TRƯỜNG ĐẠI HỌC VINH
VIỆN KỸ THUẬT VÀ CÔNG NGHỆ

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

ĐỀ TÀI

XÂY DỰNG HỆ THỐNG THƯƠNG MẠI ĐIỆN TỬ TÍCH HỢP TRỢ LÝ ẢO MULTI-AGENT VÀ GỢI Ý CÁ NHÂN HÓA

Giảng viên hướng dẫn: PGS. TS. Hoàng Hữu Việt

Sinh viên thực hiện: Đặng Ngọc Anh

MSSV: 215748020110333 Lớp: 62K6

Nghệ An, 25/12/2025

LỜI MỞ ĐẦU

Trong kỷ nguyên của nền kinh tế số và cuộc cách mạng công nghiệp 4.0, thương mại điện tử đã không còn đơn thuần là việc đưa hàng hóa lên nền tảng trực tuyến, mà đã trở thành một hệ sinh thái công nghệ phức tạp, nơi trải nghiệm người dùng đóng vai trò quyết định sự sinh tồn của doanh nghiệp. Sự bùng nổ của dữ liệu và nhu cầu mua sắm trực tuyến tăng cao đã đặt ra những thách thức mới.

Trước thực tế đó, việc ứng dụng trí tuệ nhân tạo đã trở thành một xu hướng tất yếu. Đề tài **"*Xây dựng hệ thống thương mại điện tử tích hợp trợ lý ảo Multi-Agent và hệ thống gợi ý cá nhân hóa*"** được thực hiện nhằm mục tiêu nghiên cứu và phát triển một nền tảng mua sắm thông minh thế hệ mới. Điểm khác biệt cốt lõi của đề tài nằm ở việc xây dựng hệ thống trợ lý ảo dựa trên kiến trúc đa tác tử, giúp phân hóa nhiệm vụ xử lý ngôn ngữ tự nhiên một cách tối ưu, kết hợp cùng thuật toán gợi ý lai để mang lại những trải nghiệm cá nhân hóa sâu sắc cho từng khách hàng.

Để hoàn thành tốt đồ án này, em đã nhận được sự hướng dẫn vô cùng tận tình và tâm huyết từ thầy **PGS. TS. Hoàng Hữu Việt**. Thầy không chỉ là người định hướng về mặt chuyên môn, mà còn truyền dạy những kinh nghiệm thực tiễn quý báu để biến những thuật toán lý thuyết thành các sản phẩm có khả năng ứng dụng thực tế. Những ý kiến đóng góp, sự phân tích khắt khe nhưng đầy định hướng của thầy đã giúp em vượt qua nhiều rào cản về kỹ thuật và hoàn thiện tốt sản phẩm của mình. Em cũng xin gửi lời cảm ơn chân thành đến các thầy cô trong Viện Kỹ thuật và Công nghệ cùng gia đình, bạn bè đã luôn đồng hành, khích lệ em trong suốt quá trình học tập.

Mặc dù đã dành nhiều tâm huyết và nỗ lực để hoàn thiện đồ án, song do hạn chế về kinh nghiệm thực tế cũng như kiến thức chuyên môn còn đang trong quá trình tích lũy, báo cáo chắc chắn không tránh khỏi những thiếu sót nhất định. Em rất mong nhận được những ý kiến đóng góp, phê bình từ thầy hướng dẫn và các thầy cô trong Hội đồng để đồ án có thể hoàn thiện hơn, trở thành tiền đề cho những nghiên cứu chuyên sâu hơn trong tương lai.

Em xin chân thành cảm ơn!

Sinh viên thực hiện

Đặng Ngọc Anh

MỤC LỤC

LỜI MỞ ĐẦU **0**
MỤC LỤC **1**
DANH MỤC HÌNH **4**
DANH MỤC BẢNG **6**
DANH MỤC TỪ VIẾT TẮT **7**
CHƯƠNG 1. MỞ ĐẦU **8**
1.1. Bối cảnh và hiện trạng vấn đề 8
1.2. Các mô hình thương mại điện tử tích hợp AI tiêu biểu 9
1.2.1. Nike: Kiến trúc Composable Commerce 9
1.2.2. Sephora: Cá nhân hóa trải nghiệm mua sắm bằng trợ lý ảo AI 11
1.2.3. Thế giới di động: Tối ưu vận hành Omnichannel tại thị trường Việt Nam 11
1.3. Mục tiêu đề tài 12
1.4. Đối tượng và phạm vi đề tài 13
1.5. Ý nghĩa và giá trị thực tiễn đề tài 14
1.6. Bố cục báo cáo 14
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ 15
2.1. Kiến trúc TMĐT hiện đại và nền tảng MedusaJS 15
2.1.1. Khái niệm và lợi ích của kiến trúc Headless Commerce 15
2.1.2. Giới thiệu MedusaJS: Giải pháp Headless Commerce mã nguồn mở 17
2.1.3. Sự tương thích giữa kiến trúc của Medusa với hệ thống Multi-Agent 17
2.2. Next.js: Framework Frontend cho trải nghiệm TMĐT hiện đại 18
2.3. Nền tảng trí tuệ nhân tạo cho trợ lý ảo 18
2.3.1. Mô hình ngôn ngữ lớn (LLM) 18
2.3.2. Kỹ thuật Retrieval-Augmented Generation (RAG) 19
2.3.3. Kiến trúc đa tác tử (Multi-Agent System) 19

2.4. Nền tảng lý thuyết cho hệ thống gợi ý cá nhân hóa 20
2.4.1. Lọc dựa trên nội dung (Content-Based Filtering) 20
2.4.2. Lọc cộng tác (Collaborative Filtering) 20
2.4.3. Thuật toán lai (Hybrid Algorithm) 20
CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG 22
3.1. Phân tích yêu cầu 22
3.1.1. Yêu cầu chức năng 22
3.1.2. Yêu cầu phi chức năng 25
3.2. Biểu đồ ca sử dụng 26
3.3. Thiết kế kiến trúc hệ thống 29
3.3.1. Sơ đồ kiến trúc tổng thể 29
3.3.2. Sơ đồ luồng hoạt động của hệ thống Multi-Agent 32
3.3.3. Mô tả chi tiết quy trình xử lý nội bộ của các tác tử 34
3.3.4. Sơ đồ luồng hoạt động của hệ thống Recommendation 38
3.4. Thiết kế cơ sở dữ liệu 41
3.4.1. Tận dụng lược đồ cơ sở dữ liệu của Medusa 41
3.4.2. Thiết kế cơ sở dữ liệu cho Chatbot Service 41
3.4.3. Thiết kế cấu trúc dữ liệu cho Vector Database (RAG) 44
3.4.4. Thiết kế dữ liệu cho Recommendation Service 45
CHƯƠNG 4. TRIỂN KHAI VÀ KẾT QUẢ THỰC NGHIỆM 50
4.1. Chuẩn bị tài nguyên 50
4.1.1. Công cụ và công nghệ sử dụng 50
4.1.2. Thu thập và xử lý dữ liệu sản phẩm 51
4.2. Xây dựng Backend với MedusaJS 53
4.2.1. Khởi tạo và cấu hình Medusa Server 53

4.2.2. Triển khai các Module và dịch vụ 54
4.2.3. Quản lý dữ liệu nghiệp vụ 55
4.3. Xây dựng Frontend với Next.js 55
4.3.1. Khởi tạo và cấu trúc dự án Next.js 56
4.3.2. Tích hợp với Medusa Backend 56
4.3.3. Thiết kế giao diện người dùng (UI/UX) 57
4.3.4. Cấu hình SSR/SSG 59
4.4. Xây dựng Chatbot Multi-Agent Service 59
4.4.1. Agent 1: Input Processor (tác tử tiền xử lý) 60
4.4.2. Agent 2: Intent Classifier (tác tử phân loại ý định) 61
4.4.3. Agent 3: Orchestrator (tác tử điều phối) 62
4.4.4. Agent 4: Executor (tác tử thực thi) 64
4.4.5. Agent 5: Response Generator (tác tử tạo phản hồi) 64
4.5. Xây dựng Recommendation Service 67
4.6. Đánh giá kết quả thực nghiệm 72
4.6.1. Thiết lập kịch bản và bộ dữ liệu đánh giá 72
4.6.2. Đánh giá hiệu năng và độ chính xác của hệ thống Multi-Agent = ... 73
4.6.3. Đánh giá tính phù hợp của hệ thống gợi ý (Recommendation Service) 75
4.6.4. Tổng kết đánh giá thực nghiệm 78
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 80
PHỤ LỤC 83
Phụ lục 1: 83
Phụ lục 2: 83

DANH MỤC HÌNH

Hình 1.1: Giao diện trang Web Nike.com 10
Hình 1.2: Giao diện ứng dụng Nike Training Club 10
Hình 1.3: Giao diện trang Web sephora.com 11
Hình 1.4: Giao diện trang Web thegioididong.com 11
Hình 1.5: Chức năng hỗ trợ khách hàng qua Zalo 12
Hình 3.1: Sơ đồ tổng quan kiến trúc hệ thống 28
Hình 3.2: Sơ đồ tương tác giữa các thành phần 31
Hình 3.3: Sơ đồ luồng hoạt động của hệ thống Multi-Agent 33
Hình 3.4: Sơ đồ logic Agent Input Processor 34
Hình 3.5: Sơ đồ logic Agent Intent Classifier 35
Hình 3.6: Sơ đồ logic Agent Orchestrator 36
Hình 3.7: Sơ đồ logic Agent Executor 36
Hình 3.8: Sơ đồ logic Agent Response Generator 37
Hình 3.9: Sơ đồ luồng hoạt động của hệ thống Recommendation 40
Hình 3.10: Sơ đồ ERD cho cơ sở dữ liệu của Chatbot Service 44
Hình 3.11: Mô tả Payload mẫu dữ liệu lưu trữ Vector Database 45
Hình 3.12: Sơ đồ ERD cho cơ sở dữ liệu của Recommendation Service 48
Hình 4.1: Giao diện trang chủ MedusaJS 54
Hình 4.2: Đoạn mã cấu hình database và Redis trong medusa-config.js 54
Hình 4.3: Giao diện quản lý sản phẩm Admin UI MedusaJS 55
Hình 4.4: Giao diện thiết lập cài đặt Admin UI MedusaJS 55
Hình 4.5: Cấu trúc thư mục Next.js frontend 56
Hình 4.6: Giao diện trang chủ, sản phẩm nổi bật 57
Hình 4.7: Giao diện trang danh sách sản phẩm 57
Hình 4.8: Giao diện trang chi tiết sản phẩm 58
Hình 4.9: Giao diện trang giỏ hàng và thanh toán 58

Hình 4.10: Giao diện widget chatbot trợ lý ảo 59
Hình 4.11: Cấu trúc thư mục Chatbot Multi-Agent Service..... 60
Hình 4.12: Mã nguồn Agent Input Processor 61
Hình 4.13: Mã nguồn Agent Intent Classifier 62
Hình 4.14: Mã nguồn Agent Orchestrator 63
Hình 4.15: Mã nguồn Agent Executor..... 64
Hình 4.16: Mã nguồn Agent Response Generator..... 65
Hình 4.17: Chatbot trả lời câu hỏi về sản phẩm của người dùng 66
Hình 4.18: Mã nguồn của API endpoint POST /track, nhận dữ liệu và lưu 68
Hình 4.19: Mã nguồn của Recommendation Engine 70
Hình 4.20: Hình ảnh trang chủ thay đổi khi người dùng tương tác 71
Hình 4.21: Biểu đồ độ chính xác Multi-Agent 74
Hình 4.22: Biểu đồ hiệu năng luồng phản hồi..... 74
Hình 4.23: Biểu đồ độ chính xác gợi ý (Accuracy)..... 77
Hình 4.24: Biểu đồ hiệu năng Caching (Latency)..... 77

DANH MỤC BẢNG

Bảng 1.1: Các thách thức chính trong hệ thống TMĐT hiện đại 8
Bảng 2.1: So sánh chi tiết kiến trúc Monolithic và Headless 16
Bảng 3.1: Bảng yêu cầu chức năng của hệ thống 22
Bảng 3.2: Bảng yêu cầu phi chức năng của hệ thống 25
Bảng 3.3: Bảng chatbot_context (lưu trữ ngữ cảnh phiên trò chuyện) 42
Bảng 3.4: Bảng chatbot_responses (lưu trữ các mẫu phản hồi) 42
Bảng 3.5: Bảng chatbot_analytics (lưu trữ dữ liệu phân tích) 43
Bảng 3.6: rec_user_interactions (ghi nhận tương tác thô của người dùng) 45
Bảng 3.7: rec_user_preferences (lưu trữ hồ sơ sở thích người dùng) 46
Bảng 3.8: rec_product_similarities (lưu trữ độ tương đồng giữa các sản phẩm) 47
Bảng 3.9: rec_recommendations_cache (bảng lưu trữ đệm kết quả gợi ý) 48
Bảng 4.1: Các công cụ và công nghệ chính được sử dụng trong đề tài 50
Bảng 4.1: Mẫu dữ liệu sau khi xử lý 52
Bảng 4.2: Danh sách các nhóm tình huống kiểm thử hệ thống Multi-Agent 72
Bảng 4.4: Kiểm thử độ tương quan của thuật toán gợi ý Hybrid 76

DANH MỤC TỪ VIẾT TẮT

STT	Viết tắt	Nghĩa đầy đủ	
		Tiếng Anh	**Tiếng Việt**
1	**TMĐT**	E-commerce	Thương mại điện tử
2	**LLM**	Large Language Model	Mô hình ngôn ngữ lớn
3	**RAG**	Retrieval-Augmented Generation	Tạo sinh tăng cường truy xuất
4	**MAS**	Multi-Agent Systems	Hệ thống đa tác tử
5	**SSR**	Server-Side Rendering	Kết xuất phía máy chủ
6	**CSR**	Client-Side Rendering	Kết xuất phía máy khách
7	**API**	Application Programming Interface	Giao diện lập trình ứng dụng
8	**RBAC**	Role-Based Access Control	Kiểm soát truy cập dựa trên vai trò
9	**Vector DB**	Vector Database	Cơ sở dữ liệu vector
10	**JSON**	JavaScript Object Notation	Một định dạng trao đổi dữ liệu
11	**CRUD**	Create, Read, Update, Delete	Các thao tác cơ bản với dữ liệu
12	**SEO**	Search Engine Optimization	Tối ưu hóa bộ máy tìm kiếm
13	**SDK**	Software Development Kit	Bộ công cụ phát triển phần mềm
14	**TTL**	Time To Live	Thời gian sống (của dữ liệu cache)

CHƯƠNG 1. MỞ ĐẦU

1.1. Bối cảnh và hiện trạng vấn đề

Trong bối cảnh nền kinh tế số toàn cầu đang phát triển với tốc độ vũ bão, TMĐT tại Việt Nam đã và đang khẳng định vai trò là động lực tăng trưởng mũi nhọn. Theo báo cáo "e-Conomy SEA 2024", quy mô thị trường TMĐT Việt Nam ước đạt 22 tỷ USD và dự báo sẽ chạm mốc 63 tỷ USD vào năm 2030. Sự bùng nổ này không chỉ đến từ sự gia tăng về số lượng người mua sắm mà còn phản ánh sự thay đổi sâu sắc trong hành vi tiêu dùng: khách hàng ngày nay đòi hỏi những trải nghiệm mua sắm liền mạch, tốc độ tức thì và đặc biệt là khả năng tương tác được cá nhân hóa ở mức độ cao.

Để đáp ứng nhu cầu ngày càng khắt khe này, các hệ thống TMĐT truyền thống với kiến trúc nguyên khối đang dần bộc lộ những hạn chế về khả năng mở rộng và tùy biến, dẫn đến xu hướng chuyển dịch sang kiến trúc Headless Commerce. Mô hình tách biệt frontend và backend, mang lại sự linh hoạt để tối ưu hóa trải nghiệm người dùng.

Tuy nhiên, ngay cả khi kiến trúc hệ thống được tối ưu, hai bài toán cốt lõi tại lớp tương tác người dùng là hỗ trợ khách hàng và cá nhân hóa sản phẩm vẫn còn nhiều thách thức cấp thiết. Các giải pháp hiện tại thường rơi vào một trong hai thái cực không mong muốn, tạo ra một sự đánh đổi lớn cho doanh nghiệp. Nội dung bảng sau sẽ trình bày các thách thức chính trong hệ thống TMĐT hiện đại.

Bảng 1.1: Các thách thức chính trong hệ thống TMĐT hiện đại

Lĩnh vực	Vấn đề hiện hữu	Hệ quả
Hỗ trợ khách hàng	Chatbot LLM-đơn lẻ: - Chậm: Độ trễ ~3 giây/ phản hồi. - Tốn kém: Chi phí API cao (~$0.01/request). - Không đáng tin cậy: Hiện tượng "ảo giác" (hallucination), cung cấp thông tin sai lệch.	Doanh nghiệp phải lựa chọn giữa một chatbot nhanh-rẻ nhưng cứng nhắc (rule-based) hoặc một chatbot thông minh nhưng chậm-đắt đỏ và kém tin cậy (LLM-only).

Gợi ý cá nhân hóa	Hệ thống Rule-based: - Thiếu cá nhân hóa: Chỉ dựa trên các luật cứng. - Thiếu khả năng khám phá: Không gợi ý được sản phẩm mới lạ. - Vấn đề "cold-start": Không hiệu quả với người dùng hoặc sản phẩm mới.	Trải nghiệm mua sắm của khách hàng trở nên chung chung, không có tính đột phá, dẫn đến giảm tỷ lệ chuyển đổi (CTR) và không giữ chân được người dùng.

Những hạn chế trên buộc doanh nghiệp phải lựa chọn giữa tốc độ và chất lượng, hoặc chi phí và độ chính xác. Chính bối cảnh này đã tạo ra động lực để nghiên cứu và triển khai một giải pháp kiến trúc tiên tiến hơn: hệ thống đa tác tử. Thay vì dùng một "siêu AI" để giải quyết mọi việc, kiến trúc này chia nhỏ nhiệm vụ cho các "nhân viên AI" chuyên biệt, hứa hẹn giải quyết được sự đánh đổi cố hữu, mang lại một hệ thống vừa thông minh, nhanh chóng, đáng tin cậy và tối ưu về mặt chi phí.

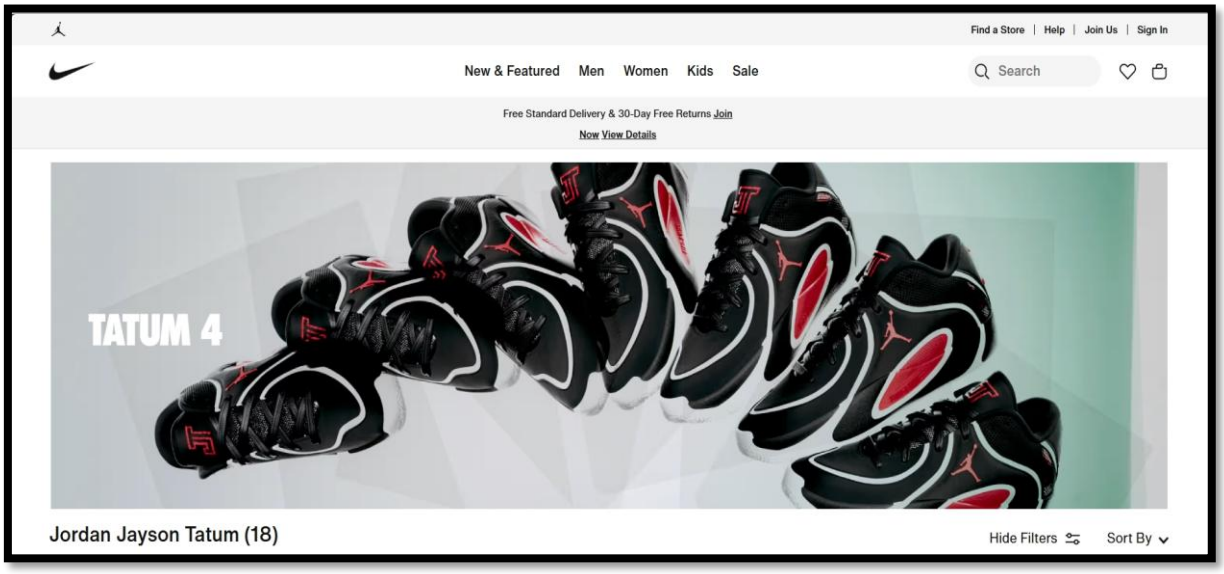
1.2. Các mô hình thương mại điện tử tích hợp AI tiêu biểu

Để làm rõ tính cấp thiết và khả thi của đề tài, việc nghiên cứu các mô hình thương mại điện tử thành công trên thực tế là vô cùng quan trọng. Các doanh nghiệp hàng đầu thế giới và tại Việt Nam đều đang tích cực ứng dụng kiến trúc Headless và trí tuệ nhân tạo để tạo ra lợi thế cạnh tranh, mang lại trải nghiệm khách hàng vượt trội.

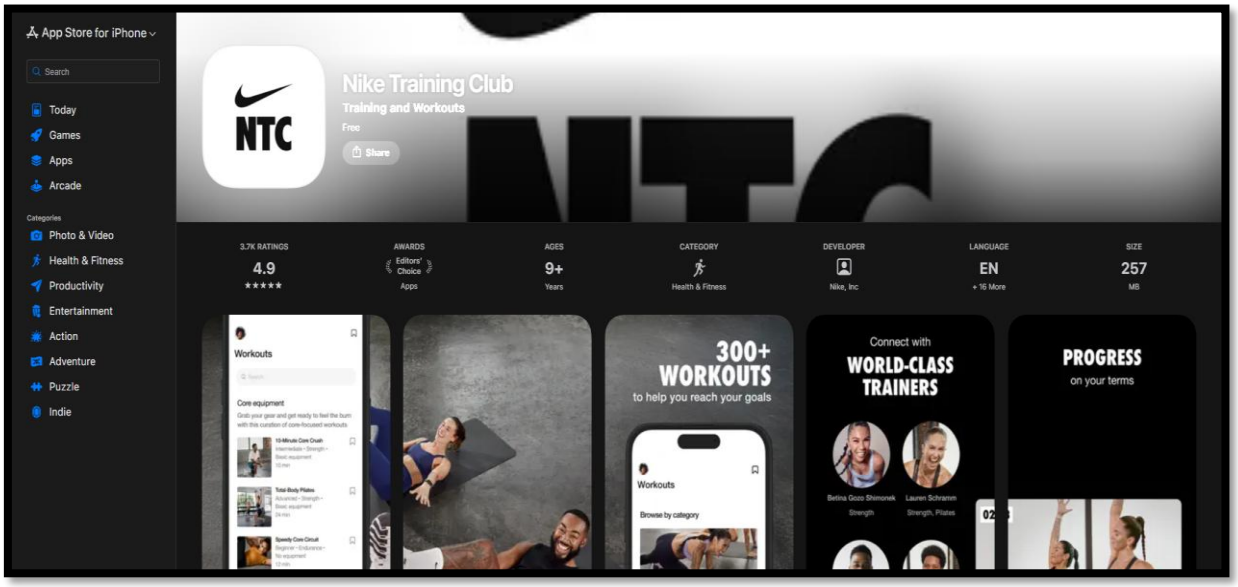
1.2.1. Nike: Kiến trúc Composable Commerce

Nike là một trong những ví dụ tiêu biểu nhất về việc chuyển đổi thành công sang một hệ sinh thái thương mại điện tử hiện đại. Thay vì vận hành một trang web bán hàng duy nhất, Nike đã xây dựng một hệ sinh thái đa dạng các giao diện người dùng như trang web chính Nike.com, ứng dụng SNKRS dành riêng cho cộng đồng yêu giày, và ứng dụng Nike Training Club cho việc luyện tập. Tất cả các giao diện này đều kết nối và đồng bộ dữ liệu với một hệ thống backend tập trung thông qua API. Kiến trúc này chính là hiện thân của Composable Commerce (một dạng nâng cao của Headless), cho phép Nike tạo ra những trải nghiệm chuyên biệt cho từng nhóm khách hàng mà không cần

phải xây dựng lại hệ thống lõi, tạo nên một hành trình mua sắm liền mạch từ online đến cửa hàng vật lý. Mô hình của Nike là minh chứng cho sức mạnh của kiến trúc Headless trong việc mang lại sự linh hoạt và khả năng mở rộng, tạo nền tảng vững chắc để tích hợp các dịch vụ thông minh như chatbot agent.

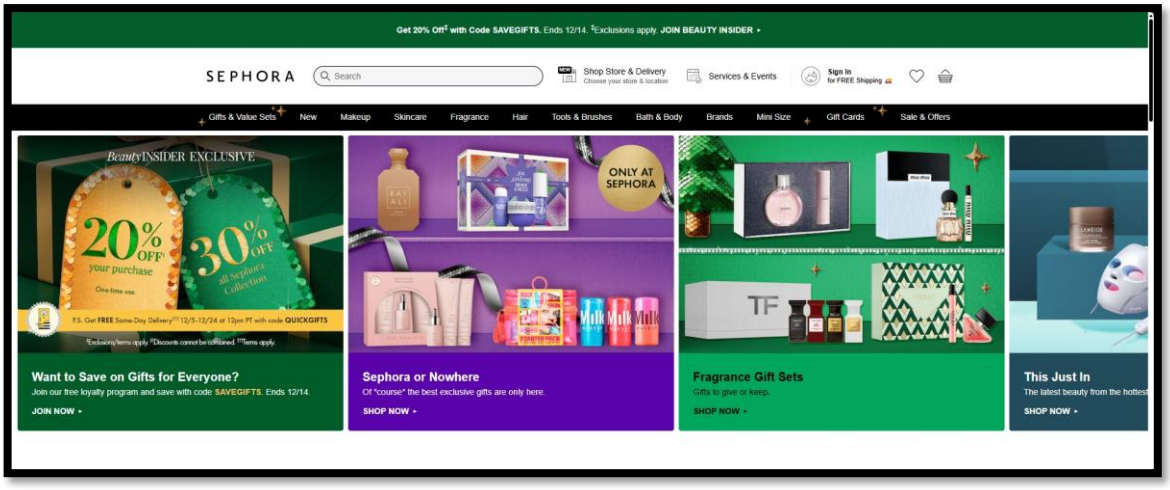


Hình 1.1: Giao diện trang Web Nike.com



Hình 1.2: Giao diện ứng dụng Nike Training Club

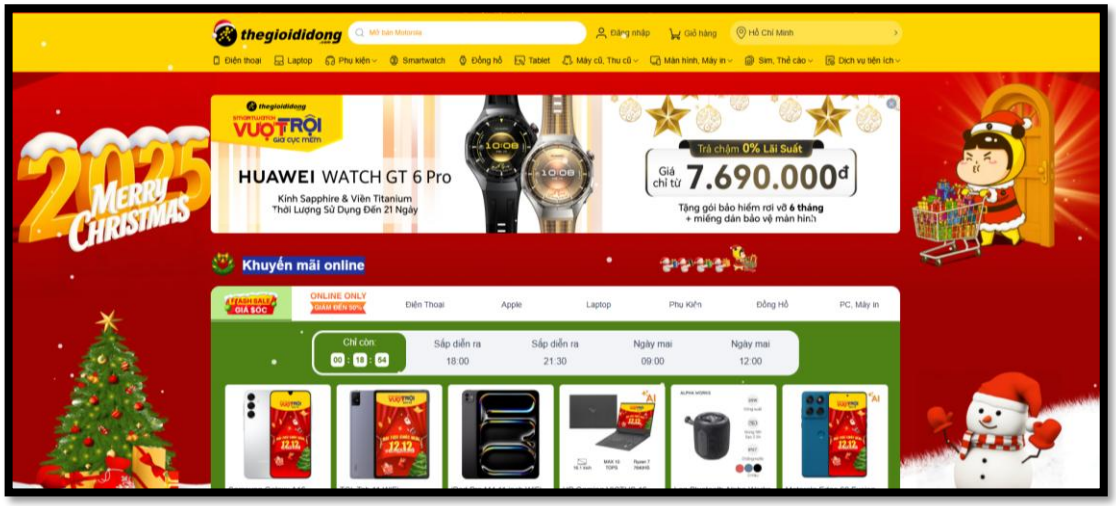
1.2.2. Sephora: Cá nhân hóa trải nghiệm mua sắm bằng trợ lý ảo AI



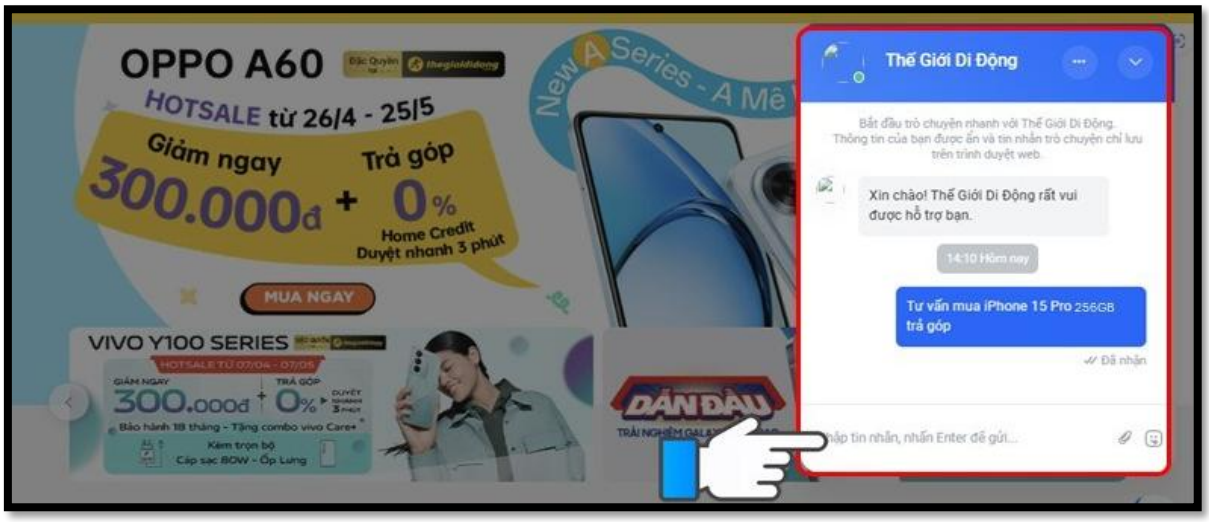
Hình 1.3: Giao diện trang Web sephora.com

Sephora, nhà bán lẻ mỹ phẩm hàng đầu, lại là một ví dụ tiên phong trong việc sử dụng trợ lý ảo AI để cá nhân hóa trải nghiệm mua sắm. Trợ lý ảo của họ không chỉ đơn thuần trả lời các câu hỏi về chính sách hay đơn hàng, mà còn đóng vai trò như một chuyên gia tư vấn sắc đẹp ảo, hoạt động như một tác tử bán hàng (sales agent) thông minh. Chatbot này có khả năng trò chuyện với khách hàng để hiểu về loại da, tông màu yêu thích, từ đó đưa ra những gợi ý sản phẩm phù hợp và thậm chí tích hợp tính năng thực tế tăng cường (AR) để khách hàng "thử" mỹ phẩm trực tuyến. Hệ thống này chứng tỏ một trợ lý ảo không chỉ là công cụ hỗ trợ mà còn là một kênh bán hàng hiệu quả, giúp tăng tỷ lệ chuyển đổi và sự gắn kết của khách hàng.

1.2.3. Thế giới di động: Tối ưu vận hành Omnichannel tại thị trường Việt Nam



Hình 1.4: Giao diện trang Web thegioididong.com



Hình 1.5: Chức năng hỗ trợ khách hàng qua Zalo

Tại thị trường Việt Nam, thế giới di động (TGDD) là một ví dụ xuất sắc về việc xây dựng trải nghiệm khách hàng omnichannel liền mạch, điều gần như không thể thực hiện được trên các nền tảng nguyên khối truyền thống. Khách hàng có thể dễ dàng kiểm tra tình trạng tồn kho của sản phẩm tại một cửa hàng cụ thể trên website, đặt hàng trực tuyến và nhận hàng tại cửa hàng chỉ trong vòng một giờ. Hơn nữa, họ cũng ứng dụng các chatbot trên nền tảng Zalo để tự động hóa việc hỗ trợ khách hàng, giải đáp các thắc mắc về đơn hàng và bảo hành. Mô hình của TGDD chứng minh rằng việc áp dụng các nguyên tắc của kiến trúc linh hoạt và tự động hóa bằng AI là yếu tố then chốt để thành công trong môi trường TMĐT cạnh tranh gay gắt tại Việt Nam.

Từ những ví dụ thành công của Nike, Sephora và TGDD, có thể thấy rõ rằng việc áp dụng kiến trúc Headless kết hợp cùng các giải pháp trí tuệ nhân tạo không chỉ là một lợi thế cạnh tranh mà còn là chìa khóa để tạo ra những trải nghiệm khách hàng vượt trội.

1.3. Mục tiêu đề tài

Xuất phát từ bối cảnh thực tiễn và thách thức đã phân tích tại mục 1.1, đề tài ***"Xây dựng hệ thống thương mại điện tử tích hợp trợ lý ảo Multi-Agent và hệ thống gợi ý cá nhân hóa"*** được thực hiện nhằm hướng tới các mục tiêu chính sau đây:

1. Nghiên cứu và xây dựng một hệ thống thương mại điện tử hoàn chỉnh theo kiến trúc Headless: Áp dụng nền tảng MedusaJS để xây dựng một backend mạnh mẽ, linh hoạt và sử dụng Next.js để phát triển một frontend có hiệu năng cao, tối ưu cho trải nghiệm người dùng và công cụ tìm kiếm.

2. Thiết kế và triển khai dịch vụ trợ lý ảo thông minh dựa trên kiến trúc đa

tác tử: Xây dựng một chatbot service không chỉ có khả năng trò chuyện tự nhiên mà còn có thể thực hiện các tác vụ cụ thể thông qua các tác tử chuyên biệt, bao gồm: tác tử tư vấn sản phẩm, tác tử tra cứu đơn hàng và tác tử hỗ trợ chung.

3. Thiết kế và triển khai hệ thống gợi ý sản phẩm cá nhân hóa: Phát triển và triển khai recommendation service dựa trên mô hình Hybrid, kết hợp kỹ thuật lọc dựa trên nội dung và lọc cộng tác để cải thiện đáng kể độ chính xác đồng thời tăng cường tính đa dạng của các sản phẩm được đề xuất.

4. Tích hợp thành công và liền mạch các thành phần của hệ thống: Đảm bảo chatbot service và recommendation service có khả năng giao tiếp hiệu quả với backend Medusa thông qua API để truy xuất dữ liệu thời gian thực và tích hợp mượt mà vào giao diện frontend Next.js để mang lại trải nghiệm tương tác tự nhiên cho người dùng cuối.

5. Đánh giá hiệu quả và tính khả thi của giải pháp: Xây dựng các kịch bản kiểm thử để đánh giá khả năng của hệ thống Multi-Agent và thuật toán gợi ý trong việc hiểu đúng ý định người dùng, điều phối tác vụ chính xác và cung cấp thông tin đáng tin cậy, từ đó chứng minh giá trị thực tiễn của mô hình so với mô hình truyền thống.

1.4. Đối tượng và phạm vi đề tài

Đối tượng nghiên cứu của đề tài là kiến trúc tích hợp giữa nền tảng TMĐT theo mô hình Headless và các dịch vụ trí tuệ nhân tạo thông minh nhằm nâng cao trải nghiệm người dùng và hiệu quả hoạt động của hệ thống. Cụ thể, đề tài tập trung nghiên cứu trợ lý ảo được xây dựng theo kiến trúc Multi-Agent, có khả năng tương tác, hỗ trợ người dùng trong quá trình tìm kiếm và lựa chọn sản phẩm, cùng với hệ thống gợi ý sản phẩm cá nhân hóa dựa trên hành vi sử dụng của người dùng.

Về phạm vi nghiên cứu, đề tài tập trung vào quá trình phân tích yêu cầu, thiết kế kiến trúc hệ thống, triển khai các thành phần kỹ thuật và đánh giá hiệu quả hoạt động của mô hình đề xuất. Các nội dung nghiên cứu chủ yếu xoay quanh việc tích hợp giữa nền tảng Headless, các AI service và backend xử lý dữ liệu, từ đó xây dựng một hệ thống có khả năng mở rộng, linh hoạt và dễ dàng phát triển trong tương lai.

Đề tài không đi sâu vào các khía cạnh liên quan đến vận hành kinh doanh, chiến lược marketing, quản lý tài chính hay nghiên cứu thuật toán AI ở mức độ lý thuyết hàn lâm. Thay vào đó, tập trung vào góc độ kỹ thuật và ứng dụng thực tiễn, nhằm chứng minh tính khả thi của mô hình tích hợp AI trong hệ thống TMĐT thông qua việc xây dựng và triển khai một sản phẩm mẫu.

1.5. Ý nghĩa và giá trị thực tiễn đề tài

Đề tài mang lại giá trị thực tiễn cao, cung cấp một giải pháp toàn diện giúp doanh nghiệp giải quyết đồng thời hai bài toán cốt lõi là tự động hóa hỗ trợ khách hàng và cá nhân hóa trải nghiệm mua sắm, qua đó nâng cao sự hài lòng của khách hàng, tăng tỷ lệ chuyển đổi và tạo ra lợi thế cạnh tranh. Về mặt khoa học, đề tài đề xuất một mô hình kiến trúc tham chiếu, minh chứng cho tính khả thi và hiệu quả của việc kết hợp kiến trúc Headless, hệ thống Multi-Agent và thuật toán gợi ý Hybrid, có thể làm tài liệu cho các nghiên cứu và ứng dụng trong tương lai.

1.6. Bố cục báo cáo

Ngoài các phần phụ như Mục lục, Danh mục hình, Danh mục bảng, Danh mục viết tắt, Tài liệu tham khảo và Phụ lục, báo cáo gồm 5 chương chính:

- Chương 1: Mở đầu: Trình bày bối cảnh, mục tiêu, phạm vi và đóng góp của nghiên cứu.

- Chương 2: Cơ sở lý thuyết: Trình bày các kiến thức nền về TMĐT hiện đại, kiến trúc Multi-Agent, kỹ thuật Hybrid và các kỹ thuật liên quan.

- Chương 3: Phân tích và thiết kế hệ thống: Mô tả kiến trúc tổng thể, quy trình hoạt động của chuỗi 5 tác tử và thiết kế cơ sở dữ liệu cho dịch vụ gợi ý.

- Chương 4: Triển khai và đánh giá thực nghiệm: Chi tiết quá trình cài đặt hệ thống, thiết lập kiểm thử và phân tích kết quả thực nghiệm về hiệu năng, độ chính xác.

- Chương 5: Kết luận và hướng phát triển: Tóm tắt kết quả, nêu rõ hạn chế và đề xuất định hướng phát triển tiếp theo.

Cấu trúc báo cáo được xây dựng nhằm đảm bảo tính logic, chặt chẽ và toàn diện trong suốt quá trình nghiên cứu và trình bày kết quả.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ

Để hiện thực hóa các mục tiêu đã đề ra, chương này sẽ đi sâu vào việc trình bày nền tảng lý thuyết và các công nghệ cốt lõi được sử dụng để xây dựng hệ thống. Nội dung chính bao gồm kiến trúc Headless Commerce với MedusaJS và Next.js; các nguyên lý về trí tuệ nhân tạo như LLM, kỹ thuật RAG, kiến trúc Multi-Agent; phương pháp luận về hệ thống gợi ý cá nhân hóa dựa trên thuật toán lai. Đây là nền tảng kiến thức quan trọng, làm cơ sở cho các chương phân tích và triển khai hệ thống sau này.

2.1. Kiến trúc TMĐT hiện đại và nền tảng MedusaJS

2.1.1. Khái niệm và lợi ích của kiến trúc Headless Commerce

Headless Commerce là một kiến trúc hệ thống hiện đại, trong đó lớp giao diện người dùng, hay còn gọi là "đầu", được tách biệt hoàn toàn khỏi lớp xử lý nghiệp vụ và quản trị dữ liệu. Khác với các nền tảng thương mại điện tử truyền thống có kiến trúc nguyên khối như Shopify hay WooCommerce ở dạng cơ bản, nơi frontend và backend được xây dựng trên cùng một nền tảng và ràng buộc chặt chẽ, kiến trúc Headless cho phép hai lớp này hoạt động độc lập và giao tiếp với nhau hoàn toàn thông qua API.

Việc tách rời này mang lại ba lợi ích chiến lược:

1. Sự linh hoạt tối đa: Doanh nghiệp có thể tự do lựa chọn bất kỳ công nghệ nào để xây dựng giao diện người dùng (ví dụ: Next.js, Gatsby, hoặc ứng dụng di động) mà không bị giới hạn bởi công nghệ của backend. Điều này cho phép tạo ra những trải nghiệm độc đáo và tối ưu nhất cho từng kênh bán hàng.

2. Trải nghiệm đa kênh liền mạch: Một hệ thống backend duy nhất có thể cung cấp dữ liệu và dịch vụ cho nhiều "head" khác nhau như website, ứng dụng di động, thiết bị IoT, hay các điểm bán hàng thông minh. Điều này tạo ra một hành trình mua sắm đồng nhất cho khách hàng trên mọi điểm chạm.

3. Hiệu năng và khả năng mở rộng vượt trội: frontend và backend có thể được tối ưu, nâng cấp và mở rộng quy mô một cách độc lập dựa trên nhu cầu thực tế, đảm bảo hệ thống luôn hoạt động với hiệu suất cao nhất.

Bảng dưới đây minh họa sự khác biệt cơ bản giữa hai kiến trúc:

Bảng 2.1: So sánh chi tiết kiến trúc Monolithic và Headless

Tiêu chí	Kiến trúc Monolithic (truyền thống)	Kiến trúc Headless (hiện đại)
Cấu trúc	Tất cả trong một: frontend và backend được xây dựng và ràng buộc trong một khối ứng dụng.	Tách biệt frontend và backend: Hai lớp này hoạt động hoàn toàn độc lập với nhau.
Giao tiếp	Các thành phần giao tiếp nội bộ thông qua các lệnh gọi hàm (function calls) trực tiếp.	Giao tiếp duy nhất thông qua API. Backend cung cấp các API endpoint để frontend thể "gọi" đến.
Linh hoạt công nghệ	Thấp: Toàn bộ hệ thống thường bị ràng buộc vào một stack công nghệ duy nhất. Khó khăn khi muốn áp dụng công nghệ mới cho chỉ một phần.	Cao: Hoàn toàn tự do lựa chọn công nghệ tốt nhất cho từng phần. Ví dụ: Backend dùng Node.js, frontend dùng Next.js, ứng dụng di động dùng React Native.
Khả năng mở rộng	Khó khăn: Khi một phần của hệ thống bị quá tải, phải mở rộng quy mô của toàn bộ ứng dụng, gây lãng phí tài nguyên.	Linh hoạt: Có thể mở rộng quy mô của từng thành phần độc lập. Nếu lượng truy cập frontend tăng, chỉ cần thêm tài nguyên cho frontend.
Trải nghiệm người dùng	Bị giới hạn: Việc cập nhật giao diện hoặc tạo ra các kênh bán hàng mới (đa kênh) thường phức tạp và tốn thời gian.	Tối ưu và đa kênh: Dễ dàng xây dựng và tối ưu hóa trải nghiệm cho nhiều "đầu ra" (website, app, …). Hỗ trợ mạnh mẽ cho chiến lược Omnichannel.
Tốc độ phát triển	Chậm hơn: Các nhóm phát triển phụ thuộc lẫn nhau. Một thay đổi nhỏ ở backend có thể yêu cầu thay đổi lớn ở frontend.	Nhanh hơn: Các nhóm có thể phát triển độc lập, song song. Miễn tuân thủ kiến trúc API, hai bên có thể làm việc mà không cản trở nhau.

2.1.2. Giới thiệu MedusaJS: Giải pháp Headless Commerce mã nguồn mở

MedusaJS được xây dựng theo kiến trúc Headless, nổi lên như một giải pháp thay thế linh hoạt và mạnh mẽ cho các nền tảng độc quyền như Shopify hay các plugin như WooCommerce. Việc lựa chọn MedusaJS cho đề tài này không phải là ngẫu nhiên, mà dựa trên những ưu điểm vượt trội phù hợp với các mục tiêu đã đề ra.

Khác với Shopify vốn là một nền tảng đóng với các giới hạn về tùy chỉnh và phí, MedusaJS là mã nguồn mở, mang lại cho nhà phát triển toàn quyền kiểm soát logic hệ thống, không bị phụ thuộc vào nhà cung cấp và không có chi phí ẩn. So với WooCommerce, vốn là một plugin của WordPress và có thể thiếu ổn định khi cài đặt nhiều plugin khác nhau, Medusa được xây dựng trên nền tảng Node.js hiện đại. Điều này không chỉ đảm bảo hiệu năng cao mà còn tạo ra một hệ sinh thái công nghệ đồng nhất từ backend đến frontend, giúp tối ưu hóa quá trình phát triển. Hơn hết, Medusa được thiết kế với triết lý "developer-first", cung cấp một kiến trúc module rõ ràng và API mạnh mẽ, tạo điều kiện lý tưởng cho việc tùy biến và tích hợp các dịch vụ phức tạp.

2.1.3. Sự tương thích giữa kiến trúc module của Medusa với hệ thống Multi-Agent

Kiến trúc module của Medusa không chỉ là một ưu điểm về mặt kỹ thuật mà còn là yếu tố then chốt giúp việc tích hợp hệ thống Multi-Agent trở nên khả thi và hiệu quả. Nền tảng Medusa được xây dựng dựa trên một lõi chứa các chức năng thương mại điện tử cơ bản, và các tính năng khác như cổng thanh toán, dịch vụ vận chuyển hay hệ thống tìm kiếm đều được triển khai dưới dạng các module độc lập. Toàn bộ các chức năng này đều được triển khai thành các API endpoints được tài liệu hóa rõ ràng.

Chính kiến trúc API-first này đã biến backend Medusa thành một toolbox hoàn hảo cho các tác tử AI. Thay vì phải tương tác phức tạp với cơ sở dữ liệu, mỗi tác tử trong hệ thống Multi-Agent có thể được lập trình để gọi đến các API cụ thể nhằm thực hiện nhiệm vụ của mình. Ví dụ, order agent có thể sử dụng API /store/orders/{id} để tra cứu trạng thái đơn hàng, trong khi sales agent có thể sử dụng API /store/products để tìm kiếm và đề xuất sản phẩm. Nếu một tác tử cần một chức năng chuyên biệt chưa có sẵn, kiến trúc module của Medusa cho phép dễ dàng phát triển một API endpoint tùy chỉnh mới mà không làm ảnh hưởng đến phần còn lại của hệ thống. Sự tương thích này đảm bảo một cơ chế giao tiếp an toàn, có cấu trúc và cực kỳ linh hoạt giữa hệ thống AI và logic nghiệp vụ, là nền tảng vững chắc để xây dựng một trợ lý ảo thực sự thông minh.

2.2. Next.js: Framework Frontend cho trải nghiệm TMĐT hiện đại

Trong kiến trúc Headless Commerce, việc lựa chọn Next.js không chỉ đơn thuần là chọn một thư viện hiển thị mà là triển khai một giải pháp tối ưu toàn diện cho lớp tương tác người dùng. Framework này vượt trội nhờ khả năng thực hiện Hybrid Rendering, cho phép kết hợp linh hoạt giữa SSR cho các thành phần cần dữ liệu thời gian thực như giỏ hàng, thông tin cá nhân và SSG cho danh mục sản phẩm nhằm đạt tốc độ tải trang gần như tức thì. Sự linh hoạt này giúp hệ thống giảm thiểu thời gian chờ đợi "trang trắng" và nâng cao tỷ lệ chuyển đổi đơn hàng.

Đặc biệt, Next.js giải quyết triệt để bài toán SEO – vốn là điểm yếu cố hữu của các ứng dụng trang đơn truyền thống – bằng cách cung cấp mã HTML đầy đủ nội dung sản phẩm cho các trình thu thập dữ liệu của công cụ tìm kiếm ngay từ phía máy chủ. Bên cạnh đó, các tính năng tích hợp sẵn như tối ưu hóa hình ảnh tự động, chia nhỏ mã nguồn và khả năng thiết lập các API Routes để làm trung gian giao tiếp mượt mà giữa frontend với MedusaJS backend cũng như các AI service đã biến Next.js thành mảnh ghép hoàn hảo. Điều này giúp xây dựng một hệ thống TMĐT có hiệu năng vượt trội và khả năng mở rộng linh hoạt trong môi trường số cạnh tranh.

2.3. Nền tảng trí tuệ nhân tạo cho trợ lý ảo

Để xây dựng một trợ lý ảo thực sự thông minh, có khả năng tương tác và thực hiện tác vụ phức tạp, việc lựa chọn và kết hợp các công nghệ trí tuệ nhân tạo tiên tiến là yếu tố then chốt. Phần này sẽ trình bày các khái niệm nền tảng, từ các mô hình ngôn ngữ làm cốt lõi giao tiếp đến các kiến trúc nâng cao giúp hệ thống có thể truy xuất tri thức và tự động hóa hành động.

2.3.1. Mô hình ngôn ngữ lớn (LLM)

LLM là các mô hình học sâu được huấn luyện trên khối lượng dữ liệu văn bản khổng lồ, cho phép chúng có khả năng hiểu, diễn giải, tóm tắt và tạo ra văn bản giống con người một cách đáng kinh ngạc. Các mô hình tiêu biểu hiện nay như dòng GPT của OpenAI, Claude của Anthropic, hay Gemini của Google đã và đang định hình lại cách con người tương tác với máy móc. Năng lực giao tiếp tự nhiên của LLM chính là nền tảng cốt lõi, giúp trợ lý ảo có thể hiểu được các câu hỏi đa dạng và phức tạp của người dùng mà không cần tuân theo các quy tắc cứng nhắc. Tuy nhiên, một hạn chế cố hữu của LLM là kiến thức của chúng bị "đóng băng" tại thời điểm huấn luyện và chúng

không có quyền truy cập vào các nguồn dữ liệu riêng tư hoặc theo thời gian thực của một doanh nghiệp cụ thể, dẫn đến nguy cơ đưa ra thông tin lỗi thời hoặc bịa đặt.

2.3.2. Kỹ thuật Retrieval-Augmented Generation (RAG)

Để giải quyết vấn đề thiếu hụt kiến thức chuyên biệt của LLM, kỹ thuật Tăng cường truy xuất thông tin RAG đã ra đời. RAG là một kiến trúc cho phép LLM truy cập và sử dụng thông tin từ một nguồn tri thức bên ngoài trước khi tạo ra câu trả lời. Quy trình này gồm hai giai đoạn chính:

1. Truy xuất (retrieval): Khi người dùng đặt câu hỏi, hệ thống sẽ chuyển câu hỏi đó thành một vector ngữ nghĩa và thực hiện tìm kiếm tương đồng trong một cơ sở dữ liệu vector để tìm ra những đoạn thông tin liên quan nhất.

2. Tạo sinh (generation): Những thông tin được truy xuất này sẽ được đưa vào làm ngữ cảnh kèm theo câu hỏi ban đầu để LLM tạo ra câu trả lời.

Bằng cách này, câu trả lời của LLM được "neo" vào nguồn dữ liệu thực tế, giúp giảm thiểu đáng kể hiện tượng ảo giác và đảm bảo tính chính xác, cập nhật thông tin.

2.3.3. Kiến trúc đa tác tử (Multi-Agent System)

Trong khi RAG giải quyết bài toán truy xuất tri thức, Multi-Agent giải quyết bài toán thực thi hành động. Thay vì sử dụng một LLM duy nhất để xử lý mọi yêu cầu, hệ thống đa tác tử hoạt động như một "đội ngũ nhân viên ảo", nơi mỗi tác tử (agent) là một thực thể AI chuyên biệt, được thiết kế để thực hiện một nhóm nhiệm vụ cụ thể. Các tác tử này có thể giao tiếp và phối hợp với nhau để giải quyết các vấn đề phức tạp.

Một quy trình hoạt động điển hình bao gồm ba thành phần chính:

1. Router Agent (tác tử điều phối): Đóng vai trò như một người quản lý, có nhiệm vụ phân tích ý định của người dùng và chuyển yêu cầu đến đúng tác tử chuyên môn.

2. Specialist Agents (các tác tử chuyên biệt): Mỗi tác tử được trang bị một bộ "công cụ" (tools) riêng. Ví dụ, sales agent có công cụ truy vấn RAG để tư vấn sản phẩm, trong khi order agent có công cụ gọi API để kiểm tra trạng thái đơn hàng.

3. Tools (công cụ): Là các hàm chức năng cụ thể mà agent có thể thực thi, ví dụ như gọi API đến Medusa backend hoặc truy vấn cơ sở dữ liệu.

Kiến trúc này cho phép hệ thống không chỉ "trả lời câu hỏi" mà còn "thực thi công việc", tạo ra một trợ lý ảo mạnh mẽ, linh hoạt và có khả năng mở rộng cao.

2.4. Nền tảng lý thuyết cho hệ thống gợi ý cá nhân hóa

Hệ thống gợi ý (Recommendation System) là một thành phần quan trọng trong TMĐT, giúp nâng cao trải nghiệm người dùng và thúc đẩy doanh số bằng cách đề xuất các sản phẩm phù hợp. Đề tài tập trung vào việc xây dựng một hệ thống gợi ý hiệu quả dựa trên các nền tảng lý thuyết sau:

2.4.1. Lọc dựa trên nội dung (Content-Based Filtering)

Lọc dựa trên nội dung hoạt động theo nguyên tắc: "Nếu bạn thích một sản phẩm, bạn cũng có thể thích những sản phẩm khác có thuộc tính tương tự". Phương pháp này đề xuất các sản phẩm bằng cách so sánh các thuộc tính của chúng (như danh mục, thương hiệu, mô tả) với tập thuộc tính của những sản phẩm mà người dùng đã thể hiện sự quan tâm trong quá khứ.

- Ưu điểm: Có khả năng giải quyết vấn đề cold start cho sản phẩm mới (chỉ cần có mô tả là có thể được gợi ý) và các gợi ý có tính giải thích cao.

- Nhược điểm: Dễ tạo ra "bong bóng lọc" (filter bubble), giới hạn khả năng khám phá của người dùng vào những sản phẩm quá giống nhau.

2.4.2. Lọc cộng tác (Collaborative Filtering)

Lọc cộng tác là phương pháp phổ biến nhất, hoạt động dựa trên nguyên tắc: "Những người dùng có cùng sở thích với bạn cũng thích sản phẩm X, vậy có lẽ bạn cũng sẽ thích sản phẩm X". Nó phân tích hành vi của một lượng lớn người dùng (lượt xem, mua hàng) để tìm ra các nhóm người dùng có sở thích tương đồng, sau đó gợi ý sản phẩm mà người dùng trong nhóm đã thích nhưng người dùng hiện tại chưa tương tác.

- Ưu điểm: Có khả năng gợi ý các sản phẩm mới lạ, mang tính khám phá cao.

- Nhược điểm: Gặp phải vấn đề cold start nghiêm trọng với người dùng mới (chưa có lịch sử hành vi) và sản phẩm mới (chưa có ai tương tác).

2.4.3. Thuật toán lai (Hybrid Algorithm)

Để khắc phục nhược điểm của từng phương pháp riêng lẻ, đề tài áp dụng thuật toán lai (Hybrid). Đây là phương pháp kết hợp sức mạnh của cả lọc dựa trên nội dung và lọc cộng tác. Mô hình hoạt động bằng cách tính toán một điểm số gợi ý cuối cùng dựa trên trọng số của điểm số từ mỗi phương pháp:

Điểm Gợi Ý = (w1 * Điểm Content-Based) + (w2 * Điểm Collaborative)

Cách tiếp cận này cho phép hệ thống vừa đảm bảo tính chính xác, vừa có khả năng khám phá, đồng thời giải quyết hiệu quả bài toán cold start. Khi một người dùng mới tương tác, hệ thống sẽ ưu tiên lọc dựa trên nội dung. Khi người dùng có nhiều dữ liệu hành vi hơn, trọng số của lọc cộng tác sẽ tăng lên.

CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

Dựa trên nền tảng kiến thức đã được trình bày ở Chương 2, Chương 3 sẽ đi sâu vào quá trình phân tích và thiết kế, nhằm chuyển hóa các yêu cầu nghiệp vụ và lý thuyết công nghệ thành một bản thiết kế hệ thống chi tiết và khả thi. Chương này đóng vai trò là bản vẽ kỹ thuật, mô tả cách thức hệ thống sẽ được xây dựng, từ việc xác định các tác nhân và chức năng, thiết kế kiến trúc tổng thể, chi tiết hóa cơ sở dữ liệu, cho đến việc mô hình hóa các luồng xử lý phức tạp của các hệ thống AI.

3.1. Phân tích yêu cầu

Trên cơ sở mục tiêu và phạm vi đã xác định, bước tiếp theo là phân tích chi tiết các yêu cầu mà hệ thống cần phải đáp ứng. Các yêu cầu này được phân loại thành hai nhóm chính: yêu cầu chức năng, mô tả những gì hệ thống phải làm, và yêu cầu phi chức năng mô tả hệ thống phải làm như thế nào.

3.1.1. Yêu cầu chức năng

Yêu cầu chức năng xác định các tác vụ và hành vi cụ thể mà hệ thống phải thực hiện từ góc nhìn của từng tác nhân chính. Các chức năng này được tóm tắt trong bảng dưới đây, làm cơ sở cho việc thiết kế và triển khai hệ thống, đảm bảo đáp ứng đầy đủ nhu cầu của người dùng và quản trị viên.

Bảng 3.1: Bảng yêu cầu chức năng của hệ thống

Tác nhân	Chức năng chính	Mô tả chi tiết
Khách hàng	Mua sắm cơ bản	Đăng ký tài khoản, đăng nhập, tìm kiếm sản phẩm theo từ khóa/bộ lọc, xem chi tiết sản phẩm, quản lý giỏ hàng (thêm/xóa/cập nhật sản lượng), thực hiện thanh toán an toàn, và xem lại lịch sử các đơn hàng đã đặt.
	Quản lý danh sách yêu thích	Cho phép người dùng thêm các sản phẩm yêu thích vào một danh sách riêng để dễ dàng theo dõi và mua sau này.

	Tương tác với trợ lý ảo	Tương tác với trợ lý ảo AI thông qua giao diện trò chuyện trực quan để nhận tư vấn sản phẩm, tra cứu thông tin đơn hàng, và được hỗ trợ các vấn đề liên quan đến mua sắm, chính sách cửa hàng.
Quản trị viên	Quản lý nghiệp vụ cửa hàng	Truy cập và sử dụng giao diện quản trị Medusa Admin để quản lý sản phẩm (thêm/sửa/xóa thông tin, hình ảnh, biến thể), quản lý đơn hàng (xem/cập nhật trạng thái), và quản lý thông tin khách hàng.
	Quản lý nội dung	Quản lý các bài viết, tin tức hoặc blog trên website (thêm mới, chỉnh sửa, xóa, duyệt đăng) nhằm cung cấp thông tin hữu ích và thu hút khách hàng.
	Thống kê, báo cáo	Xem các báo cáo thống kê về doanh số bán hàng, số lượng đơn hàng, lượng khách hàng, và hiệu suất của chatbot, cung cấp cái nhìn tổng quan về hoạt động kinh doanh.
	Giám sát trợ lý ảo	Xem lại lịch sử các cuộc hội thoại giữa người dùng và trợ lý ảo để giám sát hiệu quả, thu thập phản hồi, và có dữ liệu để đánh giá, cải tiến hệ thống AI trong tương lai.
Hệ thống trợ lý ảo	Phân tích và hiểu ý định khách hàng	Có khả năng xử lý ngôn ngữ tự nhiên, phân tích và hiểu đúng ý định cốt lõi của

		người dùng từ các câu hỏi đa dạng và không theo cấu trúc.
	Thực thi tác vụ và truy xuất dữ liệu	Dựa trên ý định đã phân loại, hệ thống tự động lựa chọn và thực thi hành động phù hợp: truy vấn cơ sở dữ liệu vector để tư vấn sản phẩm (RAG), gọi API của Medusa backend để lấy thông tin đơn hàng thời gian thực, và trả lời các câu hỏi chung dựa trên cơ sở tri thức đã được cung cấp một cách chính xác.
Hệ thống gợi ý (Recommendation)	Theo dõi hành vi người dùng	Tự động ghi nhận và phân tích các tương tác của người dùng trên website (lượt xem, thêm vào giỏ, mua hàng) để xây dựng hồ sơ sở thích.
	Tính toán và cung cấp gợi ý	Áp dụng thuật toán Hybrid để tính toán và trả về danh sách sản phẩm gợi ý cá nhân hóa thông qua API.
	Xử lý cold start	Cung cấp các gợi ý có liên quan cho người dùng mới (dựa trên các sản phẩm phổ biến hoặc tương tự) khi chưa có đủ dữ liệu hành vi.

3.1.2. Yêu cầu phi chức năng

Yêu cầu phi chức năng xác định các tiêu chí về chất lượng, hiệu suất và bảo mật của hệ thống.

Bảng 3.2: Bảng yêu cầu phi chức năng của hệ thống

Yêu cầu phi chức năng	Mô tả chi tiết
Hiệu năng và tốc độ phản hồi	Hệ thống phải đảm bảo hiệu năng cao trên mọi thành phần. - Giao diện người dùng: Tốc độ tải trang nhanh, mang lại trải nghiệm duyệt web mượt mà và tương tác tức thì. - Dịch vụ trợ lý ảo: Thời gian phản hồi phải đủ nhanh để duy trì một cuộc hội thoại tự nhiên, lý tưởng là dưới 5 giây từ khi người dùng gửi câu hỏi đến khi nhận được câu trả lời hoàn chỉnh.
Bảo mật	Bảo mật là yêu cầu tối quan trọng, bảo vệ dữ liệu người dùng và thông tin hệ thống. - Quản lý khóa API: Mọi API Key dùng để giao tiếp với các dịch vụ bên thứ ba phải được lưu trữ và quản lý một cách an toàn phía máy chủ, tuyệt đối không được lộ ra phía client. - Mã hóa dữ liệu: Toàn bộ quá trình giao tiếp giữa các thành phần của hệ thống (frontend, backend, AI service) phải được mã hóa thông qua giao thức HTTPS để ngăn chặn nghe lén và tấn công. - Xác thực và phân quyền: Đảm bảo chỉ người dùng và quản trị viên hợp lệ mới có thể truy cập các chức năng tương ứng, với cơ chế xác thực mạnh mẽ.

Tính khả dụng và độ tin cậy	Hệ thống phải đảm bảo hoạt động ổn định và cung cấp thông tin chính xác. - Thời gian hoạt động: Hệ thống phải có độ sẵn sàng cao, đảm bảo hoạt động ổn định 24/7 với thời gian ngừng hoạt động tối thiểu. - Độ tin cậy của thông tin: Các câu trả lời do trợ lý ảo cung cấp phải có độ tin cậy cao, dựa trên nguồn dữ liệu xác thực từ backend hoặc cơ sở tri thức, hạn chế tối đa việc cung cấp thông tin sai lệch.
Khả năng mở rộng	Hệ thống phải có khả năng mở rộng linh hoạt để đáp ứng nhu cầu tăng trưởng. Kiến trúc Microservices: Với kiến trúc Headless, các thành phần frontend, backend và AI service phải có khả năng mở rộng quy mô một cách độc lập dựa trên tải lượng truy cập và xử lý thực tế, mà không làm ảnh hưởng đến hiệu suất chung của hệ thống.

3.2. Biểu đồ ca sử dụng

Biểu đồ ca sử dụng (Use Case Diagram) được sử dụng để mô tả các chức năng chính của hệ thống và sự tương tác giữa các tác nhân (actors) với các chức năng đó. Đối với hệ thống này, chúng ta có hai tác nhân chính: người dùng và quản trị viên.

Giải thích biểu đồ chi tiết

Biểu đồ này phân tách hệ thống thành ba khối chính, thể hiện rõ kiến trúc Headless và vai trò của dịch vụ AI:

1. Các tác nhân:

- Người dùng: Tác nhân chính tương tác với giao diện frontend. Họ thực hiện các hoạt động mua sắm và là người khởi tạo cuộc trò chuyện với trợ lý ảo.

- Quản trị viên: Tương tác trực tiếp với Medusa backend (để quản lý các hoạt động cốt lõi của doanh nghiệp như sản phẩm, đơn hàng, khách hàng.

- Hệ thống tác tử AI: Được mô hình hóa như một tác nhân phụ. Nó không tự khởi tạo hành động mà thay mặt người dùng, tự động tương tác với Medusa backend qua API

để truy xuất thông tin hoặc thực hiện tác vụ.

2. Các thành phần hệ thống:

- Website frontend: Là lớp giao diện người dùng, nơi tất cả các tương tác của khách hàng diễn ra.

- AI chatbot service: Là dịch vụ thông minh độc lập, xử lý các cuộc hội thoại. Nó nhận yêu cầu từ frontend, phân tích ý định và điều phối đến các tác tử chuyên biệt.

- AI recommendation service: Là dịch vụ thông minh thứ hai, chịu trách nhiệm về việc cá nhân hóa. Nó ngầm theo dõi hành vi của người dùng từ frontend và tường minh cung cấp các gợi ý sản phẩm khi được frontend yêu cầu qua API.

- Medusa backend: Là "trái tim" của hệ thống, chứa toàn bộ logic nghiệp vụ và dữ liệu kinh doanh. Cung cấp API cho frontend, chatbot và recommendation service.

3. Luồng tương tác chính của các hệ thống AI:

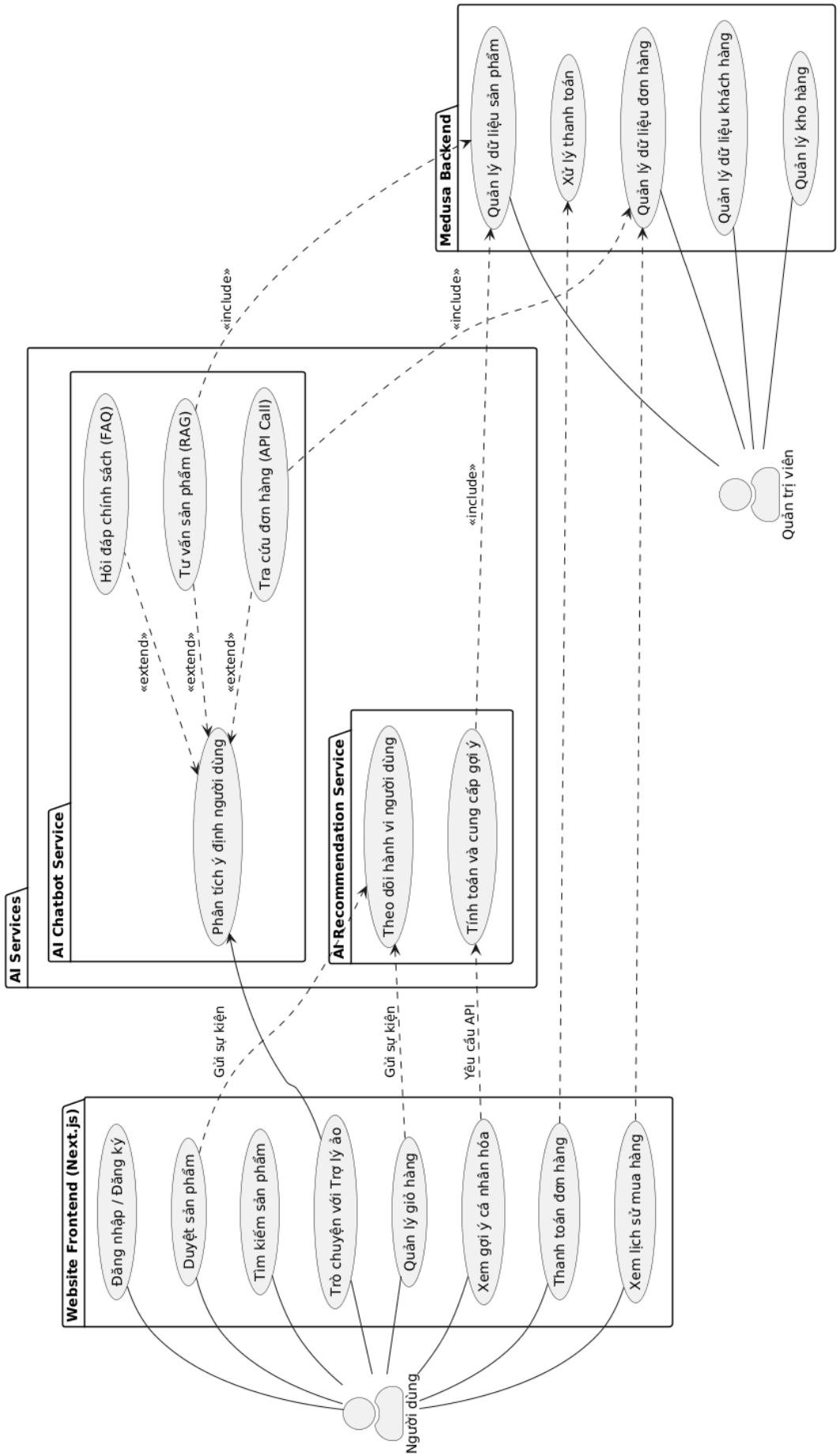
- Luồng chatbot: Người dùng "trò chuyện với trợ lý ảo". Yêu cầu được gửi đến chatbot service để "phân tích ý định". Dựa trên ý định, hệ thống sẽ mở rộng (<<extend>>) sang các ca sử dụng chuyên biệt như "tư vấn sản phẩm" (sử dụng RAG) hoặc "tra cứu đơn hàng" (sử dụng API call), các ca sử dụng này bắt buộc phải bao gồm (<<include>>) việc tương tác với Medusa backend.

- Luồng gợi ý:

+ Theo dõi: Khi người dùng thực hiện các hành động như "duyệt sản phẩm" hay "thêm vào giỏ hàng", frontend sẽ gửi sự kiện đến recommendation service để "theo dõi hành vi người dùng".

+ Cung cấp: Khi người dùng truy cập các trang có hiển thị gợi ý, ca sử dụng "xem gợi ý cá nhân hóa" ở frontend sẽ yêu cầu API từ recommendation service để "tính toán và cung cấp gợi ý". Ca sử dụng này cũng sẽ tương tác với backend để lấy thông tin chi tiết về sản phẩm.

Biểu đồ này thể hiện rõ ràng kiến trúc tách rời: Người dùng chỉ tương tác với frontend, Admin chỉ tương tác với backend, và AI service đóng vai trò trung gian thông minh, kết nối cuộc hội thoại của người dùng với logic nghiệp vụ của hệ thống.



Hình 3.1: Sơ đồ tổng quan kiến trúc hệ thống

3.3. Thiết kế kiến trúc hệ thống

Dựa trên phần phân tích, kiến trúc hệ thống được thiết kế theo triết lý Headless và module hóa, tách biệt các thành phần chính để đảm bảo tính linh hoạt, khả năng mở rộng và bảo trì. Hệ thống gồm ba khối độc lập: Frontend, Backend và AI service.

3.3.1. Sơ đồ kiến trúc tổng thể

Dựa trên các yêu cầu đã phân tích, kiến trúc hệ thống được thiết kế theo triết lý Microservices, tách biệt rõ ràng các thành phần chính để đảm bảo tính linh hoạt, khả năng mở rộng và bảo trì. Hệ thống bao gồm bốn khối dịch vụ độc lập: Frontend, Medusa Backend, AI chatbot service, và AI recommendation service.

Sơ đồ dưới đây mô tả kiến trúc tổng quan của toàn bộ hệ thống, thể hiện mối quan hệ và luồng giao tiếp giữa các thành phần.

Sơ đồ này thể hiện rõ 4 khối dịch vụ chính và các luồng tương tác quan trọng:

1. Frontend: Đóng gói toàn bộ phần giao diện người dùng (UI/UX Layer). Đây là điểm tiếp xúc duy nhất của người dùng với hệ thống. Mọi hành động của người dùng đều bắt đầu từ đây.

2. Medusa backend: Là khối dịch vụ chứa toàn bộ logic nghiệp vụ và dữ liệu kinh doanh, giao tiếp thông qua Medusa API. Nó kết nối với cơ sở dữ liệu PostgreSQL để lưu trữ dữ liệu bền vững và Redis cho các tác vụ caching và hàng đợi.

3. AI chatbot service: Là dịch vụ thông minh chuyên xử lý hội thoại. Lõi của nó là Multi-Agent System, có nhiệm vụ điều phối các tác tử. Nó giao tiếp với LLM service bên ngoài để hiểu và tạo ngôn ngữ, truy vấn Vector DB để thực hiện RAG, và gọi đến Medusa API để thực thi các tác vụ nghiệp vụ.

4. AI recommendation service: Là dịch vụ thông minh thứ hai, chịu trách nhiệm cá nhân hóa trải nghiệm. Lõi của nó là Recommendation Engine. Nó nhận các sự kiện hành vi từ frontend, xử lý và lưu trữ vào PostgreSQL, sau đó cung cấp lại các gợi ý sản phẩm khi được frontend yêu cầu. Dịch vụ này cũng gọi đến Medusa API để lấy thông tin chi tiết của sản phẩm cần gợi ý.

Các luồng dữ liệu chính:

- Luồng 1 & 5 (tương tác cơ bản): Người dùng tương tác với frontend (1), quản trị viên tương tác với Medusa backend (5).

- Luồng 2 (frontend làm trung tâm điều phối):

+ 2a: Với các tác vụ TMĐT thông thường, frontend gọi trực tiếp đến Medusa API.

+ 2b: Khi người dùng chat, frontend gửi tin nhắn đến AI chatbot service.

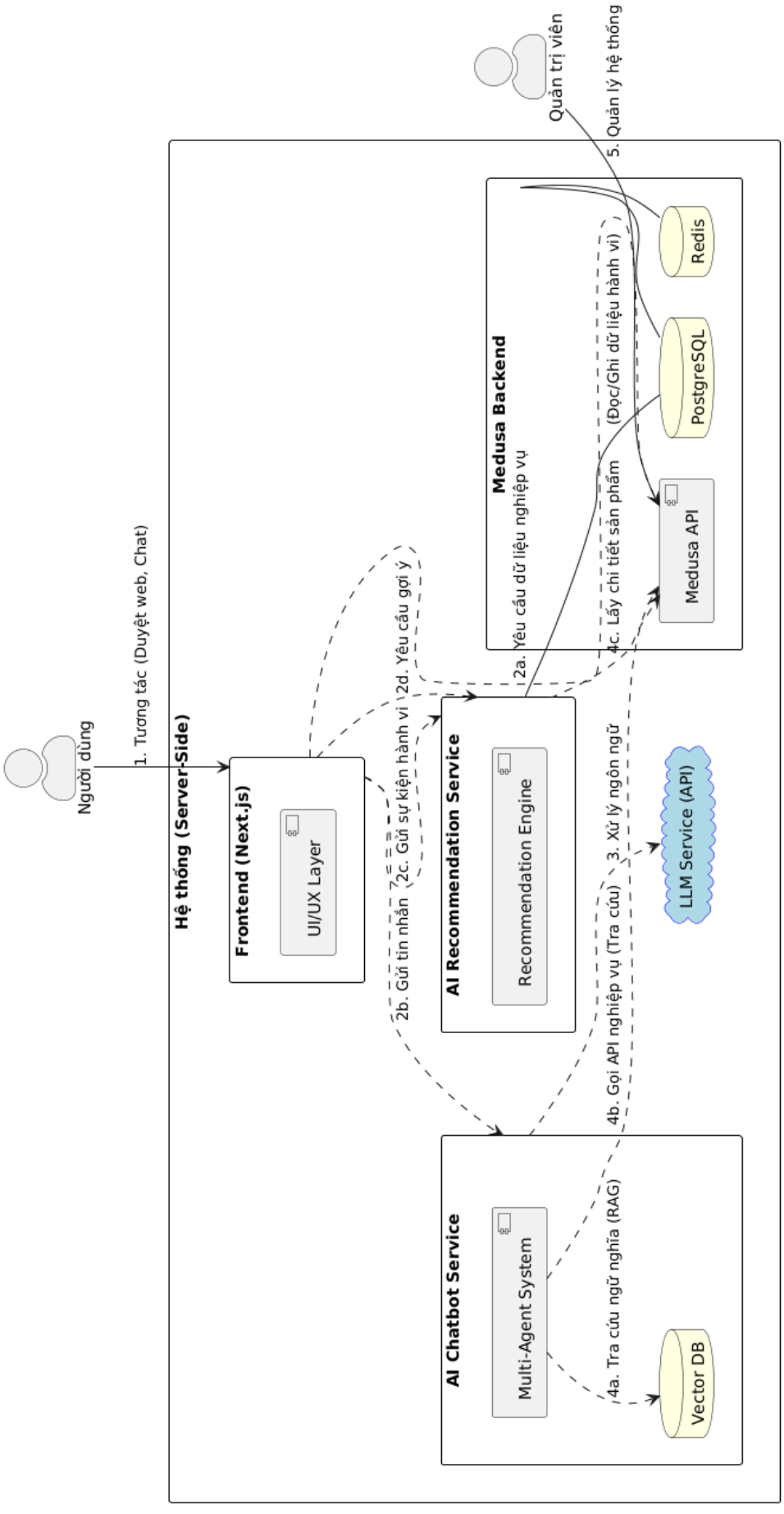
+ 2c & 2d: Frontend gửi các sự kiện hành vi đến AI recommendation service (2c) và yêu cầu danh sách gợi ý từ dịch vụ này (2d).

- Luồng 3 & 4 (AI services thực thi):

+ 3: Chatbot service sử dụng LLM service để xử lý ngôn ngữ.

+ 4a, 4b, 4c: Các dịch vụ AI thực hiện nhiệm vụ của mình bằng cách sử dụng các "công cụ": truy vấn Vector DB (4a), gọi API nghiệp vụ của Medusa (4b, 4c).

Sơ đồ này cho thấy một kiến trúc tách rời, nơi mỗi dịch vụ có một trách nhiệm rõ ràng và giao tiếp với nhau qua các giao diện (API) được xác định trước, thể hiện đúng tinh thần của Microservices.



Hình 3.2: Sơ đồ tương tác giữa các thành phần

3.3.2. Sơ đồ luồng hoạt động của hệ thống Multi-Agent

Đây là sơ đồ chi tiết mô tả quy trình xử lý tin nhắn theo mô hình Pipeline. Thay vì chỉ sử dụng các tác tử rời rạc, hệ thống phân rã quy trình thành 5 giai đoạn chuyên biệt để tối ưu hóa tốc độ xử lý và đảm bảo tính chính xác cho từng tác vụ như tư vấn sản phẩm hay tra cứu đơn hàng.

Giải thích quy trình hoạt động:

Giai đoạn tiền xử lý (Agent 1 - Input Processor): Tin nhắn từ frontend được tiếp nhận, làm sạch dữ liệu thô, phát hiện ngôn ngữ và tải ngữ cảnh hội thoại (session history) để đảm bảo các tác tử sau có đầy đủ thông tin bộ nhớ.

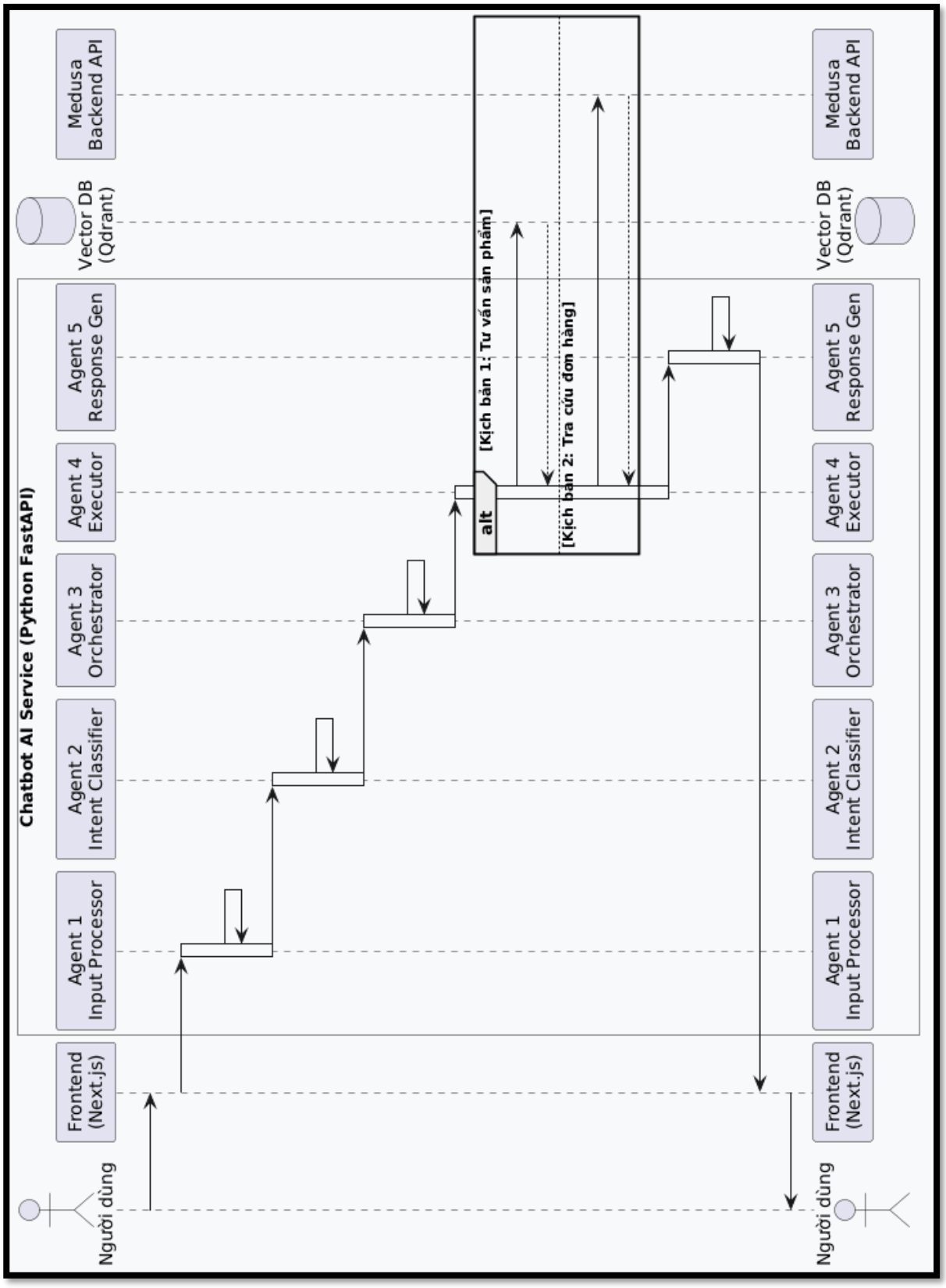
Giai đoạn phân loại (Agent 2 - Intent Classifier): Sử dụng kết hợp giữa quy tắc (Keyword/Decision Tree) và LLM để xác định ý định người dùng. Việc ưu tiên quy tắc giúp hệ thống phản hồi cực nhanh đối với các yêu cầu phổ biến.

Giai đoạn điều phối (Agent 3 - Orchestrator): Kiểm tra quyền hạn người dùng (Guest/Customer) và xác định xem thông tin có bị thiếu không (ví dụ: thiếu mã đơn hàng). Nếu dữ liệu đủ, tác tử này lập kế hoạch thực thi công cụ.

Giai đoạn thực thi (Agent 4 - Executor): Đây là "bàn tay" của hệ thống. Tác tử này trực tiếp gọi API Medusa Backend để lấy dữ liệu đơn hàng hoặc truy vấn Vector Database (Qdrant) để tìm kiếm sản phẩm theo ngữ nghĩa (RAG).

Giai đoạn phản hồi (Agent 5 - Response Generator): Tổng hợp kết quả từ bước thực thi. Với các yêu cầu thông thường, tác tử sử dụng Template để trả lời ngay lập tức (tiết kiệm chi phí), chỉ sử dụng LLM cho các câu trả lời cần sự sáng tạo và tự nhiên.

Hệ thống kết thúc bằng việc gửi phản hồi hoàn chỉnh về frontend để hiển thị cho người dùng.



Hình 3.3: Sơ đồ luồng hoạt động của hệ thống Multi-Agent

3.3.3. Mô tả chi tiết quy trình xử lý nội bộ của các tác tử

Để đảm bảo hệ thống vận hành với hiệu suất cao và tính minh bạch trong xử lý, mỗi tác tử trong chuỗi Pipeline được thiết kế với một quy trình logic độc lập. Dưới đây là mô tả chi tiết quy trình xử lý nội bộ của 5 tác tử cốt lõi:

3.3.3.1. Tác tử tiền xử lý (Agent 1: Input Processor)

Tác tử này chịu trách nhiệm làm sạch dữ liệu đầu vào và thiết lập môi trường ngữ cảnh cho toàn bộ chuỗi xử lý phía sau.

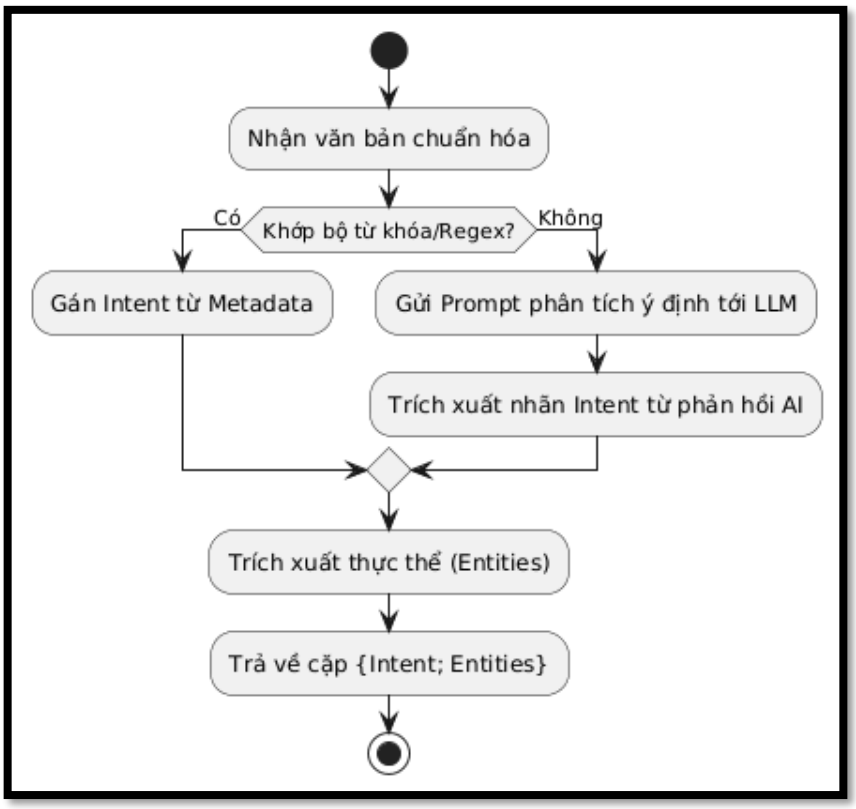
Quy trình logic: Tiếp nhận văn bản thô từ người dùng -> Thực hiện chuẩn hóa (Normalization) bằng biểu thức chính quy để loại bỏ ký tự nhiễu -> Sử dụng thư viện nhận dạng để xác định ngôn ngữ (Tiếng Việt/Tiếng Anh) -> Truy vấn cơ sở dữ liệu để tải lại lịch sử hội thoại của phiên làm việc hiện tại (Context) -> Đóng gói dữ liệu thành một đối tượng trung gian đồng nhất.

Hình 3.4: Sơ đồ logic Agent Input Processor

3.3.3.2. Tác tử phân loại ý định (Agent 2: Intent Classifier)

Đây là giai đoạn then chốt nhằm xác định mục tiêu của người dùng thông qua mô hình lai (Hybrid).

Quy trình logic: Đối soát văn bản với bộ quy tắc Keyword và Regex mẫu (Pattern Matching) -> Nếu tìm thấy sự trùng khớp (ví dụ: "giá", "tìm", "đơn hàng"), gán nhãn Intent ngay lập tức nhằm tối ưu tốc độ phản hồi -> Trường hợp ý định mơ hồ, hệ thống thực hiện cơ chế "LLM Fallback" để phân tích ngữ nghĩa sâu -> Trích xuất các thực thể đi kèm (Entities) như tên sản phẩm hoặc mã đơn hàng.

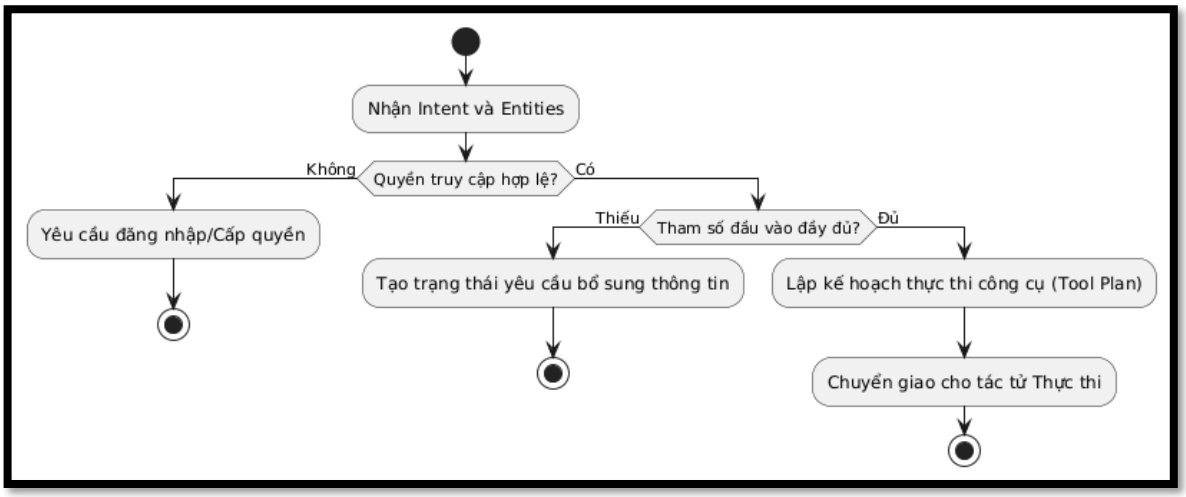


Hình 3.5: Sơ đồ logic Agent Intent Classifier

3.3.3.3. Tác tử điều phối (Agent 3: Orchestrator)

Đóng vai trò kiểm soát logic nghiệp vụ và đảm bảo tính an toàn cho hệ thống TMĐT.

Quy trình logic: Tiếp nhận nhãn Intent -> Thực hiện kiểm soát truy cập dựa trên vai trò (RBAC) để xác định quyền hạn của khách hàng -> Kiểm tra tính đầy đủ của các thực thể cần thiết (ví dụ: tra cứu đơn hàng cần mã ID) -> Nếu thiếu dữ liệu, tác tử này sẽ chuyển trạng thái sang "Hỏi bổ sung" -> Nếu dữ liệu đã đầy đủ, Orchestrator sẽ lập kế hoạch gọi công cụ (Tool Plan) cho tác tử tiếp theo.

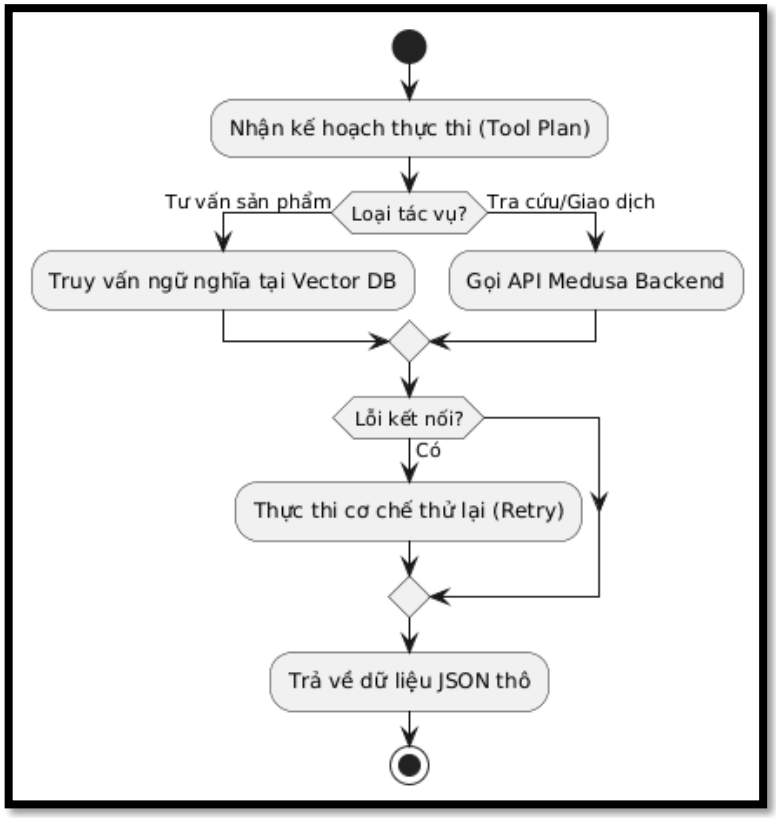


Hình 3.6: Sơ đồ logic Agent Orchestrator

3.3.3.4. Tác tử thực thi (Agent 4: Executor)

Chịu trách nhiệm tương tác trực tiếp với các tầng dữ liệu và API ngoại vi.

Quy trình logic: Phân tích kế hoạch từ Orchestrator -> Điều hướng yêu cầu đến đúng đích: Truy vấn Qdrant (Vector Database) nếu là tư vấn sản phẩm (RAG) hoặc gọi REST API của Medusa Backend nếu là tra cứu đơn hàng -> Tiếp nhận dữ liệu thô (Raw Data) -> Thực hiện xử lý ngoại lệ và cơ chế thử lại (Retry logic) nếu có lỗi kết nối.

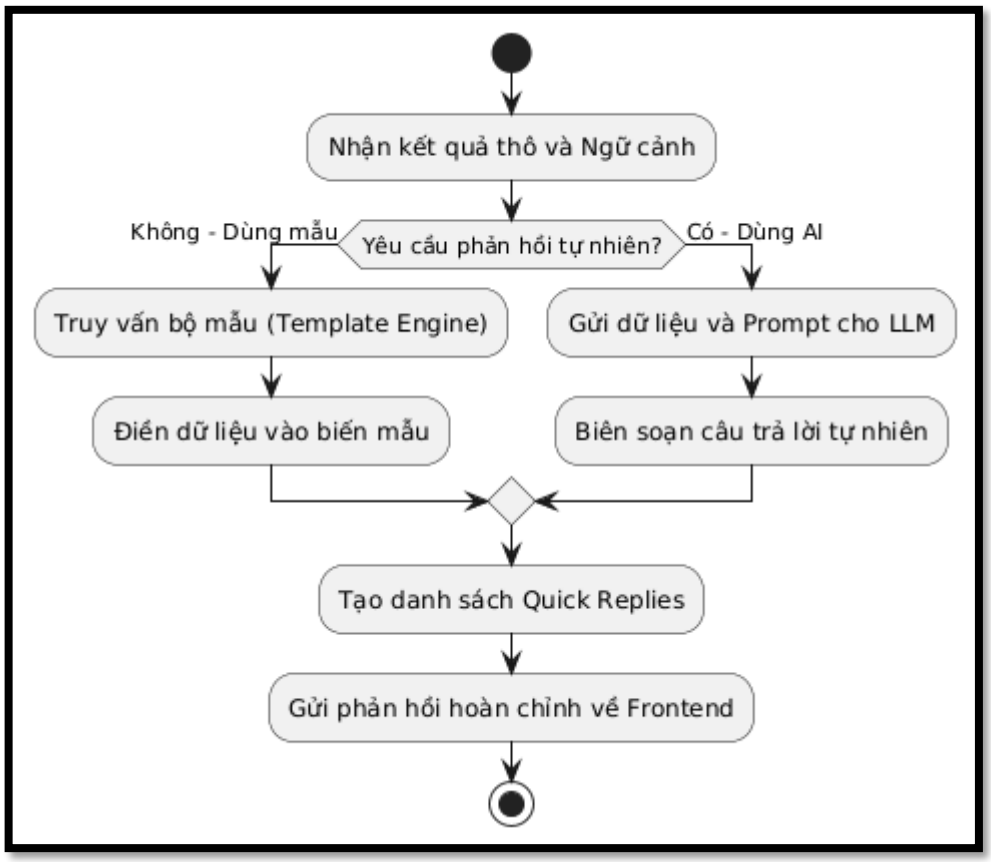


Hình 3.7: Sơ đồ logic Agent Executor

3.3.3.5. Tác tử Tạo phản hồi (Agent 5: Response Generator)

Chịu trách nhiệm biên soạn câu trả lời cuối cùng cho người dùng với trải nghiệm tốt nhất.

Quy trình logic: Tiếp nhận dữ liệu thô từ Executor -> Phân tích độ phức tạp của kết quả -> Nếu kết quả có cấu trúc đơn giản, hệ thống điền dữ liệu vào các mẫu câu có sẵn (Templates) thông qua Jinja2 để phản hồi tức thì -> Nếu kết quả cần sự diễn giải linh hoạt, hệ thống sẽ gửi dữ liệu cho LLM để tổng hợp thành văn bản tự nhiên -> Đóng gói phản hồi kèm các hành động gợi ý (Quick Replies) cho Frontend.



Hình 3.8: Sơ đồ logic Agent Response Generator

Việc chia nhỏ quy trình vào từng tác tử như trên mang lại 3 lợi ích cho hệ thống:

- Tính Module hóa: Dễ dàng bảo trì và nâng cấp logic của một tác tử mà không ảnh hưởng đến toàn bộ hệ thống.

- Tối ưu hiệu năng: Việc ưu tiên xử lý bằng Quy tắc (Rules) ở các giai đoạn phân loại và phản hồi giúp hệ thống giảm thiểu độ trễ và chi phí API đáng kể.

- Khả năng kiểm soát: Orchestrator đóng vai trò bộ lọc trung gian, ngăn chặn các hành vi truy cập dữ liệu trái phép trước khi tác tử thực thi được kích hoạt.

3.3.4. Sơ đồ luồng hoạt động của hệ thống Recommendation

Hệ thống gợi ý hoạt động dựa trên hai luồng chính, diễn ra song song:

(1) Luồng theo dõi hành vi, sở thích của người dùng một cách bất đồng bộ

(2) Luồng cung cấp gợi ý để trả về danh sách sản phẩm cá nhân hóa một cách đồng bộ khi người dùng yêu cầu.

Sơ đồ sau mô tả chi tiết hai quy trình vận hành chính của hệ thống gợi ý:

(1) Luồng 1: Theo dõi hành vi và học sở thích người dùng (bất đồng bộ)

Đây là quá trình hệ thống thu thập dữ liệu đầu vào. Nó diễn ra ngầm mỗi khi người dùng tương tác trên trang web.

1. Tương tác: Người dùng thực hiện một hành động có ý nghĩa (xem sản phẩm, thêm vào giỏ hàng, v.v.).

2. Gửi sự kiện: Frontend bắt lấy sự kiện này và gửi một yêu cầu POST /track đến Recommendation Service, chứa thông tin về người dùng, sản phẩm và loại hành động.

3. Ghi nhận và xử lý: Recommendation Service nhận sự kiện và ghi lại vào bảng rec_user_interactions trong cơ sở dữ liệu PostgreSQL. Dựa trên các tương tác này, hệ thống có thể chạy các tác vụ nền để cập nhật lại hồ sơ sở thích của người dùng (ví dụ: người này thích danh mục "áo sơ mi").

4. Phản hồi nhanh: Dịch vụ sẽ phản hồi ngay lập tức (200 OK) cho Frontend để không làm gián đoạn trải nghiệm của người dùng.

(2) Luồng 2: Cung cấp gợi ý cá nhân hóa (đồng bộ)

Đây là quá trình hệ thống trả về kết quả đầu ra. Nó được kích hoạt khi người dùng truy cập một trang cần hiển thị sản phẩm gợi ý.

1. Yêu cầu gợi ý: Frontend gửi một yêu cầu GET /recommendations đến Recommendation Service.

2. Kiểm tra Cache: Đây là bước tối ưu hiệu năng quan trọng nhất. Dịch vụ sẽ kiểm tra trong Redis Cache trước tiên xem đã có sẵn danh sách gợi ý cho người dùng này hay chưa.

3. Trường hợp Cache hit (có dữ liệu): Nếu tìm thấy, dữ liệu sẽ được trả về ngay lập tức cho frontend. Đây là luồng xử lý cực nhanh (thường dưới 50ms), đảm bảo trải nghiệm người dùng mượt mà.

4. Trường hợp Cache miss (không có dữ liệu):

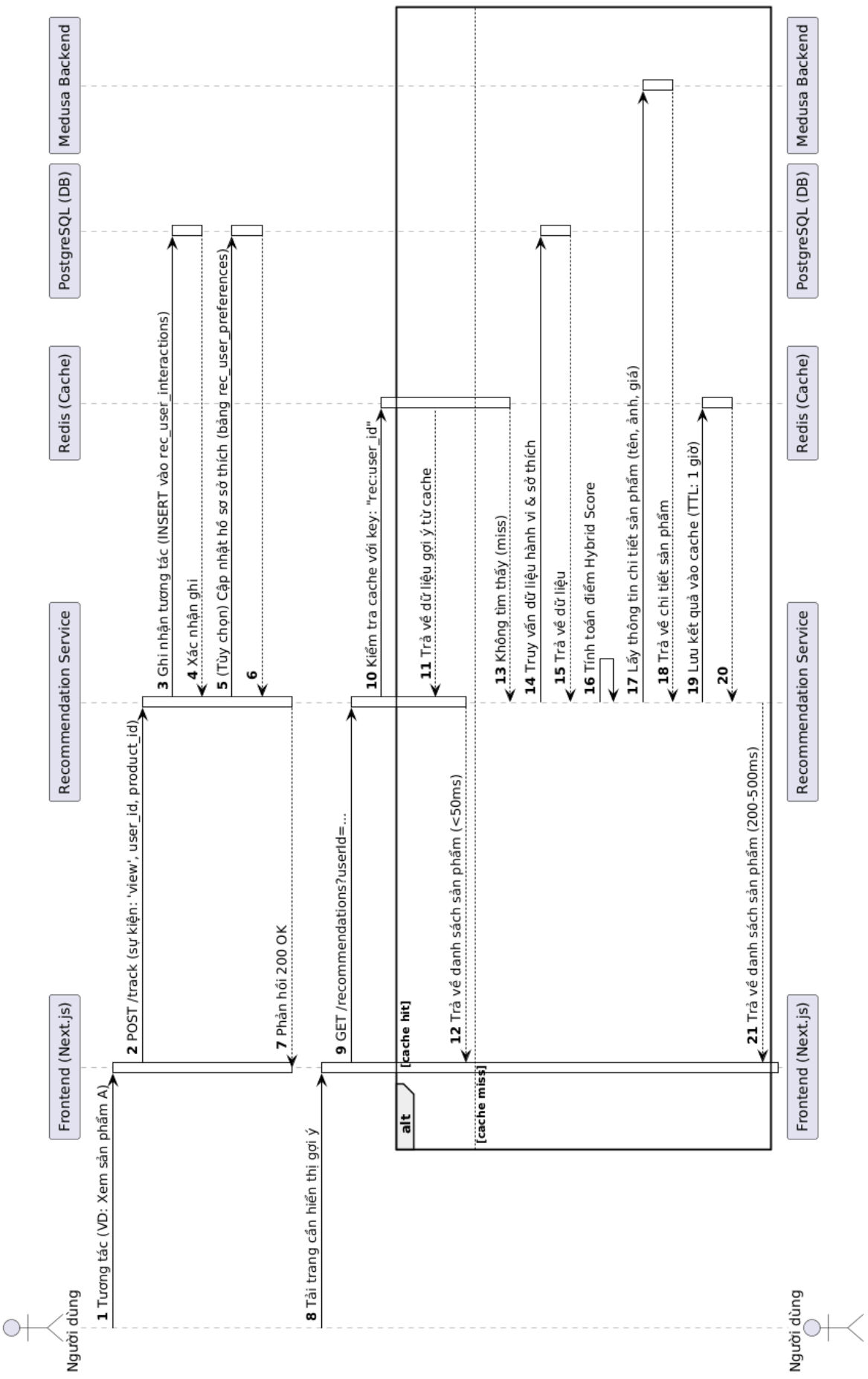
- Recommendation Service sẽ truy vấn cơ sở dữ liệu PostgreSQL để lấy lịch sử hành vi và hồ sơ sở thích của người dùng.

- Recommendation Engine thực hiện tính toán điểm số gợi ý dựa trên thuật toán Hybrid.

- Sau khi có danh sách các mã sản phẩm (product_id) được gợi ý, dịch vụ sẽ gọi API của Medusa backend để lấy thông tin chi tiết (tên, ảnh, giá) nhằm làm phong phú dữ liệu trả về.

- Kết quả cuối cùng được lưu vào Redis Cache với một thời gian sống (TTL) nhất định (ví dụ: 1 giờ) để các yêu cầu sau có thể tái sử dụng.

- Cuối cùng, danh sách sản phẩm hoàn chỉnh được trả về cho frontend. Luồng này sẽ chậm hơn (200-500ms) do phải thực hiện nhiều bước tính toán.



Hình 3.9: Sơ đồ luồng hoạt động của hệ thống Recommendation

3.4. Thiết kế cơ sở dữ liệu

Kiến trúc cơ sở dữ liệu của hệ thống được thiết kế theo hướng module hóa, kết hợp giữa cơ sở dữ liệu quan hệ để lưu trữ dữ liệu nghiệp vụ có cấu trúc và cơ sở dữ liệu vector để phục vụ các tác vụ tìm kiếm ngữ nghĩa thông minh.

3.4.1. Tận dụng lược đồ cơ sở dữ liệu của Medusa

Thay vì thiết kế lại từ đầu, hệ thống tận dụng tối đa lược đồ cơ sở dữ liệu (database schema) quan hệ đã được tối ưu hóa và kiểm chứng của MedusaJS. Medusa sử dụng PostgreSQL làm hệ quản trị cơ sở dữ liệu chính, cung cấp một cấu trúc vững chắc để quản lý toàn bộ vòng đời thương mại điện tử. Việc này không chỉ giúp tiết kiệm thời gian phát triển mà còn đảm bảo tính toàn vẹn và nhất quán của dữ liệu nghiệp vụ.

Các bảng chính được sử dụng trực tiếp từ Medusa bao gồm:

product và các bảng liên quan (product_variant, product_option): Lưu trữ toàn bộ thông tin về sản phẩm, từ tên, mô tả, hình ảnh, giá cả cho đến các biến thể (ví dụ: kích thước, màu sắc). Đây là nguồn dữ liệu đầu vào cho cả việc hiển thị trên frontend và quá trình embedding cho trợ lý ảo.

order và các bảng liên quan (line_item, shipping_method): Lưu trữ thông tin chi tiết về các đơn hàng đã được tạo, bao gồm sản phẩm trong đơn, thông tin khách hàng, trạng thái đơn hàng (ví dụ: chờ xử lý, đang giao, đã hoàn thành). Dữ liệu này được Order Agent truy xuất qua API để cung cấp cho người dùng.

customer: Lưu trữ thông tin tài khoản của khách hàng, bao gồm email, mật khẩu (đã được băm), và địa chỉ.

cart: Quản lý trạng thái giỏ hàng của người dùng trước khi họ tiến hành thanh toán.

3.4.2. Thiết kế cơ sở dữ liệu cho Chatbot Service

Để phục vụ cho hoạt động của AI Chatbot Service, một cơ sở dữ liệu riêng biệt được thiết kế để lưu trữ lịch sử hội thoại và các dữ liệu liên quan. Việc tách riêng cơ sở dữ liệu này giúp dịch vụ AI hoạt động độc lập và không làm ảnh hưởng đến hiệu năng của cơ sở dữ liệu nghiệp vụ chính.

Schema này bao gồm 3 bảng chính: chatbot_context, chatbot_responses, và chatbot_analytics.

Bảng 3.3: Bảng chatbot_context (lưu trữ ngữ cảnh phiên trò chuyện)

Tên cột	Kiểu dữ liệu	Ràng buộc	Mô tả
session_id	TEXT	PRIMARY KEY	Mã định danh duy nhất cho mỗi phiên trò chuyện.
user_id	TEXT	NULLABLE	Mã định danh của khách hàng (nếu đã đăng nhập).
conversation_history	JSONB		Lưu trữ toàn bộ lịch sử tin nhắn (người dùng và bot) của phiên dưới dạng JSON.
status	TEXT		Trạng thái của phiên (ví dụ: active, escalated, closed).
created_at	TIMESTAMP		Thời điểm tin nhắn được tạo.
updated_at	TIMESTAMP		Thời điểm phiên có tin nhắn cuối cùng.

Mục đích: Bảng này là trái tim của việc duy trì trạng thái hội thoại. Nó cho phép chatbot "nhớ" được các tin nhắn trước đó trong cùng một phiên, giúp các cuộc trò chuyện trở nên tự nhiên và liền mạch hơn.

Bảng 3.4: Bảng chatbot_responses (lưu trữ các mẫu phản hồi)

Tên cột	Kiểu dữ liệu	Ràng buộc	Mô tả
intent_key	TEXT	PRIMARY KEY	Khóa định danh cho một ý định cụ thể (ví dụ: PRODUCT.SEARCH_SUCCESS).
template_text	TEXT		Khóa định danh cho một ý định cụ thể (ví

			dụ: PRODUCT.SEARCH_SUCCESS).
language_code	VARCHAR(5)		Mã ngôn ngữ của mẫu câu (ví dụ: vi, en).

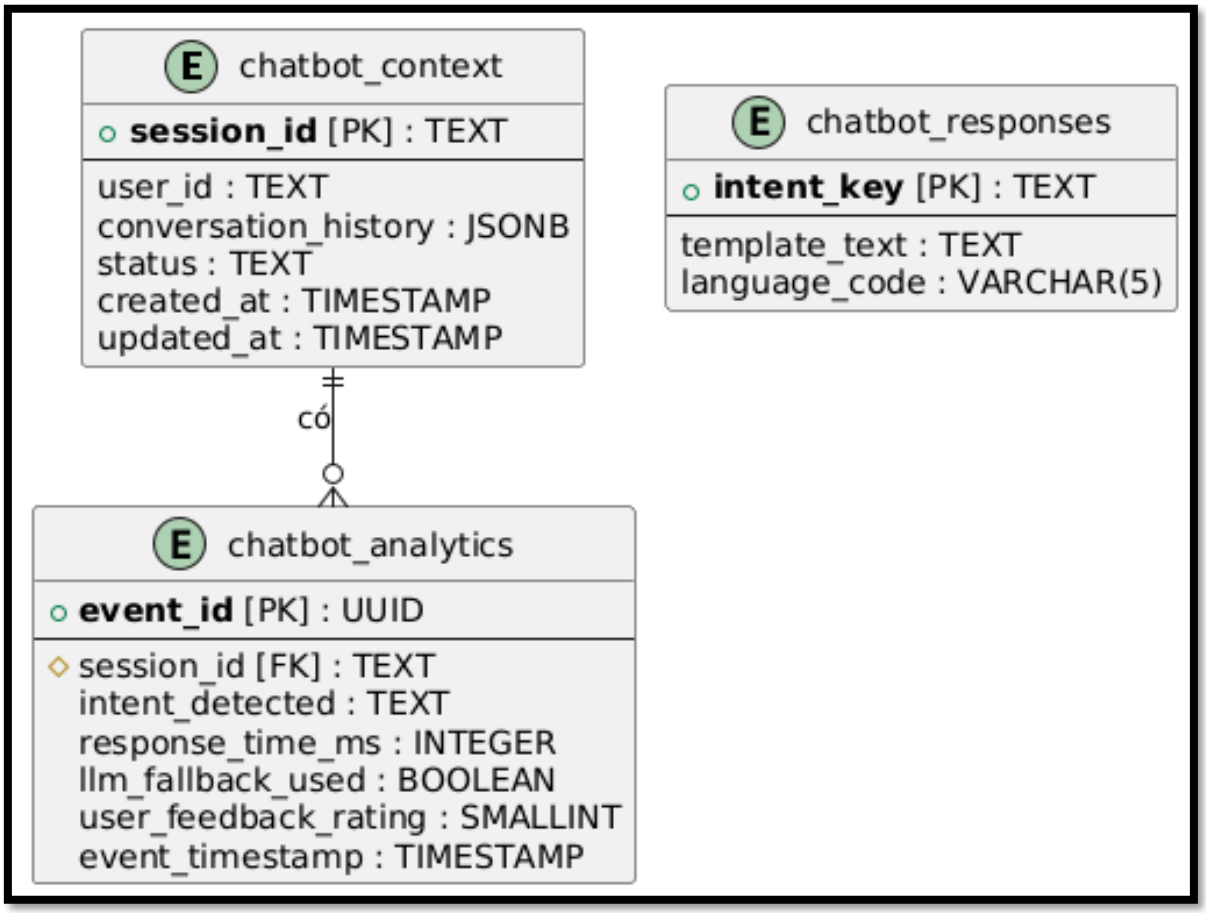
Mục đích: Bảng này phục vụ cho **Response Generator Agent**. Bằng cách lưu trữ các mẫu câu trả lời được kịch bản hóa trước, hệ thống có thể phản hồi tới 90% các yêu cầu của người dùng một cách tức thì và nhất quán mà không cần gọi đến LLM, giúp tiết kiệm chi phí và tăng tốc độ.

Bảng 3.5: Bảng chatbot_analytics (lưu trữ dữ liệu phân tích)

Tên cột	Kiểu dữ liệu	Ràng buộc	Mô tả
event_id	UUID	PRIMARY KEY	Mã định danh duy nhất cho mỗi sự kiện phân tích.
event_id	TEXT	FOREIGN KEY	Liên kết đến phiên trò chuyện trong bảng chatbot_context.
intent_detected	TEXT		Ý định được hệ thống phân loại.
response_time_ms	INTEGER		Thời gian phản hồi của bot (tính bằng mili giây).
llm_fallback_used	BOOLEAN		Ghi nhận việc có phải sử dụng LLM fallback hay không.
user_feedback_rating	SMALLINT	NULLABLE	Điểm đánh giá của người dùng cho phản hồi (ví dụ: 1-5 sao).
event_timestamp	TIMESTAMP		Thời điểm sự kiện xảy ra.

Mục đích: Bảng này ghi lại các chỉ số hiệu suất quan trọng của chatbot. Dữ liệu từ đây giúp các nhà phát triển và quản trị viên có thể đánh giá được hiệu quả của hệ thống (ví dụ: ý định nào được hỏi nhiều nhất, thời gian phản hồi trung bình, tỷ lệ phải dùng LLM) để đưa ra các phương án cải thiện.

Sơ đồ trên minh họa kiến trúc dữ liệu được thiết kế chuyên biệt để hỗ trợ hoạt động của Chatbot Service. Schema này đảm bảo chatbot có thể duy trì ngữ cảnh, phản hồi nhanh chóng và cung cấp dữ liệu cho việc phân tích và cải tiến.



Hình 3.10: Sơ đồ ERD cho cơ sở dữ liệu của Chatbot Service

3.4.3. Thiết kế cấu trúc dữ liệu cho Vector Database (RAG)

Để sales agent có thể thực hiện chức năng tư vấn sản phẩm thông minh, một Vector Database (như Qdrant hoặc Pinecone) được sử dụng để lưu trữ phiên bản "vector hóa" của dữ liệu sản phẩm. Đây không phải là một bảng quan hệ truyền thống mà là một collection các vector.

Collection: products

Mỗi đối tượng (document) trong collection này sẽ bao gồm hai phần chính:

- Vector: Một mảng số thực được tạo ra bởi mô hình embedding. Vector này đại diện cho ngữ nghĩa của sản phẩm.

- Payload (Metadata): Một đối tượng JSON chứa các thông tin có cấu trúc, được dùng để lọc kết quả và tham chiếu ngược về sản phẩm gốc.

Cấu trúc Payload mẫu:

```
{
  "product_id": "prod_01G8Z...",  // ID sản phẩm trong Medusa
  "product_handle": "ao-so-mi-trang", // Slug của sản phẩm
  "text_to_embed": "Áo sơ mi nam màu trắng, chất liệu cotton thoáng mát, phù hợp cho công sở. Giá: 500,000 VND.", // Văn bản dùng để tạo vector
  "category": "áo-sơ-mi",
  "price": 500000,
  "inventory_quantity": 50
}
```

Hình 3.11: Mô tả Payload mẫu dữ liệu lưu trữ Vector Database

Mục đích: Cấu trúc này cho phép sales agent thực hiện các truy vấn tìm kiếm ngữ nghĩa phức tạp. Ví dụ, khi người dùng hỏi "tìm đồ mặc đi làm", hệ thống sẽ tìm kiếm các vector có ngữ nghĩa tương đồng với "công sở", "lịch sự", sau đó có thể lọc thêm theo các tiêu chí trong payload như giá hoặc số lượng tồn kho trước khi đưa ra gợi ý.

3.4.4. Thiết kế dữ liệu cho Recommendation Service

Khác với chatbot, recommendation service yêu cầu một schema phức tạp hơn để quản lý nhiều khía cạnh: từ việc ghi lại từng tương tác nhỏ của người dùng, tổng hợp thành hồ sơ sở thích, cho đến việc lưu trữ các mối quan hệ tương đồng giữa sản phẩm và kết quả gợi ý đã được tính toán.

Schema này được thiết kế với 7 bảng chính, mỗi bảng phục vụ một mục đích riêng biệt trong quy trình gợi ý.

Bảng 3.6: rec_user_interactions (ghi nhận tương tác thô của người dùng)

Tên cột	Kiểu dữ liệu	Ràng buộc	Mô tả
id	TEXT	PRIMARY KEY	Mã định danh duy nhất cho mỗi sự kiện tương tác.
user_id	TEXT		Mã định danh của người dùng (nếu đã đăng nhập).

session_id	TEXT		Mã định danh của phiên (cho người dùng vãng lai).
product_id	TEXT		Mã sản phẩm được tương tác.
interaction_type	TEXT		Loại tương tác ví dụ: view, add_to_cart, purchase).
weight	FLOAT		Trọng số của tương tác (ví dụ: purchase có trọng số cao hơn view).
metadata	JSONB		Dữ liệu bổ sung như giá, danh mục tại thời điểm tương tác.
created_at	TIMESTAMP		Thời điểm tương tác.

Mục đích: Đây là bảng ghi "nhật ký" toàn bộ hành vi của người dùng. Nó là nguồn dữ liệu thô và quan trọng nhất để huấn luyện và vận hành toàn bộ hệ thống gợi ý.

Bảng 3.7: rec_user_preferences (lưu trữ hồ sơ sở thích người dùng)

Tên cột	Kiểu dữ liệu	Ràng buộc	Mô tả
id	TEXT	PRIMARY KEY	
user_id	TEXT		Mã định danh của người dùng (nếu đã đăng nhập).
category	TEXT		Tên danh mục sản phẩm.
score	FLOAT		Điểm số thể hiện mức độ yêu thích của người dùng đối với danh mục này (0-1).
interaction_count	INT		Số lần người dùng đã tương tác

			với các sản phẩm trong danh mục.
last_updated	TIMESTAMP		Thời điểm cập nhật cuối cùng.

Mục đích: Bảng này là kết quả của quá trình "học" từ dữ liệu trong bảng rec_user_interactions. Thay vì phải xử lý lại toàn bộ lịch sử tương tác mỗi lần, hệ thống sẽ truy vấn bảng này để nhanh chóng biết được sở thích của người dùng, phục vụ cho thuật toán Content-Based.

Bảng 3.8: rec_product_similarities (lưu trữ độ tương đồng giữa các sản phẩm)

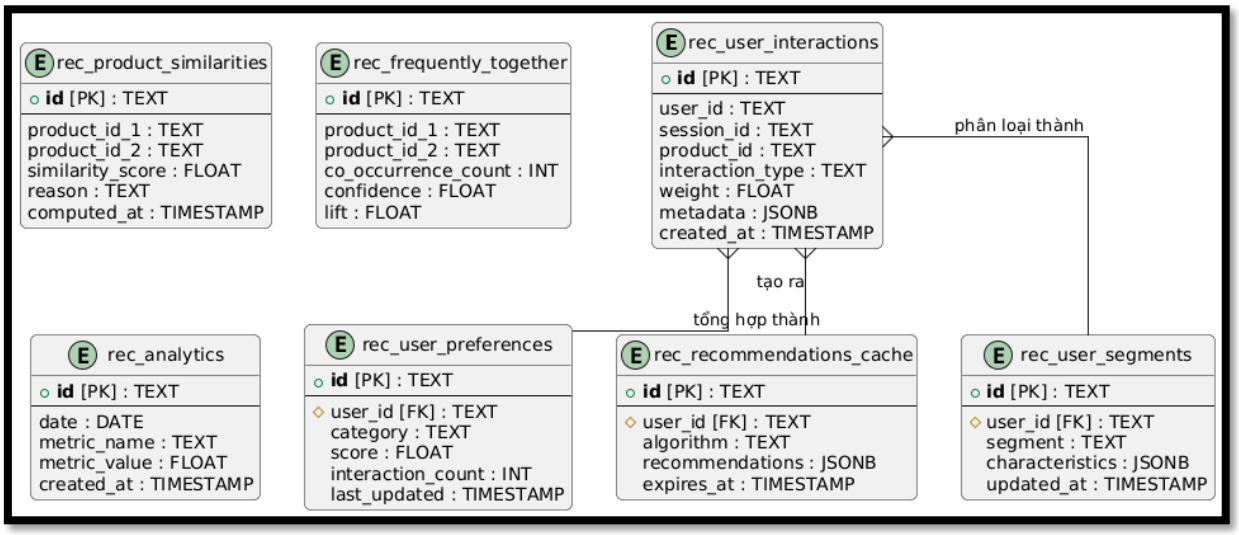
Tên cột	Kiểu dữ liệu	Ràng buộc	Mô tả
id	TEXT	PRIMARY KEY	
product_id_1	TEXT		Mã sản phẩm thứ nhất.
product_id_2	TEXT		Mã sản phẩm thứ hai.
similarity-score	FLOAT		Điểm số tương đồng giữa hai sản phẩm (0-1).
reason	TEXT		Lý do tương đồng (ví dụ: cùng danh mục, cùng thuộc tính).
computed_at	TIMESTAMP		Thời điểm độ tương đồng được tính toán.

Mục đích: Bảng này lưu trữ trước kết quả của các phép tính toán tương đồng phức tạp. Việc này được thực hiện bởi một tác vụ nền (batch job) định kỳ, giúp cho quá trình gợi ý sản phẩm tương tự diễn ra gần như tức thì.

Bảng 3.9: rec_recommendations_cache (bảng lưu trữ đệm kết quả gợi ý)

Tên cột	Kiểu dữ liệu	Ràng buộc	Mô tả
id	TEXT	PRIMARY KEY	
user_id	TEXT		Mã định danh của người dùng.
algorithm	TEXT		Tên thuật toán đã được sử dụng (ví dụ: hybrid, trending).
recommendations	JSONB		Danh sách các sản phẩm được gợi ý, lưu dưới dạng JSON.
expires_at	TIMESTAMP		Thời điểm cache hết hạn.

Mục đích: Đây là bảng tối ưu hiệu năng quan trọng nhất, hoạt động như một lớp cache. Thay vì phải tính toán lại từ đầu, hệ thống sẽ ưu tiên lấy kết quả gợi ý đã được tính sẵn từ đây, giúp giảm độ trễ API xuống dưới 50ms.



Hình 3.12: Sơ đồ ERD cho cơ sở dữ liệu của Recommendation Service

Giải thích sơ đồ:

- rec_user_interactions là bảng trung tâm, ghi lại mọi hành vi. Dữ liệu từ bảng này là đầu vào cho các bảng khác.

- rec_user_preferences và rec_user_segments là các bảng được "tổng hợp" hoặc "phân loại" từ dữ liệu tương tác thô để tạo ra hồ sơ người dùng.

- rec_product_similarities và rec_frequently_together là các bảng chứa dữ liệu được tính toán trước về mối quan hệ giữa các sản phẩm.

- rec_recommendations_cache là bảng kết quả cuối cùng, được "tạo ra" từ quá trình xử lý dữ liệu từ các bảng khác.

- rec_analytics là một bảng độc lập dùng để ghi lại các chỉ số hiệu suất theo thời gian.

Sơ đồ này thể hiện rõ kiến trúc dữ liệu "data-driven", nơi các tương tác thô được chuyển hóa thành các insight (sở thích, phân khúc) và các kết quả tính toán trước (độ tương đồng, cache) để phục vụ cho việc gợi ý sản phẩm một cách nhanh chóng và hiệu quả.

CHƯƠNG 4. TRIỂN KHAI VÀ KẾT QUẢ THỰC NGHIỆM

Sau khi hoàn thiện bản thiết kế tại Chương 3, chương này tập trung vào quá trình triển khai kỹ thuật và đánh giá hiệu năng của hệ thống. Nội dung bao gồm việc chuẩn bị dữ liệu sản phẩm thực tế, cài đặt các khối dịch vụ. Trong đó, quy trình hiện thực hóa kiến trúc Multi-Agent cho Chatbot và cơ chế cá nhân hóa của Recommendation Service sẽ được trình bày chi tiết. Chương này kết thúc bằng phần phân tích kết quả thực nghiệm dựa trên các kịch bản kiểm thử cụ thể, nhằm đánh giá mức độ đáp ứng của hệ thống so với các mục tiêu đã đề ra ban đầu.

4.1. Chuẩn bị tài nguyên

4.1.1. Công cụ và công nghệ sử dụng

Để đảm bảo quá trình phát triển diễn ra thuận lợi và hiệu quả, một môi trường làm việc phù hợp đã được thiết lập với các công cụ và công nghệ chính như sau:

Bảng 4.1: Các công cụ và công nghệ chính được sử dụng trong đề tài

Lĩnh vực	Công nghệ / Công cụ	Mục đích sử dụng
Ngôn ngữ & Nền tảng	Node.js (v18+), TypeScript	Xây dựng backend và frontend.
	Python (v3.9+)	Ngôn ngữ để phát triển logic cho AI Chatbot và Recommendation Service.
Cơ sở dữ liệu & Caching	PostgreSQL (v14+)	Lưu trữ dữ liệu sản phẩm cho TMĐT và dữ liệu cho các dịch vụ AI.
	Redis (v6+)	Sử dụng cho caching và hàng đợi trong Medusa.
	Qdrant (Vector Database)	Lưu trữ các vector nhúng của dữ liệu sản phẩm.

Dịch vụ AI & Thư viện	LangChain / LlamaIndex	Framework nền tảng để xây dựng logic, chuỗi xử lý và quản lý các Agent trong Chatbot Service.
	LLM Service (OpenAI API,...)	Cung cấp năng lực xử lý và tạo sinh ngôn ngữ tự nhiên cho các Agent.
Công cụ Phát triển & Vận hành	Visual Studio Code, Git	Môi trường lập trình và quản lý phiên bản mã nguồn.
	Docker & Docker Compose	Container hóa các dịch vụ để đảm bảo tính nhất quán và dễ dàng triển khai trên các môi trường.
	npm / yarn, pip	Các công cụ quản lý gói (package manager) cho hệ sinh thái Node.js và Python.

4.1.2. Thu thập và xử lý dữ liệu sản phẩm

Để đảm bảo hệ thống mang tính thực tiễn và có đủ độ phức tạp cho việc thực nghiệm thuật toán gợi ý và trợ lý ảo, ta sử dụng bộ dữ liệu thương mại điện tử thực tế.

4.1.2.1. Nguồn dữ liệu (data source):

Dữ liệu sản phẩm được trích xuất và chọn lọc từ bộ dữ liệu Amazon Sales Dataset được cung cấp công khai trên nền tảng Kaggle. Đây là bộ dữ liệu chứa thông tin chi tiết về các sản phẩm bán chạy trên nền tảng Amazon, bao gồm tên sản phẩm, danh mục, giá cả, đánh giá và mô tả chi tiết.

4.1.2.2. Quy trình trích xuất và chuẩn hóa:

Do cấu trúc dữ liệu thô từ Amazon không tương thích hoàn toàn với lược đồ của MedusaJS và yêu cầu của tác tử AI, tôi thực hiện các bước tiền xử lý sau:

- Lọc dữ liệu: Lựa chọn các nhóm sản phẩm phù hợp với định hướng cửa hàng (tập trung vào ngành hàng thời trang, balo và phụ kiện điện tử).

- Chuẩn hóa thuộc tính:

+ Mapping ID: Chuyển đổi mã sản phẩm Amazon sang định dạng prod_... của MedusaJS.

+ Xử lý Handle: Tự động tạo handle (URL slug) từ tên sản phẩm để tối ưu SEO cho Frontend.

+ Đơn vị tiền tệ: Chuyển đổi giá từ Rupee (INR) sang Việt Nam Đồng (VND) và USD theo tỷ giá thực tế để phù hợp với thị trường mục tiêu.

- Làm giàu dữ liệu: Bổ sung các nhãn và mô tả tóm tắt cho từng sản phẩm để cung cấp nguyên liệu cho quá trình tạo Vector Embedding trong cơ sở dữ liệu Qdrant.

4.1.2.3. Thông số dữ liệu sau xử lý:

Dữ liệu sau khi làm sạch được đóng gói thành tệp products.csv và nhập (import) vào hệ thống thông qua Medusa Admin Dashboard.

- Số lượng sản phẩm: hơn 300 sản phẩm mẫu tiêu biểu.

- Các trường thông tin cốt lõi: id, title, handle, description, price, category, image_url, …

Việc sử dụng dữ liệu thực tế từ Amazon giúp hệ thống trợ lý ảo có khả năng tư vấn dựa trên các thuộc tính sản phẩm có thật, đồng thời giúp thuật toán gợi ý Hybrid phản ánh chính xác hơn hành vi mua sắm trong môi trường thương mại điện tử chuyên nghiệp.

Bảng 4.1: Mẫu dữ liệu sau khi xử lý

ID	Mã sản phẩm	Tên sản phẩm	Handle	Danh mục	Giá bán
1	prod_01KCC54GM0BVS	JanSport Big Student Laptop Backpack	jansport-big-student	Backpack	1.250.000
2	prod_01KCC54GM1XZF	Superbreak One Lightweight	superbreak-one	Backpack	850.000

		Backpack			
3	prod_01KCC54GMJF0P	Men's Regular Fit Polo Shirt	mens-regular-polo	Shirt	450.000
4	prod_01KCC54GM67RW	35L Large Travel Backpack (Waterproof)	35l-travel-backpack	Backpack	1.500.000
5	prod_01KCC54GMYC6H	Small Leather Cosmetic Travel Pouch	leather-cosmetic-pouch	Accessory	320.000

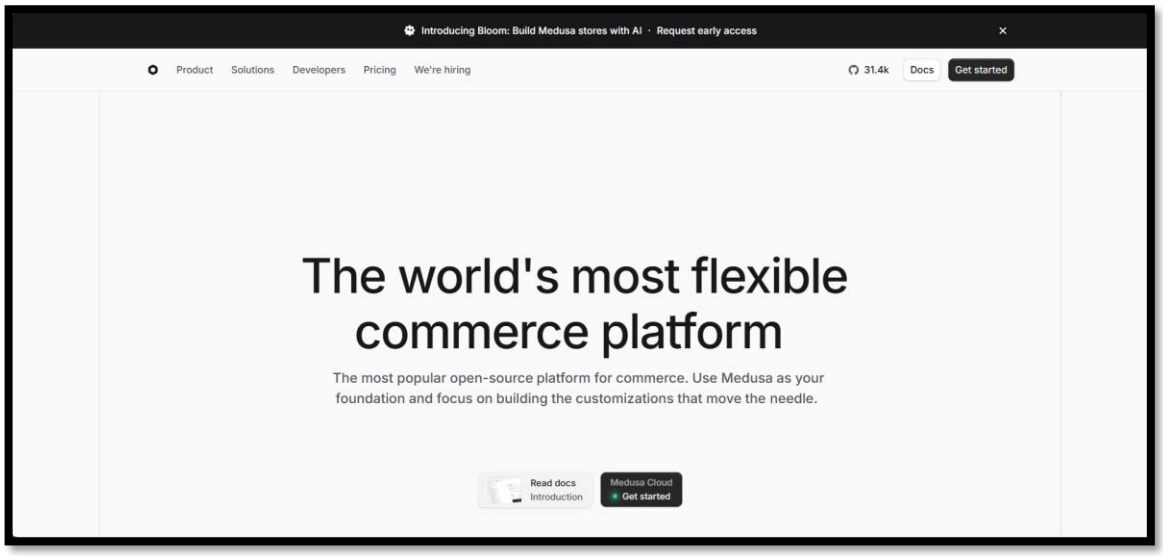
4.2. Xây dựng Backend với MedusaJS

Quá trình xây dựng backend được thực hiện dựa trên nền tảng MedusaJS, tuân thủ kiến trúc Headless và đảm bảo tính linh hoạt cho việc tích hợp.

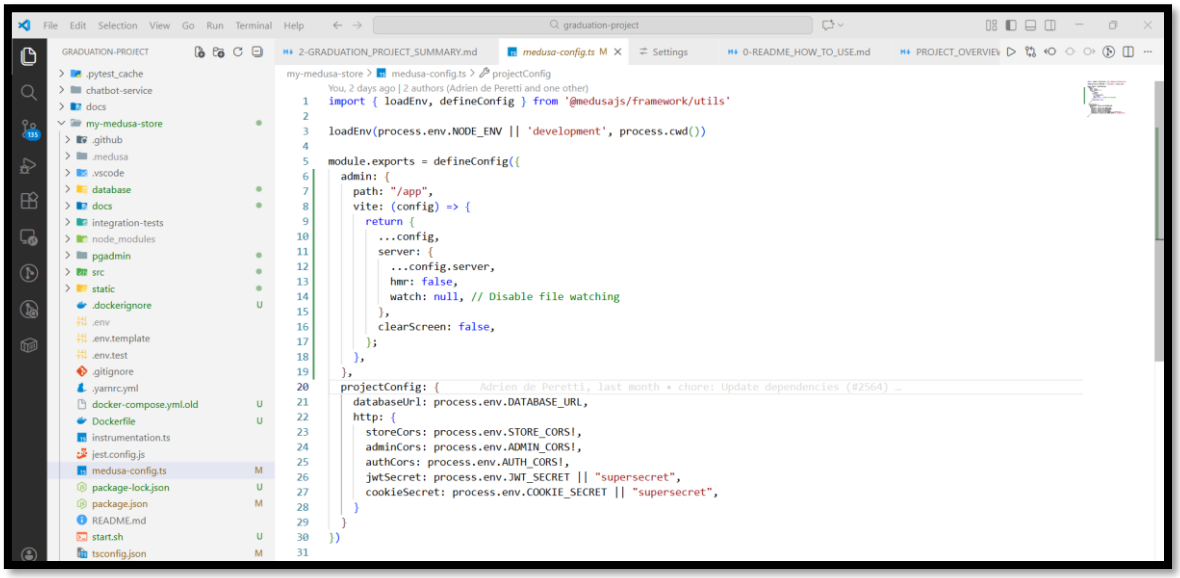
4.2.1. Khởi tạo và cấu hình Medusa Server

Thay vì phải xây dựng một hệ thống backend từ đầu, đề tài đã tận dụng nền tảng MedusaJS, một giải pháp Headless Commerce mã nguồn mở, mạnh mẽ và linh hoạt. Cách tiếp cận này giúp tiết kiệm đáng kể thời gian phát triển và cho phép tập trung nguồn lực vào việc xây dựng các dịch vụ AI giá trị gia tăng. Quá trình triển khai backend chủ yếu bao gồm việc khởi tạo, cấu hình và cài đặt các module cần thiết.

Dự án Medusa backend được khởi tạo nhanh chóng bằng câu lệnh npx create-medusa-app. Sau đó, các tệp cấu hình quan trọng như medusa-config.js và biến môi trường (.env) được tùy chỉnh để kết nối tới các dịch vụ lõi là PostgreSQL và Redis.



Hình 4.1: Giao diện trang chủ MedusaJS



Hình 4.2: Đoạn mã cấu hình database và Redis trong medusa-config.js

4.2.2. Triển khai các Module và dịch vụ

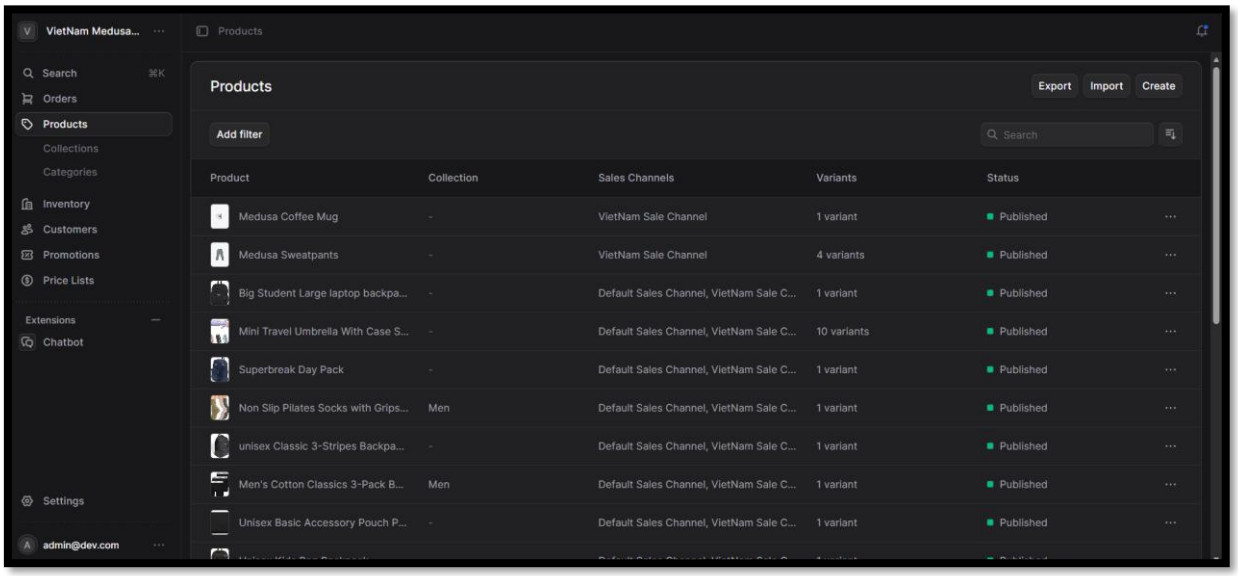
Medusa hỗ trợ cấu trúc plugin mạnh mẽ. Các plugin cần thiết cho một hệ thống TMĐT cơ bản đã được cài đặt và cấu hình:

- @medusajs/medusa-fulfillment: Xử lý các phương thức vận chuyển thủ công.
- @medusajs/medusa-payment-stripe: Tích hợp cổng thanh toán.
- @medusa-file-minio: Quản lý lưu trữ tệp tin hình ảnh sản phẩm.

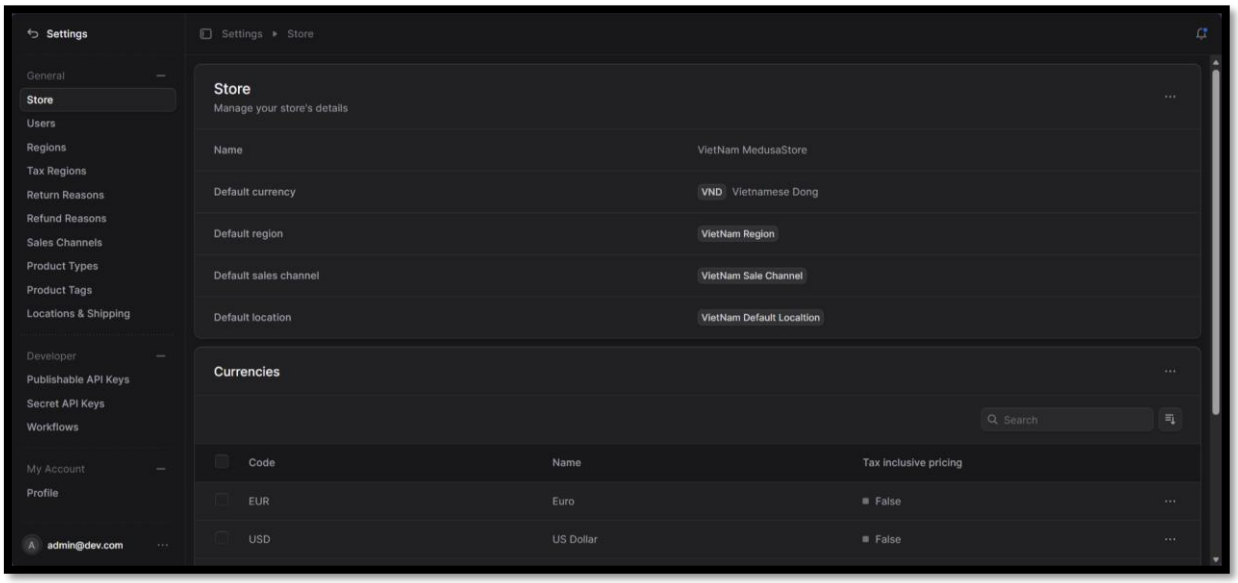
Tất cả các module này đều phơi bày các API endpoint tương ứng, cho phép Frontend và AI Service dễ dàng tương tác.

4.2.3. Quản lý dữ liệu nghiệp vụ

Dữ liệu sản phẩm (bao gồm tên, mô tả, giá, hình ảnh, biến thể), danh mục, đơn hàng và khách hàng được nhập liệu và quản lý thông qua Medusa Admin Dashboard. Dashboard này cung cấp giao diện trực quan cho quản trị viên để thực hiện các thao tác CRUD (Create, Read, Update, Delete) trên dữ liệu.



Hình 4.3: Giao diện quản lý sản phẩm Admin UI MedusaJS



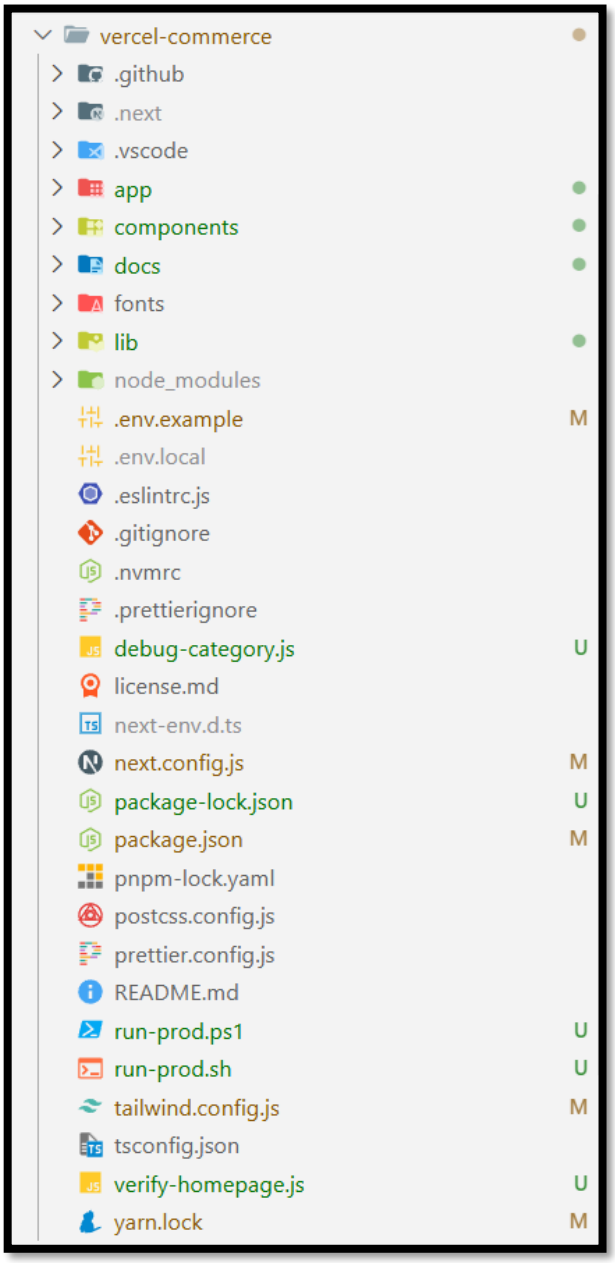
Hình 4.4: Giao diện thiết lập cài đặt Admin UI MedusaJS

4.3. Xây dựng Frontend với Next.js

Giao diện người dùng được xây dựng bằng Next.js, tập trung vào trải nghiệm người dùng tối ưu, hiệu năng cao và SEO hiệu quả, tương thích với kiến trúc Headless.

4.3.1. Khởi tạo và cấu trúc dự án Next.js

Dự án Frontend được khởi tạo bằng npx create-next-app. Cấu trúc thư mục được tổ chức theo các thành phần chức năng (ví dụ: pages/, components/, api/, lib/), đảm bảo tính module hóa và dễ bảo trì.



Hình 4.5: Cấu trúc thư mục Next.js frontend

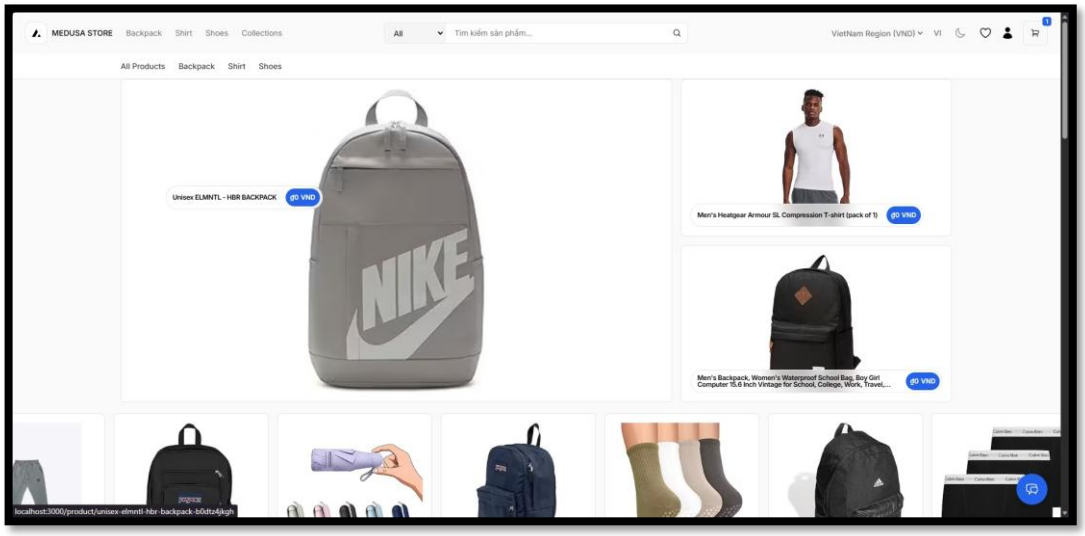
4.3.2. Tích hợp với Medusa Backend

Frontend sử dụng Medusa Client SDK (@medusajs/medusa-js) để gọi các API từ Medusa backend. Các hàm fetch dữ liệu được xây dựng để lấy danh sách sản phẩm, chi tiết sản phẩm, quản lý giỏ hàng và xử lý thanh toán.

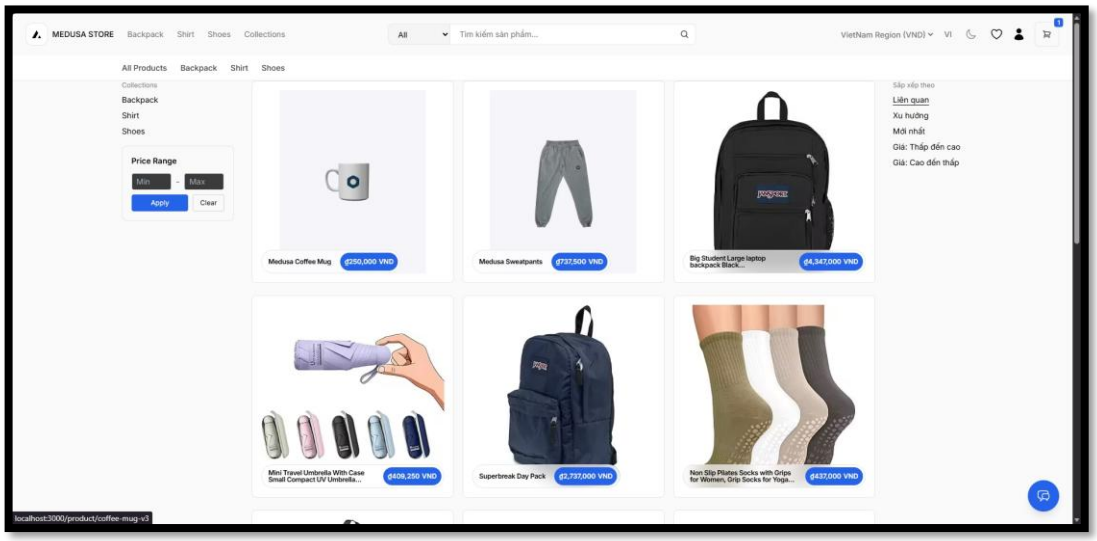
4.3.3. Thiết kế giao diện người dùng (UI/UX)

Giao diện được thiết kế hiện đại, responsive, sử dụng thư viện UI/CSS framework như Tailwind CSS hoặc Chakra UI để tăng tốc độ phát triển và đảm bảo tính nhất quán về mặt hình ảnh. Các trang chính được xây dựng bao gồm:

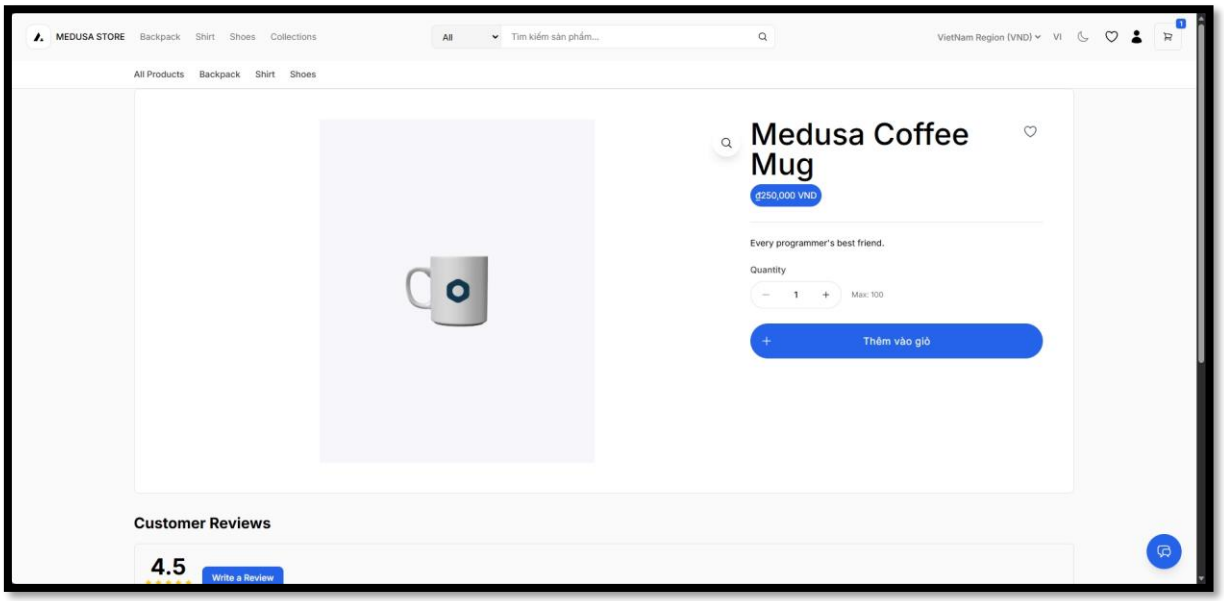
- Trang chủ: Hiển thị các sản phẩm nổi bật, danh mục.
- Trang danh sách sản phẩm: Cho phép duyệt và lọc sản phẩm.
- Trang chi tiết sản phẩm: Hiển thị thông tin đầy đủ, hình ảnh, tùy chọn mua.
- Giỏ hàng và thanh toán: Quy trình mua hàng.
- Giao diện chatbot: Một widget chat được nhúng trên mọi trang, cho phép người dùng tương tác với trợ lý ảo.



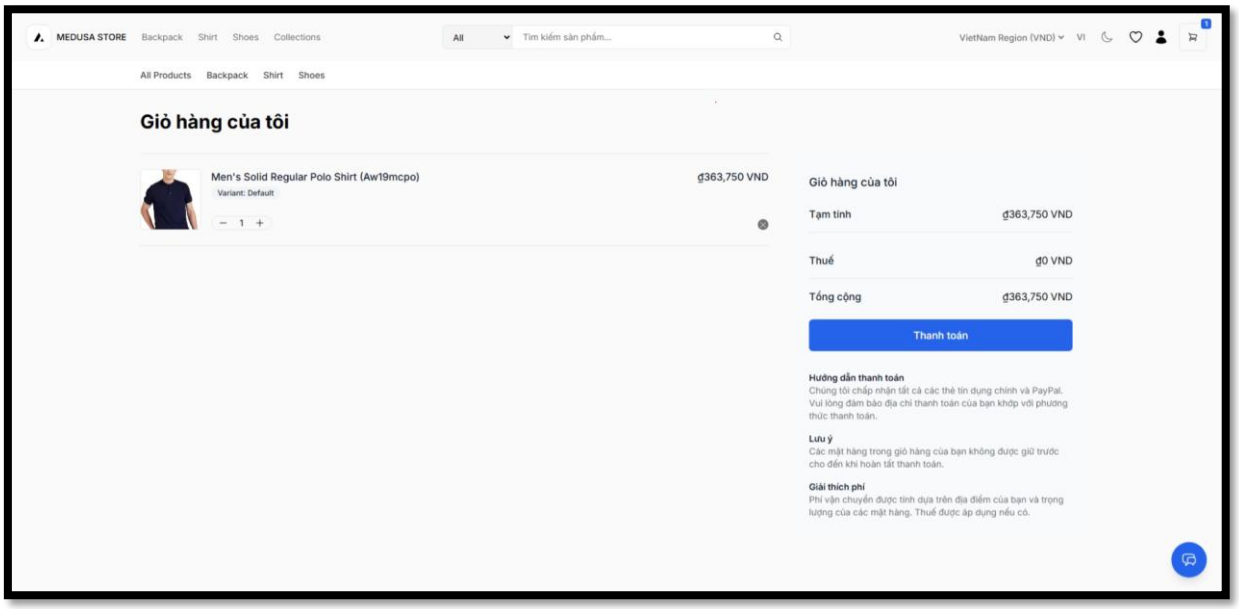
Hình 4.6: Giao diện trang chủ, sản phẩm nổi bật



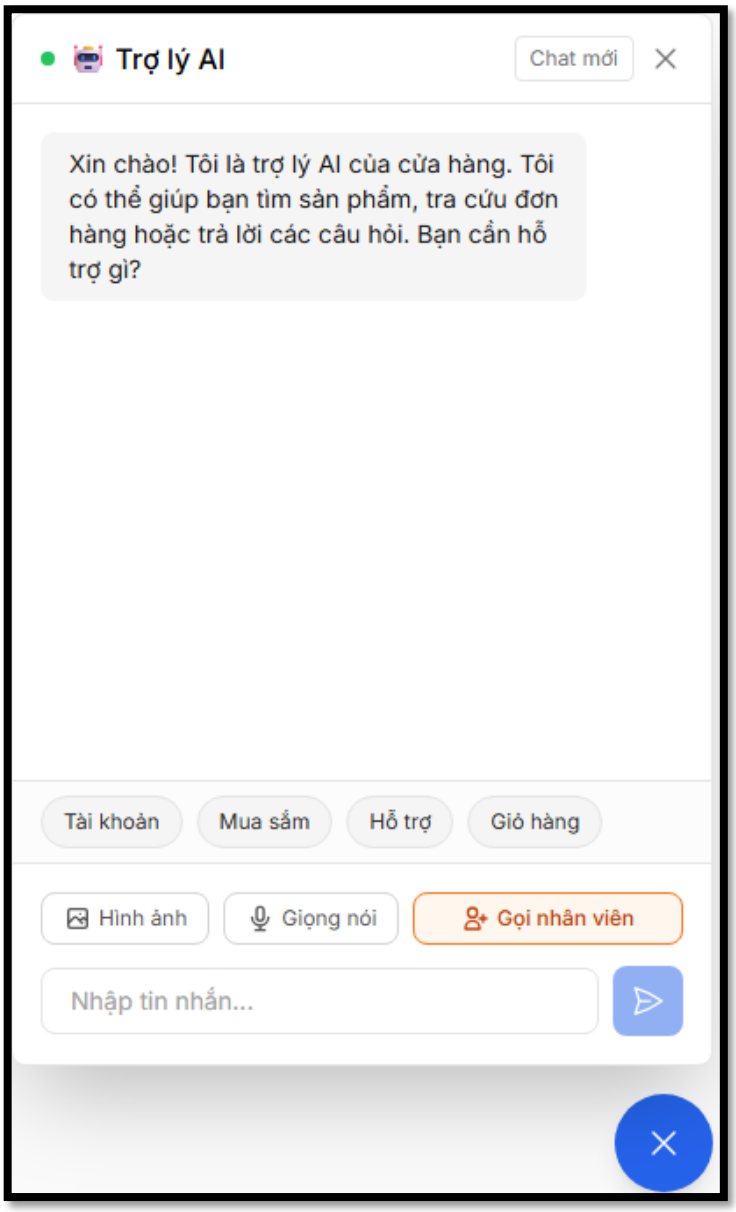
Hình 4.7: Giao diện trang danh sách sản phẩm



Hình 4.8: Giao diện trang chi tiết sản phẩm



Hình 4.9: Giao diện trang giỏ hàng và thanh toán



Hình 4.10: Giao diện widget chatbot trợ lý ảo

4.3.4. Cấu hình SSR/SSG

Các trang có nội dung tĩnh hoặc ít thay đổi (ví dụ: trang sản phẩm, trang blog) được cấu hình để sử dụng SSG bằng hàm getStaticProps. Các trang có nội dung động và yêu cầu dữ liệu thời gian thực (ví dụ: trang giỏ hàng, trang lịch sử đơn hàng) sử dụng SSR thông qua getServerSideProps hoặc Client-Side Fetching.

4.4. Xây dựng Chatbot Multi-Agent Service

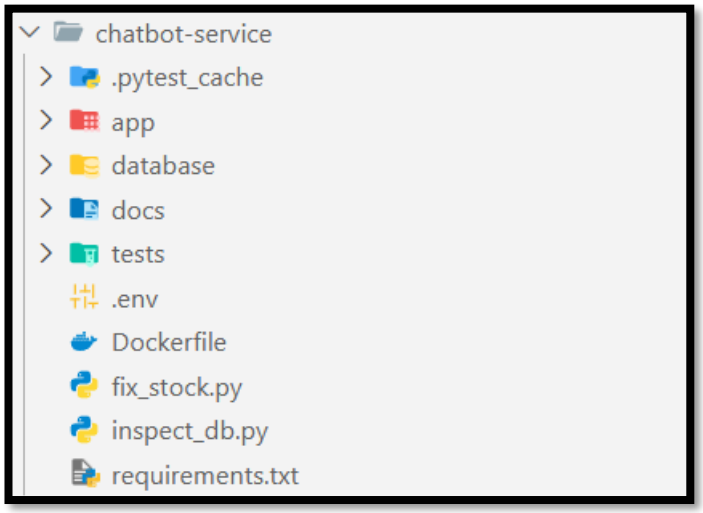
Đây là thành phần AI cốt lõi và phức tạp nhất của đề tài, được hiện thực hóa dưới dạng một dịch vụ microservice độc lập. Mục tiêu của dịch vụ này không phải là tạo ra một chatbot LLM-đơn lẻ, mà là xây dựng một hệ thống đa tác tử thông minh, có khả

năng cân bằng giữa tốc độ, chi phí, và độ tin cậy.

Dịch vụ được xây dựng bằng Python và framework FastAPI, với triết lý cốt lõi là kiến trúc Hybrid Multi-Agent. Điều này có nghĩa là:

- Hybrid: Khoảng 90% các yêu cầu thông thường của người dùng sẽ được xử lý bởi các luồng NLP dựa trên quy tắc (Rule-based NLP) để đảm bảo tốc độ cực nhanh và chi phí gần như bằng không. Chỉ khoảng 10% các yêu cầu phức tạp hoặc không xác định mới được chuyển đến LLM để xử lý (LLM Fallback).

- Multi-Agent: Thay vì một logic xử lý duy nhất, hệ thống được chia thành một chuỗi xử lý (pipeline) gồm 5 tác tử chuyên biệt, mỗi tác tử có một nhiệm vụ rõ ràng.



Hình 4.11: Cấu trúc thư mục Chatbot Multi-Agent Service

Luồng xử lý của một tin nhắn từ người dùng sẽ đi qua tuần tự 5 tác tử sau:

4.4.1. Agent 1: Input Processor (tác tử tiền xử lý)

Nhiệm vụ: Chuẩn hóa và làm giàu dữ liệu đầu vào.

Chi tiết triển khai: Tác tử này thực hiện các bước xử lý văn bản cơ bản bằng các thư viện Python chuẩn: loại bỏ ký tự thừa, chuẩn hóa dấu câu và khoảng trắng (sử dụng regex), và phát hiện ngôn ngữ. Quan trọng nhất, nó truy vấn vào bảng chatbot_context để tải ngữ cảnh của phiên trò chuyện hiện tại, cung cấp "bộ nhớ" cho các tác tử sau.

```python
def _normalize_text(text: str) -> str:
    # Trim, collapse spaces; keep diacritics (for VI) and punctuation minimally
    text = text.strip()
    text = re.sub(r"\s+", " ", text)
    return text


def _detect_language_heuristic(text: str) -> str:
    t = text.lower()
    # Very lightweight heuristic: VI keywords/diacritics vs common EN markers
    if any(ch in t for ch in "ăâêôơưđáàảãạấầẩẫậắằẳẵặéèẻẽẹếềểễệíìỉĩịóòỏõọốồổỗộớờởỡợúùủũụứừửữựýỳỷỹỵ"):
        return "vi"
    if any(word in t for word in [" the ", " and ", " price ", " order ", " product ", " hello "]):
        return "en"
    # Fallback: prefer VI as primary market
    return "vi"
```

```python
class InputProcessor:
    async def run(self, req: ChatRequest, pool: Optional[asyncpg.Pool]) -> ProcessedInput:
        print(f"[InputProcessor] Processing request: session={req.session_id}, msg='{req.message}'")
        cleaned = _normalize_text(req.message)
        lang = req.language or _detect_language_heuristic(cleaned)

        # Determine user_type:
        # 1. Trust metadata if provided (useful for testing or specific FE logic)
        # 2. Infer from customer_id (customer vs guest)
        user_type = "guest"
        if req.metadata and "user_type" in req.metadata:
            user_type = req.metadata["user_type"]
        elif req.customer_id:
            user_type = "customer"

        session_ctx = await _fetch_session_context(pool, req.session_id)
        print(f"[InputProcessor] Context loaded: last_intent={session_ctx.last_messages[0]['intent'] if session_ctx.

        return ProcessedInput(
            session_id=req.session_id,
            customer_id=req.customer_id,
            text=req.message,
            cleaned_text=cleaned,
            language=lang,
            user_type=user_type,
            tag=req.tag,
            session_ctx=session_ctx,
        )
```

Hình 4.12: Mã nguồn Agent Input Processor

4.4.2. Agent 2: Intent Classifier (tác tử phân loại ý định)

Nhiệm vụ: Xác định mục đích chính của người dùng. Đây là bước then chốt quyết định chatbot sẽ làm gì tiếp theo.

Chi tiết triển khai: Để tối ưu tốc độ và độ tin cậy, tác tử này được triển khai bằng phương pháp Keyword Matching và Decision Tree (cây quyết định) thay vì gọi LLM. Một bộ từ khóa và quy tắc được định nghĩa trước cho các ý định phổ biến (ví dụ: "tìm", "giá", "ở đâu" -> PRODUCT.SEARCH; "đơn hàng", "kiểm tra" -> ORDER.TRACK). Chỉ khi không khớp với bất kỳ quy tắc nào, ý định mới được gán là UNKNOWN để chuyển cho LLM.

```python
VI_KEYWORDS: Dict[str, List[str]] = {
    "GREETING": ["xin chào", "chào", "alo", "chào bạn", "hello", "hi", "hey", "chào buổi", "chào admin",
    "PRODUCT.SEARCH": [
        "tìm", "tìm kiếm", "có không", "còn không", "có bán", "muốn mua", "giá", "sản phẩm",
        "mua", "đặt", "kiếm", "tương tự", "giống", "cần", "shop có", "bên này có", "có hàng",
        "còn hàng", "hết hàng chưa", "còn size", "có màu", "loại", "dòng", "model"
    ],
    "PRODUCT.DETAIL": [
        "chi tiết", "thông số", "xem kỹ", "thông tin", "cụ thể", "mô tả", "spec",
        "đặc điểm", "tính năng", "chất liệu", "kích thước", "size", "màu sắc", "xuất xứ"
    ],
    "PRODUCT.RECOMMEND": [
        "gợi ý", "nên mua gì", "hot trend", "bán chạy", "phổ biến", "đề xuất", "recommend",
        "trending", "mới nhất", "best seller", "top", "tốt nhất", "hay nhất", "đáng mua"
    ],
    "ORDER.TRACK": [
        "tra cứu", "kiểm tra", "đơn hàng", "ở đâu", "tình trạng", "tracking", "giao hàng",
        "ship", "vận chuyển", "đã giao chưa", "khi nào nhận", "bao giờ về", "đến chưa"
    ],
    "ORDER.CANCEL": ["hủy đơn", "không mua nữa", "huỷ đơn", "cancel", "không muốn", "đổi ý", "không lấy"
    "ORDER.RETURN": [
        "đổi trả", "hoàn tiền", "bảo hành", "trả hàng", "return", "refund", "warranty",
        "đổi hàng", "lỗi", "hư", "không vừa", "sai", "không đúng"
    ],
```

```python
class IntentClassifier:
    async def run(self, processed: ProcessedInput) -> IntentResult:
        print(f"[IntentClassifier] Classifying: '{processed.cleaned_text}'")
        text = processed.cleaned_text
        lang = processed.language or "vi"

        # Choose keyword table
        table = VI_KEYWORDS if lang == "vi" else EN_KEYWORDS

        # Compute scores
        scores: Dict[str, int] = {}
        for key, phrases in table.items():
            scores[key] = _match_score(text, phrases)

        # Get top 3 intents for multi-intent detection
        sorted_scores = sorted(scores.items(), key=lambda kv: kv[1], reverse=True)
        top_key = sorted_scores[0][0] if sorted_scores else "UNKNOWN"
        top_score = sorted_scores[0][1] if sorted_scores else 0

        # Detect secondary intent (e.g., "tìm balo giá rẻ" = PRODUCT.SEARCH + PRODUCT.PRICE)
        secondary_intent = None
        if len(sorted_scores) > 1 and sorted_scores[1][1] >= 2:
            secondary_intent = sorted_scores[1][0]

        print(f"[IntentClassifier] Top: {top_key}({top_score}), Secondary: {secondary_intent}")
```

Hình 4.13: Mã nguồn Agent Intent Classifier

4.4.3. Agent 3: Orchestrator (tác tử điều phối)

Nhiệm vụ: Hoạt động như "bộ não" của hệ thống, nhận kết quả từ Intent Classifier và lập kế hoạch hành động.

Chi tiết triển khai: Tác tử này chứa logic nghiệp vụ phức tạp:

- Kiểm tra quyền hạn: Dựa vào user_type (guest/customer) từ ngữ cảnh, nó quyết

định người dùng có được phép thực hiện ý định đó không (ví dụ: guest không thể xem lịch sử đơn hàng).

- Kiểm tra dữ liệu: Đảm bảo các thực thể cần thiết đã được trích xuất (ví dụ: ORDER.TRACK cần có order_id). Nếu thiếu, nó sẽ tạo ra một kế hoạch để hỏi lại người dùng.

- Lập kế hoạch: Quyết định sẽ gọi "Công cụ" nào tiếp theo và với tham số gì.

```python
class Orchestrator:
    def __init__(self):
        self.agents = {
            "sales": SalesAgent(),
            "order": OrderAgent(),
            "staff": StaffAgent(),
            "manager": ManagerAgent()
        }

    async def run(
        self,
        processed: ProcessedInput,
        intent: IntentResult,
    ) -> Tuple[ActionPlan, Optional[ToolResults]]:
        """
        Hybrid Orchestrator:
        1. Executable Tag -> Execute Tool directly
        2. Scope Tag -> Route to Specialized Agent (Scoped NLU)
        3. No Tag -> Global NLU -> Route to Agent
        """
        print(f"[Orchestrator] Routing... Tag={processed.tag}, Intent={intent.intent}")
```

```python
        # 2. Scope Tag (e.g., "scope:order")
        if processed.tag and processed.tag.startswith("scope:"):
            scope = processed.tag.split(":", 1)[1]
            agent = self.agents.get(scope)
            if agent:
                print(f"[Orchestrator] Scoped routing to: {scope}")
                # Agent performs its own scoped NLU if intent is generic/unknown or we want to re-evaluate
                # For now, we pass None as intent to force agent to re-classify within scope
                return await agent.process(processed, intent=None)

        # 3. Implicit Routing (Global NLU)
        # We use the intent passed from the global classifier to decide which agent to call
        target_agent = self._route_by_intent(intent.intent)
        if target_agent:
            print(f"[Orchestrator] Implicit routing to: {target_agent.name}")
            return await target_agent.process(processed, intent)

        # Default fallback
        print("[Orchestrator] No agent found, fallback.")
        return ActionPlan(tools=[], next_step=None), None

    def _route_by_intent(self, intent_name: str):
        """Route intent to appropriate specialized agent"""
        for agent in self.agents.values():
            if agent.can_handle(intent_name):
                return agent
        return self.agents["sales"] # Default to sales if unknown
```

Hình 4.14: Mã nguồn Agent Orchestrator

4.4.4. Agent 4: Executor (tác tử thực thi)

Nhiệm vụ: "Bàn tay" của hệ thống, chịu trách nhiệm thực thi các kế hoạch mà Orchestrator đã vạch ra.

Chi tiết triển khai: Tác tử này quản lý một danh sách các "Công cụ" (Tools). Mỗi công cụ là một hàm chức năng được định nghĩa để tương tác với một hệ thống bên ngoài. Ví dụ:

- search_products_tool: Gọi đến Vector DB để thực hiện tìm kiếm ngữ nghĩa.
- get_order_tool: Gọi đến API GET /store/orders/{id} của Medusa Backend.

Tác tử này cũng bao gồm logic xử lý lỗi và thử lại (retry) khi các lệnh gọi API thất bại.

```
18  def _merge_tool_result(results: ToolResults, tool_result: Any):
19      if tool_result is None:
20          return
21      if isinstance(tool_result, ToolResults):
22          results.ok = results.ok and tool_result.ok
23          if tool_result.errors:
24              results.errors.extend(tool_result.errors)
25          if tool_result.timings_ms:
26              results.timings_ms.update(tool_result.timings_ms)
27          results.data = tool_result.data
28      else:
29          results.data = tool_result
30
31  class Executor:
32      async def run(
33          self,
34          processed: ProcessedInput,
35          intent: IntentResult,
36          plan: ActionPlan
37      ) -> Optional[ToolResults]:
38          if not plan.tools:
39              return None
```

Hình 4.15: Mã nguồn Agent Executor

4.4.5. Agent 5: Response Generator (tác tử tạo phản hồi)

Nhiệm vụ: Tổng hợp kết quả từ Executor và tạo ra một câu trả lời hoàn chỉnh để gửi lại cho người dùng.

Chi tiết triển khai: Tác tử này hoạt động ở hai chế độ:

- Chế độ mẫu (90%): Với các yêu cầu thông thường, nó truy vấn

bảng chatbot_responses để lấy một mẫu câu trả lời, sau đó điền dữ liệu từ kết quả của Executor vào.

- Chế độ LLM (10%): Với các ý định UNKNOWN hoặc các trường hợp cần sự sáng tạo, nó sẽ gửi toàn bộ ngữ cảnh và kết quả thu thập được đến LLM Service để tạo ra một phản hồi tự nhiên và phù hợp hơn.

```python
class ResponseGenerator:
    def __init__(self):
        self.client = AsyncOpenAI(
            api_key=GOOGLE_API_KEY,
            base_url=GEMINI_BASE_URL
        )
        self.context_config = self._load_context_config()

    def _load_context_config(self):
        try:
            path = os.path.join(os.path.dirname(__file__), "..", "context_config.yaml")
            with open(path, 'r', encoding='utf-8') as f:
                return yaml.safe_load(f)
        except Exception as e:
            logger.error(f"Error loading context_config: {e}")
            return {}
```

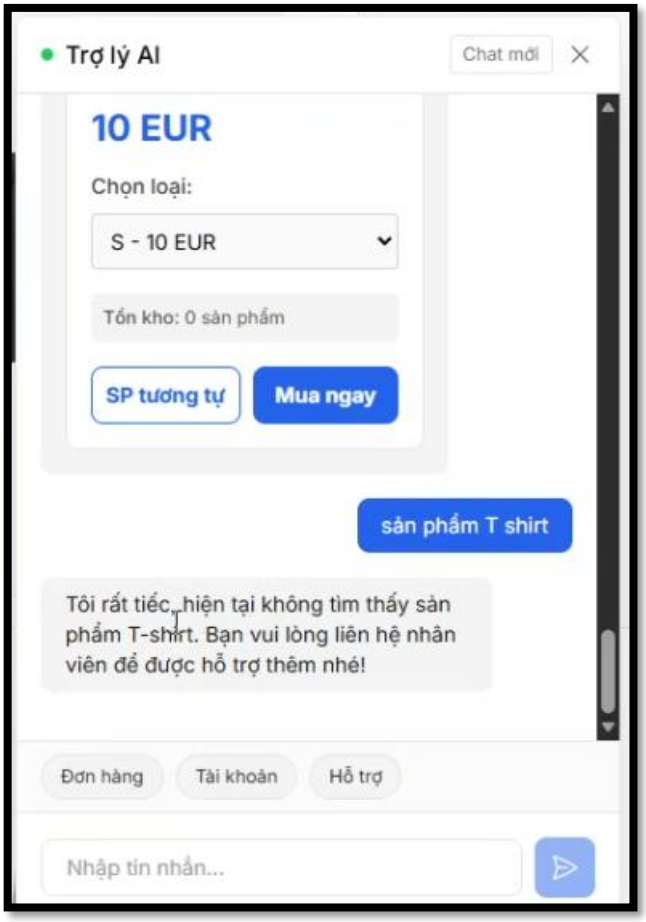
```python
    def _generate_quick_replies(self, processed) -> List[QuickReply]:
        nodes = self.context_config.get("nodes", [])

        # Helper to find node by ID recursively
        def find_node(node_list, target_id):
            for node in node_list:
                if node.get("id") == target_id:
                    return node
                if "children" in node:
                    found = find_node(node["children"], target_id)
                    if found: return found
            return None

        # Determine current context node
        current_node = None
        if processed.tag:
            # If tag is like "action:view_cart", we might not find a node if it's a leaf.
            # But if tag is a group ID, we might.
            # Let's try to find by ID first (assuming tag might be ID)
            current_node = find_node(nodes, processed.tag)

        # If no specific node found, default to user root
        if not current_node:
            user_root_id = f"{processed.user_type}_root"
            current_node = find_node(nodes, user_root_id)
```

Hình 4.16: Mã nguồn Agent Response Generator

Đây là cơ chế cốt lõi giúp chatbot có thể "hành động". Mỗi công cụ được định nghĩa như một lớp (class) trong Python, có mô tả rõ ràng về chức năng và các tham số đầu vào. Framework LangChain được sử dụng để giúp các Agent có thể tự động lựa chọn và gọi đúng công cụ dựa trên mô tả của chúng.

- Triển khai Sales Agent (RAG): Khi ý định là tư vấn sản phẩm, Orchestrator sẽ chỉ định Executor sử dụng search_products_tool. Công cụ này sẽ chuyển câu hỏi của người dùng thành vector, truy vấn Qdrant để lấy các đoạn văn bản sản phẩm liên quan, sau đó Response Generator sẽ dùng LLM để tổng hợp các đoạn văn bản này thành một câu tư vấn hoàn chỉnh.

- Triển khai Order Agent (API Tool Calling): Khi ý định là tra cứu đơn hàng, Orchestrator sẽ chỉ định Executor sử dụng get_order_tool. Công cụ này được lập trình để thực hiện một lệnh gọi GET đến API của Medusa Backend với order_id làm tham số và trả về dữ liệu JSON.



Hình 4.17: Chatbot trả lời câu hỏi về sản phẩm của người dùng

Để xử lý các tình huống phức tạp mà bot không thể giải quyết, một luồng leo thang đã được triển khai.

- Khi Intent Classifier phát hiện ý định SUPPORT.ESCALATE (ví dụ: người dùng nói "gặp nhân viên").

- Orchestrator sẽ cập nhật trạng thái của phiên trò chuyện trong bảng chatbot_context thành escalated.

- Một tin nhắn real-time (sử dụng WebSocket) được gửi đến giao diện Admin, thông báo rằng có một khách hàng đang cần hỗ trợ, kèm theo session_id để quản trị viên có thể xem lại lịch sử trò chuyện.

4.5. Xây dựng Recommendation Service

Dịch vụ gợi ý được xây dựng như một microservice độc lập bằng Python và FastAPI, chịu trách nhiệm hoàn toàn cho việc theo dõi hành vi người dùng, tính toán và cung cấp các gợi ý sản phẩm cá nhân hóa.

Đây là nền tảng thu thập dữ liệu của toàn bộ hệ thống. Một API endpoint POST /track đã được xây dựng để nhận các sự kiện từ Frontend.

- Các loại tương tác: Hệ thống được thiết kế để ghi nhận 5 loại tương tác chính, mỗi loại được gán một trọng số (weight) khác nhau để phản ánh mức độ quan tâm của người dùng:

    + view (xem sản phẩm): Trọng số thấp (ví dụ: 1.0)

    + add_to_cart (thêm vào giỏ): Trọng số cao hơn (ví dụ: 2.0)

    + purchase (mua hàng): Trọng số cao nhất (ví dụ: 5.0)

- Xử lý sự kiện: Khi nhận được một sự kiện, dịch vụ sẽ ghi một bản ghi mới vào bảng rec_user_interactions. Đồng thời, một tác vụ nền (background task) có thể được kích hoạt để cập nhật lại bảng rec_user_preferences, tổng hợp sở thích của người dùng theo từng danh mục sản phẩm.

```
13  export function ProductViewTracker({ productId, productHandle, category, price }: ProductTrackingProps) {
14    useEffect(() => {
15      // Track product view
16      trackInteraction({
17        productId,
18        productHandle,
19        interactionType: 'view',
20        metadata: {
21          category,
22          price,
23          viewedAt: new Date().toISOString(),
24        },
25      });
26    }, [productId, productHandle, category, price]);
27
28    return null; // This component doesn't render anything
29  }
```

```
31  export function trackAddToCart(productId: string, productHandle: string, quantity: number) {
32    trackInteraction({
33      productId,
34      productHandle,
35      interactionType: 'add_to_cart',
36      metadata: {
37        quantity,
38      },
39    });
40  }
41
42  export function trackWishlist(productId: string, productHandle: string, action: 'add' | 'remove') {
43    trackInteraction({
44      productId,
45      productHandle,
46      interactionType: 'wishlist',
47      metadata: {
48        action,
49      },
50    });
51  }
```

```
53  export function trackSearch(query: string, resultCount: number) {
54    trackInteraction({
55      interactionType: 'search',
56      metadata: {
57        query,
58        resultCount,
59      },
60    });
61  }
```

Hình 4.18: Mã nguồn của API endpoint POST /track, nhận dữ liệu và lưu

Đây là lõi xử lý của dịch vụ. Khi nhận được yêu cầu GET /recommendations, hệ thống sẽ thực thi thuật toán Hybrid theo các bước sau:

Bước 1: Lấy dữ liệu đầu vào: Truy vấn bảng rec_user_preferences để lấy hồ sơ sở thích của người dùng và rec_user_interactions để lấy lịch sử tương tác gần đây.

Bước 2: Tính điểm Content-Based: Hệ thống lấy ra các danh mục mà người dùng yêu thích nhất và tìm các sản phẩm khác trong cùng danh mục đó. Điểm số được tính dựa trên mức độ yêu thích (score trong bảng rec_user_preferences) và độ mới của sản phẩm.

Bước 3: Tính điểm Collaborative Filtering:

- Hệ thống xác định một nhóm nhỏ những người dùng khác có lịch sử tương tác tương tự nhất với người dùng hiện tại (user-user similarity).

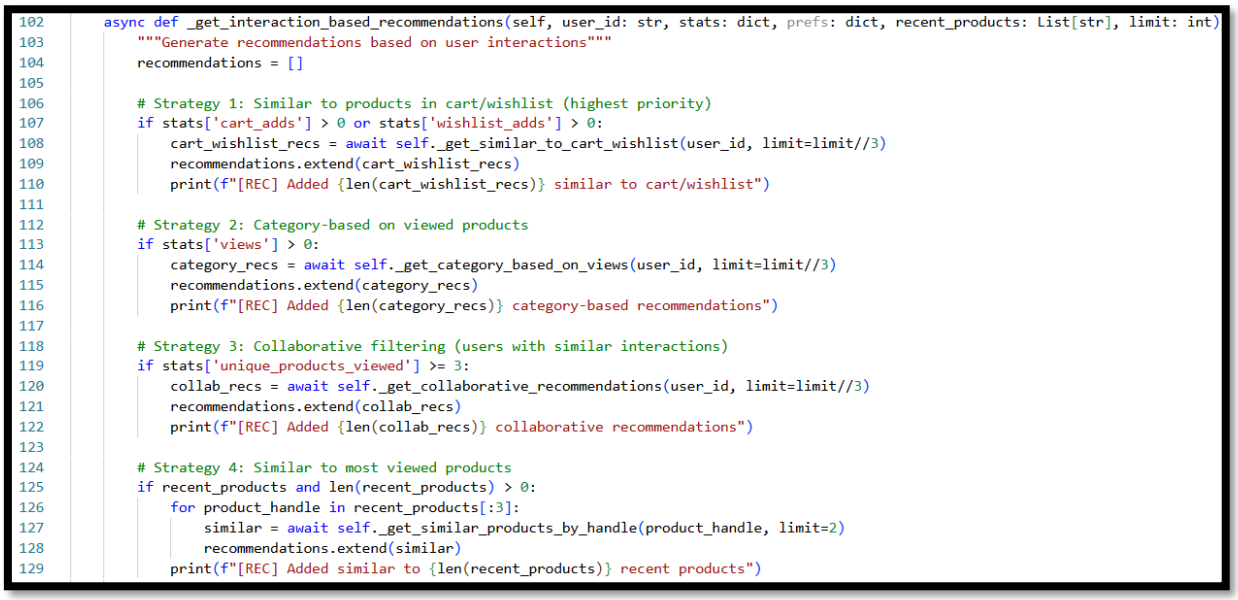
- Sau đó, nó tìm ra những sản phẩm mà nhóm người dùng tương tự này đã mua nhưng người dùng hiện tại chưa tương tác.

Bước 4: Kết hợp điểm số: Điểm số cuối cùng cho mỗi sản phẩm ứng viên được tính bằng công thức trọng số:

SCORE = (0.4 * Content_Score) + (0.6 * Collaborative_Score).

Bước 5: Lấy thông tin chi tiết: Sau khi có danh sách các product_id được xếp hạng cao nhất, dịch vụ sẽ gọi API của Medusa Backend để lấy thông tin đầy đủ (tên, ảnh, giá) trước khi trả về cho Frontend.

```python
    async def get_personalized_recommendations(self, user_id: str, limit: int = 12) -> Tuple[List[dict], str]:
        """Get personalized recommendations for user (hybrid approach)"""

        # Get user interactions analysis
        user_stats = await self._get_user_interaction_stats(user_id)

        # Get user preferences (computed from interactions)
        prefs = await self._get_user_preferences(user_id)

        # Get user's recent interactions
        recent_products = await self._get_user_recent_products(user_id, limit=10)

        # If user has ANY interactions, use intelligent recommendations
        if user_stats['total_interactions'] > 0:
            print(f"[REC] User {user_id} has {user_stats['total_interactions']} interactions - using personalized approach")
            return await self._get_interaction_based_recommendations(user_id, user_stats, prefs, recent_products, limit)

        # Only truly NEW users get trending/random
        trending = await self._get_trending_products(limit)
        if trending:
            return trending, "trending"
        # Last resort: random products
        return await self._get_random_products(limit), "random"

        # Hybrid recommendation
        recommendations = []
```



```python
    async def _get_interaction_based_recommendations(self, user_id: str, stats: dict, prefs: dict, recent_products: List[str], limit: int)
        """Generate recommendations based on user interactions"""
        recommendations = []

        # Strategy 1: Similar to products in cart/wishlist (highest priority)
        if stats['cart_adds'] > 0 or stats['wishlist_adds'] > 0:
            cart_wishlist_recs = await self._get_similar_to_cart_wishlist(user_id, limit=limit//3)
            recommendations.extend(cart_wishlist_recs)
            print(f"[REC] Added {len(cart_wishlist_recs)} similar to cart/wishlist")

        # Strategy 2: Category-based on viewed products
        if stats['views'] > 0:
            category_recs = await self._get_category_based_on_views(user_id, limit=limit//3)
            recommendations.extend(category_recs)
            print(f"[REC] Added {len(category_recs)} category-based recommendations")

        # Strategy 3: Collaborative filtering (users with similar interactions)
        if stats['unique_products_viewed'] >= 3:
            collab_recs = await self._get_collaborative_recommendations(user_id, limit=limit//3)
            recommendations.extend(collab_recs)
            print(f"[REC] Added {len(collab_recs)} collaborative recommendations")

        # Strategy 4: Similar to most viewed products
        if recent_products and len(recent_products) > 0:
            for product_handle in recent_products[:3]:
                similar = await self._get_similar_products_by_handle(product_handle, limit=2)
                recommendations.extend(similar)
            print(f"[REC] Added similar to {len(recent_products)} recent products")
```

Hình 4.19: Mã nguồn của Recommendation Engine

Để đáp ứng các ngữ cảnh khác nhau trên trang web, hệ thống không chỉ dùng một thuật toán duy nhất mà triển khai 5 chiến lược khác nhau, có thể được gọi thông qua tham số trên API:

1. Hybrid (Mặc định): Sử dụng cho các gợi ý cá nhân hóa chính trên trang chủ.

2. Content-Based: Sử dụng để giải quyết vấn đề cold start. Khi một người dùng mới vừa xem sản phẩm, chiến lược này sẽ được ưu tiên để gợi ý các sản phẩm tương tự.

3. Collaborative Filtering: Có thể sử dụng cho các người dùng đã có lịch sử tương tác rất dày đặc để tăng tính khám phá.

4. Trending (Xu hướng): Gợi ý các sản phẩm được xem hoặc mua nhiều nhất trong 7 ngày qua. Được sử dụng cho các mục "Sản phẩm thịnh hành".

5. Frequently Bought Together (Thường được mua cùng): Được tính toán trước và lưu trữ, sử dụng trên trang chi tiết sản phẩm để gợi ý các sản phẩm mua kèm.

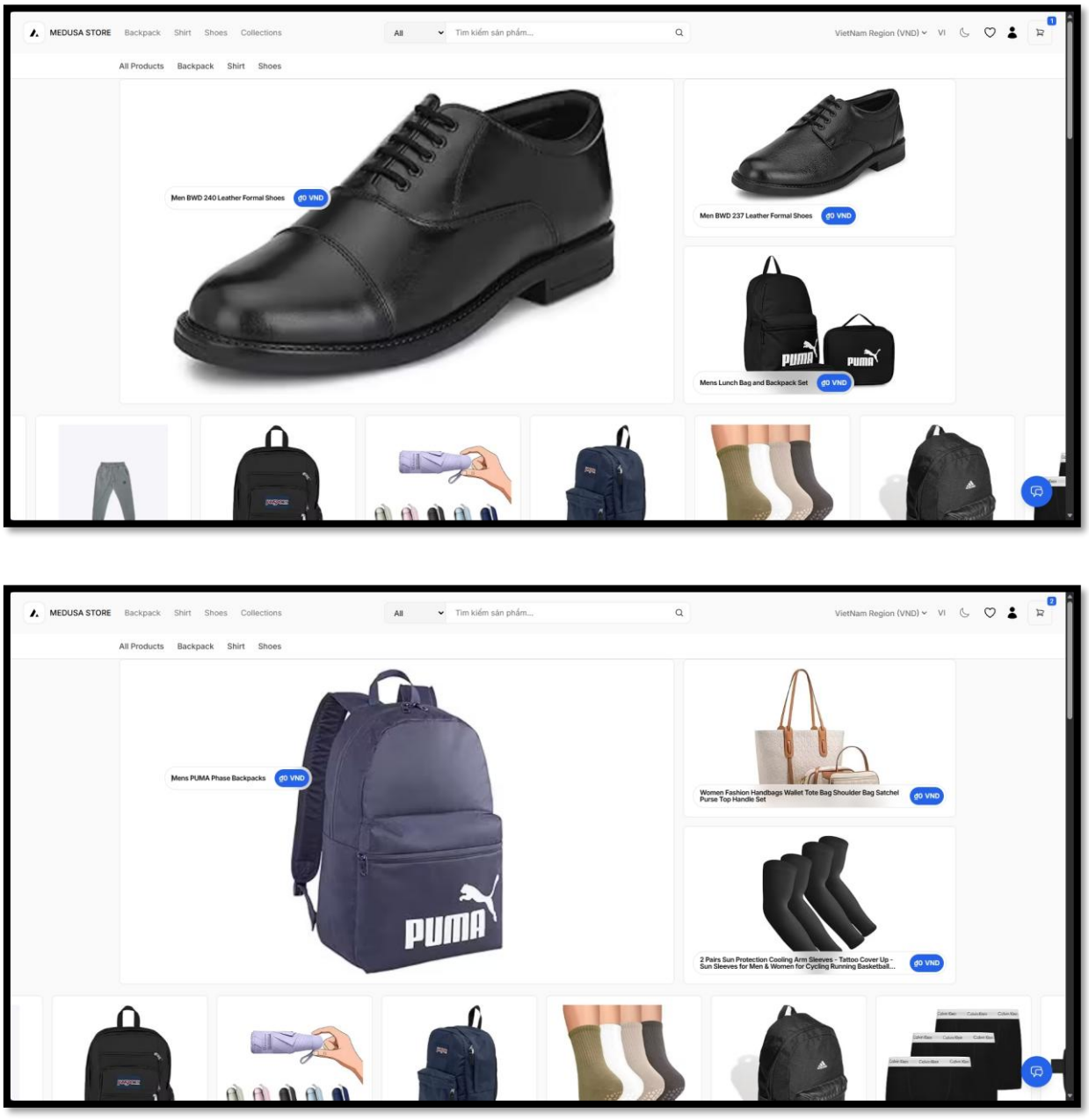
Hiệu năng là yếu tố sống còn của một hệ thống gợi ý. Một cơ chế caching nhiều lớp đã được triển khai:

- Redis Cache: Trước khi thực hiện bất kỳ tính toán nào, hệ thống sẽ kiểm tra Redis trước tiên với một khóa duy nhất (ví dụ: rec:user_id:hybrid). Nếu tìm thấy kết quả, nó sẽ được trả về ngay lập tức (<50ms). Kết quả sau khi tính toán cũng sẽ được lưu vào Redis với thời gian sống là 1 giờ.

- Database Cache: Các kết quả tính toán tốn nhiều tài nguyên như độ tương đồng

giữa các sản phẩm (rec_product_similarities) được tính toán trước bởi các tác vụ nền và lưu vào bảng trong PostgreSQL, biến các phép tính phức tạp thành các câu lệnh truy vấn đơn giản.

Chiến lược này giúp hệ thống đạt tỷ lệ cache hit mục tiêu trên 70%, đảm bảo phần lớn người dùng nhận được gợi ý một cách nhanh chóng, mang lại trải nghiệm mượt mà.



Hình 4.20: Hình ảnh trang chủ thay đổi khi người dùng tương tác

4.6. Đánh giá kết quả thực nghiệm

Nhằm kiểm chứng tính đúng đắn và hiệu quả của các giải pháp công nghệ đã đề xuất, tôi tiến hành thực nghiệm hệ thống trong môi trường giả lập sát với thực tế. Quá trình đánh giá tập trung vào khả năng xử lý của hệ thống Multi-Agent và độ chính xác của các gợi ý cá nhân hóa.

4.6.1. Thiết lập kịch bản và bộ dữ liệu đánh giá

4.6.1.1. Đối với hệ thống Multi-Agent

Để đánh giá toàn diện năng lực của hệ thống Multi-Agent, tôi không chỉ tập trung vào các câu lệnh đúng cú pháp mà còn mở rộng sang các tình huống thực tế phức tạp. Như được trình bày tại Bảng 4.2, các kịch bản được thiết kế với các cấp độ khó khác nhau:

- Nhóm chức năng cơ bản (STT 1, 2, 3): Chiếm tỷ trọng lớn nhất (74%), tập trung vào luồng mua sắm chính của người dùng, yêu cầu tốc độ phản hồi cao và dữ liệu từ API chính xác.

- Nhóm kiểm soát nghiệp vụ (STT 4): Nhằm minh chứng khả năng của tác tử Orchestrator trong việc nhận diện vai trò người dùng để mở khóa các công cụ nhạy cảm (như báo cáo doanh thu).

- Nhóm xử lý ngoại lệ (STT 5): Thử thách khả năng làm sạch dữ liệu của tác tử Input Processor và năng lực lập luận của LLM khi đối mặt với các đầu vào không có cấu trúc hoặc chứa nhiều nhiễu thông tin.

Bảng 4.2: Danh sách các nhóm tình huống kiểm thử hệ thống Multi-Agent

STT	Nhóm tình huống	Kịch bản tiêu biểu	Số lượng	Mục tiêu kiểm thử
1	Chào hỏi & FAQ	Xin chào; Cảm ơn; Hỏi về phí ship; Chính sách đổi trả; Bảo hành.	10	Kiểm tra khả năng phản hồi ngôn ngữ tự nhiên và truy xuất tri thức từ kho dữ liệu FAQ.
2	Tìm kiếm &	Tìm backpack; Tìm laptop backpack; Mặt	12	Đánh giá khả năng truy vấn ngữ nghĩa (RAG) tại

	Gợi ý	hàng bán chay; Tìm balo dưới 500k; thông tin casual socks; Gơi ý môt vài shoes; Tìm túi đeo chéo.		Vector Database và lọc sản phẩm theo thuộc tính.
3	Giỏ hàng & Đơn hàng	Thêm vào giỏ; Xem giỏ hàng; Kiểm tra tình trạng đơn hàng #12345; Hủy đơn hàng.	15	Kiểm tra tính chính xác khi gọi các API công cụ của Medusa Backend và xử lý dữ liệu thời gian thực.
4	Tác vụ quản trị	Check kho; Tìm thông tin khách hàng; Báo cáo doanh thu; Thống kê hiệu quả Chatbot.	04	Kiểm tra cơ chế phân quyền (RBAC) của Orchestrator dành riêng cho Staff và Manager.
5	Trường hợp biên & Phức tạp	Nhập văn bản nhiễu; Tin nhắn trống; Câu hỏi dài (Stress test); Trộn lẫn tiếng Việt - Anh.	09	Đánh giá độ bền vững (Robustness) của hệ thống và khả năng xử lý của cơ chế LLM Fallback.
Tổng			**50**	

4.6.1.2. Đối với hệ thống gợi ý

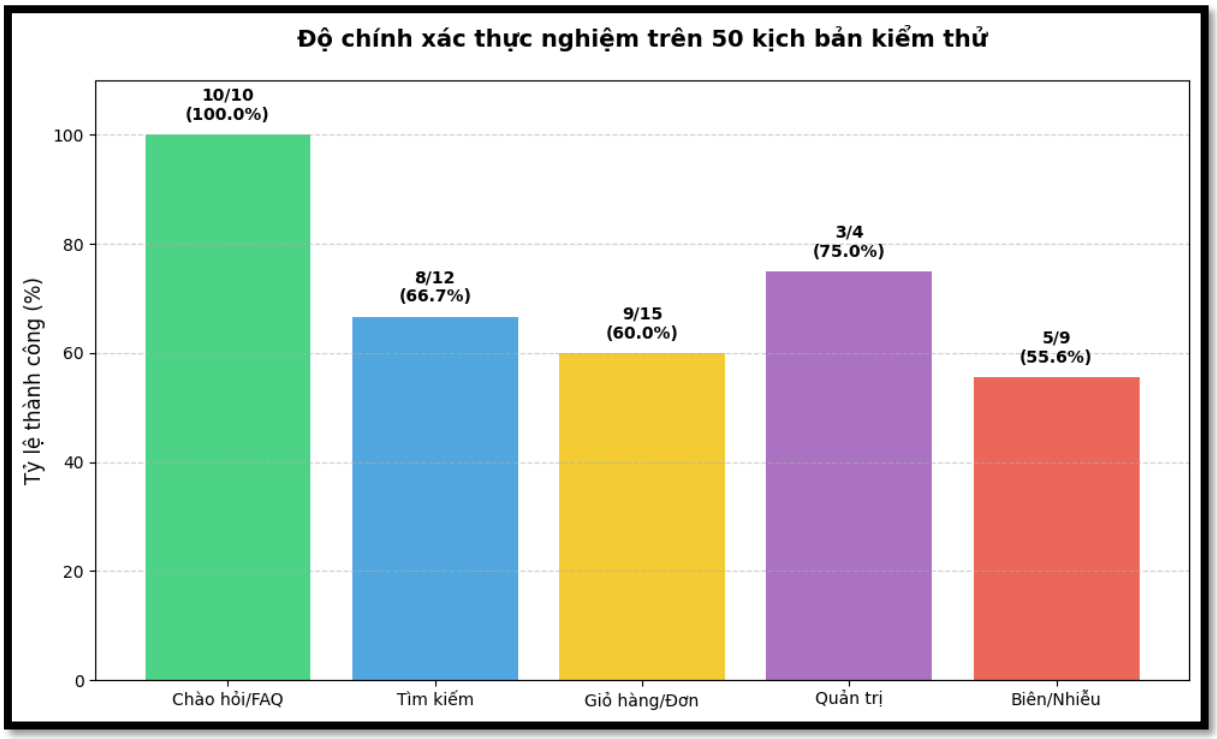
Sử dụng danh mục 50 sản phẩm thực tế từ kho hàng. Do hệ thống mới triển khai, tôi thiết lập môi trường cold start bằng cách tạo ra các tương tác giả lập cho 10 người dùng mới để đánh giá sự thay đổi của thuật toán Hybrid.

4.6.2. Đánh giá hiệu năng và độ chính xác của hệ thống Multi-Agent =

Việc đánh giá Chatbot được thực hiện thông qua việc đối soát kết quả thực tế với kỳ vọng (Ground Truth) về phân loại ý định và khả năng thực thi tác vụ.

4.6.2.1. Phân tích độ chính xác theo nhóm chức năng

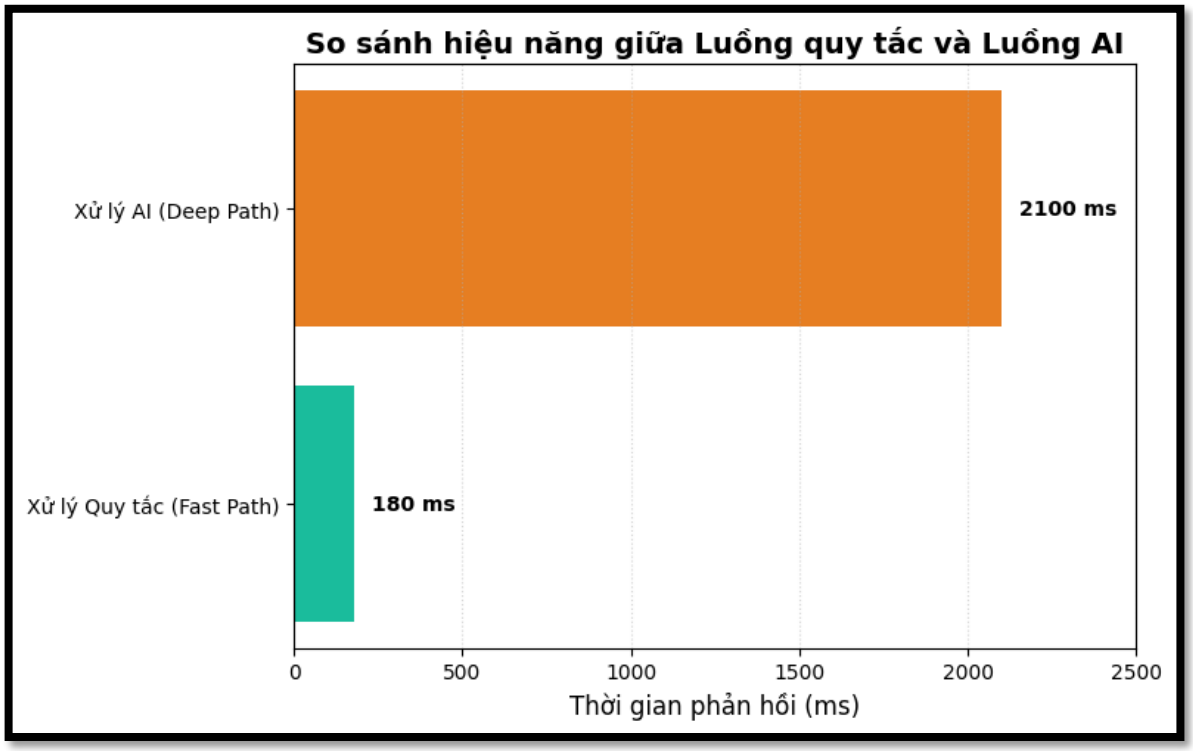
Hệ thống được vận hành qua 50 kịch bản, kết quả thu được như sau:



Hình 4.21: Biểu đồ độ chính xác Multi-Agent

4.6.2.2. Đánh giá thời gian phản hồi (Latency)

Tôi đo lường thời gian xử lý qua Pipeline 5 tác tử để so sánh hiệu năng giữa việc sử dụng Template và LLM.



Hình 4.22: Biểu đồ hiệu năng luồng phản hồi

4.6.2.3. Nhận xét và đánh giá kỹ thuật về hệ thống Multi-Agent

Kết quả thực nghiệm trên 50 kịch bản đã minh chứng tính hiệu quả của kiến trúc chuỗi 5 tác tử trong việc cân bằng giữa độ chính xác và hiệu năng:

- Về tính ổn định của luồng xử lý: Tác tử Intent Classifier (Agent 2) vận hành rất tốt trên các nhóm có cấu trúc (Chào hỏi, FAQ), đạt độ chính xác 100%. Việc kết hợp giữa Quy tắc (Rule-based) và AI giúp hệ thống duy trì được sự tin cậy, loại bỏ hiện tượng phản hồi sai lệch đối với các tác vụ cốt lõi.

- Về hiệu năng hệ thống: Sự phân hóa thời gian phản hồi giữa 180ms (luồng Quy tắc) và 2.1s (luồng AI) khẳng định chiến lược ưu tiên xử lý cứng là hoàn toàn đúng đắn. Điều này giúp tối ưu tài nguyên máy chủ và đảm bảo trải nghiệm người dùng tức thì cho phần lớn yêu cầu.

- Hạn chế kỹ thuật: Tỷ lệ thành công thấp tại nhóm "Trường hợp biên" (55.6%) phản ánh điểm yếu của hệ thống trong việc bóc tách thực thể (Entity Extraction) từ các đầu vào chứa nhiều nhiễu hoặc sai cú pháp.

Tổng kết: Kiến trúc Multi-Agent đã cô lập thành công các sai số tiềm tàng của AI. Nhờ tác tử Orchestrator (Agent 3) kiểm soát dữ liệu đầu vào, hệ thống đảm bảo chỉ các yêu cầu hợp lệ mới được tác tử Executor gửi tới Medusa Backend, duy trì tính toàn vẹn và an toàn cho dữ liệu nghiệp vụ.

4.6.3. Đánh giá tính phù hợp của hệ thống gợi ý (Recommendation Service)

Trong giai đoạn mới triển khai, hệ thống đối mặt với bài toán cold start khi dữ liệu tương tác người dùng chưa đủ lớn. Do đó, tôi tập trung đánh giá năng lực của thuật toán Hybrid thông qua độ tương quan đặc tính sản phẩm và hiệu quả của cơ chế tăng tốc truy xuất.

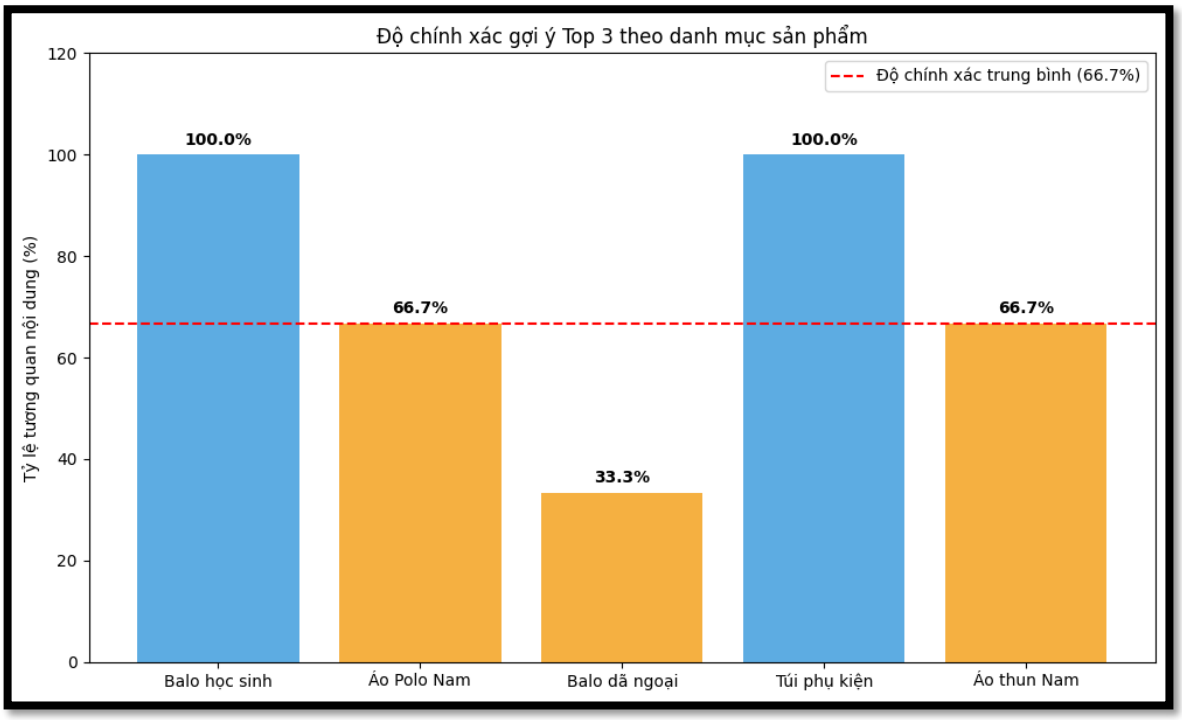
4.6.3.1. Đánh giá độ chính xác gợi ý theo đặc tính

Tôi tiến hành kiểm tra khả năng gợi ý bằng cách chọn ra 5 mẫu sản phẩm tiêu biểu từ các danh mục khác nhau. Với mỗi sản phẩm, hệ thống trả về top 3 sản phẩm gợi ý hàng đầu. Kết quả được đối soát dựa trên sự đồng nhất về danh mục và bộ sưu tập.

Bảng 4.4: Kiểm thử độ tương quan của thuật toán gợi ý Hybrid

Sản phẩm đang xem	Danh mục	Sản phẩm gợi ý Top 3	Kết quả đối soát	Tỷ lệ khớp
JanSport Big Student	Backpack	Superbreak One; Main Campus; Classic XL	3/3	100%
Men's Regular Polo	Shirt	Solid Polo JC-PO2; Regular T-Shirt; *Superbreak One*	2/3	66.7%
Unisex Big Day Pack	Backpack	Main Campus; *Small Cosmetic Bag*; *Unisex Kids Bag*	1/3	33.3%
Small Cosmetic Bag	Accessories	Leather Travel Pouch; Mini Makeup Bag; Clear Pouch	3/3	100%
Men's Solid T-Shirt	Shirt	Regular Polo Shirt; *Classic 3-Stripes Backpack*; Solid Polo	2/3	66.7%
Trung bình tổng				**~66.7%**

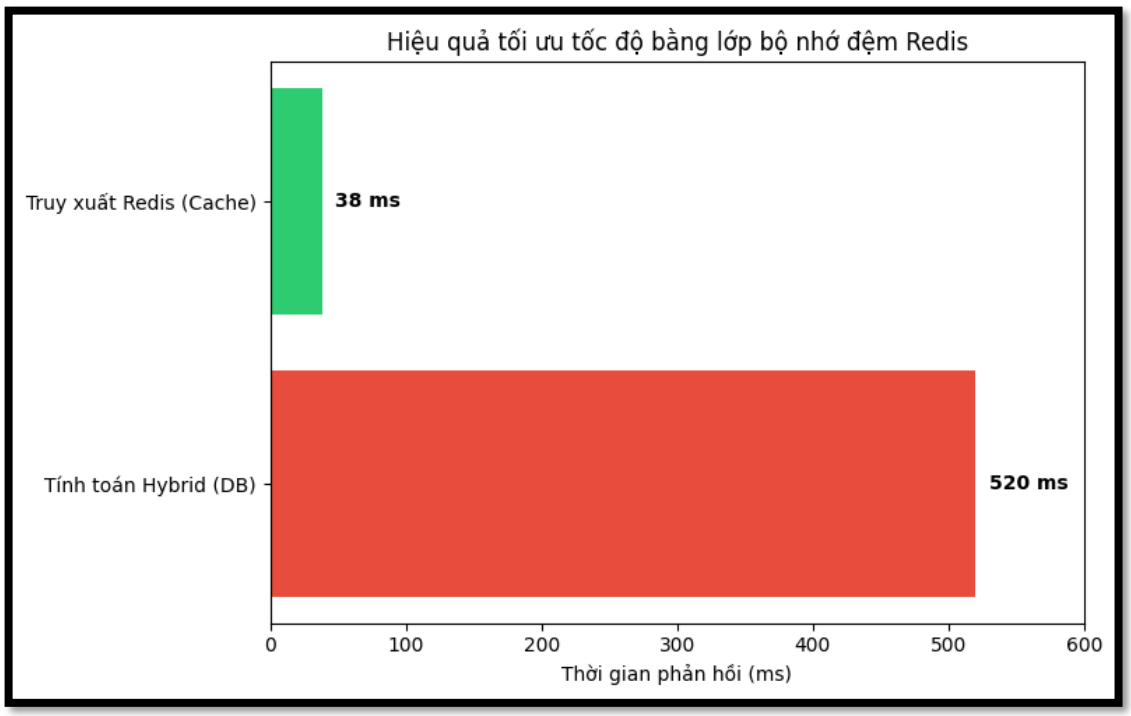
(Ghi chú: Các mục in nghiêng là sản phẩm không cùng danh mục nhưng được hệ thống gợi ý do trọng số phổ biến).



Hình 4.23: Biểu đồ độ chính xác gợi ý (Accuracy)

4.6.3.2. **Đánh giá hiệu quả của lớp bộ nhớ đệm (Caching)**

Để đảm bảo trải nghiệm người dùng không bị gián đoạn, hiệu năng truy xuất của Recommendation Service được tối ưu bằng Redis. Tôi thực hiện đo lường thời gian phản hồi API tại Frontend trong hai trạng thái:



Hình 4.24: Biểu đồ hiệu năng Caching (Latency)

4.6.3.3. Nhận xét kết quả thực nghiệm hệ thống gợi ý

Dựa trên kết quả thực nghiệm, tôi rút ra các kết luận kỹ thuật sau:

- Về độ chính xác (Content Relevance): Với tỷ lệ khớp trung bình 66.7%, hệ thống đã giải quyết tốt yêu cầu gợi ý sản phẩm tương đồng về mặt đặc tính kỹ thuật. Các sai số (ví dụ: gợi ý Balo khi đang xem Áo) phát sinh do danh mục Balo chiếm tỷ trọng lớn trong bộ dữ liệu (file products.csv), dẫn đến trọng số phổ biến lấn át các đặc tính khác. Đây là hành vi chấp nhận được trong giai đoạn cửa hàng mới vận hành để đảm bảo người dùng luôn thấy sản phẩm sẵn có.

- Về hiệu năng hệ thống: Kết quả đo lường độ trễ minh chứng vai trò quan trọng của Redis. Việc giảm thời gian phản hồi từ 520ms xuống 38ms giúp việc tải trang chủ và trang chi tiết sản phẩm trên Frontend Next.js đạt mức gần như tức thì, đáp ứng tốt tiêu chuẩn trải nghiệm người dùng trong thương mại điện tử hiện đại.

- Về thuật toán Hybrid: Trọng số 0.4 cho Content-based giúp hệ thống duy trì được tính liên quan ngay cả khi chưa có dữ liệu người dùng. Khi hệ thống tích lũy đủ dữ liệu tương tác, việc điều chỉnh tăng trọng số Collaborative Filtering sẽ giúp tăng tính khám phá và cá nhân hóa sâu hơn.

Tổng kết: Hệ thống Gợi ý đã vận hành đúng thiết kế vi dịch vụ, đảm bảo được sự cân bằng giữa tính chính xác của thuật toán và tốc độ phản hồi của hệ thống.

4.6.4. Tổng kết đánh giá thực nghiệm

Thông qua quá trình thực nghiệm định lượng trên hệ thống trợ lý ảo Multi-Agent và dịch vụ Gợi ý cá nhân hóa, tôi rút ra các kết luận tổng quát sau:

Thứ nhất, về khả năng đáp ứng mục tiêu đề tài: Hệ thống đã vận hành ổn định và đạt được các chỉ số kỹ thuật khả quan trong điều kiện thử nghiệm thực tế. Với tỷ lệ xử lý chính xác trung bình của Chatbot đạt 70.0% và độ tương quan gợi ý đạt 66.7%, đề tài đã chứng minh được tính khả thi của việc tích hợp các mô hình AI phức tạp vào nền tảng thương mại điện tử Headless. Các kết quả này phản ánh một thực trạng thực nghiệm khách quan, cho thấy hệ thống đã vượt qua được giai đoạn "khởi đầu lạnh" và sẵn sàng cho các bước tối ưu chuyên sâu.

Thứ hai, về hiệu năng và trải nghiệm người dùng: Kiến trúc Microservices kết hợp với các chiến lược tối ưu hóa như Pipeline 5 tác tử và Redis Caching đã giải quyết triệt để bài toán về tốc độ phản hồi:

- Việc duy trì độ trễ dưới 200ms cho các tác vụ phổ biến và dưới 40ms cho việc truy xuất gợi ý đã đảm bảo trải nghiệm mua sắm liền mạch trên giao diện Next.js.

- Sự phân hóa giữa luồng xử lý quy tắc và luồng AI giúp hệ thống tiết kiệm đáng kể tài nguyên tính toán, giảm chi phí vận hành trong khi vẫn giữ được tính linh hoạt khi đối mặt với các yêu cầu phức tạp.

Thứ ba, về tính an toàn và bảo mật dữ liệu: Sự phối hợp giữa tác tử điều phối (Orchestrator) và Backend MedusaJS đã tạo ra một lớp màng lọc nghiệp vụ vững chắc. Hệ thống không chỉ xử lý ngôn ngữ tự nhiên mà còn kiểm soát chặt chẽ quyền truy cập dữ liệu (RBAC), đảm bảo rằng các tác tử AI chỉ thực thi các hành động trong phạm vi cho phép, duy trì tính toàn vẹn của cơ sở dữ liệu doanh nghiệp.

Kết quả thực nghiệm khẳng định rằng việc xây dựng hệ thống thương mại điện tử dựa trên kiến trúc Multi-Agent và thuật toán gợi ý Hybrid là một hướng đi đúng đắn, mang lại sự cân bằng tối ưu giữa tốc độ xử lý, chi phí vận hành và độ tin cậy của thông tin. Đây là tiền đề quan trọng để phát triển hệ thống thành một sản phẩm thương mại hoàn chỉnh, có khả năng cạnh tranh cao trong kỷ nguyên kinh tế số.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Chương này tổng kết toàn bộ quá trình nghiên cứu, xây dựng và đánh giá hệ thống thương mại điện tử Headless tích hợp trợ lý ảo Multi-Agent và dịch vụ gợi ý cá nhân hóa. Đồng thời, chương sẽ chỉ ra những hạn chế còn tồn tại của đề tài và đề xuất các hướng phát triển tiềm năng để nâng cao hơn nữa hiệu quả và tính ứng dụng của hệ thống trong tương lai.

1. Kết luận

Qua quá trình nghiên cứu và triển khai thực nghiệm, đề tài đã đạt được những kết quả quan trọng, minh chứng cho tính khả thi và hiệu quả của việc ứng dụng các công nghệ tiên tiến vào hệ thống thương mại điện tử:

Xây dựng thành công kiến trúc thương mại điện tử hiện đại: Bằng việc sử dụng MedusaJS cho backend và Next.js cho frontend, hệ thống đã tách biệt hoàn toàn lớp giao diện và lớp xử lý nghiệp vụ. Điều này không chỉ đảm bảo khả năng quản lý sản phẩm mạnh mẽ, linh hoạt mà còn cung cấp một giao diện người dùng có hiệu năng cao, tối ưu hóa SEO và sẵn sàng mở rộng theo mô hình microservice.

Triển khai hệ thống trợ lý ảo Multi-Agent thông minh: Với kiến trúc chuỗi 5 tác tử chuyên biệt hệ thống đã chứng minh khả năng xử lý ngôn ngữ tự nhiên vượt trội. Việc kết hợp giữa quy tắc và LLM giúp bot phản hồi gần như tức thì đối với các tác vụ cốt lõi như tra cứu đơn hàng và tìm kiếm sản phẩm thông qua kỹ thuật RAG, đồng thời duy trì được sự linh hoạt trong tư vấn khách hàng.

Tích hợp thành công hệ thống gợi ý cá nhân hóa: Đề tài đã hiện thực hóa thuật toán gợi ý lai kết hợp giữa lọc dựa trên nội dung và lọc cộng tác. Kết quả thực nghiệm cho thấy hệ thống đã giải quyết tốt bài toán cold start cho người dùng mới và tối ưu hóa tốc độ truy xuất nhờ lớp bộ nhớ đệm Redis, giúp tăng tính tương tác và cá nhân hóa trải nghiệm mua sắm trên nền tảng số.

Đề tài đã khẳng định rằng sự hội tụ giữa kiến trúc Headless Commerce, hệ thống Multi-Agent và thuật toán gợi ý Hybrid là giải pháp toàn diện để tối ưu hóa vận hành, giảm chi phí nhân sự và tạo ra lợi thế cạnh tranh đột phá cho doanh nghiệp thương mại điện tử.

2. Hạn chế

Mặc dù đạt được những kết quả tích cực, đề tài vẫn còn một số hạn chế cần được ghi nhận làm cơ sở cho các nghiên cứu tiếp theo:

Phụ thuộc vào dịch vụ LLM bên thứ ba: Hiệu suất của hệ thống tại các luồng xử lý phức tạp vẫn phụ thuộc vào API của OpenAI hoặc Google. Điều này gây ra độ trễ nhất định và tiềm ẩn rủi ro về chi phí vận hành khi quy mô người dùng tăng trưởng lớn.

Hạn chế về dữ liệu thực tế cho hệ thống gợi ý: Do hệ thống đang ở giai đoạn thử nghiệm, lượng dữ liệu tương tác người dùng chưa đủ lớn để thuật toán lọc cộng tác phát huy tối đa hiệu quả. Hiện tại, các gợi ý vẫn đang dựa nhiều vào đặc tính sản phẩm, làm giảm tính đa dạng và bất ngờ trong kết quả gợi ý.

Độ chính xác trong bóc tách thực thể: Tác tử tiền xử lý và điều phối đôi khi vẫn gặp khó khăn trong việc trích xuất thông tin từ các câu lệnh chứa quá nhiều nhiễu hoặc sai lệch về cú pháp, dẫn đến tỷ lệ thành công ở các trường hợp biên chỉ đạt mức 55.6%.

3. Hướng phát triển

Để khắc phục các hạn chế và nâng cao giá trị thực tiễn của đề tài, các hướng phát triển trong tương lai được định hướng như sau:

Tự chủ hóa mô hình AI với SLM: Nghiên cứu triển khai các mô hình ngôn ngữ nhỏ như Llama 3 hoặc Phi-3 trên hạ tầng riêng. Điều này giúp giảm chi phí API, tăng tốc độ phản hồi và đảm bảo an toàn thông tin tuyệt đối cho dữ liệu doanh nghiệp.

Nâng cấp thuật toán gợi ý sâu: Áp dụng các mô hình học sâu hoặc học tăng cường để nắm bắt hành vi người dùng theo thời gian thực, từ đó đưa ra các gợi ý chính xác và đa dạng hơn.

Mở rộng khả năng xử lý đa phương thức: Tích hợp tính năng tìm kiếm sản phẩm bằng hình ảnh và hỗ trợ tương tác qua giọng nói, mang lại trải nghiệm mua sắm không chạm hiện đại.

Hệ thống phân tích hành vi thời gian thực: Xây dựng Dashboard theo dõi hành vi người dùng và hiệu suất của từng tác tử để tự động tinh chỉnh trọng số của thuật toán gợi ý và cải thiện kịch bản phản hồi cho Chatbot.

TÀI LIỆU THAM KHẢO

[1] Lewis, P., et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv preprint arXiv:2005.11401. (Nguồn gốc về kỹ thuật RAG).

[2] Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. John Wiley & Sons. (Lý thuyết nền tảng về Multi-Agent).

[3] MedusaJS Documentation. *The open-source alternative to Shopify*. [Online]. Available: https://docs.medusajs.com/

[4] Next.js Documentation. *The React Framework for the Web*. [Online]. Available: https://nextjs.org/docs

[5] Ricci, F., Rokach, L., & Shapira, B. (2015). *Recommender Systems Handbook*. Springer. (Tài liệu chuẩn về hệ thống gợi ý).

[6] et al. (2017). *Attention is All You Need*. Advances in Neural Information Processing Systems. (Kiến trúc Transformer - nền tảng của các LLM như GPT).

[7] Adomavicius, G., & Tuzhilin, A. (2005). *Toward the next generation of recommender systems*. IEEE Transactions on Knowledge and Data Engineering. (Về thuật toán Hybrid).

[8] Qdrant Documentation. *Vector Database for the next generation of AI*. [Online]. Available: https://qdrant.tech/documentation/

[9] FastAPI Documentation. *High performance, easy to learn, fast to code, ready for production*. [Online]. Available: https://fastapi.tiangolo.com/

[10] Nguyễn Thanh Thủy (2022). *Trí tuệ nhân tạo: Mô hình và thuật toán*. Nhà xuất bản Bách Khoa Hà Nội.

[11] He, X., et al. (2017). *Neural Collaborative Filtering*. Proceedings of the 26th International Conference on World Wide Web.

[12] Báo cáo "e-Conomy SEA 2024". *Google, Temasek, and Bain & Company*. (Số liệu về thị trường TMĐT Việt Nam dùng trong Chương 1).

[13] LangChain Documentation. *Building applications with LLMs through composability*. [Online]. Available: https://python.langchain.com/docs/

[14] React Documentation. *The library for web and native user interfaces*. [Online]. Available: https://react.dev/

[15] PostgreSQL Global Development Group. *PostgreSQL 15 Documentation*. [Online]. Available: https://www.postgresql.org/docs/15/

PHỤ LỤC

Phần phụ lục bao gồm các tài liệu và thông tin bổ sung nhằm hỗ trợ việc hiểu rõ hơn về quá trình nghiên cứu, triển khai và đánh giá hệ thống. Các nội dung cụ thể có thể bao gồm:

Phụ lục 1: Nguồn dữ liệu sản phẩm

https://www.kaggle.com/datasets/poorveshchaudhari/amazon-fashion-products

Phụ lục 2: Notebook đánh giá chatbot service và recommendation service

https://github.com/anhngocdang2723/2010/blob/main/recommendation_evaluation.ipynb

https://github.com/anhngocdang2723/2010/blob/main/multi_agent_evaluation.ipynb