# **Technical Design Document: Image Processing System**

## **1. Introduction**

This document outlines the technical design for an image processing system that receives a CSV file with product information, processes images listed in the CSV, and generates an output CSV file with processed image details. The system uses Flask as a web framework, SQLite for database management, and Python for backend processing.

## **2. System Overview**

The system comprises:

* A Flask-based web application to handle file uploads and status checks.
* A SQLite database to store product and request data.
* Background processing of images with compression and URL handling.
* Generation of an output CSV file with the results.

## **3. Architecture**

### **3.1 Components**

1. **Web Application (Flask)**
   * Handles HTTP requests for file uploads and status checks.
   * Provides endpoints for uploading CSV files and checking processing status.
2. **Database (SQLite)**
   * Stores product information and processing requests.
   * Tables: products and requests.
3. **Image Processing**
   * Downloads images from provided URLs.
   * Compresses images and saves them to a local directory.
4. **CSV Handling**
   * Reads input CSV files.
   * Generates output CSV files after processing.

### **3.2 Data Flow**

1. **File Upload**
   * User uploads a CSV file via the /upload endpoint.
   * The file is saved and read into a DataFrame.
2. **Database Operations**
   * Products and requests are inserted into the SQLite database.
   * Each product is associated with a unique request ID.
3. **Image Processing**
   * Images are downloaded and compressed in the background.
   * Output image URLs are updated in the database.
4. **CSV Generation**
   * After processing all images, an output CSV file is generated with results.

## **4. Database Schema**

### **4.1 products Table**

| **Field** | **Type** | **Constraints** |
| --- | --- | --- |
| id | INTEGER | PRIMARY KEY, AUTOINCREMENT |
| serial\_number | VARCHAR(255) | UNIQUE, NOT NULL |
| product\_name | VARCHAR(255) | NOT NULL |
| input\_image\_urls | TEXT | NOT NULL |
| output\_image\_urls | TEXT |  |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

### **4.2 requests Table**

| **Field** | **Type** | **Constraints** |
| --- | --- | --- |
| id | INTEGER | PRIMARY KEY, AUTOINCREMENT |
| request\_id | VARCHAR(36) | UNIQUE, NOT NULL |
| status | VARCHAR(20) | NOT NULL |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |
| updated\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |
| product\_id | INTEGER | FOREIGN KEY (products.id) |

## **5. Endpoints**

### **5.1 POST /upload**

* **Description**: Uploads a CSV file and processes the data.
* **Request**: Multipart form-data with a file field.
* **Response**: JSON object with request\_id.
* **Error Handling**: Returns error messages for missing files, invalid CSV formats, or database integrity issues.

### **5.2 GET /status/<request\_id>**

* **Description**: Checks the processing status of a given request ID.
* **Response**: JSON object with the current status (completed or processing), or an error message if the request ID is not found.

## **6. Image Processing**

### **6.1 Compression**

* **Function**: compress\_image(input\_url, output\_path, quality=50)
* **Parameters**: input\_url (URL of the image), output\_path (path to save the compressed image), quality (JPEG quality level).
* **Process**: Downloads image, converts to RGB if necessary, compresses and saves as JPEG.

### **6.2 Processing**

* **Function**: process\_images(request\_id, serial\_number, product\_name, input\_image\_urls)
* **Process**:
  1. Retrieves product data based on request\_id.
  2. Processes each image URL in the input\_image\_urls.
  3. Updates output\_image\_urls in the database.
  4. Writes a single CSV file containing all processed results.

## **7. Concurrency**

* **Threading**: Uses Python's threading module to process images in parallel.
* **Concurrency Control**: Each thread handles a separate request, ensuring no overlap or data corruption.

## **8. Error Handling**

* **Database Errors**: Caught and reported via HTTP response codes (e.g., 400 for bad requests, 500 for server errors).
* **File Handling Errors**: Ensured via validation checks before processing.
* **Thread Safety**: Local database connections are used in threads to avoid concurrency issues.

## **9. Deployment**

* **Configuration**: Ensure SQLite database is correctly set.

## **10. System Diagram**

### **Diagram Explanation**

* **Web Client** interacts with **Flask Server** via HTTP requests.
* **Flask Server** processes the requests, interacts with the **SQLite DB**, and starts the **Image Processing**.
* **Image Processing** operates in a separate thread, compressing images and updating the **SQLite DB**.
* After processing, **Output CSV Generation** creates a CSV file with the results.

**11. Publicly Accessible Link**

https://api.postman.com/collections/25062895-89010ca8-5d9e-466c-80a9-f510640f1ddf?access\_key=PMAT-01J6KQP8BH6Z1126010BZXDZBD