# Đặt vấn đề

## Lý do chọn đề tài

Với sự bùng nổ về cả số lượng và sự đa dạng của các ứng dụng với các thao tác thu thập hoặc xử lý dữ liệu, việc triển khai các kết nối để truyền dữ liệu từ nơi nó được tạo ra đến nhiều nơi khác nhau để xử lý là một phần cơ bản của phát triển ứng dụng. Messaging là một công nghệ quan trọng để kết nối dữ liệu trong môi trường này.

Trong mô hình hướng sự kiện các dữ liệu bản ghi trở thành các sự kiện và công việc truyền các bản ghi dữ liệu - bản ghi - từ hệ thống này sang hệ thống khác sẽ được thực hiện thông qua một hệ thống trung gian. Ngược lại với các kết nối trực tiếp, nơi người gửi biết được người nhận và kết nối trực tiếp đến từng người nhận, các giải pháp messaging sẽ tách rời việc gửi dữ liệu khỏi việc xử lý dữ liệu. Người gửi không cần biết người nhận nào sẽ thấy dữ liệu đó hoặc khi nào họ sẽ thấy dữ liệu đó.

Các giải pháp messaging đơn giản hóa việc phát triển ứng dụng bằng cách cung cấp cho các architects và các nhà phát triển phần mềm một thành phần hệ thống đã được chuẩn hóa, có thể tái sử dụng để xử lý tốt luồng dữ liệu để nhà phát triển có thể tập trung vào core logic trong các ứng dụng của họ. Ngoài định tuyến dữ liệu, hệ thống messaging cũng có thể cung cấp các tính năng quan trọng như khả năng chịu lỗi, logging và xử lý phân tán để cải thiện khả năng quản lý và độ tin cậy.

Với những lợi thế đó mô hình hướng sự kiện được ứng dụng ngày càng rộng rãi và giúp ích rất nhiều cho các nhà phát triển cũng như xử lý được các yêu cầu bài toán phức tạp mà việc xử lý truyền thống không thể thực hiện được. Khi các sự kiện phát sinh thì cần phải có hệ thống để mà xử lý. Việc xử lý này rất quan trọng bởi nó ảnh hưởng rất lớn đến toàn bộ hệ thống khi mà các trung tâm trung gian này là nơi để mà tiếp nhận và phân phát các sự kiện đến các hàng đợi tin nhắn để mà xử lý. Khi có nhiều hàng đợi, quản lý toàn bộ hệ thống hàng đợi là rất quan trọng. Các phụ thuộc vào một hàng đợi cụ thể thường sẽ khác nhau, yếu tố phụ thuộc bao gồm những điều sau đây:

* Sự đồng bộ;
* Thời gian hồi đáp;
* Nội dung thông điệp;
* Kích thước thông điệp;
* Thông điệp ưu tiên;
* Dung lượng hàng đợi;
* Thời gian chờ hàng đợi;
* Lưu giữ trên hàng đợi;
* Hàng đợi ưu tiên.

Hiện nay có khá nhiều phần mềm để quản lý các tin nhắn sự kiện trong số đó có như RabbitMQ, Apache Kafka, ActiveMQ, SQS (AWS), Redis Pub/Sub,... Mỗi loại message broker sẽ có những ưu nhược điểm riêng tùy theo từng nhu cầu mục đích và chi phí phải bỏ ra.

## Mục tiêu và phạm vi nghiên cứu

### Mục tiêu nghiên cứu

* Nghiên cứu về mô hình kiến trúc hướng sự kiện
* Tổng quan về hàng đợi tin nhắn RabbitMQ (các khái niệm và mô hình)
* Tìm hiểu Restful API của Laravel, xây dựng được API giao tiếp Vue với lại Laravel
* Tìm hiểu quy trình quản lý nghiệp vụ website bán hàng
* Nghiên cứu cài đặt và phát triển Website bằng Laravel và VueJS

### Phạm vi nghiên cứu

* Tìm hiểu về mô hình hướng sự kiện và ứng dụng mô hình hàng đợi tin nhắn để xây dựng hệ thống theo hướng sự kiện.
* Tìm hiểu cách sử dụng và tạo Restful API bằng Laravel Framework trả dữ liệu để VueJS có thể sử dụng.
* Tìm hiểu nguyên lý xây dựng website bán hàng trực tuyến và tích hợp hệ thống thanh toán, vận chuyển, xác thực.

## Ý nghĩa khoa học và thực tiễn

### Ý nghĩa khoa học

* Giúp sinh viên hiểu được quy trình hoạt động của công nghệ web. Hiểu được cách

thiết kế truy xuất CSDL từ phía backend (Laravel) và giao tiếp với lại Frontend

(VueJS).

* Tìm hiểu được cách thiết kế giao diện website từ HTML, CSS, Bootstrap.

Nâng cao tư duy xây dựng và phát triển hệ thống.

### Ý nghĩa thực tiễn

* Trước sự phát triển ngày càng mạnh mẽ thương mại điện tử thì việc mua sắm ngày càng trở nên phổ biến. Việc ứng dụng mô hình hướng sự kiện và áp dụng mô hình hàng đợi tin nhắn sẽ giúp cho hệ thống thương mại điện tử có thể đáp ứng được hàng ngàn thậm chí hàng triệu phản hồi đơn hàng tử hệ thống mà không ảnh hướng đến trải nghiệm chất lượng của người dùng. Do đó việc nghiên cứu mô hình hướng sự kiện sẽ là đề tài góp phần đưa ra ý tưởng hướng tiếp cận mới mà có thể là sẽ giải quyết được các bài toán mà mô hình truyền thống sẽ gặp phải.
* Thông qua đề tài này sẽ làm cơ sở tri thức để sinh viên có thể làm việc tại các công ty phát triển phần mềm.

# CƠ SỞ LÝ LUẬN

## Tổng quan về mô hình kiến trúc hướng sự kiện

### Kiến trúc hướng sự kiện là gì ?

Sự kiện là sự thay đổi trạng thái của một đối tượng khi có các tác nhân đưa vào. Một sự kiện phải bao gồm đầu vào là những yêu cầu thay đổi trạng thái của đối tượng. Còn đầu ra chính là trạng thái của đối tượng sau khi được sửa đổi. Một sự kiện có thể trải qua nhiều hoạt động để tạo ra được trạng thái đích sự kiện này cũng có thể là đầu vào của sự kiện khác tạo ra các mắt xích sự kiện đến khi mà đạt được đến trạng thái đích. Trong mô hình hướng sự kiện, đầu vào của sự kiện có thể là dữ liệu thô được đưa vào sau một loạt quá trình lưu trữ, phân tích, tổng kết thì các sự kiện này sẽ đưa ra được kết quả mà ta yêu cầu. Trong quá trình luân chuyển các sự kiện thì các sự kiện này sẽ được lưu trữ và có thể được dịch chuyển đến một nơi nào khác cần cho quá trình phân tích, các dữ liệu cũng có thể được theo dõi và được quản lý để tránh việc mất mát hay là sai sót dữ liệu.

Kiến trúc hướng sự kiện là một phương pháp thiết kế hệ thống được xây dựng để ghi lại, truyền tải và xử lý các sự kiện thông qua một kiến trúc tách rời. Điều này có nghĩa là các hệ thống không cần biết về nhau để chia sẻ thông tin và hoàn thành nhiệm vụ. Kiến trúc này thay thế kiến trúc “request/response” truyền thống, nơi các dịch vụ sẽ phải đợi phản hồi trước khi chúng có thể chuyển sang tác vụ tiếp theo. Luồng của kiến trúc hướng sự kiện được điều hành bởi các sự kiện và nó được thiết kế để phản hồi chúng hoặc thực hiện một số hành động để phản hồi lại một sự kiện.

### Sự phát triển của kiến ​​trúc hướng sự kiện

Trong vài năm qua, đã có một phong trào từ tập trung vào dữ liệu ở trạng thái nghỉ (kiến trúc hướng dịch vụ) sang tập trung vào các sự kiện (kiến trúc hướng sự kiện). Xu hướng trong việc phát triển phần mềm là chuyển từ việc tích lũy dữ liệu và hồ dữ liệu sang tập trung vào dữ liệu trong chuyến bay và theo dõi nó trong khi nó đang di chuyển từ nơi này sang nơi khác.

Đối với các mô hình truyền thống dữ liệu sẽ được tập trung và xử lý ở một chỗ ở đây có thể được gọi là trung tâm dữ liệu khi xử lý các sự kiện xảy ra thứ họ quan tâm nhất chính là sự toàn vẹn và đúng đắn của dữ liệu, đổi lại thứ họ phải đánh đổi chính là hiệu năng và tốc độ của ứng dụng, các phát sinh về sự cố sẽ diện ra ngay lập tức sau khi mà sự kiện được đưa vào xử lý và trong quá trình đó có một hoạt động nào đó phát sinh ra lỗi. Trong mô hình hướng sự kiện, các dữ liệu cũng là một phần không thể nào thiếu, nhưng các sự kiện hay là các thay đổi là quan trọng nhất. thứ họ quan tâm là sự kiện sẽ được diễn ra. Tuy nhiên trong một mô hình dụng thì hai mô hình này vẫn được sử dụng cùng nhau cả hai mô hình cùng nhau hỗ trợ để đảm bảo được mục đích của từng ứng dụng.

Điều gì sẽ xảy ra khi mà chúng ta đi vào nhà hàng thưởng thức món ăn, đối với mô hình hướng dịch vụ các nhà hàng yêu cầu khách hàng phải xếp thành một hàng đợi đến khi nào tới lượt của mình thì sẽ đến quầy phục vụ tiến hành đặt đồ ăn sau đó sẽ tiến hành thanh toán với người phục vụ, sau khi đã thanh toán xong người khách hàng vẫn phải đứng đợi người phục vụ đưa giấy đặt hàng cho bên đầu bếp để chế biến theo đơn hàng. Sau khi các món trong đơn hàng được làm xong thì người phục vụ sẽ đưa đồ ăn đến quầy phục vụ để đưa cho khách hàng. Lúc này khách hàng đã nhận được đồ ăn của mình sẽ tìm đến một bàn ăn nào đó để thưởng thức món ăn. Lúc này người phục vụ sẽ sẵn sàng tiếp nhận khách hàng tiếp theo và các quy trình vẫn sẽ diễn ra như vậy đến khi nào phục vụ đến hết người cuối cùng. Rõ ràng ta thấy các hoạt động này sẽ phải tốn rất nhiều thời gian và chi phí cho mỗi lần phục vụ, ứng dụng mô hình hướng sự kiện, khi mà khách hàng đến họ cũng sẽ được bắt vào đứng đợi ở trong hàng sau đó khi nào mà khách hàng tới lượt của mình thì người khách hàng đó sẽ đến quầy phục vụ tiến hành đặt đồ ăn. Sau khi đã nhận được đơn hàng từ phía khách hàng người phục vụ sẽ đưa thông tin món ăn trong đơn hàng cho bếp, bếp sẽ tiến hành làm món ăn ở cùng thời điểm này người phục vụ sẽ hướng dẫn khách hàng ngồi vào bàn chờ. Lúc này quầy phục vụ đã bị trống đã sẵn sàng nhận đơn hàng từ phía khách hàng tiếp theo, cũng trong quá trình này người phục vụ cũng có thể nhận yêu cầu thanh toán từ phía khách hàng. Mặc dù các hoạt động này diễn ra không liên tục nhưng mà phải đảm bảo được hết tất cả các sự kiện phải xảy ra. Đây cũng chính là ý tưởng của mô hình hàng đợi tin nhắn được nghiên cứu trong cuốn báo cáo này

Các sự kiện trong thiết kế hệ thống có chung các đặc điểm:

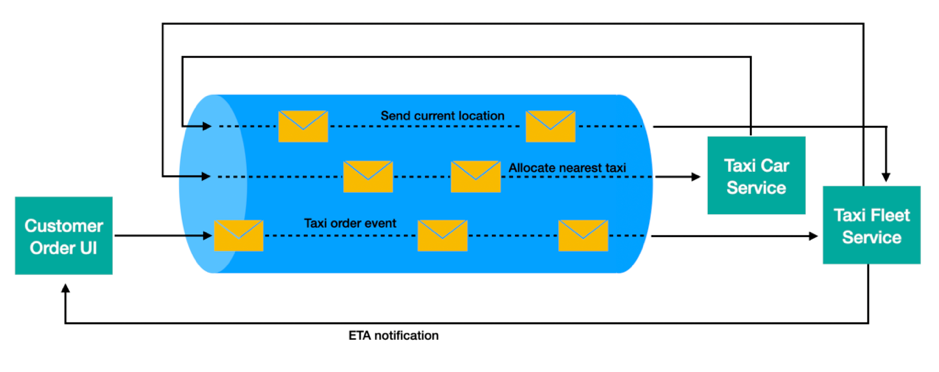
* Chúng là một bản ghi cho biết rằng điều gì đó đã xảy ra.
* Chúng là bất biến - chúng không thể thay đổi hoặc xóa được.
* Chúng có thể tồn tại vô thời hạn - các sự kiện có thể được lưu trữ và truy cập mãi mãi.
* Chúng có thể được sử dụng không giới hạn số lần - không giới hạn số lần một sự kiện có thể được xử lý bởi một dịch vụ.

Kiến trúc hướng sự kiện xử lý thông báo sự kiện - thông báo sự kiện là một tin nhắn xác nhận một hành động đã được thực hiện. Thông báo có thể là một xác nhận đơn giản về hành động hoặc chứa dữ liệu bổ sung về sự kiện mà các hệ thống hạ lưu sử dụng để áp dụng logic nghiệp vụ của chúng.

**Kiến trúc hướng sự kiện hoạt động như thế nào?**

Các thành phần của kiến ​​trúc hướng sự kiện có thể bao gồm ba phần: nhà sản xuất, bên nhận thông tin , nhà môi giới. Người môi giới có thể là tùy chọn, đặc biệt khi bạn có một nhà sản xuất và một bên nhận thông tin duy nhất đang giao tiếp trực tiếp với nhau và nhà sản xuất chỉ gửi các sự kiện cho bên nhận thông tin .

Một ví dụ đơn giản của EDA: một module quản lý việc đăng nhập của user cần chứng thực thông tin của user vừa nhập xong nên tự phát sinh và gửi đi một event gọi là LoginEvent chứa thông tin user. Event này sau đó được một module có khả năng thao tác với dữ liệu như WebServer, Database… bắt lấy, thực hiện việc kiểm chứng và sau đó trả lời kết quả thông qua LoginResultEvent để module đăng nhập bắt lấy. Theo cách xử lý này thì module đăng nhập không cần biết module nào và sẽ làm thế nào để thực hiện việc kiểm tra, nó chỉ cần biết gửi yêu cầu và nhận kết quả sau khi kiểm tra và tất cả những gì nó quan tâm chỉ là các event được định nghĩa ở mức hệ thống. Hơn nữa, event kết quả trong trường hợp trên có thể được quan tâm bởi nhiều module khác như module đảm nhận ghi log và do đó làm cho hệ thống càng mềm dẻo hơn.



**Sự tách biệt thực sự giữa Bên Event Publish và bên nhận thông tin**

Hệ thống sử dụng kiến ​​trúc hướng sự kiện tách các thành phần trong hệ thống, phân tách quyền sở hữu dữ liệu theo miền:

* Bên Event Publish không cần quan tâm đến việc các sản phẩm mà họ sản xuất sẽ được tiêu thụ như thế nào (vì vậy có thể thêm bên nhận thông tin mà không ảnh hưởng đến Bên Event Publish ).
* Bên nhận thông tin không cần quan tâm đến việc chúng được sản xuất như thế nào.

Do khớp nối lỏng lẻo này, các hệ thống microservices có thể được thực hiện bằng các ngôn ngữ khác nhau hoặc sử dụng các công nghệ khác nhau và phù hợp với các công việc cụ thể. Do đó, việc mã hóa dữ liệu sự kiện không quan trọng – nó có thể là JSON, XML, Avro, v.v.

Việc tách các thành phần của một ứng dụng cũng cho phép chúng được chia tỷ lệ dễ dàng và độc lập với nhau trên toàn mạng. Các nhà phát triển có thể sửa đổi hệ thống của họ bằng cách thêm hoặc xóa động các nhà sản xuất sự kiện và bên nhận thông tin mà không cần thay đổi bất kỳ logic nào trong bất kỳ dịch vụ nào.

**Các trường hợp sử dụng kiến trúc theo hướng sự kiện**

* **Sao chép dữ liệu**: Một sự kiện có thể được chia sẻ giữa nhiều dịch vụ cần sao chép dữ liệu của nó vào cơ sở dữ liệu của chúng.
* **Xử lý song song**: Nhiều quá trình có thể được kích hoạt bởi một sự kiện để thực thi không đồng bộ của nhau.
* **Giám sát thời gian thực**: Các hệ thống có thể tạo ra các sự kiện cho các thay đổi đối với trạng thái của chúng để một tổ chức có thể quét các điểm bất thường và các hoạt động đáng ngờ.
* **Khả năng tương tác**: Các sự kiện có thể được tồn tại và lan truyền bất kể dịch vụ mã được viết bằng gì.
* **Dự phòng**: Nếu một dịch vụ không hoạt động, các sự kiện có thể được duy trì trong bộ định tuyến cho đến khi dịch vụ khả dụng để sử dụng sự kiện.
* **Microservices**: EDA thường được ghép nối với microservices để chia sẻ thông tin hiệu quả giữa các hệ thống được tách rời trên quy mô lớn.

**Lợi ích của kiến trúc theo hướng sự kiện**

* **Khả năng chịu lỗi cao**: tính năng tách rời mang lại cho EDA khả năng chịu lỗi cao vì lỗi trong một dịch vụ sẽ không ảnh hưởng đến các dịch vụ còn lại.
* **Lưu vào bộ đệm**: do tính chất không đồng bộ của EDA, không cần đợi consumer bắt kịp nếu các sự kiện được tạo với tốc độ khác với tốc độ chúng được tiêu thụ.
* **Khả năng mở rộng**: mỗi dịch vụ có thể được mở rộng một cách độc lập để phù hợp nhất với nhu cầu của doanh nghiệp.
* **Tiết kiệm chi phí**: Việc chuyển đổi các quy trình nghiệp vụ thành kiến trúc hướng sự kiện loại bỏ nhu cầu về các quy trình hàng loạt này và thay vào đó cho phép một luồng dữ liệu liên tục được xử lý khi nó được tạo ra. Cách tiếp cận xử lý liên tục này cho phép doanh nghiệp đạt được thời gian phản hồi có thể dự đoán được và khả năng mở rộng xử lý, do đó giảm số lượng cơ sở hạ tầng cần thiết để xử lý định kỳ một lượng lớn dữ liệu. Do đó, điều này làm giảm chi phí hoạt động của việc trích xuất giá trị từ dữ liệu và tăng hiệu quả tổng thể, năng suất và lợi nhuận. EDA cũng tiêu thụ ít băng thông hơn.
* **Nhanh nhẹn**: Các sự kiện được tự động lọc và định tuyến mà không cần tạo mã tùy chỉnh dài dòng.
* **Vị trí tập trung**: Luồng sự kiện hoạt động như một vị trí tập trung, nơi các thay đổi chính sách và kiểm tra có thể được thực hiện.
* **Nâng cao trải nghiệm khách hàng**: EDA là một phương pháp tiếp cận năng động đối với trải nghiệm khách hàng và quản lý khách hàng - một phương pháp giúp dễ dàng đáp ứng kỳ vọng của khách hàng xung quanh việc cá nhân hóa mà không bỏ lỡ bất kì khoảnh khắc tương tác nào.

## Giới thiệu về Restful API

### Restful API là gì ?

Restful API là một trong những tiêu chuẩn trong quá trình phát triển phần mềm cho phép lập trình viên tạo và quản lý các tài nguyên để giao tiếp từ phía server và phía client. Dữ liệu truyền qua thông qua các giao thức HTTP, HTTPS thường sẽ là các dữ liệu dạng thô, json, binary text.

### Các thành phần của Restful API

API (Application Programming Interface) là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu mà ta cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML.

REST (REpresentational State Transfer) là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy. Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE,... đến một URL để xử lý dữ liệu.

RESTful API là một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các loại 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...) khác nhau giao tiếp với nhau. Chức năng quan trọng nhất của REST là quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE...) và cách định dạng các URL cho ứng dụng web để quản các resource. RESTful không quy định logic code ứng dụng và không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kỳ ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một RESTful API.

Chức năng quan trọng nhất của REST là quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE...) và cách định dạng các URL cho ứng dụng web để quản các resource. RESTful không quy định logic code ứng dụng và không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kỳ ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một RESTful API.

### Restful API hoạt động như thế nào

REST hoạt động chủ yếu dựa vào giao thức HTTP. Các hoạt động cơ bản nêu trên sẽ sử dụng những phương thức HTTP riêng.

* GET (SELECT): Trả về một resource hoặc một danh sách resource.
* POST (CREATE): Tạo mới một resource.
* PUT (UPDATE): Cập nhật thông tin cho resource.
* DELETE (DELETE): Xoá một resource.

Những phương thức hay hoạt động này thường được gọi là CRUD tương ứng với Create,

Read, Update, Delete – Tạo, Đọc, Sửa, Xóa.

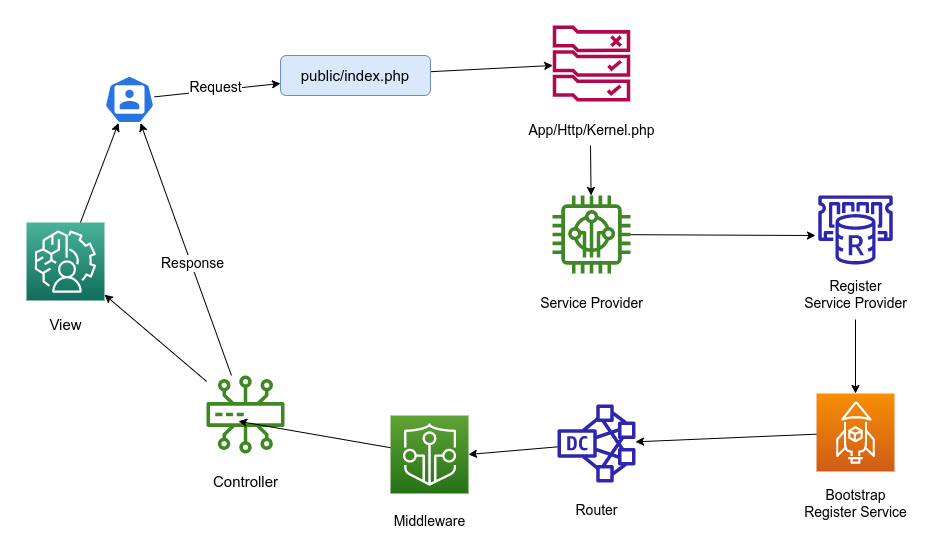
## Laravel Framework

### Laravel là gì ?

Laravel là một framework mã nguồn mở miễn phí của PHP, được tạo ra bởi Taylor Otwell (một lập trình viên kỳ cựu chuyển từ .NET sang PHP) giúp xây dựng những ứng dụng theo mô hình MVC (Model - View - Controller). Laravel xây dựng cấu trúc thư mục có tính khoa học cao, các tài liệu hướng dẫn và cộng đồng hỗ trợ và phát triển cực kỳ mạnh mẽ. Đối với người vừa bắt đầu tìm hiểu về Framework thì Laravel chính là một nguồn tài liệu để có thể mà phát triển bản thân theo hướng lập trình website.

### Cơ chế hoạt động của Laravel

Khi truy cập một ứng dụng Laravel thì một yêu cầu sẽ được gửi về server. Đầu tiên khi vào trong ứng dụng Laravel thì server sẽ tiến hành tải các service provider đã được đăng ký trước sau đó boot chúng lên. Các yêu cầu sau đó được định tuyến đến đường dẫn như yêu cầu và các yêu cầu này sẽ được xử lý ở phía bên trong controller. Lúc này controller sẽ đảm nhiệm nhiệm vụ xử lý tương tác với CSDL và trả về kết quả cho người dùng.



### Cấu trúc thư mục của Laravel

### Databases

Laravel làm cho việc kết nối tới các database và thực thi các query cực kì đơn giản với

nhiều database backend thông qua sử dụng raw SQL, fluent query builder, và Eloquent ORM. Hiện tại, Laravel hỗ trợ sẵn bốn database sau:

* MySQL
* Postgres
* SQLite
* SQL Server

Một trong những công cụ để hỗ trợ truy vấn mạnh mẽ của Laravel là Eloquent ORM. Eloquent ORM đi kèm với Laravel cung cấp một API ActiveRecord đơn giản và tiện lợi cho giao tiếp với database. Mỗi database table sẽ có một "Model" tương ứng để tương tác với table đó. Model cho phép query dữ liệu trong table, cũng như chèn thêm các dữ liệu mới.

### Routing

Tập tin định nghĩa của Laravel được đặt ở thư mục routes mặc định khi tạo dự án, Laravel sẽ tạo ra 4 file route

* Api.php: Thường để quản lý định tuyến các yêu cầu xử lý liên quan đến api
* Web.php: Thường được dùng để quản lý định tuyến để render ra các view
* Channels.php: Thường dùng để quản lý định tuyến broadcast, socket
* Console.php: Thường dùng để quản lý định tuyến các console

Các phương thức được sử dụng nhiều nhất trong Laravel

* Route::get(url, [controller, method]): Sử dụng để lấy các tài nguyên mà hệ thống trả về.
* Route::post(url, [controller, method]): Sử dụng để thêm một tài nguyên mới.
* Route::put(url, [controller, method]): Sử dụng để thay đổi dữ liệu của một tài nguyên.
* Route::delete(url, [controller, method]): Sử dụng để xóa một tài nguyên.

### Views

Views sẽ chứa nội dung định dạng HTML cho phép hiển thị giao diện người dùng. Khi có yêu cầu truy cập từ phía người dùng thì từ controller sẽ xác định được đường dẫn từ đó sẽ hiển thị giao diện đúng với lại đường dẫn được yêu cầu. Laravel sử dụng Blade Engine để render ra giao diện người dùng, blade cung cấp nhiều macro cho phép người lập trình viết các cú pháp hiển thị một cách ngắn gọn và tái sử dụng được nhiều code hơn. Một ví dụ đơn giản về blade view:

<html>

<head>

</head>

<body>

<h1> Hello {{ $name }} </h1>

</body>

</html>

Ở ví dụ trên biến $name sẽ được truyền từ phía controller sau đó sẽ được hiển thị ở trên giao diện người dùng. Ta có thể định nghĩa một route đơn giản để hiển thị blade view trên như sau:

Route::get(‘/’, function() {

return view(‘index’, [‘name' => ‘world]);

});

Giao diện sẽ được hiển thị như sau:



Với blade engine ta hoàn toàn có thể làm nhiều thứ với các macro mặc định như @extends, @if, @foreach,...

### Controllers

Controller được xem là trung tâm xử lý logic ứng dụng Laravel, tất cả các yêu cầu từ phía người dùng sẽ được đưa đến controller sau đó được xử lý và phản hồi lại cho phía bên người dùng. Controller cho phép nhóm các HTTP request liên quan đến logic do đó sẽ làm cho mã nguồn sẽ trở nên dễ đọc và dễ hiểu. Các controller sẽ được đặt ở trong thư mục app/Http/Controllers

Ví dụ về một controller cơ bản như sau:

<?php

namespace App\Http\Controllers;

class HomeController extends Controller {

public function index() {

return view(‘index', [‘name' => ‘world']);

}

}

Những resource controller làm cho việc xây dựng các RESTful controller xung quanh các nguồn tài nguyên trở nên dễ dàng hơn. Ví dụ như, ta muốn tạo một controller xử lý những HTTP request liên quan đến sản phẩm được lưu trữ trong ứng dụng. Sử dụng câu lệnh Artisan make:controller, chúng ta có thể nhanh chóng tạo ra controller:

php artisan make:controller SanPhamController --resource

Câu lệnh Artisan sẽ tạo ra file controller tại app/Http/Controllers/SanPhamController.php. Controller sẽ bao gồm method cho các hoạt động của tài nguyên sẵn có bao gồm: index, show, update, destroy.

Tiếp theo, đăng ký một định tuyến đa tài nguyên cho controller:

Route::resource(“/san-pham”, “SanPhamController”);

Laravel sẽ tạo cho ta một danh sách các url:

GET /san-pham

GET /san-pham/:id

POST /san-pham/:id

PUT /san-pham/:id

DELETE /san-pham/:id

### Middleware

HTTP middleware cung cấp một giải pháp tiện ích cho việc lock các HTTP request vào ứng dụng. Ví dụ, Laravel có chứa một middleware xác thực người dùng đăng nhập vào hệ thống. Nếu user chưa đăng nhập, middleware sẽ chuyển hướng user tới màn hình login. Còn nếu user đã đăng nhập rồi, thì middleware sẽ cho phép request được thực hiện tiếp tiến trình xử lý.

Tất nhiên là có thể viết thêm middleware để thực hiện nhiều tác vụ nữa ngoài việc kiểm tra đăng nhập vào hệ thống. Middleware CORS chịu trách nhiệm cho việc thêm các header hợp lý vào trong tất cả các response gửi ra ngoài. Middleware log có thể thực hiện ghi log cho tất cả các request tới chương trình. Vài middleware đã có sẵn trong Laravel framework, bao gồm middleware cho bảo trì, xác thực, phòng chống CSRF và còn nữa. Tất cả những middleware này nằm trong thư mục app/Http/Middleware.

### Requests

Để lấy đối tượng của HTTP request hiện tại thông qua dependency injection, ta phải type- hint Illuminate\Http\Request vào trong hàm khởi tạo của controller hay phương thức trong controller. Đối tượng của request hiện tại sẽ được tự động inject vào bởi service container:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller

{

public function store(Request $request)

{

$name = $request->get(“name”);

}

}

Một số method cơ bản của đối tượng Request:

* All: lấy hết những params được truyền lên
* Get: lấy giá trị dựa vào key truyền vào
* Input: cũng là lấy giá trị dựa vào key truyền vào
* Has: Kiểm tra xem key có được truyền lên hay không
* Only: Chỉ lấy những giá trị mà key được truyền vào

### Response

Tất cả các route và controller luôn nên trả về giá trị cho phía người dùng. Laravel cung cấp nhiều cách khác nhau để phản hồi lại phía người dùng. Cơ bản nhất sẽ là trả về dữ liệu text như ví dụ ở phía dưới

Route::get('/', function () {

return 'Hello World';

});

Ngoài ra, Laravel còn hỗ trợ rất nhiều cách trả về dữ liệu như array, string, json,...

Khi phản hồi cho phía người dùng thì ta hoàn toàn có thể:

* Đính kèm header trả về cho người dùng
* Đính kèm cookie để lưu lại thông tin đăng nhập
* Mã hóa dữ liệu

## VueJS

### VueJS là gì ? VueJS là một Frontend Framework được viết bằng ngôn ngữ Javascript, cho phép tạo ra trang giao diện người dùng Single Page Application. VueJS là một trong những framework nổi tiếng được sử dụng để tạo ra ứng dụng web. Đối với VueJS các thành phần được viết thành các component sau đó ghép chúng lại để tạo ra một trang hoàn chỉnh. Với cách viết này, VueJS giúp các nhà phát triển có thể tái sử dụng lại code một cách dễ dàng.

Cấu trúc của file vue:

Một file vue thông thường sẽ có đuôi .vue và có định dạng như sau:

<template>

//

</template>

<script>

//

</script>

<style>

//

</style>

### Data:

Trong phần script của VueJS sẽ chứa một object được sử dụng làm data để render trong template:

<script>

export default {

data() {

return {

Name: ‘Hello world'

};

}

}

</script>

### Directives

VueJS sử dụng các thuộc tính đặc biệt được gọi là directives để "điều khiển" giao diện dựa

trên phần dữ liệu trong script.

Một số directive cơ bản:

* V-if, v-show: Thường được sử dụng để ẩn hiện UI
* v-for : Thường dùng để duyệt qua dữ liệu để tạo ra các component con
* V-bind: Dùng để gắn dữ liệu vào trong component theo cú pháp: v-bind:[attribute]: [value]
* V-html: Hiển thị các text có input là dạng HTML

### Methods

Trong mỗi file .vue ta có thể định nghĩa các hàm thực hiện một chức năng bất kì để có thể sử dụng lại nó khi cần. Các hàm này có thể được gọi cả trong phần script lẫn phần template:

<script>

export default {

Methods: {

async getData() {

const response = await fetchDataHere(‘https://jsonplaceholder.com/posts');

console.log(response)

}

}

</script>

Với ví dụ trên method getData sẽ tiến hành fetch dữ liệu từ trang <https://jsonplaceholder.com/posts> sau đó sẽ tiến hành in ra dữ liệu vừa fetch được

### Computed

Computed Properties có thể hiểu là nó khai báo giống như 1 methods nhưng lại được truy

cập giống như cách chúng ta truy cập data của component:

<script>

export default {

data() {

return {

numbers: [5, 8, 3]

}

},

computed: {

numberTotal() {

Return this.numbers.reduce((sum, val) => sum + val);

}

}

}

</script>

### Watchers

Watchers cho phép theo dõi sự thay đổi thuộc tính trong data hoặc trong computed property.

<script>

export default {

data() {

return {

count: 0,

}

},

watch: {

count(newValue, oldValue) {

console.log(“Bien count da thay doi”);

}

}

}

</script>

### Filters

Filters đóng vai trò giống một bộ format cho string hoặc số.

<template>

<div id="app">

<p :data-alt="name|cAPItalize">{{name|capacity }}</p>

</div>

</template>

<script>

export default {

data() {

return {

name: “foo”,

}

}, filters: {

capacity(value) {

if (!value) return ““

value = value.toString()

Return value.charAt(0).toUpperCase() + value.slice(1)

}

}

}

</script>

### Props

Ứng dụng VueJS được xây dựng dựa trên các components khác nhau và hơn nữa các

components còn có thể truyền data của nó xuống component con bằng props.

### Events

Để có thể thực hiện các thao tác như click trong ứng dụng ta sẽ sử dụng thêm một directives

khác đó là v-on. Cú pháp sẽ là v-on:[event]="[hander]"

### Slot

Slot là cách mà ta có thể sử dụng để tạo ra các component khung với nội dung có thể thay

đổi tùy theo vị trí mà ta muốn sử dụng nó

## Supervisor

### Supervisor là gì

Supervisor là công cụ giúp quản lý giám sát các tiến trình đang chạy trên Linux. Supervisor sẽ giúp một tiến trình ở trong hệ thống có thể chạy liên tục, tạo ra các log giúp người vận hành giám sát được các công việc đã và đang thực hiện. Khi có các lỗi xảy ra thì supervisor sẽ tiến thành thử lại, các thông số này có thể được cấu hình.

### Cài đặt supervisor

Để cài Supervisor, ta chạy lệnh sau:

sudo apt install supervisor

### Cấu hình supervisor để giám sát laravel queue

Mỗi tiến trình do Supervisor giám sát sẽ được cấu hình ở file dạng .conf nằm trong thư mục /etc/supervisor/conf.d

Cấu trúc 1 file config cơ bản sẽ như sau:

[program:laravel-worker]

process\_name=%(program\_name)s\_%(process\_num)02d

directory=/home/luccui/admin

command=/home/luccui/admin/php artisan queue:work –tries=3

autostart=true

autorestart=true

user=root

numprocs=4

redirect\_stderr=true

stdout\_logfile=/home/luccui/admin/storage/logs/worker.log

Sau khi cấu hình xong ta chạy lệnh sau:

systemctl restart supervisord

# ĐỐI TƯỢNG VÀ PHƯƠNG PHÁP NGHIÊN CỨU

## Đối tượng nghiên cứu

## Phương pháp nghiên cứu

## Đạo đức nghiên cứu

# KẾT QUẢ NGHIÊN CỨU VÀ THẢO LUẬN

## Tổng quan về mô hình hàng đợi tin nhắn RabbitMQ

### AMQP là gì ?

RabbitMQ là một message broker ( message-oriented middleware) sử dụng giao thức AMQP – Advanced Message Queue Protocol. Nó được viết bằng ngôn ngữ lập trình Erlang. RabbitMQ chạy được trên đa hệ điều hành, nó cung cấp các công cụ và trình quản lý giao diện cực kỳ trực quan cho người lập trình để giám sát và một phương tiện trung gian để giao tiếp giữa nhiều thành phần trong một hệ thống lớn. RabbitMQ là trung gian nhận các tin nhắn từ nhiều thành phần của hệ thống sau đó sẽ đưa đến các thành phần chịu trách nhiệm xử lý để xử lý. Trong một hệ thống phân tán, việc giao tiếp giữa các thành phần với nhau gần như là bắt buộc, nhưng nếu có quá nhiều giao tiếp mà các liên kết này quá chặt chẽ sẽ làm cho ứng dụng trở nên phức tạp, việc bảo trì ứng dụng trở thành một việc đặt biệt khó khăn. Khi các thành phần trong ứng dụng giao tiếp và liên kết với nhau quá nhiều thì sẽ rất khó khăn trong việc triển khai code. Một khi một thành phần bị lỗi thì nó sẽ ảnh hưởng đến tất cả các thành phần mà lệ thuộc vào nó. Giải pháp ở đây là thay thế các liên kết trực tiếp, để các thành phần này được giao tiếp với nhau thì phải sử dụng một giao thức trung gian qua một message broker. Với sự tham gia của một message broker các producer có nhiệm vụ chính là đưa tin nhắn vào trong message queue, từ message queue này message broker có nhiệm vụ phân phát đến các consumer chịu trách nhiệm xử lý message đó. Vì là producer giao tiếp các consumer trung gian thông qua giao tiếp trung là là message broker nên ở hai thành phần này chúng ta hoàn toàn có thể viết được bằng các ngôn ngữ khác nhau, do đó thì sẽ tận dụng được ưu thế của từng ngôn ngữ lập trình để phát triển hệ thống phù hợp với từng yêu cầu. Một đặc tính quan trọng của RabbitMQ là các tác vụ này sẽ xảy ra bất đồng bộ. Các producer không thể biết được consumer nào sẽ được giao để xử lý tác vụ này cũng như không thể nào biết được khi nào tác vụ này được hoàn thành. Đối với producer, nhiệm vụ chính chỉ là đưa tin nhắn vào trong hàng đợi còn việc phân phát tin nhắn đến đâu thì việc này do RabbitMQ chịu trách nhiệm. Một RabbitMQ được xem như là một trung gian hoạt động của dịch vụ của hệ thống, nó có thể giảm được độ trễ và tốc độ tải của một ứng dụng bằng cách ủy thác các tác vụ cho các hệ thống bên thứ ba.



### Sự tách biệt thực sự giữa Bên Event Publish và bên nhận thông tin

Hệ thống sử dụng kiến ​​trúc hướng sự kiện tách các thành phần trong hệ thống, phân tách quyền sở hữu dữ liệu theo miền:

* Bên Event publish không cần quan tâm đến việc các sản phẩm mà họ sản xuất sẽ được tiêu thụ như thế nào (vì vậy có thể thêm bên nhận thông tin mà không ảnh hưởng đến Bên Event Publish ).
* Bên nhận thông tin không cần quan tâm đến việc chúng được sản xuất như thế nào.

Do khớp nối lỏng lẻo này, các hệ thống microservices có thể được thực hiện bằng các ngôn ngữ khác nhau hoặc sử dụng các công nghệ khác nhau và phù hợp với các công việc cụ thể. Do đó, việc mã hóa dữ liệu sự kiện không quan trọng – nó có thể là JSON, XML, Avro, v.v.

Việc tách các thành phần của một ứng dụng cũng cho phép chúng được chia tỷ lệ dễ dàng và độc lập với nhau trên toàn mạng. Các nhà phát triển có thể sửa đổi hệ thống của họ bằng cách thêm hoặc xóa động các nhà sản xuất sự kiện và bên nhận thông tin mà không cần thay đổi bất kỳ logic nào trong bất kỳ dịch vụ nào.

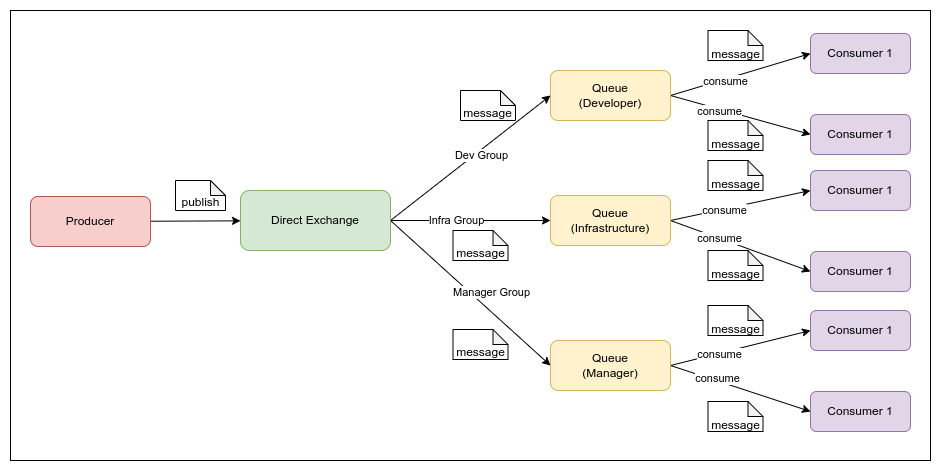
### Các loại exchange trong RabbitMQ

Direct Exchange

Trong mô hình hàng đợi hàng đợi tin nhắn, direct exchange được dùng để vận chuyển các tin nhắn ở trong hàng đợi sau đấy sẽ đưa đến các consumer bằng cách dựa vào các routing key. Loại exchange này thường thấy nhất sẽ sử dụng ở trong các trường hợp định tuyến tin nhắn đơn hướng mặc dù nó có thể sử dụng cho định tuyến đa hướng.

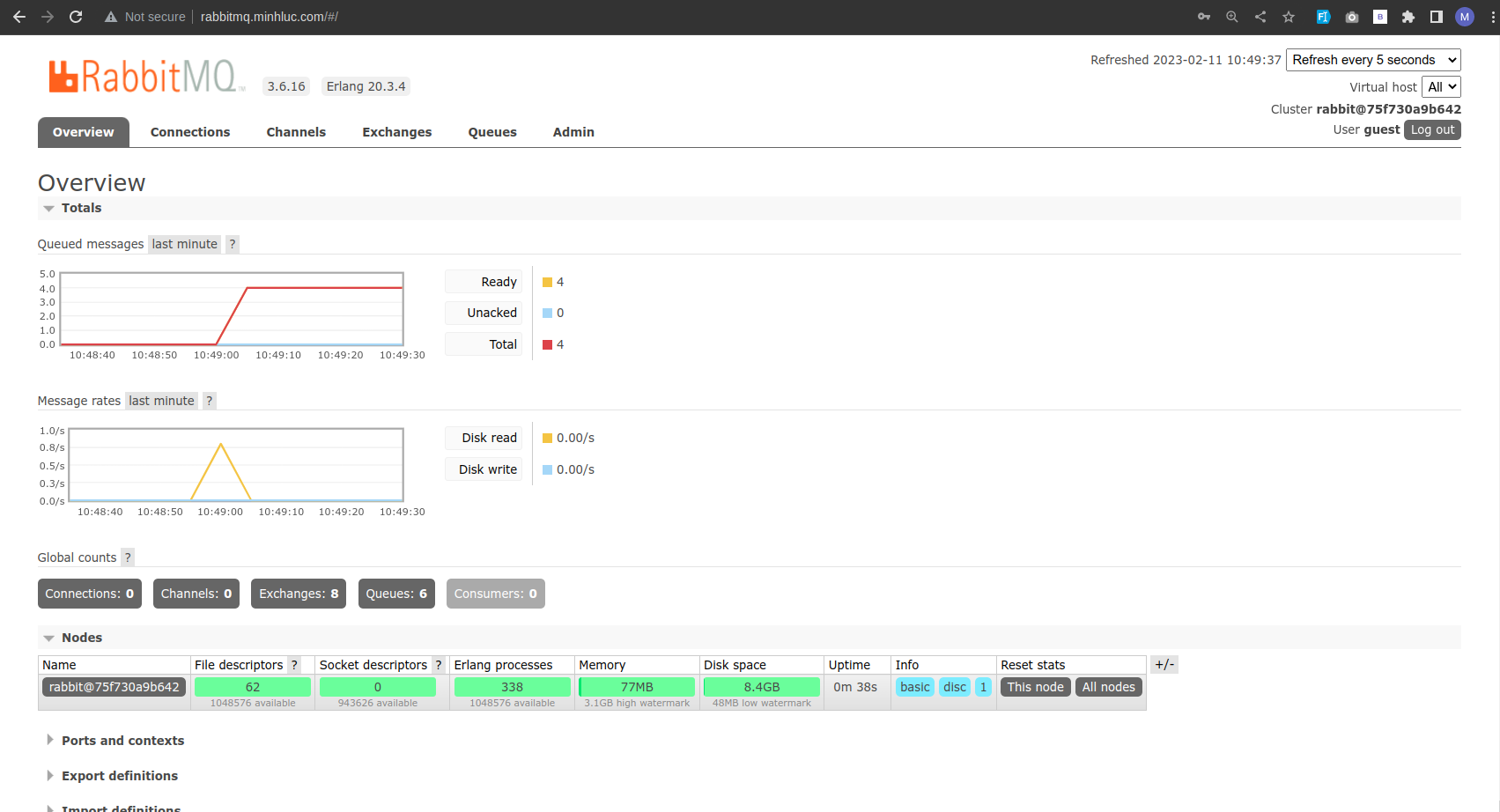
Luồng đi của một Direct exchange sẽ như sau:

* Một producer sau khi tạo ra một message thì sẽ đẩy message này tới exchange.
* Một hàng đợi sẽ tiến hành binding tới exchange này bằng cách sử dụng routing key. Ta có thể hoàn toàn tạo ra nhiều queue và binding chúng tới exchange và cũng có thể sử dụng chung routing key hoặc là khác routing key.
* Một tin nhắn khi được đẩy vào trong exchange có gắn kèm routing key, dựa theo thông tin routing key này thì tin nhắn sẽ được đưa đến một hoặc là nhiều hàng đợi đã đăng ký trước đó



**Default Exchange**

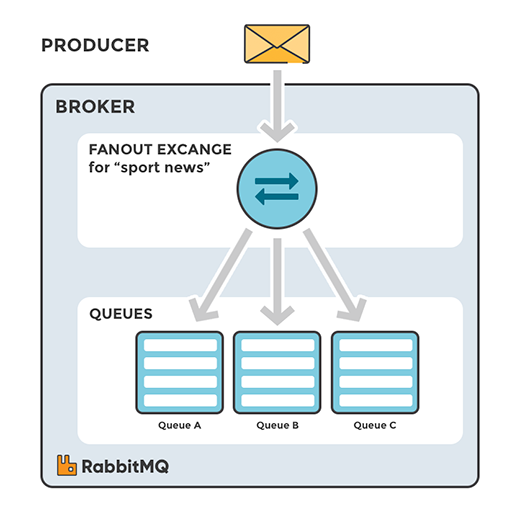
Trong rabbitMQ thì mỗi exchange sẽ không được phép đặt tên trùng nhau, default exchange thực chất cũng là một direct exchange nhưng không có tên. Nếu mà lựa chọn default exchange thì các tin nhắn sẽ được đẩy đến các hàng đợi cùng tên với routing key. Khi mà routing key có tên giống với tên của hàng đợi thì mỗi khi có tin nhắn mới thì ngay lập tức sẽ được liên kết và trao đổi.



**Fanout Exchange**

Fanout exchange sẽ vận chuyển các tin nhắn đến tất cả các queue mà ở xung quanh nó. Các tin nhắn sẽ được phân tán ra khắp các queue ngoại trừ routing key hoàn toàn sẽ bị bỏ qua.

Fanout exchange hữu ích khi mà một tin nhắn hay thông điệp cần chuyển đến một hay là nhiều queue và mỗi message sẽ có cách xử lý tin nhắn khác nhau. Giống như có một việc gì đó xảy ra chẳng hạn như một dự báo thời tiết hay là một sự kiện gì đó thì tất cả các thiết bị di động được kết nối sẽ được thông báo.



Topic exchange

Topic exchange định tuyến tới một hoặc nhiều hàng đợi dựa trên sự trùng khớp giữa routing key và pattern. Topic exchange thường được sử dụng để thực hiện định tuyến tin nhắn phát đa hướng. Một số ví dụ về các trường hợp sử dụng:

* Phân phối dữ liệu liên quan đến vị trí địa lý cụ thể.
* Xử lý tác vụ nền được thực hiện bởi nhiều workers, mỗi công việc có khả năng xử lý các nhóm tác vụ cụ thể.
* Cập nhật tin tức liên quan đến phân loại hoặc gắn thẻ (ví dụ: chỉ dành cho một môn thể thao hoặc đội cụ thể).
* Điều phối các dịch vụ của các loại khác nhau trong cloud

**Header exchange**

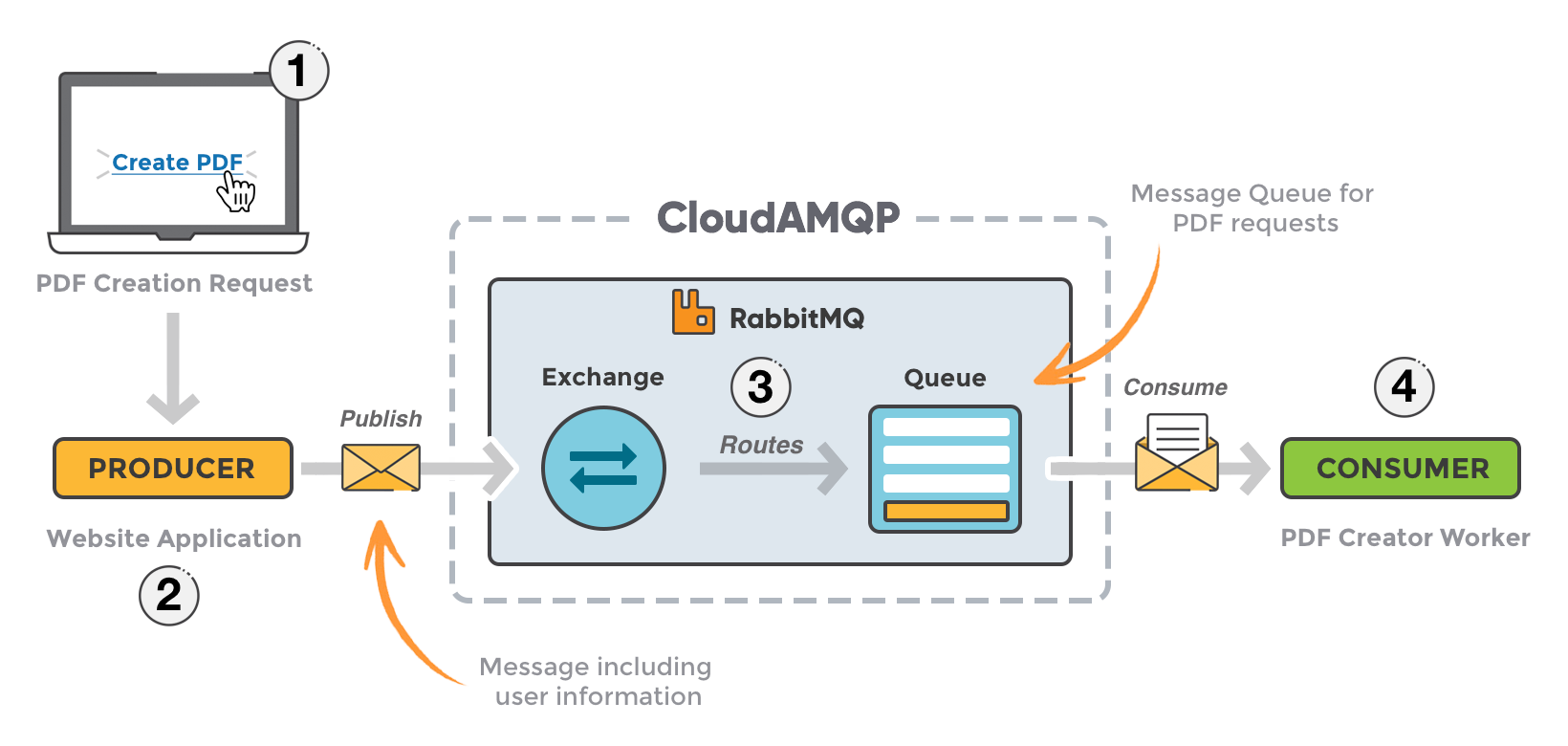
Là một hệ thống định tuyến sử dụng các đối số của header và các giá trị tùy chọn để định tuyến tin nhắn đến các hàng đợi liên quan. Header exchange cũng tương như topic exchange ngoại trừ thay vì sử dụng các routing key đối với header exchange thì sẽ sử dụng giá trị từ header để định tuyến. Nếu mà giá trị của nguồn khớp với giá trị của của header thì tin nhắn sẽ được liên kết và trao đổi với nhau.

Trong liên kết giữa exchange và hàng đợi, một đối số cụ thể được gọi là x-match sẽ cho biết tất cả các header phải khớp hay là chỉ duy nhất một header được phép. Thuộc tính x-match sẽ có giá trị có thể là any hoặc là all, all sẽ là giá trị mặc định. Giá trị của “all" cho biết rằng tất cả các cặp tiêu đề (key, value) phải khớp, trong khi “all" cho biết rằng ít nhất một cặp phải khớp. Thay vì một chuỗi, các tiêu đề có thể được tạo với nhiều loại dữ liệu hơn, chẳng hạn như số nguyên hoặc giá trị băm. Loại trao đổi tiêu đề (khi được sử dụng với tùy chọn liên kết “all”) hữu ích cho việc điều hướng các thông báo có chứa một tập hợp con các tiêu chí đã biết (không có thứ tự).

### Khi nào vào tại sao nên sử dụng RabbitMQ

Hàng đợi tin nhắn cho phép các máy chủ web phản hồi các yêu cầu một cách nhanh chóng thay vì bị buộc phải thực hiện các quy trình tốn nhiều tài nguyên ngay tại chỗ có thể làm chậm thời gian phản hồi. Hàng đợi tin nhắn cũng hữu ích khi muốn phân phối một tin nhắn cho nhiều consumer hoặc để cân bằng tải giữa các worker .

Ví dụ: Các consumer lấy một tin nhắn ra khỏi hàng đợi và bắt đầu xử lý tệp PDF. Đồng thời, producer đang xếp hàng các tin nhắn mới. Yêu cầu có thể được tạo bằng một ngôn ngữ lập trình và được xử lý bằng ngôn ngữ lập trình khác.



1. Người dùng gửi yêu cầu tạo PDF tới ứng dụng web.
2. Ứng dụng web (producer) gửi một thông báo tới RabbitMQ bao gồm dữ liệu từ yêu cầu như tên và email.
3. Một exchange chấp nhận các tin nhắn từ producer và định tuyến chúng đến đúng hàng đợi tin nhắn để tạo file PDF.
4. Các worker xử lý PDF (consumer) nhận được thông báo tác vụ và bắt đầu xử lý PDF.

### Hướng dẫn cài đặt và sử dụng RabbitMQ

Trong bài hướng dẫn này sẽ sử dụng docker để cài đặt RabbitMQ, một trong lợi ích cực kỳ lớn của docker là

Trang Docker Hub của RabbitMQ cung cấp cho các Docker Image bao gồm cả Management plugin sử dụng Alpine Linux (1 Linux version được tối ưu để có dung lượng rất nhỏ). Chúng ta có thể sử dụng những tag có tên kết thúc với management-alpine để cài đặt RabbitMQ sử dụng Docker Compose.

Tạo file cấu hình docker-compose.yml với nội dung sau:

version: '3.9'

services:

rabbitmq:

image: rabbitmq:3-management-alpine

container\_name: 'rabbitmq'

ports:

- 5672:5672

- 15672:15672

volumes:

- /home/luccui/workspace/laravel/docker/data:/var/lib/rabbitmq

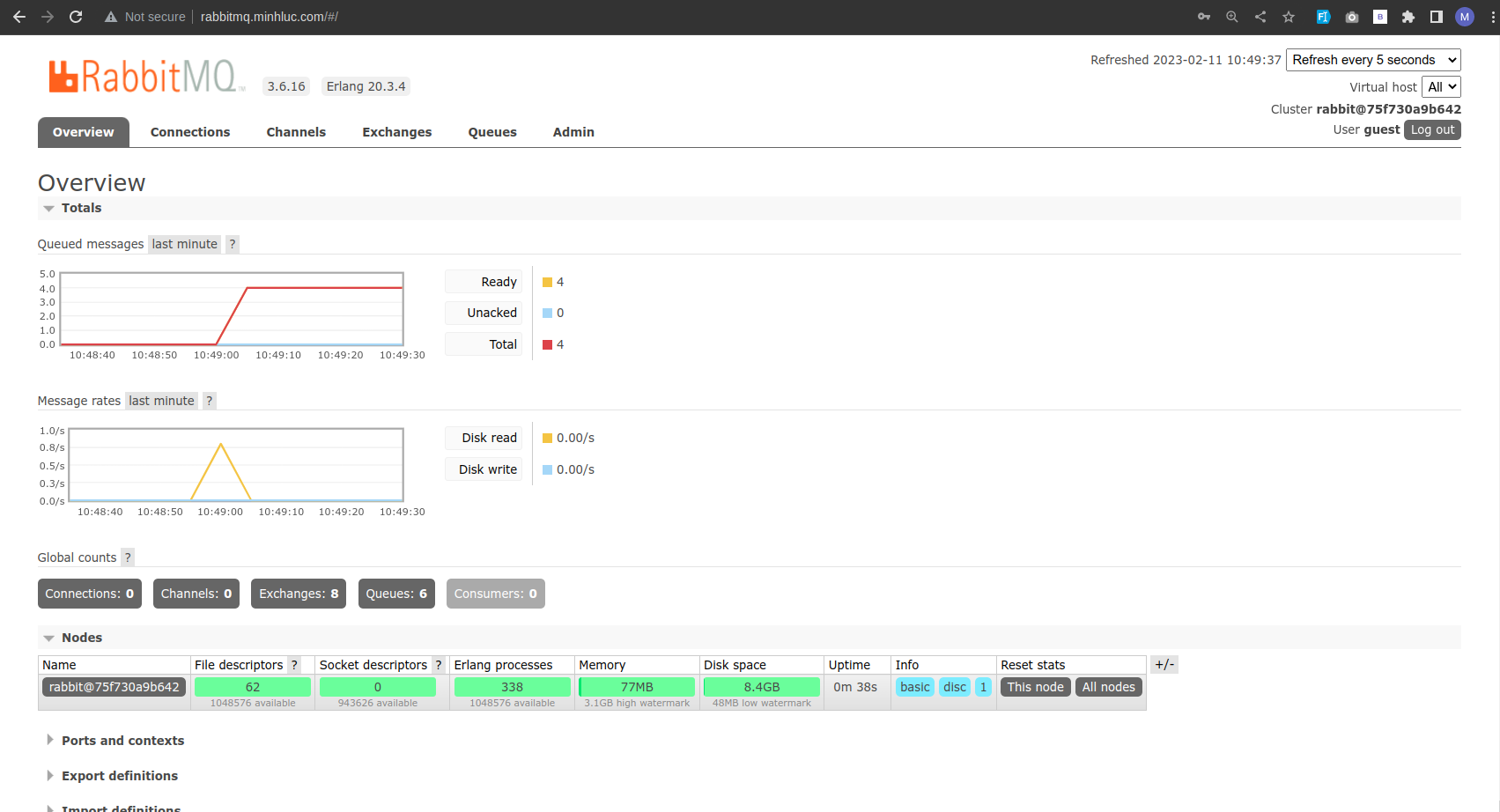
- /home/luccui/workspace/laravel/docker/log:/var/log/rabbitmq

Ở đây, đang cài đặt latest version của RabbitMQ 3.9.15. Chúng ta cần expose 2 port 5672 và 15672 của RabbitMQ server và RabbitMQ Management UI để cho bên ngoài sử dụng. 2 thư mục bên trong container của RabbitMQ cũng được mount ra các thư mục bên ngoài để có thể dễ dàng làm việc với RabbitMQ.

Kết quả khi chạy command “docker-compose up” trong thư mục chứa tập tin docker-compose.yaml với nội dung trên như sau:

Như vậy là đã start được một RabbitMQ server cùng với management UI. Bây giờ thì ta đã có thể access tới RabbitMQ Management UI sử dụng địa chỉ <http://localhost:15672/>:

Đăng nhập với username và password là guest, ta sẽ thấy kết quả như sau:

****

**Sử dụng RabbitMQ trên ứng dụng Laravel**

Trước hết ta cần phải cài đặt thư viện bằng dòng lệnh sau:

composer require vladimir-yuldashev/laravel-queue-rabbitmq

Package sẽ tự động được đăng ký trong ứng dụng Laravel

Thêm connection vào trong file config/queue.php

'connections' => [

// ...

'rabbitmq' => [

'driver' => 'rabbitmq',

'queue' => env('RABBITMQ\_QUEUE', 'default'),

'connection' => PhpAmqpLib\Connection\AMQPLazyConnection::class,

'hosts' => [

[

'host' => env('RABBITMQ\_HOST', '127.0.0.1'),

'port' => env('RABBITMQ\_PORT', 5672),

'user' => env('RABBITMQ\_USER', 'guest'),

'password' => env('RABBITMQ\_PASSWORD', 'guest'),

'vhost' => env('RABBITMQ\_VHOST', '/'),

],

],

'options' => [

'ssl\_options' => [

'cafile' => env('RABBITMQ\_SSL\_CAFILE', null),

'local\_cert' => env('RABBITMQ\_SSL\_LOCALCERT', null),

'local\_key' => env('RABBITMQ\_SSL\_LOCALKEY', null),

'verify\_peer' => env('RABBITMQ\_SSL\_VERIFY\_PEER', true),

'passphrase' => env('RABBITMQ\_SSL\_PASSPHRASE', null),

],

'queue' => [

'job' => VladimirYuldashev\LaravelQueueRabbitMQ\Queue\Jobs\RabbitMQJob::class,

],

],

/\*

\* Set to "horizon" if you wish to use Laravel Horizon.

\*/

'worker' => env('RABBITMQ\_WORKER', 'default'),

'after\_commit' => false,

],

// ...

],

Ta hoàn toàn có thể tùy chỉnh một số cấu hình của RabbitMQ như sau:

* Khi mà ta muốn trì hoãn các message được xử lý ta có thể thêm vào trong cấu hình như sau:

'options' => [

'queue' => [

// ...

'prioritize\_delayed' => false,

'queue\_max\_priority' => 10,

],

],

]

* Khi muốn publish message dưới dạng exchange routing key thì có thể sử dụng cấu hình tham khảo dưới đây:

'options' => [

'queue' => [

// ...

'exchange' => 'application-x',

'exchange\_type' => 'topic',

'exchange\_routing\_key' => '',

],

]

* Trong Laravel, các công việc thất bại được lưu trữ trong CSDL. Nhưng ta có thể hoàn toàn chỉ định một số quy trình khác cũng làm điều gì đó với thông báo.Khi muốn hướng dẫn RabbitMQ định tuyến lại các message thất bại đến một exchange hoặc một queue cụ thể, thì điều này có thể thực hiện được bằng cách thêm các tùy chọn bổ sung.

'options' => [

'queue' => [

'reroute\_failed' => true,

'failed\_exchange' => 'failed-exchange',

'failed\_routing\_key' => 'application-x.%s',

],

]

## Mô tả ứng dụng

Một cửa hàng thời trang đang phát triển và có dấu hiệu người dùng tăng lên đột hàng ngày có thể xử lý được khoảng 10000 đơn hàng cùng một lúc muốn xây dựng một website bán hàng thời trang online với mục tiêu sau:

Cửa hàng sẽ bán nhiều loại mặt hàng thời trang khác nhau có thể là quần áo, giày dép, các phụ kiện thời trang,.... Để quản lý được các mặt hàng thuộc nhóm nào thì mỗi mặt hàng được được gom vào trong một loại danh mục khác nhau. Thông tin của danh mục gồm mã và tên danh mục.

Thông thường các loại sản phẩm thời trang bán ra sẽ có hai loại chính là sản phẩm không có biến thể tức là sản phẩm này sẽ không có các tùy chọn về kích cỡ hoặc là màu sắc của sản phẩm. Thông tin chi tiết của sản phẩm gồm mã sản phẩm, tên sản phẩm, mô tả ngắn về sản phẩm, nội dung mô tả chi tiết về sản phẩm này, hình ảnh minh họa sản phẩm, sản phẩm này có thuộc loại hot, mới nhất hay không. Đối với sản phẩm có nhiều biến thế thì sẽ có biến thể về màu sắc và kích cỡ. Một sản phẩm này sẽ có một hoặc nhiều màu sắc và một hoặc nhiều kích cỡ. Các màu sắc và kích cỡ được gom lại thành một biến thể của sản phẩm và từng biến thể này sẽ có một mức giá cũng như một mã nhất định. Khi mà sản phẩm không còn hàng hoặc số lượng đặt hàng quá nhiều so với cửa hàng đang có thì website vẫn hiển thị nhưng hệ thống sẽ đưa ra gợi ý cho khách hàng thông tin liên hệ mặt hàng này. Thông tin liên hệ mặt hàng sẽ gồm mã sản phẩm, thông tin liên hệ như tên, email, số điện thoại, ngày liên hệ. Các thông tin này sau khi được khách hàng liên hệ thì bên cửa hàng sẽ có trách nhiệm gọi đến để tư vấn về sản phẩm. Khách hàng khi mua sắm tại website thì sẽ tiến hành đăng ký tài khoản bằng số điện thoại sau khi đã đăng ký thành công thì hệ thống sẽ tiến hành gửi mã xác thực số điện thoại có hợp lệ hay không sau khi đã xác nhận hợp lệ thì hệ thống sẽ thêm thông tin khách hàng vào trong CSDL thông tin này sẽ bao gồm: mã khách hàng, họ tên khách hàng, email, số điện thoại. Khi khách hàng muốn mua sắm thì sẽ tiến hành đăng nhập và tiến hành thêm các sản phẩm vào trong giỏ hàng. Mỗi khách hàng có thể thêm được nhiều loại sản phẩm. Sau khi đã lựa chọn được hết tất cả các sản phẩm muốn mua thì sẽ tiến hành thanh toán cho đơn hàng mà mình đã đặt.

Mỗi khách hàng sẽ được phân vào trong từng nhóm khách hàng. Từng nhóm khách hàng này sẽ được sắp xếp theo từng mức độ mua hàng của từng khách hàng, thông tin về nhóm khách hàng này sẽ bao gồm, tên nhóm khách hàng, và nhóm khách hàng này có thể thuộc về một nhóm khách hàng nào đó. Các nhóm khách hàng này sẽ được ưu đãi một số giảm giá về sản phẩm. Khi có một sự kiện nào đó để kích thích sự mua sắm của khách hàng thì hệ thống sẽ tạo ra một chương trình giảm giá có thể sẽ là giảm giá theo phần trăm của đơn hàng hoặc là giảm giá theo giá trị đơn hàng. Sau khi đã tạo ra được chương trình giảm giá thì hệ thống sẽ tiến hành gửi email ưu đãi đến từng khách hàng nhận được ưu đãi. Thông tin về giảm giá này sẽ gồm: mã giảm giá, loại giảm giá, giá trị giảm giá, ngày sẽ bắt đầu áp dụng giảm giá này. Đối với những khách hàng không thuộc bất kì ưu đãi này cũng có thể nhận được mã giảm giá từ các chương trình mà hệ thống tạo ra. Thông tin về mã giảm giá sẽ gồm: mã mã giảm giá, tên chương trình giảm giá, mã giảm giá, ngày bắt đầu, ngày kết thúc, loại giảm giá, giá trị giảm.

Để tăng lượng tương tác website thì bên cửa hàng cũng sẽ tạo ra những bài đánh giá về sản phẩm hay là một số bài viết gioiws thiệu về sản phẩm. Thông tin về bài viết bao gồm mã bài viết, tên bài viết, nội dung bài viết để tăng chất lượng hiển thị hệ thống cũng có một số tiêu chí về seo giúp hiển thị được top tìm kiếm trên google. Các bài viết này có thể thuộc về nhiều loại danh mục và sẽ có nhiều loại hashtag khác nhau.

## Yêu cầu hệ thống

### Yêu cầu chức năng

* Xem danh sách sản phẩm bán
* Tìm kiếm sản phẩm
* Phân loại sản phẩm theo danh mục, nhà cung cấp, quy cách
* Thêm sản phẩm vào trong giỏ hàng
* Xóa sản phẩm trong giỏ hàng
* Xem thông tin giỏ hàng
* Đặt hàng
* Theo dõi đơn hàng cho khách không đăng ký tài khoản
* Thanh toán đơn hàng
* Quản trị hệ thống
* Duyệt đơn hàng
* Chức năng thống kê sản phẩm mua nhiều nhất, xem nhiều nhất
* Chức năng báo cáo doanh thu theo tuần, tháng, năm

### Yêu cầu phi chức năng

* Giao diện thân thiện, dễ sử dụng
* Tốc độ truy xuất nhanh
* Hạn chế đến mức thấp nhất các sai sót có thể xảy ra trong quá trình sử dụng và có khả năng mở rộng và tích hợp các tính năng mới
* CSDL kết nối chính xác và toàn vẹn dữ liệu

## Mô hình CSDL

Mô tả chi tiết các bảng

Bảng customers

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | Bigint | Khóa chính | Mã tài khoản |
| 2 | first\_name | varchar(50) |  | Tên hiển thị |
| 3 | last\_name | varchar(50) |  | Địa chỉ email |
| 4 | email | varchar(100) |  | Ngày xác nhận email |
| 5 | phone | varchar(10) |  | Mật khẩu |
| 6 | password | varchar(255) |  | Token lưu đăng nhập |
| 7 | verify\_token | varchar(100) |  | Mã để xác thực |
| 8 | is\_verified | tinyint |  |  |
| 9 | address | varchar(100) |  |  |
| 10 | country | varchar(100) |  |  |
| 11 | city | varchar(100) |  |  |
| 12 | ship\_address | varchar(255) |  |  |
| 13 | created\_at | timestamp |  |  |
| 14 | updated\_at | timestamp |  |  |

*Mô tả bảng brands*

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã danh mục |
| 2 | name | varchar(100) |  | Tên danh mục |
| 6 | created\_at | timestamp |  | Ngày tạo |
| 7 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

*Mô tả bảng categories*

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã nhà cung cấp |
| 2 | name | varchar(100) |  | Tên nhà cung cấp |
| 3 | created\_at | timestamp |  | Ngày tạo |
| 1 | updated\_at | timestamp |  | Ngày cập nhật gần  nhất |

*Mô tả bảng colors*

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã màu sắc |
| 2 | name | varchar(100) |  | Tên quy cách |
| 4 | created\_at | timestamp |  | Ngày tạo |
| 5 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

*Mô tả bảng size*

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã màu |
| 2 | name | varchar(100) |  | Tên màu |
| 4 | created\_at | timestamp |  | Ngày tạo |
| 5 | updated\_at | timestamp |  | Ngày cập nhât gần nhất |

*Mô tả bảng coupons*

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã giảm giá |
| 2 | name | varchar(2100) |  | Tên giảm giá |
| 3 | code | varchar(30) |  | Mã |
| 4 | from | varchar(100) |  | Ngày bắt đầu |
| 5 | to | varchar(100) |  | Ngày kết thúc |
| 6 | desc\_by | varchar(400) |  | Giảm theo |
| 7 | value | varchar(255) |  | Giá trị giảm |
| 8 | status | varchar(255) |  | Liên kết sản phẩm |
| 10 | created\_at | timestamp |  | Ngày tạo |
| 11 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã hình ảnh |
| 2 | duong\_dan | varchar(255) |  | Đường dẫn hình ảnh |
| 3 | created\_at | timestamp |  | Ngày tạo |
| 4 | updated\_at | timestamp |  | Ngày cập nhật |

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | ID giảm giá |
| 2 | ma\_giam\_gia | varchar(20) |  | Mã giảm giá |
| 3 | giam\_theo | varchar(50) |  | Giảm theo |
| 4 | gia\_tri | int |  | Giá trị giảm theo |
| 5 | ngay\_bat\_dau | timestamp |  | Ngày bắt đầu |
| 6 | ngay\_ket\_thuc | timestamp |  | Ngày kết thúc |
| 7 | created\_at | timestamp |  | Ngày tạo |
| 8 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã tồn kho |
| 2 | danhmuc\_id | bigint | Khóa ngoại | Mã danh mục |
| 3 | thuonghieu\_id | bigint | Khóa ngoại | Mã thương hiệu |
| 4 | nhacungcap\_id | bigint | Khóa ngoại | Mã nhà cung cấp |
| 5 | quycach\_id | bigint | Khóa ngoại | Mã quy cách |
| 6 | ten\_sp | varchar(200) |  | Mã sản phẩm |
| 7 | slug | varchar(255) |  | Đường dẫn sản phẩm |
| 8 | hinh\_anh | varchar(255) |  | Hình ảnh bìa |
| 9 | ma\_sp | varchar(20) |  | Mã sản phẩm |
| 10 | mo\_ta\_ngan | varchar(200) |  | Mô tả ngắn |
| 11 | mo\_ta | text |  | Mô tả chi tiết |
| 12 | gia\_sp | int |  | Giá sản phẩm |
| 13 | gia\_km | int |  | Giá khuyến mãi |
| 14 | noi\_bat | tinyint(4) |  | Là sản phẩm nổi bật |
| 15 | created\_at | timestamp |  | Ngày tạo |
| 16 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã tồn kho |
| 2 | sanpham\_id | bigint | Khóa ngoại | Mã sản phẩm |
| 3 | so\_luong | int |  | Số lượng |
| 4 | created\_at | timestamp |  | Ngày tạo |
| 5 | created\_at | timestamp |  | Ngày cập nhật gần nhất |

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã khách hàng |
| 2 | ho | varchar(30) |  | Họ khách hàng |
| 3 | ten | varchar(30) |  | Tên khách hàng |
| 4 | so\_dien\_thoai | varchar(11) |  | Số điện thoại |
| 5 | ten\_dang\_nhap | varchar(30) |  | Tên đăng nhập |
| 6 | email | varchar(50) |  | Email |
| 7 | password | varchar(100) |  | Mật khẩu |
| 8 | dia\_chi | varchar(100) |  | Địa chỉ |
| 9 | ma\_xa | varchar(255) |  | Mã xã |
| 10 | lan\_dang\_nhap\_cuoi | timestamp |  | Lần đăng nhập cuối |
| 11 | ngay\_xac\_thuc\_emai | timestamp |  | Ngày xác thực email |
| 12 | remember\_token | varchar(100) |  | Token lưu đăng nhập |
| 13 | created\_at | timestamp |  | Ngày tạo |
| 14 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

Bảng: danhgias (Đánh giá)

*Bảng 4‑13. Chi tiết bảng đánh giá*

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã đánh giá |
| 2 | sanpham\_id | bigint | Khóa ngoại | Mã sản phẩm |
| 3 | khachhang\_id | bigint | Khóa ngoại | Mã khách hàng |
| 4 | noi\_dung | varchar(255) |  | Nội dung |
| 5 | so\_sao | int |  | Số sao đánh giá |
| 6 | created\_at | timestamp |  | Ngày tạo |
| 7 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | ID đơn hàng |
| 2 | ma\_don\_hang | bigint |  | Mã đơn hàng |
| 3 | khachhang\_id | bigint | Khóa ngoại | Mã khách hàng |
| 4 | magiamgia\_id | bigint | Khóa ngoại | Mã giảm giá |
| 5 | ma\_xa | varchar(10) |  | Mã xã |
| 6 | phi\_giao\_hang | int |  | Phí vận chuyển |
| 7 | thanh\_tien | int |  | Thành tiền |
| 8 | tong\_tien | int |  | Tổng tiền |
| 9 | dia\_chi | varchar(100) |  | Địa chỉ |
| 10 | phuong\_thuc\_thanh\_toan | varchar(100) |  | Phương thức thanh toán |
| 11 | ghi\_chu | varchar(255) |  | Ghi chú |
| 12 | trang\_thai | varchar(100) |  | Trạng thái đơn hàng |
| 13 | created\_at | timestamp |  | Ngày tạo |
| 14 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

| **STT** | **Thuộc tính** | **Kiểu dữ liệu** | **Miền giá trị** | **Diễn giải** |
| --- | --- | --- | --- | --- |
| 1 | id | bigint | Khóa chính | Mã chi tiết đơn hàng |
| 2 | donhang\_id | bigint | Khóa ngoại | Mã đơn hàng |
| 3 | sanpham\_id | bigint | Khóa ngoại | Mã sản phẩm |
| 4 | so\_luong | int |  | Số lượng |
| 5 | don\_gia | int |  | Đơn giá |
| 6 | thanh\_tien | int |  | Thành tiền |
| 7 | created\_at | timestamp |  | Ngày tạo |
| 8 | updated\_at | timestamp |  | Ngày cập nhật gần nhất |

## 

## 

## 

## 

## 

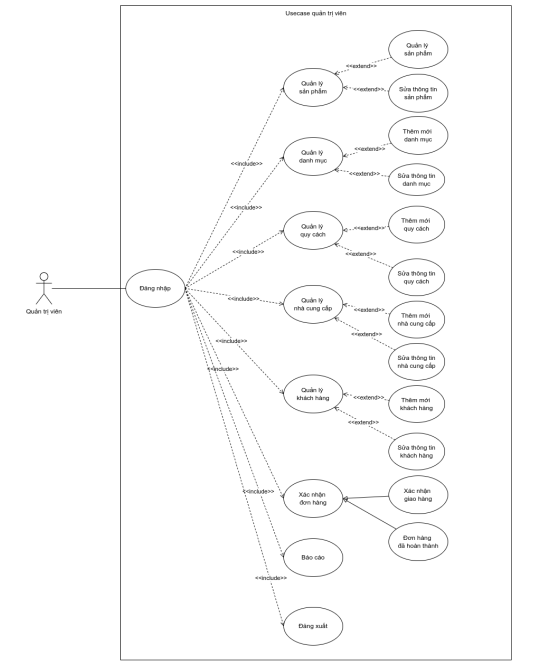
## 

## Mô hình xử lý

### Use Case khách hàng



### UseCase quản trị viên



## Mô hình tuần tự

## Mô tả hoạt động

## Cài đặt và thử nghiệm

### Thiết kế giao diện

# KẾT LUẬN