### **API Documentation**

#### **1. Upload CSV**

* **Endpoint**: /upload\_csv
* **Method**: POST
* **Description**: Uploads a CSV file containing product image URLs for processing.
* **Request Body**:
  + A multipart/form-data request containing a CSV file with at least two columns: Product Name and Input Image Urls.
* **Response**:
  + **200 OK**: Returns {"request\_id": "<uuid>"} on success.
  + **400 Bad Request**: Returns an error message if no file is uploaded or if the CSV is empty.
  + **500 Internal Server Error**: Returns an error message if there is a failure processing the CSV.

#### **2. Check Request Status**

* **Endpoint**: /check\_status/<uuid:request\_id>
* **Method**: GET
* **Description**: Checks the status of a processing request based on the provided request\_id.
* **Response**:
  + **200 OK**: Returns {"request\_id": "<uuid>", "status": "<status>"}. The status will be one of pending, completed, or error.
  + **404 Not Found**: If the request\_id does not exist.

#### **3. Download Processed CSV**

* **Endpoint**: /download\_csv/<uuid:request\_id>
* **Method**: GET
* **Description**: Downloads the processed image URLs for a specific request as a CSV file.
* **Response**:
  + **200 OK**: Returns the processed CSV file.
  + **404 Not Found**: If no data is found for the request\_id.

### **Asynchronous Workers Documentation**

#### **1. process\_csv**

* This is the main worker function for processing the uploaded CSV. It inserts a record into the requests table with the status pending and starts a new thread to process each image in the CSV.
* **Workflow**:
  1. Insert a new record into the requests table.
  2. For each row in the CSV, spawn a new thread to call process\_image for each image URL.
  3. Once all threads have completed, insert the results into the image table with the status completed.
  4. Update the requests table to mark the request as completed.

#### **2. process\_image**

* This function processes an individual image URL by downloading the image, saving it to Azure Blob Storage (original and compressed versions), and inserting relevant records into the database.
* **Workflow**:
  1. Download the image from the URL.
  2. Save the original image to Azure Blob Storage in the originalimages container.
  3. Compress the image and save it to Azure Blob Storage in the compressedimages container.
  4. Add a record to the image table with the URLs of the original and compressed images.
  5. Return the URLs to the main process\_csv function.

#### **3. check\_status**

* This function checks the status of a request by querying the requests table for the request\_id. If the request is completed, it returns the status (completed or pending).

#### **4. get\_data\_db**

* This function retrieves the processed image data for a completed request from the image and requests tables and returns it as a DataFrame. It is used when generating the download CSV file.

### **Flask Application Workflow**

1. **Uploading CSV**:
   * The user uploads a CSV file with product image URLs.
   * The server validates the file and starts a background task (process\_csv) to handle the processing.
2. **Checking Status**:
   * The user can query the processing status by providing the request\_id.
   * The server checks the database for the request status and returns it.
3. **Downloading Processed CSV**:
   * Once the images are processed, the user can download a CSV file containing the original and compressed URLs for the products.

### **Summary**

This approach ensures that:

* Image processing happens asynchronously using threads, allowing the system to handle multiple image URLs in parallel.
* The database schema is designed to keep track of requests and images, ensuring data integrity.
* The API provides clear endpoints for uploading, checking status, and downloading results.

This schema, API, and worker setup should allow for scalable, efficient image processing and management of user requests.