



PROJETO CUSTOMIZAÇÕES

Gerador PDF Genérico

Amanda Alvim

1. Objetivo

O objetivo dessa customização é realizar uma geração de PDF a partir de um DOCX com os dados do formulário e atrelar esse pdf gerado a um campo do processo.

O robô contendo essa implementação se encontra no Projeto Base no nosso git (<http://git.lecom.com.br/PSP/Projeto-Base-BPM>) com o nome de **GeradorPDF**, que está no package com.lecom.workflow.integracao.pagina, existe também algumas libs que são necessárias para o funcionamento dessa geração, essas libs estão na pasta libs/geraPdf nesse mesmo git.

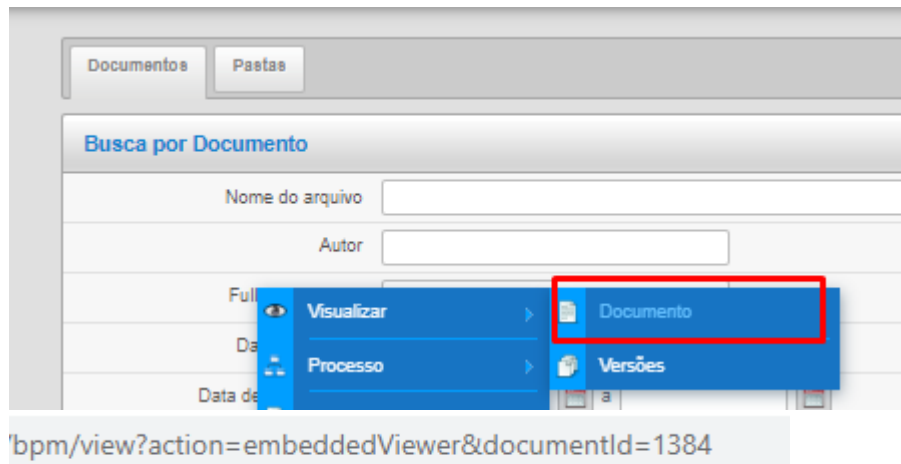
2. Como usar

No momento em que for utilizar essa implementação será necessário criar os campos que será explicado abaixo no seu modelo, subir o docx em um template do ambiente e adicionar todas as libs que estão na pasta geraPDF do Projeto Base e elas deverão ser colocadas diretamente no servidor do cliente no diretório /opt/lecom/app/tomcat/webapps/bpm/web-inf/lib para ambientes com o modo antigo de atualização das customizações e no diretório /opt/lecom/custom/libs para ambientes com o modo novo de atualização das customizações, também precisará antes de gerar o jar alterar para os valores do seu cliente o método **getUniqueldECM** que será explicado mais abaixo, após isso gerar o jar da integração e subir no ambiente como integração de página, você deverá configurar em uma atividade que irá fazer somente essa geração de pdf, e quando você usa a integração de página você consegue selecionar um campo do tipo template para receber valor.

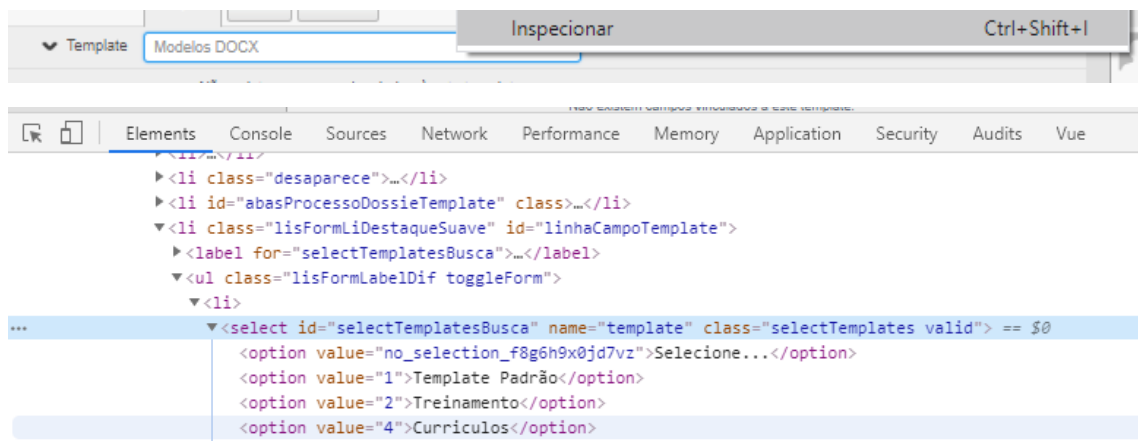
3. Campos no Modelo

Para o funcionamento dessa classe genérico será necessário criar em seu processo alguns campos:

- **PDF_ENTRADA_ID**: esse campo é utilizado para ser preenchido o código do documento do docx importado no ecm, e o id do template que está esse docx importado. Exemplo: 1384,4. Esse deve ser o valor desse campo na atividade anterior a que você configurará a integração genérica. Você irá fazer a pergunta como pego essas informações? Vamos ver:
 - O código do documento você consegue pegar na própria url da visualização do documento no ecm, quando entramos no modulo do ecm , realizamos uma busca de arquivo a partir do template, clicamos com o botão direito no documento encontrado e escolhemos a opção de visualização do documento e assim temos esse resultado abaixo, e para nós o que importa é o código 1384.



- O código do template você poderá pegar pela base de dados que nós consultores temos acesso, esse código será necessário para utilizarmos api do produto para buscar o documento e não utilizar em nenhum momento select no banco, também para quem conhece um pouco da para pegar o código do template olhando o html do produto apenas para pegar o código na tela de busca de documento quando escolhe um template, clica com o botão direito em cima do campo template e escolhe a opção inspecionar elemento no navegador (estou utilizando o Chrome para demonstração e Windows), na parte de inspecionar ele aparece o campo select do template com os values que são os ids dos template, como no exemplo poderíamos utilizar o código 4.

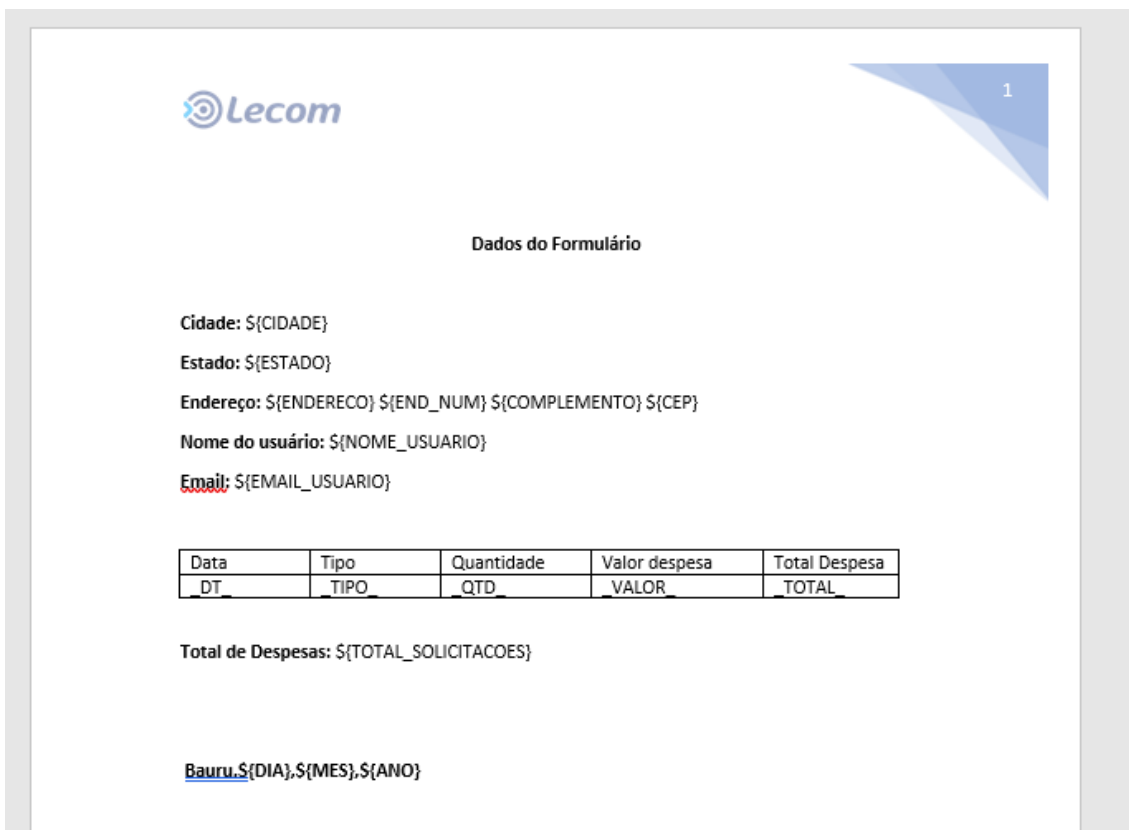


- PDF_SAIDA: esse campo deve ser do tipo template que deverá ser associado a um template específico criado por você ele que receberá o arquivo gerado no final da integração.
- PDF_TEMPLATE_SAIDA: esse campo deve conter o valor do identificador do template associado no campo acima.
- PDF_NOME: esse campo deve receber o valor do nome do arquivo que você deseja que seja gerado, ele já fará uma concatenação com o código do processo no final.
- PDF_CAMPO_ID: esse campo deve receber o valor do código do campo PDF_SAIDA, valor esse que você consegue pegar na tela do studio, na listagem de campos passando o mouse por cima você tem o código desse campo.

- PDF_GRIDS: esse campo deve conter o valor dos identificadores das grids que vocês irão utilizar no docx, podendo ser mais de uma. Exemplo: DESPESAS,ITENS.

4. Documento DOCX

Precisaremos criar um DOCX de acordo com o que você deseja que seja o conteúdo, ou algum documento que o cliente já tenha como padrão e você deve modificar colocando os campos que tem no seu processo onde eles serão modificados no documento, colocando campos normais com `#{NOME_CAMPO}` e os da grid precisa ser feito dentro de uma tabela do word, contendo a primeira linha os títulos e a segunda linha os campos da grid com `_NOME_CAMPO_`, conforme exemplo abaixo (Exemplo esse que está no Projeto Base do git no diretório **Documentação/word_pdf/Dados do Formulário.docx**).



Dados do Formulário

Cidade: `#{CIDADE}`

Estado: `#{ESTADO}`

Endereço: `#{ENDERECO}` `#{END_NUM}` `#{COMPLEMENTO}` `#{CEP}`

Nome do usuário: `#{NOME_USUARIO}`

Email: `#{EMAIL_USUARIO}`

| Data | Tipo | Quantidade | Valor despesa | Total Despesa |
|-------------------|---------------------|--------------------|----------------------|----------------------|
| <code>_DT_</code> | <code>_TIPO_</code> | <code>_QTD_</code> | <code>_VALOR_</code> | <code>_TOTAL_</code> |

Total de Despesas: `#{TOTAL_SOLICITACOES}`

Bauru.`#{DIA}`.`#{MES}`.`#{ANO}`

5. Fonte JAVA

Agora vamos entender como funciona a integração, lembrando que você não irá precisar fazer alteração para o funcionamento da mesma, somente precisa criar um processo contendo os campos informados e explicados acima com valores respectivos na atividade anterior da atividade que você irá configurar essa integração como retorno de integração de página e o campo PDF_SAIDA como campo de retorno.

| | | | 6 ativos / 0 inativos |
|--------------|----------------|--------------------|---|
| 1 | Linha de texto | PDF_ENTRADA_ID |    |
| 2 | Linha de texto | PDF_TEMPLATE_SA... |    |
| 3 | Linha de texto | PDF_CAMPO_ID |    |
| 4 | Linha de texto | PDF_GRIDS |    |
| 5 | Linha de texto | PDF_NOME |    |
| 6 | Template | PDF_SAIDA |    |
| + Novo campo | | | |

Resultado de integração por página

Integrações Java: GeradorPDF

Campos

Campos

A ordem dos campos abaixo deverá seguir a mesma ordem dos campos retornados no resultado da rotina (Java).

☐ Todos

| | | |
|--|---|--|
| <input type="checkbox"/> 1 - NOME_ESTADO | <input type="checkbox"/> 2 - SIGLA_ESTADO | <input type="checkbox"/> 3 - NOME_USUARIO |
| <input type="checkbox"/> 4 - LOGIN_USUARIO | <input type="checkbox"/> 5 - TOTAL_DESPESAS | <input type="checkbox"/> 6 - ANEXO_SOLICITANTE |
| <input type="checkbox"/> 7 - EMAIL_USER | <input type="checkbox"/> 8 - CIDADE | <input type="checkbox"/> 9 - PDF_ENTRADA_ID |
| <input type="checkbox"/> 10 - PDF_TEMPLATE_SAIDA | <input type="checkbox"/> 11 - PDF_CAMPO_ID | <input type="checkbox"/> 12 - PDF_GRIDS |
| <input checked="" type="checkbox"/> 13 - PDF_SAIDA | <input type="checkbox"/> 14 - NUMERO_NOTA | <input type="checkbox"/> 15 - NOTA_FISCAL |
| <input type="checkbox"/> 16 - PDF_NOME | | |

Cancelar Salvar

Acima as imagens mostrando os campos criados no processo, a configuração da integração por página e selecionando o campo PDF_SAIDA como retorno. Esses campos todos tem configuração invisível somente o campo PDF_SAIDA tem configuração de somente leitura nessa atividade que a integração está configurada.

```

GeradorPDF.java
1 package com.lecom.workflow.integracao.pagina;
2
3 import static br.com.lecom.api.factory.ECMFactory.documento;
4
59
60 @IntegrationModule("GeradorPDF")
61 @Version({1,0,7})
62 public class GeradorPDF {
63
64     private static final Logger logger = Logger.getLogger(GeradorPDF.class);
65     private String configPath = Funcoes.getWRootDir() + "/upload/cadastros/pdfGerado";
66
67     @SuppressWarnings("unchecked")
68     @Execution
69     public String geraPDF(IntegracaoVO integracaoVO) {
70         logger.info("Inicio Geração de PDF processo " + integracaoVO.getCodProcesso());
71         try {
72
73             Map<String, Object> camposModelo = integracaoVO.getMapCamposFormulario();
74             // logger.debug("Campos = " + camposModelo);
75             logger.debug("Nome Tabela Modelo = " + integracaoVO.getNomeTabelaModelo());
76
77             String fileId = (String) camposModelo.get("$PDF_ENTRADA_ID"); // id Arquivo no ecm fid e id template
78             String pdfSaida = (String) camposModelo.get("$PDF_SAIDA");
79             // identificador do template de saida
80             String identificadorTemplateSaida = (String) camposModelo.get("$PDF_TEMPLATE_SAIDA");
81             String nomeArquivo = (String) camposModelo.get("$PDF_NOME");
82             String codCampoSaida = (String) camposModelo.get("$PDF_CAMPO_ID");
83             String camposGrid = (String) camposModelo.get("$PDF_GRIDS");
84
85             logger.info("ID docx = " + fileId);
86             logger.info("Identificador do template de saida = " + identificadorTemplateSaida);
87             logger.info("Campo de saida = " + codCampoSaida);
88             logger.info("Arquivo Final = " + pdfSaida);
89
90             // verifica se o arquivo ja existe
91             if ((pdfSaida == null || pdfSaida.trim().equals("")) && !fileId.equals("")) {
92                 String[] dadosId = fileId.split(",");
93                 String uniqueIdECM = getUniqueIdECM(dadosId[0], dadosId[1]);
94                 logger.debug("Inicio leitura doc = " + uniqueIdECM);
95                 InputStream arquivoModelo = documento().lerArquivo(uniqueIdECM);
96
97                 logger.debug("achei input = " + arquivoModelo);
98                 // Abre o documento
99                 WordprocessingMLPackage wordprocessingMLPackage = WordprocessingMLPackage.Load(arquivoModelo);
100

```

```

GeradorPDF.java
100 // Recupera a parte principal dele
101
102 Logger.debug("Recupera a parte principal dele");
103 MainDocumentPart mainDocumentPart = wordprocessingMLPackage.getMainDocumentPart();
104
105 HashMap<String, String> hashMap = montaHashMapTrocaVariavel(camposModelo, integracaoVO);
106
107 // Efeetua as trocas
108 Logger.debug("TROCAR VALORES");
109 try {
110
111     VariablePrepare.prepare(wordprocessingMLPackage);
112
113     mainDocumentPart.variableReplace(hashMap);
114
115     if (camposGrid != null && !camposGrid.equals("")) {
116         if (camposGrid.contains(",")) {
117             String[] identificadoresGrid = camposGrid.split(",");
118
119             for (String identificador : identificadoresGrid) {
120                 geraInformacoesGrid(integracaoVO, wordprocessingMLPackage, identificador);
121             }
122         } else {
123             geraInformacoesGrid(integracaoVO, wordprocessingMLPackage, camposGrid);
124         }
125     }
126
127     Logger.debug("GERAR NOVO ARQUIVO");
128     // criando diretório
129     new File(configPath).mkdirs();
130
131     String arquivoDocGerado = configPath + "/" + nomeArquivo + "_" + integracaoVO.getCodProcesso() + ".docx";
132     String arquivoPDFGerado = configPath + "/" + nomeArquivo + "_" + integracaoVO.getCodProcesso() + ".pdf";
133
134     File destDocx = new File(arquivoDocGerado);
135     wordprocessingMLPackage.save(destDocx);
136
137     Logger.debug("GERANDO PDF");
138     File arquivoPDFDocx = new File(arquivoPDFGerado);
139
140     try (FileInputStream is = new FileInputStream(destDocx)) {
141         XMPDocument docx = new XMPDocument(is);
142         PdfOptions options = PdfOptions.create();

```

```

GeradorPDF.java
143
144     Logger.debug("CRIANDO ARQUIVO BASEADO NO DOCX");
145     try (OutputStream outputStreamDocx = new FileOutputStream(arquivoPDFDocx)) {
146         PdfConverter.getInstance().convert(docx, outputStreamDocx, options);
147     }
148
149     Logger.debug("INSERINDO ARQUIVO NO ECM");
150     // Recupera nome do arquivo e o nome criptografado
151     Document documento = documento().criarDocumentoComIdentificador(arquivoPDFDocx, identificadorTemplateSaida).salvar();
152     DocFile arquivoDoc = documento.getCurrentFile();
153     String nomeArquivo1 = arquivoDoc.getFileName();
154     String nomeCriptografado = arquivoDoc.getFileUniqueId().getValue();
155     Logger.debug("ARQUIVO GERADO : " + nomeArquivo1);
156     Logger.debug("NOME CRIPTOGRAFADO : " + nomeCriptografado);
157
158     // Apaga arquivo da pasta pública do servidor
159     arquivoPDFDocx.delete();
160     destDocx.delete();
161
162     if (!nomeCriptografado.equals("")) {
163         relacionarProcessoTemplate(integracaoVO, integracaoVO.getCodProcesso(),
164             integracaoVO.getCodEtapa(), integracaoVO.getCodCiclo(), nomeCriptografado,
165             documento.getId().getValue(), codCamposSaida);
166     }
167
168     alteraValorCamposSaida(integracaoVO, nomeArquivo1 + ":" + nomeCriptografado);
169
170     return "0|" + nomeArquivo1 + ":" + nomeCriptografado;
171
172 } catch (Exception e) {
173     Logger.error("Erro ao gerar pdf ", e);
174     e.printStackTrace();
175     return "99|Não foi possível gerar PDF";
176 }
177
178 } else {
179     if (fileId.equals("")) {
180         return "99|Não existe numero do modelo";
181     }
182     if (!pdfSaida.equals("")) {
183         return "0|" + pdfSaida;
184     }
185     Logger.info("PDF JÁ EXISTE NO FORMULÁRIO");
186 }

```

```

GeradorPDF.java
187 } catch (Exception e) {
188     Logger.error("Erro ao gerar pdf ", e);
189     e.printStackTrace();
190 }
191 }
192 return "99|Não foi possível gerar PDF";
193 }

```

Nas imagens acima temos o começo da integração onde ela fala inicialmente a anotação qual é o nome da integração, inicializa a variável de log, informa qual o caminho principal a ser gerado os caminhos, caminho esse que será de utilização apenas para geração temporária do arquivo, logo abaixo informa qual é o método principal que será executado inicialmente.

Nas **linhas 73 a 88** pegamos o map com todos os valores de campos do formulário, e começo a pegar especificamente os campos que eu preciso, que foram explicados nas páginas 2,3 e 4 e logar eles para conseguirmos avaliar a execução da integração.

Na **linha 91** é validado se o valor do campo pdfSaida é vazio se não for vazio, significando que já existi um arquivo gerado ele não vai gerar novamente, também valida se tem o valor para o campo pdf_entrada_id, se não tiver também não executará a geração do arquivo.

Na **linha 92 e 93** inicialmente separo os valores da variável fileId, que está com valores do código do documento e código do template depois passo essa informações para o método **getUniquelIdECM**, e ele vai me retornar o uniqueid do ecm para o docx passado.

Na **linha 95** utilizamos a api do ecm para ler o documento, onde ele retorna o stream do arquivo a partir do uniqueid do ecm.

Nas **linhas 99 e 103** estamos utilizando as classes da lib docx4j, onde ele faz uma leitura do stream retornado, e pega a parte principal do documento.

Na **linha 105** estou chamando o método que configurará um hashmap com todos os campos que o processo já tem, mais alguns campos padrões para serem utilizados nos documentos que forem criar.

Na **linha 111** está sendo feito uma preparação desses campos para que na **linha 113** seja feito diante do documento principal retornado faça o replace desses campos internamente.

Nas **linhas 115 a 125** está sendo validado se foi passado algum identificador de grid e assim para cada identificador é chamado o método **geralInformacoesGrid** que será explicado mais para baixo.

Na **linha 129** está sendo feito a criação da pasta temporária no nosso diretório padrão de upload/cadastros do servidor para gravarmos os arquivos gerados lá.

Nas **linhas 131 e 132** inicializamos as variáveis com os caminhos do arquivo docx com os valores alterados, e o caminho do pdf que será gerado, nesse momento utilizamos o campo PDF_NOME caso queira que o arquivo seja gerado com algum prefixo diferente.

Nas **linhas 134 e 135** utilizamos aquela variável do arquivo principal para que o que tem nela seja salvo nesse diretório do arquivo do docx, pois nesse momento é onde ele pegou tudo que sofreu alteração os campos do modelo, os campos da grid e salvou ele alterado.

Nas **linhas 138 a 147** está sendo criado o arquivo o pdf, e partir de um fileinputstream do arquivo docx salvo alterado, é instanciado classes da lib poi para realizar a transformação via java de um arquivo docx em um PDF.

Nas **linhas 151 a 156** está sendo utilizado a api do ecm para criação do documento por fora, para isso passamos o arquivo pdf gerado acima e o identificador do template, campo criado no modelo, com isso temos as informações que precisamos para atribuir a um campo do modelo, o nome do arquivo e o uniqueid do ecm.

Nas **linhas 159 e 160** é feito a deleção dos arquivos temporários já que na linha anterior foi criado no ecm o arquivo.

Nas **linhas 162 a 166** é verificado se temos o nome criptografo(uniqueid) retornado do ecm, e com isso é chamado o método **relacionarProcessoTemplate**, como estamos fazendo em uma integração de página para campos do tipos template precisamos incluir essa relação código do processo, etapa, ciclo, uniqueid, documento e o código do campo em uma tabela temporária, esse método será explicado mais abaixo.

Na **linha 168** é chamado o método **alteraValorCampoSaida** mais uma vez pelo fato de estarmos em uma integração de página, eu preciso “salvar” o valor do campo na tabela do modelo, se não fazemos essas duas relações das tabelas não é possível fazer o download do documento após ele estar no campo.

Na **linha 170** é o retorno que temos que dar para uma integração, sempre começamos com numero 0 ou 99 e colocamos | e o que quisermos na frente se é uma integração de campo ou página retornamos o valor do campos que irá receber, e se for integração de passagem de etapa retornamos a mensagem que aparecerá para o usuário em tela.

Nas próximas linhas do método principal é validado se teve algum erro e loga essa informação e retorna para a chamada do produto o que aconteceu.

```
GeradorPDF.java
195 private void geraInformacoesGrid(IntegracaoVO integracaoVO, WordprocessingMLPackage wordprocessingMLPackage,
196 String identificador) throws Exception {
197
198     logger.info("geraInformacoesGrid identificador = " + identificador);
199
200     String nomeTabelaGrid = integracaoVO.getNomeTabelaModelo().replaceAll("f_", "g_");
201     String tabelaGrid = nomeTabelaGrid + identificador;
202     logger.info("geraInformacoesGrid tabelaGrid = " + tabelaGrid);
203
204     integracaoVO.setConexao("workflow");
205     List<Map<String, Object>> gridData = new ArrayList<Map<String, Object>>();
206     try (Connection con = integracaoVO.getConexao()) {
207         StringBuilder strProesso = new StringBuilder();
208         strProesso.append("select cod_form,cod_versao from processo p ");
209         strProesso.append(" where cod_processo = ? ");
210         String codForm = "";
211         String codVersao = "";
212         try (PreparedStatement pst = con.prepareStatement(strProesso.toString())) {
213             pst.setString(1, integracaoVO.getCodProcesso());
214             try (ResultSet rs = pst.executeQuery()) {
215                 if (rs.next()) {
216                     codForm = rs.getString("cod_form");
217                     codVersao = rs.getString("cod_versao");
218                 }
219             }
220         }
221
222         StringBuilder strCamposGrid = new StringBuilder();
223         strCamposGrid.append(
224             "select DES_NOME from campo where cod_form = ? and cod_versao = ? and id_agrupamento_grid = ? ");
225         List<String> nomesCamposGrid = new ArrayList<String>();
226         try (PreparedStatement pst = con.prepareStatement(strCamposGrid.toString())) {
227             pst.setString(1, codForm);
228             pst.setString(2, codVersao);
229             pst.setString(3, identificador);
230             try (ResultSet rs = pst.executeQuery()) {
231                 while (rs.next()) {
232                     nomesCamposGrid.add(rs.getString("DES_NOME"));
233                 }
234             }
235         }
236     }
```



```
GeradorPDF.java
236
237     StringBuilder dadosGrid = new StringBuilder();
238     dadosGrid.append("select * from ");
239     dadosGrid.append(tabelaGrid);
240     dadosGrid.append(" where cod_processo = ? ");
241     dadosGrid.append("and cod_etapa = ? ");
242     dadosGrid.append("and cod_ciclo = ? ");
243     try (PreparedStatement pst = con.prepareStatement(dadosGrid.toString())) {
244         pst.setString(1, integracaoVO.getCodProcesso());
245         pst.setString(2, integracaoVO.getCodEtapa());
246         pst.setString(3, integracaoVO.getCodCiclo());
247
248         try (ResultSet rs = pst.executeQuery()) {
249             while (rs.next()) {
250                 Map<String, Object> mapCamposGrid = new HashMap<String, Object>();
251                 for (String nomeColuna : nomesCamposGrid) {
252                     mapCamposGrid.put(nomeColuna, rs.getObject(nomeColuna));
253                 }
254                 gridData.add(mapCamposGrid);
255             }
256         }
257     }
258
259     if (gridData.size() == 0) {
260         Map<String, Object> mapCamposGrid = new HashMap<String, Object>();
261         nomesCamposGrid.forEach(nomeColuna -> mapCamposGrid.put(nomeColuna, ""));
262         gridData.add(mapCamposGrid);
263     }
264
265     Logger.info("geraInformacoesGrid gridData = " + gridData);
266     List<Map<String, String>> listHashMap = gridData.stream()
267         .map(map -> map.entrySet().stream()
268             .collect(Collectors.toMap(this::formataChaveGrid, this::formataValorGrid)))
269             .collect(Collectors.toList());
270     Logger.info("geraInformacoesGrid listHashMap = " + listHashMap);
271     Set<String> mapFields = listHashMap.stream().findFirst().orElseGet(Collections::emptyMap).keySet();
272     Logger.info("geraInformacoesGrid mapFields = " + mapFields);
273
274     atualizaTabela(mapFields.toArray(new String[mapFields.size()]), listHashMap, wordprocessingMLPackage);
275 }
---
```

Nas imagens acima está sendo demonstrado o método **geraInformacoesGrid** onde nele está sendo pego a partir do identificador da grid que foi passado, nós buscamos via banco de dados todos os campos que fazem parte dessa grid, pegando a partir do código de formulário, versão e depois fazendo select na tabela da grid especifica e transformando numa lista de `list<map<String,objetc>` para depois no final do arquivo fazer uma transformação no tipo da lista retornada, formatando a chave do nome do campo da grid, e o valor que esse campo teve, está sendo utilizado nesse método ações do java 8 para facilitar essa conversão, na api que utilizamos para fazer a conversão de docx em pdf precisamos que o tipo de lista de valores da grid seja em um formato de `set<String>` e por isso é feito varias transformação no final do método, e assim passamos as informações para o outro método **atualizaTabela** porque nesse que irá buscar dentro do docx onde tem tabelas e se existe os campos da grid para fazer o preenchimento de valores.

```
GeradorPDF.java
280
281     public String formataChaveGrid(Entry<String, Object> entry) {
282         String key = entry.getKey().replace("$", "");
283         String keyret = "_" + key + "_";
284         return keyret;
285     }
286
287     public String formataValorGrid(Entry<String, Object> entry) {
288         Object valor = entry.getValue();
289         if (Funcoes.nulo(valor, "").equals("")) {
290             return "XXX";
291         } else {
292             String retorno = converteData(valor);
293             if (retorno.indexOf("&") < 0) {
294                 retorno = retorno.toUpperCase();
295             }
296             return retorno;
297         }
298     }
299
300     private static String converteData(Object valor) {
301         String valorRetorno = "";
302         String formato = String.valueOf(valor).contains("T") ? "yyyy-MM-dd'T'HH:mm:ss.SSSX" : "yyyy-MM-dd HH:mm:ss.S";
303
304         try {
305             valorRetorno = new SimpleDateFormat("dd/MM/yyyy").format(new SimpleDateFormat(formato).parse(String.valueOf(valor)));
306         } catch (ParseException e) {
307             valorRetorno = String.valueOf(valor).toUpperCase();
308             valorRetorno = convertString(valorRetorno);
309         }
310         return valorRetorno;
311     }
312 }
```

Na imagem acima temos 3 métodos, **formataChaveGrid**, **formataValorGrid** e **converteData**.

O método **formataChaveGrid**, ele pega todos os campos, valores da grid em questão e transforma a chave o nome do campo de DT por exemplo para **_DT_**.

O método **formataValorGrid** ele lê todos os valores dos campos e verifica se algum está nulo e retorna XXX se não ele faz a chamada do método de **converterdata**, e com o retorno desse método ele coloca o valor para uppercase, tudo maiúsculo, e retorna para onde ele foi chamado.

O método **converterData** ele verifica se o valor que está vindo é uma data, tentando fazer uma conversão para um formato amigável, se der erro na conversão ele já sabe que é um valor comum e chama o método **convertString** e assim só retorna o valor para o método que o chamou.

```

313
314 private static void atualizaTabela(String[] placeholders, List<Map<String, String>> textToAdd,
315 WordprocessingMLPackage template) throws Exception {
316     Logger.debug("VALORES PARA ADICIONAR : " + textToAdd);
317     if (placeholders.length > 0) {
318         List<Object> tables = getAllElementFromObject(template.getMainDocumentPart(), Tbl.class);
319         // Encontra tabela no doc
320         Tbl tempTable = getTemplateTable(tables, placeholders);
321         List<Object> rows = getAllElementFromObject(tempTable, Tr.class);
322
323         // tabela precisa ter 2 linhas (cabeçalho e conteúdo)
324         if (rows.size() == 2) {
325             Tr templateRow = (Tr) rows.get(1);
326
327             // adiciona linhas a tabela
328             textToAdd.forEach(replacements -> addRowToTable(tempTable, templateRow, replacements));
329
330             // Remove linha
331             tempTable.getContent().remove(templateRow);
332         }
333     }
334 }
335
336 private static List<Object> getAllElementFromObject(Object obj, Class<?> toSearch) {
337     List<Object> result = new ArrayList<Object>();
338     if (obj instanceof JAXBElement)
339         obj = ((JAXBElement<?>) obj).getValue();
340
341     if (obj != null && obj.getClass().equals(toSearch))
342         result.add(obj);
343     else if (obj instanceof ContentAccessor) {
344         List<?> children = ((ContentAccessor) obj).getContent();
345         children.forEach(child -> result.addAll(getAllElementFromObject(child, toSearch)));
346     }
347     return result;
348 }

```

O método acima de **atualizaTabela**, ele utiliza as classes das libs externas que estamos utilizando para começar a procurar no documento se existe uma tabela e se existir, ele verifica se a tabela tem 2 linhas, conforme mencionei na explicação do doc primeira linha com títulos segunda linha com os nomes dos campos, e a partir disso ele faz a ação de ler essas linhas e valores que veio para realizar as alterações de valores, esse método chama outros métodos para completar essa alteração, **getAllElementFromObject**, **getTemplateTable**, **addRowToTable**.

Nessa mesma imagem acima temos o método **getAllElementFromObject**, que chama ele mesmo recursivamente e vai encontrando a tabela, linha com os campos passados para ser feito a alteração de valores.

```

350 private static Tbl getTemplateTable(List<Object> tables, String[] templateKey) throws Exception {
351     for (Iterator<Object> iterator = tables.iterator(); iterator.hasNext();) {
352         Object tbl = iterator.next();
353         // Pega todas as células da tabela
354         List<?> cells = getAllElementFromObject(tbl, Tc.class);
355         for (Object cell : cells) {
356             // Pega todos os parágrafos da célula
357             List<?> paragraphs = getAllElementFromObject((Tc) cell, P.class);
358
359             for (Object paragraph : paragraphs) {
360                 // Pega todos os textos do parágrafo
361                 List<?> texts = getAllElementFromObject((P) paragraph, Text.class);
362
363                 int i = 0;
364                 for (Object text : texts) {
365                     // Somente executa do segundo texto em diante
366                     if (i > 0) {
367                         // Move o conteúdo do texto para o primeiro texto, ao final (concatena)
368                         ((Text) texts.get(0)).setValue(((Text) texts.get(0)).getValue() + ((Text) text).getValue());
369                         // Limpa o conteúdo do texto, para efetuar o recorte
370                         ((Text) text).setValue("");
371                     }
372                     i++;
373                 }
374
375                 for (Object text : texts) {
376                     Text textElement = (Text) text;
377                     // Verifica se textElement existe e se existe uma chave para o elemento
378                     if (textElement.getValue() != null
379                         && Arrays.asList(templateKey).contains(textElement.getValue()))
380                         return (Tbl) tbl;
381                 }
382             }
383         }
384     }
385     return null;
386 }

```

Nessa imagem acima temos o método **getTemplateTable**, que chama o método **getAllElementFromObject** em vários momentos para ir procurando o que existe no docx que está sendo lido.

```

387
388 private static void addRowToTable(Tbl reviewtable, Tr templateRow, Map<String, String> replacements) {
389     Tr workingRow = (Tr) XmlUtils.deepCopy(templateRow);
390     // Pega todas as células da tabela
391     List<?> cells = getAllElementFromObject(workingRow, Tc.class);
392
393     for (Object cell : cells) {
394         // Pega todos os parágrafos da célula
395         List<?> paragraphs = getAllElementFromObject((Tc) cell, P.class);
396
397         for (Object paragraph : paragraphs) {
398             // Pega todos os textos do parágrafo
399             List<?> texts = getAllElementFromObject((P) paragraph, Text.class);
400
401             int i = 0;
402             for (Object text : texts) {
403                 if (i > 0) {
404                     // Move o conteúdo do texto para o primeiro texto, ao final (concatena)
405                     ((Text) texts.get(0)).setValue(((Text) texts.get(0)).getValue() + ((Text) text).getValue());
406                     // Limpa o conteúdo do texto, para efetuar o recorte
407                     ((Text) text).setValue("");
408                 }
409                 i++;
410             }
411
412             Text text = (Text) texts.get(0);
413             String replacementValue = (String) replacements.get(text.getValue());
414             if (replacementValue != null) {
415                 text.setValue(replacementValue);
416             }
417         }
418     }
419     reviewtable.getContent().add(workingRow);
420 }

```

Nessa imagem acima temos o método **addRowToTable**, que chama o método **getAllElementFromObject** em vários momentos para ir procurando o que existe no docx que está sendo lido e alterando e preenchendo o valor corretamente para cada campo.

```
GeradorPDF.java
422 public void alteraValorCampoSaida(IntegracaoVO integracaoVO, String valorCampoPDF) throws Exception {
423     integracaoVO.setConexao("workflow");
424     try (Connection con = integracaoVO.getConexao()) {
425         String update = "update " + integracaoVO.getNomeTabelaModelo()
426             + " set PDF_SAIDA = ? where cod_processo_f = ? and cod_etapa_f = ? and cod_ciclo_f = ? ";
427         try (PreparedStatement pst = con.prepareStatement(update)) {
428             pst.setString(1, valorCampoPDF);
429             pst.setString(2, integracaoVO.getCodProcesso());
430             pst.setString(3, integracaoVO.getCodEtapa());
431             pst.setString(4, integracaoVO.getCodCiclo());
432             pst.executeUpdate();
433         }
434     }
435 }
436 }
```

Nessa imagem acima temos o método **alteraValorCampoSaida**, este método fará o update na tabela do modelo para o processo, etapa e ciclo passado, fazendo isso para garantir o valor que o campo PDF_SAIDA receberá da integração.

```
GeradorPDF.java
438 public void relacionarProcessoTemplate(IntegracaoVO integracaoVO, String codProcesso, String codEtapa,
439     String codCiclo, String uniqueId, Long idDocumento, String codCampoPDFGerado) throws Exception {
440     try (Connection con = integracaoVO.getConexao()) {
441         String selectTemplate = "select count(*) as total from processo_template where cod_processo = ? and cod_etapa = ? and cod_ciclo = ? and cod_campo = ?";
442         try (PreparedStatement pst = con.prepareStatement(selectTemplate)) {
443             pst.setString(1, codProcesso);
444             pst.setString(2, codEtapa);
445             pst.setString(3, codCiclo);
446             pst.setString(4, codCampoPDFGerado);
447             Logger.debug("codProcesso = "+codProcesso+" codEtapa = "+codEtapa+" codCiclo = "+codCiclo+" codCampoPDFGerado = "+codCampoPDFGerado);
448             try (ResultSet rs = pst.executeQuery()) {
449                 if (rs.next()) {
450                     int total = rs.getInt("total");
451                     String sqlUpdateProcessoTemplate = (total > 0)
452                         ? "update processo_template set cod_arquivo_ecm = ?, cod_documento_ecm = ? where cod_processo = ? and cod_etapa = ? and cod_ciclo = ? and cod_campo = ?"
453                         : "insert into processo_template (cod_arquivo_ecm,cod_documento_ecm,cod_processo,cod_etapa,cod_ciclo,cod_campo) values (?,?,?,?,?,?)";
454                     Logger.debug("sqlUpdateProcessoTemplate = "+sqlUpdateProcessoTemplate);
455                     Logger.debug("uniqueId = "+uniqueId+" idDocumento = "+idDocumento+" codProcesso = "+codProcesso+" codEtapa = "+codEtapa);
456                     Logger.debug("codCiclo = "+codCiclo+" codCampoPDFGerado = "+codCampoPDFGerado);
457                     try (PreparedStatement pstUpdateProcessoTemplate = con.prepareStatement(sqlUpdateProcessoTemplate)) {
458                         pstUpdateProcessoTemplate.setString(1, uniqueId);
459                         pstUpdateProcessoTemplate.setLong(2, idDocumento);
460                         pstUpdateProcessoTemplate.setString(3, codProcesso);
461                         pstUpdateProcessoTemplate.setString(4, codEtapa);
462                         pstUpdateProcessoTemplate.setString(5, codCiclo);
463                         pstUpdateProcessoTemplate.setString(6, codCampoPDFGerado);
464                         boolean retorno = pstUpdateProcessoTemplate.execute();
465                         Logger.debug("retorno = "+retorno);
466                     }
467                 }
468             }
469         }
470     }
471 }
472 }
```

Nessa imagem acima temos o método **relacionarProcessoTemplate**, este método fará a verificação se já existe na tabela temporária processo_template o valor para esse campo, processo, etapa, ciclo se não existir ele faz uma inserção ou atualização do dado, isso é feito para o funcionamento melhor da integração de página com a liberação do download do documento gerado.

```
GeradorPDF.java
474 public String getUniquIdECM(String fileId, String idTemplate){
475     Logger.debug("getUniquIdECM fileId = "+fileId+" idTemplate = "+idTemplate);
476     DadosDocumento info = null;
477     try {
478         DadosLogin dadosLogin = new DadosLogin("adm", "adm", false);
479         LoginAutenticacao login = new LoginAutenticacao("http://iron.dev.local/sso", dadosLogin);
480         Logger.debug("login = "+login.getToken());
481         DocumentosECM documentosECM = new DocumentosECM("http://iron.dev.local/ecmcore", login.getToken(), idTemplate, "0", 50);
482         List<DadosDocumento> dados;
483         dados = documentosECM.getDocumentosByTemplate();
484         Logger.debug("dados = "+dados.size());
485         info = dados.stream().filter(x -> fileId.equals(x.getDocumentId())).findAny().get();
486         Logger.debug("UNIQUE ID ECM = "+info.getFileUniqueId()+" ");
487     } catch (DocumentosECMException e) {
488         // TODO Auto-generated catch block
489         e.printStackTrace();
490     } catch (LoginAutenticacaoException e) {
491         // TODO Auto-generated catch block
492         e.printStackTrace();
493     }
494     return info.getFileUniqueId();
495 }
496 }
497 }
```

Na imagem acima temos o método **getUniquIdECM**, nesse método você irá fazer o uso da api de documentos, onde faremos a busca de documentos a partir do id do template e assim validamos nos resultados encontrando o id do documento que você passou. Para isso precisar utilizar a **RotaListarDocumentos** vista no documento de **RotasBPM**, para utilizar aqui precisa preencher os valores do ambiente de vocês.

```

GeradorPDF.java
501 public HashMap<String, String> montaHashMapTrocaVariavel(Map<String, Object> camposModelo,
502     IntegracaoVO integracaoVO) throws Exception {
503     HashMap<String, String> variaveisReplace = new HashMap<String, String>();
504     camposModelo.forEach((k, v) -> {
505         String key = k.replace("$", "");
506         String chaveNova = key;
507         variaveisReplace.put(chaveNova, formataValor(v));
508     });
509
510     // preenchendo alguns padroes
511     variaveisReplace.put("PROCESSO", integracaoVO.getCodProcesso());
512     variaveisReplace.put("USER_INICIADOR", getNomeUsuario(integracaoVO.getCodUsuarioIniciador(), integracaoVO));
513     variaveisReplace.put("USER_RESPONSAVEL", getNomeUsuario(integracaoVO.getCodUsuarioEtapa(), integracaoVO));
514     variaveisReplace.put("DIA", String.valueOf(LocalDate.now().getDayOfMonth()));
515     variaveisReplace.put("MES", String.valueOf(LocalDate.now().getMonthValue()));
516     Calendar hoje = Calendar.getInstance();
517     String mesExtenso = new SimpleDateFormat("MMMM").format(hoje.getTime());
518
519     variaveisReplace.put("MES_EXTENSO", mesExtenso);
520     variaveisReplace.put("ANO", String.valueOf(LocalDate.now().getYear()));
521
522     Logger.debug("CAMPOS DO FORMULARIO : " + variaveisReplace);
523
524     return variaveisReplace;
525 }
526
527 public static String formataValor(Object entry) {
528     Object valor = entry;
529     if (Funcoes.nulo(valor, "").equals("")) {
530         return "XXX";
531     } else {
532         String retorno = converteData(valor);
533         if (retorno.indexOf("&") < 0) {
534             retorno = retorno.toUpperCase();
535         }
536         return retorno;
537     }
538 }

```

Na imagem acima temos o método **montaHashMapTrocaVariavel** e para ele pegamos os campos do modelo retornado da integração trocando o nome do campo de \$NOMECAMPO para NOMECAMPO sem o \$ pois assim a api que utilizamos consegue encontrar no docx o campo \${NOMECAMPO} e trocar o valor, além disso no map alterado adicionamos campos padrão para serem utilizados no docx como valores de quem abriu o processo, de quem aprovou, dia, mês, mês extenso e ano.

Na imagem acima também tem o método **formataValor** mesmo conceito do método **formataValorGrid**, verifica se o valor está nulo ou branco e preenche com XXX, se não faz chama o método **converteData** e retorna o valor formatado para ser alterado no documento.

```

539
540 public static String convertString(String valor) {
541     Logger.debug("VALOR PARA CONVERTER : " + valor);
542
543     valor = (valor.trim().indexOf("&") > 0) ? valor.replaceAll("&(?!amp;)", "&") : valor;
544
545     Logger.debug("VALOR CONVERTIDO : " + valor);
546
547     return valor;
548 }
549
550 public String getNomeUsuario(String codUserIniciador, IntegracaoVO integracaoVO) throws Exception {
551     integracaoVO.setConexao("workflow");
552     String nomeUsuario = "";
553     try (Connection con = integracaoVO.getConexao()) {
554         String sql = "select nom_usuario from usuario where cod_usuario = ?";
555         try (PreparedStatement pst = con.prepareStatement(sql)) {
556             pst.setString(1, codUserIniciador);
557             try (ResultSet rs = pst.executeQuery()) {
558                 if (rs.next()) {
559                     nomeUsuario = rs.getString("nom_usuario");
560                 }
561             }
562         }
563     }
564     return nomeUsuario;
565 }
566

```

Na imagem acima temos o método de **convertString**, esse método possui uma verificação se o valor possui & (e comercial), para trocar esse valor por um regex e assim o valor sair corretamente no documento final se não tiver apenas retorna o valor.

Também temos o método **getNomeUsuario** que a partir dos valores do código iniciador ou código do usuário aprovador, ele busca no banco de dados o nome do usuário para ser usado como uma variável padrão no seu docx.

6. Resultado

Pdf gerado
3420.pdf

555.pdf x

1 / 1 66,7%

Lecom

Dados do Formulário

Cidade: TETSTE
Estado: AL
Endereço: TESTE 2 TESTE 23232-323
Nome do usuário: ADMINISTRADOR
Email: QUALIDADE@LECOM.COM.BR

| Data | Tipo | Quantidade | Valor despesa | T o t a l D e s p e s a |
|------------|-------|------------|---------------|----------------------------|
| 2020-05-11 | TESTE | 2 | 150.00 | 300.00 |

Total de Despesas: XXX

Bauru,4,5,2020

Acima um exemplo de como fica o campo no processo e o pdf gerado.