

Laravel

SEARCH

Documentation

Prologue

Release Notes

Upgrade Guide

Contribution Guide

API Documentation

Getting Started

Installation

Configuration

Directory Structure

Request Lifecycle

Dev Environments

Homestead

Valet

Core Concepts

Service Container

Service Providers

Facades

Contracts

The HTTP Layer

Routing

Middleware

CSRF Protection

Controllers

Requests

Responses

Views

Session

Validation

Frontend

Paginat

Introduction

Basic Usage

- # Paginating Que
- # Paginating Elo
- # Manually Creat

Displaying Paginat

- # Converting Res

Customizing The P

Paginator Instance

Introduction

In other frameworks, pa
[builder](#) and [Eloquent OF](#)
of the box. The HTML ge

Basic Usage

Paginating Que

There are several ways to
[query builder](#) or an [Eloq](#)
proper limit and offset b
page is detected by the v

LARAVEL: PAGINACIÓN Y VALIDACIÓN

DISEÑO DE SISTEMAS SOFTWARE

Contenido

1. Paginación de resultados
2. Validación de formularios

Laravel: Paginación y validación

PAGINACIÓN DE RESULTADOS

Paginación de resultados

- Cuando se muestra un conjunto grande de datos (p.ej. un listado o los resultados de una búsqueda) es conveniente paginarlos
 - Mejora el rendimiento de la aplicación
 - Facilita la navegación del usuario
- Laravel proporciona métodos que facilitan la paginación
 - Se pueden realizar consultas indicando el número de resultados esperados
 - Permite generar los enlaces en plantillas Blade para navegar por los resultados

Consultas con paginación

- Al realizar una consulta con Eloquent es posible indicar que queremos usar paginación y el número de resultados que debe devolver cada vez

```
class UserController extends Controller {  
    public function index() {  
        $users = DB::table('users')->paginate(15);  
        return view('user.index', ['users' => $users]);  
    }  
}
```

- No es necesario realizar ninguna operación adicional en el controlador
- Cada vez que se le vuelva a llamar usando los enlaces generados automáticamente devolverá la página solicitada

Consultas con paginación

- El método `paginate()` se puede usar también con los resultados de una búsqueda

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

- Si sólo necesitamos los enlaces a las páginas anterior y siguiente se puede usar el método `simplePaginate()`

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

Enlaces para paginación

- Para generar los enlaces en una vista Blade se usa el método `links()`

```
<div class="container">
    @foreach ($users as $user)
        {{ $user->name }}
    @endforeach
</div>

{{ $users->links() }}
```

- Se pueden añadir parámetros a los enlaces con el método `appends()`

```
{{ $users->appends(['sort' => 'votes'])->links() }}
```

Enlaces para paginación

- Los enlaces que genera el método `links()` usan el framework CSS Bootstrap por defecto
- Para cambiar la manera en que se muestran los enlaces hay que modificar las plantillas que usa Laravel
 - Primero hay que exportar las plantillas a una carpeta de la aplicación con el comando

```
php artisan vendor:publish --tag=laravel-pagination
```

- Esto creará una copia de las plantillas en la carpeta `resources/views/vendor/pagination`
 - `default.blade.php` **se usa con el método** `paginate()`
 - `simple-default.blade.php` **con** `simplePaginate()`

Laravel: Paginación y validación

VALIDACIÓN DE FORMULARIOS

Validación de formularios

- Los controladores de Laravel permiten validar los datos de entrada de forma sencilla
 - Se pueden indicar distintas reglas de validación para cada campo del formulario
 - Si la validación falla, el controlador redirigirá automáticamente de vuelta a la vista con el formulario
 - Las vistas tienen acceso a los mensajes de error producidos por la validación para mostrarlos al usuario
 - Se pueden mantener los datos introducidos por el usuario al volver a mostrar el formulario

Validación: rutas

- Supongamos que queremos crear posts en un foro, para esto definimos dos rutas en `web.php`

```
// Muestra el formulario  
Route::get('post/create', 'PostController@create');  
  
// Recibe los datos del formulario  
Route::post('post', 'PostController@store');
```

Validación: controlador

- A continuación creamos el controlador `PostController` con el siguiente código:

```
class PostController extends Controller {  
    public function create() {  
        return view('post.create');  
    }  
  
    public function store(Request $request) {  
        // Aquí añadiremos el código para validar  
        // el formulario y crear el post  
    }  
}
```

Validación: formulario

- Creamos la vista con el formulario en
resources/views/post/create.blade.php

```
<h1>Nuevo Post</h1>
<form action="{{ action('PostController@store') }}"
      method="POST">
    {{ csrf_field() }}
    <label for="title">Título</label>
    <input type="text" name="title" id="title">
    <br>
    <label for="body">Cuerpo</label>
    <textarea name="body" id="body"></textarea>
    <br>
    <input type="submit">
</form>
```

Validación: reglas de validación

- Añadimos el código en el controlador para validar los datos del formulario

```
public function store(Request $request) {  
    // Si la validación falla redirigirá automáticamente  
    // al formulario  
    $this->validate($request, [  
        'title' => 'required|unique:posts|max:255',  
        'body' => 'required',  
    ]);  
  
    // Si llega hasta aquí la validación ha sido correcta  
    // Creamos el post...  
}
```

- Reglas de validación disponibles:

<https://laravel.com/docs/5.5/validation#available-validation-rules>

Validación: mostrar valores anteriores

- Desde una vista se puede acceder a los valores que ha escrito el usuario al enviar el formulario para no volver a mostrarlo vacío si ha ocurrido un error de validación

```
<label for="title">Título</label>
<input value="{{ old('title') }}" type="text"
      name="title" id="title">
<br>
<label for="body">Cuerpo</label>
<textarea name="body" id="body">
  {{ old('body') }}</textarea>
```

Validación: mostrar errores

- De nuevo en la vista, añadimos el código para mostrar los errores de validación

```
<h1>Nuevo Post</h1>
{{-- Error messages --}}
@if (count($errors) > 0)
    <ul>
        @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
@endif
<form action="{{ action('PostController@store') }}"
    method="POST">

...

```


Validación: mostrar errores

- También es posible mostrar los errores de forma individual para cada campo del formulario

```
<form action="{{ action('PostController@store') }}"
method="POST">
  {{ csrf_field() }}
  <label for="title">Título</label>
  {{-- Error messages --}}
  @if ($errors->has('title'))
    <ul>
      @foreach ($errors->get('title') as $error)
        <li>{{ $error }}</li>
      @endforeach
    </ul>
  @endif
  <input type="text" name="title" id="title">
  ...
```

Validación: idioma de los mensajes

- Los mensajes de error están definidos en el archivo `resources/lang/<idioma>/validation.php`
- Por defecto las aplicaciones de Laravel usan el inglés e incluyen todos los mensajes en `resources/lang/en/validation.php`
- Se puede cambiar el idioma de la aplicación en `config/app.php`

```
'locale' => 'es',
```

- Al cambiar el idioma hay que proporcionar las traducciones en la carpeta correspondiente
- Proyecto con traducciones a 53 idiomas:
<https://github.com/caouecs/Laravel-lang>

Validación: mensajes personalizados

- En el archivo `validation.php` se pueden definir mensajes personalizados, especificando el nombre del campo y la regla de validación

```
'custom' => [  
    'attribute-name' => [  
        'rule-name' => 'custom-message',  
    ],  
    'title' => [  
        'required' => 'Debes escribir un título para el post',  
    ],  
],
```