



building enterprise applications in simple way.

Guia de Uso 1.8.1 - Português

Autor

Ayslan Macedo

Sumário

1. Introdução	7
1.1. O que é o xpert-framework?	7
1.2. Xpert Sistemas	7
1.3. Sobre esta documentação	7
1.4. Principais mudanças na versão 1.8.1	7
1.5. Guia de migração para a versão 1.8.1	8
2. Padrão de documentação	9
3. Links úteis	10
4. Configuração do xpert-framework	11
4.1. Download	11
4.2. Dependência maven	11
4.3. Dependências	11
4.4. Código fonte	13
5. Configurando o arquivo xpert-config.xml	14
5.1. Visão Geral	14
5.2. Configurando o EntityManagerFactory	14
5.2.1. Introdução	14
5.2.2. Exemplo de implementação	14
6. Utilitários	16
6.1. DateUtils (Utilitário para operações com data)	16
6.2. CollectionsUtils (Utilitário para operações com coleções)	16
7. Xpert-Faces	18
7.1. Introdução	18
7.2. Conversores	18
7.2.1. Entity Converter	18
7.2.2. Entity Converter List	18
7.2.3. CPF Converter	19
7.2.4. CNPJ Converter	19
7.2.5. YesNoConverter	19
7.2.6. ActiveInactiveConverter	19
7.3. Componentes JSF	20
7.3.1. BooleanSelectItens	20
7.3.2. Confirmation	21
7.3.3. CPF e CNPJ	22
7.3.4. DataTableActions	23
7.3.5. DataTableExporter	24
7.3.6. DateFilter	27
7.3.7. Download	28

7.3.8. Filter On Enter	29
7.3.9. Find All Bean	30
7.3.10. Group	32
7.3.11. Initializer	37
7.3.12. Input Number	39
7.3.13. Legends	40
7.3.14. Mask	41
7.3.15. Modal Messages	42
7.3.16. PdfPrinter	43
7.3.17. RestorableFilter	45
7.3.18. Spread Checkbox/Radio	46
7.3.19. Sum, Avg, Min e Max	47
8. Internacionalização de mensagens do BeanValidation	51
8.1. Introdução	51
8.2. Configuração	51
8.3. Como é feita a internacionalização	51
8.4. Tipos de validações suportadas	52
9. Artefatos padronizados para CRUDs	54
9.1. Classe de negócio - AbstractBusinessObject	54
9.1.1. Estrutura de um BusinessObject	54
9.1.2. Principais métodos da classe	55
9.2. ManagedBean (Controller) - AbstractManagedBean	57
9.2.1. Estrutura de um ManagedBean	57
9.2.2. Principais métodos da classe	58
10. Tratamento de Regra de Negócio com o Business Exception	62
10.1. Validação Básica	62
10.2. Validação com Múltiplas regras	62
10.3. Passagem de Parâmetros na mensagem	63
11. Exibindo mensagem com o FacesMessageUtils	64
11.1. Utilização Básica	64
11.2. Passagem de Parâmetros na mensagem	64
11.3. Business Exception em um FacesMessagesUtils	64
12. Restrictions	65
12.1. Introdução	65
12.2. Tipos de Restrictions	65
12.3. Utilizando métodos em cadeia	66
12.4. Restrictions - Utilizando a cláusula “OR”	67
13. Consultas dinâmicas com o QueryBuilder	69
13.1. Introdução	69

13.2. Criando um QueryBuilder	69
13.3. Seleccionando todos os registros	69
13.4. Ordenação do Resultado	69
13.5. Seleccionando um resultado único	69
13.6. Utilização de Restrictions	69
13.7. Definindo o Alias	70
13.8. Joins	70
13.9. Select Distinct	70
13.10. Definindo a quantidade máxima de resultados	71
13.11. Consulta paginada	71
13.12. Debug da Query	71
13.13. Definindo os atributos na query	71
13.14. Realizando o “count” dos registros	71
13.15. Somatório com o “sum”	72
13.16. Cláusula Max	72
13.17. Cláusula Min	72
13.18. Cláusula AVG	72
14. DAO Genérico - BaseDAO	73
14.1. Introdução	73
14.2. Métodos do BaseDAO	73
14.2.1. save	73
14.2.2. merge	73
14.2.3. saveOrUpdate	73
14.2.4. update (Implementação do Hibernate)	74
14.2.5. saveOrUpdate (Implementação do Hibernate)	74
14.2.6. find	74
14.2.7. listAll	74
14.2.8. list	74
14.2.9. unique	75
14.2.10. count	75
14.2.11. listAttributes	75
14.2.12. findAttribute	76
14.2.13. findList	76
14.2.14. getInitialized	76
14.2.15. delete	76
14.2.16. remove	76
14.2.17. Query personalizada	77
14.3. Passando Parâmetros nos métodos do BaseDAO	77
14.3.1. Map<String, Object>	77
14.3.2. Restriction e Restrictions	78
15. DataTable paginado no banco com o LazyDataModelImpl	79

15.1. Utilização Básica	79
15.2. Adicionar restrições na consulta do LazyDataModel	79
15.3. Filtros da coluna	79
15.4. Recuperar todos os registros de maneira não pagina	80
15.5. Manipulando a Ordenação com o OrderByHandler	80
15.6. Manipulando as restrições com o FilterByHandler	81
15.7. Definir se os dados serão ou não carregados	83
15.8. Definindo quando realizar o count com o LazyCountType	83
16. Definindo os joins com o JoinBuilder	84
16.1. Introdução	84
16.2. Entendendo o problema dos joins	84
16.3. Utilizando os joins no LazyDataModelImpl	85
16.4. Utilizando o JoinBuilder no AbstractBaseBean	86
17. Unicidade dos campos com UniqueField	87
17.1. Definindo campos únicos em um BO genérico	87
17.2. Customização de mensagem no UniqueField	87
17.3. Validação fora de um BO Genérico	88
17.4. Utilizando Restrictions com o UniqueFields	88
18. Criação de relatórios com o FacesJasper	89
19. Geração de Código (CRUD) com o Xpert-Maker	90
19.1. Introdução	90
19.2. Integração com o framework bootstrap	90
19.2.1. Inserindo o bootstrap no seu projeto	90
19.2.2. Classes CSS utilitárias do xpert-framework	90
19.2.3. Estrutura de geração dos formulários usando o bootstrap	91
19.3. Utilização Básica via componente Swing	92
19.3.1. Estrutura da classe para geração via componente Swing	92
19.3.2. Passo a passo da geração de classes via componente Swing	94
19.4. Utilização Básica via componente JSF	100
19.5. Estrutura de um CRUD gerado	104
19.5.1. Artefatos criados para cada Entidade	104
19.5.2. Artefatos únicos	105
19.5.3. Padrão da view para listagem de Registros - list{entidade}.html	105
19.5.4. Padrão da view para detalhamento do registro – detail{entidade}.html	107
19.5.5. Padrão da view para criação e edição do registro – create{entidade}.html	107
19.6. Atributos mapeados e seus respectivos componentes na View	109
19.7. Removendo o campo “id” da URL ao editar o Objeto	109
20. Auditando as entidades com o Xpert-Audit	111

20.1. Introdução	111
20.2. AbstractAuditng - Classe que representa uma auditoria feita	111
20.2.1. Introdução	111
20.2.2. Atributos	112
20.2.3. Configuração	112
20.2.4. Exemplo de uma implementação	112
20.3. AbstractMetadata - Classe que representa os metadados	113
20.3.1. Introdução	113
20.3.2. Atributos	114
20.3.3. Configuração	114
20.3.4. Exemplo de uma implementação	114
20.4. Listener para a auditar um objeto	115
20.4.1. Introdução	115
20.4.2. Configuração	115
20.4.3. Exemplo de uma implementação	115
20.5. Auditando uma entidade	116
20.5.1. Auditar um Insert	116
20.5.2. Auditar um Update	116
20.5.3. Auditar um Delete	116
20.5.4. BaseDAOImpl do xpert-framework e sua auditoria	116
20.6. Exibindo a auditoria de uma determinada entidade	117
20.7. Internacionalizar valores dos campos na Auditoria	117
20.8. Ignorar auditoria de uma classe ou de um atributo	118
20.9. Acessando alterações do objeto com o AuditContext	118
21. SequenceUpdater - Atualizando as sequences do banco de dados	120
21.1. Introdução	120
21.2. Utilização do SequenceUpdater	120
21.2.1. OracleSequenceUpdater	120
21.2.2. PostgreSQLSequenceUpdater	120
21.3. Adicionando o atualizador de sequence no projeto base (criado do archetype maven)	120
22. Controle de Acesso através do xpert-security	122
22.1. Introdução	122
22.2. Gerenciamento da Sessão a partir da classe SecuritySessionManager	122
22.3. Bean para manter usuário e permissões na sessão	123
22.4. Filtro para bloquear o acesso do usuário por página	124
22.5. Bean para realizar login/logout do usuário na aplicação	125
22.6. Login utilizando SecurityLoginBean	127
22.7. SecurityArea para verificação de acesso a nível de componente	127
23. Arquétipo para criação de projeto	128
23.1. Introdução	128

23.2. Arquétipos disponíveis do xpert-framework	128
23.3. Características de um projeto gerado pelo arquétipo	128
23.4. Configuração de banco de dados	129
23.4.1. Mapemanto das entidades	129
23.4.2. Tipo texto no banco de dados	130
23.4.3. Geração do Data Source no glassfish	130
23.5. Arquétipo maven EAR do xpert-framework	131
23.5.1. Como criar um projeto a partir do arquétipo	131
23.5.2. Executando a aplicação	134
23.6. Estrutura de um projeto gerado a partir do arquétipo maven	135
23.6.1. Main	135
23.6.2. EJB	136
23.6.3. WAR	137
24. Boas práticas para o mapeamento de entidades	138
25. Possíveis problemas ao utilizar o xpert-framework	141
25.1. O glassfish não encontra o Data Source do glassfish-resources.xml	141
25.2. A aplicação não executa no glassfish 4.x, apenas no 3.x	141
25.3 Usando o SQL Server o LazyDataModelImpl não faz a paginação e lança um erro	141
25.4 Ao fazer login no projeto do archetype a tela aparece em branco e um erro é lançado no glassfish 4	142
26. FAQ	143

1. Introdução

1.1. O que é o xpert-framework?

O xpert-framework surgiu com o objetivo de facilitar o desenvolvimento com JavaEE. Ao longo da experiência prática de desenvolvimento de projetos grandes surgiram problemas e a necessidade de alguns incrementos que a especificação do JavaEE e suas implementações não fornecem.

Além de JavaEE, o xpert-framework é fortemente ligado ao framework hibernate e ao primefaces, por isso ele também provê utilitários nessas tecnologias.

Esse framework pode ser considerado como um conjunto de componentes e utilitários, pois o núcleo do desenvolvimento nos projetos que utilizam ele continua sendo a linguagem java e especificações da tecnologia JavaEE.

1.2. Xpert Sistemas

A Xpert Sistemas é uma empresa brasileira de desenvolvimento de software e consultoria especializada em JavaEE, localizada em Teresina, Piauí. Desde 2009, estamos implementando soluções maduras e confiáveis para o mercado corporativo.

Conheça a Xpert Sistemas e nossos produtos no site <http://www.xpertsistemas.com.br>.

1.3. Sobre esta documentação

Esta documentação trata da versão 1.8.1 do xpert-framework. O objetivo dela é tratar de maneira detalhada aquilo que o framework disponibiliza.

Durante os capítulos seguintes serão exibidos exemplos de códigos e utilização do que o framework fornece. São feitas menções ao javaee, pois o framework se baseia fortemente nelas então para dúvidas sobre a especificação javaee ou outra tecnologia como hibernate ou primefaces é aconselhável consultar a documentação das respectivas tecnologias.

Alguns exemplos podem ser vistos no showcase do framework, o link para ele está disponível a sessão links úteis.

1.4. Principais mudanças na versão 1.8.1

A versão 1.8 apresenta algumas correções e melhorias

- Tipo BigInteger não estava mapeado no xpert-maker
- Criação da classe DateUtils para operação com datas
- Novo conversor: entityConverterList (conversor para pegar uma entidade dentro de uma lista)
- Mudança dos artefatos do JSF de anotação para o **faces-config.xml**
- Remoção de referências a classe EntityManagerImpl, por conta de incompatibilidade com versões mais recentes do hibernate
- Criação do método **saveOrMerge** no **BaseDAO**
- Novo componente: **dataTableExporter**
- Novo componente: **dataTableActions**

- Novo componente: **group** (para agrupamento de listas)
- Novo componente: **pdfPrinter** (geração de um PDF através de um conteúdo HTML)
- Nova função: **sum** (para somatório de um determinado campo em listas), **avg** (para a média), **min** (menor valor) e **max** (maior valor)
- Criação das classes **OracleSequenceUpdater** e **PostgreSQLSequenceUpdater** para atualização do valor atual das sequences baseado o maior "id" das tabelas
- Melhoria na geração do *ClassMB* e das mensagens de internacionalização do xpert-maker
- Adicionada a opção de esconder o "id" da URL quando se edita um objeto no maker

A lista completa pode ser encontrada no google codes no link:

<https://code.google.com/p/xpert-framework/issues/list?can=1&q=status%3AFixed+Targetversion%3D1.8.1>

1.5. Guia de migração para a versão 1.8.1

Nenhuma alteração no código será necessário da versão 1.8 para a 1.8.1.

2. Padrão de documentação

A documentação das classes, páginas e outros itens segue o seguinte padrão:

@Stateless – *Indica o Artefato*
JavaEE – *Indica a tecnologia*
Indica um EJB que não guarda estado – *Descrição do Artefato*

Para códigos fonte o cor da linha será destacada em cinza da seguinte maneira:

```
public void validate(Pessoa pessoa) throws BusinessException {  
    //carga horaria nao pode ultrapassar 9h  
    if (pessoa.getCargaHoraria() > 9) {  
        throw new BusinessException("pessoa.business.cargaHorarioAcima");  
    }  
}
```

3. Links úteis

Showcase do xpert-framework

Exemplos de funcionamento dos componentes são disponíveis aqui.

<http://showcase.xpertsistemas.com.br/>

Homepage do xpert-framework no google codes

Atualmente o código fonte do framework e do showcase está disponível no google codes.

No google codes ainda é possível acompanhar o andamento de desenvolvimento, cadastrar bugs e sugerir melhorias.

<http://code.google.com/p/xpert-framework/>

Hibernate

A implementação padrão para JPA que o xpert-framework usa é o Hibernate.

<http://www.hibernate.org/>

Primefaces

O xpert framework trabalha fortemente com os componentes do primefaces, que é um dos mais populares frameworks de componentes JSF.

<http://www.primefaces.org/>

Primefaces Extensions

Framework que vai além do primefaces e adiciona novos componentes.

<http://primefaces-extensions.github.io/>

OmniFaces

Framework JSF com muitas classes, métodos, conversores e componentes utilitários para JSF.

<http://www.omnifaces.org>

Bootstrap

Framework HTML, CSS e Javascript, que se integra a geração de CRUD do xpert-maker a partir do xpert-framework 1.8.

<http://getbootstrap.com>

4. Configuração do xpert-framework

4.1. Download

O xpert-framework é disponível através de um único jar que pode ser baixado através do link <https://code.google.com/p/xpert-framework/wiki/Download?tm=2>

4.2. Dependência maven

Para adicionar a dependência o groupId é *com.xpert* e o artefato é *xpert-framework*:

```
<dependency>
  <groupId>com.xpert</groupId>
  <artifactId>xpert-framework</artifactId>
  <version>1.8.1</version>
</dependency>
```

O artefato pode ser encontrado no link indicado na sessão “Download”.

4.3. Dependências

Algumas dependências são necessárias para o xpert-framework, elas estão listadas abaixo:

- commons-beanutils
- commons-io
- commons-collections
- freemarker
- primefaces 3.x, 4.x ou 5.x
- javaee-api
- hibernate 4.x
- JasperReports (apenas se for utilizar a classe FacesJasper)
- primefaces-extensions (apenas se for utilizar o dataTableExporter)

Para a utilização o componente **pdfPrinter**, são necessárias as seguintes dependências:

- flying-saucer-core 9.0.7
- flying-saucer-pdf-itext5 9.0.7
- batik-transcoder 1.7 (para a geração de SVG)
- itext 2.1.7
- fop 1.0
- htmlcleaner 2.10
- jsoup 18.1

Muitos dos componentes visuais do xpert-framework utilizam o primefaces 3.x, 4.x ou 5.x, por isso é importante o projeto possuí-lo, para Classes utilitárias e outros componentes não visuais sua dependência não é necessária.

É importante ressaltar que apenas a versão 4.x do hibernate é suportada, isso ocorre devido a mudança da estrutura da versão 3 para a 4, um exemplo disso é a mudança de nome de alguns pacotes.

Abaixo segue as dependências para projetos maven, essas dependências devem ser inseridas no pom.xml:

```

<dependency>
  <groupId>org.primefaces</groupId>
  <artifactId>primefaces</artifactId>
  <version>5.1</version>
</dependency>
<dependency>
  <groupId>org.primefaces.extensions</groupId>
  <artifactId>primefaces-extensions</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>4.0.0.Final</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>4.0.0.Final</version>
</dependency>
<dependency>
  <groupId>org.hibernate.javax.persistence</groupId>
  <artifactId>hibernate-jpa-2.0-api</artifactId>
  <version>1.0.1.Final</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>4.2.0.Final</version>
</dependency>
<dependency>
  <groupId>freemarker</groupId>
  <artifactId>freemarker</artifactId>
  <version>2.3.8</version>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.2</version>
</dependency>
<dependency>
  <groupId>commons-beanutils</groupId>
  <artifactId>commons-beanutils</artifactId>
  <version>1.8.0</version>
</dependency>
<dependency>
  <groupId>commons-collections</groupId>
  <artifactId>commons-collections</artifactId>
  <version>3.2.1</version>
</dependency>

```

Dependências para do componente **pdfPrinter**:

```
<dependency>
  <groupId>net.sourceforge.htmlcleaner</groupId>
  <artifactId>htmlcleaner</artifactId>
  <version>2.10</version>
</dependency>
<dependency>
  <groupId>com.lowagie</groupId>
  <artifactId>itext</artifactId>
  <version>2.1.7</version>
</dependency>
<dependency>
  <groupId>org.xhtmlrenderer</groupId>
  <artifactId>flying-saucer-core</artifactId>
  <version>9.0.7</version>
</dependency>
<dependency>
  <groupId>org.xhtmlrenderer</groupId>
  <artifactId>flying-saucer-pdf-itext5</artifactId>
  <version>9.0.7</version>
</dependency>
<dependency>
  <groupId>org.apache.xmlgraphics</groupId>
  <artifactId>batik-transcoder</artifactId>
  <version>1.7</version>
</dependency>
<dependency>
  <groupId>org.apache.xmlgraphics</groupId>
  <artifactId>fop</artifactId>
  <version>1.0</version>
</dependency>
<dependency>
  <groupId>org.jsoup</groupId>
  <artifactId>jsoup</artifactId>
  <version>1.8.1</version>
</dependency>
```

4.4. Código fonte

O código fonte do xpert-framework está disponível no google codes, e pode ser baixado pelo SVN na seguinte URL:

<https://xpert-framework.googlecode.com/svn/xpert-framework/trunk>

Versões anteriores podem ser baixadas na pasta tags:

<https://xpert-framework.googlecode.com/svn/xpert-framework/tags>

5. Configurando o arquivo xpert-config.xml

5.1. Visão Geral

Esse arquivo é necessário para fazer a configuração de alguns módulos e componentes do framework.

O xpert-config.xml deve está localizado no diretório **WEB-INF** do projeto.

Exemplo do xpert-config utilizado no showcase:

```
<xpert-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <auditing>
    <auditing-impl>com.xpert.showcase.audit.Auditing</auditing-impl>
    <metadata-impl>com.xpert.showcase.audit.Metadata</metadata-impl>
    <auditing-listener>com.xpert.showcase.session.AuditingListenerImpl</auditing-listener>
  </auditing>
  <entity-manager-factory>com.xpert.showcase.application.EntityManagerFactoryImpl</entity-manager-factory>
  <resource-bundle>bundles.messages</resource-bundle>
</xpert-config>
```

A seção **auditing** configura o módulo de auditoria, onde se indica a classe de auditoria (**auditing-impl**) e a classe de metadados (**metadata-impl**) e o listener para a auditoria (**auditing-listener**).

A tag **entity-manager-factory** define a classe que implementa **com.xpert.EntityManagerFactory**, que é necessária para operações onde se acessa o banco de dados.

A tag **resource-bundle** define o resource bundle a ser utilizado para a internacionalização do projeto.

5.2. Configurando o EntityManagerFactory

5.2.1. Introdução

Para a utilização de alguns componentes e classes utilitários do framework que fazem acesso ao banco de dados deve ser definido uma implementação da classe **com.xpert.EntityManagerFactory**, e ela deve ser colocada no xpert-config.xml, como mostrado na seção anterior.

Essa implementação só é importante caso se utilize componentes de acesso ao banco de dados, como o **initializer**, **componentes de auditoria** ou o **componente jsf do xpert-maker**.

5.2.2. Exemplo de implementação

Para esse exemplo vai ser utilizado a maneira onde se obtém o entity manager através de configuração no **web.xml**, outras implementações podem ser utilizadas também, basta que essa classe retorne o **EntityManager**.

Para esse exemplo é importante destacar que o arquivo **persistence.xml** deve ser visível para o módulo war (que possui o web.xml).

Implementação do EntityManagerFactory

```
import com.xpert.EntityManagerFactory;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.persistence.EntityManager;

public class EntityManagerFactoryImpl implements EntityManagerFactory {

    private static final Logger logger = Logger.getLogger(EntityManagerFactoryImpl.class.getName());
    private static final String ENTITY_MANAGER_REF_NAME = "java:comp/env/persistence/entityManager";

    @Override
    public EntityManager getEntityManager() {
        InitialContext ctx;
        try {
            ctx = new InitialContext();
            EntityManager entityManager = (EntityManager) ctx.lookup(ENTITY_MANAGER_REF_NAME);
            return entityManager;
        } catch (NamingException ex) {
            logger.log(Level.SEVERE, null, ex);
        } catch (Exception ex) {
            logger.log(Level.SEVERE, null, ex);
        }
        return null;
    }
}
```

Referência do entity manager no web.xml

Essa referência é indicada seguindo o padrão do javaee. O código abaixo deve ser adicionado no **web.xml**, sendo que o **persistence-unit-name** deve indicar o Persistence Unit que está no arquivo de configuração do JPA **persistence.xml**.

```
<persistence-context-ref>
  <persistence-context-ref-name>persistence/entityManager</persistence-context-ref-name>
  <persistence-unit-name>xpertShowcasePU</persistence-unit-name>
</persistence-context-ref>
```

Configurar xpert-config

Deve ser indicado a classe de implementação do EntityManagerFactory no arquivo xpert-config através da tag **entity-manager-factory**.

6. Utilitários

6.1. DateUtils (Utilitário para operações com data)

A classe **DateUtils** possui alguns métodos utilitários para operações com data, abaixo, são listados os métodos da classe:

Método	Retorno	Descrição
dateToString	String	Retorna a data formatada no padrão passado
getAge	int	Retorna a idade (em anos) baseada na data atual e a data passada por parametro
getDay	int	Retorna o dia da data passada por parâmetro
getFirstDayInYear	Date	Retorna um Date representado o primeiro dia do ano passado por parâmetro
getLastDayInYear	java.util.Date	Retorna um Date representado o último dia do ano passado por parâmetro
getMonth	int	Retorna o mês da data passada por parâmetro (considerando janeiro = 0)
getYear	int	Retorna o ano da data passada por parâmetro
removeTime	java.util.Date	Retorna a data sem hora, minuto, segundo e milisegundos
stringToCalendar	java.util.Calendar	Retorna um Calendar da data passada por parâmetro convertendo-a de String para Calendar
stringToDate	java.util.Date	Retorna um Date da data passada por parâmetro convertendo-a de String para Date

6.2. CollectionsUtils (Utilitário para operações com coleções)

A classe **CollectionsUtils** possui alguns métodos utilitários para operações com coleções.

Os métodos de ordenação de listas dessa classe se baseiam nas bibliotecas commons-collections e commons-beanutils, assim é necessário possuí-las no classpath.

Abaixo, são listados os métodos da classe:

Método	Retorno	Descrição
orderAsc	void	<p>Ordena o sentido crescente (ASC) a lista passada por parâmetro. Considerando que o bean <i>Pessoa.java</i>, possui o atributo <i>nome</i>, para fazer a ordenação por nome seria:</p> <pre>CollectionsUtils.orderAsc(pessoas, "nome");</pre> <p>O segundo parâmetro é a ordenação do campo.</p>
orderDesc	void	<p>Ordena o sentido decrescente (DESC) a lista passada por parâmetro. Considerando que o bean <i>Pessoa.java</i>, possui o atributo <i>nome</i>, para fazer a ordenação por nome seria:</p> <pre>CollectionsUtils.orderDesc(pessoas, "nome");</pre> <p>O segundo parâmetro é a ordenação do campo.</p>
order	void	<p>Ordena o sentido ASC ou DESC a lista passada por parâmetro. Esse método é útil quando se quer utilizar ordenação por mais de um campo. Considerando que o bean <i>Pessoa.java</i>, possui o atributo <i>nome</i> e <i>id</i>, para fazer a ordenação por nome e id (desc) seria:</p> <pre>CollectionsUtils.order(pessoas, "nome ASC, id DESC");</pre> <p>O segundo parâmetro é a ordenação do campo, quando não se passar ASC ou DESC, o ASC por padrão será definido.</p>

7. Xpert-Faces

7.1. Introdução

O objetivo desse módulo é prover artefatos utilitários para facilitar o uso da tecnologia Java Server Faces 2.0 (JSF), para isso, o xpert-framework disponibiliza componentes, conversores e classes utilitárias.

Os componentes pode ser importados a partir do seguintes namespace:

- `http://xpert.com/faces`Exemplo:

```
<html xmlns:x="http://xpert.com/faces" >
</html>
```

A partir da versão 1.7.1 todos os componentes estão visíveis no namespace **`http://xpert.com/faces`**. O namespace para composite componentes também continua sendo suportado.

7.2. Conversores

7.2.1. Entity Converter

Esse é um conversor genérico para entidades JPA, seguindo sua especificação. A conversão não limita apenas a essas entidades, sendo possível converter outros objetos, onde seja criado algum identificador e seu atributo/método seja anotado com `@ConverterId` do pacote `com.xpert.faces.conversion.ConverterId`.

É importante destacar que para seu funcionamento a classe deve possuir seu devido método equals sendo feito pelo identificador da entidade.

Abaixo um exemplo de sua utilização:

```
<h:selectOneMenu value="#{bean.group}" converter="entityConverter" >
  <f:selectItems value="#{bean.groups}" var="group"
    itemLabel="#{group.description}" />
</h:selectOneMenu>
```

O funcionamento desse conversor se dá da seguinte maneira, cada objeto da lista é adicionado ao ViewMap do JSF, com isso ao submeter a página o objeto é recuperado desse Map. Isso torna desnecessário consultas extras no banco de dados ou a criação de um conversor para cada entidade.

7.2.2. Entity Converter List

Conversor que se baseia no valor de uma lista, passado por parâmetro, para utilizar esse conversor é necessário adicionar a lista através do `f:attribute`.

O conversor irá buscar o valor do objeto baseado no id da entidade (seguindo a especificação do JPA).

Exemplo de utilização:

```
<h:inputHidden value="#{personMB.person}" converter="entityConverterList" >
  <f:attribute name="list" value="#{ personMB.personList}" />
</h:inputHidden>
```

```
</h:inputHidden>
```

7.2.3. CPF Converter

Conversor que remove os caracteres especiais("-", ".") do campo e coloca a máscara do CPF.

Exibir CPF com a máscara:

```
<h:outputText value="#{bean.entity.cpf}" converter="cpfConverter" >
```

Exibir CPF com a máscara e submeter sem a máscara (apenas números):

```
<h:inputText value="#{bean.entity.cpf}" converter="cpfConverter" >
```

7.2.4. CNPJ Converter

Conversor que remove os caracteres especiais("-", ".", "/") do campo e coloca a máscara do CPF.

Exibir CNPJ com a máscara:

```
<h:outputText value="#{bean.entity.cnpj}" converter="cnpjConverter" >
```

Exibir CNPJ com a máscara e submeter sem a máscara (apenas números):

```
<h:inputText value="#{bean.entity.cnpj}" converter="cnpjConverter" >
```

7.2.5. YesNoConverter

Converte valores booleanos nas Strings "sim"(true) ou "não"(false), sendo internacionalizado:

```
<h:outputText value="#{bean.entity.booleanValue}" converter="yesNoConverter" >
```

7.2.6. ActiveInactiveConverter

Converte valores booleanos nas Strings "Ativo"(true) ou "Inativo"(false), sendo internacionalizado:

```
<h:outputText value="#{bean.entity.booleanValue}" converter="activeInactiveConverter" >
```

7.3. Componentes JSF

7.3.1. BooleanSelectItens

Esse managed bean é um utilitário para se recuperar os valores booleanos quando necessários em uma página.

BooleanSelectItens

Acessível a través da EL **`#{booleanSelectItens}`** Exibe apenas as opções true ou false:

```
<h:selectOneRadio>
  <f:selectItems value="#{booleanSelectItens}"/>
</h:selectOneRadio>
```

BooleanSelectItensEmptyOption

Acessível a través da EL **`#{booleanSelectItensEmptyOption}`** Exibe as opções true, false e uma opção vazia, ela é muito útil para se carregar as opções de um filtro da coluna do primefaces:

```
<p:column headerText="Boolean Value" sortBy="#{object.booleanValue}"
  filterBy="#{object.booleanValue}"
  filterOptions="#{booleanSelectItensEmptyOption}">
  <h:outputText value="#{object.booleanValue}" converter="yesNoConverter" />
</p:column>
```

7.3.2. Confirmation

Componente para exibir uma confirmação da ação feita a partir de um `commandButton` ou `commandLink`. Sua idéia é semelhante a de um “confirm” do javascript, porém integrado com os componentes `command ajax` e não `ajax`.

Atributos

Nome	Valor Padrão	Tipo	Descrição
message	Confirma?	String	Define a mensagem a ser exibida de confirmação (o valor padrão é internacionalizado)
confirmLabel	Sim	String	Define a valor do botão de confirmação.
cancelLabel	Não	String	Define o valor do botão da não confirmação

Utilização

Usando `commandButton` do JSF com `ajax`:

```
<h:commandButton value="Submit" >
  <f:ajax render="@form" execute="@form"/>
  <x:confirmation/>
</h:commandButton>
```

Usando `commandButton` do Primefaces e definindo uma mensagem customizada:

```
<p:commandButton process="@form" update="@form" value="Submit (With custom message)" >
  <x:confirmation message="Are you sure?" confirmLabel="Of course" cancelLabel="No way"/>
</p:commandButton>
```

Ao se utilizar o `commandLink` do primefaces, devido ao seu comportamento não é possível utilizar o `confirmation`. Pelos testes realizados, quando se adiciona algum componente a ele, além do `confirmation`, então o `confirmation` é renderizado.

7.3.3. CPF e CNPJ

Esses dois componentes servem para exibir o cpf (cadastro de pessoa física) e cnpj (cadastro nacional de pessoa jurídica) a vantagem é que eles já usam os conversores cpfConverter e cnpjConverter, os validadores cpfValidator e cnpjValidator e ainda inserem a máscara no input.

Atributos

Possui basicamente os mesmos atributos de um inputMask (primefaces).

Utilização

CPF:

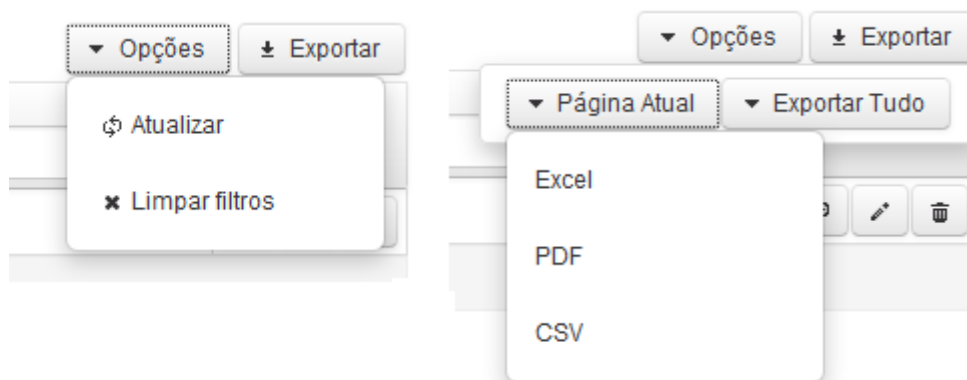
```
<x:cpf value="#{pessoaMB.pessoaFisica.cpf}" />
```

CNPJ:

```
<x:cnpj value="#{pessoaMB.pessoaJuridica.cpf}" />
```

7.3.4. DataTableActions

O DataTableActions é um componente para ações genéricas do dataTable.



Esse componente adiciona as seguintes funcionalidades para uma tabela:

- Atualizar
- Limpar Filtros
- Exportação (Excel, PDF e CSV)

A exportação é feita através do componente **x:dataTableExporter**, que por sua vez utiliza o primefaces e o primefaces-extensions, como já foi descrito anteriormente.

Atributos

Nome	Valor Padrão	Tipo	Descrição
target		String	Id do dataTable que será exportada.
fileName		String	Nome do arquivo que será gerado na exportação
export	true	boolean	Indica se deve ser renderizado a opção de Exportar
options	true	boolean	Indica se deve ser renderizado a parte de "opções"
widgetVar		String	Indica o widgetVar da tabela que as ações serão aplicadas

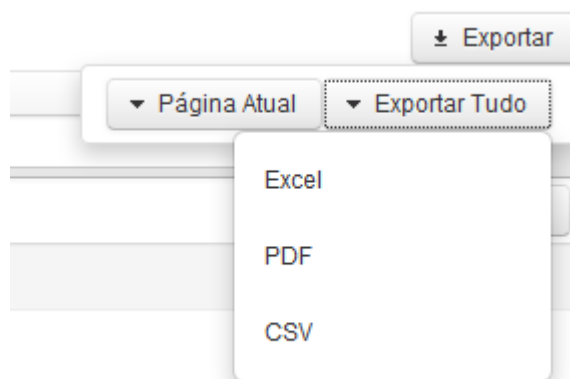
Exemplo de Utilização

```
<x:dataTableActions target=":formListPessoa:dataTablePerson" fileName="export_person"
  widgetVar="widgetTablePerson" />

<p:dataTable id="dataTablePerson" widgetVar="widgetTablePerson" >
  ....
</p:dataTable>
```


7.3.5. DataTableExporter

O DataTableExporter é um component para exportação de dados do dataTable.



Ele utiliza o componente *p:dataExporter* (primefaces) e o componente *pe:exporter* (primefaces-extensions).

Por conta de usar o primefaces-extensions, este se torna uma dependência obrigatória, caso queira utilizar esse componente.

Atributos

Nome	Valor Padrão	Tipo	Descrição
target		String	Id do dataTable que será exportada.
fileName		String	Nome do arquivo que será gerado na exportação
fontSize	7	Integer	Tamanho da fonte dos arquivos gerados
landscape	true	boolean	Indica se deve ser utilizado a orientação paisagem, false indica se é em retrato

Exemplo de Utilização

```
<x:dataTableExporter target=":formListPessoa:dataTablePerson" fileName="export_person" />
<p:dataTable id="dataTablePerson" >
  ....
</p:dataTable>
```

Facet "header" para definir o cabeçalho da coluna

É importante mencionar que os componentes exporter consideram o cabeçalho da tabela como sendo o *f:facet "header"*, sendo assim, mesmo definindo o atributo *headerText* no *column*, na exportação ao cabeçalho pode não aparecer.

Exemplo de uso do facet:

```

<p:column headerText="Name">
    <f:facet name="header" >
        <h:outputText value="Name" />
    </f:facet>
    <h:outputText value="#{person.name}"/>
</p:column>

```

Exportação de atributos LAZY em dataTable paginado

Quando se utiliza o **x:initializer** na tabela, ele irá carregar o objeto na página, porém quando exporta uma tabela (paginada) o initializer não agiu ainda nas páginas não carregadas, com isso um *LazyInitializationException* pode ser lançado.

Para evitar esse problema o ideal é que esses dados já venham carregados na listagem, via *join fetch* (JPQL/HQL) por exemplo. No caso de um *AbstractBaseBean* isso pode ser resolvido sobrescrevendo o método *getDataModelJoinBuilder*, como mostrado abaixo.

Na tabela abaixo, o atributo *person.profile*, possui o initializer:

```

<p:dataTable id="dataTablePerson" var="person" value="#{personMB.dataModel}" >
    <p:column headerText="Name">
        <f:facet name="header" >
            <h:outputText value="Name" />
        </f:facet>
        <h:outputText value="#{person.name}"/>
    </p:column>
    <p:column>
        <f:facet name="header" >
            <h:outputText value="Profile" />
        </f:facet>
        <h:outputText value="#{person.profile.name}">
            <x:initializer/>
        </h:outputText>
    </p:column>
</p:dataTable>

```

No **PersonMB** a sobrescrita do *getDataModelJoinBuilder*:

```

public class PersonMB extends AbstractBaseBean<Person> implements Serializable {

    (...)

    @Override
    public String getDataModelOrder() {
        return "p.nome";
    }

    @Override
    public JoinBuilder getDataModelJoinBuilder() {
        return new JoinBuilder("p")
            .leftJoinFetch("p.profile", "pf");
    }
}

```

Exportação de colunas que possuem um x:dateFilter

Quando o dataTable possui o componente *x:dataFilter*, ao exportar o primfaces/primefaces-extensions vai reconhecer um componente UPanel na tabela, esse componente será escrito, ou ainda será desprezado e escrito vazio. Com isso o cabeçalho da coluna pode apresentar um valor não esperado.

Para solucionar esse problema um pequeno “hack” que pode ser feito, é colocar a coluna que possui o dataFilter, como *exportable="false"* e criar uma nova coluna com *style="display: none"*, assim essa coluna não será exibida na tela, mas sairá na exportação, como no exemplo mostrado abaixo:

```
<p:column exportable="false">
  <f:facet name="header" >
    <h:outputText value="Birthday" />
    <x:dateFilter/>
  </f:facet>
  <h:outputText value="#{person.birthday}"/>
</p:column>
<!-- coluna falsa-->
<p:column style="display: none;">
  <f:facet name="header" >
    <h:outputText value="Birthday" />
  </f:facet>
  <h:outputText value="#{person.birthday}"/>
</p:column>
```

Logomarca padrão na geração da exportação

Para adicionar uma logomarca no arquivo PDF gerado na exportação, basta ter um arquivo chamado logo (png ou jpg), dentro da pasta **images**, dessa forma:

{páginas web}/images/logo.png

7.3.6. DateFilter

Filtro de datas para o LazyDataModelImpl.



Este component facilita o uso de filtros com campos de data no dataTable que utilize o LazyDataModelImpl.

Esse componente renderiza 2 componentes calendar do primefaces, um sendo a data inicial da consulta e outro sendo a data final. Ele funciona por causa do RestrictionType **"DATA_TABLE_FILTER"** e tem um tratamento diferenciado, ao selecionar a data de início da consulta é adicionado a restrição **"GREATER_EQUALS_THAN"** e ao se selecionar a data final é adicionado a restrição **"LESS_EQUALS_THAN"**.

Para ser renderizado ele deve ser definido no facet "header" do componente "column".

Utilização

```
<p:column sortBy="#{object.date}"
  filterBy="#{object.date}" style="text-align: center;">
  <f:facet name="header">
    Date Field
    <x:dateFilter/>
  </f:facet>
  <h:outputText value="#{object.date}"/>
</p:column>
```

7.3.7. Download

Componente para bloquear a tela e adicionar eventos quando a requisição for o download de algum arquivo. Ele pode ser utilizado com `commandButton`, `commandLink` do JSF ou Primefaces. Sendo para o primefaces assim como o componente *confirmation* o `commandLink` apresenta um comportamento diferente.

Para gerar o download deve ser chamado o método *FacesUtils.download()*.

Atributos

Nome	Valor Padrão	Tipo	Descrição
showModal	true	Boolean	Define a tela deve ser bloqueada enquanto a requisição é completa
message	Carregando	String	Define a mensagem a ser exibida enquanto a tela é bloqueada.
onstart		String	Javascript a ser executado ao iniciar a requisição
oncomplete		String	Javascript a ser executado ao finalizar a requisição

Utilização

Usando *commandButton* do JSF:

```
<h:commandButton action="#{downloadMB.download}" value="Download">
  <x:download/>
</h:commandButton>
```

Classe *DownloadMB.java*:

```
public class DownloadMB {

    public void download() throws IOException, InterruptedException {
        String fileContent = "This is xpert-framework";
        //wait 5s
        Thread.sleep(5000);
        FacesUtils.download(fileContent.getBytes(), "text/plain", "file.txt");
    }
}
```

7.3.8. Filter On Enter

Componente para limitar a consulta do componente *dataTable* do primefaces quando se utiliza a opção *filterBy* da coluna a ser realizada apenas quando se aperta o *Enter*.

Atributos

Nome	Valor Padrão	Tipo	Descrição
target		String	Id da tabela a ser utilizado o filterOnEnter.
selector		String	Seletor jQuery para retornar as tabelas a serem utilizadas.

Utilização

Quando os atributos target e selector não são definidos, por padrão todas as tabelas serão aplicadas a regra.

```
<p:dataTable id="dataTablePerson" var="person" value="#{personMB.dataModel}">
  <p:column headerText="#{msg['person.name']}" sortBy="#{person.name}"
    filterBy="#{person.name}">
    <h:outputText value="#{person.name}"/>
  </p:column>
</p:dataTable>
<x:filterOnEnter/>
```

7.3.9. Find All Bean

7.3.9.1. Introdução

Para consultas genéricas o xpert-framework disponibiliza a classe FindAllBean, através dela é possível fazer consultas das entidades mapeadas.

Ele é muito útil por exemplo na geração de código do módulo xpert-maker, onde para os atributos mapeados com `@ManyToOne` ou `@ManyToMany` são criados componentes `h:selectOne`, e eles já são preenchidos do banco de dados.

7.3.9.2. Exemplo de implementação

O exemplo a seguir mostra um ManagedBean do escopo de visão, e nele é mapeado a classe Group. Ao criar uma classe filha dela é necessário sobrescrever o método `getClassModel`.

Esse `classModel` pode ser usado para indicar a ordenação da entidade recuperada, nesse caso **group** será ordenado pelo atributo **description**.

```
@ManagedBean
@ViewScoped
public class FindAllBean extends com.xpert.faces.bean.FindAllBean {

    private static final Map<Class, ClassModel> MODEL = new HashMap<Class, ClassModel>();

    static {
        MODEL.put(Group.class, new ClassModel("description"));
    }

    @Override
    public Map<Class, ClassModel> getClassModel() {
        return MODEL;
    }
}
```

No `classModel` é possível ainda indicar o label a ser utilizado na exibição quando o retorno é uma lista do objeto `SelectItem`. O retorno `SelectItem` é utilizado por exemplo do atributo *filterOptions* da coluna do **dataTable** do **primefaces**.

7.3.9.3. Exemplo de utilização

Pode ser utilizado para preencher combobox na tag `selectItems`:

```
<h:selectOneMenu value="#{bean.country}" converter="entityConverter" >
    <f:selectItems value="#{findAllBean.getClassMB.country}" var="country"
        itemLabel="#{country.name}" />
</h:selectOneMenu>
```

Pode ser utilizado no *filterOptions* do `dataTable` do **primefaces**, sendo que para esse caso deve retornar uma lista de `select item` e deve-se utilizar o método **getSelect**:

```
<p:column headerText="#{msg['person.status']}" sortBy="#{person.status}"
    filterBy="#{person.status}"
    filterOptions="#{findAllBean.getSelect(classMB.status)}">
    <h:outputText value="#{person.status}"/>
</p:column>
```

Muito da geração de código fornecida pelo xpert-maker faz referência a esse `findAllBean`, e ele deve ser considerado apenas como um utilitário genérico, lógicas mais complexas de recuperação de dados não devem ser feitas nele.

7.3.10. Group

O componente “**group**” foi desenvolvido para que as listas possam ser agrupadas na view sem muita codificação. O objetivo desse componente é agrupar uma determinada lista por algum campo, de maneira que esses valores possam ser acessíveis via *expression language*.

Esse componente agrupa listas dos tipos que implementam a interface *java.util.List*. Para agrupar uma lista deve-se indicar o campo que indica o agrupamento através do atributo “**groupBy**”.

Atributos

Nome	Valor Padrão	Tipo	Descrição
groupBy		String	O nome do campo pelo qual a lista será agrupada
value		Expression Language	Indica o valor do componente, o valor deve ser uma Expression Language que retorne alguma lista (<i>java.util.List</i>)
var		String	Nome da variável (request scope) da iteração atual do componente. O objeto gerado pelo var possui 2 atributos, key e value. O atributo key é o valor do objeto pelo qual a lista foi agrupada e value é uma lista resultante desse agrupamento.
itemSortField		String	Nome do campo de ordenação dos itens de cada iteração do agrupamento
itemSortOrder	ascending	String	Indica a ordenação dos items, deve ser "ascending" ou "descending" o valor padrão é “ascending”
sortFunction		MethodExpr	Função personalizada para ordenação
sortField		String	Nome do campo pelo qual a lista será ordenada
sortOrder	ascending	String	Indica a ordenação do agrupamento, deve ser "ascending" ou "descending" o valor padrão é “ascending”
itemSortFunction		MethodExpr	Função personalizada para ordenação dos itens

Atributo “var”

Ao indicar o atributo “**var**”, a variável criada, possui 2 valores:

- **key** - representa o valor do campo indicado no *groupBy*
- **value** - representa o valor agrupado

Exemplo de Utilização

Para os exemplos demonstrados vamos considerar as classes abaixo que irão representar pessoa, cidade, estado e país, essas classes irão representar o modelo a ser agrupado.

Person.java

```
public class Person {  
  
    private String name;  
    private City city;  
  
    //getters and setters  
}
```

City.java

```
public class City {  
  
    private String name;  
    private State state;  
  
    //getters and setters  
}
```

State.java

```
public class State {  
  
    private String name;  
    private Country country;  
  
    //getters and setters  
}
```

Country.java

```
public class Country {  
  
    private String name;  
  
    //getters and setters  
}
```

Exemplo 1 - Agrupando uma lista de pessoas por cidade

Considerando as classe **Person.java**, (que possui *name* e *city*) e **City.java** (que possui *name*), o exemplo abaixo mostra o uma lista de pessoa agrupada por Cidade:

```

<x:group value="#{groupComponentMB.people}" var="personGroup"
  groupBy="city" rowIndexVar="index" sortField="name"
  itemSortField="name" itemSortField="descending" >

  <h:outputText style="font-weight: bold" value="#{index+1} - #{personGroup.key.name}"/>
  <p:separator/>
  <p:dataTable value="#{personGroup.value}" var="person" rowIndexVar="indexPerson" >
    <p:column>
      <h:outputText value="#{index+1}.#{indexPerson+1}"/>
    </p:column>
    <p:column headerText="Name" >
      <h:outputText value="#{person.name}"/>
    </p:column>
  </p:dataTable>
  <br/>
</x:group>

```

Nesse exemplo ao acessar “**personGroup.key**” vamos obter uma instância de cidade, ou seja a cidade da iteração atual, e ao acessar “**personGroup.value**” obtemos uma lista de pessoas que pertencem a essa cidade.

O resultado o agrupamento acima, seria algo parecido com o mostrado na imagem abaixo, para cada cidade serão listadas as suas pessoas em um *p:dataTable*, perceba ainda que o atributo *itemSortOrder* foi definido como *descending*, assim os itens (as pessoas) serão ordenadas de forma decrescente:

1 - CAMPINAS		
	Name	Email
1.1	PERSON CAMPINAS 2	person.campinas2@gmail.com
1.2	PERSON CAMPINAS 1	person.campinas1@gmail.com
2 - LA PLATA		
	Name	Email
2.1	PERSON LA PLATA 3	person.la.plata3@gmail.com
2.2	PERSON LA PLATA 2	person.la.plata2@gmail.com
2.3	PERSON LA PLATA 1	person.la.plata1@gmail.com
3 - MANAUS		
	Name	Email
3.1	PERSON MANAUS 1	person.manaus1@gmail.com

Exemplo 2 - Agrupando uma lista de pessoas por país e estado

Nesse exemplo, serão usados 2 vezes o componente *group* o primeiro irá agrupar por país e o segundo irá pegar o resultado do primeiro e agrupar por estado:

```

<x:group value="#{groupComponentMB.people}" var="personGroupCountry"
        groupBy="city.state.country" rowIndexVar="indexCountry"
        itemSortField="name" >
  <h:outputText style="font-weight: bold;font-size: 18px;font-size: 15px;"
    value="#{indexCountry+1} - #{personGroupCountry.key}"/>
  <p:separator/>

  <x:group value="#{personGroupCountry.value}" var="personGroupState"
    groupBy="city.state" rowIndexVar="indexState"
    itemSortField="name">
    <h:outputText style="font-style: italic;"
      value="#{indexCountry+1}.#{indexState+1} - #{personGroupState.key}"/>
    <p:separator/>
    <p:dataTable value="#{personGroupState.value}" var="person"
      rowIndexVar="indexPerson" >
      <p:column>
        <h:outputText value="#{indexCountry+1}.#{indexState+1}.#{indexPerson+1}"/>
      </p:column>
      <p:column headerText="Name" >
        <h:outputText value="#{person.name}"/>
      </p:column>
      <p:column headerText="City" >
        <h:outputText value="#{person.city.name}"/>
      </p:column>
    </p:dataTable>
    <br/>
  </x:group>
<br/>
</x:group>

```

Nesse exemplo ao acessar “**personGroupCountry.key**” vamos obter uma instância de país, ou seja o país da iteração atual, e ao acessar “**personGroupCountry.value**” obtemos uma lista de pessoas que pertencem a esse país.

No segundo agrupamento temos “**personGroupState.key**” que será uma instância de estado e “**personGroupState.value**” que será as pessoas daquele estado, sendo assim tem 2 níveis de agrupamento, primeiro por país, depois por estado.

O resultado do agrupamento acima seria algo parecido com o mostrado abaixo:

1 - BRASIL

1.1 - SAO PAULO

	Name	Email	City
1.1.1	PERSON CAMPINAS 1	person.campinas1@gmail.com	CAMPINAS
1.1.2	PERSON CAMPINAS 2	person.campinas2@gmail.com	CAMPINAS
1.1.3	PERSON SAO PAULO 1	person.sao.paulo1@gmail.com	SAO PAULO

1.2 - AMAZONAS

	Name	Email	City
1.2.1	PERSON MANAUS 1	person.manaus1@gmail.com	MANAUS

2 - ARGENTINA

2.1 - BUENOS AIRES

	Name	Email	City
2.1.1	PERSON LA PLATA 1	person.la.plata1@gmail.com	LA PLATA
2.1.2	PERSON LA PLATA 2	person.la.plata2@gmail.com	LA PLATA

SortFunction

É possível definir uma função customizável para a ordenação dos itens.

```
public int sortByCustom(Object o1, Object o2) {  
    //return -1, 0 , 1 if o1 is less than, equal to or greater than co2  
}
```

Na página, é necessário via EL o método de ordenação no atributo *sortFunction* :

```
<x:group value="#{groupComponentMB.people}" var="personGroupCountry"  
    groupBy="city.state.country" rowIndexVar="indexCountry"  
    sortFunction="#{countryMB.sortByCustom}"  
    itemSortField="name" >  
    (  
    ..  
    </x:group>
```

7.3.11. Initializer

Componente para inicializar os objetos Lazy do Hibernate/JPA na view. Este componente evita o famoso *LazyInitializationException* ao fazer referência a um objeto *lazy* na view.

Atributos

Nome	Valor Padrão	Tipo	Descrição
value		Expression Language	Caso seja informado, inicializa a expressão informada.
entityManager		javax.persistence.EntityManager	Indica o entity manager a ser utilizado. Ao fazer isso a configuração EntityManagerFactory não é necessária.
property	value	String	Indica o atributo do componente pai que será inicializado. Por padrão é inicializado o atributo "value".

Utilização

```
<h:outputText value="#{bean.entity.lazyAttribute.description}" >
  <x:initializer/>
</h:outputText>
```

Para recuperar o entity manager e inicializar o objeto o xpert-framework precisa que seja definido o EntityManagerFactory e que seja devidamente configurado, isso pode ser visto na sessão [Configurando o EntityManagerFactory](#). Essa configuração pode ser ignorada uma vez que definido o EntityManager do componente como mostrado abaixo.

Informando o Entity Manager no initializer

Caso não possua a configuração mencionada, é possível ainda usar o componente informando o Entity Manager. Isso não exige configuração, exige apenas que o esse EntityManager seja informado:

```
<h:outputText value="#{bean.entity.lazyAttribute.description}" >
  <x:initializer entityManager="#{bean.entityManager}"/>
</h:outputText>
```

Inicializar qualquer propriedade do componente pai

Por padrão o initializer busca inicializar o atributo "value" do componente pai, mas é possível indicar qual atributo do componente pai ele irá processar bastando indicar o atributo "property". No exemplo abaixo o atributo "header" do panel será inicializado:

```
<p:panel header="#{bean.entity.lazyAttribute.description}" >
  <x:initializer property="header"/>
</p:panel>
```

Inicializar qualquer expressão

É possível ainda inicializar qualquer objeto informando a expressão do objeto através do atributo “value”. Isso torna mais dinâmica sua utilização, pois independe do componente pai. A baixo a expressão `{bean.entity.lazyAttribute.description}` está sendo informada para ser inicializada:

```
<x:initializer value="{bean.entity.lazyAttribute.description}" />
<h:outputText value="{bean.entity.lazyAttribute.description}" />
```

O exemplo acima é equivalente a:

```
<h:outputText value="{bean.entity.lazyAttribute.description}" >
  <x:initializer/>
</h:outputText>
```

Inicializar múltiplas expressões

Outra opção disponível é a inicialização de múltiplas expressões:

```
<h:outputText value="{bean.entity.lazyAttribute} - {bean.entity.lazyAttribute2}" >
  <x:initializer/>
</h:outputText>
```

7.3.12. Input Number

Componente para inserir mascara para valores numéricos. Útil por exemplo para valores monetários.

Atributos

Possui basicamente os mesmos atributos de um `inputText`, adicionando os seguinte componentes:

Nome	Valor Padrão	Tipo	Descrição
<code>allowNegative</code>	<code>false</code>	Boolean	Indica se podem ser informados valores negativos (deve-se pressionar a tecla ‘-’)
<code>limit</code>	<code>15</code>	Integer	Quantidade limite de casas numéricas (decimais e interias)
<code>centsSeparator</code>	Conforme o Locale	String	Separador de centésimos
<code>thousandsSeparator</code>	Conforme o Locale	String	Separador de milhares
<code>centsLimit</code>	<code>2</code>	Integer	Quantidades limite de números não inteiros

Utilização

Utilização Básica:

```
<x:inputNumber value="#{bean.value}" />
```

Aceitando números negativos:

```
<x:inputNumber value="#{bean.value}" allowNegative="true"/>
```


7.3.13. Legends

Composite Component para exibir uma lista de comandos possíveis. Útil por exemplo para colocar um legenda no data table, especificando os botões “detalhar”, “editar” e “exclusão”.

Atributos

Possui os seguintes atributos:

Nome	Valor Padrão	Tipo	Descrição
vertical	false	Boolean	Indica se os itens devem ser dispostos na vertical, por padrão são 2 colunas.
edit	false	Boolean	Exibir item “Editar”
detail	false	Boolean	Exibir item “Detalhar”
delete	false	Boolean	Exibir item “Exclusão”
download	false	Boolean	Exibir item “Download”
report	false	Boolean	Exibir item “Download”
select	false	Boolean	Exibir item “Selecionar”
print	false	Boolean	Exibir item “Imprimir”
replace	false	Boolean	Exibir item “Substituir”

Utilização

Utilização básica:

```
<x:legends detail="true" edit="true" delete="true"/>
```

Legendas na posição vertical:

```
<x:legends detail="true" edit="true" delete="true" vertical="true"/>
```

Como cabeçalho de uma coluna do dataTable:

```
<p:column>
  <f:facet name="header">
    <x:legends detail="true" edit="true" delete="true"/>
  </f:facet>
</p:column>
```

7.3.14. Mask

O componente mask, serve para inserir máscara no componente p:calendar, esse componente só passou a usar máscara a partir da versão 5.x do primefaces. Basicamente ele pega o formato de data atribuído para o componente e a partir dele é gerado uma máscara.

Por exemplo o padrão *dd/MM/yyyy* gera a máscara *99/99/9999*, o padrão *HH:mm* gera a máscara *99:99*.

Atributos

Nome	Valor Padrão	Tipo	Descrição
placeholder	_ (underline)	String	Assim como no p:inputMask o placeholder serve para colocar um caractere que indica a máscara ao se colocar foco no campo.

Utilização

Utilizando o pattern *dd/MM/yyyy* (dia - mês - ano):

```
<p:calendar value="#{pessoaMB.entity.date}" showOn="button" pattern="dd/MM/yyyy" >
    <x:mask/>
</p:calendar>
```

Utilizando o pattern *HH:mm* (hora - minuto):

```
<p:calendar value="#{pessoaMB.entity.date}" showOn="button" pattern="HH:mm" >
    <x:mask/>
</p:calendar>
```

7.3.15. Modal Messages

Componente para exibição de mensagens do JSF dentro de um dialog. É um composite componente que utiliza os componentes *dialog* e *messages* do primefaces.

Atributos

Nome	Valor Padrão	Tipo	Descrição
autoUpdate	false	Boolean	Indica se o componente deve se atualizar na requisição ajax
escape	true	Boolean	Define se o conteúdo da mensagem deve ser escapado

Utilização

Utilização básica:

```
<x:modalMessages/>
```

Utilização com *autoUpdate*:

```
<x:modalMessages autoUpdate="true" />
```

7.3.16. PdfPrinter

PDFPrinter é um componente que exporta o HTML da página para PDF, ele foi feito usando frameworks já consolidados como o **flying-saucer** e **itext**.

Abaixo segue a lista de dependências necessárias:

Flying Saucer	https://code.google.com/p/flying-saucer/
iText	http://itextpdf.com/
Jsoup	http://jsoup.org/
HTML Cleaner	http://htmlcleaner.sourceforge.net/
Apache Batik (exportar SVG)	http://xmlgraphics.apache.org/batik/

Para projetos *maven* dependências necessárias para o componente são mostradas na [seção de dependências](#).

Atributos

Nome	Valor Padrão	Tipo	Descrição
target		String	Id do componente a ser exportado para PDF
fileName		String	Nome do arquivo PDF que será gerado para download
orientation	portrait	String	Orientação do PDF, deve-se indicar "portrait" para retrato ou "landscape" para paisagem

Utilização

Para utilizar o component deve-se indicar o "target", que será o id do componente a ser exportado:

```
<p:commandButton title="Export PDF" immediate="true" ajax="false" value="Export PDF">
    <x:pdfPrinter target="exportTargetId" fileName="File Name" />
</p:commandButton>
```

O componente download pode ser adicionado para gerar uma melhor experiência:

```
<p:commandButton title="Export PDF" immediate="true" ajax="false" value="Export PDF">
    <x:download />
    <x:pdfPrinter target="exportTargetId" fileName="File Name" />
</p:commandButton>
```

Exemplo de exportação de um *h:panelGroup*:

```
<p:commandButton title="Export PDF" immediate="true" ajax="false"
                 value="Export PDF">
    <x:download/>
    <x:pdfPrinter target="panelExport" fileName="Lorem Ipsum" />
</p:commandButton>

<h:panelGroup id="panelExport" layout="block" style="border: 1px solid black; padding: 10px;">
    <h1>Title</h1>
    <h2>Subtitle</h2>
    <p>
        Lorem ipsum dolor sit amet, <span style="color: red;">consectetur adipiscing elit</span>.
        Cras at accumsan mauris. Nulla a accumsan nisl. Nam dapibus a velit quis consequat.
        Mauris a volutpat sem. Suspendisse semper enim in enim gravida, quis maximus ante lobortis.
    </p>
</h:panelGroup>
```

Exportação de gráficos

O **pdfPrinter** é compatível na exportação de gráficos baseados em SVG, como é o caso do google charts (que também está presente no primefaces-extensions).

Gráficos do próprio primefaces não são compatíveis pois são baseados em *canvas*.

Customização de estilos

A lib *flying-saucer* é compatível com a especificação CSS, sendo assim para adicionar os estilos basta adicionar nos estilos CSS.

Lembrando que é possível ainda adicionar o um CSS inline, e indicar o media=print, para que o estilo seja adicionado apenas na impressão, ou media=screen para que seja apenas na tela como mostrado abaixo:

```
<style media="screen" style="text/css">
    .no-screen{display: none;}
</style>
<style media="print" style="text/css">
    /* mudar cor do cabeçalho das tabelas (apenas na impressão) */
    .ui-state-default {background-color: #e6e6e6 !important;}
    .no-print{display: none;}
</style>

<span class="no-screen">Este conteúdo não será exibido na tela, apenas na impressão</span>
<span class="no-print">Este conteúdo não será impresso</span>
```

7.3.17. RestorableFilter

O componente *restorableFilter* indica que os filtros do *dataTable* que utilizam o *LazyDataModelImpl* serão mantidos na sessão. Assim caso seja realizado a filtragem da tabela (que deve ser um *LazyDataModelImpl*) esses filtros serão adicionados a sessão Http, e ao entrar na tela com o *dataTable*, esse já estará filtrado conforme os filtros informados previamente.

Esse componente não guarda os dados que são apresentados na tabela, ele guarda apenas os filtros, assim quando o *dataTable* for chamado novamente, a consulta será novamente realizada.

Atributos

Nome	Valor Padrão	Tipo	Descrição
target		String	Id do dataTable que terá os filtros salvos.

Utilização

O *restorableFilter* deve ser usado antes do *dataTable*, como mostrado abaixo:

```
<x:restorableFilter target="dataTablePessoa"/>
<p:dataTable paginator="true" rows="10"
    var="pessoa" id="dataTablePessoa"
    value="#{pessoaMB.dataModel}" lazy="true" >
</p:dataTable>
```

Lembrando que no exemplo acima *#{pessoaMB.dataModel}* deve ser um *LazyDataModelImpl*.

7.3.18. Spread Checkbox/Radio

Componente para quebrar em colunas os components `h:selectManyCheckBox` e `h:selectManyRadio`.

Esses componentes são limitados a dizer apenas se são na horizontal e na vertical, com o componente spread é possível determinar a quantidade de colunas deles. Este componente adiciona ainda um estilo CSS para o *background* do *checkbox* marcado mudar a cor.

Atributos

Nome	Valor Padrão	Tipo	Descrição
columns		Integer	Quantidade de colunas
highlight	true	Boolean	Indica se o estilo deve ser aplicado ou não nos itens selecionados

Utilização

Utilização básica:

```
<h:selectManyCheckbox >  
  <x:spread columns="3"/>  
  <f:selectItems value="#{bean.items}" />  
</h:selectManyCheckbox>
```

7.3.19. Sum, Avg, Min e Max

As funções “**sum**”, “**avg**”, “**min**” e “**max**”, são funções genéricas que podem ser usadas com o *LazyDataModelImpl* ou com listas da interface *java.util.Collection*.

- O “**sum**” retorna o somatório
- O “**avg**” retorna a média
- O “**min**” retorna o menor valor
- O “**max**” retorna o maior valor

Estas funções recebem 2 parâmetros:

1. A Lista
2. O nome do campo a ser somado

Exemplo de Utilização

Considerando a classe **Person.java**, que possui o atributo “salary”, caso se tenha uma lista de pessoa, para o somar todos os salários, pode-se usar:

```
<h:outputText value="#{x:sum(personMB.list, 'salary')}">
```

Por padrão o sum, quando aplicado em coleções, retornará um *java.math.BigDecimal*, sendo assim podemos utilizar o *convertNumber* para formatá-lo na tela, como mostrado abaixo:

```
<h:outputText value="#{x:sum(pessoaMB.list, 'salary')}">  
  <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>  
</h:outputText>
```

Retornando a **média** dos salários:

```
<h:outputText value="#{x:avg(pessoaMB.list, 'salary')}">  
  <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>  
</h:outputText>
```

Retornando o **maior** dos salários:

```
<h:outputText value="#{x:max(pessoaMB.list, 'salary')}">  
  <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>  
</h:outputText>
```

Retornando o **menor** dos salários:

```
<h:outputText value="#{x:min(pessoaMB.list, 'salary')}">  
  <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>  
</h:outputText>
```

É possível ainda forçar que o retorno seja Integer ou Long, mas para isso o campo indicado deve corresponder ao tipo de retorno:

Retornando **Integer**:

```
<h:outputText value="#{x:sumInteger(pessoaMB.list, 'algumCampoInteger')}}">
```

Retornando **Long**:

```
<h:outputText value="#{x:sumLong(pessoaMB.list, 'algumCampoLong')}}">
```

Somatório em um LazyDataModelImpl

Uma das maiores vantagens do xpert-framework é a implementação do *LazyDataModel* do primefaces, utilizando JPA. Esse modelo faz as consultas de maneira paginada no banco, com isso não temos a lista completa, para que possamos utilizar o **sum**. Por conta disso, ao realizar um sum de um *LazyDataModel* será gerada uma query para trazer aquele somatório diretamente do banco, o HQL gerado é algo parecido com:

```
SELECT SUM(salary) FROM Person.class WHERE (...)
```

O mesmo vale para as funções avg, min e max:

```
SELECT AVG(salary) FROM Person.class WHERE (...)
```

```
SELECT MIN(salary) FROM Person.class WHERE (...)
```

```
SELECT MAX(salary) FROM Person.class WHERE (...)
```

Essa consulta irá considerar os filtros atuais da tabela, ou seja, poderão ser colocados restrições na cláusula “where”.

Para utilizar o **sum** com o *LazyDataModel*, é semelhante ao usado em coleções:

```
<h:outputText value="#{x:sum(personMB.dataModel, 'salary')}}">
  <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>
</h:outputText>
```

Para pegar apenas o somatório da página atual, basta apontar para o “**wrappedData**” (método que retorna os valores da página atual) do *dataModel*:

```
<h:outputText value="#{x:sum(personMB.dataModel.wrappedData, 'salary')}}">
  <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>
</h:outputText>
```

No exemplo abaixo é exibido o somatório, tanto da página atual, quanto o somatório geral (o *outputPanel autoUpdate=true* é para que ao ser filtrada a tabela, esse somatório também seja atualizado):

```
<p:outputPanel autoUpdate="true">
  <p:panelGrid columns="2">

    <h:outputLabel value="Salário (Página Atual):"/>
    <h:outputText value="#{x:sum(personMB.dataModel.wrappedData, 'salary')}">
      <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>
    </h:outputText>

    <h:outputLabel value="Salário Total:"/>
    <h:outputText value="#{x:sum(personMB.dataModel, 'salary')}">
      <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>
    </h:outputText>

  </p:panelGrid>
</p:outputPanel>
```

Exibição de somatório no footer do dataTable

Para exibir o somatório no footer do dataTable, é necessário usar o “facet=footer”, como mostrado abaixo:

```
<p:column headerText="Salário" sortBy="#{person.salary}"
  style="text-align: right;">
  <h:outputText value="#{person.salary}">
    <f:convertNumber minFractionDigits="2" maxFractionDigits="2" />
  </h:outputText>
  <f:facet name="footer">
    <h:outputText value="#{x:sum(personMB.dataModel, 'salary')}" id="sumSalary">
      <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>
    </h:outputText>
  </f:facet>
</p:column>
```

Observação Importante: O primefaces ao fazer os eventos próprios do datatable, como paginação, filtrar e ordenar, não atualiza o footer, isso é um comportamento do componente *p:dataTable* (que até a versão 5.1 ainda persiste), já discutido em fóruns e no próprio primefaces, segue alguns dos links abaixo:

- <https://code.google.com/p/primefaces/issues/detail?id=1558>
- <http://forum.primefaces.org/viewtopic.php?f=3&t=18347>
- <http://stackoverflow.com/questions/24452834/how-to-ajax-update-an-item-in-the-footer-of-a-primefaces-datatable>
- <https://code.google.com/p/primefaces/issues/detail?id=4756>

Para resolver isso, uma das soluções é usar um *remoteCommand*, e inserir um *p:ajax* para cada evento, que será disparado ao completar os eventos da tabela (gerando assim uma nova requisição). Essa solução é mostrada abaixo:

```

<h:form id="formListPerson">

    <p:remoteCommand process="@this" update=":formListPerson:dataTablePerson:sumSalario"
                    name="updateSomatorio"/>

    <p:dataTable id="dataTablePerson" (... outros atributos omitidos) >
        <p:ajax event="sort" oncomplete="updateSomatorio()" />
        <p:ajax event="page" oncomplete="updateSomatorio()" />
        <p:ajax event="filter" oncomplete="updateSomatorio()" />
        <p:column headerText="Salário" sortBy="#{person.salary}" style="text-align: right;">
            <h:outputText value="#{person.salary}">
                <f:convertNumber minFractionDigits="2" maxFractionDigits="2" />
            </h:outputText>
            <f:facet name="footer">
                <h:outputText value="#{x:sum(personMB.dataModel, 'salary')}}" id="sumSalario">
                    <f:convertNumber minFractionDigits="2" maxFractionDigits="2"/>
                </h:outputText>
            </f:facet>
        </p:column>
    </p:dataTable>
</h:form>

```

8. Internacionalização de mensagens do BeanValidation

8.1. Introdução

A api 2.0 do jsf trouxe integração automática com a especificação BeanValidation (JSR 303). Essa integração facilitou muito a utilização das anotações do bean validation em projetos que utilizam JSF.

O problema dessa integração é que as mensagens não são formatadas para um determinado atributo, por exemplo, uma classe `Person` com o atributo `name` anotado com `@NotNull`, o jsf exibiria a mensagem “value is required”, ou seja, é uma mensagem genérica.

Nesse exemplo o ideal seria exibir “Name is required” e melhor ainda exibir “Nome é obrigatório” no caso do Locale `pt_BR`.

Diante desse problema o `xpert-framework` disponibiliza um *Interpolator* próprio para tratar essas mensagens, sendo que essa classe é uma implementação da classe `javax.validation.MessageInterpolator` da api do java-ee e uma implementação do `BeanValidator` que é filha de `javax.faces.validator.BeanValidator`.

8.2. Configuração

Para indicar o interpolator do `xpert-framework` deve ser criado o arquivo **validation.xml** que é um arquivo de configuração próprio da api validation do java-ee dentro da pasta `META-INF` e indicar a classe `com.xpert.i18n.CustomInterpolator`.

Exemplo:

```
<validation-config xmlns="http://jboss.org/xml/ns/javax/validation/configuration"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/configuration validation-configuration-1.0.xsd">
  <message-interpolator>com.xpert.i18n.CustomInterpolator</message-interpolator>
</validation-config>
```

Com o interpolator configurado, é necessário indicar o resource bundle a ser utilizado pelo `xpert framework`, ele pode ser indicado através do **xpert-config.xml** na seção **resource-bundle**.

Para que o jsf reconheça o novo validador é necessário sobrescrever no **faces-config.xml** o `BeanValidator`:

```
<validator>
  <validator-class>com.xpert.core.validation.BeanValidator</validator-class>
  <validator-id>javax.faces.Bean</validator-id>
</validator>
```

8.3. Como é feita a internacionalização

O nome do campo se baseia na seguinte lógica:

Nome simples da classe concatenado com “**. (ponto)**” concatenado com o **nome do atributo**.

O nome da classe deve iniciar com minúsculo (*lower camel case*).

Considerando a classe Person abaixo o com atributo name:

```
@Entity
public class Person {

    @NotBlank
    private String name;

}
```

O arquivo de internacionalização ficaria:

pt_BR (Português do Brasil)

person.name=Nome

en (Inglês)

person.name=Name

Assim a mensagem ficaria:

Nome é Obrigatório

Para colocar uma mensagem personalizada basta adicionar o atributo message na anotação. Exemplo:

```
@Entity
public class Person{

    @NotBlank(message="Name is required. This is a custom message")
    private String name;

}
```

8.4. Tipos de validações suportadas

Algumas das validações da especificação são suportadas e além delas algumas do hibernate-validator também. Algumas validações valores podem ser definidos, como é o caso do @Size onde pode-se indicar o min e o max, a mensagem gerada seria algo do tipo “Valor deve possuir no máximo \${min} caracteres e no máximo {max}”

Java-ee api

- NotNull
- Max
- Min
- Size
- DecimalMax
- DecimalMin
- Past
- Future

Hibernate validator

- NotBlank
- NotEmpty
- Email
- Range
- URL

Para mais informações sobre cada tipo de validação é importante consultar a especificação, pois a parte de validação é feita a através dela e suas específicas implementações e o xpert-framework fica a cargo apenas a formatação das mensagens.

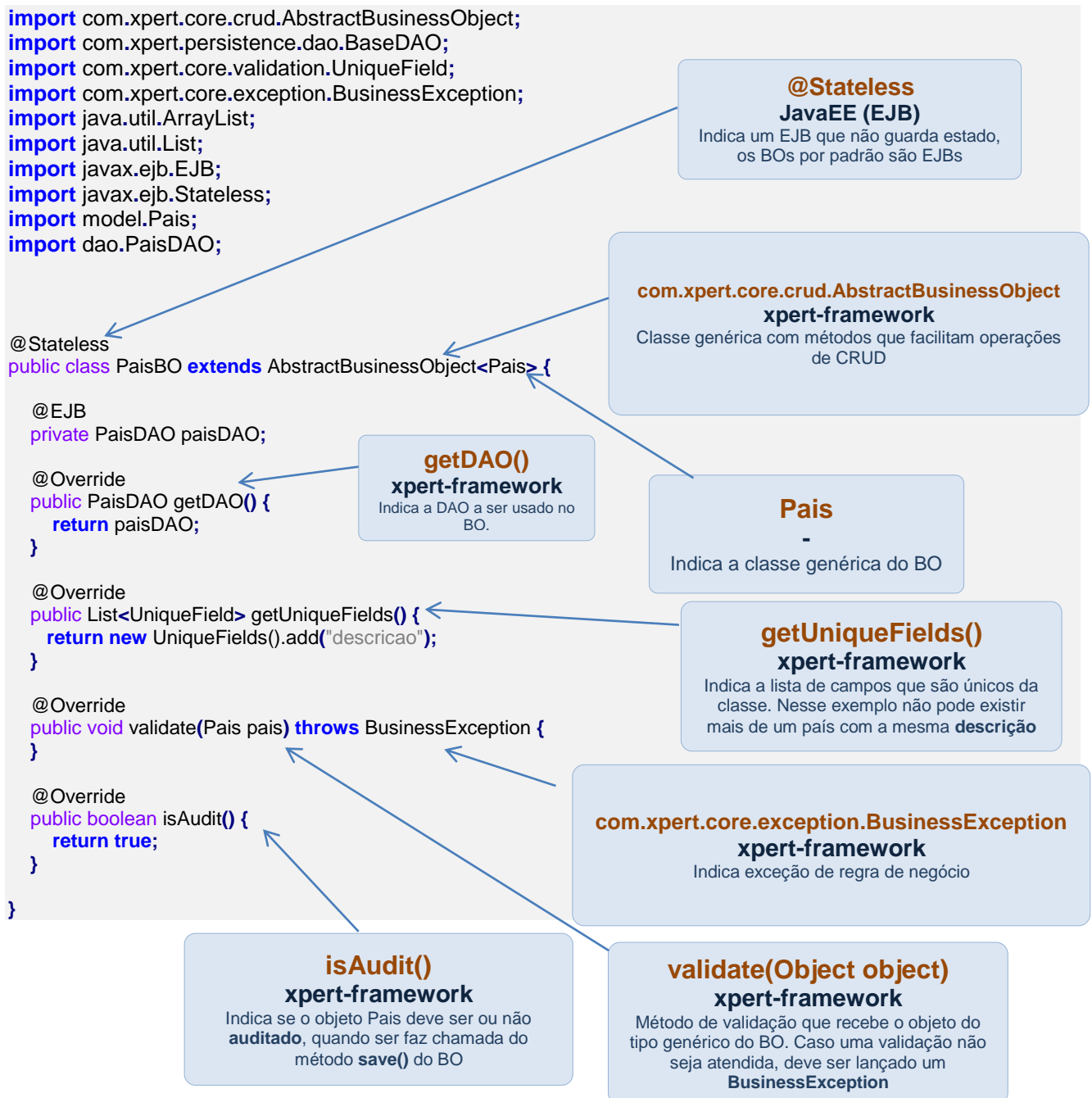
9. Artefatos padronizados para CRUDs

9.1. Classe de negócio - AbstractBusinessObject

Classe genérica de regra de negócio para facilitar a Criação de CRUDs.

9.1.1. Estrutura de um BusinessObject

Exemplo de um BO para País:



9.1.2. Principais métodos da classe

9.1.2.1. *getUniqueFields* - Definir Unicidade dos campos

Este Método retorna uma lista de UniqueField com os campos que não devem ser repetidos. Exemplo: um Estado de um País não pode conter descrição ou siglas repetidas.

```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields()
        .add("descricao", "pais")
        .add("sigla", "pais");
}
```

9.1.2.2. *isAudit* - Definir se a entidade deve ser auditada

Este método define se a entidade deve ser auditada ao se chamar o método **save** do BO.

```
@Override
public boolean isAudit() {
    return true;
}
```

9.1.2.3. *validate* - Validação simples de uma entidade

Este método recebe a entidade do BO e é possível se fazer validação lançando um BusinessException.

Exemplo:

```
@Override
public void validate(Pessoa pessoa) throws BusinessException {
    if (pessoa.getPerfil() == null) {
        throw new BusinessException("required.perfil");
    }
}
```

Este método recebe apenas a própria entidade, caso seja necessário passar mais objetos para a validação, deve ser feita uma sobrecarga do método, e para que o novo método criado seja chamado é necessário sobrescrever o método **save()**.

9.1.2.4. *save* - Método para persistir a Entidade

O método **save** da classe abstrata obedece o seguinte fluxo:

1. Chamar o método **validade()**
2. Chamar o método **validateUniqueFields()** (validação os campos definidos no **getUniqueFields()**)
3. Lançar exceção caso exista problemas na validação
4. Chamar **persist** caso seja um novo objeto, ou **merge** para atualizar um objeto do DAO.

Para mudar esse fluxo ou chamar algum validate específico, o método **save()** deve ser sobrescrito, ou ainda sobrecarregado.

9.1.2.5. remove - Método excluir a entidade

Chama o método `dao.remove()`, para excluir o objeto. Pode ser passado o próprio objeto por parâmetro, ou o id (quando este for um *Number*). O método `remove` no dao é um método que chama o `entityManager.remove()`.

Exemplo:

```
public void remove() {  
    try {  
        pessoaBO.remove(getEntity());  
    } catch (DeleteException ex) {  
        FacesMessageUtils.error(XpertResourceBundle.get("objectCannotBeDeleted"));  
    }  
}
```

9.1.2.6. delete - Método excluir a entidade

Chama o método `dao.delete()`, para excluir o objeto. O `delete` no dao, é um método montado via JPQL que faz a exclusão.

Exemplo:

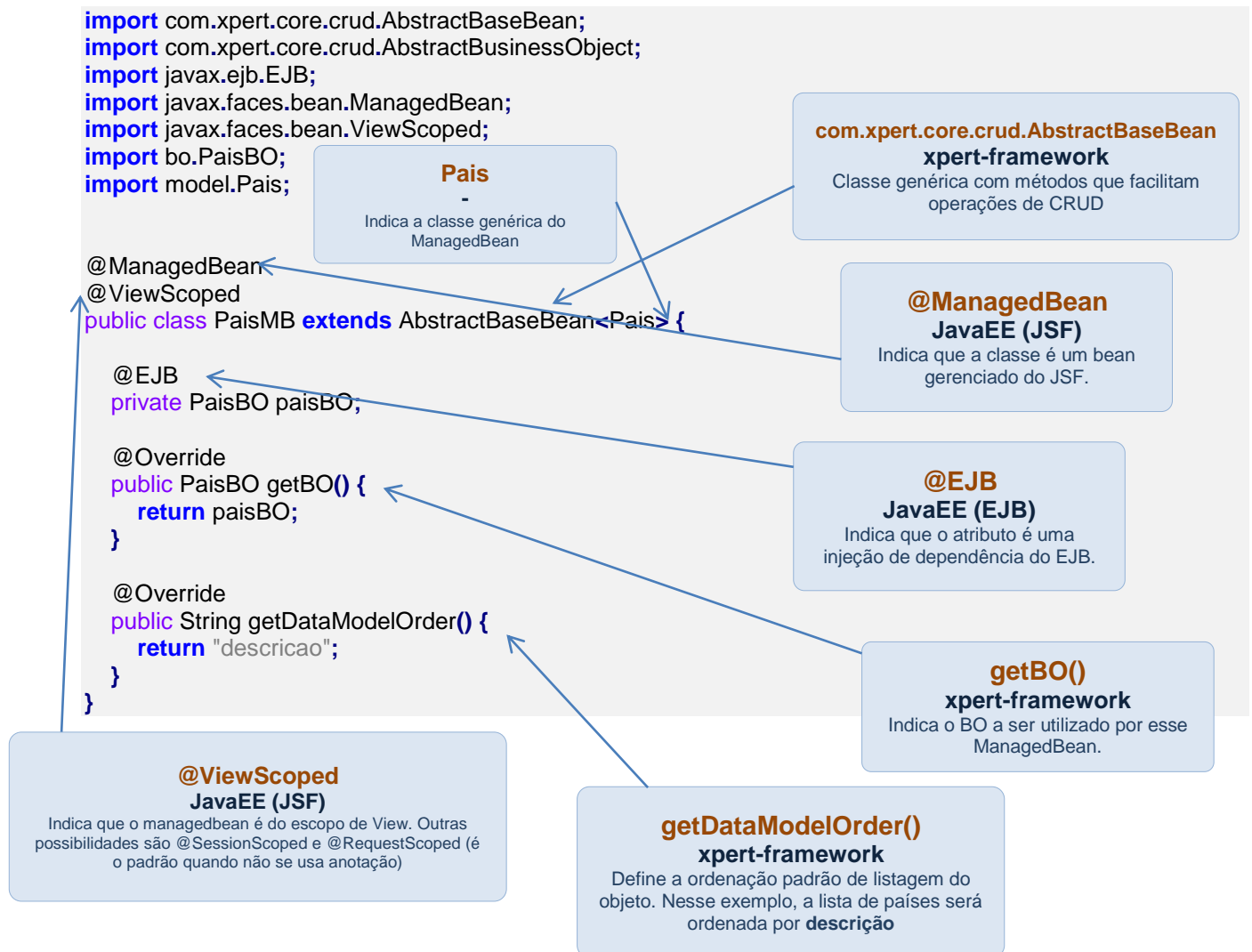
```
public void delete() {  
    try {  
        pessoaBO.delete(id);  
    } catch (DeleteException ex) {  
        FacesMessageUtils.error(XpertResourceBundle.get("objectCannotBeDeleted"));  
    }  
}
```

9.2. ManagedBean (Controller) - AbstractManagedBean

Managed Bean genérico para criação de CRUDs.

9.2.1. Estrutura de um ManagedBean

Exemplo de um ManagedBean para País:



9.2.2. Principais métodos da classe

9.2.2.1. *getDataModelOrder* - Definir ordenação padrão da Listagem

Este método define a ordenação na listagem do LazyDataModel (primefaces).

Exemplo - Ordenar pelo atributo *nome*:

```
@Override
public String getDataModelOrder() {
    return "nome";
}
```

Pode ainda ser definida se a ordenação é asc ou desc. Exemplo:

```
@Override
public String getDataModelOrder() {
    return "dataCadastro DESC";
}
```

9.2.2.2. *postConstruct* - Método chamado no evento @PostConstruct

O método **postConstruct()** é chamado obedecendo o seguinte fluxo:

1. Verificar se foi enviado o campo **id** nos parâmetros
2. Caso seja enviado **id** recuperar a entidade do banco e setar como entidade atual
3. Chamado o **createDataModel()** para criar o LazyDataModel da entidade.
4. Chamado o método **init()**.

9.2.2.3. *init* - Método chamado após o evento @PostConstruct

Este método é chamado no @PostConstruct, após a entidade ser carregada.

9.2.2.4. *getDataModelRestrictions* - Restringir dados do LazyDataModel

Este método retorna uma lista de restrições que devem ser usadas ao se fazer a consulta para renderizar o dataTable da entidade.

Exemplo - retornar apenas os registros de um usuário:

```
@Override
public List<Restriction> getDataModelRestrictions() {
    Restrictions restrictions = new Restrictions();
    restrictions.add("usuario", sessaoUsuarioMB.getUser());
    return restrictions;
}
```

9.2.2.5. *preSave* e *postSave* - Chamar eventos antes e depois de salvar a entidade

O método **preSave()** é chamado antes de salvar a entidade, enquanto o **postSave()** após. Esses métodos podem ser utilizados para controle de tela, por exemplo, quando salvar a entidade x, carregar a lista y. Lógicas complexas devem ser evitadas nesses métodos, onde o mais aconselhável é sobrescrever o método **save()** para se ter um código mais legível.

9.2.2.6. create - Criar uma nova instância da entidade

O método **create()** cria uma nova instância do objeto **entity** do Managed Bean.

9.2.2.7. getOrderByHandler - Customizar Ordenação do dataTable no sortBy

Com este método é possível definir uma ordenação customizada ao se ordenar a coluna do dataTable.

Exemplo, ao clicar para ordenar a coluna **number** adicionar também a coluna **year** na ordenação:

```
<p:column headerText="#{msg['object.number']}" sortBy="#{object.number}">
  <h:outputText value="#{object.number}"/>
</p:column>
```

Veja que o **sortBy** da tabela está pelo campo **object.number**. Sendo assim é possível tratar essa ordenação no ManagedBean sobrescrevendo o método **getOrderByHandler** da seguinte forma:

```
@Override
public OrderByHandler getOrderByHandler() {
    return new OrderByHandler() {
        @Override
        public String getOrderBy(String orderBy) {
            if (orderBy != null && orderBy.equals("number")) {
                return "number, year";
            }
            return orderBy;
        }
    };
}
```

Dessa forma é possível tratar as mais diversas ordenações, e para cada campo fazer uma maneira personalizada.

9.2.2.8. getFilterByHandler - Customizar restrições dos filtros do dataTable no filterBy

É possível manipular os filtros do **filterBy** do **p:column** através o método **getFilterByHandler**, para mais informações sobre o **FilterByHandler** veja no capítulo [LazyDataModelImpl - FilterByHandler](#).

9.2.2.9. isLoadEntityOnPostConstruct - definir se o objeto será carregado a partir do id passado na requisição

Por padrão este método é **true**, setando este método como false caso seja passado um id nos parâmetros (via GET) este não será carregado.

9.2.2.10. setDialog - definir o dialog do cadastro

Define o dialog a ser utilizado quando o método "save()" for executado, esse método é útil para se utilizar em formulários que são provenientes de dialog, pois ao salvar o objeto esse dialog será fechado.

9.2.2.11. *getDataModelJoinBuilder* - definir o dialog do cadastro

Define os joins a serem utilizados no data model da consulta. Definir os joins pode vir a melhorar muito o desempenho da LazyDataModel.

Exemplo de uso:

```
@Override
public JoinBuilder getDataModelJoinBuilder() {
    return new JoinBuilder("u")
        .leftJoinFetch("u.empresa", "e");
}
```

Ao se definir o join builder é importante definir um alias, nesse exemplo acima o alias foi "u", então em outros métodos que usem as propriedades para as consultas, deve-se usar o alias também, como é o caso do *getDataModelOrder*.

```
@Override
public String getDataModelOrder() {
    return "u.nome";
}
```

Importante! Não é aconselhado utilizar *joins fetch* com o listas (*@ManyToMany* ou *@OneToMany*), isso é um comportamento de frameworks orm como o hibernate, ao se fazer isso o poder a query paginada no banco é perdido, e a lista passa a ser paginada em memória.

9.2.2.12. *reloadEntity* - recarregar objeto atual

Esse método recarrega o objeto atual (*getEntity()*), através de uma nova consulta no banco de dados.

Deve ser usado para quando se quer recarregar o objeto, sincronizar novamente com o banco de dados, por exemplo.

9.2.2.13. *getDataModelLazyCountType* - recarregar objeto atual

Define o tipo de count do LazyDataModel, por padrão retorna o *LazyCountType.ALWAYS*. Esse método pode ser sobrescrito, por exemplo para não fazer count nas consultas paginadas:

```
@Override
public LazyCountType getDataModelLazyCountType() {
    return LazyCountType.NONE;
}
```

Para mais informações ver a seção [LazyCountType](#).

9.2.2.14. *putEntityInRequest* - Inserir entidade na requisição

Este método adiciona a entidade na requisição e caso o atributo "outcome" seja definido, a página definida nele será exibida.

Um exemplo do uso desse método é na geração de CRUD, como mostrado abaixo na coluna de um dataTable:

```
<p:commandButton icon="ui-icon-pencil" action="#{contaCorrenteMB.putEntityInRequest}"
                 ajax="false" title="#{xmsg['edit']}" >
  <f:setPropertyActionListener value="#{contaCorrente}" target="#{contaCorrenteMB.entity}" />
  <f:setPropertyActionListener value="createContaCorrente" target="#{contaCorrenteMB.outcome}" />
</p:commandButton>
```

Nesse exemplo a página a ser exibida é “createContaCorrente”.

9.2.2.15. *getOutcome/setOutcome - definindo a página*

Os métodos `getOutcome` e `setOutcome`, acessam e setam os valores da página a ser exibida. Essa página será usada por exemplo no método “putEntityInRequest”.

10. Tratamento de Regra de Negócio com o Business Exception

O **BusinessException** (com.xpert.core.exception) é uma *Exception* utilizada para **Regras de Negócio**. É uma exceção utilizada em muitas partes do framework, como o *FacesMessagesUtils* e *AbstractBusinessObject*.

Considerando o arquivo de internacionalização:

```
business.cargaHorarioAcima=Carga horária não pode ser acima de 9h
business.dataNascimentoNaoPodeSerFutura=Data de nascimento não pode ser uma data futura
business.cargaHorariaNaoPermitida=Carga horária {0} não permitida
business.pessoaComCargaHorariaInvalida=Pessoa de nome {0} com a carga horaria inválida. Carga horária informada:
{1}
```

10.1. Validação Básica

Regras:

1. Uma pessoa não pode ter a carga horaria acima de 9h.

```
public void validate(Pessoa pessoa) throws BusinessException {
    //carga horaria nao pode ultrapassar 9h
    if (pessoa.getCargaHoraria() > 9) {
        throw new BusinessException("business.cargaHorarioAcima");
    }
}
```

10.2. Validação com Múltiplas regras

Regras:

1. Uma pessoa não pode ter a carga horaria acima de 9h.
2. A data de nascimento não pode ser uma data futura

```
public void validateMultipleException(Pessoa pessoa) throws BusinessException {
    BusinessException exception = new BusinessException();

    if (pessoa.getCargaHoraria() > 9) {
        exception.add("pessoa.business.cargaHorarioAcima");
    }
    if (pessoa.getDataNascimento().after(new Date())) {
        exception.add("pessoa.business.datanascimentoNaoPodeSerFutura");
    }
    exception.check();
}
```

O método **exception.ckeck()** verifica se alguma exceção foi adicionada e lança a exceção.

10.3. Passagem de Parâmetros na mensagem

Algumas mensagens possuem parâmetros dinâmicos. Exemplo:

“Carga horária 10h não permitida.” (Note que 10h pode ser um parâmetro informado pelo usuário).

```
public void validate(Pessoa pessoa) throws BusinessException {  
    /*  
     * Mensagem no arquivo de internacionalizacao: Carga horária {0} não permitida.  
     * Esse parametro "{0}" espera um valor, no exemplo abaixo para uma carga horária 20 a mensagem ficaria:  
     *  
     * "Carga horária 20h não permitida"  
     */  
    if (pessoa.getCargaHoraria() > 9) {  
        throw new BusinessException("business.cargaHorariaNaoPermitida", pessoa.getCargaHoraria()+"h");  
    }  
  
    /*  
     * Podem ser passados múltiplos parâmetros, basta seguir a ordem dos parâmetros "{0}, {1}, {2}..."  
     * Uma pessoa de nome "Maria" e carga horária informada de 10, a mensagem ficaria:  
     *  
     * "Pessoa de nome Maria com a carga horária inválida. Carga horária informada: 10h"  
     */  
    if (pessoa.getCargaHoraria() > 9) {  
        throw new BusinessException("business.pessoaComCargaHorariaInvalida", pessoa.getNome(),  
            pessoa.getCargaHoraria()+"h");  
    }  
}
```


11. Exibindo mensagem com o FacesMessageUtils

FacesMessageUtils (com.xpert.faces.utils) é uma classe de utilitários para exibição de mensagens na View.

11.1. Utilização Básica

Informação

```
FacesMessageUtils.info("mensagem");
```

Warning

```
FacesMessageUtils.warning("mensagem");
```

Erro

```
FacesMessageUtils.error("mensagem");
```

Fatal

```
FacesMessageUtils.fatal("mensagem");
```

11.2. Passagem de Parâmetros na mensagem

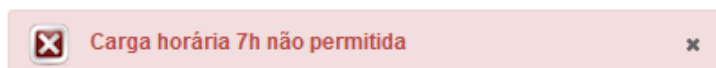
Considerando o arquivo de internacionalização:

```
pessoa.business.cargaHorariaNaoPermitida=Carga horária {0} não permitida
```

Chamada da mensagem:

```
FacesMessageUtils.error("pessoa.business.cargaHorariaNaoPermitida", 7+"h");
```

Resultado na view utilizando **p:messages** (Primefaces):



11.3. Business Exception em um FacesMessagesUtils

Um Business Exception pode ser passado para exibição no FacesMessagesUtils.

Caso essa exception possua várias mensagens, o resultado renderizado será uma lista de mensagens.

```
public void save(Pessoa pessoa){  
    try {  
        validate(pessoa);  
    } catch (BusinessException ex) {  
        FacesMessageUtils.error(ex);  
    }  
}
```

12. Restrictions

12.1. Introdução

Restrictions é maneira de se adicionar restrições nas consulta, são utilizadas em muitos módulos do xpert-framework, como o **QueryBuilder**, **BaseDAO**, **LazyDataModel**.

Ao longo dessa documentação muitos exemplos utilizam as restrições, no **AbstractBaseBean** por exemplo é possível informar as restrições do LazyDataModel

12.2. Tipos de Restrictions

Os tipos de restrições são definidos na enum “**RestrictionType**”, como mostrado abaixo:

Tipo de Restrição	Descrição
RestrictionType.EQUALS	igual a
RestrictionType.NOT_EQUALS	diferente de
RestrictionType.GREATER_THAN	maior que
RestrictionType.LESS_THAN	menor que
RestrictionType.GREATER_EQUALS_THAN	maior ou igual que
RestrictionType.LESS_EQUALS_THAN	menor ou igual que
RestrictionType.LIKE	que iniciam, terminam ou contenha
RestrictionType.NOT_LIKE	que não iniciam, não termine e não contenha
RestrictionType.IN	que estejam contidos em
RestrictionType.NOT_IN	que não estejam contidos em
RestrictionType.NULL	que seja null
RestrictionType.NOT_NULL	que não sejam null
RestrictionType.DATE_TABLE_FILTER	Tipo especial utilizado no LazyDataModelImpl
RestrictionType.OR	Cláusula “or”
RestrictionType.START_GROUP	Início de um agrupamento (abertura de parênteses), útil ao ser combinado com o “or”
RestrictionType.END_GROUP	Fim de um agrupamento (fechamento de parênteses), útil ao ser combinado com o “or”
RestrictionType.QUERY_STRING	Alguma String personalizada que será adicionada na consulta

Para restrições do tipo **LIKE** ou **NOT_LIKE**, é possível definir, se é like no início, no fim ou ambos (padrão). Os possíveis tipos são:

Tipo de Restrição LIKE	Descrição
LikeType.BEGIN	que inicie com
LikeType.END	que termine com
LikeType.BOTH	que contenha (padrão quando nenhum é definido)

Exemplos:

Recuperar pessoas com o nome Maria, ordenados por **id**:

```
Restrictions restrictions = new Restrictions();
restrictions.add("nome", "Maria");
pessoas = pessoaDAO.list(restrictions, "id");
```

Pode ser usado ainda em cadeia:

```
Restrictions restrictions = new Restrictions();
restrictions.add("nome", "Maria").add("cargo", "Auxiliar");
pessoas = pessoaDAO.list(restrictions);
```

Recuperar pessoas que possuem a String “Silva” no nome:

```
Restrictions restrictions = new Restrictions();
restrictions.add("nome", RestrictionType.LIKE, "Silva");
pessoas = pessoaDAO.list(restrictions, "nome");
```

Recuperar pessoas que possuem carga horaria entre 6h e 8h:

```
Restrictions restrictions = new Restrictions();
restrictions.add("cargaHoraria", RestrictionType.GREATER_EQUALS_THAN, 6);
restrictions.add("cargaHoraria", RestrictionType.LESS_EQUALS_THAN, 8);
pessoas = pessoaDAO.list(restrictions);
```

O mesmo exemplo utilizando os respectivos métodos:

```
Restrictions restrictions = new Restrictions();
restrictions.greaterEqualsThan("cargaHoraria", 6);
restrictions.lessEqualsThan("cargaHoraria", 8);
pessoas = pessoaDAO.list(restrictions);
```

Existem muitas combinações possíveis quando se trabalha com Restrictions.

12.3. Utilizando métodos em cadeia

Com a classe **Restrictions** é possível encadear os métodos de maneira que o código fique legível, os possíveis métodos dos restrictions são:

Método	Equivalente a
equals(String property, Object value)	RestrictionType.EQUALS
notEquals(String property, Object value)	RestrictionType.NOT_EQUALS
greaterThan(String property, Object value)	RestrictionType.GREATER_THAN
lessThan(String property, Object value)	RestrictionType.LESS_THAN
greaterEqualsThan(String property, Object value)	RestrictionType.GREATER_EQUALS_THAN
lessEqualsThan(String property, Object value)	RestrictionType.LESS_EQUALS_THAN
like(String property, Object value)	RestrictionType.LIKE
notLike(String property, Object value)	RestrictionType.NOT_LIKE
in(String property, Object value)	RestrictionType.IN
notIn(String property, Object value)	RestrictionType.NOT_IN
isNull(String property)	RestrictionType.NULL
isNotNull(String property)	RestrictionType.NOT_NULL
or()	RestrictionType.OR
startGroup()	RestrictionType.START_GROUP
endGroup()	RestrictionType.END_GROUP
addQueryString(String property)	RestrictionType.QUERY_STRING

Exemplo de Utilização

Todos de nome “MARIA” e status “true”

```
Restrictions restrictions = new Restrictions();
restrictions.equals("nome", "MARIA")
    .equals("status", true);
```

Todos que nome contenha “MARIA” e status “true”

```
Restrictions restrictions = new Restrictions();
restrictions.like("nome", "MARIA")
    .equals("status", true);
```

12.4. Restrictions - Utilizando a cláusula “OR”

Para se utilizar a cláusula “or” podemos combiná-la com o **startGroup()** e o **endGroup()** esses métodos indicam respectivamente o início de um grupo e o fim.

Exemplos

Para montar o seguinte JPQL:

```
FROM person WHERE nome = 'MARIA' OR nome = 'JOSE' OR status = true
```

Ficaria assim:

```
Restrictions restrictions = new Restrictions();
restrictions.equals("nome", "MARIA")
    .or()
    .equals("nome", "JOSE")
    .or()
    .equals("status", true);
```

Utilizando agrupamento de cláusulas

Para se fazer agrupamentos de cláusulas, podemos utilizar os métodos **startGroup()** e **endGroup()** para delimitar os grupos. Considere o seguinte select:

```
FROM person WHERE (nome = 'MARIA' AND status = true) OR (code = '123')
```

Ficaria assim:

```
Restrictions restrictions = new Restrictions();
restrictions.startGroup()
    .equals("nome", "MARIA").equals("status", true)
    .endGroup()
    .or()
    .equals("code", "123");
```

A utilização dos métodos **startGroup()** e **endGroup()** torna legível consultas complexas, como essa mostrada abaixo:

```
FROM person
WHERE (nome = 'MARIA' OR nome = 'JOSE')
    AND (code = '123' OR code = '321')
    AND status IS NOT NULL
```

Ficaria assim:

```
Restrictions restrictions = new Restrictions();
restrictions.startGroup()
    .equals("nome", "MARIA").or().equals("nome", "JOSE")
    .endGroup()
    .startGroup()
    .equals("code", "123").or().equals("code", "321")
    .endGroup()
    .isNotNull("status");
```

13. Consultas dinâmicas com o QueryBuilder

13.1. Introdução

A classe QueryBuilder é um utilitário para facilitar a criação de query dinâmica, ela facilita desde comandos mais simples, como um “count”, ou “sum”, como consultas mais complexas, que utilizem “restrições” e cláusulas “or”.

13.2. Criando um QueryBuilder

É possível criar um **QueryBuilder** a partir de um **EntityManager**, como mostrado abaixo:

```
QueryBuilder queryBuilder = new QueryBuilder(entityManager);
```

Além disso, os DAOs que implementam o BaseDAO possuem o método getQueryBuilder() esse método retorna uma nova instância do QueryBuilder:

```
QueryBuilder queryBuilder = baseDAO.getQueryBuilder();
```

13.3. Selecionando todos os registros

O método “from” define a classe que vai ser realizada a consulta. O método “getResultList()” realiza a consulta dos registros:

```
List<Person> people = queryBuilder.from(Person.class).getResultList();
```

13.4. Ordenação do Resultado

O método “orderBy” define a ordenação do resultado:

```
List<Person> people = queryBuilder.from(Person.class)  
    .orderBy("name").getResultList();
```

Múltiplas ordenações pode ser definidas (pode ser definido o ASC ou DESC):

```
List<Person> people = queryBuilder.from(Person.class)  
    .orderBy("name, code DESC").getResultList();
```

13.5. Selecionando um resultado único

O método “getSingleResult()” chama o “getSingleResult()” da interface Query do JPA, mas ele não lança um “NoResultException” quando não for encontrado resultado.

```
Person person = queryBuilder.from(Person.class).getSingleResult();
```

13.6. Utilização de Restrictions

Os mesmos métodos utilizados nas restrictions como o “like”, “equals”, “greaterThan” podem ser utilizados no QueryBuilder, como mostrado abaixo:

```
List<Person> people = queryBuilder.from(Person.class)  
    .like("name", "Maria")  
    .orderBy("name").getResultList();
```

O exemplo acima equivale ao JPQL:

```
FROM Person.class WHERE name LIKE '%Maria%' ORDER BY name
```

Outro exemplo, recuperando a pessoa de “code” 20 (retorna apenas um resultado):

```
Person> person = queryBuilder.from(Person.class)
    .equals("code", 20)
    .getSingleResult();
```

O exemplo acima equivale ao JPQL:

```
FROM Person.class WHERE code = 20
```

13.7. Definindo o Alias

O “alias” da query pode ser definido no método “from”:

```
List<Person> people = queryBuilder.select("p").from(Person.class, "p");
```

O exemplo acima equivale ao JPQL:

```
SELECT p FROM Person.class p
```

Ao utilizar as restrições pode-se utilizar o alias definido:

```
List<Person> people = queryBuilder.select("p")
    .from(Person.class, "p")
    .equals("p.name", "Peter")
    .orderBy("p.name")
    .getResultList();
```

O exemplo acima equivale ao JPQL:

```
SELECT p FROM Person.class p WHERE p.name = 'Peter' ORDER BY p.name
```

13.8. Joins

Os métodos “innerJoin()”, “innerJoinFetch()”, “rightJoin()”, “rightJoinFetch()”, “leftJoin()” e “leftJoinFetch()” podem ser utilizados para definir os joins da consulta.

```
List<Person> people = queryBuilder.select("p")
    .from(Person.class, "p")
    .innerJoin("p.group", "g")
    .orderBy("g.description")
    .getResultList();
```

O exemplo acima equivale ao JPQL:

```
SELECT p FROM Person.class p
    INNER JOIN p.group g
    ORDER BY g.description
```

13.9. Select Distinct

Para usar a cláusula “DISTINCT” deve-se utilizar o método “selectDistinct”. Uma das grandes utilidades dele, é evitar duplicidade na lista quando se usa o “join fetch”, como mostrado abaixo:

```
List<Person> people = queryBuilder.selectDistinct("p")
    .from(Person.class, "p")
    .leftJoinFetch("p.permissions", "pe")
    .getResultList();
```

13.10. Definindo a quantidade máxima de resultados

Para definir o máximo de resultados na consulta deve-se utilizar o método “setMaxResults”, abaixo a consulta retornará no máximo 10 resultados:

```
List<Person> people = queryBuilder.from(Person.class)
    .orderBy("name")
    .setMaxResults(10)
    .getResultList();
```

13.11. Consulta paginada

Assim como no Entity Manager é possível definir o “firstResult” da consulta, através do método “setFirstResult”, combinado com o “setMaxResults”, possibilitando assim a construção de consultas paginadas. O exemplo abaixo mostra uma consulta para os resultado de 0 a 10:

```
List<Person> people = queryBuilder.from(Person.class)
    .orderBy("name")
    .setFirstResult(0)
    .setMaxResults(10)
    .getResultList();
```

13.12. Debug da Query

Para visualizar os detalhes da query, como a string gerada e os parâmetros, pode se chamar o método “debug()”:

```
List<Person> people = queryBuilder.from(Person.class)
    .orderBy("name")
    .debug()
    .getResultList();
```

13.13. Definindo os atributos na query

Para definir quais os atributos serão selecionados na query, pode-se utilizar o método “select”, no exemplo abaixo apenas os atributos “name,code,id” da classe “Person” serão selecionados, note que é necessário passar a class no método “getResultList()”:

```
List<Person> people = queryBuilder.select("name,code,id")
    .from(Person.class)
    .orderBy("name")
    .getResultList(Person.class);
```

No exemplo acima, apesar de a classe Person.class possuir vários atributos, na consulta apenas serão retornados os atributos informados.

13.14. Realizando o “count” dos registros

O método count retorna o total de registros (Long), como mostrado abaixo:


```
Long count = queryBuilder.from(Person.class).count();
```

O exemplo acima equivale ao JPQL:

```
SELECT COUNT(*) FROM Person.class
```

13.15. Somatório com o “sum”

Para realizar o somatório pode-se utilizar o método “sum()”, é necessário o cast para o tipo esperado:

```
BigDecimal sumSalary = (BigDecimal) queryBuilder.from(Person.class).sum("salary");
```

O exemplo acima equivale ao JPQL:

```
SELECT SUM(salary) FROM Person.class
```

Pode ser definido o valor padrão quando o sum retornar “null”, nesse exemplo caso seja null o valor retornado será “BigDecimal.ZERO”:

```
BigDecimal sumSalary = (BigDecimal) queryBuilder.from(Person.class).sum("salary", BigDecimal.ZERO);
```

13.16. Cláusula Max

Para realizar a consulta utilizando o MAX deve-se utilizar o método max():

```
BigDecimal maxSalary = (BigDecimal) queryBuilder.from(Person.class).max("salary");
```

O exemplo acima equivale ao JPQL:

```
SELECT MAX(salary) FROM Person.class
```

13.17. Cláusula Min

Para realizar a consulta utilizando o MIN deve-se utilizar o método min():

```
BigDecimal minSalary = (BigDecimal) queryBuilder.from(Person.class).min("salary");
```

O exemplo acima equivale ao JPQL:

```
SELECT MIN(salary) FROM Person.class
```

13.18. Cláusula AVG

Para realizar a consulta utilizando o AVG (média) deve-se utilizar o método avg():

```
BigDecimal minSalary = (BigDecimal) queryBuilder.from(Person.class).avg("salary");
```

O exemplo acima equivale ao JPQL:

```
SELECT AVG(salary) FROM Person.class
```

14. DAO Genérico - BaseDAO

14.1. Introdução

O xpert-framework possui um conjunto de classes para a parte de Persistência, dentre eles o DAO genérico chamado **BaseDAO**, sendo que sua implementação é o **BaseDAOImpl** (ambos do pacote **com.xpert.persistence.dao**).

Os principais métodos do BaseDAO são:

- find
- listAll
- list
- unique
- count
- findAttribute
- listAttributes

14.2. Métodos do BaseDAO

14.2.1. save

Salva uma entidade no banco de dados através do *entityManager.persist()*. Nesse método é possível passar um parâmetro indicando se deve ser feita ou não a auditoria.

Exemplo de utilização

```
pessoaDAO. save(pessoa);
```

14.2.2. merge

Salva uma entidade no banco de dados através do *entityManager.merge()* e retorna uma instância dessa entidade fornecida pelo entity manager. Nesse método é possível passar um parâmetro indicando se deve ser feita ou não a auditoria.

Exemplo de utilização

```
pessoaDAO. merge(pessoa);
```

14.2.3. saveOrMerge

Salva uma entidade no banco de dados através do *entityManager.persist()* caso o id da entidade esteja nulo (inserção), ou o *entityManager.merge()* caso esteja preenchido (atualização). Nesse método é possível passar um parâmetro indicando se deve ser feita ou não a auditoria.

Exemplo de utilização

```
pessoaDAO. saveOrMerge(pessoa);
```

14.2.4. update (Implementação do Hibernate)

Atualiza uma entidade no banco de dados através do `session.update()`. Nesse método é possível passar um parâmetro indicando se deve ser feita ou não a auditoria.

Observação: na especificação do JPA, não existe o método `update`, ele é específico do Hibernate e é chamado através do objeto `Session` e não do `EntityManager`.

Exemplo de utilização

```
pessoaDAO.update(pessoa);
```

14.2.5. saveOrUpdate (Implementação do Hibernate)

Atualiza uma entidade no banco de dados através do `session.saveOrUpdate()`. Nesse método é possível passar um parâmetro indicando se deve ser feita ou não a auditoria.

Observação: na especificação do JPA, não existe o método `saveOrUpdate`, ele é específico do Hibernate e é chamado através do objeto `Session` e não do `EntityManager`.

Exemplo de utilização

```
pessoaDAO.saveOrUpdate(pessoa);
```

14.2.6. find

Seleciona do banco uma entidade a partir do seu id.

Exemplo de utilização

Recuperar a Pessoa do id 100 (tipo Long):

```
Pessoa pessoa = pessoaDAO.find(100L);
```

Pode ser utilizado passando a classe:

```
Pessoa pessoa = dao.find(100L, Pessoa.class);
```

14.2.7. listAll

Seleciona do banco todos os registros de uma entidade.

Exemplo de utilização

```
List<Pessoa> pessoas = pessoaDAO.listAll();
```

Trazendo o resultado ordenado por nome:

```
List<Pessoa> pessoas = pessoaDAO.listAll("nome");
```

14.2.8. list

Retorna uma lista de registros a partir dos parâmetro informados.

Exemplo de utilização

Recuperar pessoas com carga horária de 8h, ordenados por **id**:

```
Restrictions restrictions = new Restrictions();
restrictions.add("cargaHoraria", 8);
pessoas = pessoaDAO.list(restrictions, "id");
```

O método ainda pode ser chamado da seguinte maneira e possuirá o mesmo resultado:

```
pessoas = pessoaDAO.list("cargaHoraria", 8, "id");
```

14.2.9. unique

Semelhante ao *list()*, porém esse retorna apenas um único resultado.

Exemplo de utilização

Recuperar a pessoa de nome "JOHN".

```
Restrictions restrictions = new Restrictions();
restrictions.add("nome", "JOHN");
pessoas = pessoaDAO.unique(restrictions);
```

O método ainda pode ser chamado da seguinte maneira e possuirá o mesmo resultado:

```
pessoas = pessoaDAO.unique("nome", "JOHN");
```

O método *unique()* deve ser usado com cuidado, pois caso existam 2 resultados, ele limitará apenas a 1.

14.2.10. count

Retorna o total (java.lang.Long) de registros a partir dos parâmetro informados.

Exemplo de utilização

Recuperar total de pessoas com o nome carga horária de 8h, ordenados por **id**:

```
Restrictions restrictions = new Restrictions();
restrictions.add("cargaHoraria", 8);
Long total = pessoaDAO.count(restrictions);
```

O método ainda pode ser chamado da seguinte maneira e possuirá o mesmo resultado:

```
Long total = pessoaDAO.count("cargaHoraria", 8);
```

14.2.11. listAttributes

Recupera o objeto por demanda, informando apenas os campos a serem recuperados.

Exemplo de utilização

Supondo que existam 10 atributos no objeto Pessoa, desses atributos é necessário apenas o **nome** e o **id**, a chamada do método ficaria assim:

```
pessoas = pessoaDAO.listAttributes("id, nome");
```

Assim como os métodos citados anteriormente, é possível passar Map ou Restrictions, para filtrar a consulta.

14.2.12. findAttribute

Recupera algum atributo de um dado registro.

Exemplo de utilização

Supondo que você tenha em mãos o id de uma Pessoa e quer recuperar o perfil. Para evitar recuperar o objeto pessoa e depois fazer o *getPerfil()*, por questão de desempenho pode-se utilizar dessa maneira:

```
perfil = pessoaDAO.findAttribute("perfil", 1L);
```

Também é possível, passar o objeto no lugar do id:

```
perfil = pessoaDAO.findAttribute("perfil", pessoa);
```

14.2.13. findList

Semelhante ao **findAttribute**, porém este retorna uma lista.

Exemplo de utilização

```
perfil = pessoaDAO.findList("permissoes", 1L);
```

Também é possível, passar o objeto no lugar do id:

```
perfil = pessoaDAO.findList("permissoes", pessoa);
```

14.2.14. getInitialized

Força recuperar um objeto LAZY carregado. Supondo que Pessoa possua o atributo perfil, se em um dado momento, perfil esteja *lazy*, acessar esse objeto causaria um **LazyInitializationException**, para pegar esse atributo, uma das soluções seria usar o método **findAttribute()**, citado anteriormente, a outra seria usar **getInitialized()**, da seguinte maneira:

```
perfil = pessoaDAO.getInitialized(pessoa.getPerfil());
```

14.2.15. delete

Deleta a entidade a partir do id. Essa exclusão é feita a partir de HQL, montando uma Query que ficaria algo como : *"DELETE FROM Entidade WHERE id = ?1"*.

Exemplo de utilização

Deletar a pessoa de **id** 11:

```
pessoaDAO.delete(11L);
```

O método delete(), não considera o cascade mapeado no JPA.

14.2.16. remove

Deleta a entidade a partir do objeto. Essa exclusão é feita a partir do método *remove()* do EntityManager.

Exemplo de utilização

Deletar uma determinada pessoa:

```
pessoaDAO.remove(pessoa);
```

O método `remove()` faz chamada ao **`entityManager.remove()`** do JPA, sendo assim, ele considera o cascade.

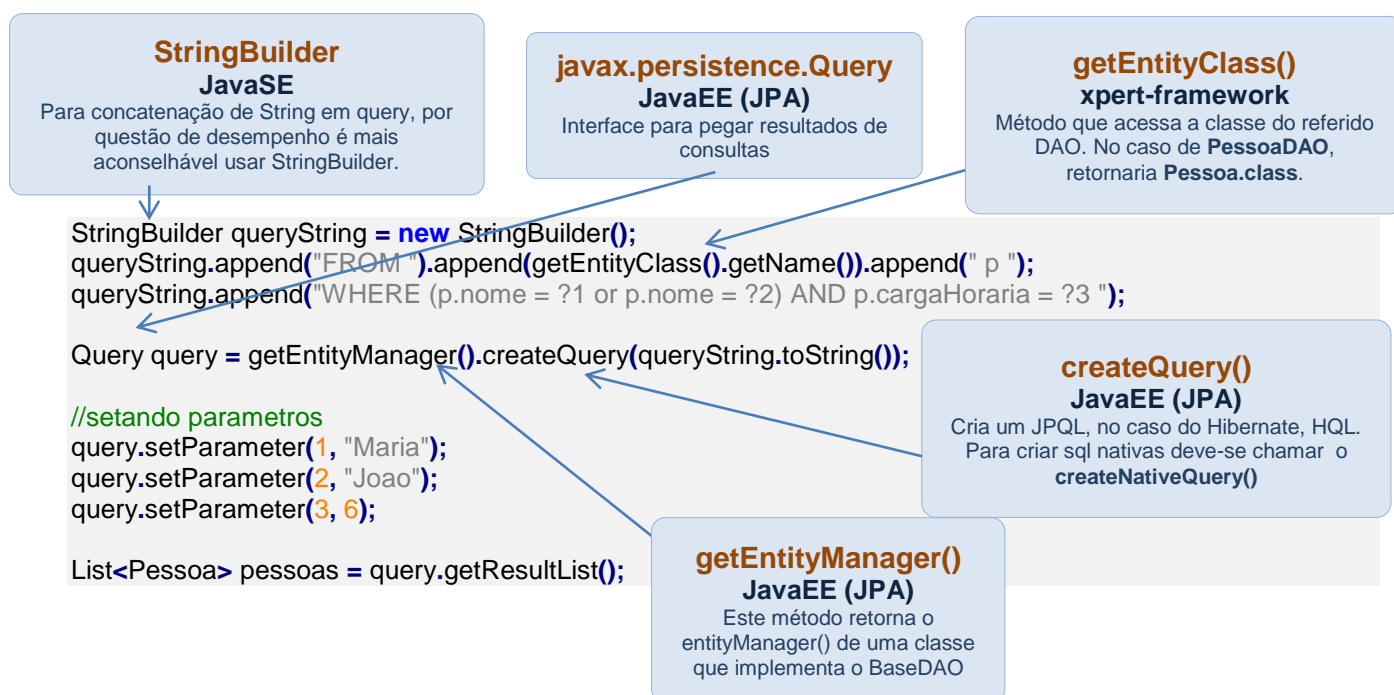
Possui um pior desempenho, pois caso um objeto passado não esteja associado a sessão, é feito um **merge**, antes da remoção para inserir este objeto na sessão.

14.2.17. Query personalizada

Algumas vezes é necessário fazer consultas personalizadas, onde os métodos genéricos citados não abrangem.

Exemplo de utilização

Consultar pessoas com nome "Maria" ou nome "Joao" e que possuam carga horaria de 6h.



A principal função deste tópico foi mostrar que o *entityManager*, e entidade (**Pessoa.class**) é acessível nas classes que implementam o **BaseDAO** e algumas boas práticas para criação de Query. Na verdade o trecho acima se refere basicamente a criação de um JPQL.

14.3. Passando Parâmetros nos métodos do BaseDAO

Nos métodos do **BaseDAO** (`count`, `list`, etc...) pode-se passar 2 tipos de parâmetros: **Map** e **Restrictions**. **Restrictions** são mais aconselháveis, pois podem determinar o tipo (`like`, `equals`, `in`, etc...).

14.3.1. Map<String, Object>

String: nome do campo, **Object**: valor do campo.

Recuperar pessoas com o nome Maria:

```
Map<String, Object> parameters = new HashMap<String, Object>();
parameters.put("nome", "Maria");
List<Pessoa> pessoas = pessoaDAO.list(parameters);
```

Podem ser inseridos vários objetos no Map, e pode-se passar a ordenação. O exemplo abaixo lista as pessoas de nome Maria e de carga horária 8h ordenados por **id**.

```
Map<String, Object> parameters = new HashMap<String, Object>();
parameters.put("nome", "Maria");
parameters.put("cargaHoraria", 8);

List<Pessoa> pessoas = pessoaDAO.list(parameters, "id");
```

A query gerada seria:

```
SELECT FROM Pessoa WHERE nome = 'Maria' and cargaHoraria = 8
```

É importante destacar que o **HashMap** não guarda a ordem de adição dos atributos, então a ordem dos campos após a cláusula **WHERE** pode não ser preservada. Para preservar a ordem pode-se usar **LinkedHashMap**.

14.3.2. Restriction e Restrictions

É possível adicionar restrições nos métodos do BaseDAO através das **Restritions** para mais detalhes da sua utilização visualizar o capítulo [Restrictions](#)

15. DataTable paginado no banco com o LazyDataModelImpl

Para criar um **LazyDataModel** (Primefaces) com paginação real no banco de dados, o xpert-framework disponibiliza a classe **LazyDataModelImpl** (com.xpert.faces.primefaces).

Esta classe recebe como parâmetro o DAO da entidade a ser consultada.

15.1. Utilização Básica

A utilização básica, basta passar a ordenação e o DAO.

Exemplo - Recuperar ofertas ordenadas por data de cadastro (da maior para menor):

```
LazyDataModel ofertas = new LazyDataModelImpl<Oferta>("dataCadastro DESC", ofertaDAO);
```

15.2. Adicionar restrições na consulta do LazyDataModel

Para consultas mais complexas, pode-se utilizar **Restrictions** para filtrar a consulta.

Exemplo - O método abaixo retorna as ofertas de um determinado site, que contenha uma determinada descrição, ordenados por descrição.

```
public LazyDataModel<Oferta> getOfertas(Site site, String descricao) {  
    Restrictions restrictions = new Restrictions();  
    restrictions.add("site", site);  
    restrictions.add("descricao", RestrictionType.LIKE, descricao);  
  
    return new LazyDataModelImpl<Oferta>("descricao", restrictions, ofertaDAO);  
}
```

Mais informações sobre essas restrições podem ser encontradas na sessão [Restriction e Restrictions](#).

15.3. Filtros da coluna

O **LazyDataModelImpl** faz a filtragem no banco a partir do campo definido no atributo “filterBy” definido no componente “column”. Para isso ele utiliza o tipo de restrição *RestrictionType.DATA_TABLE_FILTER* seguindo a seguinte regra:

Tipo de Campo	Descrição
String	Utiliza consulta “like” (que contenha a String seja no inicio, meio ou fim).
Long, Integer, BigDecimal	Utiliza consulta “igual a”
Boolean	Utiliza consulta “igual a”. Para evitar que tenha que ser digitado “true” ou “false” pode ser utilizado o filterOptions com o “booleanSelectItensEmptyOptions”. Exemplo: filterOptions="#{booleanSelectItensEmptyOption}”.
Date e Calendar	Tenta consultar pela data convertida em String, a data deve seguir o padrão java “SimpleDateFormat.MEDIUM” do Locale Atual. E caso sejam passados 2 valores separados por hífen “-” é feita uma consulta de intervalos e as 2 datas. Exemplo(pt_BR): “01/01/13 - 01/02/13” traria os registros entre essas 2 datas. O uso do componente DateFilter facilita essa filtragem.

15.4. Recuperar todos os registros de maneira não pagina

O LazyDataModelImpl, realiza a consulta por demanda, pois o *dataTable* do primefaces chama o método *load()* sempre que o *dataTable* é chamado.

Existem casos onde é necessário exibir na tela paginado, mas em um dado momento essa lista precisa ser recuperada por completo, para isso existe o método *getAllResults()*.

Supondo o seguinte *dataModel* para cidades:

```
LazyDataModelImpl<Cidade> cidadesLazy = new LazyDataModelImpl<Cidade>("descricao", cidadeDAO);
```

Para recuperar todos os registros (serão filtrados conforme os filtros do *dataTable*):

```
List<Cidade> cidades = cidadesLazy.getAllResults();
```

15.5. Manipulando a Ordenação com o OrderByHandler

Ao definir o **sortBy** no componente **p:column** podemos definir a ordenação da coluna com um atributo específico do objeto. Para manipular essa ordenação podemos setar o **OrderByHandler** no LazyDataModelImpl.

Como o OrderByHandler é uma interface, devemos instanciá-lo sobrescrevendo o método *getOrderBy(String orderBy)*, que retorna String, o código da interface é descrito abaixo:

```
public interface OrderByHandler {  
    public String getOrderBy(String orderBy);  
}
```

Para definir o OrderByHandler pode utilizar o método **setOrderByHandler**:

```
dataModel.setOrderByHandler(orderByHandler);
```

Exemplo de Utilização

O exemplo a seguir, quando a ordenação for “code”, a ordenação será manipulada para que fique “ORDER BY code, year”:

Código da coluna no data table:

```
<p:column headerText="Código" sortBy="#{ produto.codigo}" >  
    <h:outputText value="#{produto.codigo}" />  
</p:column>
```

Definindo o OrderByHandler para o LazyDataModel:

```

LazyDataModelImpl<Produto> produtos = new LazyDataModelImpl< Produto >("nome", produtoDAO);

//criando o handler
OrderByHandler orderByHandler = new OrderByHandler() {
    @Override
    public String getOrderBy(String orderBy) {
        if (orderBy != null && orderBy.equals("codigo")) {
            return "codigo, year";
        }
        return orderBy;
    }
};

//setando o handler no dataModel
produtos.setOrderByHandler(orderByHandler);

```

Dessa forma é possível tratar as mais diversas ordenações, e para cada campo fazer uma maneira personalizada.

Utilização no AbstractBaseBean

Para definir o **OrderByHandler** em um AbstractBaseBean podemos sobrescrever o método `getOrderByHandler()`, como mostrado abaixo:

```

@Override
public OrderByHandler getOrderByHandler() {
    return orderByHandler;
}

```

15.6. Manipulando as restrições com o FilterByHandler

Ao definir o **filterBy** no componente **p:column** podemos definir o filtro da coluna com um atributo específico do objeto. Para manipular esse filtro, podemos definir um **FilterByHandler** no LazyDataModel.

Como o FilterByHandler é uma interface, devemos instanciá-lo sobrescrevendo o método `getFilterBy(String property, Object value)`, que retorna o tipo **Restrictions** (lista de restrições a ser aplicada na consulta), o código da interface é descrito abaixo:

```

public interface FilterByHandler {

    public Restrictions getFilterBy(String property, Object value);

}

```

Para definir o FilterByHandler pode utilizar o método **setFilterByHandler**:

```

dataModel.setFilterByHandler(filterByHandler);

```

Exemplo de Utilização

Considere o seguinte cenário, a classe **Atividade.java** e nela temos o campo `dataPrevista`, para saber se a atividade está atrasado devemos comparar a data prevista com a data atual.

Atraso = dataPrevista < dataAtual

Classe **Atividade.java**:

```
@Entity
public class Atividade{

    @Id
    private Long id;
    @Temporal(TemporalType.DATE)
    private Date dataPrevista;
    @Transient
    private boolean apenasAtrasadas;

    public boolean isAtrasado() {
        if (dataPrevista != null) {
            return new Date().after(dataPrevista);
        }
        return false;
    }

    //getters and setters
}
```

- O método *isAtrasado* retorna *true* se a data atual for maior que a data prevista.
- O atributo *apenasAtrasadas* vai servir para manipular a consulta do handler.

Coluna com o campo “atrasado”:

```
<p:column headerText="Atrasada?" filterBy="#{atividade.apenasAtrasadas}"
           filterOptions="#{booleanSelectItensEmptyOption}" >
    <h:outputText value="#{atividade.atrasado}" converter="yesNoConverter"/>
</p:column>
```

Criando um método para retornar o handler:

```
public FilterByHandler getFilterByHandler() {

    FilterByHandler filterByHandler = new FilterByHandler() {
        @Override
        public Restrictions getFilterBy(String property, Object value) {

            if (property.equals("apenasAtrasados") && value != null && !value.toString().isEmpty()) {
                boolean atrasada = Boolean.valueOf(value.toString());
                Restrictions restrictions = new Restrictions();
                if (atrasada == true) {
                    //que data prevista seja menor que a data atual
                    restrictions.lessThan("dataPrevista", new Date(), TemporalType.DATE);
                } else {
                    //que data prevista seja maior ou igual a data atual
                    restrictions.greaterEqualsThan("dataPrevista", new Date(), TemporalType.DATE);
                }
                return restrictions;
            }
            return null;
        }
    };

    return filterByHandler;
}
```

Definindo o handler em uma instância do data model:

```
//criando o LazyDataModel
LazyDataModelImpl<Atividade> atividades = new LazyDataModelImpl<Atividade>("id", atividadeDAO);
//setando o handler
atividades.setFilterByHandler(getFilterByHandler());
```

Utilização no AbstractBaseBean

Para definir o **FilterByHandler** em um **AbstractBaseBean** podemos sobrescrever o método `getFilterByHandler()`, como mostrado abaixo:

```
@Override
public FilterByHandler getFilterByHandler() {
    return filterByHandler;
}
```

15.7. Definir se os dados serão ou não carregados

Caso exista alguma regra de negócio para definir se os dados serão ou não exibidos, podemos usar o método `setLoadData(boolean loadData)`.

Exemplo de Utilização

```
LazyDataModelImpl<Atividade> atividades = new LazyDataModelImpl<Atividade>("id", atividadeDAO);
atividades.setLoadData(false);
```

15.8. Definindo quando realizar o count com o LazyCountType

Por padrão um **LazyDataModel** faz um count para saber quantos registros existem no banco de dados. Dessa maneira é possível calcular a quantidade de páginas e quantidade total de registros.

Existem três tipos para ele, e são definidos através da enum *LazyCountType*, os possíveis valores são:

Tipo de Count	Descrição
LazyCountType.ALWAYS	Valor padrão. Sempre faz o count, ou seja, o count inicial, ao pagnar e ao se filtrar.
LazyCountType.NONE	Nunca realiza o count. Útil para grandes quantidades de registros, onde realizar um count, pode ser muito custoso.
LazyCountType.ONLY_ONCE	Realiza o count apenas quando a construir a tabela e quando acontecem mudança nos filtros de consulta.

Quando não definido tipo o padrão assumido é o *LazyCountType* .ALWAYS.

Ao se definir NONE, é necessário ter em mente que não é possível definir a quantidade de páginas que a tabela vai gerar, sendo assim é assumido o valor *Integer.MAX_VALUE*.

Exemplo de Utilização

```
LazyDataModelImpl<Atividade> atividades = new LazyDataModelImpl<Atividade>("id", atividadeDAO);
atividades.setLazyCountType (LazyCountType.ONLY_ONCE);
```

16. Definindo os joins com o JoinBuilder

16.1. Introdução

O **JoinBuilder** é uma classe que permite fazer a definição dos joins a serem utilizados em algumas partes do framework, como por exemplo o **LazyDataModelImpl**. Através dele podemos definir quais relacionamentos serão carregados através da utilização de “joins” e do “fetch” (segundo a especificação do JPA para JPQL).

16.2. Entendendo o problema dos joins

Ao se realizar um select em uma entidade que possui relacionamento, podemos enfrentar o problema do n+1 (onde para cada registro talvez seja necessário mais um select para carregar seu relacionamento).

Para descrever melhor, considere o seguinte cenário:

Classe **Pessoa.java**:

```
@Entity
public class Pessoa{

    @Id
    private Long id;
    private String nome;
    @ManyToOne
    private Empresa empresa;
}
```

Classe **Empresa.java**:

```
@Entity
public class Empresa{

    @Id
    private Long id;
    private String nome;
}
```

Veja que no relacionamento acima, pessoa tem uma empresa representada através do **@ManyToOne**.

Considerando o seguinte JPQL:

```
SELECT p FROM Pessoa p
```

Supondo que ele retornasse 5 registros distintos, e cada um com uma empresa diferente, nós teríamos gerado o seguinte sql:

```
select p.id, p.nome, p.empresa_id from pessoa p
```

E para carregar as empresas seria gerado algo parecido com os seguintes SQL (onde o prepared statement “?” seria o id da empresa recuperado do primeiro select):

```
select e.id, e.nome from empresa e where e.id = ?
select e.id, e.nome from empresa e where e.id = ?
select e.id, e.nome from empresa e where e.id = ?
select e.id, e.nome from empresa e where e.id = ?
select e.id, e.nome from empresa e where e.id = ?
```

Isso porque no exemplo são apenas 5 registros, imagine um cenário de 20, 30, 100...

Para que esse relacionamento com empresa já viesse carregado, poderíamos usar o “JOIN FETCH” no JPQL:

```
SELECT p FROM Pessoa p JOIN FETCH Empresa e
```

O resultado em SQL iria usar um join para já carregar o resulta da tabela empresa, resultando em algo parecido com:

```
select p.id, p.nome, p.empresa_id, e.id, e.nome from pessoa p inner join empresa e on p.empresa_id = e.id
```

Basicamente ao se usar o “join fetch” é indicado para o JPA que a consulta deve trazer o relacionamento já carregado.

16.3. Utilizando os joins no LazyDataModelImpl

O **LazyDataModelImpl** do xpert-framework é uma implementação do LazyDataModel do primefaces, a idéia é trazer os registros de maneira paginada. Como o LazyDataModelImpl utiliza consultas JQPL, o mesmo problema do n+1 pode acontecer.

Mas como adicionar join fetchs no LazyDataModelImpl?

Para indicar os joins, é possível usar o **JoinBuilder**, e assim é possível definir o comportamento dos joins no construtor do LazyDataModelImpl:

```
LazyDataModelImpl dataModel = new LazyDataModelImpl("nome", pessoaDAO, joinBuilder);
```

É possível ainda setar através do **setJoinBuilder**:

```
LazyDataModelImpl dataModel = new LazyDataModelImpl("nome", pessoaDAO);
dataModel.setJoinBuilder(joinBuilder);
```

Usando o exemplo anterior das classes **Pessoa.java** e **Empresa.java** poderíamos criar o join builder da seguinte maneira:

```
//criando o join builder onde "p" será o root alias da query
JoinBuilder joinBuilder = new JoinBuilder("p");
joinBuilder.leftJoinFetch("p.empresa", "e");

//criando o data model e setando o joinBuilder onde a ordenação padrão será "p.nome"
LazyDataModelImpl dataModel = new LazyDataModelImpl("p.nome", personDAO);
dataModel.setJoinBuilder(joinBuilder);
```

É importante lembrar que nesse caso foi definido o alias foi definido como “p”, então se for necessário adicionar Restrictions no data model, deve-se utilizar o alias, como mostrado abaixo:

```
//onde o nome é "like" Maria
Restrictions restrictions = new Restrictions();
restrictions.like("p.nome", "Maria");
```

É possível definir o tipo de join, para isso, podemos utilizar por exemplo os seguintes métodos:

Sem o Fetch

- leftJoin(String join, String alias)
- innerJoin(String join, String alias)
- rightJoin(String join, String alias)

Com o Fetch

- leftJoinFetch(String join, String alias)
- innerJoinFetch(String join, String alias)
- rightJoinFetch(String join, String alias)

16.4. Utilizando o JoinBuilder no AbstractBaseBean

O **AbstractBaseBean** é um Managed Bean genérico para criação de CRUD, por padrão ele possui um LazyDataModelImpl que será utilizando na listagem dos registros.

Podemos definir o JoinBuilder sobrescrevendo o método **getDataModelJoinBuilder()**, como mostrado abaixo:

```
@ManagedBean
@ViewScoped
public class PersonMB extends AbstractBaseBean<Person> {

    @EJB
    private PersonBO personBO;

    @Override
    public AbstractBusinessObject getBO() {
        return personBO;
    }
    @Override
    public String getDataModelOrder() {
        return "p.nome";
    }
    @Override
    public JoinBuilder getDataModelJoinBuilder() {
        return new JoinBuilder("p").innerJoinFetch("p.empresa", "e");
    }
}
```

17. Unicidade dos campos com UniqueField

Para verificar a unicidade dos campos, pode-se utilizar o objeto **UniqueField**. Com ele é possível definir um campo único, ou combinações de campos.

17.1. Definindo campos únicos em um BO genérico

Ao estender a classe **AbstractBusinessObject** (com.xpert.core.crud) é obrigatório implementar o método **getUniqueFields()**.

Exemplo de Utilização

BO sem validação de campos únicos:

```
@Override
public List<UniqueField> getUniqueFields() {
    return null;
}
```

BO onde a entidade não pode ter a descrição repetida:

```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add("descricao");
}
```

Pode ainda ser usado em cadeia (no exemplo o objeto não poderia possuir a mesma descrição, nem o mesmo código):

```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add("descricao").add("codigo");
}
```

Combinando mais de um campo, nesse exemplo o nome não pode se repetir combinado com a sigla, ou seja não pode existir esse objeto com essa mesma “descricao” e “sigla”.

```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add("descricao","sigla");
}
```

17.2. Customização de mensagem no UniqueField

As mensagens podem ser customizadas conforme indicado abaixo:

Utilização:

```
@Override
public List<UniqueField> getUniqueFields() {
    UniqueFields uniqueFields = new UniqueField();
    uniqueFields.add(new UniqueField("descricao").setMessage("Custom Message"));
    return uniqueFields;
}
```

O código acima também pode ser utilizado assim:


```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add(new UniqueField("descricao"), "Custom Message");
}
```

17.3. Validação fora de um BO Genérico

Exemplo - Não pode existir pessoa com CPF duplicado:

```
import com.xpert.core.exception.UniqueFieldException;
import com.xpert.core.validation.UniqueFieldsValidation;
import com.xpert.faces.utils.FacesMessageUtils;

public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add("cpf");
}

public void save(Pessoa pessoa){
    try {
        UniqueFieldsValidation.validateUniqueFields(getUniqueFields(), pessoa, pessoaDAO);
    } catch (UniqueFieldException ex) {
        FacesMessageUtils.error(ex);
    }
}
```

com.xpert.core.validation.UniqueFieldsValidation
xpert-framework
Classe para realizar validação dos campos únicos.

validateUniqueFields()
xpert-framework
recebe como parâmetro, uma lista de campos
únicos, o objeto a ser validado, e o dao. Este
método lança a exceção **UniqueFieldException**

UniqueFieldException é um **BusinessException**, sendo assim, ao ser capturado ele pode ser exibido formatado como **FacesMessageUtils**, como mostrado no exemplo acima.

17.4. Utilizando Restrictions com o UniqueFields

É possível combinar as restrictions diretamente no *UniqueField*, isso possibilita mais opções para restringir a validação.

Para isso existe um construtor tanto da classe *UniqueField*, como a *UniqueFields* (que é a lista) que recebe como parâmetro as *Restrictions*.

Adicionando o campo único “nome”, apenas os registros ativos:

```
UniqueFields uniqueFields = new UniqueFields(Restriction.equals("ativo", true), "nome");
```

Adicionando o campo único “nome”, apenas os registros que não possuem “data de exclusão”:

```
UniqueFields uniqueFields = new UniqueFields(Restriction.isNull("dataExclusao"), "nome");
```

18. Criação de relatórios com o FacesJasper

Para criar relatórios **jasperreports** pode-se utilizar a classe FacesJasper (com.xpert.faces.utils).

Exemplo de criação de um relatório:

```
FacesJasper.createJasperReport(dataSource, parameters, "WEB-INF/reports/report.jasper", "report.pdf");
```

A geração de relatório gera um download de arquivo, então ela pode ser combinada com o componente **download**, isso causa uma melhor experiência ao usuário, pois este componente bloqueia a tela enquanto o download é gerado:

Exemplo:

```
<p:commandButton value="Create Report" action="#{reportMB.generateReport}" ajax="false">
  <x:download/>
</p:commandButton>
```

Para componentes ajax, com **commandLink** e **commandButton** do primefaces, é necessário especificar **ajax="false"**, visto que um download não pode ser ajax.

19. Geração de Código (CRUD) com o Xpert-Maker

19.1. Introdução

A geração de código pode ser feita de 2 maneiras a primeira delas é via Swing (disponibilizado a partir da versão 1.3), onde o código é gerado diretamente no projeto do desenvolvedor e a segunda é via Componente JSF onde é disponibilizada uma interface para que o código seja gerado, por ser web as classes não são salvas diretamente na máquina do desenvolvedor, em vez disso é exibido o código na tela e disponibilizado o download do código-fonte compactado.

19.2. Integração com o framework bootstrap

A partir da versão 1.8, o xpert-maker passou a ser integrado com o framework bootstrap, dando assim a possibilidade de gerar código com layouts responsivos.

A utilização do bootstrap é totalmente opcional, e o xpert-framework não obriga sua utilização, ele apenas facilita na geração de CRUDs que irão utilizar o bootstrap para gerar o layout responsivo.

19.2.1. Inserindo o bootstrap no seu projeto

O bootstrap pode ser baixado diretamente no site <http://getbootstrap.com/>. Atualmente o xpert-framework gera código baseado no bootstrap 3.

Ao baixar o arquivo adicione o CSS e o javascript no seu projeto. Os arquivos principais do bootstrap são:

- **bootstrap.css** ou **bootstrap.min.css** (versão minificada)
- **bootstrap.js** ou **bootstrap.min.js** (versão minificada)

Adicione os arquivos no seu projeto, como no exemplo abaixo:

```
<link rel="stylesheet" media="screen" href="#{request.contextPath}/css/bootstrap.min.css"/>
<script src="#{request.contextPath}/js/bootstrap.min.js"></script>
```

19.2.2. Classes CSS utilitárias do xpert-framework

Para facilitar o desenvolvimento algumas classes CSS foram criadas para melhor integração do xpert-framework com o bootstrap

Nome da classe CSS	Descrição
uix-calendar	Para o component p:calendar, esse CSS adiciona algumas regras para melhor visualização do calendar responsivo
uix-dialog-fluid	Para o componente p:dialog, esse CSS deixa o tamanho do dialog dinâmico (em percentual do tamanho da tela)
uix-content-detail	Para o detalhamento, quando o campo possuir valor em branco, essa classe exibe um "espaço em branco" para melhor ajuste do conteúdo.

19.2.3. Estrutura de geração dos formulários usando o bootstrap

Ao gerar o CRUD com o bootstrap usa o xpert-framework usa conceito de dividir a tela em 12 colunas, utilizando as classes do próprio bootstrap.

Para montar o formulário de cadastro, são usadas as classes do bootstrap para formulários, como a *form-group* e *form-control*.

Considerando a classe abaixo, **Pessoa.java**, veja os exemplos de forms gerados:

```
@Entity
public class PessoaExemplo implements Serializable {

    @Id
    private Long id;

    @Size(max = 200)
    @NotBlank
    private String nome;

    @Size(max = 100)
    @Email
    private String email;

    private BigDecimal salary;
}
```

Exemplo de form para cadastro gerado

Note a presença da classe *form-group*, circundando o *label* e o *input*.

```
<div class="container-fluid">
  <div class="row">
    <div class="form-group col-lg-4 col-md-6 col-sm-6 col-xs-12">
      <h:outputLabel for="nome" value="* #{msg['pessoa.nome']}:" />
      <p:inputText id="nome" value="#{pessoaMB.entity.nome}"
        maxlength="200" styleClass="form-control" />
    </div>
    <div class="form-group col-lg-4 col-md-6 col-sm-6 col-xs-12">
      <h:outputLabel for="email" value="#{msg['pessoa.email']}:" />
      <p:inputText id="email" value="#{pessoaMB.entity.email}"
        maxlength="100" styleClass="form-control" />
    </div>
    <div class="form-group col-lg-4 col-md-6 col-sm-6 col-xs-12">
      <h:outputLabel for="salary:input" value="#{msg['pessoa.salary']}:" />
      <x:inputNumber id="salary" value="#{pessoaMB.entity.salary}"
        styleClass="form-control"/>
    </div>
  </div>
</div>
```

Exemplo de form para detalhamento gerado

Nesse caso é usada a classe *control-label* para os labels e a classe utilitário do xpert-framework *uix-content-detail* para os valores.

```

<div class="container-fluid">
  <div class="row">
    <div class="col-lg-4 col-md-6 col-sm-6 col-xs-12">
      <h:outputLabel value="#{msg['pessoa.nome']}:" styleClass="control-label" /><br/>
      <h:outputText value="#{pessoaMB.entity.nome}" styleClass="uix-content-detail"/>
    </div>
    <div class="col-lg-4 col-md-6 col-sm-6 col-xs-12">
      <h:outputLabel value="#{msg['pessoa.email']}:" styleClass="control-label" /><br/>
      <h:outputText value="#{pessoaMB.entity.email}" styleClass="uix-content-detail"/>
    </div>
    <div class="col-lg-4 col-md-6 col-sm-6 col-xs-12">
      <h:outputLabel value="#{msg['pessoa.salary']}:" styleClass="control-label" /><br/>
      <h:outputText value="#{pessoaMB.entity.salary}" styleClass="uix-content-detail">
        <f:convertNumber minFractionDigits="2" maxFractionDigits="2" />
      </h:outputText>
    </div>
  </div>
</div>

```

19.3. Utilização Básica via componente Swing

A partir da versão 1.3 do xpert-framework foi criado uma aplicação swing para geração de código, a vantagem é que o código é gerado diretamente no projeto do desenvolvedor.

19.3.1. Estrutura da classe para geração via componente Swing

Primeiramente deve ser criada uma classe geradora no projeto que possui as classes modelo. Essa classe deve ser filha de **MakerSwingFrame** (pacote com.xpert.maker) sendo que ela é uma classe abstrata onde alguns métodos devem ser sobrescritos.

```

public class Maker extends MakerSwingFrame {

    @Override
    public String getDefaultPackage() {
        return "com.xpert.showcase.model";
    }

    @Override
    public String getDefaultTemplatePath() {
        return "/template/mainTemplate.xhtml";
    }

    @Override
    public String getDefaultResourceBundle() {
        return "msg";
    }

    @Override
    public String getDefaultBaseDAOImpl() {
        return "com.xpert.showcase.application.BaseDAOImpl";
    }

    @Override
    public String getManagedBeanSuffix() {
        return "MB";
    }

    @Override
    public String getBusinessObjectSuffix () {
        return "BO";
    }

    @Override
    public PrimeFacesVersion getPrimeFacesVersion() {
        return PrimeFacesVersion.VERSION_4;
    }

    @Override
    public BootstrapVersion getBootstrapVersion () {
        return BootstrapVersion.VERSION_3;
    }

    public static void main(String args[]) {
        run(new Maker());
    }
}

```

getDefaultPackage()

xpert-framework

Define o pacote padrão de classes modelo.

getDefaultTemplatePath()

xpert-framework

Define o template facelets a ser utilizado nas views geradas.

getDefaultResourceBundle()

xpert-framework

Define o resource bundle a ser utilizado para labels das view geradas. Exemplo: informando o bundle "msg" as views serão geradas no padrão #{msg['person.name']}

getDefaultBaseDAOImpl()

xpert-framework

Define DAO genérico a ser utilizado nos DAOs gerados

getManagedBeanSuffix()

xpert-framework

Define o sufixo do nome da classe do Managed Bean, por padrão este é "MB". Exemplo: PersonMB.

getBusinessObjectSuffix()

xpert-framework

Define o sufixo do nome da classe do Business Object, por padrão este é "BO". Exemplo: PersonBO.

getPrimeFacesVersion()

xpert-framework

Define a versão do PrimeFaces a ser utilizada na geração.

main()

Método para chamar

getBootstrapVersion()

xpert-framework

Define a versão do Bootstrap a ser utilizada na geração. Caso não utilize o bootstrap, basta retornar null.

19.3.2. Passo a passo da geração de classes via componente Swing

Após a classe **Maker.java** mostrada acima ser executada o resultado é o seguinte:

The screenshot shows a Swing application window titled "Xpert-framework Maker" with three tabs: "Select Classes", "Project Configuration", and "Create Classes". The "Create Classes" tab is active, displaying a "Class Selection" panel. The panel has a header "1 Choose your java model classes, these classes must be JPA entities". Below this, there is a "Package Name:" label and a text input field containing "com.xpert.showcase.model". To the right of the input field is a "Search Classes" button. Below the input field, it says "Classes found in package:" followed by a large empty rectangular box. At the bottom left of this box, there is a "Select:" label and two buttons: "All" and "None". At the bottom right of the panel is a "Next" button. On the left side of the window, there is a sidebar with the application's name and description in English and Portuguese, along with links to the showcase, documentation, and the company website.

O campo "Package name" é utilizado para listar as classe do pacote informado, por padrão ele vem preenchido com o valor informado no método **getDefaultPackage()**.

Para buscar as classes clique no botão **“Search Classes”**, o resultado será o seguinte:

Select Classes | **Project Configuration** | **Create Classes**

1 Choose your java model classes, these classes must be JPA entities

Class Selection

Package Name:
com.xpert.showcase.model

Search Classes

Classes found in package:

- class com.xpert.showcase.model.Car
- class com.xpert.showcase.model.City
- class com.xpert.showcase.model.Country
- class com.xpert.showcase.model.Group
- class com.xpert.showcase.model.Permission
- class com.xpert.showcase.model.Person
- class com.xpert.showcase.model.State
- class com.xpert.showcase.model.StatusGroup
- class com.xpert.showcase.model.Toy
- class com.xpert.showcase.model.UserRole

Select: **All** **None**

Next

Seleção de classes
-
Selecione aqui as classes que serão gerados os CRUDs (Ctrl+click para selecionar mais de uma)

Search Classes
-
Clique aqui para buscar as classes do seu projeto

Select All
-
Selecione todas as classes encontradas na busca

Select None
-
Remover as classes selecionadas

Next
-
Avançar para o próximo passo

Selecione as classes que você deseja gerar, para selecionar todos clique em **“Select All”** e para remover a seleção clique em **“Select None”**.

Vá para a próxima aba **“Project Configuration”**.

Nessa aba deve-se informar os caminhos das classes, páginas e algumas configurações, como mostrado abaixo:

2 Configure the path where you to generate the CRUD java classes

Java Classes

Business Object Location: ... Package:

DAO (Interface) Location: ... Package:

DAO (Implementation) Location: ... Package:

Managed Bean Location: ... Package:

3 Configure the path to generate the JSF pages (XHTML). Choose the primefaces version and bootstrap (if you want a responsive layout)

View Configuration

XHTML Location: ... Facelets Template: PrimeFaces: Version 3 Bootstrap: Don't use

4 Other configurations. Do you use CDI? Need access control?

Others

BaseDAOImpl: Resource Bundle: Author: Managed Bean Suffix: Business Object Suffix:

Date pattern: Time pattern: ☒ Mask calendar (Primefaces < 5.x, don't put mask on p:calendar)

☐ Use CDI Beans (@Named instead of @ManagedBean) ☒ Generate securityArea (use access control in forms)

Back Next

Note que o painel **“Others”** dessa aba já vem preenchido com os métodos que foram sobrescritos na classe, como o caminho dos XHTML, a versão do PrimeFaces e outros.

Nessa aba você deve informar o caminho das suas classes, por exemplo informar onde vai ficar o Managed Bean, DAO, DAOImpl, BO e onde suas views serão geradas (campo **“Location”**).

Ainda nessa aba também informe o padrão de pacotes para suas classes (campo **“Package”**).

Abaixo como esta aba ficaria preenchida:

Selecionar Pasta
 -
 Clique aqui para selecionar a pasta da classe a ser gerada

Select Classes
Project Configuration
Create Classes

2 Configure the path where you to generate the CRUD java classes

Java Classes

Business Object Location:

Package:

DAO (Interface) Location:

Package:

DAO (Implementation) Location:

Package:

Managed Bean Location:

Package:

3 Configure the path to generate the JSF pages (XHTML). Choose the primefaces version and bootstrap (if you want a responsive layout)

View Configuration

XHTML Location:

Facelets Template:

PrimeFaces:

Bootstrap:

Other configurations. Do you use CDI? Need access control?

DAOImpl:

Resource Bundle:

Author:

Managed Bean Suffix:

Business Object:

Date pattern:

Time pattern:

☒ Mask calendar (Primefaces < 5.x, don't put mask on p:calendar)

☐ Use CDI Beans (@Named instead of @ManagedBean)

☒ Generate securityArea (use access control in forms)

Back

Next

Back
-
Voltar para o passo anterior

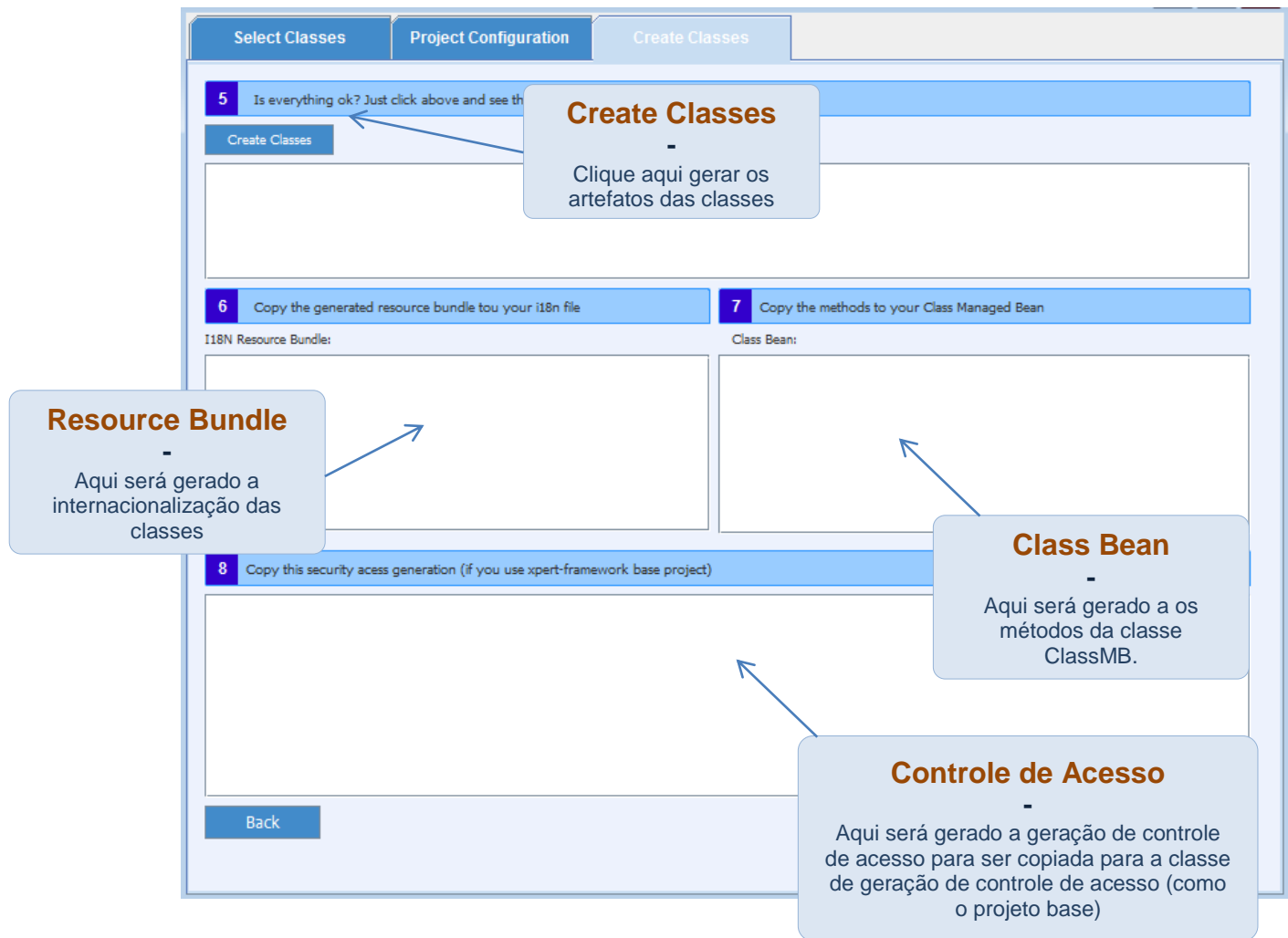
Next
-
Avançar para o próximo passo

Ainda nesta aba os seguintes parâmetros podem ser configurados:

Nome do campo	Valor Padrão	Descrição
Primefaces	Definida ao extender o MakerSwingFrame	Define a versão do primefaces do projeto
Bootstrap	null	Define a versão do bootstrap a ser utilizada
Date pattern	dd/MM/yyyy	Padrão de data para ser utilizado no "p:calendar" (gerado para os tipos <i>date</i>).
Time pattern	HH:mm	Padrão de data para ser utilizado no "p:calendar" (gerado para os tipos <i>time</i>).
Mask Calendar	true	Indica se a o calendar vai utilizar máscara. Versões anteriores a 5.x do primefaces não inserem máscara no p:calendar
Use CDI beans	false	Indica se no lugar de @ManagedBean, deve-se utilizar o @Named do CDI. O @ViewScope padrão do CDI só existe a partir do JSF 2.2
Generate securityArea	true	Indica se o CRUD gerado possui o x:securityArea para colocar o controle de acesso.

O passo seguinte é gerar as classes, vá para a aba **"Create Classes"**.

Nessa aba você pode realizar a geração das classes, e acompanhar o log de geração, como mostrado abaixo:

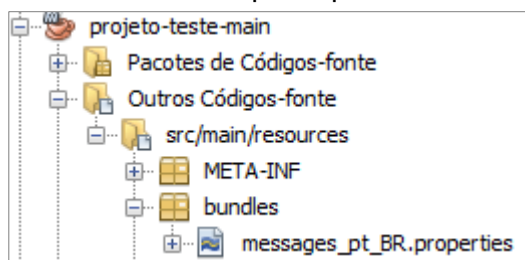


Clique em **“Create Classes”** e as classes serão escritas no seu projeto. O log de geração pode ser acompanhado no campo **“Log”**. As mensagens de internacionalização podem ser pegadas no campo **“I18N Resource Bundle”** e o Managed Bean com as classes pode ser pegado na em **“Class Bean”**.

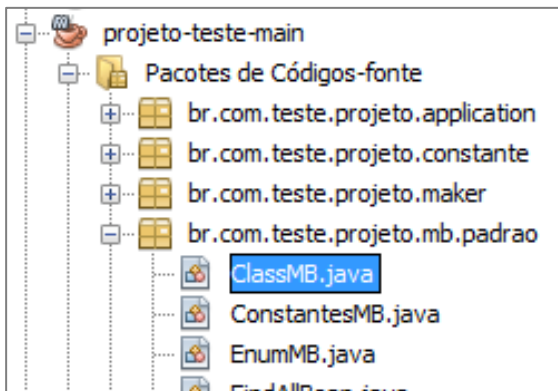
Caso você utilize uma classe para geração de permissão, pode copiar o que foi gerado na parte de controle de acesso. Esse código é gerado baseado no projeto base do xpert-framework.

No projeto do arquétipo (caso esteja utilizando o arquétipo):

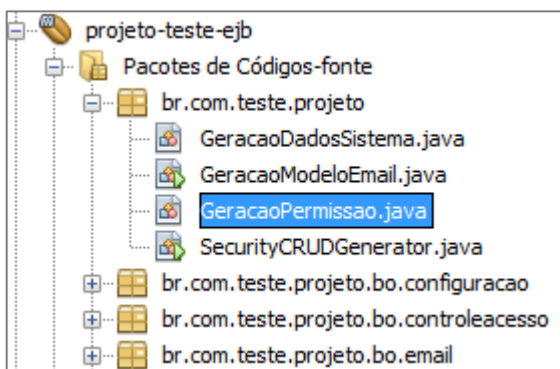
1. O i18n deve ser copiado para o **bundles/messages_pt_BR.properties** (módulo main):



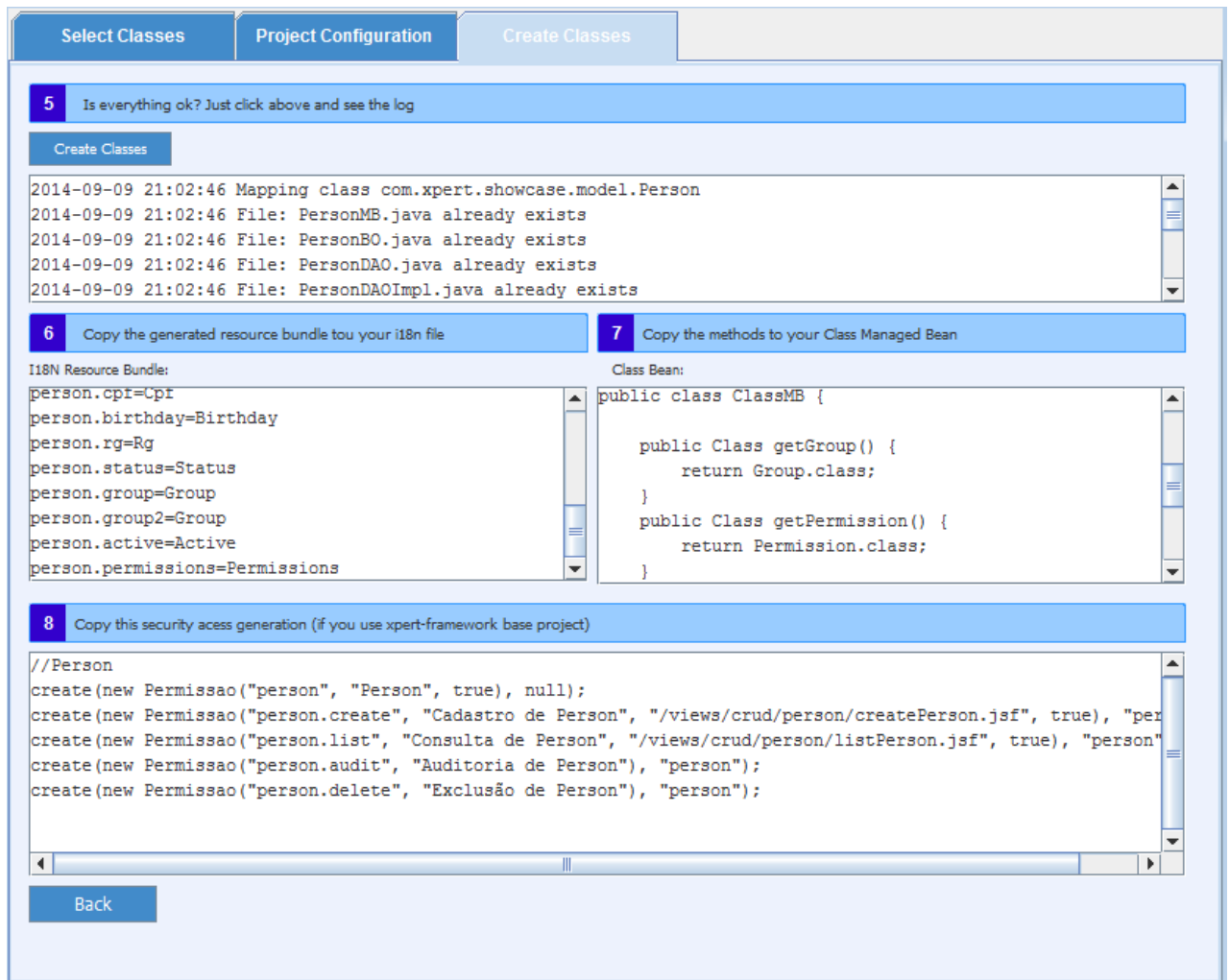
2. Os métodos do Class Bean devem ser copiados para a classe **ClassMB.java** (módulo main), é importante verificar se o método já existe na classe assim evita erro de compilação na classe:



3. A parte de controle de acesso deve ser copiada para a classe **GeracaoPermissao.java** (módulo EJB):



Abaixo o resultado de uma geração com sucesso:



19.4. Utilização Básica via componente JSF

A geração de código é feita baseada em uma entidade já existente.

Para utilizar o componente xpert-maker é necessário criar um **EntityManagerFactory** e defini-lo no arquivo **xpert-config.xml** para que o framework encontre o entity manager da aplicação.

O componente de geração de código chama-se **maker** possui a seguinte estrutura:

Chamada do componente:

```
<x:maker managedBean="com.xpert.showcase.mb.crud"
businessObject="com.xpert.showcase.bo"
dao="com.xpert.showcase.dao"
daoImpl="com.xpert.showcase.dao.impl"
baseDAO="com.xpert.showcase.dao.BaseDAOImpl"
resourceBundle="msg"
template="/template/mainTemplate.xhtml"
viewPath="/view"
author="Author" />
```

Os atributos do componente maker são:

Atributo	Descrição
managedBean	Nome padrão dos pacotes dos <i>ManagedBeans</i> gerados.
businessObject	Nome padrão dos pacotes dos <i>BusinessObjects</i> (BOs) gerados.
dao	Nome padrão dos pacotes dos DAOs (interface) gerados.
daoImpl	Nome padrão dos pacotes da implementação dos DAOs gerados.
baseDAO	Nome completo da classe padrão a ser utilizada como implementação do BaseDAO do xpert-framework.
resourceBundle	Nome do Resource Bundle a ser usado para acessar o arquivo de internacionalização.
template	Caminho do template a ser utilizados nos XHTML gerados.
viewPath	Caminho padrão das páginas geradas
author	Autoria da geração de código
managedBeanSuffix	Sufixo a ser utilizado no Managed Bean. Padrão é “MB”, exemplo: <i>PessoaMB</i>
businessObjectSuffix	Sufixo a ser utilizado no Business Object. Padrão é “BO”, exemplo: <i>PessoaBO</i>
useCDIBeans	Indica se no lugar de <i>@ManagedBean</i> , deve-se utilizar o <i>@Named</i> do CDI. O <i>@ViewScope</i> padrão do CDI só existe a partir do JSF 2.2
datePattern	Padrão de data para ser utilizado no “p:calendar” (gerado para os tipos <i>date</i>).
timePattern	Padrão de data para ser utilizado no “p:calendar” (gerado para os tipos <i>time</i>).
maskCalendar	Indica se a o calendar vai utilizar máscara. Versões anteriores a 5.x do primefaces não inserem máscara no p:calendar
timePattern	Indica se no lugar de <i>@ManagedBean</i> , deve-se utilizar o <i>@Named</i> do CDI. O <i>@ViewScope</i> padrão do CDI só existe a partir do JSF 2.2
generatesSecurityArea	Indica se o CRUD gerado possui o x:securityArea para colocar o controle de acesso.

Após inserido na tela o maker é renderizado da seguinte maneira:

A primeira aba (**Configuration**) é usada para configurações do projeto:

1 - Configuration

2 - Create Classes

Package Configuration

Insert the packages name

Managed Bean:

com.xpert.showcase.mb.crud

DAO:

com.xpert.showcase.dao

DAO Impl:

com.xpert.showcase.dao.impl

Business Object:

com.xpert.showcase.bo

Others

Primefaces Version:

☒ Version 3
☐ Version 4
☐ Version 5

Bootstrap Version:

☐ Don't use
☐ Version 3

Only if your project uses bootstrap CSS

BaseDAOImpl Class:

com.xpert.showcase.application.BaseDAOImpl

The BaseDAO class of your Project
(Must extend com.xpert.persistence.dao.BaseDAOImpl)

Faces Resource Bundle:

msg

(Default is 'msg', example: #{msg['person.name']})

Template (XHTML):

/template/mainTemplate.xhtml

(Default is 'template/mainTemplate.xhtml')

View Path:

(Default is 'view/', example: 'view/person/createPerson.jsf')

Author:

Managed Bean Suffix:

(Default is 'MB', example: 'PersonMB.java')

Business Object Suffix:

(Default is 'BO', example: 'PersonBO.java')

Date Pattern:

(Date pattern to be used in p:calendar)

Time Pattern:

(Time pattern to be used in p:calendar)

Mask Calendar:

☒ Yes
☐ No

(Primefaces < 5.x, don't put mask on p:calendar)

Use CDI Beans:

☐ Yes
☒ No

(@Named instead of @ManagedBean)

Generate securityArea:

☒ Yes
☐ No

(Generate securityArea use access control in forms)

1 - Configuration

-

Aba para definição de configurações do projeto.

Package Configuration

-

Configuração de nomes dos pacotes do projeto.

Others

-

Outras configurações do projeto

Com a configuração do projeto realizada, basta ir para a segunda aba (**Create Classes**):

2 – Create Classes
-
Aba para geração dos artefatos do CRUD.

Mapped Entities
-
Lista de entidades mapeadas.

Visualização dos Artefatos
-
Nesta área os artefatos gerados

Make Selected
-
Gerar artefatos para as classes selecionadas.

Make All
-
Gerar artefatos para todas as classes listadas.

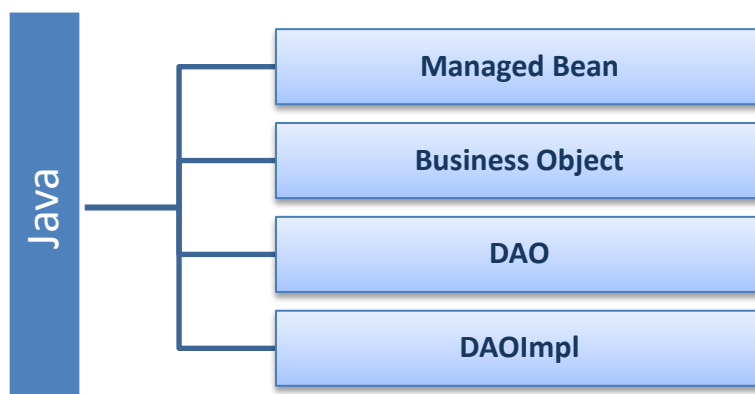
Reset
-
Limpar registros selecionados e artefatos já gerados.

Clicando em **Make Selected** ou em **Make All** será exibido o resultado da geração de código, os artefatos podem ser baixados através do botão **Download**:

Download
-
Baixa os artefatos gerados em um arquivo compactado (zip).

19.5. Estrutura de um CRUD gerado

19.5.1. Artefatos criados para cada Entidade



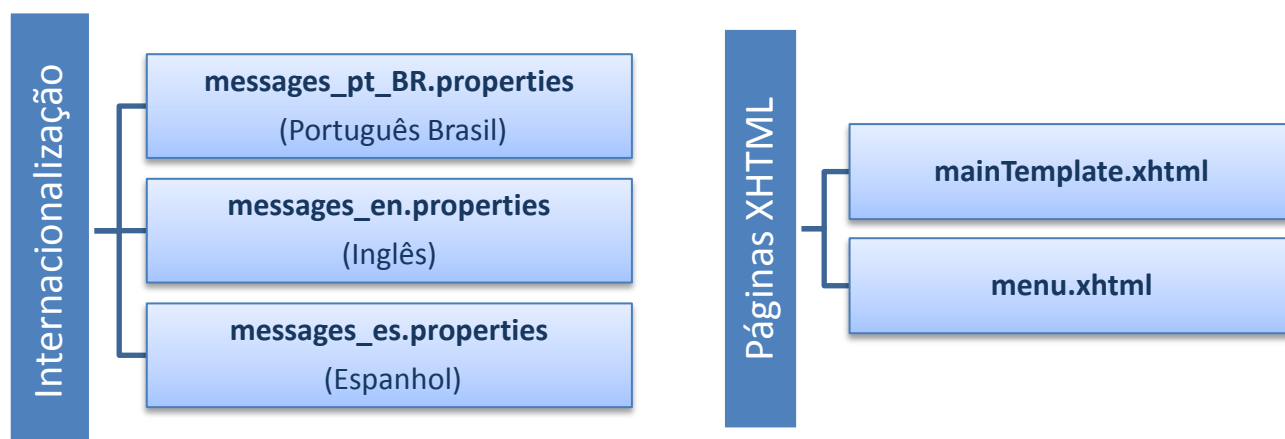
Classe	Descrição
Managed Bean	Managed Bean do JSF seguindo a especificação, as classes geradas recebem a anotação @ManagedBean (@Named para o CDI) e o escopo @ViewScoped . O padrão da classe pode ser visto na sessão Estrutura de um ManagedBean .
Business Object	Classe de regra de negócio. Esta classe é um session bean da especificação do EJB, recebe a anotação @Stateless . O padrão da classe pode ser vista em Estrutura de um Business Object .
DAO	Interface que segue o padrão Data Access Object .
DAO Impl	Implementação do DAO.



XHTML	Descrição
list	Página de listagem da entidade, nela é possível: detalhar, excluir, auditar exclusões e detalhar. Esta página recebe o include da view detail e do menu .
detail	Detalhamento da entidade, nele é possível ainda auditar da entidade.
create	Página de criação da entidade. Esta página recebe include do formCreate e do menu .
formCreate	Form para criação da entidade. Esta página é separada do create pois assim fica mais dinâmica e pode ser utilizada em forma da includes em outras páginas.
menu	Possui os links de acesso às páginas de create e list

19.5.2. Artefatos únicos

Esses são artefatos onde geralmente só se existe um por projeto. Eles só são gerados a partir do componente JSF não sendo possível a geração através do componente Swing.

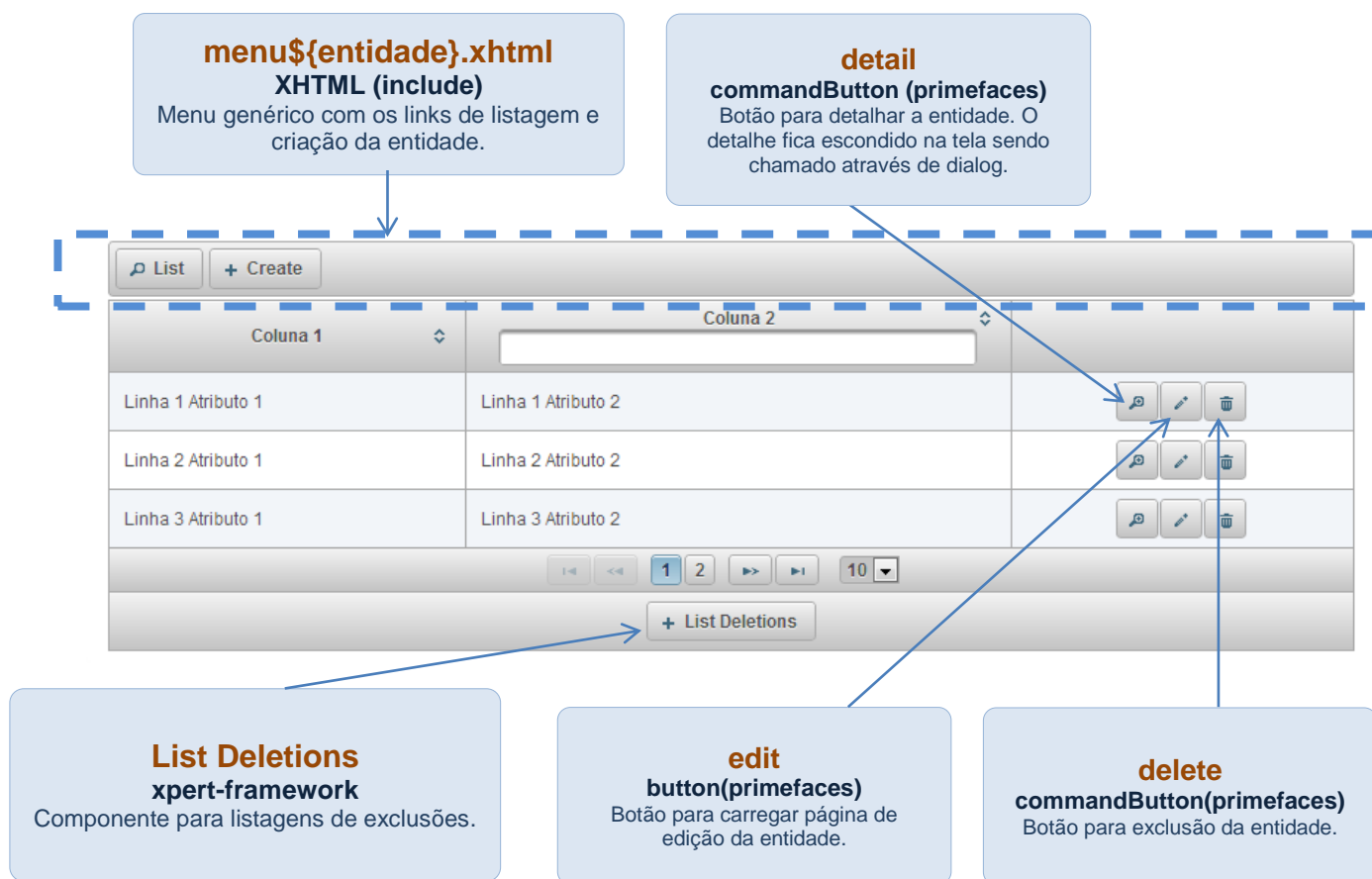


Artefato	Descrição
messages	Arquivo de internacionalização.
mainTemplate.xhtml	Template genérico para o projeto.
menu.xhtml	Possui os links de acessos aos CRUDs gerados.

19.5.3. Padrão da view para listagem de Registros - list{entidade}.xhtml

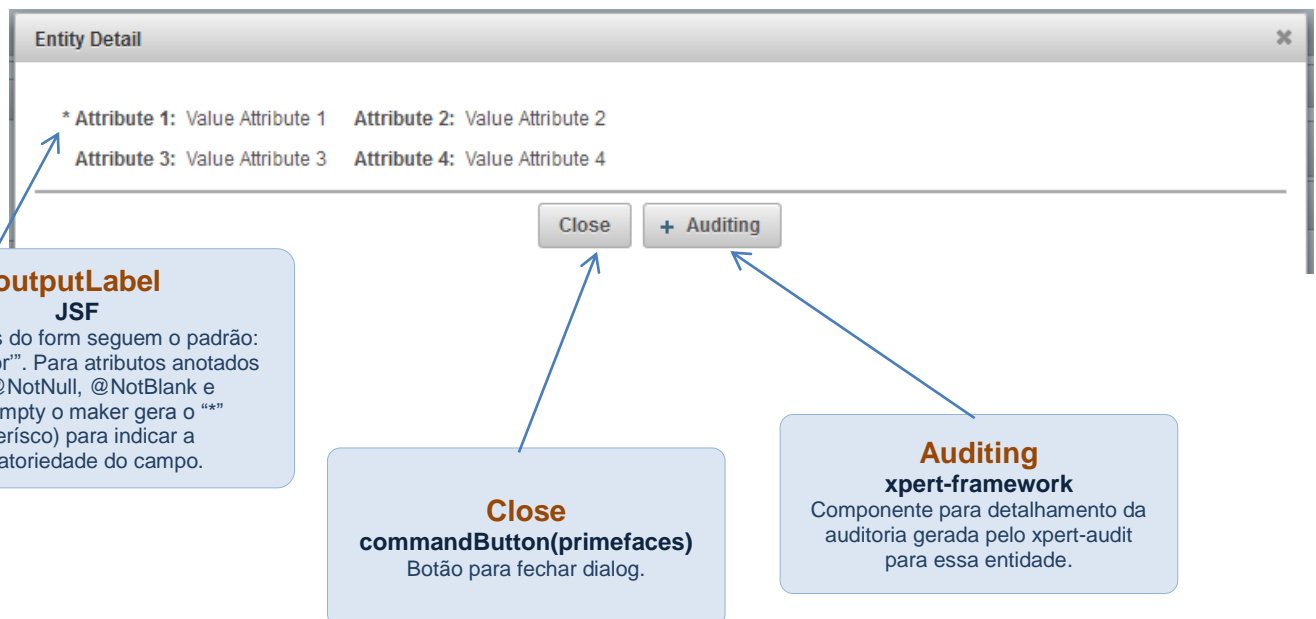
A view de listagem consiste em um **dataTable** (LazyDataModel paginado no banco), sendo que para cada registro se tem a opção de **detalhar**, **editar** e **excluir**. Essa tabela ainda a opção de listar as exclusões (que foram capturadas a partir do xpert-audit).

As colunas são geradas dinamicamente a partir dos atributos da entidade, ou seja, para cada atributo existe uma coluna (com exceção os tipos de listas (List, Collection, etc...)).



19.5.4. Padrão da view para detalhamento do registro – detail{entidade}.xhtml

A view de detalhamento é usada na forma de include (sendo exibida dentro de um dialog) para a listagem. Basicamente ele lista os atributos da classe em forma de texto. O detalhamento ainda gera o componente “Auditing” para detalhe da auditoria da entidade.



19.5.5. Padrão da view para criação e edição do registro – create{entidade}.xhtml

O XHTML gerado para a criação de views consiste basicamente em includes do **menu** e do **formCreate**. Os campos em si estão concentrados no xhtml **formCreate{entidade}.xhtml**. Essa separação possibilita futuras reutilizações na forma de include para o form.

outputLabel JSF

Os campos do form seguem o padrão: "label: 'valor'". Para atributos anotados com @NotNull, @NotBlank e @NotEmpty o maker gera o "*" (asterisco) para indicar a obrigatoriedade do campo.

menu\${entidade}.xhtml XHTML (include)

Menu genérico com os links de listagem e criação da entidade.

formCreate\${entidade}.xhtml XHTML (include)

Form com os campos da entidade.

The diagram illustrates a web form layout with several components and annotations:

- Navigation Bar:** A horizontal bar at the top containing a "List" button with a magnifying glass icon and a "+ Create" button.
- Form Fields:** Below the navigation bar, there are three input fields:
 - * Code: A text input field.
 - * Name: A text input field.
 - * Group: A dropdown menu with "Select" as the current selection.
- Permissions:** A row of checkboxes labeled "Permissions:" followed by "CREATE", "DELETE", "DELETE CASCADE", "SEARCH", and "UPDATE".
- Footer:** A horizontal line followed by the text "(*) Campos Obrigatórios".
- Save Button:** A "Save" button located at the bottom right of the form area.

Annotations with arrows point to specific parts of the form:

- A blue arrow points from the "outputLabel JSF" box to the asterisk (*) next to the "Code" field label.
- A blue arrow points from the "menu\${entidade}.xhtml XHTML (include)" box to the "+ Create" button.
- A red arrow points from the "formCreate\${entidade}.xhtml XHTML (include)" box to the "Code" input field.
- A blue arrow points from the "Save commandButton (primefaces)" box to the "Save" button.

19.6. Atributos mapeados e seus respectivos componentes na View

Cada tipo de atributo possui um componente correspondente gerado na tela através do maker, por exemplo uma **String** corresponde a um `inputText` (`p:inputText`). Abaixo segue a lista completa por tipo de atributo.

19.6.1.1. Atributos java

Tipo de Atributo	Componente Renderizado	Tipo de Componente
String	<code>p:inputText</code>	Primefaces
boolean	<code>h:selectBooleanCheckbox</code>	JSF
BigDecimal e Double	<code>x:inputNumber</code>	Xpert-framework
Integer e Long	<code>p:inputMask</code> (mask = 9?999999999)	Primefaces
Date e Calendar	<code>p:calendar</code>	Primefaces

19.6.1.2. Atributos de relacionamentos JPA

Tipo de Atributo	Componente Renderizado	Tipo de Componente
@ManyToOne	<code>h:selectOneMenu</code>	JSF
@ManyToMany	<code>h:selectManyCheckbox</code>	JSF

Observações:

- Para relacionamentos que estejam LAZY, o xpert-maker insere o componente **x:initializer**.
- Para pegar o objeto por completo os componentes **h:selectOneMenu** e **h:selectManyCheckbox** exigem um conversor, para esses casos o xpert-maker insere o conversor genérico **entityConverter**.
- Para campos boolean na exibição na listagem e no detalhamento é inserido o conversor **yesNoConverter** para exibição de Sim (true) e Não (false) na view.

19.7. Removendo o campo “id” da URL ao editar o Objeto

Por padrão o CRUD gerado pelo xpert-framework, insere o id na URL para carregar o objeto, esse id é enviado da tela de listagem para a tela de cadastro. Em determinados casos, é necessário esconder esse campo, ou ainda não permitir que seja digitado o id na URL.

O botão padrão para edição é o `p:button`, que gera uma requisição GET, ele é gerado como mostrado abaixo:

```
<p:button icon="ui-icon-pencil" outcome="createPessoaExemplo" title="#{xmsg['edit']}">
  <f:param name="id" value="#{pessoaExemplo.id}" />
</p:button>
```

Para mudar que o id não seja passado na URL é necessário definir os atributos “entity” e “outcome”, que indicam a entidade e a página de destino respectivamente, e para que seja gerado uma requisição POST deve-se usar um *p:commandButton* (ajax=false), como mostrado abaixo:

```
<p:commandButton icon="ui-icon-pencil" action="#{pessoaExemploMB.putEntityInRequest}"
                 ajax="false" title="{xmsg['edit']}" >
  <f:setPropertyActionListener value="#{pessoaExemplo}" target="#{pessoaExemploMB.entity}" />
  <f:setPropertyActionListener value="createPessoaExemplo" target="#{pessoaExemploMB.outcome}" />
</p:commandButton>
```

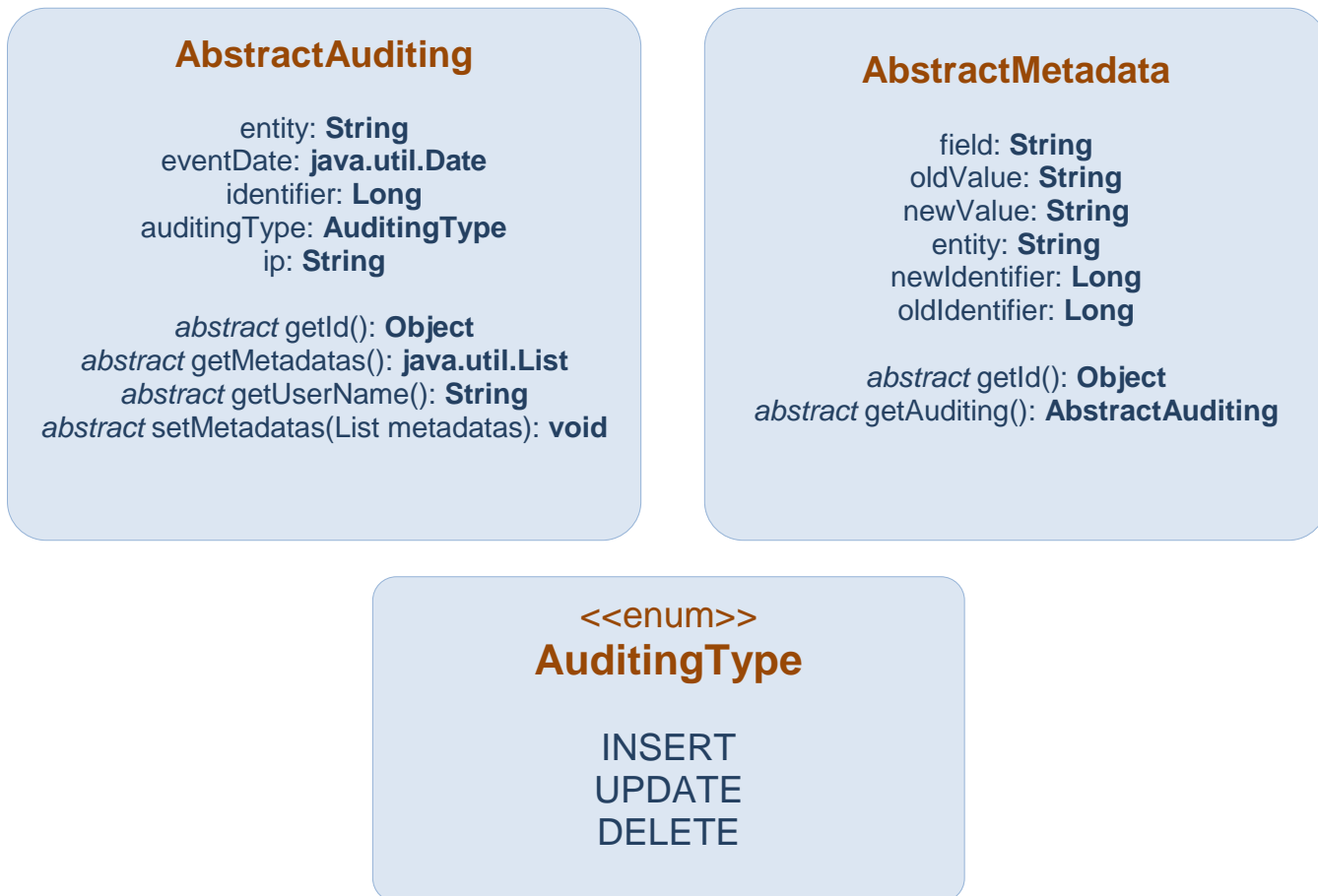
Para evitar que seja possível ainda assim digitar o id na URL e carregar o objeto deve-se sobrescrever o seguinte método no Managed Bean:

```
@Override
public boolean isLoadEntityOnPostConstruct() {
    return false;
}
```

20. Auditando as entidades com o Xpert-Audit

20.1. Introdução

O objetivo desse módulo do framework é auditar as entidades na inserção, atualização e exclusão. As classes de auditoria seguem o seguinte modelo:



20.2. AbstractAuditing - Classe que representa uma auditoria feita

20.2.1. Introdução

Esta classe representa uma auditoria feita para uma entidade. Caso um evento que gera auditoria ocorra, um objeto deste tipo será criado, por exemplo, cada inserção gera um *Auditing*, assim como atualização e exclusão. Por exemplo, ao se salvar um objeto da classe City seria criado um registro da seguinte maneira:

Detalhamento da Auditoria			
	Data	Tipo	Usuário
+	18/03/2013 23:44:21	Inclusão	
<div><div></div><div></div><div>1</div><div></div><div></div><div>10</div></div>			

20.2.2. Atributos

Atributo/Método	Tipo	Descrição
entity	String	Representa a entidade atual a ser auditada. Caso a anotação @Table esteja presente, o valor dela é usado, caso a anotação @Entity esteja com valor definido este é usado, caso contrário usa-se o nome simples da classe.
eventDate	java.util.Date	Data da auditoria.
identifier	Long	Identificador do objeto auditado.
auditingType	AuditingType	Enum que representa o tipo de auditoria (insert, update ou delete)
Ip	String	O ip da máquina que realizou o evento, é obtido através do método <i>FacesUtils.getIp()</i>
getId()	Object	Retorna o id do registro atual de auditoria
getMetadatas()	java.util.List	Retorna uma lista de metadados da auditoria atual
getUserName()	String	Método que retorna o usuário da auditoria, este método é usado no componente de auditoria.
setMetadatas()	void	Seta uma lista de metadados da auditoria atual.

20.2.3. Configuração

A implementação dessa classe deve ser declarada no arquivo xpert-config (mais detalhes sobre este arquivo na seção [Configurando o xpert-config.xml](#)) através da tag **auditing-impl**.

20.2.4. Exemplo de uma implementação

O exemplo a seguir mostra o básico da classe, sendo adicionado apenas o objeto user, que seria simulado o usuário logado de uma aplicação, mais atributos também podem ser adicionados.

```

@Entity
public class Auditing extends AbstractAuditing implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    private Person user;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "auditing")
    private List<Metadata> metadatas;

    @Override
    public Long getId() {
        return id;
    }
    public Person getUser() {
        return user;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public void setUser(Person user) {
        this.user = user;
    }
    @Override
    public List getMetadatas() {
        return metadatas;
    }
    @Override
    public void setMetadatas(List metadatas) {
        this.metadatas = metadatas;
    }
    @Override
    public String getUsername() {
        if(user != null){
            return user.getName();
        }
        return "";
    }
}

```

20.3. AbstractMetadata - Classe que representa os metadados

20.3.1. Introdução

Quando um registro de auditoria é criado ele pode está vinculado a metadados, que são basicamente os valores dos campos naquele momento. Por exemplo a classe ao se salvar um objeto da Classe City será salvo os metadados da seguinte maneira:

Detalhamento da Auditoria			
	Data	Tipo	Usuário
-	18/03/2013 23:44:21	Inclusão	
	Campo	Valor Anterior	Valor Novo
1	name		Guarulhos
2	state		SÃO PAULO
3	creation		

1 10

20.3.2. Atributos

Atributo/Método	Tipo	Descrição
field	String	Representa o campo que foi auditado.
oldValue	String	Representa o valor original do registro auditado, para inserção esse valor é vazio.
newValue	String	Representa o valor novo do registro auditado.
entity	String	Nome da classe do tipo auditado quando este se refere a um objeto complexo.
newIdentifier	Long	Novo Identificador da entidade do campo auditado quando este é um objeto complexo.
oldIdentifier	Long	Identificador original da entidade do campo auditado quando este é um objeto complexo.
getId()	Object	Retorna o id do registro atual do metadado.
getAuditing()	AbstractAuditing	Retorna o registro de auditoria para o metadado.
getMaxSizeValues()	Integer	Retorna o tamanho máximo que deve ser inserido nos campos "oldValue" e "newValue". Por padrão retorna null, ou seja, sem tamanho definido. Lembre-se ao deixa-lo sem tamanho definido é necessário mudar as colunas "oldValue" e "newValue" para um tipo sem limite de tamanho, como o "text" do PostgreSQL e o "clob" do Oracle.

20.3.3. Configuração

A implementação dessa classe deve ser declarada no arquivo `xpert-config` (mais detalhes sobre este arquivo na seção [Configurando o xpert-config.xml](#)) através da tag **metadata-impl**.

20.3.4. Exemplo de uma implementação

O exemplo a seguir mostra o básico da classe onde o atributo **auditing** é do tipo **Auditing**, mostrado na seção anterior.

```

@Entity
public class Metadata extends AbstractMetadata{

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    private Auditing auditing;

    @Override
    public Long getId() {
        return id;
    }

    @Override
    public AbstractAuditing getAuditing() {
        return auditing;
    }

    @Override
    public void setAuditing(AbstractAuditing auditing) {
        this.auditing = (Auditing) auditing;
    }
}

```

20.4. Listener para a auditar um objeto

20.4.1. Introdução

Em uma aplicação real uma auditoria deve-se indicar o usuário que realizou a operação, o xpert-framework deixa de maneira genérica essa parte. Para isso é disponibilizado a classe AbstractAuditingListener, através dela o projeto pode ter uma implementação própria e nela fazer sua lógica.

20.4.2. Configuração

A implementação dessa classe deve ser declarada no arquivo xpert-config (mais detalhes sobre este arquivo na seção [Configurando o xpert-config.xml](#)) através da tag **auditing-listener**.

20.4.3. Exemplo de uma implementação

O exemplo a seguir se baseia na implementação do tipo **Auditing**, mostrado na seção anterior. O método SessionUtils.getUser() é apenas um exemplo, nesse trecho deve ser feita a lógica para setar o usuário.

```

public class AuditingListenerImpl implements AbstractAuditingListener {

    @Override
    public void onSave(AbstractAuditing abstractAuditing) {
        Auditing auditing = (Auditing)abstractAuditing;
        //example, set here the current user
        auditing.setUser(SessionUtils.getUser());
    }
}

```

20.5. Auditando uma entidade

Para auditar uma entidade deve-se chamar a classe Audit (com.xpert.audit). Esta classe possui os métodos para auditar a inserção, atualização e exclusão da entidade.

20.5.1. Auditar um Insert

Deve ser chamado após o método de salvar

```
public void save(T object) {
    getEntityManager().persist(object);
    new Audit(getEntityManager()).insert(object);
}
```

20.5.2. Auditar um Update

Deve ser chamado antes de o objeto ser persistido. O exemplo abaixo mostra como usar dentro de um método que utiliza o merge do JPA, já verificando se é um objeto novo ou um a ser atualizado:

```
public T merge(T object) {
    boolean persisted = object.getId() != null;
    if (persisted == true) {
        new Audit(getEntityManager()).update(object);
    }
    object = (T) getEntityManager().merge(object);
    if (persisted == false) {
        new Audit(getEntityManager()).insert(object);
    }
    return object;
}
```

20.5.3. Auditar um Delete

Deve ser chamado antes de se deletar a entidade. O exemplo abaixo utiliza o método remove do EntityManager:

```
public void remove(Object object){
    new Audit(getEntityManager()).delete(object);
    getEntityManager().remove(object);
}
```

20.5.4. BaseDAOImpl do xpert-framework e sua auditoria

O DAO BaseDAOImpl(com.xpert.persistence.dao) do xpert-framework por padrão usa auditoria ao salvar ou deletar o objeto e o pode ser chamado a partir dos seguintes métodos:

```
public void save(T object);
public T merge(T object);
public void saveOrUpdate(T object);
public void update(T object);
public void saveOrMerge(T object);
public void delete(Object id) throws DeleteException;
public void remove(Object object) throws DeleteException;
```

Para indicar se a entidade deve ou não ser auditada pode se chamar os métodos (o parâmetro *audit* indica se deve ser auditado):

```
public void save(T object, boolean audit);
public void saveOrUpdate(T object, boolean audit);
public void saveOrMerge(T object, boolean audit);
public void update(T object, boolean audit);
public void delete(Object id, boolean audit) throws DeleteException;
public void remove(Object object, boolean audit) throws DeleteException;
public T merge(T object, boolean audit);
```

20.6. Exibindo a auditoria de uma determinada entidade

O xpert-framework disponibiliza o componente “audit” e “auditDelete” para que a auditoria seja visualizada na tela através de componentes JSF e Primefaces.

Para realizar a consulta da auditoria o xpert-framework precisa que seja definido o EntityManagerFactory e que seja devidamente configurado, isso pode ser visto na sessão [Configurando o EntityManagerFactory](#).

Ambos os componentes estão acessíveis através do seguinte namespace:

Para exibir a auditoria de um determinado objeto:

```
<x:audit for="#{personMB.person}"/>
```

Para exibir as exclusões de uma determinada classe de uma entidade:

```
<x:auditDelete for="#{personMB.personClass}"/>
```

O objeto personClass acima deve retornar um java.lang.Class.

20.7. Internacionalizar valores dos campos na Auditoria

Por padrão ao ser auditado um objeto o seu metadata é igual ao nome do campo no java (pego via reflection). Assim a classe “Person.java” que possui o atributo “name” ao ser auditada criaria uma metadata com o campo nome.

A regra para exibição desse valor internacionalizado segue o padrão de internacionalização o BeanValidator do xpert-framework. Para o exemplo citado, bastaria colocar no *bundle* o seguinte valor:

```
person.name=Nome
person.fullName=Nome Completo
```

É necessário informar no arquivo **xpert-config.xml** a localização do *bundle*, conforme visto na sessão de configuração:

```
<xpert-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <resource-bundle>bundles.messages</resource-bundle>
</xpert-config>
```

20.8. Ignorar auditoria de uma classe ou de um atributo

É possível ignorar auditoria para uma determinada classe ou atributo usando a anotação `@NotAudited` (com.xpert.audit), por padrão campos anotados com `@Transient` (javax.persistence) também não são auditados.

Exemplo de como ignorar auditoria de uma classe:

```
import com.xpert.audit.NotAudited;

@NotAudited
public class City {

}
```

Exemplo de como ignorar um atributo específico (nesse caso o atributo `code` será ignorado na auditoria):

```
import com.xpert.audit.NotAudited;

public class City {
    private Long id;
    private String name;
    @NotAudited
    private String code;
}
```

20.9. Acessando alterações do objeto com o AuditContext

Caso um objeto tenha sido alterado, é possível acessar essas alterações através da classe **AuditContext**.

O **AuditContext** dura apenas a requisição (request-scope), após isso ele é descartado, já que a idéia é ter um acesso rápido nas últimas alterações. Sua instância pode ser acessada da seguinte maneira:

```
AuditContext auditContext = AuditContext.getCurrentInstance();
```

Para acessar a última auditoria:

```
AbstractAuditing auditing = AuditContext.getCurrentInstance().getAuditing(object);
```

Para acessar os metadados da última auditoria:

```
List<Metadata> metadatas = AuditContext.getCurrentInstance().getMetadata(person);
```

O exemplo abaixo mostra como acessar a lista de metadados após realizar um merge no objeto:

```
@EJB
private PersonDAO personDAO;

public void save(Person person){
    //salvando
    personDAO.merge(person);

    //acessando o audit context
    AuditContext auditContext = AuditContext.getCurrentInstance();
    if(auditContext != null ){
        List<AbstractMetadata> metadatas = auditContext.getMetadata(person);
        if(metadatas != null){
            for(AbstractMetadata metadata : metadatas){
                System.out.println("Campo: "+metadata.getFieldName());
                System.out.println("Valor Antes: "+metadata.getOldValue());
                System.out.println("Valor Depois: "+metadata.getNewValue());
            }
        }
    }
}
```


21. SequenceUpdater - Atualizando as sequences do banco de dados

21.1. Introdução

Um grande problema quando se trata de *sequences*, é a manutenção desses valores, muitas vezes ao se fazer *backup* ou *restore* do banco acontece por motivos diversos (desde configuração até falha humana) de as *sequences* não virem com seus valores atualizados, ou mesmo virem zeradas.

Como é possível acessar todas as classes mapeadas pelo hibernate, o **xpert-framework** se utilizou disso (versão **1.8.1**) para que seja possível fazer a atualização da *sequence* baseada na tabelas (fazendo um `max(id)` na tabela).

21.2. Utilização do SequenceUpdater

Considerando a tabela “**pessoa**” (*id*, *nome*, *cpf*) e a *sequence* “**seq_pessoa**“, para pegar o valor atual da *sequence* seria: “`select max(id) from pessoa`“, seguindo essa idéia foi criado a classe abstrata “**SequenceUpdater**” e suas implementações, atualmente para oracle e postgresql: **OracleSequenceUpdater** e **PostgreSQLSequenceUpdater**. Ambas as classes precisam de um *EntityManager* para que as entidades sejam acessadas.

21.2.1. OracleSequenceUpdater

Atualizador de *sequence* para o banco de dados oracle, é necessário adicionar o import:

```
import com.xpert.persistence.utils.OracleSequenceUpdater;
```

Utilização

```
OracleSequenceUpdater sequenceUpdater = new OracleSequenceUpdater(entityManager);  
sequenceUpdater.updateSequences();
```

21.2.2. PostgreSQLSequenceUpdater

Atualizador de *sequence* para o banco de dados postgresql, é necessário adicionar o import:

```
import com.xpert.persistence.utils.PostgreSQLSequenceUpdater;
```

Utilização

```
PostgreSQLSequenceUpdater sequenceUpdater = new PostgreSQLSequenceUpdater(entityManager);  
sequenceUpdater.updateSequences();
```

21.3. Adicionando o atualizador de *sequence* no projeto base (criado do archetype maven)

1 - Certifique-se no **pom.xml** a versão do **xpert-framework** é a **1.8.1** ou superior

2 - Na classe **GeracaoDadosSistema** criar o seguinte método (caso seja oracle usar o *OracleSequenceUpdater*):

```
public void updateSequences(){
    SequenceUpdater sequenceUpdater = new PostgreSQLSequenceUpdater(dao.getEntityManager());
    sequenceUpdater.updateSequences();
}
```

3 - Na classe **GeracaoDadosMB** criar o seguinte método:

```
public void updateSequences() {
    geracaoDadosSistema.updateSequences();
}
```

4 - Na página **maker.xhtml** adicionar o **commandButton** (ao lado do botão “**Gerar Dados**”) que chamará a ação:

```
<p:commandButton action="#{geracaoDadosMB.updateSequences}"
    value="Update Sequences"
    update="@form" process="@this"/>
```

5 - Acessar a página **maker.jsf** na aba “**Geração de Dados**”, clicar em “**Update Sequences**”. Lembre-se que por padrão essa página *maker.jsf*, só é acessível estando logado no sistema com um *super usuário*.

22. Controle de Acesso através do xpert-security

22.1. Introdução

Através desse módulo pode-se fazer o controle de acesso da aplicação. Este controle de acesso deve ser feito tanto a nível de urls, caso um usuário acesse uma url que não possua acesso ele deve ser redirecionado quanto a nível de componentes na tela, ou seja, uma dada funcionalidade só pode ser acessada para quem possuir sua permissão.

Dentro da suíte de componentes JSF também são disponibilizados componentes que fazem o controle de acesso a nível de componente (renderizar ou não um trecho da página).

Os artefatos que estão presentes no xpert-security estão listados abaixo:

AbstractUserSession

Guarda os dados da sessão do usuário. Pode ser um @ManagedBean de Sessão

AbstractSecurityFilter

Filtro para controle de usuário logado e de permissão por página.

SecurityLoginBean

Bean que gerencia o login e logout no sistema.

securityArea

Componente JSF para fazer controle de acesso a nível de componentes na tela.

22.2. Gerenciamento da Sessão a partir da classe SecuritySessionManager

A classe SecuritySessionManager (com.xpert.security) possui alguns métodos que ajudam no controle de sessão de usuário e permissões. Abaixo a lista dos principais métodos:

Método	Retorno	Descrição
<code>clearRoles()</code>	void	Limpa as permissões da sessão atual.
<code>putRoles(roles)</code>	void	Adiciona as permissões na sessão atual. O parâmetro de entrada é uma lista de do objeto <code>com.xpert.security.model.Role</code> .
<code>hasURL(url)</code>	boolean	Retorna true se a sessão atual possui a url passada por parâmetro.
<code>hasRole(key)</code>	boolean	Retorna true se a sessão atual possui alguma permissão com a chave passada por parâmetro. Várias chaves podem ser passadas separando por vírgula.
<code>getRoles(request)</code>	List<Role>	Retorna a lista de permissões da sessão atual.

22.3. Bean para manter usuário e permissões na sessão

Um usuário logado na aplicação e suas permissões devem ser mantidos na sessão, para isso é disponibilizado a classe `AbstractUserSession` (`com.xpert.security.session`), através dela é possível definir esse dados de sessão. Esta classe pode ser utilizada em combinação com o `SecurityFilter` (próximo tópico) para fazer o controle de acesso a nível de URL.

Segue abaixo um exemplo de uma classe filha de `AbstractUserSession`:

```
@ManagedBean
@SessionScoped
public class UserSessionMB extends AbstractUserSession implements Serializable {
```

```
@EJB
private RoleBO roleBO;
private AdminUser user;
private List<Role> roles;
```

```
@Override
public void createSession() {
    roles = roleBO.getRoles(user);
    super.createSession();
}
```

```
@Override
public void setUser(User user) {
    this.user = (AdminUser) user;
}
```

```
@Override
public AdminUser getUser() {
    return user;
}
```

```
@Override
public List getRoles() {
    return roles;
}
```

```
}
```

createSession() xpert-framework

Este método é utilizado para capturar as roles a partir de um usuário, a chamada do `super.createSession()` faz com que essas roles sejam colocadas na sessão. Este método faz uso da classe `SecuritySessionManager` para colocar na sessão as permissões.

setUser() xpert-framework

Deve ser utilizado para setar o usuário.

getUser() xpert-framework

Retorna o usuário da sessão atual

getRoles() xpert-framework

Retorna as permissões da sessão atual.

22.4. Filtro para bloquear o acesso do usuário por página

Para fazer esse controle por página o xpert-framework disponibiliza o filtro `AbstractSecurityFilter` (pacote `com.anuncios.filter`), essa classe é um facilitador para o controle de acesso a nível de páginas e de usuário logado na aplicação.

Exemplo de filtro para o controle de acesso:

```
@WebFilter(filterName = "AdminFilter", urlPatterns = {"/view/*"})
public class AdminFilter extends AbstractSecurityFilter {

    @Override
    public String getUserSessionName() {
        return "userSessionMB";
    }

    @Override
    public String getHomePage() {
        return "/index.jsf";
    }

    @Override
    public String[] getIgnoredUrls() {
        return null;
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void destroy() {
    }
}
```

getUserSessionName()
xpert-framework
Nome do objeto de sessão que possui as informações do usuário logado. O objeto a ser guardado na sessão deve ser uma instância de AbstractUserSession (com.xpert.security.session).

getHomePage()
xpert-framework
Define a página inicial do sistema, caso não possua usuário logado, ou este não possua a página atual essa página será retornada.

getIgnoredUrls()
xpert-framework
Define as urls que não entrarão no controle de acesso. Este método retorna uma lista de urls onde não é feita a verificação de acesso.

Esse filtro é uma implementação de javax.servlet.Filter, então configurações de onde deve agir o filtro são dadas através de sua especificação.

22.5. Bean para realizar login/logout do usuário na aplicação

A classe SecurityLoginBean (com.xpert.security.bean) faz a abstração da lógica de login e logout do usuário. Ela disponibiliza métodos onde se define as configurações do login.

Exemplo de uma classe filha de SecurityLoginBean:

```

@ManagedBean
public class LoginMB extends SecurityLoginBean {

    @EJB
    private DAO dao;
    @ManagedProperty(value = "#{userSessionMB}")
    private UserSessionMB userSessionMB;

    @Override
    public String getRedirectPageWhenSucess() {
        return "/view/home.jsf";
    }

    @Override
    public String getRedirectPageWhenLogout() {
        return "/index.jsf";
    }

    @Override
    public String getUserNotFoundMessage() {
        return "User not found";
    }

    @Override
    public String getInactiveUserMessage() {
        return "User inactive";
    }

    @Override
    public EntityManager getEntityManager() {
        return dao.getEntityManager();
    }

    @Override
    public boolean isLoginUpperCase() {
        return true;
    }

    @Override
    public boolean isValidWhenNoRolesFound() {
        return true;
    }

    @Override
    public void onSuccess(User user) {
    }

    @Override
    public Class getUserClass() {
        return User.class;
    }

    @Override
    public UserSessionMB getUserSession() {
        return getUserSessionMB();
    }

    public UserSessionMB getUserSessionMB() {
        return userSessionMB;
    }

    public void setSessaoUsuarioMB(UserSessionMB userSessionMB) {
        this.userSessionMB = userSessionMB;
    }

}

```

getRedirectPageWhenSucess() xpert-framework

Define a página a ser retornada quando o login for efetuado com sucesso.

getRedirectPageWhenLogout() xpert-framework

Define a página a ser retornada quando realizado o logout.

getUserNotFoundMessage() xpert-framework

Define a mensagem a ser exibida quando o usuário não for encontrado.

getInactiveUserMessage() xpert-framework

Define a mensagem a ser exibida quando o usuário não estiver ativo.

getEntityManager() xpert-framework

Define o entity manager a ser utilizado na consulta do usuário ao se logar.

isLoginUpperCase() xpert-framework

Define se o login do sistema é uppercase, para que na verificação a String seja convertida. Para ignora o case, deve-se sobrescrever o método `isLoginIgnoreCase` retornando `true`

isValidWhenNoRolesFound() xpert-framework

Define se deve evitado o login de um usuário que não foi encontrado permissões.

onSuccess() xpert-framework

Método chamado quando o login é efetuado com sucesso, pode ser alguma lógica

Apesar de longa essa classe mostrada como exemplo exibe tudo que pode ser configurável no login do usuário, como por exemplo definir se o login deve ser UpperCase, ou ainda se deve ser validado quando não se encontra permissões para esse usuário.

22.6. Login utilizando SecurityLoginBean

Considerando a classe LoginMB (mostrada) acima a página para fazer esse login no sistema pode ser feito da seguinte maneira:

```
<h:form>
  <p:messages />
  <h:panelGrid columns="2">

    <h:outputLabel for="user" value="User:" />
    <p:inputText value="#{loginMB.userLogin}" required="true" requiredMessage="User is required"/>

    <h:outputLabel for="password" value="Password:" />
    <p:password feedback="false" value="#{loginMB.userPassword}"
      required="true" maxLength="20" requiredMessage="Password is required"/>

  </h:panelGrid>

  <p:commandButton process="@form" update="@form" value="Login" action="#{loginMB.login}" />
</h:form>
```

Esse exemplo mostrado utiliza componentes primefaces, mas eles não são obrigatórios para isso.

22.7. SecurityArea para verificação de acesso a nível de componente

Este componente está disponível através do namespace `xmlns:x="http://xpert.com/faces"` e recebe como parâmetro a chave de acesso (múltiplas chaves podem ser passadas separadas por vírgula) daquele trecho de código.

O exemplo abaixo mostra um `commandButton` que só deve ser exibido para quem possua o acesso "person.delete":

```
<x:securityArea rolesAllowed="person.delete">
  <p:commandButton process="@form" update="@form" action="#{personMB.delete}" />
</x:securityArea>
```

Abaixo o mesmo exemplo passando-se mais de uma chave:

```
<x:securityArea rolesAllowed="person.delete,person.create">
  <p:commandButton process="@form" update="@form" action="#{personMB.delete}" />
</x:securityArea>
```


23. Arquétipo para criação de projeto

23.1. Introdução

A idéia do arquétipo (*archetype*) é facilitar a criação de novos projetos que utilizem o xpert-framework. Através deles é possível gerar sem muito trabalho um novo projeto que já possua controle de acesso, auditoria de dados um layout base utilizando o framework *bootstrap*.

23.2. Arquétipos disponíveis do xpert-framework

Existem 2 (dois) arquétipos: o EAR (**xpert-framework-ear-archetype**), e o WAR (**xpert-framework-war-archetype**). O EAR é dividido nos módulos “main” (classes de infraestrutura e de domínio), “ejb” (DAOs e regra de negócio) e “web” (Managed Beans e artefatos web como CSS, javascript e XHTML).

Todos os arquétipos podem ser baixados na seção de downloads no link: <https://code.google.com/p/xpert-framework/wiki/Download?tm=2>

Qual módulo usar? Isso fica a cargo do desenvolvedor escolher qual deles usar, abaixo segue algumas características de cada tipo:

Archetype	Características
ear-archetype	<ul style="list-style-type: none">• Possibilidade de compartilhar os módulos com outros projetos• Facilidade para criar novos módulos independentes• Pode levar mais tempo para ser empacotado já que múltiplos módulos estão envolvidos
war-archetype	<ul style="list-style-type: none">• Menos artefatos são gerados• Projeto único (apenas um war)• Geralmente mais simples para as IDE executar/compilar, já que precisa compilar apenas o WAR

23.3. Características de um projeto gerado pelo arquétipo

Por padrão os projetos gerados pelos arquétipo do xpert-framework já possuem as seguintes tecnologias:

- Primefaces 5.2
- Hibernate 4
- bootstrap (framework CSS, HTML e javascript)
- omnifaces 1.10

O projeto já implementa o controle de acesso do xpert-framework através das classes como a *Usuario.java*, *Permissao.java*, *Perfil.java*, entre outras. Além disso, também implementa a **auditoria** do xpert-framework e registro no banco de dados dos erros gerados na aplicação.

O projeto utiliza um layout simples utilizando o framework bootstrap, deixando assim, as páginas responsivas.

Abaixo é descrito de forma resumida as características do arquétipo:

Controle de Acesso	<ul style="list-style-type: none">• Controle de acesso através de login e senha• Cadastro de usuários, permissões e perfis• Controle de senhas com possibilidade de recuperação de senha por email• Registro de acessos do usuário
Auditoria de dados	<ul style="list-style-type: none">• Implementação da auditoria do xpert-framework• Registro dos <i>inserts</i>, <i>updates</i> e <i>deletes</i>
Bootstrap	<ul style="list-style-type: none">• Layout gerado a partir da estrutura do framework bootstrap
Controle de Erro	<ul style="list-style-type: none">• Registro dos erros gerados pela aplicação• A pilha de erros é persistida em banco de dados• Funcionalidade no sistema para visualização da pilha de erros

23.4. Configuração de banco de dados

23.4.1. Mapemanto das entidades

O arquétipo padrão é voltado para o banco PostgreSQL, assim alguns detalhes devem ser mencionados quanto ao mapeamento das classes.

Por padrão as entidades que irão gerar tabelas no banco de dados usam *sequence* para geração do seu *id* (conforme a especificação para mapeamento de entidade JPA), isso significa que o projeto gerado no arquétipo quando usado em bancos que não aceitam *sequences* deverão mudar o mapeamento do seus identificadores.

Então a classe *Usuario.java* por exemplo:

```
@Entity
public class Usuario implements Serializable, User {

    @Id
    @SequenceGenerator(name = "Usuario", allocationSize = 1, sequenceName = "usuario_id_seq")
    @GeneratedValue(generator = "Usuario")
    private Long id;
}
```

Deve utilizar outra maneira para geração de id, por exemplo, o tipo IDENTITY:

```
@Entity
public class Usuario implements Serializable, User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

}
```

Outros tipos para geração de identificador devem ser considerados também, como o *GenerationType.AUTO* o *GenerationType.TABLE*.

23.4.2. Tipo texto no banco de dados

Classes como a *ErroSistema* (que mapeia os erros do sistema) utilizam tipos de text com tamanho sem definição (geralmente o varchar é gerado como 255 pelo *hibernate*) esse tipo é explicitado na anotação *@Column*, como mostrado abaixo:

```
@Entity
public class ErroSistema implements Serializable {

    @Column(columnDefinition = "text")
    private String pilhaErro;

}
```

Esse tipo foi colocado no arquivos de constantes, então para muda-lo, basta mudar no arquivo **Constantes.java** pacote `$(seupacote).constante` (no EAR fica no módulo *main*).

Assim na classe *ErroSistema*, por exemplo, é usado a constante:

```
@Column(columnDefinition = Constantes.TIPO_TEXTO_BANCO)
private String pilhaErro;
```

Abaixo alguns exemplos de tipos em diferentes bancos:

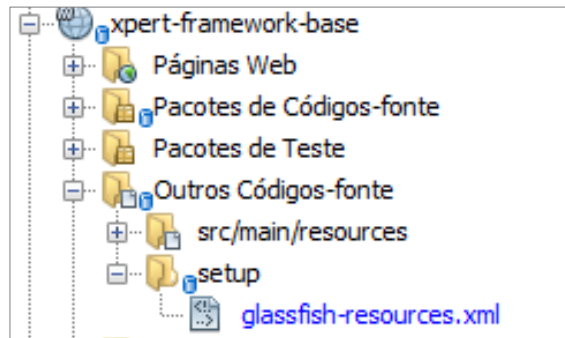
```
//oracle
public static final String TIPO_TEXTO_BANCO = "clob";

//mysql
public static final String TIPO_TEXTO_BANCO = "longtext";

//postgres
public static final String TIPO_TEXTO_BANCO = "text";
```

23.4.3. Geração do Data Source no glassfish

O arquétipo padrão possui uma configuração para facilitar a geração do Data Source (com pool de conexões) no glassfish, essa configuração é descrita no arquivo **glassfish-resources.xml** (pasta "setup" do módulo web), como mostrado na imagem abaixo:



Nesse arquivo é possível configurar o driver de conexão, nome do banco de dados, usuário, senha, url. Como por padrão é gerado o arquivo voltado para o PostgreSQL, caso seja usado outro SGBD é necessário realizar as configurações necessárias.

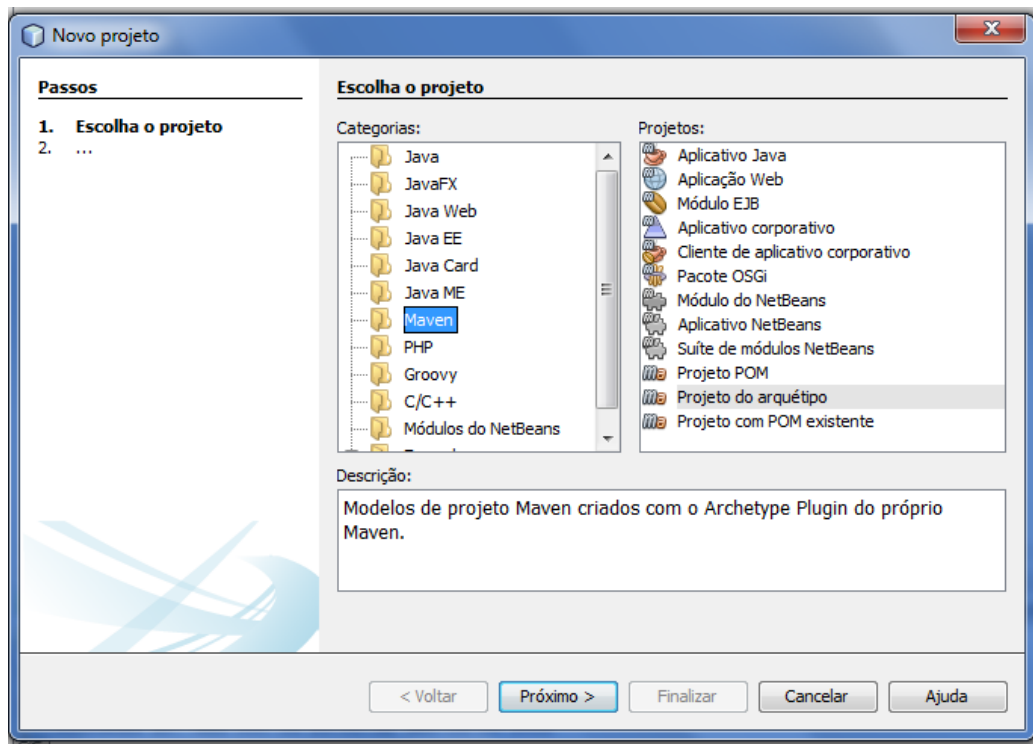
Observação: aparentemente existe algum comportamento no glassfish 4, que ele não reconhece esse arquivo glassfish resources, sendo assim talvez seja necessário criar o pool de conexões manualmente, pela própria interface admin dele por exemplo.

23.5. Arquétipo maven EAR do xpert-framework

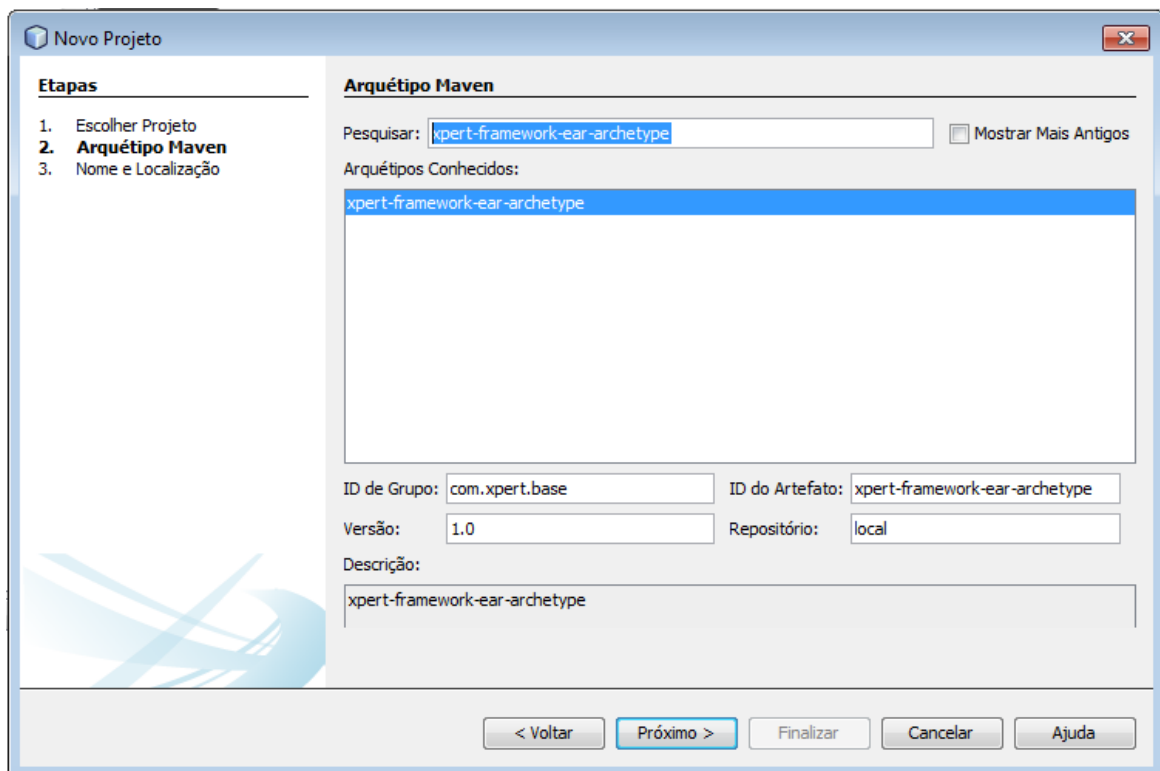
23.5.1. Como criar um projeto a partir do arquétipo

Para a criação de um projeto, o xpert-framework disponibiliza um arquétipo do maven para facilitar o passos iniciais do projeto. O passo a passo abaixo utiliza a IDE Netbeans, mas pode ser usada como base para outras IDEs.

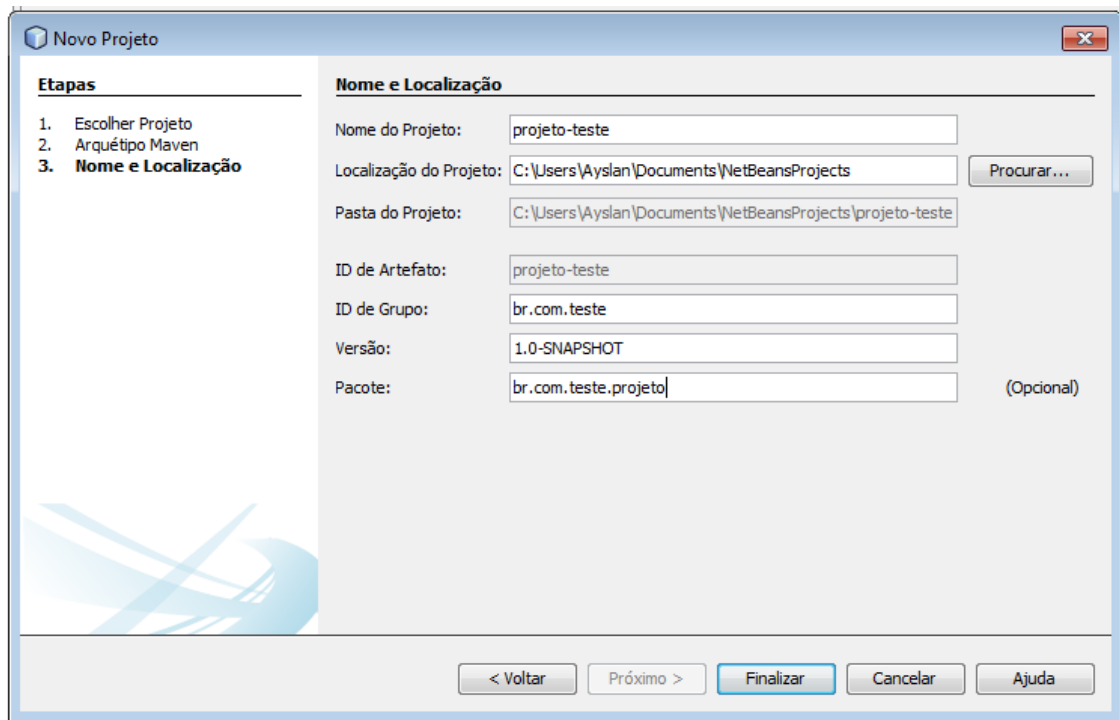
1. Caso já possua o archetype instalado na sua máquina pule para o **passo 4**, senão, baixe o projeto **xpert-framework-ear-archetype** no link:
<https://code.google.com/p/xpert-framework/wiki/Download?tm=2>.
2. Descompacte o archetype e abra o projeto com o Netbeans:
Arquivo -> Abrir Projeto -> xpert-framework-ear-archetype.
3. Clique com direito e selecione “Limpar e Construir”, isso faz com que o Netbeans instale o arquétipo na sua máquina.
4. Para criar o projeto, acesse **Arquivo -> Novo Projeto**. Ao abrir o tipo de projeto selecione **Maven -> Projeto do Arquétipo** clique em “**Próximo**”



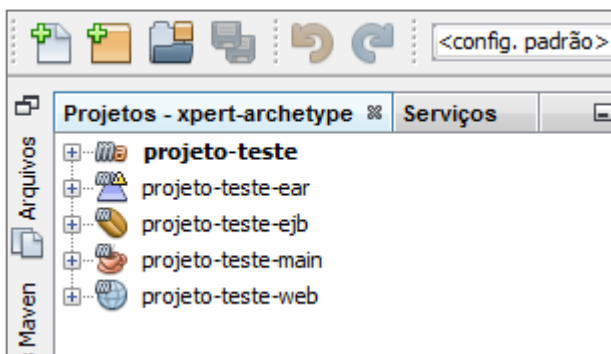
5. Pesquise por **xpert-framework-ear-archetype** e selecione o archetype encontrado, como na figura abaixo (caso não seja encontrado, a instalação não foi feita com sucesso):



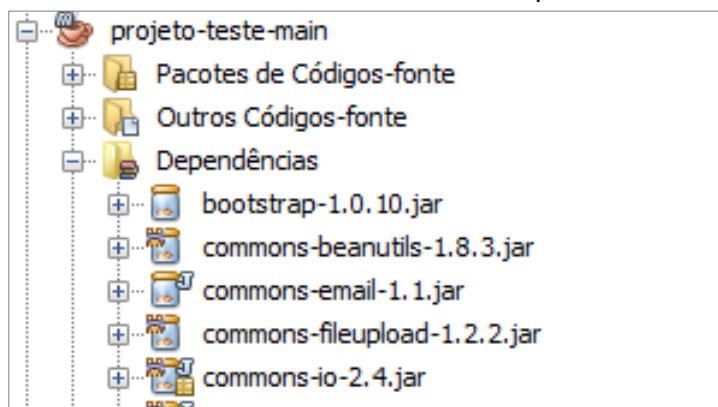
6. Clique em "Próximo", no passo seguinte será necessário informar os dados do seu projeto
7. Configure o projeto inserindo as informações como mostrado abaixo, e logo em seguida clique em **"Finalizar"**.



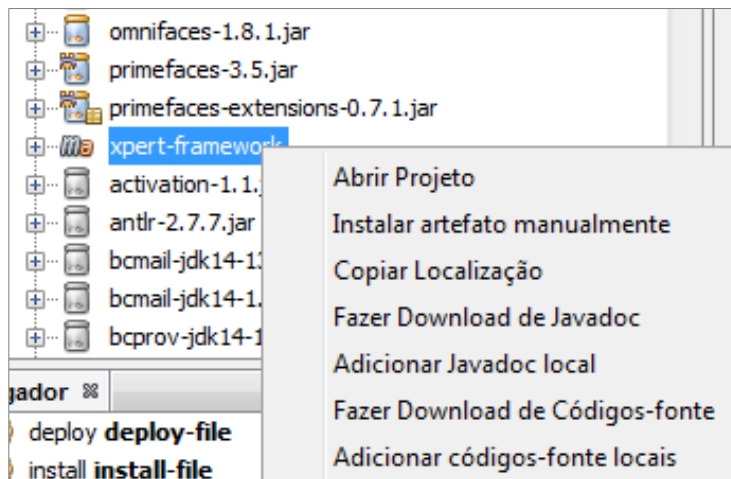
8. O projeto gerado possui a seguinte estrutura:



9. Faça o build do seu projeto (no Netbeans o “limpar e construir”)
10. Caso aconteça erro ao limpar e construir, verifique se faltou alguma dependência
11. Caso seja a dependência do xpert-framework, faça download no link:
<https://code.google.com/p/xpert-framework/wiki/Download?tm=2>
12. Para instalar manualmente o artefato, expanda o módulo “main” -> dependências:



13. Clique com o direito em cima da dependência **xpert-framework**, e clique em “Instalar artefato manualmente”:



14. Caso outra dependência não seja encontrada, repita o processo para a mesma.

Isso conclui a criação de um novo EAR a partir do arquétipo maven.

Observações importantes:

- O arquétipo pode possuir referência a uma versão antiga de alguma lib, como por exemplo, do próprio xpert-framework, para resolver o problema abra o arquivo pom.xml do projeto **main** e muda para a versão desejada.

23.5.2. Executando a aplicação

Antes de executar a aplicação faça algumas verificações para evitar problemas futuros.

1. Vai usar um banco diferente de PostgreSQL? Ele suporta sequence? Verifique a seção [Configuração de banco de dados](#)
2. Certifique-se de ter o driver JDBC do seu banco no diretório do servidor que você vai utilizar. No glassfish por exemplo, fica no diretório **{SEU_DIRETORIO_GLASSFISH}/glassfish/lib**. Segue alguns links para facilitar a encontrar o driver:
 - *PostgreSQL*:
<http://jdbc.postgresql.org/>
 - *MySQL*:
<http://www.mysql.com/products/connector/>
 - *Oracle*:
<http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html>
 - *SQL server*:
<http://msdn.microsoft.com/pt-br/sqlserver/aa937724.aspx>
3. Verifique se está criado o banco de dados no seu SGBD, conforme especificado na sua configuração do Data Source. Caso esteja usando a combinação PostgreSQL e Glassfish, o nome do banco está no **glassfish-resources.xml**, como mostrado na seção [Geração do Data Source no glassfish](#), por padrão o **nome do banco será o nome do projeto**.
4. Caso esteja usando o arquétipo EAR, execute o módulo **seuprojeto-ear**.
5. O usuário padrão da aplicação é:

Usuário: ADMIN

Senha: 1

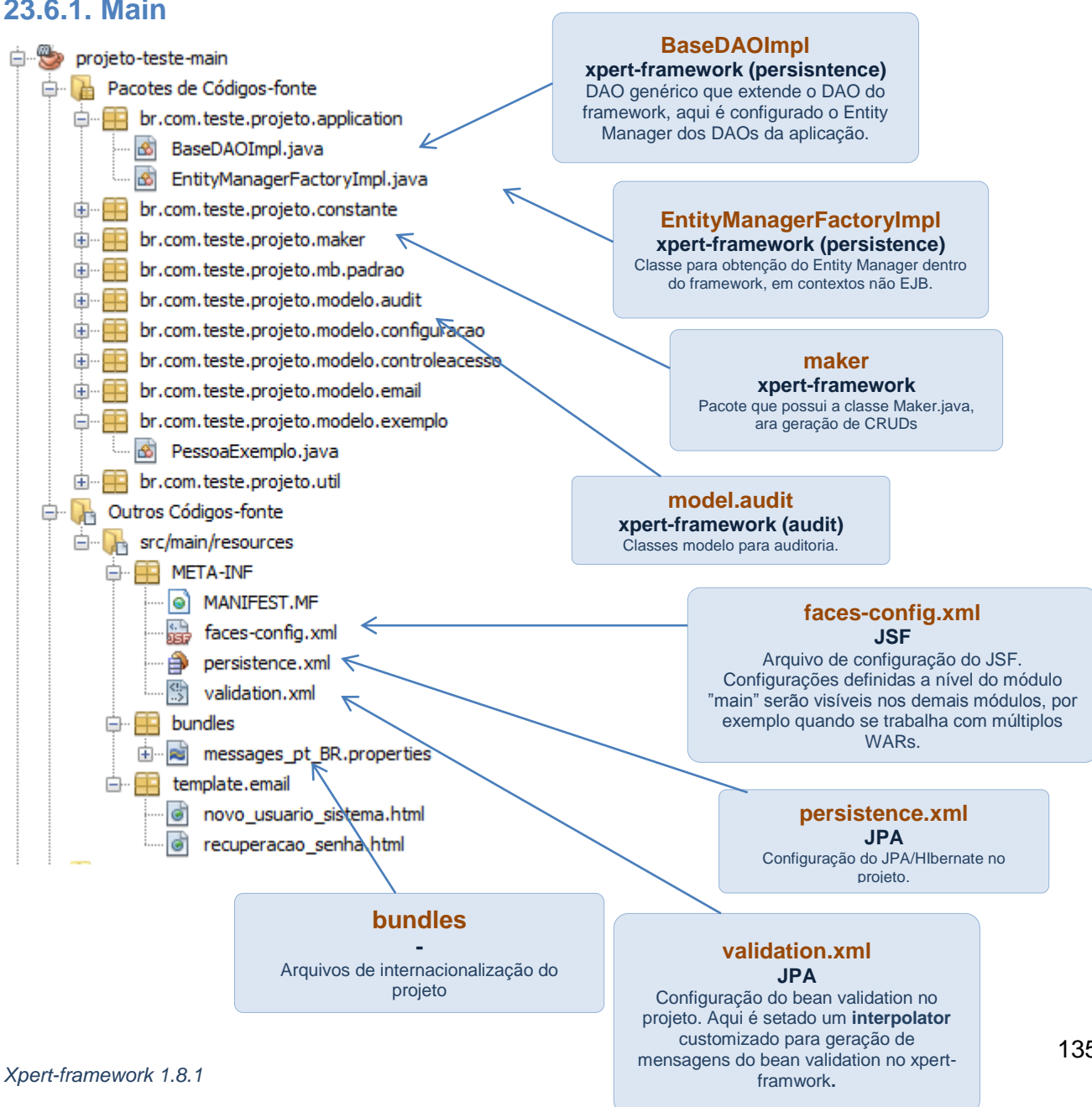
6. Caso você deseje mudar os dados que serão gerados por padrão na aplicação deve-se alterar na classe **GeracaoDadosSistema.java**
7. Para possíveis problemas do projeto gerado acesse a seção de [possíveis problemas](#)

23.6. Estrutura de um projeto gerado a partir do arquétipo maven

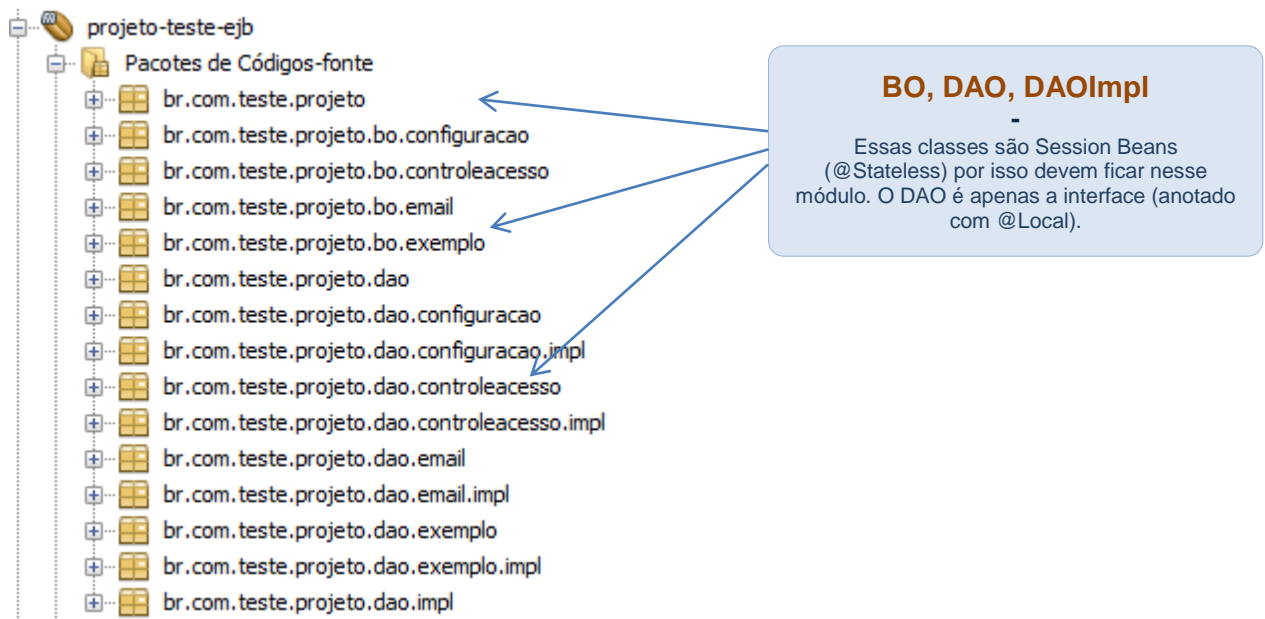
O projeto EAR gerado se divide em 4 módulos:

- **EAR**
- **main** (este projeto possui configurações comuns a todos os módulos, como o **persistence.xml**, **validation.xml**, **arquivos de i18n** e as **classes modelo**)
- **EJB** (BOs e DAOs)
- **WAR** (xhtml e ManagedBeans)

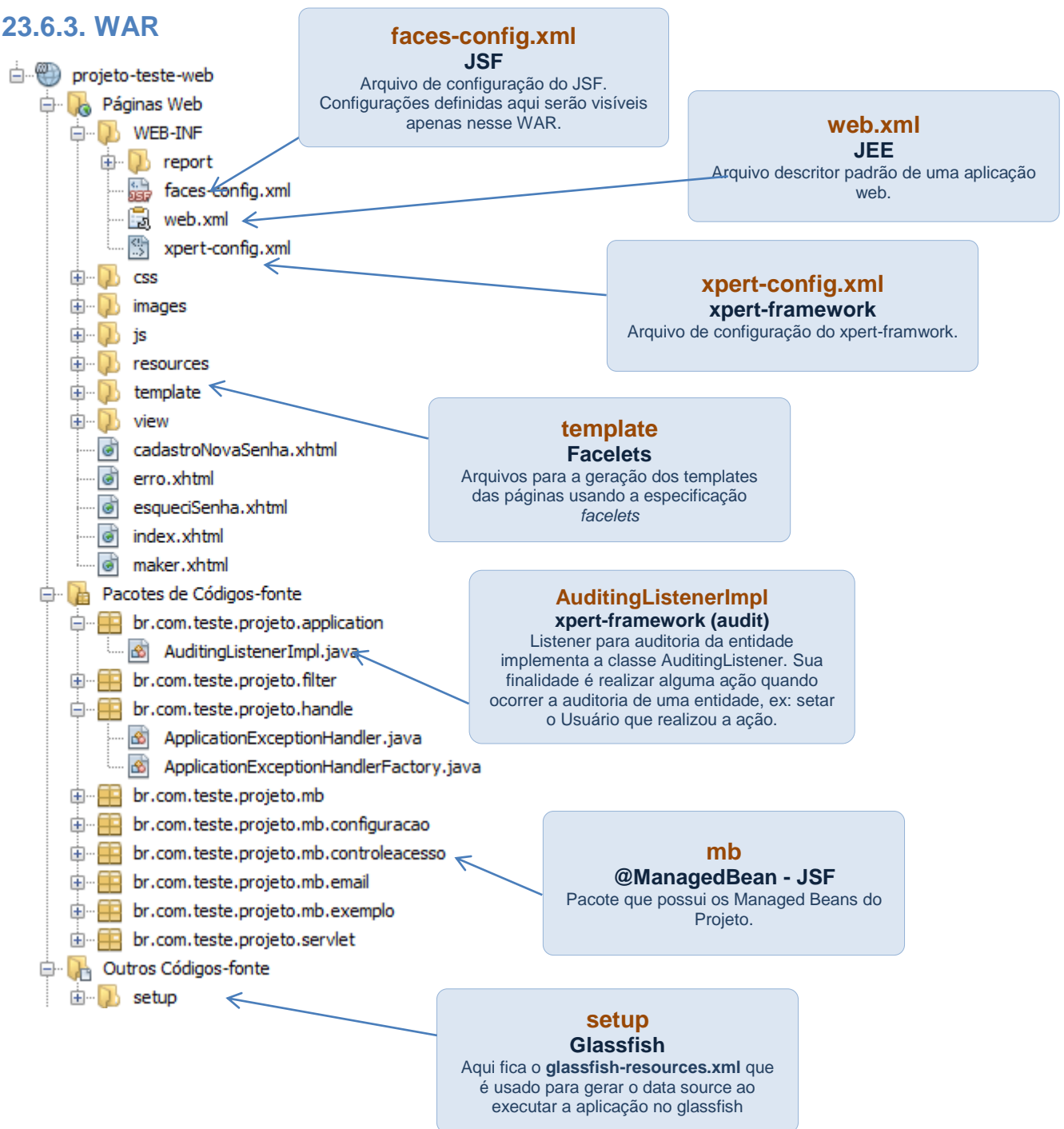
23.6.1. Main



23.6.2. EJB



23.6.3. WAR



24. Boas práticas para o mapeamento de entidades

Aqui serão listadas algumas dicas e boas práticas para o mapeamento de entidades JPA. Uma vez que o modelo esteja bem definido, se torna mais fácil criar regras de negócio com ele, além de facilitar a geração de CRUD com o xpert-maker.

1 Ao mapear uma entidade implemente *Serializable*

É considerado uma boa prática implementar a classe *Serializable* para o mapeamento do JPA, as próprias IDEs já sugerem que você faça isso, veja como é simples:

```
@Entity
public class Pessoa implements Serializable {

    @Id
    private Long id;
    private String nome;
}
```

2 Ao fazer o mapeamento dê preferência para o pacote *javax.persistence*

O pacote **javax.persistence** trata da especificação, então dê preferência a ele, exemplo: você encontra a anotação `@Entity` no pacote **javax.persistence** e no **org.hibernate.annotations**, não tenha dúvida! Prefira a especificação.

Caso alguma anotação não possua o equivalente na especificação do JPA, aí sim você utiliza a do hibernate.

3 Chave primária? Utilize o tipo *Long*!

O tipo *Long* (a classe e não o primitivo) gera no banco de dados tipos com maior quantidade de bytes, então ele consegue suportar valores altos.

Não utilize o long primitivo! Afinal de contas ele não pode ser nulo, e será inicializado com 0 (zero) e lidar com zero antes de persistir pode ser trabalhoso.

2 Valor numérico com casas decimais? Utilize o tipo *BigDecimal*!

O *BigDecimal* possui muitos métodos para tratamento de valores, como arredondamento, potência, raiz e outras operações. Evite usar *double* (mesmo o *Double* classe), prefira sempre o *BigDecimal*.

3 Chave primária? Prefira *não usar chave composta*

Considere usar chaves compostas preferencialmente em sistemas legados (se possível crie uma chave primária simples nele). A maneira que o JPA lida com chaves compostas torna o mapeamento bem mais complexo.

4 Para validação de dados considere usar o *BeanValidation*

O *BeanValidation* é uma especificação já bem madura e bem documentada, então considere usar em seu projeto essa especificação. Ela torna o código legível e também gera as restrições a nível de banco de dados (not-null, por exemplo).

Mais informações no site da especificação:

5 Cada *Bean Validation* em seu lugar

Utilize sabiamente o Bean Validation, usar uma restrição onde ela não é eficiente pode trazer mais problemas do que benefícios. Por exemplo, a tabela abaixo mostra os tipos recomendados para quando se quer validar a obrigatoriedade de campo:

Tipo	Anotação Recomendada	Descrição
String	@NotBlank	@NotBlank é específico do <i>hibernate-validator</i> e infelizmente ainda não está na especificação do <i>bean-validation</i> , sua principal vantagem é que ele verifica a String usando o "trim()". O @NotNull não funciona bem com Strings, pois geralmente elas não são nulas, e sim vazias.
Outros tipos (BigDecimal, Date, etc..)	@NotNull	O @NotNull é um indicativo de obrigatoriedade do campo, e assim como o @NotBlank gera a restrição na base de dados também, ou seja, utilize apenas quando tiver certeza da obrigatoriedade.

6 Use o @Size nas strings para delimitar o seu tamanho

Quando se trabalha com o tipo **varchar** é interessante por questões de desempenho limitar o seu tamanho, quando não se indica o tamanho o hibernate gera o tamanho 255.

Para limitar o tamanho no mapeamento existem 2 formas:

Utilizando o @Column:

```
@Column(length = 100)
private String nome;
```

Utilizando o @Size (bean validation):

```
@Size(max = 100)
private String nome;
```

Prefira o @Size, já que além de ele definir o tamanho a nível de base de dados, também irá validar o objeto seguindo a especificação do bean validation antes de ele ser persistido.

7 Para tipos *Date* utilize o @Temporal

Para tipos Date, não esqueça de definir o tipo de data com a anotação @Temporal, exemplo:

```
@Temporal(TemporalType.DATE)
private Date dataNascimento;
```

Os tipos possíveis são:

- *TIMESTAMP* (data completa)
- *DATE* (apenas dia, mês e ano)
- *TIME* (apenas hora, minuto e segundo)

8 Evite usar **EAGER** em mapeamentos **@OneToMany** ou **@ManyToMany**

Quando se usa o **EAGER** sempre a query vai ser feita para se recuperar o relacionamento. Quando se trata de listas como o **@OneToMany** e o **@ManyToMany** o desempenho da aplicação pode cair muito.

9 Para forçar uma *String* ser maiúscula ou minúscula, altere o **setter**

Fazer isso no setter, faz com que em qualquer lugar da aplicação em que o nome seja setado, ele fique em caixa alta/baixa.

Exemplo: forçar salvar uma pessoa sempre com o nome em caixa alta:

```
public void setNome(String nome) {
    if (nome != null) {
        nome = nome.trim().toUpperCase();
    }
    this.nome = nome;
}
```

10 Sobrescreva o **equals** das entidades

Já que uma entidade representa um registro no banco de dados, então é interessante sobrescrever o **equals** para que o objeto seja comparado pelo **@Id**. Ao gerar o **equals** pela IDE marque a propriedade que representa o identificador da sua classe.

11 Sobrescreva o **toString** das entidades

Afinal de contas um **toString** que retorne *br.com.teste @Pessoa552FA2*, não diz muita coisa. Isso é importante também para o módulo de auditoria do **xpert-framework**, pois em relacionamentos ele irá buscar o **toString** da propriedade.

Dica: não coloque o **toString** com atributos **LAZY**, pois é quase certeza que em algum momento um **LazyInitializationException** irá ocorrer.

12 Utilize **@Max** ou **@Min** para limitar valores numéricos

Para campos numéricos, o **@Size** não vai funcionar, então deve-se utilizar o **@Max** e **@Min**, o exemplo abaixo mostra um campo percentual, que não deve ultrapassar 100%:

```
@Max(value = 100)
private BigDecimal percentual;
```

Considere ainda usar o **@DecimalMax** ou **@DecimalMin** para precisão de casas decimais, no exemplo abaixo, deve ser informado pelo menos 0,50 no campo:

```
@DecimalMax(value = "0.50")
private BigDecimal valorMinimo;
```

25. Possíveis problemas ao utilizar o xpert-framework

25.1. O glassfish não encontra o Data Source do glassfish-resources.xml

Esse problema ocorre no glassfish 4.x, que por algum comportamento dele o arquivo **glassfish-resources.xml**, não é reconhecido e assim o DataSource não é criado.

Solução

Criar manualmente o datasource do glassfish e o recurso JNDI, a interface gráfica do admin do servidor facilita essa criação (geralmente na porta padrão 4848).

25.2. A aplicação não executa no glassfish 4.x, apenas no 3.x

Ao executar o projeto usando o glassfish 4.x, é lançado um **NullPointerException** ao tentar fazer alguma operação com o banco de dados.

Solução

Adicionar a seguinte propriedade no **persistence.xml**:

```
<property name="hibernate.transaction.jta.platform"
          value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform" />
```

Como o glassfish 4, segue a especificação Java EE 7, é interessante também atualizar as dependências do hibernate no **pom.xml** (no projeto base padrão é a 4.0.0), para que ele já fique preparado para a especificação JPA 2.1:

hibernate-core:

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>4.3.5.Final</version>
</dependency>
```

hibernate-entitymanager:

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>4.3.5.Final</version>
</dependency>
```

25.3 Usando o SQL Server o LazyDataModelImpl não faz a paginação e lança um erro

Ao tentar fazer uma paginação em um dataTable que utiliza o LazyDataModelImpl, ocorre a seguinte exceção:

javax.persistence.PersistenceException: org.hibernate.exception.GenericJDBCException: ResultSet may only be accessed in a forward direction.

Solução

Inserir a seguinte configuração no **persistence.xml**:

```
<property name="hibernate.jdbc.use_scrollable_resultset" value="false" />
```

25.4 Ao fazer login no projeto do archetype a tela aparece em branco e um erro é lançado no glassfish 4

Esse erro acontece devido a expressão **`#{now}`** do omnifaces, que pega a data atual. Esse bug acontece apenas no glassfish 4, na versão 4.1 esse problema já foi corrigido. Este bug é relatado na página no glassfish:

<https://java.net/jira/browse/GLASSFISH-20775>

Solução

Atualizar o glassfish, caso não seja possível, deve-se remover a expressão **`#{now}`** da página {páginas web}/template/mainTemplate.xml.

26. FAQ

1 Qual versão do primefaces usar?

Essa é uma decisão do próprio desenvolvedor, o xpert-framework suporta as versões 3, 4 e 5 do primefaces.

2 A dependência do xpert-framework do primefaces é diferente da que eu quero usar, como fazer?

O xpert-framework importa o primefaces apenas para compilação do código, a versão considerada no seu projeto é que você importar, assim no xpert-framework pode ser a 4.0 e no seu projeto ser a 3.5, não tem problema, a versão final será a 3.5.

3 Não consigo compilar o meu projeto maven, pois ele não consegue baixar o xpert-framework, o que há de errado?

Até a presente versão, o framework ainda não está no maven central, assim talvez seja necessário instalar manualmente o **xpert-framework.jar** no seu repositório.

4 Encontrei um bug no framework, como corrigir?

Cadastre o bug no google codes através do link <https://code.google.com/p/xpert-framework/issues/list>, é interessante que seja explicado o que aconteceu, e se possível, com trechos de código, assim facilita as correções.

5 MySQL, SQL Sever, Oracle, PostgreSQL, qual é suportado pelo xpert-framework?

Qualquer banco é suportado, desde que ele possua um driver JDBC, e seja compatível a especificação do JTA. O projeto criado do arquétipo é voltado para o postgres, mas nada impede que seja configurado para outro banco.

6 É possível utilizar o eclipse link no lugar do hibernate?

Até a versão atual, apenas o hibernate é suportado.

7 Já tenho um banco criado, porém não tenho o mapeamento JPA do modelo, o xpert-framework vai gerar?

Não. Essa geração deve ser feita através de ferramentas de engenharia reversa, como as próprias IDEs (O Netbeans por exemplo, possui uma ferramenta de engenharia reversa).

8 No primefaces 5.x o dataTable está com um layout estranho, é por causa do xpert-framework?

Não. Na versão 5.0 do primefaces o CSS da tabela foi mudado para “table-layout: fixed”, isso pode causar algumas quebras na tabela, principalmente se utilizar o bootstrap. A solução é sobrescrever essa propriedade via CSS. Exemplo:

```
.ui-datatable table{  
    table-layout: auto;  
}
```


9 Como entrar em contato com a Xpert Sistemas?

Entre em contato através do site <http://www.xpertsistemas.com.br/>