we believe in...

Interação de Páginas em Aplicações IGRP web [métodos e parâmetros]

Cliente	NOSI – IGRP WEB
Referência	
Versão	1.00
Status	

CONTEÚDO

1 ENQUADRAMENTO

Um dos aspetos mais importantes na implementação de qualquer aplicação web é a interação entre as páginas, pois são elas os principais componentes da aplicação.

No caso do IGRP web essa interação baseia-se na chamada e resposta entre os diversos métodos presentes em cada uma delas. Obviamente, toda a chamada envolve a passagem de parâmetros, por forma a serem enviados dados fundamentais no processo de comunicação.

A partir disso pretendemos com este documento abordar a interação entre as diversas páginas (classes) e métodos, e as diversas formas de passagem e receção de passagem de parâmetros quando se desenvolve sobre a framework IGRP web.

2 COMPOSIÇÃO DE CADA PÁGINA

Antes de mais e para nos situarmos, vamos entender a composição e o funcionamento de cada página.

Cada página de aplicações desenvolvidas em IGRP web funciona sobre o modelo MCV e compreende três classes java (Model – View – Controler), componentes em XML, XSL, JavaScript e CSS. A forma como estes componentes interagem entre si se encontra descrita no documento [Funcionamento de aplicações e páginas no IGRP web], logo recomendamos a leitura desse manual.

No momento interessa-nos apenas realçar que o trabalho do programador é feito no **Controller**, classe que comanda a execução de cada página. Ao chamarmos uma página estamos a chamar um de seus métodos e a enviar os parâmetros necessários.

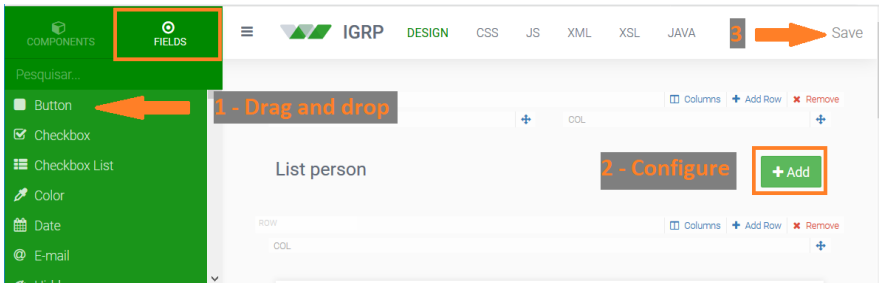
Por padrão ao chamarmos uma página é executado o método **actionIndex**, presente na classe Controller da mesma. Entretanto, a mesma pode conter, ou não, vários outros métodos. Para cada botão criado é criado um método correspondente onde devem ser programadas as ações desencadeadas ao se clicar nesse botão. Por exemplo, se eu crio um botão de nome **add**, automaticamente é criado um método **actionAdd**, que lhe corresponde.

3 CRIANDO E CONFIGURANDO BOTÕES

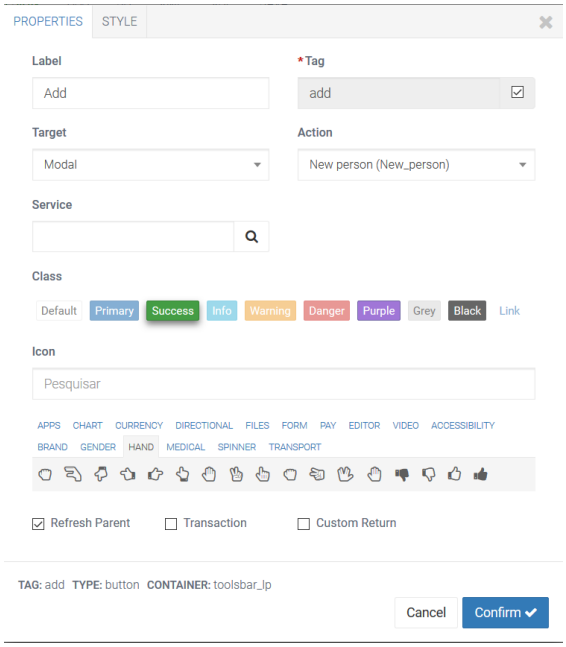
Os botões são criados no editor de página e a partir do momento de sua criação o método correspondente é criado, logo que se salve as alterações realizadas na página. Estando no IGRP Studio, acedemos o editor de páginas clicando sobre o botão  presente na linha de cada página. Para configurar um botão passe o ponteiro do rato por cima dele e click no botão  , que aparecerá, para que seja aberto um modal [Figura 2] para o efeito.

Commented [N/D-AM1]: Descrever como chegar ao editor da página

Commented [N/D-AM2]: Passar o ponteiro do rato por cima do quê? Na imagem não está ilustrado este passo



1 - Criação de botões



2 - Configuração de botões

3.1 Propriedades do botão

Entenda para que serve cada uma das propriedades do botão que podem ser configuradas nessa janela [Figura 2]:

- **Label**: Texto que aparece no botão no layout;
- **Tag**: nome do botão dentro do código – utilizado no nome do método correspondente;
- **Target**: o efeito que esperamos que o botão tenha (método e página invocados);
- **Action**: página chamada pelo botão;
- **Class** e **Icon**: Aparência do botão;
- **Refresh Parent**: disponível apenas quando a **action** selecionada for **modal** ou **submit modal** – atualiza a página ao ser fechado;
- **Transaction**: quando selecionado o acesso ao botão deve ser atribuído aos utilizadores ou às orgânicas pelo administrador no botão Access Management presente no menu ☰ do lado esquerdo do IGRP Stúdio;

- **Custom Return**: por defeito ao criarmos um botão, a linha do código de retorno do método correspondente é gerada. Quando selecionamos esta opção essa linha código deixa de ser criada pelo editor de página e deve ser feita pelo programador. Como poderá perceber a página não poderá ser compilada antes que se crie essa linha de código, uma vez que se espera um retorno do método, que não é **void**. Repare que ao "checkar" este campo, automaticamente fica indisponível o campo **action**, pois passa a ser obrigação do programador criar o código correspondente.

4 OS MÉTODOS DO CONTROLLER – CHAMADA E RETORNO

Como referimos anteriormente, cada Controller tem um método **actionIndex**, que, por defeito, é carregado, quando se chama a página – um método para cada botão criado. Estes são os métodos que representam ações. No entanto podem servir-se de outros métodos criados pelo programador para as mais diversas finalidades. Estes devem ficar no final do ficheiro num espaço reservado para este fim.

Na figura seguinte [Figura 3] podemos perceber a estrutura de uma classe Controller e os espaços reservados para programação. As restantes áreas do ficheiro não podem ser violadas.

```
package nosi.webapps.demo.pages.teste2;

import nosi.core.webapp.Controller;
import nosi.core.webapp.databse.helpers.ResultSet;
import nosi.core.webapp.databse.helpers.QueryInterface;
import java.io.IOException;
import nosi.core.webapp.Core;
import nosi.core.webapp.Response;
/*----#start-code(packages_import)----*/
                                        Imports
/*----#end-code----*/

public class Teste2Controller extends Controller {

    public Response actionIndex() throws IOException, IllegalArgumentException, IllegalAccessException{

        Teste2 model = new Teste2();
        model.load();
        Teste2View view = new Teste2View();
        /*----#start-code(index)----*/
                                        actionIndex code
/*----#end-code----*/
        view.setModel(model);
        return this.renderView(view);
    }

    public Response actionAdd() throws IOException, IllegalArgumentException, IllegalAccessException{

        Teste2 model = new Teste2();
        model.load();
        /*----#gen-example
          EXAMPLES COPY/PASTE:
          INFO: Core.query(null,... change 'null' to your db connection name added in application builder.
         this.addQueryString("p_id","12"); //to send a query string in the URL
         return this.forward("demo","Teste2","index", this.queryString()); //if submit, loads the values
          ----#gen-example */
        /*----#start-code(add)----*/
                                        actionAdd code
/*----#end-code----*/
        return this.redirect("demo","Teste2","index", this.queryString());
    }

    /*----#start-code(custom_actions)----*/
                                        other methods
/*----#end-code----*/
    }
```

3 - Estrutura da classe Controller

4.1 Chamada e retorno/redirecionamento

Por defeito, ao olharmos para o layout de uma página, estamos a ver o conteúdo que o respetivo **index** nos envia, através do código:

```
return this.renderView ( view );  // retorna um HttpServletResponseWrapper
```

Entretanto se olharmos para os outros métodos veremos que, depois de terem realizado seu "objetivo" retornam, ou seja, redirecionam a página a outros métodos, geralmente um index, para apresentar o resultado da operação realizada e oferecer interface para novas ações ao utilizador através do UI.

4.2 Redirect

```
return this.redirect("demo", "New_person", "index");
return this.redirect("demo", "New_person", "index", this.queryString() );
```

Método padrão de redirecionamento, utilizado para a maioria dos casos. Parâmetros:

1. Application
2. Page
3. Action (método a ser chamado)
4. ParamValues (lista de parâmetros)

4.3 Forward

```
return this.forward( "demo", "New_person", "index" );
return this.forward( "demo", "New_person", "index", this.queryString() );
```

Utilizado para que o formulário da página invocada herde o preenchimento do formulário que invoca. Muito útil para tentativas de guardar dados sem sucesso, para que o utilizador não tenha de preencher o formulário novamente com os mesmos dados.

4.4 Call

Utilizado em BPMN…

5 ENVIO DE PARÂMETROS

A seguir abordamos as formas de enviar e receber parâmetros. Note que quando invocamos um método de um ficheiro **Controller** diferente, para enviarmos parâmetros devemos incluir na lista dos métodos

redirect ou **forward** o quarto parâmetro (**this.queryString()**), que consiste na lista de parâmetros a serem enviados.

5.1 Método addQueryString

Este é o método padrão de passagem de parâmetros no IGRP e pode ser utilizado na maioria dos casos. Permite passar os mais variados tipos de valores, que podem ser recebidos por métodos listados mais a frente neste documento.

Envio
Integer personId = 11; this.addQueryString("p_personId", personId);
Recepção
Integer personId = Core.getParamInt("personId");

5.2 Método loadQueryString

Este método deve ser utilizado quando queremos enviar todos os dados do **Model** como parâmetros.

Envio
this.loadQueryString();
Recepção
Map <String, String[]> parameters = Core.getParameters();

Obs: Os parâmetros também podem ser recebidos individualmente, caso for conveniente ao programador.

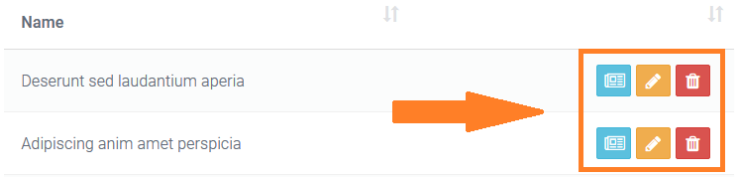
5.3 Associação de parâmetros a Botões

Existe a opção de associar parâmetros a botões. No caso do exemplo a seguir, considere a existência de um botão de nome **save**, responsável por enviar o parâmetro **lastName**. Preste atenção no parâmetro do método **setLInk**.

Envio
String lastName = "Messi"; view.btn_save.setLink("save&lastName="+ lastName);
Recepção
String lastName = Core.getParam("lastName"); // ou this.getQueryString("lastName");

5.4 Campos IsKey em tabelas

Quando temos botões em tabelas podemos enviar parâmetros através deles, correspondendo a valores de determinadas colunas nas respetivas linhas.

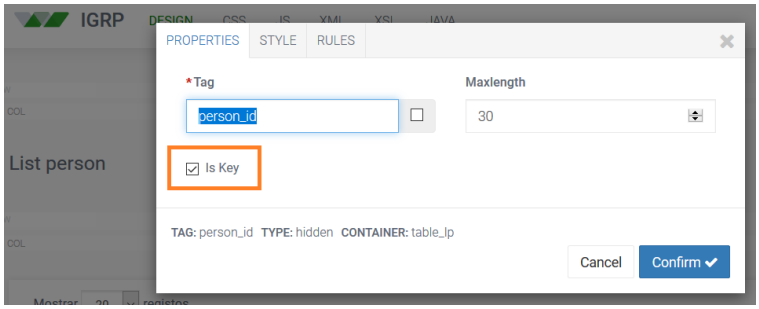


4 - Botões de tabela

Envio

Commented [N/D-AM3]: Parece estar desenquadra...

Configuramos os campos que queremos enviar enquanto **IsKey** [Figura 2]. No caso é adicionado no início do nome de cada parâmetro a ser enviado a String "p_". Um campo de nome **person_id**, por exemplo, é recebido com o nome "p_person_id".



5 - Configuração de coluna enquanto IsKey

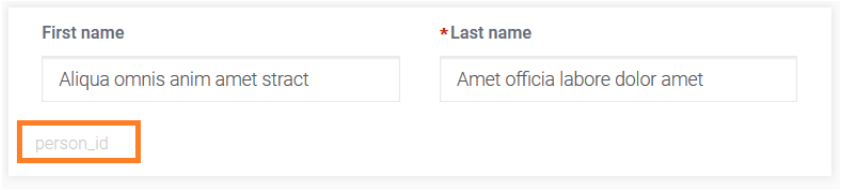
Repare que ao fazer isso no seu controller é acrescentada a linha de código seguinte:

```
view.person_id.setParam( true );
```

Recepção
String lastName = Core.getParam("p_person_id");

5.5 Campos hiden

Campos do tipo **hiden** também servem para enviar parâmetros, desde que o método que envia e o que recebe o parâmetro estejam no mesmo ficheiro **Controller**, pois o valor simplesmente é guardado no **Model** por um método e acedido pelo outro.



6 - Campo Hiden numa tabela

Envio
Integer personId = 11; model.setHiden_id("personId");
Recepção
Integer personId = model.getHiden_id(); // ou this.getQueryInteger("lastName");

5.6 Lista de métodos para receber parâmetros

Nas tabelas a seguir podemos ver os métodos utilizados para receber parâmetros, à exceção dos enviados através de campos hiden, que devem ser capturados no mesmo campo.

- Métodos do Core

Método	Retorno
Core.getParam("stringName");	String
Core.getParamArray("arrayName");	String[]
Core.getParamInt("integerName");	Integer
Core.getParamShort("shortName");	Short
Core.getParamLong("longName");	Long
Core.getParamFloat("floatName");	Float
Core.getParamDouble("doubleName");	Double
Core.getParameters();	Map < String, String[] >

- Métodos do Controller

Método	Retorno
this.getQueryString("stringName");	String
this.getQueryArray("arrayName");	String[]

this.getQueryStringInteger("integerName");	Integer
this.getQueryStringShort("shortName");	Short
this.getQueryStringLong("longName");	Long
this.getQueryStringFloat("floatName");	Float
this.getQueryStringDouble ("doubleName");	Double