

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

Nghiên cứu, phân tích, đánh giá
và ứng dụng các công nghệ mới nhất
vào xây dựng
phân hệ quản lý đơn hàng và xuất kho
trong hệ thống quản lý phân phối

TẠ QUANG TÙNG

tung.tq154280@sis.hust.edu.vn

Ngành Công nghệ thông tin

Lớp Kỹ sư tài năng - Công nghệ thông tin - K60

Giảng viên hướng dẫn: TS. Phạm Quang Dũng

Bộ môn: Khoa học máy tính

Viện: Công nghệ thông tin và truyền thông

Chữ ký của GVHD

HÀ NỘI, 6/2020

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

1. Thông tin về sinh viên

Họ và tên sinh viên: **Tạ Quang Tùng**

Điện thoại liên lạc: **0372582450**

Email: **tung.tq154280@sis.hust.edu.vn**

Lớp: **KSTN-Công nghệ thông tin-K60**

Hệ đào tạo: **Kỹ sư**

Đồ án tốt nghiệp được thực hiện tại: **Đại học Bách khoa Hà Nội**

Thời gian làm DATN: Từ ngày 07/02/2020 đến 26/06/2020

2. Mục đích nội dung của DATN

Đồ án xây dựng hệ thống quản lý phân phối trên nền tảng Web. Đồ án tập trung nghiên cứu, phân tích, đánh giá và áp dụng các công nghệ mới nhất.

3. Các nhiệm vụ cụ thể của DATN

- Tìm hiểu về ngôn ngữ ECMAScript 6, Go
- Tìm hiểu về thư viện ReactJS, Redux, Redux-Saga, React-Router
- Tìm hiểu về Redis, PostgreSQL, Docker, Kubernetes, Microsoft Azure
- Phân tích thiết kế hệ thống và cơ sở dữ liệu
- Xây dựng hệ thống quản lý phân phối phân hệ quản lý đơn hàng và xuất kho
- Xây dựng các bài thử nghiệm đánh giá hiệu năng hệ thống

4. Lời cam đoan của sinh viên

Tôi - **Tạ Quang Tùng** - cam kết đồ án tốt nghiệp là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của **TS. Phạm Quang Dũng**.

Các kết quả nêu trong DATN là trung thực, là thành quả của riêng tôi, không sao chép theo bất kỳ công trình nào khác.

Hà Nội, ngày 24 tháng 6 năm 2020

Tác giả DATN

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của đồ án tốt nghiệp và cho phép bảo vệ

Hà Nội, ngày 24 tháng 6 năm 2020

Giảng viên hướng dẫn

Lời cảm ơn

Đầu tiên, em xin được gửi lời cảm ơn chân thành đến các thầy giáo, cô giáo thuộc trường đại học Bách Khoa Hà Nội, đặc biệt là các thầy giáo, cô giáo thuộc Viện Công nghệ Thông tin và Truyền Thông đã tận tình dạy dỗ và trang bị cho em những kiến thức bổ ích trong năm năm vừa qua.

Đồng thời em cũng xin được gửi lời cảm ơn đặc biệt đến TS. Phạm Quang Dũng. Thầy đã cho em những kiến thức và kinh nghiệm quý báu giúp em có thể hoàn thành đồ án này.

Em xin gửi lời cảm ơn tới gia đình và bạn bè đã sát cánh, động viên và hỗ trợ em trong suốt những năm qua.

Tóm tắt nội dung đồ án

Trong ngày nay, xã hội ngày càng hiện đại hóa, nhu cầu con người cũng cao lên. Cùng với đó kéo theo sự phát triển của các công ty, cửa hàng bán lẻ và chợ thương mại điện tử. Nhu cầu về quản lý hàng hóa bán lẻ cũng ngày càng tăng lên. Trên thị trường đã có một số ứng dụng cho quản lý bán lẻ phổ biến như KiotViet hay Sapo, tuy nhiên những sản phẩm này còn sử dụng công nghệ cũ và còn một số hạn chế. Vì vậy chúng tôi muốn xây dựng một phần mềm quản lý bán lẻ trực tuyến cung cấp các giải pháp quản lý phân phối hàng hóa.

Trong những năm gần đây, công nghệ Web đang càng trở nên phổ biến trong xây dựng các ứng dụng quản lý doanh nghiệp. Đồng thời nhờ sự phát triển của trình duyệt và ngôn ngữ JavaScript đã đem đến nhiều các Framework hỗ trợ lập trình viên xây dựng những ứng dụng phức tạp, tương tác cao, đơn trang (Single Page Application) mà không thể hoặc khó có thể thực hiện được dựa vào công nghệ làm Web cũ. Trong đồ án này, chúng tôi đã lựa chọn những công nghệ được coi là phổ biến được sử dụng nhiều hiện nay như ReactJS, Redux, Redux-Saga, Material-UI, Go làm back-end, Redis, PostgreSQL.

Đến nay, đồ án đã hoàn thành được các vấn đề đặt ra và chúng tôi có mong muốn tiếp tục duy trì và phát triển thêm tính năng cho ứng dụng đồng thời đưa hệ thống vào chạy trong thực tế.

MỤC LỤC

Lời cảm ơn	2
Tóm tắt	3
1 GIỚI THIỆU ĐỀ TÀI	10
1.1 Đặt vấn đề	10
1.2 Mục tiêu và phạm vi đề tài	10
1.3 Định hướng giải pháp	11
1.4 Bố cục đề án	11
2 CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ SỬ DỤNG	13
2.1 Giải pháp quản lý phân phối	13
2.1.1 Các khái niệm trong quản lý phân phối	13
2.1.2 Lợi ích của hệ thống quản lý phân phối	14
2.2 Lý thuyết và công nghệ	15
2.2.1 Role-Based Access Control	15
2.2.2 Công nghệ front-end	16
2.2.3 Công nghệ lưu trữ - Redis	24
2.2.4 Công nghệ lưu trữ - PostgreSQL	27
2.2.5 Công nghệ back-end	33
2.2.6 Docker	35
2.2.7 Kubernetes	37
2.2.8 Microsoft Azure	38
3 PHÂN TÍCH THIẾT KẾ HỆ THỐNG	40
3.1 Tổng quan các chức năng	40
3.1.1 Biểu đồ use case tổng quan	40
3.1.2 Biểu đồ use case phân rã các chức năng của hệ thống	40
3.2 Thiết kế cơ sở dữ liệu	48

4	CÁC VẤN ĐỀ GẶP PHẢI, GIẢI PHÁP ÁP DỤNG VÀ MINH HỌA CÁC CHỨC NĂNG CỦA HỆ THỐNG	52
4.1	Vấn đề lựa chọn công nghệ	52
4.1.1	Công nghệ front-end	52
4.1.2	Công nghệ back-end	52
4.1.3	Công nghệ Full Text Search	53
4.2	Chức năng phân quyền động	53
4.3	Minh họa các chức năng của hệ thống	54
5	THỬ NGHIỆM VÀ ĐÁNH GIÁ	60
5.1	Triển khai chạy thử nghiệm trên cloud	60
5.2	Kiểm thử hiệu năng (performance testing)	61
5.2.1	Kiểm thử hiệu năng cơ sở dữ liệu	62
5.2.2	Kiểm thử hiệu năng REST API	63
5.3	Kiểm thử sức chịu tải (stress testing)	65
6	KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	67
6.1	Kết luận	67
6.2	Hướng phát triển của đề án trong tương lai	68

DANH MỤC HÌNH VẼ

2.1	Các mô hình RBAC	16
2.2	Kiến trúc của DOM	17
2.3	Virtual DOM Snapshots & Diffing	18
2.4	Single Page Application	19
2.5	Ví dụ React Router	20
2.6	Truyền state giữa các component	21
2.7	Quản lý state trong Redux	21
2.8	Kiến trúc của Redux	22
2.9	Kiến trúc của Redux với Middleware	23
2.10	Redis server	25
2.11	Ứng dụng Redis trong bảng xếp hạng game	26
2.12	Cấu trúc của GIN	32
3.1	Biểu đồ use case tổng quan	40
3.2	Use-case quản lý người dùng	41
3.3	Biểu đồ hoạt động ca sử dụng quản lý người dùng	42
3.4	Biểu đồ hoạt động cho thao tác phân quyền	43
3.5	Use case quản lý xuất – nhập hàng	44
3.6	Use-case quản lý sản phẩm	45
3.7	Biểu đồ hoạt động cho thao tác gán giá của sản phẩm	46
3.8	Use-case quản lý đơn hàng	47
3.9	Biểu đồ hoạt động cho thao tác thêm đơn hàng	48
3.10	Cơ sở dữ liệu quản lý phân quyền, người dùng	49
3.11	Cơ sở dữ liệu quản lý đơn hàng	50
3.12	Cơ sở dữ liệu quản lý sản phẩm	51
4.1	Chức năng đăng nhập	54

4.2	Giao diện trang chủ hệ thống	55
4.3	Quản lý người dùng	55
4.4	Thêm người dùng / khách hàng	56
4.5	Quản lý các nhóm quyền	56
4.6	Gán quyền cho tài khoản người dùng	57
4.7	Gán quyền cho tài khoản người dùng	57
4.8	Quản lý giá của các sản phẩm	58
4.9	Tạo đơn hàng	58
4.10	Hiển thị đơn hàng	59
4.11	Quản lý xuất kho	59
5.1	Kiến trúc tổng quan của hệ thống	60
5.2	Triển khai lên Azure Kubernetes Service	61
5.3	Truy cập ứng dụng từ public IP của AKS	61
5.4	Truy vấn count trên máy local	62
5.5	Truy vấn count trên máy cloud	62
5.6	Truy vấn select trên máy local	63
5.7	Truy vấn select trên máy cloud	63
5.8	Hiệu năng REST API trên máy local (no aggregation)	64
5.9	Hiệu năng REST API trên máy local (aggregated)	64
5.10	Hiệu năng REST API trên máy cloud (no aggregation)	64
5.11	Hiệu năng REST API trên máy cloud (aggregated)	65
5.12	Stress test thêm inventory item trên máy local	65
5.13	Stress test thêm inventory item trên máy cloud	66

DANH MỤC BẢNG BIỂU

2.1	So sánh Redis và MySQL	27
3.1	Thông tin các bảng nhóm quản lý phân quyền, người dùng	49
3.2	Thông tin các bảng nhóm quản lý phân quyền, người dùng	50
3.3	Thông tin các bảng nhóm quản lý sản phẩm	51
4.1	Security_group và permission tương ứng	54

DANH MỤC CÂU LỆNH

2.1	Sử dụng Redux-Saga	24
2.2	Tạo index sử dụng Hash	29
2.3	Tạo index sử dụng GIN	31
2.4	Tạo index sử dụng GiST	31
2.5	Xây dựng image của back-end server	36

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Hiện nay, với sự phát triển kinh tế và sự phát triển của thương mại điện tử, lượng hàng hóa cung và cầu càng ngày càng lớn. Kéo theo lượng hàng hóa lớn đó là vấn đề phân phối và quản lý sao cho hiệu quả. Khi hàng hóa nhiều lên, các phương pháp quản lý kho cũ không còn đáp ứng được thì mỗi công ty hay tập đoàn phân phối bán lẻ đều cần phải đưa ra giải pháp mới để quản lý hàng hóa.

Hệ thống quản lý phân phối (Distribution Management System) là một ứng dụng cung cấp giải pháp cho vấn đề này. Với hệ thống quản lý phân phối, các công ty hay tập đoàn bán lẻ có thể nắm được chính xác biến động thị trường, hàng hóa đang ở vị trí nào trong chuỗi cung ứng, hàng tồn kho còn bao nhiêu hay hiệu suất làm việc của các nhân viên bán hàng, ...

Cùng với vấn đề quản lý một lượng ngày càng lớn các hàng hóa là sự phức tạp của các hệ thống quản lý phân phối (DMS) ngày càng cao. Nhờ sự phát triển của công nghệ trong thời gian hiện nay, đặc biệt là công nghệ Web đã tạo ra nhiều hướng đi, giải pháp mới trong xây dựng các hệ thống thông tin nói chung và DMS nói riêng. Tuy nhiên, với những doanh nghiệp đã và đang sử dụng những công nghệ không còn phổ biến, việc thay đổi công nghệ luôn là một điều khó khăn.

1.2 Mục tiêu và phạm vi đề tài

Từ sự cần thiết phải chuyển đổi phương pháp quản lý phân phối, nhiều phần mềm nghiệp vụ đã được triển khai. Ở Việt Nam hiện nay, các ứng dụng quản lý bán lẻ trực tuyến phổ biến có thể kể đến như KiotViet, Sapo, Suno, ... Các ứng dụng quản lý không cần trực tuyến như Adaline, BS Silver, Perfect Warehouse, ... Các ứng dụng quản lý bán lẻ này tuy cũng có những tính năng như quản lý sản phẩm, đơn hàng, ... tuy nhiên lại đơn giản so với một hệ thống quản lý phân phối đầy đủ của một doanh nghiệp lớn về phân phối hàng hóa.

Do sự phức tạp ngày càng cao của các hệ thống quản lý phân phối, tốc độ triển khai phát triển hệ thống cũng như hiệu năng của hệ thống ngày càng suy giảm. Các công nghệ mới đang dần trở thành xu hướng hiện nay có tiềm năng để giải quyết những vấn đề trên. Tuy nhiên, các doanh nghiệp đôi khi cũng không sẵn sàng cho việc thay đổi đó.

Vì vậy, chúng tôi muốn xây dựng một ứng dụng quản lý phân phối riêng biệt, áp dụng công nghệ phát triển và triển khai mới nhất, có đánh giá định tính nhằm xác định và cải thiện hiệu năng hệ thống, có phương pháp xây dựng nhằm khai thác được lợi thế của các dịch vụ Cloud. Ứng dụng quản lý phân phối của chúng tôi sẽ tập trung vào 8 tính năng chính, đó là phân quyền động, quản lý tài khoản, quản lý sản phẩm, quản lý kho, quản lý xuất nhập, quản lý đơn hàng, quản lý tuyến bán hàng và quản lý nhân viên bán hàng.

1.3 Định hướng giải pháp

Go là một ngôn ngữ lập trình mới do Google phát triển, được sinh ra để giúp ngành công nghiệp phần mềm khai thác nền tảng đa lõi của bộ vi xử lý và hoạt động đa nhiệm tốt hơn. Vì vậy chúng tôi sử dụng Go để viết back-end server nhằm tăng hiệu năng xử lý yêu cầu trên một thời điểm.

Còn về phần giao diện, giao tiếp với người dùng (front-end) thì hiện nay các Framework Javascript như ReactJS, Angular hay VueJS đang là xu thế bởi khả năng xây dựng giao diện nhanh, bảo trì và mở rộng code dễ dàng. Ứng dụng của chúng tôi sử dụng ReactJS, kết hợp Redux và Material-UI để xây dựng giao diện đảm bảo thiết kế chuẩn Material Design của Google.

Về cơ sở dữ liệu, chúng tôi sử dụng PostgreSQL vì đó là phần mềm mã nguồn mở có hiệu năng và tính mở rộng cao. Đồng thời sử dụng Redis làm nơi lưu trữ thông tin phiên làm việc (session) và dữ liệu cache. Chúng tôi sử dụng một công nghệ container đang được ưa chuộng là Docker và công nghệ điều phối là Kubernetes để xây dựng và triển khai ứng dụng. Còn về dịch vụ cloud, chúng tôi sử dụng Microsoft Azure làm nhà cung cấp dịch vụ cloud chính để ứng dụng chạy trên.

Để có thể đánh giá hiệu năng hệ thống, chúng tôi kết hợp đánh giá hiệu năng dựa vào công cụ có trên PostgreSQL và dựa vào chương trình dòng lệnh viết bằng Go để giả lập các truy cập từ người dùng.

1.4 Bố cục đồ án

Phần còn lại của báo cáo đồ án tốt nghiệp được tổ chức thành các chương như sau.

Chương 2 trình bày về cơ sở lý thuyết của đồ án. Trong đó bao gồm các công nghệ cốt lõi đã tìm hiểu để xây dựng ứng dụng, so sánh giữa các công nghệ được sử dụng với các công nghệ khác hiện nay. Bên cạnh đó là các thuật toán cơ sở được sử dụng trong xử lý logic cho cả phần front-end, back-end và cơ sở dữ liệu như Index,

Role-Based Access Control, thuật toán phân cụm.

Chương 3 trình bày chi tiết về thiết kế hệ thống. Từ sự cần thiết của giải pháp đã nêu ở trên, chúng tôi xác định ra những tính năng cần thiết nhất và xây dựng các ca sử dụng xung quanh những tính năng này. Phần này sẽ trình bày các biểu đồ ca sử dụng cho các chức năng, biểu đồ hoạt động thể hiện cách thức tương tác với hệ thống của người dùng. Cùng với đó là thiết kế cơ sở dữ liệu, xây dựng dữ liệu mẫu.

Chương 4 đưa ra đóng góp chính của chúng tôi trong đề án tốt nghiệp này, so sánh chúng với những giải pháp hiện tại, cơ hội áp dụng trong tương lai và khả năng mở rộng của đóng góp này. Chương này cũng đưa ra những vấn đề và hướng giải quyết của những vấn đề này trong quá trình thực hiện đề án đồng thời minh họa các chức năng của hệ thống.

Chương 5 mô tả quy trình triển khai ứng dụng lên cloud. Đánh giá hiệu năng hệ thống bằng stress test và performance test thông qua cơ sở dữ liệu và REST API.

Chương 6 là chương cuối cùng đưa ra kết luận, những vấn đề còn chưa giải quyết được và hướng phát triển của ứng dụng trong tương lai.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ SỬ DỤNG

2.1 Giải pháp quản lý phân phối

2.1.1 Các khái niệm trong quản lý phân phối

Trước tiên, ta cần tìm biết về một từ khóa đã trở nên rất phổ biến trong thời gian gần đây, đó là Logistics [1]. Hiểu một cách đơn giản thì Logistics là quá trình lên kế hoạch, áp dụng và kiểm soát các luồng dịch chuyển của hàng hóa hay thông tin liên quan tới nguyên nhiên liệu vật tư (đầu vào) và sản phẩm cuối cùng (đầu ra) từ thời điểm xuất phát tới điểm tiêu thụ. Đi kèm với Logistics là khái niệm về Quản trị chuỗi cung ứng (Supply Chain Management), Quản trị chuỗi cung ứng bao gồm tất cả những hoạt động quản trị logistics cũng như những hoạt động sản xuất và thúc đẩy sự phối hợp về quy trình và hoạt động của các bộ phận marketing, kinh doanh, thiết kế sản phẩm, tài chính, công nghệ thông tin. Khái niệm chuỗi cung ứng rộng hơn, bao gồm cả logistics và quá trình sản xuất. Ngoài ra chuỗi cung ứng chú trọng hơn đến hoạt động mua hàng (procurement) trong khi logistics giải quyết các vấn đề chiến lược, phối hợp giữa marketing và sản xuất.

Khi đã có một hình dung cơ bản về chuỗi cung ứng, ta đến với phần chính là **hệ thống quản lý phân phối** (Distribution Management System). Hệ thống quản lý phân phối (Distribution Management System) là phần mềm quản lý chuỗi cung ứng hàng hóa của doanh nghiệp, giúp họ quản lý các hoạt động phân phối hàng hóa ra thị trường, kiểm soát các kênh phân phối, quản lý nhân viên, người bán hàng, kiểm soát hàng hóa trong kho, hàng tồn kho, kế hoạch vận chuyển hàng hóa đến địa chỉ mua hàng,...

Ví dụ, Masan Group là một công ty lớn trong lĩnh vực kinh tế tư nhân Việt Nam, tập trung hoạt động trong ngành hàng tiêu dùng và tài nguyên của Việt Nam. Masan quản lý các nền tảng kinh doanh có quy mô lớn nhằm phát triển và khai thác các tiềm năng trong lĩnh vực tiêu dùng và tài nguyên. Còn rất nhiều những tập đoàn bán lẻ khác có cùng cơ chế quản lý kinh doanh như Masan, mỗi tập đoàn như vậy đều cần có một hệ thống riêng để quản lý chuỗi cung ứng của mình.

Vì vậy Distribution Management System là một trong số những phần mềm quản lý doanh nghiệp có tính ứng dụng cao, phù hợp với mọi doanh nghiệp sản xuất và phân phối. Đối với các doanh nghiệp lớn (ví dụ Masan, VinGroup) có đội ngũ nhân viên bán hàng đông đảo và các kênh phân phối phức tạp, phần mềm DMS càng quan trọng và là công cụ không thể thiếu. Các nhà quản lý của các tập đoàn

lớn này luôn đau đầu vì những câu hỏi như “Làm thế nào để nắm được nhanh nhất xu thế, biến động của thị trường?”, “Làm thế nào để kiểm soát phân phối tốt, duy trì tồn kho ở mức tối ưu, tiết kiệm thời gian?”, “Tự động hóa bán hàng, tăng hiệu quả bán hàng cho đội ngũ nhân viên bán hàng như thế nào?”. Với hệ thống quản lý kênh phân phối (DMS), các doanh nghiệp lớn với vài trăm, hoặc vài nghìn nhân viên bán hàng, hàng chục nghìn điểm bán sẽ dễ dàng làm được việc này.

2.1.2 Lợi ích của hệ thống quản lý phân phối

Phần trên đã trình bày về lý do tại sao các doanh nghiệp và tập đoàn bán lẻ phải sử dụng giải pháp quản lý phân phối và sau đây là những lợi ích chính, thấy rõ nhất theo như tìm hiểu của chúng tôi.

Thứ nhất Distribution Management System [2] là công cụ tự động hóa bán hàng, giúp nhân viên bán hàng (salesman) tiết kiệm thời gian, tăng chất lượng chăm sóc khách hàng, tối ưu doanh thu. Mọi thông tin nhân viên bán hàng cần khi ghé thăm một điểm bán lẻ (rental outlet) bao gồm thông tin về khách hàng (customer), lịch sử mua hàng, các báo cáo bán hàng, thông tin sản phẩm, chương trình khuyến mãi,... có sẵn trong ứng dụng di động DMS để nhân viên có thể dễ dàng theo dõi tại điểm bán.

Thứ hai Distribution Management System có thể quản lý hiệu quả làm việc của nhân viên bán hàng, nhà quản lý có thể nắm được lộ trình ghé thăm khách hàng của nhân viên trên bản đồ lịch sử checkin. Thông qua đó đánh giá được nhân viên bán hàng có đang tích cực ngoài thị trường hay không, khách hàng có được chăm sóc tốt hay không.

Thứ ba, đặc biệt hơn là qua hệ thống nhà quản trị sẽ biết được những cửa hàng nào đang liên tục phát sinh doanh số, những cửa hàng nào lâu rồi chưa phát sinh đơn hàng mới. Từ đó có thể sắp xếp, phân chia hợp lý các nhân viên bán hàng vào các tuyến bán hàng, thay đổi tần suất viếng thăm khách hàng cho phù hợp với thực tế, tránh lãng phí nguồn lực đồng thời có thể chăm sóc được kỹ hơn các cửa hàng trọng tâm.

Thứ tư, cập nhật thị trường là điểm nổi bật của phần mềm Distribution Management System. Giúp các doanh nghiệp phân phối có thể kiểm soát được hàng tồn tại từng điểm bán lẻ, đại lý nhằm đưa ra kế hoạch sản xuất, điều phối hàng hóa phù hợp.

2.2 Lý thuyết và công nghệ

2.2.1 Role-Based Access Control

Trong an toàn thông tin, role-based access control (RBAC) là một cách tiếp cận cho vấn đề hạn chế quyền truy cập của người dùng. RBAC được sử dụng phổ biến tại các tổ chức lớn với nhiều hơn 500 nhân viên.

RBAC có nhiều mô hình khác nhau. Mô hình cơ bản là $RBAC_0$, mô hình có chứa quan hệ phân cấp là $RBAC_1$, có chứa ràng buộc $RBAC_2$ và mô hình tăng cường (consolidated) $RBAC_3$ [9].

Các thành phần chính của RBAC bao gồm user (người dùng), role (vai trò), permission (quyền) và session (phiên truy cập) cùng với quan hệ giữa chúng nhằm đơn giản hóa quá trình gán quyền cho người dùng.

User hay người dùng trong mô hình RBAC là con người, tuy nhiên khái niệm này có thể được tổng quát hóa bằng việc bao gồm cả những tác nhân tự động như robot hay phần mềm khác. Một role là một chức năng công việc hay chức vụ trong tổ chức đi kèm với các thẩm quyền mà nhân viên trong chức vụ đó nắm giữ. Còn một permission là sự một chấp nhận cho một phương thức truy cập vào một hay nhiều các đối tượng trong hệ thống. Việc một permission tương ứng với gì thì tùy thuộc vào hệ thống implement ra sao mô hình RBAC.

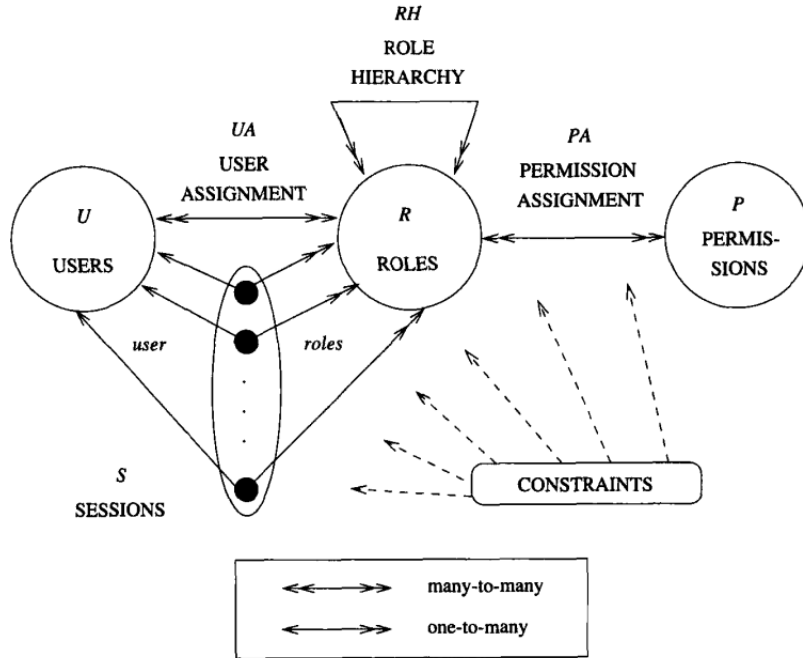
Mỗi một session là một ánh xạ từ mỗi phần tử của tập user sang tập con của tập các role. Session thiết lập một phiên truy cập với tập các role được kích hoạt (là một tập con của tập các role mà người dùng đó được gán). Mỗi một phiên truy cập tương ứng cho một user, trong $RBAC_0$ sự tương ứng này là không thay đổi theo thời gian.

Trong RBAC, permission được gán với một hoặc nhiều role, người dùng hay nhân viên được gán với một hay nhiều role. Người dùng trong RBAC không được gán trực tiếp các permission mà phải thông qua role, từ đó đơn giản hóa quá trình gán permission hay việc chuyển phòng ban công tác của nhân viên.

Trong đề án này chỉ sử dụng mô hình điều khiển truy cập dựa trên $RBAC_0$, biểu diễn toán học của $RBAC_0$ như sau:

- Tập U , R , P và S tương ứng với tập hợp của các user, role, permission và session.
- Tập $PA \subseteq P \times R$ biểu diễn quan hệ nhiều nhiều giữa permission và role.
- Tập $UA \subseteq U \times R$ biểu diễn quan hệ nhiều nhiều giữa user và role.

- Ánh xạ $user : S \rightarrow U$ gán mỗi một session s cho một user $user(s)$.
- Ánh xạ $roles : S \rightarrow \mathcal{P}(R)$ gán mỗi một session s cho tập con của các role: $roles(s) \subseteq \{r | (user(s), r) \in UA\}$ (có thể thay đổi theo thời gian) và session sẽ có các permission là $\bigcup_{r \in roles(s)} \{p | (p, r) \in PA\}$



Hình 2.1: Các mô hình RBAC

2.2.2 Công nghệ front-end

2.2.2.1 ReactJS

2.2.2.1.1 Khái quát về ReactJS

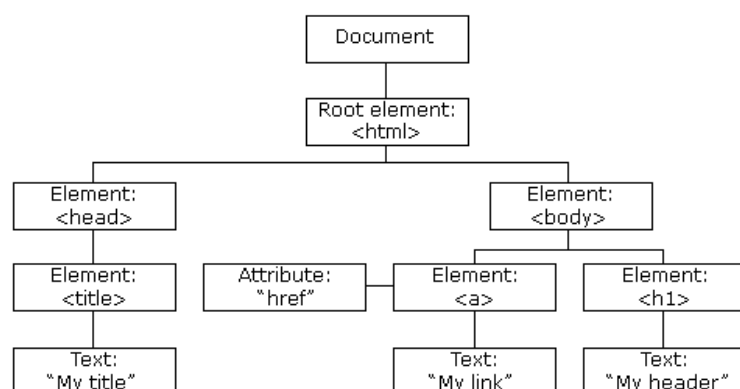
Ngày nay ReactJS đã trở nên rất phổ biến bởi những tính năng linh hoạt và đơn giản với hơn 1.300.000 developer và hơn 94.000 trang web đang sử dụng ReactJS (theo số liệu thống kê trên blog topdev.vn). ReactJS là một thư viện JavaScript mã nguồn mở được thiết kế bởi Facebook để tạo ra những ứng dụng web hấp dẫn, nhanh và hiệu quả với source code tối thiểu. Mục đích cốt lõi của ReactJS không chỉ khiến cho trang web phải thật mượt mà còn phải nhanh, khả năng mở rộng cao và đơn giản. Trên website chính thức của React tổng quan rằng: ReactJS – “A JavaScript for library for building user interface”, tức là React sinh ra để phục vụ tầng View, tập trung vào xây dựng giao diện.

Tư tưởng của ReactJS là xây dựng lên các component có tính tái sử dụng, dễ dàng cho việc chia nhỏ vấn đề, kiểm thử, giúp chúng ta dễ dàng quản lý, mở rộng hệ thống. Đặc tính của ReactJS là luôn giữ các component ở trạng thái stateless nhiều

nhất có thể, khiến ta dễ dàng quản lý nó. Bản thân các component này không có trạng thái, nó nhận đầu vào từ bên ngoài và chỉ hiển thị ra dựa vào các đầu vào đó, điều này cũng lý giải tính tái sử dụng (reuse) và tiện lợi trong kiểm thử (testing) của ReactJS.

2.2.2.1.2 Virtual DOM

Sử dụng ReactJS, ta thường hay nghe tới Virtual DOM, DOM thì rất quen thuộc với những lập trình viên front-end, còn Virtual DOM là gì? Có khác gì với DOM không?



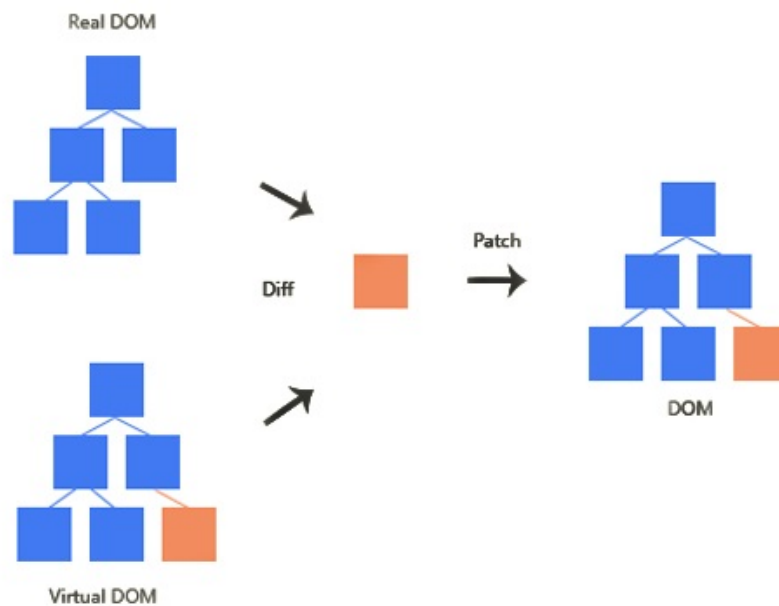
Hình 2.2: Kiến trúc của DOM

Trước tiên, DOM là viết tắt của Document Object Model là một chuẩn được định nghĩa bởi W3C dùng để truy xuất và thao tác trên code HTML bằng các ngôn ngữ lập trình thông dịch (scripting language) như JavaScript. Trong Hình 2.2 trên có thể thấy tất cả các thẻ HTML sẽ được quản lý trong các đối tượng document, thẻ cao nhất là thẻ html, tiếp đến là phân nhánh body và head, trong head có thể style, title, ... trong body chứa các thẻ html. Như vậy thông qua DOM, JavaScript có thể thay đổi tất cả các phần tử HTML, các thuộc tính HTML, CSS, loại bỏ hoặc thêm các thành phần và thuộc tính HTML mới, tạo ra các sự kiện khi tương tác, ... Tức là bạn có thể thay đổi cả trang web với DOM.

Khi DOM thay đổi, trình duyệt phải tính toán lại CSS và dựng lại trang web, điều này sẽ tốn thời gian, nhất là với những ứng dụng Single Page Application, việc sửa đổi DOM là liên tục không ngừng nghỉ. Hay khi xử lý các sự kiện (event) như click, submit, ... DOM sẽ tìm tất cả các node liên quan đến sự kiện và cập nhật nếu thấy nó cần thiết. Vậy thì có cần thiết khi phải tìm tất cả các node liên quan không? Hay sẽ hiệu quả hơn khi chỉ tìm node nào cần cập nhật.

Virtual DOM xuất hiện để giải quyết những vấn đề này. Virtual DOM gắn

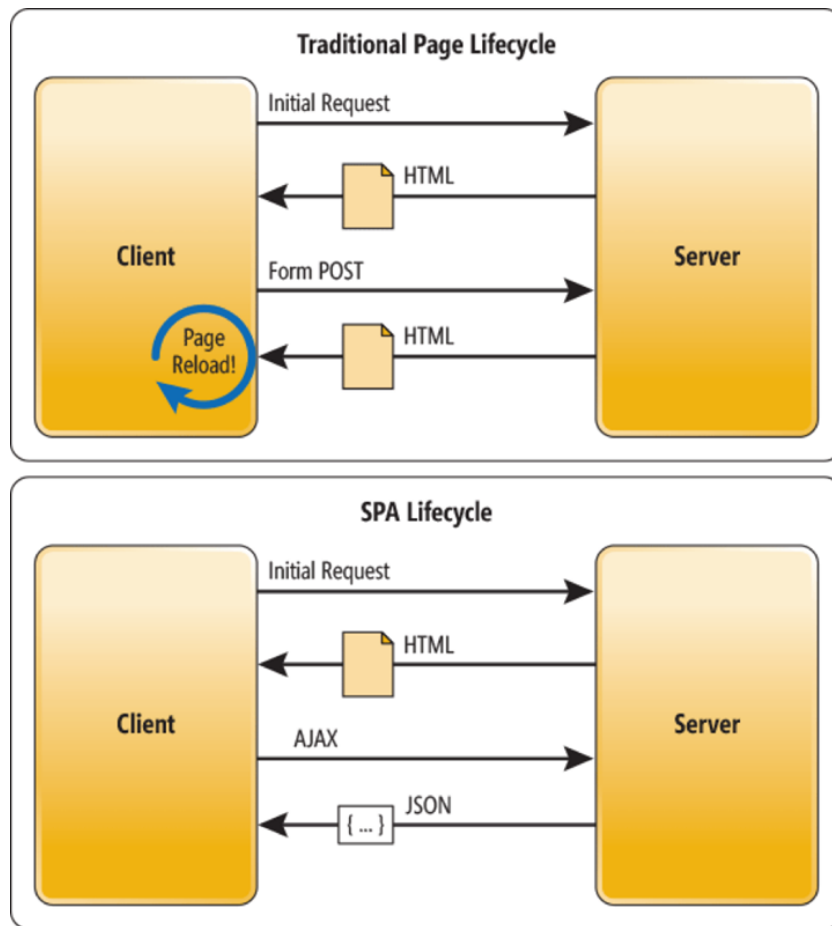
với ReactJS, thay vì xử lý DOM Tree thủ công, chúng định nghĩa các component trông giống DOM (vì vậy mà cú pháp JSX nhìn rất giống HTML), còn ReactJS sẽ thực hiện công việc ở tầng thấp hơn. Tổng quát thì Virtual DOM là một định dạng dữ liệu JavaScript nhẹ dùng để thể hiện nội dung của DOM tại một thời điểm nhất định nào đó. Nó có tất cả các thuộc tính giống như DOM nhưng không có khả năng tương tác lên màn hình như DOM. Sự đặc biệt của Virtual DOM nằm ở cơ chế Snapshots và Diffing. Khi cần cập nhật phần tử giao diện, React sẽ lấy một snapshot của Virtual DOM (có thể hiểu là bản ghi trạng thái ngay lúc đó), sử dụng snapshot này để so sánh với một Virtual DOM trước khi thực hiện thay đổi.



Hình 2.3: Virtual DOM Snapshots & Diffing

2.2.2.1.3 Single Page Application (SPA)

Với ReactJS, ta dễ dàng tạo ra một Single Page Application (SPA). Khác với những ứng dụng web truyền thống, Single Page Application có một trang gốc và trong trang gốc đó, chúng ta có thể tải nhiều trang con (tương ứng với các thành phần của trang gốc) mà không gây bất kỳ ảnh hưởng gì đến trang gốc. Trong khi các ứng dụng web truyền thống phải tải lại toàn bộ trang khi chúng ta tương tác với trang web thì Single Page Application chỉ load phần trang cần thiết. Các thành phần chung như header, footer, menu, side bar, ... thường xuất hiện ở nhiều trang của ứng dụng sẽ được Single Page Application load một lần duy nhất ở trang gốc.



Hình 2.4: Single Page Application

Do vậy Single Page Application mang lại nhiều ưu điểm như:

- Thứ nhất, việc render HTML ở server sẽ cực kì tốn tài nguyên nếu trang web có nhiều người dùng, với Single Page Application điều này chỉ xảy ra lần đầu tiên khi người dùng truy cập trang chủ (hoặc có thể không cần render trên server), còn sau đó việc render sẽ do client đảm nhiệm.
- Thứ hai, Single Page Application tách biệt front-end và back-end, SPA giao tiếp với server chủ yếu qua JSON REST API giúp cho dữ liệu gửi và trả giữa client và server giảm đến mức tối thiểu. Việc phát triển, kiểm thử cũng có thể độc lập giữa front-end và back-end.
- Thứ ba, trong suốt quá trình sử dụng, chỉ có dữ liệu là được truyền qua lại giữa client và server, còn các tài nguyên tĩnh (HTML, CSS, Script, ...) chỉ được tải một lần duy nhất, vì vậy sẽ giảm thiểu băng thông cho server.
- Thứ tư, Single Page Application giúp tăng trải nghiệm người dùng, là một ứng dụng web nhưng người dùng tương tác giống như một ứng dụng cho Desktop

vậy.

2.2.2.1.4 React Router

React Router là thư viện định tuyến (routing) chuẩn của React, nó giúp giao diện của ứng dụng đồng bộ với URL trên trình duyệt. React Router cho phép định tuyến luồng dữ liệu (data-flow) trong ứng dụng web một cách rõ ràng. Với React Router, việc xây dựng Single Page Application trở nên vô cùng dễ dàng.

```
<Switch>
  <Route exact path="/" component={Home}/>
  <Route path="/user" component={User}/>
  <Route path="/about" component={About}/>
</Switch>
```

Hình 2.5: Ví dụ React Router

Hình 2.5 trên thể hiện cấu hình cơ bản của React Router với đường dẫn (/path) và Route và giao diện tương ứng.

2.2.2.2 Redux

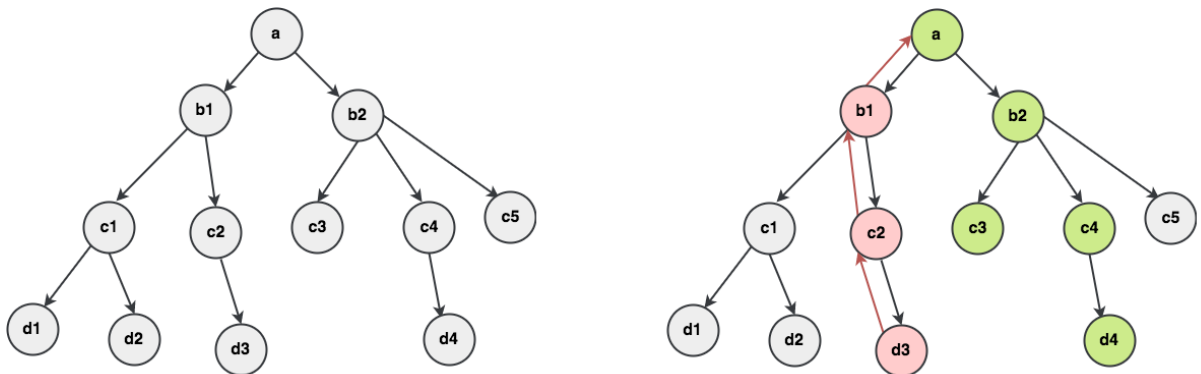
2.2.2.2.1 Khái quát về Redux

Một ứng dụng web sẽ nhận dữ liệu từ phía máy chủ (back-end), hay nhận những thao tác của người dùng (input, click, submit, ...), những thứ này chúng ta gọi đó là trạng thái (state) của ứng dụng. Nếu biết được trạng thái của ứng dụng tại một thời điểm nào đó, chúng ta sẽ biết vào thời điểm đó ứng dụng đã nhận dữ liệu nào, những thao tác nào đã được người dùng truyền lên.

Ví dụ: Khi chúng ta click vào nút Back / Forward trên trình duyệt thì mỗi trang là một trạng thái của ứng dụng.

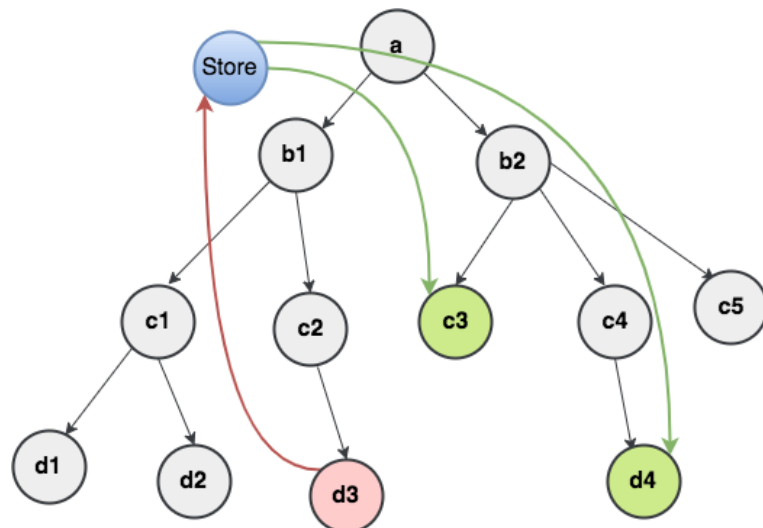
Như đã trình bày ở trên, ReactJS xây dựng lên các Single Page Application, tức chỉ render một trang, và tất cả các thành phần của ứng dụng sẽ được lưu trữ trong đó. Vì thế, nếu ứng dụng phức tạp lên theo thời gian, các component sẽ nhiều lên, và việc quản lý các state của chúng cũng ngày một lớn dần. Giao diện ứng dụng (UI) cũng trở nên phức tạp vì chúng ta cần quản lý các công việc active Routes, selected tabs, spinners, pagination, ... Trong ReactJS để truyền dữ liệu giữa các component anh em, một state phải tồn tại (live) trong một component cha, một phương thức (method) để update chính state này được cung cấp bởi component cha, từ đây sẽ truyền xuống props của các component con. Do vậy nếu một state phải được chia

sẽ giữa các component cách khá xa nhau trong một tree component thì state này sẽ phải được truyền từ một component đến một component khác cho đến khi nó đến được nơi mà nó được gọi.



Hình 2.6: Truyền state giữa các component

Trong hình vẽ trên, giả sử nếu có một sự kiện ở node d3 kích hoạt muốn thay đổi state d4 thì luồng dữ liệu sẽ được truyền từ node d3 trở về node gốc là a, sau đó từ node a lại truyền data đến các node con. Thứ tự truyền: d3 – c2 – b1 – a – b2 – c4 – d4. Tương tự nếu muốn thay đổi state ở c3 thì thứ tự truyền là: d3 – c2 – b1 – a – b2 – c3. Điều này làm cho bộ phận quản lý state trong ứng dụng trở nên phức tạp và bừa bộn, do vậy ta cần một công cụ quản lý trạng thái (state management tool) như Redux. Giải pháp Redux đưa ra như sau:

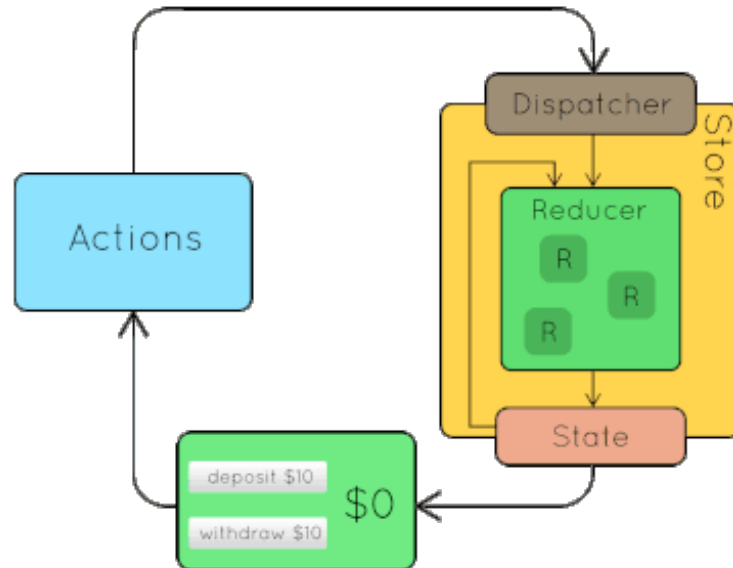


Hình 2.7: Quản lý state trong Redux

Quay lại ví dụ ở trên thì ta cần map sự kiện từ node d3 về store của Redux rồi ở node d4, c3 cần connect với store và cập nhật dữ liệu thay đổi.

2.2.2.2.2 Nguyên lý vận hành của Redux

Cách Redux hoạt động khá đơn giản. Redux có một store lưu trữ toàn bộ state của ứng dụng. Mỗi component có thể truy cập trực tiếp đến state được lưu trữ thay vì phải truyền từ component này qua component khác.



Hình 2.8: Kiến trúc của Redux

Redux có 3 thành phần là Action, Store và Reducer.

Action đơn giản là các sự kiện, mô tả những gì xảy ra như là cách mà chúng ta gửi dữ liệu từ ứng dụng đến Redux store, dữ liệu có thể đến từ sự tương tác của user và ứng dụng, API call hoặc khi submit một form, ... Tuy nhiên action lại không chỉ rõ phần state nào thay đổi, việc này do Reducer đảm nhiệm. Reducer nhận vào một state cũ và action được gửi lên sau đó trả về một state mới.

```
1 (previousState, action) => newState
```

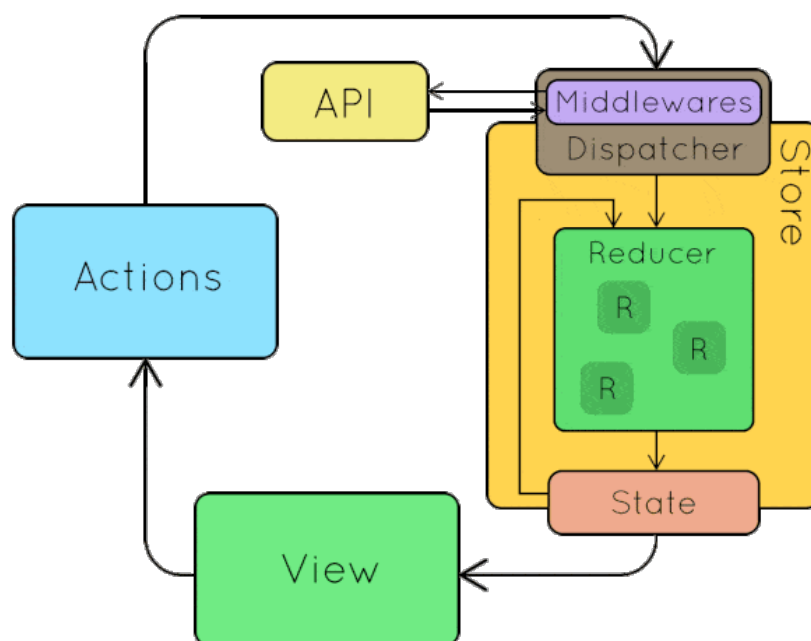
Những state này được lưu như những đối tượng (objects) và chúng định rõ cách state của một ứng dụng thay đổi trong việc phản hồi một action gửi đến store. Store là nơi lưu lại các state của ứng dụng và nó là duy nhất trong bất kì một ứng dụng Redux nào.

2.2.2.2.3 Middleware

Một ứng dụng thực tế đòi hỏi có những thao tác xử lý cần thời gian để phản hồi (các thao tác bất đồng bộ lấy dữ liệu từ api hay các thao tác đọc ghi file hay

đọc cookie từ trình duyệt, ...), các thao tác như vậy gọi là side effect. Để giải quyết được các side effect này, trong Redux ta cần thực hiện nó ở middleware.

Trong Redux, Middleware cho phép chúng ta can thiệp vào giữa thời điểm dispatch một action và thời điểm action đó đến được reducer. Kiến trúc của Redux đầy đủ khi có middleware như hình dưới đây.



Hình 2.9: Kiến trúc của Redux với Middleware

Ta có thể tự viết một middleware hoặc có thể dùng những thư viện middleware được xây dựng sẵn. Hiện tại có một vài thư viện middleware cho Redux, ví dụ như `redux-thunk`, `redux-saga`, `redux-observable`, ... mỗi thư viện có phương pháp giải quyết vấn đề side-effect riêng. Trong project này sử dụng `redux-saga` để xử lý các side-effect.

2.2.2.2.4 Redux-saga

Redux-saga là một thư viện hỗ trợ việc xử lý side-effect trong ứng dụng React/Redux (ví dụ như xử lý bất đồng bộ khi load dữ liệu,...) và làm cho các ứng dụng này trở nên đơn giản hơn. Bằng cách sử dụng Generator Function, `redux-saga` giúp ta viết code bất đồng bộ (async code) nhìn giống như là đồng bộ (synchronos).

Generator Function là function có khả năng hoãn lại quá trình thực thi mà vẫn giữ nguyên được ngữ cảnh của function. Khác với function bình thường là thực thi và trả về kết quả, thì Generator function có thể thực thi, tạm dừng trả về kết quả

và thực thi tiếp (bằng cách sử dụng từ khóa **yield**). Nếu như function bình thường khi được gọi sẽ thực thi hết tất cả các câu lệnh trong hàm thì Generator function có khả năng tạm ngưng trước khi hàm kết thúc và có thể tiếp tục chạy tại một thời điểm khác. Chính chức năng này giúp ta giải quyết được vấn đề bất đồng bộ, hàm sẽ dừng và đợi async chạy xong rồi tiếp tục thực thi.

Nguyên lý hoạt động của Redux-saga:

Redux-saga cung cấp các hàm helper effect, các hàm này sẽ trả về một effect object chứa các thông tin chỉ dẫn middleware của Redux có thể thực hiện tiếp các hành động khác. Các hàm helper effect sẽ được thực thi trong các generator function. Ví dụ một số helper effect trong Redux-saga:

- *takeEvery()*: thực thi và trả về kết quả của một action được gọi
- *takeLastest()*: nếu ta thực hiện một loạt các actions, nó sẽ chỉ thực thi và trả về kết quả của action cuối cùng.
- *put()*: dispatch một action.
- *call()*: gọi một function. Nếu nó trả về một Promise, sẽ tạm dừng saga cho đến khi Promise được giải quyết.

Ví dụ sử dụng helper effect trong Redux-saga:

```
1 // execute fetchPersonListSaga
2 // when action FETCH_PERSON_LIST is dispatched
3 yield takeEvery(FETCH_PERSON_LIST, fetchPersonListSaga);
4
5 // dispatch action pushSuccessNotification
6 yield put(pushSuccessNotification(sequence, "Saved"));
```

Câu lệnh 2.1: Sử dụng Redux-Saga

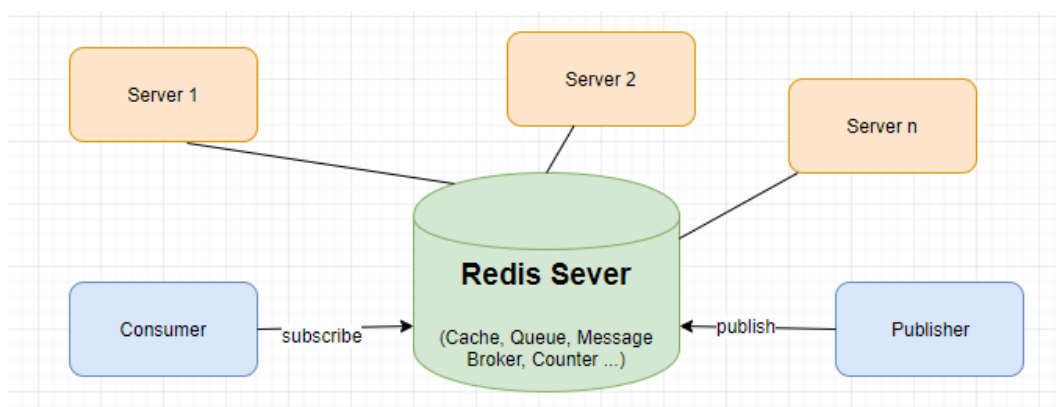
2.2.3 Công nghệ lưu trữ - Redis

Redis [5] là viết tắt của Remote Dictionary Server (máy chủ từ điển từ xa), lưu trữ dữ liệu dưới dạng KEY-VALUE trong bộ nhớ. Là phần mềm mã nguồn mở có tốc độ truy cập nhanh để dùng làm cơ sở dữ liệu đơn giản, bộ nhớ đệm (cache), trình chuyển tiếp (broker) tin nhắn hoặc được sử dụng làm danh sách tác vụ chờ xử lý (queue).

Redis hiện cung cấp thời gian phản hồi ở tốc độ chưa đến một mili giây, giúp thực hiện hàng triệu yêu cầu mỗi giây cho các ứng dụng thời gian thực trong lĩnh

vực Trò chơi, Quảng cáo, Dịch vụ tài chính, Chăm sóc sức khỏe, IoT, ... Cụ thể, Redis thường được chọn sử dụng cho các hoạt động lưu trữ bộ nhớ đệm, quản lý phiên, trò chơi, bảng xếp hạng, phân tích thời gian thực, dữ liệu không gian địa lý, ứng dụng đặt xe, trò chuyện / nhắn tin, phát trực tiếp nội dung đa phương tiện, ...

Redis là một cơ sở dữ liệu NoSQL. NoSQL là một dạng cơ sở dữ liệu phi quan hệ, sử dụng nhiều loại mô hình dữ liệu đa dạng để truy cập và quản lý dữ liệu trong bộ nhớ và tìm kiếm, NoSQL được tối ưu hóa dành riêng cho các ứng dụng yêu cầu mô hình dữ liệu linh hoạt có lượng dữ liệu lớn và độ trễ thấp, đạt được bằng cách giảm bớt một số hạn chế về tính nhất quán của dữ liệu.



Hình 2.10: Redis server


Cách thức hoạt động của Redis:

Toàn bộ dữ liệu của Redis nằm trong bộ nhớ (RAM), khác với những loại cơ sở dữ liệu thông thường khác lưu dữ liệu trên ổ đĩa hoặc ổ SSD. Phần lớn các tác vụ trên cơ sở dữ liệu truyền thống đều yêu cầu truy cập qua lại tới ổ đĩa, do vậy sẽ tốn thời gian tìm kiếm, dữ liệu của Redis nằm trên RAM nên sẽ không mất thời gian này. Do đó Redis có thể hỗ trợ thêm khá nhiều tác vụ và có thời gian phản hồi nhanh hơn. Hiệu suất của Redis rất tốt với các tác vụ đọc ghi thông thường mất chưa đầy một mili giây và hỗ trợ hàng triệu tác vụ mỗi giây.

Khác với các cơ sở dữ liệu quan hệ như MySQL hay PostgreSQL, Redis không có bảng. Redis lưu dữ liệu dưới dạng KEY-VALUE và hỗ trợ nhiều cấu trúc dữ liệu cơ bản như hash, list, set, sorted set, string, ... Bên cạnh đó Redis có 2 background threads chuyên làm nhiệm vụ định kỳ ghi dữ liệu trên đĩa cứng, cơ chế backup này giúp cho Redis có độ bảo mật và sửa lỗi cao.

Một số ứng dụng phổ biến của Redis:

- Lưu trữ bộ nhớ đệm (caching): Sử dụng bộ nhớ đệm để giảm độ trễ khi truy cập dữ liệu, tăng năng suất và giảm tải cho cơ sở dữ liệu và ứng dụng của bạn. Redis có thể phục vụ những dữ liệu thường xuyên được yêu cầu với thời gian phản hồi chưa đến một mili giây và cho phép dễ dàng thay đổi quy mô nhằm đáp ứng mức tải cao hơn mà không cần tốn kém chi phí vào back-end.
- Bảng xếp hạng game: Redis là giải pháp hay được các nhà phát triển game dùng để xây dựng bảng xếp hạng theo thời gian thực (real-time leaderboard). Sử dụng cấu trúc dữ liệu Sorted Set của Redis, cấu trúc dữ liệu này đảm bảo tính duy nhất của các thành phần trong khi vẫn duy trì danh sách được sắp xếp theo điểm số của người dùng. Danh sách cập nhật mỗi khi điểm số người dùng thay đổi.



POSITION	RACER	TIME
#1	Keanu Clark	00:14.2
#2	Nick Seiddaditi	00:15.0
#3	Shubham Kacker	00:17.0
#4	JaganY2	00:17.8
#5	Madhu Cheluvajay	00:19.0
#6	Hank	00:19.0
#7	RealPip2	00:19.0
#8	Saketh Sanganabadi	00:19.7
#9	Fredrik	00:19.4
#10	Eli Chen	00:20.0
#11	Jeremy T	00:20.2
#12	J Miller	00:20.5
#13	Griffin	00:21.0

Hình 2.11: Ứng dụng Redis trong bảng xếp hạng game

- Lưu trữ phiên (session): Các nhà phát triển ứng dụng thường sử dụng Redis để lưu trữ, quản lý phiên cho các ứng dụng quy mô Internet. Quản lý dữ liệu phiên chẳng hạn như hồ sơ người dùng, thông tin xác thực đăng nhập (token), trạng thái phiên, ...
- Trò chuyện, nhắn tin, hàng chờ xử lý tác vụ: Redis hỗ trợ Pub/Sub (là cấu trúc gửi – nhận tin nhắn mà người gửi và người nhận không biết nhau) với nhiều cấu trúc dữ liệu như list, sorted set, hash. Điều này cho phép Redis hỗ trợ những chat rooms hiệu năng cao, luồng tin nhắn theo thời gian thực.

So sánh Redis và một loại cơ sở dữ liệu quan hệ thông thường (MySQL)

	Redis	MySQL
Cấu trúc cơ sở dữ liệu	Lưu trữ dạng KEY-VALUE	Lưu trữ dạng bảng
	Lưu dữ liệu trong RAM, là máy chủ cấu trúc dữ liệu vì các key có thể chứa string, hash, list, set và sorted set	MySQL cung cấp một máy chủ cơ sở dữ liệu quan hệ SQL rất nhanh, đa luồng, đa người dùng, mạnh mẽ
Ưu điểm	<ul style="list-style-type: none"> • Dễ cài đặt, sử dụng, deploy, maintain, ... • Lưu trữ dữ liệu trong bộ nhớ nên cho hiệu năng cao và tốc độ nhanh • Mã nguồn mở, ổn định, chi phí hiệu quả • Khả năng mở rộng cao, hỗ trợ sao lưu vào đĩa cứng • Cấu trúc dữ liệu đa dạng 	<ul style="list-style-type: none"> • Cơ sở dữ liệu quan hệ mã nguồn mở được sử dụng rộng rãi • Dễ sử dụng, khả năng tương thích cao, hỗ trợ đa nền tảng, cộng đồng phát triển mạnh mẽ • Hỗ trợ index và full-text searching
Trường hợp nên sử dụng	<ul style="list-style-type: none"> • Có dữ liệu dạng KEY-VALUE • Cần lưu cache • Cần hiệu năng cao • Kích thước dữ liệu ổn định 	<ul style="list-style-type: none"> • Cần cơ sở dữ liệu quan hệ • Khi có các hoạt động phân tán • Cần bảo mật cao và hoạt động đơn giản • Không sử dụng khi dữ liệu lớn dần và không thể cache hết lên bộ nhớ
Nhược điểm	<ul style="list-style-type: none"> • Redis sử dụng RAM nên khi lượng file cache lớn sẽ dẫn đến thiếu RAM cho server • Không thể truy vấn trực tiếp các object 	<ul style="list-style-type: none"> • Nhiều vấn đề về lưu trữ procedure và trigger

Bảng 2.1: So sánh Redis và MySQL

2.2.4 Công nghệ lưu trữ - PostgreSQL

2.2.4.1 Tổng quan về PostgreSQL

PostgreSQL, còn được gọi là Postgres, là một hệ thống quản lý cơ sở dữ liệu quan hệ miễn phí mã nguồn mở chú trọng vào tính mở rộng và tính tương thích với chuẩn SQL. Ban đầu có tên là POSTGRES, có nguồn gốc từ cơ sở dữ liệu Ingress được phát triển tại

Đại học California, Berkeley. Vào năm 1996, dự án được đổi tên thành PostgreSQL. PostgreSQL được viết bằng ngôn ngữ C và có thể chạy trên nhiều hệ điều hành như Linux, macOS, Windows, ... Đồng thời hỗ trợ các tính năng tương tự như nhiều các cơ sở dữ liệu quan hệ khác. Phiên bản PostgreSQL sử dụng trong đồ án này là 11.8.

2.2.4.2 Tổng quan về Index

Index trong cơ sở dữ liệu là một cấu trúc dữ liệu giúp cải thiện tốc độ truy vấn dữ liệu trên các bảng với đánh đổi là sử dụng thêm không gian lưu trữ nhằm duy trì cấu trúc dữ liệu index cũng như làm tăng thời gian ghi dữ liệu. Index được sử dụng để nhanh chóng định vị dữ liệu mà không phải tìm duyệt qua toàn bộ các hàng trong bảng của cơ sở dữ liệu mỗi khi bảng đó được truy cập. Index có thể được tạo sử dụng một hay nhiều cột trong cùng một bảng, cung cấp cơ chế tăng tốc quá trình tra cứu ngẫu nhiên và truy cập có thứ tự các bản ghi trong cơ sở dữ liệu.

Index được chia làm hai loại kiến trúc: Clustered Index và Non-clustered Index.

Với clustered index, các hàng trong cơ sở dữ liệu được lưu trữ trên đĩa theo cùng thứ tự với index. Do đó mỗi bảng chỉ tồn tại duy nhất một clustered index. Với non-clustered index thì cấu trúc dữ liệu index là cấu trúc dữ liệu nằm ngoài bảng, các hàng trong bảng không được đảm bảo thứ tự giống như thứ tự của index. Một bảng có thể có rất nhiều non-clustered index. Thông thường với truy cập trên clustered index sẽ nhanh hơn trên non-clustered index vì dữ liệu sẽ không bị phân mảnh ở nhiều vị trí trong ổ cứng.

2.2.4.3 Index trong PostgreSQL

Trong PostgreSQL, clustered index không được hỗ trợ. Tuy nhiên PostgreSQL lại hỗ trợ nhiều kiểu index như B-Tree, Hash, GiST, SP-GiST, GIN và BRIN. Mỗi một kiểu index sử dụng một thuật toán riêng biệt để tối ưu cho nhiều loại câu truy vấn khác nhau. Mặc định khi sử dụng lệnh CREATE INDEX thì index B-Tree sẽ được tạo, index này cũng phù hợp với hầu hết các trường hợp truy vấn [3].

Index kiểu B-Tree có thể được sử dụng cho các câu truy vấn liên quan đến tìm kiếm bằng, tìm kiếm khoảng hoặc trong trường hợp dữ liệu được sắp xếp theo một trường nào đó. Trình lập kế hoạch cho các truy vấn của PostgreSQL sẽ xem xét sử dụng index B-Tree bất kì khi nào trường mà có index nằm trong một trong các phép toán so sánh như $<$ $<=$ $=$ $>=$ $>$.

Các toán tử tương đương hoặc tổ hợp của chúng, như là phép toán BETWEEN và IN, cũng có thể sử dụng được index B-Tree. Ngoài ra các phép toán liên quan đến NULL như IS NULL hay IS NOT NULL cũng có thể được tăng tốc bằng index B-Tree.

Trình tối ưu cũng có thể sử dụng index B-Tree cho các câu truy vấn sử dụng toán tử pattern matching như LIKE và nếu mà pattern được sử dụng bắt đầu là một xâu hằng số như 'foo%' nhưng không phải là '%bar'.

B-Tree có thể cũng được sử dụng để lấy dữ liệu được sắp xếp theo thứ tự của cột được index.

Index kiểu Hash chỉ có thể được sử dụng trong các câu truy vấn sử dụng toán tử so sánh bằng. Để tạo index Hash trong PostgreSQL, ta sử dụng:

```
1 CREATE INDEX name ON table USING HASH (column);
```

Câu lệnh 2.2: Tạo index sử dụng Hash

2.2.4.4 Full Text Search trên PostgreSQL

2.2.4.4.1 Cơ bản về Full Text Search

Các toán tử tìm kiếm text trên các CSDL quan hệ đã tồn tại từ rất lâu như LIKE hay ILIKE. Nhưng những toán tử đó thiếu đi những tính chất mà cần có trong các hệ thống thông tin hiện đại như:

- Hỗ trợ ngôn ngữ, thậm chí với cả tiếng Anh. Các toán tử trên không phù hợp cho việc xử lý những từ gần giống nghĩa hoặc được suy diễn, như satisfy và satisfies trong tiếng Anh. Có thể dùng toán tử logic OR để tìm kiếm cả 2 nhưng phải liệt kê rất nhiều trường hợp và một số từ có thể có rất nhiều từ được suy diễn hoặc gần nghĩa.
- Không hỗ trợ xếp hạng (ranking) trên tập kết quả, dẫn đến tìm kiếm không hiệu quả khi mà có thể có tới rất nhiều document được tìm thấy trong một câu truy vấn.
- Các phép toán đó thường chậm vì không hỗ trợ đánh chỉ số, mỗi một câu truy vấn phải duyệt qua toàn bộ các document.

PostgreSQL hỗ trợ tiên xử lý các document và đánh chỉ số để tăng tốc các câu truy vấn. Tiên xử lý document bao gồm việc:

- Phân tích document thành các token.
- Chuyển các token thành các lexeme.
- Lưu trữ các dữ liệu sau khi tiên xử lý document để tối ưu cho truy vấn.

Một document trong PostgreSQL thông thường là một cột chứa xâu của bảng CSDL, hoặc có thể là một tổ hợp (thông qua phép ghép nối xâu) của những trường đó trong một bảng hay trong nhiều bảng hoặc có thể được truyền từ ngoài vào.

Một token là một xâu biểu diễn một từ được phân tích ra từ document.

Một lexeme cũng là một chuỗi, tương tự như token, nhưng đã được chuẩn hóa (normalize) sao cho những dạng khác nhau của một từ trở thành giống nhau. Việc chuẩn hóa luôn luôn thực hiện chuyển tất cả các ký tự thành lower-case, và thông thường có cắt bỏ một vài các hậu tố như e hoặc es trong tiếng Anh. Đồng thời ở bước chuyển từ token thành còng lexeme cũng thông thường loại bỏ những stop word, là những từ rất phổ biến trở thành vô nghĩa khi tìm kiếm, PostgreSQL sử dụng Dictionary để thực thi bước này.

Dictionary trong PostgreSQL cho phép hiệu chỉnh quá trình chuẩn hóa thành lexeme, bao gồm:

- Định nghĩa các stop word được loại bỏ.
- Ánh xạ các từ đồng nghĩa thành một từ.
- Ánh xạ cụm từ thành một từ.
- Ánh xạ các biến thể của một từ thành dạng chuẩn.

PostgreSQL sử dụng kiểu dữ liệu `tsvector` để lưu trữ dữ liệu sau khi tiền xử lý document, đồng thời sử dụng kiểu `tsquery` để biểu diễn các câu truy vấn.

Để chuyển từ một `xâu` sang kiểu `tsvector` ta sử dụng hàm `to_tsvector`. Còn để chuyển một `xâu` bất kì sang một `tsquery`, ta sử dụng `plainto_tsquery`.

Để thực hiện Full Text Search một tsvector với một tsquery, ta sử dụng toán tử @@.

2.2.4.4.2 Full Text Search cho tiếng Việt

Để thực hiện Full Text Search với tiếng Việt, trong đồ án này sử dụng một hàm được định nghĩa bằng SQL như sau:

[illegible]

Khi đó, thay vì sử dụng trực tiếp `to_tsvector` và `plainto_tsquery`, ta sử dụng `to_tsvector(vn_unaccent(text))` và `plainto_tsquery(vn_unaccent(text))`.

2.2.4.4.3 Đánh chỉ số cho Full Text Search

Để sử dụng Index cho Full Text Search, trước tiên ta cần tạo một cột trong CSDL lưu trữ kiểu dữ liệu tsvector. Có hai loại Index hay được sử dụng cho Full Text Search là GIN và GiST.

Để Index sử dụng GIN, cú pháp như sau:

```
1 CREATE INDEX idx_textsearch ON sometable USING GIN (tsvector_column
   );
```

Câu lệnh 2.3: Tạo index sử dụng GIN

Còn nếu sử dụng GiST:

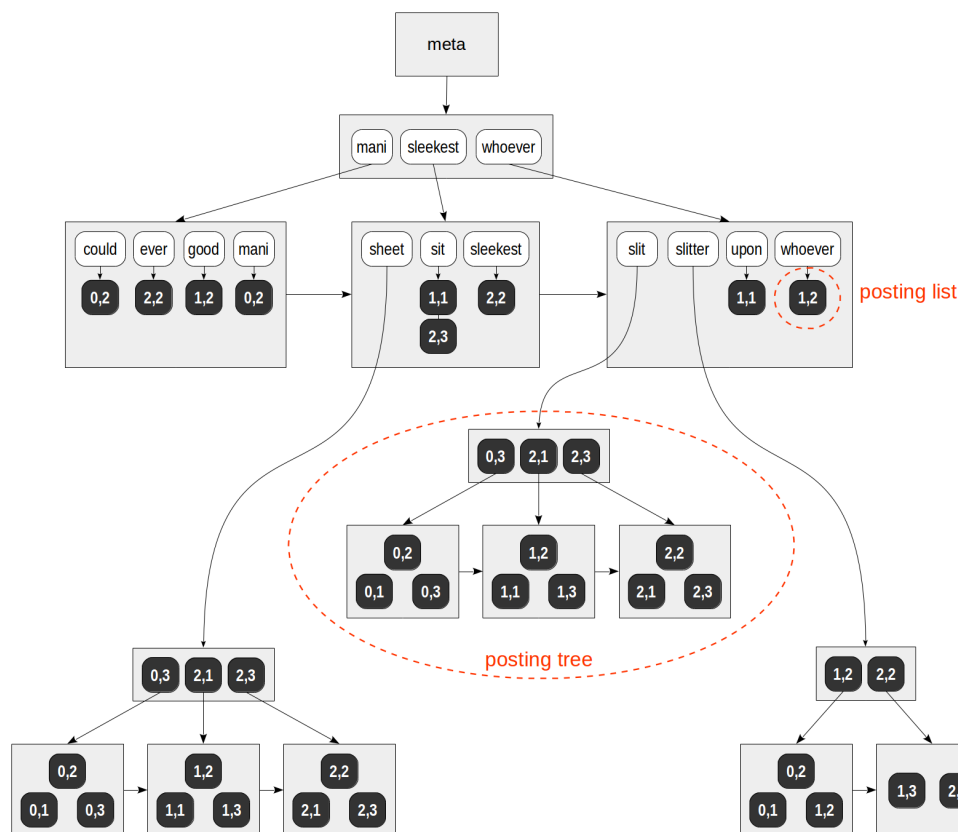
```
1 CREATE INDEX idx_textsearch ON sometable USING GIST (
   tsvector_column);
```

Câu lệnh 2.4: Tạo index sử dụng GiST

Khi đó toán tử @@ có thể sử dụng Index để tăng tốc câu truy vấn. Trong PostgreSQL, Index Full Text Search thông thường sử dụng GIN hơn sử dụng GiST.

Thông thường sẽ sử dụng trigger để cập nhật các trường tsvector trong CSDL khi chèn hoặc chỉnh sửa.

2.2.4.4.4 GIN



Hình 2.12: Cấu trúc của GIN

GIN là viết tắt của Generalized Inverted Index. GIN được thiết kế cho việc đánh chỉ số những giá trị tổng hợp (composite value), và cho phép tìm kiếm cho các phần tử xuất hiện trong các composite value.

Nếu quy ước một composite value là một item, còn phần tử trong composite value là một key thì GIN luôn luôn lưu trữ và tìm kiếm trên key chứ không phải trên item. Một GIN index lưu trữ tập các cặp (key, posting list) trong đó posting list là tập các ID của các hàng trong bảng của CSDL. Một ID có thể được nằm trong nhiều các posting list vì một item có thể có nhiều các key. Mỗi một key được lưu trữ một lần duy nhất, do đó nên GIN rất phù hợp trong trường hợp một key xuất hiện nhiều lần. posting list được lưu với thứ tự từ nhỏ đến lớn các ID của các hàng. Nếu posting list quá lớn sẽ được chuyển thành posting tree, cấu trúc tương tự như B-Tree.

2.2.5 Công nghệ back-end

2.2.5.1 Giới thiệu về ngôn ngữ Go

Go là ngôn ngữ biên dịch kiểu tĩnh được tạo ra tại Google. Cú pháp của Go tương tự như của ngôn ngữ C, đồng thời cũng bị ảnh hưởng lớn bởi ngôn ngữ này. Ngôn ngữ Go cũng thông thường được gọi với tên là Golang. Sau đây là một số đặc điểm của ngôn ngữ Go:

- Cú pháp và môi trường lập trình của Go có tính tương đồng với các ngôn ngữ lập trình kiểu động (dynamic language) như:
 - Rút gọn khai báo biến, không cần phải khai báo kiểu thông qua cơ chế type inference.
 - Biên dịch nhanh.
 - Quản lý các gói thư viện từ xa và các package có document truy cập online.
- Sử dụng cách tiếp cận đặc biệt với một số vấn đề:
 - Xây dựng sẵn trong ngôn ngữ một số cơ chế cho lập trình đồng bộ như: tiến trình nhẹ (light-weight process, hay còn gọi là goroutine), channel và câu lệnh select.
 - Toolchain mặc định khi biên dịch sinh ra file thực thi sẽ không chứa phụ thuộc ngoài.
- Ngôn ngữ được thiết kế đủ đơn giản để lập trình viên có thể dễ dàng hiểu và ghi nhớ.

2.2.5.2 Goroutine, Chanel và khối lệnh Select

Ngôn ngữ Go được xây dựng sẵn trong ngôn ngữ những cơ chế hỗ trợ việc lập trình đồng bộ. Đồng bộ ở đây không chỉ là song song ở mức CPU mà đồng thời cả việc sử dụng Asynchronous I/O cho phép các câu lệnh như truy cập database hay đọc gói tin từ mạng chạy trong khi tiến trình vẫn được sử dụng cho công việc khác. Kỹ thuật này phổ biến trong các server mà sử dụng event-based.

Trung tâm của xây dựng chương trình đồng bộ trong Go là tiến trình nhẹ (light-weight process) gọi là Goroutine. Một lời gọi hàm mà được đặt sau từ khóa go sẽ bắt đầu hàm đó trong một Goroutine mới. Trong đặc tả ngôn ngữ (Language Specification) không chỉ định Goroutine được cài đặt như nào nhưng với cài đặt hiện tại thì các Goroutine sẽ được phân bổ vào một tập nhỏ các luồng ở mức hệ điều

hành, tương tự như cơ chế phân phát tiến trình trong ngôn ngữ Erlang. Khi chương trình Go mới bắt đầu sẽ chứa duy nhất một goroutine gọi là main goroutine.

Chương trình viết bằng Go sẽ thường sử dụng Channel, một cơ chế để gửi các thông điệp (Message) giữa các goroutine. Channel trong Go có thể có chứa Buffer hoặc không. Nếu không chứa Buffer thì Channel mỗi một thời điểm chỉ chứa tối đa một Message, các lời gọi gửi message vào channel tiếp theo đó sẽ bị block. Nếu chứa Buffer thì kích thước của buffer cũng bị giới hạn, các message được lưu trữ và truy xuất theo cơ chế FIFO. Khi đọc từ một channel mà channel chưa chứa message nào cả thì goroutine đọc từ channel đó sẽ bị block.

Channel trong Go có chứa kiểu tĩnh, nghĩa là một channel có kiểu là chan T sẽ chỉ có thể gửi và nhận message thuộc kiểu T. Go có chứa cú pháp đặc biệt để tương tác với channel:

- `x <- ch` để đọc từ channel ch vào biến x.
- `ch <- x` để gửi message x vào channel ch.

Trong trường hợp tương tác với nhiều lệnh nhận hoặc gửi message với các channel, chương trình Go có thể sử dụng khối lệnh select để lựa chọn lệnh nhận/gửi đầu tiên mà kết thúc block trên các channel. Cú pháp của khối lệnh select trong Go tương tự với switch nhưng khác là mỗi một nhãn case trong khối lệnh select sẽ là một lệnh tương tác với channel.

2.2.5.3 Các công cụ tích hợp

Bản cài đặt bộ công cụ của Go bao gồm nhiều các công cụ liên quan đến xây dựng, kiểm thử và phân tích code, gồm:

- **go build**, để xuất file nhị phân từ các file mã nguồn.
- **go test**, dùng để chạy các unit test và các benchmark.
- **go fmt**, dùng để format code.
- **go get**, tải xuống và cài đặt các thư viện và các file thực thi đi kèm.
- **go vet**, một static analyzer cho việc tìm kiếm các vị trí có thể có lỗi trong code.
- **go run**, một shortcut cho phép biên dịch và chạy chương trình.
- **go doc**, dùng để hiển thị document của thư viện.

- **go mod**, dùng để quản lý module - xuất hiện từ phiên bản Go 1.11.
- **go generate**, dùng để gọi code generator.

2.2.6 Docker

2.2.6.1 Công nghệ container

Công nghệ container được sinh ra có mục đích tương tự như công nghệ máy ảo (Virtual Machine) [8]. Điểm khác biệt lớn nhất là mỗi container sẽ không yêu cầu phải chạy một hệ điều hành đầy đủ và riêng biệt. Tất cả các container trên một máy chủ đều chia sẻ chung một hệ điều hành. Chính điểm khác biệt này làm cho container sử dụng ít tài nguyên CPU, RAM và bộ nhớ lưu trữ hơn VM rất nhiều dẫn đến tiết kiệm chi phí cho việc chạy server, vá lỗi hệ điều hành và các vấn đề bảo trì hệ thống khác.

Công nghệ Container hiện đại được bắt đầu từ hệ điều hành Linux. Một số các công nghệ của nhân Linux giúp tạo nên sự phát triển của công nghệ container bao gồm: kernel namespace, control group, union filesystem. Dù cho các công nghệ này đã tồn tại từ lâu những để sử dụng nó thì rất phức tạp và ngoài tầm với của phần lớn các công ty công nghệ. Sự thay đổi bắt đầu khi Docker được tạo ra giúp đơn giản hóa rất nhiều quá trình tạo và quản lý các container.

Docker ban đầu được tạo nên chỉ để chạy trên môi trường Linux. Tuy nhiên, trong một vài năm gần đây, Microsoft đã phát triển công nghệ container trên hệ điều hành Windows 10 và Windows Server từ đó giúp việc sử dụng Docker trên Windows tương tự như trên Linux. Nhưng bởi vì container sẽ chia sẻ hệ điều hành với hệ điều hành mà Docker chạy trên (được gọi là host), từ đó dẫn đến một container dùng Linux sẽ không thể chạy trên Windows và ngược lại.

2.2.6.2 Sơ lược về Docker

Docker được bắt đầu phát triển bởi Docker, Inc vào năm 2013, bao gồm các thành phần:

- **Software**: gồm một chương trình chạy ngầm (daemon) gọi là dockerd, là chương trình quản lý các container và xử lý các container object. Daemon dockerd lắng nghe các yêu cầu từ bên ngoài gửi đến thông qua Docker Engine API. Đi kèm là một chương trình client, gọi là docker, cung cấp một giao diện dòng lệnh giúp tương tác với daemon dockerd.
- **Object**: các Docker Object là các thực thể cấu thành lên một ứng dụng chạy trên Docker. Các Docker object được chia thành ba loại là image, container

và service.

- Một Docker image là một tập các file chỉ đọc dùng làm template để tạo nên các container. Các image được dùng để lưu trữ và triển khai các ứng dụng.
- Một Docker container là một môi trường được đóng gói và chuẩn hóa để chạy ứng dụng.
- Một Docker service là một thực thể dùng để chạy nhiều container trên nhiều máy trong một cluster. Tập các máy chạy Docker trong một cluster gọi là một Swarm.

2.2.6.3 Dockerfile

Để xây dựng các image cho việc đóng gói ứng dụng, ta có thể chạy container, cài đặt các file cần thiết và sử dụng lệnh **docker commit** để lưu container thành image. Tuy nhiên thông thường việc tạo image thông qua việc sử dụng Dockerfile.

Dockerfile là một file văn bản có chứa những instruction(chỉ thị) để xây dựng một image. Trong đó có các lệnh như: chỉ định image gốc nào mà sẽ được xây dựng lên, chỉ định lệnh nào được chạy khi bắt đầu một container, chỉ định những file hay thư mục nào được sao chép từ host vào container, Để xây dựng lên image bằng Dockerfile, ta sử dụng docker build.

Ví dụ trong đồ án này sử dụng Dockerfile sau để xây dựng lên Webserver viết bằng Go:

```
1 FROM golang:1.14.3-alpine3.11 as builder
2 WORKDIR /go/baseweb/
3 COPY . .
4 RUN go build
5
6 FROM alpine:3.11
7 WORKDIR /go/
8 COPY --from=builder /go/baseweb/baseweb .
9 EXPOSE 8080
10 CMD ["/baseweb"]
```

Câu lệnh 2.5: Xây dựng image của back-end server

2.2.6.4 Registry & Docker Hub

Registry là một kho lưu trữ các image sử dụng để tải xuống image và tạo container từ đó. Registry có thể là một private server trong một cluster, hay một

dịch vụ trên cloud (như Azure Container Registry) hay phổ biến và mặc định của Docker là Docker Hub.

Để đăng nhập vào Docker Hub bằng Docker client, ta sử dụng lệnh **docker login** và để đẩy image lên registry, ta sử dụng **docker push**.

Docker Hub hỗ trợ cả public repository và private tương tự như GitHub.

2.2.7 Kubernetes

2.2.7.1 Sơ lược về Kubernetes

Kubernetes (viết tắt là K8s) là một phần mềm mã nguồn mở cho việc tự động hóa quá trình triển khai (deploy), mở rộng (scale) và quản lý các ứng dụng đã được đóng gói để chạy trên container. Ban đầu được thiết kế bởi Google và hiện tại được quản lý bởi Cloud Native Computing Foundation. K8s hoạt động với nhiều công cụ quản lý container, phổ biến nhất là với Docker. Rất nhiều các nhà cung cấp dịch vụ Cloud như AWS, Google Cloud và Azure cho phép các ứng dụng triển khai bằng Kubernetes có thể chạy trực tiếp từ dịch vụ Kubernetes mà không phải tự thiết lập cluster sao cho phù hợp.

2.2.7.2 Các object trong Kubernetes

Kubernetes định nghĩa một tập các khối căn bản khi hoạt động cùng nhau sẽ cung cấp cơ chế giúp triển khai, duy trì và mở rộng ứng dụng dựa trên CPU, bộ nhớ hoặc là một metric bất kỳ. Các object chính trong Kubernetes bao gồm:

- Pod: là một tập các container mà luôn được đảm bảo sẽ chạy trên cùng một host và cho phép chia sẻ tài nguyên với nhau [7]. Pod là đơn vị cho thuật toán phân bổ container trong Kubernetes. Mỗi pod được gán một địa chỉ IP duy nhất trong cluster. Các container trong cùng một pod có thể tham chiếu lẫn nhau sử dụng địa chỉ localhost, những các container ở khác pod sẽ không thể truy cập trực tiếp với nhau thông qua localhost, khi đó phải sử dụng địa chỉ IP mà mỗi Pod được gán. Tuy nhiên, ứng dụng chạy trên container nằm trong pod không nên sử dụng địa chỉ IP trực tiếp của Pod bởi vì địa chỉ IP của Pod là không tĩnh, có thể bị thay đổi khi một Pod khởi động lại.
- Replica Set: là một cơ chế với mục tiêu đảm bảo số lượng ổn định của các Pod trong cluster tại bất kỳ thời điểm nào. Thông thường được sử dụng để chỉ định số lượng Pod giống nhau sẽ được chạy trong cluster.
- Service: Một Service trong Kubernetes là cơ chế nhóm các pod hoạt động cùng với nhau lại, cung cấp cơ chế Load Balance theo round-robin cho phép

container nằm trong Pod ở bên ngoài service có thể truy cập các Pod ở bên trong service thông qua địa chỉ IP của Service thay vì địa chỉ IP trực tiếp của các Pod. Địa chỉ IP này có thể được lấy thông qua các biến môi trường hoặc thông qua một server DNS trong cluster. Service mặc định là public ở trong cluster. Tuy nhiên nếu muốn truy cập service từ bên ngoài cluster ta phải thay đổi kiểu của Service thành **type: LoadBalancer**. Khi đó Service sẽ chứa địa chỉ IP cho phép truy cập từ bên ngoài cluster.

- Deployment: là một cơ chế được xây dựng trên Replica Set cho phép thay đổi cấu hình của các Pod hoặc của Replica Set. Những thay đổi này được mô tả bởi người dùng bởi việc chỉ định cấu hình mong muốn, Deployment sẽ cập nhật các Pod và các Replica Set tương ứng để trạng thái hệ thống đạt được cấu hình mong muốn đó.

Ngoài ra còn các object khác như ReplicationController, StatefulSet, Volume, Endpoint, ...

2.2.8 Microsoft Azure

2.2.8.1 Sơ lược về Microsoft Azure

Microsoft Azure, thông thường được gọi là Azure, là một nền tảng điện toán đám mây (cloud computing) được tạo bởi Microsoft cho việc xây dựng, kiểm thử và quản lý các ứng dụng và dịch vụ chạy trên các trung tâm dữ liệu do Microsoft quản lý. Azure cung cấp nhiều các dịch vụ điện toán đám mây ở nhiều mức độ như Software as a Service (SaaS), Platform as a Service (PaaS) và Infrastructure as a Service (IaaS).

Azure được phát hành vào năm 2010 với tên là Windows Azure sau đó được đổi tên thành Microsoft Azure vào năm 2014.

2.2.8.2 Các dịch vụ của Microsoft Azure

Azure cung cấp rất nhiều các dịch vụ, trong đó phổ biến là:

- Máy ảo (virtual machine): là một Infrastructure as a Service (IaaS) cho phép người dùng có thể chạy bất kỳ một máy ảo Windows hay Linux đi kèm với các gói phần mềm phổ biến.
- App Service: là một Platform as a Service (PaaS) cho phép người dùng dễ dàng xuất bản và quản lý website.
- Các cơ sở dữ liệu như SQL Server, PostgreSQL, Redis, MySQL, MariaDB,...

- Container Instance: cho phép chạy và quản lý các container Docker độc lập mà không cần cơ chế điều phối như Kubernetes.
- Kubernetes Service: cho phép chạy và quản lý một Kubernetes cluster.

Và nhiều các dịch vụ khác.

2.2.8.3 Azure Kubernetes Service

Azure Kubernetes Service (AKS) là một dịch vụ của Azure cho phép triển khai một cluster Kubernetes mà không phải quan tâm đến việc cài đặt một cluster Kubernetes ra sao. Để tương tác với AKS, ta có thể sử dụng Azure portal - một giao diện web, hoặc sử dụng Azure CLI với chương trình có tên là az để cài đặt và chạy chương trình dòng lệnh kubectl (chương trình quản lý cluster Kubernetes) cho phép quản lý AKS từ xa.

CHƯƠNG 3. PHÂN TÍCH THIẾT KẾ HỆ THỐNG

3.1 Tổng quan các chức năng

3.1.1 Biểu đồ use case tổng quan



Hình 3.1: Biểu đồ use case tổng quan

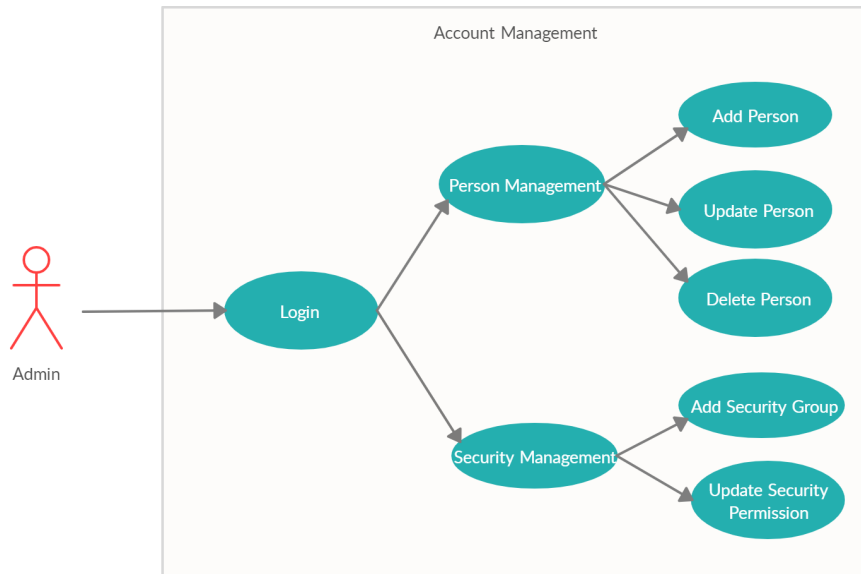
Hai tác nhân chính trong biểu đồ use case tổng quan là người quản lý (manager) và nhân viên bán hàng (salesman). Người quản lý ở đây là từ dùng chung, đại diện cho những người quản lý nghiệp vụ riêng (quản lý sản phẩm, quản lý hàng tồn kho, quản lý kho, quản lý giao dịch, quản lý tuyến bán hàng). Người quản lý sau khi đăng nhập vào hệ thống có thể thực hiện các nghiệp vụ quản lý như thêm / sửa / xóa sản phẩm, người dùng, ... Nhân viên bán hàng sau khi đăng nhập có thể xem được lịch trình di chuyển của mình, lên hóa đơn mua hàng, check-in.

3.1.2 Biểu đồ use case phân rã các chức năng của hệ thống

Trong phần này, chúng tôi sẽ trình bày các ca sử dụng chính trong hệ thống, đưa ra biểu đồ use case và biểu đồ hoạt động chỉ cách thức tương tác của người dùng với giao diện hệ thống. Do nhiều thao tác có tính chất tương đồng nên chúng tôi sẽ đưa ra các biểu đồ hoạt động điển hình, các thao tác khác được thực hiện tương tự.

3.1.2.1 Quản lý người dùng và phân quyền

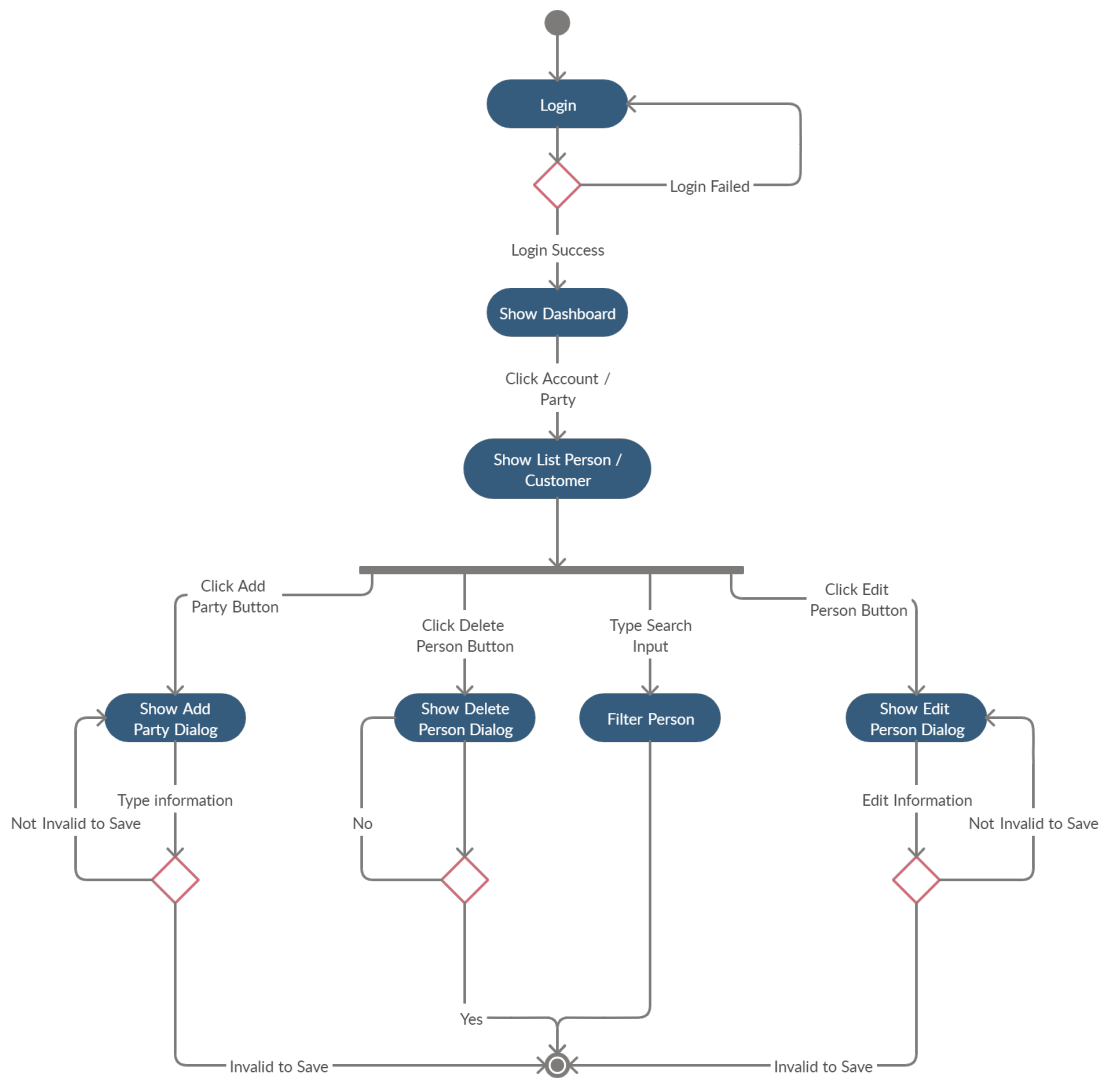
Biểu đồ use case cho ca sử dụng quản lý người dùng



Hình 3.2: Use-case quản lý người dùng

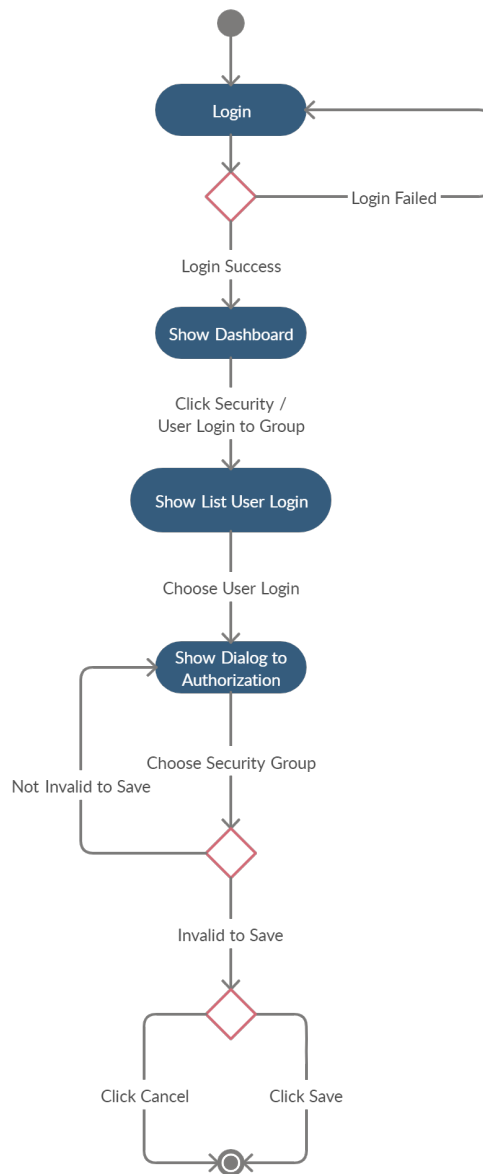
Người quản trị (admin) có thể quản lý các người dùng trong hệ thống, quản lý quyền hạn của người dùng. Có thể thêm / sửa / xóa người dùng, khách hàng, thêm / sửa quyền cho các người quản lý cấp dưới.

Biểu đồ hoạt động cho các thao tác trong quản lý người dùng:



Hình 3.3: Biểu đồ hoạt động ca sử dụng quản lý người dùng

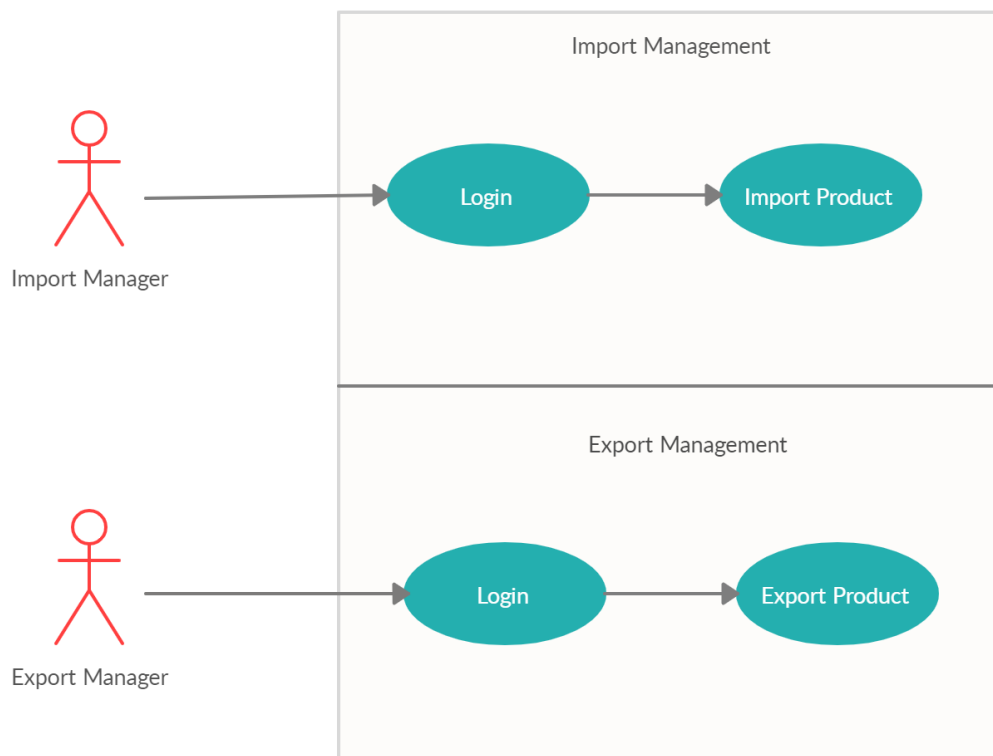
Biểu đồ hoạt động cho thao tác phân quyền:



Hình 3.4: Biểu đồ hoạt động cho thao tác phân quyền

3.1.2.2 Quản lý nhập – xuất hàng

Biểu đồ use case cho ca sử dụng quản lý nhập – xuất hàng:



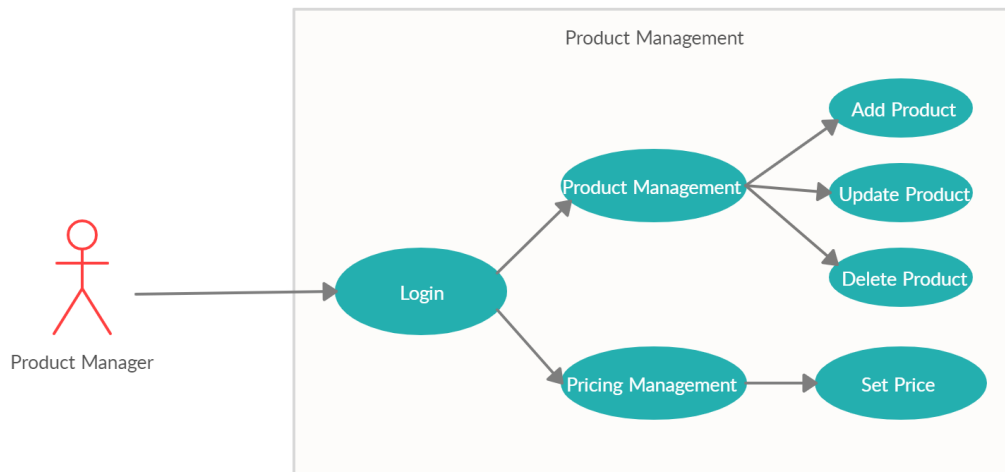
Hình 3.5: Use case quản lý xuất – nhập hàng

Người quản lý nhập hàng (Import Manager) có nhiệm vụ quản lý quy trình nhập hàng hóa từ nguồn bên ngoài (một doanh nghiệp sản xuất nào đó hoặc một tập đoàn bán lẻ khác, ...) vào kho của doanh nghiệp, tập đoàn mình.

Người quản lý xuất hàng (Export Manager) có nhiệm vụ xuất hàng từ kho của doanh nghiệp, tập đoàn sang các cửa hàng bán lẻ sau khi hóa đơn xuất hàng đã được duyệt.

3.1.2.3 Quản lý sản phẩm

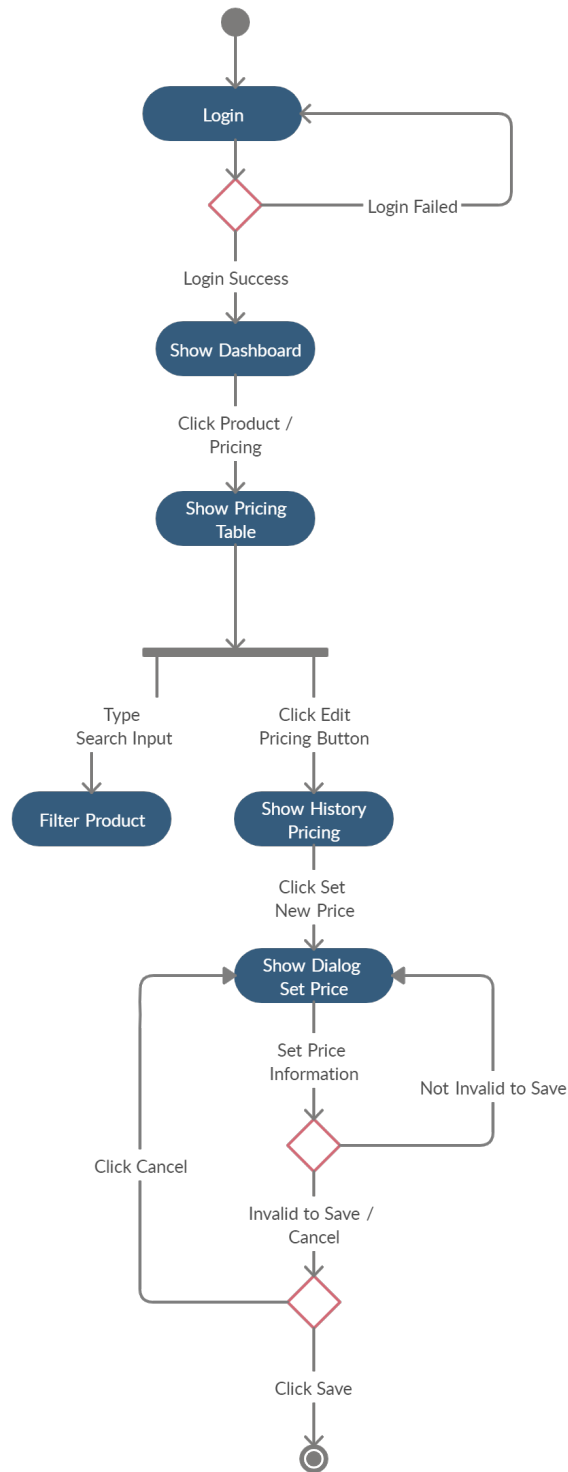
Biểu đồ use case cho ca sử dụng quản lý sản phẩm:



Hình 3.6: Use-case quản lý sản phẩm

Người quản lý sản phẩm (Product Manager) quản lý các sản phẩm đã được nhập vào và quản lý giá của sản phẩm.

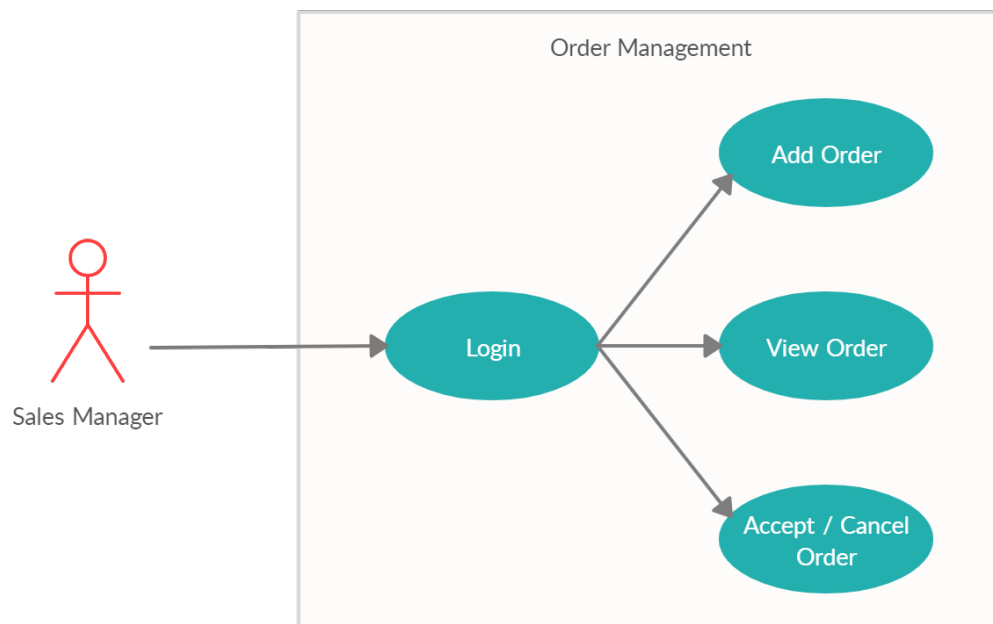
Biểu đồ hoạt động cho thao tác gán giá cho sản phẩm:



Hình 3.7: Biểu đồ hoạt động cho thao tác gán giá của sản phẩm

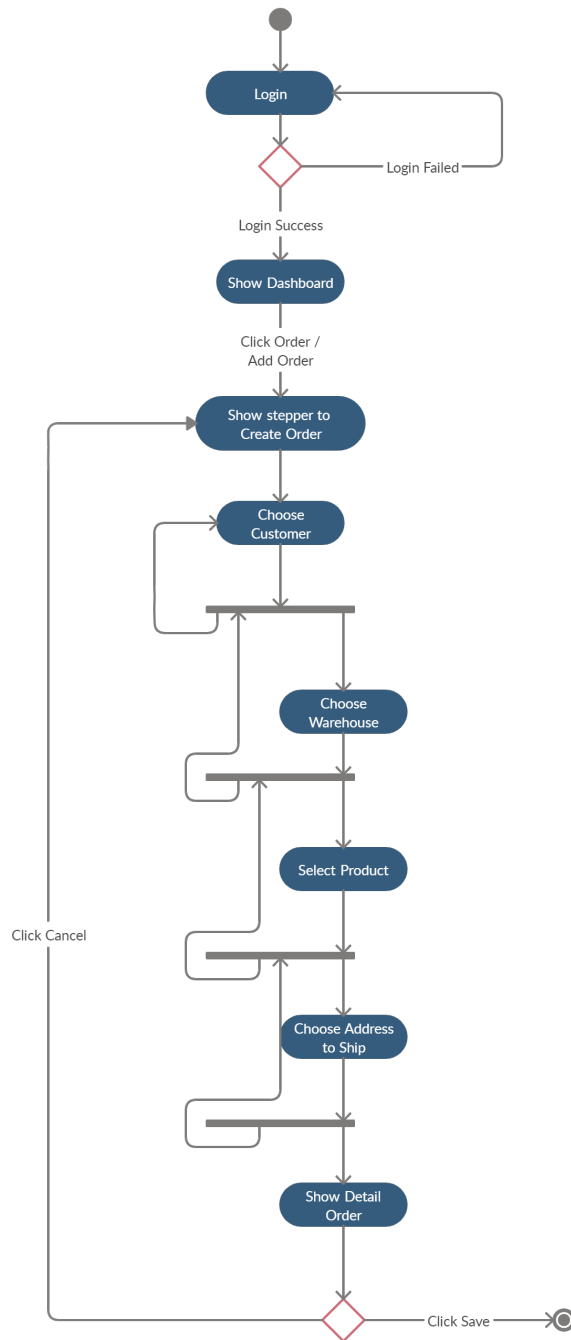
3.1.2.4 Quản lý đơn hàng

Biểu đồ use case cho ca sử dụng quản lý đơn hàng:



Hình 3.8: Use-case quản lý đơn hàng

Biểu đồ hoạt động cho thao tác thêm đơn hàng:

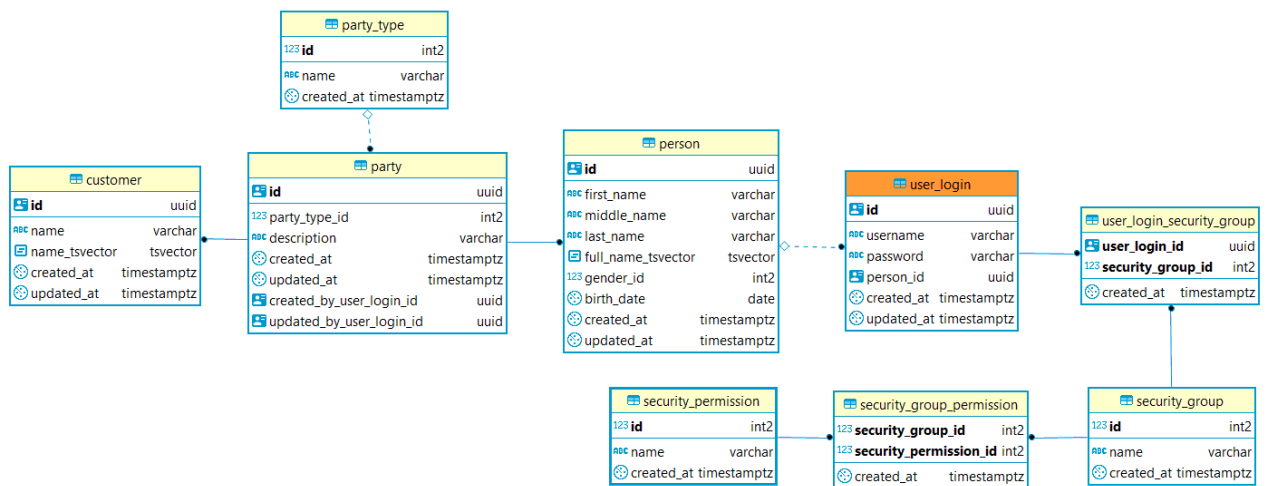


Hình 3.9: Biểu đồ hoạt động cho thao tác thêm đơn hàng

3.2 Thiết kế cơ sở dữ liệu

Phần này trình bày chi tiết thiết kế các bảng của cơ sở dữ liệu sử dụng PostgreSQL phục vụ cho việc triển khai các use case đã trình bày ở trên.

Nhóm các bảng phục vụ chức năng quản lý phân quyền, quản lý người dùng được trình bày qua sơ đồ thực thể liên kết (Hình 3.10) và bảng thông tin (Bảng 3.1).

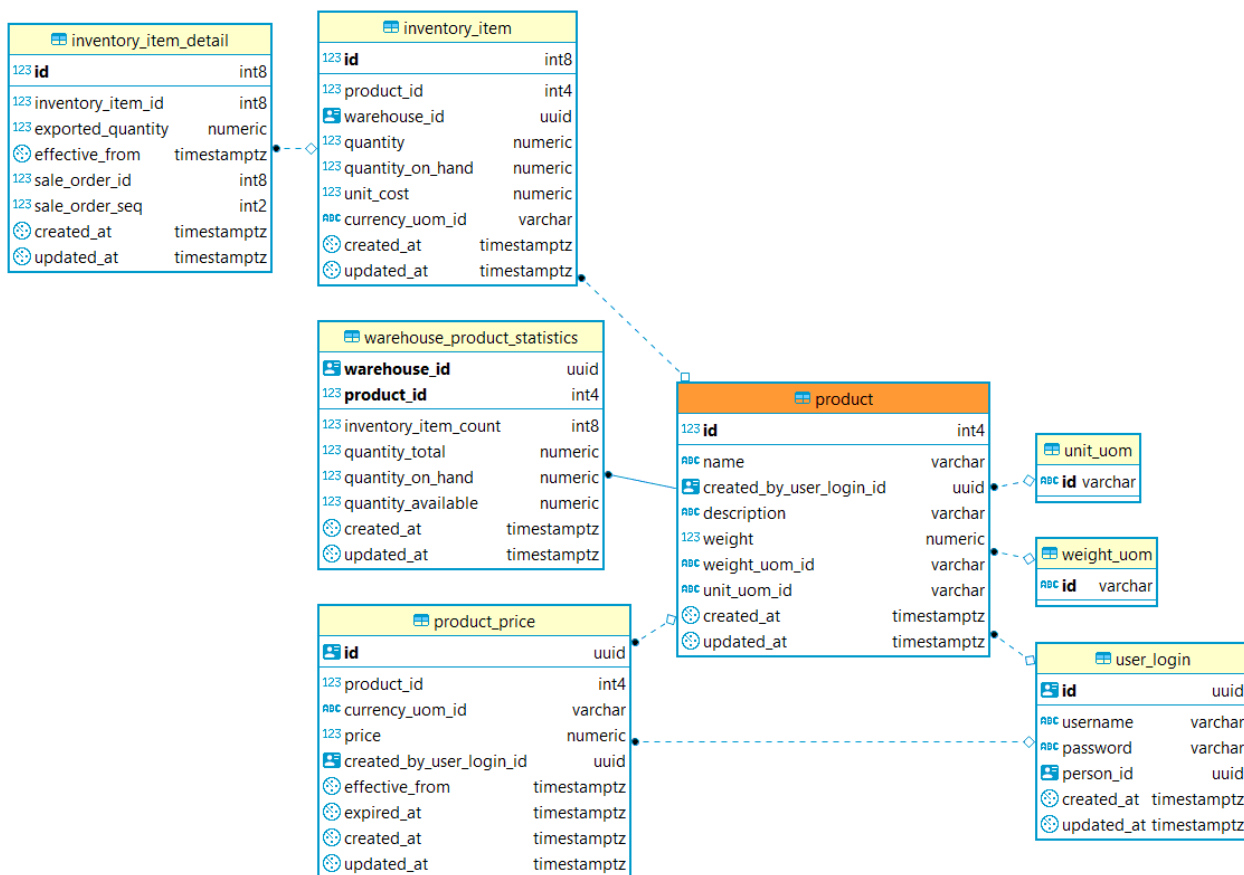


Hình 3.10: Cơ sở dữ liệu quản lý phân quyền, người dùng

Tên bảng	Thông tin chính được thể hiện
party_type	Loại người dùng (nhân viên, khách hàng)
party	Danh sách thông tin người dùng
customer	Danh sách thông tin khách hàng
person	Danh sách thông tin một nhân viên
user_login	Danh sách thông tin một tài khoản
security_group	Danh sách các role (account manager, product manager, ...)
security_permission	Danh sách các quyền (ADD_USER, EDIT_SALESMAN, ...)

Bảng 3.1: Thông tin các bảng nhóm quản lý phân quyền, người dùng

Nhóm các bảng phục vụ chức năng quản lý đơn hàng được trình bày qua sơ đồ thực thể liên kết (Hình 3.11) và bảng thông tin (Bảng 3.2).



Hình 3.12: Cơ sở dữ liệu quản lý sản phẩm

Tên bảng	Thông tin chính được thể hiện
product	Danh sách thông tin sản phẩm (product nói chung, chưa nhập kho)
product_price	Thông tin giá ứng với sản phẩm
warehouse_product_statistics	Thống kê các thông tin của sản phẩm (số lượng tổng, số lượng đặt, số lượng có sẵn, ...)
inventory_item	Thông tin sản phẩm trong kho
inventory_item_detail	Thông tin chi tiết đối với từng sản phẩm trong kho
unit_uom	Đơn vị hình dạng (hộp, chai, gói)
weight_uom	Đơn vị khối lượng (kg, g, mg)

Bảng 3.3: Thông tin các bảng nhóm quản lý sản phẩm

CHƯƠNG 4. CÁC VẤN ĐỀ GẶP PHẢI, GIẢI PHÁP ÁP DỤNG VÀ MINH HỌA CÁC CHỨC NĂNG CỦA HỆ THỐNG

4.1 Vấn đề lựa chọn công nghệ

4.1.1 Công nghệ front-end

Công nghệ front-end hiện nay phát triển rất nhanh và rộng. Có rất nhiều các phương pháp tiếp cận lập trình front-end, đi kèm là rất nhiều các thư viện, ngôn ngữ, framework, công cụ hỗ trợ. Mỗi một công nghệ mới ra đời đều đáp ứng tốt nhu cầu thị trường tại thời điểm đó và khắc phục được những nhược điểm của các công nghệ cũ. Ví dụ trước đây một lập trình viên front-end chỉ cần biết về HTML, CSS mà JavaScript để xây dựng một trang web thỏa mãn nhu cầu người dùng. Tuy nhiên theo thời gian, nhu cầu người dùng về tính tương tác của trang web ngày càng cao dẫn tới sự phát triển mạnh mẽ của ngôn ngữ JavaScript và các thư viện, framework xây dựng từ đó như thư viện JQuery, EmberJS hay các thư viện / framework nổi tiếng được dùng phổ biến hiện nay là ReactJS, Angular, VueJS.

Dựa theo xu thế công nghệ hiện nay, chúng tôi chọn ReactJS – Redux để xây dựng phần giao diện khi mà thư viện này có một cộng đồng lớn và được Facebook phát triển. React-Router để định tuyến, xây dựng Single Page Application, i18next để triển khai đa ngôn ngữ, Redux-Saga để xử lý middleware, gửi request và nhận api từ back-end server.

4.1.2 Công nghệ back-end

Về phần back-end, ban đầu chúng tôi chọn Java Spring để xây dựng. Java Spring cũng là một framework mạnh mẽ để xây dựng webserver. Tuy nhiên, Java Spring phù hợp với xây dựng các ứng dụng web monolithic (đơn khối) hơn là xây dựng theo mô hình microservices đang trở thành xu hướng trong thời gian hiện nay. Một hệ thống kiến trúc theo microservices thông thường sẽ bao gồm nhiều các service nhỏ nhưng số lượng nhiều thì Java Spring không phù hợp với những service đó do yêu cầu về tài nguyên của ứng dụng Java là khá lớn. Vì vậy chúng tôi chọn Go để xây dựng lại back-end server. Về cơ bản thì Go rất phù hợp cho việc xây dựng các service trong microservices do yêu cầu về tài nguyên nhỏ, ngoài ra Go còn hỗ trợ tốt xử lý đa luồng với Goroutine và Chanel đã trình bày ở phần công nghệ back-end nên sẽ có thể tăng số request phục vụ tại cùng một thời điểm của hệ thống.

4.1.3 Công nghệ Full Text Search

Như đã đề cập ở phần công nghệ, trong đề án này chúng tôi sử dụng tính năng Full Text Search đã có trên PostgreSQL kết hợp với sử dụng Index cho các câu truy vấn tìm kiếm người dùng, khách hàng, tài khoản, tên sản phẩm, ... Ban đầu chúng tôi chọn ElasticSearch cho tính năng này, tuy nhiên để sử dụng ElasticSearch thì dữ liệu trong PostgreSQL phải được sao lưu sang ElasticSearch. Để thực hiện việc sao lưu này có thể thực hiện trên code của web server hoặc sử dụng hệ thống Change Data Capture như Debezium.

Nếu thực hiện trên code thì không đảm bảo được dữ liệu ở PostgreSQL và trên ElasticSearch là giống nhau. Đồng thời sẽ làm code của web server trở nên phức tạp hơn do vừa phải quan tâm đến dữ liệu của PostgreSQL vừa cả dữ liệu trên ElasticSearch.

Nếu sử dụng hệ thống Change Data Capture như Debezium thì sẽ giải quyết được hai vấn đề trên, tuy nhiên lại làm hệ thống phức tạp lên rất nhiều vì phải triển khai cả Kafka làm kênh trung gian truyền dữ liệu từ PostgreSQL sang ElasticSearch. Việc sử dụng một công nghệ phức tạp như vậy vượt quá giới hạn thời gian cho đề án này. Vì vậy chúng tôi sử dụng cách đơn giản hơn là tính năng Full Text Search tích hợp sẵn trong PostgreSQL, khi đó sẽ không phải xét đến vấn đề sao lưu và tính nhất quán của dữ liệu.

4.2 Chức năng phân quyền động

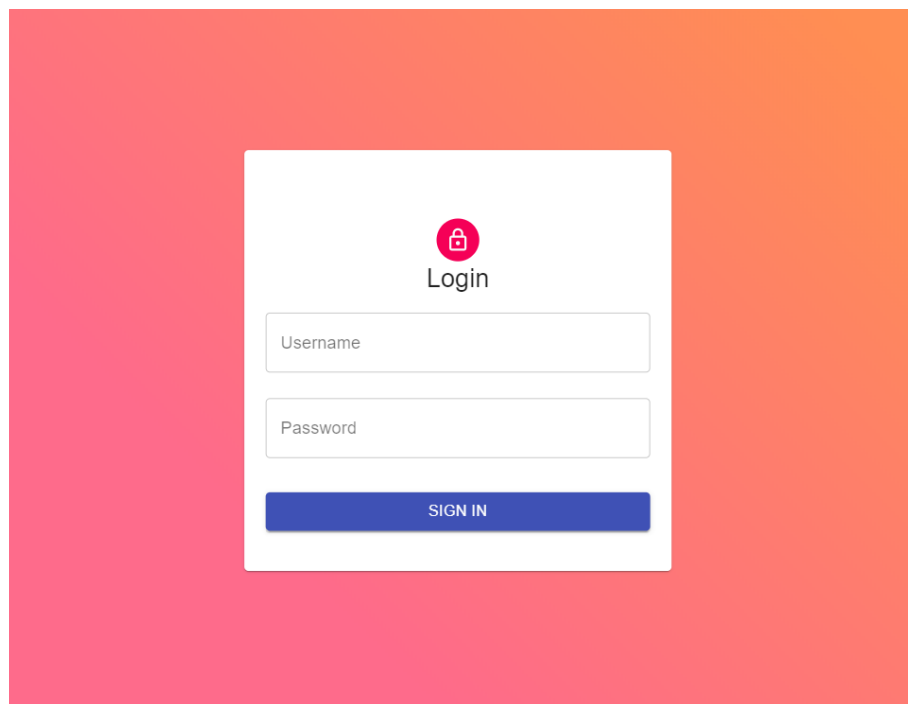
Phân quyền là tính năng không thể thiếu được trong các trang mang tính chất quản trị, mỗi người cần phải được giới hạn trong truy cập. Ví dụ không thể để một nhân viên bán hàng sử dụng được tính năng quản lý sản phẩm hay giá sản phẩm. Do đó dựa trên các tính năng đã phân tích như quản lý phân quyền, quản lý sản phẩm, quản lý tài khoản, quản lý kho, quản lý tuyến bán hàng, ... chúng tôi phân chia người dùng hệ thống vào các `security_group`, mỗi `security_group` lại có một tập quyền (permissions). Hệ thống sẽ dựa vào các permission mà người dùng có để xác định xem họ có thể truy cập được vào tính năng đó hay không.

security_group	security_permission
ADMIN	VIEW_EDIT_PARTY, VIEW_EDIT_USER_LOGIN, VIEW_EDIT_SECURITY_GROUP, VIEW_EDIT_SECURITY_PERMISSION
PRODUCT_MANAGER	VIEW_EDIT_PRODUCT
SALES_MANAGER	VIEW_EDIT_ORDER
FACILITY_MANAGER	VIEW_EDIT_FACILITY
INVENTORY_MANAGER	IMPORT
EXPORT_MANAGER	EXPORT
SALESMAN_MANAGER	VIEW_EDIT_SALESMAN
SALESMAN	SALESMAN_CHECKIN

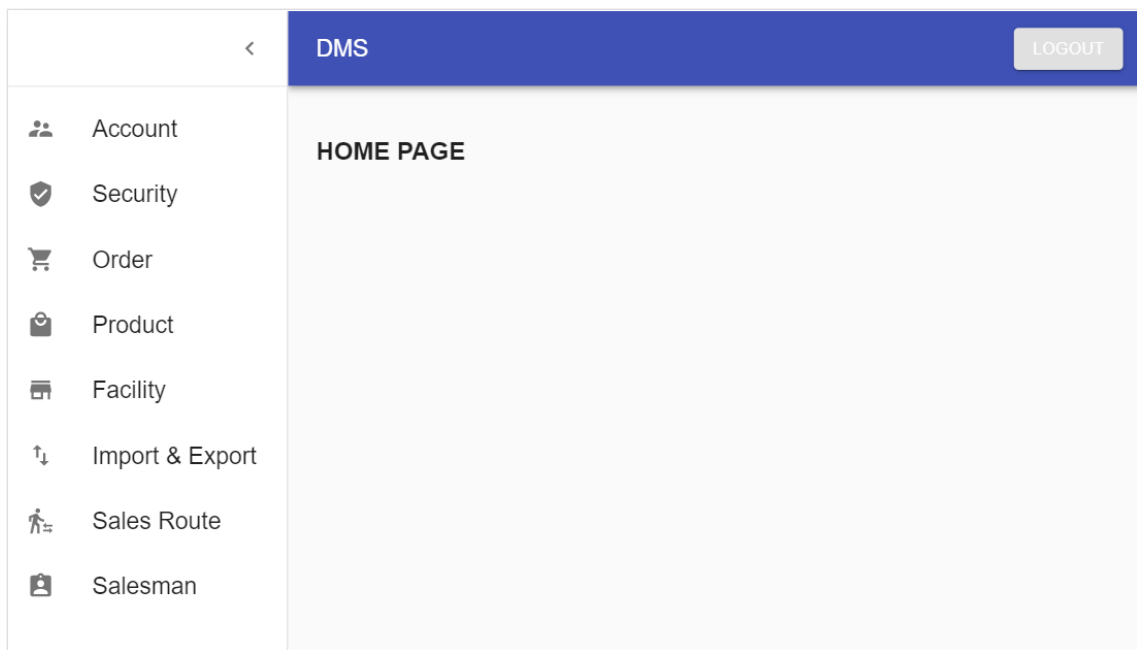
Bảng 4.1: Security_group và permission tương ứng

Khi một người dùng được thêm vào hệ thống, họ sẽ được cấp một user_login (hiểu như là username), giả sử người này là quản lý sản phẩm thì sẽ được cấp một quyền VIEW_EDIT_PRODUCT, nếu người này có khả năng quản lý rộng hơn làm được cả nhập kho thì sẽ cấp thêm quyền IMPORT, ... Đường dẫn (url) sẽ được gán theo permission để kiểm soát truy nhập. Ví dụ PRODUCT_MANAGER muốn xem danh sách các sản phẩm thì phải truy cập url có dạng .../view-product, url này chỉ được truy cập khi có quyền VIEW_EDIT_PRODUCT.

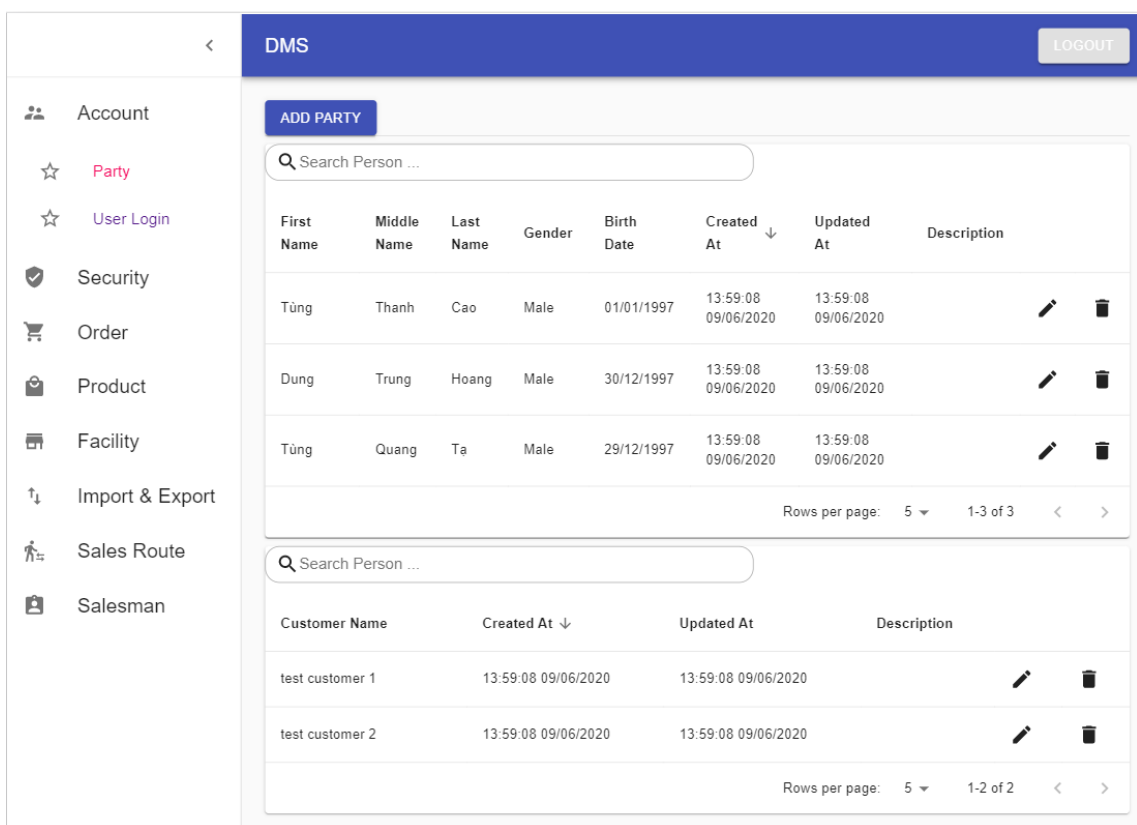
4.3 Minh họa các chức năng của hệ thống



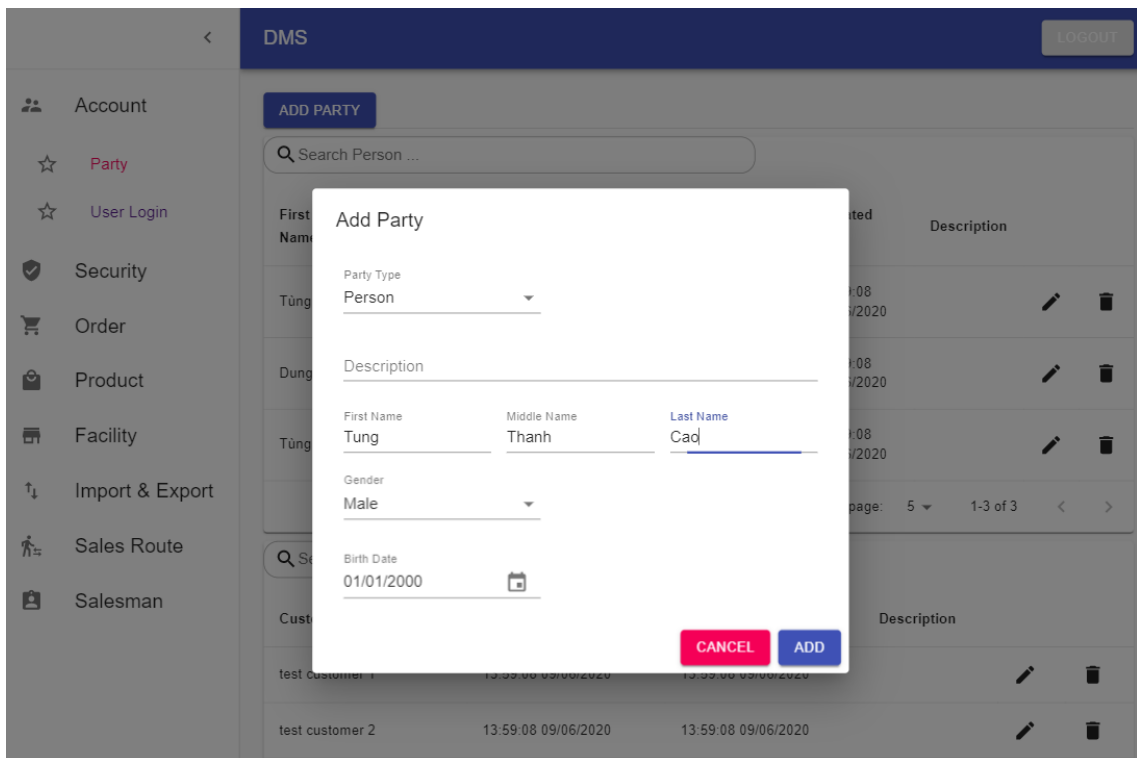
Hình 4.1: Chức năng đăng nhập



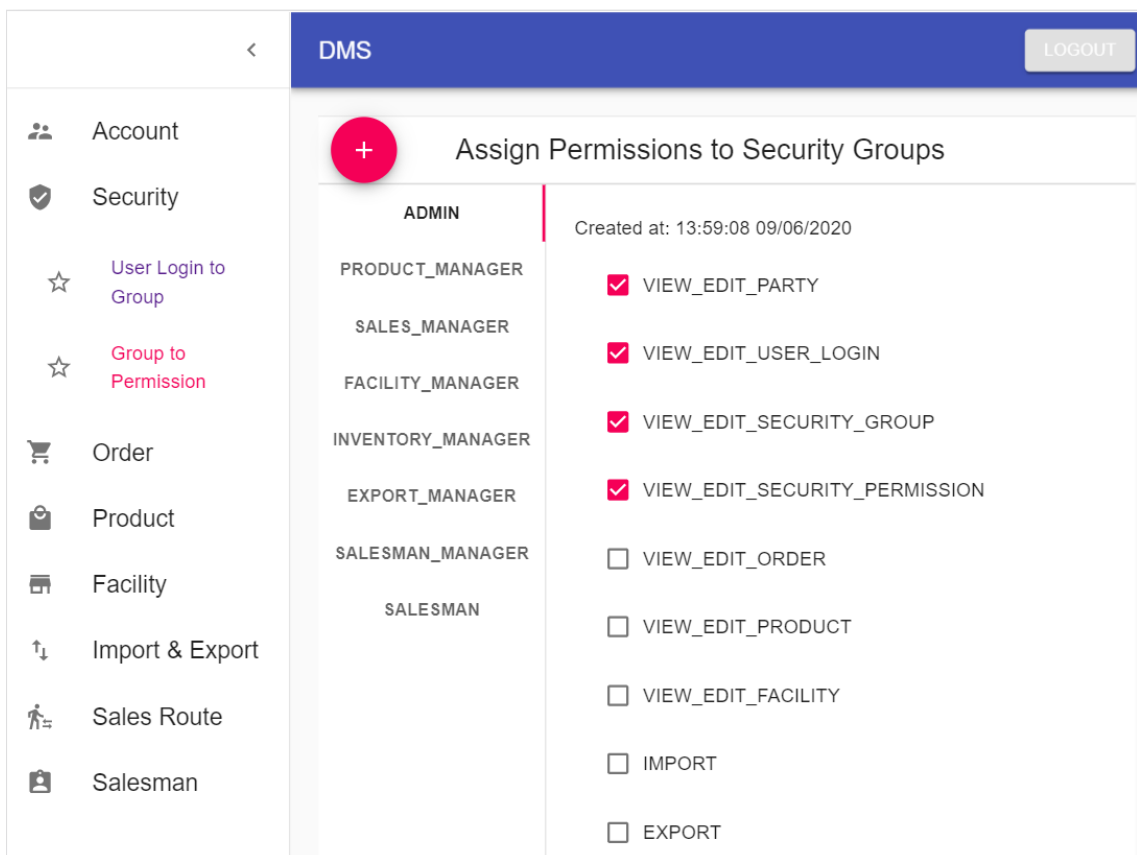
Hình 4.2: Giao diện trang chủ hệ thống



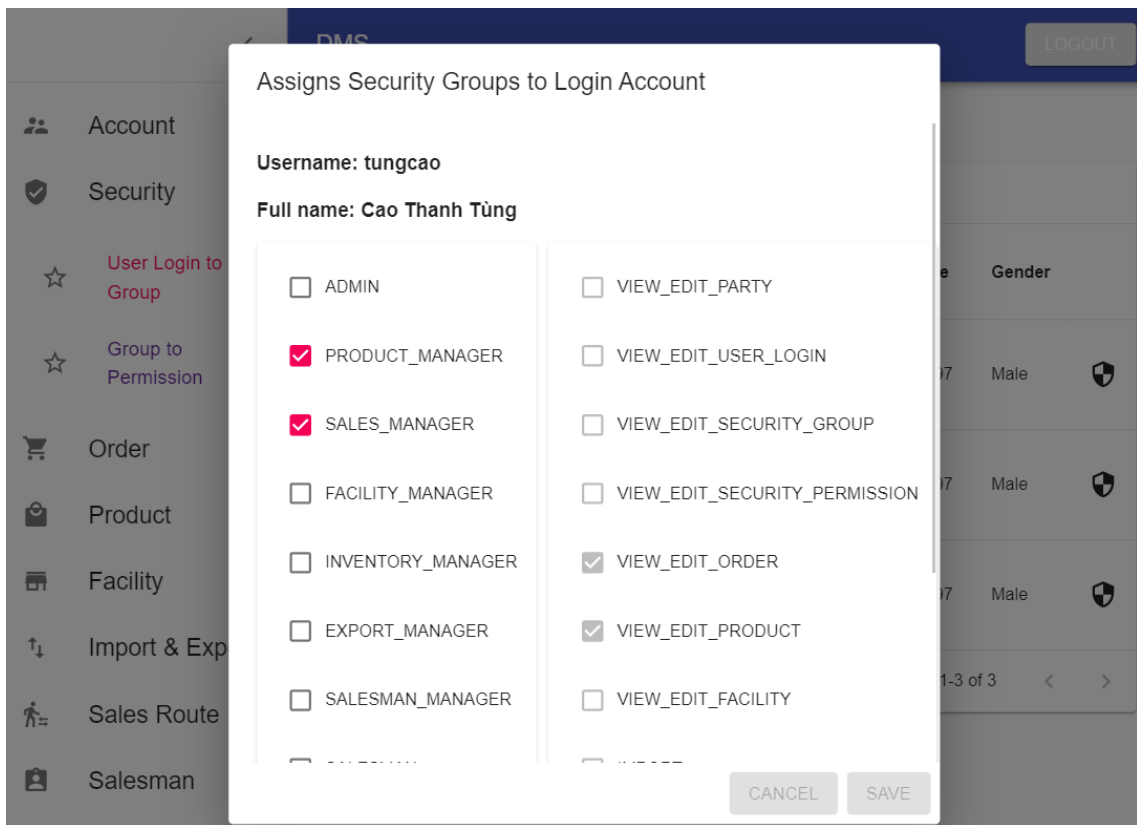
Hình 4.3: Quản lý người dùng



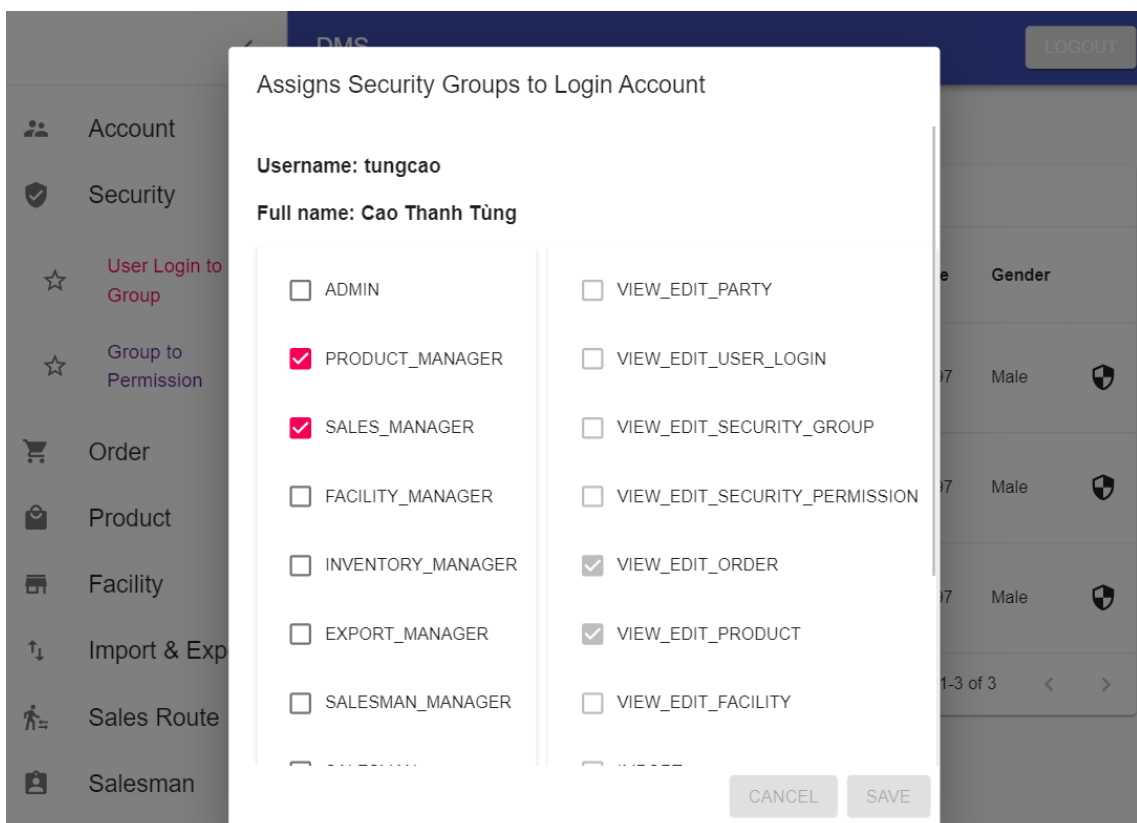
Hình 4.4: Thêm người dùng / khách hàng



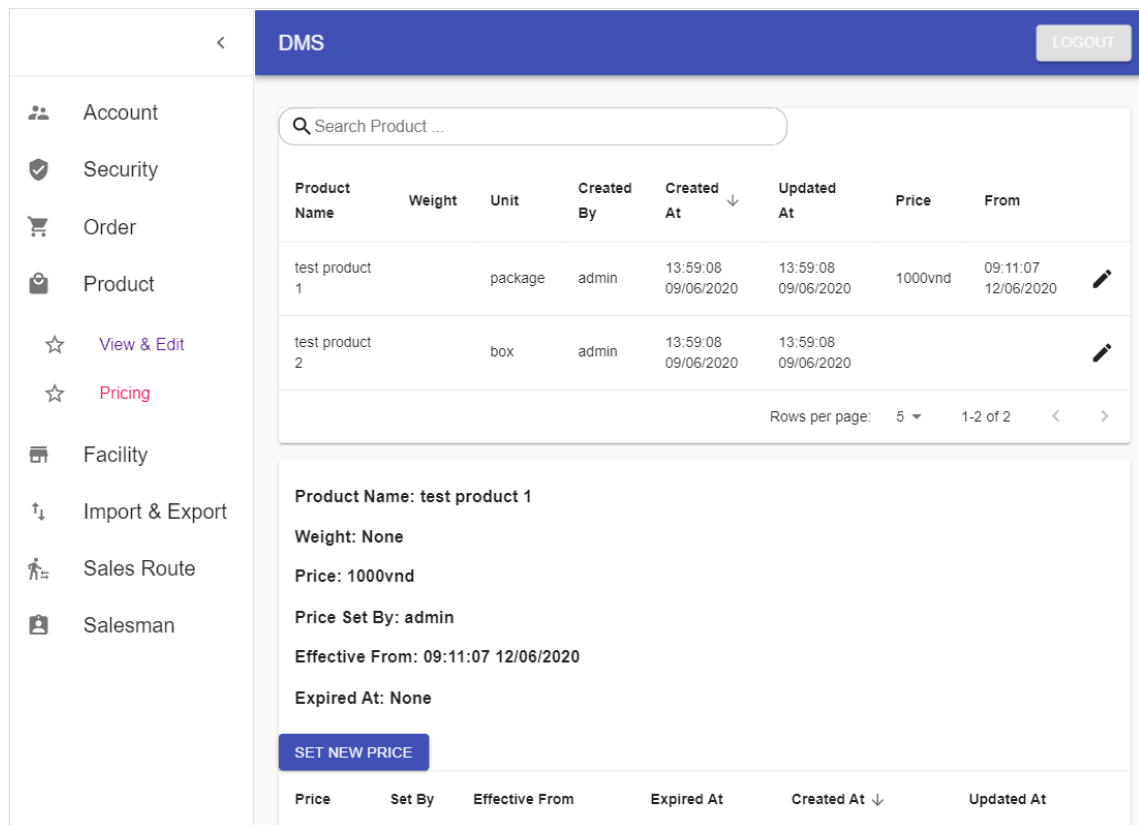
Hình 4.5: Quản lý các nhóm quyền



Hình 4.6: Gán quyền cho tài khoản người dùng

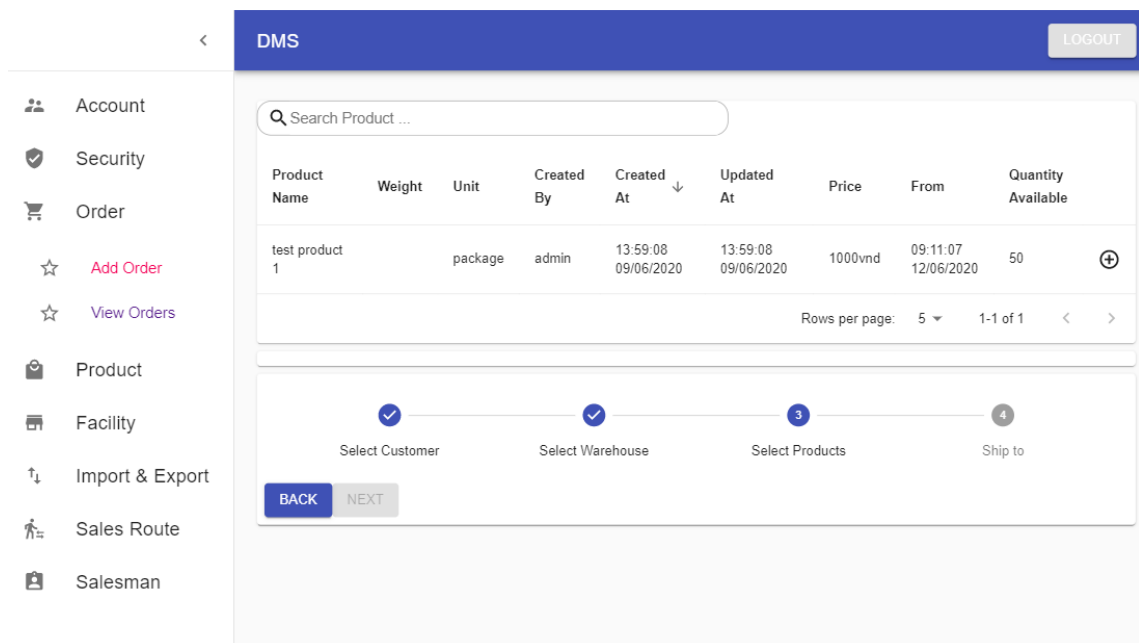


Hình 4.7: Gán quyền cho tài khoản người dùng



Hình 4.8: Quản lý giá của các sản phẩm

Tạo đơn hàng mới phải qua các bước như chọn khách hàng (để giao hàng đến), chọn kho chứa hàng hóa, chọn hàng hóa, nhập địa chỉ (nếu có hoặc chọn kho của cửa hàng).



Hình 4.9: Tạo đơn hàng

Account

Security

Order

☆

Add Order

☆

View Orders

Product

Facility

Import & Export

Sales Route

Salesman

DMS

LOGOUT

Filter Status

To Customer	From Warehouse	Created By	To Address	To Customer Store	Status	Created At	Updated At
Hoàng Trung Dũng	warehouse	admin	Ha Noi		Created	16:07:40 23/06/2020	16:07:40 23/06/2020
Cao Thanh Tùng	warehouse	admin		Do Son store	Created	16:07:22 23/06/2020	16:07:22 23/06/2020
Tạ Quang Tùng	warehouse	admin		thanh xuan store	Created	16:07:09 23/06/2020	16:07:09 23/06/2020

Rows per page:

5

1-3 of 3

Hình 4.10: Hiển thị đơn hàng

Account

Security

Order

Product

Facility

Import & Export

☆

Import

☆

Export

Sales Route

Salesman

DMS

LOGOUT

To Customer	From Warehouse	Created By	To Address	To Customer Store	Status	Created At	Updated At
Hoàng Trung Dũng	warehouse	admin	Ha Noi		Accepted	16:07:40 23/06/2020	16:09:06 23/06/2020
Cao Thanh Tùng	warehouse	admin		Do Son store	Accepted	16:07:22 23/06/2020	16:09:14 23/06/2020
Tạ Quang Tùng	warehouse	admin		thanh xuan store	Accepted	16:07:09 23/06/2020	16:09:21 23/06/2020
Rows per page: 5 1-3 of 3							

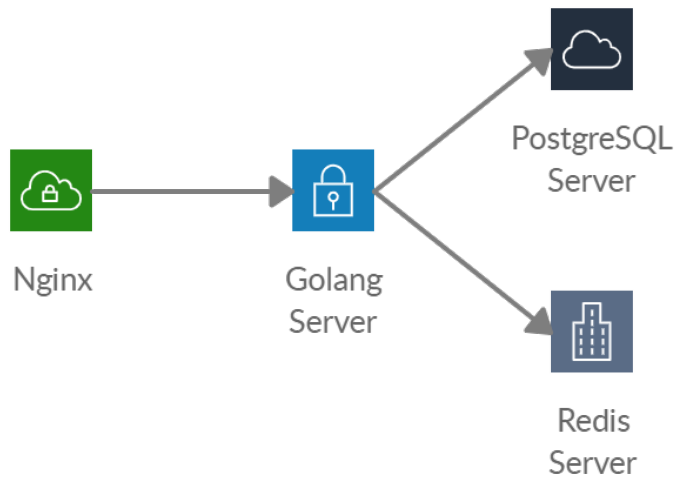
To Customer	From Warehouse	Created By	To Address	To Customer Store	Status	Created At	Updated At
Rows per page: 5 0-0 of 0							

Hình 4.11: Quản lý xuất kho

CHƯƠNG 5. THỬ NGHIỆM VÀ ĐÁNH GIÁ

5.1 Triển khai chạy thử nghiệm trên cloud

Trong đề án này, chúng tôi sử dụng Microsoft Azure làm nhà cung cấp dịch vụ cloud và sử dụng Azure Kubernetes Service để triển khai các server [4]. Kiến trúc tổng quan của hệ thống như hình sau:

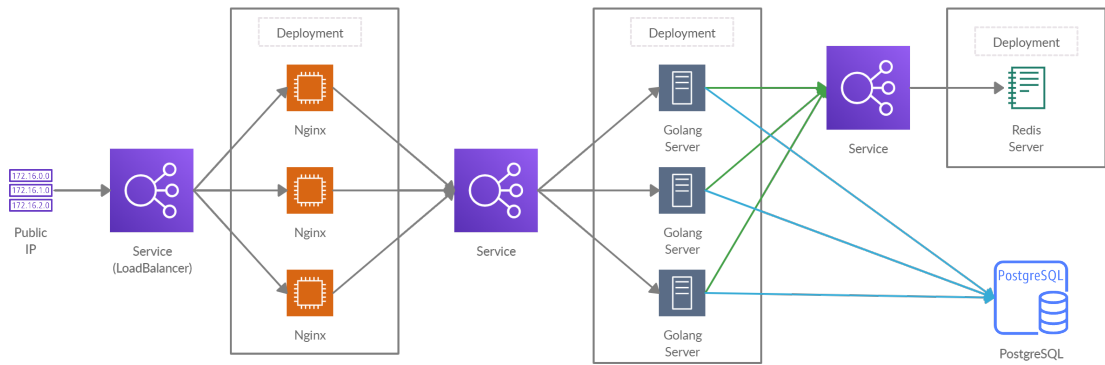


Hình 5.1: Kiến trúc tổng quan của hệ thống

Trong đó:

- Nginx: sử dụng để làm web server trả về các file tĩnh html và js khi ứng dụng được mở lên lần đầu, đồng thời được sử dụng làm Reverse Proxy đến Golang Server cho các REST API.
- Golang Server: web server cho các REST API từ trình duyệt gửi lên thông qua AJAX.
- PostgreSQL Server: cơ sở dữ liệu quan hệ của hệ thống.
- Redis Server: lưu trữ dữ liệu của các phiên làm việc (session).

Còn để triển khai lên AKS, chúng tôi đã sử dụng những thành phần sau:



Hình 5.2: Triển khai lên Azure Kubernetes Service

Kết quả sau khi triển khai là hệ thống có thể truy cập được từ địa chỉ public IP như sau:

Product Name	Weight	Unit	Created By	Created At	Updated At	Description
Ms. Agnes Witting	93.6g	bottle	admin	14:45:54 09/06/2020	14:45:54 09/06/2020	Accusantium aut voluptate sit perferendis.
Mrs. Rowena Hoeger	2.34kg	bottle	admin	14:45:54 09/06/2020	14:45:54 09/06/2020	Perferendis accusantium s consequatur voluptatem.
Queen Piper Rippin	1.76kg	bottle	admin	14:45:54 09/06/2020	14:45:54 09/06/2020	Consequatur accusantium perferendis voluptatem.
Lady Melyssa Beier	97.6mg	package	admin	14:45:54 09/06/2020	14:45:54 09/06/2020	Consequatur accusantium perferendis aut.
Ms. Kianna Gerlach	11kg	box	admin	14:45:54 09/06/2020	14:45:54 09/06/2020	Accusantium aut consequa voluptatem sit.

Hình 5.3: Truy cập ứng dụng từ public IP của AKS

5.2 Kiểm thử hiệu năng (performance testing)

Dưới đây là kết quả chạy thử nghiệm với số lượng gần một triệu bản ghi trong bảng. Do cách mà PostgreSQL hoạt động, câu lệnh select count sẽ rất chậm trong bảng kích thước lớn do PostgreSQL phải duyệt qua toàn bộ phần tử trong bảng mới xác định được số lượng phần tử tương ứng. Vì vậy, trong bài kiểm thử hiệu năng này, chúng tôi sử dụng hai cách để đếm số lượng phần tử: sử dụng select count (no

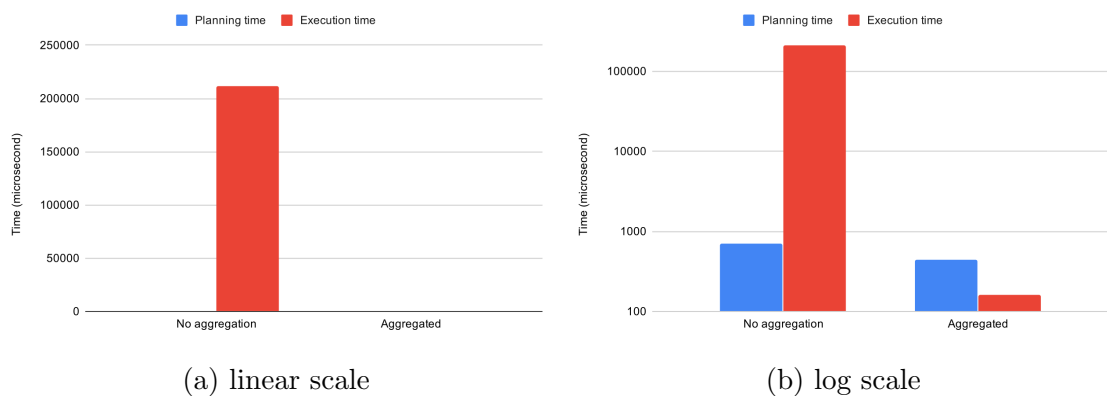
aggregation) và sử dụng một biến đếm riêng biệt (aggregated). Đồng thời cũng chia các trường hợp có sử dụng hoặc không sử dụng Index.

Bài kiểm thử hiệu năng này được thử nghiệm trên cả việc chạy trên máy tính cá nhân (máy local) và cả sau khi đã triển khai trên cloud (máy cloud). PostgreSQL trên máy local sử dụng Intel core i3 3217U còn máy trên cloud sử dụng PostgreSQL của Microsoft Azure với 2 core.

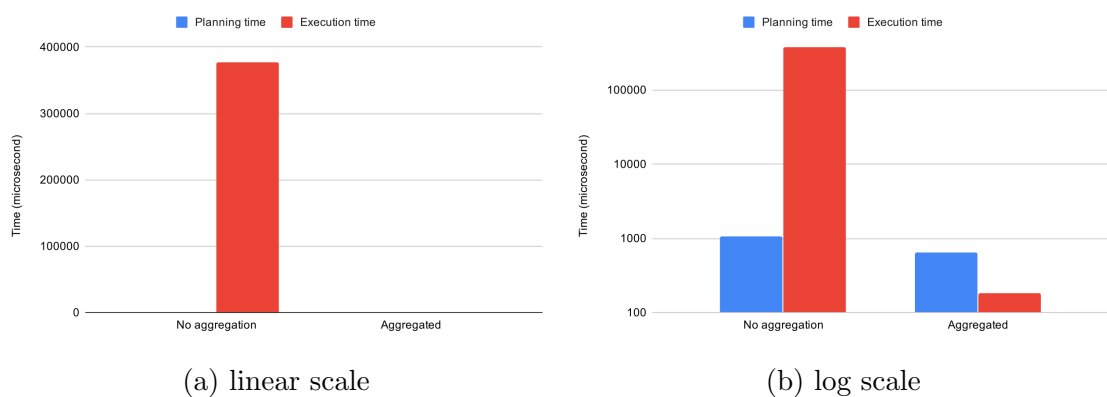
5.2.1 Kiểm thử hiệu năng cơ sở dữ liệu

Với bài kiểm tra hiệu năng cơ sở dữ liệu, chúng tôi sử dụng lệnh **EXPLAIN ANALYZE** để đồng thời lấy được query plan của câu truy vấn và lấy được thời gian lập kế hoạch (planning time) và thời gian thực thi (execution time) của truy vấn.

5.2.1.1 Truy vấn count



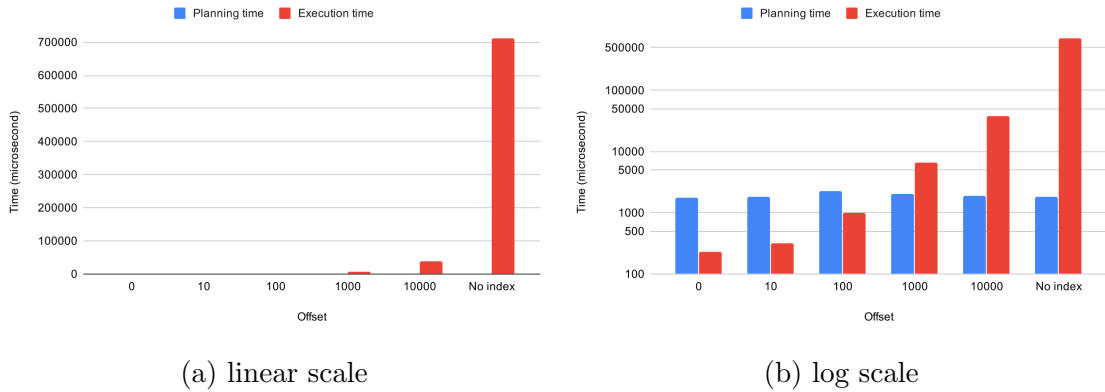
Hình 5.4: Truy vấn count trên máy local



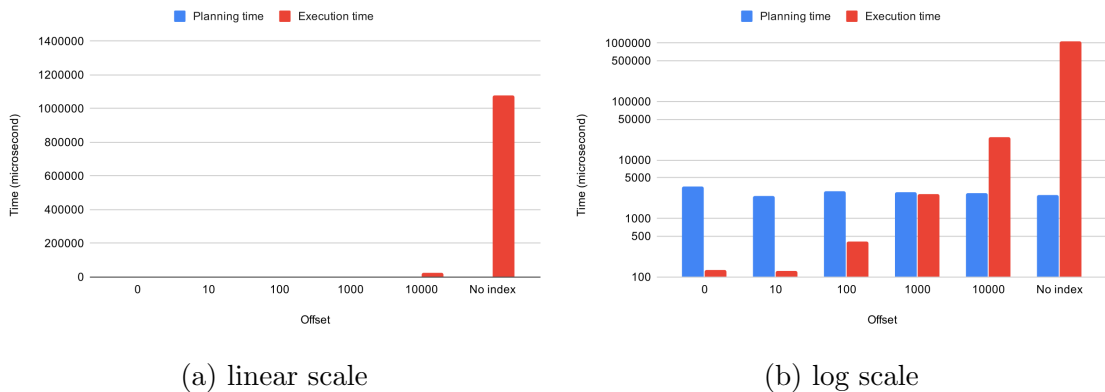
Hình 5.5: Truy vấn count trên máy cloud

Đánh giá kết quả: Có thể thấy được rằng với một triệu phần tử thì việc sử dụng trực tiếp select count là rất tốn kém, gấp rất nhiều lần so với việc sử dụng một biến đếm riêng biệt.

5.2.1.2 Truy vấn select



Hình 5.6: Truy vấn select trên máy local

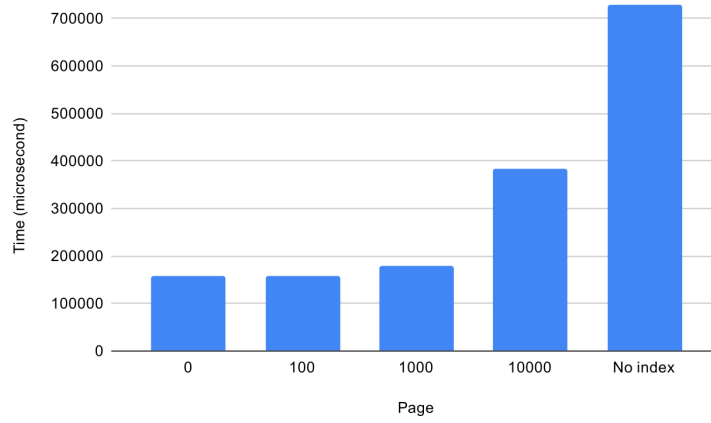


Hình 5.7: Truy vấn select trên máy cloud

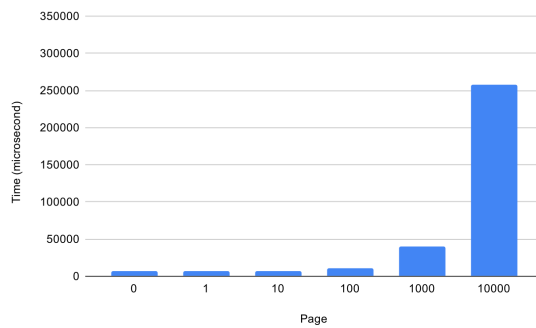
Đánh giá kết quả: Có thể thấy được việc sử dụng index giúp câu lệnh select có sắp xếp được cải thiện hiệu năng hơn rất nhiều với bảng một triệu phần tử. Đồng thời việc tăng offset của lệnh select cũng làm tăng dần thời gian thực thi. Việc không sử dụng index có thể làm cho thời gian một câu lệnh select lên đến 1s.

5.2.2 Kiểm thử hiệu năng REST API

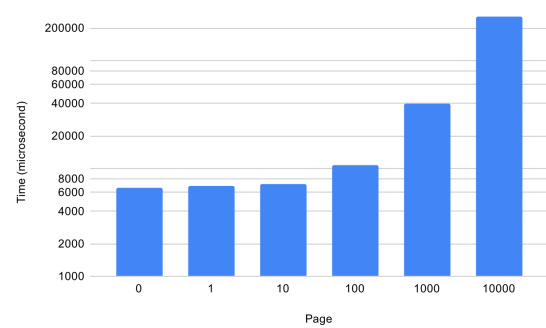
Với bài kiểm thử hiệu năng REST API cho ứng dụng chạy trên Microsoft Azure, ngoài việc sử dụng AKS để chạy các server của ứng dụng thì chúng tôi sử dụng thêm một dịch vụ khác là Container Instance để chạy các request và đo đạc thời gian.



Hình 5.8: Hiệu năng REST API trên máy local (no aggregation)

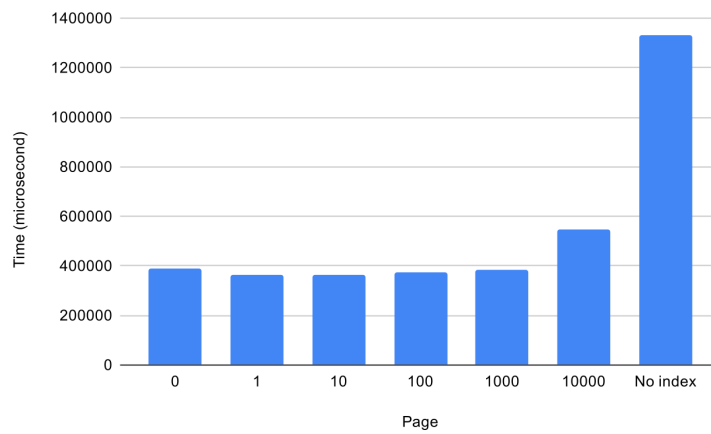


(a) linear scale

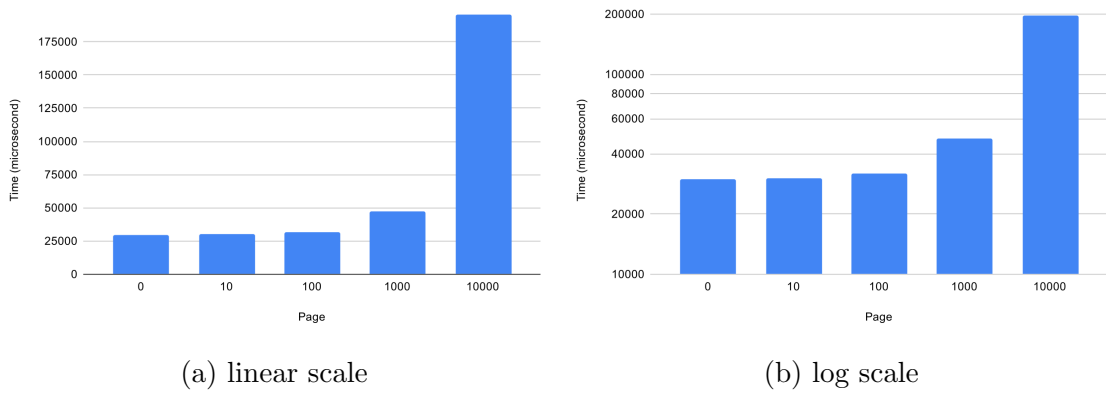


(b) log scale

Hình 5.9: Hiệu năng REST API trên máy local (aggregated)



Hình 5.10: Hiệu năng REST API trên máy cloud (no aggregation)

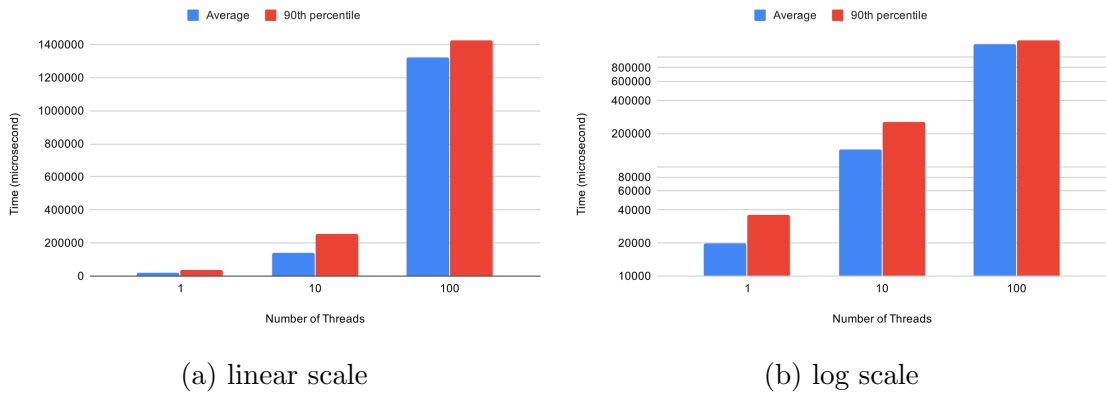


Hình 5.11: Hiệu năng REST API trên máy cloud (aggregated)

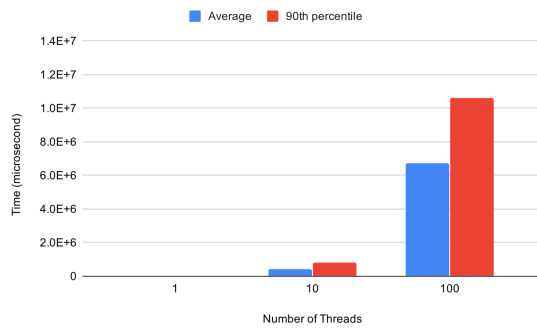
Đánh giá kết quả: Có thể thấy ảnh hưởng của việc sử dụng trực tiếp lệnh select count lên tốc độ của REST API là rất lớn. Tốc độ chậm của lệnh select count khiến cho REST API không thể kết thúc dưới 100ms.

5.3 Kiểm thử sức chịu tải (stress testing)

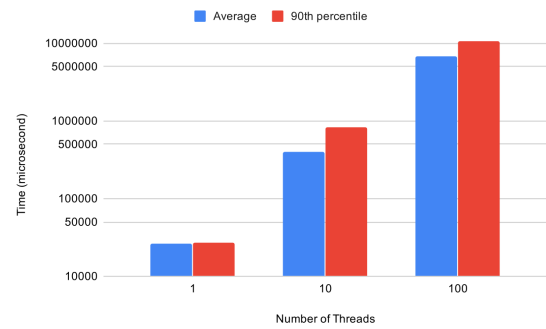
Với bài toán kiểm thử sức chịu tải, chúng tôi cho đồng thời một số lượng luồng chạy và gửi request lên server và tính thời gian hoàn thành. Số lượng luồng tương ứng là 1, 10 và 100.



Hình 5.12: Stress test thêm inventory item trên máy local



(a) linear scale



(b) log scale

Hình 5.13: Stress test thêm inventory item trên máy cloud

Đánh giá kết quả: Khi tăng số request chạy đồng thời thì thời gian chạy của các request cũng tăng lên do server chỉ có thể xử lý đồng thời được một số lượng hữu hạn các request, các request vượt giới hạn đó sẽ phải đợi request trước đó hoàn thành.

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Đồ án được thực hiện trong khuôn khổ của nhóm với hai sinh viên nhắm tới mục tiêu thiết kế và xây dựng hệ thống với các tính năng cơ bản nhất của một hệ thống quản lý phân phối bao gồm quản lý khách hàng, quản lý nhân viên bán hàng, quản lý tuyến bán hàng, xây dựng kế hoạch tuyến bán hàng, lên đơn hàng, quản lý xuất kho, tồn kho.

Trong đồ án này, chúng tôi đặt mục tiêu xây dựng phân hệ quản lý sản phẩm, quản lý đơn hàng, quản lý xuất nhập kho của các sản phẩm. Kết quả đạt được của đồ án là phân hệ quản lý đơn hàng và xuất kho với 20 use case, được phát triển dựa trên công nghệ Go phía server và ReactJS/Redux phía giao diện. Phân hệ đã sử dụng công nghệ container Docker, nền tảng Container orchestration engine là Kubernetes và sử dụng dịch vụ cloud Microsoft Azure để triển khai và thử nghiệm ứng dụng. Quá trình triển khai ứng dụng đã được hoàn thiện.

Qua quá trình tìm hiểu và thực hiện hệ thống quản lý phân phối, nghiên cứu quy trình và các chức năng nghiệp vụ, thiết kế giao diện, cơ sở dữ liệu thì phần mềm nhóm chúng tôi tạo ra đã cơ bản hoàn thiện, đáp ứng đủ những ca sử dụng như thiết kế. Phần giao diện viết bằng ReactJS theo chuẩn Single Page Application, thiết kế theo phong cách Material Design.

Để trở thành một dự án khả thi trong thực tế thì ứng dụng cần phải khắc phục nhiều hạn chế. Như đã trình bày, hiện nay KiotViet hay Sapo là những công ty cung cấp các ứng dụng quản lý phân phối rất lớn, việc cạnh tranh một chỗ đứng với sản phẩm của họ là hạn chế đầu tiên. Phần giao diện còn tương đối sơ sài vì chưa có nhiều thời gian chỉnh sửa, việc này đòi hỏi nhiều thứ như nên thiết kế trang chủ thế nào, thiết kế logo, vị trí đầu trang và chân trang cần những thông tin như thế nào, hiệu ứng trượt dọc, dropdown, ... Ở các công ty thì họ có kỹ sư thiết kế UX/UI, một đội photoshop, một đội dựng giao diện và hiệu ứng, do đó đây là hạn chế thứ hai của nhóm chúng tôi. Hạn chế tiếp theo là, dù đã sử dụng webpack để nén mã nguồn, Single Page để tối ưu tốc độ tải của trang web nhưng do dùng nhiều thư viện vẫn chưa thực sự kiểm soát được hiệu năng của front-end. Ví dụ như trang thegioididong.com có tốc độ tải trang vô cùng nhanh, vì họ có nhiều giải pháp tối ưu. Ảnh của thegioididong rất nhỏ, chỉ tầm 20-40kb, điều này cho thấy các ảnh được tối ưu rất kỹ, họ cũng không dùng bootstrap; ngoài ra trang còn áp dụng lazy

load tức khi cuộn trang xuống mới thấy các ảnh bên dưới. Thứ hai, họ xử lý CSS và JS đúng cách, không phải tải bất kì file CSS nào mà bỏ toàn bộ CSS vào head (giúp thời gian render giảm từ 378ms xuống còn 225ms). Thứ ba, họ cache mọi thứ, cụ thể các tài nguyên như ảnh, CSS, JS được cache trong vòng 1 năm, do vậy khi load lại trang trình duyệt không cần tải lại ảnh, CSS hay JS này nữa. Một hạn chế nữa, ứng dụng chưa có giải pháp xử lý trong tình huống có nhiều request lên server cùng một lúc, ví dụ như có nhiều yêu cầu lên đơn hàng đồng thời trong khi số lượng hàng trong kho lại không đủ.

6.2 Hướng phát triển của đề án trong tương lai

Là một phần mềm mang tính thương mại nên chúng tôi rất muốn có thể triển khai ứng dụng trong thực tế. Để làm được việc này thì trước tiên chúng tôi cần hoàn thiện phần giao diện của mình, sau đó là chỉnh sửa lại các tính năng đã có cho dễ sử dụng hơn, đẹp hơn; khắc phục các hạn chế đã nêu ở trên; thiết kế thêm các tính năng mới.

Chúng tôi cũng mong muốn được thử nghiệm hiệu năng với nhiều tình huống thực tế khác và đánh giá tác động của Full Text Search trong việc tìm kiếm và chèn, sửa, xóa các bản ghi. Đồng thời, chúng tôi rất có thể triển khai trên các dịch vụ cloud khác như Google Cloud hay Amazon Web Service hay triển khai nó trên một cluster tự quản lý.

TÀI LIỆU THAM KHẢO

- [1] Logistics là gì? <https://www.container-transportation.com/logistics-la-gi.html>. (Accessed: 2020-06-20).
- [2] Phần mềm dms là gì? <https://mobiwork.vn/phan-mem-dms-la-gi/>. (Accessed: 2020-06-25).
- [3] Postgresql 11.8 documentation. <https://www.postgresql.org/docs/11/index.html>. (Accessed: 2020-06-20).
- [4] Quickstart: Deploy an azure kubernetes service cluster using the azure cli. <https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough>. (Accessed: 2020-06-20).
- [5] Redis: Kho lưu trữ dữ liệu trong bộ nhớ. cách hoạt động và lý do nên sử dụng. https://aws.amazon.com/vi/redis/?nc1=h_ls. (Accessed: 2020-06-25).
- [6] Thuật toán k-means với bài toán phân cụm dữ liệu - bis. <http://bis.net.vn/forums/t/374.aspx>. (Accessed: 2020-06-25).
- [7] Marko Luksa. *Kubernetes in Action*. Manning Publications, 1 edition, 1 2018.
- [8] Nigel Poulton. *Docker Deep Dive*. 7 2017.
- [9] Ravi S.Sandhu. Role-based access control. *Advances in Computers*, 46:237–286, 1998.