



PROJETO CUSTOMIZAÇÕES

Construindo e consumindo uma API

Amanda Alvim

1. Objetivo

O objetivo dessa customização é demonstrar uma forma de criação da sua própria api que pode ser consumida via Integração de API, configuração essa do próprio produto, podemos consumir via ajax preenchendo um campo lista, e via integração utilizando formas do JAVA para consumir a api e retornar para um campo lista.

Como nesse caso vamos falar de 3 arquivos, um controller, chamado **ExemploApiController**, um JS, chamado **exemploChamadaControllerMontaLista.js** e uma integração **MontaCampoLista** todos eles estão no projeto do git (<http://git.lecom.com.br/PSP/Projeto-Base-BPM>) cada um no seu pacote/pasta.

2. Como usar

No momento de utilizar essa customização precisará mudar o fonte para se adequar ao que deseja fazer, lembrando que essa customização realmente é um exemplo de uso e não uma fórmula pronta para só utilizar em seu processo, você pode talvez utilizar pedaços dela para outros fins.

3. Controller

```
ExemploApiController.java
1 package com.lecom.workflow.cadastros.common.controller;
2
3 import java.io.IOException;
4
5 @WebServlet("/app/public/exemploApiController")
6 public class ExemploApiController extends ControllerServlet{
7     /**
8      *
9      */
10     private static final long serialVersionUID = 1L;
11
12     @SuppressWarnings("unchecked")
13     public void index(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
14         JSONObject retorno = new JSONObject();
15
16         try {
17             JSONArray collect = new JSONArray();
18
19             String sigla = request.getParameter("SIGLA");
20
21             if(sigla.equals("SP")) {
22                 JSONObject cid1 = new JSONObject();
23                 cid1.put("CIDADE", "Sorocaba");
24                 JSONObject cid2 = new JSONObject();
25                 cid2.put("CIDADE", "Agudos");
26                 JSONObject cid3 = new JSONObject();
27                 cid3.put("CIDADE", "Jatuna");
28                 JSONObject cid4 = new JSONObject();
29                 cid4.put("CIDADE", "Pederneiras");
30
31                 collect.add(cid1);
32                 collect.add(cid2);
33                 collect.add(cid3);
34                 collect.add(cid4);
35             } else if(sigla.equals("RJ")) {
36                 JSONObject cid1 = new JSONObject();
37                 cid1.put("CIDADE", "Rio de Janeiro");
38                 JSONObject cid2 = new JSONObject();
39                 cid2.put("CIDADE", "Campos");
40                 JSONObject cid3 = new JSONObject();
41                 cid3.put("CIDADE", "Niteroi");
42                 JSONObject cid4 = new JSONObject();
43                 cid4.put("CIDADE", "Resende");
44             }
45         } catch (Exception e) {
46             //
47         }
48     }
49 }
```

```
ExemploApiController.java
57         collect.add(cid1);
58         collect.add(cid2);
59         collect.add(cid3);
60         collect.add(cid4);
61         collect.add(cid5);
62     } else if (sigla.equals("MG")) {
63         JSONObject cid1 = new JSONObject();
64         cid1.put("CIDADE", "Belo Horizonte");
65         JSONObject cid2 = new JSONObject();
66         cid2.put("CIDADE", "Poços de Caldas");
67         JSONObject cid3 = new JSONObject();
68         cid3.put("CIDADE", "Ouro Preto");
69
70         collect.add(cid1);
71         collect.add(cid2);
72         collect.add(cid3);
73     } else if (sigla.equals("PR")) {
74         JSONObject cid1 = new JSONObject();
75         cid1.put("CIDADE", "Londrina");
76         JSONObject cid2 = new JSONObject();
77         cid2.put("CIDADE", "Cascavel");
78         JSONObject cid3 = new JSONObject();
79         cid3.put("CIDADE", "Curitiba");
80
81         collect.add(cid1);
82         collect.add(cid2);
83         collect.add(cid3);
84     } else {
85         JSONObject cid1 = new JSONObject();
86         cid1.put("CIDADE", "Cidade 1");
87         JSONObject cid2 = new JSONObject();
88         cid2.put("CIDADE", "Cidade 2");
89         JSONObject cid3 = new JSONObject();
90         cid3.put("CIDADE", "Cidade 3");
91
92         collect.add(cid1);
93         collect.add(cid2);
94         collect.add(cid3);
95     }
96
97     retorno.put("dados", collect);
98 } catch (Exception e) {
99     e.printStackTrace();
100     try {
101         retorno.put("error", "Erro");
102     } catch (JSONException e1) {
103         // TODO Auto-generated catch block
104         e1.printStackTrace();
105     }
106 }
107
108 response.setContentType("application/json");
109 PrintWriter out = response.getWriter();
110 out.print(retorno.toString());
111 out.close();
112 }
113 }
```

Nas imagens acima temos o fonte da classe ExemploControllerApi.java ela está no pacote com.lecom.workflow.cadastros.common.controller e ela consiste em ter valores fixos, imaginando que você está buscando cidades de um determinado estado vamos ver como está sendo feito:

Na **linha 17** conforme já vimos em outro documento essa parte está sendo feito o mapeamento da classe, dizendo como ela será chamada no caso utilizamos o app/public para que esse controller seja chamado sem precisar estar logado no ambiente.

Na **linha 30** iniciamos a variável collect um JSONArray, para que possamos no final retornar para tela um array no formato json para que possamos trabalhar com essa informação nos consumos e a forma esperada na Integração Api.

Na **linha 32** estamos recebendo um parâmetro chamado SIGLA para assim conseguirmos testar qual estado está vindo e com isso retornar as cidades.

Nas **linhas 34 a 47** estamos testando se o estado é SP, e criamos pequenos JSONObject colocando que o atributo vai ter nome CIDADE e ganhará os valores de Bauru, Agudos, Piratininga e Pederneiras, após cada JSONObject instanciado adicionamos eles a variável collect o nosso JSONArray que será retornado para tela.

Nas **linhas 48 a 83** estamos testando os estados RJ, MG e PR, caso entre em cada um desses ele faz a configuração de quais cidades serão retornadas em cada uma delas.

Nas **linhas 84 a 94** como é um exemplo fixado caso não entre em nenhum desses 4 estados mencionados acima vamos retornar para tela informações fixas como Cidade 1,

Cidade 2 e Cidade 3 apenas para demonstrar como retornamos um array de json object para tela via controller.

Na **linha 97** é onde vamos falar como chamará nosso atributo principal de retorno, dando o nome dados para o valor do JSONArray populado.

Nas **linhas 108 a 111** é a forma que utilizamos para retornar para tela um tipo json, por isso definimos o content type e depois printamos esse retorno.

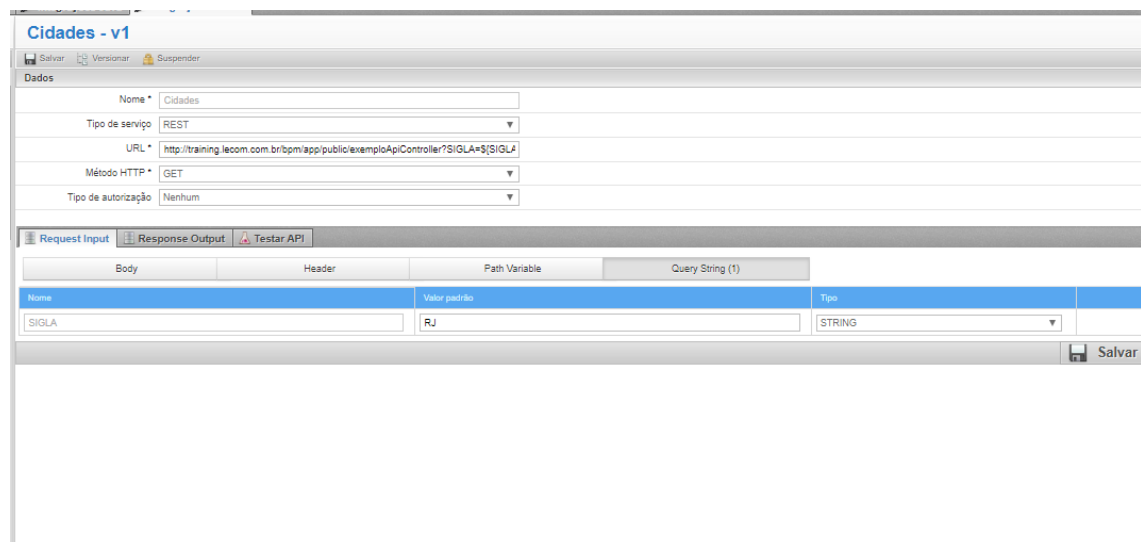
Se eu acessasse direto essa url sem passar parâmetro sigla e passando um parâmetro ela retornaria no browser:



4. Utilizando na Integração API

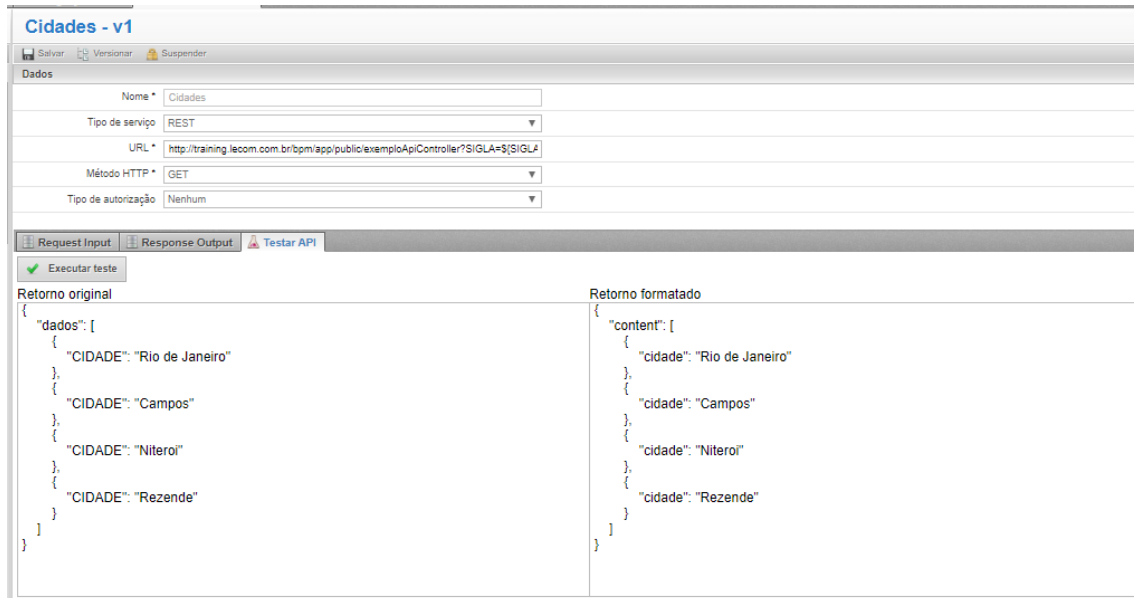
No Lecom BPM temos uma opção de serviço Integração de API, onde nela podemos utilizar consumo via REST ou SOAP, para essa configuração funcionar nós sempre configuraremos uma api que retorne um array de dados, mesmo que ela retorne só uma informação ela precisa ser um array para que o produto funcione como se fosse uma lupa.

Para acessarmos precisa entrar no Lecom BPM -> Studio -> Serviços -> Integrações de API:



Na imagem acima temos a configuração do controller que vimos acima, então primeiro coloco um nome para a API, depois defino se será REST ou SOAP, e colocamos a url, como é rest e preciso passar um parâmetro na url configura-se a chamada toda mais o nome da variável SIGLA, do mesmo jeito que está no controller, e na frente dela colocamos \${SIGLA}

que significa que é esse nome que usaremos para substituir os valores, conforme abaixo na imagem, usaremos a opção de Query String, por isso a configuração na url foi feita da forma explicada anteriormente, colocamos o nome do parâmetro e definimos um valor padrão apenas para testar nessa tela da API, quando fazemos essa configuração podemos salvar a api e assim ele habilitará a aba de Testar API.



Cidades - v1

Salvar Versionar Suspendir

Dados

Nome * Cidades

Tipo de serviço REST

URL * http://training.lecom.com.br/bpm/app/public/emploApi/Controller?SIGLA=\${SIGLA}

Método HTTP * GET

Tipo de autorização Nenhum

Request Input Response Output Testar API

Executar teste

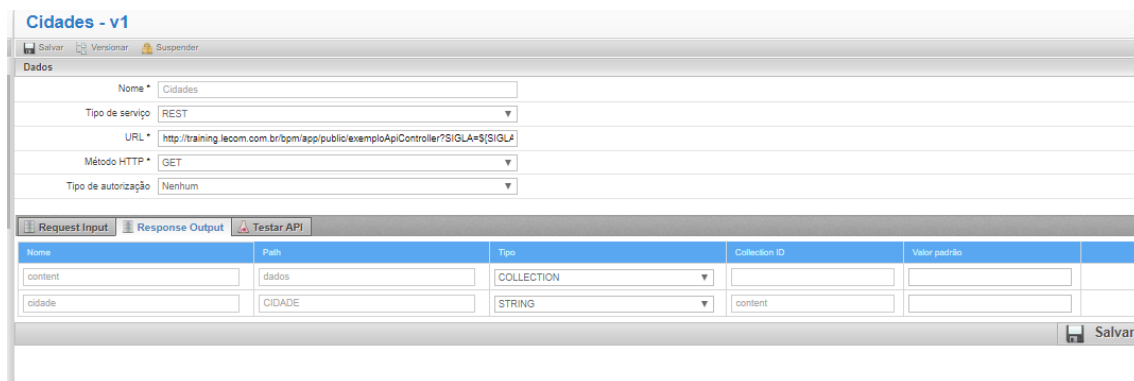
Retorno original

```
{
  "dados": [
    {
      "CIDADE": "Rio de Janeiro"
    },
    {
      "CIDADE": "Campos"
    },
    {
      "CIDADE": "Niteroi"
    },
    {
      "CIDADE": "Rezende"
    }
  ]
}
```

Retorno formatado

```
{
  "content": [
    {
      "cidade": "Rio de Janeiro"
    },
    {
      "cidade": "Campos"
    },
    {
      "cidade": "Niteroi"
    },
    {
      "cidade": "Rezende"
    }
  ]
}
```

Na imagem acima podemos clicar no botão Executar Teste ele retornará o retorno original, e o retorno formatado, o que importa para o Lecom Bpm utilizar é o retorno formatado.



Cidades - v1

Salvar Versionar Suspendir

Dados

Nome * Cidades

Tipo de serviço REST

URL * http://training.lecom.com.br/bpm/app/public/emploApi/Controller?SIGLA=\${SIGLA}

Método HTTP * GET

Tipo de autorização Nenhum

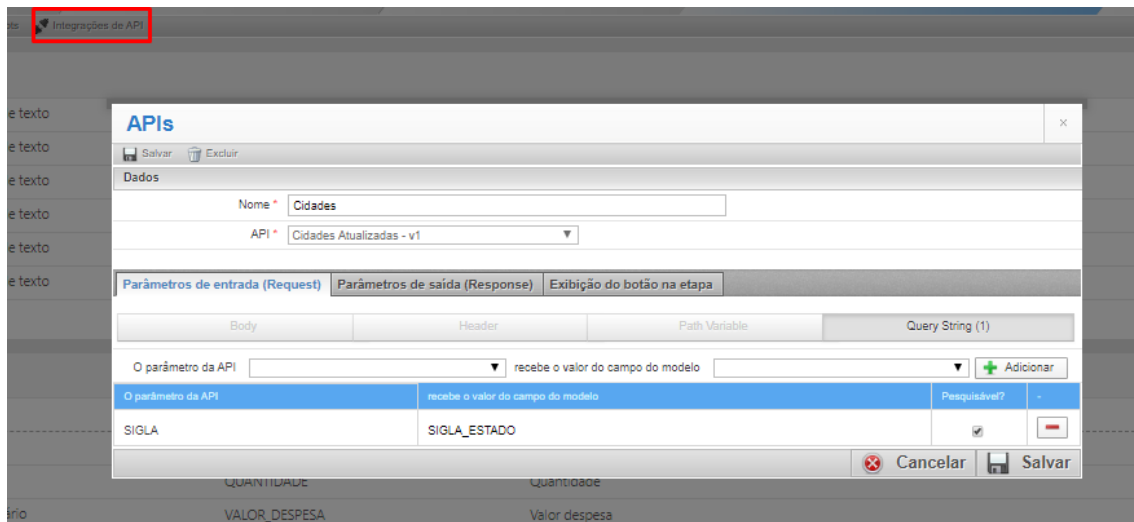
Request Input Response Output Testar API

Nome	Path	Tipo	Collection ID	Valor padrão
content	dados	COLLECTION		
cidade	CIDADE	STRING	content	

Salvar

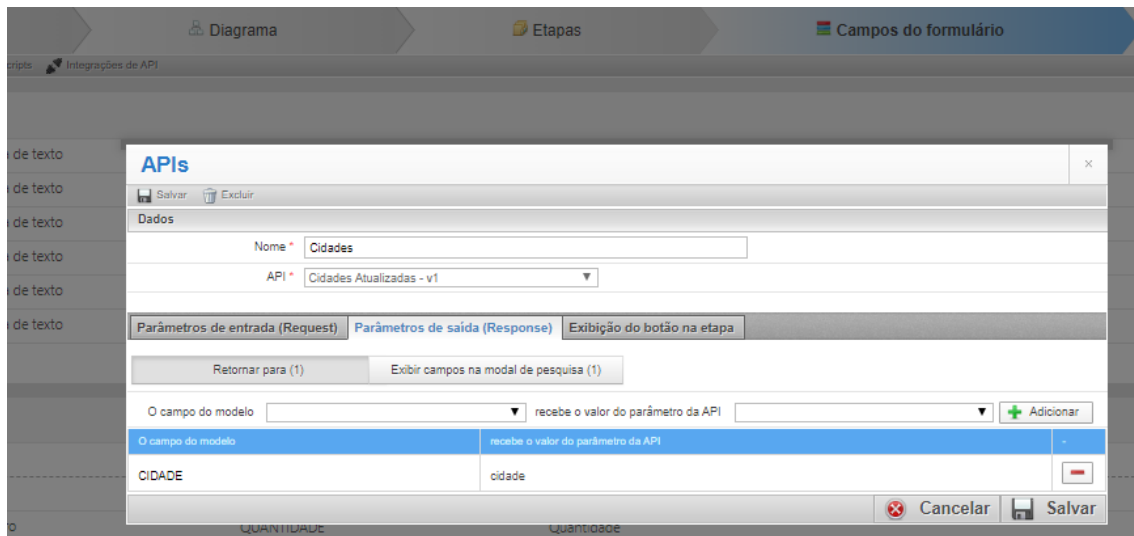
Na imagem acima vamos entrar na aba Response Output, é nesse lugar que iremos configurar o que será retornado com qual informação trabalhamos, o primeiro parâmetro que precisa ser configurado o array retornado, esse parâmetro ele obrigatoriamente precisa ter o nome content, o path deve ser configurado com o nome do array retornado no retorno original, por isso nesse exemplo chamamos de dados, e o tipo é collection porque é um array.

Na linha abaixo configuramos nesse exemplo o parâmetro cidade, colocando o nome que desejamos que apareça na configuração do processo no bpm, e o path é nome do atributo que veio do retorno original, o tipo geralmente String, e o collection id que é o nome que demos na primeira linha, content no caso.



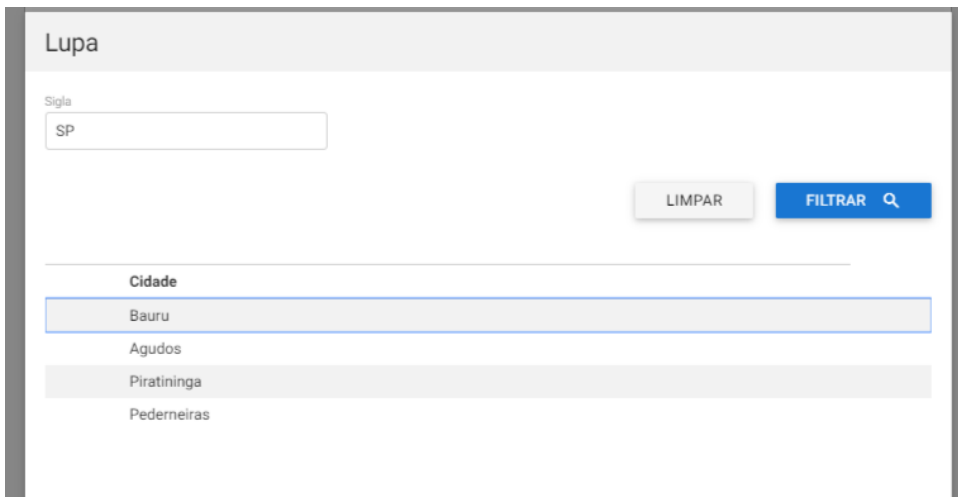
Na imagem acima precisamos entrar no processo que deseja usar essa api, e clicar no botão integração de api, depois selecionar qual api quer usar, dando um nome para essa configuração, e posteriormente configurar parâmetros de entrada, de saída e onde irá aparecer o botão de lupa.

Na imagem acima já vem fixado o botão a qual foi configurado como entrada na tela da api conforme visto acima, por isso está “checado” a opção Query String. No combo parâmetro da API vem todos os campos do tipo entrada configurado, nesse exemplo só vem a opção SIGLA, e no combo recebe valor do campo do modelo, você escolherá de qual campo do modelo pegará o valor, e checa se será pesquisável esse campo.



Na imagem acima é a configuração da aba parâmetros de saída, nela deveremos configurar o campo do modelo que receberá o valor retornado da API. Na aba exibição de botão na etapa você escolherá em qual etapa e em qual campo aparecerá o botão lupa.





Nas imagens acima é como aparecerá o botão da lupa e uma execução dela para que vocês possam ver que retorna conforme os dados fixos colocados no controller.

5. Consumindo via AJAX

Podemos fazer um consumo via javascript do modelo, imaginando que vamos acessar o controller que criamos e o retorno dele colocaremos em um campo do tipo lista. O arquivo que vamos ver é o exemploChamadaControllerMontaLista.js e ele se encontra na pasta /upload/rotinaJS/ dentro do Projeto Base.

```

1 |
2 | $(document).ready(function () {
3 |     montaListaCampo();
4 | });
5 |
6 |
7 | function montaListaCampo(){
8 |     $.ajax({
9 |         type: 'GET',
10 |         data: "SIGLA=" + Form.fields("SIGLA_ESTADO").value(),
11 |         url: location.protocol + "://" + location.hostname + "/bpm/app/public/exemploApiController",
12 |         success: function(retorno) {
13 |             debugger;
14 |             if(retorno.error == undefined){
15 |                 var arr = retorno.dados.map(function (item) {
16 |                     return item["CIDADE"];
17 |                 });
18 |                 Form.fields("CIDADES_TESTE").addOptions(arr, true).apply();
19 |             }
20 |         },
21 |         error: function(){
22 |             console.log(arguments);
23 |         },
24 |         crossDomain: true,
25 |         xhrFields: { withCredentials: true }
26 |     });
27 | }
28 |

```

Na imagens acima temos um método onde fazemos uma chamada via ajax para o controller criado.

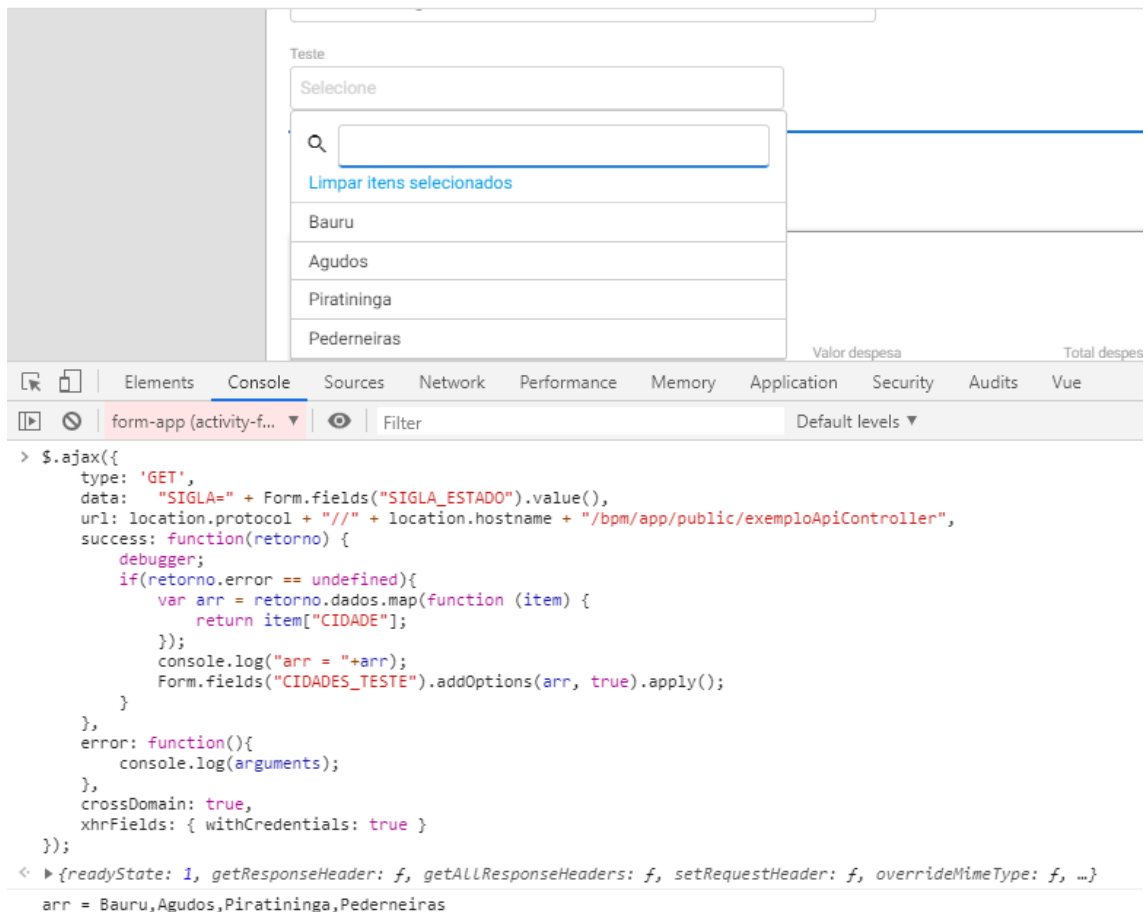
Na **linha 10** onde definimos qual será o parâmetro para ser enviado ao controller nesse caso definimos a SIGLA e passamos o valor do campo SIGLA_ESTADO pegando esses valor utilizando a api js do produto.

Nas **linhas 13 a 20** primeiro é validado se o está sem valor o atributo error, se estiver sem significa que deu sucesso na chamada, e trabalhamos em cima do retorno.dados,

conforme vimos nas imagens de retorno no browser quando o controller é chamado o nome do atributo para o array é dados, e assim no js conseguimos acessar direto utilizando o nome retorno pois é ele que está vindo dentro do function de sucesso.

Nesse bloco também faremos uma conversão do array retornado em um array somente com os valores, por conta de dentro do array ter os atributos CIDADE, por isso transformamos na variável **arr**.

Na **linha 19** passamos o valor dessa variável para o método addOptions de um campo do tipo Lista, e assim o campo recebe o valor de acordo com o parâmetro enviado.



The screenshot shows a web application interface with a form titled 'Teste'. The form has a dropdown menu labeled 'Selecione' with a search icon and a list of options: Bauru, Agudos, Piratininga, and Pederneiras. Below the dropdown is a button labeled 'Limpar itens selecionados'. The browser's developer console is open, showing the execution of an \$.ajax call. The console output shows the following code snippet:

```

> $.ajax({
  type: 'GET',
  data: "SIGLA=" + Form.fields("SIGLA_ESTADO").value(),
  url: location.protocol + "://" + location.hostname + "/bpm/app/public/exemploApiController",
  success: function(retorno) {
    debugger;
    if(retorno.error == undefined){
      var arr = retorno.dados.map(function (item) {
        return item["CIDADE"];
      });
      console.log("arr = "+arr);
      Form.fields("CIDADES_TESTE").addOptions(arr, true).apply();
    }
  },
  error: function(){
    console.log(arguments);
  },
  crossDomain: true,
  xhrFields: { withCredentials: true }
});

```

The console output shows the following result:

```

{readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, ...}
arr = Bauru,Agudos,Piratininga,Pederneiras

```

Na imagem acima apenas utilizei o ajax no console do navegador e o resultado foi preenchido no campo teste conforme na imagem acima.

6. Consumindo via Integração JAVA

Podemos também realizar o consumo desse controller criado via JAVA, esse exemplo está no arquivo **MontaCampoLista** e ele está no pacote **com.lecom.workflow.integracao.campo** e também para compor esse consumo vamos usar a classe Cidade que terá apenas um atributo é uma forma mais rápida de transformar o retorno.


```
MontaCampoLista.java
1 package com.lecom.workflow.integracao.campo;
2
3 import java.io.BufferedReader;
4
5 @IntegrationModule("MontaCampoLista")
6 @Version({1,0,2})
7 public class MontaCampoLista {
8
9     private static final Logger logger = Logger.getLogger(MontaCampoLista.class);
10
11     @SuppressWarnings("resource")
12     @Execution
13     public String consumirApi(IntegracaoVO integracaoVO) {
14
15         logger.debug("consumirApi");
16
17         //Utilizando http client lib ja existente dentro da plataforma lecom para utilizarmos os consumos de api
18         HttpClient client = HttpClientBuilder.create().build();
19         //caso a chamada seja de entrada um json
20         Gson gson;
21
22         //String que vamos utilizar para transformar o envio caso a chamada seja de entrada um json.
23         StringEntity requestEntity = null;
24
25         //Tipos de consumo, depende da chamada rest criada do "outro lado"
26         //HttpPost post = null;
27         //HttpPut put = null;
28         //HttpGet get = null;
29
30         try {
31             Map<String,String> campos = integracaoVO.getMapCamposFormulario();
32             String sigla = campos.get("SIGLA_ESTADO");
33
34             HttpPost post = new HttpPost("http://training.lecom.com.br/bpm/app/public/emploApiController?SIGLA="+sigla);
35
36             //caso tenha headers na url que será consumida precisa colocar quantos forem igual abaixo para o tipo do metodo, post/get/put
37             //post.addHeader("nomeParametro", "valorParametro");
38
39             gson = new Gson();
40             //na linha abaixo no momento em que chamo gson.toJson passando o objeto setado acima estamos fazendo uma transformação dos atributos que estão nessa classe dadosusuario
41             //para ser criado um json deles e serem enviados assim para a chamada da api.
42             //requestEntity = new StringEntity(gson.toJson(objetoClasse), ContentType.TEXT_PLAIN);
43             //post.setEntity(requestEntity);
44
45             //TODO: Retorno da chamada
46             HttpResponse response = client.execute(post);
47
48             if(response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK){
49                 BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
50
51                 StringBuilder result = new StringBuilder();
52                 String line = "";
53                 while ((line = rd.readLine()) != null) {
54                     result.append(line);
55                 }
56
57                 org.json.JSONObject json = new org.json.JSONObject(result.toString());
58
59                 org.json.JSONArray jsonArray = (JSONArray) json.get("dados");
60                 Type listType = new TypeToken<List<Cidade>>().getType();
61                 List<Cidade> cidades = gson.fromJson(jsonObject.toString(), listType);
62
63                 String cidadesString = cidades.stream().map(a -> String.valueOf(a.getCIDADE())).collect(Collectors.joining(";"));
64
65                 return "0"+cidadesString;
66             }else{
67                 BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
68
69                 StringBuilder result = new StringBuilder();
70                 String line = "";
71                 while ((line = rd.readLine()) != null) {
72                     result.append(line);
73                 }
74
75                 org.json.JSONObject json = new org.json.JSONObject(result.toString());
76                 org.json.JSONArray jsonArray = (JSONArray) json.get("errors");
77
78                 logger.debug(" error = "+jsonObject.toString());
79                 return "99 Não foi possível retornar as cidades do estado";
80             }
81
82         } catch (ClientProtocolException e) {
83             // TODO Auto-generated catch block
84             e.printStackTrace();
85         } catch (IOException e) {
86             // TODO Auto-generated catch block
87
88         }
89     }
90 }
```

Na imagem temos o fonte da integração utilizando httpclient para poder realizar os consumos, a classe em si já vai estar com mais exemplos comentados apenas para você verificarem e terem como exemplo de uso.

Nas **linhas 31 a 32** vamos inicializar as annotation que fala que a classe é do tipo integração e qual a versão da classe.

Na **linha 44** estamos instanciando o HttpClient que é o que usaremos para executar nossa chamada de api, que poderá ser um POST, GET, PUT.

Nas **linhas 57 e 58** estamos retornando todos os campos do processo e pegando o campo que precisamos, nesse caso campo SIGLA_ESTADO.

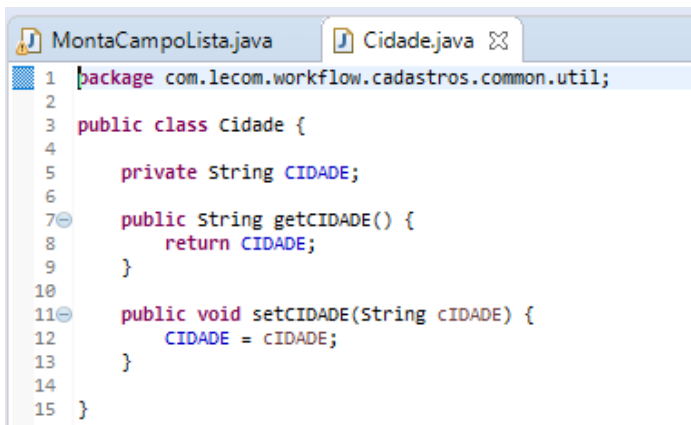
Na **linha 60** estamos inicializando o HttpPost passando a url toda do controller criamos no começo do documento, passando o parâmetro na url, porém sendo chamado via post para não ficar exposto o parâmetro.

Na **linha 66** inicializo o Gson, esse plugin será utilizado para transformar uma classe java em um json ou o contrário facilitando essas conversões.

Na **linha 72** é onde iremos realizar a chamada de fato da url, mandando executar ela em background, isso nos retorna um HttpResponse e do response verificaremos se o retorno deu sucesso ou não conforme a **linha 75**.

Nas **linhas 76 a 84** está sendo lido o retorno da chamada e transformando isso em um json, para que a partir do json retornado nós conseguimos pegar o valor do atributo dados, conforme na **linha 86**.

Nas **linhas 87 e 88** temos a conversão do retorno do json array para uma lista de objeto do tipo Cidade, para isso utilizamos o Type, TypeToken e o Gson para fazer a transformação.



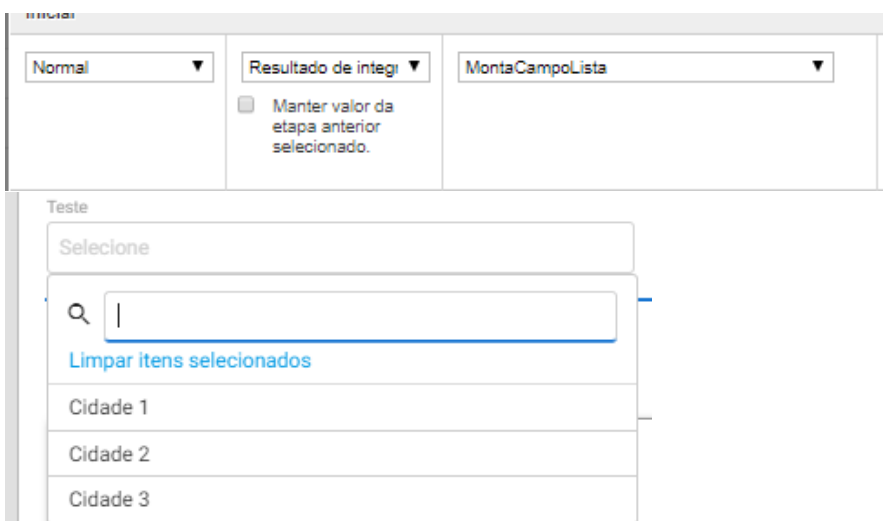
```

1 package com.lecom.workflow.cadastros.common.util;
2
3 public class Cidade {
4
5     private String CIDADE;
6
7     public String getCIDADE() {
8         return CIDADE;
9     }
10
11    public void setCIDADE(String CIDADE) {
12        CIDADE = CIDADE;
13    }
14
15 }

```

Na imagem acima é a classe útil que estamos utilizando nesse exemplo ela precisa ter os atributos com os mesmos nome que tem no retorno do controller, no caso é CIDADE.

Na **linha 90** estamos fazendo uma conversão da lista em uma String somente dos valores separando-as por ; , isso porque na linha abaixo estou retornando essa informação para o processo e é dessa forma que preciso mandar para que seja aplicado no processo as opções no campo.



Nas imagens acima é a configuração no campo utilizando a integração acima explicada e como veio os resultados no campo como está sendo no carregamento da etapa não tem valor de sigla de estado ele vem com os valores padrões.