TCG integrations

Ajouter un nouveau TCG

Ajouter un nouveau TCG

> *Dernière mise à jour : 2026-05-10*

 Page critique — procédure complète pour intégrer un TCG inédit (ex. Pokemon, Yu-Gi-Oh!, Star Wars Unlimited, Sorcery Contested Realm).

Référence : skill global `add-tcg` listé dans `~/.claude/skills/add-tcg/`. Ce skill fournit une checklist 6-axes ; cette page détaille les fichiers + snippets à toucher.

Vue d'ensemble : 6 axes

Axe	But	Fichiers principaux
1	Cartes (data + ingestion)	`services/cards/sources/<tcg>.ts`, `scripts/<tcg>-cards/`
2	Symboles UI inline	`frontend/src/lib/<tcg>-symbols.ts`, `frontend/public/<tcg>-icons/`
3	Decompose-query (synergy/deckbuilding)	`services/rag/retrieve/multi-query-<tcg>.ts`
4	Deckbuilding (pool + spec)	`services/rag/deckbuilding/{spec,pool,validate}.ts`
5	Set matching	`services/cards/set-aliases.ts`
6	Méta (tier list + tournois)	`services/meta/<tcg>.ts`, `cron/meta-sync.ts`

Tous les axes ne sont pas obligatoires. Pour un TCG sans format constructed (ex. jeu de société), les axes 4 et 6 ne s'appliquent pas.

Axe 1 — Cartes (data + ingestion)

1.1 Choisir la source

- **API CDN éditeur officiel** quand possible (FAB, MTG, Riftbound : tous trouvés sur le CDN officiel — vérifier `rules.<tcg>.com` ou équivalent)
- **JSON tiers MIT** sinon (Lorcana via LorcanaJSON, etc.)
- **Bulk download** (Scryfall MTG) ou **API live** (Riot Riftbound)
- **Package npm bundlé** (FAB) si la source est très stable et l'éditeur publie un package

1.2 Créer la source

`src/services/cards/sources/<tcg>.ts` doit implémenter l'interface `CardSource` :

```
import { CardSource } from './types.js';

export const myTcgSource: CardSource = {
  collection: 'mytcg-cards',  // doit matcher le `hasCardDatabase` qu'on assignera aux jeux

  async load() {
    // Lit la source (file, API, package) → renvoie { cards: NormalizedCard[], hash: string }
    // Le hash sert au cards-sync pour détecter les changements
  },

  normalizeCard(rawCard) {
    // Transforme le format source → schema interne (Qdrant payload)
    return {
      id: ...,
      name: ...,
      // … champs communs + spécifiques au TCG
    };
  },

  getImageUrl(card) {
    // CDN éditeur si dispo, sinon chemin local
    return card.image_url ?? `/cards/<tcg>/${card.id}.png`;
  },
};
```

1.3 Enregistrer dans le registry

`src/services/cards/sources/registry.ts` :

```
import { myTcgSource } from './<tcg>.js';

export const cardSources = new Map([
  ['magic-cards', magicSource],
  ['lorcana-cards', lorcanaSource],
  // ...
  ['mytcg-cards', myTcgSource],   // ← ajouter ici
]);
```

⚠ Sans cet enregistrement, `cards-cache.ts` ne load pas la collection.

1.4 Scripts d'ingestion

Modèle à dupliquer : `scripts/riftbound-cards/`. Crée :

```
scripts/<tcg>-cards/
├── fetch-cards.ts       # Télécharge / API call → cache JSON
├── ingest.ts            # Push Qdrant via cards-sync
├── link-game.ts         # CLI pour lier un jeu à hasCardDatabase = '<tcg>-cards'
└── (test-retrieve.ts)   # Optionnel : debug retrieval
```

1.5 Commandes npm

`package.json` :

```
{
  "scripts": {
    "cards:<tcg>:fetch": "tsx --env-file=.env scripts/<tcg>-cards/fetch-cards.ts",
    "cards:<tcg>:ingest": "tsx --env-file=.env scripts/<tcg>-cards/ingest.ts",
    "cards:<tcg>:link-game": "tsx --env-file=.env scripts/<tcg>-cards/link-game.ts"
  }
}
```

1.6 Variables d'env

⚠ **3 fichiers en parallèle** :

1. `src/config.ts` :

   ```
   MY_TCG_CARDS_DATA_DIR: z.string().default('/app/data/<tcg>-cards'),
   ```

2. `.env.example` : `MY_TCG_CARDS_DATA_DIR=/app/data/<tcg>-cards`
3. `unraid/boardgame-referee.xml` : ajouter le `<Config Type="Variable">...` correspondant

Sinon l'admin Unraid ne pourra pas la setter via l'UI.

1.7 Vérifier

```
docker exec boardgame-referee npm run cards:<tcg>:fetch
docker exec boardgame-referee npm run cards:<tcg>:ingest
# /admin/services doit montrer la collection avec count > 0
```

Axe 2 — Symboles UI inline

2.1 Récupérer les PNG officiels

Toujours vérifier le CDN officiel d'abord — FAB/MTG/Riftbound ont tous les PNG officiels sur le site de règles ou press-kit (cf. mémoire `feedback_tcg_official_icons_cdn.md`).

Stocker dans `frontend/public/<tcg>-icons/`.

2.2 Helper de remplacement

Modèle : `frontend/src/lib/riftbound-symbols.ts`. Crée `<tcg>-symbols.ts` :

```
const SYMBOL_RE = /\{([rpdhicut])\}/g;  // whitelist STRICTE — sinon collision avec d'autres
TCG

export function replaceMyTcgTokens(html: string): string {
  return html.replace(SYMBOL_RE, (_, key) => {
    return `<img src="/<tcg>-icons/${key}.png" class="card-symbol" alt="${key}">`;
  });
}
```

```
export const myTcgEnabled = true;
```

⚠ **Regex whitelist obligatoire** : `\{R\}` MTG (majuscule) collisione avec `\{r\}` FAB (minuscule). La regex doit cibler **exactement** les tokens du TCG en cours, jamais plus large.

2.3 Hook dans le markdown

`frontend/src/composables/useArbiterMarkdown.ts` :

```
import { replaceMyTcgTokens, myTcgEnabled } from '../lib/<tcg>-symbols.js';

// Dans la fonction principale, AVANT `renderHtml` :
if (game.value?.hasCardDatabase === '<tcg>-cards' && myTcgEnabled) {
  html = replaceMyTcgTokens(html);
}
```

2.4 Vérifier

Poser une question dont la réponse contient un symbole. Vérifier le rendu image dans la bulle Oracle (DevTools → inspecter le DOM, vérifier que `<img>` apparaît).

Axe 3 — Decompose-query (intent synergy / deckbuilding)

3.1 Multi-query TCG-specific

Modèle : `src/services/rag/retrieve/multi-query-mtg.ts`. Crée `multi-query-<tcg>.ts` :

```
export async function decomposeMyTcgQuery(question: string): Promise<MyTcgSpec | null> {
  const prompt = `... (prompt Haiku qui sort un JSON structuré pour ce TCG)`;
  // Appel Haiku via promptStream avec timeout DECOMPOSE_TIMEOUT_MS
  // Parse JSON, valide via Zod
  // Retourne null si parse fail (le retrieval continuera sans filters spécifiques)
}
```

Le `MyTcgSpec` est un type Zod avec les champs structuraux du TCG (couleurs, format, hero, types, etc.).

3.2 Dispatch dans `synergy-expansion.ts`

```
switch (game.hasCardDatabase) {
  case 'magic-cards':
    spec = await decomposeMtgQuery(question);
    break;
  case 'mytcg-cards':
    spec = await decomposeMyTcgQuery(question);  // ← ajouter
    break;
  // …
}
```

3.3 Vérifier

Poser une question synergy spécifique au TCG ("liste-moi les de couleur <X>"). Vérifier dans `/admin/feedback/<id>` que les filters Qdrant matchent la décomposition.

Axe 4 — Deckbuilding (pool + spec)

4.1 Spec defaults

`src/services/rag/deckbuilding/spec.ts` : ajouter une entrée dans le `DEFAULT_SPECS_BY_TCG` :

```
'mytcg-cards': {
  format: 'standard',
  mainboard: 60,
  sideboard: 15,
  maxCopies: 3,
  // selon le TCG : runes, battlefields, equipment, etc.
}
```

4.2 Pool éligible

`src/services/rag/deckbuilding/pool.ts` : étendre `fetchEligibleCards` avec le filtre Qdrant approprié :

```
case 'mytcg-cards':
  filter = {
    must: [{ key: 'card_<tcg>_legal_formats', match: { value: spec.format } }],
  };
  break;
```

Si le TCG n'a pas de `legal_formats` (collection mono-format), pas de filter Qdrant — fais un post-filtrage TS sur les champs structurels (couleurs, héro, etc.).

4.3 Validation

`src/services/rag/deckbuilding/validate.ts` : ajouter les contraintes spécifiques :

- Total exact par section
- Max copies (avec whitelist basic lands équivalente si applicable)
- Anchor cards présentes
- Champ singletons obligatoires (un héro FAB, un commandant Commander, etc.)

4.4 Vérifier

Poser "fais-moi une decklist autour de <archétype>". Vérifier que le total + max copies sont respectés et que les anchor cards sont incluses.

Axe 5 — Set matching

`src/services/cards/set-aliases.ts` : ajouter les alias FR/EN du nouveau TCG :

```
export const SET_ALIASES = new Map([
  // MTG
  ['Strixhaven', 'STX'],
  // … existants
```

```
  // <tcg>
  ['Surge', 'SUR'],
  ['Nouveau set FR', 'NSF'],
]);
```

Permet à l'utilisateur de citer un set par son nom complet, le filtre Qdrant s'applique sur le code 3 lettres.

Vérifier

Poser une question qui mentionne un set par son nom complet. Vérifier le filter dans `/admin/feedback/<id>`.

Axe 6 — Méta (tier list + tournois)

6.1 Source méta stable

Identifier une source qui :

- Survit dans le temps (pas figée comme DotGG Lorcana)
- Expose une API ou un HTML scrapable de manière fiable
- Couvre un format pertinent

6.2 Service méta

Modèle : `services/meta/mobalytics-riftbound.ts` (scrape) ou `services/meta/mtgtop8.ts` (scrape avec rate-limit). Crée `services/meta/<tcg>.ts` :

```
export async function syncMyTcgMeta(): Promise<MetaSnapshot> {
  // 1. Fetch / scrape la source
  // 2. Transformer en MetaSnapshot { tier_list, tournament_decks }
  // 3. Retourner pour ingest via services/meta/ingest.ts
}
```

6.3 Cron meta-sync

`src/cron/meta-sync.ts` : ajouter l'appel au sync à la fréquence configurée :

```
if (config.META_MY_TCG_ENABLED) {
  await ingestSnapshot(await syncMyTcgMeta());
}
```

6.4 Variables d'env méta

Encore une fois, **3 fichiers en parallèle** :

```
META_MY_TCG_ENABLED=false
META_MY_TCG_RATE_LIMIT_MS=1500
META_MY_TCG_USER_AGENT=Mozilla/5.0 (compatible; ...)
# … selon le TCG
```

6.5 Vérifier

```
docker exec boardgame-referee npm run meta:<tcg>:sync
```

Vérifier que des chunks `[META]` apparaissent dans Qdrant et sortent dans une question méta.

Verification end-to-end

1. **Créer un jeu** via `/add-game` (ou `npm run cards:<tcg>:link-game`) avec `has_card_database = '<tcg>-cards'`
2. **Tester autocomplete** `@<carte>` → cartes ressortent
3. **Tester citation** `[[card:<Nom>]]` dans une réponse Oracle → bouton zoom OK
4. **Tester intent synergy** → diagnostics montrent les filters TCG
5. **Tester intent deckbuilding** → decklist respecte spec (total + max copies + anchors)
6. **Tester import deck** (si applicable) → mapping deck → stickyCardMentions

---

Pièges classiques

Piège	Conséquence	Fix

Oublier d'enregistrer dans `registry.ts`	Cards-cache ne load pas la collection	Ajouter dans `cardSources`
Oublier `additionalDirectories` dans le settings.json oracle	Vision Read échoue	Ajouter dans le settings.json + `permissions.allow`
`process.env.X` dans une route au lieu de `config.X`	Silencieusement vide en prod	Centraliser dans `src/config.ts`
Regex symboles trop large	Collision avec un autre TCG	Whitelist stricte caractère par caractère
Oublier `unraid/boardgame-referee.xml`	Admin Unraid ne peut pas setter la var	Mettre à jour les 3 fichiers (config.ts + .env.example + xml)
Forget `link-game` après ingest	Le jeu n'a pas `hasCardDatabase`	Lancer `npm run cards:<tcg>:link-game <gameId>`
Bundlée image (FAB-style) sans rebuild	Resync ne voit pas la nouvelle data	Build + redeploy AVANT resync

Aperçu des TCG supportés

Aperçu des TCG supportés

> *Dernière mise à jour : 2026-05-10*

6 TCG intégrés. Chaque TCG a une page dédiée dans ce chapitre. Workflow de mise à jour des cartes : voir `mettre-a-jour-cartes.md`. Workflow d'ajout d'un nouveau TCG : voir `ajouter-un-tcg.md`.

Tableau récap

TCG	Collection Qdrant	Source data	Type source	Fréquence maj	Symboles UI	Deckbuilding	Méta
Magic: The Gathering	`magic-cards`	Scryfall bulk JSON + traduction Haiku	Téléchargée	À chaque set Standard (~3-4×/an)	mana-font (webfont)	✅	✅ 17Lands + MTGGoldfish + MTGTop8
Disney Lorcana	`lorcana-cards`	LorcanaJSON.org	Téléchargée (MIT)	À chaque set	PNG Ravensburger	⚠ partiel	✅ DotGG figée nov 2025
Flesh and Blood	`flesh-and-blood-cards`	Package npm `@flesh-and-blood/cards`	Bundlée image Docker	À chaque set	PNG LSS	✅	✅ fabtcg.com tournois
Riftbound	`riftbound-cards`	API Riot card-gallery	Live API	À chaque set	PNG Riot press-kit	✅	✅ Mobalytics + RiftboundStats
Terraforming Mars	`terraforming-mars-cards`	HTML parsing + cards.json	Statique locale	Quasi-statique	(cards.json)	✅ pas de format	✅
Ark Nova	`ark-nova-cards`	JSON + sprites	Statique locale	Quasi-statique	(sprites locales)	✅	✅

Caractéristiques transverses

Côté backend

- **Source** : implémente l'interface `CardSource` dans `src/services/cards/sources/<tcg>.ts` (`load()`, `normalizeCard()`, `getImageUrl()`)
- **Registry** : enregistré dans `src/services/cards/sources/registry.ts` (Map `collection → CardSource`)
- **Cache mémoire** : `cards-cache.ts` charge la collection au boot (warm-up `CARD_WARM_TIMEOUT_MS`)
- **Recherche** : `/api/cards/search` utilise BM25 ou full-text in-memory

Côté frontend

- **Symboles inline** : `frontend/src/lib/<tcg>-symbols.ts` avec regex whitelist + remplacement
- **Hook autocomplete** : `useMentionAutocomplete` debounce 150ms → `/api/cards/search`
- **Modale zoom** : `CardZoomModal.vue` (770+ lignes, async load) gère stats + ability text par TCG

Côté RAG

- **`hasCardDatabase`** sur `games` : pointe vers la collection Qdrant correspondante. Active dynamiquement :
  - Autocomplete `@card`
  - Bloc `CARTES CITÉES` dans le userPrompt
  - Mode deckbuilding (selon les champs structurels disponibles)
  - Resync via `/admin`

Côté méta

- Chunks `[META]` ingérés via `services/meta/<source>.ts`
- Ingest sources : `services/meta/ingest.ts` (snapshot → chunks)
- Cron `meta-sync.ts` orchestre la fréquence (par défaut hebdo, configurable par TCG)

Limites actuelles

- **Lorcana méta** : DotGG est figé depuis le 21 nov 2025. Pas de source alternative pour l'instant. Si DotGG reprend, juste relancer `META_LORCANA_*` (pas implémenté car source morte).
- **TM / Ark Nova deckbuilding** : pas de champs structurels (format, hero, color identity). Retombe silencieusement sur retrieval synergy seul.
- **Pokemon TCG** : pas implémenté. Procédure dans `ajouter-un-tcg.md`.
- **Pas de cross-TCG dans une même question** : un jeu = une `hasCardDatabase` unique. Le mode "MTG vs Lorcana sur le même chat" n'a pas de sens applicatif.

Ark Nova

Ark Nova

> *Dernière mise à jour : 2026-05-10*

Jeu de société (zoo), pas un TCG. Traité avec base de cartes pour l'autocomplete `@` et l'enrichissement RAG.

Source

- **JSON local + sprite sheets** : extraction one-shot
- **Collection Qdrant** : `ark-nova-cards`
- **Data dir** : `ARK_NOVA_CARDS_DATA_DIR` (défaut `/app/data/ark-nova-cards`)
  - `cards.json` : extrait
  - `images/animals/`, `images/sponsors/` : PNG découpées via sprite-slicing

Code

Fichier	Rôle
`src/services/cards/sources/ark-nova.ts`	CardSource
`scripts/ark-nova-cards/extract-cards.ts`	Parse JSON local
`scripts/ark-nova-cards/slice-sprites.ts`	Découpe sprite sheets en PNG individuelles
`scripts/ark-nova-cards/ingest.ts`	Push Qdrant
`scripts/ark-nova-cards/link-game.ts`	Lie ligne `games`

Payload Qdrant `ark-nova-cards`

```
{
  id: pointId,
  name: string,
  category: 'animal' | 'sponsor',
  latin_name: string,         // pour les animaux
  size: 'small' | 'medium' | 'large',
  conservation_point: number,  // points conservation
  endangered: boolean,
  image_url: string,          // chemin local PNG découpé
  text: string,               // texte des effets
}
```

Pas de méta ni deckbuilding

Ark Nova n'a pas de format compétitif constructed. Le mode deckbuilding ne s'applique pas — retrieval synergy seul.

Pas de symboles UI dédiés

Les pictos (taille, conservation, abreuvoir, partenaire scientifique…) restent dans le texte descriptif des cartes. Si tu veux ajouter des PNG : créer `frontend/src/lib/ark-nova-symbols.ts` + assets `frontend/public/ark-nova-icons/`.

Mise à jour

Quasi-statique. Si une nouvelle extension sort :

1. Mettre à jour le JSON local et/ou les sprites
2. `docker exec boardgame-referee npm run cards:ark-nova:slice` (re-découpe sprites)
3. `docker exec boardgame-referee npm run cards:ark-nova:extract` (re-parse JSON)
4. `docker exec boardgame-referee npm run cards:ark-nova:ingest` (push Qdrant)
5. `/admin` → Resync ARK NOVA cards (optionnel)

Cas d'usage typique

- "Quels animaux donnent le plus de points conservation ?" → retrieval cartes + tri par `conservation_point`
- "Quelle synergie avec les small animals ?" → retrieval avec filter `size='small'`
- Citation `@Tiger` dans une question → carte injectée en bloc CARTES CITÉES

Disney Lorcana

Disney Lorcana

> *Dernière mise à jour : 2026-05-10*

Source

- **LorcanaJSON.org** : https://github.com/LorcanaJSON/LorcanaJSON (MIT, JSON bulk FR + EN)
- **Collection Qdrant** : `lorcana-cards`
- **Data dir** : `LORCANA_CARDS_DATA_DIR` (défaut `/app/data/lorcana-cards`)
  - `en/allCards.json`, `fr/allCards.json`

Code

Fichier	Rôle
`src/services/cards/sources/lorcana.ts`	CardSource (load, normalize, getImageUrl)
`src/services/cards/sources/lorcana-normalize.ts`	Symboles ink/lore/willpower normalisés
`scripts/lorcana-cards/download.ts`	Télécharge allCards.json FR + EN
`scripts/lorcana-cards/ingest.ts`	Push Qdrant
`scripts/lorcana-cards/ingest-symbols.ts`	Symboles spécialisés (ink, lore, willpower)

Payload Qdrant `lorcana-cards`

```
{
  id: pointId,
  name: string,            // FR
  name_en: string,         // EN
  version: string,         // "Bandit rusé"
  set_label: string,
  rarity: string,
  card_type: 'Character' | 'Item' | 'Action' | 'Location' | 'Song',
  ink: number,             // coût en ink (mana)
  willpower: number,
  strength: number,
  lore: number,            // points de lore générés
  text: string,            // abilities
  image_url: string,       // CDN Ravensburger officiel (api.lorcana.ravensburger.com)
}
```

Symboles UI

Token texte	Symbole	PNG
`{E}`	Exert	`/lorcana-icons/exert.png`
`{I}`	Ink	`/lorcana-icons/ink.png`
`{L}`	Lore	`/lorcana-icons/lore.png`
`{S}`	Strength	`/lorcana-icons/strength.png`
`{W}`	Willpower	`/lorcana-icons/willpower.png`

PNG officiels Ravensburger (vérifier les ToS si on les redistribue publiquement, pour usage perso self-host c'est OK).

Helper : `frontend/src/lib/lorcana-symbols.ts`. Activé uniquement si `game.hasCardDatabase === 'lorcana-cards'`.

⚠ Méta : DotGG figée nov 2025

L'API DotGG (api.dotgg.gg/cgfw/) servait pour Lorcana méta-game (tier list, archétypes), mais sa data est figée au **21 novembre 2025**. Le projet a tenté l'intégration 2× (cf. mémoire `project_lorcana_integration.md`), refermée chaque fois.

Action : ne pas relancer une sync méta Lorcana tant que DotGG ne reprend pas. Si une autre source apparaît (Mobalytics? officiel Disney?), créer un nouveau service `services/meta/<source>.ts`.

Deckbuilding (partiel)

Le deckbuilding fonctionne sur les axes :

- Total carte (60 max copies 4)
- Couleurs (ink colors : Amber, Amethyst, Emerald, Ruby, Sapphire, Steel)
- Types (Character/Item/Action/Location/Song)

Mais sans méta dispo, les decklists d'inspiration ne sont pas alimentées. Tu auras une liste cohérente mais pas forcément alignée sur le métagame compétitif.

Pièges connus

- **Apostrophes typographiques dans les noms** ("Mickey Mouse - Bandit rusé") : utiliser `normalizeCardKey()` (NFC + lowercase + apostrophe normalization). `\b` regex sans flag `u` rate les noms finissant par `é`.
- **Versions multiples** : "Mickey Mouse" peut avoir 5+ versions différentes (variantes par set + version artistique). Le `name` est unique seulement avec `version` en suffixe (`Mickey Mouse - Bandit rusé`).

Source images

CDN officiel Ravensburger : https://api.lorcana.ravensburger.com/v1/... Pas de cache local pour images cartes (proxy direct via `/api/cards/image/:pointId`).

Flesh and Blood (FAB)

Flesh and Blood (FAB)

> *Dernière mise à jour : 2026-05-10*

Source

- **Package npm** : `@flesh-and-blood/cards` + `@flesh-and-blood/types` (LSS, ~3.6.246)
- **Bundle Docker** : la data est embarquée dans l'image Docker au build (pas téléchargée à runtime)
- **Collection Qdrant** : `flesh-and-blood-cards`

Code

Fichier	Rôle
`src/services/cards/sources/flesh-and-blood.ts`	CardSource utilisant le package npm
`scripts/flesh-and-blood-cards/ingest.ts`	Push Qdrant
`scripts/flesh-and-blood-cards/link-game.ts`	Lie une ligne `games` à `hasCardDatabase = 'flesh-and-blood-cards'`
`src/services/decks/parse-decklist.ts`	Parser tolérant Fabrary / FABDB / GEM
`src/services/decks/match-decklist.ts`	Index `byFullName` + `byBaseName` du cache cartes
`src/routes/decks.ts`	`POST /api/decks/parse`

Payload Qdrant `flesh-and-blood-cards`

```
{
  id: pointId,
  name: string,
  name_en: string,
  set_label: string,
  rarity: string,
  card_type: 'Action' | 'Attack' | 'Defense Reaction' | 'Equipment' | 'Hero' | ...,
  pitch: 1 | 2 | 3 | null,    // red/yellow/blue
  attack: number,
  defense: number,
  cost: number,
  card_legal_heroes: string[],  // ['Briar', 'Lexi', 'Dash', ...]
  classes: string[],            // ['Ranger', 'Wizard']
  talents: string[],            // ['Earth', 'Lightning']
  text: string,
  image_url: string,
}
```

Symboles UI

Token	Symbole	PNG
`{r}`	Red pitch	`/fab-icons/icon_r.png`
`{p}`	Power	`/fab-icons/icon_p.png`
`{d}`	Defense	`/fab-icons/icon_d.png`
`{h}`	Health	`/fab-icons/icon_h.png`
`{i}`	Intelligence	`/fab-icons/icon_i.png`
`{c}`	Cost	`/fab-icons/icon_c.png`
`{u}`	Currency	`/fab-icons/icon_u.png`
`{t}`	Tap	`/fab-icons/icon_t.png`

PNG officiels LSS depuis `rules.fabtcg.com` (cf. mémoire `project_fab_symbols_done.md`, commit `2a77d94`).

Helper : `frontend/src/lib/fab-symbols.ts`. Whitelist regex stricte `[rpdhicut]` minuscules — sinon collision avec `{R}` MTG (majuscules).

Import de deck (Fabrary)

Workflow : Fabrary → "Copy as Text" → coller dans `DeckImportModal` → preview → attacher au chat.

Choix de design : on ne fetch PAS Fabrary. L'app Fabrary est une SPA React avec API GraphQL AWS AppSync protégée par Cognito Identity Pool + signature SigV4. Implémenter ce flow côté serveur serait fragile (endpoint non documenté) et lourd. Le format texte standard FAB est stable, fonctionne avec tous les builders communauté (Fabrary, FABDB, GEM), et se passe d'auth.

Parser tolérant (`parse-decklist.ts`)

Accepte :

- `3 Card`, `(3) Card`, `3x Card`, `3 - Card`
- Suffixe pitch optionnel : `(red|yellow|blue)`
- En-têtes : `Hero:`, `Weapon:`, `Equipment:`, `Deck:`/`Mainboard:`/`Pitch 1/2/3`, `Sideboard:`, `Name:`, `Format:`
- Lignes vides et commentaires `//` ou `#` ignorés

⚠ **Le parser exige une quantité numérique en tête** : sans ça, les lignes parasites du footer Fabrary (`Voir le deck complet @ https://...`, `Fait avec ♥`) seraient comptées comme cartes à qty=1 et pollueraient `unmatched`.

Décodage automatique `decodeURIComponent` quand le presse-papier mobile renvoie du `%20`/`%0A`.

Matcher (`match-decklist.ts`)

- Index `byFullName` : match exact `<name> (<pitch>)`
- Index `byBaseName` : match sur le nom de base si pas de pitch
- Quand l'utilisateur ne précise pas le pitch et qu'il existe plusieurs variantes : priorité **red > yellow > blue** + flag `pitchAssumed=true` surfacé dans la preview

Whitelist côté backend

`POST /api/decks/parse` vérifie :

- `requireAuth` + `requireConfirmed`
- Rate-limit 10/min/user
- `game.contentType === 'base'`
- `game.hasCardDatabase ∈ SUPPORTED_COLLECTIONS = ['flesh-and-blood-cards']`

Pour étendre à un autre TCG : ajouter à `SUPPORTED_COLLECTIONS` + créer parser+matcher dédié pour le format de ce TCG.

Méta-game

`META_FAB_TOURNAMENT_ENABLED=true` active la sync `services/meta/fabtcg.ts` :

- Scrape `fabtcg.com` (officiel LSS)
- Formats configurables : `META_FAB_TOURNAMENT_FORMATS=classic-constructed,blitz,living-legend,silver-age`
- Cap : `META_FAB_TOURNAMENT_MAX_DECKS=50` decks par format
- Âge max : `META_FAB_TOURNAMENT_MAX_AGE_DAYS=60`
- Rate-limit : `META_FAB_RATE_LIMIT_MS=1500` (Cloudflare-friendly)
- User-Agent : `META_FAB_USER_AGENT` (header Cloudflare-compatible)

Workflow update

⚠ Cas spécial : la data FAB est **bundlée dans l'image Docker** via le package npm. Mettre à jour les cartes nécessite un rebuild + redeploy AVANT le resync. Cf. `mettre-a-jour-cartes.md`.

Magic: The Gathering

Magic: The Gathering

> *Dernière mise à jour : 2026-05-10*

Source

- **Bulk Scryfall** : `https://api.scryfall.com/bulk-data` → `all_cards.json` téléchargé localement
- **Traduction** : Haiku traduit les cartes manquantes (FR souvent en retard sur l'anglais sur Scryfall)
- **Collection Qdrant** : `magic-cards`
- **Data dir** : `MAGIC_CARDS_DATA_DIR` (défaut `/app/data/magic-cards`)
  - `all-cards.json` : bulk Scryfall (téléchargé)
  - `cards.json` : extrait + dédupliqué par `oracle_id`, version FR

Code

Fichier	Rôle
`src/services/cards/sources/magic.ts`	Implémente `CardSource` (load, normalize, getImageUrl)
`scripts/magic-cards/download-bulk.ts`	Télécharge `all-cards.json` Scryfall
`scripts/magic-cards/extract-cards.ts`	Filtre + dédupe par oracle_id, version FR
`scripts/magic-cards/translate-missing.ts`	Bat ch Haiku traduit cartes manquantes (~3000/batch)
`scripts/magic-cards/ingest.ts`	Pousse vers Qdrant `magic-cards`
`scripts/meta-mtg/sync.ts`	Sync 17Lands API
`scripts/meta-mtg/sync-constructed.ts`	Scrape MTGGoldfish
`scripts/meta-mtg/sync-tournament.ts`	Scrape MTGTop8

Fichier	Rôle
`scripts/meta-mtg/fix-legalities.ts`	Re-vérifie les formats Standard après rotation

Payload Qdrant `magic-cards`

```
{
  id: pointId,
  name: string,            // FR
  name_en: string,         // EN
  set_label: string,       // ex: "Strixhaven (STX)"
  rarity: 'common' | 'uncommon' | 'rare' | 'mythic',
  card_type: string,       // "Creature — Human Wizard"
  mana_cost: string,       // "{2}{U}{U}"
  cmc: number,             // 4
  card_mtg_color_identity: string[],  // ['U', 'B']
  card_mtg_legal_formats: string[],   // ['standard', 'modern', 'legacy', ...]
  card_mtg_layout: string, // 'normal' | 'modal_dfc' | 'transform' | etc.
  faces: [...],            // pour double-face
  power: string,
  toughness: string,
  text: string,            // ability text (effet)
  image_url: string,       // CDN Scryfall
}
```

Symboles UI

- Webfont `mana-font` (Andrew Gioia) chargée depuis npm package
- Helper `frontend/src/lib/mana.ts` : regex `/{[WUBRGCSTXP0-9/]+}/g` → `<i class="ms ms-w">`
- Active uniquement si `game.hasCardDatabase === 'magic-cards'` (guard dans `useArbiterMarkdown`)

Méta-game

3 sources combinées :

- **17Lands** : draft analytics (winrate, pick order). Activable via `META_MTG_SET=BLB` (code set 17Lands).
- **MTGGoldfish** : constructed metagame. Activable via `META_MTG_FORMATS=standard,modern,pioneer`.
- **MTGTop8** : top 8 tournois. Activable via `META_TOURNAMENT_ENABLED=true` + `META_TOURNAMENT_FORMATS`.

Toutes les sources poussent dans Qdrant comme chunks `[META]`. Le RAG retrieval méta filtre sur `meta_format` + `meta_set` selon la question.

Deckbuilding

- Spec Haiku par défaut Standard : `60 mainboard, 15 sideboard, max 4 par non-basique`
- Filtre Qdrant : `card_mtg_legal_formats` contient `spec.format`
- Post-filtre TS : `card_mtg_color_identity ⊆ spec.colors` (Qdrant ne gère pas bien le subset)
- Basic lands FR/EN whitelistés pour bypass max copies : Plains/Island/Swamp/Mountain/Forest/Wastes + Plaine/Île/Marais/Montagne/Forêt

Set matching

`set-aliases.ts` normalise les noms d'extensions :

- "Strixhaven" → "STX"
- "Wilds of Eldraine" → "WOE"
- etc.

Permet à l'utilisateur de citer un set par son nom complet, le filtre Qdrant s'applique sur le code 3 lettres.

Multi-query MTG

`retrieve/multi-query-mtg.ts` : Haiku décompose la question en JSON :

```
{
  colors: string[],       // ['R', 'G']
  cmcMax: number,         // 3
  types: string[],        // ['Creature']
  themes: string[],       // ['elf', 'tribal']
```

```
  format: string,         // 'standard'
  set: string             // 'BLB'
}
```

Chaque champ devient un filter Qdrant natif (payload matching) → réduit le candidate set avant le rerank.

Workflow update

Voir `mettre-a-jour-cartes.md` pour le détail (workflow 4 étapes : download → extract → translate-missing → ingest, + fix-legalities post-rotation).

Mettre à jour les cartes d'un TCG

Mettre à jour les cartes d'un TCG

> *Dernière mise à jour : 2026-05-11*

 Page critique — workflow de référence quand un nouveau set sort.

TL;DR

Depuis 2026-05-11, la majorité des resyncs se font **directement depuis `/admin` → section "Bases de cartes"**. Plus besoin de SSH dans le container pour les usages courants.

TCG	Source	Mode admin	Workflow
Riftbound	API Riot live	"Resync. (live)" — inline	Un clic. Diff + upsert via `syncCollection`.
MTG	Scryfall bulk JSON	"Lancer le pipeline" — modal	Download → extract → ingest (~25-40 min)
Lorcana	LorcanaJSON.org	"Lancer le pipeline" — modal	Download → ingest (~5-10 min)
FAB	Package npm `@flesh-and-blood/cards` (bundlé)	"Lancer le pipeline" — modal	Ingest seul (~5-10 min). ⚠ `npm update` + redeploy d'abord pour avoir les nouvelles cartes (cf. § FAB)
Terraforming Mars	Local `${TM_CARDS_DATA_DIR}`	"Lancer le pipeline" — modal	Parse-html → ingest (~5 min)

TCG	Source	Mode admin	Workflow
Ark Nova	Local `${ARK_NOVA_CARDS_DATA_DIR}`	"Lancer le pipeline" — modal	Slice-sprites → extract → ingest (~5 min)

Le bouton "Lancer le pipeline" exécute les **commandes npm** correspondantes côté serveur via `child_process.spawn`. Les logs sont streamés en SSE dans le modal de progression. Un seul pipeline tourne à la fois (lock global pour ne pas saturer TEI).

Architecture

Service `src/services/cards/sync-jobs/`

Quatre fichiers + un index :

- **`pipelines.ts`** — **whitelist statique** `CARD_SYNC_PIPELINES: Record<collection, { label, steps: [{ label, npm }] }>`. C'est le **seul** endroit qui mappe une collection à des commandes npm. Sécurité : aucune commande shell n'est jamais construite à partir d'un paramètre client. La collection envoyée par le frontend sert uniquement de clé dans cette map.
- **`queue.ts`** — file d'attente en mémoire avec **lock global**. Un seul `activeJob: CardSyncJob | null`. Expose `isAnyRunning()`, `createJob(coll)`, `pushEvent(type, data)`, `markFinished()`, `getEvents(coll, fromIndex)`. Rétention 10 min après `finished=true` pour permettre la reprise UI.
- **`runner.ts`** — orchestrateur. Pour chaque étape :
  1. Spawn `npm run <step.npm>` avec `cwd=PROJECT_ROOT` (calculé via `fileURLToPath(import.meta.url)`).
  2. Stdout / stderr lus ligne par ligne via `readline` → events `log` typés `{ stepIndex, line, stream }`.
  3. Exit code ≠ 0 → throw → event `error` + upsert `card_collection_meta.last_status = 'error'`.
  4. Succès : event `step:done` avec durationMs.
  5. À la fin : query Qdrant pour `cardsCount`, upsert `card_collection_meta` (status='done', `last_synced_at`, `duration_ms`).
- **`types.ts`** — types partagés (`CardSyncJob`, `CardSyncEvent`, `CardSyncPipeline`, `CardSyncEventType`).
- **`index.ts`** — API publique : `startPipeline(collection)`, `isAnyRunning()`, `getActiveJob()`, `getEvents()`, `resolvePipeline()`, `listPipelineCollections()`.

Table SQLite `card_collection_meta`

Définie dans `src/schema.ts`, exposée par `src/repositories/card-collection-meta.repo.ts` (`get`, `listAll`, `upsert`).

Champ	Type	Rôle
`collection`	text PK	Nom de la collection Qdrant (`magic-cards`, `lorcana-cards`, ...)
`last_synced_at`	text (ISO)	Date du dernier succès. NULL = jamais synchronisé via l'admin
`last_status`	text enum	`idle` / `running` / `done` / `error`
`last_error`	text	Message d'erreur si `last_status='error'`, NULL sinon
`cards_count`	integer	Nombre de points Qdrant après le dernier succès
`duration_ms`	integer	Durée du dernier pipeline en ms

Migration : `migrations/0011_tiny_nemesis.sql`. Appliquée automatiquement au boot via `runMigrations()` (`src/db.ts`).

Une ligne par collection (partagée entre le jeu de base et ses extensions). Alimentée par le `runner.ts` à chaque exécution.

Routes admin

Toutes sous `/api/admin/cards/*`, protégées par `requireAuth` + `requireAdmin`.

Route	Méthode	Rôle
`/admin/cards`	GET	Liste enrichie (dédoublonnée par collection) avec `chunksCount`, `supportsLiveResync`, `hasPipeline`, `pipelineSteps`, `lastSyncedAt`, `lastStatus`, `lastError`, `durationMs`
`/admin/cards/:collection/resync`	POST → SSE	Mode "live" (Riftbound) : `syncCollection()` qui diffe contre Qdrant. 409 pour les sources sans `supportsLiveResync`
`/admin/cards/:collection/pipeline/start`	POST	Lance le pipeline npm en arrière-plan. 202 si OK, 404 si pas de pipeline, 409 si un autre tourne
`/admin/cards/:collection/pipeline/stream`	GET → SSE	Replay historique + events temps réel (`pipeline:start`, `step:start`, `log`, `step:done`, `pipeline:done`, `error`, `heartbeat`)

Route	Méthode	Rôle
`/admin/cards/pipeline/active`	GET	État du lock global. Renvoie `{ collection, startedAt, finished } | null`. Utilisé par l'UI pour rouvrir le modal au refresh

Frontend

- `frontend/src/components/admin/AdminCardDecksSection.vue` — section "Bases de cartes" enrichie (badges fraîcheur, 3 modes de bouton selon `supportsLiveResync` / `hasPipeline`).
- `frontend/src/components/admin/CardSyncPipelineModal.vue` — modal de progression. Ouvre un EventSource sur `/admin/cards/:collection/pipeline/stream`, accumule les events, rend les étapes + console + chrono. Garde les 500 dernières lignes de logs côté client (les scripts MTG produisent ~30 000 lignes).
- `frontend/src/views/AdminView.vue` — orchestration : appelle `pipelineActive()` au mount pour la reprise auto, gère l'ouverture/fermeture du modal, émet `start-pipeline` / `resync` vers la section.

Workflow par TCG (en ligne de commande, si besoin de debug)

Tous les pipelines admin se basent sur ces commandes — elles restent disponibles en CLI si tu veux faire un dry-run, un debug, ou si l'admin est cassé.

FAB ⚠ cas spécial (data bundlée)

La data FAB est dans le package npm `@flesh-and-blood/cards` embarqué dans l'image Docker. **Cliquer "Lancer le pipeline" depuis `/admin` ne ramène PAS de nouvelles cartes** tant que l'image n'est pas rebuilt avec un package à jour. Pour vraiment ajouter de nouvelles cartes :

```
# 1. Mettre à jour le package localement (machine dev)
npm update @flesh-and-blood/cards @flesh-and-blood/types

# 2. Commit + push
git add package.json package-lock.json
git commit -m "chore(deps): update FAB cards to <version>"
git push
```

```
# 3. Attendre que la CI Gitea build/push l'image
# (jobs `test` + `build` dans .gitea/workflows/build.yml)

# 4. Sur Unraid : pull + restart le container
docker compose -f /mnt/user/appdata/boardgame-referee/docker-compose.yml pull app
docker compose -f /mnt/user/appdata/boardgame-referee/docker-compose.yml up -d --force-recreate app

# 5. /admin → "Lancer le pipeline" sur Flesh and Blood
```

MTG

Pipeline admin = `cards:mtg:download` → `cards:mtg:extract` → `cards:mtg:ingest` enchaînés. Si tu veux les lancer manuellement :

```
docker exec boardgame-referee npm run cards:mtg:download
docker exec boardgame-referee npm run cards:mtg:extract
docker exec boardgame-referee npm run cards:mtg:ingest

# Traduction des cartes non encore traduites (Haiku batch, hors pipeline admin)
docker exec boardgame-referee npm run cards:mtg:translate-missing

# Si Standard a tourné (rotation), refixer les légalités
docker exec boardgame-referee npm run meta:mtg:fix-legalities
```

Note : `cards:mtg:translate-missing` peut prendre 2-3h et n'est **pas** dans le pipeline admin (volontairement — c'est une étape lente et batch-bornée). À lancer en CLI quand nécessaire.

Lorcana

Pipeline admin = `cards:lorcana:download` → `cards:lorcana:ingest`. Manuellement :

```
docker exec boardgame-referee npm run cards:lorcana:download
docker exec boardgame-referee npm run cards:lorcana:ingest

# Symboles inline (1 seule fois, ne change pas entre sets)
docker exec boardgame-referee npm run cards:lorcana:ingest-symbols
```

⚠ **Pas relancer la sync méta DotGG tant qu'elle ne reprend pas** (figée 21 nov 2025).

Riftbound

Mode "live" depuis l'admin = un clic. En CLI pour debug :

```
docker exec boardgame-referee npm run cards:riftbound:fetch
docker exec boardgame-referee npm run cards:riftbound:ingest
```

Pas de rebuild image nécessaire (sauf si tu changes le code de normalisation).

Terraforming Mars

Pipeline admin = `cards:tm:parse` → `cards:tm:ingest`. Manuellement :

```
docker exec boardgame-referee npm run cards:tm:parse
docker exec boardgame-referee npm run cards:tm:ingest
```

Ark Nova

Pipeline admin = `cards:ark-nova:slice-sprites` → `cards:ark-nova:extract` → `cards:ark-nova:ingest`. Manuellement :

```
docker exec boardgame-referee npm run cards:ark-nova:slice-sprites
docker exec boardgame-referee npm run cards:ark-nova:extract
docker exec boardgame-referee npm run cards:ark-nova:ingest
```

Diff incrémental sans réembed (`cards:sync`)

Pour les mises à jour mineures (correction d'effet erraté, fix d'une traduction) sans repasser par le pipeline complet :

```
docker exec -e COLLECTION=flesh-and-blood-cards boardgame-referee npm run cards:sync
```

`scripts/cards/sync.ts` appelle `syncCollection()` qui diffe par `card_id + hash(card_text)` et n'embed que ce qui a changé. Idempotent.

C'est aussi ce que fait le mode "Resync. (live)" Riftbound côté admin.

Sécurité — pourquoi la whitelist statique

Le service `sync-jobs` utilise `child_process.spawn` pour exécuter des commandes npm. **Aucune partie de la commande n'est construite à partir d'input client** :

- `scriptName` vient uniquement de `CARD_SYNC_PIPELINES` (whitelist en dur dans le code)
- La `collection` envoyée par le client sert juste de clé dans cette map (un nom non whitelisté → 404)
- Pas de `shell: true`, pas d'argument supplémentaire, pas de `env: { ...userInput }`

Pour ajouter une nouvelle pipeline :

1. Ajouter les scripts npm dans `package.json`
2. Ajouter une entrée dans `CARD_SYNC_PIPELINES` (`src/services/cards/sync-jobs/pipelines.ts`)
3. Build + redeploy (aucun changement frontend nécessaire, la liste est servie dynamiquement par `GET /admin/cards`)

Vérifier que ça a marché

1. **Badge fraîcheur vert** dans `/admin` (≤ 7 jours) avec compteur de cartes mis à jour.
2. **Autocomplete** : sur `/play` du jeu concerné, taper `@<carte récente>` → la carte ressort.
3. **Question synergy** : "*Quelles sont les nouvelles cartes du set X ?*" → l'oracle doit pouvoir les citer.
4. **Si 0 résultat** : vérifier que `games.has_card_database` (en BDD SQLite) pointe sur la collection Qdrant exacte. Reconnect via `npm run cards:<tcg>:link-game` si le lien manque.

Restart cache mémoire après gros update

`services/cards-cache.ts` charge les collections en mémoire au boot. Après un gros update (>1000 cartes), restart le container pour s'assurer que le cache est warm avec la nouvelle data :

```
docker compose -f /mnt/user/appdata/boardgame-referee/docker-compose.yml restart app
```

⚡ **Note** : `syncCollection()` (mode "live" Riftbound) et le runner pipeline n'invalident **pas automatiquement** ce cache — le hot reload des cartes en mémoire est un TODO connu. En attendant, restart après gros update.

Debug

Symptôme	Piste
Bouton désactivé sans raison apparente	Un autre pipeline tourne — `GET /admin/cards/pipeline/active` ou regarder le log container
Modal ne s'ouvre pas au refresh	`pipelineActive()` n'a pas trouvé le job → expiré (>10 min après fin) ou backend redémarré
Logs vides dans la console	EventSource déconnecté (vérifie network tab, status code) ou heartbeat manquant côté serveur
`last_status=error` permanent	Lire `last_error` (truncated dans l'UI, complet en BDD) ; inspecter les logs container pour le traceback complet
Pipeline qui finit en 0s avec exit code 0	Script npm introuvable — vérifier que la commande dans `CARD_SYNC_PIPELINES` existe bien dans `package.json`

Riftbound

Riftbound

> *Dernière mise à jour : 2026-05-10*

TCG Riot Games (univers League of Legends).

Source

- **API Riot** : `https://riftbound.leagueoflegends.com/en-us/card-gallery` (JSON live, pas de bulk download)
- **Collection Qdrant** : `riftbound-cards`
- **Cache local** : `/app/data/riftbound-cards/cache.json` (post-fetch)

Code

Fichier	Rôle
`src/services/cards/sources/riftbound.ts`	CardSource (load depuis cache)
`scripts/riftbound-cards/fetch-cards.ts`	Appelle l'API Riot, écrit cache JSON
`scripts/riftbound-cards/normalize.ts`	Nettoyage HTML → domaines, énergie, types, tags
`scripts/riftbound-cards/ingest.ts`	Push Qdrant
`scripts/riftbound-cards/test-retrieve.ts`	Debug retrieval Qdrant
`scripts/riftbound-cards/link-game.ts`	Lie une ligne games
`services/meta/mobalytics-riftbound.ts`	Tier list Mobalytics (scrape)
`services/meta/riftboundstats.ts`	Tournois RiftboundStats (API)

Payload Qdrant `riftbound-cards`

```
{
  id: pointId,
  name: string,
  name_en: string,
  set_label: string,
  rarity: string,
  card_type: 'Unit' | 'Champion Unit' | 'Spell' | 'Gear' | 'Battlefield' | 'Legend' | 'Rune',
  card_domains: ('Fury' | 'Calm' | 'Mind' | 'Body' | 'Chaos' | 'Order')[],
  energy: number,
  might: number,            // valeur d'attaque
  text: string,             // abilities
  tags: string[],
  image_url: string,         // URL officielle Riot
}
```

Symboles UI

Token	Symbole	PNG
`:rb_might:`	Might (force)	`/riftbound-icons/might.png`
`[rune_fury]`	Domain Fury	`/riftbound-icons/fury.png`
`[rune_calm]`	Domain Calm	`/riftbound-icons/calm.png`
`[rune_mind]`	Domain Mind	`/riftbound-icons/mind.png`
`[rune_body]`	Domain Body	`/riftbound-icons/body.png`
`[rune_chaos]`	Domain Chaos	`/riftbound-icons/chaos.png`
`[rune_order]`	Domain Order	`/riftbound-icons/order.png`

PNG officiels Riot press-kit (cf. mémoire `project_riftbound_support.md`, commit `3de9cdd`).

Helper : `frontend/src/lib/riftbound-symbols.ts`. Activé si `game.hasCardDatabase === 'riftbound-cards'`.

Méta-game

Deux sources :

Mobalytics (tier list)

- `META_RIFTBOUND_ENABLED=true`
- Sync hebdo (fréquence configurable)
- Service `services/meta/mobalytics-riftbound.ts`

RiftboundStats (tournois)

- `META_RIFTBOUND_TOURNAMENT_ENABLED=true`
- Top placement max : `META_RIFTBOUND_TOURNAMENT_MAX_PLACEMENT=4`
- Format ID par défaut : `META_RIFTBOUND_TOURNAMENT_FORMAT_ID=2` (Spiritforged)
- Rate-limit : `META_RIFTBOUND_RATE_LIMIT_MS=300` (1 req/s safe)
- Service `services/meta/riftboundstats.ts`

Deckbuilding

- Spec par défaut : `40 mainboard + 0 sideboard + 12 runes + 3 battlefields + 3 gear, max 3 par carte`
- Pas de filtre Qdrant `legal_formats` (collection mono-format)
- Post-filtre TS : `card_domains ⊆ spec.domains` (multi-domain decks)
- 7 domaines à combiner : Fury / Calm / Mind / Body / Chaos / Order

Multi-query Riftbound

`retrieve/multi-query-riftbound.ts` : Haiku décompose la question en JSON `{domains, energyMax, types, themes, legend}` → filters Qdrant natifs.

Autocomplete `@card`

Pattern @ (mention-style) choisi 2026-04-13 (cf. mémoire `project_riftbound_cards_ux.md`). Première implémentation TCG du pattern, ensuite généralisée à MTG/FAB/Lorcana.

Workflow update

Source live API → workflow simple : `npm run cards:riftbound:fetch` puis `npm run cards:riftbound:ingest`. Pas de rebuild image Docker nécessaire (sauf si tu changes le code de normalisation).

Détails dans `mettre-a-jour-cartes.md`.

Terraforming Mars

Terraforming Mars

> *Dernière mise à jour : 2026-05-10*

Pas un TCG au sens strict (jeu de société avec base de cartes fixe), mais traité comme tel pour l'autocomplete `@`.

Source

- **HTML parsing local** + cards.json (extraction one-shot des règles officielles)
- **Collection Qdrant** : `terraforming-mars-cards`
- **Data dir** : `TM_CARDS_DATA_DIR` (défaut `/app/data/terraforming-mars-cards`)
  - `cards.json` : extrait via parsing HTML
  - `images/` : PNG cartes locales

Code

Fichier	Rôle
`src/services/cards/sources/terraforming-mars.ts`	CardSource
`scripts/terraforming-mars-cards/parse-html.ts`	Parse les règles HTML → cards.json
`scripts/terraforming-mars-cards/ingest.ts`	Push Qdrant
`scripts/terraforming-mars-cards/link-game.ts`	Lie ligne `games`

Payload Qdrant `terraforming-mars-cards`

```
{
  id: pointId,
  name: string,
  cost: number,
  card_type: 'Project' | 'Corporation' | 'Prelude' | 'Standard',
  effect: string,             // texte de l'effet
  victory_points: number,
  tags: string[],             // ['Plant', 'Microbe', 'Building', ...]
  requirements: string,       // ex: "Temperature -14°C+"
  image_url: string,          // chemin local
}
```

Images

Servies localement via proxy `/api/cards/image/:pointId` (resize sharp). Pas de CDN externe.

Symboles UI

Pas de symboles dédiés pour TM — les ressources (oxygène, température, océan, MC, plant, animal, microbe…) sont rendues comme texte descriptif dans les abilities. Si on veut ajouter des PNG plus tard : créer `frontend/src/lib/tm-symbols.ts` + assets dans `frontend/public/tm-icons/`.

Pas de méta ni deckbuilding

- TM n'a pas de format compétitif structuré (pas de Standard / Modern, pas d'archétypes formels au sens MTG)
- Le mode deckbuilding ne s'applique pas (le "deck" en TM dépend du draft initial, pas d'une construction préalable)
- Le RAG retombe sur retrieval synergy seul si une question deckbuilding est posée

Mise à jour

Quasi-statique. Le jeu de base est figé, seules les extensions occasionnelles ajoutent des cartes :

```
docker exec boardgame-referee npm run cards:tm:parse-html
docker exec boardgame-referee npm run cards:tm:ingest
```

Puis `/admin` → Resync TM cards (si tu veux passer par l'UI).

Cas d'usage typique

- Citer une carte spécifique : `@Birds` → l'oracle voit les détails (effet, points, requirements)
- Question synergy : "Quelles cartes synergisent avec Plant tags ?" → retrieval cartes + chunks règles