# Low-Level Design (LLD) Document for Image Processing System

## 1. Introduction

This document outlines the low-level design (LLD) for an image processing system involving two microservices:  
1. UploadAndFileStatusService  
2. ImageProcessingService  
  
The system enables users to upload an Excel file with image URLs. The UploadAndFileStatusService processes this data and enqueues jobs, while the ImageProcessingService picks jobs from the queue, compresses images, and updates the status in MongoDB via a webhook.

## 2. System Architecture

## 2.1 Components Overview

1. UploadAndFileStatusService:  
 - Accepts Excel file uploads.  
 - Parses data and stores initial metadata in MongoDB, marking the initial status as 'Pending'.  
 - Pushes job data into a Redis queue (BullMQ).  
 - Exposes a webhook for updating job statuses.  
  
2. ImageProcessingService:  
 - Consumes jobs from the queue.  
 - Compresses images to 50%.  
 - Calls the webhook to update the status and processed URLs.  
  
3. Message Queue (Redis - BullMQ):  
 - Acts as the job queue for processing requests asynchronously.  
  
4. MongoDB:  
 - Stores metadata, status, and processed URLs.

## 3. Sequence Flow

## 3.1 Upload and Processing Flow

1. User uploads an Excel file to UploadAndFileStatusService.  
2. UploadAndFileStatusService parses the file and extracts image URLs.  
3. It stores metadata in MongoDB (status: Pending).  
4. It pushes job messages to the queue.  
5. ImageProcessingService picks jobs from the queue.  
6. It fetches the image from the URL and compresses it.  
7. It uploads the compressed image to a storage service.  
8. It calls the webhook in UploadAndFileStatusService with the updated status and new URLs.  
9. UploadAndFileStatusService updates MongoDB with the processed URL and status.

## 4. Detailed Design

## /api/v1/upload/fileUpload

\*\*Payload:\*\* Accepts an Excel file.  
  
\*\*Response:\*\*  
```  
{  
 "success": true,  
 "status": 200,  
 "data": {  
 "requestId": "46289214-e138-453c-9309-65d8a3fa9617"  
 }  
}  
```

## /api/v1/status/fileStatus

\*\*Payload:\*\*  
```  
{  
 "requestId": "46289214-e138-453c-9309-65d8a3fa9617"  
}  
```  
  
\*\*Response:\*\*  
```  
{  
 "success": true,  
 "status": 200,  
 "data": {  
 "status": "Failed"  
 }  
}  
```

## /api/v1/webhook/statusUpdateWebhook

\*\*Payload:\*\* {requestId: { type: String, required: true, unique: true },  
 Status : String , ,  
 inputCsvFileData: [  
 {  
 serialNumber: String,  
 productName: String,  
 inputImages: [String],  
 requestId: { type: String }  
 }  
 ]}  
  
\*\*Response:\*\*  
```  
{ "result": "Status Updated Successfully" }  
```

## 4.2 MongoDB Schema

```javascript  
const mongoose = require('mongoose');  
const ImageProcessSchema = new mongoose.Schema({  
 requestId: { type: String, required: true, unique: true },  
 productName: String,  
 inputCsvFileData: [  
 {  
 serialNumber: String,  
 productName: String,  
 inputImages: [String],  
 requestId: { type: String }  
 }  
 ],  
 outputCsvFileData: [  
 {  
 serialNumber: String,  
 productName: String,  
 inputImages: [String],  
 requestId: { type: String }  
 }  
 ],  
 status: {  
 type: String,  
 enum: ['Pending', 'Processing', 'Completed', 'Failed'],  
 default: 'Pending'  
 }  
});  
```

## 4.3 ImageProcessingService

1. \*\*Queue Consumer\*\*  
 - Listens for new job messages.  
 - Compresses images.  
 - Calls the webhook with the update.  
  
2. \*\*Compression Algorithm\*\*  
 - Uses an image compression library (Sharp).  
 - Ensures the output file size is reduced to 50%.

## 5. Communication Flow Diagram

UploadAndFileStatusService --> Job Queue (Redis - BullMQ) --> ImageProcessingService  
 

## 8. Conclusion

This LLD outlines a scalable and modular design for handling bulk image processing through microservices, a queue system, and a webhook mechanism for status updates.