Tests

Tests

> *Dernière mise à jour : 2026-05-10*

Deux types de tests, complémentaires :

1. **Vitest unitaires** : déterministes, détectent les régressions silencieuses
2. **Banc d'éval RAG** : qualité subjective de l'oracle (judge Haiku)

Vitest unitaires

Périmètre

> **Toute fonction critique non triviale ajoutée à `src/lib/`, `src/services/`, `frontend/src/composables/` ou `frontend/src/stores/` doit avoir un test unitaire** dans le même dossier sous la forme `*.test.ts`.

Les routes et les vues n'en ont pas besoin (pure orchestration), mais leurs services sous-jacents si.

Garde-fous prioritaires

- Schémas Zod (validation API)
- Parsers (markdown, scrapers HTML, decklists)
- Normalisations (clés de cache, slugs, normalize card key)
- Wrappers d'I/O (timeout, retry)
- Stores qui persistent en localStorage

Mock le strict minimum

```
// OK — mock config + repos quand on touche fs / DB
vi.mock('../config.js', () => ({ config: { LOG_LEVEL: 'debug', ... } }))
vi.mock('../repositories/games.repo.js')

// PAS OK — pas de mock fs/path
// → utiliser os.tmpdir() + randomUUID() pour un vrai dossier isolé
```

Plus simple, moins fragile.

Lancer

```
# Backend
npm test              # Run complet
npm run test:watch    # Watch

# Frontend
cd frontend && npm test
```

CI

`npm test` doit **passer** en CI (job `test` dans `.gitea/workflows/build.yml`). Si un test échoue, on corrige le code (ou le test si l'attendu était faux). On ne skip pas.

Banc d'éval RAG

Différence avec les tests unitaires

	Vitest unit	Banc RAG
But	Régressions silencieuses dans la mécanique	Qualité subjective de l'oracle
Source	Code TS	Questions + ground truth
Verdict	Déterministe (assert)	Subjectif (judge LLM)

	Vitest unit	Banc RAG
Coût	Gratuit	Quota Claude (Haiku)
Fréquence	À chaque commit	Manuellement, après gros changements RAG

Localisation

`tests/rag/` — séparé de Vitest, lancé avec ses propres scripts.

Commandes

```
npm run test:rag              # Run complet
npm run test:rag:capture      # Capture les réponses pour archive (snapshot)
npm run test:rag:report       # Rapport markdown
npm run test:rag:html         # HTML interactif
npm run test:rag:html:serve   # Serveur local viewer (port 8888 défaut)
```

Workflow typique

1. Faire un changement RAG significatif (HyDE, RRF, prompt, retrieval params…)
2. `npm run test:rag` → run les questions test
3. `npm run test:rag:html` → générer rapport HTML interactif
4. `npm run test:rag:html:serve` → ouvrir http://localhost:8888 et comparer les réponses
5. Décider : régression acceptable / fix nécessaire

Alimenter le banc

- Source initiale : questions choisies par toi qui couvrent les cas critiques (rules / synergy / meta / deckbuilding par TCG)
- Auto-feed : feedbacks votés `down` avec commentaire dans `/admin/feedback` deviennent des candidats naturels
- Manuel actuellement (pas d'extraction auto). Workflow propre :
  1. Sélectionner les feedbacks down avec commentaire
  2. Reformuler en cas-test
  3. Ajouter à `tests/rag/dataset.json`
  4. `npm run test:rag` pour mesurer la régression

Tests frontend

Vitest + Vue Test Utils. Localisé sous `frontend/src/**/*.test.ts`.

Couvre :

- Composables (`useAskStream`, `useMentionAutocomplete`, `useArbiterMarkdown`, `useStickyMentions`, etc.)
- Stores Pinia (`auth`, `games`, `session`)
- Helpers (`mana.ts`, `fab-symbols.ts`, etc.)

Pas de tests E2E Playwright actuellement. Pourrait être ajouté plus tard pour les flows critiques (login → ingest jeu test → poser question → vérifier réponse).

Coverage

Pas de seuil enforcé. Vitest peut générer un rapport `--coverage` :

```
npm test -- --coverage
```

Mais pas de fail-on-low-coverage. Convention : si tu touches une fonction critique, le test existe.

Pièges connus

Mock partiel d'un module ESM

- Vitest avec ESM peut se mélanger les pinceaux sur `vi.mock`. Toujours déclarer le mock **avant** l'import du module à tester.

Tests qui dépendent de la timezone

- Garder `process.env.TZ = 'Europe/Paris'` ou `'UTC'` constant
- Sinon des tests passent en local et fail en CI selon le serveur

Tests qui dépendent de fichiers réels

- Préférer `os.tmpdir()` + `randomUUID()` plutôt que mock `fs`
- Cleanup dans `afterEach` (`fs.rm(dir, { recursive: true })`)

Tests qui appellent vraiment l'API Claude

- ⚠ Jamais en CI (coût + flake)
- Mock le service ou utiliser le mode `CLAUDE_USE_LOCAL=true` avec un binaire stub

---

Revision #1
Created 2026-05-10 15:20:04 UTC by thymon
Updated 2026-05-10 15:20:04 UTC by thymon