Low-Level Design (LLD)

Introduction

The Image Processing Pipeline is designed to process images asynchronously based on URLs provided in a CSV file. The system is built to handle large volumes of image processing tasks, ensuring scalability and efficiency. This document outlines the technical details of each component, including their roles, interactions, and how they contribute to the overall system.

System Components

1. API Layer

The API layer serves as the entry point for the system. It exposes endpoints for uploading CSV files and checking the processing status of the images.

Upload API

- **Function:**
  - Accepts a CSV file from the user containing product names and corresponding image URLs.
  - Validates the CSV format to ensure it adheres to the expected structure.
  - Generates a unique `requestId` for each file upload.
  - Stores initial data in MongoDB, including the `requestId`, product name, input image URLs, and status (initially set to "Processing").
  - Enqueues image processing tasks in Redis using Bull.
- **Endpoint:**
  - `POST /api/upload`
- **Inputs:**
  - CSV file (multipart/form-data)
- **Outputs:**
  - JSON response containing the `requestId` and a message confirming successful upload and processing initiation.

Status API

- **Function:**
  - Allows users to query the processing status of their image processing request using the `requestId`.
  - Returns the current status (e.g., "Processing" or "Processed") and the URLs of the processed images.

- **Endpoint:**
  - `GET /api/status/:requestId`
- **Inputs:**
  - `requestId` (URL parameter)
- **Outputs:**
  - JSON response containing the status and processed image URLs.

2. Database Interaction

The system uses MongoDB to store and track the status of each image processing request.

MongoDB

- **Schema:**
  - **Collection:** `requests`
  - **Fields:**
    - `requestId`: A unique identifier for each image processing request.
    - `productName`: The name of the product associated with the images.
    - `inputImageUrls`: An array of original image URLs provided in the CSV.
    - `outputImageUrls`: An array of processed image URLs.
    - `status`: The current status of the request, either "Processing" or "Processed".

3. Image Processing Service Interaction

Image processing is handled asynchronously using a combination of Bull, Redis, and Sharp.

Bull Queue

- **Function:**
  - Manages the queue of image processing tasks.
  - Each task consists of an image URL and the associated `requestId`.
  - Tasks are processed by a worker process running separately from the main application.

Redis

- **Function:**
  - Acts as the backend for Bull, storing the queue and managing the state of each job.
  - Ensures that tasks are processed in the order they were enqueued and handles job retries and failures.

Worker Process (Image Processor)

- **Function:**
    - Listens to the Bull queue for new image processing tasks.
    - For each task, download the image from the provided URL.
    - Processes the image using the Sharp library (e.g., compresses it to over 50% of its original quality).
    - Saves the processed image to the local file system.
    - Updates the corresponding MongoDB document with the processed image URL.
    - Checks if all images for the request have been processed, and if so, updates the status to "Processed".

4. Error Handling

- **CSV Validation:**
    - The system validates the uploaded CSV file to ensure it has the correct columns and data format before processing begins.
    - If the CSV is invalid, the upload API responds with an appropriate error message and does not proceed with processing.
- **Image Processing Errors:**
    - The worker logs any errors that occur during image processing (e.g., failed image download, Sharp processing error).
    - Failed jobs are retried a certain number of times (configurable in Bull).
    - If processing ultimately fails, the status in MongoDB remains "Processing" to indicate an incomplete request.

System Workflow

Step-by-Step Workflow

1. **CSV Upload:**
    - A user uploads a CSV file through the Upload API.
    - The server validates the CSV file, generates a `requestId`, stores the initial data in MongoDB, and enqueues tasks in Redis via Bull.
2. **Image Processing:**
    - The worker process fetches tasks from the Redis queue.
    - It downloads each image, processes it with Sharp, saves the result, and updates MongoDB with the processed image URL.

    - The worker checks if all images associated with a `requestId` have been processed and updates the status to "Processed" if complete.
3. **Status Checking:**
    - The user can query the Status API using their `requestId`.
    - The API returns the current status and, if processing is complete, the URLs of the processed images.

Conclusion

This Low-Level Design (LLD) document provides a detailed technical overview of the Image Processing Pipeline. The design ensures that the system can handle large volumes of image processing tasks efficiently and asynchronously while tracking the progress and storing the results in a structured and retrievable manner. The system's components interact seamlessly to provide a robust solution for image processing from CSV files.