# Low-Level Design (LLD) - Image Processing System

## Overview

This system is designed to handle asynchronous image processing requests. The system enables users to upload CSV files containing product information and image URLs. It returns a unique request ID that can be used to check the processing status and eventually download a CSV file containing processed image URLs.

## Tech Stack Used

1. **Python**: Used for backend development with asynchronous handling via Quart.
2. **MongoDB**: NoSQL database for storing and tracking request statuses and image URLs.
3. **AWS S3**: Storage for compressed images, accessible via public URLs.

## Components Overview:

**1. Image Processing Service Interaction**

* The system uses a single **Quart** server to manage asynchronous requests efficiently, improving user experience.
* The image processing (e.g., compression) is handled **within the Quart server itself**. It fetches the images from the URLs provided in the CSV, processes them, and uploads the compressed images to an S3 bucket.
* Asynchronous request handling ensures that large or long-running tasks do not block user interaction while processing occurs in the background.

**2. Database Interaction (MongoDB)**

* A **MongoDB** database is used to store and track the status of each image processing request.
* The database stores:
  + Product data (product names, input image URLs)
  + Output image URLs after processing
  + Timestamps marking when requests were created and completed
  + The status of each request (e.g., 'Pending', 'Processing', 'Completed') to track the progress of each request.

**3. API Endpoints**

* **Upload API**: Accepts CSV files that contain product data (product names and input image URLs). It processes the data and generates a unique request ID for tracking purposes.
* **Status API**: Checks the current status of a request using the request ID. It returns whether the request is in 'Pending', 'In Progress', or 'Completed' status.
* **Download API**: Once the image processing is complete, users can download a CSV file containing the product data along with the processed image URLs using the request ID.

**4. S3 Component**

* After processing, the compressed images are uploaded and stored in an **Amazon S3** bucket.
* The URLs of the processed images are saved in the MongoDB database and included in the CSV file that users can download once the processing is finished.
* These URLs are publicly accessible, allowing clients to easily access the processed images via the output CSV file.

## Database Schema

Each CSV file will be represented as a document in the MongoDB collection.

Initially, when the CSV is uploaded, the status will be set to Pending.

After processing, the document will be updated with the compressed image URLs in the “output\_images” field, and the status will change to Completed.

***Schema in JSON format***:

{

  "\_id": *"<unique\_request\_id>"*,                // Unique identifier for the request

  "status": "<request\_status>",                // Status of the request: "Pending", "Processing", "Completed", or "Failed"

  "input": [

    {

      "S. No.": "<serial\_number>",             // Serial number from the CSV file

      "Product Name": "<product\_name>",        // Name of the product

      "Input Image Urls": [

        "<input\_image\_url\_1>",                // URL pointing to the first input image

        "<input\_image\_url\_2>"                 // URL pointing to the second input image

      ]

    },

    {

      "S. No.": "<serial\_number>",             // Serial number for the second product

      "Product Name": "<product\_name>",        // Name of the second product

      "Input Image Urls": [

        "<input\_image\_url\_1>"                 // URL pointing to the first input image

      ]

    }

  ],

  "created\_at": "<timestamp>",                 // Timestamp when the request was created (uploaded)

  "processed\_at": "<timestamp>",               // Timestamp when the processing was completed

  "output\_images": {

    "<input\_index\_0>": {

      "Output Images URLs": [

        "<output\_image\_url\_1>",               // Compressed and uploaded output image URL (S3 URL) for the first product

        "<output\_image\_url\_2>"

      ]

    },

    "<input\_index\_1>": {

      "Output Images URLs": [

        "<output\_image\_url\_1>"               // Compressed output image URL for the second product

      ]

    }

  }

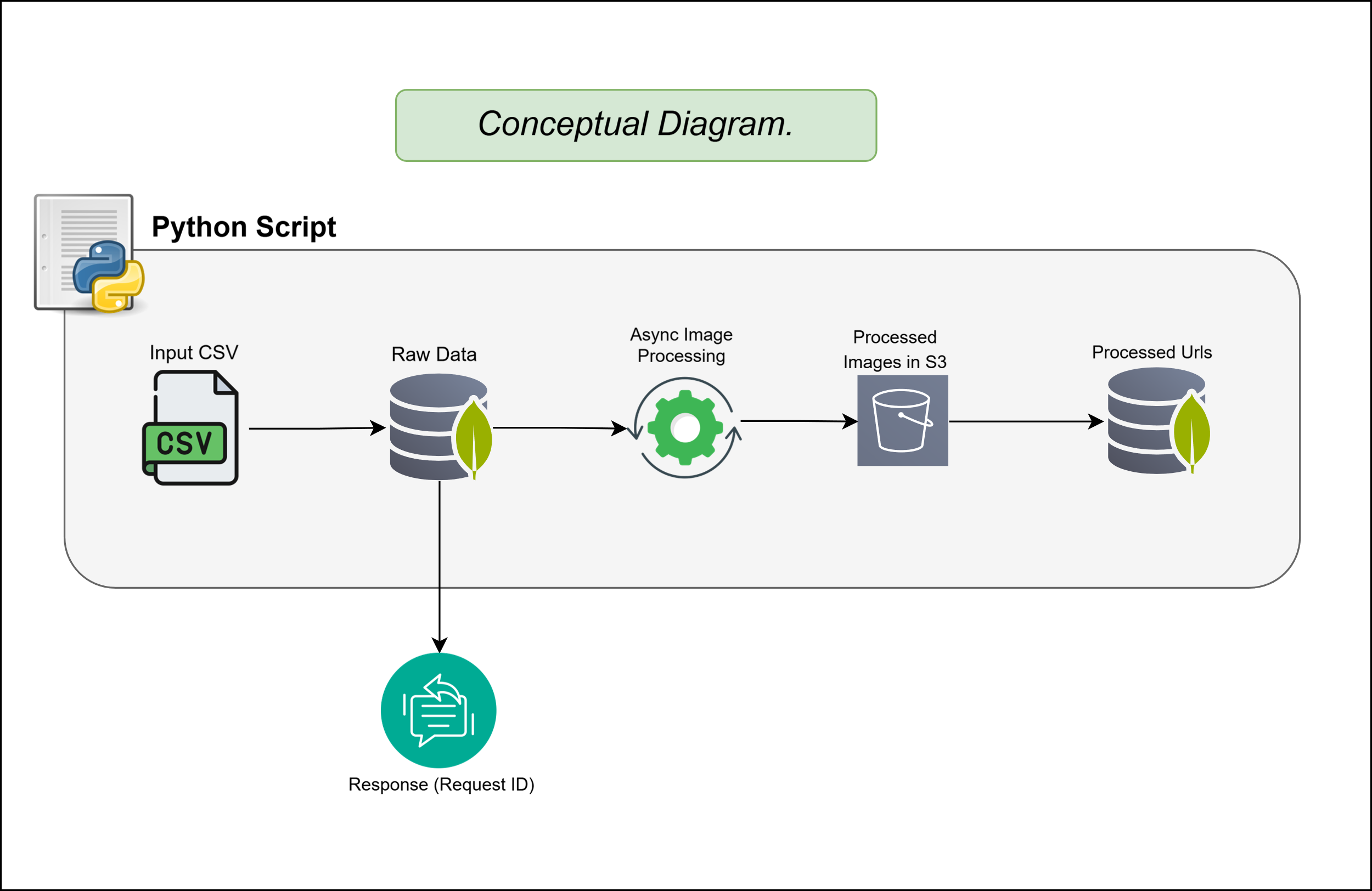
}

**MongoDB Schema Operations:**

1. **Insert Initial Record**:
   * When the CSV is uploaded, create a new record with the initial product details and set the status to "Pending".
2. **Update Status and Output URLs**:
   * After processing each image, update the corresponding output image URLs in the output\_images field.
   * Set the status to "Completed" when all images are processed.

## Flow Diagram

1. **CSV Upload and Initial Processing:**
   * CSV is uploaded via the **Upload API**.
   * The raw CSV data is saved in MongoDB with a **Request ID** and other metadata.
   * Status is marked as **Pending/Processing**.
2. **Image Processing Service:**
   * The system asynchronously interacts with an **Image Processing Service**.
   * Public image URLs from the CSV are fetched, compressed, and stored in an S3 bucket.
   * The S3 bucket URLs are updated in MongoDB.
3. **Status Update and Completion:**
   * The status is updated in MongoDB when the process completes, marking the status as **Completed**.
   * The processed image URLs are stored in MongoDB.
4. **Download Processed CSV:**

* The client can check the status via the **Status API**.
* When the status is **Completed**, the client can download the CSV from the **Download API**.
* The downloaded CSV contains the processed image URLs.

## API Documentation

* **Upload API (POST /upload)**
  + Accepts a CSV file and parses it into product data.
  + Response: A unique request ID.
  + Response Example:

{

"request\_id": "<Unique Request ID> "

}

* **Status API (GET /status/<request\_id>)**
  + Checks the processing status using the request ID.
  + Response Example:

{

"status": "Pending or Completed"

}

* **Download API (GET /download/<request\_id>)**
  + Returns a CSV file with processed data (input and output image URLs) after processing completion.
  + While processing, shows relevant message to the client.

*Response Example:*

*CSV file download containing the fields:*

**"S. No.", "Product Name", "Input Image URLs", "Output Image URLs"**

## Description of worker functions

1. **S3 Storage**:
   * **Purpose**: Store compressed images in an Amazon S3 bucket.
   * **Functionality**: Once the images are compressed, they are uploaded to an S3 bucket. The objects are made publicly accessible so that users can retrieve the processed images through URLs.
   * **Outcome**: Compressed images are available for download via public URLs in the processed CSV file.
2. **Download Function**:
   * **Purpose**: Download the images from the URLs provided in the CSV file.
   * **Functionality**: The worker fetches the original images from the URLs listed in the uploaded CSV file. It then processes the images (e.g., compresses them) before uploading them to S3.
   * **Outcome**: Images are fetched from the provided URLs for further processing.
3. **Compress Function**:
   * **Purpose**: Compress images to reduce file size.
   * **Functionality**: This function compresses each image to 50% of its original size to optimize storage and improve performance. The compressed images are then uploaded to S3.
   * **Outcome**: Images are reduced in size without significant loss in quality.
4. **MongoDB Functions**:
   * **Create Records**:
     + **Purpose**: Save the initial CSV data into the MongoDB database.
     + **Functionality**: When the CSV file is uploaded, the raw data (such as product name, input image URLs, and other metadata) is stored in MongoDB, including a unique request ID.
     + **Outcome**: The request data is persisted in MongoDB for further tracking and retrieval.
   * **Get Status**:
     + **Purpose**: Fetch the current processing status of a request.
     + **Functionality**: The function retrieves the status (e.g., "Pending", "Failed", or "Completed") of a specific request from MongoDB using the request ID.
     + **Outcome**: The current status of a request is returned to the user.
   * **Update Status**:
     + **Purpose**: Update the request status based on the processing stage.
     + **Functionality**: Once image processing is complete (or fails), the status is updated in MongoDB to either "Completed" or "Failed".
     + **Outcome**: The status reflects the real-time state of the processing request.
   * **Get the Output**:
     + **Purpose**: Fetch the complete request data once processing is completed.
     + **Functionality**: When the status of a request is "Completed", the function retrieves the entire record from MongoDB, including the processed image URLs, for the user to download.
     + **Outcome**: The full processed data (input and output URLs) is returned for CSV generation.

## Postman Collections

## GitHub Repo

* Access the codebase from this [***link.***](https://github.com/Chirag-Passi/csv-image-processor)