


MÁSTER BACK-FRONT-END

CAPITULO 1 INTRODUCCIÓN

Esto será lo que utilizaremos a lo largo de este curso:

Backend / APIs REST

 PHP moderno

 MySQL

 Laravel

 MongoDB

 NodeJS

 Symfony

Frontend / WebApps SPA

 HTML y CSS

 Bootstrap 4

 JavaScript moderno

 Angular

 Librerías y módulos

 Desarrollo web full-stack

1 INTRODUCCION

Utilizaremos Laravel para construir el BACKEND y para el FRONTEND utilizaremos java script y angular.

Empezaremos por el backend, para ello tendremos que preparar un servidor de pruebas local, es decir un entorno de desarrollo donde poder probar los códigos que desarrollemos. También es posible utilizar un servidor real para hacer la pruebas.

Recomendamos el servidor de pruebas local como XAMPP o WampServer que es el que utilizaremos en este curso.

Necesitaremos un entorno de desarrollo integrado (IDE), utilizaremos NETBEANS, pero podremos utilizar Visual Studio...

1.2 INSTALACIÓN DE LARAVEL

La instalación se hace mediante un repositorio que se encuentra en GitHub, quizá haya que modificar la variable de entorno para que el CMD pueda usar correctamente php. Pero en resumen la instalación se realizará mediante la siguiente instrucción escrita en CMD en el caso de utilizar Windows, por cierto, hay que instalarse la aplicación composer para poder hacerlo:

>composer create-project laravel/laravel api-rest-laravel "5.7.*" --prefer-dist

Con esto instalará laravel en el equipo en la carpeta api-rest-laravel, dentro de WampServer pues hemos entrado allí y creado un directorio llamado cursoMaster, pero bueno lo de siempre jaja.

Para asegurar de que todo se ha instalado correctamente, iremos a la carpeta www del WampServer y comprobamos que se creó la carpeta con un tamaño de aproximadamente 30mb.

Además, como lo hemos instalado en el servidor local, si entramos en la ruta: <http://localhost/cursoMaster/laravel/api-rest-laravel/public/> podemos ver el índice que no proporciona el framework. Si la siguiente URL va correctamente todo habrá sido instalado bien: <http://localhost/cursoMaster/api-rest-laravel/public/>

Ahora pasaremos a cargar un proyecto en el entorno de desarrollo. Yo utilizare Visual Studio, pero se puede usar cualquier otro, abrimos la carpeta con Visual Studio y listo.

Una vez todo cargado, veamos las cosas más importante que nos aporta:

El package.json que incorpora muchas bibliotecas de laravel, la versión de php que estamos gastando, en el caso de que quisiéramos modificar algo de esto tendríamos que modificar este archivo y luego con composer, composer update para actualizarlo.

Carpeta app tenemos los controladores que veremos más adelante.

La carpeta routes tenemos web.php que proporciona la ruta de <http://localhost/cursoMaster/laravel/api-rest-laravel/public/> , y obviamente la podemos modificar.

En la carpeta resources/views tenemos welcome.blade.php que es la página principal que muestra la ruta public en laravel

1.3 CREAR UN HOST VIRTUAL

La url que hemos mostrado con anterioridad, en mi opinión es demasiado larga.

Vamos al archivo routes y creemos otra ruta pero en vez de devolver la vista welcome que es el archivo php del que hemos hablado antes, devuelva un return con hola mundo, quedaría así el archivo:

```
Route::get('/', function () {
    return "<h1> hola mundo <h1>";
});

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

Ahora si cargamos la ruta: <http://localhost/cursoMaster/laravel/api-rest-laravel/public/> estaremos cargando el hola mundo. Y si cargamos <http://localhost/cursoMaster/laravel/api-rest-laravel/public/welcome/> estaremos cargando la vista welcome.

Las URL quedan muy largas, para ellos vamos a crear un host virtual, es como hacer un dominio falso dentro de nuestro equipo, de esta manera trabajamos de manera realista y la ruta no es tan larga, por ejemplo, el dominio que vamos a elegir: cursoBackFront.com.

Para ello haremos lo siguiente:

Abrimos la carpeta apache dentro de wampserver, apacheVersion, conf, y el fichero httpd.conf, dentro de este archivo buscamos Virtual host y encontraremos el archivo **conf/extra/httpd-vhosts.conf**, este es el archivo que contiene los host virtuales que tenemos en nuestro equipo y será el que tendremos que modificar para crear un nuevos host.

En este archivo encontraremos:

```
# Virtual Hosts

#

<VirtualHost *:80>

    ServerName localhost

    ServerAlias localhost

    DocumentRoot "${INSTALL_DIR}/www"

    <Directory "${INSTALL_DIR}/www/">

        Options +Indexes +Includes +FollowSymLinks +MultiViews

        AllowOverride All

        Require local

    </Directory>

</VirtualHost>
```

Este no hay que tocarlo pues es el que utiliza apache del wamp server para que todo funciones correctamente. Crearemos otro:

```
<VirtualHost *:80>    //etiqueta crea host virtual puerto 80

    DocumentRoot "${INSTALL_DIR}/www/cursoMaster/laravel/api-rest-laravel/public" //ruta donde tenemos instalado
servidor local

    ServerName cursoDesarrollo.com

    ServerAlias www.cursoDesarrollo.com

    <Directory "${INSTALL_DIR}/www/cursoMaster/laravel/api-rest-laravel/public"> //ruta del directorio

        Options Indexes FollowSymLinks

        AllowOverride All

        Order Deny,Allow //esto sirve para restringir el acceso, en este caso a dispositivos de la red local

        Allow from all

    </Directory>

</VirtualHost>
```

Una vez tenemos esto tendremos que configurar las rutas de host de Windows para que al cargar el dominio que hemos especificado cargue la ruta que le hemos indicado. Para ello entraremos en la ruta (se puede mediante la ruta de direcciones de una carpeta cualquiera), **C:\Windows\System32\drivers\etc\hosts**

Y en ese archivo añadimos una nueva ruta para el local host: 127.0.0.1 cursoDesarrollo.com

Ahora podremos acceder a cursoDesarrollo.com que nos dará el hola mundo que creamos y el /welcome a la página principal

2 PRIMERO PASOS CON BACK-END LARAVEL

2.1 REPASO

Nos quedaremos con lo más importante que nos proporciona laravel:

Crear rutas: nos vamos a la carpeta routes, al archivo web.php y dentro de este podemos crear rutas:

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

Podemos pasar parámetros a las rutas, veremos más adelante como lo hacen los controladores, pero simplificando quedaría:

```
Route::get('/pruebas/{nombre}', function ($nombre) {  
    $texto = "<h2>mostrando texto desde una ruta</h2>";  
    $texto .= "nombre: ".$nombre;  
    return $texto;  
});
```

Si lo dejamos como esta, es obligatorio poner el parámetro en la ruta si nos nos devolverá un error para hacer que el parámetro sea opcional:

```
Route::get('/pruebas/{nombre?}', function ($nombre = null) {
```

Podemos hacer más cosas, por ejemplo, que la ruta nos muestre una vista mediante el método view al que le pasamos una ruta:

```
    return view('pruebas', array(  
        'texto' => $texto  
    ));
```

La ruta a la que accede es un archivo php que creamos en la carpeta views dentro de resources. Ahora imprimimos la variable texto desde ese archivo mediante una vista que es como deben hacerse las cosas con laravel.

Pasemos a ver un poco el tema de los controladores. Para crear un controlador, nos dirigimos a la consola de Windows, entramos a la carpeta del proyecto y ejecutamos la orden: `php artisan make:controller PruebasController`.

Una vez hecho esto tendremos el controlador creado y lo encontraremos en la carpeta controller dentro de app/http.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PruebasController extends Controller
{
    public function index(){
        $titulo = 'Animales';
        $animales = ['perro', 'gato', 'tigre'];

        return view('pruebas.index', array(
            'titulo' => $titulo,
            'animales' => $animales;
        ));
    }
}
```

Dentro de la clase hemos creado la función `index` que pasa a la vista `index` situada en `view/pruebas/index` dos cosas la variable `título` y el array `animales`.

```
<h1>{{$titulo}}</h1>
<ul>
    @foreach($animales as $animal)
        <li>{{$animal}}</li>
    @endforeach
</ul>
```

Como vemos podemos poner código php como si fueran etiquetas.

Ahora crearemos una nueva ruta que cargue el método del controlador para hacerlo:

```
Route::get('/pruebas/animales', 'PruebasController@index');
```

Cargamos el controlador y con el `@` indicamos el método que queremos que se ejecute