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.

Revision #1
Created 2026-05-10 15:19:56 UTC by thymon
Updated 2026-05-10 15:19:56 UTC by thymon