ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA CÔNG NGHỆ PHẦN MỀM

ĐỒ ÁN 1

Notopia - Ứng dụng ghi chú thông minh hỗ trợ quản lý tri thức bằng biểu đồ quan hệ

GIẢNG VIÊN HƯỚNG DẪN: ThS. Trần Thị Hồng Yến

SINH VIÊN THỰC HIỆN: Trần Nguyễn Thái Bình - 23520161
Nguyễn Thái Gia Nguyễn - 23521049

TP. Hồ Chí Minh, tháng 05, năm 2026

LỜI CẢM ƠN

Để hoàn thành đồ án "Notopia - Ứng dụng ghi chú thông minh hỗ trợ quản lý tri thức bằng biểu đồ quan hệ", bên cạnh sự nỗ lực của các thành viên, chúng em đã may mắn nhận được sự đồng hành và chỉ dẫn vô cùng quý giá.

Đặc biệt, chúng em xin bày tỏ lòng tri ân sâu sắc nhất tới cô Trần Thị Hồng Yến. Không chỉ là người định hướng chuyên môn, cô còn dành tâm huyết để khích lệ, giúp chúng em tháo gỡ những nút thắt kỹ thuật và tư duy hệ thống trong suốt quá trình nghiên cứu. Những góp ý đầy tận tâm của cô chính là kim chỉ nam giúp Notopia hình thành và hoàn thiện như ngày hôm nay.

Đồng thời, nhóm xin gửi lời cảm ơn chân thành đến Khoa Công nghệ Phần mềm, trường Đại học Công nghệ Thông tin – ĐHQG TP.HCM. Cảm ơn Nhà trường đã tạo dựng một môi trường học tập hiện đại, cung cấp nền tảng tri thức và cơ sở vật chất tốt nhất để chúng em thực hiện đề tài này.

Mặc dù đã dồn hết tâm sức, song đồ án chắc chắn vẫn còn những điểm cần cải thiện. Chúng em rất mong nhận được sự chỉ dẫn và đóng góp từ quý thầy cô để nhóm có thể phát triển ứng dụng ngày một hoàn thiện hơn.

TP. Hồ Chí Minh, ngày 03 tháng 05 năm 2026

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

DANH MỤC BẢNG CHƯƠNG TRÌNH

TÓM TẮT ĐỀ TÀI

Đề tài tập trung nghiên cứu và xây dựng "Notopia - Ứng dụng ghi chú thông minh hỗ trợ quản lý tri thức bằng biểu đồ quan hệ", với mục đích giúp người dùng quản lý tri thức cá nhân trên một nền tảng web trực quan, cộng tác theo thời gian thực. Đề tài không hướng đến việc giải quyết các nhược điểm của các nền tảng sẵn có trên thị trường, mà thay vào đó tập trung nghiên cứu kiến trúc, phương pháp phát triển phần mềm, triển khai hệ thống, khai thác và sử dụng các công nghệ hiện đại.

Báo cáo trình bày các nghiên cứu, quy trình thiết kế, cài đặt và triển khai hệ thống thông qua các chương sau:

1. **Chương 1. Giới thiệu đề tài** - Giới thiệu về đề tài, mục tiêu nghiên cứu, phạm vi của báo cáo, các tính năng, công nghệ được sử dụng trong đề tài.
2. **Chương 2. Cơ sở lý thuyết** - Cơ sở lý thuyết liên quan, các công nghệ và phương pháp phát triển phần mềm được sử dụng trong đề tài.
3. **Chương 3. Phân tích và thiết kế hệ thống** - Mô tả kiến trúc hệ thống, các đặc tả use case, API, các thành phần chính của hệ thống, cơ sở dữ liệu, một số mô hình và logic của hệ thống.
4. **Chương 4. Xây dựng ứng dụng** - Trình bày kết quả thực hiện giao diện, chức năng của Web App.
5. **Chương 5. Kết luận** - Kết luận về kết quả đạt được, những thuận lợi, khó khăn, ưu điểm và hướng phát triển trong tương lai của đề tài.

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

Chương này trình bày bối cảnh thực tiễn và lý do lựa chọn đề tài, từ đó làm rõ mục tiêu, phạm vi và đối tượng nghiên cứu của dự án. Nội dung của chương nhằm giúp người đọc có cái nhìn toàn diện về định hướng nghiên cứu, cơ sở khoa học và giá trị ứng dụng thực tiễn của đề tài trước khi đi sâu vào các chương phân tích và thiết kế chi tiết ở các phần tiếp theo.

1.1. Lý do chọn đề tài

Trong kỷ nguyên bùng nổ thông tin, việc quản lý kiến thức cá nhân *(Personal Knowledge Management - PKM)* trở thành một kỹ năng thiết yếu. Các phương pháp ghi chú truyền thống theo dạng danh sách hoặc thư mục dần bộc lộ hạn chế trong việc kết nối các ý tưởng rời rạc. Lấy cảm hứng từ các ứng dụng như Notion, Obsidian, chúng em quyết định thực hiện đề tài xây dựng một nền tảng ghi chú hiện đại hỗ trợ liên kết hai chiều[1], kết hợp trực quan hóa kiến thức thành biểu đồ quan hệ.

1.2. Mục đích và mục tiêu nghiên cứu

1.2.1. Mục đích nghiên cứu

Giúp người dùng quản lý tri thức cá nhân trên nền tảng web, trực quan hoá bằng biểu đồ quan hệ *(Graph View)*, cộng tác *(collaborate)* theo thời gian thực. Ứng dụng đảm bảo tối thiểu các chức năng của một trình quản lý ghi chú *(note)* bao gồm tạo, sửa, xoá tạm thời, xoá vĩnh viễn, khôi phục ghi chú, tìm kiếm toàn văn *(full-text search)*, phân quyền truy cập vào không gian làm việc *(workspace)*.

1.2.2. Mục tiêu nghiên cứu

- Nghiên cứu các phương pháp tổ chức ghi chú dạng cấu trúc thư mục.
- Nghiên cứu cách thức liên kết các ghi chú với nhau để tạo thành mạng lưới tri thức.
- Tích hợp công nghệ hỗ trợ viết nội dung ghi chú *(sử dụng thư viện thứ ba để xử lý các tác vụ liên quan đến định dạng nội dung)* với cách quản lý các đối tượng trong hệ thống.
- Nghiên cứu cách triển khai cộng tác theo thời gian thực.
- Định danh, phân quyền người dùng đối với không gian làm việc.
- Thiết kế kiến trúc hệ thống dễ mở rộng, bảo trì và có hiệu suất cao.

1.3. Đối tượng và phạm vi nghiên cứu

1.3.1. Đối tượng nghiên cứu

Đối tượng nghiên cứu về mặt nghiệp vụ

[1]Liên hai chiều là khái niệm từ Obsidian, bao gồm outgoing link - liên kế từ một đối tượng này đến các đối tượng khác, và backlink - liên kết từ các đối tượng khác ngược lại đến đối tượng này

- Hệ thống quản lý tri thức cá nhân *(Personal Knowledge Management - PKM)*: Nghiên cứu các mô hình lưu trữ thông tin, đặc biệt là cách thức kết nối các ý tưởng rời rạc thành một mạng lưới tri thức.
- Cấu trúc dữ liệu ghi chú: Loại định dạng nội dung ghi chú Block-based với thư viện BlockNote.
- Luồng tương tác cộng tác: Nghiên cứu các cơ chế đồng bộ dữ liệu khi nhiều người cùng làm việc trên một tài liệu *(kiểu dữ liệu CRDT)* và không gian làm việc.

CRDT Conflict-free Replicated Data Type, một loại cấu trúc dữ liệu cho phép hợp nhất tự động các thay đổi từ nhiều người dùng mà không gây xung đột, rất phù hợp cho các ứng dụng cộng tác theo thời gian thực.

Đối tượng nghiên cứu về mặt kỹ thuật

- *Kiến trúc hệ thống*: Nghiên cứu và áp dụng kiến trúc Microservices kết hợp với Clean Architecture, Domain Driven Design, Event Driven Architecture đối với domain phức tạp để đảm bảo tính mở rộng.
- *Cơ chế liên kết và truy vấn dữ liệu*: Cách thức triển khai liên kết hai chiều và truy vấn quan hệ giữa các ghi chú để tạo Graph View.
- *Giao thức kết nối*: Sử dụng Rest API để giao tiếp giữa Web App với API service, SSE để cập nhật thông tin thời gian thực[2], và gRPC để giao tiếp giữa các service trong hệ thống.
- *Công nghệ tìm kiếm*: Cách tích hợp Full-text search để hỗ trợ người dùng truy xuất thông tin.
- *Hạ tầng và vận hành*: Nghiên cứu việc triển khai hệ thống với Docker, quản lý định danh, phân quyền, giám sát hệ thống *(Observation)*.

SSE Server-Sent Events, một công nghệ cho phép máy chủ gửi dữ liệu thời gian thực đến trình duyệt mà không cần thiết lập kết nối WebSocket phức tạp. Đây là một giải pháp nhẹ nhàng để cập nhật dữ liệu liên tục.

1.3.2. Phạm vi nghiên cứu

Định danh người dùng *(Identity)*

Sử dụng dịch vụ thứ bên thứ 3 để quản lý đăng ký, đăng nhập, thông qua OAuth2/ OpenID Connect, đảm bảo an toàn và dễ dàng tích hợp với các dịch vụ khác trong hệ thống.

Ghi chú *(Note)*

Quản lý không gian làm việc, sắp xếp các ghi chú theo cấu trúc thư mục, biểu diễn quan hệ giữa các ghi chú, hỗ trợ cộng tác theo thời gian thực ở góc độ lưu trữ và tổ chức thông tin.

[2]Thông tin theo thời gian thực sử dụng SSE bao gồm các sự kiện thay đổi mang tính chất tổng quát trong không gian làm việc *("metadata" thay đổi)*, khác với hệ thống hỗ trợ cộng tác sử đổi nội dung ghi chú theo thời gian thực sử dụng CRDT

Đây là phạm vi cốt lõi của hệ thống, tập trung vào việc tổ chức và lưu trữ thông tin một cách hiệu quả.

Tài liệu *(Document)*
Nội dung của ghi chú được lưu trữ dưới dạng tài liệu, sử dụng định dạng Block-based bởi thư viện BlockNote, cho phép linh hoạt trong việc trình bày và chỉnh sửa nội dung. Đồng thời, hỗ trợ cộng tác theo thời gian thực trên tài liệu. Các tệp tin đính kèm trong ghi chú thông qua giải pháp lưu trữ đối tượng *(Object Storage)*.

Phân quyền *(Authorization)*
Quản lý quyền truy cập vào không gian làm việc. Không gian làm việc có thể được chia sẻ với nhiều người dùng, mỗi người có quyền hạn khác nhau, đảm bảo an toàn và kiểm soát truy cập hiệu quả.

Tìm kiếm *(Search)*
Lắng nghe và cập nhật nội dung ghi chú khi thay đổi thông qua một "worker", cập nhật vào dịch vụ tìm kiếm bên thứ ba.

Hạ tầng *(Infrastructure)*
- Thiết kế kiến trúc hệ thống theo mô hình Microservices, giao tiếp nội bộ để tối ưu hiệu suất và message broker cho các tác vụ bất đồng bộ.
- Triển khai quy trình CI/CD, đóng gói ứng dụng vào container.
- API được thiết kế đứng sau một API Gateway.
- Xây dựng hệ thống giám sát để theo dõi vết *(tracing)*, chỉ số đo lường *(metric)* và nhật ký hệ thống *(logging)* trong môi trường phân tán.

1.4. Phương pháp nghiên cứu
Dự án áp dụng phương pháp tiếp cận kỹ thuật hệ thống, kết hợp giữa nghiên cứu lý thuyết về quản lý tri thức và triển khai thực nghiệm các công nghệ phần mềm tiên tiến.

Phương pháp thu thập và phân tích yêu cầu
Khảo sát các ứng dụng quản lý ghi chú hiện có *(Notion, Obsidian, `jackyzha0/quartz`)* để rút ra các tính năng cần thiết.

Phương pháp thiết kế và mô hình hoá hệ thống
Sử dụng UML để mô hình hoá kiến trúc hệ thống, bao gồm sơ đồ lớp *(class diagram)*, sơ đồ tuần tự *(sequence diagram)*. Sử dụng D2 để mô hình hoá sơ đồ triển khai *(deployment diagram)*, cơ sở dữ liệu quan hệ. Trong đó:
- Đối với sơ đồ tuần tự, chỉ đặc tả các use case quan trọng để làm rõ luồng tương tác giữa các thành phần chính của hệ thống.
- Đối với class diagram, chỉ tập trung vào các lớp cốt lõi liên quan đến nghiệp vụ chính của hệ thống, tránh mô hình hoá chi tiết quá mức làm rối sơ đồ.

Phương pháp phát triển API

Thiết kế API theo chuẩn RESTful, sử dụng OpenAPI 3.0 để mô tả API giữa Web App và API service, Protocol Buffers 3 cho giao tiếp giữa các service nội bộ. Áp dụng hướng tiếp cận "Contract First", đảm bảo tính nhất quán từ giai đoạn thiết kế đến triển khai.

Nhờ vào việc đặc tả này, chúng ta có thể tự động sinh mã nguồn cho client và server, giảm thiểu lỗi và tăng tốc độ phát triển.

1.5. Chức năng

Hệ thống Notopia được xây dựng như một nền tảng quản lý tri thức cá nhân, hỗ trợ người dùng tổ chức, kết nối và trực quan hóa kiến thức trên môi trường web hợp tác theo thời gian thực. Các nhóm chức năng chính bao gồm:

Tạo và quản lý không gian làm việc

Người dùng có thể tạo các không gian làm việc riêng biệt, chia sẻ với người khác và quản lý quyền truy cập một cách linh hoạt. Mỗi không gian làm việc hoạt động như một vùng lưu trữ độc lập cho các ghi chú và dự án của người dùng.

Tổ chức ghi chú theo cấu trúc thư mục

Hệ thống cho phép người dùng sắp xếp ghi chú thành các thư mục lồng nhau, tạo thành một cấu trúc dữ liệu rõ ràng. Cách tổ chức này giúp người dùng quản lý số lượng lớn ghi chú một cách hiệu quả.

Tạo các liên kết hai chiều giữa ghi chú

Người dùng có thể liên kết các ghi chú với nhau thông qua cơ chế liên kết hai chiều, tạo thành một mạng lưới tri thức động. Hệ thống tự động ghi nhận các liên kết ngược, giúp hiểu rõ mối quan hệ giữa các ý tưởng.

Soạn thảo nội dung dạng block-based

Nội dung ghi chú được tổ chức thành các khối độc lập, cho phép người dùng linh hoạt trong việc xây dựng và chỉnh sửa tài liệu. Mỗi khối có thể là văn bản, hình ảnh, mã code hoặc các loại nội dung khác.

Cộng tác theo thời gian thực

Nhiều người dùng có thể làm việc cùng một ghi chú hoặc tài liệu một cách đồng thời. Hệ thống đảm bảo các thay đổi được đồng bộ hóa tức thời và các xung đột được giải quyết tự động mà không cần can thiệp thủ công.

Trực quan hóa mối quan hệ bằng biểu đồ quan hệ

Người dùng có thể xem toàn bộ mạng lưới tri thức của mình dưới dạng biểu đồ quan hệ *(Graph View)*, giúp hiển thị các liên kết và mối quan hệ giữa các ghi chú. Biểu diễn này cung cấp một cái nhìn trực quan hơn so với cách tổ chức tuyến tính truyền thống.

Tìm kiếm nhanh chóng

Hệ thống hỗ trợ tìm kiếm toàn văn trên tất cả nội dung ghi chú, cho phép người dùng nhanh chóng tìm ra thông tin cần thiết.

Quản lý quyền truy cập và phân quyền

Chủ sở hữu không gian làm việc có thể quản lý chi tiết quyền của từng thành viên, chẳng hạn như quyền xem, chỉnh sửa hoặc xóa. Hệ thống hỗ trợ các cấp độ quyền khác nhau để đảm bảo an toàn và kiểm soát truy cập hiệu quả.

Quản lý vòng đời tài liệu

Ghi chú có thể được xóa tạm thời, di chuyển vào thùng rác hoặc xóa vĩnh viễn. Người dùng cũng có khả năng khôi phục ghi chú đã xóa tạm thời, giúp tránh mất dữ liệu không mong muốn.

1.6. Công nghệ sử dụng

Dự án Notopia ứng dụng một bộ công nghệ tiên tiến và được lựa chọn kỹ lưỡng, đảm bảo tính mở rộng, hiệu suất cao và khả năng bảo trì dài hạn.

Web App

- Framework: Next.js, được xây dựng trên nền tảng React, cung cấp server-side rendering và tối ưu hóa hiệu suất.
- State Management: Redux Toolkit, một thư viện quản lý trạng thái với DevTools integration và middleware support mạnh mẽ.
- Styling: TailwindCSS kết hợp với PostCSS cho phát triển giao diện nhanh chóng, và ShadcnUI cung cấp các component có sẵn được styled phù hợp.
- Editor Content: BlockNote, một editor block-based được xây dựng trên Tiptap, hỗ trợ tùy chỉnh cao và tích hợp dễ dàng với React.

Backend

- Identity service *(Authentik)*: Một nền tảng identity provider mã nguồn mở hỗ trợ OAuth2/OIDC, đảm bảo an toàn trong quản lý danh tính người dùng.
- `document` service: Viết bằng Typescript sử dụng framework NestJS, với kiến trúc mô-đun rõ ràng và Dependency Injection mạnh mẽ, được xây dựng với Rspack để tối ưu hiệu suất build.
- `notes` service: Viết bằng Go, một ngôn ngữ lập trình hiệu suất cao, được sử dụng với SQLC cho xử lý raw SQL tối ưu, đảm bảo truy vấn nhanh chóng đặc biệt cho các truy vấn đồ thị phức tạp.
- `authorization` service: Viết bằng Go, sử dụng thư viện Casbin, một framework authorization mã nguồn mở hỗ trợ RBAC cho phép quản lý quyền truy cập linh hoạt.
- `search-worker` worker: Viết bằng NestJS lắng nghe sự thay đổi dữ liệu, xử lý và đồng bộ nội dung với Meilisearch.

- Search Service *(Meilisearch)*: Search engine mã nguồn mở được viết bằng Rust, cung cấp tìm kiếm toàn văn nhanh chóng và dễ triển khai.

Cơ sở dữ liệu

- PostgreSQL: Hệ quản trị cơ sở dữ liệu quan hệ mạnh mẽ, tuân thủ ACID transactions và hỗ trợ các tính năng nâng cao.
- ORM/Query Builder/Query codegen: TypeORM cho NestJS service, SQLC cho Go service để tối ưu hiệu suất.

Cộng tác theo thời gian thực

- yjs: Thư viện CRDT cho phép hợp nhất tự động các thay đổi từ nhiều người dùng mà không gây xung đột.
- Hocuspocus: Máy chủ WebSocket hỗ trợ Yjs, tạo điều kiện cho đồng bộ hóa dữ liệu thời gian thực. Hocuspocus sẽ abstract yjs nên không cần quan tâm đến chi tiết triển khai của yjs.

Giao tiếp API

- REST API: Được thiết kế theo OpenAPI 3.0 cho giao tiếp giữa frontend và các dịch vụ backend.
- gRPC: Framework RPC sử dụng Protocol Buffers, cung cấp hiệu suất cao cho giao tiếp giữa các dịch vụ nội bộ.

Bên cạnh đó, các dịch vụ không thể giao tiếp qua gRPC, nhưng có SDK, hoặc có thể giao tiếp qua REST API *(ví dụ như Authentik hỗ trợ SDK cho Go và NodeJS)*.

Kiến trúc sự kiện

- Watermill: Thư viện Go cho event-driven architecture, hỗ trợ nhiều message broker khác nhau.
- Redpanda: Message broker tương thích với Kafka, được sử dụng cho streaming sự kiện phân tán.

Đối với NestJS đã hỗ trợ sẵn event-driven architecture, không cần sử dụng thêm thư viện bên ngoài hệ sinh thái NestJS.

Giám sát

- OpenTelemetry: Một tiêu chuẩn mã nguồn mở cho việc thu thập metrics, logs và traces.
- Grafana Stack: Bao gồm Prometheus cho metrics, Loki cho logs, Tempo cho distributed tracing, và Grafana cho visualization tập trung. Alloy, một agent của Grafana cho forwarding logs và metrics.

Hạ tầng

- Docker: Đóng gói ứng dụng vào container, đảm bảo tính nhất quán giữa các môi trường phát triển.

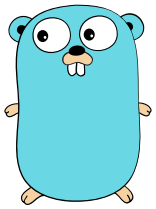
- Traefik: API Gateway hiện đại hỗ trợ routing, load balancing và auto-discovery, đứng phía trước các dịch vụ backend và Web App.
- RustFS: Giải pháp lưu trữ đối tượng mã nguồn mở, được sử dụng để lưu trữ các tệp đính kèm trong ghi chú.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Tổng quan về Go (Golang)

2.1.1. Giới thiệu

Go (còn gọi là Golang) là ngôn ngữ lập trình mã nguồn mở được phát triển bởi Google vào năm 2007 và chính thức phát hành vào năm 2009. Được thiết kế bởi Robert Griesemer, Rob Pike và Ken Thompson, Go nhằm mục tiêu tạo ra một ngôn ngữ hiệu quả, dễ học, và phù hợp cho lập trình hệ thống quy mô lớn.



Hình 1: Golang logo

Dự án sử dụng Go cùng với công cụ `goforj/wire` [1], một fork của `google/wire`, giúp dependencies injection có hỗ trợ cache fast để tối ưu thời gian build.

2.1.2. Ưu điểm

Go mang lại nhiều lợi ích trong phát triển backend:

- Hiệu suất cao, Go biên dịch thành native code và thực thi nhanh chóng tương đương C/C++
- Đồng thời dễ dàng, hỗ trợ Goroutines và Channels cho phép xử lý hàng nghìn luồng đồng thời với chi phí tài nguyên thấp
- Deployment đơn giản, chỉ cần một single binary độc lập, không phụ thuộc vào external libraries
- Cross-compilation hỗ trợ biên dịch cho nhiều nền tảng khác nhau từ một máy
- Cộng đồng lớn, Go được sử dụng trong Kubernetes, Docker, Terraform, Prometheus

2.1.3. Nhược điểm

Bên cạnh các ưu điểm, Go có một số hạn chế:

- Error handling dài dòng, pattern `if err != nil` xuất hiện lặp đi lặp lại làm code phồng phức
- Thiếu Generics trước Go 1.18, giới hạn khả năng tái sử dụng code, không support generic method.[3]
- Không có inheritance, chỉ hỗ trợ composition mà không hỗ trợ OOP truyền thống
- Learning curve với các khái niệm như interfaces và goroutines cần thời gian để hiểu rõ

[3]Generic proposal đã được phê duyệt, theo [2]

2.2. Tổng quan về TypeScript

2.2.1. Giới thiệu

TypeScript là một ngôn ngữ lập trình mã nguồn mở được phát triển và duy trì bởi Microsoft. TypeScript là một superset của JavaScript, nghĩa là mọi code JavaScript hợp lệ đều là code TypeScript hợp lệ. Được ra mắt lần đầu vào năm 2012 bởi Anders Hejlsberg (người thiết kế C#), TypeScript bổ sung hệ thống kiểu dữ liệu tĩnh trên nền tảng JavaScript để tăng độ tin cậy và khả năng duy trì của code.

TypeScript hoạt động thông qua một bước biên dịch: mã TypeScript được biên dịch thành mã JavaScript, sau đó được chạy trên JavaScript runtime (trình duyệt hoặc Node.js).

Hình 2: TypeScript logo

2.2.2. Ưu điểm

TypeScript mang lại nhiều lợi ích khi phát triển ứng dụng JavaScript:

- Type Safety, phát hiện lỗi tại compile-time thay vì runtime, giảm thiểu bugs trong quá trình phát triển
- Documentation, types tự động document code giúp developer khác hiểu code dễ dàng hơn
- JavaScript Compatibility, có thể sử dụng mọi thư viện JavaScript hiện có

2.2.3. Nhược điểm

Bên cạnh các ưu điểm, TypeScript có một số hạn chế:

- Learning Curve, cần học thêm về type system và TypeScript-specific features như decorators, generics
- Compilation Step, cần biên dịch trước khi chạy, tăng thời gian build và phức tạp hóa quy trình
- Bước compile/transpile từ TypeScript sang JavaScript được thực hiện bằng nhiều công cụ

2.2.4. Hệ sinh thái và công cụ

Dự án `microsoft/typescript-go` [3] đang phát triển một implementation của TypeScript được viết hoàn toàn bằng Go, thay vì TypeScript hiện tại được viết bằng TypeScript. Dự án này hướng tới việc cải thiện hiệu suất.

Dự án được thiết lập với monorepo, các package được chia build riêng biệt (`SWC`, `Rspack`, `tsgo`, `vite`) để tăng tốc thời gian build, Oxlint thay cho ESLint, Oxfmt thay cho Prettier để tăng tốc độ linting và formatting, CI, ngoại trừ web vì sử dụng NextJS.

2.3. Tổng quan về React và NextJS

2.3.1. Giới thiệu

Frontend của dự án được xây dựng với React [4], một thư viện JavaScript cho xây dựng user interfaces với component-based architecture. Next.js [5] được sử dụng như một framework trên React, cung cấp server-side rendering, static generation, và routing tích hợp. Redux Toolkit [6] được sử dụng cho state management.

Hình 3: React, NextJS, Redux Logo

2.3.2. React

React là thư viện JavaScript mã nguồn mở từ Meta (Facebook) cho xây dựng user interfaces [4]. React sử dụng virtual DOM để tối ưu rendering, component-based architecture cho reusability, và declarative syntax làm cho code dễ đọc hơn.

2.3.3. Next.js

Next.js là framework React được phát triển bởi Vercel [5], cung cấp Server-Side Rendering (SSR), Static Site Generation (SSG), Incremental Static Regeneration (ISR), và API routes tích hợp. Next.js giúp tối ưu hiệu suất và SEO mà không cần setup phức tạp.

2.3.4. Redux Toolkit

Redux Toolkit là state management library cho React [6], cung cấp một cách đơn giản để quản lý application state. Redux Toolkit giảm boilerplate của Redux truyền thống, hỗ trợ DevTools integration, middleware support, và immer integration cho immutable updates.

2.3.5. Ưu điểm

- React: Component reusability, large ecosystem, excellent developer experience
- Next.js: Built-in optimization, zero-config setup, excellent SEO, API routes
- Redux Toolkit: Powerful DevTools, middleware support, scalable architecture

2.3.6. Nhược điểm

- React: Learning curve cho beginners, complex state management patterns
- Next.js: Learning curve, routing may be counterintuitive, SSR adds complexity
- Redux Toolkit: More boilerplate so với lighter alternatives, steeper learning curve

2.4. Tổng quan về TailwindCSS, PostCSS, shadcnui

2.4.1. Giới thiệu

Styling được thực hiện bằng TailwindCSS [7], một CSS framework utility-first, kết hợp với PostCSS cho các biến đổi CSS nâng cao. ShadcnUI [8] cung cấp các pre-built components theo design system với styling được tích hợp sẵn.

Hình 4: TailwindCSS, ShadcnUI Logo

2.4.2. TailwindCSS

TailwindCSS là CSS framework utility-first được viết bằng PostCSS [7]. Thay vì viết CSS tùy chỉnh, developer sử dụng các utility classes được định sẵn. Approach này tăng tốc độ phát triển, tăng consistency, và giảm CSS bundle size.

2.4.3. PostCSS

PostCSS là một công cụ cho việc biến đổi CSS sử dụng JavaScript plugins. PostCSS cung cấp khả năng mở rộng, hỗ trợ modern CSS features, vendor prefixing tự động, và tích hợp tốt với TailwindCSS.

2.4.4. shadcnui

ShadcnUI cung cấp một bộ sưu tập các component React đẹp mắt, accessible, và tùy chỉnh cao [8]. Components được xây dựng trên Radix UI cho accessibility và được styled bằng TailwindCSS cho consistency với design system.

2.4.5. Ưu điểm

- TailwindCSS: Rapid development, consistent design, optimized output
- PostCSS: Extensible, powerful transformations, modern CSS support
- ShadcnUI: Beautiful components, high accessibility, full customization control

2.4.6. Nhược điểm

- TailwindCSS: HTML verbosity, steep learning curve cho CSS-first developers
- ShadcnUI: Component customization có thể phức tạp, dependency management

2.5. Tổng quan về NestJS

2.5.1. Giới thiệu

NestJS là một framework progressive Node.js được xây dựng để phát triển các ứng dụng server-side hiệu quả, đáng tin cậy và có khả năng mở rộng cao. Được phát triển bởi Kamil Myśliwiec và ra mắt lần đầu vào năm 2017, NestJS kết hợp các khái niệm từ Angular, Spring Framework và các framework hiện đại khác.

NestJS được xây dựng trên nền tảng Express.js (hoặc Fastify) và sử dụng TypeScript [9] làm ngôn ngữ chính. Framework này tổ chức code theo mô hình kiến trúc mô-đun rõ ràng, bao gồm controllers, services, middleware, guards, interceptors, và pipes, tương tự như Spring Framework của Java.

Trong dự án này, NestJS được xây dựng với:

- Rspack là bundler mới được hỗ trợ, với SWC cho quá trình transpile nhanh, đảm bảo thời gian build tối ưu

Hình 5: NestJS logo

2.5.2. Ưu điểm

NestJS mang lại nhiều lợi ích cho phát triển backend:

- TypeScript First, hỗ trợ TypeScript đầy đủ với type-safety từ đầu, không cần cấu hình thêm
- Kiến trúc rõ ràng, tổ chức code theo modules, dễ tổ chức, bảo trì và mở rộng quy mô
- Dependency Injection mạnh mẽ, quản lý dependencies tự động và dễ dàng cho testing
- Built-in Testing Utilities, hỗ trợ unit tests và e2e tests ngay từ khởi đầu
- Ecosystem phong phú, có nhiều packages chính thức như GraphQL, WebSockets, Microservices
- Scalability, phù hợp cho cả microservices lẫn monolithic applications

2.5.3. Nhược điểm

Bên cạnh các ưu điểm, NestJS có một số hạn chế:

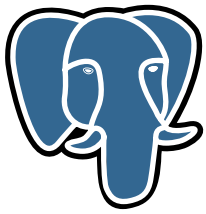
- Learning Curve cao, cần học nhiều concepts như Dependency Injection (runtime), decorators, modules, providers
- Boilerplate Code, cần viết nhiều decorators, setup code, định nghĩa providers, controllers, services
- Performance Overhead, các abstraction layers có thể ảnh hưởng hiệu suất nhẹ so với framework tối giản

2.6. Tổng quan về PostgreSQL

2.6.1. Giới thiệu

PostgreSQL (còn gọi là Postgres) là một hệ quản trị cơ sở dữ liệu quan hệ đối tượng (ORDBMS) mã nguồn mở [10], mạnh mẽ và tiên tiến. PostgreSQL được phát triển từ dự án POSTGRES tại Đại học California, Berkeley vào năm 1986 và đã phát triển hơn 35 năm với cộng đồng đóng góp tích cực.

PostgreSQL nổi tiếng vì sự tuân thủ nghiêm ngặt các chuẩn SQL, hỗ trợ ACID transactions đầy đủ, và khả năng mở rộng cao thông qua các kiểu dữ liệu tùy chỉnh và extensions như PostGIS, pgcrypto, và full-text search.



Hình 6: PostgreSQL logo

2.6.2. Ưu điểm

PostgreSQL mang lại nhiều lợi ích cho phát triển ứng dụng:

- ACID Compliance, tuân thủ đầy đủ các thuộc tính ACID (Atomicity, Consistency, Isolation, Durability) đảm bảo tính toàn vẹn dữ liệu
- Rich Data Types, hỗ trợ nhiều kiểu dữ liệu phong phú bao gồm JSON/JSONB, arrays, custom types, geometric data
- Extensibility cao, cho phép mở rộng chức năng thông qua extensions (PostGIS, pgcrypto, uuid-ossp)
- Full-Text Search tích hợp, hỗ trợ tìm kiếm toàn văn mà không cần các công cụ bên ngoài
- MVCC (Multi-Version Concurrency Control) hiệu quả, cho phép đọc và ghi đồng thời mà không cần locks
- Standards Compliance, tuân thủ đầy đủ các chuẩn SQL và hỗ trợ các features nâng cao như window functions, CTEs, lateral joins
- Open Source, miễn phí và có cộng đồng lớn hỗ trợ

2.6.3. Nhược điểm

Bên cạnh các ưu điểm, PostgreSQL có một số hạn chế:

- Performance, có thể chậm hơn MySQL trong một số workloads read-heavy đơn giản
- Memory Usage cao, sử dụng nhiều RAM hơn các RDBMS khác như MySQL
- Replication phức tạp, phức tạp hơn so với MySQL/MariaDB, đặc biệt trong setup streaming replication
- Learning Curve cao, nhiều features nâng cao cần thời gian để thành thạo như partitioning, custom types

2.7. Tổng quan về Database, ORM, và Query Patterns

2.7.1. Giới thiệu

Dự án sử dụng nhiều tool và framework cho data persistence:

- SQLC [11] cho Go, sinh type-safe SQL code
- TypeORM [12] cho NestJS, ORM object-relational mapping

2.7.2. SQLC

SQLC [11] là tool sinh Go code từ SQL queries. Thay vì viết ORM-style code, SQLC cho phép viết SQL trực tiếp và tự động sinh type-safe Go functions.

2.7.3. TypeORM

TypeORM [12] là ORM mã nguồn mở cho TypeScript, hỗ trợ multiple database backends (PostgreSQL [10], MySQL, SQLite, Oracle, v.v.). TypeORM cung cấp decorator-based API, tương thích với tốt NestJS. Đặc biệt, TypeORM tính đến thời điểm hiện tại sắp ra phiên bản 1.0.0 sau 9 năm phát triển[4]

2.7.4. Ưu điểm

- SQLC: Type safety, hiệu năng cao nhờ vào SQL thuần, được check syntax SQL từ lúc generate
- TypeORM: Abstraction cao, dễ sử dụng, tích hợp tốt với NestJS, hỗ trợ nhiều database, có cộng đồng lớn và nhiều tài liệu tham khảo

2.7.5. Nhược điểm

- SQLC: Không hỗ trợ dynamic queries mà phải nhờ plugin hỗ trợ dyanmic filter, phải viết SQL thủ công, gặp khó khăn với khả năng syntax check đối với recursive query và CTE
- TypeORM: Abstraction có thể gây performance overhead, có thể gặp vấn đề với complex queries và migrations, thiết lập script TypeORM bằng typescript phức tạp, chạy bằng tsx [13]

[4]Theo dõi tại Github issue https://github.com/typeorm/typeorm/issues/11819

2.8. Tổng quan về Yjs

2.8.1. Giới thiệu

Yjs là một thư viện CRDT (Conflict-free Replicated Data Type) được viết bằng JavaScript, thiết kế để hỗ trợ real-time collaboration trên nhiều nền tảng. CRDT cho phép các thay đổi từ nhiều người dùng được tự động hợp nhất mà không cần xung đột, làm cho Yjs trở thành lựa chọn lý tưởng cho các ứng dụng collaborative.

Yjs được sử dụng trong BlockNote [14] và nhiều editor khác để cung cấp khả năng collaborative editing. Thư viện này có thể hoạt động với các transport khác nhau như WebSocket, qua các dịch vụ như Hocuspocus [15].

2.8.2. Ưu điểm

Yjs mang lại nhiều lợi ích cho phát triển collaborative:

- CRDT-Based Merging, xung đột được giải quyết tự động mà không cần quản lý conflict manual
- Người dùng có thể tiếp tục làm việc offline và dữ liệu sẽ đồng bộ khi quay lại online
- Framework Agnostic, Yjs hoạt động độc lập với UI framework, có thể tích hợp với bất kỳ framework nào
- Được tối ưu cho hiệu suất cao, hỗ trợ xử lý các thay đổi lớn một cách hiệu quả
- Hỗ trợ nhiều binding và integration với các tool khác như Tiptap, Monaco Editor, CodeMirror
- Cung cấp khái niệm “awareness” để hiển thị con trỏ và lựa chọn của người dùng khác

2.8.3. Nhược điểm

Bên cạnh các ưu điểm, Yjs có một số hạn chế:

- Complexity, CRDT là một khái niệm phức tạp, cần hiểu rõ để tối ưu hiệu suất
- Memory Usage, lưu trữ lịch sử của tất cả thay đổi có thể tiêu thụ bộ nhớ
- Network Bandwidth, đồng bộ hóa có thể tạo ra lưu lượng network lớn với các thay đổi tần suất cao
- Learning Curve, cần thời gian để hiểu cách sử dụng và cách tích hợp với ứng dụng

2.9. Tổng quan về BlockNote

2.9.1. Giới thiệu

BlockNote là một thư viện editor được xây dựng trên nền tảng Tiptap và ProseMirror [14]. BlockNote cung cấp một công cụ soạn thảo văn bản phong phú với kiến trúc block-based tương tự như Notion, cho phép người dùng xây dựng các khối nội dung một cách linh hoạt.

Hình 7: BlockNote logo

BlockNote được thiết kế để dễ tích hợp vào các ứng dụng React *(Phần 2.3.2)*, với API rõ ràng và khả năng tùy chỉnh cao, tương thích với màu sắc của shadcnui *(Phần 2.4.4)* . Thư viện này hoạt động dựa trên ProseMirror, một editor framework mạnh mẽ và có cấu trúc rõ ràng. Có thể hình dung ProseMirror như một bộ công cụ xây dựng editor, trong khi BlockNote là một implementation cụ thể, dễ dàng sử dụng nhanh.

BlockNote là một hệ sinh thái mã nguồn mở hoàn toàn, miễn phí sử dụng công cộng. Chỉ riêng các gói thư viện `@blocknote/xl-*` có giấy phép copyleft, yêu cầu mua giấy phép nếu sử dụng trong sản phẩm mã nguồn đóng, hoặc thương mại.

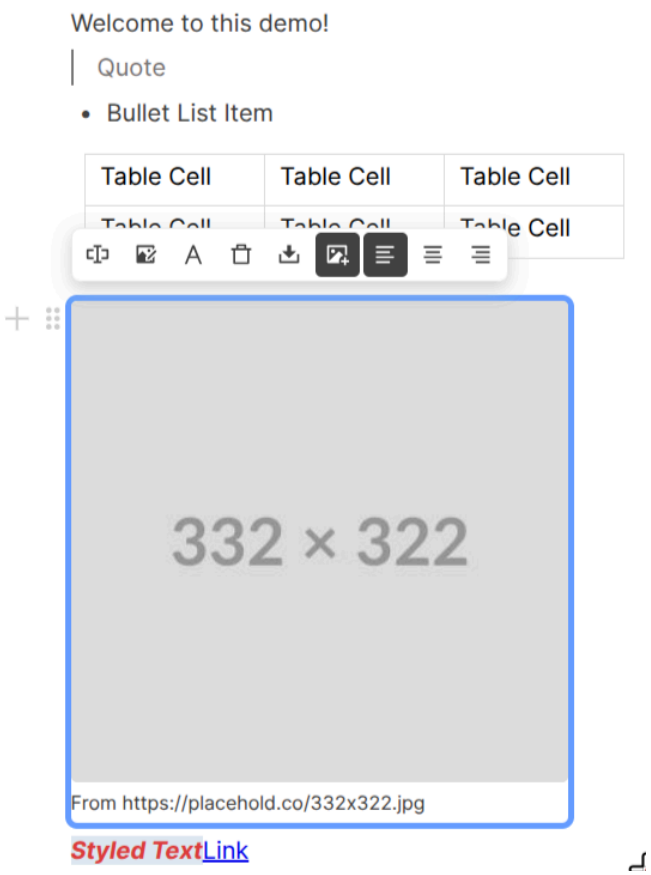
2.9.2. Model dữ liệu của BlockNote

Model dữ liệu của BlockNote được tổ chức thành các block, mỗi block đại diện cho một phần nội dung riêng biệt, như đoạn văn, hình ảnh, bảng,… Mỗi block có một cấu trúc dữ liệu riêng, bao gồm loại block, nội dung và các thuộc tính liên quan.

Dưới đây là ví dụ về cấu trúc dữ liệu của một block trong BlockNote:

```
[
  {
    "type": "paragraph",
    "content": "Welcome to this demo!"
  },
  {
    "type": "quote",
    "content": "Quote"
  },
  {
    "type": "bulletListItem",
    "content": "Bullet List Item"
  },
  {
    "type": "table",
    "content": {
      "type": "tableContent",
      "rows": [
        {
          "cells": ["Table Cell", "Table Cell", "Table Cell"]
        },
        {
          "cells": ["Table Cell", "Table Cell", "Table Cell"]
        }
      ]
    }
  },
  {
    "type": "image",
    "props": {
      "url": "https://placehold.co/332x322.jpg",
      "caption": "From https://placehold.co/332x322.jpg"
    }
  },
  {
    "type": "paragraph",
    "content": [
      {
        "type": "text",
        "text": "Styled Text",
        "styles": {
          "bold": true,
          "italic": true,
          "textColor": "red",
          "backgroundColor": "blue"
        }
      },
      {
        "type": "link",
        "content": "Link",
        "href": "https://www.blocknotejs.org"
      }
    ]
  }
]
```

Chương trình 1: Ví dụ về cấu trúc dữ liệu của một block trong BlockNote



Hình 8: Ví dụ về cấu trúc dữ liệu của một block trong BlockNote

2.9.3. Ưu điểm

BlockNote mang lại nhiều lợi ích cho phát triển editor:

- Block-based Architecture, tổ chức nội dung thành các khối độc lập, dễ quản lý và tùy chỉnh
- Tiptap Integration, được xây dựng trên Tiptap, kế thừa tất cả tính năng mạnh mẽ của framework này
- Modular Design, dễ dàng thêm hoặc xóa các tính năng thông qua hệ thống extension
- React-First, được thiết kế tối ưu cho React, với hooks và component-based API
- Real-Time Collaboration Ready, hỗ trợ tích hợp Yjs [16] và các giải pháp CRDT khác
- Rich Content Support, hỗ trợ nhiều loại nội dung: text, image, video, code blocks, tables,…

2.9.4. Nhược điểm

Bên cạnh các ưu điểm, BlockNote có một số hạn chế:

- Learning Curve, cần thời gian để hiểu kiến trúc block-based và cách tùy chỉnh, khai thác tích hợp với các đối tượng của dự án
- Browser Compatibility, một số tính năng nâng cao có thể yêu cầu trình duyệt hiện đại

2.10. Tổng quan về OpenAPI

2.10.1. Giới thiệu

OpenAPI là một specification cho mô tả HTTP APIs theo cách chuẩn hóa. OpenAPI cho phép sinh code từ contracts, tạo điều kiện cho contract-first development, và tự động tạo documentation.

OpenAPI *(trước đây được gọi là Swagger)* là một format standardized cho mô tả RESTful APIs. OpenAPI specification định nghĩa endpoints, parameters, request/response schemas, và error codes theo cách machine-readable.

Hình 9: OpenAPI, Redocly, Scalar Logo

2.10.2. Tooling Ecosystem

- Redocly [17]: Tool cho OpenAPI specification, trong dự án sử dụng làm language server, linter, bundle các file nhỏ thành một file lớn
- Scalar [18]: Tool cho OpenAPI specification, tương đối giống redocly, trong dự án sử dụng để render website documentation từ OpenAPI spec
- `heyapi/openapi-ts` [19] sinh TypeScript types từ OpenAPI spec
- `oapi-codegen` [20] sinh Go code từ OpenAPI spec, hỗ trợ HTTP API generation
- OpenAPI Generator [21]: Tool đa ngôn ngữ cho code generation từ OpenAPI spec, hỗ trợ nhiều languages và frameworks, trong dự án sử dụng generator `typescript-nestjs-server` [22]

2.10.3. Contract-First Development

Contract-first approach định nghĩa API contracts trước khi implement logic, đảm bảo tính consistency, tạo điều kiện cho parallel development, và giảm integration issues.

2.10.4. Ưu điểm

- Clear Contracts, API contracts rõ ràng được define trước implementation
- Parallel Development, frontend và backend có thể phát triển song song
- Type Safety, code generation tạo type-safe clients và servers
- Documentation, contracts tự động documentation
- Tooling Ecosystem, rộng rãi tooling hỗ trợ OpenAPI

2.10.5. Nhược điểm

- Specification Maintenance, specifications cần được updated khi API thay đổi
- Setup Complexity, code generation pipeline yêu cầu cấu hình
- Learning Curve, cần học OpenAPI specification format
- Tooling Overhead, cần maintain generation scripts và tooling

2.11. Tổng quan về gRPC

2.11.1. Giới thiệu

GRPC là một framework RPC hiện đại được phát triển bởi Google [23], sử dụng Protocol Buffers cho định nghĩa service và HTTP/2 cho communication. GRPC được thiết kế để cung cấp hiệu suất cao, latency thấp, và tích hợp tốt với distributed systems.

Hình 10: gRPC logo

2.11.2. gRPC Features

- High Performance, Protocol Buffers và HTTP/2 cho throughput cao
- Bidirectional Streaming, hỗ trợ streaming từ client-to-server và server-to-client
- OpenTelemetry Support [24], hỗ trợ native tracing với OpenTelemetry
- Type Safety, Protocol Buffers cung cấp strong typing
- Multi-Language, code generation cho nhiều ngôn ngữ

2.11.3. gRPC Tooling

- Buf [25]: Build system cho Protocol Buffers, hỗ trợ remote gen không cần cài dependencies, cung cấp code generation, linting, và breaking change detection
- GRPC Go [26]: Implementation gRPC cho Go, cung cấp high-performance gRPC server và client
- ts-proto [27]: Code generator cho TypeScript, tương thích với NestJS

2.11.4. Ưu điểm

- High Performance, Protocol Buffers và HTTP/2 cho throughput cao
- Type Safety, Protocol Buffers cung cấp strong typing
- Streaming Support, native bidirectional streaming support
- Observability, tích hợp tốt với OpenTelemetry
- Multi-Language, code generation cho nhiều ngôn ngữ

2.11.5. Nhược điểm

- Learning Curve, cần học Protocol Buffers và gRPC concepts
- Browser Compatibility, gRPC yêu cầu HTTP/2 support (limited browser support)
- Debugging Complexity, binary format khó debug so với text-based protocols
- Tooling Setup, yêu cầu setup code generation pipeline

2.12. Tổng quan về Traefik

2.12.1. Giới thiệu

Traefik là một API gateway mã nguồn mở [28], hiện đại, được viết bằng Go. Traefik được thiết kế để tự động phát hiện và kết nối các dịch vụ, loại bỏ nhu cầu cấu hình thủ công. Traefik hỗ trợ OpenTelemetry *(Phần 2.17)* cho distributed tracing, cho phép quan sát performance toàn bộ request flow.

Traefik hoạt động tốt trong các môi trường container-orchestrated như Docker, Kubernetes, Docker Swarm và cũng có thể chạy standalone cho các ứng dụng non-containerized.

Hình 11: Traefik Logo

Dự án sử dụng `traefik-plugins/traefik-jwt-plugin` [29] để xác minh request đến các API endpoint, thông tin được chuyển đổi sang request headers giúp cho các service không cần phải thực hiện quá trình xác minh riêng.

2.12.2. Ưu điểm

Traefik mang lại nhiều lợi ích cho phát triển API gateway:

- Auto-Discovery, tự động phát hiện services không cần cấu hình thủ công
- OpenTelemetry Integration, hỗ trợ native OTLP cho distributed tracing
- Container-Native, thiết kế tối ưu cho Docker và Kubernetes, hỗ trợ GatewayAPI của Kubernetes
- Middleware System, hỗ trợ middleware extensible cho custom logic
- Dynamic Configuration, cấu hình có thể thay đổi runtime mà không cần restart
- Multiple Protocol Support, hỗ trợ HTTP, HTTPS, gRPC, WebSocket
- Performance, được viết bằng Go, cung cấp hiệu suất cao và resource efficiency
- Active Development, dự án được duy trì tích cực với feature mới thường xuyên

2.12.3. Nhược điểm

Bên cạnh các ưu điểm, Traefik có một số hạn chế:

- Learning Curve, cần học provider concept và configuration syntax
- Configuration Complexity, cấu hình nâng cao có thể trở nên phức tạp
- Community Support, nhỏ hơn Nginx, ít plugin third-party có sẵn
- Debugging, khó khăn trong debugging khi routing rules không hoạt động đúng
- Ecosystem, ít integration third-party so với các gateway khác

2.13. Tổng quan về Casbin

2.13.1. Giới thiệu

Casbin là một framework authorization mã nguồn mở mạnh mẽ và linh hoạt [30], hiện thuộc Apache Software Foundation. Casbin hỗ trợ các mô hình kiểm soát truy cập như ACL, RBAC, ABAC, và các biến thể khác. Framework này cho phép định nghĩa các rule authorization một cách khai báo thông qua cấu hình, thay vì hardcode logic kiểm soát.

Hình 12: Casbin Logo

Casbin được thiết kế để hoạt động với nhiều ngôn ngữ lập trình, bao gồm Go, Java, Python, Node.js và hơn thế nữa, giúp đảm bảo tính nhất quán trong authorization logic trên toàn bộ hệ sinh thái.

So với các giải pháp authorization khác như OPA, SpiceDB *(Google Zanzibar opensource)*, Casbin tập trung vào sự đơn giản và hiệu quả, cung cấp một cách tiếp cận.

2.13.2. Ưu điểm

Casbin mang lại nhiều lợi ích cho phát triển authorization:

- Flexible Models, cùng một kiểu biểu diễn cơ sở dữ liệu nhưng có thể hỗ trợ nhiều mô hình kiểm soát từ đơn giản (ACL) đến phức tạp (ABAC)
- Declarative Configuration, định nghĩa rules thông qua file cấu hình thay vì code
- Multi-Language Support, có implementation cho nhiều ngôn ngữ lập trình
- High Performance, được tối ưu cho xử lý nhanh và kiểm tra permission hiệu quả
- Persistence, hỗ trợ lưu trữ policies trong database hoặc file
- Extensible, hỗ trợ custom matcher và effect, cho phép mở rộng chức năng

2.13.3. Nhược điểm

Bên cạnh các ưu điểm, Casbin có một số hạn chế:

- High Learning Curve, cần thời gian để hiểu các mô hình RBAC, ABAC và cách cấu hình
- Những use case phức tạp có thể dẫn đến cấu hình khó hiểu
- Khó khăn trong debugging khi policy không hoạt động như mong đợi
- Không dễ scale cho các hệ thống lớn so với các giải pháp chuyên biệt như OPA hoặc SpiceDB

2.14. Tổng quan về Meilisearch

2.14.1. Giới thiệu

Meilisearch là một search engine mã nguồn mở được viết bằng Rust [31], thiết kế để cung cấp trải nghiệm tìm kiếm nhanh, liên quan, và dễ sử dụng. Meilisearch được phát triển với mục tiêu là một giải pháp tìm kiếm dễ triển khai hơn Elasticsearch, phù hợp cho các ứng dụng từ nhỏ đến lớn.

Hình 13: Meilisearch Logo

2.14.2. Ưu điểm

Meilisearch mang lại nhiều lợi ích cho phát triển search:

- Easy to Deploy, single binary executable, dễ triển khai trên bất kỳ hạ tầng nào
- Fast Search, hiệu suất cao với latency thấp nhờ Rust implementation
- Rich Features, hỗ trợ full-text search, hybrid search, facets, filters, sorting, ranking
- Typo Tolerance, tự động xử lý typo và fuzzy search
- Multi-Language Support, hỗ trợ indexing và search cho nhiều ngôn ngữ
- Open Source, mã nguồn mở, có thể tùy chỉnh theo nhu cầu
- Low Resource Usage, sử dụng ít CPU và RAM so với Elasticsearch

2.14.3. Nhược điểm

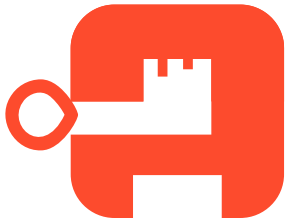
Bên cạnh các ưu điểm, Meilisearch có một số hạn chế:

- Community Size, cộng đồng nhỏ hơn Elasticsearch, ít plugin có sẵn
- Advanced Features, thiếu một số advanced features của Elasticsearch, không hỗ kiểu dữ liệu nested phức tạp

2.15. Tổng quan về Authentik

2.15.1. Giới thiệu

Authentik là một nền tảng identity provider mã nguồn mở được thiết kế để cung cấp các dịch vụ xác thực và cấp quyền tập trung [32]. Authentik hỗ trợ các chuẩn hiện đại như OIDC (OpenID Connect) và OAuth 2.0, cho phép kết nối dễ dàng với nhiều ứng dụng và dịch vụ.



Hình 14: Authentik Logo

2.15.2. Ưu điểm

Authentik có nhiều ưu điểm nổi bật:

- OIDC và OAuth 2.0 Support, hỗ trợ đầy đủ các chuẩn authorization và authentication hiện đại
- Self-Hosted, có thể triển khai trên hạ tầng riêng, tránh phụ thuộc vào dịch vụ external
- Flexible User Management, quản lý người dùng, nhóm, và phân quyền linh hoạt
- Provider Support, hỗ trợ tích hợp với nhiều external providers (Google, GitHub, SAML, v.v.)
- Phù hợp cho developement nhờ vào tính năng Authentik Blueprint, dễ dàng cấu hình và áp dụng tự động, thuận tiện hơn Keycloak rất nhiều khi phải sử dụng Terraform hay Ansible

2.15.3. Nhược điểm

Bên cạnh các ưu điểm, Authentik có một số hạn chế:

- Deployment Complexity, yêu cầu cấu hình và triển khai phức tạp
- Learning Curve, cần thời gian để hiểu các khái niệm OIDC, OAuth, và cấu hình policies
- Performance, Authentik được viết bàng Python là chính, nhưng các thành phần trọng yếu vẫn được viết bằng Go và Rust. Dù vậy, mức độ sử dụng RAM của Authentik ngang với Keycloak khi sử dụng trong quá trình phát triển
- Áp dụng Authentik Blueprint đôi khi lỗi nhưng không có log

2.16. Tổng quan về RustFS

2.16.1. Giới thiệu

RustFS [33] là một hệ thống lưu trữ object storage được viết hoàn toàn bằng Rust, tương thích với API của Amazon S3. RustFS được phát triển như một giải pháp thay thế hiệu suất cao hơn cho MinIO *(đã ngừng phát triển)*, đặc biệt là trong các trường hợp cần throughput lớn và latency thấp.

Rust được chọn vì hiệu suất và an toàn bộ nhớ, giúp RustFS cung cấp một cơ sở hạ tầng lưu trữ đáng tin cậy với resource overhead tối thiểu.

Hình 15: RustFS Logo

2.16.2. Ưu điểm

RustFS mang lại nhiều lợi ích cho phát triển object storage:

- S3 Compatibility, hỗ trợ đầy đủ API S3, cho phép dễ dàng thay thế MinIO hoặc S3
- High Performance, được viết bằng Rust, cung cấp hiệu suất vượt trội so với MinIO
- Concurrent Operations, hỗ trợ xử lý đồng thời tối ưu nhờ async/await của Rust
- Hỗ trợ
- Có thể giao tiếp thông qua MinIO CLI
- Hỗ trợ cloud native một cách chính thức, có thể triển khai trên Kubernetes

2.16.3. Nhược điểm

Bên cạnh các ưu điểm, RustFS có một số hạn chế:

- Young Ecosystem, tương đối mới so với MinIO, ecosystem có thể chưa hoàn chỉnh
- Tại thời điểm thực hiện dự án, RustFS vẫn chưa phát hành bất kỳ phiên bản ổn định

2.17. Tổng quan về Observability Stack

2.17.1. Giới thiệu

OpenTelemetry Protocol (OTLP) là một tiêu chuẩn mã nguồn mở cho việc thu thập và xuất dữ liệu observability (metrics, logs, traces) [24]. OTLP được hỗ trợ bởi hầu hết các công cụ monitoring hiện đại.

Dự án sử dụng một stack observability hoàn chỉnh bao gồm:

- Prometheus [34], cho metrics collection
- Grafana Loki [35], cho logs aggregation
- Grafana Tempo [36], cho distributed tracing
- Grafana Alloy [37], cho log/metrics forwarding agent
- Grafana [38], cho visualization tập trung

Hình 16: OpenTelemetry, Prometheus, Grafana, Loki, Tempo, Alloy Logo

Các service được instrumented bằng:

- `@opentelemetry/auto-instrumentations-node`: auto instrumentation cho Node.js
- `go.opentelemetry.io/contrib/exporters/autoexport` cho auto setup exporters cho Go

2.17.2. Ưu điểm

Stack observability mang lại nhiều lợi ích:

- Unified Observability, tập trung logs, metrics, và traces trong một platform
- Standards-Based, sử dụng OTLP, một tiêu chuẩn mở được nhiều tool hỗ trợ
- Auto Instrumentation, tự động capture telemetry data mà không cần code changes
- Cost-Effective, hầu hết components là mã nguồn mở và có thể tự triển khai
- Scalability, được thiết kế để xử lý large-scale deployments
- Multi-Language Support, hỗ trợ instrumentation cho nhiều ngôn ngữ lập trình
- Real-Time Insights, cung cấp real-time visibility vào application performance

2.17.3. Nhược điểm

Bên cạnh các ưu điểm, stack này có một số hạn chế:

- Operational Complexity, triển khai và maintain toàn bộ stack yêu cầu kiến thức sâu
- Data Volume, observability data có thể lớn, yêu cầu storage và retention planning
- Learning Curve, cần học cách cấu hình và sử dụng từng thành phần
- Network Overhead, thu thập observability data có thể thêm network traffic
- Cost of Storage, lưu trữ long-term observability data có thể tốn kém

2.18. Tổng quan về Redpanda

2.18.1. Giới thiệu

Redpanda là một nền tảng event streaming mã nguồn mở được viết bằng C++ với API tương thích Kafka [39]. Redpanda được thiết kế để cung cấp hiệu suất cao hơn Kafka trong khi duy trì tính tương thích hoàn toàn với Kafka protocol và ecosystem.

Hình 17: redpanda Logo

Redpanda được sử dụng như một message broker trong dự án, hỗ trợ pub/sub patterns cho event-driven architecture.

2.18.2. Ưu điểm

Redpanda mang lại nhiều lợi ích cho phát triển event-driven:

- Production Ready, đã được sử dụng trong production bởi nhiều công ty lớn
- Kafka Compatible, hoàn toàn tương thích với Kafka API, cho phép sử dụng Kafka clients, SDK và tools
- Single Binary, chỉ một binary executable, dễ triển khai hơn Kafka
- Resource Efficient, sử dụng ít CPU và RAM hơn Kafka
- Built-in Schema Registry, tích hợp schema registry không cần tool riêng
- Multi-Cloud, hỗ trợ triển khai trên nhiều cloud providers
- Plugin Support, hỗ trợ custom plugins bằng Go, Python thông qua Redpanda Connect SDK

2.18.3. Nhược điểm

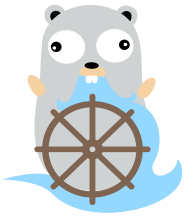
Bên cạnh các ưu điểm, Redpanda có một số hạn chế:

- Learning Curve, cần hiểu event-driven architecture và Kafka concepts
- Operational Knowledge, cần kiến thức vận hành hệ thống distributed messaging
- Community Size, cộng đồng nhỏ hơn Kafka, ít tài liệu nâng cao có sẵn
- `postgres_cdc` [40] của Redpanda Connect yêu cầu phiên bản trả phí, có thể sử dụng Debezium, hoặc sử dụng `sql_select` [41] của Redpanda Connect

2.19. Tổng quan về Watermill

2.19.1. Giới thiệu

Watermill [42] là library Go cho event-driven architecture, hỗ trợ multiple message routers. Watermill được thiết kế để tạo điều kiện cho asynchronous message processing, event streaming, request-response, và reactive architectures.



Hình 18: Watermill Logo

2.19.2. Watermill Architecture

Watermill cung cấp abstraction trên các message brokers khác nhau, cho phép viết event processing logic một lần và sử dụng với nhiều transport backends.

Watermill được sử dụng trong dự án:

- Kafka integration với Redpanda *(Phần 2.18)* cho distributed streaming
- Redis pub/sub [43] cho lightweight pub/sub messaging trong cùng cluster
- Go channels cho in-process communication
- Outbox pattern với Forwarder Component của Watermill

2.19.3. OpenTelemetry Integration

Watermill hỗ trợ OpenTelemetry *(Phần 2.17)* integration thông qua `nkonev/watermill-opentelemetry` [44], cho phép trace event processing workflow. Điều này cung cấp observability cho event-driven systems.

2.19.4. Ưu điểm

- Flexible Routing, hỗ trợ đa dạng transport backends
- Structured Event Handling, type-safe event processing

2.19.5. Nhược điểm

- Event-Driven Complexity, event-driven phức tạp hơn so với mô hình synchronous
- Learning Curve, event-driven patterns cần được hiểu rõ
- Debugging, debugging distributed events là thách thức lớn
- Operational Overhead, yêu cầu manage message broker infrastructure

2.20. Tổng quan về Nx

2.20.1. Giới thiệu

Nx [45] là build system và monorepo management tool được phát triển bởi Nrwl. Nx cung cấp các tính năng cho task orchestration, caching intelligent, visualization của dependency graph, code generation, và workspace analysis.

Hình 19: Nx Logo

Trong dự án, Nx được sử dụng như task runner và build system, thiết lập toàn bộ luồng code gen cho GRPC, OpenAPI, SQLC,… Trước đây, nhóm đã từng thiết lập Bazel nhưng đã chuyển sang Nx vì Bazel chỉ giải quyết được vấn đề build, không giúp việc dev trở nên dễ dàng hơn.

2.20.2. Ưu điểm

- Task Orchestration, quản lý và chạy tasks phụ thuộc, song song một cách tối ưu
- Intelligent Caching, caching của build artifacts dựa trên file changes
- Dependency Graph Visualization, hiểu dependencies giữa các packages
- Workspace Analysis, phân tích codebase để xác định affected projects
- Hỗ trợ thiết lập monorepo Typescript dễ dàng hơn, song, vẫn cần kiến thức vững vàng về typescript và hệ sinh thái typescript *(Phần 2.2)*

2.20.3. Nhược điểm

- Learning Curve, cần hiểu Nx concepts và configuration
- Configuration Complexity, setup phức tạp cho custom use cases, như các luồng code gen, thiết lập với Go, vì không hỗ trợ chính thức cho Go
- Overhead cho Small Projects, overhead không cần thiết cho projects nhỏ

CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

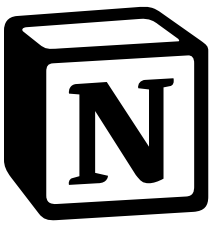
Chương 3 trình bày quá trình phân tích và thiết kế hệ thống nhằm xác định rõ cấu trúc và cách thức hoạt động của hệ thống. Nội dung chương bao gồm khảo sát hiện trạng, xác định yêu cầu chức năng và phi chức năng, từ đó xây dựng kiến trúc hệ thống và mô tả các thành phần chính. Bên cạnh đó, chương sử dụng các sơ đồ UML để mô hình hóa hành vi, luồng xử lý và mối quan hệ giữa các đối tượng trong hệ thống, thiết kế dữ liệu thông qua các bảng cơ sở dữ liệu. Hệ thống cũng phân tích BlockNote schema tuỳ chỉnh, và mô hình thư viện Casbin của hệ thống.

3.1. Khảo sát hiện trạng

3.1.1. Các hệ thống hiện có

Hiện nay có nhiều hệ thống quản lý kiến thức cá nhân khác nhau, mỗi hệ thống đều có những ưu điểm và nhược điểm riêng. Dưới đây là một số hệ thống phổ biến mà nhóm đã khảo sát.

Notion



Hình 20: Notion logo

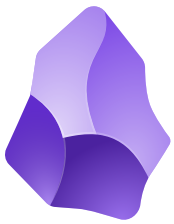
Notion là một nền tảng làm việc tích hợp được ra mắt vào năm 2013, kết hợp tài liệu, cơ sở dữ liệu, wiki và quản lý dự án trong một nền tảng duy nhất. Với hơn 30 triệu người dùng toàn cầu, Notion đã phát triển từ một ứng dụng ghi chú đơn giản thành một hệ thống kiến thức toàn diện.

Lịch sử phát triển của Notion bắt đầu với phiên bản 1.0 tập trung vào hợp tác tài liệu thời gian thực và tổ chức tài liệu phong cách wiki. Điểm đột phá lớn nhất đến với phiên bản 2.0 vào tháng 3 năm 2018 khi giới thiệu tính năng Database, nâng cấp Notion lên một tầm cao mới và cho phép tạo ra các công cụ ứng dụng. Hiện tại, Notion liên tục được cập nhật với các tính năng mới như AI agents, hệ thống tự động hóa hoàn chỉnh và nhiều loại giao diện cơ sở dữ liệu mới.

Các tính năng chính bao gồm editor module hóa với các khối có thể sắp xếp linh hoạt, cơ sở dữ liệu đa dạng *(bảng, Kanban, lịch trình, gallery)*, khả năng hợp tác thời gian thực, và tích hợp AI. Giao diện của Notion được đánh giá cao về tính thẩm mỹ và sự linh hoạt, cho phép người dùng tùy chỉnh không gian làm việc theo nhu cầu riêng.

Tuy nhiên, Notion cũng có những hạn chế nhất định, điển hình như không hỗ trợ graph view một cách trực quan.

Obsidian



Hình 21: Obsidian logo

Obsidian là một ứng dụng ghi chú và quản lý kiến thức dựa trên tệp Markdown, được phát triển bởi Shida Li và Erica Xu và ra mắt vào năm 2020. Với triết lý "local-first", Obsidian đảm bảo dữ liệu của người dùng luôn nằm trên thiết bị của họ, mang lại quyền sở hữu và tính linh hoạt tối đa.

Obsidian hoạt động với một "vault" chứa các tài liệu văn bản, mỗi ghi chú mới tạo ra một tệp Markdown riêng. Điểm mạnh nổi bật nhất của Obsidian là khả năng liên kết hai chiều và chế độ xem graph giúp người dùng hình dung mối quan hệ giữa các ghi chú. Người dùng có thể tạo liên kết nội dung bằng Wikilinks hoặc liên kết Markdown thông thường, và tất cả các liên kết này đều được hiển thị trong graph view tương tác.

Các tính năng chính bao gồm chế độ xem graph giúp phân tích mối quan hệ giữa các ghi chú, Canvas cho phép sắp xếp ghi chú trực quan không giới hạn, hệ thống plugin mở rộng do cộng đồng phát triển, và khả năng tích hợp với nhiều công cụ khác. Obsidian cũng hỗ trợ chế độ chỉnh sửa song song giữa Source Mode và Live Preview, cùng với đầy đủ các công cụ tìm kiếm và tổ chức. Obsidian còn chính hỗ trợ vim motion cho những người dùng yêu thích trải nghiệm chỉnh sửa văn bản nâng cao.

Mặc dù mạnh mẽ, Obsidian cũng có những thách thức. Learning curve khá cao, đặc biệt với người mới bắt đầu. Thiếu tính năng hợp tác thời gian thực là một hạn chế quan trọng. Giao diện mặc định có thể trông đơn giản so với các đối thủ cạnh tranh. Obsidian phù hợp nhất với các nhà nghiên cứu, nhà văn, lập trình viên và người làm việc kiến thức muốn kiểm soát sâu sắc ghi chú của mình.

jackyzha0/quartz

Hình 22: Quartz logo

Quartz là một static-site generator nhanh, được thiết kế để biến nội dung Markdown thành các website hoàn toàn chức năng. Được phát triển bởi jackyzha0, Quartz hoạt động như một giải pháp self-hosted thay thế cho Obsidian Publish, cho phép người dùng xuất bản ghi chú và digital gardens của mình lên web một cách dễ dàng.

Quartz tương thích hoàn toàn với Obsidian, hỗ trợ đầy đủ các tính năng như wikilinks, backlinks, transclusions, graph view và full-text search. Nó được xây dựng

trên công nghệ TypeScript và cung cấp khả năng tùy chỉnh cao thông qua JSX layouts và page components. Một trong những điểm mạnh của Quartz là tốc độ tải trang cực nhanh và kích thước bundle nhỏ.

Các tính năng chính bao gồm hỗ trợ LaTeX, Mermaid diagrams, dark mode, breadcrumbs, popover previews, internationalization, và hệ thống comments. Quartz cũng hỗ trợ Docker, RSS feeds, và có hệ thống plugin mở rộng. Nó đặc biệt phù hợp cho các cá nhân muốn chia sẻ kiến thức của mình dưới dạng website công khai hoặc digital garden.

Mặc dù mạnh mẽ, Quartz cũng có những hạn chế. Nó yêu cầu kiến thức kỹ thuật về Git và hosting để thiết lập. Không có tính năng hợp tác thời gian thực như Notion. Giao diện quản lý nội dung không trực quan bằng các ứng dụng desktop. Quartz phù hợp nhất với các nhà phát triển, sinh viên, và giáo viên muốn xuất bản ghi chú của mình dưới dạng website.

3.1.2. So sánh các hệ thống hiện có

Hệ thống	Ưu điểm	Nhược điểm
Notion	Giao diện đẹp, đa tính năng, hợp tác thời gian thực	Phức tạp khi quản lý dự án lớn, tốc độ tải chậm với dữ liệu lớn
Obsidian	Local-first, graph view mạnh mẽ, liên kết hai chiều, plugin đa dạng	Thiếu hợp tác thời gian thực, learning curve cao
Quartz	Tốc độ nhanh, tương thích Obsidian, dễ xuất bản website, tùy chỉnh cao	Yêu cầu kiến thức kỹ thuật, không có hợp tác thời gian thực, giao diện đơn giản

Bảng 1: So sánh các hệ thống hiện có

Xuất phát từ đó, nhóm đã xác định được những điểm mạnh và hạn chế của từng hệ thống, từ đó rút ra những bài học quan trọng để áp dụng vào thiết kế hệ thống của mình. Nhưng cũng cần nhấn mạnh rằng Notopia không nhằm mục đích thay thế hoàn toàn các hệ thống hiện có, mà chỉ thực hiện một số chức năng nhất định, mang tính chất nghiên cứu, học tập, áp dụng công nghệ và quy trình phát triển phần mềm.

3.2. Quy trình phát triển

Giai đoạn	Thời gian	Hoạt động	Kết quả
1	26/01 - 04/02	Khởi động & Nghiên cứu	Xác định yêu cầu, nghiên cứu công nghệ
2	05/02 - 12/02	Thiết lập	Môi trường làm việc, CI Pipeline sẵn sàng
3	13/02 - 25/02	Phân tích & Thiết kế	Đặc tả Use Case, thiết kế Database Schema, UI/UX
4	26/02 - 05/04	Phát triển & Tích hợp (Đợt 1)	Hệ thống đăng nhập và quản lý ghi chú cơ bản hoạt động
5	06/04 - 10/05	Phát triển & Tích hợp (Đợt 2)	Ứng dụng đầy đủ tính năng
6	11/05 - 17/05	Hoàn thiện Sản phẩm	Sản phẩm hoàn thiện về tính năng và thẩm mỹ
7	18/05 - 31/05	Báo cáo & Tổng kết	Báo cáo cuối kỳ và source code hoàn chỉnh

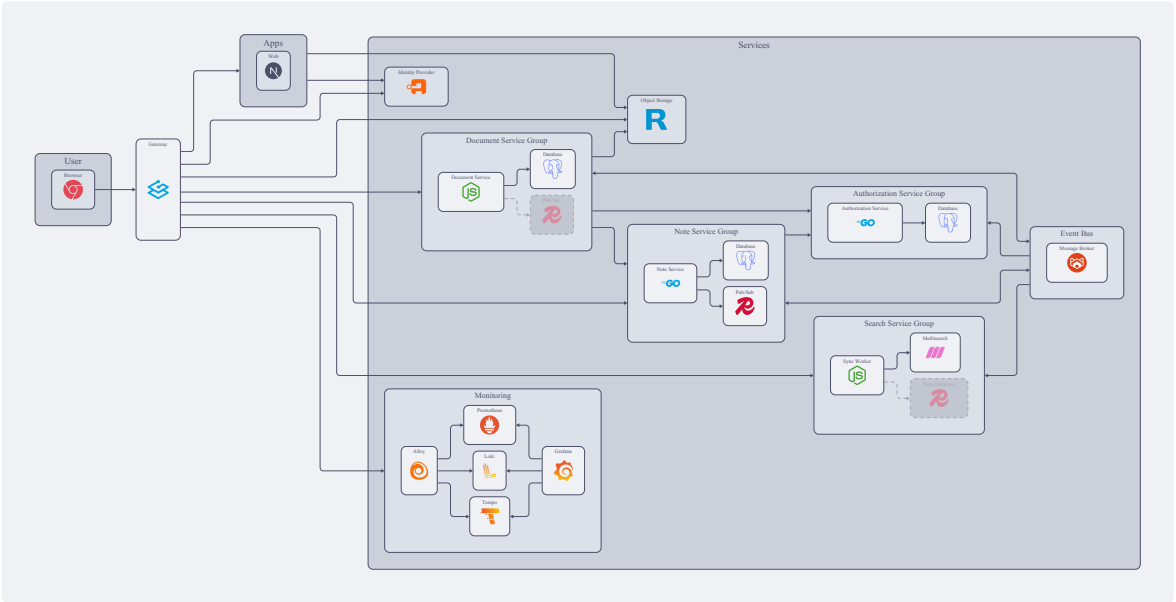
Bảng 2: Lịch trình các giai đoạn phát triển dự án

3.3. Kiến trúc hệ thống

Dự án được chia thành nhiều service khác nhau, bao gồm `note` service, `document` service, `authorization` service, `search-worker` worker, và các thành phần tái sử dụng từ các dịch vụ sẵn có. Mỗi service có trách nhiệm riêng biệt và giao tiếp với nhau thông qua API và message queue để đảm bảo tính modular và dễ bảo trì.

3.3.1. Sơ đồ kiến trúc tổng quan

Dưới đây là sơ đồ kiến trúc tổng quan của hệ thống.



Hình 23: Sơ đồ kiến trúc tổng quan

Trong đó, các thành phần chính bao gồm:

Thành phần	Mô tả
User	Người dùng tương tác với hệ thống thông qua giao diện web
Gateway	Điểm vào chính của hệ thống, chịu trách nhiệm định tuyến yêu cầu đến các service phù hợp
Web App	Ứng dụng web cung cấp giao diện người dùng để tạo và quản lý ghi chú
Note Service	Quản lý metadata và logic liên quan đến không gian làm việc và ghi chú
Document Service	Quản lý nội dung của các ghi chú
Authorization Service	Xác định quyền truy cập của người dùng đối với các tài nguyên
Identity Provider	Dịch vụ xác thực và quản lý người dùng
Object Storage	Lưu trữ các tệp liên quan đến ghi chú, như hình ảnh và tài liệu
Search Service	Cung cấp khả năng tìm kiếm nội dung trong ghi chú
Event Bus	Hệ thống message queue để giao tiếp giữa các service
Monitoring	Giám sát hiệu suất và trạng thái của hệ thống

Bảng 3: Các thành phần trong kiến trúc

Chi tiết về các thành phần chính:

Gateway

Gateway là điểm vào chính của hệ thống, chịu trách nhiệm định tuyến các yêu cầu từ người dùng đến các service phù hợp. Nó cũng thực hiện các chức năng như xác thực, và cân bằng tải để đảm bảo hiệu suất và bảo mật của hệ thống.

Hệ thống sử dụng Traefik *(Phần 2.12)* làm API Gateway, với đặc điểm gọn nhẹ và dễ cấu hình, tương thích tốt với docker compose. Sử dụng `traefik-plugins/traefik-jwt-plugin` [29] để xử lý xác thực JWT, chuyển hoá OIDC claim thành header HTTP, giúp các service phía sau có thể dễ dàng xác định người dùng, không cần phải tích hợp trực tiếp với Identity Provider.

Web App

Web App là giao diện người dùng chính, cho phép người dùng tạo, chỉnh sửa và quản lý ghi chú. Ứng dụng được xây dựng với React và NextJS *(Phần 2.3)*.

Note Service

Note Service *(`note` service)*, viết bằng Go, chịu trách nhiệm quản lý metadata và logic liên quan đến không gian làm việc và ghi chú. Dịch vụ này cung cấp API để Web App có thể tương tác với không gian làm việc và ghi chú, đồng thời tích hợp với Authorization Service để kiểm tra quyền truy cập của người dùng trước khi thực hiện các hành động liên quan đến thư mục, ghi chú.

Document Service

Document Service *(`document` service)*, viết bằng Typescript và NestJS framework, chịu trách nhiệm quản lý nội dung của các ghi chú, bao gồm lưu trữ và truy xuất dữ liệu. Nó cung cấp API để Web App có thể tương tác với nội dung ghi chú, tích hợp với Object Storage để lưu trữ các tệp liên quan, và sử dụng Hocuspocus *(Phần 2.8)* để hỗ trợ cộng tác thời gian thực trên nội dung ghi chú.

Authorization Service

Authorization Service *(`authorization` service)*, viết bằng Go, sử dụng thư viên Casbin *(Phần 2.13)* để quản lý quyền truy cập của người dùng đối với không gian làm việc và ghi chú. Dịch vụ này cung cấp API để các service khác có thể kiểm tra quyền truy cập của người dùng trước khi thực hiện các hành động liên quan đến thư mục, ghi chú.

Identity Provider

Identity Provider chịu trách nhiệm xác thực người dùng và quản lý thông tin tài khoản. Hệ thống sử dụng Authentik *(Phần 2.15)* là giải pháp xác thực, cung cấp các tính năng như đăng nhập một lần *(SSO)*, quản lý người dùng, và hỗ trợ nhiều phương thức xác thực. Hệ thống sử dụng OpenID Connect *(OIDC)* để tích hợp giữa Gateway, Web App với Identity Provider, giúp đơn giản hóa quá trình xác thực và quản lý người dùng.

Object Storage

Object Storage được sử dụng để lưu trữ các tệp liên quan đến ghi chú, như hình ảnh và tài liệu. Hệ thống sử dụng RustFS *(Phần 2.16)*, một giải pháp lưu trữ đối tượng nhẹ và hiệu quả, cung cấp API tương thích với S3 để dễ dàng tích hợp với các service khác.

Search Service

Search Service *(Meilisearch)*, viết bằng Rust, cung cấp khả năng tìm kiếm nội dung trong ghi chú *(Phần 2.14)*. Trong đó, `search-worker` worker chịu trách nhiệm đồng bộ dữ liệu đến Search Service để đảm bảo dữ liệu tìm kiếm luôn được cập nhật.

Event Bus

Event Bus là hệ thống message queue được sử dụng để giao tiếp giữa các service, đảm bảo tính modular và giảm sự phụ thuộc trực tiếp giữa các service. Hệ thống sử dụng Redpanda *(Phần 2.18)*, một giải pháp message queue hiệu suất cao, tương thích với Kafka API, giúp dễ dàng tích hợp với các service khác.

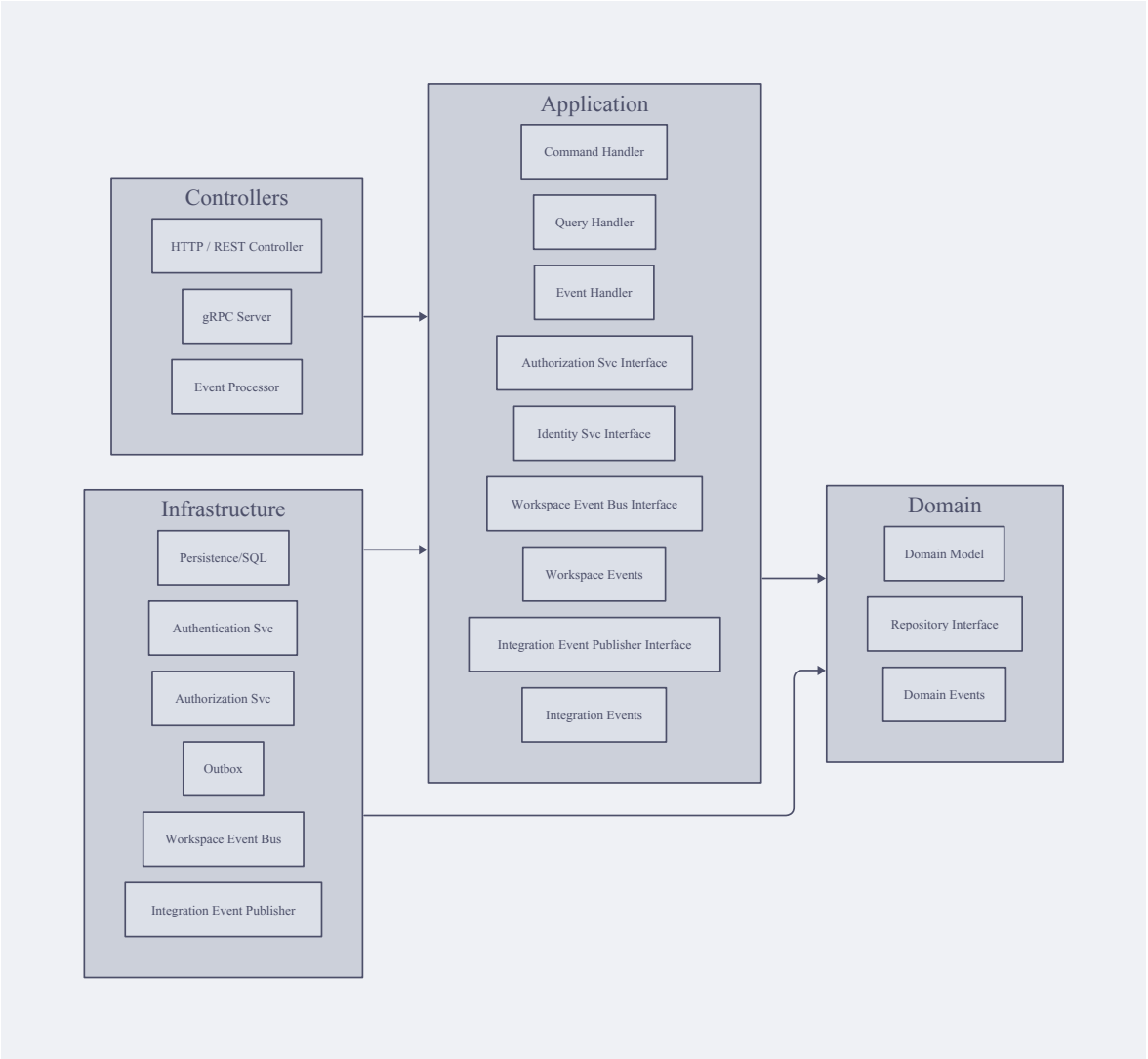
Monitoring

Monitoring là thành phần quan trọng để giám sát hiệu suất và trạng thái của hệ thống. Hệ thống sử dụng Grafana Stack và Prometheus *(Phần 2.17)* để thu thập và hiển thị các chỉ số về hiệu suất.

3.3.2. Kiến trúc `note` service

Là thành phần trung tâm trong hệ thống, `note` service chịu trách nhiệm quản lý metadata và logic liên quan đến ghi chú. Dịch vụ này được thiết kế theo kiến trúc Clean Architecture, Domain Driven Design, và Event-Driven Architecture để đảm bảo tính modular, dễ bảo trì, và khả năng mở rộng trong tương lai[5]. Dưới đây là sơ đồ kiến trúc chi tiết của `note` service.

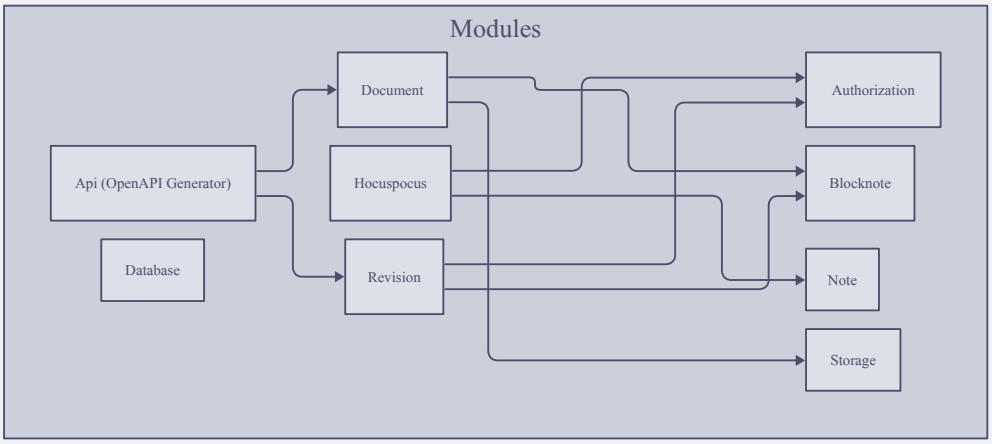
[5]Tham khảo cách tổ chức code từ [46], [47]



Hình 24: Kiến trúc của `note` service

3.3.3. Kiến trúc `document` service

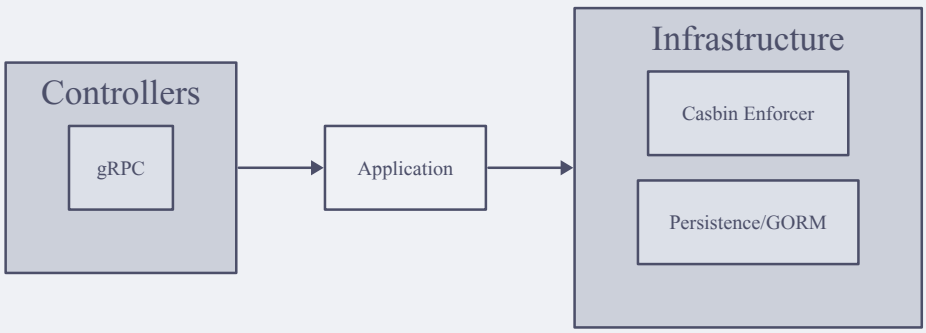
`document` service áp dụng kiến trúc theo NestJS, với các module được tổ chức theo chức năng. Dưới đây là sơ đồ kiến trúc chi tiết của `document` service, mô tả đến cấp độ module.



Hình 25: Kiến trúc của `document` service

3.3.4. Kiến trúc `authorization` service

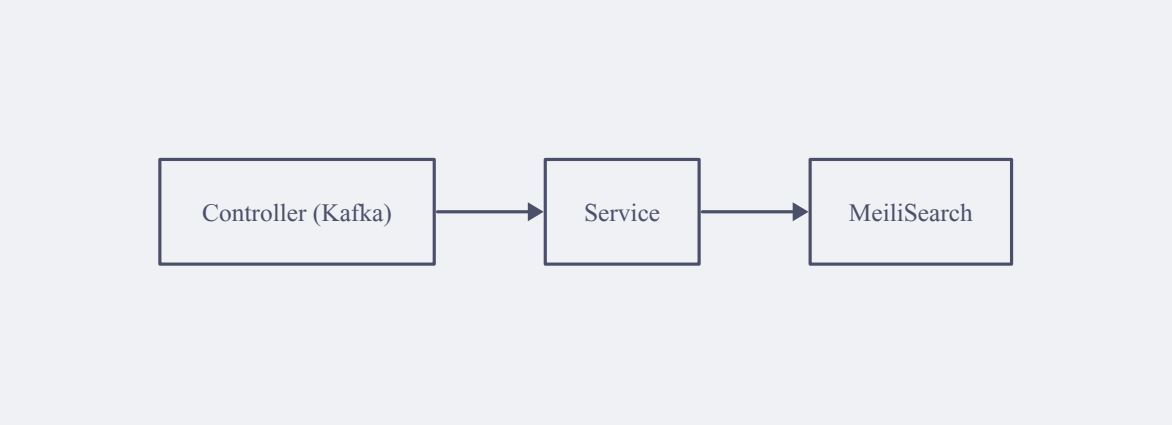
`authorization` service được thiết kế theo kiến trúc Layered Architecture, với tầng logic không phân tách rõ vì tính đặc thù của thư viện Casbin. Dưới đây là sơ đồ kiến trúc của `authorization` service.



Hình 26: Kiến trúc của `authorization` service

3.3.5. Kiến trúc `search-worker` worker

`search-worker` worker được thiết kế theo kiến trúc đơn giản, sử dụng trên 1 module chính của NestJS, không chia thành nhiều module nhỏ. Dưới đây là sơ đồ kiến trúc của `search-worker` worker.

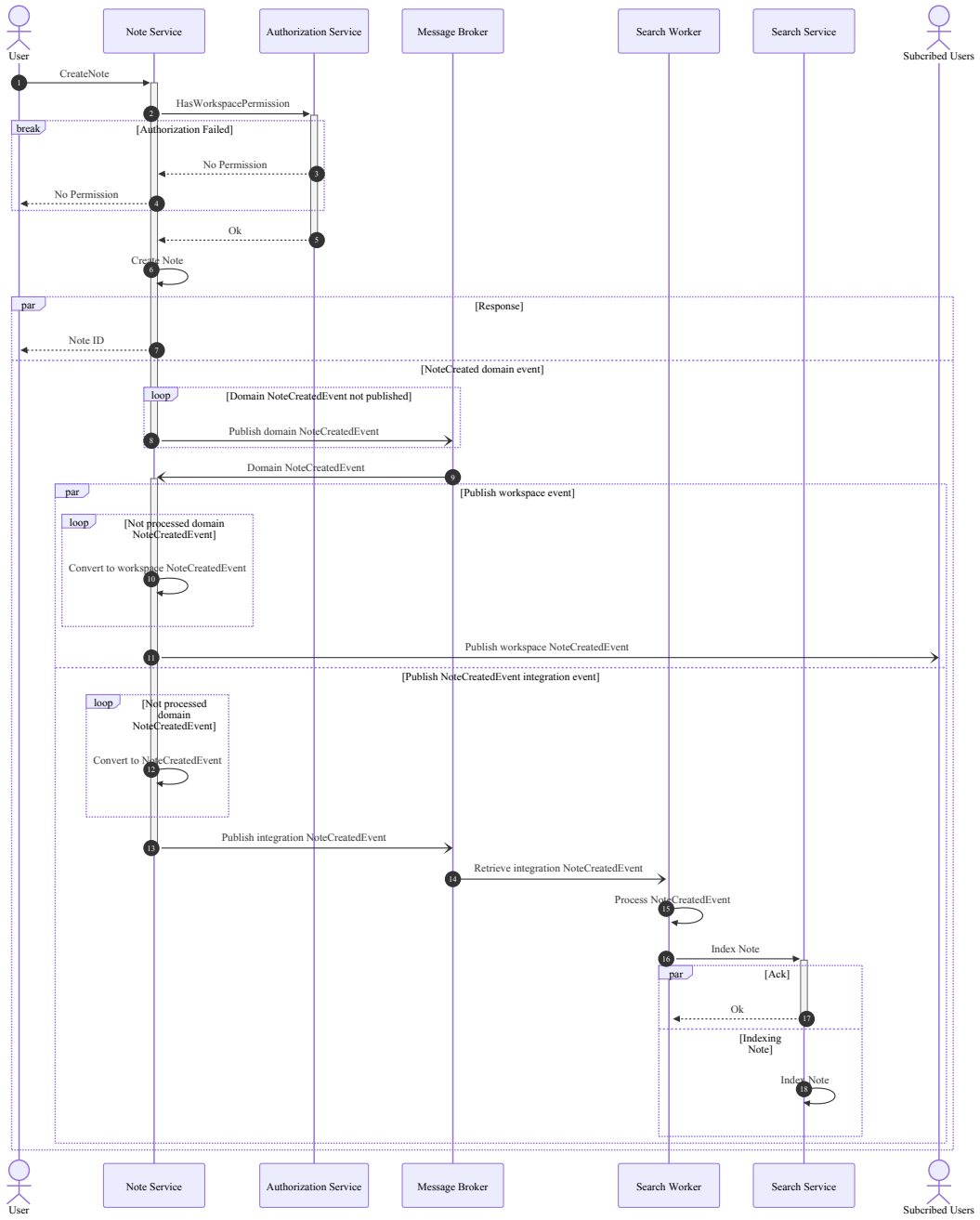


Hình 27: Kiến trúc của `search-worker` worker

3.4. Mô tả các Use Case

Nhóm chỉ xác định một vài use case phức tạp, có nhiều luồng, tác động đến nhiều dịch vụ.

3.4.1. Mô tả use case Create Note



Hình 28: Sequence diagram mô tả create note use case

Trường	Nội dung
ID	UC01
Name	Create Note
Description	Use case này mô tả quy trình tạo ghi chú mới
Actor(s)	User
Priority	Cao
Trigger	Người dùng muốn tạo một ghi chú mới
Pre-condition(s)	• Người dùng sử dụng thiết bị có kết nối internet

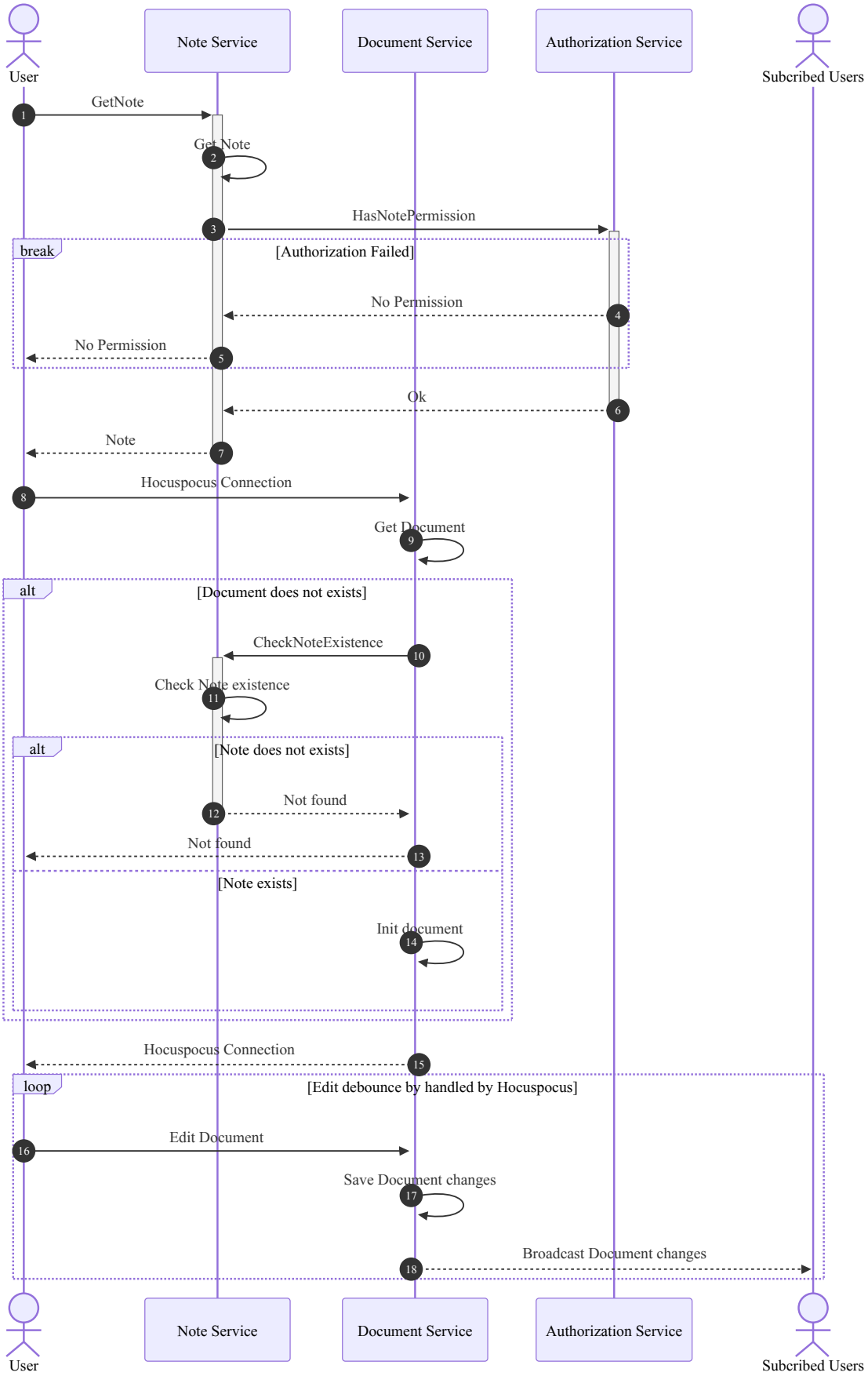
Trường	Nội dung
	• Người dùng đã đăng nhập vào hệ thống • Người dùng có quyền tạo ghi chú trong không gian làm việc hiện tại
Post-condition(s)	• Người dùng tạo ghi chú thành công
Basic Flow	1. Người dùng truy cập vào hệ thống 2. Người dùng chọn nút "Tạo ghi chú mới" được hiển thị 3. Người dùng nhập vào các trường thông tin của ghi chú *(tiêu đề, v.v...)* 4. Hệ thống kiểm tra quyền của người dùng để tạo ghi chú 5. Hệ thống tạo ghi chú mới và trả về ID của ghi chú 6. Hệ thống publish domain event `NoteCreatedEvent` tới Message Broker 7. Hệ thống chuyển đổi domain event thành workspace event và publish 8. Hệ thống publish integration event `NoteCreatedEvent` tới Message Broker 9. `search-worker` nhận integration event và xử lý 10. `search-worker` gửi yêu cầu Index Note đến Search service 11. Search service nhận yêu cầu Index Note và tiến hành indexing ghi chú mới

Alternate Flow	1. Bước 6: Hệ thống gặp lỗi khi gửi domain event `NoteCreatedEvent`
	    1. Hệ thống tự động retry publish domain event
	2. Bước 7: Hệ thống gặp lỗi khi chuyển đổi hoặc publish workspace event
	    1. Hệ thống ghi log lỗi và bỏ qua việc publish workspace event
	3. Bước 8: Hệ thống gặp lỗi khi publish integration event
	    1. Hệ thống tự động retry publish integration event
	4. Bước 9: `search-worker` gặp lỗi khi xử lý `NoteCreatedEvent`
	    1. `search-worker` ghi log lỗi và bỏ qua event
	5. Bước 10: Search service gặp lỗi khi indexing ghi chú mới
	    1. Search service ghi log lỗi và bỏ qua yêu cầu indexing

Trường	Nội dung
Exception Flow	1. Bước 4: Người dùng không có quyền tạo ghi chú trong workspace hiện tại 1. Hệ thống trả về lỗi `forbidden` 2. Use case dừng lại 2. Bước 5: Hệ thống gặp lỗi khi tạo ghi chú mới 1. Hệ thống trả về lỗi 2. Use case dừng lại

Bảng 4: Mô tả use case Create Note

3.4.2. Mô tả use case Get Note



Hình 29: Sequence diagram mô tả get note use case

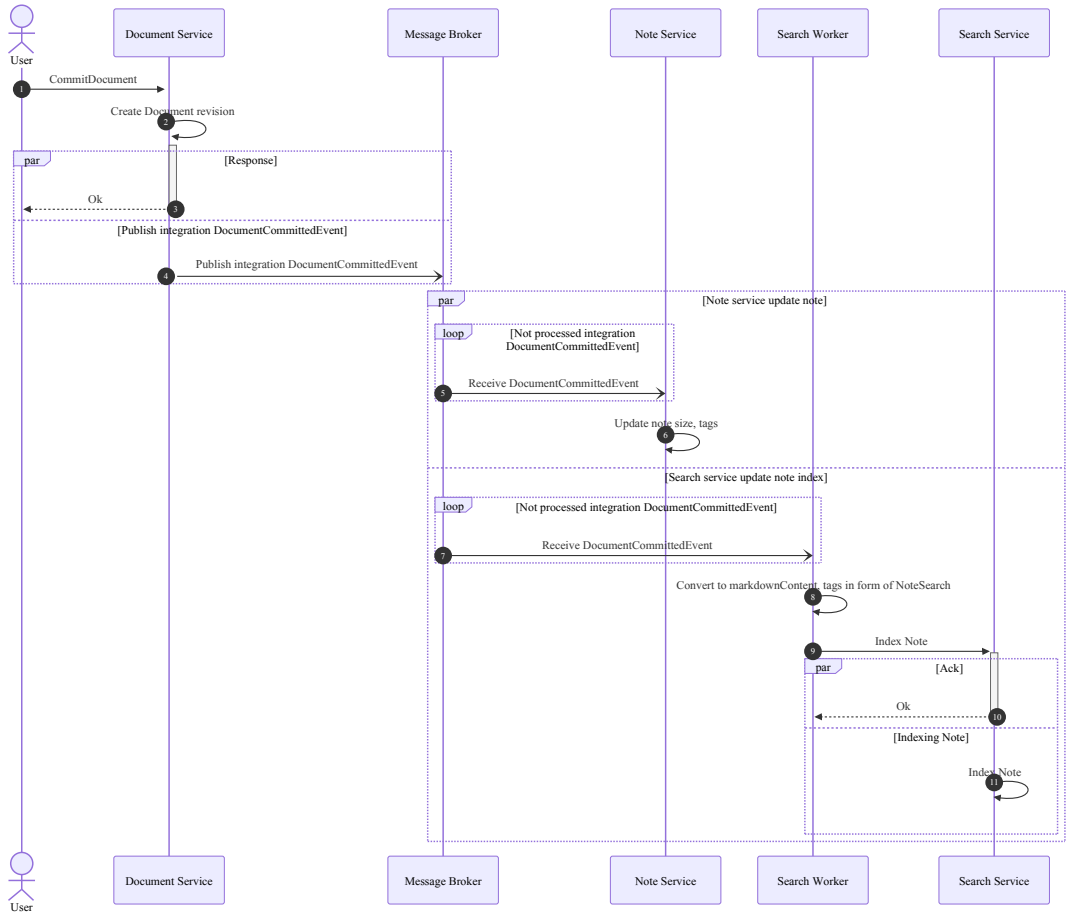
Trường	Nội dung
ID	UC02
Name	Get Note
Description	Use case này mô tả quy trình lấy nội dung ghi chú và chuyển sang chế độ chỉnh sửa
Actor(s)	User
Priority	Cao
Trigger	Người dùng chọn một ghi chú để xem hoặc chỉnh sửa
Pre-condition(s)	• Người dùng sử dụng thiết bị có kết nối internet • Người dùng đã đăng nhập vào hệ thống • Người dùng có quyền xem ghi chú trong không gian làm việc hiện tại
Post-condition(s)	• Người dùng nhận được thông tin ghi chú • Kết nối Hocuspocus được thiết lập để xem và *(tùy chọn)* chỉnh sửa ghi chú
Basic Flow	1. Người dùng chọn ghi chú cần xem 2. Hệ thống lấy thông tin ghi chú từ `note` service 3. Hệ thống kiểm tra quyền xem ghi chú với `authorization` service 4. Hệ thống trả về thông tin ghi chú cho người dùng 5. Hệ thống thiết lập kết nối Hocuspocus để xem nội dung ghi chú 6. Nếu người dùng chọn chỉnh sửa, Hệ thống sẽ giữ kết nối và cho phép gửi các thao tác chỉnh sửa 7. `document` service kiểm tra bộ nhớ nội bộ trước 8. Nếu tài liệu chưa tồn tại trong bộ nhớ nội bộ, `document` service kiểm tra với `note` service 9. Nếu ghi chú tồn tại, `document` service khởi tạo tài liệu và lưu vào bộ nhớ nội bộ 10. `document` service trả về kết nối Hocuspocus cho người dùng 11. Nếu có thao tác chỉnh sửa, `document` service lưu thay đổi và phát sóng cho các client khác

Alternate Flow	1. Bước 8-9: Tài liệu đã tồn tại trong bộ nhớ nội bộ của `document` service
	  1. `document` service bỏ qua bước kiểm tra với `note` service
	  2. `document` service trả về kết nối Hocuspocus trực tiếp
	2. Bước 12: Người dùng không chọn chỉnh sửa *(chỉ xem)*

Trường	Nội dung
	1. Kết nối Hocuspocus duy trì ở chế độ xem 2. Không có thao tác chỉnh sửa nào được thực hiện 3. Bước 13: Lỗi khi lưu thay đổi tài liệu trong quá trình debounce 1. `document` service ghi log lỗi và thử lại sau thời gian debounce
Exception Flow	1. Bước 4: Người dùng không có quyền xem ghi chú 1. Hệ thống trả về lỗi `forbidden` 2. Use case dừng lại 2. Bước 9: Ghi chú không tồn tại trong `note` service 1. Hệ thống trả về lỗi `noteNotFound` 2. Use case dừng lại

Bảng 5: Mô tả use case Get Note

3.4.3. Mô tả use case Commit Document



Hình 30: Sequence diagram mô tả commit document use case

Trường	Nội dung
ID	UC03

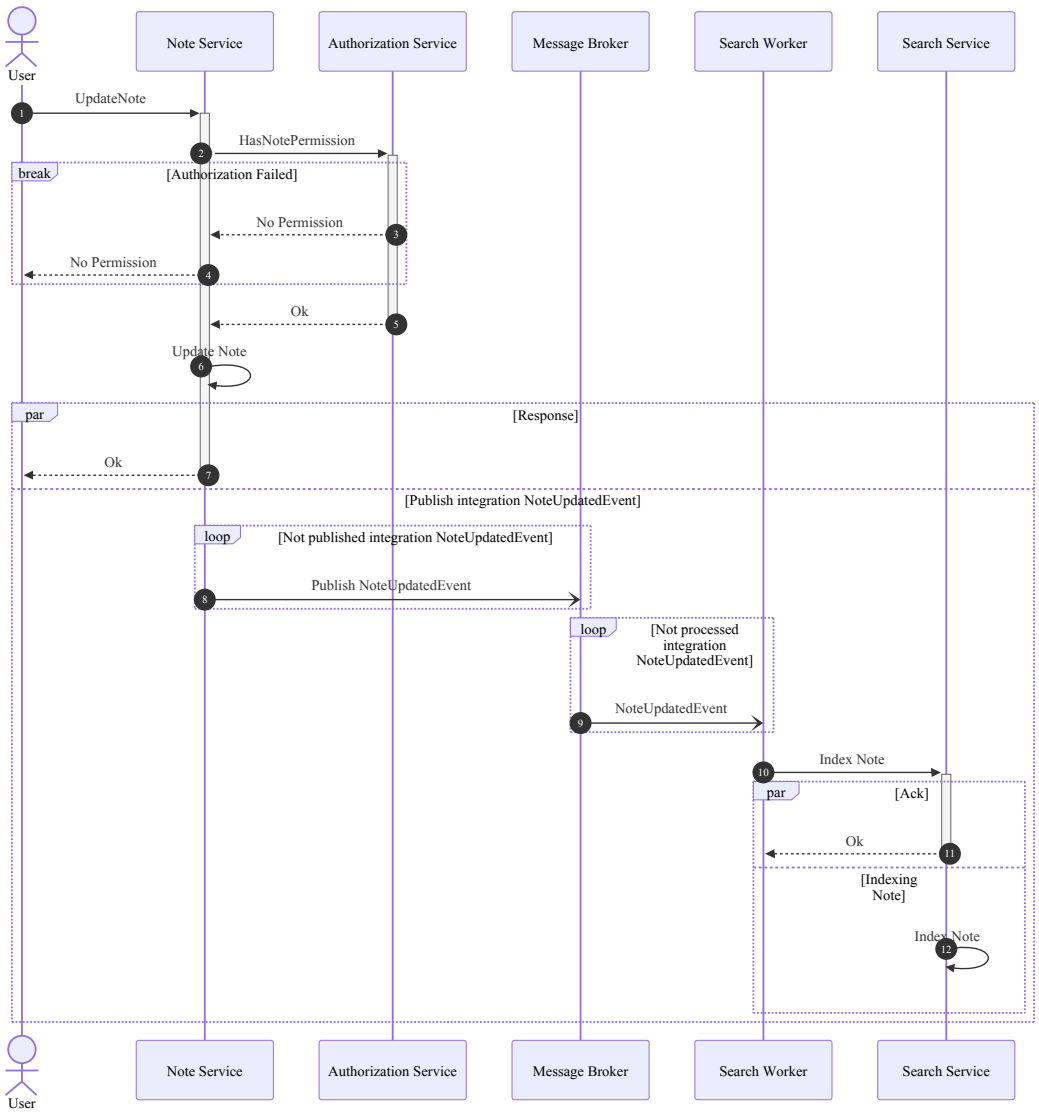
Trường	Nội dung
Name	Commit Document
Description	Use case này mô tả quy trình lưu và publish event cập nhật tài liệu *(nội dung của ghi chú)*
Actor(s)	User
Priority	Cao
Trigger	Người dùng chọn lưu tài liệu để cập nhật
Pre-condition(s)	• Người dùng đã đăng nhập vào hệ thống • Người dùng có quyền chỉnh sửa tài liệu trong không gian làm việc hiện tại • Tài liệu đã tồn tại trong hệ thống
Post-condition(s)	• Tài liệu được tạo ra phiên bản lưu trữ *(revision)* mới • Integration Event `DocumentCommittedEvent` được publish • Việc cập nhật được phản ánh vào hệ thống nhận xét và tìm kiếm
Basic Flow	1. Người dùng chọn hành động "Commit Document" trên giao diện 2. `document` service tạo phiên bản lưu trữ mới của tài liệu 3. `document` service publish integration event `DocumentCommittedEvent` vào Message Broker 4. Message Broker phân phối event cho `note` service và `search-worker` 5. `note` service nhận event và cập nhật các thông tin như size, tags 6. `search-worker` nhận event và chuyển đổi thành markdownContent, tags 7. Search service nhận yêu cầu index lại ghi chú và thực hiện indexing 8. Các thay đổi được đăng báo lại cho các client thông qua Hocuspocus

Alternate Flow	1. Bước 4: Lỗi khi tạo revision mới của tài liệu
	  1. Hệ thống ghi log và thông báo lỗi cho người dùng
	2. Bước 6: Dữ liệu BlockNote không đúng với schema
	  1. Hệ thống discard event và ghi log lỗi

Exception Flow	1. Bước 2: Lỗi khi lấy quyền edit 1. Trả về lỗi `forbidden` và dừng quy trình

Bảng 6: Mô tả use case Commit Document

3.4.4. Mô tả use case Update Note



Hình 31: Sequence diagram mô tả update note use case

Trường	Nội dung
ID	UC04
Name	Update Note
Description	Use case này mô tả quy trình cập nhật thông tin cơ bản của ghi chú
Actor(s)	User
Priority	Cao
Trigger	Người dùng chỉnh sửa và lưu thay đổi thông tin ghi chú
Pre-condition(s)	• Người dùng đã đăng nhập vào hệ thống • Người dùng có quyền chỉnh sửa ghi chú trong không gian làm việc hiện tại • Ghi chú tồn tại trong hệ thống

Trường	Nội dung
Post-condition(s)	• Thông tin ghi chú được cập nhật thành công • `NoteUpdatedEvent` được publish tới các dịch vụ khác
Basic Flow	1. Người dùng chỉnh sửa thông tin ghi chú và chọn lưu 2. Hệ thống kiểm tra quyền chỉnh sửa ghi chú với `authorization` service 3. Hệ thống cập nhật thông tin ghi chú trong `note` service 4. Hệ thống trả về kết quả thành công cho người dùng 5. Hệ thống publish `NoteUpdatedEvent` tới Message Broker (có retry nếu chưa published) 6. `search-worker` nhận `NoteUpdatedEvent` và xử lý 7. Search service nhận yêu cầu Index Note và thực hiện indexing
Alternate Flow	1. Bước 3: Hệ thống gặp lỗi kiểm tra quyền 1. Ghi log lỗi và trả về thông báo cho người dùng 2. Bước 5: Chưa published được `NoteUpdatedEvent` 1. Hệ thống tự động retry cho đến khi thành công 3. Bước 7: Lỗi khi xử lý `NoteUpdatedEvent` ở `search-worker` 1. Ghi log và bỏ qua event này 4. Bước 8: Lỗi khi indexing ở Search service 1. Ghi log và thử lại sau debounce
Exception Flow	1. Bước 2: Người dùng không có quyền chỉnh sửa ghi chú 1. Hệ thống trả về lỗi `forbidden` 2. Use case dừng lại 2. Bước 4: Lỗi khi cập nhật thông tin ghi chú 1. Hệ thống trả về lỗi 2. Use case dừng lại

Bảng 7: Mô tả use case Update Note

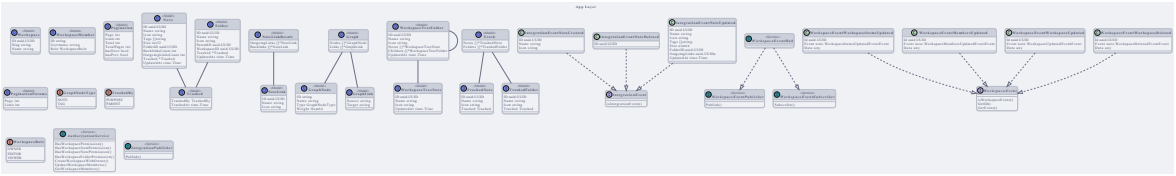
3.5. Sơ đồ lớp (Class Diagram)

Dưới đây là sơ đồ lớp cho các service trong hệ thống, bao gồm `note` service và `document` service, không bao gồm `authorization` service vì tính đặc thù của service không thể biểu diễn một cách dễ dàng, và `search-worker` vì xử lý dữ liệu không phức tạp. Sơ đồ lớp giúp minh họa cấu trúc của các lớp, các thuộc tính và phương thức của chúng, cũng như mối quan hệ giữa các lớp.

3.5.1. Sơ đồ lớp cho note service

Sơ đồ lớp cho `note` service được chia thành hai phần: tầng application và tầng domain. Tầng application tập trung vào các lớp mô hình dữ liệu và interface của service, trong khi tầng domain tập trung vào các lớp đại diện cho các aggregate và logic nghiệp vụ.

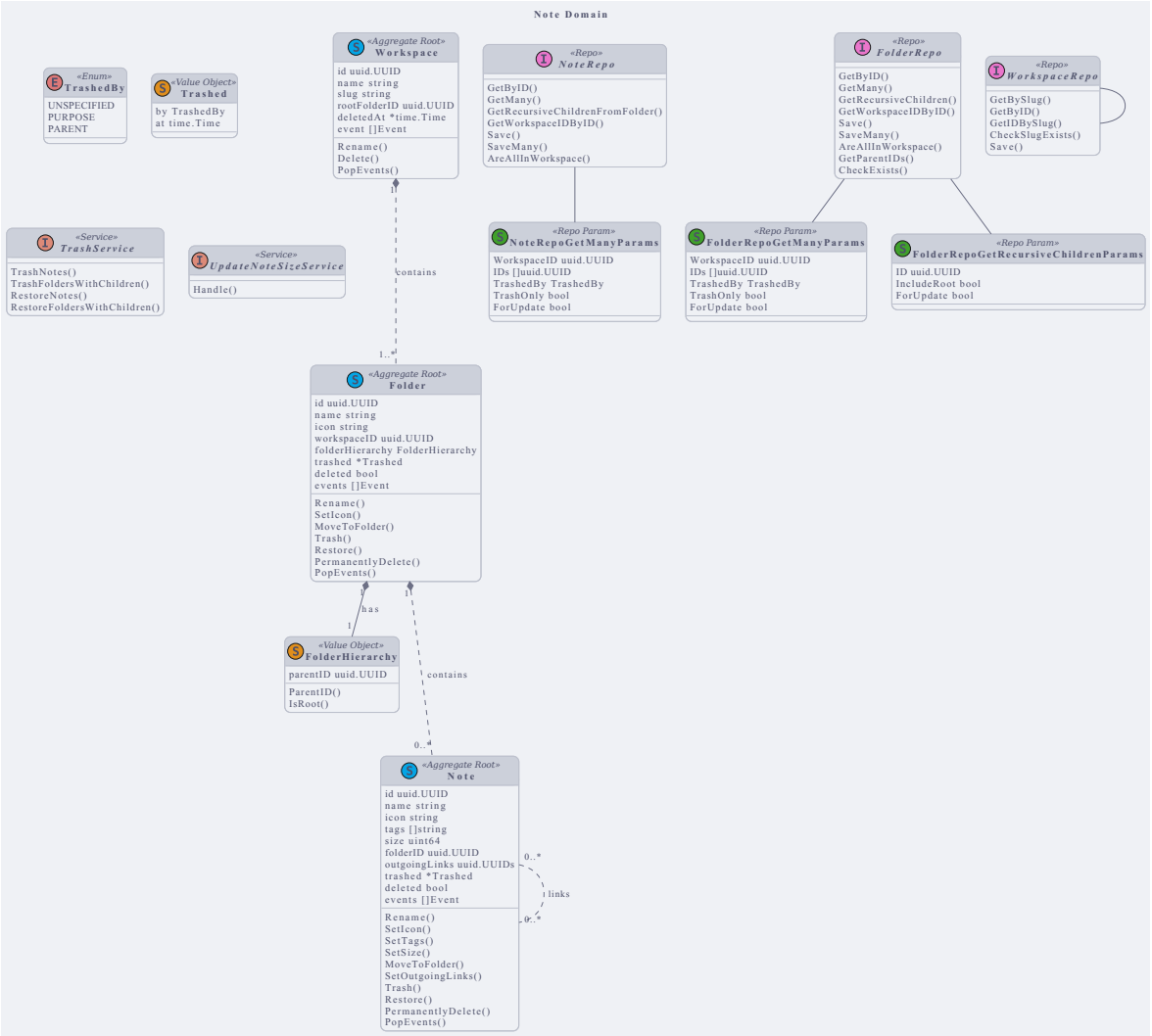
Sơ đồ lớp tầng application



Hình 32: Sơ đồ lớp tầng application cho `note` service

Tầng application bao gồm các lớp mô hình dữ liệu *(model)* cho việc read *(theo CQRS)* và các interface của service. Các lớp mô hình dữ liệu đại diện cho các thực thể trong hệ thống như `Note`, `Folder`, `Workspace`, v.v. Các interface của service định nghĩa các phương thức mà tầng infrastructure sẽ triển khai.

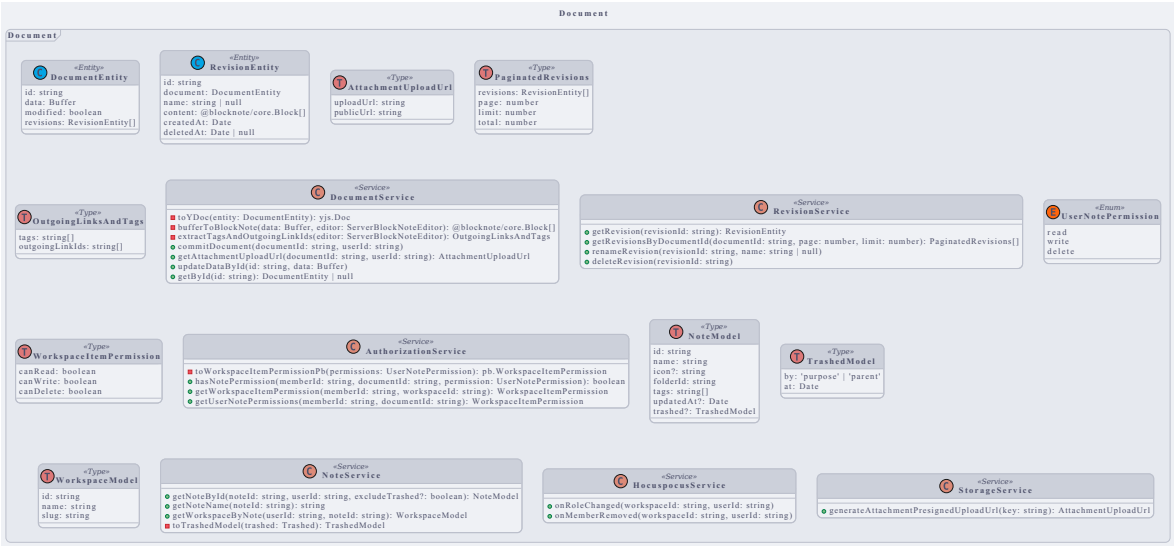
Sơ đồ lớp tầng domain



Hình 33: Sơ đồ lớp tầng domain cho `note` service

Tầng domain, áp dụng Domain Driven Design Pattern, bao gồm các lớp đại diện cho các aggregate như `Note`, `Folder`, `Workspace`, v.v. Các lớp này chứa các thuộc tính và phương thức liên quan đến logic nghiệp vụ của chúng. Ngoài ra, tầng domain cũng bao gồm các interface của repository để truy cập dữ liệu. Tuy nhiên, tầng domain không được trực tiếp gọi service được định nghĩa hay repository interface được khai báo.

3.5.2. Sơ đồ lớp cho document service



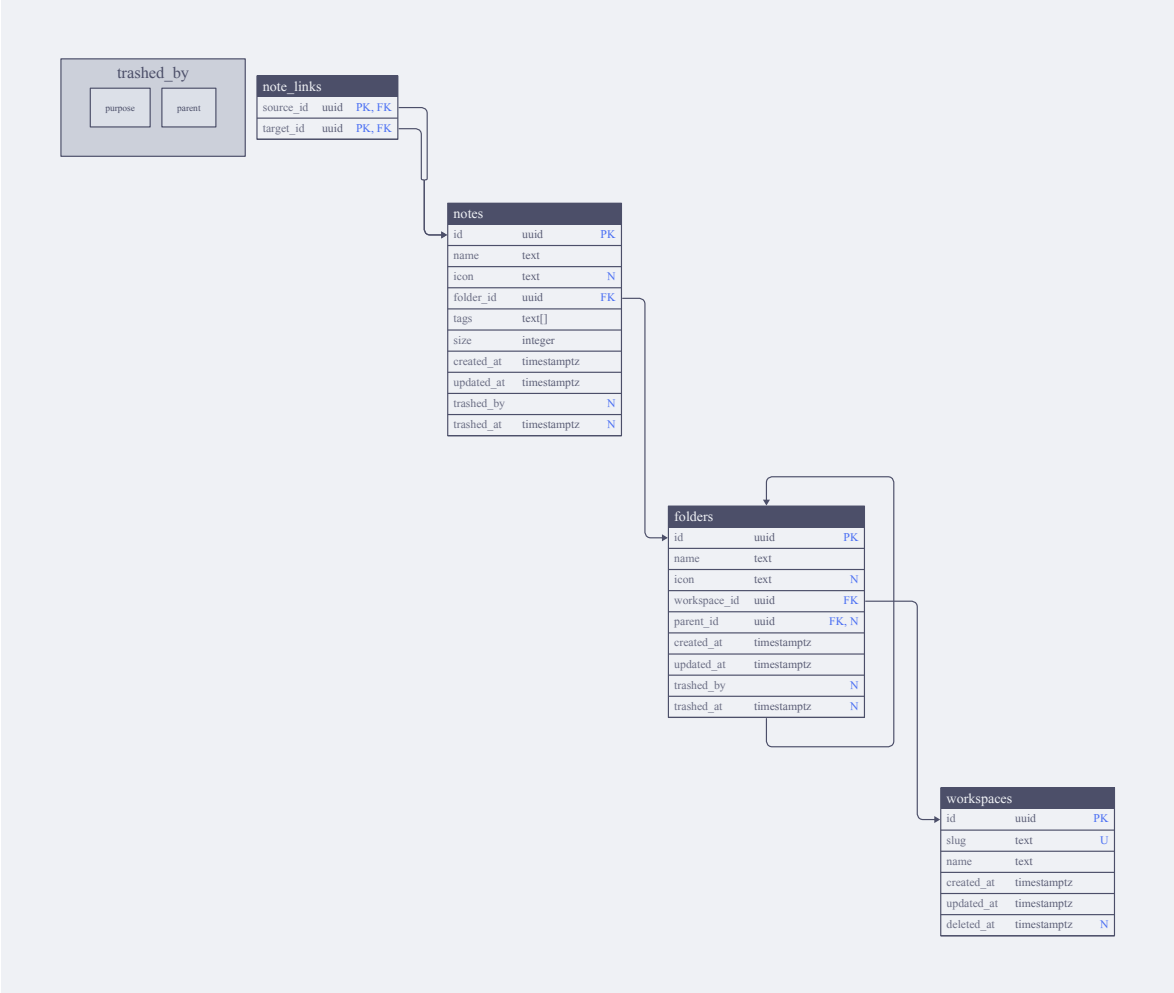
Hình 34: Sơ đồ lớp cho `document` service

Sơ đồ tinh gọn cho `document` service, bao gồm các thực thể, services, và đối tượng liên quan. Ở thiết kế này không bao gồm repository vì đã được TypeORM abstract đi, tức, service sẽ trực tiếp gọi các phương thức của TypeORM để truy cập dữ liệu.

3.6. Thiết kế cơ sở dữ liệu

Các service trong hệ thống sẽ sử dụng cơ sở dữ liệu PostgreSQL để lưu trữ và quản lý dữ liệu. Dưới đây là thiết kế cơ sở dữ liệu cho các service, bao gồm các bảng chính và mối quan hệ giữa chúng.

3.6.1. Cơ sở dữ liệu cho `note` service



Hình 35: Sơ đồ cơ sở dữ liệu cho `note` service

Bảng workspaces

Tên cột	Kiểu dữ liệu	Mô tả	Khóa
`id`	`UUID`	Mã định danh duy nhất của workspace	`PK`
`slug`	`TEXT`	URL-friendly slug của workspace	`UQ`
`name`	`TEXT`	Tên của workspace	
`created_at`	`TIMESTAMPTZ`	Thời gian tạo	
`updated_at`	`TIMESTAMPTZ`	Thời gian cập nhật gần nhất	
`deleted_at`	`TIMESTAMPTZ`	`Thời gian xóa (soft delete, Nullable)`	

Bảng 8: Bảng workspaces - `note` service

Bảng folders

Tên cột	Kiểu dữ liệu	Mô tả	Khóa
id	UUID	Mã định danh duy nhất của folder	PK
name	TEXT	Tên của folder (Nullable)	
icon	TEXT	Icon của folder (Nullable)	
workspace_id	UUID	ID workspace chứa folder này	FK
parent_id	UUID	ID folder cha *(nested structure) (Nullable)*	FK
created_at	TIMESTAMPTZ	Thời gian tạo	
updated_at	TIMESTAMPTZ	Thời gian cập nhật gần nhất	
trashed_by	ENUM	Loại xóa *(purpose|parent, Nullable)*	
trashed_at	TIMESTAMPTZ	Thời gian xóa *(Nullable)*	

Bảng 9: Bảng folders - note service

Bảng notes

Tên cột	Kiểu dữ liệu	Mô tả	Khóa
id	UUID	Mã định danh duy nhất của ghi chú	PK
name	TEXT	Tên của ghi chú	
icon	TEXT	Icon của ghi chú (Nullable)	
folder_id	UUID	ID folder chứa ghi chú này	FK
tags	TEXT[]	Danh sách tag của ghi chú *(Nullable)*	
size	INTEGER	Kích thước của ghi chú (bytes)	
created_at	TIMESTAMPTZ	Thời gian tạo	
updated_at	TIMESTAMPTZ	Thời gian cập nhật gần nhất	
trashed_by	ENUM	Loại xóa *(purpose|parent, Nullable)*	
trashed_at	TIMESTAMPTZ	Thời gian xóa *(Nullable)*	

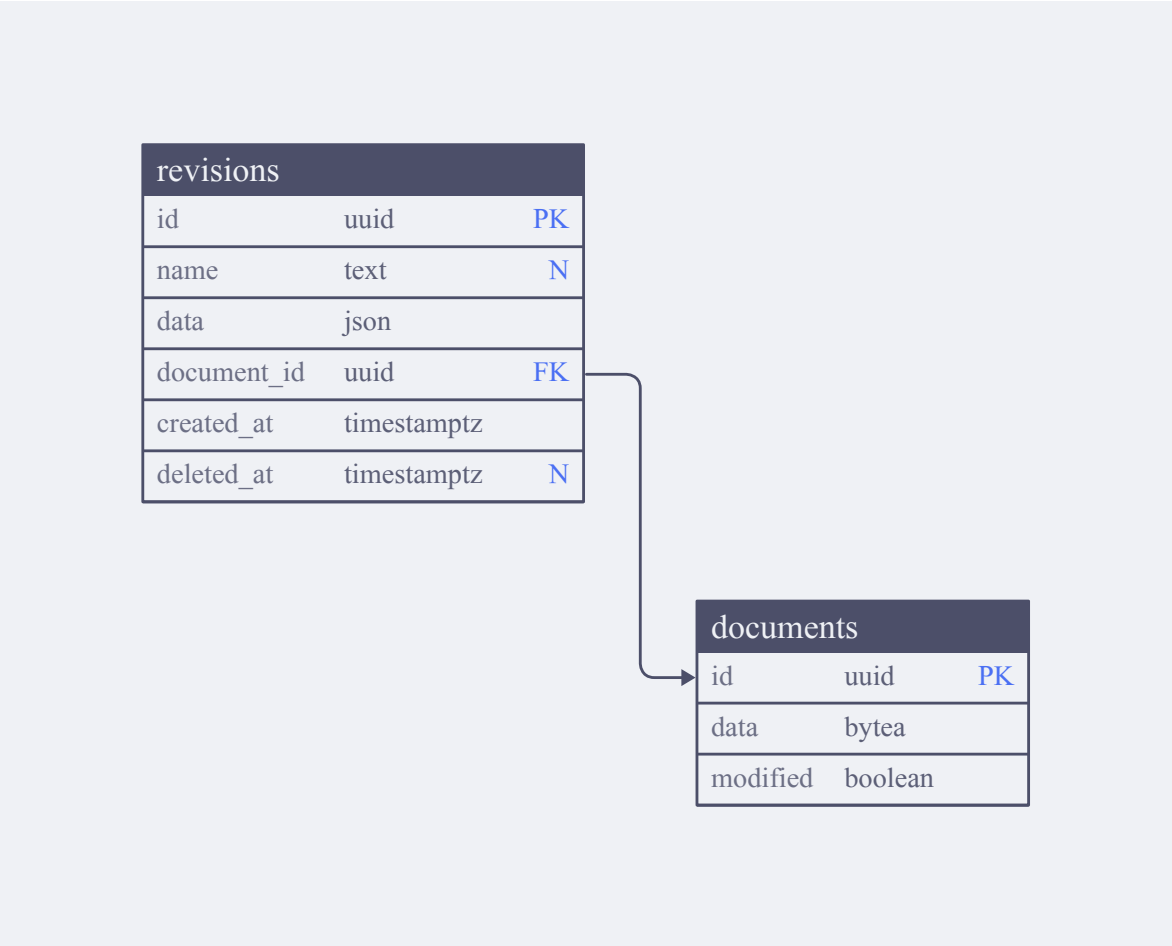
Bảng 10: Bảng notes - note service

Bảng note_links

Tên cột	Kiểu dữ liệu	Mô tả	Khóa
source_id	UUID	ID ghi chú nguồn	PK, FK
target_id	UUID	ID ghi chú đích	PK, FK

Bảng 11: Bảng note_links - note service

3.6.2. Cơ sở dữ liệu cho document service



Hình 36: Sơ đồ cơ sở dữ liệu cho document service

Bảng documents

Tên cột	Kiểu dữ liệu	Mô tả	Khóa
id	UUID	Mã định danh duy nhất của document	PK
data	BYTEA	Dữ liệu nhị phân yjs[6]	
modified	BOOLEAN	Trạng thái đã được chỉnh sửa hay chưa	

Bảng 12: Bảng documents - document service

Bảng Revisions

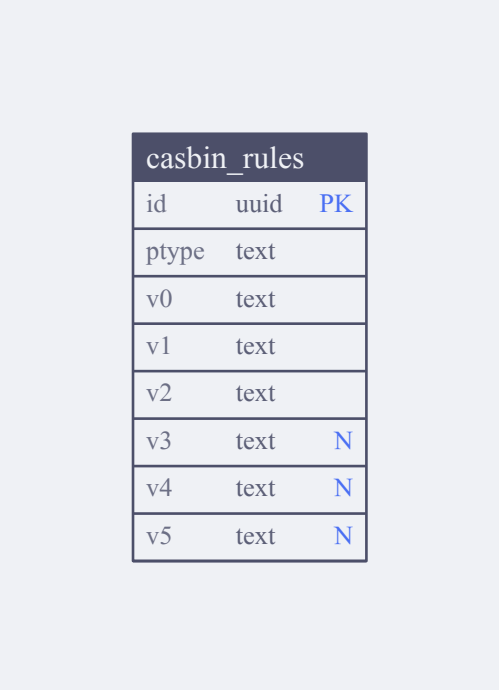
[6]Dành cho việc lưu trữ và truy xuất bởi Hocuspocus

Tên cột	Kiểu dữ liệu	Mô tả	Khóa
id	UUID	Mã định danh duy nhất của revision	PK
name	TEXT	Tên của revision *(Nullable)*	
data	JSON	Dữ liệu BlockNote	
document_id	UUID	ID document liên kết với revision	FK
created_at	TIMESTAMPTZ	Thời gian tạo	
deleted_at	TIMESTAMPTZ	Thời gian xóa *(Nullable)*	

Bảng 13: Bảng revisions - document service

3.6.3. Cơ sở dữ liệu cho authorization service

Vì tính đặc thù của thư viện Casbin, bảng dữ liệu của service này không thuộc phạm vi quản lý của hệ thống, mà sẽ được Casbin tự động tạo ra và quản lý.



Hình 37: Sơ đồ cơ sở dữ liệu cho authorization service

Bảng casbin_rules

Tên cột	Kiểu dữ liệu	Mô tả	Khóa
id	UUID	Mã định danh duy nhất của rule	PK
ptype	TEXT	Loại rule *(p, g, g2, v.v...)*	
v0	TEXT	Trường dữ liệu 0	
v1	TEXT	Trường dữ liệu 1	
v2	TEXT	Trường dữ liệu 2	
v3	TEXT	Trường dữ liệu 3 *(Nullable)*	
v4	TEXT	Trường dữ liệu 4 *(Nullable)*	
v5	TEXT	Trường dữ liệu 5 *(Nullable)*	

Bảng 14: Bảng casbin_rules - `authorization` service

3.7. Mô hình BlockNote tuỳ chỉnh trong hệ thống

BlockNote bao gồm nhiều schema nhiều schema, có thể xem tại Phần 2.9. Tuy nhiên, để có thể liên kết các ghi chú với nhau, cũng như hỗ trợ liên kết thông qua tag, hệ thống cần tuỳ chỉnh thêm hai config schema cho BlockNote, là `reference` và `tag`. Tham khảo từ tài liệu schema tuỳ chỉnh của BlockNote [48], hai config này sẽ được định nghĩa như sau.

3.7.1. Mô hình BlockNote Reference tuỳ chỉnh

```
import { CustomInlineContentConfig } from '@blocknote/core';

export const BlockNoteReferenceConfig = {
  type: 'reference',
  propSchema: {
    noteId: { default: 'unknown' },
  },
  content: 'none',
} as const satisfies CustomInlineContentConfig;
```

Chương trình 2: Cấu hình schema BlockNote tuỳ chỉnh cho reference

Một khối reference sẽ chứa một thuộc tính `noteId` để xác định ghi chú mà nó đang tham chiếu đến. Khi người dùng chèn một khối reference vào ghi chú, họ sẽ có thể chọn một ghi chú khác trong hệ thống để liên kết đến. Điều này cho phép tạo ra các mối quan hệ giữa các ghi chú.

```
{
  "type": "reference",
  "props": {
    "noteId": "156390b6-4b24-54c5-ae7f-77e462d7f106"
  }
}
```

Chương trình 3: Minh hoạ một khối reference trong BlockNote

3.7.2. Mô hình BlockNote Tag tuỳ chỉnh

```
import { CustomInlineContentConfig } from '@blocknote/core';

export const BlockNoteTagConfig = {
  type: 'tag',
  propSchema: {
    tag: { default: '' },
  },
  content: 'none',
} as const satisfies CustomInlineContentConfig;
```

Khối tag sẽ chứa một thuộc tính `tag` để lưu trữ tên của tag. Người dùng có thể chèn một khối tag vào ghi chú và gán cho nó một tên tag cụ thể.

```
{
  "type": "tag",
  "props": {
    "tag": "important"
  }
}
```

Chương trình 4: Minh hoạ một khối tag trong BlockNote

3.8. Mô hình Casbin trong hệ thống

Casbin hỗ trợ biểu diễn nhiều mô hình kiểm soát truy cập khác nhau, chi tiết xem tại Phần 2.13. Trong đó, hệ thống tập trung sử dụng mô hình RBAC để quản lý quyền truy cập dựa trên vai trò *(role)* của người dùng, trong mỗi không gian làm việc. Đối với Casbin, workspace được xem như một domain.

3.8.1. Mô hình Casbin trong hệ thống

```
[request_definition]
r = sub, dom, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _, _
g2 = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub, r.dom) \
  && g2(r.obj, p.obj) \
  && r.act == p.act
```

Chương trình 5: Model Casbin trong hệ thống

Trong đó, các phần được định nghĩa như sau:

- `request_definition` định nghĩa cấu trúc của một yêu cầu truy cập, bao gồm người dùng *(sub)*, không gian làm việc *(dom)*, đối tượng *(obj)* và hành động *(act)*
- `policy_definition` định nghĩa cấu trúc của một chính sách, bao gồm vai trò *(sub)*, đối tượng *(obj)* và hành động *(act)*
- `role_definition` định nghĩa cách thức xác định vai trò của người dùng trong một không gian làm việc *(g)* và cách thức xác định mối quan hệ giữa đối tượng và chính sách *(g2)*
- `policy_effect` định nghĩa hiệu ứng của chính sách, trong trường hợp này là cho phép truy cập nếu có ít nhất một chính sách cho phép
- `matchers` định nghĩa cách thức so khớp giữa yêu cầu truy cập và chính sách, trong đó yêu cầu truy cập sẽ được phép nếu người dùng có vai trò phù hợp trong không gian làm việc, đối tượng phù hợp với chính sách và hành động phù hợp với chính sách

3.8.2. Chính sách Casbin trong hệ thống

```
# owner
p, owner, workspace, read
p, owner, workspace, edit
p, owner, workspace, delete
p, owner, workspace_item, read
p, owner, workspace_item, write
p, owner, workspace_item, delete
# editor
p, editor, workspace, read
p, editor, workspace_item, read
p, editor, workspace_item, write
p, editor, workspace_item, delete
# viewer
p, viewer, workspace, read
p, viewer, workspace_item, read
# object hierarchy
g2, note, workspace_item
g2, folder, workspace_item
```

Chương trình 6: Policy Casbin trong hệ thống

Trong đó, các chính sách được định nghĩa như sau:

- Vai trò `owner` có quyền đọc, chỉnh sửa và xóa trên không gian làm việc và các mục trong không gian làm việc
- Vai trò `editor` có quyền đọc trên không gian làm việc và quyền đọc, chỉnh sửa và xóa trên các mục trong không gian làm việc
- Vai trò `viewer` chỉ có quyền đọc trên không gian làm việc và các mục trong không gian làm việc
- Các đối tượng `note` và `folder` được xác định là các mục trong không gian làm việc thông qua mối quan hệ g2

Mẫu minh hoạ yêu cầu truy cập và so khớp chính sách

Giả sử ta có ba người dùng: `110`, `111` và `112`, với các vai trò và quyền truy cập như sau:

```
# workspace 111
g, user:111, owner, workspace:00000000-0000-0000-0000-000000000111
g, user:112, editor, workspace:00000000-0000-0000-0000-000000000111
g, user:110, viewer, workspace:00000000-0000-0000-0000-000000000111
# workspace 112
g, user:112, owner, workspace:00000000-0000-0000-0000-000000000112
g, user:111, editor, workspace:00000000-0000-0000-0000-000000000112
# workspace 110
g, user:110, owner, workspace:00000000-0000-0000-0000-000000000110
```

Chương trình 7: Mẫu minh hoạ yêu cầu truy cập và so khớp chính sách Casbin

Trong đó:

- Người dùng `111` là chủ sở hữu *(owner)* của không gian làm việc `111`
- Người dùng `112` là biên tập viên *(editor)* của không gian làm việc `111` và chủ sở hữu của không gian làm việc `112`
- Người dùng `110` là người xem *(viewer)* của không gian làm việc `111` và chủ sở hữu của không gian làm việc `110`

Loại	Code

Request	`user:111, workspace:111, workspace, read`
	`user:111, workspace:111, note, write`
	`user:112, workspace:111, workspace, delete`
	`user:112, workspace:111, note, write`
	`user:112, workspace:111, folder, write`
	`user:110, workspace:111, workspace, read`

Result	`true Reason: ["owner","workspace","read"]`
	`true Reason: ["owner","workspace_item","write"]`
	`false`
	`true Reason: ["editor","workspace_item","write"]`
	`true Reason: ["editor","workspace_item","write"]`
	`true Reason: ["viewer","workspace","read"]`

Bảng 15: Mẫu minh hoạ yêu cầu truy cập và kết quả so khớp chính sách

Giải thích cho yêu cầu truy cập đầu tiên

Người dùng `111` yêu cầu đọc không gian làm việc `111`. Kết quả là true vì người dùng `111` có vai trò `owner` trong không gian làm việc `111` và có chính sách cho phép đọc không gian làm việc. Các bước suy luận dựa trên matcher `m`:

1. Người dùng `111` có vai trò `owner` trong không gian làm việc `111` thông qua chính sách g. Ta có thể xác định rằng `g(user:111, owner, workspace:111)` là true.
2. Đối tượng `workspace` cũng chứa trong `workspace` *(*workspace $\subseteq$ workspace*)* thông qua chính sách g2. Ta có thể xác định rằng `g2(workspace, workspace)` là true.
3. Hành động `read` phù hợp với chính sách `read` của vai trò `owner` trên không gian làm việc. Ta có thể xác định rằng `read == read` là true.

Giải thích cho yêu cầu truy cập thứ hai

Người dùng `111` yêu cầu viết vào note trong không gian làm việc `111`. Kết quả là true vì người dùng `111` có vai trò `owner` trong không gian làm việc `111` và có chính sách cho phép viết vào các mục trong không làm việc. Các bước suy luận:

1. Người dùng `111` có vai trò `owner` trong không gian làm việc `111` thông qua chính sách g. Ta có thể xác định rằng `g(user:111, owner, workspace:111)` là true.
2. Đối tượng `note` chứa trong `workspace_item` *(*note ⊆ workspace_item*)* thông qua chính sách g2 Ta có thể xác định rằng `g2(note, workspace_item)` là true.
3. Hành động `write` phù hợp với chính sách `write` của vai trò `owner` trên các mục trong không làm việc. Ta có thể xác định rằng `write == write` là true.

Giải thích cho yêu cầu truy cập thứ ba

Người dùng `112` yêu cầu xóa không gian làm việc `111`. Kết quả là false vì người dùng `112` chỉ có vai trò `editor` trong không gian làm việc `111` và không có chính sách nào cho phép `editor` xóa không gian làm việc. Các bước suy luận:

1. Người dùng `112` có vai trò `editor` trong không gian làm việc `111` thông qua chính sách g. Ta có thể xác định rằng `g(user:112, editor, workspace:111)` là true.
2. Đối tượng `workspace` cũng chứa trong `workspace` *(*workspace ⊆ workspace*)* thông qua chính sách g2. Ta có thể xác định rằng `g2(workspace, workspace)` là true.
3. Hành động `delete` không phù hợp với chính sách `delete` của vai trò `editor` trên không gian làm việc. Ta có thể xác định rằng `delete == delete` là true, nhưng vì không có chính sách nào cho phép `editor` xóa không gian làm việc, nên yêu cầu truy cập này bị từ chối.

Tương tự, các yêu cầu truy cập khác cũng được đánh giá dựa trên vai trò của người dùng trong không gian làm việc và các chính sách đã định nghĩa. Kết quả của mỗi yêu cầu sẽ cho biết liệu yêu cầu đó có được phép hay không, cùng với lý do dựa trên chính sách nào đã cho phép hoặc từ chối yêu cầu đó. Chi tiết có thể xem tại website playground của Casbin cho ví dụ trên tại https://editor.casbin.org/#E7VKBXR2T.

CHƯƠNG 4. XÂY DỰNG ỨNG DỤNG

Chương trình bày kết quả xây dựng Web App, bao gồm giao diện, các thành phần giao diện, và mô tả chức năng của từng thành phần.

4.1. Landing page

Hình 38: Giao diện landing page

STT	Tên	Loại	Mô tả
1	Button	Button	Clickable element

Bảng 16: Bảng mô tả

CHƯƠNG 5. KẾT LUẬN

Chương này trình bày phần kết luận của đồ án, nhằm tổng hợp và đánh giá các kết quả đạt được trong quá trình nghiên cứu, phân tích, thiết kế và xây dựng hệ thống. Nội dung chương tập trung vào việc đánh giá sản phẩm đã triển khai, nhận xét những thuận lợi, khó khăn, ưu điểm và hạn chế của hệ thống, đồng thời đề xuất các hướng phát triển trong tương lai nhằm nâng cao tính hoàn thiện và khả năng ứng dụng thực tế.

5.1. Kết quả đạt được

5.1.1. Về sản phẩm

Sau quá trình phát triển, nhóm đã hoàn thiện được hệ thống ghi chú với đầy đủ các tính năng đã đề ra:

- Ứng dụng Web cho người dùng cuối, cho phép quản lý và tổ chức ghi chú.
- Hỗ trợ cộng tác thời gian thực quyền hạn riêng.
- Tích hợp tìm kiếm ghi chú.
- Backend API với kiến trúc microservices, đảm bảo hiệu suất cao.
- Source code được tổ chức trong monorepo tại https://github.com/notopia-uit/notopia.
- Document website tại https://notopia-uit.github.io/notopia, gồm các diagram và OpenAPI contract.
- Source code báo cáo tại https://github.com/notopia-uit/report, viết bằng Typst.

5.1.2. Về công nghệ

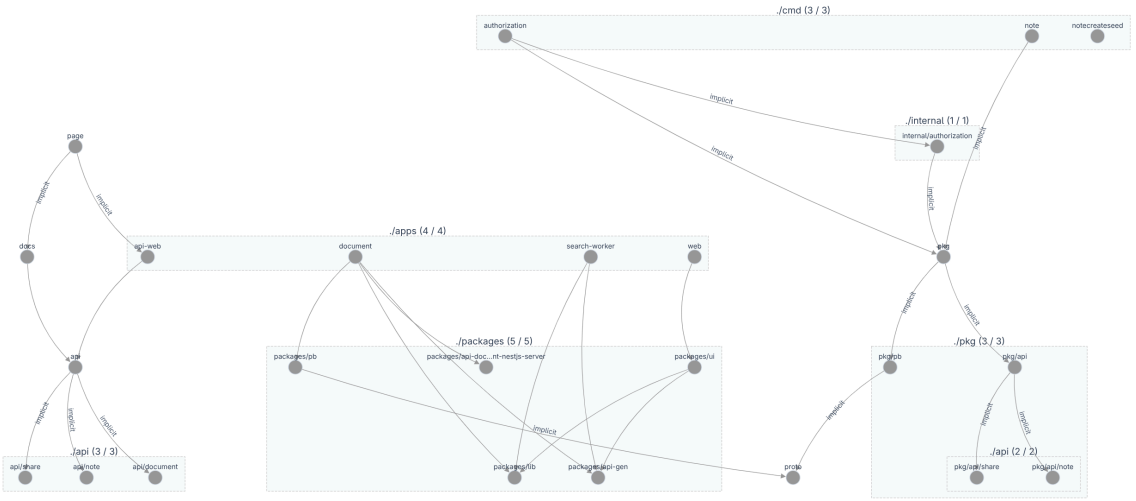
Dự án đã áp dụng thành công các công nghệ hiện đại:

- Tiếp cận mô hình block-based của BlockNote *(Phần 2.9, Phần 3.7)*, mô hình CDRT từ yjs và hỗ trợ cộng tác từ công nghệ Hocuspocus *(Phần 2.8)*.
- React, NextJS, TypeScript, TailwindCSS, Shadcnui ở frontend development
- Go *(Golang) (Phần 2.1)* và NestJS *(Phần 2.5)* framework ở backend services.
- Phân quyền với thư viện Casbin mạnh mẽ *(Phần 2.13, Phần 3.8)*.
- Quản lý người dùng thông qua Oauth2/OIDC với Authentik *(Phần 2.15)*.
- gRPC cho communication giữa các microservices *(Phần 2.11)*.
- Gateway API với Traefik *(Phần 2.12)*.
- Database management, kết hợp sử dụng SQL thuần và ORM *(Phần 2.7)*.
- Tìm kiếm với Meilisearch *(Phần 2.14)*.
- Docker containerization.
- CI/CD pipeline với GitHub Actions.
- Quản lý monorepo, CI pipeline với Nx, giúp tối ưu hóa workflow, cache riêng lẻ từng project nhỏ, và tăng hiệu quả phát triển *(Phần 2.20)*.
- Sử dụng Renovate Bot để tự động hoá việc cập nhật dependencies, giúp duy trì tính bảo mật và ổn định của hệ thống *([49])*.

Trong đó, dự án đã đặc biệt tiếp cận đến một số công nghệ đặc biệt mới như:

Công cụ quản lý monorepo Nx

Nhóm đã tối ưu hoá workflow, thiết lập từ code generate, lint, test, build, deploy cho từng project nhỏ trong monorepo bằng công cụ Nx.



Hình 39: Dependency graph của monorepo được sinh bởi Nx

Hạn chế CI của Nx

Đối với CI, Nx hướng developer sử dụng hệ sinh thái của Nx Cloud, nhưng nhóm đã tự xây dựng một giải pháp cache riêng cho Github Actions, `KevinNitroG/nx-cache-action` [50]. Script hoạt động theo cơ chế cache từng project thay vì toàn bộ cache lớn của cả workspace.

Script được lấy cảm hứng từ `raegen/nx` [51][7], cache tại project level. Nhưng cơ chế hoạt động khác biệt:

1. Script sẽ khởi động một NodeJS ExpressJS server implement OpenAPI Spec [52] chính thức từ Nx.
2. Forward lệnh Nx cho 1 child process, kèm theo thiết lập để Nx gửi request cache đến server.
3. Server nhận request cache, xử lý giao tiếp với Github Actions cache API thông qua `actions/toolkit/cache` [53].

Vì Nx sẽ lưu cache từng task ước chừng khoảng một tháng kể từ lần cuối sử dụng mới được xoá. Đối với dự án khi không sử dụng `KevinNitroG/nx-cache-action`, cache được lưu theo dạng toàn bộ project task cache, có thể lên đến khoảng 5GB trong mỗi lần lưu cache. Và hiển nhiên rằng, mỗi lần chạy thay đổi là một cache mới được tạo ra. Nếu có 10 commit được tạo ra và thay đổi source code, thì sẽ có 10 cache được tạo ra, tổng dung lượng cache có thể lên đến 50GB, vượt qua mức 10GB giới hạn của Github Actions cache. Hơn nữa, vẫn cần chừa dung lượng để cache node modules,

[7] `raegen/nx` không được hỗ trợ chính thức bởi Nx, đã deprecated

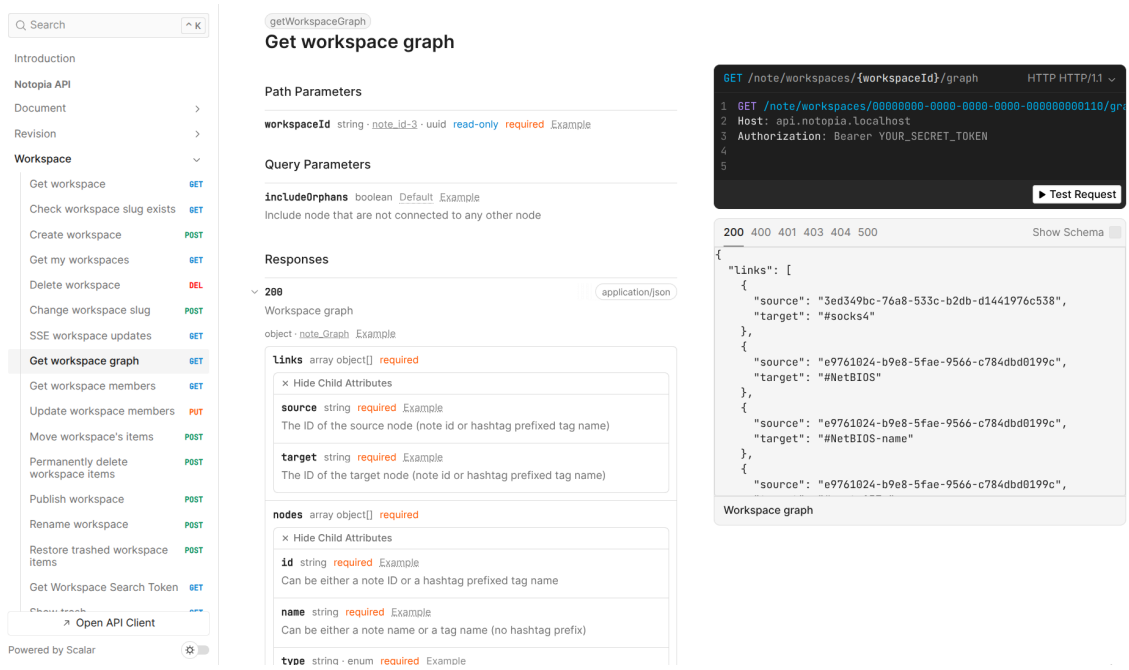
go packages, system dependencies, v.v…, nên việc cache toàn bộ project task cache là không tối ưu.

Khi sử dụng `KevinNitroG/nx-cache-action`, cache được lưu theo dạng từng project nhỏ, mỗi project task có thể chỉ khoảng vài trăm KB, đến hơn 10MB tùy vào task, và chỉ khi nào project đó thay đổi source code mới tạo cache mới. Điều này giúp tối ưu hóa dung lượng cache, tránh vượt quá giới hạn của Github Actions, và tăng hiệu quả cache hit.

Script có một nhược điểm là phải thông qua `actions/toolkit/cache` để download cache về local, và pipe cache vào lại Nx process. Có thể hiểu là cache đã được tải xuống nhưng lưu ở một nơi khác, và phải truyền vào Nx process thông qua HTTP request một lần nữa. Nhưng đây là cách dễ dàng nhất, không phải giao tiếp trực tiếp với Github Actions cache Rest API phức tạp hơn.

Contract First API development

Dự án áp dụng contract-first approach cho API development, đặc biệt với OpenAPI specification cho HTTP API được deploy và preview bằng Scalar, giao diện hiện đại, hỗ trợ mock API khi chưa có backend implementation, giúp frontend và backend phát triển song song hiệu quả.



Hình 40: API documentation được render từ OpenAPI spec bằng Scalar

SQLC Dynamic filter

SQLC là một công cụ code generation cho SQL queries, giúp tạo ra code type-safe cho database access *(Phần 2.7.2)*. Tuy nhiên, SQLC không hỗ trợ dynamic query, cũng không phải là một ORM hay query builder, mà chỉ đơn thuần là code generator cho SQL queries đã viết sẵn. Điều này gây khó khăn khi cần sinh dynamic `WHERE` conditions, ví dụ như khi có nhiều optional filters khác nhau. Trước đây *(cũng là phiên*

bản chính thức SQLC) phải sử dụng syntax `WHERE` và `CASE ... WHEN` ở dưới tầng SQL, ảnh hưởng đến hiệu năng dưới tầng database. Đặc biệt với `FOR UPDATE` queries, bắt buộc phải duplicate query cho từng trường hợp, dẫn đến code duplication và khó maintain.

Ví dụ đây là một query SQLC tiêu chuẩn cho việc lấy notes với nhiều optional filters:

```
SELECT
  *
FROM
  notes
CASE
  WHEN CARDINALITY(sqlc.arg('ids')::uuid[]) > 0
    THEN id = ANY(sqlc.arg('ids')::uuid[])
    ELSE TRUE
  END
CASE
  WHEN sqlc.narg('workspace_id')::uuid IS NOT NULL
    THEN folder_id IN (
      SELECT
        id
      FROM
        folders
      WHERE
        workspace_id = sqlc.narg('workspace_id')::uuid
    )
    ELSE TRUE
  END
CASE
  WHEN sqlc.narg('trashed_by')::text IS NOT NULL
    THEN (
      trashed_by = sqlc.narg('trashed_by')::text
      OR trashed_by IS NULL
    )
    ELSE TRUE
  END
CASE
  WHEN sqlc.arg('trashed_only')::boolean = true
    THEN trashed_by IS NOT NULL
    ELSE TRUE
  END;
```

Chương trình 8: Ví dụ SQL query với nhiều optional filters trong SQLC tiêu chuẩn

Custom plugin `vtuanjs/sqlc-gen-go` [54] *(được phát triển bởi anh Nguyễn Văn Tuấn)* hỗ trợ dynamic filter queries, giải quyết vấn đề không thể sinh dynamic WHERE conditions trong SQLC tiêu chuẩn, cũng như hỗ trợ dynamic `FOR UPDATE`. Plugin hoạt động bằng cách parse SQL query đã viết sẵn, nhận diện các phần có thể trở thành dynamic filter bằng các comment, và sinh ra code Go tương ứng để xây dựng dynamic query tại runtime.

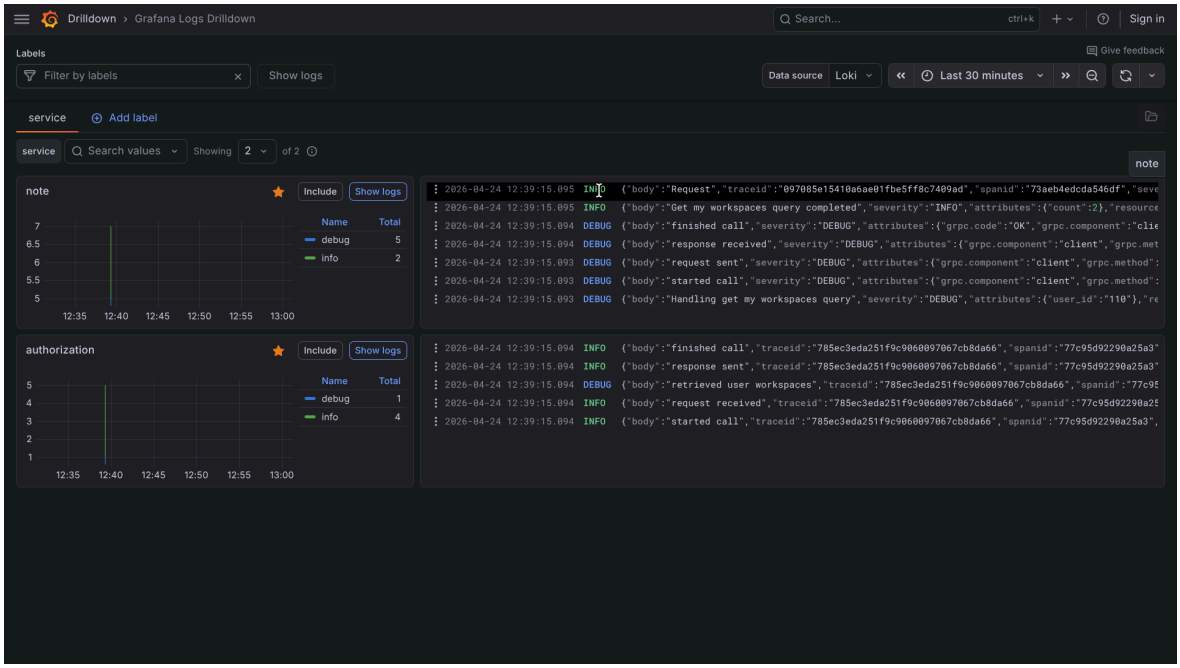
Dưới đây là ví dụ SQL query khi sử dụng với `vtuanjs/sqlc-gen-go`:

```
SELECT
  *
FROM
  notes
WHERE
  id = ANY(sqlc.narg('ids')::uuid[]) -- :if @ids
  AND folder_id IN ( -- :if @workspace_id
    SELECT
      id
    FROM
      folders
    WHERE
      workspace_id = sqlc.narg('workspace_id')::uuid
  )
  AND ( -- :if @trashed_by
    trashed_by = sqlc.narg('trashed_by')::text
    OR trashed_by IS NULL
  )
  AND trashed_by IS NOT NULL -- :if @trashed_only
FOR UPDATE -- :if @for_update
;
```

Chương trình 9: Ví dụ SQL query với dynamic filters sử dụng custom plugin `vtuanjs/sqlc-gen-go`

Observability

Dự án đã thiết lập một Observability Stack cơ bản, làm tiền đề cho việc phát triển ổn định về sau.



Hình 41: Log xem từ Grafana

5.2. Nhận xét

5.2.1. Thuận lợi

Trong quá trình thực hiện đề tài, hệ thống nhận được nhiều thuận lợi trong khâu nghiên cứu, phát triển và triển khai, tạo điều kiện cho việc hoàn thiện các chức năng và đạt được mục tiêu đề ra.

- Được sự hướng dẫn tận tình từ giảng viên, giúp định hướng rõ ràng, tránh đi lệch hướng, sử dụng công nghệ không cần thiết và giải quyết kịp thời các vấn đề phát sinh trong quá trình phát triển *(không sử dụng Neo4j để lưu quan hệ, mà sử dụng SQL Recursive CTE để build graph)*.
- Thư viện cốt lõi là BlockNote giúp xử lý nhanh việc xây dựng editor, tập trung vào việc phát triển các tính năng đặc thù của hệ thống thay vì phải xây dựng editor từ đầu.
- Sự hỗ trợ của các công nghệ hiện đại và framework mạnh mẽ như React, NextJS, NestJS, Go, gRPC, Traefik, Casbin, Authentik, Meilisearch đã giúp tăng tốc quá trình phát triển và đảm bảo hiệu suất cao cho hệ thống.
- Cộng tác hiệu quả trong nhóm, với sự phân công công việc rõ ràng, áp dụng phương pháp Contract First từ OpenAPI, Protobuf giúp giảm thiểu xung đột code và tăng hiệu quả phát triển.

5.2.2. Khó khăn

Dự án cũng gặp phải nhiều khó khăn và thách thức trong quá trình thực hiện, đòi hỏi sự nỗ lực và kiên trì từ nhóm phát triển để vượt qua và hoàn thiện hệ thống.

- Dữ liệu mẫu chuyển hoá từ Obsidian Vault từ `TrshPuppy/obsidian-notes` [55] không được chuẩn xác hoàn toàn. Vì cơ chế parse bằng text, không phải cây ngôn ngữ, nên các code block chứa comment như ký tự `#` hay shebang `#!` bị parse thành tag. Đồng thời, hệ thống không support nested tags như obsidian. Cũng như công việc chuyển hoá và seed vào service phức tạp *(markdown → custom markdown/HTML → BlockNote/yjs binary)*.
- Các block của Shadcn từ cộng đồng nhìn chung khá đa dạng tuy nhiên phần lớn chúng lại nằm trong các gói trả phí, nên vẫn phải tốn nhiều thời gian để code lại các phần giao diện từ các components nguyên thủy của Shadcn.
- Việc học và áp dụng nhiều công nghệ mới cùng lúc, tạo ra một learning curve khá dốc và đòi hỏi thời gian để làm quen và thành thạo. Dẫu vậy, các thành viên cũng đã có kinh nghiệm về một số công nghệ như Oauth2/OIDC, SQLC, nên đã phần nào giảm bớt khó khăn này.
- Độ phức tạp của kiến trúc microservices, đòi hỏi phải quản lý nhiều service nhỏ, đảm bảo communication giữa các service, và xử lý các vấn đề liên quan đến distributed systems như latency, fault tolerance, v.v…
- Mô hình Casbin rất trừu tượng và có learning curve cao, đòi hỏi phải hiểu rõ về mô hình RBAC để thiết kế chính sách phân quyền phù hợp và hiệu quả.

- Các khái niệm về event bus, event processor, command bus, command processor từ thư viện `ThreeDotsLabs/Watermill` nói riêng, và kiến trúc Event Drive Architecture nói chung, đòi hỏi phải hiểu rõ để thiết kế và triển khai đúng cách.
- `ThreeDotsLabs/watermill-kafka` sử dụng `IBM/sarama` không hỗ trợ subscribe regex topic, phải iterate toàn bộ topic bằng tay để subscribe.
- `vtuanjs/sqlc-gen-go` là một plugin mới, chưa được sử dụng rộng rãi, tính production ready chưa được kiểm chứng.
- Quá trình thiết lập monorepo, đặc biệt đối với TypeScript/JavaScript rất phức tạp, tốn nhiều thời gian để cấu hình cho đúng.
- RustFS là một công nghệ mới, chưa stable, còn gặp nhiều vấn đề. Trong đó, có vấn đề `rustfs/rustfs/issues/2587` - set server domains make RustFS cannot start [56][8].
- Việc xây dựng cây thư mục lúc đầu khá khó khăn do một số vấn đề về việc không tương thích với các components có sẵn của Shadcn, nhưng sau đó đã tìm ra được giải pháp từ `shadcn-ui/ui/issues/355` [57], nhưng vẫn phải dành thời gian để chỉnh sửa lại vì vẫn xảy ra một số lỗi.

5.2.3. Ưu điểm

Hệ thống ghi chú thông minh đã đạt được nhiều ưu điểm đáng kể.

- Trải nghiệm người dùng trực quan, giao diện hiện đại, thống nhất.
- Xử lý graph rất nhanh nhờ vào ngôn ngữ Go, hạn chế sử dụng con trỏ, tránh đối tượng trong heap trong quá trình Read.
- Kiến trúc microservices dễ mở rộng.
- Code maintainability cao nhờ vào việc áp dụng kiến trúc Clean Architecture, Domain Driven Design, CQRS đối với service `note` có business phức tạp nhất.
- DevOps practices tốt với CI/CD nhanh nhờ vào kinh nghiệm thiết lập, cũng như sử dụng Nx. 30 giây cho trường hợp cache hit toàn bộ project *(không có project nào thay đổi source code)*, đến 10 phút cho trường hợp ignore toàn bộ cache, build, lint, test, release. Nếu không được tối ưu, thời gian có thể lên đến 25 phút trong trường hợp chạy tuần tự các task cho toàn bộ project.
- Các service Go *(`note`, `authorization`)* đều có health check endpoint, đảm bảo tính sẵn sàng, tin cậy cao cho hệ thống.

5.2.4. Nhược điểm

Tuy đã đạt được nhiều ưu điểm, hệ thống cũng còn tồn tại một số nhược điểm cần được cải thiện trong tương lai.

- Độ phức tạp cao của microservices architecture.
- Cần nhiều tài nguyên hơn cho infrastructure, dù đã sử dụng Redpanda thay cho Kafka, RustFS cho MinIO, nhưng tổng RAM có thể lên đến 2.5GB khi chạy toàn bộ infrastructure. Đặc biệt với Authentik viết bằng Python, mức sử dụng RAM có

[8] `rustfs/rustfs/issues/2587` do thành viên nhóm phát hiện

thể lên đến 1.5GB chỉ trong quá trình development. Điều này phải chấp nhận đánh đổi về tính enterprise ready, feature rich, cộng đồng support tốt.
- Các service JS chưa dược thành công cấu hình gửi telemetry đến Observability Stack và health check endpoint.
- Khả năng xử lý lỗi từ các async event còn hạn chế, cần đảm bảo retry và dead letter queue cho các event thất bại cho toàn bộ các service. Hiện tại chỉ có service `note` có retry và dead letter queue.

5.3. Hướng phát triển

Dự án đã đạt được mục tiêu đề ra, tuy nhiên vẫn còn nhiều tiềm năng để phát triển và cải thiện trong tương lai. Dưới đây là một số hướng phát triển có thể xem xét trong tương lai để nâng cao tính hoàn thiện và khả năng ứng dụng thực tế của hệ thống.

- Tính năng subscription: Nhằm thương mại hóa sản phẩm dưới dạng SaaS.
- Tích hợp AI: Cung cấp các tính năng thông minh, thao tác trực tiếp với editor nhờ vào thư viện `@blocknote/xl-ai` [58] sử dụng thư viện `ai` đến từ Vercel, hybrid search nhờ vào tính năng hỗ trợ bởi Meilisearch [59], các tool thông qua API của hệ thống.
- Deploy: Hiện dự án đã được thiết lập quy trình release container các service, giúp nhanh chóng tiến đến bước triển khai và vận hành hệ thống trong môi trường deploy sau này.
- Sử dụng Istio Gateway: Thay thế Traefik trên môi trường deploy để tận dụng các tính năng nâng cao *(service mesh, authentication, authorization tại gateway, retry, v.v...)*, phù hợp với kubernetes.

5.4. Lời kết

Dự án đã đạt được mục tiêu đề ra và mang lại nhiều bài học quý giá cho nhóm phát triển. Hệ thống "Notopia - Ứng dụng ghi chú thông minh hỗ trợ quản lý tri thức bằng biểu đồ quan hệ" không chỉ là sản phẩm hoàn chỉnh mà còn là nền tảng để tiếp tục nghiên cứu và phát triển trong tương lai.

CHƯƠNG 6. DANH SÁCH TỪ VIẾT TẮT

Ký hiệu	Ý nghĩa
ABAC	Attribute-Based Access Control
ACL	Access Control List
API	Application Programming Interface
CDC	Change Data Capture
CI/CD	Continuous Integration/Continuous Deployment
CLI	Command-Line Interface
CRDT	Conflict-free Replicated Data Type
CSS	Cascading Style Sheets
DOM	Document Object Model
DX	Developer Experience
HTTP/2	Hypertext Transfer Protocol Version 2
ISR	Incremental Static Regeneration
JSX	JavaScript XML Syntax Extension
Nx	Monorepo Framework
OAuth2	OAuth 2.0 Authorization Framework
OIDC	OpenID Connect
ORM	Object-Relational Mapping
OTLP	OpenTelemetry Protocol
OpenAPI	Open API Specification
OpenTelemetry	Open Telemetry Framework
Protobuf	Protocol Buffers
RBAC	Role-Based Access Control
REST	Representational State Transfer
RPC	Remote Procedure Call
SEO	Search Engine Optimization
SQL	Structured Query Language
SQLC	SQL Compiler for Type-Safe Code Generation
SSE	Server-Sent Events
SSG	Static Site Generation
SSO	Single Sign-On

SSR	Server-Side Rendering
TOML	Tom’s Obvious, Minimal Language
TSX	TypeScript XML Syntax Extension
UML	Unified Modeling Language
gRPC	Google Remote Procedure Call

CHƯƠNG 7. TÀI LIỆU THAM KHẢO

[1] GoForJ, “Compile-time Dependency Injection for Go - Google Wire forked with caching support”. Truy cập: 17 Tháng Hai 2026. [Online]. Available at: https://github.com/goforj/wire

[2] “The proposal for generic methods for Go, from Robert Griesemer himself, has been officially accepted”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://www.reddit.com/r/golang/comments/1rfmjbq/the_proposal_for_generic_methods_for_go_from/

[3] Microsoft, “TypeScript Go - TypeScript Compiler Implementation”. Truy cập: 6 Tháng Hai 2026. [Online]. Available at: https://github.com/microsoft/typescript-go

[4] Meta, “React - A JavaScript Library for Building User Interfaces”. Truy cập: 12 Tháng Ba 2026. [Online]. Available at: https://react.dev/

[5] Vercel, “Next.js - The React Framework”. Truy cập: 22 Tháng Hai 2026. [Online]. Available at: https://nextjs.org/

[6] Redux Team, “Redux Toolkit - The Official Recommended Way to Write Redux”. Truy cập: 12 Tháng Hai 2026. [Online]. Available at: https://redux-toolkit.js.org/

[7] Tailwind Labs, “Tailwind CSS - Utility-First CSS Framework”. Truy cập: 4 Tháng Ba 2026. [Online]. Available at: https://tailwindcss.com/

[8] shadcn, “shadcn/ui - Beautifully Designed React Components”. Truy cập: 8 Tháng Ba 2026. [Online]. Available at: https://ui.shadcn.com/

[9] Microsoft, “TypeScript - JavaScript with Syntax for Types”. Truy cập: 28 Tháng Giêng 2026. [Online]. Available at: https://www.typescriptlang.org/

[10] PostgreSQL Global Development Group, “PostgreSQL - The World's Most Advanced Open Source Database”. Truy cập: 21 Tháng Ba 2026. [Online]. Available at: https://www.postgresql.org/

[11] sqlc Contributors, “sqlc - Compile SQL to Type-Safe Go Code”. Truy cập: 18 Tháng Giêng 2026. [Online]. Available at: https://sqlc.dev/

[12] TypeORM Contributors, “TypeORM - Object Relational Mapper”. Truy cập: 13 Tháng Ba 2026. [Online]. Available at: https://typeorm.io/

[13] “TypeScript Execute (tsx) | tsx”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://tsx.is/

[14] TypeCell, “BlockNote - Block-based Rich Text Editor”. Truy cập: 15 Tháng Hai 2026. [Online]. Available at: https://www.blocknotejs.org/

[15] Tiptap Collective, “Hocuspocus - Real-Time Collaboration Server”. Truy cập: 2 Tháng Ba 2026. [Online]. Available at: https://tiptap.dev/docs/hocuspocus

[16] Yjs Contributors, “Yjs - Shared Data Types for Collaborative Software”. Truy cập: 8 Tháng Hai 2026. [Online]. Available at: https://github.com/yjs/yjs

[17] Redocly, “Redocly - OpenAPI Documentation Platform”. Truy cập: 16 Tháng Ba 2026. [Online]. Available at: https://redocly.com/

[18] “Industry leading Developer Docs, SDKs & API Registry”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://scalar.com/

[19] Lubos, Nicolas Chaulet, Jordan Shatford, and HeyAPI Contributors, “HeyAPI - OpenAPI TypeScript Code Generator”. Truy cập: 20 Tháng Ba 2026. [Online]. Available at: https://heyapi.dev/openapi-ts/

[20] DeepMap and Contributors, “oapi-codegen - OpenAPI Go Code Generator”. Truy cập: 5 Tháng Hai 2026. [Online]. Available at: https://github.com/oapi-codegen/oapi-codegen

[21] “OpenAPI Generator allows generation of API client libraries (SDK generation), server stubs, documentation and configuration automatically given an OpenAPI Spec (v2, v3)”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://openapi-generator.tech/

[22] “OpenAPI Generator NestJS Server”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://github.com/OpenAPITools/openapi-generator/blob/master/docs/generators/typescript-nestjs-server.md

[23] Google, “gRPC - A High Performance RPC Framework”. Truy cập: 30 Tháng Giêng 2026. [Online]. Available at: https://grpc.io/

[24] OpenTelemetry Contributors, “OpenTelemetry - Observability Framework and Libraries”. Truy cập: 14 Tháng Hai 2026. [Online]. Available at: https://opentelemetry.io/

[25] Buf, “Buf - Protocol Buffers Compiler and Ecosystem”. Truy cập: 22 Tháng Ba 2026. [Online]. Available at: https://buf.build/

[26] “grpc/grpc-go: The Go language implementation of gRPC. HTTP/2 based RPC”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://github.com/grpc/grpc-go

[27] “stephenh/ts-proto: An idiomatic protobuf generator for TypeScript”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://github.com/stephenh/ts-proto

[28] Traefik Labs, “Traefik - Modern Edge Router and API Gateway”. Truy cập: 11 Tháng Ba 2026. [Online]. Available at: https://traefik.io/

[29] “Traefik JWT Plugin”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://github.com/traefik-plugins/traefik-jwt-plugin

[30] Casbin Contributors, “Apache Casbin - Authorization Library”. Truy cập: 15 Tháng Giêng 2026. [Online]. Available at: https://casbin.apache.org/

[31] Meilisearch Contributors, “Meilisearch - Open Source Search Engine”. Truy cập: 19 Tháng Hai 2026. [Online]. Available at: https://www.meilisearch.com/

[32] Authentik Team, “Authentik - Open Source Identity Provider”. Truy cập: 11 Tháng Hai 2026. [Online]. Available at: https://goauthentik.io/

[33] RustFS, “Instantly replace MinIO & S3. Zero GC, maximum throughput.”. Truy cập: 30 Tháng Ba 2026. [Online]. Available at: https://rustfs.com/

[34] Prometheus Contributors, “Prometheus - Open Source Monitoring System and Time Series Database”. Truy cập: 3 Tháng Hai 2026. [Online]. Available at: https://prometheus.io/

[35] Grafana Labs, “Loki - Log Aggregation System”. Truy cập: 1 Tháng Ba 2026. [Online]. Available at: https://grafana.com/oss/loki/

[36] Grafana Labs, “Tempo - Distributed Tracing Backend”. Truy cập: 25 Tháng Giêng 2026. [Online]. Available at: https://grafana.com/oss/tempo/

[37] Grafana Labs, “Alloy - OpenTelemetry collector”. Truy cập: 18 Tháng Ba 2026. [Online]. Available at: https://grafana.com/oss/alloy-opentelemetry-collector/

[38] Grafana Labs, “Grafana - The Open Observability Platform”. Truy cập: 18 Tháng Ba 2026. [Online]. Available at: https://grafana.com/

[39] Redpanda Inc., “Redpanda - Kafka-Compatible Streaming Platform”. Truy cập: 27 Tháng Hai 2026. [Online]. Available at: https://redpanda.com/

[40] “postgres_cdc | Redpanda Connect”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://docs.redpanda.com/redpanda-connect/components/inputs/postgres_cdc/

[41] “sql_select | Redpanda Connect”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://docs.redpanda.com/redpanda-connect/components/inputs/sql_select/

[42] Watermill Contributors, “Watermill - Event Stream Processing Library”. Truy cập: 9 Tháng Hai 2026. [Online]. Available at: https://watermill.io/

[43] Redis Team, “Redis - In-Memory Data Structure Store”. Truy cập: 24 Tháng Hai 2026. [Online]. Available at: https://redis.io/

[44] “nkonev/watermill-opentelemetry: OpenTelemetry integration for Watermill”. Truy cập: 29 Tháng Ba 2026. [Online]. Available at: https://github.com/nkonev/watermill-opentelemetry

[45] Nrwl, “Nx - Smart Monorepos and Fast Builds”. Truy cập: 5 Tháng Ba 2026. [Online]. Available at: https://nx.dev/

[46] “Wild Workouts - A Go DDD Example”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://github.com/ThreeDotsLabs/wild-workouts-go-ddd-example

[47] R. Laszczak, “Introduction to DDD Lite: When microservices in Go are not enough”. Truy cập: 22 Tháng Giêng 2026. [Online]. Available at: https://threedots.tech/post/ddd-lite-in-go-introduction/

[48] “BlockNote - Custom Schemas”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://www.blocknotejs.org/docs/features/custom-schemas

[49] “Renovate Docs”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: http://docs.renovatebot.com/

[50] KevinNitroG, “Nx Cache Action - GitHub Actions Cache Integration”. Truy cập: 9 Tháng Ba 2026. [Online]. Available at: https://github.com/KevinNitroG/nx-cache-action

[51] Raegen, “raegen/nx: Github action for executing nx commands cached with @actions/cache”. Truy cập: 9 Tháng Ba 2026. [Online]. Available at: https://github.com/raegen/nx

[52] “Remote Cache - Open API Specification | Nx”. Truy cập: 9 Tháng Ba 2026. [Online]. Available at: https://nx.dev/docs/guides/tasks%E2%80%93caching/self-hosted-caching#open-api-specification

[53] “actions/toolkit: The GitHub ToolKit for developing GitHub Actions. - Cache”. Truy cập: 9 Tháng Ba 2026. [Online]. Available at: https://github.com/actions/toolkit/tree/main/packages/cache

[54] N. V. Tuấn, “sqlc-gen-go - SQLC Plugin for Dynamic Filter Support”. Truy cập: 26 Tháng Ba 2026. [Online]. Available at: https://github.com/vtuanjs/sqlc-gen-go

[55] “All of TrshPuppy notes on topics including Hacking, Coding, Math, Data-Structures, and more”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://github.com/TrshPuppy/obsidian-notes

[56] “set server domains make rustfs cannot start #2587”. Truy cập: 18 Tháng Tư 2026. [Online]. Available at: https://github.com/rustfs/rustfs/issues/2587

[57] “Shadcn UI: Feature Request: Add a File Tree component #355”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://github.com/shadcn-ui/ui/issues/355

[58] “BlockNote - AI Rich Text Editing”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://www.blocknotejs.org/docs/features/ai

[59] “Meilisearch Hybrid Search”. Truy cập: 13 Tháng Tư 2026. [Online]. Available at: https://www.meilisearch.com/solutions/hybrid-search