



A customer-centric IT company.

MANUAL DE INSTALACION / CONFIGURACION

Bulk V2

Claro – Campañas efectivas

Control de Versiones

Fecha	Versión	Tipo de Cambio	Descripción	Autor
31-08-2021	0.4	Creación del documento	Primera Versión	SQDM

1. Índice

RECURSOS DE HARDWARE	3
Servidores	4
Estaciones cliente	5
Conectividad	5
RECURSOS DE SOFTWARE	5
ARQUITECTURA PROYECTO	6
Carpetas	7
paquetes y clases	9
graells.bulkapiv2	9
Bulkapiv2Application.java	9
Conexion.java	15
utils	16
ManageFile.java	16
INSTALACIÓN Y CONFIGURACIÓN DEL SISTEMA	18
ERRORES CONOCIDOS	18

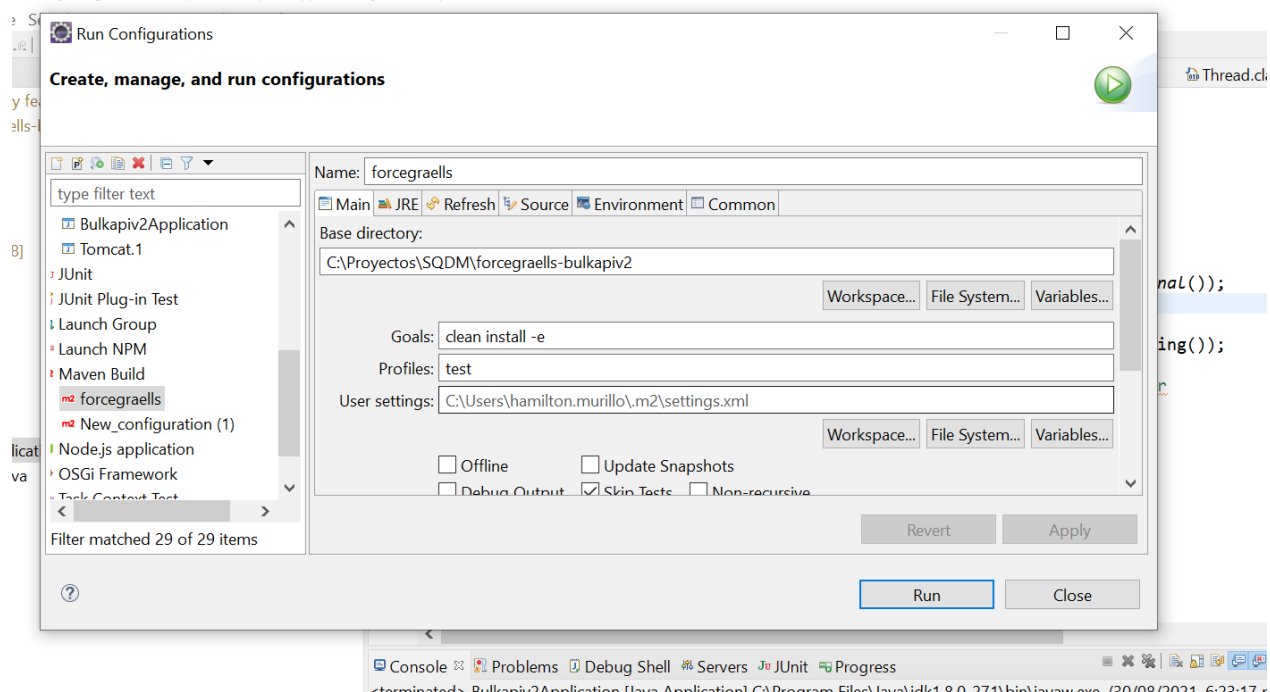
2. RECURSOS DE HARDWARE

2.1 Servidores

El proyecto se compila en una versión de maven 4.0 y java 8, el tipo de archivo generado después de la compilación es un jar

El proyecto se puede compilar desde eclipse, utilizando la siguiente configuración, el directorio base se acomoda a la ubicación donde está ubicado el proyecto, Es importante marcar la casilla para saltar los test

/main/java/graells/bulkapiv2/Bulkapiv2Application.java - Eclipse IDE



El .jar quedará en el directorio target del proyecto.

Para la ejecución del proyecto se debe usar la siguiente línea

```
C:\Proyectos\SQDM\forcegraells-bulkapiv2\target>java -jar bulkapiv2-0.0.1-SNAPSHOT.jar C:\Proyectos\SQDM\forcegraells-bulkapiv2\target\application.properties.properties_
```

La ruta del archivo .properties varía según su ubicación.

Para la ejecución del .jar se deben tener las siguientes recomendaciones en el archivo properties:

- Verificar que la ruta de la carpeta de entrada como salida correspondan a ubicaciones reales
- Validar que los parámetros para la creación del job concuerde con la plantilla que se va a subir

2.2 Estaciones cliente

No aplica

2.3 Conectividad

No aplica

3. RECURSOS DE SOFTWARE

Recursos requeridos

Para realizar el desarrollo se utilizó el IDE Eclipse (Versión: 2020-12 (4.18.0)) con los siguientes complementos.

Nombre	Versión
Maven	4
Java	1.8

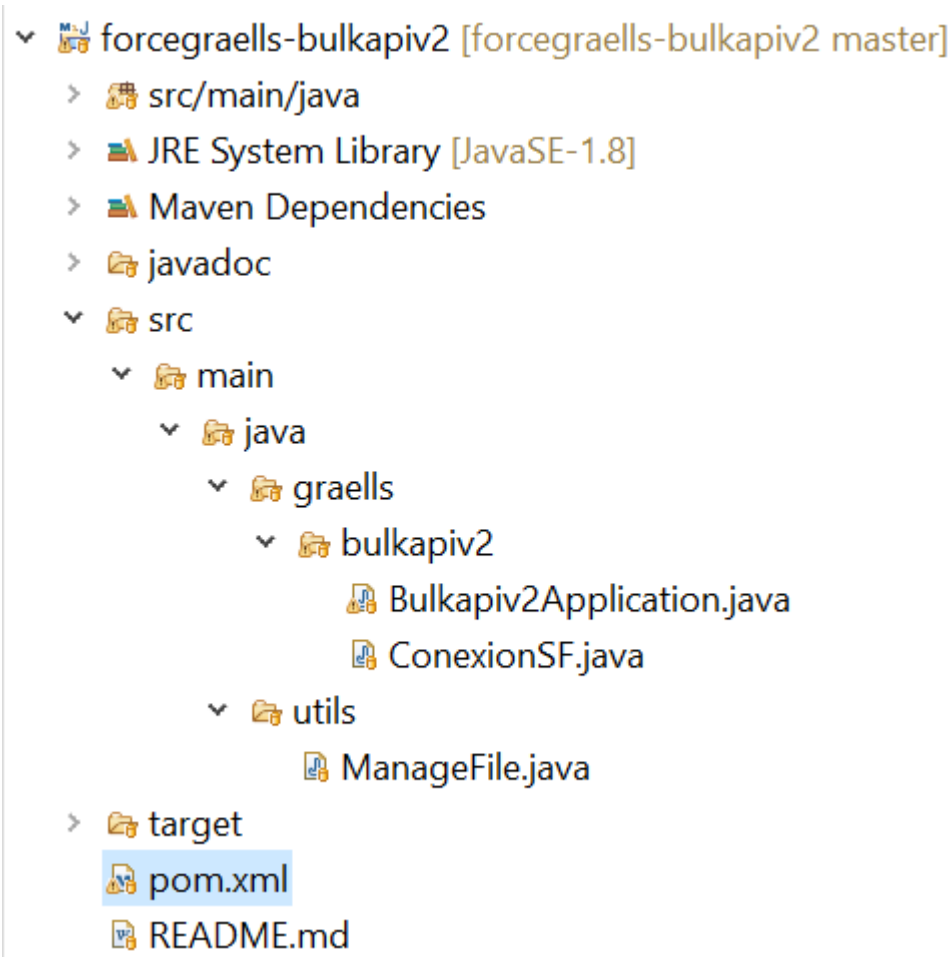
Dependencias

Nombre	Versión
net.minidev	2.3
org.apache.commons	1.3.2
org.glassfish.jersey.core	2.26
org.json	20160810
org.apache.httpcomponents	4.5.2
org.springframework.boot	

4. ARQUITECTURA PROYECTO

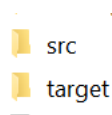
4.1 Carpetas

En el ide Eclipse versión (2020-12 (4.18.0)) en la parte de la izquierda desplegamos las carpetas del proyecto y observamos su estructura en forma de árbol de directorios





El proyecto contiene la siguiente estructura de carpeta








forcegraells-bulkapiv2: contiene las carpetas src y target





forcegraells-bulkapiv2/src: contiene la carpeta main

-  main
-  test

forcegraells-bulkapiv2/target: En esta carpeta se encuentran los .class, se guarda el jar y se recomienda ubicar el archivo application.properties.properties

-  classes
-  entrada
-  logs
-  maven-archiver
-  maven-status
-  salida
-  test-classes
-  application.properties.properties
-  application.properties2.properties
-  bulkapiv2-0.0.1-SNAPSHOT.jar
-  bulkapiv2-0.0.1-SNAPSHOT.jar.original

forcegraells-bulkapiv2/src/main/java: contiene las carpetas graells y utils, en estas se encuentra toda la lógica de la aplicación

-  graells
-  utils

4.2 paquetes y clases

- ▼ 📁 src/main/java
 - ▼ 📁 graells.bulkapiv2
 - > 📄 Bulkapiv2Application.java
 - > 📄 ConexionSF.java
 - ▼ 📁 utils
 - > 📄 ManageFile.java

graells.bulkapiv2

Este paquete contiene las clases Bulkapiv2Application.java y Conexion.java.

Bulkapiv2Application.java

Esta es la clase principal donde se encuentran los métodos de consumo del salesforce APis, contiene los siguientes métodos:

leerPropiedades()

Recibe como parámetro la ruta donde se encuentra el archivo application.properties.properties extrae las propiedades, el cual contiene los datos de conexión, parámetros para la creación del job y las rutas de los ficheros de entrada, salida y logs

```
| private static void leerPropiedades(String[] args) throws IOException {  
    String path = "";  
  
    if (args.length > 0 ) {  
        path = (args[0] == null || args[0].isEmpty()) ? "." : args[0];  
    }else{  
        System.out.println("Error: debe informarse como parámetro el path al fichero de propiedades");  
        System.exit(0);  
    }  
  
    InputStream input = new FileInputStream(path);  
    prop.load(input);  
}
```

loginORG()

Realiza una conexión con la ORG indicada en los miembros de la clase, establece los valores de token y urlInstance si se consigue un login correcto, los parámetros se encuentran en el archivo application.properties.properties

```
private static void loginORG() throws Exception {

    ConexionSF con = new ConexionSF(
        prop.getProperty("USERNAME"),
        prop.getProperty("PASSWORD"),
        prop.getProperty("TOKEN"),
        prop.getProperty("LOGIN_URL"),
        prop.getProperty("CLIENT_ID"),
        prop.getProperty("CLIENT_SECRET")
    );

    if (con != null) {
        token = con.getLoginAccessToken();
        urlInstance = con.getLoginInstanceUrl();
    } else {
        throw new Exception("No se pudo establecer conexión con la ORG");
    }
}
```

crearJobSimple()

hace una petición post a la URL /services/data/vXX.X/jobs/ingest para crear un nuevo job, con las cabeceras que indica el documento, se debe ser muy cuidadoso con los saltos de línea en el payload, retorna el id del job creado el cual se utiliza para realizar el resto de operaciones.

```
private static Boolean crearJobSimple() throws Exception {

    HttpPost petitionPost = new HttpPost(urlInstance + "/services/data/" + prop.getProperty("API_VERSION") + "/jobs/ingest");

    //Cabeceras según indica el documento
    petitionPost.setHeader("Content-Type", "application/json");
    petitionPost.setHeader("Accept", "application/json");
    petitionPost.setHeader("Authorization", "Bearer " + token);

    StringEntity requestEntity = new StringEntity(getPayload(), ContentType.APPLICATION_JSON);
    petitionPost.setEntity(requestEntity);

    jobId = enviarPetitionPost(petitionPost);

    return (!jobId.isEmpty());
}
```

getPayload()

Verifica si los datos obtenidos del archivo properties estan vacios para construir el payload, si estan vacios no se tiene en cuenta para construir el payload, retorna el payload solo con los datos validos

```
private static String getPayload() {
    String payload;

    String externalIdFieldName = prop.getProperty("EXTERNAL_ID");
    if(!externalIdFieldName.equals("")){
        externalIdFieldName = "\"externalIdFieldName\":" + "\"" + prop.getProperty("EXTERNAL_ID") + "\"" + ",";
    }

    String object = prop.getProperty("OBJETO_DESTINO");
    if(!object.equals("")) {
        object = "\"object\":" + "\"" + prop.getProperty("OBJETO_DESTINO") + "\"" + "," ;
    }

    String contentType = prop.getProperty("FORMATO_FICHERO_DATOS");
    if(!contentType.equals("")) {
        contentType = "\"contentType\":" + "\"" + prop.getProperty("FORMATO_FICHERO_DATOS") + "\"" + ",";
    }

    String operation = prop.getProperty("OPERACION_BULK");
    if(!operation.equals("")) {
        operation = "\"operation\":" + "\"" + prop.getProperty("OPERACION_BULK") + "\"" + ",";
    }

    String lineEnding = prop.getProperty("CARACTER_FINAL_LINEA");
    if(!lineEnding.equals("")) {
        lineEnding = "\"lineEnding\":" + "\"" + prop.getProperty("CARACTER_FINAL_LINEA") + "\"" + ",";
    }

    String columnDelimiter = prop.getProperty("SEPARADOR_COLUMNAS");
    if(!columnDelimiter.equals("")) {
        columnDelimiter = "\"columnDelimiter\":" + "\"" + prop.getProperty("SEPARADOR_COLUMNAS") + "\"";
    }

    //Body: los saltos de lineas son fundamentales.
    payload =
        "{"+object + contentType + operation + externalIdFieldName +
        lineEnding + columnDelimiter + "}\\n";

    return payload;
}
```

enviarPeticiónPost()

Envía una petición Post preconfigurada y retorna el id del job

```

private static String enviarPeticionPost(HttpPost post) {
    String jobId = null;

    try {
        HttpClient client = HttpClientBuilder.create().build();
        HttpResponse httpResponse = client.execute(post);

        String respuesta = IOUtils.toString(httpResponse.getEntity().getContent());

        if(respuesta.contains("error")) {
            String textLogs = respuesta;
            textLogs = textLogs.concat(" Por favor verifique los parametros dentro del archivo .properties");
            String ruta = prop.getProperty("PATH_FAILED_RESULT") + "logs.log";
            ManageFile.writeFile(textLogs, ruta, "");
        }

        JSONObject jsonObject = new JSONObject(respuesta);
        jobId = jsonObject.getString("id");

    } catch (Exception e) {
        e.printStackTrace();
    }

    return jobId;
}

```

enviarDatos()

Se encarga de hacer la carga de los datos del CSV

- * Se realiza a posteriori de su creación a la creación del job.
- * Por tanto, esta función realiz un PUT sobre la url que se obtiene del Job, para cargar en el payload
- * el contenido del CSV y enviarlo.
- * Recaltar la importancia del contenido de las cabeceras siguiendo el documento oficial

```

private static String enviarDatos() throws Exception {
    JSONObject jobInfo = null;
    if (jobId == null) {
        throw new Exception("Job id es null");
    } else {
        //URL is provided in the contentUrl field in the response from Create a Job, or the response from a Job Info request on an open job.
        jobInfo = obtenerInfoJob();
    }

    String urluploadDatos = null;
    if (jobInfo != null) {
        urluploadDatos = jobInfo.get("contentUrl").toString();
    }

    HttpClient client = HttpClientBuilder.create().build();
    HttpPut req = new HttpPut(urlInstance + "/" + urluploadDatos);

    //Cabeceras
    req.setHeader("Content-Type", "text/csv"); //Atención esta header debe indicar el Content-Type adecuado
    req.setHeader("Accept", "application/json");
    req.setHeader("Authorization", "Bearer " + token);

    String payload = LeerFicheroDatos(prop.getProperty("PATH_FICHERO_DATOS_ENTRADA"), UTF_8);

    StringEntity requestEntity = new StringEntity(payload, ContentType.TEXT_PLAIN);
    req.setEntity(requestEntity);

    HttpResponse response = client.execute(req);

    Log.info("Resultado del envio de datos: " + response.getStatusLine().toString());

    return response.getStatusLine().toString();
}

```

abortarCerrarJob()

Recibe como parámetro la operación que se va a realizar

Llamada PATCH sobre la url /services/data/vXX.X/jobs/ingest/jobID para indicar a Salesforce que ejecute un cambio de estado sobre el job, para cerrarlo y que se proceda a su ejecución o para cancelarlo

```

private static void abortarCerrarJob(String operacion) throws Exception {
    if ((jobId == null) || operacion.isEmpty()) {
        throw new Exception("Id del Job es null");
    }

    //Petición GET sin parámetros ni Body
    HttpClient client = HttpClientBuilder.create().build();
    HttpPatch req = new HttpPatch(urlInstance + "/services/data/" + prop.getProperty("API_VERSION") + "/jobs/ingest/" + jobId);
    System.out.println(req);
    //Cabeceras
    req.setHeader("Accept", "application/json");
    req.setHeader("Content-Type", "application/json; charset=UTF-8");
    req.setHeader("Authorization", "Bearer " + token);

    //Body
    String payload = null;

    if (operacion.equals("ABORTAR")) {
        payload = "{" + "\"state\": \"Aborted\"" + "}";
    } else if (operacion.equals("CERRAR")) {
        payload = "{" + "\"state\": \"UploadComplete\"" + "}";
    }

    StringEntity requestEntity = new StringEntity(payload, ContentType.APPLICATION_JSON);
    req.setEntity(requestEntity);

    //Envío de la petición
    HttpResponse response = client.execute(req);

    String respuesta = IOUtils.toString(response.getEntity().getContent());
    Log.info("Resultado de cierre/abortar q: " + respuesta);
}

```

poolingJobInfoHastaEstadoFinal()

Realiza pooling de la info de un Job, cada n segundos, hasta que el Job finaliza en estado "JobComplete" ó "Failed"

```
private static String poolingJobInfoHastaEstadoFinal() throws Exception {
    if (jobId == null) {
        throw new Exception("Job id es null");
    }

    Integer MAX_POOLING = Integer.parseInt(prop.getProperty("MAX_POOLING"));
    Integer MILLIS_POOLING = Integer.parseInt(prop.getProperty("MILLIS_POOLING"));

    String estadoJob = obtenerInfoJob().get("state").toString();
    List<String> estadosFinales = Arrays.asList("JobComplete", "Failed");

    Integer poolCounting = 0;
    while ( (!estadosFinales.contains(estadoJob)) && (poolCounting < MAX_POOLING) ) {
        poolCounting++;

        Log.info("El estado del job es: " + estadoJob);
        try {
            Thread.sleep(MILLIS_POOLING);
        } catch (InterruptedException e) {
            System.out.println(e);
        }
        estadoJob = obtenerInfoJob().get("state").toString();
    }

    if(estadosFinales.contains(estadoJob)) {
    }

    if ( poolCounting == MAX_POOLING ) return "Timeout de espera";

    return estadoJob;
}
```

obtenerInfoJob()

retorna un JSONObject

Get a la url /services/data/vXX.X/jobs/ingest/jobID para obtener la info de un Job

```
private static JSONObject obtenerInfoJob() throws Exception {
    if (jobId== null) {
        throw new Exception("Id del Job es null");
    }

    //Petición GET sin parámetros ni Body
    HttpClient client = HttpClientBuilder.create().build();
    HttpGet req = new HttpGet(urlInstance + "/services/data/" + prop.getProperty("API_VERSION") + "/jobs/ingest/" + jobId);

    //Cabeceras
    req.setHeader("Accept", "application/json");
    req.setHeader("Content-Type", "application/json; charset=UTF-8");
    req.setHeader("Authorization", "Bearer " + token);

    HttpResponse response = client.execute(req);
    String respuesta = IOUtils.toString(response.getEntity().getContent());

    JSONObject respuestaJson = null;

    if (!respuesta.contains("error") && !respuesta.contains("Error")) {
        respuestaJson = new JSONObject(respuesta);
    } else {
        String ruta = prop.getProperty("PATH_FAILED_RESULT") + jobId+ ".log";
        ManageFile.writeFile(respuesta, ruta, jobId);
    }

    return respuestaJson;
}
```

failedResults()

Llamada PATCH sobre la url /services/data/vXX.X/jobs/ingest/jobID/failedResults para consultar los resultados fallidos para luego guardar el resultado en un archivo

```

private static void failedResults() throws Exception {
    if (jobId == null) {
        throw new Exception("Id del Job es null");
    }

    //Petición GET sin parámetros ni Body
    HttpClient client = HttpClientBuilder.create().build();
    HttpGet req = new HttpGet(urlInstance + "/services/data/" + prop.getProperty("API_VERSION") + "/jobs/ingest/" + jobId + "/failedResults/");

    //Cabeceras
    req.setHeader("Accept", "application/json");
    req.setHeader("Content-Type", "application/json; charset=UTF-8");
    req.setHeader("Authorization", "Bearer " + token);

    HttpResponse response = client.execute(req);
    String respuesta = "";
    respuesta = IOUtils.toString(response.getEntity().getContent());

    String ruta = prop.getProperty("PATH_FAILED_RESULT") + jobId + ".log";
    ManageFile.writeFile(respuesta, ruta, jobId);

    //mueve archivo, carpeta salida
    String folder_int = prop.getProperty("PATH_FICHERO_DATOS_ENTRADA");
    String folder_out = prop.getProperty("PATH_FICHERO_DATOS_SALIDA");
    ManageFile.changeFolder(folder_int, folder_out);
}

```

Conexion.java

En esta clase se hace la petición para obtener el token de acceso

```

import org.apache.http.HttpResponse;

class ConexionSF {

    static final String GRANTSERVICE = "/services/oauth2/token?grant_type=password";

    private String loginAccessToken = null;
    private String loginInstanceUrl = null;
    private HttpPost httpPost = null;

    //Constructor
    public ConexionSF(String username, String password, String token, String loginURL, String clientId, String clientSecret) {

        HttpClient httpClient = HttpClientBuilder.create().build();

        // Assemble the login request URL

        String URLToLogin = loginURL + GRANTSERVICE + "&client_id=" + clientId + "&client_secret=" + clientSecret + "&username=" + username + "&password=" + password;

        System.out.println(URLToLogin);

        // Login requests must be POSTs
        httpPost = new HttpPost(URLToLogin);
        HttpResponse response = null;

        try {
            // Execute the login POST request
            response = httpClient.execute(httpPost);
        } catch (Exception e) {
            e.printStackTrace();
        }

        // verify response is HTTP OK
        final int statusCode = response.getStatusLine().getStatusCode();
        if (statusCode != HttpStatus.SC_OK) {

```

utils

Este paquete contiene la clase ManageFile.java

ManageFile.java

Esta clase contiene los siguientes métodos.

writeFile()

recibe como parámetro la respuesta de la petición a los elementos fallidos, la ruta donde va a quedar guardado el archivo(esta se encuentra en el archivo .properties y el id del job para nombrar el archivo)

Crea archivo con los resultados fallidos

```
public static void writeFile(String respuesta, String ruta, String jobId) {
    try {
        DateTimeFormatter time = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        String complement = "{" + time.format(LocalDateTime.now()) + "}" + " jobId:" + jobId + " response: ";
        respuesta = complement.concat(respuesta);
        File file = new File(ruta);
        if (!file.exists()) {
            file.createNewFile();
        }

        FileWriter fw = new FileWriter(file);
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write(respuesta);
        bw.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

changeFolder()

Recibe como parámetros la carpeta de entrada y la carpeta de salida ambas rutas se encuentran en el archivo .properties

mueve el archivo de la carpeta de entrada a la carpeta de salida cuando se ha procesado


```
public static void changeFolder(String folder_int, String folder_out) {
    File fold_int = new File(folder_int);
    String name = fold_int.getName().replace(".csv", "");
    DateTimeFormatter time = DateTimeFormatter.ofPattern("yyyy-MM-dd_HH:mm:ss");
    name = name.concat(time.format(LocalDateTime.now())).concat(".csv");
    folder_out = folder_out.concat(name);

    Path inputFolder = FileSystems.getDefault().getPath(folder_int);
    Path outputFolder = FileSystems.getDefault().getPath(folder_out);

    try {
        Files.move(inputFolder, outputFolder, StandardCopyOption.REPLACE_EXISTING);
    } catch (IOException e) {
        System.err.println(e);
    }
}
```

5. INSTALACIÓN Y CONFIGURACIÓN DEL SISTEMA

Se deben tener en cuenta las siguientes recomendaciones en el archivo `application.properties.properties`

```
#Fichero Datos
PATH_FICHERO_DATOS_ENTRADA = C:/Proyectos/SQDM/forcegraells-bulkapiv2/target/entrada/borrarClientes.csv
PATH_FICHERO_DATOS_SALIDA = C:/Proyectos/SQDM/forcegraells-bulkapiv2/target/salida/
PATH_FAILED_RESULT = C:/Proyectos/SQDM/forcegraells-bulkapiv2/target/logs/
```

Los valores registrados en las propiedades son usados como ejemplo para este manual, se deben reemplazar por las rutas donde se encuentren los archivos y donde se desean guardar los resultados.

6. ERRORES CONOCIDOS

No encuentra el Archivo

Posibles soluciones

- verificar que la ruta de la carpeta donde se guardó el archivo si está bien

referenciada.

- verificar que el nombre del archivo si es correcto

```
PATH_FICHERO_DATOS_ENTRADA =
```

La ruta del archivo de entrada es correcta pero no lo reconoce

posible solución

- verificar que la ruta del archivo no contenga caracteres como ñ o tilde

Fallo en la creación del JOB

posible solución

- Verificar que los parámetros en el archivo .properties si sean los correctos según el tipo de operación que se desea hacer.

```
#Propiedades para la creación de Job
OBJETO_DESTINO = Account
FORMATO_FICHERO_DATOS = CSV
OPERACION_BULK = upsert
EXTERNAL_ID = CAM_Llave__c
CARACTER_FINAL_LINEA = CRLF
SEPARADOR_COLUMNAS = SEMICOLON
MAX_POOLING = 10
MILLIS_POOLING = 3000
```

Fallo en la obtención del token de acceso

posible solución

- Verificar que los parámetros en el archivo application.properties.properties sean los correctos.

```
USERNAME = ca  
PASSWORD = An  
TOKEN = XITFg  
LOGIN_URL = h  
CLIENT_ID = 3  
CLIENT_SECRET  
API_VERSION =
```

No se encuentran las rutas donde se deben guardar los archivos

posibles soluciones

- Se debe verificar las rutas en el archivo application.properties.properties