

# Integration of Facilities from SIGECA To eLMIS

## Background

Goal behind the integration 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 facilities.

## External System Payload Example

Payload coming from the SIGECA Central (based on Mapa Sanitario resource):

```
[
  {
    "id": "0003101f-158a-4d51-80c9-8232256ffe63",
    "name": "HEALTH CENTER BELA VISTA",
    "is_deleted": true,
    "code": "470010",
    "acronym": "HC",
    "category": "Health Center",
    "ownership": "Public - National Health Service",
    "management": "Public",
    "municipality": "Ambriz",
    "province": "Bengo",
    "is_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": "GE001",
    "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

Source Field	Target Field	Notes
		geographic zone
units[].province	geographicZone.name	Assumed to be a 2nd level zone for geographic zone
units[].operational	active	Direct mapping
units[].latitude	latitude	Direct mapping
units[].longitude	longitude	Direct mapping
units[].services_offered[].code	supportedPrograms[].code	Direct
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

Source Field	Target Field	Notes
		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
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 except for the supported programs. Supported programs will be added to existing ones, not fully integrated.
3. **Delete Facility:** If a facility present in the target system is not included in the payload, it will be disabled. It will remain in the database but `flag for active` and `enabled` will be set.

## 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` or `province` are provided, but don't exist in the system then new geographic zones will be created as needed.
  - If `municipality` or `province` is not provided then record is skipped.

### 2. Facility Operator

- **Mapping:** Ownership is skipped for the time being.

### 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

- **Mapping:** Programs are assigned based on the `code` fields.
- **Drillwodn Logic**
  - If product with coresponding `code` is not found in the system then it's added in the most basic form.
  - If facility already exists and is updated then new products that are not yet assigned to the facility are added.
  - Products are never removed from facilities based on the sigeca content.

# 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 on both levels.

## 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 synchronization processes, including the type and 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. Each failed facility synchronization should be saved.
- **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.

## Products integration

- The full integration of services beyond their definition is outside the current scope. Mapa Sanitario doesn't provide data to handle requisitions and user assignments.
- It is recommended to configure programs for MS codes in the OpenLMIS ahead of time.
- Manual intervention will be required to handle the integration of products that is beyond simple listing.

## 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).
  - Providing missing referenced entities including: facility types, programs and geo zones.

## 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 (soft delete in form of data updating the facility with enabled=false flag).

## 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 (Python APScheduler) 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) AN
```

- Identify facilities to delete:

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

#### 6. API Trigger for Data Changes

##### Facility

Field	Type	Mandatory
id	String	Generated
code	String	Mandatory
name	String	Mandatory
geographicZone	Object	Mandatory
type	Object	Mandatory
active	Boolean	Default true

Field	Type	Mandatory
enabled	Boolean	Default true
openLmisAccessible	Boolean	Default true
supportedPrograms	Array	Optional
latitude	Numeric	Optional
longitude	Numberic	Optional

- Add new facility:

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

- Update existing facility:

```

PUT /api/facilities/{id}
{
  "id": "{id}",
  "code": "470010", # Mandatory
  "name": "HEALTH CENTER BELA VISTA", # Mandatory
  "geographicZone": { # Mandatory
    "id": "{id}"
  },
  "type": { # Mandatory
    "id": "{id}"
  },
  "active": true, # Default true
  "enabled": true, # Default true
  "openLmisAccessible": true, # Default true
  "supportedPrograms": [ # Optional
    {
      "id": "f7419e56-227b-4008-a6e0-f6cf1b0fb5bd" // New
    }
  ]
}

```

- Delete facility:

```

UPDATE /api/facilities/{id}

```

- Add new facility:

```

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

```

- Update existing facility:

```

PUT /api/facilities/{id}
{
  "id": "{id}",
  "code": "470010",
  "name": "HEALTH CENTER BELA VISTA",
  "geographicZone": {
    "id": "{id}"
  },
  "type": {
    "id": "{id}"
  },
  "active": true,
  "enabled": true,
  "openLmisAccessible": true,
  "supportedPrograms": [
    {
      "id": "{id}",
      "supportActive": true,
      "supportLocallyFulfilled": false,
      "supportStartDate": "2024-07-01"
    },
    {
      "id": "f7419e56-227b-4008-a6e0-f6cf1b0fb5bd" // New
    }
  ]
}

```

- Delete facility:

```

UPDATE /api/facilities/{id}

```

## Facility Type

Field	Type	Mandatory
code	String	Generated From Name
name	String	Mandatory
displayOrder	String	Generated (last available order+1)

- Add Type

```
POST /api/facilityTypes
{
  "code": "hp/hg",
  "name": "HP/HG",
  "displayOrder": "328"
}
```

### Geographical Zone

Field	Type	Mandatory
code	String	Generated From Name
name	String	Mandatory
displayOrder	String	Generated (last available order+1)
level	String	Always Level 3 Ref
parent	String	Either Province ID From Municipality or First available Level 1 Region for Province

- Add Type

```
POST /api/geographicZones
{
  "code": "gz-huambo",
  "name": "Huambo",
  "level": {"id": "1c430656-ea48-427c-bf05-67fb390e2d8f"},
  "parent": "1072ad09-e38e-4067-b17f-429bbc5cafe3"
}
```

### Programs

Field	Type	Mandatory
code	String	Mandatory
name	String	Mandatory
description	Same as Name	

- Add Type



POST /api/geographicZones

```
{  
  "code": "7",  
  "name": "Cardiologia",  
  "description": "Cardiologia"  
}
```