Carmic Pro — API Reference

Carmic Pro — Public API Reference

A user-facing reference for integrating with Carmic Pro from third-party applications (e.g. ApexStream).

- **Base URL (production):** `https://api.carmic.pro/api`
- **Interactive Swagger:** `https://api.carmic.pro/api/docs` *(auto-generated by Django Ninja)*
- **API version prefix:** all endpoints below sit under `/v1/`
- **Format:** JSON requests, JSON responses (`application/json`)
- **All response payloads** are wrapped as `{ "data": ..., "meta": {...}? }` unless noted

---

1. Authentication

Carmic uses **JWT bearer tokens** issued by the Auth router. There is **no API-key or service-token mechanism** — every caller must log in as a real user.

1.1 Headers required on authenticated calls

Header	Value	Notes
`Authorization`	`Bearer <access_jwt>`	Required on all non-public endpoints
`X-Workspace-ID`	`<workspace_uuid>`	Required on every endpoint that touches workspace-owned data (projects, clips, render, exports, media, templates, social)
`Content-Type`	`application/json`	Required on POST/PUT/PATCH

A request missing `X-Workspace-ID` against a workspace-scoped endpoint returns `403 Workspace context required`.

1.2 Obtaining a token

```
POST /v1/auth/login
Content-Type: application/json

{ "email": "service@your-org.com", "password": "..." }
```

Response:

```
{
  "data": {
    "access":  "eyJhbGciOi...",
    "refresh": "eyJhbGciOi...",
    "user":    { "id": "...", "email": "...", "first_name": "", "last_name": "" }
  }
}
```

The access token is short-lived. Rotate via:

```
POST /v1/auth/refresh
{ "refresh": "..." }
```

1.3 Discovering your workspace ID

```
GET /v1/me
Authorization: Bearer <access>
```

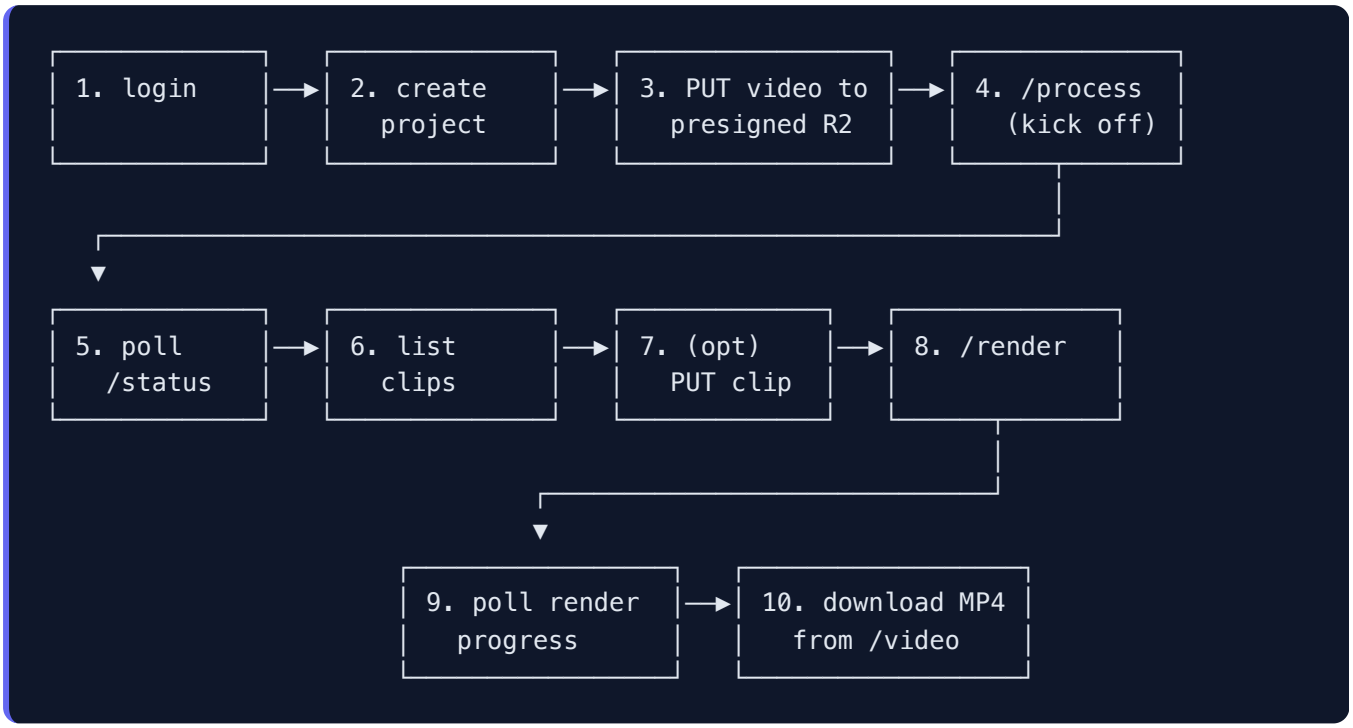
Returns the user profile and the list of workspaces the user belongs to. Pick the workspace UUID and pass it in the `X-Workspace-ID` header on every subsequent call.

1.4 Rate limits on auth

Endpoint	Limit
`POST /v1/auth/register`	3 per 15 min per IP **and** per email
`POST /v1/auth/login`	10 per 60 s per IP, 5 per 60 s per email
`POST /v1/auth/password-reset`	3 per hour per IP and per email

2. Quickstart — drive the editor end-to-end

The diagram below is the complete pipeline for: **upload a video → let Carmic detect viral clips → render → download**. Every step is one HTTP call.



Worked example (`curl`)

```
# ── 1. Login ──────────────────────────────────────────────────
ACCESS=$(curl -s https://api.carmic.pro/api/v1/auth/login \
  -H 'Content-Type: application/json' \
  -d '{"email":"service@apexstream.com","password":"..."}' \
  | jq -r '.data.access')

# ── 2. Find workspace id ──────────────────────────────────────
WS=$(curl -s https://api.carmic.pro/api/v1/me \
  -H "Authorization: Bearer $ACCESS" \
  | jq -r '.data.workspaces[0].id')

# ── 3. Create project + presigned upload URL ──────────────────
RESP=$(curl -s https://api.carmic.pro/api/v1/projects/create \
  -H "Authorization: Bearer $ACCESS" -H "X-Workspace-ID: $WS" \
  -H 'Content-Type: application/json' \
  -d '{
    "filename":"podcast.mp4",
    "filesize": 524288000,
    "content_type":"video/mp4",
    "duration": 1800,
    "language":"en",
    "video_type":"podcast"
  }')
PID=$(echo "$RESP" | jq -r '.data.project_id')
URL=$(echo "$RESP" | jq -r '.data.upload_url')

# ── 4. PUT the video directly to R2 ───────────────────────────
curl -X PUT --data-binary @podcast.mp4 -H 'Content-Type: video/mp4' "$URL"

# ── 5. Trigger ingestion ──────────────────────────────────────
curl -s -X POST https://api.carmic.pro/api/v1/projects/$PID/process \
  -H "Authorization: Bearer $ACCESS" -H "X-Workspace-ID: $WS"

# ── 6. Poll until clips ready ─────────────────────────────────
while : ; do
  S=$(curl -s https://api.carmic.pro/api/v1/projects/$PID/status \
        -H "Authorization: Bearer $ACCESS" -H "X-Workspace-ID: $WS")
  echo "$S" | jq -r '.data.current_step + " (" + (.data.progress|tostring) + "%)"'
  [ "$(echo $S | jq -r '.data.status')" = "CLIPS_READY" ] && break
  sleep 10
done

# ── 7. List clips ─────────────────────────────────────────────
CLIPS=$(curl -s https://api.carmic.pro/api/v1/projects/$PID \
  -H "Authorization: Bearer $ACCESS" -H "X-Workspace-ID: $WS")
CID=$(echo "$CLIPS" | jq -r '.data.clips[0].id')

# ── 8. (Optional) save editor template_config — see §5.4 ──────
# curl -X PUT .../v1/clips/$CID -d @template.json ...
```

```
# ── 9. Render ──────────────────────────────────────────────
curl -s -X POST https://api.carmic.pro/api/v1/clips/$CID/render \
  -H "Authorization: Bearer $ACCESS" -H "X-Workspace-ID: $WS"

# ── 10. Poll render progress + download ────────────────────
while : ; do
  P=$(curl -s https://api.carmic.pro/api/v1/clips/$CID/render-progress \
        -H "Authorization: Bearer $ACCESS" -H "X-Workspace-ID: $WS")
  echo "$P" | jq -r '.data.status + " (" + (.data.progress|tostring) + "%)"'
  [ "$(echo $P | jq -r '.data.status')" = "RENDERED" ] && break
  sleep 5
done

curl -L -o clip.mp4 https://api.carmic.pro/api/v1/clips/$CID/video \
  -H "Authorization: Bearer $ACCESS" -H "X-Workspace-ID: $WS"
```

---

3. Endpoint reference

Endpoints are grouped by the resource they operate on. Path parameters are shown as `{name}`. **All endpoints require `Authorization: Bearer <jwt>` + `X-Workspace-ID` unless explicitly marked Public.**

3.1 Auth — `/v1/auth/`

Method	Path	Body	Returns
POST	`/register`	`{email, password, first_name?, last_name?}`	`{access, refresh, user}` *(public)*
POST	`/login`	`{email, password}`	`{access, refresh, user}` *(public)*
POST	`/refresh`	`{refresh}`	`{access}` *(public)*
POST	`/logout`	`{refresh}`	`{ok: true}` *(blacklists refresh token)*
POST	`/password-reset`	`{email, redirect_base?}`	`{ok: true}` *(always returns ok to prevent email enumeration)*
POST	`/password-reset/confirm`	`{token, new_password}`	`{ok: true}`

Password rules: ≥8 chars, ≥1 uppercase, ≥1 lowercase, ≥1 digit.

3.2 Profile & Workspaces — `/v1/`

Method	Path	Description
GET	`/me`	Current user + list of workspaces they belong to
PATCH	`/me`	Update `first_name`, `last_name`, `avatar_url`
POST	`/{workspace_id}/invite`	Invite member by email *(OWNER only)*
PATCH	`/{workspace_id}/members/{user_id}`	Change role OWNER ↔ EDITOR *(OWNER only)*
DELETE	`/{workspace_id}/members/{user_id}`	Remove member *(OWNER only)*

3.3 Ingestion — `/v1/projects/...`

Method	Path	Body	Notes
POST	`/projects/create`	see below	Returns `{project_id, upload_url, object_key}`. Workspace tier-gated (5 GB max, concurrency limit).
POST	`/projects/{project_id}/process`	—	Kick off Celery ingestion after the R2 upload completes
POST	`/projects/create-from-url`	`{url, language?, language_mode?, video_type?, custom_instructions?, trim_start?, trim_end?}`	Server-side download (YouTube etc.) instead of direct upload
GET	`/projects/{project_id}/status`	—	`{id, status, title, progress (0–100), current_step, clip_count}`
POST	`/projects/{project_id}/cancel`	—	Cancel in-flight processing
DELETE	`/projects/{project_id}/cancel`	—	Same as above (legacy verb)

`POST /v1/projects/create` request body

```
{
  "filename": "podcast.mp4",          // required
  "filesize": 524288000,              // required, bytes; max 5 GB
  "content_type": "video/mp4",        // mp4 | mov | avi | webm | mkv | mpeg | flv
  "title": "Episode 12",              // optional, defaults to filename stem
  "duration": 1800,                   // seconds, used for billing pre-deduction
  "video_type": "podcast",            // auto | podcast | interview | tutorial | ...
  "language": "en",                   // ISO 639-1
  "language_mode": "single",          // single | multi
  "enable_multimodal": false,         // Gemini Vision pass
  "custom_instructions": "Focus on segments where the guest is laughing",
  "trim_start": 60,                   // optional, skip first 60 s
  "trim_end": 1740                    // optional, stop at 29 min
}
```

Project status lifecycle

```
PENDING_UPLOAD → UPLOADING → DOWNLOADING (URL ingest only) →
TRANSCRIBING → PROCESSING → FACE_TRACKING → CLIPS_READY → COMPLETED
                                                        ↘ FAILED / FAILED_DOWNLOAD /
                                                          FAILED_NO_SPEECH /
                                                          FAILED_BILLING /
                                                          FAILED_CORRUPT_VIDEO
```

Concurrency limits (per workspace):

Tier	Max concurrent processing videos
FREE / STANDARD / PROFESSIONAL	1
BUSINESS	2
ENTERPRISE	unlimited

3.4 Projects — `/v1/projects`

Method	Path	Description
GET	`/projects?limit=20&offset=0&search=&status=`	Paginated list. Returns `{data:[ProjectOut], meta:{total,limit,offset,has_more}}`
GET	`/projects/{project_id}`	Full project record + array of clips (only populated once `visual_pipeline_ready: true`)
DELETE	`/projects/{project_id}`	Soft-delete the project and purge all R2 objects under its prefix
GET	`/projects/caption-templates`	Available caption preset list

`GET /v1/projects/{id}` returns

```
{
  "data": {
    "id": "...", "title": "...", "status": "CLIPS_READY",
    "duration": 1800, "thumbnail_url": "...",
    "visual_pipeline_ready": true,
    "clips_ready_at": "2026-05-09T12:34:56Z",
    "clip_count": 12,
    "clips": [
      {
        "id": "...", "title": "...",
        "start_time": 142.4, "end_time": 187.9,
        "virality_score": 87, "reasoning": "...",
        "status": "DETECTED",
        "thumbnail_url": "...", "video_url": "...",
        "source_video_url": "https://r2-presigned...",
        "proxy_url": "https://r2-presigned...720p.mp4",
        "broll_suggestions": [...]
      }
    ]
  }
}
```

3.5 Clips & Editor — `/v1/clips`

Method	Path	Body	Description
GET	`/clips/{clip_id}`	—	Full editor hydration payload (template_config, transcript words, layout timeline, spatial matrix, shots)
PUT	`/clips/{clip_id}`	`TemplateConfig` (see §3.5.1)	Save editor state
POST	`/clips/{clip_id}/render`	—	Queue final render. Deducts processing minutes. **402** if insufficient minutes. Returns `{render_job_id, status: "QUEUED", clip_id}`
POST	`/clips/{clip_id}/retry-render`	—	Requeue a `FAILED_RENDER` clip; billing refund handled server-side
POST	`/clips/{clip_id}/rebuild-layout`	—	Re-run AI auto-director (requires existing YOLO spatial matrix → `409` otherwise)
GET	`/clips/{clip_id}/render-progress`	—	Poll `{status, progress, render_job_id?}`
GET	`/clips/{clip_id}/video`	—	Streams the rendered MP4 (302 → presigned R2). Returns 404 if not yet rendered
GET	`/clips/{clip_id}/source-video?token=`	—	Original source video. *Public when `?token=` is a server-signed thumbnail token; otherwise auth required*
POST	`/clips/{clip_id}/emoji-fill`	—	Auto-place emoji reactions over speakers
POST	`/clips/{clip_id}/subtitle-dry-run`	—	Preview subtitle render (no save)

3.5.1 `TemplateConfig` schema (PUT body)

```
{
  "layout": {
    "layoutType": "face-track",     // face-track | fit | split
    "padding": 0,
    "backgroundBlur": 20,
    "splitTopSpeaker": "A",
    "safeZoneOverlay": false,
    "aiSpeakerSwitch": true
  },
  "audio": {
    "autoSfxEnabled": false,
    "musicTrack": "none",
    "activeTrack": null,            // {id, url, name, ...} or null
    "musicVolume": 30,
    "voiceVolume": 100,
    "removePauses": false
  },
  "elements": {
    "progressBar": { "enabled": false, "color": "#8b5cf6", "position": "bottom" },
    "watermark":   { "enabled": false, "imageUrl": "", "opacity": 70 },
    "hookTitle":   { "enabled": false, "text": "", "durationSec": 3.0 }
  },
  "style": {
    "captionFont": "Inter",
    "fontSize": 72,
    "lineHeight": 1.2,
    "strokeThickness": 3,
    "captionColor": "#ffffff",
    "highlightColor": "#FFD700",
    "dropShadow": true,
    "shadowIntensity": 50,
    "uppercase": true,
    "captionPosition": "bottom",    // top | middle | bottom | custom
    "yAxisCustomPct": 75,
    "linesPerScreen": 2,            // int | "auto"
    "animationStyle": "pop",
    "animationIntensity": 50,
    "captionBg": "rgba(0,0,0,0)"
  },
  "b_roll": [
    {
      "id": "...",
      "sourceId": "pexels:1234567",
      "videoUrl": "https://...",
      "thumbnailUrl": "https://...",
      "startTime": 12.5,            // position on the clip timeline
      "duration": 4.0,
      "sourceStart": 0,
      "sourceDuration": 0
```

```
    }
  ],
  "caption_style": "HORMOZI",
  "shots": [                          // optional — must be zero-gap contiguous
    { "start_time": 0.0, "end_time": 5.0, ...},
    { "start_time": 5.0, "end_time": 10.0, ...}
  ]
}
```

3.6 B-roll & stock media — `/v1/`

Method	Path	Description
GET	`/stock-videos?q=&per_page=`	Pexels stock video search *(public)*
GET	`/stock-search?q=&type=&per_page=`	Unified video + image stock search *(public)*
GET	`/stock-videos-pexels?q=&per_page=`	Direct Pexels passthrough
POST	`/clips/{clip_id}/generate-broll`	AI-generate matching B-roll suggestions
GET	`/clips/{clip_id}/broll`	Fetch existing B-roll suggestions for a clip
POST	`/clips/{clip_id}/custom-media/presign`	Presigned R2 PUT for clip-scoped media
GET	`/clips/{clip_id}/custom-media`	List clip-scoped custom media
POST	`/workspaces/media`	Upload to workspace-level media library

3.7 Storage — `/v1/storage/`

Method	Path	Body	Description
POST	`/upload-url`	`{filename, content_type, bucket?}`	Presigned PUT URL (1 h TTL)
POST	`/playback-url`	`{object_key}`	Presigned GET URL for video playback

3.8 NLE Exports — `/v1/`

Method	Path	Body	Description
POST	`/clips/{clip_id}/exports`	`{target_format, include_proxy?}`	Returns `202 {export_id, status:"PENDING"}`. Pro/Elite tier only (**402** for FREE). 5/min/workspace (**429**).
GET	`/exports/{export_id}`	—	Poll for `{id, status, download_url, file_size, error_message}`
GET	`/clips/{clip_id}/exports`	—	List all exports for a clip
GET	`/exports/{export_id}/stream?ticket=`	—	SSE progress stream *(public, ticket-auth)*

`target_format` valid values: `premiere`, `davinci`, `fcpx`, `after_effects`. (`logic_pro` is intentionally not yet exposed.)

3.9 Templates — `/v1/templates`

Method	Path	Description
GET	`/?ratio=9_16`	List system + workspace custom templates, optionally filtered by aspect ratio
POST	`/`	Save custom brand template
DELETE	`/{template_id}`	Delete custom template *(system templates immutable)*
GET	`/defaults`	Per-ratio default template assignments
PUT	`/defaults/{ratio}`	Set workspace default for a ratio *(tier-gated → 402)*
DELETE	`/defaults/{ratio}`	Clear default
POST	`/{template_id}/regenerate-previews`	Re-render preview images

3.10 Media Library — `/v1/media`

Method	Path	Description
GET	`/?media_type=&search=&limit=&offset=`	List workspace media (image/video/audio/font)
POST	`/presign`	Presigned R2 POST URL for upload
POST	`/confirm`	Confirm upload + create `MediaFile` record
GET	`/stats`	Storage usage + tier cap
DELETE	`/{media_id}`	Delete from R2 + DB

Tier storage caps: FREE 1 GB · STANDARD 10 GB · PROFESSIONAL 50 GB · BUSINESS 100 GB · ENTERPRISE 500 GB.

3.11 Social Accounts — `/v1/social`

Method	Path	Description
GET	`/accounts`	Connected platforms for workspace
POST	`/connect/{platform}`	Returns OAuth consent URL + signed state token
POST	`/callback/{platform}`	OAuth code exchange
DELETE	`/disconnect/{account_id}`	Revoke connection
PATCH	`/toggle/{account_id}`	Enable/disable without deleting
POST	`/refresh/{platform_id}`	Force OAuth token refresh

Platforms: `youtube`, `facebook`, `instagram`.

3.12 System & realtime — `/v1/`

Method	Path	Description
GET	`/health`	Postgres + Redis health *(public)*
POST	`/sse-ticket`	Exchange JWT for a 30 s opaque single-use ticket. Pass via `?ticket=` to SSE endpoints since `EventSource` cannot send `Authorization` headers
GET	`/projects/{project_id}/stream?ticket=`	SSE: live ingestion progress *(ticket-auth)*
GET	`/exports/{export_id}/stream?ticket=`	SSE: live NLE export progress *(ticket-auth)*

Subscribing to SSE from Node

```
const ticket = await fetch(`${API}/v1/sse-ticket`, {
  method: 'POST',
  headers: { Authorization: `Bearer ${access}`, 'X-Workspace-ID': ws }
}).then(r => r.json()).then(j => j.ticket)

const es = new EventSource(`${API}/v1/projects/${pid}/stream?ticket=${ticket}`)
es.onmessage = (ev) => console.log(JSON.parse(ev.data))
```

---

4. Conventions

4.1 Response envelope

Most success responses look like:

```
{ "data": <payload>, "meta": { "total": 99, "limit": 20, "offset": 0, "has_more":
        true } }
```

`meta` is only present on listing endpoints.

4.2 Error envelope

```
{ "error": "Workspace context required (X-Workspace-ID)", "code": 403 }
```

HTTP	When
400	Validation failed (missing required fields, invalid enum values)
401	Missing or invalid JWT
402	Billing required — insufficient processing minutes (render) or wrong tier (NLE export)
403	Auth ok but no access to this workspace / resource
404	Resource not found
409	Conflict — e.g. trying to re-process a project past `PENDING_UPLOAD`
413	Upload exceeds 5 GB
415	Unsupported `content_type`
422	Schema validation failed (timeline gaps, etc.)
429	Rate-limited (see headers `Retry-After` where present)
500	Server error — also reported to Sentry

4.3 Aspect ratios & template presets

`caption_style` accepted values: `HORMOZI`, `DEFAULT`, `MINIMAL`, `IMPACT`, `KARAOKE`, `BUBBLE`, `WHISPER` (plus any custom templates returned by `/v1/templates`).

4.4 Idempotency

- `POST /projects/{id}/process` is idempotent — calling it on a project already in `PROCESSING` returns 409.
- `POST /clips/{id}/exports` is idempotent on `(clip_id, target_format)` — duplicate calls return the existing `export_id`.
- Project creation deducts billing minutes once per project ID.

---

5. Integrating from ApexStream

5.1 Auth bootstrap (one-time setup)

ApexStream needs a dedicated **service user** in Carmic (Carmic does not currently issue API keys). Suggested setup:

1. Have the Carmic admin create a workspace `apexstream` and a user `apexstream-svc@your-domain.com` with EDITOR or OWNER role.
2. Store `CARMIC_SVC_EMAIL`, `CARMIC_SVC_PASSWORD` in ApexStream secret manager.
3. On boot, ApexStream POSTs to `/v1/auth/login`, caches `access` (~30 min) and `refresh` (long-lived). On 401 → call `/v1/auth/refresh`; on refresh failure → re-login.
4. Cache the workspace UUID from `/v1/me` and inject `X-Workspace-ID` on every request.

5.2 Reference TypeScript client (sketch)

```ts
class CarmicClient {
  constructor(
    private base = 'https://api.carmic.pro/api',
    private email: string,
    private password: string,
  ) {}

  private access?: string
  private refresh?: string
  private workspaceId?: string

  async ensureSession() {
    if (this.access) return
    const r = await fetch(`${this.base}/v1/auth/login`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email: this.email, password: this.password }),
    }).then(r => r.json())
    this.access = r.data.access
    this.refresh = r.data.refresh

    const me = await this.get('/v1/me')
    this.workspaceId = me.data.workspaces[0].id
  }

  private headers() {
    return {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${this.access}`,
      'X-Workspace-ID': this.workspaceId!,
    }
  }

  async createProject(input: { filename: string; filesize: number; duration:
        number; language?: string }) {
    await this.ensureSession()
    const r = await fetch(`${this.base}/v1/projects/create`, {
      method: 'POST', headers: this.headers(),
      body: JSON.stringify(input),
    }).then(r => r.json())
    return r.data as { project_id: string; upload_url: string; object_key: string }
  }

  async putToR2(uploadUrl: string, body: Buffer | Blob, contentType = 'video/mp4')
        {
    await fetch(uploadUrl, {
      method: 'PUT',
      headers: { 'Content-Type': contentType },
      body,
    })
```

```
  }

  async startProcessing(projectId: string) {
    await fetch(`${this.base}/v1/projects/${projectId}/process`, {
      method: 'POST', headers: this.headers(),
    })
  }

  async waitForClips(projectId: string, pollMs = 10_000): Promise<string[]> {
    while (true) {
      const s = await this.get(`/v1/projects/${projectId}/status`)
      if (s.data.status === 'CLIPS_READY' || s.data.status === 'COMPLETED') break
      if (s.data.status?.startsWith('FAILED')) throw new Error(s.data.status)
      await new Promise(r => setTimeout(r, pollMs))
    }
    const p = await this.get(`/v1/projects/${projectId}`)
    return p.data.clips.map((c: any) => c.id)
  }

  async render(clipId: string) {
    return fetch(`${this.base}/v1/clips/${clipId}/render`, {
      method: 'POST', headers: this.headers(),
    }).then(r => r.json())
  }

  // ... waitForRender, downloadMp4, exportPremiere, etc.
}
```

5.3 Quotas you need to plan around

Constraint	Limit
Project creations	5 per minute per workspace
NLE exports	5 per minute per workspace
Concurrent ingestion videos	1 (FREE/STANDARD/PROFESSIONAL), 2 (BUSINESS), unlimited (ENTERPRISE)
Max upload size	5 GB
Processing minute pool	Per workspace tier — top up via Stripe

If ApexStream submits a bulk batch, queue your own jobs and pace project creation ≤ 5/min. Use the SSE stream rather than polling whenever you can — it keeps latency under a second on the `CLIPS_READY` transition.

5.4 Driving the editor without a browser

Yes — the editor UI is a thin front over the same `PUT /v1/clips/{id}` endpoint. ApexStream can:

- Pre-fetch the default `template_config` from `GET /v1/clips/{id}`.
- Mutate any of the fields in §3.5.1 (e.g. set `caption_style: "HORMOZI"`, swap fonts/colors, add `b_roll[]`, set `elements.watermark`).
- Save with `PUT /v1/clips/{id}` then call `POST /v1/clips/{id}/render`.

The only flows that today still depend on the UI are:

- **OAuth-based social uploads** — `POST /v1/social/connect/{platform}` returns a consent URL the end-user has to visit in a browser. ApexStream can complete the OAuth callback server-side, but the consent click itself must happen in a user agent.

Everything else — upload, analyze, edit template, render, NLE export, download — is fully scriptable.

---

6. Editor-via-API: feasibility summary for ApexStream

Capability	Available via API?	Notes
Upload long-form video	✅	Direct R2 PUT via presigned URL
Trigger AI clip detection	✅	`/projects/{id}/process`
Read transcript + word timings	✅	`GET /clips/{id}` returns `transcript.words[]`
Read AI virality scores	✅	`clips[*].virality_score` + `reasoning`
Read AI face-tracking matrix	✅	`advanced_spatial_matrix`
Read AI auto-director timeline	✅	`ai_suggested_layout_timeline`
Save custom layout / captions / B-roll	✅	`PUT /clips/{id}` with `TemplateConfig`
Render final 9:16 (or other) MP4	✅	`POST /clips/{id}/render` → poll → `GET /clips/{id}/video`
Export to Premiere / DaVinci / FCPX / AE	✅ (Pro tier)	`POST /clips/{id}/exports`
Live progress	✅	SSE via `/sse-ticket` exchange
Webhooks for "render complete"	❌	No outbound webhooks today — poll or use SSE
Service-token / API-key auth	❌	JWT only — provision a service user
Programmatic social-platform OAuth consent	⚠️	Initial consent click requires a browser

Verdict: ApexStream can drive the entire Carmic editor end-to-end via this API. The two real integration costs are (a) the user-bearer-only auth model — handled by a dedicated service user — and (b) the absence of completion webhooks — handled by the SSE streams.