TRƯỜNG ĐẠI HỌC TRÀ VINH
TRƯỜNG KỸ THUẬT VÀ CÔNG NGHỆ

ISO 9001:2015

NGUYỄN ĐẠI HOÀNG PHÚC

NGHIÊN CỨU RAG VÀ XÂY DỰNG CHATBOT HỖ TRỢ MÔN CƠ SỞ DỮ LIỆU

ĐỒ ÁN TỐT NGHIỆP
NGÀNH CÔNG NGHỆ THÔNG TIN

Trà Vinh, tháng 5 năm 2025

TRƯỜNG ĐẠI HỌC TRÀ VINH
TRƯỜNG KỸ THUẬT VÀ CÔNG NGHỆ

NGHIÊN CỨU RAG VÀ XÂY DỰNG CHATBOT HỖ TRỢ MÔN CƠ SỞ DỮ LIỆU

ĐỒ ÁN TỐT NGHIỆP
NGÀNH CÔNG NGHỆ THÔNG TIN

Giảng viên hướng dẫn : **TS. NGUYỄN BẢO ÂN**
Sinh viên thực hiện: **NGUYỄN ĐẠI HOÀNG PHÚC**
Mã số sinh viên: **110121087**
Lớp: **DA21TTB**

Trà Vinh, tháng 5 năm 2025

LỜI MỞ ĐẦU

Trong kỷ nguyên số hóa và sự phát triển vượt bậc của Trí tuệ nhân tạo (AI), các ứng dụng thông minh ngày càng thâm nhập sâu rộng vào mọi lĩnh vực của đời sống, đặc biệt là giáo dục. Việc học tập và giảng dạy truyền thống đang dần được bổ sung và nâng cao hiệu quả bởi các công cụ công nghệ hiện đại, trong đó chatbot nổi lên như một giải pháp tiềm năng, mang đến trải nghiệm học tập cá nhân hóa và tương tác cao.

Môn học Cơ sở dữ liệu là một trong những môn học nền tảng và quan trọng đối với sinh viên ngành Công nghệ thông tin. Tuy nhiên, với khối lượng kiến thức lớn, bao gồm nhiều khái niệm trừu tượng và yêu cầu kỹ năng thực hành cao, không ít sinh viên gặp khó khăn trong quá trình tiếp thu và tự học. Việc tìm kiếm thông tin, giải đáp thắc mắc nhanh chóng và chính xác từ nguồn tài liệu chính thống là một nhu cầu thiết yếu.

Nhận thức được tầm quan trọng của vấn đề trên, cùng với sự đam mê nghiên cứu các công nghệ AI mới, đặc biệt là kỹ thuật Retrieval-Augmented Generation (RAG) với khả năng nâng cao tính chính xác và độ tin cậy của các mô hình ngôn ngữ lớn, em đã quyết định chọn đề tài: "Nghiên cứu RAG và xây dựng Chatbot hỗ trợ môn Cơ sở dữ liệu" cho đồ án tốt nghiệp của mình.

Đề tài này không chỉ là cơ hội để em vận dụng những kiến thức đã học, tìm hiểu sâu hơn về một lĩnh vực công nghệ tiên tiến, mà còn mong muốn góp phần xây dựng một công cụ học tập hữu ích, hỗ trợ thiết thực cho các bạn sinh viên trong quá trình chinh phục môn học Cơ sở dữ liệu.

LỜI CẢM ƠN

Để hoàn thành đồ án tốt nghiệp với đề tài "Nghiên cứu RAG và xây dựng Chatbot hỗ trợ môn Cơ sở dữ liệu", bên cạnh sự nỗ lực của bản thân, em đã nhận được rất nhiều sự hướng dẫn, giúp đỡ và động viên quý báu từ quý thầy cô, gia đình và bạn bè.

Trước tiên, em xin bày tỏ lòng biết ơn sâu sắc và chân thành nhất đến TS. Nguyễn Bảo Ân, giảng viên hướng dẫn trực tiếp của em. Thầy đã tận tình chỉ bảo, định hướng, cung cấp những kiến thức chuyên môn quý giá và tạo mọi điều kiện thuận lợi cho em trong suốt quá trình thực hiện đề tài, từ những bước đầu hình thành ý tưởng cho đến khi hoàn thiện đồ án. Những góp ý sâu sắc và kịp thời của Thầy là nguồn động lực lớn giúp em vượt qua những khó khăn và hoàn thành tốt nhiệm vụ nghiên cứu của mình.

Em cũng xin trân trọng cảm ơn Ban Giám hiệu Trường Đại học Trà Vinh, quý thầy cô trong Trường Kỹ thuật và Công nghệ, đặc biệt là các thầy cô thuộc Khoa Công nghệ Thông tin đã tận tâm giảng dạy, truyền đạt kiến thức và kinh nghiệm trong suốt những năm học vừa qua. Những kiến thức nền tảng vững chắc mà thầy cô trang bị chính là hành trang quý báu để em tự tin thực hiện đồ án này và vững bước trên con đường sự nghiệp sau này.

Em xin gửi lời cảm ơn đến các bạn sinh viên lớp DA21TTB đã cùng đồng hành, chia sẻ kiến thức, kinh nghiệm và động viên em trong suốt quá trình học tập và thực hiện đồ án.

Cuối cùng, con xin gửi lời biết ơn vô hạn đến gia đình, đặc biệt là cha mẹ, đã luôn yêu thương, tin tưởng, động viên và là chỗ dựa tinh thần vững chắc, tạo mọi điều kiện tốt nhất cho con trong suốt quá trình học tập và rèn luyện.

Mặc dù đã rất cố gắng để hoàn thiện đồ án một cách tốt nhất, nhưng do kiến thức và kinh nghiệm thực tiễn còn hạn chế, đồ án chắc chắn không tránh khỏi những thiếu sót. Em rất mong nhận được những ý kiến đóng góp quý báu từ quý thầy cô và các bạn để đề tài được hoàn thiện hơn.

MỤC LỤC

DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH

KÍ HIỆU CÁC CỤM TỪ VIẾT TẮT

Từ viết tắt	Ý nghĩa
AI	Artificial Intelligence (Trí tuệ nhân tạo)
API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Deployment
CSDL	Cơ sở dữ liệu
CSS	Cascading Style Sheets
DBMS	Database Management System (Hệ quản trị CSDL)
ER	Entity-Relationship (Thực thể-Liên kết)
HTML	HyperText Markup Language
HTTP/HTTPS	Hypertext Transfer Protocol/Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
LLM	Large Language Model (Mô hình ngôn ngữ lớn)
NLP	Natural Language Processing (Xử lý ngôn ngữ tự nhiên)
RAG	Retrieval-Augmented Generation
SQL	Structured Query Language (Ngôn ngữ truy vấn có cấu trúc)
UI	User Interface
UX	User Experience
VPS	Virtual Private Server

CHƯƠNG 1 TỔNG QUAN

1.1 Đặt vấn đề và bối cảnh:

Trong kỷ nguyên chuyển đổi số, việc ứng dụng trí tuệ nhân tạo (AI) vào giáo dục ngày càng phổ biến nhằm nâng cao hiệu quả học tập. Một trong những ứng dụng nổi bật là chatbot học tập – hệ thống trả lời tự động theo ngôn ngữ tự nhiên.

Tại các trường đại học, sinh viên ngành CNTT thường gặp khó khăn với các môn nền tảng như Cơ sở dữ liệu (CSDL) do khối lượng kiến thức lớn và tính trừu tượng cao. Tuy nhiên, sinh viên ít có cơ hội được hỗ trợ ngoài giờ học chính khóa.

Trong bối cảnh đó, nhu cầu về một chatbot học tập thông minh, hỗ trợ hỏi đáp 24/7 trở nên cấp thiết. Tuy nhiên, các chatbot truyền thống thường thiếu tính chính xác và minh bạch thông tin. Kỹ thuật Retrieval-Augmented Generation (RAG) được xem là giải pháp phù hợp, khi kết hợp mô hình ngôn ngữ lớn (LLM) với khả năng truy xuất từ kho tri thức để sinh câu trả lời chính xác, có trích nguồn rõ ràng.

Đề tài này nhằm nghiên cứu và xây dựng một chatbot RAG hỗ trợ sinh viên học môn CSDL hiệu quả và chủ động hơn.

1.2 Nhu cầu hỗ trợ học tập môn Cơ sở dữ liệu và vai trò của chatbot:

Môn CSDL bao gồm nhiều kiến thức lý thuyết lẫn thực hành như: thiết kế mô hình dữ liệu, SQL, chuẩn hóa dữ liệu,… khiến việc tự học gặp khó khăn. Sinh viên cần được giải đáp nhanh chóng các thắc mắc trong quá trình học.

Chatbot học tập đóng vai trò như một "trợ giảng ảo", có thể trả lời các câu hỏi liên quan đến nội dung môn học, đồng thời cung cấp nguồn trích dẫn cụ thể. Chatbot sử dụng RAG giúp đảm bảo độ chính xác và độ tin cậy cao hơn so với việc tìm kiếm tự do trên Internet. Ngoài ra, hệ thống có thể lưu lại lịch sử hội thoại để phục vụ ôn tập và nâng cao trải nghiệm người dùng.

1.3 Mục tiêu, đối tượng và phạm vi nghiên cứu:

1.3.1 Mục tiêu:

Mục tiêu của đề tài là nghiên cứu và phát triển thành công một hệ thống chatbot RAG hỗ trợ sinh viên trong môn Cơ sở dữ liệu, với các tiêu chí chính như sau:

- Nghiên cứu lý thuyết RAG: Tìm hiểu chuyên sâu về nguyên lý và kiến trúc của Retrieval-Augmented Generation, bao gồm các thành phần cốt lõi như phương pháp tạo vector embedding cho văn bản, cơ sở dữ liệu vector để lưu trữ và tìm kiếm, các mô hình ngôn ngữ lớn (LLM) cho việc sinh văn bản. Đồng thời nghiên

cứu các kỹ thuật nâng cao liên quan như tìm kiếm ngữ nghĩa (semantic search) và reranking (xếp hạng lại kết quả truy xuất).

- Ứng dụng NLP trong xử lý tài liệu: Áp dụng các kỹ thuật xử lý ngôn ngữ tự nhiên để tiền xử lý tập tài liệu môn CSDL (loại bỏ nhiễu, tách từ, chuẩn hóa câu hỏi, v.v.) và tạo vector embedding cho các đơn vị tri thức trích xuất từ tài liệu.
- Xây dựng cơ sở tri thức dưới dạng vector: Thu thập các tài liệu liên quan đến môn CSDL (giáo trình, slide bài giảng, bài tập, đề thi, v.v. ở các định dạng PDF, DOCX, TXT, SQL…) và chuyển chúng thành vector embeddings lưu trữ trong một kho dữ liệu vector (dự kiến dùng Qdrant). Đây sẽ là "bộ nhớ ngoài" để chatbot truy xuất thông tin.
- Thiết kế, xây dựng chatbot RAG hoàn chỉnh: Phát triển một hệ thống chatbot tích hợp đầy đủ pipeline RAG. Hệ thống sử dụng tìm kiếm ngữ nghĩa trên kho vector (với Qdrant) để tìm nội dung liên quan, áp dụng kỹ thuật rerank để chọn lọc ngữ cảnh phù hợp nhất, sau đó kết hợp ngữ cảnh này trong prompt và gọi mô hình ngôn ngữ lớn (dự kiến sử dụng API mô hình Gemini của Google) để sinh ra câu trả lời. Chatbot phải có khả năng trả lời chính xác các câu hỏi trong phạm vi kiến thức môn học và trích dẫn nguồn gốc của thông tin trong câu trả lời.
- Phát triển giao diện người dùng web thân thiện: Xây dựng một giao diện web (dự kiến sử dụng Next.js – một framework React-based, phát triển bởi Vercel, giúp xây dựng ứng dụng web với khả năng Server-Side Rendering (SSR) và Static Site Generation (SSG), cùng nhiều tính năng cao cấp như Incremental Static Regeneration (ISR)) để người dùng tương tác với chatbot một cách thuận tiện. Giao diện cần có sự phân quyền rõ ràng:

  + Đối với người quản trị (Admin): Cung cấp giao diện chuyên biệt để quản lý tập trung toàn bộ kho tri thức của hệ thống. Admin là người duy nhất có quyền tải lên, cập nhật và xóa các tài liệu (giáo trình, slide, bài tập).

  + Đối với người dùng cuối (Sinh viên): Cung cấp giao diện hỏi đáp, cho phép sinh viên truy vấn thông tin từ kho tri thức chung đã được Admin kiểm duyệt, đảm bảo tính nhất quán và độ tin cậy của câu trả lời.
- Tích hợp giải pháp lưu trữ và quản lý hội thoại: Sử dụng một cơ sở dữ liệu hoặc dịch vụ lưu trữ (dự kiến Supabase) để lưu lịch sử hội thoại và có thể quản lý dữ liệu người dùng nếu cần. Mục tiêu là mỗi phiên tương tác của người dùng với

chatbot đều được lưu lại để phục vụ nghiên cứu hành vi người dùng và cải thiện hệ thống sau này.

1.3.2 Đối tượng nghiên cứu:

Đối tượng nghiên cứu chính của đề tài là các kỹ thuật và công nghệ liên quan đến RAG và việc áp dụng chúng trong bối cảnh cụ thể (hệ thống hỏi đáp cho môn học CSDL). Bao gồm: thuật toán và mô hình NLP để xử lý ngôn ngữ tiếng Việt, mô hình embedding câu văn, cơ sở dữ liệu vector, mô hình ngôn ngữ lớn (đặc biệt là API của các mô hình hiện đại như Google Gemini), các framework xây dựng chatbot (ví dụ LangChain), v.v. Bên cạnh đó, đề tài cũng nghiên cứu nhu cầu và hành vi người dùng (sinh viên) trong việc sử dụng chatbot hỗ trợ học tập.

1.3.3 Phạm vi nghiên cứu:

Phạm vi nghiên cứu được giới hạn ở việc hỗ trợ nội dung môn Cơ sở dữ liệu ở trình độ đại học cơ bản. Nghĩa là chatbot sẽ chỉ trả lời các câu hỏi nằm trong phạm vi kiến thức môn học này (ví dụ: các khái niệm về mô hình quan hệ, SQL, thiết kế CSDL quan hệ, tối ưu truy vấn cơ bản…). Hệ thống không được thiết kế để trả lời các câu hỏi ngoài phạm vi (ví dụ kiến thức chuyên sâu ngoài giáo trình hoặc các lĩnh vực CNTT khác). Về ngôn ngữ, chatbot sẽ chủ yếu hoạt động bằng tiếng Việt, do tài liệu môn học chủ yếu bằng tiếng Việt. Một số thuật ngữ chuyên môn có thể bằng tiếng Anh nhưng sẽ được giải thích bằng tiếng Việt khi cần. Ngoài ra, do giới hạn thời gian và nguồn lực, đề tài tập trung vào việc tích hợp các mô hình và dịch vụ có sẵn thay vì tự huấn luyện mô hình mới. Ví dụ: sử dụng mô hình embedding và LLM có sẵn qua API, thay vì đào tạo mô hình NLP từ đầu. Điều này giúp bám sát mục tiêu xây dựng một hệ thống nguyên mẫu hoàn thiện.

1.4 Cấu trúc đồ án:

- Chương 1: Tổng quan đề tài, nhu cầu, mục tiêu và phạm vi nghiên cứu.
- Chương 2: Cơ sở lý thuyết về NLP, RAG, vector DB, LLM,... và công nghệ sử dụng.
- Chương 3: Phân tích yêu cầu và thiết kế hệ thống.
- Chương 4: Triển khai hệ thống: backend, frontend, CICD.
- Chương 5: Đánh giá kết quả và hiệu quả của hệ thống chatbot.
- Chương 6: Kết luận và đề xuất hướng phát triển tiếp theo.

CHƯƠNG 2 CƠ SỞ LÝ THUYẾT

2.1 Giới thiệu chung về AI, NLP và Chatbot

Trí tuệ nhân tạo (Artificial Intelligence – AI) là lĩnh vực khoa học máy tính nghiên cứu cách xây dựng các hệ thống hoặc chương trình có khả năng thực hiện các tác vụ thông minh như con người. Những tác vụ này bao gồm học tập từ dữ liệu, suy luận logic, nhận diện mẫu, hiểu ngôn ngữ tự nhiên và ra quyết định. AI bao gồm nhiều nhánh, trong đó có học máy (Machine Learning), học sâu (Deep Learning), thị giác máy tính (Computer Vision), và xử lý ngôn ngữ tự nhiên (Natural Language Processing – NLP).

Xử lý ngôn ngữ tự nhiên (NLP) là một nhánh quan trọng của AI, tập trung vào việc phát triển các phương pháp và mô hình cho phép máy tính hiểu, phân tích và tạo sinh ngôn ngữ của con người. NLP kết hợp kiến thức từ ngôn ngữ học, khoa học máy tính và học máy để giải quyết nhiều bài toán quan trọng như:

- Phân tích hình thái từ (morphological analysis): xác định cấu trúc và các thành phần của từ.
- Gán nhãn từ loại (part-of-speech tagging): phân loại từ thành danh từ, động từ, tính từ,…
- Phân tích cú pháp (syntactic parsing): xác định cấu trúc ngữ pháp của câu.
- Phân tích ngữ nghĩa (semantic analysis): hiểu nội dung và ý nghĩa của câu.

Từ những tác vụ cơ bản trên, NLP được ứng dụng vào nhiều lĩnh vực như: tự động tóm tắt văn bản, trích xuất thông tin, dịch máy, phân loại văn bản, hệ thống hỏi đáp, và đặc biệt là giao diện truy vấn cơ sở dữ liệu bằng ngôn ngữ tự nhiên.

Trong số các ứng dụng nổi bật của AI và NLP, **Chatbot** là một đại diện tiêu biểu. Chatbot là các chương trình máy tính có khả năng giao tiếp với con người thông qua ngôn ngữ tự nhiên. Sự phát triển mạnh mẽ của các mô hình học sâu, đặc biệt là mô hình Transformer và các mô hình ngôn ngữ lớn (Large Language Models – LLMs), đã thúc đẩy sự ra đời của các chatbot hiện đại với khả năng hiểu và sinh phản hồi ngày càng tự nhiên, gần giống con người.

Tuy nhiên, các mô hình LLM thuần túy vẫn còn tồn tại nhiều hạn chế khi triển khai trong các tác vụ yêu cầu độ chính xác cao (knowledge-intensive tasks). Chúng có thể gặp phải hiện tượng "hallucination" – sinh ra thông tin sai nhưng có vẻ hợp lý – hoặc thiếu tính cập nhật do bị giới hạn bởi tập dữ liệu huấn luyện cố định. Ngoài

ra, việc thiếu minh bạch về nguồn gốc thông tin cũng gây khó khăn trong việc kiểm chứng tính xác thực – điều đặc biệt quan trọng trong các lĩnh vực như giáo dục và học thuật.

2.2 Kiến trúc RAG:

Để giải quyết các hạn chế của LLM đã nêu, đề tài áp dụng kiến trúc Retrieval-Augmented Generation (RAG). Đây là phương pháp được Patrick Lewis và các cộng sự tại Facebook AI Research giới thiệu lần đầu vào năm 2020 [1], kết hợp một mô hình ngôn ngữ lớn với một hệ thống truy xuất thông tin bên ngoài.

Về tổng quan, một hệ thống RAG bao gồm hai thành phần chính:

- Bộ truy xuất (Retriever): Có nhiệm vụ nhận câu hỏi của người dùng và tìm kiếm các đoạn văn bản có nội dung liên quan nhất từ một kho tri thức đã được xây dựng sẵn (ví dụ: giáo trình, slide bài giảng).
- Bộ sinh (Generator): Thường là một LLM, có nhiệm vụ tổng hợp thông tin từ các đoạn văn bản mà bộ truy xuất tìm được để tạo ra một câu trả lời cuối cùng, mạch lạc và chính xác.

Quy trình hoạt động cơ bản của RAG thường gồm bốn bước:

1. Tiếp nhận câu hỏi: Hệ thống nhận câu hỏi bằng ngôn ngữ tự nhiên từ người dùng.
2. Truy xuất thông tin: Bộ truy xuất tìm kiếm trong kho tri thức để lấy ra các đoạn văn bản (ngữ cảnh) phù hợp nhất với câu hỏi.
3. Tăng cường Prompt: Hệ thống kết hợp câu hỏi gốc với các đoạn ngữ cảnh đã truy xuất để tạo thành một prompt đầy đủ và chi tiết.
4. Sinh câu trả lời: Prompt này được đưa vào LLM để sinh ra câu trả lời cuối cùng.

Cách tiếp cận này mô phỏng quá trình con người tra cứu tài liệu tham khảo trước khi trả lời một câu hỏi phức tạp. Nhờ vậy, RAG giúp câu trả lời của chatbot có căn cứ xác thực, giảm thiểu đáng kể hiện tượng "hallucination", đồng thời cho phép trích dẫn nguồn tài liệu đã sử dụng, tăng tính minh bạch và tin cậy.

Trong khuôn khổ đề tài này, kiến trúc RAG được áp dụng để xây dựng một chatbot hỗ trợ sinh viên môn Cơ sở dữ liệu (CSDL). Chatbot sẽ truy xuất thông tin trực tiếp từ các tài liệu môn học như giáo trình, slide, bài tập để cung cấp các câu trả lời chính xác và có nguồn gốc rõ ràng.

Kể từ khi ra đời, RAG đã không ngừng phát triển, với các nghiên cứu mở rộng sang cả dữ liệu đa phương thức, điển hình là UniversalRAG (Yeo et al., 2025) - một kiến trúc kết hợp hình ảnh-văn bản giúp tăng hiệu quả đáng kể trên các bộ dữ liệu phức hợp [2].

2.3 Tiền xử lý văn bản:

Đề tài chủ yếu quan tâm đến xử lý ngôn ngữ văn bản tiếng Việt. Tiếng Việt là ngôn ngữ đơn lập và giàu ngữ điệu, việc xử lý tiếng Việt có những thách thức riêng như tách từ (vì tiếng Việt không dùng dấu cách để phân tách từ đơn lẻ như tiếng Anh) và xử lý dấu thanh. Các bước tiền xử lý văn bản điển hình bao gồm:

- Làm sạch dữ liệu: Loại bỏ những phần không liên quan hoặc gây nhiễu trong văn bản như mã HTML, ký tự đặc biệt, khoảng trắng thừa, v.v. Ví dụ, khi xử lý tài liệu PDF, cần loại bỏ header, footer, số trang, để văn bản sạch sẽ và liên tục.
- Chuẩn hóa văn bản: Đưa văn bản về dạng thống nhất, chẳng hạn chuyển về cùng một dạng chữ (viết thường), chuẩn hóa cách viết thuật ngữ (CSDL vs. cơ sở dữ liệu), thay thế các ký tự đặc biệt bằng ký tự tương ứng (như “–” thành “-”).
- Tách từ và phân tích từ loại (nếu cần): Với tiếng Việt, bước tách từ (word segmentation) rất quan trọng để thu được danh sách các từ/ngữ có nghĩa. Công cụ như VnCoreNLP có thể được sử dụng để tách từ tiếng Việt. Sau đó có thể gán nhãn từ loại để hỗ trợ các bước xử lý sau.
- Loại bỏ từ dừng: Các từ dừng là những từ xuất hiện rất nhiều nhưng ít mang nội dung (ví dụ: “là”, “các”, “những” trong tiếng Việt). Loại bỏ chúng có thể giúp tập trung vào từ khóa nội dung quan trọng hơn trong một số ứng dụng. Tuy nhiên với kỹ thuật hiện đại dùng embedding, nhiều khi có thể giữ lại từ dừng vì mô hình tự học được mức độ quan trọng.
- Stemming/Lemmatization: Với tiếng Việt, khái niệm stemming (rút về gốc từ) ít được áp dụng hơn so với tiếng Anh, do tiếng Việt không biến đổi dạng từ như ngôn ngữ phương Tây.

Sau khi văn bản đã được làm sạch và chuẩn hóa, bước tiếp theo thường là phân đoạn văn bản (text chunking). Ta cần chia tài liệu lớn thành các đoạn nhỏ (chunk) sao cho mỗi đoạn chứa một ý tương đối hoàn chỉnh và độ dài phù hợp với khả năng xử lý của mô hình (ví dụ không quá 256 token để phù hợp với mô hình embedding). Việc phân đoạn có thể dựa trên các đơn vị logic sẵn có của tài liệu: ví dụ chia theo

tiêu đề mục, hoặc chia theo đoạn văn/bullet trong slide. Nếu không có cấu trúc rõ ràng, có thể áp dụng phương pháp chia theo độ dài cố định (ví dụ mỗi đoạn ~200 từ hoặc ~1000 ký tự, có thể chồng lấn một chút giữa các đoạn để không mất ngữ cảnh). Mục tiêu là các đoạn sau khi chia phải tự chứa đủ ngữ cảnh để có thể hiểu độc lập ở mức nhất định, hỗ trợ tốt cho việc tìm kiếm ngữ nghĩa sau này. Quá trình chunking đòi hỏi cân bằng: đoạn quá ngắn có thể không đủ ý, đoạn quá dài sẽ vượt ngưỡng mô hình và gây nhiễu khi tìm kiếm.

Tóm lại, tiền xử lý văn bản là bước nền tảng đảm bảo dữ liệu đầu vào sạch, có cấu trúc, tạo điều kiện cho việc nhúng vector và truy xuất hiệu quả. Những nghiên cứu gần đây nhấn mạnh rằng chất lượng dữ liệu (như độ chính xác của việc parse PDF) ảnh hưởng rất lớn đến hiệu quả hệ thống RAG. Ví dụ, trong một thử nghiệm với tài liệu PDF chuyên ngành, một hệ thống RAG trang bị bộ parser PDF tốt (ChatDOC) đã truy xuất các đoạn chính xác và cải thiện rõ rệt chất lượng trả lời so với hệ thống baseline. Theo Lin et al. (2024), bộ parser ChatDOC nhận diện cấu trúc mục lục, bảng và hình trong PDF, giúp tăng Recall truy xuất +8 % cho tài liệu kỹ thuật, nhờ đó cải thiện đáng kể chất lượng chunking [6]. Do đó, trong thiết kế hệ thống, đề tài đặc biệt chú trọng khâu tiền xử lý và chunking tài liệu nhằm tạo ra cơ sở tri thức tối ưu nhất.

2.4 Kỹ thuật embedding và Sentence Transformers:

Sau khi đã tiền xử lý và phân đoạn văn bản, chúng ta cần chuyển các đoạn văn bản này thành biểu diễn số (vector) để máy tính có thể so sánh độ tương đồng ngữ nghĩa. Kỹ thuật embedding trong NLP cho phép ánh xạ mỗi từ, câu, hoặc đoạn văn thành một vector số học trong không gian nhiều chiều, sao cho các đoạn văn có nội dung hoặc ngữ nghĩa tương tự sẽ có vector ở gần nhau trong không gian đó.

Trước đây, các phương pháp embedding đơn giản như Bag of Words hay TF-IDF biểu diễn văn bản dưới dạng vector độ dài bằng kích thước từ vựng (vô cùng lớn) và chủ yếu dựa trên tần suất từ, không nắm bắt được ý nghĩa ngữ cảnh. Sau này, các mô hình như Word2Vec, GloVe cho phép học vector từ vựng có kích thước cố định (ví dụ 100-300 chiều), trong đó có thể phản ánh một phần ngữ nghĩa (ví dụ “database” gần “SQL” hơn là “apple”). Tuy nhiên, embedding theo từ đơn lẻ vẫn chưa giải quyết được hiện tượng đa nghĩa và ngữ cảnh của từ trong câu.

Bước đột phá đến với sự ra đời của các mô hình Transformer như BERT, RoBERTa,... Những mô hình này cung cấp contextual embedding – vector biểu diễn của từ phụ thuộc vào ngữ cảnh câu chứa nó. Ví dụ, từ "đá" trong ngữ cảnh "trái bóng đá" và "tảng đá" sẽ có vector khác nhau. Để biểu diễn cho cả câu, ta có thể lấy vector của token đặc biệt (như [CLS] trong BERT) hoặc trung bình các vector từ trong câu.

Đặc biệt, kỹ thuật Sentence Transformers được giới thiệu (Reimers & Gurevych, 2019) đã mở ra khả năng tạo vector cho cả câu hoặc đoạn văn một cách hiệu quả cho bài toán semantic similarity. Sentence Transformers thực chất là việc sử dụng mạng Transformer đã được fine-tune đặc biệt để sinh vector cho câu, sao cho những câu có nghĩa tương đồng sẽ gần nhau trong không gian vector. Có nhiều mô hình Sentence Transformer đã được huấn luyện sẵn (pre-trained) trên các tập dữ liệu lớn cho tác vụ Sentence Pair Regression (ví dụ STS – Semantic Textual Similarity) hoặc Multiple Negatives Ranking. Những mô hình này có sẵn trong thư viện sentence-transformers và thường được đặt tên theo kiến trúc và dữ liệu huấn luyện (ví dụ: intfloat/multilingual-e5-small, paraphrase-multilingual-MiniLM v.v.).

Đối với ngôn ngữ tiếng Việt, có một số mô hình sentence embedding do cộng đồng phát triển, ví dụ PhoBERT (một phiên bản BERT cho tiếng Việt) kết hợp với fine-tune STS, hay các mô hình đa ngôn ngữ như paraphrase-multilingual-MiniLM hoạt động khá tốt trên tiếng Việt.

Quy trình tạo embedding cho kho tri thức như sau: mỗi đoạn văn bản sau khi chunking sẽ được đưa qua mô hình Sentence Transformer để nhận được một vector d chiều (thông thường d = 768 đối với các mô hình dựa trên BERT). Vector này gọi là embedding vector của đoạn, mang ý nghĩa khái quát nội dung ngữ nghĩa của đoạn văn đó. Các vector này sẽ được lưu trữ vào cơ sở dữ liệu vector để phục vụ truy vấn sau này.

Ưu điểm lớn của việc dùng embedding theo câu là ta có thể thực hiện tìm kiếm ngữ nghĩa (semantic search): khi người dùng hỏi một câu, ta chỉ cần lấy embedding của câu hỏi và so sánh với embedding của các đoạn văn trong kho tri thức để tìm đoạn nào có nội dung "gần" nhất về mặt ý nghĩa. Điều này vượt trội so với tìm kiếm từ khóa truyền thống trong nhiều trường hợp, đặc biệt khi người dùng diễn đạt khác với cách viết trong tài liệu. Chẳng hạn, câu hỏi "liệt kê các ràng buộc toàn vẹn" có

thể khớp ngữ nghĩa với đoạn văn chứa "các loại ràng buộc toàn vẹn (integrity constraints) gồm: ..." dù từ ngữ không trùng khớp hoàn toàn.

Nói cách khác, vector embedding của câu văn là một đại diện toán học cho ngữ nghĩa. Trong không gian vector đó, phép đo độ tương đồng (thường dùng cosine similarity hoặc dot product) sẽ cho biết mức độ liên quan giữa câu hỏi và một đoạn tài liệu. Việc sử dụng embedding và semantic search giúp hệ thống tìm đúng thông tin cần thiết ngay cả khi cách dùng từ khác nhau.

Ví dụ minh họa: Nếu trong slide ghi "SQL cho phép định nghĩa ràng buộc toàn vẹn (integrity constraints) để đảm bảo tính đúng đắn của dữ liệu" và sinh viên hỏi "ràng buộc toàn vẹn là gì và để làm gì?", thì tìm kiếm từ khóa có thể trả về kết quả không đầy đủ, trong khi semantic search với embedding có khả năng nhận ra đoạn slide trên là phù hợp nhất về nội dung để trả lời câu hỏi.

Về mặt kỹ thuật, mô hình Sentence Transformer sẽ ánh xạ mỗi đoạn văn thành một vector trong không gian nhiều chiều (thường là 768 chiều đối với các mô hình BERT-base, nhưng có thể khác tùy phiên bản). Các vector này có thể hiểu nôm na như tọa độ trong không gian đó. Như Khan et al. (2024) [7] đã mô tả, các chunk văn bản sau khi tiền xử lý được biến đổi thành vector biểu diễn bằng cách sử dụng các mô hình embedding như BERT hoặc Sentence Transformers. Những vector này nắm bắt ý nghĩa ngữ nghĩa của đoạn văn, cho phép hệ thống thực hiện tìm kiếm tương tự. Các vector sẽ được lưu vào một Vector Store – là cơ sở dữ liệu được tối ưu cho việc truy vấn theo độ tương đồng của vector.

Tóm lại, việc lựa chọn và sử dụng mô hình embedding phù hợp sẽ quyết định trực tiếp đến chất lượng truy xuất thông tin. Do đó, trong quá trình thực hiện, tôi sẽ thử nghiệm một số mô hình Sentence Transformer... Các vector kết quả sau đó sẽ được lưu trữ trong một cơ sở dữ liệu vector để phục vụ cho việc tìm kiếm ngữ nghĩa hiệu quả, một kỹ thuật sẽ được trình bày chi tiết ở mục tiếp theo.

2.5 Truy xuất ngữ nghĩa với Cơ sở dữ liệu vector:

Sau khi các đoạn văn bản đã được chuyển thành vector embedding, bước tiếp theo trong kiến trúc RAG là truy xuất những đoạn văn có nội dung liên quan nhất đến câu hỏi của người dùng. Quá trình này được thực hiện thông qua Tìm kiếm ngữ nghĩa (Semantic Search), một kỹ thuật tiên tiến được hỗ trợ bởi Cơ sở dữ liệu vector (Vector Database).

2.5.1 Nguyên lý Tìm kiếm ngữ nghĩa:

Tìm kiếm ngữ nghĩa là kỹ thuật cho phép hệ thống hiểu được ý nghĩa thực sự của truy vấn, thay vì chỉ dựa vào sự trùng khớp từ khóa đơn thuần.

- Khác biệt với tìm kiếm từ khóa: Tìm kiếm từ khóa truyền thống sẽ thất bại nếu người dùng diễn đạt khác với văn bản gốc (ví dụ: tìm "học máy" sẽ không ra kết quả chứa "machine learning"). Ngược lại, tìm kiếm ngữ nghĩa vượt qua rào cản này. Nó sử dụng các vector embedding để biểu diễn cả câu hỏi và tài liệu trong một không gian ngữ nghĩa. Trong không gian đó, các văn bản có ý nghĩa tương tự sẽ nằm gần nhau.
- Cơ chế hoạt động: Khi người dùng đặt câu hỏi, hệ thống sẽ chuyển câu hỏi đó thành một vector và tìm kiếm các vector tài liệu "gần" nhất trong không gian (thường đo bằng độ tương đồng cosine). Kết quả là những đoạn văn liên quan nhất về mặt ý nghĩa, ngay cả khi từ ngữ không trùng khớp hoàn toàn.

Trong kiến trúc RAG, tìm kiếm ngữ nghĩa đóng vai trò là bộ truy xuất (Retriever), làm cầu nối quan trọng giữa câu hỏi của người dùng và kho tri thức, tạo nền tảng vững chắc cho các bước xử lý tiếp theo.

2.5.2 Cơ sở dữ liệu vector: Nền tảng cho Tìm kiếm ngữ nghĩa:

Để thực hiện tìm kiếm ngữ nghĩa trên hàng ngàn, thậm chí hàng triệu vector một cách hiệu quả, chúng ta cần một hệ thống chuyên dụng gọi là Cơ sở dữ liệu vector.

- Định nghĩa và vai trò: Cơ sở dữ liệu vector được thiết kế đặc biệt để lưu trữ, đánh chỉ mục (indexing) và truy vấn các vector trong không gian nhiều chiều với tốc độ cao. Trong hệ thống RAG, nó đóng vai trò là kho chứa toàn bộ các vector embedding của tài liệu, sẵn sàng cho việc tìm kiếm.
- Một số nền tảng phổ biến: Có nhiều giải pháp cho cơ sở dữ liệu vector như FAISS (thư viện của Facebook), Annoy (của Spotify), hoặc các hệ thống hoàn chỉnh hơn như Milvus, Pinecone và Qdrant.

2.5.3 Lựa chọn và triển khai Qdrant:

Đối với đề tài này, Qdrant được lựa chọn làm cơ sở dữ liệu vector vì những lý do sau:

- Hỗ trợ metadata & filtering: Cho phép đính kèm dữ liệu phụ (payload) vào mỗi vector (ví dụ: tên tài liệu, số trang) và kết hợp lọc theo các trường này trong lúc truy vấn. Đây là tính năng cực kỳ hữu ích để giới hạn phạm vi tìm kiếm.
- Dễ triển khai và tích hợp: Cung cấp phiên bản self-hosted qua Docker và các client API cho nhiều ngôn ngữ, bao gồm Python, giúp việc tích hợp vào backend FastAPI trở nên thuận tiện.
- Khả năng mở rộng: Hỗ trợ clustering để mở rộng khi quy mô dữ liệu tăng lên.

Quy trình hoạt động với Qdrant trong hệ thống:

1. Khởi tạo và Nạp dữ liệu (Indexing):

   + Một "collection" được tạo ra trong Qdrant để chứa các vector của môn học.

   + Mỗi đoạn văn bản sau khi được embedding sẽ được nạp vào collection dưới dạng một "point". Mỗi point bao gồm: một ID duy nhất, vector embedding, và phần payload chứa metadata (ví dụ: {"source": "giao_trinh.pdf", "page": 42}).

   + Qdrant sẽ tự động xây dựng chỉ mục (index) từ các point này.

2. Truy vấn (Searching):

   + Khi người dùng gửi câu hỏi, câu hỏi được chuyển thành một vector truy vấn.

   + Vector này được gửi đến Qdrant để thực hiện tìm kiếm top-K (ví dụ K=5) các point có vector gần nhất.

   + Qdrant trả về danh sách các point phù hợp nhất, kèm theo điểm số tương đồng và payload của chúng.

Những đoạn văn bản tương ứng với các payload được trả về này chính là "ngữ cảnh" được chọn lọc để đưa vào các bước xử lý tiếp theo như Reranking và Generation. Việc sử dụng tìm kiếm ngữ nghĩa trên Qdrant giúp đảm bảo chatbot cung cấp ngữ cảnh chính xác, liên quan, từ đó giảm thiểu hiện tượng "hallucination" của LLM và tăng độ tin cậy cho câu trả lời.

2.6 Kỹ thuật Reranking:

Sau khi thực hiện tìm kiếm ngữ nghĩa và thu được K đoạn văn bản liên quan ban đầu (như đã mô tả ở mục 2.5), kết quả này có thể vẫn chứa nhiễu. Ví dụ, có những đoạn chỉ liên quan một phần, hoặc chứa từ khóa nhưng không phải trọng tâm câu

hỏi. Nếu đưa tất cả K đoạn này vào prompt của LLM, mô hình có thể bị "quá tải" thông tin hoặc bị phân tán. Do đó, cần có thêm bước reranking (xếp hạng lại) để chọn lọc và sắp xếp các đoạn một cách tối ưu.

Reranking: Đây là giai đoạn tinh chỉnh sau truy xuất. Mục đích là sắp xếp lại các đoạn kết quả từ semantic search sao cho những đoạn thực sự phù hợp nhất với câu hỏi được đưa lên đầu và loại bớt các đoạn ít liên quan hoặc trùng lặp. Reranking có thể coi như một lớp lọc thứ hai, giúp thu hẹp tập ngữ cảnh sẽ đưa vào LLM. Reranking giữ vai trò kép: vừa làm tăng chất lượng ngữ cảnh (enhancer) vừa như bộ lọc để giảm số lượng dữ liệu đưa vào LLM. [4]

Có hai hướng tiếp cận reranking chính:

- Dựa trên luật (rule-based): sử dụng các tiêu chí cố định như độ đa dạng (diversity), độ phủ từ khóa, hay các chỉ số truy xuất cổ điển (MRR, TF-IDF) để sắp xếp. Cách này đơn giản nhưng thường không tối ưu về ngữ nghĩa.
- Dựa trên mô hình học (learning-based): sử dụng một mô hình học sâu (thường là cross-encoder dựa trên Transformer) để đánh giá cặp câu hỏi – đoạn văn và cho điểm mức độ phù hợp. Cross-encoder nghĩa là mô hình sẽ nhận đồng thời cả câu hỏi và một đoạn, sau đó output ra một điểm số hoặc xác suất đoạn đó trả lời được câu hỏi. Vì mô hình xem xét toàn bộ câu hỏi và đoạn cùng nhau (thay vì tách riêng như bi-encoder), nó thường cho kết quả đánh giá chính xác hơn. Các mô hình cross-encoder phổ biến cho rerank gồm: BERT đã fine-tune trên tập dữ liệu Q&A (như MS MARCO), hoặc các model chuyên biệt như Cohere Rerank, OpenAI GPT-3 (sử dụng GPT-3 để chấm điểm).

Đề tài hiện tại hướng đến việc sử dụng mô hình reranker dựa trên cross-encoder. Cụ thể có thể dùng một mô hình đã huấn luyện sẵn cho tác vụ đánh giá độ liên quan văn bản (ví dụ: cross-encoder/ms-marco-MiniLM-L-6-v2 từ HuggingFace, kích thước nhỏ ~6 layers nhưng hiệu quả khá tốt trên dữ liệu tìm kiếm). Mô hình này khi input cặp (câu hỏi, đoạn văn) sẽ đưa ra một điểm từ 0-1; ta dùng điểm đó để sắp xếp danh sách K đoạn. Sau đó có thể chọn top N đoạn (ví dụ N = 3) có điểm cao nhất để đưa vào prompt của LLM, thay vì dùng cả K đoạn ban đầu. Nghiên cứu đã chỉ ra rằng việc giới hạn ngữ cảnh vào những đoạn liên quan nhất giúp LLM tập trung hơn và giảm nhiễu, tránh việc cung cấp quá nhiều thông tin dư thừa có thể dẫn đến trả lời lan man hoặc sai trọng tâm.

Tóm lại, trong hệ thống RAG của tôi:

- Sẽ sử dụng semantic vector search trên Qdrant để nhanh chóng lọc ra các ứng viên đoạn văn liên quan đến câu hỏi.
- Sau đó áp dụng rerank bằng cross-encoder để chọn lọc những đoạn phù hợp nhất, đảm bảo ngữ cảnh đưa vào LLM là tinh gọn và sát chủ đề.
- Các đoạn ngữ cảnh được chọn sẽ được sắp xếp hợp lý (ví dụ theo thứ tự mức độ liên quan hoặc ghép nối theo cấu trúc logic nếu chúng thuộc cùng một tài liệu) trong prompt để LLM sử dụng. Việc sắp xếp này cũng quan trọng, vì LLM thường chú ý phần đầu prompt nhiều hơn (vấn đề "lost in the middle" – thông tin giữa prompt dễ bị quên). Do đó, đoạn quan trọng nên đặt lên đầu prompt.

Nhờ kết hợp semantic search và reranking, hệ thống có thể đạt được sự cân bằng: vừa bao quát tìm kiếm được những thông tin liên quan, vừa chính xác chọn ra thông tin tốt nhất để trả lời. Đây là chìa khóa để chatbot trả lời đúng và cụ thể, tránh lan man hoặc sai nguồn. Các nghiên cứu trong lĩnh vực RAG nhấn mạnh rằng sự kết hợp này giúp giảm thiểu đáng kể hiện tượng hallucination và tăng tính tin cậy của câu trả lời, vì mô hình hầu như chỉ sử dụng những thông tin đã được truy xuất và kiểm chứng thay vì "tự suy diễn".

2.7 Mô hình ngôn ngữ lớn (LLM) – Kiến trúc Transformer và ví dụ mô hình Gemini:

Mô hình ngôn ngữ lớn (Large Language Model – LLM) là các mô hình học sâu với hàng tỷ tham số, được huấn luyện trên lượng dữ liệu ngôn ngữ khổng lồ, có khả năng sinh ra văn bản mạch lạc và thực hiện nhiều nhiệm vụ NLP. Nền tảng của hầu hết các LLM hiện đại là kiến trúc Transformer (Vaswani et al., 2017). Transformer sử dụng cơ chế self-attention cho phép mô hình học được các mối quan hệ trong chuỗi từ, nhờ đó vượt trội so với RNN/LSTM trước đây trong việc xử lý ngôn ngữ dài và ngữ cảnh phức tạp.

Transformer có hai dạng chính: encoder-decoder (như BART, T5) và decoder-only (như GPT, GPT-2/3, các model dạng ChatGPT/GPT-4, v.v.). Các LLM nổi tiếng (GPT-3, GPT-4, PaLM, LLaMA…) thường thuộc loại decoder-only, được huấn luyện để dự đoán từ tiếp theo (next-token prediction) trên lượng dữ liệu rất lớn (gồm sách, bài viết, mã nguồn, v.v.). Kết quả, các LLM này có được khả năng sinh văn bản trôi chảy và "hiểu" được nhiều kiến thức ẩn trong dữ liệu huấn luyện.

Tuy nhiên, do giới hạn huấn luyện, LLM kiểu GPT thường có "kiến thức đóng băng" ở thời điểm huấn luyện và như đã đề cập, có thể tạo ra thông tin sai (hallucination) khi gặp câu hỏi ngoài kiến thức. Vì vậy ta mới cần kết hợp với truy xuất (RAG) để bổ sung kiến thức mới và kiểm chứng.

Trong đề tài này, dự kiến sử dụng API của mô hình Google Gemini – một trong những LLM mới và mạnh mẽ vào năm 2024-2025. Gemini là mô hình do Google DeepMind phát triển, được công bố là mô hình đa phương thức, có thể xử lý đồng thời nhiều dạng dữ liệu (văn bản, hình ảnh, mã code, v.v.) và được tối ưu cho nhiều quy mô triển khai. Gemini được xem là thế hệ kế tiếp của PaLM và được tích hợp vào các sản phẩm của Google như chatbot Bard. Mặc dù chi tiết kiến trúc không được công bố đầy đủ, người ta biết rằng Gemini cũng dựa trên Transformer và được huấn luyện với sự kết hợp tinh hoa từ Google Brain và DeepMind, nhằm cạnh tranh với GPT-4. Điểm đặc biệt là Gemini được kỳ vọng vượt trội GPT-4 ở khả năng lập kế hoạch và suy luận nhờ tích hợp một số kỹ thuật từ mô hình chơi cờ vây AlphaGo của DeepMind. Ngoài ra, Gemini hỗ trợ đa modal, nghĩa là ngoài văn bản nó có thể nhận hiểu thêm hình ảnh, âm thanh, mặc dù trong đề tài này ta chỉ sử dụng cho văn bản.

Việc sử dụng dịch vụ API (thay vì chạy trực tiếp mô hình) có ưu điểm là tiết kiệm tài nguyên – tôi không cần một máy tính có GPU mạnh để chạy mô hình hàng chục tỷ tham số, thay vào đó gửi yêu cầu đến máy chủ của nhà cung cấp (Google) và nhận về kết quả. Tất nhiên, việc này đòi hỏi kết nối mạng và có thể phát sinh độ trễ do gọi dịch vụ. Tuy nhiên, do số lượng người dùng không quá lớn (giới hạn trong phạm vi sinh viên môn học) nên việc dùng API là giải pháp hợp lý.

Với LLM (dù là Gemini hay mô hình tương đương), ta sẽ sử dụng nó ở bước Generation của pipeline RAG: sau khi đã có câu hỏi người dùng và các đoạn ngữ cảnh liên quan (từ kho tri thức), ta sẽ xây dựng một prompt đầy đủ rồi gọi API để mô hình sinh ra câu trả lời. Mô hình LLM có nhiệm vụ hiểu câu hỏi và tận dụng thông tin trong ngữ cảnh cung cấp để tạo câu trả lời mạch lạc, có tính kết nối với ngữ cảnh đó (grounded answer). Một thách thức là làm sao hướng dẫn mô hình trích dẫn nguồn trong câu trả lời. Điều này thường được thực hiện bằng cách định dạng ngữ cảnh kèm tên nguồn và thêm hướng dẫn trong prompt như "Hãy trả lời dựa trên

thông tin dưới đây và trích dẫn nguồn (ví dụ: [Tài liệu A]) cho mỗi thông tin bạn sử dụng."

Cuối cùng, cũng cần nói thêm rằng hiện có nhiều LLM mã nguồn mở có thể chạy cục bộ (như LLaMA-2, Bloom, GPT-J, v.v.), tuy hiệu suất có thể chưa bằng các mô hình thương mại hàng đầu nhưng ưu điểm là chủ động và không cần gọi dịch vụ. Trong phạm vi đồ án này, tôi ưu tiên độ chính xác và khả năng ngôn ngữ, do đó chấp nhận sử dụng mô hình API mạnh (Gemini) để đảm bảo chatbot có chất lượng trả lời tốt nhất. Nếu về sau có điều kiện, có thể thử nghiệm các LLM tiếng Việt mã nguồn mở (như VinAI Phoenix, VNU CocoGPT, hoặc các model trong nghiên cứu của Nguyễn Quang Đức et al. 2024) để so sánh.

Nghiên cứu của Nguyễn Quang Đức et al. (2024) giới thiệu ViRAG-7B, mô hình RAG tiếng Việt thang 7 tỷ tham số, đạt EM 46,2 % trên bộ Vietnamese Open-QA và mở nguồn weights; đây là cơ sở tham khảo quan trọng cho việc tinh chỉnh LLM tiếng Việt sau này. [3]

2.8 Tổng quan môn Cơ sở dữ liệu và các công nghệ sử dụng:

Môn Cơ sở dữ liệu (Database Systems) là một môn học nền tảng trong ngành CNTT, cung cấp cho sinh viên kiến thức về cách tổ chức, lưu trữ và quản lý dữ liệu một cách có cấu trúc. Nội dung môn học thường bao gồm:

- Các khái niệm cơ bản về cơ sở dữ liệu: cơ sở dữ liệu là gì, các đặc tính của hệ quản trị CSDL (DBMS), kiến trúc của DBMS.
- Mô hình thực thể – liên kết (ER model): cách thiết kế cơ sở dữ liệu ở mức khái niệm, với các thực thể, thuộc tính, quan hệ và biểu diễn bằng sơ đồ ER.
- Mô hình quan hệ: chuyển đổi từ mô hình ER sang mô hình quan hệ (bảng), khái niệm về lược đồ quan hệ, bộ, thuộc tính, khóa chính, khóa ngoại, các ràng buộc toàn vẹn.
- Đại số quan hệ và phép toán: các phép toán tập hợp (hợp, giao, trừ) và phép toán đặc thù (chiếu, chọn, kết nối, phân chia, v.v.) trên quan hệ.
- Ngôn ngữ truy vấn SQL: cú pháp và cách sử dụng SQL để định nghĩa dữ liệu (DDL) và thao tác dữ liệu (DML). Bao gồm các lệnh tạo bảng, thêm/xóa/sửa, truy vấn SELECT (với WHERE, GROUP BY, HAVING, JOIN, lồng truy vấn, v.v.).

- Thiết kế cơ sở dữ liệu: các vấn đề về chuẩn hóa (normalization), các dạng chuẩn (1NF, 2NF, 3NF, BCNF) và cách tách bảng để loại bỏ dư thừa, tránh anomaly.
- Quản trị cơ sở dữ liệu: một số khía cạnh như chỉ mục (index), tối ưu truy vấn, giao dịch (transaction), bảo mật dữ liệu (cấp quyền), v.v. (Tùy phạm vi môn học cơ bản, một số trường có thể chỉ giới hạn nhẹ phần này).

Như vậy, kiến thức môn CSDL khá rộng từ lý thuyết đến thực hành. Bộ tài liệu cho chatbot dự kiến sẽ bao gồm giáo trình chính (ví dụ sách "Database System Concepts" hoặc tương đương giáo trình tiếng Việt), slide bài giảng tóm tắt theo từng chương, tài liệu bài tập và lời giải và có thể cả các ví dụ SQL mẫu. Do đặc thù môn học, tài liệu có thể chứa các thành phần phi văn bản như sơ đồ ER hay mẫu bảng. Trong phiên bản hiện tại, đề tài sẽ tập trung vào nội dung văn bản, nên các phần như sơ đồ sẽ được mô tả lại bằng chú thích (nếu có) trong phần text, hoặc có thể bỏ qua nếu không thể parse được.

Về công nghệ triển khai hệ thống, đề tài dự kiến sử dụng các công cụ và nền tảng sau:

- Ngôn ngữ lập trình Python cho phần backend: Python có hệ sinh thái phong phú cho NLP (ví dụ thư viện transformers, sentence-transformers để tạo embedding, thư viện cho gọi API LLM, v.v.). Python cũng được ưa chuộng để xây dựng nhanh nguyên mẫu và có nhiều thư viện hỗ trợ tích hợp với Qdrant, Supabase.
- Framework FastAPI cho backend: FastAPI là một framework web nhẹ và nhanh trong Python để xây dựng các API RESTful. Nó cho phép dễ dàng định nghĩa các endpoint (như /ask, /upload, ...) và quản lý các request/response JSON. FastAPI cũng hỗ trợ async, phù hợp khi cần gọi API ngoài (như API của Gemini) để không chặn luồng.
- Thư viện NLP và xử lý tài liệu: Sử dụng PyMuPDF hoặc pdfplumber để trích xuất văn bản từ file PDF (giáo trình, slide). Sử dụng python-docx cho file Word, hoặc đơn giản dùng các công cụ chuyển đổi sẵn có. Có thể dùng regex hoặc BeautifulSoup nếu cần xử lý HTML (trong trường hợp tài liệu ở định dạng HTML). Đối với SQL files, có thể đọc thô nội dung hoặc sử dụng một parser để định dạng. Ngoài ra, thư viện vncorenlp hoặc UnderTheSea có thể dùng để tách từ tiếng Việt nếu cần trong giai đoạn nhất định (dù với embedding BERT thì thường không cần tách từ trước).

- Sentence Transformers library: Sử dụng thư viện sentence-transformers của Python để tải mô hình embedding và tạo vector. Ví dụ: from sentence_transformers import SentenceTransformer.
- Qdrant: Sử dụng Qdrant như đã phân tích. Triển khai Qdrant có thể thông qua Docker container. Tương tác với Qdrant trong Python thông qua thư viện qdrant-client (cung cấp các hàm như create_collection, upsert, search).
- Google Gemini API: Trong code Python, việc gọi có thể thông qua REST (dùng thư viện requests) hoặc SDK nếu Google cung cấp. Định dạng đầu vào thường là một chuỗi prompt, đầu ra là chuỗi hoàn thành.
- Supabase: Đây là một nền tảng backend-as-a-service dựa trên PostgreSQL, cung cấp khả năng lưu trữ dữ liệu, xác thực người dùng và dễ kết nối qua API. Trong dự án này, Supabase có thể được dùng để lưu lịch sử hội thoại. Mỗi bản ghi hội thoại có thể gồm: nội dung câu hỏi, nội dung câu trả lời, thời gian, user (nếu có đăng nhập) và các metadata như nguồn trích dẫn (có thể lưu riêng hoặc gộp chung). Supabase cho phép thao tác thông qua REST API hoặc qua thư viện supabase-py. Việc sử dụng Supabase giúp tránh phải tự cài đặt một database riêng cho lịch sử và dễ dàng triển khai đồng bộ cùng frontend (Supabase cũng có thư viện cho JavaScript).
- Frontend :

  + Next.js: Next.js là một framework dựa trên React, được phát triển bởi Vercel, hỗ trợ xây dựng các ứng dụng web full-stack hiệu quả. Nó cung cấp các tính năng như Server-Side Rendering (SSR), Static Site Generation (SSG) và Incremental Static Regeneration (ISR) nhằm tối ưu hiệu suất và khả năng SEO. Next.js sử dụng hệ thống routing dựa theo cấu trúc thư mục và hỗ trợ tự động phân chia mã nguồn để tăng tốc độ tải trang. Ngoài ra, framework này còn tích hợp sẵn TypeScript, hỗ trợ nhiều phương thức định dạng CSS và sử dụng bộ biên dịch SWC hiện đại.

  + React: React là một thư viện JavaScript mã nguồn mở do Meta (Facebook) phát triển, dùng để xây dựng giao diện người dùng theo mô hình component. React sử dụng Virtual DOM để cập nhật giao diện một cách hiệu quả, chỉ render lại những phần thực sự thay đổi. Thư viện này hỗ trợ JSX – một cú pháp mở rộng giúp kết hợp JavaScript với HTML, và cung cấp các hook hoặc class để xử lý logic giao

diện. React thường được dùng để xây dựng các ứng dụng web đơn trang (SPA) với khả năng tái sử dụng cao và dễ bảo trì.

+ TypeScript: TypeScript là một siêu tập hợp (superset) của JavaScript do Microsoft phát triển, cho phép thêm kiểu dữ liệu tĩnh vào mã nguồn. TypeScript cung cấp các tính năng như type inference, interface, enum, và generic giúp phát hiện lỗi trong quá trình biên dịch, trước khi chạy chương trình. Việc sử dụng TypeScript giúp tăng độ an toàn, dễ đọc và dễ bảo trì cho các dự án lớn. TypeScript được biên dịch thành JavaScript để chạy trên trình duyệt hoặc môi trường Node.js.

+ Tailwind CSS: Tailwind CSS là một framework CSS theo hướng "utility-first", cho phép lập trình viên xây dựng giao diện trực tiếp trong HTML thông qua các lớp tiện ích nhỏ như flex, text-center, mt-4,... Thay vì viết CSS riêng cho từng thành phần, Tailwind giúp tăng tốc quá trình phát triển và dễ dàng kiểm soát bố cục, màu sắc, khoảng cách, v.v. Framework này cũng hỗ trợ responsive design, trạng thái (hover, focus), dark mode, và có thể cấu hình linh hoạt qua tệp tailwind.config.js. Từ phiên bản 3 trở đi, Tailwind sử dụng chế độ biên dịch Just-in-Time (JIT) giúp tối ưu hiệu suất và giảm kích thước tệp CSS.

CHƯƠNG 3 PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1 Phân tích yêu cầu hệ thống:

Trước khi thiết kế, cần phân tích rõ những yêu cầu đặt ra đối với hệ thống chatbot RAG hỗ trợ môn CSDL. Yêu cầu được chia thành hai loại: yêu cầu chức năng (các tính năng mà hệ thống phải có) và yêu cầu phi chức năng (các tiêu chí về hiệu suất, trải nghiệm, giới hạn kỹ thuật).

3.1.1 Yêu cầu chức năng:

Hệ thống cần đáp ứng các chức năng chính sau:

- Hỏi đáp kiến thức CSDL: Cho phép người dùng (sinh viên) nhập vào một câu hỏi liên quan đến môn CSDL và nhận lại câu trả lời từ chatbot. Câu trả lời cần rõ ràng, chính xác theo kiến thức tài liệu và bằng tiếng Việt.
- Trích dẫn nguồn tài liệu: Mỗi câu trả lời của chatbot phải kèm theo trích dẫn nguồn của thông tin được sử dụng để trả lời. Cụ thể, chatbot có thể chèn chú thích [Tài liệu X, trang Y] hoặc một định dạng phù hợp để người dùng biết câu trả lời được lấy từ đâu. Việc trích dẫn nguồn giúp tăng độ tin cậy và cho phép người dùng tra cứu thêm nếu cần.
- Quản lý tài liệu kiến thức (Admin): Hệ thống áp dụng mô hình quản lý tri thức tập trung. Chỉ người quản trị (Admin) mới có quyền tải lên, cập nhật hoặc xóa các tài liệu trong kho tri thức. Toàn bộ tài liệu này sẽ được xử lý và đưa vào một collection chung duy nhất trên Qdrant, phục vụ cho tất cả người dùng. Cách tiếp cận này đảm bảo nguồn kiến thức là đồng nhất, đã qua kiểm duyệt và dễ dàng bảo trì, khác với mô hình phân tán nơi mỗi người dùng tự quản lý tài liệu của riêng mình.
- Lưu lịch sử hội thoại: Mỗi người dùng khi tương tác với chatbot sẽ có một phiên hội thoại. Hệ thống nên lưu lại lịch sử các câu hỏi và câu trả lời trong phiên đó. Điều này giúp người dùng có thể xem lại các câu hỏi đã hỏi trước đó (tránh hỏi lặp hoặc để ôn tập). Lưu lịch sử gắn với tài khoản người dùng. Lịch sử cũng hữu ích cho quản trị viên phân tích mức độ sử dụng và cải thiện hệ thống.
- Gợi ý câu hỏi: Để hỗ trợ người học, chatbot có thể cung cấp một số gợi ý câu hỏi mẫu hoặc câu hỏi liên quan. Chẳng hạn, sau khi trả lời một câu hỏi, hệ thống có thể hiện ra "Người dùng thường hỏi: ..." với vài gợi ý.

Hoặc trên giao diện có mục "Câu hỏi thường gặp" để sinh viên tham khảo. Những gợi ý này được LLM tạo ra dựa trên lịch sử hội thoại của người dùng, mục đích của việc này câu hỏi gợi ý sẽ bám sát và cá nhân hóa cho từng người dùng.

- Phân tích SQL: Một tính năng nâng cao là cho phép chatbot nhận đầu vào là một truy vấn SQL hoặc một đoạn mã SQL, sau đó phân tích hoặc giải thích nó cho người dùng. Ví dụ, nếu người dùng nhập một câu lệnh SQL, chatbot có thể diễn giải câu lệnh đó làm gì, có đúng cú pháp không, nếu sai thì lỗi ở đâu. Hoặc chatbot có thể hỗ trợ giải bài tập SQL: người dùng cho một câu hỏi SQL, chatbot hướng dẫn cách tư duy và viết truy vấn bằng việc tận dụng khả năng của LLM.

3.1.2 Yêu cầu phi chức năng:

Bên cạnh các chức năng, hệ thống cần thỏa mãn các tiêu chí phi chức năng sau:

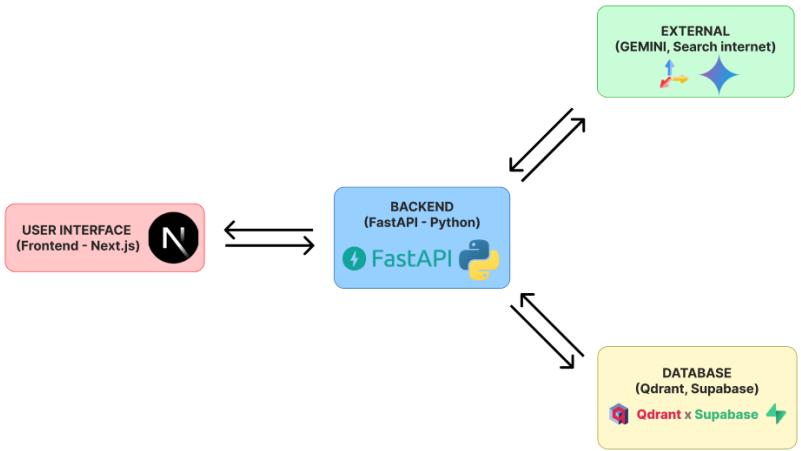
- Hiệu năng (Performance): Chatbot cần có tốc độ phản hồi tương đối nhanh để đảm bảo trải nghiệm người dùng. Mục tiêu là thời gian trả lời mỗi câu hỏi nên trong khoảng vài giây (lý tưởng 3-5 giây, tối đa ~10 giây với những truy vấn phức tạp). Để đạt được, các thành phần pipeline phải được tối ưu: ví dụ, việc tìm kiếm vector trên Qdrant cần < 1 giây, gọi API LLM nếu mất 2-3 giây, cộng thêm xử lý khác.
- Tính chính xác và tin cậy: Đây là yêu cầu quan trọng. Hệ thống phải cung cấp câu trả lời đúng với kiến thức môn học và có độ tin cậy cao. Việc trích dẫn nguồn đã hỗ trợ tính tin cậy. Ngoài ra, câu trả lời cần tránh tối đa các thông tin sai lệch. LLM đôi khi có thể "bịa" nếu ngữ cảnh chưa đủ, nên hệ thống phải cố gắng cung cấp ngữ cảnh đầy đủ và nếu không tự tin thì tốt nhất nên từ chối hoặc xin thêm thông tin hơn là trả lời sai.
- Tính thân thiện, dễ sử dụng: Giao diện người dùng cần trực quan, đơn giản. Sinh viên có thể sử dụng mà không cần hướng dẫn phức tạp. Ví dụ, giao diện chat nên giống các ứng dụng chat phổ biến (tin nhắn của người dùng căn phải, tin của bot căn trái, khác màu). Nút chức năng nên rõ ràng (ví dụ nút gửi câu hỏi).
- Khả năng mở rộng (Scalability): Mặc dù trước mắt hệ thống phục vụ cho một lớp học hoặc một nhóm sinh viên, nhưng về lâu dài có thể mở rộng

cho nhiều người dùng hoặc nhiều môn học. Thiết kế hệ thống nên theo hướng module hóa, để sau này có thể tăng cường sức mạnh. Việc lựa chọn công nghệ cloud như Supabase hay Qdrant cloud cũng giúp dễ nâng cấp tài nguyên.

- Bảo mật và riêng tư: Hệ thống cần đảm bảo an toàn cho dữ liệu. Dữ liệu ở đây gồm: tài liệu môn học (có thể không quá nhạy cảm vì thường là nội dung học thuật, nhưng vẫn nên bảo quản tránh rò rỉ), lịch sử câu hỏi (liên quan đến người dùng). Nếu có đăng nhập, cần quản lý phiên an toàn, lưu mật khẩu (nếu có) dưới dạng mã hóa. Khi gọi các API dịch vụ (như LLM), tránh gửi những thông tin nhạy cảm không cần thiết. Ở phạm vi môn học, vấn đề bảo mật không quá nghiêm trọng, nhưng cũng không nên bỏ qua.
- Dễ bảo trì: Code của hệ thống cần rõ ràng, tài liệu đầy đủ để người khác có thể tiếp tục phát triển. Cấu trúc dự án phải được tổ chức khoa học (ví dụ tách phần xử lý tài liệu, phần RAG core, phần API route riêng). Cũng nên có log ghi lại các hoạt động quan trọng để dễ debug khi có sự cố.

3.2 Thiết kế kiến trúc tổng thể:

Dựa trên các yêu cầu đã phân tích, kiến trúc tổng thể của hệ thống chatbot RAG hỗ trợ môn CSDL sẽ gồm 4 thành phần chính: Frontend, Backend, Database (bao gồm cả kho tri thức vector và cơ sở dữ liệu quan hệ cho siêu dữ liệu và lịch sử hội thoại)và các dịch vụ bên ngoài như Gemini (thông qua api), tool google search với Tavily (công cụ tìm kiếm web). Dưới đây là sơ đồ tổng hệ kiến trúc và các mô tả chi tiết các thành phần và luồng tương tác giữa chúng:



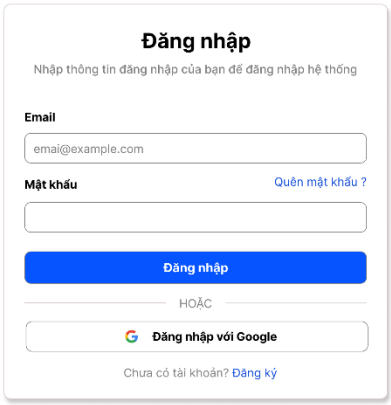
SƠ ĐỒ 3.1: Thiết kế sơ đồ kiến trúc tổng thể hệ thống

(Nguồn: Tự xây dựng)

3.2.1 Frontend:

Frontend được xây dựng bằng Next.js (một framework React-based, phát triển bởi Vercel, giúp xây dựng ứng dụng web với khả năng Server-Side Rendering (SSR) và Static Site Generation (SSG), cùng nhiều tính năng cao cấp như Incremental Static Regeneration (ISR)), cung cấp giao diện người dùng cho sinh viên để tương tác với hệ thống.

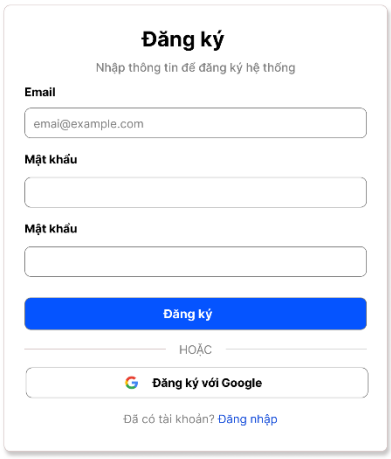
- Đối với người dùng:
  - Giao diện đăng nhập: Cho phép người dùng có thể đăng nhập vào hệ thống khi đã có tài khoản.



HÌNH 3.1 Thiết kế giao diện trang đăng nhập

(Nguồn: Tự xây dựng)

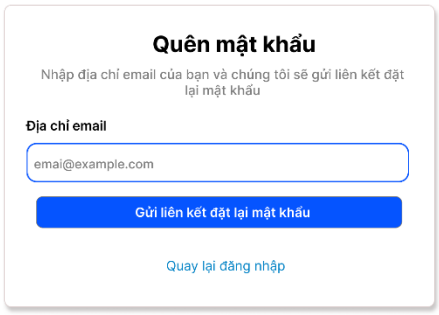
  - Giao diện đăng ký: Cho phép người dùng có thể đăng ký tài khoản hệ thốngkhi họ chưa có tài khoản.



HÌNH 3.2: Thiết kế giao diện trang đăng ký

(Nguồn: Tự xây dựng)

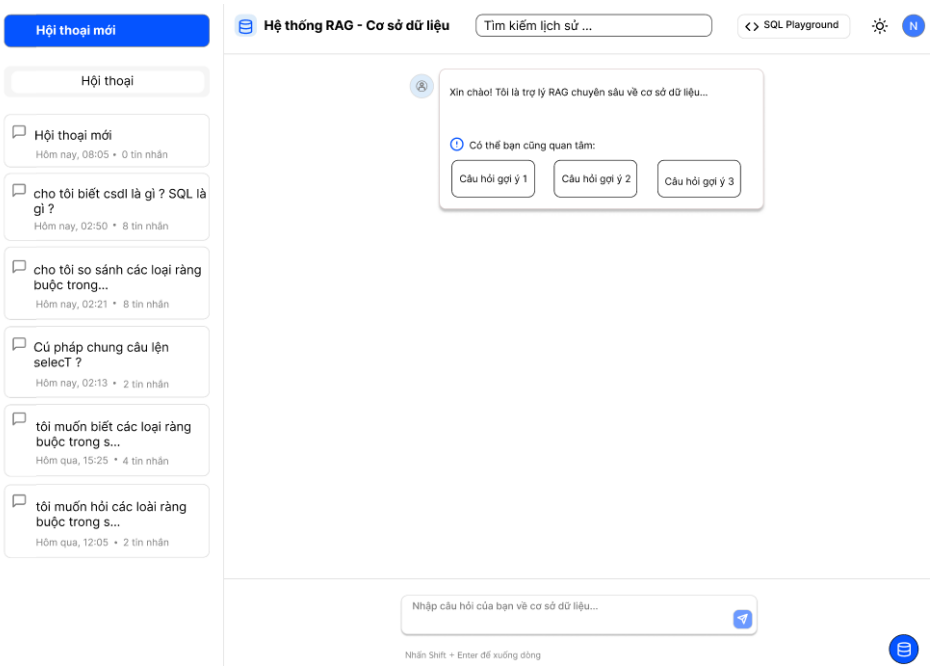
- Giao diện quên mật khẩu: Cho phép người dùng có thể đặt lại mật khẩu của mình khi quên mật khẩu.



HÌNH 3.3: Thiết kế giao diện trang quên mật khẩu

(Nguồn: Tự xây dựng)

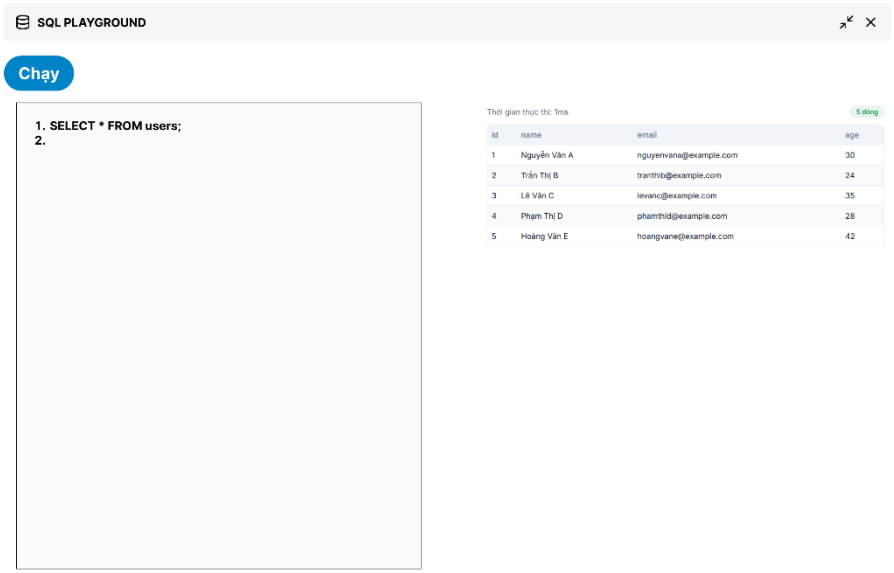
- Giao diện chính là màn hình chat, bên trái là sidebar hiển thị lịch sử chat của sinh viên. Bên dưới có ô nhập câu hỏi, sinh viên có thể nhập vào và nhận câu trả lời từ chatbot.



HÌNH 3.4: Thiết kế giao diện trang chính

(Nguồn: Tự xây dựng)

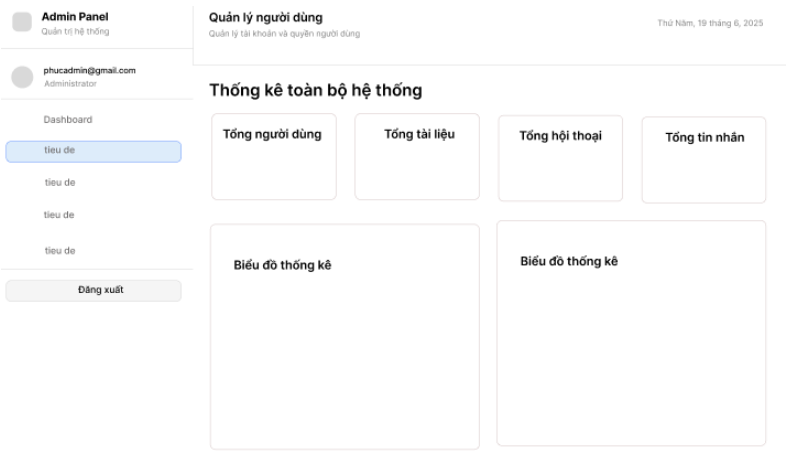
- Giao diện SQL Playground: Để tăng cường khả năng thực hành cho sinh viên, hệ thống cung cấp một trang "SQL Playground". Tại đây, người dùng có thể viết và thực thi các câu lệnh SQL trực tiếp trên trình duyệt với các bộ dữ liệu mẫu được định nghĩa sẵn (ví dụ: bảng SinhVien, MonHoc). Tính năng này được xây dựng bằng thư viện sql.js, một thư viện cho phép chạy một CSDL SQLite hoàn toàn trên client-side, giúp sinh viên kiểm tra nhanh các truy vấn mà không cần cài đặt môi trường CSDL phức tạp.



HÌNH 3.5: Thiết kế giao diện trang SQL Playground

(Nguồn: Tự xây dựng)

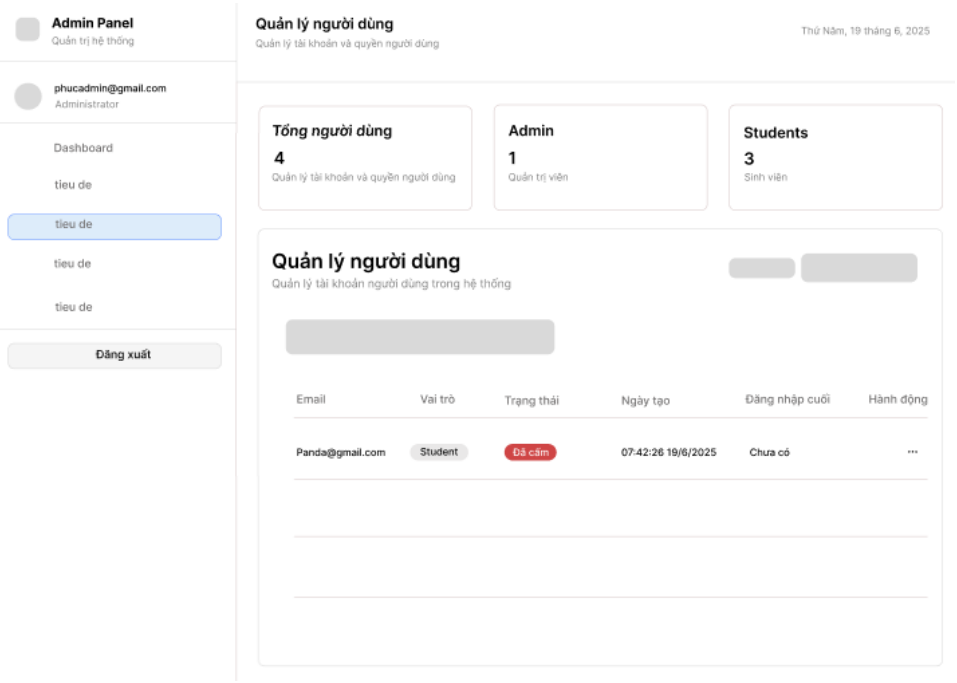
- Đối với người dùng:
  - Trang dashboard chính của admin: Hiển thị các thống kê một số thông tin như người, tài liệu, hội thoại,...



HÌNH 3.6: Thiết kế trang dashboard chính của admin

(Nguồn: Tự xây dựng)

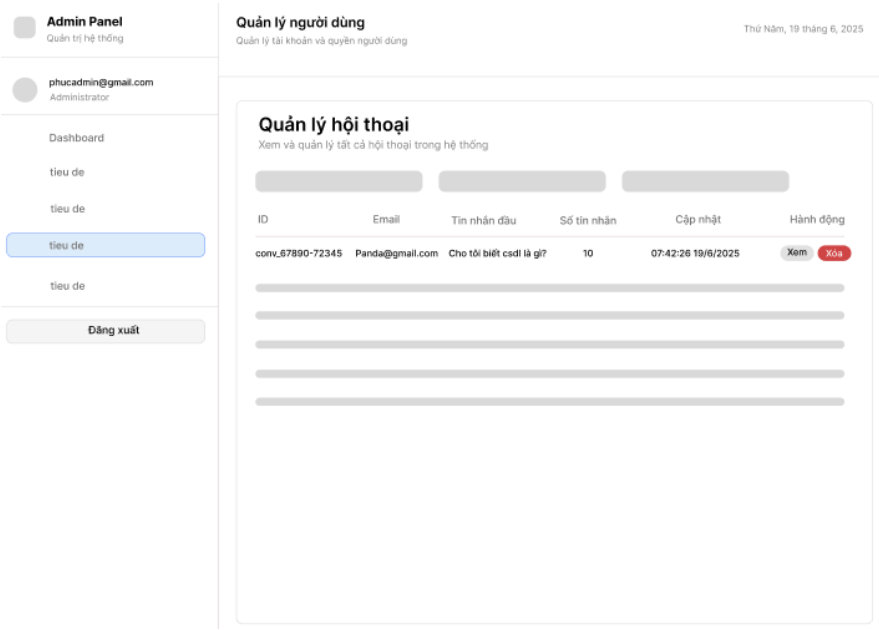
  - Trang quản lý người dùng: Đây là giao diện mà admin có thể thực hiện quản lý những người dùng đang sử dụng hệ thống.



HÌNH 3.7: Thiết kế trang quản lý người dùng của admin

(Nguồn: Tự xây dựng)

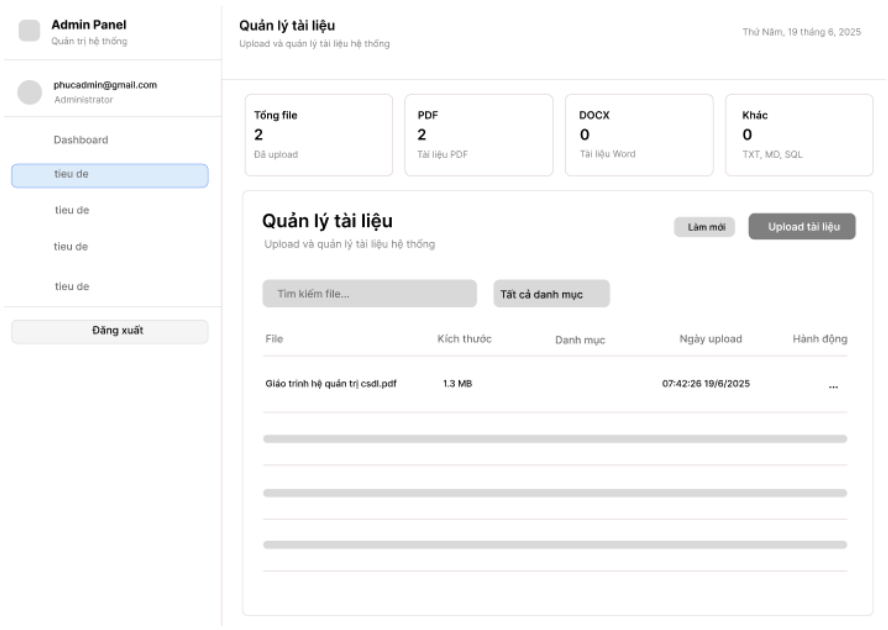
- Trang quản lý hội thoại: Cho phép admin có thể quản lý lịch sử các cuộc hôi thoại giữa người dùng và chatbot.



HÌNH 3.8: Trang quản lý lịch sử hội thoại của admin

(Nguồn: Tự xây dựng)

- Trang quản lý tài liệu: Đây là giao diện đặc quyền dành cho Admin, cho phép thực hiện các thao tác quản lý kho tri thức như tải lên tài liệu mới, xem danh sách và xóa các tài liệu không còn phù hợp.



HÌNH 3.9: Thiết kế trang quản lý tài liệu của admin

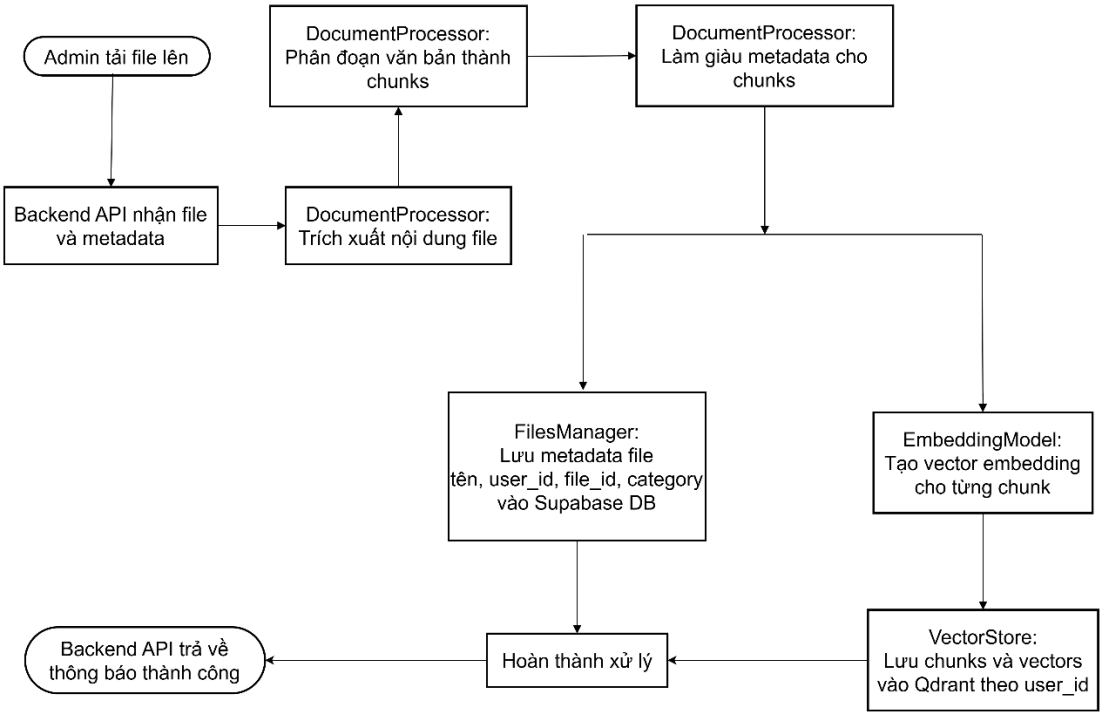
(Nguồn: Tự xây dựng)

- Tương tác với Backend:
    - Frontend giao tiếp với Backend thông qua các API endpoint RESTful, sử dụng giao thức HTTP/HTTPS. Việc quản lý các lời gọi API được thực hiện tập trung trong thư mục lib/api.ts.
    - Ví dụ về các tương tác chính:
        - Khi người dùng gửi câu hỏi, frontend gọi API POST /api/ask/stream.
        - Khi người dùng tải lên tệp tài liệu, frontend gọi API POST /api/upload để gửi tệp lên máy chủ.
        - Để lấy danh sách các tệp đã tải lên, frontend gọi API GET /api/files
        - Để nhận các câu hỏi gợi ý, frontend gọi API GET /api/suggestions

3.2.2 Backend:

Backend được xây dựng bằng Python với framework FastAPI, chịu trách nhiệm xử lý toàn bộ logic nghiệp vụ của hệ thống chatbot. Backend được cấu thành từ các module chức năng chính:

- Module Xử lý tài liệu (Data Ingestion):
    - Chức năng: Tiếp nhận tài liệu thô (PDF, DOCX, TXT,...) từ duy nhất người quản trị (admin) thông qua API POST /api/upload.
    - Quy trình xử lý theo luồng hoạt động sau:



SƠ ĐỒ 3.2: Sơ đồ hoạt động - Luồng Xử lý Tài liệu

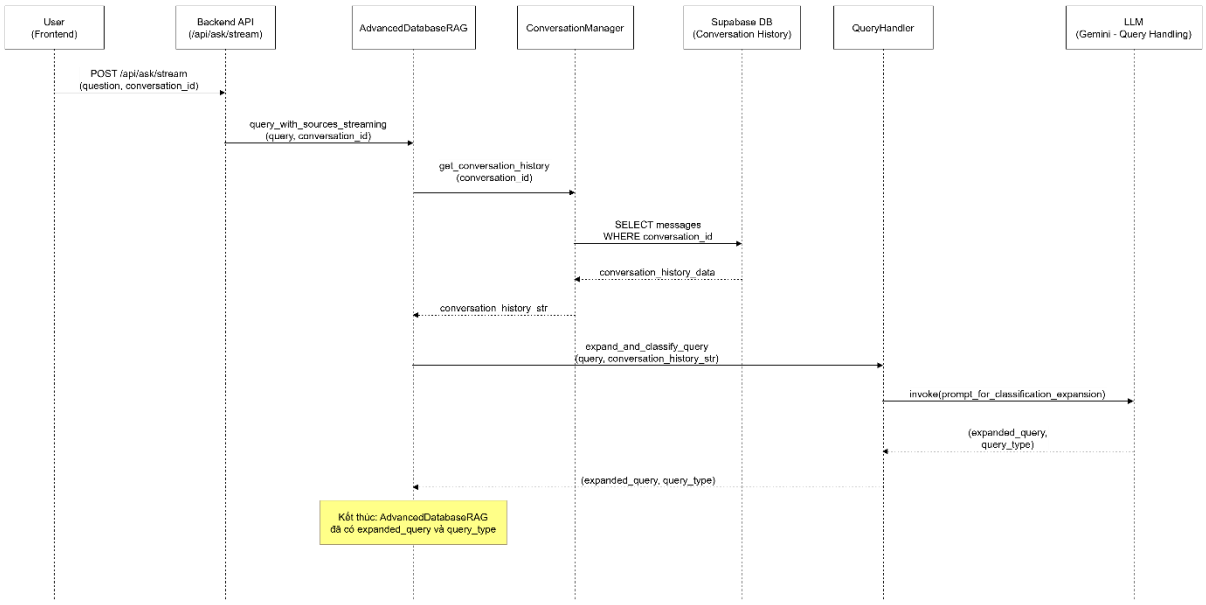
(Nguồn: tự xây dựng dựa trên thiết kế hệ thống)

- Giải thích:
  - Tệp tải lên được lưu trữ tạm thời trên máy chủ.
  - Lớp DocumentProcessor thực hiện các bước tiền xử lý: trích xuất nội dung văn bản từ các loại tệp khác nhau, làm sạch dữ liệu (loại bỏ ký tự không cần thiết, chuẩn hóa văn bản) và phân đoạn văn bản (chunking) thành các đoạn nhỏ hơn, có ý nghĩa.
  - Lớp EmbeddingModel (sử dụng mô hình Sentence Transformer) được dùng để tạo vector nhúng (embedding) cho từng đoạn văn bản đã được phân đoạn.
  - Các vector nhúng này, cùng với siêu dữ liệu (metadata) của chúng (ví dụ: ID tài liệu gốc, ID đoạn, nội dung đoạn, tên file nguồn), được lưu trữ vào một collection chung duy nhất trong Vector Database (Qdrant). Việc tương tác với Qdrant được quản lý bởi lớp VectorStore.
  - Thông tin metadata của tài liệu (như tên file, ID file, user ID, loại file, ngày tải lên) được lưu vào cơ sở dữ liệu Supabase thông qua lớp FilesManager.

- Module này được kích hoạt khi người dùng thực hiện thao tác tải lên tài liệu mới.

- Vector Database (Qdrant):
  - Qdrant được triển khai như một dịch vụ cơ sở dữ liệu vector riêng biệt (ví dụ: chạy trong một Docker container).
  - Backend tương tác với Qdrant thông qua thư viện qdrant-client của Python, được trừu tượng hóa qua lớp VectorStore.
  - Toàn bộ các vector nhúng của kho tài liệu được lưu trữ tại đây, sẵn sàng cho việc tìm kiếm ngữ nghĩa.
- Module Hỏi đáp (QA Module – RAG Core):
  - Đây là module trung tâm, thực hiện pipeline Retrieval-Augmented Generation (RAG) được định nghĩa trong lớp AdvancedDatabaseRAG. Module này được kích hoạt mỗi khi có yêu cầu câu hỏi từ người dùng qua API POST /api/ask/stream.
  - Các bước xử lý chính bao gồm các 3 giai đoạn như sau:

    + Giai đoạn 1: Xử lý và Phân loại Câu hỏi Ban đầu:



SƠ ĐỒ 3.3: Sơ đồ tuần tự - Giai đoạn Xử lý và Phân loại Câu hỏi Ban đầu

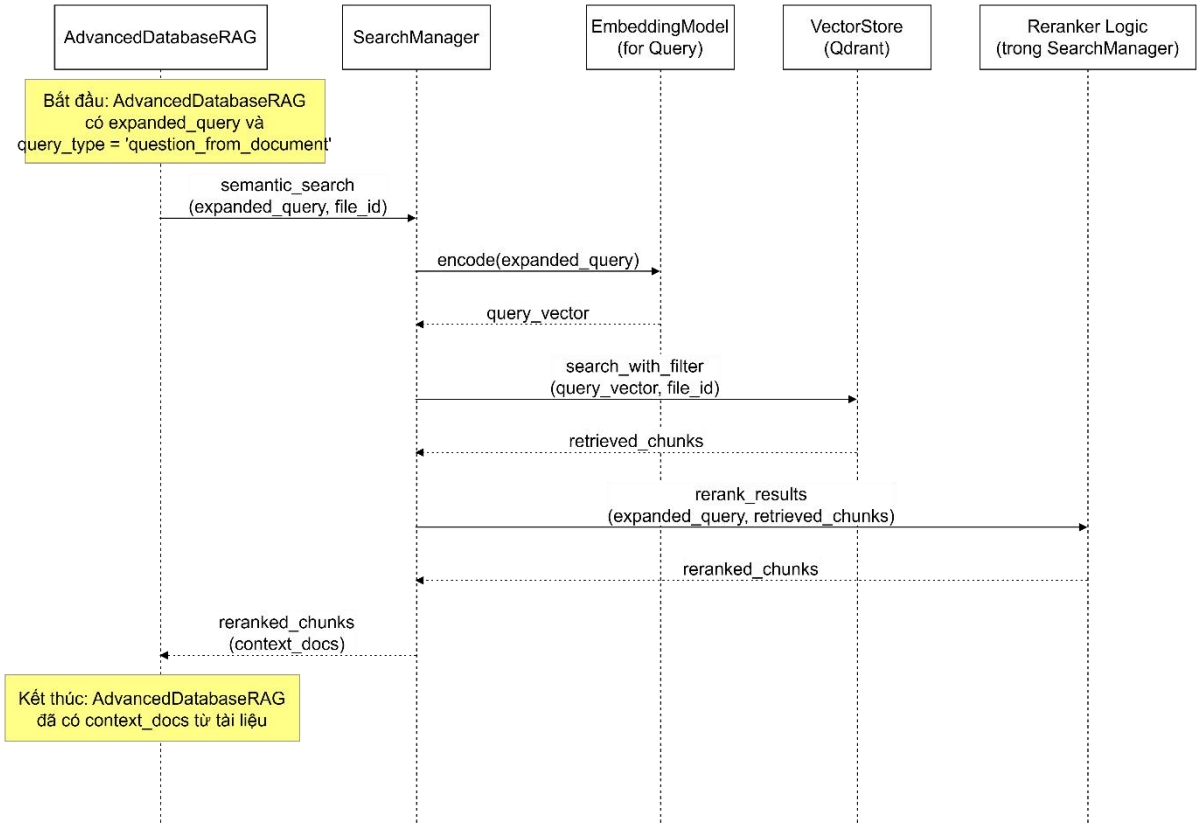
(Nguồn: tự xây dựng dựa trên thiết kế hệ thống)

Lớp QueryHandler (được khởi tạo trong lớp AdvancedDatabaseRAG) thực hiện việc mở rộng câu hỏi (giải quyết đồng tham chiếu dựa trên lịch sử hội thoại được truy xuất từ Supabase bởi ConversationManager) và phân loại

câu hỏi của người dùng thành các loại khác nhau (ví dụ: question_from_document, realtime_question, sql_code_task, other_question như định nghĩa trong QueryHandler.expand_and_classify_query) trong một bước duy nhất sử dụng mô hình LLM (Gemini). Điều này giúp làm rõ ý định của người dùng và định hướng các bước xử lý tiếp theo cho phù hợp.

+ Giai đoạn 2: Truy xuất Ngữ cảnh (Context Retrieval): Sau khi câu hỏi đã được phân loại, hệ thống tiến hành truy xuất ngữ cảnh phù hợp:

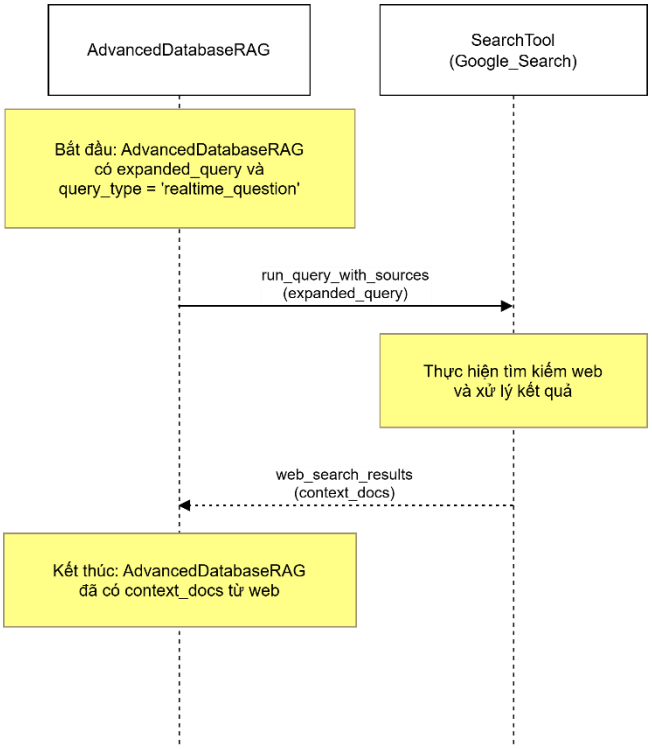
- Đối với question_from_document: Câu hỏi của người dùng (đã được mở rộng) được chuyển đổi thành một vector nhúng bằng EmbeddingModel. Vector câu hỏi này sau đó được gửi đến Qdrant (thông qua phương thức như search_with_filter của lớp VectorStore) để tìm kiếm K đoạn văn bản (chunks) có liên quan nhất từ kho tri thức chung của hệ thống. Các đoạn văn bản tìm được có thể được xếp hạng lại (Rerank) bởi SearchManager để cải thiện độ chính xác, sau đó chọn ra N đoạn có điểm số cao nhất.



SƠ ĐỒ 3.4: Sơ đồ tuần tự - Giai đoạn Xử lý "question_from_document"

(Nguồn: tự xây dựng dựa trên thiết kế hệ thống)

- Đối với realtime_question: Hệ thống sẽ sử dụng công cụ tìm kiếm web tích hợp (Google_Search.py) để thu thập thông tin từ internet.



SƠ ĐỒ 3.5: Sơ đồ tuần tự - Giai đoạn Xử lý "realtime_question"

(Nguồn: tự xây dựng dựa trên thiết kế hệ thống)

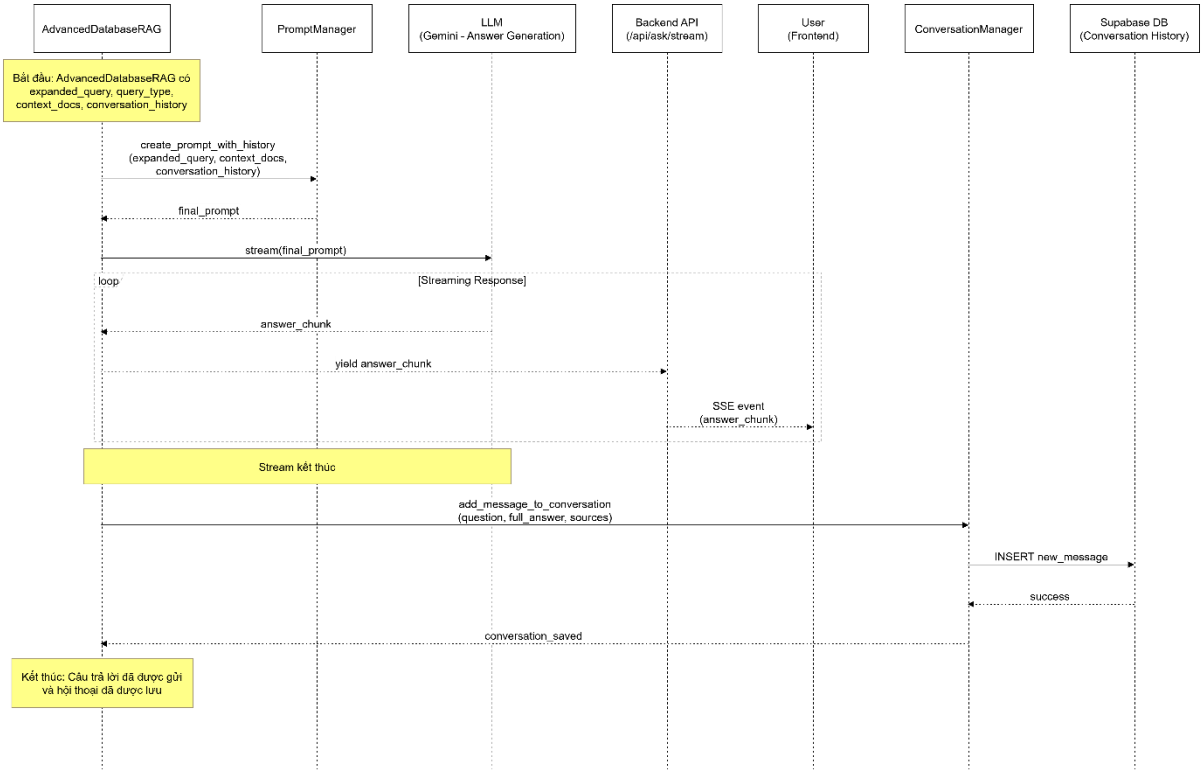
- Đối với sql_code_task và other_question: Các loại câu hỏi này thường không yêu cầu truy xuất ngữ cảnh từ tài liệu hoặc web theo cách tương tự, mà sẽ được xử lý như sau: Với sql_code_task thì sẽ được trực xử lý trực tiếp bằng LLM kết hợp với PromptManager chuyên biệt cho code task hoặc nếu là other_question thì sẽ trả về một phản hồi mặc định.

+ Giai đoạn 3: Tạo Prompt, Sinh Câu Trả Lời và Lưu trữ:

+ Câu hỏi gốc của người dùng (đã được mở rộng), cùng với N đoạn văn bản đã chọn làm ngữ cảnh (context_docs - nếu có từ Giai đoạn 2) và lịch sử hội thoại, được lớp PromptManager ghép lại thành một prompt hoàn chỉnh cho LLM. PromptManager quản lý các mẫu prompt (ví dụ: tutor_mode_system_prompt, sql_code_task_prompt), đảm bảo rằng context được định dạng rõ ràng và có kèm thông tin nguồn.

+ Prompt đã tạo được gửi đến mô hình ngôn ngữ lớn Gemini thông qua lớp GeminiLLM (ví dụ, phương thức invoke_streaming) để sinh câu trả lời (dưới dạng stream).

+ Câu trả lời từ LLM được hậu xử lý để đảm bảo định dạng và chèn trích dẫn nguồn chính xác. Cuối cùng, câu trả lời và các nguồn tham chiếu được trả về cho Frontend, đồng thời toàn bộ lượt tương tác được ConversationManager lưu vào Supabase.



SƠ ĐỒ 3.6: Sơ đồ tuần tự - Giai đoạn Tạo Prompt, Sinh Câu Trả Lời và Lưu trữ

(Nguồn: tự xây dựng dựa trên thiết kế hệ thống)

- Kết quả cuối cùng, bao gồm câu trả lời (thường được stream về Frontend để hiển thị dần) và danh sách các nguồn tài liệu chi tiết đã được sử dụng, sẽ được trả về cho Frontend.

- Module Quản lý hội thoại:
  - Sử dụng lớp SupabaseConversationManager để lưu trữ và quản lý lịch sử các cuộc hội thoại.
  - Mỗi lượt tương tác (bao gồm câu hỏi của người dùng và câu trả lời của chatbot) được lưu vào các bảng messages trong cơ sở dữ liệu Supabase. Mỗi tin nhắn được liên kết với một bản ghi trong bảng conversations.

Thông tin lưu trữ bao gồm ID người dùng, ID hội thoại, nội dung tin nhắn, vai trò của người gửi (user/assistant), thời gian tạo và có thể cả siêu dữ liệu về các nguồn tài liệu đã được sử dụng để tạo ra câu trả lời.

- Cung cấp các API endpoint cho Frontend (định nghĩa trong src/api.py) để:
  - Tạo một cuộc hội thoại mới (ví dụ: POST /api/conversations/create).
  - Lấy danh sách các cuộc hội thoại của người dùng hiện tại (ví dụ: GET /api/conversations).
  - Lấy chi tiết các tin nhắn của một cuộc hội thoại cụ thể (ví dụ: GET /api/conversations/{conversation_id}).
  - Xóa một cuộc hội thoại (ví dụ: DELETE /api/conversations/{conversation_id}).

- Module Gợi ý câu hỏi:
  - Sử dụng lớp SuggestionManager để tạo ra các câu hỏi gợi ý cho người dùng.
  - Chức năng: Dựa trên lịch sử hội thoại gần nhất của người dùng (lấy từ Supabase) hoặc dựa trên một danh sách các câu hỏi mặc định được định sẵn.
  - Khi Frontend yêu cầu qua API GET /api/suggestions module này có thể sử dụng LLM để phân tích ngữ cảnh hội thoại và đề xuất các câu hỏi liên quan, giúp người dùng khám phá sâu hơn về chủ đề. Các mẫu prompt cho việc này được quản lý trong PromptManager.

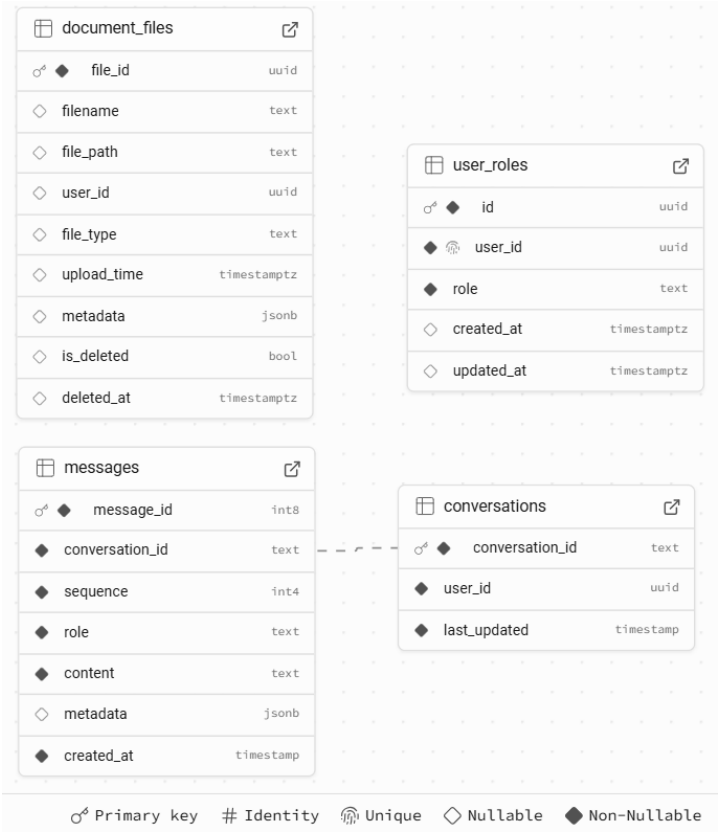
3.2.3 Database (Cơ sở dữ liệu và Kho tri thức):

Hệ thống sử dụng hai loại cơ sở dữ liệu chính, được quản lý và tương tác bởi Backend:

- Vector Store (Qdrant):
  - Như đã mô tả ở trên, Qdrant được sử dụng làm kho lưu trữ vector chuyên dụng.
  - Lưu trữ các vector nhúng của các đoạn tài liệu (chunks) cùng với metadata liên quan.
  - Hỗ trợ tìm kiếm tương đồng vector cực nhanh, là nền tảng cho bước Retrieve trong pipeline RAG.

- Hệ thống sử dụng một collection duy nhất, dùng chung cho toàn bộ tài liệu. Tất cả các truy vấn từ người dùng đều sẽ tìm kiếm trên collection chung này. Điều này giúp đơn giản hóa việc quản lý và đảm bảo mọi người dùng đều truy cập vào cùng một nguồn tri thức đã được kiểm duyệt.

- Supabase (PostgreSQL):
  - Supabase Cloud được sử dụng làm cơ sở dữ liệu quan hệ chính để lưu trữ các dữ liệu có cấu trúc. Việc định nghĩa schema cho các bảng (bao gồm conversations, messages, document_files và các policy bảo mật RLS tương ứng) được thực hiện thông qua tệp script SQL tại src/supabase/setup_supabase.sql. Script này được thực thi để thiết lập cấu trúc bảng và các quy tắc truy cập dữ liệu trên Supabase.
  - Các bảng chính bao gồm:
    - users: Lưu thông tin tài khoản người dùng (được quản lý bởi Supabase Auth).
    - conversations: Lưu thông tin về các phiên hội thoại (ví dụ: id, user_id, created_at, updated_at, title).
    - messages: Lưu trữ chi tiết từng tin nhắn trong mỗi hội thoại (ví dụ: id, conversation_id, role, content, created_at, metadata chứa thông tin nguồn).
    - document_files (hoặc tên tương tự được quản lý bởi FilesManager): Lưu thông tin metadata về các tài liệu đã được tải lên và xử lý (ví dụ: id, user_id, file_name, storage_path, file_type, status, uploaded_at, chunk_count).
    - user_roles: Dùng để quản lý quyền của người dùng. Nó lưu thông tin về roleid, user_id, role, created_at, updated_at



SƠ ĐỒ 3.7: Sơ đồ cơ sở dữ liệu quan hệ

- Backend tương tác với Supabase thông qua thư viện supabase-py, với client được khởi tạo và quản lý trong các module như SupabaseClient và SupabaseDatabase.
- Supabase cung cấp giao diện quản lý dữ liệu trực quan trên web, rất thuận tiện cho việc theo dõi và quản trị dữ liệu bởi developer.

3.2.4 Luồng tương tác chính giữa các thành phần:

- Luồng quản lý tài liệu:
  1. Admin sử dụng giao diện Frontend để tải lên tệp tài liệu.
  2. Frontend gọi API POST /api/upload của Backend (hàm upload_document trong src/api.py), gửi kèm tệp và token xác thực của Admin.
  3. Backend (Module Xử lý tài liệu):
     - Lưu tệp vào thư mục UPLOAD_DIR.
     - Xử lý tệp: trích xuất, làm sạch, phân đoạn, tạo embedding.
     - Lưu vector và metadata vào collection chung trong Qdrant.
     - Lưu metadata tài liệu vào bảng document_files trong Supabase, gắn với ID của người quản trị (admin) đã thực hiện việc tải lên để truy vết và quản lý.

4. Backend trả về kết quả cho Frontend.
5. Frontend thông báo cho người dùng.

- Luồng hỏi đáp (người dùng hỏi):
    1. Người dùng nhập câu hỏi, nhấn gửi -> frontend gọi API POST /api/ask/stream (hàm ask_question_stream trong src/api.py), gửi câu hỏi, user_id (từ token), và conversation_id.
    2. Backend nhận request, xác thực người dùng, rồi vào QA Module:
        - Tạo embedding câu hỏi.
        - Truy vấn Qdrant trên collection chung.
        - Rerank (nếu có).
        - Tạo prompt và gọi Gemini API.
        - Nhận câu trả lời từ Gemini. Hậu xử lý.
    3. Backend lưu lịch sử vào Supabase (bảng messages, conversations gắn với user_id).
    4. Backend gửi response lại cho frontend.
    5. Frontend nhận, hiển thị câu trả lời.
- Luồng gợi ý câu hỏi:
    1. Frontend gọi API GET /api/suggestions.
    2. Backend (Module Gợi ý câu hỏi) tạo gợi ý dựa trên lịch sử hội thoại của người dùng hoặc mặc định.
    3. Backend trả về danh sách gợi ý cho Frontend.
- Luồng quản lý tài liệu (ví dụ: xóa tài liệu):
    1. Người quản trị chọn tài liệu của mình cần xóa từ giao diện Frontend.
    2. Frontend gọi API DELETE /api/files/{filename} của Backend, với filename là tên file cần xóa.
    3. Backend (xác thực người dùng):
        - Tìm file_id của file đó trong bảng document_files.
        - Xóa các vector nhúng liên quan đến file_id đó khỏi collection trong Qdrant (sử dụng VectorStore).
        - Đánh dấu is_deleted = true hoặc xóa bản ghi metadata của tài liệu đó khỏi bảng document_files trong Supabase.
        - Có thể xóa tệp vật lý khỏi thư mục UPLOAD_DIR.

4. Backend trả về thông báo thành công cho Frontend.

Kiến trúc trên cho thấy các thành phần có trách nhiệm rõ ràng và giao tiếp với nhau thông qua các API.

3.3 Thiết kế cơ sở tri thức (Knowledge Base):

Cơ sở tri thức của hệ thống chatbot RAG được thiết kế để lưu trữ và truy xuất thông tin từ các tài liệu học tập chính thống một cách tối ưu, hỗ trợ tìm kiếm ngữ nghĩa chính xác cho toàn bộ người dùng từ một nguồn tri thức tập trung và nhất quán. Thiết kế bao gồm ba thành phần chính: chiến lược tiền xử lý và phân đoạn tài liệu, cấu trúc lưu trữ vector và metadata và cơ chế cập nhật tri thức.

3.3.1 Chiến lược tiền xử lý và phân đoạn tài liệu:

- Hệ thống sử dụng lớp DocumentProcessor để thực hiện quá trình tiền xử lý và phân đoạn tài liệu với các tính năng sau:
- Hỗ trợ đa định dạng tài liệu:

  + PDF: Sử dụng PyPDFLoader để trích xuất văn bản từ các file PDF.

  + DOCX: Sử dụng Docx2txtLoader để xử lý tài liệu Word.

  + TXT/SQL/MD: Sử dụng TextLoader để đọc các file văn bản thuần túy.

  + Chuyển đổi tự động: Các định dạng .doc, .ppt, .xls và các định dạng OpenDocument được chuyển đổi tự động sang PDF thông qua LibreOffice trước khi xử lý.
- Chiến lược phân đoạn thích ứng: Hệ thống triển khai hai phương pháp phân đoạn chính, được điều khiển bởi tham số use_structural_chunking:

  1. Phân đoạn theo cấu trúc:
     - Nhận diện tiêu đề tự động: Sử dụng regex pattern để phát hiện các tiêu đề trong tài liệu.
     - Phân loại nội dung: Tự động phân loại từng đoạn thành các loại: heading, text, table, list, code.
     - Gộp tiêu đề với nội dung: Khi phát hiện tiêu đề, hệ thống tự động gộp với đoạn văn tiếp theo để đảm bảo ngữ cảnh.
     - Xử lý đặc biệt cho bảng và code: Nhận diện và xử lý riêng các đoạn chứa bảng (có ký tự | và -) và code (có dấu ``` hoặc nhiều khoảng trắng).
  2. Phân đoạn theo kích thước:

- Sử dụng RecursiveCharacterTextSplitter với các tham số có thể cấu hình:
    1. Kích thước chunk mặc định: 800 ký tự (có thể điều chỉnh qua biến môi trường CHUNK_SIZE).
    2. Độ chồng lấp: 150 ký tự (có thể điều chỉnh qua biến môi trường CHUNK_OVERLAP).
    3. Separators tối ưu: ["\n\n", "\n", ".", " ", ""] để ưu tiên cắt theo đoạn văn, câu, từ.

- Phân loại nội dung tự động: Hệ thống triển khai chức năng phân loại tự động với từ khóa đặc trưng cho từng danh mục:

  + SQL: select, insert, update, delete, join, primary key, foreign key, v.v.

  + NoSQL: mongodb, redis, cassandra, neo4j, document store, v.v.

  + Database Design: schema, normalization, er diagram, data modeling, v.v.

  + Database Administration: backup, restore, replication, security, v.v.

- Làm giàu metadata: Mỗi chunk được làm giàu với metadata chi tiết:

  + Thông tin cơ bản: source, page, chunk_type, position, category.

  + Phân tích nội dung: chứa_định_nghĩa, chứa_cú_pháp, chứa_mẫu_code, ch ứa_bảng, chứa_hình_ảnh.

  + Phân tích SQL cụ thể: chứa_cú_pháp_select, chứa_cú_pháp_join, chứa_cú _pháp_ddl, chứa_cú_pháp_dml.

3.3.2 Cấu trúc lưu trữ vector và metadata:

- Thiết kế Store (Qdrant): thống sử dụng lớp VectorStore (tại src/vector_store.py) để quản lý kho lưu trữ vector với các đặc điểm:

  + Collection chung duy nhất: Toàn bộ vector nhúng từ tất cả các tài liệu do Admin tải lên đều được lưu trữ trong một collection duy nhất trên Qdrant. Thiết kế này giúp đơn giản hóa việc truy vấn, vì mọi câu hỏi của người dùng đều tìm kiếm trên toàn bộ kho tri thức đã được phê duyệt. Metadata `file_id` sẽ được sử dụng để phân biệt các đoạn văn bản thuộc về các tài liệu khác nhau khi cần.

  + Vector dimension: 384 chiều (sử dụng mô hình intfloat/multilingual-e5-small).

  + Distance metric: Cosine similarity để đo độ tương đồng ngữ nghĩa.

- Schema Payload (Metadata): Mỗi vector trong Qdrant được lưu kèm payload JSON với cấu trúc:

```
{
 "text": "Nội dung của đoạn văn bản ...",
 "source_file": "ten_file_goc.pdf",
 "page": 1,
 "chunk_index": 0,
 "file_id": "uuid_cua_file_trong_supabase",
 "category": "sql",
 "chunk_type": "text",
 "metadata": {
  "chua_dinh_nghia": true,
  "chua_mau_code": false,
  .....
 }
}
```

- Quản lý metadata trong Supabase: Bảng document_files lưu trữ thông tin tài liệu với schema:
  + file_id: UUID duy nhất cho mỗi file.
  + user_id: UUID của người upload file.
  + filename: Tên file gốc.
  + file_path: Đường dẫn lưu trữ.
  + file_type: Loại file (pdf, docx, txt, sql).
  + upload_time: Thời gian upload.
  + processing_status: Trạng thái xử lý.
  + chunk_count: Số lượng chunk được tạo.
  + file_size: Kích thước file.
  + is_deleted: Cờ đánh dấu xóa mềm.

3.3.3 Chiến lược tối ưu truy xuất:

- Tìm kiếm thông minh: Hệ thống triển khai phương pháp semantic _search trong lớp AdvancedDatabaseRAG kết hợp:
  + Semantic search: Tìm kiếm theo nghĩa sử dụng vector embedding.
  + Reranking: Sử dụng mô hình cross-encoder/ms-marco-MiniLM-L-6-v2 để tái xếp hạng kết quả.

- Filtering: Filter theo file: Hỗ trợ tìm kiếm trong các file cụ thể thông qua file_id.

3.3.4 Cơ chế cập nhật và quản lý tri thức:

- Upload và processing pipeline: Khi người dùng upload tài liệu mới, hệ thống thực hiện:
    1. Lưu tạm thời: File được lưu vào UPLOAD_DIR.
    2. Preprocessing: Chuyển đổi định dạng nếu cần và trích xuất văn bản.
    3. Chunking: Phân đoạn theo chiến lược phù hợp.
    4. Embedding: Tạo vector embedding cho từng chunk.
    5. Storage: Lưu vector vào Qdrant và metadata vào Supabase.
    6. Cleanup: Dọn dẹp file tạm và cập nhật trạng thái.

Cơ chế xóa và cập nhật:

- Xóa mềm: Đánh dấu is_deleted = true trong Supabase.
- Xóa vector: Loại bỏ tất cả vector liên quan khỏi Qdrant collection.
- Cascade delete: Xóa đồng bộ trên cả Qdrant và Supabase.
- Rollback: Khả năng khôi phục trong trường hợp lỗi.

3.3.5 Trích dẫn nguồn:

Hệ thống triển khai cơ chế trích dẫn nguồn thông qua:

- Template prompt: Sử dụng PromptManager để định dạng context với thông tin nguồn.
- Source tracking: Mỗi chunk được đánh dấu với source_file và page.
- Citation formatting: Tự động tạo trích dẫn dạng "(tên_file, trang X)".
- Source validation: Kiểm tra và làm sạch thông tin trích dẫn.

3.3.6 Bảo mật và phân quyền:

- API authentication: Xác thực token cho mọi thao tác.
- Xóa hoàn toàn: Khả năng xóa tất cả dữ liệu khi cần.
- No sharing: Không chia sẻ dữ liệu giữa các người dùng.

3.4 Thiết kế luồng xử lý RAG:

Luồng xử lý RAG (Retrieval-Augmented Generation) là thành phần cốt lõi của hệ thống chatbot, đảm bảo việc trả lời câu hỏi của sinh viên một cách chính xác, có căn cứ từ tài liệu và cung cấp trích dẫn nguồn rõ ràng. Dưới đây là thiết kế chi tiết từng bước trong luồng này, từ khi nhận câu hỏi đến khi trả lời người

dùng, được triển khai chủ yếu trong lớp AdvancedDatabaseRAG và các module phụ trợ.

3.4.1 Nhận và Tiền xử lý Câu hỏi:

1. Tiếp nhận yêu cầu:
    - Người dùng nhập câu hỏi vào giao diện chat trên Frontend (component ChatInterface trong frontend/components/chat-interface.tsx).
    - Frontend gửi yêu cầu POST đến API endpoint /api/ask/stream (định nghĩa trong src/api.py). Yêu cầu này bao gồm: câu hỏi (question), ID hội thoại hiện tại (conversation_id - nếu có).
2. Xử lý và Phân loại Câu hỏi:
    - Trong phương thức query_with_sources_streaming của lớp Advance dDatabaseRAG, câu hỏi gốc (original_query) và lịch sử hội thoại (conversation_history) được chuyển đến module QueryHandler (khởi tạo từ src/query_handler.py).
    - Phương thức expand_and_classify_query của QueryHandler thực hiện hai nhiệm vụ chính thông qua một lần gọi LLM (Gemini):
      + Mở rộng câu hỏi (Query Expansion): Dựa vào lịch sử hội thoại, giải quyết các đại từ tham chiếu (ví dụ: "nó", "cái đó") để tạo ra một expanded_query đầy đủ ngữ nghĩa và độc lập. Nếu câu hỏi đã rõ ràng, expanded_query sẽ giống câu hỏi gốc.
      + Phân loại câu hỏi (Query Classification): expanded_query được phân loại thành một trong các loại sau:
        - question_from_document: Câu hỏi yêu cầu tra cứu thông tin từ tài liệu đã tải lên. Đây là luồng RAG chính.
        - sql_code_task: Yêu cầu trực tiếp liên quan đến việc tạo, giải thích, hoặc sửa đổi mã SQL.
        - realtime_question: Câu hỏi cần thông tin cập nhật, thời sự (ví dụ: "xu hướng CSDL mới nhất").
        - other_question: Câu hỏi không liên quan đến chuyên ngành cơ sở dữ liệu.

- Câu hỏi đã được xử lý (query_to_use = expanded_query) và loại câu hỏi (query_type) được sử dụng cho các bước tiếp theo. Hệ thống ghi log lại thông tin này.

3.4.2 Định tuyến xử lý dựa trên loại câu hỏi:

- Hệ thống sẽ định tuyến xử lý dựa trên query_type đã xác định:

1. Trường hợp query_type == "other_question":
    - Hệ thống không thực hiện RAG.
    - Gọi phương thức get_response_for_other_question của QueryHandler để tạo một câu trả lời mẫu (ví dụ: "Xin lỗi, tôi chỉ có thể hỗ trợ các câu hỏi liên quan đến Cơ sở dữ liệu.").
    - Luồng trả về các sự kiện start, sources (rỗng), content (câu trả lời mẫu) và end cho Frontend.
2. Trường hợp query_type == "sql_code_task":
    - Hệ thống không thực hiện truy xuất từ Vector DB.
    - Lấy template prompt chuyên dụng cho việc xử lý SQL từ PromptManager.
    - Tạo final_prompt bằng cách kết hợp query_to_use và conversation_history (nếu có) vào template.
    - Gọi trực tiếp phương thức self.llm.stream(final_prompt) (sử dụng GeminiLLM để LLM tạo mã SQL hoặc giải thích.
    - Luồng trả về các sự kiện start (không có nguồn), content (các phần của mã SQL/giải thích) và end.
3. Trường hợp query_type == "realtime_question":
    - Hệ thống sử dụng công cụ tìm kiếm web bên ngoài thông qua google_agent_search (từ src/tools/Google_Search.py).
    - Kết quả tìm kiếm từ web (tóm tắt và URLs) được định dạng thành các retrieved_documents và gas_sources_list.
    - Thông tin nguồn từ web được gửi về Frontend qua sự kiện sources.
    - Một prompt được tạo bởi PromptManager.create_prompt_with_history sử dụng query_to_use, ngữ cảnh từ kết quả tìm kiếm web và conversation_history.
    - Gọi self.llm.stream(prompt) để LLM tổng hợp câu trả lời.

- Luồng trả về các sự kiện start, sources (từ web), content (câu trả lời tổng hợp) và end.

4. Trường hợp query_type == "question_from_document" (Luồng RAG chính): Các bước tiếp theo sẽ được thực hiện.

3.4.3 Tìm kiếm Ngữ nghĩa (Semantic Search):

- Thực hiện tìm kiếm: Gọi phương thức self.semantic_search(query_to_use, k=k, file_id=file_id).
  - Bên trong, SearchManager (từ src/search.py) sẽ:

    + Sử dụng EmbeddingModel (từ src/embedding.py, ví dụ: intfloat /multilingual-e5-small) để tạo vector nhúng cho query_to_use.

    + Gọi phương thức search_with_filter của VectorStore (từ src/vector_sto re.py) để truy vấn Qdrant.
      - Collection: Truy vấn trên collection chung duy nhất của hệ thống.
      - Vector: Vector của câu hỏi.
      - Limit (k): Số lượng kết quả cần lấy (mặc định là 10).
      - Filter: Nếu file_id được cung cấp, Qdrant sẽ chỉ tìm kiếm trong các tài liệu có file_id tương ứng trong metadata. Bộ lọc này không cần lọc theo user_id vì tất cả tài liệu đều là chung.
  - Kết quả là một danh sách search_results gồm các đoạn văn bản (chunks) liên quan nhất, mỗi chunk chứa text, metadata (bao gồm source_file, page, file_id, v.v.) và score (độ tương đồng).

3.4.4 Cơ chế Fallback (Nếu không có kết quả tìm kiếm ngữ nghĩa):

- Nếu search_results rỗng (không tìm thấy chunk nào từ tài liệu người dùng):
  - Hệ thống kích hoạt cơ chế fallback, sử dụng google_agent_search để tìm kiếm thông tin trên web cho query_to_use.
  - Nếu tìm thấy kết quả từ web, chúng sẽ được định dạng và sử dụng thay thế cho search_results. Biến gas_fallback_used được đặt thành True.
  - Nếu cả tìm kiếm trong tài liệu và fallback web đều không có kết quả, hệ thống sẽ gửi về Frontend thông báo "Không tìm thấy thông tin liên quan..." qua sự kiện content, kèm theo các sự kiện start và sources (rỗng), sau đó là end.

3.4.5 Áp dụng Reranking:

- Nếu có nhiều hơn một kết quả trong search_results (từ tài liệu hoặc fallback web):
  - Gọi phương thức self.rerank_results(query_to_use, search_results).
  - Bên trong, SearchManager sử dụng một mô hình Cross-Encoder (ví dụ: cross-encoder/ms-marco-MiniLM-L-6-v2) để đánh giá lại mức độ liên quan của từng cặp (câu hỏi, chunk) và sắp xếp lại các chunks.
  - Kết quả là danh sách reranked_results. Nếu ban đầu chỉ có một kết quả, bước rerank sẽ được bỏ qua.

3.4.6 Chuẩn bị Ngữ cảnh và Prompt cho LLM:

1. Thông báo bắt đầu và nguồn (Events start, sources):
   - Gửi sự kiện start cho Frontend, chứa thông tin query_type, file_id đã sử dụng, total_results (số lượng kết quả từ semantic search ban đầu), total_reranked (số lượng kết quả sau rerank).
   - Chuẩn bị context_docs: Lấy tối đa 10 chunks đầu tiên từ reranked_results để làm ngữ cảnh cho LLM. Mỗi context_doc chứa content (nội dung chunk), source (tên file), page (số trang, ưu tiên page_label nếu có), section, score và toàn bộ metadata của chunk.
   - Chuẩn bị sources_list: Một danh sách các đối tượng nguồn chi tiết từ reranked_results để hiển thị cho người dùng. Mỗi đối tượng nguồn bao gồm source (tên file), page, page_label, section, score, content_snippet (nội dung chunk), file_id, is_web_search (đánh dấu nếu nguồn từ fallback web) và source_filename (tên file không kèm đường dẫn).
   - Gửi sự kiện sources cho Frontend, chứa sources_list.
2. Tạo Prompt:
   - Gọi phương thức create_prompt_with_history của PromptManager.
   - Prompt này kết hợp:
     - query_to_use (câu hỏi đã xử lý của người dùng).
     - context_docs (các đoạn văn bản liên quan đã được rerank, được định dạng kèm thông tin nguồn như (Nguồn: {source_filename}, trang {page_label})).

- conversation_history (lịch sử hội thoại trước đó).
- Các chỉ dẫn hệ thống (system instructions) từ template (ví dụ: tutor_mode_system_prompt hoặc default_system_prompt yêu cầu LLM đóng vai trợ giảng, trả lời dựa trên ngữ cảnh, trích dẫn nguồn theo định dạng đã cung cấp và trả lời bằng tiếng Việt.

3.4.7 Gọi LLM để Sinh Câu trả lời (Streaming):

- Gọi phương thức self.llm.stream(prompt) của GeminiLLM.
- GeminiLLM sẽ gửi prompt đến API của Google Gemini và nhận về câu trả lời dưới dạng một luồng (stream) các phần văn bản (chunks).
- Mỗi phần văn bản (content_chunk) từ LLM được gửi ngay về Frontend thông qua sự kiện content. Điều này cho phép người dùng thấy câu trả lời xuất hiện dần.
- GeminiLLM có cơ chế thử lại với API key khác nếu gặp lỗi quota hoặc rate limit.

3.4.8 Kết thúc Luồng và Trả kết quả cho Người dùng:

1. Thông báo kết thúc (Event end):
    - Sau khi LLM hoàn tất việc sinh câu trả lời (hoặc nếu có lỗi xảy ra và luồng xử lý kết thúc sớm), hệ thống tính toán tổng thời gian xử lý (elapsed_time).
    - Gửi sự kiện end cho Frontend, chứa processing_time và query_type.
2. Hiển thị trên Frontend:
    - Component ChatInterface trên Frontend nhận các sự kiện Server-Sent Events (SSE):
        - start: Chuẩn bị cho câu trả lời mới.
        - sources: Lưu trữ danh sách các nguồn tham khảo (sources_list) để hiển thị kèm theo câu trả lời. Người dùng có thể tương tác với các nguồn này (ví dụ: click để xem chi tiết).
        - content: Nối các phần văn bản nhận được để hiển thị dần câu trả lời của chatbot.
        - end: Đánh dấu câu trả lời đã hoàn chỉnh.
        - error: Hiển thị thông báo lỗi nếu có.

- Trích dẫn nguồn trong câu trả lời của LLM (ví dụ: (Nguồn: ten_file.pdf, trang X)) được hiển thị trực tiếp.

CHƯƠNG 4 TRIỂN KHAI HỆ THỐNG

4.1 Lựa chọn công nghệ:

Việc lựa chọn công nghệ phù hợp là yếu tố then chốt để đảm bảo hiệu quả, khả năng mở rộng và bảo trì của hệ thống. Dựa trên các yêu cầu đã phân tích, các công nghệ sau đã được lựa chọn và sử dụng:

- Backend:
    - Ngôn ngữ lập trình: Python (phiên bản 3.9+) được chọn vì hệ sinh thái mạnh mẽ cho AI/ML, xử lý ngôn ngữ tự nhiên và sự phong phú của các thư viện hỗ trợ.
    - Framework: FastAPI được sử dụng để xây dựng các API backend. FastAPI nổi bật với hiệu năng cao, dễ sử dụng, tài liệu tốt và khả năng tự động sinh OpenAPI schema, phù hợp cho việc phát triển API nhanh chóng và hiện đại.
- Frontend:
    - Next.js: Là framework dựa trên React do Vercel phát triển, hỗ trợ Server-Side Rendering (SSR), Static Site Generation (SSG) và Incremental Static Regeneration (ISR). Next.js tối ưu hiệu suất, SEO, routing theo cấu trúc thư mục, tích hợp sẵn TypeScript, hỗ trợ CSS module và sử dụng trình biên dịch SWC hiện đại.
    - React: Thư viện JavaScript mã nguồn mở của Meta (Facebook), sử dụng Virtual DOM để cập nhật giao diện hiệu quả. React hỗ trợ JSX, cho phép xây dựng giao diện theo mô hình component, giúp tái sử dụng và bảo trì dễ dàng, đặc biệt phù hợp với các ứng dụng đơn trang (SPA).
    - TypeScript: Là phần mở rộng của JavaScript, bổ sung hệ thống kiểu tĩnh giúp phát hiện lỗi khi biên dịch. TypeScript hỗ trợ interface, enum, generic và được biên dịch sang JavaScript, giúp mã nguồn rõ ràng, dễ kiểm soát và phù hợp với các dự án quy mô lớn.
    - Tailwind CSS: Framework CSS “utility-first” cho phép tạo giao diện thông qua các lớp tiện ích trực tiếp trong HTML. Tailwind hỗ trợ responsive design, dark mode, trạng thái hover/focus và sử dụng trình biên dịch JIT (Just-in-Time) giúp tăng tốc phát triển và giảm kích thước tệp CSS.

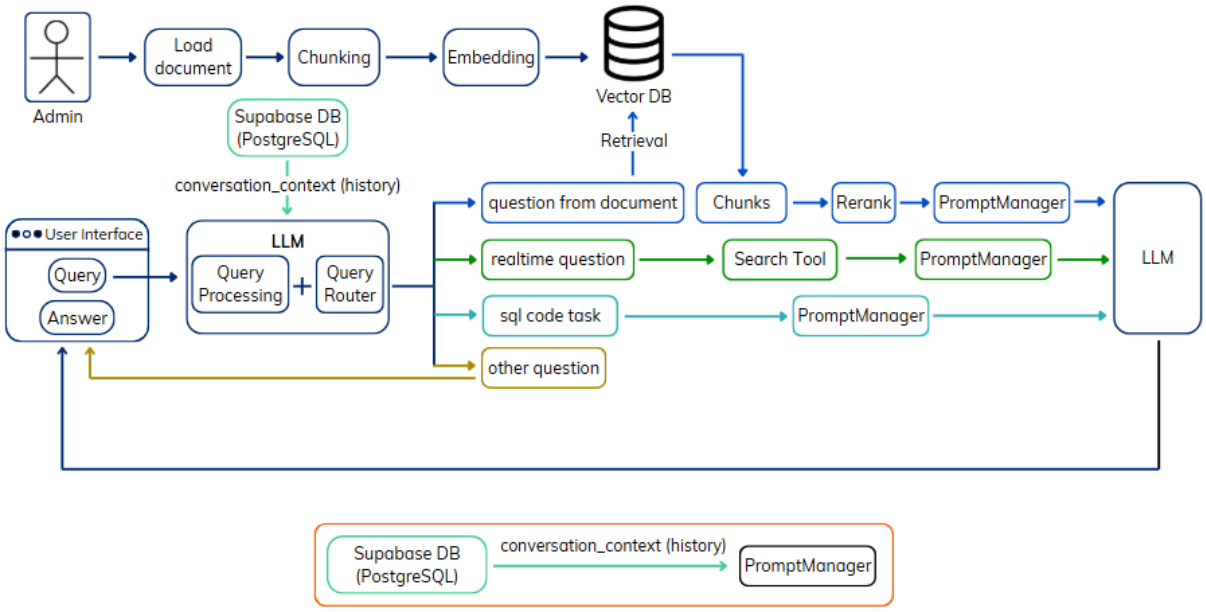
- Mô hình Nhúng (Embedding Model):
    - Sentence Transformers: Cụ thể, mô hình intfloat/multilingual-e5-small (triển khai trong src/embedding.py) được sử dụng để chuyển đổi văn bản (câu hỏi và các đoạn tài liệu) thành các vector nhúng ngữ nghĩa. Đây là một mô hình đa ngôn ngữ, hiệu quả và có kích thước nhỏ, phù hợp cho việc triển khai.
- Cơ sở dữ liệu Vector (Vector Database):
    - Qdrant: Được chọn làm cơ sở dữ liệu vector để lưu trữ và truy vấn các vector nhúng. Qdrant cung cấp khả năng tìm kiếm tương đồng vector hiệu suất cao, hỗ trợ metadata filtering và dễ dàng tích hợp (triển khai trong src/vector_store.py).
- Mô hình Ngôn ngữ Lớn (LLM):
    - Google Gemini API: Hệ thống sử dụng API của Google Gemini (cụ thể là model gemini-2.0-flash hoặc các model tương tự được cấu hình) để thực hiện các tác vụ sinh văn bản, trả lời câu hỏi dựa trên ngữ cảnh, phân loại câu hỏi và tạo gợi ý (triển khai trong src/llm.py).
- Cơ sở dữ liệu Quan hệ và Xác thực:
    - Supabase: Được sử dụng như một Backend-as-a-Service (BaaS), cung cấp cơ sở dữ liệu PostgreSQL để lưu trữ metadata của tài liệu, lịch sử hội thoại, thông tin người dùng và các chức năng xác thực người dùng (Supabase Auth). Việc tương tác được thực hiện qua các module trong src/supabase/).
- Mô hình Xếp hạng lại (Reranker Model):
    - Cross-Encoder: Mô hình cross-encoder/ms-marco-MiniLM-L-6-v2 được sử dụng để xếp hạng lại các kết quả tìm kiếm từ Qdrant, giúp cải thiện độ chính xác của các đoạn văn bản liên quan nhất được chọn làm ngữ cảnh cho LLM (triển khai trong src/search.py).
- Thư viện Xử lý Ngôn ngữ Tự nhiên (NLP) và Tài liệu:
    - LangChain: Được sử dụng cho các tác vụ như tải và xử lý tài liệu (DocumentProcessor trong src/document_processor.py sử dụng các loaders như PyPDFLoader, Docx2txtLoader, TextLoader) và chia nhỏ văn bản (RecursiveCharacterTextSplitter).

- Các thư viện khác: pypdf, python-docx, unstructured (cho việc xử lý đa dạng định dạng tài liệu), tiktoken (để ước lượng token).

- Công cụ Tìm kiếm Web (Web Search Tool):
  - Google Custom Search API (hoặc tương tự): Được tích hợp thông qua module src/tools/Google_Search.py để hỗ trợ trả lời các câu hỏi cần thông tin thời sự hoặc khi không tìm thấy thông tin trong tài liệu đã tải lên.

4.2 Xây dựng Backend:

Backend của hệ thống được xây dựng bằng Python và FastAPI, bao gồm các module chính được thiết kế để thực hiện quy trình Nạp liệu (Data Ingestion) và Hỏi đáp dựa trên kiến trúc RAG (Retrieval Augmented Generation). Sơ đồ tổng quan về kiến trúc luồng xử lý của backend được trình bày trong hình dưới đây:



SƠ ĐỒ 4.1: Sơ đồ tổng quan kiến trúc luồng xử lý của Backend hệ thống RAG

(Nguồn: Tự xây dựng theo thiết kế hệ thống)

Như minh họa trong hình trên, backend của hệ thống bao gồm hai luồng xử lý chính:

- Luồng Nạp liệu (Data Ingestion - phần trên của sơ đồ): Bắt đầu từ việc người dùng tải tài liệu lên thông qua API, tài liệu này sẽ được xử lý bởi DocumentProcessor (src/document_processor.py) để trích xuất nội dung, tiền xử lý và chia nhỏ (Chunking) thành các đoạn văn bản (chunks) có ý nghĩa. Mỗi chunk sau đó được chuyển đổi thành vector nhúng (Embedding) bởi EmbeddingModel (src/embedding.py) sử dụng mô hình Sentence

Transformer. Các vector nhúng này cùng với metadata của chúng được lưu trữ vào Cơ sở dữ liệu Vector (Vector DB - Qdrant) thông qua VectorStore (src/vector_store.py). Song song đó, thông tin metadata của tài liệu gốc (như tên file, user ID, trạng thái xử lý) được quản lý và lưu trữ vào cơ sở dữ liệu quan hệ Supabase PostgreSQL) bởi module FilesManager (src/supabase/files_manager.py).

- Luồng Hỏi đáp (Query Processing & RAG Core - phần dưới của sơ đồ): Khi người dùng đặt câu hỏi qua Giao diện Người dùng (User Interface), yêu cầu sẽ được gửi đến API Endpoints (src/api.py) và được điều phối bởi module trung tâm AdvancedDatabaseRAG (src/rag.py). Câu hỏi ban đầu, cùng với lịch sử hội thoại (được truy xuất từ Supabase DB bởi ConversationManager (src/supabase/conversation_manager.py)), sẽ được QueryHandler (src/query_handler.py) xử lý. QueryHandler sử dụng LLM để mở rộng câu hỏi (ví dụ: giải quyết đồng tham chiếu) và phân loại câu hỏi thành các loại chính:
  - question_from_document: Nếu câu hỏi liên quan đến nội dung tài liệu đã tải lên, SearchManager (src/search.py) sẽ thực hiện nhúng câu hỏi, truy xuất các chunks liên quan từ Vector DB (Qdrant) và sau đó xếp hạng lại (Rerank) các chunks này để chọn ra những ngữ cảnh phù hợp nhất. Ngữ cảnh này, cùng với câu hỏi đã xử lý và lịch sử hội thoại (nếu có, lấy từ Supabase DB), sẽ được đưa vào PromptManager (src/prompt_manager.py) để tạo prompt hoàn chỉnh trước khi đưa tới LLM.
  - realtime_question: Nếu câu hỏi đòi hỏi thông tin thời sự hoặc kiến thức chung ngoài tài liệu, hệ thống sẽ sử dụng công cụ tìm kiếm web tích hợp (Google_Search trong src/tools/ để thu thập thông tin. Thông tin thu thập được này, cùng với câu hỏi đã xử lý và lịch sử hội thoại (nếu có, lấy từ Supabase DB), sẽ được đưa vào PromptManager để tạo prompt hoàn chỉnh trước khi đưa tới LLM.
  - sql_code_task: Nếu câu hỏi liên quan đến việc tạo, giải thích hoặc sửa đổi mã SQL, câu hỏi đã xử lý và lịch sử hội thoại sẽ được PromptManager sử dụng để tạo một prompt chuyên biệt cho tác vụ SQL (sử

dụng sql_code_task_prompt) trước khi gửi đến LLM. Luồng này không yêu cầu truy xuất ngữ cảnh từ tài liệu hay web.

- other_question: Các câu hỏi không thuộc các loại trên, ví dụ như không liên quan đến lĩnh vực Cơ sở dữ liệu, sẽ được hệ thống phản hồi bằng một thông báo mặc định cho biết không thể trả lời, dựa trên logic trong QueryHandler.

- API Endpoints (src/api.py): Định nghĩa tất cả các HTTP endpoints mà Frontend sử dụng để tương tác, như bảng sau:

BẢNG 4.1: API kiểm tra hệ thống

Phương thức	**GET**
Endpoint	{API_DOMAIN}/api/
Diễn giải	Trả về thông điệp chào mừng & hướng dẫn truy cập tài liệu API

BẢNG 4.2: API đặt câu hỏi dạng stream

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/ask/stream	
Thuộc tính	question	Câu hỏi cần trả lời
	file_id	Danh sách file_id được tìm kiếm
	conversation_id	Mã cuộc trò chuyện
	max_sources	Query param – số nguồn tham khảo tối đa
Diễn giải	Đặt câu hỏi và nhận câu trả lời từ hệ thống RAG dưới dạng stream	

BẢNG 4.3: API tải lên & index tài liệu

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/upload	
Thuộc tính	file	File cần upload (PDF/DOCX/TXT/SQL/MD)
	category	Danh mục tài liệu (form-data, tùy chọn)

Diễn giải	Lưu file, cắt chunk, tạo embedding & index vào vector DB

BẢNG 4.4: API đặt lại collection

Phương thức	**DELETE**
Endpoint	{API_DOMAIN}/api/collection/reset
Diễn giải	Xoá toàn bộ dữ liệu đã index và tạo collection mới rỗng

BẢNG 4.5: API lấy danh sách file

Phương thức	**GET**
Endpoint	{API_DOMAIN}/api/files
Diễn giải	[ADMIN] Trả về danh sách file đã upload

BẢNG 4.6: API xoá file

Phương thức	**DELETE**	
Endpoint	{API_DOMAIN}/api/files/{filename}	
Thuộc tính	filename	Tên file (path-param)
Diễn giải	[ADMIN] Xoá file và toàn bộ điểm dữ liệu liên quan trong vector DB	

BẢNG 4.7: API xoá điểm (Point) bằng filter

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/collections/delete-by-filter	
Thuộc tính	filter	Định nghĩa điều kiện xoá (JSON body)
Diễn giải	Xoá các vector thoả mãn bộ lọc tuỳ chỉnh	

BẢNG 4.8 API liệt kê hội thoại của người dùng

Phương thức	**GET**

Endpoint	{API_DOMAIN}/api/conversations	
Thuộc tính	page	Trang hiện tại (query)
	page_size	Số hội thoại mỗi trang
Diễn giải	Trả về danh sách hội thoại của người dùng hiện tại.	

BẢNG 4.9: API chi tiết hội thoại

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/conversations/{conversation_id}	
Thuộc tính	conversation_id	ID hội thoại (path-param)
Diễn giải	Lấy toàn bộ tin nhắn của một hội thoại	

BẢNG 4.10: API tạo hội thoại mới

Phương thức	**POST**
Endpoint	{API_DOMAIN}/api/conversations/create
Diễn giải	Sinh một conversation_id mới cho người dùng

BẢNG 4.11: API xoá hội thoại

Phương thức	**DELETE**	
Endpoint	{API_DOMAIN}/api/conversations/{conversation_id}	
Thuộc tính	conversation_id	ID hội thoại cần xoá (path-param)
Diễn giải	Xoá hội thoại và toàn bộ tin nhắn liên quan	

BẢNG 4.12: API đăng ký tài khoản

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/auth/signup	
Thuộc tính	email	Email đăng ký

	password	Mật khẩu
Diễn giải	Tạo người dùng mới và trả về token phiên	

BẢNG 4.13: API đăng nhập

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/auth/login	
Thuộc tính	email	Email đăng nhập
	password	Mật khẩu
Diễn giải	Xác thực và lấy access token	

BẢNG 4.14: API đăng xuất

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/auth/logout	
Thuộc tính	Authorization	Bearer token hiện tại (header)
Diễn giải	Vô hiệu hoá token & kết thúc phiên	

BẢNG 4.15: API quên mật khẩu

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/auth/forgot-password	
Thuộc tính	email	Email nhận liên kết đặt lại
	redirect_to	URL chuyển hướng (tùy chọn)
Diễn giải	Gửi email đặt lại mật khẩu thông qua Supabase	

BẢNG 4.16: API đặt lại mật khẩu

Phương thức	**POST**
Endpoint	{API_DOMAIN}/api/auth/reset-password

Thuộc tính	password	Mật khẩu mới
	access_token	Token xác thực từ email
Diễn giải	Cập nhật mật khẩu mới cho tài khoản	

BẢNG 4.17: API lấy thông tin người dùng

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/auth/user	
Thuộc tính	Authorization	Bearer token (header)
Diễn giải	Trả về ID, email & ngày tạo của người dùng hiện tại	

BẢNG 4.18: API kiểm tra phiên

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/auth/session	
Thuộc tính	Authorization	Bearer token (header, tuỳ chọn)
Diễn giải	Kiểm tra token & trạng thái xác thực	

BẢNG 4.19: API đăng nhập/đăng ký Google OAuth

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/auth/google	
Thuộc tính	code	Authorization code (tùy chọn)
	access_token	Google access token (tùy chọn)
	provider	Nhà cung cấp OAuth (mặc định `google`)
Diễn giải	Xác thực với Google, tạo hoặc đăng nhập tài khoản	

BẢNG 4.20: API lấy Google sign-in URL

Phương thức	**GET**

Endpoint	{API_DOMAIN}/api/auth/google/url	
Thuộc tính	redirect_url	URL chuyển hướng sau đăng nhập (query)
Diễn giải	Tạo URL OAuth của Google với redirect tuỳ chọn	

BẢNG 4.21: API OAuth callback

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/auth/callback	
Thuộc tính	code	Authorization code
	error	Thông điệp lỗi (nếu có)
	provider	Nhà cung cấp (mặc định `google`)
Diễn giải	Xử lý callback OAuth & tạo phiên đăng nhập	

BẢNG 4.22: API đề xuất câu hỏi

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/suggestions	
Thuộc tính	num_suggestions	Số câu hỏi gợi ý (query)
Diễn giải	Trả về danh sách câu hỏi gợi ý dựa trên hội thoại gần nhất	

BẢNG 4.23: API lấy hội thoại gần nhất

Phương thức	**GET**
Endpoint	{API_DOMAIN}/api/latest-conversation
Diễn giải	Lấy hội thoại gần nhất có tin nhắn của người dùng

BẢNG 4.24: API health check

Phương thức	**GET**
Endpoint	{API_DOMAIN}/health

Diễn giải	Kiểm tra trạng thái dịch vụ & thời gian phản hồi

BẢNG 4.25: API tìm kiếm hội thoại

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/conversations/search	
Thuộc tính	query	Từ khóa tìm kiếm trong nội dung tin nhắn
	date_from	Tìm từ ngày (YYYY-MM-DD)
	date_to	Tìm đến ngày (YYYY-MM-DD)
	page	Trang hiện tại
	page_size	Số hội thoại mỗi trang
Diễn giải	Tìm kiếm hội thoại theo nội dung tin nhắn và thời gian	

BẢNG 4.26:API Admin - Liệt kê người dùng

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/admin/users	
Thuộc tính	page	Trang hiện tại
	page_page	Số người dùng mỗi trang
Diễn giải	[ADMIN] Liệt kê tất cả người dùng trong hệ thống	

BẢNG 4.27 API Admin - Tạo người dùng mới

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/admin/users	
Thuộc tính	email	Email người dùng
	password	Mật khẩu
	role	Vai trò (admin/student)

	metadata	Metadata bổ sung
Diễn giải	[ADMIN] Tạo người dùng mới trong hệ thống	

BẢNG 4.28: API Admin - Lấy thông tin người dùng

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/admin/users/{user_id}	
Thuộc tính	user_id	ID của người dùng
Diễn giải	[ADMIN] Lấy thông tin chi tiết của một người dùng	

BẢNG 4.29 API Admin - Cập nhật người dùng

Phương thức	**PUT**	
Endpoint	{API_DOMAIN}/api/admin/users/{user_id}	
Thuộc tính	user_id	ID của người dùng
	email	Email mới (tùy chọn)
	password	Mật khẩu mới (tùy chọn)
	role	Vai trò mới (tùy chọn)
	metadata	Metadata mới (tùy chọn)
Diễn giải	[ADMIN] Cập nhật thông tin người dùng	

BẢNG 4.30: API Admin - Xóa người dùng

Phương thức	**DELETE**	
Endpoint	{API_DOMAIN}/api/admin/users/{user_id}	
Thuộc tính	user_id	ID của người dùng
	hard	True = xóa vĩnh viễn, False = xóa tạm thời (query)
Diễn giải	[ADMIN] Xóa người dùng khỏi hệ thống	

BẢNG 4.31: API Admin - Cấm người dùng

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/admin/users/{user_id}/ban	
Thuộc tính	user_id	ID của người dùng
	duration	Thời gian cấm (1h, 24h, 7d)
	reason	Lý do cấm (tùy chọn)
Diễn giải	[ADMIN] Cấm người dùng trong khoảng thời gian	

BẢNG 4.32: API Admin - Bỏ cấm người dùng

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/admin/users/{user_id}/unban	
Thuộc tính	user_id	ID của người dùng
Diễn giải	[ADMIN] Bỏ cấm người dùng	

BẢNG 4.33: API Admin - Liệt kê hội thoại

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/admin/conversations	
Thuộc tính	page	Trang hiện tại
	per_page	Số conversation mỗi trang
	user_id	Lọc theo user_id (tùy chọn)
	date_from	Lọc từ ngày (tùy chọn)
	date_to	Lọc đến ngày (tùy chọn)
Diễn giải	[ADMIN] Lấy danh sách tất cả conversations trong hệ thống	

BẢNG 4.34: API Admin - Xem tin nhắn conversation

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/admin/conversations/{conversation_id}/messages	
Thuộc tính	conversation_id	ID của conversation
Diễn giải	[ADMIN] Xem chi tiết tin nhắn trong một conversation	

BẢNG 4.35: API Admin - Tìm kiếm tin nhắn

Phương thức	**POST**	
Endpoint	{API_DOMAIN}/api/admin/messages/search	
Thuộc tính	query	Từ khóa tìm kiếm
	conversation_id	ID của conversation
	user_id	Lọc theo user (tùy chọn)
	date_from	Tìm từ ngày (tùy chọn)
	date_to	Tìm đến ngày (tùy chọn)
	page	Trang hiện tại
	per_page	Số tin nhắn mỗi trang
Diễn giải	[ADMIN] Tìm kiếm tin nhắn trong toàn hệ thống	

BẢNG 4.36:API Admin - Xóa conversation

Phương thức	**DELETE**	
Endpoint	{API_DOMAIN}/api/admin/conversations/{conversation_id}	
Thuộc tính	conversation_id	ID của conversation
Diễn giải	[ADMIN] Xóa conversation và tất cả messages liên quan	

BẢNG 4.37:API Admin - Thống kê files

Phương thức	**GET**
Endpoint	{API_DOMAIN}/api/admin/files/stats
Diễn giải	[ADMIN] Lấy thống kê về files trong hệ thống

BẢNG 4.38: API Admin - Thống kê conversations

Phương thức	**GET**	
Endpoint	{API_DOMAIN}/api/admin/conversations/stats	
Thuộc tính	days	Số ngày thống kê (query, mặc định 7)
Diễn giải	[ADMIN] Lấy thống kê về conversations và messages	

- Module Xử lý Tài liệu (Data Ingestion):
  - src/document_processor.py (DocumentProcessor):
    - Chịu trách nhiệm tải và trích xuất nội dung từ nhiều định dạng tệp khác nhau (PDF, DOCX, TXT, SQL) sử dụng các loaders từ LangChain.
    - Thực hiện tiền xử lý văn bản: làm sạch, chuẩn hóa.
    - Phân đoạn văn bản (chunking) thành các đoạn nhỏ hơn, có ý nghĩa bằng RecursiveCharacterTextSplitter, với kích thước và độ chồng lấp có thể cấu hình. Hỗ trợ cả phân đoạn theo cấu trúc (ví dụ: nhận diện tiêu đề).
    - Làm giàu metadata cho từng chunk (ví dụ: nguồn, số trang, loại chunk).
  - src/embedding.py (EmbeddingModel):
    - Sử dụng mô hình Sentence Transformer (intfloat/multilingual-e5-small) để tạo vector nhúng cho từng đoạn văn bản đã được xử lý.
  - src/vector_store.py (VectorStore):
    - Tương tác với Qdrant để lưu trữ các vector nhúng cùng với metadata của collection.
    - Cung cấp các phương thức để thêm, xóa và tìm kiếm vector.
  - src/supabase/files_manager.py (FilesManager):

        - Lưu trữ và quản lý thông tin metadata của các tệp tài liệu (tên file, ID, user ID, trạng thái xử lý, v.v.) trong bảng document_files trên Supabase.
- Module Hỏi đáp (RAG Core):
    - src/rag.py (AdvancedDatabaseRAG):
        - Là module trung tâm điều phối toàn bộ luồng RAG.
        - Tích hợp các thành phần con để xử lý câu hỏi từ đầu đến cuối.
        - Phương thức query_with_sources_streaming là điểm vào chính cho việc xử lý câu hỏi.
    - src/query_handler.py (QueryHandler):
        - Sử dụng LLM (Gemini) để mở rộng câu hỏi (giải quyết đồng tham chiếu dựa trên lịch sử hội thoại) và phân loại câu hỏi thành các loại: question_from_document, sql_code_task, realtime_question, other_question.
    - src/search.py (SearchManager):
        - Thực hiện tìm kiếm ngữ nghĩa (semantic search) trên Qdrant bằng cách gọi VectorStore sau khi câu hỏi đã được nhúng.
        - Áp dụng xếp hạng lại (reranking) cho các kết quả tìm kiếm sử dụng mô hình Cross-Encoder (cross-encoder/ms-marco-MiniLM-L-6-v2) để chọn ra các đoạn ngữ cảnh liên quan nhất.
    - src/prompt_manager.py (PromptManager):
        - Quản lý và tạo các prompt hoàn chỉnh cho LLM.
        - Kết hợp câu hỏi của người dùng, lịch sử hội thoại và các đoạn ngữ cảnh đã được truy xuất (kèm thông tin nguồn) vào các template prompt được định nghĩa sẵn (ví dụ: tutor_mode_system_prompt, sql_code_task_prompt, related_questions_prompt).
    - src/llm.py (GeminiLLM):
        - Giao tiếp với Google Gemini API để sinh câu trả lời, giải thích mã SQL, hoặc tạo gợi ý câu hỏi.
        - Hỗ trợ streaming để trả về kết quả từng phần.
        - Triển khai cơ chế thử lại (retry) khi gặp lỗi API.
    - src/tools/Google_Search.py:

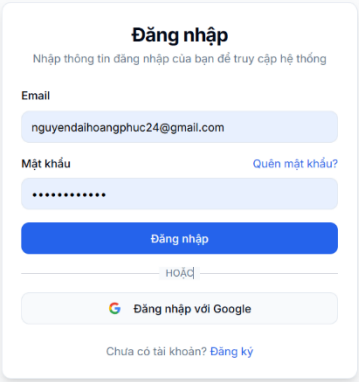
        - Cung cấp chức năng tìm kiếm thông tin trên web, được sử dụng cho các câu hỏi realtime_question hoặc làm fallback khi không tìm thấy thông tin trong tài liệu.
- Module Quản lý Hội thoại:
    - src/supabase/conversation_manager.py (SupabaseConversationManager):
        - Lưu trữ và truy xuất lịch sử các cuộc hội thoại (bao gồm câu hỏi người dùng và câu trả lời của chatbot, cùng metadata nguồn) vào các bảng conversations và messages trên Supabase.
        - Cung cấp các hàm để tạo, lấy danh sách, lấy chi tiết và xóa hội thoại.
- Module Gợi ý Câu hỏi:
    - src/suggestion_manager.py (SuggestionManager):
        - Tạo ra các câu hỏi gợi ý dựa trên lịch sử hội thoại gần nhất hoặc một danh sách mặc định.
        - Sử dụng LLM (thông qua PromptManager và GeminiLLM) để phân tích ngữ cảnh và đề xuất các câu hỏi liên quan.

4.3 Xây dựng Frontend:

Frontend của hệ thống được xây dựng bằng Next.js, React, TypeScript và Tailwind CSS, tập trung vào việc cung cấp trải nghiệm người dùng mượt mà, trực quan và có sự phân quyền rõ ràng giữa người dùng thông thường và quản trị viên.

4.3.1 Giao diện người dùng (Sinh viên):

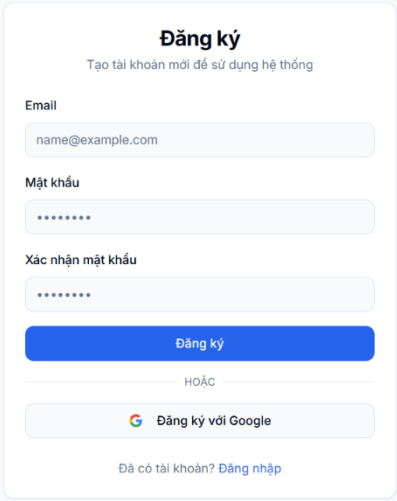
- Giao diện đăng nhập: Cho phép người dùng có thể đăng nhập vào hệ thống khi đã có tài khoản.



HÌNH 4.1: Trang đăng nhập tài khoản

(Nguồn: Tự xây dựng)

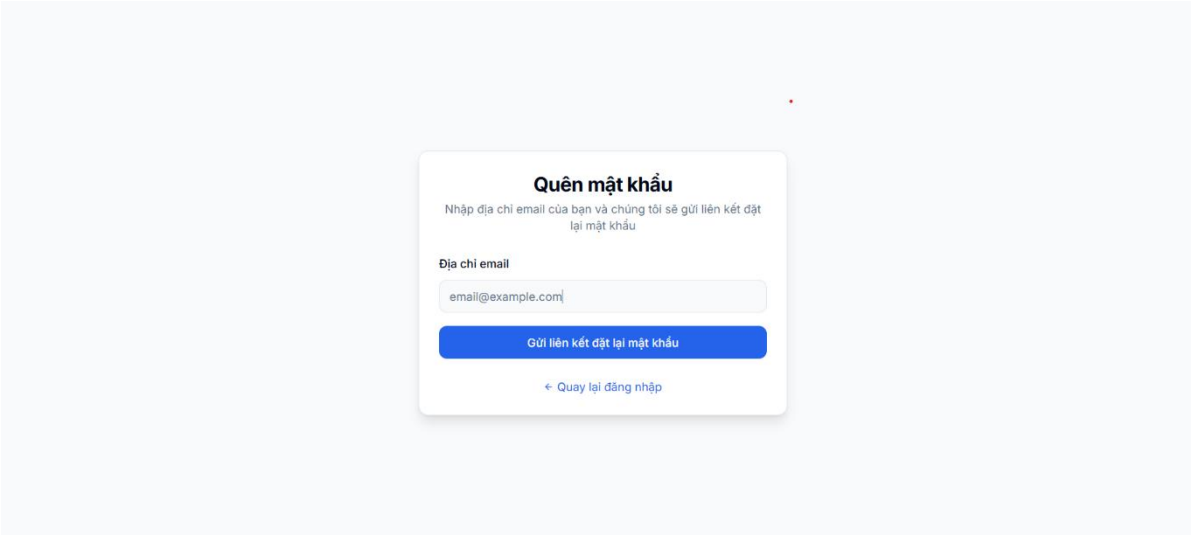
- Giao diện đăng ký: Cho phép người dùng chưa có tài khoản có thể đăng ký khi chưa có tài khoản.



HÌNH 4.2: Trang đăng ký tài khoản

(Nguồn: Tự xây dựng)

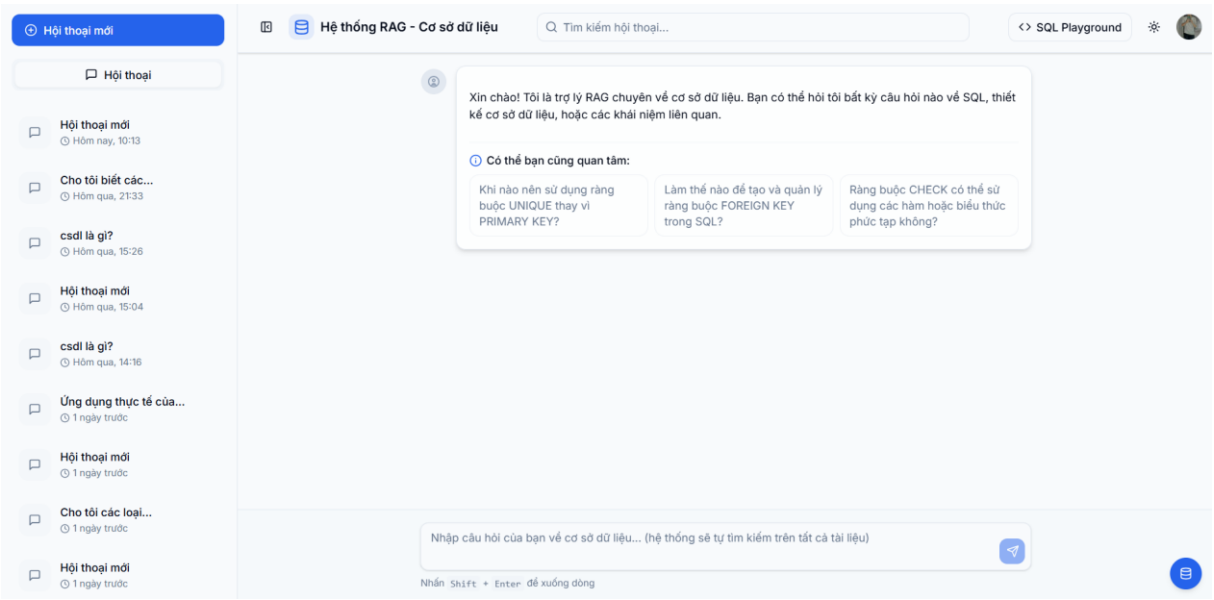
- Giao diện quên mật khẩu: Cho phép người dùng có thể đặt lại mật khẩu của mình khi quên mật khẩu.



HÌNH 4.3: Trang quên mật khẩu

(Nguồn: Tự xây dựng)

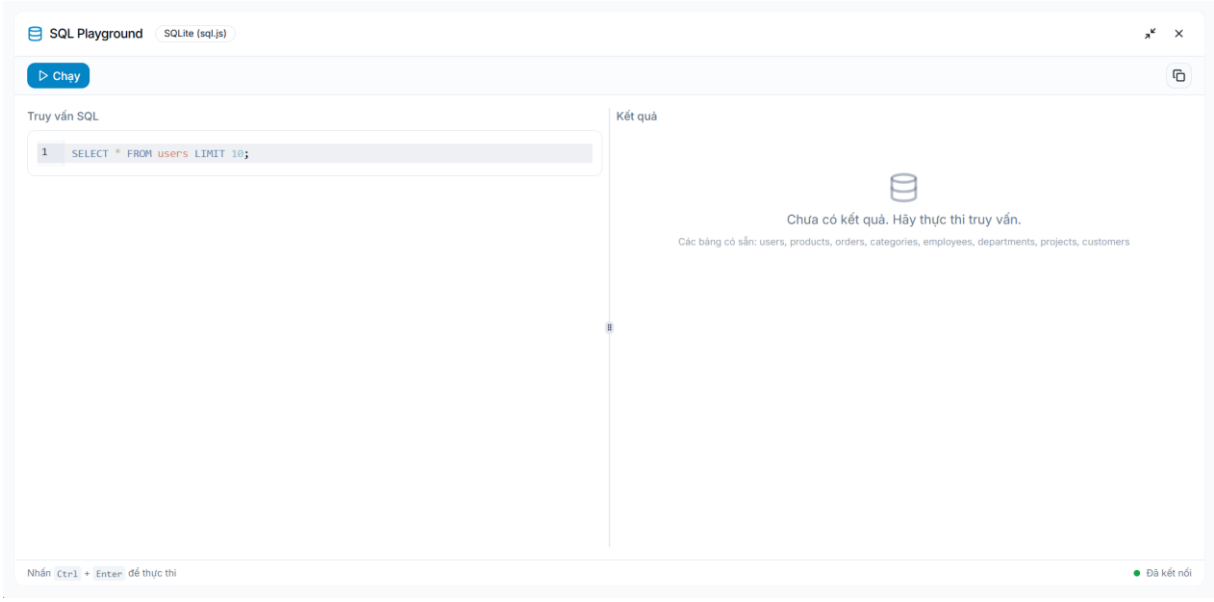
- Giao diện trang chính: Ở đây người dùng có thể xem lịch sử các cuộc hội thoại của mình trước đó. Hệ thống cũng cung cấp những câu hỏi gợi ý dựa vào lịch sử chat của người dùng ở lần trước đó. Và người dùng có thể chat với hệ thống ngay lập tức. Bên trên có ô nhập tìm kiếm lịch sử cho phép người dùng có thể tìm lại các đoạn lịch sử chat của mình.



HÌNH 4.4: Trang chủ

(Nguồn: Tự xây dựng)

- Giao diện SQL Playground: Cung cấp một môi trường thực hành đơn giản để người dùng có thể chạy thử các câu lệnh SQL cơ bản với các bộ dữ liệu mẫu.

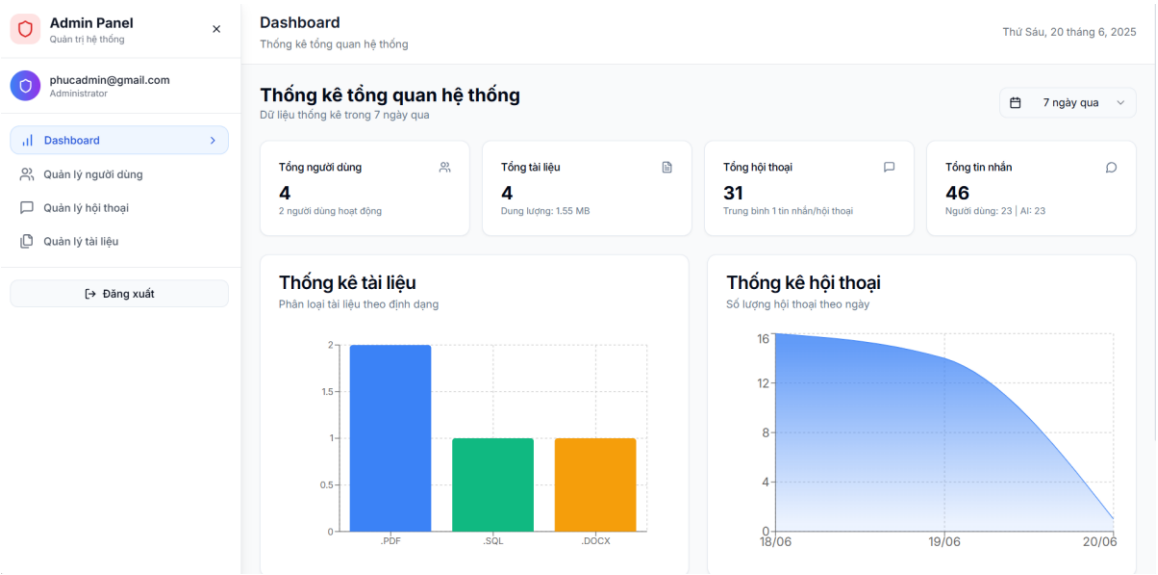


HÌNH 4.5: Trang SQL Playground

(Nguồn: Tự xây dựng)

4.3.2 Giao diện dành cho Quản trị viên (Admin):

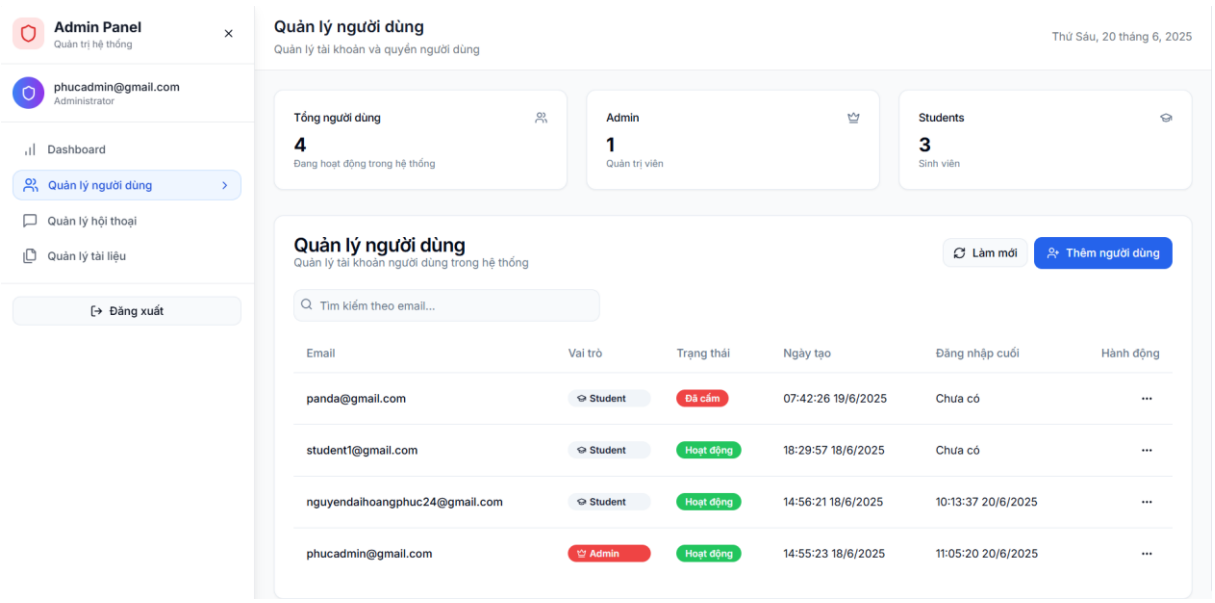
- Hiển thị các số liệu thống kê tổng quan (tổng số người dùng, tổng số tài liệu, tổng số cuộc hội thoại).



HÌNH 4.6: Trang dashboard chính của admin

(Nguồn: Tự xây dựng)

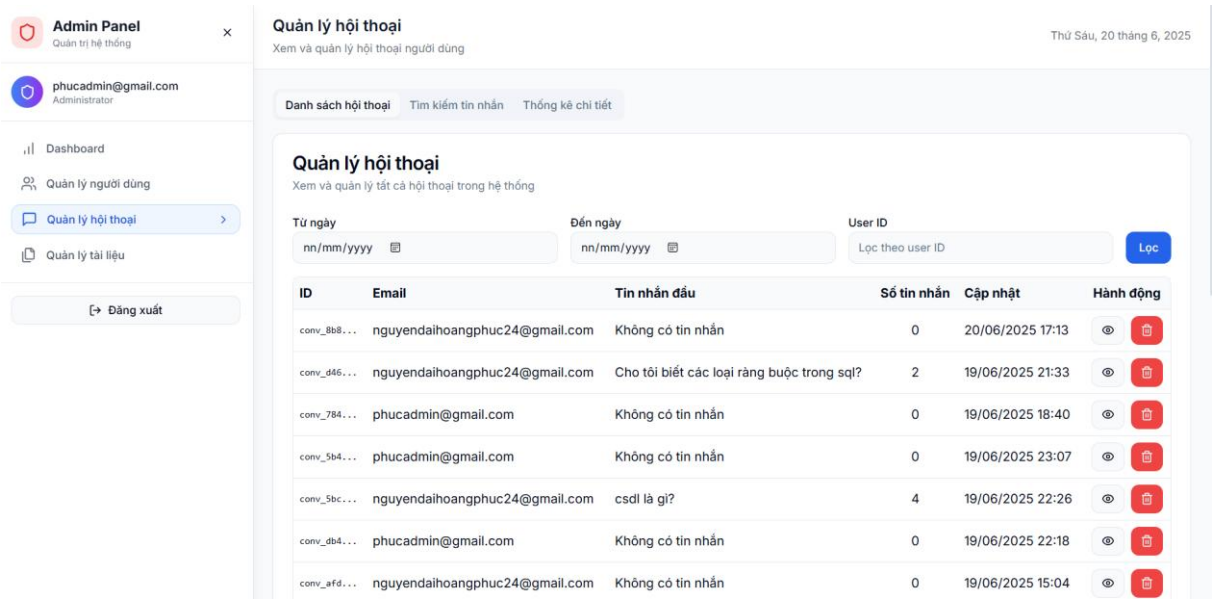
- Trang quản lý người dùng: Mô tả giao diện cho phép Admin xem danh sách người dùng, tìm kiếm, xem chi tiết, cập nhật vai trò (role) và có thể thực hiện các hành động như cấm (ban) người dùng.



HÌNH 4.7: Trang quản lý người dùng của admin

(Nguồn: Tự xây dựng)

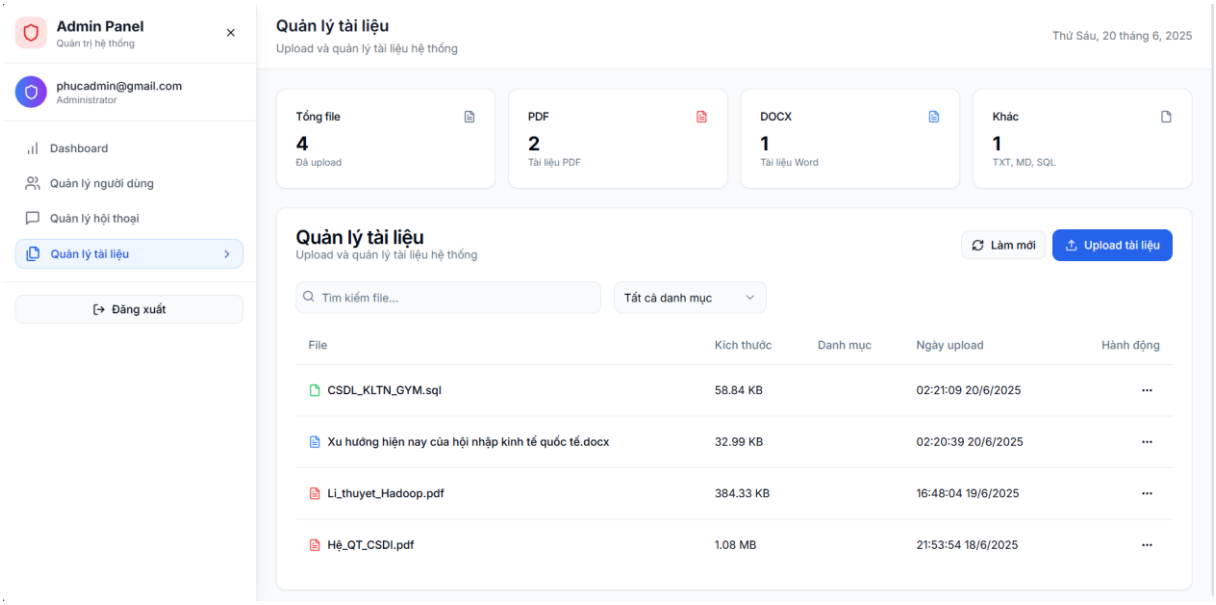
- Trang quản lý hội thoại: Mô tả giao diện cho phép Admin xem lại lịch sử hội thoại của tất cả người dùng, phục vụ cho việc gỡ lỗi và phân tích hành vi người dùng.



HÌNH 4.8: Trang quản lý lịch sử hội thoại của admin

(Nguồn: Tự xây dựng)

- Trang quản lý tài liệu: Mô tả giao diện cho phép Admin tải lên tài liệu mới, xem danh sách các tài liệu đã có, và xóa tài liệu. Khi xóa, hệ thống sẽ xóa cả file vật lý, metadata trong Supabase và các vector tương ứng trong Qdrant.



HÌNH 4.9: Trang quản lý tài liệu của admin

(Nguồn: Tự xây dựng)

4.3.3 Logic tương tác và quản lý trạng thái:

- Logic Tương tác với Backend:
  - Toàn bộ logic gọi API backend được tập trung trong thư mục frontend/lib/api.ts.
  - Sử dụng các hàm bất đồng bộ (async/await) để gọi các API RESTful đã được định nghĩa ở Backend.
  - Triển khai việc nhận dữ liệu streaming từ API /api/ask/stream bằng cách sử dụng EventSource (Server-Sent Events), cho phép câu trả lời của chatbot được hiển thị dần trên giao diện.
  - Quản lý trạng thái của ứng dụng (ví dụ: nội dung chat, lịch sử hội thoại) bằng các React Hooks (useState, useEffect, useContext) hoặc các custom hooks trong frontend/hooks/.
- Quản lý Trạng thái và Định tuyến:
  - Trạng thái của các component được quản lý chủ yếu bằng các React Hooks cơ bản như useState và useEffect.
  - Đối với trạng thái xác thực người dùng toàn cục, hệ thống sử dụng React Context API kết hợp với một custom hook là useAuth (tại frontend/hooks/useAuth.tsx). Giải pháp này cho phép mọi component

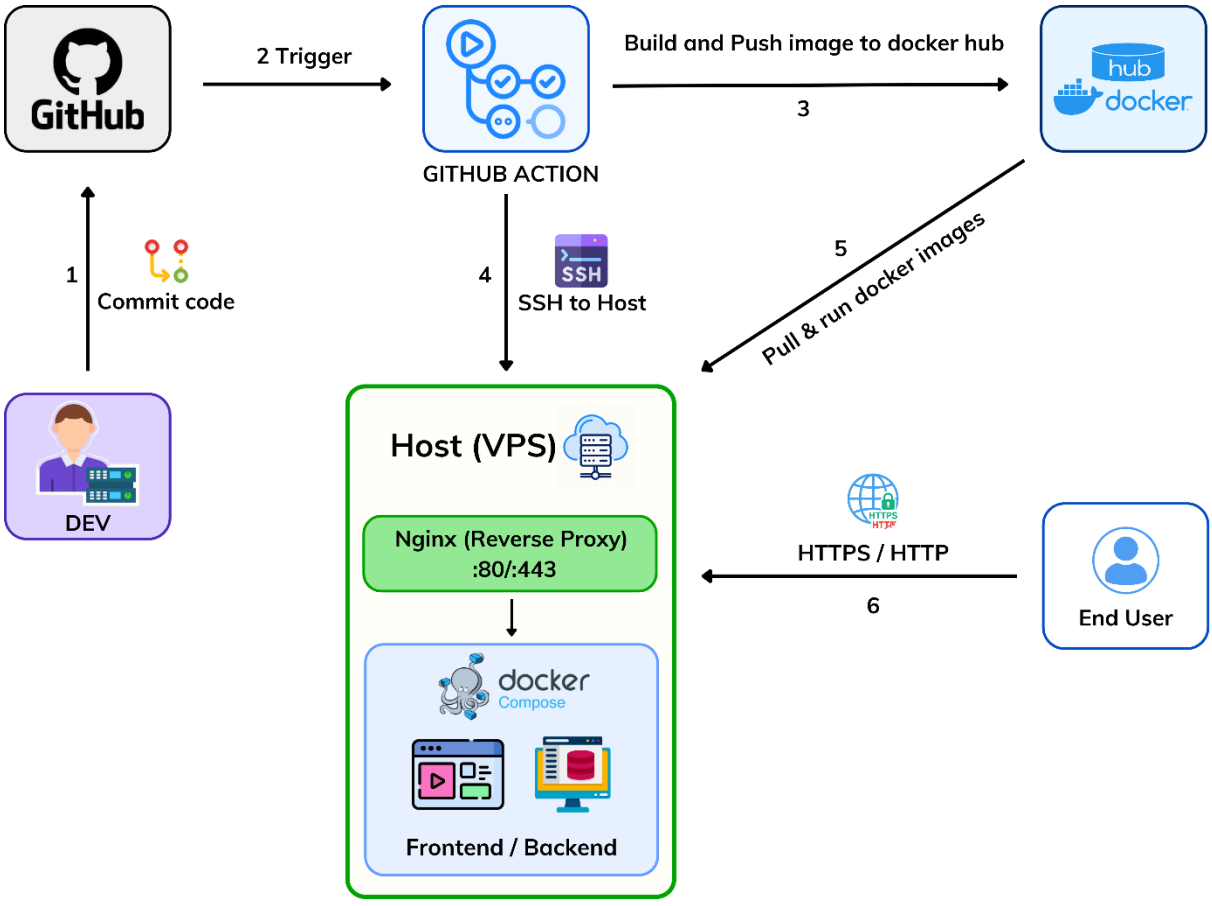
trong ứng dụng đều có thể truy cập thông tin người dùng và trạng thái đăng nhập một cách nhất quán.

- Token xác thực của người dùng được lưu trữ an toàn trong LocalStorage của trình duyệt để duy trì phiên đăng nhập giữa các lần tải lại trang.

- Xác thực:
  - Tích hợp với Supabase Auth client-side để quản lý phiên đăng nhập của người dùng. Token xác thực được gửi kèm trong các yêu cầu API tới Backend.

4.4 Tích hợp hệ thống và Đóng gói/Triển khai:

Dưới đây minh họa quy trình triển khai tự động hóa (CI/CD - Continuous Integration/Continuous Deployment) của hệ thống chatbot RAG, từ giai đoạn phát triển đến khi người dùng cuối có thể truy cập. Quy trình này đảm bảo việc cập nhật và triển khai phần mềm được thực hiện một cách nhanh chóng, nhất quán và giảm thiểu lỗi do con người.



SƠ ĐỒ 4.2: Sơ đồ triển khai hệ thống với CI/CD

(Nguồn: Tự xây dựng theo thiết kế hệ thống)

Quy trình triển khai được thực hiện qua các bước chính sau:

1. Commit Code: Lập trình viên (DEV) thực hiện các thay đổi hoặc bổ sung tính năng và commit mã nguồn lên kho lưu trữ GitHub của dự án.
2. Trigger GitHub Actions: Mỗi khi có một commit mới được đẩy lên nhánh chính (hoặc các nhánh được cấu hình khác), một quy trình tự động trong GitHub Actions sẽ được kích hoạt.
3. Build and Push Docker Images: GitHub Actions thực hiện các bước build Docker images cho cả ứng dụng Frontend và Backend dựa trên các Dockerfile đã được định nghĩa sẵn trong mã nguồn. Sau khi build thành công, các Docker images này sẽ được đẩy (push) lên một Docker Registry tập trung (ví dụ: Docker Hub hoặc một private registry).
4. SSH to Host: Sau khi các images đã được lưu trữ trên Docker Registry, GitHub Actions sử dụng kết nối SSH an toàn để truy cập vào máy chủ triển khai (Host VPS).
5. Pull & Run Docker Images: Trên Host VPS, GitHub Actions thực thi các lệnh để:
   - Dừng và gỡ bỏ các container cũ đang chạy (nếu có).
   - Kéo (pull) các phiên bản Docker images mới nhất của Frontend và Backend từ Docker Registry.
   - Sử dụng Docker Compose để khởi chạy lại các container Frontend và Backend với cấu hình đã được định nghĩa trong tệp docker-compose.yml.
6. End User Access: Người dùng cuối (End User) truy cập vào hệ thống thông qua trình duyệt web bằng giao thức HTTPS/HTTP. Yêu cầu của người dùng sẽ được Nginx, đóng vai trò là một Reverse Proxy trên Host VPS, tiếp nhận và điều hướng đến container Frontend hoặc Backend tương ứng đang chạy trong Docker. Nginx cũng chịu trách nhiệm xử lý SSL/TLS cho các kết nối HTTPS.

- Tích hợp Frontend và Backend:
  - Frontend và Backend giao tiếp với nhau thông qua các API RESTful (và SSE cho streaming) đã được định nghĩa.
  - Backend FastAPI chạy trên một cổng (ví dụ: 8000) và Frontend Next.js chạy trên một cổng khác (ví dụ: 3000) trong quá trình phát triển. Proxy

được cấu hình trong Next.js (thông qua next.config.mjs nếu cần) hoặc CORS được cấu hình ở Backend để cho phép giao tiếp.

- Biến môi trường (trong các tệp .env) được sử dụng để cấu hình các URL API, khóa API cho Gemini, Supabase, Qdrant, v.v.

- Nghiên cứu Đóng gói và Triển khai (Docker/Docker Compose):
  - Để đơn giản hóa quá trình triển khai và đảm bảo tính nhất quán giữa các môi trường, việc sử dụng Docker và Docker Compose đã được nghiên cứu.
  - Docker:
    - Có thể tạo Dockerfile cho Backend (Python/FastAPI) để đóng gói ứng dụng cùng các phụ thuộc của nó vào một image.
    - Có thể tạo Dockerfile cho Frontend (Next.js) để build ứng dụng và phục vụ nó (ví dụ: bằng Nginx hoặc Node.js server).
  - Docker Compose: giúp dễ dàng khởi chạy toàn bộ hệ thống chỉ bằng một lệnh, quản lý network giữa các container và các biến môi trường. Một tệp docker-compose.yml có thể được sử dụng để định nghĩa và quản lý các services của hệ thống, bao gồm:
    - Service Backend.
    - Service Frontend.

CHƯƠNG 5 ĐÁNH GIÁ VÀ KẾT QUẢ

5.1 Xây dựng bộ câu hỏi kiểm thử:

Để đánh giá một cách có hệ thống, một bộ câu hỏi kiểm thử đã được xây dựng, bao gồm 23 câu hỏi. Các câu hỏi này được thiết kế để bao quát các chủ đề đa dạng trong môn Cơ sở dữ liệu và kiểm tra các khía cạnh khác nhau của chatbot. Các câu hỏi sẽ được lựa chọn theo các tiêu chí sau đây:

- Đa dạng chủ đề:
  - CSDL và DBMS là gì? Phân biệt chúng." (Khái niệm cơ bản)
  - Mô hình dữ liệu là gì? Mô tả mô hình quan hệ." (Mô hình dữ liệu)
  - Lệnh SELECT trong SQL: cú pháp, chức năng, các mệnh đề thường dùng?" (Ngôn ngữ SQL)
  - Mục đích chuẩn hóa dữ liệu? Trình bày 1NF và 2NF?" (Thiết kế CSDL và Chuẩn hóa)
  - Giao dịch CSDL là gì? Các tính chất ACID?" (Giao dịch và Quản trị)
- Đa dạng loại câu hỏi:
  - Câu hỏi về các khái niệm cơ bản.
  - Câu hỏi về mô hình dữ liệu.
  - Câu hỏi về ngôn ngữ SQL.
  - Câu hỏi về thiết kế CSDL và Chuẩn hóa.
  - Câu hỏi về Giao dịch và Quản trị.
- Độ khó khác nhau: Từ dễ đến trung bình, có thể có một vài câu hỏi khó hơn để thử thách hệ thống.
- Dự kiến có trong tài liệu: Các câu hỏi được xây dựng dựa trên khả năng có thể được trả lời từ nội dung của tài liệu kiểm thử.

5.2 Phương pháp đánh giá:

Với mỗi câu hỏi trong bộ kiểm thử, hệ thống chatbot được chạy và các thông số sau được ghi nhận:

- Độ liên quan của câu trả lời:
  - Thang điểm:
    - 0 - Không liên quan: Câu trả lời hoàn toàn lạc đề, không giải quyết được ý chính của câu hỏi.

        - 1 - Liên quan một phần: Câu trả lời có đề cập đến chủ đề của câu hỏi nhưng không đầy đủ, thiếu ý chính, hoặc có phần thông tin không chính xác/không cần thiết.
        - 2 - Hoàn toàn liên quan: Câu trả lời đi thẳng vào vấn đề, giải quyết đầy đủ các khía cạnh chính của câu hỏi một cách rõ ràng.
    - Cách thực hiện: Việc đánh giá được thực hiện dựa trên sự đối chiếu cẩn thận với nội dung tài liệu gốc và kiến thức chuyên môn về môn học. Để đảm bảo tính nhất quán, một bộ quy tắc chấm điểm chi tiết đã được áp dụng cho tất cả các câu hỏi.
- Kiểm tra tính đúng đắn dựa trên nguồn:
    - Áp dụng: Thực hiện với những câu trả lời được đánh giá từ 1 điểm trở lên về độ liên quan.
    - Tiêu chí:
        - Có: Nội dung chính của câu trả lời khớp với thông tin trong (các) nguồn được trích dẫn.
        - Không: Nội dung chính của câu trả lời mâu thuẫn, không tìm thấy, hoặc diễn giải sai lệch thông tin từ (các) nguồn được trích dẫn.
    - Cách thực hiện: Người đánh giá sẽ xem xét các nguồn mà chatbot trích dẫn và đối chiếu với nội dung câu trả lời.
- Tốc độ phản hồi (Response Time):
    - Cách thực hiện: Ghi lại thời gian từ lúc người dùng gửi câu hỏi cho đến khi chatbot hiển thị toàn bộ câu trả lời.
    - Đơn vị: Giây (s).

5.3 Phân tích kết quả:

Sau khi thực hiện đánh giá thì tôi thu được kết quả như sau:

BẢNG 5.1: Kết quả đánh giá định lượng chi tiết theo từng câu hỏi

STT	Câu hỏi	Điểm Liên quan (0-2)	Khớp nguồn (Có/Không)	Tốc độ (s)
Phần 1: Khái niệm cơ bản				
1	CSDL và DBMS là gì? Phân biệt chúng.	2	Có	6.6

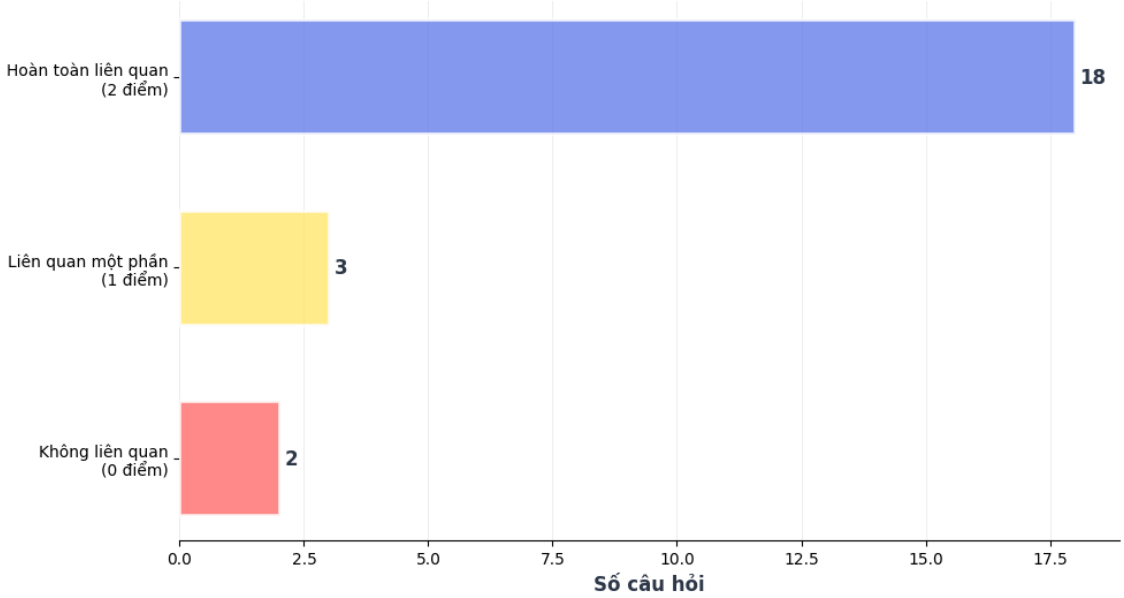
STT	Câu hỏi	Điểm Liên quan (0-2)	Khớp nguồn (Có/Không)	Tốc độ (s)
2	Ưu điểm của DBMS so với lưu trữ tệp truyền thống?	2	Có	4.2
3	Các thành phần chính của môi trường DBMS?	1	Không	4.5
4	Các mức trừu tượng CSDL và mục đích của chúng?	0	Không	5
5	Độc lập dữ liệu là gì? Phân biệt độc lập vật lý và logic.	2	Có	5.9
Phần 2: Mô hình dữ liệu				
6	Mô hình dữ liệu là gì? Mô tả mô hình quan hệ.	2	Có	4
7	Giải thích các khái niệm: Quan hệ, Thuộc tính, Bộ, Miền giá trị, Khóa chính, Khóa ngoại?	2	Có	5.2
8	Ràng buộc toàn vẹn là gì? Kể tên các loại thường gặp?	2	Có	4.9
9	Các mô hình dữ liệu khác ngoài mô hình quan hệ?	2	Có	4.3
10	Ưu nhược điểm của mô hình quan hệ?	2	Không	5.1
Phần 3: Ngôn ngữ SQL				
11	SQL là gì? Các nhóm lệnh chính của nó?	2	Có	6.2
12	Lệnh SELECT trong SQL: cú pháp, chức năng, các mệnh đề thường dùng?	2	Có	4.5
13	Các lệnh DML khác: INSERT, UPDATE, DELETE (cú pháp, chức năng)?	2	Có	5

STT	Câu hỏi	Điểm Liên quan (0-2)	Khớp nguồn (Có/Không)	Tốc độ (s)
14	Hàm gộp SQL là gì? Kể tên và ví dụ với GROUP BY.	2	Có	5.4
15	Phép JOIN trong SQL: mục đích, các loại phổ biến và khác biệt?	2	Có	4.8
Phần 4: Thiết kế CSDL và Chuẩn hóa				
16	Mục đích chuẩn hóa dữ liệu? Trình bày 1NF và 2NF?	2	Có	5.3
17	Phụ thuộc hàm là gì? Cho ví dụ?	2	Có	4.8
18	Trình bày 3NF và BCNF. So sánh sự khác biệt.	2	Có	4.5
19	Các bước chính trong thiết kế CSDL? Vai trò của mô hình ERM?	0	Không	4.1
Phần 5: Giao dịch và Quản trị				
20	Giao dịch CSDL là gì? Các tính chất ACID?	2	Có	4.3
21	Các vai trò người dùng chính trong DBMS?	1	Có	6.5
22	Sao lưu và phục hồi CSDL: khái niệm, tầm quan trọng, phương pháp?	1	Không	5.3
23	An toàn và bảo mật CSDL? Cấp quyền và xác thực trong DBMS là gì?	2	Có	4.4

(Nguồn: Tự tổng hợp từ kết quả kiểm thử hệ thống)

- Phân tích kết quả định lượng: Dựa trên kết quả chi tiết được trình bày trong Bảng 5.1, các số liệu tổng hợp về hiệu năng của hệ thống chatbot được phân tích như sau:
  a. Độ liên quan của câu trả lời: Trong tổng số 23 câu hỏi kiểm thử:

- Số câu trả lời đạt 2 điểm (Hoàn toàn liên quan): 18 câu (chiếm 18/23 ≈ 78.26%)
- Số câu trả lời đạt 1 điểm (Liên quan một phần): 3 câu (chiếm 3/23 ≈ 13.04%)
- Số câu trả lời đạt 0 điểm (Không liên quan): 2 câu (chiếm 2/23 ≈ 8.70%)



HÌNH 5.1 Biểu đồ phân bố điểm liên quan của câu trả lời

(Nguồn: Tự tổng hợp từ số liệu thống kê đánh giá định lượng)

➔ Điểm liên quan trung bình của hệ thống đạt: (18 * 2 + 3 * 1 + 2 * 0) / 23 ≈ 1.696 trên thang điểm 2.

- Nhận xét: Kết quả cho thấy hệ thống có khả năng hiểu và cung cấp câu trả lời liên quan ở mức độ khá tốt. Như Hình 5.1 minh họa, tỷ lệ lớn (78.26%) câu trả lời được đánh giá là "Hoàn toàn liên quan", cho thấy chatbot đã nắm bắt đúng trọng tâm của phần lớn các câu hỏi. Điều này phản ánh hiệu quả của kiến trúc RAG trong việc truy xuất và sử dụng ngữ cảnh từ tài liệu "Hệ_QT_CSDl.pdf". Tuy nhiên, vẫn còn khoảng 21.74% câu trả lời chưa đạt mức độ liên quan tối ưu (bao gồm 13.04% "Liên quan một phần" và 8.70% "Không liên quan").

- Các câu hỏi chatbot trả lời "Liên quan một phần" (điểm 1) bao gồm:
  - Câu 3: "Các thành phần chính của môi trường DBMS?"
  - Câu 21: "Các vai trò người dùng chính trong DBMS?"

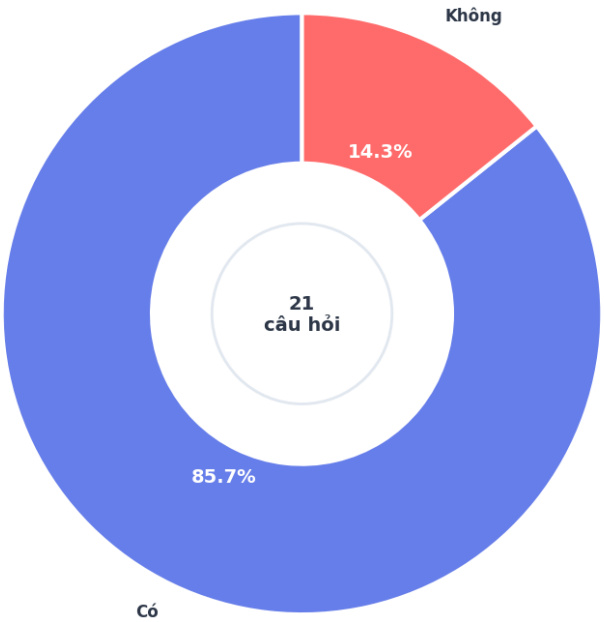
- Câu 22: "Sao lưu và phục hồi CSDL: khái niệm, tầm quan trọng, phương pháp?" Nguyên nhân có thể do chatbot chỉ truy xuất được một phần thông tin liên quan từ tài liệu, hoặc LLM chưa tổng hợp đầy đủ các khía cạnh của câu hỏi dựa trên ngữ cảnh được cung cấp. Ví dụ, với câu 3, chatbot có thể chỉ liệt kê được một vài thành phần thay vì tất cả các thành phần được mô tả trong tài liệu.

- Các câu hỏi chatbot trả lời "Không liên quan" (điểm 0) là:
  - Câu 4: "Các mức trừu tượng CSDL và mục đích của chúng?"
  - Câu 19: "Các bước chính trong thiết kế CSDL? Vai trò của mô hình ERM?" Đây là những trường hợp hệ thống chưa xử lý tốt, có thể do thông tin về các chủ đề này không rõ ràng, không có trong tài liệu, hoặc do cơ chế truy xuất thông tin chưa hiệu quả đối với loại câu hỏi này. Ví dụ, nếu tài liệu "Hệ_QT_CSDl.pdf" không trình bày chi tiết về các bước thiết kế CSDL (câu 19), chatbot sẽ khó có thể đưa ra câu trả lời liên quan. Cần xem xét lại nội dung tài liệu hoặc cải thiện khả năng truy xuất của hệ thống cho các chủ đề này.

b. Tính đúng đắn dựa trên nguồn (Khớp nguồn): Trong số 21 câu hỏi có điểm liên quan từ 1 trở lên (18 câu điểm 2 và 3 câu điểm 1), kết quả kiểm tra khớp nguồn như sau:
   - Số câu trả lời Khớp nguồn (Có): 18 câu.
   - Số câu trả lời Không khớp nguồn (Không): 3 câu.

   ➔ Tỷ lệ khớp nguồn (trong số các câu có liên quan và được đánh giá) là: 18/21 ≈ 85.7%

   ➔ Tỷ lệ không khớp nguồn là: 3/21 ≈ 14.3%.



HÌNH 5.2: Biểu đồ tỷ lệ khớp nguồn của câu trả lời (cho các câu có độ liên quan >= 1)

(Nguồn: Tự tổng hợp từ số liệu thống kê đánh giá định lượng)

- Nhận xét: Kết quả về tính đúng đắn dựa trên nguồn là rất khả quan, với 85.7% số câu trả lời (có độ liên quan từ 1 điểm trở lên) được xác định là "Khớp nguồn" với tài liệu "Hệ_QT_CSDl.pdf" (Như Hình 5.2 minh họa). Điều này cho thấy hệ thống chatbot RAG đã hoạt động hiệu quả trong việc trích xuất và sử dụng thông tin từ tài liệu được cung cấp để tạo ra câu trả lời, hạn chế đáng kể tình trạng "ảo giác" (hallucination) hay bịa đặt thông tin. Đây là một ưu điểm quan trọng, khẳng định độ tin cậy của hệ thống trong việc cung cấp kiến thức dựa trên nguồn tài liệu đã được huấn luyện.
- Tuy nhiên, vẫn còn 3 trường hợp (chiếm 14.3%) được đánh giá là "Không khớp nguồn", bao gồm:
  - Câu 3: "Các thành phần chính của môi trường DBMS?" (Điểm liên quan: 1)
  - Câu 10: "Ưu nhược điểm của mô hình quan hệ?" (Điểm liên quan: 2)
  - Câu 22: "Sao lưu và phục hồi CSDL: khái niệm, tầm quan trọng, phương pháp?" (Điểm liên quan: 1)
- Việc các câu trả lời này, dù có độ liên quan nhất định (đặc biệt là câu 10 với điểm liên quan là 2), nhưng lại không hoàn toàn khớp với nội dung

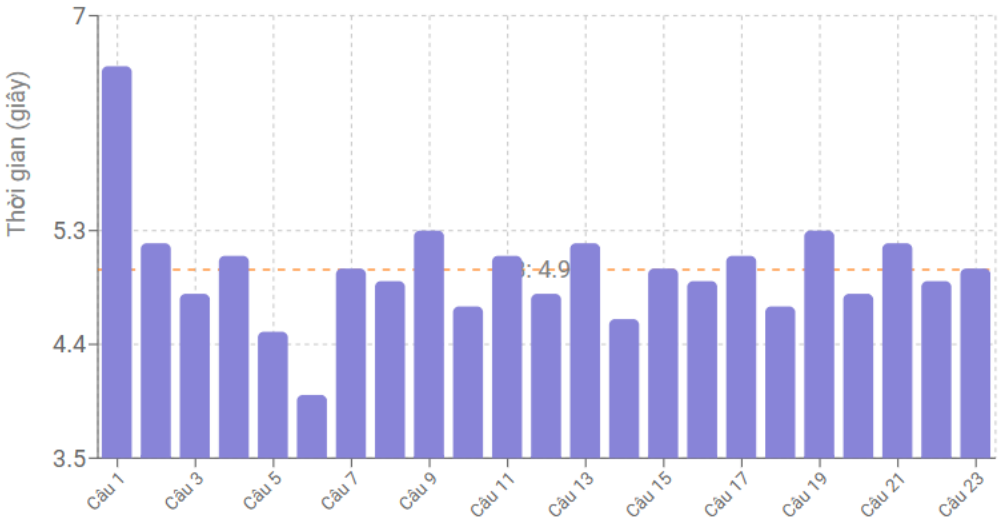
hoặc cách diễn đạt trong tài liệu gốc cho thấy một số thách thức. Nguyên nhân có thể đến từ việc:

1. LLM diễn giải hoặc tổng hợp thông tin theo cách riêng: Dù ngữ cảnh được cung cấp từ tài liệu, LLM có thể đã mở rộng, tóm tắt hoặc diễn giải thông tin theo một cách khác biệt so với nguyên văn, dẫn đến sự không khớp khi đối chiếu chi tiết.
2. Ngữ cảnh truy xuất chưa tối ưu: Đoạn văn bản được truy xuất từ tài liệu có thể chưa phải là phần chứa thông tin đầy đủ hoặc chính xác nhất để trả lời câu hỏi, khiến LLM phải dựa vào kiến thức nền hoặc suy luận nhiều hơn.
3. Sự phức tạp của câu hỏi hoặc nội dung tài liệu: Đối với những chủ đề có nhiều khía cạnh hoặc được trình bày phức tạp trong tài liệu, việc chatbot tổng hợp lại một cách hoàn toàn khớp với nguồn có thể khó khăn hơn.

- Những trường hợp "Không khớp nguồn" này chỉ ra rằng mặc dù hệ thống RAG đã giảm thiểu đáng kể việc bịa đặt thông tin, việc đảm bảo 100% câu trả lời phản ánh chính xác và đầy đủ từng chi tiết trong nguồn vẫn là một mục tiêu cần tiếp tục cải thiện. Việc phân tích kỹ hơn các đoạn trích xuất (context) mà hệ thống sử dụng cho các câu trả lời này có thể cung cấp thêm thông tin để tối ưu hóa cơ chế truy xuất hoặc cách LLM được prompt để sử dụng ngữ cảnh.

c. Tốc độ phản hồi: Dữ liệu về tốc độ phản hồi của hệ thống cho từng câu hỏi được ghi nhận trong Bảng 5.1. Các thông số tổng hợp về tốc độ phản hồi như sau:

- Tốc độ phản hồi trung bình: (Tổng thời gian của 23 câu) / 23 ≈ 4.991 giây/câu.
- Tốc độ phản hồi nhanh nhất: 4.0 giây (Câu 6: "Mô hình dữ liệu là gì? Mô tả mô hình quan hệ.").
- Tốc độ phản hồi chậm nhất: 6.6 giây (Câu 1: "CSDL và DBMS là gì? Phân biệt chúng.").



HÌNH 5.3: Biểu đồ phân bố tốc độ phản hồi cho từng câu hỏi

(Nguồn: Tự tổng hợp từ số liệu thống kê đánh giá định lượng)

- Nhận xét:
  - Tốc độ phản hồi trung bình của hệ thống vào khoảng 4,99 giây/câu, một con số đủ nhanh để bảo đảm người dùng không phải chờ đợi lâu, qua đó nâng cao trải nghiệm tương tác. Thời gian phản hồi dao động trong dải hẹp từ 4,0 giây (nhanh nhất) đến 6,6 giây (chậm nhất).
  - Không xuất hiện trường hợp tăng đột biến về thời gian phản hồi; mọi giá trị đều nằm trong khoảng mong đợi và đáp ứng chuẩn hiệu năng đề ra. Nhìn chung, hệ thống đạt mục tiêu "phản hồi nhanh và nhất quán", góp phần tạo nền tảng tin cậy cho các tác vụ truy vấn dữ liệu trong thực tế.

5.4 Kết luận phân tích:

- Hiệu năng chung: Chatbot đạt 1,696/2 điểm liên quan, 85,7 % khớp nguồn và thời gian phản hồi trung bình ≈ 5 s.
- Ưu điểm: Khả năng hiểu câu hỏi tốt, hạn chế "ảo giác" nhờ cơ chế RAG, tốc độ phản hồi nhanh.
- Hạn chế:
  - Một số câu hỏi về kiến trúc DBMS và quy trình thiết kế CSDL vẫn thiếu thông tin hoặc sai lệch nguồn.
  - Cần tối ưu thuật toán truy xuất để phủ kín chủ đề hiếm gặp trong tài liệu.

- Việc đánh giá chỉ dừng lại ở các chỉ số kỹ thuật, chưa có phản hồi trực tiếp về trải nghiệm và tính tiện dụng từ góc độ người dùng cuối (sinh viên)

- Đề xuất cải thiện:
  - Bổ sung ngữ liệu cho các chủ đề còn yếu (mức trừu tượng, mô hình ERM).
  - Tinh chỉnh bộ lọc truy xuất để ưu tiên đoạn văn giàu thông tin, giảm suy diễn của LLM.
  - Theo dõi thêm các chỉ số về ổn định thời gian phản hồi trong tải thực tế.

CHƯƠNG 6 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Tóm tắt kết quả và đóng góp của đề tài:

Đề tài đã nghiên cứu và xây dựng thành công một hệ thống chatbot ứng dụng kiến trúc Retrieval-Augmented Generation (RAG) để hỗ trợ sinh viên học tập môn Cơ sở dữ liệu. Hệ thống đáp ứng tốt các yêu cầu cốt lõi: trả lời câu hỏi dựa trên một kho tri thức chung được quản lý tập trung, trích dẫn nguồn tin cậy để tăng tính minh bạch và đảm bảo tính nhất quán của thông tin.

Kết quả đánh giá định lượng cho thấy chatbot có độ chính xác cao (85.7% câu trả lời khớp nguồn), tốc độ phản hồi nhanh (trung bình ~5 giây), chứng tỏ hệ thống hoạt động hiệu quả và ổn định.

6.2 Hạn chế của hệ thống và nghiên cứu:

Bên cạnh những kết quả đạt được, hệ thống vẫn còn một số hạn chế:

- Độ chính xác chưa tuyệt đối: Hệ thống đôi khi trả lời chưa đầy đủ hoặc thiếu chính xác với các chủ đề phức tạp, ít được đề cập trong tài liệu. Khả năng xử lý các nội dung phi văn bản như sơ đồ, hình ảnh còn chưa có.
- Chiến lược xử lý tài liệu: Việc phân đoạn văn bản (chunking) còn ở mức cơ bản, đôi khi có thể làm vỡ ngữ cảnh của các khối thông tin logic quan trọng.
- Phương pháp đánh giá: Việc đánh giá mới chỉ tập trung vào các chỉ số kỹ thuật (định lượng) và được thực hiện bởi chính tôi. Do đó, báo cáo chưa phản ánh được các khía cạnh về trải nghiệm người dùng, tính dễ sử dụng và sự hữu ích thực tế từ góc nhìn của sinh viên. Đây là một hạn chế quan trọng cần được khắc phục trong các nghiên cứu tiếp theo.

6.3 Đề xuất hướng phát triển:

Để hoàn thiện và nâng cao hệ thống, các hướng phát triển trong tương lai được đề xuất bao gồm:

- Nâng cao chất lượng truy xuất và sinh câu trả lời: Áp dụng các kỹ thuật RAG và chunking tiên tiến hơn để cải thiện độ chính xác. Thử nghiệm và tích hợp các mô hình embedding và LLM được tối ưu cho tiếng Việt.
- Mở rộng tính năng thông minh: Tích hợp khả năng xử lý đa phương thức (hiểu hình ảnh, sơ đồ ER), phát triển các công cụ tương tác SQL nâng cao (gỡ lỗi, tối ưu) và cá nhân hóa lộ trình học tập cho sinh viên.

- Tối ưu hóa hiệu năng và đánh giá toàn diện: Cải thiện tốc độ và trải nghiệm trên giao diện người dùng, đồng thời tiến hành các nghiên cứu đánh giá trên quy mô lớn hơn để kiểm chứng hiệu quả thực tiễn của hệ thống.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” *arXiv preprint* arXiv:2005.11401, May 2020. [Online]. Available: https://arxiv.org/abs/2005.11401. Accessed: May 5, 2025

[2] W. Yeo, K. Kim, S. Jeong, J. Baek, and S. J. Hwang, “UniversalRAG: Retrieval-augmented generation over corpora of diverse modalities and granularities,” *arXiv preprint* arXiv:2504.20734, Apr. 2025. [Online]. Available: https://arxiv.org/abs/2504.20734. Accessed: May 5, 2025

[3] Q. D. Nguyen, H. S. Le, D. N. Nguyen, D. N. N. Nguyen, T. H. Le, and V. S. Dinh, “Towards comprehensive Vietnamese retrieval-augmented generation and large language models,” *arXiv preprint* arXiv:2403.01616, Mar. 2024. [Online]. Available: https://arxiv.org/abs/2403.01616. Accessed: May 5, 2025

[4] Y. Gao et al., “Retrieval-augmented generation for large language models: A survey,” *arXiv preprint* arXiv:2312.10997, Dec. 2023. [Online]. Available: https://arxiv.org/abs/2312.10997. Accessed: May 5, 2025

[5] Z. Jiang et al., “Active retrieval-augmented generation,” *arXiv preprint* arXiv:2305.06983, May 2023. [Online]. Available: https://arxiv.org/abs/2305.06983. Accessed: May 5, 2025

[6] D. Lin, “Revolutionizing retrieval-augmented generation with enhanced PDF structure recognition,” *arXiv preprint* arXiv:2401.12599, Jan. 2024. [Online]. Available: https://arxiv.org/abs/2401.12599. Accessed: May 5, 2025

[7] A. A. Khan, M. T. Hasan, K. K. Kemell, J. Rasku, and P. Abrahamsson, “Developing retrieval-augmented generation (RAG)-based LLM systems from PDFs: An experience report,” *arXiv preprint* arXiv:2410.15944, Oct. 2024. [Online]. Available: https://arxiv.org/abs/2410.15944. Accessed: May 5, 2025

[8] C. Alario-Hoyos, R. Kemcha, C. D. Kloos, P. Callejo, I. Estévez-Ayres, D. Santín-Cristóbal, F. Cruz-Argudo, and J. L. López-Sánchez, “Tailoring your code companion: Leveraging LLMs and RAG to develop a chatbot to support students in a programming course,” in *Proc. IEEE Int. Conf. Teaching, Assessment and Learning for Engineering (TALE)*, Bengaluru, India, Dec. 9–12, 2024, pp. 1–8. [Online]. Available: https://dblp.org/rec/conf/tale/Alario-HoyosKKC24. Accessed: May 5, 2025

[9] G. Lang and T. Gürpinar, “AI-powered learning support: A study of retrieval-augmented generation (RAG) chatbot effectiveness in an online course,” *Information Systems Education Journal*, vol. 23, no. 2, pp. 4–13, Mar. 2025. [Online]. Available: https://files.eric.ed.gov/fulltext/EJ1467995.pdf. Accessed: May 5, 2025

[10] H. A. Modran, I. C. Bogdan, D. Ursuțiu, C. Samoilă, and P. L. Modran, “LLM intelligent agent tutoring in higher education courses using a RAG approach,” in *Proc. Int. Conf. Interactive Collaborative Learning (ICL)*, Tallinn, Estonia, Sep. 24–27, 2024, pp. 589– 599. [Online]. Available: https://www.preprints.org/manuscript/202407.0519/v1. Accessed: May 5, 2025