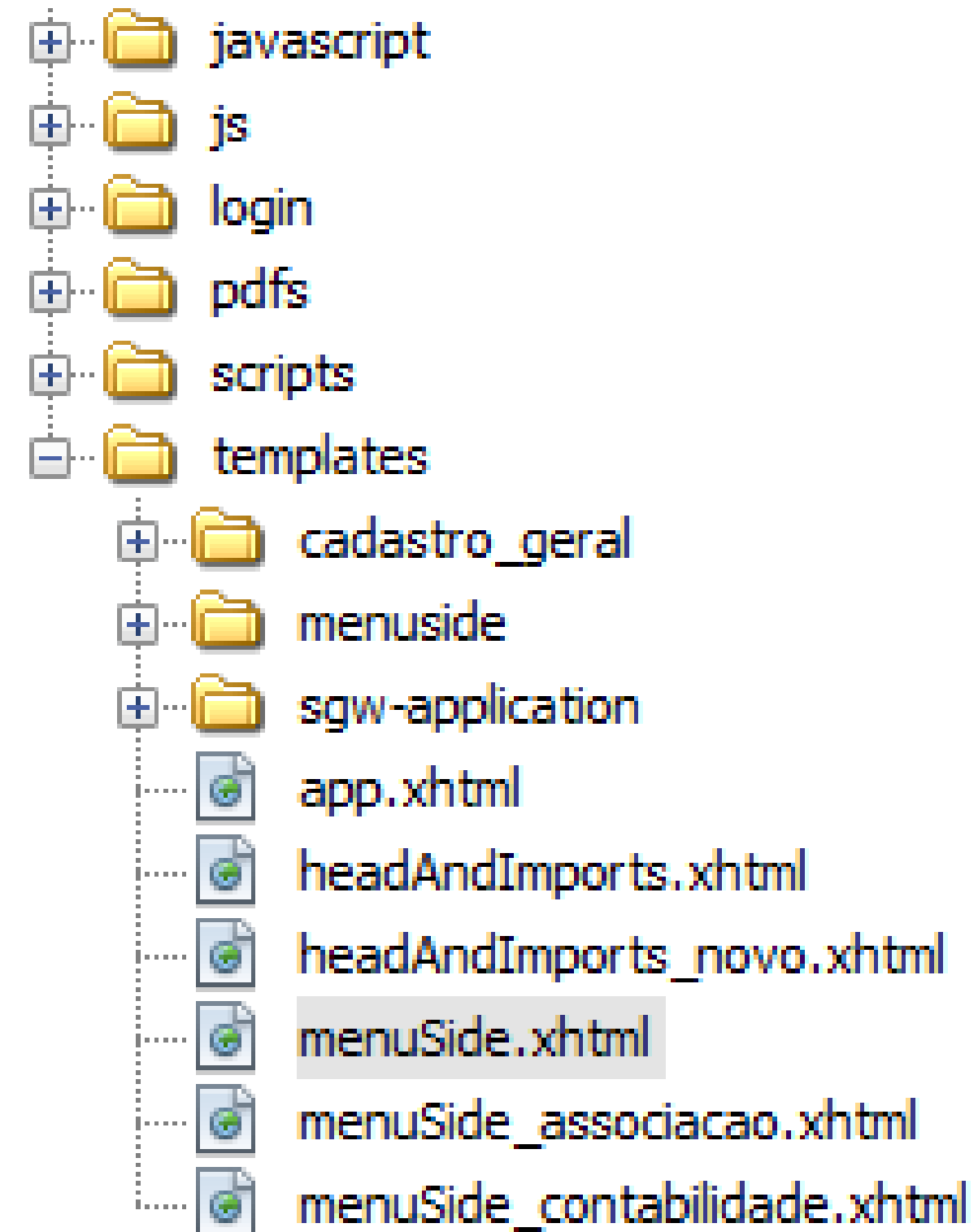
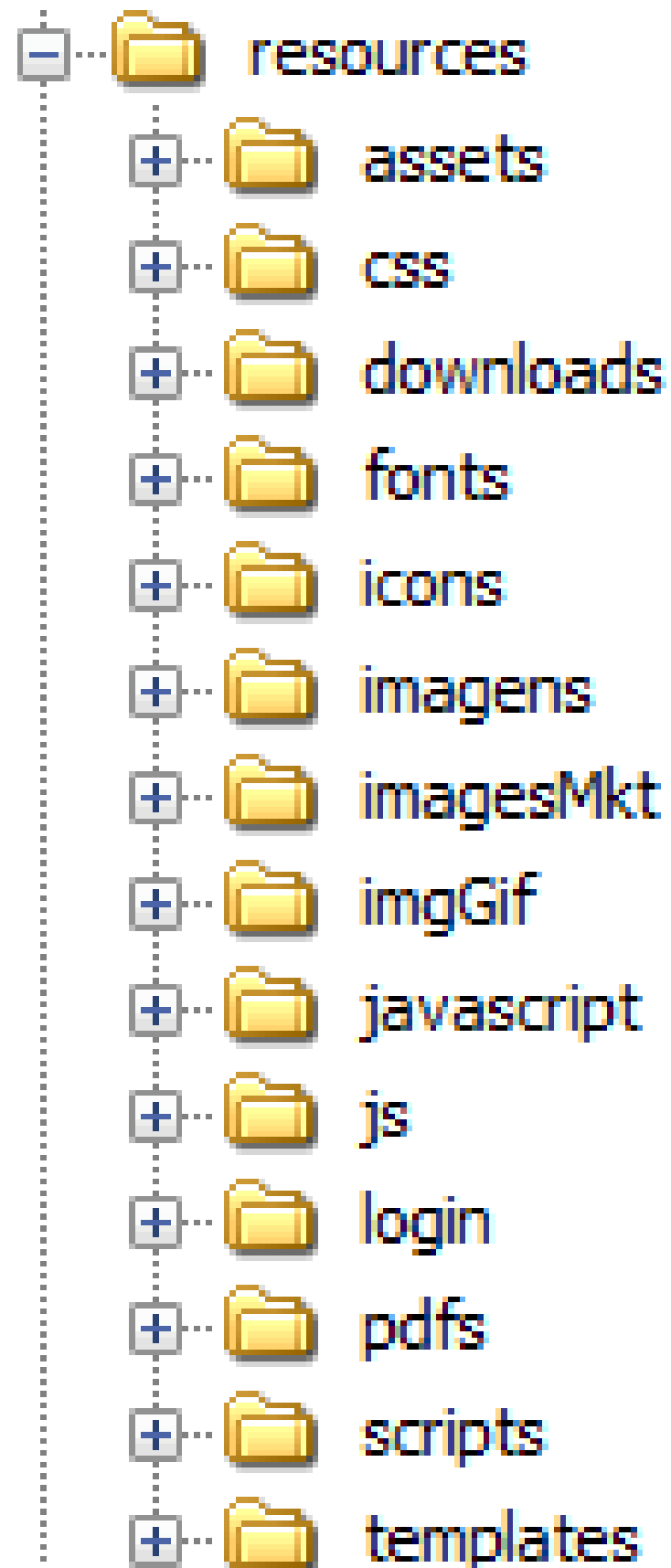
FUNCIONAMENTO JSF



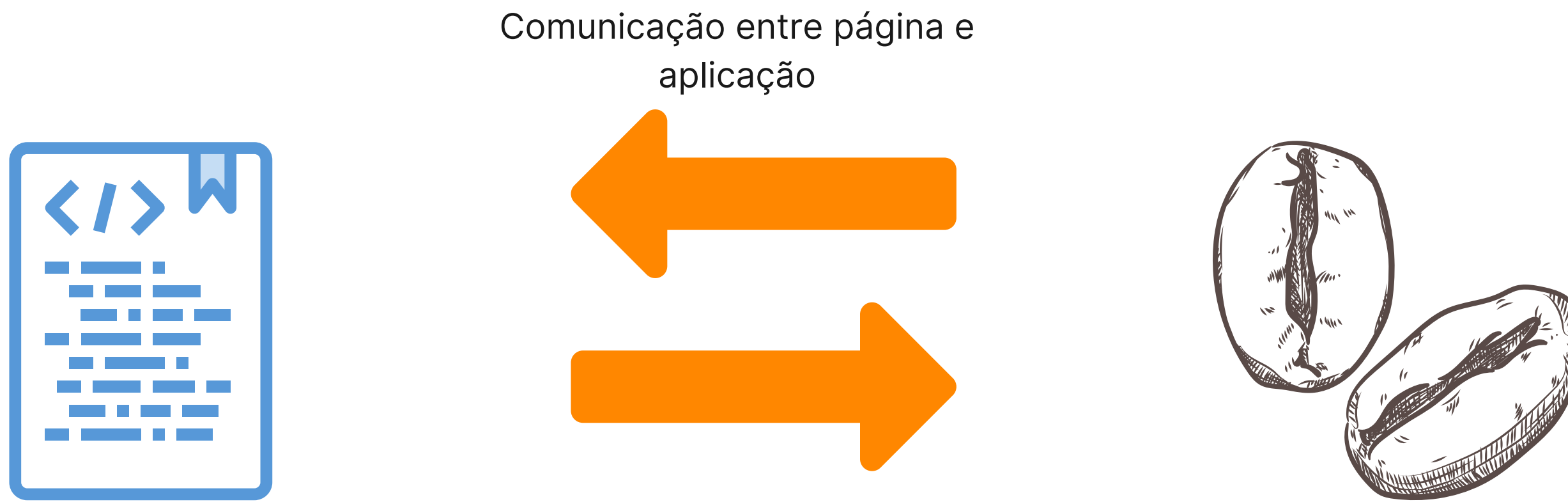
FUNCIONAMENTO JSF



FUNCIONAMENTO JSF



FUNCIONAMENTO JSF



xhtml - View

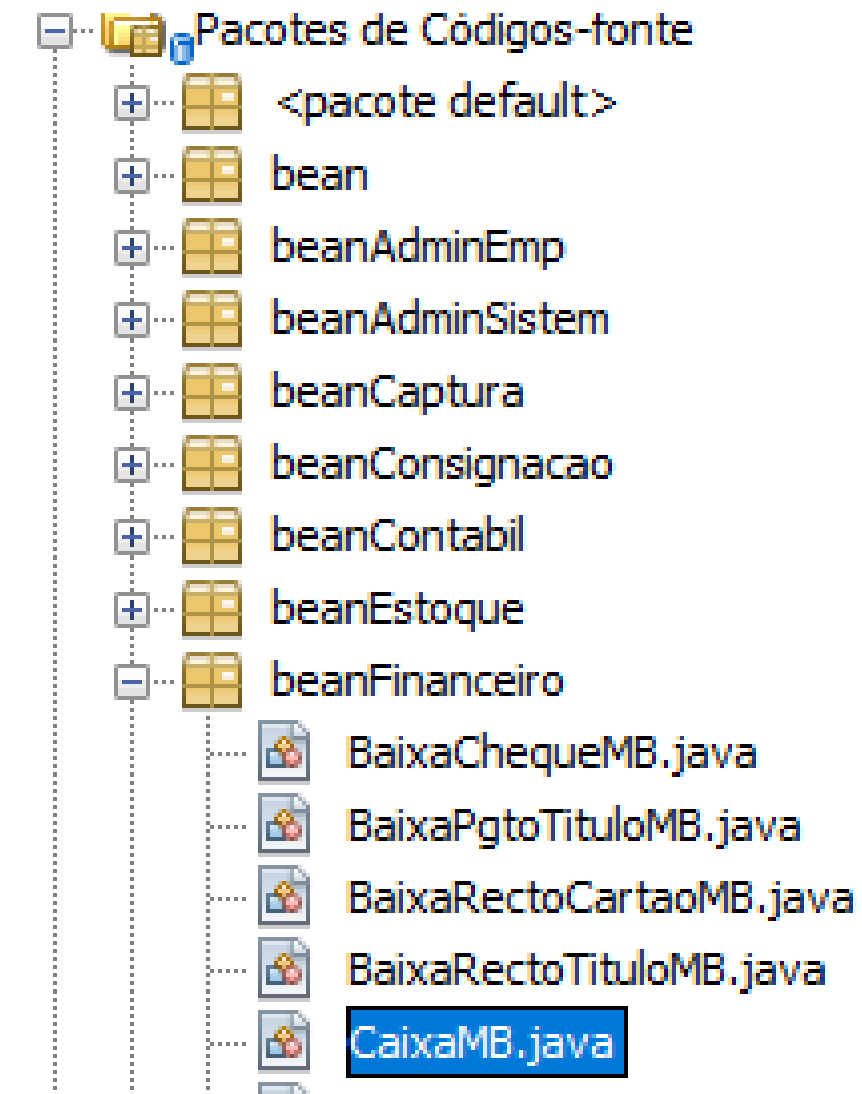
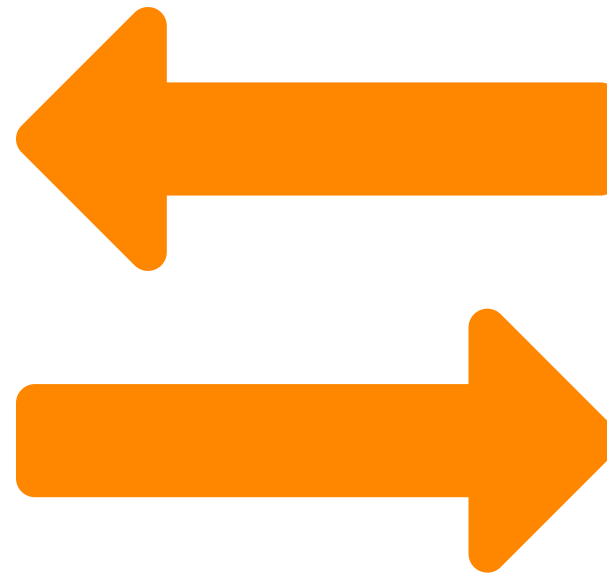
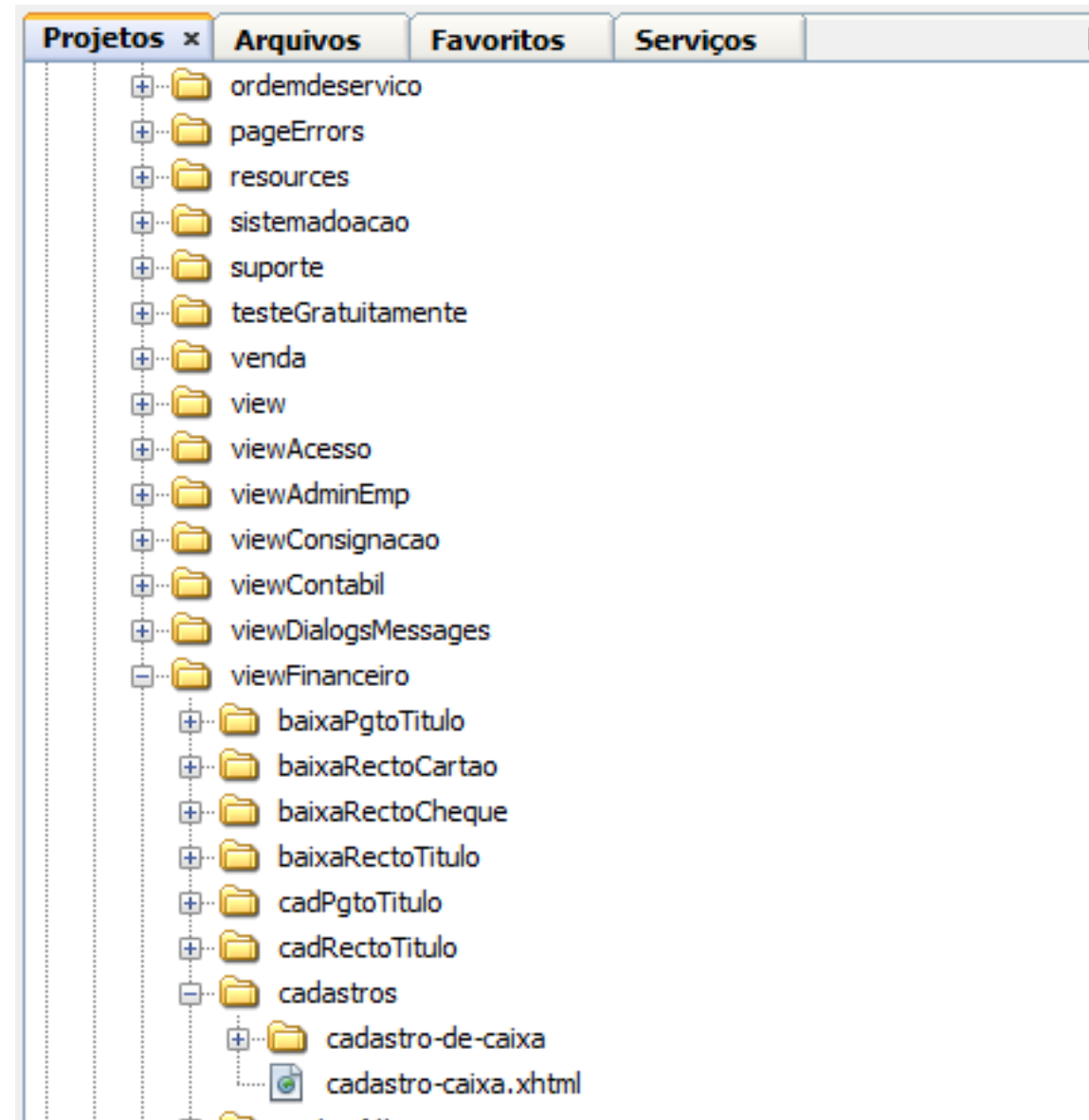
É um tipo de html com tags específicas

Managed Bean - Controller .java

Classe java que faz a comunicação com o xhtml

FUNCIONAMENTO JSF

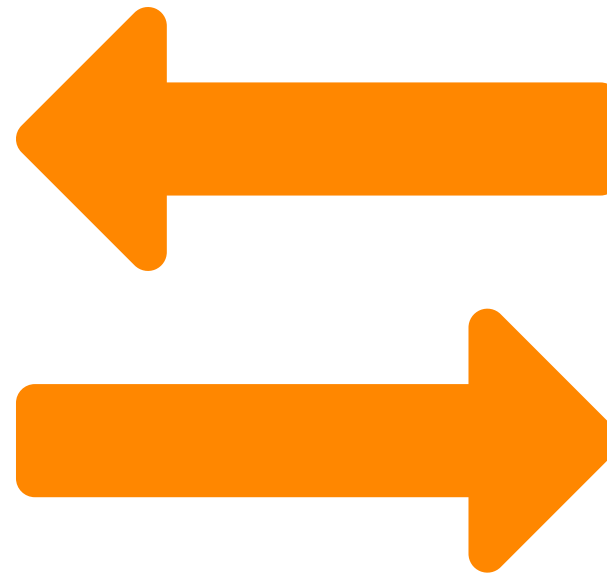
Comunicação entre página e aplicação



FUNCIONAMENTO JSF

Comunicação entre página e aplicação - Exemplo de chamada

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transit:
] <ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://xmlns.jcp.org/jsf/faces"
  template=".#{sessionScope.template}"
  xmlns:sgw-app="http://xmlns.jcp.org/jsf/sgw-app"
  >
]   <ui:define name="conteudo">
]     <sgw-app:content
]       managedBeanName="#{caixaMB}"
]     >
]     <sgw-app:panelCrud classPnCrud="pnCadCaixa"
]       headerPanel="Cadastro"
]       managedBeanName="#{caixaMB}"
]       <ui:include src="cadastro-de-caixa/faces"
]       <ui:include src="cadastro-de-caixa/faces"
]     </sgw-app:panelCrud>
]   </sgw-app:content>
] </ui:define>
</ui:composition>
```

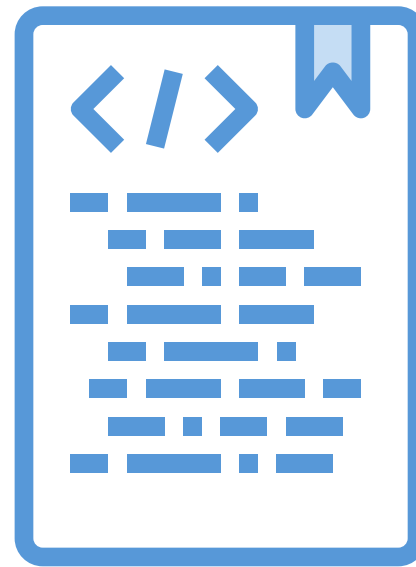


```
*/
@ManagedBean
@ViewScoped
public class CaixaMB extends PadraoAcessoObjMB implements

    private final Pagina pagina = Pagina.CAIXA;

    @PostConstruct
    private void init() {
        try {
            this.podeAcessar(pagina);
            super.viewInicio();
            this.iniciarListando = true;
            this.pesquisar();
        } catch (AcessoException ex) {
            messageERROR(ex);
        }
    }
}
```

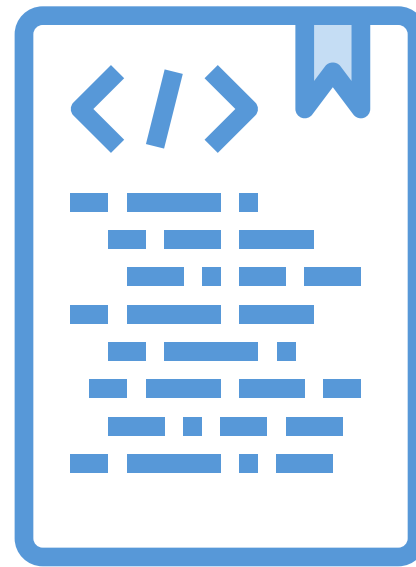

FUNCIONAMENTO JSF



O xhtml, é orientado a componentes podendo ser invocados com renderizadores e includes.

O xhtml, recebe o objeto ManagedBean, que é invocado pela linguagem EL (Expression Language) - {#,\$}

FUNCIONAMENTO JSF

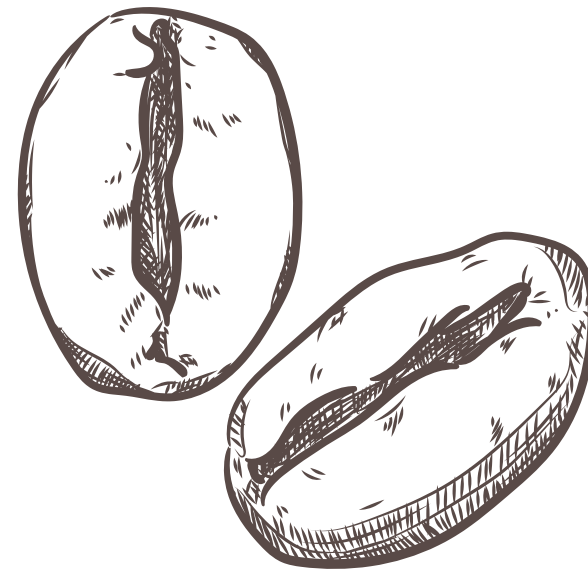


O xhtml, permite a construção de tags específicas pelas quais você pode adicionar várias Tags em uma única linha, e padronizar seu projeto como um todo.

Essas tags serão com espécies de funções, as quais você adiciona os parametros por você descritos e pronto o componente é criado.

Existem assim, grandes bibliotecas de componentes, a mais utilizada é a do Prime Faces, que possui boa documentação e com componentes responsivos e personalizaveis.

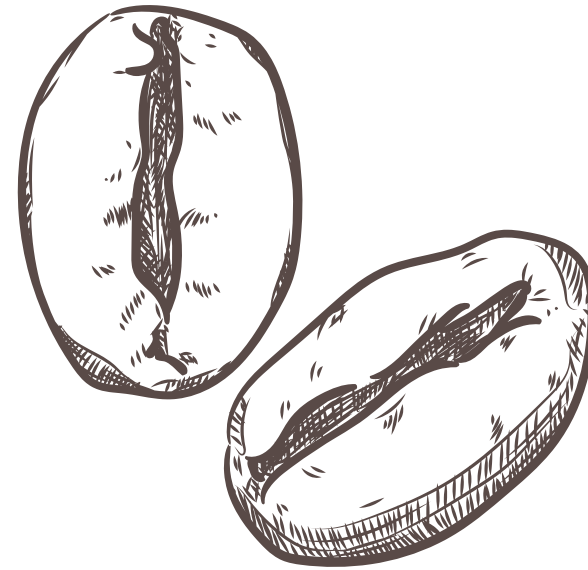
FUNCIONAMENTO JSF



Managed Bean - 'conversa' com o xhtml desde que tenha anotação na classe java do managed bean ou com anotações em xml.

Managed Bean - utiliza instancias criadas com escopos, sendo o mais usado no SGW o escopo view

FUNCIONAMENTO JSF



View Scopo - as variaveis existem enquanto você as visualiza ou o xhtml estiver sendo usado.

Session Scopo - as variaveis existem enquanto você a sessão é válida, ou seja, enquanto usuário está conectado.

Aplicação Scopo - as variaveis existem enquanto aplicação encontra-se implantada.

Request Scopo - as variaveis existem a cada request, sendo necessário recarregar muitas vezes, e manter identificadores na view para fazer esse recarregamento.

FUNCIONAMENTO JSF NO SGW

No Sgw, é trabalhado o esquema de visualização, usando renderizadores booleanos, as views são padronizadas nesse sentido.

Sendo assim não precisamos necessariamente atualizar toda a página para mudar o que o cliente vê, apenas setamos para true oque deve ser visualizado e false o que não deve.

FUNCIONAMENTO JSF NO SGW

Process e Update

Process = No process, você escolhe o que está na view que será enviado para o controller(managed bean)

Update= No update, você escolhe o que está na view que será atualizado após o **processamento**

FUNCIONAMENTO JSF NO SGW

Process e Update

No exemplo ao lado, foi adicionado um `inputText`
Nele podemos ver o funcionamento do Ajax

Uma vez apertado o botão ele irá fazer um process do
`inputText`, e um update do `outputText`, que retornara o input

Uma vez que não adicionemos o update, na tela será como se o
botão não surtisse efeito algum.

Basic

Name

Submit

Basic

Name

Pedro

Submit

Basic

Name

Pedro

Pedro

Submit

FUNCIONAMENTO JSF NO SGW

Process e Update

Events

KeyUp

Blur

Input

Nesses outros exemplo podemos ver opções caso não desejamos o botão, dessa forma usaremos os eventos
Toda vez que algum evento ocorrer, o process update (ajax) irá funcionar

Sendo alguns exemplos

KeyUp: quando qualquer tecla levantar, ou seja após apertada, e desapertada

Blur: Após clicar fora do inputText

Input: A cada tecla adicionada

FUNCIONAMENTO JSF NO SGW

Process e Update

Tambem é possível, em um form, carregar apenas algumas informações, sendo elas escolhidas no process update.

Firstname:

Middle:

Surname: *

All

Form

This

None

Parent

This Surname

templates/app.xhtml

Topo

menu side

conteudo

```

<h:body style="font-family:inherit">
  <p:tooltip id="toolTipGeral" hideEvent="click mouseout blur" escape="false"
    style="white-space:pre-line;margin-left: 250px;margin-right: 20px;"/>
  <header class="sgw-topo" >
    <a ...3 linhas />
    <o:importConstants type="sgw.VariaveisSistema"/>
    <o:importConstants type="estaticasAcesso.TpVersaoOS"/>
    
    <span style="color:white;margin-left: 5px;margin-bottom: 5px">SGW- Sistema de G
    ...31 linhas
    <p:selectOneMenu ...10 linhas />
    <p:column ...9 linhas />
  </header>
  <aside class="sgw-sidebar js-sgw-sidebar">
    <ui:include src="#{sessionScope.menuSide}"/>
  </aside>
  <section class="sgw-content js-sgw-content">
    <style ...12 linhas />
    <p:panel class="conteudo" style="padding-top: 10px" id="panelConteudo">
      <script ...13 linhas />
      <ui:include src="../../../viewDialogsMessages/dialogMessage.xhtml"/>
      <ui:insert name="conteudo"/>
    </p:panel>
    <p:ajaxStatus onStart="PF('block').show()" onSuccess="PF('block').hide()"
      onComplete="PF('block').hide()"
      />
    <p:blockUI block="@(.conteudo) formMenu" widgetVar="block" >
      <i class="fa fa-spinner fa-spin huge"></i>

```

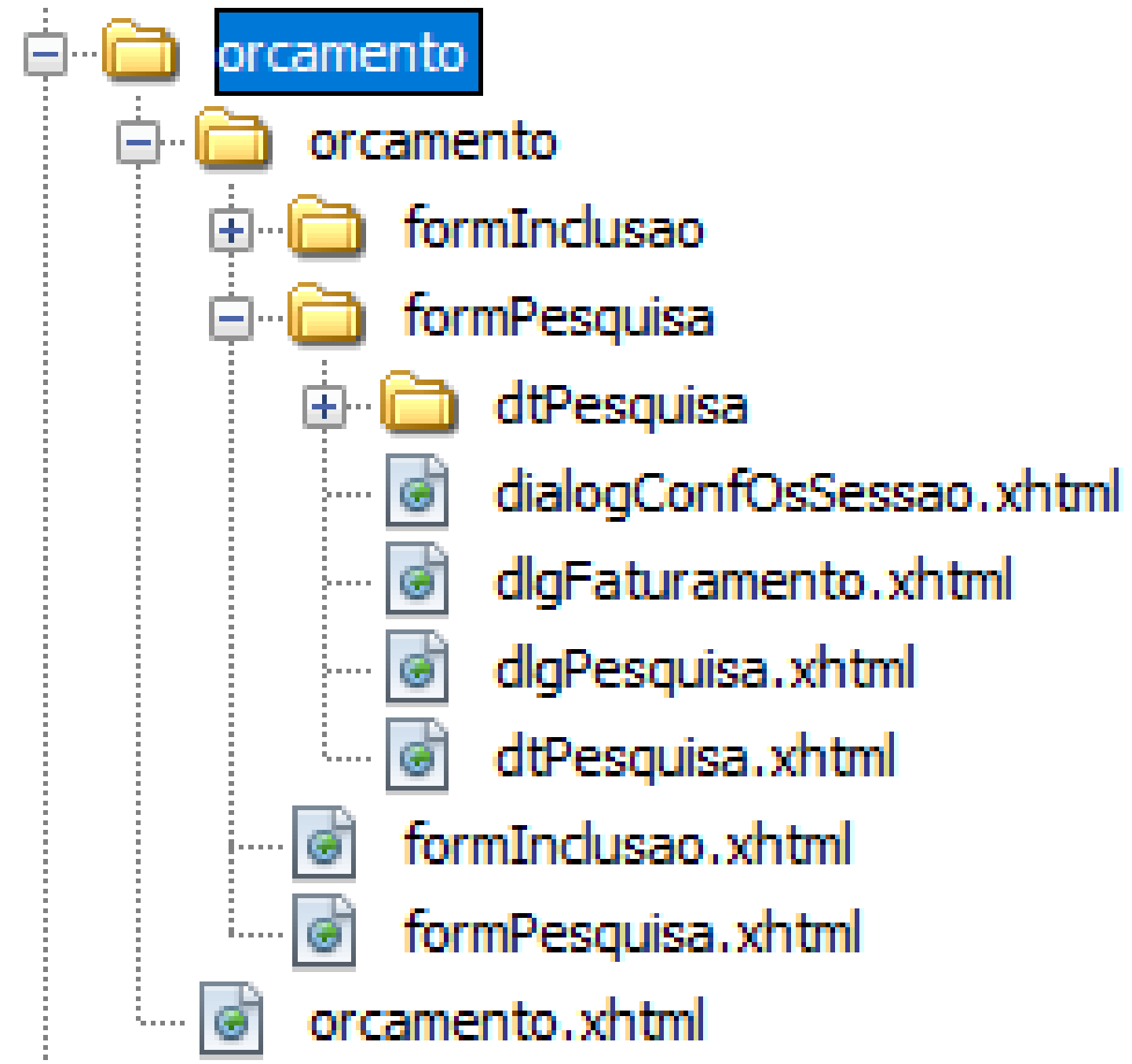
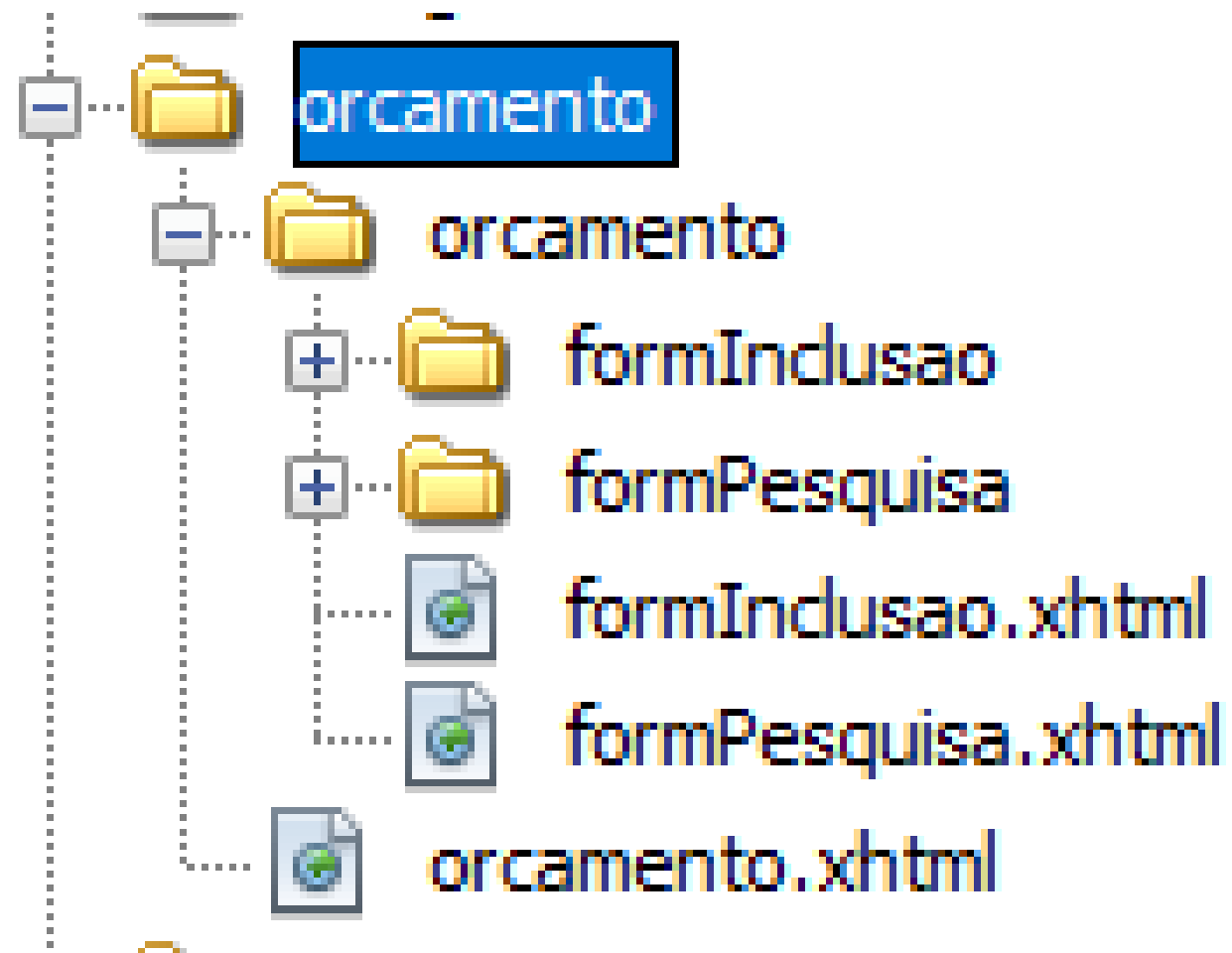
CONTEUDO

Arquivos .xhtml



CONTEUDO

Arquivos .xhtml



Form Pesquisa / render inicio

Form Inclusão/ render incluindo-alterando

Controller de views
que usa booleans
para fazer
renderização.

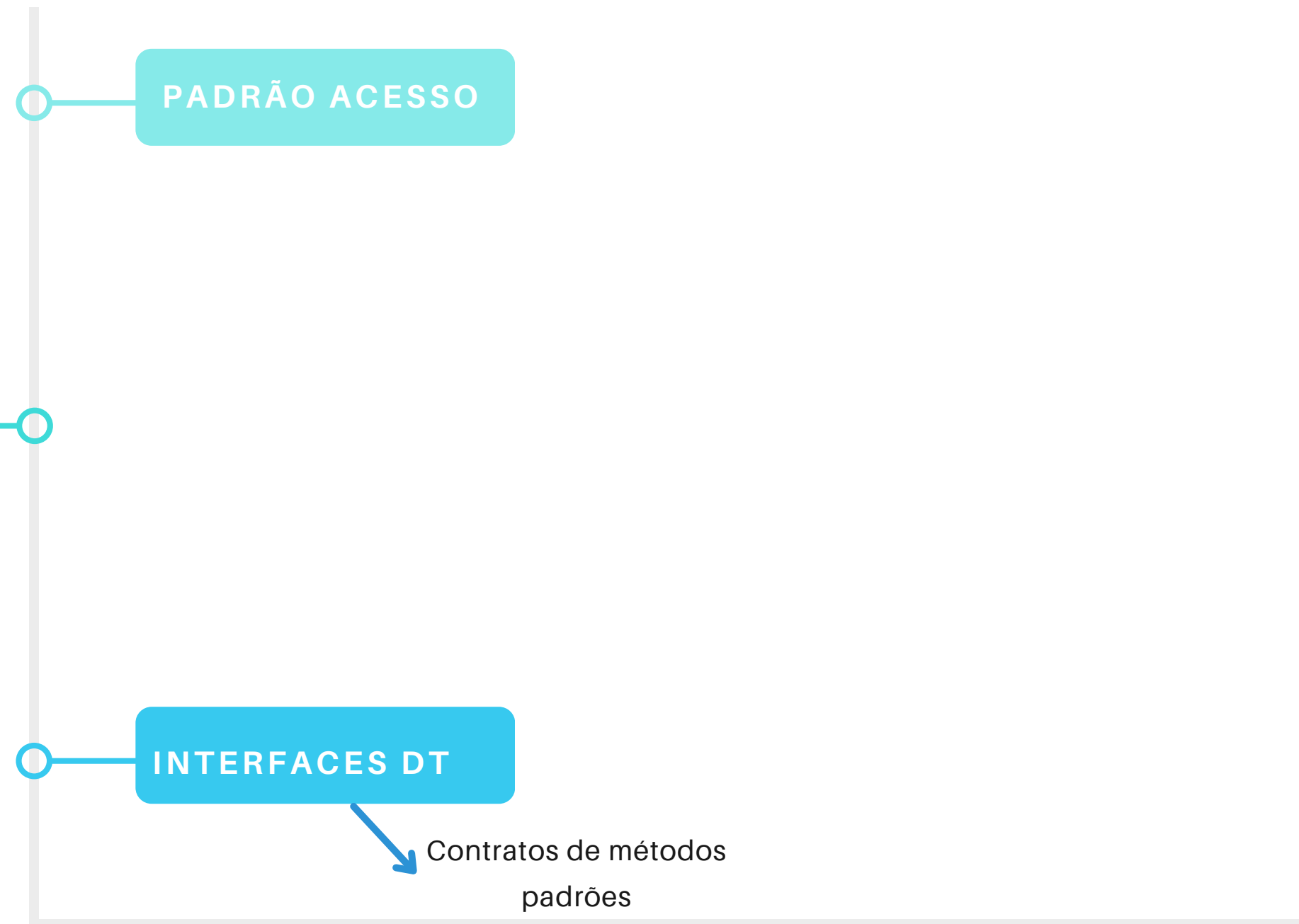
Extend de Faces Utils
que faz o controle de
utilitários de jsf, como
adicionar objeto em
sessão, mensagens,
etc.



**PADRÃO
BOOLEANS**

DINÂMICA DE VIEWS

SGW



PADRÃO BOOLEAN

Controle de renderização

```
public class PadraoBooleanMB extends FacesUtil {  
  
    protected boolean inicio = true;  
    protected boolean incluindo = false;  
    protected boolean alterando = false;  
    protected boolean excluindo = false;  
    protected boolean permitido = false;  
    protected boolean desabilitaInputs = false;  
    protected boolean desabilitaUnique = false;  
    protected boolean pesquisando = true;  
    protected boolean iniciarListando = false;  
    protected Participante partGer;  
    protected ParamPartGer paramPartGer;  
    protected Login login;  
    protected Object objMemoria;  
}
```

Onde:

partGer= empresa logada

paramPartGer=parametros da empresa logada

login = login do usuário logado

objMemoria = variável usada para guardar algum objeto na memória quando necessário

INTERFACES DT

Contratos de uso de métodos padrões com
DT(Data tables)

```
public interface InterfaceDtTableIndexMB {

    /** Prepara tela inicial de view ...3 linhas */
    public void viewInicio();

    /** Prepara view para inclusão de novo cadastro ...3 linhas */
    public void viewIncluir();

    /** Prepara view para alteração de cadastro ...6 linhas */
    public void viewAlterar(SuperTimeStampVersion beanEditar, int rowIndexDtTablePesquisa);

    /** Prepara view para copia de cadastro ...6 linhas */
    public void viewCopiar(SuperTimeStampVersion beanCopiado);

    /** Realiza operações de cancelamento e volta para a view inicio ...3 linhas */
    public void cancelar();

    /** Salva cadastro ...3 linhas */
    public void salvar();

    /** Altera cadastro ...3 linhas */
    public void alterar();

    /** Exclui cadastro ...6 linhas */
    public void excluir(SuperTimeStampVersion beanExcluido, int rowIndexDtTablePesquisa);

    public void pesquisar();

    /** Carrega todas listas necessárias para cadastro ...3 linhas */
    public void listasCrud();

    /** Verifica se é pertimitido o acesso pelo logado ...5 linhas */
    public void checarAcesso() throws AcessoException;

    void resetObjects() ;
}
```

PADRÃO DE MANAGED BEAN

Deve conter PadraoAcessoObjMB e
InterfaceDt

Exemplo:

```
] /**  
 *  
 * @author marlucio  
- */  
@ManagedBean  
@ViewScoped  
public class CaixaMB extends PadraoAcessoObjMB implements InterfaceDtTableIndexMB {  
  
    private final Pagina pagina = Pagina.CAIXA;  
  
    private int rowIndexDt;
```

Onde:

CaixaMB = managedbean que será invocado na view

Página = nome da página que é um enum que deve ser criado a cada nova página para dar permissões de acesso.

PADRÃO DE MANAGED BEAN

Deve conter PadraoAcessoObjMB e
InterfaceDt

Exemplo:

```
] /**  
 *  
 * @author marlucio  
- */  
@ManagedBean  
@ViewScoped  
public class CaixaMB extends PadraoAcessoObjMB implements InterfaceDtTableIndexMB {  
  
    private final Pagina pagina = Pagina.CAIXA;  
  
    private int rowIndexDt;
```

Onde:

CaixaMB = managedbean que será invocado na view

Página = nome da página que é um enum que deve ser criado a cada nova página para dar permissões de acesso.

PADRÃO DE MANAGED BEAN

Deve conter PadraoAcessoObjMB e
InterfaceDt

```
public class PadraoAcessoObjMB<T> extends PadraoAcessoMB {  
  
    protected T objPersist;  
    protected List<T> listObjPersist;  
    protected List<T> listObjPersistSelect;  
    protected List<T> listObjPersistFiltred;  
    protected int rowIndexDt;
```

Onde:

objPersist= objeto a ser inserido

listObjPersist= listObjetos que são carregados ao pesquisar registros no banco de dados

listObjPersistSelect= listObjetos que são selecionados na datatable pesquisa

listObjPersistFiltred= listObjetos que são filtrados na datatable pesquisa

rowIndexDt= indice do objeto persist na lista da datatable pesquisa

PADRÃO DE MANAGED BEAN

herança

```
public class PadraoAcessoMB extends PadraoBooleanMB implements Serializable {  
  
    private Pagina pagina;  
    private List<Participante> filiais;  
    private boolean indentar;  
    private boolean contabilidade;  
    private String senhaInformada;  
    private final static String SENHA_ADMIN_SISTEM_SESSAO = "$#%28381";  
}
```

Onde:

pagina= pagina a ser inserida em metodos de checar acesso

filiais= quando a empresa possui filias é armazenada nessa lista

indentar= boolean que se atribui que o participante é a própria indentar

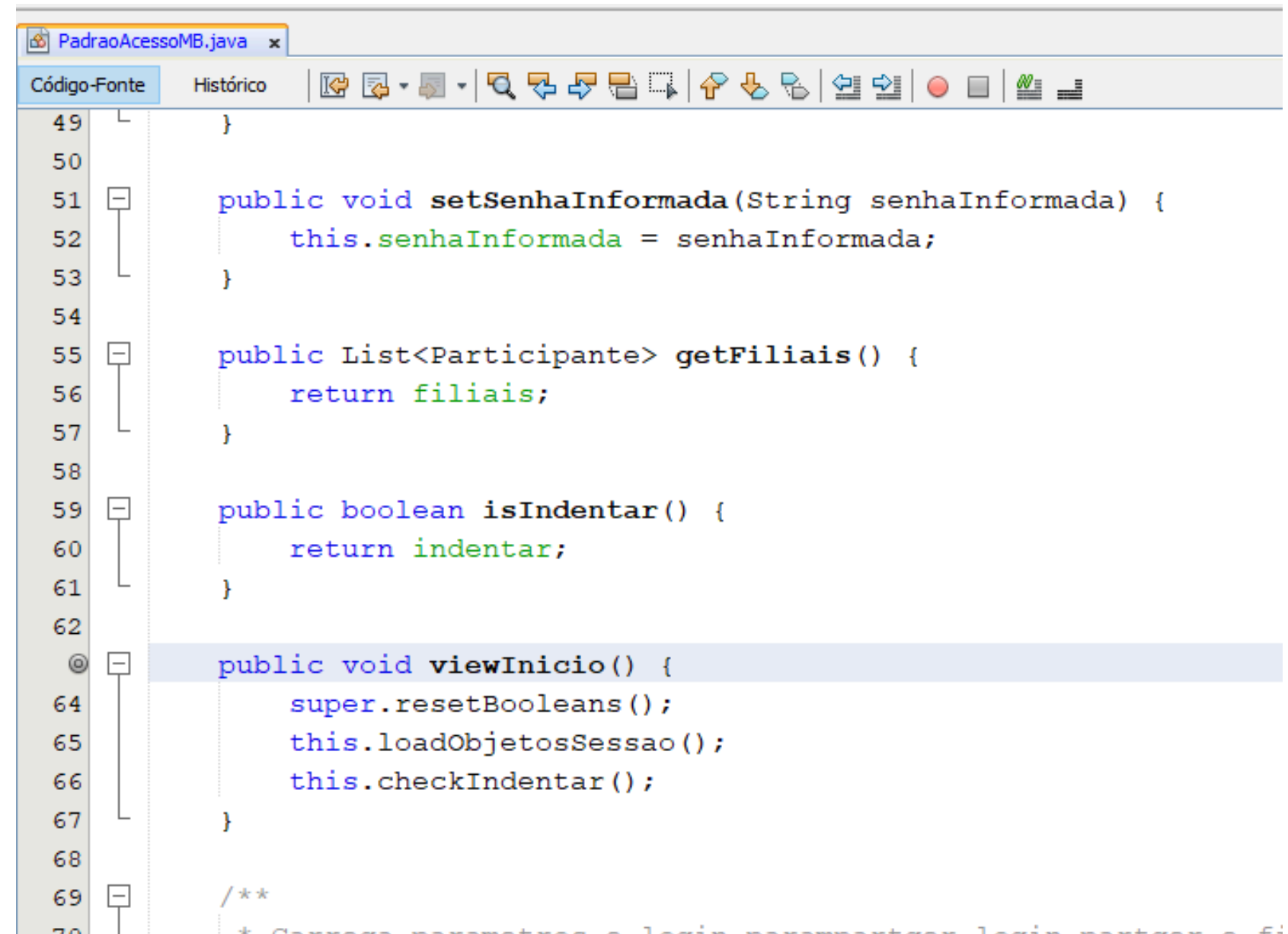
contabilidade= boolean que se atribui que o participante é uma contabilidade

senhaInformada=campo que identifica senha informada para suporte em qualquer usuário ou view, parametros e menus exclusivos para usuário de suporte

senha_admin_sistem_sessao=senha para ser checada com a senha informada

PADRÃO DE MANAGED BEAN

viewInicio()



```
49  }
50
51  public void setSenhaInformada(String senhaInformada) {
52      this.senhaInformada = senhaInformada;
53  }
54
55  public List<Participante> getFiliais() {
56      return filiais;
57  }
58
59  public boolean isIndentar() {
60      return indentar;
61  }
62
63  public void viewInicio() {
64      super.resetBooleans();
65      this.loadObjetosSessao();
66      this.checkIndentar();
67  }
68
69  /**
70   * Carrega parametros e login parametros login parametros e fi
```

Onde:

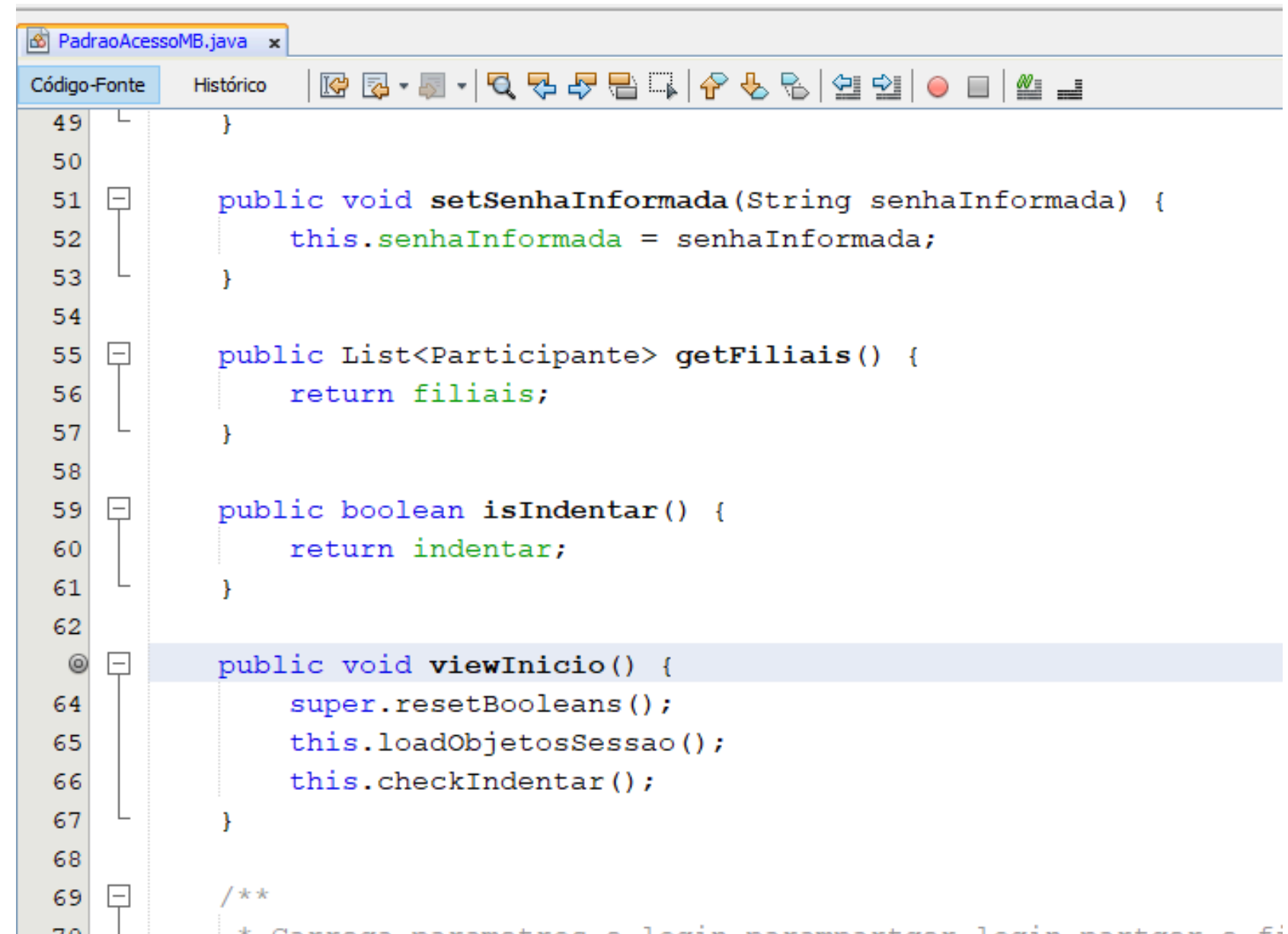
resetbooleans - atribui boolean para o estados inicial

loadObjetosSessao - atribui -partger e parampartger (parametros que indicam empresa logada e seus parametros)

checkIndentar - atribui boolean que informa que empresa logada é a própria indentar

PADRÃO DE MANAGED BEAN

viewInicio()



```
49  }
50
51  public void setSenhaInformada(String senhaInformada) {
52      this.senhaInformada = senhaInformada;
53  }
54
55  public List<Participante> getFiliais() {
56      return filiais;
57  }
58
59  public boolean isIndentar() {
60      return indentar;
61  }
62
63  public void viewInicio() {
64      super.resetBooleans();
65      this.loadObjetosSessao();
66      this.checkIndentar();
67  }
68
69  /**
70   * Carrega parametros e login parametros login parametros e fi
```

Onde:

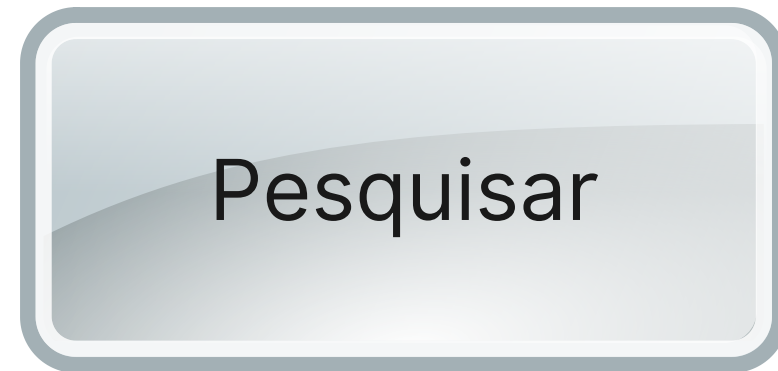
resetbooleans - atribui boolean para o estados inicial

loadObjetosSessao - atribui -partger e parampartger (parametros que indicam empresa logada e seus parametros)

checkIndentar - atribui boolean que informa que empresa logada é a própria indentar

FORM-PESQUISA

rendered=#{mb.inicio}



```
viewPesquisa  
PF('wDlgPesq').show()  
update dialogo pesquisa
```



```
viewIncluir()  
update=panel
```


Tags do JSF

O JSF permite que utilizamos tags de diferentes lugares, como por exemplo: Podemos escolher qual html usar como o html do próprio JSF que trazem consigo parâmetros específicos, ou o html 5 padrão.

Podemos também utilizar outras bibliotecas de tags com componentes por exemplo o facelets e o prime faces

Tags

Assim, utilizando tags de diferentes locais, precisamos especificar ao jsf de que biblioteca estamos falando, assim utilizamos identificadores antes das tags.

Os identificadores são definidos no início de cada página com a tag <html> com a lista de bibliotecas a ser utilizada, com seus respectivos links

```
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:h="http://xmlns.jcp.org/jsf/html">  
  <h:head>  
    <title>Facelet Title</title>  
  </h:head>  
  <h:body>  
    Hello from Facelets  
  </h:body>  
</html>
```

Composition

No sistema ao invés de utilizarmos o `<html>` utilizamos o `<ui:composition>` para fornecer o caminho das bibliotecas.

Composition

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"  
                xmlns:h="http://xmlns.jcp.org/jsf/html"  
                xmlns:p="http://primefaces.org/ui"  
                xmlns:f="http://xmlns.jcp.org/jsf/core"  
                xmlns:ui="http://xmlns.jcp.org/jsf/facelets"  
                template="#{sessionScope.template}"  
                xmlns:o="http://omnifaces.org/ui
```

Podemos verificar que utilizamos bibliotecas como primefaces, facelets, omnifaces e outras

Também é possível verificar que utilizamos um template para a página

Composition

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:p="http://primefaces.org/ui"
  xmlns:f="http://xmlns.jcp.org/jsf/core"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  template="#{sessionScope.template}"
  xmlns:o="http://omnifaces.org/ui">
```

Para utilizar uma biblioteca ou Framework, precisamos adicionar seu caminho, e uma identificação, que pode ser qualquer coisa.

Por conversão, utilizamos o padrão acima.

Primefaces

O Prime faces então é um FrameWork que contem componentes para o jsf que facilitam o trabalho do programador.

O identificador padrão para o PrimeFaces é <p:> e a biblioteca de componentes é gigante.

Contendo dês de botões prontos a visualizadores de pdf



Primefaces

Utilizando do prime faces, a única coisa que precisamos fazer é adicionar a tag, e os atributo. Cada componente terá o seus específicos, como id, valor, ícone, classe e style (css), e é claro o process update.

```
<p:commandButton id="btnAlterar" value="Alterar"  
icon="ui-icon-circle-check"  
class="sgw-btn sgw-btn-success btn-sm"  
style="margin: 1em 1em 1em 1em"  
process="@form"
```



Primefaces



Para saber melhor como manipular os componentes do Primefaces, fique sempre de olho em sua documentação, que trás consigo todos os componentes e o quais atributos podem ser adicionados a eles

3.22 CommandButton

CommandButton is an extended version of standard commandButton with ajax and theming.

Ajax Submit

Non-Ajax Submit

With Icon

Disabled

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	true	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	String	Label for the button
action	null	MethodExpr/ String	A method expression or a String outcome that'd be processed when button is clicked.
actionListener	null	MethodExpr	An actionlistener that'd be processed when button is clicked.
immediate	false	Boolean	Boolean value that determines the phaseId, when true

Criando Componentes

Como o prime faces, é possível que criemos componentes para o jsf. A indentar possui sua biblioteca que está em constante aprimoramento.

Para poder criar esses componentes, é necessário criar um arquivo xhtml, ao qual você adicione quais os atributos necessários para o uso do componente, e o que ele implementa

Criando Componentes

Para a criação, utilizamos a biblioteca composite, a qual utilizamos as tags `<comp:interface>` para adicionar os atributos do componente através da tag `<comp:interface>`

```
<comp:interface displayName="painel crud default" >
  <comp:attribute name="managedBeanName"
    shortDescription="Nome do managed bean. O objeto managed bean é o que será usado"
    required="true"
  />
  <comp:attribute name="classPnCrud"
    shortDescription="ID do panel"
    type="java.lang.String"
    required="true" method-signature="java.lang.String"
  />
  <comp:attribute name="classPnCadastro"
    shortDescription="ID do panel, classe para enviar o process"
    type="java.lang.String"
    required="true" method-signature="java.lang.String"
  />
</comp:interface>
```

Criando Componentes

É possível verificar, que alguns atributos possuem o atributo "required", basicmante ele diz se algum atributo é nescessário para o uso ou não. Sendo assim, se o required não for especificado, o componente não poderá ser usado

```
<comp:attribute name="classPnCrud"  
                shortDescription="ID do panel"  
                type="java.lang.String"  
                required="true" method-signature="java.lang.String"  
            />
```

Criando Componentes

E depois utilizamos a tag `implements`, com o que desejamos que a componente faça. Podemos adicionar componentes de outros Frameworks se quisermos.

```
<comp:implementation>  
  <p:commandButton value="Cancelar"  
    icon="ui-icon-cancel" id="btnCancelar"  
    class="sgw-btn sgw-btn-danger btn-sm"  
    process="@this"  
    title="Clique para cancelar cadastro/alteração"  
    update="@(.pnCadCaixa) "  
    tabindex="5"  
    accesskey="c"  
    actionListener="#{cc.attrs.managedBeanName.cancelar()}"  
  />
```

Criando Componentes

Um ponto muito importante é que para utilizarmos de algum atributo, precisamos utilizar cc.attrs. antes do próprio.

```
<comp:implementation>  
  <p:commandButton value="Cancelar"  
    icon="ui-icon-cancel" id="btnCancelar"  
    class="sgw-btn sgw-btn-danger btn-sm"  
    process="@this"  
    title="Clique para cancelar cadastro/alteração"  
    update="@(.pnCadCaixa) "  
    tabindex="5"  
    accesskey="c"  
    actionListener="#{cc.attrs.managedBeanName.cancelar()}"  
  />
```

Criando Componentes

E para utilizar, basta adicionar no composition

```
xmlns:pt= http://plantioes.org/pt/ui/talento  
xmlns:ui=http://xmlns.jcp.org/jsf/facelets  
xmlns:h=http://xmlns.jcp.org/jsf/html  
xmlns:f=http://xmlns.jcp.org/jsf/core  
xmlns:sgw-app=http://xmlns.jcp.org/jsf/composite/templates/sgw-application
```

```
>
```

```
<include src="../../resources/templates/headAndImports_novo.xhtml"/>  
<body>
```

Entidades e Hibernate

Hibernate é um FrameWork que consegue manipular o banco de dados através do java. Através dele podemos criar tabelas no banco a partir de objetos

Sendo assim, devemos adicionar alguns atributos nas classes dos objetos do sistema, a fim de que o hibernate saiba como manipulalos

Configurando o Hibernate

Hibernate é um FrameWork que consegue manipular o banco de dados através do java. Através dele podemos criar tabelas no banco a partir de objetos

Padrões DAO

Todas as classes deverão ter seu DAO, com construtores padrões, sendo um usado para criar listas e outro para pesquisa de objeto

```
+ /**...4 lines */
public class CaixaDAO extends PadraoDAO {

    //CONSTRUTOR OTIMIZADO PARA USO EM LISTAGENS
    private static final String SELECT_CONSTRUTOR_LISTAR = "select cx.id,"
        + "cx.num,cx.nome,cx.tpCaixa  from Caixa cx ";
    //CONSTRUTOR OTIMIZADO PARA USO DO OBJETO
    private static final String SELECT_CONSTRUTOR_PESQUISAR = "Select cx.id,"
        + "cx.num,cx.nome,cx.tpCaixa,contaDeb.id from Caixa cx "
        + "left join cx.contaDeb contaDeb ";

+ public Caixa loadT(Long idCaixa) throws ExceptionBancoDeDados {...7 lines }

+ public Caixa pesquisar(Integer num, String nome, Participante partGer) throws ExceptionBancoDeDa

+ public List<Caixa> listar(Participante partGer) throws ExceptionBancoDeDados {...7 lines }

+ public List<Caixa> listarLike(String query, Participante partGer, TpCaixa tpCaixa) throws Except

+ /** Caixa result transformers ...4 lines */
+ public static class CaixaRT {...44 lines }
}
```

Padrões DAO

E para esses construtores, cada um deve ter um resulttransforme para construção do objeto ao pesquisar.

```
/** Caixa result transformers ...4 lines */  
public static class CaixaRT {  
  
    public static ResultTransformer resultTransformListar() {...19 lines }  
  
    public static ResultTransformer resultTransformPesquisar() {...21 lines }  
}
```

Padrões DAO

Utilizando do hibernate, fazemos consultas no banco com o HQL, a sua linguagem, a qual é totalmente baseada em objetos.

```
public class CaixaDAO extends PadraoDAO {  
  
    private static final String SELECT_CONSTRUTOR_PADRAO = "select cx.id,"  
        + "cx.num,cx.nome,cx.tpCaixa  from Caixa cx ";  
  
    public Caixa pesquisar(Integer num, String nome, Participante partGer) {  
        String hql = SELECT_CONSTRUTOR_PADRAO;  
        hql += "where cx.partGer=:partGer and(cx.num=:num or cx.nome=:nome)  
        return addParamValue("partGer", partGer)  
            .addParamValue("num", num)  
            .addParamValue("nome", nome)  
            .addResultTransformer(CaixaRT.resultTransformerPadrao())  
            .pesquisarPorQueryParam(hql);  
    }  
}
```

Padrões DAO

Com o addParamValue, podemos adicionar ao hql os próprios objetos da consulta.

Nesse caso, um integer, uma string e um objeto Participante.

```
public class CaixaDAO extends PadraoDAO {  
  
    private static final String SELECT_CONSTRUTOR_PADRAO = "select cx.id,"  
        + "cx.num,cx.nome,cx.tpCaixa  from Caixa cx ";  
  
    public Caixa pesquisar(Integer num, String nome, Participante partGer) {  
        String hql = SELECT_CONSTRUTOR_PADRAO;  
        hql += "where cx.partGer=:partGer and(cx.num=:num or cx.nome=:nome)"  
        return addParamValue("partGer", partGer)  
            .addParamValue("num", num)  
            .addParamValue("nome", nome)  
            .addResultTransformer(CaixaRT.resultTransformerPadrao())  
            .pesquisarPorQueryParam(hql) ;  
    }  
}
```

Result Transformer (RT)

O resulta trasformer é utilizado para poder, transformar o objeto que é criado pelo hibernate e uma consulta banco, no objeto desejado

```
public static class CaixaRT {  
  
    public static ResultTransformer resultTransformerPadrao() {  
        return new ResultTransformer() {  
  
            @Override  
            public Object transformTuple(Object[] os, String[] strings)  
            {  
                Caixa caixa = new Caixa((Long) os[0])  
                    .comNum((Integer) os[1])  
                    .comNome((String) os[2])  
                    .comTpCaixa((TpCaixa) os[3]);  
                return caixa;  
            }  
  
            @Override  
            public List transformList(List list) {  
                return list;  
            }  
        };  
    }  
}
```

Como é mostrado no exemplo anterior, o ResultTransformer é adicionado a consulta.

Regra de negócios (RN)

As regras de negocio são responsáveis para poder impedir que os bugs aconteçam, basicamente, ao salvar, alterar ou criar alguma coisa no sistema, as regras de negocio devem ser consultadas anteriormente, para impedir problemas futuros.

```
public class CaixaRN extends CaixaDAO {  
  
    public void salvar(Caixa caixa, ParamPartGer paramPartGer)  
    {  
        this.checkParametros(paramPartGer);  
        CaixaValidadorBD.validaCamposObrigatorios(caixa);  
        this.checkDuplicidade(caixa, paramPartGer);  
        this.configuraConta(caixa, paramPartGer);  
        salvar(caixa);  
    }  
}
```

Regra de negócios (RN)

Como é possível visualizar, o RN do caixa checa todos os parametros, valida os campos obrigatorios, verifica a duplicidade. Tudo isso antes de salvar.

```
public class CaixaRN extends CaixaDAO {  
  
    public void salvar(Caixa caixa, ParamPartGer paramPartGer)  
    {  
        this.checkParametros(paramPartGer);  
        CaixaValidadorBD.validaCamposObrigatorios(caixa);  
        this.checkDuplicidade(caixa, paramPartGer);  
        this.configuraConta(caixa, paramPartGer);  
        salvar(caixa);  
    }  
}
```

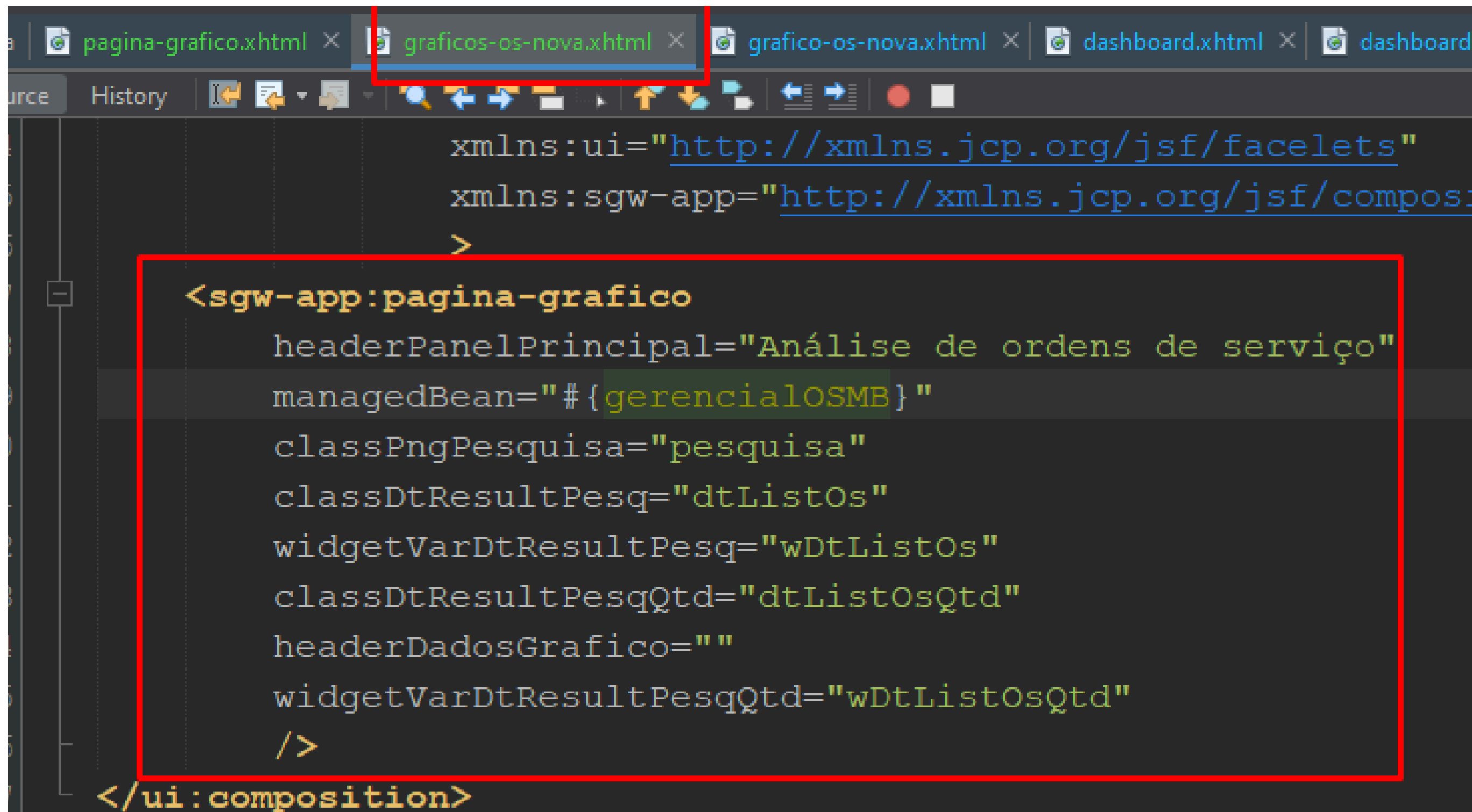
Criação de gráficos

Deve estender de graficoMB para pegar atributos
e implementar de graficomethods para usar
componente criado para pagina padrão

```
*  
* @author adm  
*/  
@ManagedBean  
@ViewScoped  
public class GerencialOSMB extends GraficoMB implements GraficoMetodos {
```


Criação de gráficos

Exemplo de uso



The image shows a web browser window with several tabs. The active tab is 'graficos-os-nova.xhtml', which is highlighted with a red box. The browser's address bar shows the URL 'http://localhost:8080/...'. The page content is a JSP page with XML namespace declarations and a highlighted code block. The code block is enclosed in a red box and contains the following XML snippet:

```
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
xmlns:sgw-app="http://xmlns.jcp.org/jsf/composi
>

<sgw-app:pagina-grafico
  headerPanelPrincipal="Análise de ordens de serviço"
  managedBean="#{gerencialOSMB}"
  classPngPesquisa="pesquisa"
  classDtResultPesq="dtListOs"
  widgetVarDtResultPesq="wDtListOs"
  classDtResultPesqQtd="dtListOsQtd"
  headerDadosGrafico=""
  widgetVarDtResultPesqQtd="wDtListOsQtd"
/>

</ui:composition>
```

PADRÃO CRIAÇÃO DE NOVO CADASTRO - TABELA OU VIEW

1

CLASSE ENTIDADE

Criar uma classe para entidade do banco de dados, mapeando a mesma com anotação e no xml do hibernate.

2

CRIAR DAO PARA ENTIDADE

Criar classe de consulta ao banco para entidade, seguindo exemplos de outras classes e herdando de PadraoDAO.java

3

CRIAR RN PARA ENTIDADE

Criar regra de negócios para entidade, onde serão criados os métodos cruds com validações

4

FAZER TESTES LOCAIS

Gerar testes de métodos criados em DAO e RN para preparar códigos para view e controller

5

CRIAR MANAGED BEAN

Criar managed bean, para inserção e consulta de cadastros

6

COPIAR VIEW COMPATÍVEL

Nesse momento se escolhe tela compatível, a padrão é a de cadastro de caixa, mas dependendo da situação, deve se escolher outra.

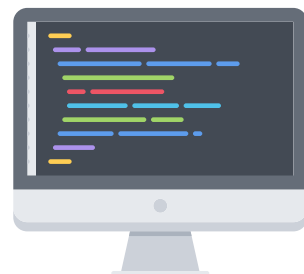
Sendo 1-3 Model
5 - controller
6 - View

PADRÃO PARA IMPLANTAÇÃO DE NOVOS MÉTODOS



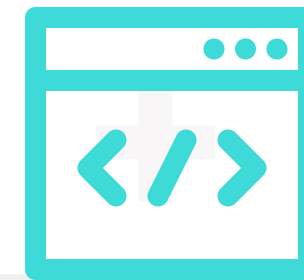
VERIFICAR SE JÁ EXISTE ALGUM PARECIDO

Pesquisar algum método
para usar de exemplo



CRIAR MÉTODO

Criar a método
verificando
detalhadamente a
aplicação de padrões já
usados no sistema, com
models, filtros e result
transformers.



TESTE O CÓDIGO LOCAL

Teste o código
isoladamente, sem
implantar junto com
outros métodos, para ver
se está funcionando.



IMPLANTE E TESTE

Implante o método e
teste novamente,
verificando todos
elementos que podem ser
influenciados.

