ADR-007 : Handlers MVC (Phase 4)

ADR-007 : Handlers MVC (Phase 4)

> *Date : 2026-Q2 (Phase 4 refactoring) — Statut : en cours / accepté*

Contexte

Avec le repo pattern (ADR-006), les routes étaient devenues plus propres :

```
// routes/games.ts (avant Phase 4)
app.post('/api/games/ingest', requireAuth, requireConfirmed, requireCanAddGames, async (c) =>
{
  const body = await c.req.parseBody();
  const validated = gameIngestMetadataSchema.parse(body);

  // 50 lignes de logique : upload PDF, validation extension parent, démarrage ingest...

  const game = await gamesRepo.upsert(...);
  // … encore 30 lignes

  return c.json({ gameId: game.id });
});
```

Mais les routes contenaient encore beaucoup de **logique métier** :

- Validation cross-table (parent existe ?)
- Décisions (ingest immédiat vs scheduled ?)
- Manipulation fichiers (PDF upload sur disque)
- Orchestration services (startIngestJob)

Conséquences :

- Routes de 100+ lignes
- Difficile à tester sans démarrer Hono
- Logique métier mélangée avec préoccupations HTTP

Décision

Architecture en 4 couches : `routes` → `handlers` → `services` → `repositories`.

```
// routes/games.ts (Phase 4)
import { ingestGameHandler } from '../handlers/games/ingest.js';

app.post('/api/games/ingest', requireAuth, requireConfirmed, requireCanAddGames, async (c) =>
{
  const formData = await c.req.parseBody();
  const validated = gameIngestMetadataSchema.parse(formData);
  const pdfBuffer = await (formData['pdf'] as File).arrayBuffer();
  const userId = c.get('user').id;

  const result = await ingestGameHandler(validated, Buffer.from(pdfBuffer), userId);

  if (!result.ok) return c.json({ error: result.error }, result.status);
  return c.json({ gameId: result.gameId, scheduled: result.scheduled });
});
```

```
// handlers/games/ingest.ts
type IngestResult =
  | { ok: true; gameId: string; scheduled?: boolean; scheduledStartAt?: string }
  | { ok: false; status: 400 | 404; error: string };

export async function ingestGameHandler(
  metadata: GameIngestMetadata,
  pdfBuffer: Buffer,
```

```
  userId: string,
): Promise<IngestResult> {
  // Validation cross-table
  if (metadata.parentGameId) {
    const parent = await gamesRepo.getById(metadata.parentGameId);
    if (!parent) return { ok: false, status: 404, error: 'parent game not found' };
  }

  // Manipulation fichiers
  const sourceFile = `/app/pdfs/${slug}-${Date.now()}.pdf`;
  await fs.writeFile(sourceFile, pdfBuffer);

  // Décision ingest immédiat vs scheduled
  if (metadata.scheduledStartAt) {
    const game = await gamesRepo.upsert({ ..., ingestStatus: 'scheduled' });
    scheduleIngestStart(game.id, new Date(metadata.scheduledStartAt));
    return { ok: true, gameId: game.id, scheduled: true, scheduledStartAt };
  }

  // Ingest immédiat
  const game = await gamesRepo.upsert({ ..., ingestStatus: 'idle' });
  startIngestJob(game.id);
  return { ok: true, gameId: game.id };
}
```

Règles du handler

1. **Pas d'import Hono** : un handler ne connaît pas le framework. Pas de `c.json`, pas de `c.req`, pas de `Context`.
2. **Reçoit des données validées** : le handler suppose que l'input est déjà passé par Zod côté route.
3. **Reçoit les dépendances explicites** : `pdfBuffer` (pas `formData`), `userId` (pas le cookie session).
4. **Retourne un Result discriminé** : `{ ok: true; ... } | { ok: false; status: ...; error: string }` pour les erreurs **attendues**. Pour les bugs / timeouts / erreurs non-récupérables, on `throw` normalement (capturé par le handler global Hono).
5. **Status HTTP dans le Result** : la route mappe le `status` au `c.json(..., status)`. Garde la connaissance HTTP côté route.

Conséquences

Bonnes

- **Routes plus courtes** : 5-15 lignes, juste parsing + délégation
- **Handlers testables sans Hono** : `await ingestGameHandler(meta, buf, 'user-123')` directement en Vitest
- **Logique métier centralisée** : ouvrir `handlers/games/ingest.ts`, voir tout ce qui se passe
- **Result discriminé** : TypeScript force à gérer tous les cas (ok / 400 / 404)
- **Réutilisabilité** : un handler peut être appelé par plusieurs routes ou par un script CLI

Mauvaises

- **Plus de fichiers** : `handlers/games/{ingest,delete,update,...}.ts`. Friction au refactor (~50% plus de fichiers).
- **Verbosité du Result** : pour des erreurs simples, le Result discriminé est plus verbeux que `throw new HTTPError(404, 'not found')`.
- **Migration en cours** : pas tous les routes sont passées au pattern. Cohabitation `routes-style` ancien et `routes + handlers` nouveau pendant la transition.

État de la migration (2026-05-10)

Route	Statut
`POST /api/games/ingest`	Phase 4 (handler)
`DELETE /api/games/:id`	Phase 4 (handler)
`POST /api/ask/stream`	encore en pattern routes-direct (logique RAG complexe, refactor reporté)
`POST /api/auth/*`	pattern routes-direct (logique simple, pas urgent)
`POST /api/admin/*`	mixte

À chaque nouveau besoin sur une route, l'extraire en handler en passant.

Workflow "ajouter une route"

1. Ajouter le schéma Zod dans `src/lib/schemas.ts` (si réutilisable)
2. Créer le handler dans `src/handlers/<domaine>/<action>.ts`

3. Définir le type `Result` discriminé
4. Implémenter la logique en utilisant les services + repos
5. Écrire le test Vitest du handler (mock services / repos)
6. Brancher la route Hono qui parse + valide + délègue + map le Result en réponse HTTP

Revision #1
Created 2026-05-10 15:19:54 UTC by thymon
Updated 2026-05-10 15:19:54 UTC by thymon