PHP

▼ Basics

▼ Print html

Embedding código se refiere a hacer un print (o echo) con PHP, para escribir utilizando otro lenguaje. Es una de las prácticas de programación más desaconsejadas, ya que dificulta la lectura de código haciendo difícil la escalabilidad y mantenimiento de aplicaciones. El php se imprime dentro del html:

```
<body>
 <p><b>Producto:</b> (<?php echo $codigo_de_producto; ?>)
 <?php echo $nombre_producto; ?><br/>
 <b>Precio:</b> USD <?php echo $precio_neto; ?>.- (IVA incluido)</p>
</body>
```

```
<?php
$nombre_de_producto_por_defecto = "Producto";
$nombre_producto = "$nombre_de_producto_por_defecto en oferta";
echo $nombre_producto; // imprime: Producto en oferta
?>
```

¿qué sucede si se necesita concatenar el valor de una variable a una cadena de texto pero sin mediar espacios?

```
<?php
$nombre_de_producto_por_defecto = "Producto";
$nombre_producto = "{$nombre_de_producto_por_defecto}s en oferta";
echo $nombre_producto; // imprime: Productos en oferta
?>
```

Concatenar variables con puntos:

```
Bien:
$detalles_del_producto = "($codigo_de_producto) $nombre_producto. Precio: USD
$precio.-";
Mal:
```

```
$detalles_del_producto = "(" . $codigo_de_producto . ") " .
$nombre_producto . ". Precio: USD " . $precio . ".-";
```

var_dump, isset, unset = destruir, null = vaciar y quitar el tipo de variable.

```
<?php
$producto = "Coca-Cola x 1,5 Lts.";
var_dump($producto);
# salida: string(20) "Coca-Cola x 1,5 Lts."
$producto = "";
var_dump($producto);
# salida: string(0) ""
$producto = NULL;
var_dump($producto);
# salida: NULL
unset($producto);
var_dump($producto);
/*
 Generará un error, ya que la variable $producto ha sido destruida
 Salida:
 PHP Notice: Undefined variable: producto ...
 NULL
*/
?>
```

▼ Include/Require

- Diferencia entre include y require:
  - include() intenta importar al archivo indicado y en caso de no poder hacerlo, arroja un error y continúa ejecutando el resto del script.
  - require(), cuando no logra importar el archivo indicado, arroja un error y finaliza sin permitir que el resto del script continúe ejecutándose.
  - include_once y require_once, si el archivo indicado con "_once" ya ha sido incluido no volverá a importarse.

▼ Functions, Arrays, Objects

▼ Functions

▼ Explode()

```
<?php// Ejemplo
1$pizza  = "porción1 porción2 porción3 porción4 porción5 porción6";
$porciones = explode(" ", $pizza);
echo $porciones[0]; // porción1echo $porciones[1]; // porción2
// Ejemplo
2$datos = "foo:*:1023:1000::/home/foo:/bin/sh";
list($user, $pass, $uid, $gid, $gecos, $home, $shell) = explode(":", $datos);
echo $user;
// foo // *
?>
```

▼ Implode()

Lo contrario, une elementos de un array en un string y eliges si separar los elementos con algún separator o no ponerlo y te deja todo junto.

implode(string `$separator`, array `$array`): string

Introducción a Funciones en PHP

Una característica interesante de PHP es la capacidad de crear funciones anónimas, también conocidas como closures. Estas funciones no tienen nombre y se pueden asignar a variables, lo que permite una mayor flexibilidad en su uso.

Uso del "&" y Asignación de Funciones a Variables

En PHP, el símbolo "&" se utiliza para pasar una referencia a una función en lugar de una copia de la función. Esto permite modificar la función original desde dentro de otra función. Aquí hay un ejemplo:

```
function incrementar(&$num) {
  $num++;
}

$valor = 5;
incrementar($valor);
echo $valor; // Output: 6
```

En este ejemplo, utilizamos el símbolo "&" al pasar la variable `$valor` a la función `incrementar`. Como resultado, la variable se modifica directamente dentro de la función.

También es posible asignar funciones a variables en PHP. Esto puede ser útil cuando se desea pasar una función como argumento.

```
function saludar($nombre) {
  echo "Hola, " . $nombre;
}

$miFuncion = "saludar";
$miFuncion("Juan"); // Output: Hola, Juan
```

En este ejemplo, hemos asignado la función `saludar` a la variable `$miFuncion`. Luego, podemos llamar a la función utilizando la variable y pasarle los argumentos necesarios.

Funciones Anónimas

Las funciones anónimas se definen utilizando la palabra clave `function` seguida de los parámetros de la función y el bloque de código entre llaves. Aquí hay un ejemplo de una función anónima que suma dos números:

```
$suma = function($a, $b) {
  return $a + $b;
};

$resultado = $suma(3, 4); // Output: 7
```

En este ejemplo, hemos asignado la función anónima a la variable `$suma`. Luego, podemos llamar a la función utilizando la variable y pasarle los argumentos necesarios.

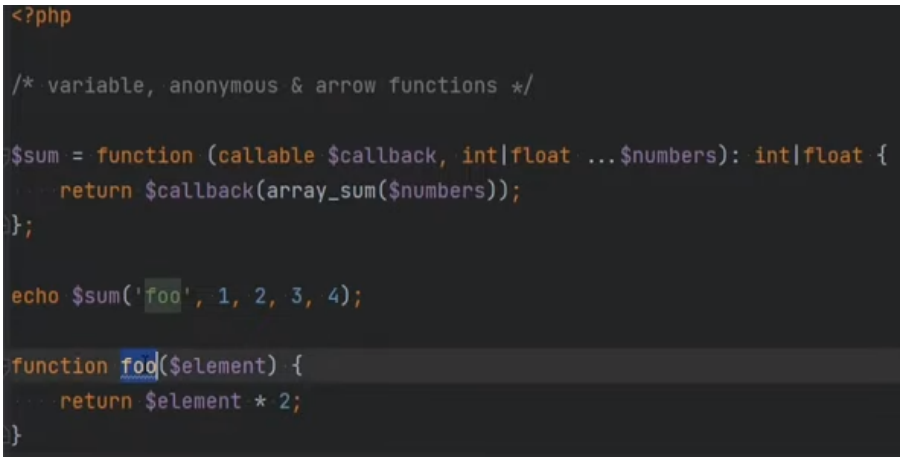
Callbacks

En PHP, los callbacks son funciones que se pasan como argumentos a otras funciones. Esto permite que una función llame a otra función para realizar una tarea específica. Aquí hay un ejemplo de cómo utilizar un callback en PHP:

```
function operacion($num1, $num2, $callback) {
  $resultado = $callback($num1, $num2);
  echo "El resultado es: " . $resultado;
```

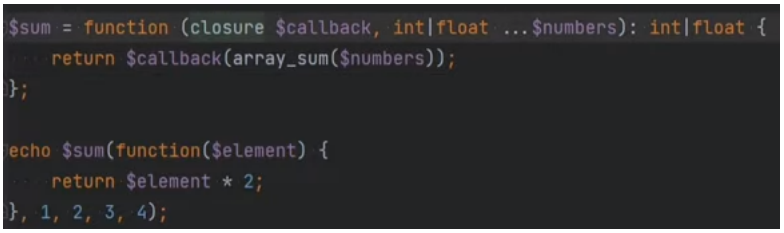
```
}

function suma($a, $b) {
  return $a + $b;
}

operacion(3, 4, "suma"); // Output: El resultado es: 7
```

En este ejemplo, la función `operacion` recibe dos números y un callback como argumentos. El callback se invoca dentro de la función y se le pasan los números como parámetros.



Closure

Si la función callback es anónima, entonces le puedes llamar closure:



El use en PHP es una función anónima que puede acceder a variables fuera de su ámbito. Esto significa que puede utilizar variables definidas fuera de la función en la que se encuentra. Aquí hay un ejemplo de cómo utilizar un closure:

```
function crearSumador($num1) {
  return function($num2) use ($num1) {
    return $num1 + $num2;
  };
}

$suma5 = crearSumador(5);
echo $suma5(3); // Output: 8
```

En este ejemplo, la función `crearSumador` devuelve un closure que suma un número dado (`$num1`) con otro número (`$num2`). Luego, asignamos el closure a la variable `$suma5` y lo llamamos con el número 3.

Los closures son útiles cuando se necesita encapsular cierta lógica y utilizar variables externas dentro de una función anónima.

▼ Arrow Functions

La sintaxis básica de una arrow function es la siguiente:

```
$suma = fn($a, $b) => $a + $b;
```

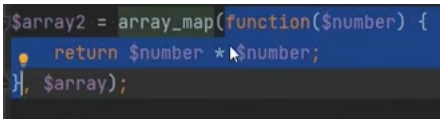
La flecha (`=>`) se utiliza para separar los parámetros del cuerpo de la función.

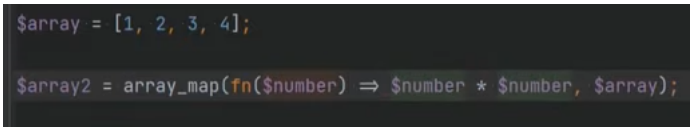
- Usas `fn` para definir la función.
- Son funciones anónimas en una sola línea (No aceptan multiline).
- No se usa `return`. El valor de la expresión se devuelve automáticamente.
- Puede acceder sin el uso de `use`. a las variables que haya en su parent scope, pero no puede modificarlas (ni con el ampersan).

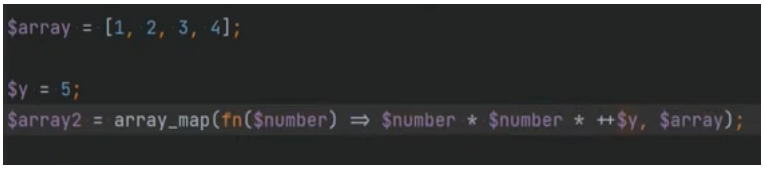
Aquí hay un ejemplo más completo que muestra cómo utilizar una arrow function como callback en la función `array_map`:

```
$nums = [1, 2, 3, 4, 5];
$result = array_map(fn($num) => $num * 2, $nums);
print_r($result); // Output: Array ( [0] => 2 [1] => 4 [2] => 6 [3] => 8 [4] =
> 10 )
```

De esto a lo siguiente:







▼ Arrays

Los arrays en PHP permiten almacenar múltiples valores en una sola variable. Pueden contener diferentes tipos de datos, como números, cadenas de texto y objetos. Aquí hay un ejemplo de cómo declarar un array en PHP:

```
$frutas = array("manzana", "banana", "naranja");
```

Para recorrer un array, puedes utilizar un bucle foreach. Aquí tienes un ejemplo de cómo recorrer el array de frutas y mostrar cada elemento:

```
foreach ($frutas as $fruta) {
  echo $fruta . "
";
}
```

La función `array_reduce` se utiliza para reducir los valores de un array a un solo valor. Aquí hay un ejemplo de cómo sumar todos los elementos de un array utilizando `array_reduce`:

```
$nums = array(1, 2, 3, 4, 5);
$total = array_reduce($nums, function($carry, $num) {
  return $carry + $num;
});
echo $total; // Output: 15
```

La función `array_filter` se utiliza para filtrar los elementos de un array según una condición. Aquí hay un ejemplo de cómo filtrar los números impares de un array utilizando `array_filter`:

```
$nums = array(1, 2, 3, 4, 5);
$oddNums = array_filter($nums, function($num) {
  return $num % 2 != 0;
});
print_r($oddNums); // Output: Array ( [0] => 1 [2] => 3 [4] => 5 )
```

La función `array_walk` se utiliza para aplicar una función a cada elemento de un array. Aquí hay un ejemplo de cómo duplicar cada elemento de un array utilizando `array_walk`:

```
$nums = array(1, 2, 3, 4, 5);
array_walk($nums, function(&$num) {
  $num *= 2;
});
print_r($nums); // Output: Array ( [0] => 2 [1] => 4 [2] => 6 [3] => 8 [4] => 10
)
```

▼ Objects

Resumen de Objetos en PHP

diego.com.es/programacion-orientada-a-objetos-en-php

En PHP, los objetos son instancias de clases que permiten encapsular datos y funcionalidad relacionada. Aquí tienes un resumen de los conceptos clave relacionados con objetos en PHP:

- **Clases**: Una clase es una plantilla para crear objetos. Define las propiedades y métodos que tendrán los objetos creados a partir de ella.
- **Objetos**: Un objeto es una instancia de una clase específica. Representa una entidad del mundo real y puede tener sus propias propiedades y métodos.
- **Constructor**: El constructor es un método especial dentro de una clase que se ejecuta automáticamente cuando se crea un nuevo objeto. Se utiliza para inicializar las propiedades del objeto.
- **ToString**: El método `__toString` es un método especial dentro de una clase que se utiliza para definir cómo se representa un objeto como una cadena de texto cuando se imprime.
- **Métodos**: Los métodos son funciones definidas dentro de una clase que pueden ser invocadas por los objetos de esa clase. Permiten realizar diferentes acciones y manipular los datos del objeto.
- **Herencia**: La herencia es un concepto en la programación orientada a objetos que permite que una clase herede propiedades y métodos de otra clase. La clase que hereda se llama clase hija o subclase, y la clase de la que hereda se llama clase padre o superclase.
- **Polimorfismo**: El polimorfismo es otro concepto clave en la programación orientada a objetos que permite que objetos de diferentes clases se utilicen de manera intercambiable. Esto significa que un objeto de una subclase puede ser tratado como un objeto de la clase padre.

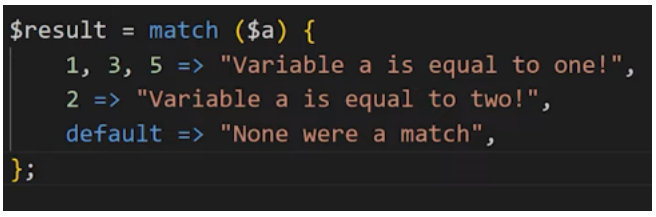
Aquí tienes un ejemplo de cómo se puede implementar la herencia y el polimorfismo en PHP:

```
class Animal {
  protected $nombre;

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

  public function hablar() {
    return "El animal hace un sonido";
  }
}

class Perro extends Animal {
  public function hablar() {
    return "El perro ladra";
  }
}

class Gato extends Animal {
  public function hablar() {
    return "El gato maulla";
  }
}

$animal1 = new Animal("Animal");
$perro1 = new Perro("Firulais");
$gato1 = new Gato("Garfield");

echo $animal1->hablar(); // Output: El animal hace un sonido
echo $perro1->hablar(); // Output: El perro ladra
echo $gato1->hablar(); // Output: El gato maulla
```

En este ejemplo, la clase `Animal` es la clase padre de las clases `Perro` y `Gato`. Ambas clases hijas sobrescriben el método `hablar` heredado de la clase padre para proporcionar su propia implementación. Al llamar al método `hablar` en los objetos de las diferentes clases, se obtienen diferentes resultados según la clase del objeto.

Espero que este resumen te sea útil para comprender los conceptos básicos de objetos en PHP.

Switch raro.



▼ Superglobal Variables

$_SESSION, $_ENV[] (particular to the environment), $_REQUEST[], $_GET , $_POST

```php
<?php
$_SERVER[""];
$_GET[""];
$_POST[""];
$_REQUEST[""];
$_FILES[""];
$_COOKIE[""];
$_SESSION[""];
$_ENV[""];
?>
```

▼ “Bugs”

- Comparación números float (épsilon)

Comparación del tipo float

Como se indica en la advertencia anterior, comprobar la igualdad de valores de punto flotante es problemático debido a la forma en que se representan internamente. Sin embargo, hay maneras de hacer comparaciones de los valores de punto flotante que evitan estas limitaciones.

Para comprobar la igualdad de valores de punto flotante, se utiliza un límite superior en el error relativo debido al redondeo. Este valor se conoce como el épsilon de la máquina o unidad de redondeo, y es la menor diferencia aceptable en los cálculos.

`$a` y `$b` son iguales en 5 dígitos de precisión.

```php
<?php
$a = 1.23456789;
$b = 1.23456780;
$épsilon = 0.00001;

if(abs($a-$b) < $épsilon) {
    echo "true";
}
?>
```

- Settype n weird maths
  - No confiar en settype... ¿?

Sinembargo, realizará operaciones aritméticas de forma correcta, aunque alguno de los números, sea de tipo string:

```
__eugenia_1978_esAR__@mydream:~$ php -r '$a = "33"; $b = 10; echo $a + $b;'
43
```

No obstante, si se intenta realizar una operación aritmética con cadenas de texto, que además de números, incluyan otro caracter, PHP, en vez de fallar y avisarnos del error, pasará por alto la variable conflictiva:

```
__eugenia_1978_esAR__@mydream:~$ php -r '$a = "E33"; $b = 10; echo $a + $b;'
10
```

▼ Config

▼ Error reporting

The `error_reporting(E_ALL);` and `ini_set('display_errors', 1);` lines are used to enable error reporting in PHP. This is very useful during development, as it helps you catch and fix errors early.

However, displaying errors directly to the user can be a security risk in a production environment, as it can expose sensitive information. Therefore, it's common to use different error reporting settings for development and production.

A common approach is to put these lines in a separate PHP file, often called `config.php` or `init.php`, and then include this file at the beginning of each of your other PHP files using `include_once` or `require_once`.

In this file, you can use a condition to check whether the application is in development or production mode, and set the error reporting level accordingly. For example:

```
<?php
if (getenv('APP_ENV') === 'development') {
    error_reporting(E_ALL);
    ini_set('display_errors', 1);
} else {
    error_reporting(0);
    ini_set('display_errors', 0);
    ini_set('log_errors', 1);
}
?>
```

In this example, `getenv('APP_ENV')` is used to get the value of the `APP_ENV` environment variable. You can set this variable to `development` in your development environment and to `production` in your production environment.

This way, you'll see all errors when developing the application, but in production, errors will be hidden from the user and logged to a file instead.

▼ Env files

The `getenv` function in PHP is used to get the value of an environment variable. Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. They are part of the environment in which a process runs.

In the context of a PHP application, environment variables are often used to store sensitive information, such as database credentials, or configuration settings that vary between deployment environments, such as your `APP_ENV` variable.

Here's how you can use `getenv` in your project:

1. **Setting Environment Variables**: First, you need to set the environment variable. This can be done in several ways. One common method is to use a `.env` file, which is a simple text file that contains key-value pairs. There are libraries like `phpdotenv` that can load these variables into the environment for you. Another method is to set them directly in your server configuration.

   ▼ Setting ways:

   1. **Using a .env file**: This is a common approach in many PHP projects. You create a file named `.env` in your project root and define your environment variables in this file. Then, you use a library like `phpdotenv` to load these variables into the environment. Here's an example of what the `.env` file might look like:

      APP_ENV=development

      DB_HOST=localhost

      DB_NAME=mydatabase

      DB_USER=myuser

      DB_PASS=mypassword

      And here's how you might load the `.env` file in your PHP script:

```
<?php
require_once 'vendor/autoload.php';
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__);
$dotenv->load();
?>
```

Note: Never commit your `.env` file to version control. It often contains sensitive information. Instead, you can provide a `.env.example` file with the same keys but dummy values, and instruct other developers to copy this file to `.env` and fill in their own values.

2. **Using the putenv function**: You can use the `putenv` function in PHP to set an environment variable. However, this only sets the variable for the current script, and the variable will disappear once the script finishes executing. Here's an example:

```
<?php
putenv('APP_ENV=development');
?>
```

3. **Using the server configuration**: If you're using an Apache server, you can set environment variables in your `.htaccess` file or `httpd.conf` file. If you're using Nginx, you can set them in your Nginx configuration file. If you're using a cloud hosting provider, they will usually provide a way for you to set environment variables through their dashboard.

Remember, environment variables are often used to store sensitive information, so make sure to keep them secure. Don't include them in your version control system, and don't display them in error messages or logs.

1. **Accessing Environment Variables**: Once the environment variable is set, you can access it in your PHP code using `getenv`. For example, `getenv('APP_ENV')` will return the value of the `APP_ENV` environment variable.

In your `config.php` file, you're using `getenv('APP_ENV')` to check whether the application is in development or production mode, and setting the error reporting level accordingly. This is a good use of environment variables, as it allows you to easily change the error reporting level depending on the environment, without having to modify the code itself.

▼ Autoload

▼ Database credentials

A Json file to load and save the credentials of the database

▼ Redirect

PHP

Redirect a otra página con parámetros GET

```
$dir = "http://" . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . "/main.ph
p?success=$loginEmail";
header("Location:$dir", true, 302);
```

Console log

```
echo "<script type='text/javascript'>console.log('loginEmail: $loginEmail');</script
>";
echo "<script type='text/javascript'>console.log('loginPassword: $loginPassword');</
script>";
```

```
echo "<script type='text/javascript'>console.log('loginCheck: $loginCheck');</script
>";
```

▼ Serialize Json

Serialización de datos en PHP

La serialización es el proceso de convertir datos complejos en una forma que se pueda almacenar o transmitir fácilmente. En PHP, puedes usar la función `serialize()` para convertir un objeto o una estructura de datos en una cadena de caracteres que se puede guardar en un archivo o enviar a través de una red. La serialización preserva la estructura y los valores de los datos, lo que permite su posterior uso.

Aquí tienes un ejemplo de cómo serializar y deserializar un objeto en PHP:

```
class Persona {
  public $nombre;
  public $edad;

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

$persona = new Persona("Juan", 30);

// Serializar el objeto
$serializado = serialize($persona);

echo $serializado; // Output: O:6:"Persona":2:{s:6:"nombre";s:4:"Juan";s:4:"edad";i:3
0;}

// Deserializar el objeto
$deserializado = unserialize($serializado);

echo $deserializado->nombre; // Output: Juan
echo $deserializado->edad; // Output: 30
```

Persistencia con Json encode

La función `json_encode()` en PHP se utiliza para convertir una estructura de datos en formato JSON. JSON es un formato de intercambio de datos ampliamente utilizado que es legible tanto para humanos como para máquinas. Al utilizar `json_encode()`, puedes convertir objetos, matrices y otros tipos de datos en una cadena JSON que se puede almacenar o transmitir.

Aquí tienes un ejemplo de cómo convertir un objeto en formato JSON y viceversa:

```
class Producto {
  public $nombre;
  public $precio;

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

$producto = new Producto("Camiseta", 20);

// Convertir el objeto a JSON
```

```php
$json = json_encode($producto);

echo $json; // Output: {"nombre":"Camiseta","precio":20}

// Convertir el JSON a objeto
$objeto = json_decode($json);

echo $objeto->nombre; // Output: Camiseta
echo $objeto->precio; // Output: 20
```

▼ Interfaces, traits

▼ Forms

¿Qué validar?

- Información requerida
- Formato correcto
- Campos de confirmación (ejemplo: password y reescribir password)
- Datos relacionados (ejemplo: fecha de vuelta posterior a fecha ida)

¿Qué hacer con los datos?

- No perder información!! Ninguna!!!
- Dar información sobre los problemas encontrados

Buenas prácticas

- Indicar ayudas sobre la información (texto o tooltip cuando estés encima)
- Expresiones regulares para formato

Otros

- Captcha

```
if the user submited the form
        if there are form errors
            fill errors array
        else
            record data to database
            Location redirect, required by HTTP standard
            exit()
    if we have some errors
        display errors
        fill form field values
    display the form
```

- Inicializas
- Extraes info y verificas de Post
- Guardas la info ya sea fichero o prepare sql
- header location
- al pintar también saneas
- Diferencia en el manejo de forms:

```php
//Más segura, no pueden inyectar código si han llegado a través de url (get).
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $nombre = $_POST["nombre"];
}
```

```
//Mal
if (isset($_POST["submit"])) { //si la var de enviar se ha seteado.
}  //esto solo con get.
```

Ejemplo y security:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $nombre = $_POST["nombre"];
}
//Más seguro para evitar Crossite Scripting:
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $nombre = htmlspecialchars($_POST["nombre"]);

    header("Location: ../index.php"); //te reenvía a donde le digas.
  die();
} else {
        header("Location: ../index.php"); //Si llega de otra manera le sacas igual.
    die();
}
//El posible código malicioso ya no funciona pues
//los carácteres especiales se tratan como html.
```

Dos maneras de hacerlo creo:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
//Jorge's way:
    if(isset($_POST["nombre"]) && $_POST["nombre"] != ""){
        $nombre = htmlspecialchars($_POST["nombre"]);
    } else {
            $mensajesError['nombre'] = "mensaje error con el nombre";
        }
//Dani's way:
    $nombre = htmlspecialchars($_POST["nombre"]);
    if (empty($nombre)){
        header("Location: ../index.php");
        //error message.
        $mensajesError['nombre'] = "mensaje error con el nombre";
        exit();
    }

    header("Location: ../index.php");
} else {
        header("Location: ../index.php");
}
```

Manera de gestionar persistencia de datos los checkbox (que sigan checkeados):

```
<?php if (isset($errors['lenguajes'])) ?>
    **for error in errors['lenguajes']
            **p error
<?php endif?>
<label for="lenguajes"> Lenguajes(ej)</label>
<input type="checkbox" name="lenguajes" value="Python" <?= in_array('Python', $lenguaje
s) ? 'checked' : '' ?> >Python
<input type="checkbox" name="lenguajes" value="PHP" <?= in_array('PHP', $lenguajes) ?
'checked' : '' ?> >PHP
```

* en la página que proceses el form sería un array la entrada del POST['lenguajes'] primero lo declaras vacío $lenguajes = []; y luego lo llenas al procesar el form.*

▼ BBDD

▼ General

https://github.com/JorgeDuenasLerin/docencia-23-24/tree/main/dwes/04-acceso-datos

Necesitamos PHP y la librería que tenga el protocolo del motor de BD que hayamos elegido:

- Postgress
- Maria BD
- MySQL
- Oracle

Para conectar con la base de datos se necesita un user y pswd, que es para la BD, no tiene que ver con nuestros users.

PDO (ya no se usa mysqli) es una manera de conectarse

▼ Proceso

sudo su apt install mariadb-server

apt update

apt-install php-mysql

mysql_secure_installation (seguridad blabla)

mysql

show databases;

```
create Database namedb;
create user 'username'@'localhost' identified by 'pswd';
grant all on namedb.* TO 'username'@'localhost';
flush privileges;
```
(creo que no)

Terminal (no root-server):

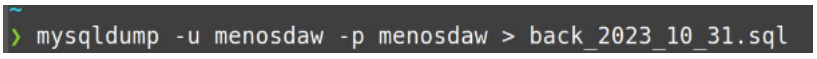
`mysql -u username -p namedb` (conectas)

create etc

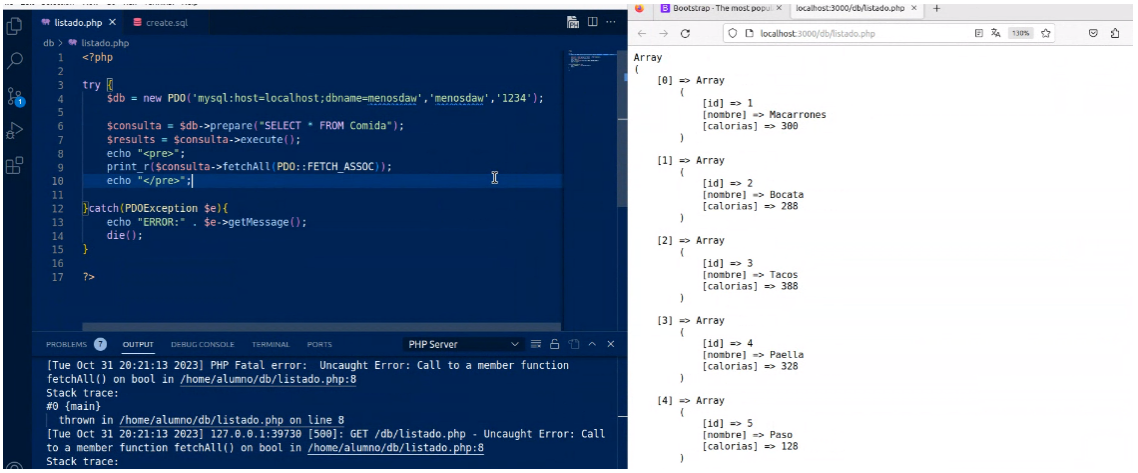
show tables;

less mycnf para ver el puerto dentro de la captura anterior, el puerto al que se conecta la base de datos

Backup:



- `mysqldump`: Es una utilidad de línea de comandos proporcionada por MySQL que se utiliza para generar una copia de seguridad de las bases de datos MySQL en forma de un archivo de volcado SQL.
- `u menosdaw`: La opción `u` se utiliza para especificar el nombre de usuario que se utilizará para conectarse a la base de datos MySQL. En este caso, el nombre de usuario es `menosdaw`.
- `p`: Esta opción indica a `mysqldump` que se requiere una contraseña para conectarse a la base de datos. Cuando se ejecuta el comando, se te pedirá que introduzcas la contraseña.
- `menosdaw`: Este es el nombre de la base de datos de la que quieres hacer una copia de seguridad.
- `> back_2024.sql`: Este es un redireccionamiento de salida en la línea de comandos de Windows. Esto significa que la salida del comando `mysqldump` (es decir, el volcado SQL de la base de datos) se guardará en un archivo llamado `back_2024.sql`.




```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <table>
        <tr>
            <th>Id</th>
            <th>Nombre</th>
            <th>Calorias</th>
        </tr>
        <?php foreach($datos as $dato) { ?>
        <tr>
            <td><?=$dato['id']?></td>
            <td><?=$dato['nombre']?></td>
            <td><?=$dato['calorias']?></td>
        </tr>
        <?php } ?>
    </table>
</body>
</html>
```

cuidado con las SQLinjection el orderby sin $ para que no nos metan ; DROP DATABASE o cosis peligrosis (ORDER BY :orderby)

```php
<?php

$orderby = 1;

try {
    $db = new PDO('mysql:host=localhost;dbname=menosdaw','menosdaw','1234');

    $consulta = $db->prepare("SELECT id, nombre, calorias FROM Comida ORDER BY :orderby DESC"); // SQLi

    $consulta->bindParam(':orderby', $orderby, PDO::PARAM_INT);

    $results = $consulta->execute();

    //echo "<pre>";
    //print_r($consulta->fetchAll(PDO::FETCH_ASSOC));
    //echo "</pre>";
    $datos = $consulta->fetchAll(PDO::FETCH_ASSOC);

}catch(PDOException $e){
    echo "ERROR:" . $e->getMessage();
```

error sql injection:

```php
$nombre = $_POST['nombre'];
$ciudad = $_POST['ciudad'];

$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES ($nombre, $ciudad)");
```

cambiar el $nombre abajo por :nombre etc.

```
'asd','asd'); DROP DATABASE;--

$nombre = $_POST['nombre'];
$ciudad = $_POST['ciudad'];


$stmt = $dbh->prepare("INSERT INTO Clientes (nombre, ciudad) VALUES ('asd','asd'); DROP DATABASE;--, $ciudad)");
```

listado.php

```
<?php

if(isset($_GET['orderby']) && is_numeric($_GET['orderby'])){
    $orderby = $_GET['orderby'];
} else {
    // LOGGEAR
    $orderby = 1;
}

echo $orderby;


$order = 'ASC';

try {
    $db = new PDO('mysql:host=localhost;dbname=menosdaw','menosdaw','1234');

    $consulta = $db->prepare("SELECT id, nombre, calorias FROM Comida ORDER BY :orderby $order"); // SQLi

    $consulta->bindParam(':orderby', $orderby, PDO::PARAM_INT);
```

LEMS OUTPUT DEBUG CONSOLE TERMINAL

```
if(isset($_GET['order'])){
    if($_GET['order']=="ASC"){
        $order = 'ASC';
    }else{
        $order = 'DESC';
    }
} else {
    $order = 'ASC';
}
```

Para que cada vez que pinches, cambie entre orden asc y desc.

```
<body>
    <div class="contenido">
    <table>
        <tr>
            <th><a href="<?=generateQueryString(1, $orderby, $order)?>">Id</a></th>
            <th><a href="<?=generateQueryString(2, $orderby, $order)?>">Nombre</a></th>
            <th><a href="<?=generateQueryString(3, $orderby, $order)?>">Calorias</a></th>
        </tr>
        <?php foreach($datos as $dato) { ?>
        <tr>
            <td><?=$dato['id']?></td>
            <td><?=$dato['nombre']?></td>
            <td><?=$dato['calorias']?></td>
        </tr>
        <?php } ?>
    </table>
    </div>
    <div class="paginacion">
        <?php for($i=1;$i<=$num_pages;$i++) { ?>
            <span><a <?=($i==$page)?"class='actual'":""?> href="?page=<?=$i?>"><?=$i?></a></span>
        <?php } ?>
    </div>
</body>
```

▼ PDO creation

You can either use the options as a parameter while creating the PDO (it should be in a try catch block).

Or use setAttribute after the creation like this: ($this→db = el PDO recién creado).

```
$this->db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

```
<?php
$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
        PDO::ATTR_PERSISTENT => true,
];
$dbh = new PDO($dsn, $username, $password, $options);
?>
```

- `PDO::ATTR_ERRMODE` is set to `PDO::ERRMODE_EXCEPTION` to throw exceptions when an error occurs. This is similar to calling `$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)` after the PDO object is created.
- `PDO::ATTR_DEFAULT_FETCH_MODE` is set to `PDO::FETCH_ASSOC` to return result sets as associative arrays.
- `PDO::ATTR_EMULATE_PREPARES` is set to `false` to use the database's native server-side prepared statements rather than emulating them in PHP.
- `PDO::ATTR_PERSISTENT` option is set to `true`, which tells PDO to use a persistent connection. This connection will not be closed when the script ends, and will be reused on subsequent requests.

▼ Methods

In PHP's PDO (PHP Data Objects) extension, `query()`, `execute()`, and `fetch()` are methods used to interact with a database. Here's what each one does:

1. **query()**: This method is used to execute an SQL statement directly and return a result set as a PDOStatement object. It's most useful when you need to execute a simple query that doesn't include variable data. For example:

```
<?php
$result = $pdo->query("SELECT * FROM table");
?>
```

1. **execute()**: This method is used to execute a prepared statement. Prepared statements are used when you have variable data that you need to include in your query. The data is sent to the database separately from the query, which helps prevent SQL injection attacks. Here's an example of how to use it:

```
<?php
$stmt = $pdo->prepare("INSERT INTO table (column) VALUES (:value)");
$stmt->bindParam(":value", $value, PDO::PARAM_INT);
$resultado = $stmt->execute();
// otra manera peor: $stmt->execute([':value' => $value]);
?>
```

   In this example, `:value` is a placeholder that gets replaced with the actual value when the statement is executed.

2. **fetch()**: This method is used to fetch the next row from a result set. This is typically used after calling `query()` or `execute()`. There are several fetch styles you can use to control how the result is returned, such as `PDO::FETCH_ASSOC` (return result as an associative array), `PDO::FETCH_NUM` (return result as a numeric array), or `PDO::FETCH_OBJ` (return result as an object). Here's an example:

```php
<?php
$stmt = $pdo->query("SELECT * FROM table");
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    // process $row
}
?>
```

▼ Código

```php
<?php
//Create connection
$db = new PDO('mysql:host=localhost;dbname=DATABASENAME;charset=utf8mb4', 'USERNAME', '
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);//https://www.php.net/manu

//Check connection status
echo $db->getAttribute(PDO::ATTR_CONNECTION_STATUS);

//SELECT with loop
$select = $db->prepare("SELECT `col`, `col2` FROM `table`");
$select->execute();
while ($row = $select->fetch(PDO::FETCH_ASSOC)) {
    $db_col = $row['col'];
    $db_col2 = $row['col2'];
    echo "$db_col $db_col2<br>";
}

//SELECT with loop AND WHERE
$select = $db->prepare("SELECT `name` FROM `table` WHERE `status` = :status;");
$select->execute(array(':status' => $status));
while ($row = $select->fetch(PDO::FETCH_ASSOC)) {
    $db_name = $row['name'];
    echo "$db_name<br>";
}

//SELECT with WHERE one value
$uid = 1610;
$select = $db->prepare("SELECT `col`, `col2`, `col3` FROM `table` WHERE `uid` = :uid LI
$select->execute(array(':uid' => $uid));
$row = $select->fetchAll(PDO::FETCH_ASSOC);
$col = $row[0]['col'];

//SELECT one row
$select = $db->prepare("SELECT `col` FROM `table` WHERE `id` = :the_id LIMIT 1;");
$select->execute(array(':the_id' => $id));
$row = $select->fetch();
$id = $row['id'];

//SELECT with WHERE shorter
$select = $db->prepare("SELECT `col`, `col2` FROM `table` WHERE `id` = :id;");
$row = $select->fetch($select->execute(array(':id' => $id)));

//fetch for one/first row fetachAll for many/all rows

//SELECT simple
$select = $db->prepare("SELECT `col`, `col2`, `col3` FROM `table`");
$select->execute();
$row = $select->fetch();
```

```
$db_col = $row['col'];
$db_col2 = $row['col2'];
$db_col3 = $row['col3'];

//SELECT with WHERE
$select = $db->prepare("SELECT `col`, `col2`, `col3` FROM `table` WHERE `id` = :id");
$select->execute(array(':id' => $value));
$row = $select->fetch();
$db_col = $row['col'];
$db_col2 = $row['col2'];

//Bind Param
$select = $db->prepare("SELECT `col` FROM `table` WHERE `id` = :id AND `first_name` = :
$select->bindParam(':id', $id, PDO::PARAM_INT);//https://www.php.net/manual/en/pdo.cons
$select->bindParam(':fname', $first_name, PDO::PARAM_STR);
$select->execute();

//Count returned columns
$select = $db->prepare("SELECT `col` FROM `table` WHERE `col` = :id");
$select->execute(array(':id' => $id));
$column_count = $select->columnCount();//Column count

//Check if row exists
$select = $db->prepare("SELECT `col` FROM `table` WHERE `col` = :id");
$select->execute(array(':id' => $id));
$result = $select->fetchColumn();
if ($result > 0) {
    echo "Row has been found";
} else {
    echo "No row in DB";
}

//Get last inserted id
$id = $db->lastInsertId();

//INSERT
$insert = $db->prepare('INSERT INTO `table` (`col`, `col2`) VALUES (?, ?)');
$insert->execute(["$value", "$value2"]);

$insert = $db->prepare('INSERT INTO `table` (`col`, `col2`, `col3`, `col4`, `col5`, `co
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)');
$insert->execute(["$value", "$value2", "$value3", "$value4", "$value5", "$value6", "$va

$insert = $link->prepare('INSERT INTO `table` (`col`, `col2`, `col3`)
    VALUES (:avalue, :avalue2, :avalue3)');
$insert->execute([
    'avalue' => '1',
    'avalue2' => $value2,
    'avalue3' => $value3,
]);

//INSERT shorter form
$data = [];//an array
$db->beginTransaction();
$stmt = $db->prepare("INSERT IGNORE INTO `table` (`col`, `col2`, `col3`) VALUES (?,?,?)
for ($i = 0; $i < count($data); $i++) {
    $stmt->execute($data[$i]);
}
```

```
$db->commit();

//INSERT with UPDATE on duplicate
$query = $db->prepare('INSERT INTO table (col, col2, col3) VALUES(:var, :var2, :var3)
    ON DUPLICATE KEY UPDATE col = :var, col2 = :var2, col3 = :var3');
$query->bindParam(':var', $avar, PDO::PARAM_INT);
$query->bindParam(':var2', $avar2, PDO::PARAM_STR);
$query->bindParam(':var3', $avar3, PDO::PARAM_STR);
$query->execute();

//UPDATE on WHERE
$update = $db->prepare("UPDATE `table` SET `col` = 1 WHERE `col2` = :the_value");
$update->execute(array(':the_value' => $value));

//UPDATE on WHERE 2
$update = $db->prepare("UPDATE `table` SET `col` = :update_to WHERE `col2` = :the_value
$update->execute(array(':update_to' => $update_to, ':the_value' => $value));

//UPDATE
$update = $db->prepare("UPDATE `table` SET `col` = 1");
$update->execute();

//DELETE
$delete = $db->prepare("DELETE FROM `table` WHERE `col2` = :the_value");
$delete->execute(array(':the_value' => $value));
```

▼ Singleton

The active selection is a PHP class named `DbConnection` that uses the Singleton design pattern to manage a database connection. The Singleton pattern ensures that only one instance of this class can exist, which can be useful for things like database connections where you typically only want one connection to be active at a time.

Here's a breakdown of the class:

- The `DbConnection` class has four private properties: `$instance`, `$db`, `CONFIG_FILE`, and `$config`.
- `$instance` is a static property that will hold the single instance of the `DbConnection` class.
- `$db` is the PDO object that represents the database connection.
- `CONFIG_FILE` is a constant that holds the path to the JSON configuration file.
- `$config` is an array that will hold the database configuration details loaded from the JSON file.

The `__construct` method is private to prevent creating new instances outside of the class. This method loads the database configuration from a JSON file, creates a new PDO object, and sets the error mode to `ERRMODE_EXCEPTION` so that PDO will throw exceptions on error.

The `getInstance` method is a public static method that returns the single instance of the `DbConnection` class. If the instance doesn't exist, it creates one by calling `new self()`.

The `loadConfig` method is a private method that loads the database configuration from a JSON file. It uses `file_get_contents` to read the file and `json_decode` to convert the JSON string into an associative array. If the configuration is not loaded correctly, it throws a `JsonException`.

Please note that there's a logical error in the `loadConfig` method. The check for `$this->config === null` should be done before trying to load the configuration, not after. The corrected method should look like this:

```
<?php
private function loadConfig()
{
    if (file_exists(self::CONFIG_FILE)) {
        $this->config = json_decode(file_get_contents(self::CONFIG_FILE), true);
    } else {
        throw new JsonException("Error loading the JSON configuration file.");
```

```
    }
    return $this->config;
}
?>
```

▼ Persistent Connections

Persistent connections in PHP are connections that do not close when the execution of your script ends. When a persistent connection is requested, PHP checks if there's already an identical persistent connection (that was left open previously) - and if it exists, it uses it. If it does not exist, it creates the link. An 'identical' connection is a connection that was opened to the same host, with the same username and the same password (where applicable).

The primary benefit of persistent connections is performance. It's relatively expensive to establish a database connection; with a persistent connection, the overhead of establishing the connection is incurred only once: the first time the connection is established. For subsequent requests, the already-open connection is used, which can lead to significant performance gains, especially for applications that make frequent database connections.

However, persistent connections come with their own set of challenges:

1. **Connection Limits**: Most database systems have a limit on the number of concurrent connections they can handle. If you have a lot of scripts using persistent connections but not a lot of concurrent users, you can run out of connections, because PHP will not close the connections to the database when the scripts end.
2. **Stale Connections**: A persistent connection might become "stale". This can happen if the database server has gone away, or if there's a network issue. The PHP script might fail if it tries to use a stale connection.
3. **Uncommitted Transactions**: If a script ends without committing a transaction, the uncommitted transaction can linger and hold locks open, causing other scripts to hang.

Because of these challenges, it's important to use persistent connections judiciously. They can be very beneficial for high-load systems where the overhead of connecting is causing performance problems, but for many applications, the potential issues outweigh the benefits.

▼ Prepared Statements

▼ BindParam vs BindValue

Yes, there is a difference between `bindParam` and `bindValue` in PHP's PDO extension:

1. **bindParam**: When you use `bindParam`, it binds a PHP variable to the corresponding placeholder in the SQL statement. This means that it binds the variable by reference. The value of the variable at the time of the `execute()` call will be used. If the variable changes after `bindParam` but before `execute`, the value that is used is the changed value.

```
<?php

$variable = 'value1';

$stmt->bindParam(':placeholder', $variable, PDO::PARAM_STR);

$variable = 'value2';

$stmt->execute(); // The query will be executed with 'value2'

?>
```

1. **bindValue**: On the other hand, `bindValue` binds a value to a placeholder. This means it binds the value as it is at the time of the `bindValue` call. If the variable changes after `bindValue` but before `execute`, the value that is used is the original value.

```
<?php

$variable = 'value1';
```

```
$stmt->bindValue(':placeholder', $variable, PDO::PARAM_STR);

$variable = 'value2';

$stmt->execute(); // The query will be executed with 'value1'

?>
```

In most cases, you can use either `bindParam` or `bindValue` and get the same results. The difference only matters if you're changing the value of the variable between the `bindParam` / `bindValue` call and the `execute` call.

▼ `PDO::ATTR_EMULATE_PREPARES`

When `PDO::ATTR_EMULATE_PREPARES` is set to `false`, the MySQL server will prepare the statement itself, and the statement will be sent to the server separately from the data. This is known as true prepared statements. It has several benefits:

- **Security**: True prepared statements are more secure against SQL injection attacks because the data is sent separately from the query itself, so there's no risk of the data being interpreted as part of the SQL command.
- **Performance**: If you're executing the same query multiple times with different data, true prepared statements can be more efficient because the server only needs to parse and optimize the query once.

When `PDO::ATTR_EMULATE_PREPARES` is set to `true`, PDO will emulate prepared statements in PHP. The query will be parsed and the data will be bound in PHP, and then the complete query will be sent to the server. This has a few potential advantages:

- **Compatibility**: Emulated prepared statements can be used with older versions of MySQL that don't support true prepared statements.
- **Flexibility**: Emulated prepared statements allow you to use certain features that aren't available with true prepared statements, like the ability to bind multiple values to a single parameter.

In general, it's recommended to use true prepared statements (`PDO::ATTR_EMULATE_PREPARES => false`) unless you have a specific reason to do otherwise, because they provide better security and can potentially offer better performance.

▼ Cómo evitar usar el BindBlabla

▼ Close connection?

Whether to close the database connection or not depends on the specifics of your application and its usage patterns. Here are some considerations:

1. **Performance**: Opening a database connection is a relatively expensive operation. If your application frequently reads from or writes to the database, it might be more efficient to keep the connection open for the duration of the session, rather than opening and closing it each time.
2. **Resource Usage**: Each open connection consumes resources on the database server. If you have many concurrent users, this could potentially exhaust the server's resources. In this case, it might be better to close connections as soon as you're done with them.
3. **Connection Pooling**: Many web hosting environments use connection pooling, which keeps a set of database connections open and reuses them as needed. This can give you the performance benefits of keeping connections open, without the resource usage drawbacks. If your environment supports connection pooling, you don't need to worry about closing connections manually.
4. **Security**: Keeping a database connection open could potentially increase the risk of unauthorized access, especially if your application is vulnerable to SQL injection attacks. However, this risk is relatively small compared to the risk of not properly sanitizing user input or not using prepared statements.

In your case, since you're using PHP, which doesn't have long-lived sessions like some other server-side languages (e.g., Node.js, Java), the connection is automatically closed at the end of the script execution, so you don't need to close it manually. If you're using persistent connections (`PDO::ATTR_PERSISTENT`), then the connection will be kept open across multiple requests by the same client, which can improve performance.

▼ Seguridad

▼ Tipos de ataque

▼ SQL injection

- Description

- How to avoid it

1. **Use Prepared Statements**: Prepared statements ensure that an attacker is not able to change the intent of a query, even if they inject malicious SQL. You can use prepared statements in PHP with either the MySQLi or PDO extensions. In your code, you're already using PDO prepared statements, which is a good practice.
2. **Use Parameterized Queries**: Parameterized queries are a type of prepared statement, but they take it a step further by ensuring that the parameters (values) are separated from the query command code. This makes it impossible for an attacker to inject malicious SQL.
3. **Escaping All User Supplied Input**: If you can't use prepared statements for some reason (like when you need to use a function that doesn't support them), you can use the `addslashes()` function in PHP or the `real_escape_string()` function in MySQLi. These functions make certain characters safe in SQL queries.
4. **Limiting Privileges**: Don't use the root database user in your application. Instead, create a user in the database for your application and only give this user the necessary privileges.
5. **Input Validation**: Validate and sanitize user input to ensure it conforms to expected formats, using functions like `filter_var()`.

6. **Use Web Application Firewall (WAF)**: A WAF can help to block SQL injection attacks by identifying and blocking malicious queries.
7. **Keep Software Up to Date**: Ensure that PHP and all extensions are kept up to date. Updates often contain security enhancements and patches for vulnerabilities.
8. **Error Reporting**: In a production environment, ensure that error reporting is turned off to prevent any database errors from being displayed to the end user. Such errors can provide useful information to an attacker.

Remember, security is about layers, and no single approach is a silver bullet. Using these techniques together will provide robust protection against SQL injection attacks.

▼ XSS Cross site scripting

- Description

- How to avoid it

▼ CSRF Cross site forging

- Description

  Cross-Site Request Forgery (CSRF) is a type of attack that tricks the victim into submitting a malicious request. It uses the identity and privileges of the victim to perform an undesired function on their behalf. For most sites, browser requests automatically include any credentials associated with the site, such as the user's session cookie, IP address, etc. Therefore, if the user is authenticated, the site cannot distinguish between legitimate requests and forged requests.

- How to avoid it

  To prevent CSRF attacks in your PHP application, you can use anti-CSRF tokens. The general idea is to generate a unique token when the user's session starts, and for every form submission, you include this token. When processing the form, you check if the token is valid.

  Here's a simple example:

```
<?php
session_start();

// Generate a CSRF token if one doesn't exist
if (!isset($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = base64_encode(openssl_random_pseudo_bytes(32));
}

// Check a POST for a valid CSRF token
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) {
        // Invalid CSRF token
        die('Invalid CSRF token');
    }
}

// Now you can generate a form with the CSRF token
?>
<form method="POST" action="your_script.php">
    <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_t
oken']; ?>">
    <!-- Rest of your form fields -->
</form>
```

This code first checks if a CSRF token is already set in the session. If not, it generates a new one. When a POST request is made, it checks if the CSRF token in the session matches the one in the POST data. If they don't match, it stops the script execution.

jS:

In a JavaScript-based application, you can implement CSRF protection using the same concept of anti-CSRF tokens as in PHP. However, the implementation will be slightly different. Here's a simple example using jQuery for AJAX requests:

```
// Assume you have the CSRF token in a meta tag
var csrfToken = document.querySelector('meta[name="csrf-token"]').getAttribute
('content');

$.ajaxSetup({
    beforeSend: function(xhr, settings) {
        if (!/^(GET|HEAD|OPTIONS|TRACE)$/i.test(settings.type) && !this.crossD
omain) {
            xhr.setRequestHeader("X-CSRF-Token", csrfToken);
        }
    }
});
```

In this example, the CSRF token is stored in a meta tag. Before sending an AJAX request, jQuery's `ajaxSetup` method is used to set the `X-CSRF-Token` header to the CSRF token for non-GET requests. On the server side, you would then validate this token to ensure the request is legitimate.

Remember to regenerate the CSRF token on the server side after each request to ensure it remains secure. Also, ensure that the CSRF token is tied to the user's session and not globally scoped.

▼ Phising

- Description
- How to avoid it

▼ DDOS attack

- Description
- How to avoid it

▼ Dotenv

▼ Difference with loadConfig()

While adding `db.json` to `.gitignore` will prevent the file from being committed to your repository, it's not quite as safe as using environment variables with `phpdotenv`.

The main reason is that environment variables are designed to be per-environment, meaning each developer or each deployment environment (like staging, production, etc.) can have different values. This is especially important for things like database credentials, API keys, and other sensitive information.

On the other hand, a `db.json` file is typically shared among all developers and environments. If you add it to `.gitignore`, each developer or environment would need to manually create this file, which can lead to inconsistencies and potential errors.

So while both methods can keep sensitive data out of your repository, using environment variables with `phpdotenv` is generally considered a better practice for managing configuration and sensitive data.

To implement `phpdotenv` in your project, follow these steps:

1. **Install phpdotenv**: First, you need to install the `phpdotenv` package using Composer. If you don't have Composer installed, you'll need to install it first. Once you have Composer, you can install `phpdotenv` by running the following command in your terminal:

```
composer require vlucas/phpdotenv
```

1. **Create .env file**: Create a new file in the root of your project named `.env`. This is where you'll store your environment variables. Based on your `db.json` file, your `.env` file would look something like this:

```
DB_HOST=localhost
DB_USERNAME=dbuser* acá datos reales etc
DB_PASSWORD=password
DB_DATABASE=dbname
DB_DIALECT=mysql
DB_SEEDER_STORAGE=sequelize
DB_CHARSET=utf8mb4
```

1. **Load .env file**: In your PHP script, you'll need to load the `.env` file using `phpdotenv`. Here's how you can do it:

```
<?php
require_once __DIR__ . '/vendor/autoload.php'; // Path may vary based on your pro
ject structure

$dotenv = Dotenv\\Dotenv::createImmutable(__DIR__);
$dotenv->load();
```

1. **Use Environment Variables**: Now you can use the environment variables in your PHP script like this:

```
<?php
$dbHost = $_ENV['DB_HOST'];
$dbUsername = $_ENV['DB_USERNAME'];
$dbPassword = $_ENV['DB_PASSWORD'];
$dbDatabase = $_ENV['DB_DATABASE'];
$dbDialect = $_ENV['DB_DIALECT'];
$dbSeederStorage = $_ENV['DB_SEEDER_STORAGE'];
$dbCharset = $_ENV['DB_CHARSET'];
```

1. **Ignore .env file in version control**: Add `.env` to your `.gitignore` file to ensure it's not committed to your version control system. This is because `.env` files should not be shared - each developer or environment should have its own.

That's it! You've now implemented `phpdotenv` in your project. Remember to never store sensitive data in your code or version control system. Always use environment variables for that.

While the `.env` file itself should not be committed to version control, you can (and should) provide a `.env.example` file in your repository. This file should have all the necessary environment variables listed, but with dummy values or placeholders.

Here's an example:

```
DB_HOST=your_database_host
DB_USERNAME=your_database_username
DB_PASSWORD=your_database_password
DB_DATABASE=your_database_name
```

```
DB_DIALECT=your_database_dialect
DB_SEEDER_STORAGE=your_database_seeder_storage
DB_CHARSET=your_database_charset
```

When a new developer clones the repository or when deploying to a new environment, they would copy the `.env.example` file to a new file named `.env` and replace the placeholders with the actual values for their environment.

This way, you're not exposing sensitive data in your version control, but you're still providing the necessary information for someone else to set up the project.

▼ HTTP

El hhtp cada petición es independiente, y no hay estado.

Se inventó una cabecera cookie. $_COOKIE[];

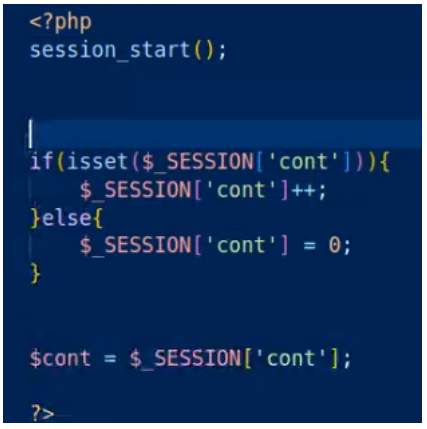
session_start(); al cargar la app, el server de PHP nos manda una cookie con un número aleatorio. y el $_SESSION se rellena con lo último que guardes en la sesión (el servidor).

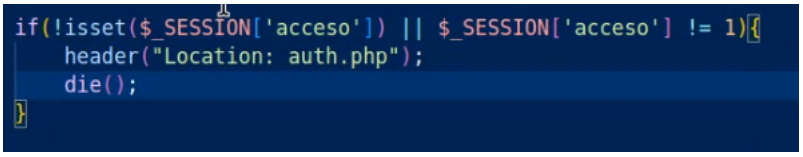
Y la idea es que le dices como server: $_SESSION[’id’]; le guardas la cookie random que genera el cliente.

Cuando te autentificas, te metes en la tabla de

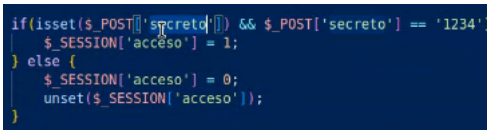
El token de autentificación es una movida de criptografía. No vamos a ver ahora el recuerdáme.

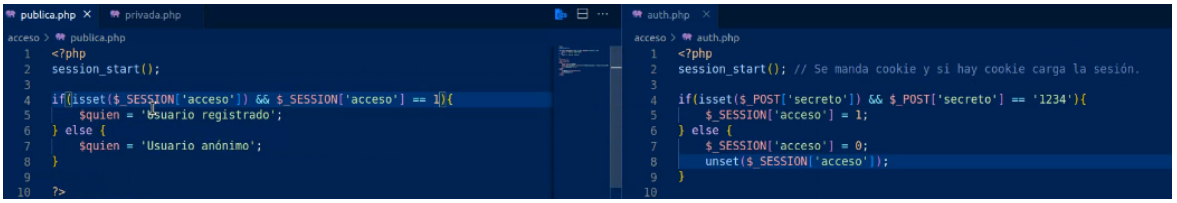
Generar en la tabla de token un token con expiración.

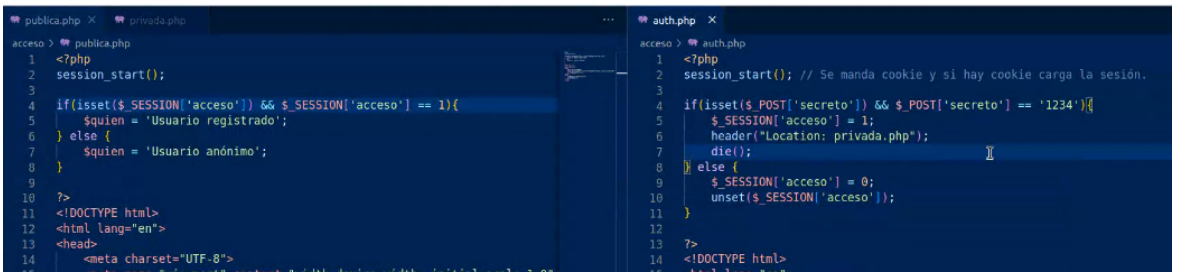




el name=’secreto’ es la contraseña







ifs

redirect header

bd para comprobar

sel us, ps from user where user = :user LIMIT 1 (solo compruebas si hay un user intentando entrar y si existe el user)

luego compruebas y verificas con otra select la password.

formlogin.php clase. que coge, procesa el post, sanea etc

el AuthManager traerusuario(id) verificapass(pass), verificaToken, etc (thisprivate si no, te larga)

▼ Guzzle

Install:

```
# Install Composer
curl -sS https://getcomposer.org/installer | php
```

You can add Guzzle as a dependency using Composer:

```
composer require guzzlehttp/guzzle:^7.0
```

Alternatively, you can specify Guzzle as a dependency in your project's existing composer.json file:

```
{
  "require": {
    "guzzlehttp/guzzle": "^7.0"
  }
}
```

After installing, you need to require Composer's autoloader:

```
require 'vendor/autoload.php';
```

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

https://docs.guzzlephp.org/en/stable/

Guzzle is a PHP HTTP client that makes it easy to send HTTP requests and trivial to integrate with web services.

- Simple interface for building query strings, POST requests, streaming large uploads, streaming large downloads, using HTTP cookies, uploading JSON data, etc...
- Can send both synchronous and asynchronous requests using the same interface.

- Uses PSR-7 interfaces for requests, responses, and streams. This allows you to utilize other PSR-7 compatible libraries with Guzzle.
- Abstracts away the underlying HTTP transport, allowing you to write environment and transport agnostic code; i.e., no hard dependency on cURL, PHP streams, sockets, or non-blocking event loops.
- Middleware system allows you to augment and compose client behavior.

```
$client = new GuzzleHttp\Client();
$res
= $client -> request('GET', 'https://api.github.com/user', [
    'auth'
=> ['user', 'pass']
]);

echo $res -> getStatusCode();

// "200" echo $res -> getHeader('content-type')[0];

// 'application/json; charset=utf8' echo $res -> getBody();

// {"type":"User"...' // Send an asynchronous request. $request = new \GuzzleHttp\Psr7\Request('GET', 'http://httpbin.org');
$promise
= $client -> sendAsync($request) -> then( function ($response) {

echo 'I completed! ' . $response -> getBody();
});
$promise
-> wait();
```

Making a Request

You can send requests with Guzzle using a `GuzzleHttp\ClientInterface` object.

Creating a Client

```
use GuzzleHttp\Client;

$client
= new Client([

// Base URI is used with relative requests 'base_uri' => 'http://httpbin.org',

// You can set any number of default request options. 'timeout' => 2.0,
]);
```

Clients are immutable in Guzzle, which means that you cannot change the defaults used by a client after it's created.

The client constructor accepts an associative array of options:

`base_uri` (string|UriInterface) Base URI of the client that is merged into relative URIs. Can be a string or instance of UriInterface. When a relative URI is provided to a client, the client will combine the base URI with the relative URI using the rules described in RFC 3986, section 5.2.

```
// Create a client with a base URI $client = new GuzzleHttp\Client(['base_uri' => 'https://foo.com/api/']);

// Send a request to https://foo.com/api/test $response = $client -> request('GET', 'test');

// Send a request to https://foo.com/root $response = $client -> request('GET', '/root');
```

Don't feel like reading RFC 3986? Here are some quick examples on how a `base_uri` is resolved with another URI.base_uriURIResult `http://foo.com` `/bar` `http://foo.com/bar` `http://foo.com/foo` `/bar` `http://foo.com/bar` `http://foo.com/foo` `bar` `bar` `bar` `http://foo.com/bar` **`handler`** (callable) Function that transfers HTTP requests over the wire. The function is called with a `Psr7\Http\Message\RequestInterface` and array of transfer options, and must return a `GuzzleHttp\Promise\PromiseInterface` that is fulfilled with a `Psr7\Http\Message\ResponseInterface` on success. `...` (mixed) All other options passed to the constructor are used as default request options with every request created by the client.

Sending Requests

Magic methods on the client make it easy to send synchronous requests:

```
$response = $client->get('http://httpbin.org/get');
$response = $client->delete('http://httpbin.org/delete');
$response = $client->head('http://httpbin.org/get');
$response = $client->options('http://httpbin.org/get');
$response = $client->patch('http://httpbin.org/patch');
$response = $client->post('http://httpbin.org/post');
$response = $client->put('http://httpbin.org/put');
```

Using Responses

In the previous examples, we retrieved a `$response` variable or we were delivered a response from a promise. The response object implements a PSR-7 response, `Psr\Http\Message\ResponseInterface`, and contains lots of helpful information.

You can get the status code and reason phrase of the response:

```
$code = $response->getStatusCode(); // 200 $reason = $response->getReasonPhrase(); // OK
```

You can retrieve headers from the response:

```
// Check if a header exists. if ($response->hasHeader('Content-Length')) {

echo "It exists";
}

// Get a header from the response. echo $response->getHeader('Content-Length')[0];

// Get all of the response headers. foreach ($response->getHeaders() as $name => $values) {

echo $name . ': ' . implode(', ', $values) . "\r\n";
}
```

The body of a response can be retrieved using the `getBody` method. The body can be used as a string, cast to a string, or used as a stream like object.

```
$body = $response->getBody();

// Implicitly cast the body to a string and echo it echo $body;

// Explicitly cast the body to a string $stringBody = (string) $body;

// Read 10 bytes from the body $tenBytes = $body->read(10);

// Read the remaining contents of the body as a string $remainingBytes = $body->getContents();
```

Query String Parameters

You can provide query string parameters with a request in several ways.

You can set query string parameters in the request's URI:

```
$response = $client->request('GET', 'http://httpbin.org?foo=bar');
```

You can specify the query string parameters using the `query` request option as an array.

```
$client->request('GET', 'http://httpbin.org', [
    'query' => ['foo' => 'bar']
]);
```

Providing the option as an array will use PHP's `http_build_query` function to format the query string.

And finally, you can provide the `query` request option as a string.

```
$client->request('GET', 'http://httpbin.org', ['query' => 'foo=bar']);
```

Uploading Data

Guzzle provides several methods for uploading data.

You can send requests that contain a stream of data by passing a string, resource returned from `fopen`, or an instance of a `Psr\Http\Message\StreamInterface` to the `body` request option.

```
use GuzzleHttp\Psr7;

// Provide the body as a string.
$r = $client->request('POST', 'http://httpbin.org/post', [
    'body' => 'raw data'
]);

// Provide an fopen resource.
$body = Psr7\Utils::tryFopen('/path/to/file', 'r');
$r = $client->request('POST', 'http://httpbin.org/post', ['body' => $body]);

// Use the Utils::streamFor method to create a PSR-7 stream.
$body = Psr7\Utils::streamFor('hello!');
$r = $client->request('POST', 'http://httpbin.org/post', ['body' => $body]);
```

An easy way to upload JSON data and set the appropriate header is using the `json` request option:

```
$r = $client->request('PUT', 'http://httpbin.org/put', [
    'json' => ['foo' => 'bar']
]);
```

POST/Form Requests

In addition to specifying the raw data of a request using the `body` request option, Guzzle provides helpful abstractions over sending POST data.

Sending form fields

Sending `application/x-www-form-urlencoded` POST requests requires that you specify the POST fields as an array in the `form_params` request options.

```
$response = $client->request('POST', 'http://httpbin.org/post', [
    'form_params' => [
        'field_name' => 'abc',
        'other_field' => '123',
        'nested_field' => [
            'nested' => 'hello'
        ]
    ]
]);
```

Sending form files

You can send files along with a form (`multipart/form-data` POST requests), using the `multipart` request option. `multipart` accepts an array of associative arrays, where each associative array contains the following keys:

- name: (required, string) key mapping to the form field name.
- contents: (required, mixed) Provide a string to send the contents of the file as a string, provide an fopen resource to stream the contents from a PHP stream, or provide a `Psr\Http\Message\StreamInterface` to stream the contents from a PSR-7 stream.

```
use GuzzleHttp\Psr7;

$response
= $client -> request('POST', 'http://httpbin.org/post', [
    'multipart'
=> [
        [
            'name'
=> 'field_name',
            'contents'
=> 'abc'
        ],
        [
            'name'
=> 'file_name',
            'contents'
=> Psr7\Utils :: tryFopen('/path/to/file', 'r')
        ],
        [
            'name'
=> 'other_file',
            'contents'
=> 'hello',
            'filename'
=> 'filename.txt',
            'headers'
=> [
                'X-Foo'
=> 'this is an extra header to include'
            ]
        ]
    ]
]);
```

▼ Cookies

```
setcookie(
    string $name,
    string $value = "",
    int $expires_or_options = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
): bool
```

https://www.php.net/manual/en/function.setcookie.php

httponly: solo se puede acceder desde http, no desde js con XSS.

secure: solo https

path: si quieres que solo se active en ciertos sitios (páginas) de tu web y parecido a domain.

Importante diferencia entre la cookie actual, y setear o cambiar el número a la cookie.

```
❯ nc localhost:3000
nc: missing port number

~
❯ nc localhost 3000
GET /acceso/cookieplay/index.php HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
Host: localhost
Date: Mon, 11 Dec 2023 17:36:32 GMT
Connection: close
X-Powered-By: PHP/8.1.2-1ubuntu2.14
Set-Cookie: num=1
Content-type: text/html; charset=UTF-8

<pre>Array
(
)
</pre><pre>Array
(
)
</pre>
```

este es el comando, lo otro sale dando a enter

```
❯ nc localhost 3000
GET /acceso/cookieplay/index.php HTTP/1.1
Host: localhost
Cookie: num=42
```

```php
<?php
include('colors.php');

$error = "";

if(isset($_GET['color'])) {
    if(in_array($_GET['color'], $colors)){
        setcookie('color', $_GET['color']);
        header("Location: info.php");
        die();
    } else {
        $error = "Opción inválida";
    }
}
?>
<!DOCTYPE html>
<html lang="en">
```

```php
<body>
    <h1>Página de preferencias</h1>
    <?php if($error!="") {
        echo "<h2>$error</h2>";
    } ?>
    <div>
        <?php foreach ($colors as $color) { ?>
            <a href="pref.php?color=<?=$color?>" style="background-color:<?=$color?>"> </a>
        <?php } ?>
    </div>
</body>
</html>
```

pref.php info.php JS script.js style.css

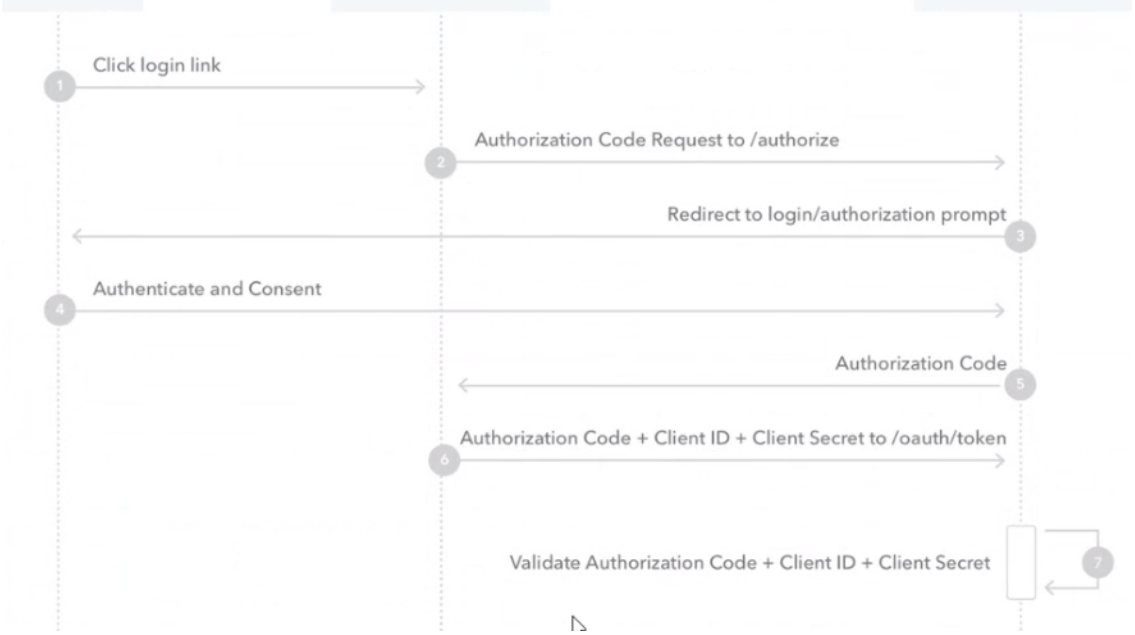
colors > JS script.js > onload > aceptaCookies

```js
// Verifica si la cookie de aceptación está establecida
window.onload = function() {
    var aceptaCookies = document.cookie.split('; ').find(row => row.startsWith('acepta_cookies'));
    console.log(aceptaCookies);
    if (!aceptaCookies) {
        // Si no se han aceptado las cookies, muestra el bloqueo
        document.getElementById('bloqueo').style.display = 'block';
    }
};
```

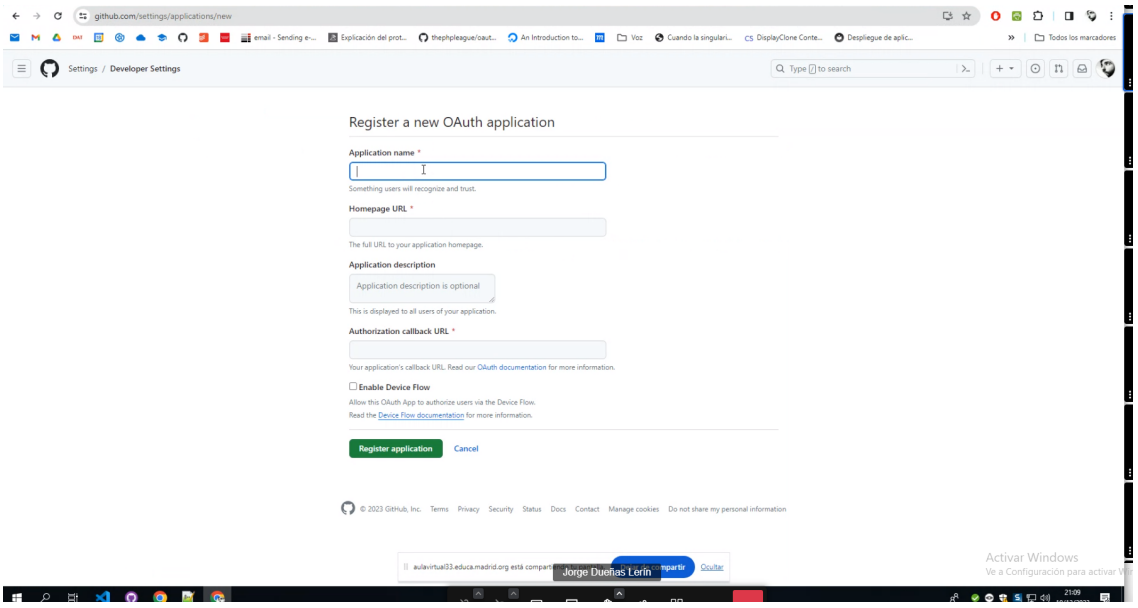
```php
<?php
include_once "colors.php";

$mostrarAviso = true;


if (isset($_COOKIE["color"]) && in_array($_COOKIE["color"], $colors)) {
    $color = $_COOKIE["color"];
    setcookie("color", $_COOKIE["color"], time() + 60 * 60 * 24 * 7);
} else {
    $color = $colors[0];
}

if (isset($_POST["acepta_cookies"])) {
    setcookie("acepta_cookies",  time() + 60 * 60 * 24 * 365);
    $mostrarAviso = false;
    header("Location: info.php");
    exit();
}else if (isset($_COOKIE["acepta_cookies"])) {
    $mostrarAviso = false;
}

?>
<!DOCTYPE html>
```

- ▼ Autentificación
  - ▼ OAuth

    Protocolo de autentificación



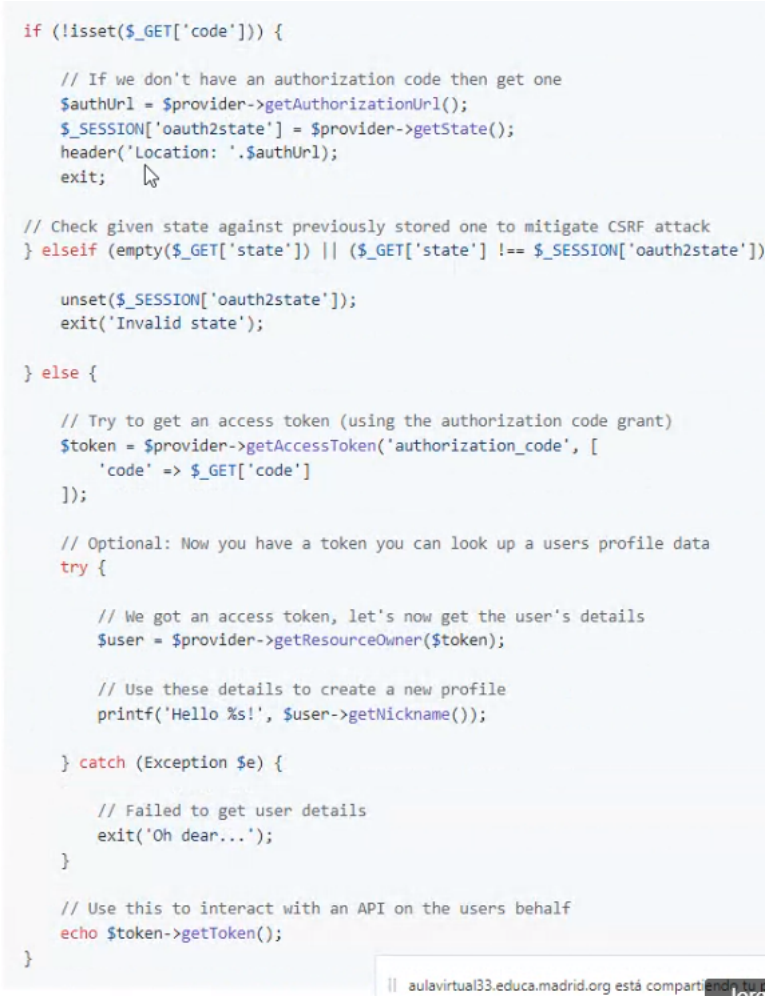
En settings, developer settings




```php
$provider = new \League\OAuth2\Client\Provider\GenericProvider([
    'clientId'                => 'XXXXXX',    // The client ID assigned to you by the pro
    'clientSecret'            => 'XXXXXX',    // The client password assigned to you by t
    'redirectUri'             => 'https://my.example.com/your-redirect-url/',
    'urlAuthorize'            => 'https://service.example.com/authorize',
    'urlAccessToken'          => 'https://service.example.com/token',
    'urlResourceOwnerDetails' => 'https://service.example.com/resource'
]);
```

```php
$provider = new League\OAuth2\Client\Provider\Github([
    'clientId'          => '{github-client-id}',
    'clientSecret'      => '{github-client-secret}',
    'redirectUri'       => 'https://example.com/callback-url',
]);
```

esto lo pegamos tal cual



```php
if (!isset($_GET['code'])) {

    // If we don't have an authorization code then get one
    $authUrl = $provider->getAuthorizationUrl();
    $_SESSION['oauth2state'] = $provider->getState();
    header('Location: '.$authUrl);
    exit;

// Check given state against previously stored one to mitigate CSRF attack
} elseif (empty($_GET['state']) || ($_GET['state'] !== $_SESSION['oauth2state'])) {

    unset($_SESSION['oauth2state']);
    exit('Invalid state');

} else {

    // Try to get an access token (using the authorization code grant)
    $token = $provider->getAccessToken('authorization_code', [
        'code' => $_GET['code']
    ]);

    // Optional: Now you have a token you can look up a users profile data
    try {

        // We got an access token, let's now get the user's details
        $user = $provider->getResourceOwner($token);

        // Use these details to create a new profile
        printf('Hello %s!', $user->getNickname());

    } catch (Exception $e) {

        // Failed to get user details
        exit('Oh dear...');
    }

    // Use this to interact with an API on the users behalf
    echo $token->getToken();
}
```

aulavirtual33.educa.madrid.org está compartiendo tu

1. Te llega el post de la select para la bbdd, con limit 1 por si acaso
2. password_verify
3. ok? $_SESSION[’user’] = $user;

```php
if(!$_SESSION['logIn']){
    //guardo en session adonde intentó entrar para luego añadirle al header
    //pero no al de abajo, sino al de todo ok si se registra bien.
    // y luego vacío ese sesssion
    $_SESSION['redirect'] = "lawebprivada.php";
    header("login.php");
    die();
```

4. Redirección
    a. Si se registra con éxito le rediriges adonde quería

```php
if (isset($_SESSION['redirect'] && (!empty($_SESSION['redirect'])){
    $_SESSION['redirect'] = ""; //o unset?
    header($_SESSION['redirect']); //le meto más cosas en la url el userid ?
}
```

Para ocultar la contraseña al almacenar en la bbdd,

password_hash y password_verify

Si intentas entrar a una pestaña tipo tu user, pero no te has registrado, te pinta

tabla tokens id usuario tipo valor expiración

te olvidaste la contraseña: login.php envía a recupera.php el mail x post

1. Envías el formulario con el $_POST del user (el mail)
2. Compruebas si existe en la bbdd, si no le rediriges para atrás (en ambos casos le dices que se ha enviado el correo y no gestionas el error por no darle esa info)
3. Creas token así si existe el mail:

```php
$token = bin2hex(openssl_random_pseudo_bytes(16));

# or in php7
$token = bin2hex(random_bytes(16));
```

4. Guardas el token en la tabla tokens con usuario, token, tipo y expiración (10 mins)
4. Si sí existe, le envías un email que dirija forgotpass.php?token=valuetoken&id=userid

id user	tipo	valor	expiracy
el user que puso	recovery	375874ytfwygdfua	10 mins

Envía el mail con el enlace a cambiar la contraseña

Cambiar contrasñea: changepass.php?token=$token&idUser=eluserquepuso

1. Recoger token
2. Verificar token
3. Pedir nueva contraseña
4. Envía el post y verificaciones
5. UPDATE pass
6. Delete token (lo hace sino el cron tipo bot que cada hora elimina los que están expirados).
7. Le logeas

1. area privada
2. publica
3. registrarse
4. olvidar contraseña

Cuando te registras con éxito genera un token de recuérdame si has checkeado el recuérdame. Y mete el valor del token en una cookie. También metes el token en la tabla de tokens.

de privada compruebas si hay cookie de token de recuérdame

y haces una select a la tabla de tokens con el id del user, el valor del token de la cookie, verificas que es válido y no ha expirado el token.

le metes todo en session (que se habría borrado en cambio la cookie no)

extiendes la sesión, seteando con más tiempo la cookie, update la tabla de tokens y le rediriges al área privada.