UP5: PHP y Laravel

Introducción

- Tras ver cómo implementar aplicaciones web de lado servidor utilizando Java, vamos a aprender un nuevo lenguaje.
- Además, veremos cómo utilizarlo junto con uno de los frameworks más populares.
- Nos centraremos también en la gestión de sesiones y usuarios, con sus respectivos roles y permisos.

Índice

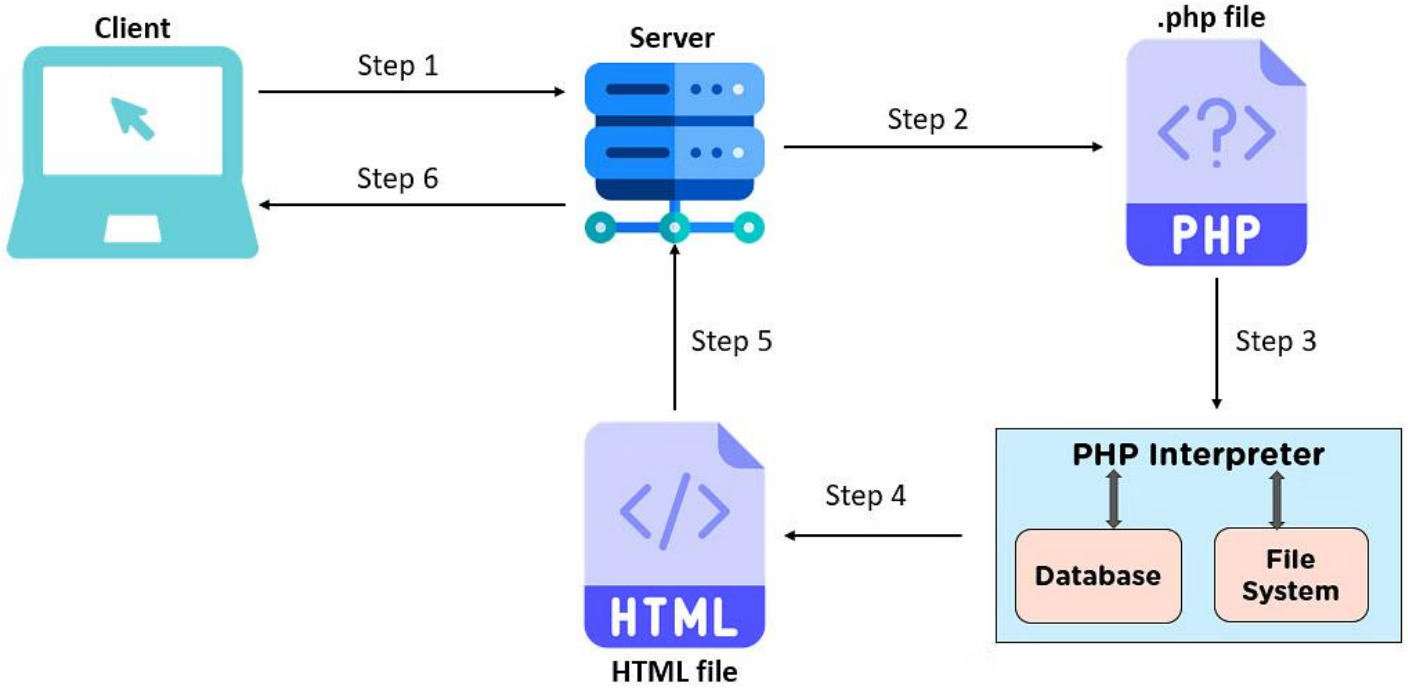
- Introducción y gestión de dependencias
- Sintaxis básica
- Programación Orientada a Objetos

Introducción y gestión de dependencias

Introducción a PHP

- PHP es un lenguaje de código abierto de lado de servidor muy popular para el desarrollo web.
- PHP significa Hypertext Preprocessor.
- Dado que es un lenguaje de lado de servidor, permite que las webs generadas sean dinámicas.
- Es un lenguaje que se integra con HTML, es decir, se puede insertar PHP en un documento formado mayoritariamente por HTML (aunque deberá tener la extensión .php)

Introducción a PHP



Introducción a PHP

- Es compatible con múltiples SGBD como MySQL, MariaDB, PostgreSQL, Oracle, etc.
- Uno de sus usos más frecuentes es para el desarrollo de sistemas de gestión de contenido (CMS), como entornos virtuales de aprendizaje (Aules/Moodle), foros, tiendas en línea, o sitios de noticias (Wordpress).
- Existen multitud de librerías que facilitan el desarrollo de aplicaciones con características muy variadas.
- Se integra con lenguajes de intercambio de datos o de desarrollo web como HTML, CSS, JavaScript, JSON, XML, etc.

Instalación de un entorno con PHP

- Disponemos de varias alternativas para tener instalado PHP junto con un servidor web.
  - Tener instalado XAMPP.
  - Si tenemos Apache en un sistema Ubuntu, ejecutando el comando

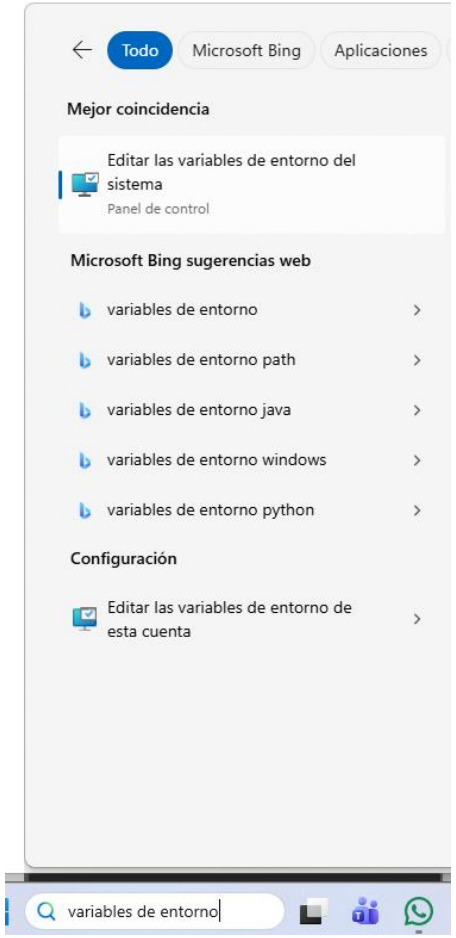
```
sudo apt install php libapache2-mod-php
```

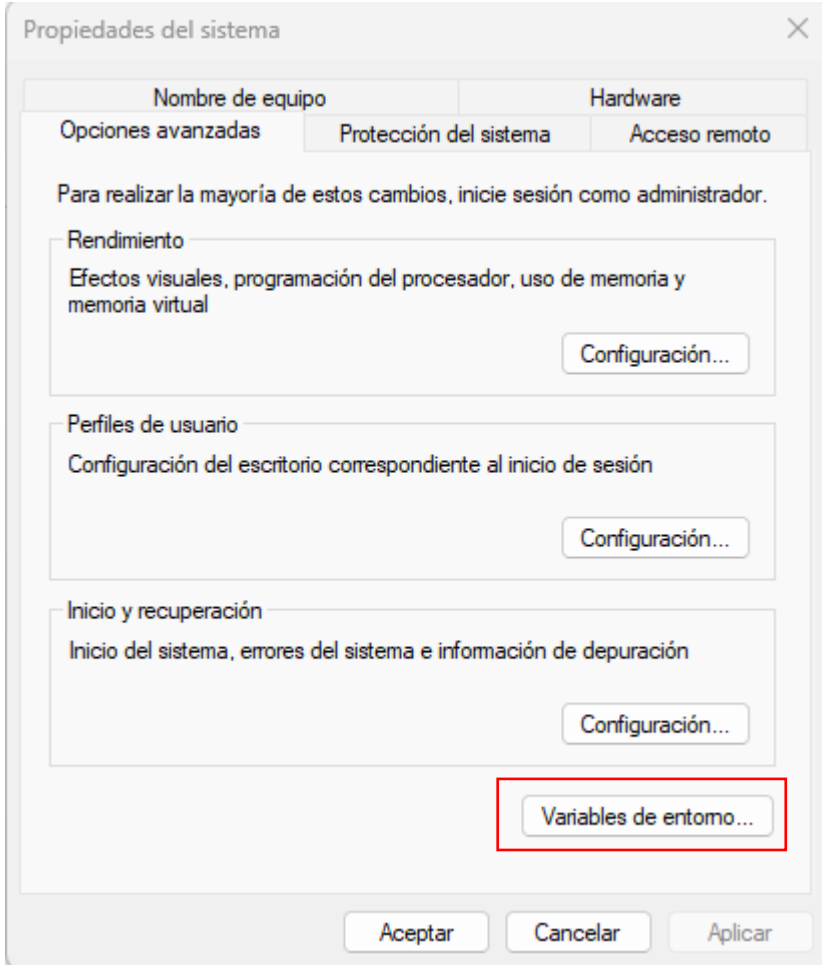
`php -v` (para verificar la instalación)

Instalación de un entorno con PHP

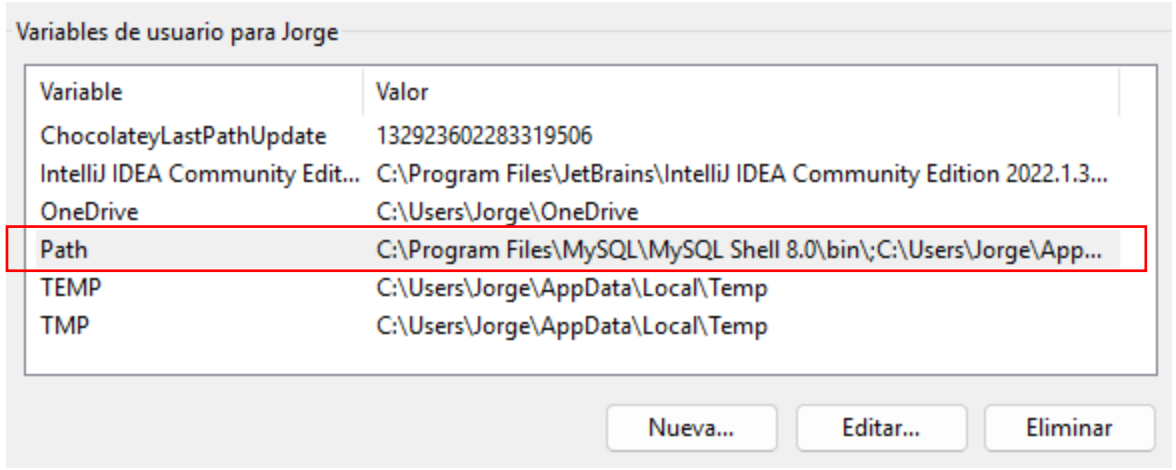
- En caso de querer instalarlo en Windows, debemos descargarlo desde la página oficial. Se recomienda utilizar la versión Thread Safe.
  - https://www.php.net/downloads.php
- A continuación, se deben extraer los archivos en un directorio fácilmente accesible. Por ejemplo, C:\PHP.
- La ruta de instalación debe agregarse al PATH del sistema operativo.
  - Accede a las variables de entorno desde el buscador.

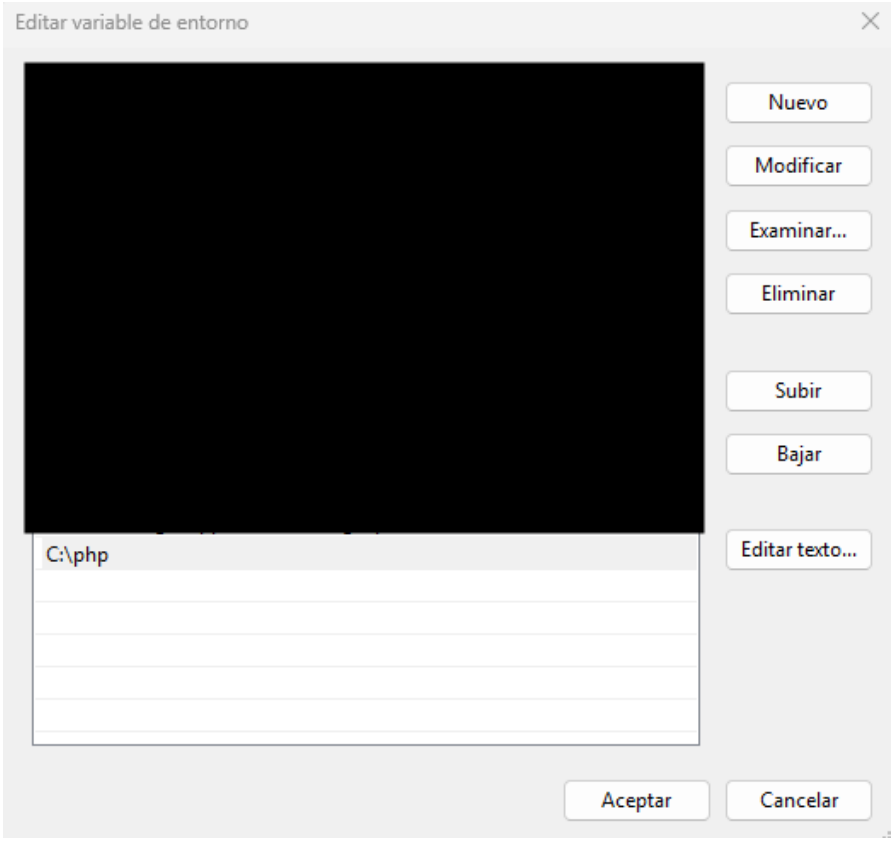
Instalación de un entorno con PHP





Instalación de un entorno con PHP





Configuración de PHP

- Configura el archivo php.ini copiando la versión de desarrollo o producción y modificándole el nombre.
- Modifica el archivo resultante (php.ini) indicando el directorio de extensiones y descomentando las líneas con las extensiones que vayas a utilizar.

Configuración de PHP

nghttp2.dll	17/12/2025 9:11	nghttp2.dll	17/12/2025 9:11	Extensión de la apl...	230 KB
phar.phar.bat	17/12/2025 9:11	phar.phar.bat	17/12/2025 9:11	Archivo por lotes ...	1 KB
pharcommand.phar	17/12/2025 9:11	pharcommand.phar	17/12/2025 9:11	Archivo PHAR	65 KB
php.exe	17/12/2025 9:11	php.exe	17/12/2025 9:11	Aplicación	179 KB
php.ini-development	17/12/2025 9:11	php.ini	17/12/2025 9:11	Archivo de origen ...	71 KB
php.ini-production	17/12/2025 9:11	php.ini-development	17/12/2025 9:11	Archivo INI-DEVEL...	71 KB
		php.ini-production	17/12/2025 9:11	Archivo INI-PROD...	71 KB

Configuración de PHP

- Para el módulo, deberéis tener los siguientes parámetros configurados:
  - error_reporting = E_ALL
  - display_errors = On
  - Extensiones (eliminar el ; de delante): mysqli, mbstring, curl, fileinfo, openssl, pdo_mysql.

Configuración de PHP

```
473  ; Common Values:
474  ;   E_ALL (Show all errors, warnings and notices including coding standards.)
475  ;   E_ALL & ~E_NOTICE  (Show all errors, except for notices)
476  ;   E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR  (Show only errors)
477  ; Default Value: E_ALL
478  ; Development Value: E_ALL
479  ; Production Value: E_ALL & ~E_DEPRECATED
480  ; https://php.net/error-reporting
481  error_reporting = E_ALL
482
483  ; This directive controls whether or not and where PHP will output errors,
484  ; notices and warnings too. Error output is very useful during development, but
485  ; it could be very dangerous in production environments. Depending on the code
486  ; which is triggering the error, sensitive information could potentially leak
487  ; out of your application such as database usernames and passwords or worse.
488  ; For production environments, we recommend logging errors rather than
489  ; sending them to STDOUT.
490  ; Possible Values:
491  ;   Off = Do not display any errors
492  ;   stderr = Display errors to STDERR (affects only CGI/CLI binaries!)
493  ;   On or stdout = Display errors to STDOUT
494  ; Default Value: On
495  ; Development Value: On
496  ; Production Value: Off
497  ; https://php.net/display-errors
498  display_errors = On
```

Configuración de PHP

```
;extension=bz2
extension=curl
;extension=exif
;extension=ffi
;extension=ftp
extension=fileinfo
;extension=gd
;extension=gettext
;extension=gmp
;extension=intl
;extension=ldap
extension=mbstring
extension=mysqli
;extension=odbc
extension=openssl
;extension=pdo_firebird
extension=pdo_mysql
;extension=pdo_odbc
;extension=pdo_pgsql
;extension=pdo_sqlite
;extension=pgsql
;extension=shmop
```

Instalación de Composer

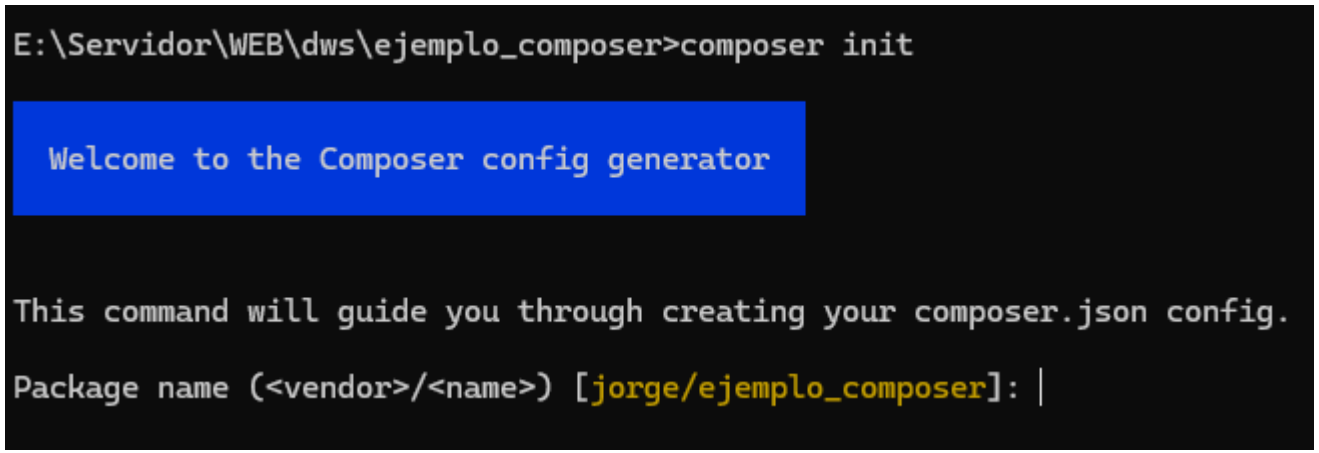
- Composer es un sistema de gestión de dependencias en PHP (como puede serlo Maven en Java).
- Se puede instalar de forma global o específica en un directorio. Nosotros la haremos global.
- En Linux, ejecuta el siguiente comando:

```
curl -sS https://getcomposer.org/installer | phpmv
composer.phar /usr/local/bin/composer
```

- En Windows, utiliza el instalador disponible en https://getcomposer.org/download/

Uso básico de Composer

- Para inicializar un nuevo proyecto de PHP utilizando Composer, ejecutamos el comando `composer init` en el directorio donde queremos desarrollar el proyecto.



Uso básico de Composer

- Para añadir dependencias, utilizaremos el comando `composer require <vendor/package>`.

```
E:\Servidor\WEB\dws\ejemplo_composer>composer require laravel/installer
./composer.json has been updated
Running composer update laravel/installer
Loading composer repositories with package information
Updating dependencies
Lock file operations: 32 installs, 0 updates, 0 removals
  - Locking carbonphp/carbon-doctrine-types (3.2.0)
  - Locking doctrine/inflector (2.1.0)
```

Uso básico de Composer

- Si modificamos manualmente alguna dependencia en el archivo composer.json y queremos instalarla, el comando a utilizar será `composer update`
- Este comando añadirá al archivo .lock la dependencia añadida a mano y la instalará.
- Se recomienda no hacer cambios a mano, sino utilizar el comando require visto anteriormente.

Uso básico de Composer

- Composer genera un archivo autoload.php en la carpeta vendor, que deberemos importar en nuestros archivos .php para acceder a las librerías.
- Composer es útil para gestionar las dependencias tanto a nivel individual como trabajando en equipo, ya que asegura que todos estén trabajando con las mismas versiones.
- Además, facilita la localización de paquetes a través del sitio web Packagist, que es el principal repositorio de paquetes en PHP.
- También permite trabajar con repositorios privados y VCS.
- Los paquetes se definen mediante un <vendor>/<package>

Consulta e instalación de librerías

- Desde la web https://packagist.org/ puedes consultar los paquetes de PHP que han sido publicados.
- Si deseas instalar alguno en tus proyectos, solamente debes ejecutar el comando composer require e indicar los datos del paquete.

Sintaxis básica

Sintaxis básica en PHP: etiquetas

- Las etiquetas son marcas que indican el inicio y el fin de un bloque de código en PHP, de igual forma que en JSP se utilizan las etiquetas <% %> para tal fin.
- En este caso, las etiquetas a utilizar son <?php para indicar el inicio de un bloque y ?> para indicar el fin.
- Para mostrar información por pantalla, se puede usar la etiqueta abreviada <?= $variable; ?>, equivalentes a la impresión de un comentario con etiquetas completas:
  <?php echo $variable; ?>

Sintaxis básica en PHP: comentarios

- Los comentarios en PHP utilizan una sintaxis idéntica a Java, con alguna adición.
  - Disponemos de los comentarios de una línea (// o #).
  - Comentarios de bloque (/* */)
  - De documentación (/** */)

Escribir por pantalla

- Para mostrar información por pantalla, se utilizan dos estrategias:
  - echo: construcción del lenguaje que puede mostrar uno o más valores (separados por comas).
  - print: similar a echo, pero tiene comportamiento de función y siempre devuelve 1. Es más lenta que echo y solo acepta un valor para imprimir.

Variables

- Para definir variables en PHP, se utiliza la nomenclatura signo del dólar seguida del nombre de la variable. No es necesario declararlas antes de utilizarlas.
- A diferencia de Java, PHP tiene tipado débil, lo que significa que no se especifica de qué tipo es una variable, ya que el intérprete lo determinará en tiempo de ejecución.
- Una variable puede tener varios tipos durante la ejecución de un programa.

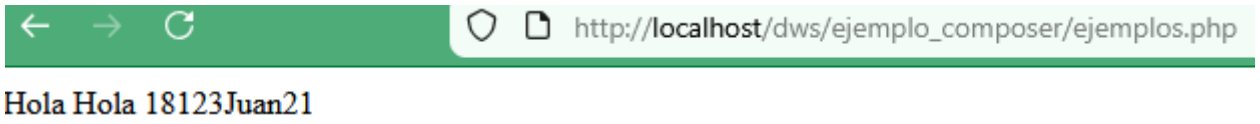
Variables

```php
<!DOCTYPE html>
<html>

<head>
    <title>Ejemplos</title>
</head>

<body>
    <?php
    $variable = "Hola";
    echo $variable;
    ?>
    <?= $variable; ?>
    <?php
    $edad = 18;
    $array = array(1, 2, 3);
    $texto = "Juan";
    $variable = 21;
    echo $edad, implode(array: $array), $texto, $variable;
    ?>
</body>

</html>
```

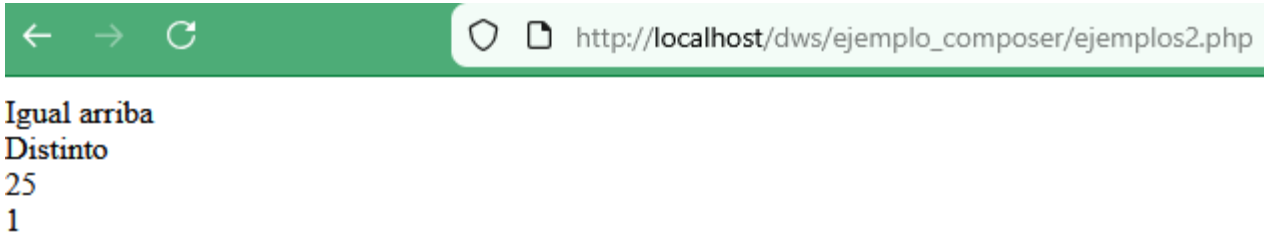


Operadores

- De igual modo, los operadores que utilizamos en PHP son muy similares a los de Java.
  - Aritméticos: +, -, *, /, % (resto), ** (exponenciación).
  - Asignación: =, +=, -=, *=, /=, .= (concatenación de strings).
  - Comparación: == (igualdad de valor), === (igualdad de tipo y valor), != o <> (diferencia), !== (diferencia de tipo y valor), >, <, >=, <=.
  - Lógicos: && o and, || u or, ! (not), xor

Operadores

```php
<?php
$string = "5";
$numero = 2;
$numero += 3;
echo (($string == $numero) ? "Igual arriba" : "Distinto").
"<br>".(($string === $numero) ? "Igual" : "Distinto")."<br>";
$numero = $numero ** 2;
echo $numero."<br>";
echo $numero == 25 or $numero%5 == 0;
?>
```



Tipos de datos

- Existen 10 tipos de datos primitivos en PHP:
  - bool, int, float, string (escalares),
  - array, object, callable, iterable (compuestos). Callable representa datos que pueden ser llamados como funciones e iterable, tipos de datos que se pueden recorrer.
  - resource (mantiene referencia a un recurso externo) y NULL (que representa el valor nulo).

Constantes

- Las constantes son variables cuyo valor no puede ser modificado una vez definido.
- Para representarlas en PHP, hay dos vías:
  - define(“NOMBRE”, VALOR);
  - const NOMBRE = valor;

Arrays

- En PHP, los arrays son una estructura mucho más flexible que en Java, ya que permiten un número indeterminado de valores.
- Además, existen tres tipos: numéricos, asociativos y multidimensionales (que contienen otros arrays).
- Los asociativos permiten utilizar parejas clave-valor, por lo que no es necesario acceder a través del índice.
- Se pueden crear mediante la función array() o con sintaxis abreviada (corchetes):
- $array_uno = array(“nombre” => “Jorge”, “posicion“ => 131);
  $array_dos = [“nombre” => “Carlos”, “posicion” => 133];

Arrays

```php
<?php
$array_uno = array("nombre" => "Jorge",  "posicion" => 131);
$array_dos = ["nombre" => "Carlos", "posicion" => 133];
$numerico = array(1,2,3,4,5,6);
$texto = ["A","B","C","D"];

echo $array_uno["posicion"]."<br>";
echo $array_dos[0]."<br>";
foreach($numerico as $i) {
    echo $i."<br>";
}
for($i = 0; $i < count(value: $texto); $i++) {
    echo $texto[$i]."<br>";
}
?>
```

131
Carlos
1
2
3
4
5
6
A
B
C
D

Arrays: operaciones comunes

- Las operaciones más comunes que realizaremos con arrays son:
  - Recorrerlos (usando for o foreach como en el ejemplo anterior).
  - Contar sus elementos usando la función count($array).
  - Filtrar arrays usando una función: array_filter($array, “nombreFuncion”). El resultado es un nuevo array que hay que asignar.
  - Mapear arrays: aplica una función a los elementos de un array. array_map(“funcion”, $array).

Arrays: operaciones comunes

- Las operaciones más comunes que realizaremos con arrays son:
  - Ordenarlos mediante funciones:
    - sort y rsort ordenan un array indexado de forma ascendente o descendente, respectivamente.
    - asort y ksort ordenan un array asociativo de forma ascendente, según su valor o clave, respectivamente.
    - arsort y krsort ordenan un array asociativo de forma descendente según su valor o clave, respectivamente.

```php
$array_uno = array("nombre" => "Jorge",  "posicion" => 131);
$array_dos = ["nombre" => "Carlos", "posicion" => 133];
$numerico = array(1,2,3,4,5,6);
$texto = ["A","D","B","C"];

print_r(value: sort(array: &$texto));
print_r(value: rsort(array: &$texto));
print_r(value: asort(array: &$array_uno));
print_r(value: ksort(array: &$array_uno));
print_r(value: arsort(array: &$array_uno));
print_r(value: krsort(array: &$array_uno));
```

Arrays: operaciones comunes

- Las operaciones más comunes que realizaremos con arrays son:
  - Combinar arrays con array_merge():
  - Dividir arrays con array_slice().
  - Comprobar si existe una clave con array_key_exists().
  - Comprobar si existe un valor con in_array().

```php
if(in_array(needle: "D", haystack: $texto)) {
    echo "Sí";
}

if(array_key_exists(key: "posicion", array: $array_dos)) {
    echo "Sí";
}
```

Strings

- En PHP también existen muchas de las operaciones sobre cadenas de texto que hemos usado en Java:
  - Expresiones regulares (preg_match, preg_replace, preg_split, preg_match_all).
  - Obtener la longitud (strlen).
  - Reemplazar texto (str_replace).
  - Convertir a mayúsculas/minúsculas (strtoupper, strtolower).
  - Substrings (substr).

Funciones

- Las funciones son bloques de código que realizan una tarea específica y que son llamados desde el flujo principal de ejecución. Pueden ser reutilizadas.
- Se definen utilizando la palabra clave function seguida del nombre de la función y paréntesis, entre los cuales puede recibir valores como argumentos).
- No se especifica el tipo de los argumentos.
- Las funciones pueden o no devolver un resultado. Es decir, no es obligatorio.

Funciones

```php
<?php
    1 reference
    function imprimeSaludo(): void {
        echo "¡Hola, buenos días!<br>";
    }

    1 reference
    function imprimeSaludoPersonalizado($nombre): void {
        echo "¡Hola, ".$nombre.", buenos días!<br>";
    }

    1 reference
    function sumaValores($n1, $n2): mixed {
        return $n1+$n2;
    }

    imprimeSaludo();
    imprimeSaludoPersonalizado(nombre: "Eusebio");
    echo sumaValores(n1: 3,n2: 5);
?>
```

¡Hola, buenos días!
¡Hola, Eusebio, buenos días!
8

Condicionales

- PHP posee las mismas estructuras condicionales que Java: estructuras if-else, operadores ternarios y bloques switch-case.

```php
<?php
$valor = 3;
$lang = "va";
if($valor > 0 && $valor <= 7) {
    switch($valor) {
        case 1: echo ($lang == "va") ? "Dilluns" : "Lunes"; break;
        case 2: echo ($lang == "va") ? "Dimarts" : "Martes"; break;
        case 3: echo ($lang == "va") ? "Dimecres" : "Miércoles"; break;
    }
} else {
    echo "Valor incorrecto";
}

?>
```

Bucles

- También disponemos de los mismos bucles que en Java (while, do-while, for), pero con la adición del foreach.

```php
<?php
$array_uno = array("nombre" => "Jorge",  "posicion" => 131);
$array_dos = ["nombre" => "Carlos", "posicion" => 133];
$numerico = array(1,2,3,4,5,6);
$texto = ["A","B","C","D"];

echo $array_uno["posicion"]."<br>";
echo $array_dos[0]."<br>";
foreach($numerico as $i) {
    echo $i."<br>";
}
for($i = 0; $i < count(value: $texto); $i++) {
    echo $texto[$i]."<br>";
}
?>
```

Programación Orientada a Objetos

Definición de clases y sus elementos

```php
<?php
class Coche {
    //Atributos
    private $color;
    private $marca;

    //Constructor
    public function __construct($color, $marca) {
        $this->color = $color;
        $this->marca = $marca;
    }

    //Getters y setters
    public function getColor() {
        return $this->color;
    }

    public function setColor($valor) {
        $this->color = $valor;
    }

    public function getMarca() {
        return $this->marca;
    }

    public function setMarca($valor) {
        $this->marca = $valor;
    }

    //Métodos
    public function describir() {
        return "El coche de color ".$this->color." es de la marca ".$this->marca;
    }
}
```

Definición de clases y sus elementos

- La definición de una clase en PHP es, nuevamente, muy similar a la de Java, con algunos pequeños matices.
- Muchos desarrolladores marcan sus propiedades como públicas y no implementan getters y setters. No siempre es recomendable.
- El constructor no se llama como la clase, sino __construct([parámetros]), y es una función más.
- No es necesario (aunque se puede) crear la clase en un archivo aparte, ni que este tenga el nombre de la clase.
- Disponemos de tres modificadores de visibilidad: public, protected y private.

Instanciación y uso de la clase

- Para instanciar un objeto, lo hacemos definiendo una variable ($coche, por ejemplo) y utilizando la palabra reservada new, como en Java.
- Los métodos se llaman utilizando la combinación de caracteres “->” tras la variable, a modo de flecha.
- Si las propiedades son públicas, se hace de igual modo, pero sin paréntesis al final.

```php
<?php

$coche = new Coche("rojo", "Kia");
$coche->setColor("blanco");

echo $coche->describir();
```

Herencia y polimorfismo

- También disponemos de herencia y polimorfismo en PHP:

```php
class Vehiculo {
    private $ruedas;

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

    public function getRuedas() {
        return $this->ruedas;
    }

    public function setRuedas($ruedas) {
        $this->ruedas = $ruedas;
    }

    public function arrancar() {
        echo "Arrancando el motor del vehículo";
    }
}
```

```php
class Moto extends Vehiculo {
    private $cilindrada;

    public function __construct($ruedas, $cilindrada) {
        parent::__construct($ruedas);
        $this->cilindrada = $cilindrada;
    }

    public function getCilindrada() {
        return $this->cilindrada;
    }

    public function setCilindrada($cilindrada) {
        $this->cilindrada = $cilindrada;
    }

    public function arrancar() {
        echo "Arrancando la moto.";
    }
}
```

Interfaces y clases abstractas

- PHP permite, además, definir también interfaces y clases abstractas como en Java:

```php
<?php

interface Encendible {
    public function encender();
}

class Bombilla implements Encendible {
    public function encender() {
        echo "Bombilla encendida";
    }
}

$bombilla = new Bombilla();
$bombilla->encender();
```

```php
<?php
//RECUERDA: las clases abstractas no son instanciables.
abstract class Instrumento {
    abstract public function tocar();
}

class Piano extends Instrumento {
    public function tocar() {
        echo "Tocando el piano";
    }
}

$piano = new Piano();
$piano->tocar();
```

Espacios de nombres

- También encontramos este concepto en PHP. Sin embargo, y aunque representa lo mismo, la sintaxis es lo suficientemente diferente para que nos detengamos en ello.
- Recordemos que en Java los namespaces hacían referencia a los packages lógicos y que se separaban por puntos.
- En PHP, los namespaces normalmente van a representar directorios y se van a separar mediante el carácter “\”.
- Se definen mediante la palabra clave “namespace”, y se usan mediante la palabra “use” (en Java era import).

Espacios de nombres

```php
<?php
namespace MediosTransporte\Terrestre;
class Coche {
    //Atributos
    private $color;
    private $marca;

    //Constructor
    public function __construct($color, $marca) {
        $this->color = $color;
        $this->marca = $marca;
    }

    //Getters y setters
    public function getColor() {
        return $this->color;
    }

    public function setColor($valor) {
        $this->color = $valor;
    }

    public function getMarca() {
        return $this->marca;
    }

    public function setMarca($valor) {
        $this->marca = $valor;
    }

    //Métodos
    public function describir() {
        return "El coche de color ".$this->color." es de la marca ".$this->marca;
    }
}
```

```php
<?php

use MediosTransporte\Terrestre\Coche;

$coche = new Coche("rojo", "Kia");
$coche->setColor("blanco");

echo $coche->describir();
```

Traits

- Los traits son un concepto nuevo, que en PHP existe para facilitar la reutilización de código y reemplazar la funcionalidad de herencia múltiple que existe en otros lenguajes de programación.
- Definen fragmentos de código reutilizables por diferentes clases.
- Cada clase puede utilizar múltiples traits, mientras que solamente puede heredar de una clase.
- Los traits se definen con su propia estructura:

```php
trait Loggable {
    public function log($mensaje) {
        echo "Log: ".$mensaje."<br>";
    }
}
```

Traits

- Para utilizar un trait en nuestra clase, utilizaremos la palabra “use”, igual que para importar elementos en otro espacio de nombres.
- Cuando se utiliza un trait, las funciones que define están disponibles dentro de la clase para ser llamadas como si estuviesen definidas en la misma.

Traits

```php
<?php
use MediosTransporte\Terrestre\Loggable;
class Moto extends Vehiculo {
    private $cilindrada;
    use Loggable;

    public function __construct($ruedas, $cilindrada) {
        parent::__construct($ruedas);
        $this->cilindrada = $cilindrada;
    }

    public function getCilindrada() {
        return $this->cilindrada;
    }

    public function setCilindrada($cilindrada) {
        $this->cilindrada = $cilindrada;
    }

    public function arrancar() {
        echo "Arrancando la moto.";
    }
}
```

```php
<?php
$moto = new Moto("2", "125");
$moto->log("Ejemplo de llamada al trait.");
```

Excepciones

- También utilizamos excepciones para gestionar los errores. El funcionamiento es el mismo que en Java, con algunas diferencias menores de sintaxis:

```php
class ExtExc extends Exception {

}
```

```php
try {
    throw new ExtExc("Mensaje de ejemplo");
} catch(ExtExc $excepcion) {
    echo $excepcion->getMessage();
}
```

Include y require

- Include y require son dos instrucciones de PHP que permiten cargar dentro de un archivo el código existente en otro archivo.
  - Include se utiliza cuando el contenido a cargar no es esencial para el funcionamiento de la aplicación.
  - Require se emplea cuando es necesario que el contenido del archivo se cargue para el correcto funcionamiento del programa.
  - Ambas cuentan con una versión _once que se asegura de que un mismo archivo no intente cargarse varias veces.

Include y require

hora.php

```php
<?php
echo date("d-m-Y H:i:s", time());
```

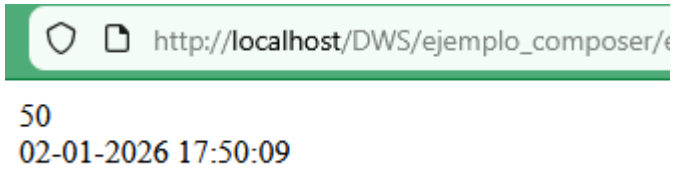
config.php

```php
<?php
$max_alumnos = 50;
```

ejemploIncludeRequire.php

```php
<?php
require "config.php";
echo $max_alumnos."<br>";
include "hora.php";

/**
 * Versión con require_once e include_once:
 * require_once "config.php";
 * echo $max_alumnos;
 * include_once "hora.php";
 */
```



Recogida de datos a través de formularios

Introducción

- Desde HTML, podemos utilizar la etiqueta <form> para crear un formulario, como habréis visto en Lenguaje de Marcas.
- Con los atributos de dicha etiqueta, podemos especificar un punto de destino para los datos de nuestro formulario.
  - Name: nombre del formulario (es opcional).
  - method: forma en la que se enviará el formulario (GET o POST). Recomendable POST.
  - action: punto de destino del formulario.

Lado cliente

- Por ejemplo:

```
<form action="procesar_formulario.php" method="post">
    <p>
        Nombre de usuario: <input type="text" name="usuario" />
    </p>
    <p>
        Contraseña: <input type="password" name="contrasena" />
    </p>
    <input type="submit" value="Iniciar sesión" />
</form>
```

Lado de servidor

- Desde PHP podemos procesar los datos que recibamos del formulario utilizando las variables superglobales, un tipo especial de variable que está disponible en todos los archivos PHP de la aplicación.
- Concretamente, para el caso de ejemplo, dado que el método de envío es POST, utilizaremos $_POST para acceder a los campos del formulario. En caso de que el método fuese GET, la variable sería $_GET. Por ejemplo, para el formulario anterior:

```php
<?php
    $usuario = $_POST["usuario"];
    $contrasena = $_POST["contrasena"];

    echo $usuario."<br>";
?>
```

Lado de servidor

- Sería posible que en un único archivo PHP mostrásemos el formulario y lo procesásemos, puesto que hemos visto que en un documento PHP se puede integrar fragmentos de HTML sin problema.
- En este caso, sería recomendable controlar el flujo de ejecución del programa para evitar incongruencias y que se muestre todo correctamente.
- Por ejemplo, en el caso de un formulario de inicio de sesión, no tendría sentido que, una vez el usuario ha introducido correctamente sus credenciales, se le volviese a mostrar el formulario de inicio de sesión.

Buenas prácticas en la recepción de datos de formulario.

- Se deben validar todas las entradas recibidas, y sanitizarlas para evitar ataques de inyección de código. Por ejemplo, mediante la función htmlspecialchars().
- Debe evitarse enviar formularios mediante peticiones GET, ya que la información es visible a través de la URL.
- Si se va a enviar gran cantidad de información, debe utilizarse POST, ya que GET está limitado en cuanto a longitud.
- Todas las peticiones GET deben ser idempotentes: para un mismo input, deben devolver el mismo output.

Gestión de sesiones

Sesiones

- En ocasiones, necesitamos mantener información sobre un usuario (o cliente) durante un tiempo determinado y en toda una página web.
- Con lo que hemos visto hasta ahora, solamente podemos manejar información durante el tiempo de ejecución de un programa en un único documento. Cuando volvemos a ejecutar ese documento, la información vuelve a su punto inicial.
- Para solucionar este inconveniente, existen las sesiones en el lenguaje PHP.

Sesiones

- Una sesión es una forma de mantener un conjunto de datos o información durante un tiempo determinado en un sitio web.
- Para poder acceder a la información, utilizaremos el array (variable superglobal) $_SESSION.
- Para poder utilizar las sesiones, es necesario llamar a la función session_start(); para arrancarlas.
- Si queremos destruir una sesión, debemos utilizar la función unset().
- Las sesiones cuentan con un identificador propio, conocido como identificador de sesiones o SID, que se almacena en el ordenador del cliente mediante una cookie.

Sesiones

```php
<?php
session_start();
$_SESSION["ejemplo"] = "Esto es un ejemplo de sesión.";

unset($_SESSION);
?>
```

Cookies

Cookies

- Las cookies son archivos de texto con un tamaño máximo de 1K (que además tienen fecha de caducidad), que permiten guardar información relacionada con el sitio web en el ordenador del cliente.
- Pueden utilizarse para almacenar perfiles de usuario, con sus preferencias, para ajustar el contenido de la web a estas.
- Dado que se almacenan en el equipo del cliente, pueden ser utilizados en sucesivas visitas a un sitio web.
- Se declaran con la función setcookie(identificador, valor);
- Se acceden con el array $_COOKIE (de manera similar a las sesiones).
- La diferencia entre COOKIES y sesiones es dónde se guarda la información (cliente y servidor, respectivamente).

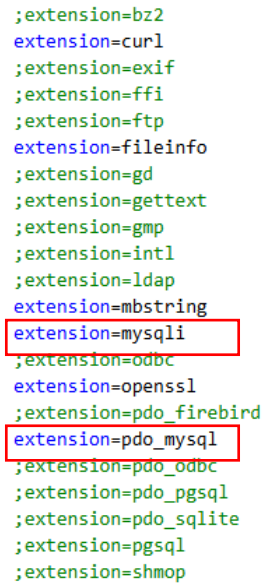
Cookies

```php
<?php
//Almacenamos la cookie usuario con el valor Pepe.
setcookie("usuario", "Pepe");
/*
Esta línea debería imprimir "El nombre de usuario es Pepe.
Nótese que $_COOKIE también es una variable superglobal y, por tanto,
la línea debería ser la misma en todos los documentos del sitio web.
*/
echo "El nombre de usuario es ".$_COOKIE["usuario"]."<br>";
?>
```

Conexión a base de datos

Configuración de las extensiones

- Para poder conectarnos a una base de datos, hay que habilitar la extensión correspondiente (pdo_mysql o mysqli) en el archivo php.ini



Creación de la conexión

- Una vez habilitadas las extensiones, deberemos crear en nuestro proyecto un archivo de conexión que contendrá la información de acceso a la BD. Vamos a llamar a este archivo conexion.php, y deberemos definir variables para el host, el nombre de la base de datos, el usuario, la contraseña y, si queremos, el puerto y la codificación.

```php
<?php

$host = "localhost";
$db = "dws";
$user = "dws";
$pass = "dws";
$puerto = "3306";
$cod = "utf8";
```

Creación de la conexión

- A continuación, en este mismo archivo, definiremos la cadena de conexión y crearemos el objeto PDO que utilizaremos en otros archivos para comunicarnos con la base de datos.

```php
$cadena_conexion = "mysql:host=$host;port=$puerto;dbname=$db";

try {
    $pdo = new PDO($cadena_conexion, $user, $pass);
} catch(\PDOException $exception) {
    die($exception->getMessage()." - Código: ".$exception->getCode());
}
```

Ejecución de consultas preparadas

- Una vez tenemos la conexión creada, podemos incluir el archivo de conexión mediante la sentencia require, y utilizar el objeto PDO creado para realizar las operaciones de BD.
- IMPORTANTE: la BD y las tablas correspondientes deben crearse manualmente en el SGBD (en nuestro caso, MySQL).
- Todas las operaciones deben realizarse mediante consultas preparadas para minimizar riesgos de inyección SQL.
- A continuación, se mostrarán ejemplos de todas las operaciones CRUD. Se recomienda incluirlas dentro de un try-catch por si hubiese algún error.
- Se utilizará una tabla “proyectos” que tiene una id, un nombre, una descripción, un estado, una fecha de inicio y un presupuesto asignado.

SELECT

- Puede devolver un solo resultado o muchos. Según el caso, la forma de obtener los resultados es diferente.

```php
try {
    $statement = $pdo->prepare("SELECT * FROM proyectos WHERE id=:id");
    $statement->execute(["id" => 1]);
    $result = $statement->fetch(\PDO::FETCH_ASSOC);
    echo $result["nombre"];
} catch(Exception $exc) {
    die($exc->getMessage());
}
```

```php
//SELECT MÚLTIPLE
try {
    $statement = $pdo->prepare("SELECT * FROM proyectos");
    $statement->execute();
    $result = $statement->fetchAll(\PDO::FETCH_ASSOC);
    foreach($result as $datos) {
        echo $datos["nombre"]."<br>";
    }
} catch(Exception $exc) {
    die($exc->getMessage());
}
```

INSERT

- En este caso, podemos obtener la ID generada tras la inserción.

```php
//INSERT CON TRANSACCIÓN
try {
    $statement = $pdo->prepare("INSERT INTO proyectos (nombre, descripcion, fecha_inicio, presupuesto_asignado, estado)
    VALUES (:nombre, :descripcion, :fecha_inicio, :presupuesto_asignado, :estado)");
    $pdo->beginTransaction();
    $statement->execute(["nombre" => "Proyecto 3", "descripcion" => "Ejemplo de inserción", "fecha_inicio" => date("Y-m-d", time()),
    "presupuesto_asignado" => 1000, "estado" => "INICIADO"]);
    $idInsercion = $pdo->lastInsertId();
    $pdo->commit();
} catch(PDOException $exc) {
    $pdo->rollback();
    die($exc->getMessage());
}
```

UPDATE

```php
//UPDATE
try {
    $statement = $pdo->prepare("UPDATE proyectos SET estado=:estado WHERE id=:id");
    $pdo->beginTransaction();
    $statement->execute(["estado" => "FINALIZADO", "id" => 4]);
    $pdo->commit();
} catch(PDOException $exc) {
    $pdo->rollback();
    die($exc->getMessage());
}
```

DELETE

```php
//DELETE
try {
    $statement = $pdo->prepare("DELETE FROM proyectos WHERE id>:id");
    $pdo->beginTransaction();
    $statement->execute(["id" => 4]);
    $pdo->commit();
} catch(PDOException $exc) {
    $pdo->rollback();
    die($exc->getMessage());
}
```

Laravel

Qué es Laravel

- Laravel es un framework de desarrollo de aplicaciones web basadas en PHP. Actualmente es uno de los más utilizados y populares del lenguaje.
- Se basa en la arquitectura MVC (modelo-vista-controlador), aunque con algunos matices.
- Su punto de entrada es el enrutador, que permite cargar los controladores correspondientes.

Características principales

- Laravel proporciona un enrutamiento sencillo de utilizar para definir URIs y a qué métodos HTTP responden.
- Facilita la gestión de la base de datos mediante un ORM, Eloquent.
- Está basado en el concepto de migración, una idea que permite mantener la estructura de la base de datos mediante archivos de migración.
- Tiene su propio motor de plantillas, Blade, que facilita mucho el desarrollo de interfaces integradas con el backend.
- Ofrece un sistema de autenticación y autorización sencillo de utilizar y muy completo, sin necesidad de desarrollar prácticamente nada.
- Posee soporte integrado para pruebas automatizadas.

Dónde trabajar

- Para llevar a cabo esta parte del tema, se recomienda:
  - Trabajar en máquina virtual Ubuntu con servidor Apache y MySQL instalado.
  - Utilizar Visual Studio Code con las extensiones PHP Extension Pack y Laravel Extension Pack.
  - Crear las nuevas aplicaciones en el directorio raíz de documentos HTML (/var/www/html).

Instalación de Laravel

- Para poder utilizar Laravel, necesitamos inicialmente tener instalado composer y PHP, algo que ya deberíamos tener del inicio del tema.
- Si no tenemos PHP/composer instalado, en Linux podemos utilizar el comando /bin/bash -c "$(curl -fsSL https://php.new/install/linux/8.5)"
- Instalamos Laravel mediante el comando `composer global require laravel/installer`.
- Necesitaremos también instalar NodeJS y npm para ejecutar los proyectos correctamente:

UP5: PHP y Laravel