PROJETO CUSTOMIZAÇÕES

Gera PDF Único com dados do processo

Amanda Alvim

1. Objetivo

O objetivo dessa customização é no final ter um PDF contendo dados específicos do processo e todos os anexos que esse processo teve, ou seja se foi anexado arquivos pdf, word, imagens todos estarão juntos em um documento só e quando fizer essa chamada retornará na tela para o usuário o arquivo final.

O controller contendo essa implementação se encontra no Projeto Base no nosso git (http://git.lecom.com.br/PSP/Projeto-Base-BPM) com o nome de **GerarPDFUnicoController**, que está no package com.lecom.workflow.cadastros.common.controller, nesse pacote estará os exemplos de controller que utilizamos.

2. Como usar

No momento em que for utilizar essa implementação será necessário alterar onde necessário para o seu modelo, e principalmente colocar o arquivo .class, que estará no diretório do seu workspace na pasta bin, ao pegar esse .class vamos ter que colocar no diretório direto do servidor do cliente, colocando essa classe exatamente no caminho do pacote (/opt/lecom/app/tomcat/webapps/bpm/classes/com/lecom/workflow/cadastros), esse caminho é o que sempre iremos utilizar para os nossos controllers dessa forma não colocamos em pacotes, caminhos que podem interferir na execução do produto, sendo assim o que tiver a frente do cadastros no seu pacote é necessário criar a pasta no diretório ficando (/opt/lecom/app/tomcat/webapps/bpm/classes/com/lecom/workflow/cadastros/commmon/controller), isso se estiver em servidor do formato antigo de atualização das customizações, para servidores com a configuração nova você colocará em /opt/lecom/custom/classes/commom/controller, após colocar nesses diretórios é necessário restart do ambiente.

3. Arquivo JAVA

Nesse arquivo temos exemplos de como vamos pegar os campos do processo pelo controller e como gerar um PDF com todos os anexos utilizando api do produto e no final juntar as duas coisas.

GerarPDFUnicoController.java

```
1  package com.lecom.workflow.cadastros.common.controller;
2
3  import java.io.BufferedReader;
58
59 @WebServlet("/app/public/geraPdfUnico")
60 public class GerarPDFUnicoController extends HttpServlet{
61
62     /**
63      *
64      */
65     private static final long serialVersionUID = 1L;
66     private static final Logger logger = Logger.getLogger(GerarPDFUnicoController.class);
67     private String codForm;
68     private String codVersao;
69     private final static String DOC_PATH = Funcoes.getWFRootDir() +"upload/cadastros/doc/";
70     private final static String IMG_PATH = Funcoes.getWFRootDir() + "upload/cadastros/img/logo.png";
71
72     public Connection getConexao() {
73         return DBUtils.getConnection("workflow");
74     }
75
76
77     @Override
78     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
79
80         String token = Funcoes.nulo(request.getParameter("token"), "");
81         String codProcesso = Funcoes.nulo(request.getParameter("codProcesso"), "");
82         String codEtapa = Funcoes.nulo(request.getParameter("codEtapa"), "");
83         String codCiclo = Funcoes.nulo(request.getParameter("codCiclo"), "");
84         String name = Funcoes.nulo(request.getParameter("name"), "");
85
86         if(!token.isEmpty() && !codProcesso.isEmpty() && !codEtapa.isEmpty() && !codCiclo.isEmpty()) {
87
88             String urlBPM = request.getScheme() + "://" + request.getServerName() + "/bpm";
89
90             buscarModelo(codProcesso);
91
92             if(!codForm.isEmpty() && !codVersao.isEmpty()) {
93                 String documento = geraDocumento(urlBPM,codProcesso, codEtapa, codCiclo,token,name);
94
95                 if(!documento.isEmpty()) {
96
97                     File file = new File(documento);
```

GerarPDFUnicoController.java

```
99                      response.setContentType("application/pdf");
100                     response.setHeader("Content-Disposition", "attachment; filename="+file.getName());
101                     InputStream in = new FileInputStream(file);
102                     OutputStream out = response.getOutputStream();
103                     byte[] buffer = new byte[4096];
104                     int length;
105                     while ((length = in.read(buffer)) > 0){
106                         out.write(buffer, 0, length);
107                     }
108                     in.close();
109                     out.flush();
110                     out.close();
111
112
113                 } else {
114
115                     response.setContentType("text/html");
116                     PrintWriter out = response.getWriter();
117                     out.println(mensagemErro());
118                     logger.error("[=== Ocorreu um erro: Nome do documento esta vazio ===]: ");
119                     System.out.println("[=== Ocorreu um erro: Nome do documento esta vazio ===]: ");
120
121                 }
122
123
124             }
125
126         }
127     }
```

Nas duas imagens acima é a parte essencial do controller, dela chamará vários métodos que iremos ver com calma cada um após exemplificar cada linha das imagens acima:

- Na **linha 59** é onde iremos informar que essa classe é um controller, nos mapearemos dentro da anotação @WebServlet o caminho a qual será chamado esse controller, sempre utilizamos o começo com app e depois podemos colocar o nome que quisermos, quando mapeamos com o nome app/public e o nome que desejar ser chamado, esse controller será acessado mesmo que o usuário não tiver logado na ferramenta, caso não tenha a palavra public ao tentar acessar ele solicitará que coloque usuário e senha do Lecom BPM.
- Na **linha 60** usamos o extends **HttpServlet** que é podermos utilizar os métodos padrões de um servlet do JAVA, que no exemplo usaremos o método **doGet**.

- Nas **linhas 65 a 70** estamos iniciando variáveis que iremos utilizar no decorrer da classe o log para logar o que está ocorrendo e conseguir avaliar caso tenha sucesso ou falha, o código do formulário, o código da versão , esses dois últimos iremos preencher no método buscarFormulario, e dois caminhos do servidor sendo um onde irá salvar os arquivos temporários e o que será gerado no final e o outro é o caminho do logo do cliente para que o documento final seja gerado com algo especifico do cliente.
- Nas **linhas 72 a 74** apenas um método que retorna a conexão com o banco do Lecom BPM.
- Nas **linhas 80 a 84** são os parâmetros que virão da chamada desse controller, iremos passar no caso por estarmos utilizando o doGet diretamente na url, vamos precisar receber os seguintes parâmetros:
    - token: token do usuário logado, mostrarei mais abaixo da onde você pode pegar.
    - codProcesso: código do processo que você deseja que seja gerado o pdf.
    - codEtapa: código da atividade.
    - codCiclo: código do ciclo.
    - name: nome do usuário que está executando essa atividade, ele será usado na marca d'agua que é gerado no documento final.
- Na **linha 86** está sendo validado se tem algum valor nos parâmetros acima, se caso algum não tiver com valor não será feito a geração do pdf.
- Na **linha 88** está pegando a partir do request a url do ambiente, e colocando o valor bpm na frente.
- Na **linha 90** está sendo chamado o método **buscaFormulario** que a partir do código de processo irá retornar o código do formulário e versão.
- Na **linha 92** está sendo validado se foi retornado corretamente o valor do código do formulário e código da versão.
- Na **linha 93** ele chamará o método **gerarDocumento** que irá unir um pdf gerado com informações simples do processo, com um documento que conterá todos os anexos do processo.
- Na **linha 95** é validado se o documento gerado está com valor, essa variável estará com o valor do caminho inteiro de onde o arquivo final foi gerado.
- Na **linha 97** como o retorno do método **gerarDocumento** é uma String eu preciso transformá-lo em um file para conseguir devolver para tela esse arquivo.
- Nas **linhas 99 a 110** está sendo atribuído um header para o response da chamada passando o nome do arquivo e escrevo o conteúdo do arquivo no output e esse arquivo é "cuspido" como download na tela do usuário.
- Nas **linhas 115 a 119** caso não tenha sido gerado corretamente o documento é retornado para a tela uma mensagem de erro padrão que veremos também o conteúdo do método **mensagemErro** mais para baixo.

GerarPDFUnicoController.java

```
129    private void buscarModelo(String codProcesso) {
130
131        StringBuilder query = new StringBuilder();
132        query.append("SELECT ");
133        query.append("PR.COD_FORM, ");
134        query.append("PR.COD_VERSAO ");
135        query.append("FROM PROCESSO PR ");
136        query.append("WHERE PR.COD_PROCESSO = ? ");
137
138        try(Connection con = getConexao()) {
139            try(PreparedStatement pst = con.prepareStatement(query.toString())) {
140
141                pst.setString(1, codProcesso);
142
143                try(ResultSet rs = pst.executeQuery()){
144
145                    while(rs.next()) {
146
147                        codForm = Funcoes.nulo(rs.getString("COD_FORM"), "");
148                        codVersao = Funcoes.nulo(rs.getString("COD_VERSAO"), "");
149
150                    }
151                }
152
153            }
154        }catch (Exception e) {
155            logger.error("Erro ao buscar o modelo [Controle de Documentos]: " + e);
156        }
157
158    }
```

Na imagem acima temos o conteúdo do método **buscarModelo** que irá na base de dados e retornará o código do formulário e código da versão a partir do código do processo.

GerarPDFUnicoController.java

```
160    @SuppressWarnings("resource")
161    private String geraDocumento(String urlBPM,String codProcesso,String codEtapa,String codCiclo,String token,String name) {
162        String documento = "";
163
164        LocalDateTime localDate = LocalDateTime.now();
165        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy-HH-mm-ss");
166        String data = localDate.format(formatter);
167
168        try(Connection con = getConexao()) {
169
170            Map<String, String> campos = setCampos(con, codProcesso, codEtapa, codCiclo);
171            ArrayList<Map<String, String>> materiais = getMateriais(con, codProcesso, codEtapa, codCiclo);
172
173            File capa = gerarCapa(campos, materiais);
174            File arquivos = buscarDocumentos(urlBPM, codProcesso, token, "true");
175
176            if(capa.exists() && arquivos.exists()) {
177
178                String arquivoJuntada = DOC_PATH+"dados-do-processo.pdf";
179                String arquivoJuntadaMarcaDagua = DOC_PATH+"dados-do-processo"+codProcesso+"-"+data+".pdf";
180
181                File fileJuntada = new File(arquivoJuntada);
182                File fileJuntadaMarcaDagua = new File(arquivoJuntadaMarcaDagua);
183
184                PdfReader readCapa = new PdfReader(capa.getPath());
185                PdfReader readAquivos = new PdfReader(arquivos.getPath());
186
187                PdfCopyFields copy = new PdfCopyFields(new FileOutputStream(fileJuntada));
188                copy.addDocument(readCapa);
189                copy.addDocument(readAquivos);
190                copy.close();
191
192                PdfReader readVistas = new PdfReader(fileJuntada.getPath());
193
194                // Gerar Marca d'agua
195                Font FONT = new Font(Font.NORMAL, 30, Font.BOLD, new GrayColor(0.7f));
196                Phrase phraseCopia = new Phrase("Cópia Controlada", FONT);
197                Phrase phrasePor = new Phrase("Gerado por: ", FONT);
198                Phrase phraseAutor = new Phrase(name, FONT);
199                PdfStamper stamper = new PdfStamper(readVistas, new FileOutputStream(fileJuntadaMarcaDagua));
200                PdfGState gstate = new PdfGState();
```

GerarPDFUnicoController.java

```
199                 PdfStamper stamper = new PdfStamper(readVistas, new FileOutputStream(fileJuntadaMarcaDagua));
200                 PdfGState gState = new PdfGState();
201                 // Adicionar Marca d'agua
202                 for(int i=1; i<= readVistas.getNumberOfPages(); i++){
203                     PdfContentByte content= stamper.getOverContent(i);
204                     gState.setFillOpacity(0.5f);
205                     content.setGState(gState);
206                     ColumnText.showTextAligned(content, Element.ALIGN_CENTER, phraseCopia, 300f, 450f, 30f);
207                     ColumnText.showTextAligned(content, Element.ALIGN_CENTER, phrasePor, 300f, 400f, 30f);
208                     ColumnText.showTextAligned(content, Element.ALIGN_CENTER, phraseAutor, 300f, 350f, 30f);
209                 }
210
211                 // Fechamento das classes de leitura
212                 stamper.close();
213                 readCapa.close();
214                 readAquivos.close();
215                 readVistas.close();
216
217                 // Exclusão de arquivos auxiliares
218                 capa.delete();
219                 arquivos.delete();
220                 fileJuntada.delete();
221
222                 documento = fileJuntadaMarcaDagua.getAbsolutePath();
223
224             } else {
225                 logger.error("Capa ou Anexos nao foram gerados!");
226             }
227         }catch (Exception e) {
228             logger.error("Erro ao buscar dados do processo/Modelo " + e);
229         }
230
231         return documento;
232     }
```

Nas imagens acima temos o conteúdo do método **geraDocumento** que irá realizar a geração do documento unificando o arquivo gerado dos dados do processo com o arquivo gerado com todos os anexos do processo.

Nas **linhas 164 a 166** estamos realizando a inicialização da data atual, e formatando para pegar dia, mês, ano, hora, minuto e segundo.

Na **linha 170** ele está chamando o método **setCampos**, que irá retornar os valores de alguns campos que serão especificados no select que está sendo feito, com esse map usará os valores para montar uma capa na geração final do documento, com dados específicos do processo, mesma coisa acontece na **linha 171** que está sendo chamado o método **getMateriais** onde nele está sendo feito um select para pegar todos os registros da grid materiais, será visto mais abaixo detalhes desses dois métodos.

Na **linha 173** é chamado o método **gerarCapa** onde será criado um arquivo com os dados dos campos normais e os dados da grid que desejar.

Na **linha 174** é chamado o método **buscarDocumentos**, onde será chamado uma api do produto que retorna em um arquivo todos os arquivos que foram anexados a partir do código de processo, etapa e ciclo.

Na **linha 176** é validado se os arquivos gerados nos métodos acima existem, se existirem significa que gerou corretamente e assim começa a junção deles.

Na **linha 178 e 179** está sendo criado duas variáveis com os arquivos que serão gerados, um sem marca d'agua e o outro com a marca d'agua logo após isso nas **linhas 181 e 182** está sendo instanciado o objeto file passando o caminho do arquivo.

Nas **linhas 184 e 185** estamos utilizando a api itext para realizar as leituras dos arquivos pdfs que foram gerados nas **linhas 173 e 174** e com isso inicializamos a classe PdfReader passando o caminho desses arquivos.

Nas **linhas 187 a 190**, está sendo instanciada a classe **PdfCopyFields** e no construtor dela passamos um **Fileoutputstream** que seria dizer que a saída desse **PdfCopyFields** será nesse

caminho passado no construtor do **fileoutputstream** e assim nas **linhas 188 e 189** está sendo feito uma adição dos dois arquivos gerados: o de capa e o de todos os anexos.

Na **linha 192** instancio a classe PdfReader passando o caminho do arquivo gerado com a cópia dos outros, caminho esse que foi definido na **linha 181.**

Nas **linhas 195 a 200** está sendo utilizados classes da api itext para geração, leitura de arquivos PDF's, assim ele está criando uma fonte com o tipo e tamanho dela, e frases para sem escritas como marca d'agua no documento, então quando fizermos a chamada e passar o nome do usuário que está gerando, é esse nome que será gerado na marca d'agua.

Nas **linhas 202 a 209** ele está lendo todas as páginas do pdf gerado e em cada uma delas escrevendo a marca d'agua.

Nas **linhas 212 a 220** está sendo feito o fechamento de cada objeto instanciado para que não fique aberto em memoria e assim poderia dar erro de java heap space e outros erros no servidor, por isso realizamos esse fechamentos, alguns desses objetos daria até para usar o formato try/resource para que o próprio java cuidasse dessa parte de abertura e fechamento dessas classes.

Na **linha 222** configura-se o retorno do método para ser o caminho absoluto do arquivo com a marca d'agua gerado e esse caminho é usado na **linha 93** explicado no começo desse documento, sendo usado esse caminho para "jogar" para tela o download do arquivo.

GerarPDFUnicoController.java

```
233
234     private Map<String, String> setCampos(Connection con, String codProcesso, String codEtapa, String codCiclo) {
235
236         Map<String, String> campo = new HashMap<>();
237
238         StringBuilder query = new StringBuilder();
239         query.append("SELECT ");
240         query.append("TD.NOME, ");
241         query.append("TD.TIPO_SOLICITACAO, ");
242         query.append("DATE_FORMAT(TD.DT_PREVISAO_NECESSIDADE,'%d/%m/%Y') as datContratacao, ");
243         query.append("TD.BUDGET, ");
244         query.append("DATE_FORMAT(PE.DAT_GRAVACAO, '%d/%m/%Y') AS DAT_GRAVACAO ");
245         query.append("FROM f_lec_m_comp TD, PROCESSO_ETAPA PE ");
246         query.append("WHERE TD.COD_PROCESSO_F = PE.COD_PROCESSO ");
247         query.append("AND PE.COD_ETAPA = TD.COD_ETAPA_F ");
248         query.append("AND PE.COD_CICLO = TD.COD_CICLO_F ");
249         query.append("AND TD. COD_PROCESSO_F = ? ");
250         query.append("AND TD.COD_ETAPA_F = ? ");
251         query.append("AND TD.COD_CICLO_F = ? ");
252         try(PreparedStatement pst = con.prepareStatement(query.toString())) {
253             pst.setString(1, codProcesso);
254             pst.setString(2, codEtapa);
255             pst.setString(3, codCiclo);
256             try(ResultSet rs = pst.executeQuery()){
257                 while(rs.next()) {
258                     campo.put("nome", Funcoes.nulo(rs.getString("NOME"), ""));
259                     campo.put("tipo", Funcoes.nulo(rs.getString("TIPO_SOLICITACAO"), ""));
260                     campo.put("dataPrevisao", Funcoes.nulo(rs.getString("datContratacao"), ""));
261                     campo.put("budget", Funcoes.nulo(rs.getString("BUDGET"), ""));
262                     campo.put("data", Funcoes.nulo(rs.getString("DAT_GRAVACAO"), ""));
263                 }
264             }
265         } catch(Exception e) {
266             logger.info("Erro ao buscar campos: " + e);
267         }
268
269         return campo;
270     }
```

Conforme mencionado na explicação do método acima de gerarDocumento, precisamos no método **setCampos,** gerar um Map com alguns campos específicos que desejamos que apareça na capa do documento final.

Nas **linhas 238 a 251** está sendo montado um select na tabela do modelo em questão realizando um join com a tabela processo etapa para pegar dados do processo, etapa , ciclo passado, assim nas **linhas 252 a 264** é executado a query, e cada linha retornada está sendo colocado no map que será retornado, como será para um processo, etapa, ciclo somente um

registro será retornado. Nesse caso estou pegando Nome, Tipo, uma data, um campo monetário e a data do início dessa atividade retornada.

GerarPDFUnicoController.java

```
272    private ArrayList<Map<String, String>> getMateriais(Connection con, String codProcesso, String codEtapa, String codCiclo) {
273
274        ArrayList<Map<String, String>> materiais = new ArrayList<>();
275
276        StringBuilder query = new StringBuilder();
277        query.append("SELECT ");
278        query.append("RC.MATERIAL, ");
279        query.append("RC.QTD, ");
280        query.append("RC.UNIDADE_MEDIDA ");
281        query.append("FROM g_lec_m_compG_MATERIAL RC ");
282        query.append("WHERE RC.COD_PROCESSO = ? ");
283        query.append("AND RC.COD_ETAPA = ? ");
284        query.append("AND RC.COD_CICLO = ? ");
285        try(PreparedStatement pst = con.prepareStatement(query.toString())) {
286            pst.setString(1, codProcesso);
287            pst.setString(2, codEtapa);
288            pst.setString(3, codCiclo);
289            try(ResultSet rs = pst.executeQuery()){
290                while(rs.next()) {
291                    Map<String, String> dados = new HashMap<>();
292                    dados.put("material", Funcoes.nulo(rs.getString("MATERIAL"), ""));
293                    dados.put("qtd", Funcoes.nulo(rs.getString("QTD"), ""));
294                    dados.put("unidade", Funcoes.nulo(rs.getString("UNIDADE_MEDIDA"), ""));
295
296                    materiais.add(dados);
297                }
298            }
299        } catch(Exception e) {
300            logger.info("Erro ao buscar representante: " + e);
301        }
302
303        return materiais;
304    }
```

Conforme mencionado na explicação do método acima de **gerarDocumento**, precisamos no método **getMateriais,** gerar uma Lista de Map da grid com alguns campos específicos que desejamos que apareça na capa do documento final.

Nas **linhas 276 a 284** está sendo feito um select na tabela da grid, nesse exemplo a tabela do meu modelo chama f_lec_m_comp e o identificador da minha grid g_material por isso a tabela a qual faremos o select será g_lec_m_compG_MATERIAL para o código do processo, etapa e ciclo pegando os campos da grid que são material, quantidade e unidade, com essas informações nas **linhas 285 a 298** é executado a query e montado a lista de map com esse valores para com essas informações serem geradas a capa do documento final.

GerarPDFUnicoController.java

```
305
306    private File gerarCapa(Map<String, String> campo, ArrayList<Map<String, String>> representante) {
307        File retorno = null;
308
309        try {
310
311            File arquivo = new File(DOC_PATH + "vistas.pdf");
312
313            OutputStream output = new FileOutputStream(arquivo);
314            Document doc = new Document();
315            PdfWriter writer = PdfWriter.getInstance(doc, output);
316
317            doc.open();
318            InputStream input = new ByteArrayInputStream(conteudo(campo, representante).getBytes());
319            XMLWorkerHelper.getInstance().parseXHtml(writer, doc, input);
320
321            doc.close();
322            input.close();
323            output.close();
324
325            retorno = arquivo;
326
327        } catch (Exception e) {
328
329            logger.error("[== Erro ao gerar a capa ==]: " + e);
330
331        }
332
333        return retorno;
334    }
```

Conforme mencionado na explicação do método acima de **gerarDocumento**, precisamos no método **gerarCapa,** criar um arquivo com os dados básicos do modelo que no final será juntado ao arquivo com todos os anexos do processos, etapa, ciclo.

Para esse método nós iremos passar os retornos dos métodos **setCampos e getMateriais**, na **linha 311** ele só definiu um nome para ser gerado esse arquivo de capa, passando o caminho do arquivo.

Nas **linhas 313 a 323** está sendo utilizado classes do java e do itext para realizar a criação de um arquivo pdf a partir de um html, por isso na **linha 319** é feito um **parseXHtml** , na linha anterior **318**, é chamado o próximo método que iremos explicar **conteúdo**, nele será montado um html com todos os campos e dados da grid passado.

GerarPDFUnicoController.java

```
336    private static String conteudo(Map<String, String> campo, ArrayList<Map<String, String>> materiais) {
337
338        LocalDate localDate = LocalDate.now();
339        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
340        String hoje = localDate.format(formatter);
341
342
343        StringBuilder header = new StringBuilder();
344
345        header.append(" <html> ");
346        header.append(" <head> ");
347        header.append("     <style> ");
348        header.append("         p { font-size: 14px; padding-right: 15px; }");
349        header.append("         h3 { text-align: center; }");
350        header.append("         h4 { text-align: center; }");
351        header.append("         label { font-weight: bold; }");
352        header.append("         .linha { margin: 5px; padding: 0px }");
353        header.append("         .grids { padding-bottom: 10px; }");
354        header.append("     </style> ");
355        header.append(" </head> ");
356        header.append(" <body> ");
357        header.append("     <table width='100%'> ");
358        header.append("         <thead> ");
359        header.append("             <tr> ");
360        header.append("                 <th> ");
361        header.append("                     <img src='"+IMG_PATH+"' width='100px'></img> ");
362        header.append("                 </th> ");
363        header.append("                 <th style='vertical-align: top'> ");
364        header.append("                     <p align='right'>"+hoje+"</p> ");
365        header.append("                 </th> ");
366        header.append("             </tr> ");
367        header.append("         </thead> ");
368        header.append("     </table> ");
369        header.append("     <br /> ");
370        header.append("     <br /> ");
371        header.append("     <div width='100%'> ");
372        header.append("         <h3>Exemplo Geração PDF UNICO</h3>");
373        header.append("     </div> ");
374        if(!campo.get("nome").isEmpty()) header.append("<p class='linha'><label>Nome: </label>"+campo.get("nome").toUpperCase()+"</p>");
375        if(!campo.get("tipo").isEmpty()) header.append("<p class='linha'><label>Tipo: </label>"+campo.get("tipo").toUpperCase()+"</p>");
376        if(!campo.get("dataPrevisao").isEmpty()) header.append("<p class='linha'><label>Data Previsão: </label>"+campo.get("dataPrevisao").toUpperCase()+"</p>");
377        if(!campo.get("budget").isEmpty()) header.append("<p class='linha'><label>Budget: </label>"+campo.get("budget").toUpperCase()+"</p>");
378        if(!campo.get("data").isEmpty()) header.append("<p class='linha'><label>Data da abertura: </label>"+campo.get("data").toUpperCase()+"</p>");
379        if(!materiais.isEmpty()) {
```

GerarPDFUnicoController.java

```
378        if(!campo.get("data").isEmpty()) header.append("<p class='linha'><label>Data da abertura: </label>"+campo.get("data").toUpperCase()+"</p>");
379        if(!materiais.isEmpty()) {
380            header.append("     <div width='100%'> ");
381            header.append("         <h4>Dados dos Materiais</h4>");
382            header.append("     </div> ");
383            materiais.forEach(map ->{
384                header.append("<div class='grids'>");
385                if(!map.get("material").isEmpty()) header.append("  <p class='linha'><label>Material: </label>" + map.get("material").toUpperCase() + "</p>");
386                if(!map.get("qtd").isEmpty()) header.append("   <p class='linha'><label>Quantidade: </label>" + map.get("qtd").toUpperCase() + "</p>");
387                if(!map.get("unidade").isEmpty()) header.append("   <p class='linha'><label>Unidade: </label>" + map.get("unidade").toUpperCase() + "</p>");
388                header.append("</div>");
389            });
390        }
391        header.append(" </body> ");
392        header.append("</html> ");
393
394        return header.toString();
395
396    }
```

Para esse método nós iremos passar os retornos dos métodos **setCampos e getMateriais** e a partir da **linha 343** é iniciado um StringBuilder com todo o conteúdo do html , montando em tabelas os dados, colocando um logo, a data de hoje e nas **linhas 374 a 378** ele valida se tem valor para aquele campo e monta uma tag p para colocar um label e o valor do campo.

A partir da **linha 379** é validado se existem itens na grid e assim coloca um título e realiza um foreach e vai preenchendo cada linha da grid dentro de uma div.

GerarPDFUnicoController.java

```
397
398     @SuppressWarnings("resource")
399     public File buscarDocumentos(String urlBPM, String codProcesso, String token, String currentVersion) {
400
401         logger.info("[=== Juntada ===]");
402
403         File arquivo = null;
404
405         if(!codProcesso.isEmpty() && !token.isEmpty() && !currentVersion.isEmpty()) {
406
407             StringBuilder url = new StringBuilder();
408             url.append(urlBPM);
409             url.append("/api");
410             url.append("/v1");
411             url.append("/process-instances");
412             url.append("/" + codProcesso);
413             url.append("/export-attachments-to-pdf?");
414             url.append("current-version=" + currentVersion);
415
416             try {
417                 HttpClient client = HttpClientBuilder.create().build();
418                 HttpGet get = new HttpGet(url.toString());
419                 get.addHeader("ticket-sso",token);
420                 HttpResponse response = client.execute(get);
421
422                 BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
423
424                 StringBuilder result = new StringBuilder();
425                 String line = "";
426                 while ((line = rd.readLine()) != null) {
427                     result.append(line);
428                 }
429
430                 org.json.JSONObject json = new org.json.JSONObject(result.toString());
431
```

GerarPDFUnicoController.java

```
432                 if (response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK) {
433
434
435                     String body = (String) json.get("body");
436
437                     //logger.debug("[JUNTADA BODY]: " +  body);
438
439                     byte[] decodedBody = Base64.getDecoder().decode(body.getBytes(StandardCharsets.UTF_8));
440                     Path destination = Paths.get(DOC_PATH + "juntada.pdf");
441
442                     logger.debug(destination.getFileName());
443
444                     arquivo = Files.write(destination, decodedBody).toFile();
445
446                     logger.debug(arquivo.getAbsoluteFile());
447
448                 } else {
449                     JSONArray jsonArray = (JSONArray) json.get("errors");
450                     JSONObject jsonObject = jsonArray.getJSONObject(0);
451
452                     logger.error("Status: " + response.getStatusLine().getStatusCode() + " - " + jsonObject);
453                 }
454
455             } catch (Exception e) {
456                 e.printStackTrace();
457                 logger.error("Erro ao executar [Juntada]: " , e);
458                 System.out.println("Erro ao executar [Juntada]: " + e);
459             }
460         }
461
462         return arquivo;
463
464     }
```

Para esse método **buscarDocumentos** nós iremos passar a url do BPM, o token passado, o código do processo, e a variável currentVersion, que iremos sempre passar como true.

Nas **linhas 407 a 414** está sendo montado a url da api do produto que retorna em um pdf todos os arquivos anexados nessa instancia de processo passado, pegando sempre a última versão dos arquivos.

Nas **linhas 417 a 420** está sendo utilizado a api httpclient para realizar o consumo dessa api, no caso essa url é chamada via Get, por isso utilizamos o HTTPGet para realizar essa chamada passando como header da chamada o token do usuário logado, e no final de fato chamamos o método de execute, que irá executar e nos retornar algo no objeto HttpResponse.

Nas **linhas 422 a 430**, estou transformando o que veio no HttpResponse para um JSON para que possamos pegar o valor ou o erro retornado conforme verá nas próximas linhas.

Na **linha 432** é validado se o retorno da execução foi sucesso, se tiver sido um sucesso na **linha 435** pegamos do json o atributo body e nesse atributo body está sendo retornado em bytes o arquivo gerado com todos os anexos do processo, por isso realizamos nas **linhas 439 a 444** a decodificação desse byte para ser transformado em um arquivo físico do servidor e conseguirmos utilizarmos no método anterior de **gerarDocumentos** e juntar com o arquivo de capa gerado.

Nas **linhas 449 a 452** está pegando os erros se a execução não teve sucesso e está logando dentro do arquivo, para caso dê algum erro consiga identificar onde que foi a falha.

```
465
466     private String mensagemErro() {
467
468         StringBuilder msg = new StringBuilder()
469         .append("<html>")
470         .append("   <head>")
471         .append("       <title>Erro</title>")
472         .append("   </head>")
473         .append("   <body>")
474         .append("       <h1 style='text-align: center;  font-size: 40px; color: red; margin-top:100px'>ERRO!</h1>")
475         .append("       <p style='text-align: center; font-size: 20px;'>Ocorreu um erro ao executar <strong>Vistas do Processo</strong>. ")
476         .append("       Favor entrar em contato com o administrador do sistema.</p>")
477         .append("   </body>")
478         .append("</html>");
479
480         return msg.toString();
481     }
```

Na imagem acima temos o método de mensagemErro, que no primeiro método de execução da chamada caso tenha alguma falha ele vai e retorna para tela do usuário esse html definido nas **linhas 468 a 478.**

4. Chamada do método

Como visto acima o arquivo de controller ele tem as suas definições e particularidades, esse exemplo de geração de PDF Único com dados e anexos do processo ele deve ser chamado via JS do modelo, ou seja poderia ser criado no processos um campo do tipo botão aplicação externa, e esse campo será mostrado em todas as etapas por exemplo, para isso no javascript do modelo você deveria fazer a seguinte função utilizando esse campo de botão aplicação externa.

exemploChamadaController.js | GerarPDFUnicoController.java

```
1  /**
2   *
3   */
4  $(document).ready(function () {
5      gerarPdfUnico();
6  });
7
8
9  function gerarPdfUnico() {
10
11     Form.actions("BTN_PROCESSO").type("");
12     Form.actions("BTN_PROCESSO").url("");
13
14     var codProcesso = ProcessData.processInstanceId;
15     var codEtapa = ProcessData.activityInstanceId;;
16     var codCiclo = ProcessData.cycle;
17     var token = buscarToken();
18     var url = window.location.protocol+"//"+window.location.hostname+"/bpm/app/public/geraPdfUnico?token="+token+"&codProcesso="+codProcesso+"&codEtapa="+codEtapa;
19     url += "&codCiclo="+codCiclo+"#tversion=true&name="+Form.fields("USUARIO_LOGADO").value();
20
21     $("#"+Form.actions("BTN_PROCESSO").name()).on("click", function(e){
22         window.open(url,"_blank");
23     });
24
25 }
26
27 function buscarToken(){
28
29     var token = "";
30     var cookies = document.cookie.split(";");
31
32     cookies.forEach(function(cookie){
33         if(cookie.includes("LecomSSOTicket")) {
34             token = cookie.split("=")[1];
35         }
36     })
37
38     return token;
39 }
```

O arquivo acima se encontra no mesmo projeto que o controller explicado no diretório, upload/rotinaJs/exemploChamadaController.js, nesse arquivo que o conteúdo estaria no arquivo do seu modelo.

Na **linha 11 e 12** estamos pegando o campo criado como botão de aplicação externa e limpando o valor da url e type dele, pois iremos preencher o conteúdo manualmente.

Nas **linhas 14 a 16** estamos pegando direto da api do FormApp os dados de código do processo, atividade e ciclo**.**

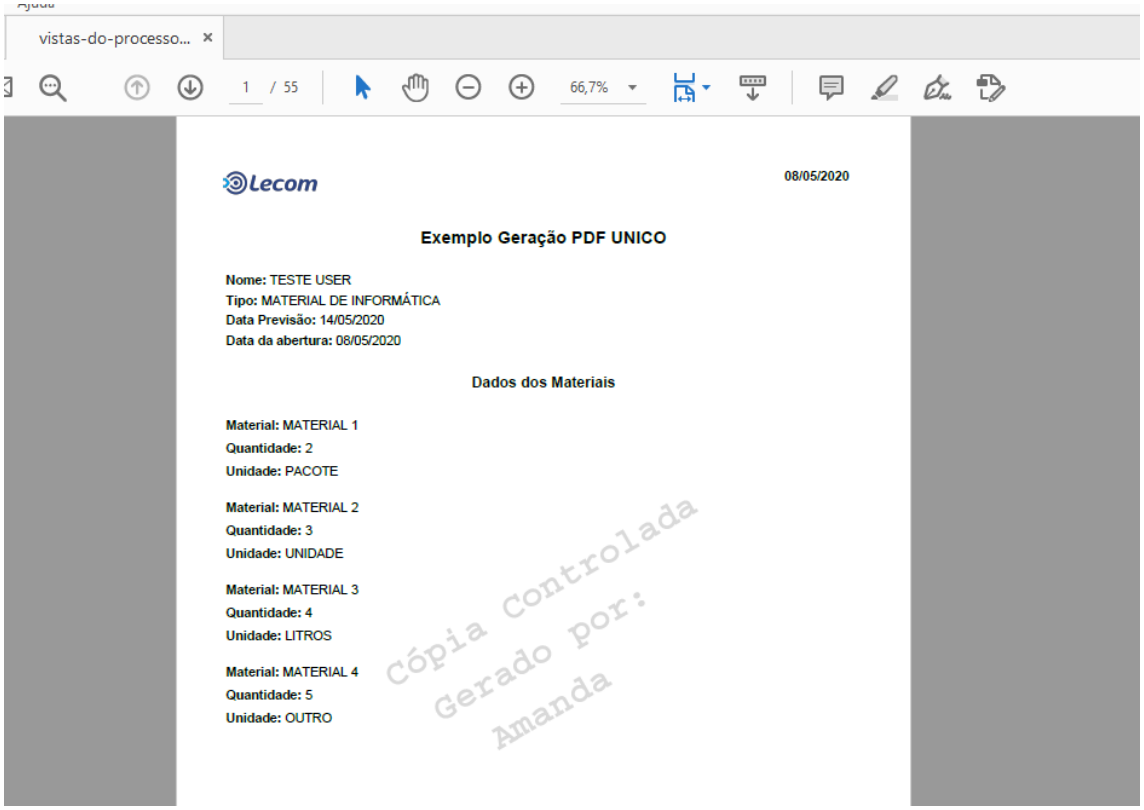
Na **linha 17** ele chama diretamente a outra função que está nesse arquivo chamada buscarToken, ela está pegando direto do navegador o cookie do token do SSO que vai vir o token do usuário logado, para que possamos utilizar essa informação na geração do documento a partir da api do produto conforme vimos acima.

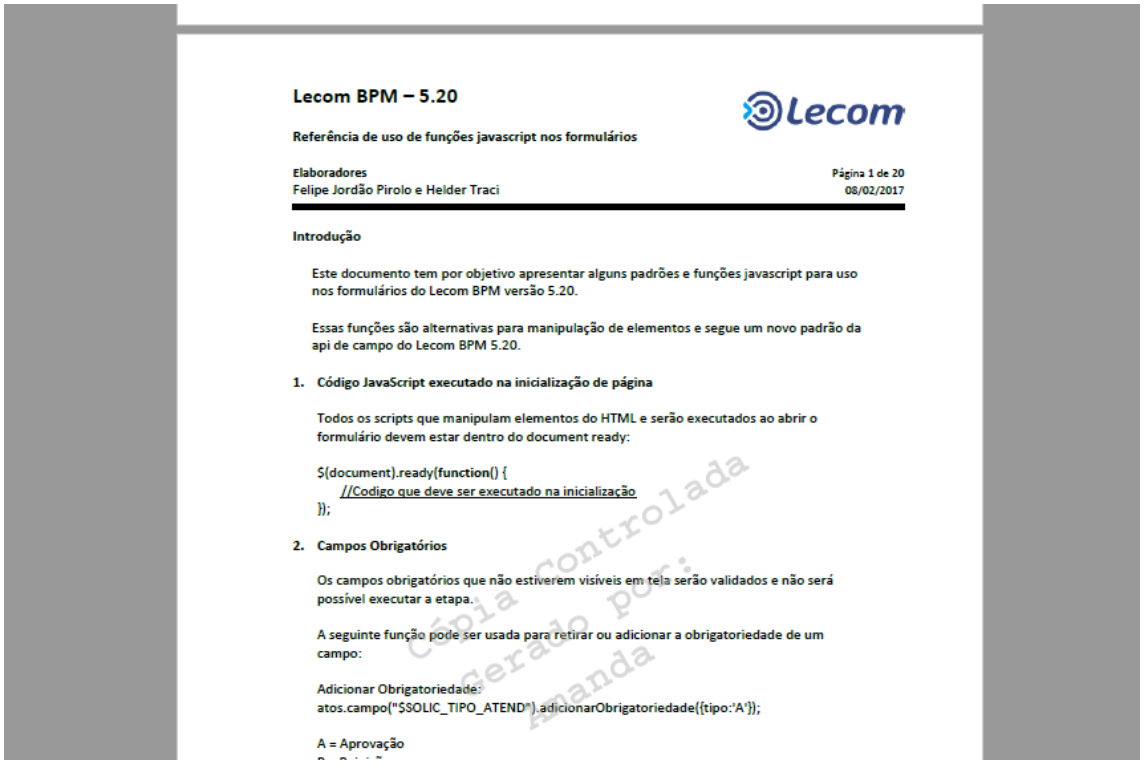
Na **linha 18 e 19** está sendo montado a url do controller, então começamos pegando o protocol e o hostname do cliente e complementamos com o /bpm e o mapeamento que colocamos na **linha 59** do arquivo java e passamos via get os parâmetros que precisamos código do processo, atividade, ciclo, token, currentVersion com valor fixo de true, e o nome, esse nome estamos pegando de um campo do modelo USUARIO_LOGADO, esse campo ele tem como valor inicial apenas a variável de ambiente contendo o nome do usuário logado para que possamos usar essa informação na marca d'agua gerada no documento final.

Nas **linhas 21 a 23** está sendo trocado a ação do click desse botão de aplicação externa, assim na hora que clicar nele iremos fazer a chamada do window.open para a url que inicializamos pedindo para abrir no _blank que significa chamar esse url em uma outra aba do navegador, como o retorno da nossa chamada é um arquivo ele automaticamente irá fazer o download do arquivo final no navegador do usuário.

5. Arquivo Final gerado

Depois de toda essa explicação do fonte como que foi gerado esse arquivo de fato? Vamos ver alguns prints abaixo, lembrando que são exemplos apenas de valores de acordo com o que nós traremos nos selects.





Acima a primeira e segunda página do documento final, a primeira nossa capa gerada e a segunda é a primeira página de um documento anexado no processo, todas tem essa marca d'agua com a frase que escrevemos no fonte java, cópia controlada gerado por: nome do usuário logado que chamou o método.