**Image Processing System – Low-Level Design Document**

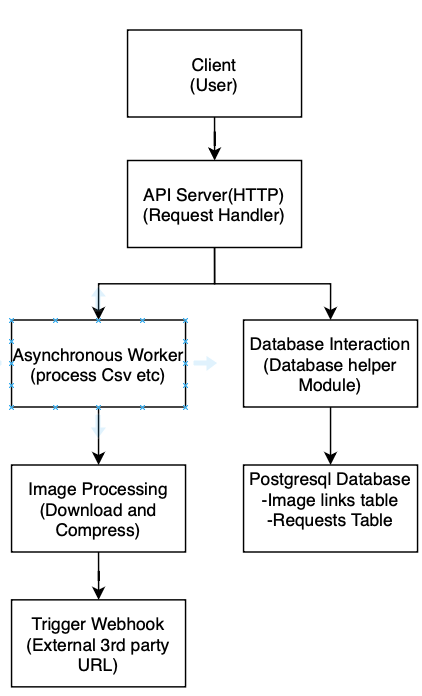
**1. Overview**

The Image Processing System is designed to process CSV uploads that contain product information and image URLs. Upon receiving a CSV file, the system validates the file, downloads and compresses the images asynchronously, stores a processed CSV, and inserts processed data into a PostgreSQL database. It also triggers an external API through webhooks when request is completed.Clients can query the processing status via a status API. The system is built using Python, asyncio for concurrency, and PostgreSQL for persistent storage.

## 2. System Architecture & Components

### 2.1 High-Level Architecture Diagram

Below is a simplified diagram (conceptual view) of the system architecture:



### 2.2 Component Descriptions

* **API Server (HTTP Endpoints):**
  + **Upload API (/upload):** Accepts multipart/form-data CSV file uploads, performs preliminary CSV validation, and returns a unique request ID along with processing results (success/failure counts).
  + **Status API (/status/{request\_id}):** Allows clients to check the processing status of a request using the unique request ID.
* **Asynchronous Worker (Image Processing Service):**
  + Uses asynchronous tasks (via asyncio) to download and compress images concurrently.
  + Reads the CSV file, validates required fields, downloads images from URLs using the requests module, compresses images with Pillow (PIL), and saves the processed image URLs.
  + Creates a processed CSV file stored in a designated static directory.
  + Inserts each processed row into the public.image\_links table in the database.
* **Database Interaction:**
  + **DatabaseHelper Module:** Centralizes all database operations (e.g., saving request, updating status, inserting image links, deleting a request).
  + Utilizes psycopg2 for PostgreSQL connectivity.
  + Two main tables:
    - requests: Tracks each processing request and its status.
    - public.image\_links: Stores product details and associated image URLs after processing.
* **Webhook(External API):**
  + **Trigger\_Webhook:**  Sends a POST request with processing results to an external webhook receiver.

## 3. Database Schema

### 3.1 requests Table

* **Columns:**
  + id (text, primary key): Unique request identifier.
  + status (text): Current processing status (e.g., “processing”, “completed”).

### 3.2 public.image\_links Table

* **Columns:**
  + id (serial, primary key)
  + request\_id (text): References the requests table (with cascade deletion).
  + serial\_number (integer): Product serial number.
  + product\_name (text): Name of the product.
  + input\_image\_urls (text): Original image URLs from the CSV.
  + output\_image\_urls (text): Processed/compressed image URLs.

## 4. API Documentation

### 4.1 Upload API

* **Endpoint:** POST /upload
* **Content-Type:** multipart/form-data
* **Description:**  
  Accepts a CSV file upload containing columns "Serial Number", "Product Name", and "Input Image Urls". Validates the CSV, returns an error if invalid, or proceeds to process the file asynchronously.Triggers an external webhook on ccomnpletion.
* **Response:**
  + **200 OK:** All rows processed successfully.
  + **206 Partial Content:** Some rows processed successfully while others failed.
  + **400 Bad Request:** CSV validation failed or no successful rows.
* **Response Body Example:**

{

"request\_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",

"success\_count": 10,

"failure\_count": 2

}

* **Note: In case of all invalid image urls, it will show bad request but the request will be processed since the data is not invalid.**

### 4.2 Status API

* **Endpoint:** GET /status/{request\_id}
* **Description:**  
  Retrieves the current status of a processing request.
* **Response:**
  + **200 OK:** Returns the status of the request.
* **Response Body Example:**

{

"status": "completed"

}

Can show status as completed, processing or not found depending on status id.

**5. Asynchronous Workers Documentation**

* **Function:** process\_csv(csv\_content, request\_id, validate\_only=False)
  + **Purpose:**  
    Validates CSV content, downloads images, compresses images asynchronously, writes a processed CSV file, and inserts each valid row into the database.
  + **Workflow:**
    1. **Validation:** Reads CSV rows and checks for required fields.
    2. **Task Scheduling:** For valid rows, schedules asynchronous image download and compression tasks.
    3. **CSV Output:** Writes the processed data (including processed image URLs) to a new CSV file.
    4. **Database Insertion:** Inserts processed row data into public.image\_links using insert\_image\_link.
    5. **Status Update:** Once complete, updates the request status to "completed".
    6. **Webhook:** Triggers an external webhook on completion of request.
  + **Error Handling:**  
    Catches exceptions during processing, logs errors, and if needed, cleans up any temporary files.
* **Function:** download\_and\_compress(url, request\_id, index)
  + **Purpose:**  
    Downloads the image from the provided URL, compresses it using JPEG quality settings, and stores it in a request-specific folder.
  + **Returns:**  
    The relative URL path if successful or "error" if any issue occurs.

**6. GitHub Repository**

**Link:** [**https://github.com/kanishk-ramrakhiyani/Image-Processing**](https://github.com/kanishk-ramrakhiyani/Image-Processing)

**7. Postman Collection**

**Link:** [**https://www.postman.com/maintenance-engineer-22657487/workspace/image-processing/collection/36634943-9099507d-1f6e-4006-a43d-facd50f55b27?action=share&creator=36634943**](https://www.postman.com/maintenance-engineer-22657487/workspace/image-processing/collection/36634943-9099507d-1f6e-4006-a43d-facd50f55b27?action=share&creator=36634943)