Auto Verification of Data Post Migration

- **Overview-**

Contributors- Sampada Joshi
Sanyukta Muley
Prajakta Deokule

In this project we are doing verification of data after the data is migrated from source to target machine.

- **Context-**

During migration, some data might be overwritten or skipped hence verification becomes important.
If a customer starts using the target system with a new version and finds out of sync data, the issue might escalate to customer support.
The verification is necessary to ensure the same experience on both sides and detect any missing data.
The data to be verified includes configurations, the feature settings, metadata and the transactional data.
The auto verification is useful when a software has newer versions to be migrated and to keep data intact.

- **Goals-**

The project aims to -

- ➢ Check if the required data is transferred
- ➢ Validation of correct values in destination tables
- ➢ Verification of any data loss

To measure the success the details about how much data was transferred, what failed to be transferred , correct and incorrect values are displayed in an .xlsx format.

- **Non Goals-**

Only the details about the verification and validation of the data is displayed. There is no attempt towards correction of the data migrated in case of any data loss or if any incorrect data found.

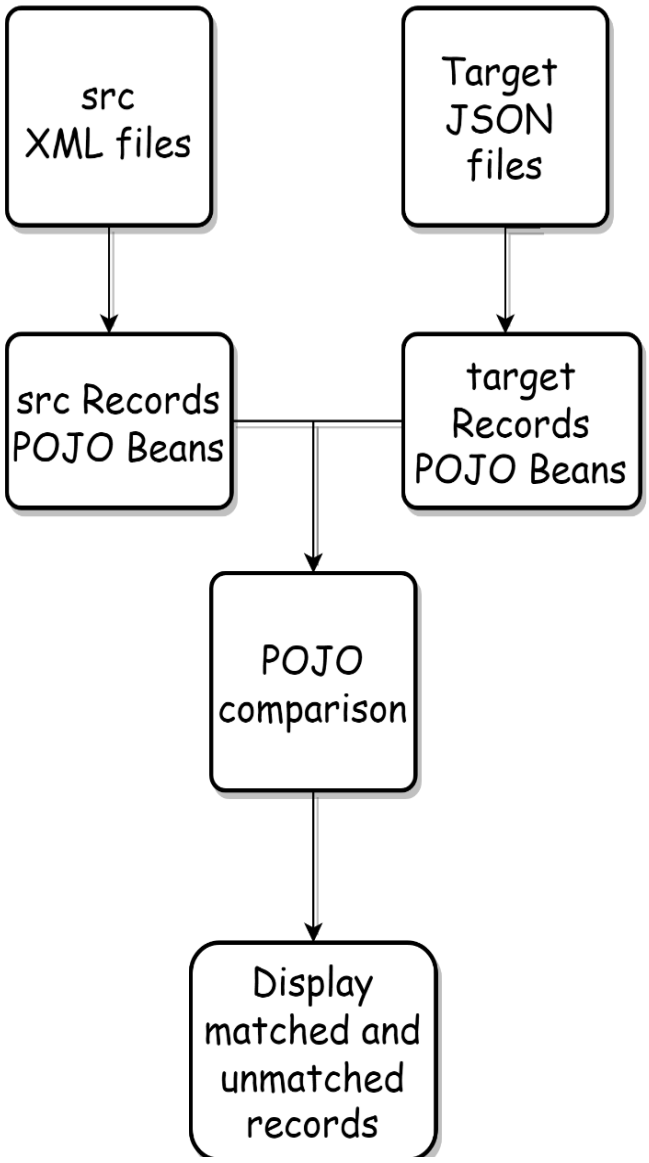
Technology -

- Backend Framework(Spring)
- Language : Java
- Rest API
- Markup Language: XML
- JSON
- Excel
- Swing
- Apache poi
- BMC Product
- BMC Helix Platform

- **Proposed Solution-**

The export utility is the start point of the data migration.The src DB contains the records of the BMC Helix digital workplace. These records are fetched in the form of XML files.
Each file corresponds to a record in DWP. These XML files are deserialized into respective POJOs to verify the fields values.

Similarly, the import utility imported into the target DB is fetched through REST APIs. The response received is in the form of JSON which is then deserialized into POJOs for further field value comparisons and validations.



➢ **Source side implementation-**

The XML files which are to be exported are fetched from the xml source file path provided in configuration file .Parsing of xml files is done using Jsoup and converted it to Document.This document elements are fetched and deserialized into POJO objects using JAXB unmarshalling.

```
@Override
public String getValue() {
    String properties = String.valueOf(ConfigProperties.properties.get("source.appointment"));
    String value = "";
    if (StringUtils.isNotEmpty(properties)) {
        List<String> lstProperties = Arrays.asList(properties.split("\\s*,\\s*"));
        for(String property: lstProperties) {
            value += InvokeUtil.invokeGetter(this, property) + ";";
        }
    }
    return value;
}
```

The records are returned in the form of a list of the POJO class type.

The POJO classes of all the records in DWP are formed respectively for source and target.

These POJOs return the required field values for comparison and data verification.

Each attribute value in the list of data record is fetched through getValue() method for the required attributes as specified in the configuration file which can be updated as per the requirements. Fetched data is then converted to string format and added to List of records and given for comparison.

POJOs for Appointment record on the source side-

```
public String getTenantId() {

    return tenantId;
}


public void setTenantId(String tenantId) {
    this.tenantId = tenantId;
}


public String getDate() {
    return date;
}

public void setDate(String date) {
    this.date = date;
}


public String getIncidentId() {

    return incidentId;
}


public void setIncidentId(String incidentId) {
    this.incidentId = incidentId;
}
```

➢ **Target side implementation-**

The first call is made to get data in the form of a list of objects of the type POJO class created through REST APIs.
The dataPageQueryService calls getData method passing the parameters fieldIDs of attributes, record name in DWP and the POJO class name of the corresponding record.

DATA_PAGE_QUERY_API = "/api/rx/application/datapage?dataPageType=com.bmc.arsys.rx.application.record.datapage.RecordInstanceDataPageQuery&pageSize=-1&startIndex=0&shouldIncludeTotalSize=false";

In the getData() method of datapageQueryService, there is a RequestURL for fetching data from target using the REST endpoint URL.
The response is then provided by the executeGet method of IRestHelper interface implemented in InnovationSuiteHelper provided the media type as JSON.
The executeGET method has endpoint URL, media type(JSON) which is the requested Response type and additional headers.This executeGET method calls the execute method of REST Client which requires login before getting the data.

Fetching the values of username,password and base URL from applications.properties file using @value method the Innovation Suite Login is made possible for the user.

The login is posted by REST Client passing the login URL, headers and request Body(This involves a string that contains the username and the password along with en as language) and media type(JSON type).The login url includes "http://clm-pune-v8cd6.bmc.com:8008"+/api/rx/authentication/loginrequest"

The status code of the response returned is checked.If the status code is 200 then the login is successful else the Innovation Suite Login has failed for the user.

The responses returned enable the execute method to get data through HTTP.GET method in the form of JSON.

In InnovationSuiteRestHelper, there are methods for HTTP get , HTTP post and HTTP Put .

GET: Get data from the server.

POST: Create resource to the server.

HTTPGet :method implemented in the RestClientImpl returns the required JSON file.

This JSON file is deserialized to POJO beans using the GSON library and the list of records is returned.

Each attribute value in the list of data records is fetched through getValue() method for the required attributes as specified in the configuration file which can be updated as per the requirements.

POJO for Appointment record from target JSON file-

```
package com.bmc.utility.vo.target;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonIgnoreProperties(ignoreUnknown = true)
public class Appointment extends Base {

    public static final String RECORD_DEF_NAME = "Appointment";
    public static final String FIELD_START_DATE_TIME_ID = "420003505";
    public static final String FIELD_INCIDENT_ID = "420003506";
    public static final String FIELD_INCIDENT_NUMBER_ID = "420003507";
    public static final String FIELD_NOTES_ID = "420003508";
    public static final String FIELD_USER_ID_ID = "420003511";

    @JsonProperty(FIELD_START_DATE_TIME_ID)
    private Double startDateTime;

    @JsonProperty(FIELD_INCIDENT_ID)
    private String incidentId;

    @JsonProperty(FIELD_INCIDENT_NUMBER_ID)
    private String incidentNumber;

    @JsonProperty(FIELD_NOTES_ID)
    private String notes;

    @JsonProperty(FIELD_USER_ID_ID)
    private String userId;

    public Double getStartDateTime() {
        return startDateTime;
    }

    public void setStartDateTime(Double startDateTime) {
        this.startDateTime = startDateTime;
    }
```

➢ **Comparison of Source and target POJOs -**

The comparison takes place once the target and source side data for a particular record are fetched.

The comparison between the records takes place on the basis of a primary key which remains constant and other attribute values are compared for verification.

The comparison takes place for all the tables specified in the configuration file.We are converting the POJO objects to Strings and storing them in a list.Then we are using the removal method to remove the target records from the source list and these remaining records on the source side are considered as unmatched records.

- **Alternate approach considered for comparing source and target POJOs -**

  Overriding hashcode() and equals() method.This approach was not suitable as it required creating a new Class for every class object that had to be compared.This class had the common attributes to be compared.

Configuration file-

An application.properties file outside the jar is provided to include and update the table names, attributes, any user login details, mapping of values in source and target.

Table.name specifies the table name, name of the xml file and the attribute IDs required on the target side.

The file path from where the Xml files are to be read is stored here.The excel file path where the unmatched records will be inserted is also specified here.

Value.mapping specifies the mappings on the source and target.For example, the OS type has value WEB on the source side records but on the target side the value gets mapped with 3.These mappings are considered so that this does not come under unmatched data as only the data representation format is different.

The source.table_name and target.table_name provides the attributes mentioned to be verified for respective table.

The output .xlsx format file path is provided along with the source side xml file paths.

```
file.path=C:\\Users\\samjoshi\\OneDrive - BMC Software, Inc\\Desktop\\application.properties
output.excel=C:\\Users\\samjoshi\\OneDrive - BMC Software, Inc\\Desktop\\Project\\UnmatchedRecords.xlsx
sourcexml.path=C:\\Users\\samjoshi\\Downloads\\allTenants.export\\


Table.name=device_token:DeviceToken:3,6,379,420003507-\
appointment:Appointment:3,6,37-\
appointment_schedules:AppointmentSchedule:3,6,379-\
administrators:Administrator:379,3,6-\
attachments:Attachment:379,3,6,420003507,420003513-\
branding_configuration:BrandingConfiguration:379,3,6,420003505,420003506-\
branding_state:BrandingState:379,3,6,420003505-\
broadcast_notifications:BroadcastNotification:379,3,6,420003512,420003509-\
cart_items:CartItem:379,3,6,420003505-\
cart:Cart:379,3,6,420003506-\
```

Test Cases-

- The source records are greater than the target records in a table -

Check for the records on the target side and compare the values. For remaining source records the target records were failed to import.A pop up window

appears stating that the import failed and shows the total number of records failed.

- The source records and target records are equal in a table-

If all the values match in the records, no detailed data about unmatched records will be provided in the .xlsx format. A pop up window stating that the import is successful is shown.

- Dynamically change the table names and attributes to be verified -

The updation of configuration file is done to update the table names and its attributes or any other changes in data specifications.

- Mapping values on the source and target side tables-

The values of the attributes may be considered with different mappings on both sides. Therefore, to compare the values on common ground the target side value are mapped with respective source value of the attribute.

For example, the OS type has value WEB on the source side records but on the target side the value gets mapped with 3. The change of 3 on target side takes place every time OS type attribute is requested for a record in a table.

- The change in the data type for the value of the attribute-

The values of the attributes may be integer, strings, boolean or date and time. These were fetched in the specified data type and compared as strings.

For example, The values 0 and 1 for TRUE and FALSE were changed in the common data type.

The date and time format of the createdDate and modifiedDate attributes were also taken as Date type and parsed to string values.

- Same value attributes discarded-

While comparing the records the attributes which are same are discarded and only unmatched attributes are displayed in .xlsx format. So that user will get a clear picture about the unmatched attributes and values.

- Configuration file outside jar

Configuration file is placed outside the jar. Changes can be made to config file and the utility will run accordingly.

Data verification Performed on Tables from DigitalWorkspace-

Appointment

AppointmentSchedule

Administrator

Attachment

BrandingConfiguration

BrandingState

BroadcastNotification

CartItem

Cart

CategoryConfiguration

DeviceToken

FeatureSettings

HowTo

Icon

LocationFloorMapAsset

LocationFloorMap

Location

NotificationContentTemplate

NotificationPreferences

Page

PrivacyNotice

PrivacyPolicy

Provider

ProviderSettings

ContactInformation

UnifiedCatalogItem

PushNotificationCert

SrdInfo

SrdSettings

QuickPickPreferences

ScheduleSlot

UserPreferences

UserSession

Dependencies-

1. org.springframework.boot (spring-boot-starter) project

2. org.glassfish.jersey.ext (jersey-spring4 version- 2.34)
3. org.glassfish.jersey.media(jersey-media-json-jackson version- 2.35)
4. org.apache.commons (commons-lang3 and commons-collections4)
5. jakarta.ws.rs-api (version- 2.1.6)
6. com.google.code.gson (version- **2.8.5**.)
7. javax.xml.bind jaxb-api (version-2.3.1)
8. org.glassfish.jaxb (jaxb-runtime)
9. org.jsoup (jsoup version 1.14.3)
10. org.apache.poi [poi version- 5.2.2 and poi-ooxml version- 5.2.2]

User Interface-

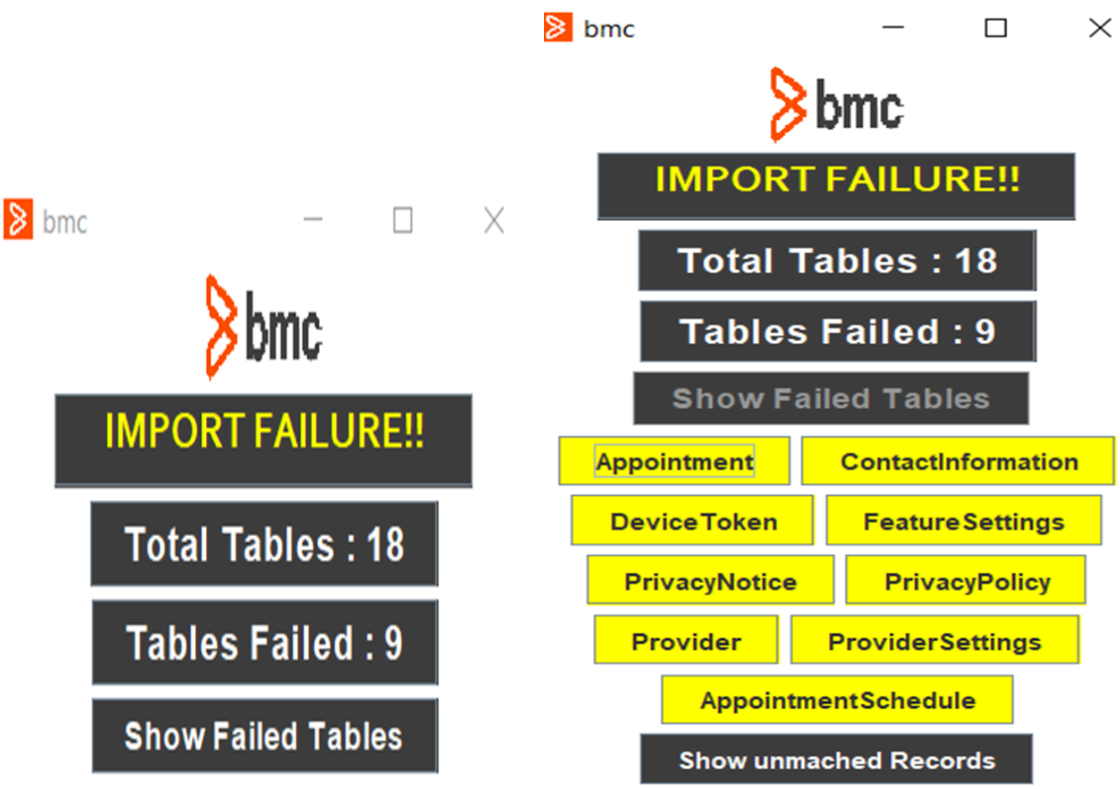
Responsive UI has been created using java swing package.

Frame,Panel and buttons are created using JFrame,JPanel, and JButton inbuilt classes.

Formating like changing button color,size, frame size, color , font size, font style is done for better UI. Logo icon image is added using ImageIcon in the swing package.

Using addActionListener() after clicking action is performed

Ex- after clicking “Show Failed Tables” Names of tables are displayed



Output file (UnmatchedRecords.xlsx) -

The file path of the .xlsx file can be provided in the configuration file.

The file contains the details of the records with incorrect values imported on the target for all the tables. The file also shows a pie chart showing the record comparison.

The Apache POI API is used to create, modify and display the data in the .xlsx format.

The file shows the unique id for each record and the unmatched attributes for the records for the table failed to import.

18				
19				
20	Appointment	id	tenantId	notes
21	Source Record	d6263dde-4d5b-4c11-97f0-a8b7aaaeada1	1	
22	Target Record		AGGADG1AANVNNARJI9CCRJI9CCBCA	
23	Source Record	e626ab7c-c6fe-4f02-b08d-8484eef34155	1	ABC
24	Target Record		AGGADG1AANVNNARJI9CCRJI9CCBCA	New
25	Source Record	1c27f87a-0b14-4d84-82cd-7c81dab27661	1	
26	Target Record		AGGADG1AANVNNARJI9CCRJI9CCBCA	
27	Source Record	70e750c4-5b6a-4320-a065-9f25bebb58cd	1	
28	Target Record		AGGADG1AANVNNARJI9CCRJI9CCBCA	
29	Source Record	d63b241d-5738-4e8d-8969-abb528b457b6	1	
30	Target Record		AGGADG1AANVNNARJI9CCRJI9CCBCA	
31	Source Record	a1490a0b-0a6f-4f54-9ba5-233cf1f71e33	1	
32	Target Record		AGGADG1AANVNNARJI9CCRJI9CCBCA	
33				
34				
35	ContactInformation	id	name	
36	Source Record	58632e23-5f20-428e-81a3-460d0e73918a	General no Help	
37	Target Record		General Help	

References-

Apache POI- [Overview (POI API Documentation) (apache.org)]

JAXB - [Lesson: Introduction to JAXB (The Java™ Tutorials > Java Architecture for XML Binding (JAXB)) (oracle.com)]

Java Swing- [javax.swing (Java Platform SE 7) (oracle.com)]

Spring Boot- [Spring Boot Reference Documentation]