NextGate Media Delivery — Redesign

Target: a safe, scalable media layer for the feed, modelled on patterns observed in TikTok's delivery (signed expiring URLs, CDN edge, variant matrix, batch minting) and mapped onto NextGate's existing media engine (MinIO + RabbitMQ + workers + Redis).

---

1. What's wrong today

Observed in the current `GET /posts` (feed) response.

#	Problem	Why it matters
1	`originalUrl` is a raw, permanent, **unsigned** link to `s3.nexgate.co`	Post visibility is enforced at the API only. A PRIVATE post's bytes stay reachable at a permanent public URL → media bypasses access control entirely.
2	Storage origin exposed, no CDN	Every view hits MinIO directly. High latency from East Africa, full egress cost, zero edge cache.
3	`thumbnailUrl: null`, `placeholderBase64: null`	Full-res images (e.g. 1536×2048) served into feed thumbnail slots. Heavy on expensive mobile data. Blurhash field exists but unused.
4	Video = raw `.mp4`, `width/height/duration` = null	No transcode, no poster frame, no ladder, no HLS. No adaptive playback.
5	Per-user path prefix doesn't match owner	e.g. joshdoe's avatar stored under eddiesmith19's `stg-acc-{uuid}` prefix. Leaks UUIDs and misattributes ownership.
6	`dev.s3.nexgate.co` leaking into staging data	Environment bleed in a live feed response.

What to KEEP (already good): batch-hydrated feed (one call, all posts + nested quotes + engagement), permanent `shareCode` short IDs, graceful deleted-post handling (`unavailable: true`), the rich post data model.

The fix is scoped to the **media delivery layer**, not the data model.

2. Core principle

Two URLs, two lifespans (same split TikTok uses):

- **Permanent identity** — the post `id` / `shareCode` and the storage `objectKey`. Stored in DB. Shareable forever.
- **Ephemeral delivery ticket** — a signed, expiring CDN URL. Minted on every feed build. Never stored.

Rule: **store the `objectKey`, never store the signed URL.**

---

3. Three delivery tiers

Tier	Content	Auth gate?	URL signing	Expiry	CDN cache
1 — public asset	avatars, thumbnails	no	none (or very long)	n/a / long	aggressive, long TTL
2 — public content, protected delivery	public post images/video	no	yes (short-ish)	hours	careful (see §6)
3 — private content	private/followers posts, paid digital	yes (ownership/visibility check)	yes	minutes	none / private

The only code difference between tier 2 and tier 3 is **whether the permission check runs before minting**. Same minting call underneath.

---

4. Storage layer changes

4.1 Buckets

- `public` — tier 1 assets (avatars, generated thumbnails)
- `private` — tier 2 + tier 3 post media
- `digital` — paid products (tier 3, strictest)

Shared buckets, **owner-prefixed object keys**, prefix must match the real owner. Opaque object keys — no user UUID needed in the public URL.

```
private/{ownerId}/{mediaId}/original.jpg
private/{ownerId}/{mediaId}/thumb_320.webp
private/{ownerId}/{mediaId}/thumb_720.webp
```

4.2 DB: store keys, not URLs

Persist the storage coordinates and processing state, not a full URL.

```
// ── MediaEntity (delivery-relevant fields) ──
private String bucket;          // "private"
private String objectKey;       // "ownerId/mediaId/original.jpg"  (opaque, owner-correct)
private MediaType mediaType;    // IMAGE | VIDEO
private Integer width;
private Integer height;
private Double  duration;       // video only
private String  placeholder;    // blurhash / LQIP — fills the empty placeholderBase64
private MediaStatus status;      // PROCESSING | READY | FAILED
// variants: thumb sizes, video rungs, formats — JSONB
private String variantsJson;     // {"thumb":["320","720"],"formats":["webp"],"rungs":[...]}
```

---

5. Delivery layer changes

5.1 CDN in front of MinIO

Serve everything through `media.nexgate.co` (Cloudflare) → MinIO origin. **Never** return `s3.nexgate.co` (or `dev.s3…`) in an API response again.

5.2 Minting helper (SDK does the crypto)

```
// ── Presigned URL minting (MinIO SigV4) ──
public String mintUrl(String bucket, String objectKey, int seconds) {
    return minioClient.getPresignedObjectUrl(
        GetPresignedObjectUrlArgs.builder()
            .method(Method.GET)
            .bucket(bucket)
            .object(objectKey)
```

```
            .expiry(seconds, TimeUnit.SECONDS)   // → X-Amz-Expires + X-Amz-Signature
            .build());
    // returned host is rewritten to media.nexgate.co at the edge / via MinIO public endpoint
}
```

5.3 Feed response shape (mint at build time, batch)

Mirror TikTok's batch mint: build the page, mint every media URL inline, ship once. No per-item round trip from the client.

```
"media": [{
  "id": "014ab4eb-...",
  "mediaType": "IMAGE",
  "url": "https://media.nexgate.co/.../original.jpg?X-Amz-Expires=3600&X-Amz-Signature=...",
  "thumbnailUrl": "https://media.nexgate.co/.../thumb_720.webp?...sig...",
  "placeholderBase64": "LEHV6nWB2yk8...",   // blurhash, no longer null
  "width": 945, "height": 2048,
  "expiresIn": 3600
}]
```

```
// ── In the feed mapper ──
String url   = mintUrl(m.getBucket(), m.getObjectKey(), tier2Seconds);          // hours
String thumb = mintUrl(m.getBucket(), thumbKey(m, 720),  tier2Seconds);
// tier 3 (private/paid): run accessService.canAccess(user, m) first, use short expiry
```

---

6. CDN + signed-URL gotcha

Do not let the CDN cache a per-user signed URL under a key that includes the signature, or one user's link can be served to another. Options:

- cache key = path only (strip query/signature), short edge TTL for tier 2; or
- tier 1 (avatars/thumbs) = truly public objects, cache hard; tier 2/3 originals = bypass shared cache or use very short edge TTL.

Tier 1 assets are the ones you cache aggressively. Tier 3 never shares cache.

---

7. Variant + transcode pipeline (reuse existing media engine)

On upload (presigned PUT direct to MinIO → enqueue job → workers → webhook back):

- **ClamAV** scan first; reject on hit.
- **Image worker**: generate `thumb_320` + `thumb_720` (WebP), compute blurhash → `placeholder`, read width/height. Write variants to JSONB, set status READY.
- **Video worker**: extract poster frame (tier-1 thumbnail), read width/height/duration, transcode to 1–2 MP4 rungs now (e.g. 540p + 720p), HLS later when analytics/scale justify it.
- **Redis** tracks job state; **webhook** flips `MediaEntity.status` PROCESSING → READY.

Until a media row is READY, the feed returns the blurhash placeholder + a `processing` flag instead of a broken/raw link.

---

8. Migration sequence (non-breaking)

1. **CDN** — stand up `media.nexgate.co` → MinIO, dual-serve. Stop emitting `s3.nexgate.co`.
2. **Schema** — add `bucket` / `objectKey` / `status` / `placeholder` / `variantsJson`. Backfill `objectKey` by parsing existing `originalUrl`; fix owner prefix + drop `dev.` rows here.
3. **Mint** — change feed to mint from `objectKey` instead of returning raw `originalUrl`. Roll out tier 1 + tier 2 first (public) so nothing visibly breaks.
4. **Backfill variants** — run image/video workers over existing media; populate thumbs + blurhash.
5. **Lock the bucket** — remove public-read from MinIO; force all access through signed CDN URLs. At this point the visibility-bypass hole (problem #1) is closed.
6. **Tier 3** — gate private/followers/paid media behind `canAccess` + short expiry.

Closing problem #1 happens at step 5; everything before it is safe groundwork.

---

9. Security checklist ("safe" part)

- ☐ No storage origin host (`s3.nexgate.co`, `dev.s3…`) in any API response.
- ☐ No user UUID in public URLs (opaque object keys).
- ☐ Private/followers post media served tier 3 (auth gate + short signed URL).
- ☐ MinIO buckets are **not** public-read once migration completes.
- ☐ Owner prefix on every key matches the real owner.

☐ Dev/staging/prod storage hosts strictly separated; no cross-env URLs.

Revision #1
Created 29 May 2026 07:58:28 by Admin Qbit
Updated 29 May 2026 07:58:43 by Admin Qbit