

Título de la tarea:

PROGRAMACIÓN ORIENTADA A OBJETOS EN PHP MODELO-VISTA-CONTROLADOR FRAMEWORK WEB LARAVEL

Unidad de Trabajo 4

Módulo profesional: Desarrollo Web en Entorno Servidor

Ciclo Superior de Desarrollo de Aplicaciones Web

Curso: 2022/2023.

23 de abril de 2023

*Trabajo realizado por: Pedro Godoy Polaina
Ciclo Formativo Grado Superior Desarrollo Aplicaciones Web.
IES Trassierra (Córdoba).*

Índice de contenido

INTRODUCCIÓN.....	3
ACTIVIDADES DE OOP, MVC Y OTROS.....	4
1. Repasa la POO y detalla sus conceptos y características principales.....	4
2. Ejecuta y explica todos los ejemplos del tutorial que se indica: https://www.w3schools.com/php/php_oop_what_is.asp	5
Definición de una clase:.....	5
Palabra clave this.....	6
Instance Of.....	7
Constructor. La función __construct.....	7
La función __destruct.....	8
Modificadores de acceso.....	8
Herencia. Palabra clave "extends".....	9
Herencia y modificador de acceso protegido.....	9
Anulación de métodos heredados.....	10
La palabra clave final.....	11
Constantes de clase. "const".....	11
Clases abstractas y método abstractos.....	11
Interfaces.....	13
Trait. Rasgos.....	14
Métodos Estáticos.....	14
Propiedades estáticas.....	15
Namespaces. Espacios de nombres de PHP.....	16
Iterables de PHP.....	16
3. Resuelve estos diez ejercicios sobre clases usando POO en PHP.....	17
Ejercicio 1. Creación de la clase Punto.....	17
Ejercicio 2. Creación de la clase Línea.....	18
Ejercicio 3. Creación de la clase Rectangulo.....	19
Ejercicio 4. Clase Circulo con cálculo de área y circunferencia.....	19
Ejercicio 5. Clase Estudiante.....	20
Ejercicio 6. Cálculo de distancia euclídea de dos puntos.....	21
Ejercicio 7. Clase línea Linea2D.....	22
Ejercicio 8. Clase Forma.....	24
Ejercicio 9. Clase Estudiante ampliación de funciones.....	26
Ejercicio 10. Clases A, B, C. Herencia.....	28
4. Explica lo que es patrón de arquitectura y enumera todos sus ejemplos.....	30
5. Define programación por capas y sus capas como BLL (lógica de negocio) o DAL y detalla la three-tier.....	32
6. Explica el patrón de arquitectura MVC, sus componentes y cómo interactúan.....	33
7. Resumen de vídeo explicativo de MVC. Explica el flujo de MVC empezando por la vista.....	33
8. Plantilla MVC básica. Explicación. Modificaciones.....	34
9. Explica al detalle lo que es un framework web.....	36
10. Explica al detalle lo que es Laravel.....	37
ACTIVIDADES DE LARAVEL.....	39
1. Framework web e instalación.....	39
A) MVC con Routing.....	39
B) framework web.....	40
C) Laravel.....	40
D) Instalación de Laravel 10.....	41
E) Creación de un proyecto en blanco de Laravel 10.....	42
2. Rutas y controladores.....	44
3. Vistas mediante Blade.....	47
4. Bases de datos.....	49
6. CRUD.....	57

DESPLIEGUE EN ALWAYS DATA.NET.....	65
CONCLUSIÓN DE LA TAREA.....	66

INTRODUCCIÓN

El presente documento titulado: "*POO en PHP, MVC y el framework web Laravel.*" tiene como objetivo la realización de la tarea de la unidad 4 del módulo de Desarrollo de Aplicaciones en Entorno Servidor, del Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web.

Este documento está dedicado especialmente a reflejar la parte teórica que se pide para la tarea. Si bien es cierto que he realizado muchas más anotaciones con objeto de dejar constancia de mi forma de realizar los ejercicios y con la idea de volver más adelante a retomar este ejercicio para finalizarlo completo. Por ello he añadido una especie de tutorial del modo en que voy avanzando en la tarea.

Dejo algunas capturas de pantalla donde se ve mi usuario de la plataforma de estudio y con esto demuestro la autoría de los ejercicios tal y como se pide. Pero he querido insertar mucho más código explicativo y lo he hecho sin capturas de imagen para hacerlo más útil ya que en cualquier momento puedo copiarlo, pegarlo y modificarlo. De esta forma hago que el documento no se alargue en una extensión sin sentido.

Entiendo que el documento se ha hecho muy largo, pero añado un índice que facilita la navegación y ayuda para la corrección del profesor.

ACTIVIDADES DE OOP, MVC Y OTROS

1. Repasa la POO y detalla sus conceptos y características principales.

La programación orientada a objetos (POO) es un paradigma de programación que aplica una idea basada en que los programas informáticos son un compendio de objetos que interactúan entre sí para realizar las tareas encargadas a los programas informáticos.

Los objetos son instancias o creaciones individualizadas basadas en una "plantilla" llamada "clase" en la que encontramos la definición de los objetos, sus características, atributos y comportamientos. Los objetos creados partiendo de una clase encapsulan datos y operaciones relacionadas en una sola entidad.

Los objetos pueden comunicarse entre sí mediante llamadas a métodos de otros objetos, si bien cada objeto tiene su propia estructura interna y su propio estado, que se mantiene oculto al resto del programa.

En la POO son claves los conceptos de herencia y el polimorfismo, que permiten crear clases basadas en otras clases y utilizar objetos de diferentes clases de manera intercambiable.

La POO aporta a la programación de software grandes **ventajas**, como son:

- **Modularidad** ya que permite dividir los programas en partes independientes con la capacidad de comunicarse entre ellas.
- **Principio de ocultación:** Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una "interfaz" a otros objetos que detalla cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas.
- **Extensibilidad.** Facilita la labor de añadir nuevas características a los programas, con nuevos métodos en los objetos, o con nuevos objetos que extienden el comportamiento los ya existentes.
- **Mantenimiento.** Los programas son más sencillos de mantener, debido a su estructura más organizada y modular.

Para entender la POO debemos conocer los siguientes **conceptos** asociados a ella.

- **Clase:** es la "plantilla" en la que se definen los atributos y métodos predeterminados de un tipo de objeto.
- **Objeto:** Instancia de una clase. Está dotada de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos).
- **Atributos:** Características que tiene la clase. También podemos entenderlo como el contenedor o variable de un tipo de dato asociado a un objeto que podría ser alterado mediante la ejecución de algún método.
- **Método:** Son las acciones que un objeto puede realizar que se implementan mediante algoritmos asociados a este objeto. Este algoritmo puede generar un cambio en el propio objeto o un evento hacia otra parte del programa.

- **Mensaje:** las comunicaciones dirigidas a los objetos que desencadenan la ejecución de un método.
- **Evento:** es la reacción que puede desencadenar un objeto a través de un método.
- **Estado interno:** variable privada que solo puede ser accedida y alterada por un método del objeto. No es visible al programador que maneja una instancia de la clase.
- **Miembros de un objeto:** Atributos, identidad, relaciones y métodos.

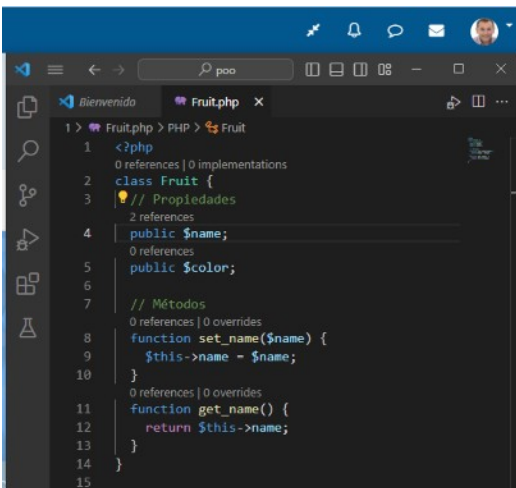
Los conceptos que forman el ***pilar fundamental de la POO*** son:

- **Herencia.** Es el proceso de crear una clase a partir de otra, heredando su comportamiento y características, como si fueran propias y que además da la oportunidad de poder ser redefinidos estos comportamientos y características en la nueva clase. Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación.
- **Abstracción.** Cada clase oculta en su interior las peculiaridades de su implementación, y presenta al exterior una serie de métodos (interface) cuyo comportamiento está bien definido. Visto desde el exterior, cada objeto es un ente abstracto que realiza un trabajo.
- **Polimorfismo.** Un mismo método puede tener comportamientos distintos en función del objeto con que se utilice. Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando.
- **Encapsulación.** En la POO se juntan en un mismo lugar los datos y el código que los manipula. Es decir, reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción.

2. Ejecuta y explica todos los ejemplos del tutorial que se indica:

https://www.w3schools.com/php/php_oop_what_is.asp

Definición de una clase:



```

1 <?php
2 class Fruit {
3     // Propiedades
4     public $name;
5     public $color;
6
7     // Métodos
8     function set_name($name) {
9         $this->name = $name;
10    }
11    function get_name() {
12        return $this->name;
13    }
14 }
15
  
```

Una clase se define usando la palabra clave "class"

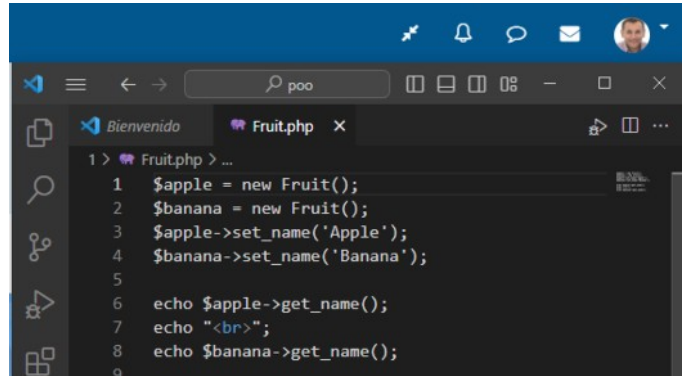
En este ejemplo declaramos una clase llamada Fruit que consta de dos propiedades (\$name y \$color) y dos métodos set_name() y get_name() para configurar y obtener la propiedad \$name:

Añadimos código al anterior esta vez para definir objetos instanciando la clase que hemos creado.

Los objetos de una clase se crean utilizando la palabra clave new.

\$apple y \$banana son instancias de la clase Fruit.

Añado además la presentación en pantalla de los atributos de los objetos. El resultado en pantalla es: Apple Banana



Para terminar este ejemplo añado un método más: get_color

```

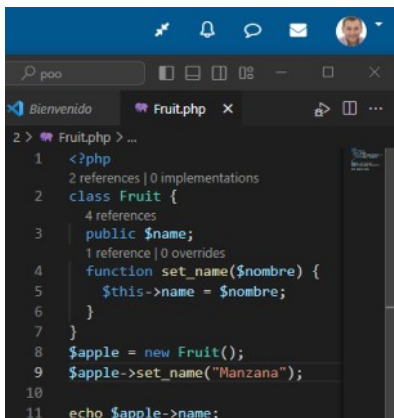
<?php
class Fruit {
    // Propiedades
    public $name;
    public $color;

    // Métodos
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
    function set_color($color) {
        $this->color = $color;
    }
    function get_color() {
        return $this->color;
    }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();

```

Palabra clave this



La palabra clave \$this se refiere al objeto actual y solo está disponible dentro de los métodos.

Con este ejemplo cambiamos el valor del atributo \$name dentro de la clase (agregando un método set_name() y usando \$this)

Observamos que name (sin \$) se refiere al propio atributo y en este caso \$nombre se refiere al parámetro de entrada de la función.

¿Nos permite cambiar el valor de la propiedad \$name desde fuera del objeto? Sí.

```
<?php
class Fruit {
    public $name;
}
$apple = new Fruit();
$apple->name = "Apple";

echo $apple->name;
?>
```

Instance Of

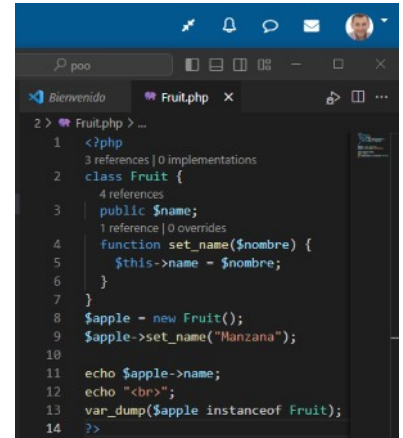
Nos devuelve un booleano indicando a que clase pertenece el objeto.

Modifico el programa anterior y añado nueva linea,

```
<?php
    var_dump($apple instanceof Fruit);
?>
```

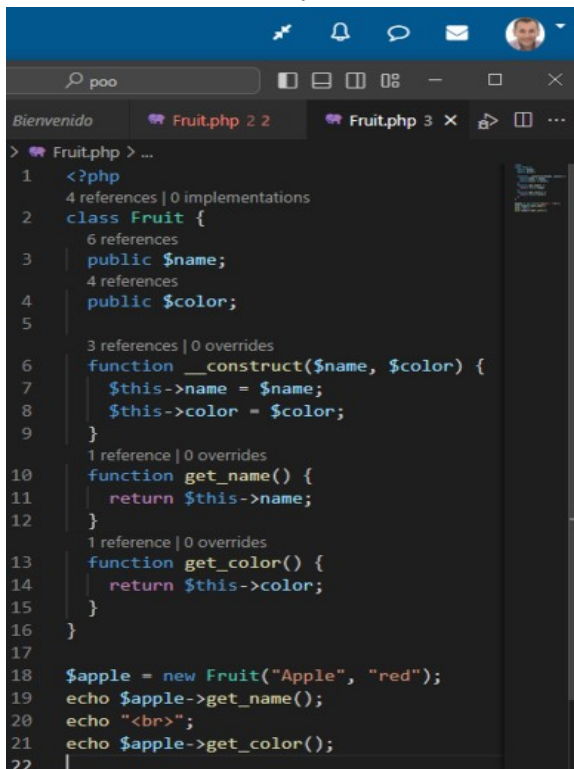
obteniendo el resultado:

```
bool(true)
```



Constructor. La función __construct

Un constructor nos permite inicializar las propiedades de un objeto al crear el objeto.



Si crea una __construct() función, PHP llamará automáticamente a esta función cuando cree un objeto de una clase.

La función de construcción comienza con dos guiones bajos (__)

La función constructora sería en este ejemplo:

```
function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
}
```

cuando instanciamos el objeto pasamos directamente los parámetros

```
$apple = new Fruit("Apple");
```

Esto nos permite reducir código al no tener que llamar al método set

```
<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    function get_name() {
        return $this->name;
    }
    function get_color() {
        return $this->color;
    }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
```

La función **__destruct**

Se llama a un destructor cuando se destruye el objeto o se detiene o se sale del script.

```
function __destruct() {
    echo "The fruit is {$this->name}.";
}
```

Modificadores de acceso

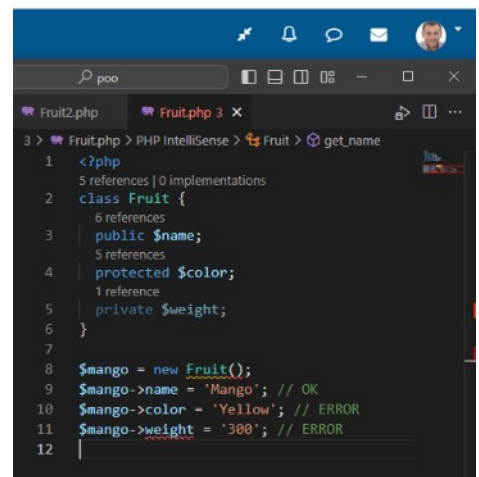
Las propiedades y los métodos pueden tener modificadores de acceso que controlan dónde se puede acceder a ellos. Hay tres modificadores de acceso:

public- se puede acceder a la propiedad o método desde cualquier lugar. esto es por defecto

protected- se puede acceder a la propiedad o método dentro de la clase y por clases derivadas de esa clase

private- SOLO se puede acceder a la propiedad o método dentro de la clase

Si ejecutamos el ejemplo vemos que solo es válida la asignación de \$name, ya que \$color y \$mango nos arrojan error por tener los modificadores de acceso protected y private.



Ejemplo en el que los atributos son públicos y se les ha aplicado modificador a los métodos.

```
<?php
class Fruit {
    public $name;
    public $color;
    public $weight;

    function set_name($n) { // función pública, por defecto
        $this->name = $n;
    }
    protected function set_color($n) { // función protected
        $this->color = $n;
    }
    private function set_weight($n) { // función private
        $this->weight = $n;
    }
}
```



```

    }

    $mango = new Fruit();
    $mango->set_name('Mango'); // OK
    $mango->set_color('Yellow'); // ERROR
    $mango->set_weight('300'); // ERROR
    ?>

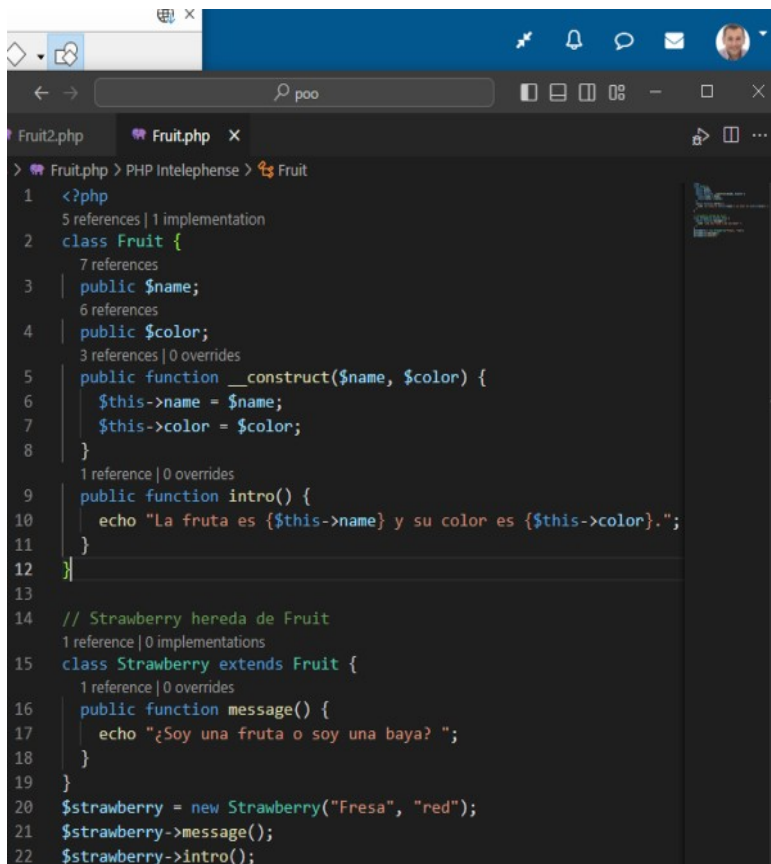
```

Herencia. Palabra clave “extends”

Llamamos así cuando una clase deriva de otra clase.

La clase secundaria heredar  todas las propiedades y m todos p blicos y protegidos de la clase principal. Adem s, puede tener sus propias propiedades y m todos.

Una clase heredada se define utilizando la palabra clave extends



En el ejemplo vemos como se ha declarado la clase Fruit

```

class Fruit {
    // ...atributos
    // ...Funciones
}

```

Luego hemos declarado la clase Strawberry que hereda de Fruta

```

// Strawberry hereda de Fruit
class Strawberry extends Fruit {
    public function message() {
        echo " Soy una fruta o soy una baya? ";
    }
}

```

Ahora Strawberry tiene las propiedades y m todos de Fruta y adem s a ade su propio m todo message()

Herencia y modificador de acceso protegido

Como vimos el modificador “protected”, con este modificador se puede acceder a las propiedades o m todos dentro de la clase y tambi n por clases derivadas de esa clase.

Este ejemplo que pongo es el mismo que el anterior, pero vemos que ahora el m todo intro es protected. En este ejemplo que inserto vemos que si intentamos llamar a un m todo “protected” (intro()) desde fuera de la clase, recibiremos un error.

```

<?php
class Fruit {
    public $name;
    public $color;
    public function __construct($name, $color) {

```

```

        $this->name = $name;
        $this->color = $color;
    }
    protected function intro() {
        echo "La fruta es {$this->name} y su color es {$this->color}.";
    }
}

// Strawberry hereda de Fruit
class Strawberry extends Fruit {
    public function message() {
        echo "¿Soy una fruta o soy una baya? ";
    }
}

// Prueba de llamada a metodos desde fuera de la clase
$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() es público
$strawberry->message(); // OK. message() es público
$strawberry->intro(); // ERROR. intro() es protected

```

En este siguiente ejemplo todo funciona bien, Es porque llamamos al método protected intro() desde dentro de la clase derivada. Como vemos la llamada está en la clase Strawberry que es pública y llama a intro() (que está protegido) desde dentro de la clase derivada.

```

<?php
class Fruit {
    public $name;
    public $color;
    public function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    protected function intro() {
        echo "The fruit is {$this->name} and the color is {$this->color}.";
    }
}

class Strawberry extends Fruit {
    public function message() {
        echo "Am I a fruit or a berry? ";
        // Call protected method from within derived class - OK
        $this->intro();
    }
}

$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() es público
$strawberry->message(); // OK. message() es público y llama a intro() (que está
protegido) desde dentro de la clase derivada
?>

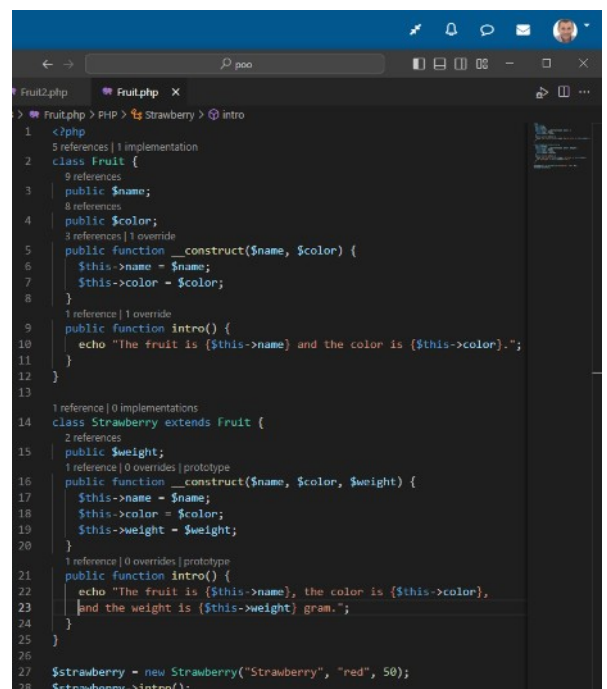
```

Anulación de métodos heredados

Los métodos heredados se pueden anular redefiniendo los métodos, usando el mismo nombre en la clase secundaria.

Vemos como la clase Fruit define el método intro()

Posteriormente la clase Strawberry que hereda de Fruit vuelve a definir la función, anulando con ello la definición anterior.



La palabra clave final

La palabra clave "final" se puede usar para evitar la herencia de clases o para evitar la anulación de métodos. Una clase definida con "final" nunca tendrá clases que hereden de ella.

```
<?php
final class Fruit {
    // código de la clase
}

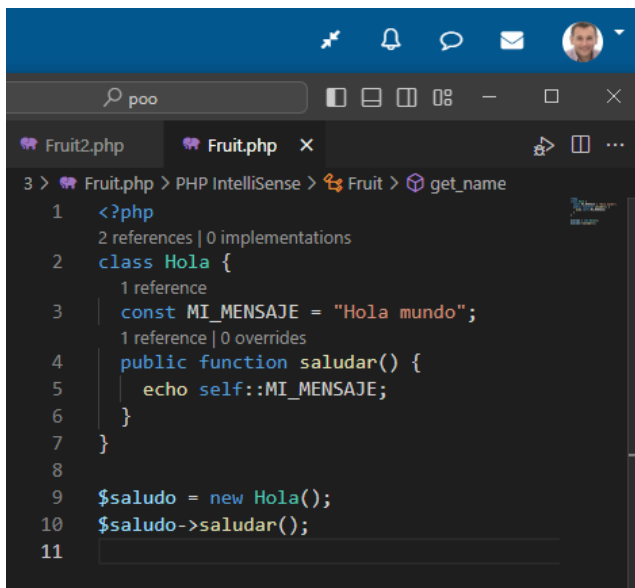
// Esto es un error
class Strawberry extends Fruit {
    // código de la clase
}
?>
```

Constantes de clase. "const"

Las constantes no se pueden cambiar una vez que se declaran.

Se definen con la palabra clave "const" y se recomienda que estén en MAYÚSCULA

Podemos acceder a una constante desde fuera de la clase usando el nombre de la clase seguido del operador de resolución de alcance (::) seguido del nombre de la constante:



Aquí vemos como en la clase Hola se declara la constante MI_MENSAJE

Se realiza la llamada a la constante desde un método de la propia clase usando

```
echo self::MI_MENSAJE;
```

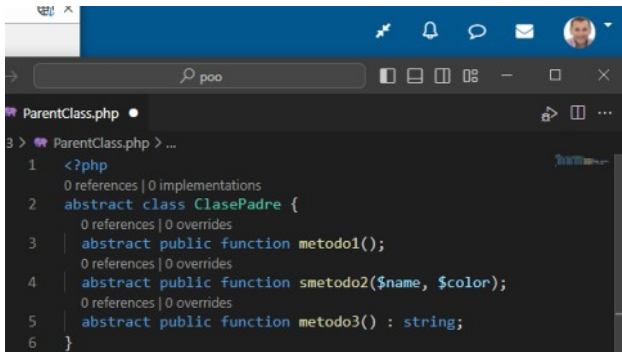
Se instancia un objeto de la clase finalmente y luego se llama al método del objeto instanciado

Clases abstractas y método abstractos

Las clases y métodos abstractos son cuando la clase principal tiene un método con nombre, pero necesita su(s) clase(s) secundaria(s) para completar las tareas.

Una clase abstracta es una clase que contiene al menos un método abstracto. Un método abstracto es un método que se declara, pero no se implementa en el código.

Una clase o método abstracto se define con la palabra clave "abstract"



En este ejemplo vemos que he declarado una clase llamada ClasePadre, declarada como clase abstracta.

Contiene tres métodos declarados también como abstractos y que solo se hace la definición y se declaran cuáles serán sus parámetros de entrada, pero no se declara código ninguno ni métodos ni atributos.

Estos deberán ir implementados en otro lugar.

Ahora vemos qué debemos tener en cuenta:

- El método de la clase secundaria, al heredar de una clase abstracta debe definirse con el mismo nombre y el mismo modificador de acceso restringido o menos.
- El método de la clase secundaria debe definirse con el mismo modificador de acceso o uno menos restringido.
- El número de argumentos requeridos debe ser el mismo. Sin embargo, la clase secundaria puede tener argumentos opcionales además

Ejemplo:

```

<?php
// Clase padre
abstract class Coche {
    public $name;
    public function __construct($name) {
        $this->name = $name;
    }
    abstract public function intro() : string;
}

// Clases hijas
class Audi extends Coche {
    public function intro() : string {
        return "Coche alemán! soy $this->name!";
    }
}

class Volvo extends Coche {
    public function intro() : string {
        return "Coche sueco! Yo soy $this->name!";
    }
}

class Citroen extends Coche {
    public function intro() : string {
        return "Coche francés! Yo soy $this->name!";
    }
}

// Creación de objetos de las clases hijas
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();

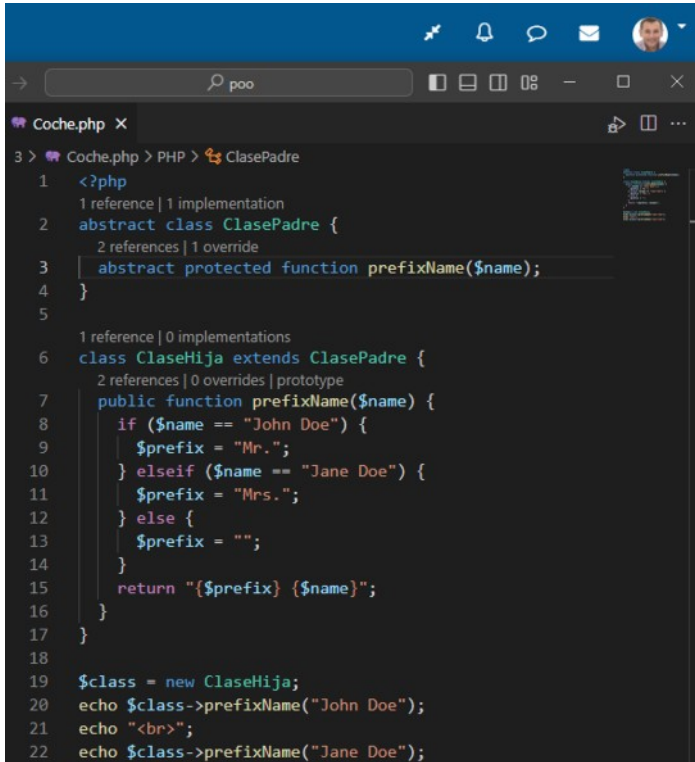
```

Vemos que se declara en la clase padre una función abstracta intro() y se define el tipo de salida de datos que será string. De esta clase heredarán tres clases que pueden usar la propiedad

pública \$name así como el método público __construct() de la clase Car debido a la herencia.

En cada una de las clases hijas se define el código de la función intro() siendo distinto para cada tipo de clase.

Se realiza la instanciación de los objetos y se llama a su clase intro() devolviendo la cadena de texto.



```

3 > Coche.php > PHP > ClasePadre
1 <?php
2 1 reference | 1 implementation
  abstract class ClasePadre {
3 2 references | 1 override
    abstract protected function prefixName($name);
4  }
5
6 1 reference | 0 implementations
  class ClaseHija extends ClasePadre {
7 2 references | 0 overrides | prototype
    public function prefixName($name) {
8      if ($name == "John Doe") {
9          $prefix = "Mr.";
10     } elseif ($name == "Jane Doe") {
11         $prefix = "Mrs.";
12     } else {
13         $prefix = "";
14     }
15     return "{$prefix} {$name}";
16 }
17 }
18
19 $class = new ClaseHija;
20 echo $class->prefixName("John Doe");
21 echo "<br>";
22 echo $class->prefixName("Jane Doe");
  
```

Otro ejemplo, esta vez donde el método abstracto prefixName() tiene un argumento

En el código de la clase hija se define una nueva cadena de texto de salida dependiendo del argumento de entrada

salida en pantalla:

```

Mr. John Doe
Mrs. Jane Doe
  
```

Otro ejemplo en el que el método abstracto tiene un argumento y la clase secundaria tiene dos argumentos opcionales que no están definidos en el método abstracto del padre:

```

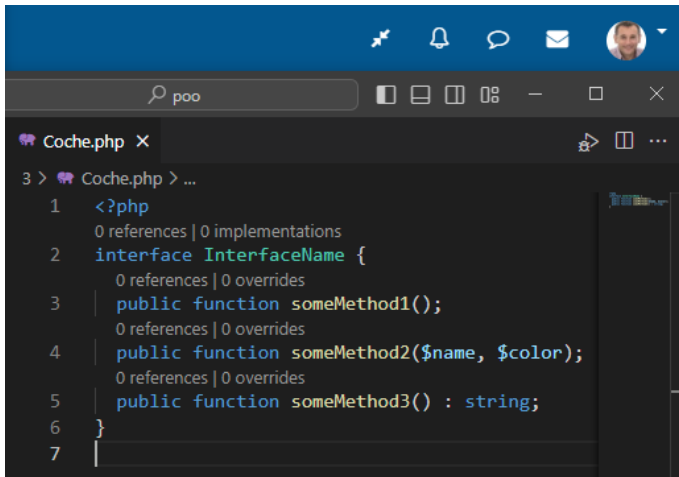
<?php
abstract class ClasePadre {
    abstract protected function prefixName($name);
}

class ClaseHija extends ClasePadre {
    public function prefixName($name, $separator = ".", $greet = "Dear") {
        if ($name == "John Doe") {
            $prefix = "Mr";
        } elseif ($name == "Jane Doe") {
            $prefix = "Mrs";
        } else {
            $prefix = "";
        }
        return "{$greet} {$prefix}{$separator} {$name}";
    }
}

$class = new ClaseHija;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
  
```

Interfaces

Las interfaces le permiten especificar qué métodos debe implementar una clase. Las interfaces facilitan el uso de una variedad de clases diferentes de la misma manera. Cuando una o más clases utilizan la misma interfaz, se denomina "polimorfismo"



La interfaz es similar a las clases abstractas. La diferencia entre interfaces y clases abstractas son:

- Las interfaces no pueden tener propiedades, mientras que las clases abstractas pueden
- Todos los métodos de interfaz deben ser públicos, mientras que los métodos de clase abstracta son públicos o protegidos.
- Todos los métodos en una interfaz son abstractos, por lo que no se pueden implementar en el código y la palabra clave abstracta no es necesaria.
- Las clases pueden implementar una interfaz mientras heredan de otra clase al mismo tiempo

Trait. Rasgos.

PHP solo admite herencia simple: una clase secundaria solo puede heredar de un solo padre. Si queremos que una clase herede de múltiples clases, para heredar sus comportamientos, los Traits resuelven este problema.

Los rasgos se declaran con la palabra clave "trait"

```

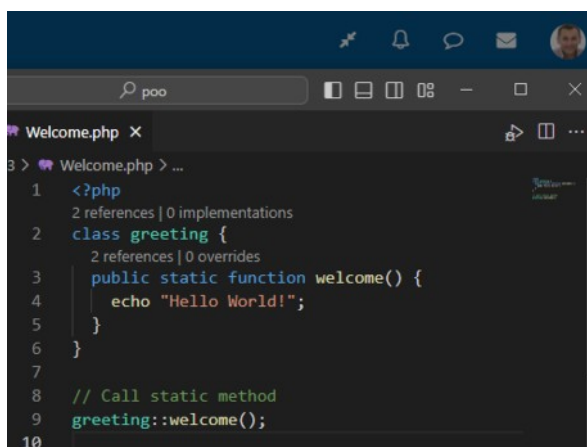
//DECLARANDO UN RASGO
<?php
trait TraitName {
    // some code...
}
?>

// USÁNDO UN RASGO DE OTRA CLASE
<?php
class MyClass {
    use TraitName;
}
?>

```

Métodos Estáticos

Los métodos estáticos se pueden llamar directamente, sin crear primero una instancia de la clase. Se declaran con la palabra clave "static"



En el ejemplo vemos como se declara el método estático welcome() dentro de la clase greeting.

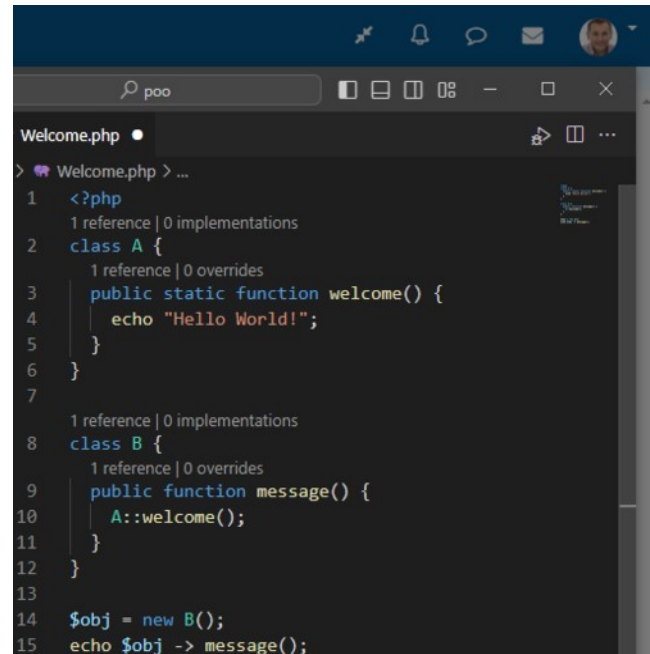
Fuera de la clase llamamos al método estático son necesidad de hacer una instancia de la clase.

```
greeting::welcome();
```

En este otro ejemplo vemos como el método estático `welcome()` es llamado desde un método de una clase distinta.

Para hacer esto, el método estático ha sido declarado como "public".

La llamada se realiza instanciando primero un objeto llamado `$obj` de la clase B y desde ahí hacemos la llamada a la función `message()` que a su vez ha tomado el método estático de la Clase A.



Propiedades estáticas

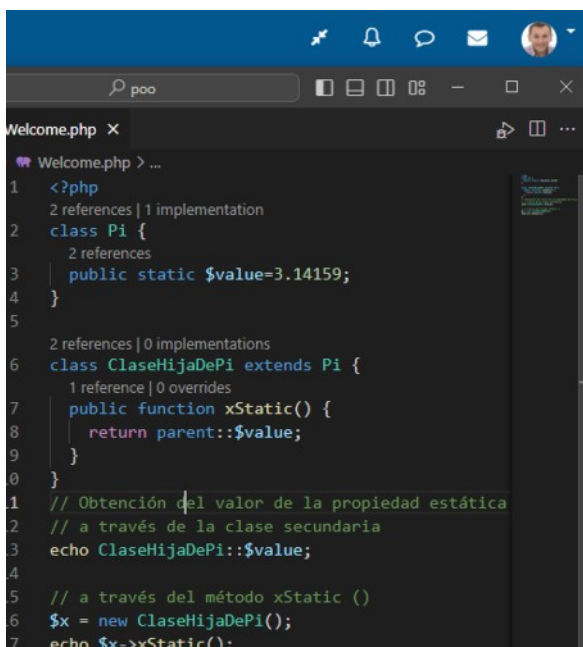
Las propiedades estáticas se pueden llamar directamente, sin crear una instancia de una clase.

```
class Pi {
    public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

En este ejemplo para acceder a la propiedad estática de Pi llamada `$value`, usamos el nombre de la clase, dos puntos dobles (`::`) y el nombre de la propiedad

Para llamar a una propiedad estática desde una clase secundaria, vamos a usar la **palabra clave "parent"** dentro de la clase secundaria:



En este ejemplo vemos como se crea una clase llamada Pi que tiene un atributo público y estático llamado `$value`

Declaramos una clase que hereda de Pi. Esta clase tiene una función llamada `xStatic` que retorna un valor obtenido de la siguiente forma: **return parent::\$value;** o sea, usando `parent` está llamando al valor que se encuentra en la clase padre.

Ahora mostramos el valor obteniéndolo a de la propiedad estática a través de la clase secundaria.

```
echo ClaseHijaDePi::$value;
```

O también, usando el método `xStatic()` de la clase

hija

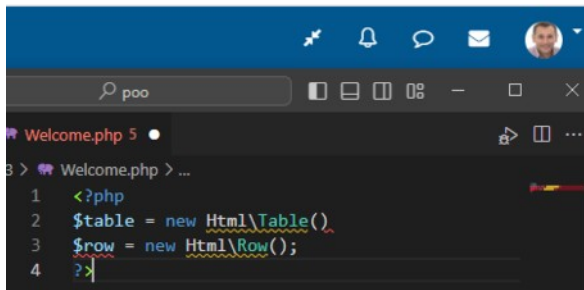
```
$x = new ClaseHijaDePi();
echo $x->xStatic();
```

Namespaces. Espacios de nombres de PHP

Los espacios de nombres son calificadores que resuelven dos problemas diferentes:

- Permiten una mejor organización al agrupar clases que trabajan juntas para realizar una tarea.
- Permiten utilizar el mismo nombre para más de una clase

Los espacios de nombres se declaran al principio de un archivo usando la palabra clave "namespace". Una declaración de namespace debe ser lo primero en el archivo PHP.

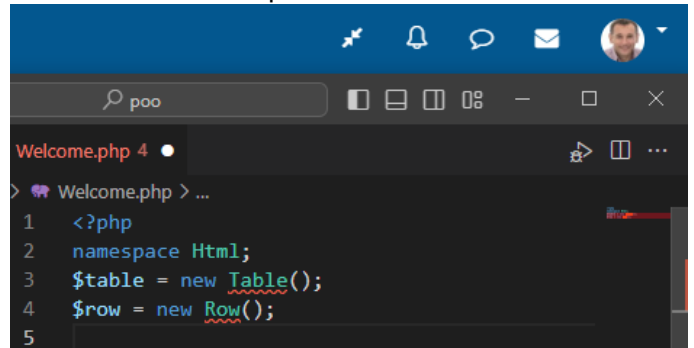


Voy a explicar este ejemplo escrito de dos formas, la primera uso clases del espacio de nombres Html, escribiendo primero el nombre del calificador del espacio de nombres, ya que estamos fuera de este espacio de nombres y por lo tanto la clase debe llevar el nombre del calificador.

La otra forma más fácil, es que cuando se usan muchas clases del mismo espacio de nombres al mismo tiempo, es más fácil usar la palabra clave namespaces:

Se pueden usar alias de la siguiente forma:

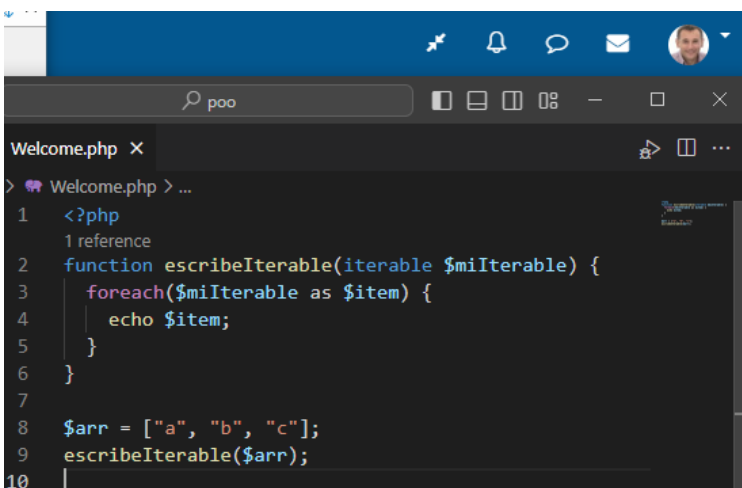
```
<?php
use Html as H;
$table = new H\Table();
?>
```



Iterables de PHP

Un iterable es cualquier valor que se puede recorrer con un bucle foreach().

Iterable es un pseudotipo y se puede usar como un tipo de datos para argumentos de funciones y valores de retorno de funciones.



Con este ejemplo definimos una función llamada *escribeIterable* que acepta un parámetro iterable llamado *\$miIterable*.

Luego, la función utiliza un bucle foreach para iterar sobre cada elemento del iterable y mostrarlo en la salida estándar usando la función echo.

Para el ejemplo se define un array

con varios valores y luego se llama a la función creada pasando el array como argumento.

El resultado es la impresión en pantalla de los elementos del array.

Otro ejemplo explicado:

```
<?php
function getIterable():iterable {
    return ["a", "b", "c"];
}

$myIterable = getIterable();
foreach($myIterable as $item) {
    echo $item;
}
```

Este código PHP define una función llamada "getIterable" que devuelve un iterable que contiene los valores "a", "b" y "c" y especifica que el tipo de retorno de esta función es "iterable". Luego, se llama a la función "getIterable" y se almacena su valor de retorno en la variable `\$myIterable`. Después se usa un bucle "foreach" para iterar sobre cada elemento del iterable y mostrarlo usando la función "echo".

3. Resuelve estos diez ejercicios sobre clases usando POO en PHP.

Ejercicio 1. Creación de la clase Punto.

Crea una clase llamada Punto con dos propiedades/atributos denominados x e y, con constructor y con cuatro métodos (getter y setter), uno para obtener x, otro para obtener y, otro para modificar x y otro método para modificar y. Crea 3 instancias/objetos de la clase Punto y ejecuta en ellos los cuatro métodos creados.

```
<?php
/*
EJERCICIO 1. Alumno Pedro Godoy Polaina
Crea una clase llamada Punto con dos propiedades/atributos denominados x e y, con
constructor y con cuatro métodos (getter y setter), uno para obtener x, otro para
obtener y, otro para modificar x y otro método para modificar y. Crea 3 instancias/objetos
de la clase Punto y ejecuta en ellos los cuatro métodos creados.
*/

// Creación de la clase Punto
class Punto
{
    // Atributos privados
    private $x;
    private $y;

    // Método constructor con dos parámetros
    public function __construct($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    // Implementación métodos Get
    public function getX(){return $this->x;}
    public function getY(){return $this->y;}
    // Implementación métodos Set
    public function setX($x) {$this->x = $x;}
    public function setY($y) {$this->y = $y;}
}

// Instanciando tres objetos
$punto1 = new Punto(10, 24);
$punto2 = new Punto(11, 12);
$punto3 = new Punto(8, 14);
```

```
// Ejecutanco cada uno de los métodos creados
$punto1->getX();
$punto2->getY();
$punto3->setX(7);
$punto3->setY(8);

echo 'Punto 1: X = ' . $punto1->getX() . ' Y = ' . $punto1->getY() . '<br>';
echo 'Punto 2: X = ' . $punto2->getX() . ' Y = ' . $punto2->getY() . '<br>';
echo 'Punto 3: X = ' . $punto3->getX() . ' Y = ' . $punto3->getY() . '<br>';
```

Ejercicio 2. Creación de la clase Línea.

Crema una clase llamada Línea con cuatro propiedades/atributos denominados x1, x2, y1 e y2, con constructor y con un método que obtenga el punto medio del segmento usando dichas propiedades/atributos. Crema 3 instancias/objetos de la clase Línea y ejecuta en ellos el método creado.

```
<?php
/*
EJERCICIO 2. Alumno Pedro Godoy Polaina
Crema una clase llamada Línea con cuatro propiedades/atributos denominados x1, x2, y1 e y2,
con constructor y con un método que obtenga el punto medio del segmento usando dichas
propiedades/atributos. Crema 3 instancias/objetos de la clase Línea y ejecuta en ellos
el método creado.
*/
class Línea
{
    // Atributos declarados como privados
    private $x1;
    private $x2;
    private $y1;
    private $y2;

    // Método constructor con cuatro parámetros de entrada
    public function __construct($x1, $x2, $y1, $y2)
    {
        $this->x1 = $x1;
        $this->x2 = $x2;
        $this->y1 = $y1;
        $this->y2 = $y2;
    }

    // Método para calcular el punto medio de los segmentos
    public function getPuntoMedio()
    {
        return array(
            'x' => ($this->x1 + $this->x2) / 2,
            'y' => ($this->y1 + $this->y2) / 2
        );
    }
}

// Instanciación de tres objetos de la clase Línea
$linea1 = new Línea(1, 4, 2, 5);
$linea2 = new Línea(6, 9, 7, 10);
$linea3 = new Línea(11, 14, 12, 15);

// Asignación del punto medio para cada instancia de la clase Línea
$puntoMedio1 = $linea1->getPuntoMedio();
$puntoMedio2 = $linea2->getPuntoMedio();
$puntoMedio3 = $linea3->getPuntoMedio();

// Presentación en pantalla del resultado
echo 'Punto medio de la línea 1: (' . $puntoMedio1['x'] . ', ' . $puntoMedio1['y'] . ')<br>';
echo 'Punto medio de la línea 2: (' . $puntoMedio2['x'] . ', ' . $puntoMedio2['y'] . ')<br>';
echo 'Punto medio de la línea 3: (' . $puntoMedio3['x'] . ', ' . $puntoMedio3['y'] . ')<br>';
```

Ejercicio 3. Creación de la clase Rectangulo.

Crema una clase llamada Rectangulo con dos propiedades/atributos denominados longitud y ancho, con constructor y con un método que calcule el area del rectángulo usando dichas propiedades/atributos. Crema 3 instancias/objetos de la clase Rectangulo y ejecuta en ellos el método creado.

```
<?php

/*
EJERCICIO 3. Alumno Pedro Godoy Polaina
Crema una clase llamada Rectangulo con dos propiedades/atributos denominados longitud
y ancho, con constructor y con un método que calcule el area del rectángulo
usando dichas propiedades/atributos. Crema 3 instancias/objetos de la clase
Rectangulo y ejecuta en ellos el método creado.
*/

// Creación de la Clase Rectangulo
class Rectangulo
{
    // Atributos privados
    private $longitud;
    private $ancho;

    // Método constructor con dos parámetros de entrada
    public function __construct($longitud, $ancho)
    {
        $this->longitud = $longitud;
        $this->ancho = $ancho;
    }

    // Método que calculará el área del rectángulo
    public function getArea()
    {
        return $this->longitud * $this->ancho;
    }
}

// Creamos 3 objetos de la clase Rectangulo
$rectangulo1 = new Rectangulo(14, 12);
$rectangulo2 = new Rectangulo(2, 8);
$rectangulo3 = new Rectangulo(10, 19);

// Presentación en pantalla del cálculo del área para cada objeto rectángulo
echo 'Área del rectángulo 1 = ' . $rectangulo1->getArea() . '<br>';
echo 'Área del rectángulo 2 = ' . $rectangulo2->getArea() . '<br>';
echo 'Área del rectángulo 3 = ' . $rectangulo3->getArea() . '<br>';
```

Ejercicio 4. Clase Circulo con cálculo de área y circunferencia.

Crema una clase llamada Circulo con una propiedad/atributo denominado radio, con constructor y con dos métodos que calculen el area del círculo y la circunferencia del círculo usando dichas propiedades/atributos. Crema 3 instancias/objetos de la clase Circulo y ejecuta en ellos los dos métodos creados.

```
<?php

/*
EJERCICIO 4. Alumno Pedro Godoy Polaina
Crema una clase llamada Circulo con una propiedad/atributo denominado radio, con constructor
y con dos métodos que calculen el area del círculo y la circunferencia del círculo
usando dichas propiedades/atributos. Crema 3 instancias/objetos de la clase Circulo y
ejecuta en ellos los dos métodos creados.
*/

// Creación de la Clase Circulo
class Circulo
{
    // Atributos privados
    private $radio;
```

```

// Método constructor con un parámetro de entrada
public function __construct($radio)
{
    $this->radio = $radio;
}

// Método para cálculo del área del círculo
public function getArea()
{
    return pi() * pow($this->radio, 2);
}

// Método para cálculo de la circunferencia del círculo
public function getCircunferencia()
{
    return 2 * pi() * $this->radio;
}
}

// Instanciando tres objetos de la clase Circulo
$circulo1 = new Circulo(4);
$circulo2 = new Circulo(6);
$circulo3 = new Circulo(8);

// Mostrando en pantalla los cálculos para cada instancia
echo 'Área del círculo 1: ' . $circulo1->getArea() . ' Circunferencia: ' . $circulo1->
getCircunferencia() . '<br>';
echo 'Área del círculo 2: ' . $circulo2->getArea() . ' Circunferencia: ' . $circulo2->
getCircunferencia() . '<br>';
echo 'Área del círculo 3: ' . $circulo3->getArea() . ' Circunferencia: ' . $circulo3->
getCircunferencia() . '<br>';

```

Ejercicio 5. Clase Estudiante.

Crema una clase llamada Estudiante con dos propiedades/atributos denominados nombre y notas (array/lista), con constructor y con métodos que obtenga el nombre, modifique el nombre, obtenga las notas, modifique las notas y, por último, que obtenga la media de esas notas y las muestre. Crema 3 instancias/objetos de la clase Estudiante y ejecuta en ellos el método creado.

```

<?php

/*
EJERCICIO 5. Alumno Pedro Godoy Polaina
Crema una clase llamada Estudiante con dos propiedades/atributos denominados
nombre y notas (array/lista), con constructor y con métodos que obtenga el nombre,
modifique el nombre, obtenga las notas, modifique las notas y, por último, que
obtenga la media de esas notas y las muestre. Crema 3 instancias/objetos de la
clase Estudiante y ejecuta en ellos el método creado..
*/

// Creación de la Clase Estudiante
class Estudiante
{
    private $nombre;
    private $notas;

    // Método constructor con dos parámetros de entrada
    public function __construct($nombre, $notas)
    {
        $this->nombre = $nombre;
        $this->notas = $notas;
    }

    // Método GET para el nombre
    public function getNombre()
    {
        return $this->nombre;
    }

    // Método SET para modificar el nombre
    public function setNombre($nombre)
    {
        $this->nombre = $nombre;
    }
}

```

```

    }

    // Método GET para obtener las notas
    public function getNotas()
    {
        return $this->notas;
    }

    // Método SET para modificar las notas
    public function setNotas($notas)
    {
        $this->notas = $notas;
    }

    // Método que obtendrá la media de las notas
    public function getMediaNotas()
    {
        $sumaNotas = 0;
        foreach ($this->notas as $nota) {
            $sumaNotas += $nota;
        }
        return $sumaNotas / count($this->notas);
    }
}

// Instanciando 3 objetos de la clase Estudiante
$estudiante1 = new Estudiante('Pedro Godoy', array(8, 9, 7));
$estudiante2 = new Estudiante('María Sánchez', array(7, 5, 10));
$estudiante3 = new Estudiante('Pablo López', array(9, 8, 5));

// Imprimimos el nombre y la nota media de cada estudiante
echo 'El nombre del estudiante 1 es: ' . $estudiante1->getNombre() . ' y su nota media es: ' .
$estudiante1->getMediaNotas() . '<br>';
echo 'El nombre del estudiante 2 es: ' . $estudiante2->getNombre() . ' y su nota media es: ' .
$estudiante2->getMediaNotas() . '<br>';
echo 'El nombre del estudiante 3 es: ' . $estudiante3->getNombre() . ' y su nota media es: ' .
$estudiante3->getMediaNotas() . '<br>';

```

Ejercicio 6. Cálculo de distancia euclídea de dos puntos

Crea una función que reciba dos parámetros de entrada de tipo clase Punto (realizado en ejercicio 01) y que devuelva la distancia euclídea entre esos dos puntos. Ejecuta 3 llamadas de ejemplo de la función creada.

```

<?php
/*
EJERCICIO 6. Alumno Pedro Godoy Polaina
Crea una función que reciba dos parámetros de entrada de tipo clase Punto
(realizado en ejercicio 01) y que devuelva la distancia euclídea entre esos
dos puntos. Ejecuta 3 llamadas de ejemplo de la función creada.
*/

// Creación de la clase Punto
class Punto
{
    // Atributos privados
    private $x;
    private $y;

    // Método constructor con dos parámetros
    public function __construct($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    // Implementación métodos Get
    public function getX()
    {
        return $this->x;
    }
    public function getY()
    {
        return $this->y;
    }
}

```

```

// Implementación métodos Set
public function setX($x)
{
    $this->x = $x;
}
public function setY($y)
{
    $this->y = $y;
}

// Función que realiza el cálculo de la distancia euclídea entre dos puntos
// Lo hacemos public static para que pueda ser llamado sin tener que crear instancia
// de la función
public static function getDistanciaEuclídea(Punto $a, Punto $b)
{
    return sqrt(pow($b->getX() - $a->getX(), 2) + pow($b->getY() - $a->getY(), 2));
}

}

// Instanciando tres objetos
$punto1 = new Punto(10, 24);
$punto2 = new Punto(1, 12);
$punto3 = new Punto(8, 14);

// Calculamos la distancia euclídea entre los puntos. La llamada se hace anteponiendo
// por ser un método static
echo 'Distancia euclídea entre Punto 1 y Punto 2 = '.Punto::getDistanciaEuclídea($punto1,
$punto2).'\n';
echo 'Distancia euclídea entre Punto 2 y Punto 3 = '.Punto::getDistanciaEuclídea($punto2,
$punto3).'\n';
echo 'Distancia euclídea entre Punto 1 y Punto 3 = '.Punto::getDistanciaEuclídea($punto1,
$punto3).'\n';

```

Ejercicio 7. Clase línea Linea2D

Creas una clase llamada Linea2D con dos propiedades/atributos denominados p1 y p2 de tipo clase Punto (realizado en ejercicio 01) y con dos métodos, uno que obtenga el punto medio del segmento y otro que obtenga la distancia euclídea, ambos usando dichas propiedades/atributos. Creas 3 instancias/objetos de la clase Linea2D y ejecutas en ellos los dos métodos creados.

Implementación de la clase Linea2D

```

<?php
/*
EJERCICIO 7. Alumno Pedro Godoy Polaina
Crea una clase llamada Linea2D con dos propiedades/atributos denominados p1 y p2
de tipo clase Punto (realizado en ejercicio 01) y con dos métodos, uno que
obtenga el punto medio del segmento y otro que obtenga la distancia euclídea,
ambos usando dichas propiedades/atributos. Creas 3 instancias/objetos de la clase
Linea2D y ejecutas en ellos los dos métodos creados.
*/

// Importamos la clase Punto
require_once 'Punto.php';

// Creación de la clase Linea2D
class Linea2D
{
    // Atributos privados de la clase Linea2D, que son dos puntos
    private $p1;
    private $p2;

    // Constructor de la clase Linea2D, que recibe dos parámetros que son objetos Punto
    public function __construct(Punto $p1, Punto $p2)
    {
        $this->p1 = $p1;
        $this->p2 = $p2;
    }
}

```

```

// Devuelve el punto medio de la línea en forma de objeto de la clase Punto
public function getPuntoMedio()
{
    $x = ($this->p1->getX() + $this->p2->getX()) / 2;
    $y = ($this->p1->getY() + $this->p2->getY()) / 2;
    // Con los parámetros creo un objeto Punto que será el punto medio.
    $pMedio = new Punto($x, $y);
    // Devuelvo el objeto creado
    return $pMedio;
}

// Función que devuelve la distancia euclídea entre los dos puntos de la línea
public function getDistanciaEuclídea()
{
    return Punto::getDistanciaEuclídea($this->p1, $this->p2);
}
}

// Creación de tres objetos Linea2D con diferentes puntos
$linea1 = new Linea2D(new Punto(-1, 7), new Punto(3, 6));
$linea2 = new Linea2D(new Punto(1, 5), new Punto(5, 9));
$linea3 = new Linea2D(new Punto(0, 0), new Punto(3, 0));

// Ejecución de las funciones y mostrado en pantalla
echo "RESULTADO DE LAS OPERACIONES:" . "<br>";
echo "Punto medio de línea 1: (" . $linea1->getPuntoMedio()->getX() . ", " . $linea1->getPuntoMedio()->getY() . ")<br>";
echo "Distancia euclídea de línea 1: " . $linea1->getDistanciaEuclídea() . "<br>";
echo "-----" . "<br>";
echo "Punto medio de línea 2: (" . $linea2->getPuntoMedio()->getX() . ", " . $linea2->getPuntoMedio()->getY() . ")<br>";
echo "Distancia euclídea de línea 2: " . $linea2->getDistanciaEuclídea() . "<br>";
echo "-----" . "<br>";
echo "Punto medio de línea 3: (" . $linea3->getPuntoMedio()->getX() . ", " . $linea3->getPuntoMedio()->getY() . ")<br>";
echo "Distancia euclídea de línea 3: " . $linea3->getDistanciaEuclídea() . "<br>";

```

Para este ejercicio **uso la clase Punto** que será incluida en la clase Linea2D mediante: *require_once 'Punto.php';*

```

<?php

// Creación de la clase Punto. Esta clase fue creada en ejercicios anteriores y no ha sufrido cambio
// será importada a para la utilización de la clase Linea2D

class Punto
{
    // Atributos privados
    private $x;
    private $y;

    // Método constructor con dos parámetros
    public function __construct($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    // Implementación métodos Get
    public function getX()
    {
        return $this->x;
    }

    public function getY()
    {
        return $this->y;
    }

    // Implementación métodos Set
    public function setX($x)
    {
        $this->x = $x;
    }
}

```

```

public function setY($y)
{
    $this->y = $y;
}

// Función que realiza el cálculo de la distancia euclídea entre dos puntos
// Lo hacemos public static para que pueda ser llamado sin tener que crear instancia
// de la función
public static function getDistanciaEuclídea(Punto $a, Punto $b)
{
    return sqrt(pow($b->getX() - $a->getX(), 2) + pow($b->getY() - $a->getY(), 2));
}
}

```

Ejercicio 8. Clase Forma

Crea una clase llamada Forma con una propiedad/atributo denominada centro de tipo clase Punto y un método que se llame area y que devuelva un número, por ejemplo 0. A continuación, crea dos clases llamadas Rectangulo y Circulo (realizados en ejercicios 03 y 04) que hereden de la clase Forma ya creada. Crea 3 instancias/objetos de las clases Rectangulo, Circulo, de la clase que hereda Forma y ejecuta sus métodos.

Creemos la clase padre Forma de la que heredarán la clase Circulo y Rectangulo

```

<?php

/*
EJERCICIO 8. Alumno Pedro Godoy Polaina
Crea una clase llamada Forma con una propiedad/atributo denominada centro de tipo
clase Punto y un método que se llame area y que devuelva un número, por ejemplo 0.
A continuación, crea dos clases llamadas Rectangulo y Circulo (realizados en ejercicios 03 y 04)
que hereden de la clase Forma ya creada. Crea 3 instancias/objetos de las clases Rectangulo,
Circulo, de la clase que hereda Forma y ejecuta sus métodos.
*/

// Insertamos la clase Punto, Rectangulo y Circulo
require_once 'Punto.php';
require_once 'Circulo.php';
require_once 'Rectangulo.php';

// Creación de la clase Forma que es clase padre de Circulo y Rectangulo
class Forma
{
    protected $centro;

    public function __construct(Punto $centro)
    {
        $this->centro = $centro;
    }

    public function area()
    {
        return 0; // no se usa, pero se pide para el ejercicio
    }
}

// Crear un objeto Punto para el centro que usaremos en la clase Forma
$centro = new Punto(10, 10);

// Crear objetos Rectangulo y Circulo
$rectangulo = new Rectangulo($centro, 8, 12);
$circulo = new Circulo($centro, 5);

// Mostrar resultados con el área de cada una de las formas
echo "RESULTADOS DE LAS ÁREAS DE LAS DISTINTAS FORMAS" . "<br>";
echo "Área del rectángulo: " . $rectangulo->getArea() . "<br>";
echo "Área del círculo: " . $circulo->getArea() . "<br>";

```


Usamos la clase Punto que habíamos creado anteriormente

```
<?php

class Punto
{
    // Atributos privados
    private $x;
    private $y;

    // Método constructor con dos parámetros
    public function __construct($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    // Implementación métodos Get
    public function getX()
    {
        return $this->x;
    }

    public function getY()
    {
        return $this->y;
    }

    // Implementación métodos Set
    public function setX($x)
    {
        $this->x = $x;
    }

    public function setY($y)
    {
        $this->y = $y;
    }

    // Función que realiza el cálculo de la distancia euclídea entre dos puntos
    // Lo hacemos public static para que pueda ser llamado sin tener que crear instancia
    // de la función
    public static function getDistanciaEuclídea(Punto $a, Punto $b)
    {
        return sqrt(pow($b->getX() - $a->getX(), 2) + pow($b->getY() - $a->getY(), 2));
    }
}
```

Creamos la nueva clase Circulo que hereda de Forma

```
<?php

// Importamos la clase Forma y Punto
require_once 'Forma.php';
require_once 'Punto.php';

// La clase Circulo hereda de la clase Forma
class Circulo extends Forma
{
    // Atributos privados
    private $radio;

    // Método constructor con dos parámetros de entrada
    public function __construct(Punto $centro, $radio)
    {
        parent::__construct($centro);
        $this->radio = $radio;
    }
}
```

```

// Método para cálculo del área del círculo
public function getArea()
{
    return pi() * pow($this->radio, 2);
}

// Método para cálculo de la circunferencia del círculo
public function getCircunferencia()
{
    return 2 * pi() * $this->radio;
}
}

```

Creamos la nueva clase Rectangulo que hereda de Forma

```

<?php

// Importamos la clase Forma y Punto
require_once 'Forma.php';
require_once 'Punto.php';

// Creación de la Clase Rectangulo que hereda de la clase Forma
class Rectangulo extends Forma
{
    // Atributos privados
    private $longitud;
    private $ancho;

    // Método constructor con tres parámetros de entrada
    public function __construct(Punto $centro, $longitud, $ancho)
    {
        parent::__construct($centro);
        $this->longitud = $longitud;
        $this->ancho = $ancho;
    }

    // Método que calculará el área del rectángulo
    public function getArea()
    {
        return $this->longitud * $this->ancho;
    }
}

```

Ejercicio 9. Clase Estudiante ampliación de funciones

crea una función que reciba un parámetro de entrada de tipo array/lista, con identificador "grupos", de tamaño 3 cuyos elementos sean de tipo array/lista de clase Estudiante (realizado en ejercicio 05). La función tiene que devolver el índice del array/lista "grupos" cuyo promedio de notas del grupo de estudiantes sea el más alto. Ejecuta 1 llamada de ejemplo de la función creada.

Implementación de la función promedioMayor() que usará la clase Estudiante

```

<?php

/*
EJERCICIO 9. Alumno Pedro Godoy Polaina
crea una función que reciba un parámetro de entrada de tipo array/lista, con identificador
"grupos", de tamaño 3 cuyos elementos sean de tipo array/lista de clase Estudiante
(realizado en ejercicio 05). La función tiene que devolver el índice del array/lista
"grupos" cuyo promedio de notas del grupo de estudiantes sea el más alto.
Ejecuta 1 llamada de ejemplo de la función creada.

```

```

*/

// Importamos la clase Estudiante
require_once 'Estudiante.php';

function promedioMayor($grupos)
{
    $promedioMasAlto = null;
    $indiceMasAlto = null;

    // Iterador de estudiantes
    for ($i = 0; $i < count($grupos); $i++) {
        $promedioGrupo = 0;
        $estudiantes = $grupos[$i];

        // Iterador de grupos
        for ($j = 0; $j < count($estudiantes); $j++) {
            $promedioGrupo += $estudiantes[$j]->getMediaNotas();
        }

        // cálculo del promedio del grupo
        $promedioGrupo /= count($estudiantes);

        // Guarda el índice más alto encontrado
        if ($promedioGrupo > $promedioMasAlto) {
            $promedioMasAlto = $promedioGrupo;
            $indiceMasAlto = $i;
        }
    }

    // Devuelve el índice mayor hallado
    return $indiceMasAlto;
}

// Introducción de datos en array de grupos mediante la instanciación de estudiantes
$grupoA = [
    new Estudiante("Estudiante 1", [5, 3, 9]),
    new Estudiante("Estudiante 2", [2, 9, 6]),
    new Estudiante("Estudiante 3", [4, 0, 6]),
];
$grupoB = [
    new Estudiante("Estudiante 4", [8, 9, 7]), // pongo aquí las mejores notas
    new Estudiante("Estudiante 5", [9, 10, 8]),
    new Estudiante("Estudiante 6", [5, 9, 10]),
];
$grupoC = [
    new Estudiante("Estudiante 7", [3, 4, 5]),
    new Estudiante("Estudiante 8", [4, 5, 6]),
    new Estudiante("Estudiante 9", [6, 5, 7]),
];

// carga del array grupos con los arrays de estudiantes
$grupos = [$grupoA, $grupoB, $grupoC];

// El índice se calcula sumando 1 a la posición en el array
$indice = promedioMayor($grupos);
echo "El mejor promedio de notas lo tiene el grupo cuyo índice es: " . ($indice + 1) . "\n";

```

Clase Estudiante usada para este ejercicio y que fue creada en un ejercicio anterior

```

<?php

// Clase Estudiante creada para ejercicio anterior
class Estudiante
{
    private $nombre;
    private $notas;

    // Método constructor con dos parámetros de entrada
    public function __construct($nombre, $notas)
    {

```

```

        $this->nombre = $nombre;
        $this->notas = $notas;
    }

    // Método GET para el nombre
    public function getNombre()
    {
        return $this->nombre;
    }

    // Método SET para modificar el nombre
    public function setNombre($nombre)
    {
        $this->nombre = $nombre;
    }

    // Método GET para obtener las notas
    public function getNotas()
    {
        return $this->notas;
    }

    // Método SET para modificar las notas
    public function setNotas($notas)
    {
        $this->notas = $notas;
    }

    // Método que obtendrá la media de las notas
    public function getMediaNotas()
    {
        $sumaNotas = 0;
        foreach ($this->notas as $nota) {
            $sumaNotas += $nota;
        }
        return $sumaNotas / count($this->notas);
    }
}

```

Ejercicio 10. Clases A, B, C. Herencia

Crea una clase C que herede de una clase B y que la clase B herede de A. La clase C heredará todos los métodos y atributos de B y B de A. Como mínimo, una función de A, de B y de C tienen que tener el mismo nombre pero que hagan cosas distintas. Crea 3 instancias/objetos de las clases A, B y C y ejecuta todos los métodos que hayas creado.

Voy a crear tres clases tal y como se pide en el ejercicio que representarán: la clase A serán seres vivos, la clase B serán invertebrados, la clase C serán insectos. Les pondré un método a cada una y un atributo como ejemplo. Cada clase tendrá además un método con el mismo nombre, para agilizar pongo un texto fijo que no dependerá de ningún parámetro.

Luego podremos crear objetos de las clases

archivo index.php que ejecuta el ejercicios

```

<?php
/*
EJERCICIO 10. Alumno Pedro Godoy Polaina
Crea una clase C que herede de una clase B y que la clase B herede de A.
La clase C heredará todos los métodos y atributos de B y B de A. Como mínimo,
una función de A, de B y de C tienen que tener el mismo nombre pero que hagan
cosas distintas. Crea 3 instancias/objetos de las clases A, B y C y
ejecuta todos los métodos que hayas creado.

```

```

*/

// Importamos la clase A y B y C
require_once 'A.php';
require_once 'B.php';
require_once 'C.php';

// Creando los objetos
$a = new A("Seres vivos");
$b = new B("Invertebrados", 6);
$c = new C("Insectos", 6, "tacto y vibración");

// Ejecutando todos los métodos
$a->respirar(); // Seres vivos respiran.
$a->reproducirse(); // Seres vivos -asexual.

$b->respirar(); // Invertebrados está respirando.
$b->andar(); // Invertebrados se está moviendo con 6 patas.
$b->reproducirse(); // Invertebrados resproduccion mediante huevos.

$c->respirar(); // Insectos respiran.
$c->andar(); // Insectos andan con con 6 patas.
$c->interactuar(); // Insectos se está comunicando con sus antenas.
$c->reproducirse(); // Insectos resproduccion por fecundación.

```

Implementando la clase A

```

<?php
/* EJERCICIO 10. Alumno Pedro Godoy Polaina*/

// Definición de la clase A
// Se refiere a seres vivos, pero no se implementa nada que la función contrucctora y la función
respirar
class A
{
    protected $nombre;

    public function __construct($nombre)
    {
        $this->nombre = $nombre;
    }

    public function respirar()
    {
        echo $this->nombre . " está respirando.<br>";
    }

    public function reproducirse()
    {
        echo $this->nombre . " se reproduce de manera sexual.<br>";
    }
}

```

Implementando la clase B

```

<?php
/* EJERCICIO 10. Alumno Pedro Godoy Polaina*/
// Importamos la clase A
require_once 'A.php';

// Definición de la clase B
// Se refiere a invertebrados, pero solo se implementa la función contrucctora y la función andar
class B extends A
{
    protected $num_patas;

    public function __construct($nombre, $num_patas)
    {

```

```

        parent::__construct($nombre);
        $this->num_patas = $num_patas;
    }

    public function andar()
    {
        echo $this->nombre . " anda con " . $this->num_patas . " patas.<br>";
    }

    public function reproducirse()
    {
        echo $this->nombre . " se reproduce por huevos.<br>";
    }
}

```

Implementando la clase C

```

<?php
/* EJERCICIO 10. Alumno Pedro Godoy Polaina*/
// Importamos la clase A y B
require_once 'A.php';
require_once 'B.php';

// Definición de la clase C
// Se refiere a insectos, pero solo se implementa la función constructora y la función interactuar
class C extends B
{
    protected $tipo_antena;

    public function __construct($nombre, $num_patas, $tipo_antena)
    {
        parent::__construct($nombre, $num_patas);
        $this->tipo_antena = $tipo_antena;
    }

    public function interactuar()
    {
        echo $this->nombre . " interactua con sus antenas mediante " . $this->tipo_antena . ".<br>";
    }

    public function reproducirse()
    {
        echo $this->nombre . " se reproduce por fecundación.<br>";
    }
}

```

4. Explica lo que es patrón de arquitectura y enumera todos sus ejemplos.

Intento comprender el concepto, por ello cuando habla de patrón de arquitectura del software estamos hablando de las distintas formas en las que se pueden diseñar los sistemas más o menos complejos de software. Estas formas distintas deberían ser elegidas atendiendo a la mejor funcionalidad deseada y la mejor resolución de problemas teniendo en cuenta el equipo en que se usará, la seguridad, la escalabilidad, la eficiencia, etc.

Entonces habrá que valorar diferentes arquitecturas, antes de aplicarlas a nuestro diseño.

Un patrón arquitectónico no es una arquitectura. Un patrón arquitectónico es un concepto ya probado y comprobado, que resuelve y delinea algunos elementos de una arquitectura de

software. Por ello muchas arquitecturas podría implementar patrones iguales para resolver problemas. Diremos entonces que un patrón de arquitectura de software es una solución general y reutilizable para un problema común en el diseño de sistemas de software.

Algunos ejemplos de patrones de arquitectura de software incluyen el patrón Modelo-Vista-Controlador (MVC), el patrón de arquitectura cliente-servidor, el patrón de arquitectura basado en eventos, etc. Que ya sabemos que ayudan a la escalabilidad de los programas y ayudan a los desarrolladores a estructurar, planificar y repartir su trabajo.

Ejemplos de patrones de arquitecturas. Se relaciona una selección

- Modelo-vista-controlador
- IAE (Integración de aplicaciones empresariales)
- Centro de datos maestros
- Almacén de datos operativos (ODS)
- mercado de datos
- Almacén de datos
- SOA (Arquitectura orientada a servicios)
- Patrón de capas
- Patrón cliente-servidor
- Patrón maestro-esclavo
- Patrón de filtro de tubería
- Patrón de intermediario
- Patrón de igual a igual
- Patrón de pizarra
- Patrón de intérprete
- patrón de corredor
- Arquitectura impulsada por eventos
- Invocación implícita
- arquitectura hexagonal
- microservicios
- Presentación-abstracción-control
- Modelo-vista-presentador
- Modelo-vista-modelo de vista
- Modelo-vista-adaptador
- Sistema de componentes de entidad
- Entidad-control-límite
- Arquitectura de varios niveles (a menudo de tres niveles o de n niveles)
- Programación orientada a objetos
- Objetos desnudos
- De igual a igual
- Arquitectura de tuberías y filtros

- Arquitectura orientada a Servicios
- Arquitectura basada en el espacio
- tabla hash distribuida
- Patrón de publicación-suscripción
- Agente de mensajes
- Modelo jerárquico-vista-controlador

5. Define programación por capas y sus capas como BLL (lógica de negocio) o DAL y detalla la three-tier.

La programación por capas es un enfoque de diseño de software que consiste en estructurar un sistema en módulos independientes, organizados en niveles y por responsabilidades. Cada capa es responsable de una tarea específica y se comunica con las capas adyacentes a través de interfaces.

Arquitectura Three-tier: Un típico patrón de arquitectura por capas divide se divide la aplicación en tres capas:

- **Capa de presentación:** o capa de interfaz de usuario. Es la capa que interactúa con el usuario, a través de una interfaz gráfica o de línea de comandos.
- **Capa de lógica de negocio:** Se encarga de procesar los datos recibidos de la capa de presentación, aplicar las reglas de negocio y generar la salida correspondiente.
- **Capa de acceso a datos:** Es la capa que se encarga de interactuar con la base de datos o cualquier otro sistema de almacenamiento de datos.

También son conocidas estas capas como: de usuario, negocio, de persistencia, de servicios, de infraestructura.

Ampliando conceptos

La capa BLL:

La capa BLL (Business Logic Layer) o de lógica de negocio es una capa esencial de la arquitectura de aplicación que se encarga de implementar las reglas de negocio y la lógica de procesamiento de datos. Es decir, es la capa que se encarga de realizar las operaciones que son específicas de la aplicación y que no están relacionadas con la presentación de datos o el acceso a la base de datos. Procesar la información recibida desde la capa de presentación, aplicar las reglas de negocio y generar la salida correspondiente.

La capa DAL

La capa DAL (Data Access Layer) o de acceso a datos, es una capa de la arquitectura de una aplicación que se encarga de interactuar con la base de datos o cualquier otro sistema de almacenamiento de datos. Hace las operaciones de lectura o escritura en la base de datos y devuelve los resultados a la capa BLL.

Nivel de presentación

Este es el nivel más alto de la aplicación. El nivel de presentación muestra información relacionada con servicios tales como búsqueda de productos, compras y contenido del carrito de

compras. Se comunica con otros niveles mediante los cuales envía los resultados al nivel del navegador/cliente ya todos los demás niveles de la red. En términos simples, es una capa a la que los usuarios pueden acceder directamente (como una página web o la GUI de un sistema operativo).

6. Explica el patrón de arquitectura MVC, sus componentes y cómo interactúan.

El patrón de arquitectura MVC (Modelo Vista Controlador) es un patrón de arquitectura de software que divide la lógica de la aplicación en tres componentes principales: el modelo, la vista y el controlador. Controlador, Vista y Modelo son cada una de las divisiones que se responsabilizan de cometidos como la lógica de negocio, la interfaz de usuario o el acceso a los datos, interactuando entre sí para procesar las solicitudes del usuario y mostrar la salida.

Ampliando los conceptos:

- **Modelo (Model):** Es la capa que representa los datos y la lógica de negocio de la aplicación. Esta capa se encarga de la gestión de los datos y de implementar las operaciones y reglas de negocio necesarias.
- **Vista (View):** Es la capa que se encarga de la presentación de la información al usuario. Esta capa muestra los datos al usuario de manera visual y proporciona una interfaz de usuario interactiva para que el usuario pueda interactuar con la aplicación.
- **Controlador (Controller):** Es la capa que se encarga de recibir las acciones del usuario y coordinar la interacción entre el modelo y la vista. Esta capa recibe las entradas del usuario y se encarga de actualizar el modelo o la vista según corresponda.

Como interactúan entre sí los componentes:

El usuario realiza una acción en la vista, el controlador recibe la entrada y actualiza el modelo correspondiente. El modelo procesa la entrada y actualiza su estado interno y seguidamente notifica al controlador de los cambios realizados. El controlador actualiza la vista correspondiente con los cambios en el modelo. La vista se actualiza para mostrar los cambios al usuario. Este ciclo se repite cada vez que el usuario realiza una acción en la vista, lo que permite una interacción fluida y coherente entre los diferentes componentes de la arquitectura.

7. Resumen de vídeo explicativo de MVC. Explica el flujo de MVC empezando por la vista.

Ya he explicado lo que es MVC (Modelo Vista Controlador), como un patrón de diseño de software, como una forma de organizar la lógica de nuestra aplicación.

No está constreñido a ningún lenguaje concreto ya que puede ser utilizado este patrón en muchos ámbitos de trabajo. De hecho es tan potente el concepto que se puede decir que es el "más utilizado" dentro del mundo de la web.

El objetivo principal de este patrón MVC es la separación de responsabilidades.

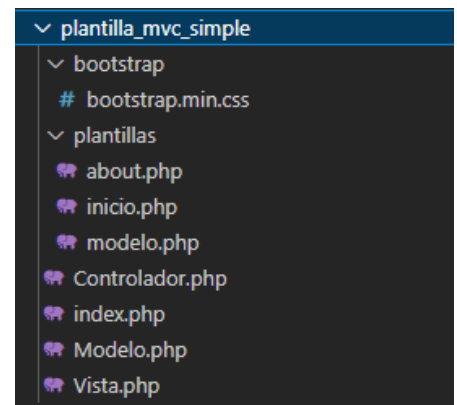
Un flujo de MVC como ejemplo práctico podría ser:

1. Un usuario usando el navegador web entra en un sitio web.
2. La petición del usuario "request" viaja hasta el controlador que funciona como el cerebro de la aplicación.
3. El controlador pide la información solicitada por el usuario al modelo, por ejemplo para autenticar.
4. El modelo devuelve al controlador la información del usuario.
5. y éste a su vez le entrega los datos necesarios al usuario o bien gestiona la autenticación.
6. Si el proceso se repite por ejemplo esta vez desde otro navegador o desde un teléfono móvil, el proceso no cambia desde la perspectiva del controlador o el modelo, ya que solo será necesario una vista distinta sin que esto influya en el resto de las capas.

8. Plantilla MVC básica. Explicación. Modificaciones.

Ya hemos visto los conceptos que rodean al patrón de diseño MVC (Modelo Vista Controlador) y ahora voy a estudiar una plantilla que sigue este patrón para aprender más sobre esto.

La plantilla entregada presenta una serie de archivos separados en los que tenemos principalmente un index.php como página de inicio y tres archivos más uno para Modelo, otro para Vista y otro para Controlador. Además la plantilla contiene dos carpetas en una encontramos el diseño CSS realizado con bootstrap y en otra carpeta llamada plantillas hay la implementación de tres páginas web con extensión php que serán las que nuestro menú situado en la página de inicio nos presentará cuando seleccionemos una u otra.



Index.php

Este archivo escrito íntegramente en php, está usando el patrón de arquitectura MVC para procesar la solicitud del usuario. Primero, se crean instancias de los componentes Modelo, Vista y Controlador. Luego, se comprueba si hay petición de acción en la solicitud del usuario. Si hay una acción, el controlador la ejecuta. Si no hay ninguna acción, se muestra la vista de inicio.

Vista.php

Este archivo escrito también íntegramente en php, define una clase Vista que se usa para mostrar la interfaz de usuario. La clase contiene tres métodos: inicio(), modelo() y about(). El método inicio() carga la plantilla de inicio, el método modelo() carga la plantilla de modelo y el método about() carga la plantilla about. El método constructor sirve para inicializar el modelo.

Controlador.php

Este archivo contiene el código en php que define una clase "Controlador" que se usa para procesar las solicitudes del usuario. La clase contiene tres métodos: `modificar_modelo()`, `mostrar_modelo()` y `mostrar_about()`.

- El método constructor se usa para inicializar el modelo y la vista.
- El método `modificar_modelo()` modifica el modelo y luego carga la vista de modelo.
- El método `mostrar_modelo()` carga la vista de modelo sin modificar el modelo.
- El método `mostrar_about()` carga la vista about.

Modelo.php

Este archivo contiene el código en php que define una clase Modelo que se usa para almacenar los datos de la aplicación. La clase contiene dos métodos: `get_x()` y `set_x()` y el constructor:

- El método `__construct()` se usa para inicializar el valor de x.
- El método `get_x()` devuelve el valor de x.
- El método `set_x()` establece el valor de x.

Otros archivos de la plantilla

La plantilla presenta inicialmente tres archivos con extensión php, programados con HTML5 y que representan las tres páginas web a las que nos dirige nuestro menú principal.

Las tres páginas presentan una copia exacta de un menú que se repite en cada página.

Esas páginas están confeccionadas a modo de ejemplo y el estilo de página se asigna en un link a un archivo con extensión css y programado con bootstrap.

Modificación de la plantilla

Tras estudiar la plantilla y el proceso y funcionamiento que sigue cada una de las partes con las que se crea el sitio web, he realizado una modificación de la plantilla que me permite comprender mejor la forma en la que está programada esta plantilla.

- He creado una página web básica que llamaré "fotografia.php", similar a las tres anteriores que están contenidas en la carpeta "plantillas" y que me permitirá enlazarla con un nuevo botón en el menú.
- He creado un nuevo botón en el menú que tiene el texto "Mostrar fotografía" y cuyo enlace asociado al botón nos lleva a `index.php` con la acción=`mostrar_foto`. Este nuevo botón del menú ha sido insertado en las tres páginas previas y en la nueva página que he creado.

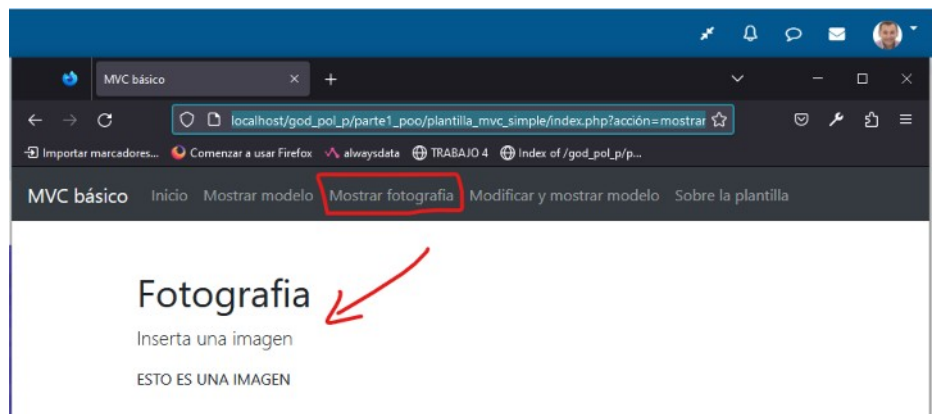
```
<li class='nav-item'>
  <a class='nav-link' href='index.php?acción=mostrar_foto'>Mostrar fotografia</a>
</li>
```

- En el archivo `vista.php` nos encontramos la definición de una clase llamada Vista, a la que añadido un nuevo método público llamado "foto". Ese archivo con la clase "Vista" tiene un atributo privado llamado "modelo" que se inicializa en su propio constructor. Tiene además cuatro métodos públicos, de los que solo uno precisa acceder al "modelo" que es precisamente el método público "modelo".
- El método que he insertado nuevo llamado "foto" requiere el archivo de plantilla

"fotografia.php" y no necesita acceder al modelo.

- En el archivo "controlador.php" define una clase llamada "Controlador" según el patrón MVC. Esta clase tiene dos atributos privados: "modelo", "vista", cuatro métodos públicos además de un constructor con dos argumentos: un objeto del modelo y un objeto de la vista, y los asigna a las propiedades correspondientes.
- He creado un nuevo método público llamado "mostrar_foto()": Este método llama al método "foto()" del objeto de la vista, que a su vez se encargará de gestionar el enlace a la página: 'plantillas/fotografia.php'
- El archivo modelo.php que define una clase llamada "Modelo" no lo he modificado.

El resultado de la modificación es un nuevo enlace a una página que sigue el mismo patrón MVC:



9. Explica al detalle lo que es un framework web.

En los framework web por lo tanto podemos tener una serie de herramientas y bibliotecas que permiten a los desarrolladores crear aplicaciones web de manera más rápida y eficiente. Un framework web, o marco de trabajo web, nos proporciona una forma estándar de crear e implementar aplicaciones web. El objetivo de los framework web como estructuras de software es automatizar los procesos generales asociados con las actividades comunes realizadas en el desarrollo web. Lo pueden hacer por ejemplo proporcionándonos bibliotecas para acceder a bases de datos, plantillas, gestión de sesiones, y reutilización de código que simplificará el desarrollo de sitios web.

En un framework web nos podemos encontrar las siguientes características:

- Un sistema de enrutamiento que permite a los desarrolladores definir las rutas de URL para su aplicación web.
- Un sistema de plantillas que permite definir la estructura y el diseño de las páginas web de una aplicación.
- Un sistema de gestión de bases de datos que permite trabajar con bases de datos y realizar operaciones CRUD (crear, leer, actualizar y eliminar).
- Un conjunto de bibliotecas que proporcionan una funcionalidad común, como la autenticación de usuarios, la validación de formularios, la generación de gráficos y la manipulación de archivos.

Podemos encontrar frameworks web de código abierto o propietarios y muchos de ellos siguen el patrón de arquitectura MVC Modelo-Vista-Controlador. Además pueden estar realizados para diferentes lenguajes de programación como podría ser PHP.

Uno de los ejemplos de frameworks que estoy estudiando es Laravel, si bien hay muchos

más.

Los primeros frameworks web tal y como hoy los podemos considerar “maduros” aparecen a finales de la década de los 90 y nos proporcionan muchas bibliotecas útiles listas para ser usadas para el desarrollo web. Entre ellos tenemos algunos muy conocidos como: ASP.NET , Java EE , Laravel , Django, web2py , OpenACS, Catalyst, Ruby on Rails, etc.

Existen otras arquitecturas usadas en los frameworks web, distantes a MVC, como podrían ser los basados en acciones: Push-based vs. Pull-based, o los modelos de tres niveles: Three-tier organization.

10. Explica al detalle lo que es Laravel.

Laravel es un framework web, basado en PHP, gratuito y de código abierto, creado por Taylor Otwell en 2011 y destinado al desarrollo de aplicaciones web siguiendo el patrón arquitectónico modelo-vista-controlador (MVC). Laravel está basado en Symfony es otro framework web de PHP de código abierto que se basa en el patrón de arquitectura de software Modelo-Vista-Controlador.

Laravel es un sistema empaquetado y modular, con un administrador de dependencias dedicado, diferentes formas de acceder a bases de datos relacionales, utilidades que ayudan en la implementación y el mantenimiento de aplicaciones. Utiliza el motor de plantillas Blade para generar vistas.

La versión actual de Laravel es: Laravel 10 y se lanzó el 14 de febrero de 2023.

Alguna de las características y herramientas para el desarrollo de aplicaciones web que ofrece son:

- **Sistema de enrutamiento:** proporciona un sistema de enrutamiento potente y fácil de usar que permite definir las rutas de URL de manera clara y organizada.
- **Sistema de migraciones de bases de datos:** que permite a los desarrolladores definir y modificar la estructura de la base de datos de manera programática, lo que facilita la implementación y actualización de la base de datos.
- **Sistema de autenticación:** permite a los desarrolladores agregar fácilmente la autenticación de usuarios a su aplicación.
- **Sistema de correo electrónico:** tiene un sistema de correo electrónico integrado que permite a los desarrolladores enviar correos electrónicos de manera fácil y rápida.
- **Sistema de caché:** permite a los desarrolladores almacenar en caché los resultados de las consultas de base de datos y otros datos para mejorar el rendimiento de la aplicación.
- **Sistema de colas de trabajo:** permite procesar tareas en segundo plano, lo que puede mejorar significativamente la velocidad y la escalabilidad de la aplicación.
- Bibliotecas que se pueden utilizar para una variedad de tareas, como la integración con servicios de terceros, la manipulación de imágenes, la generación de PDF, etc.

Para saber más sobre Laravel

Laravel, desde la versión 4 utiliza **Composer** como administrador de dependencias para agregar paquetes PHP específicos de Laravel e independientes y que podemos encontrar en el repositorio de Packagist.

Eloquent ORM es una implementación avanzada de PHP y que presenta las tablas de la base de datos como clases, con sus instancias de objetos vinculadas a filas de una sola tabla.

El **generador de consultas**, es una alternativa de acceso a la base de datos más directa que Eloquent ORM. Es

El motor de plantillas **Blade** combina una o más plantillas con un modelo de datos para producir vistas.

Homestead: una máquina virtual de Vagrant que brinda a los desarrolladores de Laravel todas las herramientas necesarias para desarrollar Laravel directamente, incluidas Ubuntu , Gulp , Bower y otras herramientas de desarrollo que son útiles para desarrollar aplicaciones web a gran escala.

Lazy Collection: esta característica del marco PHP Laravel le permite principalmente manejar grandes cargas de datos, mientras mantiene bajo el uso de la memoria.

Cashier: proporciona una interfaz para administrar los servicios de facturación.

Envoy: proporciona una sintaxis mínima y limpia para definir tareas comunes que ejecuta en sus servidores remotos.

Socialite: proporciona mecanismos simplificados para la autenticación con diferentes proveedores de OAuth, incluidos Facebook, Twitter, Google, GitHub y Bitbucket.

Artisan: la interfaz de línea de comandos (CLI) de Laravel llamada Artisan, se introdujo inicialmente en Laravel 3 con un conjunto limitado de capacidades. La migración posterior de Laravel a una arquitectura basada en Composer permitió a Artisan incorporar diferentes componentes del marco Symfony , lo que resultó en la disponibilidad de funciones adicionales de Artisan en Laravel 4.

Hay muchos otros paquetes propios que no menciono, pero que presentan un marco muy completo de trabajo y que será necesario conocer con detalle en el futuro.

ACTIVIDADES DE LARAVEL

1. Framework web e instalación

- a) explica con tus propias palabras qué es un MVC con Routing.
- b) explica con tus propias palabras qué es un framework web.
- c) explica con tus propias palabras qué es Laravel.
- d) explica todas las maneras de instalar Laravel.
- e) Crea un proyecto en blanco en Laravel usando Composer ayudándote de su documentación oficial.

A) MVC con Routing

El término MVC hace referencia al patrón de arquitectura llamado MODELO-VISTA-CONTROLADOR. Este patrón de arquitectura de software divide la lógica de la aplicación en tres componentes principales: el modelo, la vista y el controlador, correspondiendo con cada una de las divisiones que se responsabilizan de cometidos como la lógica de negocio, la interfaz de usuario o el acceso a los datos, interactuando entre sí para procesar las solicitudes del usuario y mostrar la salida.

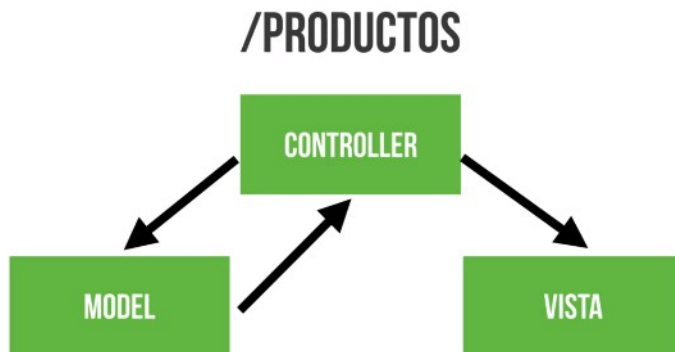
- **Vista (View):** Es la capa que se encarga de la presentación de la información al usuario. Esta capa muestra los datos al usuario de manera visual y proporciona una interfaz de usuario interactiva para que el usuario pueda interactuar con la aplicación.
- **Modelo (Model):** Es la capa que representa los datos y la lógica de negocio de la aplicación. Esta capa se encarga de la gestión de los datos y de implementar las operaciones y reglas de negocio necesarias.
- **Controlador (Controller):** Es la capa que se encarga de recibir las acciones del usuario y coordinar la interacción entre el modelo y la vista. Esta capa recibe las entradas del usuario y se encarga de actualizar el modelo o la vista según corresponda.

Resumiendo el funcionamiento de un patrón de diseño MVC podemos decir que la *vista* muestra el *modelo* al usuario, el usuario a su vez activa eventos para que la *vista* se los mande al *controlador* quien los recibe y manipula el *modelo* de acuerdo con la solicitud. El *modelo* a su vez realiza las *operaciones* necesarias para modificar la información y se lo notifica al *controlador* para que finalmente este muestre una nueva *vista* al usuario con la información del *modelo* actualizado.

MVC con Routing se refiere a la combinación del patrón de arquitectura Modelo-Vista-Controlador (MVC) con un sistema de enrutamiento. El sistema de enrutamiento se utiliza para definir las rutas de URL de una aplicación web y para determinar qué controlador y método de controlador se deben utilizar para manejar una solicitud en particular.

El *router* es el encargado de registrar todas las URL's que la aplicación soporta. Cada vez que el usuario accede a una parte concreta del sitio web es el router quien llama a un controlador que se comunica con el modelo para obtener los datos que son pasados hacia la vista para ser

mostrados.



Siguiendo el sistema de rutas, si accedemos a una de ellas el enrutador sabe qué controlador debe ejecutar y lo llama cuando se accede a esa ruta.

Luego el controlador se comunica con el modelo para solicitar datos, que nuevamente gestionará y la pasará a la vista para que se los muestre al usuario.

B) framework web

Los framework web nos proporcionan una serie de herramientas y bibliotecas que permiten a los desarrolladores crear aplicaciones web de manera más rápida y eficiente. Un framework web, o marco de trabajo web, nos proporciona una forma estándar de crear e implementar aplicaciones web.

El objetivo de los framework web como estructuras de software es automatizar los procesos generales asociados con las actividades comunes realizadas en el desarrollo web. Lo pueden hacer por ejemplo proporcionándonos bibliotecas para acceder a bases de datos, plantillas, gestión de sesiones, y reutilización de código que simplificará el desarrollo de sitios web.

Los framework web nos ayudan a trabajar en equipo desarrollando entornos de trabajo comunes y con tareas diferenciadas, en los que aplicamos buenas prácticas para desarrollar nuestro código. Utilizan convenciones y estándares que son aplicados por los usuarios de ese framework web, y además nos proporcionan tareas ya programadas que sería laborioso programar nuevamente pero que al ser ampliamente usadas nos ahorran mucho tiempo, como podría ser la autenticación de usuarios, el trabajo CRUD con bases de datos, etc.

Los framework web nos ayudan también a hacer un eficiente mantenimiento del código, tanto por las personas que lo programaron inicialmente, como por los nuevos equipos de trabajo que se incorporen que verán como al aplicar unas buenas prácticas comunes de trabajo se les facilita la modificación posterior del código.

C) Laravel

Laravel es un framework web, basado en PHP, gratuito y de código abierto, creado en 2011 y destinado al desarrollo de aplicaciones web siguiendo el patrón arquitectónico modelo-vista-controlador (MVC). Es uno de los framework más utilizados en la actualidad.

Lo usaremos para hacer cualquier tipo de proyecto en PHP y nos evitará hacer código del que se conoce como "spaguetti" en el que cuando más grande se hace un proyecto menos comprensible resulta el código y su estructura.

Laravel es un sistema empaquetado y modular, con un administrador de dependencias dedicado, diferentes formas de acceder a bases de datos relacionales, utilidades que ayudan en la implementación y el mantenimiento de aplicaciones. Utiliza el motor de plantillas Blade para

generar vistas.

Blade: es un sistema de plantillas para crear las vistas en L  ravel que ser  n aprovechables en el c  digo.

Eloquent: es el sistema de L  ravel para tratar con las bases de datos. De esta forma se trabaja con programaci  n orientada a objetos en vez de directamente con la base de datos en su lenguaje.

Routing: es el propio sistema de rutas de Laravel, que gestionar   todas las rutas usadas en un sitio web.

Middlewares: son controladores que podemos usar para hacer tareas repetitivas como podr  an ser chequeo de permisos de usuarios, etc.

Laravel tiene una comunidad muy amplia de usuarios y su documentaci  n es muy completa.

D) Instalaci  n de Laravel 10

Antes de iniciar la instalaci  n de Laravel 10 como   ltima versi  n a fecha actual, debemos tener instalado en nuestra m  quina: PHP y Composer.

Nos es el prop  sito de este apartado describir estas instalaciones previas, pero a  adir   que podemos hacerlo a trav  s de los siguientes enlaces:

- Instalaci  n de XAMMP: <https://www.apachefriends.org/es/index.html>
- Instalaci  n de COMPOSSER: <https://getcomposer.org/download/>

En Windows 10 y tambi  n desde Ubuntu 22.04 que son los dos sistemas que uso habitualmente y usando la terminal de comandos en ambos sistemas, podemos comprobar las versiones instaladas de la siguiente forma:

```
php -version
composer --version
```

La instalaci  n de Laravel consiste en la creaci  n de un nuevo proyecto de Laravel a trav  s del comando de Composer: create-project.

Escribimos para ello en la terminal de comandos una vez que nos hemos posicionado en el director que contendr   el proyecto:

```
composer create-project laravel/laravel nombre-aplicacion
```

donde "nombre-aplicacion" ser   el nombre de nuestra aplicaci  n que a su vez nombrar   as   al directorio que la contiene.

Tambi  n Laravel dispone de un instalador propio, si bi  n primero habr   que instalarlo igualmente usando Composer:

```
composer global require laravel/installer
```

Una vez instalado el instalador escribimos el siguiente comando:

```
laravel new nombre-aplicacion
```

Laravel dispone de su propio servidor local que podemos iniciar de la siguiente forma:
nos posicionamos con la terminal de windows o de ubuntu en la carpeta del proyecto:

```
cd nombre-aplicacion
```

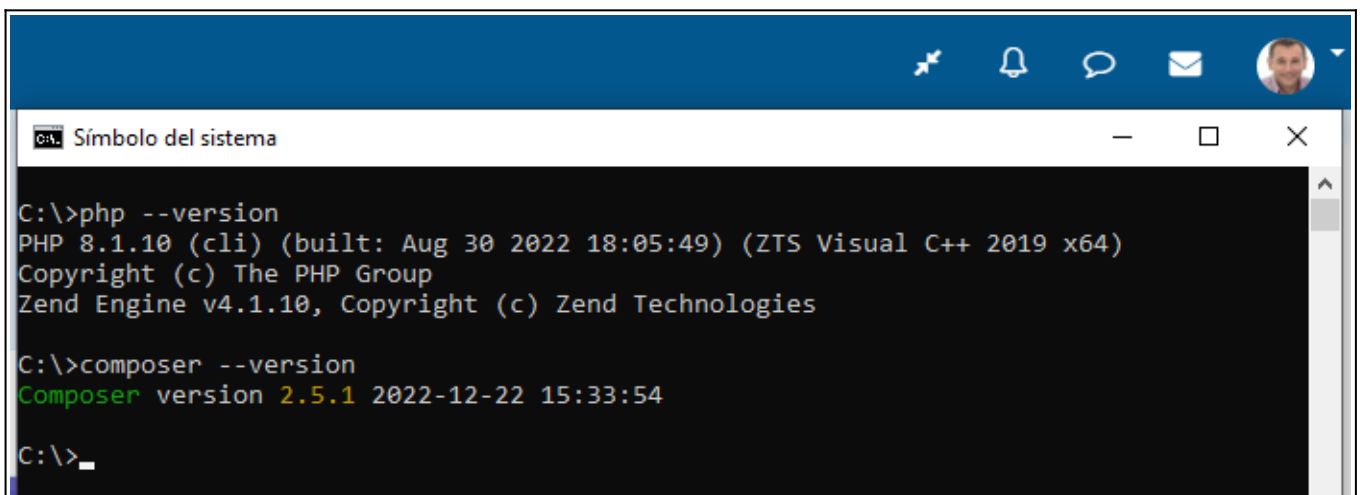
y desde ahí iniciamos el servidor con el comando

```
php artisan serve
```

Una vez que haya iniciado el servidor de desarrollo de Artisan, podrá acceder a la aplicación en un navegador web en <http://localhost:8000>

E) Creación de un proyecto en blanco de Laravel 10

Siguiendo las instrucciones que he dado realizaré la instalación tal y como se ha descrito:
Comprobamos las versiones instaladas de PHP y de Composer

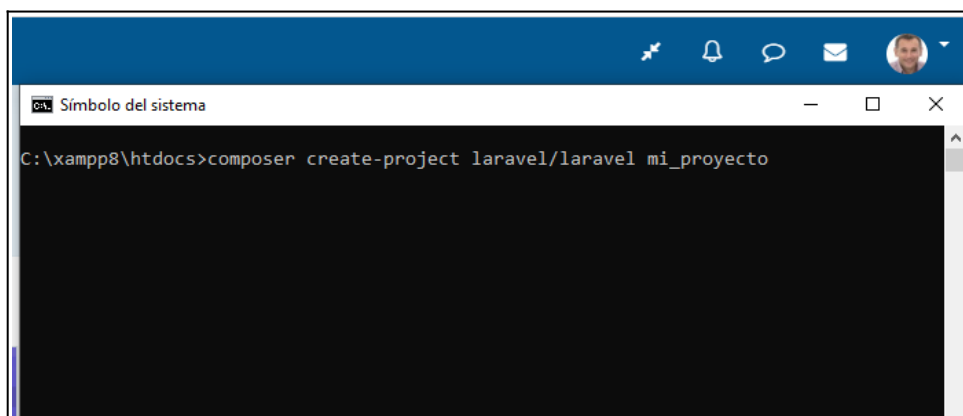


```
C:\>php --version
PHP 8.1.10 (cli) (built: Aug 30 2022 18:05:49) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.1.10, Copyright (c) Zend Technologies

C:\>composer --version
Composer version 2.5.1 2022-12-22 15:33:54

C:\>_
```

Me posiciono en el directorio contenedor que será el htdocs de xampp y ejecuto la línea de comando para la instalación.



```
C:\xampp8\htdocs>composer create-project laravel/laravel mi_proyecto
```

```

C:\xampp\htdocs> - Installing spatie/ignition (1.5.0): Extracting archive
- Installing spatie/laravel-ignition (2.1.0): Extracting archive
69 package suggestions were added by new dependencies, use `composer suggest` to see
details.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

  INFO  Discovering packages.

laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition ..... DONE

80 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

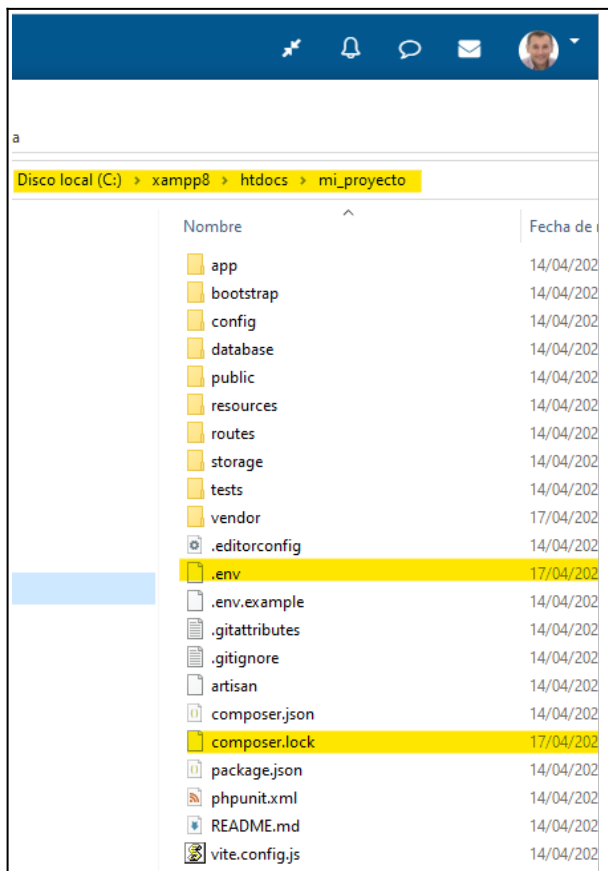
  INFO  No publishable resources for tag [laravel-assets].

No security vulnerability advisories found
> @php artisan key:generate --ansi

  INFO  Application key set successfully.

C:\xampp\htdocs>

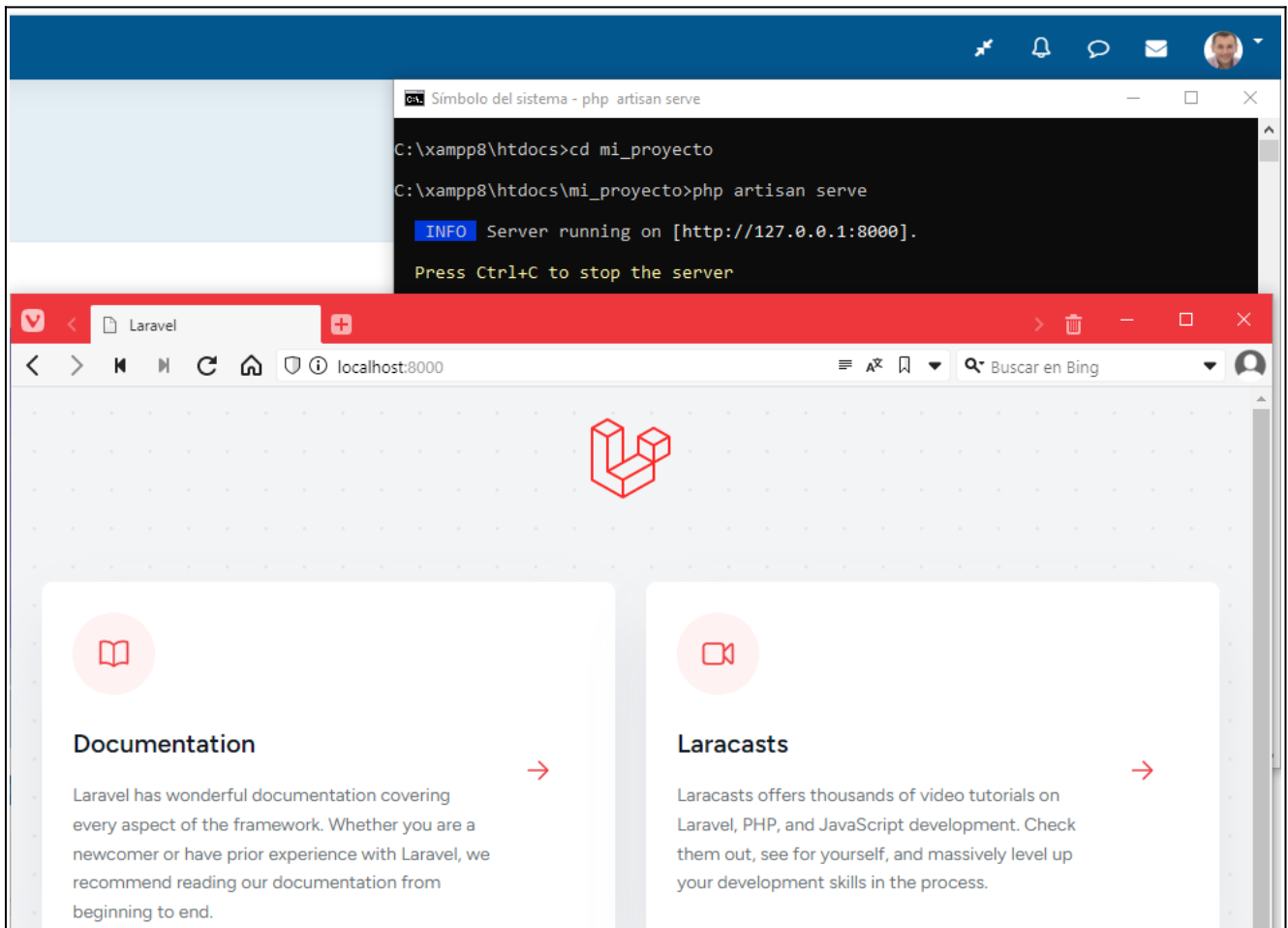
```



El proyecto nuevo llamado "mi_proyecto" se ha creado correctamente

Podemos ver que se ha creado en el interior de la carpeta htdocs de xampp, lo que estaría listo para ejecutar desde ese servidor.

Si bien también es ejecutable usando el servidor propio como vemos a continuación con la captura de pantalla en la que vemos la consola de comandos de windows y un navegador con nuestro primer proyecto abierto en su página de inicio por defecto



2. Rutas y controladores

Crea una aplicación Laravel y añade un controlador llamado LibroController y agrega una ruta para mostrar una lista de libros utilizando un array de ejemplo en dicho controlador. Luego, añade otros dos controladores llamados AutorController y EditorialController e incluye rutas para mostrar información sobre autores y editoriales con arrays de ejemplo en cada controlador.

Creemos un nuevo controlador llamado LibroController. Para ello, posicionados en la terminal de comando en la carpeta raíz del proyecto, ejecutamos el siguiente comando en la terminal:

```
php artisan make:controller LibroController
```

editamos el archivo creado y hacemos el código correspondiente el controlador:

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class LibroController extends Controller
{
    public $libros = [
        ['titulo' => 'La Iliada', 'autor' => 'Homero', 'editorial' => 'Editorial Gredos'],
        ['titulo' => 'La Divina Comedia', 'autor' => 'Dante Alighieri', 'editorial' => 'Alianza
```

```

        Editorial'],
        ['titulo' => 'Don Quijote de la Mancha', 'autor' => 'Miguel de Cervantes', 'editorial'
        => 'Castalia'],
        ['titulo' => 'Cien años de soledad', 'autor' => 'Gabriel García Márquez', 'editorial' =>
        'Sudamericana'],
        ['titulo' => 'Orgullo y Prejuicio', 'autor' => 'Jane Austen', 'editorial' => 'Penguin'],
    ];

    public function index()
    {
        foreach ($this->libros as $libro) {
            echo "<p>Título: {$libro['titulo']}</p>";
            echo "<p>Autor: {$libro['autor']}</p>";
            echo "<p>Editorial: {$libro['editorial']}</p>";
            echo "<br>";
        }
    }
}

```

Creamos de forma similar un nuevo controlador llamado **AutorController**.

php artisan make:controller AutorController

editamos el archivo creado y hacemos el código correspondiente el controlador:

```

<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class AutorController extends Controller
{
    public $autor = [
        ['nombre' => 'Gabriel García Márquez', 'nacionalidad' => 'Nacionalidad A', 'libros' =>
        'Título A'],
        ['nombre' => 'Miguel de Cervantes', 'nacionalidad' => 'Nacionalidad A', 'libros' =>
        'Título B'],
    ];

    public function index()
    {
        foreach ($this->autor as $autor) {
            echo "<p>Nombre: {$autor['nombre']}</p>";
            echo "<p>Nacionalidad: {$autor['nacionalidad']}</p>";
            echo "<p>Libros: {$autor['libros']}</p>";
            echo "<br>";
        }
    }
}

```

Creamos de forma similar un nuevo controlador llamado **EditorialController**.

php artisan make:controller EditorialController

editamos el archivo creado y hacemos el código correspondiente el controlador:

```

<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;

class EditorialController extends Controller
{
    public $editorial = [
        ['nombre' => 'Editorial Gredos', 'ubicacion' => 'Población A', 'libros' => 'Título A'],
        ['nombre' => 'Alianza Editorial', 'ubicacion' => 'Población B', 'libros' => 'Título B'],
    ];

    public function index()
    {
        foreach ($this->editorial as $editorial) {
            echo "<p>Nombre: {$editorial['nombre']}</p>";
            echo "<p>Ubicación: {$editorial['ubicacion']}</p>";
            echo "<p>Libros: {$editorial['libros']}</p>";
        }
    }
}

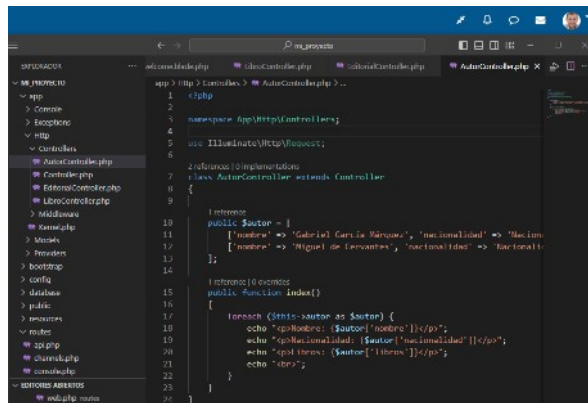
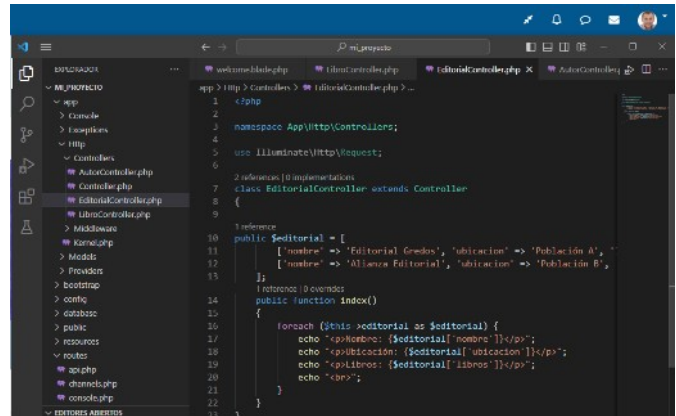
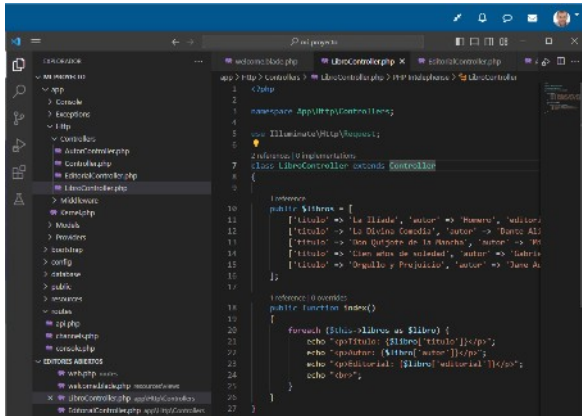
```

```

        echo "<br>";
    }
}
}

```

Los tres controladores son:



A continuación editamos el archivo web.php situado en la carpeta /routes, con ello enlazamos los controladores con las rutas correspondientes llamando a la función que corresponde:

```

<?php

use App\Http\Controllers\AutorController;
use App\Http\Controllers\EditorialController;
use App\Http\Controllers\LibroController;
use Illuminate\Support\Facades\Route;

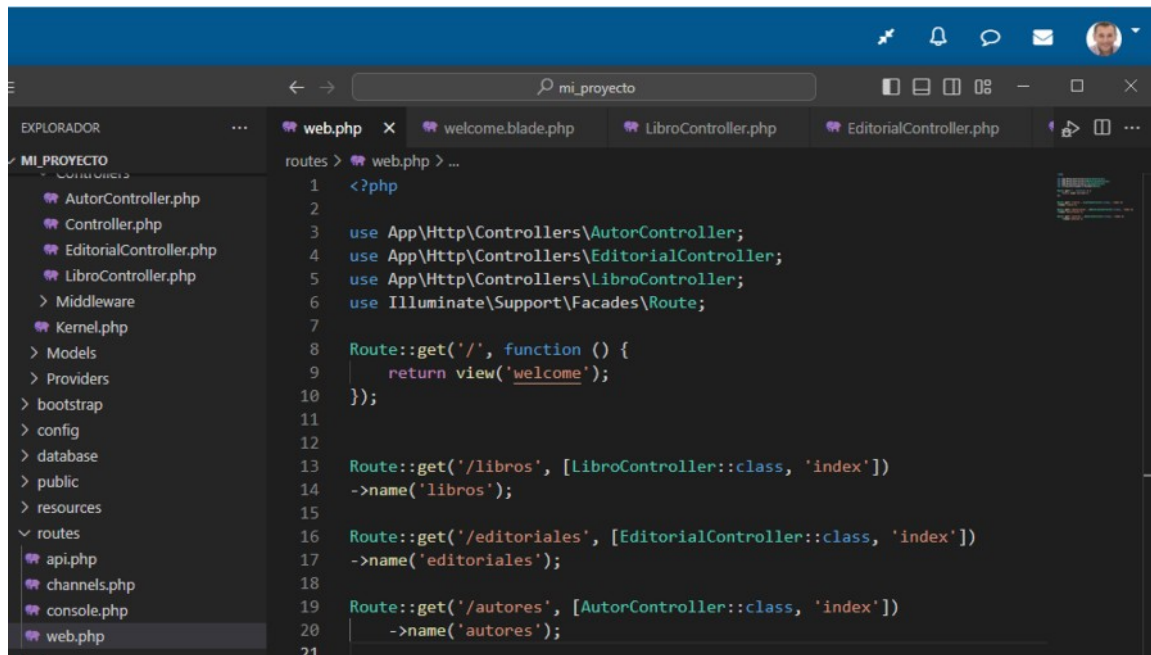
Route::get('/', function () {
    return view('welcome');
});

Route::get('/libros', [LibroController::class, 'index'])
->name('libros');

Route::get('/editoriales', [EditorialController::class, 'index'])
->name('editoriales');

Route::get('/autores', [AutorController::class, 'index'])
->name('autores');

```



3. Vistas mediante Blade

Crea una aplicación en Laravel que permita a los usuarios navegar entre cuatro páginas diferentes: una página de inicio, una página de libros, una página de contacto y una página de acerca de nosotros. Utiliza un controlador para manejar las solicitudes de cada página y cuatro vistas diferentes con Blade para mostrar el contenido en cada una de ellas. En la vista de libros se tiene que mostrar varios libros de ejemplo. Los libros estarán almacenados en un array dentro del controlador.

Este nuevo ejercicio lo haré en el mismo proyecto de Laravel en el que hice el ejercicio anterior. Por ello adaptaré los nombres de los archivos de forma que puedan convivir entre ellos con nombres similares. El motivo de hacer esto es el intento de poner en una sola web de www.alwaysdata.net, todo el contenido del ejercicio.

Para el ejercicio se pide que haya un solo controlador que manejará todas las solicitudes y que además contendrá los datos de los libros en un array.

La vista que contiene los libros es la vista principal que contiene código; las otras tres vistas no se pide que contengan nada en particular.

Archivos creados o modificados para este ejercicio

routes/web.php

views/u3/indice.blade.php

views/u3/libros.blade.php
 views/u3/contacto.blade.php
 views/u3/acercade.blade.php
 controllers/U3_LibroController.php
 public/css/estilos.css

El archivo de las rutas una vez editado queda así:

```
<?php

use App\Http\Controllers\AutorController;
use App\Http\Controllers\EditorialController;
use App\Http\Controllers\LibroController;
use App\Http\Controllers\U3_LibroController;
use Illuminate\Support\Facades\Route;

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

Route::get('/libros', [LibroController::class, 'index'])
->name('libros');

Route::get('/editoriales', [EditorialController::class, 'index'])
->name('editoriales');

Route::get('/autores', [AutorController::class, 'index'])
->name('autores');

Route::get('/u3/libros', [U3_LibroController::class, 'u3_libros'])
->name('libros');

Route::get('/u3/indice', [U3_LibroController::class, 'u3_indice'])
->name('u3_indice');

Route::get('/u3/contacto', [U3_LibroController::class, 'u3_contacto'])
->name('u3_contacto');

Route::get('/u3/acercade', [U3_LibroController::class, 'u3_acercade'])
->name('u3_acercade');
```

El archivo controlador una vez editado queda así:

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class U3_LibroController extends Controller
{
    public function u3_libros()
    {
        $libros = [
            ['titulo' => 'La Iliada', 'autor' => 'Homero', 'editorial' => 'Editorial Gredos'],
            ['titulo' => 'La Divina Comedia', 'autor' => 'Dante Alighieri', 'editorial' => 'Alianza Editorial'],
            ['titulo' => 'Don Quijote de la Mancha', 'autor' => 'Miguel de Cervantes', 'editorial' => 'Castalia'],
            ['titulo' => 'Cien años de soledad', 'autor' => 'Gabriel García Márquez', 'editorial' => 'Sudamericana'],
            ['titulo' => 'Orgullo y Prejuicio', 'autor' => 'Jane Austen', 'editorial' => 'Penguin'],
        ];
        return view('u3.libros', ['var_libros' => $libros]);
    }
}
```



```

public function u3_indice(){
    return view('u3.indice');
}

public function u3_contacto()
{
    return view('u3.contacto');
}

public function u3_acercade()
{
    return view('u3.acercade');
}
}

```

El archivo de vista que contiene los libros queda así:

```

<?php
echo "<a href='../'>IR A LA PÁGINA DE ÍNDICE GENERAL</a>";
echo "<br>";
echo "<a href='../u3/indice'>IR A LA PÁGINA DE ÍNDICE DEL EJERCICIO 3</a>";

foreach ($var_libros as $libro) {
    echo "<p>Título: {$libro['titulo']}</p>";
    echo "<p>Autor: {$libro['autor']}</p>";
    echo "<p>Editorial: {$libro['editorial']}</p>";
    echo '<br>';
}
echo "<a href='../'>IR A LA PÁGINA DE ÍNDICE GENERAL</a>";
echo "<br>";
echo "<br>";
echo "<a href='../u3/indice'>IR A LA PÁGINA DE ÍNDICE DEL EJERCICIO 3</a>";

```

Los archivos de vista que contienen las páginas contacto, acerca de nosotros e índices solo contienen datos básicos.

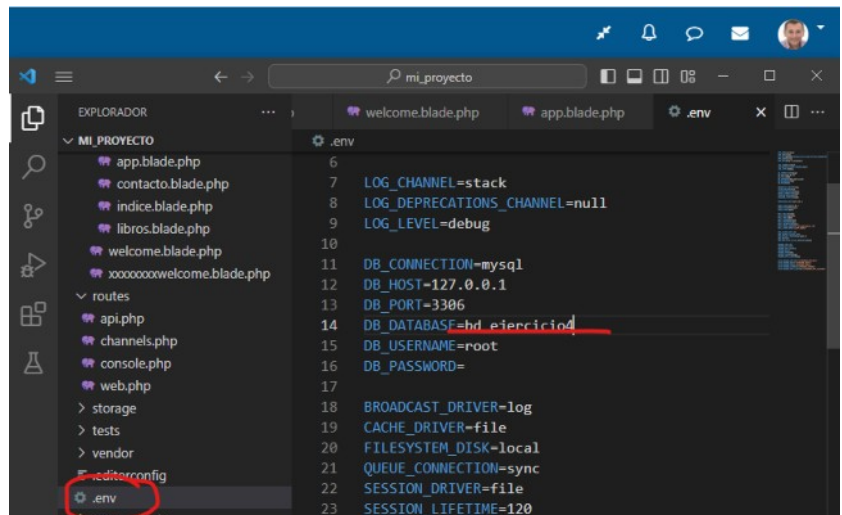
4. Bases de datos

Crea una aplicación en Laravel que tenga una base de datos para almacenar información sobre películas y discos. Utiliza migraciones para crear las tablas necesarias en la base de datos. Luego, usa query builder de Laravel e inserta algunos ejemplos en la base de datos así como modificar, borrar y mostrar películas y discos.

Para este ejercicio voy a crear una base de datos en blanco en phpmyadmin, la voy a llamar bd_ejercicio4

seguidamente voy a editar el archivo .env para establecer con Laravel los datos de conexión a la base de datos

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=bd_ejercicio4
DB_USERNAME=root
DB_PASSWORD=
```



Voy a crear una sola tabla en la base de datos, ya que se pide que la base de datos contenga películas y discos, no es necesario nada más que una tabla con una anotación del tipo de obra que se inserta.

Creo una migración con php artisan y para ello desde la terminal y posicionándome en la carpeta del proyecto escribo:

```
php artisan make:migration create_obras_table
```

recibo el mensaje:

```
Migration [C:\xampp8\htdocs\mi_proyecto\database\Migrations\2023_04_18_192841_create_obras_table.php]
created successfully.
```

Seguidamente voy a editar el archivo de migración que he creado

```
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('obras', function (Blueprint $table) {
            $table->id();
            $table->string('tipo');
            $table->string('titulo');
            $table->string('artista');
            $table->date('lanzamiento');
            $table->string('genero');
            $table->integer('duracion');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {

```

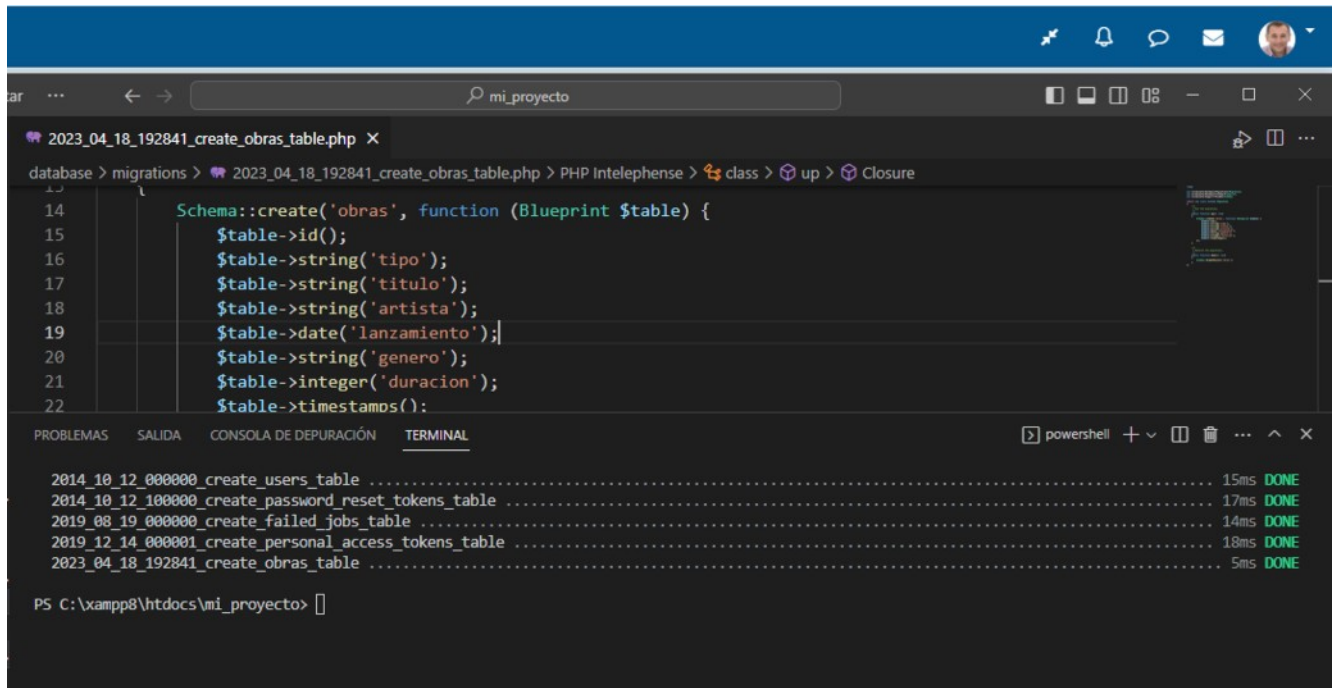
```

        Schema::dropIfExists('obras');
    }
};

```

Una vez guardado el archivo voy a realizar la migración con `php artisan migrate` para crear la tabla dentro de la base de datos en blanco que tengo en mi servidor.

```
php artisan migrate
```



dejo captura de la terminal de Visual Studio Code que es con la terminal que he realizado la migración.

Tengo en cuenta que en caso de que quiera modificar las migraciones deberé realizar lo siguiente:

```
php artisan migrate:fresh o migrate:refresh
```

A continuación voy a crear un controlador con

```
php artisan make:controller ObraController
```

Desde la terminal recibo este mensaje:

```
Controller [C:\xampp8\htdocs\mi_proyecto\app\Http\Controllers\ObraController.php] created successfully.
```

```
<?php
```

```

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

```

```

class ObraController extends Controller
{
    public function home()
    {
        return view('u4.inicio');
    }

    public function obras()
    {
        $obras = DB::table('obras')->get();

        return view('u4.obras', ['obras' => $obras]);
    }

    public function create()
    {
        return view('u4.insertar');
    }

    public function destroy(Request $id)
    {
        DB::table('obras')->where('id', '=', $id)->delete();

        return view('u4.borrar');
    }

    public function store(Request $request)
    {
        DB::table('obras')->insert([
            'tipo' => $request->input('tipo'),
            'titulo' => $request->input('titulo'),
            'artista' => $request->input('artista'),
            'lanzamiento' => $request->input('lanzamiento'),
            'genero' => $request->input('genero'),
            'duracion' => $request->input('duracion'),
        ]);

        return redirect()->route('inicio');
    }
}

```

Para modelar el aspecto de las páginas que voy a crear, editaré el archivo `public/css/estilo.css` con el código que vaya necesitando.

No inserto el archivo, porque iré modificándolo según la necesidad.

El siguiente paso que realizaré será editar el archivo que ya tengo creado de ejercicios anteriores, `routes/web.php` con el siguiente código, que inserto completo si bien las líneas que me sirven para este ejercicio están en **negrita**:

```

<?php

use App\Http\Controllers\AutorController;
use App\Http\Controllers\EditorialController;
use App\Http\Controllers\LibroController;
use App\Http\Controllers\ObraController;
use App\Http\Controllers\U3_LibroController;
use Illuminate\Support\Facades\Route;

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

Route::get('/libros', [LibroController::class, 'index'])

```

```

->name('libros');

Route::get('/editoriales', [EditorialController::class, 'index'])
->name('editoriales');

Route::get('/autores', [AutorController::class, 'index'])
->name('autores');

Route::get('/u3/libros', [U3_LibroController::class, 'u3_libros'])
->name('libros');

Route::get('/u3/indice', [U3_LibroController::class, 'u3_indice'])
->name('u3_indice');

Route::get('/u3/contacto', [U3_LibroController::class, 'u3_contacto'])
->name('u3_contacto');

Route::get('/u3/acercade', [U3_LibroController::class, 'u3_acercade'])
->name('u3_acercade');

Route::get('/u4/inicio', [ObraController::class, 'home'])
->name('inicio');

Route::get('/u4/obras', [ObraController::class, 'obras'])
->name('obras');

Route::get('/u4/insertar', [ObraController::class, 'create'])
->name('anadir');

Route::post('/u4/insertar', [ObraController::class, 'store'])
->name('insertar');

Route::get('/u4/borrar', [ObraController::class, 'destroy'])
->name('borrar');

```

En este paso, y para mejorar lo que hice en el ejercicio 3, voy a crear una vista como plantilla, y con blade haré que se modifiquen las partes que no son comunes.

La plantilla es la siguiente:

`resources/views/layouts/app.blade.php`

con el siguiente código:

```

<html xmlns='http://www.w3.org/1999/xhtml' lang='es'>
  <head>
    <meta charset='utf-8' />
    <title>@yield('title') - Almacén de obras</title>
    <link rel='stylesheet' href='{ asset("css/estilo.css") }'>
  </head>
  <body>
    <h1>Mi almacén de obras: películas y discos</h1>

    @yield('content')

    <footer>Creado por Pedro Godoy</footer>

  </body>
</html>

```

Creo a continuación las siguientes páginas:

`resources/views/u4/inicio.blade.php`

```
resources/views/u4/insertar.blade.php
resources/views/u4/obras.blade.php
resources/views/u4/borrar.blade.php
```

resources/views/u4/inicio.blade.php

```
@extends('layouts.app')

@section('title', 'Inicio')

@section('content')

    <nav class="menu">
        <a href="..">Índice general</a>
        Inicio ejercicio 4
        <a href="{{ route("obras") }}">Películas y discos</a>
        <a href="{{ route("anadir") }}">Añadir obras</a>

    </nav>

    <h2>Inicio</h2>

    <p>Web de ejemplo que muestra y añade obras como películas y discos en una base de datos.</p>

@endsection
```

resources/views/u4/insertar.blade.php

```
@extends('layouts.app')

@section('title', 'Insertar')

@section('content')

    <nav class="menu">
        <a href="..">Índice general</a>
        <a href="{{ route("inicio") }}">Inicio ejercicio 4</a>
        <a href="{{ route("obras") }}">Películas y discos</a>
        Añadir obras
    </nav>

    <h2>Insertar una nueva obra: Película o disco</h2>

    <p>Formulario para insertar nueva película o disco</p>
    <br>

    <form method="post" action="{{ route("insertar") }}">
        @csrf
        <label>Tipo:</label>
        <input type="text" name="tipo" required="required"><br />
        <label>Título:</label>
        <input type="text" name="titulo" required="required"><br />
        <label>Artista:</label>
        <input type="text" name="artista" required="required"><br />
        <label>Lanzamiento:</label>
        <input type="date" name="lanzamiento" required="required"><br />
        <label>Género:</label>
        <input type="text" name="genero" required="required"><br />
        <label>Duración:</label>
        <input type="number" name="duracion" required="required"><br /><br />
        <input type="submit" value="Enviar">
    </form>

@endsection
```

resources/views/u4/obras.blade.php

```

@extends('layouts.app')

@section('title', 'Obras')

@section('content')

    <nav class="menu">
        <a href="..">Índice general</a>
        <a href="{{ route("inicio") }}">Inicio ejercicio 4</a>
        Películas y discos
        <a href="{{ route("anadir") }}">Añadir obras</a>
    </nav>

    <h2>Obras: peliculas y discos</h2>

    <p>Películas y discos insertados en la base de datos.</p>

    @if (count($obras) > 0)

        <table>
            <thead>
                <tr>
                    <th>Tipo</th>
                    <th>Título</th>
                    <th>Artista</th>
                    <th>Lanzamiento</th>
                    <th>Género</th>
                    <th>Duración</th>
                </tr>
            </thead>
            <tbody>
                @foreach ($obras as $obra)
                    <tr>
                        <td>{{ $obra->tipo }}</td>
                        <td>{{ $obra->titulo }}</td>
                        <td>{{ $obra->artista }}</td>
                        <td>{{ $obra->lanzamiento }}</td>
                        <td>{{ $obra->genero }}</td>
                        <td>{{ $obra->duracion }}</td>
                    </tr>
                @endforeach
            </tbody>
        </table>

    @else

        <p>No hay obras añadidas actualmente!</p>

    @endif

@endsection

```

resources/views/u4/borrar.blade.php

```

@extends('layouts.app')

@section('title', 'Borrar registro')

@section('content')

    <nav class="menu">
        <a href="..">Índice general</a>
        <a href="{{ route("inicio") }}">Inicio ejercicio 4</a>
        <a href="{{ route("obras") }}">Películas y discos</a>
        Añadir obras
    </nav>

    <h2>Borrar un registro de la base de datos</h2>

    <p>Inserta el ID del registro a eliminar</p>
    <br>

```

```

<form method='get' action='{ route("borrar") }'>
  @csrf
  <label>ID:</label>
  <input type='number' name='id' required='required'><br />
  <br />
  <input type='submit' value='Enviar'>
</form>

@endsection

```

Antes de terminar este ejercicio voy a crear un archivo para llevar la base de datos con contenido. Para ello voy a crear un archivo seeder con el que introduciré datos a nuestra tabla

```
php artisan make:seeder ObraSeeder
```

Edito este archivo que se ha creado en /database/seeder/ObraSeeder.php

```

<?php
namespace Database\Seeders;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
class ObraSeeder extends Seeder
{
    public function run(): void
    {
        DB::table('obras')->insert([
            'tipo' => 'Disco', 'titulo' => 'The wall', 'artista' => 'Pink Floyd',
            'lanzamiento' => '1970/01/01', 'genero' => 'musical', 'duracion' => '55'
        ]);
        DB::table('obras')->insert([
            'tipo' => 'Pelicula', 'titulo' => 'La naranja mecánica', 'artista' => 'Stanley Kubrick',
            'lanzamiento' => '1967/01/01', 'genero' => 'guerra', 'duracion' => '120'
        ]);
        DB::table('obras')->insert([
            'tipo' => 'Pelicula', 'titulo' => 'Amélie', 'artista' => 'Jean-Pierre Jeunet',
            'lanzamiento' => '2001/01/01', 'genero' => 'romantica', 'duracion' => '110'
        ]);
        DB::table('obras')->insert([
            'tipo' => 'Pelicula', 'titulo' => 'Lawrence de Arabia', 'artista' => 'David Lean',
            'lanzamiento' => '1962/01/01', 'genero' => 'aventuras', 'duracion' => '150'
        ]);
        DB::table('obras')->insert([
            'tipo' => 'Disco', 'titulo' => 'Back in Black', 'artista' => 'ACDC',
            'lanzamiento' => '1980/01/01', 'genero' => 'rock', 'duracion' => '90'
        ]);
        DB::table('obras')->insert([
            'tipo' => 'Disco', 'titulo' => 'Born in the USA', 'artista' => 'Bruce Springsteen',
            'lanzamiento' => '1984/01/01', 'genero' => 'rock', 'duracion' => '120'
        ]);
    }
}

```

Lo ejecuto desde la terminal del mismo programa Visual Studio Code

```
php artisan db:seed --class=ObraSeeder
```

La ejecución es correcta

Para terminar este ejercicio he creado un nuevo apartado en el menú inicial y todas las páginas enlazan con este menú que nos sirve para distribuir los distintos ejercicios.

6. CRUD

Crea una aplicación de Laravel que permita a los visitantes añadir, ver, actualizar y eliminar libros. La aplicación debe tener una pestaña de inicio, una pestaña que muestre una lista de los libros existentes, otra pestaña para insertar un nuevo libro, una pestaña para actualizar y una última pestaña para eliminar un libro existente.

Pasos seguidos para este ejercicio:

He creado una nueva tabla como base de datos llamada Librería.

Creamos el modelo y a la vez la migración de la tabla Librería

```
PS C:\xampp8\htdocs\mi_proyecto> php artisan make:model Libreria -m
```

Se han creado el modelo y el archivo de migración

```
INFO Model [C:\xampp8\htdocs\mi_proyecto\app\Models\Libreria.php]
created successfully.
```

```
INFO Migration [C:\xampp8\htdocs\mi_proyecto\database\Migrations/
2023_04_20_160924_create_librerias_table.php]
created successfully.
```

Ahora voy a editar el archivo de migración para crear las columnas de la tabla

```
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('librerias', function (Blueprint $table) {
            $table->id();
            $table->string('titulo',150);
            $table->string('autor', 100);
            $table->string('editorial', 100);
            $table->timestamps();
        });
    }
    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('librerias');
    }
};
```

Hacemos la migración con

```
php artisan migrate
```

Comprobamos nuestra base de datos y vemos que se ha creado la tabla librerias y las columnas correctamente

Voy a crear un archivo seeder con el que introduciré datos a nuestra tabla

```
php artisan make:seeder EditorialSeeder
```

Edito este archivo que se ha creado en /database/seeder/EditorialSeeder.php

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class EditorialSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        DB::table('librerias')->insert(['titulo' => 'La Iliada', 'autor' => 'Homero', 'editorial' =>
'Editorial Gredos']);
        DB::table('librerias')->insert(['titulo' => 'La Divina Comedia', 'autor' => 'Dante Alighieri',
'editorial' => 'Alianza Editorial']);
        DB::table('librerias')->insert(['titulo' => 'Don Quijote de la Mancha', 'autor' => 'Miguel de
Cervantes', 'editorial' => 'Castalia']);
        DB::table('librerias')->insert(['titulo' => 'Cien años de soledad', 'autor' => 'Gabriel García
Márquez', 'editorial' => 'Sudamericana']);
        DB::table('librerias')->insert(['titulo' => 'Orgullo y Prejuicio', 'autor' => 'Jane Austen',
'editorial' => 'Penguin']);
    }
}
```

Lo ejecuto desde la terminal del mismo programa Visual Studio Code

```
php artisan db:seed --class=EditorialSeeder
```

La ejecución es correcta y ahora hay datos en la tabla de la bd

Queda terminado el trabajo con las tablas

Seguimos trabajando con los controladores

Creamos el controlador

```
php artisan make:controller LibreriaController --model=Libreria --resource
```

Creado satisfactoriamente en app/Http/Controllers/LibreriaController.php

```
<?php

namespace App\Http\Controllers;
```

```

use App\Models\Libreria;
use Illuminate\Http\Request;

class LibreriaController extends Controller
{
    public function index()
    {
        return view('u6.index');
    }

    public function create()
    {
        //
    }

    public function store(Request $request)
    {
        //
    }

    public function show(Libreria $libreria)
    {
        //
    }

    public function edit(Libreria $libreria)
    {
        //
    }

    public function update(Request $request, Libreria $libreria)
    {
        //
    }

    public function destroy(Libreria $libreria)
    {
        //
    }
}

```

Vemos que se han creado todos los métodos necesarios pero aún no tienen contenido.

He editado la función index y he añadido la vista que voy a crear a continuación

Para crear la vistas primero creo la plantilla que me servirá de "template"

Creo un archivo de plantilla dentro de la carpeta
/resources/views/layouts/u6plantilla.blade.php

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="{{ asset('css/estilo.css') }}">
    <title>@yield('titulo')</title>
</head>
<body>
    <h1>Ejercicio 6 base de datos de libros</h1>

    @yield('contenido')

    <footer>Creado por Pedro Godoy</footer>
</body>
</html>

```

Observa que he puesto un @yield para el título y un @yield para el contenido

Ahora crearé el archivo index.blade.php dentro de la carpeta de este ejercicio que es la u6/

```

@extends('layouts/u6plantilla')
@section('titulo', 'Libreria')
@section('contenido')
    <main>

    <h2>Listado de Libros</h2>

```

```

<nav class="menu">
  <a href="..">Índice general</a>
  <a href="{{url('u6/create')}}">Nuevo registro</a>
</nav>

</main>
@endsection

```

Aprovecho para crear dos enlaces para de esta forma iniciar un navegador en esa página que iré ampliando

Modifico la función create del controlador

```

public function create()
{
    return view('u6/create');
}

```

Ahora voy a crear la vista create.blade.php

```

@extends('layouts/u6plantilla')
@section('titulo', 'Registrar libros')
@section('contenido')
    <main>
        <div>
            <h2>Registrar nuevo libro</h2>
        </div>

        <p>Formulario para insertar nueva película o disco</p>
        <br>

        <form action="{{ url('u6') }}" method="post">

            @csrf

            <label>Título:</label>
            <input type="text" name="titulo" id="titulo" value="{{ old('titulo') }}"
required="required"><br />
            <label>Autor:</label>
            <input type="text" name="autor" id="autor" value="{{ old('autor') }}"
required="required"><br />
            <label>Editorial:</label>
            <input type="text" name="editorial" id="editorial" value="{{ old('editorial') }}"
required="required"><br />

            <input type="submit" value="Enviar">
        </form>

    </main>
@endsection

```

Voy a editar el controlador la función store

voy a ponerle una validación de parámetros introducidos

luego creo un objeto de la clase Librería e introduzco los datos de un nuevo registro

luego hago un save()

```

public function store(Request $request)
{
    $request->validate([
        'titulo' => 'required|max:150',

```

```

        'autor' => 'required|max:100',
        'editorial' => 'required|max:100',

    ]);

    $libreria = new Libreria();
    $libreria->titulo = $request->input('titulo');
    $libreria->autor = $request->input('autor');
    $libreria->editorial = $request->input('editorial');

    $libreria->save();

    return view("u6.mensaje", ['msg'=> "Registro guardado correctamente"]);
}

```

Para que el usuario esté informado voy a crear una vista y voy a enviarle a esta vista un mensaje para que muestre. Lo haré de esta forma, usando una variable, para usar la misma vista para distintos mensajes

CREAMOS LA VISTA PARA EL mensaje informativo al usuario

```

@extends('layouts/u6plantilla')
@section('titulo', 'Mensaje')
@section('contenido')
    <main>
        <div>
            <h2>{{ $msg }}</h2>
            <a class="boton-formulario" href="{{ url('u6') }}">Volver al menú</a>
        </div>
    </main>
@endsection

```

Para hacer la parte de la lectura de la base de datos voy a editar el controlador para enviar el contenido de la base de datos a la vista que va a generar la tabla

```

public function index()
{
    $librerias = Libreria::all();
    return view('u6.index', ['librerias' => $librerias]);
}

```

Ahora voy a editar la vista donde tenemos el índice que me va a servir para mostrar también el contenido de la base de datos

Añado la tabla tal y como se ve y los enlace a las variables

```

@extends('layouts/u6plantilla')
@section('titulo', 'Libreria')
@section('contenido')
    <main>

        <h2>Listado de Libros</h2>

        <nav class="menu">
            <a href="..">Índice general</a>
            <a href="{{ url('u6/create') }}">Nuevo registro</a>
        </nav>

        <h2 class="texto-cabecera">Tabla con el contenido de la base de datos</h2>
        <table>
            <thead>
                <tr>

```

```

        <th>ID</th>
        <th>Titulo</th>
        <th>Autor</th>
        <th>Editorial</th>
        <th></th>
        <th></th>
    </tr>
</thead>
<tbody>
    @foreach ($librerias as $libreria)
        <tr>
            <td>{{ $libreria->id }}</td>
            <td>{{ $libreria->titulo }}</td>
            <td>{{ $libreria->autor }}</td>
            <td>{{ $libreria->editorial }}</td>
            <td></td>
            <td></td>
        </tr>
    @endforeach
</tbody>
</table>

</main>
@endsection

```

Añado dos botones junto a la tabla de libros para borrar y editar:

```

@extends('layouts/u6plantilla')

@section('titulo', 'Libreria')

@section('contenido')

    <main>

        <h2>Listado de Libros</h2>

        <nav class="menu">
            <a class="boton-menu" href="..">Índice general</a>
            <a class="boton-menu" href="{{ url('u6/create') }}">Nuevo registro</a>
        </nav>

        <h2 class="texto-cabecera">Tabla con el contenido de la base de datos</h2>
        <table>
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Titulo</th>
                    <th>Autor</th>
                    <th>Editorial</th>
                    <th></th>
                    <th></th>
                </tr>
            </thead>
            <tbody>
                @foreach ($librerias as $libreria)
                    <tr>
                        <td>{{ $libreria->id }}</td>
                        <td>{{ $libreria->titulo }}</td>
                        <td>{{ $libreria->autor }}</td>
                        <td>{{ $libreria->editorial }}</td>
                        <td class="boton-formulario"><a href="{{ url('u6/' . $libreria->id .
'/edit') }}">Editar</a></td>
                        <td>
                            <form action="{{ url('u6/' . $libreria->id ) }}" method="post">
                                @method('DELETE')
                                @csrf
                                <button type="submit" class="boton-formulario">Borrar</button>
                            </form>
                        </td>
                    </tr>
                @endforeach
            </tbody>

```

```

        </table>

    </main>

@endsection

```

El controlador también lo modifiko y finalmente queda así:

```

<?php

namespace App\Http\Controllers;

use App\Models\Libreria;
use Illuminate\Http\Request;

class LibreriaController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $librerias = Libreria::all();
        return view('u6.index', ['librerias' => $librerias]);
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        return view('u6/create');
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        $request->validate([
            'titulo' => 'required|max:150',
            'autor' => 'required|max:100',
            'editorial' => 'required|max:100',
        ]);

        $libreria = new Libreria();
        $libreria->titulo = $request->input('titulo');
        $libreria->autor = $request->input('autor');
        $libreria->editorial = $request->input('editorial');

        $libreria->save();

        return view("u6.mensaje", ['msg'=> "Registro guardado correctamente"]);
    }

    /**
     * Display the specified resource.
     */
    public function show(Libreria $libreria)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     */
    public function edit($id)
    {
        $libreria = Libreria::find($id);
        return view('u6.edit', ['libreria'=> $libreria]);
    }
}

```

```

    }

    /**
     * Update the specified resource in storage.
     */
    public function update(Request $request, $id)
    {

        $request->validate([
            'titulo' => 'required|max:150',
            'autor' => 'required|max:100',
            'editorial' => 'required|max:100',

        ]);

        $libreria = Libreria::find($id);
        $libreria->titulo = $request->input('titulo');
        $libreria->autor = $request->input('autor');
        $libreria->editorial = $request->input('editorial');

        $libreria->save();

        return view("u6.mensaje", ['msg' => "Registro editado correctamente"]);
    }

    /**
     * Remove the specified resource from storage.
     */
    public function destroy($id)
    {
        $libreria = Libreria::find($id);
        $libreria->delete();

        return redirect("u6");
    }
}

```

El código completo del ejercicio se puede consultar abriendo con un editor de código cada uno de los archivos que componen el proyecto, y que relaciono a continuación:

```

routes/web.php
resources/views/welcome.blade.php
resources/views/u6/create.blade.php
resources/views/u6/edit.blade.php
resources/views/u6/index.blade.php
resources/views/u6/mensaje.blade.php
database/seeder/EditorialSeeder.php
migrations/2023_04_20_160924_create_librerias_table.php
app/models/libreria.php
app/http/controllers/LibreriaController.php

```


DESPLIEGUE EN ALWAYSDATA.NET

Llega el momento de desplegar la tarea en un servidor en modo de producción.

Sigo los siguientes pasos

1.- Dado que la aplicación está distribuida en dos directorios generales, uno para los ejercicios de programación orientada a objetos y otra para los archivos creados en Laravel, hago un menú simple que se llamará index.html y que situo en el directorio raíz de alwaysdata.net

2.- Usando el cliente FTP FileZilla accedo al directorio www raíz y usando ftp subo las dos carpetas principales del proyecto y el archivo index.html

3.- Creo una nueva base de datos que llamaré pgodpol_bd4 y desde la página web del servidor accedo a <https://phpmyadmin.alwaysdata.com> donde compruebo que la base de datos está creada correctamente.

4.- usando el protocolo SSH para acceso remoto al servidor, configuro con una contraseña y abro una consola usando la dirección: <https://ssh-pgodpol.alwaysdata.net> esta consola solicita nombre de usuario y la contraseña que acabo de configurar en la página principal de alwaysdata.net

5.- La consola de comandos funciona igual que las que ya conozco de Linux/ubuntu, así que usando un editor de texto (nano) edito el archivo .env; hago algunos cambios y el resto del archivo no lo modifico. Los cambios realizados son los siguientes:

```
APP_NAME=god_pol_p_laravel
APP_ENV=production
APP_KEY=base64:9HfqEliJFLIzjsbUrZVrFAjinaZEB37RYueaFZA3NKM=
APP_DEBUG=true
APP_URL=https://pgodpol.alwaysdata.net/god_pol_p_laravel/

DB_CONNECTION=mysql
DB_HOST=mysql-pgodpol.alwaysdata.net
DB_PORT=3306
DB_DATABASE=pgodpol_bd4
DB_USERNAME=pgodpol
DB_PASSWORD=la contraseña sin usar comillas
```

El resto del archivo se queda tal y como estaba

6.- Usando la terminal me posiciono en la carpeta www/god_pol_p_laravel/ y desde este directorio hago la migración de los archivos que ya estaban creados para realizar la base de datos. La migración se ejecuta igual que en local:

```
php artisan migrate
```

Compruebo que las tablas se han creado

7.- A continuación voy a llenar de contenido de prueba dos de las tablas creadas. Durante la realización de los programas me serví de dos archivos *seeders* que contienen clases con el mismo nombre: EditorialSeeder.php y ObraSeeder.php que me servirán para rellenar las tablas.

Para ello ejecuto desde la terminal los comandos:

```
php artisan db:seed --class=EditorialSeeder  
php artisan db:seed --class=ObraSeeder
```

La ejecución es correcta y ahora hay datos en las tablas de la base de datos

8.- El despliegue funciona correctamente y accediendo desde <https://pgodpol.alwaysdata.net> puedo entrar en todo el contenido creado.

CONCLUSIÓN DE LA TAREA.

Se adjunta al proyecto completo que contiene esta tarea un archivo en formato .txt, que explica los problemas y circunstancias que he ido sorteando a lo largo de la realización. Lo realizo en archivo aparte porque con ello cumplo lo que se solicita con el enunciado

Est documento está realizado con OpenOffice (.odt) con el que he generado el documento .pdf que explica la tarea.

*Trabajo realizado por: Pedro Godoy Polaina
Ciclo Formativo Grado Superior Desarrollo Aplicaciones Web.
IES Trassierra (Córdoba).
Finalizado el día: 23 de abril de 2023*