**My Image Processing System**

I built a simple image processing system using Python, FastAPI, and SQLAlchemy. I’ll describe what I did, how the code is structured, and how each part works. I also completed a bonus webhook flow that triggers after all images are processed.

Project Overview:

* Language and Framework: Python (FastAPI)
* Database: SQLAlchemy with a SQL database (for example, PostgreSQL)
* Image Processing: Requests + Pillow (PIL) for downloading and compressing images
* Asynchronous Tasks: FastAPI background tasks
* Webhook Flow: Optional callback URL triggered upon completion
* Tracking: I store each request’s status and timestamps in a ProcessingRequest record, including completed\_at

Folder and File Structure:

1. app/main.py
   * Creates the FastAPI application.
   * Mounts a static directory (processed\_images) so I can serve compressed images locally.
   * Includes a root endpoint (/) for a welcome message.
2. app/config.py
   * Loads environment variables (like the database URL).
3. app/database.py
   * Creates a SQLAlchemy engine and session maker to handle database connections.
4. app/models.py
   * Defines two models:
     + ProcessingRequest: Tracks each CSV upload, including request\_id, status, created\_at, completed\_at, and an optional callback\_url.
     + Product: Stores each row from the CSV, including serial\_number, product\_name, input\_image\_urls, and output\_image\_urls.
5. app/routes.py
   * Holds the API endpoints:
     + POST /upload:
       - Accepts a CSV file and an optional webhook\_url.
       - Validates the CSV header and each row.
       - Creates a ProcessingRequest record and a Product record for each CSV row.
       - Schedules a background task to download, compress, and store images.
     + GET /status/{request\_id}:
       - Returns the status of the request, along with all products.
       - Shows created\_at, completed\_at, and callback\_url if set.
       - If the request is complete, includes a link to download a final CSV with both input and output URLs.
     + GET /download/{request\_id}:
       - Lets me download a CSV with columns for S. No, Product Name, Input Image Urls, and Output Image Urls.

How Image Processing Works:

In the background task, I:

1. Look up all products for a given request\_id.
2. Download each image using a library that can fetch URLs.
3. Open and compress each image to 50% quality, saving it in a local processed\_images folder.
4. Update each Product record with new local URLs for the processed images.
5. Mark the ProcessingRequest as “completed,” set completed\_at to the current time, and save these changes in the database.

Bonus: Webhook Flow:

I also implemented an optional webhook flow. When I upload the CSV, I can provide a webhook\_url. Once all images are processed, I:

1. Check if the ProcessingRequest has a callback\_url.
2. If it does, I send a POST request to that URL with a JSON payload containing the request\_id and a status of “completed.”
3. This allows any external service to be notified immediately when my image processing finishes.

Testing the System:

1. Run the Server
   * I start FastAPI with a command that launches the server, usually on <http://localhost:8000>.
2. Upload a CSV
   * I make a POST request to /upload (using form-data) with:
     + file = the CSV file.
     + webhook\_url = (optional) a URL to notify when processing completes.
   * The response returns a unique request\_id.
3. Check Status
   * I do a GET request to /status/<request\_id>.
   * It shows the current status, plus created\_at, completed\_at, and callback\_url if provided.
   * It also lists all products with their input and output URLs.
4. Download the Output CSV
   * Once processing is finished, I see an output\_csv\_link.
   * Visiting that link downloads a CSV containing both input and output URLs.

Conclusion:

I built this system to handle CSV uploads of product data, download and compress images by 50%, and store them locally. I also store completed\_at and allow an optional webhook flow. The final CSV is available to download, and any external service can be notified automatically upon completion. This approach keeps everything simple by using FastAPI’s background tasks along with Pillow for image compression and a SQL database for tracking.