Image Processing Pipeline — Low-Level Design (LLD)

**Stack**: Node.js (Express) · MongoDB (Mongoose) · BullMQ (Redis) · Sharp · Axios · json2csv · Local/S3 storage

1) Scope & Goals

* Accept a CSV of products with comma‑separated image URLs.
* Asynchronously download & compress each image to ~50% quality.
* Persist inputs/outputs and per‑item status.
* Provide:
  + **Upload API** → returns a requestId immediately.
  + **Status API** → consolidated progress + per‑product status + output CSV URL when ready.
  + **(Bonus)** Webhook → POST to a client URL when the whole request finishes.

**Non‑functional**

* Horizontal worker scaling (BullMQ concurrency, multiple worker replicas)
* At‑least‑once processing with idempotency on storage path
* Bounded retries with backoff
* Structured logging (correlation by requestId)
* Timeouts for network I/O

2) High‑Level Architecture

flowchart LR  
 A[Client] -- CSV (multipart) --> B[Upload API (Express)]  
 B -->|validate + persist| C[(MongoDB)]  
 B -->|enqueue request| Q1[(BullMQ: requestQueue)]  
 Q1 --> W1[processRequest.worker]  
 W1 -->|enqueue N images| Q2[(BullMQ: imageQueue)]  
 Q2 --> W2[processImage.worker xN]  
 W2 -->|download+compress| IMG[Storage (Local/S3)]  
 W2 -->|update| C  
 W1 -->|poll/aggregate| C  
 W1 -->|generate output.csv| IMG  
 W1 -->|update final status| C  
 W1 -->|POST completion| WH[Client Webhook]  
 A <-- status JSON / outputCsvUrl --> B

3) Detailed Sequence

**3.1 Upload Request**

sequenceDiagram  
 participant Client  
 participant API as Upload API  
 participant DB as MongoDB  
 participant Q1 as requestQueue  
 Client->>API: POST /api/v1/upload (multipart: file, webhook\_url?)  
 API->>DB: create Request(status=VALIDATING)  
 API->>API: parse CSV & validate rows  
 API->>DB: upsert Products & Images (PENDING)  
 API->>DB: Request.totalItems  
 API->>Q1: add({requestId})  
 API-->>Client: 201 {requestId, status: "QUEUED"}

**3.2 Request Processing & Image Fan‑out**

sequenceDiagram  
 participant W1 as processRequest.worker  
 participant DB as MongoDB  
 participant Q2 as imageQueue  
 W1->>DB: update Request(PROCESSING)  
 W1->>DB: find Images by requestId  
 W1->>Q2: add N jobs {imageId, requestId}  
 loop Every 3s  
 W1->>DB: count(done, failed, total)  
 alt all finished  
 W1->>W1: generate output CSV  
 W1->>DB: update Request(finalStatus, counts, outputCsvUrl)  
 W1-->>Client: POST webhook (optional)  
 end  
 end

**3.3 Image Worker**

sequenceDiagram  
 participant W2 as processImage.worker  
 participant DB as MongoDB  
 participant Net as Internet  
 participant ST as Storage  
 W2->>DB: find Image by id  
 W2->>Net: GET inputUrl (timeout 15s)  
 W2->>W2: sharp().jpeg({quality:50})  
 W2->>ST: saveBytes(requestId/filename)  
 W2->>DB: set Image{status:DONE, outputUrl}  
 W2->>DB: inc Request.processedItems  
 Note over W2: on error → status=FAILED, inc failedItems

4) Components & Responsibilities

**4.1 API Layer (Express)**

* **Upload Route**
  + Accept multipart CSV under field file.
  + Optional webhook\_url form field.
  + Validate header names + each row (S. No., Product Name, Input Image Urls).
  + Persist Request, Product, Image docs; enqueue requestQueue.
* **Status Route**
  + GET by :requestId.
  + Returns request summary + per‑product image array + outputCsvUrl when ready.
* **Webhook Test Route** (dev only)
  + Echo receiver to verify outgoing callbacks.

**4.2 Services**

* **csvValidator.service** — streams CSV, validates, writes to DB.
* **image.service** — axios download → sharp compress → storage.saveBytes.
* **storage.service** — pluggable local/S3 backends, returns public URL.
* **csvGenerator.service** — builds and stores output.csv.
* **webhook.service** — POST completion payload (swallows errors).

**4.3 Workers (BullMQ)**

* **processRequest.worker**
  + Sets Request → PROCESSING; enqueues one imageQueue job per image.
  + Periodically aggregates counts and finalizes request; generates output CSV; optionally fires webhook.
* **processImage.worker**
  + Does the actual download/compress/store. Marks image DONE/FAILED and increments counters.

5) Data Model (MongoDB / Mongoose)

**5.1 Request**

| Field | Type | Notes |
| --- | --- | --- |
| requestId | String (unique, indexed) | Correlation key returned to client |
| status | Enum( RECEIVED, VALIDATING, QUEUED, PROCESSING, PARTIAL\_SUCCESS, COMPLETED, FAILED ) | Pipeline state |
| totalItems | Number | #images discovered |
| processedItems | Number | #images done |
| failedItems | Number | #images failed |
| webhookUrl | String | Optional |
| outputCsvUrl | String | Public URL to generated CSV |
| error | String | Error summary when FAILED |
| createdAt/updatedAt | Date | Auto timestamps |

**Indexes**: { requestId: 1 } unique.

**5.2 Product**

| Field | Type | Notes |
| --- | --- | --- |
| requestId | String (indexed) | Parent request |
| serialNo | Number | From CSV |
| productName | String | From CSV |
| createdAt/updatedAt | Date |  |

**Indexes**: { requestId: 1, serialNo: 1 } (optional for faster status).

**5.3 Image**

| Field | Type | Notes |
| --- | --- | --- |
| requestId | String (indexed) | Parent request |
| productId | ObjectId → Product | FK |
| inputUrl | String | Source URL |
| outputUrl | String | Result URL |
| position | Number | Order within a product |
| status | Enum(PENDING, DONE, FAILED) |  |
| error | String | Failure reason |
| createdAt/updatedAt | Date |  |

**Indexes**: { requestId: 1 }, { productId: 1, position: 1 }.

6) APIs — Contracts

**6.1 Upload API**

**POST** /api/v1/upload

* **Content-Type**: multipart/form-data
* **Form fields**:
  + file (File, required) — CSV
  + webhook\_url (String, optional)
* **Responses**
  + 201 Created
* { "requestId": "uuid", "status": "QUEUED" }
  + 400 Bad Request — { error: string }

**CSV Columns (exact names)** S. No. | Product Name | Input Image Urls

* Input Image Urls is comma‑separated list of absolute URLs; order preserved.

**6.2 Status API**

**GET** /api/v1/status/:requestId

* **200 OK**

{  
 "requestId":"uuid",  
 "status":"PROCESSING",  
 "totalItems":12,  
 "processedItems":8,  
 "failedItems":1,  
 "outputCsvUrl": null,  
 "products":[  
 {"serialNo":1,"productName":"SKU1","images":[{"inputUrl":"...","outputUrl":"...","status":"DONE"}]}]  
}

* 404 Not Found — requestId unknown.

**6.3 Webhook (outgoing)**

* Triggered when all images are DONE|FAILED.
* **POST** to provided webhook\_url with JSON:

{  
 "event":"request.completed",  
 "requestId":"uuid",  
 "status":"COMPLETED|PARTIAL\_SUCCESS|FAILED",  
 "outputCsvUrl":"https://.../output.csv",  
 "totals":{"total":12,"done":11,"failed":1},  
 "completedAt":"2025-01-01T00:00:00.000Z"  
}

* **Security** (optional hardening):
  + X-Signature: sha256=... HMAC of body using shared secret.
  + Receiver should verify before accepting.

7) Configuration

| Env | Example | Notes |
| --- | --- | --- |
| PORT | 8000 | API port |
| MONGO\_URI | mongodb://localhost:27017/imgsvc |  |
| REDIS\_URL | redis://localhost:6379 |  |
| STORAGE\_BACKEND | local |  |
| BASE\_PUBLIC\_URL | <http://localhost:8000> | serves /static |
| LOCAL\_UPLOAD\_DIR | ./uploads | local storage root |
| S3\_BUCKET/REGION/KEYS | — | only if STORAGE\_BACKEND=s3 |
|  |  |  |

8) Validation

* CSV headers: "S. No.", "Product Name", "Input Image Urls"
* Input Image Urls: comma-separated absolute URLs

9) Security

Limits, sanitization, axios limits/timeouts, CORS, optional webhook HMAC

10) Postman

Upload CSV, Status by requestId, Webhook test receiver