Integration of Facilities from SIGECA To eLMIS

Background

Goal behind the integartion is to have facilities stored in SIGECA Central integrated into the OpenLMIS platform. This will allow users to use predefined resource and have single point of truth for the resource.

External System Payload Example

Payload comming from the SIGECA Central (based on Mapa Sanitario reosurce):


```
{
    "units": [
        {
            "name": "HEALTH CENTER BELA VISTA",
            "code": "470010",
            "abbreviation": "HC",
            "category": "Health Center",
            "ownership": "Public - National Health Service",
            "management": "Public",
            "municipality": "Ambriz",
            "province": "Bengo",
            "operational": true,
            "latitude": "7.81807",
            "longitude": "1380299",
            "services_offered": [
                {
                    "service_offered_id": 2,
                    "name": "Internal Medicine"
                },
                [...]
            ]
        },
        [...]
    ]
}
```


Mapping of the data

OpenLMIS Facility Payload:

Example payload:

```
{
  "id": "fac-12345",
  "code": "FAC001",
  "name": "Main Health Center",
  "description": "A primary health facility providing general medical services.",
  "geographicZone": {
    "id": "geo-123",
    "code": "GEO01",
    "name": "Zone 1",
    "level": {
      "id": "level-1",
      "code": "LEVEL1",
      "name": "District Level"
    },
    "catchmentPopulation": 50000,
    "latitude": -1.2921,
    "longitude": 36.8219,
    "extraData": {
      "region": "Region A",
      "subRegion": "SubRegion B"
    }
  },
  "type": {
    "id": "type-1",
    "code": "TYPE1",
    "name": "Hospital"
  },
  "operator": {
    "id": "operator-1",
    "code": "OPERATOR1",
    "name": "Health Organization"
  },
  "active": true,
  "goLiveDate": "2023-01-01",
  "goDownDate": "2030-12-31",
  "comment": "This facility is scheduled for expansion in 2025.",
  "enabled": true,
  "openLmisAccessible": true,
  "supportedPrograms": [
    {
      "id": "prog-1",
      "name": "Immunization"
    },
    {
      "id": "prog-2",
      "name": "Maternal Health"
    }
  ]
```

```
}
```

Data Mapping

Here is a table mapping the fields from the source JSON schema to the target JSON schema:

Source Field	Target Field	Notes
`units[].name`	`name`	Direct mapping
`units[].code`	`code`	Direct mapping
`units[].abbreviation`	Not applicable	No corresponding field in target schema
`units[].category`	`type.name`	Assumed to map to the type of facility
`units[].ownership`	`operator.name`	Assumed to map to the facility operator
`units[].management`	Not applicable	No corresponding field in target schema
`units[].municipality`	`geographicZone.name`	Assumed to be a 3rd level zone for geographic zone
`units[].province`	`geographicZone.name`	Assumed to be a 2nd level zone for geographic zone
`units[].operational`	`active`	Direct mapping

Source Field	Target Field	Notes
`units[].latitude`	`latitude`	Direct mapping
`units[].longitude`	`longitude`	Direct mapping
`units[].services_offered[].service_offered_id`	Not Applicable	Not possible to map with internal id due to format.
`units[].services_offered[].name`	`supportedPrograms[].name`	Direct mapping
Not applicable	`id`	Generated or obtained from source
Not applicable	`description`	Not provided
Not applicable	`geographicZone.id`	Generated or obtained from source
Not applicable	`geographicZone.code`	Generated as `gz-name` or obtained from source
Not applicable	`geographicZone.level`	Mapped autimatically based on the name of variable
Not applicable	`geographicZone.catchmentPopulation`	Provided or estimated separately
Not applicable	`type.id`	Generated or obtained from source
Not applicable	`type.code`	Generated as `type.name` or obtained from source

Source Field	Target Field	Notes
Not applicable	`operator.id`	Provided separately
Not applicable	`operator.code`	Provided separately
Not applicable	`goLiveDate`	Provided separately
Not applicable	`goDownDate`	Provided separately
Not applicable	`comment`	Provided separately
Not applicable	`enabled`	Set to `true`
Not applicable	`openLmisAccessible`	Set to `true`

This table outlines how each field in the source data maps to the corresponding field in the target schema, along with notes on assumptions and default values where direct mappings are not applicable. `Obtained from source` means using data that is already available in openLMIS.

Foreign Keys Matching

No matchings will rely on the actual Foreign Keys.

Facility Identification

Facilities will be identified by their `code`. The integration process will follow these steps:

1. **Add New Facility**: If a facility with the given `code` does not exist in the target system, a new facility will be created.
2. **Update Existing Facility**: If a facility with the given `code` exists but its details have changed, the existing facility will be updated with the new details.
3. **Delete Facility**: If a facility present in the target system is not included in the payload, it will be deleted.

Foreign Relations

1. **Geographic Zone**
   - **Mapping**: Facilities will be assigned to geographic zones based on the `municipality` and `province` fields.
   - **Drilldown Logic**:
     - If `municipality` is provided, the facility will be added at the municipality level.
     - If `municipality` is not provided but `province` is provided, the facility will be added at the province level.
     - If neither `municipality` nor `province` are provided, new geographic zones will be created as needed.

2. **Facility Operator**
    - **Mapping**: The `ownership` field in the payload will be matched with the `name` field in facility operators in the target system.
3. **Type of Facility**
    - **Mapping**: The `category` field in the payload will be matched with the `name` field in the target system's type of facility.
4. **Services Offered**
    - **Current Status**: The matching logic for services offered is still under development by another team.
    - **Expected Logic**: It is anticipated that the provision names will match information stored in the supported facilities products. They will be matched based on the name.

Example Mapping Logic

Geographic Zone Example

- Payload:

```
{
    "municipality": "Ambriz",
    "province": "Bengo"
}
```

- Target System:
    - Check if "Ambriz" exists in the geographic zones. If yes, map the facility to this municipality.
    - If "Ambriz" does not exist, check for "Bengo". If "Bengo" exists, map the facility to this province.
    - If neither exist, create new geographic zones as necessary.

Type of Facility Example

- Payload:

```
{
    "category": "Health Center"
}
```

- Target System:
    - Match `category` "Health Center" to the corresponding `name` in the type of facility records.

Facility Operator

- Payload:

```
{
    "ownership": "Public - National Health Service"
}
```

- Target System:

    - Match `ownership` "Public - National Health Service" to the corresponding `name` in the operators of facility records.

Policy for Missing Data

Mandatory Fields

- If a field that is marked as mandatory in the target system is missing in the payload, the facility will not be synchronized.
- An error log entry will be created detailing the missing mandatory field and the facility information.

Malformed Data

- If data is malformed (e.g., invalid format for latitude and longitude), a warning log will be created.
- The integration process will continue despite the malformed data.

Logging Discrepancies

- All discrepancies between the payload and the target system will be logged.
- The logs will capture the nature of the discrepancy, the affected facility, and any relevant details.

Full Automation

- The synchronization process will be fully automated, with no manual flagging for review.
- Logs will be maintained for auditing and troubleshooting purposes.

Logging

Level of Detail

- Every transaction will be logged.
- The logs will include details of successful transactions, errors, and warnings.

Log Storage

- Logs will be stored in the file system.
- As the synchronization will always start from the full facility list in the external system, persistent logs in the database are not required.

Log Details

- **Transaction Logs**: Record each facility synchronization, including the timestamp, facility details, and the result of the synchronization (e.g., added, updated, deleted).
- **Error Logs**: Capture errors such as missing mandatory fields and detail the facility and the missing field.

- **Warning Logs**: Record warnings such as malformed data, including details of the facility and the nature of the malformed data.
- **Discrepancy Logs**: Document any discrepancies between the payload and the target system, detailing the discrepancy, the affected facility, and relevant information.

Example Log Entry

```
{
    "timestamp": "2024-06-12T12:34:56Z",
    "facility_code": "470010",
    "action": "update",
    "result": "success",
    "details": {
        "error": "message"
    }
}
```

Products integration

- The integration of services offered by facilities is outside the current scope. If the product is not stored in the database it will be skipped.
- If a service is not present in the target system, it cannot be created solely based on the name provided in the payload.
- Manual intervention will be required to handle the integration of products.

Special Cases and Exceptions

- **API Failures**:
  - If the API of the external system fails, the synchronization will halt, and an error log will be generated.
  - If the API of the target system fails, the synchronization will halt, and an error log will be generated.
  - If the synchronization task halt next one will be executed nevertheless. System will not require manual reboot.

Technical Synchronization Process

Overview

The synchronization process will be handled by a dedicated microservice. This microservice will schedule tasks to pull data from the third-party system, perform necessary transformations and checks, and then update the target system accordingly.

Steps Involved

1. **Scheduling and Data Pulling**
   - The microservice will schedule tasks to run at predefined intervals.
   - Each task will initiate a data pull from the third-party system using a REST API with basic authentication.

2. **Data Transformation and Validation**
    - The pulled data will undergo transformations to match the target system's format.
    - Validation checks will be performed to ensure data integrity, including:
        - Verifying the presence of mandatory fields.
        - Checking for malformed data (e.g., invalid latitude/longitude formats).
3. **Determining Relevant Data for Update**
    - The transformed data will be compared with the existing data in the target system.
    - SQL queries will be used to identify:
        - New facilities to be added.
        - Existing facilities to be updated.
        - Facilities to be deleted (if not present in the payload).
4. **Triggering Application API**
    - For identified changes, the microservice will trigger the target system's application API to perform the necessary operations (add/update/delete).
    - This approach avoids direct SQL inserts, ensuring that all changes go through the proper channels and other connected processes remain unaffected.

Detailed Process Flow

1. **Task Scheduling**
    - The microservice uses a scheduler (cron job) to run synchronization tasks at regular intervals (e.g., hourly, daily).
2. **Data Pulling**
    - REST API Request:
        - Source: `SIGECA SERVER INSTANCE`
        - Authentication: Basic Authentication (username and password)
3. **Data Transformation**
    - Convert payload data to match the target system's schema.
    - Example transformation:

```
{
    "name": "HEALTH CENTER BELA VISTA",
    "code": "470010",
    "abbreviation": "HC",
    "category": "Health Center",
    ...
}
```

4. **Data Validation**
    - Mandatory field checks:
        - Ensure fields like `name`, `code`, `category` are present.
    - Malformed data checks:
        - Validate latitude and longitude formats.
    - Log errors and warnings as needed.
5. **Relevance Check using SQL Queries**

- Identify new facilities:

```
SELECT * FROM facilities WHERE code NOT IN (SELECT code FROM existing_facilities)
```

- Identify facilities to update:

```
SELECT * FROM facilities WHERE code IN (SELECT code FROM existing_facilities) AND (name != exi
```

- Identify facilities to delete:

```
SELECT * FROM existing_facilities WHERE code NOT IN (SELECT code FROM facilities)
```

6. **API Trigger for Data Changes**
   - Add new facility:

```
POST /api/facilities
{
    "name": "HEALTH CENTER BELA VISTA",
    "code": "470010",
    "abbreviation": "HC",
    "category": "Health Center",
    ...
}
```

   - Update existing facility:

```
PUT /api/facilities/470010
{
    "name": "HEALTH CENTER BELA VISTA",
    "abbreviation": "HC",
    "category": "Health Center",
    ...
}
```

   - Delete facility:

```
DELETE /api/facilities/470010
```

Error Handling

- Errors during the data pull will be logged and flagged for retry.
- Data validation errors will result in logging and the affected data will be skipped.
- API call failures will be logged and may trigger retries or alerts as necessary.