BLADE TEMPLATES (tiếp theo)

1.1. Xây dựng layout sử dụng component trong Blade

Hầu hết các ứng dụng web hiện này đều phân chia bố cục theo các layout riêng, sau đó ghép chúng lại với nhau khi cần sử dụng đến. Theo cách này giúp cho code phần view đơn giản, dễ hiểu, dễ bảo trì hơn.

Định nghĩa component trong Blade template

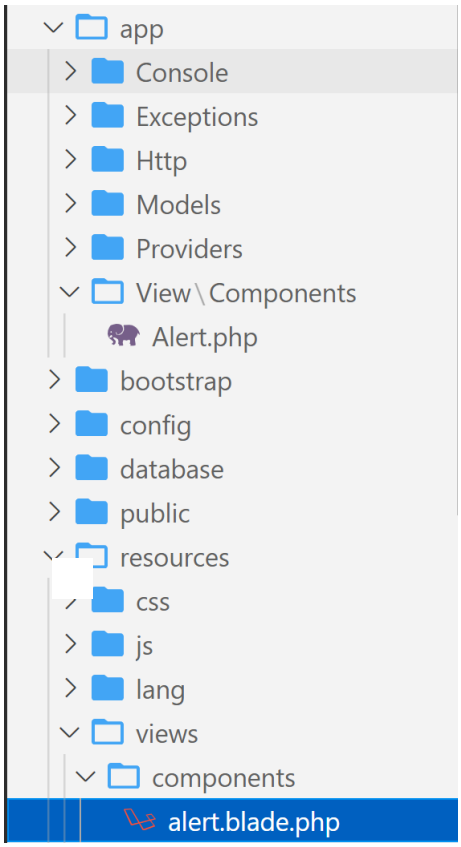
Để tạo component, sử dụng lệnh Artisan `make:component`. Lệnh the `make:component` sẽ đặt component trong thư mục `App\View\Components`:

Ví dụ:

```
php artisan make:component Alert
```

Lệnh `make:component` cũng sẽ tạo view template cho component. View sẽ được đặt trong thư mục `resources/views/components`. Khi viết các components cho ứng dụng, các components sẽ tự động được tìm trong thư mục `app/View/Components` và thư mục `resources/views/components`, vì vậy thường không cần đăng ký thêm cho component.

Kết quả:



Để khai báo một component cơ bản trong Laravel, chỉ cần khai báo chúng ở trong file `resources/views/components`.

Khai báo component `resources/views/components/alert.blade.php`:

```
<div>
    <h1>Demo Component</h1>
    <h2>{{ $slot }}</h2>
</div>
```

Trong đó:

- `$slot` là nơi sẽ hiện thị nội dung truyền vào component khi chúng được gọi ở view khác.

Rendering Components

Sau khi đã khai báo được component, muốn gọi chúng ở trong view thì có thể sử dụng cú pháp sau:

```
<x-componentname>
    //...
</x-componentname> Hoặc
<x-componentname/>
```

Trong đó:

- `componentname` là tên file component muốn gọi.

Ví dụ:

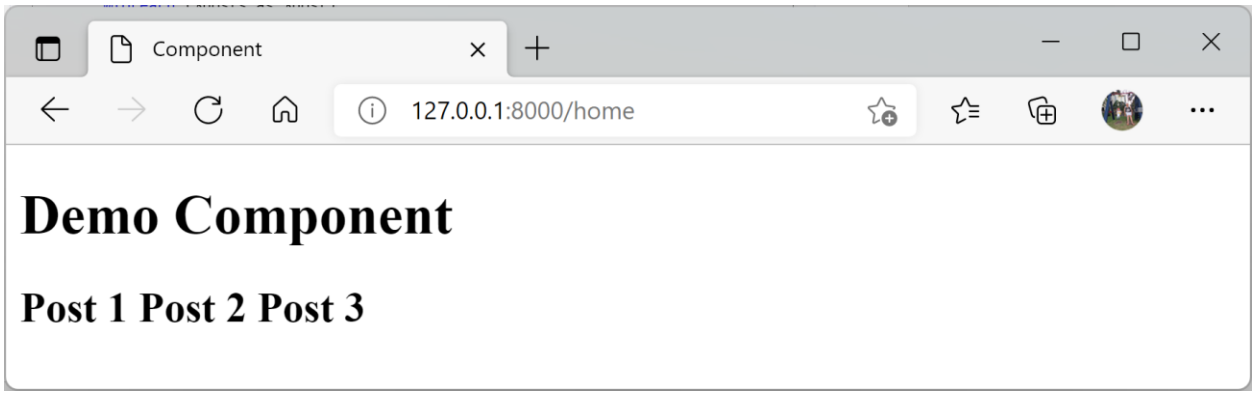
File `resources/views/home.blade.php`:

```
<body>
    <x-alert>
        @foreach ($posts as $post)
            <span>{{ $post }}</span>
        @endforeach
    </x-alert>
</body>
```

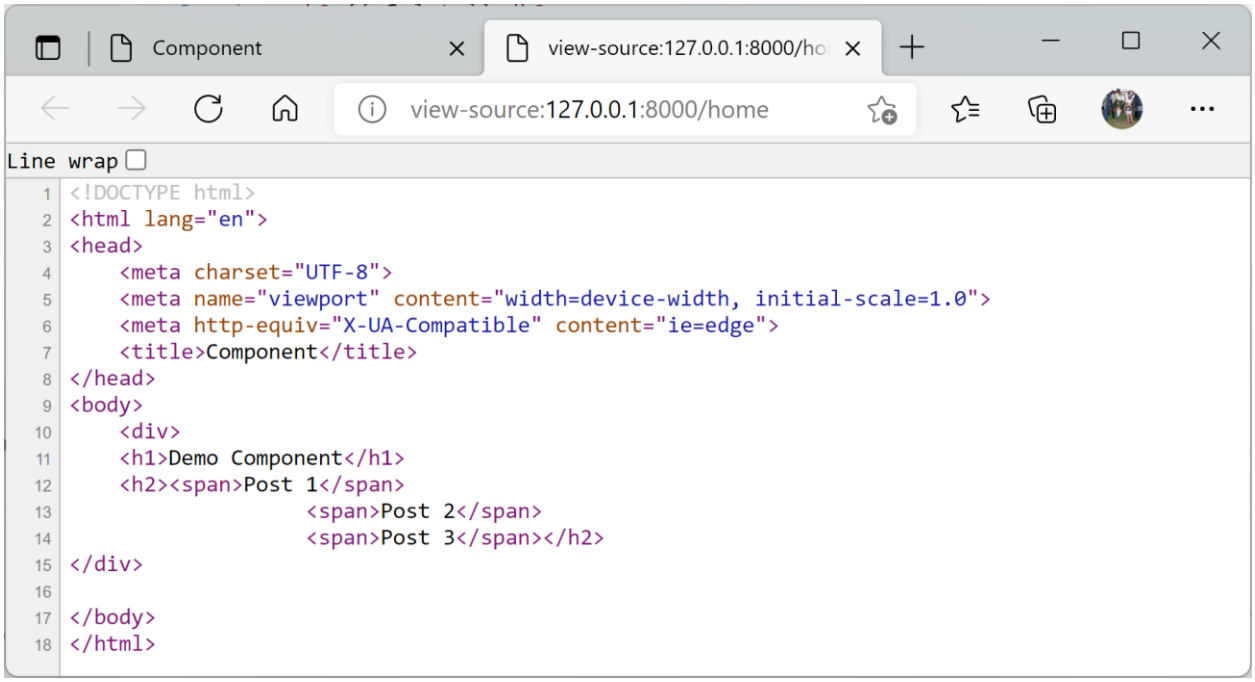
File `routes/web.php`:

```
Route::get('/home', function () {
    $posts = ['Post 1', 'Post 2', 'Post 3'];
    return view('home', ['posts' => $posts]);
});
```

Kết quả:



View page source:



Passing Data To Components

Cú pháp:

```
<x-slot name="VariableName">
    Data
</x-slot>
```

Trong đó:

- `VariableName` là tên biến trong component muốn truyền data vào;
- `Data` là dữ liệu muốn truyền vào.

Ví dụ: Truyền name vào component `resources/views/home.blade.php`:

```
<div>
    <h1>Demo Component</h1>
    <h2>{{ $slot }}</h2>

    <h3>{{ $message }}</h3>
</div>
```

File resources/views/home.blade.php:

```
<body>
    <x-alert>
        <x-slot name="message">
            Truyen du lieu vao Component!
        </x-slot>

        @foreach ($posts as $post)
            <span>{{ $post }}</span>
        @endforeach
    </x-alert>
</body>
```

File routes/web.php:

```
Route::get('/home', function () {
    $posts = ['Post 1', 'Post 2', 'Post 3'];
    return view('home', ['posts' => $posts]);
});
```

Kết quả:

1.2. Xây dựng layout sử dụng kế thừa trong Blade

Lợi ích của việc sử dụng Blade Template là kế thừa (inheritance) và sections. Hầu hết các ứng dụng web đều có cùng bố cục trên nhiều trang, vì vậy cách tốt nhất là xác định một mẫu chính, nơi đặt tất cả các code giống nhau trong các trang. Trong Laravel, Blade Template cho phép chúng ta xác định một mẫu chính có thể được mở rộng bởi các trang con khác nhau.

Xây dựng Layout Master

Xây dựng một layout trong File `resources/views/layouts/app.blade.php`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Layout</title>
</head>
<body>
    @section('sidebar')
        Day la master sidebar.
    @show

    @yield('content')

</body>
</html>
```

Trong đó:

- `@section` chỉ thị dùng để định nghĩa một nội dung của section;
- `@yield` chỉ thị dùng để chỉ định section có name trong yield sẽ được hiển thị. Ví dụ trên `@yield('title')` sẽ hiện thị nội dung của `@section('title')`.

Layout kế thừa

Xây dựng một view kế thừa layout ở trên, file `resources/views/home.blade.php`:

```
@extends('layouts.app')

@section('sidebar')
    @parent
   <h1>Day la noi dung them vao master sidebar!</h1>
@endsection
```

```
@section('content')
    <h2>Day la noi dung duoc them vao content.</h2>
@endsection
```

Trong đó:

- `@parent` lấy nội dung của layout cha (“Day la master sidebar.”);
- `@extends` chỉ thị để chỉ ra layout của trang này được kế thừa từ đâu.

  View kế thừa một Blade layout có thể chèn nội dung vào trong sections sử dụng `@section`, nội dung của những section này được hiển thị khi sử dụng `@yield`;

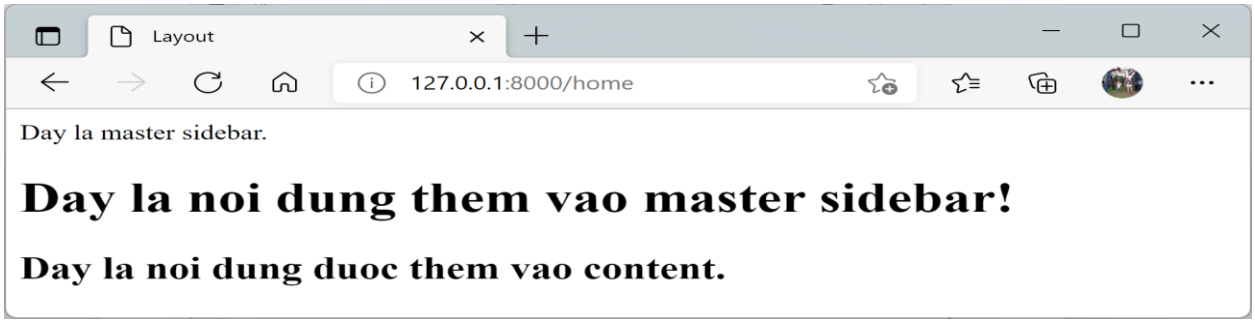
  Trong trường hợp muốn xác định giá trị mặc định sẽ hiển thị khi không có dữ liệu truyền vào trong `@yield` thì sử dụng cú pháp như sau:

```
@yield('name', 'Default content')
```

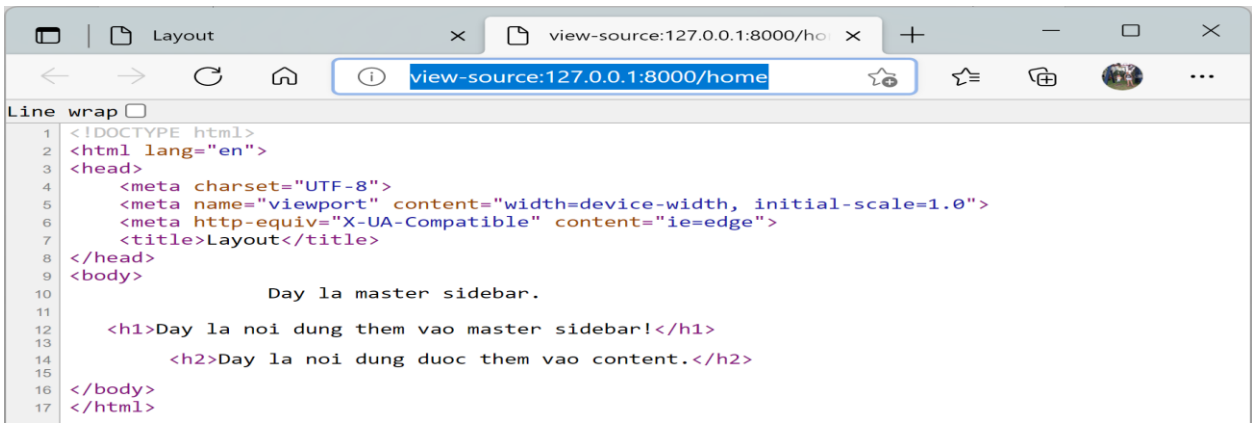
File `routes/web.php`:

```
Route::get('/home', function () {
    return view('home');
});
```

Kết quả:



View page source:



1.3. Form trong Blade template

Trong Blade Template, Laravel cũng đã cung cấp sẵn cho chúng ta một số chỉ thị hỗ trợ cho việc tạo ra các method, CSRF token.

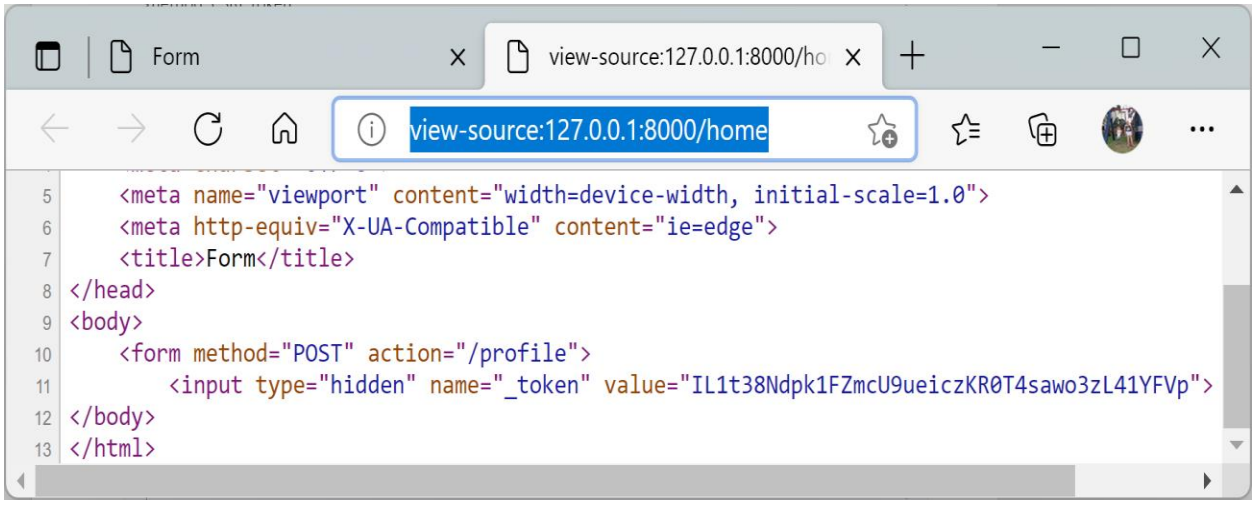
Để thêm một Input Field Hidden chứa `csrf_token` trong form, có thể sử dụng chỉ thị `@csrf`, cú pháp:

```
<form method="POST" action="/profile">
    @csrf
</form>
```

Ví dụ: File `resources/views/home.blade.php`:

```
<body>
    <form method="POST" action="/profile">
        @csrf
    </form>
</body>
```

Kết quả:

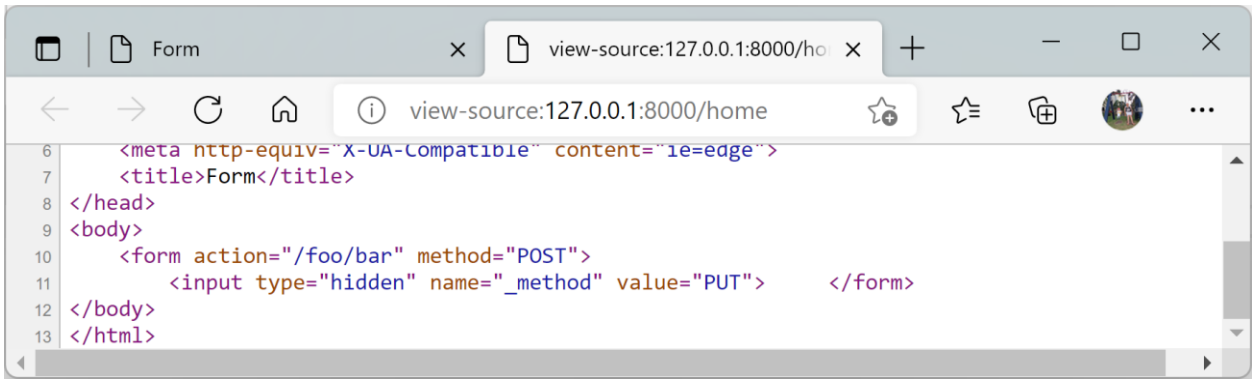


Ngoài ra, có thể sử dụng chỉ thị `@method` để thêm một Input Hidden chứa method của Form. Vì HTML Form không thể tạo được method `PUT`, `PATCH`, `DELETE`,...

Ví dụ: File `resources/views/home.blade.php`:

```
<body>
    <form action="/foo/bar" method="POST">
        @method('PUT')
    </form>
</body>
```

Kết quả:



1.4. Raw PHP

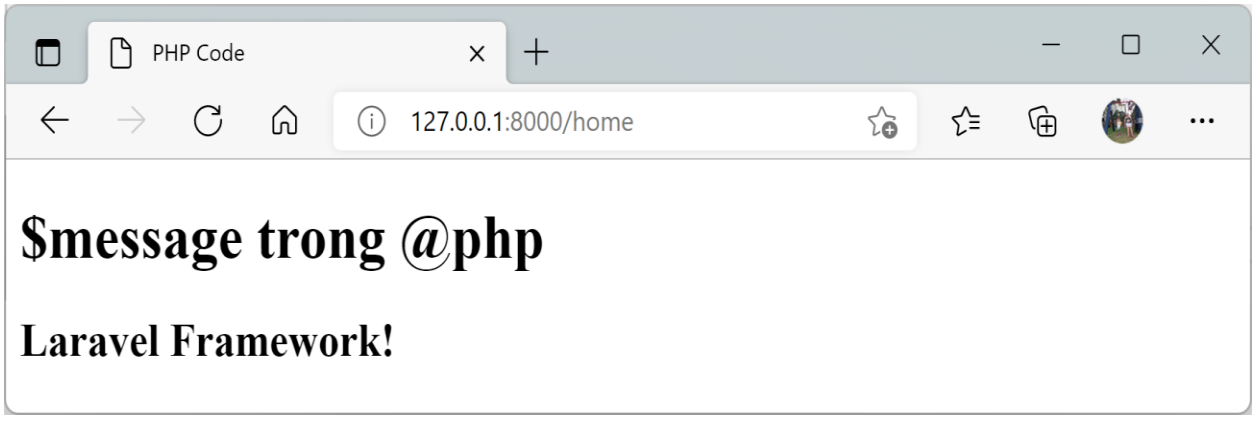
Trong một số trường hợp, nếu muốn sử dụng code PHP chứa logic code trong Blade Template, có thể sử dụng chỉ thị @php với cú pháp như sau:

```
@php(//Code PHP here)
Hoặc
@php
    //Code PHP here
@endphp
```

Ví dụ: File resources/views/home.blade.php:

```
<body>
    @php
        $message = "Laravel Framework!";
    @endphp
    <h1>$message trong @php</h1>
    <h2>{{ $message }}</h2>
</body>
```

Kết quả:



1.5. Stack trong Blade Template

Blade Template cung cấp chỉ thị `@push` để thực thi việc đẩy các dữ liệu vào `@stack` trong view.

Ví dụ:

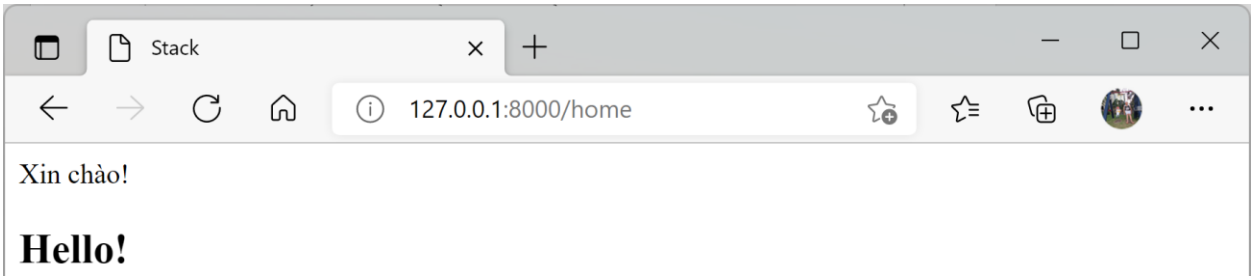
Tạo file `resources/views/layouts/app.blade.php`:

```
<body>
    @stack('content')
</body>
```

File `resources/views/home.blade.php`:

```
@extends('layouts.app')

@push('content', 'Xin chào!')

@push('content')
    <h2>Hello!</h2>
@endpush
```

Kết quả:



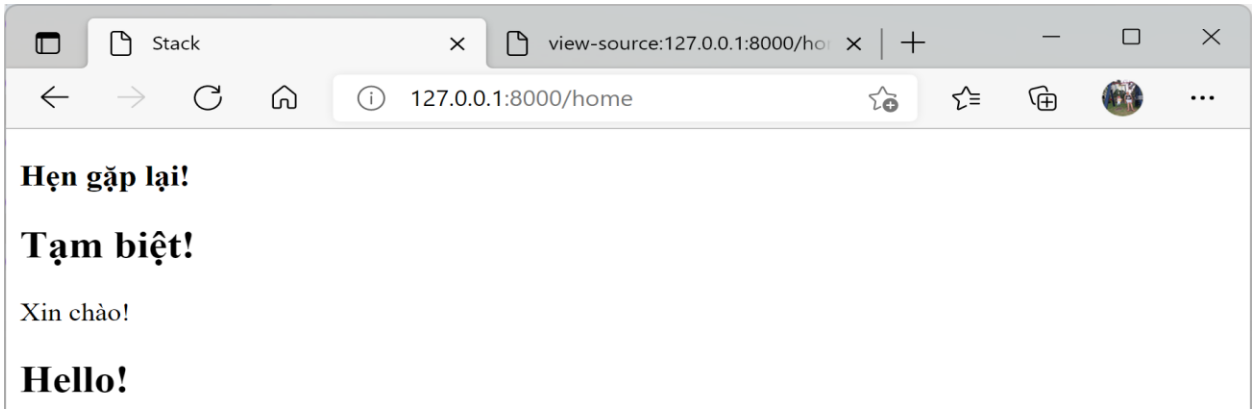
View page source:

Nếu muốn đẩy ngược nội dung lên phía trên thì có thể sử dụng chỉ thị `@prepend` thay cho `@push`.

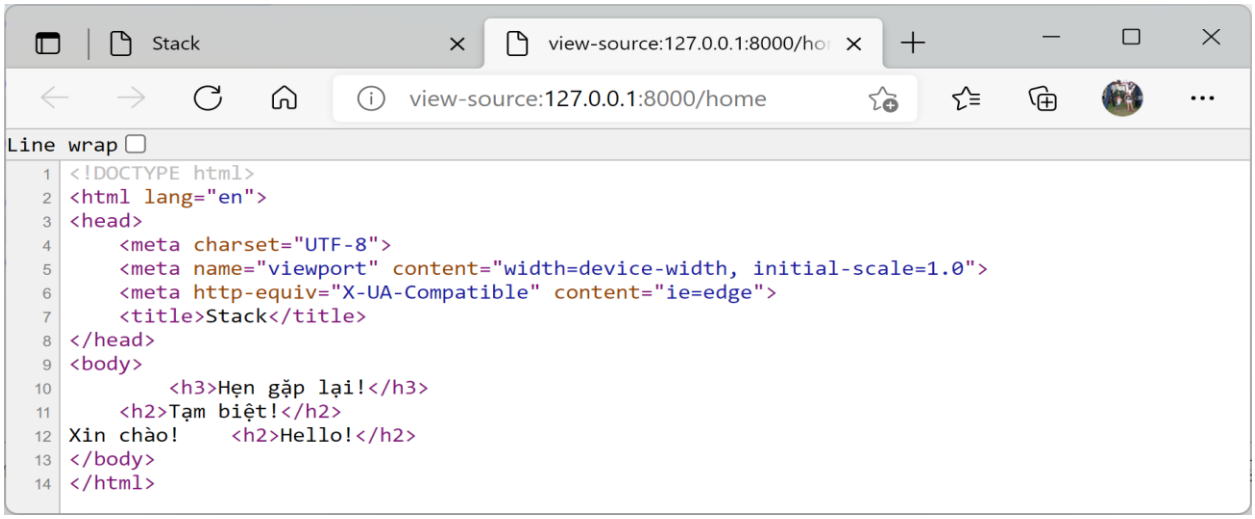
File `resources/views/home.blade.php`:

```
@extends('layouts.app')

@push('content', 'Xin chào!')

@push('content')
    <h2>Hello!</h2>
@endpush

@prepend('content')
    <h2>Tạm biệt!</h2>
@endprepend

@prepend('content')
    <h3>Hẹn gặp lại!</h3>
@endprepend
```

Kết quả:



View page source:



1.6. Tạo directive trong Blade

Trong trường hợp muốn tạo thêm directive riêng, thì Laravel cũng hỗ trợ tạo mới directive một cách hơn giản với cú pháp như sau:

```
use Illuminate\Support\Facades\Blade;

Blade::directive($directiveName, function ($value) {
    // Code here
});
```

Trong đó:

- `$directiveName` là tên directive mà bạn muốn tạo;
- `$value` là giá trị truyền vào khi sử dụng directive.

***Lưu ý**: Để việc tạo mới directive hoạt động tốt nhất có thể thì nên đưa code vào trong Provider.*

Ví dụ: Tạo mới direcive in ra ngày giờ với format **"y:m:d h:m:s"**:

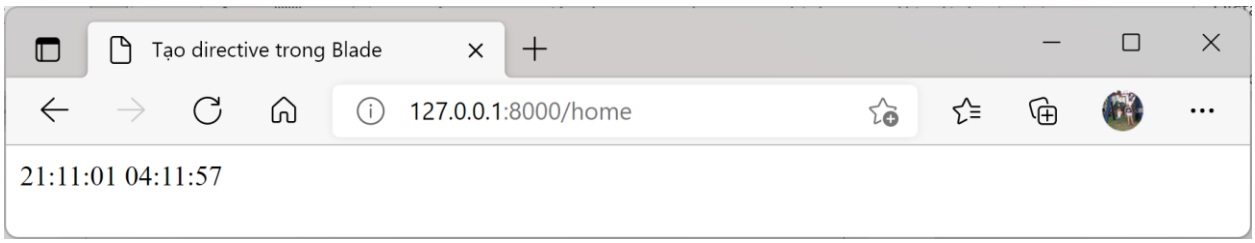
File `app/Providers/AppServiceProvider.php`:

```
use Illuminate\Support\Facades\Blade;

    public function boot()
    {
        Blade::directive('ngaygio', function($expression) {
            return "<?php echo ($expression)->format('y:m:d h:m:s'); ?>";
        });
    }
```

File `resources/views/home.blade.php`:

```
<body>
    @ngaygio(now())
</body>
```

Kết quả:



1.7. Cache view trong Laravel

Mặc định, Laravel luôn luôn cache lại các view đã được compile vào trong `storage/framework/views` sau đó mỗi request gọi đến, Laravel sẽ kiểm tra xem cache đó đã tồn tại hoặc hết hạn hay chưa? Nếu cache đã hết hạn (thời gian thay đổi file blade lớn hơn với thời gian thay đổi file cache) hoặc file cache chưa tồn tại thì Laravel sẽ thực hiện việc compile Blade vào cache lại.

Quá trình compile này có thể làm cho ứng dụng chậm đi (không đáng kể). Chính vì điều này Laravel có cung cấp hai command để cache và clear cache view trong một số trường hợp cần thiết.

Cache:

```
php artisan view:cache
```

Clear cache:

```
php artisan view:clear
```

BÀI TẬP

Cho Template HTML của Eshopper. Hãy chuyển sang Blade View trong Laravel.