

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



**LUẬN VĂN TỐT NGHIỆP
Hệ thống vận chuyển hàng hóa liên tỉnh**

Hội đồng LVTN:

Khoa học máy tính

Giảng viên hướng dẫn:

ThS. Nguyễn Thị Ái Thảo

Khoa KH & KT Máy tính, ĐHBK

ThS. Nguyễn Đình Thành

Khoa KH & KT Máy tính, ĐHBK

Giảng viên phản biện:

ThS. Phạm Nguyễn Hoàng Nam Khoa KH & KT Máy tính, ĐHBK

Sinh viên thực hiện:

Vương Anh Khoa

1711803

Cao Đăng Dũng

1710849

Đặng Văn Dũng

1710853

Tp. Hồ Chí Minh, tháng 07, 2021

Lời cam đoan

Chúng tôi cam đoan mọi điều được trình bày trong báo cáo, cũng như mã nguồn là do tôi tự thực hiện - trừ các kiến thức tham khảo có trích dẫn cũng như mã nguồn mẫu do chính nhà sản xuất cung cấp, hoàn toàn không sao chép từ bất cứ nguồn nào khác. Nếu lời cam đoan trái với sự thật, tôi xin chịu mọi trách nhiệm trước Ban Chủ Nhiệm Khoa và Ban Giám Hiệu Nhà Trường.

Nhóm sinh viên thực hiện đề tài

Lời cảm ơn

Từ những ngày đầu bước chân vào trường Đại học Bách Khoa - Đại học Quốc gia TPHCM, chúng em đã nhận được sự chỉ bảo tận tình của các Thầy, Cô trong trường. Đó chính là niềm cảm hứng và động lực to lớn thúc đẩy tụi em phấn đấu học tập và rèn luyện tại ngôi trường nổi tiếng là “khó nhằn” này.

Quá trình thực hiện luận văn tốt nghiệp là giai đoạn quan trọng nhất trong quãng đời của mỗi sinh viên. Là cơ hội để chúng em có thể tổng hợp lại nguồn kiến thức nền tảng mà các Thầy, Cô trong trường đã dạy. Từ đó chúng em có sự chuẩn bị tốt hơn trên con đường lập nghiệp đầy gian nan và thử thách.

Trước hết, chúng em xin chân thành cảm ơn quý Thầy, Cô khoa Khoa Học và Kỹ Thuật Máy Tính đã tận tình chỉ bảo và trang bị cho chúng em những kiến thức cần thiết trong suốt thời gian ngồi trên ghế nhà trường. Làm nền tảng để cho chúng em có thể hoàn thành được bài luận văn này.

Đặc biệt, chúng em xin gửi lời cảm ơn chân thành nhất đến giảng viên ThS. Nguyễn Thị Ái Thảo đã trực tiếp hướng dẫn nhóm làm đề tài này. Trong quá trình hướng dẫn cô đã tận tình hướng dẫn, định hướng, giải đáp thắc mắc, chỉ ra những sai sót, khuyết điểm để giúp tụi em có thể hoàn thành luận văn của mình. Một lần nữa tụi em xin chân thành cảm ơn cô.

Và chúng em xin gửi lời cảm ơn đến gia đình và bạn bè. Những người đã luôn đồng hành và giúp đỡ chúng em trong những giai đoạn khó khăn nhất trên ghế giảng đường để chúng em có thêm động lực và quyết tâm hoàn thành ước mơ của mình.

Trong quá trình hoàn thành luận văn, vì trình độ lý luận cũng như kinh nghiệm còn rất hạn chế nên không thể tránh khỏi sai sót, em rất mong nhận được ý kiến của thầy, cô để tụi em có thể cải thiện và hoàn thành luận văn của mình tốt hơn.

Tóm tắt

Ngày nay, việc vận chuyển hàng hóa đã trở thành một trong những nhu cầu thiết yếu của mọi người. Nó có ảnh hưởng lớn đến sự vận hành của nền kinh tế, sự vận chuyển liên tục hiệu quả sẽ thúc đẩy sự phát triển của nền kinh tế. Cùng với đó thì dịch vụ vận chuyển hàng hóa đang là một trong những mắt xích quan trọng nằm trong chuỗi cung ứng và đang trở thành một trong những ngành đóng vai trò quan trọng cho sự phát triển của kinh tế xã hội, giúp cho các hoạt động lưu thông, chuyên chở hàng hóa được thực hiện nhanh chóng, dễ dàng đưa sản phẩm, hàng hóa tiếp cận đến tận tay người dùng và đến mọi vùng miền trên tổ quốc một cách nhanh chóng và hiệu quả nhất.

Cùng với sự phát triển của công nghệ thì chúng ta có thể ứng dụng chúng vào để quản lý việc giao nhận, vận chuyển hàng hóa, giúp quá trình đó trở nên đơn giản và hiệu quả hơn đối với các bên liên quan. Để thực hiện đề tài này thì nhóm đã khảo sát các dịch vụ giao hàng, vận chuyển hàng hóa đang có hiện nay để có thể định hình được cơ bản nhu cầu của người dùng và tìm hiểu cái mà người dùng mong đợi ở một hệ thống vận chuyển hàng hóa. Từ những nhu cầu như vậy thì nhóm đã đưa ra được những bài toán lớn cần phải giải quyết cho hệ thống vận chuyển mà nhóm xây dựng. Cùng với đó nhóm cũng đã tìm hiểu và áp dụng một số công nghệ phổ biến, phù hợp để nâng cao tính mở rộng và phát triển của hệ thống.

Mục lục

1	Giới thiệu đề tài	1
1.1	Tổng quan đề tài	1
1.2	Mục tiêu và phạm vi đề tài	2
1.3	Cấu trúc luận văn	3
2	Cơ sở lý thuyết và công nghệ	4
2.1	Front-end	4
2.1.1	SPA	4
2.1.2	ReactJS	6
2.1.3	Continuous integration / Continuous deployment (CI/CD)	7
2.1.4	Những thư viện nổi bật giúp hiện thực chức năng hệ thống	8
2.1.5	Giám sát lỗi hệ thống bằng dịch vụ Sentry	9
2.1.6	Phân tích số lượng và hành vi người dùng với Google Analytics . .	12
2.1.7	Front-end Unit testing	15
2.1.8	Tổng kết	18
2.2	Back-end	18
2.2.1	Giới thiệu kiến trúc Microservices	18
2.2.2	Công nghệ tổng quan	19
2.2.3	Kiến trúc RESTful API	20
2.2.4	MySQL	22
2.2.5	Cassandra	23
2.2.6	Redis	24
2.2.7	ActiveMQ	26
2.2.8	Kafka	29
2.2.9	Docker	31
2.2.10	NGINX	32

3 Bài toán và giải pháp	34
3.1 Bài toán xác thực người dùng	34
3.2 Bài toán xác thực giữa các services	36
3.3 Bài toán nhận nhiều Request của người dùng	36
3.4 Bài toán giảm thời gian xử lý khi người dùng truy xuất thông tin	37
3.5 Bài toán xác định kho nguồn - kho đích ngắn nhất và tính chi phí vận chuyển	39
3.6 Bài toán thanh toán chi phí vận chuyển	39
3.7 Bài toán phân phối đơn hàng cho tài xế	40
3.8 Bài toán lưu trữ và truy xuất dữ liệu thống kê	41
3.9 Bài toán triển khai các services và cơ sở hạ tầng	42
3.10 Bài toán bảo mật các service trong private network, cấu hình SSL, cân bằng tải	45
3.11 Bài toán gợi ý địa điểm khi người dùng nhập địa chỉ	47
3.12 Bài toán giảm thời gian load trang web lần đầu tiên	47
4 Phân tích và thiết kế hệ thống	49
4.1 Phân tích yêu cầu	49
4.1.1 Yêu cầu người dùng	49
4.1.2 Yêu cầu hệ thống	51
4.2 Thiết kế hệ thống	52
4.2.1 Kiến trúc hệ thống	52
4.2.2 Thiết kế và đặc tả Use Case	55
4.2.3 Mô tả trạng thái đơn hàng, kiện hàng	67
4.2.4 Thiết kế cơ sở dữ liệu	71
5 Hiện thực hệ thống	77
5.1 Công nghệ sử dụng	77
5.2 Quản lý mã nguồn	78
5.3 Kết quả hiện thực	79
5.3.1 Chức năng đăng nhập, đăng ký	79
5.3.2 Chức năng dành cho người gửi	81
5.3.3 Chức năng dành cho tài xế	86
5.3.4 Chức năng dành cho thủ kho	90
5.3.5 Chức năng dành cho Quản lý	95
5.3.6 Chức năng dành cho người nhận	100
6 Kiểm thử hệ thống	104
6.1 Kiểm thử API	104
6.2 Kiểm thử chức năng	122
6.3 Kiểm thử phi chức năng	123

7	Tổng kết	127
7.1	Kết quả đạt được	127
7.2	Hạn chế	127
7.3	Hướng phát triển tương lai	128

Danh sách hình vẽ

1.1	Luồng cơ bản giao nhận hàng hóa	2
2.1	Luồng dữ liệu của các trang web MPA truyền thống	5
2.2	Luồng dữ liệu của các trang web SPA	5
2.3	ReactJS và những lợi ích khi sử dụng [5]	6
2.4	TypeScript là phiên bản mở rộng của JS có hỗ trợ kiểm tra kiểu tĩnh	7
2.5	Gitlab CICD [6]	8
2.6	Giao diện dashboard để xem những lỗi được gửi về	10
2.7	Giao diện xem lỗi gửi về chi tiết (Nửa trên màn hình)	11
2.8	Giao diện xem lỗi gửi về chi tiết (Nửa dưới màn hình)	11
2.9	Giao diện dashboard của Google Analytic (Nửa trên)	13
2.10	Giao diện dashboard của Google Analytic (Nửa dưới)	13
2.11	Màn hình thống kê thời gian thực tổng quan	14
2.12	Màn hình thống kê thời gian thực theo địa lý	14
2.13	Màn hình thống kê thời gian thực theo sự kiện	15
2.14	Đoạn mã hiện thực	17
2.15	Kết quả chạy 2 test case	17
2.16	RESTful API [7]	21
2.17	Mysql	22
2.18	Cassandra	23
2.19	Redis[13]	25
2.20	Active MQ[15]	27
2.21	Point-to-point	28
2.22	Publisher-Subscriber	28
2.23	Vai trò Kafka	30
2.24	Giao tiếp giữa các hệ thống	30
2.25	Kiến trúc của Kafka	31
2.26	Giới thiệu về Docker	32

2.27 Reverse Proxy[20]	33
3.1 Xác thực dựa trên Session và Cookies [21]	34
3.2 Xác thực dựa trên token[22]	35
3.3 Quá trình tạo đơn hàng	37
3.4 Chiến lược Cache Aside[14]	38
3.5 Hình thức thanh toán	40
3.6 Ví dụ về HyperLoglog	42
3.7 Flow docker	43
3.8 Dockerfile OrderService	43
3.9 Docker compose OrderService	44
3.10 Kết quả	45
3.11 Cấu hình của NGINX	46
3.12 Giao diện gợi ý địa điểm khi người dùng nhập địa chỉ	47
4.1 Kiến trúc hệ thống	52
4.2 Sơ đồ Use case	55
4.3 Trạng thái đơn hàng	67
4.4 Trạng thái kiện hàng	69
4.5 Lược đồ ERD	71
5.1 Giới thiệu về Gitlab	79
5.2 Giao diện đăng ký dành cho người muốn gửi đơn hàng	80
5.3 Giao diện đăng nhập dành cho actor của hệ thống	80
5.4 Giao diện điền thông tin để lên đơn hàng	81
5.5 Thông tin xác nhận đơn hàng muốn tạo	81
5.6 Giao diện list yêu cầu của người gửi (Rỗng)	82
5.7 Giao diện list yêu cầu của người gửi (Có yêu cầu)	83
5.8 Giao diện xem danh sách các đơn hàng nhỏ được tách từ yêu cầu	83
5.9 Giao diện chỉnh sửa thông tin của người gửi	84
5.10 Giao diện chọn những đơn hàng đã giao cần thanh toán	84
5.11 Giao diện chọn phương thức thanh toán	85
5.12 Giao diện xác nhận lại tổng số tiền và phương thức thanh toán	85
5.13 Giao diện hệ thống thanh toán trung gian ZaloPay	86
5.14 Mail thông báo tình trạng đơn hàng của hệ thống	86
5.15 Giao diện tìm những yêu cầu được gán	87
5.16 Giao diện xác nhận lại thông tin yêu cầu muốn nhận	87
5.17 Giao diện xem thông tin đơn hàng đang đến lấy	88
5.18 Giao diện xác nhận lại thông tin đơn hàng đang muốn nhận	88
5.19 Giao diện xem thông tin các đơn hàng đang ở trong xe	89
5.20 Giao diện báo cáo vấn đề gãy sứt cỗ	89
5.21 Giao diện xem lịch sử giao nhận hàng	90
5.22 Giao diện nhập ID của tài xế để nhập các đơn hàng trong xe tài xế vào kho	90
5.23 Nhập ID của khách hàng để nhập đơn vào kho	91

5.24 Cho phép nhập khối lượng mà khách muốn nhập vào kho	91
5.25 Xác nhận nhập hàng trong xe tài xế vào kho	92
5.26 Nhập hàng thành công và cho thủ kho có thể tài phiếu nhập kho	92
5.27 Format mẫu của 1 đơn nhập/xuất kho	93
5.28 Danh sách các đơn hàng trong kho	93
5.29 Lịch sử nhập/xuất kho	94
5.30 Thông tin chi tiết của một lần nhập/xuất đơn hàng (Cho phép tải pdf về)	94
5.31 Xem danh sách các tài xế trong hệ thống	95
5.32 Xem danh sách các kho và thủ kho phụ trách của hệ thống	96
5.33 Xem thống kê đơn hàng và số tiền khách hàng đã thanh toán	96
5.34 Thêm tài xế vào hệ thống	97
5.35 Thêm kho vào hệ thống	97
5.36 Thêm thủ kho vào hệ thống	98
5.37 Giao diện trả lời những yêu cầu cần hỗ trợ từ khách hàng	98
5.38 Giao diện xem lại những yêu cầu hỗ trợ đã trả lời	99
5.39 Giao diện xem những sự cố báo cáo từ tài xế	99
5.40 Giao diện gán lại trạng thái cho những đơn có sự cố	100
5.41 Giao diện đăng nhập cho người nhận	101
5.42 Giao diện không tìm thấy mã yêu cầu	101
5.43 Giao diện tìm thấy mã yêu cầu	102
5.44 Giao diện nhập mã OTP nếu chọn theo dõi yêu cầu bằng số điện thoại . .	102
5.45 Giao diện các yêu cầu và đơn hàng của người nhận	103
 6.1 Kiểm thử với API đăng nhập	 105
6.2 Kết quả kiểm thử với API đăng nhập	105
6.3 Kiểm thử với API đăng ký	106
6.4 Kết quả kiểm thử với API đăng ký	106
6.5 Kiểm thử với API thêm kho	107
6.6 Kết quả kiểm thử với API thêm kho	107
6.7 Kiểm thử với API thêm tài xế	108
6.8 Kết quả kiểm thử với API thêm tài xế	108
6.9 Kiểm thử với API thêm thủ kho	109
6.10 Kết quả kiểm thử với API thêm thủ kho	109
6.11 Kiểm thử với API thêm phương tiện	110
6.12 Kết quả kiểm thử với API thêm phương tiện	110
6.13 Kiểm thử với API đổi mật khẩu	111
6.14 Kết quả kiểm thử với API đổi mật khẩu	111
6.15 Kiểm thử với API lấy phí vận chuyển	112
6.16 Kết quả kiểm thử với API lấy phí vận chuyển	112
6.17 Kiểm thử với API lấy tài xế liên tỉnh ở kho	113
6.18 Kết quả kiểm thử với API lấy tài xế liên tỉnh ở kho	113
6.19 Kiểm thử với API lấy thông tin tài xế	114
6.20 Kết quả kiểm thử với API lấy thông tin tài xế liên	114
6.21 Kiểm thử với API lấy danh sách tài xế ở kho	115

6.22 Kết quả kiểm thử với API lấy danh sách tài xế ở kho	115
6.23 Kiểm thử với API lấy danh sách tài xế	116
6.24 Kết quả kiểm thử với API lấy danh sách tài xế	116
6.25 Kiểm thử với API lấy danh sách thủ kho	117
6.26 Kết quả kiểm thử với API lấy danh sách thủ kho	117
6.27 Kiểm thử với API lấy danh sách kho	118
6.28 Kết quả kiểm thử với API lấy danh sách kho	118
6.29 Kiểm thử với API lấy thông tin người dùng	119
6.30 Kết quả kiểm thử với API lấy thông tin người dùng	119
6.31 Kiểm thử với API lấy danh sách phương tiện chưa sử dụng	120
6.32 Kết quả kiểm thử với API lấy danh sách phương tiện chưa sử dụng	120
6.33 Kiểm thử với API cập nhật thông tin người dùng	121
6.34 Kết quả kiểm thử với API cập nhật thông tin người dùng	121
6.35 Kiểm thử với API cập nhật trạng thái tài xế	122
6.36 Kết quả kiểm thử với API cập nhật trạng thái tài xế	122
6.37 Kiểm thử với 10 users đồng thời	123
6.38 Kiểm thử với 20 users đồng thời	124
6.39 Kiểm thử với 30 users đồng thời	124
6.40 Kiểm thử với 40 users đồng thời	125
6.41 Kiểm thử với 50 users đồng thời	125

Giới thiệu đề tài

1.1 Tổng quan đề tài

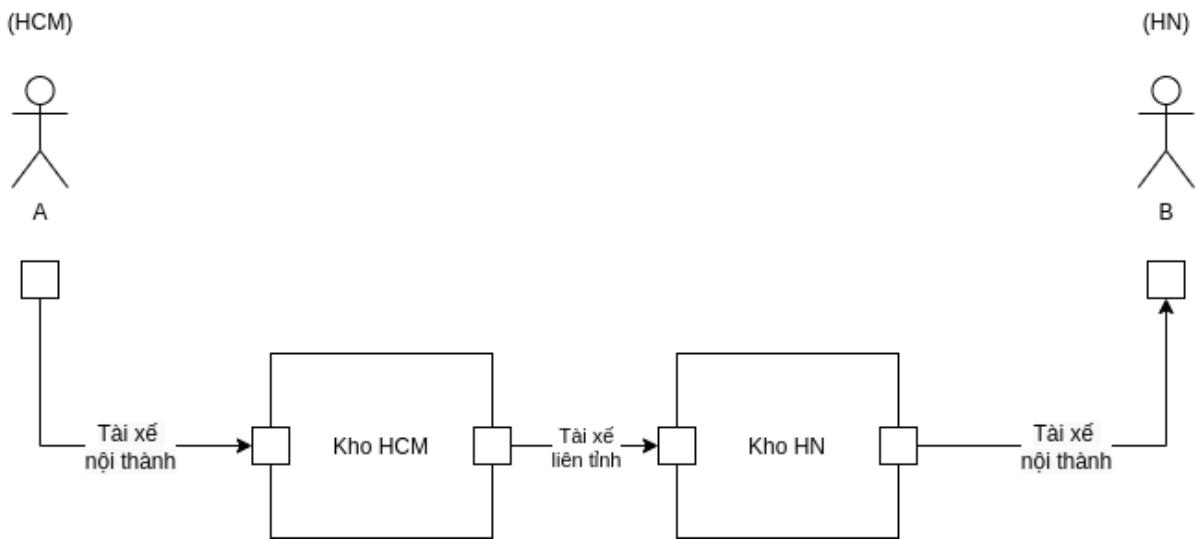
Nhu cầu vận chuyển hàng hóa đã có từ rất lâu. Cách thức vận chuyển hàng hóa phổ biến trước đây là thông qua đường bộ và đường sắt. Tuy vậy, trong thời đại 4.0 này thì hầu hết mọi dịch vụ đều đang được tự động hóa. Dịch vụ vận chuyển hàng hóa cũng không phải là ngoại lệ. Hiện nay có rất nhiều hệ thống vận chuyển hàng hóa đã được xây dựng. Chẳng hạn ở trong nước có thể kể đến Giao hàng nhanh [1], Giao hàng tiết kiệm [2] vv... và ở ngoài nước là FedEx hay DHL. Những hệ thống hỗ trợ tự động hóa quy trình logistic và giao nhận hàng hóa là không ít. Tuy vậy việc có thêm một hệ thống sẽ giúp cho người dùng có nhiều lựa chọn để cân nhắc phù hợp với bản thân. Đặc biệt hệ thống nhóm xây dựng sẽ đào sâu hơn về quá trình vận chuyển hàng hóa **liên tỉnh**, tập trung vào những đơn hàng và kiện hàng có khối lượng lớn cùng với đó là khả năng chịu tải cao, phục vụ được nhiều khách hàng trong cùng thời điểm.

Đề tài sẽ được thực hiện trên nền tảng web. Ứng dụng web app sẽ cung cấp các dịch vụ để quy trình vận chuyển hàng hóa liên tỉnh trở nên dễ dàng hơn cho người gửi/nhận, tài xế, chủ kho và quản lý.

Ứng dụng sẽ bắt đầu từ việc người dùng gửi yêu cầu muốn giao món hàng bằng cách điền các thông tin cần thiết trên website hệ thống (Tên, địa chỉ bên gửi/nhận, chủ yêu bên nhận và bên gửi sẽ khác tỉnh và cách nhau xa). Sau đó, tài xế sẽ được hệ thống phân công đến những nơi lấy hàng theo yêu cầu bên gửi và tiến hành di lấy hàng đến khi đầy xe.

Sau đó, tài xế sẽ trở lại kho tập trung và gỡ hàng ra đặt trong kho. Quá trình này sẽ được lặp lại cho đến khi tổng hàng hóa thu gom về kho có thể chất đầy một xe container để tiến hành vận chuyển những kiện hàng đó đi liên tỉnh. Xe container đầy sẽ đi xuyên tỉnh và đến kho tập trung ở tỉnh ấy, gỡ hàng ra và sẽ có xe nội thành chuyển trực tiếp đến người nhận theo yêu cầu. Bên liên quan trong từng giai đoạn sẽ trực tiếp cập nhật tình trạng của đơn hàng để người dùng có thể lấy mã đơn hàng và xem trạng thái hiện

tại cửa của nó (Đang chờ, đang giao hàng, hoàn thành, etc.). Mỗi khi hàng được chuyển giao đều sẽ có biên bản giao nhận hàng được tạo ra cho mỗi bên.



Hình 1.1: Luồng cơ bản giao nhận hàng hóa

1.2 Mục tiêu và phạm vi đề tài

Mục tiêu của đề tài là xây dựng một hệ thống vận chuyển hàng hóa liên tỉnh một cách có hiệu quả, giảm thiểu tối đa các sai sót có thể xảy ra trong quá trình giao nhận hàng hóa, cung cấp cho người dùng, tài xế, thủ kho, quản lý một giao diện trực quan, sinh động, dễ tương tác, giúp cho việc đặt hàng, nhận hàng, quản lý trở nên đơn giản và tin cậy hơn. Người dùng có thể tạo đơn hàng một cách rất dễ dàng và nhanh chóng. Hệ thống cho phép người dùng có nhiều lựa chọn về việc tự vận chuyển hàng đến kho hay nhân viên bên kho sẽ đến lấy nhằm giảm thiểu tối đa khó khăn khi vận chuyển một lượng hàng hóa lớn của người dùng. Sau khi đặt hàng người dùng có thể dễ dàng theo dõi quá trình hàng hóa được vận chuyển như thế nào, đã đến đâu, được vận chuyển bởi ai. Ngoài ra hệ thống còn tự động gửi thông báo qua mail để người dùng có thể biết chính xác trạng thái đơn hàng của mình. Hệ thống tích hợp chức năng thanh toán, giúp người dùng có thể theo dõi các đơn hàng đã hoàn thành và dễ dàng thanh toán qua các kênh trung gian như ZaloPay, Momo chỉ bằng một vài thao tác đơn giản. Tài xế có thể dễ dàng nhận đơn thông qua ứng dụng, quản lý các đơn hàng đang vận chuyển một cách tiện lợi, nhanh chóng. Từ lâu, việc quản lý đơn hàng, tài xế, phiếu xuất kho cũng là một vấn đề khá khó khăn, thì giờ đây hệ thống cung cấp khả năng lưu trữ tin cậy, đồng thời việc tìm kiếm rất nhanh, biểu đồ thống kê trực quan sinh động giúp cho nhân viên quản lý có thể dễ dàng theo dõi quá trình hoạt động của doanh nghiệp để có thể đưa ra các thay đổi, điều chỉnh khi cần thiết. Việc xây dựng một hệ thống quản lý vận chuyển hàng hóa liên tỉnh như vậy đặt ra một số bài toán cần phải giải quyết như:

- Công nghệ sử dụng là gì?
- Kiến trúc hệ thống như thế nào để có thể dễ dàng mở rộng khi có nhu cầu?
- Tổ chức lưu trữ dữ liệu thế nào để có thể truy xuất nhanh chóng, tránh làm ảnh hưởng không tốt đến trải nghiệm của người dùng?
- Quá trình giao nhận hàng hóa của tài xế với người dùng hay của tài xế đối với thủ kho làm thế nào để quản lý và lưu trữ hiệu quả, tránh làm mất dữ liệu về sau?
- Việc phân phối đơn hàng cho tài xế đi giao/lấy làm thế nào để hạn chế tối đa chi phí bỏ ra cho doanh nghiệp cũng như người dùng?
- Làm thế nào để hệ thống có thể chịu tải cao, có thể xử lý lượng lớn các yêu cầu đồng thời?
- Làm thế nào để xác định được con đường vận chuyển giữa kho nguồn - kho đích là ngắn nhất để giảm thiểu chi phí?
- Bảo mật cho ứng dụng cũng như xác thực người dùng như thế nào cho hiệu quả?

1.3 Cấu trúc luận văn

Phần còn lại của luận văn sẽ được tổ chức như sau:

- Chương 2: Trình bày cơ sở lý thuyết về các công nghệ sử dụng
- Chương 3: Nêu ra các bài toán và cách giải quyết cho từng bài toán
- Chương 4: Trình bày kiến trúc, đặc tả các chức năng, thiết kế cơ sở dữ liệu cho hệ thống
- Chương 5: Trình bày chi tiết về hiện thực hệ thống và kết quả đạt được
- Chương 6: Trình bày kết quả kiểm thử hệ thống
- Chương 7: Tổng kết, đánh giá kết quả

2

Cơ sở lý thuyết và công nghệ

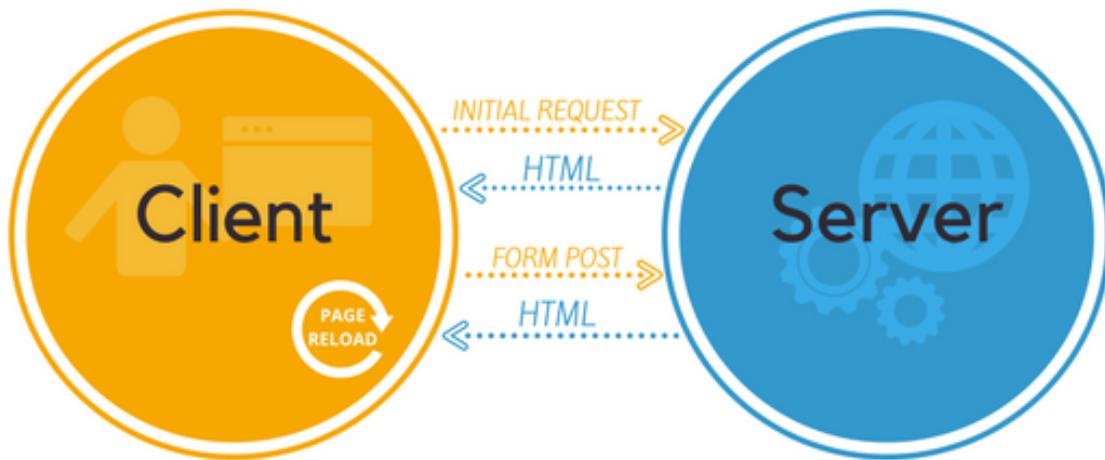
Trong quá trình làm luận văn, khi tìm hướng giải quyết cho từng bài toán, nhóm đã nghiên cứu và tìm tòi ra được những nguyên lý đằng sau cũng như những công nghệ giúp nhóm giải quyết được những khía cạnh gắp phải từ phía người dùng cho đến máy chủ. Sau đây, nhóm xin trình bày về các cơ sở lý thuyết và công nghệ chính mà nhóm sẽ sử dụng theo 2 góc nhìn là Front-end và Back-end

2.1 Front-end

2.1.1 SPA

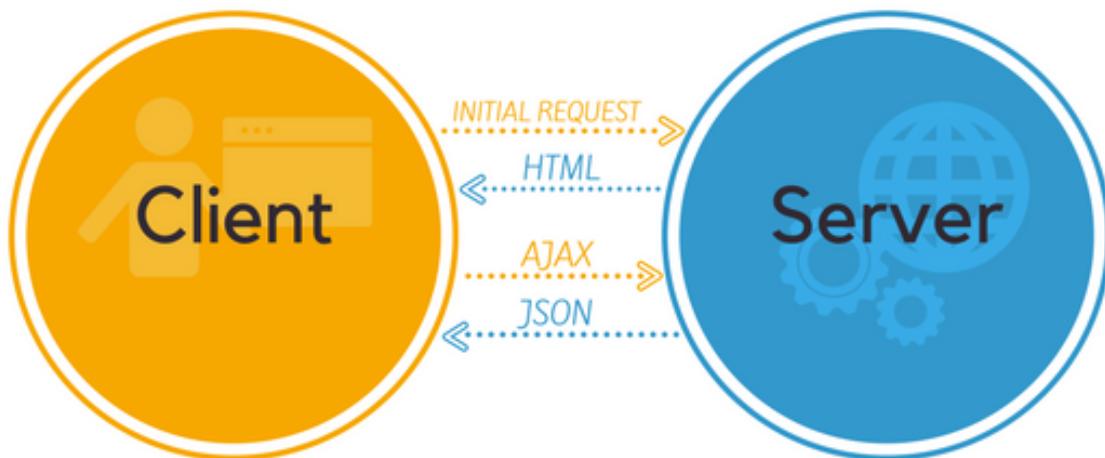
Web sẽ thiết kế theo kiểu Single Page Application (SPA) thay vì theo kiểu truyền thống là Server side rendering. Ưu điểm của cách thiết kế này là web sẽ có trải nghiệm như khi dùng ứng dụng trên điện thoại hoặc máy tính. Mỗi lần thực hiện một thao tác hay chuyển trang trên web thì trình duyệt sẽ không phải tải lại trang nữa. Việc này được thực hiện bằng cách sử dụng AJAX để lấy dữ liệu một cách bất đồng bộ từ phía máy chủ mỗi khi có một event do người dùng thực hiện trên trình duyệt. Sau đó, ta sẽ sử dụng dữ liệu lấy được (thường là dưới dạng JSON) để đổ giao diện lên trình duyệt hiển thị cho người dùng [3].

Traditional page lifecycle



Hình 2.1: Luồng dữ liệu của các trang web MPA truyền thống

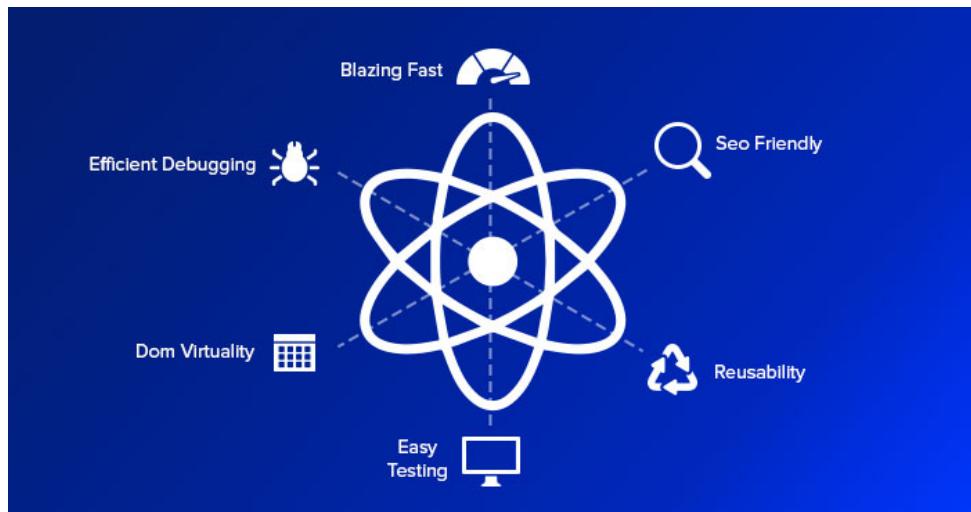
SPA lifecycle



Hình 2.2: Luồng dữ liệu của các trang web SPA

2.1.2 ReactJS

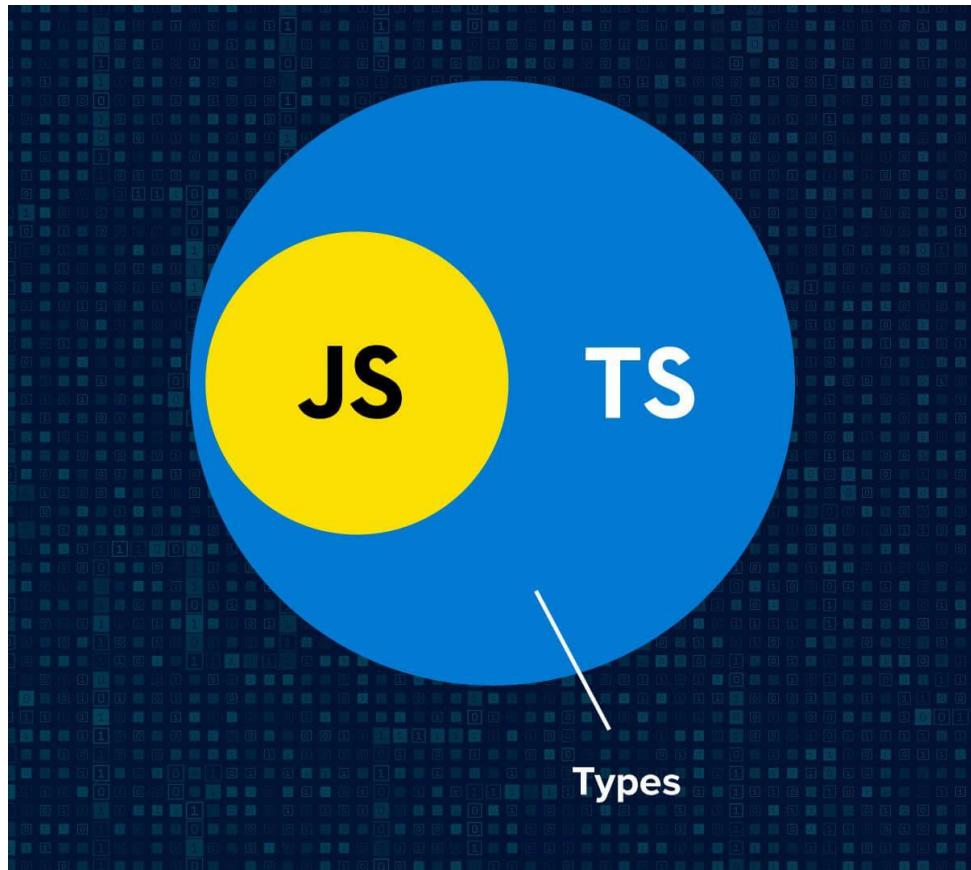
ReactJS [4] được phát triển bởi Facebook là một thư viện rất phổ biến để phát triển các ứng dụng web SPA hiện nay. React cho phép người lập trình có thể chia ứng dụng thành các component và tái sử dụng, nhờ vậy mà tiết kiệm được thời gian lập trình. Hơn nữa, cơ chế DOM ảo (**Virtual DOM**) mới chính là điểm đặc biệt hơn cả của thư viện ReactJS. Cập nhật cây DOM là một tác vụ khá tốn chi phí. Nếu như để lập trình một cách thông thường, lập trình viên tự quản lí DOM bằng các Native API của môi trường web thì sẽ dễ gây ra vấn đề về hiệu suất. Chính vì vậy ReactJS sẽ giúp ta cập nhật DOM bằng cách quản lí ngầm một cây DOM ảo và sẽ đổi chiều những thay đổi so với các phiên bản DOM ảo trước đó để cập nhật lên cây DOM thật, giúp ứng dụng web tăng hiệu suất. Chính vì những lí do trên mà nhóm quyết định sẽ sử dụng thư viện này để phát triển luận văn này.



Hình 2.3: ReactJS và những lợi ích khi sử dụng [5]

Bên cạnh đó, ReactJS cho phép người lập trình có thể viết bằng 2 ngôn ngữ: Javascript và Typescript. Javascript đã ra đời và được sử dụng từ rất lâu. Hiện nay, đây là ngôn ngữ được rất nhiều lập trình viên sử dụng, cả trong việc phát triển ứng dụng phía client-side lẫn phía server-side. Javascript là một ngôn ngữ prototype-based hỗ trợ cả lập trình hướng đối tượng và lập trình hàm, các tính năng được bảo trì và phát triển liên tục. Tuy vậy, quá linh động cũng là một điểm yếu của ngôn ngữ này. Javascript không kiểm tra ràng buộc kiểu khi chúng ta viết mã nguồn. Người lập trình có thể sử dụng một biến mà không cần khai báo kiểu, khiến cho việc truyền tham số hay khi tương tác các biến khác kiểu trở nên khó kiểm soát. Lấy ví dụ đơn giản như biểu thức "`1`" + 1 sẽ là hợp lệ và không bị bắt lỗi trong quá trình kiểm tra kiểu tĩnh khi chúng ta lập trình. Hay khi ta định nghĩa một hàm với các tham số cần truyền vào, do Javascript không hỗ trợ khai báo kiểu nên khi gọi hàm ta có thể truyền bất cứ tham số với bất cứ kiểu nào, làm cho hàm có thể thực

thi ngoài ý muốn. Chính vì vậy, TypeScript ra đời để giải quyết vấn đề của Javascript kể trên. TypeScript yêu cầu người dùng định nghĩa kiểu rõ ràng cho các biến, kiểu trả về của hàm và bắt lỗi các biểu thức mà có sự tương tác của các biến không tương thích kiểu. Nhờ vậy, Sử dụng TypeScript sẽ giúp cho mã nguồn của dự án dễ đọc, minh bạch, dễ bảo trì và phát triển hơn rất nhiều.



Hình 2.4: TypeScript là phiên bản mở rộng của JS có hỗ trợ kiểm tra kiểu tĩnh

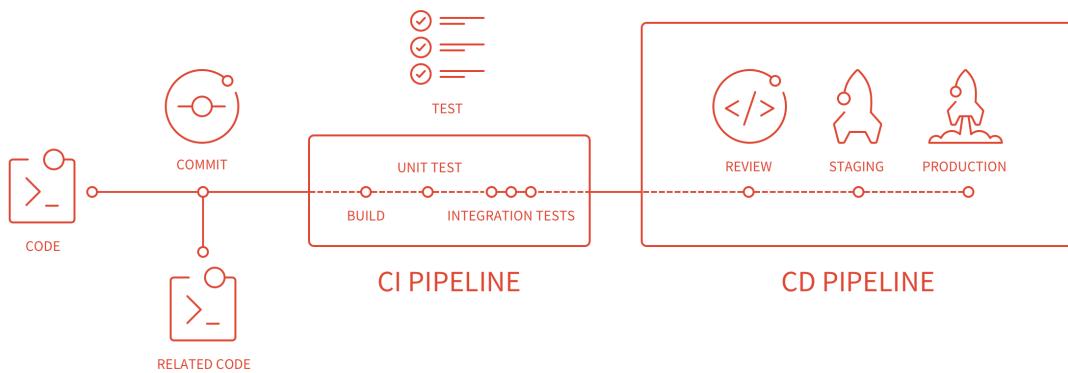
2.1.3 Continuous integration / Continuous deployment (CI/CD)

Do web SPA sẽ trả về tập tin html tĩnh cho người dùng tương tác (dữ liệu hiển thị được cập nhật bằng cách tương tác với API phía server) nên ta có thể host website trên những dịch vụ cho host static web miễn phí. Nhóm sẽ chọn dịch vụ host web tĩnh miễn phí của gitlab. Ưu điểm của gitlab là vừa có thể giúp nhóm quản lí mã nguồn cũng như cung cấp dịch vụ Continuous integration (CI) và continuous delivery (CD) giúp có thể deploy ứng dụng một cách tự động mỗi khi ta push code lên gitlab.

Cụ thể, CI/CD giúp chúng ta chỉ phải code, chia branch đẩy lên git. Các công đoạn như unit test, deployment sẽ được gitlab-runner thực hiện một cách tự động mỗi khi có

thay đổi mã nguồn. Bất kì ứng dụng được phát triển hiện nay đều cần phải qua khâu Unit Test do chính người lập trình viết trước khi được chuyển qua cho bộ phận kiểm thử phần mềm. Khi ta đẩy mã nguồn lên git mà quên chạy lệnh để kiểm thử unit test ở dưới máy local thì gitlab-runner sẽ giúp nhóm lập trình chạy các test case đó và sẽ thông báo trên giao diện git nếu có test case bị fail. Sau giai đoạn chạy các unit test và đã qua hết các test case thì tất nhiên, ứng dụng phải được deploy lên các môi trường (staging, production, etc). Việc phải SSH vào server host ứng dụng, sau đó tải mã nguồn mới nhất về, build ứng dụng ra các file tối ưu để chạy là khá tốn thời gian. Các thao tác trên thực chất cũng chỉ là chạy các câu lệnh trên terminal, vậy tại sao không để hệ thống tự động chạy những câu lệnh đó? Ta chỉ cần liệt kê các câu lệnh của một hành động cần phải thực hiện theo đúng thứ tự và hệ thống CI/CD của gitlab sẽ tự động chạy các câu lệnh đó cho chúng ta.

1



Hình 2.5: Gitlab CICD [6]

2.1.4 Những thư viện nổi bật giúp hiện thực chức năng hệ thống

Một số lập trình viên rất thích việc xây dựng mọi thứ từ đầu và hạn chế sử dụng thư viện càng nhiều càng tốt. Tuy vậy, có những tính năng đã được một số thư viện hỗ trợ đầy đủ và đã được kiểm thử qua hàng ngàn người dùng. Khi đó, ta nên sử dụng thư viện để tiết kiệm thời gian phát triển cũng như có thể sử dụng ngay lập tức tính năng chất lượng đã qua kiểm thử. Nhóm sẽ liệt kê một số thư viện nổi bật (Ngoài thư viện chính React) mà nhóm đã dùng để hiện thực một vài chức năng của hệ thống:

Nhóm sẽ giới thiệu một số thư viện được áp dụng trong việc viết code phía client mà nhóm thấy nổi bật để trình bày:

- **chart.js (version 2.9.4):** Dùng để hiện thực chức năng tạo biểu đồ thống kê của actor quản lý.

- **date-fns (version 2.21.1):** Dùng để thực hiện việc tính toán ngày tháng như lấy thời gian bắt đầu / kết thúc của ngày, tăng giảm số ngày vv...
- **antd (version 4.9.4):** Thư viện CSS và component tương thích với React.
- **json-bigint (version 1.0.0):** Giúp parse ra được ID lớn của đơn hàng thành chuỗi dạng string.
- **@react-pdf/renderer (version 2.0.8):** Giúp tạo ra template cho phiếu nhập/xuất kho dưới dạng pdf.
- **gzipper (version 4.5.0):** Giúp tạo ra những file gzip của những file static đã build ra. Như vậy sẽ giảm dung lượng đường truyền mạng cần thiết để tải file và trang web sẽ hiển thị nhanh hơn vào lần đầu tiên truy cập.

2.1.5 Giám sát lỗi hệ thống bằng dịch vụ Sentry

Giới thiệu về Sentry

Một hệ thống không những phải hoạt động tốt mà còn cần phải có hệ thống ghi log và báo cáo lỗi không mong muốn xảy ra từ người dùng. Thay vì phải tự hiện thực lại hệ thống như trên thì nhóm đã quyết định sử dụng một dịch vụ trực tuyến khá phổ biến và nổi tiếng hiện nay: Sentry. Về cơ bản, Sentry giúp chúng ta theo dõi lại những lỗi xảy ra bên trong ứng dụng trong quá trình người dùng sử dụng. Thông thường khi lỗi xảy ra, người dùng sẽ có xu hướng tắt ứng dụng và mở lại chứ không báo cáo lại cho bên phía đội ngũ phát triển. Sentry sẽ tự động gửi báo cáo chi tiết về lỗi đó một cách tự động và nhanh chóng. Giúp cho người lập trình có thể nhanh chóng cập nhật bản vá sửa lỗi đó trước khi nhiều người dùng bị lỗi tương tự.

Áp dụng sentry cho hệ thống giao hàng liên tỉnh

Nhận thấy tác dụng hữu ích và đồng thời muốn học thêm những công nghệ đang thịnh hành ngoài thị trường. Nhóm chúng em đã thử áp dụng Sentry vào hệ thống giao hàng liên tỉnh này. Sau đây là một số kết quả mà nhóm đã thu được từ việc áp dụng Sentry:

Type	Title	Status	Last Seen	Graph	Events	Users	Assignee
TypeError	render(main.2894aeef2d28aa1486fb8.hot-update)	New Issue	3d ago 3d old		1	1	
TypeError	c/modules/stockkeeper/StockList]	New Issue	3d ago 3d old		8	1	
TypeError	c/modules/stockkeeper/StockList]	New Issue	3d ago 3d old		1	1	
TypeError	StockList/main.chunk]	New Issue	3d ago 3d old		1	1	
TypeError	sendSentryError(main.chunk)	New Issue	3d ago 3d old		4	1	
TypeError	h/modules/stockkeeper/StockAccount]	New Issue	3d ago 3d old		1	1	
TypeError	h/modules/driver/DriverAccount]	New Issue	3d ago 3d old		1	1	
TypeError	sendSentryError(main.2f5d51440d6a7b0ffaf.hot-update)	New Issue	3d ago 3d old		1	1	
ReferenceError	Direct(main.7fec33745df7ff62fd.hot-update)	New Issue	3d ago 3d old		2	1	
ReferenceError	Direct(main.7fec33745df7ff62fd.hot-update)	New Issue	3d ago 3d old		1	1	
ReferenceError	Direct(main.345a1fc983335f359bb6d.hot-update)	New Issue	3d ago 3d old		1	1	

Hình 2.6: Giao diện dashboard để xem những lỗi được gửi về

Có thể thấy giao diện trên liệt kê ra những lỗi chưa được xử lý của ứng dụng khi người dùng tương tác. Ở giao diện dashboard đó ta có thể chọn một hoặc nhiều lỗi được gửi về và đánh dấu đã xử lí (Resolve) sau khi đã thực hiện bản vá sửa lỗi hoặc bỏ qua (Ignore) nếu như lỗi đó không cần phải giải quyết. Những lỗi đã được xử lí sẽ bị mất đi trong danh sách lỗi ở màn hình dashboard đó.

Ngoài ra, khi click vào liên kết của một lỗi cụ thể. Sentry sẽ đưa ta đến một giao diện để xem một cách chi tiết thông tin về lỗi đó:

Chương 2

Cơ sở lý thuyết và công nghệ

The screenshot shows a detailed error report for a `TypeError` in the `render` function. The error message is `((intermediate value)) is not a function`. The stack trace shows the error occurring in `main.2894eaef28aa1486fb8.hot-update.js` at line 262. The environment is `Production` on a `Linux` machine with `Chrome 91.0.4472`. The report includes a timeline of events, ownership rules, and linked issues.

Hình 2.7: Giao diện xem lỗi gửi về chi tiết (Nửa trên màn hình)

The screenshot shows a detailed error report for a TypeScript error. The error message is `'driverPhone' is assigned a value but never used`. The stack trace shows the error occurring in `src/modules/stockkeeper/StockHistory.tsx` at line 153:37. The environment is `Production` on a `Linux` machine with `Chrome 91.0.4472`. The report includes a timeline of events, ownership rules, and additional data like browser and operating system information.

Hình 2.8: Giao diện xem lỗi gửi về chi tiết (Nửa dưới màn hình)

Giao diện đó cung cấp cho người lập trình toàn bộ thông tin quan trọng để có thể bắt đầu việc debug và sửa lỗi. Có thể kể đến một số thông tin hữu ích như ngày xảy ra lỗi,

trình duyệt web, địa chỉ IP, OS của người dùng. Không những thế, Sentry còn liệt kê cho ta chi tiết lỗi xảy ra ở đoạn mã lệnh nào giúp cho người lập trình dễ dàng phát hiện đoạn mã lỗi để sửa. Ở ví dụ cụ thể trong hình trên. Ta có thể thấy được lỗi xảy ra tại địa chỉ IP 42.116.148.203, trình duyệt web Chrome và hệ điều hành Linux. Đoạn lệnh lỗi cũng được chỉ ra nằm ở tập tin nào và nguyên nhân xảy ra lỗi.

Tóm tắt lại, Sentry là một nền tảng rất phổ biến hiện nay giúp cho hệ thống chúng ta xây dựng có thể nhanh chóng phát hiện lỗi từ phía người dùng. Hầu như mọi dự án lớn đều đang sử dụng Sentry và hiệu quả nó mang lại là không phải bàn cãi.

2.1.6 Phân tích số lượng và hành vi người dùng với Google Analytics

Một website không chỉ cần có hệ thống giám sát lỗi không mong muốn từ người dùng mà cũng cần phải có một hệ thống dùng để thống kê, theo dõi lưu lượng truy cập và những tương tác của người dùng với trang web. Từ đó đội ngũ phát triển có thể đưa ra những quyết định và những bản cập nhật để phục vụ nhu cầu người dùng tốt hơn. Cũng tương tự như trên, thay vì phải tốn thời gian và công sức xây dựng hệ thống đó thì nhóm cũng quyết định sử dụng một dịch vụ thống kê rất phổ biến và được ưa chuộng hiện nay: **Google Analytics**. Đây là một dịch vụ được cung cấp miễn phí bởi Google.

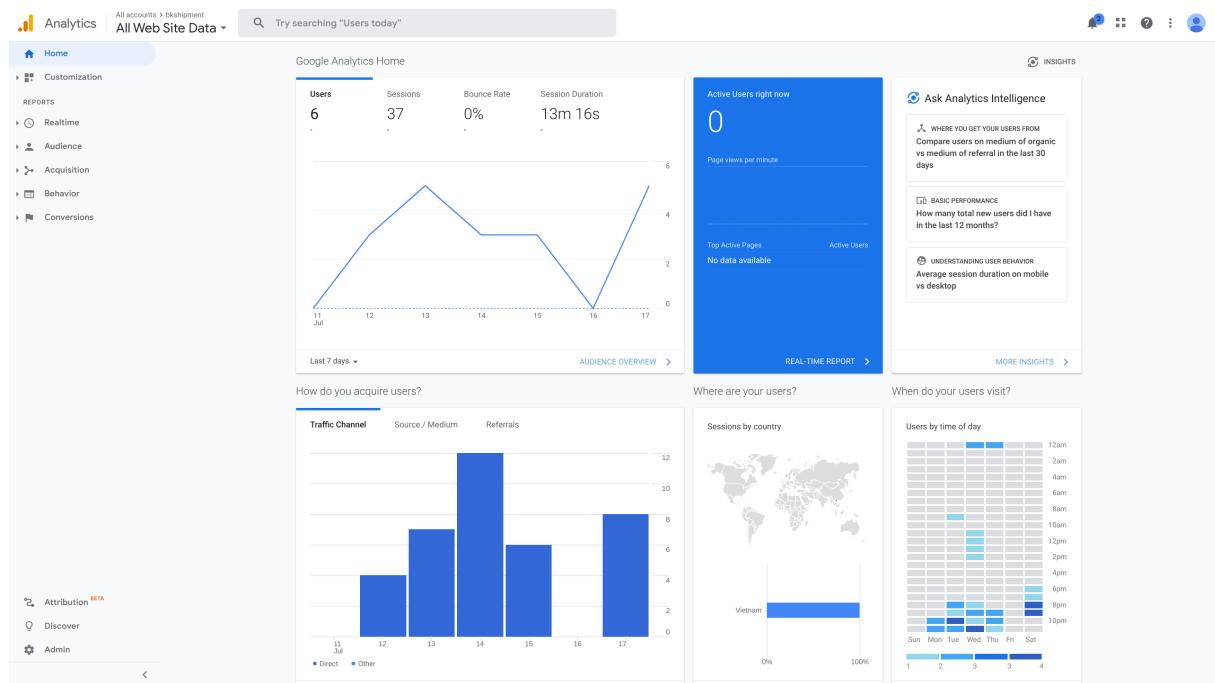
Áp dụng Google Analytics vào hệ thống

Nhận thấy đây là công nghệ rất phổ biến và hữu dụng. Nhóm đã quyết định sử dụng Google Analytics vào để có thể xem được chi tiết những thống kê người dùng truy cập đến trang web. Đây cũng là một cơ hội để học hỏi thêm một công nghệ mới được sử dụng bởi nhiều doanh nghiệp.

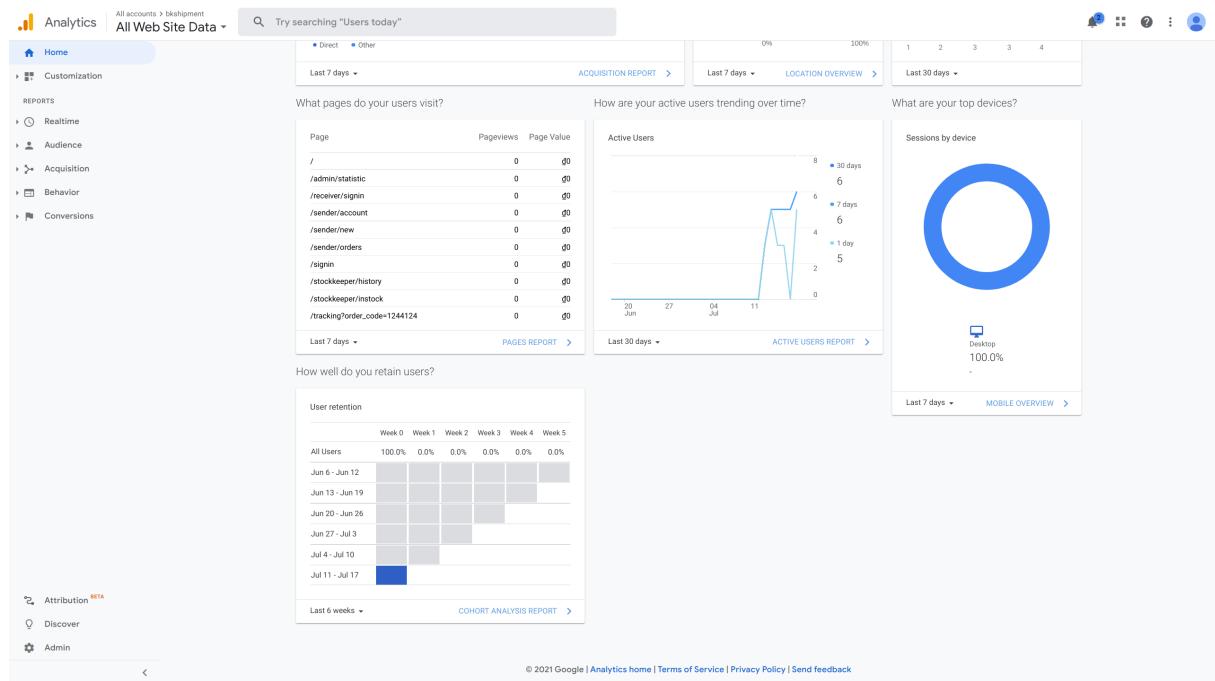
Sau đây là một số tính năng như trang dashboard, theo dõi và thống kê thời gian thực của người dùng của website do Google Analytics đã thu thập:

Chương 2

Cơ sở lý thuyết và công nghệ



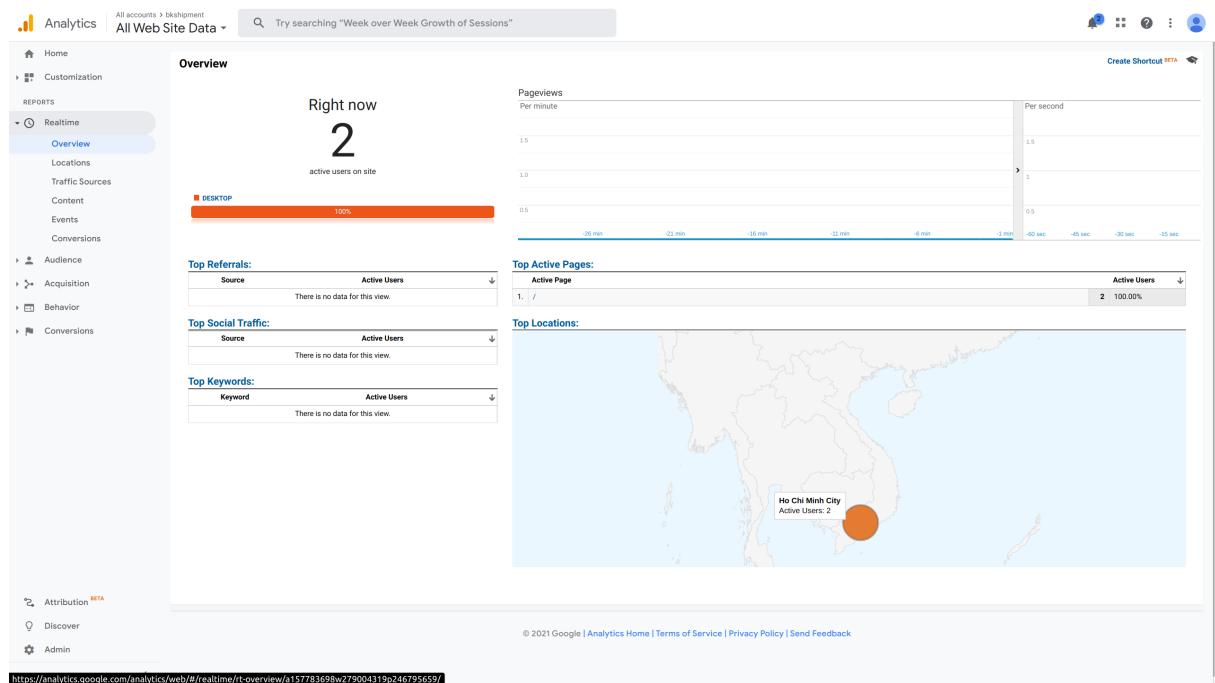
Hình 2.9: Giao diện dashboard của Google Analytic (Nửa trên)



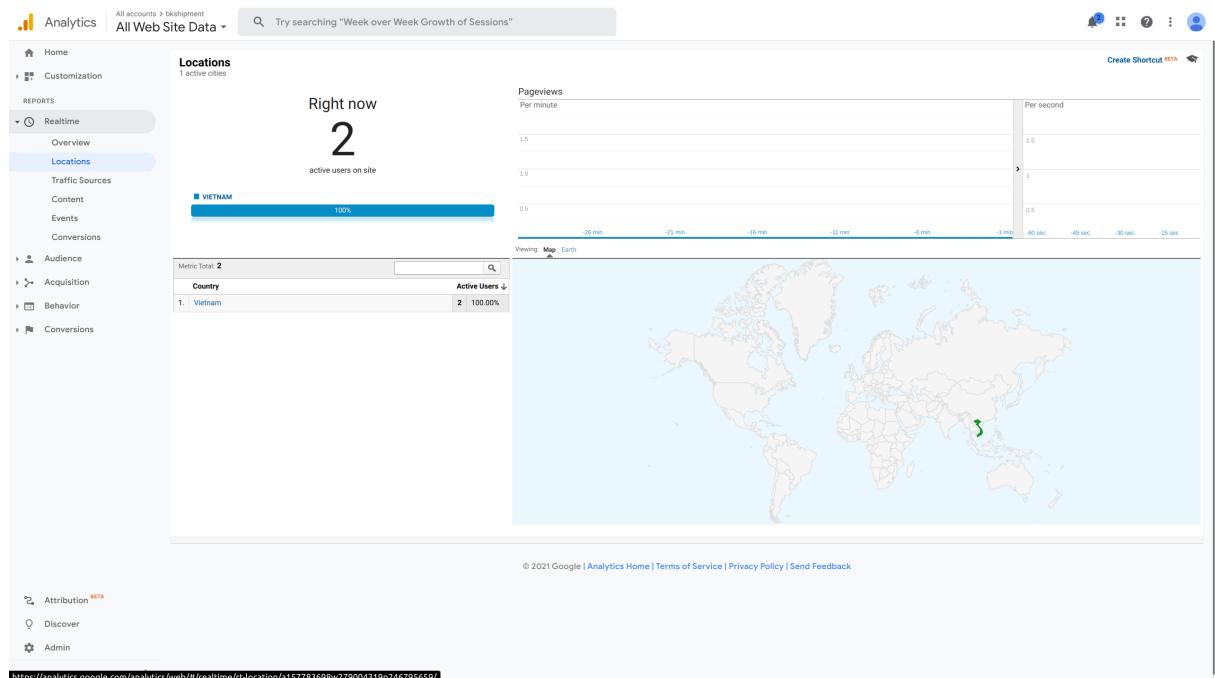
Hình 2.10: Giao diện dashboard của Google Analytic (Nửa dưới)

Chương 2

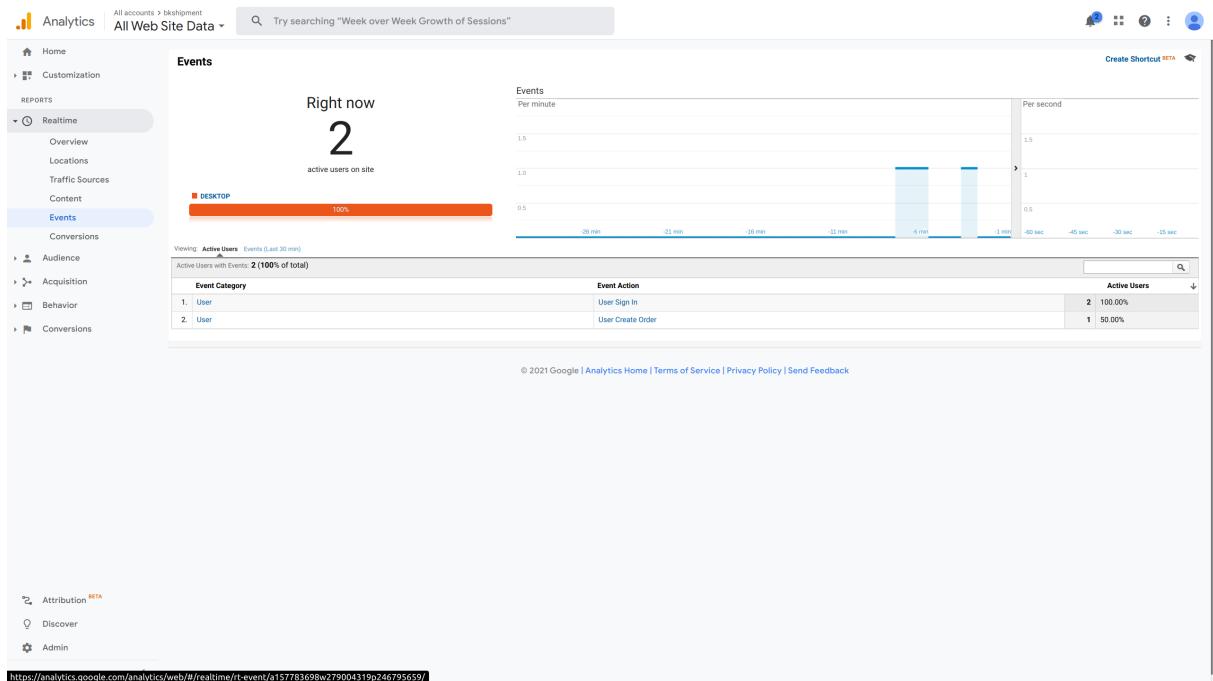
Cơ sở lý thuyết và công nghệ



Hình 2.11: Màn hình thống kê thời gian thực tổng quan



Hình 2.12: Màn hình thống kê thời gian thực theo địa lý



Hình 2.13: Màn hình thống kê thời gian thực theo sự kiện

Ở các màn hình kể trên, ta đều có thể thấy số lượng người dùng đang truy cập website hiện tại (Ở ví dụ trong hình là 2). Google Analytics cũng giúp chúng ta thống kê người truy cập là ở những nước nào (Ví dụ trong hình là ở Việt Nam) và thống kê những tác vụ mà người dùng thực hiện (Ở ví dụ trong hình ta có thể thấy có 2 tác vụ là người dùng đăng nhập và người dùng tạo yêu cầu gửi hàng). Nhờ những thống kê trên mà đội ngũ phát triển hệ thống có thể đưa ra những quyết định phát triển cho tương lai phù hợp hơn.

2.1.7 Front-end Unit testing

Trong quá trình xây dựng phần mềm thì việc đảm bảo chất lượng và qui trình kiểm thử phần mềm cũng là một công đoạn rất quan trọng. Không có phần mềm nào là không có lỗi. Chính vì vậy, việc kiểm thử phần mềm giúp hạn chế tối đa lỗi trước khi đưa phần mềm cho khách hàng. đương nhiên, ngoài đội ngũ xây dựng và phát triển phần mềm cũng cần một đội ngũ tester đảm bảo chất lượng sản phẩm. Tuy vậy, không có nghĩa là lập trình viên chỉ việc viết mã nguồn mà không phải thực hiện kiểm thử phần mềm. Lập trình viên ít nhất cũng nên viết kiểm thử đơn vị (Unit Test) để đảm bảo module mình viết ra ít lỗi nhất có thể và khi có sự thay đổi trong tương lai thì đảm bảo không gây lỗi cho các thành phần cũ của phần mềm.

Khái niệm về Unit Test

Unit Test là một loại kiểm thử phần mềm trong đó các đơn vị hay thành phần riêng lẻ của phần mềm được kiểm thử. Kiểm thử đơn vị được thực hiện trong quá trình phát

triển ứng dụng. Mục tiêu của kiểm thử đơn vị là cô lập một phần code và xác minh tính chính xác của đơn vị đó.

Lợi ích và đánh đổi của Unit Test

Thông thường, ở một vài qui trình phát triển phần mềm, đội ngũ kiểm thử phần mềm sẽ yêu cầu lập trình viên phải thực hiện Sanity Test (Kiểm thử sơ bộ) để chắc chắn rằng những chức năng chính của phần mềm hoạt động ổn định. Mỗi lần sanity test thường có từ vài chục cho đến vài trăm test case và những test case đó có thể xuất hiện lại sau mỗi bản cập nhật phần mềm. Việc phải kiểm đi kiểm lại những test case đó là điều cần thiết, tuy vậy nó khá tốn thời gian và nhảm chán. Chính vì vậy, nếu có thể viết unit test cho những test case đơn giản đó một cách tự động thì sẽ tiết kiệm được rất nhiều thời gian và đảm bảo tính chính xác nếu unit test của người lập trình viết ra có chất lượng tốt. Không những vậy, việc viết unit test sẽ giúp cho người lập trình cảm thấy tự tin hơn mỗi khi xây dựng một module mới hay chỉnh sửa module cũ. Nhờ vào những test case đã viết sẵn đó mà ta có thể chắc chắn sẽ không có lỗi liên quan xảy ra.

Tuy rằng việc viết unit test có rất nhiều lợi ích nhưng có cũng những đánh đổi sau:

- Tốn thời gian và tương đối khó khăn. Viết test tuy giảm bớt thời gian và công sức cho giai đoạn duy trì (maintain) phần mềm nhưng sẽ làm mất thời gian lúc đầu để viết một unit test tốt.
- Test pass không có nghĩa ứng dụng function của chúng ta chạy đúng hoàn toàn.
- Cũng đôi khi, test fail, nhưng ứng dụng function vẫn chạy hoàn toàn bình thường. Việc viết test không tốt như vậy sẽ còn nguy hiểm hơn do nó tạo cho người lập trình một suy nghĩ rằng phần mềm viết ra không có lỗi, trong khi thực sự thì có.

Viết unit test cho client-side code của hệ thống

Có thể hiểu nôm na, viết unit test cho Front-end thì ngoài viết test cho những hàm tính toán, ta còn có thể viết test để kiểm thử giao diện và hành vi khi một số sự kiện người dùng xảy ra. Chẳng hạn sau đây là một số thứ ta có thể kiểm thử ở client-side code:

- Kiểm thử xem một node trong cây DOM có xuất hiện (tồn tại) trong giao diện sau khi đã render component hay chưa. Chẳng hạn kiểm thử xem giao diện đăng nhập có tồn tại một nút với dòng chữ là "Đăng nhập" hay không. Hay nút đó có được áp dụng đúng class hay không để có style css đúng như mong muốn.
- Kiểm thử xem sau khi người dùng đã nhập đủ thông tin đăng nhập và ấn nút đăng nhập, người dùng có chuyển đến trang màn hình chính hay không.
- Chẳng hạn giao diện ta có một carousel có yêu cầu cứ mỗi 2 giây sẽ chuyển sang hình kế tiếp. Ta có thể viết unit test để kiểm chứng điều đó.

Trên đây chỉ là một số ví dụ về những chức năng ta có thể thực hiện bằng việc viết unit test cho front-end code của ứng dụng.

Nhóm sẽ ví dụ 2 test case đơn giản mà nhóm đã viết để demo việc áp dụng việc viết unit test vào hệ thống. Nhóm sẽ dùng 2 thư viện hỗ trợ cho việc viết test cho những component của React khá phổ biến hiện nay: **Jest** và **React Testing Library**. 2 test case đơn giản mà nhóm sẽ viết sẽ như sau:

- Kiểm thử xem component để hiển thị khi data rỗng có dòng chữ **Không có dữ liệu hiển thị** tồn tại trong giao diện nếu component được render hay không.
- Kiểm thử xem component hiển thị data rỗng đó có hiển thị hình ảnh đi kèm mong muốn hay không.

```
import { render } from '@testing-library/react';
import React from 'react';
import { EmptyTableContent } from './EmptyTable';
import '@testing-library/jest-dom';

describe('React unit test', () => {
  it('Test display text of empty table', () => {
    const { queryByText } = render(<EmptyTableContent />);
    expect(queryByText('Không có dữ liệu hiển thị')).toBeInTheDocument();
  });

  it('SVG image to be in the empty table', () => {
    const { queryByTestId } = render(<EmptyTableContent />);
    expect(queryByTestId('empty-svg').innerHTML).toBe('empty_table.svg');
  });
});
```

Hình 2.14: Đoạn mã hiện thực

```
PASS | src/components/EmptyTable.test.tsx
React unit test
  ✓ Test display text of empty table (28 ms)
  ✓ SVG image to be in the empty table (6 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.801 s, estimated 1 s
Ran all test suites related to changed files.
```

Hình 2.15: Kết quả chạy 2 test case

2.1.8 Tổng kết

Trên đây là những công nghệ nhóm đã dùng để hiện thực tương đối hoàn chỉnh phần front-end của hệ thống. Một hệ thống Front-end hoàn chỉnh không những chỉ cần những thành phần để hệ thống có thể chạy được mà cần phải có hệ thống track lỗi, thống kê truy cập vào website và đảm bảo lỗi thông qua unit test.

2.2 Back-end

2.2.1 Giới thiệu kiến trúc Microservices

Trước khi Microservices [8] xuất hiện, các ứng dụng thường phát triển theo mô hình Monolithic architecture (Kiến trúc một khối). Có nghĩa là tất cả các module (view, business, database) đều được gộp trong một project, một ứng dụng được phát triển theo mô hình kiến trúc một khối thường được phân chia làm nhiều module. Nhưng khi được đóng gói và cài đặt sẽ thành một khối (monolithic). Lợi ích của mô hình kiến trúc một khối đó là dễ dàng phát triển và triển khai. Nhưng bên cạnh đó nó cũng có nhiều hạn chế ví dụ như khó khăn trong việc bảo trì, tính linh hoạt và khả năng mở rộng kém, đặc biệt với những ứng dụng doanh nghiệp có quy mô lớn. Đó chính là lí do ra đời của kiến trúc Microservices. Với lý do đó nhóm sẽ chọn phát triển hệ thống theo kiến trúc Microservices.

- Những đặc điểm của Microservices
 - **Decoupling** - Các service trong một hệ thống phần lớn được tách rời. Vì vậy, toàn bộ ứng dụng có thể dễ dàng được xây dựng, thay đổi và thu nhỏ.
 - **Componentization** - Microservices được coi là các thành phần độc lập có thể dễ dàng thay thế và nâng cấp.
 - **Business Capabilities** - Mỗi một thành phần trong kiến trúc microservice rất đơn giản và tập trung vào một nhiệm vụ duy nhất.
 - **Autonomy** - Các lập trình viên hay các nhóm có thể làm việc độc lập với nhau trong quá trình phát triển.
 - **Continous Delivery** - Cho phép phát hành phần mềm thường xuyên, liên tục.
 - **Decentralized Governance** - Không có mẫu chuẩn hóa hoặc bất kỳ mẫu công nghệ nào. Được tự do lựa chọn các công cụ hữu ích tốt nhất để có thể giải quyết vấn đề.
 - **Agility** - Microservice hỗ trợ phát triển theo mô hình Agile.
- **Ưu điểm**

Kiến trúc Microservices được sinh ra để khắc phục những hạn chế của kiến trúc một khối. Kiến trúc Microservices giúp đơn giản hóa hệ thống, chia nhỏ hệ thống

ra làm nhiều service nhỏ lẻ dễ dàng quản lý và triển khai từng phần so với kiến trúc nguyên khối. Phân tách rõ ràng giữa các service nhỏ. Cho phép việc mỗi service được phát triển độc lập. Cũng như cho phép lập trình viên có thể tự do chọn lựa công nghệ và ngôn ngữ cho mỗi service mình phát triển. Mỗi service có thể được triển khai một cách độc lập (VD: Mỗi service có thể được đóng gói vào một docker container độc lập, giúp giảm tối đa thời gian deploy). Nó cũng cho phép mỗi service có thể được mở rộng một cách độc lập với nhau. Việc mở rộng có thể được thực hiện dễ dàng bằng cách tăng số instance cho mỗi service rồi phân tải bằng load balancer.

- **Independent Development** - Tất cả các service có thể được phát triển dễ dàng dựa trên chức năng cá nhân của từng service. Có thể chia nhỏ để phát triển độc lập.
- **Independent Deployment** - Có thể được triển khai riêng lẻ trong bất kỳ ứng dụng nào.
- **Fault Isolation** - Khi một service của ứng dụng không hoạt động, hệ thống vẫn tiếp tục hoạt động.
- **Mixed Technology Stack** - Các ngôn ngữ và công nghệ khác nhau có thể được sử dụng để xây dựng các service khác nhau của cùng một ứng dụng.
- **Nhược điểm**
 - Kiến trúc Microservices đang là một xu hướng, nhưng nó cũng có nhược điểm của nó. Microservice khuyến khích làm nhỏ gọn các service, nhưng khi chia nhỏ sẽ dẫn đến những thứ vụn vặt, khó kiểm soát. Hơn nữa chính từ đặc tính phân tán khiến cho các lập trình viên phải lựa chọn cách thức giao tiếp phù hợp khi xử lý request giữa các service.
 - Hơn nữa việc quản lý nhiều database, và transaction giữa các service trong một hệ thống phân tán cũng là một khó khăn không nhỏ. Hay khi thực hiện test một service thì cũng cần test các service có liên quan.
 - Triển khai microservice cũng sẽ phức tạp hơn so với ứng dụng kiến trúc một khối, cần sự phối hợp giữa nhiều service, điều này không đơn giản như việc triển khai WAR trong một ứng dụng kiến trúc một khối.
- **Kết luận:** Với những ưu điểm và nhược điểm đã được nói ở trên nên nhóm đã quyết định sử dụng kiến trúc Microservices để xây dựng hệ thống của nhóm.

2.2.2 Công nghệ tổng quan

Công nghệ nền tảng dự kiến sử dụng của nhóm: SPRING BOOT, SPRING CLOUD VÀ SPRING SECURITY

- **Spring Boot** [9] là một dự án nổi bật trong hệ sinh thái Spring Framework. Nếu như trước đây, công đoạn khởi tạo một dự án Spring khá vất vả từ việc khai báo

các dependency trong file pom.xml cho đến cấu hình bằng XML hoặc annotation phức tạp, thì giờ đây với Spring Boot, chúng ta có thể tạo các ứng dụng Spring một cách nhanh chóng và cấu hình cũng đơn giản hơn.

- **Spring Cloud** [9] là nền tảng khá mới mẻ trong gia đình Spring.io dùng để xây dựng microservice một cách nhanh chóng. Spring Cloud cung cấp các công cụ cho các lập trình viên để nhanh chóng xây dựng một số common patterns trong các hệ thống phân tán (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Chúng sẽ hoạt động tốt trong bất kỳ môi trường phân tán nào, bao gồm máy tính xách tay của chính lập trình viên, các data center hoặc trên cloud.
- **Spring Security** là một trong những core feature quan trọng của Spring Framework, nó giúp chúng ta phân quyền và xác thực người dùng trước khi cho phép họ truy cập vào các tài nguyên của chúng ta.

Công nghệ sử dụng cho Secure API

- **Json Web Token(JWT)** [10]- Là 1 tiêu chuẩn mở (RFC 7519) định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng 1 đối tượng JSON. Thông tin này có thể được xác thực và đánh dấu tin cậy nhờ nó có chứa chữ ký số (digital signature). Phần chữ ký của JWT sẽ được mã hóa lại bằng HMAC hoặc RSA. Sử dụng JWT là cách tốt để áp dụng cơ chế bảo mật đối với các dịch vụ API RESTful mà có thể được sử dụng để truy cập vào cơ sở dữ liệu.

2.2.3 Kiến trúc RESTful API

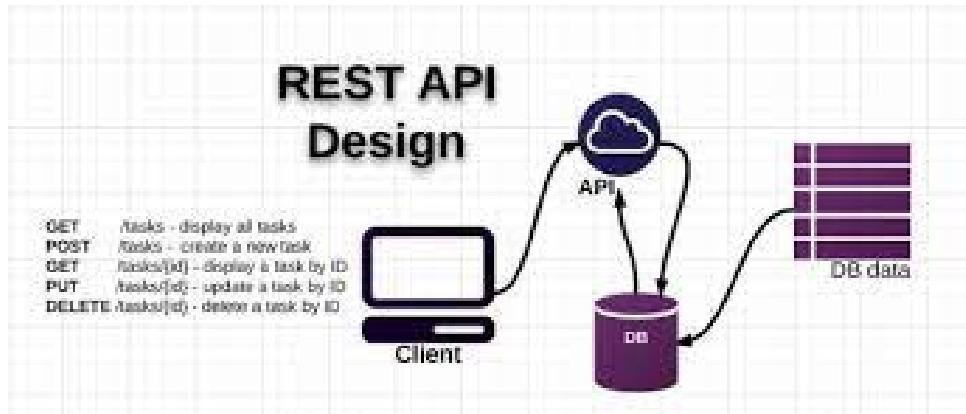
RESTful API là một tiêu chuẩn dùng trong việc thiết kế các API cho ứng dụng web để quản lý tài nguyên. RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile, web service...) khác nhau có thể giao tiếp với nhau.

Khi làm việc với server sẽ gồm 4 hoạt động thiết yếu đó là: lấy dữ liệu, tạo mới, cập nhật, xóa dữ liệu.

REST hoạt động chủ yếu dựa vào giao thức HTTP. Mỗi trong 4 hoạt động cơ bản trên sẽ sử dụng những phương thức HTTP riêng (HTTP method):

- POST (CREATE) : Tạo mới một tài nguyên.
- GET (READ) : Trả về một tài nguyên hoặc một danh sách tài nguyên.
- PUT (UPDATE) : Cập nhật, thay thế thông tin cho tài nguyên.
- DELETE (DELETE) : Xóa một tài nguyên.

REST là một kiến trúc thống nhất giúp thiết kế các website để có thể dễ dàng quản lý các tài nguyên. Nó không phải là một quy luật buộc phải tuân theo mà đơn giản là một kiến trúc được đề xuất ra và kiến trúc này hiện đang được sử dụng rất phổ biến vì tính đơn giản, dễ hiểu và rất ưu việt của nó. Với các ứng dụng web được thiết kế sử dụng RESTful, có thể dễ dàng biết được URL và HTTP method để quản lý một tài nguyên.



Hình 2.16: RESTful API [7]

Ưu điểm

- REST cũng có ưu điểm khi sử dụng giao thức stateless (không trạng thái). Hệ thống này không sử dụng session, cookie, không cần biết những thông tin đó trong mỗi lần request đến máy chủ ngoài. Điều này giúp REST giảm tải cho máy chủ ngoài, nâng cao hiệu suất làm việc.
- Tính khả biến: với các hệ thống cần thay đổi các tài nguyên liên tục, sử dụng REST với việc tạo request đơn giản sẽ giúp mọi chuyện trở nên đơn giản hơn.
- Tính mở rộng: các hệ thống REST có khả năng mở rộng rất cao nhờ sự tách biệt giữa các thành phần và các quy ước giao tiếp được quy định sẵn.
- Tính linh hoạt: việc chuẩn hóa interface giúp hệ thống trở nên linh hoạt hơn, có thể sử dụng cho cho nhiều nền tảng khác nhau, mobile, web,...
- Trong sáng: trong giao tiếp giữa các thành phần, các request trở nên rất rõ ràng, dễ hiểu.

Nhược điểm

- REST chỉ hoạt động trên các giao thức HTTP.

2.2.4 MySQL

MySQL[11] là hệ quản trị cơ sở dữ liệu tự do nguồn mở phổ biến nhất thế giới và được các nhà phát triển rất ưa chuộng trong quá trình phát triển ứng dụng. Vì MySQL là cơ sở dữ liệu tốc độ cao, ổn định và dễ sử dụng, có tính khả chuyển, hoạt động trên nhiều hệ điều hành cung cấp một hệ thống lớn các hàm tiện ích rất mạnh. Với tốc độ và tính bảo mật cao, MySQL rất thích hợp cho các ứng dụng có truy cập cơ sở dữ liệu trên internet.



Hình 2.17: Mysql

Ưu điểm

- **Dễ sử dụng:** MySQL là cơ sở dữ liệu tốc độ cao, ổn định, dễ sử dụng và hoạt động trên nhiều hệ điều hành.
- **Độ bảo mật cao:** MySQL rất thích hợp cho các ứng dụng có truy cập cơ sở dữ liệu trên Internet khi sở hữu nhiều tính năng bảo mật thậm chí là ở cấp cao.
- **Đa tính năng:** MySQL hỗ trợ rất nhiều chức năng SQL được mong chờ từ một hệ quản trị cơ sở dữ liệu quan hệ cả trực tiếp lẫn gián tiếp.
- **Khả năng mở rộng và mạnh mẽ:** MySQL có thể xử lý rất nhiều dữ liệu và hơn thế nữa nó có thể được mở rộng nếu cần thiết.

Nhược điểm

- **Khả năng scale rất khó và tốn chi phí:** Nếu bản ghi lớn dần lên thì việc truy xuất dữ liệu là khá khó khăn, khi đó chúng ta sẽ phải áp dụng nhiều biện pháp để tăng tốc độ truy xuất dữ liệu như là chia tải database này ra nhiều server.

2.2.5 Cassandra

Ngày nay, các dịch vụ trên Internet phải xử lý khối lượng dữ liệu rất lớn. Hầu hết dữ liệu sẽ được lưu trữ phân tán trên nhiều máy chủ khác nhau. Vì vậy, các hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) tỏ ra không còn phù hợp với các dịch vụ như thế này nữa. Người ta bắt đầu nghĩ tới việc phát triển các DBMS mới phù hợp để quản lý các khối lượng dữ liệu phân tán này. Các DBMS này thường được gọi là NoSQL. Một đại diện nổi bật của các NoSQL là Cassandra[12].

Cassandra ban đầu được tạo ra bởi Facebook. Sau đó nó đã được tặng cho Quỹ Apache và tháng 2 năm 2010 và được nâng cấp lên thành dự án hàng đầu của Apache. Cassandra là một cơ sở dữ liệu phân tán kết hợp mô hình dữ liệu của Google Bigtable với thiết kế hệ thống phân tán như bản sao của Amazon Dynamo.



Hình 2.18: Cassandra

Ưu điểm

- Khả năng chịu lỗi cao: Do dữ liệu khi lưu vào cassandra sẽ được nhân bản và lưu trữ trên các node khác nhau. Vì thế nếu có 1 node nào đó chứa dữ liệu cần đọc thì hệ thống có thể điều phối để đọc dữ liệu đó ở node khác.
- Kiến trúc không có SPOF (Single Point Of Failure): Bởi vì trong Cassandra không có node chính. Các node kết hợp với nhau tạo thành 1 Ring. Và các node đều có vai trò như nhau. Cho nên hệ thống sẽ không dừng lại khi một hoặc một số node cho phép trong mạng có vấn đề.

- Hỗ trợ nhiều ngôn ngữ khác nhau: Do cassandra thu thập dữ liệu bằng một framework có tên là Thrift và Thrift có một cơ chế để giao tiếp với các ngôn ngữ khác nhau.
- Dễ dàng scale và mở rộng: Cassandra có hệ thống cluster bao gồm nhiều node được kết nối với nhau tạo thành Ring. Vì thế để mở rộng khả năng lưu trữ và xử lý thì ta chỉ việc thêm node vào mà không ảnh hưởng đến các node khác trong mạng do Cassandra sử dụng một thuật toán có tên là Consistency Hashing để phân chia dữ liệu đến các node trong Ring.
- Tốc độ đọc/ghi nhanh: Do Cassandra không có ràng buộc giữa các table như RDBMS nên việc truy xuất cực kì nhanh.

Nhược điểm

- Cassandra không hỗ trợ cho việc tính toán trên storage. Ở đây ta có thể tính toán ở application trước khi lưu vào Database.
- Do dữ liệu được phân tán trên các node nên dữ liệu giữa các có thể không giống nhau (chỉ đảm bảo Eventually Consistency). Ở đây ta có thể set Consistency Level để quản lý việc đọc/ghi dữ liệu. Nếu ta ưu tiên về độ chính xác thì có thể set là QUORUM hay ALL nhưng bù lại sẽ tăng thời gian phản hồi. Ngược lại, nếu ưu tiên về tốc độ thì ta có thể set là ONE, TWO, ... nhưng ở đây ta phải đánh đổi về độ chính xác của dữ liệu.

Kết luận

- Sau quá trình tìm hiểu và thử nghiệm thì nhóm nhận thấy một số điều. Đối với MySQL thì có một số ưu điểm như: dễ sử dụng, hỗ trợ transaction, hỗ trợ ràng buộc nghiêm ngặt, ... nhưng nhóm cũng nhận thấy một nhược điểm rất lớn đó là khó mở rộng. Đối với lượng dữ liệu còn ít thì việc thực thi các câu truy vấn trong MySQL diễn ra rất trơn tru nhưng khi dữ liệu càng phình to ra thì việc truy vấn trở nên nặng nề và rất chậm. Vì thế nhóm sẽ sử dụng MySQL để lưu các thiết lập ít thay đổi trong hệ thống. Ví dụ như: path, port của service khác, mã lỗi, ...
- Ngoài ra với những ưu điểm của Cassandra thì nhóm sẽ sử dụng để lưu trữ các dữ liệu về business, các dữ liệu về log, dữ liệu về người dùng, ... để có thể đảm bảo tính chịu lỗi và sẵn sàng.

2.2.6 Redis

Ngày nay việc tăng tốc truy vấn để đáp ứng request một cách nhanh chóng đang dần trở nên quan trọng. Mỗi khi muốn truy xuất lấy dữ liệu từ Database thì ta phải xuông ổ cứng, mà tốc độ đọc/ghi dữ liệu từ ổ cứng chậm hơn rất nhiều so với RAM (khoảng 200 lần). Vì thế người ta nghĩ ra cách lưu dữ liệu thường xuyên truy xuất lên RAM để

có thể phản hồi yêu cầu của người dùng với tốc độ nhanh nhất. Từ đó, Redis ra đời và được biết đến như là 1 cache database rất phổ biến trên thế giới.



Hình 2.19: Redis[13]

Đặc điểm

- Là cơ sở dữ liệu NoSQL, lưu trữ dữ liệu dưới dạng KEY-VALUE.
- Lưu trữ dữ liệu trên RAM, giúp việc truy xuất dữ liệu cực kì nhanh chóng.
- Hỗ trợ nhiều cấu trúc dữ liệu cơ bản như Hash, List, Set, Sorted Set, String,....
- Hỗ trợ cơ chế Pub/Sub messaging.
- Hỗ trợ cơ chế Persistence giúp bảo vệ khỏi mất mát dữ liệu khi có sự cố xảy ra.

Nhờ đặc điểm giúp giảm thời gian truy vấn, nên Redis có tác dụng rất mạnh mẽ trong việc sử dụng để cache dữ liệu thường xuyên được truy xuất của các ứng dụng web.

Ưu điểm

- Dữ liệu lưu trữ trong bộ nhớ
- Hỗ trợ nhiều cấu trúc dữ liệu linh hoạt, phù hợp với nhiều ngôn ngữ lập trình như: String, List, Hash, Set, ZSet, ...
- Đơn giản và dễ sử dụng
- Sao chép và độ bền: Redis hỗ trợ xây dựng theo nhiều kiến trúc khác nhau tùy vào nhu cầu và mục đích sử dụng. Ví dụ như: master-slave thì slave sẽ sao lưu dữ liệu từ master. Nếu master bị down thì slave sẽ được đưa lên thay thế master nhằm tránh việc bị mất mát dữ liệu, ngoài ra còn có thể phân chia việc đọc/ghi dữ liệu. Slave sẽ xử lý các request đọc dữ liệu, còn master sẽ xử lý các request ghi dữ liệu.

- Khả năng mở rộng: Redis là dự án open source được cộng đồng đóng góp ủng hộ. Không có giới hạn về nhà cung cấp hoặc công nghệ vì thế Redis còn có thể mở rộng và cải tiến thêm nhiều tính năng hơn nữa.

Nhược điểm

- Do Redis lưu trữ dữ liệu trong RAM nên phần nào vẫn có nguy cơ mất mát dữ liệu.
- Mỗi lần chỉnh sửa hay thêm dữ liệu thì ta đồng thời phải thêm/sửa vào cả database và Redis nên có thể mang đến một số phức tạp.
- Key trong Redis có giới hạn: Vì thế trong quá trình lưu trữ nếu không cẩn thận có thể gây chết server Redis.
- Cache Invalidations: Kho khăn khi loại bỏ đi dữ liệu không còn được sử dụng trong Redis. Thông thường người ta thường đặt Time to live cho mỗi dữ liệu cache tùy theo tần số thay đổi, tần suất truy vấn và mức độ quan trọng của dữ liệu.

Kết luận

Ở đây nhóm sẽ sử dụng Redis cho mục đích cache là chủ yếu. Bởi vì có một số thông tin thường xuyên được truy xuất nhiều lần. Nếu để trong database được lưu ở ổ cứng thì dẫn tới việc làm tăng thời gian truy xuất, có thể dẫn tới chết database. Ngoài ra do Redis hỗ trợ thêm cơ chế **Persistence** để định kỳ sao lưu dữ liệu lên ổ cứng nên có thể phần nào đảm bảo dữ liệu không bị mất mát khi có sự cố xảy ra.

2.2.7 ActiveMQ

Message queue[16] là một kiến trúc cung cấp cơ chế giao tiếp không đồng bộ. Ý nghĩa của queue ở đây chính là 1 hàng đợi chứa message chờ để được xử lý tuân tự theo cơ chế vào trước thì ra trước (FIFO - First In First Out). Một message là các dữ liệu cần vận chuyển giữa người gửi và người nhận. Ta có thể hiểu đơn giản việc sử dụng message queue cũng giống như việc ta đi gửi thư vậy. Lúc đến bưu điện ta chỉ việc bỏ thư vào hòm mà không cần quan tâm lúc nào thư sẽ được gửi đi mà chỉ biết là nó chắc chắn sẽ được xử lý. Đây là một ví dụ về việc giao tiếp bất đồng bộ.



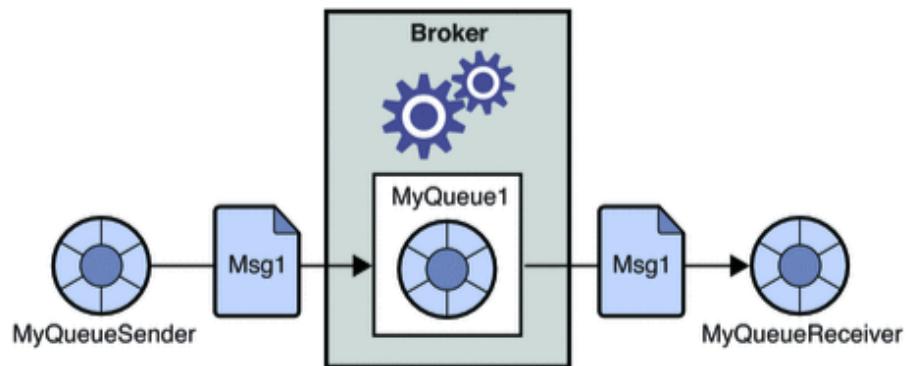
Hình 2.20: Active MQ[15]

Kiến trúc cơ bản

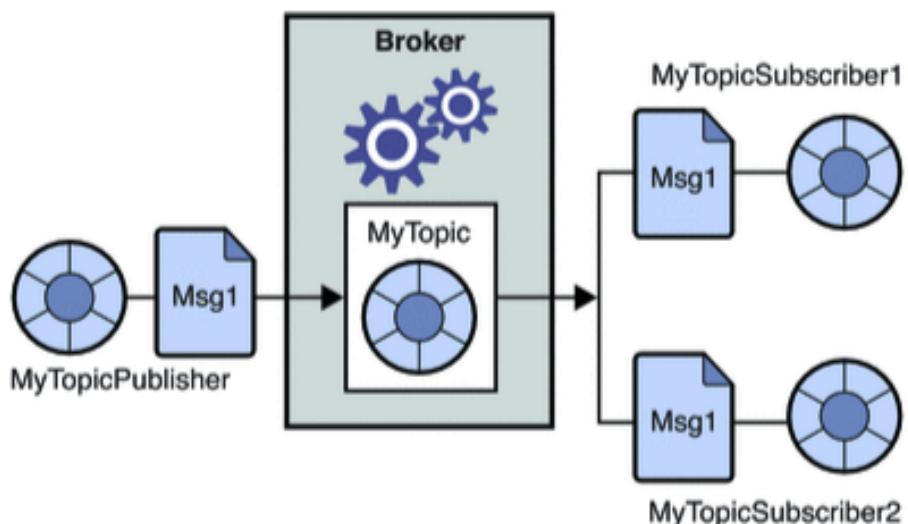
- Message: Là thông tin cần gửi đi
- Producer: Là thành phần sẽ gửi message
- Consumer: Là thành phần nhận message
- Message Queue: Nơi chứa message, cho phép producer và consumer có thể trao đổi với nhau.

Phân loại

- Point-to-point: Tức là message chỉ được consume bởi 1 thành phần duy nhất

**Hình 2.21:** Point-to-point

- Publisher-Subscriber: Ở đây ta có thể có nhiều consumer cùng consume 1 message giống nhau khi chúng cùng subscriber vào 1 topic.

**Hình 2.22:** Publisher-Subscriber

Ưu điểm

- Đảm bảo 1 message chỉ được xử lý đúng 1 lần duy nhất: Mỗi khi consumer lấy được message thì nó sẽ gửi 1 ACK về cho Broker, sau khi nhận được tín hiệu ACK thì Broker sẽ tiến hành xóa message đó đi để tránh việc 1 message được xử lý 2 lần.

- Nhắn tin không đồng bộ: Phù hợp với những việc không cần phải xử lý ngay lập tức. Lúc đó ta cứ đưa message vào queue rồi đi xử lý các công việc tiếp theo. Nhằm giảm thời gian phản hồi đối với người dùng.
- Tăng khả năng sẵn sàng: Giả sử ta có 2 service và lúc đó 1 trong 2 service đang trong trạng thái không hoạt động. Thì lúc đó hệ thống vẫn có thể hoạt động bình thường. Ta giữ các message trong queue đến khi service hoạt động lại thì nó consume message và tiếp tục công việc xử lý của nó
- Dễ dàng mở rộng: Mỗi khi lượng request tăng cao. Producer bắn message nhanh hơn tốc độ xử lý cả consumer thì lúc này ta chỉ đơn giản là thêm consumer vào để tăng tốc độ xử lý. Chú ý: Lúc này ta cần set tất cả các consumer chung 1 group-id để tránh việc 1 message có thể được xử lý nhiều lần.

Nhược điểm

- Làm phức tạp hệ thống: Khi thêm 1 thành phần thì ta phải chấp nhận việc xử lý phức tạp và có thể xảy ra sai sót trong quá trình xử lý.
- Producer và Consumer phải thống nhất về format của message: Mặc dù cả producer và consumer không quan tâm đến nhau (chỉ cần gửi message vào queue rồi thẳng kia tới lấy) nhưng cả 2 cần có chung 1 format message để có thể dễ dàng xử lý sau khi consume.
- Monitor Queue: Ta cần phải theo dõi queue để biết được tình trạng hiện tại, queue có đầy hay không, có bị crash không để đưa ra phương án xử lý.

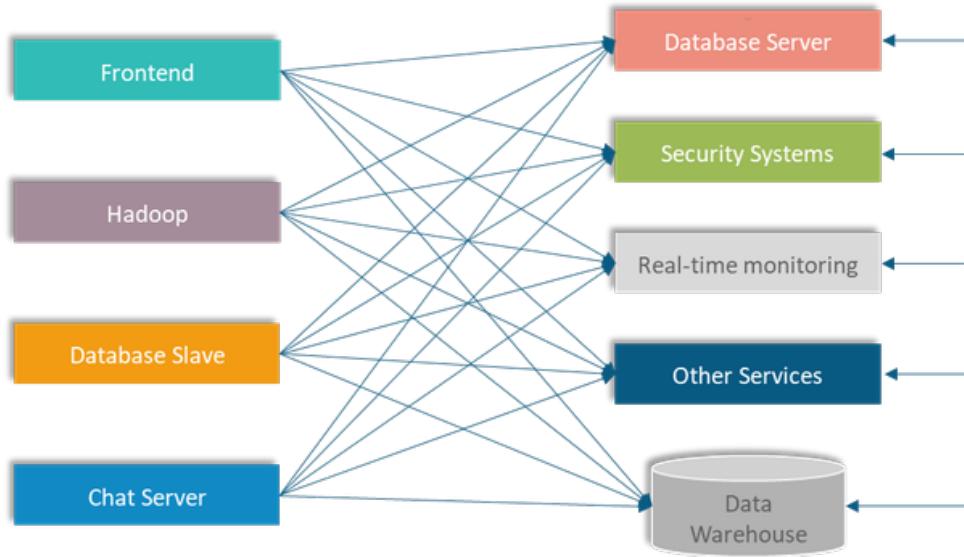
Kết luận

Mặc dù message queue rất phù hợp để giao tiếp giữa các service trong hệ thống microservice. Nhưng ở đây nhóm chỉ sử dụng message để hiện thực cơ chế bất đồng bộ khi xử lý request tại một service cụ thể. Ví dụ như: Sau khi xử lý request của người dùng ta cần vào Database để cập nhật lại status của đơn hàng nhưng việc truy xuất vào Database ngay thời điểm đó là không cần thiết. Ta có thể đưa message vào queue và sau đó trả kết quả về cho client ngay lập tức. Điều đó giúp ta tránh được việc truy xuất Database quá lâu sẽ ảnh hưởng đến trải nghiệm của người dùng.

2.2.8 Kafka

Kafka[17] là dự án mã nguồn mở, đã được đóng gói hoàn chỉnh, khả năng chịu lỗi cao và là hệ thống nhắn tin nhanh. Vì tính đáng tin cậy của nó, kafka đang dần được thay thế cho hệ thống nhắn tin truyền thống. Nó được sử dụng cho các hệ thống nhắn tin thông thường trong các ngữ cảnh khác nhau. Đây là hệ quả khi khả năng mở rộng ngang và chuyển giao dữ liệu đáng tin cậy là những yêu cầu quan trọng nhất.

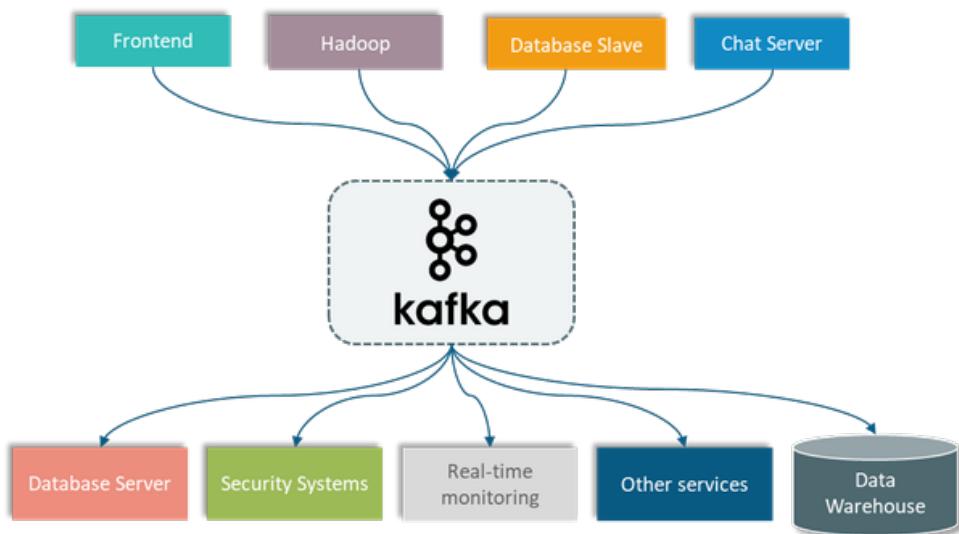
Để có thể hiểu rõ vai trò của Kafka ta có thể xem ảnh sau



Hình 2.23: Vai trò Kafka

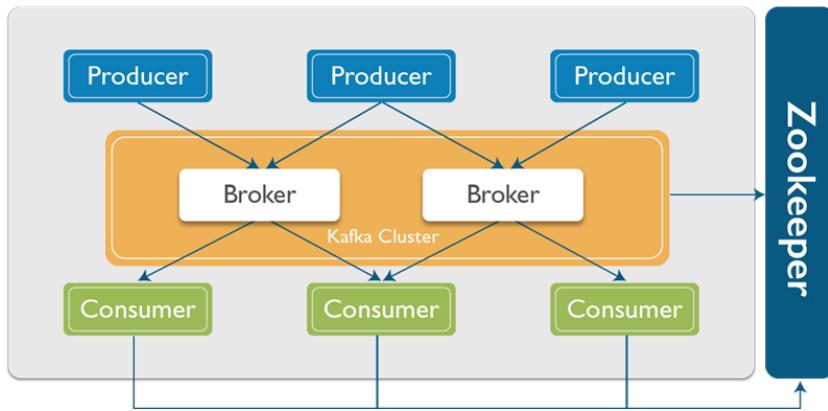
Thông thường việc giao tiếp giữa các thành phần trong hệ thống diễn ra hết sức phức tạp. Điều này làm tăng chi phí giám sát và làm giảm hiệu suất của hệ thống.

Và từ đó Kafka ra đời giúp chúng ta đơn giản đi việc giao tiếp giữa các thành phần trong hệ thống với nhau.



Hình 2.24: Giao tiếp giữa các hệ thống

Kiến trúc



Hình 2.25: Kiến trúc của Kafka

- Producer: Thành phần gửi message
- Message: Là message cần trao đổi đến các thành phần khác nhau trong hệ thống.
- Broker: Là tập hợp các server Kafka để lưu trữ các message
- Zookeeper: Được dùng để quản lý và bố trí các Broker

Kết luận: Tại sao lại sử dụng Kafka và không sử dụng Message Queue truyền thống?

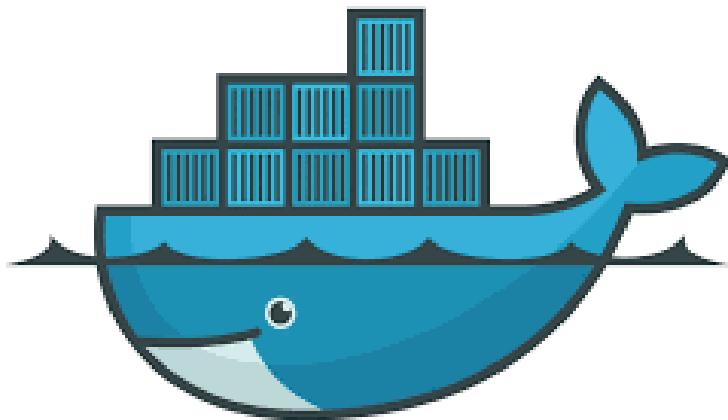
Như nhóm đã trình bày ở phần trước thì Message Queue truyền thống sẽ đảm nhận nhiệm vụ xử lý bất đồng bộ trong từng service. Còn kafka sẽ là thành phần trung gian để gửi message từ service này đến service khác. Bởi vì kafka hoạt động rất nhanh do nó sử dụng cơ chế ZeroCopy, dữ liệu được sắp xếp theo khối, nén dữ liệu và giảm độ trễ I/O. Ngoài ra kafka rất dễ mở rộng khi lượng message cần trao đổi tăng lên và có cơ chế sao chép message ra nhiều partition để làm giảm rủi ro mất mát message trong quá trình xử lý.

2.2.9 Docker

Docker[18] là nền tảng cung cấp cho các công cụ, service để các lập trình viên, quản lý hệ thống có thể phát triển, thực thi, chạy các ứng dụng với containers. Hay nói một cách khác nó là một nền tảng để cung cấp cách để xây dựng, triển khai và chạy các ứng dụng một cách dễ dàng trên nền tảng ảo hóa - "**Build once, run anywhere**". Hay nói một cách dễ hiểu như sau: Khi chúng ta muốn chạy ứng dụng thì chúng ta phải thiết lập môi trường chạy cho nó. Thay vì chúng ta sẽ đi cài môi trường chạy cho nó thì chúng ta sẽ chạy docker.

Ứng dụng Docker chạy trong vùng chứa (container) có thể được sử dụng trên bất kỳ hệ thống nào: máy tính xách tay của nhà phát triển, hệ thống trên cơ sở hoặc trong hệ thống đám mây. Và là một công cụ tạo môi trường được "đóng gói" (còn gọi là Container) trên máy tính mà không làm tác động tới môi trường hiện tại của máy, môi trường trong Docker sẽ chạy độc lập.

Kết luận: Do nhóm viết ứng dụng theo kiến trúc Microservice vì thế có rất nhiều cơ sở hạ tầng phải deploy như: Cassandra, Kafka, Redis, Mysql, ActiveMQ, ... Ngoài ra còn rất nhiều service mà nhóm đã viết. Cho nên việc triển khai và quản lý trên server vật lý hay virtual machine đều gặp rất nhiều khó khăn. Vì thế nhóm đã tìm hiểu và sử dụng Docker để đơn giản hóa quá trình triển khai cơ sở hạ tầng và quản lý tài nguyên của mỗi service sao cho hiệu quả và tiết kiệm nhất. Đặc biệt việc sử dụng **Docker Compose** giúp chúng ta khởi tạo và quản lý các containers rất dễ dàng chỉ thông qua 1 dòng lệnh.



Hình 2.26: Giới thiệu về Docker

2.2.10 NGINX

NGINX[19] là một web server mạnh mẽ và sử dụng kiến trúc đơn luồng, hướng sự kiện vì thế nó hiệu quả hơn Apache server nếu được cấu hình chính xác. Nó cũng có thể làm những thứ quan trọng khác, chẳng hạn như load balancing, HTTP caching, hay sử dụng như một reverse proxy.

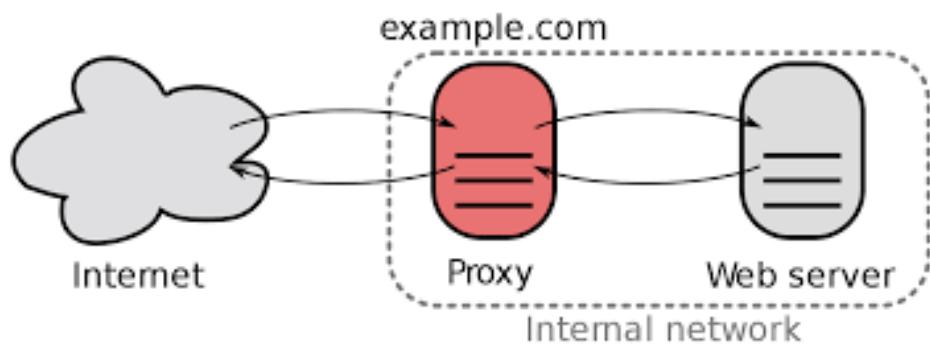
NGINX được xây dựng để cung cấp việc sử dụng bộ nhớ thấp và đồng thời cao. Thay vì tạo các quy trình mới cho mỗi yêu cầu web, NGINX sử dụng cách tiếp cận theo hướng sự kiện, không đồng bộ trong đó các yêu cầu được xử lý trong một luồng.

Reverse proxy là một proxy server mà khi đúng trước client, chúng hoạt động giống như những server bình thường. Reverse proxy chuyển tiếp yêu cầu đến một hoặc nhiều

server thật, kết quả sau đó trả về cho client như thể là chúng được trả về từ reverse proxy, khiến cho client không biết về những server thật nói trên. Reverse proxy được cài đặt trong một private network của một hoặc nhiều server, và tất cả lưu lượng truy cập đều phải đi qua proxy này.

Nhờ vào reverse proxy nhóm có thể:

- Che giấu được thông tin các service được triển khai bên trong private network.
- Tăng khả năng xử lý nhiều request đồng thời.
- Phân chia tải cho các service bên trong.
- Thiết lập cấu hình HTTPS.



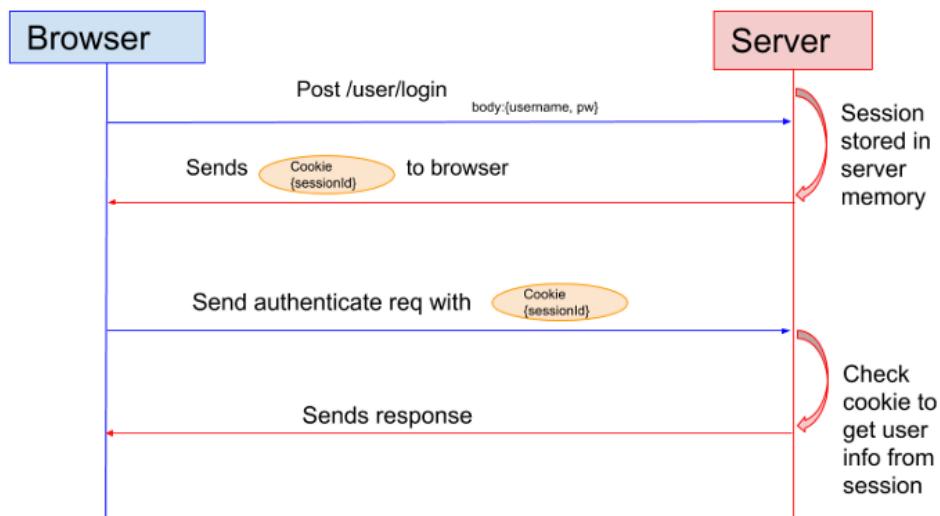
Hình 2.27: Reverse Proxy[20]

Bài toán và giải pháp

Sau khi phân tích, nghiên cứu các hệ thống và sản phẩm khác thì nhóm đã liệt kê ra một số bài toán cần phải giải quyết. Sau đây là phần trình bày của nhóm.

3.1 Bài toán xác thực người dùng

Như chúng ta đã biết thì HTTP là 1 giao thức Stateless, nghĩa là khi Client gửi dữ liệu lên Server, Server thực thi xong và trả kết quả về cho Client thì quan hệ giữa Client và Server sẽ bị cắt đứt, Server sẽ không lưu lại bất cứ dữ liệu trạng thái gì của Client. Nói một cách dễ hiểu hơn là khi ta đăng nhập vào MyBK và sau đó di chuyển sang trang Thông tin Sinh viên thì chúng ta phải đăng nhập lại vì Server không lưu giữ bất kỳ thông tin gì của chúng ta. Để giải quyết vấn đề này thì Session/Cookies ra đời.

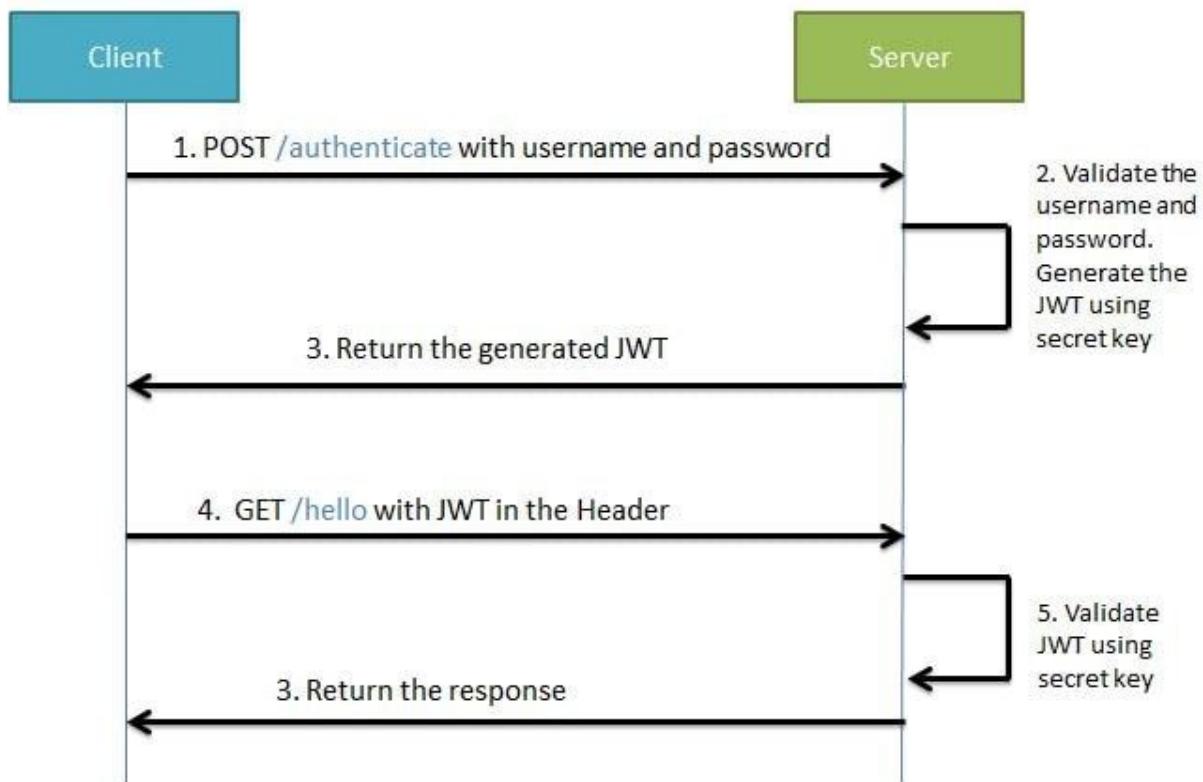


Hình 3.1: Xác thực dựa trên Session và Cookies [21]

Đầu tiên người dùng nhập thông tin tài khoản của mình. Sau khi kiểm tra đúng thông tin đăng nhập thì Server sẽ tạo ra 1 sessionID, lưu lại vào Database và trả về cho Client. Sau đó mỗi request gửi lên thì Client sẽ chèn sessionId vào Cookies và gửi lên cho Server từ đó Server kiểm tra và biết được chính xác Client đó là ai.

Đối với hệ thống Microservices thì việc xác thực theo cách này có 1 vấn đề đó là cần phải có 1 nơi để lưu trữ thông tin về session, ngoài ra cần phải xử lý các session đã hết hạn. Việc này có thể là giảm hiệu năng của 1 hệ thống Microservices.

Sau khi tìm hiểu thì nhóm sử dụng cơ chế xác thực dựa trên token. Với cơ chế như sau:



Hình 3.2: Xác thực dựa trên token[22]

- Client gửi thông tin đăng nhập lên Server
- Server xác thực thông tin đăng nhập. Nếu đúng sẽ tạo ra 1 token và gửi về cho Client
- Client dính kèm token mỗi khi request lên Server

3.2 Bài toán xác thực giữa các services

Do các services được triển khai trên môi trường mạng vì thế có thể tiềm ẩn nhiều rủi ro bảo mật khi có sự giao tiếp giữa các services với nhau. Một kẻ tấn công nào đó có thể giả danh và gửi các request lên cho service của hệ thống xử lý. Do đó kẻ tấn công có thể lấy được các thông tin nhạy cảm của người dùng (như số điện thoại, tên, địa chỉ, ngày tháng năm sinh, ...). Đây là một điều rất cấm kỵ khi chúng ta phải có nghĩa vụ bảo mật các thông tin nhạy cảm cho người dùng.

Sau khi tìm hiểu và so sánh các phương pháp thì nhóm sử dụng phương pháp tạo chữ ký số để xác thực request giữa các service với nhau. Khi service A muốn gọi service B thì service A phải có key của service B, Service A sẽ gửi thêm 1 trường thông tin là **sig = SHA256(data + key)**, service B khi nhận request thì sẽ tạo sig dựa vào key của mình và so sánh 2 sig đó có giống nhau không. Nếu giống thì mới tiếp tục thực hiện request của Service A, còn nếu không thì sẽ trả về lỗi cho service A.

3.3 Bài toán nhận nhiều Request của người dùng

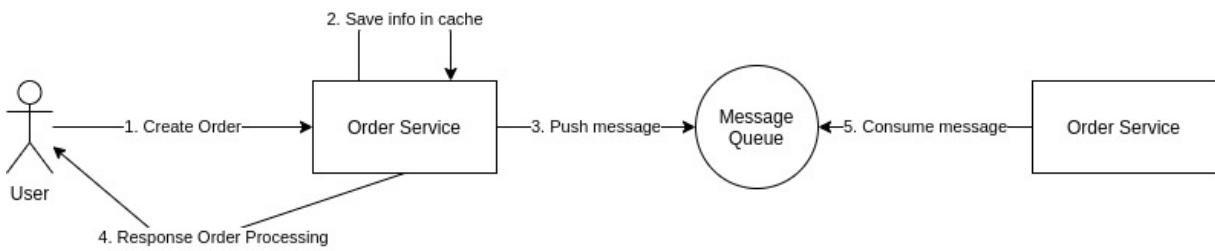
Thông thường 1 yêu cầu của người dùng sẽ được xử lý như sau:

- Người dùng gửi yêu cầu lên Server
- Server xử lý
- Người dùng nhận phản hồi trả về từ server

Trong quá trình chờ Server xử lý thì người dùng sẽ phải chờ đợi và nếu quá trình xử lý diễn ra lâu (bởi vì Server phải lưu thông tin vào database hay phải gọi Server khác,...) chính điều này sẽ ảnh hưởng đến trải nghiệm của người dùng.

Ngoài ra, trong những khoảng thời gian diễn ra các chương trình khuyến mãi giảm chi phí vận chuyển hoặc trong thời gian dịch bệnh thì nhu cầu vận chuyển hàng hóa tăng cao để đáp ứng nhu cầu tiêu thụ của người dân. Do đó có thể trong 1 khoảng thời gian ngắn lượng yêu cầu đến hệ thống tăng cao. Nếu vẫn xử lý với cơ chế như bình thường thì hệ thống sẽ chết hoặc có thể không nhận được yêu cầu của người dùng nữa. Điều này cũng ảnh hưởng nghiêm trọng đến trải nghiệm của người dùng.

Sau khi tìm hiểu và phân tích thì nhóm hiện thực chức năng tạo đơn hàng bằng cơ chế bất đồng bộ như sau:

**Hình 3.3:** Quá trình tạo đơn hàng

Ở đây nhóm sử dụng MessageQueue(ActiveMQ) để hiện thực cơ chế bất đồng bộ. Sau khi người dùng thực hiện tạo đơn hàng thì hệ thống sẽ lưu thông tin đơn hàng đó vào Cache, sau khi đã lưu thông tin đơn hàng vào Cache thì ta ngay lập tức phản hồi về cho người dùng là đơn hàng đang được xử lý. Tiếp theo ta push message tạo đơn hàng đó vào MessageQueue, và sau đó đơn hàng sẽ được consume và tiếp tục xử lý.

Việc sử dụng MessageQueue làm tách rời quá trình xử lý, Producer (service push message) chỉ cần đẩy message lên mà không cần quan tâm là message đó đã được xử lý hay không và Consumer (service consume message) chỉ cần lấy message và xử lý. Các mesage sẽ được lưu trữ trong Queue và chỉ mất đi khi được consume, điều này đảm bảo tất cả các message đều được xử lý duy nhất 1 lần và tất cả message sẽ không mất đi ngày cả khi OrderService bị chết do sự cố nào đó. Và đặc biệt lợi ích cuối cùng đó là trải nghiệm của người dùng, người dùng sẽ nhận phản hồi gần như lập tức sau khi tạo đơn hàng (mặc dù đơn hàng vẫn đang tiếp tục được xử lý) mà không cần phải chờ đợi các bước xử lý mất thời gian ở phía sau.

3.4 Bài toán giảm thời gian xử lý khi người dùng truy xuất thông tin

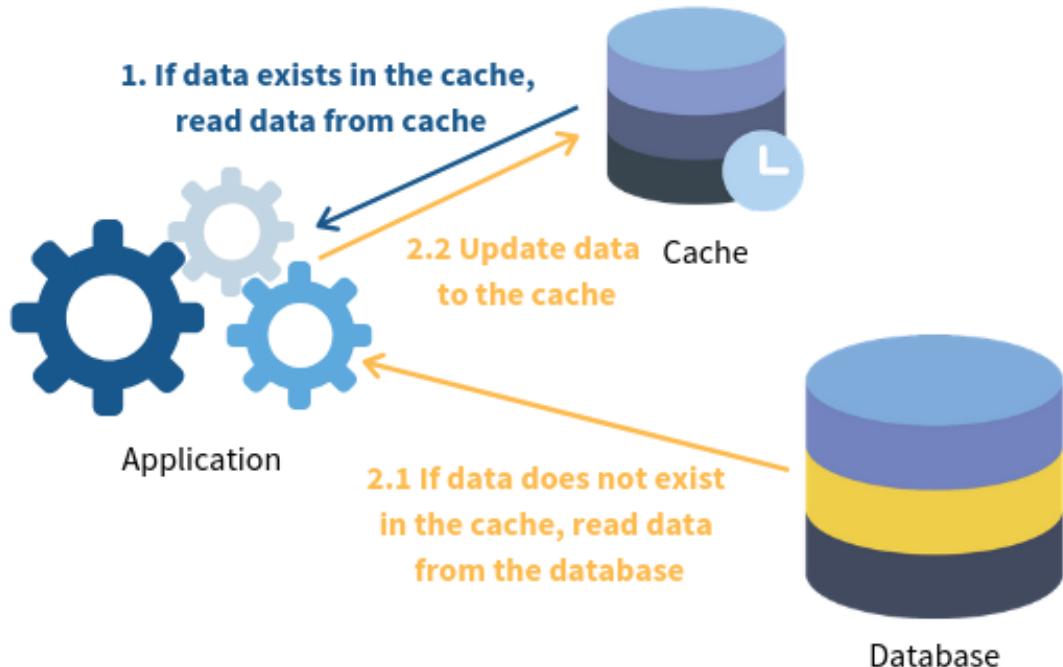
Trong quá trình sử dụng ứng dụng thì việc lấy và hiển thị thông tin ra cho người dùng diễn ra rất thường xuyên. Thông thường quá trình đó diễn ra như sau:

- Người dùng gửi yêu cầu truy xuất thông tin lên Server
- Server truy xuất vào Database
- Người dùng nhận phản hồi về từ Server

Ở đây việc truy xuất vào Database thường xuyên có thể làm giảm thời gian phản hồi do phải thực thi nhiều I/O operation.

Vì thế, sau khi tìm hiểu và nghiên cứu thì nhóm sẽ sử dụng Cache để lưu lại các thông tin mà người dùng thường xuyên truy xuất trong memory. Do thông tin được lưu trữ

trong memory nên việc truy xuất và lấy các thông tin đó được thực hiện rất nhanh (gấp 500 lần so với thông tin lấy từ Database). Có rất nhiều chiến lược Cache có thể sử dụng nhưng nhóm sử dụng chiến lược Cache Aside để phù hợp với nhu cầu của ứng dụng:



Hình 3.4: Chiến lược Cache Aside[14]

Đầu tiên khi muốn lấy thông tin nào đó thì ta tiến hành tìm trong Cache. Nếu trong Cache có thông tin cần tìm kiếm thì lập tức trả thông tin đó về cho người dùng, nếu không tìm thấy trong Cache thì ta tiến hành tìm kiếm trong Database, sau đó lưu thông tin đó vào Cache rồi mới trả về cho người dùng.

Việc sử dụng Cache đem lại hiệu quả nhưng cũng có 1 số điểm cần lưu ý như:

- Thứ nhất là **Stale Cache**: Có nghĩa là dữ liệu trong Database và trong Cache không nhất quán với nhau dẫn đến trả về các thông tin cũ cho người dùng. Nhóm giải quyết vấn đề này bằng cách mỗi khi cập nhật lại thông tin trong Database thì đều cập nhật luôn thông tin trong Cache. Việc này có thể làm tốn thêm chi phí nhưng nhóm chỉ lưu các thông tin mà người dùng thường xuyên truy xuất nên chi phí sẽ giảm bớt và rất xứng đáng so với hiệu quả to lớn mà Cache mang lại.
- Thứ hai là **Invalidate Cache**: Có nghĩa là nếu để Cache quá lâu trong memory thì sẽ gây tiêu tốn dung lượng, còn nếu để Cache quá nhanh thì sẽ không có hiệu quả do cứ phải truy xuất Database nhiều. Trong quá trình tìm hiểu và thử nghiệm thì nhóm đã thiết lập thời gian tồn tại trong memory là 60 ngày. Là con số có thể chấp

nhận được đối với mỗi đơn hàng mà người dùng tạo ra. Còn đối với thông tin về mã OTP thì chỉ được lưu trữ trong 60 giây.

3.5 Bài toán xác định kho nguồn - kho đích ngắn nhất và tính chi phí vận chuyển

Để doanh nghiệp có thể tiết kiệm chi phí cũng như giảm được chi phí vận chuyển cho khách hàng thì bài toán xác định kho nguồn - kho đích so với địa chỉ của người gửi - người nhận ngắn nhất là một bài toán cần phải giải quyết sao cho tối ưu.

Để xác định kho nguồn - kho đích ngắn nhất nhóm đã tìm quãng đường vận chuyển ngắn nhất giữa địa chỉ của người gửi so với tất cả địa chỉ của kho nguồn và giữa địa chỉ của người nhận so với tất cả địa chỉ của kho đích. Để xác định được quãng đường (km) nhóm đã sử dụng **Google Map API** để có thể tính đường quãng đường giữa hai địa chỉ cần xác định. Từ đó nhóm đã tìm được quãng đường tối ưu ngắn nhất và sử dụng tổng quãng đường vẫn chuyển ngắn nhất giữa hai địa chỉ người gửi - người nhận để tính toán chi phí cần vận chuyển đơn hàng cho người gửi.

Công thức tính chi phí vận chuyển:

$$\text{Transportation Cost} = \text{Price} * \text{Distance}$$

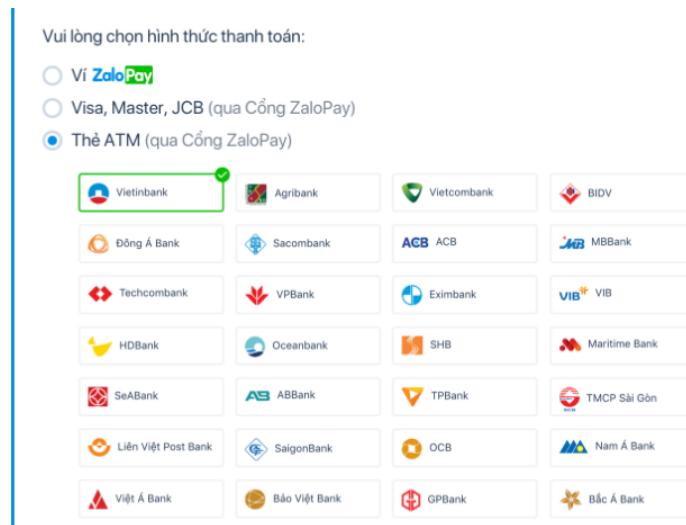
Trong đó,

Price: được tính dựa vào khối lượng vận chuyển từ **0 - 3kg** (mặc định **16000VND**) cứ thêm **0.5kg** sẽ cộng thêm **4000VND**.

Distance: quãng đường vận chuyển ngắn nhất giữa hai địa chỉ người gửi - người nhận.

3.6 Bài toán thanh toán chi phí vận chuyển

Thông thường để thanh toán chi phí vận chuyển thì người dùng sử dụng phương thức thu hộ (COD) hoặc thanh toán bằng tiền mặt khi nhận hàng, điều này gây ra nhiều bất tiện cho người dùng. Vì vậy, với xu hướng thanh toán không tiền mặt đang phát triển rộng rãi như hiện nay nên nhóm đã bắt kịp xu thế tích hợp cổng thanh toán online ZaloPay để người dùng có thể sử dụng thanh toán qua 3 hình thức thanh toán ví ZaloPay, Visa, Master, JCB (qua Cổng ZaloPay), thẻ ATM (qua Cổng ZaloPay).

**Hình 3.5:** Hình thức thanh toán

3.7 Bài toán phân phối đơn hàng cho tài xế

Trong các ứng dụng vận chuyển hàng thì bài toán phân phối là một bài toán rất hay, việc giải quyết bài toán này sao cho tối ưu có ảnh hưởng không nhỏ đến doanh thu của doanh nghiệp cũng như là chi phí của khách hàng.

Ở hệ thống vận chuyển liên tỉnh mà nhóm đang làm thì sẽ có 3 giai đoạn phân phối: Đầu tiên là phân phối đơn hàng cho tài xế đi lấy, tiếp theo là phân phối đơn hàng cho tài xế liên tỉnh, cuối cùng là phân phối đơn hàng cho tài xế đi giao.

- Phân phối đơn hàng cho tài xế đi lấy:** Sau khi đơn hàng được tạo ra bởi người dùng thì sẽ có 2 cách để nhận hàng. Thứ nhất là người dùng sẽ trực tiếp đem hàng đến kho, thứ hai là hệ thống sẽ tự động gán đơn đó cho tài xế và tài xế sẽ đi lấy hàng. Ở đây nhóm sẽ trình bày cụ thể về cách thứ hai. Sau khi đơn hàng được tạo ra thì hệ thống sẽ xác định được con đường ngắn nhất giữa 2 kho là kho nguồn và kho đích. Tài xế đi lấy sẽ là nhân viên ở kho nguồn. Hệ thống sẽ ngẫu nhiên thông báo cho một số tài xế ở kho nguồn là có đơn cần lấy. Tài xế nào đồng ý nhận đơn trước thì sẽ được ưu tiên đi lấy hàng trước.

Do hàng vận chuyển liên tỉnh, có thể số lượng sẽ rất lớn vì vậy nếu tài xế đầu tiên không lấy hết thì hệ thống sẽ tự động thông báo cho các tài xế tiếp theo. Cứ như vậy đến khi hàng được lấy hết và chở về kho nguồn.

Ở đây hệ thống sẽ thông báo cho một số tài xế của kho nguồn nhằm hạn chế việc tài xế được thông báo nhưng đang bận và chưa thể vận chuyển hàng về kho ngay

lập tức được. Điều đó có thể làm chậm trễ thời gian giao vận mà người dùng mong muốn gây ảnh hưởng đến trải nghiệm của người dùng.

- **Phân phối đơn hàng cho tài xế liên tỉnh:** Sau khi đơn hàng được vận chuyển đến kho nguồn thì việc tiếp theo hệ thống sẽ tự động gán đơn hàng đó cho tài xế liên tỉnh để vận chuyển hàng đến kho đích.

Việc gán liên tỉnh sẽ được thực hiện vào 1 khung giờ cố định trong ngày (ví dụ: 19h mỗi ngày) và sẽ gán đơn hàng lên xe của 1 tài xế đến khi xe đầy thì chuyển sang xe của tài xế khác.

Việc gán vào 1 khung giờ cố định trong ngày nhằm để không làm giảm hiệu năng của hệ thống bởi vì việc gán tài xế sẽ phải gọi service khác để lấy thông tin và lưu database. Việc gán đơn hàng cho 1 xe đến khi đầy nhằm hạn chế tối đa chi phí bỏ ra để vận chuyển các đơn hàng đó. Nếu cùng 1 số lượng đơn hàng nhưng được gán lên rất nhiều xe để vận chuyển liên tỉnh thì sẽ tốn chi phí rất nhiều.

- **Phân phối đơn hàng cho tài xế đi giao:** Sau khi đến kho đích thì đơn hàng tiếp tục được gán cho tài xế của kho đích đi giao. Việc gán đơn cho tài xế đi giao cũng được thực hiện vào 1 khung giờ cố định trong ngày và cách gán là lấy ra tất cả tài xế đi giao của kho đích và gán đều các đơn hàng cho tất cả tài xế. Việc gán này nhằm đảm bảo thời gian nhận hàng của người nhận là nhanh nhất có thể.

3.8 Bài toán lưu trữ và truy xuất dữ liệu thống kê

Trong quá trình hoạt động và vận hành của hệ thống thì người dùng thường xuyên xem lại những lịch sử mua hàng hay quản lý cần thống kê lượng tiền, lượng giao dịch xảy ra trong một khoảng thời gian nào đó.

Thông thường chúng ta sẽ lưu hết vào Database rồi query `SELECT COUNT(*) WHERE STARTDATE<=Date AND Date<=ENDDATE`. Việc query như vậy vào Database sẽ được xử lý rất lâu cho nên sẽ làm giảm hiệu năng của hệ thống cũng như trải nghiệm của người dùng.

Sau khi tìm hiểu và nghiên cứu thì nhóm sử dụng cấu trúc dữ liệu **HyperLoglog**[23] được hỗ trợ bởi Redis nhằm lưu trữ dữ liệu thống kê để mỗi khi cần chúng ta có thể truy xuất trong nháy mắt.

HyperLoglog giúp chúng ta tính gần đúng các giá trị khác nhau trong tập hợp, thuật toán này có độ chính xác cao mà vẫn giữ được dung lượng dữ liệu rất bé, hỗ trợ nhiều thao tác liên quan tới việc đếm 1 phần tử trong 1 hoặc nhiều tập hợp. Và quan trọng nhất là nó rất nhanh nên chúng ta tổ chức dữ liệu một cách hợp lý.

```

// Statistic Trans success
public void addTransSuccess(String date, long orderId) {
    String key = redisKey.genTransSuccessKeyByDate(date);
    RHyperLogLog<Long> rHyperLogLog = client.getHyperLogLog(key);
    rHyperLogLog.add(orderId);
}

public long countTransSuccess(String date) {
    String key = redisKey.genTransSuccessKeyByDate(date);
    RHyperLogLog<Long> rHyperLogLog = client.getHyperLogLog(key);
    return rHyperLogLog.count();
}

```

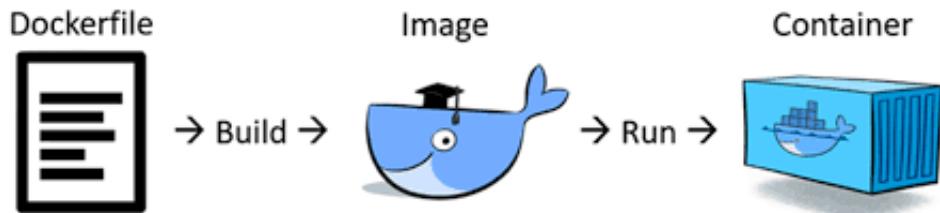
Hình 3.6: Ví dụ về HyperLoglog

Trong đoạn code trên ta lưu thông tin của số giao dịch thành công trong 1 ngày. Khi có yêu cầu lấy thông tin về số giao dịch thành công trong một khoảng thời gian nào đó thì ta chỉ cần gọi hàm như ví dụ để lấy. Do các thông tin này được lưu trên memory nên việc xử lý rất nhanh đảm bảo trải nghiệm tốt cho người dùng.

3.9 Bài toán triển khai các services và cơ sở hạ tầng

Việc triển khai hệ thống có thể được hỗ trợ bởi rất nhiều bên như: Git, Heroku, ... thông thường chúng ta chỉ đưa mã nguồn lên Git, thiết lập một vài bước là có thể dễ dàng triển khai hệ thống lên cho khách hàng sử dụng. Nhưng do nhóm sử dụng khá nhiều hạ tầng liên quan để tăng cường hiệu suất cho hệ thống như: Redis, Kafka, Mysql, Message Queue, Cassandra, ... cho nên việc sử dụng các nền tảng đó sẽ không được hỗ trợ một cách tối đa. Ngoài ra để hiểu rõ hơn về quy trình triển khai một ứng dụng cũng như là cơ hội để có thể áp dụng các kiến thức đã được học về Network, System, Security, ... vào thực tế thì nhóm quyết định thuê 1 VPS (Virtual Private Server) trên Google Cloud Platform để tự cấu hình và triển khai hệ thống của mình.

Như đã trình bày ở phần Công nghệ để triển khai nhanh chóng, tiện lợi các services và cơ sở hạ tầng đi kèm thì nhóm sử dụng nền tảng **Docker** và **Docker compose**.

**Hình 3.7:** Flow docker

Đầu tiên thì mỗi service cần phải có một **Dockerfile**, sau đó tiến hàng build Dockerfile thì ta có được **Image**, cuối cùng chỉ cần chạy Image đó là ta đã deploy được service trong **Container**.

```

FROM maven:3.6-jdk-8 as builder
COPY . .
RUN mvn -DskipTests clean package spring-boot:repackage

FROM openjdk:8-jdk-alpine

COPY src/main/resources/ ./resources/
COPY --from=builder target/*.jar ./order-management.jar

ENTRYPOINT ["java", "-jar", "order-management.jar"]
  
```

Hình 3.8: Dockerfile OrderService

Dockerfile thực thi các câu lệnh để build ra file *.jar và chạy file jar đó trong containers. Ở đây thì nhóm có tìm hiểu một số kỹ thuật để tăng cường hiệu suất cho Docker như sau:

- **Multi-stage build:** Thông thường để chạy được service ta cần phải tạo môi trường, cấu hình package, module, ... và sẽ cần dung lượng khá cao, làm cho Image nặng lên từ đó làm tăng thời gian xây dựng, thực thi và làm giảm hiệu suất của Docker. Vì thế nhóm sẽ build Image trên 2 stage, ở stage thứ nhất thì ta thiết lập môi trường, các package liên quan sau đó tiến hành build file *.jar. Ở stage thứ 2 ta chỉ cần copy file *.jar qua và thực thi. Điều này giúp cho quá trình xây dựng và chạy Image trở nên nhanh hơn đáng kể.

```

version: '3'
services:
  order-management:
    container_name: order-management
    build: .
    environment:
      SPRING_DATA_CASSANDRA_CONTACT_POINTS: cassandra
    ports:
      - 8080:8080
    restart: always

networks:
  default:
  external:
    name: bk-shipment

```

Hình 3.9: Docker compose OrderService

- **Sử dụng base image được dùng trên alpine:** Alpine Linux là một distro linux dựa trên musl và busybox, được phát triển chú trọng về đơn giản, bảo mật và hiệu quả tài nguyên. Ở đây nhóm đã sử dụng base image đó là: openjdk:8-jdk-alpine để thực thi file *.jar.

Một hệ thống thông thường sẽ phải triển khai rất nhiều container khác nhau. Khi có nhiều container việc phải start, stop, run từng container trở nên khó khăn và khó quản lý. Vì thế nhóm đã dùng **Docker compose** cho phép quản lý nhiều container một cách rất dễ dàng.

Tương tự như các service thì các cơ sở hạ tầng như: MySQL, Cassandra, Redis, Kafka, ActiveMq, ... cũng được xây dựng và triển khai dựa trên Docker.

```
dungdv5@bkshipment:~$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
4edcc6005604 order-management_order-management "java -jar order-man..." 20 hours ago Up 20 hours
e233b820fb0f notify-service_notify-service "java -jar notify-se..." 21 hours ago Up 21 hours
4c83589cbc21 support-service_support-service "java -jar support-s..." 22 hours ago Up 22 hours
a7f3ae0b2bbf wurstmeister/kafka "start-kafka.sh" 22 hours ago Up 22 hours
45ee3a75073d wurstmeister/zookeeper "/bin/sh -c '/usr/sb..." 22 hours ago Up 22 hours
0abde8a59a94 cassandra "docker-entrypoint.s..." 22 hours ago Up 22 hours
138bdb79df22 webcenter/activemq:5.14.3 "/app/run.sh" 22 hours ago Up 22 hours
b7c467cfb5c0 redis:6.2.1 "docker-entrypoint.s..." 22 hours ago Up 22 hours
96261915f335 mysql:5.7 "docker-entrypoint.s..." 22 hours ago Up 22 hours
02cec6e0b9a7 nginx:1.20.1 "/docker-entrypoint...." 4 days ago Up 22 hours
```

Hình 3.10: Kết quả

3.10 Bài toán bảo mật các service trong private network, cấu hình SSL, cân bằng tải

Thông thường sau khi triển khai thì các service được gọi thông qua **IP:port/service-name/...**, điều này có nghĩa là service sẽ được public ra bên ngoài. Điều này có thể gặp một số rủi ro về vấn đề an ninh và bảo mật. Vì thế nhóm đã sử dụng NGINX như là một **Reverse Proxy** để tăng cường bảo mật, cũng như cân bằng tải.

```
worker_processes 1;
events { worker_connections 1024; }

http {
    ssl_session_cache    shared:SSL:10m;
    ssl_session_timeout 10m;

    server {
        server_name dungdang.codes;
        listen 80 default;
        listen 443 ssl;
        keepalive_timeout 70;

        ssl_certificate      /etc/nginx/cert/server.crt;
        ssl_certificate_key /etc/nginx/cert/server.key;
        ssl_protocols        TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers          HIGH:!aNULL:!MD5;

        location /order-management {
            proxy_pass http://order-management:8080;
        }

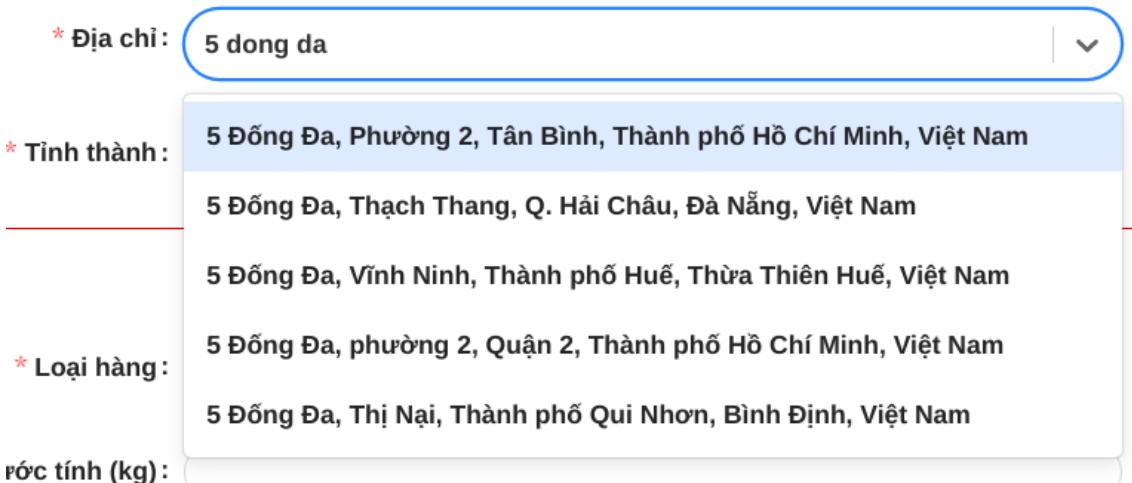
        location /support-service {
            proxy_pass http://support-service:7070;
        }
    }
}
```

Hình 3.11: Cấu hình của NGINX

Như trên file đã cấu hình thì server Front end có thể gửi yêu cầu xuống server Back end thông qua domain là: **dungdang.codes** ở port 80 cho HTTP và 443 cho HTTPS. Đối với những yêu cầu có domain là: [https://dungdang.codes/order-management/...](https://dungdang.codes/order-management/) sẽ được route vào service được triển khai ở port 8080 còn [https://dungdang.codes/support-service/...](https://dungdang.codes/support-service/) thì sẽ được route vào service triển khai ở port 7070. Điều này đã giúp che giấu và bảo mật các services chạy trong private network.

3.11 Bài toán gợi ý địa điểm khi người dùng nhập địa chỉ

Khi điền địa chỉ muốn gửi trong giao diện tạo đơn hàng, hệ thống sẽ tự động gợi ý địa điểm cho người dùng khi người dùng nhập vài từ trong địa chỉ. Điều này được thực hiện nhờ sử dụng API **Places Autocomplete** mà Google cung cấp. Cụ thể, nhóm đã đăng ký một tài khoản trên Google Cloud Platform, kích hoạt dịch vụ **Maps JavaScript API** và tiếp tục làm theo hướng dẫn để lấy được một api key giúp cho ta có thể gọi và sử dụng api của Google. Việc sau đó là chỉ cần gọi API trên ở phía client-side code. Ví dụ cho việc gợi ý địa điểm trong ứng dụng như sau:



Hình 3.12: Giao diện gợi ý địa điểm khi người dùng nhập địa chỉ

Từ địa chỉ gợi ý mà người dùng chọn, nhóm có thể xác định được thành phố người dùng muốn gửi đến là ở đâu để từ đó tiếp tục thực hiện quy trình vận chuyển liên tỉnh.

3.12 Bài toán giảm thời gian load trang web lần đầu tiên

Như đã trình bày ở phần công nghệ, nhóm sử dụng dịch vụ của gitlab pages để host và serve các static files (HTML, CSS, JS, image vv...) khi người dùng truy cập vào tên miền của trang web. Tuy vậy, gitlab pages không tự động compress (nén) các static files theo định dạng gzip cho người dùng. Chính vì thế nhóm đã tự nén các file lại theo định dạng gzip khi build code để giúp tiết kiệm băng thông cũng như giúp thời gian load trang web trở nên nhanh hơn khi người dùng truy cập.

Cơ chế hoạt động cơ bản của gzip trong giao tiếp giữa máy khách và máy chủ:

- Bước 1: Trình duyệt gửi một header request đến server (máy chủ) để thông báo rằng nó chấp nhận file được nén. Header này sẽ có dạng như sau: **Accept-Encoding: gzip**.
- Bước 2: Server gửi phản hồi đồng ý và truyền file dữ liệu đã được nén cho trình duyệt với tín hiệu: **Content-Encoding: gzip**

4

Phân tích và thiết kế hệ thống

4.1 Phân tích yêu cầu

4.1.1 Yêu cầu người dùng

- **Đối với người dùng**
 - **Đăng ký, đăng nhập đổi mật khẩu** cho người dùng.
 - **Chỉnh sửa thông tin cá nhân**: Người dùng có thể dễ dàng thay đổi thông tin cá nhân của mình.
 - **Tạo đơn hàng**: Lúc tạo đơn hàng người dùng có thể tự đến gửi hàng trực tiếp tại kho hoặc người dùng lựa chọn có tài xế đến lấy, phí vận chuyển có thể là người gửi thanh toán hoặc người nhận thanh toán.
 - **Hủy đơn hàng**: Trong khoảng thời gian đơn hàng được xác nhận bởi tài xế thì đơn hàng có thể bị hủy. Chú ý rằng sau khi đơn hàng đã được xác nhận thì người dùng không thể hủy.
 - **Xem danh sách các đơn hàng**: Người dùng có thể dễ dàng xem thông tin đơn hàng. Danh sách đơn hàng có thể được lọc theo trạng thái, theo mã đơn hàng hoặc theo từng khoảng thời gian khác nhau nhanh chóng.
 - **Xem nợ công và thanh toán**: Người dùng có thể xem các đơn hàng đã hoàn tất và tiến hành thanh toán trung gian qua Ví điện tử hoặc thẻ ngân hàng.
 - **Nhận thông báo**: Sau khi đơn hàng được xác nhận bởi tài xế, thì người dùng sẽ liên tục nhận được thông báo mỗi khi chuyển trạng thái của đơn hàng. Điều này giúp cho người dùng có thể theo dõi đơn hàng chặt chẽ và dễ dàng dự báo thời gian nhận hàng của mình.
 - **Tìm kiếm đơn hàng**: Đối với người nhận không phải là người dùng của hệ thống thì vẫn có thể dễ dàng xem thông tin các đơn hàng thông qua số điện thoại của mình hoặc theo dõi yêu cầu thông qua mã yêu cầu.

- **Yêu cầu hỗ trợ:** Khi có các vấn đề thắc mắc thì người dùng có thể gửi yêu cầu để quản lý trực tiếp phản hồi. Danh sách các yêu cầu có thể được lọc theo mã yêu cầu, ngày, tháng, năm.

- **Đối với tài xế**

- **Chỉnh sửa thông tin cá nhân:** Tài xế có thể dễ dàng thay đổi thông tin cá nhân của mình.
- **Nhận đơn hàng:** Sau khi đơn hàng được tạo ra thì tài xế có thể nhận đơn và tiến hành vận chuyển hàng hóa.
- **Xem danh sách kiện hàng cần lấy:** Hệ thống sẽ tự động gán đơn đối với tài xế liên tỉnh và tài xế đi giao cho nên tài xế cần theo dõi danh sách đơn hàng cần lấy để lấy đúng kiện hàng mà mình vận chuyển.
- **Xem danh sách kiện hàng cần giao:** Tài xế có thể xem thông tin chi tiết về các đơn hàng mà mình cần giao.
- **Xác nhận đơn hàng với người dùng:** Trong quá trình giao nhận hàng giữa tài xế và người dùng (người nhận, người gửi) thì tài xế đều phải xác nhận đơn hàng.
- **Báo cáo vấn đề:** Trong quá trình vận chuyển có thể xảy ra các vấn đề ngoài ý muốn như: làm thất lạc hàng, gặp sự cố ngoài ý muốn không thể đảm bảo thời gian giao hàng,... thì tài xế có thể báo cáo lên để quản lý có thể giải quyết.
- **Nhận thông báo:** Mỗi khi đơn hàng được gán cho tài xế thì sẽ có mail thông báo cho tài xế. Việc này giúp tài xế nắm thông tin đơn hàng, đảm bảo việc giao nhận diễn ra kịp thời.

- **Đối với thủ kho**

- **Chỉnh sửa thông tin cá nhân** thủ kho.
- **Nhập kho:** Sau khi đơn hàng vận chuyển đến kho thì thủ kho có trách nhiệm xác nhận đơn hàng đúng về số lượng và đảm bảo chất lượng nguyên vẹn. Sau khi xác nhận thì tiến hành nhập kho để lưu trữ và phân phối tiếp tục cho tài xế.
- **Xuất kho:** Sau khi đơn hàng đã phân phối cho tài xế thì thủ kho có trách nhiệm kiểm tra hàng một lần nữa và bàn giao cho tài xế.
- **Xuất hóa đơn:** Trong quá trình nhập kho và xuất kho thì thủ kho có thể xuất hóa đơn để xác nhận đã nhập hàng hoặc bàn giao hàng hóa cho tài xế.
- **Xem danh sách đơn hàng:** Thủ kho có thể xem danh sách đơn hàng đang ở trong kho mà mình quản lý. Danh sách đơn hàng được lọc theo mã hoặc có thể theo ngày, tháng, năm.
- **Xem lịch sử nhập xuất:** Thủ kho có thể xem lịch sử nhập xuất đơn hàng vào/ra kho.

- **Đối với quản lý**

- **Tạo tài khoản cho tài xế:** Quản lý có thể tạo tài khoản và phân kho cho tài xế khi tài xế mới nhận việc.
- **Xem biểu đồ thống kê:** Quản lý có thể xem biểu đồ thống kê về tình hình hoạt động của các kho, tình hình vận chuyển hàng hóa.
- **Xem thông tin** các tài xế, thủ kho và kho tương ứng của hệ thống.
- **Xử lý sự cố:** Khi nhận được sự cố từ tài xế thì quản lý có thể xử lý sự cố và cập nhật trạng thái đơn hàng cho người dùng.
- **Xử lý yêu cầu từ người dùng:** Khi người dùng yêu cầu thì quản lý có thể phản hồi thắc mắc hoặc các phản ánh từ người dùng.

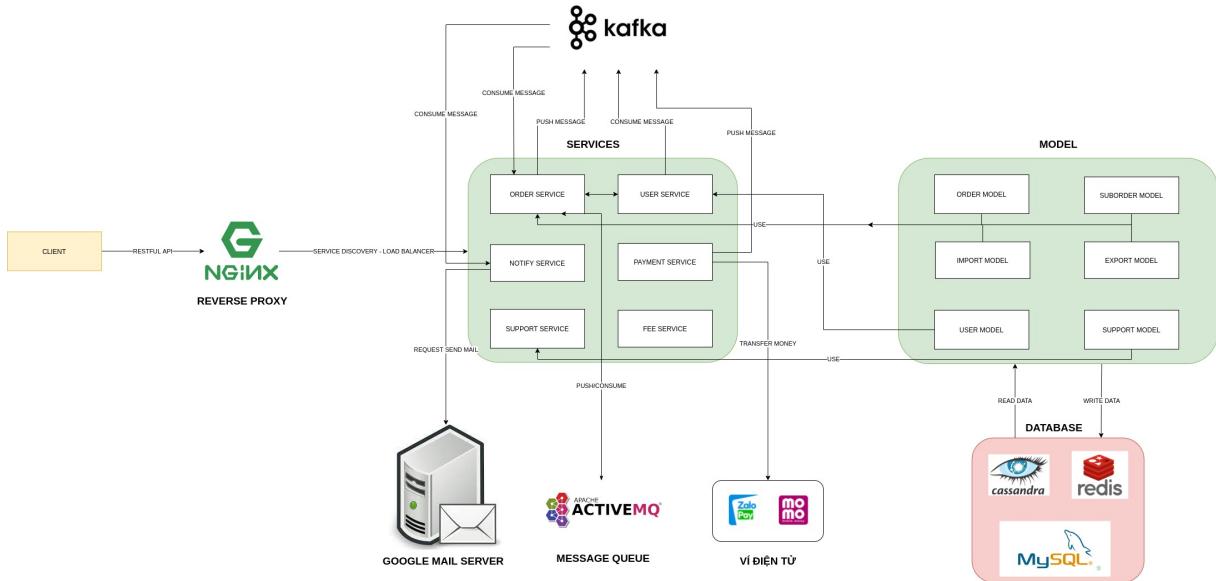
4.1.2 Yêu cầu hệ thống

Hệ thống vận chuyển hàng hóa liên tỉnh cần đáp ứng một số yêu cầu về hệ thống như sau:

- Thời gian xử lý mỗi request từ người dùng không quá 200 ms trong điều kiện bình thường và 2s trong điều kiện tải cao.
- Hệ thống gửi thông báo cho người dùng hoặc tài xế không quá 2s.
- Hệ thống có thể chịu tải 100 yêu cầu mỗi giây.
- Thời gian tải hóa đơn nhập hàng/xuất hàng không quá 1s trong điều kiện bình thường và 5s trong điều kiện tải cao.
- Hệ thống cần mã hóa thông tin giao tiếp giữa máy khách và máy chủ.
- Hệ thống phải được thiết kế để khi có nhu cầu mở rộng, thêm tính năng mới phải dễ dàng.
- Hệ thống cần xuất thống kê về số lượng đơn hàng được vận chuyển thành công, thất bại, doanh thu vào ngày cuối cùng trong tháng.

4.2 Thiết kế hệ thống

4.2.1 Kiến trúc hệ thống



Hình 4.1: Kiến trúc hệ thống

- **User Service**

- Service chịu trách nhiệm quản lý user trong hệ thống. Ở đây sẽ lưu trữ Database của user.
- Mỗi khi người dùng đăng nhập vào hệ thống thì sẽ vào service này để xác thực: Ban đầu người dùng được yêu cầu sẽ nhập username và password, Sau khi xác thực là đúng người dùng đó thì hệ thống sẽ sinh ra JWT (Json Web Token) để gửi trả về cho người dùng. Sau này cứ mỗi request lên hệ thống thì sẽ gửi kèm theo JWT để hệ thống có thể xác thực và phân quyền.
- Người dùng có thể lấy thông tin để xem và chỉnh sửa

- **Order Service**

- Service chịu trách nhiệm tạo, xử lý, lưu trữ các thông tin về đơn hàng cho người dùng.
- Ngoài ra service còn chịu trách nhiệm phân phối các đơn hàng đến tài xế
- Đây là service chịu tải khá lớn trong hệ thống vì nhóm sử dụng kết hợp các giải pháp sau nhằm tăng hiệu suất cho service:

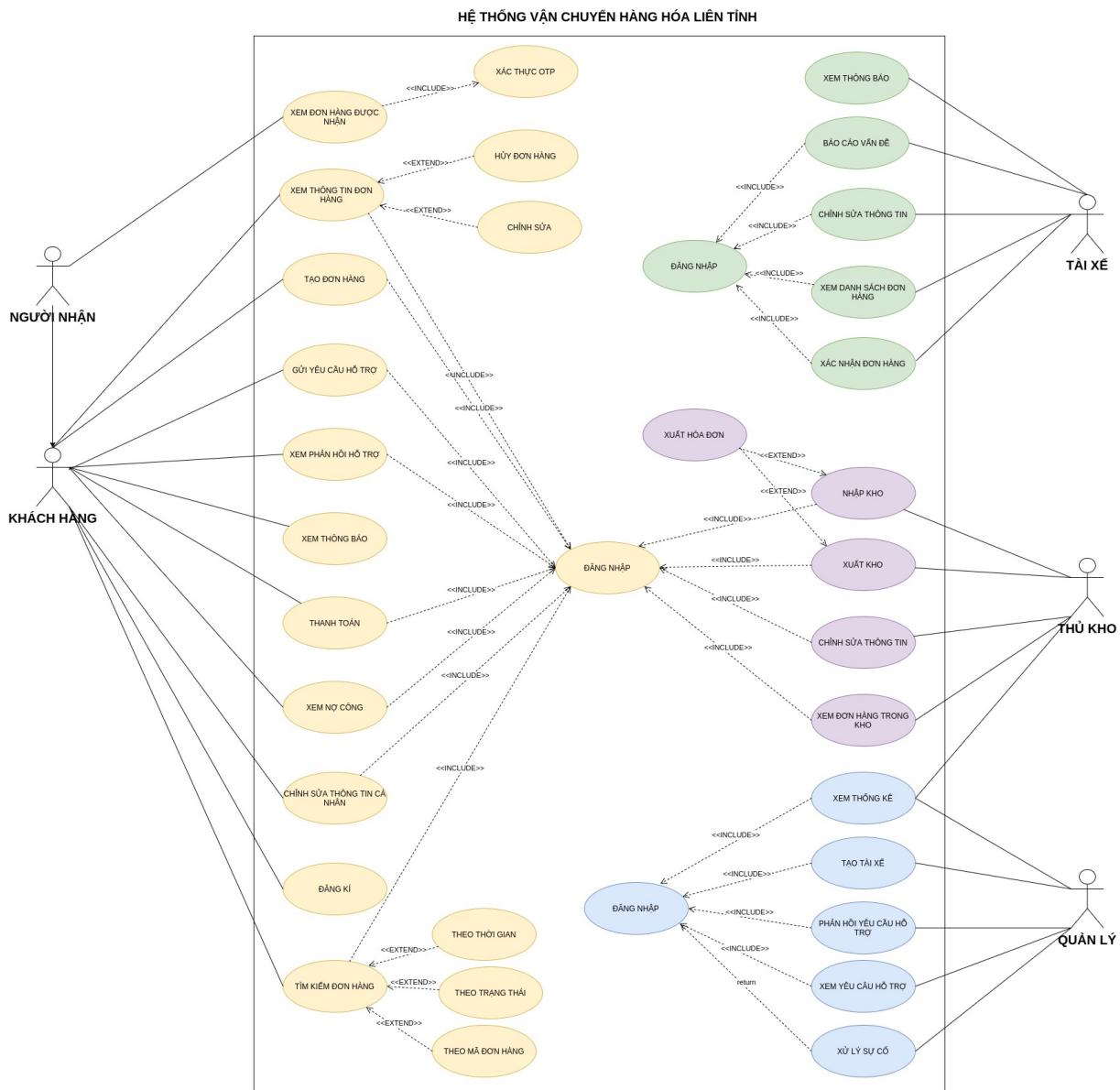
- * Sử dụng **Message Queue** để hiện thực cơ chế bất đồng bộ cho các tác vụ: Tạo đơn hàng, ghi dữ liệu vào Database.
- * Sử dụng **Kafka** để gửi message cho service khác
- * Trong quá trình xử lý thì dữ liệu của đơn hàng được lưu trên Cache. Đến khi nào có trạng thái cuối thì đơn hàng mới được lưu vào Database. Bởi vì đơn hàng được lấy và thay đổi trạng thái rất nhiều lần trong quá trình xử lý, nếu ghi vào Database (thực hiện nhiều I/O operations) sau mỗi lần xử lý như vậy thì sẽ làm giảm hiệu năng của hệ thống rất nhiều. Việc xử lý dữ liệu trên Cache cũng có thể gặp rủi ro như Cache Server sập thì dữ liệu có thể mất hết, nhưng rất may Redis có hỗ trợ cơ chế Persistence (định kỳ sao lưu dữ liệu) nên rủi ro này có thể chấp nhận được.
 - Mỗi khi đơn hàng thay đổi trạng thái hoặc tài xế được phân phối đơn thì service sẽ produce message vào **Kafka Broker** để **Notify Service** consume và tiếp tục xử lý
- **Notify Service**
 - Service chịu trách nhiệm quản lý các thông báo cho hệ thống.
 - Service sẽ consume message từ **Order Service** và gửi yêu cầu về mail cho **Google Mail Server**.
 - Hệ thống sẽ gửi thông báo mỗi khi:
 - * Gửi thông báo cho khách hàng mỗi khi trạng thái đơn hàng thay đổi.
 - * Gửi thông báo cho tài xế mỗi khi được phân phối đơn.
 - * Gửi mã xác thực OTP
- **Fee Service**
 - Service chịu trách nhiệm hiện thực giải thuật, công thức để tính toán các khoản phí phát sinh cho khách hàng.
 - Database lưu thông tin về phí của mỗi đơn hàng để người dùng có thể đối soát.
- **Payment Service**
 - Service này sẽ liên kết với 1 bên thứ 3 như: ngân hàng, ví điện tử, để thực hiện chức năng thanh toán cho người dùng.
- **Reverse Proxy**
 - Hệ thống sử dụng NGINX như là một **Reverse Proxy** nhằm thực hiện một số chức năng như:

- * **Bảo mật:** Kiểm soát các yêu cầu của Client gửi tới, nếu hợp lệ sẽ luân chuyển đến các service bên trong để xử lý. Che giấu địa chỉ IP của các service bên trong nhằm giúp các service tránh khỏi các cuộc tấn công mạng. Ngoài ra còn có thể dễ dàng thiết lập SSL nhằm mã hóa dữ liệu trong quá trình giao tiếp giữa Client và các service bên trong.
- * **Cân bằng tải:** Mỗi service có thể được deploy với nhiều instance để có thể chịu tải cao hơn. Reverse proxy sẽ thực hiện một số giải thuật như: Round Robin, Least Connection, ... để đẩy các request vào cho các service xử lý.

- **Database**

- **MySQL:** Hệ thống sử dụng MySql để lưu các thông tin ít được truy xuất như log, cấu hình service,...
- **Cassandra:** Với khả năng đọc/ghi có tốc độ nhanh cùng với đó là khả năng chịu lỗi cao hệ thống sử dụng Cassandra để lưu các thông tin về Order, Hóa đơn nhập/xuất kho,...
- **Redis:** Với Redis thì dữ liệu được lưu trữ trong memory vì thế giúp cho service truy xuất thông tin rất nhanh chóng. Hệ thống sẽ lưu các thông tin thường xuyên được truy xuất bởi người dùng như thông tin về yêu cầu, đơn hàng,... Ngoài ra còn tận dụng cấu trúc dữ liệu đa dạng trong Redis để giải quyết một số bài toán phức tạp.

4.2.2 Thiết kế và đặc tả Use Case



Hình 4.2: Sơ đồ Use case

- **Use case cho Khách Hàng**

- **Tạo đơn hàng:** Người gửi cần điền những thông tin sau:

- * Bên gửi: Hệ thống sẽ tự động nhập thông tin khi người gửi đăng nhập bao gồm:
 - Tên người gửi
 - Số điện thoại
 - Địa chỉ, chia làm 4 phần gồm Số địa chỉ + tên đường, Phường - Xã, Quận - Huyện, Thành phố - Tỉnh.
 - * Bên nhận:
 - Tên người nhận
 - Số điện thoại
 - Địa chỉ, tương tự như bên gửi. Khi người gửi nhập địa chỉ hệ thống sẽ tự động lấy chính xác địa chỉ thông qua Google Map API.
 - * Thông tin về món hàng muốn gửi:
 - Loại hàng hóa
 - Khối lượng (để tính phí vận chuyển). Nhân viên đến lấy hàng sẽ tiến hành xác nhận bằng cách đo lại và sẽ yêu cầu khách hàng cập nhật lại khối lượng nếu như khối lượng đã nhập trước đó là không chính xác.
 - Giá trị hàng hóa (VND): Giá trị ước tính của hàng hóa để đền bù khi gặp sự cố.
 - * Lựa chọn tính phí cho bên gửi hoặc bên nhận
 - * Lựa chọn gửi hàng/nhận hàng tại kho sẽ không tính phí nội tỉnh (phí liên tỉnh luôn có). Ngược lại sẽ tính phí nội tỉnh (gửi và nhận giống nhau).
 - * Hệ thống sẽ tự tính cước vận chuyển dựa trên khoảng cách bên gửi/nhận và khối lượng món hàng.
 - * Các thông tin ghi chú khác (Người gửi tự điền vào ô textbox).

Use case name	Tạo đơn hàng
Actor	Khách hàng
Description	Người dùng có thể tạo đơn hàng của mình
Preconditions	Người dùng đã đăng nhập vào hệ thống
Postconditions	Đơn hàng được tạo ra, các thông tin về đơn hàng được lưu trữ, đơn hàng sẽ được gán cho tài xế đến lấy (nếu người dùng không đến kho gửi hàng)
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ khách hàng chọn Tab "Quản lý đơn hàng". 2. Chọn "Lên đơn hàng" ở góc trên bên trái màn hình. 3. Nhập đầy đủ các thông tin yêu cầu. 4. Nhấn "Tạo đơn".

Bảng 4.1: Tạo đơn hàng

- **Xem đơn hàng được nhận:** Người nhận có thể nhập số điện thoại để xem các đơn hàng được nhận, các đơn hàng sẽ được chia theo trạng thái đã định nghĩa ở trên và người gửi có thể xem danh sách các đơn hàng lọc theo trạng thái đó.

Use case name	Xem đơn hàng được nhận
Actor	Người nhận
Description	Người nhận có thể xem đơn hàng được nhận thông qua số điện thoại mà không cần đăng nhập
Preconditions	Thông tin đơn hàng phải có số điện thoại và email của người nhận
Postconditions	Danh sách đơn hàng được hiển thị cho người nhận
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang landing page người nhận chọn "Xem danh sách đơn hàng". 2. Nhập số điện thoại. 3. Nhập mã OTP nhận được thông qua Email.

Bảng 4.2: Xem đơn hàng được nhận

- **Xem thông tin đơn hàng:** Khách hàng có thể xem thông tin đơn hàng theo các trạng thái khác nhau, theo mã đơn hàng hoặc theo khoảng thời gian nào đó.

Use case name	Xem thông tin đơn hàng
Actor	Khách hàng
Description	Khách hàng xem thông tin đơn hàng sau khi đã đăng nhập vào hệ thống
Preconditions	Khách hàng đã đăng nhập vào hệ thống
Postconditions	Danh sách đơn hàng được hiển thị. Khách hàng có thể lọc đơn hàng theo mã đơn hàng, trạng thái, ngày tháng.
Normal Flow	<ol style="list-style-type: none"> Ở trang chủ nhấn "Quản lý đơn hàng". Mặc định sẽ hiển thị tất cả đơn hàng của khách hàng đó. Khách hàng có thể lọc theo các tiêu chí khác nhau nếu cần

Bảng 4.3: Xem thông tin đơn hàng

- **Hủy đơn hàng:** Nếu vẫn đang trong quá trình bàn giao và xác nhận (trong trạng thái **Chờ bàn giao**), thì người gửi có thể hủy đơn hàng nếu muốn.

Use case name	Hủy đơn hàng
Actor	Người gửi
Description	Người gửi có thể hủy nếu đơn hàng chưa được bàn giao cho tài xế.
Preconditions	Người gửi đã đăng nhập vào hệ thống
Postconditions	Đơn hàng bị hủy
Normal Flow	<ol style="list-style-type: none"> Ở trang chủ nhấn "Quản lý đơn hàng". Mặc định sẽ hiển thị tất cả yêu cầu của khách hàng đó. Tại đơn hàng khách hàng nhấn vào "Hủy đơn hàng". Nhấn "Xác nhận".

Bảng 4.4: Hủy đơn hàng

- **Gửi yêu cầu hỗ trợ:** Trong quá trình vận chuyển nếu có thắc mắc nào thì khách hàng có thể gửi yêu cầu để quản lý hỗ trợ.

Use case name	Gửi yêu cầu hỗ trợ
Actor	Khách hàng
Description	Khách hàng gửi yêu cầu và quản lý sẽ trực tiếp phản hồi yêu cầu đó.
Preconditions	Khách hàng đã đăng nhập vào hệ thống
Postconditions	Yêu cầu được submit và chờ quản lý phản hồi
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Yêu cầu hỗ trợ". 2. Nhấn "Lên yêu cầu". 3. Nhập các thông tin cần thiết như: Mã đơn hàng, trình bày nội dung cần hỗ trợ. 4. Nhấn "Xác nhận".

Bảng 4.5: Gửi yêu cầu hỗ trợ

- **Xem phản hồi hỗ trợ:** Sau khi yêu cầu hỗ trợ được phản hồi bởi quản lý thì khách hàng có thể xem thông tin phản hồi đó.

Use case name	Xem phản hồi hỗ trợ
Actor	Khách hàng
Description	Khách hàng xem yêu cầu hỗ trợ được phản hồi bởi quản lý
Preconditions	Khách hàng đã đăng nhập vào hệ thống
Postconditions	Hiển thị danh sách yêu cầu phản hồi
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Yêu cầu hỗ trợ". Mặc định sẽ hiển thị ra thông tin các yêu cầu và phản hồi của quản lý (nếu yêu cầu đã được xử lý) 2. Khách hàng có thể lọc yêu cầu theo: Mã yêu cầu, ngày tháng.

Bảng 4.6: Xem phản hồi hỗ trợ

- **Xem thông báo:** Mỗi khi trạng thái đơn hàng thay đổi thì khách hàng đều nhận được email thông báo để khách hàng có thể nắm bắt được tình trạng của đơn hàng. Ngoài ra một số chức năng cần phải nhập OTP để xác thực khách hàng thì cũng được gửi thông qua email.

Use case name	Xem thông báo
Actor	Khách hàng
Description	Khách hàng xem thông báo của mình thông qua Email
Preconditions	Khách hàng đã đăng nhập vào email của mình
Postconditions	Khách hàng xem được thông báo của mình từ BKShipment
Normal Flow	<ol style="list-style-type: none"> 1. Người dùng đăng nhập Email của mình 2. Mở hộp thư đến

Bảng 4.7: Xem thông báo

- **Xem nợ công:** Sau khi đơn hàng được xác nhận là hoàn thành thì sẽ có thông kê về nợ công để khách hàng dễ dàng quan sát và thanh toán.

Use case name	Xem nợ công
Actor	Khách hàng
Description	Khách hàng có thể xem nợ công của mình
Preconditions	Khách hàng đã đăng nhập vào hệ thống
Postconditions	Hiển thị danh sách nợ công với tương ứng với các đơn hàng của người dùng
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Thanh toán". Mặc định sẽ hiển thị ra thông tin các đơn hàng đã hoàn thành và số tiền phải thanh toán.

Bảng 4.8: Xem nợ công

- **Thanh toán:** Sau khi xem xét nợ công của các đơn hàng thì khách hàng có thể tiến hành thanh toán chi phí cho doanh nghiệp. Khách hàng có thể thanh toán từng đơn hàng hoặc chọn nhiều đơn hàng cùng một lúc.

Use case name	Thanh toán
Actor	Khách hàng
Description	Khách hàng thanh toán đơn hàng thông qua trung gian là Ví điện tử
Preconditions	Khách hàng đã đăng nhập vào hệ thống
Postconditions	Đơn hàng đã được thanh toán. Tiền trong ví điện tử của khách hàng được trả cho doanh nghiệp.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Thanh toán". Mặc định sẽ hiển thị ra thông tin các đơn hàng đã hoàn thành và số tiền phải thanh toán. 2. Khách hàng có thể lựa chọn từng đơn hàng hoặc nhiều đơn hàng. 3. Chọn phương thức thanh toán. 4. Nhấn "Xác nhận".

Bảng 4.9: Thanh toán

- **Chỉnh sửa thông tin cá nhân:** Khách hàng có thể thay đổi một số thông tin cá nhân khi cần thiết.

Use case name	Chỉnh sửa thông tin cá nhân
Actor	Khách hàng
Description	Khách hàng có thể thay đổi thông tin cá nhân của mình.
Preconditions	Khách hàng đã đăng nhập vào hệ thống
Postconditions	Thông tin cá nhân của khách hàng được thay đổi.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Chỉnh sửa thông tin". 2. Khách hàng có thể thay đổi một số thông tin như: Email, SĐT, tên tài khoản, địa chỉ, mật khẩu. 3. Nhấn "Cập nhật".

Bảng 4.10: Chính sửa thông tin cá nhân

- **Use case cho Tài xế**

- **Xem thông báo:** Tương tự như Khách hàng thì khi mỗi đơn hàng được gán cho Tài xế thì sẽ có mail thông báo giúp cho tài xế tiếp nhận thông tin kịp thời.
- **Báo cáo vấn đề:** Trong quá trình vận chuyển có thể xảy ra các trường hợp ngoài ý muốn như mất hàng, chậm trễ việc chuyển hàng thì Tài xế có thể báo cáo vấn đề lên để Quản lý có thể giải quyết kịp thời.

Use case name	Báo cáo vấn đề
Actor	Tài xế
Description	Tài xế báo cáo vấn đề lên Quản lý để xử lý
Preconditions	Tài xế đã đăng nhập vào hệ thống
Postconditions	Nội dung vấn đề được gửi lên để chờ Quản lý xử lý.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Báo cáo vấn đề". 2. Nhấn "Tạo yêu cầu". 3. Nhập các thông tin cần thiết 4. Nhấn "Gửi yêu cầu".

Bảng 4.11: Báo cáo vấn đề

- **Xem danh sách đơn hàng:** Tài xế có thể xem danh sách đơn hàng cần lấy, danh sách đơn hàng đã giao, đã hoàn thành, đã hủy, các đơn hàng đang ở trong xe...

Use case name	Xem danh sách đơn hàng
Actor	Tài xế
Description	Tài xế xem các thông tin về đơn hàng
Preconditions	Tài xế đã đăng nhập vào hệ thống
Postconditions	Đơn hàng được hiển thị. Tài xế có thể lọc theo mã đơn hàng, ngày tháng, trạng thái.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Quản lý đơn hàng". 2. Chọn tab "Đơn hàng cần lấy", "Đơn hàng đang giao", "Đơn hàng đã giao" để xem thông tin về đơn hàng cần biết.

Bảng 4.12: Xem danh sách đơn hàng

- **Xác nhận đơn hàng:** Mỗi khi giao nhận hàng thì tài xế có nhiệm vụ phải kiểm tra kĩ tình trạng và số lượng của kiện hàng. Sau đó cần phải xác nhận lại với hệ thống và phải chịu hoàn toàn trách nhiệm nếu đơn hàng có sai sót về tình trạng và số lượng.

Use case name	Xác nhận đơn hàng
Actor	Tài xế
Description	Tài xế xác nhận đơn hàng
Preconditions	Tài xế đã đăng nhập vào hệ thống
Postconditions	Đơn hàng được xác nhận với hệ thống. Hệ thống lưu trữ lại thông tin mới nhất về đơn hàng.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Quản lý đơn hàng". 2. Chọn đơn hàng cần xác nhận. 3. Chính sửa các thông tin sai sót nếu có. 4. Nhấn "Xác nhận".

Bảng 4.13: Xác nhận đơn hàng

- **Use case cho Thủ kho**

- **Nhập kho:** Sau khi hàng được vận chuyển về kho thì thủ kho chịu trách nhiệm kiểm tra lại tình trạng và số lượng của đơn hàng. Nếu không có vấn đề gì xảy ra thì thủ kho nhập hàng vào kho để lưu trữ và chờ phân phối cho tài xế.

Use case name	Nhập kho
Actor	Thủ kho
Description	Thủ kho tiến hành nhập hàng hóa vào kho
Preconditions	Thủ kho đã đăng nhập vào hệ thống.
Postconditions	Hàng được nhập vào kho. Thông tin nhập kho được lưu trữ lại để đối soát về sau.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Nhập kho". 2. Nhập mã tài xế 3. Chọn từng đơn hàng hoặc nhiều đơn hàng. 4. Nhấn "Nhập hàng". 5. Thủ kho có thể nhấn "Xuất hóa đơn" và bàn giao hóa đơn cho Tài xế để dễ dàng đối soát về sau.

Bảng 4.14: Nhập kho

- **Xuất kho:** Sau khi đơn hàng được phân phối cho tài xế. Thì tài xế đến kho nhận hàng và tiếp tục vận chuyển. Khi xuất hàng thủ kho có trách nhiệm kiểm tra hàng hóa về số lượng và tình trạng một lần nữa trước khi bàn giao cho Tài xế.

Use case name	Xuất kho
Actor	Thủ kho
Description	Thủ kho tiến hành xuất hàng hóa để tài xế tiếp tục vận chuyển
Preconditions	Thủ kho đã đăng nhập vào hệ thống.
Postconditions	Hàng được xuất kho giao cho Tài xế đi giao. Thông tin xuất kho được lưu trữ lại để đối soát về sau.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Xuất kho". 2. Nhập mã tài xế 3. Chọn từng đơn hàng hoặc nhiều đơn hàng. 4. Nhấn "Xuất hàng". 5. Thủ kho có thể nhấn "Xuất hóa đơn" và bàn giao hóa đơn cho Tài xế để dễ dàng đối soát về sau.

Bảng 4.15: Xuất kho

- **Xem đơn hàng đang có trong kho:** Thủ kho có thể xem thông tin về các đơn hàng có trong kho mà mình đang quản lý.
- **Xem lịch sử nhập xuất kho:** Thủ kho có thể xem lịch sử nhập/xuất kho. Các thông tin bao gồm mã yêu cầu/dơn hàng, thông tin tài xế, thông tin thủ kho và thông tin chi tiết của đơn hàng.
- **Use case cho Quản lý**
 - **Xem thống kê:** Quản lý có thể xem thông tin thống kê về tình hình kinh doanh của cửa hàng nghiệp như: dòng tiền, nhân sự, số lượng đơn hàng thành công, thất bại,...
 - **Tạo tài xế:** Mỗi khi tài xế mới nhận việc làm thì quản lý có trách nhiệm tạo tài xế và chỉ định kho hoạt động cho tài xế.

Use case name	Tạo tài xế
Actor	Quản lý
Description	Quản lý tạo tài xế để tài xế hoàn thành quá trình nhận việc.
Preconditions	Quản lý đã đăng nhập vào hệ thống.
Postconditions	Tài xế được tạo thành công. Thông tin tài khoản và mật khẩu có thể được bàn giao cho tài xế.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Tạo tài xế". 2. Nhập các thông tin cần thiết như: Kho hoạt động, phương tiện, mã phương tiện, ... 3. Nhấn "Xác nhận".

Bảng 4.16: Tạo tài xế

- **Phản hồi yêu cầu hỗ trợ:** Sau khi nhận được yêu cầu hỗ trợ từ khách hàng thì quản lý có trách nhiệm phản hồi, giải đáp các yêu cầu, thắc mắc của khách hàng.

Use case name	Phản hồi yêu cầu hỗ trợ
Actor	Quản lý
Description	Quản lý có thể phản hồi yêu cầu hỗ trợ từ khách hàng.
Preconditions	Quản lý đã đăng nhập vào hệ thống.
Postconditions	Yêu cầu được phản hồi và thông báo về khách hàng.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Danh sách yêu cầu". 2. Chọn yêu cầu và nhập thông tin phản hồi. 3. Nhấn "Xác nhận".

Bảng 4.17: Phản hồi yêu cầu hỗ trợ

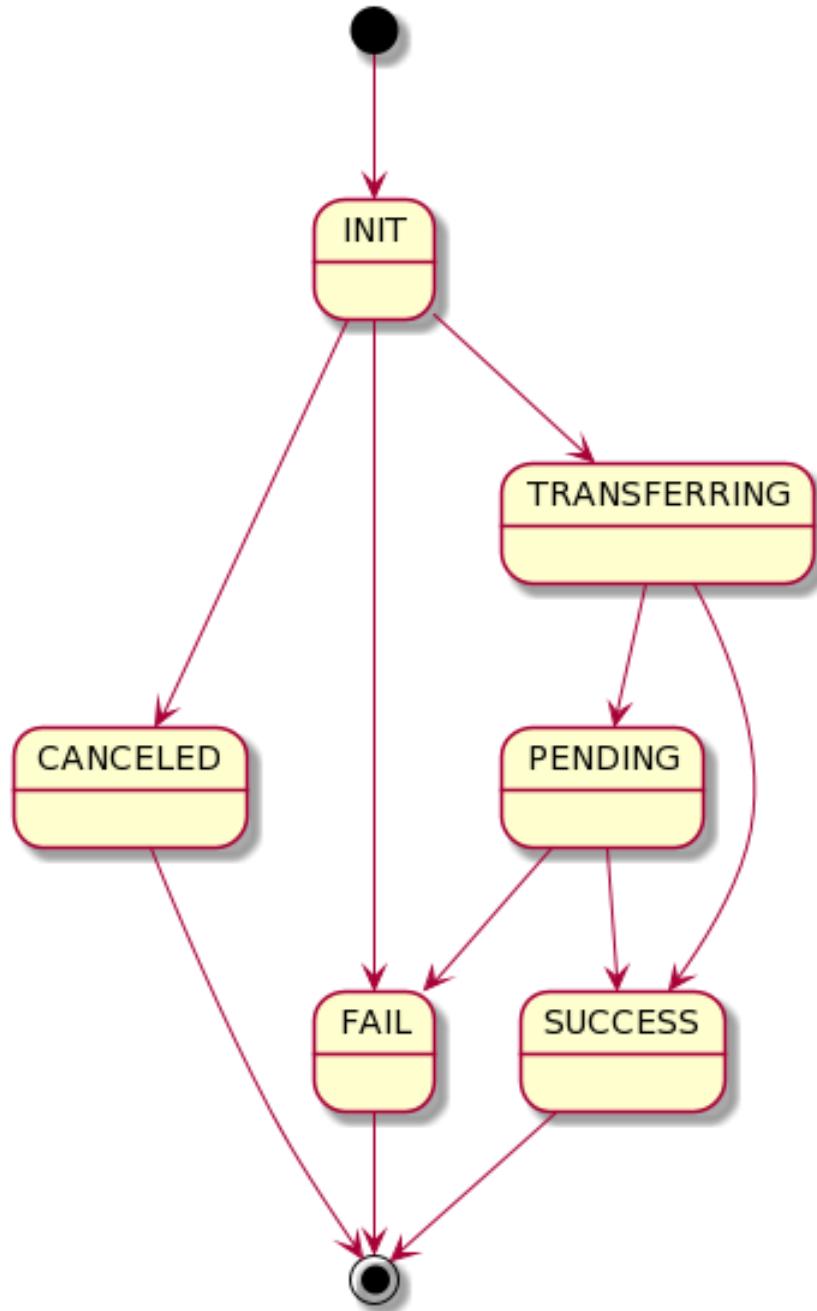
- **Xử lý sự cố:** Sau khi nhận được thông tin có sự cố từ phía Tài xế thì Quản lý có trách nhiệm giải quyết các vấn đề.

Use case name	Xử lý sự cố
Actor	Quản lý
Description	Quản lý có thể xử lý sự cố bất ngờ từ tài xế.
Preconditions	Quản lý đã đăng nhập vào hệ thống.
Postconditions	Sự cố được xử lý. Trạng thái đơn hàng được cập nhật.
Normal Flow	<ol style="list-style-type: none"> 1. Ở trang chủ nhấn "Danh sách sự cố". 2. Chọn sự cố, giải quyết và cập nhật trạng thái đơn hàng. 3. Nhấn "Xác nhận".

Bảng 4.18: Xử lý sự cố

4.2.3 Mô tả trạng thái đơn hàng, kiện hàng

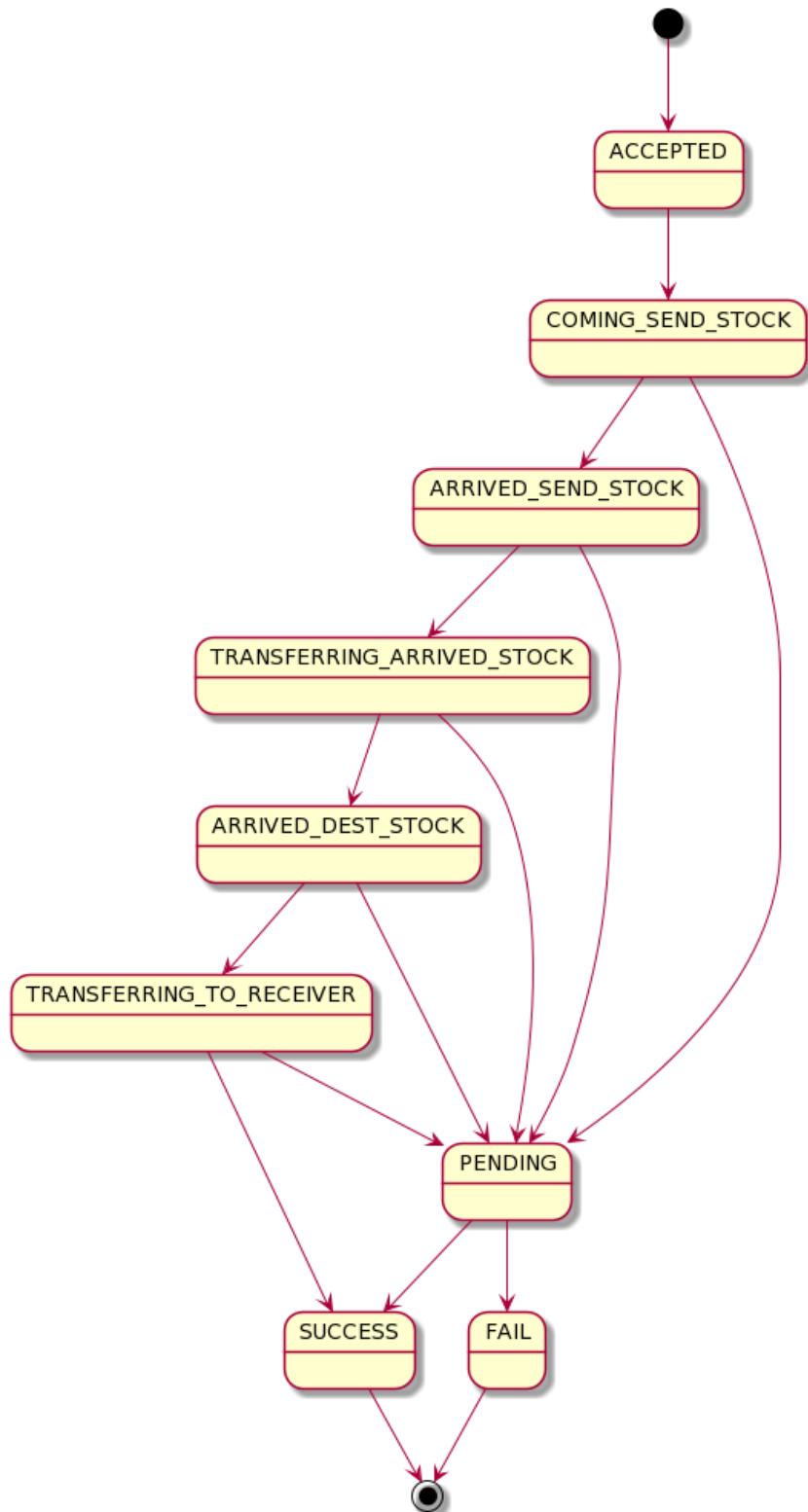
- Trạng thái đơn hàng



Hình 4.3: Trạng thái đơn hàng

1. **INIT (Khởi tạo)**: Trạng thái ban đầu của đơn hàng khi mới khởi tạo.

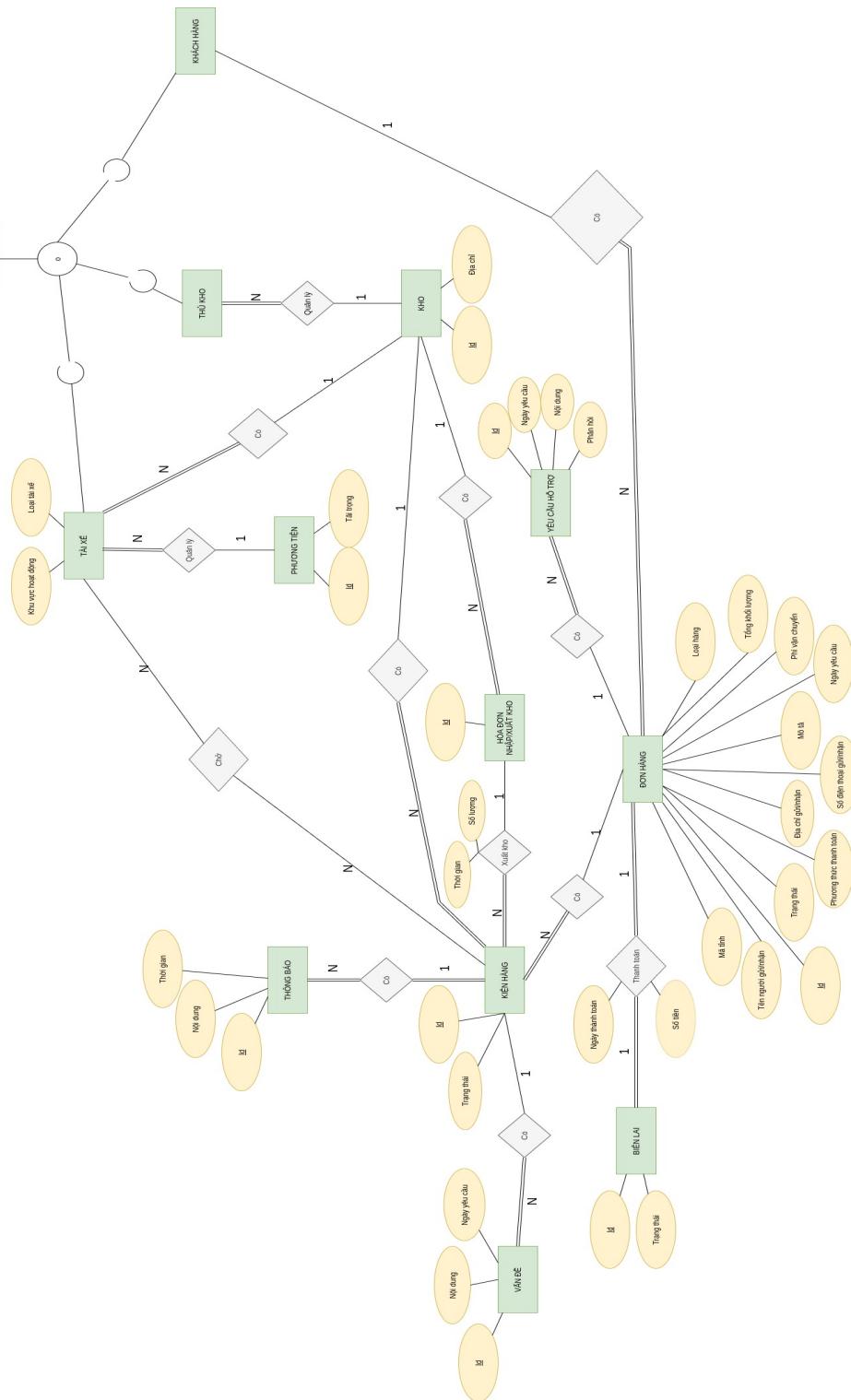
2. **CANCELED (Đã hủy)**: Nếu trạng thái đơn hàng chưa chuyển sang TRANSFERRING (đang vận chuyển) thì khách hàng vẫn có thể hủy đơn hàng.
 3. **TRANSFERRING (Đang vận chuyển)**: Đơn hàng chuyển sang trạng thái đang vận chuyển khi tài xế đang vận chuyển kiện hàng về kho nguồn (kiện hàng đầu tiên nếu đơn hàng có nhiều kiện hàng).
 4. **PENDING (Đang xử lý)**: Đơn hàng có trạng thái đang xử lý nếu tài xế báo cáo sự cố về kiện hàng trong đơn hàng đó.
 5. **FAIL (Thất bại)**: Sau khi có vấn đề được về kiện hàng hoặc đơn hàng được báo cáo từ tài xế thì quản lý có thể cập nhật đơn hàng thất bại và có chính sách xử lý theo quy định của công ty.
 6. **SUCCESS (Thành công)**: Đơn hàng chuyển sang trạng thái thành công khi tài xế đã giao kiện hàng cho người nhận (hoặc tất cả kiện hàng đã được giao trong trường hợp đơn hàng có nhiều kiện hàng).
- **Trạng thái kiện hàng**



Hình 4.4: Trạng thái kiện hàng

1. **ACCEPTED (Đã chấp nhận)**: Sau khi tài xế lấy hàng chấp nhận đơn hàng thì 1 kiện hàng mới sẽ được tạo ra và có trạng thái ban đầu là đã chấp nhận.
2. **COMING-SEND-STOCK (Đang đến kho nguồn)**: Kiện hàng có trạng thái đang đến kho nguồn sau khi tài xế đã lấy hàng và xác nhận khối lượng kiện hàng đã lấy cho hệ thống.
3. **ARRIVED-SEND-STOCK (Đã đến kho nguồn)**: Kiện hàng có trạng thái đã đến kho nguồn sau khi tài xế đã nhập hàng vào kho nguồn.
4. **TRANSFERRING-ARRIVED-STOCK (Đang đến kho đích)**: Kiện hàng có trạng thái đang đến kho đích sau khi tài xế liên tỉnh nhận hàng từ kho và bắt đầu vận chuyển liên tỉnh.
5. **ARRIVED-DEST-STOCK (Đã đến kho đích)**: Kiện hàng có trạng thái đã đến kho đích sau khi tài xế liên tỉnh đến kho và nhập hàng vào kho đích.
6. **TRANSFERRING-TO-RECEIVER (Đang vận chuyển đến người nhận)**:
Kiện hàng có trạng thái đang vận chuyển đến người nhận sau khi tài xế giao hàng đến kho lấy hàng và xuất hàng đi giao.
7. **PENDING (Đang xử lý)**: Kiện hàng có trạng thái đang xử lý nếu quản lý nhận được báo cáo về sự cố đối với kiện hàng đó. Sau khi xử lý thì quản lý có thể cập nhật trạng thái cuối của kiện hàng là Thất bại hoặc Thành công.
8. **FAIL (Thất bại)**: Trạng thái của kiện hàng thất bại khi kiện hàng xảy ra các vấn đề trong quá trình vận chuyển như mất hàng,...
9. **SUCCESS (Thành công)**: Kiện hàng có trạng thái thành công sau khi khách hàng đã nhận hàng thành công.

4.2.4 Thiết kế cơ sở dữ liệu



Hình 4.5: Lược đồ ERD

Sau khi thiết kế ERD nhóm tiến hành chuyển đổi thành các bảng tương ứng trong database như sau:

- Order

orderId	Mã đơn hàng
assignedDrivers	Lưu danh sách tài xế đang được gán để đi lấy hàng
countcompletesuborder	Đếm số lượng kiện hàng đã hoàn tất
driverIds	Danh sách các tài xế vận chuyển đơn hàng
fee	Phí vận chuyển
isDirectlyReceive	Người nhận có nhận trực tiếp hay không
isDirectlySend	Người gửi có gửi trực tiếp hay không
destStockId	Mã kho đích
detailOrder	Chi tiết đơn hàng
description	Mô tả đơn hàng
orderStatus	Trạng thái đơn hàng
orderTime	Thời gian tạo đơn hàng
paymentSide	Người thanh toán
paymentStatus	Trạng thái thanh toán
provinceCode	Mã tỉnh thành
remainWeight	Khối lượng còn lại
srcStockId	Mã kho nguồn
subOrderIds	Danh sách các kiện hàng tạo thành
totalWeight	Tổng khối lượng đơn hàng
userId	Mã khách hàng
value	Giá trị đơn hàng

Bảng 4.19: Schema Order

- SubOrder

subOrderId	Mã kiện hàng
destDriverId	Mã tài xế đi giao
destDriverName	Tên tài xế đi giao
destDriverPhone	Số điện thoại tài xế đi giao
destStockId	Mã kho đích
driverId	Mã tài xế đi lấy
driverName	Tên tài xế đi lấy
driverPhone	Số điện thoại tài xế đi lấy
externalDriverId	Mã tài xế liên tỉnh
externalDriverName	Tên tài xế liên tỉnh
externalDriverPhone	Số điện thoại tài xế liên tỉnh
issue	Nội dung vấn đề phát sinh
orderId	Mã đơn hàng
subOrderStatus	Trạng thái kiện hàng
provinceCode	Mã tỉnh thành
srcStockId	Mã kho nguồn
subOrderTime	Thời gian tạo kiện hàng
timeHistory	Danh sách thời gian update trạng thái đơn hàng
userId	Mã người dùng
weight	Khối lượng kiện hàng

Bảng 4.20: Schema SubOrder

- ImportInfo

id	Mã nhập kho
driverId	Mã tài xế nhập kho
driverName	Tên tài xế nhập kho
driverPhone	Số điện thoại tài xế nhập kho
importTime	Thời gian nhập kho
stockId	Mã kho
stockkeeperId	Mã thủ kho
stockkeeperName	Tên thủ kho
stockkeeperPhone	Số điện thoại thủ kho
subOrderIds	Danh sách kiện hàng nhập kho

Bảng 4.21: Schema ImportInfo

- **ExportInfo**

id	Mã xuất kho
driverId	Mã tài xế xuất kho
driverName	Tên tài xế xuất kho
driverPhone	Số điện thoại tài xế xuất kho
importTime	Thời gian xuất kho
stockId	Mã kho
stockkeeperId	Mã thủ kho
stockkeeperName	Tên thủ kho
stockkeeperPhone	Số điện thoại thủ kho
subOrderIds	Danh sách kiện hàng xuất kho

Bảng 4.22: Schema ExportInfo

- **SupportRequest**

id	Mã yêu cầu
userId	Mã khách hàng
orderId	Mã đơn hàng
reqTime	Thời gian yêu cầu
content	Nội dung yêu cầu
emailAddress	Địa chỉ mail của khách hàng
reply	Thông tin phản hồi

Bảng 4.23: Schema SupportRequest

- **StatisticFee**

date	Ngày tháng năm theo format yyMMdd
totalFee	Mã khách hàng

Bảng 4.24: Schema StatisticFee

- **StatisticTrans**

date	Ngày tháng năm theo format yyMMdd
transSuccess	Số đơn hàng thành công
transCancel	Số đơn hàng bị hủy
transFail	Số đơn hàng thất bại

Bảng 4.25: Schema StatisticFee

- Driver

id	Mã tài xế
area	Mã khu vực hoạt động của tài xế
status	Trạng thái hoạt động của tài xế
vehicleId	Mã phương tiện

Bảng 4.26: Schema Driver

- Role

userId	Mã user
role	Vai trò của user

Bảng 4.27: Schema Role

- Stocker

id	Mã thủ kho
areaId	Mã kho

Bảng 4.28: Schema Stocker

- User

id	Mã user
address	Địa chỉ
email	Email
password	Mật khẩu
phone	Số điện thoại
provinceCode	Mã tỉnh
username	Tên tài khoản

Bảng 4.29: Schema User

- Vehicle

id	Mã phương tiện
currentWeight	Trọng tải hiện tại
totalWeight	Trọng tải
number	Biển số

Bảng 4.30: Schema Vehicle

- **Warehouse**

id	Mã kho
address	Địa chỉ kho
areaCode	Mã khu vực

Bảng 4.31: Schema Warehouse

5

Hiện thực hệ thống

5.1 Công nghệ sử dụng

Để xây dựng một hệ thống lớn, phức tạp, khả năng chịu tải cao và dễ mở rộng như vậy thì nhóm đã sử dụng rất nhiều thư viện cả phía Frontend lẫn Backend. Sau đây là một số công nghệ nổi bật.

Thư viện, framework	Phiên bản	Chức năng
SpringBoot	2.2.6	Giúp tạo service nhanh chóng và dễ dàng
guava	20.0	Cung cấp các hàm để xử lý Collections
data-jpa	2.2.6	Cung cấp các API để làm việc với database
redisson	3.12.3	Cung cấp API để service tương tác với Redis server
mysql-connector-java	2.2.6	Giúp kết nối service đến database Mysql
gson	2.8.2	Cung cấp các hàm để làm việc với JSON object
spring-kafka	2.2.6	Cung cấp API để tương tác với Kafka brokers
httpclient	4.5.12	Hỗ trợ việc gửi HTTP Request để giao tiếp với service khác
log4j2	2.2.6	Hỗ trợ cơ chế ghi log
data-cassandra	2.2.6	Giúp kết nối service đến database Cassandra
activemq	2.2.6	Cung cấp API để tương tác với ActiveMQ server
spring-boot-starter-mail	2.2.6	Giúp hiện thực chức năng gửi mail
chart.js	2.9.4	Dùng để hiện thực chức năng tạo biểu đồ thống kê của admin.
date-fns	2.21.1	Dùng để thực hiện việc tính toán ngày tháng như lấy thời gian bắt đầu / kết thúc của ngày, tăng giảm số ngày vv...
antd	4.9.4	Thư viện CSS và component tương thích với React.
json-bigint	1.0.0	Giúp parse ra được ID lớn của đơn hàng thành chuỗi dạng string.
@react-pdf/renderer	2.0.8	Giúp tạo ra template cho phiếu nhập/xuất kho dưới dạng pdf.
gzipper	4.5.0	Giúp tạo ra những file gzip của những file static đã build ra. Như vậy sẽ giảm dung lượng đường truyền mạng cần thiết để tải file và trang web sẽ hiển thị nhanh hơn vào lần đầu tiên truy cập.

Bảng 5.1: Công nghệ sử dụng

5.2 Quản lý mã nguồn

Để các thành viên trong nhóm có thể cùng nhau phát triển mà không gặp xung đột gì thì cả nhóm sử dụng Git để quản lý mã nguồn.

Git là hệ thống kiểm soát phiên bản phân tán mà nguồn mở. Các dự án thực tế thường có nhiều nhà phát triển làm việc song song. Vì vậy, một hệ thống kiểm soát phiên bản

như Git là cần thiết để đảm bảo không có xung đột mã giữa các nhà phát triển. Ngoài ra, các yêu cầu trong dự án thay đổi thường xuyên. Vì vậy, cần một hệ thống cho phép nhà phát triển quay lại phiên bản cũ hơn của mã nguồn.



Hình 5.1: Giới thiệu về Gitlab

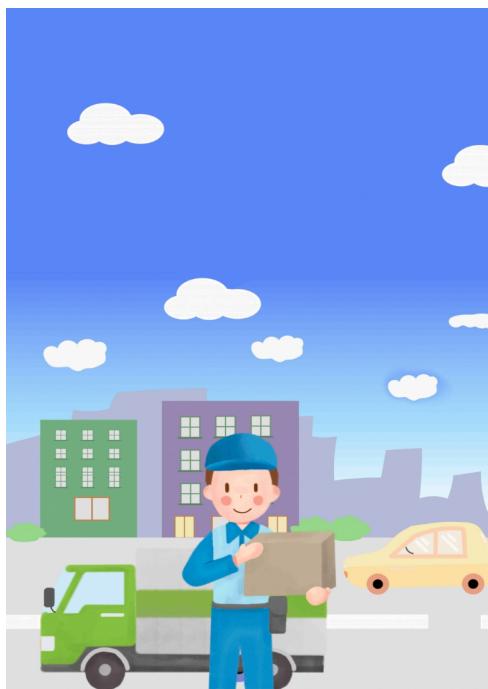
Về việc lưu trữ mã nguồn thì nhóm sử dụng **Gitlab**. GitLab khá nổi tiếng và là một mã nguồn mở của máy chủ Git được thực hiện bởi hơn 50.000 tổ chức. Trong vài năm gần đây Gitlab đã phát triển mạnh mẽ với sự hỗ trợ của cộng đồng mạng, hàng nghìn người sử dụng trên một máy chủ duy nhất hoặc một số máy chủ hoạt động tương tự.

5.3 Kết quả hiện thực

Sau quá trình dài thực hiện luận văn, sử dụng những công nghệ và kiến thức được đưa ra trong các phần trước, nhóm đã hoàn thành một hệ thống vận chuyển hàng hóa liên tỉnh tương đối hoàn chỉnh.

5.3.1 Chức năng đăng nhập, đăng kí

- Đăng kí



Tạo tài khoản

Email: khoavnganh@gmail.com

Số điện thoại: 0944744802

Tên tài khoản: Vương Anh Khoa

Địa chỉ: 5 Đồng Đa, phường 2, Bình Thạnh, Thành phố Hồ Chí Minh, Việt Nam

Mật khẩu: *****

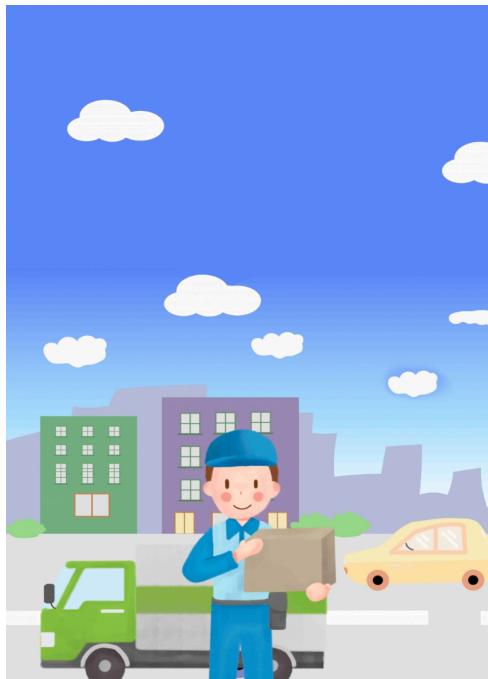
Nhập lại mật khẩu: *****

Đăng ký

Bạn đã có tài khoản? [Đăng nhập ngay](#)

Hình 5.2: Giao diện đăng kí dành cho người muốn gửi đơn hàng

- Đăng nhập



Đăng nhập

Tài khoản:

Mật khẩu:

Đăng nhập

Bạn chưa có tài khoản [Đăng ký ngay](#)

Hình 5.3: Giao diện đăng nhập dành cho actor của hệ thống

5.3.2 Chức năng dành cho người gửi

- Lên đơn hàng

Bên gửi

Số điện thoại: 0944744802
Họ tên: khoa vuong
Địa chỉ: 5 Đồng Đa, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam
Email: khoavnganh@gmail.com

Gửi hàng trực tiếp tại kho gửi
 Nhận hàng trực tiếp tại kho đến

Bên nhận

* Số điện thoại:
* Họ tên:
* Email:
* Địa chỉ:
* Tỉnh thành:

Hàng hóa

* Loại hàng:
* Khối lượng ước tính (kg):
* Giá trị hàng hóa (đ):

Tùy chọn thanh toán
Chọn bên trả phí
Ghi chú
Tạo đơn

Hình 5.4: Giao diện điền thông tin để lên đơn hàng

Xác nhận tạo đơn hàng

| Bên nhận
Số điện thoại: 0913154708
Họ tên: Tan Vuong
Email: dangvandung0812@gmail.com
Địa chỉ: 54A Nguyễn Chí Thanh, Láng Thượng, Đống Đa, Hà Nội, Việt Nam

| Hàng hóa
Loại hàng: Linh kiện điện tử
Khối lượng ước tính (kg): 3,500
Giá trị ước tính (vnđ): 15,000,000

| Ghi chú
Hàng dễ vỡ, xin nhẹ tay

| Tuỳ chọn thanh toán
Bên gửi trả phí

| Cách thức vận chuyển hàng
Chiều gửi: Xe giao vận đến lấy hàng vervo
Chiều nhận: Xe giao vận đi gửi hàng

Tổng phí
44,990,086 VND

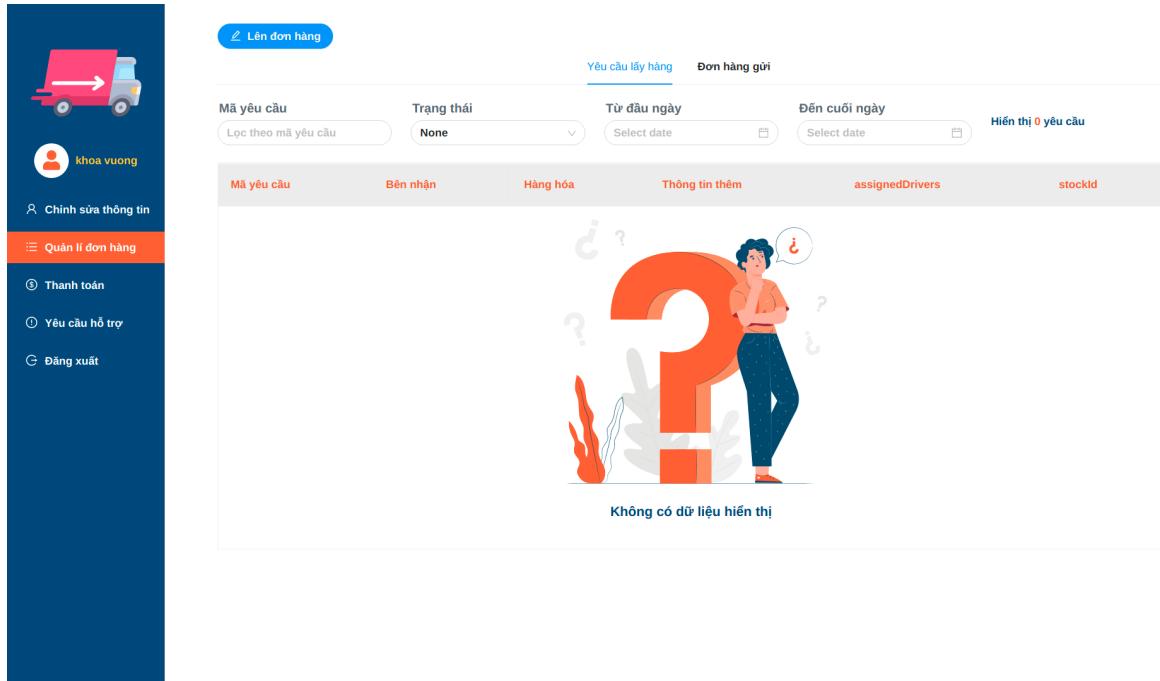
* Khối lượng
* Giá trị

Cancel Xác nhận

Hình 5.5: Thông tin xác nhận đơn hàng muốn tạo

Khung nhập địa chỉ hỗ trợ gợi ý địa chỉ nhờ sử dụng Google Place Autocomplete API. Khi ấn nút tạo đơn sẽ có màn hình chờ để phía server trả về phí vận chuyển.

- Xem danh sách đơn hàng



Hình 5.6: Giao diện list yêu cầu của người gửi (Rỗng)

Mã yêu cầu	Bên nhận	Hàng hóa	Thông tin thêm
№ 2105301648783759 5/30/2021, 11:00:57 AM Hủy yêu cầu	⋮ Tam Vuong ✉ 0913154708 ✉ dangvandung0812@gmail.com ⌚ 54A Nguyễn Chí Thanh, Láng Thượng, Đống Đa, Hà Nội, Việt Nam	⌚ Máy tính ⌚ 2,700 kg ⌚ 34.703.699 VND	Trạng thái: Đang được tiếp nhận Giá trị hàng hóa: 6,500,000 VND Ghi chú: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua Bên gửi trả phí Khối lượng chưa lấy: 2,700kg
№ 2105301956406486 5/30/2021, 11:00:00 AM Hủy yêu cầu	⋮ Hanh Thuy ✉ 0918032337 ✉ hanhthuy@gmail.com ⌚ 1 Đại Cồ Việt, Bách Khoa, Hai Bà Trưng, Hà Nội, Việt Nam	⌚ Xi măng ⌚ 2,700 kg ⌚ 34.579.221 VND	Trạng thái: Đang được tiếp nhận Giá trị hàng hóa: 1,534,896 VND Ghi chú: Bên nhận trả phí Khối lượng chưa lấy: 2,700kg Gửi hàng trực tiếp tại kho: 1358560893985047192 Nhận hàng trực tiếp tại kho: 3028981646957560464

Hình 5.7: Giao diện list yêu cầu của người gửi (Có yêu cầu)

- Xem danh sách các kiện hàng

Mã yêu cầu	Mã đơn	Trạng thái	Từ đầu ngày	Đến cuối ngày
Mã yêu cầu: 210530363294524 Mã đơn: 2105301755234006 Ngày tạo: 5/30/2021, 12:33:05 PM	⋮ Tam Vuong ✉ 0913154708 ✉ dangvandung0812@gmail.com ⌚ 54A Nguyễn Chí Thanh, Láng Thượng, Đống Đa, Hà Nội, Việt Nam	⌚ Linh kiện điện tử ⌚ 1000kg / 1500kg	⌚ Đang đến kho gửi	
Mã yêu cầu: 210530363294524 Mã đơn: 210530822789641 Ngày tạo: 5/30/2021, 12:33:23 PM	⋮ Tam Vuong ✉ 0913154708 ✉ dangvandung0812@gmail.com ⌚ 54A Nguyễn Chí Thanh, Láng Thượng, Đống Đa, Hà Nội, Việt Nam	⌚ Linh kiện điện tử ⌚ 500kg / 1500kg	⌚ Đang đến kho gửi	

Hình 5.8: Giao diện xem danh sách các đơn hàng nhỏ được tách từ yêu cầu

- Chỉnh sửa thông tin của người gửi

Chỉnh sửa thông tin tài khoản

ID
861922297081251169

Email
khoavanganh@gmail.com

Số điện thoại
0944744802

Tên tài khoản
khoa vuong

Địa chỉ
5 Đồng Đa, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam

Mật khẩu

Đổi mật khẩu

Cập nhật

Hình 5.9: Giao diện chỉnh sửa thông tin của người gửi

- Thanh toán

1 Chọn yêu cầu muốn thanh toán

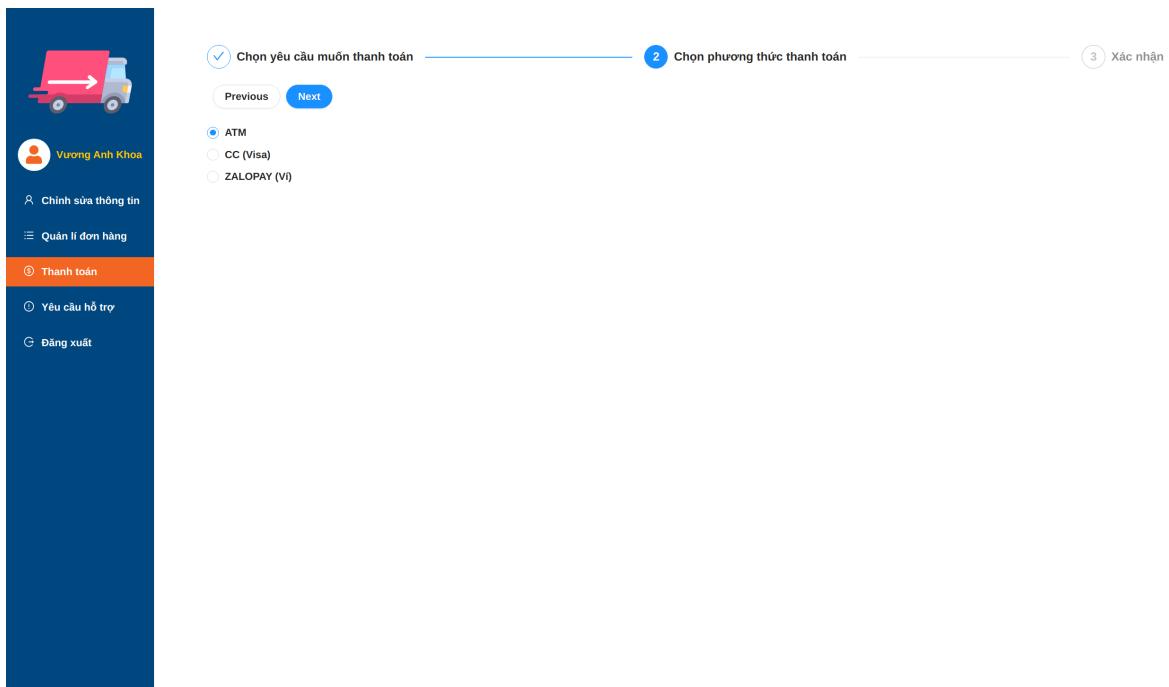
Next

Mã yêu cầu	Bên nhận	Hàng hóa	Thông tin thêm
№ 2107211030802357	Tuan Vuong 0944744802	Computer 1,500 kg	Gia trị hàng hóa ước tính: 1,000,000 Ghi chú: lorem ipsum
7/21/2021, 10:08:47 PM	khoavanganh@gmail.com 1 Đại Cồ Việt, Bách Khoa, Hai Bà Trưng, Hà Nội, Việt Nam	20.483.967 VND	

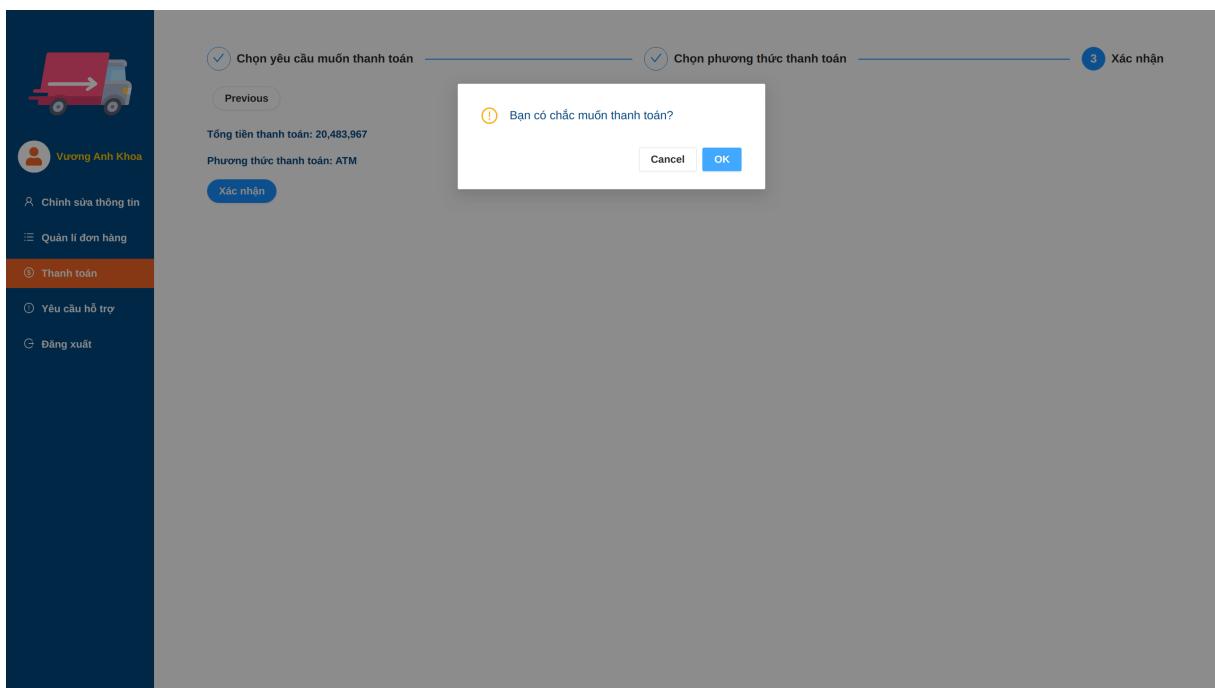
2 Chọn phương thức thanh toán

3 Xác nhận

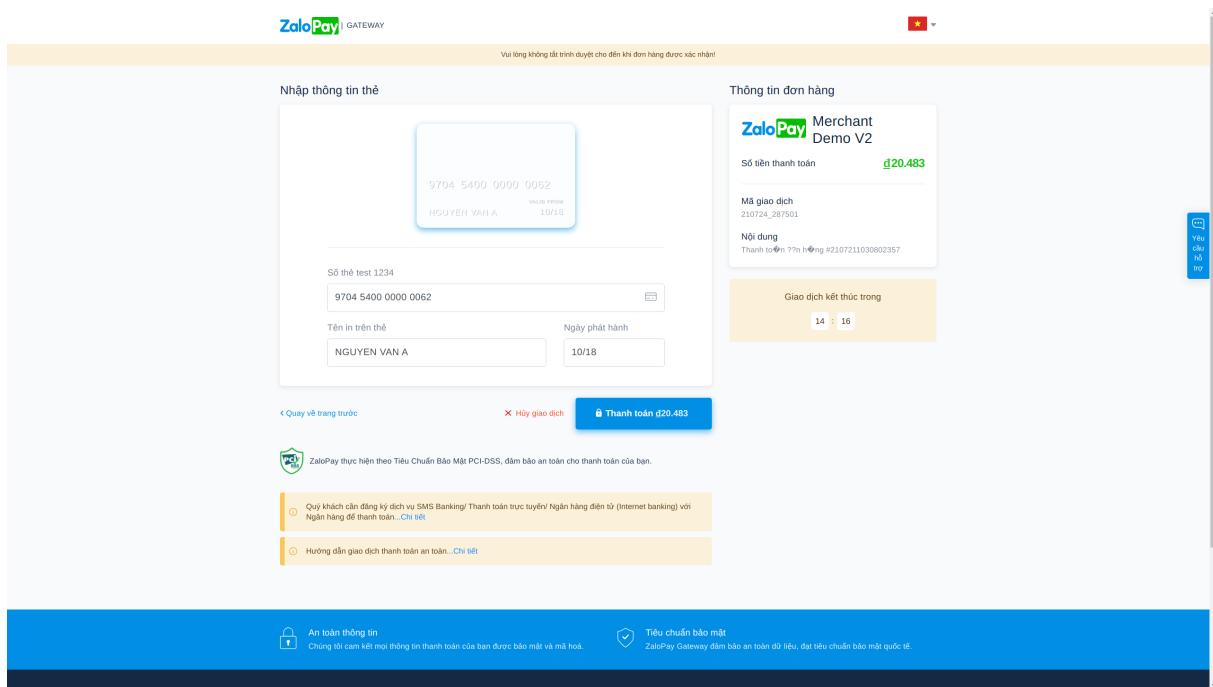
Hình 5.10: Giao diện chọn những đơn hàng đã giao cần thanh toán



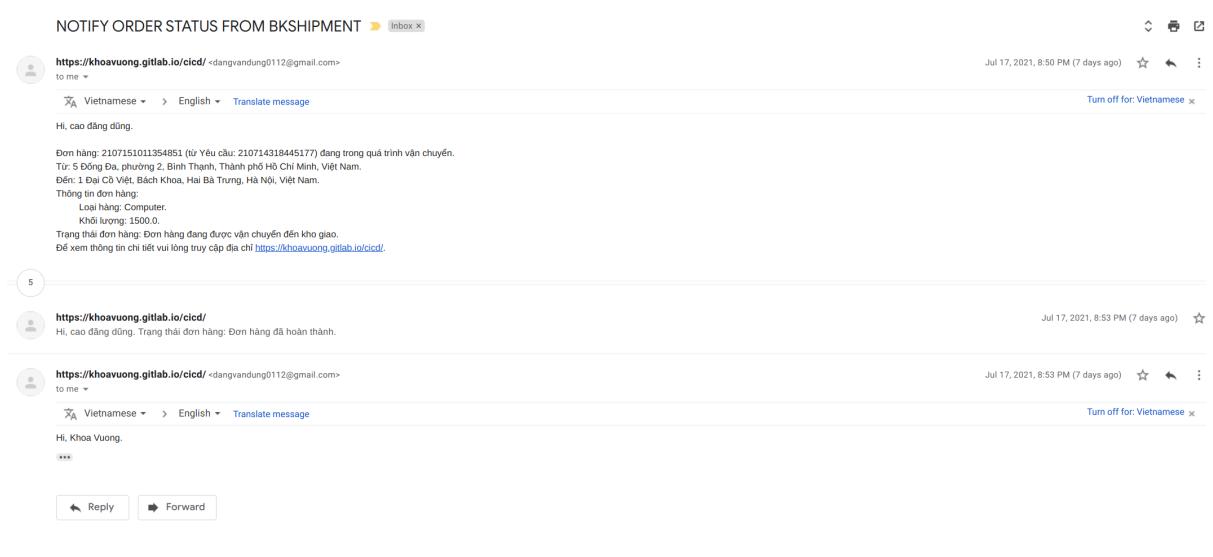
Hình 5.11: Giao diện chọn phương thức thanh toán



Hình 5.12: Giao diện xác nhận lại tổng số tiền và phương thức thanh toán



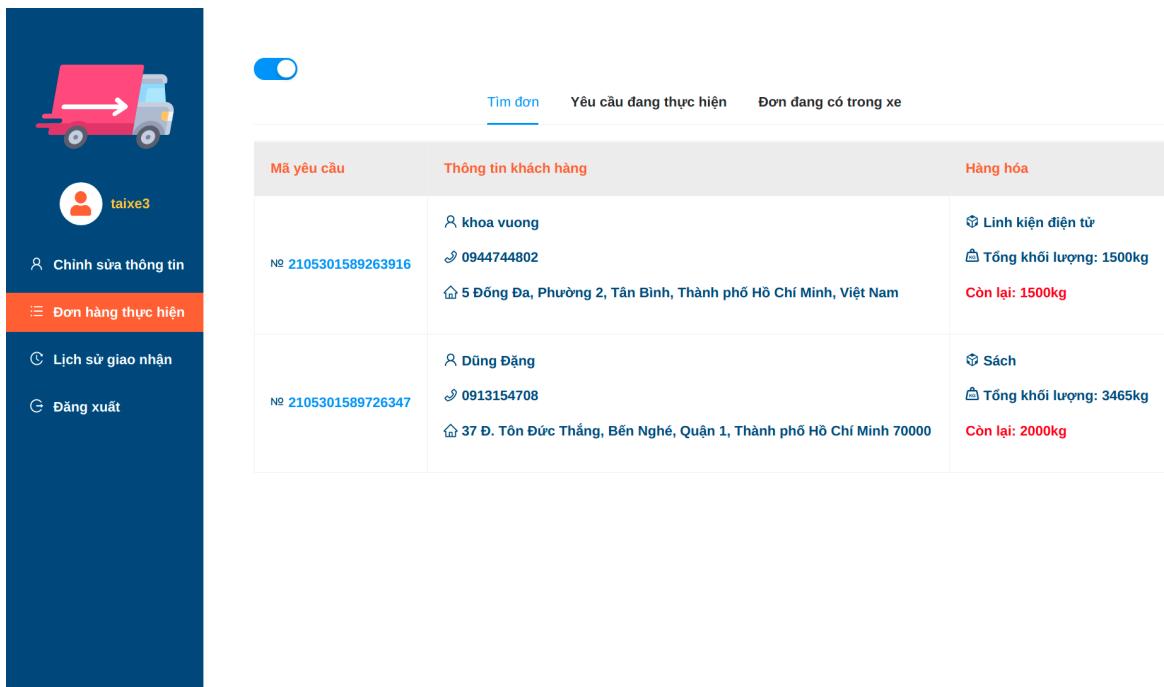
Hình 5.13: Giao diện hệ thống thanh toán trung gian ZaloPay



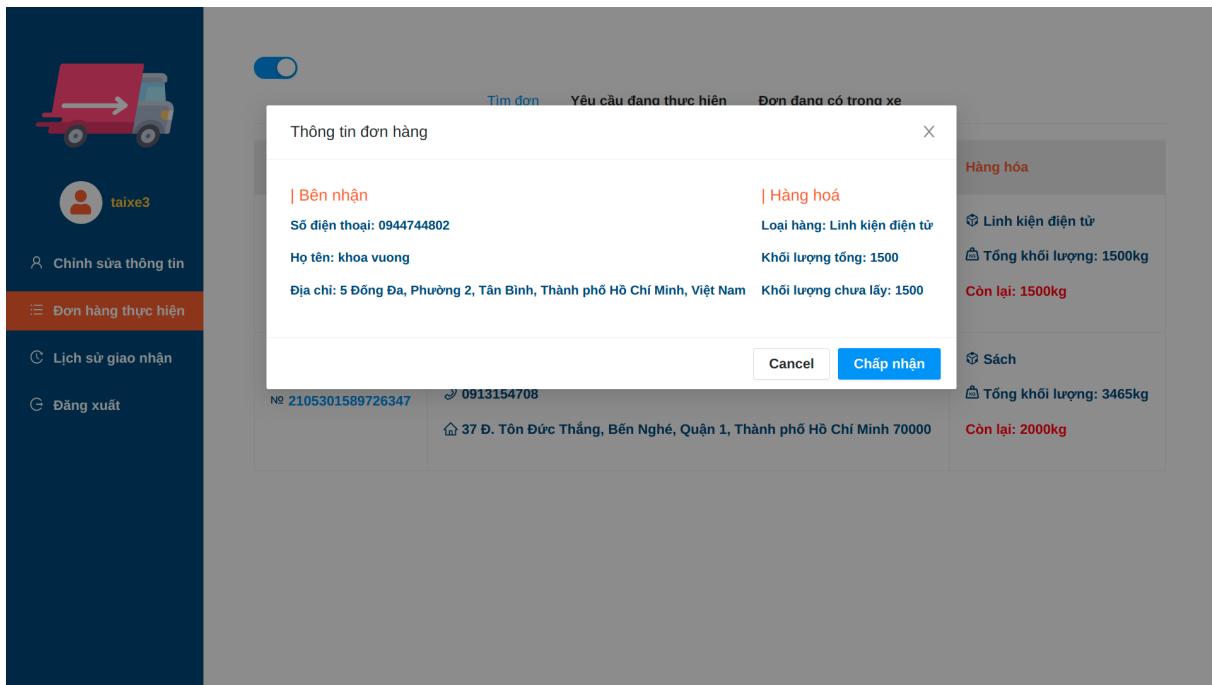
Hình 5.14: Mail thông báo tình trạng đơn hàng của hệ thống

5.3.3 Chức năng dành cho tài xế

- Tìm những yêu cầu được gán

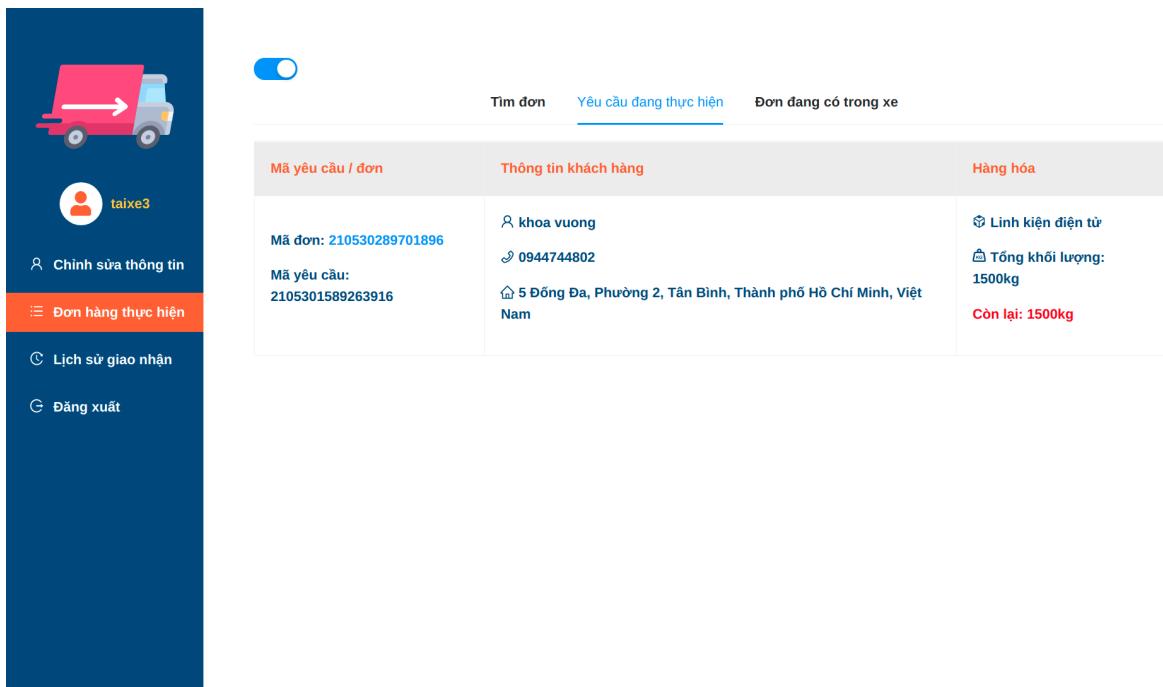


Hình 5.15: Giao diện tìm những yêu cầu được gán

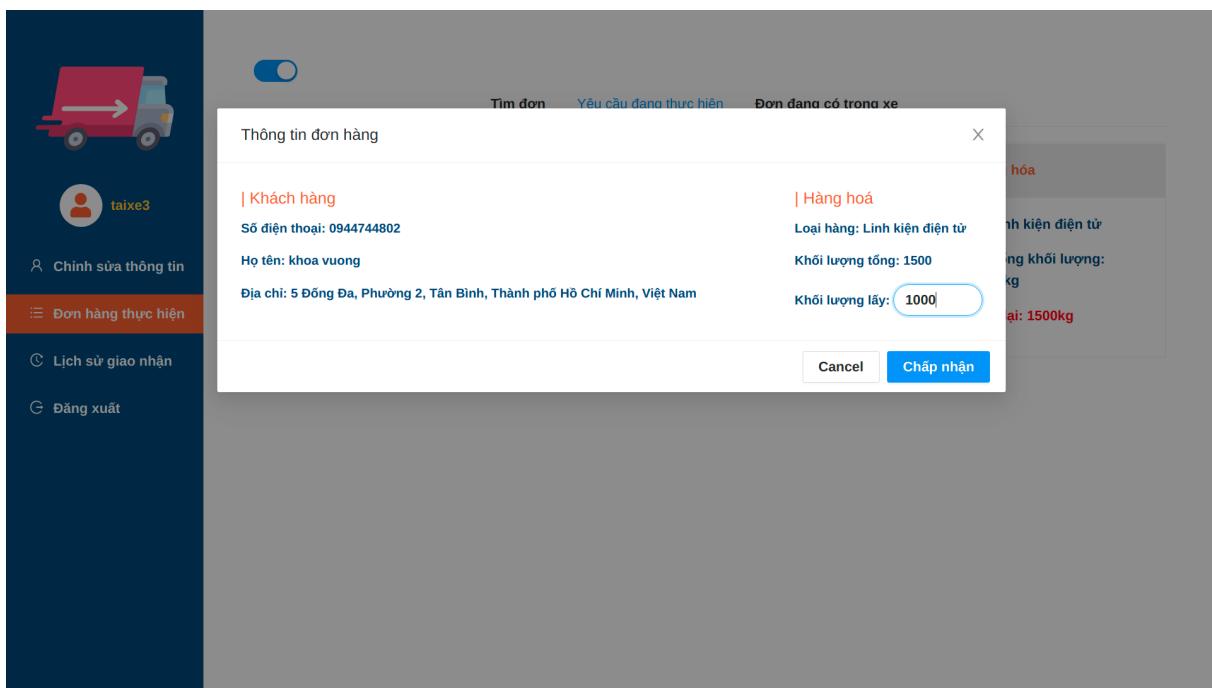


Hình 5.16: Giao diện xác nhận lại thông tin yêu cầu muốn nhận

- Xem đơn hàng đang đến lấy

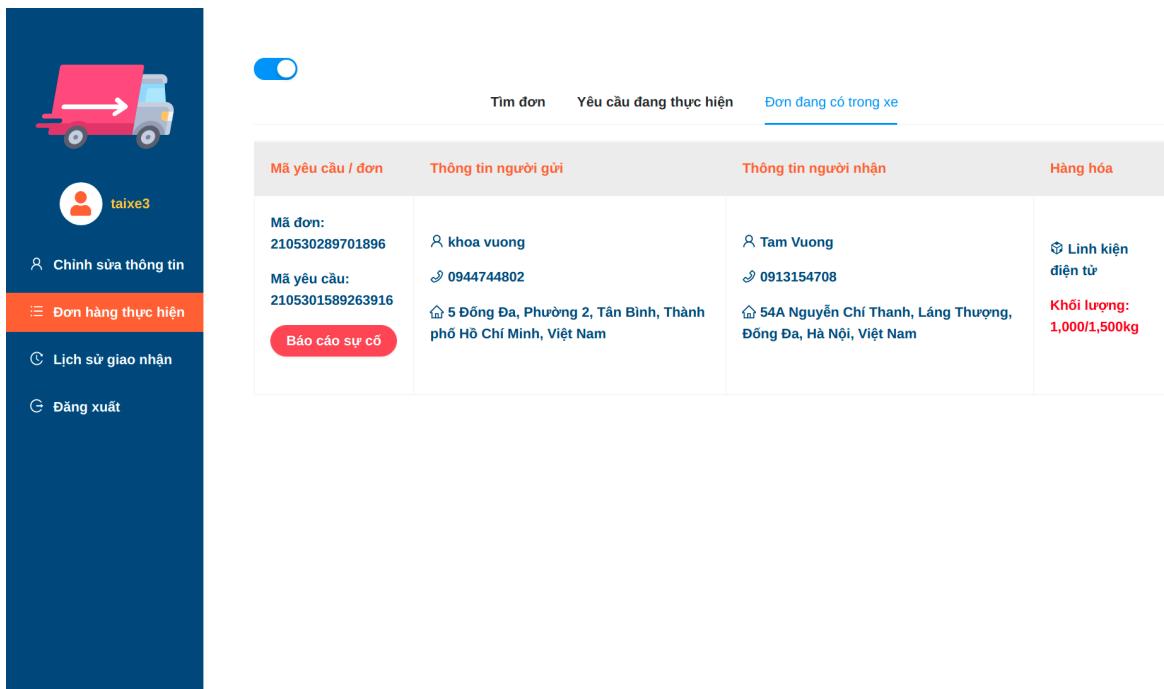


Hình 5.17: Giao diện xem thông tin đơn hàng đang đến lấy



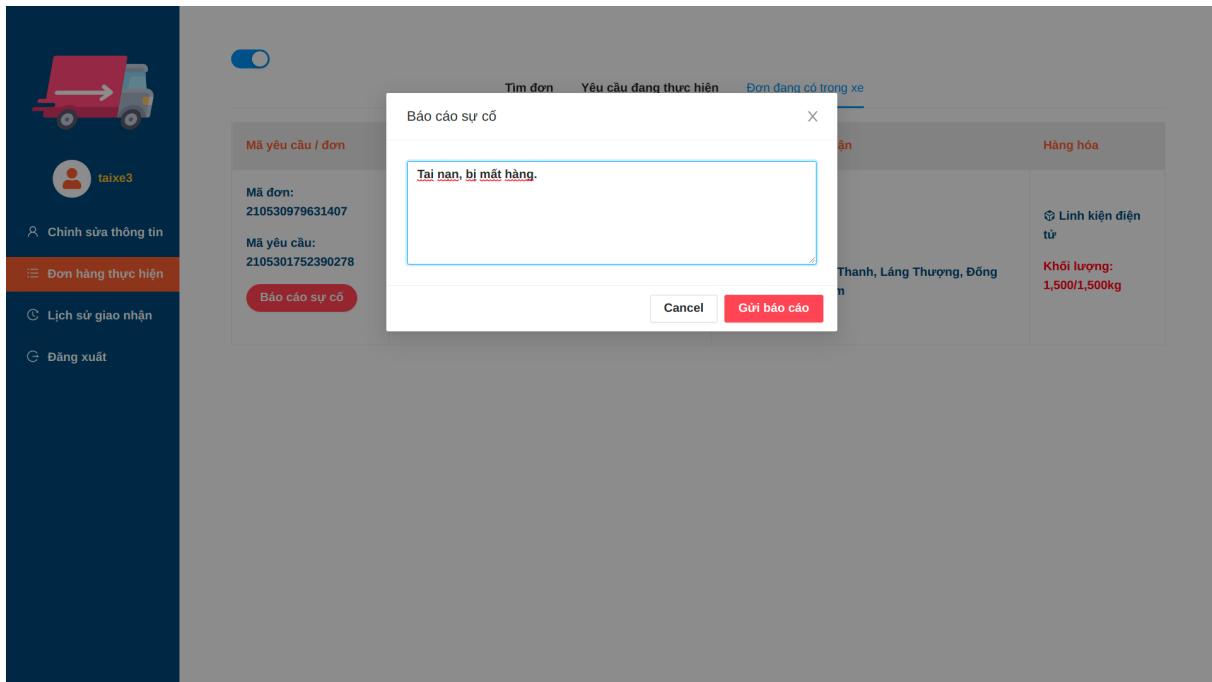
Hình 5.18: Giao diện xác nhận lại thông tin đơn hàng đang muôn nhận

- Xem danh sách các kiện hàng trong xe



Hình 5.19: Giao diện xem thông tin các đơn hàng đang ở trong xe

- **Báo cáo vấn đề**



Hình 5.20: Giao diện báo cáo vấn đề ghi gấp sự cố

- **Xem lịch sử giao nhận hàng**

ID	Người gửi	Người nhận	Hàng hóa
Mã yêu cầu: 2105301593006216	Khoa Vuong 0944744902	Hanh Thuy 0918032337	Xe măng 3500kg
Mã đơn: 2105301972414591	khoavanganh123@gmail.com	dangvandung0812@gmail.com	
5/30/2021, 4:06:17 PM	7 Đồng Đa, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	1 Đại Cồ Việt, Bách Khoa, Hai Bà Trưng, Hà Nội, Việt Nam	
Mã yêu cầu: 2105301489060381	khoa vuong 0944744802	Tam Vuong 0913154708	Linh kiện điện tử
Mã đơn: 210530269627124	khoavanganh@gmail.com	dangvandung0812@gmail.com	
5/30/2021, 4:01:20 PM	5 Đồng Đa, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	54A Nguyễn Chí Thanh, Láng Thượng, Đống Đa, Hà Nội, Việt Nam	1500kg

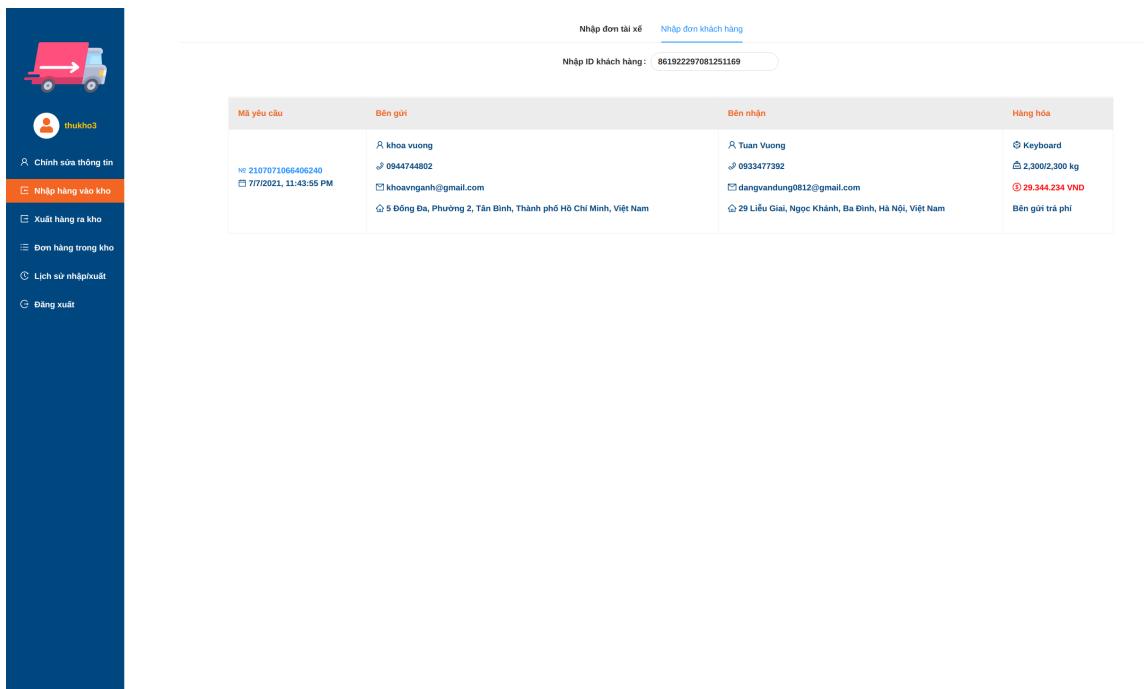
Hình 5.21: Giao diện xem lịch sử giao nhận hàng

5.3.4 Chức năng dành cho thủ kho

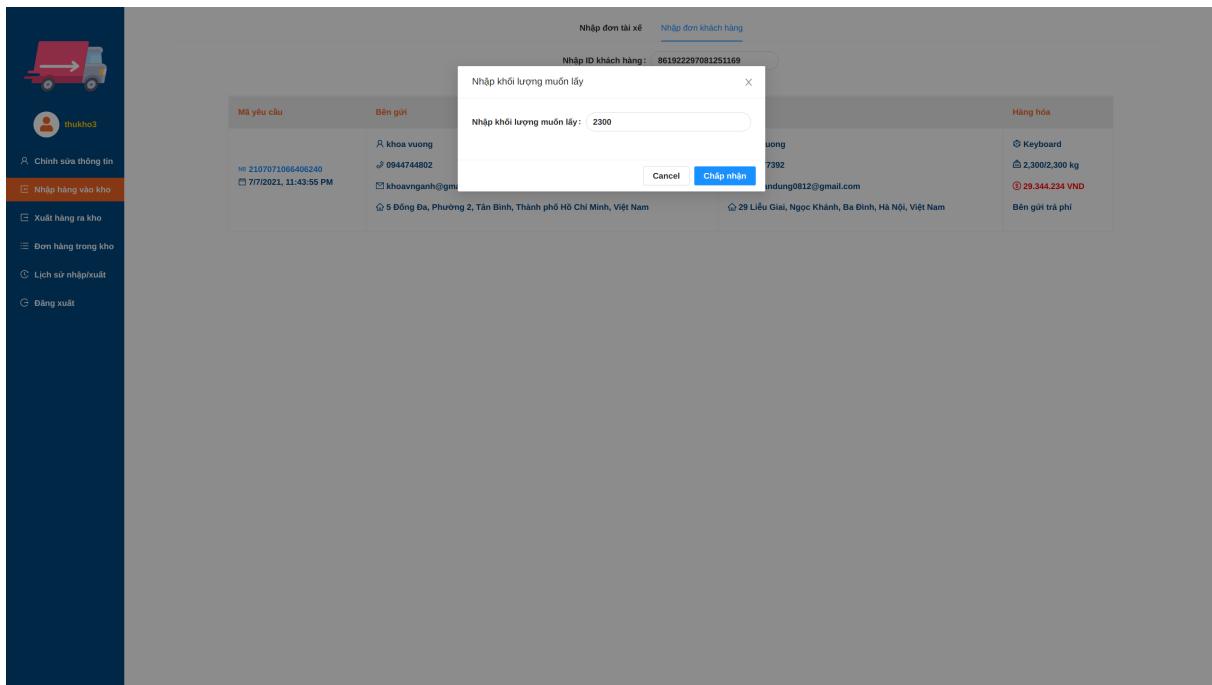
- Nhập kho

Mã yêu cầu / đơn	Thông tin người gửi	Thông tin người nhận	Hàng hóa
Mã đơn: 210530269627124	khoa vuong 0944744802	Tam Vuong 0913154708	Linh kiện điện tử
Mã yêu cầu: 2105301489060381	khoavanganh@gmail.com	dangvandung0812@gmail.com	
	5 Đồng Đa, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	54A Nguyễn Chí Thanh, Láng Thượng, Đống Đa, Hà Nội, Việt Nam	Khối lượng: 1500kg/1500kg
Mã đơn: 2105301972414591	Khoa Vuong 0944744902	Hanh Thuy 0918032337	Xe măng
Mã yêu cầu: 2105301593006216	khoavanganh123@gmail.com	dangvandung0812@gmail.com	
	7 Đồng Đa, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	1 Đại Cồ Việt, Bách Khoa, Hai Bà Trưng, Hà Nội, Việt Nam	Khối lượng: 3500kg/3500kg

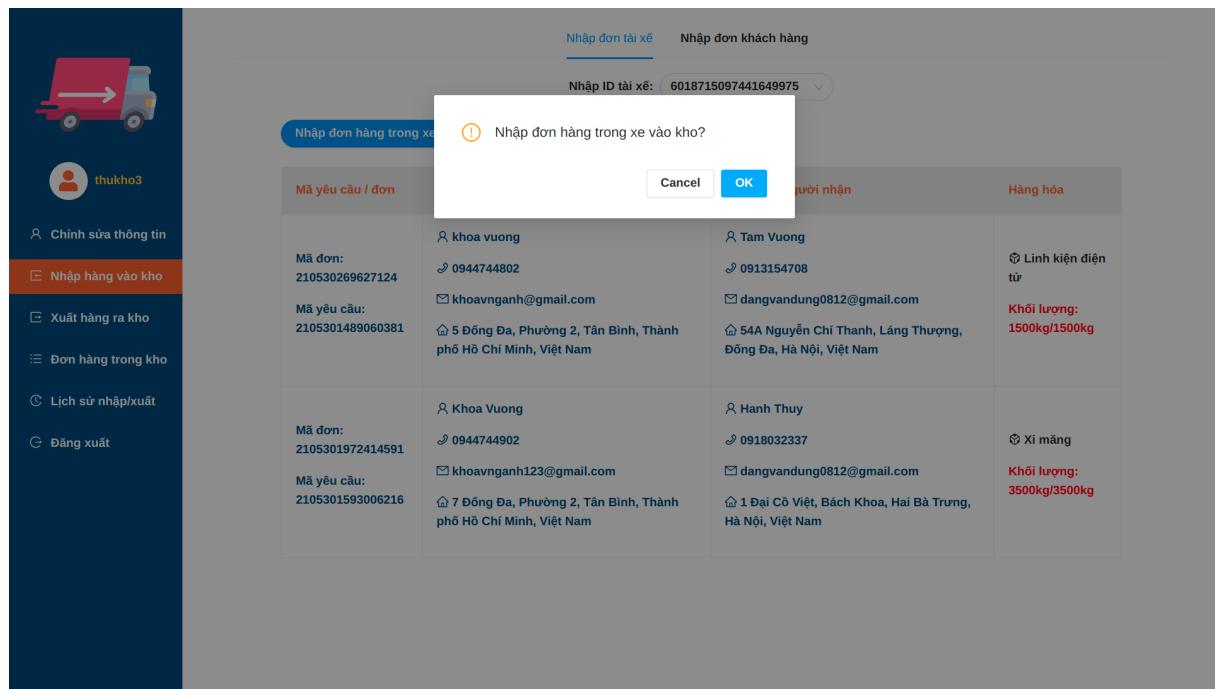
Hình 5.22: Giao diện nhập ID của tài xế để nhập các đơn hàng trong xe tải xế vào kho



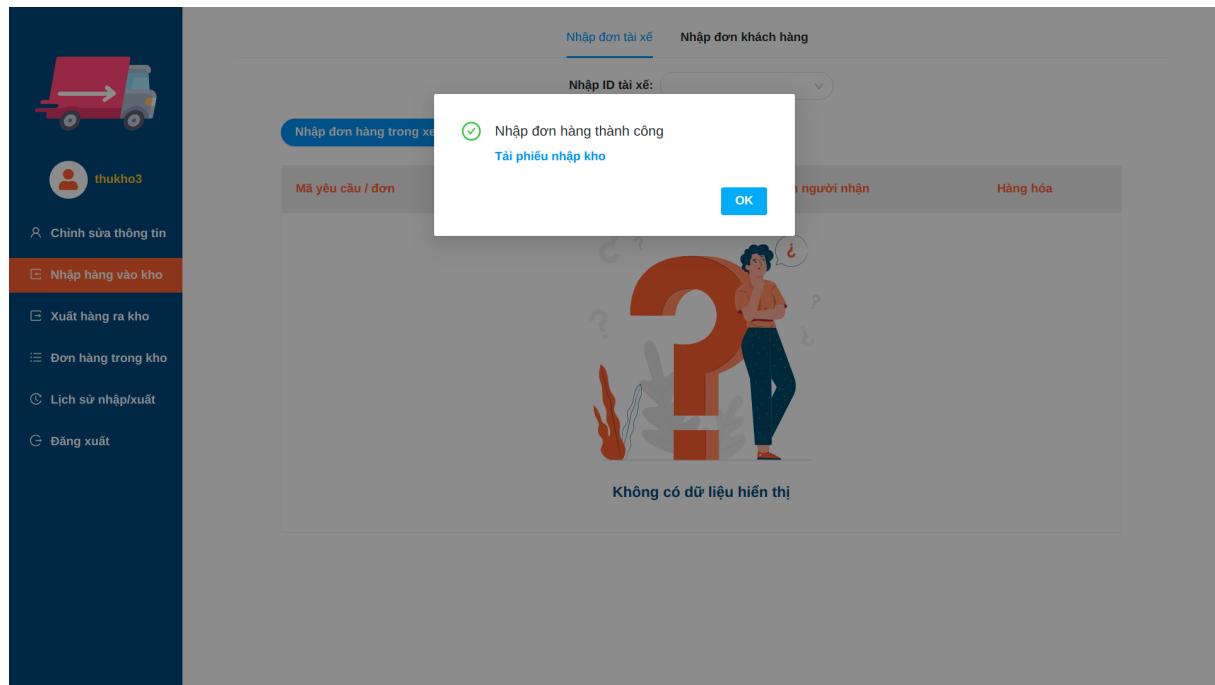
Hình 5.23: Nhập ID của khách hàng để nhập đơn vào kho



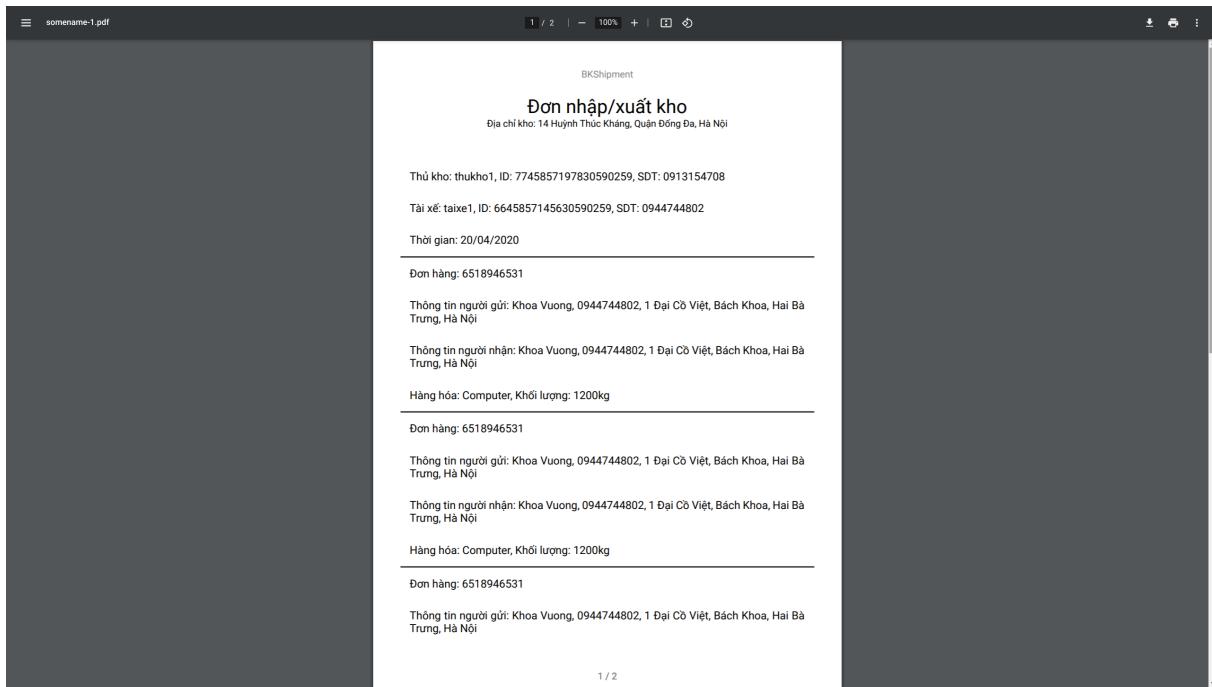
Hình 5.24: Cho phép nhập khối lượng mà khách muôn nhập vào kho



Hình 5.25: Xác nhận nhập hàng trong xe tài xế vào kho



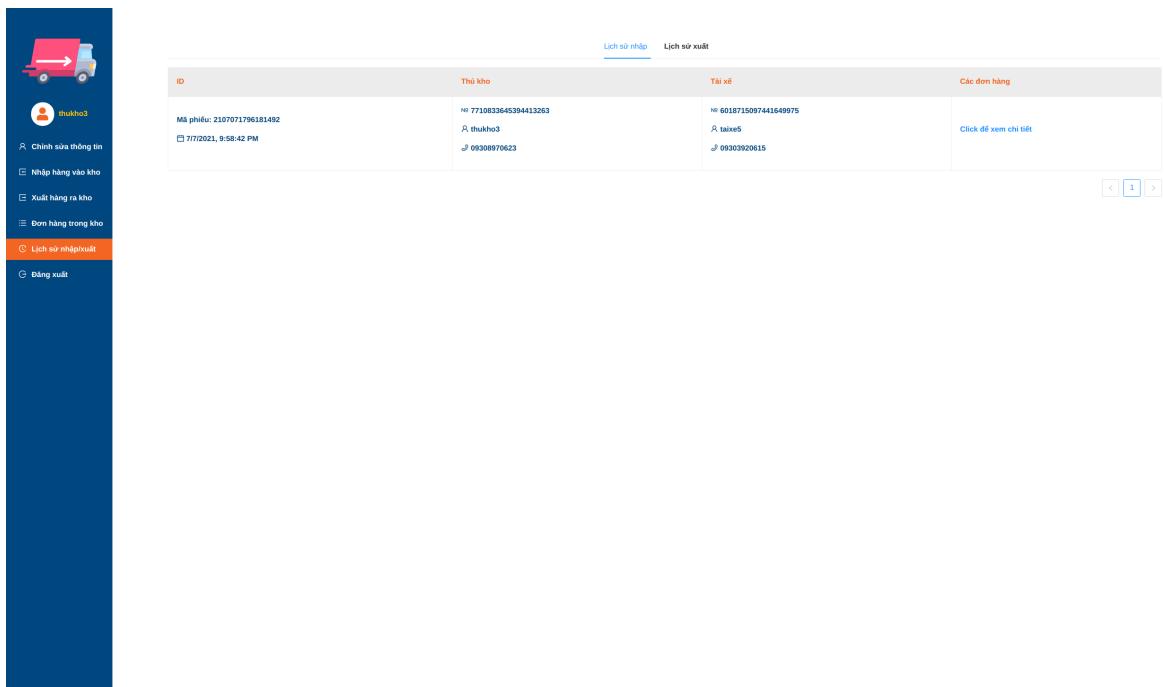
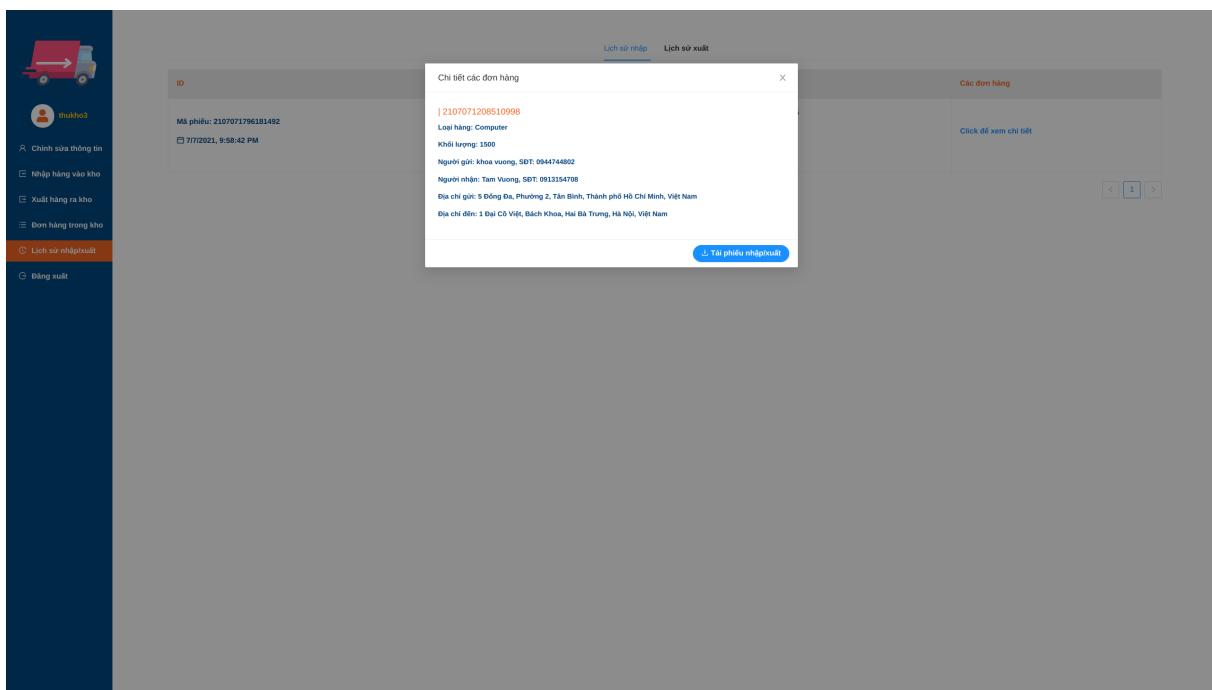
Hình 5.26: Nhập hàng thành công và cho thủ kho có thẻ tài phiếu nhập kho



Hình 5.27: Format mẫu của 1 đơn nhập/xuất kho

- Xem danh sách đơn hàng

Hình 5.28: Danh sách các đơn hàng trong kho

**Hình 5.29:** Lịch sử nhập/xuất kho**Hình 5.30:** Thông tin chi tiết của một lần nhập/xuất đơn hàng (Cho phép tải pdf về)

Chức năng xuất kho có giao diện và tính năng tương tự lúc nhập kho.

5.3.5 Chức năng dành cho Quản lý

- Quản lý tài xế

Tài xế	Phương tiện	Kho hoạt động
№ 110003658692643575 Ⓜ taixe_intern_hn_7 ⓐ taixe_intern_hn_7@gmail.com Role: Nội tỉnh	№ 4010125043573084090 Ⓛ 0 / 5000	№ 1181267876917690786 Ⓛ 321 Đường Ngọc Hồi, thị trấn Văn Điền, Thanh Trì, Hà Nội, Việt Nam
№ 1649294361521086575 Ⓜ taixe_extern_hn_10 ⓐ taixe_extern_hn_10@gmail.com Role: Liên tỉnh	№ 7092894266940472365 Ⓛ 0 / 40000	№ 1181267876917690786 Ⓛ 321 Đường Ngọc Hồi, thị trấn Văn Điền, Thanh Trì, Hà Nội, Việt Nam
№ 219062391252470889 Ⓜ taixe_intern_hcm_14 ⓐ taixe_intern_hcm_14@gmail.com Role: Nội tỉnh	№ 8112675337965487516 Ⓛ 0 / 7000	№ 3359064593015587782 Ⓛ 291 Nguyễn Sơn, Phú Thạnh, quận Tân Phú, Thành phố Hồ Chí Minh, Việt Nam

Hình 5.31: Xem danh sách các tài xế trong hệ thống

- Quản lý thủ kho

Thủ kho	Kho hoạt động
№ 616194410572172711 A. thukho_hn_2 ✉ thukho_hn_2@gmail.com	№ 191598677987957556 ⌚ 4b Đèn Lử 2, Hoàng Văn Thụ, quận Hoàng Mai, Hà Nội, Việt Nam
№ 906828411704329174 A. thukho_hn_1 ✉ thukho_hn_1@gmail.com	№ 1181267876917690786 ⌚ 321 Đường Ngọc Hồi, thị trấn Văn Điển, Thanh Trì, Hà Nội, Việt Nam
№ 1870406363455311203 A. thukho1 ✉ thukho_hcm_1@gmail.com	№ 2967718649523490580 ⌚ 69 Hoàng Văn Thụ, phường 8, Phú Nhuận, Thành phố Hồ Chí Minh, Việt Nam
№ 6135757458357767038 A. thukho2 ✉ thukho_hcm_2@gmail.com	№ 33590645893015587782 ⌚ 291 Nguyễn Sơn, Phú Thạnh, quận Tân Phú, Thành phố Hồ Chí Minh, Việt Nam

Hình 5.32: Xem danh sách các kho và thủ kho phụ trách của hệ thống

- **Xem thống kê**



Hình 5.33: Xem thống kê đơn hàng và số tiền khách hàng đã thanh toán

- **Thêm tài xế vào hệ thống**

THÊM TÀI XẾ

Email: khoavnghanh@gmail.com

Số điện thoại: 0918032237

Tên tài khoản: Khoa Vuong

Địa chỉ: 120 Dinh Tiên Hoàng, Đà Kao, Quận 1, Thành phố Hồ Chí Minh, Việt Nam

Role: Liên tỉnh

Kho: 321 Đường Ngọc Hồi, thị trấn Văn Điển, Thanh Trì, Hà Nội, Việt Nam

ID phương tiện: 6040756074083994749

Mật khẩu: *****

Nhập lại mật khẩu: *****

Đăng ký

Hình 5.34: Thêm tài xế vào hệ thống

- Thêm thủ kho vào hệ thống

THÊM KHO

Địa chỉ: 153 Nguyễn Chí Thanh, Phường 9 (Quận 5), Quận 5, Thành phố Hồ Chí Minh, Việt Nam

Thêm kho

Hình 5.35: Thêm kho vào hệ thống

THÊM THỦ KHO

Email: khoavanganh@gmail.com

Số điện thoại: 0933477392

Tên tài khoản: Khoa Vuong

Địa chỉ: 120 Dinh Tiên Hoàng, Đà Kao, Quận 1, Thành phố Hồ Chí Minh, Việt Nam

Kho: 291 Nguyễn Sơn, Phú Thạnh, quận Tân Phú, Thành phố Hồ Chí Minh, Việt Nam

Mật khẩu: *****

Nhập lại mật khẩu: *****

Đăng ký

Hình 5.36: Thêm thủ kho vào hệ thống

- Xem yêu cầu hỗ trợ

Mã đơn hàng	Phản hồi yêu cầu	Phản hồi
№ 12312421312412 ⌚ 7/14/2021, 12:15:32 AM	<p>Mã đơn hàng: 12312421312412</p> <p>Nội dung phản hồi:</p> <p>Chúng tôi đang cố gắng xử lý nhanh nhất có thể.</p>	<p>Click để phản hồi</p>
№ 41241232421312412 ⌚ 7/14/2021, 12:15:32 AM	<p>Mã đơn hàng: 41241232421312412</p> <p>Nội dung phản hồi:</p> <p>Aliquam a nulla semper,</p>	<p>Click để phản hồi</p>
№ 51232312421312412 ⌚ 7/14/2021, 12:15:32 AM	<p>Mã đơn hàng: 51232312421312412</p> <p>Nội dung phản hồi:</p> <p>Lorem ipsum</p>	<p>Click để phản hồi</p>
№ 351232312421312412 ⌚ 7/14/2021, 12:15:32 AM	<p>Mã đơn hàng: 351232312421312412</p> <p>Nội dung phản hồi:</p> <p>mperdiet lectus</p>	<p>Click để phản hồi</p>
№ 51232312421312412 ⌚ 7/14/2021, 12:15:32 AM	<p>Mã đơn hàng: 51232312421312412</p> <p>Nội dung phản hồi:</p> <p>Maecenas rutrum</p>	<p>Click để phản hồi</p>

Hình 5.37: Giao diện trả lời những yêu cầu cần cầu hỗ trợ từ khách hàng

Mã đơn hàng	Nội dung	Phản hồi
№ 51232312421312412 7/14/2021, 12:17:39 AM	Bon den chiec	sed eu enim
№ 351232312421312412 7/14/2021, 12:17:39 AM	Aliquam a nulla semper,	Đang thúc tài xế giao nhanh
№ 351232312421312412 7/14/2021, 12:17:39 AM	Loem ipsum	Morbi eget laoreet diam, scelerisque hendrerit quam
№ 51232312421312412 7/14/2021, 12:17:39 AM	mperdiet lectus	Maecenas rutrum

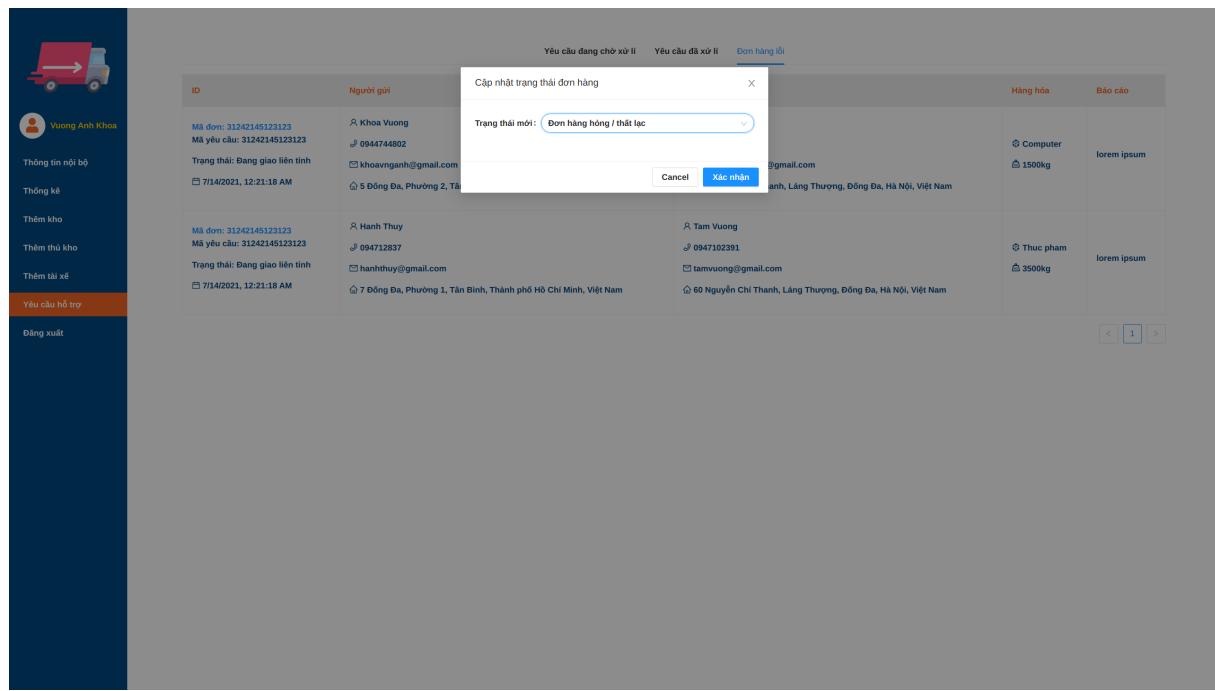
Hình 5.38: Giao diện xem lại những yêu cầu hỗ trợ đã trả lời

- Xem báo cáo sự cố

ID	Người gửi	Người nhận	Hàng hóa	Báo cáo
Mã đơn: 31242145123123 Mã yêu cầu: 31242145123123 Trạng thái: Đang giao liên tỉnh 7/14/2021, 12:21:18 AM	Khoa Vuong 0944744802 khoavanganh@gmail.com 5 Đồng Da, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	Dung Dang 0913154708 dangvanduong0812@gmail.com 54A Nguyễn Chí Thanh, Láng Thương, Đồng Da, Hà Nội, Việt Nam	Computer 1500kg	lorem ipsum
Mã đơn: 31242145123123 Mã yêu cầu: 31242145123123 Trạng thái: Đang giao liên tỉnh 7/14/2021, 12:21:18 AM	Hanh Thuy 094712837 hanhthuy@gmail.com 7 Đồng Da, Phường 1, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	Tam Vuong 0947102391 tamvuong@gmail.com 60 Nguyễn Chí Thanh, Láng Thương, Đồng Da, Hà Nội, Việt Nam	Thuc pham 3500kg	lorem ipsum

Hình 5.39: Giao diện xem những sự cố báo cáo từ tài xế

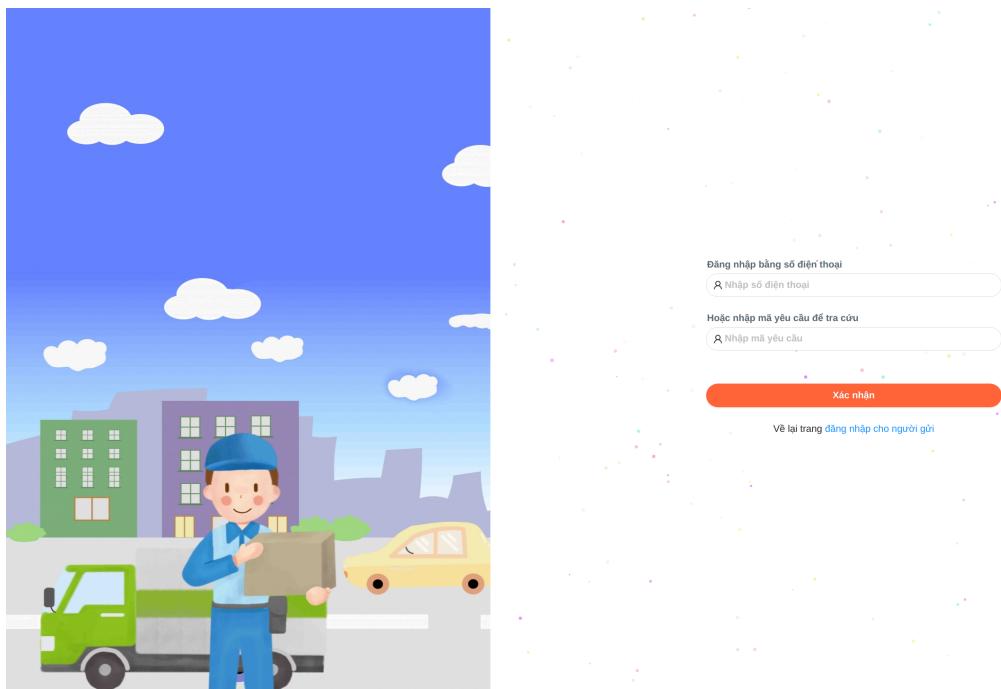
- Xử lý sự cố



Hình 5.40: Giao diện gán lại trạng thái cho những đơn có sự cố

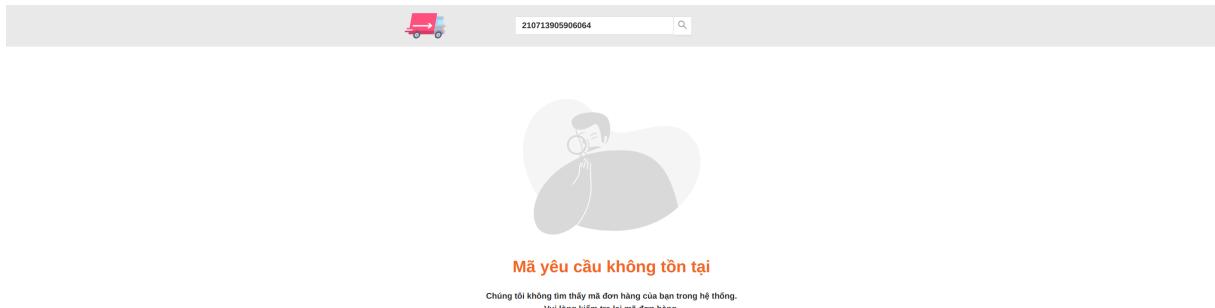
5.3.6 Chức năng dành cho người nhận

- Màn hình đăng nhập của người nhận, người nhận có thể nhập mã yêu cầu để xem tình trạng của yêu cầu đó hoặc đăng nhập bằng số điện thoại để thấy tất cả các đơn hàng gửi cho mình.

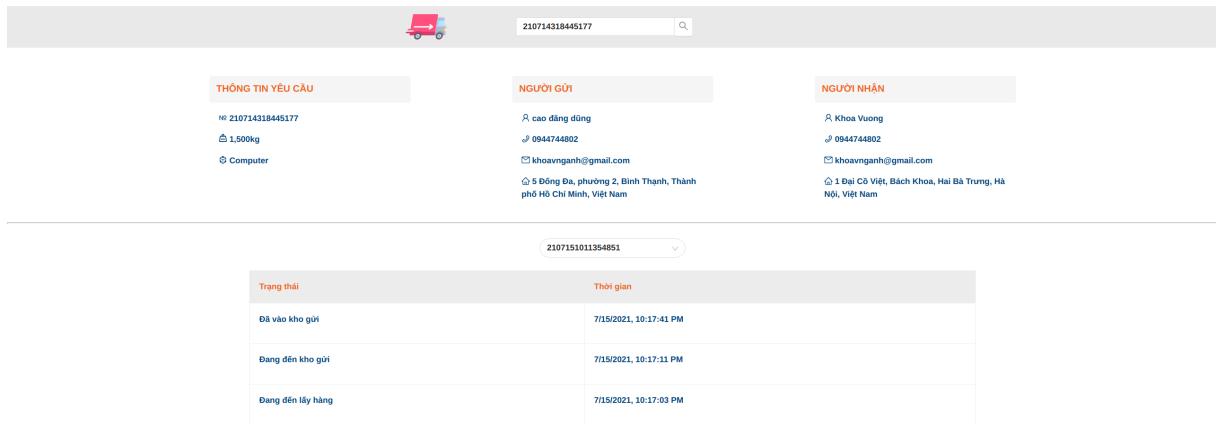


Hình 5.41: Giao diện đăng nhập cho người nhận

- Nhập mã yêu cầu để xem thông tin và trạng thái yêu cầu và các đơn hàng liên quan.

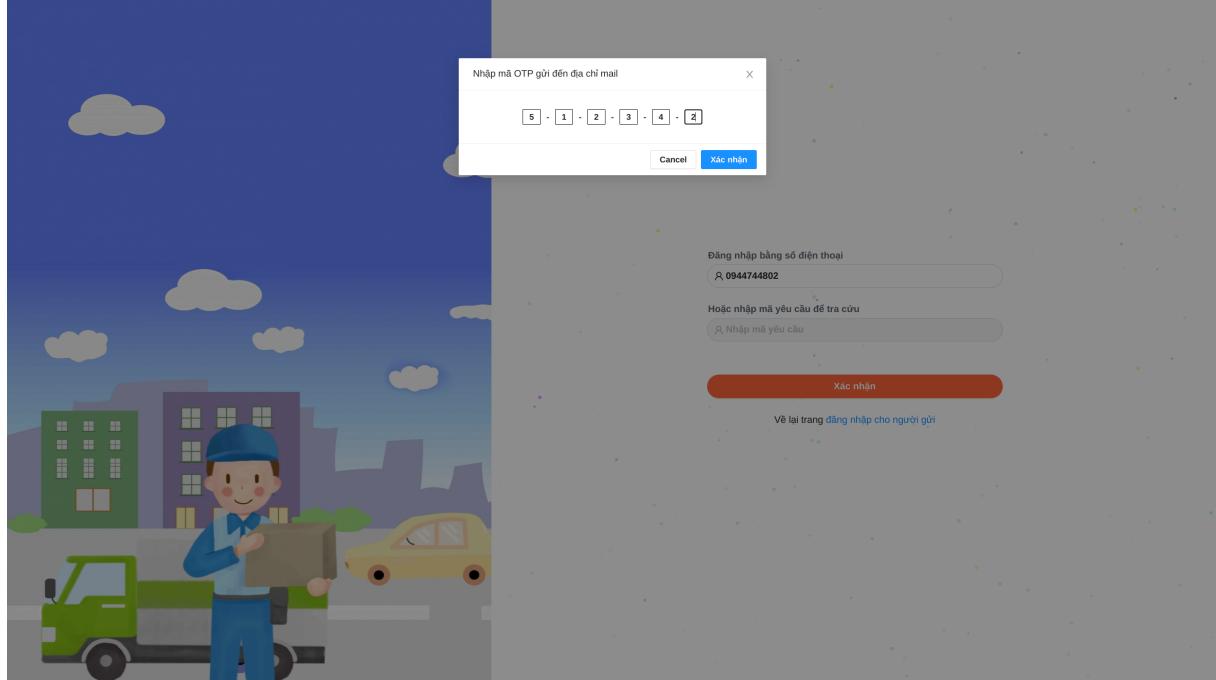


Hình 5.42: Giao diện không tìm thấy mã yêu cầu



Hình 5.43: Giao diện tìm thấy mã yêu cầu

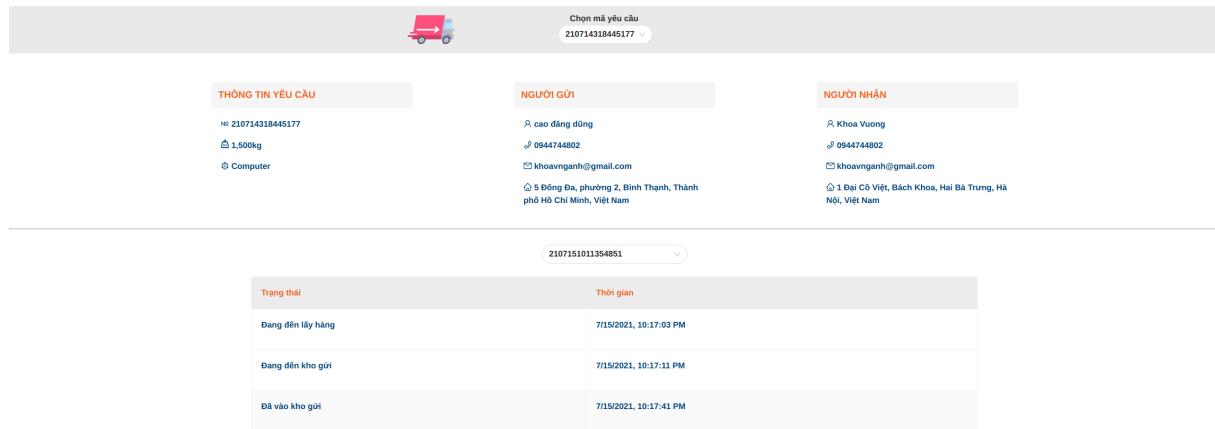
- Nhập số điện thoại và mã OTP.



Hình 5.44: Giao diện nhập mã OTP nếu chọn theo dõi yêu cầu bằng số điện thoại

- Nếu nhập mã OTP thành công, người nhận sẽ được đưa đến giao diện để xem các

yêu cầu và đơn hàng được gửi cho mình



Hình 5.45: Giao diện các yêu cầu và đơn hàng của người nhận

6

Kiểm thử hệ thống

Kiểm thử phần mềm là một việc rất quan trọng trong quy trình phát triển phần mềm. Để một hệ thống có thể đi vào hoạt động ổn định, các nhà phát triển cần kiểm tra tất cả tính năng, các thành phần của hệ thống, mọi thứ phải hoạt động một cách trơn tru mới có thể chuyển giao đến tay người sử dụng. Bất kỳ một lỗi nhỏ nào nếu không kiểm tra kỹ càng cũng có thể để lại hậu quả về sau, làm cho hệ thống không đáng tin cậy và không được sự hưởng ứng từ phía người dùng.

Quy trình kiểm thử cần phải qua nhiều loại khác nhau: Unit test, Intergration test, System test, Acceptance test, Functional testing, Non Functional testing, End to end test... Hiện nay có rất nhiều thư viện kiểm thử hiệu quả và mạnh mẽ. Katalon Recorder là một extension dành cho chrome đã tích hợp tính năng kiểm thử vào trong nó để kiểm thử chức năng. Jmeter là công cụ để đo độ tải và performance của trang web và nhóm đã sử dụng K6 cloud platform có tích hợp jmeter để có thể kiểm thử phi chức năng cho trang web của nhóm.

6.1 Kiểm thử API

Đối với hệ thống nhóm xây dựng, các thành phần giao tiếp với nhau thông qua API. Vì vậy, cần phải kiểm thử riêng biệt các API trước khi tích hợp vào các bên. Nhóm chọn phần mềm **Postman** để kiểm thử các API.

API đăng nhập

The screenshot shows the Postman interface for a POST request to <https://auth-management-staging.herokuapp.com/api/auth/sign-in>. The 'Body' tab is selected, showing a JSON payload:

```

1
2   "username": "0944744802",
3   "password": "123456"
4

```

Hình 6.1: Kiểm thử với API đăng nhập

The screenshot shows the Postman interface after sending the request. The response status is 200 OK, with a response time of 3.06 s and a size of 992 B. The 'Pretty' tab is selected, displaying the JSON response:

```

1
2   "id": 1218206887201882556,
3   "username": "Vuong Anh Khoa",
4   "email": "khoavanganh@gmail.com",
5   "phone": "0944744802",
6   "address": "5 Đống Đa, phường 2, Bình Thạnh, Thành phố Hồ Chí Minh, Việt Nam",
7   "token": "eyJhbGciOiJIUzUxMiJ9.
8     eyJqdGkiOiIxMjE4MjA2Dg3MjAxODgyNTU2Iiwic3ViIjoiMDk0NDc0NDgwMiIsImIhdCI6MTYyNjI3MzAzOSwiZXhwIjoxNjI2
9     Mjc2NjM5fQ.3EoCScQcFhk0cqY_IM9H0w98p5jKv6FAPyN7hMyWYnHxoTbDVoOwojMwbUbCED75J5-N9QaE1I4vqyaPbExzw",
10  "type": "Bearer",
11  "role": "ROLE_USER",
12  "provinceCode": "2"
13

```

Hình 6.2: Kết quả kiểm thử với API đăng nhập

API đăng ký

The screenshot shows the Postman interface for testing a POST request to the 'Sign Up' endpoint of the 'API auth-management'. The URL is <https://auth-management-staging.herokuapp.com/api/auth/sign-up>. The 'Body' tab is selected, showing a JSON payload:

```

1
2   "email": "sender2@gmail.com",
3   "phone": "0999899990",
4   "username": "cao-dang dung",
5   "address": "63 Thân Nhân Trung, Phường 13, Tân Bình, Thành phố Hồ Chí Minh",
6   "password": "123456",
7   "provinceCode": "58"
8

```

Hình 6.3: Kiểm thử với API đăng ký

The screenshot shows the Postman interface after sending the POST request. The status bar indicates a successful 200 OK response. The 'Body' tab is selected, showing the JSON response:

```

1
2   "message": "User registered successfully!"
3

```

Hình 6.4: Kết quả kiểm thử với API đăng ký

API thêm kho

The screenshot shows a Postman interface for a POST request to "https://auth-management-staging.herokuapp.com/api/crud/add-address-warehouse". The "Body" tab is selected, showing a JSON payload:

```

1
2   ...
3     "areaCode": "58",
4     "address": "63 Thân Nhân Trung, Phường 13, Tân Bình, Thành phố Hồ Chí Minh"

```

Hình 6.5: Kiểm thử với API thêm kho

The screenshot shows the Postman interface after sending the request. The status bar indicates "200 OK" with a response time of "3.25 s" and a size of "557 B". The "Pretty" tab in the results section displays the JSON response:

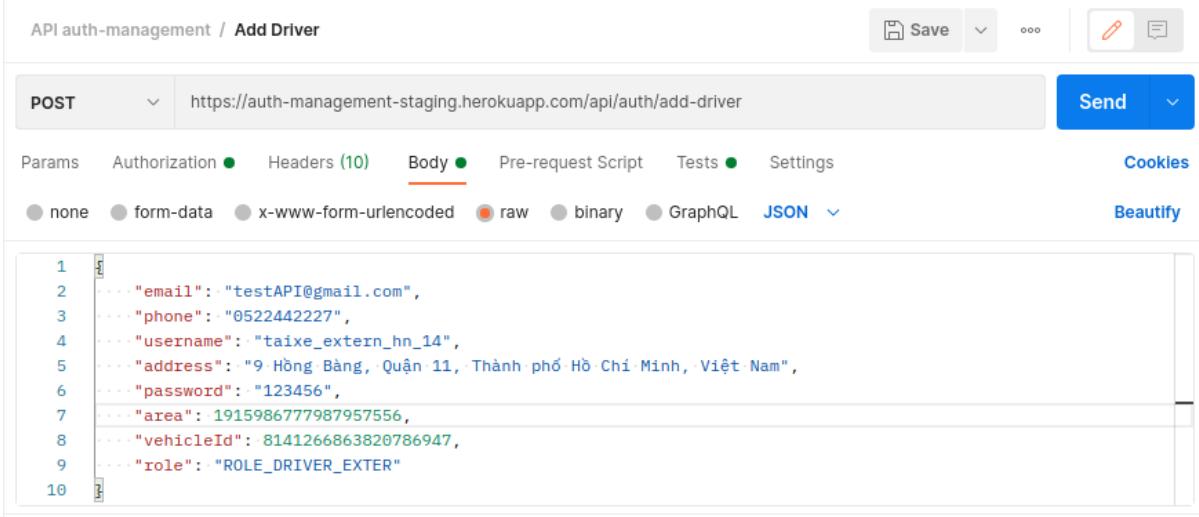
```

1
2   ...
3     "message": "Add address warehouse successfully!"

```

Hình 6.6: Kết quả kiểm thử với API thêm kho

API thêm tài xế



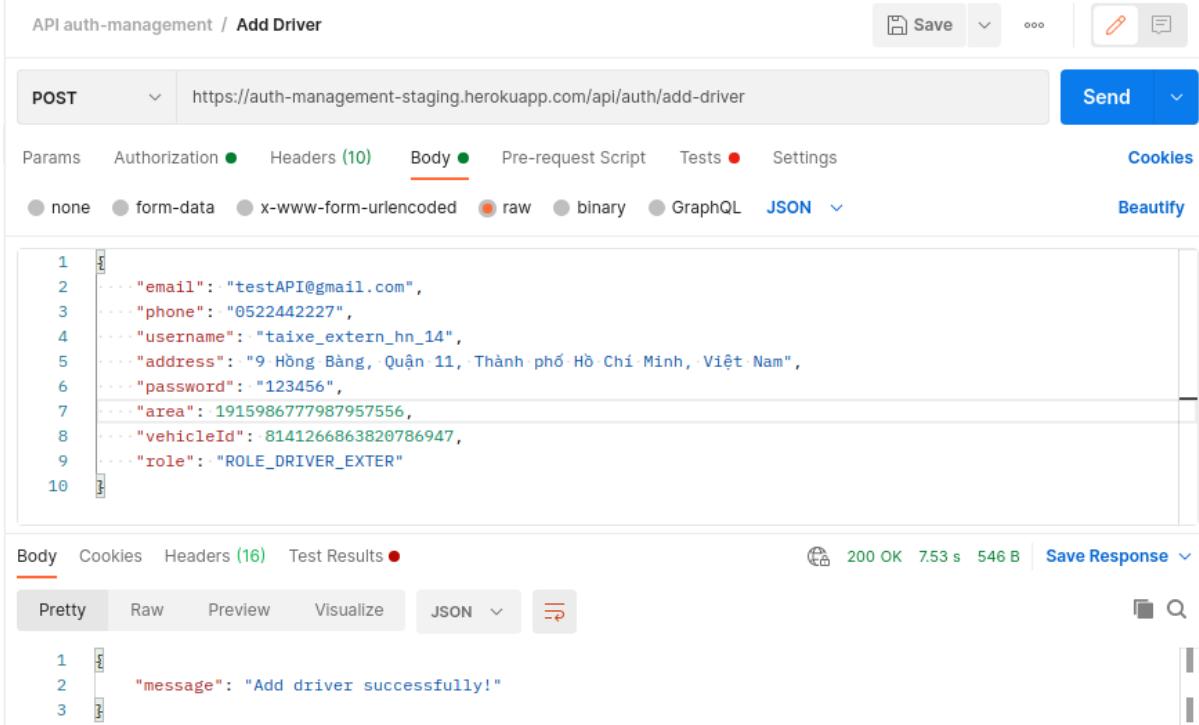
The screenshot shows the Postman interface for testing an API endpoint. The URL is `https://auth-management-staging.herokuapp.com/api/auth/add-driver`. The request method is set to `POST`. The `Body` tab is selected, showing the following JSON payload:

```

1
2   "email": "testAPI@gmail.com",
3   "phone": "0522442227",
4   "username": "tai xe_extern_hn_14",
5   "address": "9 Hồng Bàng, Quận 11, Thành phố Hồ Chí Minh, Việt Nam",
6   "password": "123456",
7   "area": 1915986777987957556,
8   "vehicleId": 8141266863820786947,
9   "role": "ROLE_DRIVER_EXTER"
10

```

Hình 6.7: Kiểm thử với API thêm tài xế



The screenshot shows the Postman interface after sending the request. The status bar indicates a `200 OK` response. The `Body` tab displays the following JSON response:

```

1
2   "message": "Add driver successfully!"
3

```

Hình 6.8: Kết quả kiểm thử với API thêm tài xế

API thêm thủ kho

The screenshot shows the Postman interface for testing an API endpoint. The URL is <https://auth-management-staging.herokuapp.com/api/auth/add-stocker>. The method is set to POST. The Body tab is selected, showing a JSON payload:

```

1
2   "email": "thukho_hn_51111@gmail.com",
3   "phone": "0944666445",
4   "username": "thukho_hn_5",
5   "address": "321 Đường Ngọc Hồi, thị trấn Văn Điển, Thanh Trì, Hà Nội, Việt Nam",
6   "password": "123456",
7   "areaWarehouseId": "5824032806563105525"
8

```

Hình 6.9: Kiểm thử với API thêm thủ kho

The screenshot shows the Postman interface after sending the POST request. The status is 200 OK, and the response body is:

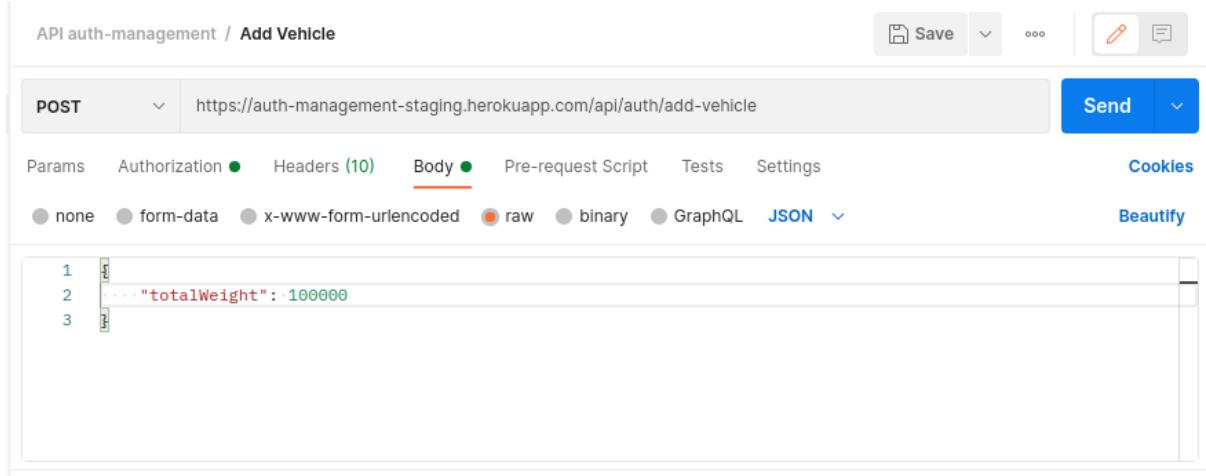
```

1
2   "message": "Add stocker successfully!"
3

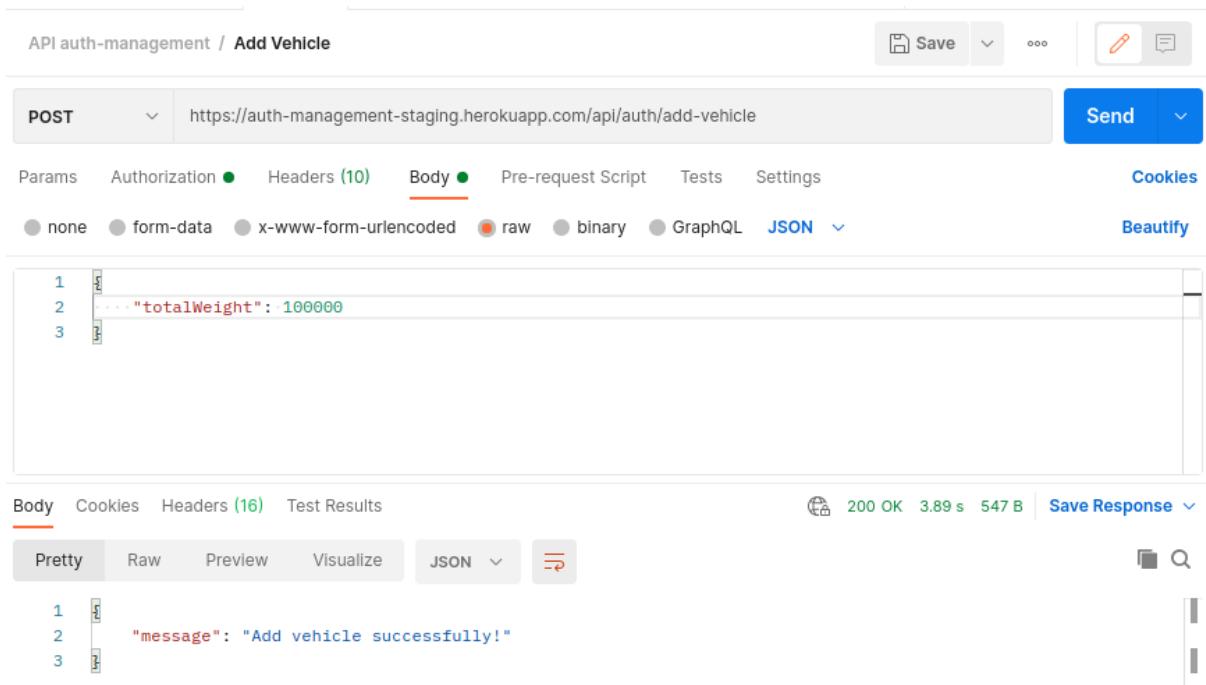
```

Hình 6.10: Kết quả kiểm thử với API thêm thủ kho

API thêm phương tiện

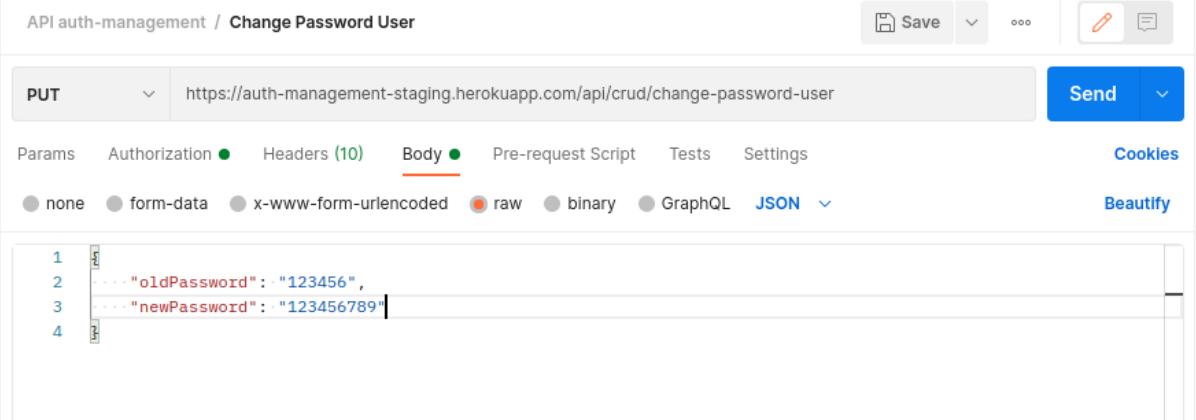


Hình 6.11: Kiểm thử với API thêm phương tiện



Hình 6.12: Kết quả kiểm thử với API thêm phương tiện

API đổi mật khẩu



The screenshot shows a POST request in Postman to the URL `https://auth-management-staging.herokuapp.com/api/crud/change-password-user`. The request method is set to `PUT`. The `Body` tab is selected, showing the following JSON payload:

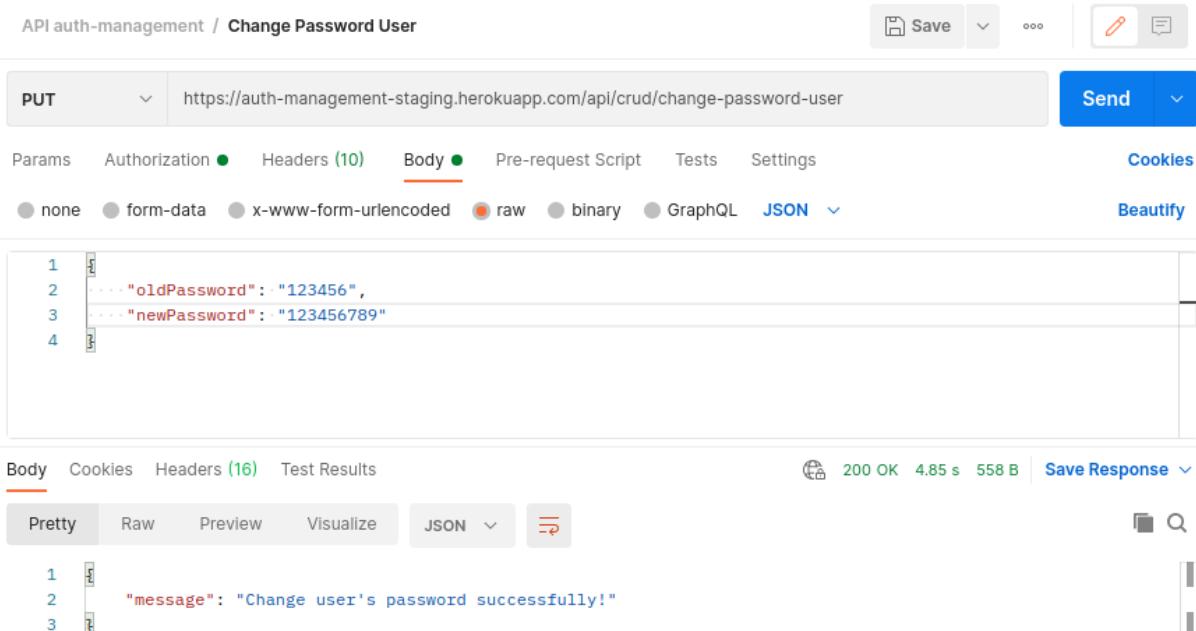
```

1
2   ...
3   ...
4

```

The payload contains two fields: `oldPassword` and `newPassword`, both set to the value `"123456"`.

Hình 6.13: Kiểm thử với API đổi mật khẩu



The screenshot shows the response from the previous POST request. The status code is `200 OK`, and the response time is `4.85 s`. The response body is as follows:

```

1
2   ...
3   ...
4

```

The response message is `"message": "Change user's password successfully!"`.

Hình 6.14: Kết quả kiểm thử với API đổi mật khẩu

API lấy phí vận chuyển

The screenshot shows a Postman interface for a POST request to <https://auth-management-staging.herokuapp.com/api/get-transportation-cost>. The 'Body' tab is selected, showing the following JSON payload:

```

1
2   ...
3   ...
4   ...
5   ...
6   ...
7
  
```

```

1
2   ...
3   ...
4   ...
5   ...
6   ...
7
  
```

Hình 6.15: Kiểm thử với API lấy phí vận chuyển

The screenshot shows the results of the POST request. The status is 200 OK, with a response time of 5.39 s and a response size of 862 B. The response body is as follows:

```

1
2
3
4
5
6
7
  
```

```

1
2
3
4
5
6
7
  
```

Body Cookies Headers (16) Test Results

Pretty Raw Preview Visualize JSON

```

1
2
3
4
5
6
7
8
  
```

```

1
2
3
4
5
6
7
8
  
```

Hình 6.16: Kết quả kiểm thử với API lấy phí vận chuyển

API lấy tài xế liên tỉnh ở kho

The screenshot shows the Postman interface for a POST request to <https://auth-management-staging.herokuapp.com/auth-management/get-external-drivers-by-stockid>. The 'Body' tab is selected, showing the following JSON payload:

```

1 "stockId":1181267876917690786,
2 "reqTime":1619448542005,
3 "clientId":1,
4 "sig":"eceab686481cb29b1042bf0e4367339e89298c30d407b6198f82611681a4f1ad"
5
6

```

Hình 6.17: Kiểm thử với API lấy tài xế liên tỉnh ở kho

The screenshot shows the Postman interface after sending the POST request. The response status is 200 OK, and the response body is displayed in the 'Pretty' tab:

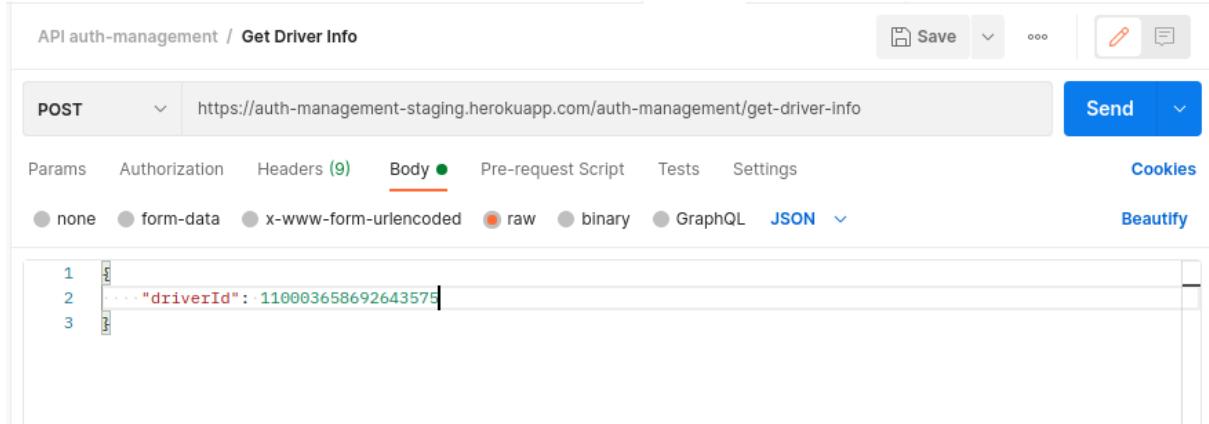
```

1 "data": [
2     "drivers": [
3         {
4             "driverId": 164929436152186575,
5             "status": 1,
6             "driverName": "tai xe_extern_hn_10",
7             "driverEmail": "tai xe_extern_hn_10@gmail.com",
8             "phone": "097777775",
9             "truckInfo": {
10                 "remainWeight": 40000.0
11             }
12         },
13         {
14             "driverId": 5540822577411280220,
15             "status": 1,
16             "driverName": "tai xe_extern_hn_9",
17             "driverEmail": "tai xe_extern_hn_9@gmail.com",
18             "phone": "097777774",
19             ...
20         }
21     ]
22 ]

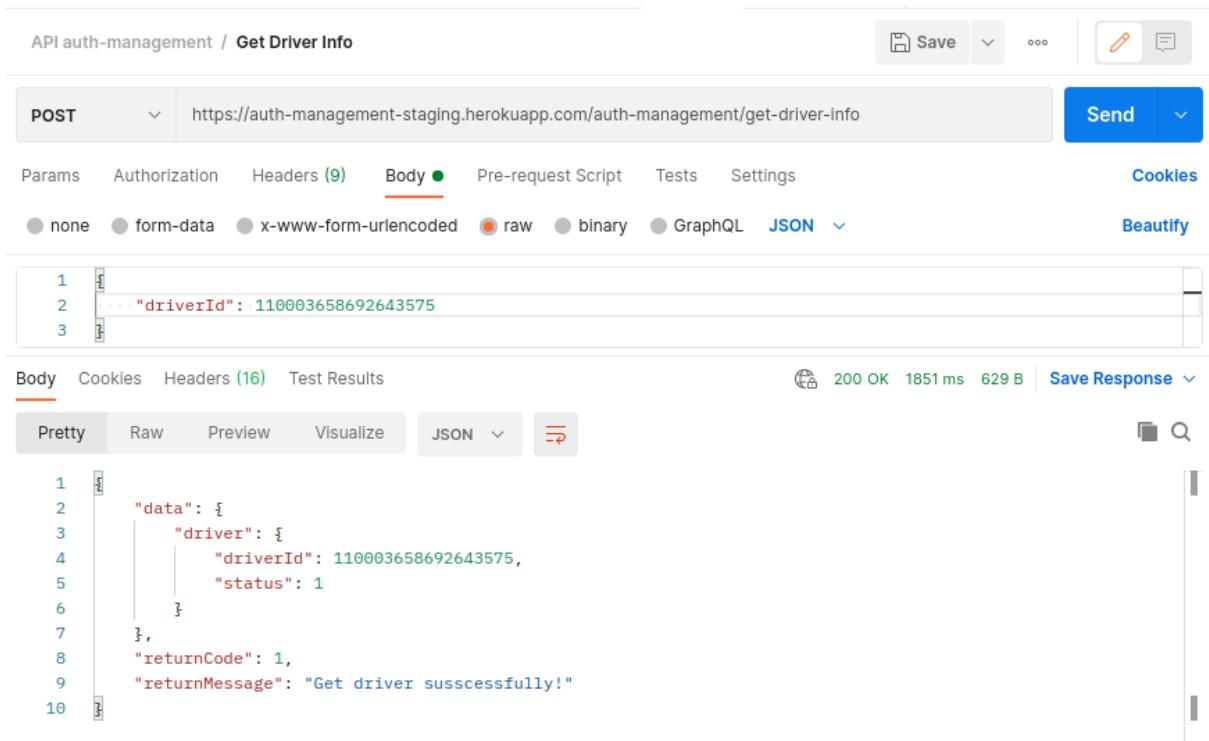
```

Hình 6.18: Kết quả kiểm thử với API lấy tài xế liên tỉnh ở kho

API lấy thông tin tài xế



Hình 6.19: Kiểm thử với API lấy thông tin tài xế



Hình 6.20: Kết quả kiểm thử với API lấy thông tin tài xế liên

API lấy danh sách tài xế ở kho

API auth-management / Get Driver Each Warehouse

GET https://auth-management-staging.herokuapp.com/api/crud/get-driver-each-warehouse

Authorization (●)

Type: Bearer ...

The authorization header will be automatically generated when you send the request.

Learn more about authorization ↗

Token: eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiI2MTM ...

Hình 6.21: Kiểm thử với API lấy danh sách tài xế ở kho

API auth-management / Get Driver Each Warehouse

GET https://auth-management-staging.herokuapp.com/api/crud/get-driver-each-warehouse

Headers (8)

Body

```

1 [
2   {
3     "username": "taixe_intern_hcm_14",
4     "phone": "0511111113",
5     "driverEntity": {
6       "id": 219062391252470889,
7       "area": 3359064593015587782,
8       "vehicleId": 8112675337965487516,
9       "status": 1
10    }
11  },
12  {
13    "username": "taixe_extern_hcm_2",
14    "phone": "0955555557",
15    "driverEntity": {
16      "id": 1875794268572830242,
17      "area": 3359064593015587782,
18      "vehicleId": 5989767163782974681,
19      "status": 1
20    }
21  ]

```

Hình 6.22: Kết quả kiểm thử với API lấy danh sách tài xế ở kho

API lấy danh sách tài xế

The screenshot shows the Postman interface for an 'API FOR ADMIN / Get List Driver'. The request method is 'GET' and the URL is 'https://auth-management-staging.herokuapp.com/api/crud/get-list-driver'. The 'Authorization' tab is selected, showing a 'Bearer...' dropdown and a token input field containing a long string of characters. Other tabs include 'Params', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'. A note about sensitive data and variables is displayed in a tooltip.

Hình 6.23: Kiểm thử với API lấy danh sách tài xế

The screenshot shows the Postman interface after sending the 'Get List Driver' request. The status is '200 OK' with a response time of '3.66 s' and a size of '17.37 KB'. The 'Pretty' tab is selected, displaying the JSON response. The response contains multiple objects, each representing a driver. One object is shown in detail:

```

1
2   {
3     "id": 110003658692643575,
4     "username": "taixe_intern_hn_7",
5     "role": "ROLE_DRIVER_INTER",
6     "email": "taixe_intern_hn_7@gmail.com",
7     "phone": "0977777777",
8     "vehicleResponse": {
9       "id": 4010125043573084090,
10      "totalWeight": 5000.0,
11      "currentWeight": 0.0
12    },
13    "warehouseResponse": {
14      "id": 1181267876917690786,
15      "address": "321 Đường Ngọc Hồi, thị trấn Văn Điển, Thanh Trì, Hà Nội, Việt Nam",
16      "areaCode": "1"
17    }
18  },
19  {
20    ...
}

```

Hình 6.24: Kết quả kiểm thử với API lấy danh sách tài xế

API lấy danh sách thủ kho

The screenshot shows the Postman interface for a GET request to <https://auth-management-staging.herokuapp.com/api/crud/get-list-stocker>. The 'Authorization' tab is selected, showing a 'Bearer ...' dropdown and a note about sensitive data. The 'Token' field contains a long JWT token. Other tabs include Params, Headers (8), Body, Pre-request Script, Tests, Settings, and Cookies.

Hình 6.25: Kiểm thử với API lấy danh sách thủ kho

The screenshot shows the Postman interface after sending the request. The response status is 200 OK, with a duration of 1821 ms and a size of 1.53 KB. The 'Pretty' tab displays the JSON response, which includes two user objects. Each user object has an 'id', 'username', 'role', 'email', and a nested 'warehouseResponse' object containing an 'id', 'address', and 'areaCode'. The JSON is numbered from 1 to 19.

```

1 {
2   "id": 616194410572172711,
3   "username": "thukho_hn_2",
4   "role": "ROLE_STOCKER",
5   "email": "thukho_hn_2@gmail.com",
6   "warehouseResponse": {
7     "id": 1915986777987957556,
8     "address": "4b Đền Lừ 2, Hoàng Văn Thụ, quận Hoàng Mai, Hà Nội, Việt Nam",
9     "areaCode": "1"
10   }
11 },
12 {
13   "id": 906828411704329174,
14   "username": "thukho_hn_1",
15   "role": "ROLE_STOCKER",
16   "email": "thukho_hn_1@gmail.com",
17   "warehouseResponse": {
18     "id": 1181267876917690786,
19     ...
20   }
21 }

```

Hình 6.26: Kết quả kiểm thử với API lấy danh sách thủ kho

API lấy danh sách kho

The screenshot shows the Postman interface for a 'GET' request to 'https://auth-management-staging.herokuapp.com/api/crud/get-list-warehouse'. The 'Authorization' tab is selected, showing a 'Bearer...' dropdown and a token input field containing 'eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiI2MTM1...'. Other tabs include 'Params', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'.

Hình 6.27: Kiểm thử với API lấy danh sách kho

The screenshot shows the Postman interface after sending the request. The response status is '200 OK' with a duration of '1761 ms' and a size of '1.05 KB'. The 'Pretty' tab is selected, displaying a JSON array of four warehouse objects. Each object has an 'id', 'areaCode', and 'address' field.

```

1
2 [
3     {
4         "id": 1181267876917690786,
5         "areaCode": "1",
6         "address": "321 Đường Ngọc Hồi, thị trấn Văn Điển, Thanh Trì, Hà Nội, Việt Nam"
7     },
8     {
9         "id": 1915986777987957556,
10        "areaCode": "1",
11        "address": "4b Dãen Lừ 2, Hoàng Văn Thu, quận Hoàng Mai, Hà Nội, Việt Nam"
12    },
13    {
14        "id": 2967718649523490580,
15        "areaCode": "2",
16        "address": "69 Hoàng Văn Thủ, phường 8, Phú Nhuận, Thành phố Hồ Chí Minh, Việt Nam"
17    },
18    {
19        "id": 3359064593015587782,
20        "areaCode": "2",
21        ...
22    }
]
  
```

Hình 6.28: Kết quả kiểm thử với API lấy danh sách kho

API lấy thông tin người dùng

The screenshot shows the Postman interface for a 'GET' request to the URL `https://auth-management-staging.herokuapp.com/api/auth/get-user`. The 'Authorization' tab is active, with the 'Type' dropdown set to 'Bearer ...'. A note in a callout box says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' Below the note, there is a 'Token' input field containing a long string of characters.

Hình 6.29: Kiểm thử với API lấy thông tin người dùng

The screenshot shows the Postman interface after sending the GET request. The status bar indicates a `200 OK` response with `2.43 s` duration and `992 B` size. The 'Body' tab is selected, displaying the JSON response in a pretty-printed format. The response contains user information such as id, username, email, phone, address, token, type, role, and provinceCode.

```

1  {
2      "id": 1218206887201882556,
3      "username": "Vuong Anh Khoa",
4      "email": "khoaavngh@gmail.com",
5      "phone": "0944744802",
6      "address": "5 Đống Đa, phường 2, Bình Thạnh, Thành phố Hồ Chí Minh, Việt Nam",
7      "token": "eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiIxMjE4MjA2ODg3MjAxODgyNTU2Iiwic3ViIjoiMDk0NDc0NDgwMiIsImhdCI6MTYyNjI3MzAzOSwiZXhwIjoxNjI2Mjc2NjM5fQ.3EoCScQcFVhk0cqY_IM9H0w98p5jKv6FAPyN7hMyWYnHxoTbDvoOWojMwbUbCED75J5-N9QaE1I4vqyaPbEXzw",
8      "type": "Bearer",
9      "role": "ROLE_USER",
10     "provinceCode": "2"
11 }

```

Hình 6.30: Kết quả kiểm thử với API lấy thông tin người dùng

API lấy danh sách phương tiện chưa sử dụng

The screenshot shows the Postman interface for an API endpoint named 'Get Vehicle Unused'. The method is set to 'GET' and the URL is 'https://auth-management-staging.herokuapp.com/api/crud/get-vehicle-unused'. The 'Authorization' tab is selected, showing a dropdown menu with 'Bearer ...'. A note in a callout box says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables'.

Hình 6.31: Kiểm thử với API lấy danh sách phương tiện chưa sử dụng

The screenshot shows the results of the 'Get Vehicle Unused' API call. The status is '200 OK' with a response time of '1826 ms' and a size of '1.1 KB'. The 'Pretty' tab is selected in the results view, showing a JSON array of vehicle records. Each record includes an ID, total weight, and current weight.

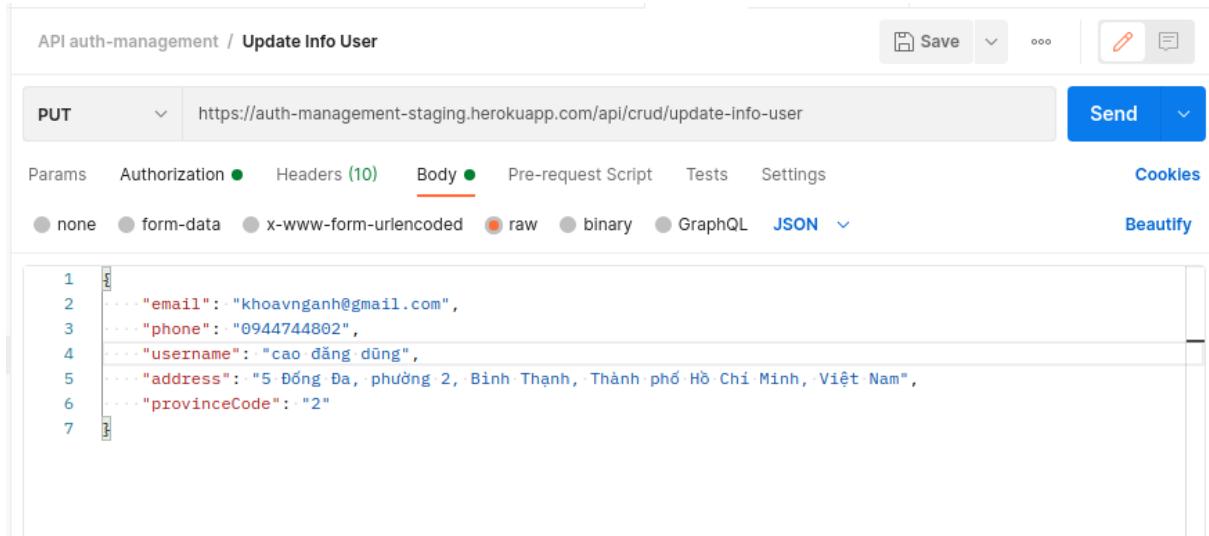
```

1 [
2   {
3     "id": 2582297806224509432,
4     "totalWeight": 10000.0,
5     "currentWeight": 0.0
6   },
7   {
8     "id": 3750723643175684685,
9     "totalWeight": 10000.0,
10    "currentWeight": 0.0
11  },
12  {
13    "id": 4039840245322174022,
14    "totalWeight": 15000.0,
15    "currentWeight": 0.0
16  },
17  {
18    "id": 6040756074083994749,
19    "totalWeight": 50000.0,
20    ...
21  }
]

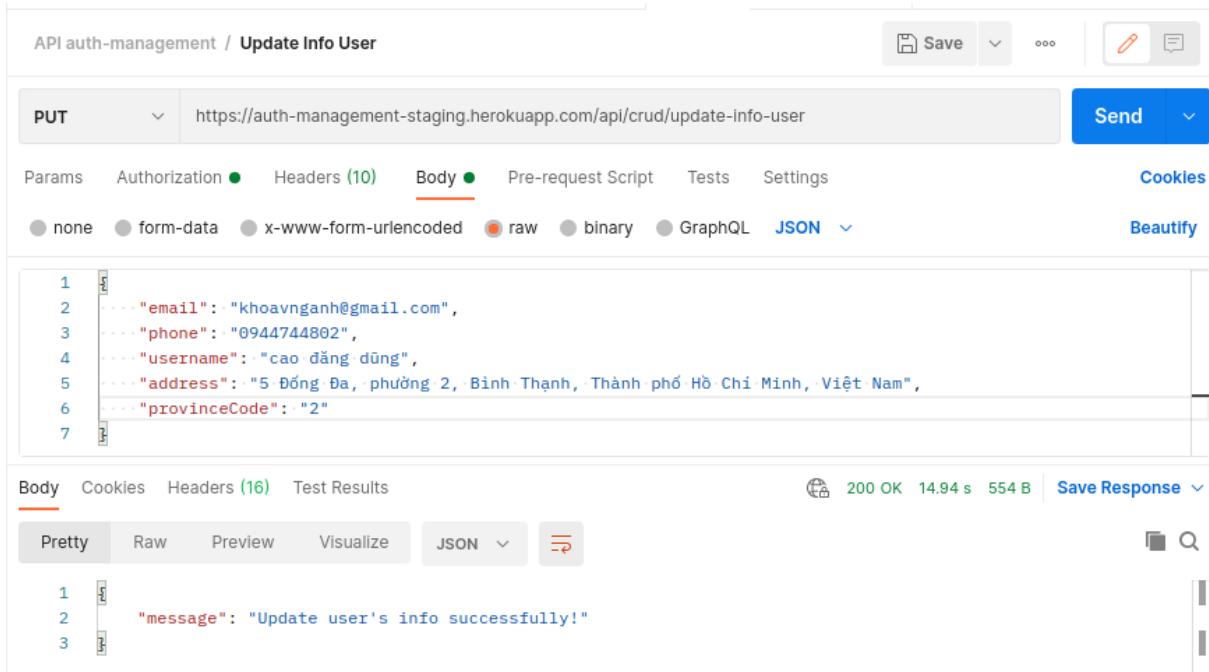
```

Hình 6.32: Kết quả kiểm thử với API lấy danh sách phương tiện chưa sử dụng

API cập nhật thông tin người dùng

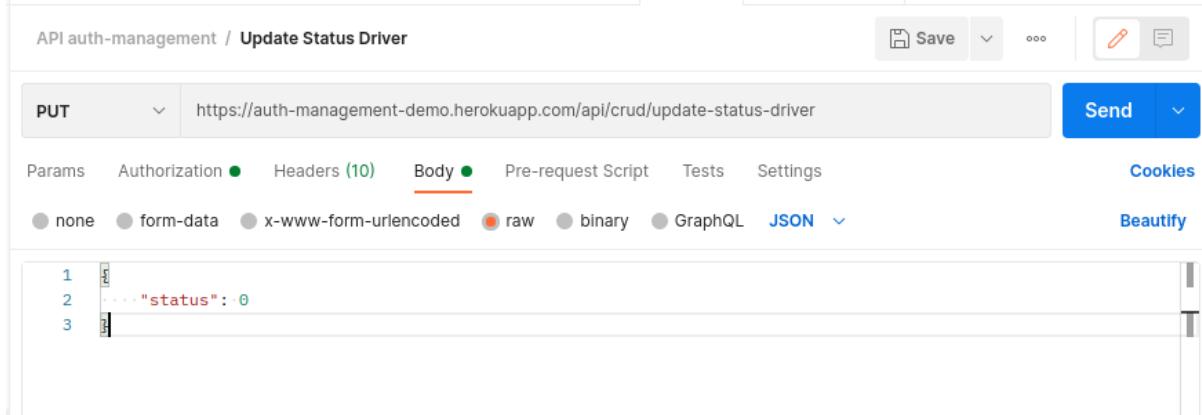


Hình 6.33: Kiểm thử với API cập nhật thông tin người dùng

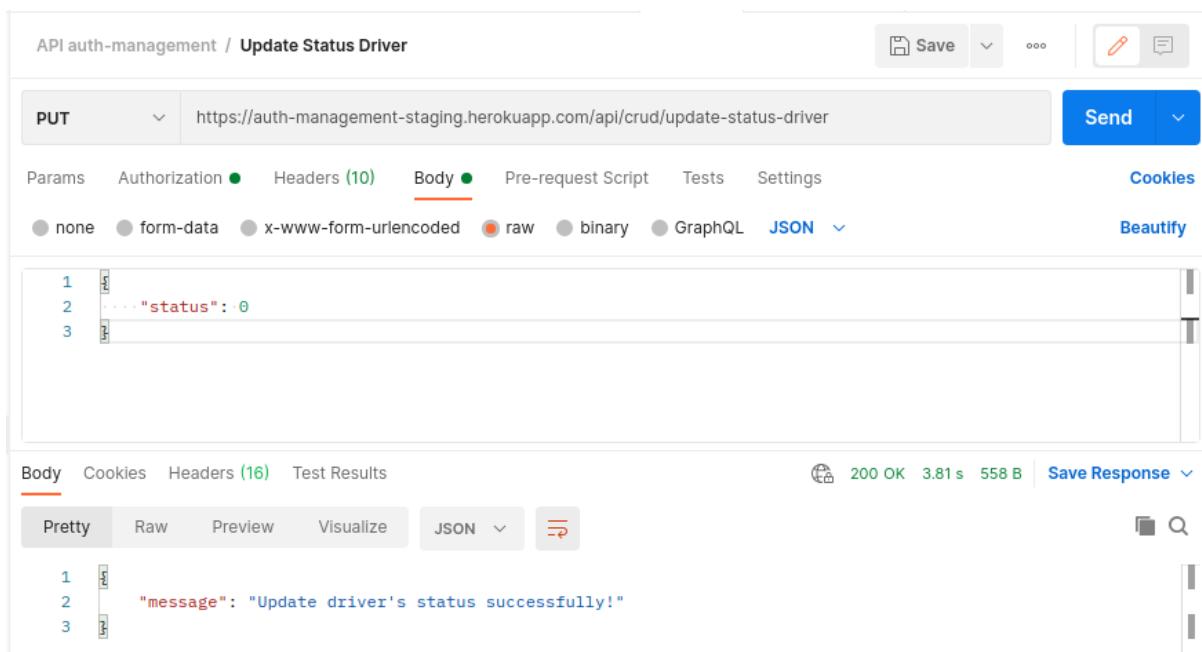


Hình 6.34: Kết quả kiểm thử với API cập nhật thông tin người dùng

API cập nhật trạng thái tài xế



Hình 6.35: Kiểm thử với API cập nhật trạng thái tài xế



Hình 6.36: Kết quả kiểm thử với API cập nhật trạng thái tài xế

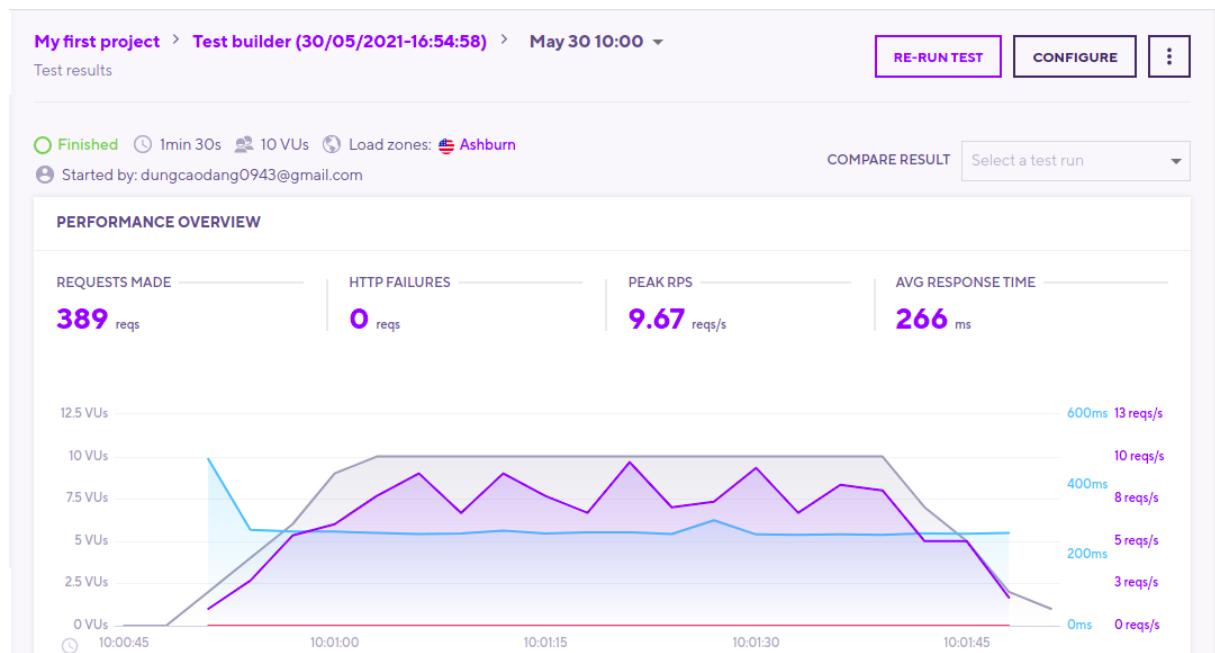
6.2 Kiểm thử chức năng

Vì tính chất nghiệp vụ có nhiều vai trò cũng như chức năng của các vai trò đó của trang web nên nhóm đã sử dụng Katalon Recorder để có thể kiểm thử tự động theo luồng các chức năng cơ bản của hệ thống.

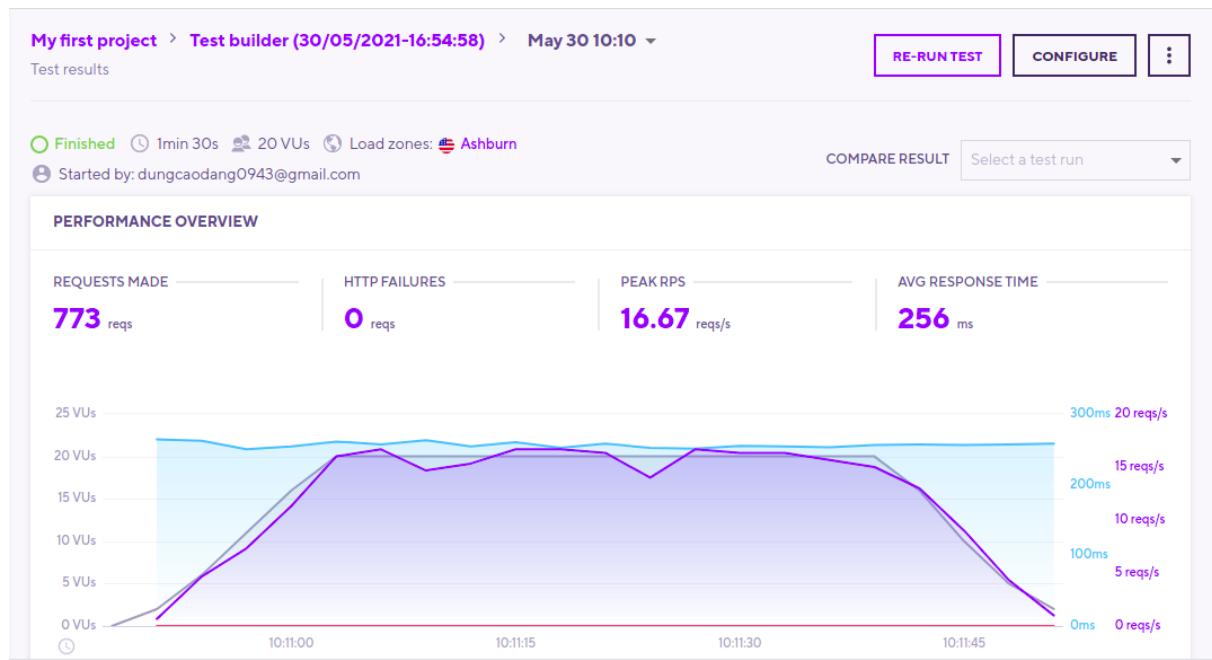
6.3 Kiểm thử phi chức năng

Kiểm thử phi chức năng (Non-functional testing) là kiểm tra một ứng dụng hoặc hệ thống phần mềm cho các yêu cầu phi chức năng của nó: cách thức hoạt động của một hệ thống, thay vì các hành vi cụ thể của hệ thống đó. Có rất nhiều loại kiểm thử phi chức năng như: performance test, baseline test, stress test... Đối với hệ thống của nhóm, nhóm sẽ tiến hành kiểm thử tải (load test). Kiểm tra tải là loại kiểm thử đề cập đến việc mô hình hóa sử dụng dự kiến của một hệ thống bằng cách mô phỏng nhiều người dùng truy cập hệ thống đồng thời.

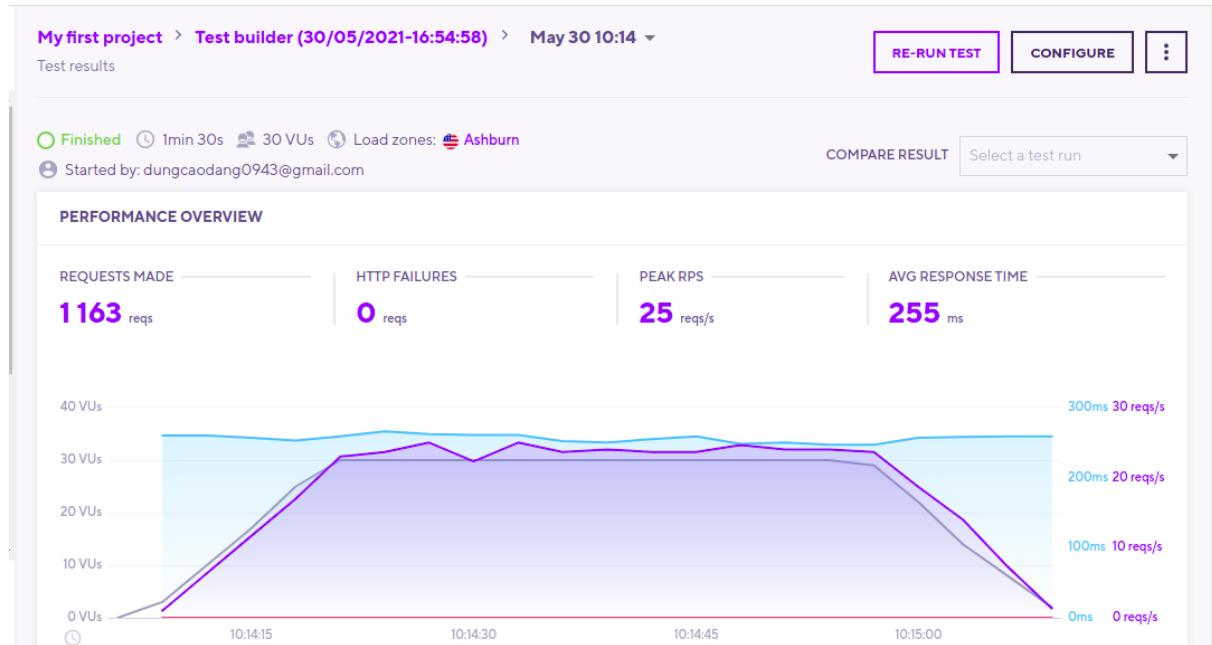
Nhóm sẽ sử dụng K6 cloud platform để tiến hành xây dựng kiểm thử với API đăng nhập: <https://auth-management-staging.osc-fr1.scalingo.io/api/auth/sign-in>



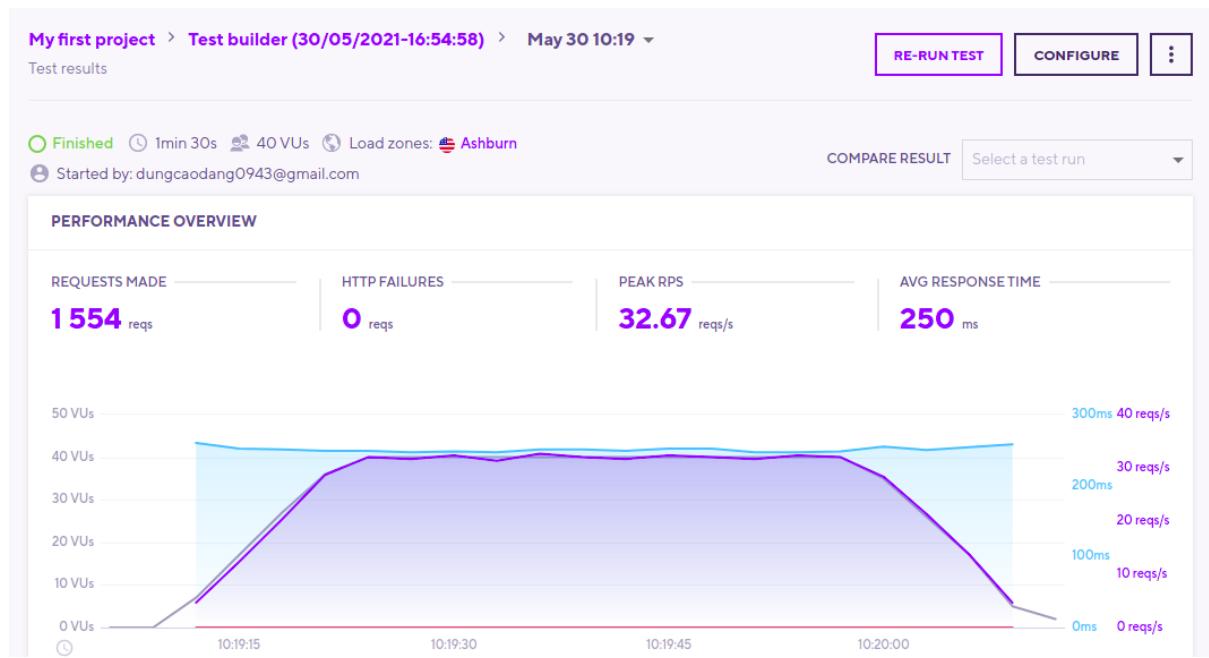
Hình 6.37: Kiểm thử với 10 users đồng thời



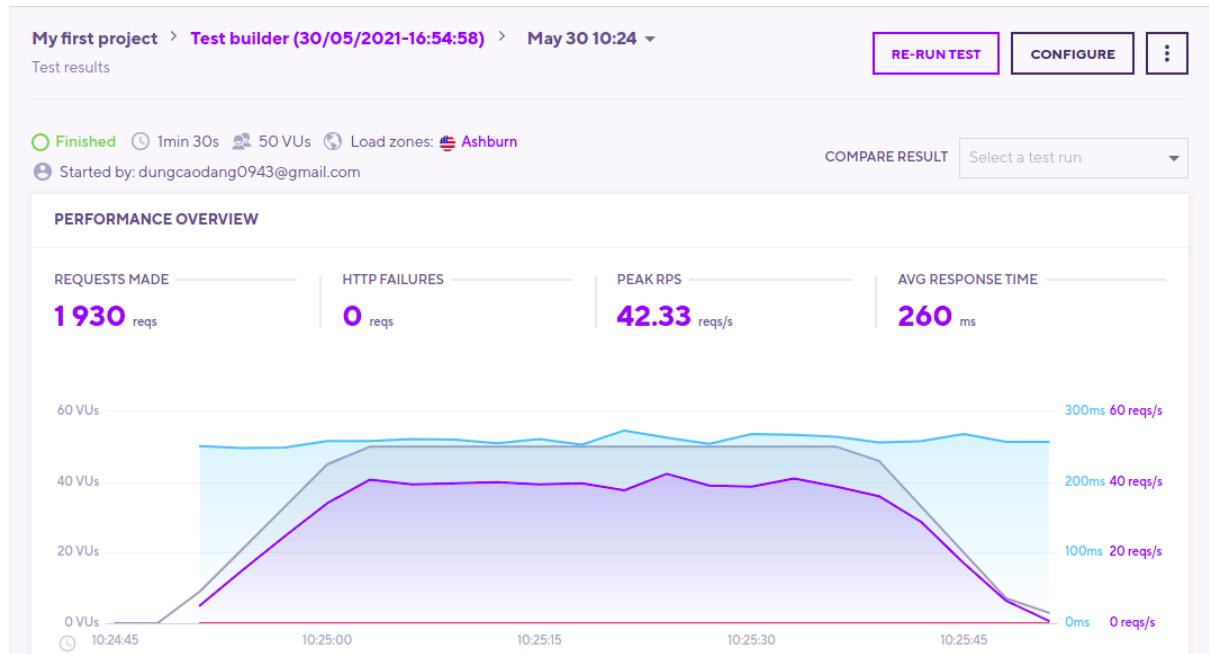
Hình 6.38: Kiểm thử với 20 users đồng thời



Hình 6.39: Kiểm thử với 30 users đồng thời



Hình 6.40: Kiểm thử với 40 users đồng thời



Hình 6.41: Kiểm thử với 50 users đồng thời

Kết quả kiểm thử được thống kê trong bảng sau:

Concurrent users	Duration	Requests Made	Peak RPS	Avg Response Time	Http Failures
10	1m 30s	389	9.67 reqs/s	266 ms	0 reqs
20	1m 30s	773	16.67 reqs/s	256 ms	0 reqs
30	1m 30s	1163	25 reqs/s	255 ms	0 reqs
40	1m 30s	1554	32.67 reqs/s	250 ms	0 reqs
50	1m 30s	1930	42.33 reqs/s	260 ms	0 reqs

Bảng 6.1: Kết quả kiểm thử

Nhìn vào bảng số liệu trên, ta thấy được hệ thống hoạt động ổn định với 50 user truy cập đồng thời cùng với số lượng request là 1930 mà không xảy ra lỗi. Bước đầu hệ thống có những kết quả tương đối tốt, nhưng sẽ cần phải có những cải thiện hơn nữa cũng như kiểm thử hệ thống với nhiều user hơn để có thể mở rộng hệ thống.

Tổng kết

7.1 Kết quả đạt được

Sau quá trình nỗ lực tìm hiểu, nghiên cứu và thực hiện đề tài "Hệ thống vận chuyển hàng hóa liên tỉnh" thì nhóm đã đạt được một số kết quả đáng khích lệ:

- Xây dựng tương đối hoàn chỉnh một hệ thống quản lý vận chuyển hàng hóa liên tỉnh ở cả phía client lẫn server.
- Hoàn thành được hầu hết các chức năng đề ra ban đầu.
- Giao diện thân thiện và dễ sử dụng.
- Kiến trúc hệ thống microservice giúp dễ dàng mở rộng khi có nhu cầu.
- Sử dụng được các công nghệ, công cụ đang phổ biến hiện nay như: ReactJS, Kafka, Message Queue, Cassandra, Redis,...
- Hiểu được nhiều hơn về quá trình triển khai, cấu hình, bảo mật một hệ thống phức tạp.
- Củng cố và rèn luyện các kiến thức được học ở trường về: Software, System design, Network, System, Database,...
- Nâng cao được khả năng làm việc nhóm, phản biện, trình bày ý kiến, quản lý công việc,...

7.2 Hạn chế

Bên cạnh một số kết quả đáng khích lệ, vì điều kiện thời gian cũng như số lượng thành viên có hạn thì đề tài của nhóm vẫn cần phải cải thiện nhiều hơn nữa. Một số hạn chế có thể nêu ra như:

- Chưa hoàn thành toàn bộ các tính năng như đã tìm hiểu và nghiên cứu.
- Chưa được triển khai vào thực tế.
- Chưa triển khai ứng dụng trên nền tảng di động.
- Chưa triển khai tích hợp với các hệ thống khác.

7.3 Hướng phát triển tương lai

Với những kết quả đã đạt được về "Hệ thống vận chuyển liên tỉnh" thì nhóm tin rằng hệ thống vẫn còn có thể mở rộng hơn nữa theo một số hướng sau:

- Thêm một số chức năng liên quan đến Khuyến mãi cho người dùng.
- Thêm chức năng liên quan đến Chat, Hỗ trợ online.
- Triển khai thêm việc vận chuyển trong nội tỉnh.
- Xây dựng ứng dụng trên nền tảng di động.
- Tích hợp với các hệ thống bán hàng khác.

Tài liệu tham khảo

- [1] Giao hàng nhanh - xem 05/10/2020
<https://giaohangtietkiem.vn/>
- [2] Giao hàng tiết kiệm - xem 05/10/2020
<https://ghn.vn/>
- [3] SPA vs MPA - xem 05/10/2020
<https://www.mindk.com/blog/single-page-applications-the-definitive-guide/>
- [4] ReactJS - xem 06/10/2020
<https://reactjs.org/>
- [5] ReactJS benefits - xem 6/10/2020
<https://www.cloudways.com/blog/why-reactjs-for-front-end/>
- [6] Gitlab CICD - xem 10/10/2020
<https://git.metabarcoding.org/help/ci/README.md>
- [7] What is a REST API? - xem 15/10/2020
<https://www.redhat.com/en/topics/api/what-is-a-rest-api>
- [8] Microservices là gì - xem 20/10/2020
<https://viblo.asia/p/gioi-thieu-ve-kien-truc-microservices-4P8566035Y3>
- [9] SPRING BOOT, SPRING CLOUD - xem 20/10/2020
<https://topdev.vn/blog/building-microservices-application-phan-1-su-dung-netflix-eureka-ribbon-va-zuul/>
- [10] JWT - xem 20/10/2020
<https://techmaster.vn/posts/33959/khai-niem-ve-json-web-token>

- [11] MySQL document - xem 7/11/2020
<https://www.mysql.com/>
- [12] Cassandra document - xem 15/11/2020
<https://cassandra.apache.org/>
- [13] Redis document - xem 8/12/2020
<https://redis.io/>
- [14] Cache Strategy - xem 8/12/2020
<https://bluzelle.com/blog/things-you-should-know-about-database-caching>
- [15] ActiveMQ document - xem 20/12/2020
<https://activemq.apache.org/>
- [16] Message Queue - xem 20/12/2020
<https://viblo.asia/p/microservice-tai-sao-lai-su-dung-message-queue-maGK734AKj2>
- [17] Kafka document - xem 5/1/2021
<https://kafka.apache.org/>
- [18] Cùng nhau học Docker - xem 16/04/2021
<https://viblo.asia/s/2018-cung-nhau-hoc-docker-Wj530mjb56m>
- [19] NGINX - xem 20/05/2021
<https://www.nginx.com/>
- [20] Reverse Proxy - xem 20/05/2021
<https://www.nginx.com/resources/glossary/reverse-proxy-server/>
- [21] Session/Cookies Authen - xem 01/06/2021
<https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4>
- [22] JWT Authen - xem 01/06/2021
<https://www.freecodecamp.org/news/how-to-setup-jwt-authorization-and-authentication-in-spring/>
- [23] HyperLogLog - xem 07/06/2021
<https://en.wikipedia.org/wiki/HyperLogLog#:~:text=HyperLogLog>