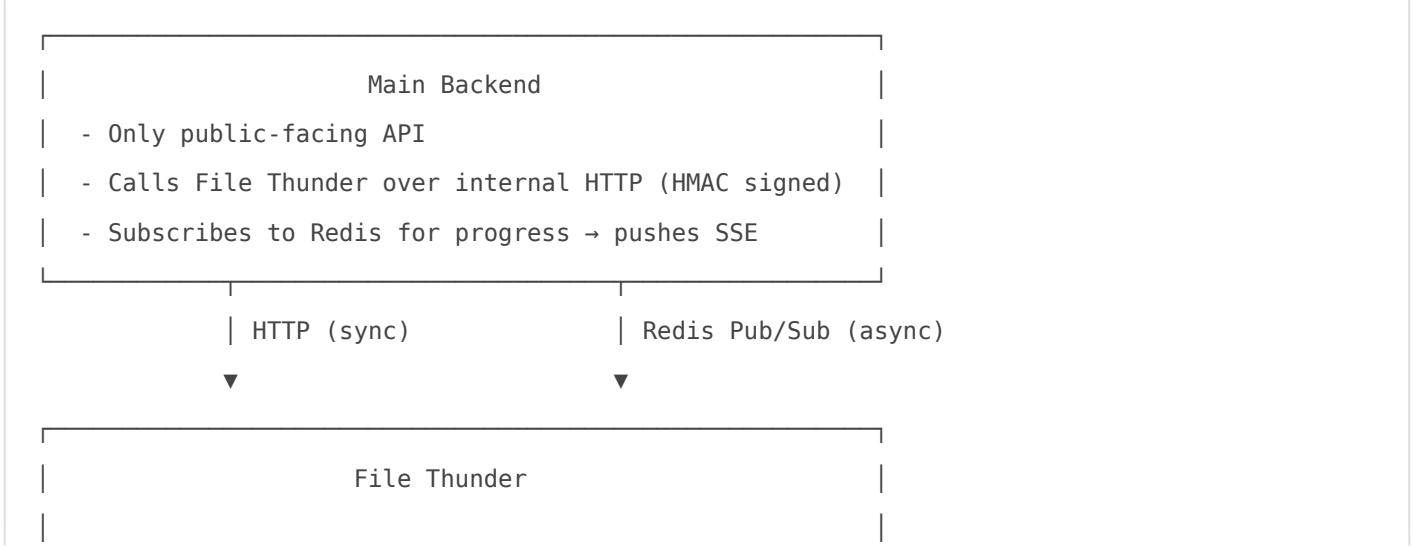
File Thunder Arch & Flow

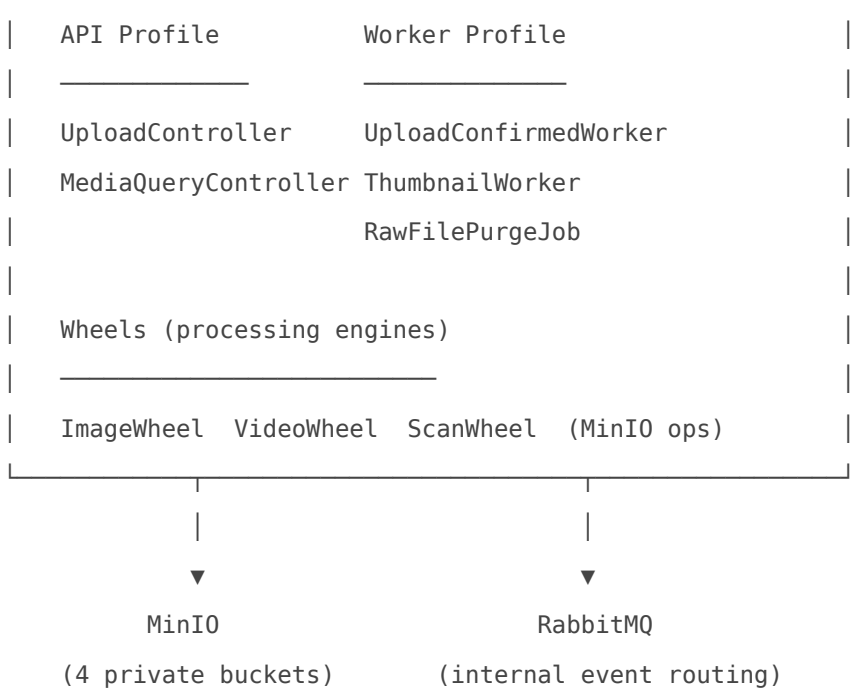
File Thunder is the dedicated media-processing microservice for the NexGate / Veepii platform. It handles all file ingestion, processing, storage, and serving-readiness — the main backend never touches raw bytes.

Table of Contents

---

1. Architecture Overview





Key constraints:

- File Thunder is **internal only** — never exposed to the internet directly
- Main backend is the **only caller** of File Thunder HTTP APIs
- File Thunder **never touches file bytes on upload** — client uploads directly to MinIO via presigned URL
- Redis is **FT-internal** — used for progress pub/sub and nonce replay protection
- RabbitMQ is **FT-internal** — used to route events between listeners and workers

---

2. Storage: Buckets & Object Keys

Buckets

Bucket	Purpose	Access
`nexgate-raw`	Temporary upload landing zone — 24h TTL	Private
`nexgate-public`	Processed media served to end users	Private (CDN in front)
`nexgate-private`	Internal system assets (outros, future forensic assets)	Private
`nexgate-digital`	Digital product files — ClamAV scanned, download only	Private

All buckets are **fully private**. MinIO is never directly accessible from the internet. Public content is served exclusively through the CDN (Cloudflare), which pulls from MinIO origin.

Object Key Pattern

```
{domain}/{ownerId}/{fileId}/{variant}
```

Examples:

```
posts/550e8400-e29b-41d4-a716-446655440000/f7e8d9.../original
posts/550e8400-e29b-41d4-a716-446655440000/f7e8d9.../large.webp
posts/550e8400-e29b-41d4-a716-446655440000/f7e8d9.../thumb.webp
posts/550e8400-e29b-41d4-a716-446655440000/f7e8d9.../hls/master.m3u8
posts/550e8400-e29b-41d4-a716-446655440000/f7e8d9.../hls/360p/360p.m3u8

system/outro/{ownerId}/{height}p.mp4   ← in nexgate-private
```

Rule: The database stores **object keys only**, never full URLs. URLs are assembled at the resolver boundary (CDN base URL + key in prod, presigned GET in local).

---

3. MediaDomain & MediaContext

MediaDomain

Defines **where** in the storage hierarchy a file lives. It is the top-level folder in the object key. This is an enum — the main backend must send one of these exact values.

Value	Used For
`POSTS`	Social posts — images and videos
`PROFILES`	Profile pictures, cover photos
`MESSAGES`	Direct message attachments
`PRODUCTS`	Product images, product videos, digital downloads
`SHOPS`	Shop banners, shop logos
`EVENTS`	Event covers, event gallery images

MediaContext

Defines **how** a file is processed. This drives the entire processing pipeline. No logic is attached to domain — all logic is driven by context.

Value	Type	Processing Pipeline
`SOCIAL_IMAGE`	Image	ImageWheel — orient, strip EXIF, WebP variants, blurhash, lqip
`SOCIAL_VIDEO`	Video	VideoWheel — transcode + **social watermark + outro** + thumbnail
`PROFILE_PICTURE`	Image	ImageWheel
`COVER_PHOTO`	Image	ImageWheel
`DM_IMAGE`	Image	ImageWheel
`DM_VIDEO`	Video	VideoWheel — transcode, no watermark
`DM_DOCUMENT`	Any	ScanWheel — ClamAV scan
`PRODUCT_IMAGE`	Image	ImageWheel
`PRODUCT_VIDEO`	Video	VideoWheel — transcode + text watermark, no outro
`DIGITAL_PRODUCT`	Any	ScanWheel — ClamAV dual scan
`SHOP_BANNER`	Image	ImageWheel
`SHOP_LOGO`	Image	ImageWheel
`EVENT_COVER`	Image	ImageWheel
`EVENT_GALLERY`	Image	ImageWheel

Validation rules enforced at API layer:

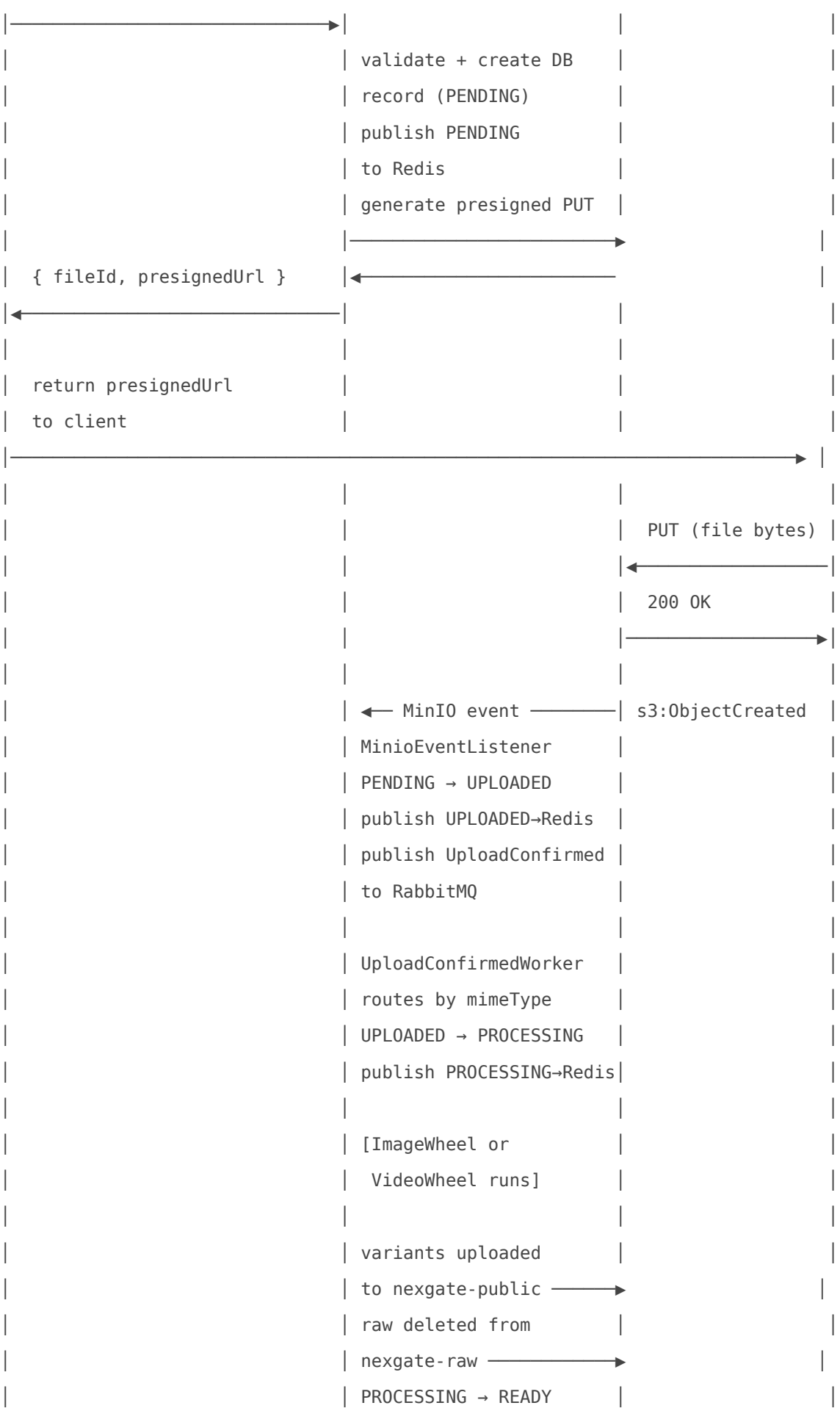
- Image contexts only accept image MIME types; video contexts only accept video MIME types
- `SOCIAL_VIDEO` requires `username` field (used for personalized watermark + outro)
- Max file sizes: images 20 MB, videos 2 GB, digital products 500 MB
- HEIC/HEIF are rejected — parser attack surface too high

---

4. Upload Flow — End to End

Standard Media Upload

```
Main Backend                    File Thunder                MinIO          Client (Browser)
     |                               |                        |                 |
     |  POST /api/v1/upload/request  |                        |                 |
     |  (HMAC signed)                |                        |                 |
```


validate + create DB
record (PENDING)
publish PENDING
to Redis
generate presigned PUT
{ fileId, presignedUrl }
return presignedUrl
to client
PUT (file bytes)
200 OK
MinIO event
s3:ObjectCreated
MinioEventListener
PENDING → UPLOADED
publish UPLOADED→Redis
publish UploadConfirmed
to RabbitMQ
UploadConfirmedWorker
routes by mimeType
UPLOADED → PROCESSING
publish PROCESSING→Redis
[ImageWheel or
VideoWheel runs]
variants uploaded
to nexgate-public
raw deleted from
nexgate-raw
PROCESSING → READY

```
     |                              | publish READY → Redis   |                  |
     |                              |                         |                  |
```

User Custom Thumbnail Upload

```
Main Backend                    File Thunder                MinIO
     |                              |                         |
     |  POST /api/v1/upload/        |                         |
     |  thumbnail/{fileId}          |                         |
     |─────────────────────────────►|                         |
     |                              | generate presigned PUT  |
     |                              | key: .../custom-        |
     |                              | thumbnail               |
     |  { presignedUrl }            |                         |
     |◄─────────────────────────────|                         |
     |  client uploads thumbnail    |                         |
     |─────────────────────────────────────────────────────►  |
     |                              |◄── MinIO event ─────────|
     |                              | MinioEventListener      |
     |                              | detects suffix=         |
     |                              | "custom-thumbnail"      |
     |                              | → ThumbnailWorker       |
     |                              | download → verify magic |
     |                              | bytes → ImageMagick     |
     |                              | variants → save to      |
     |                              | entity.userThumbnail    |
     |                              | → delete raw            |
```

File Status Lifecycle

```
PENDING → UPLOADING → UPLOADED → SCANNING → PROCESSING → LIVE_PARTIAL → READY
                                                              ▲
                                                  (short video: after 360p uploaded)
                                                  (long video: after 360p + partial HLS)
```

---

5. The Four Wheels

Wheels are the processing engines. Each runs in the worker profile.

Wheel 1 — ImageWheel (ImageMagick)

Triggered by: image MIME type on any image context

Pipeline:

```
download raw from nexgate-raw
      │
      ▼
auto-orient + strip EXIF
      │
      ├──► large.webp    (1920px max width, shrink only, Q85)
      ├──► medium.webp   (800px max width, shrink only, Q82)
      ├──► thumb.webp    (300px max width, shrink only, Q80)
      ├──► og.webp       (1200×630 center crop, Q85)
      │    └─ falls back to medium if source width < 1200px
      ├──► blurhash      (encoded from 32×32 downsample)
      └──► lqip          (10×10 forced, base64 WebP data URI)

all WebP variants → nexgate-public
keys + blurhash + lqip → media_files.variants (JSONB)
raw deleted from nexgate-raw
status → READY
```

Skipping logic: variant is skipped if source is already within bounds (never upscale).

Wheel 2 — VideoWheel (FFmpeg / Jaffree)

Triggered by: video MIME type on any video context

Step 1 — Probe & Validate

```
download raw from nexgate-raw
magic byte verify (must be real video container)
FFprobe → codedWidth, codedHeight, rotation, duration, codec
displayWidth/displayHeight = rotation-aware (swap for 90°/270°)
size gate: > 2 GB → reject
```

```
duration gate: > 4h → reject
route: duration < 3min → processShort()
       duration ≥ 3min → processLong()
```

Step 2a — Short Path (< 3 min)

```
transcode() runs 3 variants in sequence:
  360p  (CRF 28, 96k audio)  → upload → status LIVE_PARTIAL
  720p  (CRF 23, 128k audio) → upload
  1080p (CRF 21, 192k audio) → upload

filter_complex:
  [0:v] split=2 [main][blur]
  [blur] scale={W}:{H}, boxblur=20:5 [bg]
  [main] scale={w}:{h} (fit within target, keep AR) [fg]
  [bg][fg] overlay=x=(W-w)/2:y=(H-h)/2 [out]
  + rotation transpose prepended if needed
  + watermark drawtext chain appended

if SOCIAL_VIDEO:
  append outro via concat (no re-encode)

skip logic: source displayWidth >= targetW OR displayHeight >= targetH
```

Step 2b — Long Path (≥ 3 min) — HLS Adaptive

```
transcodeHls() per variant:
  360p → segments + .m3u8 → upload → partial master.m3u8 → LIVE_PARTIAL
  720p → segments + .m3u8 → upload
  1080p → segments + .m3u8 → upload → full master.m3u8 → READY

segment length: 2s
playlist type: VOD
master.m3u8 key: {keyBase}/hls/master.m3u8
variant keys:   {keyBase}/hls/{name}/{name}.m3u8
```

Step 3 — Thumbnail & Preview (both short and long)

```
selectBestFrame():
  5 candidates at 15/30/45/60/75% of duration
  scored by: brightness gate (30–230) + Laplacian variance (sharpness)
```

```
  FFmpeg extracts 720px JPEG per candidate → pick winner

buildThumbnailVariants():
  poster.webp      (1280px, Q85)
  thumb.webp       (480px, Q80)
  og.webp          (1200×630 center crop, Q85)
  blurhash         (from 32×32 downsample)
  lqip             (10×10 forced, base64 WebP data URI)
  dominant_color   (#RRGGBB from 1×1 squish)

extractPreviewClip():
  skip first 5% (max 2s) → read 6s → setpts=0.5*PTS → 3s at 2× speed
  blur-pad 360×640, muted
  watermarked (same moving watermark as main video)

all thumbnail variants → nexgate-public
raw deleted from nexgate-raw after all variants done
```

Wheel 3 — ScanWheel (ClamAV)

Triggered by: `DIGITAL_PRODUCT` and `DM_DOCUMENT` contexts only Social content (images and videos) is **never** scanned by ClamAV — FFmpeg/ImageMagick re-encode is the sanitisation.

Pipeline:

```
download raw from nexgate-raw
      │
      ▼
SHA-256 hash → check file_hashes table
      │
      ├── hash known + clean → skip scan, copy to nexgate-digital, READY (dedup fast lane)
      │
      └── hash unknown →
            ClamAV scan #1 (upload scan)
            if clean → move to nexgate-digital
            ClamAV scan #2 (pre-download scan, confirms in-transit integrity)
            if clean → save hash, status READY
            if infected → status FAILED, quarantine
```

Wheel 4 — MinIO Operations (used by all wheels)

Not a standalone service — MinIO operations are helpers used across all wheels:

Operation	Used By
Presigned PUT URL	Upload request endpoint
Download object	All wheels (download raw for processing)
Upload object	All wheels (upload processed variants)
Remove object	All wheels (delete raw after processing)
Stat object	OutroService (cache check), ScanWheel (dedup check)

---

6. Progress Tracking (Redis ? SSE)

File Thunder publishes status changes to Redis. The main backend subscribes and drives SSE to the client. **SSE is the main backend's responsibility — File Thunder only publishes.**

What File Thunder Does

On every status change:

```
// 1. Cache current status in Redis (24h TTL)
redisTemplate.opsForValue().set("ft:status:{fileId}", status.name(), 24h);

// 2. Publish to per-file channel
redisTemplate.convertAndSend("ft:progress:{fileId}", status.name());

// 3. Append to timeline in Postgres
entity.timeline.add({ status, at })
```

What the Main Backend Must Do

```
// Subscribe to the file's progress channel
redisTemplate.subscribe((message, pattern) -> {
    String status = message.toString();
```

```
    sseEmitter.send(SseEmitter.event().data(status));
}, "ft:progress:" + fileId);
```

Status Channel Key

```
ft:progress:{fileId}     ← subscribe here for live updates
ft:status:{fileId}       ← read here for current status (24h cached)
```

---

7. Watermarking

Moving Watermark

Applied to all video variants and preview clips. Watermark position cycles through 4 corners every 3 seconds (5 seconds for social).

Text-only (PRODUCT_VIDEO):

```
drawtext: fontsize=max(14,H/40), white@0.65 + black shadow
position: floor(t/3) mod 4 → top-left, top-right, bottom-right, bottom-left
text: from watermark.text property (default: "NexGate")
```

Social watermark (SOCIAL_VIDEO):

```
2-point diagonal: Pos A (22%, 28%) ↔ Pos B (58%, 65%) — switches every 5s
Logo: nexgate_logo_white.svg, 36×36, 65% opacity
@ symbol: orange #F06023@0.85 + shadow
username: white@0.85 + shadow, below logo
```

Outro (SOCIAL_VIDEO only)

A personalized outro clip is appended to the watermarked variant (not clean variants).

Generation:

```
template: outro_template_with_sfx.mp4 (classpath resource)
font: Sora SemiBold 600 (classpath resource)
drawtext: @ in orange #F06023, username in cream #F4EEE9
fade-in: invisible before t=0.75s, fully opaque by t=1.2s
```

```
scale to match variant resolution (360p / 720p / 1080p)
append via: -f concat -safe 0 -c copy (no re-encode, fast)
```

Caching:

```
Generated once per ownerId + resolution
Cached to: nexgate-private / system/outro/{ownerId}/{height}p.mp4
On next upload: cache hit → download and reuse, skip generation
```

---

8. CDN & File Serving Per Environment

All MinIO buckets are private. Access to public content is environment-dependent.

Environment Matrix

Environment	Serving Method	Config Value
Local dev	Presigned GET URLs (MinIO direct, short-lived)	`ft.storage.mode=local`
Staging	CDN — `cdn-staging.nexgate.com`	`ft.storage.mode=cdn`
Production	CDN — `cdn.nexgate.com`	`ft.storage.mode=cdn`

Local / Dev

- App generates a presigned GET URL from MinIO (e.g. 1-hour expiry)
- URL is returned directly to the client
- MinIO must be reachable by the client
- Good enough for development and manual testing

Staging & Production (CDN)

- App assembles: `{ft.cdn.base-url} + "/" + objectKey`
- Cloudflare sits in front of MinIO origin
- Cloudflare pulls from MinIO on cache miss using a service credential
- Every subsequent request served from Cloudflare edge — MinIO never hit again
- Cost: Cloudflare bandwidth is free; MinIO only pays VPS bandwidth once per cache miss

Properties

```
# Local
ft.storage.mode=local
ft.cdn.base-url=

# Staging
ft.storage.mode=cdn
ft.cdn.base-url=https://cdn-staging.nexgate.com

# Production
ft.storage.mode=cdn
ft.cdn.base-url=https://cdn.nexgate.com
```

URL Assembly Logic (resolver boundary)

```
// local mode → presigned GET from MinIO
// cdn mode   → CDN base + object key
String url = storageMode.equals("cdn")
    ? cdnBaseUrl + "/" + objectKey
    : minioClient.getPresignedObjectUrl(GET, bucket, objectKey, 1h);
```

Private content (DIGITAL_PRODUCT, DM_DOCUMENT): always presigned GET URLs regardless of environment — these are never cached by CDN. Generated fresh per download request (15-minute expiry, audit logged).

---

9. Security

CORS

Configured via `ft.cors.*` properties. The CORS filter runs **before all other filters** so OPTIONS preflight requests are handled without requiring HMAC headers.

```
# Local
ft.cors.allowed-origins=http://localhost:3000

# Production
ft.cors.allowed-origins=https://nexgate.com,https://www.nexgate.com
```

```
ft.cors.allowed-methods=GET,POST,PUT,DELETE,OPTIONS
ft.cors.allowed-headers=*
ft.cors.allow-credentials=false
ft.cors.max-age-seconds=3600
```

Service-to-Service HMAC Authentication

Every HTTP request from the main backend to File Thunder must be HMAC-SHA256 signed. File Thunder rejects any unsigned or incorrectly signed request with `401 Unauthorized`.

Filter order:

```
Request → CorsFilter (HIGHEST_PRECEDENCE)
        → HmacAuthFilter (HIGHEST_PRECEDENCE + 1)
        → Controllers
```

Required Headers

Header	Description	Example
`X-Service-Id`	Identifier of the calling service	`nexgate-main`
`X-Timestamp`	Unix epoch seconds at time of request	`1718123456`
`X-Nonce`	Random UUID — unique per request	`f47ac10b-58cc-...`
`X-Signature`	HMAC-SHA256 hex of the canonical string	`a3f5c2...`

Canonical String

```
METHOD\n
REQUEST_URI\n
TIMESTAMP\n
NONCE\n
HEX(SHA-256(requestBody))
```

Example for `POST /api/v1/upload/request`:

```
POST
/api/v1/upload/request
1718123456
f47ac10b-58cc-4372-a567-0e02b2c3d479
e3b0c44298fc1c149afb...   ← SHA-256 of the JSON body
```

Signature Computation

```
// Main backend — how to sign a request
String bodyHash = HexFormat.of().formatHex(
    MessageDigest.getInstance("SHA-256").digest(requestBodyBytes)
);

String canonical = method + "\n" + uri + "\n" + timestamp + "\n" + nonce + "\n" + bodyHash;

Mac mac = Mac.getInstance("HmacSHA256");
mac.init(new SecretKeySpec(sharedSecret.getBytes(UTF_8), "HmacSHA256"));
String signature = HexFormat.of().formatHex(mac.doFinal(canonical.getBytes(UTF_8)));

// Set headers on outgoing request
request.setHeader("X-Service-Id", "nexgate-main");
request.setHeader("X-Timestamp",  String.valueOf(Instant.now().getEpochSecond()));
request.setHeader("X-Nonce",      UUID.randomUUID().toString());
request.setHeader("X-Signature",  signature);
```

What File Thunder Verifies (in order)

```
1. All 4 headers present               → 401 if any missing
2. X-Service-Id in allowed list        → 401 if unknown
3. X-Timestamp within ±5 minutes       → 401 if stale (replay protection)
4. X-Nonce not seen before             → 401 if duplicate
   (stored in Redis with 10-min TTL)      (replay attack blocked)
5. Recompute HMAC, timing-safe compare → 401 if mismatch
   (MessageDigest.isEqual)                (prevents timing oracle)
```

Properties

```
ft.security.hmac.secret=<long-random-secret-same-in-both-services>
ft.security.hmac.allowed-service-ids=nexgate-main
ft.security.hmac.timestamp-tolerance-seconds=300
ft.security.hmac.nonce-ttl-seconds=600
```

The `secret` must be identical in both File Thunder and the main backend config. Use a different value per environment (local / staging / prod).

---

10. API Reference & How to Consume

Base URL: `http://file-thunder:8081` (internal network only) All requests must include HMAC headers — see Section 9.

POST /api/v1/upload/request

Request a presigned URL to upload a file directly to MinIO.

Request body:

```
{
  "ownerId":          "550e8400-e29b-41d4-a716-446655440000",
  "domain":           "POSTS",
  "context":          "SOCIAL_VIDEO",
  "originalFilename": "my-video.mp4",
  "mimeType":         "video/mp4",
  "fileSizeBytes":    104857600,
  "username":         "josh_dev"
}
```

`username` is required when `context` is `SOCIAL_VIDEO`. All other fields are always required.

Response:

```
{
  "status": "success",
  "message": "Presigned upload URL generated",
  "data": {
    "fileId":           "f7e8d9fa-...",
    "presignedUrl":     "http://minio:9000/nexgate-raw/posts/.../original?X-Amz-...",
    "objectKey":        "posts/{ownerId}/{entityId}/{fileId}/original",
    "bucket":           "nexgate-raw",
    "expiresInSeconds": 1800
  }
}
```

What to do next: PUT the file bytes directly to `presignedUrl` from the client browser. No auth headers needed on the PUT — the presigned URL is self-authenticating.

POST /api/v1/upload/thumbnail/{fileId}

Request a presigned URL to replace the system-generated video thumbnail with a user-supplied one. Only valid after the video file has been processed (status READY).

Response:

```
{
  "status": "success",
  "message": "Thumbnail presigned upload URL generated",
  "data": {
    "fileId":           "f7e8d9fa-...",
    "presignedUrl":     "http://minio:9000/nexgate-raw/.../custom-thumbnail?X-Amz-...",
    "objectKey":        "posts/.../custom-thumbnail",
    "bucket":           "nexgate-raw",
    "expiresInSeconds": 1800
  }
}
```

Accepted formats: JPEG, PNG, WebP only (verified by magic bytes).

GET /api/v1/media/{fileId}

Get full metadata for a file, including all processed variants.

Response:

```
{
  "status": "success",
  "data": {
    "fileId":    "f7e8d9fa-...",
    "ownerId":   "550e8400-...",
    "domain":    "POSTS",
    "context":   "SOCIAL_VIDEO",
    "status":    "READY",
    "mimeType":  "video/mp4",
    "variants": {
      "360p":          "posts/.../360p.mp4",
      "720p":          "posts/.../720p.mp4",
      "1080p":         "posts/.../1080p.mp4",
```

```
      "poster":        "posts/.../poster.webp",
      "thumb":         "posts/.../thumb.webp",
      "og":            "posts/.../og.webp",
      "preview_3s":    "posts/.../preview_3s.mp4",
      "blurhash":      "LKO2?U%2Tw=w]~RBVZRi};RPxuwH",
      "lqip":          "...",
      "dominant_color":"#1A2B3C"
    },
    "userThumbnail": null,
    "timeline": [
      { "status": "PENDING",    "at": "2026-06-15T10:00:00" },
      { "status": "UPLOADED",   "at": "2026-06-15T10:00:05" },
      { "status": "PROCESSING", "at": "2026-06-15T10:00:06" },
      { "status": "LIVE_PARTIAL","at": "2026-06-15T10:00:45" },
      { "status": "READY",      "at": "2026-06-15T10:02:10" }
    ],
    "createdAt": "2026-06-15T10:00:00",
    "updatedAt": "2026-06-15T10:02:10"
  }
}
```

Thumbnail resolution rule:

```
userThumbnail != null ? use userThumbnail : use variants
```

Both `userThumbnail` and `variants` use the same key names (`poster`, `thumb`, `og`, `blurhash`, `lqip`, `dominant_color`).

GET /api/v1/media/{fileId}/download?requesterId={uuid}

Generate a time-limited download URL for private files (`DIGITAL_PRODUCT`, `DM_DOCUMENT` only). Every call is audit-logged.

Response:

```
{
  "status": "success",
```

```
  "data": {
    "url":              "http://minio:9000/nexgate-digital/...?X-Amz-...",
    "expiresInSeconds": 900
  }
}
```

GET /api/v1/quota/{ownerId}

Get the total storage used by an owner across all their processed files. Updated atomically as each file is processed — never stale.

Response:

```
{
  "status": "success",
  "message": "Storage usage",
  "data": {
    "ownerId":   "550e8400-e29b-41d4-a716-446655440000",
    "usedBytes": 1073741824,
    "usedMb":    1024.0,
    "usedGb":    1.0
  }
}
```

How quota is tracked:

- Every wheel (`ImageWheel`, `VideoWheel`, `ScanWheel`) calls `StorageUsageService.trackUsage(ownerId, bytes)` after uploading processed variants
- Tracked bytes = size of the **processed variants**, not the raw upload
- Stored in `user_storage_quota` table, upserted atomically per owner
- `releaseUsage()` is called on hard delete (soft deletes do not release quota)
- Returns `0` if the owner has no tracked usage yet (never uploaded)

Use this endpoint to:

- Display storage usage in the main backend dashboard
- Enforce storage limits before allowing a new upload request

---

Error Responses

```
{ "error": "Missing required security headers" }          ← 401
{ "error": "Request timestamp out of acceptable window" } ← 401
{ "error": "Nonce already used — possible replay attack"} ← 401
{ "error": "Invalid signature" }                          ← 401
```

Validation errors return 400 with field-level messages via `@ControllerAdvice`.

This document covers the full File Thunder system as of 2026-06-15. CDN wiring (Cloudflare), Kafka integration, lazy transcode, and forensic watermarks are deferred.

Revision #2
Created 15 June 2026 08:38:50 by Admin Qbit
Updated 15 June 2026 10:37:44 by Admin Qbit