## Low Level Design

The system is designed to process image data from CSV files asynchronously. It involves accepting CSV files, validating their format, processing images by reducing their quality, and storing processed data. The system also provides APIs to upload CSV files, check the status of processing requests, and handle webhook callbacks.

### System Components and Their Roles

#### ****1. Image Processing Service Interaction****

* **Role:** This component interacts with an external asynchronous image processing service that compresses images by reducing their quality by 50%.
* **Functionality:**
  + Sends image URLs to the processing service.
  + Receives callbacks from the service once the images are processed.
  + Stores the output URLs of the processed images.

#### ****2. Webhook Handling****

* **Role:** Processes callbacks from the image processing service when images are processed.
* **Functionality:**
  + Listens for HTTP requests from the image processing service.
  + Updates the status of the image processing request in the database.
  + Stores output image URLs in the database.

#### ****3. Database Interaction****

* **Role:** Stores product data and tracks the status of each processing request.
* **Functionality:**
  + Stores initial product and image data from the CSV file.
  + Updates processing status and stores output image URLs.
  + Provides data for status checks via API.

#### ****4. API Endpoints****

* **Upload API:**
  + **Functionality:** Accepts CSV files, validates their format, and returns a unique request ID to the user.
  + **Error Handling:** Handles file validation errors and improper formatting issues.
* **Status API:**
  + **Functionality:** Checks the processing status of the images using the request ID.
  + **Response:** Provides current status (e.g., pending, processing, completed) and URLs of processed images if available.

## Visual Diagram of the System

The system consists of several components that interact as shown below



· **CSV Upload**: User uploads a CSV file through the Upload API.

· **Validation Service**: Validates the CSV format.

· **Asynchronous Worker**: Reads the CSV, sends image URLs to the Image Processing Service.

· **Image Processing Service**: Processes images and sends results via a webhook callback.

· **Webhook Handler**: Updates the database with the output image URLs.

· **Database**: Stores the processing request, input, and output image URLs.

· **Status API**: Retrieves and returns the status of image processing requests.

## Database Schema

Here's a proposed database schema to track product data and processing statuses:

### requests Table

**Columns**:

id (Primary Key): Unique request ID.

status: Status of the processing (pending, processing, completed).

### products Table

**Columns:**

id (Primary Key): Auto-incremented product ID.

request\_id: Foreign key to the requests table.

product\_name: Name of the product.

input\_image\_urls: Comma-separated input image URLs.

output\_image\_urls: Comma-separated output image URLs.

## API Documentation

#### ****1. Upload API****

* **Endpoint:** /upload
* **Method:** POST
* **Request Body:** CSV file (Multipart form data).
* **Response:** { "request\_id": "<unique\_id>" }
* **Error Responses:**
  + 400 - Invalid CSV format.
  + 500 - Internal server error.

#### ****2. Status API****

* **Endpoint:** /status/<request\_id>
* **Method:** GET
* **Response:** { "status": "completed", "output\_image\_urls": [<URLs>] }
* **Error Responses:**
  + 404 - Request ID not found.
  + 500 - Internal server error.

### 6. Asynchronous Workers Documentation

Asynchronous workers are responsible for:

1. **Reading CSV files**: Extracting product names and image URLs.
2. **Interacting with the Image Processing Service**: Sending requests for image compression.
3. **Error Handling**: Managing retries or failures during image processing.
4. **Updating Database**: Upon receiving webhook callbacks, the worker updates the processed image URLs.