Low-Level Design: Image Processing System

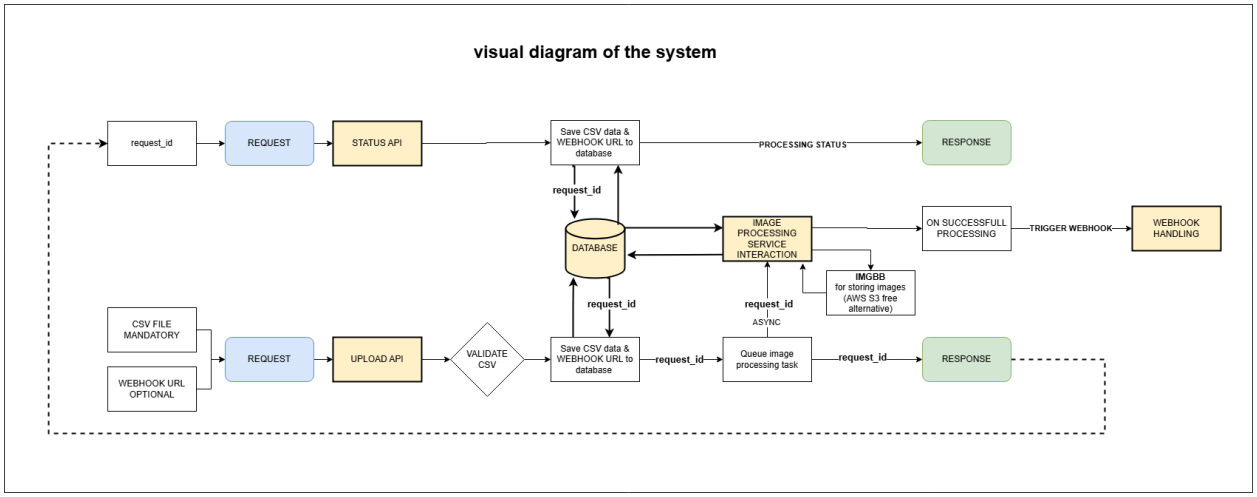
1. Introduction

This document details the technical design of an image processing system that accepts CSV files with product data and image URLs, validates the data, asynchronously compresses images to 50% quality, uploads them to ImgBB, stores results in a database, and provides status updates via APIs with optional webhook trigger. Built with Django, Celery, PostgreSQL, and ImgBB integration.

2. System Architecture

The system comprises interconnected components for handling CSV uploads, image processing, and status tracking.

Diagram



Components

- **Django REST APIs**: Manage CSV uploads and status queries.
- **Celery + Redis**: Handle asynchronous image processing.
- **PostgreSQL**: Store request and product data.
- **ImgBB**: Host processed images with public URLs.
- **Webhook**: Notify users post-processing.

3. Component Design

Upload API

- **Purpose**: Accept CSV, validate, store data, queue processing.
- **Input**: CSV file (Serial Number, Product Name, Input Image Urls), optional webhook_url.
- **Output**: JSON with request_id.
- **Logic**: Validates CSV, saves to DB, triggers Celery task.

Celery Image Processor

- **Purpose**: Compress and upload images.
- **Input**: request_id.
- **Output**: ImgBB URLs saved in DB.
- **Logic**: Downloads images, compresses to 50% quality, uploads to ImgBB, triggers webhook.

Status API

- **Purpose**: Retrieve processing status and results.
- **Input**: request_id.
- **Output**: JSON with status and product data.
- **Logic**: Queries DB for request and entries.

Database

- **Purpose**: Store CSV requests and product entries.
- **Interaction**: Updated by APIs and Celery.

Webhook Handler

- **Purpose**: Notify user when processing completes.
- **Input**: request_id, processed data.
- **Output**: POST request to webhook_url.

4. DATABASE SCHEMA

Table: CSVRequest

Sr no.	Column Name	Data Type	Constraints	Description
1	request_id	UUID	Primary Key	Unique identifier for each CSV file submission.
2	status	VARCHAR(20)	Default: 'pending'	Tracks processing status (pending, processing, completed).
3	created_at	TIMESTAMP	Not Null	Timestamp of when the CSV was uploaded.
4	webhook_url	TEXT	Nullable	Optional URL to send webhook callback after processing.

Table: ProductEntry

Sr no.	Column Name	Data Type	Constraints	Description
1	id	Integer	Primary Key, Auto-Increment	Unique identifier for each product row.
2	request_id	UUID	Foreign Key (CSVRequest)	Links to the CSV request this row belongs to.
3	serial_no	Integer	Not Null	Serial number from the CSV.
4	product_name	VARCHAR(255)	Not Null	Name of the product from the CSV.
5	input_urls	TEXT	Not Null	Comma-separated list of input image URLs.
6	output_urls	TEXT	Nullable	Comma-separated list of output image URLs (populated after processing).

5. API DOCUMENTATION

Base URL: http://localhost:8000/

1. Upload API

Description

Accepts a CSV file, validates its format, stores the data, initiates asynchronous image processing, and returns a unique request_id.

Endpoint

URL: /api/p/upload/
Method: POST

Request

Content-Type: multipart/form-data
Parameters:

Parameter	Type	Required	Description
csv_file	File	Yes	CSV file with columns: "Serial Number", "Product Name", "Input Image Urls".
webhook_url	String	No	Optional URL to receive a webhook callback when processing completes.

CSV Format:

Headers: "Serial Number", "Product Name", "Input Image Urls"
Example:

```
S. No.,Product Name,Input Image Urls
1,SKU1,https://example.com/img1.jpg,https://example.com/img2.jpg
2,SKU2,https://example.com/img3.jpg,https://example.com/img4.jpg
```

Response:

Success:

Status Code: 202 Accepted
Content-Type: application/json
Description: Returns a unique request_id for the CSV submission.
Body:

```
{
  "request_id": "550e8400-e29b-41d4-a716-446655440000"
}
```

Error:

Status Code: 400 Bad Request
Content-Type: application/json
Description: Indicates missing file or incorrect CSV format (e.g., wrong headers).
Body:

```
{
  "error": "No CSV file provided"
}
```

OR

```
{
  "error": "Invalid CSV format"
}
```

2. Status API

Description

Allows users to query the processing status of a CSV submission using its request_id and retrieve processed image URLs when complete.

Endpoint

URL: /status/<request_id>/
Method: GET
Path Parameter:

Parameter	Type	Required	Description
request_id	**UUID**	**Yes**	**Unique identifier returned by the Upload API.**

Request: No body required.

Response:

Success:

Status Code: 200 OK

Content-Type: application/json

Description: Returns the current status and product details. Output Image Urls is null until processing is complete.

Body (Pending/Processing):


```
{
  "request_id": "550e8400-e29b-41d4-a716-446655440000",
  "status": "pending",
  "products": [
    {
      "Serial Number": 1,
      "Product Name": "SKU1",
      "Input Image Urls":
"https://example.com/img1.jpg,https://example.com/img2.jpg",
      "Output Image Urls": null
    },
    {
      "Serial Number": 2,
      "Product Name": "SKU2",
      "Input Image Urls":
"https://example.com/img3.jpg,https://example.com/img4.jpg",
      "Output Image Urls": null
    }
  ]
}
```


Body (Completed):


```
{
  "request_id": "550e8400-e29b-41d4-a716-446655440000",
  "status": "completed",
  "products": [
    {
      "Serial Number": 1,
      "Product Name": "SKU1",
      "Input Image Urls":
"https://example.com/img1.jpg,https://example.com/img2.jpg",
      "Output Image Urls":
"media/output_550e8400..._1_0.jpg,media/output_550e8400..._1_1
.jpg"
```

```
    },
    {
      "Serial Number": 2,
      "Product Name": "SKU2",
      "Input Image Urls":
"https://example.com/img3.jpg,https://example.com/img4.jpg",
      "Output Image Urls":
"media/output_550e8400..._2_0.jpg,media/output_550e8400..._2_1
.jpg"
    }
  ]
}
```

Error:

Status Code: 404 Not Found
Content-Type: application/json
Description: Indicates the request_id does not exist.
Body:

```
{
  "error": "Request ID not found"
}
```

3. Webhook Callback (Bonus)

Description
Triggered automatically after all images in a CSV are processed, sending the output data to the provided webhook_url.

Endpoint

URL: User-provided webhook_url **(from Upload API)**
Method: POST

Request (Sent by the System)

Content-Type: application/json
Description: A list of objects matching the output CSV format, sent when status becomes "completed".

Body:

```
[
  {
```

```
    "Serial Number": 1,
    "Product Name": "SKU1",
    "Input Image Urls":
"https://example.com/img1.jpg,https://example.com/img2.jpg",
    "Output Image Urls":
"media/output_550e8400..._1_0.jpg,media/output_550e8400..._1_1.jpg"
  },
  {
    "Serial Number": 2,
    "Product Name": "SKU2",
    "Input Image Urls":
"https://example.com/img3.jpg,https://example.com/img4.jpg",
    "Output Image Urls":
"media/output_550e8400..._2_0.jpg,media/output_550e8400..._2_1.jpg"
  }
]
```

Notes

- No response is expected from the webhook endpoint.
- Errors (e.g., unreachable URL) are silently ignored.

Error Codes

Status Code	Meaning	Description
202	Accepted	CSV upload accepted and processing queued.
200	OK	Status retrieved successfully.
400	Bad Request	Invalid request (e.g., missing file, wrong CSV format).
404	Not Found	request_id not found in the system.

6. Asynchronous Workers Documentation

Overview

The asynchronous worker handles background image processing for CSV submissions using Celery with Redis as the message broker. It compresses images to 50% quality, uploads them to ImgBB, updates the database with public URLs, and triggers a webhook when complete.

Worker Function: **process_images**

Description

The process_images task processes all image URLs for a given CSV submission identified by request_id. It compresses images, uploads them to ImgBB for public hosting, stores the resulting URLs in the database, and sends a webhook notification upon completion.

Details

- **Module**: processing/tasks.py
- **Task Name**: process_images
- **Trigger**: Called via process_images.delay(request_id) from the Upload API.
- **Dependencies**: Celery, Redis, Pillow, Requests, Django ORM, ImgBB API key (from settings.IMGBB_API_KEY).

Input

- **Parameter**: request_id (String)
    - **Description**: UUID of the CSV request.
    - **Example**: "550e8400-e29b-41d4-a716-446655440000"

Functionality

1. Start Processing:
    - Fetches CSVRequest by request_id and sets status to "processing".
2. Process Entries:
    - Iterates over ProductEntry objects for the request.
    - For each entry:
        i. Splits input_urls into a list.
        ii. For each URL:
            1. Skips processing if no "http" (keeps as-is).
            2. Downloads image, compresses to 50% quality with Pillow.

3. Uploads to ImgBB via API and retrieves public URL.
4. Adds URL to output_urls.

iii. Saves output_urls to the entry.

3. Complete Request:
    - Updates CSVRequest.status to "completed".
4. Webhook Notification:
    - If webhook_url exists, sends a JSON payload:

```
[
  {"Serial Number": <int>, "Product Name": <str>, "Input Image Urls": <str>, "Output Image Urls": <str>},
  ...
]
```

Error Handling

- **Invalid URLs**: Non-HTTP URLs are passed through unchanged.
- **Upload/Webhook Failures**: Silently ignored.

Output

- **Database**: Updates *ProductEntry.output_urls* with ImgBB URLs and *CSVRequest.status* to "completed".
- **Webhook**: Sends payload with processed data.

Example

- **Input**: request_id = "550e8400...", input_urls = "https://example.com/img1.jpg,not-a-url"
- **Output**: output_urls = "https://ibb.co/img1-compressed.jpg,not-a-url"

Implementation

```
@shared_task
def process_images(request_id):
    csv_request = CSVRequest.objects.get(request_id=request_id)
    csv_request.status = 'processing'
    csv_request.save()

    # Process all entries for this request
    entries = ProductEntry.objects.filter(request=csv_request)
```

```
    for entry in entries:
        input_urls = entry.input_urls.split(',')
        output_urls = []

        for idx, url in enumerate(input_urls):
            print(url,"start")
            if "http" not in url:
                output_urls.append(url)
                continue
            response = requests.get(url.strip())
            img = Image.open(BytesIO(response.content))
            output_buffer = BytesIO()
            img.save(output_buffer, format="JPEG", quality=50)

            output_buffer.seek(0)

            # Upload to ImgBB
            imgbb_api_key = settings.IMGBB_API_KEY
            response = requests.post(
                "https://api.imgbb.com/1/upload",
                data={"key": imgbb_api_key},
                files={"image": output_buffer}
            )
            print(response)
            print(response.json())
            output_urls.append(response.json()["data"]["url"])

        entry.output_urls = ','.join(output_urls)
        entry.save()

    # Update request status
    csv_request.status = 'completed'
    csv_request.save()

    # Trigger webhook if provided
    if csv_request.webhook_url:
        webhook_payload = [
            {
                "Serial Number": entry.serial_no,
                "Product Name": entry.product_name,
```

```
                "Input Image Urls": entry.input_urls,
                "Output Image Urls": entry.output_urls
            } for entry in entries
        ]
        try:
            requests.post(csv_request.webhook_url, json=webhook_payload,
timeout=5)
        except requests.RequestException as e:
            # Log the error (in production); for now, ignore silently
            pass
```

Notes

- **Run Command**: celery -A image_processor worker -l info
- **Configuration**: Requires Redis and IMGBB_API_KEY in settings.py.
- **Storage**: Uses ImgBB for public URLs instead of local files.

7. Tech Stack

- **Backend**: Django + Django REST Framework
- **Async**: Celery + Redis
- **Database**: PostgreSQL
- **Image Processing**: Pillow
- **Storage**: ImgBB API