结论先行：**你的同步分工思路是对的，但要把「接口契约」写死得足够早、足够稳。**

做mock JSON让两条流水线并行是敏捷常用招式，关键是保证**定位方式、主键、资源路径**最终不会因为真-EPUB 解析而变动；否则一旦偏了，后期返工比等你把解析脚本写完更费时间。

逐点点评

你的步骤	合理性	建议补强
1. 先手工 / 脚本做 chapter.json 样本，交给工作流团队	☑ 能并行开发、压缩总体周期。	- **文件尽量从真实 EPUB 拆一章生成**，不要人工拼；确保标签、换行、特殊字符和正式版一致。 - **字段粒度不要拍脑袋**：把你真正想保留的 CFI、段落 ID、innerHTML 都列进去，宁可多一列也别后面加。
2. 先锁定工作流输出格式，再落表	☑ 接口先行是正确的。	- 定义 **version 字段**，以后扩展靠 semver； - 明确 **必填 / 可选字段**，例如 `audio_url` 可能为空； - 约定 **错误/空占位**（如 `null` 或空数组），否则批量 insert 时报错。
3. 双方同步进行	☑ 整体合理	在 Notion / Confluence 开**「数据契约」页面**，把： ① `chapter.json` schema ② 工作流输入输出 schema ③ 数据库表映射 先写成表格，所有变更必须先 PR 这份文档。

你现在就该定死的 6 个要素

1. **定位方式**
   - 用 **EPUB CFI**（`epubcfi(/6/12[xhtml]!/4/2/14)`) or **段落索引 + 字符偏移**?
   - 一旦决定，chapter 解析脚本和工作流都按同一套来，后期不可改。
2. **主键 & 外键**
   - `chapter_id`, `hotspot_id` 生成规则（UUID 还是自增）。
   - 工作流输出里就带着这些 ID，落表不需要再二次查找。
3. **资源路径策略**
   - 图片/音频直接 OSS URL 还是先落本地再上传?
   - 建议工作流拿到 **pre-signed upload URL**，回写的就是可公开读的最终地址。
4. **版本控制**
   - 给 `chapter.json` 加 `"schema_version": "1.0.0"`。
   - 工作流输出加 `"workflow_version": "ai-v0.4"`。
   - 将来升级拆分表或加字段，不会把历史数据搞崩。
5. **失败回退字段**

- `status`: pending / success / error
- `error_message`: text
- 方便你们做重试、监控。

6. **时间戳 & 作者**

- `created_at`, `updated_at`, `generated_by` 保证可审计。

最小可行落地路径（推荐）

1. **写一个原型解析脚本**

- 用 `unzip && grep '</body>'` 把第一章拆成 HTML。
- 输出 `chapter.json`（至少 3 段文本、模拟 2 个 hotspot）。

2. **Workflow 团队从 `chapter.json` → 返回**

```
{
  "hotspot_id": "uuid-123",
  "image_url": "https://cdn/.../uuid.jpg",
  "audio_url": null,
  "prompt": "{...}",
  "cfi": "epubcfi(...)"
}
```

3. **你写一段临时脚本**

- 读工作流结果 → insert into `hotspots` / `assets`。
- React 前端直接 fetch S3 HTML + 再拉热点表塞圆点。

4. **第一个可点击-弹图 demo 出炉**

- 能完整跑通「显示文本 → 点击圆点 → 看到图片」。
- 接下来再把解析脚本完善、批量跑全书。

可能踩坑提醒

坑	规避方式
EPUB 解析后段落变动，offset 全错位	先定 CFI；mock JSON 也用同算法生成。
工作流早期用 HTTP 链接占位，后来你想换 CDN 域名	一开始就用 `https://img.example.com/{id}.jpg` 这种逻辑地址；到时改 DNS / CDN，不动数据库。
批量插表锁表 & 性能	用 `COPY` 或批量 insert；热路径查询加 composite index `(chapter_id, offset)`.

结论（直说）

- **思路可行且典型敏捷**，关键是 *schema first*：把字段、ID、定位方式一次写死，再 mock。
- mock JSON 一定要**用真实解析方法**产出，别手敲；否则定位/转义差异会坑你。
- 所有团队对 **「数据契约文档」** 建立强依赖，任何改动先改文档再改代码。

照这条路走，你们就能并行开发，且把未来重构成本压到最低。