

FRAMEWORK LARAVEL PARA DESARROLLO WEB

DOCENTE

CARLOS ANDRES CASTAÑEDA OSORIO

**Ingeniería Informática
Universidad de Caldas
Manizales
2023**

1. CONCEPTOS BÁSICOS

El presente documento es creado con el fin de mostrar el funcionamiento básico de Laravel el framework de PHP con la comunidad más grande siendo este probablemente el más usado del mercado.

Para iniciar es necesario tener claros algunos conceptos clave del desarrollo web con el lenguaje de programación PHP. A continuación se definen conceptos y se dejan referencias con algunos enlaces de interés para que el estudiante profundice el conocimiento adquirido en clase.

1.1. PHP



PHP (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML. [Versión actual](#)

Bien, pero ¿qué significa realmente? Un ejemplo nos aclara las cosas:

Ejemplo #1 Un ejemplo introductorio

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>

    <?php
      echo "¡Hola, soy un script de PHP!";
    ?>

  </body>
</html>
```

En lugar de usar muchos comandos para mostrar HTML (como en C o en Perl), las páginas de PHP contienen HTML con código incrustado que hace "algo" (en este

caso, mostrar "¡Hola, soy un script de PHP!). El código de PHP está encerrado entre las etiquetas especiales de comienzo y final <?php y ?> que permiten entrar y salir del "modo PHP".

Lo que distingue a PHP de algo del lado del cliente como Javascript es que el código es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá el resultado de ejecutar el script, aunque no se sabrá el código subyacente que era. El servidor web puede ser configurado incluso para que procese todos los ficheros HTML con PHP, por lo que no hay manera de que los usuarios puedan saber qué se tiene debajo de la manga.

Lo mejor de utilizar PHP es su extrema simplicidad para el principiante, pero a su vez ofrece muchas características avanzadas para los programadores profesionales. No sienta miedo de leer la larga lista de características de PHP. En unas pocas horas podrá empezar a escribir sus primeros scripts.

Aunque el desarrollo de PHP está centrado en la programación de scripts del lado del servidor, se puede utilizar para muchas otras cosas. Siga leyendo y descubra más en la sección ¿Qué puede hacer PHP?, o vaya directo al tutorial introductorio si solamente está interesado en programación web.

Documentación oficial: <https://www.php.net/manual/es/intro-whatis.php>

Cursos de PHP:

- **Platzi:** <https://platzi.com/cursos/php/>
- **Udemy:** <https://www.udemy.com/courses/search/?q=php&src=sac&kw=php>
- **Codecademy:** <https://www.codecademy.com/learn/learn-php>
- **Coursera:** <https://es.coursera.org/learn/web-applications-php>
- **W3Schools:** <https://www.w3schools.com/php/>

Al ser un lenguaje de programación web tan ampliamente utilizado tiene asociados muchos frameworks que amplían sus funcionalidades y permiten al desarrollador facilitar labores que ya se identificaron como usadas en todas las aplicaciones web.

Algunos de los Frameworks más conocidos de PHP son:

1. [Laravel](#)
2. [CodeIgniter](#)
3. [Symfony](#)
4. [Zend](#)
5. [Phalcon](#)
6. [CakePHP](#)
7. [Yii](#)
8. [FuelPHP](#)

Tomados de: <https://www.hostinger.co/tutoriales/mejores-frameworks-php/>

1.2. Laravel



Laravel es un marco de aplicación web con una sintaxis expresiva y elegante. Un marco web proporciona una estructura y un punto de partida para crear su aplicación, lo que le permite concentrarse en crear algo increíble mientras nos preocupamos por los detalles. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti". Fue creado en 2011 y tiene una gran influencia de frameworks como Ruby on Rails, Sinatra y ASP.NET MVC.2

Laravel tiene como objetivo ser un framework que permita el uso de una sintaxis elegante y expresiva para crear código de forma sencilla y permitiendo multitud de funcionalidades. Intenta aprovechar lo mejor de otros frameworks y aprovechar las características de las últimas versiones de PHP.2

Gran parte de Laravel está formado por dependencias, especialmente de Symfony, esto implica que el desarrollo de Laravel dependa también del desarrollo de sus dependencias.

Características

- Sistema de ruteo, también RESTful3
- Blade, Motor de plantillas
- Peticiones Fluent
- Eloquent ORM7
- Basado en Composer
- Soporte para el caché
- Soporte para MVC
- Usa componentes de Symfony
- Adopta las especificaciones PSR-2 y PSR-4

Documentación Oficial: <https://laravel.com/docs/9.x>

1.3 Composer



Composer es una herramienta para la gestión de dependencias en PHP. Le permite declarar las bibliotecas de las que depende su proyecto y las administra (instalará / actualizará) por usted.

Gestión de dependencias

Composer no es un administrador de paquetes en el mismo sentido que Yum o Apt. Sí, trata con "paquetes" o bibliotecas, pero los administra por proyecto, instalándose en un directorio (por ejemplo vendor) dentro de su proyecto. Por defecto, no se instala nada a nivel global. Por lo tanto, es un administrador de dependencias. Sin embargo, admite un proyecto "global" por conveniencia a través del comando global.

Esta idea no es nueva y Composer está fuertemente inspirado en el paquete npm y el paquete de Ruby.

Suponer:

1. Tiene un proyecto que depende de varias bibliotecas.
2. Algunas de esas bibliotecas dependen de otras bibliotecas.

Composer:

1. Le permite declarar las bibliotecas de las que depende.
2. Averigua qué versiones de qué paquetes pueden y deben instalarse, y los instala (lo que significa que los descarga en su proyecto).
3. Puede actualizar todas sus dependencias en un solo comando.

Consulte el capítulo [Uso básico](#) para obtener más detalles sobre la declaración de dependencias.

Requisitos del sistema

Composer requiere PHP 7.2.5+ para ejecutarse. También se requieren algunas configuraciones sensibles de php y marcas de compilación, pero al usar el instalador se le advertirá sobre cualquier incompatibilidad.

Para instalar paquetes de fuentes en lugar de simples archivos zip, necesitará git, svn, fossil o hg dependiendo de cómo se controle la versión del paquete.

Composer es multiplataforma y funciona igualmente bien en Windows, Linux y macOS.

2. Aprovisionamiento del ambiente de desarrollo

Como PHP es un lenguaje del lado del servidor, necesita ser procesado por un servidor web, para el ambiente de desarrollo podemos encontrar entornos completos que traen instalados por defecto los programas básicos con los que PHP es usado. El primer paso del curso será ver conceptos básicos de PHP, por ello instalaremos un ambiente inicial en el que PHP se encuentre soportado para pruebas.

Utilizaremos XAMPP como entorno de desarrollo, al instalar esta herramienta se instalará Apache como servidor web, MariaDB como servidor de bases de datos, los paquetes de PHP adicionales requeridos para la compilación y algunos paquetes adicionales que requiere para su funcionamiento.

Enlace de XAMPP: <https://www.apachefriends.org/es/index.html>

Cambiar de puerto a XAMPP:

<https://stackoverflow.com/questions/11294812/how-to-change-xampp-apache-server-port>

Para instalar XAMPP seguiremos los siguientes pasos:

1. Descargar XAMPP de su página oficial, se recomienda que sea el que soporta la versión de PHP 8.1.2

Apache Friends Descargar Complementos Alojamiento Comunidad Acerca de Buscar.. Buscar ES

XAMPP Apache + MariaDB + PHP + Perl

¿Qué es XAMPP?

XAMPP es el entorno más popular de desarrollo con PHP

XAMPP es una distribución de Apache completamente gratuita y fácil de instalar que contiene MariaDB, PHP y Perl. El paquete de instalación de XAMPP ha sido diseñado para ser increíblemente fácil de instalar y usar.



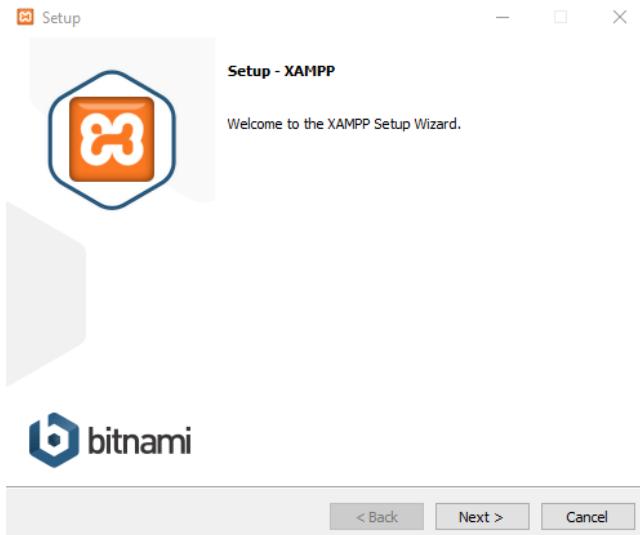
Descargar
Pulsa aquí para otras versiones

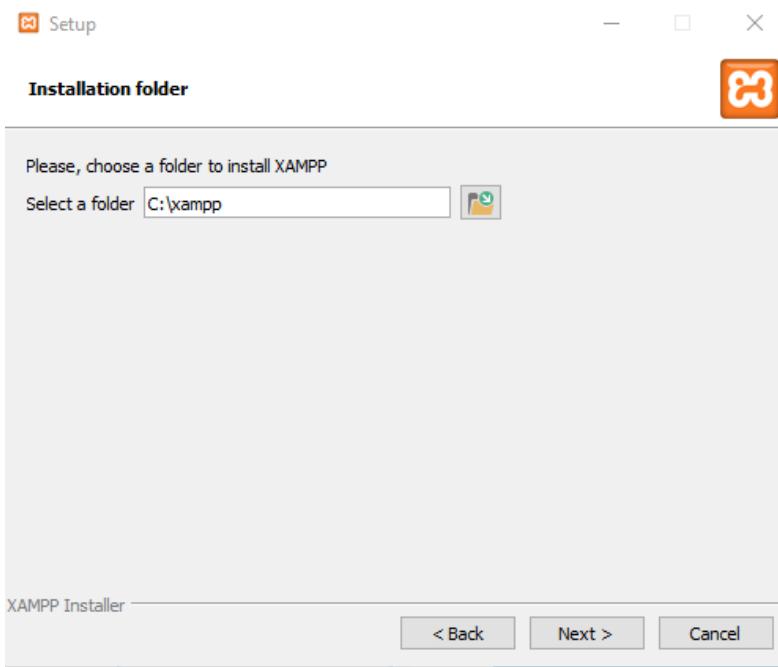
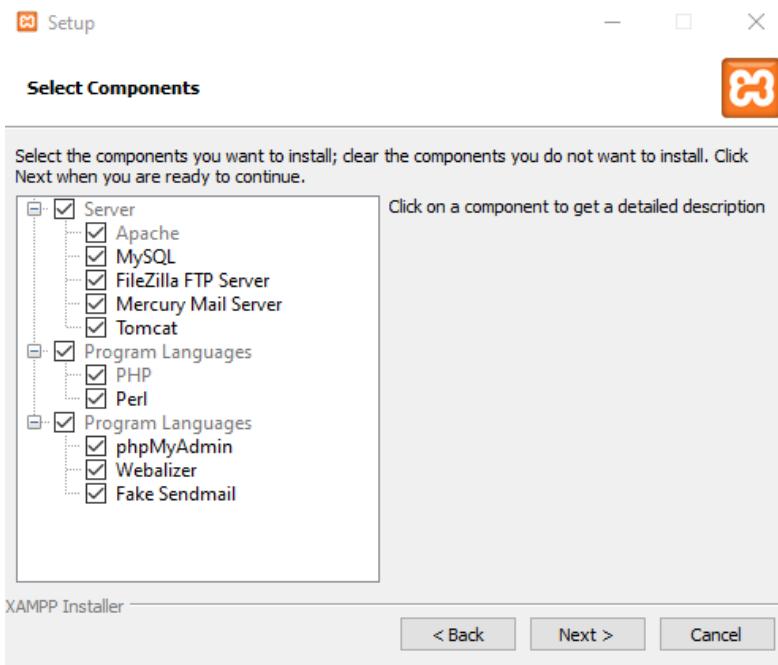
XAMPP para Windows
8.1.2 (PHP 8.1.2)

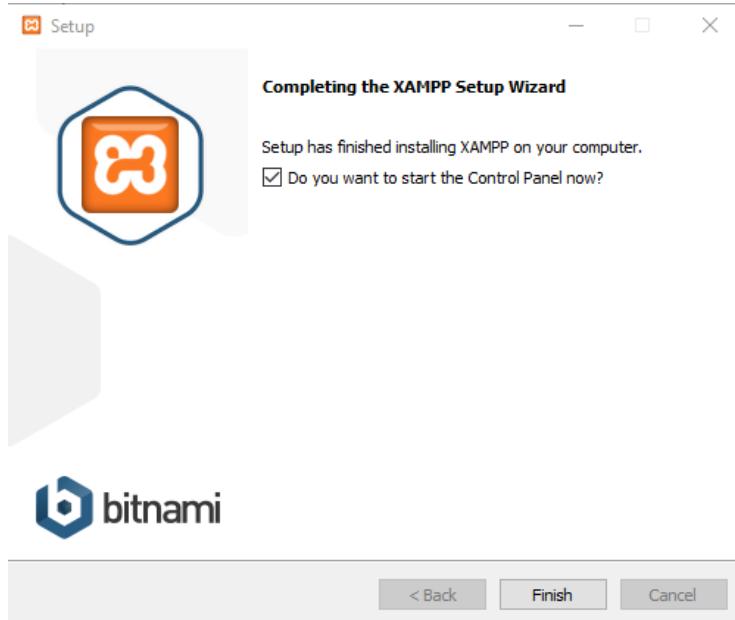
XAMPP para Linux
8.1.2 (PHP 8.1.2)

XAMPP para OS X
8.1.2 (PHP 8.1.2)

- Al descargar el archivo se procede a abrirlo y realizar la instalación guiada, en esta instalación no se tendrán que modificar parámetros y se podrá dejar por defecto todos los valores. A continuación se muestra el paso a paso en imágenes:



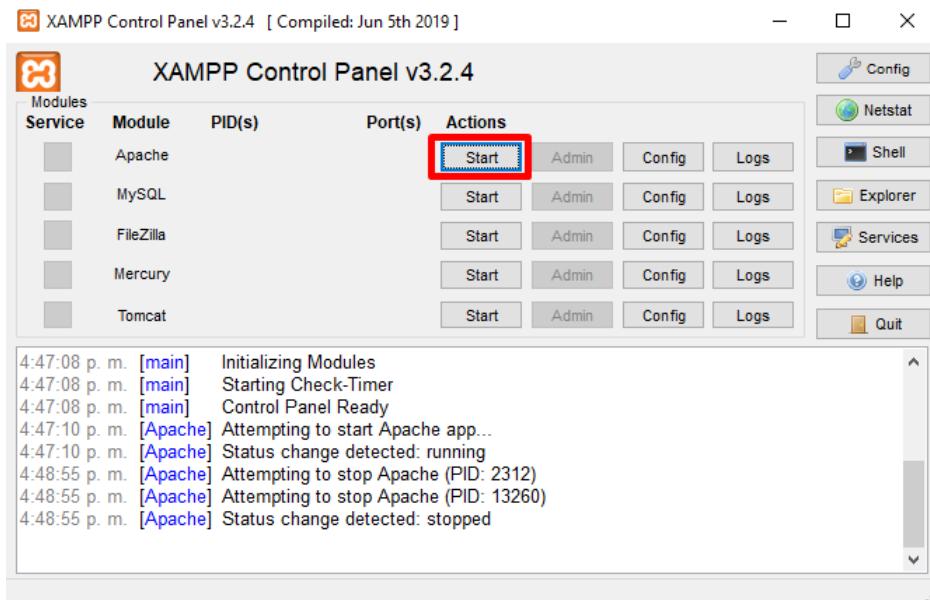




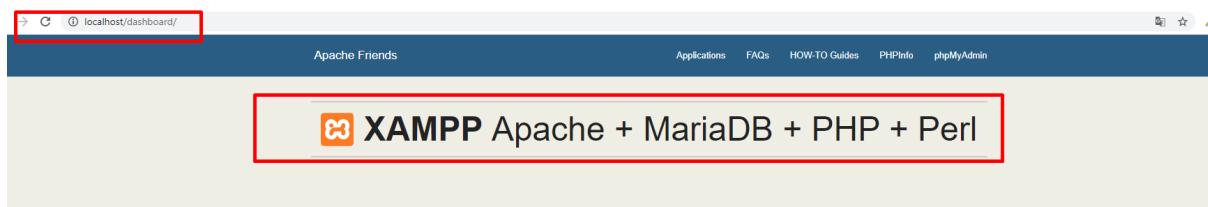
3. Por defecto en la carpeta `C://xampp/htdocs` se encuentran alojados los aplicativos que el servidor lee y nos permite ver en nuestra página web.

Portapapeles				Organizar	Nuevo	Abrir
→	▼	↑	Este equipo >	OS (C:) > xampp > htdocs >		
Acceso rápido			Nombre		Fecha de modificación	Tip
Escritorio	↗		dashboard		21/02/2020 4:14 p. m.	Ca
Descargas	↗		img		21/02/2020 4:14 p. m.	Ca
Documentos	↗		webalizer		21/02/2020 4:14 p. m.	Ca
Imágenes	↗		xampp		21/02/2020 4:14 p. m.	Ca
3. VS Aguilas			applications.html		27/08/2019 9:02 a. m.	Ch
Análisis de SI I			bitnami.css		27/08/2019 9:02 a. m.	Ar
Focus T25 ALPHA P			favicon.ico		16/07/2015 10:32 a. m.	Icc
Tesis			index.php		16/07/2015 10:32 a. m.	Ar
Creative Cloud Files						

4. Cada vez que deseemos habilitar el servidor para iniciar a desarrollar, debemos buscar el programa dentro de nuestro sistema operativo, al abrir hacer clic en start que se encuentra ubicado enseguida de Apache.



5. Si todos los pasos anteriores fueron realizados de forma correcta, será posible desde un navegador web acceder a la dirección URL <http://localhost>

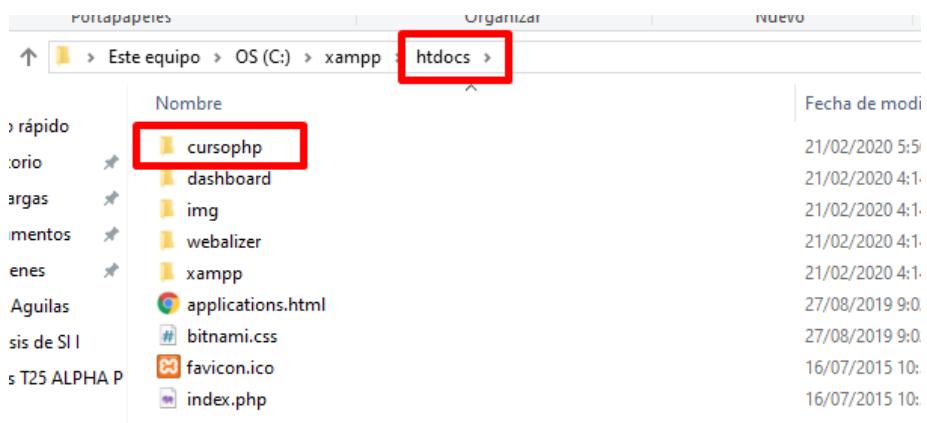


3. MANOS AL CÓDIGO

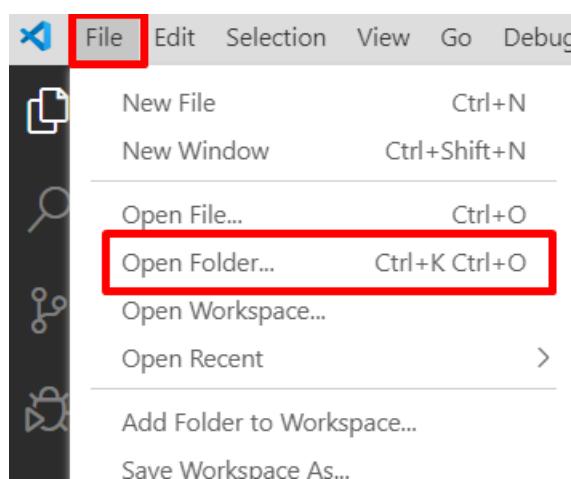
Los códigos a realizar en clase se seguirán bajo el IDE [PHP Storm](#) ([Licencia estudiantil con correo de la universidad](#)). Es de aclarar que el código escrito en el curso puede ser creado en otros editores de texto y código como Visual Studio Code o Sublime Text.

3.1. Mi primera página web usando PHP

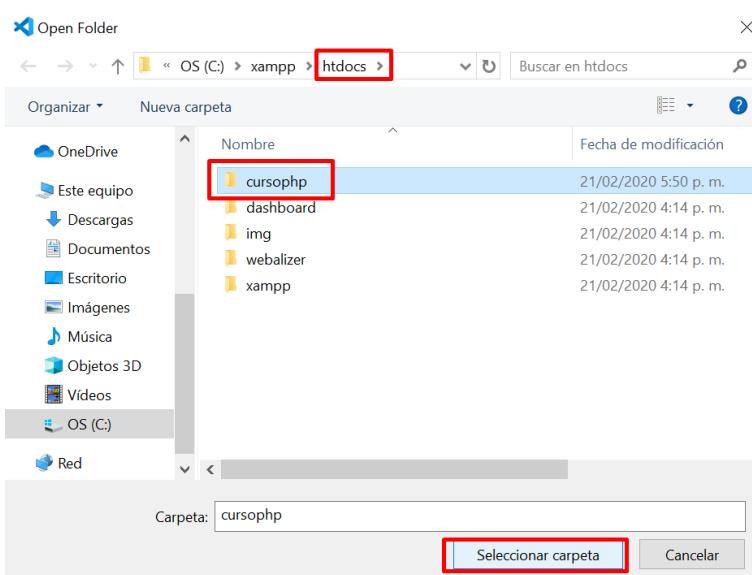
Para iniciar en la carpeta htdocs de XAMPP crearemos una carpeta llamada *cursophp* en donde almacenaremos todo el contenido de la página web.



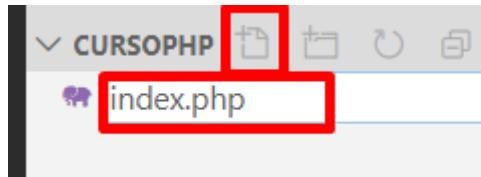
A continuación, en Visual Studio Code abrimos la carpeta creada anteriormente.



Buscamos la carpeta de cursophp y hacemos clic en seleccionar carpeta



A continuación procedemos a crear nuestro primer archivo, lo llamaremos *index.php*. Este nombre es sólo una convención, por tanto el nombre del archivo puede ser el que el desarrollador desee, sin embargo es importante saber que el *index* se reconoce como el punto de acceso inicial a la página web.

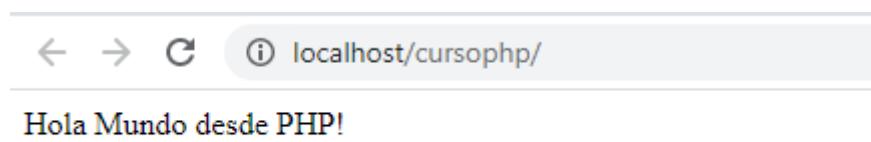


A continuación agregamos la estructura base de un documento HTML, y nuestra primera línea de código de PHP haciendo uso del tag de apertura de código `<?php` y cerrando con `?>`, como se muestra a continuación:

```
index.php < />
index.php
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <title>Clase de PHP</title>
6  </head>
7  <body>
8      <?php
9          echo 'Hola Mundo desde PHP!';
10     ?>
11  </body>
12  </html>
```

Nota: Al ser un curso de PHP con Laravel se obvian detalles de las estructuras HTML y del CSS que se pueda llegar a utilizar dentro de las pruebas. Se toma como un conocimiento previo con el que el estudiante debe llegar.

Finalmente para ver en ejecución nuestra página ingresamos al navegador web a la dirección <http://localhost/cursophp>



Nota: Al colocar el nombre de *index.php* al archivo no es necesario especificarlo en la URL, sin embargo si se le colocó otro nombre se debe agregar a la URL.

3.2. Variables

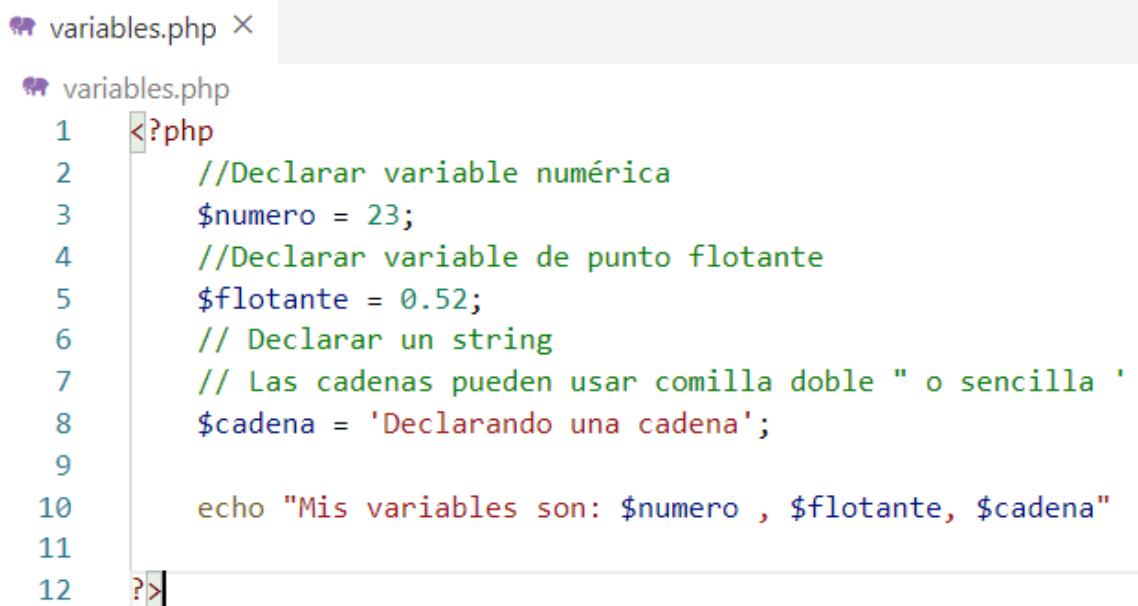
Como todos los lenguajes de programación, en PHP también es posible tener variables, por ello a continuación se explican los tipos de variables disponibles junto a un ejemplo de declaración de variables de diferentes tipos.

Para declarar una variable en PHP se debe iniciar con el símbolo \$ seguido del nombre que se le quiere dar a la variable. Un nombre de variable válido tiene que empezar con una letra o un carácter de subrayado (underscore), seguido de cualquier número de letras, números y caracteres de subrayado. Como expresión regular se podría expresar como: `^[a-zA-Z_]\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'`

Los tipos de datos admitidos en PHP se pueden consultar en su [documentación oficial](#).

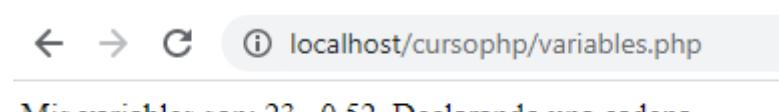
PHP no es estáticamente tipado, es decir que no tenemos que decirle qué tipo de dato es esa variable. Además, es débilmente tipado porque podemos fácilmente cambiar el tipo de dato, es decir PHP ejecuta una conversión de datos interna.

Con la finalidad de ver en la práctica los conceptos anteriormente expuestos crearemos un archivo *variables.php* de la misma manera que creamos el *index.php* en el punto 3.1. y crearemos algunas variables de diferentes tipos de dato.



```
variables.php <input type="text" value="variables.php" style="width: 100%; height: 100%; opacity: 0; border: none; font-size: inherit; color: inherit; background-color: transparent; position: absolute; left: 0; top: 0; z-index: -1;"/>
<?php
    //Declarar variable numérica
    $numero = 23;
    //Declarar variable de punto flotante
    $flotante = 0.52;
    // Declarar un string
    // Las cadenas pueden usar comilla doble " o sencilla '
    $cadena = 'Declarando una cadena';
    echo "Mis variables son: $numero , $flotante, $cadena"
?>
```

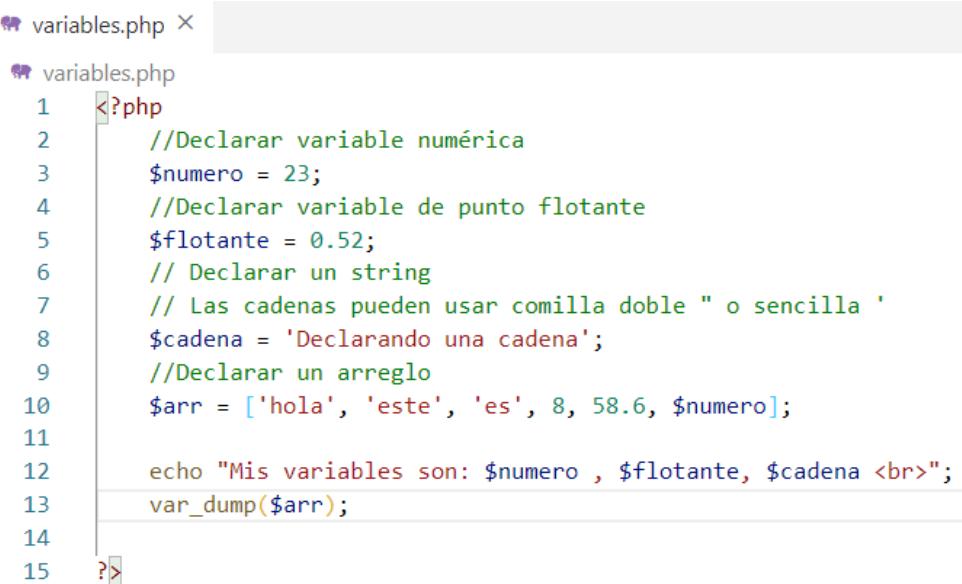
El resultado que se obtiene en el navegador al hacer llamado al archivo variables.php es:



Mis variables son: 23 , 0.52, Declarando una cadena

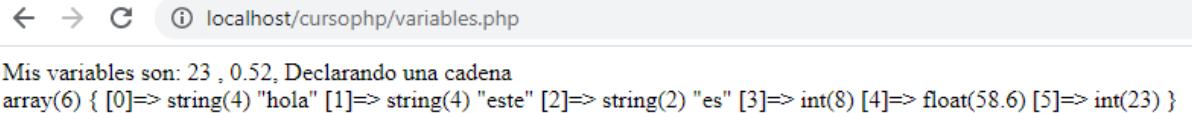
Para más información de las variables, sus tipos de datos y sus posibles concatenaciones referirse a la [documentación oficial](#).

Al momento de trabajar con PHP una cosa muy importante es hacer debugging a nuestras variables, para ello utilizamos la función var_dump(); pasándole por parámetro la variable a revisar.



```
variables.php
1 <?php
2 //Declarar variable numérica
3 $numero = 23;
4 //Declarar variable de punto flotante
5 $flotante = 0.52;
6 // Declarar un string
7 // Las cadenas pueden usar comilla doble " o sencilla '
8 $cadena = 'Declarando una cadena';
9 //Declarar un arreglo
10 $arr = ['hola', 'este', 'es', 8, 58.6, $numero];
11
12 echo "Mis variables son: $numero , $flotante, $cadena <br>";
13 var_dump($arr);
14
15 ?>
```

El resultado en el navegador es el siguiente:



Mis variables son: 23 , 0.52, Declarando una cadena
array(6) { [0]=> string(4) "hola" [1]=> string(4) "este" [2]=> string(2) "es" [3]=> int(8) [4]=> float(58.6) [5]=> int(23) }

En PHP tenemos dos tipos de cadenas, las que son con comillas simples y las de comillas dobles. La diferencia entre estas dos cadenas es que la de comillas simples recibe de forma literal lo que le escribas mientras que la de comillas dobles intenta interpretar cualquier variable dentro de ella.

```

//Declaración con comillas dobles
$nombre = "Carlos";
//Declaración con comillas simples
$apellido = 'Castañeda';
//Mensaje con comillas dobles
echo "Nombre completo: $nombre $apellido <br>";
//Mensaje con comillas simples (Mal construido)
echo 'Nombre completo: $nombre $apellido <br>';
//Mensaje con comillas simples (Bien construido)
echo 'Nombre completo:' . $nombre . ' ' . $apellido . '<br>';

```

Esto nos da como resultado:

```

Nombre completo: Carlos Castañeda
Nombre completo: $nombre $apellido
Nombre completo:Carlos Castañeda

```

En resumen, los tipos de variables se muestran a continuación:

- **Tipos Escalares**

- **boolean**: Representa solamente un valor verdadero o falso.

<http://php.net/manual/es/language.types.boolean.php>

Valores válidos: true (verdadero) false (falso)

```

<?php
$a = true;
$b = false;
?>

```

- **Integer**: Representa un número entero positivo, negativo o 0.

<http://php.net/manual/es/language.types.integer.php>

```

<?php
$a = -123;
$b = 0;
$c = 7763;
?>

```

- **float o double**: Representa un número de punto flotante, existen problemas de precisión con los números flotantes debido a la naturaleza binaria de las computadoras.

<http://php.net/manual/es/language.types.float.php>

```

<?php
$a = 12.24;
$b = 1.5e3;
$c = 7E-10;
?>

```

- **string:** Representa una cadena de caracteres.
 - Existen 4 formas de representar una cadena. Las 2 principales son usando comillas simples o comillas dobles.
 - ---- Usando comillas simples donde el texto será exactamente como se escribe.
 - ---- Usando comillas dobles permite usar caracteres de escape y además expanden los nombres de las variables, es decir sustituye el valor de las variables dentro de las cadenas.
 - – Hay 2 formas adicionales llamadas Heredoc y Nowdoc que sirven para crear cadenas de múltiples líneas.
<https://www.php.net/manual/es/language.types.string.php#language.types.string.details>

```
<?php
$a = "Hola";
$b = 'Mundo';
?>
```

- Tipos Compuestos

- **array:** Representa una colección de valores, aunque por defecto PHP usará índices numéricos, la realidad es que la estructura se representa como un mapa que colecciona pares clave-valor. La sintaxis para definir un arreglo será a partir de corchetes cuadrados, aunque en versiones anteriores de PHP era necesario usar la función array(). Las llaves pueden ser enteros o cadenas y los valores pueden ser de cualquier tipo de PHP, incluso de tipo array.

<http://php.net/manual/es/language.types.array.php>

```
<?php
$array = array(
    "curso1" => "php",
    "curso2" => "js",
);

// a partir de PHP 5.4
$array = [
    "curso1" => "php",
    "curso2" => "js",
];

// índices numéricos
$array = [
    "php",
    "js",
];
?>
```

- **object**: Representa una instancia de una clase.

```
<?php
class Car
{
    function move()
    {
        echo "Going forward...";
    }
}

$myCar = new Car();
$myCar->move();
?>
```

- **callable**: Es un tipo de dato especial que representa a algo que puede ser “llamado”, por ejemplo una función o un método.

```
<?php
// Variable que guarda un callable
$firstFromArray = function(array $array) {
    if (count($array) == 0) { return null; }
    return $array[0];
};

// Este es nuestro arreglo
$values = [3, 2, 1];

// Usamos nuestro callable y se imprime el valor 3
echo $firstFromArray($values);
?>
```

- **iterable**: A partir de PHP 7.1 iterable es un pseudo tipo de datos que puede ser recorrido.

```
<?php

function foo(iterable $iterable) {
    foreach ($iterable as $valor) {
        // ...
    }
}

?>
```

- Tipos Especiales

- **resource**: Es un tipo de dato especial que representa un recurso externo, por ejemplo un archivo externo a tu aplicación.

```
<?php
$res = fopen("c:\\dir\\file.txt", "r");
?>
```

- **NULL**: Es un valor especial que se usa para representar una variable sin valor.

<http://php.net/manual/es/language.types.null.php>

```
<?php  
$a = null;  
?>
```

Para imprimir los valores de las variables y datos adicionales de las mismas tenemos esencialmente cuatro comandos, estos son comparados en el siguiente post:

https://cybmeta.com/php-diferencias-entre-echo-print-print_r-y-var_dump

3.3. Condicionales

Los condiciones son estructuras de control utilizadas para permitir o denegar el permiso de ejecución a un bloque de código. PHP como cualquier lenguaje de programación tiene incorporada dicha funcionalidad y la sintaxis es como se muestra a continuación:

```
<?php  
if ($a > $b) {  
    echo "a es mayor que b";  
} elseif ($a == $b) {  
    echo "a es igual que b";  
} else {  
    echo "a es menor que b";  
}  
?>
```

<https://www.php.net/manual/es/control-structures.if.php>

3.4. Ciclos

- **while**: El significado de una sentencia while es simple. Le dice a PHP que ejecute las sentencias anidadas, tanto como la expresión while se evalúe como TRUE. El valor de la expresión es verificado cada vez al inicio del bucle, por lo que incluso si este valor cambia durante la ejecución de las sentencias anidadas, la ejecución no se detendrá hasta el final de la iteración (cada vez que PHP ejecuta las sentencias

contenidas en el bucle es una iteración). A veces, si la expresión while se evalúa como FALSE desde el principio, las sentencias anidadas no se ejecutarán ni siquiera una vez.

```
<?php
/* ejemplo 1 */

$i = 1;
while ($i <= 10) {
    echo $i++; /* el valor presentado sería
                   $i antes del incremento
                   (post-incremento) */
}
```

- **do - while** : Los bucles do-while son muy similares a los bucles while, excepto que la expresión verdadera es verificada al final de cada iteración en lugar que al principio. La diferencia principal con los bucles while es que está garantizado que corra la primera iteración de un bucle do-while (la expresión verdadera sólo es verificada al final de la iteración), mientras que no necesariamente va a correr con un bucle while regular (la expresión verdadera es verificada al principio de cada iteración, si se evalúa como FALSE justo desde el comienzo, la ejecución del bucle terminaría inmediatamente).

```
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```

<http://php.net/manual/es/control-structures.do.while.php>

- **for**: Los bucles for son los más complejos en PHP. Se comportan como sus homólogos en C. La sintaxis de un bucle for es:

```
for (expr1; expr2; expr3)
    sentencia
```

La primera expresión (*expr1*) es evaluada (ejecutada) una vez incondicionalmente al comienzo del bucle.

En el comienzo de cada iteración, se evalúa *expr2*. Si se evalúa como TRUE, el bucle continúa y se ejecutan la/s sentencia/s anidada/s. Si se evalúa como FALSE, finaliza la ejecución del bucle.

Al final de cada iteración, se evalúa (ejecuta) *expr3*.

```
<?php  
/* ejemplo 1 */  
  
for ($i = 1; $i <= 10; $i++) {  
    echo $i;  
}
```

<https://www.php.net/manual/es/control-structures.for.php>

- **foreach**: El constructor foreach proporciona un modo sencillo de iterar sobre arrays. foreach funciona sólo sobre arrays y objetos, y emitirá un error al intentar usarlo con una variable de un tipo diferente de datos o una variable no inicializada. Existen dos sintaxis:

```
foreach (expresión_array as $valor)  
    sentencias  
foreach (expresión_array as $clave => $valor)  
    sentencias
```

La primera forma recorre el array dado por *expresión_array*. En cada iteración, el valor del elemento actual se asigna a *\$valor* y el puntero interno del array avanza una posición (así en la próxima iteración se estará observando el siguiente elemento).

La segunda forma además asigna la clave del elemento actual a la variable *\$clave* en cada iteración.

```
<?php  
$array = array(1, 2, 3, 4);  
foreach ($array as &$valor) {  
    $valor = $valor * 2;  
}  
// $array ahora es array(2, 4, 6, 8)  
unset($valor); // rompe la referencia con el último elemento  
?>
```

<https://www.php.net/manual/es/control-structures.foreach.php>

Ejercicios de Arreglos

Ejercicio 1.

Escribe el código necesario para generar la cadena final usando el arreglo dado

```
$arreglo = [  
    'keyStr1' => 'lado',  
    0 => 'ledo',  
  
    'keyStr2' => 'lido',  
    1 => 'lodo',  
    2 => 'ludo'  
];
```

Lado, ledo, lido, lodo, ludo,
decirlo al revés lo dudo.
Ludo, lodo, lido, ledo, lado,
¡Qué trabajo me ha costado!

Ejercicio 2.

Crea un arreglo que contenga como clave los nombres de 5 países y como valor otro arreglo con 3 ciudades que pertenezcan a ese país, después utiliza un ciclo foreach, para imprimir el nombre del país seguido de las ciudades que definiste:

Ejemplo,

México: Monterrey Querétaro Guadalajara

Colombia: Bogota Cartagena Medellin

Ejercicio 3.

Escribe el código necesario para encontrar los 3 números más grandes y los 3 números más bajos de la siguiente lista:

```
$valores = [23, 54, 32, 67, 34, 78, 98, 56, 21, 34, 57, 92, 12, 5, 61]
```

3.5. Operadores

Un operador es un símbolo que toma uno más valores (o expresiones, en la jerga de programación) y produce otro valor (de modo que la construcción en sí misma se convierte en una expresión).

Los operadores se pueden agrupar de acuerdo con el número de valores que toman. Los operadores unarios toman sólo un valor, por ejemplo ! (el [operador lógico de negación](#)) o ++ (el [operador de incremento](#)). Los operadores binarios

toman dos valores, como los familiares operadores aritméticos + (suma) y - (resta), y la mayoría de los operadores de PHP entran en esta categoría. Finalmente, hay sólo un operador ternario, ? :, el cual toma tres valores; usualmente a este se le refiere simplemente como "el operador ternario" (aunque podría tal vez llamarse más correctamente como el operador condicional).

<https://www.php.net/manual/es/language.operators.php>

La precedencia de operadores en PHP se puede encontrar en el siguiente enlace:
<https://www.php.net/manual/es/language.operators.precedence.php>

- **Aritméticos:**

Funcionan para realizar operaciones aritméticas.

<http://php.net/manual/es/language.operators.arithmetic.php>

Ejemplo	Nombre	Resultado
+\$a	Identidad	Conversión de \$a a int o float según el caso.
-\$a	Negación	Opuesto de \$a.
\$a + \$b	Adición	Suma de \$a y \$b.
\$a - \$b	Sustracción	Diferencia de \$a y \$b.
\$a * \$b	Multiplicación	Producto de \$a y \$b.
\$a / \$b	División	Cociente de \$a y \$b.
\$a % \$b	Módulo	Resto de \$a dividido por \$b.
\$a ** \$b	Exponenciación	Resultado de elevar \$a a la potencia \$bésima. Introducido en PHP 5.6.

- **Asignación:**

El operador principal de asignación es el símbolo = (igual). Es importante tener en cuenta que este operador no sirve para comparar, normalmente del lado izquierdo del operador tendremos una variable, y este operador sirve para asignar el resultado de lo que se encuentre a la derecha a dicha variable.

```
$variable = 5;
```

Lo que tenemos en la derecha puede ser un valor, otra variable, o el resultado de una operación o función.

También existen otros operadores de asignación que se combinan con operadores aritméticos o para strings y nos permiten simplificar algunas sentencias dentro de PHP. Estos son ejemplos de cómo funcionan:

```
$a = 23;  
$b = 19;  
$a += $b;  
$a = $a + $b;  
  
$a -= $b;  
$a = $a - $b;  
  
$a *= $b;  
$a = $a * $b;  
  
$a /= $b;  
$a = $a / $b;  
  
$a %= $b;  
$a = $a % $b;  
  
$a .= $b;  
$a = $a . $b;
```

- **Condicionales.**

Nos permiten comparar valores.

<http://php.net/manual/es/language.operators.comparison.php>

Ejemplo	Nombre	Resultado
<code>\$a == \$b</code>	Igual	TRUE si \$a es igual a \$b después de la manipulación de tipos.
<code>\$a === \$b</code>	Idéntico	TRUE si \$a es igual a \$b, y son del mismo tipo.
<code>\$a != \$b</code>	Diferente	TRUE si \$a no es igual a \$b después de la manipulación de tipos.
<code>\$a <> \$b</code>	Diferente	TRUE si \$a no es igual a \$b después de la manipulación de tipos.
<code>\$a !== \$b</code>	No idéntico	TRUE si \$a no es igual a \$b, o si no son del mismo tipo.
<code>\$a < \$b</code>	Menor que	TRUE si \$a es estrictamente menor que \$b.
<code>\$a > \$b</code>	Mayor que	TRUE si \$a es estrictamente mayor que \$b.
<code>\$a <= \$b</code>	Menor o igual que	TRUE si \$a es menor o igual que \$b.
<code>\$a >= \$b</code>	Mayor o igual que	TRUE si \$a es mayor o igual que \$b.
<code>\$a <=> \$b</code>	Nave espacial	Un integer menor que, igual a, o mayor que cero cuando \$a es respectivamente menor que, igual a, o mayor que \$b. Disponible a partir de PHP 7.
<code>\$a ?? \$b ?? \$c</code>	Fusión de null	El primer operando de izquierda a derecha que exista y no sea NULL. NULL si no hay valores definidos y no son NULL. Disponible a partir de PHP 7.

- **Incremento.**

Permiten incrementar o decrementar un valor en 1.

<http://php.net/manual/es/language.operators.increment.php>

Ejemplo	Nombre	Efecto
<code>++\$a</code>	Pre-incremento	Incrementa \$a en uno, y luego retorna \$a.
<code>\$a++</code>	Post-incremento	Retorna \$a, y luego incrementa \$a en uno.
<code>--\$a</code>	Pre-decremento	Decrementa \$a en uno, luego retorna \$a.
<code>\$a--</code>	Post-decremento	Retorna \$a, luego decrementa \$a en uno.

- **Operadores Lógicos:**

Nos permiten combinar resultados de comparaciones.

<http://php.net/manual/es/language.operators.logical.php>

Ejemplo	Nombre	Resultado
\$a and \$b	And (y)	TRUE si tanto \$a como \$b son TRUE.
\$a or \$b	Or (o inclusivo)	TRUE si cualquiera de \$a o \$b es TRUE.
\$a xor \$b	Xor (o exclusivo)	TRUE si \$a o \$b es TRUE, pero no ambos.
! \$a	Not (no)	TRUE si \$a no es TRUE.
\$a && \$b	And (y)	TRUE si tanto \$a como \$b son TRUE.
\$a \$b	Or (o inclusivo)	TRUE si cualquiera de \$a o \$b es TRUE.

- **Operadores de Cadenas (strings):**

Existen 2 operadores para strings el . (punto) que nos permite concatenar cadenas, y el .= que ya fue visto anteriormente y nos permite simplificar la sintaxis de concatenar algo a una misma cadena, ejemplo:

```
$var1 = 'Hola ' . ' php';
$var1 .= '!!!';
echo $var1;
```

Imprime

```
Hola php!!!
```

- **Operadores para Arrays:**

<http://php.net/manual/es/language.operators.array.php>

Ejemplo	Nombre	Resultado
\$a + \$b	Unión	Unión de \$a y \$b.
\$a == \$b	Igualdad	TRUE si \$a i \$b tienen las mismas parejas clave/valor.
\$a === \$b	Identidad	TRUE si \$a y \$b tienen las mismas parejas clave/valor en el mismo orden y de los mismos tipos.
\$a != \$b	Desigualdad	TRUE si \$a no es igual a \$b.
\$a <> \$b	Desigualdad	TRUE si \$a no es igual a \$b.
\$a !== \$b	No-identidad	TRUE si \$a no es idéntica a \$b.

Nota: La precedencia de operaciones está presente en PHP por lo que es importante validar el orden de escritura de nuestras operaciones. Por ejemplo, en la sentencia:

1 + 2 * 3

Se ejecutará primero la operación $2 * 3$

Luego se ejecutará la suma con 1

Esto es debido a que $*$ tiene más valor en la precedencia que el $+$.

Una forma sencilla de controlar la precedencia es utilizando () paréntesis, de esta forma podemos forzar el orden que nosotros queramos, por ejemplo $(1 + 2) * 3$ será una versión diferente y se ejecutará primero la suma y luego la multiplicación.

Más información:

<http://php.net/manual/es/language.operators.precedence.php>

3.6. Funciones

Una función es una parte de un programa (subrutina) con un nombre, que puede ser invocada (llamada a ejecución) desde otras partes tantas veces como se deseé. Un bloque de código que puede ser ejecutado como una unidad funcional. Opcionalmente puede recibir valores; se ejecuta y puede devolver un valor.

Las funciones en PHP se denotan por la palabra reservada function seguida por el nombre de la función, las funciones nos servirán para llamar y reutilizar código en nuestros proyectos.

Cuando trabajemos con funciones es muy importante cuidar el scope (alcance) de las variables pues, algunas podrían entrar en su scope y otras no.

Las funciones en PHP pueden o no retornar un dato particular. Si deseamos hacerlo podemos indicarlo con la palabra reservada return.

<https://www.php.net/manual/es/functions.user-defined.php>

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Función de ejemplo.\n";
    return $valor_devuelto;
}
?>
```

3.6. Archivos externos

A medida que la aplicación crece, se hace necesario separar las responsabilidades y archivos, por lo que debemos agregar archivos externos a nuestro código. Para ello tenemos esencialmente 4 métodos:

- **include**: Incluyelo si lo encuentras, si no no WARNING. <https://www.php.net/manual/es/function.include.php>
- **require**: Incluyelo si lo encuentras, si no se para la ejecución del programa. <https://www.php.net/manual/es/function.require.php>

Los métodos include y require ejecutan el código del archivo cada vez que lo incluyen, esto puede traer errores en la ejecución de tu código si tienes archivos con funciones pues te dirá que no puedes declarar dos veces una función con el mismo nombre. Para resolver esto existen **include_once** y **require_once** que obligan a incluir una sola vez el archivo.

Ejemplo:

1. Crear un archivo PHP llamado `externo.php` en el cual se declara una variable y una función como se muestra a continuación:

```
externo.php
externo.php > ...
1  <?php
2  $arrMaterias = [
3      1 => [
4          'Fundamentos de programación',
5          'Matemáticas Fundamentales'
6      ],
7      2 => [
8          'Programación Orientada a Objetos',
9          'Algebra Lineal'
10     ],
11     3 => [
12         'Programación III',
13         'Calculo I'
14     ]
15 ];
16
17 function mostrarArregloMaterias($arr){
18     foreach($arr as $semestre=>$materias){
19         echo "Semestre $semestre: <br><ul>";
20         foreach($materias as $materia){
21             echo "<li> $materia </li>";
22         }
23         echo "</ul> <br>";
24     }
}
```

2. En otro archivo de PHP que se encuentre al mismo nivel de carpeta que `externo.php` se deben declarar las siguientes sentencias:

```
index.php
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4     <meta charset="UTF-8">
5     <title>Curso de PHP</title>
6 </head>
7 <body>
8     <h1>Curso de PHP</h1>
9     <?php
10    include('externo.php');
11    mostrarArregloMaterias($arrMaterias);
12    ?>
13 </body>
14 </html>
```

Obteniendo el siguiente resultado en el navegador:

← → ⌂ ⓘ localhost/cursophp/

Curso de PHP

Semestre 1:

- Fundamentos de programación
- Matemáticas Fundamentales

Semestre 2:

- Programación Orientada a Objetos
- Algebra Lineal

Semestre 3:

- Programación III
- Calculo I

3.7. Programación Orientada a Objetos (POO)

La programación orientada a objetos nos ayudará a estructurar mejor nuestros programas. PHP a partir de su versión 5 tiene implementaciones orientadas a objetos, lo que lo hace tener código más reutilizable y mantenable.

Una **clase** es una plantilla o definición de algo. Y una **instancia** es la representación concreta de la clase.

Encapsulamiento será el nivel de visibilidad que queramos darle a alguna variable, para ello podemos utilizar los modificadores de acceso private, public y protected.

Con la palabra reservada this estaremos haciendo referencia a la variable que pertenece a la clase.

<https://www.php.net/manual/es/language.oop5.php>

3.7.1. Clase

La definición básica de una clase comienza con la palabra reservada *class*, seguida de un nombre de clase, y continuando con un par de llaves que encierran las definiciones de las propiedades y métodos pertenecientes a dicha clase.

El nombre de clase puede ser cualquier etiqueta válida, siempre que no sea una palabra reservada de PHP. Un nombre válido de clase comienza con una letra o un guión bajo, seguido de una cantidad arbitraria de letras, números o guiones bajos. Como expresión regular, se expresaría de la siguiente forma: ^[a-zA-Z_]\x7f-\xff][a-zA-Z0-9_]\x7f-\xff]*\$.

Una clase puede tener sus propias constantes, variables (llamadas "propiedades"), y funciones (llamados "métodos").

```
<?php
class ClaseSencilla
{
    // Declaración de una propiedad
    public $var = 'un valor predeterminado';

    // Declaración de un método
    public function mostrarVar() {
        echo $this->var;
    }
?>
```

La pseudo variable *\$this* está disponible cuando un método es invocado dentro del contexto de un objeto. *\$this* es una referencia al objeto invocador (usualmente el objeto al cual el método pertenece, aunque puede que sea

otro objeto si el método es llamado estáticamente desde el contexto de un objeto secundario). A partir de PHP 7.0.0, la llamada estática a un método no estático desde un contexto incompatible resulta en que \$this no esté definido dentro del método. Una llamada estática a un método no estático desde un contexto no compatible está obsoleta desde PHP 5.6.0. A partir de PHP 7.0.0, una llamada estática a un método no estático está obsoleta en general (incluso si se llama desde un contexto compatible). Antes de PHP 5.6.0, tales llamadas ya ocasionaron un aviso de estricto.

- **Constructor:**

PHP 5 permite a los desarrolladores declarar métodos constructores para las clases. Aquellas que tengan un método constructor lo invocarán en cada nuevo objeto creado, lo que lo hace idóneo para cualquier inicialización que el objeto pueda necesitar antes de ser usado.

```
<?php
class BaseClass {
    function __construct() {
        print "En el constructor BaseClass\n";
    }
}
```

Todas las funciones que tienen __ antes del nombre de la función se conocen como métodos mágicos

<https://www.php.net/manual/es/language.oop5.magic.php>

- **Métodos:**

La sintaxis de los métodos es igual a la de las funciones vista en secciones anteriores, sin embargo lo importante a tener en cuenta de estos es que pueden acceder a las variables globales definidas dentro de la clase a través de la palabra reservada \$this. Ejemplo:

```
public function getDurationAsString() {
    $years = floor($this->months / 12);
    $extraMonths = $this->months % 12;
}

return "$years years $extraMonths months";
}
```

- **Herencia y Polimorfismo:**

La herencia permite que ciertas clases tengan características de una clase padre. Esta clase se llamará hijo.

Como una buena práctica en PHP lo mejor es tener dividido el código en diferentes archivos.

Es muy conveniente utilizar `require_once` cuando queremos utilizar herencia e incluir clases que están en otros archivos.

Dentro de nuestra clase hijo podemos sobreescribir algún método del padre, esto es un concepto que conocemos como polimorfismo. Lo que polimorfismo quiere decir es que tendremos un método que va a funcionar de acuerdo con su contexto donde es llamado.

Si tenemos propiedades con la palabra `private` en nuestra clase padre, desde nuestra clase hija no podremos acceder a esta propiedad, pero si queremos que siga siendo privada y que las clases hijas tengan acceso podemos usar la palabra clave `protected`.

```
<?php  
  
class Job extends BaseElement {  
}  
}
```

<https://www.php.net/manual/es/language.oop5.inheritance.php>

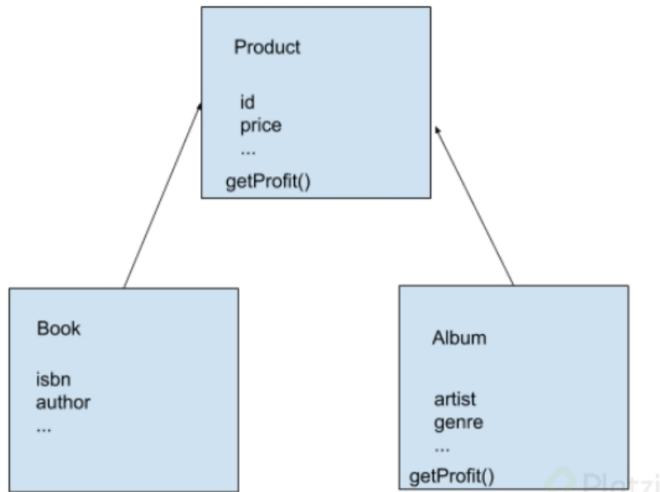
Cuando trabajamos con objetos y clases, algunas veces podemos encontrarnos con clases que son muy similares, incluso que comparten algunos métodos o propiedades, pero que no son completamente iguales.

En este punto hablaremos de la Herencia, un concepto que nos permitirá reutilizar todo las partes que son comunes y nos permitirá tener lo que no es común en clases separadas.

La Herencia funciona como una cadena de herencia, es decir podemos tener una clase y generar una “clase hija” a partir de ella, la clase “hija” reutilizara todas las propiedades y métodos de la clase

“padre” y además le permitirá implementar esas partes que la hacen diferente.

Por ejemplo, pensemos que estamos construyendo un sistema de comercio electrónico que maneja libros digitales y álbumes musicales, para esto podríamos generar una cadena de herencia como la siguiente:



Product(id, title, price, description)

Book(isbn, publisher, author, pages, profitBonus) extends Product

Album(company, artist, duration, genre) extends Product

En este ejemplo, un libro es diferente a un álbum en algunas cosas, sin embargo existen ciertas propiedades que se comparten a través de la clase padre Product, de este modo ambas clases comparten las propiedades y métodos de Product pero además de eso implementan propiedades y métodos únicos.

Ahora vamos a hablar de un concepto adicional, el cual también es muy importante, el término es polimorfismo y significa “muchas formas”.

Vamos a pensar que queremos calcular la ganancia que obtendremos de la venta de ciertos productos, y en este caso los libros y los álbumes manejan diferentes porcentajes de ganancia, si generamos un método getProfit en la clase Product este método podría definir cuánto ganaremos de cada producto. Por ejemplo, pensamos que ganamos 10%.

```
public function getProfit() {  
    return $price * 0.1;  
}
```

El agregar este método dentro de Product nos permitirá usarlo en objetos de la clase Product y también en objetos basados en las clases hijas de Product, ahora, vamos a pensar que los libros manejan una fórmula diferente porque maneja un valor de bonus adicional, en este caso podríamos tener el método getProfit pero ahora declarado dentro de la clase Book y utilizando la lógica única de esta clase:

```
public function getProfit() {  
    return $price * (0.1 + $this->profitBonus);  
}
```

Este concepto es un tipo de polimorfismo el cual llamamos Sobreescritura y lo que nos permite es reemplazar algo que ya estaba definido en una clase padre.

Un ejemplo de uso para esta cadena de herencia es, por ejemplo, si tenemos una lista de productos, algunos de ellos son libros y otros álbumes, y si queremos saber las ganancias totales, simplemente tenemos que recorrer los elementos e ir sumando el resultado del método getProfit y en cada caso el objeto sabrá cuál fórmula utilizar porque está definida dentro de su clase.

En resumen la herencia nos permite reutilizar código entre nuestras clases y el polimorfismo, en este ejemplo la sobreescritura, nos ayudará a que las clases puedan reaccionar de una manera diferente a métodos con el mismo nombre.

- **Interfaces:**

Las interfaces se pueden ver como un contrato o un acuerdo en el que se pueden estandarizar ciertas cosas.

La palabra reservada que utilizaremos para declarar una interfaz será `interface`. Y la que nos indicará que estamos usando una interfaz en una clase será `implements`.

Usando Type Hinting estableceremos el tipo de dato que esperamos ya sea una clase o un tipo de dato específico.

La herencia en PHP será de forma sencilla es decir solo que podrá hacer herencia de una sola clase, por lo contrario, con las interfaces que sí podemos implementar varias al mismo tiempo.

```
<?php

// Declarar la interfaz 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Implementar la interfaz
// Ésto funcionará
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }
}
```

<https://www.php.net/manual/es/language.oop5.interfaces.php>

- **Namespaces:**

Esta es una forma de mantener únicos los nombres de los archivos en el mismo directorio.

Esto nos permite tener mejor organizado el proyecto.

Para declarar un espacio de nombres privado se utiliza la palabra reservada `namespace`.

<https://www.php.net/manual/es/language.namespaces.basics.php>

Como buena práctica a la hora de escribir código PHP se recomienda que los nombres de espacios correspondan a las carpetas donde se encuentran alojados nuestros archivos.

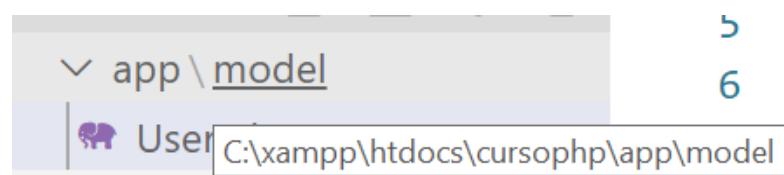
Ahora para utilizar en archivos externos las clases de PHP definidas dentro de un NameSpace debemos hacer uso de la palabra reservada `use`.

Ejemplo:

1. Dentro del proyecto, crear la siguiente estructura de carpetas:



Que se verá visualizada en Visual Studio Code de la siguiente manera:



Nota: Dependiendo de la versión del editor de texto instalada la interfaz puede variar pero ambos nombres de las carpetas se deben visualizar.

2. Crear dentro de la carpeta model el archivo User.php, a continuación crear la clase User y agregar el namespace App\Models como se muestra a continuación:

```

EXPLORER
OPEN EDITORS
CURSOPHP
app\ model
User.php

User.php X
app > model > User.php > User
1 <?php
2
3 namespace App\Models;
4
5 class User {
6     private $email;
7
8     function getEmail(){
9         return $this->email;
10    }
11
12     function setEmail($newEmail){
13         $this->email = $newEmail;
14    }
15

```

3. Para crear una instancia del objeto anteriormente creado usando el namespace asignado, en el archivo index.php agregamos las siguientes líneas de código:

```

index.php ●
index.php

<body>
    <h1>Curso de PHP</h1>
    <?php
        // include('externo.php');
        // mostrarArregloMaterias($arrMaterias);

        use App\Models\User;
        require_once('app/model/User.php');

        $user = new User;
        $user->setEmail('carlos.castaneda@ucaldas.edu.co');

        echo "The user email is: " . $user->getEmail();
    ?>
</body>
</html>

```

4. PSR (PHP Standard Recommendation)

Una comunidad de programadores de PHP unieron fuerzas para crear algunas recomendaciones estándar en proyectos de PHP ya que al ser un lenguaje ampliamente utilizado es muy probable que se comparan funcionalidades de lenguaje que todos los programadores deben llevar a

cabo, para exponer sus resultados crearon [PHP-FIG](#), página web que muestra uno a uno los estándares con su respectiva explicación.

Uno de estándar es el PSR-4, en este se facilita la autocarga de clases con la finalidad de que no tengamos que enlazar todos los archivos de PHP a través de la instrucción require si no que automáticamente al cargar el aplicativo todos los archivos de PHP sean cargados y queden listos para su uso. El ejemplo práctico de la puesta en marcha del estándar se mostrará en el siguiente capítulo usando el gestor de dependencias de PHP llamado composer.

5. Composer

5.1. Instalación de Composer

El primer paso a realizar es instalar composer en el sistema operativo, para ello nos dirigimos a la [página oficial](#) y hacemos clic en la sección de descargas.

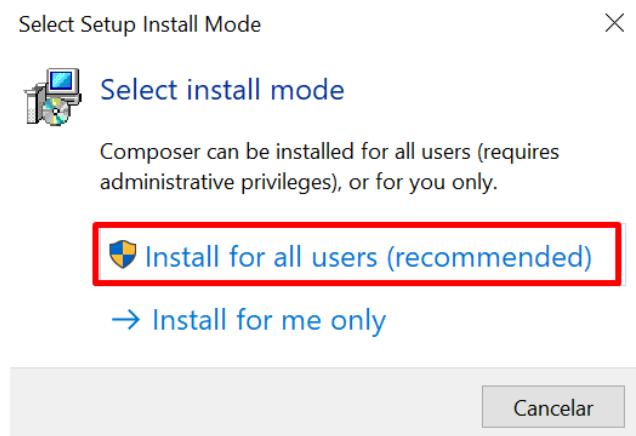
The screenshot shows the official Composer website. At the top center is a cartoon illustration of a person with brown hair, wearing a white shirt, looking upwards with arms raised. Below the illustration, the word "COMPOSER" is written in large, bold, black capital letters. Underneath the title, the text "A Dependency Manager for PHP" is displayed. A small note indicates "Latest: 1.9.3". Below this, there are several buttons arranged in a grid-like pattern. The "Download" button is highlighted with a thick red border. Other visible buttons include "Getting Started", "Documentation", "Browse Packages", "Issues", and "GitHub".

[Download Composer](#) Latest: v1.9.3

Windows Installer

The installer will download composer for you and set up your PATH environment variable so you can simply call `composer` from any directory.

Download and run [Composer-Setup.exe](#) - it will install the latest composer version whenever it is executed.



En los pasos de instalación no es necesario realizar configuraciones adicionales, en caso de que el instalar pregunte si se desea agregar composer al PATH se debe aceptar para usar composer desde la línea de comandos. Para verificar que la instalación de composer se haya realizado correctamente abrimos la línea de comandos de windows y ejecutamos el comando *composer*, la salida debe ser como se muestra a continuación:

5.2. Usando Composer

El uso de composer inicia con su archivo de configuración `composer.json` en donde agregamos las dependencias requeridas en el proyecto. Inicialmente crearemos el archivo de configuración manualmente en visual studio code en la base de directorios del proyecto. En nuestro caso lo que haremos será implementar el uso de [PSR-4](#) para el auto cargado de las clases.

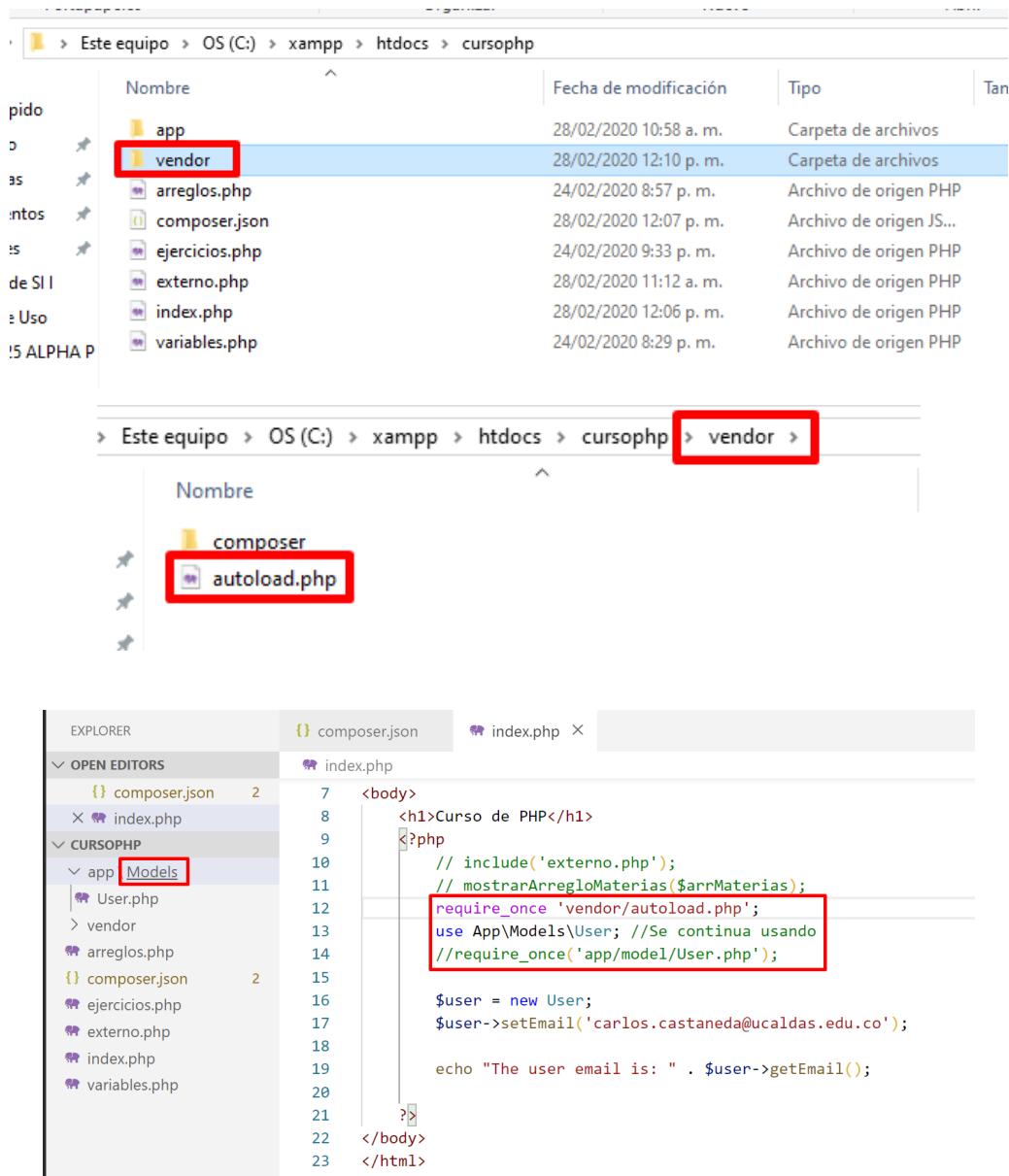
The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project structure under 'CURSOPHP' containing files like 'app\\model', 'arreglos.php', 'ejercicios.php', 'externo.php', 'index.php', and 'variables.php'. A file named 'composer.json' is selected and highlighted with a red border. On the right, the code editor displays the contents of 'composer.json':

```
1 "autoload": {  
2     "psr-4": {  
3         "App\\": "app/"  
4     }  
5 },  
6 "require": {}  
7  
8 }  
9  
10
```

The screenshot shows a terminal window with the following command and output:

```
C:\xampp\htdocs\cursophp>cd C:\xampp\htdocs\cursophp [redacted]  
C:\xampp\htdocs\cursophp>dir  
El volumen de la unidad C es OS  
El n mero de serie del volumen es: AA61-60FD  
  
Directorio de C:\xampp\htdocs\cursophp  
  
28/02/2020 11:49 a. m. <DIR> .  
28/02/2020 11:49 a. m. <DIR> ..  
28/02/2020 10:58 a. m. <DIR> app  
24/02/2020 08:57 p. m. 752 arreglos.php  
28/02/2020 12:07 p. m. 118 composer.json [redacted]  
24/02/2020 09:33 p. m. 1.104 ejercicios.php  
28/02/2020 11:12 a. m. 552 externo.php  
28/02/2020 12:06 p. m. 498 index.php  
24/02/2020 08:29 p. m. 879 variables.php  
6 archivos 3.903 bytes  
3 dirs 138.332.819.456 bytes libres  
  
C:\xampp\htdocs\cursophp>composer install [redacted]  
Loading composer repositories with package information  
Updating dependencies (including require-dev)  
Nothing to install or update  
Generating autoload files
```

Luego de ejecutar el comando de instalación se crea automáticamente una carpeta llamada *vendor* que es donde composer almacenará todo lo que necesita para prestar el servicio de gestor de dependencias. Dentro de la carpeta se encuentra el archivo *autoload.php* que será el que nosotros referenciamos en nuestro proyecto para hacer uso de composer:



Importante: Para la creación del proyecto teníamos la carpeta *model*, para que funcione la autocarga es necesario que el nombre de la carpeta sea igual al del *namespace* por lo que debemos cambiar de *model* a *Models*.

Con los cambios en el código realizados ya se implementó el estándar **PSR-4** en donde a partir de este momento para carga de archivos externos no tendremos que hacer uso de las instrucciones *require* o *include* para los

archivos propios, si no que únicamente usamos el require a composer y este se encarga de dejar disponibles los archivos en nuestro proyecto.

5.3. Repositorio de Librerías de PHP

Composer es el gestor de dependencias que nos permite instalar las librerías de terceros en nuestros proyectos sin tener que preocuparnos por las dependencias adicionales que esta tenga, sin embargo nos preguntaremos ¿Dónde puedo encontrar las librerías disponibles de PHP?.

Aunque no son todas, ya que no es obligatorio subir las librerías al repositorio, [Packagist](#) incluye un gran repertorio de librerías, en donde podremos encontrar librerías específicas para aquello que busquemos. La interfaz de dicho repositorio se puede apreciar en la siguiente imagen:

The screenshot shows the homepage of Packagist, "The PHP Package Repository". At the top, there's a navigation bar with links for "Browse", "Submit", "Create account", and "Sign in". Below the header is a search bar with the placeholder "Search packages...". A small icon of a dog wearing a developer's hat is positioned next to the search bar. The main content area is divided into two main sections: "Getting Started" on the left and "Publishing Packages" on the right.

Getting Started

- Define Your Dependencies**: Instructions to put a file named `composer.json` at the root of your project, containing your project dependencies. It includes a code snippet:

```
{  
    "require": {  
        "vendor/package": "1.3.2",  
        "vendor/package2": "1.4.0",  
        "vendor/package3": "~2.0.3"  
    }  
}
```
- Install Composer In Your Project**: Instructions to run `curl -sS https://getcomposer.org/installer | php`. It also provides a link to download `composer.phar` and a note about reading the full documentation for installation instructions on various platforms.
- Install Dependencies**: Instructions to execute `php composer.phar install`.
- Autoload Dependencies**: Instructions to add code to autoload all the dependencies by adding this to your code:

```
require 'vendor/autoload.php';
```
- Browse**: A link to find more great libraries you can use in your project.

Publishing Packages

- Define Your Package**: Instructions to put a file named `composer.json` at the root of your package's repository, containing this information. It includes a code snippet:

```
{  
    "name": "your-vendor-name/package-name",  
    "description": "A short description of what your package does",  
    "require": {  
        "7.2",  
        "another-vendor/package": "1.4"  
    }  
}
```
- Commit The File**: Instructions to add the `composer.json` to your git or other VCS repository and commit it.
- Publish It**: Instructions to log in or register on the site, then hit the `submit` button in the menu. It notes that once entered, the public repository URL will be automatically crawled periodically.
- Sharing Private Code**: Instructions to use Private Packagist if you want to share private code as a Composer package with colleagues or customers without publishing it for everyone on Packagist.org. It mentions that Packagist allows you to manage your own private Composer repository with per-user authentication, team management and integration in version control systems.

5.4. Usando una librería de Packagist

Con la finalidad de probar la facilidad que nos brinda composer, vamos a instalar una de sus librerías y a continuación utilizarla para crear un documento en PDF. Lo primero que haremos será elegir la librería que queremos usar, para ello a través de la barra de navegación de la parte superior escribimos las palabras clave de la funcionalidad requerida como se puede apreciar a continuación:

The screenshot shows the Packagist website interface. At the top, there's a navigation bar with links for 'Browse', 'Submit', 'Create account', and 'Sign in'. Below the navigation is a search bar with the word 'pdf' typed into it. A small icon of a dog is positioned next to the search bar. The main content area displays a list of PHP packages related to PDF generation:

Package	Type	Downloads	Rating
dompdf/dompdf	PHP	19 020 989	★ 6 809
tecnickcom/tcpdf	PHP	11 909 503	★ 2 429
phpoffice/phpword	PHP	3 127 085	★ 4 892
mpdf/mpdf	PHP	8 788 190	★ 2 623

On the right side of the page, there are two sections: 'Package type' and 'Tags', both listing various PHP modules.

Luego de obtener la lista de resultados de las librerías que cumplen con nuestro criterio de búsqueda, elegimos una de las opciones basados en la cantidad de descargar y las puntuaciones recibidas por los usuarios que la han utilizado. En nuestro caso usaremos la librería *mpdf/mpdf* para crear nuestro PDF.

Al ingresar a la librería debería mostrarse una página como la siguiente:

The screenshot shows the detailed view of the *mpdf/mpdf* package on Packagist. At the top, there's a header with the package name and a 'Browse' link. Below the header is a search bar containing the text 'Search packages...'. The main content area features the package name 'mpdf/mpdf' in large blue text, followed by a brief description: 'PHP library generating PDF files from UTF-8 encoded HTML'. There's also a note indicating the package can be required via Composer.

La mayoría de las librerías tienen sus repositorios oficiales en github y desde packagist es posible ir a su documentación oficial.

Para instalar la librería en nuestro proyecto usaremos el comando composer requiere como se muestra a continuación:

```
composer require mpdf/mpdf
```

Obteniendo un resultado como el que se muestra a continuación:

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/cursophpcopy
$ composer require mpdf/mpdf
Using version ^8.0 for mpdf/mpdf
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 5 installs, 0 updates, 0 removals
- Installing myclabs/deep-copy (1.9.5): Downloading (100%)
- Installing paragonie/random_compat (v9.99.99): Downloading (100%)
- Installing setasign/fpdi (v2.3.0): Downloading (100%)
- Installing psr/log (1.1.2): Loading from cache
- Installing mpdf/mpdf (v8.0.5): Downloading (100%)
paragonie/random_compat suggests installing ext-libodium (Provides a
setasign/fpdi suggests installing setasign/fpdf (FPDI will extend this
setasign/fpdi suggests installing setasign/fpdi-fpdf (Use this package
setasign/fpdi suggests installing setasign/fpdi-tcpdf (Use this package
setasign/fpdi suggests installing setasign/fpdi-tfpdf (Use this package
Writing lock file
Generating autoload files
```

Con esto ya tendremos la librería disponible en nuestro proyecto siempre y cuando sea requerido `autoload.php` que se encuentra ubicado en la carpeta vendor.

Siguiendo la documentación de mpdf crearemos el siguiente archivo de PHP llamado `GenerarPDF.php`, siguiendo la estructura básica con la finalidad de que cada vez que sea ejecutado dicho archivo se cree un archivo PDF con información por defecto.

```
GENERARPDF.PHP X
GenerarPDF.php
1 <?php
2
3 require_once __DIR__ . '/vendor/autoload.php';
4
5 $mpdf = new \Mpdf\Mpdf();
6 $mpdf->WriteHTML('<h1>Hola, Así de sencillo fue crear un PDF en PHP!</h1>', 7 | | | | <p> Todo usando <code>Composer</code> y la librería MPDF', 8 $mpdf->Output('miPDFNuevo.pdf');
9
10 echo 'PDF creado exitosamente';
```

A continuación ejecutamos la dirección URL que nos lleva a el archivo de PHP recién creado para validar si el PDF es creado.

localhost/cursophpcopy/GenerarPDF.php

PDF creado exitosamente

Por último se verifica si el PDF fue creado y cuál fue el contenido con el que quedó:

Este equipo > OS (C:) > xampp > htdocs > cursophpcopy

	Nombre	Fecha de mod
lo	app	9/03/2020 2:52
os	vendor	9/03/2020 3:01
os	arreglos.php	24/02/2020 8:5
r	composer.json	9/03/2020 3:00
r	composer.lock	9/03/2020 3:01
r	ejercicios.php	2/03/2020 6:57
ALPHA P	funciones.php	2/03/2020 7:37
ALPHA P	GenerarPDF.php	9/03/2020 3:08
ud Files	index.php	2/03/2020 9:24
	miPDFNuevo.pdf	9/03/2020 3:10
ud Files	operadores.php	2/03/2020 7:18
ud Files	variables.php	24/02/2020 8:2

Archivo | C:/xampp/htdocs/cursophpcopy/miPDFNuevo.pdf

miPDFNuevo.pdf

1 / 1

Hola, Así de sencillo fue crear un PDF en PHP!

Todo usando Composer y la librería MPDF

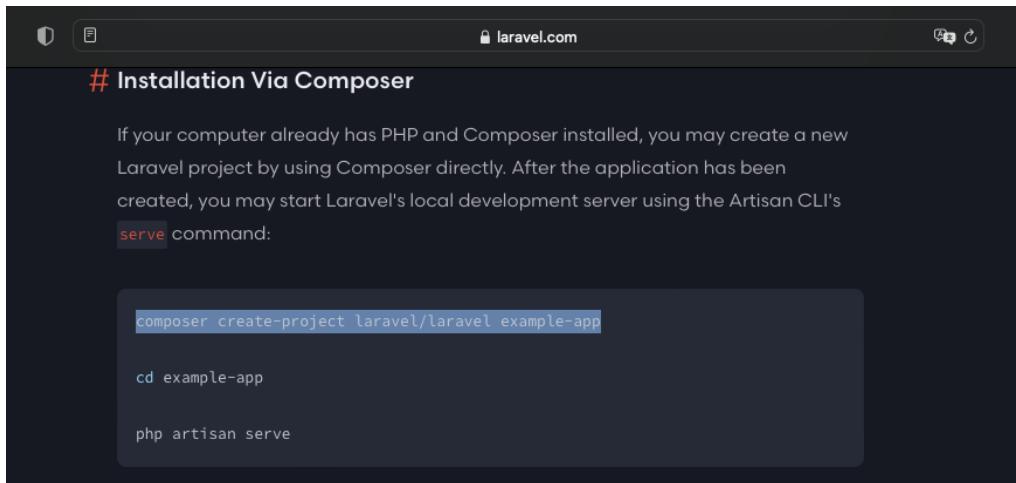
Con esto queda demostrado que a través de composer es posible usar librerías de externos que nos facilitan labores ya reconocidas como creación de archivos en diferentes formatos, envío de email, carga de variables de entorno entre otros.

6. Laravel !

Con las bases de PHP claras es momento de iniciar con Laravel, el framework de PHP con la comunidad más grande ampliamente utilizado en el mundo.

6.1. Instalando Laravel

Para realizar la instalación de Laravel utilizaremos el gestor de paquetes Composer (En caso de no haber visto composer o no tenerlo instalado puede regresar a la sección 5 de este documento). El primer paso será acceder al [sitio oficial de Laravel](#) en donde encontraremos las instrucciones precisas para la instalación del framework.



The screenshot shows a dark-themed web browser window with the URL 'laravel.com' in the address bar. The main content area displays the 'Installation Via Composer' section of the Laravel documentation. It contains text instructions and three terminal command examples:

```
# Installation Via Composer

If your computer already has PHP and Composer installed, you may create a new
Laravel project by using Composer directly. After the application has been
created, you may start Laravel's local development server using the Artisan CLI's
serve command:

Composer create-project laravel/laravel example-app
cd example-app
php artisan serve
```

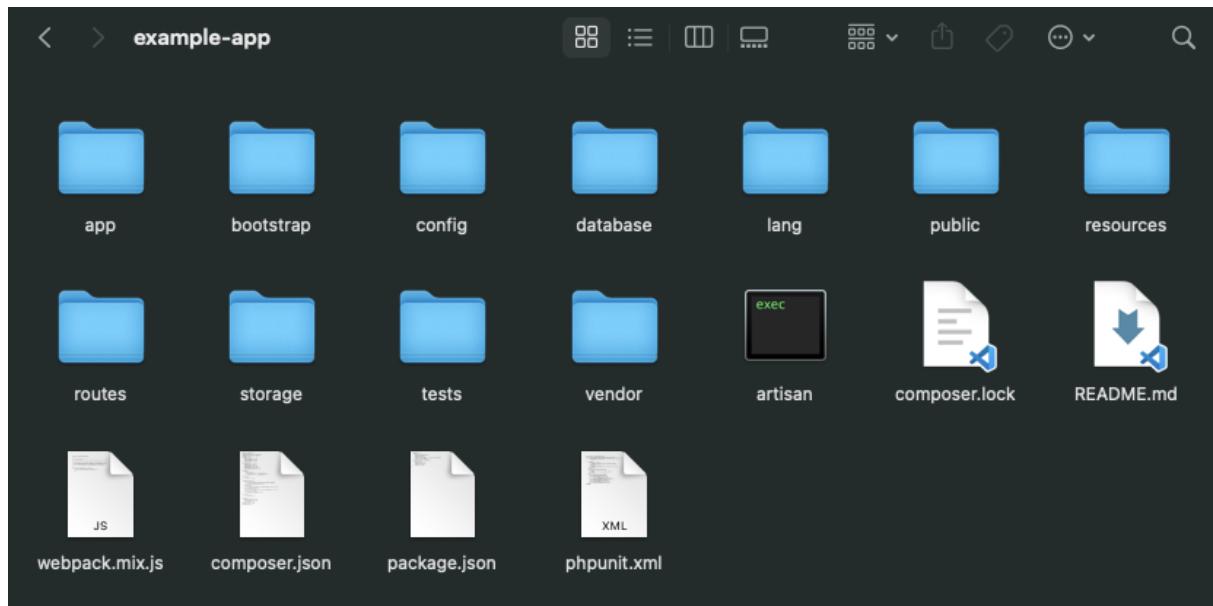
6.2. Mi primer proyecto de Laravel

Usaremos el comando con el que iniciamos un nuevo proyecto, como lo indican las opciones disponibles ejecutaremos el comando de composer (Prestar especial atención a la carpeta donde se ejecuta el comando ya que allí se va a crear la carpeta con el proyecto de Laravel), como se puede observar a continuación:

```
composer create-project laravel/laravel example-app
```

```
[ ~Sites] composer create-project laravel/laravel example-app
Creating a "laravel/laravel" project at "./example-app"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v9.1.2)
- Downloading laravel/laravel (v9.1.2)
- Installing laravel/laravel (v9.1.2): Extracting archive
Created project in /Users/ccasta23/Sites/example-app
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 108 installs, 0 updates, 0 removals
- Locking brick/math (0.9.3)
- Locking dflydev/dot-access-data (v3.0.1)
- Locking doctrine/inflector (2.0.4)
- Locking doctrine/instantiator (1.4.1)
- Locking doctrine/lexer (1.2.3)
- Locking dragonmantank/cron-expression (v3.3.1)
```

Laravel crea una nueva carpeta con el nombre que le asignamos al proyecto, y dentro escribe toda su [estructura de carpetas](#) y archivos que se ve como se muestra a continuación:



Por defecto laravel contiene un archivo llamado *artisan*, este archivo contiene un conjunto de utilidades básicas que nos permiten hacer un mejor uso de Laravel, es importante visitar la documentación oficial de *artisan* y ver el listado de comandos que tiene disponibles a nuestro servicio. Al ejecutar el comando `php artisan` podremos observar el listado de opciones, una muestra de ello se relaciona a continuación:

```
└─ php artisan

Laravel Framework 9.5.1

Usage:
  command [options] [arguments]

Options:
  -h, --help           Display help for the given command. When no command is given display help for the list command
  -q, --quiet          Do not output any message
  -V, --version         Display this application version
  --ansi|--no-ansi     Force (or disable --no-ansi) ANSI output
  --n, --no-interaction Do not ask any interactive question
  --env[=ENV]           The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

A continuación usaremos uno de los comandos de *artisan* que con mayor frecuencia ejecutaremos ya que se trata de un servidor de pruebas para PHP instalado por defecto en Laravel.

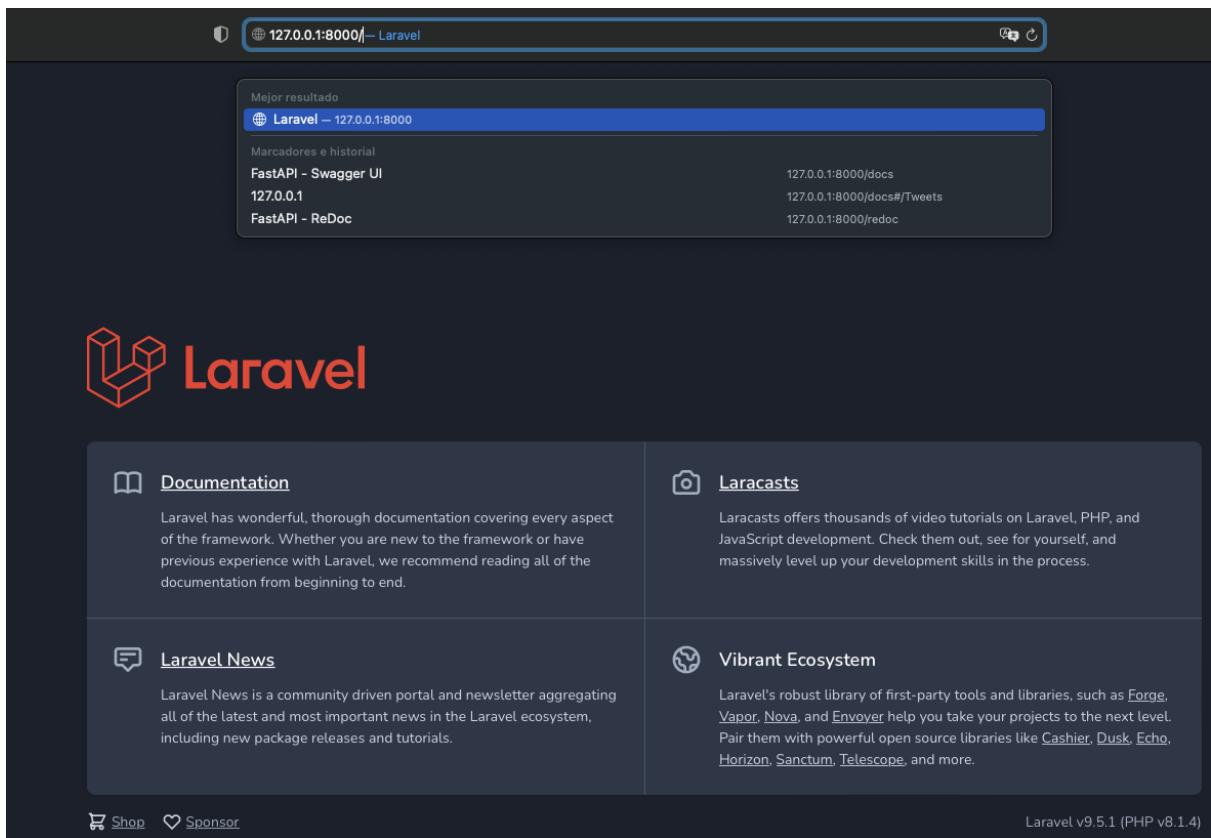
```
php artisan serve
```

Este comando desplegará un servidor que tiene lo necesario para una correcta ejecución de Laravel y su salida se muestra a continuación:

```
└─ php artisan serve

INFO Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server
```

Luego de tener el servidor corriendo, ya debemos tener disponible desde nuestro navegador el proyecto, accediendo a la dirección URL <http://127.0.0.1:8000>



Y con esto ya tenemos un Laravel instalado en nuestra máquina con un proyecto ejecutándose desde un servidor levantado a través de artisan.

6.3. Rutas en Laravel

Actualmente en el proyecto, tenemos disponible únicamente la ruta del punto de entrada del aplicativo la cual venía instalada por defecto en el proyecto. Para validar las rutas disponibles en el código nos dirigimos al archivo **routes/web.php** que debe tener un contenido como el que se muestra a continuación:

```

example-app > routes
Project  PHP web.php ×
example-app ~/Sites/example-a
> app
> bootstrap
> config
> database
> lang
> public
> resources
  > routes
    > api.php
    > channels.php
    > console.php
    > web.php
  > storage
  > tests
  > vendor
  .editorconfig
  .env
  .env.example
  .gitattributes
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  /*
6  |--------------------------------------------------------------------------
7  | Web Routes
8  |--------------------------------------------------------------------------
9
10 | Here is where you can register web routes for your application. These
11 | routes are loaded by the RouteServiceProvider within a group which
12 | contains the "web" middleware group. Now create something great!
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });

```

Nota: Es importante validar la [documentación oficial de rutas de Laravel](#), y conocer los métodos HTTP con sus peticiones ([Request](#)) que tenemos disponibles.

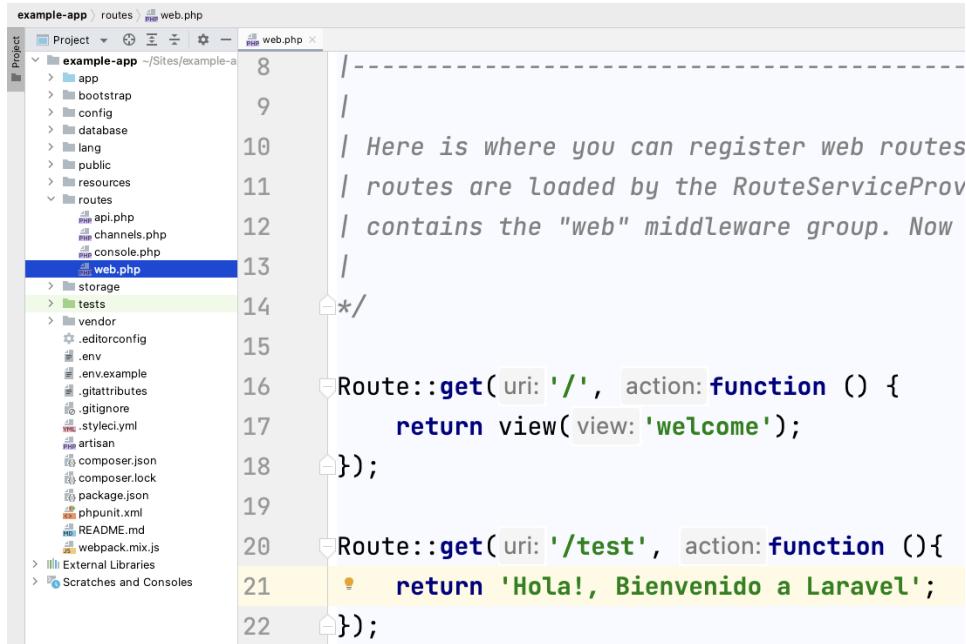
Como se puede apreciar en el archivo, la ruta principal del proyecto está redireccionando a una vista de PHP llamada `welcome`. Las vistas en Laravel se encuentran ubicadas en la carpeta **resources/views** y para el caso de la vista de entrada en el archivo **welcome.blade.php**. El contenido del archivo se encuentra a continuación:

```

example-app > resources > views > welcome.blade.php
Project  PHP web.php ×  welcome.blade.php ×
example-app ~/Sites/example-a
> app
> bootstrap
> config
> database
> lang
> public
> resources
  > css
  > js
  > views
    > welcome.blade.php
  > routes
  > storage
  > tests
  > vendor
  .editorconfig
  .env
1  <style>
2  </style>
3
4  </head>
5  <body class="antialiased">
6
7      <div class="relative flex items-top justify-center min-h-screen bg-gray-100 sm:items-center sm:pt-0" >
8          @if (Route::has('login'))
9              <div class="hidden fixed top-0 right-0 px-6 py-4 sm:block" >
10                  @auth
11                      <a href="{{ url('home') }}" class="text-sm text-gray-700 font-medium" >{{ __('Home') }} </a>
12                  @else
13                      <a href="{{ route('login') }}" class="text-sm text-gray-700 font-medium" >{{ __('Log in') }} </a>
14                  @endif
15              </div>
16          @endif
17      </div>
18  </body>
19</html>

```

Como primera prueba de rutas en Laravel crearemos una propia, para ello iremos al archivo de rutas `web.php` en donde escribiremos lo siguiente:

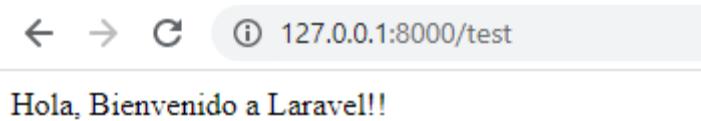


```
example-app > routes > web.php
Project  example-app ~/Sites/example-a
> app
> bootstrap
> config
> database
> lang
> public
> resources
> routes
> web.php
> storage
> tests
> vendor
> .editorconfig
> .env
> .env.example
> .gitattributes
> .gitignore
> .styleci.yml
> artisan
> composer.json
> composer.lock
> package.json
> phpunit.xml
> README.md
> webpack.mix.js
> External Libraries
> Scratches and Consoles

8 | -----+
9 | |
10| / Here is where you can register web routes
11| routes are loaded by the RouteServiceProvider
12| contains the "web" middleware group. Now
13| |
14| */ 
15|
16| Route::get(uri: '/', action: function () {
17|     return view('welcome');
18| });
19|
20| Route::get(uri: '/test', action: function () {
21|     return 'Hola!, Bienvenido a Laravel!';
22| });

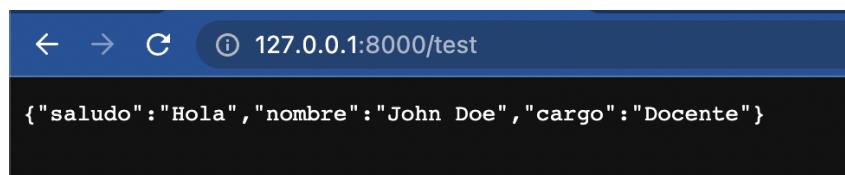

```

Para acceder a la ruta creada vamos al navegador y escribimos <http://127.0.0.1:8000/test> obteniendo el siguiente resultado:



En este caso retornamos una cadena de texto que es mostrada en el navegador. Podemos regresar otros tipos de datos como un arreglo asociativo en donde Laravel lo que hace es convertirlo a formato JSON y enviarlo de regreso al navegador como se muestra a continuación:

```
Route::get(uri: '/test', function () {
    return [
        'saludo' => 'Hola',
        'nombre' => 'John Doe',
        'cargo' => 'Docente'
    ];
});
```



Ahora, cuando queremos retornar una página completa, tomamos ejemplo de la ruta creada por defecto en donde se retorna una vista completa que corresponde a un archivo con extensión **.blade.php**, estas páginas son creadas con el motor de templates blade que veremos más adelante. Para probar crearemos una nueva página copiando el archivo anterior creado con blade llamado **test.blade.php** copia del anterior.

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER**: Shows the project structure with a tree view of files and folders.
- OPEN EDITORS**: Shows the currently open files: `test.blade.php`, `resources > views > test.blade.php`, and `resources > views > welcome.blade.php`.
- Code Editor**: The main area displays the `test.blade.php` file content. The code uses Blade templating syntax (`@auth`, `@else`, `@if`, `@endif`, `@endauth`) and HTML classes (`top-right-links`, `content`, `title`, `links`). A red box highlights the `<div class="title m-b-md">` block, which contains the text "Curso de Laravel".

```
<div class="top-right-links">
    @auth
        <a href="{{ url('/home') }}">Home</a>
    @else
        <a href="{{ route('login') }}">Login</a>

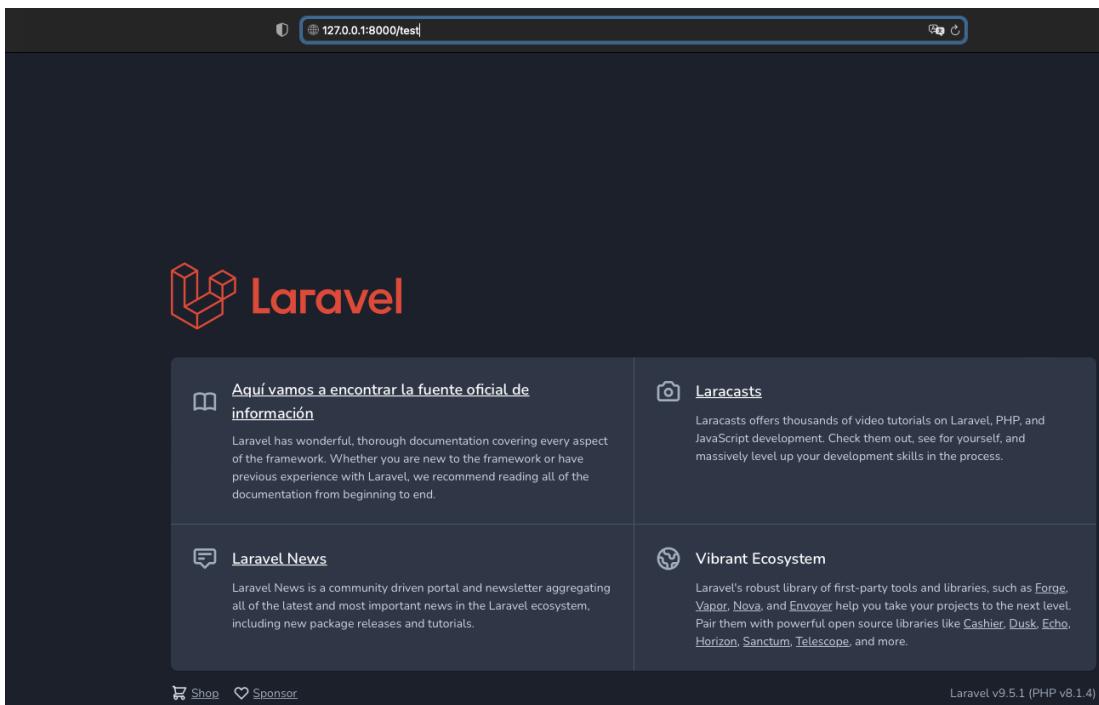
        @if (Route::has('register'))
            <a href="{{ route('register') }}">Register</a>
        @endif
    @endauth
</div>
@endif

<div class="content">
    <div class="title m-b-md">
        Curso de Laravel
    </div>
<div class="links">
    <a href="https://laravel.com/docs">Docs</a>
    <a href="https://laracasts.com">Laracasts</a>
    <a href="https://laravel-news.com">News</a>
</div>
```

Y en nuestro archivo `web.php` se retorna la nueva vista creada como se muestra a continuación:

```
11 | routes are loaded by the RouteServiceProvider within a g
12 | contains the "web" middleware group. Now create somethin
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19
20 Route::get('/test', function () {
21     return view('test');
22 });
23
```

Obteniendo la siguiente respuesta:



Para listar todas las rutas que se tienen activas actualmente en el sistema, se puede usar el siguiente comando:

```
php artisan route:list
```

Obteniendo un resultado como el que se muestra a continuación:

```
apple:~/Sites/example-app ➜ 16:37:00
php artisan route:list
GET|HEAD / ...
POST _ignition/execute-solution ... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user ...
GET|HEAD sanctum/csrf-cookie ...
GET|HEAD test/{nombre} ... Laravel\Sanctum > CsrfCookieController@show
```

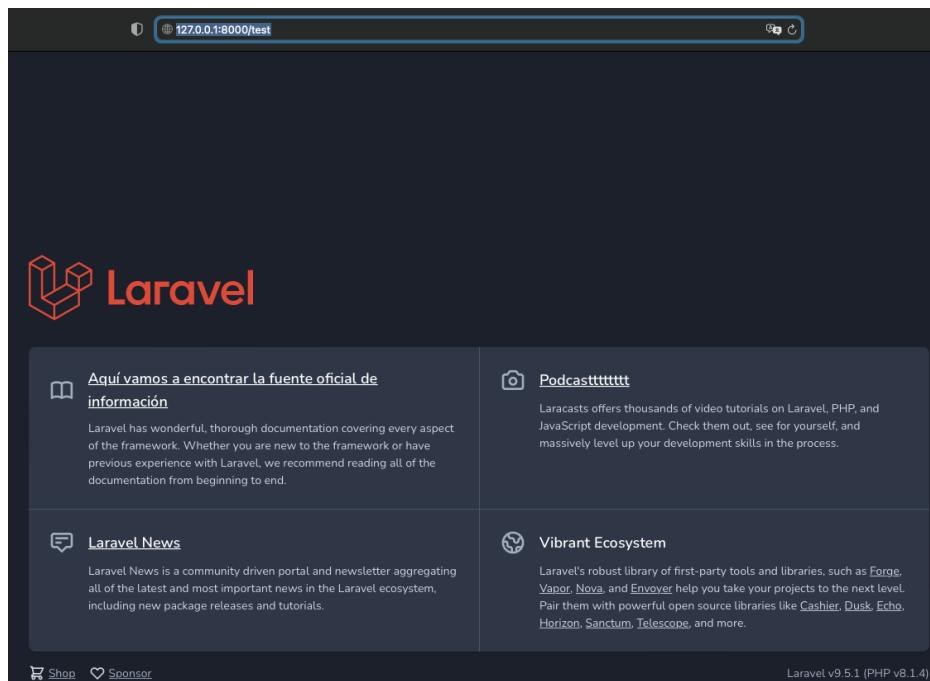
6.4. Template Engine Blade

Blade es el motor de plantillas simple pero potente provisto con Laravel. A diferencia de otros motores de plantillas PHP populares, Blade no le restringe el uso de código PHP simple en sus vistas. De hecho, todas las vistas de Blade se compilan en código PHP simple y se almacenan en caché hasta que se modifican, lo que significa que Blade agrega esencialmente cero sobrecarga a su aplicación. Los archivos de vista Blade usan la extensión de archivo .blade.php y generalmente se almacenan en el directorio `resources/views`.

Blade dentro de sus plantillas acepta código de PHP puro como se muestra a continuación:

```
test.blade.php.x
63
64
65      ="0 0 24 24" class="w-8 h-8 text-gray-500">><path d="M3 9a2 2 0 012-
66      00 dark:text-white"><?php echo "Podcasttttttt"; ?></a></div>
/7
```

Obteniendo el siguiente resultado:



A pesar de que Blade nos permite **el uso de PHP directamente en el código, no es recomendado** y no debe usarse ya que el template engine contiene su propia estructura y sus propios tags que pueden ser consultados en su [documentación oficial](#).

Es posible pasarle información a las vistas (Por ejm. a través de un arreglo asociativo), para ello modificaremos nuestro archivo web.php, agregando un arreglo asociativo con alguna información como se muestra a continuación:

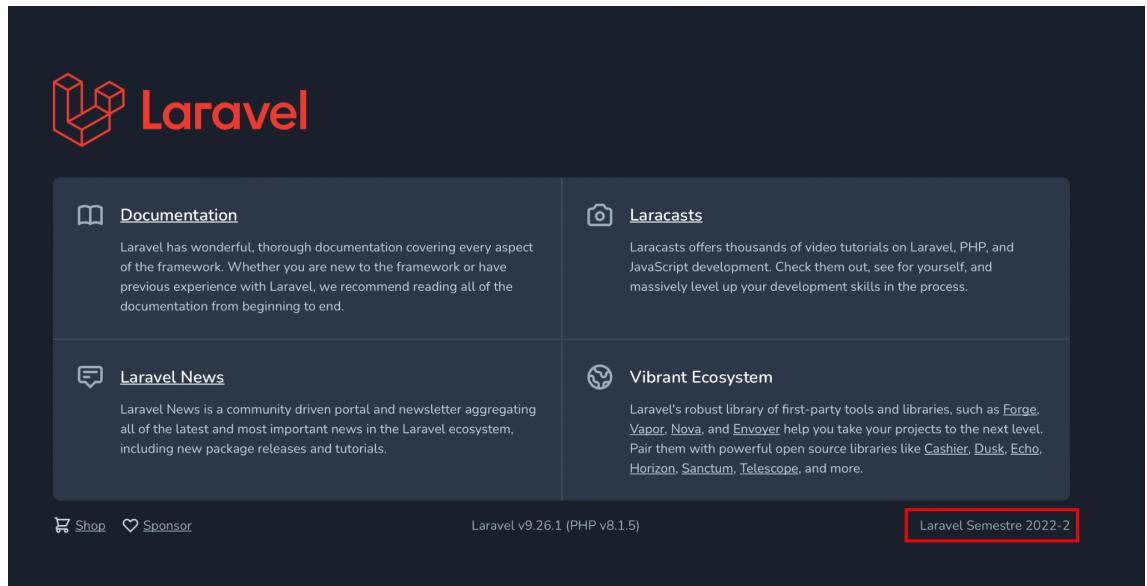
```
Route::get('/test', function () {
    return view('test', [
        'subject' => 'Laravel',
        'semester' => '2022-2'
    ]);
});
```

A continuación desde nuestro template de ***test.blade.php*** agregaremos el código necesario para recibir dicha información y mostrarla en pantalla.

```
<div class="ml-4 text-center text-sm text-gray-500 sm:text-right sm:ml-0">
    Laravel v{{ Illuminate\Foundation\Application::VERSION }} (PHP v{{ PHP_VERSION }})
</div>

<div class="ml-4 text-center text-sm text-gray-500 sm:text-right sm:ml-0">
    {{ $subject }} Semestre: {{ $semester }}
</div>
</div>
</div>
</body>
</html>
```

Obteniendo el siguiente resultado:



Adicionalmente, para probar algunos helpers de blade, verificaremos si el título recibido si existe como se muestra a continuación:

```
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
```

```
<div class="ml-4 text-center text-sm text-gray-500 sm:text-right sm:ml-0">
    Laravel v{{ Illuminate\Foundation\Application::VERSION }} (PHP v{{ PHP_VERSION }})
</div>

<div class="ml-4 text-center text-sm text-gray-500 sm:text-right sm:ml-0">
    Subject:
    @isset($subject)
        {{ $subject }}
    @else
        Framework Laravel para Desarrollo Web.
    @endisset
    Semestre: {{ $semester }}
</div>
</div>
</div>
```

6.5. Controladores en Laravel

Laravel funciona con la arquitectura MVC (Model, View, Controller), las vistas las manejamos como lo vimos en el capítulo anterior, y a continuación veremos cómo funcionan los controladores.

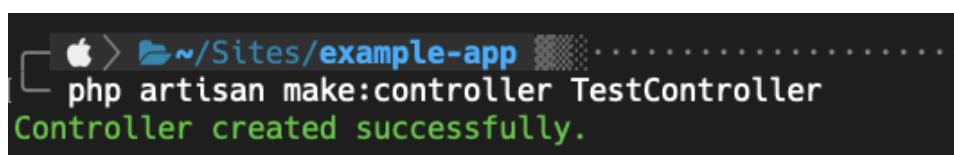
En la estructura de nuestro proyecto Laravel dentro de la carpeta App, se encuentra la carpeta Http en donde por defecto deben ser almacenados todos nuestros controladores. Nuestra herramienta de línea de comandos *artisan* permite a través de su sección de comandos *make* la creación con los parámetros y métodos por defecto de un controlador, para ello ejecuto la siguiente sentencia:

```
php artisan make:controller nombreControlador
```

Para más información sobre el comando *make:controller* y sus diferentes banderas ejecutar el siguiente comando:

```
php artisan make:controller --help
```

En nuestro proyecto crearemos un controlador llamado *homeController*, mostrando una salida por línea de comandos como la mostrada a continuación:



A terminal window showing the command "php artisan make:controller TestController" being run. The output "Controller created successfully." is displayed in green text at the bottom.

```
php artisan make:controller TestController
Controller created successfully.
```

Adicionalmente, si revisamos en nuestro código fuente, en la carpeta app/Http/Controllers veremos que el nuevo archivo fue creado, su estructura se detalla a continuación:

```

example-app > app > Http > Controllers > TestController.php > TestController
Project  TestController.php
example-app ~/Sites/example-a
  app
    > Console
    > Exceptions
    > Http
      > Controllers
        Controller.php
        TestController.php
        > Middleware
        Kernel.php
      > Models
      > Providers
    > bootstrap
    > config

```

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class TestController extends Controller
8 {
9
10
11
12 }

```

Para efectos de probar la funcionalidad de un controlador crearemos un nuevo método adentro llamado *index*, el cual me permitirá responder a una petición HTTP como lo veníamos haciendo en nuestro *web.php*, adicionalmente enlazaremos una ruta directamente con el controlador. A continuación se muestran los cambios realizados en los archivos *app/Http/Controllers/HomeController.php* y en el archivo *routes/web.php*.

```

app
  > Console
  > Exceptions
  > Http
    > Controllers
      Controller.php
      TestController.php
      > Middleware
      Kernel.php
    > Models
    > Providers
  > bootstrap
  > config
  > database
  > lang

```

```

4
5 use Illuminate\Http\Request;
6
7 class TestController extends Controller
8 {
9   public function saladar()
10  {
11    return 'Hola, bienvenido a Laravel';
12  }
13
14  public function index()
15  {
16    return view( view: 'test', [
17      'semester' => '2022-2'
18    ]);
19  }
20
21

```

```

PHP web.php
1
2
3 use Illuminate\Support\Facades\Route;
4
5 /*
6 |--------------------------------------------------------------------------
7  | Web Routes
8  |--------------------------------------------------------------------------
9  |
10 | Here is where you can register web routes for your application. These
11 | routes are loaded by the RouteServiceProvider within a group which
12 | contains the "web" middleware group. Now create something great!
13 |
14 */
15
16 Route::get( uri: '/', action: [\App\Http\Controllers\TestController::class, 'index']);
```

Con esto, logramos que nuestra aplicación continúe funcionando correctamente, la diferencia es que esta vez en `web.php`, al recibir la petición ya no se utiliza un closure (Función anónima) sino que por el contrario se ejecuta un método de un controlador.

Reto Controladores

Crear un controlador llamado `DashboardController` que será el punto de ingreso a nuestra aplicación, crear su respectiva ruta y retornar la vista de test creada en puntos anteriores.

Finalmente, para expandir la información se recomienda visitar la [documentación oficial](#) de los controladores de Laravel

6.6. Request

Las peticiones HTTP son la base del funcionamiento de los aplicativos web, por ello es necesario saber en donde recibimos la petición, de qué tipo es y qué parámetros trae.

Para ello en nuestro controlador de `DashboardController` creado en el punto anterior agregaremos un parámetro adicional al método `index`, de tipo `Request`. Gracias a la inyección de dependencias PHP sabrá que cuando se solicite un parámetro de tipo `Request` él entregará al método toda la información de la petición HTTP.



```
DashboardController.php ×
app > Http > Controllers > DashboardController.php > DashboardController > index
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class DashboardController extends Controller
8  {
9      public function index(Request $request) //Entra por inyección de dependencias
10     var_dump($request);
11     return view('test', [
12         'title' => 'Curso PHP Con Laravel - Dashboard',
13         'description' => 'Curso creado para aprender PHP'
14     ]);
15 }
16
17 }
```

Agregamos el método `var_dump()` para verificar el contenido completo de la petición HTTP, el resultado en el navegador se muestra a continuación:

The screenshot shows a browser window with the URL `127.0.0.1:8000/dashboard/?title=Mi%20Titulo%20Personalizado`. The page content is a large block of PHP code, specifically the output of `var_dump($request)`. A red box highlights the first few lines of the output, which represent the protected properties of the `Illuminate\Http\Request` object. The code is heavily annotated with comments and recursion markers (*RECURSION*), indicating the complex nature of the object's internal structure.

```
object(Illuminate\Http\Request)#50 (28) { ["json":protected]=> NULL ["convertedFiles":protected]=> NULL ["userResolver":protected]=> object(Closure)#216 (3) { ... } C:\xampp\htdocs\cursoalvarez\1\masBeenBootstrapped":protected]=> bool(true) ["booted":protected]=> bool(true) ["bootingCallbacks":protected]=> array(1) { [0]=> ["app":protected]=>> *RECURSION* } } ["this"]=> *RECURSION* } } ["bootedCallbacks":protected]=> array(1) { [0]=> object(Closure)#192 (1) { ["this"]=> object(["app":protected]=>> *RECURSION* ) } } ["terminatingCallbacks":protected]=> array(0) { } ["serviceProviders":protected]=> array(24) { [0]=> object(Illuminate\Even... object(Illuminate\Log\LogServiceProvider)#15 (1) { ["app":protected]=>> *RECURSION* } [2]=> object(Illuminate\Routing\RoutingServiceProvider)#17 (1) { ["app":protected]=>> *RECURSION* } [4]=> object(Illuminate\Cookie\CookieServiceProvider)#57 (1) { ["app":protected]=>> *RECURSION* } [5]=> object(Illuminate\Database\Database... object(Illuminate\Encryption\EncryptionServiceProvider)#67 (1) { ["app":protected]=>> *RECURSION* } [7]=> object(Illuminate\Filesystem\FilesystemServiceProvider)... object(Illuminate\Foundation\Providers\FormRequestServiceProvider)#75 (1) { ["app":protected]=>> *RECURSION* } [9]=> object(Illuminate\Foundation\Providers..."Illuminate\Foundation\Providers\FormRequestServiceProvider" ) ["instances":protected]=> array(1) { [0]=> object(Illuminate\Foundation\Providers\FormRequestService... object(Illuminate\Notifications\NotificationServiceProvider)#79 (1) { ["app":protected]=>> *RECURSION* } [11]=> object(Illuminate\Pagination\PaginationServiceProvider)... ["app":protected]=>> *RECURSION* } [13]=> object(Illuminate\View\ViewServiceProvider)#90 (1) { ["app":protected]=>> *RECURSION* } [14]=> object(Facade\I... object(Fideloper\Proxy\TrustedProxyServiceProvider)#117 (1) { ["app":protected]=>> *RECURSION* } [16]=> object(Fruitcake\Cors\CorsServiceProvider)#116 (1) ... }
```

Aquí se muestra información completa y relevante de la petición, sin embargo probablemente no queremos disponer de toda esta información y por ahora sólo queremos acceder al parámetro `title` agregado a la petición a través de la URL ([Petición HTTP GET con parámetros](#)). Para obtener el valor del parámetro `title` se modifica el código del método `index` como se especifica a continuación:

```
public function index(Request $request) //Entra por inyección de dependencias
{
    var_dump($request->query('title'));
    die;
    return view('test', [
        'title' => 'Curso PHP Con Laravel - Dashboard',
        'description' => 'Curso creado para aprender PHP'
    ]);
}
```

Obteniendo el siguiente resultado:

The screenshot shows a browser window with the URL `127.0.0.1:8000/dashboard/?title=Mi%20Titulo%20Personalizado`. The page content is the result of the modified `index` method. It shows the output of `var_dump($request->query('title'))`, which is a string "Mi Titulo Personalizado", followed by a `die();` statement that stops the execution of the script.

```
string(23) "Mi Titulo Personalizado"
```

Laravel ofrece el método `dd` que es la combinación entre `var_dump` y `die`, el resultado con el método `dd` se muestra a continuación:

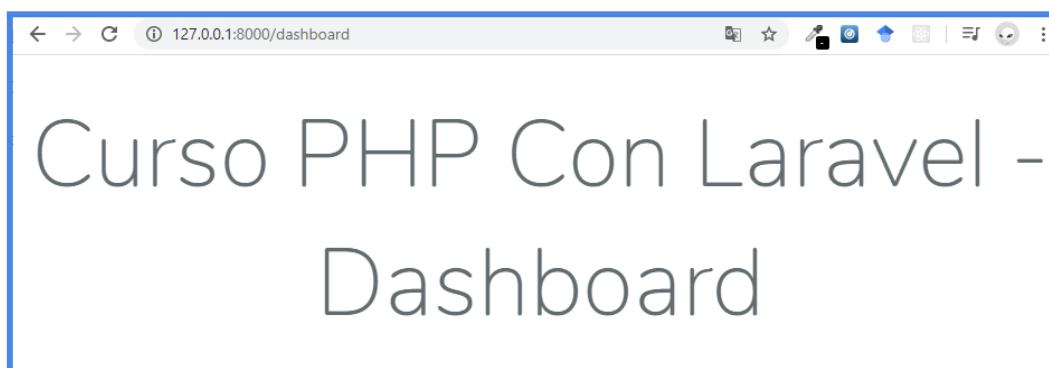
The screenshot shows a browser window with the URL `127.0.0.1:8000/dashboard?title=Mi%20Titulo%20Personalizado`. The page content is the result of the modified `index` method using the `dd` method instead of `var_dump`. It shows the string "Mi Titulo Personalizado" enclosed in green double quotes, indicating it is being displayed directly to the user.

```
"Mi Titulo Personalizado"
```

El método `query` puede recibir un segundo parámetro con la finalidad de especificar un valor por defecto en caso tal de que el parámetro esperado no se reciba. A continuación se muestra el ejemplo de cómo se especifica un

valor por defecto y la salida de nuestra página en caso de enviar o no el parámetro:

```
public function index(Request $request) { //Entra por inyección de dependencias
    //var_dump($request->query('title')); die;
    //dd($request->query('title'));
    $title = $request->query('title', 'Curso PHP Con Laravel - Dashboard');
    return view('test', [
        'title' => $title,
        'description' => 'Curso creado para aprender PHP'
    ]);
}
```



6.7. Configuración de Laravel

Por defecto Laravel ya trae algunas utilidades preinstaladas, algunas de ellas requieren configuraciones como credenciales, puntos de acceso, puertos entre otros. El archivo en donde colocaremos dichos datos es en `.env`, archivo que se encuentra a nivel de la raíz del proyecto.

The screenshot shows a code editor interface with the following details:

- EXPLORER**: Shows the project structure with files like `DashboardController.php`, `.env`, and `CURSOLARAVEL` (which contains subfolders `app`, `bootstrap`, `config`, etc.).
- OPEN EDITORS**: Shows the `.env` file being edited.
- Content of the .env file:**

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:LU7EwWDItafzTA572ANoq9DzFKc/UgMVW3NiFTEU7R4=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=cursolaravel
13 DB_USERNAME=root
14 DB_PASSWORD=
15
16 BROADCAST_DRIVER=log
17 CACHE_DRIVER=file
18 QUEUE_CONNECTION=sync
19 SESSION_DRIVER=file
20 SESSION_LIFETIME=120
21
22 REDIS_HOST=127.0.0.1
23 REDIS_PASSWORD=null
24 REDIS_PORT=6379
25
26 MAIL_MAILER=smtp
27 MAIL_HOST=smtp.mailtrap.io
28 MAIL_PORT=2525
29 MAIL_USERNAME=null
30 MAIL_PASSWORD=null
31 MAIL_ENCRYPTION=null
32 MAIL_FROM_ADDRESS=null
33 MAIL_FROM_NAME="${APP_NAME}"
34
35 AWS_ACCESS_KEY_ID=
36 AWS_SECRET_ACCESS_KEY=
37 AWS_DEFAULT_REGION=us-east-1
38 AWS_BUCKET=
```

Por ejemplo, unos datos que podemos configurar inicialmente son las credenciales a nuestra base de datos, para ello debemos ocupar todas las variables del archivo que contienen el prefijo DB. En la [documentación oficial](#) podremos encontrar las variables de configuración con sus valores por defecto y su respectiva explicación.

Ejemplo de variables de entorno configuradas con mysql

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=cursolaravel
DB_USERNAME=homestead
DB_PASSWORD=secret
```

Ejemplo de variables de entorno configuradas con postgresql

```

DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=cursolaravel
DB_USERNAME=postgres
DB_PASSWORD=admin

```

Todos los valores de la configuración de las variables de entorno están siendo colocados desde el archivo `.env`, sin embargo en PHP son cargados desde los archivos que se encuentran dentro de la carpeta de `config`. A continuación veremos el ejemplo del archivo de configuración de PHP que toma las variables de la base de datos y muestra qué valores tiene por defecto.

```

EXPLORER          DashboardController.php    database.php ×
OPEN EDITORS       config > database.php
DashboardController.php...      config > database.php
CURSOLAVAREL
  app
  bootstrap
  config
    app.php
    auth.php
    broadcasting.php
    cache.php
    cors.php
  database.php
    filesystems.php
    hashing.php
    logging.php
    mail.php
    queue.php
    services.php
    session.php
    view.php
  database
  public
  resources
  routes
  storage
  tests
  vendor
.editorconfig
.env
.env.example
.gitattributes
.gitignore
.styleci.yml

38   'sqlite' => [
39     'driver' => 'sqlite',
40     'url' => env('DATABASE_URL'),
41     'database' => env('DB_DATABASE', database_path('database.sqlite')),
42     'prefix' => '',
43     'foreign_key_constraints' => env('DB_FOREIGN_KEYS', true),
44   ],
45
46   'mysql' => [
47     'driver' => 'mysql',
48     'url' => env('DATABASE_URL'),
49     'host' => env('DB_HOST', '127.0.0.1'),
50     'port' => env('DB_PORT', '3306'),
51     'database' => env('DB_DATABASE', 'forge'),
52     'username' => env('DB_USERNAME', 'forge'),
53     'password' => env('DB_PASSWORD', ''),
54     'unix_socket' => env('DB_SOCKET', ''),
55     'charset' => 'utf8mb4',
56     'collation' => 'utf8mb4_unicode_ci',
57     'prefix' => '',
58     'prefix_indexes' => true,
59     'strict' => true,
60     'engine' => null,
61     'options' => extension_loaded('pdo_mysql') ? array_filter([
62       PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
63     ]) : [],
64   ],
65
66   'pgsql' => [
67     'driver' => 'pgsql',
68     'url' => env('DATABASE_URL'),
69     'host' => env('DB_HOST', '127.0.0.1'),
70     'port' => env('DB_PORT', '5432'),
71     'database' => env('DB_DATABASE', 'forge'),
72     'username' => env('DB_USERNAME', 'forge'),
73     'password' => env('DB_PASSWORD', ''),
74     'charset' => 'utf8',
75     'prefix' => ''
  ],

```

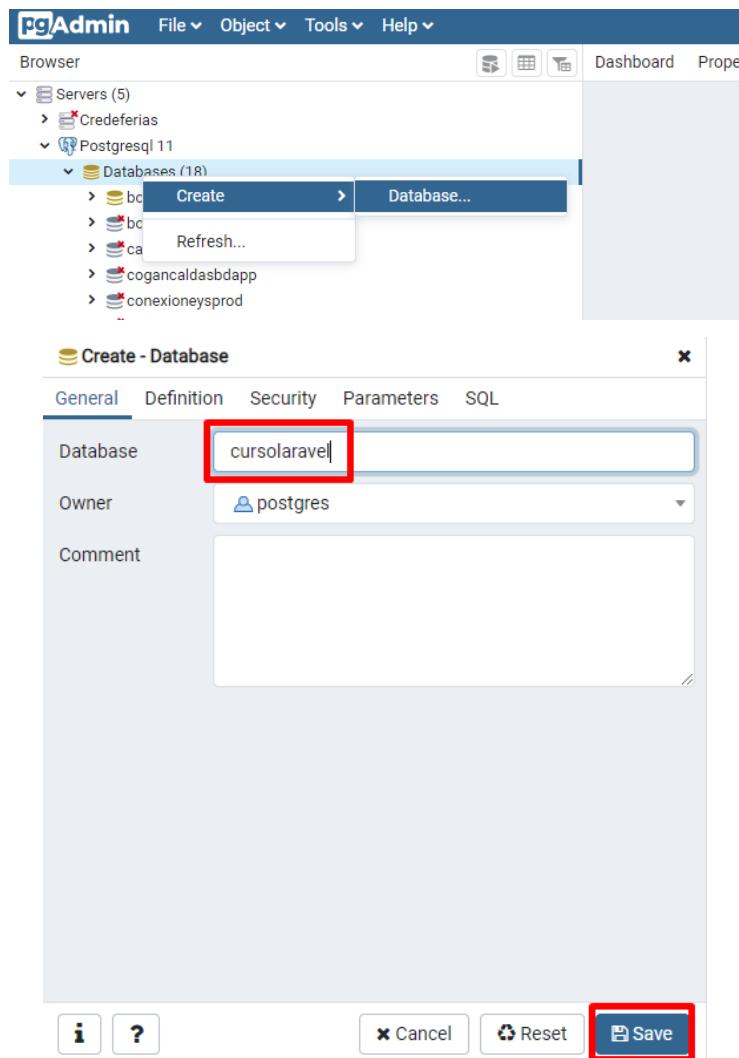
El paso final, es crear la base de datos que usaremos en nuestro proyecto, al utilizar el ORM de Eloquent preinstalado con Laravel, esta creación se puede realizar en la base de datos de su preferencia. Como ejemplo en el documento se utilizará postgresql.

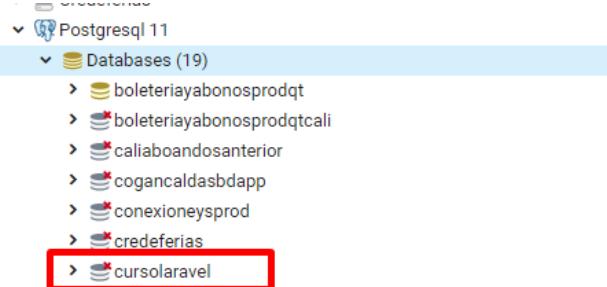
6.8. Pruebas de conexión a la base de datos

A continuación se mostrará la manera de probar la conexión a la base de datos desde una aplicación de Laravel, se probará el funcionamiento con una base de datos postgres y una en mysql.

La precondición de esta sección es que la base de datos de su preferencia ya se encuentre instalada y tenga al menos un cliente desde el cual pueda gestionar su motor de bases de datos.

Primero, a través del cliente pgAdmin crearemos la base de datos en postgresql como se muestra a continuación:





Luego de creada la base de datos, es momento de cambiar los valores de las variables de entorno del archivo de configuración `.env` como se mostró en la sección anterior, para este caso en específico los valores quedarían de la siguiente manera:

```
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=cursolaravelg1
DB_USERNAME=postgres
DB_PASSWORD=admin
```

A continuación para probar la conexión, aprovecharemos el conocimiento adquirido anteriormente para crear una nueva ruta que direccione al método `testDBConection` en el controlador ya creado con anterioridad.

```
routes > web.php
43
44
45 Route::get('/testdb', 'HomeController@testDBConection');
```

```
HomeController.php ×
app > Http > Controllers > HomeController.php > HomeController
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use DB;
7
8 class HomeController extends Controller
9
10 public function testDBconection() {
11     try {
12         DB::connection()->getPdo();
13         if(DB::connection()->getDatabaseName()){
14             return "Yes! Successfully connected to the DB: " . DB::connection()->getDatabaseName();
15         }else{
16             die("Could not find the database. Please check your configuration.");
17         }
18     } catch (\Exception $e) {
19         die("Could not open connection to database server. Please check your configuration.");
20     }
21 }
22
23
```

En caso de que todo esté configurado correctamente deberíamos recibir un mensaje como el que se muestra a continuación:



Yes! Successfully connected to the DB: cursolaravelg1

Si tenemos algún error de escritura o configuración de nuestro servidor de bases de datos recibiremos un error como el mostrado a continuación:



En ese caso se debe revisar al detalle el archivo de configuración .env dejando los valores establecidos correctamente para las bases de datos.

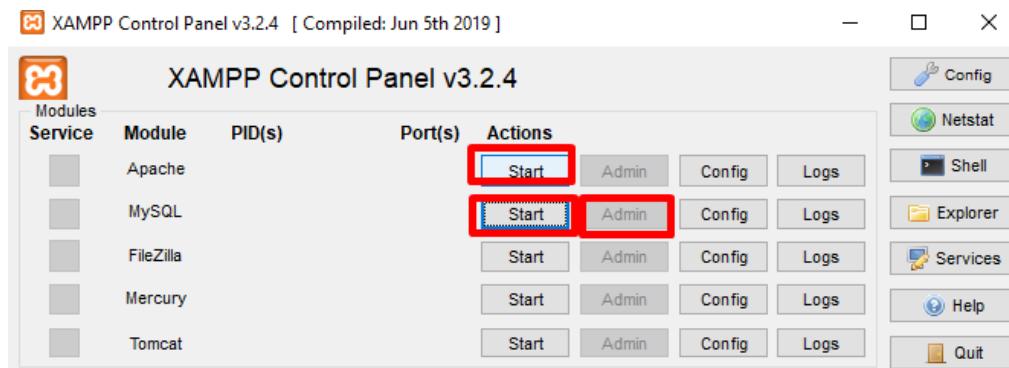
Nota: Cada vez que se realicen cambios en el archivo de configuración .env, para que los cambios se vean reflejados, es necesario reiniciar el servidor (Parar e iniciar de nuevo con el comando php artisan serve)

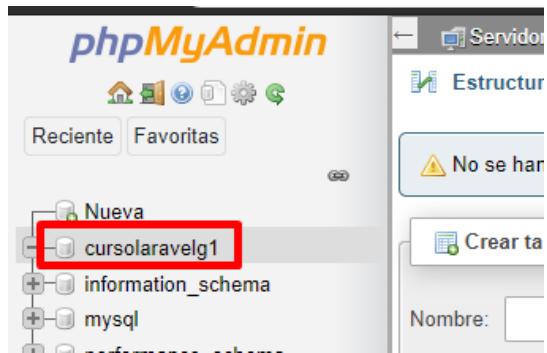
Como Laravel utilizar Eloquent como ORM, es posible que nuestro código funcione de la misma manera bajo dos servidores de bases de datos distintos, es por esto que también mostraremos la manera en que debe ser parametrizado el archivo de configuraciones para establecer una conexión a la base de datos mysql instalada con XAMPP que se instaló previamente en el documento.

La configuración del archivo de .env se vería de la siguiente manera:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=cursolaravelg1
DB_USERNAME=root
DB_PASSWORD=
```

Luego para usar el mysql instalado con XAMPP debemos iniciar apache, iniciar mysql y por último hacer clic en Admin que nos llevará al módulo administrativo como se muestra a continuación:





Nuevamente, reiniciamos nuestro servidor de artisan y actualizamos nuestra página para ver si la conexión se realizó correctamente. El mensaje que deberíamos ver si todo se encuentra correctamente se muestra a continuación:



Nota: Por defecto MySQL no tiene contraseña, esto se puede actualizar como se muestra en [este manual](#), y para entornos productivos es imprescindible tener un password fuerte.

7. Migraciones de Bases de datos

Las migraciones son como el control de versiones de nuestra base de datos, lo que permite al equipo modificar y compartir fácilmente el esquema de la base de datos de la aplicación. Las migraciones generalmente se combinan con el generador de esquemas de Laravel para construir fácilmente el esquema de la base de datos de nuestra aplicación. Si alguna vez ha tenido que decirle a un compañero de equipo que agregue manualmente una columna al esquema de base de datos local, se enfrenta al problema que resuelven las migraciones de la base de datos.

Como en puntos anteriores haremos uso de las utilidades de artisan, este permite realizar diferentes operaciones a las migraciones. Los archivos de las migraciones quedan almacenados en *database/migrations*. Lo primero que haremos será ejecutar el comando de PHP artisan para validar qué utilidades tienen que ver con migraciones.

<code>make</code>	
<code>make:channel</code>	Create a new channel class
<code>make:command</code>	Create a new Artisan command
<code>make:component</code>	Create a new view component class
<code>make:controller</code>	Create a new controller class
<code>make:event</code>	Create a new event class
<code>make:exception</code>	Create a new custom exception class
<code>make:factory</code>	Create a new model factory
<code>make:job</code>	Create a new job class
<code>make:listener</code>	Create a new event listener class
<code>make:mail</code>	Create a new email class
<code>make:middleware</code>	Create a new middleware class
<code>make:migration</code>	Create a new migration file
<code>make:model</code>	Create a new Eloquent model class
<code>make:notification</code>	Create a new notification class
<code>make:observer</code>	Create a new observer class
<code>make:policy</code>	Create a new policy class
<code>make:provider</code>	Create a new service provider class
<code>make:request</code>	Create a new form request class
<code>make:resource</code>	Create a new resource
<code>make:rule</code>	Create a new validation rule
<code>make:seeder</code>	Create a new seeder class
<code>make:test</code>	Create a new test class
<code>migrate</code>	
<code>migrate:fresh</code>	Drop all tables and re-run all migrations
<code>migrate:install</code>	Create the migration repository
<code>migrate:refresh</code>	Reset and re-run all migrations
<code>migrate:reset</code>	Rollback all database migrations
<code>migrate:rollback</code>	Rollback the last database migration
<code>migrate:status</code>	Show the status of each migration

Como en cada sección del documento, se recomienda revisar la [documentación oficial](#) de las migraciones en Laravel.

Antes de crear nuestra primera migración es importante entender que cada migración de Laravel por defecto cuenta con dos métodos que se explican a continuación:

- **up:** que nos dice qué va a crear la migración.
- **down:** que revierte lo que se hizo en la migración, activado al hacer rollback.

Desde el momento de crear el proyecto, Laravel ya trae por definición una primera migración para la tabla de usuarios, su estructura se muestra a continuación:

```

    1  ?php
  2
  3  use Illuminate\Database\Migrations\Migration;
  4  use Illuminate\Database\Schema\Blueprint;
  5  use Illuminate\Support\Facades\Schema;
  6
  7  class CreateUsersTable extends Migration
  8  {
  9
 10     /**
 11      * Run the migrations.
 12      *
 13      * @return void
 14     */
 15     public function up()
 16     {
 17         Schema::create('users', function (Blueprint $table) {
 18             $table->id();
 19             $table->string('name');
 20             $table->string('email')->unique();
 21             $table->timestamp('email_verified_at')->nullable();
 22             $table->string('password');
 23             $table->rememberToken();
 24             $table->timestamps();
 25         });
 26     }
 27
 28     /**
 29      * Reverse the migrations.
 30      *
 31      * @return void
 32     */
 33     public function down()
 34     {
 35         Schema::dropIfExists('users');
 36     }
 37

```

Algunos comandos básicos de las migraciones de Laravel que por defecto traen las utilidades de artisan se muestran a continuación:

- Tenemos un comando llamado **make:migration** que nos permite generar archivos de migraciones.
- Contamos con una sección entera llamada **migrate** que nos servirá para realizar diferentes acciones relacionadas con las migraciones. Si ejecutamos migrate sin nada más, ejecutará todas las migraciones pendientes en el equipo.
- **migrate:fresh** va a borrar todas las tablas y las creará de nuevo utilizando todas las migraciones que tenemos.
- **migrate:rollback** nos permite regresar un paso.

7.1. Mi primera migración

Para crear una migración propia, realizaremos la creación de una tabla en la base de datos. Para comenzar usaremos el comando make:migration para crear la migración, sus opciones de uso se muestran a continuación:

```
php artisan make:migration --help
```

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan make:migration --help
Description:
    Create a new migration file

Usage:
    make:migration [options] [--] <name>

Arguments:
    name           The name of the migration

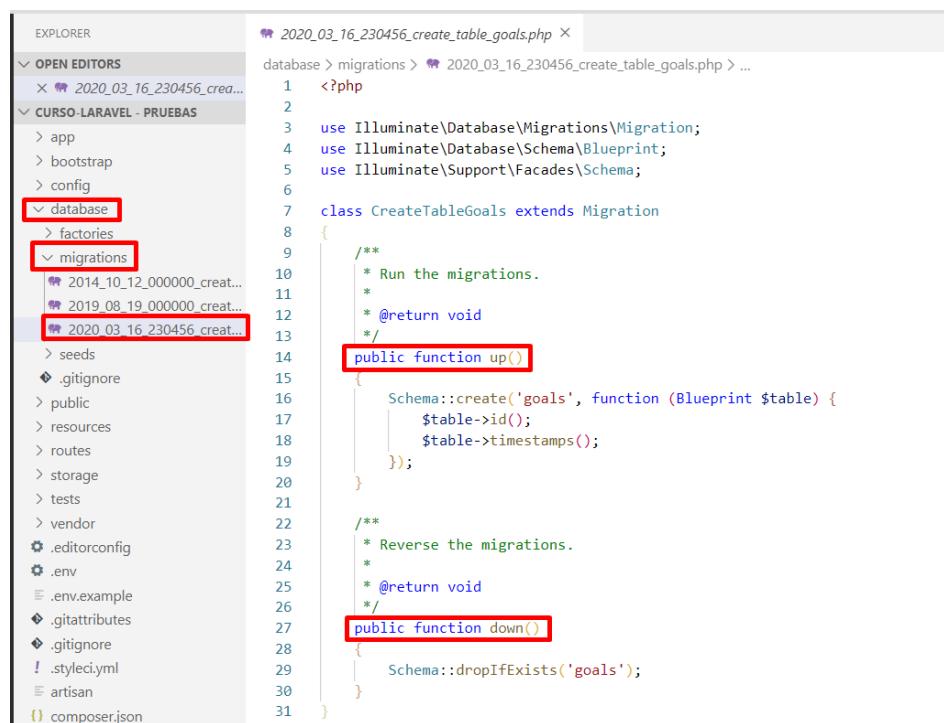
Options:
    --create[=CREATE]  The table to be created
    --table[=TABLE]   The table to migrate
    --path[=PATH]     The location where the migration file should be created
    --realpath        Indicate any provided migration file paths are pre-resolved absolute paths
    --fullpath        Output the full path of the migration
    -h, --help        Display this help message
    -q, --quiet       Do not output any message
    -V, --version     Display this application version
    --ansi           Force ANSI output
    --no-ansi         Disable ANSI output
    -n, --no-interaction  Do not ask any interactive question
    --env[=ENV]        The environment the command should run under
    -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

Adicionando banderas al comando de creación de migración es posible hacer que el archivo a crear se ajuste mucho más a la acción que yo quiero realizar, por ello es importante validar las opciones disponibles. Para crear nuestra primera migración ejecutaremos el siguiente comando:

```
php artisan make:migration nombre_migracion --create=Tabla_Nueva
```

Obteniendo el siguiente resultado:

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan make:migration create_table_goals --create=goals
Created Migration: 2020_03_16_230456_create_table_goals
```



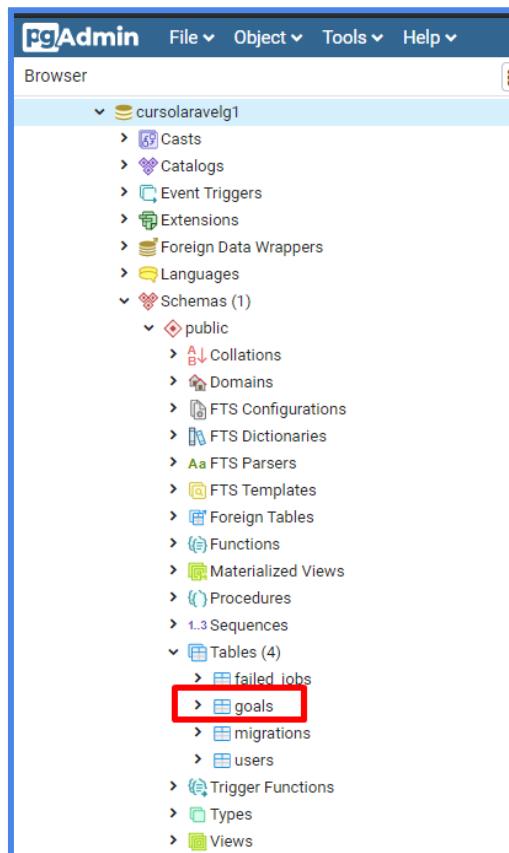
```
2020_03_16_230456_create_table_goals.php
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateTableGoals extends Migration
8  {
9      /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14      public function up()
15      {
16          Schema::create('goals', function (Blueprint $table) {
17              $table->id();
18              $table->timestamps();
19          });
20
21      }
22
23      /**
24      * Reverse the migrations.
25      *
26      * @return void
27      */
28      public function down()
29      {
30          Schema::dropIfExists('goals');
31      }
32  }
```

Sin hacer ninguna modificación al código generado, ejecutaremos nuestra migración para validar que la tabla sea creada en nuestra base de datos, para ello ejecutamos el comando mostrado a continuación:

```
php artisan migrate
```

Obteniendo los siguientes resultados:

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.01 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.02 seconds)
Migrating: 2020_03_16_230456_create_table_goals
Migrated: 2020_03_16_230456_create_table_goals (0.01 seconds)
```



Como aún no tenemos todos los campos de la base de datos y queremos que estos sean creados desde nuestra migración inicial, la retrocedemos (Hacer *rollback*) a la versión previa, para ello ejecutaremos el siguiente comando:

```
php artisan migrate
```

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan migrate:rollback
Rolling back: 2020_03_16_230456_create_table_goals
Rolled back: 2020_03_16_230456_create_table_goals (0.01 seconds)
Rolling back: 2019_08_19_000000_create_failed_jobs_table
Rolled back: 2019_08_19_000000_create_failed_jobs_table (0 seconds)
Rolling back: 2014_10_12_000000_create_users_table
Rolled back: 2014_10_12_000000_create_users_table (0.01 seconds)
```

Obteniendo el siguiente resultado:



Como se puede observar todas las tablas fueron borradas dado que los cambios de la migración fueron devueltos. Para tener control de la ejecución de las migraciones, laravel cuenta con la tabla migrations en donde podremos observar qué migraciones fueron ejecutadas y en qué momento y cuales fueron al tiempo (batch). Gracias al valor de batch al hacer rollback se devuelven todas las migraciones que tengan el valor más alto.

	id [PK] integer	migration character varying (255)	batch integer
1	1	2014_10_12_000000_create_...	1
2	2	2019_08_19_000000_create_f...	1
3	3	2020_03_16_230456_create_t...	1

Un comando adicional a probar es migrate:fresh que lo que hace es eliminar las tablas por completo y crearlas nuevamente. Tener cuidado al ejecutar este comando ya que todos los datos dentro de las tablas serán eliminados.

```
php artisan migrate:fresh
```

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan migrate
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.03 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.02 seconds)
Migrating: 2020_03_16_230456_create_table_goals
Migrated: 2020_03_16_230456_create_table_goals (0 seconds)

CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.01 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.02 seconds)
Migrating: 2020_03_16_230456_create_table_goals
Migrated: 2020_03_16_230456_create_table_goals (0 seconds)
```

Pueden ver el [video abierto del curso de Platzi](#) de PHP con Laravel en donde se explican detalles adicionales a los aquí mencionados.

En [este enlace](#) podrán encontrar los tipos de datos que se pueden usar en las columnas de Laravel.

7.2. Creando nuevas columnas a una tabla

Se recomienda que antes de realizar esta sección, se realicen las prácticas 8.1 y 8.2.

A pesar de que ya se creó la tabla students, esta no tiene campos o columnas activas por lo que debemos crear una nueva migración para modificar la estructura de la tabla.

Nota: NO se recomienda modificar la migración actual dado que es probable que si trabajamos en equipo otro miembro ya haya ejecutado y se preste a confusiones, por eso siempre es importante crear una nueva y no es recomendado modificar las migraciones ya creadas.

Para hacerlo se crea una nueva migración y se agregan las columnas en el método up() como se muestra a continuación:

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan make:migration create_goals_columns --table=goals
Created Migration: 2020_03_23_213233_create_goals_columns
```

```
database > migrations > 2020_03_23_213233_create_goals_columns.php > CreateGoalsColumns > down
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateGoalsColumns extends Migration
8  {
9      /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14
15      public function up()
16      {
17          Schema::table('goals', function (Blueprint $table) {
18              $table->string('name');
19              $table->string('description');
20              $table->integer('months');
21              $table->integer('priority');
22              $table->boolean('active');
23          });
24      }

```

Además, como en esta migración lo que estamos haciendo es modificar la tabla y no crearla de nuevo, si queremos hacer rollback lo que tendrá que tener configurado la migración es borrar las columnas creadas como se muestra a continuación:

```
/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::table('goals', function (Blueprint $table) {
        $table->dropColumn('name');
        $table->dropColumn('description');
        $table->dropColumn('months');
        $table->dropColumn('priority');
        $table->dropColumn('active');
    });
}
```

Al ejecutar la migración obtenemos un error dado que el registro que habíamos creado en la sección 8.2 no tiene el campo nombre y este no puede ser null. Para solucionar esto lo que hacemos es ejecutar el comando refresh, que elimina todas las tablas, las crea nuevamente y las deja sin registros.

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.02 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.02 seconds)
Migrating: 2020_03_16_230456_create_table_goals
Migrated: 2020_03_16_230456_create_table_goals (0.01 seconds)
Migrating: 2020_03_23_213233_create_goals_columns
Migrated: 2020_03_23_213233_create_goals_columns (0.01 seconds)
```

Quedando con la tabla y sus nuevas columnas como se muestra a continuación:

The screenshot shows a database schema viewer. At the top, there's a tree view under 'Tables (4)'. One branch leads to 'failed_jobs', and another leads to 'goals'. Under 'goals', there's a sub-tree for 'Columns (8)'. The columns listed are: id, created_at, updated_at, name, description, months, priority, and active. The 'name' column is highlighted with a light gray background, indicating it's selected or being focused on.

8. Modelos con Eloquent

El ORM Eloquent incluido con Laravel proporciona una implementación de ActiveRecord hermosa y simple para trabajar con su base de datos. Cada tabla de base de datos tiene un "Modelo" correspondiente que se utiliza para interactuar con esa tabla. Los modelos le permiten consultar datos en sus tablas, así como insertar nuevos registros en la tabla.

Un **ORM** es un sistema que nos permite mapear registros de la base de datos a objetos dentro de nuestro código. No es exclusivo de PHP ya que se usa mucho en los lenguajes de programación orientada a objetos.

- **make:model** crea una nueva clase para representar un modelo de Eloquent.

- Cuando creamos las bases de datos es estándar que las tablas tengan el nombre en plural pero los modelos como representan una clase que representa un objeto, tendrán su nombre en singular.
- Todos los modelos los podemos encontrar dentro de la carpeta app. Laravel no tiene carpeta models porque los creadores creen que *model* puede tener muchos significados.
- El comando tinker nos ofrece un entorno de pruebas para ver cómo funcionan las cosas que estamos haciendo. Tiene en cuenta variables de entorno, lo que inicializa Laravel y también sabe que estamos usando Eloquent.

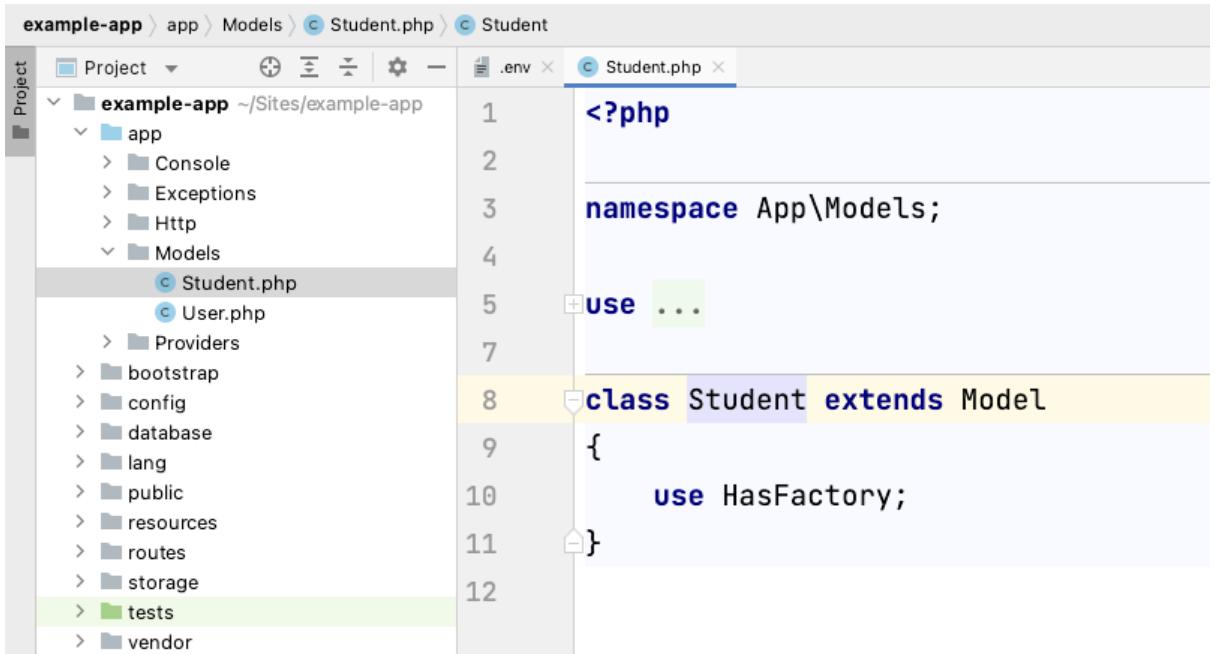
8.1. Crear un modelo con artisan

Para crear un modelo, las utilidades de artisan también vienen aprovisionadas con utilidades que facilitan la operación. A continuación se muestra el uso del comando para crear un modelo de Eloquent:

```
php artisan make:model Student
```

Obteniendo el siguiente resultado

El archivo creado se puede ver bajo la ruta app/Models.



```

example-app > app > Models > Student.php > Student
Project  Project  .env  Student.php
example-app ~Sites/example-app
  app
    Console
    Exceptions
    Http
    Models
      Student.php
      User.php
    Providers
    bootstrap
    config
    database
    lang
    public
    resources
    routes
    storage
    tests
    vendor
1  <?php
2
3  namespace App\Models;
4
5  use ...
6
7
8  class Student extends Model
9  {
10
11    use HasFactory;
12

```

Nota: En versiones de Laravel 7.X y anteriores los modelos se creaban directamente en la carpeta app

8.2. Probar el modelo con tinker

Con estos comandos ejecutados, ya es posible almacenar un elemento en la base de datos. Sin embargo como aún no tenemos interfaz gráfica, para realizar pruebas podemos usar una utilidad de *artisan* llamada tinker.

En [este enlace](#) pueden encontrar más información referente a tinker.

Para usar la utilidad de tinker es necesario ejecutar el siguiente comando:

```
php artisan tinker
```

Donde se nos habilita una entrada de comandos como la mostrada a continuación:



```
~/Sites/example-app
php artisan tinker
Psy Shell v0.11.2 (PHP 8.0.17 - cli) by Justin Hileman
```

Dentro de esta interfaz es posible escribir código de PHP y recibir la salida de la ejecución en la línea de comandos, como ejemplo realizaremos un simple mensaje por pantalla usando echo.

```
>>> echo 'Hola!, Bienvenidos a Tinker';
Hola!, Bienvenidos a Tinker
```

Lo interesante de tinker es que carga la totalidad de la configuración cargada a Laravel, carga todas las extensiones con las que venimos trabajando en el proyecto incluidos nuestro modelo y migraciones. A continuación lo que deseamos hacer es crear un objeto de tipo *Student*, con el modelo que recién creamos y almacenarlo en la base de datos.

```

└ php artisan tinker
Psy Shell v0.11.8 (PHP 8.1.5 - cli) by Justin Hileman
>>> $student = new App\Models\Student;
=> App\Models\Student {#3578}

>>> $student->document = '1234567890';
=> "1234567890"

>>> $student->code = '1234';
=> "1234"

>>> $student->name = 'John Doe';
=> "John Doe"

>>> $student->save();
=> true

```

También es posible consultar todos los registros almacenados en la base de datos desde tinker, sin embargo también se desea verlo desde el cliente de bases de datos, en el caso de este documento pgAdmin.

```

>>> App\Models\Student::all();
=> Illuminate\Database\Eloquent\Collection {#4510
    all: [
        App\Models\Student {#4512
            id: 1,
            document: "1234567890",
            code: "1234",
            name: "John Doe",
            created_at: "2022-09-01 02:40:05",
            updated_at: "2022-09-01 02:40:05",
        },
    ],
}

```

Data Output Explain Messages Notifications						
	id [PK] bigint	document character varying (255)	code character varying (255)	name character varying (255)	created_at timestamp without time zone	updated_at timestamp without time zone
1	1	1234567890	1234	John Doe	2022-09-01 02:40:05	2022-09-01 02:40:05

Nota: Al usar el método `::all()` se está usando el concepto de colecciones y todos los métodos disponibles se encuentran en la [documentación oficial](#).

9. Operaciones CRUD (Create, Read, Update, Delete)

Luego de tener creada la estructura del proyecto en la base de datos, es necesario permitir las operaciones básicas de un recurso. Debemos poder crearlo, leerlo, actualizarlo y eliminarlo según sea el caso.

Primero crearemos el controlador usando el comando que php artisan aprovisiona para ello, esto ya lo vimos en la sección 6.4 , sin embargo en este caso usaremos la bandera adicional --resource que traerá los métodos básicos del crud ya declarados por defecto. El ejemplo se muestra a continuación:

```
CARLOS CASTAÑEDA@LAPTOP-MDDN2KPQ MINGW64 /c/xampp/htdocs/curso-laravel - pruebas
$ php artisan make:controller --help
Description:
    Create a new controller class

Usage:
    make:controller [options] [--] <name>

Arguments:
    name           The name of the class

Options:
    --api          Exclude the create and edit methods from the controller.
    --force        Create the class even if the controller already exists
    -i, --invokable
    -m, --model[=MODEL]
    -p, --parent[=PARENT]
    -r, --resource Generate a resource controller class. -----
    -h, --help      Display this help message
    -q, --quiet     Do not output any message
    -V, --version   Display this application version
    --ansi         Force ANSI output
    --no-ansi      Disable ANSI output
    -n, --no-interaction
    --env[=ENV]     The environment the command should run under
    -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output,
```

```
apple:~/Sites/example-app
$ php artisan make:controller --resource StudentController
Controller created successfully.
```

Como se muestra a continuación la diferencia en el controlador con respecto a comandos que hemos utilizado anteriormente es que trae algunos métodos por defecto.

```

use Illuminate\Http\Request;
class StudentController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }
    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }
    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }
}

```

A continuación se describe brevemente cuál es la funcionalidad recomendada para cada uno de los métodos creado por defecto por Laravel:

- **index:** Aquí se despliegan todos los elementos a mostrar.
- **create:** Creará nuevos elementos.
- **store:** Guardará los elementos creados en *create*.
- **show:** Mostrará a detalle un solo elemento y por eso es que recibe un id.
- **edit:** Edita los elementos que mostramos en *show*.
- **update:** Almacena los cambios de *edit* en la base de datos.
- **destroy:** Eliminará los elementos.

Luego de crear el recurso, es necesario crear una ruta que apunte al mismo, por ello vamos a nuestro archivo web.php y escribimos lo siguiente:

```

> lang
> public
> resources
> routes
  > api.php
  > channels.php
  > console.php
  > web.php
> storage
> tests
> vendor
  .editorconfig
  .env
  .env.example
  .gitattributes
  .gitignore
  artisan
  composer.json
  composer.lock
  package.json
  phpunit.xml

```

```

2022_09_01_023630_create_students_table.php x | web.php x
4
5  /*
6  |--------------------------------------------------------------------------
7  | Web Routes
8  |--------------------------------------------------------------------------
9  |
10 | Here is where you can register web routes for your application. These
11 | routes are loaded by the RouteServiceProvider within a group which
12 | contains the "web" middleware group. Now create something great!
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19
20 Route::get('/test', [\App\Http\Controllers\TestController::class, 'index']);
21 Route::get('/testdb', [\App\Http\Controllers\TestController::class, 'testDB']);
22
23 Route::resource('students', \App\Http\Controllers\StudentController::class);
24

```

Es importante resaltar que en esta ruta no colocamos el tipo de petición HTTP, si no que colocamos que era de tipo recurso. Laravel ya reconoce que por defecto es un tipo de ruta válido y al consultar el listado de rutas disponible del aplicativo veremos un resultado como el que se muestra a continuación:

```

php artisan route:list

GET|HEAD / ..... TestController@index
POST _ignition/execute-solution ignition.executeSolution > Spatie\LaravelIgnition\ExecuteSolutionCommand@handle
GET|HEAD _ignition/health-check ignition.healthCheck > Spatie\LaravelIgnition\HealthCheckCommand@handle
POST _ignition/update-config ignition.updateConfig > Spatie\LaravelIgnition\UpdateConfigCommand@handle
GET|HEAD api/user ..... Laravel\Sanctum\Guard@user
GET|HEAD sanctum/csrf-cookie .... Laravel\Sanctum\Cookie\CookieJar@getCsrfToken
GET|HEAD students ..... StudentController@index
POST students ..... StudentController@store
GET|HEAD students/create ..... StudentController@create
GET|HEAD students/{student} ..... StudentController@show
PUT|PATCH students/{student} ..... StudentController@update
DELETE students/{student} .... StudentController@destroy
GET|HEAD students/{student}/edit ..... StudentController@edit
GET|HEAD test ..... TestController@test
GET|HEAD testdb ..... TestController@testDBconection

```

Al ser una ruta de tipo recurso, utiliza el estándar de HTTP para enlazar de manera correcta los métodos con su tipo de petición, por ejemplo para actualizar un recurso utilizaremos el método update, el cual está enlazado al recurso con el tipo de petición HTTP PUT | PATCH.

Al correr nuestro servidor de desarrollo, e ir a la ruta creada debemos obtener un resultado como el que se muestra a continuación:

```
[{"id":1,"document":"1234567890","code":"20244","name":"Cristian Echeverri","created_at":"2022-09-09T02:02:29.000000Z","updated_at":"2022-09-09T02:02:29.000000Z"}]
```

9.1. Consultar la lista de recursos

Como ya tenemos claridad de que la ruta se encuentra correctamente enlazada, crearemos una nueva vista para mostrar a los estudiantes del proyecto de manera ordenada y limpia con HTML. Para ello crearemos una carpeta dentro de las vistas y dentro crearemos el archivo index.blade.php.



The screenshot shows a code editor interface with a project structure on the left and the content of the `index.blade.php` file on the right. The project structure includes `example-app`, `resources`, `views`, and `students`. Inside `views`, there are files `index.blade.php`, `test.blade.php`, and `welcome.blade.php`. The `index.blade.php` file contains the following code:

```
<!doctype html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Estudiantes</title>
</head>
<body>
    <h1 class="alert alert-success center">Students list</h1>
</body>
</html>
```

Ahora en nuestro método `index` del controlador creado, retornaremos un objeto de tipo vista en donde enviamos como parámetro adicional un arreglo asociativo con todos los estudiantes almacenados en base de datos.

```

example-app > app > Http > Controllers > StudentController.php > StudentController > index

Project
example-app ~ /Sites/example-app
  > app
    > Console
    > Exceptions
    > Http
      > Controllers
        > Controller.php
        > StudentController.php
        > TestController.php
      > Middleware
        > Kernel.php
      > Models
      > Providers
      > bootstrap
      > config
      > database
      > lang
      > public
      > resources
      > routes
      > storage
      > tests
      > vendor

```

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Models\Student;
6 use Illuminate\Http\Request;
7
8 class StudentController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      *
13      * @return \Illuminate\Http\Response
14     */
15     public function index()
16     {
17         //return Student::all();
18         return view( view: 'students.index', data: [
19             'students' => Student::all()
20         ]);
21     }
}

```

Nota: Es importante resaltar que como nuestra vista se encuentra dentro de una carpeta deberemos dar la ruta exacta de la misma. La separación de carpetas se simboliza con un punto.

En la vista de blade tendremos un código como el que se muestra a continuación:

```

Proj... > Student.php > index.blade.php
public
resources
  > css
  > js
  > views
    > students
      > index.blade.php
      > test.blade.php
      > welcome.blade.php
  > routes
    > api.php
    > channels.php
    > console.php
    > web.php
  > storage
  > tests
  > vendor
    .editorconfig
    .env
    .env.example
    .gitattributes
    .gitignore
    artisan
    composer.json
    composer.lock
    package.json
    phpunit.xml
    README.md
    vite.config.js
External Libraries

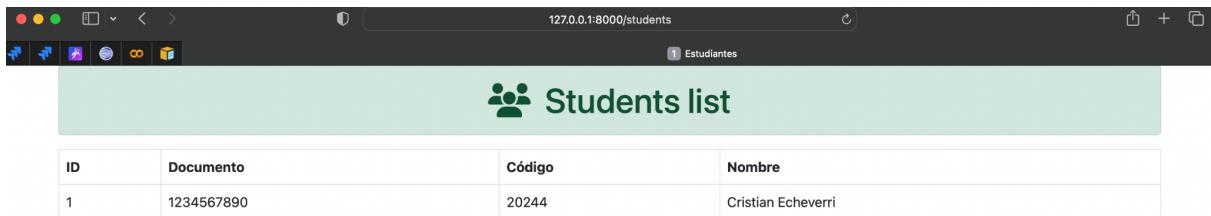
```

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
          integrity="sha384-EVSTQN3+azpR1zqjWtIzP+g1N7Q8hWV+rT1CSXe8zQf+a4&lt;!-->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.2.0/css/all.min.css">
    <title>Estudiantes</title>
  </head>
  <body>
    <main class="container">
      <h1 class="alert alert-success text-center"><i class="fa-solid fa-users"></i> Students list</h1>
      <table class="table table-bordered table-striped">
        <thead>
          <th>ID</th>
          <th>Documento</th>
          <th>Código</th>
          <th>Nombre</th>
        </thead>
        <tbody>
          @foreach($students as $student)
            <tr>
              <td>{$student->id}</td>
              <td>{$student->document}</td>
              <td>{$student->code}</td>
              <td>{$student->name}</td>
            </tr>
          @endforeach
        </tbody>
      </table>
      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
             integrity="sha384-MrcW6ZMFYlzcLA8NlUpF0sA7MsXsP1UYJoMp4YLEuNSfAP+JcXn/tWtiaxVXM&lt;!-->
             crossorigin="anonymous"></script>
    </main>
  </body>
</html>

```

Obteniendo un resultado como el que se muestra a continuación:



A screenshot of a web browser window titled "127.0.0.1:8000/students". The title bar also shows "1 Estudiantes". The main content area has a green header bar with a user icon and the text "Students list". Below is a table with four columns: "ID", "Documento", "Código", and "Nombre". A single row is visible with ID 1, Documento 1234567890, Código 20244, and Nombre Cristian Echeverri.

ID	Documento	Código	Nombre
1	1234567890	20244	Cristian Echeverri

9.2. Blade Layout

A lo largo del proyecto hemos copiado y pegado nuestras vistas, sin embargo esto genera código duplicado en cada una de ellas, situación que pretendemos evitar. Por lo anterior, en Laravel podemos crear en un sólo lugar código que es igual en nuestras vistas y posteriormente extenderlo y añadirle las cosas específicas de cada vista. Una mejor práctica para evitar esta repetición es crear layouts y extenderlos. De esta manera el layout tendrá el contenido que siempre se repite y los hijos el código específico de ellos.

- **@yield** marca la parte en donde irá el código de los hijos que extiendan o hereden del layout.
- En las vistas hijas se utiliza **@section** para decir que esa parte del código es la que concuerda con el **@yield** del layout.

En este curso usaremos como framework frontend a [bootstrap](#) del que no se hará énfasis ni se darán detalles de su implementación por lo que se recomienda que si no se tiene conocimiento del mismo se consulte en su [documentación oficial](#) y se profundice su conocimiento.

Para nuestro layout usaremos el [starter template](#) de bootstrap ofrecido directamente en su documentación oficial.

9.2.1. Crear el layout

Para crear el layout en nuestro proyecto, en la sección de vistas crearemos una nueva carpeta llamada `layouts`, y para el primero de ellos usaremos un archivo llamado `base.blade.php`, obteniendo un resultado como el que se muestra a continuación:

```

<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-gQ29O9xO6K47jO0FzWpD/P6R�qL2v9Jml1yGnQXe7zJLH0/0oZuL9JN9Jg==" crossorigin="anonymous" />

    <title>@yield('title')</title>
</head>
<body>
    @yield('container')

```

Vemos que dentro del div con clase container agregamos la etiqueta `@yield`, que es donde vendrá el contenido de aquellas vistas que extiendan este layout. Para más información visitar la [documentación oficial](#).

9.2.2. Usar el layout

Para probar el funcionamiento de este template, en nuestro archivo `index.blade.php` que se encuentra dentro de la carpeta students, cambiaremos el contenido y lo dejaremos como se muestra a continuación:

```

@extends('base')
@section('container')
    <main class="container">
        <h1 class="alert alert-success text-center"> <i class="fa-solid fa-users"></i> Students list</h1>
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Document</th>
                    <th>Code</th>
                    <th>Name</th>
                    <th>Last Name</th>
                    <th>Birth Date</th>
                </tr>
            </thead>
            <tbody>
                @foreach($students as $student)
                    <tr>
                        <td>$student->id</td>
                        <td>$student->document</td>
                        <td>$student->code</td>
                        <td>$student->name</td>
                        <td>$student->lastname</td>
                        <td>$student->birth_date</td>
                    </tr>
                @endforeach
            </tbody>
        </table>
    </main>
@endsection

```

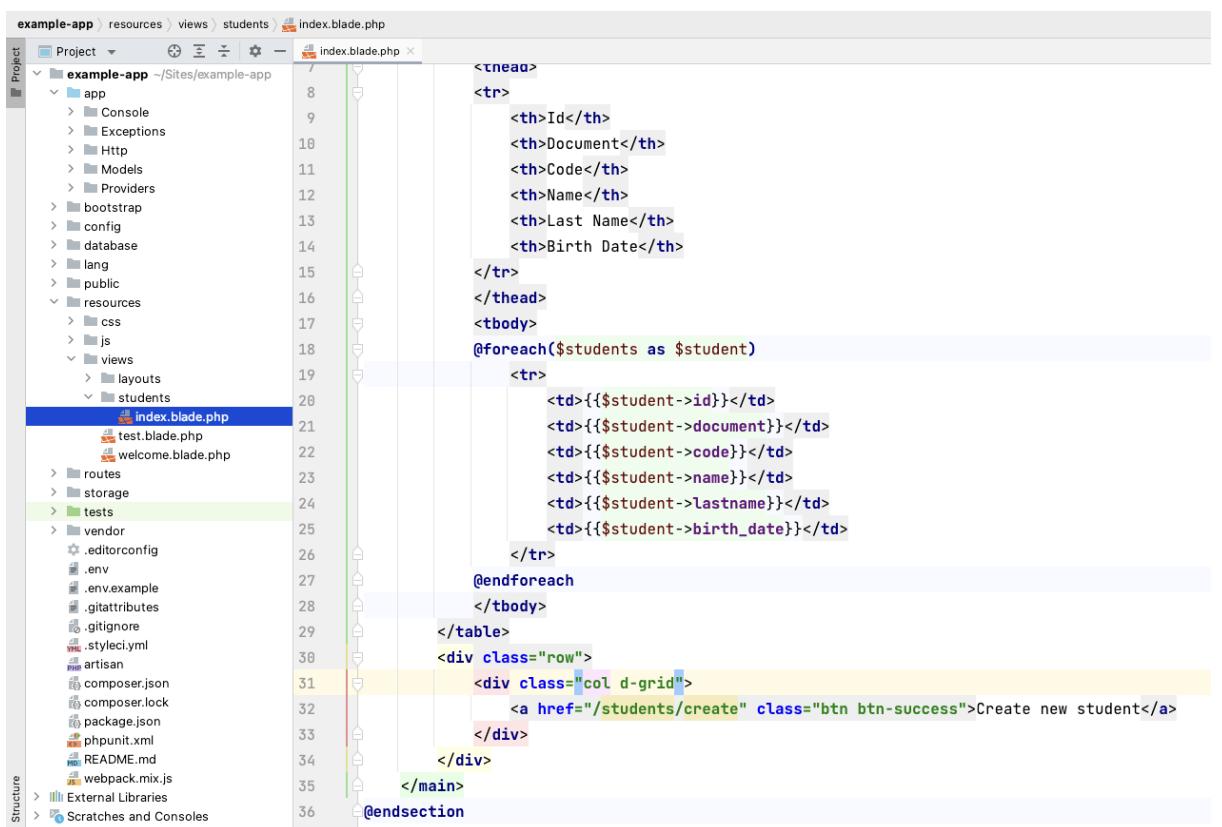
Como se puede apreciar, en la primera línea usamos la etiqueta `@extends`, en donde mencionamos la ubicación y el nombre del layout del cual queremos extender, además con la etiqueta `@section` especificamos el lugar

dentro del layout donde queremos colocar nuestro contenido. El valor de `@section` debe coincidir con el valor de `@yield`.

9.3. Insertar Recurso

Para insertar un recurso es necesario crear una nueva vista, y esta puede ser ser referenciada desde nuestra vista actual a través de un ancla HTML. Recordemos que al momento de crear el controlador tipo recurso se me habilita la ruta `create` que será la encargada de retornar la vista con el formulario para la creación del nuevo recurso.

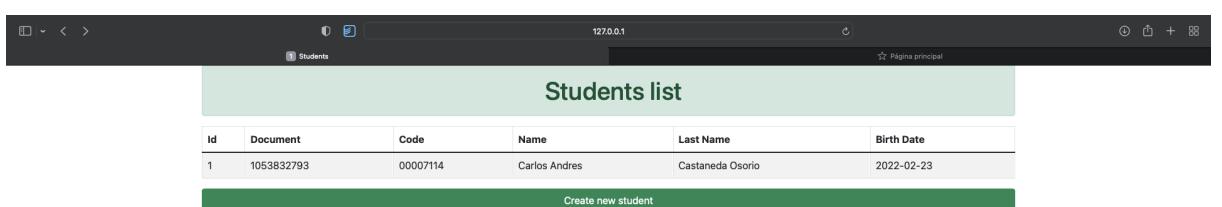
Nuestro `index.blade.php` que se encuentra en la carpeta `students` quedaría de la siguiente manera:



The screenshot shows the PhpStorm IDE interface. On the left is the project structure sidebar with the following tree:
example-app / resources / views / students / index.blade.php
 |- Project
 |- example-app (~Sites/example-app)
 |- app
 | |- Console
 | |- Exceptions
 | |- Http
 | |- Models
 | |- Providers
 | |- bootstrap
 | |- config
 | |- database
 | |- lang
 | |- public
 |- resources
 |- css
 |- js
 |- views
 |- layouts
 |- students
 |- index.blade.php
 |- test.blade.php
 |- welcome.blade.php
 |- routes
 |- storage
 |- tests
 |- vendor
 |- editorconfig
 |- .env
 |- .env.example
 |- .gitattributes
 |- .gitignore
 |- .styleci.yml
 |- artisan
 |- composer.json
 |- composer.lock
 |- package.json
 |- phpunit.xml
 |- README.md
 |- webpack.mix.js
 |- External Libraries
 |- Scratches and Consoles

The code editor shows the `index.blade.php` file content:<thead><tr><th>Id</th><th>Document</th><th>Code</th><th>Name</th><th>Last Name</th><th>Birth Date</th></tr></thead><tbody>/foreach(\$students as \$student)
<tr><td>{{\$student->id}}</td><td>{{\$student->document}}</td><td>{{\$student->code}}</td><td>{{\$student->name}}</td><td>{{\$student->lastname}}</td><td>{{\$student->birth_date}}</td></tr>/endforeach</tbody></table><div class="row"><div class="col d-grid">Create new student</div></div></main>/endsection

Obteniendo el siguiente resultado:



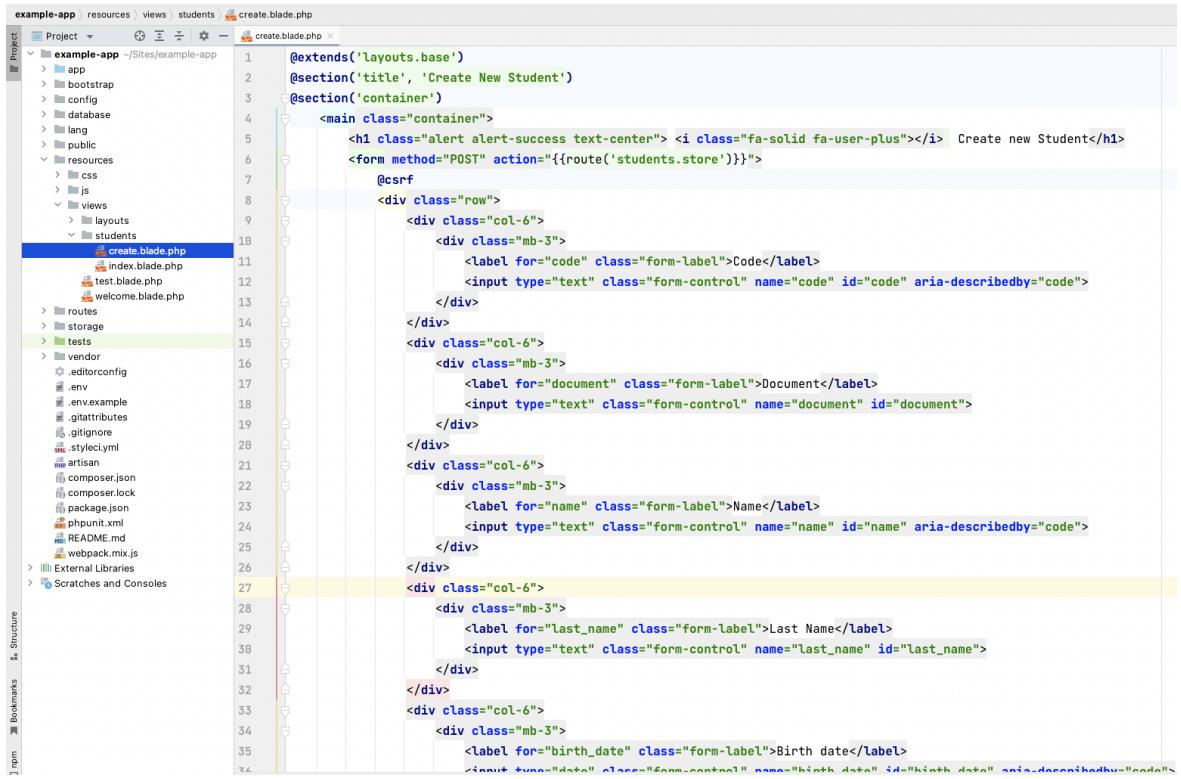
The screenshot shows a web browser window with the following details:
Title bar: Students
Address bar: 127.0.0.1
Page content:

Students list

ID	Document	Code	Name	Last Name	Birth Date
1	1053832793	00007114	Carlos Andres	Castaneda Osorio	2022-02-23

A green button at the bottom of the table says "Create new student".

A continuación se debe crear la plantilla del formulario, en la carpeta de student agregamos a create.blade.php, con su respectivo código HTML, como se muestra a continuación:



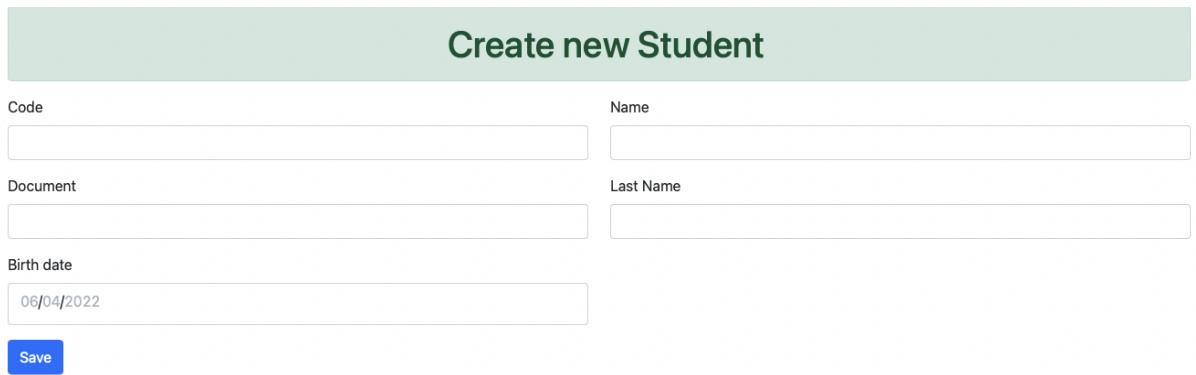
```

example-app > resources > views > students > create.blade.php
Project  example-app ~/Sites/example-app
1  @extends('layouts.base')
2  @section('title', 'Create New Student')
3  @section('container')
4      <main class="container">
5          <h1 class="alert alert-success text-center"> <i class="fa-solid fa-user-plus"></i> Create new Student</h1>
6          <form method="POST" action="{{route('students.store')}}">
7              @csrf
8              <div class="row">
9                  <div class="col-6">
10                     <div class="mb-3">
11                         <label for="code" class="form-label">Code</label>
12                         <input type="text" class="form-control" name="code" id="code" aria-describedby="code">
13                     </div>
14                     <div class="col-6">
15                         <div class="mb-3">
16                             <label for="document" class="form-label">Document</label>
17                             <input type="text" class="form-control" name="document" id="document">
18                         </div>
19                         <div class="col-6">
20                             <div class="mb-3">
21                                 <label for="name" class="form-label">Name</label>
22                                 <input type="text" class="form-control" name="name" id="name" aria-describedby="code">
23                             </div>
24                         </div>
25                         <div class="col-6">
26                             <div class="mb-3">
27                                 <label for="last_name" class="form-label">Last Name</label>
28                                 <input type="text" class="form-control" name="last_name" id="last_name">
29                             </div>
30                         </div>
31                         <div class="col-6">
32                             <div class="mb-3">
33                                 <label for="birth_date" class="form-label">Birth date</label>
34                                 <input type="date" class="form-control" name="birth_date" id="birth_date" aria-describedby="code">
35                             </div>
36                     </div>
37                 </div>
38             </div>
39         </main>
40     </body>
41 </html>

```

Nota: En el atributo “action” del formulario es posible colocar el método route como se muestra en el ejemplo o colocar directamente la ruta

Obteniendo el siguiente resultado:



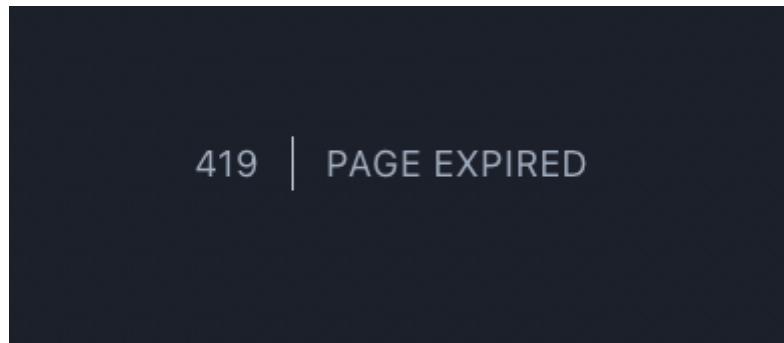
Create new Student

Code	Name
<input type="text"/>	<input type="text"/>
Document	Last Name
<input type="text"/>	<input type="text"/>
Birth date	<input type="date" value="06/04/2022"/>
<input type="button" value="Save"/>	

Con esto ya estamos casi listos para almacenar filas de estudiantes dentro de nuestra base de datos, sin embargo al hacer una petición POST a nuestro proyecto hay que tener en cuenta una propiedad adicional.

9.4. CSRF (Cross Site Request Forgery)

Laravel trae por defecto activado el [middleware](#) de [cross-site request forgery](#) que evita que se realicen peticiones externas al sistema de información en todos los métodos HTTP diferentes a GET, en nuestro formulario debemos decirle que la petición se realiza internamente desde nuestro sistema para que esta sea aceptada. El error que arroja el navegador al intentar enviar el formulario con datos es el HTTP 419, como se muestra a continuación:



Para adicionar este parámetro a nuestro formulario escribimos la anotación `@csrf` dentro del formulario como se muestra a continuación:

```
<h1 class="alert alert-success text-center"> <i class="fa-solid fa-user-plus"></i> Create new Student</h1>
<form method="POST" action="{{route('students.store')}}">
    @csrf
    <div class="row">
        <div class="col-6">
            <div class="mb-3">
                <label for="code" class="form-label">Code</label>
                <input type="text" class="form-control" id="code" aria-describedby="code">
            </div>
```

Al actualizar la página vamos a verificar que nuestro código HTML tiene un nuevo campo, este es un token que nos identifica ante el sistema de información y nos permite realizar peticiones sin que sean rechazadas.

The screenshot shows a browser window with the title "Create new Student". The form has fields for "Code", "Name", "Document", "Last Name", and "Birth date" (with a value of "06/04/2022"). Below the form is a "Save" button. To the right of the browser window is the DevTools Elements tab, which displays the generated HTML code. A blue highlight is applied to the `<input type="hidden" name=" _token" value="81BarvtxikmHZVV3C123gCN0TcvT TXVNU0ip" >` line, indicating the presence of the CSRF token.

```
<!DOCTYPE html>
<html lang="en">
    <head></head>
    <body>
        <div class="container">
            <h1 class="alert alert-success text-center"> <i class="fa-solid fa-user-plus"></i> Create new Student</h1>
            <form method="POST" action="http://127.0.0.1:8000/students">
                <input type="hidden" name=" _token" value="81BarvtxikmHZVV3C123gCN0TcvT TXVNU0ip" > = $0
                <div class="row">
                    <div class="col-6">
                        <div class="mb-3">
                            <label for="code" class="form-label">Code</label>
                            <input type="text" class="form-control" id="code" aria-describedby="code">
                        </div>
                    </div>
                </div>
                <div class="row">
                    <div class="col-6">
                        <button type="submit" class="btn btn-primary">Save</button>
                    </div>
                </div>
            </form>
        </div>
        <!-- Optional JavaScript; choose one of the two! -->
        <!-- Option 1: Bootstrap Bundle with Popper -->
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-kZSkQGln4gmtz2MlQnikT1wXgYs0g+OMhuP+LkhsqENB00LRh5q+8nIov4+1p" crossorigin="anonymous"></script>
```

Con esto solucionado, procedemos a terminar nuestra petición de almacenamiento de un estudiante.

9.5. Finalizando el método store

Luego de recibir la petición correctamente, es hora de persistir la información que recibimos en la petición, para ello recordamos que el método que estamos llamando con nuestro formulario es el que se encuentra bajo la ruta **/students** con el tipo de petición HTTP POST, en este caso mapeado automáticamente, que tiene el nombre **store**.

Inicialmente vamos a obtener todos los datos del formulario para estar seguros de estar recibiendo todos los campos en la petición HTTP, para ello escribimos el siguiente código en el método store:

```
/**  
 * Store a newly created resource in storage.  
 *  
 * @param \Illuminate\Http\Request $request  
 * @return \Illuminate\Http\Response  
 */  
public function store(Request $request)  
{  
    dd(...vars: $request->all());  
}
```

Si todo se encuentra correcto, al llenar el formulario y enviar los datos deberíamos ver un resultado como el mostrado a continuación:

```
^ array:6 [▼  
  "_token" => "81ABarwtxikmlHZVYR3Cl23gCN0TcvTTXVhUo1p"  
  "code" => "0001"  
  "document" => "123456"  
  "name" => "Carlos"  
  "last_name" => "Castañeda"  
  "birth_date" => "1994-02-23"  
]
```

Teniendo claro que todos los campos se están recibiendo de manera correcta, es momento de crear un nuevo estudiante en base de datos a partir de los datos recibidos, para ello en el método store agregamos el siguiente código:

```

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    //Create a new student object
    $student = new Student;

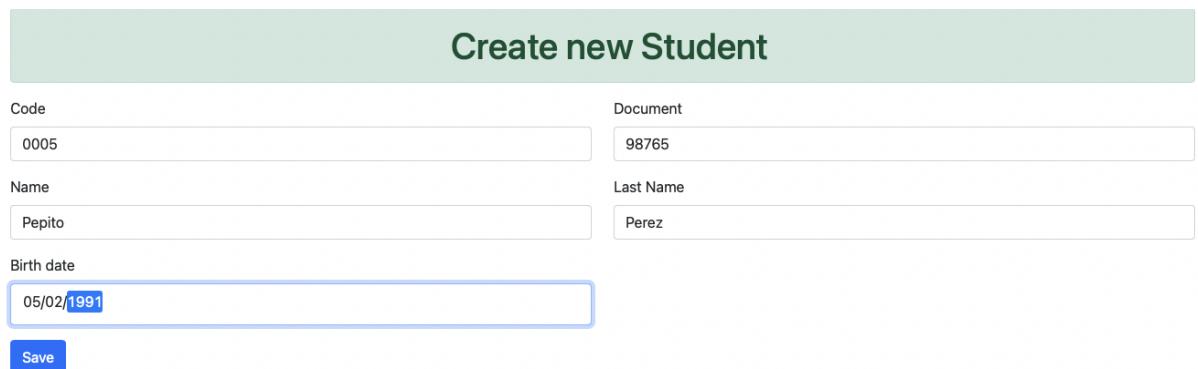
    //Set attributes values with the info received from the form
    $student->code = $request->input('code');
    $student->document = $request->input('document');
    $student->name = $request->input('name');
    $student->last_name = $request->input('last_name');
    $student->birth_date = $request->input('birth_date');

    //Save the record in the database
    $student->save();
    //use Symfony\Component\HttpFoundation\Response;
    return redirect(to: '/students', status: Response::HTTP_CREATED);
}

```

Nota: Es importante resaltar que el método usado para obtener los valores recibidos del formulario es input y no query como lo habíamos usado anteriormente. Ver [documentación oficial](#) de Request.

Probamos nuevamente el formulario para validar que la información se esté almacenando correctamente en la base de datos obteniendo el siguiente resultado:



Create new Student	
Code	Document
0005	98765
Name	Last Name
Pepito	Perez
Birth date	05/02/1991
<input type="button" value="Save"/>	

Students list					
Id	Document	Code	Name	Last Name	Birth Date
1	1053832793	00007114	Carlos Andres	Castaneda Osorio	2022-02-23
2	98765	005	Pepito	Perez	1991-02-05

Con esto ya queda funcionando el ingreso de información a nuestro proyecto, las validaciones de que los datos son correctos las realizaremos puntos más adelante del documento.

9.6. Actualizar recurso

Luego de insertar recursos a nuestra base de datos, el paso siguiente es poder actualizarlos, para ello tenemos disponibles los métodos ***edit*** y ***update***, para actualizar laravel propone usar los métodos HTTP PUT/PATCH.

```
ls ~/Sites/example-app  
php artisan route:list
```

Method	URI	Controller	Action
GET HEAD	/	...	TestController@Index
POST	_ignition/execute-solution	ignition.executeSolution > Spatie\LaravelIgnition\ExecuteSolutionController@execute	
GET HEAD	_ignition/health-check	ignition.healthCheck > Spatie\LaravelIgnition\HealthCheckController@check	
POST	_ignition/update-config	ignition.updateConfig > Spatie\LaravelIgnition\UpdateConfigController@update	
GET HEAD	api/user	...	Laravel\Sanctum\CsrfCookieController@show
GET HEAD	sanctum/csrf-cookie	...	
GET HEAD	students	students	students.index > StudentController@index
POST	students	students	students.store > StudentController@store
GET HEAD	students/create	students	students.create > StudentController@create
GET HEAD	students/{student}	students	students.show > StudentController@show
PUT PATCH	students/{student}	students	students.update > StudentController@update
DELETE	students/{student}	students	students.destroy > StudentController@destroy
GET HEAD	students/{student}/edit	students	students.edit > StudentController@edit
GET HEAD	test	test	TestController@testDBConnection
GET HEAD	testdb	testdb	TestController@testDBConnection

Para empezar, en nuestra tabla que listamos todos los estudiantes agregaremos una columna con un botón que nos permita realizar la edición de un recurso.

```

6 <table class="table table-bordered table-striped">
7   <thead>
8     <tr>
9       <th>Id</th>
10      <th>Document</th>
11      <th>Code</th>
12      <th>Name</th>
13      <th>Last Name</th>
14      <th>Birth Date</th>
15      <th>Edit</th>
16    </tr>
17  </thead>
18  <tbody>
19  @foreach($students as $student)
20    <tr>
21      <td>{{$student->id}}</td>
22      <td>{{$student->document}}</td>
23      <td>{{$student->code}}</td>
24      <td>{{$student->name}}</td>
25      <td>{{$student->lastname}}</td>
26      <td>{{$student->birth_date}}</td>
27      <td><a href="{{route('students.edit', $student)}}>Edit</a></td>
28    </tr>

```

Es importante resaltar que la referencia a donde apunta el botón es la ruta definida en Laravel que apunta a la petición GET del método edit, el cual quedará de la siguiente manera:

```

/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    $student = Student::find($id);
    return view( view: 'students.edit', data: [
        'student' => $student
    ]);
}

```

En este método obtenemos el recurso de la base de datos basados en el ID recibido en la URL y redireccionamos al template de **edit** enviando el estudiante consultado.

A continuación crearemos el template de edit que copiaremos del template de adicionar, con algunos cambios como se muestran a continuación:



```
1 @extends('layouts.base')
2 @section('title', 'Create New Student')
3 @section('container')
4     <main class="container">
5         <h1 class="alert alert-success text-center">  Edit Student</h1>
6         <form method="POST" action="{{route('students.update', $student)}}>
7             @csrf
8             <div class="row">
9                 <div class="col-6">
10                     <div class="mb-3">
11                         <label for="code" class="form-label">Code</label>
12                         <input type="text" class="form-control" name="code" id="code" aria-describedby="code" value="{{ $student->code }}>
13                     </div>
14                 </div>
15                 <div class="col-6">
16                     <div class="mb-3">
17                         <label for="document" class="form-label">Document</label>
18                         <input type="text" class="form-control" name="document" id="document" value="{{ $student->document }}>
19                     </div>
20                 </div>
21                 <div class="col-6">
22                     <div class="mb-3">
23                         <label for="name" class="form-label">Name</label>
24                         <input type="text" class="form-control" name="name" id="name" value="{{ $student->name }}>
25                     </div>
26                 </div>
27                 <div class="col-6">
28                     <div class="mb-3">
29                         <label for="lastname" class="form-label">Last Name</label>
30                         <input type="text" class="form-control" name="lastname" id="lastname" value="{{ $student->lastname }}>
31                     </div>

```

A pesar de que en el estándar HTTP para realizar actualizaciones de recursos se recomienda el uso de los métodos PUT/PATH, nuestro elemento html **form** no soporta dicho valor en su atributo **method**, es por ello que nos toca usar una anotación adicional para especificar que lo que se desea realizar es un método HTTP de tipo **PUT**, este se agrega justo debajo del **@csrf** y se ve como se muestra a continuación:



```
<form method="POST" action="{{route('students.update', $student)}}>
    @csrf
    @method('PUT')
    <div class="row">
        <div class="col-6">
            <div class="mb-3">
                <label for="code" class="form-label">Code</label>
                <input type="text" class="form-control" name="code" id="code" aria-describedby="code" value="{{ $student->code }}>
            </div>
        </div>
```

Por último debemos modificar el método **update**, en donde debemos obtener el recurso de la base de datos, obtener los valores recibidos del formulario y actualizar los valores del recurso y finalmente persistir el recurso actualizado.

```

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    //Create a new student object
    $student = Student::find($id);

    //Set attributes values with the info received from the form
    $student->code = $request->input('code');
    $student->document = $request->input('document');
    $student->name = $request->input('name');
    $student->lastname = $request->input('lastname');
    $student->birth_date = $request->input('birth_date');

    //Save the record in the database
    $student->save();
    //use Symfony\Component\HttpFoundation\Response;
    return redirect(to: '/students');
}

```

Con esto ya es posible actualizar los valores de un recurso como se muestra a continuación:

The screenshot shows a web-based form titled "Edit Student". The form has two columns. The left column contains fields for "Code" (with value "00005") and "Name" (with value "Pepito Andres"). The right column contains fields for "Document" (with value "9876523") and "Last Name" (with value "Perez Diaz"). Below these fields is a "Birth date" field containing "05/02/1991". At the bottom of the form is a blue "Update" button.

Students list						
Id	Document	Code	Name	Last Name	Birth Date	Edit
1	1053832793	00007114	Carlos Andres	Castaneda Osorio	2022-02-23	<button>Edit</button>
2	9876523	00005	Pepito Andres	Perez Diaz	1991-02-05	<button>Edit</button>
Create new student						

9.7.Eliminar un recurso

Ahora que tenemos la posibilidad de crear y editar recursos en el sistema, la única acción básica que falta para completar el CRUD es eliminarlo, para ello debemos usar el método HTTP delete que apunta a la ruta que por defecto se crea para tal fin como se muestra a continuación:

```
laragon -> Sites/example-app [master] 11:16:59 C
php artisan route:list
```

METHOD	URI	NAME	DESCRIPTION
GET HEAD	/		TestController@index
POST	_ignition/execute-solution		ignition.executeSolution : Spatie\Ignition\ExecuteSolutionController
GET HEAD	_ignition/health-check		ignition.healthCheck : Spatie\Ignition\HealthCheckController
POST	_ignition/update-config		ignition.updateConfig : Spatie\Ignition\UpdateConfigController
GET HEAD	api/user		Laravel\Sanctum\CSRFCookieController@show
GET HEAD	sanctum/csrf-cookie		students.index : StudentController@index
GET HEAD	students		students.store : StudentController@store
POST	students		students.create : StudentController@create
GET HEAD	students/create		students.show : StudentController@show
GET HEAD	students/{student}		students.update : StudentController@update
PUT PATCH	students/{student}		students.destroy : StudentController@destroy
DELETE	students/{student}		students.edit : StudentController@edit
GET HEAD	test		TestController@testDBConnection
GET HEAD	testdb		

Como se puede apreciar en la lista de rutas, el método al que apunta esta ruta es **destroy**.

Para eliminar el recurso crearemos una nueva columna en la tabla con el botón de eliminar. El código quedaría como se muestra a continuación.

```

index.blade.php
15 <th>Edit</th>
16 <th>Delete</th>
17 </tr>
18 </thead>
19 <tbody>
20 @foreach($students as $student)
21 <tr>
22 <td>{{$student->id}}</td>
23 <td>{{$student->document}}</td>
24 <td>{{$student->code}}</td>
25 <td>{{$student->name}}</td>
26 <td>{{$student->lastname}}</td>
27 <td>{{$student->birth_date}}</td>
28 <td><a href="{{route('students.edit', $student)}}>Edit</a></td>
29 <td>
30 <form action="{{route('students.destroy', $student)}}" method="POST">
31 @csrf
32 @method('DELETE')
33 <input
34 type="submit"
35 value="Delete"
36 class="btn btn-danger"
37 onclick="return confirm('¿Delete {{$student->name}}?')">
38 </input>
39 </form>
40 </td>
41 </tr>

```

Obteniendo el resultado que se muestra a continuación:

Students list							
Id	Document	Code	Name	Last Name	Birth Date	Edit	Delete
1	1053832793	00007114	Carlos Andres	Castaneda Osorio	2022-02-23	<button>Edit</button>	<button>Delete</button>
2	9876523	00005	Pepito Andres	Perez Diaz	1991-02-05	<button>Edit</button>	<button>Delete</button>

Create new student

Nota: Es importante resaltar que se utilizó una nueva manera de invocar a una ruta HTTP en el action del formulario, para más información sobre esta manera de invocación consultar la [documentación oficial](#).

Luego de tener la tabla completa, con nuestras acciones y el uso de Javascript para la confirmación de la petición, es necesario en el controlador realizar la acción para que el recurso sea eliminado de la base de datos en el método destroy. El código necesario para realizar la eliminación se muestra a continuación:

```

108  /**
109   * Remove the specified resource from storage.
110  *
111  * @param int $id
112  * @return \Illuminate\Http\Response
113  */
114 public function destroy(Student $student)
115 {
116     $student->delete();
117
118     return back();
119 }

```

Luego de tener implementada toda nuestra lógica de eliminación de recurso, procedemos a probarla intentando eliminar “Pepito Andres Perez”, en donde obtenemos el siguiente resultado:

The screenshot shows a 'Students list' page with two entries:

ID	Document	Code	Name	Last Name	Birth Date	Edit	Delete
1	1053832793	00007114	Carlos Andres	Castaneda Osorio	2022-02-23	<button>Edit</button>	<button>Delete</button>
2	9876523	00005	Pepito Andres	Perez Diaz	1991-02-05	<button>Edit</button>	<button>Delete</button>

A green button at the bottom says "Create new student". A modal dialog box is open, asking "¿Delete Pepito Andres?". It has "Cancelar" and "OK" buttons.

Si le damos clic en cancelar la acción no se ejecuta y el estudiante no es eliminado, pero si hacemos clic en aceptar el recurso el eliminado y regresamos nuevamente a la página donde nos encontrábamos ya con el recurso eliminado como se muestra a continuación:

The screenshot shows the same 'Students list' page, but now it only displays one entry:

ID	Document	Code	Name	Last Name	Birth Date	Edit	Delete
1	1053832793	00007114	Carlos Andres	Castaneda Osorio	2022-02-23	<button>Edit</button>	<button>Delete</button>

A green button at the bottom says "Create new student".

9.8. Validación de datos

En las secciones anteriores ya completamos las acciones básicas a realizar con un recurso, sin embargo confiábamos en los datos ingresados por el cliente y no realizamos ninguna validación de los mismos.

En Laravel podemos validar los datos que recibimos del lado del servidor, para ello vamos a los métodos en donde se recibe información y se adiciona el siguiente código:

```
$request->validate( rules: [
    'code' => 'required',
    'document' => ['required', 'integer', 'min:5'],
    'name' => 'required|min:5|max:30',
    'lastname' => ['required'],
    'birth_date' => ['required', 'date']
]);
```

Con este código verificamos que los elementos sean del tipo adecuado y que los campos que son obligatorios contengan algún valor. Tenemos muchas validaciones disponibles, para verificar cuales son se puede consultar la [documentación oficial de laravel](#).

Con esto, al momento de enviar el formulario, no guardará datos incorrectos, sin embargo lo único que sucede es que se recarga la página y no se le muestra al usuario el resultado de la operación, para ello vamos a agregar el código que permite mostrar los errores a los usuarios:

```
create.blade.php
39  </div>
40  @if($errors->any())
41      <div class="alert alert-danger">
42          <ul>
43              @foreach($errors->all() as $error)
44                  <li>{{$error}}</li>
45              @endforeach
46          </ul>
47      </div>
48  @endif
49  <button type="submit" class="btn btn-primary">Save</button>
50  </form>
```

Con estas dos porciones de código configuradas correctamente, intentamos agregar un registro con datos inválidos a nuestro sistema y obtenemos la siguiente información:

The screenshot shows a 'Create new Student' form with the following fields and their current values:

- Code**: An empty input field.
- Document**: An empty input field.
- Name**: An empty input field.
- Last Name**: An empty input field.
- Birth date**: A dropdown menu set to "07/04/2022".

A red error message box contains the following list of validation errors:

- The code field is required.
- The document field is required.
- The name field is required.
- The lastname field is required.
- The birth date field is required.

A blue 'Save' button is located at the bottom left of the form.

Los mensajes de error recibidos se encuentran en inglés y corresponden a la configuración por defecto de Laravel, para colocar nuestros mensajes personalizados, agregamos un parámetro adicional a la función de validación como se muestra a continuación:

```

public function store(Request $request)
{
    $request->validate([
        'code' => 'required',
        'document' => ['required', 'integer', 'min:5'],
        'name' => 'required|min:5|max:30',
        'lastname' => ['required'],
        'birth_date' => ['required', 'date']
    ], ...params: [
        'code.required' => 'The code is required. Try using SIA code',
        'document.integer' => 'The document should contains only numbers'
    ]);
}

```

Obteniendo el resultado que se muestra a continuación:

The screenshot shows a form titled "Create new Student" with fields for Code, Document, Name, Last Name, and Birth date. The Birth date field contains "07/04/2022". A red error box at the bottom lists validation errors:

- The code is required. Try using SIA code
- The document should contains only numbers
- The name field is required.
- The lastname field is required.
- The birth date field is required.

A blue "Save" button is visible at the bottom left.

Nota: Para ver más documentación de cómo obtener los errores en los templates de blade se recomienda visitar la [documentación oficial](#).

9.9. Validación a través de request

A pesar de que en la manera mostrada anteriormente los datos son validados, si quisiéramos realizar la misma validación para el método editar, deberíamos copiar y pegar el código (**MALA PRÁCTICA!**), por ello existe una manera más elegante de realizar validaciones.

Lo primero que haremos será un nuevo [Request](#) (Tipo de clase predefinida por Laravel). Para ello utilizaremos las utilidades de la línea de comandos de artisan como se muestra a continuación:

```

example-app > app > Http > Requests > StudentRequest.php
Project 1 <?php
example-app ~Sites/example-app
app
> Console
> Exceptions
> Http
> Controllers
> Middleware
> Requests
StudentRequest.php
Kernel.php
> Models
> Providers
bootstrap
> config
> database
> factories
1
2
3
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;
4
5
6
7 class StudentRequest extends FormRequest
8 {
9 /**
10 * Determine if the user is authorized to make this request.
\App\Http\Requests > StudentRequest

```

Terminal: Local Local (2) +

Apple > ~Sites/example-app

php artisan make:request StudentRequest

Request created successfully.

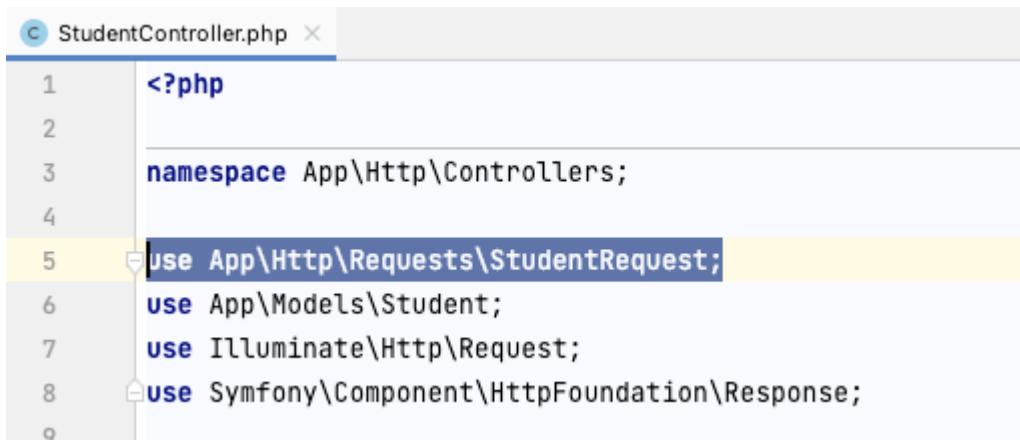
Con esto se crea una nueva clase en la carpeta app/Http/Request con el nombre asignado en el comando, el nombre puede ser cualquiera, sin embargo, por convención se recomienda colocar la palabra Request seguida por el nombre del modelo. Esta clase la debemos configurar de la siguiente manera:

```

StudentRequest.php
13 */
14 public function authorize()
15 {
16     return true;
17 }
18
19 /**
20 * Get the validation rules that apply to the request.
21 *
22 * @return array
23 */
24 public function rules()
25 {
26     return [
27         'code' => 'required',
28         'document' => ['required', 'integer', 'min:5'],
29         'name' => 'required|min:5|max:30',
30         'lastname' => ['required'],
31         'birth_date' => ['required', 'date']
32     ];
33 }
34

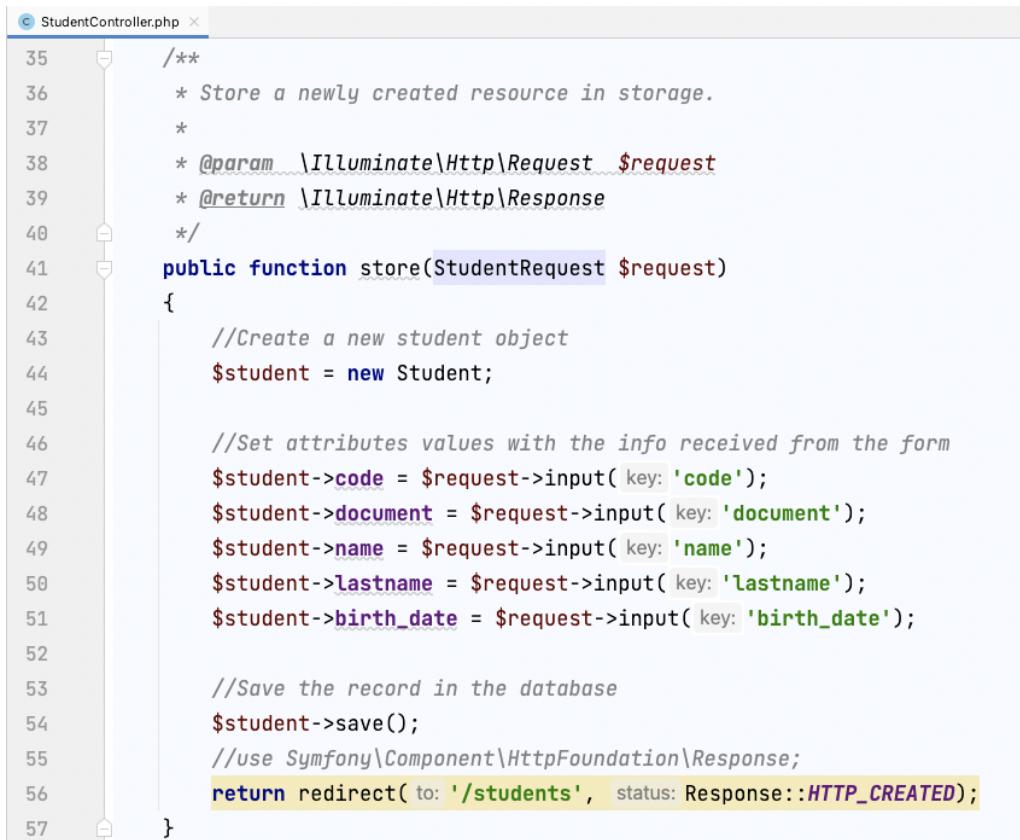
```

Con la clase configurada, es momento de decirle a nuestro controlador que la utilice en los métodos, para ello agregamos el siguiente código:



```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Http\Requests\StudentRequest;
6 use App\Models\Student;
7 use Illuminate\Http\Request;
8 use Symfony\Component\HttpFoundation\Response;
```

Y a continuación en los métodos que se desea usar la configuración, se realiza la siguiente actualización:



```
35 /**
36 * Store a newly created resource in storage.
37 *
38 * @param \Illuminate\Http\Request $request
39 * @return \Illuminate\Http\Response
40 */
41 public function store(StudentRequest $request)
42 {
43     //Create a new student object
44     $student = new Student();
45
46     //Set attributes values with the info received from the form
47     $student->code = $request->input('code');
48     $student->document = $request->input('document');
49     $student->name = $request->input('name');
50     $student->lastname = $request->input('lastname');
51     $student->birth_date = $request->input('birth_date');
52
53     //Save the record in the database
54     $student->save();
55     //use Symfony\Component\HttpFoundation\Response;
56     return redirect(to: '/students', status: Response::HTTP_CREATED);
57 }
```

```

  C StudentController.php x
184  /**
185   * Update the specified resource in storage.
186   *
187   * @param \Illuminate\Http\Request $request
188   * @param int $id
189   * @return \Illuminate\Http\Response
190   */
191   public function update(StudentRequest $request, $id)
192   {
193       //Create a new student object
194       $student = Student::find($id);
195
196       //Set attributes values with the info received from the form
197       $student->code = $request->input('code');
198       $student->document = $request->input('document');
199       $student->name = $request->input('name');
200       $student->lastname = $request->input('lastname');
201       $student->birth_date = $request->input('birth_date');
202
203       //Save the record in the database
204       $student->save();
205       //use Symfony\Component\HttpFoundation\Response;
206       return redirect(to: '/students');
207   }

```

Con esto obtenemos el mismo resultado obtenido en la sección anterior:

The screenshot shows a 'Create new Student' form with the following fields and their current values:

- Code:** (empty)
- Document:** (empty)
- Name:** (empty)
- Last Name:** (empty)
- Birth date:** 08/04/2022

A red callout box at the bottom left lists validation errors:

- The code field is required.
- The document field is required.
- The name field is required.
- The lastname field is required.
- The birth date field is required.

A blue 'Save' button is located at the bottom left of the form area.

Para agregar mensajes personalizados, es necesario crear un nuevo método como lo muestra la [documentación oficial](#):

```

  StudentRequest.php x
24     public function rules()
25     {
26         return [
27             'code' => 'required',
28             'document' => ['required', 'integer', 'min:5'],
29             'name' => 'required|min:5|max:30',
30             'lastname' => ['required'],
31             'birth_date' => ['required', 'date']
32         ];
33     }
34
35     public function messages()
36     {
37         return [
38             'code.required' => 'The code is required. Try using SIA code',
39             'document.integer' => 'The document should contains only numbers'
40         ];
41     }
42 }

```

Obteniendo el mismo resultado de mensajes personalizados de la sección anterior:

The screenshot shows a 'Create new Student' form with the following fields and errors:

- Code:** An error message is displayed below the input field: "The code is required. Try using SIA code".
- Document:** An error message is displayed below the input field: "The document should contains only numbers".
- Name:** No error message is displayed.
- Last Name:** No error message is displayed.
- Birth date:** The value "08/04/2022" is displayed in the input field.

A red box highlights the error messages for the 'Code' and 'Document' fields. A blue 'Save' button is located at the bottom left of the form.

9.9.1. No perder información al validar

Durante el proceso de validación que hemos llevado a cabo hasta el momento ocurre que si la petición no pasa el proceso de validación, toda la información de nuestros formularios se pierde y el usuario final debe volver a escribir la totalidad de la información. Este comportamiento se puede evitar gracias al helper old como se encuentra descrito en la [documentación oficial](#).

Para implementar el helper, en nuestro formulario de creación de recurso agregaremos las siguientes líneas:

```

1  @extends('layouts.base')
2  @section('title', 'Create New Student')
3  @section('container')
4      <main class="container">
5          <h1 class="alert alert-success text-center"> <i class="fa-solid fa-user-plus"></i> Create new Student</h1>
6          <form method="POST" action="{{route('students.store')}}">
7              @csrf
8              <div class="row">
9                  <div class="col-6">
10                     <div class="mb-3">
11                         <label for="code" class="form-label">Code</label>
12                         <input type="text" class="form-control" name="code" id="code" value="{{old('code')}}">
13                     </div>
14                     <div class="col-6">
15                         <div class="mb-3">
16                             <label for="document" class="form-label">Document</label>
17                             <input type="text" class="form-control" name="document" id="document" value="{{old('document')}}">
18                         </div>
19                     </div>
20                     <div class="col-6">
21                         <div class="mb-3">
22                             <label for="name" class="form-label">Name</label>
23                             <input type="text" class="form-control" name="name" id="name" value="{{old('name')}}">
24                         </div>
25                     </div>
26                     <div class="col-6">
27                         <div class="mb-3">
28                             <label for="lastname" class="form-label">Last Name</label>
29                             <input type="text" class="form-control" name="lastname" id="lastname" value="{{old('lastname')}}">
30                         </div>

```

En el formulario de edición del recurso tenemos el mismo caso por lo que también tendremos que agregar dichos valores, sin embargo como también tenemos la posibilidad de recibir una variable de tipo student, debemos mezclar las soluciones obteniendo la siguiente solución:

```

3  @section('container')
4      <main class="container">
5          <h1 class="alert alert-success text-center"> <i class="fa-solid fa-user-pen"></i> Edit Student</h1>
6          <form method="PUT" action="{{route('students.update', $student)}}>
7              @csrf
8              @method('PUT')
9              <div class="row">
10                 <div class="col-6">
11                     <div class="mb-3">
12                         <label for="code" class="form-label">Code</label>
13                         <input type="text" class="form-control" name="code" id="code" value="{{old('code', $student->code)}}>
14                     </div>
15                 </div>
16                 <div class="col-6">
17                     <div class="mb-3">
18                         <label for="document" class="form-label">Document</label>
19                         <input type="text" class="form-control" name="document" id="document" value="{{old('document', $student->document)}}>
20                     </div>
21                 </div>
22                 <div class="col-6">
23                     <div class="mb-3">
24                         <label for="name" class="form-label">Name</label>
25                         <input type="text" class="form-control" name="name" id="name" value="{{old('name', $student->name)}}>
26                     </div>
27                 </div>
28                 <div class="col-6">
29                     <div class="mb-3">
30                         <label for="lastname" class="form-label">Last Name</label>
31                         <input type="text" class="form-control" name="lastname" id="lastname" value="{{old('lastname', $student->lastname)}}>
32                     </div>
33                 </div>
34                 <div class="col-6">
35                     <div class="mb-3">
36                         <label for="birth_date" class="form-label">Birth date</label>
37                         <input type="date" class="form-control" name="birth_date" id="birth_date" value="{{old('birth_date', $student->birth_date)}}>
38                     </div>

```

10. Relaciones de bases de datos

Ahora que ya realizamos las operaciones básicas CRUD sobre un recurso de la base de datos, es necesario pensar en cómo se debe trabajar con otros recursos. Lo primero que haremos será mostrar la información detallada del recurso que ya teníamos creado, para ello implementamos su método show como se muestra a continuación:

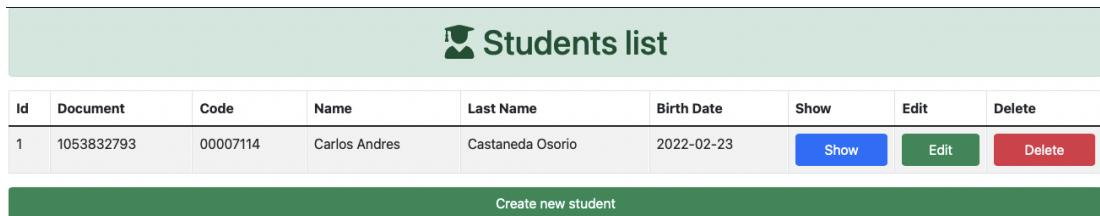
10.1. Ver el detalle de un recurso

Lo primero que haremos será crear un ancla en nuestra página index de los estudiantes que nos redireccione a la ruta que está definida para el método show, con un enlace como el que se muestra a continuación:



```
index.blade.php
14 <th>Birth Date</th>
15 <th>Show</th>
16 <th>Edit</th>
17 <th>Delete</th>
18 </tr>
19 </thead>
20 <tbody>
21 @foreach($students as $student)
22 <tr>
23 <td>{{$student->id}}</td>
24 <td>{{$student->document}}</td>
25 <td>{{$student->code}}</td>
26 <td>{{$student->name}}</td>
27 <td>{{$student->lastname}}</td>
28 <td>{{$student->birth_date}}</td>
29 <td><div class="d-grid"><a href="{{route('students.show', $student)}}" class="btn btn-primary">Show</a></div></td>
30 <td><div class="d-grid"><a href="{{route('students.edit', $student)}}" class="btn btn-success">Edit</a></div></td>
31 <td>
32 <form action="{{route('students.destroy', $student)}}" method="POST">
33 @method('DELETE')
34 <div class="d-grid">
35 <input
36 type="submit"
37 value="Delete"
38 class="btn btn-danger"
39 onclick="return confirm('¿Delete {{$student->name}}?')">
40 </div>
41 </form>
42 </td>
```

Que nos genera un resultado como el que se muestra a continuación:



Id	Document	Code	Name	Last Name	Birth Date	Show	Edit	Delete
1	1053832793	00007114	Carlos Andres	Castaneda Osorio	2022-02-23	Show	Edit	Delete

Create new student

Recordemos que el enlace al que atiende el método show lo podemos encontrar listando nuestras rutas como se muestra a continuación:

```

Laravel [example-app] 12:37:57
php artisan route:list

GET|HEAD / ..... IgnitionController@index
POST _ignition/execute-solution ..... IgnitionController@executeSolution
GET|HEAD _ignition/health-check ..... IgnitionController@healthCheck
POST _ignition/update-config ..... IgnitionController@updateConfig
GET|HEAD api/user ..... Laravel\Sanctum\CsrfCookieController@show
GET|HEAD sanctum/csrf-cookie ..... Laravel\Sanctum\CsrfCookieController@show
GET|HEAD students ..... StudentController@index
POST students ..... StudentController@create
GET|HEAD students/create ..... StudentController@create
GET|HEAD students/{student} ..... StudentController@show
PUT|PATCH students/{student} ..... StudentController@update
DELETE students/{student} ..... StudentController@destroy
GET|HEAD students/{student}/edit ..... StudentController@edit
GET|HEAD test ..... TestController@testDBConnection
GET|HEAD testDB ..... TestController@testDBConnection

```

Luego de enlazar a la ruta a través del ancla, será necesario implementar el método show, en el cual buscaremos el recurso en la base de datos y retornaremos una nueva vista con el recurso encontrado.

```

59     /**
60      * Display the specified resource.
61      *
62      * @param int $id
63      * @return \Illuminate\Http\Response
64      */
65     public function show(Student $student)
66     {
67         return view( view: 'students.show' . [
68             'student' => $student
69         ]);
70     }

```

Ahora, la vista *show* no existe, por lo que debemos crearla y colocarle contenido acorde al estudiante recibido, este proceso se muestra a continuación:

```

1 <extends='layouts.base'
2 @section('title', "Student | {$student->name} {$student->lastname}")
3 @section('content')
4     <main class="container">
5         <div class="row">
6             <div class="col-md-8">
7                 <div class="card">
8                     <div class="card-body">
9                         <h1 class="alert alert-success text-center"><i class="fa-solid fa-user-graduate"></i> {$student->name} {$student->lastname}</h1>
10                    <div class="row g-0">
11                        <div class="col-md-4">
12                            
13                        </div>
14                        <div class="col-md-8">
15                            <div class="card">
16                                <div class="card-title">{$student->name} {$student->lastname}</h5>
17                                <p class="card-text">Code: {$student->code}</p>
18                                <p class="card-text">Document: {$student->document}</p>
19                                <p class="card-text">Email: {$student->email}</p>
20                                <p class="card-text"><small class="text-muted">Birth date: {$student->birth_date}</small></p>
21                            </div>
22                        </div>
23                    </div>
24                </div>
25            </div>
26        </main>
27    @endsection

```

Obteniendo el siguiente resultado:

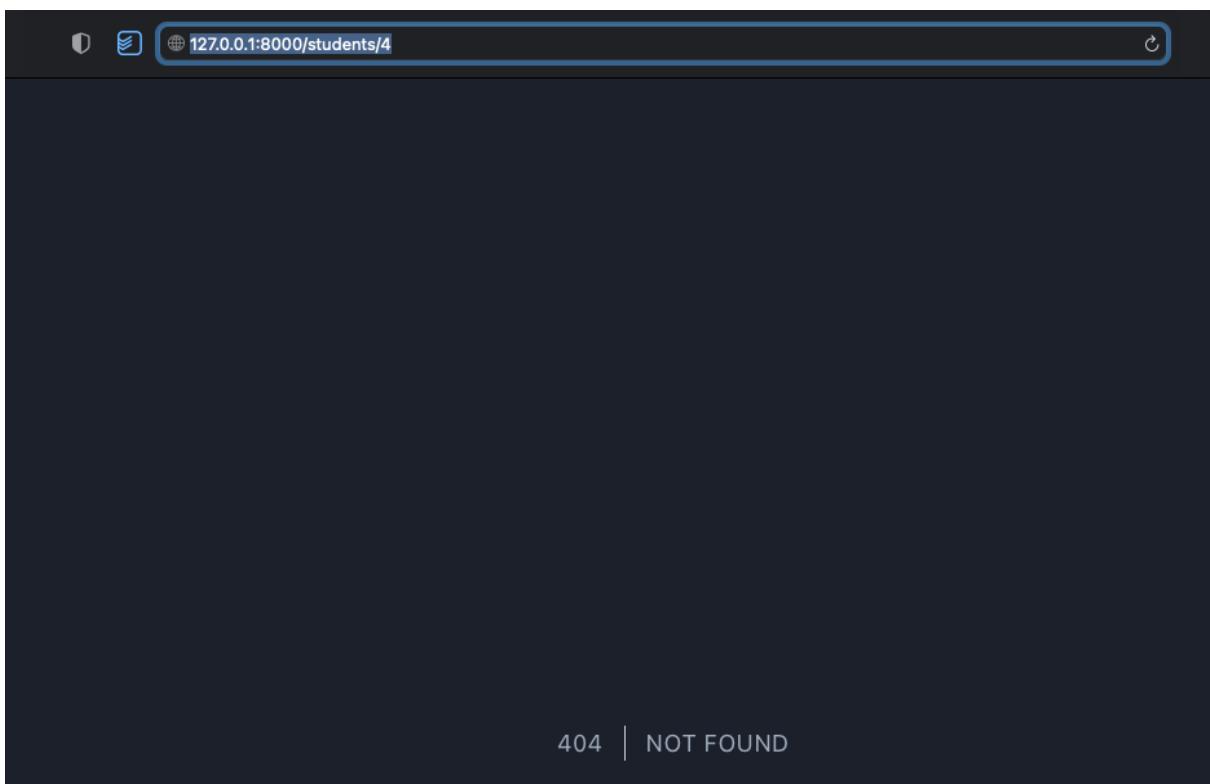
 Carlos Andres Castaneda Osorio



Carlos Andres Castaneda Osorio
Code: 00007114
Document: 1053832793
Email: carlos.castaneda@ucaldas.edu.co
Birth date: 2022-02-23

Recordemos que con el contenido recibido desde el backend, en nuestra vista de blade que actúa como el frontend de nuestra aplicación podemos mostrar información en diferentes presentaciones y usando diferentes componentes del framework frontend utilizado, en este caso [bootstrap](#).

Si intentamos acceder a un recurso inexistente intentando cambiar la dirección URL, nos retornará un error 404, esto gracias a que estamos usando el método `findOrFail`.



Dentro de esta sección se completó la tabla de estudiantes añadiendo el email del mismo, para esto se creó una nueva migración y se añadió la columna email.

Adicionalmente, para mostrar una imagen mientras se carga la foto real del estudiante se usó la herramienta externa Gravatar. Para más información visitar la [documentación oficial](#).

Finalmente, la herramienta Gravatar necesita un hash del correo electrónico, una opción sería crear una nueva columna y almacenar el email como hash al momento de crear un nuevo registro, sin embargo como Gravatar es un recurso momentáneo no vale la pena afectar nuestra base de datos y por ello se decide crear un atributo directamente en el modelo. Esta funcionalidad está incluida en Laravel y puede ser consultada en su [documentación oficial](#).

Lo que hacemos es crear la función `emailHashed` y crearla con la estructura requerida por la documentación oficial de Laravel.



```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Casters\Attribute;
6 use Illuminate\Database\Eloquent\Factories\HasFactory;
7 use Illuminate\Database\Eloquent\Model;
8 use Illuminate\Support\Facades\Log;
9
10 class Student extends Model
11 {
12     use HasFactory;
13
14     protected $appends = ['email_hashed'];
15
16     protected function emailHashed(): Attribute
17     {
18         return new Attribute(
19             get: fn () => md5( string: strtolower( string: trim( string: $this->email ) ) ),
20         );
21     }
22 }
```

Al agregar esta función al modelo, ya será posible acceder al atributo `email_hashed` como se muestra a continuación:

```

11 </div>
12 

```

Nota: Prestar especial atención a la convención de nombres de las funciones (*camelCase*) frente a la convención de nombres de los atributos (*snake_case*)

10.2. Model Binding

Una funcionalidad de Laravel en cuanto a trabajar con recursos es el *model binding*, este nos permite recibir no sólo el ID del recurso y luego buscarlo, si no que el automáticamente realiza la búsqueda del ID recibido. Para implementar el *Model Binding* en nuestro método show debemos realizar los siguientes cambios.

```

GET|HEAD /ignition/execute-solution Ignition\ExecuteSolution@index
POST /ignition/health-check Ignition\HealthCheck@spatieLaravel
GET|HEAD /ignition/update-config Ignition\UpdateConfig@spatieLaravel
GET|HEAD /api/user ... Laravel\Sanctum\CsrfCookieController@...
GET|HEAD /sanctum/csrf-cookie Laravel\Sanctum\CsrfCookieController@...
GET|HEAD /students ... Students\Index@StudentController@index
POST /students ... Students\Store@StudentController@store
GET|HEAD /students/create ... Students\Create@StudentController@create
GET|HEAD /students/{student} ... Students\Show@StudentController@show
PUT|PATCH /students/{student} ... Students\Update@StudentController@update
DELETE /students/{student} ... Students\Destroy@StudentController@des...
GET|HEAD /students/{student}/edit ... Students>Edit@StudentController@...
GET|HEAD /test ... TestController@testDBConnection

```

```

62 * @param int $id
63 * @return \Illuminate\Http\Response
64 */
65 public function show(Student $student)
66 {
67     return view('students.show', [
68         'student' => $student
69     ]);
70 }
71

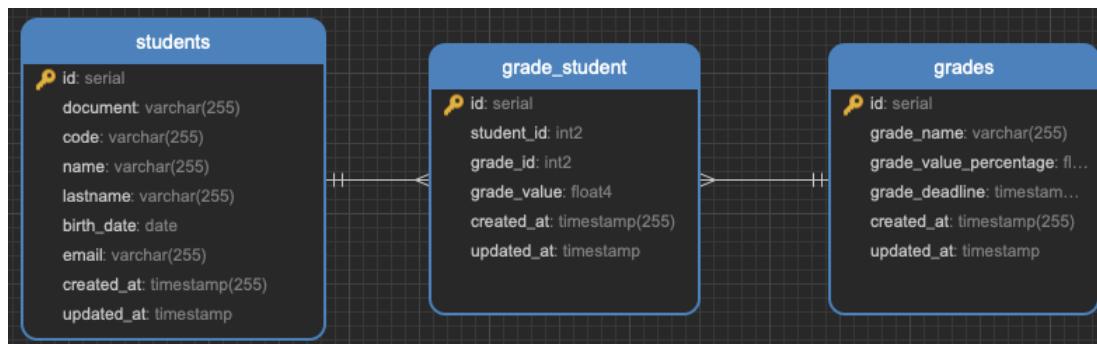
```

Importante: El nombre del parámetro usado en *model binding* debe coincidir con el establecido en la ruta.

10.3. Relaciones con Eloquent

Las relaciones en bases de datos son muy comunes cuando nos encontramos desarrollando un sistema de información, en esto encontramos los modelos entidad relación en el que se detalla la cardinalidad entre tablas.

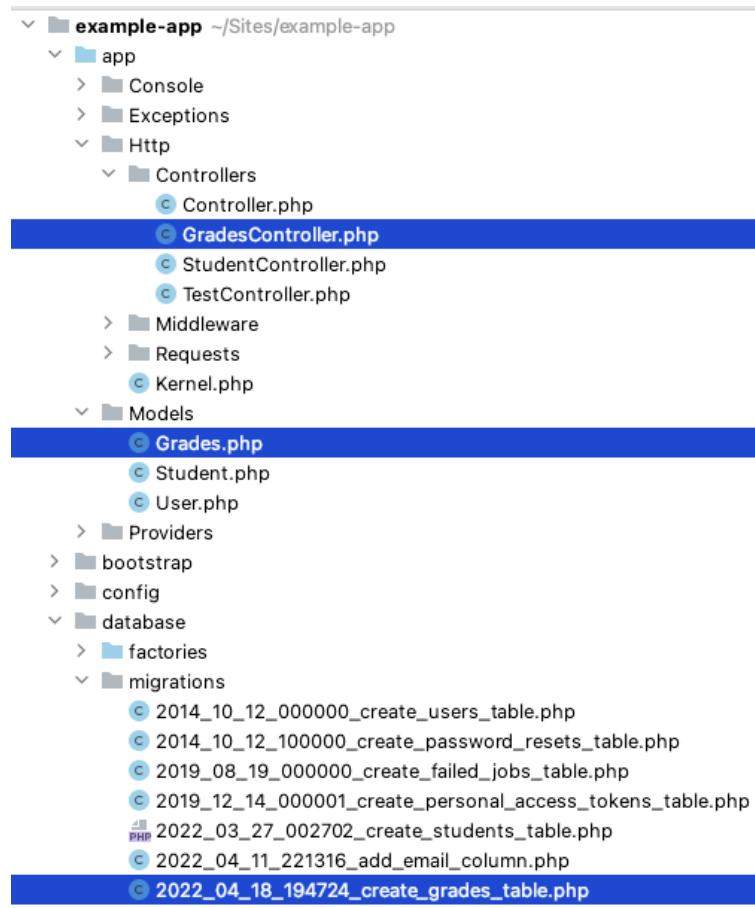
Recordemos que para el desarrollo de esta guía nos encontramos trabajando con la siguiente relación entre tablas:



Como tenemos un nuevo recurso llamado “*Grades*”, es necesario crear un nuevo modelo con su respectiva migración y controlador. Las tres acciones se pueden llevar a cabo ejecutando sólo un comando y habilitando las banderas correctas como se muestra a continuación.

```
apple ~ /Sites/students
└─ php artisan make:model -mr Grade
Model created successfully.
Created Migration: 2022_05_21_011854_create_grades_table
Controller created successfully.
```

Obteniendo el siguiente resultado:



Para estar seguros de la funcionalidad de las banderas anteriormente utilizadas, se puede abrir el comando de ayuda de `make:model` como se muestra a continuación:

```
php artisan make:model --help
Description:
Create a new Eloquent model class

Usage:
make:model [options] [--] <name>

Arguments:
name           The name of the class

Options:
-a, --all      Generate a migration, seeder, factory, policy, and resource controller for the model
-c, --controller Create a new controller for the model
-f, --factory   Create a new factory for the model
--force        Create the class even if the model already exists
-m, --migration Create a new migration file for the model
--morph-pivot  Indicates if the generated model should be a custom polymorphic intermediate table model
--policy       Create a new policy for the model
-s, --seed      Create a new seeder for the model
-p, --pivot     Indicates if the generated model should be a custom intermediate table model
-r, --resource  Indicates if the generated controller should be a resource controller
--api          Indicates if the generated controller should be an API controller
-R, --requests Create new form request classes and use them in the resource controller
--test         Generate an accompanying PHPUnit test for the Model
--pest         Generate an accompanying Pest test for the Model
-h, --help      Display help for the given command. When no command is given display help for the list command
-q, --quiet    Do not output any message
-V, --version   Display this application version
--ansi|-no-ansi Force (or disable --no-ansi) ANSI output
-n, --no-interaction Do not ask any interactive question
--env[=ENV]     The environment the command should run under
-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
```

Como en el modelo se aprecia una tabla intermedia, también es necesario crear su migración. Las tablas intermedias resultado de relaciones N a N entre tablas se les conoce en Laravel como tablas “Pivot”. Para crear esta tabla pivot con su modelo ejecutaremos el siguiente comando:

```
php artisan make:model -pm GradeStudent
Model created successfully.
Created Migration: 2022_04_18_195247_create_grade_student_table
```

Al abrir el modelo, es posible apreciar que la clase de la cual hereda es diferente a los modelos de las tablas principales:

```
GradeStudent.php x
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Relations\Pivot;
6
7 class GradeStudent extends Pivot
8 {
9     //
10 }
```

Luego de haber creado los recursos, es momento de relacionar los modelos. En este caso nuestra relación es de muchos a muchos, lo cual es importante para el código. Para ver la manera de relacionar recursos se debe visitar la [documentación oficial](#) de Laravel que habla sobre relaciones.

Lo primero que haremos será modificar las migraciones agregando las nuevas columnas y las llaves foráneas que conectan entre las tablas, para ello establecemos los método up de las migraciones como se muestra a continuación:

```
public function up()
{
    Schema::create('grades', function (Blueprint $table) {
        $table->id();
        $table->string('grade_name');
        $table->float('grade_value_percentage');
        $table->timestamp('grade_deadline');
        $table->timestamps();
    });
}
```

```

public function up()
{
    Schema::create('grade_student', function (Blueprint $table) {
        $table->id();
        $table->integer('student_id');
        $table->foreignId('grade_id')->references('id')->on('grades');
        // $table->foreignId('grade_id')->constrained();
        $table->float('grade_value');
        $table->timestamps();

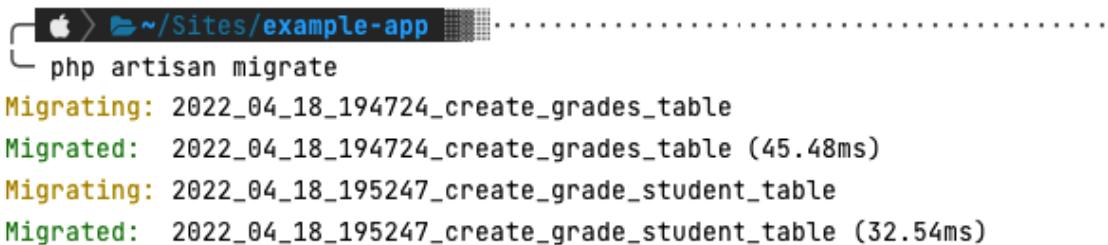
        $table->foreign('student_id')->references('id')->on('students');
    });
}

```

Es importante que el nombre del campo de la llave foránea se ajuste a la convención de nombres de Laravel, el caso contrario al establecer la relación será necesario especificar el nombre del campo que contiene la llave foránea. Para más información visitar la [documentación oficial](#).

Para añadir llaves foráneas en las migraciones tenemos diferentes sintaxis, para ver a mayor detalle las opciones disponibles visitar la [documentación oficial](#).

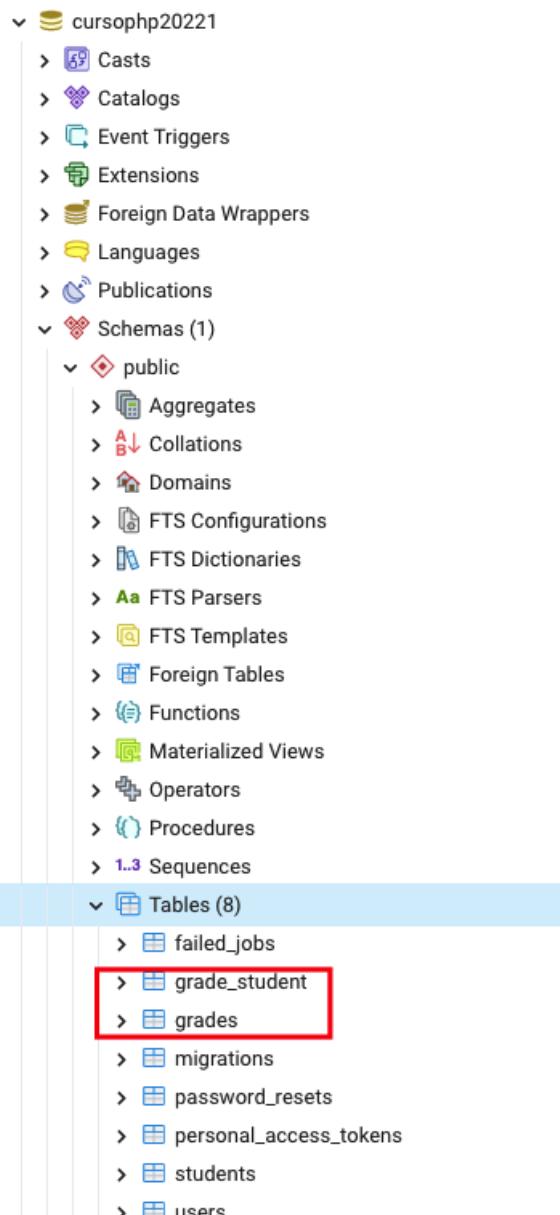
Ejecutamos la migración para que la tabla sea creada y verificamos en la base de datos que efectivamente se haya creado satisfactoriamente.



```

❯ php artisan migrate
Migrating: 2022_04_18_194724_create_grades_table
Migrated: 2022_04_18_194724_create_grades_table (45.48ms)
Migrating: 2022_04_18_195247_create_grade_student_table
Migrated: 2022_04_18_195247_create_grade_student_table (32.54ms)

```



Como ya se tienen las tablas creadas, es momento de relacionar los modelos a través de código, para ello escribimos el siguiente código en nuestros modelos:

The screenshot shows two code editors side-by-side. The top editor is for `Student.php` and the bottom one is for `Grade.php`. Both files are part of the `App\Models` namespace and extend the `Model` class.

```

Student.php
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Attribute;
6 use Illuminate\Database\Eloquent\Factories\HasFactory;
7 use Illuminate\Database\Eloquent\Model;
8 use Illuminate\Support\Facades\Log;
9
10 class Student extends Model
11 {
12     use HasFactory;
13
14     protected $appends = ['email_hashed'];
15
16     protected function emailHashed(): Attribute
17     {
18         return new Attribute(
19             get: fn () => md5( strtolower(trim($this->email)) ),
20         );
21     }
22
23     public function grades()
24     {
25         return $this->belongsToMany(Grade::class)->withPivot(['id', 'grade_value']);
26     }
27 }

```

```

Grade.php
1 <?php
2
3 namespace App\Models;
4
5 use ...
6
7 class Grade extends Model
8 {
9     use HasFactory;
10
11     /**
12      * The attributes that are mass assignable.
13      *
14      * @var array
15      */
16     protected $fillable = ['grade_name', 'grade_value_percentage', 'grade_deadline'];
17
18     public function students()
19     {
20         return $this->belongsToMany(Student::class)->withPivot(['id', 'grade_value']);
21     }
22 }

```

Es importante tener en cuenta que la tabla intermedia, llamada tabla *pivot* en el mundo de Laravel tiene un campo adicional a las llaves foráneas, y en la relación creada en el código [se debe definir qué se van a incluir estos campos](#).

Para comprobar que la relación quedó correctamente configurada, abrimos tinker, y ejecutamos los siguientes comandos:

```

php artisan tinker
Psy Shell v0.11.2 (PHP 8.1.4 - cli) by Justin Hileman
>>> $student = \App\Models\Student::find(1);
=> App\Models\Student {#4431
    id: 1,
    document: "1053832793",
    code: "00007114",
    name: "Carlos Andres",
    lastname: "Castaneda Osorio",
    birth_date: "2022-02-23",
    created_at: "2022-03-29 20:00:20",
    updated_at: "2022-04-11 22:18:21",
    email: "carlos.castaneda@ucaldas.edu.co",
    +email_hashed: "d0ca4a2f2c3343925d89abc0f7d2f701",
}
>>> return $student->grades;
=> Illuminate\Database\Eloquent\Collection {#4435
    all: [],
}

```

A pesar de que el arreglo se encuentre vacío podemos concluir que nuestra relación quedó correctamente configurada ya que hasta el momento no se han creado notas asociadas a dicho estudiante.

Nota: Es importante resaltar que *grades* se está llamando como un atributo y no como un método.

Sin embargo, para ver más claro el funcionamiento, insertamos a través de la base de datos una nota y a través de la tabla intermedia la asociamos al primer estudiante.

id [PK] bigint	grade_name character varying (255)	grade_value_percentage double precision	grade_deadline timestamp without time zone	created_at timestamp without time zone	updated_at timestamp without time zone
1	Taller PHP		10	2022-03-18 23:59:00	2022-04-20 11:36:51

id [PK] bigint	student_id integer	grade_id bigint	grade_value double precision	created_at timestamp without time zone	updated_at timestamp without time zone
1	1	1	5	2022-04-20 11:38:59	[null]

Obteniendo ahora la siguiente salida desde tinker:

```

>>> return $student->grades;
=> Illuminate\Database\Eloquent\Collection {#4436
    all: [
        App\Models\Grade {#4437
            id: 1,
            grade_name: "Taller PHP",
            grade_value_percentage: "10",
            grade_deadline: "2022-03-18 23:59:00",
            created_at: "2022-04-20 11:36:51",
            updated_at: "2022-04-20 11:36:51",
            pivot: Illuminate\Database\Eloquent\Relations\Pivot {#4438
                student_id: 1,
                grade_id: 1,
                grade_value: "5",
            },
        },
    ],
}

```

Para ver los métodos disponibles para obtener colecciones se puede visitar la [documentación oficial](#) de laravel.

Conociendo que la relación se encuentra activa, a continuación es necesario que en nuestra vista show de nuestro estudiante se muestren todas las notas relacionadas. Para ello modificamos la vista como se muestra a continuación:

```

<div class="col-md-8">
    <div class="card-body">
        <h5 class="card-title">{{ $student->name }} {{ $student->lastname }}</h5>
        <p class="card-text">Code: {{ $student->code }}</p>
        <p class="card-text">Document: {{ $student->document }}</p>
        <p class="card-text">Email: {{ $student->email }}</p>
        <p class="card-text"><small class="text-muted">Birth date: {{ $student->birth_date }}</small></p>
        <h2 class="alert alert-success text-center"> <i class="fa-solid fa-file-pen"></i> Grades</h2>
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th>Grade Name</th>
                    <th>Grade deadline</th>
                    <th>Grade value percentage</th>
                    <th>Grade Value</th>
                    <th>Update Grade</th>
                </tr>
            </thead>
            <tbody>
                @foreach($student->grades as $grade)
                    <tr>
                        <td>{{$grade->grade_name}}</td>
                        <td>{{$grade->grade_deadline}}</td>
                        <td>{{$grade->grade_value_percentage}}%</td>
                        <td>
                            <form action="/students/{{ $student->id }}/grades/{{ $grade->pivot->id }}" method="POST">
                                <input disabled hidden type="number" value="" name="id">
                                <td><input type="number" value="{{ $grade->pivot->grade_value }}" name="grade_value" class="form-control"></td>
                                <td><button type="submit" class="btn btn-success"><i class="fa-solid fa-file-pen"></i> Update</button></td>
                            </form>
                        </td>
                    </tr>
                @endforeach
            </tbody>
        </table>
    </div>

```

Obteniendo el siguiente resultado:

Carlos Andres Castaneda Osorio



Carlos Andres Castaneda Osorio
Code: 00007114
Document: 1053832793
Email: carlos.castaneda@ucaldas.edu.co
Birth date: 2022-02-23

Grades

Grade Name	Grade deadline	Grade value percentage	Grade Value	Update Grade
Taller PHP	2022-03-18 23:59:00	10%	5	 Update

A continuación lo que debemos hacer es el CRUD del recurso *grades*, siguiendo el mismo proceso contemplado en esta guía en la sección 9. A continuación se presentan las interfaces resultado:

Grades list

ID	Name	Value Percentage	Deadline	Edit	Delete
1	Taller PHP	10%	2022-03-18 23:59:00	 Edit	 Delete
4	Taller PHP - Composer	10%	2022-04-22 12:27:00	 Edit	 Delete

 Create new grade

Create new Grade

Name: Value Percentage: %

Deadline:

 Save

Edit Grade

Name: Value Percentage: %

Deadline:

 Update

Grades list					
ID	Name	Value Percentage	Deadline	Edit	Delete
1	Taller PHP	10%	2022-03-18 23:59:00	<button>Edit</button>	<button>Delete</button>
4	Taller PHP - Composer	10%	2022-04-22 12:27:00	<button>Edit</button>	<button>Delete</button>
Create new grade					

¿Delete Taller PHP?

[Cancelar](#) [OK](#)

A continuación, es necesario que a cada estudiante se le asignen las notas que le corresponden, para ello en la interfaz show del estudiante crearemos una tabla con todas las notas y al tiempo le asignaremos los valores a aquellas notas que ya tienen un valor, para ello llevamos a cabo los siguientes cambios:

En la tabla, mostrar todas las notas disponibles en el sistema y que cada una de ellas sea un formulario que permita crear la nota para el estudiante cuando no exista o actualizar el valor existente, para ello en la interfaz de show hacemos el siguiente cambio:

```

@foreach($student->grades as $grade)
    <tr>
        <td>{{$grade->grade_name}}</td>
        <td>{{$grade->grade_deadline}}</td>
        <td>{{$grade->grade_value_percentage}}%</td>
        <form action="{{isset($grade->pivot) ? "/students/{$student->id}/grade-student/{$grade->pivot->id}" : "/students/{$student->id}/grade-student"}}"
              method="POST">
            @csrf
            @isset($grade->pivot)
                @method('PUT')
            @endisset
            <input hidden type="number" value="{{$grade->id}}" name="grade_id">
            <td>
                <input
                    type="number"
                    value="{{isset($grade->pivot) ? $grade->pivot->grade_value : '0'}}"
                    name="grade_value"
                    class="form-control"
                    step="0.1"
                >
            </td>
            <td>
                <button type="submit" class="btn btn-success">
                    <i class="fa-solid fa-file-pen"></i> Update
                </button>
            </td>
        </form>
    </tr>
@endforeach

```

A continuación se deben crear las rutas de actualización y creación de notas de estudiantes, para ello en el archivo de rutas agregamos los siguientes registros:



```
PHP web.php x
30
31     //Asociar una nota a un estudiante
32     Route::post(
33         uri: '/students/{student}/grade-student',
34         action: [\App\Http\Controllers\StudentController::class, 'associateStudentGrade']
35     );
36
37     //Actualizar una nota existente de un estudiante
38     Route::put(
39         uri: '/students/{student}/grade-student/{grade_student}',
40         action: [\App\Http\Controllers\StudentController::class, 'updateStudentGrade']
41     );

```

Al momento de retornar las notas a la vista *show*, es necesario mostrar todas las notas, no sólo aquellas que están relacionadas con el estudiante, por ello hacemos uso de la [función merge](#) que nos permite combinar las notas completas con las del estudiante. El método se ve como se muestra a continuación:

```
/*
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show(Student $student)
{
    $grades = Grade::all()->merge(items: $student->grades);
    // $grades->dd();
    $student->grades = $grades;
    return view( view: 'students.show' , data: [
        'student' => $student
    ]);
}
```

Por último se realiza la implementación de los métodos para asociar y actualizar una nota a un estudiante como se muestra a continuación:

```
example-app app / Http / Controllers / StudentController.php / StudentController > m s show
Project StudentController.php
130
131     function associateStudentGrade(Request $request, Student $student) {
132         //dd($request->all());
133         $student->grades()->attach( id: $request->input( key: 'grade_id'), attributes: [
134             'grade_value' => $request->input( key: 'grade_value')
135         ]);
136
137         return back();
138     }
139
140     function updateStudentGrade(Request $request, Student $student, GradeStudent $gradeStudent) {
141         $gradeStudent->update( attributes: [
142             'grade_value' => $request->input( key: 'grade_value')
143         ]);
144
145         return back();
146     }
147 }
```

Con esto ya se tiene el CRUD completo de los estudiantes y las notas, además se puede asociar el valor de cada nota a los estudiantes.

11. Envío de correo electrónico

Es normal dentro de los sistemas de información que sea necesario notificar por correo electrónico algunas acciones realizadas dentro del sistema. Laravel conoce que este requerimiento es solicitado por muchas personas, por lo que trae implementado un módulo de envío de correo electrónico en el que sólo tendremos que configurar los valores de las variables de entorno de acuerdo a nuestro proveedor de envíos de correos electrónicos. Para más información visitar la [documentación oficial](#).

En nuestro caso para el envío del correo electrónico, al actualizar el valor de la nota para el estudiante, se enviará un email notificando sobre el cambio de la misma.

Posterior a la definición del lugar donde enviaremos correo electrónico en nuestro aplicativo, es momento de usar un servicio SMTP que facilite el envío de correos. Por comodidad se recomienda no usar servidores de correos reales ya que al enviar correos de manera masiva pueden bloquear nuestras cuentas. para probar que efectivamente nuestros correos se están despachando usamos el servicio de [Mailtrap.io](#). Se deben registrar y obtener datos de SMTP que se ven como se muestra a continuación:

Create Free Forever Account

[Use Google account](#)

[Use Github account](#)

OR

[Signup with Your Email](#)

By creating a Mailtrap account, you agree to our [Terms of Service](#) and [Privacy Policy](#). We'll occasionally send you account related emails.

Shared Inboxes Billing

Start with...

Reset Password Notification To: <ccasta23@outlook.com> 4 months ago

Summary Report To: <ccasta23@outlook.com> 4 months ago

Summary Report To: <ccasta23@outlook.com> 4 months ago

SMTP Settings Email Address Auto Forward Manual Forward Team Members

Credentials [Reset SMTP/POP3](#)

SMTP

Host: smtp.mailtrap.io
Port: 25 or 465 or 587 or 2525
Username: 306bceaa442cba
Password: e18b1eab6f0cac
Auth: PLAIN, LOGIN and CRAM-MDS
TLS: Optional (STARTTLS on all ports)

POP3

Host: pop3.mailtrap.io
Port: 1100 or 9950
Username: 306bceaa442cba
Password: e18b1eab6f0cac
Auth: USER/PASS, PLAIN, LOGIN, APOP and CRAM-MDS
TLS: Optional (STARTTLS on all ports)

Integrations

Ruby on Rails

```
In config/environments.rb specify ActionMailer defaults for your development or staging servers:
config.action_mailer.delivery_method = :smtp
config.action_mailer.smtp_settings = {
  :user_name => '306bceaa442cba',
  :password => 'e18b1eab6f0cac',
  :address => 'smtp.mailtrap.io',
  :domain => 'smtp.mailtrap.io',
  :port => '2525',
  :authentication => :cram_md5
}
```

Tomamos estos valores y los llevamos a nuestro archivo de configuración de entorno .env

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=5d8140f6da1340
MAIL_PASSWORD=497907cb6dfeb0
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS="carlos.castaneda@ucaldas.edu.co"
MAIL_FROM_NAME="${APP_NAME}"
```

El siguiente paso, luego de tener configurada nuestra cuenta, es crear una plantilla de correo electrónico. Laravel nos ayuda con este proceso desde su gestor de línea de comandos artisan como se muestra a continuación:

```

php artisan make:mail --help
Description:
Create a new email class

Usage:
make:mail [options] [--] <name>

Arguments:
name          The name of the class

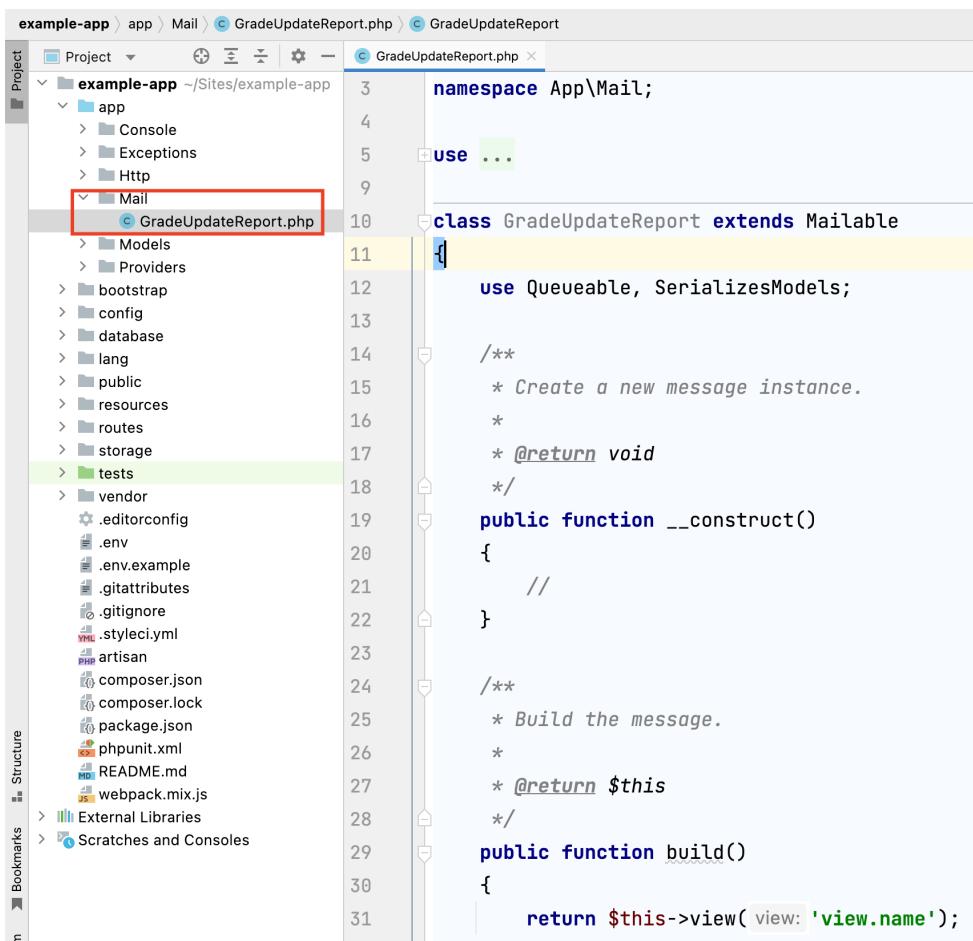
Options:
-f, --force      Create the class even if the mailable already exists
-m, --markdown[=MARKDOWN] Create a new Markdown template for the mailable [default: false]
--test          Generate an accompanying PHPUnit test for the Mail
--pest          Generate an accompanying Pest test for the Mail
-h, --help       Display help for the given command. When no command is given display help for the list command
-q, --quiet      Do not output any message
-V, --version    Display this application version
--ansi|--no-ansi Force (or disable --no-ansi) ANSI output
-n, --no-interaction Do not ask any interactive question
--env[=ENV]      The environment the command should run under

```

```

php artisan make:mail GradeUpdateReport
Mail created successfully.

```



```

namespace App\Mail;

use ...

class GradeUpdateReport extends Mailable
{
    use Queueable, SerializesModels;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this->view('view.name');
    }
}

```

The screenshot shows a code editor interface with a project structure on the left. The project is named 'example-app' and contains an 'app' folder which further contains 'Console', 'Exceptions', 'Http', and 'Mail'. The 'Mail' folder is expanded, and a file named 'GradeUpdateReport.php' is selected and highlighted with a red box. The code editor shows the PHP code for the 'GradeUpdateReport' class, which extends the 'Mailable' class and implements 'Queueable' and 'SerializesModels'. It includes a constructor and a build method that returns a view named 'view.name'.

A continuación modificamos el archivo creado para que reciba un objetivo de tipo estudiante y poder obtener todos los datos que enviaremos en nuestro correo electrónico.

```
c GradeUpdateReport.php x
3 namespace App\Mail;
4
5 use App\Models\Student;
6 use Illuminate\Bus\Queueable;
7 use Illuminate\Contracts\Queue\ShouldQueue;
8 use Illuminate\Mail\Mailable;
9 use Illuminate\Queue\SerializesModels;
10
11 class GradeUpdateReport extends Mailable
12 {
13     use Queueable, SerializesModels;
14
15     private $student;
16
17     /**
18      * Create a new message instance.
19      *
20      * @return void
21      */
22     public function __construct(Student $student)
23     {
24         $this->student = $student;
25     }
26
27     /**
28      * Build the message.
29      *
30      * @return $this
31      */
32     public function build()
33     {
34         return $this->view( view: 'mail.gradeUpdateReport', data: [
35             'student' => $this->student
36         ]);
37     }
}
```

Ahora creamos la plantilla de email basados en la plantilla del método show usado para cada estudiante eliminando los botones que realizan acciones como se muestra a continuación:

```

<div class="row">
    <div class="col-md-4">
        name }} {{ $student->lastname }}"
    </div>
    <div class="col-md-8">
        <div class="card-body">
            <h5 class="card-title">{{ $student->name }} {{ $student->lastname }}</h5>
            <p class="card-text">Code: {{ $student->code }}</p>
            <p class="card-text">Document: {{ $student->document }}</p>
            <p class="card-text">Email: {{ $student->email }}</p>
            <p class="card-text"><small class="text-muted">Birth date: {{ $student->birth_date }}</small></p>
            <h2 class="alert alert-success text-center"><i class="fa-solid fa-file-pen"></i> Grades</h2>
            <table class="table table-bordered table-striped">
                <thead>
                    <tr>
                        <th>Grade Name</th>
                        <th>Grade deadline</th>
                        <th>Grade value percentage</th>
                        <th>Grade Value</th>
                        <th>Update Grade</th>
                    </tr>
                </thead>
                <tbody>
                    @foreach($student->grades as $grade)
                    <tr>
                        <td>{{ $grade->grade_name }}</td>
                        <td>{{ $grade->grade_deadline }}</td>
                        <td>{{ $grade->grade_value_percentage }}%</td>
                        <td>{{ $grade->pivot ? $grade->pivot->grade_value : 'Sin calificar' }}</td>
                    </tr>
                    @endforeach
                </tbody>
            </table>
        </div>
    </div>
</div>

```

Ahora que todo se encuentra preparado, es momento de modificar el método *updateStudentGrade* creado en el controlador *StudentController*. En este método se realizará el envío de un correo electrónico una vez que la nota fue actualizada en base de datos. El primer paso será importar las clases para hacer envío de un email como lo muestra laravel en su [documentación oficial](#):

```

namespace App\Http\Controllers;

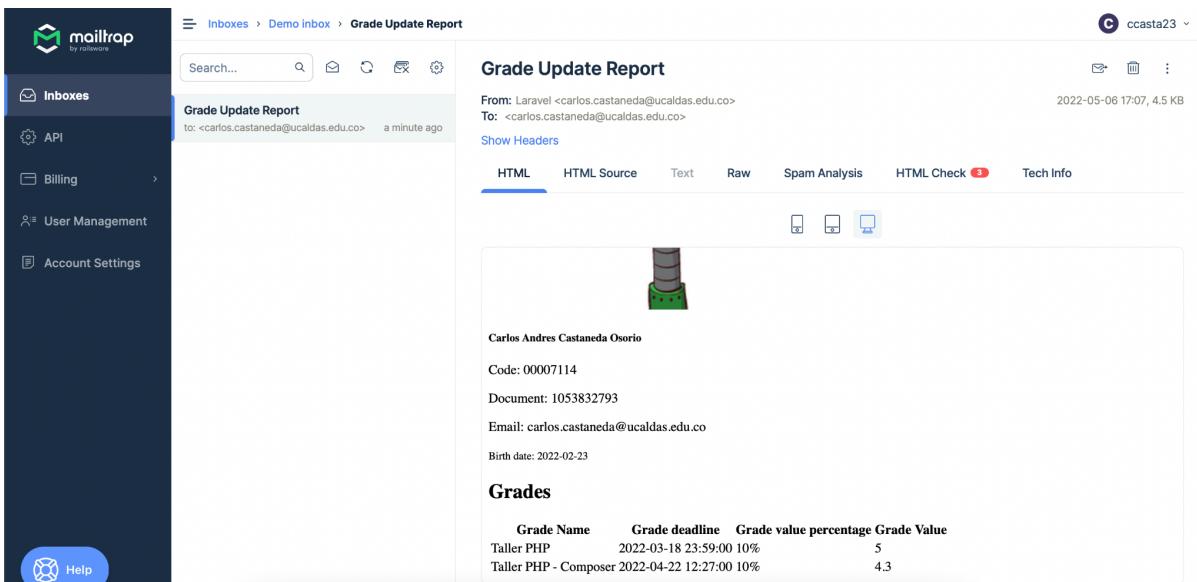
use App\Http\Requests\StudentRequest;
use App\Models\Grade;
use App\Models\GradeStudent;
use App\Models\Student;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;
use Illuminate\Support\Facades\Mail;
use App\Mail\GradeUpdateReport;

```

Ahora es momento de implementar la nueva funcionalidad en el método como se muestra a continuación:

```
function updateStudentGrade(Request $request, Student $student, GradeStudent $gradeStudent) {  
    $gradeStudent->update( attributes: [  
        'grade_value' => $request->input( key: 'grade_value')  
    ]);  
  
    Mail::to( users: $student->email)  
        ->send( mailable: new GradeUpdateReport( student: $student));  
  
    return back();  
}
```

Luego de probar nuestra funcionalidad en mailtrap.io deberemos poder ver el correo enviado como se muestra a continuación:



The screenshot shows the Mailtrap.io interface. On the left, there's a sidebar with options like 'Inboxes', 'API', 'Billing', 'User Management', and 'Account Settings'. The main area shows an inbox entry for a 'Grade Update Report' from 'Laravel <carlos.castaneda@ucaldas.edu.co>' to 'carlos.castaneda@ucaldas.edu.co' a minute ago. The email subject is 'Grade Update Report'. The message content includes a student profile with a photo, name (Carlos Andres Castaneda Osorio), code (00007114), document number (1053832793), email (carlos.castaneda@ucaldas.edu.co), and birth date (2022-02-23). Below this is a 'Grades' section with a table:

Grade Name	Grade deadline	Grade value percentage	Grade Value
Taller PHP	2022-03-18 23:59:00	10%	5
Taller PHP - Composer	2022-04-22 12:27:00	10%	4.3

Nota: Para enviar correos electrónicos reales sólo se debe configurar las credenciales correctas de SMTP, se puede buscar en proveedores como gmail, outlook, zoho mail entre otros para conocer nuestras credenciales de envío de correos.

12. Autenticación

Normalmente los sistemas de información tienen autenticación de usuarios, Laravel conoce de esta necesidad y por ello ya trae implementado muchos métodos que podemos simplemente instalar y utilizar.

En la [documentación oficial](#) los creadores de Laravel dejan abiertas las opciones disponibles y permiten que cada usuario de Laravel defina qué modelo es más conveniente para su aplicación. En nuestro caso usaremos el sistema de autenticación de [Laravel Jetstream](#) en su versión de [Livewire + Blade](#).

El primer paso es proceder con la instalación de los componentes requeridos y para ello seguimos la [guía de instalación de Laravel Jetstream](#). En esta guía se indica que lo primero que se debe realizar es la instalación del paquete, para lo cual ejecutaremos el siguiente comando:



```
composer require laravel/jetstream
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^2.7 for laravel/jetstream
./composer.json has been updated
Running composer update laravel/jetstream
Loading composer repositories with package information
Updating dependencies
Lock file operations: 9 installs, 0 updates, 0 removals
- Locking bacon/bacon-qr-code (2.0.7)
- Locking dasprid/enum (1.0.3)
- Locking jaybizzle/crawler-detect (v1.2.111)
```

Al realizar esta instalación, ahora tendremos disponible en nuestra utilidad de artisan un nuevo comando que se puede ver en la lista de las opciones como se muestra a continuación :

```
jetstream
jetstream:install     Install the Jetstream components and resources
```

El paso siguiente será ejecutar el comando y seguir las instrucciones de instalación como se muestra a continuación:

```
Apple ~ ~/Sites/example-app
└ php artisan jetstream:install livewire
  Migration created successfully.
  ./composer.json has been updated
  Running composer update livewire/livewire
  Loading composer repositories with package information
  Info from https://repo.packagist.org: #StandWithUkraine
```

Nota: En este caso se usó Livewire + Blade para continuar con la línea usada durante todo el curso, sin embargo como trabajo adicional a esta guía se propone revisar las diferencias de instalación con Inertia + Vue.js

Luego de finalizar la instalación es necesario instalar las librerías de Javascript requeridas por la interfaz. Para ello ejecutamos el siguiente comando:

```
Apple ~ ~/Sites/example-app
└ npm install && npm run dev
(#####) :: idealTree:serve-index: timing idealTree:node_modules/serve-index Completed in 4ms
```

Nota: Es imprescindible tener instalado **npm** en el computador para ejecutar este paso, para ello se pueden referir a la [documentación oficial](#) de node.

Adicionalmente, la librería requiere la ejecución de dos migraciones que trae por defecto, para ello ejecutamos las migraciones como se muestra a continuación:

```
Apple ~ ~/Sites/example-app
└ php artisan migrate
Migrating: 2014_10_12_200000_add_two_factor_columns_to_users_table
Migrated: 2014_10_12_200000_add_two_factor_columns_to_users_table (34.54ms)
Migrating: 2022_05_06_172342_create_sessions_table
Migrated: 2022_05_06_172342_create_sessions_table (40.07ms)
```

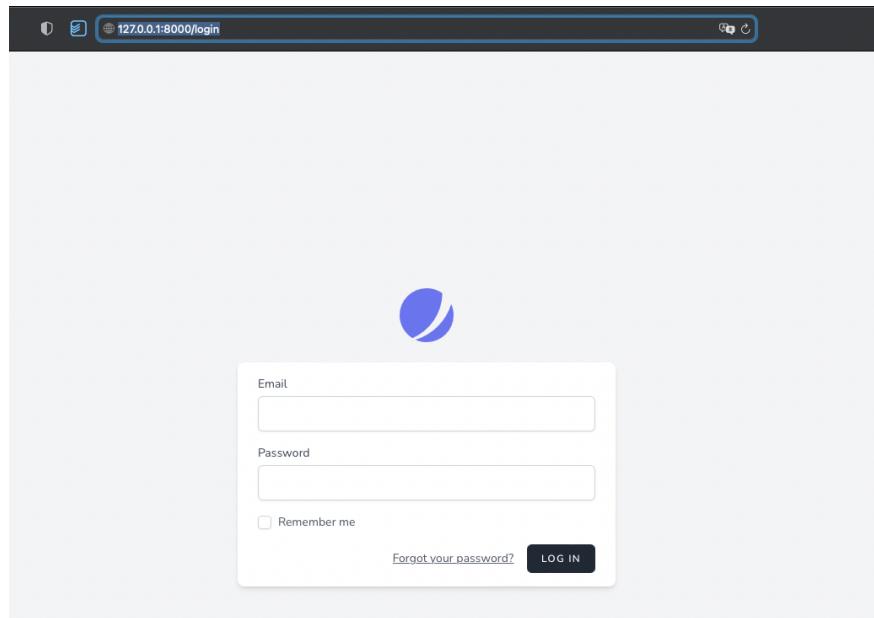
Para validar la instalación de las librerías debemos revisar qué rutas fueron creadas, se procede a ver el resultado del listado de rutas como se muestra a continuación:

```

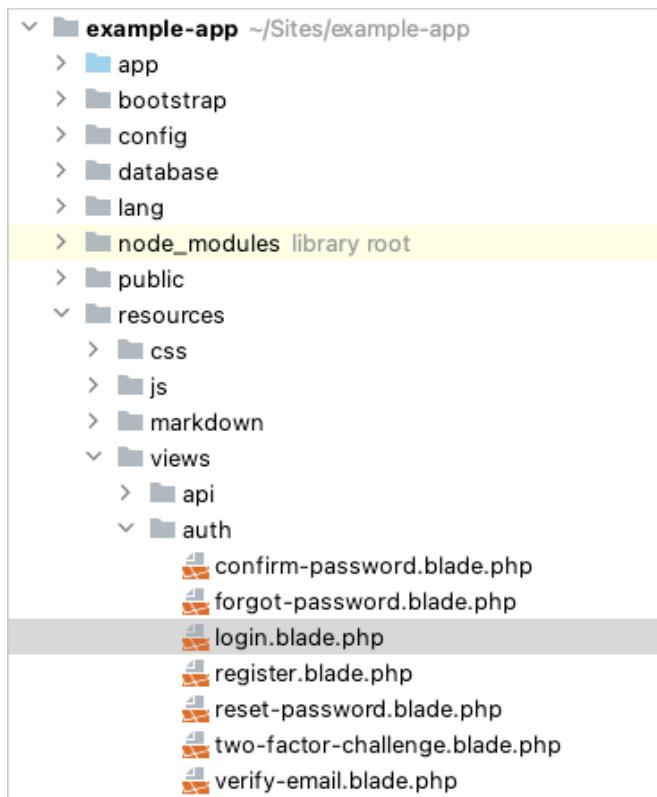
[MacBook-Pro:~/Sites/example-app] ➜  php artisan route:list
GET|HEAD / ..... TestController@index
POST _ignition/execute-solution ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user ..... 
GET|HEAD dashboard ..... dashboard
GET|HEAD forgot-password ..... password.request > Laravel\Fortify > PasswordResetLinkController@create
POST forgot-password ..... password.email > Laravel\Fortify > PasswordResetLinkController@store
GET|HEAD grades ..... grades.index > GradesController@index
POST grades ..... grades.store > GradesController@store
GET|HEAD grades/create ..... grades.create > GradesController@create
PUT|PATCH grades/{grade} ..... grades.update > GradesController@update
DELETE grades/{grade} ..... grades.destroy > GradesController@destroy
GET|HEAD grades/{grade}/edit ..... grades.edit > GradesController@edit
GET|HEAD livewire/livewire.js ..... Livewire\Controllers > LivewireJavaScriptAssets@source
GET|HEAD livewire/livewire.js.map ..... Livewire\Controllers > LivewireJavaScriptAssets@maps
POST livewire/message/{name} ..... livewire.message > Livewire\Controllers > HttpConnectionHandler
GET|HEAD livewire/preview-file/{filename} ..... livewire.preview-file > Livewire\Controllers > FilePreviewHandler@handle
POST livewire/upload-file ..... livewire.upload-file > Livewire\Controllers > FileUploadHandler@handle
GET|HEAD login ..... login > Laravel\Fortify > AuthenticatedSessionController@create
POST login ..... Laravel\Fortify > AuthenticatedSessionController@store
POST logout ..... logout > Laravel\Fortify > AuthenticatedSessionController@destroy
GET|HEAD register ..... register > Laravel\Fortify > RegisteredUserController@create
POST register ..... register > Laravel\Fortify > RegisteredUserController@store
POST reset-password ..... password.update > Laravel\Fortify > NewPasswordController@store
GET|HEAD reset-password/{token} ..... password.reset > Laravel\Fortify > NewPasswordController@create
GET|HEAD sanctum/csrf-cookie ..... Laravel\Sanctum > CsrfCookieController@show
GET|HEAD students ..... students.index > StudentController@index
POST students ..... students.store > StudentController@store
GET|HEAD students/create ..... students.create > StudentController@create
GET|HEAD students/{student} ..... students.show > StudentController@show
PUT|PATCH students/{student} ..... students.update > StudentController@update
DELETE students/{student} ..... students.destroy > StudentController@destroy
GET|HEAD students/{student}/edit ..... students.edit > StudentController@edit
POST students/{student}/grade-student ..... StudentController@associateStudentGrade
PUT students/{student}/grade-student/{grade_student} ..... StudentController@updateStudentGrade
GET|HEAD test ..... 
GET|HEAD testdb ..... TestController@testDBConnection
GET|HEAD two-factor-challenge ..... two-factor.login > Laravel\Fortify > TwoFactorAuthenticatedSessionController@create
POST two-factor-challenge ..... Laravel\Fortify > TwoFactorAuthenticatedSessionController@store
GET|HEAD user/confirm-password ..... Laravel\Fortify > ConfirmablePasswordController@show
POST user/confirm-password ..... password.confirm > Laravel\Fortify > ConfirmablePasswordController@store
GET|HEAD user/confirmed-password-status ..... password.confirmation > Laravel\Fortify > ConfirmedPasswordStatusController@show
POST user/confirmed-two-factor-authentication/two-factor.confirm > Laravel\Fortify > ConfirmedTwoFactorAuthenticationController@store
PUT user/password ..... user-password.update > Laravel\Fortify > PasswordController@update
GET|HEAD user/profile ..... profile.show > Laravel\Jetstream > UserProfileController@show
PUT user/profile-information ..... user-profile-information.update > Laravel\Jetstream > ProfileInformationController@update
POST user/two-factor-authentication ..... two-factor.enable > Laravel\Fortify > TwoFactorAuthenticationController@store
DELETE user/two-factor-authentication ..... two-factor.disable > Laravel\Fortify > TwoFactorAuthenticationController@destroy
GET|HEAD user/two-factor-qr-code ..... two-factor.qr-code > Laravel\Fortify > TwoFactorQrCodeController@show
GET|HEAD user/two-factor-recovery-codes ..... two-factor.recovery-codes > Laravel\Fortify > RecoveryCodeController@index
POST user/two-factor-recovery-codes ..... Laravel\Fortify > RecoveryCodeController@store
GET|HEAD user/two-factor-secret-key ..... two-factor.secret-key > Laravel\Fortify > TwoFactorSecretKeyController@show

```

Con esto ya tenemos las rutas creadas. Si todos los pasos se realizaron de manera correcta, al ingresar a la ruta login del aplicativo deberíamos ver una interfaz como la que se muestra a continuación:

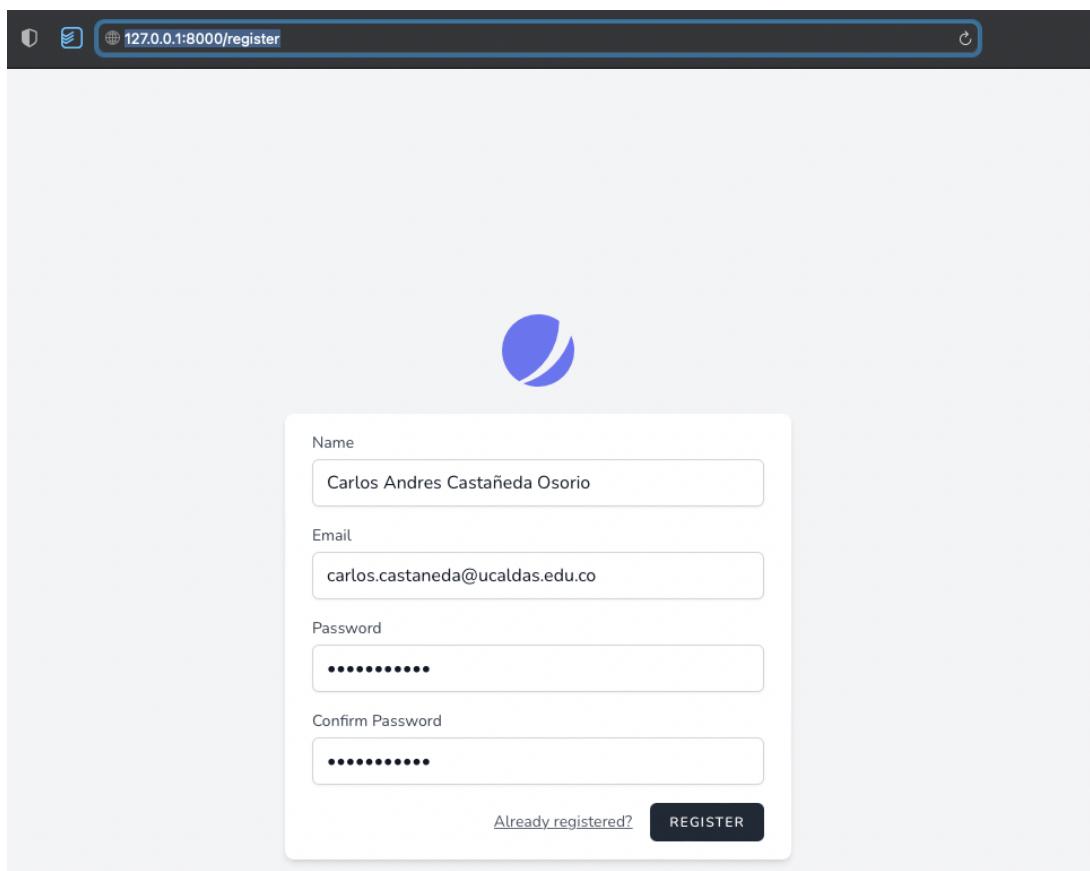


Todas las interfaces son modificables, para ello nos podemos dirigir a la siguiente ruta:



Con esto ya podemos registrar nuevos usuarios, loguearnos y realizar algunas acciones sólo para las acciones protegidas. En este punto se

recomienda ser curioso y visitar las rutas. Por ejemplo para registrar un nuevo usuario se encuentra la ruta /register y se ve como se muestra a continuación:



Para proteger las rutas existentes teniendo como prerequisito que el usuario actual se encuentre logueado realizamos el siguiente cambio en nuestro archivo de rutas.

```
Route::resource('name: '/students', controller: \App\Http\Controllers\StudentController::class)
->middleware(middleware: 'auth');
```

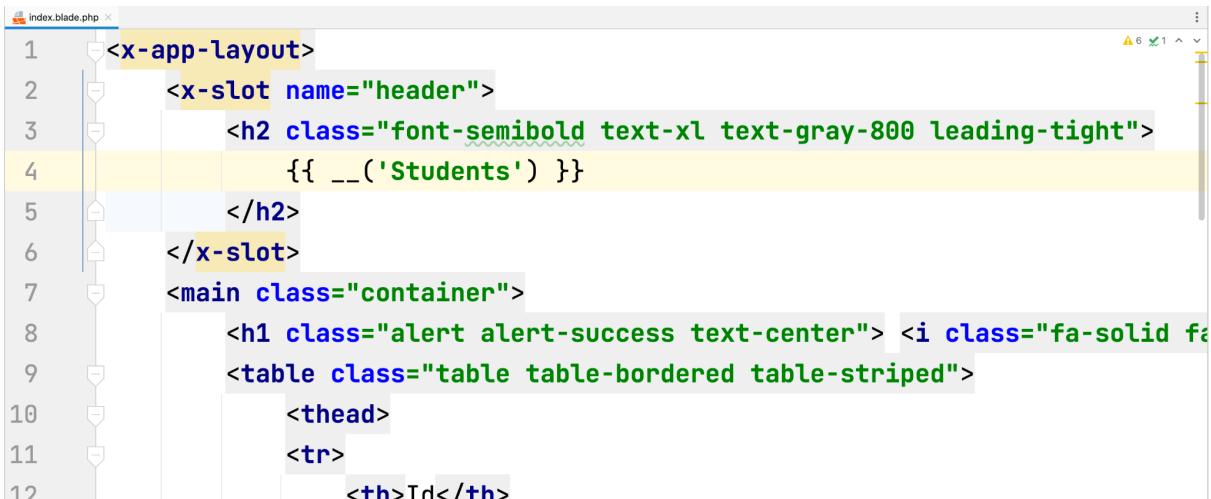
Otra opción para realizar esto mismo es desde el constructor de las clases, como se muestra a continuación:

```

class GradesController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }
}

```

En adelante, la manera de extender de los layout cambió (aún funciona la manera previa de hacerlo) y se empezó a utilizar el concepto de componentes, este concepto no hace parte del presente curso por lo que se invita a revisar la [documentación oficial de blade](#) en donde se evidencian los cambios en el código. A continuación, vamos a modificar el archivo de rutas y todas nuestras interfaces para que usen el *layout app*, para esto realizamos los siguientes cambios:



```

<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Students') }}
        </h2>
    </x-slot>
    <main class="container">
        <h1 class="alert alert-success text-center"> <i class="fa-solid fa-table" style="font-size: 2em; color: #007bff; margin-right: 10px;"></i> Students
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th>Id</th>

```

Obteniendo el siguiente resultado:

The screenshot shows a web-based application interface for managing students. At the top left is a blue circular logo. Next to it are the words "Dashboard". On the right side, there is a user profile box with the name "Carlos Castañeda" and a dropdown arrow. Below the header, the word "Estudiantes" is displayed. A green header bar contains the text "Students list". The main content area is a table with the following columns: Id, Document, Code, Name, Last Name, Birth Date, Show, Edit, and Delete. There are two rows of data:

Id	Document	Code	Name	Last Name	Birth Date	Show	Edit	Delete
1	1053832793	00007114	Carlos Andres	Castaneda Osorio	2022-02-23	<button>Show</button>	<button>Edit</button>	<button>Delete</button>
4	764537	002	Pepito	Perez	1991-04-22	<button>Show</button>	<button>Edit</button>	<button>Delete</button>

At the bottom of the table is a green button labeled "Create new student".

Con todos estos pasos cubiertos ya se tiene el proyecto completo y se permite gestionar las notas de los estudiantes de una materia.