OpenLCA Integration Guide

For the Replit Development Agent

Version: 1.0
Date: 2025-07-18
Author: Replit Coach Too
Status: Draft

1. Objective

This document provides a detailed, step-by-step technical guide for the Replit Agent to implement the integration between our Python/Flask backend and the server-hosted **OpenLCA API**. The goal is to perform a "cradle-to-gate" Life Cycle Assessment (LCA) for a specific beverage product based on user-provided data.

2. Core Concepts & Workflow

The OpenLCA API requires us to build a **Product System** before we can calculate its impact. A Product System is a model of all the interconnected processes and flows that go into making a product. Our backend will construct this system dynamically for each user's SKU.

The overall workflow will be executed by a Celery background worker and will follow these steps:

1. **Data Collection:** Gather all user inputs from our PostgreSQL database.
2. **Flow & Process Mapping:** For each user input (e.g., "1kg of glass"), find the corresponding reference "flow" in the OpenLCA database (e.g., the ecoinvent entry for 'glass production').
3. **Create a Product System:** Create a new, empty product system in OpenLCA named after the user's SKU.
4. **Model the Life Cycle:** Add and connect all the necessary processes to the product system in a logical sequence.
5. **Calculate & Retrieve Results:** Run the LCA calculation on the completed product system and retrieve the results for Carbon, Water, and Waste.

3. Detailed Implementation Steps

The following instructions detail the logic for the Celery worker that will be triggered by a POST /api/reports/generate request.

Prerequisites

- The OpenLCA server URL and API credentials will be stored in Replit Secrets.

- The backend will use a Python HTTP client library (e.g., requests) to communicate with the OpenLCA JSON-RPC API.
- We will primarily use the **ecoinvent** database, which must be available on the OpenLCA server.

Step 1: Data Retrieval

- Fetch all product_inputs from our database for the given product_id.
- Organize these inputs into logical categories based on the input_type field, corresponding to the user's request:
  1. agricultural_inputs
  2. transport_inputs
  3. processing_inputs (Distillery/Winery/Brewery)
  4. primary_packaging_inputs (Bottle/Can)
  5. label_inputs
  6. stopper_inputs
  7. secondary_packaging_inputs

Step 2: Create the Product System

- **API Call:** Use the olca.createProductSystem(system) endpoint.
- **Logic:**
  - Create a ProductSystem object in our Python code.
  - Name it uniquely based on the product SKU and a timestamp (e.g., "Avallen Calvados - 2025-07-18").
  - Define the **Reference Process** and **Target Amount**. For a beverage, this will be the final bottling/canning process, with a target amount of **1 unit** (e.g., one 0.7L bottle).
  - This call will return a unique @id for the new product system. We must store this ID for all subsequent steps.

Step 3: Model the Life Cycle Stages

For each stage, we will find the appropriate reference processes in OpenLCA and add them to our product system.

3.1 Agricultural Inputs

- **User Data:** A list of ingredients and their weights (e.g., 1.5 kg of apples).
- **Logic:**
  1. For each ingredient, search the OpenLCA database for a matching elementary flow. Use a keyword search (e.g., "apple production, at farm").
  2. Create a new **Process** within our product system for each ingredient.
  3. Set the quantitative reference for each process to match the user's input

(e.g., 1.5 kg).
4. Link the output of these agricultural processes as inputs to the main "Processing" step.

3.2 Transportation of Inputs

- **User Data:** Average distance and mode of transport.
- **Logic:**
  1. Search OpenLCA for a transport process that matches the mode (e.g., "transport, freight, lorry >32 metric ton").
  2. Create a new **Process** for transportation.
  3. The quantitative reference for a transport process is typically in **tonne-kilometers (tkm)**. We will calculate this by converting the total weight of agricultural inputs to tonnes and multiplying by the distance in km.
  4. Link the output of this process to the main "Processing" step.

3.3 Processing (Distillery/Winery/Brewery)

- **User Data:** Energy consumption (kWh), water usage (litres), and waste output (kg) for the facility, allocated per product unit. Also, for aged spirits, the "Angel's Share" (evaporation loss percentage).
- **Logic:**
  1. Create a central **Process** for "Distillation" or "Winemaking". This will be the reference process for our entire system.
  2. Add **Inputs** to this process:
     - The outputs from the Agricultural and Transportation steps.
     - Energy: Find the flow for 'electricity, medium voltage' for the correct region (e.g., Great Britain) and add it as an input with the specified kWh.
     - Water: Find the flow for 'water, mains' and add it as an input with the specified litres.
  3. Add **Outputs** from this process:
     - The primary output is the finished liquid (e.g., "1 Litre of Spirit").
     - Waste: Find the flow for 'municipal solid waste' and add it as an output with the specified kg.
     - **Angel's Share:** Model this as an emission to air. Find the flow for 'ethanol, air' and calculate the amount based on the loss percentage during aging.

3.4 Packaging (Bottle, Label, Stopper, Secondary)

- **User Data:** Weight and material type for each packaging component (e.g., 0.5 kg of Glass, 0.01 kg of Paper, 0.02 kg of Cork).
- **Logic:**
  1. This stage will be modeled as a final "Bottling & Packaging" process.

2. For each packaging component, search OpenLCA for the corresponding material production flow (e.g., "glass production, container", "paper production, kraft", "cork production").
3. Add each of these flows as an **Input** to the "Bottling & Packaging" process, with the quantity specified by the user.
4. The finished, packaged product is the final output of this process and the entire system.

Step 4: Calculate and Retrieve Results

- **API Call:** Use olca.calculate(setup) where the setup object contains the @id of our product system, the desired **Impact Assessment Method** (e.g., IPCC 2013 GWP 100a for carbon), and the target amount (1 unit).
- **Logic:**
  1. First, run the calculation for **Global Warming Potential (GWP 100a)** to get the Carbon Footprint (kg CO2-eq).
  2. Next, run a separate calculation using a **Water Footprint** impact method to get water usage (litres).
  3. Finally, retrieve waste outputs directly from the process inventory.
  4. The API call will return a SimpleResult object containing the total impact.
- **Data Storage:**
  - Parse the results from the SimpleResult object.
  - Store the total values for carbon, water, and waste in the reports.report_data_json field in our PostgreSQL database.
  - Update the report status to draft.
  - Notify the user that their report is ready.