GIỚI THIỆU TỔNG QUAN

1. GIỚI THIỆU

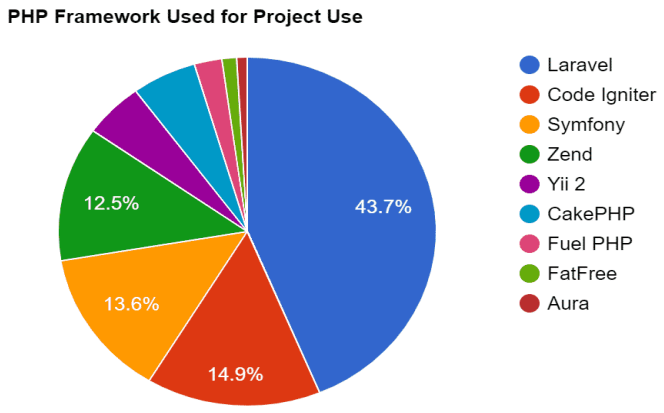
1.1. Framework là gì?

Framework chính là một thư viện với các tài nguyên có sẵn cho từng lĩnh vực để lập trình viên sử dụng thay vì phải tự thiết kế. Có Framework, lập trình viên chỉ tìm hiểu và khai thác những tài nguyên đó, gắn kết chúng lại với nhau và hoàn chỉnh sản phẩm. Đối với lập trình viên trong mỗi một lĩnh vực, họ cần phải xây dựng các lớp chương trình để xây dựng nên những phần mềm, ứng dụng thành phẩm.

1.2. PHP Framework là gì?

PHP là một trong những ngôn ngữ lập trình web được sử dụng rộng rãi nhất hiện nay. Với sự phát triển của PHP, nhiều PHP Framework đã xuất hiện nhưng chỉ một ít trong số chúng thực sự sử dụng toàn bộ tiềm năng của ngôn ngữ PHP. Mỗi một framework của PHP đều có khai triển, tính năng và các khả năng riêng biệt.

PHP Framework là thư viện làm cho sự phát triển của những ứng dụng Web viết bằng ngôn ngữ PHP trở nên trôi chảy hơn. Bằng cách cung cấp 1 cấu trúc cơ bản để xây dựng những ứng dụng. Hay nói cách khác, PHP Framework giúp bạn thúc đẩy nhanh chóng quá trình phát triển ứng dụng. Giúp bạn tiết kiệm được thời gian, tăng sự ổn định cho ứng dụng. Giảm thiểu số lần phải viết lại code cho lập trình viên.

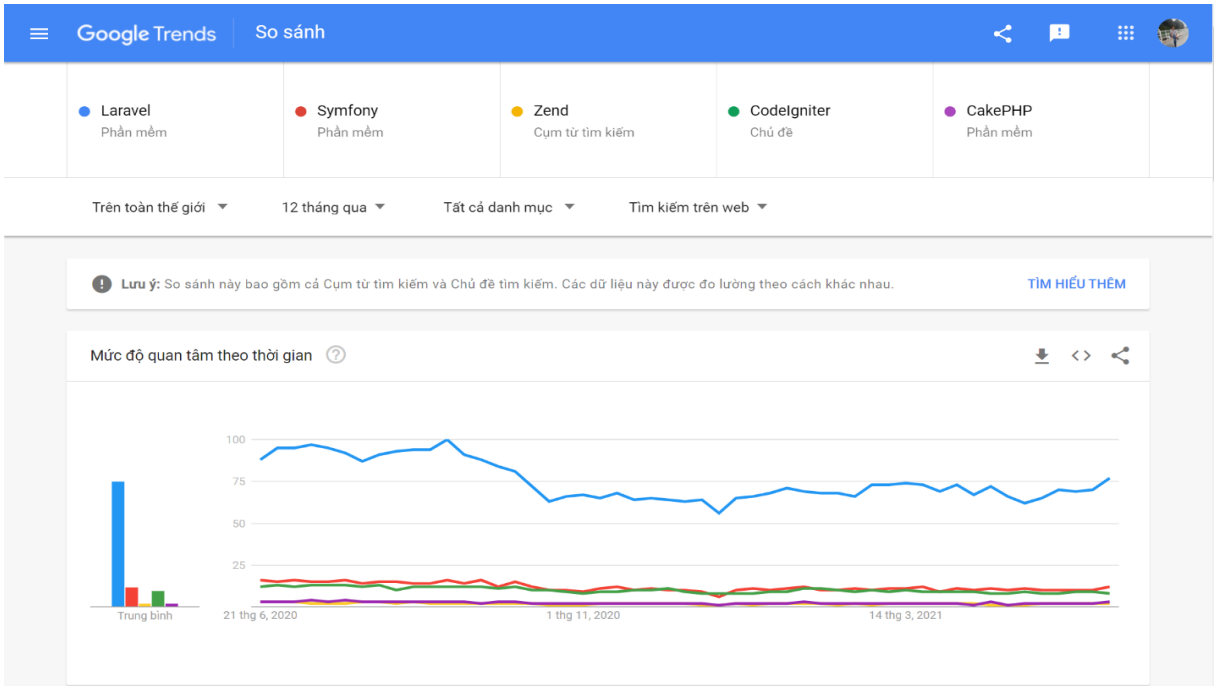


Top PHP Framework Statistics for 2021[1]

[1] https://www.excellentwebworld.com/best-php-frameworks/

1.3. Laravel là gì?

Laravel là một PHP Framework mã nguồn mở và miễn phí, được phát triển bởi Taylor Otwell và nhắm vào mục tiêu hỗ trợ phát triển các ứng dụng web theo kiếm trúc model-view-controller (MVC). Những tính năng nổi bật của Laravel bao gồm cú pháp dễ hiểu – rõ ràng, một hệ thống đóng gói modular và quản lý gói phụ thuộc, nhiều cách khác nhau để truy cập vào các cơ sở dữ liệu quan hệ, nhiều tiện ích khác nhau hỗ trợ việc triển khai vào bảo trì ứng dụng.



Google Trends [2021][2]

1.4. Các tính năng chính của Laravel Framework

Laravel Framework sở hữu một hệ sinh thái lớn bao gồm các tính năng như: instant deployment, routing, ORM, DB query, Routing, Templating,... Những tính năng có thể kể đến làm cho Laravel Framework khác biệt so với các Web Framework khác:

- Composer: sử dụng để nâng cấp, cài đặt ,...
- Eloquent ORM: thao tác với cú pháp đẹp mắt và đơn giản.
- Restful API: hỗ trợ biến Laravel thành một web service API.
- Artisan: cung cấp các lệnh cần thiết để phát triển ứng dụng.

2

https://trends.google.com/trends/explore?geo=VN&q=%2Fm%2F0jwy148,%2Fm%2F09cjcl,Zend,%2Fm%2F02qgdkj,%2Fm%2F09t3sp#TIMESERIES

- View: giúp code sạch sẽ hơn rất nhiều.
- Migrations: hỗ trợ tạo các trường trong cơ sở dữ liệu, thêm các cột trong bảng, tạo mối quan hệ giữa các bảng, hỗ trợ quản lý cơ sở dữ liệu.
- Authentication: cung cấp sẵn các tính năng đăng nhập, đăng ký, quên mật khẩu...
- Unit Testing: hỗ trợ test lỗi để sửa chữa.

2. CÁC PHIÊN BẢN CỦA LARAVEL

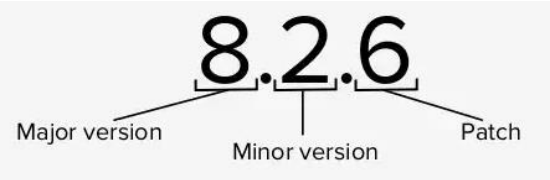
2.1. Lịch sử phát triển

Phiên bản đầu tiên của Laravel được Taylor Otwell tạo ra vào tháng 6 năm 2011 như một giải pháp thay thế cho CodeIgniter. Với framework này, lập trình viên được hỗ trợ nhiều tính năng mới mẻ, hiệu quả và dễ thực hiện hơn.

Laravel 6, phát hành vào ngày 3 tháng 9 năm 2019. Đây là version LTS.

Từ version 6, thì Laravel follow theo các tiêu chuẩn của Semantic Versioning nên cách đặt tên cũng thay đổi so với ngày trước.

Cụ thể, tên version của Laravel sẽ theo quy tắc `MAJOR.MINOR.PATCH`, với MAJOR là version lớn, được release kèm theo sự thay đổi về API khiến code của version cũ không hoạt động; MINOR là version nhỏ, được release kèm theo những chức năng mới, nhưng đảm bảo tính `backwards compatible`, tức code cũ vẫn sẽ hoạt động; PATCH version là release bao gồm bug fixes.



Nghĩa là:

Các bản release lớn sẽ có tên là số tăng dần từ 6.0.0 đến 7.0.0 rồi sắp tới sẽ là 8.0.0, 9.0.0 . Từ 6 lên 7 kéo theo nhiều thay đổi khiến bạn buộc phải update codes của mình.

Bản 7.1.0 là bản update nhỏ cho 7.0.0 , bản update này bổ sung thêm các tính năng, nhưng vẫn phù hợp với code cũ, nên bạn có thể update từ 7.0.0 lên 7.1.0 mà không gặp vấn đề gì.

Bản 7.1.1 là bản update có chứa các bản vá cho các lỗi được tìm ra ở phiên bản 7.1.0 , update sẽ không mang lại tính năng mới gì, nhưng sẽ giúp code base của framework hoạt động ổn định hơn.

6.X là tên để chi toàn bộ các phiên bản bắt đầu 6. Như 6.0.0, 6.2.0 hay 6.3.4 thì đều gọi chung là 6.X ... Tương tự như vậy, 7.X là viết tắt cho toàn bộ các phiên bản đầu 7.

Laravel 7, ra mắt ngày 3 tháng 3 năm 2020 với nhiều tính năng cũng như cải thiện tốc độ.

2.2. Các phiên bản đang được hỗ trợ

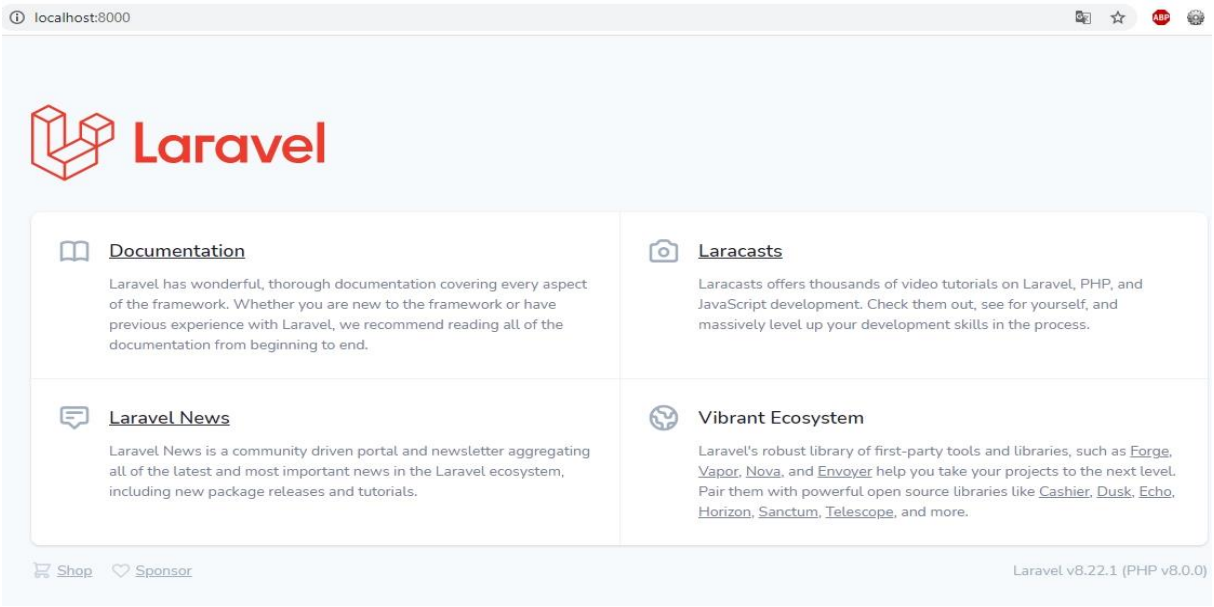
Version	Release	Bug Fixes Until	Security Fixes Until
6 (LTS)	September 3rd, 2019	January 25th, 2022	September 6th, 2022
7	March 3rd, 2020	October 6th, 2020	March 3rd, 2021
8	September 8th, 2020	July 26th, 2022	January 24th, 2023
9 (LTS)	January 25th, 2022	January 30th, 2024	January 28th, 2025
10	January 24th, 2023	July 30th, 2024	January 28th, 2025

2.3. Laravel 8 có gì mới[3]

Laravel 8 tiếp tục những cải tiến được thực hiện trong Laravel 7.x bằng việc ra mắt Laravel Jetstream, model factory classes, migration squashing, job batching, cải tiến rate limiting, cải tiến queue, dynamic Blade components, Tailwind pagination views, time testing helpers, cải tiến `artisan serve`, cải tiến event listener, sửa lỗi và nhiều cải tiến khác.

Trang welcome mới

Trang welcome default mới của Laravel 8 sử dụng TailwindCSS thay vì sử dụng bootstrap như trước. Đây là một css framework cũng khá mới và nổi ở thời điểm hiện tại.



[3] https://laravel.com/docs/8.x/releases#laravel-8

Laravel Jetstream

Laravel Jetstream là một bộ khung được xây dựng cho Laravel, là điểm khởi đầu cho dự án tiếp theo của bạn. Jetstream bao gồm các chức năng đăng nhập, đăng ký, xác thực địa chỉ email, xác thực 2 lớp (two-factor authentication), quản lý session, hỗ trợ API thông qua Laravel Sanctum, và tuỳ chọn quản lý team. Laravel Jetstream thay thế và cải tiến dựa trên authentication UI có sẵn ở phiên bản Laravel trước.

Jetstream được thiết kế bằng Tailwind CSS và cung cấp lựa chọn sử dụng Livewire hoặc Inertia.

Thư mục app/models

Kể từ Laravel 8, tất cả các Models sẽ được mặc định nằm trong thư mục `app/Models`, thay vì nằm bên trong `app` như các phiên bản trước. Theo một cuộc thăm dò do Taylor thực hiện, hơn 80% nhà phát triển đã phải thực hiện thủ công để làm việc này.

Ví dụ, khi chạy command `php artisan make:model Post`, thì class Post sẽ được tự động nằm trong `app/Models` thay vì như trước kia sẽ phải gõ `php artisan make:model Models/Post`.

Tuy nhiên, nếu muốn giữ các Model của mình trong thư mục `app` và xóa thư mục `models` mới, Laravel sẽ tạo các Model bên trong thư mục `app` như mong muốn.

Model Factory Classes

Eloquent model factories đã được viết lại hoàn toàn dựa trên class và cải tiến để hỗ trợ relationship.

Ví dụ, `UserFactory` có sẵn trong Laravel được viết như sau:

```
<?php

namespace Database\Factories;

use App\Models\User;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
```

```
     */
    protected $model = User::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'name' => $this->faker->name(),
            'email' => $this->faker->unique()->safeEmail(),
            'email_verified_at' => now(),
            'password' =>
'$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
            'remember_token' => Str::random(10),
        ];
    }
}
```

Nhờ `HasFactory` có sẵn khi khởi tạo models, model factory có thể sử dụng như sau:

```
use App\Models\User;

User::factory()->count(50)->create();
```

Vì model factories giờ đây là những class PHP, các state transformations có thể viết dưới dạng phương thức của class. Ngoài ra, có thể sử dụng bất kỳ helper class nào vào Eloquent model factory nếu muốn.

Ví dụ, `User` model có một state là `suspended` để thay đổi một trong những thuộc tính mặc định của nó. Có thể định nghĩa state transformations bằng phương thức `state` của base factory. Có thể đặt trên phương thức state tuỳ ý. Dù sao nó cũng chỉ là một phương thức PHP:

```
/**
 * Indicate that the user is suspended.
 *
 * @return \Illuminate\Database\Eloquent\Factories\Factory
 */
public function suspended()
{
    return $this->state([
```

```
            'account_status' => 'suspended',
        ]);
    }
```

Sau khi định nghĩa phương thức state transformation, có thể sử dụng như sau:

```
use App\Models\User;

User::factory()->count(5)->suspended()->create();
```

Như đã đề cập, Model factory của Laravel 8 hỗ trợ relationships. Vì vậy, giả sử `User` model chứa phương thức relationship là `posts` , có thể chạy đoạn code sau để tạo 3 post cho user:

```
$users = User::factory()
            ->hasPosts(3, [
                'published' => false,
            ])
            ->create();
```

Để đơn giản hoá quá trình nâng cấp, package laravel/legacy-factories đã được phát hành, cung cấp hỗ trợ cho các phiên bản trước của model factories trong Laravel 8.x.

Migration Squashing

Trong các phiên bản trước các file migration sẽ được đặt trong file trong thư mục `database/migrations` dẫn đến trường hợp về lâu dài thì số lượng này có thể tăng lên đến vài trăm file. Lúc này quả thật quản lý rất khó, nhận thấy điều này Laravel 8 đã tối ưu chúng và thêm một feature cho phép chuyển đổi tất cả file migation vào một file SQL và file này sẽ nằm trong thư mục `database/schema`. Nếu muốn, có thể "squash" migrations vào trong một file SQL duy nhất. để bắt đầu, chạy lệnh `schema:dump`:

```
php artisan schema:dump

// Dump database schema hiện tại và xoá tất cả migrations đang có...
php artisan schema:dump --prune
```

Lưu ý: Chức năng này chỉ hỗ trợ MySQL, PostgreSQL và SQLite.

Job Batching

Tính năng job batching của Laravel cho phép dễ dàng thực hiện một loạt các job, sau đó thực

hiện một số hành động khi batch thực thi xong. Phương thức `batch` của `Bus` facade có thể được sử dụng để dispatch một batch. Tất nhiên, batching chỉ thực sự hữu ích khi kết hợp với các callback. Vì vậy, bạn có thể sử dụng phương thức `then`, `catch` và `finally` để khai báo các callback cho batch. Mỗi callback này sẽ nhận instance của `Illuminate\Bus\Batch` khi được gọi.

Có 3 callback có sẵn, đó là:

- `then()`: sẽ kích hoạt khi tất cả các Job trong batch đã hoàn thành và thành công.
- `catch()`: sẽ kích hoạt khi có lỗi xảy ra trong batch.
- `finally()`: sẽ kích hoạt khi tất cả các Job trong batch đã hoàn thành thực thi (một số có thể đã thành công, một số có thể không).

Ví dụ, chúng ta có thể sử dụng `$batch` trong `catch()` để dừng phần còn lại của batch nếu có lỗi xảy ra. Đây là cách sử dụng để có thể tương tác với batch:

```
use App\Jobs\ProcessPodcast;
use App\Podcast;
use Illuminate\Bus\Batch;
use Illuminate\Support\Facades\Bus;
use Throwable;

$batch = Bus::batch([
    new ProcessPodcast(Podcast::find(1)),
    new ProcessPodcast(Podcast::find(2)),
    new ProcessPodcast(Podcast::find(3)),
    new ProcessPodcast(Podcast::find(4)),
    new ProcessPodcast(Podcast::find(5)),
])->then(function (Batch $batch) {
    // Xử lý khi tất cả các Job đã được thực thi thành công…
})->catch(function (Batch $batch, Throwable $e) {
    // Xử lý khi có lỗi xảy ra ở đây...
})->finally(function (Batch $batch) {
    // Tất cả các Job đã được chạy xong…
})->dispatch();

return $batch->id;
```

Ngoài ra còn có các tính năng nâng cao như khả năng thêm các Job khác vào một batch hiện có từ bên trong một Job trong batch đó bằng cách gọi `$this->batch()` trong một Job bất kỳ. Điều này hữu ích nếu đang xử lý một lượng lớn dữ liệu và cần phải làm việc với chunk.

Nếu có UUID của batch, có một phương thức helper khác có sẵn trên Bus

`Bus::findBatch($batchUuid)`. Thao tác này sẽ trả về một đối tượng chứa thông tin về batch, bao gồm cả tiến trình hiện tại. Có thể sử dụng nó để hiển thị tiến trình theo thời gian thực của batch trong giao diện người dùng bằng AJAX.

Cải thiện Rate Limiting

Tính năng Request rate limiter của Laravel đã được mở rộng với sự linh hoạt và mạnh mẽ, trong khi vẫn giữ được khả năng tương thích với `throttle` middleware API trong các phiên bản trước. Rate limiters được khai báo bằng phương thức `for` của `RateLimiter` facade. Phương thức `for` chấp nhận một rate limiter name và một Closure trả về thiết lập limit áp dụng cho route được gán rate limiter:

```
use Illuminate\Cache\RateLimiting\Limit;
use Illuminate\Support\Facades\RateLimiter;

RateLimiter::for('global', function (Request $request) {
    return Limit::perMinute(1000);
});
```

Vì rate limiter callback nhận instance của HTTP Request là tham số, bạn có thể xây dựng rate limit phân loại được dựa trên request hoặc người dùng:

```
RateLimiter::for('uploads', function (Request $request) {
    return $request->user()->vipCustomer()
                ? Limit::none()
                : Limit::perMinute(100);
});
```

Đôi khi muốn phân chia rate limit theo một số giá trị tuỳ ý. Ví dụ, muốn người dùng truy cập 1 route nào đó 100 lần mỗi phút dựa trên địa chỉ IP. Để thực hiện điều này, có thể sử dụng phương thức `by` khi xây dựng rate limit:

```
RateLimiter::for('uploads', function (Request $request) {
    return $request->user()->vipCustomer()
                ? Limit::none()
                : Limit::perMinute(100)->by($request->ip());
});
```

Rate limiter có thể được gán vào route bằng cách sử dụng `throttle` middleware. Middleware này nhận vào tên của rate limiter muốn gán cho route:

```
Route::middleware(['throttle:uploads'])->group(function () {
    Route::post('/audio', function () {
        //
    });

    Route::post('/video', function () {
        //
    });
});
```

Cải thiện Maintenance Mode

Trong các phiên bản trước của Laravel, tính năng maintenance mode `php artisan down` có thể bỏ qua bằng cách sử dụng "allow list" chứa các địa chỉ IP được phép truy cập vào ứng dụng. Tính năng này đã được loại bỏ để chuyển sang giải pháp "secret" / token đơn giản hơn. Khi ở maintenance mode, bạn có thể sử dụng tuỳ chọn `secret` để định nghĩa một bypass token.

```
php artisan down --secret="1630542a-246b-4b66-afa1-dd72a4c43515"
```

Sau khi đặt ứng dụng vào maintenance mode, có thể truy cập ứng dụng thông qua URL chứa token. Laravel sẽ trả về cookie để bypass maintenance mode cho trình duyệt.

```
https://example.com/1630542a-246b-4b66-afa1-dd72a4c43515
```

Sau khi truy cập route này, bạn sẽ được redirect về route `/`. Một khi cookie được tạo ra, bạn có thể truy cập ứng dụng bình thường như đang không trong maintenance mode.

Pre-Rendering The Maintenance Mode View

Nếu sử dụng `php artisan down` trong khi đang deployment, người dùng có thể đôi khi gặp phải lỗi nếu truy cập ứng dụng trong khi các dependencies của composer hoặc các infrastructure components khác đang được cập nhật. Điều này xảy ra bởi vì một phần của Laravel phải khởi chạy để kiểm tra xem ứng dụng có đang trong maintenance mode hay không và render view bằng template engine. Vì lý do này, Laravel 8.x cho phép pre-render một view cho maintenance mode sẽ được trả về ngay khi bắt đầu request cycle. View này đã được render trước khi bất kỳ dependencies nào của ứng dụng được load. Có thể pre-render một template tuỳ chọn bằng option `render` của lệnh `down` :

```
php artisan down --render="errors::503"
```

Closure Dispatch / Chain catch `catch`

Bằng cách sử dụng phương thức `catch` mới, bây giờ bạn có thể cung cấp một Closure sẽ được thực thi nếu queue Closure thất bại sau khi kết thúc tất cả các lần thử được thiết lập trong queue:

```
use Throwable;

dispatch(function () use ($podcast) {
    $podcast->publish();
})->catch(function (Throwable $e) {
    // This job has failed...
});
```

Dynamic Blade Components

Đôi khi muốn render một component nhưng không biết trước tên component cho đến khi thực chạy. Trong trường hợp này, có thể sử dụng component dynamic-component được xây dựng sẵn trong Laravel để render component dựa trên giá trị khi chạy hoặc biến:

```
<x-dynamic-component :component="$componentName" class="mt-4" />
```

Event Listener Improvements

Trong các phiên bản Laravel trước, khi đăng ký một Event Listener dựa trên Closure, trước tiên phải xác định Event class, sau đó đăng ký một Closure và type-hint Event class cho nó. Ví dụ:

```
Event::listen(OrderShipped::class, function(OrderShipped $event) {
    // Làm gì đó ở đây…
});
```

Thì trong Laravel 8, sẽ có thể bỏ qua tham số đầu tiên vì framework giờ đây đã có thể suy ra nó từ type-hint. Ví dụ:

```
Event::listen(function(OrderShipped $event) {
    // Làm gì đó ở đây…
});
```

Từ 8.x, event listener dựa trên Closure có thể được register chỉ bằng cách truyền Closure vào phương thức `Event::listen`. Laravel sẽ kiểm tra Closure để xác định loại event được xử lý.

```
use App\Events\PodcastProcessed;
use Illuminate\Support\Facades\Event;

Event::listen(function (PodcastProcessed $event) {
    //
});
```

Ngoài ra, event listener dựa trên Closure có thể được đánh dấu là queueable bằng cách sử dụng hàm `Illuminate\Events\queueable`:

```
use App\Events\PodcastProcessed;
use function Illuminate\Events\queueable;
use Illuminate\Support\Facades\Event;

Event::listen(queueable(function (PodcastProcessed $event) {
    //
}));
```

Giống như queued jobs, có thể sử dụng các phương thức `onConnection`, `onQueue` và `delay` để tuỳ chỉnh việc thực thi của queued listener.

```
Event::listen(queueable(function (PodcastProcessed $event) {
    //
})->onConnection('redis')->onQueue('podcasts')->delay(now()->addSeconds(10)));
```

Nếu muốn xử lý lỗi của anonymous queued listener, có thể truyền một Closure cho phương thức `catch` khi khai báo `queueable` listener:

```
use App\Events\PodcastProcessed;
use function Illuminate\Events\queueable;
use Illuminate\Support\Facades\Event;
use Throwable;

Event::listen(queueable(function (PodcastProcessed $event) {
    //
})->catch(function (PodcastProcessed $event, Throwable $e) {
    // The queued listener failed...
}));
```

Queueable anonymous event listeners

Trong Laravel 8, bạn sẽ có thể gửi một Job dựa trên Closure vào hàng đợi từ các callback trong Model Event. Ví dụ:

```
class User extends Model {
      protected static function booting()
      {
            static::created(queueable(function(User $user) {
                  // Làm gì đó ở đây…
            }))
      }
}
```

Các phiên bản Laravel trước đó không thể làm được việc này, mà bạn sẽ phải tạo một Event class và Event Listener class, sau đó implement interface `ShouldQueue` và sử dụng `ShouldQueue` trait. Tính năng mới này giúp bạn làm việc nhanh hơn và chính là hàm có namespace đầu tiên được giới thiệu trong Laravel: `Illuminate\Events\queueable`

Time Testing Helpers

Khi testing, có thể muốn thay đổi thời gian trả về bởi helper như `now` hoặc `Illuminate\Support\Carbon::now()`. Từ 8.x, class base feature test của Laravel sẽ có các helper cho phép thao tác với thời gian hiện tại:

```
public function testTimeCanBeManipulated()
{
    // Travel into the future...
    $this->travel(5)->milliseconds();
    $this->travel(5)->seconds();
    $this->travel(5)->minutes();
    $this->travel(5)->hours();
    $this->travel(5)->days();
    $this->travel(5)->weeks();
    $this->travel(5)->years();

    // Travel into the past...
    $this->travel(-5)->hours();

    // Travel to an explicit time...
    $this->travelTo(now()->subHours(6));

    // Return back to the present time...
```

```
    $this->travelBack();
```

Artisan `serve` Improvements

Lệnh `serve` của Artisan được cải tiến để tự động reload khi biến môi trường trong file `.env` được thay đổi. Trước đây khi bạn đổi thông tin `env` thì cần phải restart lại command thì env mới được apply. Nhưng giờ thì không cần nữa, bởi Laravel đã code thêm chức năng detect thay đổi trong file `.env` nếu thay đổi nó sẽ tự động restart lại.

Tailwind Pagination Views

Paginator của Laravel được cập nhật, thêm template cho Tailwind CSS. Sẽ được giới thiệu cụ thể sau.

Routing Namespace Updates

Trong các phiên bản trước của Laravel, `RouteServiceProvider` chứa thuộc tính `$namespace`. Thuộc tính này sẽ được tự động thêm vào trước các khai báo controller route và khi gọi đến helper `action` / `URL::action`. Trong Laravel 8.x, thuộc tính này mặc định được đặt là `null`. Nghĩa là sẽ không có prefix namespace nào được thực hiện bởi Laravel. Thay vào đó, trong ứng dụng Laravel 8.x, định nghĩa controller route sẽ được xác định bằng cú pháp chuẩn của PHP:

```
use App\Http\Controllers\UserController;

Route::get('/users', [UserController::class, 'index']);
```

Gọi đến `action` và các phương thức liên quan:

```
action([UserController::class, 'index']);

return Redirect::action([UserController::class, 'index']);
```

Nếu muốn sử dụng prefix giống như Laravel 7.x, chỉ cần thêm thuộc tính `$namespace` vào `RouteServiceProvider`.

Lưu ý: Những thay đổi này chỉ áp dụng cho ứng dụng khởi tạo bằng Laravel 8.x. Ứng dụng upgrade từ Laravel 7.x vẫn sẽ có thuộc tính `$namespace` trong `RouteServiceProvider`.

Gỡ bỏ Controller namespace prefix

Trong các phiên bản trước, file `RouteServiceProvider.php` có thuộc tính `$namespace` được sử dụng để đặt tiền tố (`App\Http\Controllers`) cho namespace của Controller một cách tự động . Nếu đang sử dụng **callable syntax** trong file route `web.php`, thì việc này sẽ dẫn đến việc tiền tố `App\Http\Controllers` bị chèn vào 2 lần dẫn đến namespace bị sai. Thuộc tính này đã bị gỡ bỏ trong Laravel 8, vì vậy có thể import các Controller của mình vào các file route mà không gặp vấn đề gì.

Xử lý lỗi các Queue Closure

Đôi khi chúng ta có thể gửi một Closure vào hàng đợi cho các tiến trình chạy dưới nền, thì việc xử lý lỗi xảy ra trong các tiến trình này lại khá khó khăn. Bởi nếu điều đó xảy ra, các lỗi sẽ được ghi lại vào bảng **failed_jobs** trong trong cơ sở dữ liệu, và chúng ta không thể thực thi bất kỳ đoạn code nào khác sau đó.

Giờ đây trong Laravel 8, bạn sẽ có thể đăng ký một callback để chạy khi công việc không được xử lý thành công, thông qua phương thức `catch()` ,tương tự như `fail()` trong một Job class thông thường. Ví dụ:

```
dispatch(function() {
     // Làm gì đó ở đây…
})->catch(function(Throwable $e) {
     // Xử lý nếu có lỗi được ném ra…
});
```

Thuộc tính Blade component

Trong Laravel 7, nếu bạn extend một Blade component (ví dụ: `DangerButton` component được extend từ `Button` component) thì DangerButton sẽ không có các thuộc tính được truyền cho nó từ `Button`. Điều này đã thay đổi trong Laravel 8, giờ đây tất cả các component con sẽ có các thuộc tính của component cha, giúp việc mở rộng các component dễ dàng hơn.

Tùy chỉnh thời gian thử lại sau mỗi Job failed

Bắt đầu với Laravel 8, có thể thêm một phương thức `backoff()` mới vào các Job class để trả về một mảng các số nguyên nhằm quyết định thời gian chờ giữa các lần thử nếu không thành công. Ví dụ:

```
class ExampleJob
```

```
{
    //
    public function backoff()
    {
        return [1, 5, 10];
    }
    //
}
```

Trong ví dụ trên, nếu Job không thành công trong lần thử đầu tiên, nó sẽ đợi 1 giây trước khi thử lại. Nếu Job sau đó tiếp tục không thành công trong lần thử thứ hai, nó sẽ đợi 5 giây trước khi thử lại. Sau đó, nếu Job vẫn không thành công vào lần thứ ba (và các lần sau), nó sẽ phải đợi 10 giây trước khi được tiếp tục thử lại.

Điều này có thể rất hữu ích nếu đang làm việc với một API giới hạn số lượt truy cập để tăng thời gian chờ đợi mỗi khi Job không thành công.

Cải thiện route caching

Route caching là chức năng mà ở đó Laravel sẽ cache lại tất cả các route của ứng dụng dưới dạng một mảng array giúp cho việc truy vấn route nhanh hơn thay vì phải luôn phân tích các file route để tìm ra chính xác route nào được yêu cầu.

```
php artisan route:cache
```

Tuy nhiên cho đến trước Laravel 8, route caching chỉ có thể hoạt động nếucác package đã cài đặt, không sử dụng Clousure trong route. Nếu không sẽ gặp lỗi khi chạy command bên trên.

Laravel 8 đã hỗ trợ route caching cho bất kỳ route nào, kể cả có sử dụng Clousure hay không.