Image Processing & CSV Upload API Documentation

# Table of Contents

1. API Endpoints

2. CSV File Format

3. Image Processing Flow

4. Webhook Support

5. Error Handling

6. Dependencies

# 1. API Endpoints

## 1.1 Upload CSV

\*\*Endpoint:\*\* `/api/upload`  
\*\*Method:\*\* `POST`  
\*\*Request Body:\*\*  
- \*\*file\*\* (required): CSV file containing product details and image URLs.  
- \*\*webhookUrl\*\* (optional): A URL to notify when processing is complete.  
  
\*\*Response:\*\*  
- Status Code: `202 Accepted`  
- JSON:  
{  
 "message": "File validated and processing started",  
 "requestId": "unique-request-id"  
}

## 1.2 Get Status

\*\*Endpoint:\*\* `/api/status/:requestId`  
\*\*Method:\*\* `GET`  
\*\*Request Params:\*\*  
- `requestId` (required): The unique ID provided when the file was uploaded.  
  
\*\*Response:\*\*  
- Status Code: `200 OK` (if found) or `404 Not Found` (if requestId not found)  
- JSON:  
{  
 "requestId": "unique-request-id",  
 "status": "complete/pending",  
 "entries": [  
 {  
 "entityId": "1",  
 "title": "Product1",  
 "status": "complete/pending",  
 "inputImageUrls": ["input-url-1", "input-url-2"],  
 "outputImageUrls": ["output-url-1", "output-url-2"]  
 },  
 ...  
 ]  
}

# 2. CSV File Format

\*\*Columns in CSV:\*\*  
1. \*\*EntityID\*\*: Unique identifier for the product.  
2. \*\*Title\*\*: Product name.  
3. \*\*InputImageUrls\*\*: Comma-separated URLs of the input images.  
  
\*\*Example CSV:\*\*  
```csv  
EntityID,Title,InputImageUrls  
1,Product1,"https://example.com/image1.jpg, https://example.com/image2.jpg"  
2,Product2,"https://example.com/image3.jpg"  
```

# 3. Image Processing Flow

## 3.1 File Upload & Validation

The uploaded CSV is parsed using `csv-parser`. Each row contains:  
- `EntityID` (Product ID)  
- `Title` (Product Name)  
- `InputImageUrls` (Comma-separated URLs)  
  
Data is grouped by `EntityID` so that multiple images for the same entity are processed together.

## 3.2 Image Download & Compression

For each input image URL:  
- The image is downloaded using `axios`.  
- The image is compressed using `sharp` (JPEG format, 50% quality).  
- The compressed image is saved locally.

## 3.3 Uploading to Imgur

Once compressed, each image is uploaded to Imgur using their API. The image URL is returned by Imgur and stored as part of the output.

# 4. Webhook Support

If a `webhookUrl` is provided in the request body of the `/api/upload` endpoint, a POST request is sent to the webhook URL once image processing is complete, including the `requestId` and the status of the task.  
  
\*\*Example webhook payload:\*\*  
```json  
{  
 "requestId": "unique-request-id",  
 "status": "complete"  
}  
```

# 5. Error Handling

- If the CSV file is missing or invalid, the API returns a `400 Bad Request` error.  
- If image processing fails (e.g., download error, upload error), the error is logged, but processing continues for the remaining images.  
- The status endpoint returns `404` if the `requestId` is not found in the database.

# 6. Dependencies

- \*\*`axios`\*\*: For downloading images and making HTTP requests (Imgur upload, webhook).  
- \*\*`sharp`\*\*: For compressing and resizing images.  
- \*\*`csv-parser`\*\*: For parsing the CSV file.  
- \*\*`mongoose`\*\*: For storing and retrieving the image processing details.  
- \*\*`uuid`\*\*: For generating unique request IDs.  
- \*\*`form-data`\*\*: For handling multipart form data during Imgur uploads.  
  
To install all dependencies, run:  
```bash  
npm install axios sharp csv-parser mongoose uuid form-data  
```