

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



**ĐỀ CƯƠNG 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. Đỗ Thanh Thái 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 12, 2020

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 1 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 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	Tổng quan và mô tả nghiệp vụ	1
2	Lược đồ ERD	3
3	Sơ đồ Use case	4
3.1	Đặc tả Use case	5
3.1.1	Trạng thái đơn hàng	5
3.1.2	Người gửi	6
3.1.3	Người nhận	8
3.1.4	Tài xế nội tỉnh đi lấy	8
3.1.5	Tài xế nội tỉnh đi giao	10
3.1.6	Tài xế liên tỉnh	10
3.1.7	Hệ thống phân phối	10
3.1.8	Quản lí	11
3.1.9	Thủ kho	11
3.1.10	Hệ thống thông báo	14
4	Công nghệ sử dụng	15
4.1	Front-end	15
4.1.1	SPA	15
4.1.2	ReactJS	15
4.1.3	Continuous integration / Continuous deployment (CI/CD)	17
4.1.4	Những thư viện nổi bật giúp hiện thực chức năng hệ thống	18
4.1.5	Giám sát lỗi hệ thống bằng dịch vụ Sentry	19
4.2	Back-end	21
4.2.1	Giới thiệu kiến trúc Microservices	21
4.2.2	Kiến trúc Microservices	23
4.2.3	Công nghệ tổng quan	27

4.2.4	Kiến trúc RESTful API	32
4.2.5	Công nghệ gRPC	33
4.2.6	MySQL	35
4.2.7	Cassandra	35
4.2.8	Redis	37
4.2.9	ActiveMQ	38
4.2.10	Kafka	41
5	Kiểm thử hệ thống	44
5.1	Kiểm thử API	44
5.2	Kiểm thử chức năng	44
5.3	Kiểm thử phi chức năng	45
6	Kết quả hiện thực	48
6.1	Chức năng đăng nhập, đăng ký	48
6.1.1	Đăng kí	48
6.1.2	Đăng nhập	49
6.2	Chức năng dành cho người gửi	49
6.2.1	Lên yêu cầu gửi hàng	49
6.2.2	Xem danh sách yêu cầu đã tạo	50
6.2.3	Xem danh sách các đơn hàng nhỏ được tách từ yêu cầu	51
6.2.4	Chỉnh sửa thông tin của người gửi	52
6.3	Chức năng dành cho tài xế	52
6.3.1	Tìm những yêu cầu được gán	52
6.3.2	Xem thông tin đơn hàng đang đến lấy	53
6.3.3	Xem thông tin các đơn hàng đang ở trong xe	54
6.3.4	Xem lịch sử giao nhận hàng	55
6.4	Chức năng dành cho thủ kho	56

Danh sách hình vẽ

1.1	Luồng cơ bản giao nhận hàng hóa	2
2.1	Lược đồ ERD	3
3.1	Sơ đồ Use case	5
3.2	Giao diện tạo đơn hàng (dự kiến)	7
3.3	Giao diện xem danh sách các đơn hàng (dự kiến)	8
3.4	Giao diện theo dõi tình trạng đơn hàng (dự kiến)	9
3.5	Wireframe list các kiện hàng cần lấy của tài xế (dự kiến)	9
3.6	Wireframe miêu tả qui trình xác nhận thông tin sai sót	10
3.7	Wireframe xem danh sách hàng trong kho của thủ kho	12
3.8	Wireframe xác nhận kiện hàng vào kho	13
3.9	Wireframe xác nhận đơn hàng vào kho	13
3.10	Ví dụ mail thông báo dự kiến khi đơn hàng chuyển trạng thái	14
4.1	Luồng dữ liệu của các trang web MPA truyền thống	16
4.2	Luồng dữ liệu của các trang web SPA	16
4.3	ReactJS và những lợi ích khi sử dụng	16
4.4	TypeScript là phiên bản mở rộng của JS có hỗ trợ kiểm tra kiểu tĩnh	17
4.5	Gitlab CICD	18
4.6	Giao diện dashboard để xem những lỗi được gửi về	20
4.7	Giao diện xem lỗi gửi về chi tiết (Nửa trên màn hình)	20
4.8	Giao diện xem lỗi gửi về chi tiết (Nửa dưới màn hình)	21
4.9	Mô hình Microservices	25
4.10	Mô hình services	25
4.11	Công nghệ sử dụng	27
4.12	Canvas	31
4.13	Dev Tool	32
4.14	gRPC	34

4.15 Active MQ	39
4.16 Point-to-point	40
4.17 Publisher-Subscriber	40
4.18 Vai trò Kafka	42
4.19 Communicate giữa các hệ thống	42
4.20 Cấu trúc Kafka	43
5.1 Kiểm thử với 10 users đồng thời	45
5.2 Kiểm thử với 10 users đồng thời	45
5.3 Kiểm thử với 20 users đồng thời	46
5.4 Kiểm thử với 30 users đồng thời	46
5.5 Kiểm thử với 40 users đồng thời	46
5.6 Kiểm thử với 50 users đồng thời	47
6.1 Giao diện đăng ký dành cho người muốn gửi đơn hàng	48
6.2 Giao diện đăng nhập dành cho actor của hệ thống	49
6.3 Giao diện điền thông tin để lên yêu cầu gửi hàng	49
6.4 Thông tin xác nhận yêu cầu muốn tạo	50
6.5 Giao diện list yêu cầu của người gửi (Rỗng)	50
6.6 Giao diện list yêu cầu của người gửi (Có yêu cầu)	51
6.7 Giao diện xem danh sách các đơn hàng nhỏ được tách từ yêu cầu	51
6.8 Giao diện chỉnh sửa thông tin của người gửi	52
6.9 Giao diện tìm những yêu cầu được gán	52
6.10 Giao diện xác nhận lại thông tin yêu cầu muốn nhận	53
6.11 Giao diện xem thông tin đơn hàng đang đến lấy	53
6.12 Giao diện xác nhận lại thông tin đơn hàng đang muôn nhận	54
6.13 Giao diện xem thông tin các đơn hàng đang ở trong xe	54
6.14 Giao diện báo cáo vấn đề ghi gấp sự cố	55
6.15 Giao diện xem lịch sử giao nhận hàng	55
6.16 Giao diện nhập ID của tài xế để nhập các đơn hàng trong xe tài xế vào kho	56
6.17 Xác nhận nhập hàng vào kho	56
6.18 Nhập hàng thành công và cho thủ kho có thẻ tài phiếu nhập kho	57
6.19 Danh sách các đơn hàng trong kho	57
6.20 Lịch sử nhập/xuất kho	58
6.21 Thông tin chi tiết của một lần nhập/xuất đơn hàng (Cho phép tải pdf về)	58
6.22 Format mẫu của 1 đơn nhập/xuất kho	59
6.23 Nhập ID của khách hàng để nhập đơn vào kho	59
6.24 Cho phép nhập khối lượng mà khách muốn nhập vào kho	60

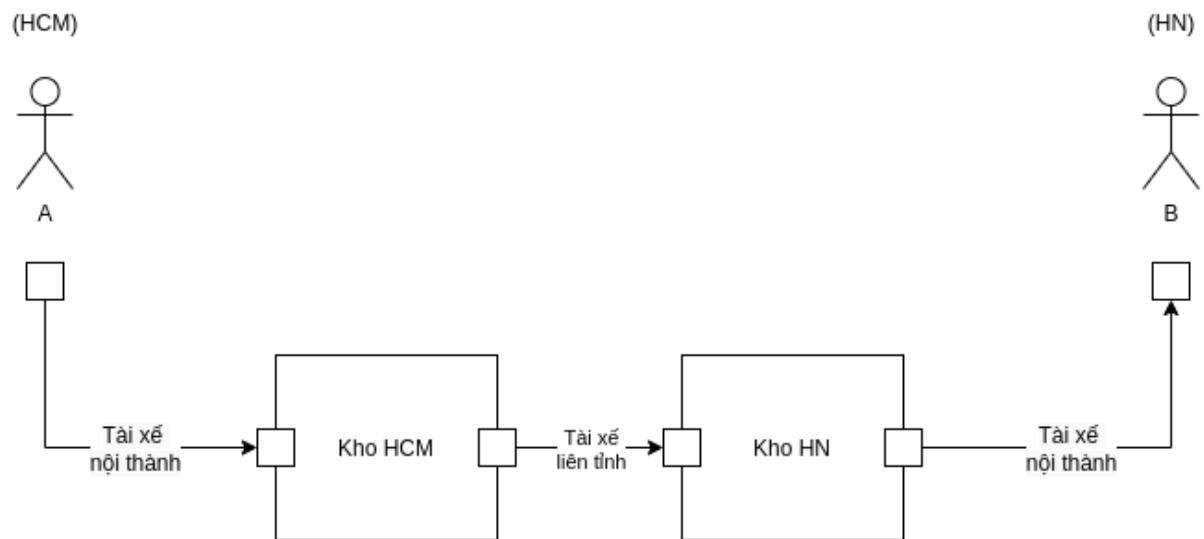
Tổng quan và mô tả nghiệp vụ

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 bưu điện. 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, Giao hàng tiết kiệm vv... và ở ngoài nước là FedEx hay DHL. Những hệ thống hỗ trợ tự động hóa qui 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.

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, thủ 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 yếu 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 đi lấy hàng đến khi đầy xe.

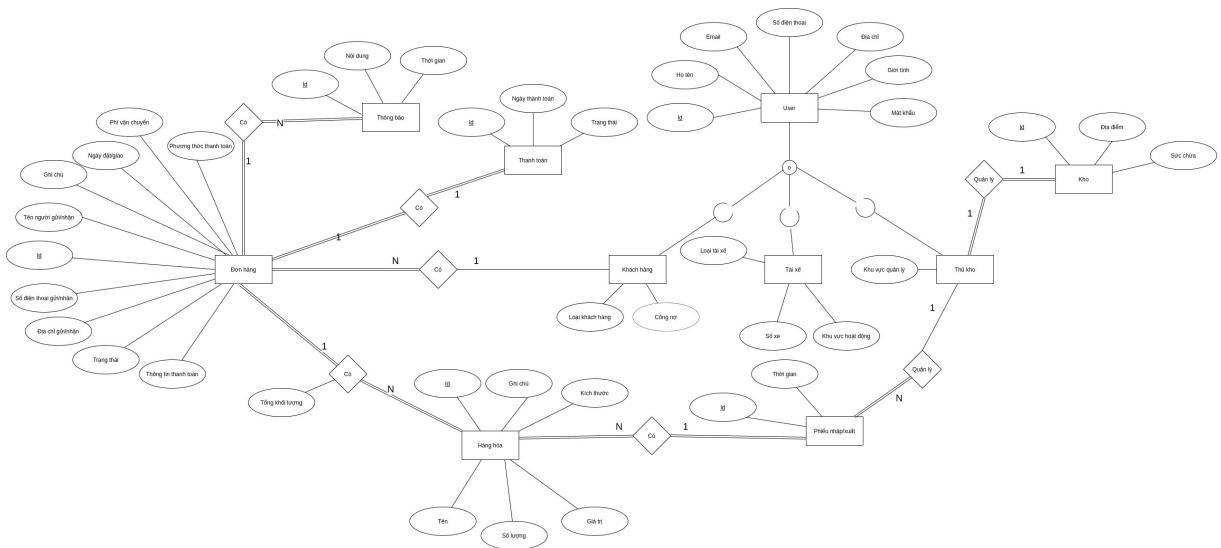
Sau đó, tài xế sẽ trở lại kho tập trung và gỡ hàng ra đặt trong kho. Quá trình này sẽ được lặp lại cho đến khi tổng hàng hóa thu gom về kho có thể chất đầy một xe container để tiến hành vận chuyển những kiện hàng đó đi liên tỉnh. Xe container đầy sẽ đi xuyên tỉnh và đến kho tập trung ở tỉnh ấy, gỡ hàng 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 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

2

Lược đồ ERD



Hình 2.1: Lược đồ ERD

3

Sơ đồ Use case



Hình 3.1: Sơ đồ Use case

3.1 Đặc tả Use case

• 3.1.1 Trạng thái đơn hàng

- Đơn nhập: Trạng thái đầu, khi người dùng tạo yêu cầu đơn hàng

- Chờ bàn giao: Khi người dùng ấn nút gửi yêu cầu. Cho phép tối đa 30s để người dùng có thể chỉnh sửa hoặc hủy đơn (Có thể nhấn nút submit để chốt sớm hơn). Sau thời gian đó đơn hàng sẽ chuyển status sang trạng thái kế tiếp
- Trong hàng chờ phân phối: Dựa vào hàng chờ của hệ thống phân phối (Chi tiết ghi trong actor "Hệ thống phân phối")
- Đang lấy hàng: Các kiện hàng của đơn hàng đã và đang được đến lấy về kho (Có thông tin thêm là <x / tổng số kiện hàng> đã được lấy).
- Đã vào kho tập trung <Địa điểm 1>: Các kiện hàng của đơn hàng đã được lấy hết và được đưa vào kho <Địa điểm 1>
- Đang giao liên tỉnh: Đơn hàng đã xuất kho 1 và đang được giao liên tỉnh từ kho 1 đến kho 2
- Đã vào kho tập trung <Địa điểm 2>: Các kiện hàng của đơn hàng đã được lấy hết và được đưa vào kho <Địa điểm 2>
- Đang giao hàng: Các kiện hàng của đơn hàng đã và đang được giao đến người nhận cuối
- Hoàn tất: Khi tất cả các kiện hàng của đơn hàng đã đến địa điểm cuối
- Đơn hủy: Trạng thái khi khách hàng hủy đơn ở trạng thái chờ bàn giao (Khách hàng chỉ được hủy ở trạng thái chờ bàn giao).

3.1.2 Người gửi

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

- * Bên gửi:
 - Tên người gửi
 - Số điện thoại
 - Địa chỉ, chia làm 3 phần gồm Số địa chỉ + tên đường, Quận - Huyện, Phường - Xã. Quận - Huyện và Phường - Xã được nhập bằng cách chọn trong một dropdown có sẵn.
- * Bên nhận:
 - Tên người nhận
 - Số điện thoại
 - Địa chỉ, tương tự như bên gửi.
- * Thông tin về (các) món hàng muốn gửi:
 - Tên sản phẩm + số lượng (Mặc định là 1). Có thể thêm nhiều sản phẩm và có thể chỉnh sửa số lượng của từng sản phẩm đã thêm.
 - 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.

- * Lựa chọn cho bên nhận:
 - Không cho xem hàng.
 - Cho xem hàng nhưng không cho thử.
 - Cho thử hàng.
- * 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).
- * Mockup / Wireframe (Dự kiến tham khảo thiết kế UI của GHN):

Bên gửi

0944744802 - 0944744802
5 đồng da phường 2 quận bình thạnh tp.hcm

Bên nhận

Số điện thoại
0944744801

Họ tên
Nguyễn Văn A

Địa chỉ

Thanh Hà, Thanh Hà, Hải Dương, Việt Nam

Quận - Huyện
Huyện Thuận Bắc - Ninh Thuận

Phường - Xã
Xã Phước Kháng

Hàng hóa

Loại hàng - Số lượng 15
Lúa

Tổng khối lượng ước tính (kg)
1.500

Kích thước (cm)
10 10 10

Khối lượng quy đổi là gì?
10

Lưu ý - Ghi chú

Lưu ý giao hàng
Không cho xem hàng

Ghi chú
Ví dụ: Lấy sản phẩm 12 cái, lấy sản phẩm 21 cái

Tổng phí

Bên gửi trả phí
53.600 vnđ
Bên gửi trả phí - Chưa tính tiền thu hộ

Xóa đơn Tạo đơn

Hình 3.2: Giao diện tạo đơn hàng (dự kiến)

- **Chỉnh sửa thông tin đơn hàng:** Nếu vẫn đang trong quá trình bàn giao và xác nhận (trong trạng thái **Chờ bàn giao**), người gửi có thể chỉnh sửa thông tin đơn hàng (Thông tin như trong mục tạo đơn, sẽ có giới hạn thông tin được chỉnh sửa).
- **Hủy đơn hàng:** Nếu vẫn đang trong quá trình bàn giao và xác nhận (trong trạng thái **Chờ bàn giao**), người gửi có thể hủy đơn hàng. Đơn hàng bị hủy sẽ được chuyển vào mục thùng rác và cho phép người dùng tái sử dụng thông tin của đơn hàng bị hủy ấy (Tạo lại đơn hàng mới dựa trên thông tin đơn hàng bị xóa ấy).

- **Xem danh sách các đơn hàng:** Các đơn hàng sẽ được chia theo trạng thái đã định nghĩa ở trên và người gửi có thể xem list các đơn hàng lọc theo trạng thái đó. Mockup / Wireframe (Dự kiến tham khảo thiết kế UI của GHN):



Hình 3.3: Giao diện xem danh sách các đơn hàng (dự kiến)

- **Xem nợ công và thanh toán:** Do phí có thể được tính cho người gửi hoặc người nhận (theo nhu cầu của người gửi), người gửi và người nhận đều có thể xem nợ công và thực hiện thanh toán. Thanh toán dự định sẽ được thực hiện qua ZaloPay

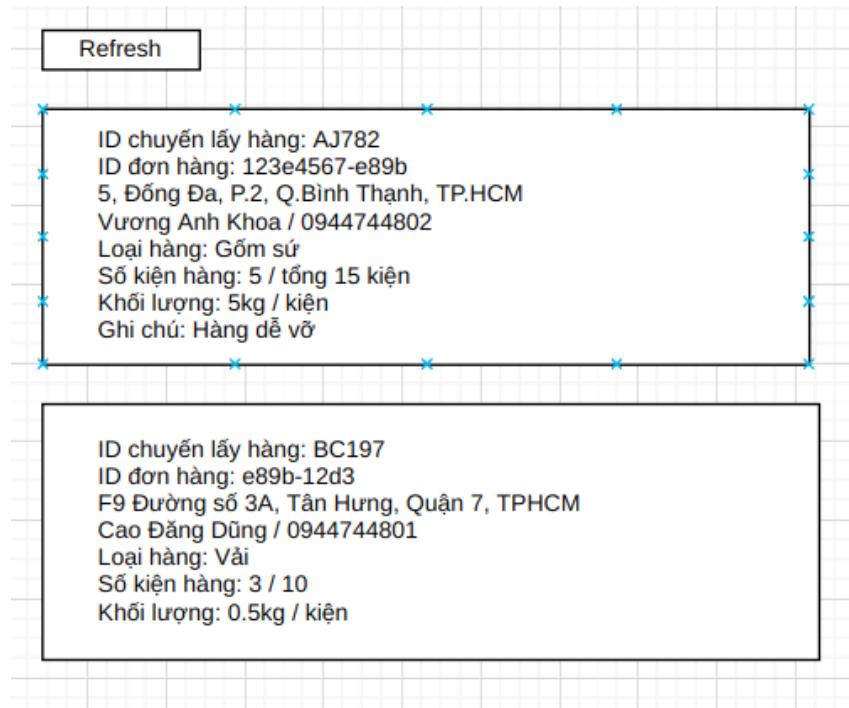
3.1.3 Người nhận

- **Xem nợ công và thanh toán:** Tương tự UC của người gửi
- **Tra cứu thông tin đơn hàng:** Mỗi đơn hàng được tạo sẽ có một mã riêng biệt dùng cho người gửi/nhận tra cứu thông tin và tình trạng hiện tại của đơn hàng. Mockup / Wireframe (Dự kiến tham khảo thiết kế UI của GHN):
- **Xác nhận đơn hàng đã nhận:** Người nhận sẽ check vào những kiện hàng sẽ được nhận (trong đơn hàng của người nhận) có trong xe hoặc trong kho (nếu lựa là người nhận tự đến lấy). Nếu tất cả kiện hàng ở trong đơn hàng đã được check thì trạng thái đơn hàng cập nhật từ (Đang giao hàng -> Hoàn tất). Trong trường hợp nếu người nhận tự đến lấy thì sẽ có thêm khoản phí lưu kho (Được tính theo khoảng thời gian từ lúc đơn hàng nhập kho đến khi người nhận đến lấy).

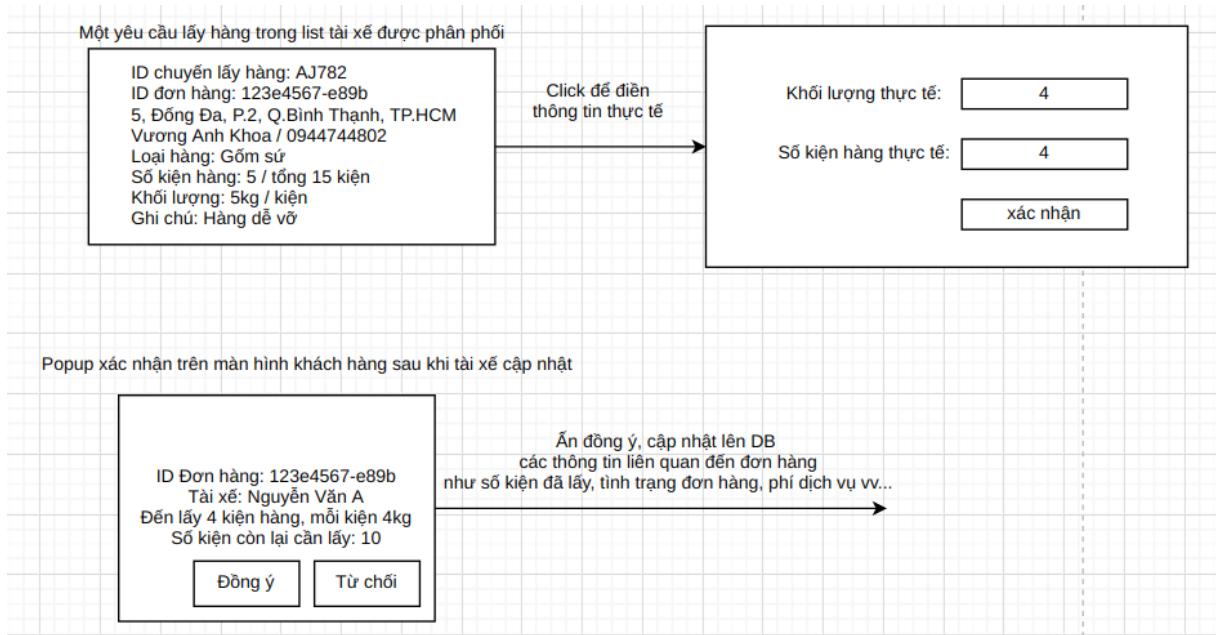
3.1.4 Tài xế nội tỉnh đi lấy

- **Xem danh sách kiện hàng cần lấy:** Tài xế nội tỉnh sẽ được assign những kiện hàng cần đi lấy để thu gom về kho. Tài xế nội tỉnh có thể xem những thông tin cần thiết để tiến hành xử lý vận chuyển list danh sách các kiện hàng đó.
- **Cập nhật nếu có thông tin sai sót:** Tài xế đến lấy hàng sẽ tiến hành đo lại và cập nhật thông tin kiện hàng nếu thông tin kiện hàng có sai sót (Như khối lượng, kích thước vv...). Sau đó, khách hàng sẽ tiến hành xác nhận lại thông tin tài xế cập nhật là chính xác để tiến hành giao nhận.

Hình 3.4: Giao diện theo dõi tình trạng đơn hàng (dự kiến)



Hình 3.5: Wireframe list các kiện hàng cần lấy của tài xế (dự kiến)



Hình 3.6: Wireframe miêu tả qui trình xác nhận thông tin sai sót

3.1.5 Tài xế nội tỉnh đi giao

Xem danh sách kiện hàng cần giao: Tài xế nội tỉnh sẽ được assign những kiện hàng cần được giao cho người nhận. Tài xế nội tỉnh có thể xem những thông tin cần thiết để tiến hành xử lý vận chuyển list danh sách các kiện hàng đó.

3.1.6 Tài xế liên tỉnh

Xem danh sách kiện hàng cần lấy: Tài xế liên tỉnh sẽ được assign những đơn hàng cần đi lấy để vận chuyển đến kho liên tỉnh. Tài xế liên tỉnh có thể xem những thông tin cần thiết để tiến hành xử lý vận chuyển list danh sách các đơn hàng đó.

3.1.7 Hệ thống phân phối

- Phân phối đơn hàng cho tài xế nội tỉnh:** Các đơn hàng sẽ được đưa vào 2 loại hàng chờ (nội tỉnh): Hàng chờ hôm nay và hàng chờ các đơn hàng cho ngày hôm sau. Hệ thống qui định những kiện hàng được tạo trước <6h> tối sẽ được đưa vào hàng chờ hôm nay, còn những kiện hàng được tạo sau <6h> tối sẽ được đưa vào hàng chờ ngày hôm sau. Hệ thống sẽ chỉ phân phối các kiện hàng trong hàng chờ ngày hôm nay cho các tài xế nội tỉnh. Nếu hàng chờ ngày

hôm nay chưa xử lí hết các kiện hàng thì sẽ được đẩy qua ngày hôm sau. Đơn vị ở đây là kiện hàng do xe nội tỉnh thường có kích thước nhỏ, không thể lấy hết cả 1 đơn hàng mà chỉ có thể lấy một số kiện hàng từ đơn ấy. Hệ thống sẽ cập nhật trạng thái từ <Trong hàng chờ phân phôi> -> <Đang lấy hàng>

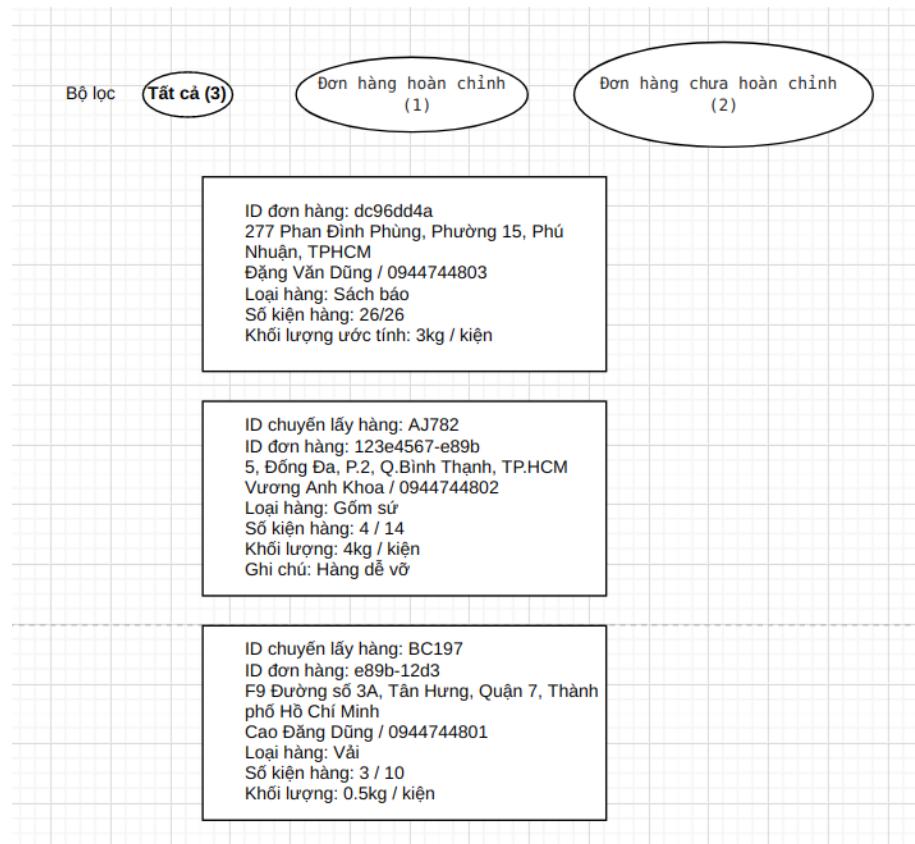
- **Phân phôi đơn hàng cho tài xế liên tỉnh:** Đơn hàng đã được tập kết đầy đủ (Tức toàn bộ kiện hàng của đơn hàng đã đến kho) sẽ được đưa vào hàng chờ liên tỉnh. Hệ thống qui định những đơn hàng đã được tập kết trước <12h> đêm sẽ được đưa vào hàng chờ hôm nay, còn những đơn hàng được tập kết sau <12h> đêm sẽ được đưa vào hàng chờ ngày hôm sau. Hệ thống sẽ chỉ phân phôi các đơn hàng trong hàng chờ ngày hôm nay cho các tài xế liên tỉnh. Nếu hàng chờ ngày hôm nay chưa xử lí hết các đơn hàng thì sẽ được đẩy qua ngày hôm sau. Đơn vị ở đây là đơn hàng do xe liên tỉnh thường có kích thước lớn nên có thể chở cả 1 hoặc nhiều đơn hàng.

3.1.8 Quản lý

Xem thống kê: Người quản lý có thể xem thống kê về số lượng đơn hàng trong tuần, tháng, năm thông qua biểu đồ; xem thống kê hiệu suất làm việc của tài xế ; số đơn hàng giao thành công và đơn hàng bị hủy trong tuần, tháng, năm.

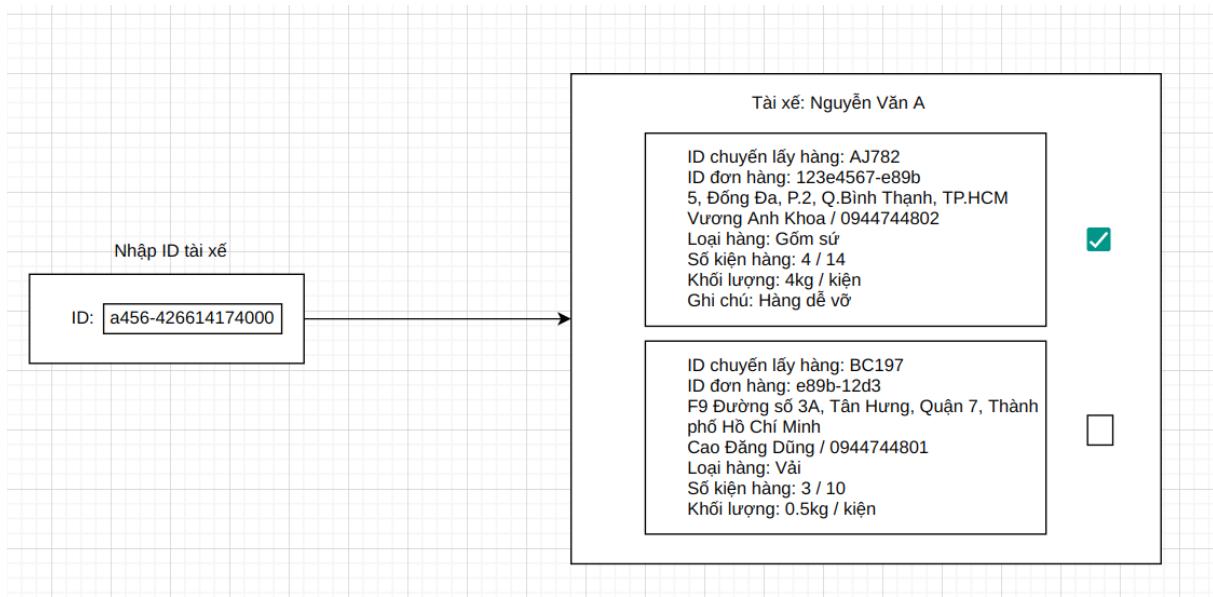
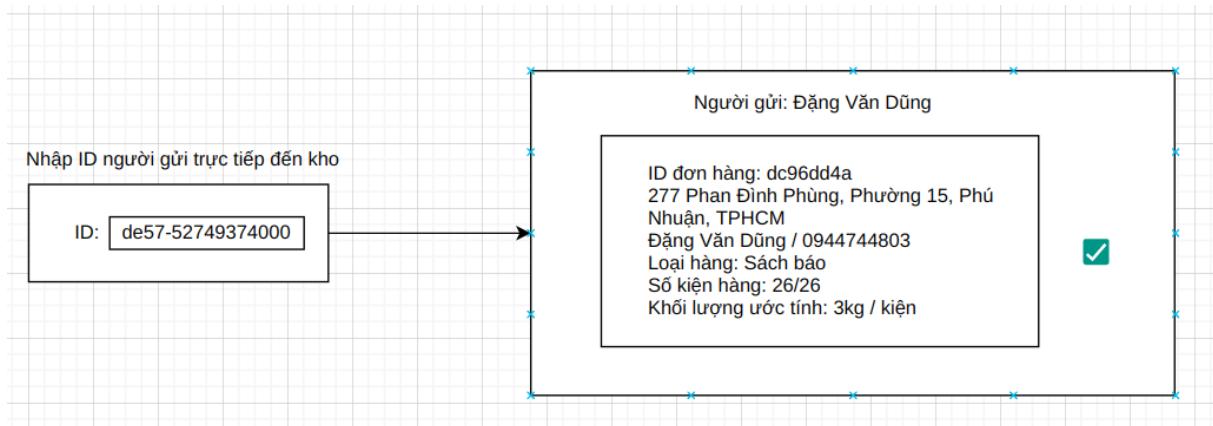
3.1.9 Thủ kho

- **Xem danh sách hàng trong kho:** Thủ kho sẽ xem list các đơn hàng có trong kho (Lọc theo tiêu chí tất cả, đơn hàng mà các kiện đã vào đủ, đơn hàng mà các kiện chưa vào đủ). Thông tin từng đơn hàng là loại hàng, <số kiện hàng đã vào kho / tổng số kiện>.
- **Xác nhận kiện hàng vào kho:** Khi tài xế nội tỉnh đến kho, thủ kho sẽ search ID của tài xế để xem list kiện hàng được phân công của tài xế Thủ kho sẽ kiểm tra xe của tài xế và check vào những kiện hàng thật sự có trong xe để hoàn tất quá trình nhập kho của kiện hàng. (Tức hệ thống sẽ cập nhật số kiện hàng đã vào kho của đơn hàng). Trạng thái đơn hàng cập nhật từ <Đang lấy hàng> -> <Đã vào kho tập trung 1> nếu tất cả các kiện hàng của đơn đã được check vào kho.
- **Xác nhận đơn hàng xuất kho:** Khi tài xế liên tỉnh xuất kho đến yêu cầu thủ kho những đơn hàng cần giao liên tỉnh thì thủ kho sẽ search ID của tài xế liên tỉnh đó và xem được list các đơn hàng mà tài xế liên tỉnh đó được phân phôi. Khi tài xế liên tỉnh chuyển những đơn hàng đó lên xe liên tỉnh thì thủ kho sẽ check vào những kiện hàng sẽ lên xe để hoàn tất quá trình xuất kho của kiện hàng. Việc check như vậy sẽ cập nhật trạng thái của đơn hàng trên hệ thống (Đã vào kho tập trung 1 -> Đang giao liên tỉnh).



Hình 3.7: Wireframe xem danh sách hàng trong kho của thủ kho

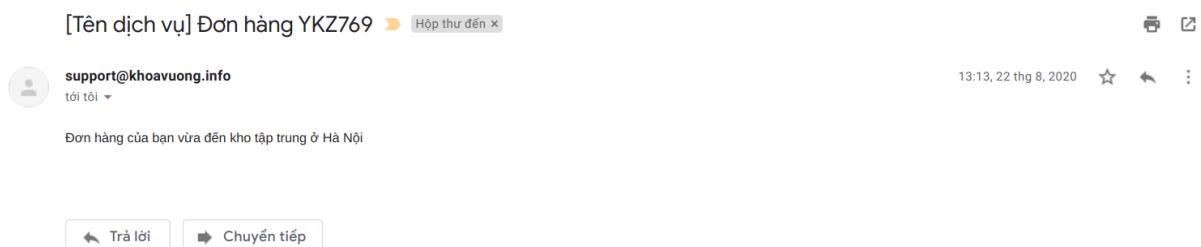
- **Xác nhận đơn hàng vào kho:** Khi tài xế liên tỉnh đến kho, thủ kho sẽ search ID của tài xế liên tỉnh để xem list đơn hàng được phân công của tài xế Thủ kho sẽ kiểm tra xe của tài xế và check vào những đơn hàng thật sự có trong xe để hoàn tất quá trình nhập kho của đơn hàng. Trạng thái đơn hàng cập nhật từ (Đang giao liên tỉnh -> Đã vào kho tập trung 2). Nếu người gửi chọn option tự gửi đến kho, thủ kho sẽ search ID của người gửi và thực hiện tiếp các bước như trên.

**Hình 3.8:** Wireframe xác nhận kiện hàng vào kho**Hình 3.9:** Wireframe xác nhận đơn hàng vào kho

- Xác nhận kiện hàng xuất kho:** Khi tài xế nội tỉnh đến yêu cầu thủ kho những kiện hàng cần giao nội tỉnh thì thủ kho sẽ search ID của tài xế nội tỉnh đó và xem được list các kiện hàng mà tài xế nội tỉnh đó được phân phổi. Khi tài xế nội tỉnh chuyển những kiện hàng đó lên xe nội tỉnh thì thủ kho sẽ check những kiện hàng trong list xem ở trên để xác nhận việc tài xế nội tỉnh đã chất hàng lên xe. Việc check như vậy sẽ cập nhật trạng thái của đơn hàng trên hệ thống (Đã vào kho tập trung 2 -> Đang giao hàng (x/tổng số kiện)).

3.1.10 Hệ thống thông báo

Gửi thông báo đến mail người dùng cho mỗi mức chuyển trạng thái: Hệ thống sẽ gửi mail đến người dùng mỗi khi đơn hàng chuyển qua một trạng thái



Hình 3.10: Ví dụ mail thông báo dự kiến khi đơn hàng chuyển trạng thái

4

Công nghệ sử dụng

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 gặp 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.

4.1 Front-end

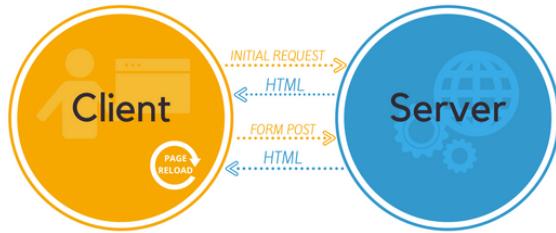
4.1.1 SPA

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 load lại trang nữa. Việc này được thực hiện bằng cách sử dụng AJAX để lấy dữ liệu một cách bất đồng bộ từ phía server 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.

4.1.2 ReactJS

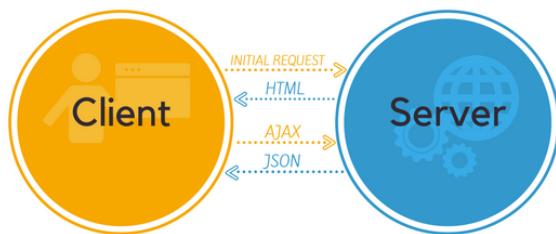
ReactJS được phát triển bởi Facebook là một thư viện rất phổ biến để phát triển các ứng dụng web SPA hiện nay. React Cho phép người lập trình có thể chia ứng dụng thành các component và tái sử dụng, nhờ vậy mà tiết kiệm được thời gian lập trình. Hơn nữa, cơ chế DOM ảo (**Virtual DOM**) mới chính là điểm 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 lý 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 lý ngầm một cây DOM ảo và sẽ đổi chiếu những thay đổi so với các phiên bản DOM

Traditional page lifecycle



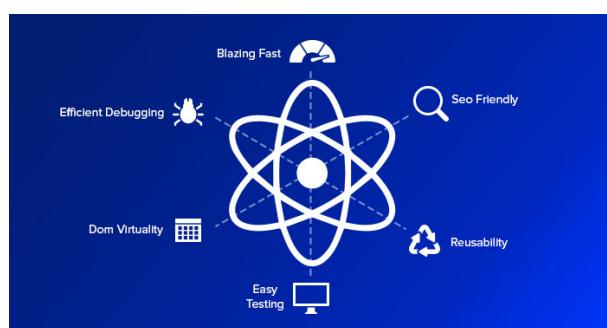
Hình 4.1: Luồng dữ liệu của các trang web MPA truyền thống

SPA lifecycle



Hình 4.2: Luồng dữ liệu của các trang web SPA

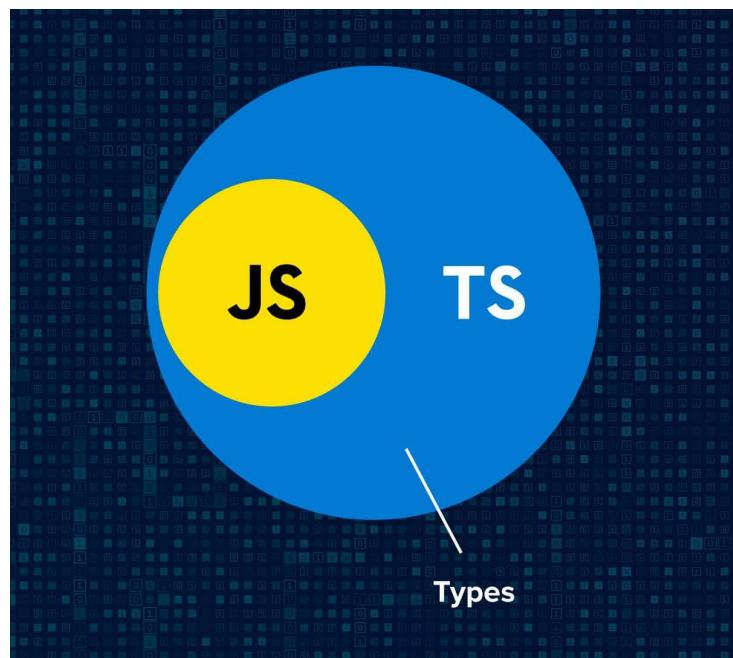
ảo trước đó để cập nhật lên cây DOM thật, giúp ứng dụng web tăng hiệu suất. Chính vì những lí do trên mà nhóm quyết định sẽ sử dụng thư viện này để phát triển luận văn này.



Hình 4.3: ReactJS và những lợi ích khi sử dụng

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. JS 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. JS 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. Vậy 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 JS 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 JS 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 TS sẽ giúp cho mã nguồn của dự án dễ đọc, minh bạch, dễ bảo trì và phát triển hơn rất nhiều.

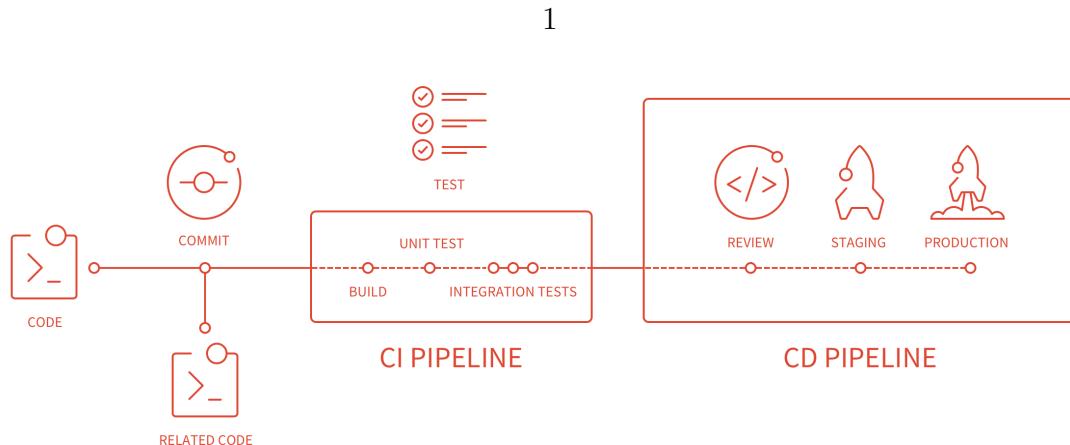


Hình 4.4: TypeScript là phiên bản mở rộng của JS có hỗ trợ kiểm tra kiểu tĩnh

4.1.3 Continuous integration / Continuous deployment (CI/CD)

Do web SPA sẽ trả về file 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 để làm môi trường test (staging). Ưu điểm của gitlab là vừa có thể giúp nhóm quản lý source code 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.



Hình 4.5: Gitlab CICD

4.1.4 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:

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

4.1.5 Giám sát lỗi hệ thống bằng dịch vụ Sentry

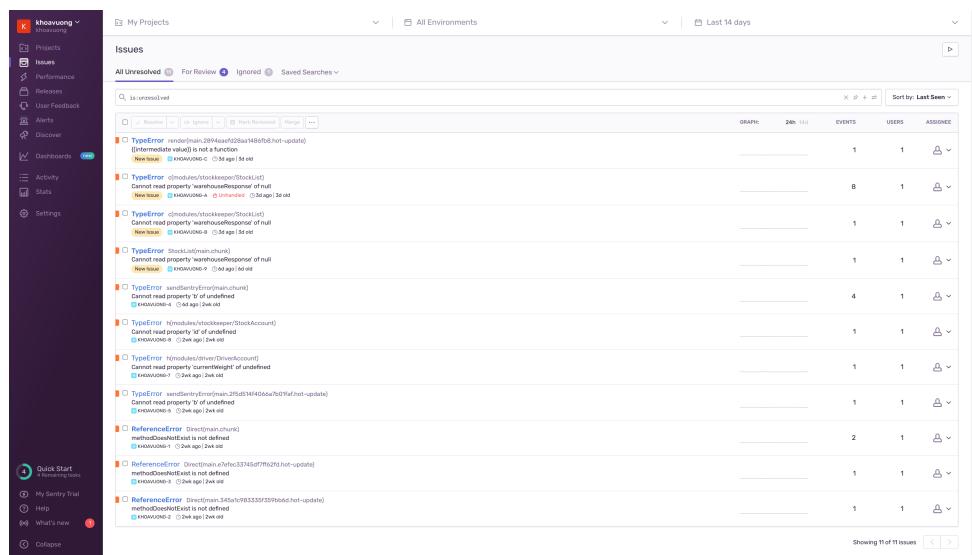
Giới thiệu về Sentry

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 report lại cho bên phía đội ngũ phát triển. Sentry sẽ tự động gửi report 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.

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:

Áp dụng sentry cho hệ thống giao hàng liên tỉnh

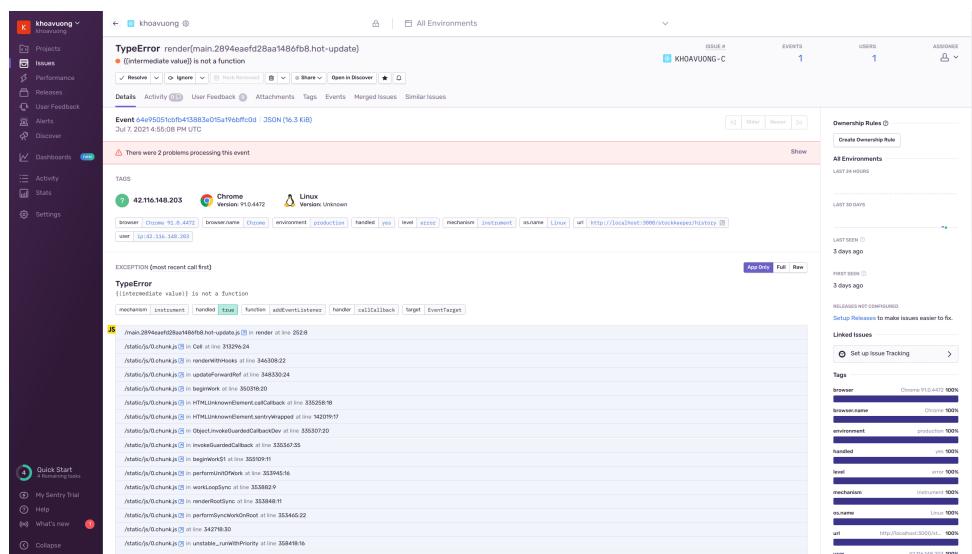
Nhận thấy tác dụng hữu ích và đồng thời muốn học thêm những công nghệ đang thịnh hành ngoài thị trường. Nhóm chúng em đã thử áp dụng Sentry vào hệ thống giao hàng liên tỉnh này. Sau đây là một số kết quả mà nhóm đã thu được từ việc áp dụng Sentry:



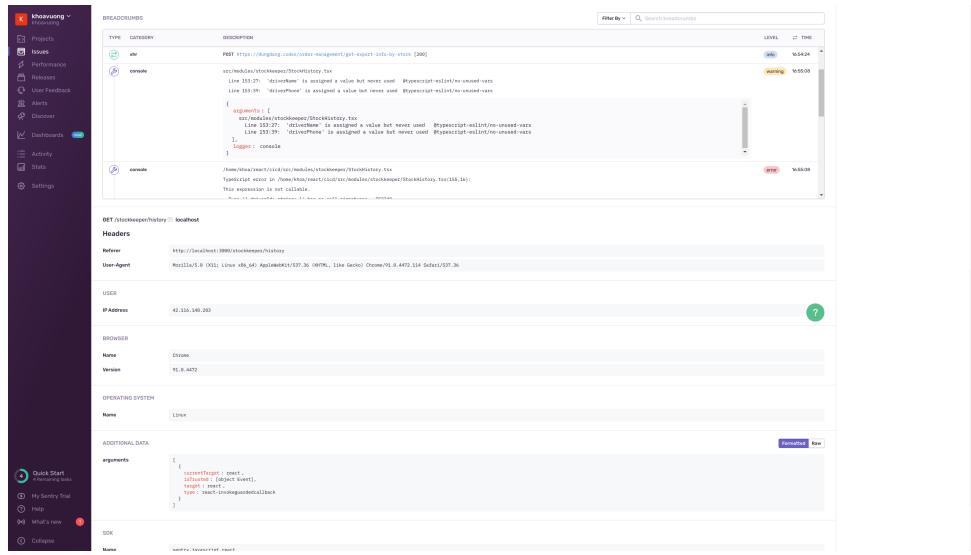
Hình 4.6: Giao diện dashboard để xem những lỗi được gửi về

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:



Hình 4.7: Giao diện xem lỗi gửi về chi tiết (Nửa trên màn hình)



Hình 4.8: 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.

4.2 Back-end

4.2.1 Giới thiệu kiến trúc Microservices

Trước khi Microservices 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 technology stack 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 scale một cách độc lập với nhau. Việc scale 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, bạn 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.
- 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.

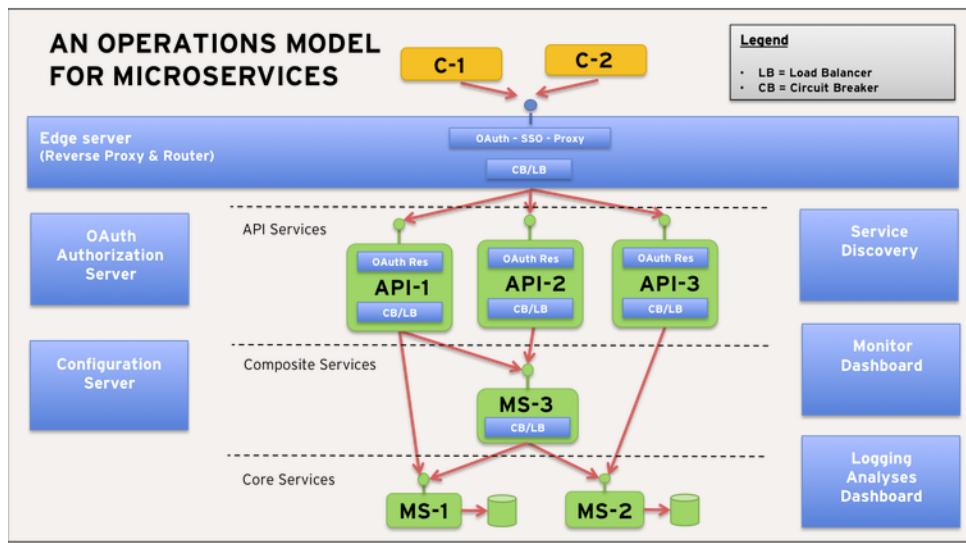
4.2.2 Kiến trúc Microservices

Các thành phần chính dự kiến xây dựng hệ thống trong Microservices

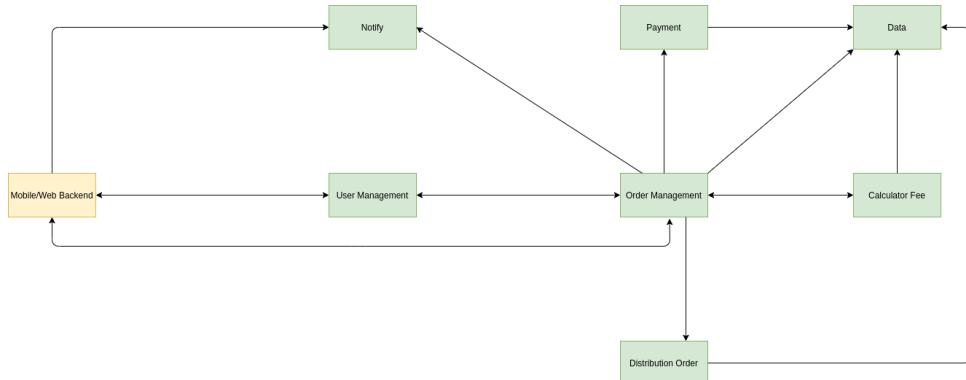
- **Edge Server** - Để đưa các API services ra bên ngoài và để ngăn chặn truy cập trái phép vào các microservices nội bộ, chúng ta cần một edge server nối tất cả request bên ngoài đi qua. Một edge server có thể tái sử dụng khả năng định tuyến động và cân bằng tải dựa trên service discovery được mô tả ở trên. Edge server sẽ hoạt động như một proxy ngược chủ động mà không cần cập nhật thủ công khi hệ thống nội bộ thay đổi.
- **Dynamic Routing and Load Balancer** - Với chức năng service discovery, các thành phần định tuyến có thể sử dụng discovery API để tra cứu nơi mà microservice được yêu cầu được triển khai và các thành phần cân bằng tải có thể quyết định định tuyến yêu cầu tới instance nào nếu nhiều instance được triển khai cho một service được yêu cầu.
- **Service Discovery** - Thay vì theo dõi thủ công những microservices nào được triển khai hiện tại và trên máy chủ và cổng nào, chúng ta cần chức năng Service Discovery cho phép microservices tự đăng ký khi khởi động thông qua API.
- **Circuit Breaker** - Để tránh chuỗi sự cố, cần phải áp dụng Circuit Breaker pattern. Đây là một pattern dùng để ngắt một quá trình xử lý khi hệ thống gặp sự cố, để đảm bảo số lượng message bị lỗi không tăng cao, làm cho việc khắc phục trở nên khó khăn, cũng như có thể làm cho hệ thống bị xụp đổ hàng loạt do ảnh hưởng lẫn nhau.
- **Monitoring** - Vì đã có circuit breakers, chúng ta có thể bắt đầu theo dõi trạng thái của chúng và thu thập số liệu thống kê thời gian chạy từ chúng để có được một bức tranh về tình trạng sức khỏe của hệ thống.
- **Secure API** - Để bảo vệ các API services được expose ra bên ngoài sử dụng OAuth 2.0, quy trình của OAuth 2.0 có thể như sau:

- Một component mới có thể đóng vai trò như một Máy chủ ủy quyền (OAuth Authorization Server)
 - Các API services sẽ đóng vai trò như các Máy chủ tài nguyên (OAuth Resource Server)
 - Các Client bên ngoài gọi đến API services với tư cách là OAuth Clients
 - Edge server sẽ làm việc như một OAuth Token Relay: nó sẽ hoạt động như một OAuth Resource Server, Nó sẽ chuyển các OAuth Access Tokens có trong request bên ngoài đến các API services
- **Central Configuration Server.** - Thay vì cấu hình cục bộ cho mỗi đơn vị triển khai (tức là microservice), chúng ta cần quản lý cấu hình tập trung. Chúng ta cũng cần một configuration API để các microservices có thể sử dụng để lấy thông tin cấu hình.
 - **Centralized Log Analysis** - Để có thể theo dõi messages và phát hiện khi chúng bị kẹt (stuck), chúng ta cần một chức năng phân tích log tập trung có khả năng tiếp cận với máy chủ và thu thập các tệp log mà mỗi services microservice tạo ra. Chức năng phân tích log lưu trữ thông tin log này trong cơ sở dữ liệu trung tâm và cung cấp khả năng tìm kiếm và dashboard. Lưu ý : Để có thể tìm thấy các messages liên quan, điều quan trọng là tất cả các microservices phải sử dụng id đồng nhất trong các log message.

Mô hình minh họa

**Hình 4.9:** Mô hình Microservices

Các services dự kiến hệ thống của nhóm

**Hình 4.10:** Mô hình services

- **User Management**

- Service chịu trách nhiệm quản lý user trong hệ thống. Ở đây sẽ lưu trữ Database của user
- Mỗi khi user login vào hệ thống thì sẽ vào Service này để authen: Ban đầu user được yêu cầu sẽ nhập username và password, Sau khi authen là đúng user đó thì hệ thống sẽ sinh ra JWT (Json Web Token) để gửi trả về cho user. Sau này

cứ mỗi request lên hệ thống thì user sẽ gửi kèm theo JWT để hệ thống có thể authen và author.

- User có thể get thông tin để có thể xem và chỉnh sửa

- **Order Management**

- Service chịu trách nhiệm tạo Order cho user. Ở đây sẽ lưu trữ database về order.
- Mỗi khi có request lên thì service đi authen sau đó gọi qua service Calculator Fee để tính toán các chi phí rồi lưu Order đó vào DB với trạng thái là “Đơn nhập”. Sau đó sẽ gửi thông tin về Order cho user xác nhận. Sau khi nhận được request xác nhận của user thì trạng thái của đơn hàng chuyển thành “Chờ bàn giao”.

- **Distribution Order**

- Service chịu trách nhiệm chia nhỏ các đơn hàng theo từng mặt hàng và chạy giải thuật để phân phối các đơn hàng cho tài xế.
- Ở đây lưu trữ Database về thông tin của đơn hàng sau khi được chia nhỏ thành các mặt hàng.

- **Notify**

- Service chịu trách nhiệm quản lý các thông báo.
- Mobile/Web Backend sẽ liên tục gọi API getNotify để thông báo cho các user mỗi khi cần xác nhận bàn giao hàng
- Ngoài ra mỗi khi trạng thái đơn hàng thay đổi thì hệ thống sẽ tự động gửi mail thông báo cho các user

- **Calculator Fee**

- Service chịu trách nhiệm tính toán các khoản phí phát sinh
- Database lưu thông tin về phí của mỗi đơn hàng để các user có thể đối soát

- **Payment**

- Service này sẽ liên kết với 1 bên thứ 3 như: ngân hàng, ví điện tử, để thực hiện chức năng thanh toán cho các user

- **Data**

- Các service khác sẽ gửi data qua service này phục vụ cho việc tạo dashboard thống kê các hoạt động theo tuần, tháng, năm. Ngoài ra còn cung cấp data để team dev có thể monitor các service khác hoạt động như thế nào.

4.2.3 Công nghệ tổng quan

Bảng bên dưới là công nghệ dự tính thực hiện của nhóm:

Operations Component	Netflix, Spring, ELK
Service Discovery server	Netflix Eureka
Dynamic Routing and Load Balancer	Netflix Ribbon
Circuit Breaker	Netflix Hystrix
Monitoring	Netflix Hystrix dashboard and Turbine
Edge Server	Netflix Zuul
Central Configuration server	Spring Cloud Config Server
OAuth 2.0 protected API's	Spring Cloud + Spring Security OAuth2
Centralised log analyses	Logstash, Elasticsearch, Kibana (ELK)

Hình 4.11: Công nghệ sử dụng

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

- **Spring Boot** 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** 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 developer 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 developer, các data center hoặc trên cloud.
- Bản thân hệ sinh thái Spring không tự build hết tất cả các viên gạch cần để xây dựng ứng dụng Microservices. Thay vào đó, có một kho gạch dành cho Microservices được Netflix ban đầu phát triển trong nội bộ của họ để xây dựng dịch vụ xem phim trực tuyến nổi tiếng của họ và sau đó public ra dưới dạng open source với cái tên **Netflix OSS** (Open Source Software). Và Spring đơn giản chỉ làm nhiệm vụ wrap

chúng lại và dùng trong hệ sinh thái của Spring (tất nhiên Spring nó còn wrap nhiều thằng khác nữa). Bằng cách kết hợp Spring Boot, Spring Cloud và Netflix OSS sẽ đủ cung cấp các công cụ và nguyên liệu cần thiết nhất để giúp bạn có thể nhanh chóng và dễ dàng xây dựng các Microservices của mình.

Công nghệ sử dụng cho Edge Server, Dynamic Routing and Load Balancer và Service Discovery

- **Netflix Eureka** – Là một Service Discovery Server cho phép các dịch vụ microservices tự đăng ký mình vào danh sách các services hoạt động lúc khởi chạy. Server này sẽ chịu trách nhiệm lưu trữ và cung cấp thông tin của các service trong một hệ thống Microservice. Khi một service đăng ký thông tin với server, nó sẽ cung cấp các thông tin như host, port, trạng thái của nó. Và nó cũng thường xuyên gửi heartbeat message để thông báo tình trạng của mình cho server. Do đó, server này có thể dễ dàng cung cấp thông tin về các service khi chúng được gọi tới từ các service khác.
- **Netflix Ribbon** – Với chức năng Dynamic Routing và Load Balancing có thể được sử dụng bởi Service client để tra cứu dịch vụ lúc runtime. Ribbon sử dụng thông tin có sẵn trong Eureka để định vị các instance thích hợp. Nếu tìm thấy nhiều hơn một instance, Ribbon sẽ áp dụng cân bằng tải để phân phối các request đến các instance rảnh việc nhất. Ribbon không chạy dưới dạng một dịch vụ riêng biệt mà thay vào đó nó là một thành phần được nhúng trong mỗi Service client.
- **Netflix Zuul** – Đóng vai trò như một Edge Server hoặc gatekeeper với thế giới bên ngoài, không cho phép bất kỳ yêu cầu bên ngoài trái phép nào đi qua. Các request đi tới services đều phải qua anh chàng Zuul này để check hàng trước. Zuul sử dụng Ribbon để tra cứu các dịch vụ sẵn có và định tuyến yêu cầu bên ngoài đến instance dịch vụ thích hợp. Trong bài đăng trên blog này, tôi sẽ chỉ sử dụng Zuul như một điểm vào (entry point).

Công nghệ sử dụng cho Circuit Breaker

- **Netflix Hystrix** – Cung cấp khả năng ngắt mạch cho service consumer. Nếu một dịch vụ không đáp ứng (do timeout hoặc lỗi giao tiếp), Hystrix có thể chuyển hướng cuộc gọi đến phương thức dự phòng trong service consumer. Nếu một dịch vụ liên tục không hoạt động, Hystrix sẽ mở mạch và thực hiện fail fast (tức là gọi phương thức dự phòng mà không gọi đến service đó luôn) trên tất cả request tiếp theo cho đến khi dịch vụ “sống” trở lại. Để xác định khi nào một dịch vụ sống lại, lâu lâu nó sẽ thử bằng cách cho phép các request tới dịch vụ đó để xem nó sống lại hay chưa. Hystrix được cài đặt ở phía service consumer.

Công nghệ sử dụng cho Monitoring

- **Netflix Hystrix dashboard và Netflix Turbine** – Hystrix giúp kiểm soát sự tương tác giữa các dịch vụ bằng cách cung cấp khả năng chịu lỗi và khả năng chịu độ trễ. Nó cải thiện khả năng phục hồi tổng thể của hệ thống bằng cách cô lập các

dịch vụ bị lỗi và ngăn chặn hiệu ứng phân tầng của các lỗi. Và Hystrix dashboard cung cấp trực quan tổng quát bộ ngắt mạch. Turbine là một công cụ mã nguồn mở của Netflix để tổng hợp nhiều luồng từ Hystrix vào một luồng duy nhất.

Công nghệ sử dụng cho Secure API

- **Json Web Token(JWT)** - 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.

Công nghệ sử dụng cho Central Configuration Server

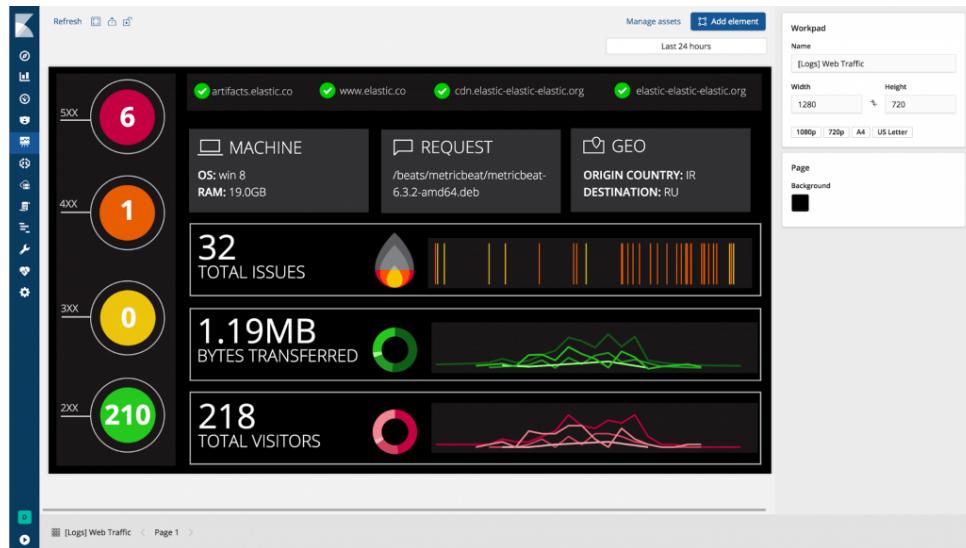
- **Spring Cloud Config** là một mô-đun của Spring Cloud cung cấp việc lưu trữ và phục vụ các cấu hình phân tán trên nhiều ứng dụng và trên các môi trường phát triển (dev, staging, product, ...). Spring Cloud Config hoạt động theo mô hình kiến trúc client - server, bao gồm: Spring Cloud Config Server và Spring Cloud Config Client.
 - **Spring Cloud Config Server:** Các property của các ứng dụng Spring (được lưu trong các property source như file properties hoặc file YAML) được tập trung trên một hệ thống backend: Git repository (mặc định), File System, Vault, JDBC, Redis... Nhiệm vụ của Spring Config Server là pull các property này về và dùng EnvironmentRepository để lưu trữ. EnvironmentRepository cung cấp các đối tượng Spring Environment. Sau đó cung cấp các property cho Config Client thông qua các HTTP resource-based API (HTTP Method là GET).
 - **Spring Cloud Config Client:** Đây chính là các ứng dụng Spring đã tách biệt các property. Khi startup, Config Client sẽ đọc các property từ API của Config Server và khởi tạo đối tượng Environment với property source phù hợp.

Công nghệ sử dụng cho Centralized Log Analysis

- **Logstash** - Có thể thu thập các sự kiện nhật ký từ nhiều loại nguồn bằng cách sử dụng các plug-ins đầu vào, chuyển đổi nó sang định dạng bạn thích bằng cách sử dụng các plug-ins filter và codec và gửi nó đến một số điểm đến bằng các plug-ins đầu ra.
 - **Input:** tiếp nhận/thu thập dữ liệu sự kiện log ở dạng thô từ các nguồn khác nhau như file, redis, rabbitmq, beats, syslog,...
 - **Filter:** Sau khi tiếp nhận dữ liệu sẽ tiến hành thao tác dữ liệu sự kiện log (như thêm, xoá, thay thế,... nội dung log) theo cấu hình của quản trị viên để xây dựng lại cấu trúc dữ liệu log event theo mong muốn.

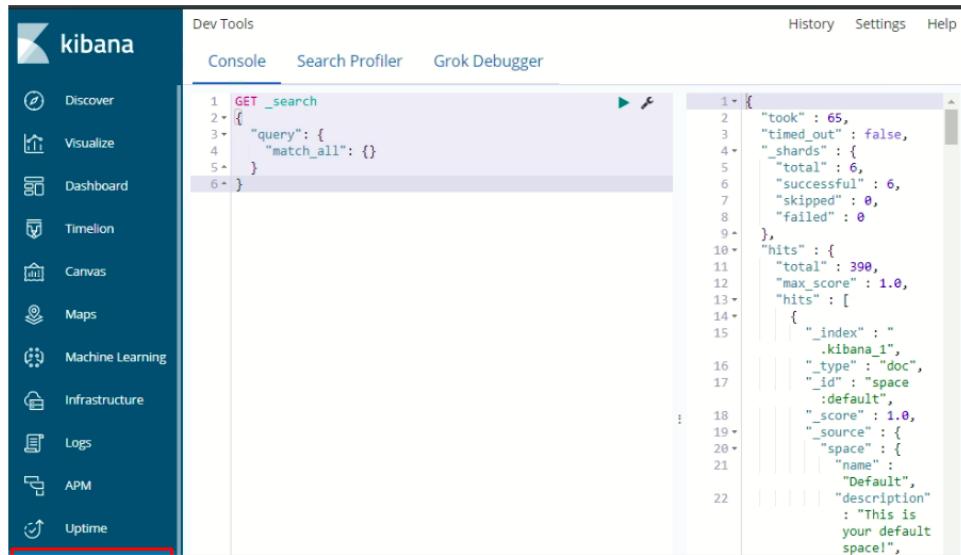
- **Output:** Sau cùng sẽ thực hiện chuyển tiếp dữ liệu sự kiện log về các dịch vụ khác như Elasticsearch tiếp nhận lưu trữ log hoặc hiển thị log,..
- **Elasticsearch** - Là một cơ sở dữ liệu tìm kiếm toàn văn bản được phân phối và có thể mở rộng, cho phép bạn lưu trữ và tìm kiếm khối lượng lớn các sự kiện nhật ký. Thay vì tìm kiếm trên file, trên các database như MySQL, Oracle, MongoDB... thì chuyển dữ liệu đó sang Elasticsearch và thực hiện tìm kiếm trên Elasticsearch sẽ mang lại hiệu quả rất lớn, đặc biệt là trong những trường hợp dữ liệu log lớn.
 - **Ưu điểm**
 - * Tìm kiếm dữ liệu rất nhanh chóng, mạnh mẽ dựa trên Apache Lucene (near-realtime searching)
 - * Có khả năng phân tích dữ liệu
 - * Khả năng mở rộng theo chiều ngang
 - * Hỗ trợ tìm kiếm mờ tức là từ khóa tìm kiếm có thể bị sai lối chính tả hay không đúng cú pháp thì vẫn có khả năng Elasticsearch trả về kết quả tốt.
 - * Hỗ trợ Structured Query DSL (Domain-Specific Language), cung cấp việc đặc tả những câu truy vấn phức tạp một cách cụ thể và rõ ràng bằng JSON.
 - * Hỗ trợ nhiều Elasticsearch client như Java, PHP, Javascript, Ruby, .NET, Python
 - **Nhược điểm**
 - * Trong Elasticsearch không có khái niệm database transaction, tức là nó sẽ không đảm bảo được toàn vẹn dữ liệu trong các hoạt động Insert, Update, Delete. Tức khi chúng ta thực hiện thay đổi nhiều bản ghi nếu xảy ra lỗi thì sẽ làm cho logic của mình bị sai hay dẫn tới mất mát dữ liệu. Đây cũng là 1 phần khiến Elasticsearch không nên là database chính.
 - * Không thích hợp với những hệ thống thường xuyên cập nhật dữ liệu. Sẽ rất tốn kém cho việc đánh index dữ liệu.
 - * Elasticsearch được thiết kế cho mục đích search, do vậy với những nhiệm vụ khác ngoài search như CRUD thì Elasticsearch kém thê hơn so với những database khác như MongoDB, MySQL....
- **Kibana** - Cho phép bạn hình dung và phân tích các sự kiện nhật ký của bạn được lưu trữ trong Elasticsearch.
 - Cung cấp biểu đồ tương tác: Kibana cung cấp các biểu đồ và báo cáo trực quan mà bạn có thể sử dụng để điều hướng tương tác thông qua một lượng lớn dữ liệu. Bạn có thể tự động kéo các cửa sổ thời gian, phóng to, thu nhỏ các tập hợp dữ liệu cụ thể và xem chi tiết các báo cáo để trích xuất thông tin chi tiết có thể hành động từ dữ liệu của bạn.
 - Tập hợp và bộ lọc dữ liệu sẵn: Sử dụng các tập hợp và bộ lọc dữ liệu sẵn của Kibana, bạn có thể chạy nhiều loại phân tích như biểu đồ, truy vấn N hàng đầu và xu hướng.

- Bảng điều khiển dễ dàng truy cập: có thể dễ dàng thiết lập bảng điều khiển và báo cáo và chia sẻ chúng với những người khác. Chỉ cần một trình duyệt để xem và khám phá dữ liệu.
- Công cụ trình bày Canvas: Canvas kết hợp dữ liệu với màu sắc, hình dạng, và trí tưởng tượng để mang lại sự hiển thị dữ liệu động, đa trang, pixel hoàn hảo cho màn hình lớn và nhỏ, làm cho dữ liệu tuyệt vời hơn.



Hình 4.12: Canvas

- Dev Tools: Chứa các công cụ phát triển giúp bạn tương tác với dữ liệu dễ dàng hơn



The screenshot shows the Kibana Dev Tools interface. On the left is a sidebar with various navigation options: Discover, Visualize, Dashboard, Timelion, Canvas, Maps, Machine Learning, Infrastructure, Logs, APM, and Uptime. The 'Uptime' option is highlighted with a red border. The main area is titled 'Dev Tools' and contains three tabs: Console, Search Profiler, and Grok Debugger. The 'Console' tab is selected. It displays a code editor with a query and its execution results. The query is:

```

1 GET _search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }

```

The results are:

```

1 {
2   "took" : 65,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 6,
6     "successful" : 6,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : 398,
12    "max_score" : 1.0,
13    "hits" : [
14      {
15        "_index" : ".kibana_1",
16        "_type" : "doc",
17        "_id" : "space_default",
18        "_score" : 1.0,
19        "_source" : {
20          "space" : {
21            "name" : "Default",
22            "description" : "This is your default space!"
23          }
24        }
25      }
26    ]
27  }
28 }

```

Hình 4.13: Dev Tool

4.2.4 Kiến trúc RESTful API

RESTful API là một tiêu chuẩn dùng trong việc thiết kế các API cho ứng dụng web để quản lý resource. 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 bạn 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, bạn có thể dễ dàng biết được URL và HTTP method để quản lý một resource

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 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

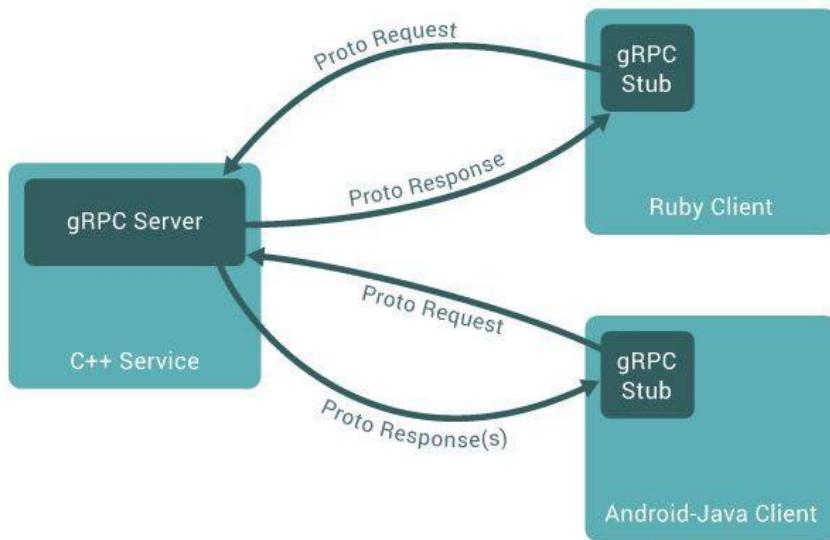
- REST Chỉ hoạt động trên các giao thức HTTP.

4.2.5 Công nghệ gRPC

gRPC có thể được xem là một giao thức request và response thông thường tuy nhiên nó được dùng cho việc giao tiếp giữa các server với nhau (server-server) nhiều hơn là client-server. Việc này có ý nghĩa rất quan trọng vì trong các hệ thống phân tán, request sẽ được xử lý bởi nhiều server hơn là một server. Ví dụ thường thấy nhất chính là kiến trúc Microservices.

Thông thường mỗi khi có request từ client, thì request đó có thể được xử lý bởi nhiều server khác nhau thì mới trả kết quả về cho client. Chính sự giao tiếp giữa các server có thể làm tăng thời gian xử lý của request đó. Chính điều này thúc đẩy sự ra đời của gRPC nhằm tăng hiệu suất giao tiếp giữa các server với nhau.

gRPC sử dụng Protocol Buffer và giao thức HTTP/2 để transfer data thay vì JSON/XML và giao thức HTTP/1.1 truyền thống nên tốc độ được gia tăng đáng kể.

**Hình 4.14:** gRPC

Cách hoạt động: Ở phía server sẽ hiện thực cách xử lý method mà nó cung cấp, sau đó ở phía client chỉ việc gửi các tham số cùng với tên method muốn gọi. Sau khi nhận được request phía server sẽ dựa trên các tham số và tên method để thực thi. Sau khi có kết quả thì server sẽ trả về cho client.

Ưu điểm

- Dựa trên file protobuf thì dữ liệu sẽ được serialize ra dạng binary nên tốc độ truyền đi trong mạng rất nhanh.
- Sử dụng HTTP/2 giúp tận dụng tối đa băng thông, giúp client trải nghiệm sản phẩm tốt hơn.

Nhược điểm

- Chưa được hỗ trợ toàn diện
- Khó hiện thực
- Dữ liệu ở dạng binary nên khó debug
- Các tool để test API như Postman, curl, ... chưa hỗ trợ

Trong quá trình nghiên cứu và tìm hiểu thì nhóm nhận thấy có rất nhiều công nghệ hỗ trợ cho việc giao tiếp giữa client-server, server-server như: SOAP, RESTful API, gRPC,...

Đối với SOAP thì nhóm nhận thấy nó có một số nhược điểm như: Nó duy trì trạng thái (stateful) khiến chúng ta tiêu tốn tài nguyên cho các metadata, không phổ biến như REST cho các ứng dụng web và chỉ sử dụng XML vì thế nhóm quyết định chọn

RESTful API để giao tiếp giữa client-server, server-server. Ngoài ra còn sử dụng gRPC cho server-server nhằm làm tăng tốc độ quá trình truyền tải dữ liệu.

4.2.6 MySQL

MySQL 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 CSDL trên internet.

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 CSDL 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 số bản ghi của bạn lớn dần lên thì việc truy xuất dữ liệu của bạn 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 tách database này ra nhiều server.

4.2.7 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.

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.

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 down 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 down.
- 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 ra 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 scale. 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 query trở nên nặng nề và rất chậm. Vì thế nhóm sẽ sử dụng MySQL để lưu các config 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.

4.2.8 Redis

Ngày nay việc tăng tốc truy vấn để đáp ứng request một cách nhanh chóng đang dần trở nên phổ biến. Mỗi khi muốn truy xuất lấy dữ liệu từ Database thì ta phải xuống 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 request của client 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.

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 làm cache có 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 b亲身: 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 đồng đảo ủng hộ. Không có giới hạn về nhà cung cấp công nghệ vì thế Redis còn có thể mở rộng và cải tiến thêm nhiều tính năng hơn nữa.

Nhược điểm

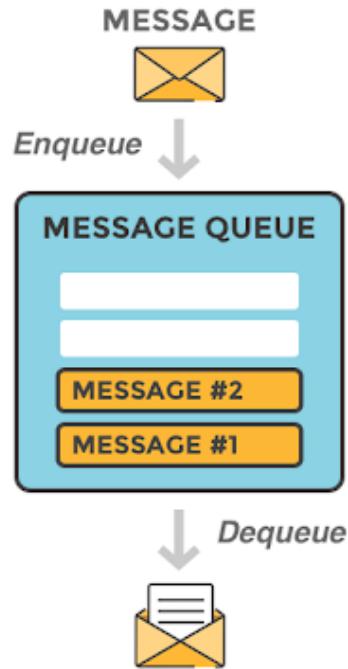
- Do Redis lưu trữ dữ liệu trong RAM nên phần nào vẫn có nguy cơ mất mát dữ liệu.
- Mỗi lần chỉnh sửa hay thêm dữ liệu thì ta đồng thời phải thêm/sửa vào cả database và Redis nên có thể mang đến một số phức tạp.

- Key trong Redis có giới hạn: Vì thế trong quá trình lưu trữ nếu không cẩn thận có thể gây chết server Redis
- Cache Invalidation: Khó 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.

Ở đây nhóm sẽ sử dụng Redis cho mục đích cache là chủ yếu. Bởi vì có một số thông tin thường xuyên được request 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.

4.2.9 ActiveMQ

Message queue là một kiến trúc cung cấp giao tiếp không đồng bộ. Ý nghĩa của queue ở đây chính là 1 hàng đợi chứa message chờ để được xử lý tuần 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.



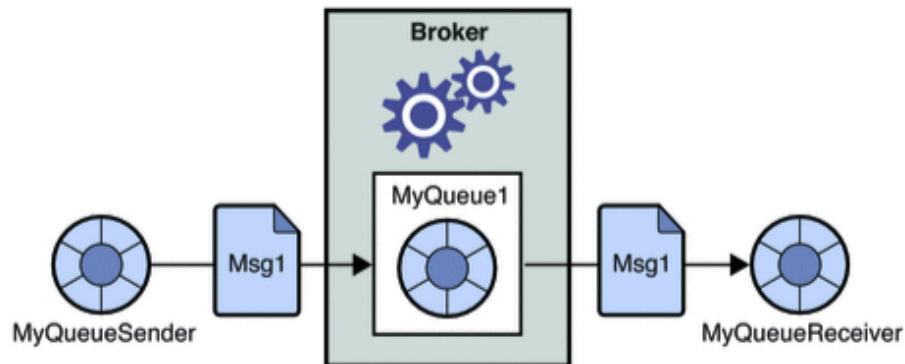
Hình 4.15: Active MQ

Kiến trúc cơ bản Message Queue

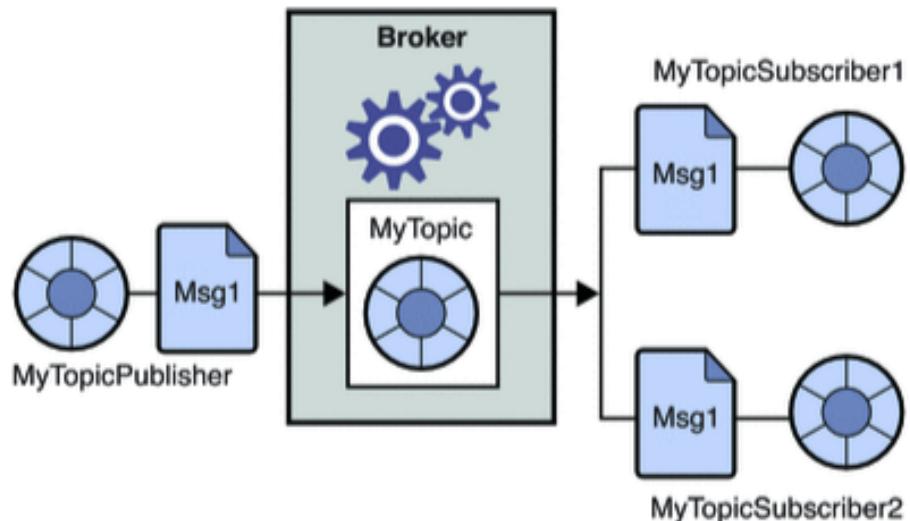
- Message: Là thông tin cần gửi đi
- Producer: Là thành phần sẽ gửi message
- Consumer: Là thành phần nhận message
- Message Queue: Nơi chứa message, cho phép producer và consumer có thể trao đổi với nhau.

Các loại Message Queue

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

**Hình 4.16:** 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 4.17:** 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 response time đối với client.
- 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 thằng 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 tính trạng hiện tại, queue có đầy hay không, có bị crash không để đưa ra phương án xử lý.

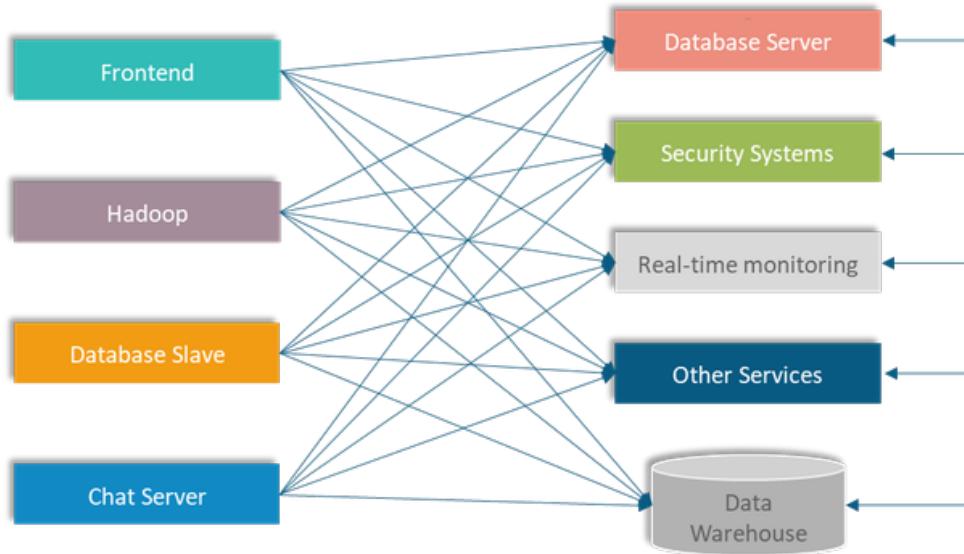
Cách sử dụng

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 client 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.

4.2.10 Kafka

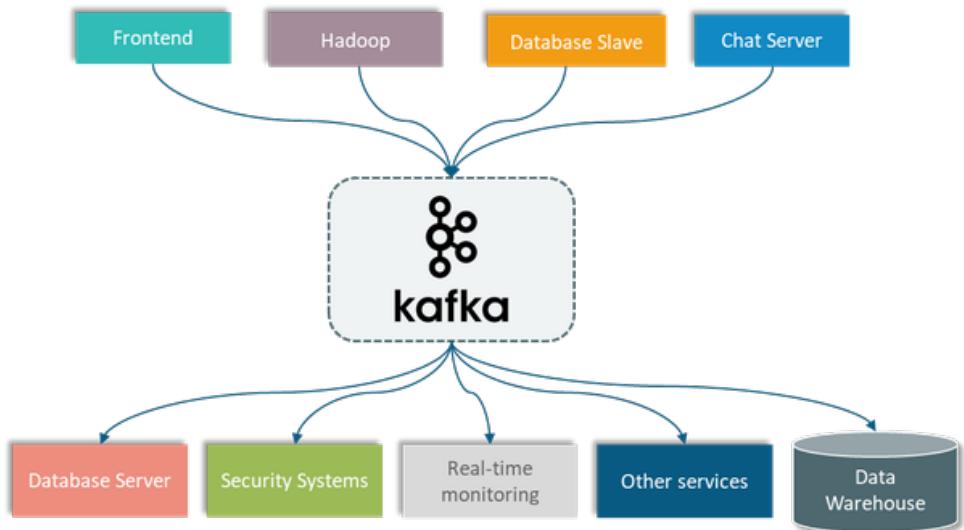
Kafka là dự án mã nguồn mở, đã được đóng gói hoàn chỉnh, khả năng chịu lỗi cao và là hệ thống nhắn tin nhanh. Vì tính đáng tin cậy của nó, kafka đang dần được thay thế cho hệ thống nhắn tin truyền thống. Nó được sử dụng cho các hệ thống nhắn tin thông thường trong các ngữ cảnh khác nhau. Đây là hệ quả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

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



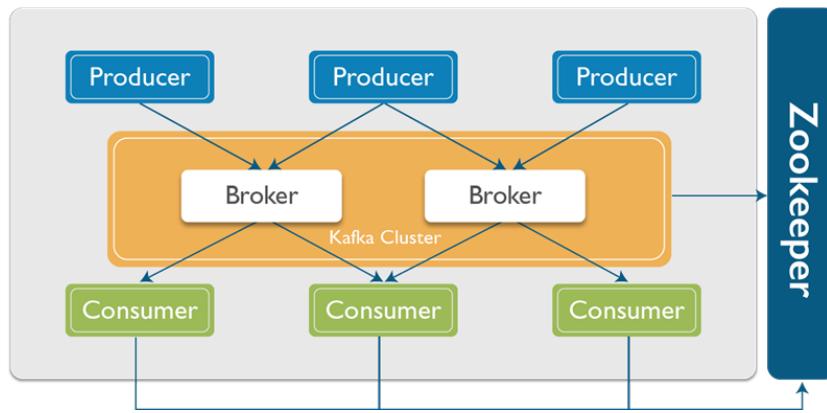
Hình 4.18: Vai trò Kafka

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 4.19: Communicate giữa các hệ thống

Cấu trúc của Kafka



Hình 4.20: Cấu trúc 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

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.

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.

5.1 Kiểm thử API

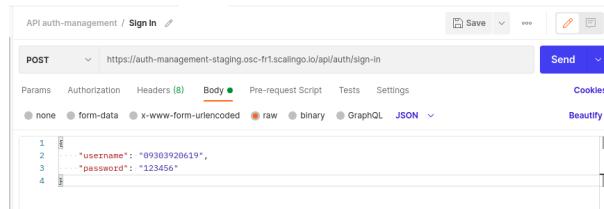
Đối với hệ thống nhóm xây dựng, các thành phần giao tiếp với nhau thông qua API. Vì vậy, cần phải kiểm thử riêng biệt các API trước khi tích hợp vào các bên. Nhóm chọn phần mềm Postman để kiểm thử các API.

API đăng nhập

Gọi API với postman

5.2 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



Hình 5.1: Kiểm thử với 10 users đồng thời

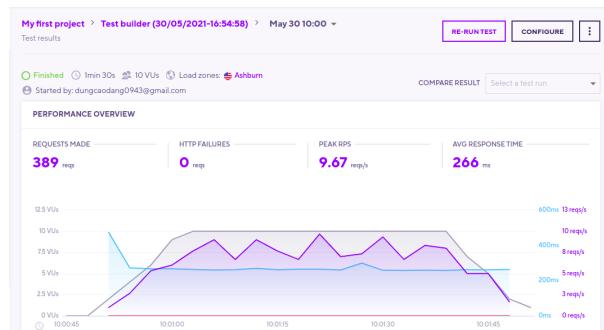
các chức năng cơ bản của hệ thống.

5.3 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>

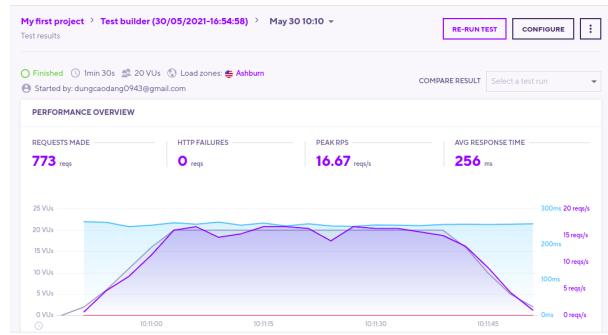
Hình ảnh kiểm thử:



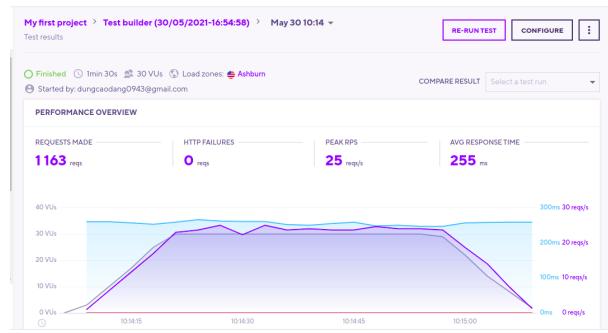
Hình 5.2: Kiểm thử với 10 users đồng thời

Kết quả kiểm thử được thống kê trong bảng sau:

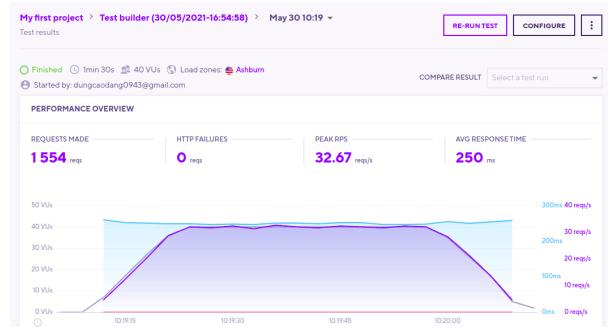
Nhìn vào bảng số liệu trên, ta thấy được hệ thống hoạt động ổn định với 50 user truy cập đồng thời cùng với số lượng request là 1930 mà không xảy ra lỗi. Bước đầu hệ thống có những kết quả tương đối tốt, nhưng sẽ cần phải có những cải thiện hơn nữa cũng như kiểm thử hệ thống với nhiều user hơn để có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



Hình 5.3: Kiểm thử với 20 users đồng thời



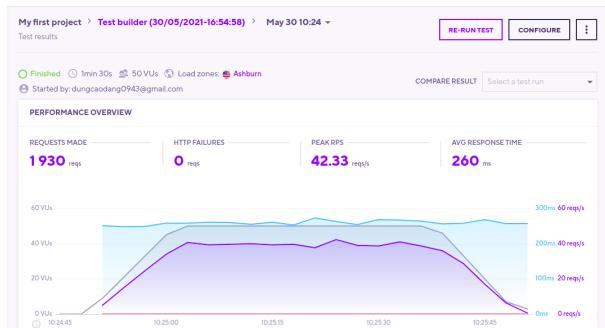
Hình 5.4: Kiểm thử với 30 users đồng thời



Hình 5.5: Kiểm thử với 40 users đồng thời

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 5.1: Kết quả kiểm thử



Hình 5.6: Kiểm thử với 50 users đồng thời

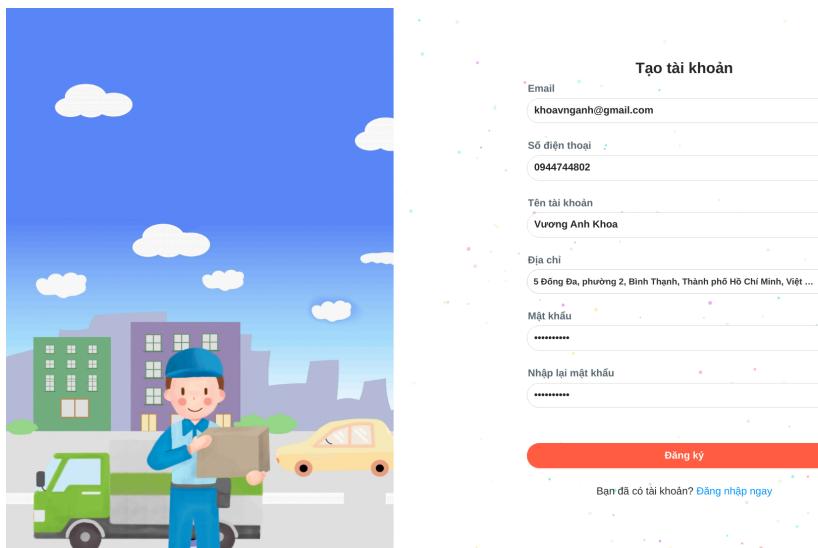
6

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.

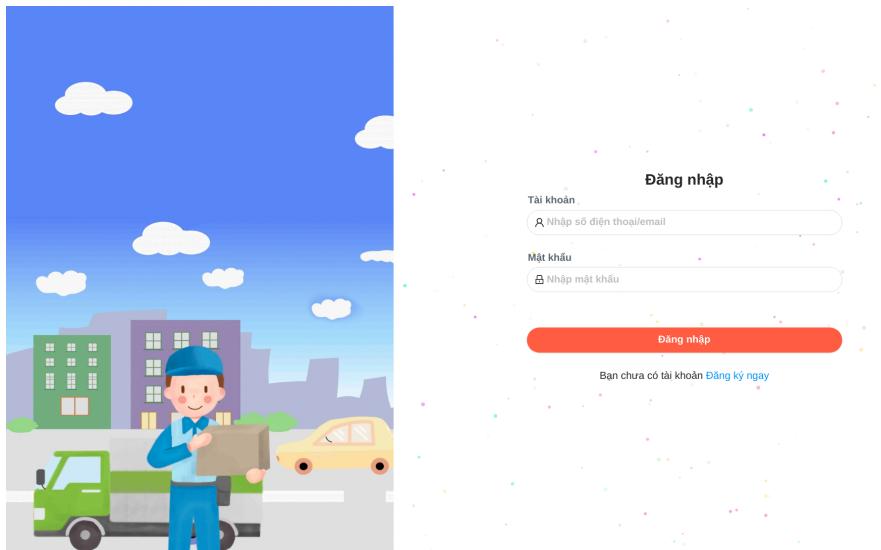
6.1 Chức năng đăng nhập, đăng ký

6.1.1 Đăng ký



Hình 6.1: Giao diện đăng ký dành cho người muốn gửi đơn hàng

6.1.2 Đăng nhập



Hình 6.2: Giao diện đăng nhập dành cho actor của hệ thống

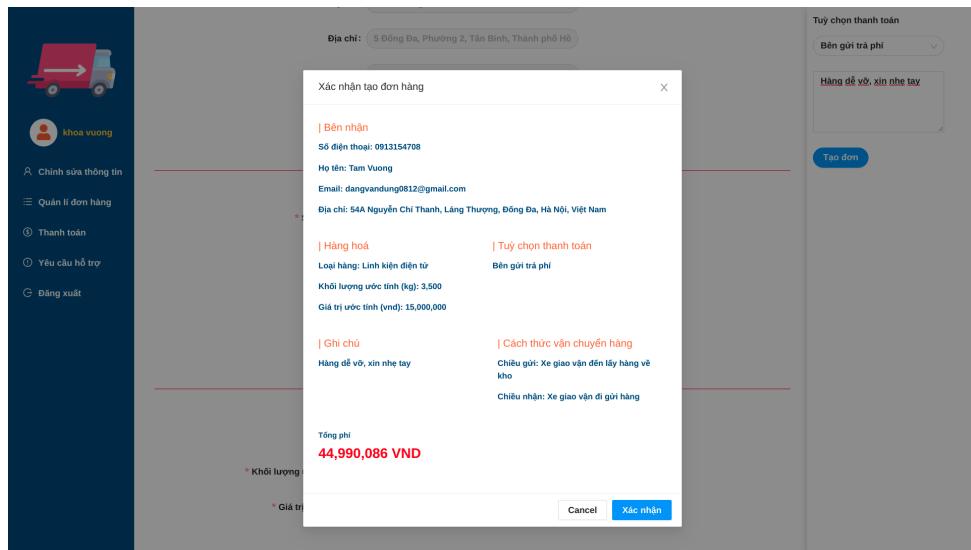
6.2 Chức năng dành cho người gửi

6.2.1 Lên yêu cầu gửi hàng

Giao diện web này cho phép người dùng tạo yêu cầu gửi hàng. Cạnh bên là menu với các mục: Quản lý đơn hàng (selected), Thành toán, Yêu cầu hỗ trợ, Đăng xuất. Trong khung chính:

- 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: khoainganh@gmail.com. Các ô sau có dấu sao (*) đều là bắt buộc. Hai ô trống là: Gửi hàng trực tiếp tại kho gửi và Nhận hàng trực tiếp tại kho đến.
- Bên nhận:** Các ô bắt buộc: Số điện thoại, Họ tên, Email, Địa chỉ, Tỉnh thành.
- Hàng hóa:** Các ô bắt buộc: Loại hàng, Khối lượng ước tính (kg), Giá trị hàng hóa (đ).
- Tùy chọn thanh toán:** Chọn hình thức trả phí (Chọn hình trả phí) và ô ghi chú.
- Nút:** Tạo đơn.

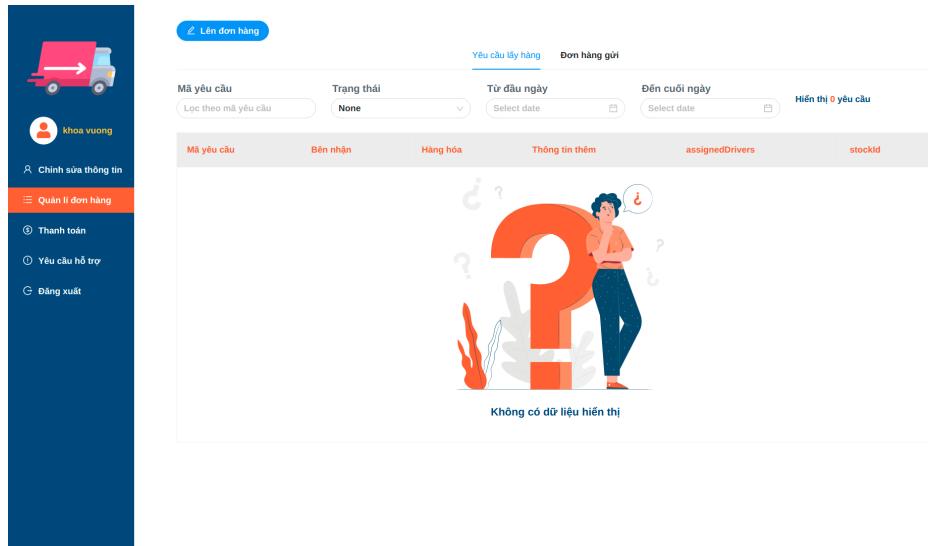
Hình 6.3: Giao diện điền thông tin để lên yêu cầu gửi hàng



Hình 6.4: Thông tin xác nhận yêu cầu 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.

6.2.2 Xem danh sách yêu cầu đã tạo



Hình 6.5: 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	⋮ Tam Vuong ☞ 0913154708 ✉ dangvandung0812@gmail.com 家住 54A Nguyễn Chí Thanh, Láng Thượng, Đông 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: Loren 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, 11:02:00 AM	⋮ Hanh Thuy ☞ 0918032337 ✉ hanhthuy@gmail.com 家住 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 6.6: Giao diện list yêu cầu của người gửi (Có yêu cầu)

Các yêu cầu đã tạo có thể được lọc theo những tiêu chí: Mã yêu cầu, trạng thái (Gồm 4 trạng thái yêu cầu đã nêu trong phần trước), thời gian tạo đơn hàng.

6.2.3 Xem danh sách các đơn hàng nhỏ được tách từ yêu cầu

Mã yêu cầu / đơn	Bên nhận	Hàng hóa	Trạng thái đơn hàng
Mã yêu cầu: 210530363294524 Mã đơn: 2105301755234006 Ngày tạo: 5/30/2021, 12:33:05 PM	⋮ Tam Vuong ☞ 0913154708 ✉ dangvandung0812@gmail.com 家住 54A Nguyễn Chí Thanh, Láng Thượng, Đông 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 家住 54A Nguyễn Chí Thanh, Láng Thượng, Đông Đa, Hà Nội, Việt Nam	⌚ Linh kiện điện tử ⌚ 500kg / 1500kg	Đang đến kho gửi

Hình 6.7: Giao diện xem danh sách các đơn hàng nhỏ được tách từ yêu cầu

6.2.4 Chính 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

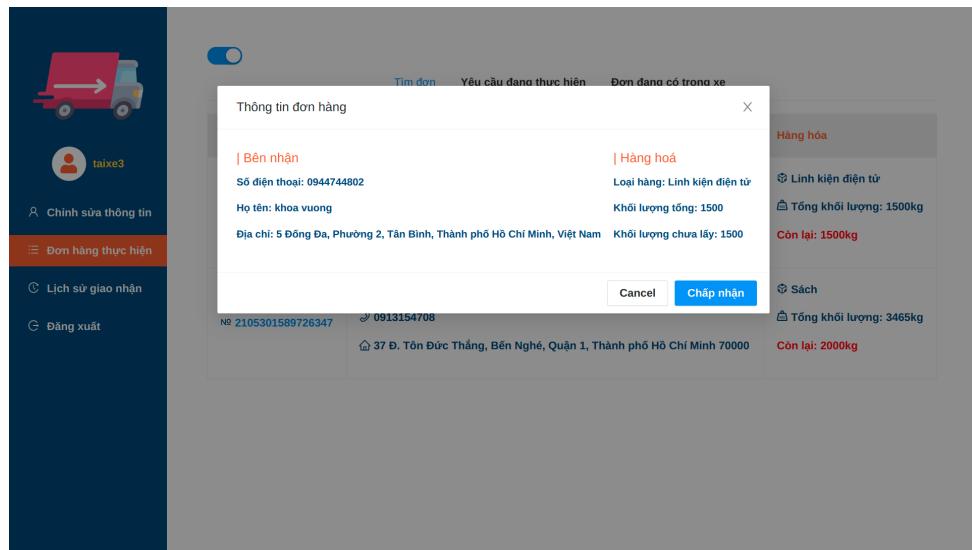
Hình 6.8: Giao diện chỉnh sửa thông tin của người gửi

6.3 Chức năng dành cho tài xế

6.3.1 Tìm những yêu cầu được gán

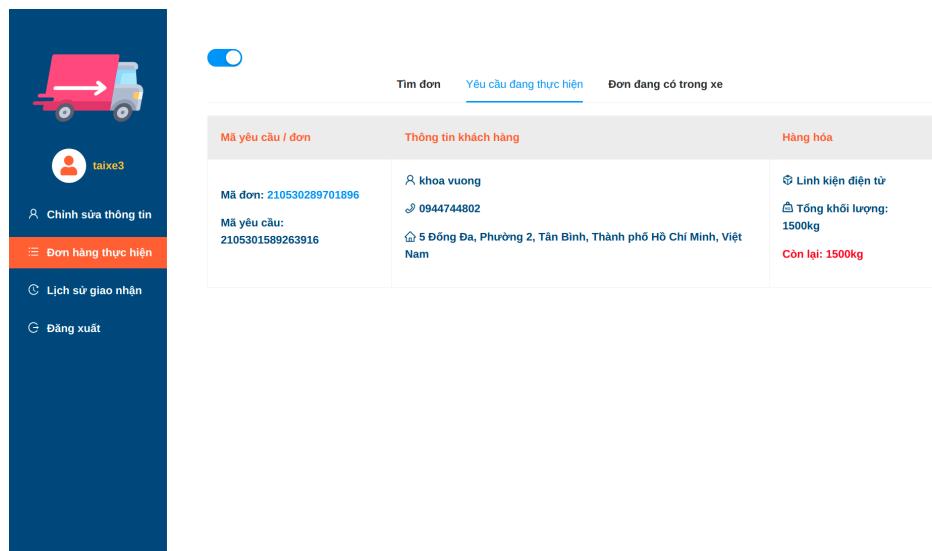
Mã yêu cầu	Thông tin khách hàng	Hàng hóa
Nº 2105301589263916	khoa vuong Ⓛ 0944744802 ⌂ 5 Đồng Đa, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	⚡ Linh kiện điện tử ⚡ Tổng khối lượng: 1500kg Còn lại: 1500kg
Nº 2105301589726347	Dũng Đặng Ⓛ 0913154708 ⌂ 37 Đ. Tôn Đức Thắng, Bến Nghé, Quận 1, Thành phố Hồ Chí Minh 70000	⚡ Sách ⚡ Tổng khối lượng: 3465kg Còn lại: 2000kg

Hình 6.9: Giao diện tìm những yêu cầu được gán

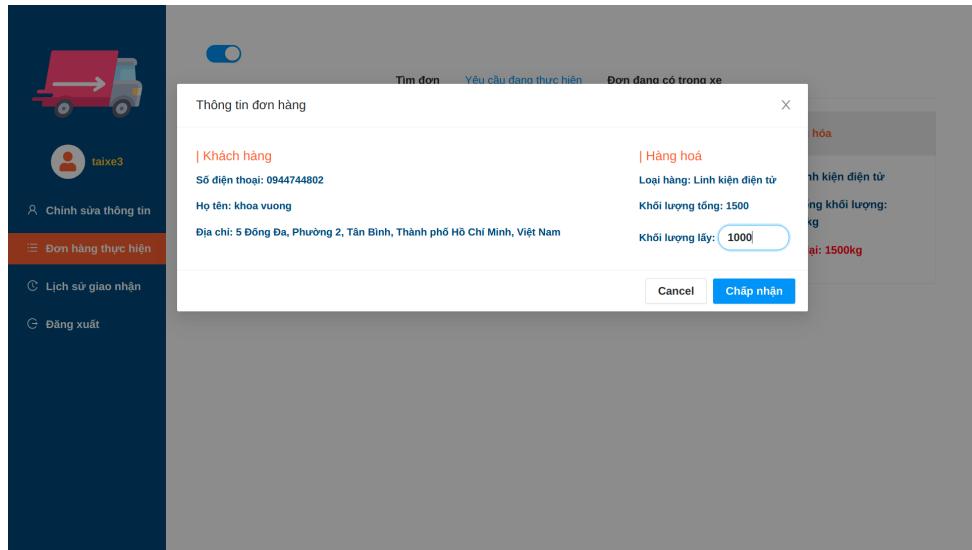


Hình 6.10: Giao diện xác nhận lại thông tin yêu cầu muốn nhận

6.3.2 Xem thông tin đơn hàng đang đến lấy

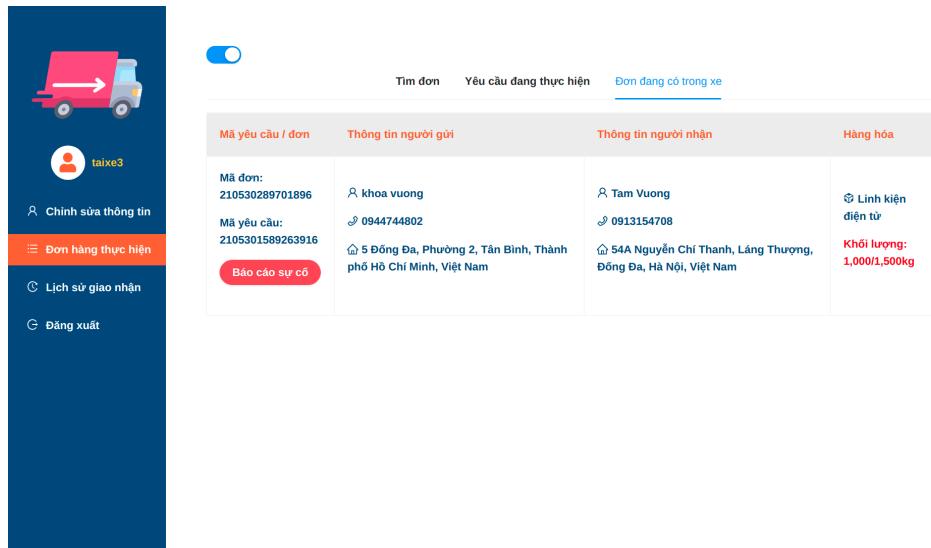


Hình 6.11: Giao diện xem thông tin đơn hàng đang đến lấy

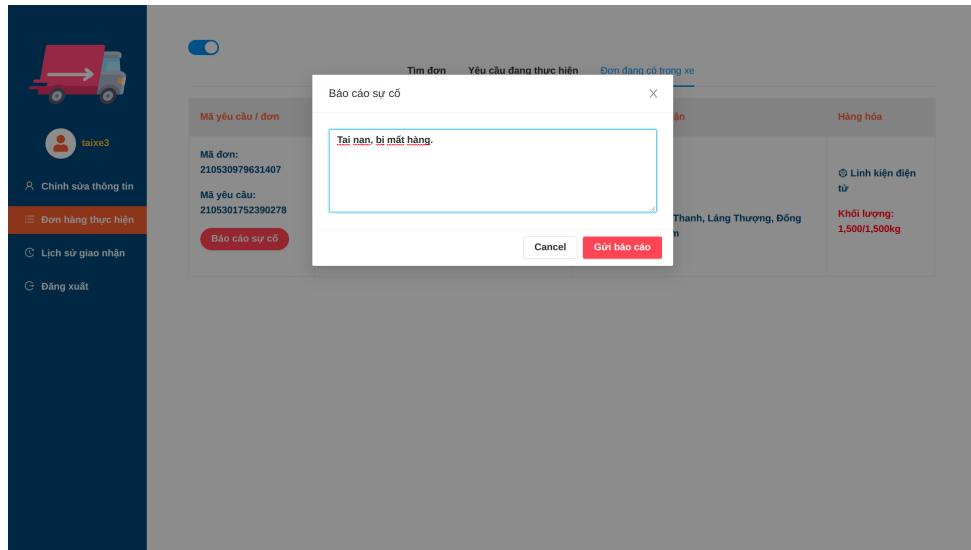


Hình 6.12: Giao diện xác nhận lại thông tin đơn hàng đang muôn nhận

6.3.3 Xem thông tin các đơn hàng đang ở trong xe

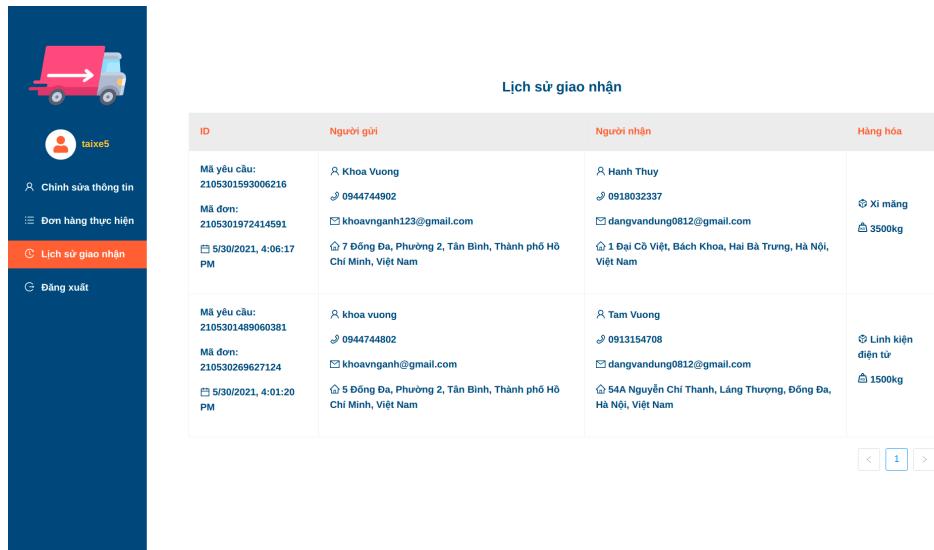


Hình 6.13: Giao diện xem thông tin các đơn hàng đang ở trong xe



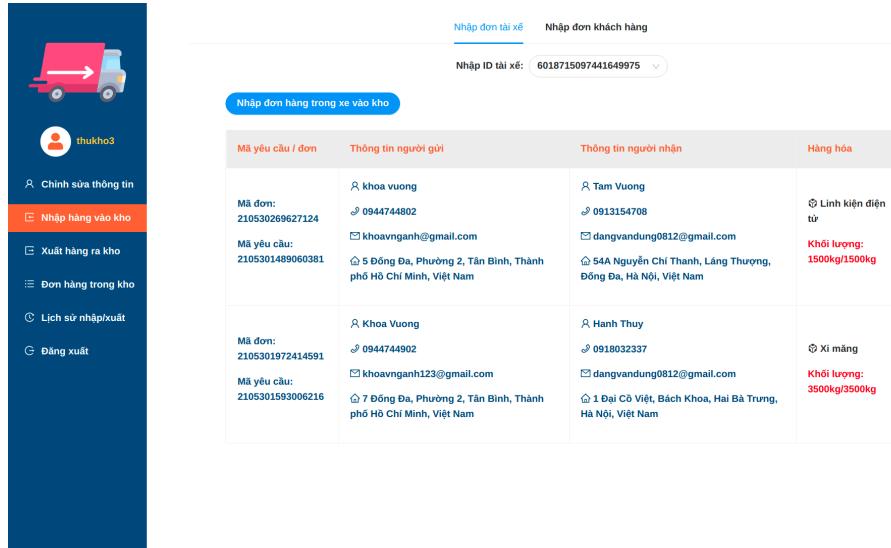
Hình 6.14: Giao diện báo cáo vấn đề ghi gấp sự cố

6.3.4 Xem lịch sử giao nhận hàng

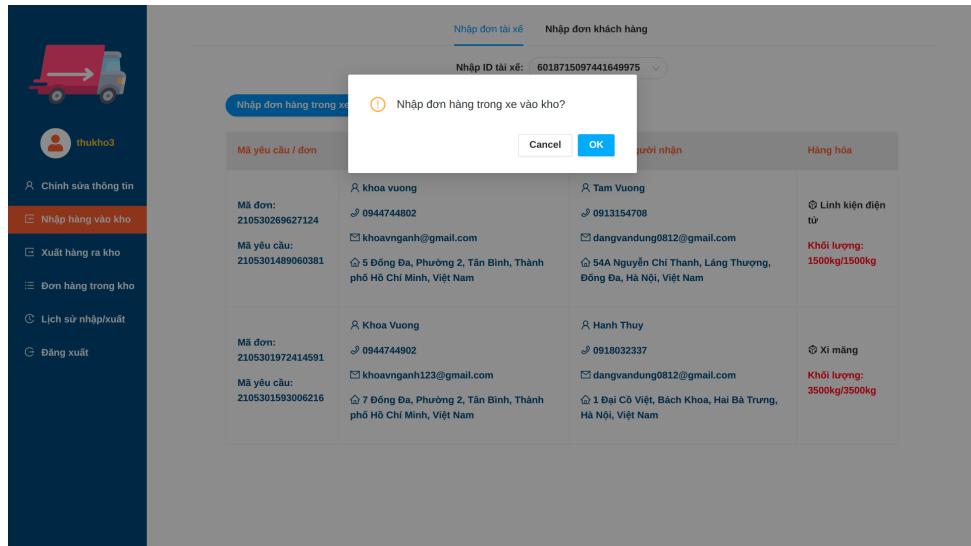


Hình 6.15: Giao diện xem lịch sử giao nhận hàng

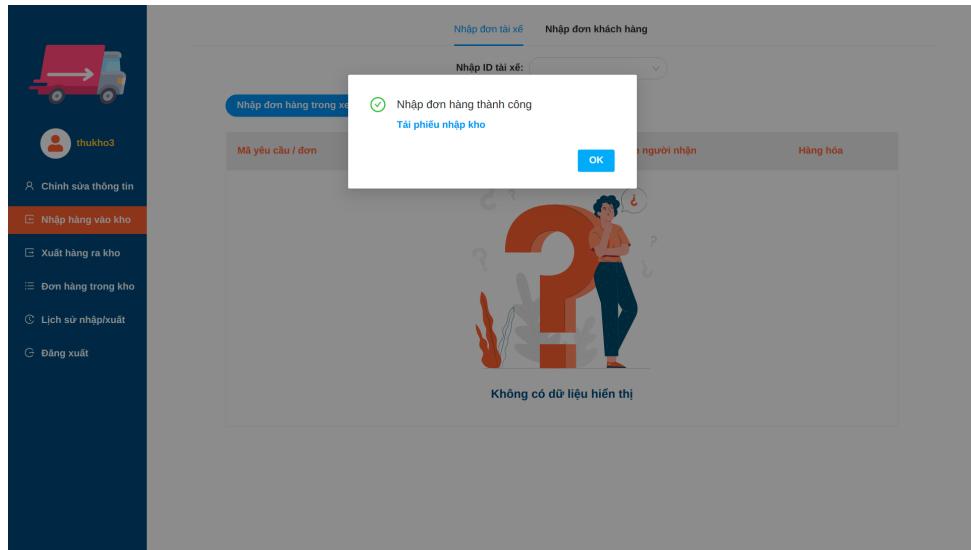
6.4 Chức năng dành cho thủ kho



Hình 6.16: Giao diện nhập ID của tài xế để nhập các đơn hàng trong xe tài xế vào kho



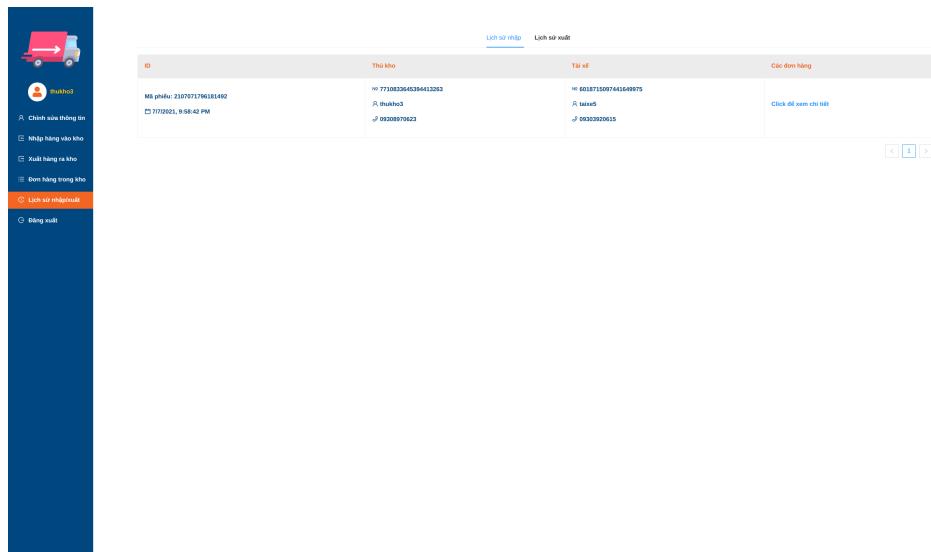
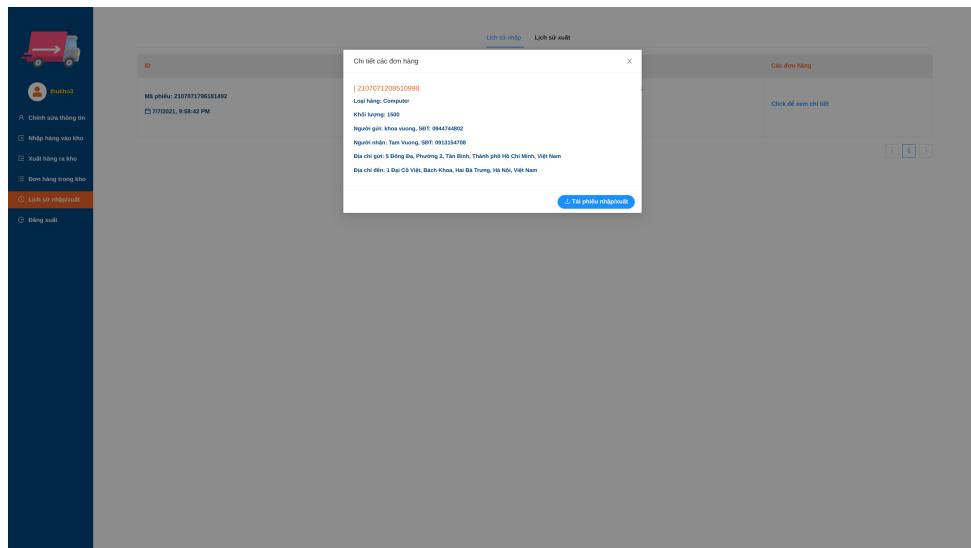
Hình 6.17: Xác nhận nhập hàng vào kho

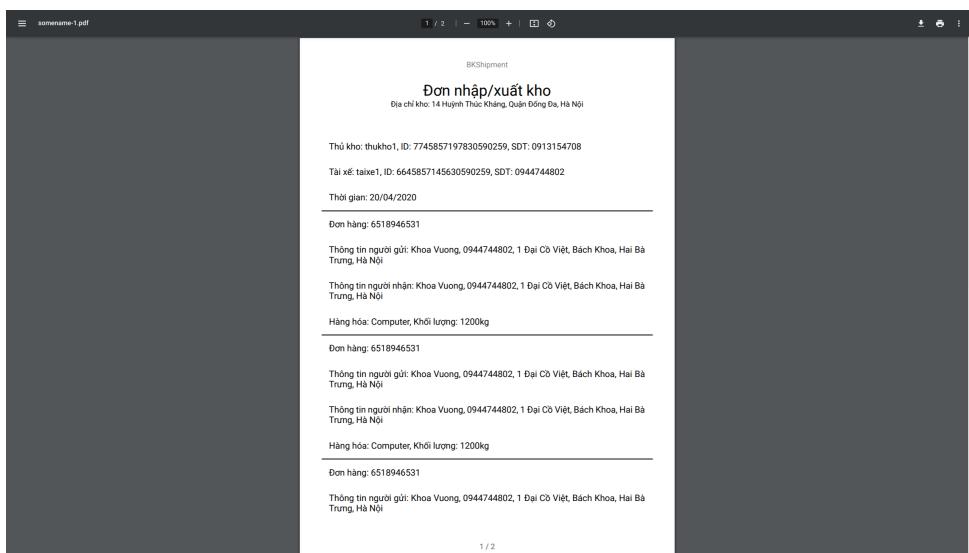


Hình 6.18: Nhập hàng thành công và cho thủ kho có thẻ tài phiếu nhập kho

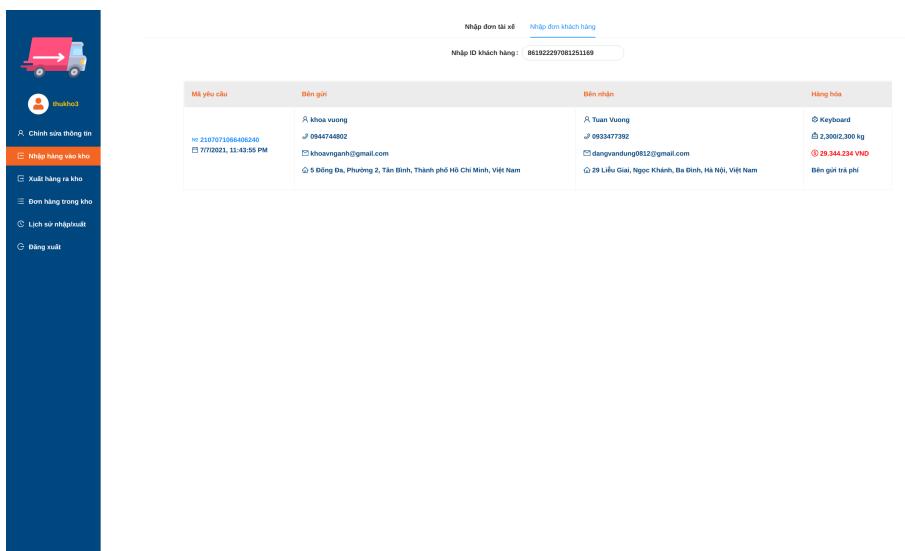
Danh sách đơn hàng trong kho 63 Thủ Nhân Trung, Phường 13, Tân Bình, Thành phố Hồ Chí Minh				
Mã yêu cầu / đơn	Thông tin người gửi	Thông tin người nhận	Hàng hóa	Tài xế phi hành
Mã đơn: 31242145123123 Mã yêu cầu: 31242145123123	A. Khoa Vuong J 0984748802 Mã yêu cầu: 31242145123123 Q 9 Đồng Da, Phường 2, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	R. Dung Dang J 0933154798 E dangdung9812@gmail.com Q 544 Nguyễn Chí Thanh, Láng Thương, Đồng Da, Hà Nội, Việt Nam	Computer Khối lượng: 1500kg/1500kg	Tài xế phi hành: H 123214234561 P. Vuong Anh Khoa J 0944744802 Tài xế phi hành: H 61090418038123 P. Dang Van Dung J 0913154708
Mã đơn: 31242145123123 Mã yêu cầu: 31242145123123	A. Hạnh Thúy J 094712827 E hanhthuy@gmail.com Q 7 Đồng Da, Phường 1, Tân Bình, Thành phố Hồ Chí Minh, Việt Nam	R. Tam Vuong J 0947102291 E tamvuong@gmail.com Q 40 Nguyễn Chí Thành, Láng Thương, Đồng Da, Hà Nội, Việt Nam	Thực phẩm Khối lượng: 2500kg/2500kg	Tài xế phi hành: H 123214234561 P. Nguyen Xuan Huy J 0944744802 Tài xế phi hành: H 61090418038123 P. Nguyen Trung Tinh J 0913154708
				Tài xế phi hành: H 123214234561 P. Bui Phi Long J 0955477892

Hình 6.19: Danh sách các đơn hàng trong kho

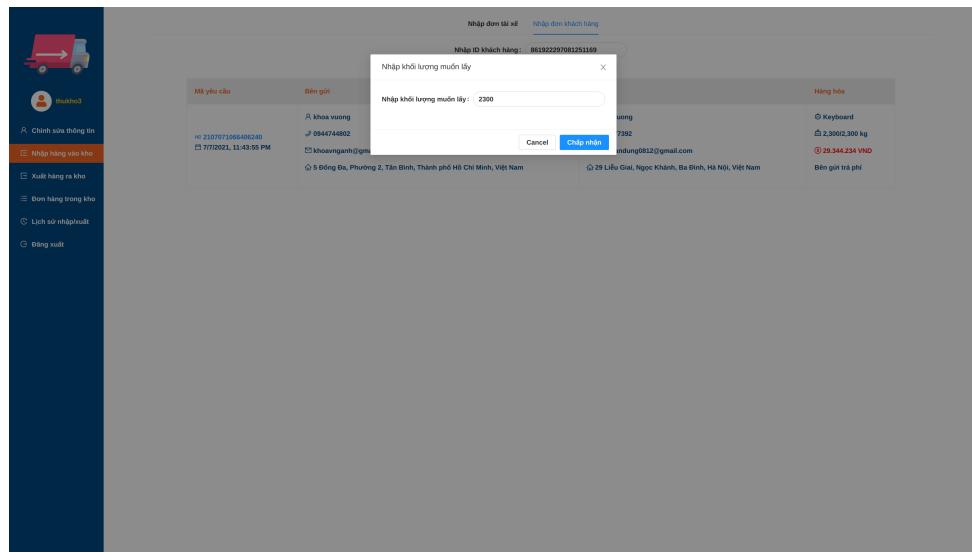
**Hình 6.20:** Lịch sử nhập/xuất kho**Hình 6.21:** Thông tin chi tiết của một lần nhập/xuất đơn hàng (Cho phép tải pdf về)



Hình 6.22: Format mẫu của 1 đơn nhập/xuất kho



Hình 6.23: Nhập ID của khách hàng để nhập đơn vào kho



Hình 6.24: Cho phép nhập khối lượng mà khách muôn nhập vào kho

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

Tài liệu tham khảo

- [1] ReactJS - xem 05/10/2020
<https://reactjs.org/>
- [2] GHN - 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 benefits - xem 27/10/2020
<https://www.cloudways.com/blog/why-reactjs-for-front-end/>
- [5] Gitlab CICD - xem 27/10/2020
<https://git.metabarcoding.org/help/ci/README.md>
- [6] Kiến trúc Microservices - xem 27/11/2020
<https://callistaenterprise.se/>
- [7] Kafka - xem 27/11/2020
<https://viblo.asia/p/kafka-la-gi-gDVK2Q7A5Lj>
- [8] MySQL - xem 27/11/2020
<https://bizflycloud.vn/tin-tuc/mysql-la-gi-tai-sao-nen-su-dung-mysql-20200917180705499.htm>
- [9] Redis - xem 27/11/2020
<https://www.hnammobilecare.com/blog/tu-van/redis-la-gi-vi-sao-redis-tro-thanh-lua-chon-hang-dau-cua-developer.583.html>

[10] Cassandra - xem 27/11/2020

<https://viblo.asia/p/tim-hieu-ve-cassandra-cassandra-la-gi-3P01PzJoKox>

[11] Message Queue - xem 27/11/2020

<https://viblo.asia/p/microservice-tai-sao-lai-su-dung-message-queue-maGK734AKj2>