Caixa Econômica Federal

**SIAOI - Sistema de Administração de Operações Imobiliárias**

*FRAMEWORK SIAOI*

Sumário

[1 Framework 3](#_Toc508302983)

[1.1 API de acesso a dados 3](#_Toc508302984)

[1.1.1 Principais classes 6](#_Toc508302985)

[1.1.2 Exemplo de utilização 7](#_Toc508302986)

[1.1.3 Utilizando entidade 8](#_Toc508302987)

[1.2 API de configuração 10](#_Toc508302988)

[1.2.1 Como utilizar a API de configuração 10](#_Toc508302989)

[1.2.2 Localização das configurações 11](#_Toc508302990)

[1.2.3 Sobreposição das configurações 11](#_Toc508302991)

[1.2.4 Observações importantes 11](#_Toc508302992)

[1.2.5 Exemplo de utilização 11](#_Toc508302993)

[1.2.6 Recuperação das configurações 12](#_Toc508302994)

[1.2.7 Sobreposição de Configurações 12](#_Toc508302995)

[1.3 API de relatórios 13](#_Toc508302996)

[1.3.1 Instrução de acesso aos dados: xxxSQL-sql.xml 13](#_Toc508302997)

[1.3.2 Serviço Java: RelatorioXxxService.java 13](#_Toc508302998)

[1.3.3 Estrutura básica da classe de serviço de relatório (MODELO NOVO) 13](#_Toc508302999)

[1.3.4 Estrutura básica da classe de serviço de relatório (MODELO ANTIGO) 14](#_Toc508303000)

[1.3.5 Trecho de JavaScript exemplo: relatorioXxx.conteudo.js 15](#_Toc508303001)

[2 Tratamento de Exceções 16](#_Toc508303002)

[3 Camada de apresentação 16](#_Toc508303003)

[3.1 Exemplo 16](#_Toc508303004)

[4 Camada de negócio 17](#_Toc508303005)

[4.1 Exemplo 17](#_Toc508303006)

[4.2 Principais anotações 17](#_Toc508303007)

[5 Comunicação entre as camadas 17](#_Toc508303008)

[5.1 Exemplo de uso 17](#_Toc508303009)

[5.1.1 Código JavaScript 17](#_Toc508303010)

[5.1.2 Mensagem XML 18](#_Toc508303011)

[5.1.3 Classe de serviço 18](#_Toc508303012)

[5.1.4 Retornando valores através do response 19](#_Toc508303013)

[6 Páginas Velocity 20](#_Toc508303014)

[6.1 Template Velocity 20](#_Toc508303015)

# Framework

O Vivere Framework foi desenvolvido para dar suporte ao desenvolvedor de negócios. Nos seus componentes são concentrados todos os requisitos não funcionais da aplicação. Os components do Framework são distribuídos em cinco unidades de deployment no formato jar, e são eles:

* **siaoi-framework.shared** – possui os objetos compartilhados do sistema, tais como interfaces e DTO/VO’s;
* **siaoi-framework.tools** – componente que concentra todos os utilitários do sistema, tais como formatadores e conversores;
* **siaoi-framework.core** – coração do framework. Possui a implementação da maior parte das interfaces e dos contextos de apoio ao negócio;
* **siaoi-framework.container** – componente que implementa o container de serviços, responsável por criar e gerenciar a execução dos serviços de negócio;
* **siaoi-framework.controller** – implementação do controller web;

## API de acesso a dados

API que lida com o acesso a dados, atualmente pode ser feito através de XML ou entidades mapeadas. O acesso via XML concentra todas as consultas em um arquivo XML onde as consultas são nomeadas.

O arquivo XML que armazena as consulta segue a seguinte nomenclatura:

* Conter o sufixo –sql.xml;
* Deve ser salvo na pasta siaoi-servicos/src/main/queries/db2.

O documento XML deve respeitar o seguinte formato:

<?xml version="1.0" encoding="iso-8859-1"?>

<root>

<sql type="{select|insert|update|delete}" name="{nome da operação}" [convertable="{true|false}"] [audited=”{false|true}”] >

<query [complete="{false|true}"] >

<![CDATA[ corpo da consulta ]]>

</query>

[

<return>

<column name="{nome coluna1}" type="{string, int, decimal, date}"/>

<column name="{nome coluna2}" type="{string, int, decimal, date}"/>

</return>

<restriction name="{nome da restrição-0}">

<![CDATA[ corpo da restrição, ( WHERE) ]]>

</restriction>

<restriction name="{nome da restrição-n}">

<![CDATA[ corpo da restrição, ( WHERE) ]]>

</restriction>

<composite name=”{nome grupo-0}” [type=”{and|or}”]>

<restriction name="{nome da restrição-1}">

<![CDATA[ corpo da restrição ]]>

</restriction>

<restriction name="{nome da restrição-2}">

<![CDATA[ corpo da restrição ]]>

</restriction>

<order name="{nome da ordenacao-2}" [by="{coluna1, coluna2}"]/>

<order name="{nome da ordenacao-3}" [by="{coluna1, coluna2}"] [optional="{true|false}"]/>

</composite>

<order name="{nome da ordenacao-n}" [by="{coluna1, coluna2}"] applyTo="{query, {nome da restrição-0}, {nome da restrição-n}, {nome do grupo-0} }" [optional="{true|false}"] [type="{asc|desc}"]>

<![CDATA[ coluna1, coluna2 ]]>

</order>

<executionProcess class="{FQCN}" type="{pre|post}"/>

]

</sql>

</root>

Onde:

[] – indica conteúdo opcional

{} – indica valor que deve ser especificado pelo desenvolvedor

| - Separador de opções de valores válidos

Descrição dos elementos:

|  |  |
| --- | --- |
| **Elemento** | sql |
| **Número de ocorrências** | 1..\* |
| **Função** | Representa um acesso a dados |
| **Atributos** | ***type*** - Identifica o tipo de acesso que será feito, o valor dever ser uma das opções : select, insert, update, delete  ***convertable*** - Identifica se este acesso deve ser convertido caso um sistema de conversão esteja sendo utilizado.  Valores : true – se deve ser convertido ou false – se não deve. O padrão é true  ***audited*** - Identifica se este acesso deve ser auditado. Se sim em toda execução deste acesso será necessário passar o usuário e identificação da organização.  Valores : true ou false. O padrão é false |

|  |  |
| --- | --- |
| **Elemento** | query |
| **Número de ocorrências** | 1 |
| **Função** | Corpo principal do acesso |
| **Atributos** | ***complete*** - Identifica se o corpo representa um acesso completo que é utilizado independente de outras partes (restriction ou order).  Valores: true – é completo ou false – não é completo. O padrão é false. |

|  |  |
| --- | --- |
| **Elemento** | return |
| **Número de ocorrências** | 0..1 |
| **Função** | Corpo principal do acesso |
| **Atributos** | Lista o tipo de dados esperados pela aplicação para cada coluna do resultado. |

|  |  |
| --- | --- |
| **Elemento** | column |
| **Número de ocorrências** | 1..\* |
| **Função** | Meta informação sobre uma coluna do resultado |
| **Atributos** | ***name*** - O nome da coluna que este elemento descreve.  ***type*** - Tipo de dados esperado pela aplicação. Os valores possíveis são : string, int, decimal, date and timestamp |

|  |  |
| --- | --- |
| **Elemento** | restriction |
| **Número de ocorrências** | 0..\* |
| **Função** | Especifica uma restrição (where, having e etc) que pode ser aplicada a definição base do acesso. |
| **Atributos** | ***name*** - Nome que identifica esta restrição. Tem que ser único dentro da definição de um acesso. |

|  |  |
| --- | --- |
| **Elemento** | order |
| **Número de ocorrências** | 0..\* |
| **Função** | Especifica uma lista, separada por virgulas, de nome de colunas que devem ser utilizadas para ordenar o resultado. |
| **Atributos** | ***name*** - Nome que identifica esta ordenação. Tem que ser único dentro da definição de um acesso.  ***by*** - Lista separada por virgulas de campos que devem ser utilizados na ordenação. Este atributo é opcional pois a mesma informação pode estar contida no corpo do elemento.  ***applyTo*** - Lista, separada por virgulas, com o nome dos elementos para qual esta ordenação se aplica. Como elementos entende-se “query” para especificar que esta ordenação se aplica a definição base do acesso, ou o nome de uma restrição para especificar que esta ordenação pode ser aplicada em conjunto com a restrição.  ***optional*** - Indica se a utilização desta ordenação pode ser opcional. Se o valor deste atributo for false, então esta ordenação será utilizada sempre que um dos elementos do atributo applyTo for utilizado. Valores válidos : true ou false. O valor padrão é true.  ***type*** - Indica o tipo de ordenação a ser utilizada. Valores válidos são: asc – para ordenação ascendente ou desc – para ordenação descendente; |

|  |  |
| --- | --- |
| **Elemento** | composite |
| **Número de ocorrências** | 0..\* |
| **Função** | Especifica um agrupamento de Restrictions e Orders que podem ser combinados e aplicados em conjunto. |
| **Atributos** | ***name*** - Nome que identifica este agrupamento. Tem que  ser único dentro da definição de um acesso e não pode ser  igual aos nomes das restrições.  ***type*** - Indica o tipo de relacionamento entre as restrições.  Valores válidos são : and - quando mais de uma restrição é  utilizada elas são encadeadas usando a instrução 'and', or –  o encadeamento é feito usando a instrução 'or'.  O valor padrão é and |

|  |  |
| --- | --- |
| **Elemento** | executionProcess |
| **Número de ocorrências** | 0..\* |
| **Função** | Especifica uma implementação de QueryExecutionProcess que deve ser inserido na corrente de execução deste acesso. |
| **Atributos** | ***class*** - Nome totalmente qualificado da implementação a  ser utilizada.  ***type*** - Especifica qual o tipo de inserção que deve ser feita  na corrente de execução. Os valores válidos são:  pre – a inserção deve ser feita no topo da corrente de  execução ou post – a inserção deve ser feita no final da  corrente de execução. |

### Principais classes

* StatementSql
  + Representa uma acesso a ser executado no banco de dados.
  + O acesso pode ser uma consulta ou nome de uma consulta externa;
  + Armazena os valores a serem utilizados durante a execução do acesso.
* ResultSetSql
  + Representa o resultado gerado por um acesso ao banco de dados;
  + Contém métodos para obter dados convertidos;
  + Mantém estado da paginação;
  + Acesso ao meta-dados do resultado.
* QueryMatcher
  + Representa um comando SQL;
  + Prepara o comando para ser executado no BD, substituindo parâmetros nomeados pelo caractere ‘?’;
  + Converte os parâmetros para o tipo esperado pelo acesso.
* ModelsController
  + Repositório central de acesso ao BD. É o responsável por criar instancias de QueryMatcher;
  + Se o parâmetro for o nome de um acesso, esta classe lê o acesso a partir do arquivo XML e retorna a instância de QueryMatcher.
* SqlQueryExecutor
  + Classe que de fato manipula os objetos JDBC para acesso ao BD;
  + Efetua todo o procedimento necessário para execução do comando SQL e BD.

Para utilizar os acessos definidos nos XML basta obter um StatementSql usando o seguinte padrão de nomenclatura:

* Utilizando a interface ServiceExecutionContext (recomendado):
  + context.getDataContext().prepareStatementSql("avaliacao.select.dadosAvaliacao");
* Utilizando a interface IServiceSession(fonte legado);
  + this.session.prepareStatementSQL("situacaoFGTS.select.faseAtiva");

A implementação das interfaces acima são carregadas pelo Spring através da anotação ServiceContext pacote br.com.viverebrasil.framework.container.annotations.

### Exemplo de utilização

#### Recuperando dados

@ProcessMethod

public void testeQuery(){

StatementSql stm =

this.context.getDataContext()

.prepareStatementSql("nome\_do\_arquivo.select.nome\_da\_query")

.set("nome\_da\_propriedade\_bind", 1L)

.execute();

if(stm.hasResult()){

//Obter valores do retorno da query

for(ResultSetSql result : stm.getResult()){

Long valorNumerico = result.getLong("nr\_qualquer\_coisa");

String valorString = result.getString("ds\_qualquer\_coisa");

}

}

}

O mesmo procedimento é utilizado para as queries de update, insert e delete, mudando apenas o tipo da ação no descritivo da query, e também o nome da query caso seja diferente.

#### Alterando dados

@ProcessMethod

public void testeQuery(){

//Update

this.context.getDataContext()

.prepareStatementSql("nome\_do\_arquivo.update.nome\_da\_query")

.set("nome\_da\_propriedade\_bind\_valor", "valor a ser alterado")

.set("nome\_da\_propriedade\_bind\_clausula\_where", 10L)

.execute();

//Insert

this.context.getDataContext()

.prepareStatementSql("nome\_do\_arquivo.insert.nome\_da\_query")

.set("nome\_da\_propriedade\_bind", "Nome do fulano")

.set("nome\_da\_propriedade\_bind", new Date())

.set("nome\_da\_propriedade\_bind", 1L)

.set("nome\_da\_propriedade\_bind", 0.0d)

.execute();

//Delete

this.context.getDataContext()

.prepareStatementSql("nome\_do\_arquivo.delete.nome\_da\_query")

.set("nome\_da\_propriedade\_bind\_clausula\_where", 10L)

.execute();

}

### Utilizando entidade

A entidade é a representação de uma tabela do banco de dados através de uma classe do tipo PersistentObject. Além disso utilizamos algumas anotações para defini-la.

#### Exemplo de utilização

package com.viverebrasil.commons.entidades.pessoa;

import com.viverelib.core.persistence.annotations.Column;

import com.viverelib.core.persistence.annotations.Entity;

import com.viverelib.core.persistence.annotations.sources.Table;

import com.viverelib.core.persistence.annotations.types.Expression;

import com.viverelib.core.persistence.annotations.types.Id;

import com.viverelib.core.persistence.enums.ColumnType;

import com.viverelib.core.persistence.types.PersistentObject;

@Entity

@Table( name = "prt\_pessoa" )

public class PessoaEntidade extends PersistentObject {

/\*\*

\* @column id\_cliente

\*/

@Id( sequence = "seq\_prt\_pessoa" )

@Column( name = "id\_cliente", type = ColumnType.LONG, nullable = false )

public long id;

/\*\*

\* @column cd\_cliente

\*/

@Column( name = "cd\_cliente", type = ColumnType.LONG, nullable = false )

public long codigoCliente;

/\*\*

\* @column nm\_cliente

\*/

@Column( name = "nm\_cliente", length = 100 )

public String nomeCliente;

}

#### Principais anotações:

* @Entity
* @Table
* @Column
* @Id
* @ForeignKey
* @JoinColumn
* @JoinColumns
* @Transient

#### Persistindo objetos

##### Utilizando a interface IServiceSession

session.save(minhaEntidade); //Persiste uma entidade no banco de dados

session.findALL(MinhaEntidade.class); //Recupera todas os dados da tabela mapeada

session.delete(MinhaEntidade.class, meuFiltro); //Remove uma entidade

session.findByID(MinhaEntidade.class, id); //Recupera uma entidade persistida no banco de dados

session.findByFilter(MinhaEntidade.class, filtro); //Recupera uma lista de entidades, o filtro é um instancia de FilterSQL

#### Utilizando a interface ServiceExecutionContext

context.getDataContext().save(); //ver session.save

context.getDataContext().findALL(); //ver session.findAll

context.getDataContext().delete(); //ver session.delete

context.getDataContext().findByID(); //ver session.findByID

context.getDataContext().findByFilter(); //ver session.findByFilter

## API de configuração

A API de configuração centraliza as operações para recuperação de parâmetros de configuração do sistema. As configurações do sistema vêm de duas fontes:

* Configuração de Ambiente: Configurações do ambiente de execução, cujas informações se referem à instalação do ambiente, e não têm relação com informações de negócio da aplicação (tais como ID de organização).
* Configuração da Aplicação: essas configurações representam parâmetros de configuração da aplicação. Exemplos podem ser o parâmetro que ativa a obrigatoriedade, debug, etc.

### Como utilizar a API de configuração

A classe central da API de Configuração é a classe

***br.com.viverebrasil.framework.config.ConfigService.***

Para recuperar uma instancia desse serviço, é necessário fazer a injeção via anotação. Esse procedimento consiste em se criar um atributo de classe do tipo ConfigService, e anotá-lo com a anotação ***@SupportService***.

De posse do objeto ConfigService, utilize um dos métodos utilitários para a recuperação dos valores tipados. A recuperação desses parâmetros deve ser contextualizada de acordo com o ID de Organização corrente, o que geralmente é determinado no Login do Usuário. Os serviços não autenticados deverão recuperar o ID da organização de outra fonte.

Os métodos disponíveis estão listados abaixo. Todos eles possuem duas assinaturas, uma que recebe o ID da organização e outra sem:

* ***getString(String key, Long id)***: recuperam o valor de configuração sob a chave no formato String;
* ***getDouble(String key, Long id)***: recuperam o valor de configuração sob a chave no formato Double;
* ***getInt(String key, Long id)***: recuperam o valor de configuração sob a chave no formato Integer;
* ***getLong(String key, Long id)***: recuperam o valor de configuração sob a chave no formato Long;
* ***getBoolean(String key, Long id)***: recuperam o valor de configuração sob a chave no formato Boolean. Os valores reconhecidos como true são: "true", "y", "s" e "1", independente se maiúsculo ou minusculo onde aplicável;
* ***getDate(String key, Long id)***: recuperam o valor de configuração sob a chave no formato java.util.Date. O formato das datas depende da configuração do atributo 'vivere.default.date.format'. Vide Chaves de configuração da API para maiores detalhes;

Para maiores informações sobre os métodos utilitários, consulte o Javadoc da classe ConfigService.

### Localização das configurações

A configuração do ambiente deverá ficar armazenada num arquivo .properties dentro do diretório conf/ do JBoss (para outros servidores de aplicação, consultar documentação específica).

O nome do arquivo de configuração é determinado pela propriedade vivere.env.config, cujo valor padrao é vivere-env, ou seja, caso não seja especificado no momento em que o servidor esteja sendo iniciado, a API irá buscar pelo arquivo vivere-env.properties.

Após carregar o arquivo de configuração do ambiente, a API irá se conectar ao Banco de Dados para recuperar os dados de configuração da aplicação que residem na tabela SIS\_PROFILE.

### Sobreposição das configurações

Em tempo de desenvolvimento, algumas vezes é necessário que determinadas configurações sejam aplicadas apenas em um ambiente de desenvolvimento específico, tal como no caso da máquina do desenvolvedor. Suponha que você esteja aplicando uma manutenção no recurso da obrigatoriedade, que depende de um parâmetro de configuração, mas que se for modificado no banco de dados do ambiente DSV, impactará não só a máquina dos outros desenvolvedores como o próprio ambiente DSV. Nesse caso é necessária a sobreposição de configurações, de modo que apenas a sua máquina contenha determinado parâmetro.

A sobreposição consiste em incluir no arquivo de configurações de ambiente (.properties) o parâmetro desejado. A API, por sua vez, SEMPRE busca os parâmetros primeiro nas configurações do ambiente, depois nas configurações da aplicação.

Interpolação

A API possui o recurso de interpolação, que é a possibilidade de se utilizar uma chave de configuração como parte de outra configuração. Para realizar a interpolação, utilize a sequencia parametro1=${<chave do parametro2>} para "injetar" o valor de "parametro2" no valor de "parametro1".

### Observações importantes

Não é possível fazer interpolação entre contextos, ou seja, se determinada chave está no contexto do ambiente, ele nao pode ser utilizado no contexto da aplicação, e

vice-versa.

A interpolação leva em conta o ID da organização. Isso significa que caso a API nao encontre a chave a ser interpolada com o mesmo ID de organização da chave que está sendo chamada, a substituição não será feita.

### Exemplo de utilização

@Service("CONFIG:TestConfig")

public class TestConfigService {

@ServiceContext

private ServiceContext serviceContext;

public void execute(){

ConfigContext conf = this.serviceContext.getConfigContext();

}

}

### Recuperação das configurações

Recupera o atributo 'proposta.idade.max' da organização de ID 3:

Integer idadeMax = conf.getInt("proposta.idade.max", 3L);

Recupera o atributo de obrigatoriedade para a organização de ID 5;

Boolean obrigatorio = conf.getBoolean("vivere.app.obrigatorio", 5L);

Recupera a data de inicio de vigência do scheduler sem levar em consideração o ID organização. Para que essa recuperação tenha êxito, uma das configurações possíveis seria a seguinte:

vivere.default.date.format=dd/MM/yyyy

app.inicio.scheduler=25/12/2010

Nesse caso, o código abaixo devolveria um objeto do tipo java.util.Date apontando para 25 de Dezembro de 2010:

java.util.Date sched = conf.getDate("app.inicio.scheduler");

### Sobreposição de Configurações

Suponha o seguinte arquivo de configuração de ambiente e em seguida a tabela ***SIS\_PROFILE***:

application.server.ip=192.168.10.221

application.root.path=/source/viverebrasil/dsv/app\_jboss

|  |  |  |
| --- | --- | --- |
| CD\_PROFILE | DS\_VALOR\_DEFAULT | ID\_ORGANIZACAO |
| app.obrigatorio | false | 2 |
| app.obrigatorio | true | 3 |

Existe a possibilidade de determinado desenvolvedor precisar ajustar algo no sistema de obrigatoriedade, cujas configurações estão no contexto da aplicação. Se a alteração for feita diretamente no banco de dados, todos os ambientes "pendurados" nesse banco sofrerão a alteração. Nesse caso, se aplica a sobreposição de configurações, incluindo o parâmetro desejado no contexto de ambiente da máquina do desenvolvedor:

application.server.ip=192.168.10.221

application.root.path=/source/viverebrasil/dsv/app\_jboss

app.obrigatorio.3=false

A partir de agora, quando o código ***config.getBoolean("app.obrigatorio", 3L);*** for chamado, a API irá buscar pela chave "app.obrigatorio" concatenado com "." mais o id da organização e, encontrando essa chave no contexto do ambiente, ele nao fará a busca no contexto da aplicação.

Perceba que se a chamada for ***config.getBoolean("app.obrigatorio",2L);***, a API apenas a encontrará no contexto da aplicação, ou ainda se for config.getBoolean("app.obrigatorio");, a chave não será encontrada.

## API de relatórios

A API de relatório é uma pequena camada de abstração do JasperReport.

A estrutura de um relatório é composta de quatro arquivos:

* ***xxxSQL-sql.xml*** - arquivo XML que descreve a instrução de acesso aos dados.
* ***RelatorioXxxService.java*** - serviço java que prepara a instrução de acesso aos dados e solicita a geração do relatório.
* ***relatorioXxx.conteudo.js*** - javascript para chamada do serviço de geração de relatórios.
* ***relatorioXxx.jrxml*** - arquivo com o template feito pelo iReport.

### Instrução de acesso aos dados: xxxSQL-sql.xml

Arquivo XML que descreve a instrução a ser executada no banco de dados, deve respeitar a API de acesso a dados.

### Serviço Java: RelatorioXxxService.java

Classe Java com a responsabilidade de:

* Preparar a instrução de acesso aos dados (bind entre os valores e os parâmetros do arquivo que descreve a instrução).
* Delegar a construção do relatório para o framework Vivere.
* Adicionar o caminho do relatório na sessão.

### Estrutura básica da classe de serviço de relatório (MODELO NOVO)

@Service( "BACKOFFICE:RELATORIO:Xxx" ) //este valor esta associado ao arquivo HTML que chama o servico

public class RelatorioXxxService {

@ServiceContext

private ServiceExecutionContext context;

@ProcessMethod

public void gerar() {

//lê valores da sessão

DataGroup dataGroup = context.getParameter("ABC");

Long organizacaoID = dataGroup.get("organizationId").asLong();

//Formato de saída: a String reportFormat deve ser em maiúsculas respeitando os valores da Enum ReportFormat (TEXT, CSV, HTML, PDF)

String format = dataGroup.get("reportFormat").asString();

//bind dos valores e parâmetros que descrevem a instrução de acesso aos dados

StatementSql stmt = this.context.getDataContext().prepareStatementSql("xxxSQL.yyy.zzz");

stmt.set("id\_organizacao", organizacaoID);

//obtém o contexto de relatórios da sessão

ReportContext reportContext = this.context.getReportContext();

//prepara o relatório utilizando a dependência com vivere-framework

Report report = reportContext.createReport("Xxx")

.set("someExtraParam", "someExtraValue") //valores adicionais disponíveis para o relatório

.with(stmt) // \_PreparedStatement\_ utilizado para obter os dados do relatório

.as(ReportFormat.valueOf(format));

//constrói o relatório

File reportFile = reportContext.build(report.getReportConfiguration());

//adicona o caminho do relatório gerado a sessão

this.context.getResult().getWriter().addGroup("reports").set("ds\_caminho", reportFile);

}

}

### Estrutura básica da classe de serviço de relatório (MODELO ANTIGO)

@Service( "BACKOFFICE:RELATORIO:Xxx" ) //este valor esta associado ao arquivo HTML que chama o servico

public class RelatorioXxxService {

@ServiceContext

private IServiceSession session;

@ProcessMethod

public void gerar() {

//lê valores da sessão

long organizacaoID = this.session.getOrganizationID();

//Formato de saída: a String reportFormat deve ser em maiúsculas respeitando os valores da Enum ReportFormat (TEXT, CSV, HTML, PDF)

String format = this.session.getString("reportFormat");

//bind dos valores e parâmetros que descrevem a instrução de acesso aos dados

StatementSql stmt = this.session.prepareStatementSQL( "xxxSQL.yyy.zzz" );

stmt.set("id\_organizacao", organizacaoID);

//obtém o contexto de relatórios da sessão

ReportContext reportContext = this.session.getReportContext();

//prepara o relatório utilizando a dependência com vivere-framework

Report report = reportContext.createReport("Xxx")

.set("someExtraParam", "someExtraValue") //valores adicionais disponíveis para o relatório

.with(stmt) // \_PreparedStatement\_ utilizado para obter os dados do relatório

.as(ReportFormat.valueOf(format));

//constrói o relatório

File reportFile = reportContext.build(report.getReportConfiguration());

//adicona o caminho do relatório gerado a sessão

this.session.getResult().group("relatorio").set("ds\_caminho", reportFile);

}

}

### Trecho de JavaScript exemplo: relatorioXxx.conteudo.js

function someFunctionName() {

var dataSource = vivereLIB.dataSource("BACKOFFICE:RELATORIO:Xxx->gerar")

.group("ABC")

.set("organizationId", 2)

.set("format", "PDF")

dataSource.open();

if( service.hasError() ){

//TODO: Aqui o tratamento de erros

}

};

# Tratamento de Exceções

O sistema possui uma árvore de exceções que inicia com duas exceções principais:

* VivereCheckedException – exceção que requer tratamento com try/catch;
* VivereUncheckedException – exceção que não requer tratamento;
* VivereSystemException – exceção pai de todas as exceções do core do sistema;
* VivereIntegrationException – exceção lançada em métodos que implementam classes de contexto de serviço (EAI)

Em caso de falha, uma exceção é lançada pelo código e é interceptada na fronteira entre a camada de negócio e a camada de controle. O tratamento dos erros previstos e realizado pela classe VivereInformationException que é responsável por transmitir a mensagem para a tela do usuário. No tratamento de erros previstos, a transação não sofre rollback. No tratamento de erros imprevistos, o sistema realiza o rollback da transação garantindo o último estado consistente.

Não existe impedimento no tratamento de exceções dentro do serviço, de modo que, conforme a necessidade local, o uso to try…catch é permitido.

# Camada de apresentação

A camada de apresentação não possui páginas que rodam do lado do servidor, as páginas são construídas em HTML com JavaScript. Páginas estáticas em HTML fica com o proposito de armazenar informações e o JavaScript controla todos os eventos da página.

Para cada funcionalidade, cada tela existe ao menos um arquivo JavaScript associado, e o padrão de nomenclatura destes arquivos deve seguir o mesmo padrão adotado para o arquivo HTML correspondente.

## Exemplo

Foi solicitado a criação de um nova página para negociar contratos em atraso.

Será criado no diretório negociar um arquivo chamado negociarContratos.html e um arquivo js/negociarContratos.js.

siaoi-web/

webapps/

negociar/

negociarContratos.html

js/

negociarContratos.js

O nome do arquivo e diretório é definido em caixa baixa e sem caractere especial.

Em caso de nome composto não existe separação por espaço e a primeira letra da

segunda palavra fica em caixa alta.

# Camada de negócio

A Camada de negócio - composta por um conjunto de regras negociais, que somada ao framework disponibiliza uma camada de serviços que pode ser consumida pela apresentação ou por outro serviço.

Para chamadas de serviços o framework vivere utiliza o Spring MVC como framework, sendo assim, para criarmos um serviço basta criarmos uma classe em um pacote dentro da estrutura com.viverebrasil.applications.services e anotá-la com a anotação (Annotation) @Service.

## Exemplo

package br.com.viverebrasil.servico.hello;

….

@Service("POC:HelloWorld")

@NoAuthenticationRequired

public class HelloWorld{

@ServiceContext

private ServiceExecutionContext context;

@ProcessMethod

public void sayHello(){

}

}

## Principais anotações

* ***@ServiceContext*** – Anotação do projeto vivere-framework.container, que possui a responsabilidade de preparar a propriedade ServiceExecutionContext.
* ***@ProcessMethod*** – marca os método que podem ser executados pela camada de apresentação.
* **@NoAuthenticationRequired** – desabilita para o filtro de autenticação

# Comunicação entre as camadas

A comunicação entre a camada de apresentação e a camada de serviço é feita através de XML, o módulo do framework que lida com a comunicação é o siaoi-framework.

A requisição de uma classe de negocio é feita em AJAX, utilizando a biblioteca vivereLIB, a biblioteca vivereLIB utiliza o framework jQuery para executar requisições de HTTP GET e HTTP POST.

## Exemplo de uso

### Código JavaScript

var srv = vivereLIB.dataSource("AUTH:login").group("user")

.set("login", "root")

.set("passwd", "root");

srv.open();

O trecho de código acima carrega o serviço de nome AUTH:login. Quando o trecho srv.open() é executado é gerado uma requisição para o controlador callService.do que recebe o nome do serviço e os parâmetros para execução, com base no script acima segue o XML gerado.

### Mensagem XML

<?xml version="1.0" ?>

<request>

<user type="group">

<login>root</login>

<passwd>root</passwd>

</user>

</request>

### Classe de serviço

//Arquivo LoginService.java

package br.com.viverebrasil.applications.services.analise;

import org.springframework.stereotype.Service;

import br.com.viverebrasil.framework.container.annotations.ProcessMethod;

import br.com.viverebrasil.framework.container.annotations.ServiceContext;

@Service("AUTH:login")

public class LoginService {

@ServiceContext private ServiceExecutionContext serviceContext;

@ProcessMethod

public void execute()

{

String login = serviceContext.getParameter("user")

.get("login").asString();

String senha = serviceContext.getParameter("user")

.get("passwd").asString();

}

}

### Retornando valores através do response

Assim como nos parâmetros recebidos em um serviço, o retorno também é feito via xml, no entanto há outra API responsável por escrever no response, é a interface Writer.

Writer writer = serviceContext.getResult().getWriter();

writer.addGroup("user");

writer.set("id", 10);

writer.set("name", "Administrador");

writer.set("email", "root@localhost");

Executando o código acima, em um método de serviço será disponibilizado a seguinte resposta em XML.

<response>

<user type="group">

<id>10</id>

<name>Administrador</name>

<email>root@localhost</email>

</user>

</response>

No exemplo anterior não foi especificado nenhum nome de método, quando isso ocorre o framework faz uma chamada ao método padrão (execute).

Para executar um método especifico basta utilizar o separador ->, conforme trecho abaixo.

var service = vivereLIB.dataSource("EMPRENDIMENTO:pesquisar->porCpf")

.group("cliente")

.set("cpf", "111.111.111-11");

service.open();

Note que no código acima foi adicionado ao nome do serviço o separador -> seguido do valor da palavra porCpf, esse parâmetro é o nome do método que será executado.

Assinatura do método porCPF:

Segue a assinatura do método porCpf.

@ProcessMethod

public void porCpf()

{

//PESQUISAR POR CPF

}

# Páginas Velocity

Velocity é um projeto Open Source mantido pela Apache Software Foundation, o Velocity é um template engine feito em Java. O Velocity é utilizado em algumas telas do sistema. Geralmente é utilizado para renderizar objetos ou até mesmo criar páginas dinâmicas.

## Template Velocity

Um template velocity é um arquivo ou texto com a extensão .vm, que contém textos, tags HTML e VTL (Velocity Template Language). Através da VTL é possível incorporar conteúdo dinâmico no arquivo de template.

Onde toda diretiva é precedida pelo character “#”, como por exemplo:

#set( $idCliente = $request.getParameter("idCliente") )

A executar o trecho de código acima, a diretiva set atribui o valor da variável de requisição idCliente na variável $idCliente do Velocity.

O Velocity possui outras diretivas que são utilizadas, tais como:

* #set()
* #if()
* #else
* #elseif
* #end
* #foreach
* #include
* #parse

O Velocity é utilizado em conjunto com o Spring Framework, no arquivo vivere-servlet.xml do projeto siaoi-web é definico o bean velocityResolver que tem por responsabilidade resolver as tags VTL e rederizar o conteúdo dinâmico da página.