1. **Low-Level Design(LLD)**

This document provides a detailed low-level design for an asynchronous image processing system that:

1. Accepts a CSV file of image URLs.
2. Processes these images (e.g., resize and compress).
3. Stores both input and output image URLs in a database.
4. Provides an API to check the processing status.
5. Optionally triggers a webhook upon completion.
6. **System Architecture**

The system is composed of the following components:

* **API Layer**
  + **Upload API**: Accepts a CSV file and returns a unique request ID.
  + **Status API**: Allows users to query the processing status using the request ID.
* **Asynchronous Processing Layer**
  + **Task Queue**: Handles asynchronous image processing tasks.
* **Data Storage Layer**
  + **SQLite Database**: Stores the request information, image paths, and processing statuses.
* **External Components**
  + **Webhook**: External URL that receives a notification after processing is complete.
* **Output**
  + **Output CSV**: Generated CSV file containing input and output image URLs.

**3. System Components and Their Roles**

**3.1 API Layer**

* **Upload API**
  + **Role**: Accepts a CSV file, validates it, and initiates the processing workflow. Returns a unique request ID.
  + **Functionality**:
    1. Generates a unique request\_id.
    2. Stores the file information and initial status in the database.
    3. Initiates asynchronous processing by adding tasks to the task queue.
    4. Returns the request\_id to the user.
* **Status API**
  + **Role**: Allows the user to query the current status of image processing using the request\_id.
  + **Functionality**:
    1. Queries the database for the status of the request.
    2. Returns the status to the user.

**3.2 Asynchronous Processing Layer**

* **Task Queue**
  + **Role**: Manages and executes the image processing tasks asynchronously.
  + **Functionality**:
    1. Asynchronously processes each row in the CSV file.
    2. Downloads, resizes, and compresses images.
    3. Stores the processed image paths in the database.
    4. Updates the status of each image and the overall request in the database.
    5. Generates the output CSV upon completion and triggers the webhook if provided.

**3.3 Data Storage Layer**

* **SQLite Database**
  + **Role**: Stores all persistent data, including request statuses, input/output image URLs, and processing logs.
  + **Tables**:
    - requests: Stores metadata about each CSV upload and its processing status.

request\_id, file\_name, and status.

* + - images: Stores details about each processed image.

id, request\_id, serial\_number, product\_name, input\_image\_urls, output\_image\_urls, and status.

**3.4 External Components**

* **Webhook**
  + **Role**: Receives a notification once image processing is complete.
  + **Functionality**:
    1. Accepts the output CSV and request\_id via a POST request.
    2. Can be used to integrate with other systems for further processing or notification.

**3.5 Output**

* **Output CSV**
  + **Role**: Contains a summary of the image processing task, mapping input image URLs to output image URLs.
  + **Functionality**:
    1. Generated automatically once all images have been processed.
    2. Delivered via the webhook or can be manually retrieved.

**4. System Workflow**

**4.1 Upload and Processing Workflow**

1. **Upload CSV**:
   * User uploads a CSV file through the Upload API.
   * The system generates a request\_id and initiates asynchronous processing.
2. **Processing**:
   * The system asynchronously processes each row in the CSV:
     + Validates the row data.
     + Downloads and processes each image URL.
     + Saves the processed image and updates the database.
3. **Completion**:
   * Upon completion, the system updates the request status to completed.
   * An output CSV is generated.
   * Optionally, a webhook is triggered.

**4.2 Status Check Workflow**

1. **Check Status**:
   * User checks the processing status using the Status API by providing the request\_id.
   * The system returns the current status (processing, completed).

**5. Database Schema Design**

**5.1 requests Table**

* **Columns**:
  + request\_id: Unique identifier for the request (Primary Key).
  + file\_name: Name of the uploaded CSV file.
  + status: Current processing status (processing, completed).

**5.2 images Table**

* **Columns**:
  + id: Auto-incremented ID (Primary Key).
  + request\_id: Foreign key linking to the requests table.
  + serial\_number: Serial number from the CSV.
  + product\_name: Product name from the CSV.
  + input\_image\_urls: Original image URLs from the CSV.
  + output\_image\_urls: Processed image URLs generated by the system.
  + status: Status of image processing (processed, failed).

**6. Component Interaction**

* **Upload API**:
  + Receives CSV, generates a request\_id, stores initial data in the SQLite database, and starts the processing tasks.
* **Asynchronous Processor**:
  + Handles image downloading, processing (resize/compress), saving processed images, and updating the database with results.
* **Status API**:
  + Queries the SQLite database for the current status of a request and returns it to the user.
* **Webhook**:
  + Triggered by the processor after completion, sending the output CSV and request\_id to a predefined URL.

### ****Explaination of Schema****

1. **requests Table**:
   * **request\_id**: A unique identifier for each processing request, typically generated using UUID.
   * **file\_name**: The name of the CSV file associated with the request.
   * **status**: Tracks the current status of the request, such as "processing", "completed", or "failed".
   * **created\_at**: Automatically records the timestamp when the request is created.
   * **updated\_at**: Automatically records the timestamp when the request's status is last updated. It is managed via a trigger.
2. **images Table**:
   * **id**: A unique identifier for each image processing record, automatically incremented.
   * **request\_id**: Links each image record to the corresponding request in the requests table.
   * **serial\_number**: Reflects the serial number from the CSV file, ensuring the images can be matched to their original row.
   * **product\_name**: The product name associated with the image.
   * **input\_image\_urls**: A list of URLs from which the images are downloaded for processing.
   * **output\_image\_urls**: A list of URLs where the processed images are stored.
   * **status**: Tracks the status of the image processing, with a default value of "pending".
   * **created\_at** and **updated\_at**: Automatically managed timestamps for when the image record is created and last updated, respectively. The updated\_at is managed via a trigger.
3. **Indexes**:
   * An index is created on request\_id in the images table to speed up queries that filter by request\_id.
4. **Triggers**:
   * **update\_request\_timestamp**: Automatically updates the updated\_at field in the requests table whenever the status or other fields are updated.
   * **update\_image\_timestamp**: Similarly updates the updated\_at field in the images table on updates.

This schema is designed to efficiently handle and track the status of image processing requests, allowing for quick lookups and ensuring data consistency through the use of foreign keys and triggers.

System architecture

