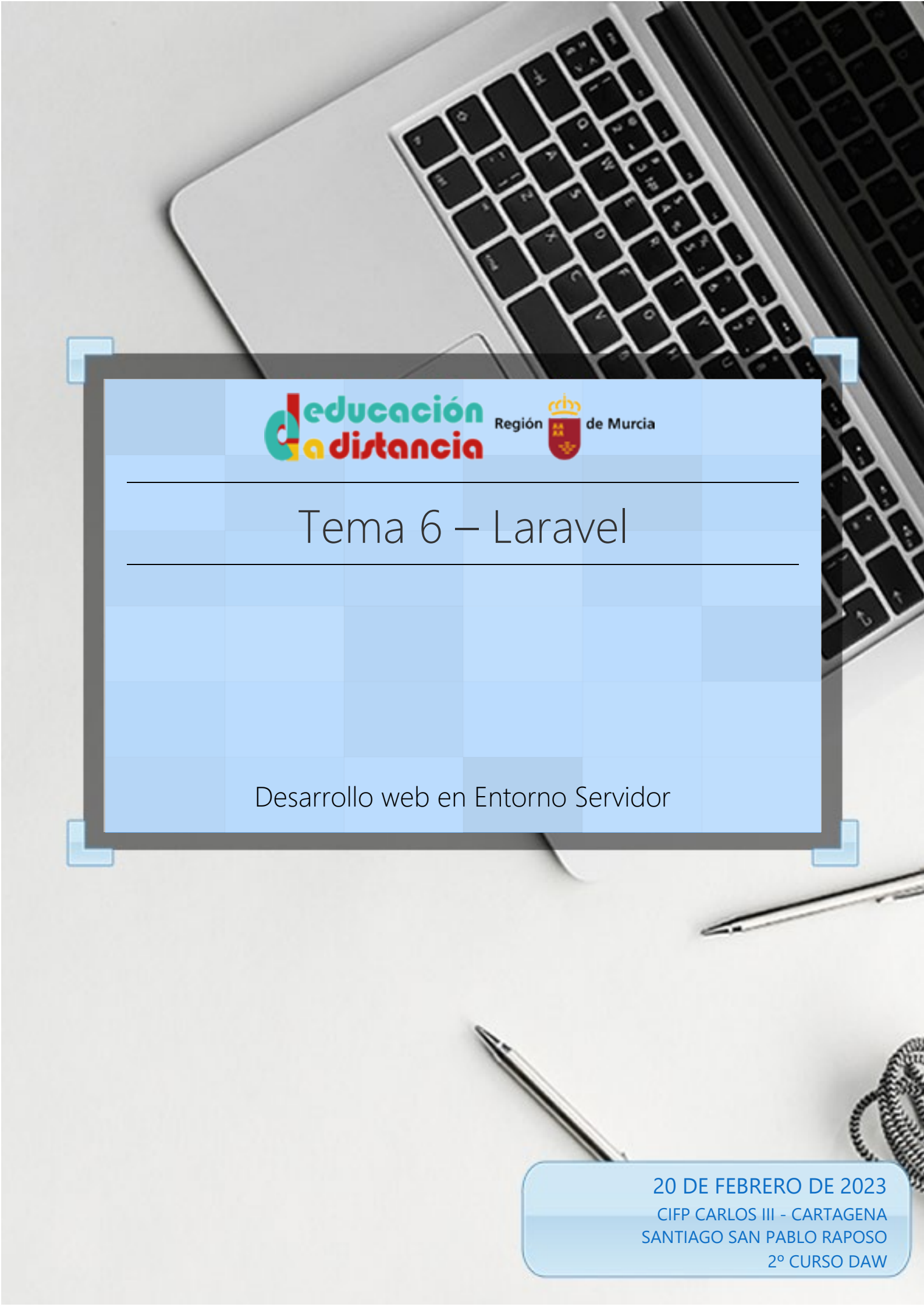


educación a distancia Región de Murcia

Tema 6 – Laravel

Desarrollo web en Entorno Servidor

20 DE FEBRERO DE 2023
CIFP CARLOS III - CARTAGENA
SANTIAGO SAN PABLO RAPOSO
2º CURSO DAW

Contenido

No se encontraron entradas de tabla de contenido.

Índice de ilustraciones.

No se encuentran elementos de tabla de ilustraciones.

Índice de tablas.

No se encuentran elementos de tabla de ilustraciones.

Resumen tema 6.

- (31) APRENDE LARAVEL 8 DESDE CERO EN MENOS DE 2 HRS - YouTube
  - https://youtu.be/a-4923Uyu54?t=1897
- Laravel Facades, ¿qué son y cómo y por qué usarlas? (victorfalcon.es)
- php - ¿Para qué, cómo y cuándo utilizar las palabras reservadas 'use' y 'namespace'? - Stack Overflow en español
  - PHP: Resumen de los espacios de nombres - Manual
  - Más sobre use: palabra reservada use php - Búsqueda (bing.com)

1.- Introducción a Laravel.

Los Frameworks son conjuntos de herramientas que nos facilitan desarrollar aplicaciones en un lenguaje de programación. Sus principales ventajas son:

- **Velocidad de desarrollo**: especialmente cuando hayamos dominado su curva de aprendizaje.
- **Seguridad y robustez**: el hecho de apostar por soluciones estables y comprobadas mejora la seguridad frente al código realizado sin estructura tipada.
- **Mantenimiento más sencillo**: La estructuración que hace el Framework sobre la aplicación facilita futuras modificaciones en el código.

Como desventajas podríamos destacar la dependencia de código externo y la curva de aprendizaje:

Características:

- Uno de los frameworks más **populares**.
- Continuas **actualizaciones**.
- **Curva de aprendizaje no muy difícil**.
- **Arquitectura MVC** (Modelo-Vista-Controlador). Es un sistema de desarrollo de software que separa una aplicación en tres capas:
  - **Modelo**: se refiere a la estructura de datos de la aplicación, ya sea proveniente de una BB.DD, clases…
  - **Vista**: representación de la información en una interfaz de usuario.
  - **Controlador**: donde se implementa la lógica de la aplicación. Son los algoritmos a desarrollar. El controlador funciona de interfaz entre modelo y vista.
- Utiliza el sistema de plantillas **Blade**.
- Para BB.DD usa el mapeo **ORM Eloquent**.
- Tiene su propio sistema de comandos, **Artisan**.

2.- Instalación de Laravel.

Instalar y actualizar paquetes con Composer – Styde.net

Requisitos previos: tener instalado LAMP. La instalación de Laravel la haremos con Composer, que es un gestor para instalación de software de terceros.

1. Habilitamos el módulo rewrite de Apache:

```
sudo a2enmod rewrite
```

2. **Descargar Composer** siguiendo sus instrucciones: https://getcomposer.org/download/

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') ===
'55ce33d7678c5a611085589f1f3ddf8b3c52d662cd01d4ba75c0ee0459970c2200a51f492d557530c71c15d8d
ba01eae') { echo 'Installer verified'; } else { echo 'Installer corrupt';
unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Que no se nos olvide después de instalarlo, hacer esto (lo pone en la página, pero por si acaso, lo recuerdo):

```
sudo mv composer.phar /usr/local/bin/composer
```

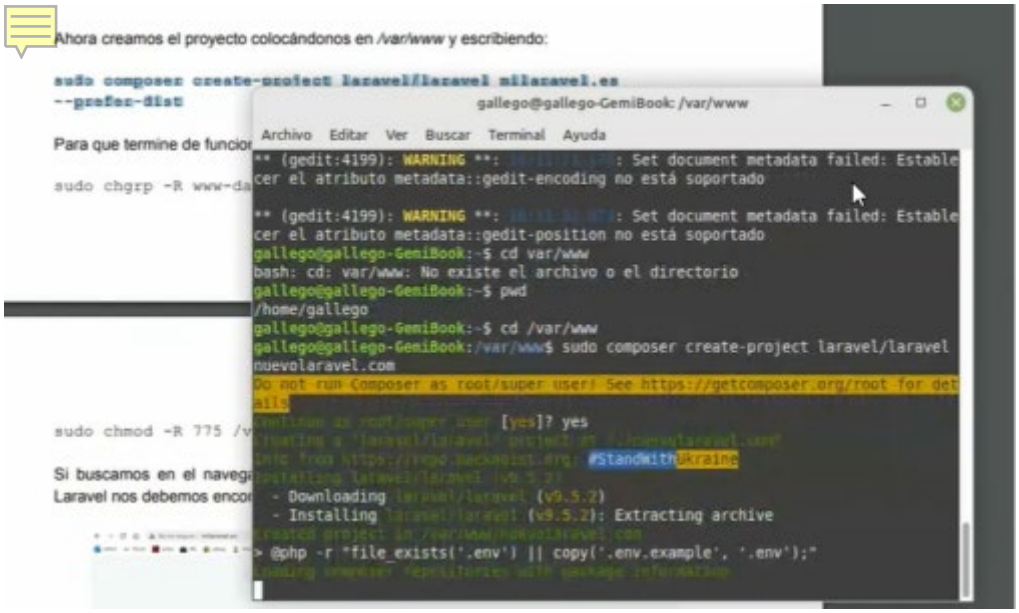
3. Actualizar Composer:

```
sudo composer self-update
```

4. **Crear un host virtual de Apache** donde instalaremos nuestro proyecto de Laravel.
    a. cd /etc/apache2/sites-available
    b. sudo copy 000-default.conf milaravel.es.conf
        i. Volcamos el siguiente contenido dentro del archivo

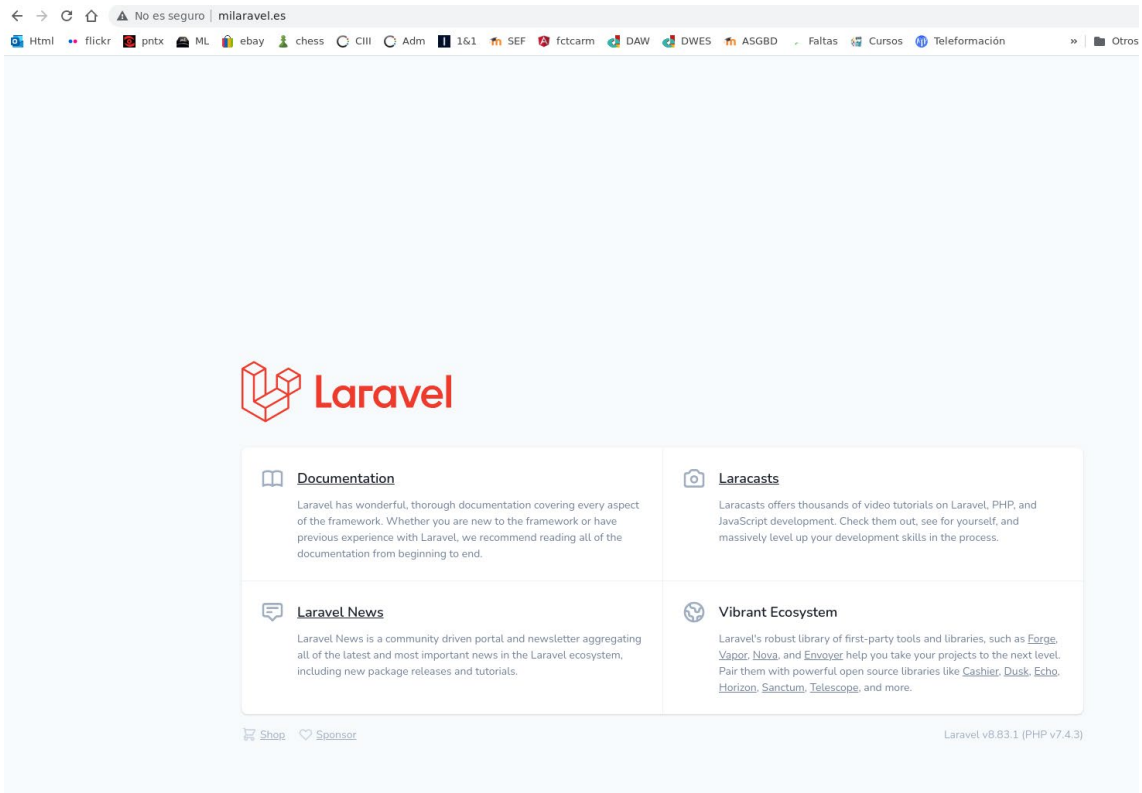
```
<VirtualHost *:80>
ServerName milaravel.es
ServerAdmin webmaster@localhost
DocumentRoot /var/www/milaravel.es/public
<Directory /var/www/milaravel.es>
AllowOverride All
</Directory>
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

    c. sudo a2ensite milaravel.es
    d. mkdir /var/www/milaravel.es
    e. sudo service apache2 restart
        i. sudo gedit /etc/hosts y añadimos la siguiente entrada al fichero de hosts:
            1. 127.0.0.1 milaravel.es
5. **Creamos el proyecto** de Laravel en /var/www:
    a. cd /var/www
    b. composer create-project laravel/laravel milaravel.es “9.5.1” --prefer-dist



c. **Permisos**:
   i. sudo chgrp -R www-data /var/www/milaravel.es
   ii. sudo chmod -R 775 /var/www/milaravel.es/storage

6. **Listo**. Si buscamos en el navegador la dirección que hemos puesto para nuestro proyecto de Laravel, nos debemos encontrar con esto:



Laravel, instala este framework para PHP en Ubuntu | Ubunlog

3.- Rutas en Laravel.

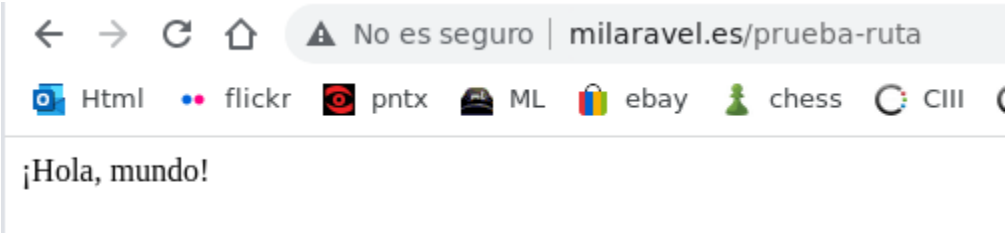
Básicamente, el trabajo con rutas consiste en lo siguiente:

- **Laravel comprueba a qué ruta se hizo la petición**.
- **Si la ruta está definida en el sistema de rutas, se ejecuta el código vinculado** a dicha ruta.

Laravel destina un directorio (routes) **para ubicar todas las rutas de la aplicación**. Por defecto tiene dos archivos de rutas: web.php y api.php. Inicialmente trabajaremos con web.php.

Ejemplo 1:

```
Route::get('/prueba-ruta', function () {
    return '¡Hola, mundo!';
});
```



Además del método "get" de la clase Route, **tendremos los siguientes métodos HTTP**:

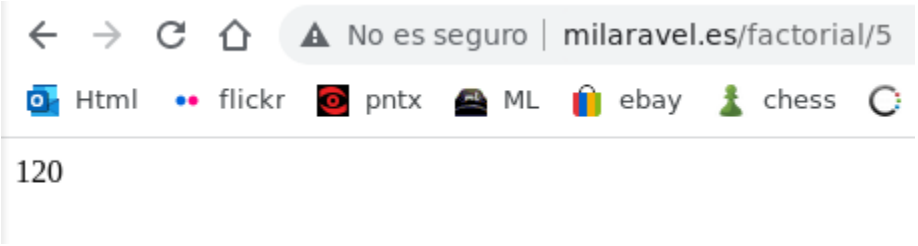
- Route::post();
- Route::put();
- Route::delete();
- Route::any();

Por ahora sólo usaremos los métodos GET (Acceso normal a la web) y POST (Petición con parámetros para el servidor).

Ejemplo 2: Con el sistema de rutas también se pueden crear **rutas más complejas que necesiten parámetros dinámicos**, para lo que tendremos que hacer dos modificaciones en la definición de la ruta.

1. Primero tendremos que indicarle que en la ruta relativa irá un parámetro, poniéndolo entre llaves; detrás habrá que especificar que la función recibirá una variable de PHP.

```
Route::get('/factorial/{n}', function ($n) {
    for ($i=1,$a=1;$i<=$n;$i++)
        $a*=$i;

    return $a;
});
```



Rutas con Laravel – Styde.net

4.- Vistas en Laravel.

Normalmente, como respuesta a las peticiones de ruta, no **querremos enviar** texto plano, sino **código HTML** para que el navegador lo procese.

Para definir el HTML a devolver usaremos las vistas, que básicamente son unas plantillas que dan el formato de la parte visual de la aplicación. **Las vistas se encuentran en el directorio *resources/views***.

Para crear una vista: deberemos añadir un archivo PHP en esa carpeta. Ahí vamos a insertar un archivo llamado básico.php.

```
<!-- resources/views/basico.php -->
<!doctype html>
<html lang="es">
   <head>
      <meta charset="UTF-8">
      <title>Desarrollo Web en Entorno Servidor</title>
   </head>
   <body>
      <p>Página principal del curso mediante una vista.</p>
   </body>
</html>
```

Vamos a utilizar esta vista para dar formato a la página principal del proyecto:

Dos formas de hacerlo

```
Route::get('/', function () {
    return View::make('basico');
});
```

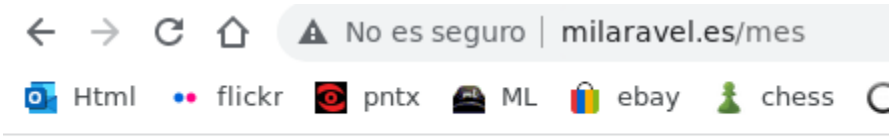
```
Route::get('/', function () {
    return view('basico');
});
```

Es posible pasar parámetros a una vista. Dentro de la ruta le pasaremos el parámetro **con el método with**:

```
Route::get('/mes', function () {
   $mes="Febrero";
   return view('mes')->with([
      'mes'=>$mes
   ]);
});
```

Y así podremos usar el parámetro dentro de la vista:

```
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <title>Desarrollo Web en Entorno Servidor</title>
    </head>
    <body>
        <p>Estamos en el mes de <?php echo $mes; ?></p>
    </body>
</html>
```



Estamos en el mes de Febrero

Vistas en Laravel – Styde.net

Introducción a las vistas en Laravel (desarrolloweb.com)

5.- Formularios en Laravel.

La **creación de formularios** la haremos en un principio **dentro de una vista**, y usaremos el **método POST**. A continuación, tenemos un ejemplo:

```
<html>
<head>
    Formulario de ejemplo - DWES
</head>
<body>
    <br><br>
    <form action="{{ url('/formulario2') }}" method="post">
        Parámetro 1<input type="text" name="p1" id="p1"><br>
        Parámetro 2<input type="text" name="p2" id="p2"><br>
        <input type="submit" value="Enviar" />
```

```
        </p>
    </form>
</body>
</html>
```

Código revisado en la tutoría:

```
<html>
    <head>
        Formulario de ejemplo - DWES
    </head>
    <body>
        <br><br>
        <form action="formulario2" method="get">
            Parámetro 1<input type="text" name="p1"
                             id="p1"><br>
            Parámetro 2<input type="text" name="p2"
                             id="p2"><br>
            <input type="submit" value="Enviar" />
        </p>
    </form>
</body>
</html>
```

Después, **añadimos una ruta para procesar los datos enviados por el formulario** y mostramos la información recibida:

```
Route::get('/formulario2', function () {
    $datos = Request::all();
    return $datos;
});
```

Si nos interesa tener solo uno de los datos enviados por el formulario, usamos Request de la siguiente forma:

```
Route::get('/formulario2', function () {
    $dato = Request::get('p1');
    return $dato;
});
```

Cómo crear formularios en Laravel 5 (jesuschicano.es)
Formularios - Laravel 5 (gitbook.io)
Creando y consumiendo nuestro primer formulario de ruta tipo POST en Laravel - Desarrollolibre

6.- Controladores en Laravel.

Hasta ahora, el código lo hemos insertado en la misma definición de las rutas. Para estos ejemplos puede ser válido, pero en proyectos de cierta envergadura, no es factible ni aconsejable tener toda la lógica de la aplicación en un solo archivo. **Para evitar esto usaremos los controladores y veremos cómo asignarlos a rutas**.

Primer método para crear un controlador

Añadiendo un archivo PHP en el directorio app/Http/Controllers

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class PruebaController extends Controller {
    public function index() {
        return view('formulario');
    }
}
```

Con esto, **hemos creado la clase PruebaController** y un **método llamado *index*** que conecta con la vista de nuestro formulario. Ahora toca añadir la ruta en web.php:

```
Route::get('/f', 'App\Http\Controllers\PruebaController@index');
```

Donde **escribimos como primer parámetro la URI de la ruta, y como segundo *controlador@método* del controlador**. Esta forma de crear la ruta es propia de Laravel 8, por lo que difiere a lo que se puede encontrar en la mayoría de manuales, pues suelen ser anteriores a esta versión.

Segundo método para crear un controlador

Sería mediante *Artisan*, que es la interfaz de comandos de *Laravel*. **En el terminal de comandos de *Linux*** y **sobre el directorio del proyecto**, **escribimos**:

```
/var/www/milaravel.es$ sudo php artisan make:controller PruebaController2
```

La utilización de *sudo* se debe a los permisos de los directorios.

Ampliación del segundo método: Usos de Artisan para crear controladores CRUD

Artisan puede ser de mayor utilidad cuando vamos a crear un controlador CRUD para un tipo de recurso. Estos controladores suelen tener siempre métodos para el listado, el detalle, la creación, la edición y la eliminación de elementos. Lo ejecutaríamos de la siguiente forma:

```
php artisan make:controller PruebaController2 --resource
```

Automáticamente, **creará una serie de métodos en nuestro controlador**:

Método	Qué hace

index	Es el **método inicial de las rutas resource**, usualmente **lo usamos para mostrar una vista como página principal**, que puede contener un catálogo o resumen de la información del modelo al cual pertenece, o bien no mostrar información y solo tener la función de página de inicio.
create	Este método lo podemos usar **para direccionar el sistema a la vista** donde se van a recolectar los datos (probablemente con un **formulario**) para después almacenarlos en un **registro nuevo**. Usualmente redirige al index.
show	Permite **crear una consulta de un elemento de la BB.DD** o de todos los elementos o registros por medio del modelo para realizar una descripción.
edit	Este método es **similar al create**, porque lo podemos usar para mostrar una vista que recolecta los datos. Pero **este sirve para actualizar un registro**.
store	Aquí es donde se **actualiza un registro en específico** que **proviene del método create** y normalmente redirige al index.
update	Al igual que store, solo que **proviene de edit**, y en vez de crear un nuevo registro, busca un existente y lo modifica. También suele redirigir al index.
destroy	En este método, usualmente **se destruye o elimina un registro** y la petición puede provenir de donde sea, siempre y cuando sea llamado con el método **DELETE**, después puede redirigir al index o a otro sitio dependiendo de si logró eliminar o no.

Para no tener que enrutar uno a uno cada método, usamos:

Route::resource('f2', 'App\Http\Controllers\PruebaController2');

Los métodos de enrutación se hacen de la siguiente forma:

Método	URI	Acción
GET	/f2	f2.index
GET	/f2/create	f2.create
POST	/f2	f2.store
GET	/f2/{id}	f2.show
GET	/f2/{id}/edit	f2.edit
PUT/PATCH	/f2/{id}	f2.update
DELETE	/f2/{id}	f2.destroy

Capítulo 11. Controladores · laravel-5 (gitbooks.io)
Controladores en laravel 8 - Norvic Software
Controladores en Laravel (pleets.org)
[Solved] Target class does not exist in laravel 8 - Exception Error (exerror.com)

7.- Migraciones de BB.DD.

Para poder trabajar con una BB.DD en Laravel,

1. Tenemos que **editar el archivo .env** y configurar los siguientes campos:

```
DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=dwes
DB_USERNAME=super
DB_PASSWORD=123456
```

2. **La base de datos debe estar creada**. Crearla previamente con PHPMyAdmin o bien, usando la terminal de SQL.
    a. Si no la creamos, obtendremos errores al tratar de acceder a ella.
3. **La migración de la BB.DD nos permitirá crear las tablas** sobre las que trabajaremos en la aplicación.
    a. **Esto nos permitirá estructurar nuestra BB.DD desde la misma aplicación** y además, si necesitamos volver a crear las tablas porque hemos trasladado la aplicación o hemos perdido la BB.DD, podremos regenerar toda la estructura ejecutando un comando.

Como ejemplos de migraciones vamos a generar unas tablas similares a las de la tarea 3, aunque no vamos a generar toda la base de datos y en algunas columnas habrá algunas diferencias para poder ejemplificar mejor los conceptos de la migración.

7.1.- Creación de una tabla.

1. Ejecutar el comando de Artisan para **crear la migración** que se encargará de crear la tabla.

```
sudo php artisan make:migration create_libros_table
```

Este comando nos crea un nuevo archivo en la carpeta database/migration, con la fecha actual y el nombre indicado (libros).

Por defecto, nos viene con el atributo timestamps, que lo que hace es guardar en una columna la fecha y hora de cuándo se creó el registro en la tabla:

```
/**
 * Run the migrations.
 *
 * @return void
 */
public function up()
{
    Schema::create('libros', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

Sin embargo, nosotros lo omitiremos, y en el siguiente paso, usaremos nuestras propias columnas.

Otra cosa: el id() es bueno dejarle, porque automáticamente, Laravel ya reconoce esa columna como clave primaria. (Si la hacemos llamar a esa columna de otra forma, como

vamos a hacer a continuación, luego, en el modelo con ORM Eloquent, necesitarás especificar mediante un atributo el nombre de la clave primaria):

2. **Describimos la estructura de la tabla** en ese nuevo archivo.

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateLibrosTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('libros', function (Blueprint $table) {
            $table->increments('lib_id'); // Crea una clave primaria
autoincremental
            $table->string('lib_titulo'); // Columna tipo texto
            $table->integer('lib_paginas'); // Tipo entero
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('libros');
    }
}
```

Laravel - Base de datos:Migraciones - Introduction Las migraciones son como un control de versiones para tu base de da - Español (runebook.dev)

Con esto hemos creado dos funciones:

- **up**: que crea la tabla.
- **down**: que la borra.

3. **Para finalizar la migración**, ejecutamos en la línea de comandos:

```
php artisan migrate
```

Si queremos **deshacer la migación**, escribimos:

```
php artisan migrate:rollback
```

Esto desharía la última migración que hayamos realizado. Si por ejemplo, tenemos que hemos hecho todas estas migraciones con el comando migrate:

En este caso, desharía la última, la de id=5.

7.2.- Crear una columna en una tabla existente.

Para añadir una columna a una tabla existente:

```
sudo php artisan make:migration add_autor_to_libros
```

Nótese que hay que usar sudo cada vez que implique la creación de un archivo, para hacer rollback no ha sido necesario

A continuación, editamos el nuevo archivo:

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class AddAutorToLibros extends Migration {
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up() {
        Schema::table('libros', function (Blueprint $table) {
            $table->string('lib_autor')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
```

```
     * @return void
     */
    public function down() {
        Schema::table('libros', function (Blueprint $table) {
            $table->dropColumn('lib_autor');
        });
    }
}
```

Y volvemos a invocar la migración desde el intérprete de comandos:

```
php artisan migrate
```

7.3.- Creación de una relación.

A continuación, y tal y como estaba en la BB.DD de la tarea 3, vamos a crear una tabla para Autores.

1. **Tenemos que deshacer la columna que habíamos puesto como string**, para ello, hacemos rollback y eliminamos el archivo que la creaba (add_autor_to_libros).

```
php artisan migrate:rollback
```

2. Escribimos en el intérprete de comandos el comando para **crear la nueva migración**:

```
sudo php artisan make:migration create_autores_table_and_relationship
```

Ahora implementamos las funciones:

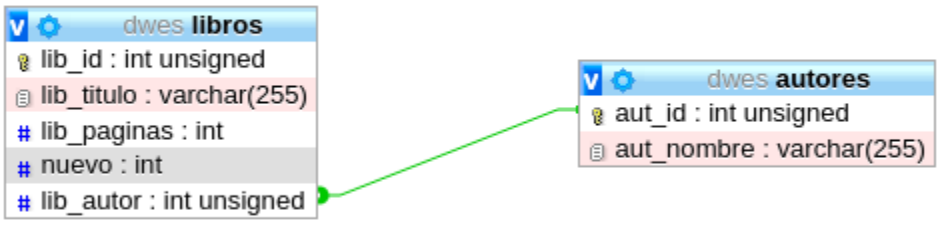
```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateAutoresTableAndRelationship extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('autores', function (Blueprint $table) {
            $table->increments('aut_id');
            $table->string('aut_nombre');
        });
        Schema::table('libros', function (Blueprint $table) {
            $table->integer('lib_autor')->unsigned()->nullable();
```

```
            $table->foreign('lib_autor')->references('aut_id')-
>on('autores')->onDelete('set null'); // También podemos poner cascade o
restrict
        });
    }
    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('libros', function (Blueprint $table) {
            $table->dropForeign(['lib_autor']);
            $table->dropColumn('lib_autor');
        });
        Schema::dropIfExists('autores');
    }
}
```

3. **Ejecutar la migración** para ver los efectos.

```
php artisan migrate
```

Ahora, deberíamos ser capaces de ver la segunda tabla y la relación desde PHPMyAdmin:



Capítulo 6. Migraciones y Seeders · laravel-5 (gitbooks.io)

Database: Migrations - Laravel - The PHP Framework For Web Artisans

8.- Modelos con ORM Eloquent.

El ORM (Modelo Objeto-Relacional) Eloquent, viene integrado en Laravel y **permite trabajar con la BB.DD interactuando directamente con objetos en los que se mapean las filas de las tablas**, abstrayéndolos así del sistema de BB.DD que vayamos a usar.

8.1.- Creación de un modelo.

1. Para crear un modelo, usamos Artisan:

```
sudo php artisan make:model Autor
```

2. Eso nos crea un archivo en el directorio /App/Models. Lo implementamos con el siguiente contenido:

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Autor extends Model
{
    use HasFactory;
    protected $table = 'autores'; // Indicamos que cree una clase a
partir de la tabla autores
    public $timestamps = false; // La clase no incluirá los timestamps
}
```

Mencionar una cosa: si la tabla se llamase como la clase, pero con una "s" final, es decir, Autors, no haría falta especificar mediante el atributo $table el nombre de la tabla.

8.2.- Inserción de una fila.

Eloquent realiza una asignación entre clases y tablas, por lo que, para trabajar con las filas de una tabla, tendremos que tratarlas como objetos instanciados de esas clases (correspondientes a las tablas). El código para realizar una inserción sería:

```php
<?php
use App\Models\Autor;

Route::get('/', function () {
    $autor = new Autor; // Funciona con () y sin ellos
    $autor->aut_nombre = "Miguel de Cervantes";
    $autor->save(); // Método que guarda el objeto en la tabla
    echo 'Autor guardado en la tabla';
});
```

En este caso, **hemos puesto el código en el archivo de rutas**, pero **en proyectos de mayores dimensiones, deberemos insertarlo en los controladores**.

Mencionar que hay que declarar una directiva "use" para indicarle a Laravel que usamos la clase Autor.

8.3.- Recuperar una fila por su clave primaria.

El código sería así:

```php
<?php
use App\Models\Autor;

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

```
    $autor = Autor::find('1');
    return $autor->aut_nombre;
});
```

Pero **Eloquent supone que la clave primaria se llama "id", en nuestro caso se llama "aut_id", por lo que hay que especificárselo en la definición del modelo**, añadiendo la línea:

```
protected $primaryKey = 'aut_id';
```

8.4.- Otras operaciones.

Operación	Código

Actualización	`<?php`
	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`  $autor = Autor::find(1);`
	`  $autor->aut_nombre = "Calderón de la Barca";`
	`  $autor->save();`
	`});`

Eliminación	`<?php`
	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`  $autor = Autor::find(1);`
	`  $autor->delete();`
	`});`

Eliminación mediante clave primaria	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`  Autor::destroy(1);`
	`});`

Eliminación múltiple mediante clave primaria	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`  Autor::destroy([3, 4]);`
	`});`

9.5.- Consultas.

Operación	Código

Todas las filas	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`  $autores = Autor::all();`
	`  var_dump($autores); // Para ver todo el array`

	`echo $autores[0]->aut_nombre; // Para ver un elemento`
	`});`

Primera fila	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`$autor = Autor::first();`
	`var_dump($autor); // Para ver todo el array`
	`echo $autor->aut_nombre; // Para ver un elemento`
	`});`

Actualización	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`Autor::where('aut_id', '=', '5')`
	`->update([`
	`'aut_nombre' => 'Lope de Vega'`
	`]);`
	`});`

Borrado	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`Autor::where('aut_id', '=', '5')->delete();`
	`});`

Where (consulta)	`use App\Models\Autor;`
	
	`Route::get('/', function () {`
	`$autores = Autor::where('aut_id', '>', '5')->get();`
	`var_dump($autores);`
	`});`

Eloquent: Getting Started - Laravel - The PHP Framework For Web Artisans

Aprende a usar Eloquent el ORM de Laravel – Styde.net

9.6.- Consultas SQL.

Si necesitamos hacer consultas más complejas, o directamente, usar SQL en Laravel, tendremos que hacer uso de las fachadas o Facades, concretamente de la Facade \Facades\DB.

Las fachadas son una especie de interfaz que tiene Laravel para proporcionarnos funciones muy potentes simplificando el código.

1. Primero, **debemos poner en la parte superior del archivo**:

```
use Illuminate\Support\Facades\DB;
```

2. **Después**, en el código, **podremos utilizar**:

```
$autores = DB::select("select * from autores");
```

3. Esto nos dejará en la variable $autores un array de objetos, concretamente, es un array de arrays asociativos:

```
foreach ($autores as $autor) {
    echo $autor->aut_nombre;
}
```