



PROJETO CUSTOMIZAÇÕES

Integração com D4Sign

Amanda Alvim

1. Objetivo

O objetivo desse documento é demonstrar e explicar como realizar uma integração utilizando o D4Sign.

O D4sign, é uma plataforma online para assinaturas digitais validas sendo assim nossos clientes poderiam fazer o cadastro na plataforma e nós entramos com a integração do Lecom BPM.

2. Como usar

Para utilizar essa integração no Projeto Base (<http://git.lecom.com.br/PSP/Projeto-Base-BPM>) tem 4 arquivos que compõe esse exemplo, e assim você poderá copiar os arquivos e colocar os dados no properties de acordo com o cadastro criado no D4Sign e utilizar no processo do BPM.

A utilização dentro do Lecom BPM funciona melhor sendo uma integração que na aprovação de uma atividade gerará conteúdo no D4sign e um robô posteriormente para aguardar as assinaturas feitas no D4sign e trazer o arquivo assinado para o processo é esse mecanismo e exemplo que veremos aqui.

No projeto base também já tem um jardesc criado para a geração do jar da integração que gera o .Java dela mais um arquivo utilitário e ele está na pasta jardesc do projeto com o mesmo nome da integração.

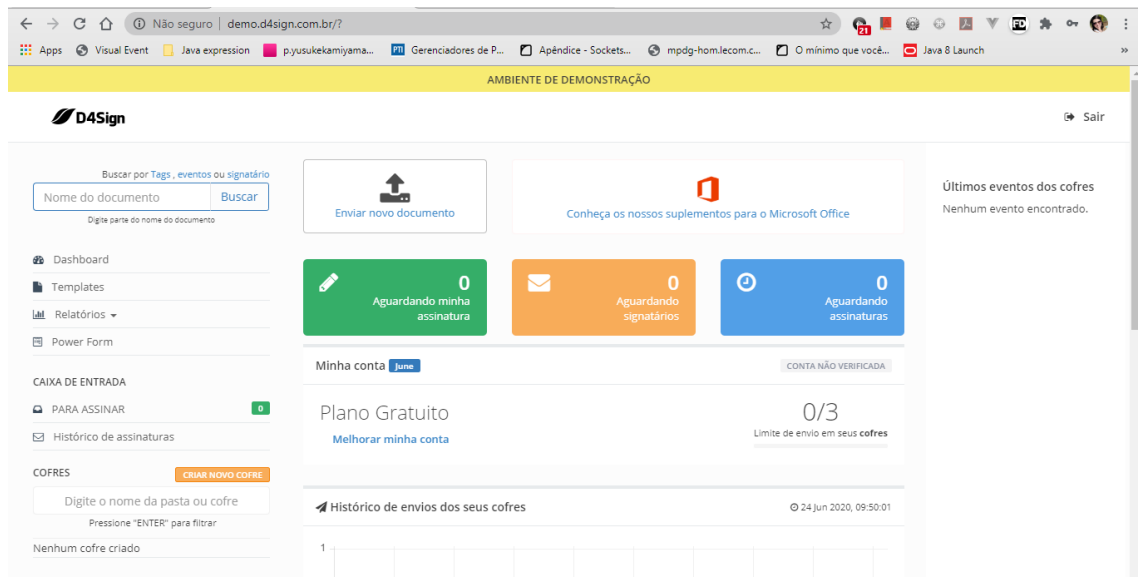
3. Iniciando D4Sign

Para conhecer um pouco mais sobre a plataforma o site dela <https://d4sign.com.br/> terá mais explicações sobre como funciona.

Dentro do nosso exemplo aqui utilizaremos o demo para criar uma conta e utilizá-la.

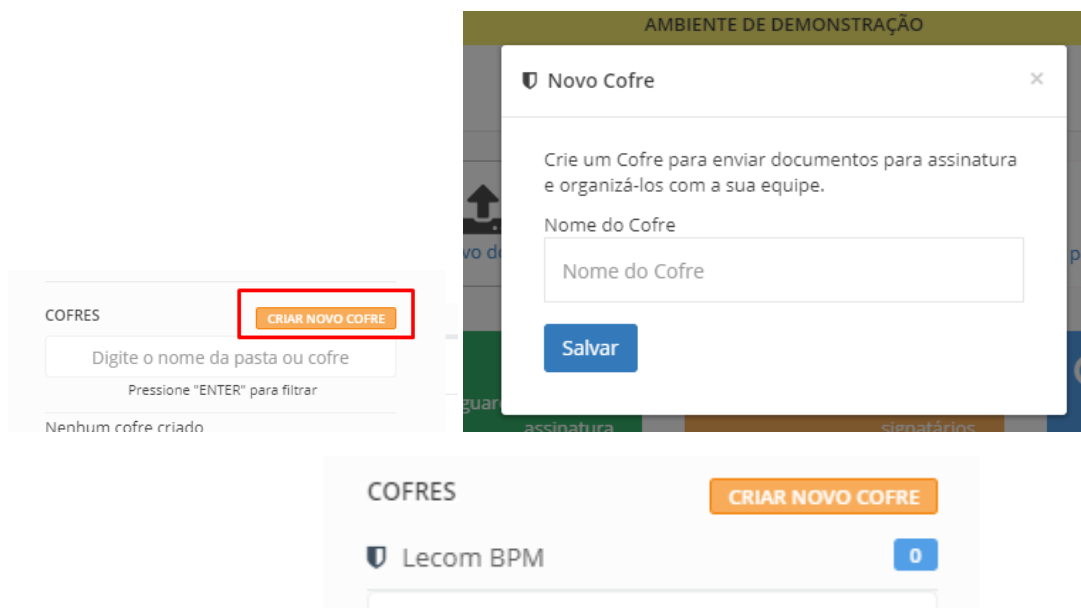
No site <http://demo.d4sign.com.br/>, optaremos por criar uma conta gratuita, após preencher seu e-mail você receberá um e-mail para complementar esse cadastro e criar uma senha de sua preferência, e também escolherá o plano gratuito, o plano gratuito nos dá direito a enviar 3 documentos apenas.

Depois de ter feito esse cadastro ao logar na ferramenta apenas para conhecimento vocês terão uma tela como essa abaixo:

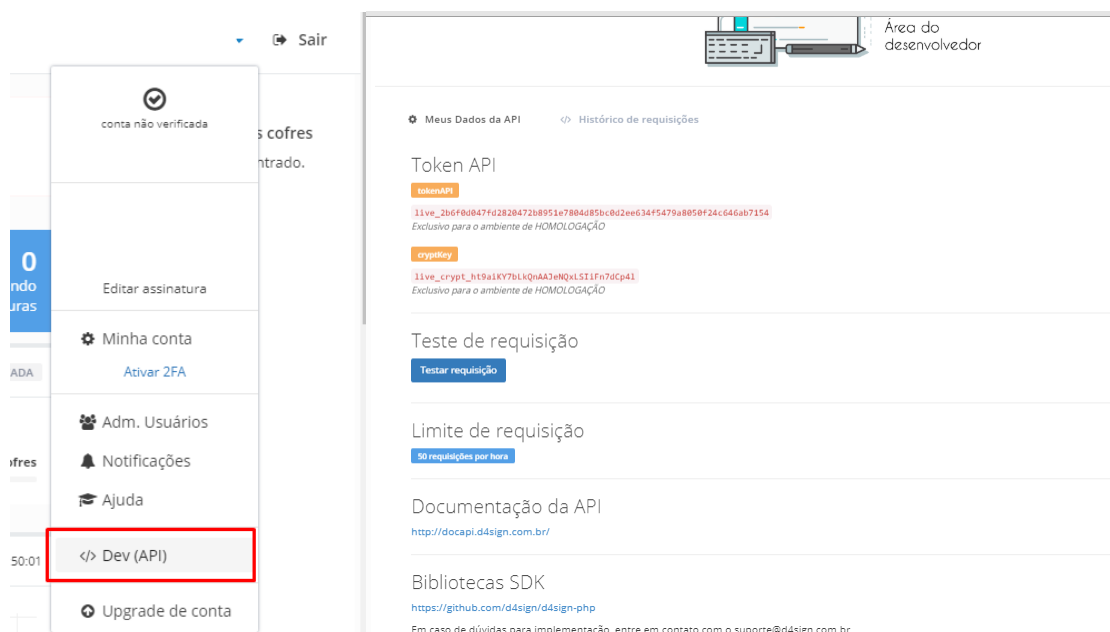


Nessa tela temos a informação de quantos documentos tem para assinar, quantos estão esperando cadastro de pessoas para assinar e quantos aguardando assinatura, são alguns status dos documentos dentro da plataforma.

Uma coisa importante que precisamos fazer quando inicia o cadastro é realizar o cadastro de um Cofre, esse cofre será usado diretamente na integração pois ao subir um documento nós o subimos dentro de um cofre e precisamos desse nome, dessa informação na nossa integração.



Chamei meu cofre de **Lecom BPM** para utilizar no nosso exemplo, temos outras informações que vamos precisar para realizar nossa integração, para isso entraremos no seu menu do canto direito e escolher a opção Dev(API), nessa tela terá a parte de desenvolvedor, com a documentação da api e as chaves **tokenApi** e **cryptKey**, são essas duas que usaremos na nossa integração além do nome do Cofre.

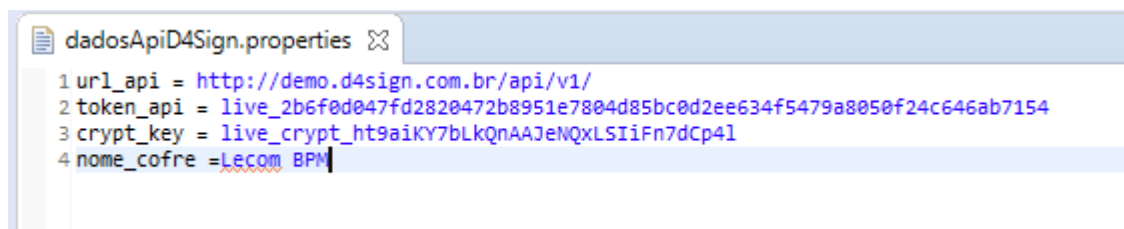


Agora vamos ver como utilizamos essa informação na integração e robô, depois veremos como aparece no site do D4sign para assinar.

4. Classes Java

Todas as classes utilizada nesse exemplo estão no pacote com.lecom.workflow.d4sign e no properties está na pasta upload/cadastros/config.

a. Properties



No properties acima temos 4 parâmetros que precisamos preencher:

- **url_api:** Url da api que chamaremos, no caso como exemplificado estamos usando o demo. d4sign, tendo que ser trocada quando for apontar para a licença de produção do cliente.
- **token_api:** A chave do token api a ser copiado da página de Dev (Api) da licença do cliente que vimos no tópico acima.
- **crypt_key:** A chave de cryptkey a ser copiado da página de Dev (Api) da licença do cliente que vimos no tópico acima.
- **nome_cofre:** Nome do Cofre criado que iremos “subir” os arquivos.

b. Integração

A classe que iremos ver e entender o fonte é a InsereArquivoD4Sign.

```
InserArquivoD4Sign.java
Projeto Base | src | com.lecom.workflow.d4sign | InserArquivoD4Sign | enviaEmailsAssinaturas(String, String, String, String): String

44 @IntegrationModule("InserArquivoD4Sign")
45 @Version({1,0,5})
46 public class InserArquivoD4Sign {
47
48     private static final Logger logger = Logger.getLogger(InserArquivoD4Sign.class);
49     private String configPath = Funcoes.getWfRootDir() + "/upload/cadastrados/config/";
50
51     @SuppressWarnings("unchecked")
52     @Execution
53     public String inserArquivo(IntegracaoVO integracaoVO) {
54         logger.info("INICIO Integração de Envio de Documento para Assinatura D4Sign.");
55         String retorno = "";
56         Map<String,String> camposEtapa = integracaoVO.getMapCamposFormulario();
57
58         try {
59             String arquivo = "";
60             String anexo1 = camposEtapa.get("SANEEXO");
61             String nomeArquivo = nomeArquivoAnexo(anexo1);
62             String caminhoAnexoTemporario = "";
63             String caminhoRaizArquivos = Funcoes.getWfRootDir()+File.separator+"upload"+File.separator+"cadastrados"+File.separator+"tempFiles"+File.separator;
64             if(!new File(caminhoRaizArquivos).exists()) {
65                 new File(caminhoRaizArquivos).mkdir();
66             }
67             InputStream input = getWfFilePath(anexo1);
68             caminhoAnexoTemporario = UtilECM.getOutputStreamDiretorio(input, caminhoRaizArquivos, nomeArquivo);
69             byte[] bytes = Files.readAllBytes(new File(caminhoAnexoTemporario).toPath());
70             arquivo = Base64.getEncoder().encodeToString(bytes);
71
72             Map<String, String> dadosConf = Funcoes.getParametrosIntegracao(configPath + "dadosApiD4Sign");
73             String urlApi=dadosConf.get("url_api");
74             String tokenApi=dadosConf.get("token_api");
75             String cryptKey=dadosConf.get("crypt_key");
76             String nomeCofre=dadosConf.get("nome_cofre").trim();
77
78             String uuidCofre = getUuidCofre(urlApi, tokenApi, cryptKey, nomeCofre);
79             logger.debug("uuidCofre = "+uuidCofre);
80             if(!uuidCofre.equals("Erro")) {
81                 String uuidDocumento = enviarDocumento(caminhoAnexoTemporario,arquivo, uuidCofre, nomeArquivo,tokenApi,cryptKey,urlApi);
82                 logger.debug("uuidDocumento = "+uuidDocumento);
83                 if(!uuidDocumento.equals("Erro")) {
84                     new File(caminhoAnexoTemporario).delete(); //deletando arquivo temporario apos upload no D4sign
85                     List<Map<String,Object>> assinantes = integracaoVO.getDadosModeloGrid("ASSINANTES");
```

```
InserArquivoD4Sign.java
Projeto Base | src | com.lecom.workflow.d4sign | InserArquivoD4Sign | atualizaRelacionamento(IntegracaoVO, String, String): String

86 String retornoCadastroAssinantes = cadastraAssinantes(urlApi, tokenApi, cryptKey, uuidDocumento, assinantes);
87 logger.debug("retornoCadastroAssinantes = "+retornoCadastroAssinantes);
88 if(retornoCadastroAssinantes.equals("OK")) {
89     String retornoEnvioAssinatura = enviaEmailsAssinaturas(urlApi, uuidDocumento, tokenApi, cryptKey);
90     logger.debug("retornoEnvioAssinatura = "+retornoEnvioAssinatura);
91     if(retornoEnvioAssinatura.equals("OK")) {
92         retorno = atualizaRelacionamento(integracaoVO, retorno, uuidDocumento);
93     }else {
94         retorno = "99|Erro ao enviar para assinatura, informe o administrador do sistema para validar";
95     }
96 }else {
97     retorno = "99|Erro ao cadastrar os assinantes, informe o administrador do sistema para validar";
98 }
99 }else {
100     retorno = "99|Não foi possível realizar o upload do arquivo no D4sign, informe o administrador do sistema para validar";
101 }
102 }else {
103     retorno = "99|Não foi possível recuperar o id do cofre, informe o administrador do sistema para validar";
104 }
105 }catch (Exception e) {
106     e.printStackTrace();
107     logger.error("Erro ao executar a integração",e);
108     retorno = "99|Erro ao executar a integração, informe o administrador do sistema para validar";
109 }
110
111 return retorno;
112 }
```

As imagens acima são do método principal da classe.

Na **linha 44,45** estamos definindo as anotações referente a informar que essa classe é do tipo integração e qual versão dessa classe atualmente.

Na **linha 48** estamos iniciando a variável de log para que possamos avaliar depois a execução da integração nos logs da aplicação.

Na **linha 49** inicializamos o caminho de onde irá buscar o properties.

Nas **linhas 59 a 61**, estamos pegando o valor do campo ANEXO, criado em nosso formulário para que o usuário input o arquivo a ser assinado no d4sign, após pegar esse valor do campo nos chamamos função para pegar somente o nome do arquivo inputado (**linha 61**).

Nas **linhas 63 a 66** estamos verificando se existe uma pasta temporária no servidor se não existir nós criamos para que possamos transformar nosso InputStream, tipo retornado pela api do ecm a partir do valor único que é salvo no campo (**linha 67**).

Na **linha 68** chamamos um utilitário para transformar nosso InputStream em um arquivo file e ele nos retorna o caminho completo de onde o arquivo foi criado. Nas **linhas 69 e 70** estamos transformando o File em um byte pois é esse tipo de arquivo que a api do d4sign espera.

Nas **linhas 72 a 76** estamos recuperando os valores do properties que vimos acima.

Na **linha 78** chamamos o método **getUuidCofre**, veremos ele detalhado mais abaixo, ele irá nos retornar uma chave única para esse cofre, isso serve para que possa ser dinâmico por exemplo a escolha do cofre a ser feito o upload do arquivo, podendo estar pegando esse nome de um campo do processo ao invés de pegar do properties.

Caso não tenha nenhum erro vindo de retorno de chave do cofre, na **linha 81** chamaremos o método **enviarDocumento**, esse método fará o “upload” do arquivo dentro do d4sign nos retornando uma chave única para esse documento.

Caso não tenha nenhum erro vindo de retorno da chave do documento, na **linha 84** deletaremos o arquivo temporário criado para não ficar sujeira no servidor do cliente.

Nas **linhas 85 e 86** estou retornando todos os valores da grid Assinantes, grid criada no modelo de exemplo para que possa ser enviado um documento para mais de uma pessoa assinar na plataforma. Na **linha 86** chamamos o método **cadastraAssinantes**, esse método é responsável por adicionar os assinantes ao documento que criamos no cofre.

Caso tenha retorno ok desse cadastro, iremos fazer na **linha 89** a chamada do método **enviaEmailAssinaturas**, esse método é responsável por fazer o disparo das assinaturas para os assinantes cadastrados no documento.

Todos os métodos mencionados acima que vamos ver detalhado na sequência utilizam chamadas das apis do D4sign por isso temos um seguindo do outro para termos o passo a passo que é necessário dentro do D4sign.

Caso o método retorne ok iremos chamar na **linha 92** o método **atualizaRelacionamento**, esse método fará um insert em uma tabela auxiliar para gravar o relacionamento entre esse código de processo e o uuid gerado do documento, pois é esse uuid que usaremos no robô para identificar se o documento já foi assinado.

```
InserArquivoD4Sign.java
Projeto Base > src > com.lecom.workflow.d4sign > InserArquivoD4Sign > atualizaRelacionamento(IntegracaoVO, String, String) : String
1140 private String atualizaRelacionamento(IntegracaoVO integracaoVO, String retorno, String uuidDocumento) {
115     integracaoVO.setConexao("aux_act");
116     try(Connection conBpm = integracaoVO.getConexao()){
117         StringBuilder sql = new StringBuilder();
118         sql.append(" insert into lecom_d4sign_controle(UUID_DOCUMENTO,COD_PROCESSO) ");
119         sql.append(" values (?,?) ");
120
121         Logger.debug(sql.toString());
122         try(PreparedStatement pst = conBpm.prepareStatement(sql.toString())){
123             pst.setString(1,uuidDocumento);
124             pst.setInt(2,Integer.parseInt(integracaoVO.getCodProcesso()));
125             pst.executeUpdate();
126
127             Logger.debug(retorno);
128         }
129
130         retorno = "0|Documento inserido para assinatura no D4sign, os devidos assinantes receberam notificações para assinar";
131     }catch(Exception e){
132         e.printStackTrace();
133         Logger.error("Erro ao conectar à base",e);
134         retorno = "99|Erro ao inserir UUID Documento/Processo.";
135     }
136
137     return retorno;
138 }
139
140 private String nomeArquivoAnexo(String anexo) throws Exception {
141     String nomeArquivo = "";
142     if (anexo != null && !"".equals(anexo)) {
143         // Realiza o split do valor do campo template, pois o primeiro valor é o nome do arquivo físico e o segundo é o uniqueid do documento, os dois separados por "."
144         String[] valores = anexo.split("[:]");
145         nomeArquivo = valores[0];
146     }
147     return nomeArquivo;
148 }
```

```
InserArquivoD4Sign.java
Projeto Base | src | com.lecom.workflow.d4sign | InserArquivoD4Sign | getUUidCofre(String, String, String, String) : String
149 private InputStream getWFFilePath(String anexo) throws Exception {
150     InputStream is = null;
151     if (anexo != null && !"".equals(anexo)) {
152         // Realiza o split do valor do campo template, pois o primeiro valor é o nome do arquivo físico e o segundo é o uniqueID do documento, os dois separados por ":"
153         String[] valores = anexo.split(":");
154         String identificacaoDocumento = valores[1];
155         // InputStream do arquivo retornado pelo método da API do ECM
156         is = documento().lerArquivo(identificacaoDocumento);
157     }
158     return is;
159 }
160
---
```

Acima temos os métodos **atualizaRelacionamento**, **nomeArquivoAnexo** e **getWFFilePath**.

O método **atualizaRelacionamento**, começamos ele informando que vamos utilizar uma conexão diferente da do Lecom BPM, vamos utilizar nossa base aux pois é nela que criamos as tabelas auxiliares necessárias em nossos clientes. Assim realizaremos o insert na tabela gravando o código do processo e o uuid do documento gerado pelo d4sign.

Os métodos **nomeArquivoAnexo** e **getWFFilePath** está sendo feito a separação do valor do campo template com o nome do arquivo : unique id do ecm, no primeiro método é retornado o nome do arquivo anexado e no segundo utiliza-se da api do ecm para retornar um inputstream a partir do unique id do ecm.

```
InserArquivoD4Sign.java
Projeto Base | src | com.lecom.workflow.d4sign | InserArquivoD4Sign | getUUidCofre(String, String, String, String) : String
161
162 @SuppressWarnings("resource")
163 public String getUUidCofre(String urlApi,String tokenApi,String cryptKey,String nomeCofre) {
164     Logger.info(" procura getUUidCofre nomeCofre = "+nomeCofre);
165     String uuidCofre = "";
166     try {
167         HttpClient client = HttpClientBuilder.create().build();
168         HttpGet get = new HttpGet(urlApi+"safes");
169         get.addHeader("tokenAPI",tokenApi);
170         get.addHeader("cryptKey",cryptKey);
171
172         HttpResponse response = client.execute(get);
173         Gson gson1 = new Gson();
174         if(response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK){
175             BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
176
177             StringBuilder result = new StringBuilder();
178             String line = "";
179             while ((line = rd.readLine()) != null) {
180                 result.append(line);
181             }
182
183             String resultado= "{\n\"dados\": "+result.toString()+"}";
184             org.json.JSONObject json = new org.json.JSONObject(resultado);
185
186             org.json.JSONArray jsonArray = (JSONArray) json.get("dados");
187
188             Type listType = new TypeToken<List<DadosCofre>>().getType();
189             String json_alterado = jsonArray.toString().replace("name-safe", "name_safe");
190             List<DadosCofre> dadosCofres = gson1.fromJson(json_alterado, listType);
191
192             for (DadosCofre map : dadosCofres) {
193                 if(map.getName().equals(nomeCofre)) {
194                     uuidCofre=map.getUUid();
195                     Logger.debug("uuidCofre = "+uuidCofre);
196                     break;
197                 }
198             }
199         }else {
200             BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
201             StringBuilder result = new StringBuilder();
202             String line = "";
203             while ((line = rd.readLine()) != null) {

```

```
InserArquivoD4Sign.java
Projeto Base ▸ src ▸ com.lecom.workflow.d4sign ▸ InserArquivoD4Sign ▸ getUUidCofre(String, String, String, String) : String ▸
202     String line = "";
203     while ((line = rd.readLine()) != null) {
204         result.append(line);
205     }
206
207     logger.debug("result = "+result);
208
209     uuidCofre = "Erro";
210 }
211
212 } catch (ClientProtocolException e) {
213     e.printStackTrace();
214     logger.error("Erro ao conectar",e);
215     uuidCofre = "Erro";
216 } catch (IOException e) {
217     e.printStackTrace();
218     logger.error("Erro ao conectar",e);
219     uuidCofre = "Erro";
220 } catch (Exception e) {
221     e.printStackTrace();
222     logger.error("Erro geral",e);
223     uuidCofre = "Erro";
224 }
225 return uuidCofre;
226 }
```

Acima temos o método **getUUidCofre** para ele passamos os parâmetros que retornamos do properties criado.

Nas **linhas 167 a 171** estamos realizando a chamada da api utilizando a biblioteca do httpclient, nesse caso a chamada para retornar o uuid do cofre é um get então utilizaremos a classe HTTPGet, ao instanciar ela passamos a url que queremos chamar nesse caso adicionamos a url base da api o valor **saftes** e adicionamos como header o tokenApi, cryptKey, esse método **saftes** nos retornará todos os cofres cadastrados.

Por isso quando o retorno da chamada http é ok, faremos a conversão do resultado da chamada para uma List<DadosCofre>, isso tudo é feito nas **linhas 174 a 190**, utilizando a api Gson, para transformar o json de retorno na lista do objeto que temos, essa classe DadosCofre contém as mesmas chaves que esse método retorna por isso essa conversão funciona da maneira esperada. Para assim facilitar a leitura de todo esse retorno e verificar se o nome do cofre que estamos procurando foi retornado e pegar o uuid dele (**linhas 192 a 198**).

Caso não tenha sucesso a chamada http ou aconteça alguma exceptions em algum momento, isso está sendo tratado nas **linhas 199 a 225** colocando o devido retorno para poder travar a execução da integração.

```
InserArquivoD4Sign.java
Projeto Base ▸ src ▸ com.lecom.workflow.d4sign ▸ InserArquivoD4Sign ▸ getUUidCofre(String, String, String, String) : String ▸
227
228 private String enviarDocumento(String caminhoAnexoTemporario, String fileInBytes, String uuidCofre, String nomeArquivo, String tokenApi, String cryptKey, String urlApi) {
229     logger.info("procura enviarDocumento caminho = "+caminhoAnexoTemporario);
230     String uuidDocumento = "";
231     try {
232         HttpClient client = HttpClientBuilder.create().build();
233         String url = urlApi + "documents/" + uuidCofre + "/uploadbinary?tokenAPI=" + tokenApi + "&cryptKey=" + cryptKey;
234
235         Map<String, String> header = new HashMap<String, String>();
236         header.put("language", "pt_BR");
237         header.put("Content-Type", "application/json");
238
239         HttpPost post = new HttpPost(url);
240         for (Entry<String, String> map : header.entrySet()) {
241             post.addHeader(map.getKey(), map.getValue());
242         }
243
244         logger.info("fileInBytes = "+fileInBytes);
245
246         Path path = new File(caminhoAnexoTemporario).toPath();
247         String mimeType = Files.probeContentType(path);
248         logger.info("mimeType = "+mimeType);
249
250         JSONObject jsonRequest = new JSONObject();
251         jsonRequest.put("base64_binary_file", fileInBytes);
252         jsonRequest.put("mime_type", mimeType);
253         jsonRequest.put("name", new File(caminhoAnexoTemporario).getName());
254
255         StringEntity entity = new StringEntity(jsonRequest.toString());
256         post.setEntity(entity);
257
258         HttpResponse response = client.execute(post);
259
260         if (response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK) {
261             BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent(), "UTF-8"));
262
263             StringBuilder result = new StringBuilder();
264             String line = "";
265             while ((line = rd.readLine()) != null) {
266                 result.append(line);
267             }
268         }
269     }
270 }
```



```

262
263     StringBuilder result = new StringBuilder();
264     String line = "";
265     while ((line = rd.readLine()) != null) {
266         result.append(line);
267     }
268
269     Logger.info("Result: " + result.toString());
270     org.json.JSONObject json = new org.json.JSONObject(result.toString());
271
272     if (json.get("message").equals("success")) {
273         uuidDocumento = json.getString("uuid");
274     }
275     else {
276         BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
277         StringBuilder result = new StringBuilder();
278         String line = "";
279         while ((line = rd.readLine()) != null) {
280             result.append(line);
281         }
282
283         Logger.debug("result = "+result);
284
285         uuidDocumento = "Erro";
286     }
287
288     } catch (ClientProtocolException e) {
289         e.printStackTrace();
290         Logger.error("Erro ao conectar",e);
291         uuidDocumento = "Erro";
292     } catch (IOException e) {
293         e.printStackTrace();
294         Logger.error("Erro ao conectar",e);
295         uuidDocumento = "Erro";
296     } catch (Exception e) {
297         e.printStackTrace();
298         Logger.error("Erro geral",e);
299         uuidDocumento = "Erro";
300     }
301
302     return uuidDocumento;
303 }

```

Acima temos o método **enviarDocumento** para ele iremos passar o caminho do arquivo temporário criado, o arquivo em bytes, o uuid do cofre, o nome do arquivo anexado e as informações do properties.

Na **linha 233** montamos a url que devemos chamar a partir sempre da url base da api do d4sign, passando na url diretamente o tokenapi e o cryptkey.

Nas **linhas 235 a 242** montamos os headers necessários e adicionamos ao HTTPPost.

Nas **linhas 246 a 248** estaremos realizando uma conversão para encontrar qual é o mime type do arquivo que está sendo enviado, isso da uma abertura para que qualquer tipo de extensão consiga ser aberta e assinada no d4sign, vimos a necessidade de 3 tipo no momento imagem, pdf e docx.

Após essa conversão montamos o json que será enviado no momento da chamada da api, passando o arquivo em bytes, o mimetype e o nome do arquivo. Com esse json criado transformamos ele em uma String Entity e adicionamos no post (**linhas 250 a 256**).

Ao validarmos se a chamada retornou ok, nas **linhas 260 a 274**, faremos a tratativa do retorno pegando apenas o uuid retornado. Se o retorno não foi ok da chamada, trataremos e retornaremos ao método principal com mensagem de erro, essas tratativas estão sendo feitas nas **linhas 275 a 302**.

```
InserirArquivoD4Sign.java
Projeto Base > src > com.lecom.workflow.d4sign > InserirArquivoD4Sign > getUUidCofre(String, String, String, String): String
305
306 public String cadastraAssinantes(String urlApi, String tokenApi, String criptKey, String uuidDocumento, List<Map<String, Object>> assinantes) {
307     Logger.info("procura cadastraAssinantes");
308     String retorno = "";
309
310     //montando json para envio das assinaturas com um array de possíveis assinantes
311     JSONObject jsonObj = new JSONObject();
312     JSONArray signers = new JSONArray();
313     int cont = 1;
314     for (Map<String, Object> map : assinantes) {
315         Logger.debug(map.get("EMAIL_ASSINANTE"));
316         JSONObject json = new JSONObject();
317         json.put("email", map.get("EMAIL_ASSINANTE"));
318         json.put("act", "1");
319         json.put("foreign", "0");
320         json.put("after_position", cont);
321         json.put("certificadoicpb", "0");
322         signers.put(json);
323         cont++;
324     }
325     jsonObj.put("signers", signers);
326
327     try {
328         HttpClient client = HttpClientBuilder.create().build();
329         String url = urlApi + "documents/" + uuidDocumento + "/createlist?tokenAPI=" + tokenApi + "&criptKey=" + criptKey;
330         Map<String, String> header = new HashMap<String, String>();
331         header.put("language", "pt_BR");
332         header.put("Content-Type", "application/json");
333         HttpPost post = new HttpPost(url);
334         for (Entry<String, String> map : header.entrySet()) {
335             post.addHeader(map.getKey(), map.getValue());
336         }
337         StringEntity entity = new StringEntity(jsonObj.toString());
338         post.setEntity(entity);
339         HttpResponse response = client.execute(post);
340         List<String> listRetorno = new ArrayList<String>();
341     }
342 }
```

```
InserirArquivoD4Sign.java
Projeto Base > src > com.lecom.workflow.d4sign > InserirArquivoD4Sign > getUUidCofre(String, String, String, String): String
347
348 if (response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK) {
349     BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent(), "UTF-8"));
350     StringBuilder result = new StringBuilder();
351     String line = "";
352     while ((line = rd.readLine()) != null) {
353         result.append(line);
354     }
355
356     Logger.info("Result: " + result.toString());
357     org.json.JSONObject json = new org.json.JSONObject(result.toString());
358
359     JSONArray ret = json.getJSONArray("message");
360     JSONObject explrObject;
361     listRetorno.add("OK");
362     for (int i = 0; i < ret.length(); i++) {
363         explrObject = ret.getJSONObject(i);
364         Logger.info("explrObject: " + explrObject);
365         Logger.info("status: " + explrObject.get("status"));
366         if (!explrObject.get("status").equals("created")) {
367             listRetorno.add("ERRO");
368             break;
369         }
370     }
371 } else {
372     BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
373     StringBuilder result = new StringBuilder();
374     String line = "";
375     while ((line = rd.readLine()) != null) {
376         result.append(line);
377     }
378     Logger.debug("result = " + result);
379     listRetorno.add("ERRO");
380 }
381
382 if (listRetorno.contains("ERRO")) {
383     retorno = "ERRO";
384 } else {
385     retorno = "OK";
386 }
387
388 }
```

```
InserirArquivoD4Sign.java
Projeto Base > src > com.lecom.workflow.d4sign > InserirArquivoD4Sign > getUUidCofre(String, String, String, String): String
389
390 } catch (ClientProtocolException e) {
391     e.printStackTrace();
392     Logger.error("Erro ao conectar", e);
393     retorno = "Erro";
394 } catch (IOException e) {
395     e.printStackTrace();
396     Logger.error("Erro ao conectar", e);
397     retorno = "Erro";
398 } catch (Exception e) {
399     e.printStackTrace();
400     Logger.error("Erro geral", e);
401     retorno = "Erro";
402 }
403
404 return retorno;
405 }
```

Acima temos o método **cadastraAssinantes** para ele passamos as informações do properties criado, o uuid documento gerado, e a lista de assinantes informado na grid.

Nas **linhas 311 a 327** está sendo criado um jsonarray com todos os e-mails dos assinantes inseridos na grid e com informações necessárias pela api, como pode ser visto nas **linhas 319**

a **322** e para cada linha coloca um contador começando no 1 e indo até a quantidade de itens inseridos na grid.

Na **linha 331** montamos a url da chamada da api tendo como base a url do properties, para ela passamos o uuid do documento, o tokenapi e o cryptkey.

Montamos um map dos headers necessários para essa chamada e adicionamos ao `HttpPost`.

Como precisamos enviar como parâmetros o json que criamos no começo, convertamos esse jsonarray em uma `stringentity` para ser enviado, conforme setamos na **linha 343**.

Assim executamos a chamada e verificamos se deu ok na chamada, conforme teste feito na **linha 348**, se der certo iremos transformar em json o retorno e explorar o resultado para avaliar se deu sucesso ou erro, caso tenha algum registro com retorno diferente de created é adicionado para lista de retorno a palavra erro, conforme **linhas 367 a 369**.

Nas **linhas 373 a 405** é feito algumas validações e retornos de erros específicos para retornar a chamada principal e travar a execução caso ocorra algum problema.

```
InserirArquivoD4Sign.java
Projeto Base > src > com.lecom.workflow.d4sign > InserirArquivoD4Sign > enviaEmailsAssinaturas(String, String, String, String): String
407
408 public String enviaEmailsAssinaturas(String urlApi, String uuidDocumento, String tokenApi, String cryptKey) {
409     Logger.info(" enviaEmailsAssinaturas");
410     String retorno = "";
411     try {
412         HttpClient client = HttpClientBuilder.create().build();
413         String url = urlApi + "documents/" + uuidDocumento + "/sendtosigner?tokenAPI=" + tokenApi + "&cryptKey=" + cryptKey;
414
415         Map<String, String> header = new HashMap<String, String>();
416         header.put("language", "pt_BR");
417         header.put("Content-Type", "application/json");
418
419         HttpPost post = new HttpPost(url);
420         for (Entry<String, String> map : header.entrySet()) {
421             post.addHeader(map.getKey(), map.getValue());
422         }
423
424         JSONObject jsonEnviar = new JSONObject();
425         jsonEnviar.put("message", "Segue documento para assinatura.");
426         jsonEnviar.put("workflow", "0");
427         jsonEnviar.put("skip_email", "0");
428
429         StringEntity entity = new StringEntity(jsonEnviar.toString());
430         post.setEntity(entity);
431
432         HttpResponse response = client.execute(post);
433         List<String> listRetorno = new ArrayList<String>();
434
435         if (response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK) {
436             BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent(), "UTF-8"));
437
438             StringBuilder result = new StringBuilder();
439             String line = "";
440             while ((line = rd.readLine()) != null) {
441                 result.append(line);
442             }
443
444             Logger.info("Result: " + result.toString());
445             org.json.JSONObject json = new org.json.JSONObject(result.toString());
446             if (json.get("message").toString().contains("successfully")) {
447                 listRetorno.add("OK");
448             } else {
449                 listRetorno.add("ERRO");
450             }
451         }
452     } catch (Exception e) {
453         listRetorno.add("ERRO");
454     }
455     return listRetorno.toString();
456 }
```

```

InserArquivoD4Sign.java
Projeto Base > src > com.lecom.workflow.d4sign > InserArquivoD4Sign > enviaEmailsAssinaturas(String, String, String, String) : String
446         if (json.get("message").toString().contains("successfully")) {
447             listRetorno.add("OK");
448         } else {
449             listRetorno.add("ERRO");
450         }
451     }
452 } else {
453     BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
454     StringBuilder result = new StringBuilder();
455     String line = "";
456     while ((line = rd.readLine()) != null) {
457         result.append(line);
458     }
459     Logger.debug("result = "+result);
460     listRetorno.add("ERRO");
461 }
462 }
463 if(listRetorno.contains("ERRO")){
464     retorno = "ERRO";
465 }else {
466     retorno = "OK";
467 }
468 }
469 }
470 } catch (ClientProtocolException e) {
471     e.printStackTrace();
472     Logger.error("Erro ao conectar",e);
473     retorno = "Erro";
474 } catch (IOException e) {
475     e.printStackTrace();
476     Logger.error("Erro ao conectar",e);
477     retorno = "Erro";
478 } catch (Exception e) {
479     e.printStackTrace();
480     Logger.error("Erro geral",e);
481     retorno = "Erro";
482 }
483 }
484 }
485 return retorno;
486 }
487 }

```

Acima temos o método **enviaEmailsAssinaturas**, para ele passamos as informações contidas no properties criado e o uuid do documento.

Na **linha 413** criamos a url para a chamada da api, passando o uuid do documento, tokenapi e cryptkey.

Nas **linhas 419 a 422** estamos criando um map dos headers que precisamos para essa chamada e adicionamos eles no HttpPost.

Nas **linhas 424 a 430** montamos um json com informações necessárias e passando uma mensagem de que os documentos foram enviados para a assinatura e enviamos como entity.

Quando o resultado der ok na chamada, nas **linhas 435 a 450** fazemos a tratativa do retorno transformando para json e verificando se na message retornada contém a palavra successfully, caso tenha retorna para o método principal que deu certo se não que tem erro.

Nas **linhas 453 a 485** fazemos a tratativa do erro, e retornamos para o método se executou corretamente a chamada.

c. Robô

A classe que iremos ver e entender o fonte agora é a RBValidaArquivosAssinadosD4Sign.

```
RBValidaArquivosAssinadosD4Sign.java  InserirArquivoD4Sign.java  dadosApiD4Sign.properties
Projeto Base  src  com.lecom.workflow.d4sign  RBValidaArquivosAssinadosD4Sign  validaArquivosAssinados(): void

59 @RobotModule("RBValidaArquivosAssinadosD4Sign")
60 @Version({1,1,0})
61 public class RBValidaArquivosAssinadosD4Sign {
62
63     private String configPath = Funcoes.getWRootDir() + "/upload/cadastros/config/";
64     private static final Logger Logger = Logger.getLogger(RBValidaArquivosAssinadosD4Sign.class);
65     private static final int BUFFER_SIZE = 4096;
66
67     @Execution
68     public void validaArquivosAssinados() {
69         Logger.debug("validaArquivosAssinados");
70         try {
71             Map<String, String> parametros = Funcoes.getParametrosIntegracao(configPath + "tarefa.automatica");
72
73             Map<String, String> dadosConf = Funcoes.getParametrosIntegracao(configPath + "dadosApiD4Sign");
74             String urlApi=dadosConf.get("url_api");
75             String tokenApi=dadosConf.get("token_api");
76             String cryptKey=dadosConf.get("crypt_key");
77
78             StringBuilder queryProcesso = new StringBuilder();
79             queryProcesso.append("SELECT p.cod_processo, p.cod_etapa_atual, p.cod_ciclo_atual, p.cod_versao, p.cod_form, p.ide_beta_teste ")
80             .append(" FROM processo p ")
81             .append(" INNER JOIN processo_etapa pe ")
82             .append(" ON pe.cod_processo = p.cod_processo ")
83             .append(" AND pe.cod_etapa = p.cod_etapa_atual ")
84             .append(" AND pe.cod_ciclo = p.cod_ciclo_atual ")
85             .append(" WHERE p.cod_form = 2 and p.cod_etapa_atual = 3 ")
86             .append(" AND p.ide_finalizado = 'A' ")
87             .append(" and pe.cod_usuario_etapa = ? ");
88
89             List<String> docAssinados = documentsByStatus(tokenApi, cryptKey,urlApi);
90             Logger.debug("docAssinados = "+ docAssinados.size());
91             if(!docAssinados.contains("ERRO")) {
92                 try(connection con = DBUtils.getConnection("workflow")) {
93                     try(PreparedStatement pst = con.prepareStatement(queryProcesso.toString())){
94                         pst.setString(1,parametros.get("codUsuarioAutomatico"));
95                         try(ResultSet rs = pst.executeQuery()){
96                             while(rs.next()) {
97                                 String codProcesso = rs.getString("cod_processo");
98                                 String codEtapa = rs.getString("cod_etapa_atual");
99                                 String codCiclo = rs.getString("cod_ciclo_atual");
100                                 String modoTeste = rs.getString("ide_beta_teste");
101
102                                 String modoTeste = rs.getString("ide_beta_teste");
103                                 Logger.debug("codProcesso = "+ codProcesso);
104                                 Logger.debug("codEtapa = "+ codEtapa);
105                                 Logger.debug("codCiclo = "+ codCiclo);
106                                 Logger.debug("modoTeste = "+ modoTeste);
107
108                                 String uuidDocumento = getUuidDocumentoByProcesso(codProcesso);
109                                 Logger.debug("uuidDocumento = "+ uuidDocumento);
110                                 if(!uuidDocumento.equals("")) {
111                                     for(String docId : docAssinados) {
112                                         Logger.debug(" Documento assinado encontrado = "+ docId);
113                                         if (docId.equals(uuidDocumento)) {
114                                             Logger.debug(" Documento do processo ja está assinado ");
115                                             String urlDownload = getUrlDocumentoDownload(urlApi, tokenApi, cryptKey, uuidDocumento, docAssinados);
116                                             Logger.debug("urlDownload = "+ urlDownload);
117                                             if(!urlDownload.equals("")){
118                                                 String valorArquivoGeradoECH = downloadFile(urlDownload, codProcesso);
119                                                 Logger.debug("valorArquivoGeradoECH = "+ valorArquivoGeradoECH);
120                                                 if(!valorArquivoGeradoECH.equals("ERRO")) {
121                                                     String aprovacao = processExecution(parametros, codProcesso, codEtapa, codCiclo,modoTeste, valorArquivoGeradoECH);
122                                                     Logger.info("RETORNO APROVACAO: " + aprovacao);
123                                                 }
124                                             }
125                                         }
126                                     }
127                                 }
128                             }
129                         }
130                     }
131                 } catch (ClientProtocolException e) {
132                     e.printStackTrace();
133                     Logger.error("Erro ao conectar",e);
134                 } catch (IOException e) {
135                     e.printStackTrace();
136                     Logger.error("Erro ao conectar",e);
137                 } catch (Exception e) {
138                     e.printStackTrace();
139                     Logger.error("Erro ao conectar",e);
140                 }
141             }
142         }
143     }
144 }
```

As imagens acima são do método principal do robô. Primeiramente iniciamos com as anotações RobotModule e Version e iniciamos as variáveis de log, do caminho dos arquivos de configuração e um tamanho do buffer que vamos precisar no momento do download do documento assinado.

Nas **linhas 71 a 76** estamos pegando todas as variáveis dos arquivos properties que precisamos, nesse caso vamos usar o tarefa.automatica, arquivo que tem as informações do ambiente e do usuário automático que utilizamos em nossos clientes.

O select, que está sendo montado nas **linhas 78 a 87**, irá trazer todos os processos desse formulário que estão parados na atividade em questão com o usuário automático.

Na **linha 89** estamos chamando o método **documentsByStatus**, esse método irá trazer todos os ids de documento já assinados no d4sign.

Nas **linhas 91 a 109** é feito a execução da query e a partir do processo retornado é chamado o método **getUUidDocumentoByProcesso**, para que ele traga qual é o id do documento desse processo para validar se ele já foi assinado no D4sign e seguir com o processo.

Caso tenha retornado o id do documento, fazemos um for dentro de todos os documentos retornados da api que foram assinados e verificamos se é igual ao o id atrelado a este processo, esses procedimentos estão nas **linhas 109 a 112**.

Na **linha 114** chamamos o método **getUrlDocumentoDownload** que irá chamar uma api do d4sign para retornar a url disponível para download do documento gerado com as devidas assinaturas. Com essa url retornada passamos ela para o método **downloadFile**, que realiza o download do documento e insere ele no ecm e retorna valor para que possamos preencher o campo do nosso processo e seguir.

No caso na **linha 120** é chamado o método **processExecution** que irá fazer a aprovação do processo automaticamente e devolverá ao usuário iniciador com o documento assinado.



```

144 public String getUUidDocumentoByProcesso(String codProcesso) throws SQLException {
145     String retornoUuid = "";
146     try(Connection conAux = DBUtils.getConnection("aux_act")){
147         String sql = "select * from lecom_d4sign_controle where COD_PROCESSO = ? ";
148         try(PreparedStatement pst = conAux.prepareStatement(sql)){
149             pst.setString(1, codProcesso);
150             try(ResultSet rs = pst.executeQuery()){
151                 if(rs.next()) {
152                     retornoUuid = rs.getString("UUID_DOCUMENTO");
153                 }
154             }
155         }
156     }
157     return retornoUuid;
158 }

```

Na imagem acima temos o método **getUUidDocumentoByProcesso**, que está realizando um select na base auxiliar para retornar qual é o id do documento a partir do código do processo.



```

160 @SuppressWarnings("resource")
161 public List<String> documentsByStatus(String tokenApi, String cryptKey, String urlApi) throws ClientProtocolException, IOException {
162     Logger.info(" -- documentsByStatus -- ");
163     List<String> ret = new ArrayList<String>();
164
165     String URL = urlApi+"documents/4/status?tokenAPI=" + tokenApi+"&cryptKey=" + cryptKey;
166
167     HttpClient client = HttpClientBuilder.create().build();
168     HttpGet get = new HttpGet(URL);
169     get.addHeader("tokenAPI", tokenApi);
170     get.addHeader("cryptKey", cryptKey);
171
172     HttpResponse response = client.execute(get);
173     if(response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK){
174         BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
175
176         StringBuilder result = new StringBuilder();
177         String line = "";
178         while ((line = rd.readLine()) != null) {
179             result.append(line);
180         }
181
182         Logger.info(" result = "+result);
183
184         JSONArray arr = new JSONArray(result.toString());
185         JSONObject docInfo = arr.getJSONObject(0);
186         int docQt = docInfo.getInt("total_documents");
187
188         if (docQt > 0) {
189             for (int i = 1; i < arr.length(); ++i) {
190                 JSONObject jo = arr.getJSONObject(i);
191                 ret.add(jo.getString("uuidDoc"));
192             }
193         } else {
194             ret.add("Sem documentos assinados");
195         }
196     }
197 }
198
199 else{
200     BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
201     StringBuilder result = new StringBuilder();
202     String line = "";

```

```

RBValidaArquivosAssinadosD4Sign.java  InserirArquivoD4Sign.java
Projeto Base  src  com.lecom.workflow.d4sign  RBValidaArquivosAssinadosD4Sign  validaArquivosAssinados(): void
197
198     }else{
199         BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
200         StringBuilder result = new StringBuilder();
201         String line = "";
202         while ((line = rd.readLine()) != null) {
203             result.append(line);
204         }
205
206         Logger.debug("result = "+result);
207
208         ret.add("ERRO");
209     }
210
211     return ret;
212 }

```

Nas imagens acima temos o método **documentsByStatus**, para ele passamos as informações do properties criado.

Nas **linhas 165 a 172** é feito a criação da url da chamada da api, tendo a url base da api como início nessa url o que chama atenção é um numero 4 passado, que significa que quero que retorne todos os documentos com status de assinado para esse tokenapi e cryptkey.

Nas **linhas 173 a 197** validamos se a chamada da url deu ok, e diante do retorno é transformado em jsonarray, e vamos fazer um for diante da quantidade de documentos retornados e adicionamos a uma lista de String para que seja validado no método principal se o id do documento existente para o processo em questão está dentro dessa lista.

```

RBValidaArquivosAssinadosD4Sign.java  InserirArquivoD4Sign.java
Projeto Base  src  com.lecom.workflow.d4sign  RBValidaArquivosAssinadosD4Sign  getUrlDocumentoDownload(String urlApi, String tokenApi, String cryptKey, String idDocumento) throws ClientProtocolException, IOException: String
214 public String getUrlDocumentoDownload(String urlApi, String tokenApi, String cryptKey, String idDocumento) throws ClientProtocolException, IOException {
215     Logger.info("getUrlDocumentoDownload");
216     String retorno = "";
217
218     String url = urlApi+"documents/"+idDocumento+"/download?tokenAPI="+tokenApi+"&cryptKey="+ cryptKey;
219
220
221     HttpClient client = HttpClientBuilder.create().build();
222     HttpPost post = new HttpPost(url);
223
224     List<NameValuePair> params = new ArrayList<NameValuePair>();
225     params.add(new BasicNameValuePair("type", "pdf"));
226     post.setEntity(new UrlEncodedFormEntity(params, "UTF-8"));
227
228     HttpResponse response = client.execute(post);
229     if(response.getStatusLine().getStatusCode() == HttpURLConnection.HTTP_OK){
230         BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
231
232         StringBuilder result = new StringBuilder();
233         String line = "";
234         while ((line = rd.readLine()) != null) {
235             result.append(line);
236         }
237
238         Logger.info(" result = "+result);
239
240         org.json.JSONObject json = new org.json.JSONObject(result.toString());
241         String urlDownload = json.getString("url");
242         retorno = urlDownload;
243     }else {
244         BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
245         StringBuilder result = new StringBuilder();
246         String line = "";
247         while ((line = rd.readLine()) != null) {
248             result.append(line);
249         }
250         Logger.debug("result = "+result);
251         retorno = "Erro";
252     }
253
254     return retorno;
255 }

```

Na imagem acima temos o método **getUrlDocumentoDownload**, onde nela recebemos o id do documento e as informações que estão no properties referentes ao d4sign.

Na **linha 218** é montado a url da chamada da api, passando os parâmetros que o método recebe.

Nas **linhas 225 a 227** montamos os parâmetros que precisamos enviar na chamada, por isso passamos o type igual a pdf, pois assim a url retornada será a do arquivo pdf contendo as assinaturas, isso se da ao fato de qualquer tipo de arquivo de entrada ter como saída um pdf assinado.

Nas **linhas 230 a 242** está sendo tratado o retorno de sucesso da chamada da api, transformando em um json e retorna a url em questão.

```

RBValidaArquivosAssinadosD4Sign.java  InserirArquivoD4Sign.java
Projeto Base  src  com.lecom.workflow.d4sign  RBValidaArquivosAssinadosD4Sign  downloadFile(String, String): String
257 public String downloadFile(String fileURL, String codProcesso) throws Exception {
258     String retorno = "";
259     try {
260         URL url = new URL(fileURL);
261         HttpURLConnection httpConn = (HttpURLConnection) url.openConnection();
262         int responseCode = httpConn.getResponseCode();
263
264         if (responseCode == HttpURLConnection.HTTP_OK) {
265             String filename = "";
266             String disposition = httpConn.getHeaderField("Content-Disposition");
267
268             if (disposition != null) {
269                 int index = disposition.indexOf("filename=");
270                 if (index > 0) {
271                     filename = disposition.substring(index + 10, disposition.length() - 1);
272                 }
273             } else {
274                 filename = fileURL.substring(fileURL.lastIndexOf("/") + 1, fileURL.length());
275             }
276
277             File fileDirTemp = new File(configPath + "../tempFiles/");
278             //especifica se tem anexos para incluir no processo
279             if (!fileDirTemp.exists()) {
280                 fileDirTemp.mkdirs();
281             }
282
283             InputStream inputStream = httpConn.getInputStream();
284             File file = new File(configPath + "../tempFiles/" + filename);
285             FileOutputStream outputStream = new FileOutputStream(file);
286
287             int bytesRead = -1;
288             byte[] buffer = new byte[BUFFER_SIZE];
289             while ((bytesRead = inputStream.read(buffer)) != -1) {
290                 outputStream.write(buffer, 0, bytesRead);
291             }
292
293             Map<String,String> metadados = new HashMap<String, String>();
294             metadados.put("PROCESSO", codProcesso);
295
296             br.com.docsys.ecm.client.dto.document.Document documento = documento().criarDocumentoComIdentificador(file, "DOCS_ASSINADOS_D4SIGN").criarMetadados(metadados).salvar();
297
298             br.com.docsys.ecm.client.dto.document.Document documento = documento().criarDocumentoComIdentificador(file, "DOCS_ASSINADOS_D4SIGN").criarMetadados(metadados).salvar();
299             DocFile arquivoDoc = documento.getCurrentFile();
300             String nomeArquivo = arquivoDoc.getFilename();
301             String nomeCriptografado = arquivoDoc.getFileUniqueId().getValue();
302
303             logger.debug("ARQUIVO GERADO : " + nomeArquivo);
304             logger.debug("NOME CRIPTOGRAFADO : " + nomeCriptografado);
305
306             retorno = nomeArquivo + ":" + nomeCriptografado;
307
308             outputStream.close();
309             inputStream.close();
310
311             httpConn.disconnect();
312         } catch (ConteudoDocumentoException e) {
313             logger.error("ConteudoDocumentoException: ", e);
314             e.printStackTrace();
315             retorno = "ERRO";
316         } catch (DocumentoException e) {
317             logger.error("DocumentoException: ", e);
318             e.printStackTrace();
319             retorno = "ERRO";
320         } catch (ProfileException e) {
321             logger.error("ProfileException: ", e);
322             e.printStackTrace();
323             retorno = "ERRO";
324         } catch (CriarDocumentoException e) {
325             logger.error("CriarDocumentoException: ", e);
326             e.printStackTrace();
327             retorno = "ERRO";
328         } catch (ArquivoInvalidoException e) {
329             logger.error("ArquivoInvalidoException: ", e);
330             e.printStackTrace();
331             retorno = "ERRO";
332         } catch (ArquivoNaoEncontradoException e) {
333             logger.error("ArquivoNaoEncontradoException: ", e);
334             e.printStackTrace();
335             retorno = "ERRO";
336         } catch (TemplateNaoEncontradoException e) {
337             logger.error("TemplateNaoEncontradoException: ", e);
338             e.printStackTrace();
339             retorno = "ERRO";
340         } catch (TemplateException e) {
341             logger.error("TemplateException: ", e);
342             e.printStackTrace();
343             retorno = "ERRO";
344         }
345     }
346 }

```

Nas imagens acima temos o método **downloadFile**, recebendo como parâmetro a url do documento retornado no método anterior e o código do processo.

Nas **linhas 260 a 262** estamos utilizando nesse caso o **HttpURLConnection**, pois para o que precisamos retornar a partir da url a tratativa fica melhor utilizando esses métodos, então inicializamos a classe URL passando a url do documento.

Nas **linhas 265 a 291**, estamos pegando o retorno dessa chamada e conseguindo pegar o nome do arquivo, conforme retorno na **linha 271 ou 274**. A partir do inputStream que conseguirmos retornar também dessa chamada, conforme **linha 283**, vamos transformar esse inputStream em um file temporário para utilizarmos na api de criação do documento no ecm.

Nas **linhas 293 e 294** estamos criando um Map para que possamos preencher os campos metadados do nosso template e assim passamos o identificador do campo que criamos PROCESSO, e o valor que ele vai ter, passando o código do processo.

Nas linhas 296 a 304 estamos utilizando a api do ecm para criar o documento nele, fazendo assim a chamada do método `criarDocumentoComIdentificador`, passando o identificador do template que estamos utilizando, e chamamos o método `criarMetadados` passando o map dos campos e na sequência o método `salvar`.

O retorno do método `salvar` já conseguimos ter os valores do nome do arquivo e nome criptografado que são as informações que precisamos para preencher o campo do nosso modelo e seguir com o processo.

```

RBValidaArquivosAssinadosD4Sign.java  InserirArquivoD4Sign.java
Projeto Base  src  com.lecom.workflow.d4sign  RBValidaArquivosAssinadosD4Sign  downloadFile(String, String): String
347      e.printStackTrace();
348      retorno = "ERRO";
349  }
350  return retorno;
351  }
352
353  private String processExecution(Map<String, String> parametros, String codProcesso, String codEtapa, String codCiclo, String modoTeste, String valorCampoAssinatura) {
354      String ret = "";
355
356      String sso = parametros.get("enderecoSso");
357      String bpm = parametros.get("enderecoBpm");
358      String test = modoTeste.equals("s") ? "true" : "false";
359      String user = parametros.get("codusuarioAutomatico");
360
361      String login = parametros.get("loginUsuarioAutomatico");
362      String pw = parametros.get("senhaUsuarioAutomatico");
363
364      String token = applicationLogin(sso, login, pw);
365
366      Map<String, String> processedData = new HashMap<String, String>();
367
368      try {
369
370          DadosProcesso dadosProcesso = new DadosProcesso("P");
371          processedData.put("ANEXO_ASSINADO", valorCampoAssinatura);
372          dadosProcesso.geraPadroes(processedData);
373
374          DadosProcessoAbertura dadosProcessoAbertura = new DadosProcessoAbertura();
375          dadosProcessoAbertura.setProcessInstanceId(codProcesso);
376          dadosProcessoAbertura.setCurrentActivityInstanceId(codEtapa);
377          dadosProcessoAbertura.setCurrentCycle(codCiclo);
378          dadosProcessoAbertura.setModoTeste(test);
379
380          AprovaProcesso aprovaProcesso = new AprovaProcesso(bpm, token, dadosProcessoAbertura, dadosProcesso, test, user);
381          ret = aprovaProcesso.aprovaProcesso();
382
383      } catch (AprovaProcessoException e) {
384          Logger.error("AprovaProcessoException: ", e);
385          e.printStackTrace();
386      }
387      return ret;
388  }
389

```

Na imagem acima temos o método **processExecution**, recebendo como parâmetros as informações retornadas do properties do tarefa.automatica, o código do processo, o código da etapa, o código do ciclo, se o processo está em modo teste e o valor retornado no método visto anteriormente.

Nesse método é utilizado classes que estão sendo explicadas no documento RotasBPM para utilizar por fora a aprovação de processos, para isso precisamos fazer o login primeiro e depois a aprovação, e para ela passamos o valor do campo que queremos preencher no exemplo o campo ANEXO_ASSINADO.

```

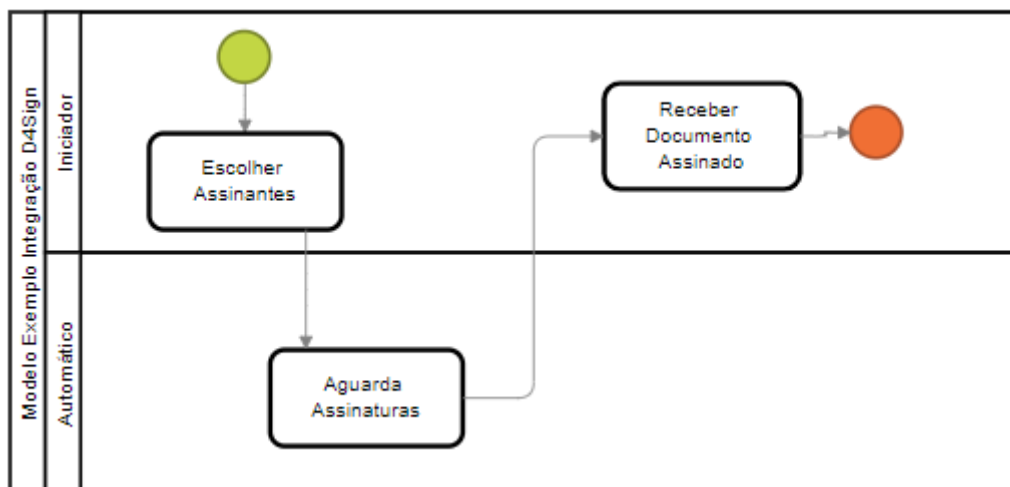
389
390  private String applicationLogin(String url, String login, String pw) {
391
392      String ret = "";
393
394      try {
395          DadosLogin loginUtil = new DadosLogin(login, pw, true);
396          LoginAutenticacao loginAutenticacao = new LoginAutenticacao(url, loginUtil);
397          ret = loginAutenticacao.getToken();
398      } catch (LoginAuthenticationException e) {
399          Logger.error("LoginAuthenticationException - Method applicationLogin: ", e);
400          e.printStackTrace();
401      }
402      return ret;
403  }
404

```

Na imagem acima temos o método **applicationLogin**, recebendo como parâmetro a url, o login e a senha do usuário automático, utilizado para que possamos realizar o login e a aprovação do processo no método anterior.

5. Processo

O processo de exemplo foi criado em um ambiente de treinamento nosso, abaixo o diagrama do processo e os campos criados.



Documento					3 ativos / 0 inativos	
1	Template	ANEXO	Documento a ser assinado:			
2	Linha de texto	UUID_CONTRATO	Uuid			
3	Template	ANEXO_ASSINADO	Documento assinado:			
+ Novo campo						
Assinantes					2 ativos / 0 inativos	
1	Assinante	Linha de texto	NOME_ASSINANTE	Nome Assinante:		
2	Assinante	Linha de texto	EMAIL_ASSINANTE	Email Assinante:		
+ Novo campo						

Foi criado dois template para separar o arquivo importado pelo usuário para ser assinado e o arquivo assinado em questão retornado do D4Sign e esse último possui um campo de metadado criado para que preenchamos via robô.

Identificador	Nome	Descrição
DOCS_ASSINADOS_D4SIGN	Template Documentos Assinados pelo D4Sign	Template destinado a guardar os documentos assinados vindo do d4sign
TEMPLATE_DOCS_D4SIGN	Template Documentos D4Sign	Template destinado a guardar os arquivos inputados a serem assinados no d4sign

Editar Template - Template Documentos Assinados pelo D4Sign

Informações Gerais

Nome
Template Documentos Assinados pelo D4Sign

Ativar PDF Assinado
☐ Sim
☒ Não

Identificador
DOCS_ASSINADOS_D4SIGN

Descrição
Template destinado a guardar os documentos assinados vindo do d4sign

Modelo padrão
Selecione...

Marca d'água

Categoria

Idioma
Português

Tipos de Arquivos Permitidos

Tamanho Máximo do Arquivo (KB)
10000

Campos

Posição	Nome	Identificador	Tipo	Info. adicio...	Tama...	Obrig...	Pe...	Res...	Vali...
1	Número instancia	PROCESSO	Texto		50	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

+

Salvar

O processo consiste na chamada da integração após a primeira atividade e a segunda atividade ficará por conta do robô para executar após o documento ser assinado no D4Sign.

Tipo da mensagem

Padrão

Aprovação

Responsável pela etapa

Vai para

Aguarda Assinaturas

Label do botão

Aprovar

Integração

InserirArquivoD4Sign

6. Resultado

Vamos ver agora alguns prints da execução do processo e assinatura do documento.

Na primeira atividade preenchi um pdf de exemplo e um usuário como assinante:

Documento

Documento a ser assinado:

555.pdf

Documento assinado:

Assinantes

Nome Assinante:

Email Assinante:

ADICIONAR DADOS NA TABELA
+

Nome Assinante:	Email Assinante:	Ações
Nome Usuário	emailusuario	<div> <div></div> <div></div> </div>

20
1 - 1 de 1

Ao Aprovar a integração rodou corretamente retornando para o usuário:

Aguarda Assinaturas
Ciclo: 01

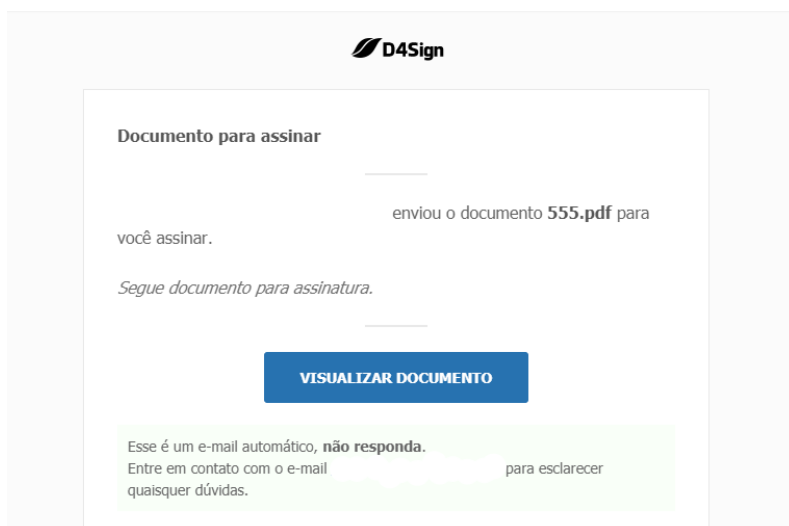
Início da Atividade
24/06/2020

Próximos responsáveis: Atividade automática

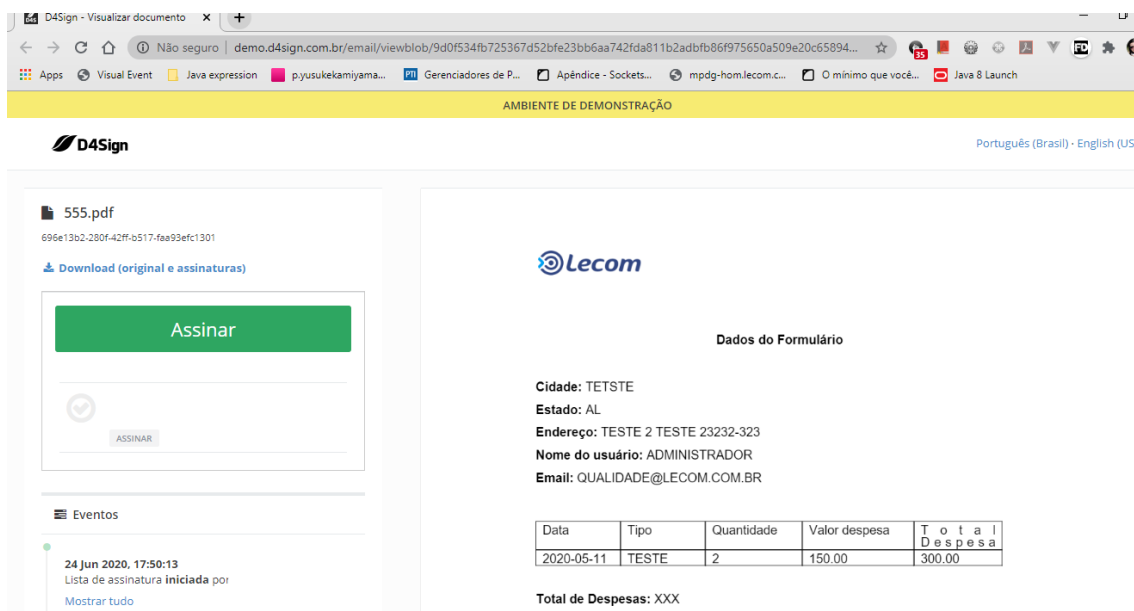
Integration InsereArquivoD4Sign executada: Documento inserido para assinatura no D4Sign, os devidos assinantes receberam notificações para assinar

VOLTAR
↶

O e-mail recebido pelo usuário informado para ser assinante é como este abaixo contendo informações do usuário em questão cadastrado no D4Sign:



Ao clicar no visualizar documento a tela que aparecerá para o assinante será como essa abaixo:



Ao clicar em assinar ele solicita a senha do cadastro do usuário ou apenas algumas informações para realizar a assinatura, e já recarrega a tela com o documento visualizado contendo assinaturas e ao lado atualiza a informação falando que o documento foi assinado, conforme a abaixo.

AMBIENTE DE DEMONSTRAÇÃO


Português (Brasil) · English (US)

555.pdf
696e13b2-280f-42ff-b517-faa93efc1301

[Download \(original e assinaturas\)](#)



ASSINOU

Eventos

- 24 Jun 2020, 17:59:17 **Assinou** (Conta #fff0a2a9-9f)

[Mostrar tudo](#)
- 24 Jun 2020, 17:50:13 Lista de assinatura **Iniciada** por

[Mostrar tudo](#)



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

Ao voltar a área do Lecom BPM temos um processo parado na atividade final com o documento assinado:


Receber Documento Assinado Ciclo: 01
Início da Atividade
24/06/2020


Etapa de Finalização.

Documento

Documento a ser assinado:


555.pdf


Documento assinado:


555.pdf-D4Sign.pdf

Assinantes

Nome Assinante:

Email Assinante:

FINALIZAR 

Ao baixar o documento a segunda página dele contém as assinaturas que foram feitas:

D4Sign 059e13b2-280f-42ff-b517-faa93efc1301 - Para confirmar as assinaturas acesse http://demo.d4sign.com.br/verificar/ Documento assinado eletronicamente, conforme MP 2.200-2/01, Art. 1ºº, Iº, §2.

2 páginas - Datas e horários baseados em Brasília, Brasil
Sincronizado com o NTP.br e Observatório Nacional (ON)
 Certificado de assinaturas gerado em 24 de junho de 2020, 18:02:59

555.pdf
Código do documento 696e13b2-280f-42ff-b517-faa93efc1301