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

Revision #1
Created 2026-05-10 15:20:21 UTC by thymon
Updated 2026-05-10 15:20:21 UTC by thymon