DOCUMENTATION CELLULE-PRO

Référence API

Endpoints OpenAI-compatible · tokens sk-cellule-*

cellule.ai · pro.cellule.ai
2026-05-18

Table des matières

Table des matières **2**
0.1 Cellule-PRO — Référence API 3
Sommaire 3
1. Base URL et versioning 3
2. Authentification 3
3. Conventions 4
4. Endpoints par domaine 6
5. Codes d'erreur 12
6. Exemples d'intégration 13
7. Swagger / ReDoc interactifs 14
Références croisées 15

0.1 Cellule-PRO — Référence API

Public : intégrateur applicatif, dev qui branche un client métier sur Cellule-PRO (application interne, workflow automation, assistant dédié).

Principe : Cellule-PRO est **API-first**. Pas d'UI chat user-facing. Tout passe par ces endpoints. Cf. `feedback_enterprise_api_first_no_limits`.

État : squelette créé session 2026-04-21 (chantier RAG Phase 6). Sections à compléter au fil des phases 1-5.

Sommaire

1. Base URL et versioning
2. Authentification
3. Conventions
4. Endpoints par domaine
5. Codes d'erreur
6. Exemples d'intégration
7. Swagger / ReDoc interactifs

---

1. Base URL et versioning

- **Base URL par défaut** : `https://<pool-host>/` où `<pool-host>` = l'entrée unifiée nginx du cluster (`:18080` en interne, derrière TLS client / Cloudflare en prod).
- **Pattern path** : toutes les routes publiques sont préfixées `/v1/`.
- **Versioning** : changement de préfixe `v1` → `v2` seulement en cas de breaking change. Les ajouts (nouveau champ optionnel, nouveau endpoint) restent sous `v1` sans incrément.
- **Swagger auto-généré** : `/docs` et `/redoc` (cf. §7).

---

2. Authentification

2.1 v1 — Header `X-Cellule-User`

État actuel : auth simplifiée, le pool fait confiance au user_id transmis par la gateway en amont. Chaque requête doit porter :

```
X-Cellule-User: <user_id>
```

Sans ce header → `401`. Ce design suppose que l'authentification réelle (SSO / token bearer) est assurée par un reverse-proxy en amont du pool (nginx entry, Cloudflare Access, tunnel VPN). Le pool traite le user_id comme source de vérité.

Ne jamais exposer directement le pool sur Internet sans reverse-proxy qui authentifie au préalable.

2.2 v1.5 — Token bearer interne (planifié)

Prévu : `Authorization: Bearer <token>` branché sur le module session du pool public. Rotation, grace period, révocation seront documentés à l'implémentation. Pas livré en v1.

2.3 Rôles intra-projet

L'authentification identifie **qui** (user_id). L'autorisation sur une ressource projet dépend du rôle du user dans `project_members` :

Rôle	Peut
`viewer`	GET /projects, /documents, /search
`editor`	+ POST /documents, /memory
`owner`	+ PATCH/DELETE project, invite/remove members, GET /audit

Hiérarchie : `owner > editor > viewer`. Un endpoint qui exige `min_role=editor` accepte `editor` et `owner`, refuse `viewer`.

Pas de scope pool-admin séparé dans cette API : l'admin infrastructure passe par `/v1/admin/*` (dashboard), pas par les endpoints projets.

---

3. Conventions

3.1 Format

- Requêtes : JSON UTF-8, `Content-Type: application/json`
- Réponses : JSON UTF-8
- Dates : ISO 8601 UTC (`2026-04-21T18:30:00Z`)

3.2 Pagination

- `GET /v1/projects` : retourne tous les projets visibles (pas de pagination v1, liste bornée par `DEFAULT_MAX_PROJECTS_PER_USER=20`).
- `GET /v1/projects/{id}/documents` : liste complète, tri `uploaded_at DESC`.
- `GET /v1/projects/{id}/memory` : liste bornée 200 lignes les plus récentes.
- `GET /v1/projects/{id}/audit?limit=<1..500>` : seul endpoint avec pagination explicite via query param (défaut 100).

Pas de curseur ni de header `X-Total-Count` en v1. Volumes plus grands → utiliser `/v1/projects/{id}/search` avec filtre sémantique.

3.3 Rate limits

Rate limit applicatif sur les endpoints mutants sensibles :

- `POST /v1/projects` : max `DEFAULT_MAX_PROJECTS_PER_USER=20` projets actifs simultanés par user. Dépassement → `429`.

Pas de quota global par user ni de header `X-RateLimit-*` v1. Si protection anti-abus nécessaire → nginx entry ou middleware gateway en amont.

3.4 Idempotence

- **Table** `project_documents` : `UNIQUE (project_id, content_hash)` — deux POST identiques sur le même document hash produisent un seul row. Le second renvoie `409 Conflict`.
- **Table** `project_memories` : `UNIQUE (project_id, fact_hash)` — même logique.
- **Ingest RAG** (`_insert_chunks`) : DELETE puis INSERT dans une transaction → ré-ingest du même document remplace les chunks existants atomiquement.

Pas de header `Idempotency-Key` v1 ; les contraintes UNIQUE au niveau schema jouent ce rôle.

3.5 Résilience SDK — pattern retry (chantier RAID-2)

Le pool maintient une queue locale `pending_jobs` à durée de vie courte (TTL 5 minutes via `cleanup_expired_jobs`). Cette queue est **strictement locale** par design — elle n'est PAS répliquée cross-pool, conformément au pattern industrie (OpenAI, Anthropic, AWS Bedrock SDK).

Conséquence : si un pool tombe entre l'acceptation d'une requête inférence (`POST /v1/chat/completions`) et son exécution, la requête peut être perdue. Le client doit implémenter un retry idempotent.

Pattern recommandé côté SDK / client :

```python
import time
import httpx
from httpx import HTTPStatusError, TimeoutException

def chat_completion_with_retry(client, payload, max_retries=3):
    """Exponential backoff sur 503/504/timeout. Pattern OpenAI SDK."""
    last_error = None
    for attempt in range(max_retries):
        try:
            r = client.post("/v1/chat/completions", json=payload, timeout=120)
            r.raise_for_status()
            return r.json()
        except HTTPStatusError as e:
            last_error = e
            if e.response.status_code in (503, 504, 502):
                # Pool unavailable / failover en cours / gateway timeout
                wait = 2 ** attempt  # 1s, 2s, 4s
                time.sleep(wait)
                continue
            raise  # 4xx = pas de retry (auth, format, etc.)
        except TimeoutException as e:
            last_error = e
            time.sleep(2 ** attempt)
    raise last_error
```

Pourquoi côté client et pas côté pool : - Latence gossip Ed25519 LAN (~50-200ms) » latence inférence rapide : une réplication de la queue serait plus lente que le job lui-même - Le client connaît le contexte (idempotency naturelle des requêtes chat), peut décider du timeout métier (UI = 30s, batch overnight = 1h) - Lock distribué pour partager la queue = très lourd pour un gain hypothétique (failover réel ~secondes)

Idempotency naturelle : les requêtes `chat/completions` ne sont pas « exécutées 2 fois » de manière problématique — un retry après un timeout de 120s génère simplement une nouvelle réponse (le LLM est non-déterministe par construction). Le client peut afficher la réponse qui arrive en premier.

Pour les workloads strictement idempotents (batch overnight, ingest documents), passer un identifiant client dans `metadata` du payload et dédupliquer côté application.

Ce qui EST répliqué cross-pool en cas de failover (chantier RAID livré rc89→rc92) : - Identités (`accounts`, `api_tokens`, `employees`) — propagation Ed25519 - Mémoire utilisateur (`user_memories`, `agent_episodes`, `conversations`) — gossip memory_replication - Projets RAG (`projects`, `project_documents`, `project_document_chunks`, `project_memories`, `project_conversations`, `project_members`) - Audit des inférences (`jobs_audit`) — historique opérationnel cross-pool - Catalogue modèles (`pool_models`, `pool_model_assignments`) - Configuration (`moe_local_config`, `pool_config`, `signup_config`)

Ce qui n'est PAS répliqué (par design) : - `pending_jobs` (queue inférence locale 5min TTL — voir ci-dessus) - `jobs` (historique opérationnel + embeddings smart routing — local par pool, mais audit minimal propagé via `jobs_audit`) - `workers`, `worker_benchmarks`, `worker_tasks` (workers physiquement attachés à un pool unique) - `sessions`, `routing_feedback` (transient)

---

4. Endpoints par domaine

4.1 Projects (chantier #10)

Méthode	Path	Rôle min	Description
`POST`	`/v1/projects`	auth seul	Créer un projet (owner = auteur)
`GET`	`/v1/projects`	auth seul	Lister les projets dont je suis membre
`GET`	`/v1/projects/{pid}`	viewer	Détail projet
`PATCH`	`/v1/projects/{pid}`	owner	Modifier name/description/settings
`DELETE`	`/v1/projects/{pid}`	owner	Archiver (soft delete, status=‘archived’)
`POST`	`/v1/projects/{pid}/members`	editor+	Inviter un user
`GET`	`/v1/projects/{pid}/members`	viewer	Lister les membres
`DELETE`	`/v1/projects/{pid}/members/{uid}`	owner	Retirer un membre
`PATCH`	`/v1/projects/{pid}/members/{uid}`	owner	Changer le rôle d’un membre

```
{
  "name": "Cabinet-Martin-Dossier-X",
```

```
  "description": "Espace collaboratif dossier client Martin SA"
}
```

POST `/v1/projects` — **body**

- `name` : 1..256 chars, pas de saut de ligne, unicité non garantie globalement (deux projets peuvent avoir le même nom si owners différents).
- `description` : optionnel, texte libre.

Réponses : - `201` → `{ project_id, name, owner_user_id, created }` - `400` → nom invalide - `429` → user a déjà `DEFAULT_MAX_PROJECTS_PER_USER=20` projets actifs

```
{
  "user_id": "bob",
  "role": "editor"
}
```

POST `/v1/projects/{pid}/members` — **body**

- `role` `("owner", "editor", "viewer")`, défaut `editor` si omis
- Signé Ed25519 par le pool hôte avant gossip — invariant chantier #10

Réponses : `201` / `400` (rôle invalide) / `403` (pas editor+) / `404` (projet inexistant) / `409` (déjà membre).

4.2 Documents

Méthode	Path	Description
`POST`	`/v1/projects/{id}/documents`	Upload metadata document (binaire v1.5 multipart)
`GET`	`/v1/projects/{id}/documents`	Lister documents + statut indexation
`GET`	`/v1/projects/{id}/documents/{doc_id}`	Détail + chunks_count
`DELETE`	`/v1/projects/{id}/documents/{doc_id}`	Supprime (CASCADE chunks)

Rôle minimum requis : `editor` pour POST/DELETE, `viewer` pour GET.

POST `/v1/projects/{id}/documents` — **attach metadata** Body JSON :

```
{
  "filename": "contrat-client-acme.pdf",
  "content_hash": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "size_bytes": 123456,
```

```
  "mime_type": "application/pdf"
}
```

Réponses : - `201` → `{ project_id, document_id, filename, hosting_pool_id }` - `400` → champ manquant ou invalide (`content_hash` doit être SHA256 hex 64 chars, `size_bytes >= 0`) - `403` → membre de rôle < `editor` - `413` → `size_bytes > 50 MB` (cap verrouillé v1) - `415` → `mime_type` hors whitelist (TXT / MD / PDF / DOCX / ODT uniquement)

Pas d'upload binaire en v1 — ce endpoint attache uniquement la metadata. Le binaire sera uploadé par un endpoint multipart v1.5 séparé. Le RAG indexe dès que le binaire arrive.

Cycle de vie `indexed_status` Champ sur chaque document :

Valeur	Sens
`NULL`	Doc legacy (uploadé avant le rollout RAG) ou binaire pas encore arrivé
`pending`	Ingest dispatché, en cours d'exécution background
`done`	Chunks + embeddings stockés dans `project_document_chunks`, `chunks_count > 0`
`skipped`	Format non supporté / taille dépassée / texte vide / lib extract absente
`failed`	Erreur pipeline (embedder indispo, DB error) — retry possible via re-POST

Réponse GET /documents (exemple, champs indexation inclus) :

```
{
  "project_id": "proj_abc123",
  "documents": [
    {
      "id": 42,
      "filename": "contrat-client-acme.pdf",
      "size_bytes": 123456,
      "mime_type": "application/pdf",
      "indexed_status": "done",
      "chunks_count": 37,
      "indexed_at": "2026-04-21T18:45:12Z",
      "uploaded_at": "2026-04-21T18:44:58Z"
    }
  ]
}
```

Polling du statut Le client peut poller `GET /v1/projects/{id}/documents` après un upload binaire pour voir la bascule `pending → done`. Temps typique : 1-10s pour un PDF 5 MB (embedding 384d MiniLM sur CPU dormant, cf. §1 ADMIN_GUIDE).

> *À compléter quand l'endpoint multipart binaire v1.5 est implémenté : body form-data, streaming upload, reprise sur interruption.*

4.3 Search RAG (chantier RAG)

Méthode	Path	Description
POST	/v1/projects/{id}/search	KNN cosine sur project_document_chunks

Rôle minimum requis : viewer.

```
{
  "q": "retrouve les jugements similaires au dossier Martin SA",
  "k": 10,
  "sources": ["doc"]
}
```

Body

Champ	Type	Défaut	Contrainte
q	string	— (requis)	non vide, tronqué à 2000 chars
k	int	10	1 k 50
sources	list[string]	["doc"]	{"doc", "memory", "conv"}

Note v1 : seule la source "doc" est implémentée. "memory" et "conv" sont acceptés mais retournent vide avec un notes explicite — en attente de l'ALTER project_memories → vector(384) (v1.5).

```
{
  "project_id": "proj_abc123",
  "q": "retrouve les jugements similaires au dossier Martin SA",
  "k": 10,
  "sources": ["doc"],
  "results": [
    {
      "source": "doc",
      "project_id": "proj_abc123",
      "document_id": 42,
      "chunk_id": 137,
      "chunk_idx": 4,
      "content_text": "… texte du chunk le plus proche sémantiquement …",
      "similarity": 0.87,
      "filename": "jugement-martin-2024.pdf",
      "mime_type": "application/pdf"
    }
  ],
  "notes": []
}
```

Réponse 200

- `similarity` : cosine similarité $[0, 1]$, 1 = identique
- Seuil min par défaut : `min_similarity = 0.3` (silence si < seuil)
- Tri décroissant sur `similarity`

Codes d'erreur

Code	Cas
400	`q` vide, `k` hors [1,50], `sources` contient une valeur inconnue
401	header `X-Cellule-User` manquant
403	utilisateur non-membre du projet
404	`project_id` inexistant
410	projet archivé

Invariant isolation projet (non-négociable) Le filtre `WHERE project_id = $1` est **toujours** présent dans le SQL (vérifié par test source-level + validation molecule-guardian). **Aucun leak cross-project possible par design** — compliance attorney-client privilege + RGPD data segregation.

Exemple curl

```
curl -X POST https://pool.example.com/v1/projects/proj_abc123/search \
  -H "X-Cellule-User: alice" \
  -H "Content-Type: application/json" \
  -d '{"q": "clauses de non-concurrence", "k": 5}'
```

4.4 Chat completions

Endpoint `POST /v1/chat/completions` — schéma 100 % compatible OpenAI. Tout client qui parle l'API OpenAI fonctionne sans modification : opencode, Cursor, Continue.dev, OpenWebUI, openai-py, aichat, llm, curl…

Authentification

```
Authorization: Bearer sk-cellule-<token>
```

Le token est généré côté user (onglet « Utiliser l'API ») ou côté admin. Il authentifie l'appel **et** sert de clé crypto pour chiffrer la mémoire long-terme (RAG par-utilisateur).

Payload minimal

```
{
  "model": "CELLULE",
  "messages": [{"role": "user", "content": "Bonjour"}]
}
```

Le pool expose un seul modèle virtuel `CELLULE`. Le routage interne (smart routing tier-aware, MoE sharding, forwarding cross-pool) est **transparent** côté client : il choisit le worker le plus pertinent selon le prompt (Coder pour code, long-contexte pour gros prompts, 9B pour chat rapide, etc.) et forward vers un autre pool LAN si nécessaire.

Streaming SSE : `"stream": true` dans le payload. Format event stream OpenAI standard. Marche aussi quand la requête est forwardée vers un peer (rc40+).

Tool-calling : `tools` field OpenAI-compat. Marche transparent avec opencode/Cursor/Continue — ces clients envoient leurs tools dans le payload, le LLM les utilise (Read/Glob/Edit/Bash sur le poste local du dev).

Mémoire long-terme automatique (RAG) Chaque user (token `sk-cellule-*`) a une **mémoire chiffrée** côté pool, retrieved automatiquement à chaque appel selon la similarité sémantique avec le dernier message user.

Activer / désactiver : onglet « Préférences » de l'espace user. Persistant mode = ON par défaut pour les tokens api_gateway. Mode privé = pas de RAG, pas de save de conversation.

Pattern `[MEMORIZE: fait]`

Pour stocker explicitement un fait dans la mémoire long-terme, demander au LLM de terminer sa réponse par :

```
[MEMORIZE: votre fait à mémoriser]
```

Le pool intercepte ce tag dans la réponse, l'embed avec le token user (chiffrement par-utilisateur), et le strip avant de renvoyer au client. Le fait est ensuite retrievable en session suivante (autre process, autre machine, autre conversation) tant que le token est le même.

Comportement cross-session

- Session N : conversation normale, ou explicit `[MEMORIZE: ...]`. La conversation est embed (auto-summarization) **et** les `[MEMORIZE: ...]` sont persistés.
- Session N+1 (autre process, sans `--continue`) : nouveau prompt user. Le pool fait un retrieval RAG sur le user's facts, inject les facts pertinents, le LLM répond en s'appuyant dessus.

Validé E2E avec opencode (process séparés, no `--continue`) en mode tool-calling — voir `feedback_killer_commerc` côté équipe interne.

Exemple

```
# Session 1 (terminal A) : planter un fait
curl $POOL/v1/chat/completions \
  -H "Authorization: Bearer sk-cellule-..." \
  -d '{"model":"CELLULE","messages":[{"role":"user",
       "content":"Note pour demain : le bug est en parse_log() ligne 42, current_section non init. Termine
```

```
# Session 2 (terminal B, plus tard, autre laptop) : retrouver
curl $POOL/v1/chat/completions \
  -H "Authorization: Bearer sk-cellule-..." \
  -d '{"model":"CELLULE","messages":[{"role":"user",
        "content":"Quel bug devais-je corriger ?"}]}'

# → "UnboundLocalError ligne 42 dans parse_log() quand le log
#    est vide. Cause probable: current_section non initialisée."
```

Tools & mémoire en mode agent (opencode/Cursor/Continue) Quand le payload contient `tools`, le pool injecte le contexte RAG **dans le dernier user message** (au lieu du system prompt) pour forcer la saillance face aux 5+ KB d'instructions tool-calling envoyé par les clients agent. Comportement transparent — les tools du client fonctionnent normalement, mais le LLM consulte les facts user avant de tool-call exploratoire (Glob/Read/grep).

Sans cette injection ciblée, le LLM agent tend à privilégier ses tools sur le system prompt RAG. Avec, le killer feature « mon LLM se souvient de moi » fonctionne aussi en mode agent. Implémenté pool.py rc46.

4.5 Admin

Les endpoints `/v1/admin/*` sont réservés au dashboard operator interne. Documentation séparée `ADMIN_GUIDE.md`. Non destinés à l'intégration applicative client.

---

5. Codes d'erreur

5.1 HTTP

Code	Cas
400	Body invalide (format, champs manquants)
401	Token absent ou invalide
403	Token valide mais scope insuffisant
404	Ressource inexistante (ou invisible pour ce token)
409	Conflit (ex. nom projet déjà pris)
413	Upload > 50 MB
415	Format document non supporté
422	Validation Pydantic échouée
429	Rate limit dépassé
500	Erreur interne (avec trace_id dans response body)

5.2 Structure response erreur

```
{
  "error": {
    "code": "project_not_found",
```

```
    "message": "Project 'xyz' does not exist or is not visible to you.",
    "trace_id": "abc123"
  }
}
```

À compléter : liste exhaustive des `code` métier par domaine.

---

6. Exemples d'intégration

6.1 curl — workflow complet projet + RAG

```
BASE=https://pool.example.com
USER=alice

# 1. Créer un projet
curl -X POST $BASE/v1/projects \
  -H "X-Cellule-User: $USER" \
  -H "Content-Type: application/json" \
  -d '{"name": "Cabinet-Martin-Dossier-X"}'
# → 201 {"project_id": "proj_abc123", ...}

PID=proj_abc123

# 2. Attacher un document (metadata v1)
curl -X POST $BASE/v1/projects/$PID/documents \
  -H "X-Cellule-User: $USER" \
  -H "Content-Type: application/json" \
  -d '{
    "filename": "contrat-martin-2024.pdf",
    "content_hash": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
    "size_bytes": 123456,
    "mime_type": "application/pdf"
  }'
# → 201 {"document_id": 42, ...}

# 3. Poller le statut d'indexation
curl -s $BASE/v1/projects/$PID/documents \
  -H "X-Cellule-User: $USER" | jq '.documents[0].indexed_status'
# → "pending" puis "done"

# 4. Rechercher sémantiquement dans le projet
curl -X POST $BASE/v1/projects/$PID/search \
  -H "X-Cellule-User: $USER" \
  -H "Content-Type: application/json" \
  -d '{"q": "clauses de non-concurrence 24 mois", "k": 5}'
# → 200 {"results": [{"source":"doc", "content_text": "...", "similarity": 0.87, ...}]}
```

6.2 Python (httpx)

```python
import httpx

BASE = "https://pool.example.com"
HEADERS = {"X-Cellule-User": "alice"}

async def search_project(pid: str, query: str, k: int = 10) -> list[dict]:
    async with httpx.AsyncClient(base_url=BASE, headers=HEADERS) as client:
        r = await client.post(
            f"/v1/projects/{pid}/search",
            json={"q": query, "k": k},
        )
        r.raise_for_status()
        return r.json()["results"]
```

6.3 TypeScript (fetch)

```typescript
const BASE = "https://pool.example.com";
const HEADERS = { "X-Cellule-User": "alice", "Content-Type": "application/json" };

async function searchProject(pid: string, q: string, k = 10) {
  const r = await fetch(`${BASE}/v1/projects/${pid}/search`, {
    method: "POST",
    headers: HEADERS,
    body: JSON.stringify({ q, k }),
  });
  if (!r.ok) throw new Error(`${r.status} ${await r.text()}`);
  return (await r.json()).results;
}
```

6.4 Intégrations framework RAG (LangChain / LlamaIndex)

Cellule-PRO expose un endpoint `POST /v1/projects/{id}/search` compatible avec un wrapper Retriever custom. L'implémentation d'un wrapper officiel est tracée dans `project_todo_client_framework_integ`
Pas livré v1.

---

7. Swagger / ReDoc interactifs

FastAPI expose automatiquement deux UIs auto-générées :

- **Swagger UI** : `https://<pool-host>/docs`
- **ReDoc** : `https://<pool-host>/redoc`
- **OpenAPI JSON** : `https://<pool-host>/openapi.json`

Pour un "try it out" : ajouter le header `X-Cellule-User: alice` dans l'interface Swagger avant de déclencher une requête. Génération de clients typés : `openapi-generator-cli generate -i openapi.json -g <target>` (python, typescript-fetch, rust…).

Références croisées

- ADMIN_GUIDE.md — déploiement, topologie, opérations
- BOUNDARY.md — contrat public PRO (quels endpoints sont hérités du pool public)
- SECURITY_MODEL.md — modèle d'authn/authz
- CHANTIER_10_MODE_PROJET.md — genèse du scope projet
- CHANTIER_RAG_PROJECT_DOCUMENTS.md — genèse du search RAG