Technical Design Document

1. Introduction

This document describes the technical design for a system to efficiently process image data from CSV files. The system includes components for uploading CSV files, asynchronously processing images, storing data, and providing status updates.

2. System Overview

The system processes images listed in a CSV file, compresses them by 50%, and provides status updates through an API. The components of the system include:

1. **Flask Application**: Manages API requests and interacts with the database and Celery.
2. **Celery Worker**: Handles asynchronous image processing tasks.
3. **Redis**: Serves as the message broker for Celery.
4. **Database**: Stores product data and processing status.
5. **Webhook (Future Enhancement)**: Will notify an external system upon completion of processing.

3. Components and Their Functions

3.1 Flask Application

Description: The Flask application serves as the main entry point for the system, handling HTTP requests for uploading CSV files and checking processing status.

Key Functions:

- **/upload Endpoint:**
    - **Role:** Receives and validates CSV files containing image data.
    - **Functionality:**
        - Accepts CSV file uploads.
        - Validates the file format and data.

        - Generates a unique request ID.
        - Adds a task to Celery to process the images.
        - Responds with the unique request ID.
- **/status Endpoint:**
    - **Role:** Provides the status of image processing for a given request ID.
    - **Functionality:**
        - Retrieves processing status and output URLs from the database.
        - Responds with status and output image URLs.

Configuration:

- **Celery Integration:** Uses Celery to handle asynchronous image processing tasks.
- **Database Integration:** Interacts with the database to store and retrieve product data and processing status.

3.2 Celery Worker

Description: The Celery worker performs background image processing tasks asynchronously, allowing the Flask application to remain responsive.

Key Functions:

- **process_images Task:**
    - **Role:** Processes images listed in the CSV file.
    - **Functionality:**
        - Fetches images from URLs.
        - Resizes each image by 50%.
        - Saves processed images with a new filename.
        - Updates the database with output URLs and processing status.

Configuration:

- **Broker:** Uses Redis for message brokering.
- **Backend:** Configured with Redis for storing task results.

3.3 Redis

Description: Redis serves as the message broker for Celery, managing the communication between Flask and Celery workers.

Key Functions:

- **Message Brokering:** Routes messages between the Flask application and Celery workers.
- **Task Queuing:** Handles queuing and scheduling of background tasks.

Configuration:

- **Host:** 127.0.0.1
- **Port:** 6379

3.4 Database

Description: The database stores product information, including input and output image URLs and processing status.

Key Functions:

- **Storing Data:** Keeps records of products, input and output image URLs, and processing status.
- **Retrieving Data:** Provides data for the status endpoint.

Schema:

- **Product Table:**
    - **Fields:**
        - serial_number: Integer
        - product_name: String

        - input_image_urls: String (Comma-separated URLs)
        - output_image_urls: String (Comma-separated URLs)
        - status: String (Processing status)

Configuration:

- **Type:** SQLite
- **URI:** sqlite:///images.db

3.5 Webhook (Future Enhancement)

Description: The webhook component will notify an external system upon the completion of image processing.

Key Functions:

- **Notification:** Sends a notification to a predefined endpoint with processing results.

Configuration:

- **Endpoint:** To be defined
- **Payload:** To include processing results and status

4. Data Flow

1. **CSV File Upload:**
    - User uploads a CSV file through the /upload endpoint.
    - The Flask application validates the file and creates a unique request ID.
    - Celery schedules the process_images task with the request ID.
2. **Image Processing:**
    - Celery worker retrieves the task, processes the images (resizing and saving), and updates the database with output URLs and processing status.
3. **Status Check:**

    - User queries the status of processing through the /status endpoint using the request ID.
    - The Flask application retrieves status and output URLs from the database and responds to the user.

5. Error Handling

- **Invalid CSV Format:** Return a 400 Bad Request response with an error message.
- **Processing Failures:** Log errors and update the database with a failure status.
- **Database Errors:** Handle errors gracefully and return appropriate error messages to the user.

6. Future Enhancements

- **Webhook Integration:** Implement a webhook to notify external systems upon processing completion.
- **Additional Image Processing Features:** Add support for other image manipulations (e.g., cropping, filtering).