**Laravel**

**Carpeta app ->** archivos de nuestro proyecto.

**Bootstrap ->** Entornos y codificación de laravel (no tocar).

**Vendor ->** El directorio vendor contiene todos los paquetes de Composer que son utilizados por tu aplicación.

**CARPETA PUBLIC:**

**Public -> Packages ->** Será usado para contener cualquier fichero que necesite ser instalado por paquetes de terceras partes. Se mantienen en un directorio separado para que no creen conflictos con los ficheros de nuestra propia aplicación.

**Public -> htaccess ->** para servidores Apache

**Public -> favicon.ico ->** Este es el archivo que controla la pequeña imagen de 16 x 16px (en blanco, cambiar cuando queramos)

**CARPETA APP:**

**config ->** La configuración tanto para el framework como para tu aplicación se mantiene en este directorio.

**Controllers->** Como el nombre sugiere, este directorio contendrá tus controladores. Los controladores pueden ser usados para facilitar lógica a la aplicación, y hacer de pegamento entre las partes separadas de tu aplicación. Este directorio ha sido añadido al archivo composer.json por defecto para la auto carga de clases.

**Database->** Si escoges una base de datos como método para guardar cosas a largo plazo, este directorio sera usado para contener los archivos que crearán el esquema de tu base de datos, y los métodos para completarla con datos de ejemplo. La base de datos por defecto en SQLite también está ubicada en este directorio.

**lang ->**El directorio lang contiene archivos PHP con matrices de cadenas que pueden ser usados para dar soporte de traducción a tu aplicación. Se pueden crear sub carpetas por región para que tengas distintos ficheros para múltiples idiomas.

**storage ->**Cuando Laravel necesita escribir algo en el disco, lo hace en el directorio storage. Por este motivo tu servidor web debe poder escribir en esta ubicación.

**Tests->** El directorio tests contiene todas las pruebas unitarias y de aceptación para tu aplicación. La configuración por defecto de PHPUnit que ha sido incluida por Laravel buscará pruebas en este directorio por defecto

**Views**-> El directorio views es usado para contener las plantillas visuales de tu aplicación. Se facilita una vista hello por defecto para tu conveniencia.

**Routes.php->** El archivo para guardar las rutas de tu pagina.

**APP/ROUTES:**

Ejemplo:

<http://dominio.com/mi/pagina>

Para añadir esta ruta, tenemos que añadir en el archivo route.php

<?php

// app/routes.php

Route::get('mi/pagina', function() {

return '¡Hola mundo!';

});

**Route::get('primera/pagina', function() {**

**6return '¡Primera!';**

**});**

**Route::get('segunda/pagina', function() {**

**return '¡Segunda!';**

**});**

**Route::get('tercera/pagina', function() {**

**return '¡Patata!';**

**});**

Intenta navegar a las siguientes URLs para ver cómo se comporta nuestra aplicación.

**http://dominio.com/primera/pagina**

**http://dominio.com/segunda/pagina**

**http://dominio.com/tercera/pagina**

**EJEMPLO BÁSICO DE PÁGINAS CON RUTAS Y SUBPAGINAS**

Estructura = /libros a /libros/genero

Route::get('/libros', function() {});

Route::get('/libros/ficcion', function() {});

Route::get('/libros/ciencia', function() {});

Route::get('/libros/romance', function() {});

Route::get('/libros', function() {

return 'Índice de libros.'; //En la página inicial libros retorna “índice de libros”

});

Route::get('/libros/{genero}', function($genero){

//en la pagina /libros/crimen o /ciencia o/ficción o/romance la entenderá y saldrá “Libros en la categoría $genero(ciencia,ficción,romance,crimen)”

return "Libros en la categoría {$genero}.";

});

CÓMO CREAR UNA PÁGINA

**<!-- app/views/simple.php -->** //el archivo simple.php es nuestra pagina



Entonces en routes.php ->

ruta.:get('/', function() {

return **View::make('simple');**

});

Con **.blade.php** (en views, igual que las paginas normales) programamos de la siguiente manera:



<!-- Este es un encantador comentario HTML. -->

{{-- Este es un comentario bonito y secreto de Blade. --}}

@foreach ($muchasCosas as $cosa)

<p>{{ $cosa }}</p>

@endforeach

@for ($i = 0; $i < 999; $i++)

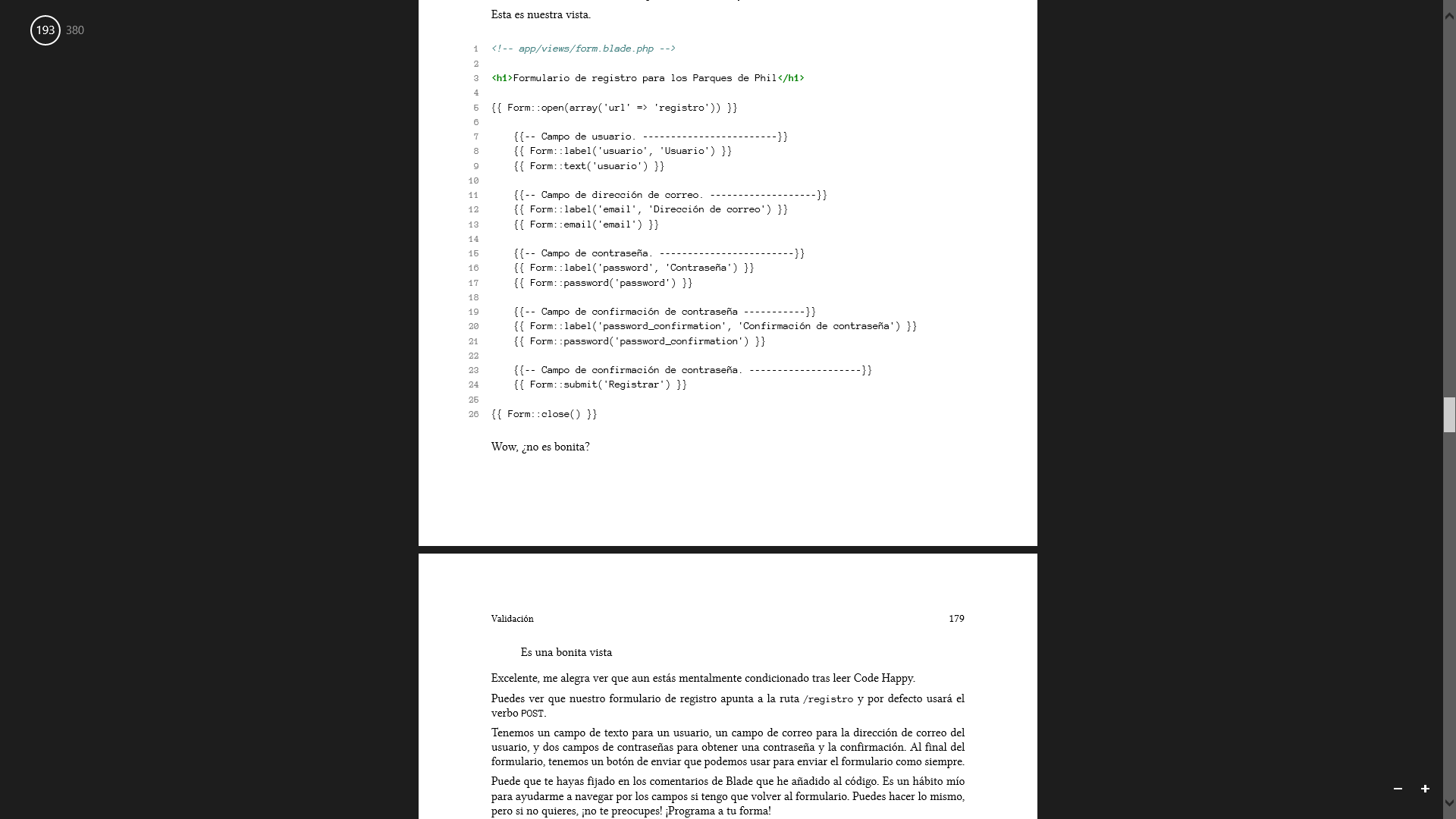
<p>¡Ni siquiera {{ $i }} pandas rojos, son suficientes!</p>

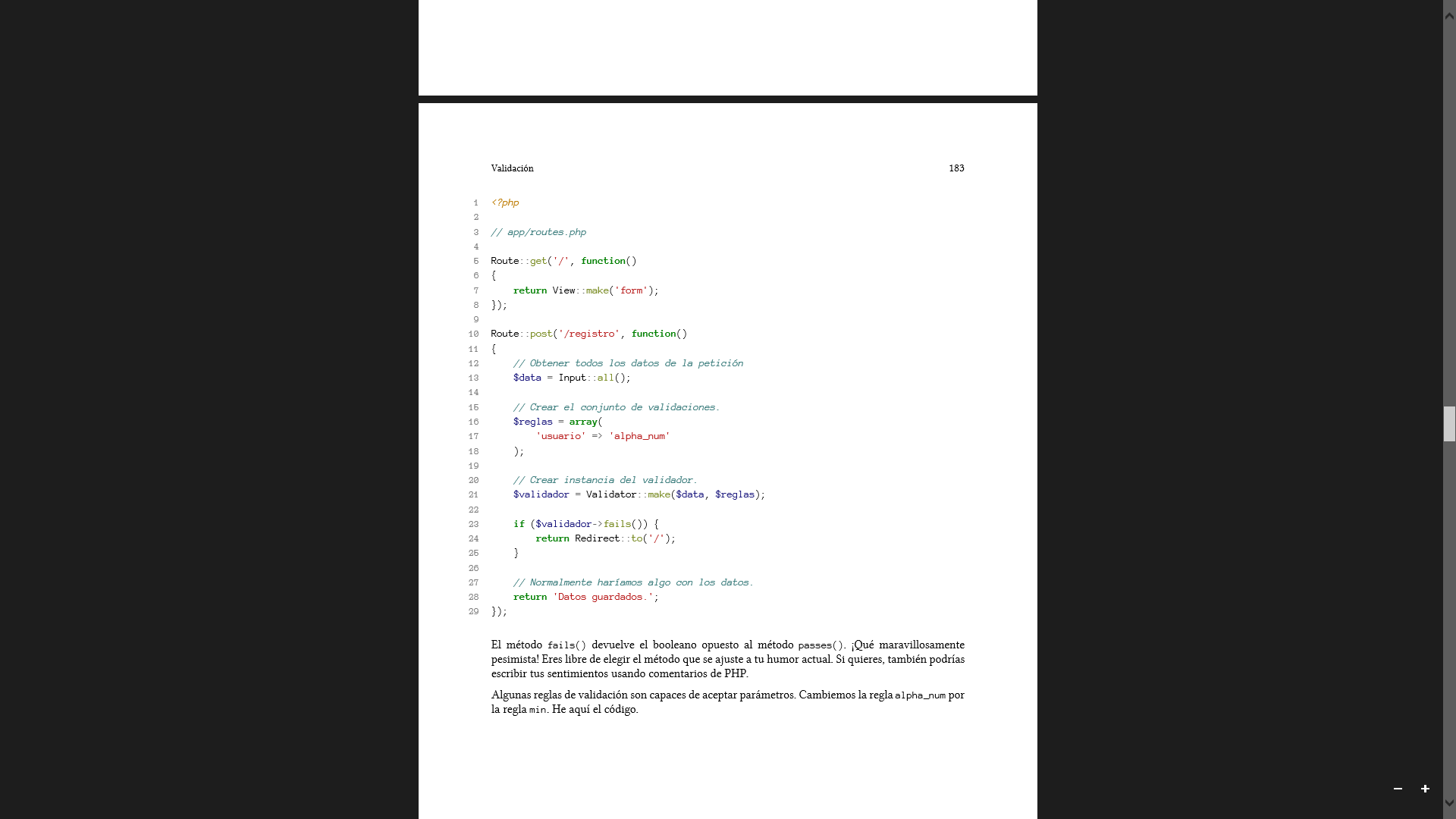
@endforeach

**Resumen: No hace falta <?php ?> cada vez que insertemos php.**

@include(‘header’) -> Incluyrá la página que tengas en views header.blade.php

**CREAR FORMULARIO**





**REGLAS DE FORMULARIO:**

**alpha\_dash** La regla alpha\_dash seasegura de que el valor contiene valoresalfabéticos ytambiénguiones - y/o guiones bajos \_.

**accepted** Estareglapuedeserusadaparaasegurardequesehafacilitadounaconfirmaciónpositiva.Pasarási el valor es uno de los siguientes valores: yes, on o el valor numérico 1. Su propósito es para cuando necesitas asegurarte de que un usuario ha aceptado algo, por ejemplo, los términos y condiciones.

**after** La regla de validación after acepta un único parámetro, una cadena que representa una fecha. La regla se asegurará de que el campo contiene una fecha que ocurre tras el parámetro facilitado. Laravel usará el método startotime() de PHP para convertir ambos valores y compararlos.

**alpha** La regla de validación alpha se asegura de que el campo solo contiene caracteres alfabéticos.

**alpha\_num** La regla alpha\_num se asegura de que el valor consiste en caracteres alfanuméricos. Me gusta usar esta regla en campos de nombres de usuario.

**date** La regla de validación date se asegurará de que nuestro valor es una fecha válida. Será confirmada usando el método strtotime() de PHP.

**date\_format** La regla de validación date\_format se asegura de que nuestro valor es una cadena de fecha que coincide con el formato facilitado como parámetro. Para aprender a cómo construir cadenas de formato de fechas, echa un vistazo a la documentación de PHP del método date().

**email** La reglade validación email se asegura de que el campo validado es una dirección de correo válida. Esta regla es muy útil a la hora de crear formularios de registro

**in** La validación in se asegura de que el valor coincide con uno de los parámetros.

array( 'campo' => 'in:rojo,marron,blanco' );

integer ¡Esta es fácil! La validación integer se asegura de que el campo es un entero. ¡Eso es todo!

array( 'campo' => 'integer' );

**max** Mayor o igual

array( 'campo' => ‘max:5' );

**min** menor o igual

array( 'campo' => 'min:5' );

**numeric** La regla numeric s asegura de que el campo contiene un valor numérico.

**required** La regla de validación required puede ser usada para asegurarte de que el campo actual existe en la matriz de datos.

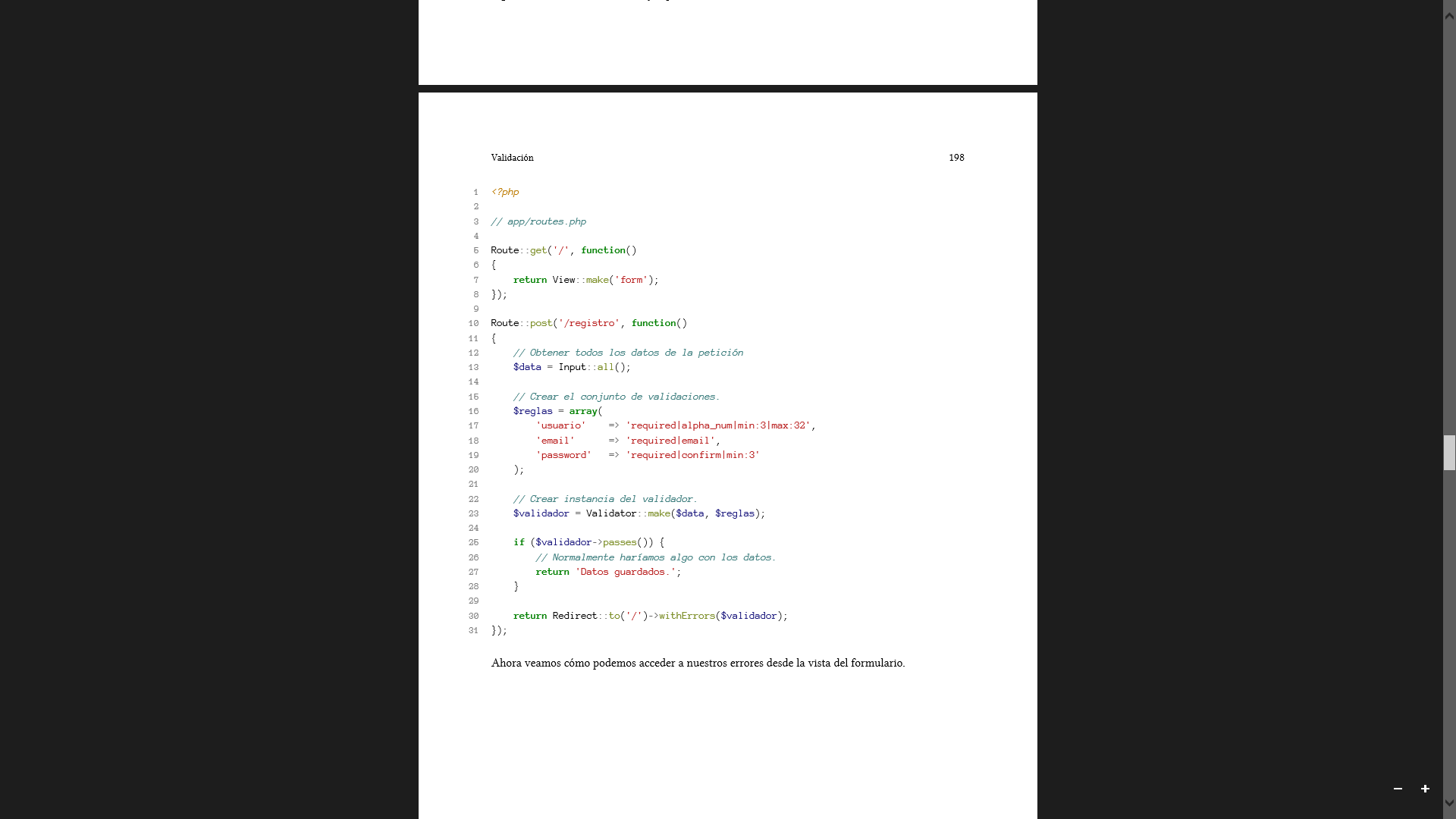
**size** Laregla size puede serusada paraasegurarde que el valorde uncampoes deun tamañoconcreto. Sielcampoesunacadena,elparámetrosereferiráalalongituddelamisma.Paravaloresnuméricos, la comparación será matemática. Para subidas de archivos, la comparación será contra el tamaño del archivo en bytes

array( 'campo' => 'size:8' );

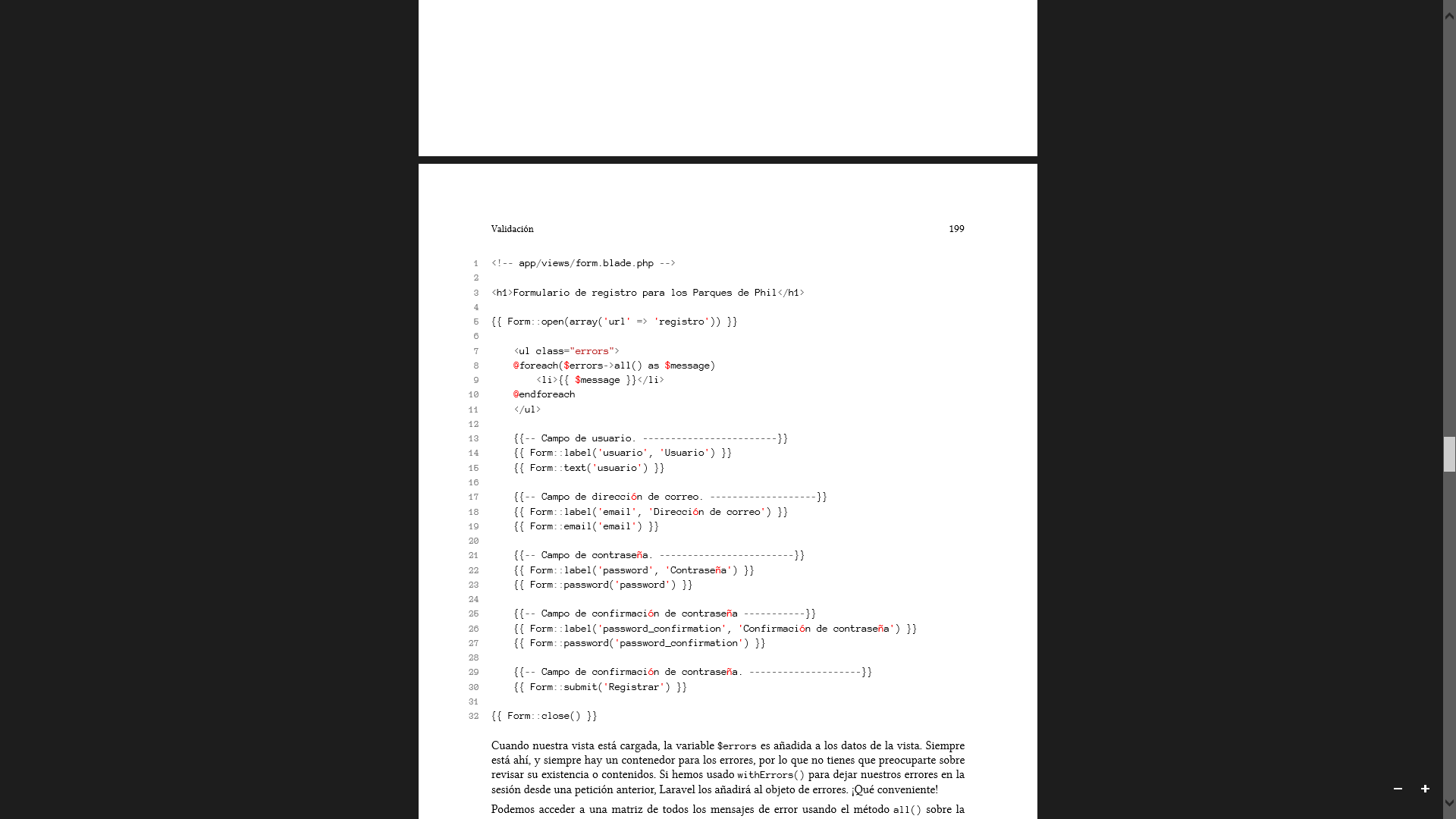
**unique** La regla unique se asegura de que el valor del campo actual no está presente en la tabla de la base dedatosdefinidaporelparámetrodelaregla.Pordefecto,lareglausaráelnombredelcampocomo columna de la tabla en la que buscar el valor, no obstante, puedes usar una columna alternativa en el segundo parámetro de la regla. Esta regla es útil para revisar si un usuario provee un nombre de usuario único a la hora de gestionar formularios de registro.

array( 'campo' => 'unique:users,usuario' );

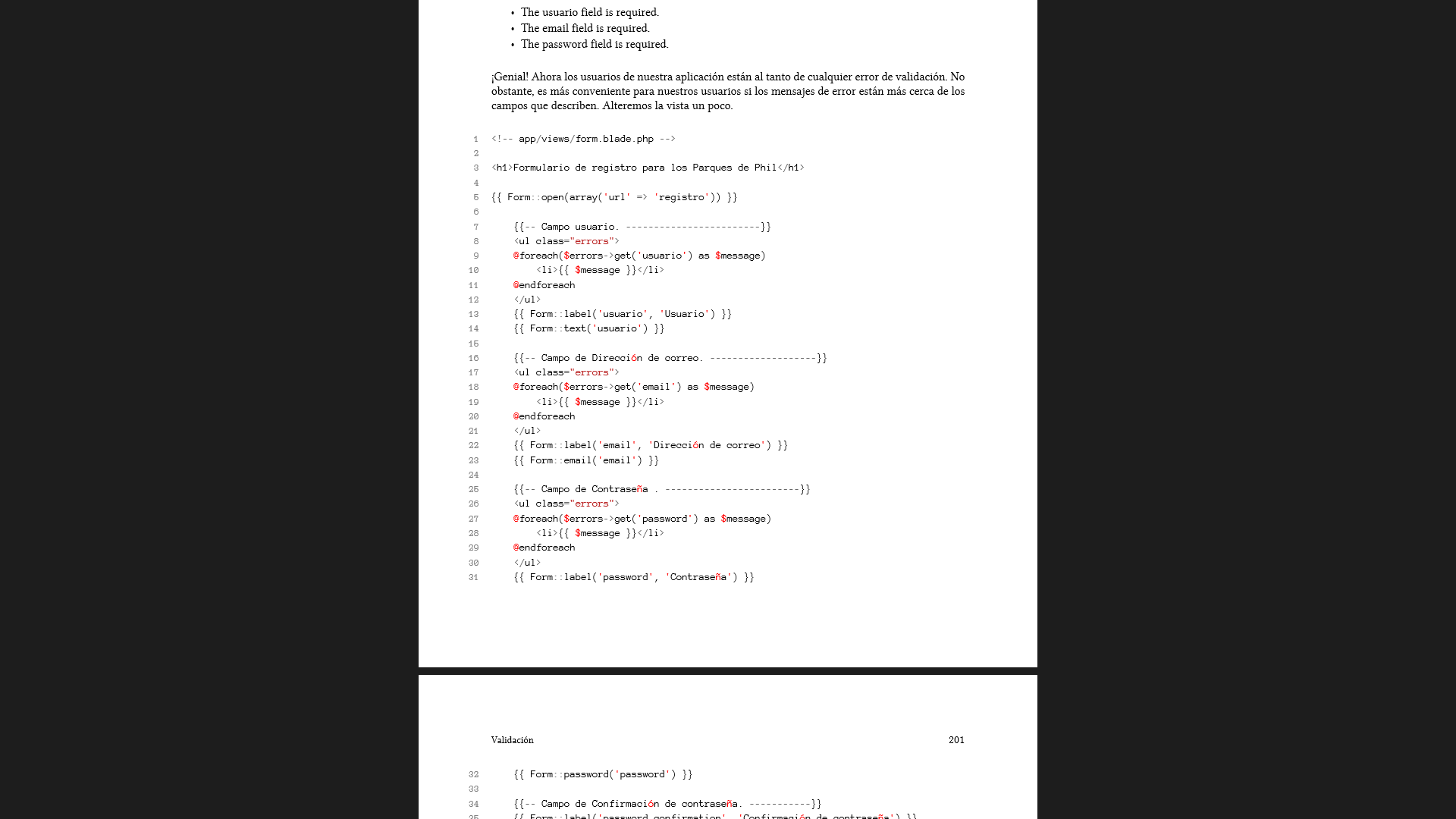
**CONTROL DE ERRORES FORMULARIOS**



**Mensajes de error arriba del formulario:**



**Mensajes de error en el mismo input con la variable get**



P.D: Se puede poner <span> </span> entre los errores.

**BASE DE DATOS**

--“formularioregistro.php”—

Public function blablá(){

$con = DB::connection(“nombrebbdd”);

$sql = “INSERT INTO usuarios(id,nombre,pass) VALUES(?,?,?)”;

$con->insert($sql, array(‘1’,’manolo’,’12345’))

Return View::make(‘Index.php’);

}

$selectusuarios = $con->select

$con->update

$con->delete

@foreach ($selectusuarios as $usuarios)

Usuario: {{$usuarios->nombre}}

Pass: {{$usuarios->pass}}

@endforeach