Files Handling API (NEW)

Author: Josh S. Sakweli, Backend Lead Team **Last Updated**: 2026-06-21 **Version**: v1.0

Base URL: `https://your-api-domain.com/api/v1`

Short Description: This document covers everything a frontend developer needs to handle files on the NexGate/Veepii platform — uploading, tracking processing progress, reading variant URLs from entity responses, and downloading private files. Files are managed by an internal service called **FileThunder**; the frontend never talks to FileThunder directly. All file-related operations go through the main backend endpoints documented here.

Hints:

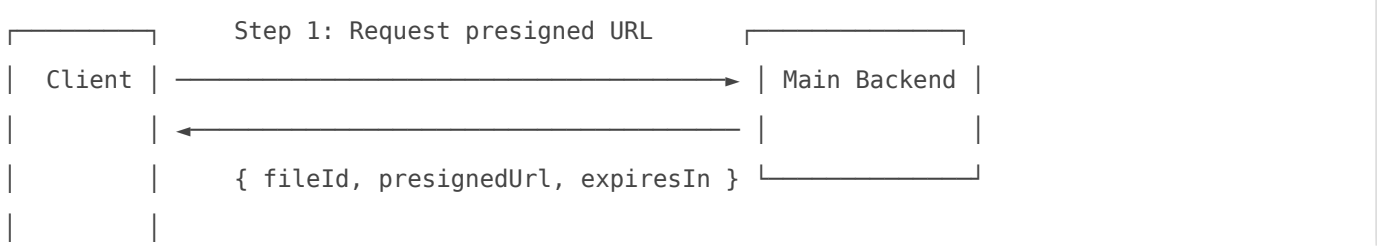
- All upload endpoints require a valid Bearer token
- Files are **never** uploaded through the backend server — you upload directly to object storage using a time-limited presigned URL returned by the backend
- The backend stores only object key paths; full CDN/MinIO URLs are assembled at response time — always use the URLs in entity responses as-is
- Processing is asynchronous — after uploading, you track progress via SSE or polling, then the entity response will include the resolved variant map once processing is complete
- Private files (digital products, DM documents) are never served via CDN — always request a fresh download URL before initiating a download

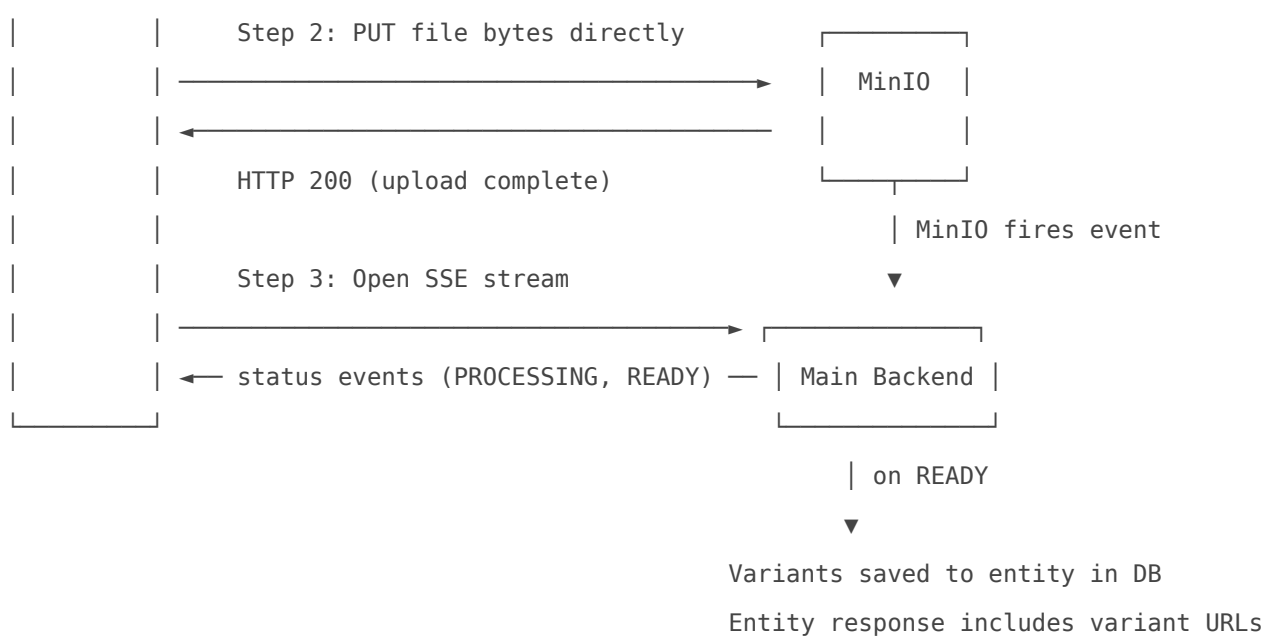
---

How File Handling Works — Big Picture

Understanding this flow will save you from confusion. There are **two separate progress concepts**:

1. **Upload progress** — bytes travelling from the client to object storage. You can show a progress bar for this using `XMLHttpRequest`.
2. **Processing progress** — FileThunder processing those bytes into variants (WebP images, transcoded video resolutions, virus scans). You track this via SSE or polling.





Standard Response Format

Success Response Structure

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Operation completed successfully",
  "action_time": "2025-09-23T10:30:45",
  "data": {}
}
```

Error Response Structure

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Error description",
  "action_time": "2025-09-23T10:30:45",
  "data": "Error description"
}
```

Standard Response Fields

Field	Type	Description
`success`	boolean	`true` for success, `false` for errors
`httpStatus`	string	HTTP status name (OK, BAD_REQUEST, NOT_FOUND, etc.)
`message`	string	Human-readable result description
`action_time`	string	ISO 8601 timestamp of the response
`data`	object/string	Response payload or error details

HTTP Method Badge Standards

- **GET** — GET
- **POST** — POST
- **PUT** — PUT

File Contexts

Every upload must declare a **context**. Context tells FileThunder what this file is for, which determines how it is processed, what variants are generated, and how it is stored.

Context	Domain	What it is	Who uploads it
`SOCIAL_IMAGE`	Posts	Image attached to a post or story	Authenticated user
`SOCIAL_VIDEO`	Posts	Video attached to a post or story	Authenticated user
`PROFILE_PICTURE`	Profiles	User avatar / profile photo	Authenticated user
`COVER_PHOTO`	Profiles	Profile cover / banner image	Authenticated user
`DM_IMAGE`	Messages	Image sent in a direct message	Authenticated user
`DM_VIDEO`	Messages	Video sent in a direct message	Authenticated user
`DM_DOCUMENT`	Messages	Document sent in a direct message	Authenticated user
`PRODUCT_IMAGE`	Products	Product listing photo	Shop owner

Context	Domain	What it is	Who uploads it
`PRODUCT_VIDEO`	Products	Product demo/preview video	Shop owner
`DIGITAL_PRODUCT`	Products	Purchasable digital file (PDF, ZIP, software, etc.)	Shop owner
`SHOP_BANNER`	Shops	Shop header banner image	Shop owner
`SHOP_LOGO`	Shops	Shop logo / avatar	Shop owner
`EVENT_COVER`	Events	Event banner/hero image	Event organiser
`EVENT_GALLERY`	Events	Additional event gallery image	Event organiser

File Format & Size Constraints

These are the accepted formats and recommended limits per context. FileThunder enforces the actual limits server-side — passing incorrect `mimeType` or oversized files will result in a rejection at the presigned URL stage.

Images

Context	Accepted MIME Types	Max Size	Min Dimensions	Recommended Dimensions
`SOCIAL_IMAGE`	`image/jpeg`, `image/png`, `image/webp`, `image/gif`	20 MB	200 × 200 px	1080 × 1080 px (square) or 1080 × 1920 px (portrait)
`PROFILE_PICTURE`	`image/jpeg`, `image/png`, `image/webp`	10 MB	100 × 100 px	400 × 400 px square
`COVER_PHOTO`	`image/jpeg`, `image/png`, `image/webp`	15 MB	600 × 200 px	1500 × 500 px
`DM_IMAGE`	`image/jpeg`, `image/png`, `image/webp`, `image/gif`	20 MB	—	—
`PRODUCT_IMAGE`	`image/jpeg`, `image/png`, `image/webp`	20 MB	400 × 400 px	1000 × 1000 px square
`SHOP_BANNER`	`image/jpeg`, `image/png`, `image/webp`	15 MB	800 × 200 px	1200 × 400 px

Context	Accepted MIME Types	Max Size	Min Dimensions	Recommended Dimensions
`SHOP_LOGO`	`image/jpeg`, `image/png`, `image/webp`	5 MB	100 × 100 px	400 × 400 px square
`EVENT_COVER`	`image/jpeg`, `image/png`, `image/webp`	15 MB	800 × 400 px	1200 × 630 px
`EVENT_GALLERY`	`image/jpeg`, `image/png`, `image/webp`	20 MB	200 × 200 px	1080 × 1080 px

Videos

Context	Accepted MIME Types	Max Size	Max Duration	Notes
`SOCIAL_VIDEO`	`video/mp4`, `video/quicktime`, `video/webm`, `video/x-msvideo`, `video/x-matroska`	100 MB	60 min	Videos ≥ 3 min get HLS adaptive streaming
`DM_VIDEO`	`video/mp4`, `video/quicktime`, `video/webm`	100 MB	60 min	Always sequential MP4 transcoding
`PRODUCT_VIDEO`	`video/mp4`, `video/quicktime`, `video/webm`, `video/x-msvideo`, `video/x-matroska`	100 MB	60 min	Videos ≥ 3 min get HLS adaptive streaming

Other

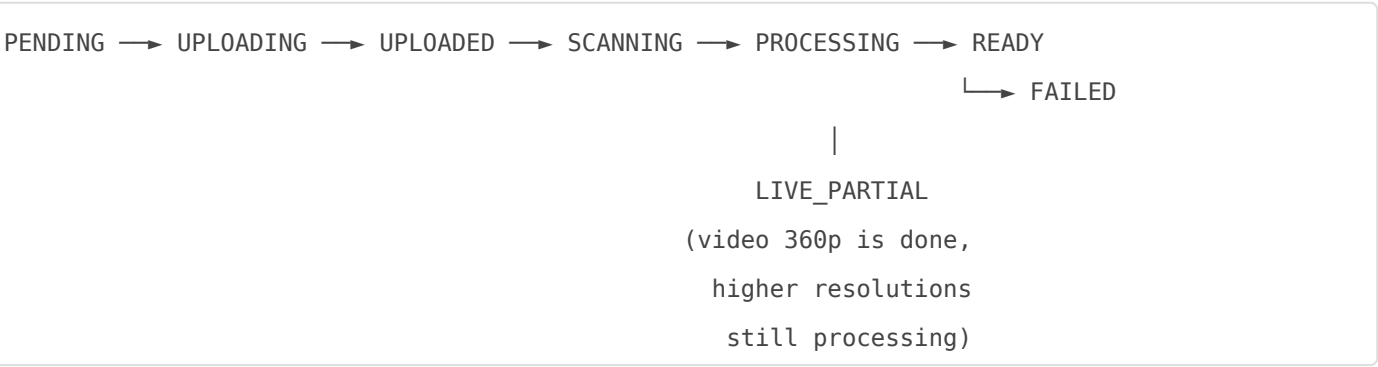
Context	Accepted MIME Types	Max Size	Notes
`DM_DOCUMENT`	`application/pdf`, `application/msword`, `application/vnd.openxmlformats-officedocument.wordprocessingml.document`, `application/vnd.ms-excel`, `application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`, `application/zip`, `text/plain`	50 MB	ClamAV scanned; no variants generated

Context	Accepted MIME Types	Max Size	Notes
`DIGITAL_PRODUCT`	Any — PDF, ZIP, EXE, APK, MP3, etc.	5 GB	ClamAV dual-scan + SHA-256 deduplication; never CDN served; always presigned download

---

Processing Status Lifecycle

After the file reaches object storage, FileThunder processes it through these states. You will receive these values from both the SSE stream and the status polling endpoint.



Status	Meaning	UI suggestion
`PENDING`	Upload slot created, waiting for file bytes	Show spinner
`UPLOADING`	File bytes are being received by storage	Show upload progress bar (XHR)
`UPLOADED`	All bytes received, handing off to processing	Show spinner
`SCANNING`	ClamAV virus scan in progress (digital products / DM docs only)	"Scanning for safety..."
`PROCESSING`	Transcoding / image variant generation in progress	"Processing..."
`LIVE_PARTIAL`	**Videos only** — 360p variant is ready, higher resolutions still processing	Show video with 360p, overlay "HD processing" badge
`READY`	All variants generated and available	Dismiss progress UI, display media
`FAILED`	Processing failed	Show error, offer re-upload option

Variant Keys

When a file reaches `READY`, its variants are stored in the entity's database record and returned in entity API responses. Here are all possible variant keys and what they contain.

Image Variants

Key	Format	Typical Use
`large`	WebP URL	Full-size display, lightbox, detail view
`medium`	WebP URL	Feed cards, grid thumbnails
`thumb`	WebP URL	Tiny previews, avatar chips, comment icons
`og`	WebP URL	`<meta property="og:image">` Open Graph tag
`blurhash`	String (e.g. `LGF5?xYk^6...`)	CSS blur placeholder while image loads
`lqip`	`...` inline data URI	Inline `<img src>` placeholder, no extra request
`dominant_color`	Hex string (e.g. `#F06023`)	Background color while loading, skeleton screen tint

> **Not every key is present for every context.** `blurhash` and `lqip` are always generated. `large`, `medium`, `og`, and `dominant_color` depend on the context — see the Context → Variants Cheat Sheet at the bottom of this document for the exact set per context.

Usage priority: Render `lqip` or apply `blurhash` immediately. Swap in `medium` or `large` once loaded. Use `thumb` for tiny contexts. Always include `og` in page meta where available.

Video Watermarking

Video processing produces **two separate sets of variants**: clean variants for streaming, and a watermarked variant for download. They are different keys in the variants map.

Context	Clean variants	Watermarked variant	Watermark style	Cycle
`SOCIAL_VIDEO`	`360p_clean`, `720p_clean`, `1080p_clean`	`720p_watermarked` (or `360p_watermarked`)	Diagonal 2-point: NexGate logo + `@username` overlay	Every 5 seconds, alternates between upper-left and lower-right

Context	Clean variants	Watermarked variant	Watermark style	Cycle
`PRODUCT_VIDEO`	`360p_clean`, `720p_clean`, `1080p_clean`	`720p_watermarked` (or `360p_watermarked`)	Text-only (`NexGate` label) cycling all 4 corners	Every 3 seconds
`DM_VIDEO`	`360p_clean`, `720p_clean`	None	No watermark	—

Which watermarked key do you get?

- If the source video is tall/wide enough for 720p → key is `720p_watermarked`
- If the source is smaller (e.g. a 360p source) → key is `360p_watermarked`
- Always check which key is present rather than assuming 720p

When to use which:

- Use `360p_clean` / `720p_clean` / `1080p_clean` for in-app streaming and playback
- Use `720p_watermarked` / `360p_watermarked` when you want to offer a **download** of the video — the watermark protects the content

Social video outro: `SOCIAL_VIDEO` files have a personalized branded outro clip appended (username + orange accent, no re-encode). The outro is baked into the `720p_watermarked` variant only, not the clean variants.

The authenticated user's `username` is passed to FileThunder automatically by the backend — you send nothing extra for this.

Video Variants — Short Clips (< 3 minutes)

Key	Format	Typical Use
`360p_clean`	MP4 URL	First available at LIVE_PARTIAL — use for in-app streaming immediately
`720p_clean`	MP4 URL	Standard quality in-app streaming
`1080p_clean`	MP4 URL	HD in-app streaming (only present if source is 1080p-capable)
`720p_watermarked`	MP4 URL	Watermarked download variant — offer this for user downloads
`360p_watermarked`	MP4 URL	Watermarked download fallback — present instead of `720p_watermarked` when source is too small

Key	Format	Typical Use
`preview`	MP4 URL	Auto-generated 3-second silent preview clip (speed-doubled from 6s of footage at ~5% into the video) — use for hover previews on cards
`poster`	WebP URL	Best auto-selected frame — show as video thumbnail before play
`thumb`	WebP URL	Small thumbnail for cards and grids
`og`	WebP URL	1200×630 Open Graph image
`blurhash`	String	BlurHash string for placeholder
`lqip`	data URI	Inline WebP placeholder, no extra request
`dominant_color`	Hex string	Background tint for skeleton screens

Video Variants — Long Form (? 3 minutes, HLS)

Key	Format	Typical Use
`master`	HLS master playlist URL	**Default — use this for all playback.** Plug into HLS.js or native `<video>` on Safari; the player handles quality switching automatically based on bandwidth
`360p_playlist`	HLS per-rendition playlist URL	Only use when building a manual quality selector — swap the player src to this to force 360p
`720p_playlist`	HLS per-rendition playlist URL	Only use when building a manual quality selector — swap the player src to this to force 720p
`1080p_playlist`	HLS per-rendition playlist URL	Only use when building a manual quality selector — swap the player src to this to force 1080p (only present if source is 1080p-capable)
`preview`	MP4 URL	3-second preview clip for card hover effects
`poster`	WebP URL	Best auto-selected frame
`thumb`	WebP URL	Card thumbnail
`og`	WebP URL	Open Graph image
`blurhash`	String	BlurHash placeholder
`lqip`	data URI	Inline placeholder
`dominant_color`	Hex string	Background tint

> **HLS Note**: Always use the `master` key for adaptive streaming — it switches automatically between 360p/720p/1080p based on viewer bandwidth. Long-form videos do **not** include an MP4 fallback — if the environment does not support HLS, you will need to inform the user or handle it at the player level (e.g. HLS.js handles this for most browsers). Long-form videos do not produce a `watermarked` download variant — downloads of long videos are not supported.

The Upload Flow — Step by Step

Step 1 — Request a presigned upload URL (backend)

Call `POST /api/v1/files/request-upload` with the file metadata. You get back a `fileId` and a `presignedUrl`.

Step 2 — Upload the file directly to object storage (MinIO/CDN)

Make an HTTP `PUT` request to the `presignedUrl` with the file bytes as the body. This request does **not** go to the backend — it goes directly to object storage. Use `XMLHttpRequest` (not `fetch`) if you want to track upload progress, because XHR exposes an `upload.onprogress` event that `fetch` does not.

The request must include:

- `Content-Type` header matching exactly the `mimeType` you sent in Step 1
- The raw file bytes as the body (no multipart, no form data)
- No `Authorization` header — the presigned URL is self-authenticating

A successful upload returns `HTTP 200` with an empty body.

Upload progress tracking via XHR: XHR fires `upload.onprogress` events with `loaded` (bytes sent) and `total` (total bytes). Divide `loaded / total` to get a 0–1 progress value for your progress bar. This tracks **network transfer progress**, not processing progress. The progress bar should complete at 100% when the PUT returns 200, then transition to the processing state UI.

Step 3 — Track processing progress (SSE or polling)

After the PUT succeeds, open an SSE stream to `GET /api/v1/files/progress/{fileId}`. You will receive server-sent events as FileThunder processes the file. Close the stream when you receive `READY` or `FAILED`.

If SSE is inconvenient in your environment, use `GET /api/v1/files/status/{fileId}` to poll instead. Recommended polling interval: every 2–3 seconds.

Step 4 — Entity response contains the variants

Once `READY`, the entity API (product, shop, post, etc.) will include the variant map in its response. You do not need to call any file endpoint to get URLs — they come embedded in the entity response.

Endpoints

---

1. Request Upload URL

Purpose: Generate a presigned PUT URL and a `fileId` for a new file upload. Always call this before any file upload.

Endpoint: **POST** `{base_url}/files/request-upload`

Access Level: □□ Protected (Requires authenticated user)

Authentication: Bearer Token — `Authorization: Bearer <token>`

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	`Bearer <your_jwt_token>`
`Content-Type`	string	Yes	`application/json`

Request JSON Sample:

```
{
  "context": "PRODUCT_IMAGE",
  "originalFilename": "shoe-red-side.jpg",
  "mimeType": "image/jpeg",
  "fileSizeBytes": 2457600
}
```

Request Body Parameters:

Parameter	Type	Required	Description	Validation
`context`	string	Yes	The purpose of this file — determines processing pipeline and storage bucket	Must be one of the valid context values listed in the File Contexts section
`originalFilename`	string	Yes	Original name of the file including extension	Max 255 characters
`mimeType`	string	Yes	MIME type of the file exactly as the browser reports it	Must match the context's accepted formats; see File Format & Size Constraints
`fileSizeBytes`	number	Yes	File size in bytes	Must be a positive integer

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Upload URL generated",
  "action_time": "2026-06-21T14:22:10",
  "data": {
    "fileId": "a3f7c21b-09d4-4e8b-bf12-3c7d09e1f556",
    "presignedUrl": "https://storage.nexgate.com/nexgate-raw/products/.../shoe-red-side.jpg?X-Amz-Signature=...",
    "expiresInSeconds": 3600
  }
}
```

Success Response Fields:

Field	Description

`data.fileId`	UUID identifying this file — save this, you will need it for SSE tracking and to associate the file with an entity
`data.presignedUrl`	The URL to PUT the file bytes to — send directly to this URL, do not proxy through your server
`data.expiresInSeconds`	How many seconds before the presigned URL expires (typically 3600 = 1 hour) — start the upload immediately

Error Response JSON Sample:

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "context is required. Valid values: SOCIAL_IMAGE, SOCIAL_VIDEO, PROFILE_PICTURE,
...",
  "action_time": "2026-06-21T14:22:10",
  "data": "context is required."
}
```

Standard Error Types:

- `400 BAD_REQUEST` — Missing or invalid `context`
- `401 UNAUTHORIZED` — Missing or expired Bearer token
- `422 UNPROCESSABLE_ENTITY` — Missing required fields

---

2. Upload File to Object Storage (Presigned PUT)

Purpose: Upload the raw file bytes directly to object storage using the presigned URL from Step 1. This is **not a backend endpoint** — it is a direct PUT to the object storage URL. It is documented here because understanding this step is essential to the flow.

Endpoint: **PUT** `{presignedUrl}` *(the full URL returned in Step 1)*

Access Level: Self-authenticated (The presigned URL carries authentication in its query parameters — no `Authorization` header needed or allowed)

Request Headers:

Header	Type	Required	Description

`Content-Type`	string	Yes	Must exactly match the `mimeType` sent in Step 1 — e.g. `image/jpeg`, `video/mp4`

Request Body: Raw file bytes. No multipart encoding. No form fields. Just the file content.

Upload Progress:

Use `XMLHttpRequest` for this PUT request to gain access to upload progress events. The `XMLHttpRequest.upload` object fires `progress` events containing:

Property	Type	Description
`loaded`	number	Bytes sent so far
`total`	number	Total bytes to send (equals your `fileSizeBytes`)

Divide `loaded / total` to get a 0–1 ratio for a progress bar. This tracks **network transfer only** — once the PUT returns 200, transition the UI to the "Processing..." state and begin tracking via SSE (Step 3).

Success Response: `HTTP 200` with an empty body. No JSON.

Failure Responses:

Status	Cause
`400`	`Content-Type` header does not match what was declared in Step 1
`403`	Presigned URL has expired — go back to Step 1 and request a new one
`413`	File exceeds the size declared in `fileSizeBytes` in Step 1

---

3. Stream File Processing Progress (SSE)

Purpose: Receive real-time server-sent events as FileThunder processes the uploaded file. Open this stream immediately after the PUT in Step 2 returns 200. The stream closes automatically when `READY` or `FAILED` is reached.

Endpoint: `GET` `{base_url}/files/progress/{fileId}`

Access Level: □□ Protected (Only the file owner can stream progress for a given `fileId`)

Authentication: Bearer Token — `Authorization: Bearer <token>`

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	`Bearer <your_jwt_token>`
`Accept`	string	Yes	`text/event-stream`

Path Parameters:

Parameter	Type	Required	Description	Validation
`fileId`	UUID	Yes	The `fileId` returned in Step 1	Must be a valid UUID owned by the authenticated user

Response: This endpoint returns a stream of Server-Sent Events, not a JSON body. Each event has a `name` (the status) and `data` (JSON payload).

SSE Event Format:

```
event: PROCESSING
data: {"status":"PROCESSING","fileId":"a3f7c21b-09d4-4e8b-bf12-3c7d09e1f556"}

event: LIVE_PARTIAL
data: {"status":"LIVE_PARTIAL","fileId":"a3f7c21b-09d4-4e8b-bf12-3c7d09e1f556"}

event: READY
data: {"status":"READY","fileId":"a3f7c21b-09d4-4e8b-bf12-3c7d09e1f556"}
```

SSE Event Sequence:

Short video:

```
PENDING → UPLOADING → UPLOADED → PROCESSING → LIVE_PARTIAL → READY
```

Image:

```
PENDING → UPLOADING → UPLOADED → PROCESSING → READY
```

Long video (≥ 3 minutes):

```
PENDING → UPLOADING → UPLOADED → PROCESSING → LIVE_PARTIAL → READY
```

Digital product or DM document:

```
PENDING → UPLOADING → UPLOADED → SCANNING → READY
```

Event Handling:

Event name	Action
`PENDING`	Show spinner
`UPLOADING`	Show spinner (XHR progress bar already running from Step 2)
`UPLOADED`	Show "Processing..." state
`SCANNING`	Show "Scanning for safety..." state
`PROCESSING`	Show "Processing..." state
`LIVE_PARTIAL`	Video 360p is available — can begin playback, show "HD loading" badge
`READY`	**Close the SSE connection. Show success. Refresh entity data.**
`FAILED`	**Close the SSE connection. Show error. Offer re-upload.**

SSE Timeout: The stream has a 5-minute server-side timeout. If your file is still processing after 5 minutes (large video), reconnect to this endpoint — it will immediately send the current status snapshot before continuing to stream.

Error Responses:

- `401 UNAUTHORIZED` — Invalid or missing token
- `404 NOT_FOUND` — `fileId` not found or does not belong to the authenticated user

---

4. Poll File Processing Status

Purpose: An alternative to SSE for environments where persistent connections are difficult (e.g. React Native background states, certain browser extensions). Returns the current processing status as a single JSON response. Poll every 2–3 seconds; stop when `ready` is `true` or `failed` is `true`.

Endpoint: `GET` `{base_url}/files/status/{fileId}`

Access Level: □□ Protected (Only the file owner)

Authentication: Bearer Token — `Authorization: Bearer <token>`

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	`Bearer <your_jwt_token>`

Path Parameters:

Parameter	Type	Required	Description	Validation
`fileId`	UUID	Yes	The `fileId` returned in Step 1	Must be a valid UUID owned by the authenticated user

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Status retrieved",
  "action_time": "2026-06-21T14:23:45",
  "data": {
    "fileId": "a3f7c21b-09d4-4e8b-bf12-3c7d09e1f556",
    "status": "PROCESSING",
    "ready": false,
    "failed": false,
    "processing": true
  }
}
```

Success Response Fields:

Field	Type	Description
`data.fileId`	UUID	The file identifier
`data.status`	string	Current status — one of `PENDING`, `UPLOADING`, `UPLOADED`, `SCANNING`, `PROCESSING`, `LIVE_PARTIAL`, `READY`, `FAILED`
`data.ready`	boolean	`true` when file is fully processed and variants are available
`data.failed`	boolean	`true` when processing failed — stop polling and show error
`data.processing`	boolean	`true` while processing is in progress — continue polling

Ready state response:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Status retrieved",
  "action_time": "2026-06-21T14:24:10",
  "data": {
    "fileId": "a3f7c21b-09d4-4e8b-bf12-3c7d09e1f556",
    "status": "READY",
    "ready": true,
    "failed": false,
    "processing": false
  }
}
```

Error Response JSON Sample:

```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "File not found or access denied",
  "action_time": "2026-06-21T14:24:10",
  "data": "File not found or access denied"
}
```

Standard Error Types:

- `401 UNAUTHORIZED` — Invalid or missing token
- `404 NOT_FOUND` — `fileId` not found or does not belong to authenticated user

---

5. Replace Video Thumbnail

Purpose: Upload a custom thumbnail image to replace the auto-selected thumbnail for a video file. Call this after the video has reached `READY` status. Returns a presigned URL — upload the image bytes to it (same pattern as Step 2 above).

Endpoint: **POST** `{base_url}/files/thumbnail/{fileId}`

Access Level: □□ Protected (Only the video owner)

Authentication: Bearer Token — `Authorization: Bearer <token>`

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	`Bearer <your_jwt_token>`

Path Parameters:

Parameter	Type	Required	Description	Validation
`fileId`	UUID	Yes	The `fileId` of the video whose thumbnail you want to replace	Must be a valid video file owned by the authenticated user

Request Body: None — this is a POST with no body.

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Thumbnail upload URL generated",
  "action_time": "2026-06-21T14:25:00",
  "data": {
    "fileId": "a3f7c21b-09d4-4e8b-bf12-3c7d09e1f556",
    "presignedUrl": "https://storage.nexgate.com/nexgate-raw/products/.../thumb-
a3f7c21b.jpg?X-Amz-Signature=...",
    "expiresInSeconds": 3600
  }
}
```

Success Response Fields:

Field	Description
`data.fileId`	The video file's ID
`data.presignedUrl`	PUT the thumbnail image bytes here — same process as the main upload in Step 2
`data.expiresInSeconds`	Seconds until the presigned URL expires

After uploading: PUT `image/jpeg` or `image/png` bytes to the `presignedUrl`. FileThunder will generate the thumbnail variants and update the video's effective thumbnail. Refresh the entity response to get the updated thumbnail URLs.

Error Response JSON Sample:

```
{
  "success": false,
  "httpStatus": "UNAUTHORIZED",
  "message": "Token has expired",
  "action_time": "2026-06-21T14:25:00",
  "data": "Token has expired"
}
```

Standard Error Types:

- `401 UNAUTHORIZED` — Invalid or missing token
- `404 NOT_FOUND` — `fileId` not found

---

6. List Digital Files Available for Download (Order)

Purpose: For a purchased digital product order, list all the files the buyer is entitled to download, along with download availability and remaining download counts.

Endpoint: `GET` `{base_url}/e-commerce/orders/{orderId}/downloads`

Access Level: □□ Protected (Only the order buyer)

Authentication: Bearer Token — `Authorization: Bearer <token>`

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	`Bearer <your_jwt_token>`

Path Parameters:

Parameter	Type	Required	Description	Validation
`orderId`	UUID	Yes	The order ID containing digital files	Must be an order owned by the authenticated buyer

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "3 file(s) available for download",
  "action_time": "2026-06-21T14:26:00",
  "data": [
    {
      "fileId": "b9c1d33e-22f4-4a0b-9e67-1d4f22c3a881",
      "fileName": "nexgate-design-kit-v2.zip",
      "contentType": "application/zip",
      "fileSize": 52428800,
      "downloadCount": 1,
      "downloadsRemaining": 4,
      "accessExpiresAt": "2026-12-21T00:00:00",
      "canDownload": true
    },
    {
      "fileId": "c2e5f44a-33g5-5b1c-af78-2e5g33d4b992",
      "fileName": "user-manual.pdf",
      "contentType": "application/pdf",
      "fileSize": 1048576,
      "downloadCount": 0,
      "downloadsRemaining": 5,
      "accessExpiresAt": "2026-12-21T00:00:00",
      "canDownload": true
    }
  ]
}
```


Success Response Fields:

Field	Type	Description
`[].fileId`	UUID	File identifier — use this in the next endpoint to get a download URL
`[].fileName`	string	Original filename including extension
`[].contentType`	string	MIME type of the file
`[].fileSize`	number	File size in bytes
`[].downloadCount`	number	How many times the buyer has already downloaded this file

Field	Type	Description
`[].downloadsRemaining`	number	How many more downloads the buyer is allowed
`[].accessExpiresAt`	string	ISO 8601 datetime after which download access expires
`[].canDownload`	boolean	`false` if download limit reached or access has expired — show disabled button

Error Response JSON Sample:

```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "Order not found",
  "action_time": "2026-06-21T14:26:00",
  "data": "Order not found"
}
```

Standard Error Types:

- `400 BAD_REQUEST` — Order does not contain digital files, or buyer access is expired
- `401 UNAUTHORIZED` — Invalid or missing token
- `404 NOT_FOUND` — Order not found or does not belong to authenticated user

---

7. Generate Digital File Download URL

Purpose: Generate a time-limited, single-use download URL for a specific digital file in an order. Call this endpoint when the user clicks the download button — do not cache this URL. Open it immediately in a new tab or trigger a browser download.

Endpoint: `GET` `{base_url}/e-commerce/orders/{orderId}/downloads/{fileId}`

Access Level: □□ Protected (Only the order buyer)

Authentication: Bearer Token — `Authorization: Bearer <token>`

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	`Bearer <your_jwt_token>`

Path Parameters:

Parameter	Type	Required	Description	Validation
`orderId`	UUID	Yes	The order ID	Must belong to the authenticated buyer
`fileId`	UUID	Yes	The specific file to download (from endpoint 6)	Must be a file within the specified order

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Download URL generated — link expires in 15 minutes",
  "action_time": "2026-06-21T14:27:00",
  "data": {
    "fileId": "b9c1d33e-22f4-4a0b-9e67-1d4f22c3a881",
    "fileName": "nexgate-design-kit-v2.zip",
    "downloadUrl": "https://storage.nexgate.com/nexgate-digital/products/.../nexgate-design-kit-v2.zip?X-Amz-Signature=...&X-Amz-Expires=300",
    "expiresAt": "2026-06-21T14:32:00",
    "downloadsRemaining": 3,
    "downloadCount": 2
  }
}
```


Success Response Fields:

Field	Type	Description
`data.fileId`	UUID	File identifier
`data.fileName`	string	Filename to use when saving locally
`data.downloadUrl`	string	Pre-signed download URL — valid for **15 minutes only**. Open immediately. Do not cache.
`data.expiresAt`	string	ISO 8601 datetime when the download URL expires
`data.downloadsRemaining`	number	How many downloads remain after this one
`data.downloadCount`	number	Total downloads made so far including this one

Error Response JSON Sample:

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Download limit reached for this file",
  "action_time": "2026-06-21T14:27:00",
  "data": "Download limit reached for this file"
}
```

Standard Error Types:

- `400 BAD_REQUEST` — Download limit reached, access expired, or buyer is not eligible
- `401 UNAUTHORIZED` — Invalid or missing token
- `404 NOT_FOUND` — Order or file not found

How Variants Appear in Entity Responses

You never call a dedicated endpoint to get variant URLs. When an entity (product, post, shop, profile, event) is fetched via its own API, the response already contains the assembled variant map. Here is what to expect per entity type:

Product Image Variants

```
{
  "productImageVariants": [
    {
      "large": "https://cdn.nexgate.com/products/owner123/fileabc/large.webp",
      "medium": "https://cdn.nexgate.com/products/owner123/fileabc/medium.webp",
      "thumb": "https://cdn.nexgate.com/products/owner123/fileabc/thumb.webp",
      "og": "https://cdn.nexgate.com/products/owner123/fileabc/og.webp",
      "blurhash": "LGF5?xYk^6#M@-5c,1J5@[or[Q6",
      "lqip": "...",
      "dominant_color": "#C8A882"
    }
  ]
}
```

`productImageVariants` is an array — one entry per uploaded image. Index 0 is the primary image.

Video Variants (short clip)

```
{
  "previewVariants": {
    "360p_clean": "https://cdn.nexgate.com/products/owner123/filevid/360p_clean.mp4",
    "720p_clean": "https://cdn.nexgate.com/products/owner123/filevid/720p_clean.mp4",
    "1080p_clean": "https://cdn.nexgate.com/products/owner123/filevid/1080p_clean.mp4",
    "720p_watermarked":
"https://cdn.nexgate.com/products/owner123/filevid/720p_watermarked.mp4",
    "preview": "https://cdn.nexgate.com/products/owner123/filevid/preview_3s.mp4",
    "poster": "https://cdn.nexgate.com/products/owner123/filevid/poster.webp",
    "thumb": "https://cdn.nexgate.com/products/owner123/filevid/thumb.webp",
    "og": "https://cdn.nexgate.com/products/owner123/filevid/og.webp",
    "blurhash": "LGF5?xYk^6#M@-5c,1J5@[or[Q6",
    "lqip": "...",
    "dominant_color": "#1A1A2E"
  }
}
```

> If the source video was too small for 720p, `720p_watermarked` will be absent and `360p_watermarked` will be present instead. Always check which watermarked key exists before rendering a download button.

Video Variants (long form / HLS)

```
{
  "previewVariants": {
    "master": "https://cdn.nexgate.com/products/owner123/filevid/hls/master.m3u8",
    "360p_playlist": "https://cdn.nexgate.com/products/owner123/filevid/hls/360p/360p.m3u8",
    "720p_playlist": "https://cdn.nexgate.com/products/owner123/filevid/hls/720p/720p.m3u8",
    "1080p_playlist":
"https://cdn.nexgate.com/products/owner123/filevid/hls/1080p/1080p.m3u8",
    "preview": "https://cdn.nexgate.com/products/owner123/filevid/preview_3s.mp4",
    "poster": "https://cdn.nexgate.com/products/owner123/filevid/poster.webp",
    "thumb": "https://cdn.nexgate.com/products/owner123/filevid/thumb.webp",
    "og": "https://cdn.nexgate.com/products/owner123/filevid/og.webp",
```

```
    "blurhash": "LGF5?xYk^6#M@-5c,1J5@[or[Q6",
    "lqip": "...",
    "dominant_color": "#1A1A2E"
  }
}
```

> Feed `master` into HLS.js or a native `<video>` src on Safari. Long-form videos do not include an MP4 fallback and do not have a watermarked download variant.

LIVE_PARTIAL state (video still processing)

When you get an entity response while the video is in `LIVE_PARTIAL`:

- **Short clips**: only `360p_clean` is present alongside the thumbnail keys. Higher resolutions and the watermarked variant arrive once `READY`.
- **Long form (HLS)**: only `360p_playlist` and `master` (pointing to the 360p-only rendition) are present. Additional renditions and thumbnail keys arrive once `READY`.

You can safely begin playback in both cases and show a "HD loading" badge.

Null variants

If a file is still in `PROCESSING` and you fetch the entity, `variants` fields may be `null` or an empty array. Always null-check before rendering. Display a placeholder (blurhash, dominant_color, or a skeleton) until variants are available.

Best Practices

Upload

- **Always validate before requesting a presigned URL.** Check file size and MIME type on the client before making the Step 1 request. Rejecting obvious bad inputs early avoids wasted presigned URL slots.
- **Start the SSE connection before the PUT finishes.** Open the SSE stream as soon as you receive `fileId`, so you do not miss early status events like `UPLOADING`.
- **Never proxy the file through your frontend server.** The presigned URL uploads directly to object storage. Do not relay bytes through a backend API route.

- **Handle presigned URL expiry.** If a user leaves the upload screen and comes back, the presigned URL may have expired. Call Step 1 again to get a fresh one.
- **Send exactly the declared `mimeType` as `Content-Type`** in the PUT. A mismatch causes a `400` from object storage.

Progress UI

- Use **XHR upload progress** for the bytes-in-flight phase (Step 2). Show a numeric percentage or progress bar.
- Transition to a **spinner or indeterminate progress** at 100% upload — processing time is unpredictable.
- For **short images**, processing is usually 2–5 seconds.
- For **short videos (< 3 min)**, processing typically takes 30 seconds to 2 minutes.
- For **long videos (≥ 3 min)**, processing can take 5–15 minutes. Show `LIVE_PARTIAL` content early at 360p with an overlay badge.
- On `FAILED`, offer a clear re-upload action — do not auto-retry silently.

SSE vs Polling

Use SSE when	Use polling when
Web browser context	React Native / mobile apps
Single in-progress upload screen	Background upload (app minimised)
You can keep the tab open	Network conditions drop connections often

SSE is preferred — it is lower overhead and gives instant events. Polling at 2-second intervals is a reliable fallback.

Variants

- **Always show a placeholder first.** Use `lqip` as an inline `src` or apply `blurhash` as a CSS background immediately — before the real image loads.
- **Use `dominant_color`** as the background tint on skeleton screens and image containers before any image variant is available.
- **Use `medium` for feed cards and grids**, not `large`. `large` is for detail views and lightboxes only.
- **Always set `og` in Open Graph meta tags** when rendering shareable pages (products, posts, events).
- **For HLS video**, prefer `master.m3u8` on supported browsers. Long-form videos have no MP4 fallback — use HLS.js to cover non-native HLS environments (most Android browsers, older Chrome).
- **Do not hardcode or cache variant URLs.** They come from entity responses. If a CDN or storage URL base changes, your app automatically picks up the new URLs without any

code change.

Digital Downloads

- **Never store download URLs.** They expire in 15 minutes. Always call endpoint 7 on each user-initiated download click.
- **Check `canDownload` from endpoint 6** before showing the download button. If `false`, show a disabled button with a reason ("Download limit reached" or "Access expired").
- **Open the `downloadUrl` in a new browser tab** or trigger a native download — do not `fetch()` it through your app.
- **Re-fetch endpoint 6 after each download** to update the `downloadsRemaining` count shown to the user.

Quick Reference

Context ? Variants Cheat Sheet

Context	Variants Generated
`SOCIAL_IMAGE`	large, medium, thumb, og, blurhash, lqip, dominant_color
`SOCIAL_VIDEO` (short)	360p_clean, 720p_clean, 1080p_clean, 720p_watermarked (or 360p_watermarked), preview, poster, thumb, og, blurhash, lqip, dominant_color
`SOCIAL_VIDEO` (long)	master, 360p_playlist, 720p_playlist, 1080p_playlist, preview, poster, thumb, og, blurhash, lqip, dominant_color
`PROFILE_PICTURE`	medium, thumb, blurhash, lqip
`COVER_PHOTO`	large, thumb, og, blurhash, lqip
`DM_IMAGE`	medium, thumb, blurhash, lqip
`DM_VIDEO`	360p_clean, 720p_clean
`DM_DOCUMENT`	*(none — secure download only)*
`PRODUCT_IMAGE`	large, medium, thumb, og, blurhash, lqip, dominant_color
`PRODUCT_VIDEO` (short)	360p_clean, 720p_clean, 1080p_clean, 720p_watermarked (or 360p_watermarked), preview, poster, thumb, og, blurhash, lqip, dominant_color
`PRODUCT_VIDEO` (long)	master, 360p_playlist, 720p_playlist, 1080p_playlist, preview, poster, thumb, og, blurhash, lqip, dominant_color
`DIGITAL_PRODUCT`	*(none — secure download only)*
`SHOP_BANNER`	large, thumb, og, blurhash, lqip

Context	Variants Generated
`SHOP_LOGO`	medium, thumb, blurhash, lqip
`EVENT_COVER`	large, medium, thumb, og, blurhash, lqip
`EVENT_GALLERY`	large, medium, thumb, blurhash, lqip

Status Codes

Code	Meaning
`200 OK`	Success
`400 BAD_REQUEST`	Invalid request data, limit reached, or item already exists
`401 UNAUTHORIZED`	Missing, expired, or invalid token
`403 FORBIDDEN`	Authenticated but not permitted (not the file owner, not the buyer)
`404 NOT_FOUND`	File, order, or resource not found
`422 UNPROCESSABLE_ENTITY`	Validation error — response `data` will contain field-level error messages
`500 INTERNAL_SERVER_ERROR`	Server error — report to backend team with `action_time`

Authentication

All endpoints require: `Authorization: Bearer <your_jwt_token>`

Tokens are obtained from the auth endpoints (see Auth API documentation). A `401` means the token is expired or invalid — redirect the user to login.

Revision #1
Created 21 June 2026 11:27:12 by Admin Qbit
Updated 21 June 2026 11:27:57 by Admin Qbit