1️⃣ System Components & Responsibilities

Component	Responsibility
API Gateway (`server.js`)	Handles incoming HTTP requests, routes them to the correct controllers.
Multer (`uploadRoutes.js`)	Handles CSV file uploads and validation.
CSV Parser (`uploadRoutes.js`)	Reads and parses CSV data.
MongoDB (`Request.js`, `Product.js`)	Stores metadata about processing requests and images.
Image Processing Service (`imageProcessing.js`)	Downloads, compresses, and saves images locally.
Static File Server (`server.js`)	Serves processed images via `http://localhost:3000/images/{filename}`.
Webhook Handling (`imageProcessing.js`)	Sends a POST request to a webhook URL after processing completes.

2️⃣ Sequence Flow

Client Uploads CSV

- The `/upload` API is called with a CSV file and optional `webhookUrl`.
- The file is parsed, and request metadata is stored in MongoDB.
- The `processImages()` function is triggered asynchronously.

Image Processing

- Each image URL is downloaded using `axios`.
- The image is compressed using `sharp` (50% quality).
- The compressed image is saved locally in `/uploads/`.
- The new image URL (`http://localhost:3000/images/{filename}`) is stored in MongoDB.

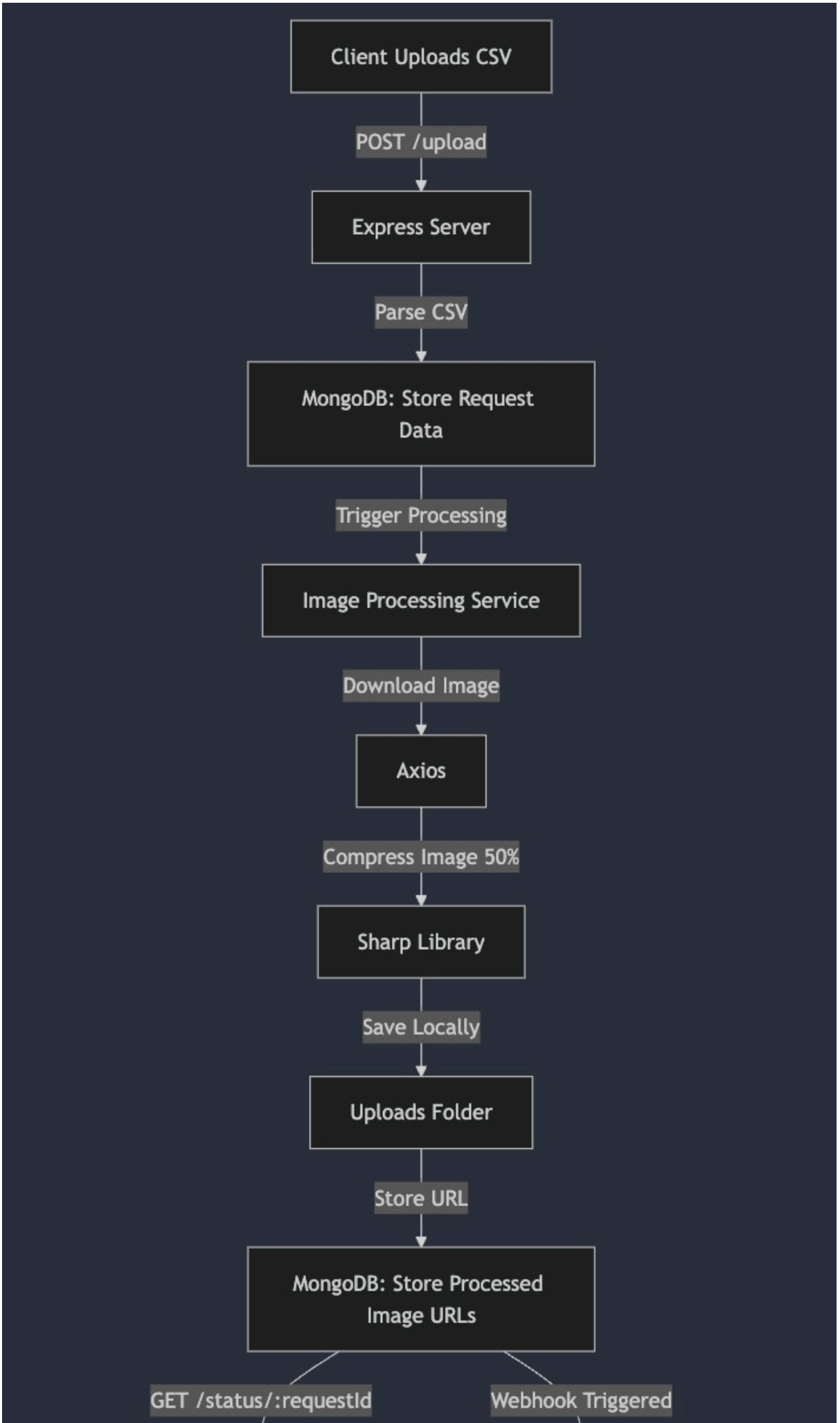
Status Checking

- The `/status/:requestId` API retrieves processing status and image URLs from MongoDB.

Webhook Notification

- If a `webhookUrl` was provided, a POST request is sent once processing is complete.

3️⃣ Visual Diagram


Client Uploads CSV
POST /upload
Express Server
Parse CSV
MongoDB: Store Request Data
Trigger Processing
Image Processing Service
Download Image
Axios
Compress Image 50%
Sharp Library
Save Locally
Uploads Folder
Store URL
MongoDB: Store Processed Image URLs
GET /status/:requestId
Webhook Triggered

4️⃣ Summary of Components & Functionality

🟢 API Gateway (`server.js`)

- Handles `/upload` and `/status/:requestId` routes.
- Serves images via `/images/{filename}`.

🟠 File Upload & Parsing (`uploadRoutes.js`)

- Uses `multer` to handle CSV uploads.
- Parses CSV and stores metadata in MongoDB.

🔵 Image Processing (`imageProcessing.js`)

- Fetches images using `axios`.
- Compresses images with `sharp`.
- Saves and stores processed image URLs.

🟣 Webhook Handling (`imageProcessing.js`)

- Sends notifications once processing completes.

1️⃣ Database Schema

The system uses **MongoDB** with two collections:

1. `requests` → Tracks processing requests.
2. `products` → Stores product details and processed image URLs.

MongoDB Schema Definition

1. `requests` Collection

Field	Type	Description
`_id`	`ObjectId`	Unique ID generated by MongoDB.
`requestId`	`String`	Unique identifier for the request (UUID).
`status`	`String`	Status of processing (`pending`, `processing`, `completed`).
`products`	`Array`	List of products associated with this request.
`webhookUrl`	`String`	URL to notify when processing completes (optional).

2. `products` Collection

Field	Type	Description
`_id`	`ObjectId`	Unique ID generated by MongoDB.
`requestId`	`String`	Unique request ID this product belongs to.
`productName`	`String`	Name of the product.
`inputImages`	`Array[String]`	List of original image URLs.
`outputImages`	`Array[String]`	List of processed image URLs.

2️⃣ API Documentation

Base URL

```
http://localhost:3000
```

1. Upload CSV (Asynchronous Processing)

- **Endpoint**: `POST /upload`
- **Description**: Uploads a CSV file, validates the format, and starts image processing asynchronously.
- **Request Format (Multipart Form-Data)**:

Key	Type	Required	Description
`file`	`File`	✅ Yes	CSV file containing product & image URLs.
`webhookUrl`	`String`	❌ No	Optional webhook URL for processing completion notification.

- **Response** (`201 Created`):

```
{
  "requestId": "1e25f846-de38-4d6c-8e8e-c157e1e73bb9"
}
```

2. Check Processing Status

- **Endpoint:** `GET /status/:requestId`
- **Description:** Retrieves the processing status and processed image URLs.
- **Request Parameters:**

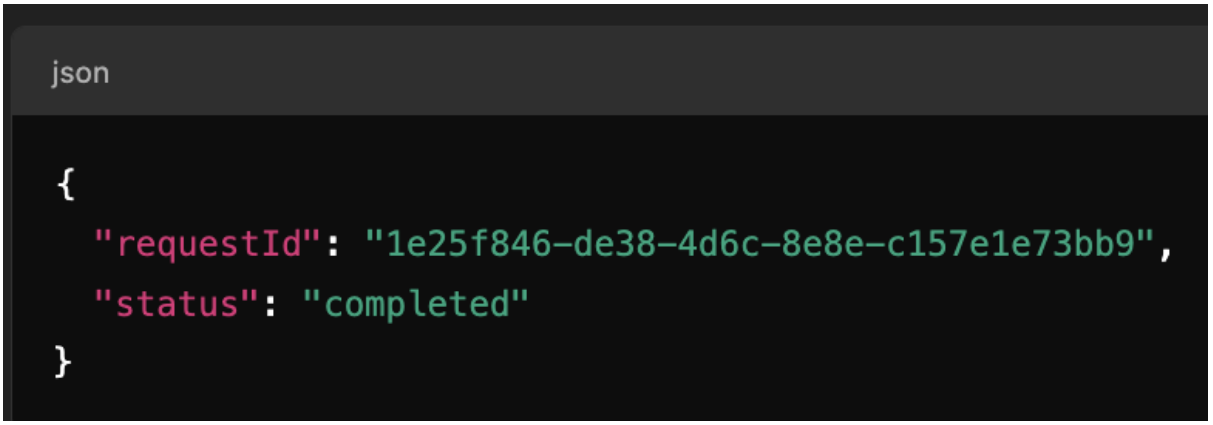
Parameter	Type	Required	Description
`requestId`	`String`	✅ Yes	Request ID returned from `/upload`.

- **Response (`200 OK`):**

```
{
  "status": "completed",
  "products": [
    {
      "productName": "SKU1",
      "inputImages": [
        "https://sample-videos.com/img/Sample-jpg-image-50kb.jpg"
      ],
      "outputImages": [
        "http://localhost:3000/images/processed-1.jpg"
      ]
    }
  ]
}
```

3. Webhook Notification (Sent by Server)

- Method: POST
- Description: The backend automatically sends a webhook notification when processing is completed.
- Example Payload:



3️⃣ Asynchronous Workers Documentation

Overview

The system processes images asynchronously to avoid blocking the API.

- Step 1: The `/upload` API stores the request in MongoDB with `status: "processing"`.
- Step 2: A background worker function (`processImages()`) handles image compression outside the request-response cycle.
- Step 3: After processing, images are stored, and the request `status` is updated to `"completed"`.
- Step 4: If a `webhookUrl` was provided, the system sends a notification.

Worker Function (`processImages()`)

```
async function processImages(requestId, products, webhookUrl) {
  for (let product of products) {
    if (!product["Input Image Urls"]) {
      console.error(`Skipping product ${product["Product Name"]}: Missing Input Image Urls`);
      continue;
    }
    const outputImages = [];
    for (let url of product["Input Image Urls"].split(",")) {
      try {
        const response = await axios({ url: url.trim(), responseType: "arraybuffer" });
        const compressedBuffer = await sharp(response.data).jpeg({ quality: 50 }).toBuffer();
        const filename = `${uuidv4()}.jpg`;
        const filepath = path.join(__dirname, "../uploads", filename);
        fs.writeFileSync(filepath, compressedBuffer);
        outputImages.push(`http://localhost:${port}/images/${filename}`);
      } catch (error) {
        console.error("Image processing error", error);
      }
    }
    await Product.create({
      requestId,
      productName: product["Product Name"],
      inputImages: product["Input Image Urls"].split(","),
      outputImages,
    });
  }
  await Request.findOneAndUpdate({ requestId }, { status: "completed" });
  if (webhookUrl) axios.post(webhookUrl, { requestId, status: "completed" });
}
```

Why is This Asynchronous?

- The worker does not block API requests.
- Clients get an immediate response (`requestId`) without waiting for images to be processed.
- Scalable: Can be extended with message queues like RabbitMQ or Redis.

Section	Details
Database Schema	`requests` collection tracks processing, `products` stores image data.
API Documentation	`/upload`, `/status/:requestId`, webhook documentation.
Asynchronous Workers	`processImages()` runs separately for efficient image compression.

Postman collection :-

https://speeding-flare-935611.postman.co/workspace/CRUD-App~e7795aaa-54ef-4505-a156-e36196e348dd/collection/15877212-0b27d794-f5af-4136-82af-0e19c20c99f0?action=share&creator=15877212

Github Link:-

https://github.com/Sumedh-stack/csv-image-processing-api