Low-Level Design (LLD) Document

1. System Overview

The system is designed to process image data from CSV files asynchronously. It accepts a CSV file containing product information and image URLs, validates the data, processes the images (compresses them by 50%), stores the processed images and product information in a database, and provides APIs for users to check the processing status.

2. Components and Their Roles

1. Upload API

- **Role**: Accepts a CSV file from the user, validates the file format, and initiates the image processing workflow.
- **Function**:
    - Validate the CSV file (check for required columns and valid URLs).
    - Generate a unique request_id for the processing request.
    - Save the request details in the database with a status of "Pending".
    - Trigger the asynchronous image processing task.

2. Image Processing Service

- **Role**: Processes images asynchronously by downloading them, compressing them by 50%, and uploading the compressed images to storage.
- **Function**:
    - Download images from the input URLs.
    - Compress images using a library like Pillow (Python).
    - Upload the compressed images to a storage service.
    - Update the database with the output image URLs and change the request status to "Completed".

3. Status API

- **Role**: Allows users to check the processing status of a request using the request_id.
- **Function**:
    - Query the database for the request status.
    - Return the status (Pending, Completed, or Failed) to the user.

4. Database

- **Role**: Stores product data, input image URLs, output image URLs, and processing status.
- **Function**:
    - Track the status of each processing request.
    - Store product information and image URLs.
    - Provide data for the Status API.

5. Celery (Task Queue)

- **Role**: Handles asynchronous task processing.
- **Function**:
    - Queue image processing tasks.
    - Distribute tasks to worker nodes for parallel processing.
    - Retry failed tasks.

3. Workflow

1. **User Uploads CSV**:
    - The user sends a CSV file to the /upload API.
    - The API validates the CSV and generates a request_id.
    - The request details are saved in the database with a status of "Pending".
2. **Image Processing**:
    - The process_images Celery task is triggered.
    - The task downloads images, compresses them, and uploads the compressed images to storage.
    - The database is updated with the output image URLs, and the request status is changed to "Completed".
3. **Check Status**:
    - The user queries the /status API with the request_id.
    - The API returns the current status of the request.
4. **Webhook Notification (Bonus)**:
    - After processing all images, the system triggers a webhook to notify the user.

4. Database Schema

Tables:

1. **Requests**:
    - id (Primary Key): Unique ID for each request.
    - request_id: Unique request ID generated for each CSV upload.
    - status: Status of the request (Pending, Completed, Failed).
    - created_at: Timestamp when the request was created.
    - updated_at: Timestamp when the request was last updated.

Column	Type	Description
id	Primary Key	Unique ID
request_id	String	Unique request identifier
status	Enum	`Pending`, `Completed`, `Failed`
created_at	Timestamp	Request creation time
updated_at	Timestamp	Last update time

2. **Products**:
    - id (Primary Key): Unique ID for each product.
    - serial_number: Serial number of the product.
    - product_name: Name of the product.
    - input_image_urls: Comma-separated input image URLs.
    - output_image_urls: Comma-separated output image URLs.
    - request_id (Foreign Key): Links to the Requests table.

Column	Type	Description
id	Primary Key	Unique ID
serial_number	String	Product serial number
product_name	String	Product name
input_image_urls	String	Comma-separated image URLs
output_image_urls	String	Comma-separated compressed image URLs
request_id	Foreign Key	Links to Requests table

6. API Specifications

1. Upload API

- **Endpoint**: POST /upload
- **Request Body**: CSV file.
- **Response**:

```
{
  "request_id": "unique-request-id",
  "status": "Pending"
}
```

2. Status API

- **Endpoint**: GET /status?request_id=<request_id>
- **Response**:

```
{
 "request_id": "unique-request-id",
 "status": "Completed",
 "output_csv_url": "https://output-csv-url.com"
}
```

6. Error Handling

- **Invalid CSV**: Return a 400 Bad Request with an error message.
- **Invalid Request ID**: Return a 404 Not Found if the request_id is not found.
- **Image Download Failure**: Retry the download or mark the task as failed.