Course - Laravel Framework

Boardcast

Trong nhiều ứng dụng web hiện đại, WebSockets được sử dụng để triển khai các giao diện người dùng cập nhật trực tiếp, theo thời gian thực. Khi một số dữ liệu được cập nhật trên máy chủ, thông báo thường được gửi qua kết nối WebSocket để máy khách xử lý.

Tags: laravel boardcast, laravel

WebSockets cung cấp một giải pháp thay thế hiệu quả hơn để liên tục thăm dò máy chủ ứng dụng của bạn để biết các thay đổi dữ liệu sẽ được phản ánh trong giao diện người dùng của bạn.

Ví dụ: hãy tưởng tượng ứng dụng của bạn có thể xuất dữ liệu của người dùng sang tập tin CSV và gửi nó qua email cho họ. Tuy nhiên, việc tạo tập tin CSV này mất vài phút nên bạn chọn tạo và gửi CSV qua thư trong một job đã được xếp hàng đợi . Khi CSV đã được tạo và gửi tới người dùng, chúng tôi có thể sử dụng tính năng phát event để gửi một event `App\Events\UserDataExported` được nhận bằng JavaScript của ứng dụng của chúng ta. Sau khi nhận được event, chúng ta có thể hiển thị thông báo cho người dùng rằng CSV của họ đã được gửi qua email cho họ mà họ không cần phải làm mới trang.

Để hỗ trợ bạn xây dựng các loại tính năng này, Laravel giúp bạn dễ dàng "phát" các event Laravel phía máy chủ của bạn qua kết nối WebSocket. Việc phát các event Laravel của bạn cho phép bạn chia sẻ cùng một tên event và dữ liệu giữa ứng dụng Laravel phía máy chủ và ứng dụng JavaScript phía máy khách của bạn.

Các khái niệm cốt lõi đằng sau việc phát sóng rất đơn giản: máy khách kết nối với các kênh được đặt tên trên giao diện người dùng, trong khi ứng dụng Laravel của bạn truyền phát các event đến các kênh này trên phần phụ trợ. Những event này có thể chứa bất kỳ dữ liệu bổ sung nào bạn muốn cung cấp cho giao diện người dùng.

Trình điều khiển được hỗ trợ

Theo mặc định, Laravel bao gồm hai driver phát sóng phía máy chủ để bạn lựa chọn: `Pusher` và `Ably` . Tuy nhiên, các gói do cộng đồng điều khiển như `laravel-websockets` và `soketi` cung cấp driver phát sóng bổ sung mà không yêu cầu provider phát sóng thương mại nào.

> Trước khi đi sâu vào việc phát đi event, hãy đảm bảo rằng bạn đã đọc tài liệu của Laravel về events và listener.

Cài đặt phía máy chủ

Để bắt đầu sử dụng event boardcast của Laravel, chúng ta cần thực hiện một số cấu hình trong ứng dụng Laravel cũng như cài đặt một vài package. Quá trình event boardcast được thực hiện bởi driver phát sóng phía máy chủ, nó sẽ phát đi các event Laravel của bạn để cho Laravel Echo (một thư viện JavaScript) có thể nhận chúng trong ứng dụng trình duyệt khách. Đừng lo lắng - chúng tôi sẽ hướng dẫn từng bước từng phần của quá trình cài đặt.

Cấu hình

Tất cả cấu hình event boardcase của ứng dụng của bạn được lưu trữ trong tập tin cấu hình *config/broadcasting.php*. Laravel hỗ trợ một số driver boardcast: Pusher Channel, Redis và log - một driver để phát triển cục bộ và gỡ lỗi. Ngoài ra, driver `null` được đưa vào cho phép bạn tắt hoàn toàn tính năng boardcast trong quá trình thử nghiệm. Một cấu hình ví dụ được đưa vào cho mỗi driver này trong tập tin cấu hình *config/broadcasting.php*.

Nhà cung cấp dịch vụ boardcast

Trước khi boardcast bất kỳ event nào, trước tiên bạn sẽ cần đăng ký `App\Providers\BroadcastServiceProvider`. Trong các ứng dụng Laravel mới, bạn chỉ cần bỏ phần comment cho provider này trong mảng `providers` trên tập tin cấu hình `config/app.php` của mình. `BroadcastServiceProvider` này chứa mã cần thiết để đăng ký các route và callback xác thực boardcast.

Cấu hình hàng đợi

Bạn cũng sẽ cần phải chỉ định cấu hình và chạy một queue hàng đợi. Tất cả việc boardcast một event được thực hiện thông qua các job được xếp hàng để cho thời gian phản hồi của ứng dụng của bạn không bị ảnh hưởng nghiêm trọng bởi các event đang được boardcast.

Các kênh Pusher Channels

Nếu bạn định phát các event của mình bằng cách sử dụng Pusher Channels , bạn nên cài đặt SDK PHP của Pusher Channels bằng cách sử dụng chương trình quản lý package Composer như sau:

```
composer require pusher/pusher-php-server
```

Tiếp theo, bạn nên chỉ định cấu hình thông tin xác thực Pusher Channel của mình trong tập tin cấu hình *config/broadcasting.php*. Một cấu hình Pusher Channels mẫu đã được gộp sẵn trong tập tin này, cho phép bạn nhanh chóng chỉ định key, secret và ID ứng dụng của mình. Thông thường, các giá trị này phải được đặt thông qua các biến môi trường `PUSHER_APP_KEY`, `PUSHER_APP_SECRET` và `PUSHER_APP_ID`:

```
PUSHER_APP_ID=your-pusher-app-id
PUSHER_APP_KEY=your-pusher-key
```

```
PUSHER_APP_SECRET=your-pusher-secret
PUSHER_APP_CLUSTER=mt1
```

Cấu hình `pusher` của tập tin *config/broadcasting.php* cũng cho phép bạn chỉ định bổ sung `options` được các Kênh hỗ trợ, chẳng hạn như cluster.

Tiếp theo, bạn sẽ cần thay đổi driver của chương trình boardcast của mình thành `pusher` trong tập tin `.env` của mình:

```
BROADCAST_DRIVER=pusher
```

Cuối cùng, bạn đã sẵn sàng cài đặt và cấu hình Laravel Echo, nó sẽ nhận các boardcast events ở phía máy khách.

Các lựa chọn thay thế Pusher nguồn mở

Các package `laravel-websockets` và `soketi` cung cấp các máy chủ WebSocket tương thích với Pusher cho Laravel. Các package này cho phép bạn tận dụng toàn bộ sức mạnh của Laravel boardcast mà không cần đến nhà cung cấp WebSocket thương mại nào. Để biết thêm thông tin về cách cài đặt và sử dụng các gói này, vui lòng tham khảo tài liệu của chúng ta về các lựa chọn nguồn mở thay thế.

Ably

Nếu bạn định boardcast các event của mình bằng *Ably*, thì bạn nên cài đặt Ably PHP SDK bằng trình quản lý gói Composer:

```
composer require ably/ably-php
```

Tiếp theo, bạn nên cấu hình thông tin đăng nhập *Ably* của mình trong tập tin cấu hình *config/broadcasting.php*. Một cấu hình Ably ví dụ đã được cài sẵn trong tập tin này, cho phép bạn nhanh chóng có được key của mình. Thông thường, giá trị này phải được đặt thông qua biến môi trường `ABLY_KEY`:

```
ABLY_KEY=your-ably-key
```

Tiếp theo, bạn sẽ cần thay đổi driver của chương trình board của mình thành `ably` trong

tập tin *.env* của mình:

```
BROADCAST_DRIVER=ably
```

Cuối cùng, bạn đã sẵn sàng cài đặt và cấu hình Laravel Echo, nó sẽ nhận các boardcast event ở phía máy khách.

Các giải pháp bằng nguồn mở

PHP

Package ***laravel-websockets*** là một package WebSocket tương thích với PHP, Pusher dành cho Laravel. Gói này cho phép bạn tận dụng toàn bộ sức mạnh của boardcast trên Laravel mà không cần bất kỳ nhà cung cấp WebSocket thương mại nào. Để biết thêm thông tin về cách cài đặt và sử dụng gói này, vui lòng tham khảo tài liệu chính thức của nó.

Node

Soketi là một WebSocket server tương thích với Pusher dựa trên Node dành cho Laravel. Bên cạnh đó, *Soketi* sử dụng *µWebSockets.js* cho khả năng mở rộng và tốc độ xử lý cực cao. Package này cho phép bạn tận dụng toàn bộ sức mạnh của Laravel Boardcast mà không cần bất kỳ nhà cung cấp WebSocket thương mại nào. Để biết thêm thông tin về cách cài đặt và sử dụng package này, vui lòng tham khảo tài liệu chính thức của nó.

Cài đặt phía máy khách

Pusher Channels

Laravel Echo là một thư viện JavaScript giúp bạn đăng ký channel và theo dõi các event được phát đi bởi driver của boardcast trên máy chủ của bạn một cách dễ dàng. Bạn có thể cài đặt Echo thông qua NPM. Trong ví dụ này, chúng ta cũng sẽ cài đặt ***pusher-js*** vì chúng ta sẽ sử dụng service của Pusher Channels:

```
npm install --save-dev laravel-echo pusher-js
```

Sau khi Echo được cài đặt, bạn đã sẵn sàng tạo một phiên bản Echo mới trong JavaScript của ứng dụng của mình. Một nơi tuyệt vời để thực hiện việc này là ở cuối tập tin *resources/js*

/bootstrap.js được gộp vào trong framework Laravel. Mặc định, một cấu hình Echo mẫu đã được đặt sẵn trong tập tin này - bạn chỉ cần bỏ comment ra khỏi nó:

```
import Echo from 'laravel-echo';

window.Pusher = require('pusher-js');

window.Echo = new Echo({
  broadcaster: 'pusher',
  key: process.env.MIX_PUSHER_APP_KEY,
  cluster: process.env.MIX_PUSHER_APP_CLUSTER,
  forceTLS: true
});
```

Khi bạn đã bỏ comment và điều chỉnh cấu hình Echo theo nhu cầu của mình, bạn có thể build nội dung ứng dụng của mình:

```
npm run dev
```

> Để tìm hiểu thêm về cách build nội dung JavaScript của ứng dụng của bạn, vui lòng tham khảo tài liệu về Laravel Mix.

Sử dụng phiên bản client hiện có

Nếu bạn đã có phiên bản ứng dụng Pusher Channels được cấu hình sẵn mà bạn muốn Echo sử dụng, thì bạn có thể truyền nó cho Echo thông qua tùy chọn cấu hình `client`:

```
import Echo from 'laravel-echo';

const client = require('pusher-js');

window.Echo = new Echo({
  broadcaster: 'pusher',
  key: 'your-pusher-channels-key',
  client: client
});
```

Ably

Laravel Echo là một thư viện JavaScript giúp bạn đăng ký channel và theo dõi các event được phát đi bởi driver của boardcast trên máy chủ của bạn một cách dễ dàng. Bạn có thể cài đặt Echo thông qua NPM. Trong ví dụ này, chúng ta cũng sẽ cài đặt package ***pusher-js***.

Bạn có thể thắc mắc tại sao chúng ta cài đặt thư viện JavaScript ***pusher-js*** mặc dù chúng ta đang sử dụng Ably để phát đi các event của mình. Vì Ably bao gồm chế độ tương thích với Pusher cho phép chúng ta sử dụng giao thức Pusher khi theo dõi các event trong ứng dụng phía máy khách của chúng ta:

```
npm install --save-dev laravel-echo pusher-js
```

Trước khi tiếp tục, bạn nên bật hỗ trợ giao thức Pusher trong cài đặt ứng dụng Ably của mình. Bạn có thể bật tính năng này trong phần "Protocol Adapter Settings" của màn hình cài đặt ứng dụng Ably của bạn.

Sau khi Echo được cài đặt, bạn đã sẵn sàng tạo một phiên bản Echo mới trong JavaScript của ứng dụng của mình. Một nơi tuyệt vời để thực hiện việc này là ở cuối tập tin *resources/js/bootstrap.js* được bao gồm trong framework Laravel. Mặc định, một cấu hình Echo mẫu đã được cài đặt sãn trong tập tin này; tuy nhiên, cấu hình mặc định trong tập tin *bootstrap.js* là dành cho Pusher. Bạn có thể sao chép cấu hình bên dưới để chuyển đổi thành cấu hình của mình sang Ably:

```
import Echo from 'laravel-echo';

window.Pusher = require('pusher-js');
```

```
window.Echo = new Echo({
  broadcaster: 'pusher',
  key: process.env.MIX_ABLY_PUBLIC_KEY,
  wsHost: 'realtime-pusher.ably.io',
  wsPort: 443,
  disableStats: true,
  encrypted: true,
});
```

Lưu ý rằng cấu hình ***Ably Echo*** của chúng ta tham chiếu đến một biến môi trường `MIX_ABLY_PUBLIC_KEY`. Giá trị của biến này phải là key công khai Ably của bạn. Key công khai của bạn là một phần của key Ably xuất hiện trước ký tự `:`.

Khi bạn đã bỏ comment và điều chỉnh cấu hình Echo theo nhu cầu của mình, là bạn đã có thể build nội dung ứng dụng của mình:

```
npm run dev
```

> Để tìm hiểu thêm về cách build nội dung JavaScript của ứng dụng của bạn, vui lòng tham khảo tài liệu về Laravel Mix.

Tổng quan về khái niệm

Phát đi các event của Laravel cho phép bạn phát tán các event Laravel ở phía máy chủ của mình tới ứng dụng JavaScript ở phía máy khách của bạn bằng cách sử dụng WebSockets. Hiện tại, Laravel chuyển giao với Pusher Channels và Ably driver. Có thể dễ dàng sử dụng các event ở phía máy khách bằng cách sử dụng package JavaScript ***Laravel Echo***.

Sự kiện được phát qua "channel", có thể được chỉ định là công khai hoặc riêng tư. Bất kỳ chương trình ứng dụng khách nào truy cập vào ứng dụng của bạn cũng đều có thể đăng ký kênh công khai mà không cần bất kỳ xác thực hoặc ủy quyền nào; tuy nhiên, để đăng ký một kênh riêng tư, người dùng phải được xác thực và ủy quyền để theo dõi channel đó.

> Nếu bạn muốn khám phá các lựa chọn thay thế mã nguồn mở cho Pusher, hãy xem phần Các lựa chọn thay thế mã nguồn mở.

Sử dụng một ứng dụng mẫu

Trước khi đi sâu vào từng thành phần của event boardcast, chúng ta hãy tìm hiểu một cách tổng quát cách sử dụng cửa hàng thương mại điện tử làm ví dụ.

Trong ứng dụng của chúng ta, giả sử chúng ta có một trang cho phép người dùng xem trạng thái giao hàng cho các đơn đặt hàng của họ. Cũng giả sử rằng một event có tên `OrderShipmentStatusUpdated` được kích hoạt khi ứng dụng xử lý cập nhật trạng thái giao hàng:

```
use App\Events\OrderShipmentStatusUpdated;


OrderShipmentStatusUpdated::dispatch($order);
```

Chức năng của `ShouldBroadcast` interface

Khi người dùng đang xem một trong các đơn đặt hàng của họ, chúng ta không muốn họ phải làm mới trang để xem cập nhật trạng thái. Thay vào đó, chúng ta muốn phát đi các bản cập nhật cho ứng dụng khi chúng đang được xem bởi người dùng. Vì vậy, chúng ta cần đánh dấu event `OrderShipmentStatusUpdated` bằng interface `ShouldBroadcast`. Điều này sẽ cho Laravel phát đi sự kiện khi nó được kích hoạt:

```
<?php
namespace App\Events;
use App\Order;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Queue\SerializesModels;


class OrderShipmentStatusUpdated implements ShouldBroadcast
{
  /**
   * The order instance.
   *
   * @var \App\Order
```

```
     */
    public $order;
}
```

Thông thường `ShouldBroadcast` interface yêu cầu event của chúng ta phải khai báo phương thức broadcastOn. Phương thức này có trách nhiệm trả về các kênh mà event sẽ phát trên đó. Phần sơ khai trống của phương thức này đã được khai báo sẵn trên các class event được tạo, vì vậy chúng ta chỉ cần điền thông tin chi tiết của nó. Chúng ta chỉ muốn người tạo đơn đặt hàng có thể xem cập nhật trạng thái, vì vậy, chúng ta sẽ phát event trên một channel riêng được liên kết với đơn đặt hàng:

```
/**
 * Get the channels the event should broadcast on.
 *
 * @return \Illuminate\Broadcasting\PrivateChannel
 */
public function broadcastOn()
{
    return new PrivateChannel('orders.'.$this->order->id);
}
```

Cấp phép kênh

Hãy nhớ rằng người dùng phải được phép theo dõi trên các channel riêng tư. Chúng ta có thể xác định các quy tắc xác thực channel của mình trong hồ sơ đăng ký của chúng ta *routes/channels.php*. Trong ví dụ này, chúng ta cần xác minh rằng bất kỳ người dùng nào cố gắng theo dõi channel riêng tư `orders.1` phải thực sự là người tạo ra lệnh:

```
use App\Models\Order;

Broadcast::channel('orders.{orderId}', function ($user, $orderId) {
    return $user->id === Order::findOrNew($orderId)->user_id;
});
```

Phương thức `channel` chấp nhận hai đối số: tên của channel và một lệnh callback trả về `true` hoặc `false`, sẽ cho biết liệu người dùng có được phép theo dõi kênh này hay không.

Tất cả các callback sẽ xác nhận người dùng hiện tại đã được xác thực làm đối số đầu tiên của nó và bất kỳ tham số ký tự đại diện bổ sung nào làm đối số tiếp theo của nó. Trong ví dụ này, chúng ta đang sử dụng `{orderId}` để chỉ ra rằng phần "ID" của tên channel là ký tự đại diện.

Theo dõi boardcast event

Tiếp theo, tất cả những gì còn lại là theo dõi các sự kiện trong ứng dụng JavaScript của chúng ta. Chúng ta có thể làm điều này bằng cách sử dụng Laravel Echo. Đầu tiên, chúng ta sẽ sử dụng phương thức `private` để đăng ký channel riêng tư. Sau đó, chúng tôi có thể sử dụng phương thức `listen` để theo dõi event `OrderShipmentStatusUpdated`. Mặc định, tất cả các thuộc tính công khai của event sẽ được đưa vào boardcast event:

```
Echo.private(`orders.${orderId}`)
  .listen('OrderShipmentStatusUpdated', (e) => {
    console.log(e.order);
  });
```

Khai báo boardcast event

Để thông báo cho Laravel rằng một event nào đó sẽ được phát sóng, bạn phải thực thi interface `Illuminate\Contracts\Broadcasting\ShouldBroadcast` trên class event đó. Interface này đã được nhập vào tất cả các class event do framework tạo ra, vì vậy bạn có thể dễ dàng thêm nó vào bất kỳ event nào của mình.

Interface yêu cầu bạn khai báo một phương thức duy nhất là `ShouldBroadcast::broadcastOn`. Phương thức `broadcastOn` sẽ trả về một channel hoặc một mảng channel mà event sẽ phát đi trên đó. Các channel phải là đối tượng của `Channel`, `PrivateChannel` hoặc `PresenceChannel`. Các trường hợp `Channel` sẽ đại diện cho các channel công khai mà bất kỳ người dùng nào cũng có thể đăng ký, trong khi `PrivateChannels` và `PresenceChannels` sẽ đại diện cho các channel riêng tư yêu cầu xác thực:

```
<?php
namespace App\Events;
use App\Models\User;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
```

```php
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Queue\SerializesModels;

class ServerCreated implements ShouldBroadcast
{
  use SerializesModels;

  /**
   * The user that created the server.
   *
   * @var \App\Models\User
   */
  public $user;

  /**
   * Create a new event instance.
   *
   * @param  \App\Models\User  $user
   * @return void
   */
  public function __construct(User $user)
  {
    $this->user = $user;
  }

  /**
   * Get the channels the event should broadcast on.
   *
   * @return Channel|array
   */
  public function broadcastOn()
  {
    return new PrivateChannel('user.'.$this->user->id);
  }
}
```

Sau khi thực thi interface `ShouldBroadcast`, bạn chỉ cần kích hoạt event như bình thường. Khi event đã được kích hoạt, một job sẽ được xếp hàng đợi và sẽ được tự động phát đi event bằng cách sử dụng driver boardcast do bạn chọn.

Tên của boardcast

Mặc định, Laravel sẽ phát đi các event bằng cách sử dụng tên class của event. Tuy nhiên, bạn có thể tự chọn tên boardcast bằng cách khai báo phương thức `broadcastAs` trên event:

```
/**
 * The event's broadcast name.
 *
 * @return string
 */
public function broadcastAs()
{
  return 'server.created';
}
```

Nếu bạn tự tạo tên cho boardcast bằng phương thức `broadcastAs`, thì bạn nên đảm bảo việc đăng ký listener của mình bằng ký tự (`.`) đứng đầu. Điều này sẽ cho Echo biết không cần thêm namespace của ứng dụng vào event:

```
.listen('.server.created', function (e) {
  ....
});
```

Dữ liệu truyền phát

Khi một event được phát đi, tất cả các thuộc tính public của event đó sẽ tự động được tuần tự hóa và phát đi dưới dạng payload của event, cho phép bạn truy cập bất kỳ dữ liệu công khai nào của event đó từ ứng dụng JavaScript của bạn. Vì vậy, ví dụ: nếu event của bạn có một thuộc tính public `$user` chứa Eloquent model, thì payload phát đi của event sẽ là:

```
{
```

```
  "user": {
    "id": 1,
    "name": "Patrick Stewart"
    ...
  }
}
```

Tuy nhiên, nếu bạn muốn có nhiều quyền kiểm soát chi tiết hơn đối với payload phát đi của mình, bạn có thể thêm một phương thức `broadcastWith` vào event của mình. Phương thức này sẽ trả về mảng dữ liệu mà bạn muốn phát dưới dạng payload của event:

```
/**
 * Get the data to broadcast.
 *
 * @return array
 */
public function broadcastWith()
{
  return ['id' => $this->user->id];
}
```

Hàng đợi phát sóng

Mặc định, mỗi boardcast event được chỉ định trong tập tin cấu hình xếp hàng đợi của bạn *queue.php*. Bạn có thể điều chỉnh kết nối hàng đợi và tên được boardcast sử dụng bằng cách chỉ định thuộc tính `connection` và `queue` trên class event của bạn:

```
/**
 * The name of the queue connection to use when broadcasting the event.
 *
 * @var string
 */
public $connection = 'redis';


/**
 * The name of the queue on which to place the broadcasting job.
```

```
     *
     * @var string
     */
    public $queue = 'default';
```

Ngoài ra, bạn có thể điều chỉnh tên hàng đợi bằng cách chỉ định phương thức `broadcastQueue` cho event của mình:

```
/**
 * The name of the queue on which to place the broadcasting job.
 *
 * @return string
 */
public function broadcastQueue()
{
  return 'default';
}
```

Nếu bạn muốn phát đi event của mình bằng việc sử dụng hàng đợi `sync` thay cho driver hàng đợi mặc định, bạn có thể thực thi interface `ShouldBroadcastNow` thay cho interface `ShouldBroadcast`:

```
<?php

use Illuminate\Contracts\Broadcasting\ShouldBroadcastNow;

class OrderShipmentStatusUpdated implements ShouldBroadcastNow
{
  //
}
```

Điều kiện phát sự kiện

Đôi khi bạn chỉ muốn phát đi event của mình nếu một điều kiện đã cho là thỏa mãn `true`. Bạn có thể khai báo các điều kiện này bằng cách thêm phương thức `broadcastWhen` vào class event của mình:

```php
/**
 * Determine if this event should broadcast.
 *
 * @return bool
 */
public function broadcastWhen()
{
  return $this->order->value > 100;
}
```

Boardcast & DB transaction

Khi các event được gửi đi trong các DB transaction, chúng có thể được hệ thống hàng đợi xử lý trước khi DB transaction được commit. Khi điều này xảy ra, bất kỳ cập nhật nào bạn đã thực hiện cho các model hoặc record cơ sở dữ liệu trong quá trình transaction với cơ sở dữ liệu có thể sẽ chưa có hiệu lực trong cơ sở dữ liệu. Ngoài ra, bất kỳ model hoặc record cơ sở dữ liệu nào được tạo trong transaction có thể không tồn tại trong cơ sở dữ liệu. Nếu event của bạn phụ thuộc vào các model này, các lỗi không mong muốn có thể xảy ra khi chức năng phát event được xử lý.

Nếu tùy chọn cấu hình `after_commit` của kết nối hàng đợi của bạn được đặt thành `false`, thì bạn vẫn có thể chỉ ra rằng một event được phát đi cụ thể sẽ được gửi đi sau khi tất cả các transaction cơ sở dữ liệu đã bắt đầu được commit bằng cách chỉ định một thuộc tính `$afterCommit` trên class event:

```php
<?php

namespace App\Events;

use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Queue\SerializesModels;

class ServerCreated implements ShouldBroadcast
{
  use SerializesModels;

  public $afterCommit = true;
```

```
}
```

> Để tìm hiểu thêm về cách giải quyết những vấn đề này, vui lòng xem lại tài liệu liên quan đến các job được xếp hàng đợi và giao dịch với transaction trong cơ sở dữ liệu.

Channel và Xác thực

Các channel riêng tư yêu cầu bạn xác nhận người dùng hiện tại đã được xác thực thực sự có thể theo dõi các channel này. Điều này được thực hiện bằng cách thực hiện một HTTP request đến ứng dụng Laravel của bạn với tên channel và cho phép ứng dụng của bạn xác định xem người dùng có thể theo dõi channel đó hay không. Khi sử dụng Laravel Echo, việc HTTP request xác nhận đăng ký cho các channel riêng tư sẽ được thực hiện tự động; tuy nhiên, bạn cần chỉ định các route thích hợp để đáp ứng các yêu cầu này.

Khai báo các route xác nhận

Rất may, Laravel giúp dễ dàng xác định các route để phản hồi các request cấp quyền channel. Trong ứng dụng Laravel có đi kèm `App\Providers\BroadcastServiceProvider`, trong đó bạn sẽ thấy một câu lệnh gọi đến phương thức `Broadcast::routes`. Phương thức này sẽ đăng ký route `/broadcasting/auth` để xử lý các yêu cầu xác thực:

```
Broadcast::routes();
```

Phương thức `Broadcast::routes` sẽ tự động đặt các route của nó trong nhóm middleware `web`; tuy nhiên, bạn có thể truyền một mảng các thuộc tính route cho phương thức nếu bạn muốn điều chỉnh các thuộc tính được chỉ định:

```
Broadcast::routes($attributes);
```

Tự chọn endpoint cho route xác thực

Mặc định, Echo sẽ sử dụng endpoint `/broadcasting/auth` để cấp quyền truy cập channel. Tuy nhiên, bạn có thể chỉ định điểm cuối xác thực của riêng mình bằng cách truyền tùy chọn cấu hình `authEndpoint` cho đối tượng Echo của bạn:

```
window.Echo = new Echo({
  broadcaster: 'pusher',
  // ...
  authEndpoint: '/custom/endpoint/auth'
});
```

Điều chỉnh yêu cầu xác thực

Bạn có thể điều chỉnh cách Laravel Echo thực hiện các yêu cầu xác thực bằng cách cung cấp chương trình xác thực tự tạo khi khởi tạo Echo:

```
window.Echo = new Echo({
  // ...
  authorizer: (channel, options) => {
    return {
      authorize: (socketId, callback) => {
        axios.post('/api/broadcasting/auth', {
          socket_id: socketId,
          channel_name: channel.name
        })
        .then(response => {
          callback(false, response.data);
        })
        .catch(error => {
          callback(true, error);
        });
      }
    };
  },
})
```

Xác định callback xác thực

Tiếp theo, chúng ta cần xác định logic sẽ thực sự xác định xem người dùng hiện tại đang được xác thực có thể theo dõi một channel nhất định hay không. Điều này được thực hiện trong tập tin *routes/channels.php* đi kèm với ứng dụng của bạn. Trong tập tin này, bạn có thể

sử dụng phương thức `Broadcast::channel` để đăng ký callback xác thực channel:

```
Broadcast::channel('orders.{orderId}', function ($user, $orderId) {
  return $user->id === Order::findOrNew($orderId)->user_id;
});
```

Phương thức `channel` chấp nhận hai đối số: tên của channel và callback trả về `true` hoặc `false` cho biết liệu người dùng có được phép theo dõi channel hay không.

Tất cả các callback xác thực sẽ xác nhận người dùng hiện tại đã được xác thực làm đối số đầu tiên của họ và bất kỳ tham số ký tự đại diện bổ sung nào làm đối số tiếp theo của họ. Trong ví dụ này, chúng tôi đang sử dụng `{orderId}` để chỉ ra rằng phần "ID" của tên channel là ký tự đại diện.

Ràng buộc model callback xác thực

Cũng giống như các HTTP route, các channel route cũng có thể tận dụng lợi thế của ràng buộc route model công khai và minh bạch. Ví dụ: thay vì nhận một chuỗi hoặc ID thứ tự số, bạn có thể yêu cầu một đối tượng `Order` model:

```
use App\Models\Order;

Broadcast::channel('orders.{order}', function ($user, Order $order) {
  return $user->id === $order->user_id;
});
```

> **Chú ý:** Không giống như ràng buộc route model HTTP, ràng buộc channel model không hỗ trợ xác định phạm vi ràng buộc model ngầm định tự động. Tuy nhiên, điều này hiếm khi là một vấn đề vì hầu hết các channel có thể được xác định phạm vi dựa trên primary-key, duy nhất của một model.

Xác nhận callback xác thực

Các channel riêng tư và xác nhận người dùng xác thực hiện tại thông qua chương trình bảo vệ xác thực mặc định của ứng dụng của bạn. Nếu người dùng không được xác thực, việc xác nhận xác thực trên channel sẽ tự động bị từ chối và callback xác thực sẽ không bao giờ được thực thi. Tuy nhiên, bạn có thể chỉ định nhiều lớp bảo vệ tùy chỉnh, nó sẽ xác thực

request gửi đến nếu cần:

```
Broadcast::channel('channel', function () {
  // ...
}, ['guards' => ['web', 'admin']]);
```

Tạo các channel class

Nếu ứng dụng của bạn sử dụng nhiều channel khác nhau, thì tập tin *routes/channels.php* của bạn có thể trở nên cồng kềnh. Vì vậy, thay vì sử dụng các hàm nặc danh để cấp quyền cho các channel, bạn có thể sử dụng các channel class. Để tạo một channel class, hãy sử dụng lệnh Artisan `make:channel`. Lệnh này sẽ đặt một channel class mới trong thư mục *App/Broadcasting*.

```
php artisan make:channel OrderChannel
```

Tiếp theo, đăng ký channel của bạn trong tập tin *routes/channels.php* của bạn:

```
use App\Broadcasting\OrderChannel;

Broadcast::channel('orders.{order}', OrderChannel::class);
```

Cuối cùng, bạn có thể đặt logic xác thực cho channel của mình trong phương thức `join` của channel class. Phương thức `join` sẽ chứa cùng một logic mà bạn thường đặt khi đóng xác thực channel của mình. Bạn cũng có thể tận dụng lợi thế của ràng buộc channel model:

```
<?php
namespace App\Broadcasting;
use App\Models\Order;
use App\Models\User;

class OrderChannel
{
  /**
   * Create a new channel instance.
```

```
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Authenticate the user's access to the channel.
     *
     * @param  \App\Models\User  $user
     * @param  \App\Models\Order  $order
     * @return array|bool
     */
    public function join(User $user, Order $order)
    {
        return $user->id === $order->user_id;
    }
}
```

Giống như nhiều class khác trong Laravel, các channel class sẽ tự động được trả lại từ service container. Vì vậy, bạn có thể khai kiểu với bất kỳ class nào mà channel của bạn yêu cầu trong constructor của nó.

Boardcast event

Khi bạn đã xác định một event và đánh dấu event đó bằng interface `ShouldBroadcast`, bạn chỉ cần kích hoạt event bằng cách sử dụng phương thức điều phối của event. Chương trình điều phối event sẽ nhận thấy rằng event được đánh dấu bằng interface `ShouldBroadcast` và sẽ đặt lịch event để phát đi:

```
use App\Events\OrderShipmentStatusUpdated;

OrderShipmentStatusUpdated::dispatch($order);
```

Chỉ cho những người khác

Khi xây dựng một ứng dụng sử dụng tính năng boardcast, đôi khi bạn có thể cần phải phát đi các event cho tất cả người dùng đăng ký channel nào đó ngoại trừ người dùng hiện tại. Bạn có thể thực hiện điều này bằng cách sử dụng hàm trợ giúp `broadcast` và phương thức `toOthers`:

```
use App\Events\OrderShipmentStatusUpdated;

broadcast(new OrderShipmentStatusUpdated($update))->toOthers();
```

Để hiểu rõ hơn khi nào bạn có thể muốn sử dụng phương thức `toOthers` này, thì bạn hãy tưởng tượng một ứng dụng danh sách task trong đó người dùng có thể tạo một task mới bằng cách nhập tên task. Để tạo một task, ứng dụng của bạn có thể đưa ra một yêu cầu tới `/task` URL truyền tải quá trình tạo của task và trả về một JSON của task mới. Khi ứng dụng JavaScript của bạn nhận được phản hồi từ điểm cuối, nó có thể chèn trực tiếp task mới vào danh sách task của nó như sau:

```
axios.post('/task', task)
  .then((response) => {
    this.tasks.push(response.data);
  });
```

Tuy nhiên, hãy nhớ rằng chúng ta cũng phát đi việc tạo task. Nếu ứng dụng JavaScript của bạn cũng đang theo dõi event này để thêm task vào danh sách task, thì bạn sẽ có các task trùng lặp trong danh sách của mình: một từ điểm cuối và một từ chương trình boardcast. Bạn có thể giải quyết vấn đề này bằng cách sử dụng phương thức `toOthers` để chương trình boardcast biết không phát đi event cho người dùng hiện tại.

> **Chú ý:** Sự kiện của bạn phải sử dụng `Illuminate\Broadcasting\InteractsWithSockets` để gọi phương thức `toOthers`.

Cấu hình

Khi bạn khởi tạo đối tượng Laravel Echo, một ID sẽ được gán cho kết nối. Nếu bạn đang sử dụng một đối tượng Axios để thực hiện các HTTP request từ ứng dụng JavaScript của mình, ID socket sẽ tự động được đính kèm với mọi request gửi đi dưới header `X-Socket-ID`. Sau đó, khi bạn gọi phương thức `toOthers`, Laravel sẽ trích xuất ID từ header và cho

boardcast biết không phát tới bất kỳ kết nối nào với ID đó.

Nếu bạn không sử dụng đối tượng Axios, thì bạn sẽ cần phải chỉ định cấu hình ứng dụng JavaScript của mình theo cách thủ công để gửi header `X-Socket-ID` với tất cả các request gửi đi. Bạn có thể truy xuất ID bằng phương thức `Echo.socketId`:

```
var socketId = Echo.socketId();
```

Điều chỉnh kết nối

Nếu ứng dụng của bạn tương tác với nhiều kết nối boardcast và bạn muốn phát đi event bằng một boardcast khác nào đó với chương trình mặc định của mình, bạn có thể chỉ định kết nối để đẩy một event đến bằng phương thức `via`:

```
use App\Events\OrderShipmentStatusUpdated;

broadcast(new OrderShipmentStatusUpdated($update))->via('pusher');
```

Ngoài ra, bạn có thể chỉ định kết nối boardcast của event bằng cách gọi phương thức `broadcastVia` bên trong constructor của Event. Tuy nhiên, trước khi làm như vậy, bạn nên đảm bảo rằng event class sử dụng `InteractsWithBroadcasting`:

```
<?php
namespace App\Events;
use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithBroadcasting;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Queue\SerializesModels;

class OrderShipmentStatusUpdated implements ShouldBroadcast
{
  use InteractsWithBroadcasting;
```

```
    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->broadcastVia('pusher');
    }
}
```

Nhận các boardcast

Theo dõi event

Khi bạn đã cài đặt và khởi tạo Laravel Echo, thì bạn đã sẵn sàng bắt đầu theo dõi các event được phát đi từ ứng dụng Laravel của mình. Đầu tiên, sử dụng phương thức `channel` để truy xuất một đối tượng của channel, sau đó gọi phương thức `listen` để theo dõi một sự kiện được chỉ định:

```
Echo.channel(`orders.${this.order.id}`)
    .listen('OrderShipmentStatusUpdated', (e) => {
        console.log(e.order.name);
    });
```

Nếu bạn muốn theo dõi các event trên channel riêng tư, thì hãy sử dụng phương thức `private` để thay thế. Bạn có thể tiếp tục gọi thêm các lệnh gọi đến phương thức `listen` để theo dõi nhiều event trên một channel:

```
Echo.private(`orders.${this.order.id}`)
    .listen(...)
    .listen(...)
    .listen(...);
```

Ngừng theo dõi event

Nếu bạn muốn ngừng theo dõi một event nào đó mà không rời khỏi channel, thì bạn có thể sử dụng phương thức `stopListening`:

```
Echo.private(`orders.${this.order.id}`)
    .stopListening('OrderShipmentStatusUpdated')
```

Rời khỏi channel

Để rời khỏi một channel, bạn có thể gọi phương thức `leaveChannel` trên đối tượng Echo của mình:

```
Echo.leaveChannel(`orders.${this.order.id}`);
```

Nếu bạn muốn rời khỏi một channel cũng như các channel riêng tư và channel hiện diện liên quan, bạn có thể gọi phương thức `leave`:

```
Echo.leave(`orders.${this.order.id}`);
```

Namespace

Bạn có thể nhận thấy trong các ví dụ trên một điều rằng chúng ta không chỉ định namespace `App\Events` cho các class event. Điều này là do Echo sẽ tự động giả định các event được đặt trong namespace `App\Events`. Tuy nhiên, bạn có thể cấu hình namespace gốc khi khởi tạo Echo bằng cách truyền giá trị cho tùy chọn cấu hình `namespace`:

```
window.Echo = new Echo({
    broadcaster: 'pusher',
    // ...
    namespace: 'App.Other.Namespace'
});
```

Ngoài ra, bạn có thể đặt trước các class event bằng một ký tự "dấu chấm" `.` khi đăng ký chúng bằng Echo. Điều này sẽ cho phép bạn luôn chỉ định đầy đủ tên của class:

```
Echo.channel('orders')
```

```
    .listen('.Namespace\\Event\\Class', (e) => {
      //
    });
```

Channel hiện diện

Các channel hiện diện được xây dựng dựa trên tính bảo mật của các channel riêng tư đồng thời thể hiện thêm tính năng nhận biết ai đã đăng ký channel. Điều này giúp bạn dễ dàng xây dựng các tính năng ứng dụng cộng tác, mạnh mẽ chẳng hạn như thông báo cho người dùng khi người dùng khác đang xem cùng một trang hoặc liệt kê các thành viên trong phòng trò chuyện.

Cho phép các channel hiện diện

Tất cả các channel hiện diện cũng là kênh riêng tư; do đó, người dùng phải được cấp phép để truy cập chúng. Tuy nhiên, khi xác định lệnh callback xác thực cho các channel hiện diện, bạn sẽ không trả lại `true` nếu người dùng được xác thực tham gia channel. Thay vào đó, bạn nên trả về một mảng dữ liệu cho người dùng.

Dữ liệu được trả về bởi lệnh callback xác thực sẽ được cung cấp cho listener của kênh hiện diện trong ứng dụng JavaScript của bạn. Nếu người dùng không được phép tham gia channel hiện diện, bạn nên trả lại `false` hoặc `null`:

```
Broadcast::channel('chat.{roomId}', function ($user, $roomId) {
    if ($user->canJoinRoom($roomId)) {
        return ['id' => $user->id, 'name' => $user->name];
    }
});
```

Tham gia các channel hiện diện

Để tham gia channel hiện diện, bạn có thể sử dụng phương `join` của Echo. Phương thức `join` sẽ trả về một thực thi `PresenceChannel`, cùng với việc hiển thị phương thức `listen`, cho phép bạn đăng ký các event `here` và `joining`, `leaving`.

```
Echo.join(`chat.${roomId}`)
```

```
    .here((users) => {
      //
    })
    .joining((user) => {
      console.log(user.name);
    })
    .leaving((user) => {
      console.log(user.name);
    })
    .error((error) => {
      console.error(error);
    });
```

Lệnh `here` sẽ được thực hiện ngay lập tức khi channel được nối thành công và sẽ nhận được một mảng chứa thông tin người dùng cho tất cả những người dùng khác hiện đã đăng ký kênh. Phương thức `joining` sẽ được thực thi khi người dùng mới tham gia một channel, trong khi phương thức `leaving` sẽ được thực thi khi người dùng rời khỏi channel. Phương thức `error` sẽ được thực thi khi điểm cuối xác thực trả về mã trạng thái HTTP khác 200 hoặc nếu có sự cố khi phân tích dữ liệu JSON được trả về.

Phát đến các channel hiện diện

Các channel hiện diện có thể nhận các event giống như các channel công khai hoặc riêng tư. Sử dụng ví dụ về phòng trò chuyện, chúng ta có thể muốn phát các event `NewMessage` tới channel hiện tại của phòng. Để làm như vậy, chúng ta sẽ trả về một đối tượng của `PresenceChannel` từ phương thức `broadcastOn` của sự kiện:

```
/**
 * Get the channels the event should broadcast on.
 *
 * @return Channel|array
 */
public function broadcastOn()
{
    return new PresenceChannel('room.'.$this->message->room_id);
}
```

Cũng như các event khác, bạn có thể sử dụng hàm trợ giúp `broadcast` và phương thức `toOthers` để loại trừ người dùng hiện tại nhận boardcast:

```
broadcast(new NewMessage($message));

broadcast(new NewMessage($message))->toOthers();
```

Như điển hình của các loại event khác, bạn có thể lắng nghe các event được gửi đến các channel hiện diện bằng phương thức `listen` của Echo:

```
Echo.join(`chat.${roomId}`)
  .here(...)
  .joining(...)
  .leaving(...)
  .listen('NewMessage', (e) => {
    //
  });
```

Boardcast model

> **Chú ý:** Trước khi đọc tài liệu sau đây về boardcast model, chúng tôi khuyên bạn nên làm quen với các khái niệm chung về các boardcast model service của Laravel cũng như cách tạo và theo dõi các event phát đi theo cách thủ công.

Thông thường các event phát đi khi các mô hình Eloquent model của ứng dụng của bạn được tạo, cập nhật hoặc xóa. Tất nhiên, điều này có thể dễ dàng được thực hiện bằng cách xác định thủ công các event tự tạo cho các trạng thái thay đổi của mô hình Eloquent model và đánh dấu các event đó bằng interface `ShouldBroadcast`.

Tuy nhiên, nếu bạn không sử dụng các event này cho bất kỳ mục đích nào khác trong ứng dụng của mình, thì việc tạo các event class với mục đích duy nhất là phát chúng có thể rất phức tạp. Để khắc phục điều này, Laravel cho phép bạn chỉ ra rằng một mô hình Eloquent model sẽ được tự động phát đi các trạng thái thay đổi của nó.

Để bắt đầu, mô hình Eloquent model của bạn nên sử dụng `Illuminate\Database\Eloquent\BroadcastsEvents`. Ngoài ra, model phải khai báo một phương thức `broadcastsOn`, phương thức này sẽ trả về một mảng các channel mà các event của model sẽ phát trên đó:

```php
<?php
namespace App\Models;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Database\Eloquent\BroadcastsEvents;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
  use BroadcastsEvents, HasFactory;

  /**
   * Get the user that the post belongs to.
   */
  public function user()
  {
    return $this->belongsTo(User::class);
  }

  /**
   * Get the channels that model events should broadcast on.
   *
   * @param  string  $event
   * @return \Illuminate\Broadcasting\Channel|array
   */
  public function broadcastOn($event)
  {
    return [$this, $this->user];
  }
}
```

Sau khi model của bạn sử dụng các trait này và xác định các channel phát tán, model sẽ bắt đầu tự động phát tán các event khi một đối tượng model được tạo, cập nhật, xóa, bỏ vào thùng rác hoặc khôi phục.

Ngoài ra, bạn có thể nhận thấy rằng phương thức `broadcastOn` sẽ nhận một đối số chuỗi `$event`. Đối số này chứa loại event đã xảy ra trên model và sẽ có giá trị là `created`,

`update`, `deleted`, `trashed`, hoặc `restored`. Bằng cách kiểm tra giá trị của biến này, bạn có thể xác định channel nào (nếu có) mà model sẽ phát tán một event cụ thể:

```
/**
 * Get the channels that model events should broadcast on.
 *
 * @param  string  $event
 * @return \Illuminate\Broadcasting\Channel|array
 */
public function broadcastOn($event)
{
  return match ($event) {
    'deleted' => [],
    default => [$this, $this->user],
  };
}
```

Tự tạo boardcast event model

Đôi khi, bạn có thể muốn điều chỉnh cách Laravel tạo model cơ bản event phát tán. Bạn có thể thực hiện điều này bằng cách xác định một phương thức `newBroadcastableEvent` trên mô hình Eloquent model của bạn. Phương thức này sẽ trả về một đối tượng `Illuminate\Database\Eloquent\BroadcastableModelEventOccurred`:

```
use Illuminate\Database\Eloquent\BroadcastableModelEventOccurred

/**
 * Create a new broadcastable model event for the model.
 *
 * @param  string  $event
 * @return \Illuminate\Database\Eloquent\BroadcastableModelEventOccurred
 */
protected function newBroadcastableEvent($event)
{
  return (new BroadcastableModelEventOccurred(
    $this, $event
  ))->dontBroadcastToCurrentUser();
```

```
}
```

Quy ước về boardcast model

Quy ước channel

Như bạn có thể nhận thấy, phương thức `broadcastOn` trong ví dụ model ở trên không trả về các đối tượng `Channel`. Thay vào đó, các mô hình Eloquent model được trả lại trực tiếp. Nếu một đối tượng mô hình Eloquent model được phương thức `broadcastOn` của model của bạn trả về (hoặc được chứa trong một mảng được phương thức trả về), Laravel sẽ tự động khởi tạo một đối tượng channel riêng cho model bằng cách sử dụng tên class và mã định danh *primary-key* của model làm tên channel.

Vì vậy, một model `App\Models\User` với một id là 1 sẽ được chuyển đổi thành một đối tượng `Illuminate\Broadcasting\PrivateChannel` có tên là `App.Models.User.1`. Tất nhiên, ngoài việc trả về các đối tượng mô hình Eloquent model từ phương thức `broadcastOn` của model, bạn có thể trả lại các đối tượng hoàn chỉnh của `Channel` để có toàn quyền kiểm soát tên channel của model:

```
use Illuminate\Broadcasting\PrivateChannel;

/**
 * Get the channels that model events should broadcast on.
 *
 * @param  string  $event
 * @return \Illuminate\Broadcasting\Channel|array
 */
public function broadcastOn($event)
{
  return [new PrivateChannel('user.'.$this->id)];
}
```

Nếu bạn định trả về một cách công khai một đối tượng channel từ phương thức broadcastOn của model, thì bạn có thể truyền một đối tượng mô hình Eloquent model cho constructor của channel. Khi làm như vậy, Laravel sẽ sử dụng các quy ước channel model được thảo luận ở trên để chuyển đổi mô hình Eloquent model thành một chuỗi tên channel:

```
return [new Channel($this->user)];
```

Nếu bạn cần xác định tên của channel của một model nào đó, thì bạn có thể gọi phương thức `broadcastChannel` trên bất kỳ đối tượng model nào. Ví dụ: phương thức này sẽ trả về chuỗi `App.Models.User.1` cho một model `id` của `App\Models\User` với 1:

```
$user->broadcastChannel()
```

Quy ước sự kiện

Vì các mô hình boardcast event model sẽ không được liên kết với một event "thực tế" trong thư mục ứng dụng của bạn `App\Events`, chúng được gán tên và payload dựa trên các quy ước. Quy ước của Laravel là phát event bằng cách sử dụng tên class của mô hình model (không bao gồm namespace) và tên của mô hình event model đã kích hoạt phát tán.

Vì vậy, ví dụ: một bản cập nhật cho mô hình `App\Models\Post` model sẽ truyền phát một event đến ứng dụng phía máy khách của bạn như `PostUpdated` với payload sau:

```
{
  "model": {
    "id": 1,
    "title": "My first post"
    ...
  },
  ...
  "socket": "someSocketId",
}
```

Việc xóa trong mô hình `App\Models\User` model sẽ phát tán đi một event có tên `UserDeleted`.

Nếu muốn, bạn có thể xác định tên boardcast tùy chỉnh và payload bằng cách thêm một phương thức `broadcastAs` và `broadcastWith` vào mô hình của mình. Các phương thức này nhận tên của event/activity đang xảy ra trong model, cho phép bạn tùy chỉnh tên event và payload cho mỗi hoạt động trong mô hình. Nếu `null` được trả về từ phương thức `broadcastAs`, Laravel sẽ sử dụng các quy ước về tên event phát tán trong model được thảo luận ở trên khi phát tán event:

```php
/**
 * The model event's broadcast name.
 *
 * @param  string  $event
 * @return string|null
 */
public function broadcastAs($event)
{
  return match ($event) {
    'created' => 'post.created',
    default => null,
  };
}

/**
 * Get the data to broadcast for the model.
 *
 * @param  string  $event
 * @return array
 */
public function broadcastWith($event)
{
  return match ($event) {
    'created' => ['title' => $this->title],
    default => ['model' => $this],
  };
}
```

Theo dõi các chương trình boardcast model

Khi bạn đã thêm trait `BroadcastsEvents` vào model của mình và khai báo phương thức broadcastOn lên model, là bạn đã sẵn sàng bắt đầu theo dõi các event model được phát đi trong ứng dụng phía máy khách của mình. Trước khi bắt đầu, bạn có thể tham khảo tài liệu đầy đủ về theo dõi các event.

Đầu tiên, sử dụng phương thức `private` để lấy một đối tượng của một channel, sau đó gọi phương thức `listen` để theo dõi một event được chỉ định. Thông thường, tên channel

được cung cấp cho phương thức `private` phải tương ứng với các quy ước boardcast model của Laravel.

Khi bạn đã có được một đối tượng channel, bạn có thể sử dụng phương thức `listen` này để theo dõi một event cụ thể. Vì các event phát tán model không được liên kết với một event "thực tế" trong thư mục ứng dụng của bạn `App\Events`, nên tên event phải được bắt đầu bằng một "dấu chấm" `.` để cho biết nó không thuộc một namespace cụ thể nào. Mỗi event trong boardcast model có một thuộc tính `model` chứa tất cả các thuộc tính có thể phát tán của mô hình:

```
Echo.private(`App.Models.User.${this.user.id}`)
  .listen('.PostUpdated', (e) => {
    console.log(e.model);
  });
```

Các event trên Client

> Khi sử dụng ***Pusher Channels***, bạn phải bật tùy chọn "Client Events" trong phần "App Settings" trên trang điều hành ứng dụng của bạn để gửi các client event.

Đôi khi bạn có thể muốn truyền phát một event cho các máy khách được kết nối khác mà không cần nhấn vào ứng dụng Laravel của bạn. Điều này có thể đặc biệt hữu ích cho những thứ như thông báo "đang nhập", nơi bạn muốn thông báo cho người dùng ứng dụng của mình rằng người dùng khác đang nhập tin nhắn trên một màn hình nhất định.

Để phát đi các event của client kiểu này, thì bạn có thể sử dụng phương thức `whisper` của Echo:

```
Echo.private(`chat.${roomId}`)
  .whisper('typing', {
    name: this.user.name
  });
```

Để theo dõi các event của client, bạn có thể sử dụng phương thức `listenForWhisper`:

```
Echo.private(`chat.${roomId}`)
  .listenForWhisper('typing', (e) => {
```

```
    console.log(e.name);
  });
```

Message - Thông báo

Bằng cách ghép nối truyền phát event với message, ứng dụng JavaScript của bạn có thể nhận được message mới khi chúng xuất hiện mà không cần phải làm mới trang. Trước khi bắt đầu, hãy nhớ đọc qua tài liệu về cách sử dụng message channel.

Khi bạn đã cấu hình message để sử dụng channel, bạn có thể theo dõi các event được phát tán đi bằng phương thức `notification` của Echo. Hãy nhớ rằng tên channel phải khớp với tên class của đối tượng nhận thông báo:

```
Echo.private(`App.Models.User.${userId}`)
  .notification((notification) => {
    console.log(notification.type);
  });
```

Trong ví dụ này, tất cả các message được gửi đến các đối tượng `App\Models\User` qua channel `broadcast` sẽ được nhận lại bởi callback. Một callback xác thực channel cho channel `App.Models.User.{id}` sẽ được đưa vào trong `BroadcastServiceProvider` mặc định đi kèm với framework Laravel.