

ARCHITECTURE SPECIFICATION

AERO-KB

Knowledge Base as Code cho dự án hàng không —
tối ưu chi phí token trong quy trình BA / Dev / Agent

Phiên bản	0.1 — Draft for review
Ngày	09/07/2026
Tác giả	Lam (PM) — soạn cùng Claude
Phạm vi	Full specification + roadmap triển khai
Đối tượng đọc	BA, Dev, SME, PM các dự án SIA

1. Tóm tắt

AERO-KB là một bộ kit (Python package + CLI + MCP server) biến các tài liệu chính thống của ngành hàng không — ARINC 424, ICAO Annex, quy định khai thác mặt đất, SOP nội bộ — thành **knowledge base sống ngay trên source code**, được version bằng Git và tối ưu cho AI agent truy vấn.

Vấn đề cốt lõi: tài liệu hàng không cực lớn (một mình ARINC 424 đã hơn 1.000 trang), trong khi agent chỉ có ngân sách context giới hạn và mỗi token đều có chi phí. Nếu mỗi lần BA viết requirement hay Dev implement đều phải "đọc lại" tài liệu gốc, chi phí token bùng nổ và chất lượng không ổn định.

AERO-KB giải quyết bằng ba cơ chế:

- **Compile once, query many** — tài liệu được cô đọng một lần duy nhất thành cấu trúc 4 tầng (L0-L3); chi phí LLM chỉ tốn lúc "compile", còn truy vấn về sau gần như miễn phí.
- **Progressive disclosure** — agent chỉ load đúng phần cần: index luôn nhỏ hơn 1K token, chi tiết chỉ được nạp khi tag hoặc câu hỏi khớp đúng section.
- **Requirement mang theo context** — mỗi requirement BA viết ra đính kèm block citation máy-đọc-được (doc-id, section, KB version); Dev nhận ticket là nhận luôn đúng nguồn tri thức BA đã dùng, không tam sao thất bản.

Tài liệu này mô tả kiến trúc chi tiết, flow làm việc cho từng role (BA, Dev, SME, PM) và roadmap triển khai 3 phase.

Mục lục

1. Tóm tắt	2
2. Bài toán & mục tiêu	3
3. Nguyên tắc thiết kế	3
4. Kiến trúc tổng thể	4
5. Cấu trúc knowledge base 4 tầng	5
6. Pipeline ingest tài liệu	7
7. Tag routing & retrieval	8
8. Codebase như một knowledge base	9
9. Federation cho microservices / multi-repo	10
10. Interface: CLI & MCP	11
11. Flow theo từng role	12
12. Governance, versioning & bản quyền	15
13. Roadmap triển khai	16
14. Rủi ro & giảm thiểu	17
Phụ lục A. Ví dụ requirement kèm kb-context	18
Phụ lục B. Ví dụ manifest của một tài liệu	18

2. Bài toán & mục tiêu

2.1 Bối cảnh

Các dự án hàng không (Talora / ARINC 424 dashboard, các hệ thống SIA...) phụ thuộc vào một khối lượng tài liệu chuẩn ngành khổng lồ: đặc tả ARINC 424, các ICAO Annex, quy định khai thác mặt đất, SOP nội bộ của hãng. Khi đưa AI agent vào quy trình — BA viết requirement qua agent, Dev code bằng agent — mọi câu trả lời đúng đều phải bám vào các tài liệu này.

Cách làm ngây thơ (đưa cả tài liệu vào context, hoặc RAG thuần túy trên raw text) gặp ba vấn đề:

- **Chi phí token:** tài liệu hàng nghìn trang, mỗi truy vấn kéo theo hàng chục nghìn token dư thừa, lặp lại mỗi lần hỏi.
- **Chất lượng không ổn định:** retrieval trên raw text dễ trả về nhầm section; agent "tự tin" trích sai điều khoản — rủi ro nghiêm trọng với domain hàng không.
- **Không có traceability:** không trả lời được câu hỏi "requirement này viết dựa trên tài liệu nào, bản sửa đổi nào?" — điều gần như bắt buộc trong môi trường có audit.

2.2 Mục tiêu

Mục tiêu	Thước đo
Giảm chi phí token cho truy vấn tri thức domain	$\geq 90\%$ so với việc nạp raw document; index thường trực $< 1K$ token
Mọi requirement / câu trả lời agent đều có citation về nguồn chính thống	100% requirement đạt Definition of Ready có block kb-context
BA và Dev dùng chung một nguồn tri thức, đúng version	Citation trong ticket resolve được về đúng section, đúng commit
Cập nhật tài liệu (amendment) là sự kiện phát hiện được	kb doctor cảnh báo mọi ref bị stale
Hoạt động được với mô hình microservices / multi-repo	Thêm 1 repo mới không làm tăng đáng kể token cost của truy vấn

2.3 Ngoài phạm vi (non-goals)

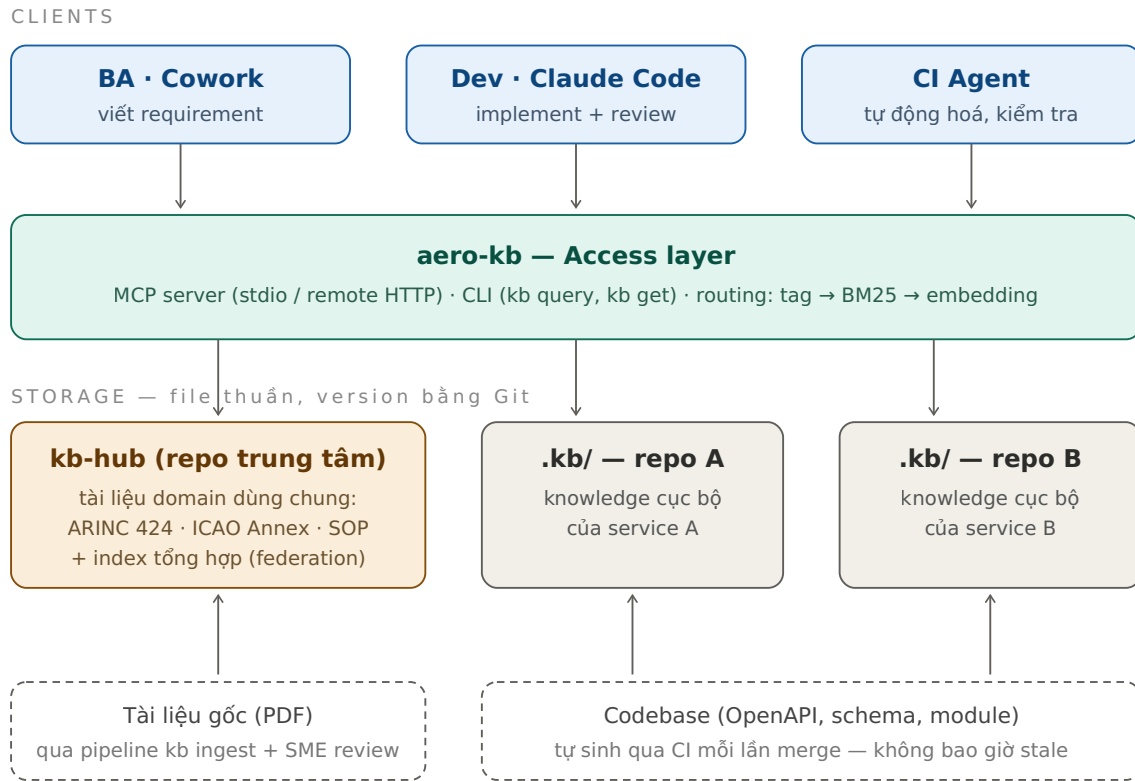
- Không thay thế hệ thống quản lý tài liệu chính thống (nguồn gốc vẫn là bản phát hành của ICAO / ARINC / hãng).
- Không phải search engine toàn văn cho người dùng cuối — đối tượng phục vụ là AI agent và quy trình BA/Dev.
- Phase đầu không xây dựng UI riêng; giao diện là CLI, MCP và chính các agent client (Cowork, Claude Code).

3. Nguyên tắc thiết kế

Nguyên tắc	Ý nghĩa
Docs-as-code	KB là các file markdown/YAML nằm trong repo, version bằng Git, review bằng PR — không cần database hay hạ tầng riêng ở phase đầu.
Progressive disclosure	Tri thức chia tầng L0→L3; agent đi từ nông xuống sâu, chỉ nạp phần cần thiết (cùng pattern với SKILL.md của Anthropic).
Compile once, query many	Chi phí LLM (summarize) chỉ trả một lần lúc ingest; truy vấn về sau là đọc file thuần + routing bằng code.
Citation bắt buộc	Mọi output của agent dựa trên KB phải cite <code>doc-id §section</code> . Với hàng không, traceability không phải tùy chọn.
Human-in-the-loop	Summary do LLM sinh ra phải được SME duyệt trước khi merge; summary sai thì mọi requirement dựa trên nó sai theo.
Routing bằng code, không bằng LLM	Tag matching, BM25, ranking chạy deterministic ngoài model — không tốn token, không sai ngẫu nhiên.

4. Kiến trúc tổng thể

AERO-KB gồm ba lớp: **storage** (các thư mục `.kb/` và `kb-hub`), **access** (CLI + MCP server), và **clients** (các agent). Toàn bộ storage là file thuần trong Git — MCP server chỉ là lớp giao tiếp, không phải nền móng.



Hình 1 — Kiến trúc tổng thể: *clients* → *access layer* → *storage*. Mũi tên từ nguồn đi lên *storage* thể hiện pipeline nạp tri thức.

Vì sao cần MCP server? Không có nó, KB vẫn hoạt động (agent Read/Grep trực tiếp). MCP server bổ sung 3 giá trị: (1) routing chạy bằng code — không tốn token, không sai ngẫu nhiên; (2) kiểm soát token budget tập trung — server quyết định trả tầng nào, cắt theo budget, kèm citation chuẩn; (3) một interface chung cho mọi client. Dạng chuẩn là **stdio local process** khai báo trong `.mcp.json` của repo — không cần deploy, ai clone repo là dùng được.

5. Cấu trúc knowledge base 4 tầng

Mỗi tài liệu sau khi ingest được tổ chức thành 4 tầng, từ nông đến sâu. Nguyên tắc: **tầng càng nông càng rẻ và càng hay được đọc; tầng sâu chỉ nạp khi thực sự cần.**

L0 — index.yaml

Mỗi tài liệu 1 dòng: ID, tags, mô tả ngắn. Agent luôn thấy "kho có gì".

~0,3-0,5K token

luôn trong context

L1 — _manifest.yaml (mỗi tài liệu)

Cây section + 1 câu summary/section. Load khi tài liệu được kích hoạt.

~1-3K token/tài liệu

load theo tag match

L2 — section đã cô đọng (.md)

Summary có cấu trúc + bảng/giá trị quan trọng giữ nguyên vẹn.

~0,5-2K token/section

load đúng section cần

L3 — trích xuất gốc (.raw.md)

Nguyên văn theo section — chỉ đọc khi cần verify nguyên văn.

theo nhu cầu

hiếm khi cần

Từ trên xuống: độ sâu và chi phí tăng dần — tầng sâu chỉ được load khi thực sự cần

Hình 2 — Bốn tầng progressive disclosure. Đa số truy vấn dừng ở L1-L2.

5.1 Cấu trúc thư mục

```
.kb/
├── index.yaml           # L0 — danh mục toàn bộ KB
├── arinc-424/
│   ├── _manifest.yaml  # L1 — cây section + summary
│   ├── ch5-nav-data.md  # L2 — nội dung cô đọng
│   ├── ch5-nav-data.raw.md # L3 — trích xuất gốc
│   └── ...
├── icao-annex-14/
│   └── ...
└── ground-ops/
    └── ...

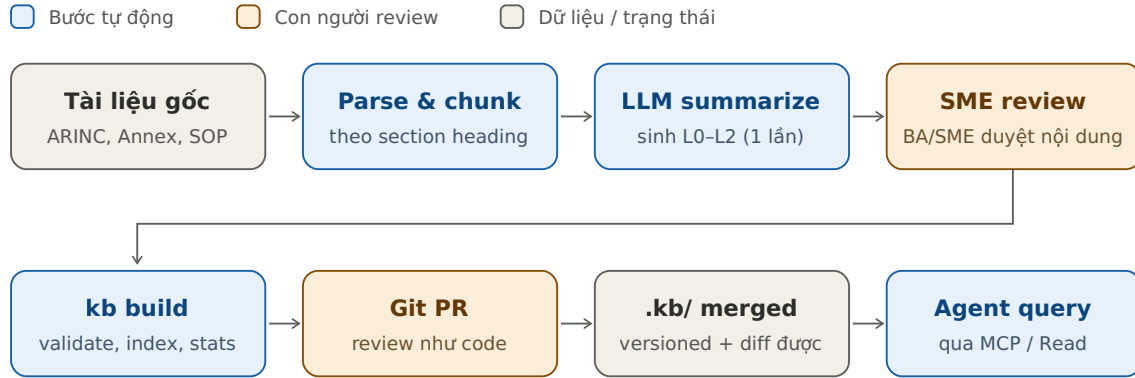
# quy định mặt đất, SOP nội bộ
```

5.2 Ví dụ một entry trong index.yaml (L0)

```
docs:
- id: arinc-424
  title: "ARINC 424 – Navigation System Data Base"
  revision: "Supplement 22"
  tags: [arinc424, navdata, airspace, airport, navaid, airway]
  summary: "Đặc tả cấu trúc dữ liệu dẫn đường: record types, field, coding rules."
- id: icao-annex-14
  title: "ICAO Annex 14 – Aerodromes"
  revision: "Ed 9, Amdt 17"
  tags: [icao, aerodrome, runway, lighting]
  summary: "Tiêu chuẩn thiết kế và khai thác sân bay."
```

6. Pipeline ingest tài liệu

Thêm một tài liệu mới vào KB là một quy trình có kiểm soát, kết thúc bằng PR review như code. Chi phí LLM chỉ phát sinh một lần tại bước summarize.



Khi tài liệu có amendment: re-ingest → tool diff với bản cũ → SME chỉ review phần thay đổi

Hình 3 — Pipeline ingest: từ PDF gốc đến KB sẵn sàng cho agent truy vấn.

6.1 Các bước chi tiết

- Ingest:** `kb ingest annex14.pdf --id icao-annex-14 --tags icao,aerodrome` — parse PDF, chunk theo heading, giữ nguyên bảng và giá trị số.
- Summarize:** LLM sinh summary từng section (L2), mục lục kèm 1 câu mô tả (L1), entry danh mục (L0). Bảng/field spec quan trọng được giữ nguyên văn, không "tóm tắt hệ".
- SME review:** BA/SME đọc và sửa summary — bước bắt buộc; với domain hàng không, summary sai nguy hiểm hơn không có summary.
- Build:** `kb build` — validate schema, cập nhật index, tính token stats từng tầng (`kb stats`).
- PR & merge:** KB thay đổi qua PR như code. Git history trả lời được "requirement X viết dựa trên bản tài liệu nào".

Amendment flow: ICAO Annex sửa đổi theo chu kỳ. Khi có bản mới: chạy lại `kb ingest` → tool tự diff summary với bản cũ → SME chỉ review phần thay đổi → `kb doctor` quét toàn bộ requirement đang cite các section bị ảnh hưởng và cảnh báo cho BA.

7. Tag routing & retrieval

Khi BA viết requirement có tag (#arinc424 #airspace) hoặc nêu đích danh tài liệu, resolver tìm đúng section theo thứ tự **từ rẻ đến đắt** — đa số truy vấn dừng ở bước 1-2 với chi phí token gần bằng 0:

Bước	Cơ chế	Cách hoạt động	Chi phí
1	Exact tag / doc-name match	Tag trong requirement khớp với tags trong L0/L1; nêu đích danh tài liệu thì kích hoạt thẳng manifest của nó.	~0 token, deterministic
2	Keyword / BM25	Tìm trên toàn bộ manifest (L1) — dữ liệu nhỏ nên nhanh và chính xác cao.	~0 token, chạy bằng code
3	Embedding search (tùy chọn)	sqlite-vec local trên L1+L2, chỉ dùng khi 2 bước trên không đủ. Không cần hạ tầng ngoài.	Rẻ, chạy local

Kết quả trả về luôn là **danh sách section đã xếp hạng kèm citation**, và server cắt nội dung theo token budget do client khai báo. Agent không bao giờ nhận về "cả chương" nếu chỉ hỏi một field.

```
# BA viết trong Cowork:
"Requirement: hiển thị thông tin restrictive airspace khi click
trên bản đồ #arinc424 #airspace"

# Resolver:
tag match → arinc-424 → manifest → §5.3 Restrictive Airspace (UR/PR records)
trả về: L2 của §5.3 (~800 token) + citation "arinc-424 §5.3 (Suppl 22)"
```

8. Codebase như một knowledge base

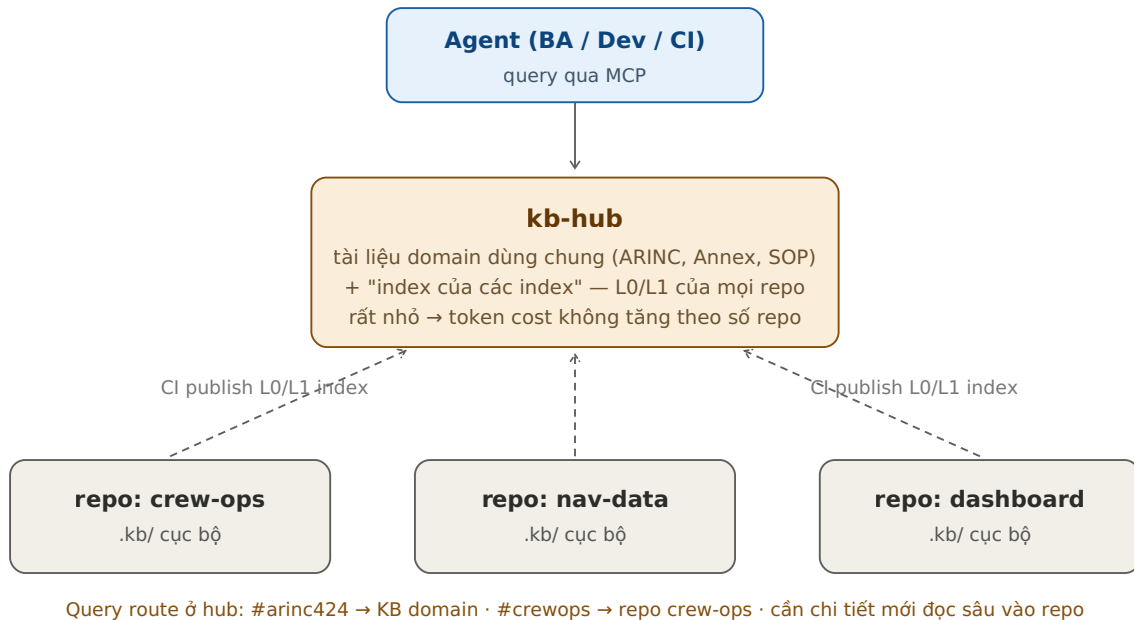
Code khác tài liệu ở một điểm cốt tử: **tài liệu ổn định, code thay đổi hàng ngày**. Nếu summarize code bằng tay như tài liệu, KB sẽ stale sau vài sprint — và KB stale nguy hiểm hơn không có KB. Vì vậy tri thức về code tách làm hai loại với hai vòng đời khác nhau:

	Generated (tự sinh)	Curated (con người viết)
Nội dung	API contract (OpenAPI), DB schema, danh sách module/service, dependency graph	Lý do kiến trúc, mapping domain→code ("restrictive airspace xử lý ở service X, bảng UR trong arinc424.sqlite"), quy ước, edge case đã trả giá
Cách tạo	Extract bằng code (không cần LLM), chạy trong CI mỗi lần merge	Viết tay, review PR như tài liệu domain
Độ tươi	Không bao giờ stale (sinh lại tự động)	Thay đổi chậm, ít, đáng công duy trì
Nguy cơ	Thấp	Phải có owner; nếu không có vòng feedback sẽ chết dần

Bản thân source code thì agent đọc trực tiếp — **code là ground truth**, không nhét vào KB. KB chỉ giữ những gì đọc code không suy ra được.

9. Federation cho microservices / multi-repo

Mô hình mở rộng tự nhiên cho nhiều repo: mỗi repo giữ `.kb/` riêng cho tri thức cục bộ; một **kb-hub** trung tâm chứa tài liệu domain dùng chung (không bao giờ duplicate vào từng repo) và index tổng hợp do CI của từng repo publish lên.



Hình 4 — Federation: hub là "index của các index", các repo publish index qua CI (mũi tên đứt).

9.1 Tính chất quan trọng

- **Token cost gần như không tăng theo số repo** — tầng routing "service nào" chỉ thêm vài chục token ở L0 của hub.
- **Tài liệu domain một bản duy nhất** ở hub; các repo tham chiếu bằng doc-id, không copy.
- **Rủi ro chính: index ở hub lệch với repo con** (CI publish fail). `kb doctor` so commit hash để phát hiện index stale và đánh dấu cho agent biết.

10. Interface: CLI & MCP

10.1 CLI

Lệnh	Chức năng
<code>kb ingest <file> --id --tags</code>	Parse + chunk + summarize tài liệu mới (hoặc bản amendment).
<code>kb build</code>	Validate schema, cập nhật index, sinh artifacts.
<code>kb query "... " --tags --budget</code>	Truy vấn thử từ terminal, trả section + citation trong token budget.
<code>kb get <doc-id> §<section> --level L2</code>	Lấy đúng một section ở tầng chỉ định.
<code>kb diff <doc-id> --against <rev></code>	So sánh summary giữa hai bản tài liệu (phục vụ amendment review).
<code>kb doctor</code>	Kiểm tra sức khỏe: ref stale, index lệch hub, citation gãy.
<code>kb stats</code>	Token size từng tầng, từng tài liệu — theo dõi chi phí.

10.2 MCP tools

Tool	Chức năng
<code>kb_search(query, tags?, budget?)</code>	Routing 3 bước, trả danh sách section xếp hạng + nội dung trong budget + citation.
<code>kb_get_section(doc, section, level)</code>	Lấy chính xác một section (dùng khi resolve citation từ ticket).
<code>kb_resolve(kb_context)</code>	Nhận block kb-context từ ticket, trả toàn bộ section được cite — đúng version đã pin.

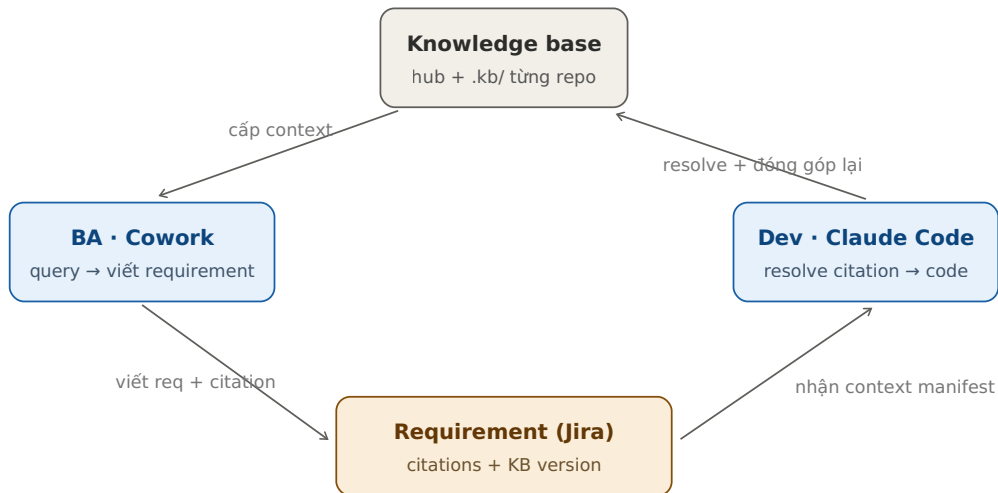
10.3 Khai báo trong repo

```
// .mcp.json – đặt tại root repo, ai clone là dùng được
{
  "mcpServers": {
    "aero-kb": {
      "command": "python",
      "args": ["-m", "aero_kb.mcp", "--kb", ".kb/", "--hub", "git@...:kb-hub.git"]
    }
  }
}
```

Phase sau, khi BA không clone repo và nhiều team dùng chung: chuyển transport sang **remote HTTP MCP** deploy nội bộ — code gần như giữ nguyên, chỉ đổi lớp giao tiếp.

11. Flow theo từng role

Nguyên tắc phối hợp: **một KB duy nhất, và chính requirement là sợi dây kết nối**. Requirement mang theo "context manifest" — block citation máy-đọc-được — để Dev nhận được chính xác nguồn tri thức BA đã dùng.



Hình 5 — Vòng phối hợp BA-Dev quanh một KB duy nhất; requirement là vật mang context.

11.1 Flow của BA (viết requirement)

- Soạn requirement trong Cowork** — gắn tag nội dung (`#arinc424 #airspace`) hoặc nêu đích danh tài liệu.
- Agent tự query KB** qua `kb_search` — nhận đúng section liên quan (L1/L2) kèm citation, trong token budget.
- Viết requirement dựa trên nguồn** — mọi khẳng định về chuẩn ngành phải bám citation; agent chèn sẵn `doc-id §section`.
- Sinh block `kb-context`** (refs + tags + KB commit hash) và đính vào Jira ticket.
- Definition of Ready**: requirement liên quan chuẩn ngành mà thiếu citation thì **chưa đạt DoR** — đây là convention bắt buộc của team.

11.2 Flow của Dev (implement)

- Mở ticket bằng Claude Code** — agent đọc block `kb-context` trong ticket.
- `kb_resolve(kb_context)`** — nhận chính xác các section BA đã dùng, đúng version đã pin. Không search lại từ đầu, không tam sao thất bản.
- Kiểm tra freshness** — nếu `kb doctor` báo section đã thay đổi so với HEAD (amendment sau khi BA viết), trao đổi lại với BA trước khi code.
- Implement** — kết hợp KB domain + tầng mapping domain→code trong `.kb/` của repo; source code vẫn là ground truth.
- Đóng góp ngược** — phát hiện KB thiếu/mơ hồ thì mở PR vào KB; tri thức quay về hub, BA sau này hưởng lợi.

11.3 Flow của SME / Reviewer

- 1 **Review ingest** — duyệt summary do LLM sinh trước khi merge; sửa trực tiếp trên PR.
- 2 **Review amendment** — khi tài liệu có bản sửa đổi, chỉ review phần `kb diff` chỉ ra là thay đổi.
- 3 **Duyệt đóng góp từ Dev/BA** — gác cổng chất lượng cho phần curated knowledge.

11.4 Flow của PM

- 1 **Theo dõi sức khỏe KB** — `kb doctor` chạy định kỳ trong CI: ref stale, citation gãy, index lệch hub.
- 2 **Theo dõi chi phí** — `kb stats` cho biết token size từng tầng; bất thường (L0 phình to) là tín hiệu cần tái cấu trúc.
- 3 **Enforce quy trình** — DoR có citation, SME review bắt buộc, KB thay đổi qua PR.

12. Governance, versioning & bản quyền

12.1 Versioning & chống drift

- **Pin version:** block `kb-context` trong ticket ghi commit hash của kb-hub tại thời điểm BA viết. Từ lúc viết đến lúc code có thể vài tuần — Dev luôn đọc được đúng bản BA đã dùng.
- **Thay đổi tài liệu là sự kiện phát hiện được:** `kb doctor` đối chiếu mọi ref đang pin với HEAD; section thay đổi → cảnh báo trên ticket, BA xác nhận lại thay vì lỗi ngầm.
- **Git history = audit trail:** trả lời được "requirement TAL-xxxx viết dựa trên ARINC 424 bản nào" — bằng chính lịch sử commit.

12.2 Quyền truy cập theo role

Role	Cửa vào	Phạm vi nhìn thấy
BA	Cowork + remote MCP (không cần clone repo)	Tầng domain của hub (L0-L2); không cần tầng code
Dev	Claude Code + stdio MCP trong repo	Toàn bộ: domain + mapping domain→code của repo mình
SME	Git PR review	Nội dung KB trước khi merge
CI	stdio MCP / CLI	Kiểm tra tự động, publish index lên hub

12.3 Bản quyền tài liệu

Lưu ý pháp lý: ARINC 424 và các ICAO Annex là tài liệu có bản quyền/giấy phép. KB dạng summary + trích dẫn chọn lọc phục vụ nội bộ, đặt trong repo **private**, là hợp lý; tuyệt đối không commit raw full-text ra repo công khai hoặc chia sẻ ngoài phạm vi giấy phép tổ chức đang có. Tầng L3 (raw extract) cần cân nhắc giữ tối thiểu — chỉ những section thực sự cần verify nguyên văn.

13. Roadmap triển khai

Phase	Thời lượng gợi ý	Nội dung	Tiêu chí hoàn thành
1 — PoC	2-3 tuần	Python package + CLI (<code>ingest</code> / <code>build</code> / <code>query</code> / <code>stats</code>); cấu trúc <code>.kb/</code> 4 tầng; ingest thử 1-2 chương ARINC 424 + 1 Annex; SME review quy trình đầu tiên.	BA query thử từ CLI ra đúng section, đúng citation; token stats chứng minh mức tiết kiệm.
2 — Tích hợp workflow	3-4 tuần	MCP server studio + <code>.mcp.json</code> ; block <code>kb-context</code> chuẩn hóa + tích hợp Jira; <code>kb_resolve</code> , <code>kb_diff</code> , <code>kb_doctor</code> ; đưa DoR có citation vào quy trình team.	1 requirement đi trọn vòng BA → Jira → Dev với citation resolve được; amendment thử nghiệm được phát hiện.
3 — Scale	4-6 tuần	kb-hub federation + CI publish index; codebase extraction (OpenAPI, schema) tự động; remote HTTP MCP cho BA không clone repo; embedding search (sqlite-vec) nếu cần.	≥ 2 repo tham gia federation; <code>kb_doctor</code> chạy định kỳ trong CI; BA dùng qua remote MCP.

14. Rủi ro & giảm thiểu

Rủi ro	Mức độ	Giảm thiểu
Summary do LLM sinh sai lệch so với tài liệu gốc	Cao	SME review bắt buộc trước merge; bảng/giá trị số giữ nguyên văn; L3 raw luôn sẵn để verify.
KB curated chết dần vì không ai cập nhật	Trung bình	Vòng feedback Dev→KB là một phần của DoD; PM theo dõi qua <code>kb doctor</code> ; phần generated tự sinh nên không phụ thuộc kỷ luật.
Drift giữa lúc BA viết và lúc Dev code	Trung bình	Pin commit hash trong <code>kb-context</code> ; <code>kb doctor</code> cảnh báo section thay đổi.
Index hub lệch với repo con (CI publish fail)	Trung bình	<code>kb doctor</code> so commit hash, đánh dấu index stale cho agent.
Vi phạm giấy phép tài liệu chuẩn ngành	Cao	Repo private; chỉ summary + trích chọn lọc; L3 tối thiểu; rà soát phạm vi license của tổ chức.
Team không theo convention (tag, citation)	Trung bình	Đưa vào DoR/DoD; agent tự chèn citation nên gánh nặng thao tác gần bằng 0; lint tự động trong CI.

Phụ lục A — Ví dụ requirement kèm kb-context

Jira TAL-1580 – Hiển thị pop-up Restrictive Airspace

Là một dispatcher, tôi muốn click vào restrictive airspace trên bản đồ để xem thông tin chi tiết (designation, type, multiple code, level...).

Acceptance criteria:

- Hiển thị các field P1 theo đúng coding rule của UR record [arinc-424 §5.3]
- Loại airspace (P, R, D, A, C...) hiển thị theo bảng phân loại [arinc-424 §5.3.2]

kb-context:

```
version: a3f9c21          # commit hash kb-hub lúc viết
refs:
  - arinc-424 §5.3        # Restrictive Airspace (UR/PR records)
  - arinc-424 §5.3.2      # Airspace type codes
tags: [arinc424, airspace, talora]
```

Phụ lục B — Ví dụ manifest của một tài liệu (L1)

```
# .kb/arinc-424/_manifest.yaml
id: arinc-424
title: "ARINC 424 – Navigation System Data Base"
revision: "Supplement 22"
ingested: 2026-07-09
source_sha256: "...
sections:
  - id: "5.1"
    title: "Airport Records (PA)"
    summary: "Cấu trúc record sân bay: ICAO code, elevation, magnetic variation..."
    tokens: {l2: 740, l3: 3100}
  - id: "5.3"
    title: "Restrictive Airspace (UR/PR)"
    summary: "Record vùng trời hạn chế: designation, type, multiple code, level."
    tokens: {l2: 810, l3: 2900}
  - id: "5.3.2"
    title: "Airspace Type Codes"
    summary: "Bảng mã loại airspace: P, R, D, A, C, T, W..."
    tokens: {l2: 350, l3: 900}
```

AERO-KB Architecture Specification v0.1 — Draft. Soạn ngày 09/07/2026. Tài liệu nội bộ, không phân phối ra ngoài tổ chức.