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`.

Revision #1
Created 2026-05-10 15:20:24 UTC by thymon
Updated 2026-05-10 15:20:24 UTC by thymon