Paginación

Configuración

En otros frameworks, la paginación puede resultar muy tediosa. Laravel hace este proceso muy sencillo. Laravel puede genera run "rango" inteligente de enlaces basados en la página actual. El HTML generado es compatible con el framework CSS TwitterBootstrap.

Usage

Hay varias formas de paginar elementos. La más sencilla es utilizar el método `paginate` del generador de consultas (querybuilder) o de un modelo Eloquent.

Paginando resultados de la base de datos

```
$users = DB::table('users')->paginate(15);
```

Nota: Actualmente, las operaciones de paginación que usan una declaración `groupBy` no se ejecutan de forma eficiente por Laravel. Si necesitas usar un `groupBy` con un conjunto de resultados paginados, se recomienda que consultes la base de datos y crees un paginador manual.

Crear un paginador manualmente

A veces es recomendable crear una instancia de paginación manualmente, pasándole un array de elementos. Puedes hacerlo mediante la creación de una instancia `Illuminate\Pagination\Paginator` o `Illuminate\Pagination\LengthAwarePaginator`, dependiendo de tus necesidades.

Paginar un modelo Eloquent

También puedes paginar modelos Eloquent:

```
$allUsers = User::paginate(15);

$someUsers = User::where('votes', '>', 100)->paginate(15);
```

El argumento pasado al método `paginate` es el número de elementos que deseamos mostrar por página. Una vez que se han obtenido los resultados puedes mostrarlos en tu vista y crear los enlaces de paginación utilizando el método `render`:

```
<div class="container">
<?php foreach ($users as $user): ?>
<?php echo $user->name; ?>
<?php endforeach; ?>
</div>

<?php echo $users->render(); ?>
```

¡Y esto es todo lo necesario para crear un sistema de paginación! Ten en cuenta que no hemos notificado al framework la página actual. Laravel gestionará esto automáticamente por nosotros.

También puedes acceder a información adicional sobre la paginación con los siguientes métodos:

- `currentPage // página actual`
- `lastPage // última página`
- `perPage // elementos por página`
- `hasMorePages // tiene más páginas?`
- `url`
- `nextPageUrl // URL de la página siguiente`
- `total`
- `count // número de páginas`

"Paginación simple"

Si solo estás mostrando los enlaces "siguiente" y "anterior" en tu vista, tienes la opción de utilizar el método `simplePaginate` para ejecutar una consulta mas eficiente. Esto es útil para grandes conjuntos de datos cuando no se requiere mostrar el número exacto de páginas en tu vista:

```
$someUsers = User::where('votes', '>', 100)->simplePaginate(15);
```

Personalizar la URI del paginador

Además permite personalizar la URI utilizada por el paginador utilizando el método `setPath`:

```
$users = User::paginate();

$users->setPath('custom/url');
```

El ejemplo anterior generará URLs del tipo: http://ejemplo.com/custom/url?page=2

Adjuntar variables o anclas a los enlaces de paginación

Se pueden añadir variables a la URL generada por el generador de links utilizando el método `appends` del paginador:

```
<?php echo $users->appends(['sort' => 'votes'])->render(); ?>
```

Esto generará unas URLs como las siguientes:

```
http://ejemplo.com/something?page=2&sort=votes
```

Si quieres además añadir un elemento de ancla (hash fragment) a las URLs generadas, puedes utilizar el método `fragment`:

```
<?php echo $users->fragment('foo')->render(); ?>
```

Esto generará la siguiente URL:

```
http://ejemplo.com/something?page=2#foo
```

Conversión a JSON

La clase `Paginator` implementa el contrato `Illuminate\Contracts\Support\JsonableInterface` y expone el método `toJson`. También puedes convertir una instancia de `Paginator` a JSON retornándolo desde una ruta. La forma JSON de la instancia incluirá alguna información "meta" como `total`, `current_page`, y `last_page`. La instancia de datos estará disponible en el elemento `data` del array JSON.