TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

TRẦN THANH TỊNH – TRẦN KHÁNH LINH

XÂY DỰNG HỆ THỐNG CUNG CẤP DỊCH VỤ NHẬN DẠNG ÂM THANH TIẾNG VIỆT

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

TP. HCM, 2020

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

TRẦN THANH TỊNH – 1612704

TRẦN KHÁNH LINH – 1612339

XÂY DỰNG HỆ THỐNG CUNG CẤP DỊCH VỤ NHẬN DẠNG ÂM THANH TIẾNG VIỆT

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

GIÁO VIÊN HƯỚNG DẪN

TS. NGÔ HUY BIÊN

TP.HCM, 2020

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

TpHCM, ngày . . . tháng . . . năm 2020

Giáo viên hướng dẫn

[Kí tên và ghi rõ họ tên]

LỜI CẢM ƠN

Tri ân thầy - Tiến sĩ Ngô Huy Biên, người đã luôn trực tiếp hướng dẫn, định hướng cho hướng đi của luận văn, góp ý giúp đỡ nhiệt tình chúng em trong các vấn đề về kiến thức, nội dung, cách thức trình bày đồng thời luôn tạo điều kiện thoải mái nhất để chúng em có thể hoàn thành khóa luận, chúng em xin gửi đến thầy lời cảm ơn chân thành và sâu sắc nhất.

Chúng em xin gửi lời cảm ơn đến quý Thầy Cô trong khoa Công nghệ Thông tin của trường đại học Khoa Học Tự Nhiên đã tận tình giảng dạy nâng bước chúng em trong suốt gần 4 năm học vừa qua. Em xin chân thành cảm ơn Khoa Công Nghệ Thông Tin, trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Tp. Hồ Chí Minh đã tạo điều kiện thuận lợi cho chúng em trong quá trình học tập và thực hiện đề tài tốt nghiệp. Đồng thời chúng em cũng không quên gửi những lời cảm ơn chân thành đến những người thân trong gia đình và bạn bè đã giúp đỡ chúng em trong quá trình thực hiện luận văn, đặc biệt là quá trình định hướng kiến trúc tổng quan cho đề tài.

Do trình độ nghiên cứu và thời gian có hạn, chúng em đã cố gắng hết sức nhưng chắc chắn không tránh khỏi những thiếu sót và hạn chế. Rất mong nhận được sự góp ý và chỉ dẫn của quý Thầy Cô.

Cuối cùng, chúng em xin trân trọng cảm ơn và chúc sức khỏe quý Thầy Cô!

TpHCM, ngày . . . tháng . . . năm 2020

Sinh viên

Khoa Công Nghệ Thông Tin
Bộ môn Công Nghệ Phần Mềm

ĐỀ CƯƠNG CHI TIẾT

Tên đề tài: Xây dựng hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt

Giáo viên hướng dẫn: TS. Ngô Huy Biên

Thời gian thực hiện: 5/11/2019 đến 19/06/2020

Sinh viên thực hiện: Trần Thanh Tịnh - 1612704, Trần Khánh Linh - 1612339

Loại đề tài: Tìm hiểu công nghệ có ứng dụng minh hoạ

Nội dung đề tài:

Mục tiêu đề tài nhằm tìm hiểu, nghiên cứu phương pháp và công nghệ để xây dựng hệ thống có tính mở rộng cao, đáp ứng được số lượng truy cập lớn trong một thời gian nhất định thể hiện cụ thể trên dự án cung cấp dịch vụ nhận dạng âm thanh tiếng Việt.
Nội dung thực hiện chi tiết bao gồm:

1. Trình bày lý do xây dựng hệ thống cung cấp dịch vụ web nhận dạng âm thanh tiếng Việt.
2. Xây dựng dịch vụ web (API) để nhận 1 file âm thanh tiếng Việt và xuất ra nội dung ở dạng văn bản (có thể sử dụng các công cụ hay thư viện có sẵn).
3. Quản lý người đăng ký, khóa truy cập, số lượng truy cập vào dịch vụ.
4. Tạo thư viện SDK, ví dụ mẫu và tài liệu hỗ trợ lập trình viên sử dụng dịch vụ, với các ngôn ngữ khác nhau.
5. Thiết kế và hiện thực hóa khả năng mở rộng dịch vụ, khi số lượng truy cập cao.
6. Xây dựng một trang web demo việc sử dụng API để tải lên 1 file âm thanh tiếng Việt và xuất ra nội dung ở dạng văn bản.
7. Xây dựng một ứng dụng demo việc sử dụng SDK.

Kế hoạch thực hiện:

Thời gian thực hiện	Công việc thực hiện	Người thực hiện
05/11/2019 - 10/11/2019	- Nhận đề tài - Xây dựng bản kế hoạch sơ bộ cho các công việc cần thực hiện	Linh, Tịnh
11/11/2019 - 17/11/2019	- Khảo sát và dùng thử các hệ thống cung cấp dịch vụ mẫu có sẵn trên thị trường - Xây dựng 1 bản mẫu prototype, proof of concept (POC)	Linh, Tịnh
18/11/2019 - 25/11/2019	- Viết chương 1 luận văn - Thiết kế kiến trúc hệ thống và phác họa hệ thống một cách sơ bộ	Linh, Tịnh
26/11/2019 - 1/12/2019	- Tìm hiểu lý thuyết nền tảng về Microservice.	Linh, Tịnh
02/12/2019 - 07/12/2019	- Chỉnh sửa chương 1 luận văn	Linh, Tịnh
08/12/2019 - 16/12/2019	- Tìm hiểu lý thuyết nền tảng về Microservice. - Tìm hiểu công nghệ phù hợp cho mô hình (NestJS, EventStore, JS).	Linh, Tịnh
17/12/2019 - 25/12/2019	- Tìm hiểu lý thuyết nền tảng kiến trúc Command query responsibility segregation (CQRS). - Xây dựng bản mẫu cho mô hình CQRS.	Linh, Tịnh
26/12/2019 - 04/1/2020	- Viết chương 2 luận văn - Tìm hiểu lý thuyết nền tảng về mô hình Domain Driven Design (DDD).	Linh, Tịnh

	- Xây dựng bản mẫu cho mô hình Domain Driven Desin kết hợp với kiến trúc CQRS.	
05/01/2020 - 12/01/2020	- Chỉnh sửa chương 2 luận văn. - Tìm hiểu lý thuyết nền tảng về mô hình Event Sourcing. - Xây dựng bản mẫu cho mô hình Event Sourcing.	Linh, Tịnh
13/01/2020 - 20/01/2020	- Chỉnh sửa chương 2 luận văn. - Kết hợp mô hình Event Sourcing, DDD và CQRS dựa trên cơ sở dữ liệu không quan hệ (MongoDB).	Linh, Tịnh
21/01/2020 - 26/01/2020	- Thực hiện chi tiết mã nguồn quản lý khóa, các tác vụ cho dịch vụ. - Tìm hiểu chi tiết các quản lý luồng sự kiện và vận hành chi tiết của hệ thống.	Linh, Tịnh
27/01/2020 - 02/02/2020	- Thiết kế giao diện hệ thống. - Đưa tất cả hệ thống lên máy chủ Google Cloud và kiểm tra năng suất hệ thống. - Chỉnh sửa luồng sự kiện, phân quyền và viết tài liệu cài đặt.	Linh, Tịnh
03/02/2020 - 09/02/2020	- Viết chương 3. - Thực hiện cài đặt cho quản lý sự kiện trên Event Store, đảm bảo lưu vết và tái hiện sự kiện dịch vụ.	Linh, Tịnh

	- Viết mã nguồn giao diện cho phía người dùng cuối.	
10/02/2020 - 22/03/2020	- Chỉnh sửa chương 3. - Tiếp tục viết mã nguồn giao diện phía khách hàng. - Kết nối giữa giao diện và dịch vụ, đảm bảo luồng sự kiện, hiệu suất đạt yêu cầu và ổn định.	Linh, Tịnh
23/03/2020 - 29/03/2020	- Viết chương 4. - Viết tài liệu SDK và ví dụ sử dụng với nhiều ngôn ngữ . - Viết mã nguồn cho giao diện của người quản trị.	Linh, Tịnh
30/03/2020 - 05/04/2020	- Chỉnh sửa giao diện của người quản lý và kết nối với dịch vụ web. - Chỉnh sửa tài liệu SDK. - Viết mã nguồn giao diện cho ứng dụng Android .	Linh, Tịnh
06/04/2020 - 12/04/2020	- Chỉnh sửa chương 4 luận văn. - Viết mã nguồn kết nối dịch vụ Web cho ứng dụng Android. - Viết tài liệu cài đặt cho ứng dụng Android.	Linh, Tịnh
13/04/2020 - 26/04/2020	- Viết chương 5 luận văn. - Chỉnh sửa ứng dụng Android và kiểm tra lại hiệu năng hệ thống với các loại kiểm thử.	Linh, Tịnh

	- Tìm hiểu và nghiên cứu giải pháp mở rộng cho dịch vụ Web khi lượt tải tăng cao.	
27/04/2020 - 15/05/2020	- Chỉnh sửa chương 5 luận văn. - Thực hiện các giải pháp tăng hiệu suất hệ thống và đảm bảo hệ thống luôn hoạt động khi lượt sử dụng tăng cao.	Linh, Tịnh
16/05/2020 - 24/05/2020	- Xây dựng trang Web demo sử dụng dịch vụ. - Rà soát và chỉnh sửa cuốn luận văn.	Linh, Tịnh
25/05/2020 - 05/06/2020	- Hoàn chỉnh cuốn luận văn. - Thiết kế slide và chuẩn bị nội dung cho buổi bảo vệ luận văn. - Đóng gói mã nguồn và hoàn thiện tất cả tài liệu cài đặt, SDK.	Linh, Tịnh
06/06/2020 - 19/06/2020	- Hoàn tất cuốn luận văn. - Hoàn tất slide trình bày. - Luyện tập trình bày cho buổi bảo vệ.	Linh, Tịnh

Xác nhận của giáo viên hướng dẫn	**Ngày . . . tháng . . . năm 2020** **Sinh viên thực hiện**

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU LUẬN VĂN

1.1 GIỚI THIỆU ĐỀ TÀI

Trong thời đại bùng nổ công nghệ thông tin hiện nay, nhu cầu sử dụng công nghệ để thay thế con người thực hiện các tác vụ hằng ngày trong đời sống, công việc, học tập cũng như giao tiếp ngày càng tăng lên. Kéo theo đó là hàng loạt các công nghệ mới được đưa vào nghiên cứu với khả năng ứng dụng thực tiễn cao.

Âm thanh là một trong những yếu tố giúp con người trở nên khác biệt so với các chủng loài khác tồn tại trên Trái Đất. Con người sử dụng âm thanh, cụ thể là giọng nói để giao tiếp, thể hiện các cung bậc cảm xúc như vui, buồn, nổi giận,… cũng như giao tiếp với nhau trong đời sống hằng ngày.

1.1.1 Lợi ích mang lại

Với thời đại kinh tế phát triển dựa vào lao động có tay nghề cao như hiện nay, kéo theo đó quỹ thời gian của mỗi người trong ngày sẽ phải tối ưu hóa triệt để nhằm phục vụ cho chất lượng công việc. Cũng vì lẽ đó việc sử dụng các dịch vụ chuyển đổi giọng nói thay thế cho các tác vụ thủ công như ghi chép cho các cuộc họp, ghi chép các cuộc trao đổi điện thoại, soạn các tin nhắn văn bản trên điện thoại,… là rất cần thiết.

1.1.2 Sự đa dạng của ngôn ngữ và yêu cầu về hiệu năng (performance)

Có tới gần ba nghìn loại ngôn ngữ trên toàn thế giới, mỗi ngôn ngữ lại có các ngữ điệu khách nhau theo từng vùng miền, quốc gia mà cách phát âm khác nhau dẫn đến sự hình thành của nhiều phương ngữ lẫn các biến thể ngôn ngữ, dịch vụ nhận dạng tiếng nói vì thế mà bị giới hạn đi rất nhiều. Ở châu Âu và châu Mỹ đã xuất hiện rất nhiều các dịch vụ nhận dạng tiếng nói, tiêu biểu như Google Speech, Microsoft Azure (Speech Service). Tuy nhiên, các dịch vụ hiện có này chỉ hỗ trợ mạnh mẽ trong việc nhận dạng ngôn ngữ Tiếng Anh, còn những ngôn ngữ khác thì chưa được hỗ trợ và sử dụng phổ biến.

Ngoài các đặc điểm về độ phổ biến như trên, một đặc điểm sống còn ở các hệ thống cung

cấp dịch vụ như trên là phải đáp ứng được các yêu cầu về hiệu năng khi có một lượng lớn truy cập trong cùng một thời điểm. Vì các dịch vụ âm thanh này sẽ được tích hợp vào các hệ thống lớn hơn, cung cấp cho số lượng người dùng lớn hơn, kéo theo đó là nhu cầu ghi lại lịch sử sử dụng, truy cập, thống kê số lượng sử dụng trong các thời điểm khác nhau.

1.1.3 Ứng dụng thực tiễn

Dữ liệu âm thanh là một loại dữ liệu ẩn chứa trong đó nhiều thông tin riêng biệt, mà thông qua những thông tin được trích xuất ra từ các loại hình dữ liệu này, ta có thể vận dụng chúng vào trong những mục đích khác nhau. Sau đây sinh viên sẽ điểm qua các sản phẩm nổi bật để làm rõ sự đa dạng của loại hình ứng dụng này trên nhiều lĩnh vực.

- ❖ Amazon Echo

Amazon Echo là một thiết bị gia đình thông minh có cơ chế hoạt động giống một chiếc loa cầm tay. Bạn có thể thực hiện nhiều việc với Echo bằng cách ra lệnh giọng nói như: nghe nhạc, thiết lập lịch trình, cập nhật tin tức, … Tuy nhiên mục tiêu của Amazon Echo ra đời là không phải là để cạnh tranh trong lĩnh vực nhận diện giọng nói, mà là để phục vụ cho việc bán hàng qua mạng. Thay vì người dùng phải lên trang web tìm kiếm sản phẩm, thì nay có thể ra lệnh cho Echo thực hiện việc đó thông qua giọng nói.

- ❖ Google Assistant

Google Assistant là một trợ lý ảo cá nhân được phát triển bởi Google. Đây là một sản phẩm sinh ra cho cuộc cạnh tranh trong lĩnh vực nhận diện giọng nói, người dùng có thể tìm kiếm trên Internet, đặt lịch sự kiện và báo thức, tham gia vào các cuộc trò chuyện hai chiều giữa phần mềm và người dùng. Sự kết hợp giữa Google Assistant với Google

Maps, Google Photos, Yotube và một loạt các dịch vụ tiện ích khác tạo nên hệ sinh thái vô cùng mạnh mẽ.

- ❖ Cortana

Cortana là một trợ lý ảo được phát triển bởi Microsoft. Nhưng thay vì bước vào cuộc cạnh tranh trong lĩnh vực nhận diện giọng nói với Google Assistant, hay phục vụ bán hàng như Amazon Echo, thì Microsoft hướng Cortana là một công cụ tăng năng suất trong phần mềm và tập trung mạnh vào các doanh nghiệp. Tương lai của Cortana có thể không phải là một sản phẩm tiêu dùng thú vị mà là xương sống cho các giải pháp trợ lý ảo mà các doanh nghiệp có thể tùy chỉnh thành trợ lý ảo hoặc chatbox riêng của họ.

- ❖ Siri

Siri là một ứng dụng trợ lý ảo phát triển bởi Apple cho phép nhận diện giọng nói để làm các công việc mà bình thường phải làm bằng tay ví dụ như: gọi điện, soạn và gửi tin nhắn, mở đèn flash, mở ứng dụng cài trên máy, kiểm tra thời tiết, tạo các ghi chú, tìm đường, gửi email, bật nhạc hay điều chỉnh độ sáng màn hình… Người sử dụng có thể đưa ra câu hỏi và Siri sẽ tìm ra câu trả lời, hoặc người sử dụng có thể ra yêu cầu để Siri thực hiện trong một phạm vi cho phép.

Thông qua các ứng dụng đã trình bày, ta có thể thấy được nguồn dữ liệu âm thanh là một nguồn dữ liệu bao hàm rất nhiều thông tin, và bằng những kĩ thuật khai thác riêng biệt, ta có thể tận dụng nguồn thông tin này để phục vụ cho nhiều lĩnh vực kinh doanh chiến lược khác nhau.

1.1.4 Khả năng tích hợp dịch vụ vào các ứng dụng và thiết bị công nghệ cao

Hiện nay, với nhu cầu sử dụng ngày càng nhiều ở thị trường di động, việc phát triển ra một ý tưởng mới cho các nhà phát triển phần mềm ngày càng trở nên khó khăn. Dưới sự ra đời của một hệ thống nhận dạng tiếng nói hợp lý, các nhà phát triển phần mềm không chỉ có thể sáng tạo ra nhiều ý tưởng sản phẩm mới mà còn có thể nâng cấp cải tiến cách

thức sử dụng các sản phẩm có sẵn, từ đó ngày càng tạo ra sự tiện dụng và thu hút khách hàng nhiều hơn.

Ví dụ như Google Maps trước đây khi người dùng muốn tìm đường đến một địa điểm nào đó, họ phải nhập địa điểm và thực hiện các thao tác khá phức tạp khác, nhưng với việc tích hợp Google Assistant, Google Maps hiện nay đã cho phép người dùng điều khiển được việc chuyển hướng, chỉ đường bằng giọng nói trong khi họ đang bận rộn với việc lái xe và không thể sử dụng tay để điều khiển ứng dụng.

1.1.5 Tiềm năng kinh doanh lợi nhuận cao

Ứng dụng cho phép người dùng giao tiếp bằng giọng nói đang thu hút một lượng lớn người dùng trên nhiều độ tuổi, đặc biệt là nhóm người dùng bận rộn với nhiều việc cùng phải làm trong cùng lúc. Đối với những ứng dụng đã có sẵn trên thị trường hoàn toàn có thể sử dụng dịch vụ này để nâng cấp chức năng cho phép người dùng ra lệnh bằng giọng nói, còn đối với những ứng dụng mới, hoàn toàn có thể dựa vào việc khai thác dịch vụ này để phục vụ ý tưởng cho nhà phát triển phần mềm.

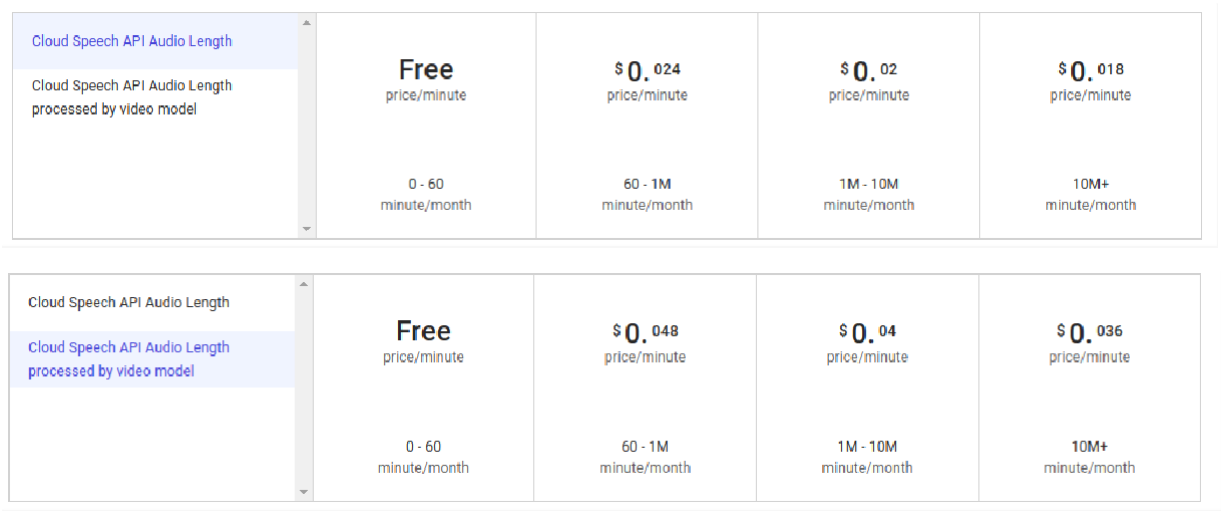
Với những công ty công nghệ phần mềm và thiết bị phần cứng, có những sự lựa chọn như sử dụng tiếp dịch vụ nhận dạng tiếng nói hoặc tùy chỉnh lại dịch vụ để phục vụ cho mục đích riêng của họ. Khả năng tích hợp dịch vụ này vào các ứng dụng phần mềm và các thiết bị phần cứng là rất cao, làm cho thị trường nhận dạng giọng nói phong phú, đa dạng và vì thế lợi nhuận đem lại từ thị trường này là rất to lớn.

1.2 LÝ DO LỰA CHỌN ĐỀ TÀI

Nhằm mục đích hệ thống hóa các kiến thức của bản thân trong quá trình học tập vào một sản phẩm có ý nghĩa thực tế, có tiềm năng đầu tư cao trong tương lai, nhóm sinh viên lựa chọn đề tài “Xây dựng hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt”.

Như đã trình bày ở trên, ứng dụng sử dụng dịch vụ nhận diện giọng nói ngày càng thu hút các nhà phát triển phần mềm lẫn người tiêu dùng phần mềm. Thị trường Việt Nam đang là thị trường tiềm năng vì đây là lĩnh vực mới. Hiện nay hầu hết các ứng dụng nhận dạng

giọng nói tiếng Việt đang sử dụng dịch vụ từ Google Cloud Speech API và một phần nhỏ sử dụng dịch vụ từ OpenFPT Speech. Có thể thấy, chi phí chi trả cho việc sử dụng các dịch vụ này khá cao. Ví dụ như Google Cloud Speech API, thì mức phí là:



Cloud Speech API Audio Length	Free price/minute	$0.024 price/minute	$0.02 price/minute	$0.018 price/minute
Cloud Speech API Audio Length processed by video model	0 - 60 minute/month	60 - 1M minute/month	1M - 10M minute/month	10M+ minute/month
Cloud Speech API Audio Length	Free price/minute	$0.048 price/minute	$0.04 price/minute	$0.036 price/minute
Cloud Speech API Audio Length processed by video model	0 - 60 minute/month	60 - 1M minute/month	1M - 10M minute/month	10M+ minute/month

Hình 1.5.1 Mức phí cho dịch vụ của Google (Nguồn: [cloud.google.com])

Chi phí cho dịch vụ được cung cấp bởi FPT.AI là:

Pricing

Plan	Quantity	Price (VND)	Description
Trial	60 minutes	Free of charge	60 minutes/ year No technical support
Business	2,000 minutes	1,400,000	Standard Technical Support
	5,000 minutes	3,100,000	
	10,000 minutes	5,400,000	
Business Premium	Over 10,000 minutes	Please contact for detailed pricing Hotline: 0911886353 Email: support@fpt.ai	

Hình 1.6 Mức phí cho dịch vụ của FPT.AI (Nguồn: [docs.fpt.ai/])

Xét trong phân khúc thị trường ở Việt Nam, một dịch vụ nhận diện giọng nói tiếng Việt với độ chính xác chưa cao và mức phí khá đắt đỏ đang vô hình làm chậm lại sự phát triển của các phần mềm tự động. Các nhà đầu tư mới bắt đầu sẽ không có đủ nguồn lực và nguồn vốn để chi trả cho việc phát triển một ứng dụng có tích hợp dịch vụ nhận diện giọng nói tiếng Việt.

Với hai lý do kể trên, nhóm sinh viên quyết định chọn đề tài "Xây dựng hệ thống cung cấp dịch vụ nhận dạng tiếng Việt" để tạo ra một dịch vụ có mức phí thấp hơn (hoặc miễn phí) và với độ chính xác chấp nhận được.

1.3 HƯỚNG PHÁT TRIỂN CỦA LUẬN VĂN

Tuy dịch vụ nhận dạng âm thanh trên thị trường khá nhiều, nhưng dịch vụ hỗ trợ nhận dạng âm thanh tiếng Việt lại thực sự hiếm hoi và còn nhiều hạn chế. Độ chính xác về khả năng nhận dạng âm thanh tiếng Việt chưa được tối ưu hóa và đồng thời mức chi phí cho dịch vụ lại khá tốn kém. Điều này làm cho việc tích hợp dịch vụ nhận dạng âm thanh

(ASR) vào ứng dụng ở Việt Nam còn gặp nhiều hạn chế nên chưa tận dụng và phát huy được tối ưu các tiềm năng trong lĩnh vực này. Vì vậy, để hoàn thành tốt đề tài luận văn, nhóm sinh viên tiến hành xây dựng một hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt sẽ hỗ trợ tối thiểu những tiêu chí sau:

- Hỗ trợ nhận dạng âm thanh tiếng Việt từ tập âm thanh tiếng Việt đã thu được với độ chính xác chấp nhận được.
- Mức chi phí để sử dụng dịch vụ là thấp nhất có thể hoặc miễn phí.

1.4 MỤC TIÊU CỦA LUẬN VĂN

Để hoàn thành tốt đề tài luận văn, bản luận văn và sản phẩm hệ thống dịch vụ cuối cùng của sinh viên sẽ đảm bảo tối thiểu các mục tiêu sau đây:

- Bản luận văn trình bày chi tiết, rõ ràng về lý do xây dựng hệ thống cung cấp dịch vụ web (API) nhận dạng âm thanh tiếng Việt.
- Bản luận văn trình bày các lý thuyết nền tảng và giải pháp để xây dựng một hệ thống cung cấp dịch vụ đáp ứng được số lượng truy cập dịch vụ cao. Từ đó áp dụng phát triển hệ thống. Bên cạnh đó, luận văn cũng trình bày các quy trình và cơ cấu cụ thể của sản phẩm luận văn để đạt được sản phẩm tốt nhất.
- Sản phẩm hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt của luận văn bảo đảm tính năng cơ bản là chuyển đổi một tập tin âm thanh tiếng Việt thành nội dung văn bản.
- Xây dựng một ứng dụng di động trên nền tảng Android để làm rõ mục đích cũng như ứng dụng hoá việc sử dụng dịch vụ mà hệ thống cung cấp.

1.5 PHẠM VI ĐỀ TÀI

Yêu cầu của đề tài là "Xây dựng hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt". Đúng như tên đề tài, sản phẩm của khóa luận sẽ tập trung vào việc xây dựng và phát triển một hệ thống cung cấp dịch vụ nhận dạng âm thanh nói tiếng Việt. Đồng thời, hệ thống hỗ trợ quản lý khoá truy cập cũng như cung cấp các báo cáo về việc sử dụng dịch vụ của người dùng.

CHƯƠNG 2: LÝ THUYẾT NỀN TẢNG

2.1. LÝ THUYẾT NỀN TẢNG CỦA MICROSERVICES

Trong những năm gần đây, thuật ngữ "microservices" hay kiến trúc microservice (microservice architecture) đã trở nên khá quen thuộc và rất được quan tâm trong giới phát triển phần mềm. Chúng ta có thể thấy rất nhiều dự án phần mềm được xây dựng và phát triển dựa trên microservices. Vậy microservices thực sự là gì?

Để trả lời câu hỏi trên, chúng ta sẽ tìm hiểu các khái niệm cốt lõi, các tính chất cũng như một số vấn đề gặp phải khi xây dựng và phát triển phần mềm theo hướng microservices.

2.1.1. Định nghĩa

Sự liên quan giữa thuật ngữ "microservices" và cách đặc biệt để xây dựng và phát triển phần mềm đến từ một buổi họp mặt với một số lượng ít các kĩ sư phần mềm tham dự. Tại đây, các kĩ sư thấy được sự tương đồng trong cách phát triển phần mềm của một tập hợp các công ty và đặt tên cho cách phát triển đó [?].

2.1.1.1. Microservices là một kiểu kiến trúc phần mềm (architectural style)

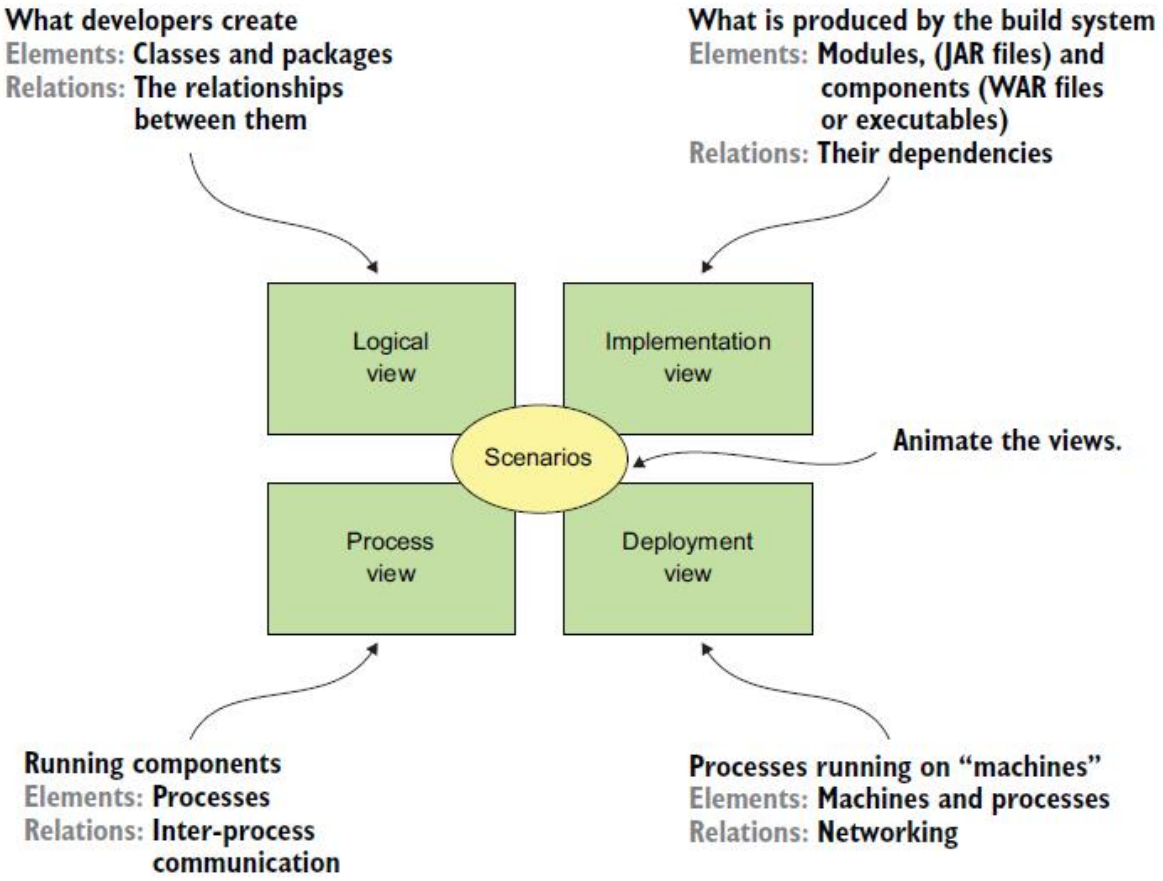
2.1.1.1.1. Kiến trúc phần mềm (software architecture)

Hãy bắt đầu phần định nghĩa bằng cách tiếp cận thuật ngữ kiến trúc phần mềm. Có rất nhiều định nghĩa về kiến trúc phần mềm. Nhưng định nghĩa mà chúng em muốn giới thiệu đến từ Len Bass – một kỹ sư phần mềm người Mỹ và những đồng nghiệp của ông tại Viện kỹ thuật phần mềm (Software Engineering Institute) [?]. Họ định nghĩa kiến trúc phần mềm như sau: Kiến trúc phần mềm của một hệ thống sử dụng máy tính (computing system) là một tập hợp các cấu trúc cần thiết để giải quyết những vấn đề của hệ thống, bao gồm các yếu tố phần mềm (software elements), mối quan hệ giữa chúng và những tính chất (properties) của cả hai [?]. Từ định nghĩa, kiến trúc của một ứng dụng là sự phân rã thành nhiều thành phần (các yếu tố phần mềm) và mối quan hệ giữa các thành

phần đó. Sự phân rã hệ thống đóng vai trò quan trọng vì hai lý do sau:

- Nó tạo điều kiện cho việc phân chia công việc và kiến thức. Nghĩa là, nó cho phép nhiều người (hoặc đội ngũ) với kiến thức chuyên môn khác nhau làm việc trên cùng một ứng dụng (application).
- Nó giải thích cách tương tác của các yếu tố phần mềm.

Cụ thể hơn, hãy tìm hiểu kiến trúc của một ứng dụng từ nhiều góc nhìn thông qua mô hình 4+1 (4+1 view model) của Phillip Krutchen [?]. Mô hình này giải thích bốn góc nhìn khác nhau của một kiến trúc phần mềm. Mỗi góc nhìn chứa một tập hợp yếu tố phần mềm và mối quan hệ giữa chúng.



Hình 2.1.1 Mô hình 4+1 sử dụng bốn góc nhìn và những kịch bản (thể hiện sự hợp tác trong việc xử lí yêu cầu giữa những yếu tố trong mỗi khía cạnh) để mô tả kiến trúc của một ứng dụng [?]

Trong đó:

- Góc nhìn logic (logical view): Những yếu tố phần mềm được tạo bởi lập trình viên. Trong ngôn ngữ hướng đối tượng (object-oriented languagues), những yếu tố này là lớp (classes) và gói (packages) – tập các lớp được nhóm lại với nhau theo một qui định nào đó. Mối liên hệ giữa chúng là mối quan hệ giữa lớp và gói, bao gồm kế thừa (inheritance), liên kết (associations), phụ thuộc (depends-on).
- Góc nhìn thực hiện (implementation view): đầu ra (output) của việc xây dựng hệ thống, bao gồm nhiều mô-đun (module) – gói chức năng (functional package), phản ánh một tính năng kỹ thuật (technical capability) trong ứng dụng và nhiều thành phần (component) – những đơn vị trong hệ thống có thể triển khai (deployable) hoặc thực thi (executable), gồm một hay nhiều mô-đun. Trong ngôn ngữ Java, một mô-đun là một tập tin (file) JAR (Java archive) và một thành phần là một tập tin WAR (web application archive) hoặc một tập tin JAR có thể thực thi. Mối liên hệ giữa chúng là mối quan hệ phụ thuộc giữa các mô-đun và kết hợp (composition) giữa mô-đun và thành phần.
- Góc nhìn quá trình (process view): Các thành phần trong thời gian thực thi (runtime). Mỗi thành phần là một tiến trình (process) và mối liên hệ giữa những tiến trình này tượng trưng cho sự giao tiếp liên tiến trình (interprocess communication - IPC).
- Góc nhìn triển khai (deployment view): mô tả sự ánh xạ các tiến trình vào máy móc (machines). Những yếu tố phần mềm bao gồm máy móc vật lý (physical) hoặc ảo (virtual) và những tiến trình. Góc nhìn này thể hiện mối quan hệ giữa những tiến trình và máy móc.
- Những kịch bản (scenarios) – tượng trưng cho +1 trong 4+1: kịch bản chính là nơi mô tả cách mà những thành phần kiến trúc trong một góc nhìn phối hợp để xử lí một yêu cầu.

Nhìn chung, mô hình 4+1 là một cách mô tả xuất sắc kiến trúc của một ứng dụng. Mỗi góc nhìn mô tả một khía cạnh quan trọng của kiến trúc, những kịch bản diễn giải cách mà những yếu tố của một góc nhìn hợp tác với nhau.

Vậy kiến trúc phần mềm có tầm ảnh hưởng như thế nào đối với ứng dụng? Một ứng dụng có hai loại yêu cầu. Một là yêu cầu chức năng (functional requirements) giải thích ứng dụng phải làm được gì. Và kiến trúc thể hiện tầm quan trọng của nó thông qua việc cho phép ứng dụng đáp ứng loại yêu cầu thứ hai: yêu cầu về chất lượng dịch vụ (quality of service requirements) hay còn gọi là yêu cầu phi chức năng (nonfunctional requirements). Yêu cầu phi chức năng xác định chất lượng ứng dụng trong thời gian thực thi (runtime qualities) như khả năng mở rộng (scalability) và độ tin cậy (reliability). Đồng thời, nó còn xác định chất lượng ứng dụng trong thời gian phát triển (development time qualities) như khả năng bảo trì (maintainability), kiểm tra (testability) và khả năng triển khai (deployability). Như vậy, kiến trúc quyết định mức độ ứng dụng đáp ứng những yêu cầu về chất lượng này.

2.1.1.1.2. Địa ngục nguyên khối (monolithic hell)

Microservices là một kiểu kiến trúc phần mềm. Và kiến trúc nguyên khối (monolithic architecture) cũng là một kiểu kiến trúc phần mềm. Trước khi kiến trúc microservice xuất hiện, các ứng dụng thường được xây dựng và phát triển dựa theo kiến trúc nguyên khối. Và kể từ khi kiến trúc microservice ra đời, đó cũng là lúc các ứng dụng doanh nghiệp hay ứng dụng có quy mô lớn, phức tạp (enterprise applications) thay đổi kiến trúc, cân nhắc lựa chọn microservices. Lý do cho sự thay đổi này là gì? microservices là kiểu kiến trúc như thế nào? Trước khi đến với định nghĩa tiếp theo về kiến trúc microservice, cùng tìm hiểu về địa ngục nguyên khối.

Kiến trúc nguyên khối là kiểu kiến trúc mà góc nhìn thực hiện được tổ chức dưới dạng một thành phần đơn lẻ có thể thực thi hay triển khai. Đối với các ứng dụng có quy mô tương đối nhỏ, kiến trúc này là một sự lựa chọn tốt. Tuy nhiên, khi ứng dụng phát triển, quy mô và yêu cầu chức năng dần trở nên phức tạp, sự lựa chọn này khiến ứng dụng rơi vào địa ngục nguyên khối.

- Sự phức tạp đe doạ các lập trình viên: ứng dụng có quy mô lớn và phức tạp khiến không một lập trình viên nào có thể hoàn toàn hiểu được để tiếp tục làm việc hiệu quả.

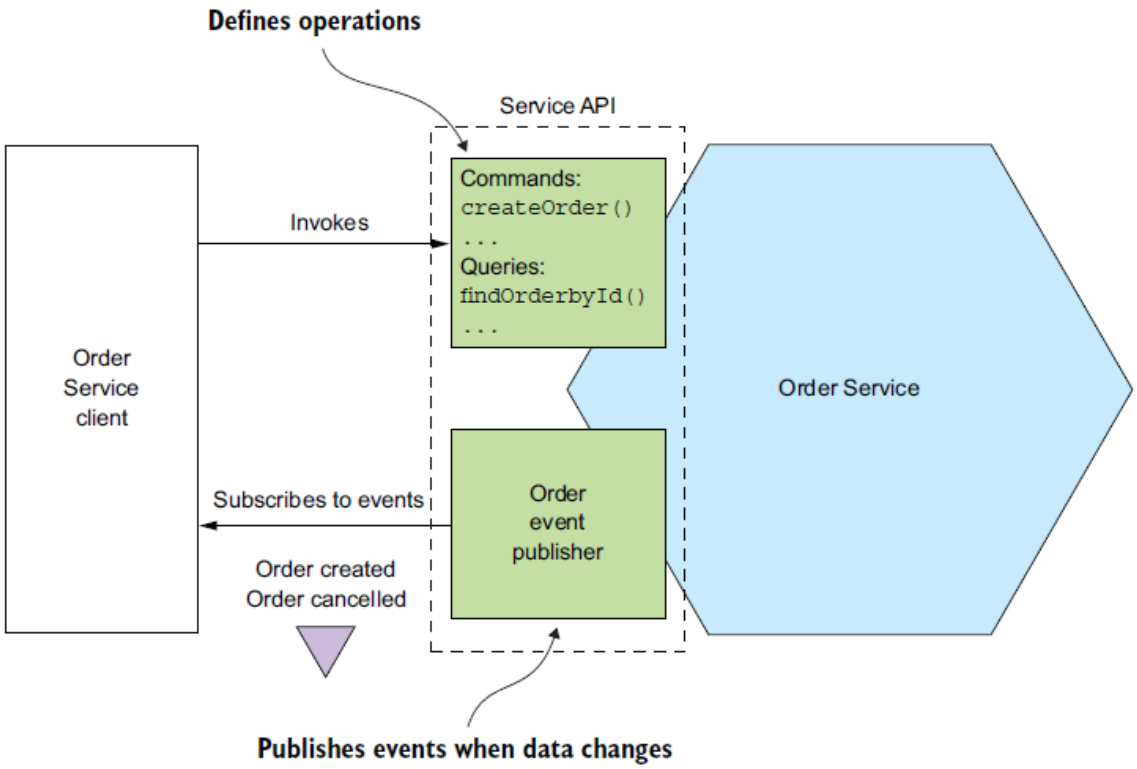
- Việc phát triển ứng dụng diễn ra chậm chạp: ứng dụng lớn khiến môi trường tích hợp để phát triển ứng dụng (Intergrated Development Environment – IDE) quá tải và hoạt động chậm. Từ đó, việc xây dựng và phát triển ứng dụng tốn rất nhiều thời gian.
- Con đường từ cài đặt đến triển khai là một quá trình gian nan: có rất nhiều lập trình viên tham gia viết mã (code) vào cùng một nơi và việc kiểm thử (testing) do đó chiếm khá nhiều thời gian.
- Khó mở rộng ứng dụng: nhiều mô-đun (module) khác nhau có những yêu cầu về tài nguyên khác nhau như bộ nhớ máy chủ có dung lượng lớn, máy chủ có nhiều CPU (Central Processing Unit),…
- Ứng dụng thiếu sự tin cậy: việc kiểm thử ứng dụng một cách kĩ lưỡng là rất khó vì quy mô của ứng dụng khá lớn. Điều này dẫn đến nhiều lỗi xảy ra trong quá trình sử dụng ứng dụng.
- Bị ràng buộc bởi những công nghệ lỗi thời: việc chuyển đổi sử dụng nền tảng (framework) mới hay ngôn ngữ mới rất mạo hiểm vì phải viết lại toàn bộ ứng dụng nguyên khối.

Để thoát khỏi địa ngục này, ứng dụng nên cân nhắc chuyển sang áp dụng kiến trúc microservice. Kiến trúc microservice có thể giải quyết hầu hết các vấn đề của địa ngục nguyên khối dựa vào định nghĩa kế tiếp của nó.

2.1.1.2. Microservices phân rã chức năng (functionally decompose) của ứng dụng thành một tập hợp những dịch vụ (service) có thể triển khai độc lập

2.1.1.2.1. Định nghĩa dịch vụ

Dịch vụ là một thành phần phần mềm có thể triển khai độc lập, một ứng dụng phiên bản nhỏ tập trung thực hiện một vài chức năng cụ thể. Ví dụ, hình bên dưới là góc nhìn bên ngoài của một dịch vụ, dịch vụ về đơn hàng (OrderService).



Hình 2.1.2 Một dịch vụ có một API đóng gói triển khai nội bộ của nó. API định nghĩa những hoạt động, được thực hiện bởi khách hàng của nó [?]

Trong đó:

- Mỗi dịch vụ có một API (Application Programming Interface) cho phép truy cập vào những hàm chức năng của nó.
- Có hai loại hoạt động (operation): lệnh (command) và truy vấn (query). API bao gồm nhiều lệnh, truy vấn và sự kiện (event). Một lệnh, ví dụ như createOrder() dùng để biểu diễn một hành động và cập nhật dữ liệu. Một hoạt động truy vấn, ví dụ như findOrderbyId() dùng để lấy dữ liệu. Một dịch vụ đồng thời công bố sự kiện (publish event), ví dụ như OrderCreated() hay OrderCancelled(). Khi dữ liệu thay đổi, dịch vụ sẽ công bố sự kiện và khách hàng (client) có thể theo dõi (subscribe) sự kiện này.

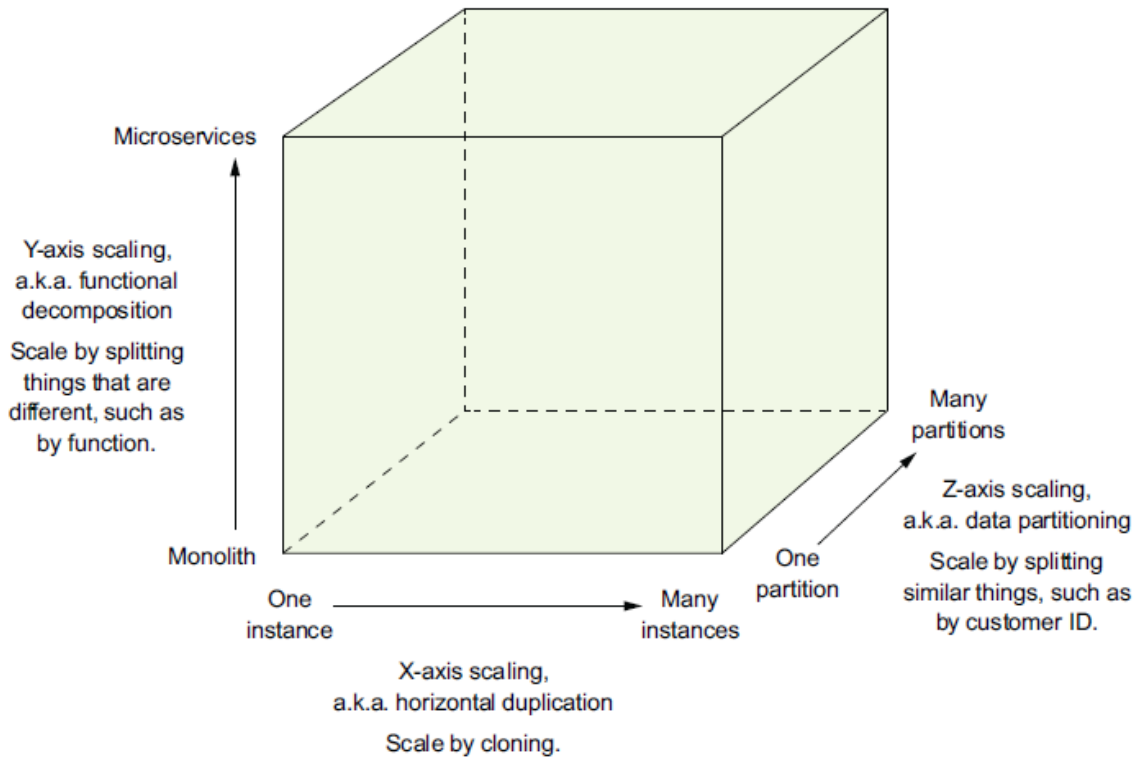
2.1.1.2.2. Dịch vụ trong kiến trúc microservice

Sự gắn kết (cohesion) hay nỗ lực để gom nhóm những dòng code liên quan với nhau là

một ý tưởng quan trọng khi nói về microservices. Điều này được củng cố bởi định nghĩa về nguyên lí đơn nhiệm (Single Responsibility Principle) của Robert C. Martin [?]: Tập hợp những thứ có thể thay đổi vì cùng một lí do, tách biệt những thứ có thể thay đổi vì những lí do khác nhau. Microservices có cách tiếp cận tương tự đối với các dịch vụ của nó. Một ứng dụng được xây dựng và phát triển theo kiến trúc microservice được phân rã thành nhiều dịch vụ. Dịch vụ là khía cạnh quan trọng nhất trong microservices. Mỗi dịch vụ có kiến trúc riêng và có thể sử dụng các nền tảng công nghệ riêng, thực hiện một tập chức năng chuyên biệt. Đặc biệt, các dịch vụ trong hệ thống ít phụ thuộc vào nhau (loosely coupled). Đây chính là một trong những tính chất quan trọng của kiến trúc này.

2.1.1.2.3. Khối lập phương về khả năng mở rộng (AKF Scale Cube)

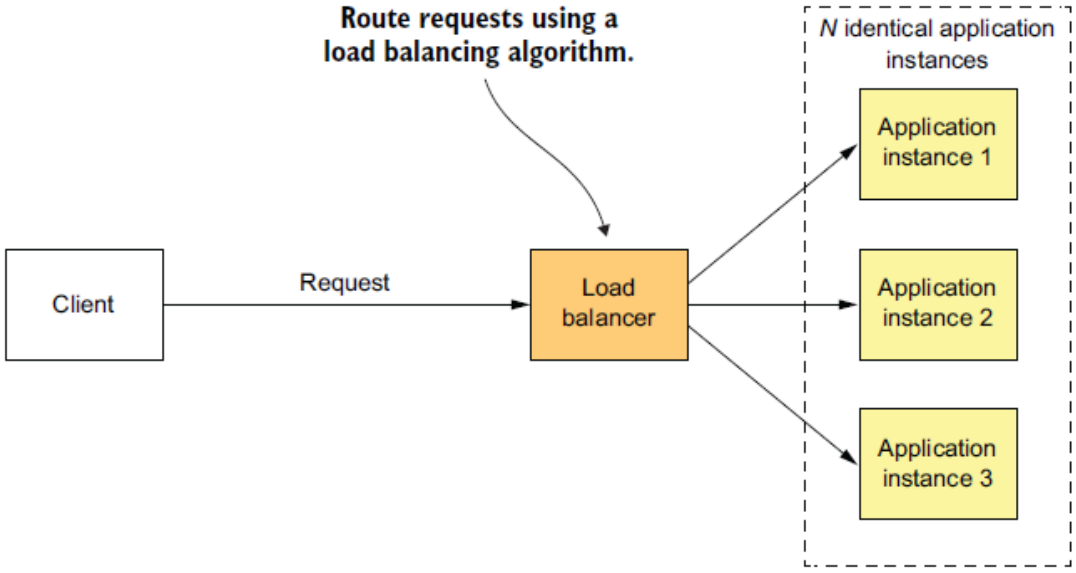
Kiến trúc microservice hoạt động tương đương trục Y trong khối lập phương về khả năng mở rộng. Khối lập phương này được giới thiệu trong quyển sách The Art of Scalability, viết bởi hai tác giả Martin L.Abbott và Michael T.Fisher [?]. Nó là một cách tiếp cận ba chiều để xây dựng ứng dụng có khả năng mở rộng vô hạn.



Hình 2.1.3 Khối lập phương về khả năng mở rộng định nghĩa ba cách khác nhau để mở rộng ứng dụng: trục X, Y, Z [20]

Trong đó:

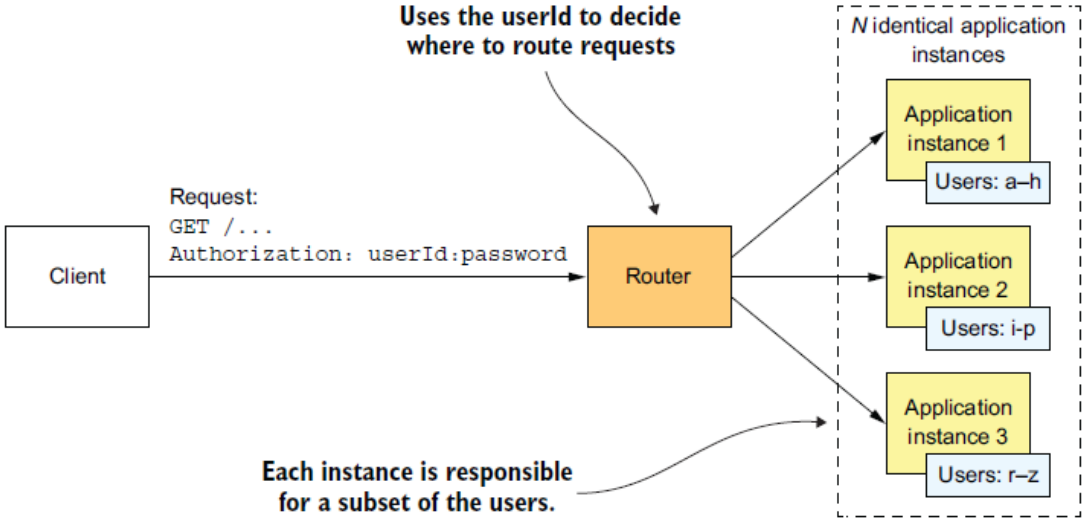
- Mở rộng theo trục X là cách thông thường để mở rộng một ứng dụng có kiến trúc nguyên khối. Bằng cách chạy N thực thể (instance) của một ứng dụng phía sau bộ cân bằng tải (load balancer), bộ cân bằng tải sẽ phân phối các yêu cầu cho N thực thể giống nhau, do đó mỗi thực thể chỉ xử lí 1/N yêu cầu. Đây là một cách tuyệt vời để cải thiện khả năng sẵn sàng phục vụ (availability) của ứng dụng.



Hình 2.1.4 Mở rộng theo trục X trong khối lập phương về khả năng mở rộng

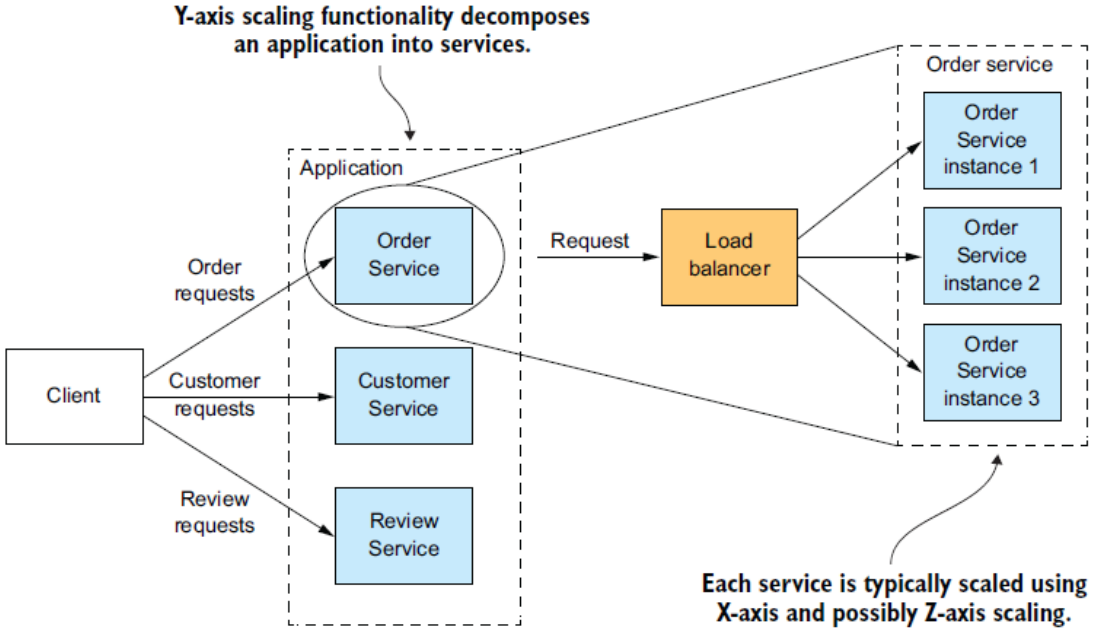
- Mở rộng theo trục Z là một cách cho phép ứng dụng xử lí khối lượng giao dịch và dữ liệu ngày càng tăng. Cũng bằng cách chạy nhiều thực thể của một ứng dụng, nhưng khác với cách mở rộng theo trục X, mỗi thực thể chịu trách nhiệm xử lí cho một phần dữ liệu. Bộ định tuyến (router) nằm phía trước các thực thể sử dụng một thuộc tính của yêu cầu để định tuyến nó đến thực thể thích hợp.

  Ví dụ, một ứng dụng có thể định tuyến yêu cầu dựa trên userId như hình bên dưới. Bộ định tuyến sử dụng userId lấy từ Authorization trong phần đầu của HTTP (Hypertext Transfer Protocol) trong mỗi yêu cầu để chọn lựa một trong N thực thể ứng dụng.



Hình 2.1.5 Mở rộng theo trục Z trong khối lập phương về khả năng mở rộng. Mỗi thực thể chịu trách nhiệm xử lí một tập con dữ liệu người dùng.

- Mở rộng theo trục Y chính là sự phân rã ứng dụng theo chức năng thành nhiều dịch vụ. Cả hai cách mở rộng theo trục X và Z đều không giải quyết được sự phát triển phức tạp ngày càng tăng của ứng dụng. Mở rộng theo trục Y là cách giải quyết cho vấn đề này. Nó được thể hiện thông qua ví dụ hình 2.1.6. Ứng dụng được phân rã thành các dịch vụ như OrderService, CustomerService, ReviewService. Mỗi dịch vụ độc lập này sẽ được mở rộng theo trục X và có thể cả trục Z. Dựa vào hình, có thể thấy OrderService được mở rộng theo trục X.



Hình 2.1.6 Mở rộng theo trục Y trong khối lập phương về khả năng mở rộng. Mỗi dịch vụ chịu trách nhiệm cho một chức năng đặc biệt.

Tóm lại, bản chất của kiến trúc microservice là sự phân rã chức năng của ứng dụng thành nhiều dịch vụ nhỏ.

2.1.1.2.4. So sánh microservices với SOA (Service Oriented Architecture)

Có một số ý kiến cho rằng kiến trúc microservice không khác gì kiến trúc hướng dịch vụ (SOA). Xét ở mức độ nâng cao, SOA và microservices đều là kiểu kiến trúc tổ chức hệ thống dưới dạng một tập hợp nhiều dịch vụ. Tuy nhiên, nếu tìm hiểu sâu hơn, có một vài sự khác nhau giữa hai kiến trúc này được liệt kê theo bảng bên dưới.

	SOA	**Microservices**
IPC	Sử dụng đường truyền thông minh (smart pipes), ESB (Enterprise Service Bus) là một ví dụ. Nó sử dụng những	Sử dụng đường truyền đơn giản, ví dụ như mô-đun trung gian trung chuyển tin nhắn (message broker). Nó sử

	giao thức (protocol) phức tạp, như SOAP (Simple Object Access Protocol) và các chuẩn dịch vụ web (Web Service – WS*) khác.	dụng những giao thức đơn giản và nhẹ như REST (Representational State Transfer) hay gRPC – một hệ thống mã nguồn mở RPC (Remote Procedure Call) phát triển bởi Google.
Dữ liệu	Mô hình dữ liệu (data model) toàn cục và các dịch vụ chia sẻ chung cơ sở dữ liệu.	Mô hình dữ liệu và mỗi dịch vụ có một cơ sở dữ liệu riêng.
Kích thước dịch vụ	Thường lớn hơn, một vài dịch vụ giống với một ứng dụng nguyên khối.	Bé hơn nhiều.

Bảng 2.1.1 So sánh SOA với microservices.

2.1.2. Tính chất

2.1.2.1. Microservices là kiến trúc lý tưởng cho hệ thống với quy mô lớn

Điều này được xác định dựa trên nguồn gốc ra đời của kiến trúc microservice. Tại phần mở đầu định nghĩa microservices, chúng em đã đề cập đến buổi họp mặt giữa các kĩ sư phần mềm, trong đó có James Lewis, ông kể lại: những vấn đề mà mọi người đang gặp phải đều liên quan đến việc xây dựng những hệ thống có quy mô lớn. Microservices được thiết kế để giải quyết những vấn đề cho hệ thống có quy mô lớn [?]. Điều đáng quan tâm không phải là định nghĩa về quy mô lớn của hệ thống. Thay vào đó, nó là tình huống mà hệ thống phát triển vượt khỏi ranh giới quy mô ban đầu được xác định, dẫn đến một số vấn đề đặc biệt khi muốn tác động thay đổi hệ thống. Nói cách khác, những vấn đề mới phát sinh do quy mô của hệ thống thay đổi.

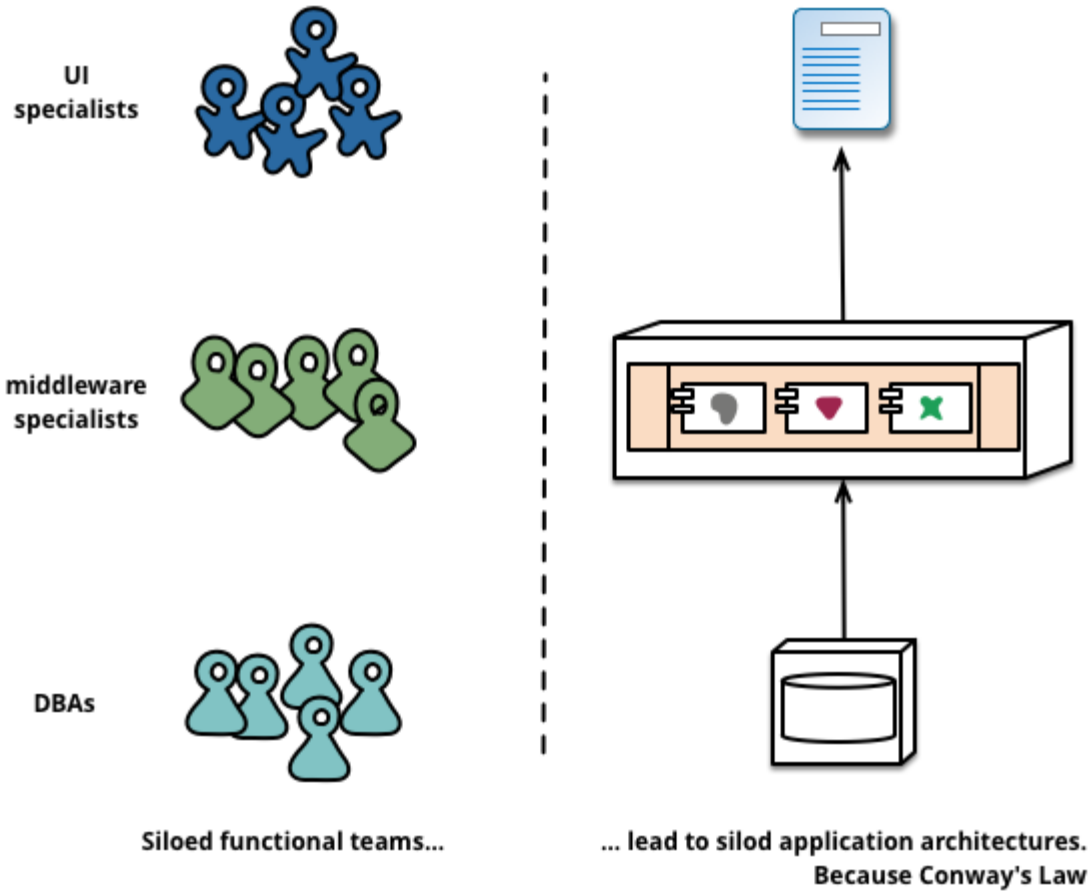
2.1.2.2. Kiến trúc microservice như một hình thức mô-đun hoá (modularity)

Khi phát triển những ứng dụng lớn và phức tạp, mô-đun hoá là một việc thiết yếu. Vì sự

phức tạp, ứng dụng phải được chia thành các mô-đun, mỗi mô-đun được phát triển và quản lí bởi một nhóm khác nhau. Kiến trúc microservice sử dụng các dịch vụ của nó như một mô-đun. Mỗi dịch vụ có một API – một ranh giới khó để xâm phạm. Nó không cho phép truy cập vào những lớp nội bộ, dẫn đến việc dễ dàng bảo toàn tính mô-đun hoá của ứng dụng.

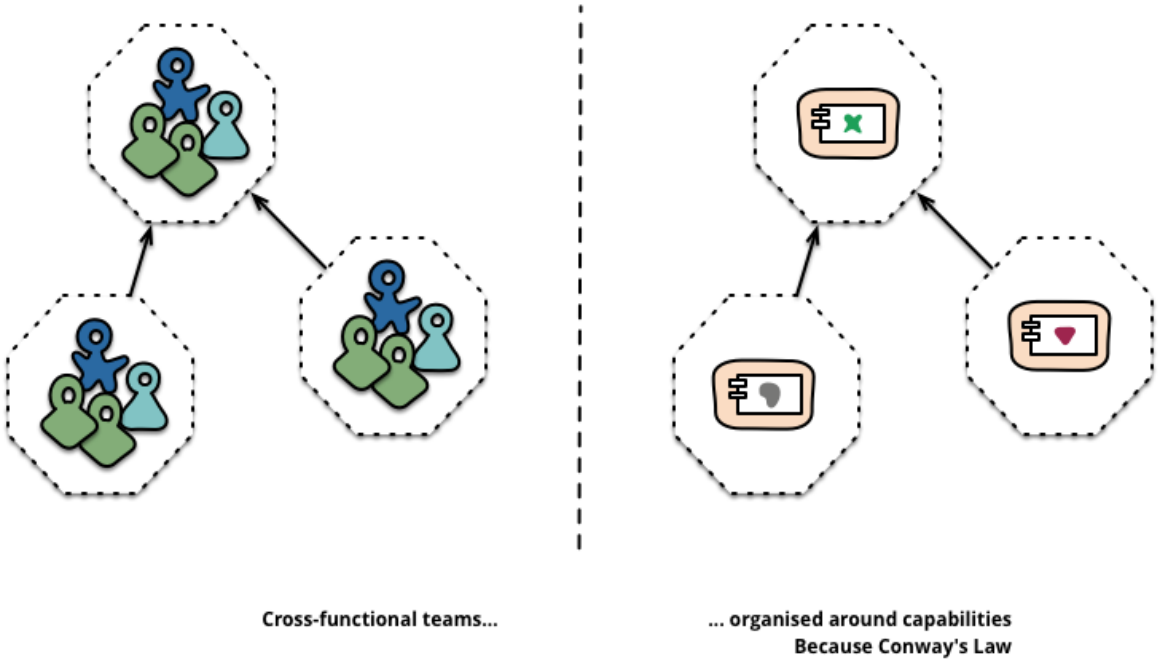
2.1.2.3. Các dịch vụ trong microservices được tổ chức dựa trên khả năng về nghiệp vụ

Khi muốn tổ chức một ứng dụng lớn thành nhiều phần nhỏ hơn, thông thường ứng dụng được chia dựa trên tính năng kĩ thuật, hình thành những nhóm làm về giao diện người dùng, những nhóm xử lí logic phía máy chủ, những nhóm quản lí cơ sở dữ liệu. Khi chia nhóm theo cách này, một vài yêu cầu thay đổi đơn giản sẽ khiến dự án mất khá nhiều thời gian để đáp ứng. Đây là một ví dụ cho luật của Conway: Bất cứ một tổ chức nào thiết kế một hệ thống sẽ tạo ra một thiết kế có cấu trúc rất giống với cấu trúc của hệ thống giao tiếp của tổ chức đó [?].



Hình 2.1.7 Luật của Conway khi áp dụng vào cách tổ chức ứng dụng dựa trên tính năng kĩ thuật [?]

Cách tổ chức ứng dụng theo kiến trúc microservice lại khác. Nó chia ứng dụng thành nhiều dịch vụ dựa trên khả năng về nghiệp vụ. Mỗi dịch vụ này triển khai phần mềm theo nhiều mặt phục vụ cho nghiệp vụ của nó, bao gồm giao diện người dùng, dữ liệu lưu trữ liên tục và các kết nối với dịch vụ bên ngoài nếu có. Vì vậy, các nhóm trở nên đa chức năng, trang bị đầy đủ những kĩ năng cần thiết để phát triển dịch vụ như trải nghiệm người dùng, cơ sở dữ liệu và quản lí dự án.



Hình 2.1.8 Ranh giới giữa các nhóm phát triển củng cố ranh giới giữa các dịch vụ [?]

Khi tổ chức dịch vụ trong microservices dựa trên khả năng nghiệp vụ, vì khả năng nghiệp vụ của một tổ chức thường ổn định nên kiến trúc của nó cũng sẽ tương đối ổn định. Trong khi từng thành phần trong kiến trúc có thể phát triển khi nghiệp vụ của nó thay đổi cách thực hiện, bản thân kiến trúc vẫn được giữ nguyên.

2.1.2.4. Các dịch vụ trong microservices có thể triển khai độc lập, ít phụ thuộc vào nhau

Một tính chất quan trọng của kiến trúc microservice đó là các dịch vụ hoạt động độc lập, ít phụ thuộc nhau. Tất cả tương tác đến một dịch vụ đều phải thông qua API của nó. Điều này cho phép dịch vụ che giấu chi tiết cài đặt bên trong, cho phép nó thay đổi cách cài đặt mà không ảnh hưởng đến khách hàng của nó. Tính chất này là chìa khoá để cải thiện khả năng bảo trì và kiểm thử ứng dụng trong thời gian phát triển ứng dụng.

Để các dịch vụ ít ràng buộc nhau và chỉ giao tiếp với nhau qua API, mỗi dịch vụ cần có cơ sở dữ liệu riêng. Trong thời gian phát triển dịch vụ, lập trình viên có thể thay đổi lược đồ cơ sở dữ liệu (database schema) của dịch vụ mà không cần phải làm việc với những lập trình viên phát triển các dịch vụ khác. Trong thời gian thực thi, do các dịch vụ cô lập

với nhau, một dịch vụ không thể bị khoá chỉ vì một dịch vụ khác đang giữ khoá cơ sở dữ liệu (database lock).

2.1.2.5. Giao tiếp liên tiến trình là một phần quan trọng

Một ứng dụng xây dựng dựa trên kiến trúc microservice là một hệ thống phân tán (distributed system). Do đó, việc giao tiếp liên tiến trình là một phần quan trọng trong microservices. Giao tiếp liên tiến trình là một phương pháp cho phép các tiến trình riêng biệt trao đổi, giao tiếp với nhau. Nó cho phép các dịch vụ có thể tương tác với nhau và với thế giới bên ngoài.

2.2. LÝ THUYẾT MẪU THIẾT KẾ DOMAIN DRIVEN DESIGN – DDD

Trong phần này, chúng em sẽ trình bày các kiến thức nền tảng về cách tiếp cận đối với các mô hình phức tạp bằng mẫu thiết kế Domain Driven Design (DDD) qua các ví dụ thực tiễn, từ đó thu được các kiến thức nền tảng về cách xây dựng và phát triển đối với các dự án lớn có yêu cầu nghiệp vụ phức tạp.

2.2.1. Định nghĩa

DDD là một phương pháp phân tích và thiết kế khuyến khích sử dụng các mô hình và các ngôn ngữ phổ biến để thu hẹp cách tiếp cận giữa các yêu cầu kinh doanh và nhóm phát triển bằng cách thúc đẩy sự hiểu biết chung về các khái niệm nghiệp vụ.

Để xây dựng một phần mềm tốt, lập trình viên cần phải hiểu về phần mềm đó, nghĩa là phải có kiến thức về các yêu cầu nghiệp vụ có trong đó. Để đi sâu hơn, chúng ta sẽ xem xét ví dụ về kiến thức nghiệp vụ cần có cho việc xây dựng “Hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt”. Đối với góc độ của khách hàng, muốn sử dụng dịch vụ sẽ phải mua một “mã truy cập”, mà qua đó khách hàng sẽ tạo ra những “dự án” để chứa những mã này, đồng thời có thể chia sẻ quyền truy cập đối với những người trong dự án của mình.

Hình 2.2.1 Mô tả trừu tượng hóa nghiệp vụ thành đối tượng trong mô hình dự án từ mã truy cập, dự án, người dùng.

Từ những yêu cầu nghiệp vụ thật này, chúng sẽ được trừu tượng hóa trong mã nguồn với tên gọi là domain, thể hiện cho các đối tượng nghiệp vụ thật trong hiện tại.

2.2.2. Đặc điểm và các thành phần

2.2.2.1. Ngôn ngữ chung

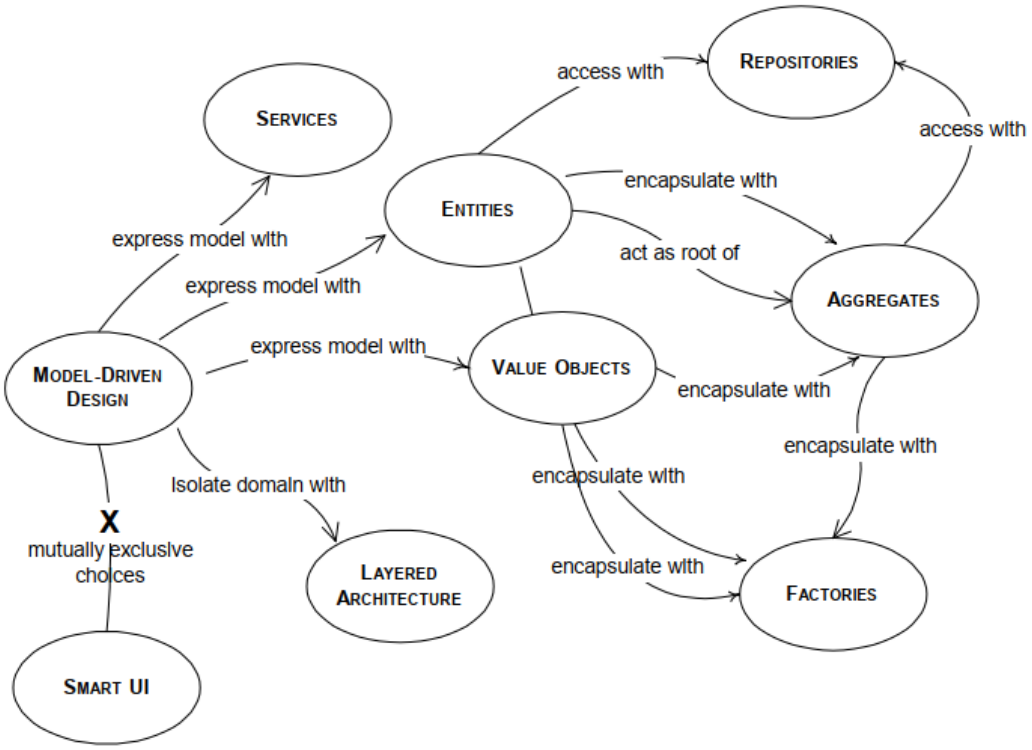
Trong phần trước chúng ta có thể thấy sự cần thiết không thể thiếu của việc phát triển mô hình cho domain qua sự làm việc giữa khách hàng nắm vững kiến thức nghiệp vụ và đội ngũ phát triển phần mềm. Tuy nhiên, cách làm này ban đầu thường gặp khó khăn về rào cản giao tiếp cơ bản. Lập trình viên chưa có nhiều hiểu biết về nghiệp vụ, trong khi đó khách hàng có kiến thức hạn chế về công nghệ. Để vượt qua sự khác nhau, chúng ta xây dựng mô hình, chúng ta phải trao đổi, giao tiếp ý tưởng về mô hình, về những thành phần liên quan đến mô hình, cách chúng ta liên kết chúng, chúng có liên quan với nhau hay không. Giao tiếp ở mức độ này là tối quan trọng cho sự thành công của dự án. Nếu ai đó nói điều gì đó và người khác không hiểu, hoặc tệ hơn, hiểu sai, thì liệu chúng ta có cơ hội tiếp tục dự án không?

2.2.2.2. Thiết kế hướng mô hình

Các phần trước đã nhấn tầm quan trọng của cách tiếp cận tới quy trình phát triển phần mềm tập trung vào lĩnh vực nghiệp vụ. Chúng ta đã biết nó có ý nghĩa then chốt để tạo ra một đối tượng có gốc rễ là domain. Ngôn ngữ chung nên được thực hiện đầy đủ trong suốt quá trình mô hình hóa nhằm thúc đẩy giao tiếp giữa các chuyên gia phần mềm với khách hàng, và khám phá ra các khái niệm chính của domain nên được sử dụng trong đối tượng. Mục đích của quá trình mô hình hóa là nhằm tạo ra một đối tượng tốt. Bước tiếp theo là hiện thực nó trong mã nguồn. Đây là một giai đoạn quan trọng không kém của quá trình phát triển phần mềm.

2.2.2.3. Các khối ghép của mô hình

Phần sau đây của chương này sẽ giới thiệu các khuôn mẫu quan trọng nhất được sử dụng trong DDD. Mục đích của những *khuôn mẫu* này là để trình bày một số yếu tố chính của mô hình hóa hướng đối tượng và thiết kế phần mềm từ quan điểm của DDD. Sơ đồ sau đây là một biểu đồ của các khuôn mẫu sẽ được trình bày và các mối quan hệ giữa chúng.



Hình 2.2.2 Biểu đồ và quan hệ của các khuôn mẫu trong mô hình DDD.

Khi chúng ta tạo ra một ứng dụng phần mềm, một lượng lớn thành phần của ứng dụng không liên quan trực tiếp đến nghiệp vụ, nhưng chúng là một phần của hạ tầng phần mềm hoặc phục vụ chính bản thân ứng dụng. Nó có khả năng và ổn cho phần nghiệp vụ của ứng dụng nhỏ so với các phần còn lại, vì một ứng dụng điển hình chứa rất nhiều đoạn mã liên quan đến truy cập cơ sở dữ liệu, tệp hoặc mạng, giao diện người dùng,…

Trong một ứng dụng hướng đối tượng thuần túy, giao diện người dùng, mã truy cập cơ sở dữ liệu, và các đoạn mã hỗ trợ khác thường được viết trực tiếp vào trong các đối tượng nghiệp vụ. Thêm vào đó, đối tượng nghiệp vụ này lại được nhúng vào trong các hành vi của giao diện người dùng và các kịch bản cơ sở dữ liệu. Đôi khi điều này diễn ra bởi vì nó là cách dễ nhất để làm cho mọi việc trở nên nhanh chóng.

Tuy nhiên, khi các đoạn code liên quan đến nghiệp vụ được trộn lẫn giữa các tầng lại với nhau, nó trở nên vô cùng khó khăn cho việc đọc cũng như suy nghĩ về chúng. Các thay đổi ở giao diện người dùng cũng có thể thực sự thay đổi cả logic nghiệp vụ. Để thay đổi

logic nghiệp vụ có thể yêu cầu tới truy vết tỉ mỉ các đoạn mã của giao diện người dùng, cơ sở dữ liệu, hoặc các thành phần khác của chương trình. Mô hình phát triển hướng đối tượng trở nên phi thực tế. Kiểm thử tự động sẽ khó khăn. Với tất cả các công nghệ và mã nguồn liên quan trong từng hoạt động, chương trình cần được giữ rất đơn giản hoặc không thì sẽ biến chúng trở nên không thể hiểu được.

Do đó, hãy phân chia một chương trình phức tạp thành các lớp. Phát triển một thiết kế cho mỗi lớp để chúng trở nên gắn kết và chỉ phụ thuộc vào các tầng bên dưới. Tuân theo khuôn mẫu kiến trúc chuẩn để cung cấp liên kết lỏng lẻo tới các tầng phía trên. Tập trung các đoạn mã liên quan đến các đối tượng nghiệp vụ trong một lớp và cô lập chúng khỏi lớp giao diện người dùng, ứng dụng, và hạ tầng. Các đối tượng nghiệp vụ, được giải phóng khỏi việc hiển thị, lưu trữ chính chúng, hay quản lý các nhiệm vụ của ứng dụng, vân vân, và có thể tập trung vào việc biểu hiện của domain. Điều này cho phép một đối tượng tiến hóa đủ phong phú và rõ ràng, nhằm nắm bắt kiến thức nghiệp vụ thiết yếu và áp dụng nó vào làm việc.

Một giải pháp kiến trúc chung cho DDD chứa bốn lớp (trên lý thuyết)

- Giao diện người dùng: chịu trách nhiệm trình bày thông tin tới người dùng và thông tin lệnh của người dùng.
- Tầng ứng dụng: phối hợp hoạt động của ứng dụng, không chứa logic nghiệp vụ, không lưu giữ trạng thái của các đối tượng nghiệp vụ nhưng nó có thể giữ trạng thái của một tiến trình của ứng dụng.
- Lớp domain: Tầng này chứa thông tin về các lĩnh vực. Đây là trọng tâm của nghiệp vụ phần mềm. Trạng thái của đối tượng nghiệp vụ được giữ tại đây, quản lý vòng đời đối tượng nghiệp vụ và trạng thái của chúng được ủy quyền cho lớp hạ tầng.
- Lớp hạ tầng: lớp này đóng vai trò như một thư viện hỗ trợ cho tất cả các lớp còn lại. Nó cung cấp thông tin liên lạc giữa các lớp, cài đặt cho đối tượng nghiệp vụ, đồng thời chứa các thư viện hỗ trợ cho tầng giao diện người dùng…

2.2.2.4. Mô-đun

Với hệ thống lớn và phức tạp, mô hình thường có xu hướng phình càng ngày càng to. Khi mô hình phình tới một điểm ta khó nắm bắt được tổng thể, việc hiểu quan hệ và tương tác giữa các phần khác nhau trở nên khó khăn. Vì lý do đó, việc tổ chức mô hình thành các mô-đun là cần thiết. Mô- đun được dùng như một phương pháp để tổ chức các khái niệm và tác vụ liên quan nhằm giảm độ phức tạp.

Mô-đun được dùng rộng rãi trong hầu hết các dự án. Sẽ dễ dàng hơn để hình dung mô hình lớn nếu ta nhìn vào những mô-đun chứa trong mô hình đó, sau đó là quan hệ giữa các mô-đun. Khi đã hiểu tương tác giữa các mô-đun, ta có thể bắt đầu tìm hiểu phần chi tiết trong từng mô-đun. Đây là cách đơn giản và hiệu quả để quản lý sự phức tạp.

Với hệ thống lớn và phức tạp, mô hình thường có xu hướng phình càng ngày càng to. Khi mô hình phình tới một điểm ta khó nắm bắt được tổng thể, việc hiểu quan hệ và tương tác giữa các phần khác nhau trở nên khó khăn. Vì lý do đó, việc tổ chức mô hình thành các mô-đun là cần thiết. Mô- đun được dùng như một phương pháp để tổ chức các khái niệm và tác vụ liên quan nhằm giảm độ phức tạp. Mô-đun được dùng rộng rãi trong hầu hết các dự án. Sẽ dễ dàng hơn để hình dung mô hình lớn nếu ta nhìn vào những mô-đun chứa trong mô hình đó, sau đó là quan hệ giữa các mô-đun. Khi đã hiểu tương tác giữa các mô-đun, ta có thể bắt đầu tìm hiểu phần chi tiết trong từng mô-đun. Đây là cách đơn giản và hiệu quả để quản lý sự phức tạp.

2.2.2.5. Aggregate

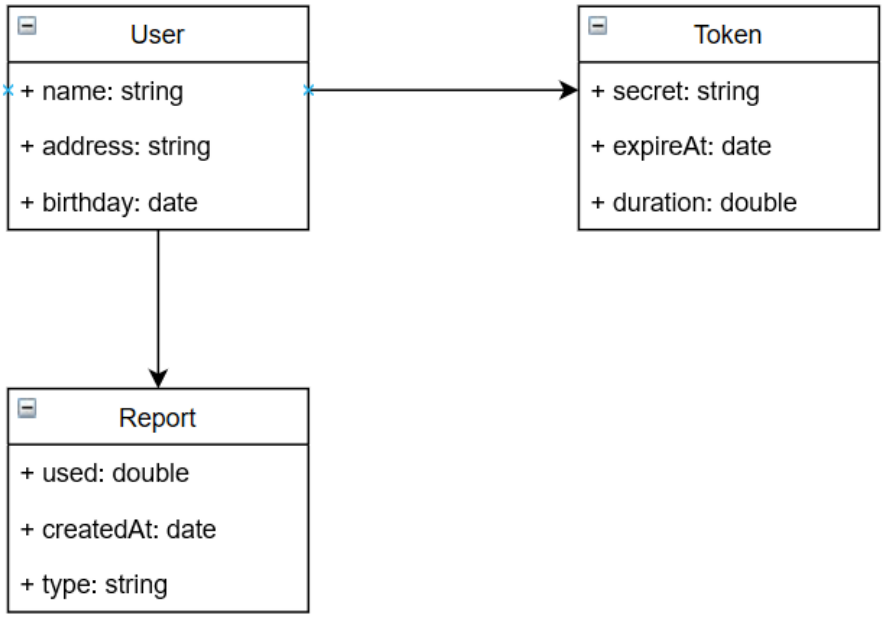
Việc quản lý vòng đời các đối tượng trong domain không hề đơn giản, nếu như làm không đúng sẽ có thể gây ảnh hưởng đến việc mô hình hóa domain. Sau đây chúng ta sẽ đề cập đến mẫu thiết kế thường dùng để hỗ trợ giải quyết vấn đề này. Aggregate là mẫu thiết kế để định nghĩa việc sở hữu đối tượng và phân cách giữa chúng.

Một mô hình có thể được tạo thành từ nhiều đối tượng trong domain. Cho dù có cẩn thận trong việc thiết kế như thế nào thì chúng ta cũng không thể tránh được việc sẽ có nhiều mối quan hệ chằng chịt giữa các đối tượng, tạo thành một lưới các quan hệ. Có thể có

nhiều kiểu quan hệ khác nhau, với mỗi kiểu quan hệ giữa các mô hình cần có một cơ chế phần mềm để thực thi nó. Các mối quan hệ sẽ được thể hiện trong mã nguồn phần mềm, và trong nhiều trường hợp trong cả cơ sở dữ liệu nữa.

Một Aggregate là một nhóm các đối tượng, nhóm này có thể được xem như là một đơn vị thống nhất đối với các thay đổi dữ liệu. Một Aggregate được phân tách với phần còn lại của hệ thống, ngăn cách giữa các đối tượng nội tại và các đối tượng ở ngoài. Mỗi Aggregate có một "*gốc*" (root), đó là một thực thể và cũng là đối tượng duy nhất có thể truy cập từ phía ngoài của Aggregate. Gốc của Aggregate có thể chứa những tham chiếu đến các đối tượng khác trong Aggregate, và những đối tượng trong này có thể chứa tham chiếu đến nhau, nhưng các đối tượng ở ngoài chỉ có thể tham chiếu đến gốc. Nếu như trong Aggregate có những thực thể khác thì định danh của chúng là nội tại, chỉ mang ý nghĩa trong aggregate.

Vậy bằng cách nào mà Aggregate có thể đảm bảo được tính toàn vẹn và các ràng buộc của dữ liệu? Vì các đối tượng khác chỉ có thể tham chiếu đến gốc của Aggregate, chúng không thể thay đổi trực tiếp đến các đối tượng nằm bên trong mà chỉ có thể thay đổi thông qua gốc, hoặc là thay đổi gốc Aggregate trực tiếp. Tất cả các thay đổi của Aggregate sẽ thực hiện thông qua gốc của chúng, và chúng ta có thể quản lý được những thay đổi này, so với khi thiết kế cho phép các đối tượng bên ngoài truy cập trực tiếp vào các đối tượng bên trong thì việc đảm bảo các invariant sẽ đơn giản hơn nhiều khi chúng ta phải thêm các logic vào các đối tượng ở ngoài để thực hiện. Nếu như gốc của Aggregate bị xóa và loại bỏ khỏi bộ nhớ thì những đối tượng khác trong Aggregate cũng sẽ bị xóa, vì không còn đối tượng nào chứa tham chiếu đến chúng.



Hình 2.2.3 Mối quan hệ trong Aggregate có gốc là đối tượng người dùng – *User* cùng các quan hệ với các báo cáo – *Report,* mã truy cập – *Token.*

2.2.3. Duy trì tính toàn vẹn của mô hình

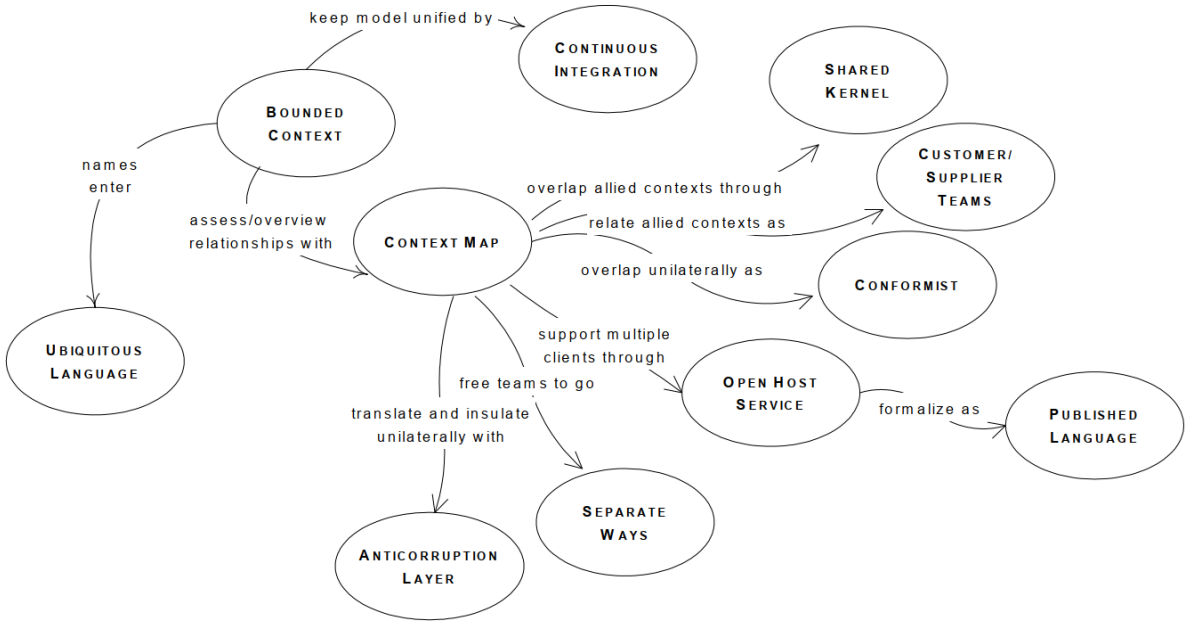
Chương này đề cập tới những dự án lớn, đòi hỏi sự tham gia của nhiều nhóm. Chúng ta gặp nhiều loại thách thức khác nhau khi có nhiều nhóm, được quản lý và phối hợp theo cách khác nhau. Những dự án cho doanh nghiệp thường là những dự án lớn, đòi hỏi nhiều loại công nghệ và nguồn lực. Thiết kế của những dự án ấy vẫn thường dựa trên mô hình domain và chúng ta cần đo đạc thích hợp để đảm bảo sự thành công của dự án.

Khi nhiều nhóm làm cùng một dự án, họ viết mã nguồn song song, mỗi nhóm được chỉ định làm một phần của mô hình. Những phần này không độc lập, nhưng ít liên hệ chéo. Họ đều bắt đầu từ một mô hình lớn và chia nhỏ ra để thực thi. Giả sử rằng một nhóm nào đó đã tạo ra một mô-đun và họ mở cho các nhóm khác dùng. Một lập trình viên từ nhóm khác dùng mô-đun này và phát hiện ra rằng mô-đun nó thiếu một số chức năng anh ta cần. Anh ta thêm những chức năng nó và check-in mã nguồn để mọi người đều có thể dùng. Điều anh ta không ngờ đến là việc anh ta làm thay đổi cả mô hình, và có thể rằng thay đổi này sẽ làm hỏng tính năng của ứng dụng. Điều này rất dễ xảy ra, không ai có

thời gian để hiểu cả mô hình. Mọi người đều có sân chơi riêng của mình và những "*sân riêng*" này không đủ rõ với người khác.

Sẽ là dễ dàng nếu ta bắt đầu từ một mô hình tốt và sau đó biến thành các mô hình không nhất quán. Yêu cầu đầu tiên của mô hình là phải nhất quán, với những điều khoản không bất biến và không có mâu thuẫn. Sự nhất quán nội tại của một mô hình được gọi là *sự thống nhất*. Một dự án doanh nghiệp có thể có một mô hình phủ mọi domain của doanh nghiệp, không có mâu thuẫn và không có điều khoản chồng chéo. Một mô hình doanh nghiệp thống nhất lý tưởng không dễ đạt được và đôi khi không đáng giá để làm thử. Những dự án như thế cần nỗ lực tổng hợp của nhiều nhóm. Các nhóm này cần độ độc lập cao trong quy trình phát triển vì họ không có thời gian gặp nhau thường xuyên để trao đổi về thiết kế. Việc điều phối những nhóm như thế là một nhiệm vụ khó khăn. Họ có thể thuộc về nhiều phòng ban khác nhau với cách quản lý khác nhau. Khi thiết kế của mô hình tiến hóa từng phần một cách độc lập, chúng ta đối mặt với rủi ro mất tính nhất quán của mô hình. Duy trì tính nhất quán của mô hình bằng việc cố gắng giữ một mô hình thống nhất lớn cho toàn dự án doanh nghiệp sẽ không thể hoạt động được. Giải pháp không dễ vì nó đi ngược với tất cả những gì chúng ta đã tìm hiểu tới nay. Thay vì giữ một mô hình lớn nhưng sẽ bị phân nhỏ về sau này, chúng ta nên chia một cách có ý thức mô hình thành nhiều mô hình khác. Những mô hình này sẽ tích hợp và tiến hóa độc lập nếu chúng tuân thủ giao ước của chúng. Mỗi mô hình cần có một ranh giới rõ ràng, và quan hệ giữa các mô hình phải được định nghĩa chính xác.

Chúng ta sẽ trình bày một số kỹ thuật dùng để duy trì tính nhất quán của mô hình. Hình sau đây là những kỹ thuật và quan hệ giữa chúng.



Hình 2.2.4 Một số kỹ thuật duy trì tính nhất quán của mô hình và mối quan hệ giữa chúng.

2.2.3.1. Ngữ cảnh giới hạn (bounded context)

Mỗi mô hình đều có một ngữ cảnh tương ứng với nó. Chúng ta làm việc với một mô hình, ngữ cảnh là ẩn. Chúng ta không cần định nghĩa chúng. Khi chúng ta tạo một chương trình tương tác với phần mềm khác, ví dụ một ứng dụng cũ, hiển nhiên là ứng dụng mới có mô hình và ngữ cảnh riêng của nó và chúng được chia riêng với mô hình và ngữ cảnh cũ. Chúng không thể gộp, trộn lẫn và không bị hiểu lầm. Tuy nhiên, khi chúng ta làm việc với một ứng dụng doanh nghiệp lớn, chúng ta cần định nghĩa ngữ cảnh cho từng mô hình ta tạo ra.

Dự án lớn nào cũng có nhiều mô hình. Tuy vậy, mã nguồn dựa trên các mô hình riêng biệt lại được gộp lại và phần mềm trở nên nhiều lỗi, kém tin tưởng và khó hiểu. Giao tiếp giữa các thành viên trong nhóm trở nên rối. Thường thì việc quyết định mô hình nào áp dụng cho ngữ cảnh nào là không rõ ràng.

Không có một công thức nào để chia mọi mô hình to thành nhỏ thành nhỏ hơn. Hãy thử đặt những thành phần có liên quan vào mô hình theo những khái niệm một cách tự nhiên.

Một mô hình cần đủ nhỏ để nó phù hợp với một nhóm. Sự phối hợp và trao đổi nhóm sẽ thông suốt và hoàn chỉnh, giúp cho lập trình viên làm việc trên cùng một mô hình. Ngữ cảnh của mô hình là tập các điều kiện cần được áp dụng để đảm bảo những điều khoản của mô hình có ý nghĩa cụ thể.

Ý tưởng chính cho việc định nghĩa một mô hình là vẽ ra ranh giới giữa các ngữ cảnh, sau đó cố gắng giữ các mô hình có được sự thống nhất càng cao càng tốt. Việc giữ một mô hình "thuần khiết" khi nó mở rộng ra cả dự án doanh nghiệp là rất khó, nhưng dễ dàng hơn khi chúng ta hạn chế trong một mảng cụ thể. Định nghĩ một cách hiển minh ngữ cảnh trong ngữ cảnh giới hạn. Hãy xác định hiển minh ranh giới giữa các điều khoản của tổ chức nhóm, dùng nó trong một phần cụ thể của mô hình, thể hiện nó một cách vật lý bằng, chẳng hạn như schema của mã nguồn. Duy trì mô hình này tương thích chặt chẽ với các giới hạn nó. Tuy vậy, đừng để mất tập trung hay rối loạn vì những vấn đề bên ngoài.

Một ngữ cảnh giới hạn không phải là một mô-đun. Môt ngữ cảnh giới hạn cung cấp khung logic bên trong của mô hình. Mô-đun tổ chức các thành phần của mô hình, đo đó ngữ cảnh giới hạn chứa mô-đun. Khi nhiều nhóm làm việc trên cùng một mô hình, chúng ta phải cẩn thận sao cho chúng không dẫm chân lên nhau. Chúng ta phải luôn ý thức tới sự thay đổi có thể ảnh hưởng đến chức năng. Khi dùng nhiều mô hình, mọi người phải làm việc tự do trên mảng của họ. Chúng ta đều biết hạn chế trong mô hình của chúng ta và không bước ra ngoài giới hạn. Chúng ta phải đảm bảo giữ được mô hình tinh khiết, nhất quán và thống nhất. Mỗi mô hình cần hỗ trợ việc tái cấu trúc dễ hơn, không gây ảnh hưởng tới mô hình khác. Thiết kế có thể được làm mịn và chắt lọc để đạt được sự tinh khiết tối đa.

Việc có nhiều mô hình có cái giá phải trả. Chúng ta cần định nghĩa ranh giới và quan hệ giữa các mô hình khác nhau. Điều này đòi hỏi nhiều công sức và có thể cần sự "phiên dịch" giữa các mô hình khác nhau. Chúng ta không thể chuyển đối tượng bất kỳ giữa các mô hình khác nhau và chúng ta không thể gọi tự do giữa các mô hình như là không có ranh giới nào. Đây không phải là một công việc quá khó và lợi ích nó đem lại đáng được

đầu tư.

Ví dụ chúng ta muốn tạo ra một phần mềm thương mại điện tử để bán phần mềm trên Internet. Phần mềm này cho phép khách hàng đăng ký, và chúng ta thu thập dữ liệu cá nhân, bao gồm số thẻ tín dụng. Dữ liệu được lưu trong cơ sở dữ liệu quan hệ. Khách hàng có thể đăng nhập, duyệt trang để tìm hàng và đặt hàng. Chương trình cần công bố sự kiện khi có đơn đặt hàng vì ai đó cần phải gửi thư có hạng mục đã được yêu cầu. Chúng ta cần xây dựng giao diện báo cáo dùng cho việc tạo báo cáo để có thể theo dõi trạng thái hàng còn, khách hàng muốn mua gì, họ không thích gì... Ban đầu, chúng ta bắt đầu bằng một mô hình phủ cả domain thương mại điện tử. Chúng ta muốn làm thế vì sau chúng, chúng ta cũng sẽ cần làm một chương trình lớn. Tuy nhiên, chúng ta xem xét công việc này cẩn thận và phát hiện ra rằng chương trình e-shop thực ra không liên quan đến chức năng báo cáo. Chúng quan tâm đến cái khác, vận hành theo một cách khác và thậm chí, không cần dùng công nghệ khác. Điểm chung duy nhất ở đây là khách hàng và dữ liệu hàng hóa được lưu trong cơ sở dữ liệu mà cả hai đều truy cập vào.

Cách tiếp cận được khuyến nghị ở đây là tạo mô hình riêng cho mỗi domain, một domain cho e-commerce, một cho phần báo cáo. Chúng có thể tiến hóa tự do không cần quan tâm tới domain khác và có thể trở thành những chương trình độc lập. Có trường hợp là chương trình báo cáo cần một số dữ liệu đặc thù mà chương trình e-commerce lưu trong cơ sở dữ liệu dù cả hai phát triển độc lập.

Một hệ thống thông điệp cần để báo cáo cho người quản lý kho về đơn đặt hàng để họ có thể gửi mail về hàng đã được mua. Người gửi mail sẽ dùng chương trình và cung cấp cho họ thông tin chi tiết về hàng đã mua, số lượng, địa chỉ người mua cũng như yêu cầu giao hàng. Việc có mô hình e-shop phủ cả hai domain là không cần thiết. Sẽ đơn giản hơn nhiều cho chương trình e-shop chỉ gửi đối tượng giá trị chứa thông tin mua tới kho bằng phương thức thông điệp phi đồng bộ. Rõ ràng là có hai mô hình có thể phát triển độc lập và chúng ta chỉ cần đảm bảo giao diện giữa chúng hoạt động tốt.

2.2.3.2. Tích hợp liên tục

Khi một ngữ cảnh giới hạn đã được định nghĩa, chúng ta cần đảm bảo rằng nó luôn mới và hoạt động như kỳ vọng. Khi nhiều người cùng làm việc trên cùng một ngữ cảnh giới hạn, mô hình có khuynh hướng bị phân mảnh. Nhóm càng lớn, vấn đề càng lớn, nhóm nhỏ từ ba đến bốn người cũng có thể gặp vấn đề nghiêm trọng.

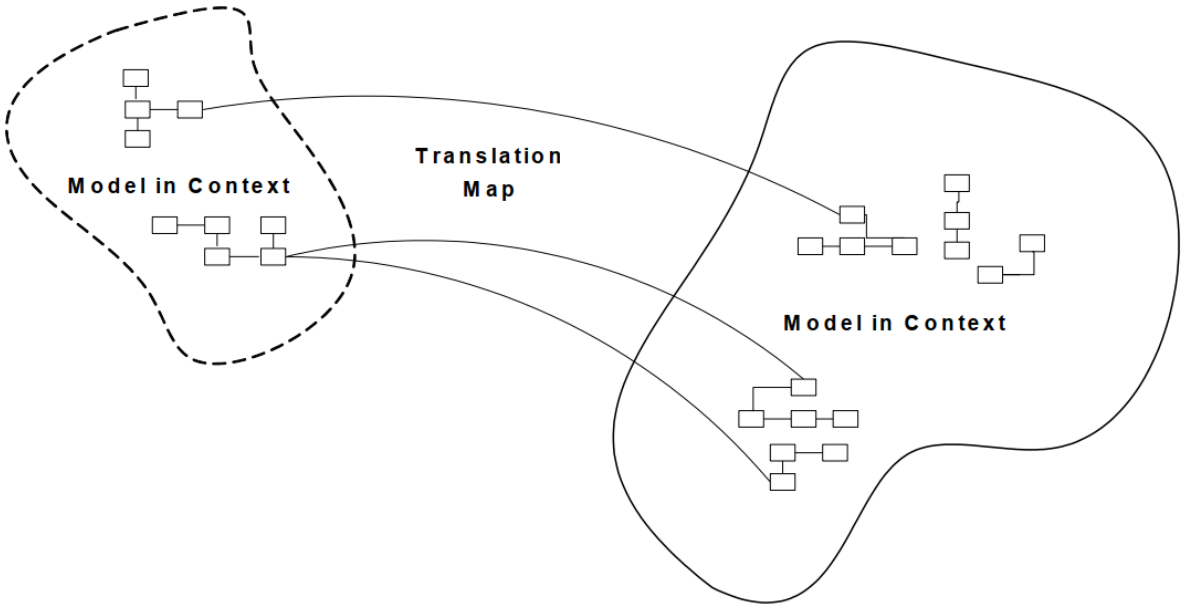
Tuy nhiên, nếu ta chia nhỏ hệ thống thành những ngữ cảnh rất-rất nhỏ thì thực ra, ta đánh mất mức độ giá trị về tích hợp và tính tương liên. Ngay cả khi nhóm làm việc cùng trên một ngữ cảnh giới hạn thì vẫn có thể có lỗi. Chúng ta cần giao tiếp trong nội bộ nhóm để đảm bảo rằng mọi người đều hiểu vai trò của từng phần tử trong mô hình. Nếu có người không hiểu quan hệ giữa các đối tượng, họ có thể sửa mã nguồn làm nó mâu thuẫn với mô hình ban đầu. Lỗi dễ xảy ra và chúng ta không thể đảm bảo 100% tập trung vào sự tinh khiết của mô hình. Một thành viên của nhóm có thể thêm mã nguồn trùng với mã đã có mà không biết, hay có thể lặp lại mã nguồn mà không hề thay đổi mã hiện tại vì sợ rằng có thể làm hỏng chức năng đang đang có.

Không thể định nghĩa đầy đủ một mô hình ngay từ đầu. Mô hình được tạo ra, tiến hóa liên tục dựa trên thực tế của domain và phản hồi từ quy trình phát triển. Điều này nghĩa là khái niệm mới có thể xuất hiện trong mô hình, phần tử mới được thêm vào mã nguồn. Mọi nhu cầu kiểu này được tích hợp vào một mô hình thống nhất, được thực thi tương ứng trong mã nguồn. Đó là lý do tại sao tích hợp liên tục là quy trình cần thiết trong ngữ cảnh giới hạn. Chúng ta cần quy trình tích hợp để đảm bảo rằng mọi phần tử được thêm vào một cách hài hòa trong toàn bộ phần còn lại của mô hình và được thực thi đúng trong mã nguồn. Chúng ta cần có một thủ tục khi tích hợp mã nguồn, tích hợp mã nguồn càng sớm càng tốt. Với một nhóm nhỏ, nên tích hợp hàng ngày, chúng ta cũng cần xây dựng quy trình. Mã nguồn đã được merge cần được build tự động và được kiểm thử. Một yêu cầu cần thiết khác là kiểm thử tự động. Nếu nhóm có công cụ kiểm thử và tạo ra một bộ kiểm thử thì việc kiểm thử có thể chạy mỗi khi build để để đưa ra mọi cảnh báo khi có lỗi. Sửa mã nguồn để sửa lỗi đã được báo cáo là dễ vì chúng ta phát hiện sớm. Sau đó chúng ta tiếp tục quy trình với các bước: merge, build, kiểm thử tiếp.

Tích hợp liên tục dựa trên khái niệm tích hợp của mô hình, tìm cách thực thi những chỗ được kiểm thử. Bất kỳ sự không nhất quán nào trong mô hình có thể được phát hiện trong quá trình thực thi. Tích hợp liên tục áp dụng vào một ngữ cảnh giới hạn và nó được dùng để đối phó với quan hệ giữa các ngữ cảnh xung quanh.

2.2.3.3. Ngữ cảnh ánh xa

Một phần mềm doanh nghiệp có nhiều mô hình, mỗi mô hình có ngữ cảnh giới hạn riêng. Bạn nên dùng ngữ cảnh như là cơ sở để tổ chức nhóm. Người trong cùng nhóm sẽ giao tiếp dễ hơn, tích hợp công việc và thực thi công việc dễ hơn. Khi mọi nhóm làm việc với mô hình của riêng họ, mọi người cần hiểu bức tranh tổng thể. Một ánh xạ ngữ cảnh mô tả khái quát các ngữ cảnh giới hạn và quan hệ giữa chúng. Một ánh xạ ngữ cảnh có thể là một giản đồ như hình dưới đây, hoặc là tài liệu viết. Mức độ chi tiết của nó tùy trường hợp. Điều quan trọng là mọi người làm việc trên cùng một dự án phải hiểu và cùng chia sẻ bức tranh này.



Hình 2.2.5 Ngữ cảnh ánh xạ.

Sẽ là không đủ nếu phải tách riêng mô hình thống nhất. Chúng được tích hợp lại vì chức năng của mỗi mô hình chỉ là một phần của hệ thống. Cuối cùng mỗi mảnh ghép đó gộp

lại và cả hệ thống phải chạy đúng. Nếu ngữ cảnh không được định nghĩa rõ ràng, có thể là chúng sẽ dẫm chân lên nhau. Nếu quan hệ giữa các ngữ cảnh không được chỉ ra, hệ thống có thể sẽ không chạy khi tích hợp.

Khi việc tích hợp chức năng bị hạn chế, chi phí cho tích hợp liên tục có thể tăng cao. Điều này đặc biệt đúng khi nhóm không đủ kỹ năng hay yếu tố chính trị của tổ chức để duy trì tích hợp liên tục, hoặc khi cỡ của một nhóm quá lớn và không thể hành động được. Do đó việc phân chia ngữ cảnh giới hạn lúc này mới tách và phân chia thành nhiều nhóm.

Nhiều nhóm không gắn kết cùng làm trên một phần mềm có liên kết chặt chẽ có thể chạy độc lập, nhưng kết quả họ làm ra có thể không ăn khớp. Họ sẽ cần nhiều thời gian để dịch lớp và chỉnh lại hơn là dành thời gian cho việc tích hợp liên tục ngay từ ban đầu, tức là trong lúc họ làm việc ấy, họ tốn công vào việc người khác đã làm và đánh vật với lợi ích của ngôn ngữ chung. Do đó, hãy chỉ định một vài tập con của mô hình domain mà các nhóm đồng ý chia sẻ.

Tất nhiên, việc này bao gồm, đi cùng với tập con của mô hình, là tập con của mã nguồn và thiết kế cơ sở dữ liệu đi cùng với phần tương ứng của mô hình. Rõ ràng, việc làm này chia sẻ những cái có trạng thái đặc biệt và không thể được thay đổi nếu không có sự tham vấn của nhóm khác.

Hãy tích hợp một hệ thống chức năng thường xuyên, nhưng ít thường xuyên hơn nhịp của tích hợp liên tục trong nội bộ nhóm. Trong những phiên tích hợp này, hãy chạy kiểm thử cho cả hai nhóm. Mục đích của nhân chung là giảm sự lặp lại, nhưng vẫn giữ riêng hai ngữ cảnh riêng biệt. Phát triển trên một nhân chung đòi hỏi độ chú ý cao hơn. Cả hai nhóm có thể thay đổi mã nguồn của nhân, và họ phải tích hợp các thay đổi. Nếu các nhóm chia riêng bản sao chép của mã nguồn nhân, họ phải tích hợp mã nguồn lại càng sớm càng tốt, ít nhất là một tuần một lần. Một bộ kiểm thử là cần thiết để mọi thay đổi tới nhân cần được báo cho các nhóm khác để họ biết khi có chức năng mới.

2.3. LÝ THUYẾT NỀN TẢNG CỦA MẪU COMMAND QUERY RESPONSIBILITY SEGREGATION – CQRS

Nhiều ứng dụng doanh nghiệp sử dụng một hệ quản trị cơ sở dữ liệu quan hệ (Relational Database Management System – RDBMS) làm hệ thống lưu trữ dữ liệu và một cơ sở dữ liệu hỗ trợ truy vấn, tìm kiếm dữ liệu như Elasticsearch, Solr. Một số ứng dụng đảm bảo hai cơ sở dữ liệu này được đồng bộ bằng cách ghi dữ liệu vào cả hai cùng một lúc. Một số khác thì định kì sao chép dữ liệu từ RDBMS sang cơ sở dữ liệu văn bản (text database). Các ứng dụng với kiến trúc này tận dụng ưu điểm của nhiều cơ sở dữ liệu: các tính chất khi thực hiện các transactions (transaction là một chuỗi các hành động được thực thi một cách tuần tự và độc lập với nhau trong cơ sở dữ liệu) trong RDBMS và khả năng truy vấn trong cơ sở dữ liệu văn bản.

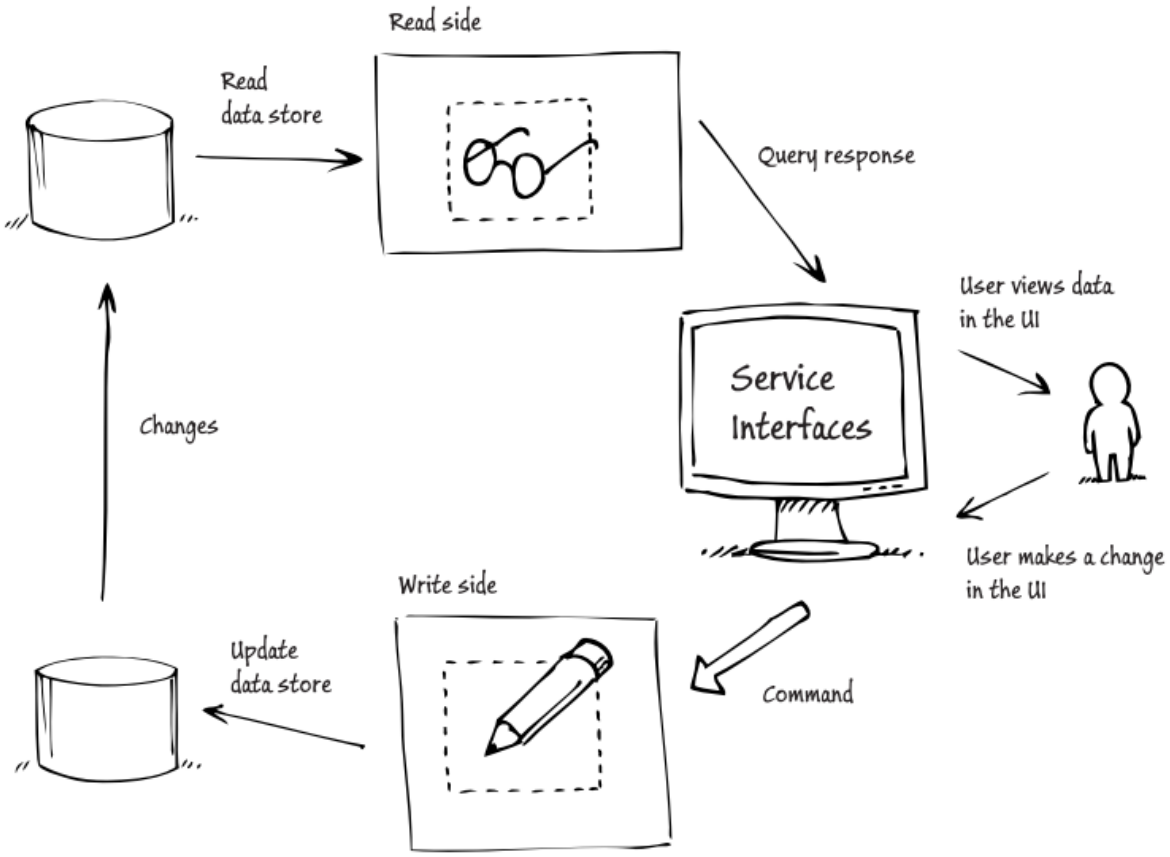
Mẫu Command Query Responsibility Segregation (CQRS) là một khái quát của dạng kiến trúc này. Để hiểu rõ hơn về CQRS, cùng tìm hiểu về định nghĩa, cách hoạt động cũng như tầm ảnh hưởng đối với kiến trúc ứng dụng khi áp dụng nó qua các phần sau.

2.3.1. Định nghĩa

Trong quyển sách "Object Oriented Software Construction", Betrand Meyer đã giới thiệu một thuật ngữ "Command Query Separation" để mô tả một nguyên tắc rằng các phương thức của đối tượng nên là những lệnh (commands) hoặc những câu truy vấn (queries). Một query có nhiệm vụ trả về dữ liệu và không thay đổi trạng thái của dối tượng. Một command thì ngược lại, nó thay đổi trạng thái của đối tượng nhưng không trả về dữ liệu nào. Nguyên tắc này giúp ta hiểu rõ hơn cái gì có thể và không thể thay đổi trạng thái của các đối tượng trong hệ thống. CQRS dựa trên nguyên tắc này và mở rộng thêm để định nghĩa nên một mẫu (pattern) đơn giản.

Mẫu CQRS được giới thiệu lần đầu bởi Greg Young và được định nghĩa như sau: "CQRS đơn giản là việc tạo ra hai đối tượng trong khi trước đó chỉ có một. Sự phân tách này xảy ra dựa trên việc định nghĩa các phương thức là một command hay một query".

Đúng như tên gọi của nó, CQRS giao trách nhiệm sửa đổi và truy vấn dữ liệu hệ thống cho những đối tượng khác nhau: mô hình đọc và mô hình ghi. Hình 2.3.1 mô tả một ứng dụng điển hình sử dụng mẫu CQRS cho một phần của hệ thống.



Hình 2.3.1 Một cách triển khai kiến trúc khi áp dụng mẫu CQRS

Hình 2.3.1 cho thấy cách chia một phần của hệ thống thành hai phía, phía ghi (write side) và phía đọc (read side) dữ liệu. Những đối tượng bên phía đọc chỉ chứa những queries, còn những đối tượng bên phía ghi chỉ chứa những commands.

Có nhiều lý do dẫn đến sự tách biệt này, bao gồm:

- Trong nhiều hệ thống, có một sự mất cân đối khá lớn giữa số lần đọc và số lần ghi dữ liệu. Thời gian một hệ thống xử lí hàng ngàn yêu cầu truy vấn dữ liệu có thể tương ứng với thời gian hệ thống đó xử lí một yêu cầu ghi dữ liệu. Sự phân tách thành hai phía cho phép tối ưu hoá hai hành động đọc và ghi dữ liệu một

cách độc lập.

- Thông thường, các commands chứa logic nghiệp vụ phức tạp để đảm bảo rằng hệ thống ghi dữ liệu chính xác và nhất quán vào kho lưu trữ dữ liệu (data store). Thao tác đọc thường đơn giản hơn rất nhiều so với thao tác ghi. Sự phân tách thành hai phía tạo ra những mô hình đơn giản, dễ bảo trì và linh hoạt hơn.
- Sự tách biệt cũng có thể xảy ra ở cấp lưu trữ dữ liệu. Phía ghi dữ liệu có thể sử dụng một lược đồ cơ sở dữ liệu được chuẩn hoá gần đạt đến dạng chuẩn 3 (third normal form – 3NF) và được tối ưu hoá phục vụ cho việc sửa đổi dữ liệu. Trong khi phía đọc dữ liệu có thể sử dụng một cơ sở dữ liệu không được chuẩn hoá (denormalized) để tối ưu các thao tác truy vấn. Mặc dù hình 2.3.1 sử dụng hai kho lưu trữ dữ liệu, việc áp dụng mẫu CQRS không bắt buộc phải chia tách kho lưu trữ dữ liệu hay phải sử dụng một công nghệ lưu trữ cụ thể nào như cơ sở dữ liệu quan hệ hay phi quan hệ,… Mẫu CQRS chỉ tạo điều kiện cho việc phân tách kho lưu trữ dữ liệu và cho phép lựa chọn tuỳ ý cơ chế lưu trữ dữ liệu.

Điều quan trọng và thú vị về mẫu CQRS nằm ở việc tại sao lại sử dụng nó, khi nào nên sử dụng và sử dụng như thế nào khi xây dựng hệ thống. Những phần tiếp theo sẽ lần lượt giải đáp các câu hỏi này.

2.3.2. CQRS và DDD

Nhiều ý kiến cho rằng mẫu CQRS ra đời để giải quyết một số vấn đề gặp phải khi áp dụng DDD tiếp cận các vấn đề trong thế giới thực. Do vậy, nếu sử dụng DDD, có thể thấy rằng mẫu CQRS rất phù hợp cho một vài ngữ cảnh giới hạn được xác định bên trong hệ thống. Một số chuyên gia cho rằng việc áp dụng DDD là điều kiện tiên quyết để triển khai mẫu CQRS. Tuy nhiên, Greg Young cho rằng việc áp dụng DDD không phải là điều kiện tiên quyết khi triển khai mẫu CQRS, nhưng trong thực tế, chúng thường đi đôi với nhau.

Có hai lĩnh vực quan trọng cần được nhắc đến:

- Khái niệm về mô hình: Trong hình 2.3.1 việc triển khai mô hình tồn tại bên phía

ghi dữ liệu. Nó gói gọn những logic nghiệp vụ phức tạp diễn ra trong một phần của hệ thống. Phía đọc dữ liệu là một góc nhìn các trạng thái hệ thống. Góc nhìn này đơn giản, chỉ được phép đọc và truy cập thông qua các queries.

- Cách DDD phân chia các hệ thống lớn, phức tạp thành nhiều đơn vị dễ quản lí hơn hay còn gọi là các ngữ cảnh giới hạn. Một ngữ cảnh giới hạn định nghĩa một ngữ cảnh cho một mô hình.

Khi nói rằng nên áp dụng mẫu CQRS cho một phần của hệ thống, nó có nghĩa rằng nên áp dụng mẫu CQRS trong một ngữ cảnh giới hạn định nghĩa bởi DDD. Nên xác định rõ những phần khác nhau trong hệ thống mà có thể thiết kế và triển khai độc lập với nhau và chỉ nên áp dụng mẫu CQRS vào những phần có thể đem lại lợi ích nghiệp vụ rõ ràng khi áp dụng nó.

2.3.3. Commands, events và các thông điệp (messages)

DDD là một phương pháp phân tích và thiết kế khuyến khích sử dụng các mô hình và một ngôn ngữ phổ biến để thu hẹp khoảng cách giữa doanh nghiệp và đội ngũ phát triển bằng cách bồi dưỡng sự hiểu biết thông thường về một lĩnh vực. Do đó, DDD được định hướng để phân tích hành vi thay vì chỉ phân tích dữ liệu trong lĩnh vực nghiệp vụ, tập trung mô hình hoá và triển khai cài đặt hành vi trong phần mềm. Để triển khai mô hình nghiệp vụ bằng những dòng mã một cách tự nhiên, cần sử dụng commands và events.

DDD không phải là cách tiếp cận duy nhất sử dụng commands và events để triển khai các nhiệm vụ và hành vi được xác định trong mô hình nghiệp vụ. Tuy nhiên, nhiều người ủng hộ mẫu CQRS cũng bị tác động bởi DDD nên mỗi khi có một cuộc thảo luận về mẫu CQRS, thuật ngữ DDD cũng thường được nhắc đến.

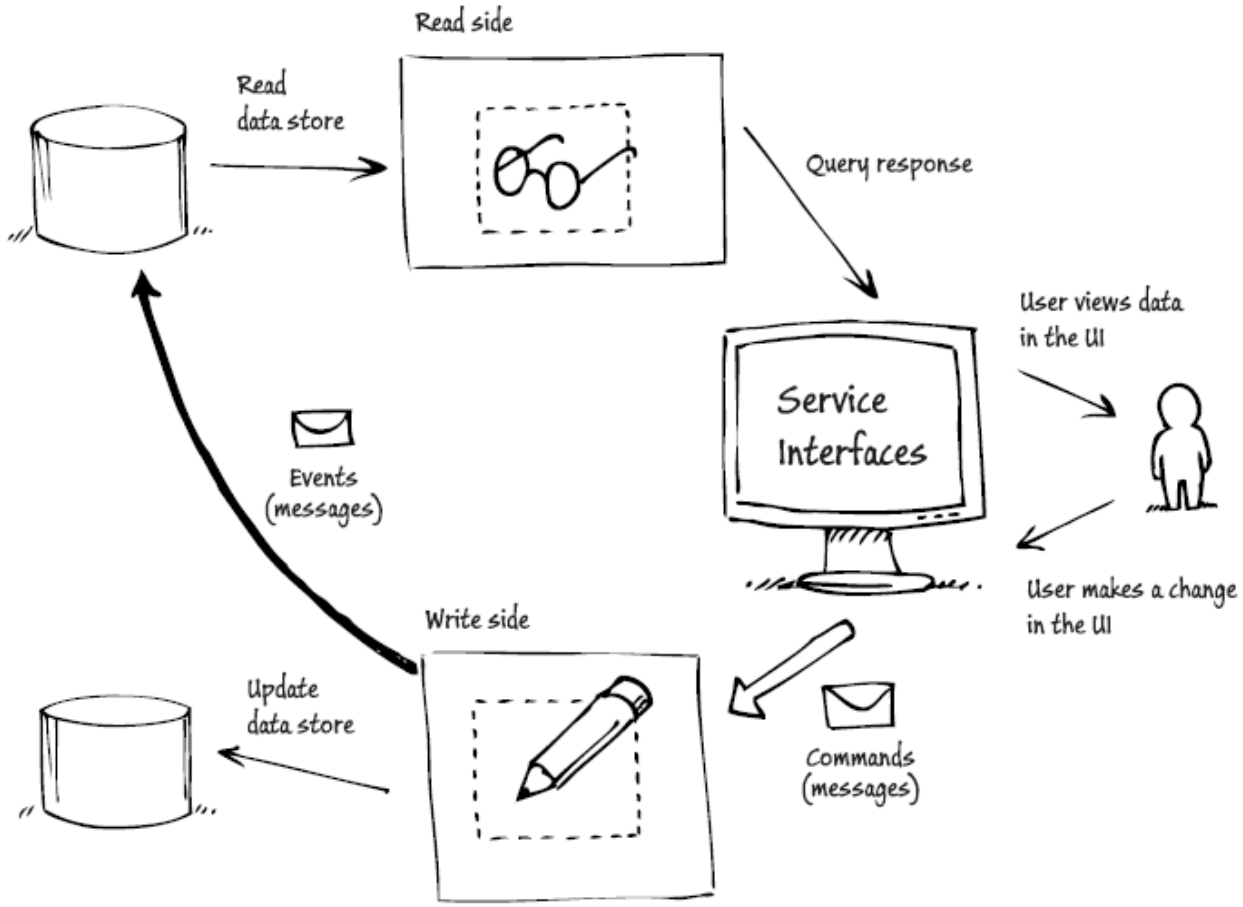
Commands là những lệnh yêu cầu hệ thống thực hiện một tác vụ hay một hành động. Ví dụ, “đặt hai chỗ ngồi tại hội nghị X” hay “phân bổ người phỏng vấn Y vào phòng Z”. Commands thường được xử lí chỉ một lần bởi một bên nhận được nó.

Events là những thông báo mô tả điều đã xảy ra đến các bên quan tâm khác. Ví dụ, “đơn

hàng đã được giao đến địa chỉ A" hay "mười chỗ ngồi đã được đặt tại hội nghị X". Events có thể được xử lí nhiều lần bởi nhiều bên nhận được nó.

Commands và events là những loại thông điệp dùng để trao đổi dữ liệu giữa các đối tượng. Trong DDD, những thông điệp này đại diện cho những hành vi nghiệp vụ. Do đó, chúng giúp hệ thống nắm bắt được hành vi nghiệp vụ ẩn sau các thông điệp.

Một cách triển khai mẫu CQRS có thể sử dụng nhiều kho lưu trữ dữ liệu riêng biệt cho phía đọc và phía ghi dữ liệu. Mỗi kho lưu trữ dữ liệu được tối ưu hoá phục vụ các trường hợp thao tác với dữ liệu mà nó hỗ trợ. Events làm cơ sở cho cơ chế đồng bộ các thay đổi tại phía ghi (kết quả của việc xử lí các commands) với phía đọc. Nếu phía ghi đưa ra một event mỗi khi trạng thái hệ thống có sự thay đổi, phía đọc sẽ phản hồi đối với event đó và cập nhật dữ liệu được dùng để truy vấn. Hình 2.3.2 mô tả cách sử dụng commands và events nếu triển khai mẫu CQRS.



Hình 2.3.2 Commands và events trong mẫu CQRS

2.3.4. Tại sao phải sử dụng mẫu CQRS?

Một trong những lợi ích của việc phân chia lĩnh vực nghiệp vụ thành nhiều ngữ cảnh giới hạn trong DDD là có thể xác định và tập trung vào những phần (ngữ cảnh giới hạn) phức tạp hơn, những phần bị ảnh hưởng bởi những qui tắc nghiệp vụ luôn bị thay đổi hay phải triển khai một chức năng quan trọng, đặc trưng trong hệ thống. Nên xem xét áp dụng mẫu CQRS trong một ngữ cảnh giới hạn cụ thể khi và chỉ khi nó đem lại lợi ích nghiệp vụ rõ ràng. Những lợi ích nghiệp vụ phổ biến có thể đạt được khi áp dụng mẫu CQRS là tăng cường khả năng mở rộng, đơn giản hoá khía cạnh phức tạp trong nghiệp vụ, tăng tính linh hoạt trong các giải pháp và thích ứng tốt hơn đối với sự thay đổi các yêu cầu nghiệp vụ.

2.3.4.1. Khả năng mở rộng

Trong nhiều hệ thống lớn, lượng yêu cầu đọc dữ liệu lớn hơn rất nhiều so với lượng yêu cầu ghi, do đó yêu cầu mở rộng sẽ khác nhau ở hai phía. Bằng cách phân tách phía đọc và phía ghi thành hai mô hình riêng biệt trong ngữ cảnh giới hạn, giờ đây có thể mở rộng mỗi phía một cách độc lập.

2.3.4.2. Đơn giản hoá sự phức tạp

Tại nhiều khía cạnh phức tạp trong nghiệp vụ, thiết kế và triển khai các đối tượng chịu trách nhiệm cho cả hai phần đọc và ghi dữ liệu có thể làm tăng độ phức tạp sẵn có. Trong nhiều trường hợp, logic xử lí nghiệp vụ phức tạp chỉ nên được áp dụng khi hệ thống phải xử lí các cập nhật và thao tác với dữ liệu trong cơ sở dữ liệu. Logic đọc dữ liệu thường đơn giản hơn nhiều. Khi logic nghiệp vụ và logic đọc kết hợp với nhau trong cùng một mô hình, việc giải quyết các vấn đề khó như nhiều người dùng, dữ liệu có thể chia sẻ, hiệu suất, tính nhất quán, dữ liệu cũ (stale data) trở nên khó khăn hơn nhiều. Việc tách logic nghiệp vụ và logic đọc dữ liệu thành những mô hình riêng biệt giúp dễ dàng phân chia và giải quyết những vấn đề phức tạp này.

Một lợi ích tiềm năng khác của việc đơn giản hoá ngữ cảnh giới hạn bằng cách tách biệt logic đọc và logic nghiệp vụ là nó cho phép kiểm thử một cách dễ dàng hơn.

2.3.4.3. Tính linh hoạt

Tính linh hoạt của một giải pháp áp dụng mẫu CQRS xuất phát từ việc phân tách thành những mô hình phía đọc và phía ghi. Việc thay đổi ở phía đọc giờ đây trở nên dễ dàng hơn nhiều, ví dụ như việc thêm một câu truy vấn mới hỗ trợ hiển thị dữ liệu cho màn hình thống kê trên giao diện người dùng, không cần phải lo lắng nó sẽ ảnh hưởng đến hành vi xử lí bên logic nghiệp vụ. Tại phía ghi, có một mô hình chỉ quan tâm đến phần logic xử lí cốt lõi trong nghiệp vụ đồng nghĩa với việc xử lí một mô hình đơn giản hơn là một mô hình bao gồm cả logic đọc dữ liệu.

Về lâu dài, một mô hình tốt, hữu dụng, mô tả chính xác logic xử lí nghiệp vụ cốt lõi sẽ trở thành một tài sản có giá trị. Nó cho phép bạn trở nên nhanh nhẹn hơn khi đối mặt với một môi trường nghiệp vụ thay đổi và áp lực cạnh tranh đối với tổ chức của bạn.

Tính linh hoạt và nhanh nhẹn này liên quan đến khái niệm tích hợp liên tục trong DDD. Theo Eric Evans, tích hợp liên tục có nghĩa là mọi công việc trong ngữ cảnh đang được đồng nhất và việc này được thực hiện thường xuyên đủ để khi có sự cố xảy ra, chúng được phát hiện và sửa chữa nhanh chóng.

Trong một vài trường hợp, có thể có những đội ngũ phát triển khác nhau làm việc bên phía đọc và phía ghi dữ liệu, mặc dù trong thực tế, điều này có thể phụ thuộc vào ngữ cảnh giới hạn cụ thể có mức độ lớn như thế nào.

2.3.4.4. Tập trung vào nghiệp vụ

Việc áp dụng mẫu CQRS giúp tập trung vào nghiệp vụ và xây dựng giao diện người dùng theo hướng nhiệm vụ. Công nghệ có xu hướng định hình giải pháp. Kết quả của việc tách biệt thành hai phía đọc và ghi dữ liệu là một giải pháp thích ứng tốt đối với sự thay đổi yêu cầu nghiệp vụ. Về lâu dài, giải pháp này giúp tiết kiệm chi phí phát triển và

bảo trì.

2.3.5. Khi nào nên áp dụng mẫu CQRS?

Mặc dù đã có những lý do giúp đưa ra quyết định có nên áp dụng mẫu CQRS vào một số ngữ cảnh giới hạn trong hệ thống hay không, vẫn cần những quy tắc giúp xác định ngữ cảnh giới hạn nào nên áp dụng CQRS để đạt được lợi ích nhất định.

Nhìn chung, những ngữ cảnh giới hạn có tính tương tác, phức tạp, gồm những qui tắc nghiệp vụ luôn bị thay đổi và phải triển khai một chức năng quan trọng, đặc trưng trong hệ thống khi áp dụng CQRS sẽ mang lại giá trị cao nhất. Phân tích các yêu cầu nghiệp vụ, xây dựng một mô hình hữu dụng, thể hiện nó bằng các dòng mã và triển khai nó bằng việc áp dụng mẫu CQRS cho một ngữ cảnh giới hạn đều mất nhiều thời gian và chi phí. Nếu không kỳ vọng những lợi ích như khả năng thích ứng và tính linh hoạt trong hệ thống tăng hay chi phí bảo trì giảm thì không đáng để đầu tư thời gian và chi phí như vậy.

2.3.5.1. Những nghiệp vụ có tính tương tác

Theo Udi Dahan, trong một nghiệp vụ mang tính tương tác, một thuộc tính vốn có của nghiệp vụ là có nhiều hoạt động diễn ra song song trên cùng một bộ dữ liệu. Một hệ thống đặt chỗ cho một buổi hoà nhạc là một ví dụ điển hình cho một nghiệp vụ mang tính tương tác, mọi người đều muốn đặt được chỗ ngồi tốt nhất. Cả Udi Dahan và Greg Young đều xác định sự tương tác là một đặc điểm trong một ngữ cảnh giới hạn mà có thể cho thấy các lợi ích một cách rõ ràng nhất khi áp dụng mẫu CQRS.

Mẫu CQRS đặc biệt hữu ích khi sự tương tác bao gồm các quyết định phức tạp về kết quả sẽ như thế nào khi có nhiều hoạt động diễn ra trên cùng tập dữ liệu. Các phần mang tính tương tác như vậy trong hệ thống thường là những ngữ cảnh giới hạn phức tạp nhất. Tuy nhiên, đặc điểm này chỉ là một sự gợi ý. Không phải tất cả những nghiệp vụ có tính tương tác đều đạt được lợi ích khi sử dụng mẫu CQRS.

2.3.5.2. Dữ liệu cũ

Trong một môi trường có tính tương tác, nơi mà nhiều người dùng sẽ thao tác đồng thời trên cùng một tập dữ liệu, vấn đề về dữ liệu cũ sẽ xuất hiện. Nếu một người dùng đang xem một phần dữ liệu trong khi có một người dùng khác thay đổi nó, phần dữ liệu mà người dùng đang xem trở thành dữ liệu cũ.

Dù xây dựng theo kiến trúc nào, vấn đề này cũng cần được giải quyết. Mẫu CQRS giúp giải quyết vấn đề này một cách rõ ràng ở cấp kiến trúc. Những thay đổi về dữ liệu xảy ra bên phía ghi, người dùng xem dữ liệu bằng cách truy vấn bên phía đọc. Cơ chế được chọn dùng để đẩy các thay đổi dữ liệu từ phía ghi sang phía đọc cũng là cơ chế dùng để kiểm soát khi nào dữ liệu phía đọc trở nên cũ và thời gian duy trì như vậy là bao lâu.

2.4. LÝ THUYẾT NỀN TẢNG MÔ HÌNH EVENT SOURCING – ES

Mô hình Event Sourcing (ES) và mô hình CQRS là những mô hình thường được kết hợp cùng nhau để phát huy tối ưu hiệu năng. Mặc dù không có bất cứ ràng buộc nào về kiến trúc, nhưng những yếu tố bổ sung cho nhau đã làm cho hầu hết các kiến trúc phần mềm có ES đều được xây dựng và phát triển với sự tham gia của CQRS. Phần này sẽ trình bày lý thuyết nền tảng của mô hình ES phục vụ cho kiến trúc hệ thống và mối quan hệ mật thiết với mô hình CQRS.

2.4.1. Định nghĩa

2.4.1.1. Events là gì?

Events là những sự kiện xảy ra trong quá khứ. Ví dụ, ta có các events “chỗ ngồi đã được đặt”, “đơn hàng đã được tạo”, “tiền đã được hoàn trả”,…
Events là dữ liệu bất biến. Vì events là những sự kiện đã xảy ra nên không thể thay đổi hay hoàn tác chúng. Tuy nhiên, các sự kiện diễn ra sau đó có thể thay đổi hay phủ nhận các sự kiện trước đó. Ví dụ, ta có event “đơn hàng đã bị huỷ” là một event thay đổi kết quả của event trước nó “đơn hàng đã được tạo”.

Events là những tin nhắn một chiều (one-way messages). Chỉ tồn tại một nguồn (publisher) để công bố (publish) các events và có thể có một hoặc nhiều nguồn nhận (subscribers) nhận được các events này.
Đặc biệt, events bao gồm những thông số cung cấp thông tin về events. Ví dụ, event "Chỗ ngồi có mã số J26 đã được đặt bởi Linh" bao gồm 2 thông số "J26" và "Linh" là những thông tin quan trọng xác định chỗ ngồi nào được đặt bởi ai.

2.4.1.2. Event Sourcing là gì?

Mô hình Event Sourcing là phương pháp duy trì trạng thái ứng dụng bằng cách lưu trữ lịch sử để xác định trạng thái hiện tại của ứng dụng. Ví dụ, một hệ thống quản lý hội nghị cần theo dõi số lượng chỗ ngồi đã được đặt cho một hội nghị để có thể kiểm tra số ghế còn trống khi có yêu cầu đặt chỗ. Hệ thống sẽ lưu trữ tổng số lượng đặt chỗ cho một hội nghị theo 2 cách:

- Lưu trữ tổng số lượng đặt chỗ cho một hội nghị và điều chỉnh số lượng này mỗi khi có yêu cầu đặt hoặc huỷ chỗ.
- Lưu trữ tất cả events đặt chỗ và hủy chỗ cho mỗi hội nghị. Sau đó, tính toán số lượng đặt chỗ hiện tại bằng cách tái diễn (replay) các events liên quan đến hội nghị có nhu cầu kiểm tra tổng số lượng đặt chỗ hiện tại. Cách này chính là việc xây dựng hệ thống theo mô hình ES.

2.4.1.2.1. So sánh giải pháp sử dụng tầng ORM (Object Relational Mapping) và ES

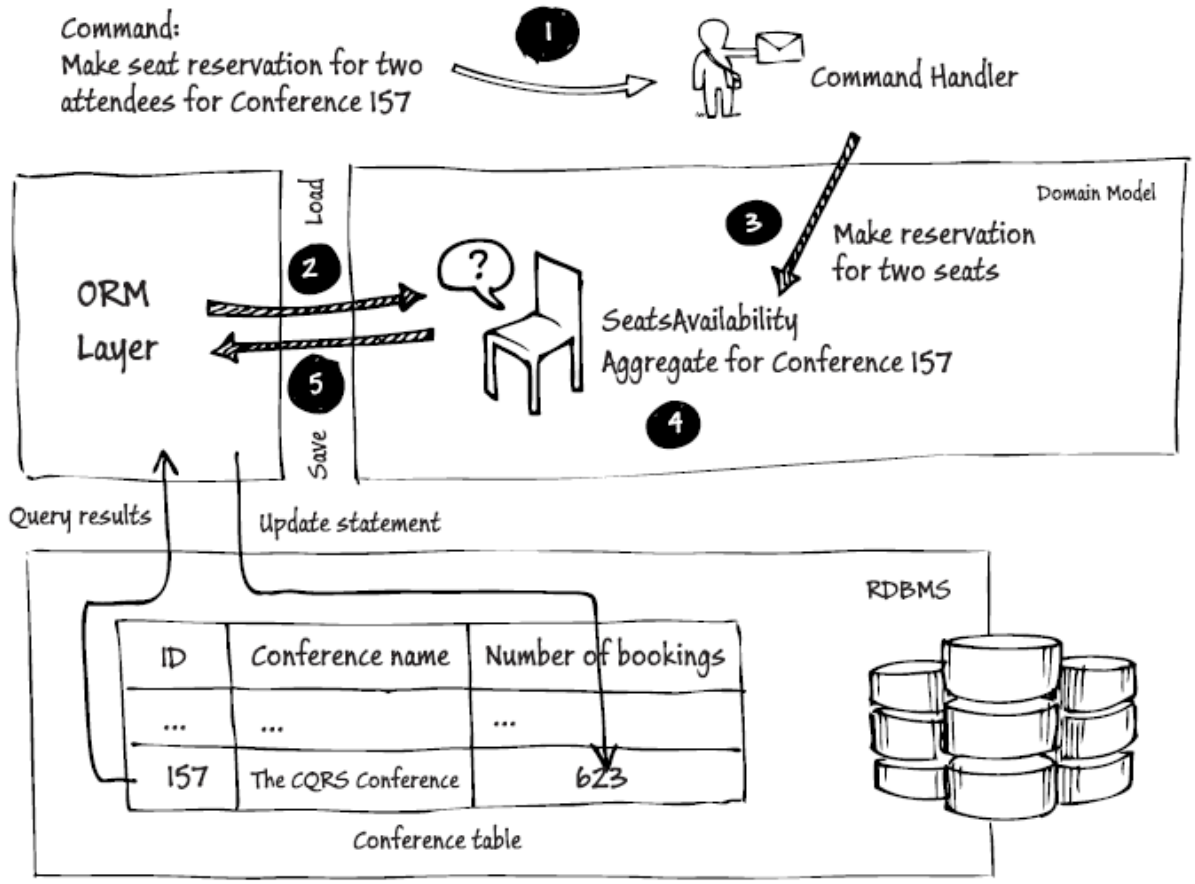


FIGURE 1
Using an object-relational mapping layer

Hình 2.4.1 Mô tả cách tiếp cận thứ nhất để lưu trữ tổng số lượng đặt chỗ, sử dụng tầng ORM.

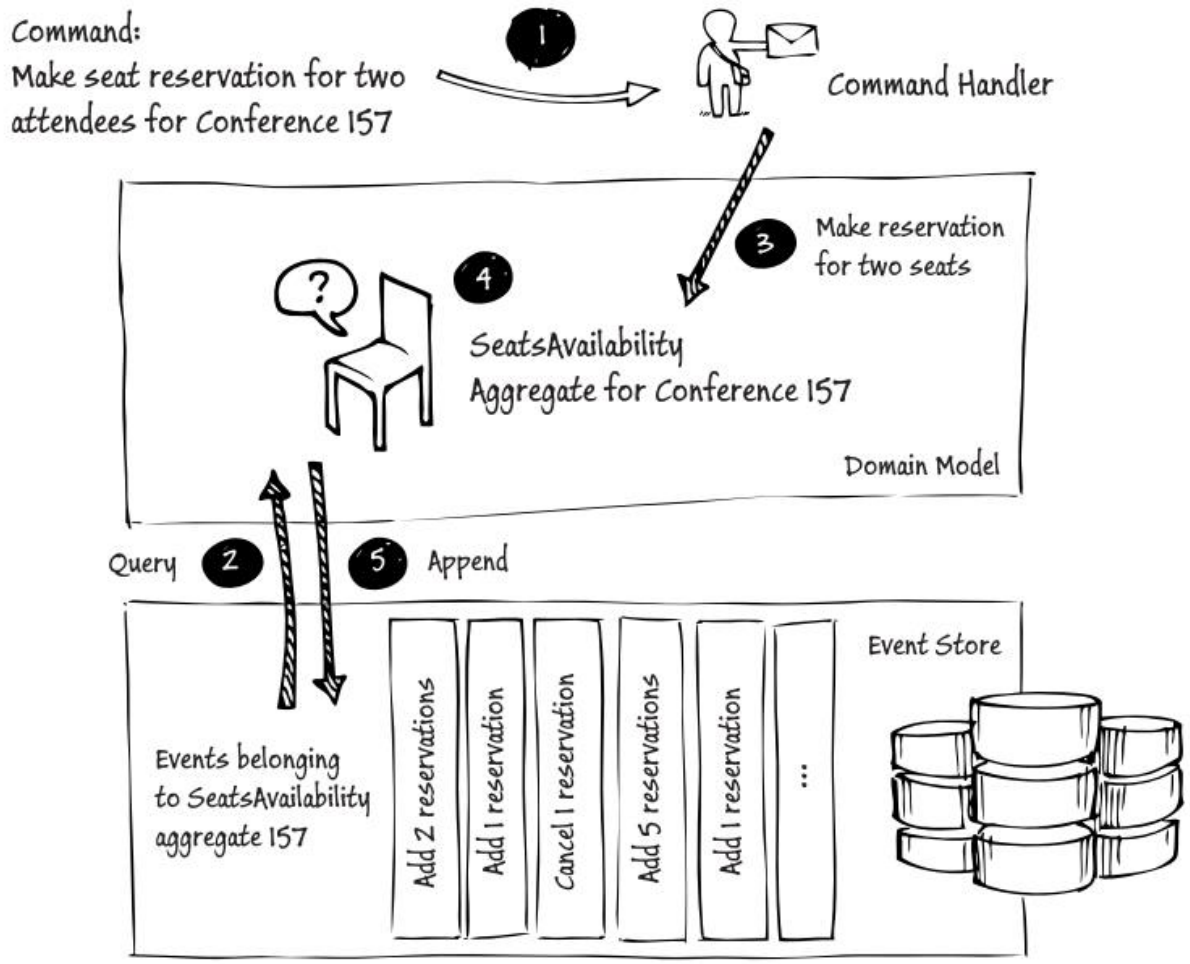
Trong đó, ORM là một kỹ thuật lập trình giúp ánh xạ dữ liệu trong RDBMS sang đối tượng được định nghĩa trong các lớp.

- Bước 1: Một saga hoặc giao diện người dùng (user interface - UI) sẽ đưa ra một command đặt chỗ cho hai người tại một hội nghị có ID là 157. Command này được xử lí bởi command handler của loại aggregate SeatsAvailability.
- Bước 2: Nếu cần thiết, tầng ORM sẽ tạo một thể hiện (instance) của aggregate với dữ liệu truy vấn tại kho lưu trữ dữ liệu. Dữ liệu này bao gồm số lượng đặt chỗ hiện tại cho hội nghị có ID là 157.
- Bước 3: Command handler sẽ gọi những phương thức xử lí trên instance được tạo ở bước 2 để thực hiện đặt chỗ.

- Bước 4: Aggregate SeatsAvailability thực hiện logic xử lí của nó. Trong ví dụ này, nó tính toán lại số lượng đặt chỗ cho hội nghị.
- Bước 5: Tầng ORM cập nhật thông tin vừa được xử lí của instance được tạo ở bước 2 tại kho lưu trữ dữ liệu.

Cách tiếp cận này hiệu quả vì hầu hết các ứng dụng doanh nghiệp đều lưu trữ dữ liệu theo cách này. Tuy nhiên, tồn tại nhiều hạn chế và giới hạn:

- Khó khăn trong việc lưu trữ những đối tượng dữ liệu phức tạp của một ứng dụng được viết theo ngôn ngữ hoặc kiểu lập trình hướng đối tượng vào kho lưu trữ dữ liệu quan hệ (relational store), thuật ngữ gọi là Object-Relational impedance mismatch.
- Thiếu lịch sử thay đổi trạng thái của aggregate. Cách tiếp cận này chỉ lưu lại trạng thái hiện tại của aggregate. Một khi aggregate được cập nhật, không thể tìm thấy trạng thái trước đó của nó. Nếu một hệ thống phải lưu giữ lịch sử của một aggregate, các lập trình viên phải tự triển khai cơ chế này. Việc triển khai lưu trữ lịch sử cho các aggregate sẽ tốn rất nhiều thời gian.
- Công bố events là một cơ chế hữu dụng trong việc đồng bộ dữ liệu và gửi thông báo trong kiến trúc microservice. Tuy nhiên, cách tiếp cận này không hỗ trợ tự động công bố events mỗi khi aggregate được cập nhật. Do đó, lập trình viên phải tự triển khai dẫn đến khả năng gây ra sự bất đồng bộ với logic xử lí.



Hình 2.4.2 Mô tả cách tiếp cận thứ hai để lưu trữ tổng số lượng đặt chỗ, sử dụng ES thay vì tầng ORM và một RDBMS.

Trong đó, bước 1, 3, 4 giống với giải pháp sử dụng tầng ORM.

- Bước 1: Một saga hoặc giao diện người dùng (user interface - UI) sẽ đưa ra một command đặt chỗ cho hai người tại một hội nghị có ID là 157. Command này được xử lí bởi command handler của loại aggregate SeatsAvailability.
- Bước 2: Một instance được tạo bằng cách truy vấn tất cả những events liên quan đến aggregate SeatsAvailability với ID 157.
- Bước 3: Command handler sẽ gọi những phương thức xử lí trên instance được tạo ở bước 2 để thực hiện đặt chỗ.
- Bước 4: Aggregate SeatsAvailability thực hiện logic xử lí của nó. Trong ví dụ này, nó tính toán lại số lượng đặt chỗ cho hội nghị. Đồng thời, nó tạo một event

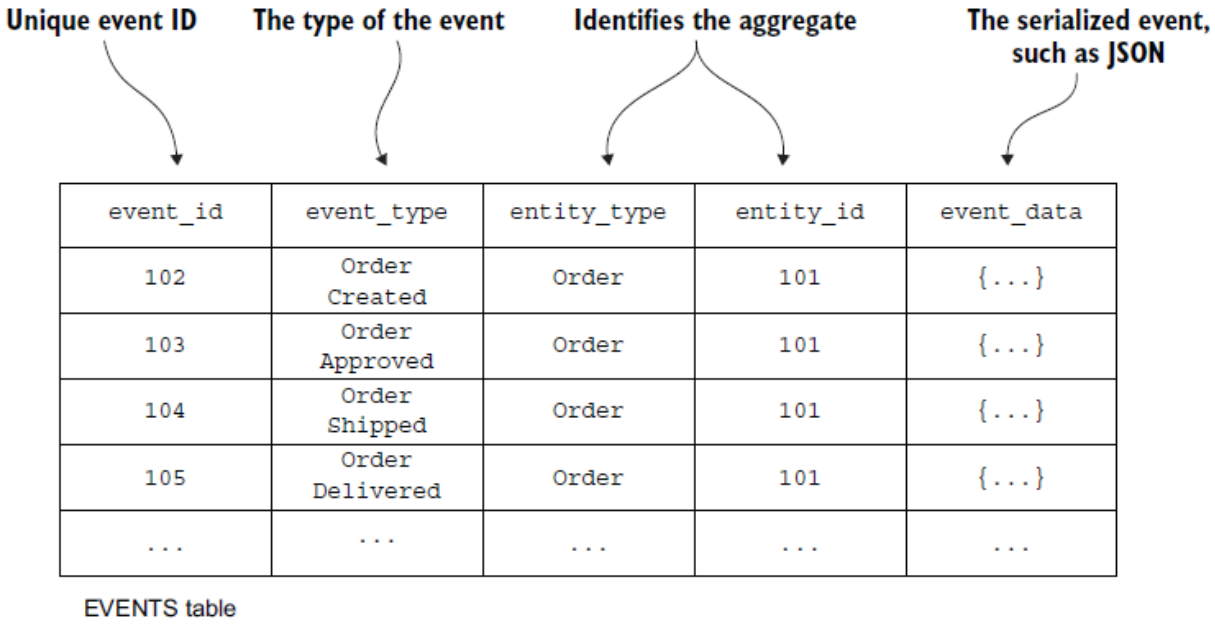
tương ứng với command ban đầu.

- Bước 5: Hệ thống sẽ thêm event đặt chỗ vào danh sách các events liên quan đến aggregate SeatsAvailability với ID 157 trong event store.

Có thể thấy cách tiếp cận thứ hai đơn giản hơn vì nó không cần dùng đến tầng ORM và nó thay thế một lược đồ quan hệ phức tạp trong kho lưu trữ dữ liệu với một cái đơn giản hơn – event store. Event store là nơi lưu trữ các events và trả về luồng events liên quan đến một instance của aggregate để có thể tái diễn các events và tạo lập trạng thái cho aggregate. Event store chỉ cần hỗ trợ truy vấn events bằng ID của aggregate và thêm vào những events mới. Với cách tiếp cận này, ta có được một lịch sử đầy đủ chứa các yêu cầu đặt hoặc huỷ chỗ của một hội nghị. Do đó, luồng sự kiện (event stream) trở thành "source of truth", các events được truy xuất từ một nguồn duy nhất là event stream, dẫn đến mọi trạng thái của ứng dụng đều được tái lập từ event stream. Việc khôi phục trạng thái của ứng dụng tại bất kì thời điểm nào trở nên dễ dàng nhờ việc tái diễn events.

2.4.1.2.2. Event Sourcing duy trì trạng thái của aggregate bằng cách sử dụng events

ES duy trì mỗi aggregate dưới dạng một chuỗi các events trong cơ sở dữ liệu, gọi là event store. Hình 2.4.3 cho thấy một ví dụ về việc duy trì trạng thái của aggregate Đơn hàng. Thay vì lưu trữ mỗi đơn hàng như một hàng (row) trong bảng Đơn hàng, ES lưu trữ mỗi đơn hàng như một hay nhiều hàng trong bảng Events. Mỗi hàng là một events, ví dụ như "Đơn hàng đã được tạo" (Order Created), "Đơn hàng được chấp nhận" (Order Approved), "Đơn hàng đã được giao" (Order Shipped),…



Hình 2.4.3 ES duy trì mỗi aggregate dưới dạng một chuỗi các events, lưu trong bảng Events

Khi hệ thống tạo hay cập nhật một aggregate, nó thêm events được tạo ra bởi aggregate vào bảng Events. Hệ thống sẽ lấy một aggregate từ event store bằng cách lấy các events liên quan đến nó và tái diễn các events này. Cụ thể, việc lấy một aggregate bao gồm ba bước:

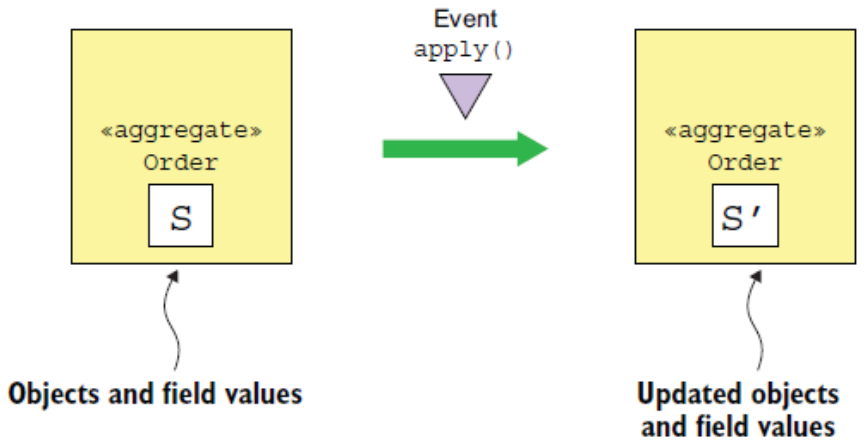
- Bước 1: Lấy các sự kiện liên quan đến aggregate từ event store.
- Bước 2: Tạo một instance cho aggregate sử dụng hàm khởi tạo mặc định.
- Bước 3: Lặp qua các events và gọi phương thức *apply()*.

Cách lấy aggregate này thực chất không khác biệt lắm so với việc tầng ORM lấy một thực thể (entity). Tầng ORM lấy một đối tượng bằng cách thực hiện một hoặc nhiều câu truy vấn SELECT để lấy được trạng thái hiện tại, sau đó khởi tạo đối tượng bằng hàm khởi tạo mặc định của nó. Điều khác biệt ở ES là việc xây dựng lại trạng thái của aggregate được thực hiện bằng các events.

2.4.1.2.3. Events đại diện cho sự thay đổi trạng thái

Mỗi sự thay đổi trạng thái của aggregate, bao gồm sự tạo lập, được đại diện bởi một event. Mỗi khi trạng thái aggregate thay đổi, nó phải đưa ra một event. Ví dụ, một aggregate Đơn hàng phải đưa ra event "Đơn hàng đã được tạo" khi nó được tạo, và event tương ứng khi nó được cập nhật.

Một event phải chứa dữ liệu cần thiết để aggregate thực hiện thay đổi trạng thái. Trạng thái của aggregate bao gồm giá trị của các thuộc tính của các đối tượng tạo nên aggregate. Một sự thay đổi trạng thái có thể chỉ là một sự thay đổi giá trị của một thuộc tính của một đối tượng. Đồng thời, nó cũng có thể bao gồm việc thêm hoặc xoá đối tượng, như việc thêm hay xoá các mặt hàng trong một đơn hàng. Ví dụ trạng thái hiện tại của aggregate là S và trạng thái mới là S' như trong hình 2.4.4. Một event E đại diện cho sự thay đổi trạng thái phải chứa dữ liệu để khi đơn hàng đang ở trạng thái S, gọi phương thức *apply(E)* thì trạng thái đơn hàng sẽ được cập nhật thành S'.



Hình 2.4.4 Mô tả sự thay đổi trạng thái aggregate Đơn hàng từ S sang S' khi gọi phương thức apply(E). Event E phải chứa dữ liệu cần thiết để thực hiện việc chuyển đổi này

2.4.2. Lợi ích của mô hình Event Sourcing

Event Sourcing chứa một lịch sử đầy đủ các events liên quan đến các aggregates. Đây là một tính năng quan trọng trong một số lĩnh vực, chẳng hạn như kế toán, ngân hàng, nơi luôn cần đến một lịch sử giao dịch tài chính đầy đủ và các giao dịch không thể bị thay đổi. Khi một giao dịch xảy ra, hệ thống không thể xóa hoặc thay đổi nó, mà chỉ có thể tạo

một giao dịch mới để điều chỉnh hoặc hoàn tác nếu cần thiết.

- Hiệu suất: Events là dữ liệu bất biến, chỉ có thể dùng thao tác thêm vào (append-only) khi lưu trữ chúng. Đồng thời, events là những đối tượng đơn giản, độc lập. Hai yếu tố này làm tăng hiệu suất và khả năng mở rộng cho hệ thống so với việc sử dụng các mô hình lưu trữ phức tạp.
- Đơn giản hoá: Events là những đối tượng mô tả những gì đã xảy ra trong hệ thống. Bằng cách đơn giản là lưu trữ các events, sẽ tránh được sự phức tạp liên quan đến việc lưu trữ các đối tượng dữ liệu vào cơ sở dữ liệu quan hệ.
- Lưu vết các thay đổi: Event Sourcing lưu trữ một lịch sử đầy đủ những trạng thái của hệ thống. Từ đó, có thể kiểm tra chi tiết những gì đã diễn ra trong hệ thống.
- Tích hợp với các hệ thống khác: Events cho phép tương tác một cách hiệu quả với các hệ thống khác. Event store có thể công bố sự kiện để thông báo cho các hệ thống sự thay đổi trạng thái của hệ thống.
- Trích xuất dữ liệu nghiệp vụ từ lịch sử sự kiện: Việc lưu trữ events cho phép khả năng xác định trạng thái của hệ thống tại thời điểm bất kỳ bằng cách truy vấn những events có liên quan đến đối tượng dữ liệu cho đến thời điểm đó. Điều này giúp trả lời những câu hỏi nghiệp vụ về lịch sử dữ liệu trong hệ thống. Ngoài ra, việc lưu trữ các events sẽ tránh trường hợp loại bỏ những thông tin có giá trị trong tương lai.
- Khắc phục sự cố ứng dụng khi chạy trên môi trường của khách hàng: Có thể dùng event store để khắc phục các sự cố trong hệ thống đang được khách hàng sử dụng bằng cách sao chép event store này và tái diễn lại các events trong môi trường kiểm thử. Nếu biết được thời điểm sự cố xảy ra, có thể dễ dàng tái diễn các sự kiện xảy ra cho đến thời điểm đó.
- Sửa lỗi: Khi phát hiện có lỗi trong quá trình viết mã dẫn đến hệ thống tính toán giá trị sai, thay vì sửa lỗi và điều chỉnh dữ liệu đã được lưu trữ một cách thủ công và rủi ro, có thể sửa lỗi và tái diễn luồng sự kiện.
- Tính linh động: Một chuỗi các events có thể được chuyển hoá thành bất kì dạng biểu diễn cấu trúc nào, ví dụ như cơ sở dữ liệu SQL (Structured Query Language).

2.4.3. Kết hợp cùng mẫu CQRS

Mẫu CQRS và ES thường được kết hợp với nhau, hỗ trợ lẫn nhau. ES là một mô hình tuyệt vời hỗ trợ việc kết nối giữa bên đọc và bên ghi dữ liệu. Events có thể trở thành nền tảng cho việc đồng bộ hoá trạng thái của hệ thống giữa kho lưu trữ dữ liệu bên ghi và bên đọc. Dữ liệu bên đọc là dữ liệu không được chuẩn hoá để tối ưu hoá quá trình truy vấn dữ liệu, phục vụ cho việc hiển thị dữ liệu cho người dùng. Khi có sự thay đổi dữ liệu bên ghi, có thể sử dụng events lưu trong event store làm dữ liệu để thông báo đến bên đọc. Bên đọc dữ liệu sẽ dùng thông tin lưu trong các events để thực hiện chuẩn hoá dữ liệu cần thiết phục vụ cho quá trình truy vấn.

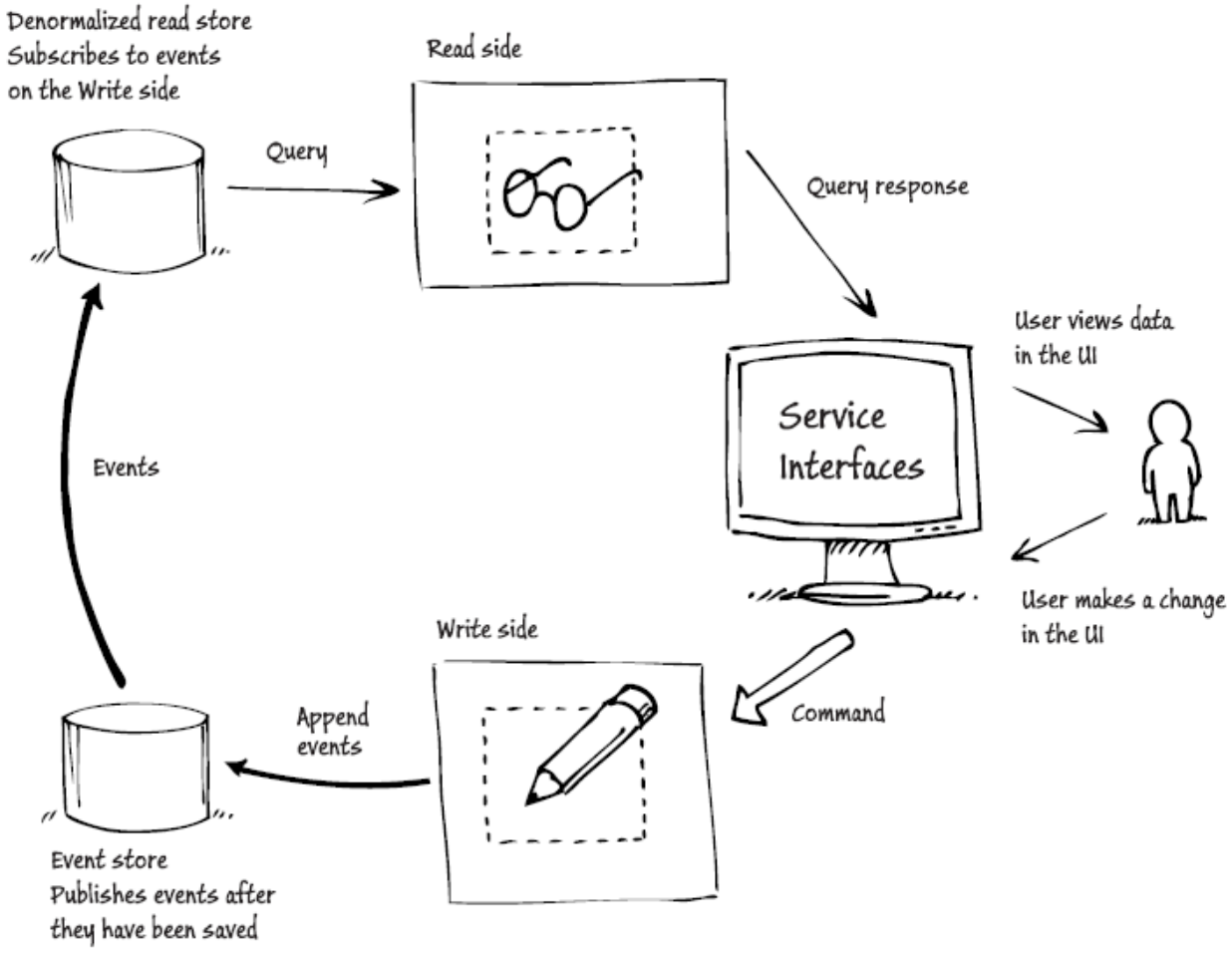


FIGURE 3
CQRS and event sourcing

Hình 2.4.3 Mô tả sự kết hợp giữa CQRS và ES.

Trong đó

- Khi bên ghi nhận một yêu cầu thay đổi dữ liệu, nó thực hiện logic xử lí và đưa ra một event tương ứng, đẩy vào event store. Ngay sau đó, nó công bố event này. Bên đọc theo dõi các events bên ghi và xử lí ngay khi có event được truyền đến. Do vậy, dữ liệu giữa bên ghi và bên đọc được đồng bộ trong thời gian thực.

Thông thường, khi triển khai mẫu CQRS, các aggregate sẽ đưa ra events để thông báo thông tin đến các bên quan tâm như các aggregate khác, kho lưu trữ dữ liệu bên đọc,… Khi sử dụng ES, các events này sẽ được lưu vào event store. Mỗi event đại diện cho một sự thay đổi trạng thái của aggregate. Điều này cho phép sử dụng các events đó để lấy được trạng thái của aggregate bằng cách tái diễn chuỗi các events liên quan đến aggregate đó. Do đó, mỗi khi một instance của aggregate đưa ra một event, hai điều sau phải xảy ra: hệ thống phải lưu event vào event store, hệ thống phải công bố event.

Dữ liệu bên đọc có thể được tái xây dựng tại mọi thời điểm bằng cách tái diễn các events từ event store tại bên ghi. Điều này là cần thiết nếu dữ liệu bên đọc trở nên bất đồng bộ với bên ghi hoặc dữ liệu bên đọc cần được điều chỉnh cấu trúc để hỗ trợ cho một câu truy vấn mới.

2.5. TỔNG KẾT

Kết thúc chương 2, chúng em đã đề cập những lý thuyết nền tảng liên quan tới các thuật ngữ Microservices, DDD, CQRS và ES. Chương 3 sẽ giới thiệu một số giải pháp thiết kế mà chúng em dự định sẽ áp dụng trong quá trình xây dựng và phát triển hệ thống.

CHƯƠNG 3: THIẾT KẾ GIẢI PHÁP

3.1. GIẢI PHÁP TỔNG QUÁT

Trong phần này, chúng em sẽ trình bày các giải pháp tổng quát cho những chức năng

chính của hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt, bao gồm: chức năng thanh toán chi phí sử dụng dịch vụ, cung cấp dịch vụ nhận dạng âm thanh tiếng Việt, cung cấp các báo cáo về việc sử dụng dịch vụ của người dùng. Bên cạnh đó, trình bày giải pháp xây dựng ứng dụng di động để ứng dụng hoá việc sử dụng dịch vụ mà hệ thống cung cấp. Và quan trọng hơn cả là trình bày giải pháp để hiện thực hóa khả năng mở rộng dịch vụ khi số lượng truy cập cao.

3.1.1. Giải pháp thanh toán chi phí sử dụng dịch vụ nhận dạng âm thanh tiếng Việt

Hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt cung cấp dịch vụ chuyển đổi một tập tin âm thanh tiếng Việt thành nội dung văn bản. Hệ thống hỗ trợ hai hình thức sử dụng dịch vụ, miễn phí và mất phí. Đối với hình thức mất phí, hiện nay, các hệ thống cung cấp dịch vụ tương tự trên thị trường như VAIS hay FPT.AI đều áp dụng mô hình thanh toán trước, sử dụng sau. Có thể thấy mô hình này mang lại lợi ích và sự thoải mái cho cả bên cung cấp dịch vụ và bên sử dụng dịch vụ. Bên cung cấp dịch vụ không phải xử lí trường hợp khách hàng sử dụng dịch vụ nhưng không thanh toán cũng như bên sử dụng dịch vụ không phải khó chịu vì bị nhắc nhở thanh toán chi phí sử dụng dịch vụ mỗi khi đến hạn.

Chúng em đề xuất áp dụng hình thức thanh toán trước, sử dụng sau này khi cài đặt chức năng thanh toán chi phí sử dụng dịch vụ cho hệ thống. Cụ thể, hệ thống sẽ hỗ trợ một số gói dịch vụ để lựa chọn khi khách hàng có nhu cầu sử dụng hình thức mất phí. Hệ thống sẽ liên kết với dịch vụ bên thứ ba, hỗ trợ thanh toán trực tuyến thông qua thẻ tín dụng. Khách hàng lựa chọn gói dịch vụ phù hợp và nhập thông tin cần thiết và thực hiện giao dịch. Hệ thống kiểm tra trạng thái giao dịch với dịch vụ bên thứ ba, nếu thành công, hệ thống tạo một khoá truy cập có số phút tương ứng với gói dịch vụ đã chọn cho khách. Hệ thống cho phép khách hàng sử dụng khoá truy cập này để sử dụng dịch vụ cho đến khi số phút bằng 0.

3.1.2. Giải pháp cung cấp dịch vụ nhận dạng âm thanh tiếng Việt

Cung cấp dịch vụ nhận dạng âm thanh tiếng Việt là một chức năng rất quan trọng, không

thể thiếu của hệ thống. Cụ thể hơn, hệ thống có nhiệm vụ cung cấp dịch vụ chuyển đổi một tập tin âm thanh thành nội dung văn bản. Để thực hiện chức năng này, chúng em đề xuất giải pháp như sau:

Bên sử dụng dịch vụ chọn lựa một khoá truy cập kèm một tập tin âm thanh gửi đến hệ thống. Hệ thống xử lí yêu cầu bằng việc sử dụng dịch vụ bên thứ ba để lưu trữ tập tin âm thanh. Sau đó, dựa vào khoá truy cập, hệ thống xác định được định danh (ID) khách hàng và số phút hợp lệ, số phút đã sử dụng của khoá. Hệ thống gọi API chuyển đổi âm thanh thành văn bản. Nếu API trả về dữ liệu thành công, hệ thống sử dụng dịch vụ bên thứ ba lưu trữ tập tin văn bản đã dịch, đồng thời tăng số phút đã sử dụng của khoá truy cập lên một lượng bằng với số phút của tập tin âm thanh và trả về dữ liệu văn bản cho khách hàng. Việc lưu trữ tập tin âm thanh và tập tin văn bản tại bên thứ ba giúp tiết kiệm bộ nhớ và dung lượng cho hệ thống, giảm độ phức tạp trong việc lưu trữ xuống cơ sở dữ liệu. Đồng thời, việc lưu trữ lại các tập tin giúp hệ thống cung cấp tập tin trong trường hợp người dùng có nhu cầu xem lại hoặc tải về.

3.1.3. Giải pháp cung cấp các báo cáo sử dụng dịch vụ của người dùng

Giải pháp này thể hiện ở hai quá trình lưu trữ và truy vấn. Thông thường, đối với quá trình lưu trữ, mỗi khi khách hàng truy cập dịch vụ của hệ thống, yêu cầu truy cập này sẽ được hệ thống lưu lại vào kho lưu trữ dữ liệu. Đối với quá trình truy vấn, khi cung cấp báo cáo sử dụng dịch vụ của khách hàng theo từng loại, ví dụ như báo cáo theo ngày, tuần, quí, tháng, năm, hệ thống sẽ truy vấn kho lưu trữ dữ liệu những dữ liệu liên quan và tiến hành lọc và tổng hợp dữ liệu phù hợp với từng loại báo cáo.

Đối với hệ thống cung cấp dịch vụ, trong một giây, có thể có hàng trăm, hàng ngàn yêu cầu truy cập, tương ứng với hàng trăm, hàng ngàn dòng trong kho lưu trữ dữ liệu. Việc cung cấp báo cáo theo cách này có thể mất rất nhiều thời gian, làm giảm hiệu suất của hệ thống để lấy được dữ liệu phù hợp. Chúng em đề xuất giải pháp đối với quá trình lưu trữ dữ liệu như sau. Hệ thống sẽ tự động tổng hợp các yêu cầu truy cập dịch vụ của khách hàng trong kho lưu trữ dữ liệu định kì theo ngày, tuần, quí, tháng, năm, tương ứng với

các loại báo cáo. Nhờ vậy, thời gian truy vấn dữ liệu cho việc cung cấp các báo cáo của hệ thống giảm đáng kể, phản hồi trên giao diện người dùng nhờ đó cũng nhanh hơn.

3.1.4. Giải pháp xây dựng ứng dụng di động

Để thể hiện mục đích của việc sử dụng dịch vụ nhận dạng âm thanh tiếng Việt được cung cấp bởi hệ thống, chúng em quyết định xây dựng một ứng dụng di động giúp rèn luyện phát âm tiếng Việt trên nền tảng Android thông qua các câu hỏi.

Ứng dụng di động có chức năng chính là rèn luyện phát âm tiếng Việt của người dùng thông qua việc đưa ra các câu hỏi và yêu cầu người dùng trả lời bằng cách ghi âm giọng nói. Ứng dụng sẽ chuyển đổi âm thanh tiếng Việt được ghi âm sang một văn bản tương ứng. Sau đó ứng dụng sẽ so sánh văn bản này với kết quả đúng của câu hỏi và hiển thị kết quả đúng/ sai cho người dùng. Thông qua kết quả này, người dùng sẽ điều chỉnh phát âm tiếng Việt của mình cho phù hợp.

3.2. THIẾT KẾ HỆ THỐNG

3.2.1. Thiết kế giao diện hệ thống

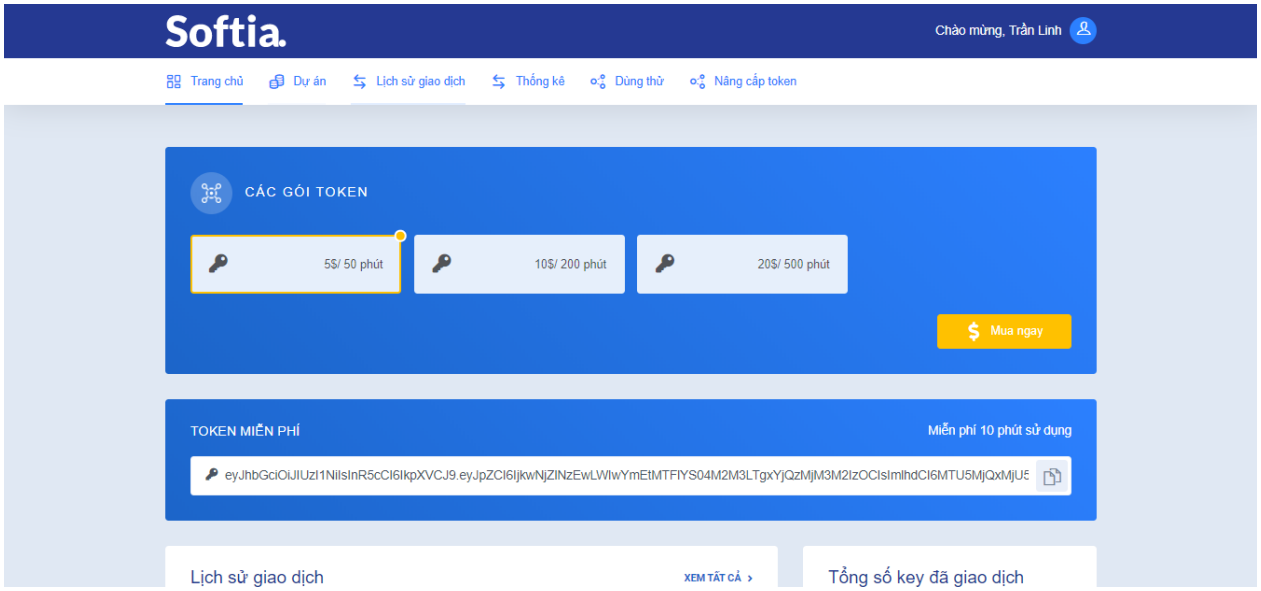
Giao diện hệ thống đóng góp một phần quan trọng quyết định lượng khách hàng sử dụng dịch vụ mà hệ thống cung cấp. Vì vậy, chúng em đã tập trung thiết kế giao diện sao cho trải nghiệm của khách hàng là tốt nhất. Phần sau đây sẽ liệt kê một số giao diện chính của hệ thống.

3.2.1.1. Giao diện trang chủ của hệ thống

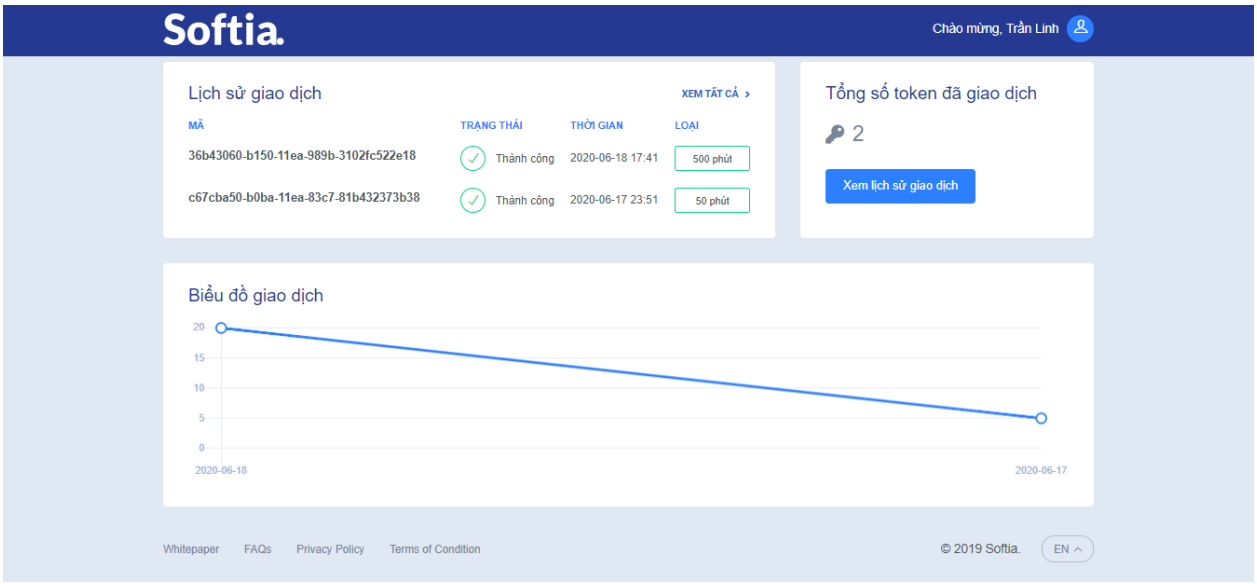
Trang chủ là nơi khách hàng được điều hướng đến sau khi đăng nhập thành công. Tại đây, khách hàng có thể lấy được khoá truy cập miễn phí tại mục *Token miễn phí* trong hình 3.2.1 hay lấy khoá truy cập mất phí thông qua việc thực hiện thanh toán chi phí sử dụng dịch vụ dựa trên các gói dịch vụ được liệt kê tại mục *Các gói token* trong hình 3.2.1, phục vụ cho mục đích chính là sử dụng dịch vụ nhận dạng âm thanh tiếng Việt hệ

thống cung cấp.

Ngoài ra, khách hàng có thể có cái nhìn tổng quan về các thông tin như các lần giao dịch gần nhất tại mục *Lịch sử giao dịch*, biểu đồ thống kê các lần giao dịch tại mục *Biểu đồ giao dịch* và tổng số khoá truy cập đã giao dịch tại mục *Tổng số token đã giao dịch* (hình 3.2.2).

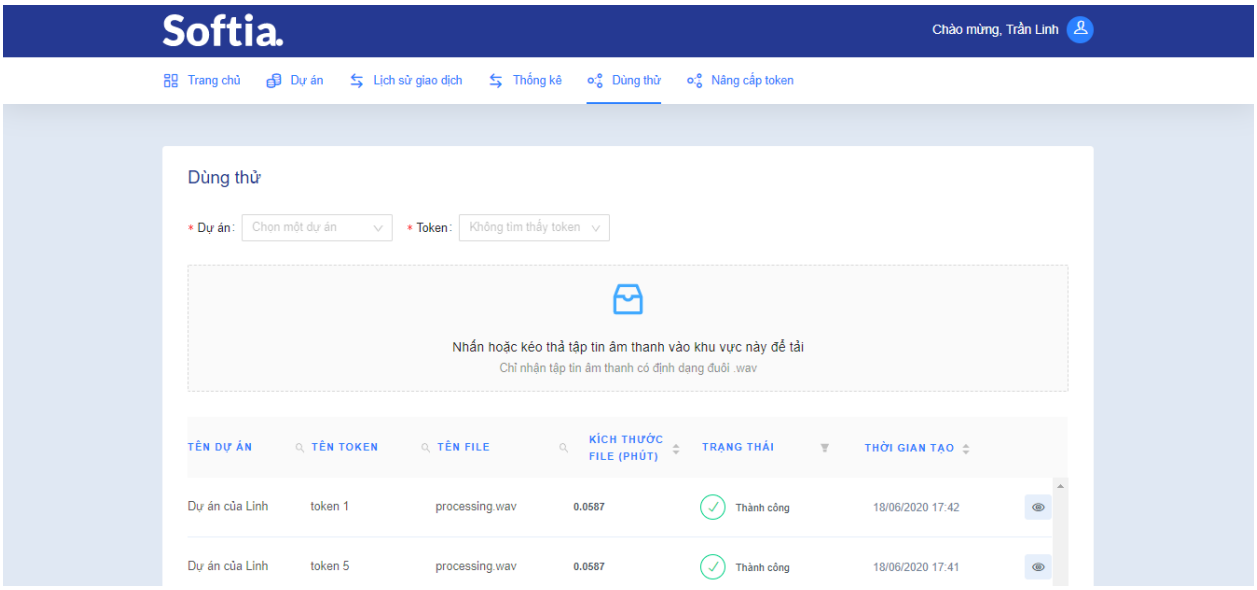


Hình 3.2.1 Giao diện trang chủ (phần 1)



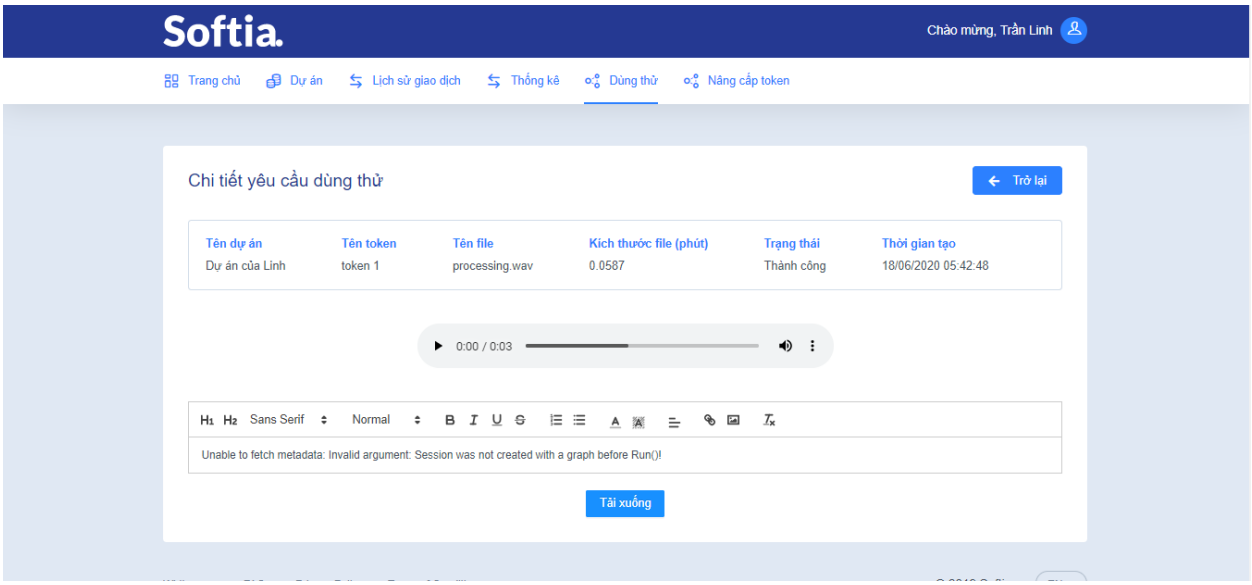
Hình 3.2.2 Giao diện trang chủ (phần 2)

3.2.1.2. Giao diện trang sử dụng dịch vụ



Hình 3.2.3 Giao diện trang sử dụng dịch vụ

Đây là trang nơi khách hàng có thể dùng thử dịch vụ hệ thống cung cấp với các khoá truy cập. Khách hàng chọn một dự án, một dự án chứa nhiều khoá truy cập, khách hàng sẽ chọn một khoá trong số đó và tải một tập tin âm thanh bằng cách nhấn hoặc kéo thả vào khu vực *Nhấn hoặc kéo thả tập tin âm thanh vào khu vực này để tải* trong hình 3.2.3. Sau khi hoàn tất, hệ thống sẽ xử lí yêu cầu và đưa khách hàng đến trang Chi tiết yêu cầu dùng thử như hình 3.2.4.

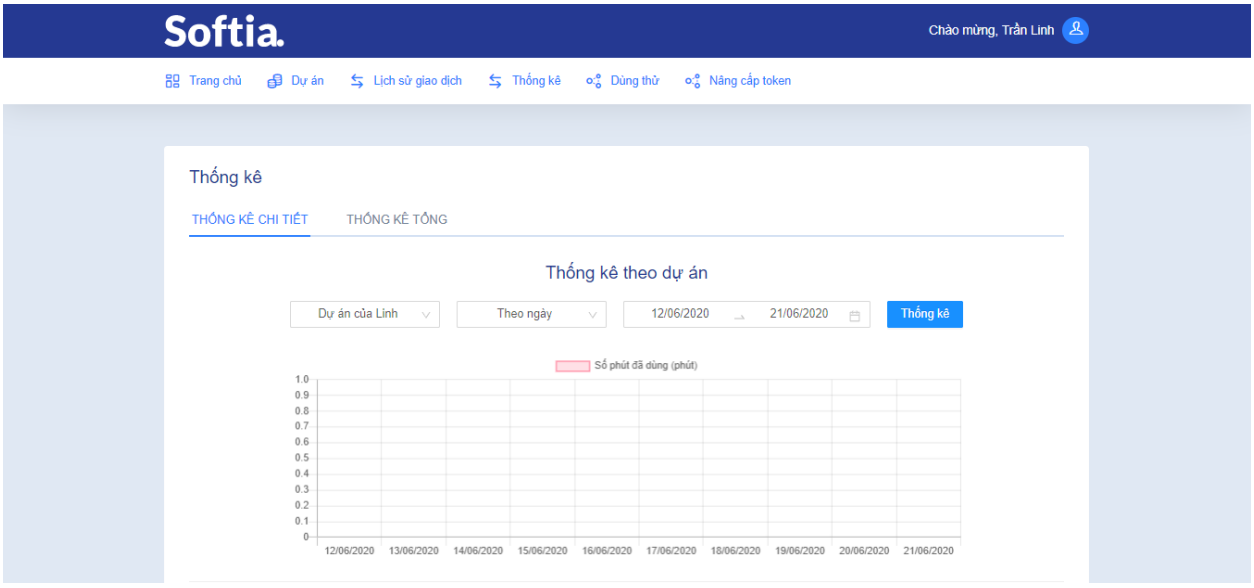


Hình 3.2.4 Giao diện trang kết quả khi sử dụng dịch vụ

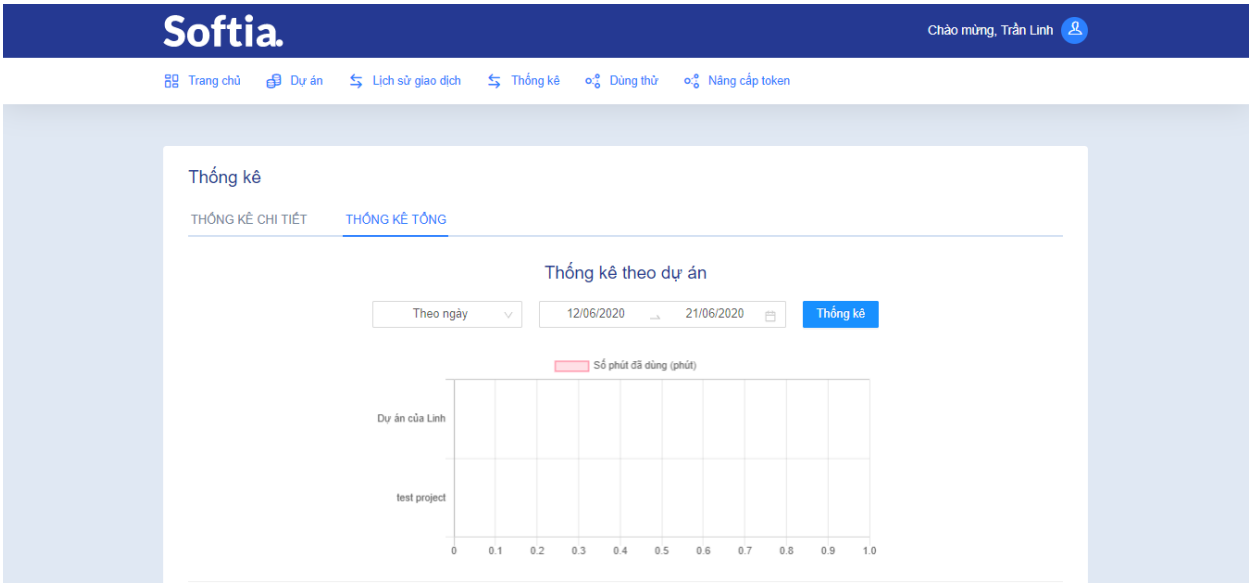
Trang Chi tiết yêu cầu dùng thử thể hiện kết quả hệ thống trả về khi khách hàng sử dụng dịch vụ, bao gồm thông tin dự án, khoá truy cập, tên tập tin, kích thước tập tin, trạng thái yêu cầu, thời gian tạo yêu cầu. Kết quả quan trọng nhất là hệ thống cung cấp tập tin âm thanh khách hàng đã chọn và văn bản được dịch tương ứng thể hiện trong khung chỉnh sửa nội dung văn bản trong hình 3.2.4. Hệ thống cho phép khách hàng tải xuống tập tin âm thanh và đoạn văn bản được dịch sau khi chỉnh sửa (nếu có).

3.2.1.3. Giao diện trang thống kê

Trang thống kê là nơi khách hàng có thể xem lại các báo cáo về việc sử dụng dịch vụ. Có hai loại thống kê, thống kê chi tiết (hình 3.2.5) và thống kê tổng (hình 3.2.6), cung cấp cho khách hàng một cách trực quan, đầy đủ các báo cáo về việc sử dụng dịch vụ của mình theo ngày, tuần, quí, tháng, năm. Thống kê chi tiết bao gồm thống kê theo một dự án, theo một khoá truy cập, theo một gói dịch vụ. Thống kê tổng bao gồm thống kê theo các dự án, theo các khoá truy cập, theo các gói dịch vụ.



Hình 3.2.5 Giao diện trang thống kê, phần thống kê chi tiết



Hình 3.2.6 Giao diện trang thống kê, phần thống kê tổng

3.2.2. Thiết kế và giải pháp lưu trữ dữ liệu

Đối với hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt, lịch sử truy cập dịch vụ của khách hàng là quan trọng nhất. Chúng em thực hiện việc lưu trữ dữ liệu này như sau. Dữ liệu được tổ chức như bảng 3.2.1.

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	tokenId	string	Định danh của khoá truy cập dùng để truy cập dịch vụ.
2	projectId	string	Định danh dự án chứa khoá truy cập dùng để truy cập dịch vụ.
3	userId	string	Định danh khách hàng sử dụng dịch vụ.
4	fileName	string	Tên tập tin âm thanh.
5	encoding	string	Quy tắc mã hoá kí tự của tập tin.
6	size	string	Kích thước tập tin tính theo đơn vị byte.
7	duration	number	Kích thước tập tin tính theo đơn

			vị phút.
8	mimeType	string	Tiêu chuẩn Internet về định dạng cho tập tin âm thanh, đa số có giá trị là audio/wav.
9	status	string	Trạng thái yêu cầu truy cập dịch vụ, có thể bằng một trong ba giá trị: PENDING (đang chờ xử lí), SUCCESS (thành công), FAILED (thất bại).
10	audioFileUrl	string	Đường dẫn truy cập tập tin âm thanh.
11	transcriptFileUrl	string	Đường dẫn truy cập tập tin văn bản đã được dịch.

Bảng 3.2.1 Tổ chức các thuộc tính để lưu trữ dữ liệu truy cập dịch vụ của khách hàng.

3.2.3. Thiết kế kiến trúc hệ thống

Trong hệ thống cung cấp dịch vụ, nhu cầu ghi dữ liệu thường lớn hơn rất nhiều so với nhu cầu truy xuất dữ liệu. Do đó, yêu cầu về khả năng mở rộng tại hai phía rất khác nhau. Đồng thời, một hệ thống cung cấp dịch vụ cần chú trọng phát triển nghiệp vụ xử lí yêu cầu truy cập dịch vụ của khách hàng một cách linh hoạt. Nhận thấy được sự phù hợp về mặt lợi ích mà mẫu CQRS mang lại được đề cập tại mục 2.3.4 ở chương 2, chúng em đề xuất thiết kế kiến trúc hệ thống áp dụng mẫu CQRS kết hợp.

3.3. TỔNG KẾT

Thông qua chương 3, chúng em đã làm rõ một số giải pháp tổng quan và cách thiết kế hệ thống để thực hiện xây dựng và phát triển hệ thống. Đồng thời, chúng em cũng trình bày sơ lược lý do vì sao chọn lựa các giải pháp này.

Ở chương kế tiếp, chúng em sẽ trình bày cách áp dụng những giải pháp đã nêu trong

chương 3 để cài đặt và xây dựng hệ thống. Cụ thể hơn, chúng em sẽ trình bày các thư viện, công cụ phục vụ quá trình cài đặt giải pháp và những cách khắc phục khó khăn cụ thể nếu có cho các giải pháp.

CHƯƠNG 4: CÀI ĐẶT GIẢI PHÁP

4.1. GIỚI THIỆU VỀ TYPESCRIPT VÀ NESTJS

4.1.1. TypeScript

Mã nguồn xây dựng hệ thống máy chủ phát triển dựa vào TypeScript. TypeScript là một ngôn ngữ mã nguồn mở miễn phí đang được phát triển và bảo trì bởi Microsoft. Nó có thể được xem là phiên bản nâng cao của JavaScript vì được bổ sung các tuỳ chọn kiểu tĩnh và lớp trên cơ sở lập trình hướng đối tượng (Object Oriented Programming – OOP). TypeScript có thể được sử dụng để phát triển ứng dụng chạy ở phía khách (client-side) và phía chủ (server-side).

4.1.1.1. Bắt đầu và kết thúc với JavaScript

TypeScript bắt đầu từ cùng một cú pháp và ngữ nghĩa được hàng triệu lập trình viên ngôn ngữ JavaScript biết đến. TypeScript biên dịch thành mã JavaScript đơn giản và gọn gàng chạy trên mọi trình duyệt, trong Node.js hay bất kì chương trình thực thi mã JavaScript (JavaScript engine) nào hỗ trợ ECMAScript 3 (hoặc mới hơn).

4.1.1.2. Công cụ mạnh cho các ứng dụng lớn

TypeScript cho phép các lập trình viên ngôn ngữ JavaScript sử dụng các công cụ và kĩ thuật phát triển như kiểm tra tĩnh (static checking) hay tái cấu trúc mã khi phát triển ứng dụng JavaScript. Việc TypeScript sử dụng các kỹ thuật mới nhất, kết hợp các thư viện JavaScript phổ biến và phương pháp OOP giúp phát triển các dự án lớn một cách dễ dàng.

4.1.1.3. Phiên bản nâng cao của JavaScript

TypeScript hỗ trợ các tính năng mới nhất và đang phát triển của JavaScript, bao gồm các tính năng từ ECMAScript 2015 và các đề xuất như các hàm bất đồng bộ (async functions) giúp xây dựng các thành phần hoạt động hiệu quả.

4.1.2. NestJS

Nest hay NestJS là một nền tảng để xây dựng các ứng dụng phía chủ bằng Node.js một cách hiệu quả và dễ mở rộng. Nó sử dụng ngôn ngữ TypeScript (tuy nhiên vẫn cho phép các lập trình viên sử dụng JavaScript thuần tuý) kết hợp các tính chất của phương pháp OOP, lập trình hướng hàm (Functional Programming – FP) và lập trình hướng phản hồi hàm (Functional Reactive Programming - FRP).

Về bản chất, Nest sử dụng các nền tảng máy chủ HTTP mạnh như là Express (mặc định) và có thể tùy chọn cấu hình để sử dụng Fastify. Nest cung cấp một mức độ trừu tượng vượt trên các nền tảng Node.js phổ biến này (Express/Fastify), nhưng cũng hỗ trợ giao diện lập trình ứng dụng (Application Programming Interface – API) trực tiếp cho lập trình viên. Điều này cho phép các lập trình viên có thể tự do sử dụng vô số các mô-đun của bên thứ ba.

Trong những năm gần đây, nhờ có Node.js, JavaScript đã trở thành ngôn ngữ chung cho cả ứng dụng front-end và back-end. Điều này đã tạo ra các dự án tuyệt vời như Angular, React và Vue, giúp cải thiện năng suất của lập trình và cho phép tạo ra các ứng dụng front-end nhanh, có thể kiểm thử và mở rộng. Tuy nhiên, trong khi có rất nhiều thư viện, công cụ tuyệt vời cho Node, không tồn tại thư viện nào giải quyết hiệu quả vấn đề chính là kiến trúc. Nest cung cấp một kiến trúc vượt trội cho ứng dụng, cho phép các lập trình viên và đội ngũ lập trình tạo ra các ứng dụng dễ kiểm thử, có thể mở rộng, kết nối lỏng lẻo và dễ bảo trì. Kiến trúc này được lấy cảm hứng từ Angular.

4.2. GIỚI THIỆU VỀ REACTJS

Đối với ứng dụng phía khách, nơi cho phép khách hàng đăng kí tài khoản và quản lí khoá truy cập phục vụ cho việc truy cập dịch vụ nhận dạng âm thanh tiếng Việt mà hệ thống cung cấp, chúng em xây dựng bằng React.

React hay ReactJS là một thư viện JavaScript mã nguồn mở để xây dựng giao diện người

dùng, được phát triển bởi Facebook và ra mắt năm 2013.

4.2.1. Lập trình khai báo

React giúp tạo giao diện người dùng mang tính tương tác một cách nhanh chóng, thiết kế các góc nhìn đơn giản cho từng trạng thái của ứng dụng. Hơn nữa, khi dữ liệu thay đổi, React sẽ cập nhật và hiển thị đúng các thành phần một cách hiệu quả.

Khi giải quyết một vấn đề, thay vì mô tả rõ những việc cần làm, với React, chỉ cần khai báo những việc muốn làm. Điều này giúp cho những đoạn mã trở nên dễ hiểu và dễ tìm lỗi hơn.

4.2.2. Phát triển dựa trên thành phần (Component-Based)

React xây dựng các thành phần có thể quản lý các trạng thái của riêng chúng, sau đó kết hợp lại để tạo ra giao diện người dùng phức tạp.
Vì logic xử lí của thành phần được viết bằng JavaScript, có thể dễ dàng truyền dữ liệu giữa các thành phần và tránh thao tác với cây DOM (Document Object Model).

4.3. CÀI ĐẶT KIẾN TRÚC HỆ THỐNG

Chúng em bắt đầu cài đặt kiến trúc cơ bản của hệ thống áp dụng mẫu CQRS kết hợp Event Sourcing bằng cách tham khảo kiến trúc mẫu tại https://github.com/qas/examples-nodejs-cqrs-es-swagger. Kiến trúc này được xây dựng dựa trên nền tảng NestJS được giới thiệu tại mục 4.1.2, kết hợp áp dụng mẫu CQRS và ES một cách cơ bản nhất. Dựa vào kiến trúc này, chúng em xây dựng và phát triển hệ thống phù hợp với yêu cầu mà luận văn đưa ra.

4.4. CÀI ĐẶT CHỨC NĂNG THANH TOÁN CHI PHÍ SỬ DỤNG DỊCH VỤ NHẬN DẠNG ÂM THANH TIẾNG VIỆT

Chức năng thanh toán chi phí sử dụng dịch vụ nhận dạng âm thanh tiếng Việt giúp khách hàng có được khoá truy cập để sử dụng dịch vụ cung cấp bởi hệ thống. Để thực hiện chức

năng này, chúng em sử dụng Stripe (một phần mềm tốt và phù hợp cho các hoạt động kinh doanh online) để thực hiện phần thanh toán chi phí qua thẻ tín dụng và JWT (JSON Web Token) (một tiêu chuẩn mở (RFC 7519) định nghĩa cách thức truyền tin an toàn giữa các bên dưới dạng đối tượng JSON) để tạo khoá truy cập cho khách hàng.

Tại ứng dụng phía khách xây dựng bằng React, chúng em cài đặt hai gói thư viện hỗ trợ sử dụng Stripe là *@stripe/react-stripe-js* và *@stripe/stripe-js*. Đồng thời, cài đặt gói thư viện *stripe* tại phía server. Tại trang chủ ứng dụng, khách hàng lựa chọn gói dịch vụ phù hợp nhu cầu sử dụng. Sau đó chọn dự án, đặt tên cho khoá truy cập muốn tạo và nhập thông tin thẻ tín dụng để thực hiện giao dịch. Hệ thống sẽ xử lí quá trình thực hiện giao dịch này như sau.

- Bước 1: Ứng dụng phía khách gửi yêu cầu tạo mã bí mật cho hệ thống cùng thông tin về giá gói dịch vụ được chọn.
- Bước 2: Hệ thống nhận yêu cầu từ ứng dụng, tạo mã bằng phương thức mà gói thư viện *stripe* cung cấp *stripe.paymentIntents.create()* với tham số là số tiền phải trả, đơn vị tiền tệ. Sau đó, hệ thống trả về mã cho ứng dụng.
- Bước 3: Ứng dụng tạo phương thức thanh toán bằng cách gọi hàm *useStripe().createPaymentMethod()* cung cấp bởi gói *@stripe/react-stripe-js* với tham số là các thông tin liên quan (số thẻ tín dụng, tên khách hàng, email khách hàng). Sau đó, xác nhận thanh toán bằng cách gọi hàm *useStripe().confirmCardPayment()* với tham số truyền vào là mã bí mật nhận từ phía server và định danh phương thức thanh toán vừa tạo.
- Bước 4: Sau khi xác nhận thanh toán thành công, Stripe trả về một định danh chứa thông tin thanh toán. Ứng dụng tiến hành gửi về phía server các thông tin sau: loại gói dịch vụ, định danh khách hàng, dự án, tên khoá truy cập và định danh thông tin thanh toán.
- Bước 5: Phía server khi nhận được đầy đủ thông tin, đầu tiên sẽ xác minh khách hàng đã thanh toán gói dịch vụ chưa thông qua thông tin thanh toán nhận được. Quá trình này được thực hiện bằng cách gọi hàm *stripe.paymentIntents.retrieve*() với tham số là định danh thông tin thanh toán. Nếu thanh toán thành công, hàm này sẽ

trả về một đối tượng chứa thuộc tính *status* với giá trị *succeeded*.

- Bước 6: Sau khi xác nhận thanh toán thành công, hệ thống sẽ kiểm tra tính hợp lệ của các thông tin còn lại như định danh người dùng, dự án,… Cuối cùng, hệ thống tạo một đơn hàng chứa các thông tin cần thiết và lưu trữ xuống cơ sở dữ liệu, đồng thời tạo và lưu trữ một khoá truy cập sử dụng chuỗi mã hoá JWT chứa thông tin người sử hữu, thông tin đơn hàng vừa tạo và trả về ứng dụng phía khách, hiển thị khoá truy cập cho khách hàng.

Hệ thống quản lí những khoá truy cập này dựa trên hai thuộc tính: số phút cho phép - *minutes* và số phút đã dùng - *usedMinutes*. *Minutes* có giá trị phụ thuộc vào gói dịch vụ khách hàng chọn khi thực hiện thanh toán chi phí. Mỗi khi khách hàng dùng khoá truy cập sử dụng dịch vụ, hệ thống sẽ cộng dồn số phút tương ứng vào thuộc tính *usedMinutes* của khoá cho đến khi giá trị hai thuộc tính bằng nhau. Lúc này, khách hàng không thể sử dụng khoá được nữa vì đã dùng hết số phút cho phép. Tuy nhiên, thay vì giao dịch mua khoá truy cập mới, hệ thống hỗ trợ chức năng nâng cấp khoá truy cập. Chức năng này chỉ khác với chức năng thanh toán sử dụng dịch vụ ở chỗ hệ thống sẽ cập nhật giá trị *minutes* của khoá thay vì tạo một khoá mới.

4.5. CÀI ĐẶT CHỨC NĂNG CUNG CẤP DỊCH VỤ NHẬN DẠNG ÂM THANH TIẾNG VIỆT

Chức năng cung cấp dịch vụ nhận dạng âm thanh tiếng Việt được xem là chức năng cốt lõi của hệ thống. Nó cho phép khách hàng sử dụng dịch vụ nhận dạng hệ thống cung cấp. Chúng em cài đặt chức năng này kết hợp sử dụng Firebase Storage. Đây là một dịch vụ được xây dựng trên nền tảng Google Cloud Platform, cho phép lưu trữ và quản lý các nội dung như ảnh, video, dữ liệu dạng tập tin.

Đầu tiên, chúng em cài đặt gói thư viện hỗ trợ sử dụng Firebase Storage là *firebase* tại ứng dụng phía khách. Tại trang dùng thử dịch vụ, khi khách hàng hoàn tất chọn tập tin âm thanh để tải lên, quá trình tải tập tin này lên Firebase Storage sẽ diễn ra thông qua việc gọi phương thức *firebase.storage().ref(url).put(file)*. Trong đó, *url* là đường dẫn đến tập tin

lưu trữ trong Firebase Storage, *file* là tập tin âm thanh khách hàng cung cấp.

Tiếp theo, ứng dụng gửi yêu cầu sử dụng dịch vụ về phía server cùng những thông tin: tập tin âm thanh, đường dẫn đến tập tin trên Firebase, khoá truy cập. Ở phía server, sau khi nhận được yêu cầu, hệ thống sẽ kiểm tra sự hợp lệ của các thông tin và thực hiện gọi API nhận dạng âm thanh tiếng Việt. Kết quả mà API trả về được hệ thống trả về cho ứng dụng phía khách, đồng thời hệ thống tiến hành lưu trữ yêu cầu sử dụng dịch vụ của khách hàng xuống cơ sở dữ liệu và cập nhật lại số phút đã dùng của khoá truy cập.
Ngay sau đó, ứng dụng phía khách nhận được văn bản đã dịch từ tập tin âm thanh và hiển thị cho khách hàng. Như đã đề cập tại mục 3.1.2, ứng dụng sẽ tạo tập tin với dữ liệu là văn bản được dịch và thực hiện lưu trữ ở Firebase Storage. Sau đó, gửi định danh yêu cầu sử dụng dịch vụ của khách hàng, yêu cầu hệ thống cập nhật đường dẫn đến tập tin được dịch. Hệ thống sẽ cập nhật đường dẫn này cho yêu cầu với định danh tương ứng.

4.6. CÀI ĐẶT CHỨC NĂNG CUNG CẤP CÁC BÁO CÁO SỬ DỤNG DỊCH VỤ CỦA NGƯỜI DÙNG

Chức năng cung cấp các báo cáo sử dụng dịch vụ của người dùng cho phép khách hàng quản lí việc truy cập dịch vụ của hệ thống, đồng thời giúp ban quản trị hệ thống quản lý được việc truy cập dịch vụ của khách hàng. Chúng em sử dụng kĩ thuật tạo cron job để định kì tổng hợp dữ liệu, cho phép hệ thống cung cấp các loại báo cáo một cách nhanh chóng và chính xác. Cron job là các lệnh thực thi định kì một hành động nào đó vào một thời điểm nhất định.

Thực hiện cài đặt gói thư viện *@nestjs/schedule* tại phía server. Tiến hành viết các phương thức cài đặt logic xử lí việc lọc và tổng hợp dữ liệu theo từng loại báo cáo (báo cáo theo ngày, tuần, quí, tháng, năm). Sau đó, thêm decorator *@cron()* với một trong các tham số sau. *cron* và *CronExpression* đều do gói thư viện *@nestjs/schedule* cung cấp.

- CronExpression.EVERY_DAY_AT_MIDNIGHT: thực hiện mỗi ngày vào lúc nửa đêm.
- CronExpression.EVERY_WEEK: thực hiện mỗi tuần.

- CronExpression.EVERY_1ST_DAY_OF_MONTH_AT_MIDNIGHT: thực hiện vào ngày đầu của tháng vào lúc nửa đêm.
- CronExpression.EVERY_QUARTER: thực hiện mỗi quí.
- CronExpression.EVERY_YEAR: thực hiện mỗi năm.

Bằng cách này, hệ thống định kì chạy các phương thức xử lí việc tổng hợp dữ liệu cho báo cáo, cho phép hệ thống cung cấp báo cáo một cách nhanh chóng và hiệu quả.

4.7. TỔNG KẾT

Trong chương 4, chúng em đã trình bày cách thức cài đặt và triển khai cho một số chức năng hệ thống cung cấp. Nội dung chi tiết cho một số phần cài đặt và kiến thức liên quan được chúng em trình bày ở phần phụ lục. Cuối cùng, chúng em sẽ tổng kết quá trình thực hiện luận văn cùng những kiến thức, sản phẩm thu được tại chương 5.

CHƯƠNG 5: TỔNG KẾT, ĐÁNH GIÁ

5.1. KIẾN THỨC THU ĐƯỢC

Trong suốt quá trình thực hiện khoá luận, chúng em đã học tập, lĩnh hội và áp dụng được nhiều kiến thức mới.

- Tiếp cận và áp dụng phương pháp quản lí công việc bằng mô hình Kanban.
- Thử sức ở nhiều vai trò khi xây dựng và phát triển hệ thống cung cấp dịch vụ như lập trình viên, quản lí dự án, kiểm thử dự án, thiết kế giao diện người dùng, khách hàng.
- Tiếp cận và áp dụng kiến trúc microservice, mẫu thiết kế DDD, mô hình nâng cao CQRS kết hợp Event Sourcing vào hệ thống cung cấp dịch vụ nhận dạng âm thanh tiếng Việt, tạo nền tảng cho quá trình phát triển trong lĩnh vực công nghệ phần mềm sau này.
- Cải thiện khả năng tự học, tìm kiếm và nghiên cứu tài liệu. Khả năng hệ thống

hoá kiến thức cũng được nâng cao.

- Học hỏi kỹ năng quản lí thời gian, sắp xếp các công việc hiệu quả.
- Tích luỹ thêm kinh nghiệm làm việc nhóm và cải thiện kỹ năng giao tiếp.
- Nâng cao kỹ năng viết mã nguồn, tìm kiếm và sửa lỗi, tái cấu trúc ứng dụng.

5.2. SẢN PHẨM THU ĐƯỢC

5.2.1. Môi trường phát triển

- Hệ điều hành: Windows 10 Pro
- Công cụ xây dựng hệ thống và phát triển ứng dụng di động: Microsoft Visual Studio Code 1.46.1
- Công cụ hỗ trợ phát triển ứng dụng di động: Android Studio 3.6
- Công cụ kiểm thử API: Postman 7.26.0
- Các thư viện / nền tảng sử dụng

Tên thư viện / nền tảng	Tóm tắt chức năng
NodeJS	Là một môi trường thực thi JavaScript miễn phí, đa nền tảng, cho phép thực thi các đoạn mã JavaScript phía back-end, vượt ra khỏi pham vi trình duyệt.
ExpressJS	Là một nền tảng nhỏ, được thiết kế để xây dựng ứng dụng web và API.
KafkaJS	Nền tảng stream dữ liệu phân tán sử dụng trong Node.js
Socket.io	Thư viện JavaScript phục vụ cho các ứng dụng thời gian thực, giúp các bên ở những địa điểm khác nhau kết nối với nhau, truyền dữ liệu ngay lập tức thông qua server trung gian.
ReactJS	Thư viện JavaScript mã nguồn mở giúp xây dựng giao diện người dùng.

Ant Design	Tập hợp các thành phần của React, phục vụ cho quá trình thiết kế liên quan đến giao diện.
NestJS	Là một nền tảng để xây dựng các ứng dụng phía chủ bằng Node.js một cách hiệu quả và dễ mở rộng.

5.2.1. Môi trường triển khai hệ thống

Nền tảng đám mây Google Cloud.

5.2.2. Các chức năng đã cài đặt

- Cho phép người dùng đăng kí/ đăng nhập
- Thanh toán chi phí sử dụng dịch vụ của hệ thống
- Cho phép người quản trị quản lý người đăng ký, khoá truy cập sử dụng dịch vụ
- Cung cấp API cung cấp dịch vụ chuyển một tập tin âm thanh tiếng Việt thành nội dung văn bản
- Cung cấp các báo cáo về việc sử dụng dịch vụ của người dùng
- Cung cấp thư viện SDK, ví dụ mẫu và tài liệu hỗ trợ lập trình viên sử dụng dịch vụ, với các ngôn ngữ khác nhau

5.2.3. So sánh với một số hệ thống khác trên thị trường

Tên chức năng	**VAIS**	**FPT.AI**	**SONIX**	**Hệ thống của chúng em (ViSpeech)**
Cho phép người dùng đăng kí/ đăng nhập	✓	✓	✓	✓
Thanh toán chi phí sử dụng dịch vụ của hệ thống	✓	✓	✓	✓

Cho phép người quản trị quản lý người đăng ký, khoá truy cập sử dụng dịch vụ	✓	✓	✓	✓
Cung cấp API cung cấp dịch vụ chuyển một tập tin âm thanh tiếng Việt thành nội dung văn bản	✓	✓	✓	✓
Cung cấp các báo cáo về việc sử dụng dịch vụ của người dùng	✓	✓	✓	✓
Cung cấp thư viện SDK, ví dụ mẫu và tài liệu hỗ trợ lập trình viên sử dụng dịch vụ, với các ngôn ngữ khác nhau				✓

Bảng 0.1.1 So sánh chức năng giữa các hệ thống

5.2.4. Đánh giá lợi ích sản phẩm khi áp dụng vào môi trường thực tế

5.3. SO SÁNH KẾT QUẢ THU ĐƯỢC SO VỚI MỤC TIÊU BAN ĐẦU

Mục tiêu ban đầu	**Nhận xét mức độ hoàn thành**

Bảng 0.1.2 Bảng so sánh mục tiêu ban đầu với kết quả thu được

5.4. Phương hướng phát triển và nghiên cứu trong tương lai

TÀI LIỆU THAM KHẢO

[1] www.sei.cmu.edu

[2] Documenting Software Architectures: Views and Beyond (2nd Edition), Clements et al, AddisonWesley, 2010, page 1.

[3] Philippe Kruchten. "*Architectural Blueprints—The '4+1' View Model of Software Architecture*" (www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf) Paper published in IEEE Software 12 (6), November 1995, pp. 42-50.

[4] Chris Richardson. "*Microservices Patterns: With examples in Java*", Figure 2.1, page 36

[5] Chris Richardson. "*Microservices Patterns: With examples in Java*", Figure 2.4, page 42

[6] Martin L. Abbott; Michael T. Fisher. "The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise", (Addison-Wesley, 2015)

PHỤ LỤC

PHỤ LỤC 1: