Desarrollo web en entorno servidor.

Unidad 4: Desarrollo de aplicaciones web utilizando código embebido

ÍNDICE

INTRODUCCIÓN....3
OBJETIVOS / CAPACIDADES....4
PROYECTO DE LA UNIDAD....5
1. MANTENIMIENTO DEL ESTADO....8
2. SESIONES....10
Cuestionario....13
3. COOKIES. APLICACIONES PRÁCTICAS. LAS COOKIES Y EL NAVEGADOR....14
4. SEGURIDAD: USUARIOS, PERFILES, ROLES....16
5. AUTENTIFICACIÓN DE USUARIOS. OPENID, OAUTH. ACCESO AL SERVICIO DIRECTORIO LDAP....18
Cuestionario....22
6. HERRAMIENTAS DE PROGRAMACIÓN....23
7. ADAPTACIÓN A APLICACIONES WEB: GESTORES DE CONTENIDOS Y TIENDAS VIRTUALES ENTRE OTRAS....25
8. PRUEBAS Y DEPURACIÓN....27
Cuestionario....30
RESUMEN....31
RECURSOS PARA AMPLIAR....32
BIBLIOGRAFÍA....33
GLOSARIO....34

INTRODUCCIÓN

Con lo que hemos aprendido hasta ahora ya podemos desarrollar aplicaciones más o menos complejas, pero en un entorno web, donde manda la **premisa request-response**, nos será útil que el servidor nos recuerde como clientes.

Seguro que conceptos como **cookies** o **sesiones** no te serán desconocidos porque los usas como cliente a diario. A partir de ahora los conocerás también desde el lado del desarrollo.

Para ello, en esta unidad didáctica veremos mecanismos para el **mantenimiento del estado**. Y más allá de estos mecanismos, también veremos métodos para **autenticarnos** en nuestras aplicaciones, ya sea de forma manual o con herramientas de terceros.

Como ya estamos **programando a un nivel muy complejo** es posible que se produzcan errores en nuestro código. En esta unidad veremos también algunas **herramientas** que nos ayudarán a **depurarlos**.

OBJETIVOS / CAPACIDADES

En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:

- ✔ **Utilizar lenguajes, objetos y herramientas, interpretando las especificaciones para desarrollar aplicaciones web con acceso a bases de datos.**
- ✔ **Utilizar herramientas y lenguajes específicos, cumpliendo las especificaciones, para desarrollar e integrar componentes software en el entorno del servidor web.**
- ✔ **Programar y realizar actividades para gestionar el mantenimiento de los recursos informáticos.**
- ✔ **Verificar los componentes de software desarrollados, analizando las especificaciones, para completar el plan de pruebas.**

PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

Enunciado:

Rendimiento de un empleado

En las actividades de evaluación anteriores hemos creado scripts para controlar el número de horas trabajadas por un empleado, así como su sueldo semanal.

Ahora queremos crear un script para que cada empleado pueda consultar sus datos personales. Nadie más a parte de él podrá verlos.

El workflow de la aplicación debe ser:

- El usuario deberá loguearse.
- Si el login es correcto deberá incrementarse en una unidad una cookie destinada a contar cuantos inicios de sesión se han producido desde un ordenador en particular.
- Los datos del usuario se recuperan de un archivo y se cargan en una variable de sesión.

- En la página principal se muestran los datos personales. En una página secundaria el recuento de horas trabajadas.

El archivo con todos los datos se llama **empleados.json**, y contiene los siguientes datos en él:

Código: datos del archivo empleados.json

```
{
  "Empleados":[{
    "Nombre":"Juan Palomarez Hernández",
    "Usuario": "jpalomares@empresa.es",
    "Contraseña":"jy3Rtbm",
    "Historial": [{
      "Lunes": 8,
         "Martes": 9,
         "Miércoles": 7,
         "Jueves": 7,
         "Viernes": 5
    }]
    },
    {
    "Nombre":"Ana Fernández Marín",
    "Usuario": "afernandez@empresa.es",
    "Contraseña":"sw9Ewph",
    "Historial": [{
      "Lunes": 6,
         "Martes": 6,
         "Miércoles": 8,
         "Jueves": 7,
         "Viernes": 8
    }]
  }]
}
```

Para cargar los datos en una variable y consultarlos como si de un array multidimensional se tratase puedes utilizar el siguiente código:

Código: cargar y consultar los datos como un array multidimensional

```
<?php
// Leer el contenido del fichero
$json = file_get_contents('empleados.json');
// Decodificar los datos
$json_data = json_decode($json,true);
// Mostrar los datos
print_r($json_data);
?>
```

1. MANTENIMIENTO DEL ESTADO

*Algunas variables predefinidas en PHP son "**superglobales**", lo que significa que siempre están accesibles, independientemente del alcance, y puede acceder a ellas desde cualquier función, clase o archivo sin tener que hacer nada especial.*

A continuación, veremos cuales son estas **variables superglobales** y la **misión** que cumplen:

- Las **variables superglobales** de PHP son:
  - ✔ $GLOBALS.
  - ✔ $_SERVER.
  - ✔ $_REQUEST.
  - ✔ $_POST.
  - ✔ $_GET.
  - ✔ $_FILES.
  - ✔ $_ENV.
  - ✔ $_COOKIE.
  - ✔ $_SESSION.

- Algunas de ellas ya las hemos visto ($_GET, $_POST) y otras las veremos en breve, pero todas ellas comparten como mínimo una **misión**: **mantener el estado de un valor entre espacios de distinto alcance**.

  Quizás la variable con el propósito más claro de mantener el estado es **$_SESSION**. Esta es un **array especial** utilizado para guardar información a través de los requests que un cliente hace durante su visita a un sitio web o

aplicación. La información guardada en una sesión puede llamarse en cualquier momento mientras la sesión esté abierta.

A cada usuario que accede a la aplicación e inicia sesión se le asigna un **identificador único (PHPSESSID),** y es lo que le permite identificarlo en el servidor. Cuando el servidor recibe el PHPSESSID de un cliente puede mostrarle a él, y sólo a él, sus datos personales.

2. SESIONES

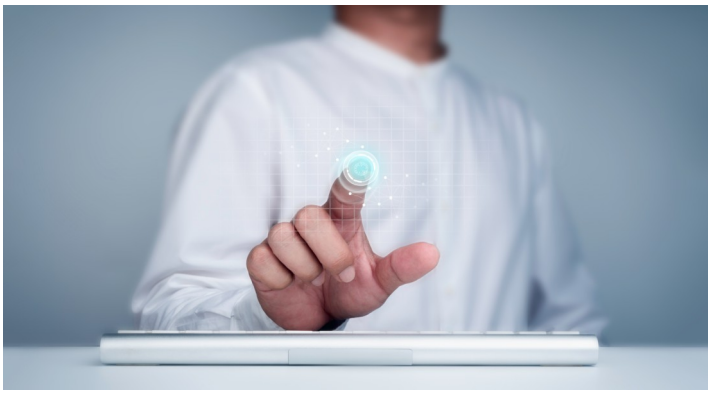
Normalmente los **formularios de *login*** sirven para iniciar una sesión. Una **sesión** es una forma de guardar cierta información para ser utilizada en múltiples páginas. Esta información se guarda en la variable superglobal llamada **$_SESSION**.

A diferencia de las cookies, la **información** de las sesiones no se guarda en la máquina cliente sino en el **servidor**.

*Las **sesiones se inician** con el método `session_start()`. Mientras la sesión esté abierta, todos los datos colocados en la variable $_SESSION serán accesibles.*

*Dependiendo de cómo esté configurado nuestro servidor, las sesiones se cerrarán automáticamente al cerrar el navegador, o caducarán al cabo de un rato. De todas formas, si lo queremos controlar nosotros deberemos cerrar la sesión manualmente. Para ello nos ayudarán los métodos `session_unset()`, para **eliminar los datos**, y `session_destroy()` para **eliminar la sesión**.*

En el siguiente ejemplo **construiremos 3 páginas web**:



➔ **Primera web**.

La primera web, que se llama **web1.php,** servirá para guardar un nombre en la variable $_SESSION:

```
<?php
      session_start();
?>
<!DOCTYPE html>
<html>
      <body>
        <?php
            $_SESSION["nombre"] = "Juan";
            echo "La variable nombre se ha guardado en \$_SESSION.";
        ?>
        <a href="./web2.php">Click me</a>
      </body>
</html>
```

➔ **Segunda web**.

La segunda web, que se llama **web2.php**, nos recuerda y nos saludará con nuestro nombre:

```
<?php
      session_start();
?>
<!DOCTYPE html>
<html>
      <body>
        <?php
            echo "Hola " . $_SESSION ["nombre"];
        ?>
         <a href="./web3.php">Click me</a>
      </body>
</html>
```

➔ **Tercera web**.

Finalmente, la tercera web, que se llama **web3.php**, destruirá la sesión.

```php
<?php
	session_start();
?>
<!DOCTYPE html>
<html>
	<body>
	<?php
		session_unset();
		session_destroy();
		if (isset($_SESSION ["nombre"]))
		echo "Hola " . $_SESSION ["nombre"];
		else
		echo "Hola desconocido";
	?>
	</body>
</html>
```

Fíjate en que, para manejar sesiones, todos los scripts empiezan con `session_start();`

Vídeo: sesiones

Visualiza el siguiente vídeo en el que se explicará más a detalle los distintos ejemplos que se han puesto en este apartado sobre las tres páginas web que se han creado.

Cuestionario

Lee el enunciado e indica la opción correcta:

Las variables superglobales en PHP:

a. No se pueden destruir sus datos.

b. Tienen datos accesibles desde cualquier punto del código.

c. No existen las variables superglobales.

Lee el enunciado e indica la opción correcta:

El identificador asignado a un cliente para una sesión de PHP se llama:

a. CLIENTID.

b. PHPCLIENTID.

c. PHPSESSID.

Lee el enunciado e indica la opción correcta:

Si se quiere trabajar con sesiones con PHP el código debe empezar con:

a. Session_unset().

b. Session_destroy().

c. Session_start().

3. COOKIES. APLICACIONES PRÁCTICAS. LAS COOKIES Y EL NAVEGADOR

DESTACADO

*Las **cookies** son pequeños archivos que el servidor incrusta en el ordenador del usuario. Habitualmente sirven para identificar al usuario o alguno de sus datos (perfil, preferencias, lista de la compra, etc.).*

Veremos más detalles sobre las **cookies**:

- PHP nos permite generar, recuperar y modificar cookies. La **sintaxis para crearlas** es:

```
bool setcookie(string $name, string $value, int $expire, string $path, string $domain, bool $secure, bool $httponly)
```

  Para crear la cookie sólo es requisito **darle un nombre** (1r parámetro) pero también es recomendable **darle algún valor** y decirle hasta cuándo debe durar. El resto de parámetros hacen referencia a qué páginas afecta a la cookie dentro del dominio.

- La **función setcookie()** debe escribirse antes del resto del documento HTML.
- Las cookies que creamos podremos recuperarlas después con la **variable superglobal $_COOKIE**.

Veamos un ejemplo de código para la **creación y lectura de una cookie**:

Ejemplo: código para la creación y lectura de una cookie

```
$name = "cookieExample";
$value = "Lorem ipsum...";
$time = time() + 3600;  //1h = 3600sg
```

```
setcookie($name, $value, $time);

if(count($_COOKIE) > 0) {
    if(!isset($_COOKIE[$name])) {
      echo "La cookie aún no está definida.";
    }
    else {
      echo "¡La cookie está guardada!<br>";
      echo "Su valor es: " . $_COOKIE[$name] . "<br>";
    }
} else {
    echo "Las cookies están deshabilitadas.";
}
```

¿Qué hemos creado en este ejemplo?

En este ejemplo hemos creado una cookie con una hora de validez. La primera vez que corremos el script, la cookie se guardará y, en sucesivas ejecuciones, nos mostrará el valor guardado.

Actividad de aprendizaje 1: recordando al usuario

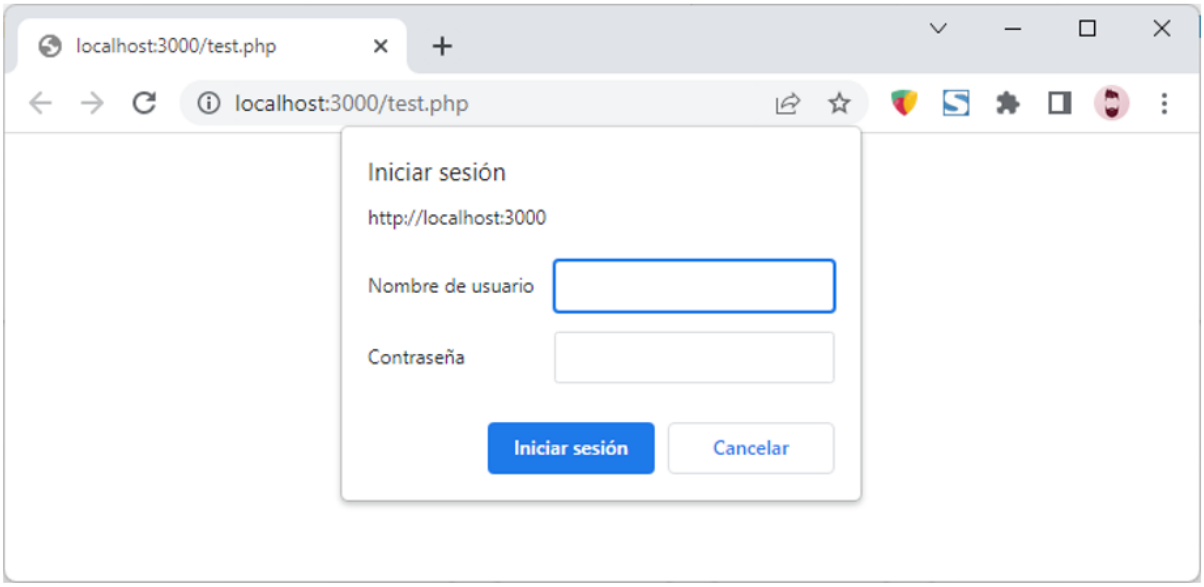
Crea un script que recuerde 3 datos personales de un usuario: nombre, fecha de nacimiento y el pueblo o ciudad donde vives. Después crea 3 scripts más, en un fichero cada uno de ellos, con esta estructura:

- **nombre.php**: script que te salude por tu nombre.
- **fecha.php**: script que muestre tu fecha de nacimiento.
- **pueblo.php**: script que muestre tu pueblo natal.

Puedes utilizar sessions o cookies para realizar esta actividad.

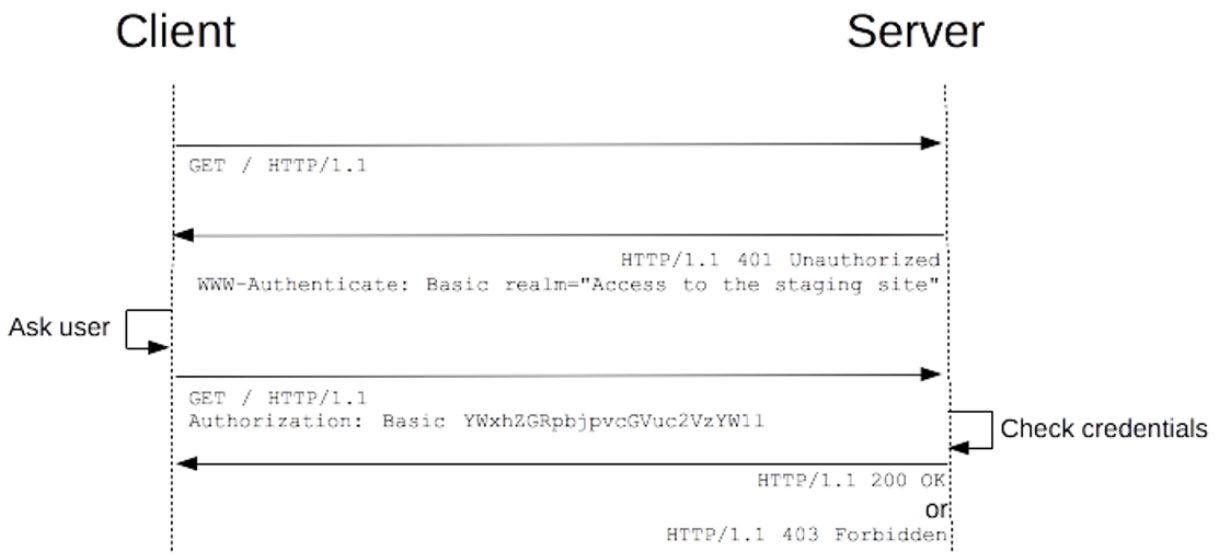
4. SEGURIDAD: USUARIOS, PERFILES, ROLES

Con la **función** `header()` se puede enviar un mensaje de "autenticación requerida" en el navegador del cliente para mostrar una ventana emergente donde introducir el usuario y la contraseña.



Ventana de autenticación.

Una vez introducidos estos datos, la URL que contiene el script de PHP será invocada de nuevo con las variables predefinidas **PHP_AUTH_USER, PHP_AUTH_PW y AUTH_TYPE** establecidas con el **nombre de usuario, contraseña y tipos de autenticación** respectivamente. Estas variables se encuentran en el array de la variable superglobal $_SERVER.



Workflow del proceso de autenticación HTTP. Fuente: https://developer.mozilla.org/. Esta imagen se reproduce acogiéndose al derecho de cita o reseña (art. 32 LPI), y está excluida de la licencia por defecto de estos materiales.

RECUERDA

Para que el manejo de headers funcione, todo el código debe escribirse antes que ningún otro código o etiquetas HTML.

5. AUTENTIFICACIÓN DE USUARIOS. OPENID, OAUTH. ACCESO AL SERVICIO DIRECTORIO LDAP

RECUERDA

*En el punto anterior hemos implementado un **mecanismo de autenticación** y hemos tenido que gestionar nosotros todo el proceso, sobre todo la tediosa faena de controlar que el usuario es quien dice ser.*

Para comodidad del desarrollador, pero también de los usuarios, a día de hoy es común la práctica de delegar dicha autenticación a un servicio externo como Google, Facebook, Apple, etc.

Implementar una autenticación con el **protocolo OAuth2** es muy sencillo, pero la forma de hacerlo varía en función de la empresa que nos preste el servicio. A pesar de ello cada empresa tiene su manual de implementación.

Google, por ejemplo, en su documentación nos enseña a crear un sistema de autenticación con PHP. En el apartado de Recursos para ampliar encontrarás todos los pasos propuestos, pero los scripts finales son los siguientes:

➔ **Fichero index.php**:

```php
<?php
require_once __DIR__.'/vendor/autoload.php';

session_start();

$client = new Google_Client();
$client->setAuthConfig('client_secrets.json');
$client->addScope(Google_Service_Drive::DRIVE_METADATA_READONLY);

if (isset($_SESSION['access_token']) && $_SESSION['access_token']) {
  $client->setAccessToken($_SESSION['access_token']);
  $drive = new Google_Service_Drive($client);
  $files = $drive->files->listFiles(array())->getItems();
  echo json_encode($files);
} else {
        $redirect_uri    =    'http://'    .    $_SERVER['HTTP_HOST']    .
'/oauth2/oauth2callback.php';
  header('Location: ' . filter_var($redirect_uri, FILTER_SANITIZE_URL));
}
```

➔ **Fichero oauth2callback.php**:

```php
<?php
require_once __DIR__.'/vendor/autoload.php';

session_start();

$client = new Google_Client();
$client->setAuthConfigFile('client_secrets.json');
$client->setRedirectUri('http://' . $_SERVER['HTTP_HOST'] .
'/oauth2/oauth2callback.php');
$client->addScope(Google_Service_Drive::DRIVE_METADATA_READONLY);
$client->addScope("email");
$client->addScope("profile");

if (! isset($_GET['code'])) {
  $auth_url = $client->createAuthUrl();
  header('Location: ' . filter_var($auth_url, FILTER_SANITIZE_URL));
} else {
  $client->authenticate($_GET['code']);
  $google_oauth = new Google_Service_Oauth2($client);
  $google_account_info = $google_oauth->userinfo->get();
  $email =  $google_account_info->email;
  $name =  $google_account_info->name;
  echo "Hola $name";
}
```

Actividad de aprendizaje 2: autenticación

Genera un pequeño script que autentifique a un usuario y lo salude con su nombre y apellidos. Los únicos usuarios del sistema son:

1. Usuario1: juan.

   Contraseña usuario1: j1@n.

   Nombre usuario1: Juan García Mohedano.

2. Usuario2: sara.

   Contraseña usuario2: s@r@.

   Nombre usuario2: Sara Fernández Agudo.

En caso que se intente loguear otro usuario, o utilice unas credenciales incorrectas, el sistema debería mostrar un mensaje de error.

Cuestionario

Lee el enunciado e indica la opción correcta:

Para crear una cookie es obligatorio:

a. El tiempo de duración.

b. El valor.

c. El nombre.

Lee el enunciado e indica la opción correcta:

Las variables PHP_AUTH_USER, PHP_AUTH_PW y AUTH_TYPE se encuentran en:

a. $_SERVER.

b. $GLOBALS.

c. $_COOKIE

Lee el enunciado e indica la opción correcta:

El protocolo Oauth sirve para:

a. La gestión de cookies.

b. Autenticar usuarios en aplicaciones web.

c. Enviar datos encriptados al servidor.

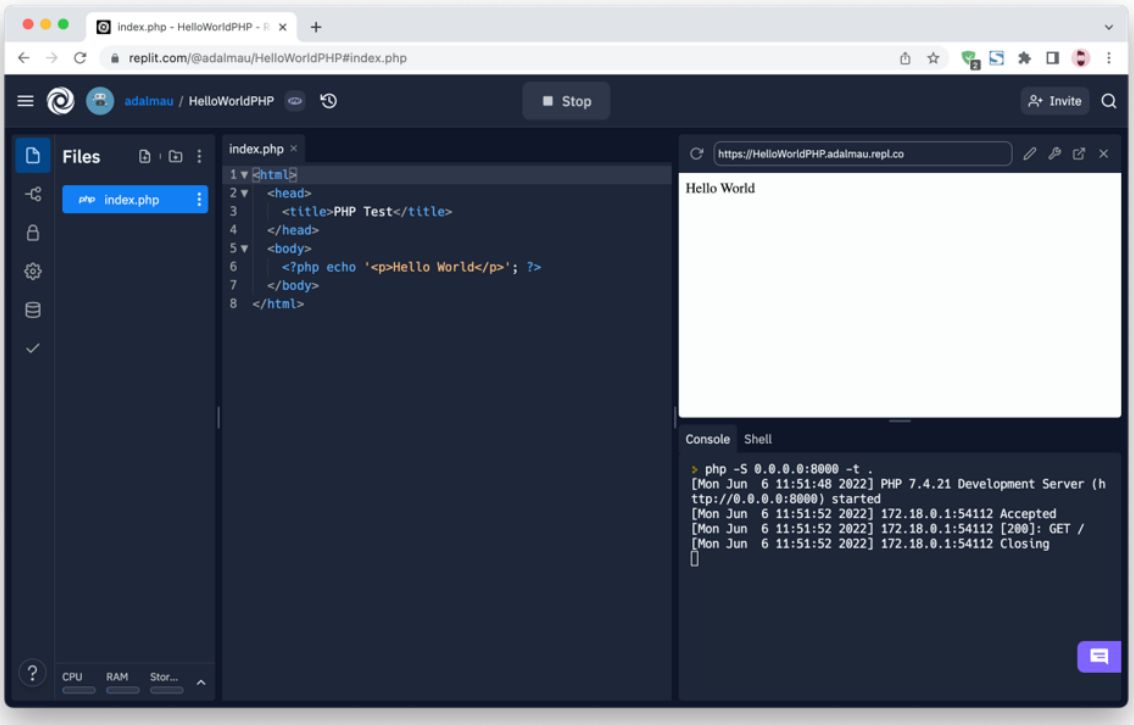
6. HERRAMIENTAS DE PROGRAMACIÓN

En la unidad didáctica 1 ya hablamos sobre todas aquellas **herramientas** que nos ayudan en el **desarrollo de aplicaciones web**. En los vídeos ilustrativos de este curso también las ves en acción.

*Como habrás observado el editor de código que normalmente se usa en los vídeos es **Visual Studio Code** ya que es uno de los más populares actualmente e integra multitud de herramientas útiles: plugins para depurar código, servidor web integrado, etc.*

Pero también muchas veces habrás visto que en ejercicios simples se ha utilizado un **editor de código online**:

- Con la ayuda de los editores de PHP en línea, se puede escribir y **ejecutar el código en línea** y no hará falta preocuparse por la configuración del entorno.
- Estos editores en línea admiten **programación básica y avanzada**. También proporcionan funciones de control de versiones y uso compartido de código. Suelen incorporar muchas más funciones como autocompletar, soporte avanzado para el *frameworks* de PHP, trabajo colaborativo, etc.
- Los **editores en línea más aconsejables** para trabajar con PHP son:
  - **Editor en línea de W3Schools**: ideal para utilizar los ejemplos de la plataforma y probar código.
  - **PHPSandbox**: es una herramienta de sandboxing en línea para proyectos PHP con soporte para paquetes de Composer.
  - **Replit**: es una plataforma online, parecida a PHPSandbox, pero con soporte a más de 40 lenguajes de programación.



Apariencia de un script PHP con el editor de código online Replit.

TOMA NOTA

Encontrarás las distintas páginas web en recursos para ampliar.

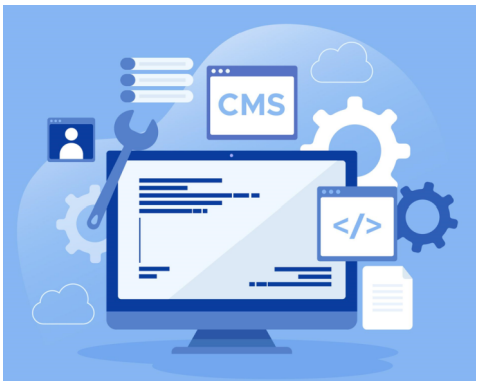
Vídeo: evolución de los editores de código

VÍDEO

Visualiza el siguiente vídeo en el que verás la evolución de los editores de código:
https://www.youtube.com/embed/qwrgwS-K3Uk

7. ADAPTACIÓN A APLICACIONES WEB: GESTORES DE CONTENIDOS Y TIENDAS VIRTUALES ENTRE OTRAS

En este apartado estudiaremos qué son los **CMS**:



- Un sistema de gestión de contenidos, **Content Management System (CMS)**, es una aplicación que permite crear una estructura de soporte para la creación y administración de contenidos, principalmente, por parte de los participantes.

  Técnicamente, es un software que nos permite crear una web sin necesidad de conocer el lenguaje HTML para publicar todo tipos de textos, archivos, etc. La publicación, normalmente, se realiza desde el propio navegador.

- **Clasificación**.

  Cada CMS tiene requisitos técnicos (plataforma tecnológica), está especializado en una función específica y tiene una licencia de uso determinada. Según estos puntos de vista se pueden **clasificar**:

  - ✔ Según la **plataforma** (incluido el lenguaje de programación empleado).
  - ✔ Según el **tipo de licencia** (propiedad del código: propietario o de código abierto).
  - ✔ Según el **tipo de aplicación** que queremos desarrollar (foro, blog, portal, tienda virtual, etc.).

Clasificación de los CMS por tipo

➔ **Funcionamiento**.

Un CMS siempre funciona en el **servidor web donde esté alojada la propia web**. El acceso al gestor se realiza generalmente a través del navegador web, pudiendo requerir el uso de FTP para subir el contenido (normalmente en el proceso de instalación).

Cuando un usuario (tanto sea administrador como visitante) accede a una URL concreta de la web en cuestión, se ejecuta una llamada al servidor, se selecciona el esquema gráfico y se introducen los datos que corresponden de la base de datos. A continuación, la página se genera automáticamente por este usuario de forma dinámica.

Todo CMS realiza el siguiente **proceso de gestión**:

- ✔ **Creación de la información**: proceso en el que un usuario genera información nueva para el público: texto, imágenes, gráficos, etc.
- ✔ Gestión de **publicación de la información**.
- ✔ **Presentación de la información**.
- ✔ **Mantenimiento de la información**: proceso donde un usuario edita, actualiza o borra información.

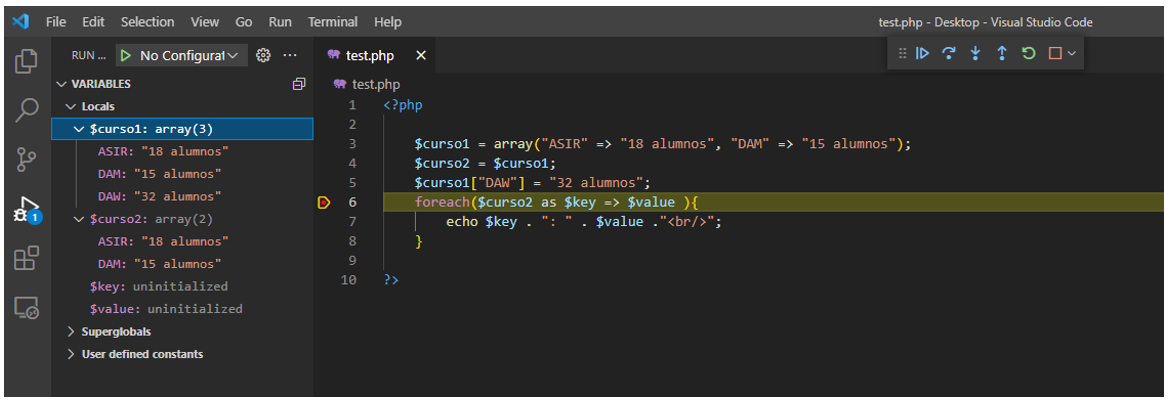
8. PRUEBAS Y DEPURACIÓN

En la unidad 1 ya hablamos sobre la depuración. En esa unidad mencionamos la herramienta estrella para la depuración en PHP: **Xdebug**. Vamos a ver un ejemplo de su funcionamiento. Para ello, imaginemos el siguiente script:

```
<?php
    $curso1 = array("ASIR" => "18 alumnos", "DAM" => "15 alumnos");
    $curso2 = $curso1;
    $curso1["DAW"] = "32 alumnos";
    foreach($curso2 as $key => $value ){
        echo $key . ": " . $value ."<br/>";
    }
?>
```

- **I**. Se ha creado un array llamado `$curso1` con una serie de datos. Justo después se crea `$curso2` y se le asigna el contenido de `$curso1`, esperando que cuando se modifique una también se modifique la otra. Esto no pasará porque los parámetros se han pasado por valor y no por referencia.
- **II**. Pero si no somos conscientes de ello, esperaremos que el resultado del `foreach()` nos muestre ASIR, DAM y DAW, pero sólo nos mostrará ASIR y DAM.
- **III**. Si no somos capaces de resolver el misterio siempre podemos poner un *breakpoint*, que es una interrupción de la ejecución del código en un punto determinado, y observar que está pasando.

➔ **IV**. En la siguiente imagen se ve el *breakpoint* en la línea 6, donde está el `foreach()`, para poder analizar los valores de los dos *arrays*, que se muestran a la izquierda de la pantalla.



Depuración con Xdebug

TOMA NOTA

En recursos para ampliar encontrarás un vídeo explicativo sobre cómo instalar Xdebug.

Vídeo: herramienta de depuración

VÍDEO

Visualiza el siguiente vídeo en el que verás cómo actúan las herramientas de depuración.

Actividad de aprendizaje 3: depuración

En el siguiente código hay, como mínimo, un par de errores que hacen que el script no funcione:

```php
<?php
$numerador = array(5, 6, 10, 0, 8);
$divisor = array(2, 3, 33, 0, 8);

for ($i=0; $i<=count($numerador); $i++) {
     $operacion = $numerador[$i]/$divisor[$i];
     echo "El resultado de dividir $numerador[$i] entre $divisor[$i] es $operacion<br/>";
}
?>
```

Con la ayuda de un depurador ¿eres capaz de encontrarlos?

Comparte en el foro que depurador has utilizado, como has cazado los errores, y qué has hecho para solucionarlos.

Cuestionario

Lee el enunciado e indica la opción correcta:

Replit es:

a. Un editor de código online.

b. Un protocolo de autenticación de usuarios.

c. Un método de PHP.

Lee el enunciado e indica la opción correcta:

Las siglas CMS hacen referencia a:

a. Create Make Service.

b. Content Management System.

c. Content of MicroSoft.

Lee el enunciado e indica la opción correcta:

Xdebug es:

a. Un IDE online.

b. Una función de PHP.

c. Una herramienta para la depuración de código.

RESUMEN

En esta unidad hemos visto conceptos ya más avanzados de la programación del lado del servidor, como los mecanismos que nos ayudan a preservar los datos del usuario: **cookies y sesiones**.

También hemos visto algunas estrategias para identificar y validar usuarios como la **autentificación HTTP y el protocolo OAuth**. Estos mecanismos están presentes en todos los CMS que requieran identificar usuarios y diferenciarlos por roles.

Alrededor de todos estos conceptos hemos vuelto a hablar sobre herramientas de programación, centrándonos en las herramientas online, y de estrategias de depuración y prevención de errores.

RECURSOS PARA AMPLIAR

PÁGINAS WEB

- Cómo usar OAuth 2.0 para acceder a las API de Google: *https://developers.google.com/identity/protocols/oauth2* [Consulta noviembre 2022].
- Instalación de Xdebug: *https://www.youtube.com/watch?v=LNIvugvmCyQ* [Consulta noviembre 2022].
- Página web PHPSandbox: *https://phpsandbox.io/* [Consulta noviembre 2022]
- Página web Replit: *https://replit.com/* [Consulta noviembre 2022]
- Página web W3Schools: *https://www.w3schools.com/* [Consulta noviembre 2022]
- Tipos de autenticación HTTP: *https://keepcoding.io/blog/tipos-de-autenticacion-http/* [Consulta noviembre 2022].

BIBLIOGRAFÍA

PÁGINAS WEB

- Referencia oficial de PHP: *https://www.php.net/* [Consulta noviembre 2022].
- Tecnologías de Java: *https://www.oracle.com/java/technologies/* [Consulta noviembre 2022].

GLOSARIO

- **Brakpoint**: punto de pausa en la ejecución de un código para poder observar con atención el estado del mismo.
- **CMS**: acrónimo de Content Management System.
- **Cookie**: archivo pequeño que el servidor incrusta en el ordenador del usuario para fines como la identificación, recolección de datos, etc.
- **Framework**: conjunto de herramientas, conceptos y técnicas estandarizados para un lenguaje de programación.
- **OAuth (Open Authorization)**: es un estándar abierto que permite flujos simples de autorización para sitios web o aplicaciones informáticas. Se trata de un protocolo que permite autorización segura de una API de modo estándar y simple para aplicaciones de escritorio, móviles y web.