Introducción a Laravel

Preparando el entorno: Composer y servidor web

Para instalar Laravel, primero debemos instalar Composer (https://getcomposer.org/) en la máquina. Posteriormente, Composer se utiliza para instalar todos los paquetes de Laravel.

A continuación, necesitamos tener un servidor web PHP en la máquina. Hay diferentes herramientas para diferentes sistemas operativos que incluyen Nginx/Apache, PHP, MySQL y algunas cosas más para proyectos web PHP:

- Laravel Herd (MacOS)
- Laravel Valet (MacOS/Linux)
- Laragon/XAMPP (Windows)
- etc.

Nota: Laravel 11 requiere al menos la versión PHP 8.2.

Instalación en Visual Studio Code

Para instalar Laravel desde Visual Studio Code utilizando Composer, sigue estos pasos:

- Asegúrate de tener Composer instalado en tu sistema.
- Abre Visual Studio Code y accede al terminal integrado
- En el terminal, navega hasta el directorio donde deseas crear tu proyecto Laravel.
- Ejecuta el siguiente comando para crear un nuevo proyecto Laravel usando Composer:

```
composer create-project laravel/laravel nombre-de-tu-proyecto
```

- Reemplaza "nombre-de-tu-proyecto" con el nombre que desees para tu aplicación.
- Una vez finalizada la instalación, navega al directorio del proyecto:

```
cd nombre-de-tu-proyecto
```

- Abre la carpeta del proyecto en Visual Studio Code seleccionando File > Open Folder y elige la carpeta de tu proyecto Laravel.
- Para iniciar el servidor de desarrollo de Laravel, ejecuta en el terminal:

```
php artisan serve
```

- Esto iniciará tu aplicación Laravel en http://localhost:8000

Adicionalmente, puedes instalar extensiones útiles para Laravel en Visual Studio Code, como "Laravel Artisan", etc, buscándolas en la sección de extensiones e instalándolas

Si abres el proyecto en el editor de código y específicamente el archivo route/web.php, verás que hay una ruta definida.

Fichero routes/web.php:

```
<?php

use Illuminate\Support\Facades\Route;

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

Aquí tenemos una página de inicio /, que conduce a una vista (view) llamada welcome. Las vistas se encuentran en la carpeta “resources/views”.

Eso es todo, ya tenemos instalado Laravel.

Enrutamiento básico: URL y página de inicio predeterminada

Ahora debemos entender cómo crear nuevas páginas con Laravel. Echemos un vistazo nuevamente a cómo se define la página de inicio predeterminada:

Fichero routes/web.php:

```
<?php

use Illuminate\Support\Facades\Route;

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

Para definir una ruta con un URL, debemos hacer ***Route::get()***.

- El primer argumento es la URL: en este ejemplo, es /
- El segundo argumento es una función de callback que devuelve una vista (***view***).

View es un archivo de plantilla escrito en un lenguaje específico de Laravel llamado Blade. Podemos considerar un archivo Blade como un archivo HTML con variables en su interior.

La parte ***welcome*** en el código significa que el archivo está en una carpeta llamada ***resources/views*** con la extensión .blade.php. Esta extensión es crucial.

En ese archivo ***welcome.blade.php***, tenemos HTML con alguna sintaxis Blade como @if o @auth.

Nota: Si cambiamos algo en este fichero y recargamos se ven los cambios en el navegador.

Ahora vamos a añadir una nueva ruta.

Fichero routes/web.php:

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

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

Esto hará que esté disponible un enlace ***localhost:8000/second***.

La nmueva ruta devolverá una vista llamada ***second.blade.php***. Para hacer una prueba, copia el contenido del archivo ***welcome.blade.php*** a ***second.blade.php*** y modifica algo de texto.

Si, tras hacer la modificación, navegamos al url localhost:8000/second debemos ver la nueva página y si intentamos visitar una ruta que no existe, se mostrará un mensaje 404 estándar.

De esta forma que se ha descrito, se crean rutas y páginas en Laravel con lenguaje Blade, por supuesto, las páginas tendrán más lógica en proyectos reales.

Si tenemos páginas estáticas similares con solo un archivo Blade y sin mucha lógica, podemos usar una sintaxis más corta en el archivo de rutas: en lugar de ***Route::get()*** y una función de callback, podemos usar ***Route::view( )*** y proporcione el nombre del archivo Blade como segundo parámetro. En el fichero de rutas podríamos haber escrito:

```
...
Route::view('/third', 'third');
```

Blade con recursos CSS

En este punto, vamos a reemplazar la página de inicio predeterminada de Laravel con algún diseño. Para el diseño, utilizaremos una plantilla de blog Bootstrap 5 sencilla.

La página de bienvenida predeterminada de Laravel usa Tailwind, pero para este punto, usaremos un ejemplo de Bootstrap para centrarnos solo en la estructura de Laravel, sin pensar en cómo compilar clases de Tailwind con NPM.

Primero, preparemos nuestro proyecto Laravel para el nuevo archivo Home Blade.

Creamos un nuevo archivo de vista home.blade.php dentro de la carpeta recursos/vistas. Cambie la Ruta de la página de inicio para usar este archivo Ver en los archivos de Rutas.

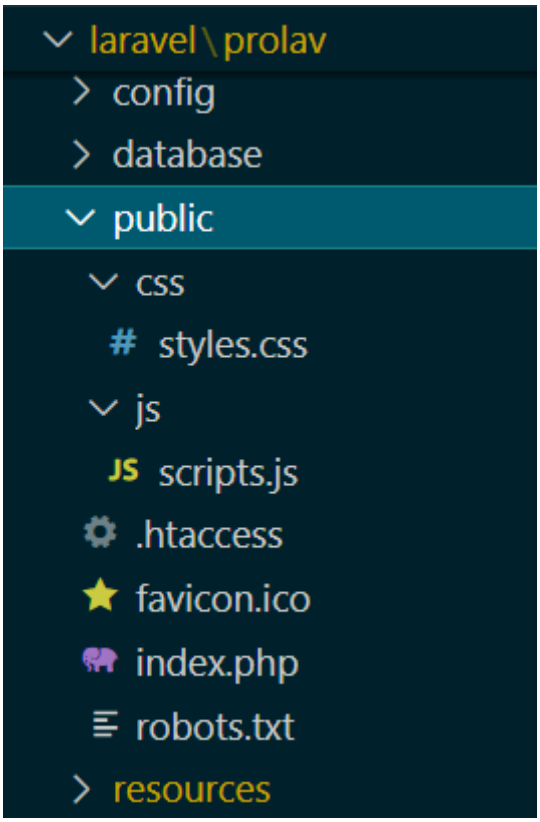
El fichero web.php debe quedar:

```
<?php

use Illuminate\Support\Facades\Route;

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

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

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

Route::view('/third', 'third');
```

A continuación, descargamos la plantilla Bootstrap (https://startbootstrap.com/template/blog-home) y la extraemos en nuestra máquina. Luego, abrimos el fichero index.html en un editor y copiamos todo su contenido en ***resources/views/home.blade.php***.

Después de actualizar la página de inicio, debería ver el texto, pero sin estilos CSS.

Ahora, necesitamos darle estilo a esta página. Cuando extrajiste la plantilla Bootstrap, había dos carpetas dentro de css y js. Primero, debemos copiar estas dos carpetas a la carpeta ***public*** de nuestro proyecto.



Ahora debemos decirle a Laravel cómo acceder a estos activos. Para esto, usaremos una función auxiliar ***asset()***. Esta función auxiliar devolverá una ruta completa al activo, incluido el dominio.

Para usar la función auxiliar ***asset()*** dentro de un archivo Blade, debemos usar la sintaxis de Blade ***{{ $variable }}***, que simplemente refleja el resultado. En nuestro caso, esa $variable será el resultado del método ***asset()***:

Fichero resources/views/home.blade.php:

En el bloque de la cabecera hay que añadir :

<link href="{{ asset('css/styles.css') }}" rel="stylesheet" />

y en el de pie (footer):

<script src="{{ asset('js/scripts.js') }}"></script>

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
        <meta name="description" content="" />
        <meta name="author" content="" />
        <title>Blog Home - Start Bootstrap Template</title>
        <!-- Favicon-->
        <link rel="icon" type="image/x-icon" href="assets/favicon.ico" />
        <!-- Core theme CSS (includes Bootstrap)-->
        <!-- Esto ya no sirve si usamos BLADE-->
        <!-- <link href="css/styles.css" rel="stylesheet" /> -->
        <link href="{{ asset('css/styles.css') }}" rel="stylesheet" />
    </head>
```

```
        <!-- Footer-->
        <footer class="py-5 bg-dark">
            <div class="container"><p class="m-0 text-center text-white">Copyright &copy;
        </footer>
        <!-- Bootstrap core JS-->
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle
        <!-- Core theme JS-->
        <!-- Esto ya no sirve en BLADE -->
        <!-- <script src="js/scripts.js"></script> -->
        <script src="{{ asset('js/scripts.js') }}"></script>
    </body>
</html>
```

Si recargamos depués de hacer estas modificaciones la página se muestra con sus estilos.

Por supuesto, esta plantilla es sólo una página estática, por ahora. Pero con este ejemplo, deberías poder entender lo principal: que los activos CSS/JS se colocan en la carpeta /public y se hace referencia a ellos con la función auxiliar asset().

Para Tailwind y proyectos full-stack más modernos, los activos en realidad se colocan primero en las carpetas resources/js y resources/css y luego se compilan automáticamente en la carpeta /public, con comandos como npm run dev o npm run build.

Menú principal, estructura de BLADE y páginas estáticas

(todo)

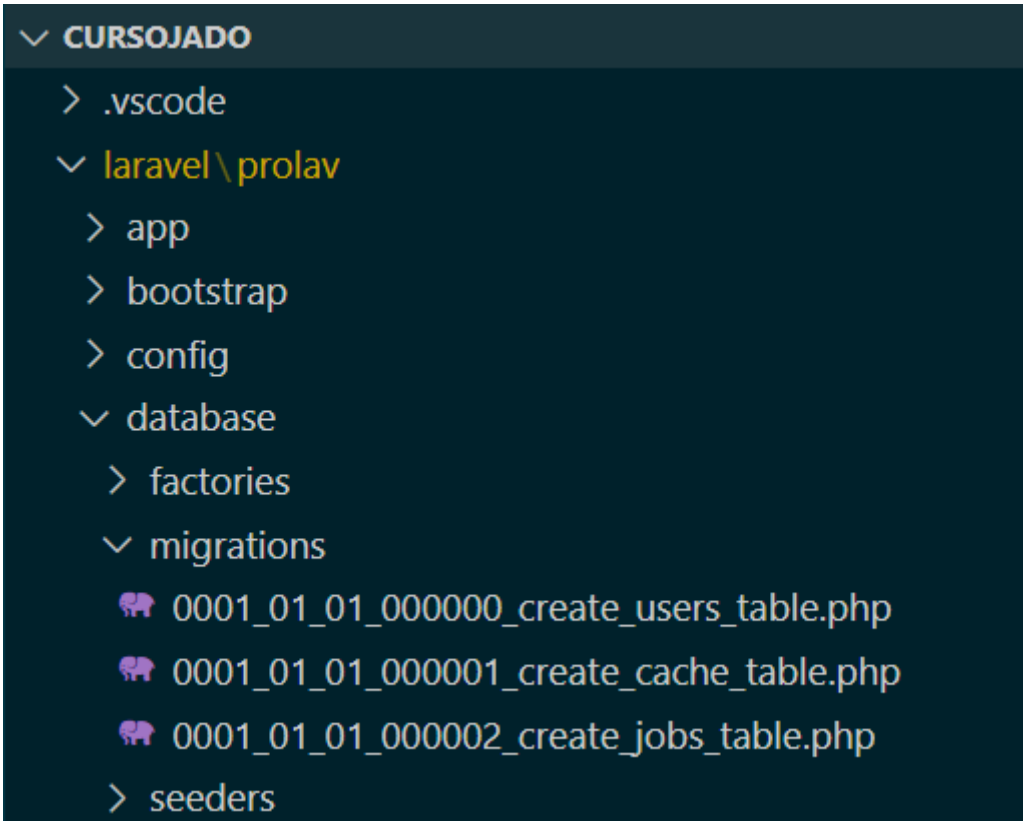
Estructura de base de datos, Migrations y configuración .env

Hasta ahora hemos construido páginas estáticas. Ahora, hablaremos del backend, la estructura de la base de datos y los datos, y cómo Laravel maneja eso.

No es necesario crear tablas de base de datos con columnas manualmente. Para eso, Laravel utiliza un concepto llamado migraciones (***migrations***). Describe su esquema de migración en un archivo de migración y luego ejecuta las migraciones, creando tablas con columnas.

Las migraciones se guardan en la carpeta ***database/migrations***. Cuando creas un nuevo proyecto Laravel, hay tres archivos de migración. El único archivo de migración importante al iniciarnos en Laravel es el primero donde se crea la tabla Usuarios (users). Normalmente, un archivo de migración crearía una tabla. El framework crea tres tablas en un archivo de migración, todas correspondientes a los usuarios.

El



contenido del fichero ***database/migrations/0001_01_01_000000_create_users_table.php*** es:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
```

```
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });

        Schema::create('password_reset_tokens', function (Blueprint $table) {
            $table->string('email')->primary();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });

        Schema::create('sessions', function (Blueprint $table) {
            $table->string('id')->primary();
            $table->foreignId('user_id')->nullable()->index();
            $table->string('ip_address', 45)->nullable();
            $table->text('user_agent')->nullable();
            $table->longText('payload');
            $table->integer('last_activity')->index();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('users');
        Schema::dropIfExists('password_reset_tokens');
        Schema::dropIfExists('sessions');
    }
};
```

Si vemos el contenido del fichero, es bastante legible, incluso si no sabes cómo funcionan las migraciones.
Primero, se define un nombre de tabla y todas las columnas dentro de unas llaves de cierre.

Vemos qué campos hay dentro de la tabla de usuarios:

- ***id,*** campo de identificación, que es clave principal y se incrementa automáticamente
- Strings para los campos de nombre (name) y correo electrónico (email). Además, el correo electrónico es único, por lo que no se puede repetir.
- ***rememberToken()*** es un campo específico de Laravel para marcar la casilla "Recordarme".
- Las marcas de ***timestamps()*** son un atajo para dos campos: ***created_at*** y ***updated_at***. Se completan automáticamente. Eso lo veremos más tarde.

Ahora bien, ¿cómo ejecutar estas Migraciones?

Si usamos un gestor de base de datos diferente a SQLite, la base de datos estará vacía inicialmente
Para ejecutar migraciones, usamos el comando ***php artisan migrate*** en la terminal. Este comando ejecutará todas las migraciones desde la carpeta ***database/migrations***.

¿Cómo cambiar el controlador de base de datos? Variables de configuración y entorno.

Ahora, ¿qué pasa si deseamos cambiar el controlador de la base de datos, de SQLite a MySQL, por ejemplo? Primero, vamos a verificar dónde se almacenan los valores.

Se almacenan en el archivo de configuración. En el archivo ***config/database.php***, podemos ver la clave de conexiones con diferentes controladores de base de datos disponibles listos para usar como SQLite, MySQL, MariaDB, etc.

Podemos ver valores con la función ***env()*** en cada controlador. Esta función auxiliar toma valores del archivo .env. Estos valores se denominan variables de entorno. Si el valor .env no está establecido, el segundo parámetro es el valor predeterminado.

Entonces, por ejemplo, si queremos usar MySQL en lugar de SQLite, primero en el .env, debemos cambiar el valor DB_CONNECTION a mysql. Luego, descomentamos los valores DB_xxxxx y los configuramos si los valores predeterminados no se adaptan a nuestra configuración. Los valores DB_DATABASE, DB_USERNAME y DB_PASSWORD en producción deben cambiarse.

Consejo profesional: nunca utilice la funcióne env() en su código fuera de los archivos de configuración.

Crear un nuevo archivo de migración

Enlace fuente del documento:
https://laraveldaily.com/course/laravel-beginners?mtm_campaign=search-results-course