

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1**

_____o0o_____



**BÁO CÁO NGHIÊN CỨU
HỌC PHẦN: THỰC TẬP CƠ SỞ
ĐỀ TÀI**

XÂY DỰNG CHATBOT HỎI ĐÁP MÔN HỌC

LỚP : N13

Sinh viên thực hiện:

Phạm Công Minh MSSV: B22DCCN542

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

HÀ NỘI, 05/2025

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành nhất đến thầy **Kim Ngọc Bách** đã quan tâm, chỉ bảo và thoải mái tạo điều kiện để cho em có thể hoàn thành đề tài một cách thuận lợi. Bản thân em cũng đã chọn cho mình một chủ đề mới để thử sức qua đó cũng tích lũy và học hỏi thêm được nhiều kỹ năng cũng như công nghệ mới. Em xin chúc thầy một lần nữa thật nhiều sức khỏe, niềm vui, thành công hơn nữa trên con đường giảng dạy và nghiên cứu ạ.

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....	1
1.1 Đặt vấn đề.....	1
1.2 Mục tiêu và phạm vi đề tài.....	1
1.3 Định hướng giải pháp.....	1
1.4 Bố cục bài tập lớn.....	2
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT.....	3
2.1 Giới thiệu về hệ thống chatbot hỏi đáp.....	3
2.2 Mô hình ngôn ngữ lớn (LLM).....	3
2.3 Ollama – Nền tảng chạy mô hình LLM và embedding cục bộ	4
2.4 Langchain – Framework tích hợp LLM và dữ liệu.....	4
2.5 FastAPI – Framework xây dựng API backend.....	5
2.6 React và Tailwind CSS – Xây dựng giao diện người dùng.....	6
2.7 MongoDB – Cơ sở dữ liệu NoSQL.....	6
2.8 Docker – Công cụ đóng gói và triển khai	6
2.9 AWS – Nền tảng triển khai hệ thống trên đám mây	7
CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG.....	8
3.1 Khảo sát yêu cầu.....	8
3.2 Kiến trúc hệ thống tổng thể	8
3.3 Thiết kế chức năng.....	9
3.4 Thiết kế cơ sở dữ liệu	9
3.5 Thiết kế logic xử lý	11
3.6 Thiết kế hệ thống backend	16
3.7 Thiết kế giao diện người dùng.....	23

CHƯƠNG 4. CÀI ĐẶT VÀ TRIỂN KHAI HỆ THỐNG	26
4.1 Cài đặt môi trường local với docker	26
4.2 Triển khai hệ thống trên AWS EC2.....	26
CHƯƠNG 5. ĐÁNH GIÁ VÀ KẾT QUẢ THỰC NGHIỆM.....	28
5.1 Kịch bản thử nghiệm.....	28
5.2 Kết quả thực nghiệm	28
5.3 Đánh giá hệ thống.....	28
5.4 Hướng phát triển hệ thống	29

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong bối cảnh số lượng sinh viên ngày càng đông và khối lượng tài liệu học tập ngày càng nhiều, nhu cầu được hỗ trợ thông tin nhanh chóng, chính xác và tiện lợi là vô cùng cấp thiết. Thực tế cho thấy, nhiều sinh viên thường gặp khó khăn khi tìm kiếm thông tin liên quan đến môn học, mất nhiều thời gian để tra cứu và rất dễ trở nên chán nản với đồng tài liệu. Việc tiếp cận thông tin không kịp thời có thể ảnh hưởng đến quá trình học tập và kết quả của sinh viên.

Vì vậy, việc xây dựng một hệ thống hỗ trợ hỏi đáp, có khả năng phản hồi thông minh và liên tục, đang trở thành một xu hướng cần thiết trong môi trường giáo dục. Nếu được triển khai hiệu quả, hệ thống này không chỉ hỗ trợ sinh viên hơn trong việc học mà còn góp phần giảm tải cho giảng viên, đồng thời có tiềm năng mở rộng thành chatbot tự động hỗ trợ thời gian thực và hỗ trợ học tập cá nhân hóa cho sinh viên.

1.2 Mục tiêu và phạm vi đề tài

Hiện nay, nhiều trường đại học và tổ chức giáo dục đã bắt đầu áp dụng các công nghệ chatbot để hỗ trợ sinh viên trong việc tra cứu thông tin, hỏi đáp hành chính hoặc cung cấp nội dung học tập cơ bản. Tuy nhiên, phần lớn các chatbot hiện tại vẫn mang tính chất tĩnh, chủ yếu phản hồi theo kịch bản định sẵn, thiếu khả năng hiểu ngữ cảnh và trả lời linh hoạt theo nội dung học thuật.

Trên cơ sở đó, đề tài hướng đến việc phát triển một chatbot hỗ trợ sinh viên trong quá trình học tập, với chức năng chính là: **hỏi đáp dựa trên nội dung giáo trình**, đồng thời khắc phục hạn chế của các chatbot truyền thống bằng khả năng hiểu ngữ cảnh tốt hơn và hỗ trợ cá nhân hóa nội dung cho từng người học.

1.3 Định hướng giải pháp

Để giải quyết bài toán hỏi đáp dựa trên nội dung giáo trình, đề tài định hướng sử dụng mô hình ngôn ngữ lớn (LLM) kết hợp với kỹ thuật truy hồi tri thức (Retrieval-Augmented Generation – RAG). Cụ thể, nhóm lựa chọn sử dụng Langchain để xây dựng pipeline truy hồi và tổng hợp thông tin, FastAPI để triển khai backend xử lý truy vấn, và MongoDB để lưu trữ lịch sử trò chuyện. Docker được sử dụng để đóng gói và triển khai hệ thống một cách nhất quán.

Giải pháp của đề tài là xây dựng một chatbot cho phép sinh viên đặt câu hỏi liên quan đến nội dung giáo trình môn học. Câu hỏi sẽ được xử lý qua pipeline truy hồi – LLM để tìm kiếm thông tin phù hợp từ tài liệu đã cung cấp và tạo câu trả lời tự

nhien, rõ ràng.

Đóng góp chính của bài tập lớn là hiện thực hóa một hệ thống hỏi đáp đơn giản nhưng hiệu quả, sử dụng công nghệ hiện đại, có khả năng mở rộng trong tương lai. Kết quả đạt được là một ứng dụng chatbot có thể tiếp nhận câu hỏi, tìm kiếm trong tài liệu PDF, sinh câu trả lời và lưu lại lịch sử tương tác của sinh viên phục vụ mục đích học tập lâu dài.

1.4 Bố cục bài tập lớn

Phần còn lại của báo cáo bài tập lớn này được tổ chức như sau.

Chương 2 trình bày cơ sở lý thuyết phục vụ cho việc xây dựng hệ thống chatbot hỏi đáp tài liệu học tập, bao gồm các kiến thức nền tảng về mô hình ngôn ngữ lớn (LLM), kỹ thuật truy hồi tăng cường (RAG), vector database Milvus, framework Langchain, và các công nghệ sử dụng trong dự án như FastAPI, Docker và MongoDB. Việc tổng hợp các khái niệm này nhằm giúp người đọc hiểu rõ nền tảng công nghệ làm cơ sở cho thiết kế và triển khai hệ thống trong các chương sau.

Chương 3 trình bày quá trình phân tích yêu cầu người dùng và thiết kế kiến trúc tổng thể của hệ thống chatbot. Chương này bao gồm sơ đồ các thành phần chức năng, luồng xử lý dữ liệu, thiết kế frontend và backend, cơ sở dữ liệu, cũng như cách lưu trữ và truy vấn tài liệu. Đây là bước quan trọng giúp định hướng rõ ràng cách triển khai hệ thống một cách mạch lạc và hiệu quả.

Chương 4 trình bày chi tiết quá trình cài đặt và triển khai hệ thống trên môi trường local cũng như trên nền tảng đám mây AWS EC2. Nội dung chương bao gồm các bước chuẩn bị môi trường, build frontend, khởi chạy hệ thống bằng Docker Compose, pull mô hình ngôn ngữ và cấu hình các thành phần. Ngoài ra, chương này còn giải thích lý do lựa chọn AWS EC2 và cách tối ưu tài nguyên với swap RAM để đảm bảo hệ thống hoạt động ổn định.

Chương 5 trình bày kết quả thực nghiệm và đánh giá hiệu quả của hệ thống chatbot sau khi triển khai. Các khía cạnh được đánh giá bao gồm độ chính xác trong phản hồi câu hỏi, hiệu suất truy vấn tài liệu, khả năng mở rộng, và mức độ thân thiện với người dùng. Cuối chương, nhóm đề xuất một số hướng phát triển trong tương lai nhằm nâng cao chất lượng chatbot, chẳng hạn như tích hợp AI tạo câu hỏi trắc nghiệm, tự động tổng hợp nội dung buổi học, và đồng bộ hóa với lịch học cá nhân để nhắc nhở sinh viên hiệu quả hơn.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

TỔNG QUAN:

Sau khi đã trình bày phần giới thiệu và đặt vấn đề trong chương 1, chương 2 này sẽ cung cấp các kiến thức nền tảng cần thiết để hiểu rõ hơn về công nghệ và kỹ thuật được sử dụng trong hệ thống chatbot hỗ trợ học tập. Việc xây dựng một hệ thống chatbot hỏi đáp đòi hỏi sự kết hợp giữa nhiều lĩnh vực như xử lý ngôn ngữ tự nhiên, truy hồi thông tin, lưu trữ vector, và phát triển ứng dụng web. Chương này sẽ lần lượt trình bày các khái niệm quan trọng bao gồm mô hình ngôn ngữ lớn (LLM), kiến trúc RAG (Retrieval-Augmented Generation), cơ sở dữ liệu vector Milvus, framework Langchain, và các công nghệ hỗ trợ như FastAPI, MongoDB và Docker.

2.1 Giới thiệu về hệ thống chatbot hỏi đáp

Hệ thống hỏi đáp (Question Answering – QA) là một nhánh quan trọng trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP). Mục tiêu của QA là trả lời tự động các câu hỏi do con người đặt ra, bằng cách khai thác thông tin từ cơ sở tri thức hoặc tài liệu đầu vào. Với sự phát triển của các mô hình ngôn ngữ lớn (LLM), hệ thống QA ngày càng hiệu quả hơn, đặc biệt khi áp dụng kiến trúc RAG (Retrieval-Augmented Generation) để truy xuất thông tin từ nguồn dữ liệu cụ thể như giáo trình.

1. Kiến trúc Retrieval-Augmented Generation (RAG)

Khái quát kiến trúc RAG:

RAG là kiến trúc kết hợp giữa retriever và generator:

- **Retriever:** Truy xuất đoạn văn bản liên quan từ kho tài liệu.
- **Generator:** Dựa vào đoạn truy xuất được, sinh câu trả lời bằng LLM.

Cơ chế này cho phép hệ thống QA (Question Answering) vừa đảm bảo tính chính xác của thông tin, vừa tận dụng sức mạnh ngôn ngữ của LLM.

2.2 Mô hình ngôn ngữ lớn (LLM)

1. **Khái niệm và vai trò LLM** (Large Language Model) là mô hình học sâu với hàng tỷ tham số, được huấn luyện trên tập dữ liệu văn bản rất lớn. Mục tiêu là học cách sinh ngôn ngữ tự nhiên và hiểu ngữ cảnh để trả lời câu hỏi, dịch thuật, tóm tắt, và nhiều tác vụ NLP khác. Cơ chế này cho phép hệ thống QA (Question Answering) vừa đảm bảo tính chính xác của thông tin, vừa tận dụng sức mạnh ngôn ngữ của LLM.

2. Một số mô hình tiêu biểu

- GPT (OpenAI)
- Command-R+ (Cohere)
- DeepSeek
- Mistral
- LLaMA

2.3 Ollama – Nền tảng chạy mô hình LLM và embedding cục bộ

1. Giới thiệu

Ollama là nền tảng hỗ trợ chạy các mô hình ngôn ngữ lớn (LLM) trực tiếp trên máy tính cá nhân, giúp phát triển ứng dụng AI một cách độc lập, không cần phụ thuộc vào API từ bên thứ ba. Ollama đặc biệt hữu ích trong việc vận hành các mô hình embedding cục bộ – một bước quan trọng trong quy trình xây dựng hệ thống hỏi đáp dựa trên tìm kiếm ngữ nghĩa.

Các mô hình embedding phổ biến như ‘bge-m3’, ‘E5’, hoặc ‘all-MiniLM’ có thể được tải và triển khai dễ dàng qua Ollama. Điều này giúp:

- **Tiết kiệm chi phí:** Không cần gọi API embedding từ dịch vụ đám mây như OpenAI hay Cohere.
- **Bảo mật dữ liệu:** Dữ liệu không cần gửi lên server bên ngoài.

2. Vai trò trong hệ thống

Trong hệ thống chatbot hỏi đáp, Ollama được sử dụng để:

- Chạy mô hình embedding để chuyển văn bản từ giáo trình thành vector số.
- Tích hợp với Langchain thông qua OllamaEmbeddings, hỗ trợ pipeline nhúng và truy xuất dữ liệu.
- Cung cấp nền tảng linh hoạt để thử nghiệm nhiều mô hình khác nhau như DeepSeek, Mistral, LLaMA theo nhu cầu.

Nhờ khả năng chạy offline, nhẹ và dễ tích hợp, Ollama là giải pháp phù hợp để phát triển hệ thống hỏi đáp học thuật dựa trên mô hình embedding một cách chủ động và hiệu quả.

2.4 Langchain – Framework tích hợp LLM và dữ liệu

1. Giới thiệu

Langchain là thư viện mã nguồn mở trong Python, giúp xây dựng ứng dụng

sử dụng LLM một cách linh hoạt. Framework này hỗ trợ pipeline gồm: tải dữ liệu, chia đoạn, nhúng vector, lưu trữ, truy xuất và gọi mô hình.

2. Các thành phần chính

- Document Loaders: Dùng PyPDFLoader để tải dữ liệu từ các nguồn file PDF.
- Text Splitters: Dùng RecursiveCharacterTextSplitter để chia nội dung lớn thành các đoạn nhỏ (chunk) để phù hợp với giới hạn đầu vào của mô hình LLM và tăng hiệu quả truy xuất cũng như embedding.
- Embeddings + Vector Store: Sau khi chia nhỏ, các đoạn văn bản sẽ được chuyển thành vector bằng mô hình embedding (ví dụ: bge-m3) thông qua OllamaEmbeddings. Các vector này được lưu trữ trong vector database Milvus để phục vụ truy xuất ngữ nghĩa hiệu quả.
- Retriever Tool: Sử dụng create_retriever_tool để chuyển retriever (kết hợp giữa Milvus retriever và BM25 retriever) thành một tool. Tool này được tích hợp vào agent để lấy thông tin từ dữ liệu đã lưu trữ.
- Agent + Tools: Langchain hỗ trợ xây dựng agent có thể dùng nhiều tool cùng lúc. Ở đây sử dụng create_tool_calling_agent để tạo một agent biết cách gọi các công cụ như find_documents. Agent được thực thi bằng AgentExecutor, cho phép xử lý truy vấn linh hoạt, giống một trợ lý thông minh.
- Prompt với ChatPromptTemplate: Để giao tiếp hiệu quả giữa người dùng và agent, Langchain dùng ChatPromptTemplate để thiết kế prompt hội thoại. Prompt này hỗ trợ truyền hệ thống hướng dẫn (system prompt), đầu vào người dùng và lịch sử trò chuyện (MessagesPlaceholder) để mô hình hiểu bối cảnh đầy đủ.

2.5 FastAPI – Framework xây dựng API backend

1. Giới thiệu

FastAPI là một framework hiện đại trong Python để xây dựng API RESTful. Nó có cú pháp đơn giản, hiệu năng cao, hỗ trợ bất đồng bộ (async) và được thiết kế để tích hợp dễ dàng với các công nghệ như Pydantic và OpenAPI.

2. Ứng dụng

- Nhận câu hỏi từ frontend
- Gửi truy vấn vào Langchain để xử lý

- Nhận và trả lời kết quả từ mô hình
- Lưu lịch sử trò chuyện vào MongoDB

2.6 React và Tailwind CSS – Xây dựng giao diện người dùng

1. React – Thư viện xây dựng giao diện

React là thư viện JavaScript phổ biến dùng để xây dựng giao diện người dùng theo dạng component. React hỗ trợ cập nhật giao diện theo trạng thái, tối ưu hiệu năng và dễ mở rộng trong các ứng dụng tương tác cao như chatbot. Trong hệ thống, React được dùng để:

- Tạo giao diện hội thoại giữa người dùng và chatbot
- Gửi truy vấn từ người dùng đến backend FastAPI
- Hiển thị kết quả phản hồi từ mô hình
- Quản lý trạng thái hội thoại và lịch sử trao đổi

2. Tailwind CSS – Thiết kế giao diện hiện đại

Tailwind CSS là framework CSS tiện lợi cho phép lập trình viên thiết kế nhanh chóng với các lớp utility. So với viết CSS truyền thống, Tailwind giúp:

- Tạo giao diện đẹp, hiện đại mà không cần viết quá nhiều CSS tùy chỉnh
- Phối hợp linh hoạt với React qua các lớp class ngắn gọn
- Tăng tốc quá trình phát triển frontend và dễ bảo trì

Nhờ sự kết hợp của React và Tailwind, giao diện người dùng của hệ thống chatbot trở nên thân thiện, tối ưu trải nghiệm và dễ tích hợp với backend.

2.7 MongoDB – Cơ sở dữ liệu NoSQL

1. Giới thiệu

MongoDB là một cơ sở dữ liệu dạng document, lưu dữ liệu dưới dạng BSON (một dạng mở rộng của JSON). Nó phù hợp với dữ liệu có cấu trúc linh hoạt như lịch sử trò chuyện, người dùng và metadata của tài liệu.

2. Ứng dụng

- Lưu thông tin người dùng
- Lưu lịch sử hỏi đáp giữa người dùng và chatbot
- Quản lý môn học và các tài liệu giáo trình liên quan

2.8 Docker – Công cụ đóng gói và triển khai

1. Giới thiệu

Docker là công cụ giúp đóng gói ứng dụng cùng toàn bộ môi trường của nó vào các container. Nhờ đó, ứng dụng có thể chạy đồng nhất trên mọi môi trường, từ máy cá nhân đến máy chủ đám mây.

2. Ứng dụng

- Triển khai toàn bộ hệ thống chatbot, bao gồm các thành phần: Ollama, FastAPI, MongoDB
- Quản lý các dịch vụ bằng docker-compose
- Hỗ trợ tái sử dụng, dễ mở rộng và phát triển theo mô hình microservice

2.9 AWS – Nền tảng triển khai hệ thống trên đám mây

1. Giới thiệu

Amazon Web Services (AWS) là một trong những nền tảng điện toán đám mây hàng đầu, cung cấp đa dạng dịch vụ hỗ trợ triển khai, lưu trữ, xử lý và bảo mật hệ thống. AWS phù hợp để triển khai các ứng dụng AI/ML nhờ khả năng mở rộng linh hoạt và hiệu suất cao.

2. Ứng dụng

- Triển khai hệ thống chatbot lên máy chủ EC2 để có thể hoạt động liên tục 24/7

KẾT CHƯƠNG:

Chương này đã trình bày cơ sở lý thuyết của các thành phần quan trọng làm nền tảng cho hệ thống chatbot, giúp định hướng rõ ràng cho các bước thiết kế và cài đặt về sau. Kiến thức về LLM, RAG, vector database và các công nghệ đi kèm sẽ là cơ sở để triển khai hệ thống có khả năng xử lý ngôn ngữ hiệu quả, truy vấn tài liệu chính xác và phản hồi người dùng nhanh chóng. Tiếp theo, chương 3 sẽ trình bày cụ thể cách phân tích và thiết kế hệ thống dựa trên những nền tảng lý thuyết đã trình bày.

CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

TỔNG QUAN:

Tiếp nối chương 2 về các khái niệm lý thuyết và công nghệ cốt lõi, chương 3 này sẽ trình bày quá trình phân tích yêu cầu và thiết kế hệ thống chatbot hỏi đáp học tập. Đây là bước quan trọng nhằm chuyển hóa các mục tiêu lý thuyết thành mô hình kiến trúc thực tế. Chương này sẽ bao gồm các phần chính: phân tích chức năng hệ thống, sơ đồ kiến trúc tổng thể, thiết kế thành phần frontend và backend, thiết kế cơ sở dữ liệu, cũng như mô hình lưu trữ và truy vấn dữ liệu từ tài liệu học tập.

3.1 Khảo sát yêu cầu

1. Yêu cầu chức năng

- Cho phép người dùng đăng ký, đăng nhập tài khoản.
- Upload tài liệu môn học (PDF).
- Tự động xử lý, trích xuất và lưu trữ thông tin trong vector database.
- Hỏi đáp dựa trên nội dung tài liệu đã tải lên.
- Lưu lịch sử trò chuyện theo từng môn học.

2. Yêu cầu phi chức năng

- Hệ thống phản hồi nhanh (<5 giây).
- Có khả năng mở rộng, dễ tích hợp thêm model hoặc nguồn dữ liệu.
- Giao diện đơn giản, dễ sử dụng với sinh viên.
- Dễ triển khai cục bộ hoặc trên môi trường cloud (sử dụng Docker).

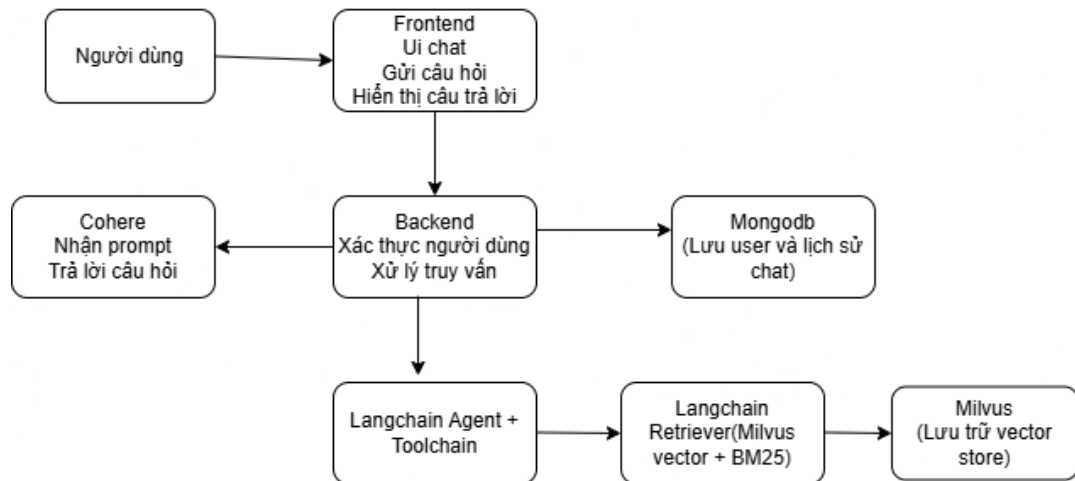
3.2 Kiến trúc hệ thống tổng thể

1. Hệ thống được thiết kế theo mô hình client-server, kết hợp các thành phần sau:

- Frontend (React): giao diện người dùng, cho phép đăng nhập, trò chuyện, tải tài liệu.
- Backend (FastAPI): cung cấp các API cho frontend; xử lý người dùng, truy vấn, vector hóa tài liệu.
- MongoDB: lưu trữ thông tin người dùng và lịch sử trò chuyện.
- Milvus: lưu trữ vector embedding của nội dung tài liệu.
- LLM (Ollama / Cohere): dùng để embedding data, hỏi đáp và tổng hợp.
- Langchain: tích hợp các thành phần trên, quản lý luồng truy vấn trong

RAG pipeline.

2. Sơ đồ kiến trúc



Hình 1. Sơ đồ tổng thể kiến trúc

3.3 Thiết kế chức năng

1. Sơ đồ ca sử dụng (Use Case Diagram)

Các tác nhân chính

- Người dùng (Sinh viên)
- Hệ thống

2. Các ca sử dụng chính:

- Đăng ký, đăng nhập
- Tải lên tài liệu
- Đặt câu hỏi
- Nhận câu trả lời
- Lưu lịch sử

3.4 Thiết kế cơ sở dữ liệu

1. MongoDB

MongoDB sử dụng mô hình document để lưu trữ dữ liệu phi quan hệ. Cấu trúc lưu trữ chính của hệ thống gồm 3 lớp dữ liệu: User, Subject, và Message, tương ứng với mỗi quan hệ 1 người dùng - nhiều môn học - nhiều đoạn hội thoại. Cấu trúc các collection:

1.1. User

Lưu thông tin về họ tên, mã sinh viên, mật khẩu, và danh sách các môn học (được khởi tạo là mảng rỗng)

```
class User(BaseModel):
    fullname: str
    stu_id: str
    password: str
    subject: List[Subject] = Field(default_factory=list)
```

Hình 2. Model User

1.2. Subject

Lưu thông tin về mã môn học, tên môn học, và danh sách các lịch sử hội thoại

```
class Subject(BaseModel):
    subjectId: str
    subjectName: str
    messages: List[Message]
```

Hình 3. Model Subject

1.3. Message

Lưu thông tin về role(user, assistance), và nội dung tin nhắn

```
class Message(BaseModel):
    role: str
    content: str
```

Hình 4. Model Message

1.4. Ví dụ document trong MongoDB Atlas

```
{
  "_id": ObjectId("6821cab58d86c2b21a9237f1"),
  "fullname": "Phạm công minh",
  "stu_id": "B22DCCN542",
  "password": "123",
  "subject": Array (1)
    0: Object
      subject_id: "t"
      subject_name: "tri_tue_nhan_tao"
      messages: Array (8)
        0: Object
        1: Object
        2: Object
        3: Object
  }
```

Hình 5. Ví dụ document MongoDB Atlas

Nhận xét:

- Dữ liệu được lưu dạng lồng nhau (nested document) giúp việc truy vấn và hiển thị lịch sử theo môn học rất thuận tiện. Điều này rất phù hợp với dữ liệu dạng conversation thread mà không cần thiết kế quan hệ phức tạp

như trong SQL.

- Tốc độ đọc/ghi nhanh cho các thao tác như lưu câu trả lời ngay sau khi sinh ra hoặc truy xuất lịch sử theo môn học.

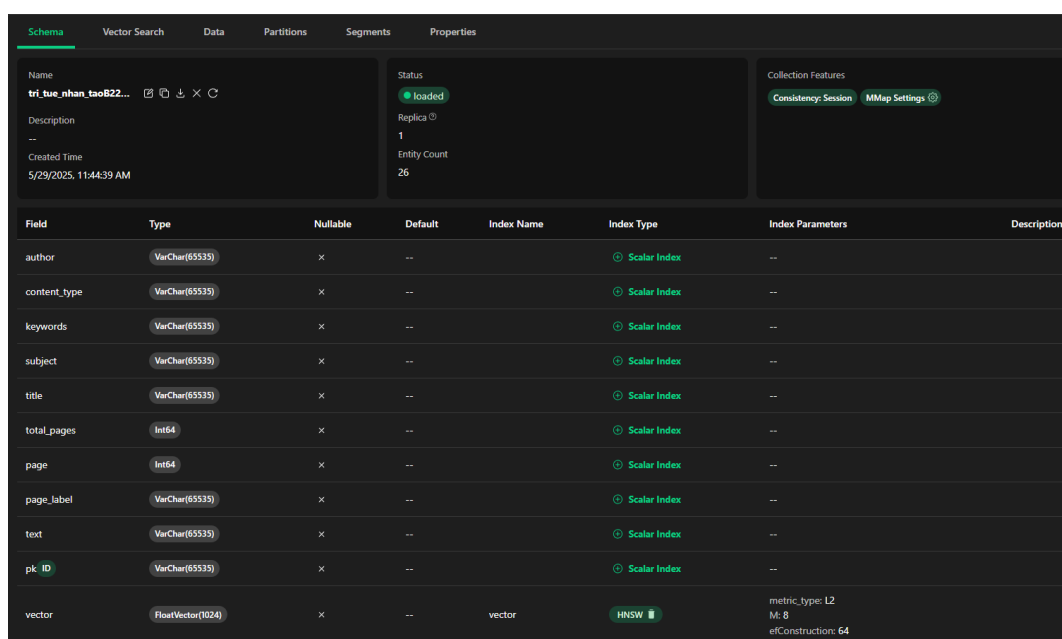
2. Milvus

Milvus là cơ sở dữ liệu vector mã nguồn mở, được thiết kế để lưu trữ và truy vấn dữ liệu có cấu trúc vector hóa — rất phù hợp với các ứng dụng trí tuệ nhân tạo sử dụng mô hình truy xuất tăng cường (RAG - Retrieval-Augmented Generation).

Trong hệ thống chatbot hỏi đáp theo giáo trình, Milvus được sử dụng để lưu trữ các vector embedding của từng đoạn văn bản sau khi trích xuất từ các tài liệu PDF. Mỗi đoạn văn được mã hóa thành vector và gắn kèm metadata giúp việc truy vấn linh hoạt, chính xác hơn.

Cấu trúc dữ liệu lưu trong Milvus:

- **page_content**: Nội dung văn bản gốc.
- **metadata**: Thông tin phụ trợ để lọc hoặc truy vấn theo điều kiện cụ thể.



Field	Type	Nullable	Default	Index Name	Index Type	Index Parameters	Description
author	VarChar(65535)	x	--		Scalar Index	--	
content_type	VarChar(65535)	x	--		Scalar Index	--	
keywords	VarChar(65535)	x	--		Scalar Index	--	
subject	VarChar(65535)	x	--		Scalar Index	--	
title	VarChar(65535)	x	--		Scalar Index	--	
total_pages	Int64	x	--		Scalar Index	--	
page	Int64	x	--		Scalar Index	--	
page_label	VarChar(65535)	x	--		Scalar Index	--	
text	VarChar(65535)	x	--		Scalar Index	--	
pk_ID	VarChar(65535)	x	--		Scalar Index	--	
vector	FloatVector(1024)	x	--	vector	HNSW	metric_type: L2 M: 8 efConstruction: 64	

Hình 6.Schema Milvus

3.5 Thiết kế logic xử lý

1. Embedding dữ liệu vào Milvus

1.1. Giới thiệu

Để chatbot có thể tìm kiếm và trả lời theo nội dung của các tài liệu PDF, hệ thống cần thực hiện quá trình nhúng văn bản (embedding) để chuyển các đoạn văn bản thành vector số học. Các vector này sẽ được lưu trữ trong cơ

sở dữ liệu vector Milvus để truy xuất hiệu quả thông qua cơ chế tìm kiếm gần đúng (Approximate Nearest Neighbor Search). Hệ thống sử dụng mô hình OllamaEmbeddings từ Langchain để sinh vector embedding, với đầu vào là các đoạn văn bản nhỏ được trích xuất từ tài liệu PDF. File mã nguồn seed_data.py đảm nhiệm vai trò chính trong quá trình xử lý này.

1.2. Quy trình embedding dữ liệu:

- **Đọc dữ liệu từ file PDF, chia nhỏ dữ liệu:**

Sử dụng PyPDFLoader để đọc toàn bộ nội dung văn bản từ tệp PDF, không trích xuất ảnh. Tài liệu sau đó được chia nhỏ bằng RecursiveCharacterTextSplitter với chunk_size = 800 và chunk_overlap = 200 nhằm đảm bảo tính toàn vẹn ngữ nghĩa.

```
def load_file_pdf(file_path:str):
    """
    Hàm này dùng để load file pdf từ đường dẫn file_path
    """
    loader = PyPDFLoader(file_path, extract_images=False)
    documents = loader.load()
    return documents

Windsurf: Refactor | Explain | Generate Docstring | X
def seed_data_milvus_ollama(file_path:str, use_model:str, collection_name :str,url:str):
    documents = load_file_pdf(file_path)
    # Chia nhỏ văn bản thành các đoạn nhỏ (chunks)
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=800, chunk_overlap=200)
    chunks = text_splitter.split_documents(documents)
```

Hình 7. Đọc file và chia nhỏ dữ liệu

- **Tạo đối tượng Document kèm metadata**

Mỗi đoạn văn bản nhỏ được gắn metadata để phục vụ mục đích lọc, tìm kiếm nâng cao. Các trường metadata gồm:

- **author:** Tác giả của tài liệu.
- **content_type:** Loại nội dung, mặc định là pdf.
- **keywords:** Từ khóa mô tả nội dung chính của tài liệu hoặc đoạn văn bản.
- **subject:** Môn học tương ứng với tài liệu, dùng để phân loại theo học phần.
- **title:** Tên của tài liệu hoặc giáo trình.
- **total_pages:** Tổng số trang của tài liệu gốc.
- **page:** Chỉ số trang chứa đoạn văn bản.

- **page_label**: Nhân trang (có thể là số, ký hiệu chương, hoặc phân đoạn tùy theo định dạng file).

```
# Chuyển đổi `chunks` thành `Document` để lưu vào Milvus
documents = [
    Document(
        page_content=chunk.page_content,
        metadata={
            'author': chunk.metadata.get('author', ''),
            'content_type': chunk.metadata.get('content_type', 'pdf'),
            'keywords': chunk.metadata.get('keywords', ''),
            'subject': chunk.metadata.get('subject', ''),
            'title': chunk.metadata.get('title', ''),
            'total_pages': chunk.metadata.get('total_pages', 0),
            'page': chunk.metadata.get('page', 0),
            'page_label': chunk.metadata.get('page_label', 0),
        }
    )
    for chunk in chunks
]
```

Hình 8. Tạo documents

1.3. Kết nối Milvus, ollama mdel, sinh vector embedding và lưu trữ vào Milvus

Hệ thống khởi tạo kết nối đến Milvus thông qua API của pymilvus, sau đó sử dụng OllamaEmbeddings để tạo embedding cho từng đoạn. Embedding được lưu vào collection cụ thể trong Milvus, mỗi vector gắn với một UUID và metadata tương ứng.

```
uuids = [str(uuid4()) for _ in range(len(documents))]
# Kết nối với Milvus
connect_milvus()
#khởi tạo OllamaEmbeddings
embeddings = OllamaEmbeddings(
    model=use_model,
    base_url="http://ollama:11434"
)
try:
    vectorstore = Milvus(
        embedding_function=embeddings,
        collection_name=collection_name,
        drop_old=False,
        connection_args={"host": "milvus-standalone", "port": "19530"} # Chỉ định rõ host và port
    )
    vectorstore.add_documents(documents=documents, ids = uuids) # Thêm documents vào Milvus
    print("Đã lưu embeddings vào MilvusDB!")
except Exception as e:
    print(f"Kết nối Milvus thất bại: {e}")
```

Hình 9. Gán uuid, embedding và lưu vào Milvus

2. Retriever và cơ chế truy xuất tài liệu

Trong hệ thống chatbot hỗ trợ học tập, việc tìm kiếm thông tin chính xác từ các tài liệu đầu vào là yêu cầu then chốt. Để hiện thực điều này, nhóm sử dụng cơ chế retriever trong Langchain – một thành phần chuyên dụng dùng để truy xuất các đoạn văn bản liên quan đến truy vấn người dùng, từ đó phục vụ cho

mô hình ngôn ngữ lớn (LLM) xử lý tiếp theo.

2.1. Milvus Retriever – Truy xuất ngữ nghĩa

Milvus là cơ sở dữ liệu vector hiệu năng cao, hỗ trợ tìm kiếm theo độ tương đồng ngữ nghĩa giữa truy vấn và nội dung văn bản đã được embedding. Retriever Milvus được khởi tạo từ vectorstore đã lưu embeddings từ trước (xem phần 3.4.3), với tham số $k = 4$, tức trả về 4 đoạn văn bản gần nhất về mặt ngữ nghĩa.

```
# Milvus retriever
milvus_retriever = vectorstore.as_retriever(
    search_type="similarity",
    search_kwargs={"k": 4}
)
```

Hình 10. Milvus Retriever

Cơ chế này giúp hệ thống không chỉ tìm văn bản chứa từ khóa, mà còn hiểu ngữ cảnh và nghĩa mở rộng của truy vấn người dùng, từ đó đưa ra kết quả sát với mong đợi hơn.

2.2. BM25 Retriever – Truy xuất theo từ khóa

Song song với tìm kiếm ngữ nghĩa, hệ thống bổ sung một retriever khác dựa trên thuật toán BM25 – một mô hình truyền thống mạnh mẽ trong tìm kiếm từ khóa. BM25 sẽ xét độ liên quan dựa trên tần suất và vị trí từ khóa trong văn bản.

```
# Lấy tất cả documents từ vectorstore
documents = [
    Document(page_content=doc.page_content, metadata=doc.metadata)
    for doc in vectorstore.similarity_search("", k=100)
]
if not documents:
    raise ValueError(f"Không tìm thấy documents trong collection '{collection_name}'")
# BM25 retriever
bm25_retriever = BM25Retriever.from_documents(documents)
bm25_retriever.k = 4
```

Hình 11. BM25 Retriever

Các tài liệu được truyền cho BM25 retriever là toàn bộ nội dung đã lưu trong Milvus (được lấy ra bằng truy vấn `similarity_search("", k=100)`).

2.3. Ensemble Retriever – Kết hợp đa chiến lược tìm kiếm

Để tận dụng điểm mạnh của cả hai phương pháp (Milvus và BM25), hệ thống sử dụng EnsembleRetriever – một lớp tổng hợp cho phép trộn nhiều retriever với trọng số tùy chỉnh.

```
# Ensemble retriever
ensemble_retriever = EnsembleRetriever(
    retrievers=[milvus_retriever, bm25_retriever],
    weights=[0.7, 0.3]
)
```

Hình 12. Ensemble Retriever

- Milvus retriever được ưu tiên hơn (trọng số 0.7) nhằm tăng độ chính xác ngữ nghĩa.
- BM25 retriever bổ trợ khả năng tìm kiếm chính xác theo từ khóa.

2.4. Trường hợp xử lý lỗi

Trong tình huống không thể kết nối tới Milvus hoặc không có dữ liệu, hệ thống fallback về BM25 với một tài liệu báo lỗi để tránh gây gián đoạn:

```
# Trả về retriever default
default_doc = [
    Document(
        page_content="Có lỗi xảy ra khi kết nối database. Vui lòng thử lại sau.",
        metadata={"source": "error"}
    )
]
return BM25Retriever.from_documents(default_doc)
```

Hình 13. Xử lý lỗi retriever

3. Agent và Tool

Để xây dựng một chatbot thông minh có khả năng sử dụng các công cụ như tìm kiếm tài liệu và tóm tắt nội dung, hệ thống sử dụng Langchain Agent. Đây là một kiến trúc cho phép LLM có thể ra quyết định và gọi các công cụ tương ứng để xử lý yêu cầu người dùng, tương tự như một quy trình suy luận đa bước.

3.1. Truy xuất tài liệu với retriever

- Hệ thống khởi tạo một retriever tool để tìm kiếm thông tin trong tài liệu đã nhúng vào Milvus. Tool này được định nghĩa bằng phương thức `create_retriever_tool()`:

```
# Khởi tạo retriever tool
find_documents_tool = create_retriever_tool(
    retriever,
    "find_documents",
    "Search for information of lịch sử đảng."
)
```

Hình 14. Định nghĩa tool tìm kiếm tài liệu

3.2. Khởi tạo mô hình LLM và mảng tools

- Sau khi khởi tạo tool, hệ thống cấu hình một mô hình ngôn ngữ lớn (LLM) – ở đây là command-r-plus từ Cohere, với thiết lập temperature = 0.5 thấp để đảm bảo tính chính xác trong phản hồi tạo thêm mảng tools gồm tool find_documents_tool đã được định nghĩa ở trên:

```
# Khởi tạo chatbot Cohere
if not os.environ.get("COHERE_API_KEY"):
    os.environ["COHERE_API_KEY"] = getpass.getpass("Enter API key for Cohere: ")

llm = init_chat_model("command-r-plus", model_provider="cohere", model_kwargs={"temperature": 0.5})
tools = [find_documents_tool]
```

Hình 15. Khởi tạo mô hình LLM và mảng tools

3.3. Tạo Agent với khả năng suy luận và gọi tool

- Cuối cùng, agent được khởi tạo với mô hình và tools đã định nghĩa. Hệ thống sử dụng create_tool_calling_agent() để cho phép LLM quyết định khi nào nên gọi tool (trong trường hợp này là tìm kiếm tài liệu):

```
# Tạo agent
agent = create_tool_calling_agent(llm=llm, tools=tools, prompt=prompt)
return AgentExecutor(agent=agent, tools=tools, verbose=True)
```

Hình 16. Khởi tạo chatbot Agent

3.4. Luồng xử lý tổng thể

- Người dùng nhập truy vấn (ví dụ: "Tư tưởng Hồ Chí Minh").
- Agent phân tích truy vấn và gọi tool find_documents
- Retriever trả về các đoạn văn bản liên quan. _tool
- LLM đọc và tổng hợp trả lời dựa trên đoạn văn đã truy xuất.
- Nếu không có thông tin, Agent sẽ trả lời mặc định: "Xin lỗi, tôi không tìm thấy thông tin trong giáo trình."

3.6 Thiết kế hệ thống backend

Backend được xây dựng bằng FastAPI nhằm cung cấp các API phục vụ các chức năng chính của hệ thống chatbot như: đăng ký, đăng nhập, thêm môn học, lưu và truy xuất lịch sử hội thoại, tải file giáo trình PDF, gửi câu hỏi và nhận phản hồi từ mô hình AI. Dữ liệu người dùng được lưu trữ trong MongoDB, và nội dung giáo trình sau khi xử lý được lưu vào Milvus để hỗ trợ tìm kiếm ngữ nghĩa trong mô hình RAG.

Các API chính

1. API Đăng ký

- Endpoint: POST /api/signup
- Input: JSON đối tượng User (gồm: fullname, stu_id, password)
- Xử lý
 - Kiểm tra sinh viên đã tồn tại trong MongoDB chưa (dựa theo stu_id).
 - Nếu chưa có thì tiến hành insert User mới vào database.
- Output: Trạng thái success/fail.

```
async def signup(user: User):  
  
    thisUser = jsonable_encoder(user)  
    existing_user = await collection.find_one({"stu_id":thisUser["stu_id"]})  
    if existing_user:  
        return {"status": "fail", "message": "User already exists"}  
    result = await collection.insert_one(thisUser)  
    return {"status": "success", "user_id": str(result.inserted_id)}
```

Hình 17.Route đăng ký

2. API Đăng nhập

- Endpoint: POST /api/login/stu_id/password
- Input: stu_id, password (từ URL Path)
- Xử lý:
 - Tìm user theo stu_id.
 - So sánh password.
- Output: Thông báo success hoặc thông báo lỗi sai mật khẩu/không tìm thấy user.

```
async def login(stu_id:str = Path(...), password:str = Path(...)):
    thisUser = await collection.find_one({"stu_id": stu_id})

    if not thisUser:
        return {
            "status": "fail",
            "message": "User not found"
        }
    else:
        if thisUser["password"] != password:
            return {
                "status": "fail",
                "message": "Password is incorrect"
            }
        else:
            return {
                "status": "success",
                "message": "Login successfully",
            }
```

Hình 18.Route đăng nhập

3. API Thêm môn học

- Endpoint: POST /api/add_subject/stu_id
- Input: subject_id, subject_name (body) + stu_id (path)
- Xử lý
 - Kiểm tra user tồn tại.
 - Kiểm tra xem user đã tạo môn học trùng tên chưa.
 - Nếu chưa, thì push môn học mới vào danh sách trong MongoDB.

```

async def add_subject(
    stu_id: str = Path(...),
    subject_id: str = Body(...),
    subject_name: str = Body(...)
):
    new_subject = {
        "subject_id": subject_id,
        "subject_name": subject_name,
        "messages": []
    }
    thisUser = await collection.find_one({"stu_id": stu_id})
    if not thisUser:
        return {"status": "fail",
                "message": "User not Found"}
    else:
        for subject in thisUser.get("subject", []):
            if subject["subject_name"] == subject_name:
                return {
                    "status": "success",
                    "message": "Subject already exists"
                }

    result = await collection.update_one(
        {"stu_id": stu_id},
        {"$push": {"subject": new_subject}}
    )
    if result.modified_count == 0:
        return {"status": "fail", "message": "User not found or subject not added"}
    return {"status": "success", "message": "Subject added successfully"}

```

Hình 19.Route thêm môn học

4. API Lấy danh sách môn học

- Endpoint: GET /api/get_subjects/stu_id
- Input: stu_id
- Xử lý:
 - Tìm user theo stu_id
 - Trả về danh sách subject_name trong danh sách subject

```

async def get_subjects(stu_id :str = Path(...)):

    thisUser = await collection.find_one({"stu_id": stu_id})
    subjects = []
    if not thisUser:
        return {
            "status": "fail",
            "message": "User not found"
        }
    for subject in thisUser.get("subject", []):
        subjects.append(subject["subject_name"])
    return {
        "status": "success",
        "subjects": subjects
    }

```

Hình 20.Route lấy danh sách môn học

5. API Upload file PDF

- Endpoint: POST /api/post_file
- Input: file PDF, subject_name, stu_id (form-data)
- Xử lý:
 - Lưu file vào thư mục /data của backend.
 - Gọi hàm seed_data_milvus_ollama() để tự động tách text + chunking + embedding và indexing vào Milvus.
- Output: Trạng thái thành công hoặc lỗi.

```

async def post_file(file: UploadFile = File(...), subject_name: str = Form(...), stu_id: str = Form(...)):
    if not file.filename.endswith(".pdf"):
        raise HTTPException(status_code=400, detail="Chỉ hỗ trợ file PDF.")

    save_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), "../data"))
    os.makedirs(save_dir, exist_ok=True)
    file_path = os.path.join(save_dir, file.filename)

    collection = subject_name + stu_id

    try:
        with open(file_path, "wb") as buffer:
            shutil.copyfileobj(file.file, buffer)

        print(f"File saved to {file_path}")

        await run_in_threadpool(
            seed_data_milvus_ollama,
            file_path,
            "bge-m3:567m",
            collection,
            "http://milvus-standalone:19530"
        )

        return {"message": "Upload thành công", "filename": file.filename}

    except Exception as e:
        import traceback
        print(traceback.format_exc())
        raise HTTPException(status_code=500, detail=f"Lỗi khi lưu file: {str(e)}")

```

Hình 21.Route upload file

6. API Gửi truy vấn đến AI (LLM)

- Endpoint: POST /api/get_response/stu_id/subject_name
- Input: input (form), stu_id, subject_name (path)
- Xử lý
 - Lấy retriever từ Milvus theo môn học (subject+stu_id).
 - Lấy message history từ MongoDB.
 - Tạo chat history để cung cấp ngữ cảnh cho LLM.
 - Gọi truy vấn và nhận kết quả từ LLM.
 - Cập nhật câu hỏi-và-trả lời vào MongoDB.

- Output: Phản hồi của LLM.

```

async def get_response(
    stu_id: str = Path(...),
    subject_name: str = Path(...),
    input: str = Form(...)
):
    retriever = get_retriever(subject_name+stu_id)
    agent_executor = get_llm_and_agent(retriever)

    result = await get_message(stu_id, subject_name)
    mongo_mess_history = result.get("messages", [])
    thisUserMessage = {"role": "user", "content": input}
    input_chatbot = mongo_mess_history[-10:] # Giới hạn số dòng lịch sử
    chat_history = []
    for msg in input_chatbot:
        if msg["role"] == "user":
            chat_history.append(HumanMessage(content=msg["content"]))
        elif msg["role"] == "assistant":
            chat_history.append(AIMessage(content=msg["content"]))

    response = agent_executor.invoke({
        "input": input,
        "chat_history": chat_history
    })

    thisAssistantMessage = {"role": "assistant", "content": response["output"]}
    mongo_mess_history.append(thisUserMessage)
    mongo_mess_history.append(thisAssistantMessage)

    await update_message(stu_id, subject_name, mongo_mess_history)

    return {
        "status": "success",
        "message": response["output"]
    }

```

Hình 22.Route lấy phản hồi

7. API Lấy lịch sử hội thoại theo môn

- Endpoint: GET /api/get_message/stu_id/subject_name
- Input: stu_id, subject_name
- Xử lý:
 - Tìm user theo stu_id.
 - Trả về danh sách messages trong subject.

```

async def get_message(stu_id :str = Path(...), subject_name: str = Path(...)):
    thisUser = await collection.find_one({"stu_id": stu_id})
    if not thisUser:
        return {
            "status": "fail",
            "message": "User not found"
        }
    for subject in thisUser.get("subject", []):
        if subject["subject_name"] == subject_name:
            return {
                "status": "success",
                "messages": subject.get("messages", [])
            }
    return {
        "status": "fail",
        "message": "Subject not found"
    }

```

Hình 23.Route lấy lịch sử hội thoại

8. API Cập nhật lịch sử hội thoại

- Endpoint: POST /api/update_message/stu_id/subject_name
- Input: List[Message] (body)
- Xử lý:
 - Cập nhật danh sách messages mới cho môn học trong MongoDB.

```

async def update_message(stu_id :str = Path(...), subject_name: str = Path(...), messages: List[Message] = Body(...)):
    message_json = jsonable_encoder(messages)
    result = await collection.find_one_and_update(
        {"stu_id": stu_id, "subject.subject_name": subject_name},
        {"$set": {"subject.$messages": message_json}}
    )
    if result:
        return {
            "status": "success",
            "message": "Message updated successfully"
        }
    else:
        return {
            "status": "fail",
            "message": "Error on updating message"
        }

```

Hình 24.Route update lịch sử hội thoại

9. API Xóa môn học

- Endpoint: POST /api/delete_subject/stu_id/subject_name
- Xử lý:
 - Xóa subject trong MongoDB.
 - Gọi hàm delete_collection() để xóa vector collection tương ứng trong Milvus.

```
async def delete_subject(stu_id : str = Path(...), subject_name:str = Path(...)):
    try:
        await collection.update_one(
            {"stu_id": stu_id},
            {"$pull": {"subject": {"subject_name": subject_name}}}
        )

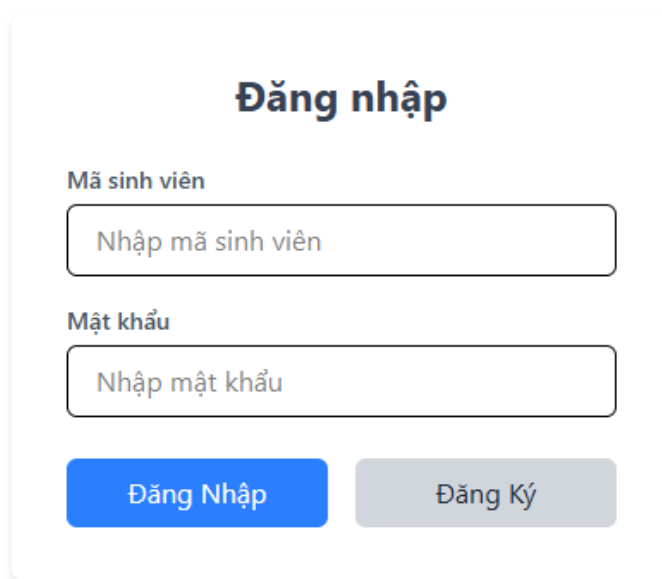
        collection_name = subject_name + stu_id
        delete_collection(collection_name)
        return {
            "status":"success",
            "message":"Delete subject successfully"
        }

    except Exception as e:
        return{
            "status":"fail",
            "message":"Server error"
        }
```

Hình 25.Route xóa môn học

3.7 Thiết kế giao diện người dùng

1. Trang đăng nhập, đăng ký: đơn giản gồm các ô nhập liệu và 2 nút đăng nhập, đăng ký



The image shows a login and registration form titled "Đăng nhập" (Login). It contains two input fields: "Mã sinh viên" (Student ID) and "Mật khẩu" (Password). Below the input fields are two buttons: "Đăng Nhập" (Login) and "Đăng Ký" (Register).

Đăng nhập

Mã sinh viên

Nhập mã sinh viên

Mật khẩu

Nhập mật khẩu

Đăng Nhập Đăng Ký

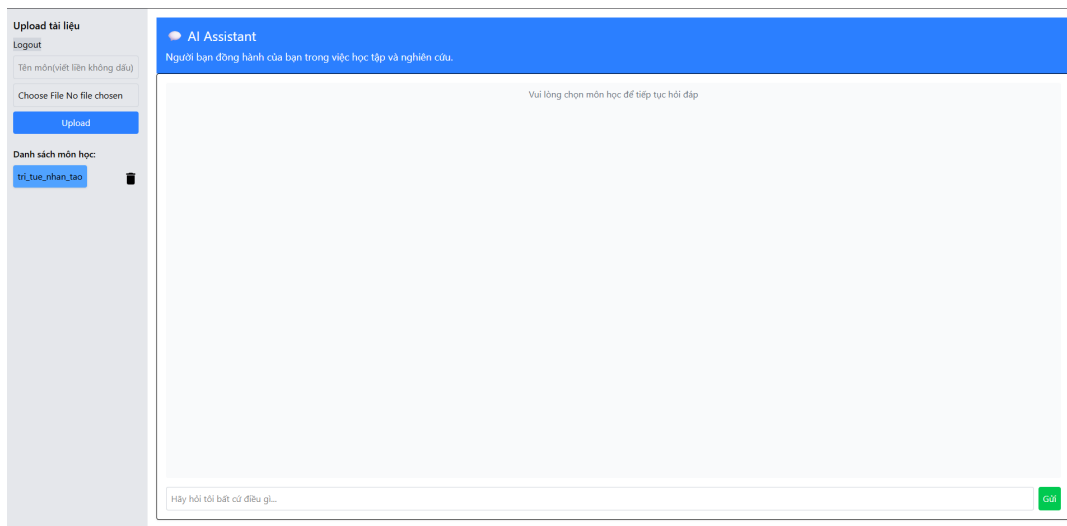
Hình 26.Trang đăng nhập

The image shows a registration form titled "Đăng ký". It contains three input fields: "Họ và tên" (Name) with placeholder text "Nhập họ và tên", "Mã sinh viên" (Student ID) with placeholder text "Nhập mã sinh viên", and "Mật khẩu" (Password) with placeholder text "Nhập mật khẩu". Below the fields are two buttons: "Đăng nhập" (Login) in grey and "Đăng ký" (Register) in blue.

Hình 27. Trang đăng ký

2. Trang chính: gồm Side Bar và Chat Box

- Side bar: Nút upload tài liệu theo tên môn học và danh sách các môn học đã hỏi đáp
- Chat Box: Nơi đặt câu hỏi và xem câu trả lời.



Hình 28. Giao diện chính

KẾT CHUƠNG:

Chương này đã mô tả chi tiết quá trình phân tích và thiết kế hệ thống, từ yêu cầu người dùng đến cấu trúc logic và các thành phần phần mềm cần triển khai. Các sơ đồ và mô hình thiết kế đã thể hiện rõ cách các phần tử của hệ thống phối hợp với nhau để xử lý yêu cầu người dùng một cách hiệu quả. Những thiết kế này là cơ sở

để triển khai hệ thống thực tế trong chương tiếp theo – Chương 4, nơi các bước cài đặt và triển khai sẽ được thực hiện cụ thể.

CHƯƠNG 4. CÀI ĐẶT VÀ TRIỂN KHAI HỆ THỐNG

TỔNG QUAN:

Sau khi hoàn thiện thiết kế trong chương 3, chương 4 sẽ trình bày chi tiết quá trình hiện thực hóa hệ thống chatbot bằng cách cài đặt và triển khai trên cả môi trường local và đám mây. Việc triển khai thực tế đóng vai trò kiểm chứng hiệu quả của mô hình thiết kế cũng như khả năng hoạt động ổn định của hệ thống. Chương này gồm hai nội dung chính: (i) cài đặt hệ thống trên môi trường local bằng Docker, (ii) triển khai trên AWS EC2 nhằm đảm bảo khả năng truy cập từ xa, cấu hình và tối ưu hệ thống để đảm bảo hiệu suất khi vận hành.

4.1 Cài đặt môi trường local với docker

Trong chương này, nhóm em trình bày các bước cài đặt, triển khai và chạy hệ thống chatbot hỗ trợ học tập dựa trên các mô hình ngôn ngữ lớn (LLM), sử dụng FastAPI, MongoDB, Milvus và Langchain.

1. Cài đặt Docker

- Cài đặt Docker Desktop tại: <https://www.docker.com/products/docker-desktop/>
- Kích hoạt WSL2 và cài Ubuntu WSL nếu chưa có.

2. Build dự án frontend

- Di chuyển vào thư mục giao diện: `cd src/frontend`
- Chạy lệnh build: `npm run build`

3. Build và chạy hệ thống với docker-compose

- Di chuyển về thư mục gốc: `cd ../../`
- Vào thư mục production: `cd production`
- Build và chạy hệ thống: `docker compose -p chatbot up -d`
- Pull model bge-m3:567m
 - Truy cập container ollama: `docker exec -it chatbot-ollama-1 /bin/bash`
 - Pull model về: `ollama pull bge-m3:567m`

4.2 Triển khai hệ thống trên AWS EC2

1. Cài đặt swap RAM

Do EC2 (gói Free Tier) chỉ có 1GB RAM, không đủ chạy toàn bộ hệ thống, nhóm tiến hành tạo RAM ảo (swap RAM) để tăng hiệu năng.

- Tạo file swap 5GB: `sudo fallocate -l 5G /swapfile`

- Cấp quyền file: `sudo chmod 600 /swapfile`
- Định dạng swap: `sudo mkswap /swapfile`
- Kích hoạt swap: `sudo swapon /swapfile`
- Tự động bật sau reboot: `echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab`

2. Pull image model

Do quá trình build mô hình và Ollama trên EC2 mất nhiều thời gian và tài nguyên, nhóm đã build sẵn image chứa FastAPI và mô hình rồi push lên DockerHub.

- Pull image từ DockerHub: `docker pull hminh729/chatbot-model`

3. Cập nhật docker-compose

Trong file `docker-compose.yml`, phần cấu hình service model được chỉ định dùng image có sẵn:

```
services:
  model:
    image: hminh729/chatbot-model:latest
    ports:
      - 8000:8000
    expose:
      - 8000
    networks:
      - app-network
    # environment:
    #   - MODEL_NAME=your_model_name
    #   - MODEL_VERSION=1.0.0
```

Hình 29. Sửa file `docker-compose.yml`

KẾT CHƯƠNG:

Chương này đã trình bày chi tiết các bước cài đặt và triển khai hệ thống chatbot thành công trên cả môi trường local và nền tảng AWS EC2. Việc sử dụng Docker và Docker Compose đã giúp tự động hóa và đồng nhất quy trình triển khai, trong khi cấu hình swap RAM và sử dụng image model sẵn có đã giúp tối ưu tài nguyên EC2. Nhờ đó, hệ thống hoạt động ổn định ngay cả trên gói Free Tier. Chương tiếp theo – Chương 5 – sẽ đánh giá hiệu quả của hệ thống và đưa ra những kết luận và hướng phát triển tương lai.

CHƯƠNG 5. ĐÁNH GIÁ VÀ KẾT QUẢ THỰC NGHIỆM

TỔNG QUAN:

Chương 4 đã trình bày chi tiết quá trình triển khai hệ thống, giúp hệ thống chatbot hoạt động được trong thực tế. Chương 5 này sẽ đánh giá hiệu quả của hệ thống thông qua các tiêu chí như khả năng truy vấn, chất lượng phản hồi, thời gian đáp ứng, và độ ổn định khi chạy trong điều kiện giới hạn tài nguyên. Ngoài ra, chương này cũng thảo luận những điểm mạnh, điểm hạn chế và đề xuất một số hướng phát triển trong tương lai để cải thiện hệ thống.

5.1 Kịch bản thử nghiệm

Các chức năng chính của hệ thống được kiểm tra bao gồm:

1. Đăng ký và đăng nhập tài khoản người dùng.
2. Thêm môn học và upload tài liệu (PDF).
3. Xử lý embedding tài liệu và lưu trữ trong Milvus.
4. Tạo truy vấn và nhận câu trả lời dựa trên nội dung tài liệu.
5. Lưu và truy xuất lịch sử trò chuyện từ MongoDB.
6. Tương tác với chatbot qua giao diện frontend đơn giản.

5.2 Kết quả thực nghiệm

1. Hệ thống hoạt động ổn định trên môi trường local.
2. Trên AWS EC2 còn hạn chế về mặt phần cứng nhưng cũng đã hoạt động được đầy đủ chức năng
3. Tốc độ phản hồi khi đặt câu hỏi trung bình từ 2–5 giây.
4. Mô hình trả lời chính xác các câu hỏi có nội dung nằm trong tài liệu đã tải lên.
5. Giao diện đơn giản, dễ thao tác.

5.3 Đánh giá hệ thống

1. Ưu điểm:
 - Sử dụng RAG giúp truy xuất thông tin chính xác theo ngữ cảnh.
 - Kiến trúc microservice linh hoạt, dễ mở rộng.
 - Sử dụng Docker giúp tái triển khai nhanh và dễ dàng.
2. Hạn chế:

- Chưa xử lý tốt với các tài liệu có cấu trúc phức tạp (nhiều biểu đồ, bảng biểu).
- Tốn nhiều RAM khi chạy mô hình LLM, cần tối ưu nếu triển khai diện rộng.

5.4 Hướng phát triển hệ thống

Để hệ thống chatbot hỗ trợ học tập ngày càng hoàn thiện và phù hợp hơn với nhu cầu thực tế của sinh viên, trong tương lai, nhóm đề xuất một số hướng phát triển sau:

- Tích hợp nền tảng mạng xã hội: Kết nối chatbot với các nền tảng như Facebook Messenger hoặc Zalo qua webhook và công cụ như n8n để hỗ trợ người dùng truy cập nhanh chóng và tiện lợi hơn.
- Gợi ý lộ trình học cá nhân hóa: Dựa vào nội dung giáo trình và lịch sử tương tác, chatbot có thể đưa ra lộ trình học phù hợp cho từng sinh viên.
- Tự động nhắc lịch học: Quét ảnh thời khóa biểu hoặc nhập tay để chatbot gửi thông báo nhắc lịch học, buổi học sắp tới và kiến thức cần ôn tập.
- Tổng hợp nội dung sau mỗi buổi học: Dựa vào tiến độ học và tài liệu đã cung cấp, hệ thống có thể tạo bản tóm tắt nội dung đã học để người dùng dễ dàng ôn lại.
- Tạo câu hỏi trắc nghiệm ôn tập: Tự động sinh câu hỏi dựa trên tài liệu để sinh viên luyện tập sau mỗi chương.

KẾT CHƯƠNG:

Chương này đã đánh giá toàn diện hệ thống chatbot sau khi triển khai, cho thấy hệ thống đạt được mục tiêu hỗ trợ người học truy xuất thông tin từ tài liệu một cách nhanh chóng và chính xác. Mặc dù còn một số hạn chế về hiệu suất và phạm vi dữ liệu, hệ thống đã chứng minh được tính khả thi và tiềm năng mở rộng. Một số hướng phát triển trong tương lai bao gồm: tự động tạo câu hỏi trắc nghiệm sau mỗi buổi học, tích hợp với hệ thống nhắc lịch học, và cải thiện giao diện người dùng. Đây sẽ là tiền đề cho việc tiếp tục nghiên cứu và nâng cấp hệ thống trong các dự án sau này.