https://gemini.google.com/u/1/app/431cc17871bb821b

⚙️ Part 3: Recommended Image Processing Pipeline (High-Level)

-------------------------------------------------------------

This outlines a conceptual pipeline for processing uploaded images and generating the variants defined in the manifest. This would typically be implemented as a backend service or a series of serverless functions (e.g., Supabase Edge Functions).

Trigger:

- An image is successfully uploaded to Supabase Storage.

- A new record is created in the `public.media` table with basic metadata (original path, uploader, mime type, etc.). `media\_status` might be `'processing\_upload'`.

Pipeline Steps:

1. Queue & Intake:

- The new `media` record (or a message with its ID and original storage path) is added to a processing queue (e.g., Supabase Functions, RabbitMQ, SQS) to decouple processing from the upload request.

2. Validation & Initial Metadata Extraction (Worker Process):

- Retrieve original image from Supabase Storage.

- Validate file type and integrity. Check against any platform limits (max dimensions, file size).

- Extract metadata:

- Original dimensions (`image\_width\_px\_original`, `image\_height\_px\_original`).

- (Optional) Dominant color (`dominant\_color\_hex`).

- (Optional) SHA256 checksum (`checksum\_sha256\_original`) if not done at upload.

- Update the corresponding `public.media` record with this extracted metadata.

3. Variant Generation:

- For each relevant definition in the "Image Variant Manifest":

- Resize/Crop: Apply transformations (e.g., resize to width, scale to fit, center-crop) based on manifest rules for the current variant.

- Reformat: Convert to target format(s) (e.g., WebP, optimized JPG/PNG).

- Optimize: Apply compression. Adjust quality settings to balance file size and visual fidelity.

- Naming: Use a consistent naming convention for variant files (e.g., `media\_uuid/variant\_key.webp`).

4. Store Variants:

- Upload each generated variant to a designated "variants" path in Supabase Storage (e.g., `gs://[bucket\_name]/variants/[original\_media\_id]/[variant\_key].webp`).

5. Update `public.media` Record:

- Populate the `image\_variants\_json` field in the `public.media` record. The JSON should store key-value pairs where the key is the "Variant Key" from the manifest (e.g., `hero\_lg\_16x9\_webp`) and the value is the storage path or a retrievable identifier for that variant.

- Example `image\_variants\_json`:

JSON

```

{

"hero\_lg\_16x9\_webp": "variants/your-media-uuid/hero\_lg\_16x9.webp",

"hero\_md\_16x9\_webp": "variants/your-media-uuid/hero\_md\_16x9.webp",

"card\_std\_4x3\_jpg": "variants/your-media-uuid/card\_std\_4x3.jpg"

}

```

- Update `media\_status` in `public.media` to `'pending\_review'` or `'published\_approved'` depending on workflow.

6. Cleanup & Error Handling:

- Log all successes and failures.

- Implement retry mechanisms for transient errors.

- If processing fails critically, update `media\_status` to `'error\_uploading'` and notify administrators.

- (Optional) Delete original from temporary processing location if applicable.

Tools & Technologies (Suggestions):

- Image Processing Libraries:

- Node.js: Sharp (highly performant, uses libvips)

- Python: Pillow, Wand (ImageMagick bindings)

- Server-side: ImageMagick, GraphicsMagick

- Backend/Functions:

- Supabase Edge Functions (for lighter tasks or orchestrating external services).

- Dedicated backend service (e.g., Node.js, Python, Go) for more intensive processing.

- Queueing: If handling high volume, a message queue (RabbitMQ, Kafka, Google Pub/Sub, AWS SQS) can manage processing tasks. Supabase's built-in function invocation might be sufficient for moderate loads.

This pipeline ensures that original images are processed into a standardized set of optimized variants, ready for performant delivery to frontend applications, which can then request the appropriate variant based on UI context and device.