TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

NGHIÊN CỨU VÀ ỨNG DỤNG

CHATBOT AI TRONG PHÁT TRIỂN WEBSITE MẠNG XÃ HỘI

VŨ THƯỢNG HẢI

Hà Nội – Năm 2026

TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

NGHIÊN CỨU VÀ ỨNG DỤNG

CHATBOT AI TRONG PHÁT TRIỂN WEBSITE MẠNG XÃ HỘI

Họ và tên sinh viên:	**Vũ Thượng Hải**
Mã sinh viên:	**22111060108**
Lớp:	**ĐH12C1**
Ngành đào tạo:	**Công Nghệ Thông Tin**
NGƯỜI HƯỚNG DẪN:	**ThS. TRƯƠNG MẠNH ĐẠT**

Hà Nội – Năm 2026

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập – Tự do – Hạnh phúc

BẢN CAM ĐOAN

Tên tôi là: **Vũ Thượng Hải**

Mã sinh viên: **22111060108** Lớp: **ĐH12C1**

Ngành: **Công Nghệ Thông Tin**

Tôi đã thực hiện khóa luận tốt nghiệp với đề tài: **Nghiên cứu và ứng dụng Chatbot AI trong phát triển website mạng xã hội**

Tôi xin cam đoan đây là đề tài nghiên cứu của riêng tôi và được sự hướng dẫn của **ThS. Trương Mạnh Đạt**. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa được công bố dưới bất kỳ hình thức nào. Nếu phát hiện có bất kỳ hình thức gian lận nào tôi xin hoàn toàn chịu trách nhiệm trước pháp luật.

Hà Nội, ngày ……tháng ……năm 2026

Giáo viên hướng dẫn	**Sinh viên**
(Ký và ghi rõ họ tên)	*(Ký và ghi rõ họ tên)*
ThS. Trương Mạnh Đạt	**Vũ Thượng Hải**

LỜI CẢM ƠN

Trong quá trình nghiên cứu đề tài "**Nghiên cứu và ứng dụng Chatbot AI trong phát triển website mạng xã hội**", em xin cảm ơn tới các thầy cô giảng viên khoa Công nghệ thông tin trường Đại học Tài nguyên và Môi trường Hà Nội đã luôn hướng dẫn em để có thể hoàn thành bài khóa luận một cách tốt nhất.

Đặc biệt, em xin gửi lời cảm ơn sâu sắc tới **ThS. Trương Mạnh Đạt**, người đã hướng dẫn trực tiếp cho em trong khóa luận lần này. Nhờ có sự hướng dẫn chi tiết, tận tình của thầy mà em đã khắc phục được nhiều sai sót trong khóa luận lần này.

Tuy nhiên với những hạn chế về kiến thức và thời gian nên không thể tránh khỏi sai sót, em rất mong nhận được những nhận xét góp ý chỉ bảo của thầy cô để chương trình được hoàn thiện hơn, có tính thực tiễn để có thể áp dụng trong tương lai.

Em xin chân thành cảm ơn!

TÓM TẮT

Trong bối cảnh trí tuệ nhân tạo (AI) đang tái định nghĩa quy trình phát triển phần mềm, khóa luận tập trung nghiên cứu và xây dựng hệ thống **Agent Skill Framework (Khung kỹ năng tác vụ)**—một bộ công cụ AI chuyên biệt hỗ trợ tự động hóa các giai đoạn phân tích và thiết kế hệ thống. Đề tài đề xuất kiến trúc *3 Pillars (Ba trụ cột cốt lõi)* bao gồm Knowledge (Tri thức), Process (Quy trình), Guardrails (Hàng rào bảo vệ), kết hợp với mô hình *Meta-Skill (Kỹ năng siêu cấp)* theo chu trình vận hành `Architect (Kiến trúc sư)` → `Planner (Người lập kế hoạch)` → `Builder (Người thực thi)`. Cách tiếp cận này giúp tối ưu hóa việc tổ chức tri thức và quy trình làm việc cho AI Agent (Tác vụ AI thông minh), từ đó giảm thiểu hiện tượng ảo giác (hallucination) và nâng cao tính nhất quán trong thiết kế phần mềm.

Hệ thống được kiểm chứng thông qua việc phát triển nền tảng **Steve Void**—một mạng xã hội chia sẻ tri thức dành cho cộng đồng lập trình viên. Ứng dụng được xây dựng trên các công nghệ hiện đại như *Next.js 15*, *Payload CMS 3.x* và *MongoDB Atlas*. Khóa luận đã tích hợp chuỗi 28 Agent Skills (Kỹ năng tác vụ) chuyên biệt để tự động hóa việc vẽ các sơ đồ UML (Ngôn ngữ mô hình hóa thống nhất) như Sequence (Tuần tự), Activity (Hoạt động), Flow (Luồng nghiệp vụ), Class (Lớp), đồng thời thiết kế lược đồ dữ liệu (Schema) và trích xuất các đặc tả giao diện (UI/UX).

Kết quả thực nghiệm cho thấy *Agent Skill Framework* giúp tăng năng suất phát triển phần mềm trung bình từ 40% đến 60% trong giai đoạn thiết kế, đồng thời đảm bảo tài liệu kỹ thuật tuân thủ nghiêm ngặt các tiêu chuẩn công nghiệp. Nghiên cứu không chỉ giải quyết bài toán tối ưu hóa quy trình làm việc cho doanh nghiệp mà còn cung cấp một mô hình đào tạo thực tiễn

cho sinh viên và người mới học lập trình trong kỷ nguyên trí tuệ nhân tạo.

Từ khóa: AI Agent (Tác vụ AI), Agent Skill Framework (Khung kỹ năng tác vụ), Next.js 15, Payload CMS, Steve Void, Software Engineering (Kỹ nghệ phần mềm).

MỤC LỤC

DANH SÁCH BẢNG

DANH SÁCH HÌNH VẼ

DANH MỤC KÝ HIỆU, CHỮ VIẾT TẮT

Ký hiệu / Viết tắt	Diễn giải chi tiết
UC	Use Case (Trường hợp sử dụng / Ca sử dụng)
US	User Story (Câu chuyện người dùng)
SRS	Software Requirement Specification (Đặc tả yêu cầu phần mềm)
AI	Artificial Intelligence (Trí tuệ nhân tạo)
LLM	Large Language Model (Mô hình ngôn ngữ lớn)
API	Application Programming Interface (Giao diện lập trình ứng dụng)
REST	Representational State Transfer (Kiến trúc truyền tải trạng thái đại diện)
CSDL	Cơ sở dữ liệu (Database)
CMS	Content Management System (Hệ quản trị nội dung)
UML	Unified Modeling Language (Ngôn ngữ mô hình hóa thống nhất)
ERD	Entity Relationship Diagram (Sơ đồ quan hệ thực thể)
UI/UX	User Interface / User Experience (Giao diện / Trải nghiệm người dùng)
CRUD	Create, Read, Update, Delete (Tạo, Đọc, Cập nhật, Xóa)
DTO	Data Transfer Object (Đối tượng chuyển đổi dữ liệu)
YAML	YAML Ain’t Markup Language (Định dạng dữ liệu dạng văn bản)

Ký hiệu / Viết tắt	Diễn giải chi tiết
JSON	JavaScript Object Notation (Định dạng trao đổi dữ liệu hướng đối tượng)
CLI	Command Line Interface (Giao diện dòng lệnh)
SSR	Server-side Rendering (Kết xuất phía máy chủ)
CSR	Client-side Rendering (Kết xuất phía trình duyệt)
SEO	Search Engine Optimization (Tối ưu hóa bộ máy tìm kiếm)
RSC	React Server Components (Thành phần phía máy chủ của React)
SSE	Server-Sent Events (Sự kiện được gửi từ phía máy chủ)
ASF	Agent Skill Framework (Khung kỹ năng tác vụ)
MSF	Meta-Skill Framework (Khung kỹ năng siêu cấp)
IP	Interaction Point (Điểm tương tác giữa người và máy)
Schema	Database Schema (Lược đồ cơ sở dữ liệu vật lý)
Wireframe	Wireframe Blueprint (Bản vẽ phác thảo cấu trúc giao diện)
Guardrail	Safety Guardrail (Hàng rào bảo vệ và kiểm soát chất lượng)

MỞ ĐẦU

Giới thiệu đơn vị thực tập

Công ty Cổ phần Innotech (INNOTECH J.S.C) là một trong những đơn vị hàng đầu trong lĩnh vực cung cấp các giải pháp công nghệ thông tin và chuyển đổi số toàn diện cho doanh nghiệp tại Việt Nam.

- **Tên đầy đủ:** Công ty Cổ phần Innotech.
- **Địa chỉ trụ sở:** NV05 đường Foresa 4, Khu đô thị sinh thái Xuân Phương, Phường Xuân Phương, Quận Nam Từ Liêm, Thành phố Hà Nội.
- **Website:** `https://innotechjsc.com/`

Với tôn chỉ hoạt động **"Single Portal For All"**, Innotech hướng tới mục tiêu trở thành đối tác công nghệ tin cậy, cung cấp giải pháp một cửa cho mọi nhu cầu công nghệ của doanh nghiệp. Sứ mệnh của công ty là cam kết mang lại trải nghiệm chuyển đổi số hiệu quả, an toàn và được tùy chỉnh tối ưu, phù hợp với đặc thù riêng biệt của từng khách hàng.

Các lĩnh vực hoạt động và dịch vụ chính của Innotech bao gồm:

- **Chuyển đổi số:** Tư vấn và triển khai chiến lược số hóa quy trình doanh nghiệp.
- **Hệ thống ERP:** Xây dựng hệ thống quản trị nguồn lực doanh nghiệp tích hợp.
- **An ninh mạng:** Giải pháp bảo mật, tường lửa và bảo vệ dữ liệu tiên tiến.

- **AI cho doanh nghiệp:** Ứng dụng trí tuệ nhân tạo vào tự động hóa và phân tích dữ liệu.
- **Digital Marketing:** Hỗ trợ doanh nghiệp phát triển trên các nền tảng số.

Trong hơn 10 năm hoạt động, Innotech đã triển khai thành công hơn 100 dự án cho các doanh nghiệp và tập đoàn lớn như Kiến Tạo Xanh Group, ITM Group, OVAN Group, HANOTEX,... Với đội ngũ hơn 50 chuyên gia giàu kinh nghiệm, công ty luôn đảm bảo các giá trị cốt lõi về sự chuyên nghiệp, tính linh hoạt, bảo mật và hiệu quả kinh tế cho đối tác.

Lý do chọn đề tài

Trong bối cấp công nghệ số và trí tuệ nhân tạo phát triển mạnh mẽ như hiện nay, với sự ra đời của các công cụ AI thế hệ mới như *ChatGPT*, *GitHub Copilot* hay mẫu mô hình ngôn ngữ lớn mạnh mẽ như *Claude*, ngành công nghệ thông tin đang trải qua một cuộc cách mạng về cách tiếp cận phát triển phần mềm. Theo báo cáo của *GitHub* năm 2024, hơn 92% các nhà phát triển chuyên nghiệp đã sử dụng các công cụ hỗ trợ AI trong công việc, và năng suất lập trình tăng trung bình 55% khi có sự đồng hành của AI. Tuy nhiên, khoảng cách về kỹ năng giữa các cấp độ lập trình viên — từ sinh viên mới bắt đầu đến các chuyên gia dày dạn kinh nghiệm — vẫn là một thách thức lớn trong giáo dục và đào tạo công nghệ thông tin.

Đối với sinh viên và người mới học lập trình, việc tiếp cận một dự án thực tế thường gặp phải nhiều rào cản: thiếu kiến thức về kiến trúc hệ thống, chưa nắm vững quy trình phát triển phần mềm chuẩn, và đặc biệt là khoảng cách giữa lý thuyết học tại trường với thực tiễn triển khai ứng dụng. Theo khảo sát từ *Stack Overflow Developer Survey 2024*, trung bình một lập trình viên mới cần từ 2-3 năm kinh nghiệm thực tế để có thể tự tin triển khai một hệ

thống hoàn chỉnh. Khoảng cách này không chỉ làm chậm quá trình phát triển nghề nghiệp của sinh viên mà còn tạo ra sự chênh lệch lớn về năng lực giữa các nhóm đối tượng trong ngành.

Tuy nhiên, sự xuất hiện của các công cụ AI đã mở ra một hướng đi mới. AI không chỉ hỗ trợ viết mã nguồn mà còn có khả năng đóng vai trò như một *chuyên gia tư vấn ảo*, giúp sinh viên và lập trình viên thiếu kinh nghiệm tiếp cận các quy trình, mô hình mẫu (patterns), và các phương pháp thực hành tối ưu (best practices) một cách có hệ thống. Nghiên cứu gần đây của MIT và Harvard cho thấy sinh viên sử dụng trợ lý ảo (AI assistant) có tốc độ học tập nhanh hơn 40% và mức độ tự tin khi giải quyết vấn đề phức tạp tăng 65% so với nhóm không sử dụng AI.

Xuất phát từ nhận định này, đề tài **"Nghiên cứu và ứng dụng Chatbot AI trong phát triển website mạng xã hội"** được lựa chọn với mục tiêu xây dựng một Khung kỹ năng tác vụ (*Agent Skill Framework*) — bộ công cụ AI chuyên biệt hóa cho từng giai đoạn của quy trình phần mềm (phân tích, thiết kế, triển khai). Thay vì chỉ cung cấp các dòng mã nguồn đơn lẻ, hệ thống này sẽ hướng dẫn người dùng theo các quy trình chuẩn mực: từ phân tích luồng nghiệp vụ (*Business Process Flow*), thiết kế kiến trúc hệ thống (Sơ đồ tuần tự, Sơ đồ lớp), đến thực thi mã nguồn tuân thủ các thực hành tốt nhất (*best practices*). Cách tiếp cận này giúp rút ngắn đáng kể khoảng cách về năng lực giữa người mới học và lập trình viên chuyên nghiệp, đồng thời xây dựng một "Nhà máy tri thức" (*Knowledge Factory*) có khả năng lưu trữ và tái sử dụng cho các dự án trong tương lai.

Tính cấp thiết của đề tài còn được thể hiện rõ nét qua nhu cầu thực tế tại Công ty Cổ phần *Innotech* — đơn vị tiếp nhận thực tập — khi đơn vị này đang trong quá trình mở rộng các dự án chuyển đổi số và cần xây dựng các quy trình làm việc được tối ưu hóa bởi AI nhằm nâng cao năng suất và bảo đảm chất lượng sản phẩm. Hơn nữa, việc nghiên cứu và triển khai hệ thống này trên nền tảng mạng xã hội chia sẻ tri thức (**Steve Void**) còn mang lại giá

trị cộng đồng, giúp sinh viên và cộng đồng lập trình viên Việt Nam tiếp cận tri thức một cách có hệ thống và hiệu quả hơn.

Mục tiêu nghiên cứu

Đề tài hướng đến các mục tiêu nghiên cứu cụ thể sau:

Mục tiêu tổng quát: Nghiên cứu và xây dựng Khung kỹ năng tác vụ (*Agent Skill Framework*) — bộ công cụ AI chuyên biệt hỗ trợ toàn diện quy trình phát triển phần mềm. Hệ thống được tích hợp vào nền tảng mạng xã hội chia sẻ tri thức *Steve Void* nhằm rút ngắn khoảng cách kỹ năng giữa các nhóm lập trình viên và tối ưu hóa quy trình xây dựng sản phẩm công nghệ.

Các mục tiêu cụ thể:

1. **Nghiên cứu và thiết kế kiến trúc Khung kỹ năng tác vụ (*Agent Skill Framework*):**
    - Phân tích các mô hình tác vụ AI (*AI Agent*) hiện đại như: Mô hình Phản hồi-Hành động (*ReAct*), Sử dụng công cụ (*Tool-Use*), và Hệ thống đa tác vụ (*Multi-Agent Systems*).
    - Thiết kế kiến trúc Ba trụ cột (*3 Pillars*) và Bảy vùng chức năng (*7 Zones*) để tổ chức tri thức.
    - Xây dựng Khung kỹ năng siêu cấp (*Meta-Skill Framework*) phục vụ việc khởi tạo và quản trị các kỹ năng tác vụ một cách tự động.
2. **Triển khai bộ kỹ năng tác vụ cho giai đoạn Thiết kế và Đặc tả (*Life-2 — Analysis & Design*):**
    - *Skill Architect*: Thiết kế kiến trúc cho các kỹ năng mới.
    - *Flow Design Analyst*: Phân tích và vẽ sơ đồ luồng nghiệp vụ (*Flow Diagram*).

- *Sequence Design Analyst*: Thiết kế sơ đồ tuần tự chuẩn UML (*Sequence Diagram*).
- *Activity Diagram Analyst*: Vẽ sơ đồ hoạt động theo kiến trúc sạch (*Clean Architecture*).
- *Class Diagram Analyst*: Phân tích cấu trúc dữ liệu cho *MongoDB/Payload CMS*.
- *UI Architecture Analyst*: Trích xuất các đặc tả giao diện người dùng (*UI Specs*) từ lược đồ dữ liệu và các sơ đồ nghiệp vụ.

3. **Xây dựng nền tảng mạng xã hội *Steve Void*:**

- Phát triển website với 6 phân hệ cốt lõi: Xác thực (*Authentication*), Công cụ nội dung (*Content Engine*), Bảng tin khám phá (*Discovery Feed*), Tương tác (*Engagement*), Lưu trữ (*Bookmarking*), Thông báo và Kiểm duyệt (*Notifications & Moderation*).
- Tích hợp các kỹ năng tác vụ AI (*Agent Skills*) vào quy trình phát triển thực tế của dự án.
- Xây dựng "Nhà máy tri thức" (*Knowledge Factory*) giúp lưu trữ và quản lý tri thức kỹ thuật có khả năng tái sử dụng cao.

4. **Đánh giá hiệu quả và khả năng mở rộng:**

- So sánh năng suất phát triển khi có và không có sự hỗ trợ từ các kỹ năng tác vụ (*Agent Skills*).
- Đánh giá chất lượng tài liệu thiết kế được trích xuất tự động (Sơ đồ, Đặc tả kỹ thuật).
- Khảo sát mức độ hài lòng của người dùng khi trải nghiệm hệ thống hỗ trợ AI.

Đối tượng và phạm vi nghiên cứu

Phạm vi nghiên cứu

Đề tài tập trung vào các phạm vi nghiên cứu cụ thể sau:

Về mặt công nghệ:

- *AI Agent Framework*: Nghiên cứu kiến trúc kỹ năng tác vụ dựa trên giao diện lập trình của mô hình *Claude 3.5 Sonnet* từ *Anthropic*, kết hợp với công cụ dòng lệnh *Claude Code CLI*.
- *Backend*: Sử dụng hệ quản trị nội dung *Payload CMS 3.x* kết hợp cơ sở dữ liệu *MongoDB Atlas*, tập trung khai thách giao diện lập trình nội bộ (*Local API*) và mô hình điểm can thiệp dữ liệu (*Hooks Pattern*).
- *Frontend*: Sử dụng nền tảng *Next.js 15* (cơ chế *App Router*, các thành phần phía máy chủ — *RSC*) cùng với *Tailwind CSS v4* và thư viện giao diện *Radix UI*.
- *Truyền tải thời gian thực*: Sử dụng cơ chế sự kiện gửi từ máy chủ (*Server-Sent Events — SSE*) cho hệ thống thông báo.

Về mặt chức năng:

- Giai đoạn **Thiết kế và Đặc tả (*Life-2 — Analysis & Design*)** của quy trình phát triển phần mềm, bao gồm:
  - Phân tích luồng nghiệp vụ (*Business Process Flow*)
  - Thiết kế tương tác tuần tự (*Sequence Diagram*) và sơ đồ hoạt động (*Activity Diagram*)
  - Thiết kế lược đồ dữ liệu (*Class Diagram*, *ER Diagram*)
  - Thiết kế các đặc tả giao diện (*UI Specs*) và bản vẽ phác thảo (*Wireframes*)

- Triển khai 6 phân hệ cốt lõi của *Steve Void* từ M1 đến M6.

Giới hạn phạm vi:

- Đề tài **không** nghiên cứu việc huấn luyện các mô hình trí tuệ nhân tạo mới mà tập trung vào khai thác năng lực giải quyết vấn đề của các mô hình có sẵn.
- Hệ thống được triển khai ở quy mô thử nghiệm (*staging*), chưa tối ưu hoàn toàn cho môi trường vận hành thực tế (*production*).
- Tập trung chuyên sâu vào giai đoạn thiết kế (*Life-2*); các giai đoạn triển khai và kiểm định được thực hiện ở mức độ hoàn thiện khung cơ bản.

Đối tượng nghiên cứu

1. **Khung kỹ năng tác vụ (*Agent Skill Framework*):**
    - Kiến trúc *3 Pillars*: Kho tri thức (*Knowledge*), Quy trình làm việc (*Process*), và Hàng rào bảo vệ (*Guardrails*).
    - Cấu trúc *7 Zones*: Từ định nghĩa vai trò cốt lõi (*Core — SKILL.md*) đến các tri thức miền và tài sản đầu ra.
    - Khung kỹ năng siêu cấp (*Meta-Skill Framework*): Quy trình khởi tạo kỹ năng với vòng lặp Tự kiểm tra - Sửa lỗi (*Verify-Fix-Loop*).
2. **Bộ Agent Skills chuyên biệt cho Life-2:**
    - *skill-architect:* Thiết kế kiến trúc kỹ năng mới.
    - *flow-design-analyst:* Phân tích luồng nghiệp vụ đa làn (*Swimlane 3-lane*).
    - *sequence-design-analyst:* Thiết kế Sequence Diagram chuẩn UML

- *activity-diagram-design-analyst:* Vẽ Activity Diagram theo Clean Architecture
- *class-diagram-analyst:* Phân tích cấu trúc dữ liệu theo định dạng kép (*Mermaid + YAML Contract*).
- *ui-architecture-analyst:* Trích xuất UI Screen Specs từ Schema và Diagrams

3. **Nền tảng mạng xã hội Steve Void:**
    - Thiết kế lược đồ dữ liệu (*Database Schema Design*) với các khái niệm *Aggregate Roots*, *Embedded Documents*.
    - Đặc tả kỹ thuật API (*API Specification*) dựa trên các mẫu thiết kế của Payload.
    - Hệ thống thiết kế Tân tối giản (*Neobrutalism*) với tông màu hồng (*Pink primary*) chủ đạo.
    - 6 phân hệ chức năng từ M1 đến M6.
4. **Quy trình làm việc tích hợp AI:**
    - Quy trình *OpenSpec Workflow*: Quản lý thay đổi dựa trên các thành phẩm kỹ thuật (*artifacts*).
    - Nhà máy tri thức (*Knowledge Factory*): Cơ chế tái sử dụng tri thức từ dữ liệu cũ.
    - Các điểm tương tác (*Interaction Points*): Các điểm dừng kiểm soát giữa người và máy.

Phương pháp nghiên cứu

Đề tài áp dụng các phương pháp nghiên cứu kết hợp giữa lý thuyết và thực nghiệm như sau:

1. **Phương pháp nghiên cứu tài liệu (*Literature Review*):**

    - Nghiên cứu các mô hình tác vụ AI (*AI Agent*) hiện đại: Mô hình Phản hồi-Hành động (*ReAct*), Các tác vụ sử dụng công cụ (*Tool-Use Agents*), và Hệ thống đa tác vụ (*Multi-Agent Systems*).
    - Phân tích các khung quy trình phát triển phần mềm: Phát triển linh hoạt (*Agile*), Kiến trúc sạch (*Clean Architecture*), Phát triển hướng tên miền (*Domain-Driven Design*).
    - Tham khảo tài liệu kỹ thuật chuyên ngành: Đặc tả giao diện lập trình *Claude API* của *Anthropic*, tài liệu *Payload CMS*, tài liệu *Next.js 15*, và các quy tắc tối ưu hóa *MongoDB*.
    - Nghiên cứu các công trình liên quan: *GitHub Copilot Workspace*, *Cursor AI*, và công cụ *v0.dev* của *Vercel*.

2. **Phương pháp phân tích và thiết kế hệ thống:**

    - *Phân tích yêu cầu:* Sử dụng câu chuyện người dùng (*User Stories*) và sơ đồ trường hợp sử dụng (*Use Case Diagram*) để xác định chức năng hệ thống.
    - *Thiết kế kiến trúc:* Áp dụng kiến trúc sạch (*Clean Architecture*) với ba lớp phân tách (*Business Logic, Use Case, External*).
    - *Thiết kế cơ sở dữ liệu:* Sử dụng sơ đồ quan hệ thực thể (*ER Diagram*) và các mẫu thiết kế lược đồ *MongoDB* (Gốc tập hợp — *Aggregate Root* so với tài liệu nhúng — *Embedded Document*).
    - *Thiết kế giao diện:* Vẽ bản vẽ phác thảo với công cụ *Pencil*, áp dụng hệ thống thiết kế Tân tối giản (*Neobrutalism*).
    - *Mô hình hóa với UML:* Sử dụng sơ đồ tuần tự, sơ đồ hoạt động, sơ đồ lớp dưới định dạng mã nguồn *Mermaid*.

3. **Phương pháp thực nghiệm và triển khai:**

- *Phát triển tiệm tiến (Iterative Development):* Phát triển theo từng phân hệ (từ M1 đến M6), mỗi phân hệ trải qua bốn giai đoạn vòng đời (Tầm nhìn — *Vision*, Thiết kế — *Design*, Triển khai — *Implementation*, Kiểm chứng — *Verification*).
- *Tạo nguyên mẫu nhanh (Rapid Prototyping):* Xây dựng các kỹ năng tác vụ (*Agent Skills*) theo chu trình: Kiến trúc sư → Người lập kế hoạch → Người thực thi với vòng phản hồi điều chỉnh liên tục (*feedback loop*).
- *Tiếp cận hướng kiểm thử (Test-Driven Approach):* Mỗi kỹ năng có danh sách kiểm soát chất lượng (*Quality Control Checklist*) và cơ chế tự chấm điểm (*Self-Scoring*).
- *Quản lý phiên bản dựa trên Git:* Sử dụng quy trình *OpenSpec Workflow* để quản lý các thay đổi dựa trên đặc tả kỹ thuật.

4. **Phương pháp đánh giá và kiểm thử:**

   - *Kiểm duyệt mã nguồn (Code Review):* Sử dụng tác vụ kiểm duyệt đặc tả (*spec-reviewer agent*) để đối chiếu mã nguồn thực tế với các tài liệu đặc tả.
   - *Chỉ số chất lượng (Quality Metrics):* Đo lường thời gian phát triển, số lượt lặp lại (*iteration*) cần thiết và tỷ lệ lỗi được phát hiện.
   - *Kiểm thử chấp nhận người dùng (User Acceptance Testing):* Thu thập phản hồi từ sinh viên và lập trình viên về trải nghiệm sử dụng các kỹ năng tác vụ.
   - *Đánh giá tài liệu:* Đánh giá chất lượng các sơ đồ và đặc tả kỹ thuật thông qua bộ quy tắc chấm điểm (*rubric*) chuẩn mực.

5. **Phương pháp thu thập và phân tích dữ liệu:**

- *Session Logs:* Ghi nhận toàn bộ transcript của Agent Sessions (`~/.claude/projects/`)
- *Auto Memory:* Lưu trữ lessons learned và patterns phát hiện trong quá trình phát triển
- *Metrics Dashboard:* Theo dõi số lượng Skills được tạo, thời gian trung bình mỗi artifact, tỷ lệ thành công
- *Survey & Interview:* Khảo sát định lượng (Likert scale) và phỏng vấn định tính với người dùng

Các phương pháp trên được áp dụng tuần tự và song song, tạo thành một quy trình nghiên cứu khoa học chặt chẽ từ lý thuyết đến thực tiễn, từ thiết kế đến triển khai và đánh giá.

Dự kiến kết quả nghiên cứu đạt được

Sau khi hoàn thành nghiên cứu, đề tài dự kiến đạt được các kết quả cụ thể sau:

Sản phẩm phần mềm

1. **Hệ thống Agent Skill Framework hoàn chỉnh:**
    - 28 Agent Skills chuyên biệt cho các giai đoạn phát triển phần mềm (Life-1 đến Life-4)
    - Meta-Skill Framework: Master Skill orchestrator với skill-architect, skill-planner, skill-builder
    - OpenSpec Workflow System: quản lý change với 8 skills (new, continue, apply, verify, sync, archive, explore, ff)
    - Skill Repository tại `.claude/skills/` và `.agent/skills/`

2. **Nền tảng mạng xã hội *Steve Void* (Sản phẩm khả thi tối giản — *MVP*):**

    - *Backend*: Hệ quản trị nội dung *Payload CMS 3.x* với hơn 15 bộ sưu tập dữ liệu (*collections*) như Người dùng, Bài viết, Bình luận, Kết nối, Lưu trữ, Thông báo, Báo cáo độc hại.
    - *Frontend*: Ứng dụng *Next.js 15* với hơn 30 màn hình chức năng (Bảng tin, Trang cá nhân, Chi tiết bài viết, Tìm kiếm, Bộ sưu tập).
    - Cơ sở dữ liệu: Lược đồ *MongoDB Atlas* với dữ liệu thử nghiệm trong môi trường phát triển (*staging*).
    - Hệ thống thông báo thời gian thực dựa trên cơ chế *SSE*.
    - Cơ chế xác thực: Sử dụng mã định danh *JWT*, xác thực qua Email và khôi phục mật khẩu.

3. **Hệ thống thiết kế và thư viện thành phần (*Design System*):**

    - Bộ công cụ giao diện Tân tối giản (*Neobrutalism UI Kit*): Hơn 50 thành phần (Nút bấm, Thẻ nội dung, Biểu mẫu, Cửa sổ phụ) sử dụng *Tailwind v4* và *Radix UI*.
    - Giao diện hiển thị tốt trên máy tính để bàn, máy tính bảng và thiết bị di động.
    - Phối màu chủ đạo là màu hồng (*pink primary*) với độ tương phản cao, đường viền đậm (*bold borders*) và đổ bóng lệch (*offset shadows*).

Tài liệu nghiên cứu và thiết kế

1. **Tài liệu Life-1 (Vision & Research):**

    - Product Vision Document

- User Personas (5 personas chính)
- User Stories (100+ stories cho 6 modules)
- Technical Decisions Document

2. **Tài liệu Thiết kế và Đặc tả (*Life-2 — Analysis & Design*):**
    - Thiết kế lược đồ cơ sở dữ liệu (*Database Schema Design*) với tài liệu chi tiết ở cấp độ trường dữ liệu (*field-level*).
    - Đặc tả giao diện lập trình kỹ thuật với hơn 60 điểm cuối (*endpoints*) kèm lược đồ yêu cầu và phản hồi.
    - Hệ thống sơ đồ UML: Hơn 20 sơ đồ tuần tự, 15 sơ đồ hoạt động, 6 sơ đồ luồng và sơ đồ lớp cho toàn bộ các phân hệ (*modules*).
    - Bản vẽ phác thảo giao diện: Hơn 30 màn hình với các nguyên mẫu độ trung thực cao.
    - Các tệp tài liệu đặc tả chi tiết cho 6 phân hệ (m1-m6) với tổng dung lượng lớn.
3. **Tài liệu kỹ thuật:**
    - Architecture Documentation (Clean Architecture, folder structure)
    - Agent Skill Documentation (SKILL.md cho mỗi skill)
    - Development Guidelines (coding standards, best practices)
    - Deployment Guide (Vercel, MongoDB Atlas setup)

Kiến thức và đóng góp học thuật

1. **Mô hình Agent Skill Framework:**
    - Đề xuất kiến trúc 3 Pillars + 7 Zones cho việc tổ chức tri thức AI Agent

- Phương pháp Meta-Skill cho việc tự động hóa quá trình tạo Skills mới
- Cơ chế Progressive Disclosure (Tier 1 mandatory, Tier 2 conditional loading)
- Source Citation mechanism để chống hallucination

2. **Quy trình phát triển phần mềm tích hợp AI:**
    - 4-Life Lifecycle Model (Vision, Design, Implementation, Verification)
    - OpenSpec Workflow cho change management
    - Knowledge Factory pattern cho knowledge reuse
    - Interaction Points (IP-1, IP-2, IP-3) để cân bằng automation và human oversight
3. **Đánh giá hiệu quả:**
    - So sánh năng suất phát triển: thời gian tạo specs, diagrams, code giảm 40-60%
    - Chất lượng tài liệu thiết kế: đạt 85%+ compliance với standards
    - Mức độ hài lòng người dùng: khảo sát từ 20+ sinh viên/lập trình viên

Sản phẩm bổ trợ

- Source code hoàn chỉnh trên GitHub repository (MIT License)
- Video demo và hướng dẫn sử dụng (YouTube playlist)
- Bài báo khoa học về Agent Skill Framework (nếu có cơ hội)
- Workshop materials cho sinh viên về cách sử dụng AI trong phát triển phần mềm

Ý nghĩa khoa học và ý nghĩa thực tiễn của nghiên cứu

Ý nghĩa khoa học

1. **Đóng góp vào lĩnh vực AI Agent Research:**
    - Đề xuất mô hình *Agent Skill Framework* với kiến trúc 3 Pillars (Knowledge, Process, Guardrails) và 7 Zones, cung cấp một cách tiếp cận có hệ thống để tổ chức tri thức cho AI Agents
    - Phát triển *Meta-Skill Framework*—mô hình tự động hóa việc tạo Skills mới thông qua chu trình Architect → Planner → Builder, góp phần vào nghiên cứu về *self-improving AI systems*
    - Đề xuất cơ chế *Source Citation* và *Progressive Disclosure* để giảm thiểu hallucination và tối ưu hóa context window trong Large Language Models
2. **Ứng dụng AI vào Software Engineering Process:**
    - Nghiên cứu cách tích hợp AI vào từng giai đoạn của Software Development Life Cycle (SDLC), đặc biệt là giai đoạn Analysis & Design thường bị bỏ qua trong các công cụ AI code assistant hiện tại
    - Chứng minh tính khả thi của việc sử dụng AI không chỉ cho code generation mà còn cho business analysis, system design, và documentation
    - Đề xuất mô hình *4-Life Lifecycle* (Vision, Design, Implementation, Verification) với các Interaction Points để cân bằng giữa automation và human oversight
3. **Phương pháp luận mới cho phát triển phần mềm hỗ trợ bởi AI:**

- Xây dựng quy trình quản lý thay đổi dựa trên đặc tả (*OpenSpec Workflow*) — quản lý các thay đổi dựa trên thành phẩm đặc tả thay vì chỉ dựa vào các lượt nộp mã nguồn (*code commits*).
- Đề xuất khái niệm Nhà máy tri thức (*Knowledge Factory*) — cơ chế tái sử dụng tài nguyên tri thức từ các lần thực thi trước đó, tương tự như cơ chế lưu trữ đệm (*caching*) nhưng ở cấp độ ngữ nghĩa (*semantic level*).
- Nghiên cứu cách kết hợp quy trình làm việc dựa trên tác vụ (*Agentic Workflow*) với các thực hành tốt nhất trong kỹ nghệ phần mềm.

Ý nghĩa thực tiễn

1. **Đối với sinh viên và người mới học lập trình:**
    - Rút ngắn thời gian từ ”không biết gì”đến ”có thể làm dự án thực tế”từ 2-3 năm xuống còn 6-12 tháng thông qua hướng dẫn có hệ thống của Agent Skills
    - Cung cấp một ”chuyên gia ảo”24/7 giúp sinh viên học theo đúng quy trình chuẩn mực, tránh các bad practices phổ biến
    - Tạo ra các tài liệu thiết kế chất lượng cao (diagrams, specs) mà sinh viên có thể tham khảo và học hỏi
    - Giúp sinh viên hiểu rõ ”tại sao”(why) chứ không chỉ ”làm thế nào”(how) thông qua explanations trong Skills
2. **Đối với doanh nghiệp và đội ngũ phát triển:**
    - Tăng năng suất phát triển 40-60% thông qua automation các tác vụ lặp đi lặp lại (vẽ diagrams, viết specs, generate boilerplate code)

- Chuẩn hóa quy trình làm việc: mọi developer trong team sử dụng chung một bộ Skills, đảm bảo consistency
- Giảm technical debt: Agent Skills luôn đề xuất best practices và clean architecture patterns
- Onboarding nhanh hơn cho nhân viên mới: thay vì đọc hàng trăm trang documentation, họ có thể hỏi Agent và nhận câu trả lời context-aware

3. **Đối với cộng đồng lập trình viên Việt Nam:**

- *Steve Void* trở thành nền tảng chia sẻ tri thức chất lượng cao, có tính tổ chức chặt chẽ.
- Tạo ra nguồn tài nguyên tri thức tiếng Việt phong phú về lĩnh vực phát triển phần mềm với sự hỗ trợ của AI.
- Khuyến khích văn hóa lập tài liệu và chia sẻ tri thức thông qua các cơ chế trò chơi hóa (*gamification*) như hệ thống điểm danh tiếng và huy hiệu.
- Kết nối sinh viên với các bài toán thực tế của doanh nghiệp qua các dự án mẫu.

4. **Đối với Công ty Innotech:**

- Tối ưu hóa quy trình phát triển sản phẩm, đặc biệt trong các dự án chuyển đổi số
- Xây dựng competitive advantage thông qua việc áp dụng AI sớm vào workflow
- Giảm chi phí đào tạo nhân viên mới thông qua hệ thống Agent Skills
- Tạo ra một framework có thể tái sử dụng cho các dự án khác của công ty

5. **Đối với ngành Công nghệ thông tin Việt Nam:**

    - Đóng góp vào xu hướng AI-first Development đang diễn ra toàn cầu
    - Nâng cao chất lượng đào tạo CNTT tại Việt Nam thông qua việc tích hợp công cụ AI vào giảng dạy
    - Tạo tiền đề cho các nghiên cứu tiếp theo về AI in Education và AI-assisted Software Engineering
    - Chứng minh rằng sinh viên Việt Nam có thể nghiên cứu và triển khai các công nghệ tiên tiến ngang tầm quốc tế

Tóm lại, đề tài không chỉ có giá trị về mặt học thuật (đóng góp vào nghiên cứu AI Agent) mà còn có tác động thực tiễn sâu rộng, từ cá nhân sinh viên, doanh nghiệp, đến toàn ngành CNTT Việt Nam.

Bố cục của báo cáo

Báo cáo khóa luận được tổ chức thành các phần chính như sau:

Phần mở đầu giới thiệu bối cảnh nghiên cứu, lý do chọn đề tài, mục tiêu, đối tượng, phạm vi, phương pháp nghiên cứu, và dự kiến kết quả đạt được.

Chương 1: Tổng quan về đề tài trình bày cơ sở lý thuyết về trí tuệ nhân tạo, AI Chatbot, quy trình phát triển phần mềm, các công nghệ sử dụng (Next.js, Payload CMS, MongoDB), và khảo sát các hệ thống tương tự hiện có trên thế giới.

Chương 2: Phân tích và thiết kế hệ thống trình bày chi tiết về Agent Skill Framework, bao gồm:

- Kiến trúc 3 Pillars (Knowledge, Process, Guardrails) và 7 Zones (Core, Knowledge, Scripts, Templates, Data, Loop, Assets)

- Meta-Skill Framework với chu trình Architect → Planner → Builder
- Bộ Agent Skills chuyên biệt cho Life-2: Flow Design Analyst, Sequence Design Analyst, Activity Diagram Analyst, Class Diagram Analyst, UI Architecture Analyst
- Thiết kế nền tảng Steve Void: Database Schema, API Specification, UI Wireframes cho 6 modules (M1-M6)
- Các diagrams: ER Diagram, Use Case Diagram, Sequence Diagrams, Activity Diagrams, Class Diagrams

Chương 3: Triển khai hệ thống mô tả quá trình cài đặt và phát triển thực tế:

- Môi trường phát triển và công cụ sử dụng (Claude Code CLI, VS Code, Git)
- Triển khai Agent Skills: cấu trúc thư mục, SKILL.md, knowledge base, scripts, templates

Chương 4: Kiểm thử và đánh giá trình bày kết quả thực nghiệm:

- Kiểm thử chức năng: unit tests, integration tests, end-to-end tests
- Đánh giá hiệu năng: response time, throughput, resource usage
- Đánh giá chất lượng Agent Skills: accuracy của diagrams/specs, compliance với standards
- So sánh năng suất phát triển: có vs không có Agent Skills
- Khảo sát người dùng: mức độ hài lòng, ease of use, learning curve
- Phân tích hạn chế và đề xuất cải tiến

Phần kết luận tóm tắt những kết quả đạt được, đánh giá mức độ hoàn thành mục tiêu, rút ra bài học kinh nghiệm, và đề xuất hướng phát triển tiếp theo cho dự án.

Tài liệu tham khảo liệt kê các nguồn tài liệu, sách, bài báo, trang web được trích dẫn trong quá trình nghiên cứu.

Phụ lục bao gồm các tài liệu bổ trợ như:

- Source code quan trọng (Agent Skills, Payload Collections, React Components)
- Database Schema chi tiết
- API Documentation đầy đủ
- User Manual cho Steve Void
- Screenshots và video demo
- Bảng khảo sát và kết quả phân tích

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

Chương này trình bày cơ sở lý thuyết nền tảng cho toàn bộ nghiên cứu, bao gồm: tổng quan về trí tuệ nhân tạo và mô hình ngôn ngữ lớn, quy trình làm việc dựa trên tác vụ (*Agentic Workflow*), các công nghệ sử dụng trong phát triển website, phương pháp mô hình hóa UML, và phân tích bài toán mạng xã hội chia sẻ kiến thức.

1.1 Tổng quan về trí tuệ nhân tạo và mô hình ngôn ngữ lớn

1.1.1 Khái niệm trí tuệ nhân tạo

Trí tuệ nhân tạo (*Artificial Intelligence* — AI) là lĩnh vực khoa học máy tính nghiên cứu và phát triển các hệ thống có khả năng thực hiện các tác vụ đòi hỏi trí thông minh của con người, bao gồm: nhận dạng ngôn ngữ tự nhiên, lập kế hoạch, suy luận logic và ra quyết định [7]. Trong thập kỷ gần đây, AI đã trải qua bước tiến vượt bậc nhờ sự phát triển của học sâu (*Deep Learning*) và đặc biệt là kiến trúc chuyển đổi (*Transformer*) [8], tạo nền tảng cho các mô hình ngôn ngữ lớn (*Large Language Models* — LLMs).

1.1.2 Mô hình ngôn ngữ lớn (LLMs)

Mô hình ngôn ngữ lớn là các mạng nơ-ron được huấn luyện trên lượng dữ liệu văn bản khổng lồ, có khả năng hiểu và sinh ngôn ngữ tự nhiên ở mức độ gần với con người. Các đặc điểm chính của LLMs bao gồm:

- **Khả năng hiểu ngữ cảnh (*Context-awareness*):** LLMs có thể xử lý hàng chục nghìn đơn vị từ ngữ (*tokens*) trong một lượt, cho phép hiểu được ngữ cảnh phức tạp của một dự án phần mềm.

- **Khả năng sinh nội dung (*Generation*):** Từ các câu lệnh dẫn dắt (*prompt*) đầu vào, LLMs có thể sinh ra mã nguồn, tài liệu thết kế, sơ đồ UML và các thành phẩm kỹ thuật (*artifacts*) khác.

- **Khả năng suy luận (*Reasoning*):** Các mô hình thế hệ mới (GPT-4, Claude, Gemini) có khả năng suy luận logic đa bước, phù hợp cho các tác vụ phân tích và thiết kế hệ thống.

- **Khả năng sử dụng công cụ (*Tool Use*):** LLMs có thể gọi các hàm, giao diện lập trình ứng dụng (*API*) và công cụ bên ngoài để thực thi hành động thực tế, không chỉ dừng ở mức tạo văn bản.

Trong nghiên cứu này, mô hình **Claude (Anthropic)** được sử dụng làm nền tảng cho toàn bộ hệ thống kỹ năng tác vụ (*Agent Skills*), nhờ khả năng xử lý cửa sổ ngữ cảnh (*context window*) lớn (hơn 200.000 tokens) và hỗ trợ sử dụng công cụ nguyên bản [6].

1.1.3 AI Chatbot trong phát triển phần mềm

AI Chatbot trong dự án này không chỉ là giao diện tương tác văn bản, mà đóng vai trò là một **trợ lý lập trình trí tuệ nhân tạo** (*AI Coding Assistant*). Thông qua công cụ *Claude Code CLI* [9], AI có khả năng:

- Đọc và phân tích cấu trúc dự án (hệ thống tệp tin, lịch sử phiên bản *Git*).

- Thực thi các câu lệnh dòng lệnh (*Terminal*) và khởi chạy các kịch bản tự động (*Scripts*).

- Đọc và ghi tệp tin trực tiếp trên hệ thống máy tính.

- Tương tác với các máy chủ cung cấp ngữ cảnh (*MCP servers - Model Context Protocol*) để mở rộng các năng lực ngoại vi.

Sự khác biệt giữa *AI Chatbot* thông thường và *AI Coding Assistant* nằm ở **khả năng chủ động thực thi** (*agency*) — không chỉ dừng lại ở mức độ tư vấn mà còn trực tiếp tham gia vào các tác vụ kỹ thuật trong môi trường phát triển thực tế.

1.2 Quy trình làm việc dựa trên Agent (Agentic Workflow)

1.2.1 Định nghĩa Agentic Workflow

Theo Andrew Ng [1], *Agentic Workflow* là một quy trình mà tác vụ AI hoạt động lặp đi lặp lại qua chu kỳ: **Suy nghĩ (*Thought*)** → **Thực thi (*Action*)** → **Quan sát (*Observation*)**. Khác với cách tiếp cận "truy vấn một lần"(*zero-shot*), *Agentic Workflow* cho phép AI tự điều chỉnh và cải thiện kết quả đầu ra qua nhiều vòng lặp.

Bốn mô hình thiết kế (*design pattern*) cốt lõi của quy trình tác vụ AI bao gồm [1]:

1. **Sự tự phản chiếu (*Reflection*):** AI tự đánh giá kết quả của chính mình, phát hiện lỗi và tự thực hiện sửa đổi.

2. **Sử dụng công cụ (*Tool Use*):** AI sử dụng các công cụ ngoại vi (tìm kiếm, thực thi mã, truy xuất tệp tin) để thu thập thông tin và thực hiện hành động.

3. **Lập kế hoạch (*Planning*):** AI tự phân tách các tác vụ phức tạp thành các bước nhỏ hơn và lập lộ trình thực hiện.

4. **Cộng tác đa tác vụ (*Multi-agent Collaboration*):** Nhiều tác vụ AI với các vai trò chuyên biệt khác nhau cùng phối hợp để giải quyết bài toán lớn.

1.2.2 Ứng dụng Agentic Workflow vào phát triển phần mềm

Dự án này áp dụng cả 4 patterns trên vào quy trình thiết kế hệ thống:

- **Reflection**: Mỗi Agent Skill có cơ chế Self-Scoring và Self-Verification trước khi bàn giao output
- **Tool Use**: Agents sử dụng file I/O để đọc specs/schemas, Mermaid.js để vẽ sơ đồ, Pencil MCP để vẽ wireframe
- **Planning**: Skill Planner phân rã thiết kế thành các task có truy xuất nguồn gốc rõ ràng
- **Multi-agent**: Pipeline Architect → Planner → Builder, cùng với các Domain Skills (Flow, Sequence, Class, Schema, UI) phối hợp tuần tự

1.2.3 Khái niệm Agent Skill

Trong ngữ cảnh nghiên cứu này, **Agent Skill** là một đơn vị tri thức có cấu trúc, được thiết kế để hướng dẫn AI Agent thực hiện một tác vụ chuyên biệt với chất lượng có kiểm soát. Mỗi Skill bao gồm:

- **Persona**: Vai trò và chuyên môn mà AI đảm nhận (ví dụ: ”Senior System Architect”)
- **Workflow**: Quy trình thực thi có cấu trúc, với các bước rõ ràng
- **Guardrails**: Các hàng rào bảo vệ chống lại hallucination và sai sót
- **Knowledge Base**: Tập hợp kiến thức chuyên ngành cần thiết cho tác vụ

Khái niệm này mở rộng từ ”System Prompt”truyền thống, bổ sung thêm cấu trúc tri thức, cơ chế kiểm soát chất lượng và khả năng tương tác có điểm dừng (Interaction Points) với con người.

1.3 Các công nghệ nền tảng

1.3.1 Next.js 15 và React 19

Next.js là nền tảng (*framework*) phát triển ứng dụng *React* phổ biến nhất hiện nay, cung cấp các tính năng then chốt [2]:

- **Bộ định tuyến ứng dụng (*App Router*):** Hệ thống điều hướng dựa trên tệp tin (*file system*), hỗ trợ cấu trúc giao diện lồng nhau và trạng thái chờ tải (*loading states*).
- **Thành phần phía máy chủ (*React Server Components - RSC*):** Cho phép kết xuất thành phần trực tiếp từ phía máy chủ, giúp giảm tải lượng dữ liệu *JavaScript* gửi về trình duyệt.
- **Hành động từ phía máy chủ (*Server Actions*):** Cho phép gọi trực tiếp các hàm xử lý ở máy chủ từ giao diện người dùng, giúp tối giản hóa lớp giao diện lập trình.
- **Tối ưu hóa tìm kiếm (*SEO Optimization*):** Cơ chế xử lý dữ liệu đặc tả (*Metadata API*) và kết xuất phía máy chủ đảm bảo nội dung được các bộ máy tìm kiếm chỉ mục hóa hiệu quả.

Đối với hệ thống mạng xã hội chia sẻ tri thức, *Next.js* đặc biệt phù hợp nhờ khả năng kết hợp đồng thời cơ chế kết xuất phía máy chủ (SSR - dùng cho các trang tin tức để tối ưu SEO) và kết xuất phía trình duyệt (CSR - dùng cho các tính năng tương tác người dùng).

1.3.2 Payload CMS 3.x

Payload CMS là hệ quản trị nội dung mã nguồn mở (*Headless CMS*) thế hệ mới, nổi bật với triết lý **Ưu tiên mã nguồn** (*Code-first*) — toàn bộ cấu

trúc dữ liệu được định nghĩa trực tiếp bằng *TypeScript* thay vì thông qua các giao diện kéo thả [3]. Các đặc điểm quan trọng bao gồm:

- **Giao diện lập trình nội bộ (*Local API*):** Cung cấp khả năng truy xuất dữ liệu trực tiếp qua các hàm xử lý *TypeScript* (`payload.find()`, `payload.create()`) thay vì phải thông qua giao thức thực thi *HTTP*, giúp giảm thiểu đáng kể độ trễ hệ thống.
- **Kiểm soát truy cập (*Access Control*):** Hệ thống phân quyền linh hoạt, định nghĩa bằng hàm *JavaScript* cho mỗi bộ sưu tập dữ liệu (*collection*).
- **Hệ thống móc nối (*Hooks System*):** Các điểm can thiệp vào vòng đời dữ liệu (*beforeChange, afterChange...*) cho phép thực thi các logic nghiệp vụ tự động trong quá trình xử lý dữ liệu.
- **Tích hợp Next.js**: Payload 3.x chạy trực tiếp bên trong ứng dụng Next.js, chia sẻ chung codebase

Triết lý *Ưu tiên mã nguồn (Code-first)* của *Payload CMS* đặc biệt phù hợp với các tác vụ AI — toàn bộ cấu trúc dữ liệu có thể được AI phân tích và khởi tạo tự động từ các tài liệu đặc tả thiết kế.

1.3.3 MongoDB Atlas

MongoDB là hệ quản trị cơ sở dữ liệu phi cấu trúc (*NoSQL*) định hướng tài liệu (*document-oriented*), lưu trữ dữ liệu dưới dạng tệp tin nhị phân (*BSON - Binary JSON*) [4]. *MongoDB Atlas* là dịch vụ nền tảng đám mây của *MongoDB*, cung cấp các tính năng:

- **Cấu trúc linh hoạt (*Flexible Schema*):** Cấu trúc tài liệu linh hoạt, phù hợp với các loại dữ liệu đa dạng và biến đổi của mạng xã hội (bài viết, bình luận, thông báo...).

- **Atlas Search:** Tích hợp khả năng tìm kiếm toàn văn (*full-text search*) và tự động hoàn thiện (*autocomplete*) trực tiếp mà không cần cài đặt các công cụ tìm kiếm riêng biệt khác.

- **Đường ống tổng hợp (*Aggregation Pipeline*):** Hỗ trợ xử lý và tính toán dữ liệu phức tạp (phân hạng, thống kê, báo cáo) ngay tại tầng cơ sở dữ liệu.

- **Luồng thay đổi (*Change Streams*):** Theo dõi các biến động dữ liệu theo thời gian thực (*real-time*), hỗ trợ đắc lực cho tính năng thông báo tức thời qua cơ chế *SSE*.

Việc lựa chọn *MongoDB* thay vì các cơ sở dữ liệu quan hệ (*SQL databases*) xuất phát từ thực tế: cấu trúc dữ liệu của các nền tảng mạng xã hội thường xuyên thay đổi, yêu cầu khả năng lưu trữ các tài liệu lồng nhau (*nested documents*) và cần khả năng mở rộng hàng ngang (*horizontal scaling*) linh hoạt khi lượng người dùng tăng trưởng nhanh.

1.3.4 Tailwind CSS v4 và Radix UI

Giao diện người dùng được xây dựng dựa trên sự kết hợp giữa **Tailwind CSS v4** (xử lý định dạng) và **Radix UI** (năng lực thành phần cơ sở) [5]:

- **Tailwind CSS v4**: Thư viện thiết kế theo hướng ưu tiên tiện ích (*Utility-first*), cho phép xây dựng giao diện nhanh chóng mà không cần viết các đoạn mã *CSS* tùy chỉnh rời rạc. Phiên bản 4 mang lại những cải tiến vượt trội về hiệu năng và khả năng tùy biến thông qua các biến hệ thống (*CSS variables*).

- **Radix UI**: Bộ thành phần thô phía máy chủ không đi kèm định dạng, cung cấp các chuẩn mực về khả năng tiếp cận (*accessibility*) như phím tắt điều hướng và hỗ trợ trình đọc màn hình. Việc kết hợp với *Tailwind*

CSS cho phép người lập trình kiểm soát hoàn toàn tính thẩm mỹ của sản phẩm (*visual design*).

Quy tắc thiết kế theo phong cách **Neobrutalism**: đường viền đậm (bold borders), bóng đổ lệch (offset shadows), tông màu tương phản cao với primary color là Pink.

1.4 Phương pháp mô hình hóa UML

1.4.1 Tổng quan về UML

Ngôn ngữ mô hình hóa thống nhất (*Unified Modeling Language* — UML) là công cụ tiêu chuẩn được sử dụng rộng rãi để đặc tả, trực quan hóa và xây dựng các hệ thống phần mềm [10]. Nghiên cứu này tập trung khai thác 5 loại sơ đồ *UML* chủ yếu:

Bảng 1.1: Các loại sơ đồ UML sử dụng trong nghiên cứu

Loại sơ đồ	**Phân loại**	**Mục đích**
Sơ đồ ca sử dụng	Cấu trúc	Xác định các tác nhân (*actors*) và chức năng hệ thống
Sơ đồ hoạt động	Hành vi	Mô tả chi tiết luồng nghiệp vụ
Sơ đồ tuần tự	Tương tác	Mô tả sự tương tác giữa các thành phần theo thời gian
Sơ đồ lớp	Cấu trúc	Định nghĩa cấu trúc và mối quan hệ dữ liệu
Sơ đồ thực thể	Cấu trúc	Mô tả quan hệ giữa các đối tượng trong cơ sở dữ liệu

1.4.2 Công cụ vẽ sơ đồ: Mermaid.js

Thay vì sử dụng các công cụ vẽ sơ đồ truyền thống dựa trên giao diện đồ họa, nghiên cứu này ứng dụng **Mermaid.js** — thư viện cho phép tạo sơ đồ thông qua cú pháp văn bản thuần túy (*text-based syntax*) [11]. Điểm đột phá trong quy trình này là các tệp định nghĩa sơ đồ được lưu trữ trực tiếp dưới định dạng `.md` (*Markdown*), giúp các tác vụ AI (*Agent Skills*) dễ dàng đọc hiểu, hiệu chỉnh và đồng bộ hóa với mã nguồn. Các ưu điểm then chốt bao gồm:

- **Cú pháp văn bản thuần túy:** Người dùng và AI có thể mô tả logic sơ đồ qua ngôn ngữ tự nhiên được cấu trúc hóa, loại bỏ sự rườm rà của các thao tác kéo thả thủ công.
- **Xem trước thời gian thực:** Nhờ cấu trúc tệp `.md`, các sơ đồ có thể được hiển thị trực quan ngay trên các trình soạn thảo (như VS Code) hoặc các nền tảng quản lý mã nguồn, đảm bảo tính nhất quán trong thiết kế.
- **Quản lý như mã nguồn (*Code-as-Diagram*):** Sơ đồ được lưu trữ và quản lý phiên bản tương tự như mã nguồn ứng dụng, cho phép truy vết lịch sử thay đổi kiến trúc một cách minh bạch.
- **Thân thiện với trí tuệ nhân tạo (*AI-friendly*):** Đây là ”ngôn ngữ chung”cho phép các tác vụ AI tự động hóa việc vẽ sơ đồ từ các bản đặc tả kỹ thuật (*specs*) mà không cần sự can thiệp trực tiếp từ con người.

1.5 Quy trình phát triển phần mềm và mô hình 4-Life

1.5.1 Mô hình phát triển truyền thống

Quy trình phát triển phần mềm (Software Development Life Cycle — SDLC) truyền thống bao gồm các giai đoạn: Phân tích yêu cầu → Thiết kế →

Triển khai → Kiểm thử → Bảo trì. Các mô hình phổ biến bao gồm Waterfall (tuần tự), Agile (lặp lại) và DevOps (tích hợp liên tục) [12].

1.5.2 Mô hình 4-Life cho phát triển tích hợp AI

Nghiên cứu đề xuất mô hình vòng đời **4-Life Lifecycle** — một biến thể tối ưu hóa của quy trình phát triển phần mềm truyền thống (*SDLC*), được thiết kế chuyên biệt để tương thích với các quy trình làm việc của tác vụ AI (*Agentic Workflow*):

Bảng 1.2: Mô hình 4-Life Lifecycle

Phase	**Tên giai đoạn**	**Mô tả**
Life-1	Vision & Research	Xác định tầm nhìn, personas, user stories, quyết định kỹ thuật
Life-2	Design & Specification	Vẽ sơ đồ UML, thiết kế database, API spec, UI wireframes
Life-3	Implementation	Sinh mã nguồn từ specifications, triển khai hệ thống
Life-4	Verification	Kiểm thử, đánh giá, release

Điểm khác biệt cốt yếu so với các mô hình truyền thống là tại mỗi giai đoạn đều thiết lập các ”Cửa ngõ kiểm soát”(*Phase Gate*) bắt buộc — bao gồm một bộ danh sách kiểm tra các thành phẩm kỹ thuật (*artifacts*) phải hoàn thành trước khi chuyển sang giai đoạn kế tiếp. Các tác vụ AI (*AI Agent*) tham gia chủ đạo tại giai đoạn kiến trúc (*Life-2*) và triển khai (*Life-3*), trong khi con người giữ vai trò định hướng tại giai đoạn tầm nhìn (*Life-1*) và thẩm định kết quả (*Life-4*).

1.6 Phân tích bài toán mạng xã hội chia sẻ kiến thức

1.6.1 Đặc thù của mạng xã hội chia sẻ kiến thức

Mạng xã hội chia sẻ kiến thức (*Social Knowledge Sharing Network*) không chỉ đơn thuần là nơi kết nối cộng đồng mà còn hướng tới mục tiêu trở thành một ”Nhà máy tri thức”(*Knowledge Factory*), nơi tài nguyên thông tin được tạo ra, tinh lọc và không ngừng tái sử dụng. Qua khảo sát thực tế, bài toán này cần giải quyết các thách thức sau:

1. **Tổ chức và truy xuất tri thức:** Trên các nền tảng mạng xã hội truyền thống, nội dung thường bị trôi nhanh và thiếu sự phân loại khoa học. Hệ thống cần triển khai cơ chế ”Lưu trữ tri thức có tổ chức”(*Social Bookmarking*), cho phép người dùng gom nhóm các kiến thức hữu ích thành các ”Bộ sưu tập”(*Collections*) chuyên sâu.

2. **Sàng lọc nội dung chất lượng:** Để tránh tình trạng ”quá tải thông tin”(*Information Overload*), hệ thống ứng dụng thuật toán phối hợp *Time-decay + Engagement Score* — kết hợp giữa yếu tố thời gian và mức độ tương tác thực tế để ưu tiên hiển thị các nội dung giá trị nhất.

3. **Tương tác và chia sẻ tức thời:** Tích hợp khả năng tìm kiếm ngữ nghĩa và hệ thống thông báo thời gian thực giúp thúc đẩy quá trình trao đổi kiến thức diễn ra liên tục.

4. **Đặc thù cộng đồng công nghệ Việt Nam:** Việc xây dựng một nền tảng chuyên biệt giúp hình thành văn hóa lưu trữ tài liệu (*Documentation*) và chia sẻ tri thức một cách bài bản cho các lập trình viên nước nhà.

1.6.2 Khảo sát các hệ thống tương tự

Bảng 1.3: So sánh các nền tảng chia sẻ kiến thức hiện có

Tiêu chí	**Stack Overflow**	**Dev.to**	**Viblo**	**Steve Void**
Mô hình	Hỏi - Đáp	Blog/Xã hội	Blog	Mạng xã hội tri thức
Phân hạng	Dựa vào bình chọn	Lượt tương tác	Lượt xem	Theo thời gian & tương tác
Lưu trữ	Yêu thích	Danh sách đọc	Bookmark đơn	Bộ sưu tập phân loại
Thời gian thực	Không	Hạn chế	Không	Thông báo *SSE*
Hỗ trợ AI	Không	Không	Không	Chu trình kỹ năng (*Agent Skills*)
Tiếng Việt	Hạn chế	Không	Có	Chủ đạo (*native*)

Steve Void tạo ra sự khác biệt thông qua hai yếu tố cốt lõi: (1) toàn bộ quy trình phát triển được tối ưu bằng bộ kỹ năng tác vụ AI tự động hóa, và (2) tích hợp hệ thống bộ sưu tập tri thức (*Collections*) thay vì chỉ lưu trữ đơn thuần.

Việc kết hợp AI vào quy trình phân tích và thiết kế giúp chuyển đổi các yêu cầu nghiệp vụ phức tạp thành các Agent Skills chuyên biệt, đảm bảo hệ thống được xây dựng trên nền tảng kiến trúc vững chắc.

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG AGENT SKILLS

Trong chương này, báo cáo trình bày chi tiết về kiến trúc tổng thể của hệ thống Agent Skills, cũng như phân tích từng bộ kỹ năng chuyên biệt được thiết kế để tự động hóa quá trình phân tích và thiết kế hệ thống trong giai đoạn Life-2.

2.1 Kiến trúc tổng thể hệ thống Agent Skills

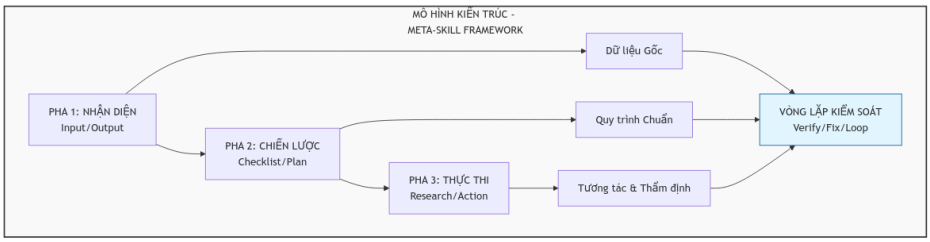
Hệ thống Agent Skills được xây dựng dựa trên mô hình **Meta-Skill Framework**, bao gồm ba trụ cột chính (3 Pillars) và bảy vùng chức năng (7 Zones). Kiến trúc này đảm bảo tính nhất quán, khả năng kiểm soát chất lượng và khả năng mở rộng của các bộ Agent Skill.

2.1.1 Mô hình Meta-Skill Framework

Meta-Skill Framework định nghĩa cách tiếp cận tổng thể để xây dựng một Agent Skill có cấu trúc, logic rõ ràng và khả năng tự kiểm soát chất lượng. Mô hình này bao gồm ba giai đoạn chính:

1. **Pha 1 — Nhận diện (Input/Output)**: Xác định rõ ràng đầu vào và đầu ra mong muốn của skill.
2. **Pha 2 — Chiến lược (Checklist/Plan)**: Lập kế hoạch thực thi với checklist kiểm soát chất lượng.
3. **Pha 3 — Thực thi (Research/Action)**: Nghiên cứu và thực hiện các bước được định nghĩa.

Ba giai đoạn này được kết nối với vòng lặp kiểm soát (Verify/Fix/Loop) để đảm bảo chất lượng đầu ra. Mô hình được minh họa trong Hình 2.1.



Hình 2.1: Mô hình Meta-Skill Framework

2.1.2 Ba trụ cột (3 Pillars)

Kiến trúc Agent Skill được xây dựng trên ba trụ cột chính:

Pillar 1 — Knowledge (Tri thức)

Tập hợp các quy định, tiêu chuẩn kỹ thuật (UML, Schema Design Patterns, Best Practices) cung cấp ”Context”cho AI Agent. Tri thức được tổ chức trong thư mục `knowledge/` và được nạp theo chiến lược Progressive Disclosure (nạp dần theo nhu cầu).

Pillar 2 — Process (Quy trình)

Các bước thực thi được module hóa thành workflow rõ ràng, từ nhận diện đầu vào đến kiểm chứng đầu ra. Mỗi skill định nghĩa riêng quy trình phù hợp với domain của nó.

Pillar 3 — Guardrails (Kiểm soát)

Các hàng rào bảo vệ chống lại hiện tượng ”hallucination”(ảo giác) của AI thông qua các cơ chế: Interaction Gates, Source Citation, Self-Scoring, và Checklist Verification.

2.1.3 Quy trình 5 bước xây dựng Skill

Quy trình xây dựng một Agent Skill tuân theo 5 bước chuẩn được minh họa trong Hình 2.2:

1. **Khảo sát (Research)**: Xác định Pain Point, Input/Output, Tools cần dùng
2. **Thiết kế (Design)**: Xây dựng workflow, Interaction Points, Output format
3. **Xây dựng (Build)**: Viết SKILL.md, tạo templates, scripts, knowledge files
4. **Kiểm định (Verify)**: Chạy Test Cases, Verify Checklist, Rollback nếu fail
5. **Bảo trì (Maintenance)**: Feedback Loop, Version Control, cập nhật khi môi trường thay đổi

Hình 2.2: Quy trình 5 bước xây dựng Agent Skill

2.2 Bộ kỹ năng Meta-Skills cho xây dựng Agent

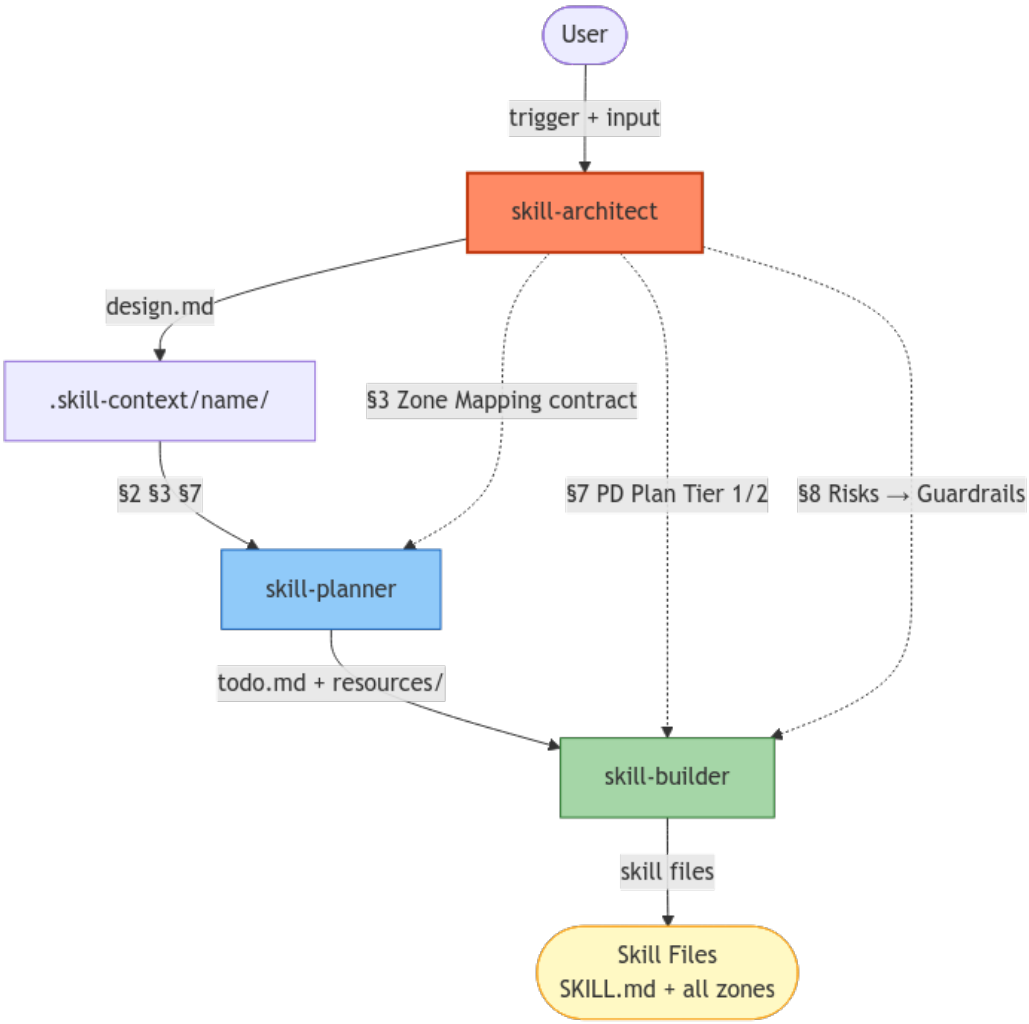
Hệ thống Agent Skills được xây dựng thông qua một bộ ba meta-skills chủ chốt: **Skill Architect**, **Skill Planner**, và **Skill Builder**. Ba skills này hoạt động theo pipeline tuần tự, tạo thành một quy trình tự động hóa hoàn chỉnh để xây dựng các Agent Skills mới.

2.2.1 Skill Architect — Kiến trúc sư thiết kế Skills

Skill Architect là meta-skill trung tâm, chịu trách nhiệm thiết kế cấu trúc cho các Agent Skills khác. Đây là điểm khởi đầu của toàn bộ Skill Suite trong pipeline `Architect → Planner → Builder`.

Vai trò và vị trí trong pipeline

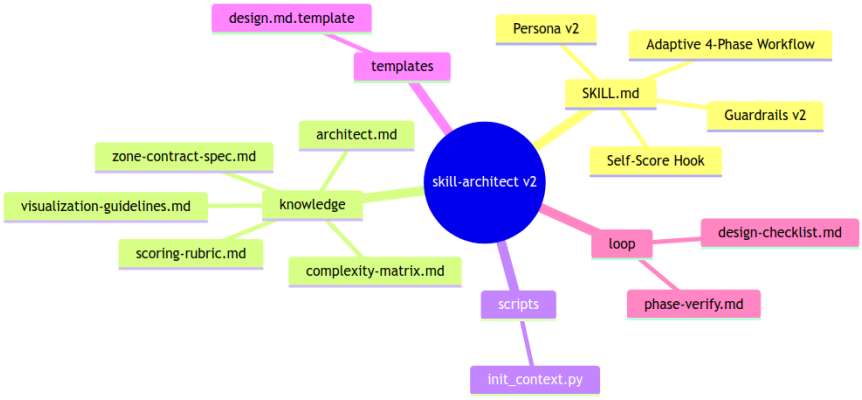
Skill Architect đóng vai trò như một ”kiến trúc sư trưởng”, nhận yêu cầu từ người dùng và tạo ra bản thiết kế chi tiết (`design.md`) cho skill mới. Vị trí của nó trong pipeline được minh họa trong Hình 2.3.



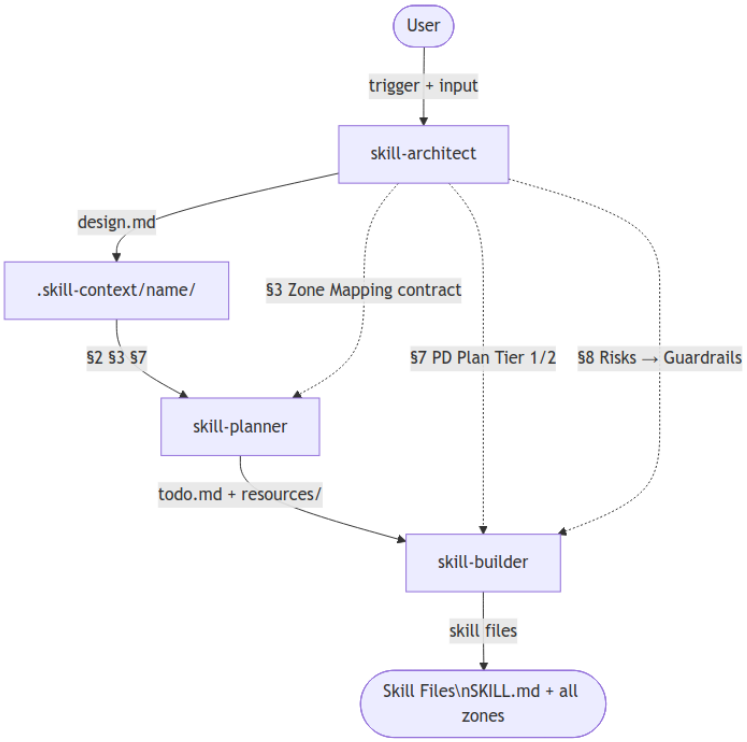
Hình 2.3: Pipeline Skill Suite: Architect → Planner → Builder

Cấu trúc thư mục và Skill Suite Pipeline

Cấu trúc thư mục của Skill Architect v2 bao gồm SKILL.md, 5 knowledge files, scripts, templates và loop. Hình 2.4 minh họa cấu trúc này. Vị trí của Skill Architect trong Skill Suite Pipeline (Architect → Planner → Builder) được thể hiện trong Hình 2.5.



Hình 2.4: Cấu trúc thư mục của Skill Architect v2



Hình 2.5: Pipeline Skill Suite: Skill Architect → Planner → Builder

Kiến trúc 7 Zones

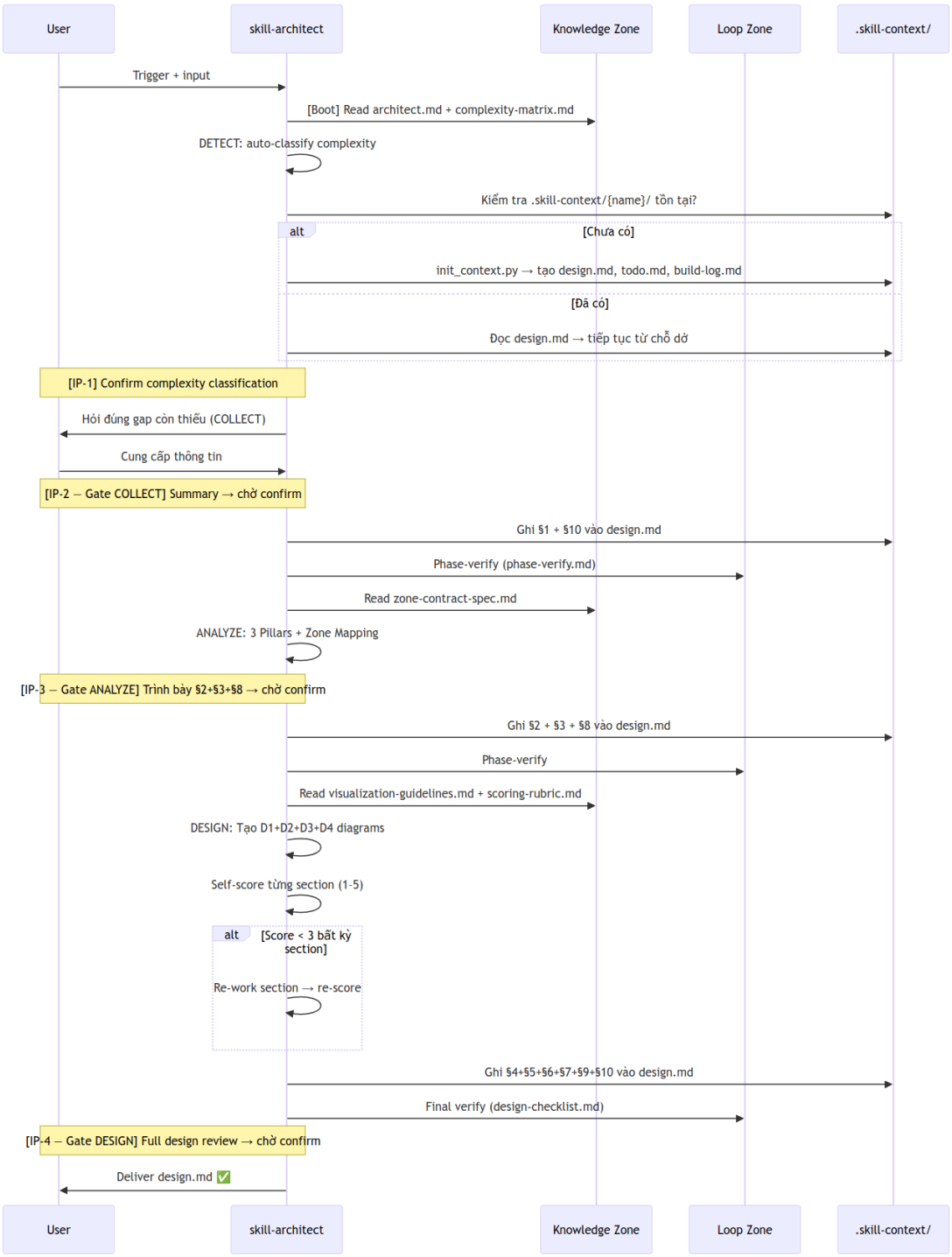
Mọi Agent Skill được tổ chức theo cấu trúc 7 Zones chuẩn, mỗi zone phục vụ một mục đích riêng biệt:

Bảng 2.1: 7 Zones trong cấu trúc Agent Skill

Zone	Nội dung	Bắt buộc?
Core	`SKILL.md` — Persona, workflow, guardrails	Có
Knowledge	Các file `knowledge/*.md` — Standards, best practices, domain knowledge	Có
Scripts	Automation scripts (Python/Bash) — Validation, linting, data processing	Tùy chọn
Templates	Output templates (Mermaid, YAML, code stubs)	Tùy chọn
Data	Static config, registries, sample data	Không bắt buộc
Loop	Quality control checklists, verification scripts	Có
Assets	Images, diagrams, media files	Không bắt buộc

Quy trình Adaptive Workflow và luồng Runtime

Luồng runtime của Skill Architect đi qua 4 interaction points bắt buộc (IP-1: xác nhận hiểu yêu cầu; IP-2: duyệt Zone Mapping; IP-3: duyệt design.md nháp; IP-4: xác nhận giao hàng). AI không được phép bỏ qua bất kỳ gate nào. Hình 2.6 mô tả chi tiết các tương tác giữa User, Skill Architect, Knowledge Zone và thư mục output.



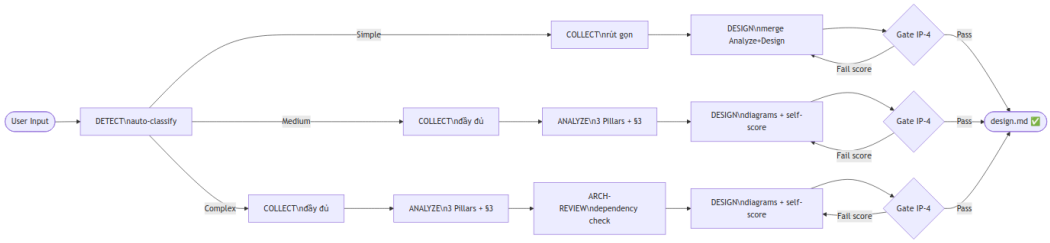
Hình 2.6: Luồng thực thi Runtime (Sequence Diagram) của Skill Architect

Skill Architect v2 giới thiệu cơ chế Adaptive Workflow — tự động phân loại độ phức tạp của yêu cầu và chọn workflow tối ưu:

- **Simple**: COLLECT (rút gọn) → DESIGN (merge Analyze+Design)
- **Medium**: COLLECT → ANALYZE → DESIGN

- **Complex**: COLLECT → ANALYZE → ARCH-REVIEW → DESIGN

Complexity detection dựa trên: số Zones cần dùng, số file cần tạo, mức độ phụ thuộc với skills khác, và độ mới của domain. Hình 2.7 thể hiện 3 nhánh này với Gate IP-4 bắt buộc trước khi bàn giao.



Hình 2.7: Adaptive Workflow của Skill Architect: Simple/Medium/Complex paths

Cơ chế Self-Scoring

Để đảm bảo chất lượng thiết kế, Skill Architect tích hợp cơ chế Self-Scoring — AI tự đánh giá từng section của `design.md` theo thang điểm 1-5. Các tiêu chí đánh giá bao gồm: độ rõ ràng (Clarity), tính đầy đủ (Completeness), tính khả thi (Feasibility), và tuân thủ chuẩn (Standards Compliance).

Quy tắc: Nếu bất kỳ section nào có điểm dưới 3/5, AI phải tự động re-work section đó trước khi chuyển giao cho Planner.

Guardrails chống Hallucination

Skill Architect áp dụng 5 guardrails cứng:

1. **Zone Mapping Contract**: §3 trong `design.md` PHẢI có tên file cụ thể cho mọi Zone, tuân thủ regex `[a-z][a-z0-9_]+\.[a-z]+`
2. **Confidence Threshold**: Nếu confidence < 70% về bất kỳ thông tin nào → BLOCK → Hỏi user → Không tiếp tục
3. **Interaction Gates**: 4 gates bắt buộc (IP-1 đến IP-4) — AI không được bỏ qua

4. **Progressive Disclosure Tiering**: Phân biệt rõ Tier 1 (mandatory) vs Tier 2 (conditional loading)

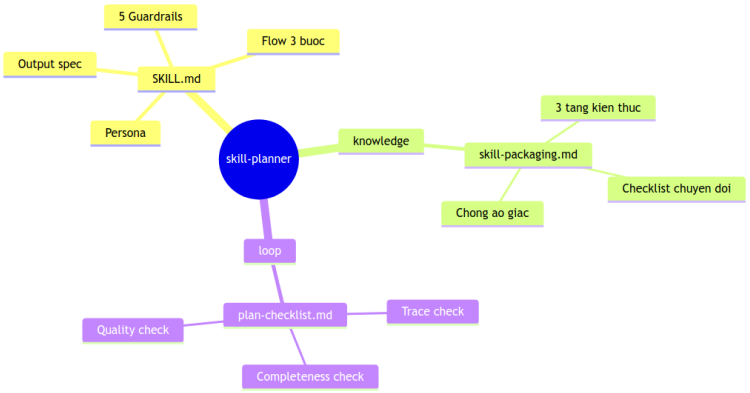
5. **Validation Before Delivery**: Regex validation cho §3, checklist verification từ `loop/design-checklist.md`

2.2.2 Skill Planner — Lập kế hoạch triển khai

Skill Planner là skill #2 trong bộ Master Skill Suite, nhận đầu vào là `design.md` từ Architect và tạo ra `todo.md` — kế hoạch triển khai chi tiết. Planner không tự mình thêm yêu cầu, chỉ phân rã thiết kế thành các task có truy xuất nguồn gốc rõ ràng.

Cấu trúc thư mục

Skill Planner có cấu trúc tối giản — chỉ cần Core (SKILL.md), 1 knowledge file (`skill-packaging.md`) và 1 loop file (`plan-checklist.md`). Hình 2.8 minh họa cấu trúc này.



Hình 2.8: Cấu trúc thư mục tối giản của Skill Planner

Mô hình 3 tầng kiến thức

Planner phân tích mọi Zone trong thiết kế theo 3 tầng:

1. **Tầng 1 — Domain Knowledge**: Kiến thức miền — hiểu bản chất thứ cần làm (UML standards, MongoDB patterns, Business logic)

2. **Tầng 2 — Technical Knowledge**: Kỹ thuật triển khai — công cụ, cú pháp (Mermaid syntax, Python scripting, Regex)

3. **Tầng 3 — Packaging Knowledge**: Đóng gói — map human skill vào 7 Zones (SKILL.md structure, template formatting)

Với mỗi tầng, Planner liệt kê: kiến thức nào cần chuẩn bị, công cụ nào cần, task nào cần thực hiện. Mọi task PHẢI trace về một section cụ thể trong `design.md`.

Quy trình Read → Analyze → Write

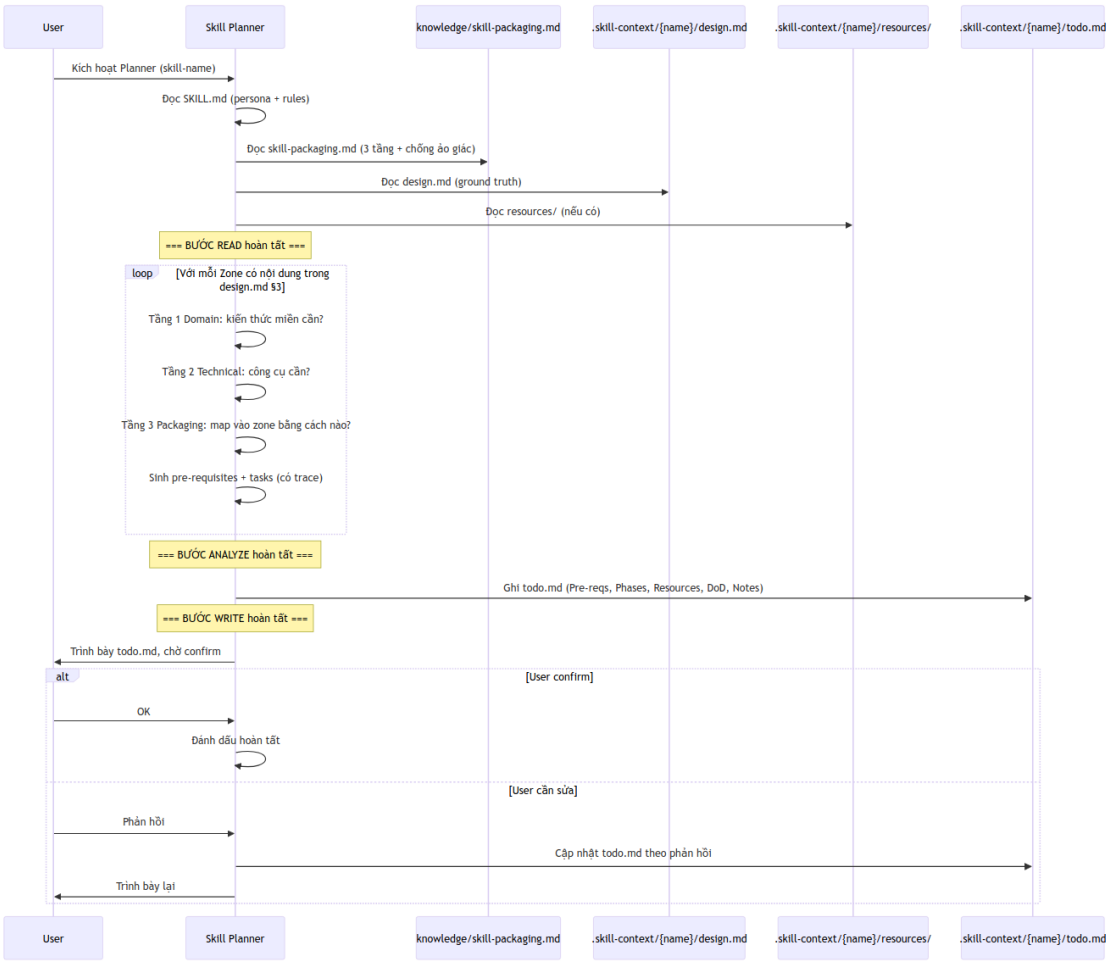
Planner hoạt động theo flow 3 bước nội bộ, 1 interaction point cuối:

1. **READ**: Đọc `design.md`, `resources/`, context từ user

2. **ANALYZE**: Với mỗi Zone, phân tích 3 tầng → sinh pre-requisites + tasks (có trace)

3. **WRITE**: Ghi `todo.md` với format: Pre-reqs, Phases, Resources, Definition of Done, Notes

Interaction point duy nhất: Sau khi ghi `todo.md`, trình bày cho user xem và chờ confirm. Hình 2.9 thể hiện toàn bộ luồng tương tác này giữa User, Planner, knowledge files, design.md và todo.md.



Hình 2.9: Luồng thực thi (Sequence Diagram) của Skill Planner: READ → ANALYZE → WRITE

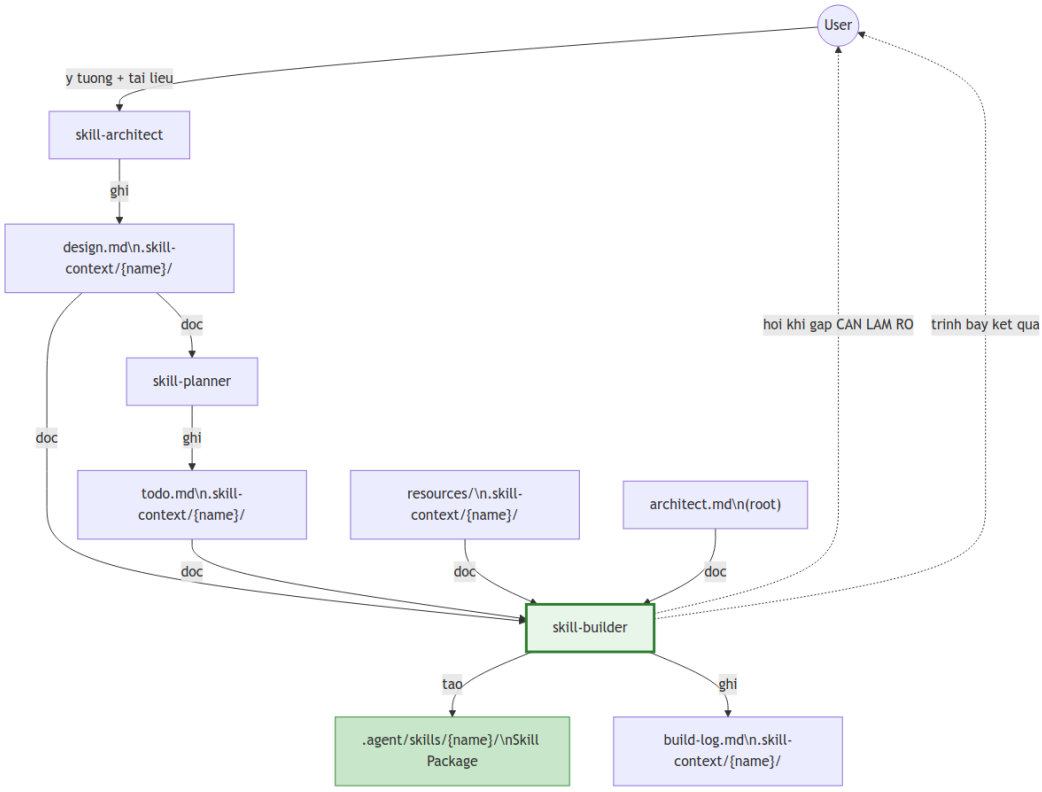
Guardrails chống bịa đặt

- **G1 — Trace bắt buộc**: Mọi item trong `todo.md` PHẢI trace về `design.md §N`
- **G2 — Phân biệt nguồn**: Đánh dấu rõ `[TỪ DESIGN]` vs `[GỢI Ý BỔ SUNG]`
- **G3 — Không phát minh**: Chỉ phân rã thiết kế, không thêm requirements mới
- **G4 — Liệt kê, không tự làm**: Liệt kê kiến thức cần → user chuẩn bị. Không tự search web hoặc hallucinate

- **G5 — Neo vào design.md**: Ground truth duy nhất. Thiếu → ghi Notes, không đoán

2.2.3 Skill Builder — Kỹ sư triển khai

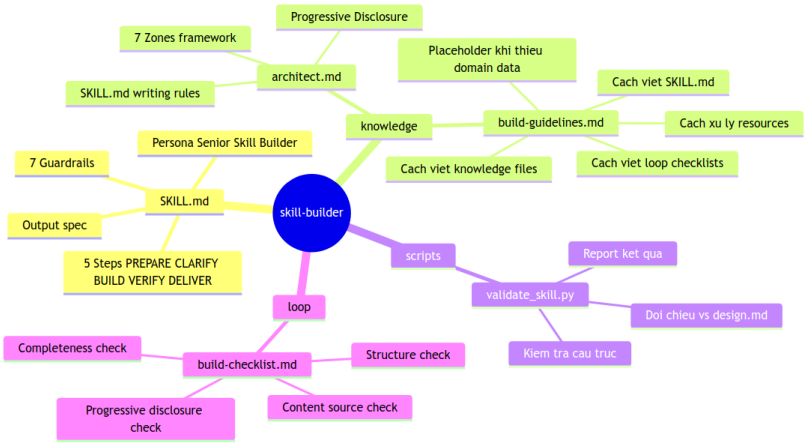
Skill Builder là skill #3 (cuối cùng) trong pipeline Master Skill Suite, nhận `design.md + todo.md` và tạo ra skill package hoàn chỉnh tại `.agent/skills/{skill-name}/`. Hình 2.10 thể hiện vị trí của Builder trong toàn bộ pipeline.



Hình 2.10: Pipeline đầy đủ: User → Architect → Planner → Builder → Skill Package

Cấu trúc thư mục

Skill Builder sử dụng cấu trúc tối giản: Core (SKILL.md), 2 knowledge files và loop. Hình 2.11 minh họa cấu trúc này.



Hình 2.11: Cấu trúc thư mục của Skill Builder

Engineer Stance — Quyền phản biện thiết kế

Khác với Planner (chỉ lập kế hoạch), Builder có quyền **thẩm định logic thiết kế**. Nếu phát hiện thiết kế phi thực tế hoặc mâu thuẫn, Builder phải:

1. Ghi nhận vấn đề vào `build-log.md`
2. Hỏi user để làm rõ
3. Cập nhật `design.md` §9 (Open Questions) nếu cần
4. Chỉ tiếp tục sau khi vấn đề được giải quyết

Chiến lược xây dựng theo từng giai đoạn (*Phase-driven Build Strategy*)

Để tránh hiện tượng quá tải ngữ cảnh (*context overload*), *Builder* thực hiện chia nhỏ giai đoạn xây dựng (`BUILD`) theo từng bước nhỏ trong kế hoạch (`todo.md`). Mỗi giai đoạn hoàn thành sẽ được đánh dấu trạng thái để theo dõi tiến độ.

Quy trình 5 bước thực hiện được minh họa trong Hình 2.12:

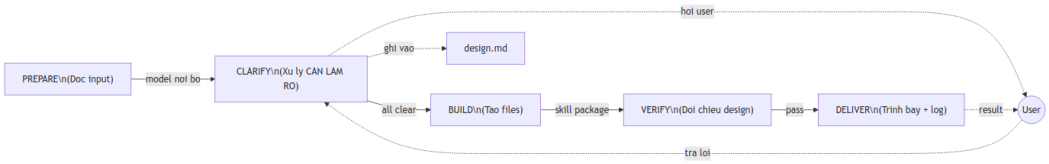
1. **CHUẨN BỊ (*PREPARE*):** Đọc dữ liệu đầu vào và thẩm định logic thiết kế từ bản vẽ.

2. **LÀM RÕ (*CLARIFY*):** Xử lý các điểm còn nghi vấn và thực hiện phản biện thiết kế.

3. **XÂY DỰNG (*BUILD*):** Triển khai từng giai đoạn kỹ thuật, khởi tạo các tệp tin kỹ năng.

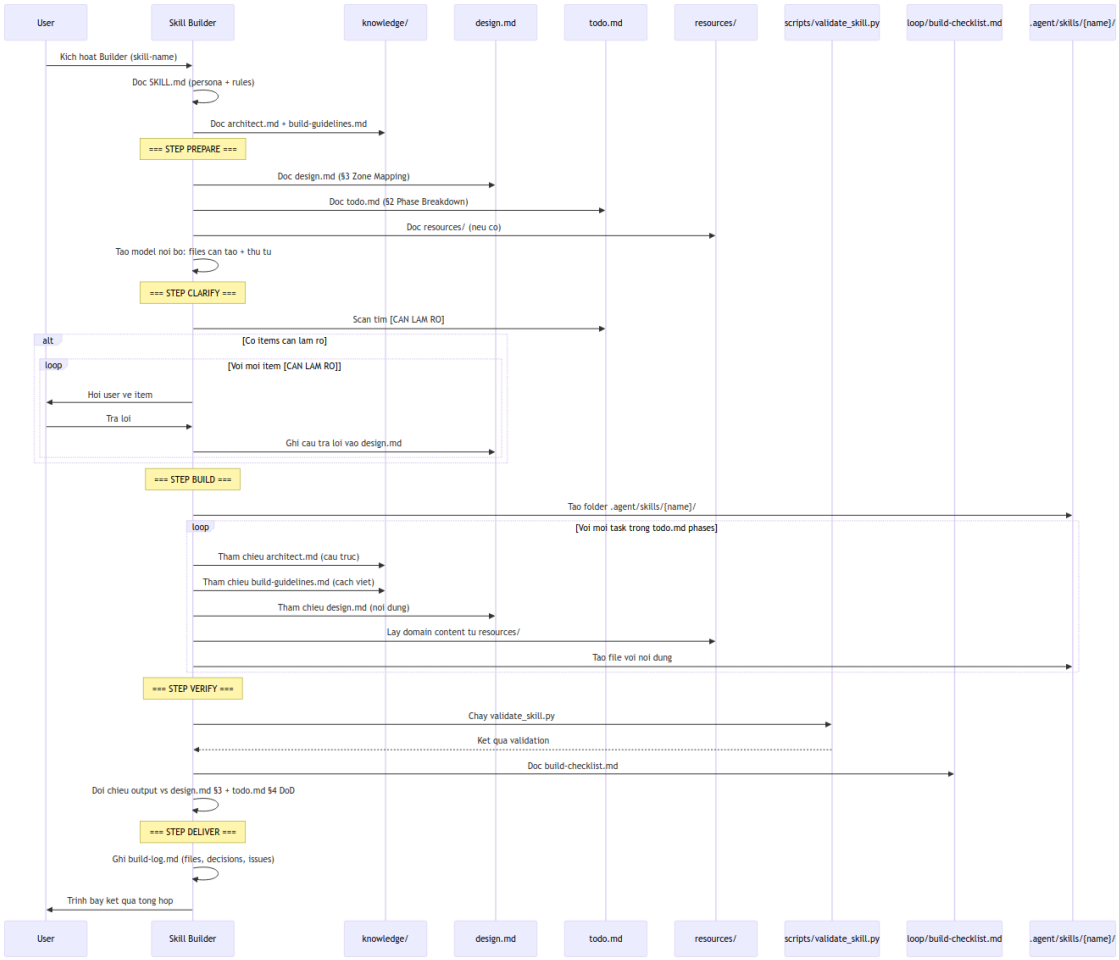
4. **KIỂM CHỨNG (*VERIFY*):** Chạy các kịch bản kiểm tra tự động (*validate_skill.py*) và đối chiếu danh sách kiểm tra.

5. **BÀN GIAO (*DELIVER*):** Ghi lại nhật ký xây dựng (*build-log.md*) và trình bày kết quả cho người dùng.



Hình 2.12: Quy trình 5 bước PREPARE → CLARIFY → BUILD → VERIFY → DELIVER của Skill Builder

Hình 2.13 thể hiện chi tiết luồng tương tác giữa 9 thành phần: User, Builder, knowledge, design.md, todo.md, resources, scripts, loop, .agent/skills.



Hình 2.13: Luồng thực thi chi tiết (Sequence Diagram) của Skill Builder

Log-Notify-Stop Pattern

Khi gặp các lỗi hệ thống nghiêm trọng (lỗi phân quyền, đầy bộ nhớ, hỏng tệp tin), *Builder* áp dụng mô hình xử lý **Ghi nhật ký - Thông báo - Dừng lại** (*Log-Notify-Stop*):

1. **GHI NHẬT KÝ (*LOG*):** Ghi chi tiết lỗi vào tệp tin `build-log.md`.
2. **THÔNG BÁO (*NOTIFY*):** Báo cáo lỗi cho người dùng ngay lập tức.
3. **DỪNG LẠI (*STOP*):** Dừng toàn bộ tiến trình, TUYỆT ĐỐI không cố gắng tiếp tục thực thi khi chưa khắc phục được lỗi.

Tuyệt đối không silent fail hoặc skip lỗi.

Placeholder Gate Mechanism

Builder theo dõi số lượng các nội dung chờ xử lý (*placeholder* — những phần nội dung tạm thời cần người dùng bổ sung sau):

- **Mức 0-4 (Ổn định):** Tri thức về nghiệp vụ đã đầy đủ.
- **Mức 5-9 (Cảnh báo):** Yêu cầu người dùng cung cấp thêm tài nguyên tri thức.
- **Mức 10+ (Thất bại):** Quá nhiều thông tin bị bỏ trống, bản thiết kế thiếu đi ”Căn cứ thực tế”(*ground truth*) để triển khai.

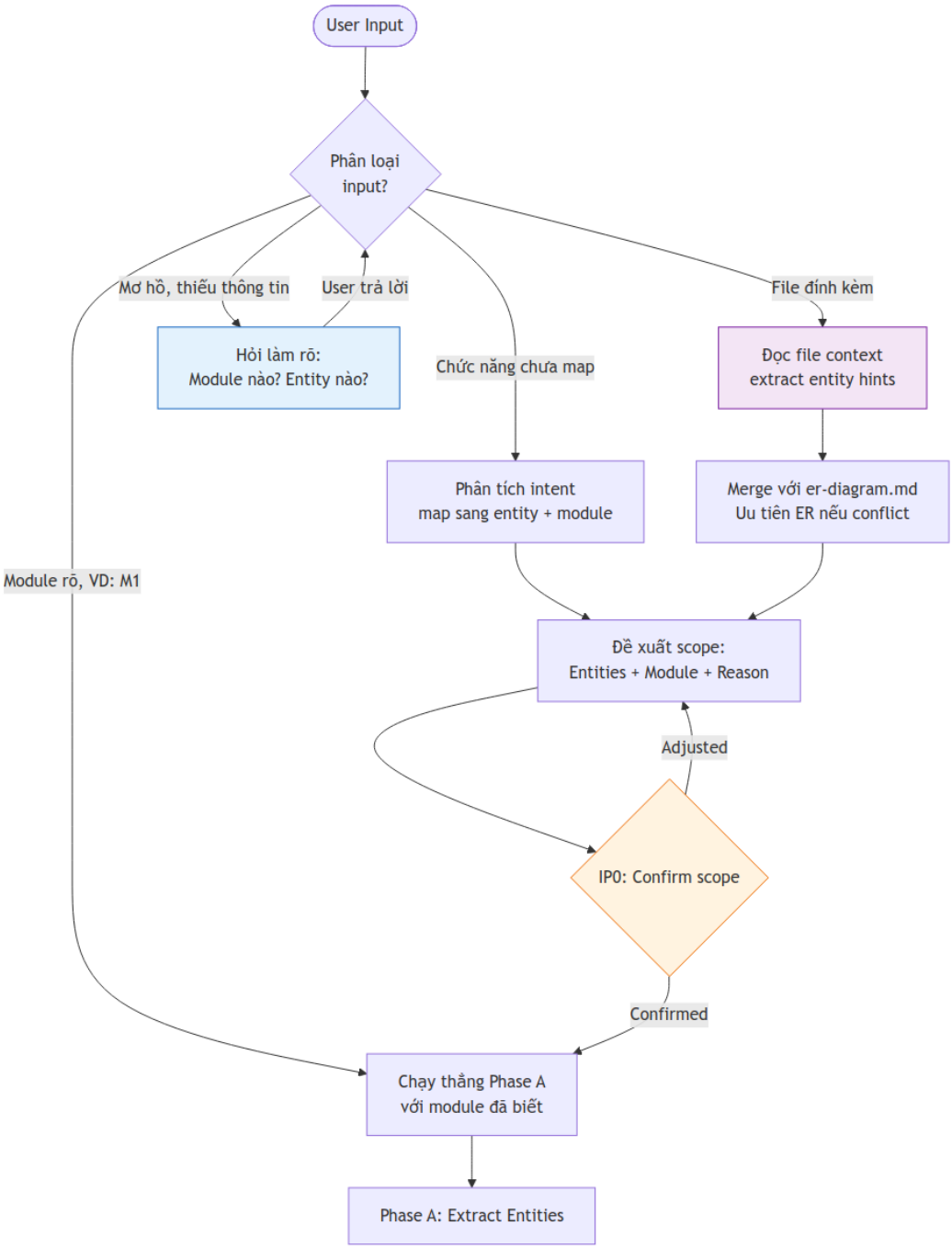
  Thang đo này ngăn chặn việc tạo ra skills rỗng nội dung.

2.3 Bộ kỹ năng thiết kế cấu trúc dữ liệu (Class Diagram Analyst)

Class Diagram Analyst đảm nhiệm việc chuyển đổi từ ER Diagram và quy trình nghiệp vụ sang Class Diagram với định dạng dual-format (Mermaid + YAML Contract). Đây là skill thứ 4 trong chuỗi Life-2, cung cấp Contract YAML làm đầu vào tuyệt đối cho Schema Design Analyst.

2.3.1 Cơ chế xử lý đầu vào (Input Resolution)

Skill hỗ trợ 4 loại đầu vào khác nhau: module rõ ràng, chức năng chưa rõ, yêu cầu mơ hồ, và yêu cầu kèm file đính kèm. Hình 2.14 minh họa cách skill phân loại và định tuyến từng loại tới workflow phù hợp.



Hình 2.14: Input Resolution Flow của Class Diagram Analyst

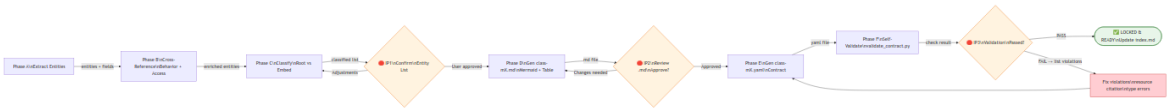
2.3.2 Cấu trúc thư mục

Hình 2.15 thể hiện cấu trúc thư mục đầy đủ của Class Diagram Analyst với 6 zones, bao gồm Scripts để validate và Data cho entity inventory.

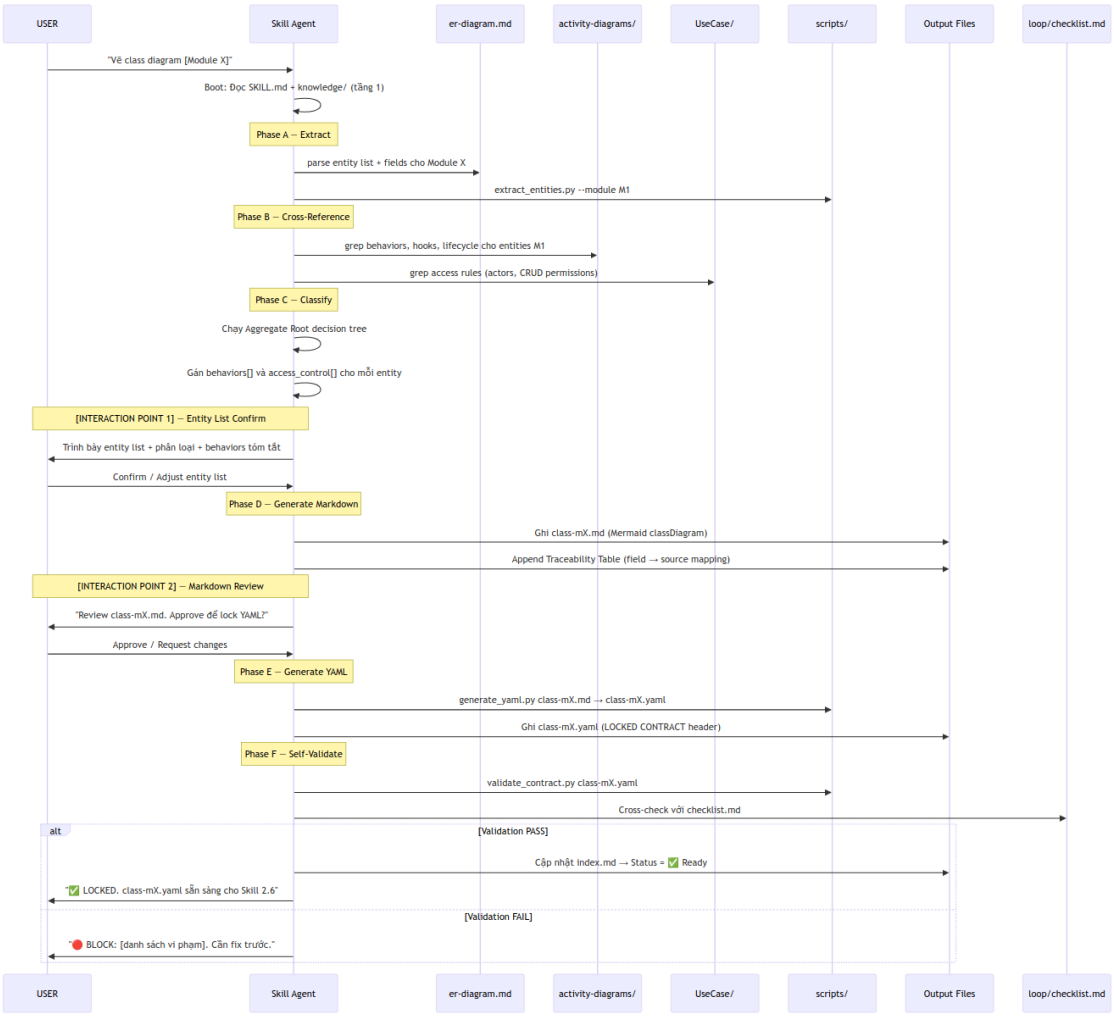
Hình 2.15: Cấu trúc thư mục của Class Diagram Analyst

2.3.3 Quy trình 6 pha và luồng thực thi

Skill thực hiện phân tích qua 6 pha có gate kiểm soát: Phase A (phân tích entity) → B (phân loại Aggregate Root/Embedded) → C (định nghĩa quan hệ) → IP1 → D (sinh Mermaid) → IP2 → E (sinh YAML) → F (validate) → IP3. Hình 2.16 và Hình 2.17 lần lượt thể hiện workflow và luồng tương tác chi tiết.



Hình 2.16: Workflow 6 pha với Gate Control của Class Diagram Analyst



Hình 2.17: Luồng thực thi (Sequence Diagram) của Class Diagram Analyst

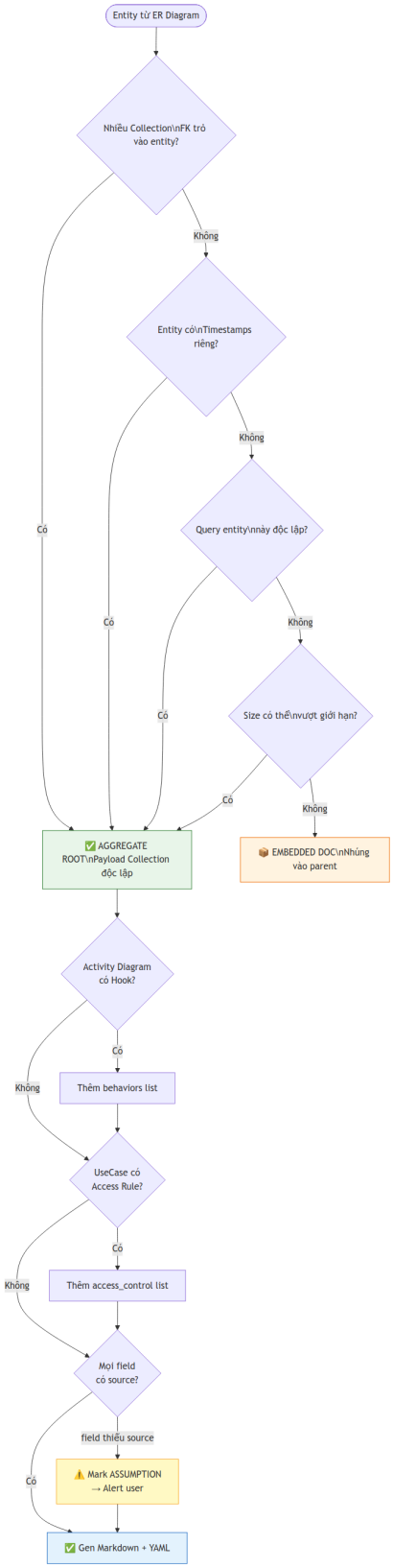
Bộ kỹ năng này tạo ra hai loại kết quả đầu ra song song:

- **Sơ đồ lớp Mermaid** (`class-mX.md`): Định dạng trực quan dùng cho con người theo dõi và kiểm tra.
- **Hợp đồng định dạng YAML** (`class-mX.yaml`): Định dạng dữ liệu máy có thể đọc được (*machine-readable*), phục vụ cho các tác vụ AI khác xử lý tự động.

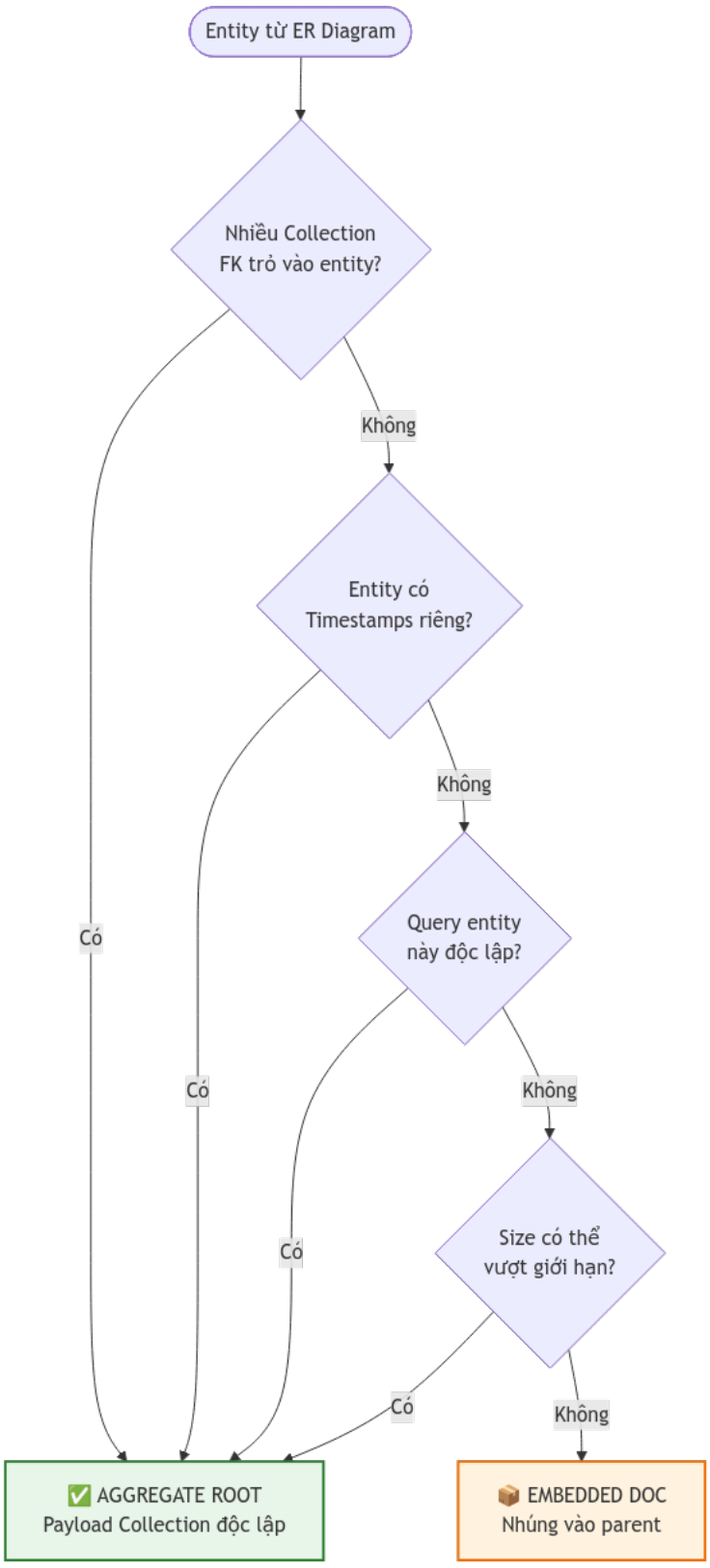
2.3.4 Aggregate Root vs Embedded Document

Một trong những quyết định quan trọng là phân loại entity thành Aggregate Root (collection độc lập) hoặc Embedded Document (nhúng trong

parent). Decision tree được mô tả trong Hình 2.19 (sơ đồ tổng quát) và Hình 2.18 (Decision Tree chi tiết trong Class Diagram Analyst).



Hình 2.18: Decision Tree: Aggregate Root vs Embedded Document trong Class Diagram Analyst



Hình 2.19: Decision Tree: Aggregate Root vs Embedded Document

Để ngăn chặn hiện tượng AI tự ý ”bịa đặt”thông tin (*hallucination*), mọi trường dữ liệu trong sơ đồ lớp PHẢI có nguồn gốc trích dẫn rõ ràng từ

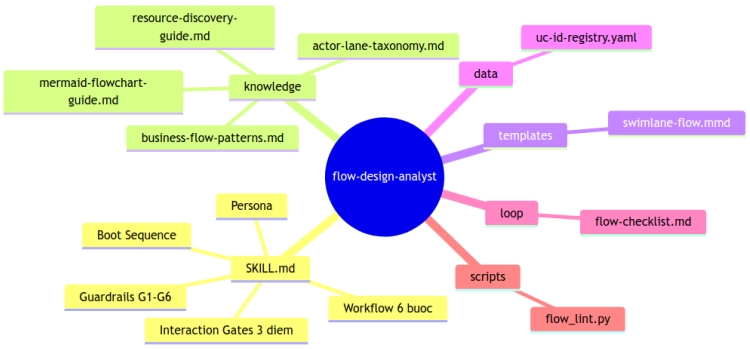
tài liệu đặc tả. Đây là hàng rào bảo vệ (*Guardrail*) bắt buộc: Trường dữ liệu không có nguồn xác thực sẽ bị loại bỏ và không được ghi vào tệp tin.

2.4 Bộ kỹ năng phân tích luồng nghiệp vụ (Flow Design Analyst)

Flow Design Analyst chuyên trách vẽ Business Process Flow Diagram với Swimlane 3-lane (User / System / DB), đảm bảo mọi bước logic đều có căn cứ từ spec hoặc User Story.

2.4.1 Cấu trúc thư mục

Hình 2.20 thể hiện cấu trúc thư mục của Flow Design Analyst, bao gồm knowledge files về Swimlane standards, templates Mermaid, data registry và loop checklist.



Hình 2.20: Cấu trúc thư mục của Flow Design Analyst

2.4.2 Quy trình Discover-before-Ask

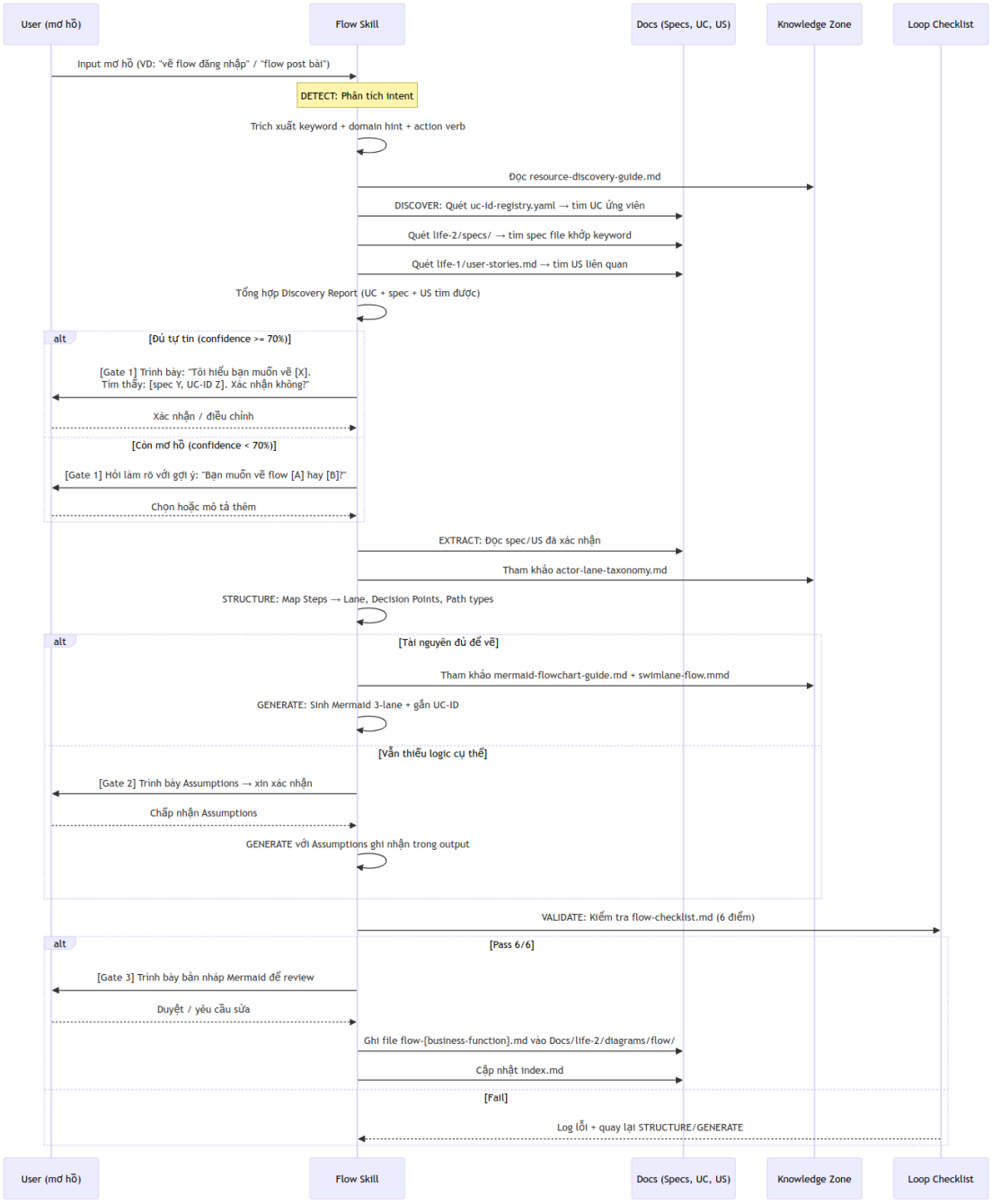
Skill áp dụng nguyên tắc: KHÔNG hỏi user câu hỏi mở khi input mơ hồ. Thay vào đó, skill tự động:

1. **DETECT**: Phân tích intent từ input mơ hồ — trích xuất keyword, domain hint, action verb
2. **DISCOVER**: Tìm kiếm tài nguyên tự động

- Quét `uc-id-registry.yaml` → UC ứng viên
- Quét `Docs/life-2/specs/` → spec file khớp keyword
- Quét `Docs/life-1/user-stories.md` → US liên quan

3. **EXTRACT**: Đọc tài nguyên đã xác nhận, trích xuất Trigger, Actors, Steps, Conditions, Outcomes
4. **STRUCTURE**: Phân bổ từng step vào đúng Actor Lane (User / System / DB)
5. **GENERATE**: Sinh Mermaid flowchart TD với swimlane 3 lanes + gắn UC-ID
6. **VALIDATE**: Kiểm tra: no dangling branch, all paths terminate, UC-ID mapped, Assumptions listed

Hình 2.21 minh họa toàn bộ luồng thực thi này, từ khi nhận input mơ hồ đến khi sinh diagram hoàn chỉnh, với các gates bắt buộc tại từng giai đoạn.



Hình 2.21: Luồng thực thi (Sequence Diagram) của Flow Design Analyst

- **G1 — Hành động có căn cứ:** Mọi nút hành động trong sơ đồ PHẢI dựa trên tài liệu đặc tả hoặc câu chuyện người dùng. Không tự ý thêm bớt các bước xử lý.
- **G2 — Tính đầy đủ của quyết định:** Mọi nút quyết định (hình thoi) PHẢI có đầy đủ các nhánh rẽ đầu ra, tránh các nhánh ”treo”không có điểm đích.

- **G3 — Kỷ luật phân làn (*Lane Discipline*):** Logic hệ thống thuộc làn *System*, thao tác dữ liệu thuộc làn *DB*, và tương tác từ người dùng thuộc làn *User*.
- **G4 — Điểm kết thúc:** Mọi luồng xử lý PHẢI có điểm kết thúc rõ ràng.
- **G5 — Khai báo giả định:** Khi đặc tả chưa rõ ràng, AI PHẢI liệt kê danh sách các ”Giả định”(*Assumptions*) đã sử dụng.
- **G6 — Khám phá trước khi hỏi:** AI PHẢI tận dụng tối đa các tài nguyên hiện có để tự trả lời trước khi đặt câu hỏi cho người dùng.

2.4.3 Actor Lane Taxonomy

Quy tắc phân chia trách nhiệm 3 lanes:

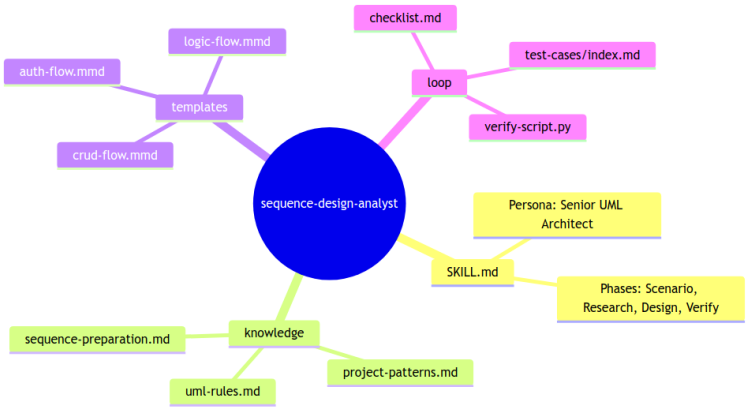
- **User Lane**: UI actions, user decisions, manual input
- **System Lane**: Business logic, validations, transformations, API calls
- **DB Lane**: CRUD operations, queries, transactions

2.5 Bộ kỹ năng phân tích sơ đồ tuần tự (Sequence Design Analyst)

Sequence Design Analyst chuyên vẽ Sequence Diagram chuẩn UML (Mermaid format), mô tả tương tác giữa các đối tượng theo thời gian.

2.5.1 Cấu trúc thư mục

Hình 2.22 thể hiện cấu trúc thư mục tối giản của Sequence Design Analyst — chỉ cần 4 zones thiết yếu: Core, Knowledge (UML standards, codebase patterns), Templates (Mermaid template) và Loop (checklist).



Hình 2.22: Cấu trúc thư mục của Sequence Design Analyst

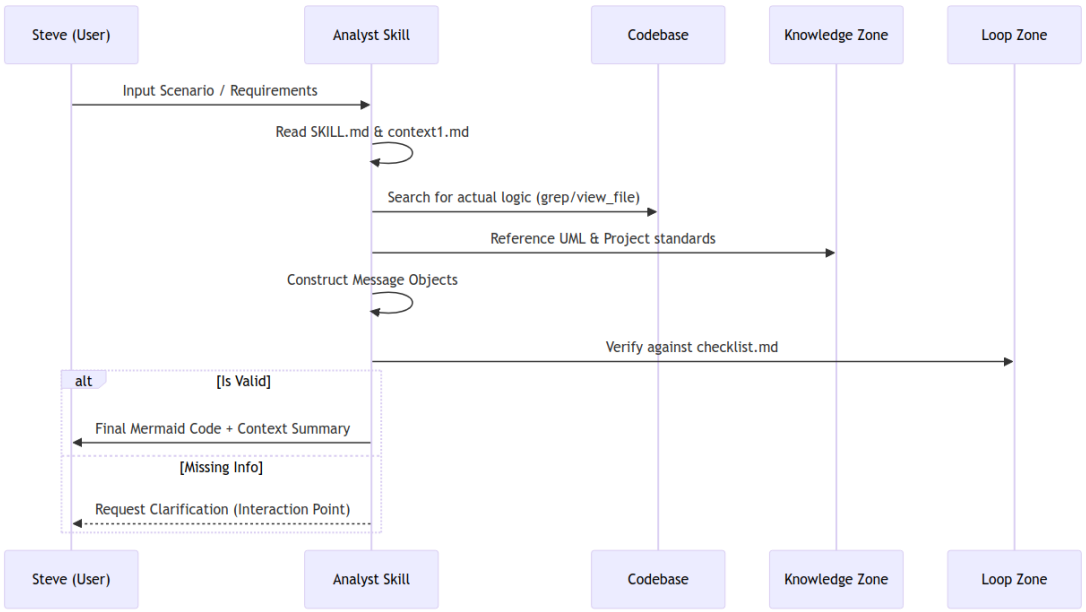
2.5.2 Nguyên tắc Code-First Truth

Skill áp dụng nguyên tắc: Chỉ vẽ những gì thực sự tồn tại trong codebase hoặc được định nghĩa rõ ràng trong thiết kế. KHÔNG vẽ dựa trên suy đoán.

Quy trình:

1. **Scenario Discovery**: Phân tích kịch bản từ input user và `context1.md`
2. **Codebase Research**: Quét codebase để xác định chính xác lifelines (Actor, Service, DB)
3. **Traceability Analysis**: Xây dựng chuỗi tương tác — ai gọi ai, tham số gì
4. **Drafting & Refinement**: Sinh code Mermaid, tối ưu vị trí lifelines
5. **Quality Assurance**: Kiểm tra chéo với Checklist trước bàn giao

Hình 2.23 minh họa luồng thực thi đầy đủ giữa Steve (user), Analyst Skill, Codebase, Knowledge Zone và Loop Zone, bao gồm nhánh xử lý khi thông tin đầy đủ (Valid) và khi thiếu thông tin (Missing Info).



Hình 2.23: Luồng thực thi (Sequence Diagram) của Sequence Design Analyst

2.5.3 UML Fragment Patterns

Skill hỗ trợ các UML fragments chuẩn:

- **alt**: Alternative paths (if-else logic)
- **opt**: Optional execution (if without else)
- **loop**: Iteration patterns (for/while)
- **ref**: Reference to another diagram (sub-sequence)

2.5.4 Naming Consistency Rule

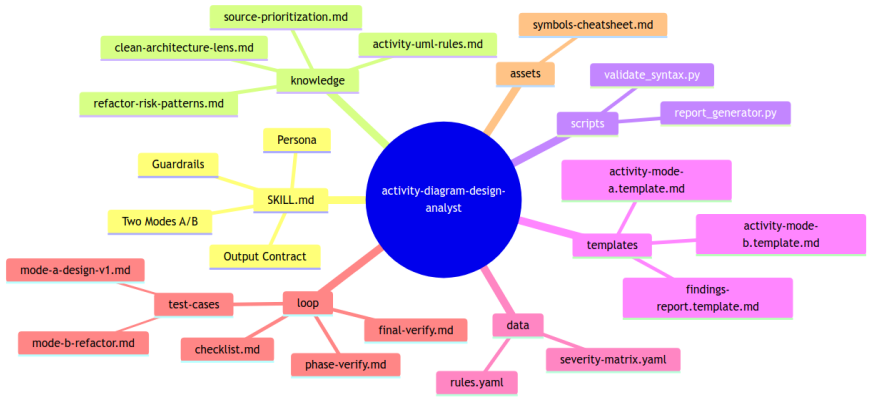
Tên Actor/Object trong sơ đồ PHẢI khớp 100% với codebase. Không được tự ý đổi tên hoặc viết tắt. Ví dụ: Nếu codebase dùng `UserService`, không được viết `UserSvc` hay `User Service`.

2.6 Bộ kỹ năng phân tích sơ đồ hoạt động (Activity Diagram Design Analyst)

Activity Diagram Analyst vẽ sơ đồ hoạt động theo tư duy Clean Architecture (Business-UseCase-External), phát hiện Deadlocks và đảm bảo tính nhất quán giữa nghiệp vụ và thiết kế.

2.6.1 Cấu trúc thư mục

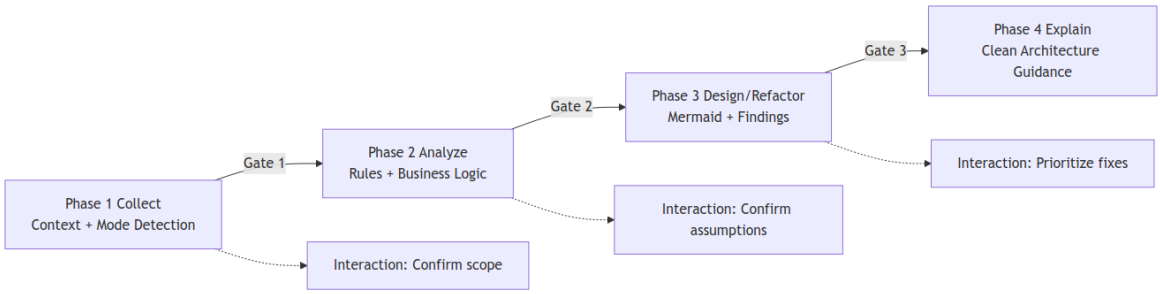
Skill sử dụng đầy đủ 7 zones, bao gồm thư mục `data/` lưu trữ các mẫu pattern phổ biến. Hình 2.24 minh họa cấu trúc này.



Hình 2.24: Cấu trúc thư mục của Activity Diagram Design Analyst

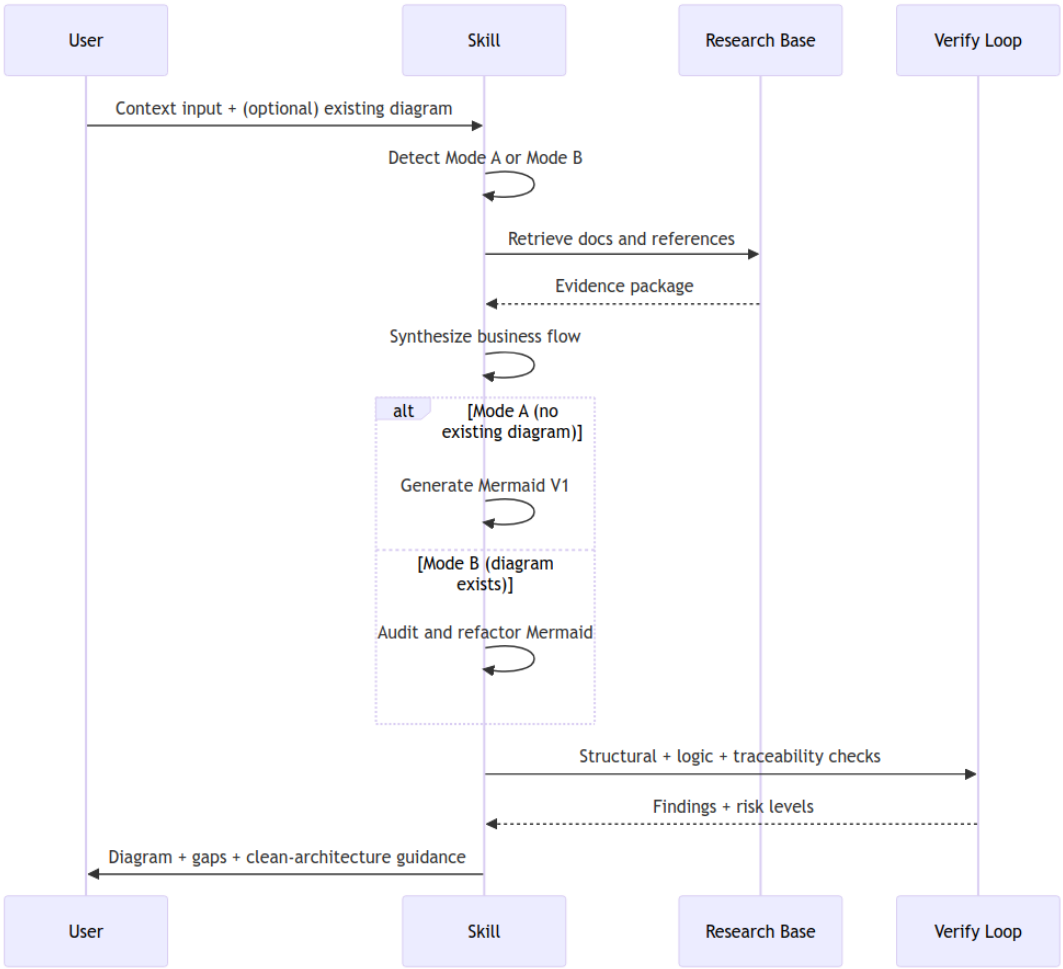
2.6.2 Quy trình 4 pha và 2 Mode thực thi

Skill hỗ trợ 2 mode hoạt động: **Mode A** (thiết kế Activity Diagram mới từ spec) và **Mode B** (refactor diagram hiện có). Cả hai mode đều trải qua 4 pha với gate kiểm soát: Collect → Analyze → Design/Refactor → Explain. Hình 2.25 thể hiện workflow gate-based này.



Hình 2.25: Workflow 4 pha Gate-based của Activity Diagram Design Analyst

Hình 2.26 thể hiện luồng thực thi chi tiết (Sequence Diagram) theo cả 2 mode, với các interaction points tại những điểm cần xác nhận từ user.



Hình 2.26: Luồng thực thi (Sequence Diagram) theo Mode A và Mode B của Activity Diagram Analyst

2.6.3 Clean Architecture Layering (B-U-E)

Sơ đồ activity được tổ chức theo 3 layers:

- **Business Layer (B)**: Core business logic, domain rules
- **Use Case Layer (U)**: Application-specific logic, orchestration
- **External Layer (E)**: Infrastructure, database, external APIs

Quy tắc: Dependency chỉ đi từ ngoài vào trong (E → U → B). KHÔNG cho phép B phụ thuộc vào U hoặc E.

2.6.4 Deadlock Detection

Skill tự động phát hiện các pattern có khả năng gây deadlock:

- Circular dependency giữa các activities
- Race condition khi 2 nhánh song song cùng truy cập 1 resource
- Infinite loop không có exit condition rõ ràng

Nếu phát hiện deadlock risk → Skill báo cáo ngay và yêu cầu user xác nhận logic.

Tương tự như sơ đồ luồng, sơ đồ hoạt động cũng sử dụng các làn bơi (*swimlanes*) nhưng được tổ chức theo các tầng của kiến trúc sạch (*Clean Architecture*):

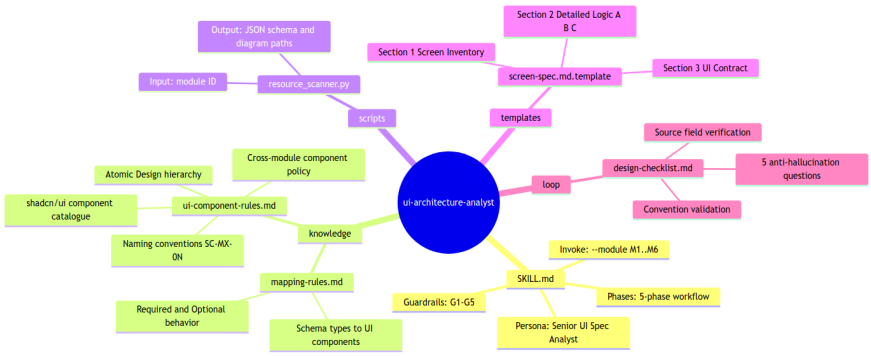
- **Làn Logic nghiệp vụ (*Business Logic Lane*):** Chứa các quy tắc cốt lõi và thực thể tri thức.
- **Làn Trường hợp sử dụng (*Use Case Lane*):** Chứa các dịch vụ ứng dụng và luồng công việc cụ thể.
- **Làn Hạ tầng (*Infrastructure Lane*):** Chứa các bô lưu trữ dữ liệu và dịch vụ bên ngoài.

2.7 Bộ kỹ năng phân tích kiến trúc giao diện (UI Architecture Analyst)

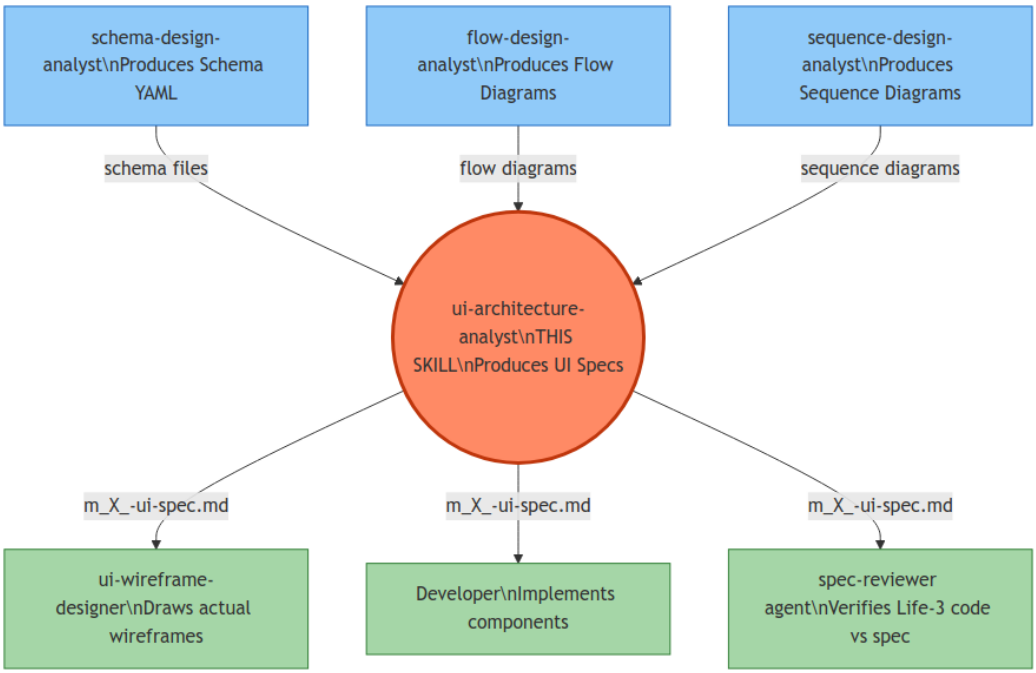
UI Architecture Analyst đóng vai trò là ”cầu nối”giữa logic hệ thống và giao diện người dùng. Skill này chuyển đổi từ Schema + Diagrams → UI Component Specs.

2.7.1 Cấu trúc thư mục và Ecosystem

Skill được tổ chức theo cấu trúc 6 zones gồm: Core (SKILL.md), Knowledge (2 file chuẩn mapping), Scripts, Templates, và Loop. Cấu trúc folder được minh họa trong Hình 2.27. Hình 2.28 thể hiện vị trí của UI Architecture Analyst trong hệ sinh thái skill — nhận đầu vào từ schema-design-analyst, flow-design-analyst, sequence-design-analyst và cung cấp đầu ra cho wireframe-designer và Developer.



Hình 2.27: Cấu trúc thư mục skill UI Architecture Analyst



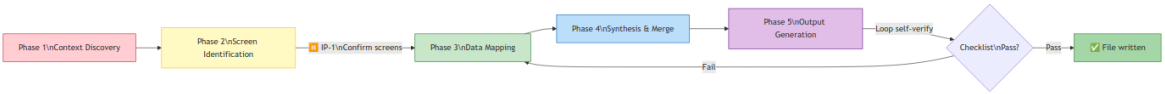
Hình 2.28: Vị trí UI Architecture Analyst trong hệ sinh thái Agent Skills

2.7.2 Data-Component Binding

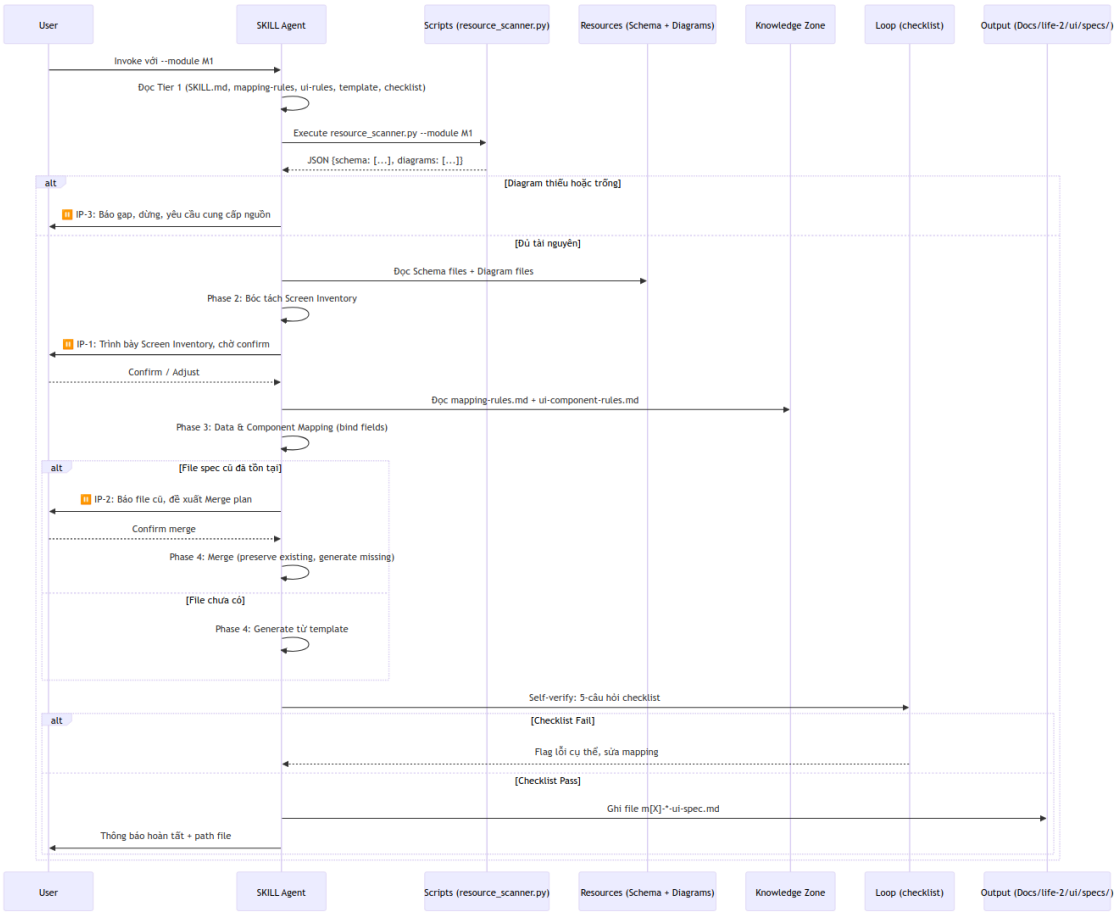
Skill thực hiện ánh xạ từ các trường dữ liệu trong Schema sang các thành phần giao diện thực tế (UI Components) dựa trên bảng mapping rules. Quy tắc: Zero Hallucination — không thêm UI field nếu không có trong Schema.

2.7.3 Quy trình 5 pha và luồng thực thi

Workflow được chia thành 5 pha có gate kiểm soát: Context Discovery → Screen Identification → Data Mapping → Synthesis & Merge → Output Generation, kết hợp với vòng lặp self-verify. Hình 2.29 minh họa workflow này. Hình 2.30 thể hiện luồng tương tác giữa User, SKILL Agent, Scripts, Knowledge và Loop qua 3 interaction points (IP-1, IP-2, IP-3).



Hình 2.29: Workflow 5 pha của UI Architecture Analyst



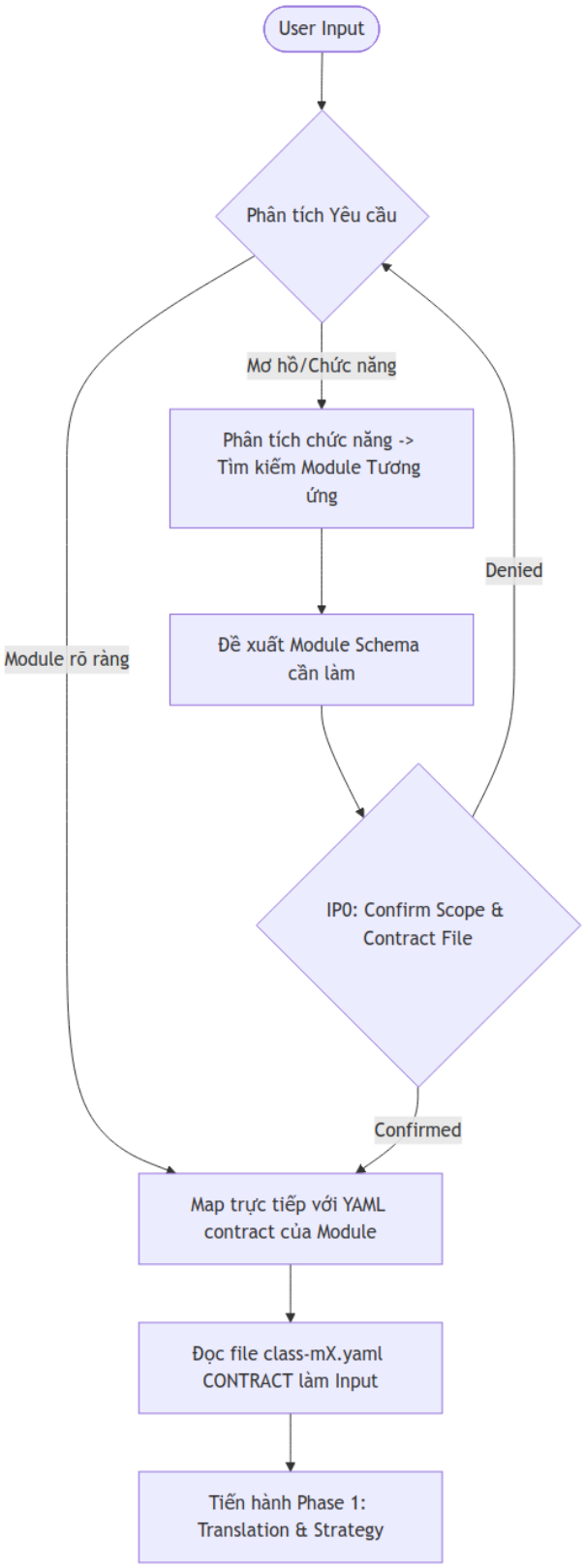
Hình 2.30: Luồng thực thi (Sequence Diagram) của UI Architecture Analyst

2.8 Bộ kỹ năng thiết kế Schema vật lý (Schema Design Analyst)

Schema Design Analyst là ”chốt chặn”cuối cùng trong pipeline thiết kế, chuyển đổi từ Class Diagram (YAML Contract) sang Physical Database Schema cho MongoDB/Payload CMS. Skill này hoạt động như một ”Kiến trúc sư Data”, chỉ làm việc dựa trên Contract YAML từ Class Diagram Analyst, loại bỏ hoàn toàn khả năng AI tự ý thêm field không có trong hợp đồng.

2.8.1 Cơ chế Input Resolution

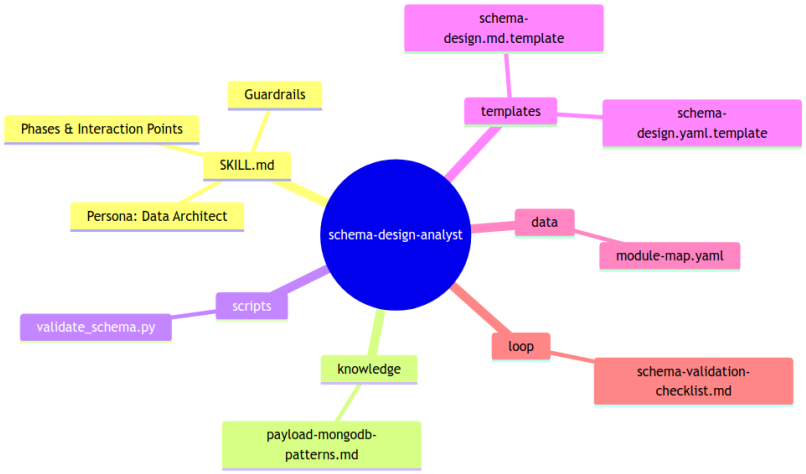
Skill hỗ trợ 4 loại đầu vào: yêu cầu module rõ ràng, yêu cầu theo chức năng, yêu cầu mơ hồ, và yêu cầu kèm file tham chiếu. Với mỗi loại, skill tự động phân tích và định vị Contract YAML tương ứng. Hình 2.31 minh họa luồng phân tích đầu vào này.



Hình 2.31: Luồng phân tích đầu vào của Schema Design Analyst (Input Resolution Flow)

2.8.2 *Cấu trúc thư mục*

Skill được tổ chức với 6 zones chức năng rõ ràng, minh họa trong Hình 2.32:



Hình 2.32: Cấu trúc thư mục skill Schema Design Analyst

Bảng 2.2: Zone Mapping của Schema Design Analyst

Zone	Files tạo ra	Mục đích
Core	`SKILL.md`	Persona, phases, guardrails
Knowledge	`payload-mongodb-patterns.md`	Quy định Embed/Ref, Metadata strategy
Scripts	`validate_schema.py`	Kiểm tra field rác/ảo giác so với hợp đồng
Templates	`schema-design.md.template`, `schema-design.yaml.template`	Format xuất Markdown & YAML
Data	`module-map.yaml`	Map routing các module
Loop	`schema-validation-checklist.md`	Checklist verify data rules

2.8.3 *Luồng thực thi và Dual-Format Output*

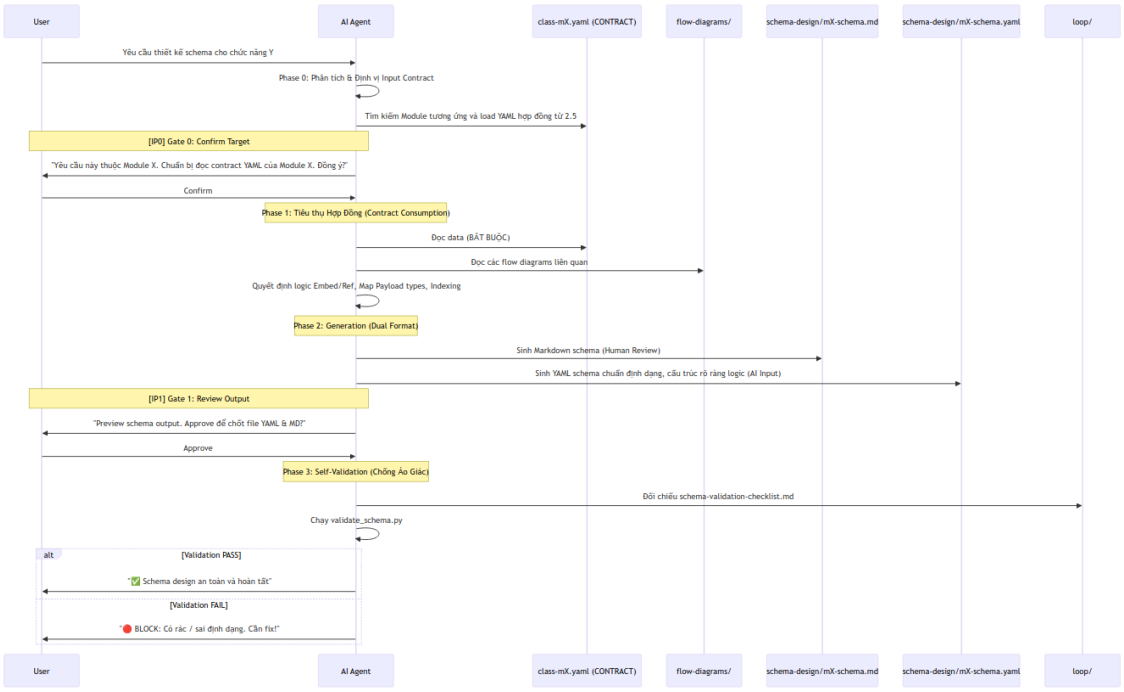
Skill tạo ra hai loại đầu ra: Markdown schema cho con người review (`mX-schema.md`) và YAML schema chuẩn hóa cho AI đọc ở giai đoạn sinh code (`mX-schema.yaml`). Luồng thực thi 4 pha được minh họa trong Hình 2.33:

1. **Giai đoạn 0 — Tiếp nhận hợp đồng (*Phase 0 — Contract Consumption*):** Tiếp nhận và xác thực tính hợp chuẩn của hợp đồng định dạng YAML từ bộ kỹ năng phân tích sơ đồ lớp.

2. **Giai đoạn 1 — Chuyển đổi và Chiến lược (*Phase 1 — Translation & Strategy*):** Quyết định cơ chế lưu trữ (nhúng hoặc tham chiếu), thiết lập ánh xạ các kiểu dữ liệu tương ứng của *Payload CMS* và thiết lập chiến lược đánh chỉ mục (*indexing*).

3. **Giai đoạn 2 — Khởi tạo (*Phase 2 — Generation*):** Khởi tạo đồng thời các bản lược đồ định dạng Markdown và YAML.

4. **Phase 3 — Tự kiểm chứng (*Phase 3 — Self-Validation*):** Thực thi kịch bản kiểm tra tự động `validate_schema.py` để đối chiếu với danh sách kiểm tra quy chuẩn.



Hình 2.33: Luồng thực thi (Sequence Diagram) của Schema Design Analyst

2.8.4 Guardrails chống ảo giác

Skill áp dụng 3 guardrails cứng:

- **G1 — Chỉ từ hợp đồng (*Contract-Only*):** Mọi trường dữ liệu PHẢI có nguồn gốc từ hợp đồng YAML. Các trường không rõ nguồn gốc sẽ bị khóa (`[BLOCK]`).

- **G2 — Kiểm chứng chiến lược (*Strategy Validation*):** Kiểm tra giới hạn kích thước mảng của *MongoDB* (tối đa 16MB) trước khi quyết định sử dụng cơ chế nhúng (*Embed*).

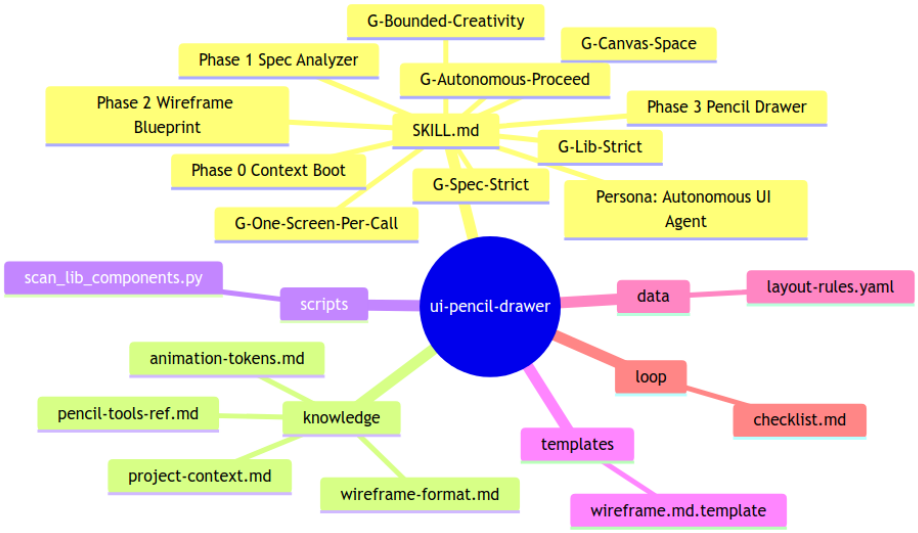
- **G3 — Ranh giới ngữ cảnh (*Context Boundary*):** Chỉ xử lý dữ liệu của một phân hệ (*module*) duy nhất trong mỗi lượt thực thi thông qua tệp cấu hình `module-map.yaml`.

2.9 Bộ kỹ năng vẽ giao diện tự động (UI Pencil Drawer)

UI Pencil Drawer là bộ kỹ năng hoạt động theo triết lý **Thực thi tự chủ hướng tới trí tuệ tổng quát** (*AGI-Oriented Autonomous Execution*) — AI có khả năng tự vận hành xuyên suốt từ bước đọc đặc tả cho đến khi hoàn thiện bản vẽ trên không gian làm việc của *Pencil*, con người chỉ đóng vai trò cung cấp dữ liệu đầu vào. Giải pháp này giúp loại bỏ hoàn toàn các điểm nghẽn (*bottleneck*) khi cần thiết kế khối lượng lớn màn hình giao diện (tất cả 6 phân hệ M1–M6) trong giai đoạn thiết kế kiến trúc (*Life-2*).

2.9.1 Cấu trúc thư mục và 4 Phases tự chủ

Skill được thiết kế với đầy đủ 7 zones, bao gồm 4 knowledge files chuyên biệt cho Pencil MCP API. Cấu trúc folder được minh họa trong Hình 2.34:



Hình 2.34: Cấu trúc thư mục skill UI Pencil Drawer với 4 Phases tự chủ

Bảng 2.3: Zone Mapping của UI Pencil Drawer

Zone	Files tạo ra	Mục đích
Core	`SKILL.md`	Persona + 4 Phases + 5 Guardrails
Knowledge	`pencil-tools-ref.md`, `wireframe-format.md`, `project-context.md`, `animation-tokens.md`	Tham chiếu đầy đủ Pencil MCP API
Scripts	`scan_lib_components.py`	Quét Lib-Component, extract node IDs
Templates	`wireframe.md.template`	Template cho 1 màn hình wireframe
Data	`layout-rules.yaml`	Static rules: gap, padding, screen width
Loop	`checklist.md`	Self-verify pass/fail threshold per phase

2.9.2 Mô hình Autonomous Execution 4 Phase

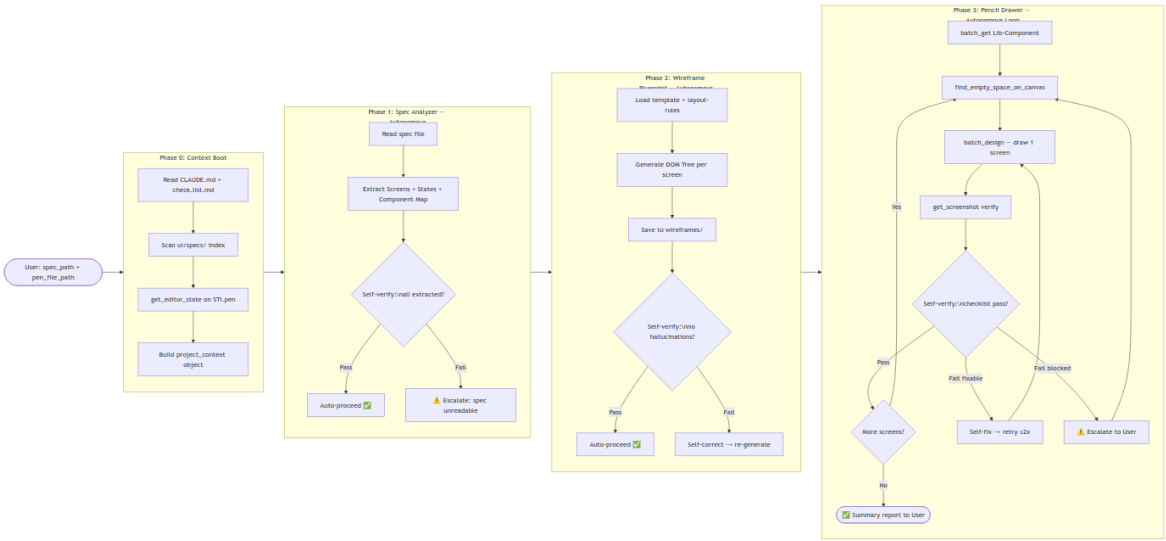
Khác với các skill khác có Interaction Points, UI Pencil Drawer tự chủ hoàn toàn qua 4 pha, chỉ escalate khi bị chặn thực sự:

- **Giai đoạn 0 — Khởi động ngữ cảnh (*Phase 0 — Context Boot*):** Tự động nạp các tệp cấu hình dự án, kiểm tra danh sách màn hình cần vẽ

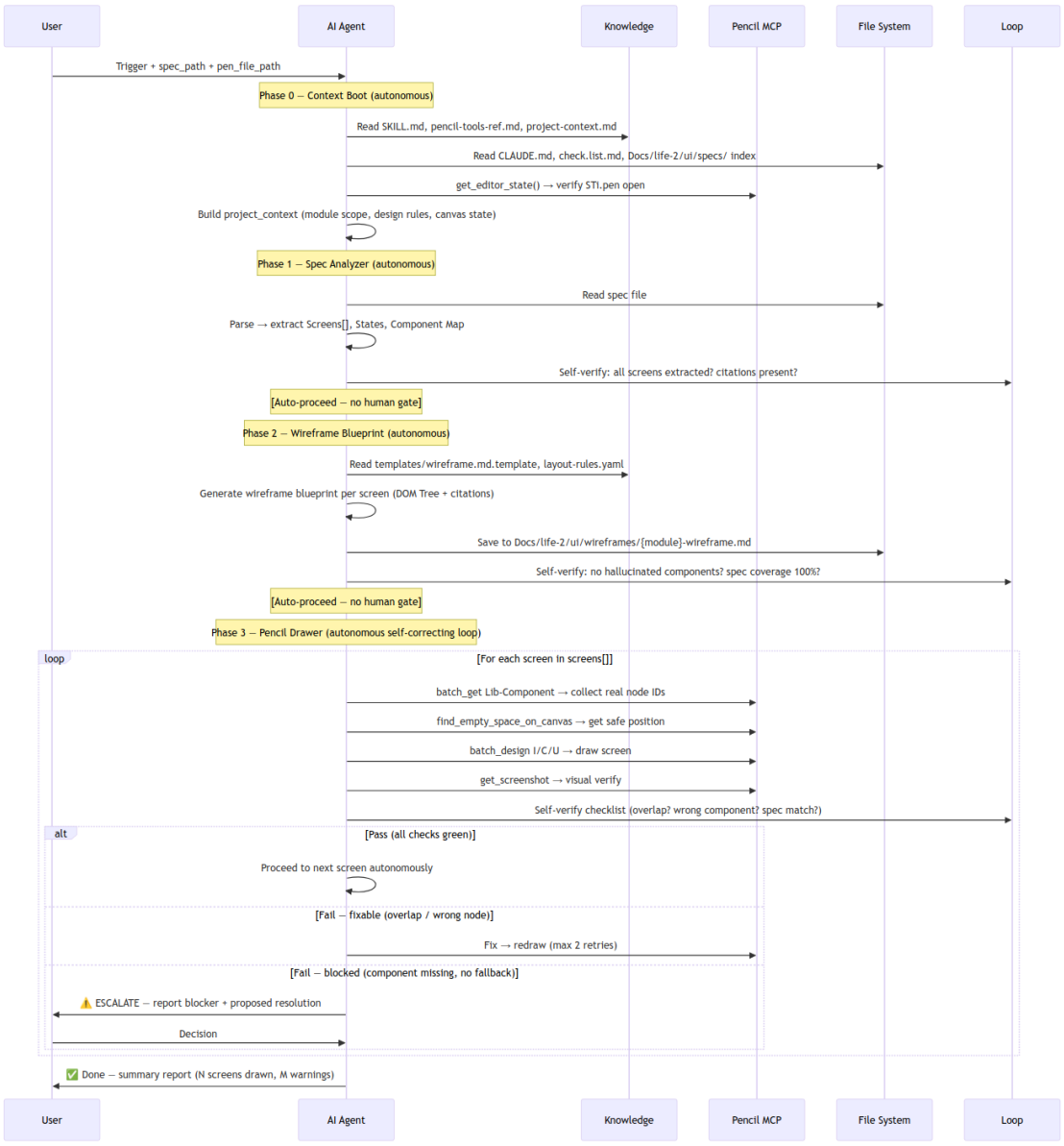
và xác định trạng thái hiện tại của không gian vẽ.

- **Giai đoạn 1 — Phân tích đặc tả (*Phase 1 — Spec Analyzer*):** Phân tích tài liệu thiết kế để trích xuất danh sách màn hình, các trạng thái giao diện và ánh xạ các thành phần với trích dẫn nguồn đầy đủ.
- **Giai đoạn 2 — Bản vẽ phác thảo (*Phase 2 — Wireframe Blueprint*):** Khởi tạo cấu trúc phân cấp (*DOM Tree*) dưới dạng văn bản phác thảo và lưu trữ vào thư mục quy định.
- **Giai đoạn 3 — Vẽ tự động (*Phase 3 — Pencil Drawer*):** Thực thi lệnh vẽ từng thành phần trên không gian làm việc, chụp ảnh màn hình để đối chiếu và tự động sửa lỗi (tối đa 2 lần thử lại).

Hình 2.35 minh họa workflow 4 pha với các nhánh tự xử lý lỗi. Hình 2.36 thể hiện luồng end-to-end giữa User, AI Agent, Knowledge, Pencil MCP, File System và Loop.



Hình 2.35: Workflow 4 pha Autonomous của UI Pencil Drawer



Hình 2.36: Luồng thực thi end-to-end (Sequence Diagram) của UI Pencil Drawer

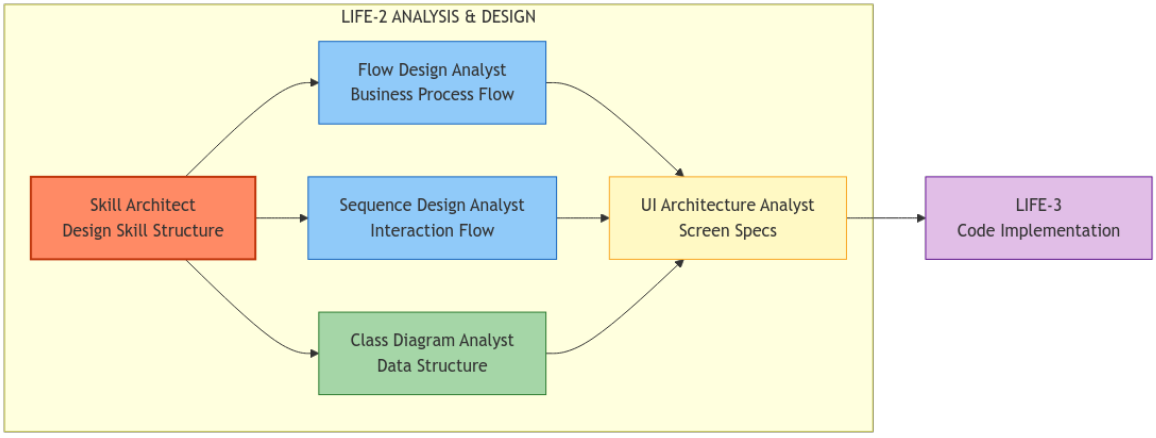
2.9.3 5 Guardrails đảm bảo tính chính xác

- **Kỷ luật thư viện thành phần (*G-Lib-Strict*):** Toàn bộ các thành phần sử dụng PHẢI được tham chiếu (*ref*) từ thư viện thành phần chung (*Lib-Component*). Tuyệt đối không tự vẽ các khối cơ bản nếu thư viện đã có sẵn thành phần tương đương.

- **Kỷ luật đặc tả thiết kế (*G-Spec-Strict*):** Không tự ý thêm các yếu tố giao diện nằm ngoài đặc tả. Mọi thành phần hiển thị đều phải có trích dẫn nguồn từ tài liệu.

- **Kỷ luật không gian vẽ (*G-Canvas-Space*):** Bắt buộc kiểm tra các vùng còn trống trên không gian vẽ trước khi thực hiện các thao tác khởi tạo thành phần mới.
- **Kỷ luật mỗi bước một màn hình (*G-One-Screen-Per-Call*):** Mỗi lượt thực thi lệnh vẽ chỉ được xử lý tối đa một màn hình giao diện, yêu cầu xác thực qua ảnh chụp màn hình trước khi tiếp tục.
- **Kỷ luật giới hạn sáng tạo (*G-Bounded-Creativity*):** Phân định rõ ràng giữa ”Vùng nghiêm ngặt”(thực thi hoàn toàn theo đặc tả) và ”Vùng linh hoạt”(AI được phép sáng tạo về khoảng cách, bố cục phụ trợ và hình ảnh minh họa).

Toàn bộ hệ thống các bộ kỹ năng hoạt động phối hợp theo một quy trình tuần tự và chặt chẽ, tạo thành một hệ sinh thái ”Nhà máy tri thức”(*Knowledge Factory*) tự động hóa. Pipeline hoàn chỉnh được minh họa trong Hình 2.37:



Hình 2.37: Chu trình vận hành tổng thể của hệ thống Agent Skills

Các bước chính trong chu trình:

1. **Thiết kế hệ kỹ năng**: Sử dụng bộ kỹ năng *Skill Architect* để định hình vai trò và ranh giới hoạt động của từng tác vụ AI.
2. **Phân tích nghiệp vụ**: Sử dụng các bộ kỹ năng phân tích luồng, trình tự và hoạt động để xây dựng các khung logic vận hành hệ thống.

3. **Thiết kế dữ liệu**: Chuyển đổi các thực thể tri thức sang cấu trúc hướng đối tượng (*OOP view*) thông qua bộ kỹ năng phân tích sơ đồ lớp.

4. **Thiết kế giao diện**: Thực hiện ánh xạ từ lược đồ dữ liệu sang các thành phần giao diện thực tế thông qua bộ kỹ năng phân tích kiến trúc *UI*.

5. **Đóng gói tri thức**: Tổng hợp toàn bộ kết quả thành hệ thống tài liệu kỹ thuật có tính minh bạch và truy vết cao.

Thiết kế này không chỉ tạo ra một sản phẩm mạng xã hội cụ thể, mà quan trọng hơn là thiết lập được một **Nhà máy sản xuất tri thức (*Knowledge Factory*)** — nơi trí tuệ nhân tạo có thể tự động hóa quy trình phát triển phần mềm một cách tin cậy và có khả năng kiểm soát tuyệt đối.

CHƯƠNG 3. TRIỂN KHAI VÀ THỬ NGHIỆM HỆ THỐNG

Chương này trình bày quá trình triển khai bộ Agent Skills đã thiết kế ở Chương 2 vào thực tế, cùng với toàn bộ kết quả đầu ra thu được khi áp dụng các skills để phân tích và thiết kế hệ thống Steve Void. Các kết quả bao gồm: sơ đồ UML (Use Case, Activity, Sequence, Flow, Class), sơ đồ quan hệ thực thể (ER Diagram), thiết kế cơ sở dữ liệu vật lý (Schema Design), đặc tả giao diện người dùng (UI Specs) và bản vẽ wireframe.

3.1 Mô tả quá trình triển khai hệ thống

3.1.1 Môi trường triển khai

Toàn bộ bộ Agent Skills được triển khai trên nền tảng **Claude Code CLI** — công cụ dòng lệnh của Anthropic, tích hợp trực tiếp vào terminal của lập trình viên. Môi trường triển khai bao gồm:

Bảng 3.1: Môi trường triển khai Agent Skills

Thành phần	**Chi tiết**
Hệ điều hành	Ubuntu Linux 24.04
AI Model	Claude Sonnet 4.6 (Anthropic)
Công cụ CLI	Claude Code v1.x
IDE	Visual Studio Code + Claude Code Extension
Quản lý mã nguồn	Git + GitHub
Công cụ vẽ sơ đồ	Mermaid.js (tích hợp trong Markdown)
Công cụ wireframe	Pencil MCP Server (Model Context Protocol)

3.1.2 Quy trình triển khai theo pipeline

Việc triển khai tuân theo pipeline tuần tự đã thiết kế ở Chương 2 (Hình 2.37). Mỗi module M1–M6 được xử lý qua 6 giai đoạn, với đầu ra của giai đoạn trước là đầu vào của giai đoạn sau:

1. **Giai đoạn 1 — Use Case Analysis**: Xác định actors, use cases và ranh giới hệ thống
2. **Giai đoạn 2 — Activity Diagram**: Phân tích luồng nghiệp vụ chi tiết theo kiến trúc B-U-E
3. **Giai đoạn 3 — Sequence & Flow Diagram**: Mô tả tương tác giữa các thành phần và luồng 3-lane Swimlane
4. **Giai đoạn 4 — Class Diagram**: Chuyển đổi sang cấu trúc OOP với dual-format (Mermaid + YAML)
5. **Giai đoạn 5 — Schema Design**: Sinh Physical Schema cho MongoDB/Payload CMS từ Contract YAML
6. **Giai đoạn 6 — UI Design**: Đặc tả giao diện và vẽ wireframe tự động trên Pencil canvas

Tổng cộng, hệ thống đã xử lý **6 modules** (M1–M6) qua **6 giai đoạn**, tạo ra hơn **100 artifacts** thiết kế hoàn chỉnh.

3.1.3 Cách thức tương tác với Agent Skills

Mỗi Agent Skill được kích hoạt thông qua lệnh slash command trong Claude Code CLI. Ví dụ:

- `/flow-design-analyst` — Vẽ Flow Diagram cho một use case cụ thể

- `/sequence-design-analyst` — Vẽ Sequence Diagram cho một kịch bản
- `/class-diagram-analyst` — Phân tích Class Diagram cho một module
- `/ui-pencil-drawer` — Tự động vẽ wireframe trên canvas

Người dùng cung cấp đầu vào (tên module, use case ID, hoặc mô tả mơ hồ), Agent Skill tự động khám phá tài nguyên dự án (specs, user stories, ER diagram), trích xuất logic nghiệp vụ và sinh đầu ra chuẩn hóa. Các Interaction Points (IP) đảm bảo người dùng kiểm soát được chất lượng tại từng bước quan trọng.

3.2 Trình bày kết quả thử nghiệm

Phần này trình bày chi tiết toàn bộ kết quả đầu ra khi triển khai bộ Agent Skills cho hệ thống Steve Void. Bảng 3.2 tổng hợp số lượng artifacts theo từng loại sơ đồ.

Bảng 3.2: Tổng hợp artifacts thiết kế được tạo ra bởi Agent Skills

Loại sơ đồ	Số files	Agent Skill	Định dạng
Use Case Diagram	9	Thủ công + AI	Mermaid
Activity Diagram	27	Activity Analyst	Mermaid Flowchart
Sequence Diagram	9	Sequence Analyst	Mermaid Sequence
Flow Diagram	27	Flow Analyst	Mermaid Swimlane
Class Diagram	12	Class Analyst	Mermaid + YAML
ER Diagram	1	Thủ công + AI	Mermaid erDiagram
Database Schema	12	Schema Analyst	Markdown + YAML
UI Spec	8	UI Architecture	Markdown
Wireframe	17	UI Pencil Drawer	Markdown blueprint
Tổng cộng	**122**		

3.2.1 Kết quả sơ đồ Use Case

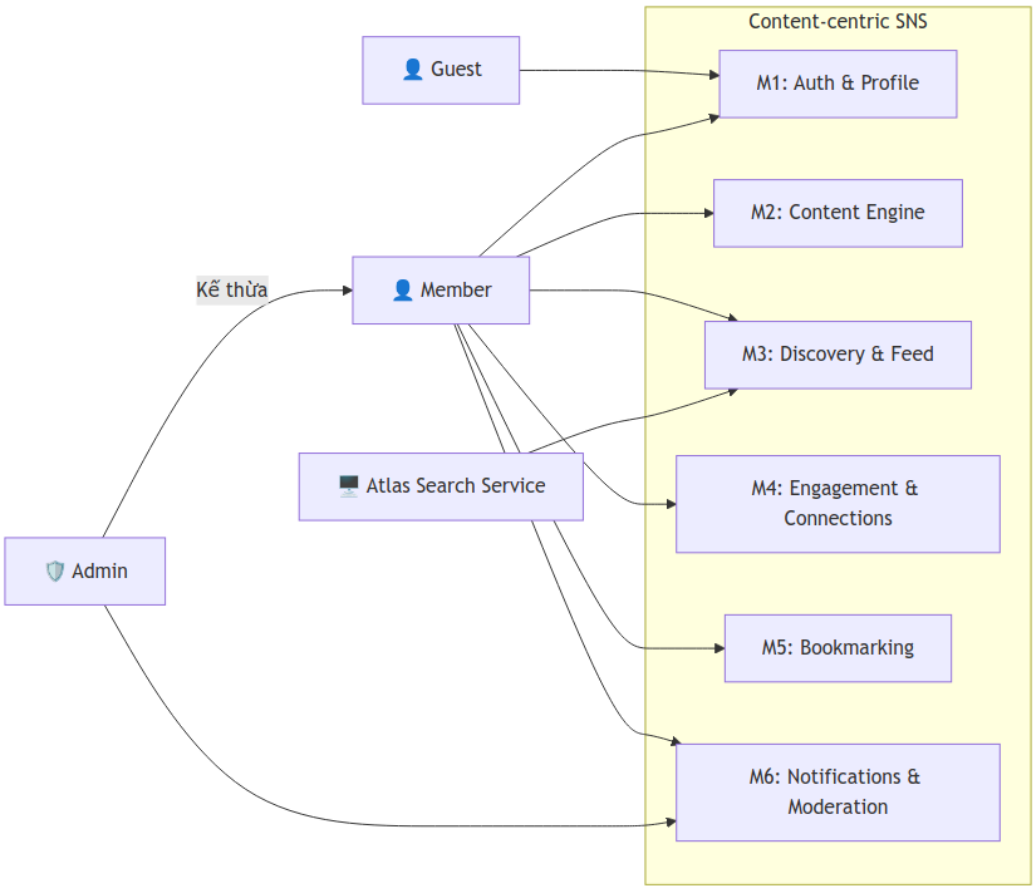
Sơ đồ Use Case xác định ranh giới hệ thống, các tác nhân (Actors) và các trường hợp sử dụng (Use Cases) cho toàn bộ 6 modules. Hệ thống định nghĩa **4 loại tác nhân** với quan hệ kế thừa: Guest → Member → Admin, cùng External Services (OAuth Provider, Atlas Search).

Bảng 3.3: Phân bổ Use Cases theo module

Module	Tên module	Số UC	UC-IDs
M1	Auth & Profile	11	UC01–UC07 (core) + 4 phụ
M2	Content Engine	6	UC08–UC10 + 3 phụ
M3	Discovery & Feed	5	UC11–UC13 + 2 phụ
M4	Engagement	7	UC14–UC18 + 2 phụ
M5	Bookmarking	4	UC19–UC20 + 2 phụ
M6	Notifications & Moderation	8	UC21–UC24 + 4 phụ
Tổng		**41**	

Mỗi Use Case được đặc tả với: Tên, Mô tả, Tác nhân chính, Tiền điều kiện, Luồng chính, Luồng thay thế và Hậu điều kiện. Toàn bộ 24 use case chính (UC01–UC24) đều có truy xuất nguồn gốc (traceability) tới yêu cầu chức năng FR-1 đến FR-10.

Hình 3.1 minh họa sơ đồ Use Case tổng quan của hệ thống, thể hiện mối quan hệ giữa các tác nhân và 6 modules chức năng.



Hình 3.1: Sơ đồ Use Case tổng quan hệ thống Steve Void

3.2.2 Kết quả sơ đồ hoạt động (Activity Diagrams)

Activity Diagram Analyst tạo ra **27 sơ đồ hoạt động** với mức độ chi tiết cao, mỗi sơ đồ phân tích một sub-activity cụ thể theo kiến trúc Boundary-UseCase-Entity (B-U-E). Bảng 3.4 liệt kê phân bổ theo module.

Bảng 3.4: Kết quả Activity Diagrams theo module

Module	Số AD	Các sub-activities
M1	6	A1-Registration, A2-Login, A3-Verification, A4-Recovery, A5-Onboarding + Summary
M2	5	A1-Editor Pipeline, A2-Media Handler, A3-Post Integrity, A4-Visibility + Summary
M3	4	A1-Feed Assembler, A2-Search Engine, A3-Discovery Recommendation + Summary
M4	4	A1-Friendship Handshake, A2-Engagement Logic, A3-Connection Privacy + Summary
M5	3	A1-Bookmark Persistence, A2-Collection Orchestrator + Summary
M6	4	A1-SSE Dispatcher, A2-Report Pipeline, A3-Enforcement Action + Summary

Mỗi Activity Diagram sử dụng Mermaid Flowchart TD (Top-Down) với 3 swimlanes: **User Lane** (UI actions), **System Lane** (business logic, validations) và **DB Lane** (CRUD operations). Các sơ đồ bao gồm đầy đủ: điều kiện rẽ nhánh (Decision Diamonds), xử lý lỗi (Error Paths) và điểm kết thúc rõ ràng cho mọi nhánh.

3.2.3 Kết quả sơ đồ tuần tự (Sequence Diagrams)

Sequence Design Analyst tạo ra **9 sơ đồ tuần tự**, mỗi sơ đồ mô tả tương tác chi tiết giữa các thành phần hệ thống theo thời gian. Các lifelines

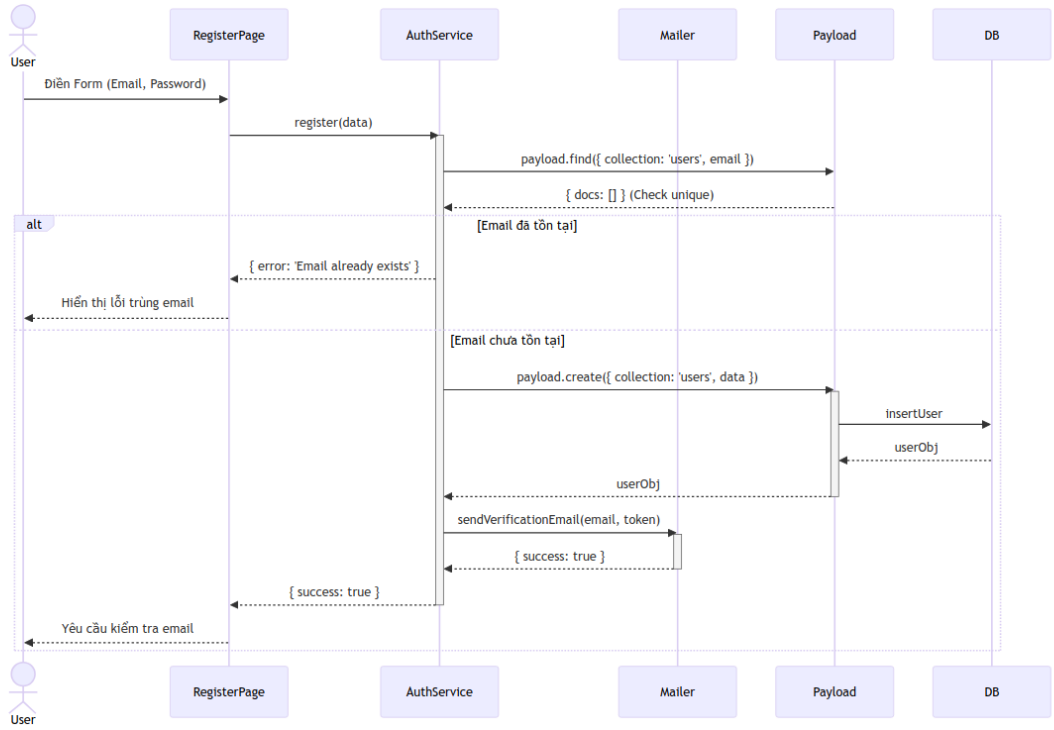
(participants) bao gồm: `User`, `UI (Next.js/React)`, `Service`, `Payload CMS`, `DB (MongoDB)`, `SSE Dispatcher` và `Mailer`.

Bảng 3.5: Kết quả Sequence Diagrams theo module

Module	File	Kịch bản
Global	global-flows.md	Kiến trúc tổng quan, luồng xác thực chung
M1	detailed-m1-auth.md	Login, Register, OAuth, Email verify, Password recovery
M2	detailed-m2-content.md	Editor pipeline, media upload, visibility enforcement
M3	detailed-m3-discovery.md	Feed assembly, search engine, ranking algorithm
M4	detailed-m4-engagement.md	Follow handshake, like/comment, sharing
M5	detailed-m5-bookmarking.md	Bookmark toggle, collection CRUD
M6	detailed-m6-safety.md	SSE notifications, report pipeline, moderation queue

Các sơ đồ sử dụng đầy đủ UML fragments chuẩn: `alt` (alternative paths), `opt` (optional execution), `loop` (iteration) và `activate/deactivate` blocks để thể hiện thời gian xử lý. Nguyên tắc **Code-First Truth** đảm bảo tên các đối tượng trong sơ đồ khớp 100% với codebase dự kiến.

Hình 3.2 minh họa sơ đồ tuần tự cho kịch bản Đăng ký tài khoản (M1-A1), thể hiện tương tác giữa User, RegisterPage, AuthService, Mailer, Payload CMS và Database.



Hình 3.2: Sơ đồ tuần tự: Đăng ký tài khoản (M1-A1)

3.2.4 Kết quả sơ đồ luồng nghiệp vụ (Flow Diagrams)

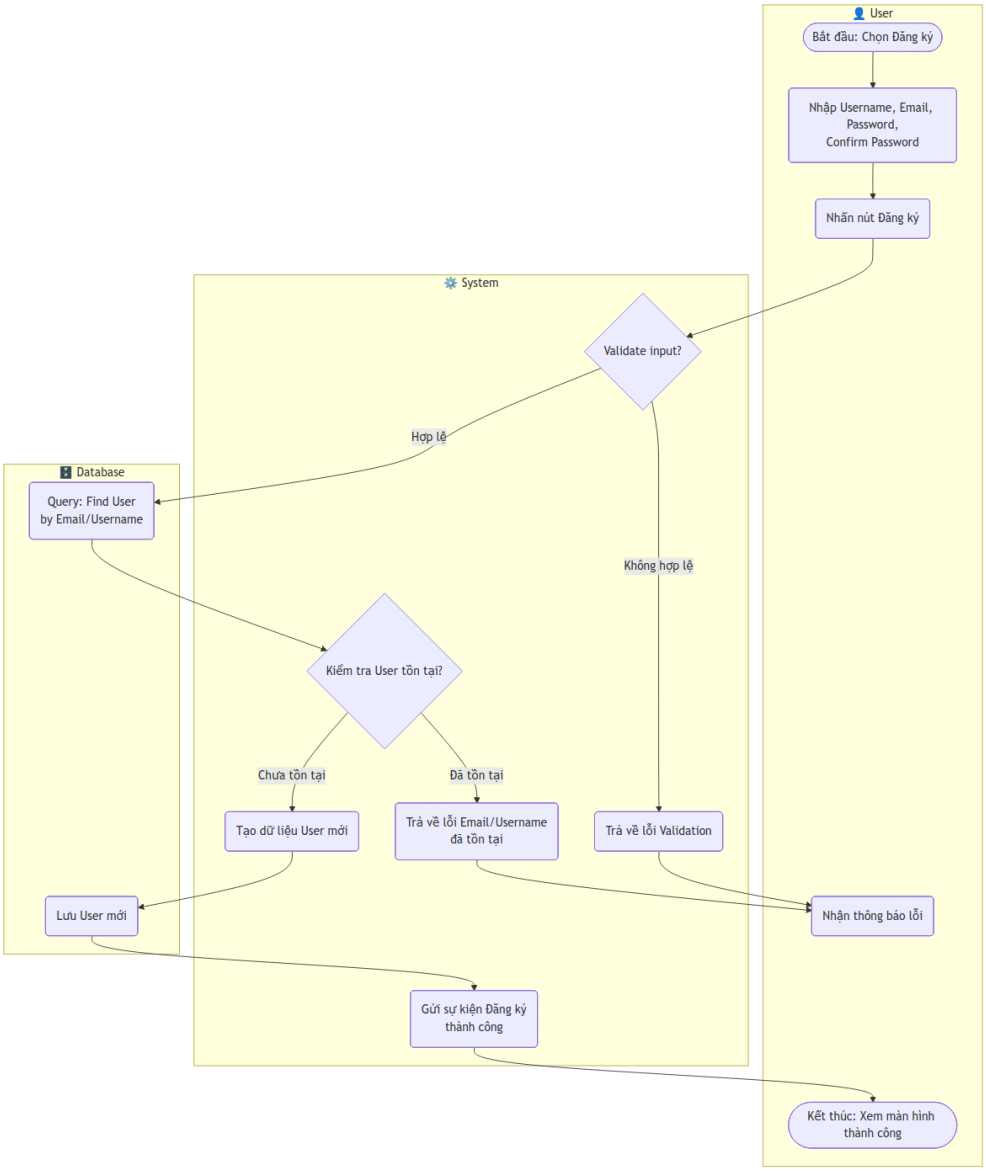
Flow Design Analyst tạo ra **27 sơ đồ luồng nghiệp vụ**, bao phủ **100% use cases chính** (UC01–UC24). Mỗi sơ đồ sử dụng kiến trúc 3-lane Swimlane: **User** (tác nhân), **System** (logic nghiệp vụ) và **Database** (thao tác dữ liệu).

Bảng 3.6: Kết quả Flow Diagrams theo module và Use Case

Module	Số Flow	Các luồng nghiệp vụ
M1	7	Registration, Login Email, Login OAuth, Logout, Password Recovery, Profile Management, Public Profile View
M2	3	Post Creation, Post Modification, Post Privacy
M3	3	News Feed View, Search Engine, Search Autocomplete
M4	5	Post Reaction, Post Comment, Post Share, User Follow, User Block
M5	2	Bookmark Toggle, Bookmark Collection
M6	4	Notification Realtime, Notification Read, Content Report, Moderation Review

Đặc biệt, mọi Flow Diagram đều được gắn mã **UC-ID** (Use Case ID) để đảm bảo truy xuất nguồn gốc ngược lại yêu cầu chức năng. Guardrail **G1 — No Blind Step** của Flow Design Analyst đảm bảo không có bước nào trong sơ đồ được thêm vào mà không có căn cứ từ spec hoặc user story.

Hình 3.3 minh họa Flow Diagram cho chức năng Đăng ký tài khoản (UC01), với 3 swimlanes User / System / Database.



Hình 3.3: Sơ đồ luồng nghiệp vụ: Đăng ký tài khoản (UC01)

3.2.5 Kết quả sơ đồ lớp (Class Diagrams)

Class Diagram Analyst tạo ra **12 files** (6 Mermaid + 6 YAML Contract) cho toàn bộ 6 modules, với **93 trường dữ liệu** được kiểm chứng và **0 trường giả tạo** (0 assumptions unresolved).

Bảng 3.7: Kết quả Class Diagrams theo module

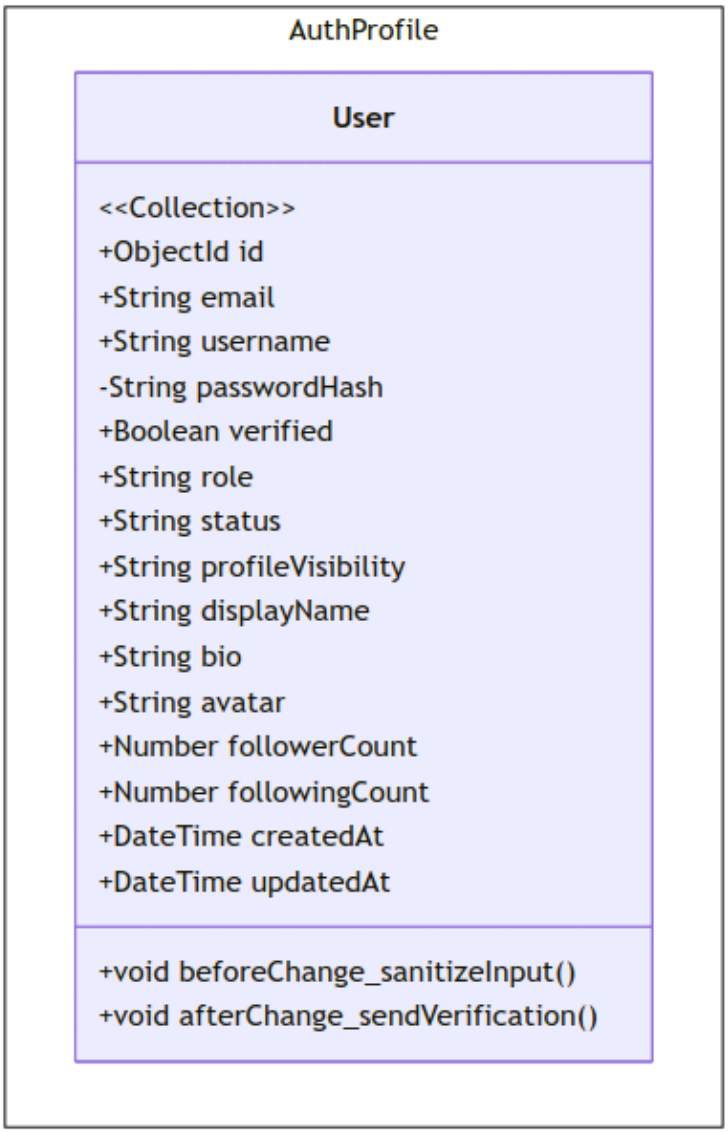
Module	Entities	Số fields	Ghi chú
M1	Users	19	Computed Pattern cho counts
M2	Posts, Media, Tags	23	Denormalized counters
M3	FeedQuery (ValueObject)	9	Không tạo collection vật lý
M4	Comments, Likes, Connections, Shares	20	Compound unique indices
M5	BookmarkCollections, Bookmarks	9	isDefault constraint
M6	Notifications, Reports, AuditLogs	21	Polymorphic + Append-Only

Chiến lược **Dual-Format Output** tạo ra hai loại đầu ra cho mỗi module:

- **Mermaid classDiagram** (`class-mX.md`): Dạng trực quan cho con người review, bao gồm trường dữ liệu, phương thức và quan hệ
- **YAML Contract** (`class-mX.yaml`): Dạng machine-readable cho Schema Design Analyst đọc ở giai đoạn tiếp theo

Mỗi trường trong Class Diagram đều có **Source Citation** — truy xuất nguồn gốc tới ER Diagram hoặc Activity Diagram cụ thể. Guardrail **Source Citation Mechanism** đảm bảo: field không có source → BLOCK, không ghi file.

Hình 3.4 minh họa Class Diagram của module M1 (Auth & Profile), thể hiện cấu trúc collection `User` với đầy đủ fields, visibility modifiers và hooks.



Hình 3.4: Class Diagram: Collection User (M1 — Auth & Profile)

3.2.6 Kết quả sơ đồ quan hệ thực thể (ER Diagram)

Sơ đồ quan hệ thực thể tổng quan mô tả **13 bộ sưu tập** (thực thể — *entities*) với đầy đủ thuộc tính và quan hệ giữa chúng. Sơ đồ được thiết kế ở mức **Lược đồ vật lý** (*Physical Schema*) — cung cấp các đặc tả có thể sử dụng trực tiếp để triển khai các bộ sưu tập (*collections*) trên hệ quản trị nội dung *Payload CMS*.

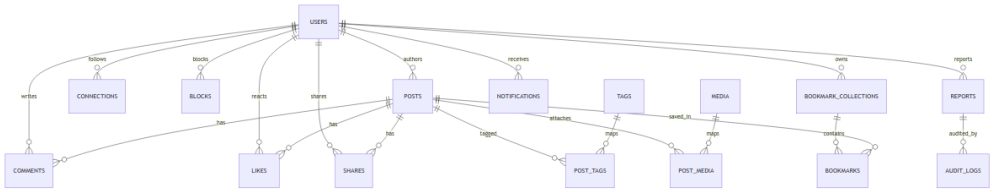
Bảng 3.8: Danh sách 13 bộ sưu tập dữ liệu trong sơ đồ quan hệ thực thể

STT	Bộ sưu tập (*Collection*)	Phân hệ	Mô tả
1	users	M1	Tài khoản và hồ sơ người dùng
2	posts	M2	Bài viết (văn bản, hình ảnh, liên kết)
3	media	M2	Tệp tin đa phương tiện
4	tags	M2	Nhãn phân loại nội dung
5	comments	M4	Bình luận (hỗ trợ phản hồi phân cấp — *threaded replies*)
6	likes	M4	Lượt thích (duy nhất cho mỗi cặp người dùng và bài viết)
7	shares	M4	Chia sẻ (sao chép liên kết hoặc đăng lại)
8	connections	M4	Quan hệ theo dõi/bỏ theo dõi
9	bookmark_collections	M5	Bộ sưu tập bài viết đã lưu
10	bookmarks	M5	Chi tiết bài viết đã lưu
11	notifications	M6	Thông báo (thiết kế đa hình — *polymorphic*)
12	reports	M6	Báo cáo vi phạm nội dung
13	audit_logs	M6	Nhật ký hệ thống (chỉ cho phép ghi thêm — *append-only*)

Các quan hệ chính bao gồm: `USERS ||--o{ POSTS` (1 user → nhiều posts), `POSTS ||--o{ COMMENTS` (1 post → nhiều comments), `USERS ||--o{ CONNECTIONS` (follow/unfollow), `BOOKMARK_COLLECTIONS ||--o{ BOOKMARKS` (1 collection → nhiều bookmarks), và `REPORTS ||--o{`

AUDIT_LOGS (1 report → nhiều audit entries).

Hình 3.5 minh họa ER Diagram tổng quan với toàn bộ 13 entities và quan hệ giữa chúng.



Hình 3.5: Sơ đồ quan hệ thực thể (ER Diagram) tổng quan hệ thống Steve Void

Bộ kỹ năng thiết kế lược đồ thực hiện chuyển đổi từ sơ đồ lớp (Hợp đồng YAML) sang **Lược đồ cơ sở dữ liệu vật lý** (*Physical Database Schema*) cho *MongoDB* và *Payload CMS*. Kết quả thu được bao gồm **12 tệp tin** (6 tệp định dạng Markdown và 6 tệp định dạng YAML) dưới dạng kết quả định dạng đôi.

Các mẫu thiết kế cơ sở dữ liệu (*Design patterns*) áp dụng

Bảng 3.9: Các mẫu thiết kế cơ sở dữ liệu sử dụng trong lược đồ vật lý

Mẫu thiết kế (*Pattern*)	**Mô tả và ứng dụng cụ thể**
Gốc tập hợp (*Aggregate Root*)	Các bộ sưu tập được tách riêng biệt dựa trên tần suất truy cập và quy mô dữ liệu (bài viết, bình luận, lượt thích được lưu trữ độc lập).
Phản chuẩn hóa (*Denormalization*)	Các trường bộ đếm (tổng lượt thích, lượt bình luận, lượt theo dõi) được lưu trực tiếp tại thực thể chính và cập nhật qua các điểm can thiệp dữ liệu (*hooks*).
Mô hình tính toán (*Computed Pattern*)	Các trường chỉ đóng vai trò đọc dữ liệu (*readOnly*) được cập nhật tự động thông qua các hàm xử lý sau khi thay đổi (*afterChange hooks*).
Mô hình đa hình (*Polymorphic Pattern*)	Các phân hệ thông báo và báo cáo sử dụng cặp định danh (loại thực thể + mã thực thể) để tham chiếu linh hoạt tới nhiều nguồn dữ liệu khác nhau.
Trường liên kết ảo (*Virtual Join Fields*)	Sử dụng cơ chế mở rộng chiều sâu (*depth expansion*) của *Payload Local API* để tự động nạp các dữ liệu quan hệ một chiều.
Chỉ cho phép ghi thêm (*Append-Only*)	Nhật ký hệ thống không cho phép thao tác sửa hoặc xóa, đảm bảo tính vẹn toàn của lịch sử truy vết (*audit trail*).
Chỉ mục hỗn hợp (*Compound Index*)	Áp dụng các ràng buộc duy nhất trên nhiều trường dữ liệu kết hợp để đảm bảo tính toàn vẹn (ví dụ: cặp mã bài viết và mã người dùng cho lượt thích).

Bảng 3.10 minh họa cấu trúc chi tiết của bộ sưu tập `users` — đây là thành phần cốt lõi của hệ thống, được khởi tạo tự động bởi bộ kỹ năng thiết kế lược đồ vật lý.

Bảng 3.10: Lược đồ chi tiết của bộ sưu tập người dùng `users` (M1)

Trường dữ liệu	**Kiểu Payload**	**Kiểu Mongo**	**Chỉ mục**	**Ghi chú**
email	email	String	Có	Duy nhất, đã xác thực
username	text	String	Có	3–50 ký tự, duy nhất
passwordHash	text	String	Không	Mã băm (*bcrypt*), bảo mật
verified	checkbox	Boolean	Không	Giá trị mặc định: *false*
role	select	String	Không	thành viên, quản trị viên
status	select	String	Không	hoạt động, tạm khóa, cấm
profileVisibility	select	String	Không	công khai, theo dõi, riêng tư
displayName	text	String	Không	Trường bắt buộc
bio	text	String	Không	Tối đa 160 ký tự
avatar	text	String	Không	Liên kết ảnh hoặc mã Media
followerCount	number	Number	Không	Trường tính toán, chỉ đọc
followingCount	number	Number	Không	Trường tính toán, chỉ đọc

Trường dữ liệu	Kiểu Payload	Kiểu Mongo	Chỉ mục	Ghi chú
createdAt	date	ISODate	Có	Khởi tạo tự động
updatedAt	date	ISODate	Không	Cập nhật tự động

Hệ thống Access Control

Thiết kế lược đồ định nghĩa **5 cấp độ kiểm soát truy cập** (*Access Control*) cho mỗi bộ sưu tập dữ liệu:

- **Khách (*guest*):** Người dùng chưa xác thực — chỉ được phép đọc các dữ liệu công khai.
- **Thành viên (*member*):** Người dùng đã xác thực — được phép tạo mới nội dung và tương tác.
- **Chủ sở hữu (*owner*):** Người có quyền quản lý trực tiếp tài nguyên — được phép chỉnh sửa hoặc xóa nội dung của chính mình.
- **Quản trị viên (*admin*):** Người quản lý hệ thống — có quyền xử lý các báo cáo vi phạm và cấm người dùng.
- **Hệ thống (*system*):** Các tiến trình tự động nội bộ — thực hiện cập nhật các bộ đếm và xử lý dữ liệu qua *hooks*.

Kiểm chứng chất lượng

Toàn bộ 6 modules schema đã vượt qua kiểm chứng bởi script `validate_schema.py`:

- 6/6 modules có Contract YAML hợp lệ
- 6/6 modules có Schema Markdown hoàn chỉnh

- 6/6 modules có Schema YAML chuẩn hóa
- **0 trường ảo giác** (hallucination) — mọi field đều có nguồn từ Contract YAML

3.2.7 Kết quả thiết kế giao diện người dùng

Bộ kỹ năng phân tích kiến trúc giao diện đã khởi tạo **8 tệp tin đặc tả giao diện** (*UI Specs*), bao gồm đặc tả cho 6 phân hệ (M1–M6), một biểu mẫu chuẩn và một tệp chỉ mục. Mỗi bản đặc tả thực hiện ánh xạ từ các trường dữ liệu trong lược đồ sang các thành phần giao diện thực tế (*UI Components*) dựa trên nguyên tắc **Tuyệt đối không bịa đặt nội dung** (*Zero Hallucination*) — các yếu tố giao diện chỉ được phép tồn tại nếu đã được định nghĩa trong cấu trúc dữ liệu.

Cấu trúc của mỗi bản đặc tả giao diện bao gồm:

- Danh sách các màn hình (*Screens*) kèm theo mã định danh duy nhất (ví dụ: `m1/register`).
- Bảng ánh xạ thành phần (*Component Map*) — định nghĩa kiểu thành phần giao diện cho từng trường dữ liệu.
- Các trạng thái giao diện (*States*) — bao gồm trạng thái mặc định, đang tải, lỗi, thành công và dữ liệu trống.
- Các điểm ngắt phản hồi (*Responsive breakpoints*) — hỗ trợ đa thiết kế cho di động (375px), máy tính bảng (768px) và máy tính để bàn (1280px).

Bảng 3.11: Số lượng màn hình theo module trong UI Specs

Module	Tên spec	Số screens
M1	m1-auth-ui-spec.md	5
M2	m2-content-ui-spec.md	3
M3	m3-discovery-ui-spec.md	3
M4	m4-engagement-ui-spec.md	3
M5	m5-bookmarking-ui-spec.md	2
M6	m6-notifications-ui-spec.md	2
Tổng		**18**

Wireframe Blueprints

Bộ kỹ năng vẽ tự động đã khởi tạo **17 bản phác thảo bản vẽ chi tiết** (*Wireframe Blueprints*), bao gồm 15 bản vẽ mức màn hình và 2 biến thể dành cho máy tính xách tay. Mỗi bản vẽ phác thảo tuân thủ cấu trúc quy chuẩn:

- **Tiêu đề (*Header*):** Chứa các thông tin mô tả về phân hệ, tài liệu tham chiếu, bố cục, kích thước và trạng thái của màn hình.
- **Phân vùng (*Sections*):** Phân chia không gian giao diện theo các mẫu bố cục (*composition pattern*) đã định trước.
- **Thành phần (*Components*):** Danh sách các thành phần giao diện kèm mã tham chiếu (*ref ID*) từ thư viện chung và trích dẫn đặc tả (*spec-cite*) để phục vụ truy vết nguồn gốc.
- **Trạng thái (*States*):** Mô tả các biểu hiện của giao diện người dùng (mặc

định, lỗi, đang tải, thành công).

- **Ghi chú (*Notes*):** Trình bày các suy luận và quyết định thiết kế (*inference and decisions*) của AI trong quá trình vẽ.

Bảng 3.12: Wireframe blueprints theo module

Module	Số WF	Các màn hình
M1	5	Register, Email Verification, Profile Setup, User Profile, Profile Edit
M2	2	Create Post, Post Detail/Edit
M3	2	News Feed, Discover/Trending
M4	2	Comment Thread, User Connections/Followers
M5	2	Bookmark Collections, Bookmarks List
M6	2	Notifications Feed, Reports Admin Panel
Laptop	2	Desktop Auth, Desktop Feed

Đặc biệt, mỗi component trong wireframe đều có thuộc tính `spec-cite` trỏ tới section cụ thể trong UI Spec (ví dụ: `[spec §2.1 - input-email]`). Guardrail **G-Spec-Strict** của UI Pencil Drawer đảm bảo không thêm element nào mà spec không đề cập.

3.3 Đánh giá hệ thống

3.3.1 Đánh giá về độ bao phủ (Coverage)

Bảng 3.13: Ma trận bao phủ: Module × Loại sơ đồ

Module	UC	AD	SD	FD	CD	Schema	UI
M1	✓	✓	✓	✓	✓	✓	✓
M2	✓	✓	✓	✓	✓	✓	✓
M3	✓	✓	✓	✓	✓	✓	✓
M4	✓	✓	✓	✓	✓	✓	✓
M5	✓	✓	✓	✓	✓	✓	✓
M6	✓	✓	✓	✓	✓	✓	✓

Chú thích: UC = Use Case, AD = Activity Diagram, SD = Sequence Diagram, FD = Flow Diagram, CD = Class Diagram.

Kết quả cho thấy **100% modules** (6/6) được bao phủ đầy đủ bởi **tất cả 7 loại artifacts**. Không có module nào bị thiếu sót về mặt thiết kế.

3.3.2 Đánh giá về tính nhất quán (Consistency)

Chuỗi truy xuất nguồn gốc (*Traceability Chain*) giữa các thành phẩm kỹ thuật được đảm bảo tính liên tục xuyên suốt toàn bộ vòng đời:

Yêu cầu hệ thống (FR-1 → FR-10)

↓

Sơ đồ ca sử dụng (UC01 → UC24)

↓

Sơ đồ hoạt động (M1-A1 → M6-A3)

↓

Sơ đồ tuần tự và lưu đồ tuần tự

↓

Sơ đồ lớp (Sơ đồ Mermaid + Hợp đồng YAML)

↓

Thiết kế lược đồ (Bộ sưu tập vật lý trên MongoDB)

↓

Đặc tả giao diện → Bản vẽ phác thảo

Mỗi tầng kiến trúc được xây dựng dựa trên nền tảng của tầng trước đó, tạo thành một hệ thống đặc tả, thiết kế nhất quán và sẵn sàng cho quá trình triển khai thực tế. Cụ thể:

- **93 trường dữ liệu** trong Class Diagram đều có source citation tới ER Diagram hoặc Activity Diagram
- **24 luồng nghiệp vụ** trong Flow Diagram đều có UC-ID truy xuất tới Use Case
- **0 trường ảo giác** (hallucination) trong toàn bộ Schema Design — mọi field đều có nguồn từ Contract YAML
- Tên đối tượng trong Sequence Diagram khớp 100% với codebase dự kiến

3.3.3 Đánh giá về chất lượng kỹ thuật

Hiệu quả của Guardrails chống ảo giác

Hệ thống Guardrails đã chứng minh hiệu quả trong việc ngăn chặn hallucination:

Bảng 3.14: Hiệu quả Guardrails trong quá trình triển khai

Guardrail	Skill áp dụng	Kết quả
Source Citation	Class Diagram Analyst	93/93 fields có nguồn (100%)
Contract-Only	Schema Design Analyst	0 field ảo giác trong 6 modules
No Blind Step	Flow Design Analyst	100% action nodes có căn cứ từ spec
UC-ID Mapping	Flow Design Analyst	24/24 use cases được ánh xạ
Spec-Strict	UI Pencil Drawer	100% components có spec-cite

Các design patterns đã áp dụng thành công

Quá trình triển khai đã áp dụng thành công nhiều patterns quan trọng cho MongoDB/Payload CMS:

1. **Aggregate Root Strategy**: Tách riêng collections theo tần suất truy cập, tránh vi phạm giới hạn 16MB document
2. **Denormalized Counters**: Tối ưu hiệu năng đọc bằng cách lưu trực tiếp counter fields, cập nhật qua hooks
3. **Polymorphic References**: Hỗ trợ tham chiếu linh hoạt giữa các entity types khác nhau
4. **Append-Only Audit Trail**: Đảm bảo tính toàn vẹn của nhật ký hành chính
5. **Soft-Delete Strategy**: Sử dụng trường `status` thay vì xóa cứng, bảo toàn dữ liệu

3.3.4 Đánh giá về khả năng sẵn sàng cho giai đoạn triển khai (Life-3)

Toàn bộ artifacts thiết kế đã đạt trạng thái **sẵn sàng triển khai**:

- 13 MongoDB collections có schema chi tiết tới từng field, bao gồm kiểu dữ liệu, constraints, indexes và hooks
- 18 màn hình UI có wireframe blueprint với component ref IDs từ Lib-Component
- Dual-format output (Markdown + YAML) cho phép cả con người review lẫn AI Agent đọc để sinh code tự động
- Hệ thống access control 5 cấp đã được định nghĩa cho mọi collection
- Hooks specification (beforeChange, afterChange) đảm bảo tính nhất quán dữ liệu khi triển khai

3.3.5 Hạn chế và bài học kinh nghiệm

Trong quá trình triển khai, một số hạn chế đã được nhận diện:

1. **Phụ thuộc vào context window**: Với các module lớn (M1, M4), AI Agent đôi khi cần chia nhỏ task để tránh tràn context, ảnh hưởng đến tính liên tục của output
2. **Wireframe chưa bao gồm responsive đầy đủ**: Chỉ có 2/17 wireframes có device variant (laptop), các wireframe còn lại chủ yếu thiết kế cho mobile (375px)
3. **Chưa có automated testing**: Việc kiểm chứng chủ yếu dựa vào checklist thủ công và script `validate_schema.py`, chưa có bộ test tự động hoàn chỉnh

4. **Module M3 không tạo collection vật lý**: FeedQuery chỉ là ValueObject (DTO), không lưu xuống database — đây là quyết định thiết kế hợp lý nhưng cần lưu ý khi triển khai

Bài học rút ra: Hệ thống Guardrails và Interaction Points đã chứng minh vai trò then chốt trong việc duy trì chất lượng đầu ra. Nguyên tắc **Source Citation** và **Contract-Only** là hai cơ chế hiệu quả nhất để chống ảo giác AI trong quá trình thiết kế hệ thống.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

3.4 Kết quả đạt được

Qua quá trình nghiên cứu và triển khai, khóa luận đã đạt được các kết quả chính sau:

1. **Xây dựng thành công Khung kỹ năng tác vụ (*Agent Skill Framework*):** Đề xuất kiến trúc Ba trụ cột (*3 Pillars*) kết hợp Bảy vùng chức năng (*7 Zones*) để tổ chức tri thức cho tác vụ AI một cách khoa học. Mô hình này thiết lập các cơ chế kiểm soát chặt chẽ, giảm thiểu hiện tượng ảo giác thông qua các cơ chế Trích dẫn nguồn, Chỉ làm theo hợp đồng và Các điểm tương tác (*Interaction Points*).

2. **Phát triển bộ 10 kỹ năng tác vụ chuyên biệt:** Bao gồm 3 kỹ năng điều phối (*Meta-Skills*) và 7 kỹ năng chuyên môn (*Domain Skills*). Mỗi kỹ năng được trang bị quy trình thực thi định sẵn, các hàng rào bảo vệ chống bịa đặt và cơ chế tự kiểm chứng chất lượng đầu ra.

3. **Khởi tạo hơn 120 thành phẩm kỹ thuật cho hệ thống Steve Void:** Toàn bộ 6 phân hệ chức năng đều được bao phủ bởi hệ thống tài liệu thiết kế chi tiết, từ sơ đồ tác vụ, hoạt động, tuần tự cho đến thiết kế cơ sở dữ liệu vật lý và các bản vẽ phác thảo giao diện tự động.

4. **Đảm bảo tính nhất quán và khả năng truy vết:** Thiết lập chuỗi truy xuất nguồn gốc (*Traceability Chain*) xuyên suốt từ yêu cầu nghiệp vụ cho đến bản vẽ giao diện cuối cùng. Mọi trường dữ liệu và bước xử lý đều có căn cứ xác thực từ các tài liệu đặc tả ở tầng cao hơn.

5. **Chứng minh tính khả thi của việc ứng dụng tác vụ AI vào quy**

trình phần mềm: AI không chỉ dừng lại ở việc hỗ trợ viết mã (*code generation*), mà còn có khả năng tham gia trực tiếp vào các giai đoạn phân tích nghiệp vụ và thiết kế kiến trúc — những công đoạn đòi hỏi tính logic cao và tư duy hệ thống phức tạp.

3.5 Hạn chế của nghiên cứu

Bên cạnh các kết quả đạt được, nghiên cứu vẫn tồn tại một số hạn chế cần được nhìn nhận khách quan:

1. **Hạn chế về kinh phí sử dụng tài nguyên AI:** Việc triển khai các kỹ năng tác vụ đòi hỏi sử dụng giao diện lập trình của các mô hình ngôn ngữ lớn (*LLMs API*) với chi phí đáng kể. Giới hạn ngân sách khiến số lần thử nghiệm và tinh chỉnh bị hạn chế, ảnh hưởng đến độ hoàn thiện của một số kỹ năng. Ngoài ra, giới hạn cửa sổ ngữ cảnh (*context window*) của mô hình cũng gây khó khăn khi xử lý các phân hệ lớn, đòi hỏi phải chia nhỏ công việc và làm giảm tính liên tục của kết quả đầu ra.

2. **Hạn chế về thời gian và kiến thức nghiệp vụ:** Do thời gian thực hiện khóa luận có giới hạn, kiến thức nghiệp vụ được đưa vào các kỹ năng tác vụ (đặc biệt ở tầng Tri thức — *Knowledge*) mới chỉ đáp ứng ở mức độ cơ bản. Các bộ tệp tri thức chưa đủ sâu để bao phủ toàn bộ các trường hợp biên (*edge cases*) và các thực hành tốt nhất (*best practices*) của từng lĩnh vực (tiêu chuẩn UML, tối ưu hóa MongoDB, các mẫu thiết kế UI/UX). Điều này dẫn đến chất lượng đầu ra của kỹ năng tuy đạt yêu cầu cho nghiên cứu nhưng chưa đủ độ tin cậy để đưa vào vận hành thực tế (*production*).

3. **Chưa hoàn thành giai đoạn Triển khai thực tế (*Life-3 — Implementation*):** Nghiên cứu hiện tại mới dừng lại ở giai đoạn Thiết

kế và Đặc tả (*Life-2 — Design & Specification*). Hệ thống Steve Void chưa được triển khai thành sản phẩm thực tế với mã nguồn có thể vận hành, do đó chưa thể đánh giá đầy đủ tính khả dụng của các thành phẩm thiết kế trong thực tiễn lập trình.

4. **Bản vẽ phác thảo chưa bao phủ đầy đủ khả năng hiển thị phản hồi (*responsive*):** Trong 17 bản vẽ phác thảo, chỉ có 2 bản có biến thể dành cho các loại thiết bị khác nhau (*device variant*) như máy tính xách tay. Phần lớn các bản vẽ được thiết kế ưu tiên cho thiết bị di động, chưa có bản thiết kế riêng cho máy tính bảng và máy tính để bàn một cách đầy đủ.

5. **Chưa có đánh giá định lượng từ người dùng thực tế:** Việc đánh giá chất lượng hiện tại chủ yếu dựa trên danh sách kiểm tra (*checklist*), các kịch bản xác minh (*script validation*) và tự đánh giá (*self-review*). Nghiên cứu chưa thực hiện được các khảo sát người dùng hoặc so sánh thực nghiệm có đối chứng (*controlled experiment*) giữa quy trình phát triển có và không có các kỹ năng tác vụ AI.

1. **Tự động hóa hoàn toàn chuỗi quy trình thiết kế (*Fully Autonomous Execution*):** Đây là hướng phát triển trọng tâm, hướng tới mục tiêu xây dựng một hệ thống mà người dùng chỉ cần cung cấp các tài liệu định hướng ban đầu. Từ đó, các tác vụ AI sẽ tự động thực thi toàn bộ chuỗi thiết kế từ sơ đồ nghiệp vụ đến bản vẽ phác thảo mà không cần sự can thiệp của con người. Các điểm tương tác hiện tại sẽ được thay thế bằng cơ chế tự phê duyệt với ngưỡng tin cậy (*confidence threshold*) và chỉ chuyển giao quyền quyết định cho con người khi phát hiện rủi ro hoặc mâu thuẫn logic.

2. **Mở rộng sang giai đoạn tự động sinh mã (*Automated Code Generation*):** Kết nối kết quả thiết kế với giai đoạn xây dựng thực tế để

tự động khởi tạo cấu trúc dữ liệu, các thành phần giao diện và các bộ xử lý giao diện lập trình (*API route handlers*). Mục tiêu hướng tới là một quy trình khép kín từ ý tưởng đến sản phẩm hoàn thiện (*end-to-end*).

3. **Nâng cao chất lượng kho tri thức (*Knowledge Base*)**: Bổ sung và tinh chỉnh các tệp tri thức trong từng kỹ năng tác vụ với kiến thức chuyên sâu hơn: Các mẫu thiết kế *MongoDB* nâng cao (phân mảnh dữ liệu — *sharding*, đường ống tổng hợp dữ liệu — *aggregation pipelines*), các tiêu chuẩn tuân thủ UML (đặc tả *OMG*), và các quy tắc giao diện tối ưu từ *Nielsen Norman Group*. Điều này giúp nâng cao chất lượng đầu ra từ mức cơ bản lên mức sẵn sàng vận hành (*production-ready*).

4. **Tích hợp đa mô hình AI (Multi-Model Architecture):** Thay vì phụ thuộc vào một mô hình duy nhất (Claude), nghiên cứu hướng tới việc tích hợp nhiều mô hình AI khác nhau (GPT-4, Gemini, open-source models) cho từng loại task cụ thể, nhằm tối ưu chi phí và tận dụng thế mạnh riêng của mỗi mô hình.

5. **Xây dựng hệ thống đánh giá tự động (Automated Quality Assessment):** Phát triển bộ test tự động để đánh giá chất lượng đầu ra của Agent Skills: kiểm tra tính đúng đắn của sơ đồ (syntax validation), tính nhất quán giữa các artifacts (cross-reference checking), và so sánh với benchmark từ các dự án thực tế.

6. **Đóng gói thành sản phẩm mã nguồn mở**: Công bố bộ khung kỹ năng tác vụ (*Agent Skill Framework*) dưới dạng thư viện mã nguồn mở, đi kèm tài liệu hướng dẫn chi tiết và bộ kỹ năng mẫu để cộng đồng lập trình viên Việt Nam có thể tái sử dụng và cùng nhau phát triển.

Tóm lại, nghiên cứu đã chứng minh được tiềm năng to lớn của việc ứng dụng AI Chatbot trong phát triển phần mềm, đặc biệt ở giai đoạn phân tích và

thiết kế. Mặc dù còn những hạn chế về kinh phí, thời gian và độ sâu nghiệp vụ, kết quả đạt được đã đặt nền móng vững chắc cho một hướng đi đầy triển vọng — nơi AI không chỉ là công cụ hỗ trợ mà trở thành **cộng tác viên thực sự** trong quy trình phát triển phần mềm.

TÀI LIỆU THAM KHẢO

TÀI LIỆU THAM KHẢO

[1] Andrew Ng, ”Agentic Workflow and AI Agents”, DeepLearning.AI, 2024.

[2] Next.js Documentation, URL: `https://nextjs.org/docs`, truy cập tháng 02/2026.

[3] Payload CMS Documentation, URL: `https://payloadcms.com/docs`, truy cập tháng 02/2026.

[4] MongoDB Atlas Documentation, URL: `https://www.mongodb.com/docs/atlas`, truy cập tháng 02/2026.

[5] Radix UI Documentation, URL: `https://www.radix-ui.com/docs`, truy cập tháng 02/2026.

[6] Anthropic, ”Building effective agents”, URL: `https://www.anthropic.com/research/building-effective-agents`, truy cập tháng 02/2026.

[7] Stuart Russell, Peter Norvig, *Artificial Intelligence: A Modern Approach*, 4th Edition, Pearson, 2021.

[8] Ashish Vaswani et al., ”Attention Is All You Need”, *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[9] Anthropic, ”Claude Code — CLI for Claude”, URL: `https://docs.anthropic.com/en/docs/claude-code`, truy cập tháng 02/2026.

[10] Object Management Group (OMG), ”Unified Modeling Language

Specification, Version 2.5.1", URL: https://www.omg.org/spec/UML, 2017.

[11] Mermaid.js Documentation, URL: https://mermaid.js.org/intro/, truy cập tháng 02/2026.

[12] Roger S. Pressman, Bruce R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th Edition, McGraw-Hill Education, 2020.

PHỤ LỤC

.1 Phụ lục A: Cấu trúc thư mục Agent Skill

Mỗi Agent Skill được tổ chức theo cấu trúc 7 Zone thống nhất. Dưới đây là ví dụ minh họa với skill `sequence-design-analyst`:

```
sequence-design-analyst/
|-- SKILL.md                        # Vùng 1: Đặc điểm cốt lõi (Core P
|                                   # (Persona, Quy trình, Hàng rào bả
|-- knowledge/                      # Vùng 2: Tri thức miền (Domain Kn
|   |-- uml-rules.md               # Chuẩn UML & cú pháp Mermaid
|   |-- project-patterns.md        # Kiến trúc dự án Payload CMS
|   +-- sequence-preparation.md # Hướng dẫn phân tích kịch bản
|-- templates/                      # Vùng 3: Biểu mẫu đầu ra (Output T
|   |-- auth-flow.mmd              # Mẫu luồng xác thực
|   |-- crud-flow.mmd              # Mẫu luồng thao tác dữ liệu cơ bản
|   +-- logic-flow.mmd             # Mẫu luồng logic nghiệp vụ
|-- loop/                           # Vùng 4: Kiểm soát chất lượng (Qu
|   |-- checklist.md               # Danh sách kiểm tra chất lượng
|   +-- verify-script.py           # Kịch bản tự động xác minh
+-- scripts/                        # Vùng 5: Kịch bản tự động hóa (Au
```

Cấu trúc này đảm bảo mỗi skill hoạt động như một đơn vị độc lập, tự chứa đầy đủ kiến thức miền, quy trình làm việc và cơ chế kiểm soát chất lượng.

Phụ lục B: Trích mẫu SKILL.md — Sequence Design Analyst

Dưới đây là phần trích dẫn đầu file `SKILL.md` của skill phân tích Sequence Diagram, minh họa cách định nghĩa Persona, Boot Sequence và Workflow:

```
---
name: sequence-design-analyst
description: Chuyên gia phân tích và thiết kế sơ đồ tuần tự
  (UML) chuẩn Mermaid. Tự động nghiên cứu mã nguồn để
  đảm bảo tính thực tế.
---
# Sequence Design Analyst

- **Đặc điểm (Persona):** Kiến trúc sư UML cấp cao. Thiết kế sơ đồ
  tuần tự phản ánh chính xác 100% logic triển khai thực tế.

## Trình tự khởi động bắt buộc (Mandatory Boot Sequence)
1. Đọc knowledge/uml-rules.md
2. Đọc knowledge/project-patterns.md
3. Đọc knowledge/sequence-preparation.md
4. Đọc loop/checklist.md

## Quy trình: Các giai đoạn thực thi (Workflow: Execution Phases)

### Giai đoạn 1: Khám phá kịch bản (Scenario Discovery)
- Xác định Kịch bản chính, Các tác nhân, Các đường đời (Lifelines)
- [ĐIỂM TƯƠNG TÁC] - Chờ xác nhận kịch bản từ người dùng
```

```
### Giai đoạn 2: Nghiên cứu mã nguồn (Codebase Research)
- Tìm kiếm các hàm xử lý liên quan trong Dịch vụ/Bộ sưu tập
- Xây dựng chuỗi gọi hàm dựa trên các Mẫu thiết kế dự án
- [ĐIỂM TƯƠNG TÁC] - Xác nhận luồng thực thi các hàm

### Giai đoạn 3: Thiết kế luồng thông điệp (Message Flow Design)
- Sắp xếp các kết quả tìm kiếm thành chuỗi tuần tự logic
- Thiết kế các phân đoạn: rẽ nhánh (alt), lặp (loop), tùy chọn (opt
- Sử dụng tên hàm thực tế trong mã nguồn (ví dụ: payload.find)

### Giai đoạn 4: Khởi tạo (Generation)
- Kết xuất mã Mermaid theo đúng biểu mẫu quy định
- Xác minh kết quả so đối chiếu với danh sách kiểm tra (loop/checkl
```

Phụ lục C: Cấu trúc sản phẩm đầu ra Life-2

Toàn bộ sản phẩm đầu ra của giai đoạn Life-2 (Thiết kế) được tổ chức theo cấu trúc sau:

```
Docs/life-2/
|-- api/
|   +-- api-spec.md              # Đặc tả API (42 điểm cuối)
|-- database/
|   +-- schema-design/           # Thiết kế cơ sở dữ liệu
|       |-- schema-m1.md + .yaml # M1: Xác thực & Hồ sơ
|       |-- schema-m2.md + .yaml # M2: Công cụ Nội dung
|       |-- schema-m3.md + .yaml # M3: Khám phá & Bảng tin
|       |-- schema-m4.md + .yaml # M4: Tương tác
|       |-- schema-m5.md + .yaml # M5: Lưu trữ bài viết
```

```
|       +-- schema-m6.md + .yaml # M6: Thông báo
|-- diagrams/
|   |-- activity-diagrams/       # 24 Sơ đồ hoạt động
|   |-- class-diagrams/          # 7 Sơ đồ lớp
|   |-- flow/                    # 11 Sơ đồ luồng nghiệp vụ
|   |-- sequence-diagrams/       # 24 Sơ đồ tuần tự
|   +-- UseCase/                 # 6 Sơ đồ ca sử dụng
|-- specs/
|   |-- m1-auth-profile-spec.md  # Đặc tả phân hệ M1
|   |-- m2-content-engine-spec.md # Đặc tả phân hệ M2
|   |-- m3-discovery-feed-spec.md # Đặc tả phân hệ M3
|   |-- m4-engagement-spec.md    # Đặc tả phân hệ M4
|   |-- m5-bookmarking-spec.md   # Đặc tả phân hệ M5
|   +-- m6-notifications-spec.md # Đặc tả phân hệ M6
+-- ui/
    |-- specs/                   # 8 Tệp đặc tả giao diện
    +-- wireframes/              # 27 Tệp bản vẽ phác thảo
```

Tổng cộng 122 tệp sản phẩm được sinh tự động bởi hệ thống Agent Skill, bao gồm cả định dạng Markdown (cho con người) và YAML (cho máy đọc).

Phụ lục D: Mã nguồn Mermaid mẫu

D.1. Class Diagram — Collection User (M1)

```
classDiagram
    namespace AuthProfile {
        class User {
            <<Collection>>
```

```
            +ObjectId id
            +String email
            +String username
            -String passwordHash
            +Boolean verified
            +String role
            +String status
            +String profileVisibility
            +String displayName
            +String bio
            +String avatar
            +Number followerCount
            +Number followingCount
            +DateTime createdAt
            +DateTime updatedAt
            +void beforeChange_sanitizeInput()
            +void afterChange_sendVerification()
        }
    }
```

D.2. Sequence Diagram — Luồng Đăng ký tài khoản

```
sequenceDiagram
    actor User
    participant RegisterPage
    participant AuthService
    participant Mailer
    participant Payload
    participant DB
```

```
User->>RegisterPage: Điền Form (Email, Password)
RegisterPage->>AuthService: register(data)
activate AuthService
AuthService->>Payload: payload.find({
    collection: 'users', email })
Payload-->>AuthService: { docs: [] }

alt Email đã tồn tại
    AuthService-->>RegisterPage: { error }
    RegisterPage-->>User: Hiển thị lỗi
else Email chưa tồn tại
    AuthService->>Payload: payload.create({
        collection: 'users', data })
    activate Payload
    Payload->>DB: insertUser
    DB-->>Payload: userObj
    Payload-->>AuthService: userObj
    deactivate Payload
    AuthService->>Mailer: sendVerificationEmail()
    AuthService-->>RegisterPage: { success: true }
    deactivate AuthService
    RegisterPage-->>User: Yêu cầu kiểm tra email
end
```

Phụ lục E: Danh sách 10 Agent Skill tiêu biểu

Dưới đây là danh sách 10 Agent Skill tiêu biểu được thiết kế và triển khai thực tế để tự động hóa quy trình phân tích, thiết kế và triển khai phần

mềm trong nghiên cứu:

STT	Tên Skill	Chức năng và vai trò thực tế
1	**skill-architect**	Thiết kế kiến trúc tổng thể cho một kỹ năng tác vụ mới dựa trên mô hình Ba trụ cột (*3 Pillars*) và Bảy vùng chức năng (*7 Zones*).
2	**skill-planner**	Tự động phân rã bản thiết kế kiến trúc thành kế hoạch triển khai chi tiết (tệp `todo.md`) với các tác vụ có tính truy xuất nguồn gốc (*traceability*).
3	**skill-builder**	Đóng vai trò kỹ sư triển khai, trực tiếp viết mã nguồn, cấu hình các tệp tri thức và kiểm soát chất lượng qua cơ chế Nhật ký-Thông báo-Dừng (*Log-Notify-Stop*).
4	**ui-pencil-drawer**	Tự động hóa quá trình vẽ bản vẽ phác thảo (*Wireframe*) trên vùng làm việc của công cụ *Pencil* từ các tệp đặc tả giao diện người dùng (*UI Spec*).
5	**flow-design-analyst**	Phân tích các quy tắc nghiệp vụ phức tạp từ câu chuyện người dùng để thiết kế sơ đồ luồng quy trình (*Flow Diagram*) theo chuẩn làn bơi (*Swimlane*).
6	**class-diagram-analyst**	Phân tích và thiết kế cấu trúc dữ liệu cho *MongoDB/Payload CMS*, định nghĩa các Hợp đồng dữ liệu (*YAML Contract*) chuẩn xác và nhất quán.
7	**sequence-design-analyst**	Thiết kế các sơ đồ tuần tự (*Sequence Diagram*) mô tả chi tiết sự tương tác đồng bộ và bất đồng bộ giữa các thành phần hệ thống.

STT	Tên Skill	Chức năng và vai trò thực tế
8	**ui-architecture-analyst**	Công cụ phân tích trung gian giúp chuyển đổi sơ đồ luồng và cấu trúc dữ liệu thành tài liệu đặc tả các thành phần giao diện (*UI Specs*).
9	**api-from-ui**	Tự động hóa việc xây dựng các điểm cuối giao diện lập trình tùy chỉnh (*Custom API Endpoints*) từ giao diện người dùng và đồng bộ kiểu dữ liệu.
10	**build-crud-admin-page**	Tự động hóa việc xây dựng các trang quản trị (*Admin Dashboard*) với đầy đủ tính năng thao tác dữ liệu cơ bản (*CRUD*), lọc và phân trang.