Architecture

API REST

API REST

> *Dernière mise à jour : 2026-05-10*

34 endpoints, tous prefixés `/api`. Auth par cookie de session (HTTP-only). Validation Zod systématique.

Auth

Méthode	Path	Auth	Description
POST	`/api/auth/register`	Non	Crée user (username, email, password). Hash argon2id. Statut `pending`. Notif admin email
POST	`/api/auth/login`	Non	Login → session 7j. Rate-limit 5 tentatives / 15 min
POST	`/api/auth/logout`	Oui	Invalide la session
GET	`/api/confirm-user/:token`	Non	Email confirmation one-time → role `user`

Games

Méthode	Path	Auth	Description
GET	`/api/games`	Confirmed	Liste tous jeux + metadata BGG
GET	`/api/games/:id`	Confirmed	Détail jeu (rules_language, hasCardDatabase, etc.)

Méthode	Path	Auth	Description
GET	`/api/games/:id/pdf`	Confirmed	Stream PDF (`Content-Type: application/pdf`)
GET	`/api/games/:id/page-image/:page`	Confirmed	PNG 300 DPI page N (rendu via `pdftoppm`)
GET	`/api/games/search?q=`	Confirmed	Recherche fulltext (LIKE)
POST	`/api/games/ingest`	Confirmed + canAddGames	Multipart : PDF + metadata. Si `scheduled_start_at` : queue scheduled, sinon démarrage immédiat
DELETE	`/api/games/:id`	Admin	Supprime jeu, questions, purge collection Qdrant
DELETE	`/api/games/:id/scheduled`	Confirmed + canAddGames	Annule ingestion scheduled. 409 si pas en `scheduled`

Ask (RAG)

Méthode	Path	Auth	Description
POST	`/api/ask/retrieve`	Confirmed	Retrieval seul (chunks sans génération) — pour évaluation
POST	`/api/ask/stream`	Confirmed	RAG streaming SSE (question → retrieval → Claude). Body : `{ game_id, question, extensions, history, cardMentions, stickyCardMentions }`
GET	`/api/ask/:questionId`	Confirmed	Récupère la réponse persistée (fallback SSE après crash connexion)
PUT	`/api/ask/:questionId/feedback`	Confirmed	Vote pouce ↑↓ + comment

Cards

Méthode	Path	Auth	Description
GET	`/api/cards/search?gameId=&q=&limit=`	Confirmed	Autocomplete par collection (BM25 ou full-text)

Méthode	Path	Auth	Description
GET	`/api/cards/image/:pointId?w=&gameId=`	Confirmed	Proxy image cachée (sharp resize, fallback CDN)

Decks

Méthode	Path	Auth	Description
POST	`/api/decks/parse`	Confirmed	Parse decklist texte → pointIds Qdrant. Whitelist `flesh-and-blood-cards`. Rate-limit 10/min

BGG

Méthode	Path	Auth	Description
GET	`/api/bgg/hot`	Confirmed	Top 20 jeux BGG (cache 6h)
GET	`/api/bgg/search?q=`	Confirmed	Recherche BGG XML API
GET	`/api/bgg/game/:bggId`	Confirmed	Détail jeu BGG
GET	`/api/bgg/game/:bggId/expansions`	Confirmed	Extensions d'un jeu BGG

Lorcana

Méthode	Path	Auth	Description
GET	`/api/lorcana-symbols/:symbolId`	Confirmed	SVG symboles spécialisés Lorcana

Admin

Méthode	Path	Auth	Description
GET	`/api/admin/health`	Admin	Health Qdrant, TEI, reranker, Claude SSH, SMTP + stats

Méthode	Path	Auth	Description
GET	`/api/admin/users`	Admin	Liste users (id, username, role, canAddGames)
DELETE	`/api/admin/users/:id`	Admin	Supprime user + questions, réassigne ses jeux à l'admin
POST	`/api/admin/users/:id/set-can-add-games`	Admin	Toggle `canAddGames`
POST	`/api/admin/confirm-user/:userId`	Admin	Force confirmation user pending → role `user`
GET	`/api/admin/feedback?gameId=&vote=&from=&to=&page=`	Admin	Pagine feedbacks filtrés
GET	`/api/admin/feedback/:id`	Admin	Détail feedback + diagnostics complets
POST	`/api/admin/feedback/export`	Admin	Export CSV feedbacks filtrés
POST	`/api/admin/games/:id/sync-cards`	Admin	Force sync collection Qdrant vs source
GET	`/api/admin/cards/list`	Admin	Liste collections + counts
POST	`/api/admin/send-test-email`	Admin	Test SMTP
POST	`/api/admin/send-password-reset/:userId`	Admin	Force reset email

Health

Méthode	Path	Auth	Description
GET	`/api/health`	Non	`{ status: 'ok', timestamp }` (Docker healthcheck)

Patterns globaux

- **UUID v4** partout (`game_id`, `question_id`, `user_id`)
- **Rate-limit ask** : 20 questions/min/user
- **Rate-limit deck parse** : 10/min/user
- **Rate-limit login** : 5 tentatives → 15 min cooldown / IP
- **Limites taille** : question ≤500 chars, comment ≤1000 chars, decklist ≤50 000 chars
- **SSE events** : `meta` (1er, porte questionId), `phase`, `context`, `token`, `done`, `error`, `quota_pause`, `heartbeat` (8s)

Pas d'OpenAPI/Swagger — le tableau ci-dessus est la source de vérité.

Architecture — Backend

Architecture — Backend

> *Dernière mise à jour : 2026-05-10*

Couches

```
src/
├── routes/           # Contrats HTTP (Hono), validation Zod, auth
├── handlers/         # Logique métier pure (Phase 4 MVC)
├── services/         # Domaine RAG / Qdrant / TEI / Claude / cards / méta / OCR
├── repositories/     # Data access Drizzle — SEUL endroit qui importe drizzle-orm
├── middleware/       # auth.ts (sessions, roles), security.ts (CORS, headers)
├── lib/              # logger, schemas Zod partagés, utils, with-timeout
├── cron/             # ingest-scheduler, meta-sync, forum-sync
├── config.ts         # Validation Zod env vars (boot-time)
├── schema.ts         # Tables Drizzle (users, games, questions)
├── db.ts             # Connexion SQLite + migrations
├── types.ts          # Types globaux (SessionUser, AppEnv)
└── index.ts          # App Hono, montage routes, CORS, init
```

Règles d'archi

1. **`routes/`** ne fait que parser/valider l'entrée (Zod), appeler un handler ou un service, renvoyer la réponse Hono.
2. **`handlers/`** ne connaît pas Hono. Reçoit des données validées + dépendances, retourne un `Result` discriminé. Pour les erreurs attendues : `type DeleteGameResult = { ok: true } | { ok: false; status: 404; error: string }`.
3. **`services/`** : logique métier, pas de DB directe — passe par les repos.

4. **`repositories/`** : SEUL endroit qui importe `db`, `drizzle-orm` ou les tables du schéma. Nommer les fonctions par intention métier (`getByBggId`, `setIngestStatus`).
5. **`config.ts`** : seul endroit qui peut lire `process.env.X`. Tous les autres fichiers font `import { config }`.
6. **`logger.ts`** : seul endroit où `console.*` est autorisé. Convention : préfixer le scope dans le message (`logger.info('[meta-sync] ...')`).
7. **Aucun fichier > 350 lignes**. Si un fichier enfle, le découper : `types.ts` pour les interfaces, un fichier par responsabilité, `index.ts` comme barrel.
8. **Imports services** : toujours via le barrel (`from '../services/rag/index.js'`), jamais un sous-module direct.

Routes (vue d'ensemble)

Préfixe	Module
`/api/auth/*`	`routes/auth.ts` — register, login, logout, change password, reset
`/api/games/*`	`routes/games.ts` — CRUD jeux, PDF, ingest, page-image
`/api/ask/*`	`routes/ask.ts` — RAG retrieve + stream + feedback
`/api/cards/*`	`routes/cards.ts` — autocomplete, image proxy
`/api/decks/parse`	`routes/decks.ts` — import deck FAB
`/api/bgg/*`	`routes/bgg.ts` — search, hot, expansions
`/api/lorcana-symbols/*`	`routes/lorcana-symbols.ts`
`/api/admin/*`	`routes/admin.ts` — health, users, feedback, sync cards
`/api/health`	`routes/health.ts` — pour healthcheck Docker
`/api/confirm-user/:token`	`routes/auth.ts` — lien email confirmation

Tableau exhaustif des 34 endpoints : voir `architecture/api-rest.md`.

Middleware

- **`auth.ts`** :
  - `requireAuth` : 401 si pas de session
  - `requireConfirmed` : 403 si role='pending'
  - `requireAdmin` : 403 si role≠'admin'
  - `requireCanAddGames` : 403 si user.canAddGames=false (admins bypass)
- **`security.ts`** : CORS whitelist (origines exactes + CIDR IPv4 LAN), headers sécurité de base. La majorité des headers (HSTS, CSP) sont gérés par NPM en amont.

Services principaux

```
services/
├── qdrant.ts            # Client Qdrant (search, upsert, scroll, health)
├── tei.ts               # Client TEI bge-m3
├── reranker.ts          # Client TEI reranker
├── claude-ssh.ts        # SSH Claude Code avec streaming JSON
├── claude-local.ts      # Mode dev local (bypass SSH)
├── claude-quota.ts      # Détection ClaudeQuotaError + parse resetAt
├── validate-model.ts    # Regex stricte modèles Claude (anti-injection shell)
├── bm25.ts              # BM25 sparse retrieval (Qdrant natif)
├── email.ts             # SMTP (notif admin, password reset)
├── bgg.ts + bgg-forums.ts  # API BoardGameGeek + scrape forums Rules
├── hierarchy.ts         # LLM hiérarchie chapter/section
├── conflict-detect.ts   # Détecte conflits extension/base via similarité + LLM
├── contextual-cache.ts  # Cache JSON contextes générés
├── contextual-llm.ts    # LLM contextuel par chunk (Contextual Retrieval B)
├── cards-cache.ts       # Cache mémoire cartes + recherche par nom
├── cards-sync.ts        # Sync collections Qdrant vs sources locales
├── query-expand.ts      # Expansion query (HyDE, synonymes)
├── pdf-images.ts        # Rendu PDF→PNG via pdftoppm (300 DPI)
│
├── chunking/            # Pipeline chunking (chunker, contextual, pdf-extract, types)
├── ocr/                 # Phase 1 : tesseract auto (decideOCR, ocrPages, cache)
├── qdrant/              # Wrappers bas niveau (client, collections, points, payload)
├── ingest/              # Machine à états (queue, coordinator, stages)
├── rag/                 # Pipeline RAG (retrieve, answer, classify, decompose, deckbuilding)
├── cards/sources/       # Normalisations par TCG (magic, lorcana, fab, riftbound, tm, ark-
nova) + registry
└── meta/                # Méta-game (17lands, mtggoldfish, mtgtop8, mobalytics, fabtcg,
riftboundstats, ingest)
```

Patterns importants

- **`withTimeout(promise, ms, label)`** : wrap toute promesse externe (HyDE, decompose, embeddings) pour ne jamais bloquer un boot ou une réponse RAG.

- **Pool SSH parallèle** (10 workers) : pattern standard pour Contextual Retrieval B et conflict detection. Chaque worker check `quotaError` avant de poll la queue.
- **EventPusher** : signature des stages d'ingestion `(game, ..., push: EventPusher) => Promise<...>`. Les stages ne connaissent rien de l'IngestJob.
- **Heartbeat SSE** : tout endpoint streamSSE qui peut rester silencieux >30s ouvre un `setInterval` qui émet `{ type: 'heartbeat' }` toutes les 8s.

Architecture — Frontend

Architecture — Frontend

> *Dernière mise à jour : 2026-05-10*

Structure

```
frontend/src/
├── views/              # 15 pages routables (LoginView, PlayView, AdminView, etc.)
├── components/         # ~50 composants Vue, par domaine
│   ├── admin/          # AdminGamesSection, AdminUsersSection, AdminCardDecksSection,
AdminFeedbackDetail
│   ├── add-game/       # Wizard 3 étapes (BGG search → upload → ingest ritual)
│   ├── play/           # PlayComposer, PlayChatMessages, PlayBackground, PlayHeader
│   ├── deck-import/    # DeckImportForm, DeckImportPreview
│   ├── card-zoom/      # CardZoomStats (modal zoom cartes)
│   ├── home/           # HomeGameGrid, HomeResumeBanner
│   └── (standalone)    # ChatMessage, ArbiterResponse, CardPreview, NavSidebar, etc.
├── composables/        # ~14 composables (useAskStream, useMentionAutocomplete,
useArbiterMarkdown, usePlaySession…)
├── stores/             # Pinia (auth, games, session)
├── services/           # `api.ts` — couche client unique vers backend
├── lib/                # Utilitaires : fab-symbols.ts, mana.ts, riftbound-symbols.ts, lorcana-
symbols.ts
├── assets/             # CSS tokens : tokens.css, main.css, motion-tokens.css, TCG-specific
(fab.css, lorcana.css, riftbound.css)
└── router/index.ts     # Routes + guards auth/admin
```

Routeur

`router/index.ts` (78 lignes). `router.beforeEach` :

- Bloque routes `requiresAuth` si déloggué
- Bloque `requiresAdmin` si non-admin
- Redirige `/pending` si `user.role === 'pending'`

Route	Composant	Auth	Admin
`/login`	LoginView	N	N
`/register`	RegisterView	N	N
`/confirm-success`	ConfirmSuccessView	N	N
`/reset-password`	ResetPasswordView	N	N
`/pending`	PendingView	Y	N
`/`	HomeView	Y	N
`/play`	PlayView	Y	N
`/history`	HistoryView	Y	N
`/add-game`	AddGameView	Y	N
`/ingest/:id`	IngestView	Y	N
`/me`	AccountView	Y	N
`/me/settings`	SettingsView	Y	N
`/admin`	AdminView	Y	Y
`/admin/feedback`	AdminFeedbackView	Y	Y

State management — Pinia stores

`stores/auth.ts`

- `user: { id, username, role, canAddGames } | null`
- Computed : `isLoggedIn`, `isAdmin`, `isPending`, `canAddGames`
- Actions : `checkSession()`, `login()`, `register()`, `logout()`
- Auth via cookie HTTP-only — `credentials: 'include'` dans tous les fetch

`stores/games.ts`

- Cache TTL 60s (collator fr-FR numeric pour tri)
- Dedup appels concurrents (promise inflight)
- `fetch()` retourne la liste triée

- `invalidate()` force reload (après ajout/suppression)

`stores/session.ts`

- Persistence localStorage 6h (`STORAGE_KEY = 'vellum.session.v1'`)
- `activeGame + activeExtensions + messages[]` avec `citations[]`, `cardMentions[]`, `mentionedCards[]`, `synergyCards[]`
- Watch deep avec throttle 200ms localStorage

Composables clés

Composable	Rôle
`useAskStream`	Wrapper SSE `/api/ask/stream` + fallback polling 3s × 15 si SSE casse
`useEventStream`	Bas niveau : fetch streaming + parsing SSE générique (filtre les heartbeats)
`useMentionAutocomplete`	Popover `@`-cartes : debounce 150ms, nav clavier, sync mentions au submit
`useArbiterMarkdown`	marked.parse + injection citations + tokens TCG (mana, FAB, Riftbound, Lorcana)
`usePlaySession`	Orchestre PlayView : SSE, table mode, sticky mentions, deck attachment, card lookup
`useTableMode`	Toggle localStorage `'table-mode'`
`useDeckAttachment`	AttachedDeck { deckName, format, hero, cards[] }
`useCardLookup`	Map cardKey → { id, name, imageUrl, orientation }
`useStickyMentions`	buildStickyMentions() : extrait + dédoublonne, FIFO 20 (80 si deck)
`useDeckImport`	parseDeck(gameId) : POST `/api/decks/parse`

Service `api.ts` (351 lignes)

Couche client unique pour tous les appels backend. Centralise :

- `credentials: 'include'` (cookies)
- 401 → redirect `/login` (sauf sur `/login`, `/register`, `/reset-password`)
- erreur → `throw Error(body.error)`
- OK → `return res.json()`

Types exportés : `Game`, `CardSearchResult`, `MentionedCard`, `DeckParseResponse`, etc.

Design tokens

`frontend/src/assets/tokens.css` (palette OKLCH dark-first) :

- **Bois** `--ref-wood-950` à 600 (table sombre, grain chaud)
- **Feutre** `--ref-felt-950` à 600 (tapis vert)
- **Parchemin** `--ref-parchment-50` à 500 (ivoire chaud)
- **Or** `--ref-gold-300` à 700 (cuivre signature)
- **Meeple** `--ref-meeple-400` à 600 (vert CTA alternatif)
- **Dé/Sang** `--ref-dice-400` à 600 (rouge erreur)

Sémantiques (`main.css @theme`) :

- `--color-bg`, `--color-bg-felt`, `--color-ink`, `--color-primary`, `--color-citation-bg`, `--color-citation-bar`

Build & dev

Commande	Effet
`npm run dev`	Vite dev `:5173` + proxy `/api → :3000`
`npm run build`	type-check (`vue-tsc --build`) + `vite build` (en parallèle)
`npm run build-only`	`vite build` seul (dist/)
`npm run type-check`	`vue-tsc --build` (pas `--noEmit` : compile full)
`npm test` / `npm run test:watch`	Vitest

Vite config : alias `@` → `src/`, plugin `@tailwindcss/vite`, proxy `/api → :3000`.

Pièges connus

1. **Scoped vs Tailwind hidden** : un `display: flex` en CSS scoped écrase le `hidden lg:flex` du template. Préférer Tailwind utilities partout.
2. **`vue-tsc --noEmit`** : passe parfois en local mais foire en CI (`vue-tsc --build`). Tester avec `npm run build`.
3. **Unicode / accents dans card names** : utiliser `normalizeCardKey()` (`useArbiterMarkdown.ts:74-81`) — NFC + lowercase + apostrophe/tiret normalization. Évite

les bugs sur Lorcana / Magic japonais.

4. `\b` **(word boundary) sans flag** `u` : ne reconnaît que `[A-Za-z0-9_]`. Un nom finissant par `é` ne match pas. Utiliser un lookahead Unicode-aware avec flag `u` : `(?=\\s|$|[^\\p{L}\\p{N}_])`.
5. **Modales async** : `CardZoomModal` + `DeckImportModal` chargés via `defineAsyncComponent()` — gain ~50 KB gzip sur le bundle initial PlayView.

Intégrations externes

Intégrations externes

> *Dernière mise à jour : 2026-05-10*

Pour chaque service tiers : à quoi il sert, endpoints consommés, où sont les credentials, quel est le mode dégradé.

Qdrant

- **Rôle** : Vector DB. Une collection par jeu (`rules_<slug>`) + une par TCG (`magic-cards`, etc.).
- **Endpoint** : `QDRANT_URL` (ex. `http://192.168.10.100:6333`)
- **Credentials** : aucun (cluster privé LAN, accès filtré au niveau réseau Docker)
- **Vecteurs** : dense 1024 (bge-m3) + sparse BM25 natif
- **Mode dégradé** : aucun. Si Qdrant est down, le retrieval échoue, l'app renvoie 500.
- **Health** : `GET /collections` (vérifié par `/api/admin/health`)

TEI bge-m3 (embeddings)

- **Rôle** : Génère les embeddings dense 1024 dim pour les questions et chunks.
- **Endpoint** : `TEI_URL` (ex. `http://192.168.10.100:8099`)
- **Credentials** : aucun
- **Modèle** : `BAAI/bge-m3` servi par TEI (HuggingFace)
- **Latence typique** : ~50-100ms pour un batch de 32 sur RTX 3060
- **Mode dégradé** : aucun fallback. Sans TEI : pas de retrieval.
- **Health** : `GET /health`

TEI Reranker bge-v2-m3

- **Rôle** : Cross-encoder qui re-classe les top-50 candidats fusionnés (RRF) avant la génération.
- **Endpoint** : `TEI_RERANKER_URL` (ex. `http://192.168.10.100:8990`)
- **Credentials** : aucun
- **Modèle** : `BAAI/bge-reranker-v2-m3`
- **Mode dégradé** : si reranker down, on retombe sur le score RRF brut (suboptimal mais fonctionnel).

Claude Code CLI (Anthropic)

- **Rôle** : Génération streamée des réponses RAG, HyDE, decompose-query, contextual retrieval, hierarchy, conflict detect, classify intent.
- **Accès** : SSH ed25519 vers la VM `oracle` (compte dédié).
- **Endpoint** : `CLAUDE_SSH_HOST` + `CLAUDE_SSH_USER=oracle` + clé privée `CLAUDE_SSH_KEY_PATH`
- **Credentials Anthropic** : stockés sur la VM dans `/home/oracle/.claude/.credentials.json` (compte personnel Pro/Team)
- **Modèles** :
  - `claude-haiku-4-5-20251001` : HyDE, decompose, classify, contextual, conflict
  - `claude-opus-...` (par défaut) : réponses RAG, deckbuilding
- **Quota** : géré par `services/claude-quota.ts`. Pause auto + reprise programmée à `resetAt + 2 min`.
- **Sécurité** : ForceCommand wrapper `/home/oracle/bin/oracle-claude.sh` valide préfixe `cd /home/oracle/work && claude` . `validateModel()` côté backend bloque l'injection.

BoardGameGeek (XML API v2)

- **Rôle** : Récupère metadata jeux (image, méchaniques, catégories), liste extensions, scrape forums "Rules" pour ingest dans le RAG.
- **Endpoint** : https://boardgamegeek.com/xmlapi2/...
- **Credentials** : `BGG_API_TOKEN` (optionnel). Sans token, rate-limit plus strict mais fonctionnel.
- **Cache** : hot games 6h (mémoire), forums via cron quotidien (configurable)
- **Mode dégradé** : sans BGG, tu peux toujours créer un jeu manuellement (skip étape 1 du wizard).

Sources de cartes par TCG

TCG	Endpoint / source

MTG	https://api.scryfall.com/bulk-data → JSON `all_cards` (téléchargé localement)
Lorcana	https://github.com/LorcanaJSON/LorcanaJSON (raw GitHub, MIT)
FAB	npm `@flesh-and-blood/cards` + `@flesh-and-blood/types` (bundlé dans l'image Docker)
Riftbound	https://riftbound.leagueoflegends.com/en-us/card-gallery (API JSON Riot)
Terraforming Mars	HTML parsing local + cards.json
Ark Nova	JSON local + sprites découpées

Sources méta-game

Source	TCG	Méthode
17Lands	MTG (draft analytics)	API publique
MTGGoldfish	MTG (constructed metagame)	Scrape (Cheerio) + rate-limit 1.5s
MTGTop8	MTG (top 8s tournois)	Scrape + rate-limit 1.5s
Mobalytics	Riftbound (tier list)	Scrape
RiftboundStats	Riftbound (tournois)	API
fabtcg.com	FAB (tournois LSS)	Scrape (header `META_FAB_USER_AGENT` Cloudflare-compatible)

Tous gérés via `src/services/meta/*.ts`. Cron `meta-sync.ts` pilote la fréquence (par défaut hebdo, configurable).

Email (SMTP)

- **Rôle** : Notification admin au register, password reset.
- **Credentials** : `SMTP_HOST`, `SMTP_PORT`, `SMTP_USER`, `SMTP_PASS`, `SMTP_FROM`, `SMTP_SECURE`
- **Mode dégradé** : si `SMTP_HOST` vide, tous les emails sont silencieusement skippés. L'admin doit checker `/admin` à la main.

Reverse proxy (Nginx Proxy Manager)

- **Rôle** : TLS Let's Encrypt + reverse proxy `rules.thymon.fr` → container `:3000`.
- **Pas dans le code** : config NPM via UI, hors scope du repo.
- **Heartbeats SSE 8s** : nécessaires pour ne pas se faire fermer la connexion par le `proxy_read_timeout 60s` par défaut.
- `TRUST_PROXY=true` : Hono fait confiance aux `X-Forwarded-*` headers de NPM.

Aucune autre intégration

- Pas de Sentry / Datadog / monitoring tiers
- Pas de S3 / cloud storage (tout en local sur Unraid)
- Pas de service d'analytics (Umami pourrait être ajouté facilement, pas en place actuellement)
- Pas de webhook entrant (tout est trigger soit user, soit cron)

Modèle de données

Modèle de données

> *Dernière mise à jour : 2026-05-10*

SQLite (Drizzle ORM)

Schéma : `src/schema.ts`. Migrations : `migrations/*.sql` + meta JSON.

Diagramme entité-relation

```
erDiagram
    users ||--o{ games : "addedBy"
    users ||--o{ questions : "userId"
    games ||--o{ games : "parentGameId (extensions)"
    games ||--o{ questions : "gameId"

    users {
        text id PK
        text username UNIQUE
        text passwordHash "argon2"
        text email
        text role "admin/user/pending"
        bool canAddGames
        text createdAt
    }
    games {
        text id PK
        text name
        text parentGameId FK "nullable"
```

```
        bool isExtension
        text contentType "base/extension/advanced_rules/faq"
        text rulesLanguage "fr/en"
        text sourceFile "/app/pdfs/<slug>-<ts>.pdf"
        int chunksCount
        text ingestStatus "idle/running/done/error/scheduled"
        text ingestScheduledAt
        text addedBy FK
        text createdAt
        int bggId "nullable"
        text imageUrl
        text bggType
        text bggMechanics "JSON array"
        text bggCategories "JSON array"
        text hasCardDatabase "magic-cards/lorcana-cards/etc."
    }
    questions {
        text id PK
        text userId FK
        text gameId FK
        text question
        text answer "nullable, injecté post-génération"
        real bestScore
        text createdAt
        text vote "up/down/null"
        text feedbackComment
        text diagnostics "JSON blob"
    }
```

Migrations livrées

Fichier	Apport
`0000_new_forge.sql`	Initial : users, games, questions
`0001_vellum_bgg.sql`	Colonnes BGG (bggId, imageUrl, mechanics, categories)
`0002_noisy_the_initiative.sql`	`ingestScheduledAt`
`0003_pale_rhodey.sql`	`contentType`, `isExtension`, `parentGameId`, `rulesLanguage`
`0004_sharp_the_captain.sql`	`hasCardDatabase`
`0005_pretty_lady_bullseye.sql`	`bggType`

Fichier	Apport
`0006_goofy_terror.sql`	`ingestStatus` enum complet
`0007_clever_spectrum.sql`	Schéma conflicts (extension detection)
`0008_panoramic_rocket_raccoon.sql`	Contextual cache support
`0009_daffy_tana_nile.sql`	Feedback (vote, feedbackComment)
`0010_daily_sentinels.sql`	Diagnostics RAG (JSON blob)

Workflow migration :

```
# 1. Modifier src/schema.ts
# 2. Générer la migration
npm run db:generate
# 3. Appliquer
npm run db:migrate
```

Qdrant (Vector DB)

Collections

- **`rules_<slug>`** : une par jeu (chunks de règles + forums BGG)
- **`magic-cards`** : cartes MTG normalisées
- **`lorcana-cards`** : cartes Lorcana
- **`flesh-and-blood-cards`** : cartes FAB
- **`riftbound-cards`** : cartes Riftbound
- **`terraforming-mars-cards`** : cartes TM
- **`ark-nova-cards`** : cartes Ark Nova

Vecteurs

- **Dense** : 1024 dims (TEI bge-m3)
- **Sparse** : BM25 natif Qdrant

Payload des chunks règles (`rules_<slug>`)

```
{
  // Identité
```

```
  chunk_id: string,
  source_file: string,         // chemin PDF
  source_kind: 'pdf' | 'forum',

  // Hiérarchie
  hierarchy_path: string[],     // ["Chap 3", "Section 2"]
  hierarchy_level: number,
  section_title: string,
  page_start: number,
  page_end: number,

  // Sémantique
  is_extension: boolean,
  is_advanced_rules: boolean,
  is_forum_chunk: boolean,
  game_name: string,
  game_id: string,

  // Conflit (extensions seulement)
  conflict_type: 'replaces' | 'modifies' | 'extends' | null,
  conflict_base_chunk_id: string | null,
  conflict_base_page: number | null,
  conflict_summary: string | null,

  // Texte
  text: string,                 // chunk brut
  contextual_text: string,      // 1-2 phrases LLM préfixées
}
```

Payload des chunks cartes (par TCG, variantes)

Champs communs : `id`, `name`, `name_en`, `set_label`, `rarity`, `type`, `image_url`, `text` (effet/ability).

Champs MTG : `card_mtg_color_identity`, `card_mtg_legal_formats`, `card_mtg_layout`, `mana_cost`, `cmc`, faces (double-face).

Champs FAB : `card_legal_heroes`, `pitch`, `talents`, `class`, `intelligence`, `defense`.

Champs Riftbound : `card_domains`, `energy`, `card_type` (Unit/Champion Unit/Spell/Gear/Battlefield/Legend/Rune), `might`.

Champs Lorcana : `ink`, `lore`, `willpower`, `strength`, `card_type`.

Champs TM : `cost`, `victory_points`, `tags[]`, `requirements`.

Champs Ark Nova : `category` (animal/sponsor), `latin_name`, `size`, `conservation_point`.

Payload des chunks méta (`[META]`)

```
{
  meta_type: 'tier' | 'tournament_deck' | 'forum_thread',
  meta_format: string,          // ex: 'standard', 'classic-constructed', 'spiritforged'
  meta_set: string,             // ex: 'BLB' (17Lands)
  meta_source: string,          // '17lands' / 'mtggoldfish' / 'mobalytics' / 'fabtcg'
  meta_archetype: string,
  meta_placement: number,       // 1-N pour les top 8s
  meta_date: string,            // ISO
  ...
}
```

Filesystem

```
/app/data/                                  # Volume persistant
├── database.db + .db-shm + .db-wal         # SQLite (WAL mode)
├── card-images-cache/                      # Cache sharp-resize
├── magic-cards/                            # Bulk Scryfall + cards.json FR
├── lorcana-cards/                          # LorcanaJSON FR + EN
├── terraforming-mars-cards/
├── ark-nova-cards/
├── ocr-v1-<slug>.json                      # Cache OCR par jeu
├── contexts-v2-<slug>.json                 # Cache Contextual Retrieval B
├── conflicts-v1-<slug>.json                # Cache détection conflits
└── logs/server.log                         # Logger (rotation 50Mo / 30j)

/app/pdfs/                                  # Volume persistant
├── <slug>-<timestamp>.pdf                  # PDF uploadé
└── images/<slug>/page-XX.png               # PNG rendus 300 DPI
```

```
/app/ssh/                                   # Volume RO
└── id_ed25519                              # Clé SSH oracle (dédiée)
```