Laravel

Curso práctico avanzado

José López Quijado

libros R

alphaeditorial

LARAVEL

Curso práctico avanzado

LARAVEL

Curso práctico avanzado

José López Quijado

ALPHA EDITORIAL S.A.
Calle 62 20-46 esquina, Bogotá
Teléfono (57-601) 746 0102
cliente@alpha-editorial.com
www.alpha-editorial.com

LIBROS DIGITALES
wwww.alphaeditorialcloud.com

RC LIBROS
Calle Mar Mediterráneo, 2. N-6
28830 San Fernando de Henares, Madrid
Teléfono +34 91 677 57 22
info@rclibros.es
www.rclibros.es

Primera edición: Madrid, 2021
Bogotá, 2022

Diseño de colección y pre-impresión: Grupo RC
Diseño de cubierta: Cuadratín

ISBN: 978-958-778-738-2 (Colombia)
ISBN: 978-958-778-739-9 DIGITAL
ISBN: 978-84-121069-5-4 (España)

Impreso en Colombia
Printed in Colombia

ÍNDICE

INTRODUCCIÓN

Este libro, Laravel: Curso práctico avanzado, contiene recursos y técnicas que no están incluidos en el volumen básico anteriormente publicado, *Laravel: Aprende a crear aplicaciones web desde cero*, como gestión de usuarios (tanto en Laravel 5 como en la versión 6), integración y uso de distintos paquetes de mejoras, y otras técnicas que nos llevarán a otro nivel.

El volumen básico es una iniciación. Contiene información muy detallada sobre sobre las técnicas que se describen en él y, de hecho, se referencia mucho en este segundo volumen. La información que en el volumen básico se ofrece al lector no es fácil de encontrar, recopilar y estructurar, por lo que es una ayuda inestimable para quienes buscar iniciarse en Laravel.

Sin embargo, con este volumen llegamos a explorar las posibilidades y prestaciones de este framework (que, por otro lado, no dejan de crecer) a un nivel que pocos desarrolladores alcanzan. Ahora ya jugamos en las Ligas Mayores, y nos estamos preparando para ser, realmente, Laravel Developers. Sin embargo, que no espere sacar todo el partido posible a este libro quien no haya leído previamente el volumen básico. No se puede construir un buen edificio sin buenos cimientos, por extraordinarias que sean las herramientas de que dispongamos (y Laravel es, no le quepa duda, una herramienta realmente excepcional).

Espero que usted, lector, disfrute de este volumen, y llegue a explotar al máximo los recursos que aquí se ofrecen. Si ya ha leído el volumen básico, lea este libro con calma, sin apresuramientos y, sobre todo, pruebe (y analice detalladamente) los ejercicios que se ofrecen en el material complementario. No se limite a “ver que funcionan”. Aprenda cómo funcionan; por tanto, si es usted comprador del mismo podrá acceder gratuitamente a ellos, visitando: **http://rclibros.es**, y después la

página individual del libro. Tal vez se dé cuenta de que aprende más de los ejemplos que del propio texto.

Y no olvide que los desarrolladores somos inquietos y creativos por naturaleza. Quizá a usted, viendo los códigos, se le ocurra, para una determinada situación, alguna solución incluso mejor que la que yo le ofrezco. Eso sería estupendo, porque le diría que su intuición natural para el desarrollo web está despierta.

Adelante, y bienvenido.

RECONOCIMIENTOS

Quiero agradecer a Laravel por la magnífica herramienta que nos proporciona, y el esfuerzo realizado para que la versión 5.8 sea, sin duda, uno de los mejores frameworks PHP del mercado. Además, es de agradecer el esfuerzo que ha hecho para presentar una documentación tan detallada, amplia y bien estructurada.

Asimismo, quiero valorar positivamente la inestimable ayuda que siempre ha representado el foro Stack Overflow (**https://es.stackoverflow.com/** y **https://stackoverflow.com/**), tanto para mí, como para mis compañeros de trabajo.

También quiero reconocer a Google, por sus magníficas aportaciones al mundo del desarrollo web, en especial (aunque no únicamente), por su buscador, por Angular 6 (**https://angular.io/**) y por su magnífica aportación en lo que a CSS se refiere, con Material Design (**https://material.io/design/**).

Por último, quiero destacar la importancia que tienen, en el desarrollo web actual, las herramientas Bootstrap (**https://getbootstrap.com/**), jQuery (**https://jquery.com/**), Font Awesome (**https://fontawesome.com/**) y Github (**https://github.com/**). Por supuesto, no son las únicas ayudas con las que contamos. La lista de recursos técnicos e informativos en Internet es interminable, y yo he hecho un profundo uso de muchos más en estos libros. Sin embargo, referirlos aquí todos sería imposible, así que solo he reseñado los más destacados.

EL AUTOR

José López Quijado es un desarrollador vocacional que lleva coqueteando con el mundo de la informática desde mediados de los años 80 del pasado siglo, y que se enfrentó con la falta de documentación, ya que la poca que entonces circulaba era en lengua inglesa, rudimentaria y escasa. Sus primeros ordenadores personales se los confeccionó él mismo comprando las piezas, y ensamblándolos en su propia casa; las

largas horas transcurridas conectando dispositivos a una placa base, atornillando todo en una caja de las que llamaban XT y AT, y buscando en bibliotecas y grupos de estudio toda la información necesaria para que sus pequeños modelos cobraran vida, le han permitido crecer profesionalmente.

Todavía faltaban varios años para el advenimiento de Internet, los ordenadores eran enormes aparatos, toscos, con grandes consumos de energía, y unos precios prohibitivos para la mayoría de los mortales, sin embargo, él no encuentra barreras y descubre su Santo Grial, entre placas base de más de medio metro de lado, y procesadores de 4 y, posteriormente, 8 bits; en aquella época el Saturno H48, la serie Z de Zilog y la serie 6800 de Motorola.

Con la llegada de los 90 y el comienzo de la popularidad de la World Wide Web, continúa su proceso de aprendizaje con Internet tanto con el desarrollo de los toscos documentos HTML, las primeras bases de datos como dBase II, III y IV, explorando al máximo cada tecnología, cliente servidor, o cualquiera de las que considera que mantienen posibilidades de las nuevas tecnologías y que le permitan dar rienda suelta a su creatividad.

Actualmente es el responsable de desarrollo en el Grupo Quirón, donde se ocupa de desplegar la operativa de grandes cuentas en los sectores financiero, industrial, alimentario, inmobiliario y de otros clientes; también mantiene dos blogs sobre temas técnicos (**https.eldesvandejose.com** y **https://httpmasters.es**) donde escribe sobre aquellas tecnologías que más le gustan, procurando mantenerse actualizado y evolucionar con todo lo que significa el desarrollo de la Red de Redes.

PARTE. I
USUARIOS

1

AUTENTICACIÓN EN LARAVEL

Cualquier aplicación web actual debe contar con un sistema de acceso para los usuarios, de modo que puedan existir partes de dicha aplicación a las que pueda acceder un visitante anónimo, y otras cuyo acceso esté reservado a los usuarios registrados y convenientemente logueados.

Además, en este último apartado, consideraremos la posibilidad de que no todos los usuarios logueados puedan acceder a todas las partes de la aplicación. Un ejemplo clásico es una aplicación en la que hay una zona en la que se diseñan, redactan y colocan los contenidos, o se configura la aplicación. A estas funcionalidades solo podrán acceder aquellos usuarios que tengan permisos específicos. Luego hay otra parte, para el público, en la que se pueden consumir los contenidos de la aplicación. A esta parte podría acceder, en principio, cualquier usuario registrado, aunque no tenga permisos de administrador.

LA AUTENTICACIÓN EN LARAVEL

Los creadores de Laravel entendieron muy bien el problema desde el principio. El registro y gestión de usuarios y administradores es, siempre, en cualquier aplicación, una tarea inherentemente compleja y limitativa. Sin embargo, en Laravel, es un

sistema que se puede montar con mucha facilidad y rapidez, que requiere muy poca codificación, y que, no obstante, ofrece gran potencia y flexibilidad. Laravel implementa un sistema de gestión de usuarios fácil de aprender a manejar, y que nos permitirá llevar a cabo, con un trabajo mínimo, todo lo que podamos necesitar.

PRESTACIONES DE AUTENTICACIÓN

El sistema de autenticación de Laravel incluye, entre otras, las siguientes prestaciones:

- Autenticación HTTP Basic. Es el sistema más simple y rudimentario de autenticación. Muy de moda en los primeros tiempos de la web, hoy ha caído en desuso por las limitaciones, tanto estéticas como funcionales, que impone. Lo veremos en este capítulo (porque hay que conocerlo, aunque solo sea para no usarlo), y luego no volveremos a hablar de este sistema nunca más.
- Creación de sistema de autenticación mediante Artisan. Esta potente herramienta nos permite, con un solo comando (**`make:auth`**), crear un completo y eficiente sistema de autenticación en nuestra aplicación que, además, podremos personalizar según necesitemos.
- Adición de campos personalizados a los usuarios, lo que nos permitirá activarlos o desactivarlos, asignarles roles o conjuntos de permisos, etc.
- Enviar emails personalizados a los nuevos usuarios, cuando creen su cuenta, para que la confirmen. El nuevo usuario no podrá acceder hasta confirmar la cuenta. Como alternativa, podremos crear autenticación sin contraseña. El nuevo usuario recibe un correo y, al confirmar la cuenta, queda logueado para establecer la contraseña.
- Establecer un sistema de reseteo de contraseñas, por ejemplo ante olvidos de la misma.
- Personificar (en la práctica, suplantar) a otros usuarios. Esta es una funcionalidad que permite a un usuario hacerse pasar por otro. Se le atribuye esta potestad a los administradores de la aplicación con la finalidad de que accedan simulando ser un usuario, y viendo la web tal como la ve el usuario personificado. Se usa, por ejemplo, si un usuario manifiesta tener alguna incidencia.
- Autenticación con redes sociales, como Google, Facebook o Twitter.

Para disponer de un sistema adecuado de autenticación, necesitaremos aprender algunas cosas sobre middlewares, sistemas específicos de enrutamiento, guards, correos electrónicos, sesiones, tokens y otras prestaciones. Evidentemente no

entraremos en todas estas materias a fondo, sino que las conoceremos en aquellos aspectos en los que nos sean necesarias para la autenticación. A lo largo de los próximos capítulos iremos viendo todo lo necesario.

EL USUARIO POR DEFECTO

Cuando se crea un nuevo proyecto Laravel, este ya viene preparado, por defecto, para contar con un usuario. Esto no es nada nuevo. En cada proyecto que hemos creado hemos eliminado el modelo **`User.php`**, así como las migrations y la factory. Además, podríamos haber eliminado, dentro de la ruta **`app/Http/Controllers`**, la carpeta **`Auth`** y todo su contenido. Estos son controladores específicos para la autenticación. En este caso vamos a crear un proyecto llamado **`usuarios_01`**, en el que no vamos a eliminar nada de todo esto, ya que lo vamos a usar en esta parte del libro.

ATENCIÓN. Durante los proyectos que hemos hecho en el volumen de Laravel básico, también modificábamos la ruta de la vista principal, para que accediera a esta a través de un controlador. Aquí, al menos por ahora, no vamos a tocar eso.

PREPARANDO EL PROYECTO

Ya que la mayoría de las cosas que veníamos haciendo antes no las haremos esta vez (de momento) poco es lo que nos queda por preparar. Dentro del archivo **`.env`** estableceremos el nombre de la aplicación, la ruta base y el motor de base de datos que emplearemos:

```
APP_NAME=usuarios_01
APP_URL=http://usuarios_01.lar
DB_CONNECTION=sqlite
```

En el directorio **`database`** crearemos el archivo **`database.sqlite`**.

Ahora tenemos que buscar y abrir el archivo que se encuentra en la ubicación **`app/Providers/AppServiceProvider.php`**. Debemos buscar la siguiente línea:

```
use Illuminate\Support\ServiceProvider;
```

Debajo de ella agregaremos la siguiente:

```
use Illuminate\Support\Facades\Schema;
```

En el cuerpo del método `boot()` agregaremos otra línea, quedando así:

```
public function boot()
{
Schema::defaultStringLength(170);
}
```

Esto es un mecanismo que, en realidad, casi podría haberse creado por defecto. Sirve para evitar un error de claves demasiado largas. Vamos a modificar el archivo `database/seed/DatabaseSeeder.php`, para que ejecute la `UserFactory`, creando un usuario de prueba. Lo dejamos así:

```
<?php

use App\User;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        factory(User::class, 1)->create();
    }
}
```

Aunque hace tiempo que no usamos las factories y el seeder, se ve claramente lo que ocurre. A continuación, desde la consola, ejecutamos las migrations y el seeder. Podemos hacerlo en una sola instrucción, así:

```
php artisan migrate --seed
```

La factory nos crea un usuario en la tabla `users`. La tabla `password_resets` (cuyo nombre es bastante descriptivo de lo que almacenará), de momento queda vacía. La estructura básica ya la tenemos. Si abrimos la base de datos (con SQLite Studio, por ejemplo) verá el usuario que se acaba de crear. Ve su nombre, su email y también una fecha en el campo `email_verified_at`, como si el usuario se hubiera registrado realmente en la aplicación, y hubiera confirmado su cuenta por correo eletrónico. Más adelante hablaremos de este tema. Por ahora, nos basta con haber creado estos datos con la factory. Además, tenemos una clave, que es el encriptado de `password`, que es la contraseña que la factory le ha asignado al usuario.

También vemos un campo llamado `remember_token`, del que hablaremos más adelante. Por ahora, basta saber que es un token único e irrepetible.

ACERCA DE LA ENCRIPTACIÓN

Laravel utiliza, para la encriptación de contraseñas, un algoritmo irreversible conocido como **bcrypt**. Si es usted desarrollador de PHP, sin duda lo ha usado en gran cantidad de ocasiones. Si no es el caso, puede leer al respecto, a título de curiosidad, en **https://eldesvandejose.com/2016/07/14/seguridad-de-acceso-en-php-ii-contrasenas-no-reversibles/**.

En la factory de ejemplo que se monta cuando se crea un nuevo proyecto encontramos que se le asigna al usuario una contraseña ya encriptada. Esta es la encriptación de la palabra `password` (en Laravel 5.8) o la palabra `secret` (en Laravel 5.7). En la factory encontramos algo similar a esto:

```
'password' =>
'$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', //
password
```

El mecanismo funciona. Sin embargo, esto se puede hacer de un modo más limpio y elegante, sustituyendo la línea anterior por:

```
'password' => bcrypt('password'),
```

El resultado es el mismo: encripta la palabra `password` (o la que sea que decidamos), y almacena la contraseña encriptada en la tabla de usuarios. El helper `bcrypt()` se usa, precisamente, con esa finalidad y, al ser un helper nativo de Laravel, está disponible en toda la aplicación.

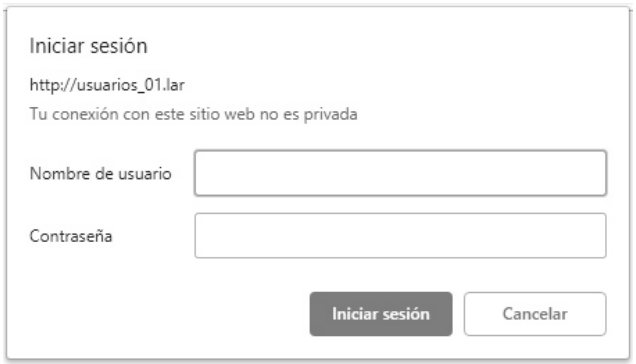
Como ya sabe (y si no se lo cuento brevemente ahora) la encriptación mediante el algoritmo bcrypt no es constante: si encripta dos veces la misma contraseña va a obtener dos cadenas completamente diferentes. Esto es un mecanismo de seguridad de PHP que se emplea para que la encriptación no pueda ser revertida de ningún modo mediante ingeniería inversa. Al hacer la encriptación, PHP ya usa una semilla que luego permite la verificación de la contraseña, pero no su reversión. Es un sistema internamente bastante complejo, pero muy seguro y fiable.

LA AUTENTICACIÓN HTTP BÁSICA

Este es, como hemos dicho, el sistema más primitivo y rudimentario de autenticación de un usuario. Consiste en un mecanismo que pide el identificador y la contraseña del usuario, y le da acceso a la web directamente.

Tiene varios inconvenientes. Por un lado, no es muy estético. Por otro lado, y esto sí es un problema, el usuario ya queda autenticado de modo indefinido. No provee un mecanismo de cierre de sesión. Si accede así a una web, el único modo de cerrar sesión es eliminando manualmente las cookies de su navegador.

DESCRIPCIÓN



Lo habrá visto en páginas muy antiguas o, si es usted demasiado joven, en vídeos vintage de Internet. Cuando intenta acceder a la página, en la parte superior se le muestra algo parecido a la imagen de la izquierda.

Si pulsa el botón de inicio de sesión sin introducir los datos correctos en los campos, se le vuelve a mostrar la ventana que le pide los datos de acceso. Si pulsa el botón de cancelación, no le da acceso a la web. En su lugar le muestra un mensaje de error de tipo `401`, que le indica que no está autorizado a acceder.

Sin embargo, si introduce los datos de un usuario que conste en la base de datos del sistema, le da paso a la web. Usted tiene un usuario en la tabla `users`, que hemos creado en el apartado anterior. En el campo **Nombre de usuario** teclee el correo electrónico con el que se generó el usuario. En el campo **Contraseña**, teclee `password`, que es la clave que, encriptada, tiene en el usuario de su tabla. Pulse el botón de *Iniciar sesión* y la página se carga libremente.

LA PREPARACIÓN

Como ve, el sistema funciona, para ilustrar el uso de contraseñas (aunque sea tosco y poco flexible). Sin embargo, tenemos que preparar nuestra aplicación para que, al intentar acceder, pida las credenciales. Si no, será como cualquier otro proyecto nuevo, que da acceso libremente sin credenciales, como hemos ido viendo en el volumen de Laravel básico.

Lo primero que tenemos que hacer es ver cómo llama Laravel a este tipo de autenticación. Para ello vamos al archivo `app/Http/Kernel.php` (no se confunda de ruta: hay otros achivos con el nombre `Kernel.php` en Laravel, con otros usos). Dentro de este archivo vamos a buscar una matriz llamada `$routeMiddleware`. El

nombrecito ya nos indica, de algún modo, lo que tiene esta matriz: una lista de los distintos middlewares que se pueden aplicar a los enrutamientos, para filtrar las llamadas y modificar, así, el comportamiento de las rutas. La matriz tiene, más o menos, este aspecto:

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' =>
\Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' =>
\Illuminate\Routing\Middleware\SubstituteBindings::class,
    'cache.headers' =>
\Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'signed' =>
\Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' =>
\Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' =>
\Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
];
```

Aquí nos interesa el middleware que se ocupa de implementar una autenticación básica, que es el que aparece en la siguiente línea:

```
'auth.basic' =>
\Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
```

Observe la clave que tiene: **`auth.basic`**. Pues ahora nos vamos al enrutador y buscamos la ruta que queremos proteger mediante este middleware. Como el proyecto está recién creado, solo tenemos una ruta, que es la que "viene" por defecto con cualquier proyecto nuevo de Laravel. Lo vemos en **`routes/web.php`**:

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

La modificamos encadenándole el método **middleware()** de Laravel. Este no lo habíamos usado hasta ahora. Se emplea, como vemos en este ejemplo, para que las peticiones a una ruta, antes de pasar a su destino, sean filtradas por un middleware específico. Observe cómo queda:

```
Route::get('/', function () {
    return view('welcome');
})
->middleware('auth.basic');
```

Hemos pasado, como argumento, la clave del middleware **AuthenticateWithBasicAuth**, tal como figuraba en la matriz de middlewares de rutas.

En Laravel hay middlewares para rutas, para grupos, y… bueno, para casi todo lo que podamos necesitar, y se pueden llamar donde se necesiten en cada caso (en una ruta, en un controlador, o donde proceda). Además, si lo necesitamos, podremos escribir nuestros propios middlewares, aunque este no es ahora el caso. Hablaremos de ese tema más adelante.

Lo que importa es que, al haber añadido el método **middleware()** a nuestra ruta, con la referencia de un middleware de Laravel, las peticiones a esa ruta pasan, sí o sí, por el middleware especificado. Eso permite que la autenticación se dispare cuando llamamos a la ruta principal.

Podemos comprobar que el middleware está filtrando las peticiones a esa ruta, con el comando, ya bien conocido, **php artisan route:list**. Veremos un resultado como este:

```
+--------+----------+----------+------+---------+----------------+
| Domain | Method   | URI      | Name | Action  | Middleware     |
+--------+----------+----------+------+---------+----------------+
|        | GET|HEAD | /        |      | Closure | web,auth.basic |
|        | GET|HEAD | api/user |      | Closure | api,auth:api   |
+--------+----------+----------+------+---------+----------------+
```

Como ve, en la ruta principal (/) aparece ahora, aparte del middleware por defecto (**web**), el que hemos incluido (**auth.basic**).

DESCONECTANDO

Como ya hemos comentado, uno de los problemas de este sistema de autenticación es que no puede cerrar sesión. Tiene que eliminar las cookies de su navegador. Si no, cuando vuelva a entrar, como el navegador ya le tiene autenticado, le da paso directamente, como si no hubiera autenticación alguna.

En capítulos posteriores vamos a ir viendo sistemas de acceso mucho más depurados y eficientes.

EL MODELO User

Cuando se crea un proyecto nuevo de Laravel, este incluye, como ya sabe, por defecto, la definición del modelo **`User`**, en el archivo **`app/User.php`**. A priori es, *casi*, como cualquier modelo de cualquier tabla que hayamos usado hasta ahora en los distintos proyectos de pruebas. En realidad, es un esqueleto muy básico del modelo para los usuarios y, según vayamos aprendiendo, deberemos ir agregando campos, relaciones, etc. No obstante, vamos a echarle un vistazo rápido, aunque solo sea por familiarizarnos con él, y repasar algún concepto básico y, de paso, comentar un poco ese "casi".

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password',
    ];
```

```
    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];
    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];
}
```

Lo primero que nos llama la atención es que se hacen unas importaciones que, en otros modelos no se hacen. Estas son interfaces y traits del propio Laravel, diseñados específicamente para la operativa con usuarios:

```
use Illuminate\Notifications\Notifiable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
```

ATENCIÓN. Si está leyendo este libro y ha llegado hasta aquí, le supongo más que familiarizado con la mayoría de los conceptos de PHP. No obstante, si lo considera adecuado, puede leer sobre interfaces y traits en las siguientes páginas de la documentación:http://php.net/manual/es/language.oop5.interfaces.php y en http://php.net/manual/es/language.oop5.traits.php.

Lo siguiente que nos llama la atención es que este modelo no hereda de la clase `Model`, como todos los que construimos nosotros. Un modelo para usuarios hereda de la clase `Authenticatable`, que implementa funcionalidades específicas que otros modelos no usan ni necesitan, como ve a continuación:

```
class User extends Authenticatable
```

Además, la clase usa el trait **Notifiable**, que servirá para gestionar ciertas notificaciones inherentes a los objetos de este modelo:

```
use Notifiable;
```

Dentro de la clase encontramos ya algunas cosas que sí nos son familiares. Una de ellas es la matriz **$fillable**, que ya conocemos:

```
protected $fillable = [
    'name', 'email', 'password',
];
```

Como ve, por defecto se definen los campos básicos de la estructura de los objetos **User**, pero podremos modificarla más adelante, cuando nos haga falta.

A continuación vemos la matriz **$hidden**:

```
protected $hidden = [
    'password', 'remember_token',
];
```

Esta contiene, como sabe, campos que no deben mostrarse, por defecto, en un listado. En este modelo, Laravel incluye, en esta categoría, por defecto, los campos **password** y **remember_token**. Como tenemos un usuario en la base de datos, podemos ver cómo opera esta matriz. En la consola de mandatos, dentro del directorio de su proyecto, entre en tinker:

```
php artisan tinker
```

Ahora teclee:

```
App\User::all()
```

Verá que le muestra todos los datos del usuario que tiene (le mostraría más si los tuviera), excepto los campos que están incluidos en la matriz **$hidden**.

Por último, tenemos la matriz **$casts**:

```
protected $casts = [
    'email_verified_at' => 'datetime',
];
```

Esto es algo que, hasta ahora, no habíamos empleado en ningún modelo. Esta matriz define campos en los que el contenido que se grabe se debe forzar a un tipo específico de dato.

2

AUTENTICACIÓN COMPLETA

En el capítulo anterior hemos visto ideas elementales sobre la autenticación y los usuarios, y también cómo esta es gestionada mediante middlewares, entre otras cosas. Conocimos la antigua autenticación HTTP-Basic (hoy en desuso) para ver cómo regular la petición del acceso a una ruta mediante el método **`middleware()`**, y un poco por encima ciertos conceptos básicos.

En este capítulo ya vamos a ver un sistema de autenticación completo (bueno, bastante completo), para aprender cómo Laravel nos ayuda en una tarea que parece muy simple, pero que es de crucial importancia en cualquier aplicación web.

Para no cambiar nada del proyecto anterior, vamos a crear uno nuevo, llamado **`usuarios_02`**. Vamos a empezar modificando la factory **`UserFactory.php`**, sustituyendo la línea que define la clave por defecto por la siguiente:

```
'password' => bcrypt('clave'),
```

En el método **`run()`** del seeder teclearemos:

```
factory(User::class, 1)->create();
```