MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO SERVIDOR

TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Acceso al servicio de directorio, duración y pruebas

06

ÍNDICE

OBJETIVOS

Aprender a utilizar sesiones para mantener el estado de las conexiones de aplicaciones.

Describir aplicaciones que integran mecanismos de autenticación de usuarios.

Realizar adaptaciones de aplicaciones ya existentes como gestores de contenido.

Utilizar herramientas y entornos para facilitar la programación, prueba y depuración del código.

/ 1. Introducción y contextualización práctica

Los directorios son un elemento esencial en cualquier ámbito de desarrollo, puesto que permiten mostrar la información de una forma mucho más organizada, optimizando el tiempo de búsqueda y acceso al fichero o elemento deseado, entre otros factores. Podríamos decir que es una especie de base de datos que muestra la información de manera muy organizada y descriptiva.

Asimismo, el diseño, desarrollo y puesta en marcha de un buen sistema de pruebas de depuración del código serán clave para la correcta implantación de cualquier proyecto. Será posible utilizar pruebas manuales e incluso crear paquetes de pruebas autorizadas.

Ahora bien, no todas las pruebas pueden automatizarse, por ejemplo, las pruebas que implican evaluar la usabilidad de una herramienta, la creación y adecuación de pruebas de escenarios o las simulaciones se hacen siempre de forma manual. En cambio, las pruebas unitarias se suelen realizar de forma automatizada.

En este tema, veremos el funcionamiento de una herramienta clave en desarrollo de pruebas unitarias con PHP, hablamos de PHPUnit.

Finalmente, se analizarán algunos de los gestores de contenidos más importantes en la actualidad que permiten la inclusión de PHP y JavaScript.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y resolución del caso práctico.

Fig. 1. Utilización de una herramienta gráfica para generar un informe.

Audio intro.Fases de desarrollo"
https://bit.ly/2EKO0B6

/ 2. LDAP y control de sesiones

Para garantizar el acceso a los directorios en los que se recoge y estructura todo el contenido, es posible utilizar ciertos protocolos que permiten este acceso de una forma óptima.

Es el caso de **LDAP**, Lightweight Directory Access Protocol (Protocolo Ligero de Acceso al Servicio de Directorios), basado en el protocolo X.500. LDAP permite el acceso a un servicio de directorios **ordenado y distribuido** para localizar la información requerida en un entorno en red.

Resulta totalmente seguro en cuanto a la confidencialidad de los datos se refiere, puesto que LDAP soporta SSL (Capa de Conexión segura) y TLS (Seguridad de la Capa de transporte).

Una de las principales ventajas de uso de este protocolo es que permite centralizar toda la información y que sea accesible para una red desde cualquier parte.

LDAP funciona como un modelo de conexión cliente-servidor, en el cual vamos a encontrar varios servidores LDAP contenedores de información que forman la estructura total del directorio LDAP.

Cuando un cliente realiza una petición al servidor LDAP, este recibe como respuesta siempre la misma vista del directorio, independientemente del servidor LDAP desde el que se conecte.

Ante la recepción de una petición en el servidor, si este tiene la respuesta, se la enviará al cliente, pero en el caso de no tenerla, dirigirá la petición a otro servidor que sí la tenga. Finalmente, el cliente recibirá la misma respuesta, esté en el servidor LDAP que esté.

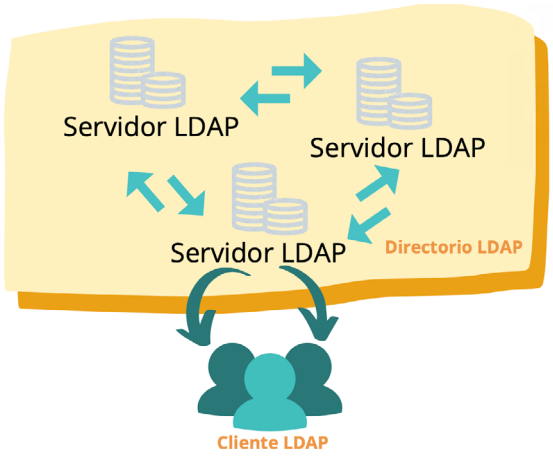

Fig. 2. Modelado de acceso en función de cliente.

Los usuarios podrían realizar varias acciones en función de sus perfiles. Podrá consultar la información contenida en el servidor LDAP y también modificar esta información. Normalmente, esta última acción está reservada para los usuarios de tipo administrador.

2.1. Estructura de LDAP

La estructura de un directorio LDAP está basada en un conjunto de nodos conectados que crean una estructura en forma de árbol. Cuando hablamos de LDAP, hemos de tener presentes cuatro conceptos clave: árboles, entradas, atributos y LDIF.

Fig. 3. Identificador LDAP.

- **Árboles**: Constituyen el sistema de organización de las entradas. En un sistema basado en un árbol, la información se clasifica en diferentes capas y subcapas que habitualmente se distribuyen desde las más genéricas a las más particulares. Por ejemplo, en una empresa, podríamos partir desde el directorio central hasta las subdivisiones correspondientes a cada departamento.

- **Entradas**: Se trata de cada uno de los elementos que forman el directorio, quedan caracterizados por un identificador único. Cada entrada está formada por varios atributos.

- **Atributos**: Los campos de datos que forman cada una de las entradas. Cada uno de estos atributos será de un tipo concreto y queda identificado por un conjunto de abreviaturas (dc, domain component, cn, common name...).

- **LDIF**: Se trata del formato de intercambio de datos, consiste en ficheros en formato ASCII con una estructura característica que permite ser interpretada por LDAP y, por lo tanto, puede tratar la información contenida.

```
[<id>]
dn: <distinguished name>
<attrtype>: <attrvalue>
<attrtype>: <attrvalue>
<attrtype>: <attrvalue>
<attrtype>: <attrvalue>
```

Fig. 4. Ejemplo codificación LDIF.

2.2. Modelos LDAP

Es posible diferenciar varios tipos de organización de modelos para los servicios de directorio LDAP. Estos modelos establecen desde pautas de diseño, en cuanto a los identificadores para cada uno de los elementos del modelo, hasta el sistema de protección de la información, en función de los perfiles de usuario.

MODELO	DESCRIPCIÓN
Modelo de nombre	Este modelo permite describir cómo se identifica la información.
Modelo funcional	El modelo funcional describe qué operaciones están permitidas sobre la información recogida en el directorio.
Modelo de información	El modelo de información describe cómo está organizada la información. El elemento básico de información definido bajo este modelo es la entrada.
Modelo de seguridad	El modelo de seguridad describe el sistema de protección para la información contenida en el directorio frente a accesos no deseados.

Tabla 1. Modelos LDAP.

LDAP es utilizado, sobre todo, para gestionar el inicio y acceso a las sesiones de equipos en red, por lo que la gestión de los datos de usuario es un aspecto clave.

La información almacenada en cada una de las entradas se encuentra organizada en un conjunto de atributos. La extracción de la información se hará en base al nombre identificador de cada uno de estos atributos.

MODELO	DESCRIPCIÓN
dn	Domian name.
dc	Forma parte del nombre de dominio e identifica las partes del mismo.
cn	Common name, es el nombre del atributo, su identificador.
sn	Surname.
objectClass	Se utilizan para definir las propiedades que tienen los atributos.

Tabla 2. Atributos en LDAP. Identificadores.

En la actualidad, podemos encontrar un amplio catálogo de aplicaciones que utilizan este protocolo de comunicación: OpenLDAP, Apache Directory Server, Novell Directory Services o Active Directory, entre otros.

2.3. Conexión a servidor LDAP

En primer lugar, para habilitar la conexión entre la aplicación web que se está desarrollando y un servidor LDAP, es necesario acceder al fichero de configuración php.ini y activar este acceso. Para ello, hay que eliminar el carácter de inhabilitación que se encuentra delante de la llamada a la librería LDAP.

1. Accedemos al fichero **php.ini**.

2. Se ha de localizar la línea siguiente y **eliminar el carácter punto y coma** (;) por el que aparece precedida.

```
;extension = php_ldap.dll
```

Código 1. Eliminar el carácter punto y coma

A continuación, sería conveniente reiniciar el servidor de aplicaciones y ya se tendrá acceso a todas las funciones contenidas en esta librería.

FUNCIONES	DESCRIPCIÓN
ldap_connect();	Establece la conexión con el servidor y recibe, por parámetro, el nombre hostname y el puerto (se utiliza el puerto 369).
ldap_connect (string $hostname, int $port = 389)	
ldap_bind();	Permite la autenticación de un usuario en el servidor LDAP.
ldap_bind ($link_identifier [, string $bind_rdn, string $bind_password])	
ldap_close();	Cierra la conexión con el servidor.
ldap_close ($link_identifier)	

Tabla 3. Funciones LDAP.

Si se desea crear una nueva conexión con el servidor LDAP, sería necesario indicar la URL y el puerto de acceso como parámetro de la función ldap_connect.

```
<?php
$hostname = “ldap.servidor.com”;
$port = 389;
/*Se ejecuta la función de conexión al servidor LDAP*/
$idConexion = ldap_connect($hostname, $port)
or die (“No se ha podido estable la conexión con el servidor LDAP
 $hostname”);

?>
```

Código 2. Código creación conexión LDAP.

2.4. Autenticación de usuario en LDAP. Control de sesiones

Tras completarse con éxito la conexión con el servidor LDAP descrito en el apartado anterior, se realiza el proceso de autenticación del usuario.

La función utilizada para la autenticación es ldap_bind(), que recibe como parámetro obligatorio el identificador de conexión LDAP, y de manera opcional, el usuario y el contraseña de acceso. En el caso de no enviarse como parámetro, se considerará un acceso anónimo.

```
<?php
$hostname = “ldap.servidor.com”;
$port = 389;
$user  = ‘user’;
$pass = ‘pass’;
/*Se ejecuta la función de conexión al servidor LDAP*/
$idConexion = ldap_connect($hostname, $port)
or die(“No se ha podido estable la conexión con el servidor LDAP
$hostname”);
if ($idConexion) {
/*Si la conexión está verificada, se ejecuta la función que comprueba si los
credenciales del usuario son correctos y, por tanto, queda autenticado.*/
$enlaceCorrecto = ldap_bind($idConexion,$user, $pass);
/*Se comprueba si la autenticación ha sido correcta. Si el usuario está
autenticado, devuelve el valor true*/
   if($enlaceCorrecto)
{
    echo “Autenticación correcta”;
   }else{
echo “Autenticación incorrecta”;
}
}

?>
```

Código 3. Autenticación de usuario en LDAPO.

Como se puede ver en el código, solo si se ha recuperado un identificador válido de conexión tras la ejecución de la función ldap_connect(), se continúa con el proceso de autenticación.

Finalmente, para deshacer el enlace establecido con el servidor LDAP, es decir, para cerrar la conexión, se utiliza la función **ldap_unbild() o ldap_close()**, que, en este caso, solo recibe por parámetro el identificador devuelto por la función de conexión ldap_connect().

/ 3. Pruebas y duración

Las pruebas son una fase indispensable en el desarrollo de cualquier aplicación, herramienta o sistema, puesto que permiten inspeccionar el software desarrollado atendiendo a todos los posibles escenarios que se pueden contemplar.

Casi todos los desarrollos de software están basados en dos grupos de pruebas claramente diferenciados, por un lado, las **pruebas de caja negra** en las cuales se evalúa la aplicación desde un punto de vista externo, es decir, sin preocuparnos del "interior". Son las habituales para la prueba de interfaces.

Por otro lado, encontramos las **pruebas de caja blanca**, estas se basan en la evaluación del código interno del software. Un buen diseño para estas pruebas implica la evaluación de todos los posibles caminos que se han implementado en el diseño de un programa.

Podemos diferenciar también entre **las pruebas realizadas a nivel de módulos (pruebas modulares)** o **las pruebas de funcionamiento general.** Las pruebas de funcionamiento general se realizan cuando todos los módulos del sistema se encuentren integrados, puesto que lo que se pretende es validar el funcionamiento global de la aplicación antes de ser entregada al cliente.

Para contemplar todos los posibles escenarios, es recomendable realizar la fase de pruebas desde múltiples puntos de vista:

1. Evaluando errores habituales.
2. Realizando pruebas sobre las acciones más comunes.
3. Implementando casos de pruebas para todos los tipos de usuarios y escenarios posibles.

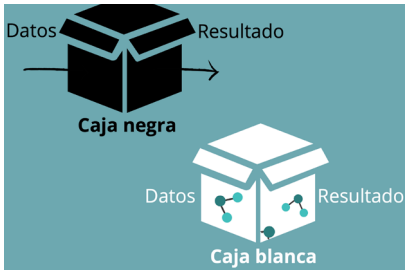

Fig. 5. Caja negra y caja blanca.

Audio 1. "Ventajas de desarrollo de un sistema de pruebas"
https://bit.ly/2EWMMSS

3.1. Tipos de pruebas

Además de la clasificación genérica de pruebas de caja negra y pruebas de caja blanca, es posible dividir las fases de pruebas en diferentes tipos atendiendo al tipo de funcionalidad evaluada en cada caso. Algunas de las más habituales son:

- **Pruebas unitarias**: Este tipo de pruebas evalúan funcionalidades concretas, examinando todos los caminos posibles implementados en el desarrollo de un algoritmo, función o clase.
- **Pruebas de integración**: Tras realizar las pruebas unitarias, se utilizan las de integración en toda la aplicación ya totalmente integrada.
- **Pruebas de regresión**: Este tipo de pruebas es muy común y a la vez, muy importante, puesto que implica volver a verificar aquello que ya había sido evaluado previamente y había generado algún tipo de error. La superación con éxito de este tipo de pruebas es imprescindible para validar que los cambios han sido correctos.

- **Pruebas de seguridad**: Este tipo de pruebas se centran, sobre todo, en la evaluación de los sistemas de protección y autenticación de una aplicación.

- **Pruebas de volumen y de carga**: En algunas ocasiones, es necesario manejar una gran cantidad de datos y puede que el software no soporte un acceso masivo, por lo que será necesario comprobar este tipo de acceso. Este tipo de pruebas son especialmente útiles en tiendas virtuales que pueden ver sobrecargado el servidor ante un gran número de accesos.

Finalmente, también podemos diferenciar entre pruebas manuales y pruebas automáticas. Mientras que las primeras son realizadas por un desarrollador, experto o no en el software a testear, las segundas utilizan herramienta que permiten agilizar este proceso de evaluación.

Fig. 6. Diagrama de pruebas.

3.2. Depuración del código

La depuración de código, como su propio nombre indica, se centra en la revisión del código para la detección de posibles errores y la corrección de los mismos.

Podemos encontrar tres tipos de errores atendiendo al desarrollo del código:

- **Errores de compilación**: Este tipo de errores se producen por un fallo en la sintaxis del lenguaje de programación utilizado; dado que no está escrito de forma correcta, este no puede ser interpretado por completo por el compilador y devuelve este tipo de errores. Por ejemplo, si no utilizan los ";" en PHP para concluir las instrucciones, se producirá un error de compilación.

- **Errores de ejecución**: Este tipo de errores se producen cuando la sintaxis es correcta, pero se ha implementado algún tipo de operación cuyo resultado es erróneo.

- **Errores de lógica**: Por último, este tipo de errores son producidos por un fallo en el diseño de la lógica del programa, es decir, el resultado no es el esperado. Su detección es más compleja que los anteriores, puesto que mientras que los otros tipos devuelven algún mensaje de error, este tipo de errores no lo hace.

El diseño de un buen programa de pruebas es clave para la detección del último tipo de errores. Los dos primeros serán detectados en tiempo de desarrollo, habitualmente.

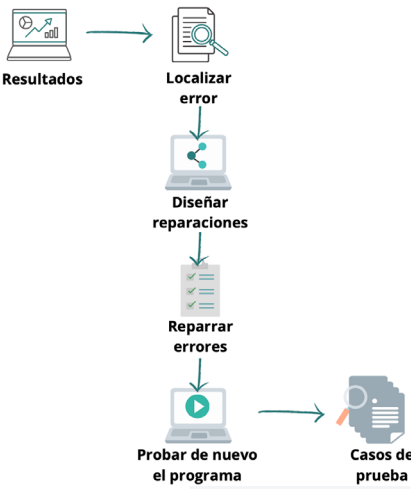

Fig. 7. Diagrama completo para depuración de código.

/ 4. Herramientas para la gestión de pruebas. PHPUnit

PHPUnit es un framework que permite la implementación de pruebas unitarias en PHP. Se trata de una herramienta para testing que permite realizar todo tipo de pruebas para aquellas aplicaciones desarrolladas en lenguaje de programación PHP.

Tras completar el proceso de descarga e instalación de este framework, se realiza la implementación oportuna para los casos de prueba. El sitio web de PHPUnit incorpora ejemplos de uso que resultan de gran utilidad para la puesta en funcionamiento de este entorno.

Para la puesta en marcha de PHPUnit, se ha de instalar el fichero PHAR, que contiene todo lo necesario para usar PHPUnit.

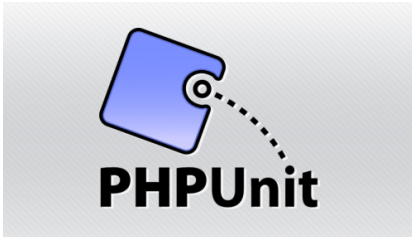

Fig. 8. Logotipo PHPUnit.

El proceso completo está formado por los siguientes pasos:

1. Se instala wget, una utilidad que permite la descarga de recursos desde una URL específica. Desde **Windows**, se realiza la descarga desde este enlace y, a continuación, se ejecuta para instarlo en el sistema. Desde Mac X OS y Linux, será necesario la instalación de Homebrew y, a continuación, de wget.

```
>> /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
Homebrew/install/master/install)"

>> brew install wget
```

Código 4. Instalación wget.

2. A continuación, utilizando wget, se realiza la descarga de PHPUnit y se le dan los permisos de ejecución oportunos.

```
>> wget -O phpunit https://phar.phpunit.de/phpunit-9.phar

>> chmod +x phpunit
```

Código 5. Instalación PHPUnit.

3. Finalmente, se comprueba que el proceso ha concluido con éxito si al ejecutar la siguiente instrucción nos devuelve algo similar a "PHPUnit 9.3.7 by Sebastian Bergmann and contributors".

```
>>./phpunit --version
```

Código 6. Versión PHPUnit descargado.

4.1. Diseño de las pruebas con PHPUnit

El desarrollo de los ficheros de prueba implica la creación de aquellas funciones de prueba que se vayan a evaluar con cada caso. La elaboración de estas pruebas requiere de ciertas pautas de programación.

Para implementar el código de prueba, hay que tener en cuenta las siguientes directrices de diseño:

1. Se debe utilizar la siguiente instrucción al inicio del fichero en PHP:

```
use PHPUnit\Framework\TestCase;
```

Código 7. Instrucción 1

2. La clase de prueba debe extender de la clase PHPUnit_Framework_TestCase.

```
. . . extends TestCase {. . .}
```

Código 8. Clase de prueba

3. Dentro de la declaración de cada función de testing, se utilizan las aserciones que comprueban si el resultado de la prueba es el esperado o no. Por ejemplo, en este fragmento, se comprueba si el valor recibido por parámetro está vacío.

```
$this -> assertEmpty (['foo']);
```

Código 9. Ejemplo

4. Los métodos de prueba deben comenzar con la palabra test.

5. Los métodos de la clase de prueba deben ser públicos.

6. Es aconsejable que el nombre de la clase de prueba sea similar al de la clase a probar, es habitual añadir la palabra test delante.

La creación de las funciones de prueba va a utilizar las funciones de aserción que permiten realizar comprobaciones sobre los resultados obtenidos en la ejecución de las funciones durante las pruebas.

FUNCIÓN	DESCRIPCIÓN
assertSame()	Comprueba si el valor obtenido coincide con el esperado.
assertFalse()	Comprueba si el valor obtenido es false.
assertFileExists()	Comprueba si el archivo indicado existe.
assertContainsOnly()	Comprueba si los valores obtenidos son todos de un mismo tipo concreto.
assertAcount()	Comprueba si el número de elementos es el esperado.
assertEmpty()	Comprueba si el resultado de la prueba está vacío.

Tabla 4. Aserciones en PHPUnit.

Existe una gran variedad de aserciones, todas las funciones se encuentran disponibles desde el sitio web de PHPUnit (https://phpunit.readthedocs.io/es/latest/assertions.html#appendixes-assertions).

4.2. Configuración del entorno con PHPUnit y ejecución

Antes de continuar con el proceso de ejecución de las pruebas diseñadas, es importante tener en cuenta que la instalación que se ha realizado de PHPUnit ha de estar situada en la carpeta donde se encuentran todos los ficheros ejecutables o, de lo contrario, las pruebas solo podrían lanzarse desde la ruta en la que se instaló el framework al principio.

- **MAC X OS y Linux:** En estos dos sistemas operativos, basta con cambiar la ubicación del fichero phpunit desde la ruta en la que se instaló hasta la carpeta /user/local/bin/ utilizando el comando mv.

```
sudo mv  phpunit /user/local/bin/phpunit
```

Código 10. Instrucción ubicación phpunit MAC y Linux.

- **Windows**: Para este sistema operativo, será necesario acceder a la configuración de las variables de entorno y modificar el PATH añadiendo la ruta C:\bin en la que se habrá almacenado previamente la descarga de phpunit. En este enlace se puede ver un ejemplo sencillo para realizar esta instalación. Para concluir la instalación, se debe ejecutar la siguiente instrucción por línea de comandos desde la ruta C:\bin:

```
echo @php "%~p0phpunit.phar" %* > phpunitcmd
```

Código 11. Instrucción ubicación phpunit Windows.

Tras la implementación de los ficheros de testing, la ejecución de las pruebas se realiza por línea de comandos utilizando la siguiente sintaxis: en primer lugar, se indica el fichero donde está implementado el programa que va a ser evaluado, y, a continuación, se indican los ficheros de pruebas creados para la evaluación del primero.

```
phpunit --bootstrap FicheroPrincipal.php  FicheroTest.php
```

Código 12. Ejecución PHPUnit.

Si la ejecución se ha completado con éxito, significa que los resultados obtenidos en las pruebas coinciden con los esperados y especificados mediante las aserciones en los ficheros de prueba en PHP.

```
MBP-de-Diana:phpUnit diana$ phpUnit --bootstrap Email.php EmailTest.php
PHPUnit 9.3.7 by Sebastian Bergmann and contributors.

...                                                                 3 / 3 (100%)

Time: 00:00.004, Memory: 16.00 MB

OK (3 tests, 3 assertions)
MBP-de-Diana:phpUnit diana$ phpUnit --bootstrap Email.php EmailTest.php
```

Fig. 9. Ejecución en termina sin aserciones.

Vídeo 1. "Instalación y uso de PHPUnit"
https://bit.ly/352LY9R

/ 5. Caso práctico 1: "Autenticación usando LDAP"

Planteamiento: Se pide diseñar un fichero en PHP que permita autenticar a un usuario. Los datos correspondientes al usuario y la contraseña han sido introducidos utilizando un formulario en HTML y usando el método POST del protocolo de comunicación HTTP.

Si la conexión no se puede realizar, se debe devolver el mensaje "Conexión no realizada" por pantalla. Si la conexión se ha establecido y tanto si el usuario está autenticado o como no, el mensaje en el que se indica el resultado final debe contener el nombre de usuario.

Nudo: Dado que se utiliza el envío de datos a través de un formulario en HTML para obtener los datos ,se utilizará:

```
$idUsuario = $_POST ["login"];

$pass = $_POST ["pass"];
```

Código 13. Caso práctico 1

Desenlace: Para el modelado de este fichero PHP, se utilizan estructuras de control if-else, donde evalúa si el usuario es el que se espera para poder acceder; si lo es, se le da acceso, y si no, no.

```
<?php
$hostname = "ldap.servidor.com";
$port = 389;
$user  = $_POST ["login"];
$pass = $_POST ["pass"];
/*Conexión al servidor LDAP*/
$idConexion = ldap_connect($hostname, $port)
or die("CONEXIÓN NO REALIZADA");
if ($idConexion) {
/*Se comprueba si los credenciales del usuario son correctos y, por tanto,
queda autenticado.*/
$enlaceCorrecto = ldap_bind($idConexion,$user, $pass);
/*Se comprueba si la autenticación ha sido correcta. En cualquiera de los
casos, se concatena el nombre del usuario*/
   if($enlaceCorrecto)
{
    echo "Autenticación correcta ".$user;
   }else{
echo "Autenticación incorrecta ".$user;
}

}
```

Código 14. Código PHP en LDAP autenticación usuario.

/ 6. Caso práctico 2: "Pruebas unitarias con PHPUnit"

Planteamiento: Se pide implementar una clase de test utilizando PHPUnit para probar el funcionamiento de la función inicioCadena:

```
class CadenasDeTexto{
protected $cadena;
public function __construct($cadena, $encoding = 'UTF-8', $forceEncode
= true){
if (mb_detect_encoding($cadena) != 'UTF-8' && $forceEncode) {
$cadena = mb_convert_encoding($cadena, 'UTF-8');
}
$this->cadena = $cadena;
 }
public function inicioCadena($cadenaAuxiliar){
return mb_substr($this->cadena, 0, mb_strlen($cadenaAuxiliar)) ===
$cadenaAuxiliar;
 }

}
```

Código 15. Código PHPUnit.

Nudo: A continuación, se va a crear una prueba que permite evaluar el funcionamiento de la función inicioCadena. En la creación de esta, se incluye:

1. use PHPUnit\Framework\TestCase;

2. extends TestCase

3. Los métodos de prueba deben comenzar con la palabra test

4. Los métodos de la clase de prueba deben ser públicos

Desenlace: En el fichero de testing creado con PHPUnit, se incluyen los elementos clave señalados anteriormente. Se utilizan las aserciones que permiten comprobar si el resultado de una función booleana es true o false.

```
<?php
use PHPUnit\Framework\TestCase;
class XStringsTest extends PHPUnit_Framework_TestCase{
public function testStartWith(){
$nuevaCadena= new CadenasDeTexto('Hola Mundo');
/*assertTrue o assertFalse comprueban la salida de una función booleana*/
$this->assertTrue($nuevaCadena->startWith('Hola'));
$this->assertFalse($nuevaCadena->startWith('Mundo'));
  }

?>
```

Código 16. Código PHPUnit test.

/ 7. Gestores de contenido: Dreamweaver y Joomla!

Los gestores de contenido o CMS (Content Management System) son programas que permiten crear, editar y publicar contenido web utilizando una interfaz gráfica y posibilitando personalizar sus componentes si se tienen los conocimientos de programación necesarios.

Los gestores de contenido se pueden dividir en dos grandes grupos, los de software propietario, que implican el uso de una licencia, y los de software libre.

Los de software libre son los más utilizados entre los desarrolladores, puesto que permiten modificar el código fuente adaptándolo así a casi todo tipo de especificaciones deseadas. En la actualidad, los gestores de contenidos están basados en lenguaje de programación PHP y HTML, y gestores de datos con MySQL.

a. **Dreamweaver**: Dreamweaver es un gestor de contenidos dirigido, sobre todo, a la edición y desarrollo desde cero de un nuevo sitio web, puesto que incorpora un potente editor de código para HTML, CSS, PHP y Javascript.

   También incorpora multitud de plantillas que permiten diseñar un sitio web de una forma más ágil y que, posteriormente, puede ser personalizado a través de la inclusión o modificación de código HTML, PHP o JavaScript.

b. Joomla!: Joomla! es uno de los gestores de contenidos que permite publicar, editar y modificar el contenido en múltiples sitios web. Se trata de unos de los CMS más utilizados en la actualidad, sobre todo, por desarrolladores y expertos en programación, puesto que resulta más complejo que otros, como WordPress. Este sistema está basado en PHP y algunas de sus características principales son que está orientado a objetos y que permite a los desarrolladores la creación de nuevas extensiones.

Joomla! permite la creación de páginas web dinámicas. El sitio web oficial de este gestor (https://www.joomla.org) permite la descarga del software necesario.

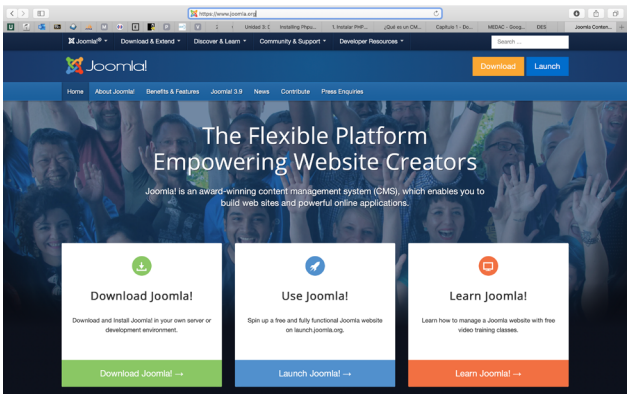

Fig. 10. Sitio web oficial Joomla!

/ 8. Gestores de contenido: WordPress

Wordpress es uno de los gestores de contenido más utilizados en la actualidad a nivel mundial. Al tratarse de un gestor de software libre, permite una amplia personalización de las plantillas ya incorporadas o la creación de nuevas desde cero utilizando PHP o JavaScript.

Tras la creación de los nuevos scripts en JavaScript, estos se insertan en el fichero de funciones implementado (functions.php) correspondiente al tema o plantilla descargado para la creación del sitio en PHP.

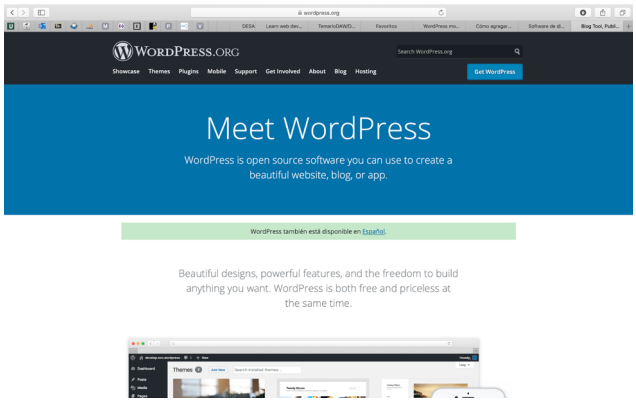

Fig. 11. Sitio web oficial WordPress

Una de las tendencias es el uso de XAMPP para la instalación de WordPress en local. Para ello, en primer lugar, se realiza la descarga del gestor (https://wordpress.org/download/), siempre es aconsejable utilizar una de las versiones más recientes.

A continuación, se coloca el archivo descargado y descomprimido en la carpeta de desarrollo en el entorno XAMPP (htdocs), en concreto, en la carpeta de nombre "wordpress". Si ahora accedemos a la URL http://localhost/wordpress/, nos aparecerá una pantalla como la siguiente si todo ha funcionado correctamente.

Fig. 12. Mensaje instalación correcta en XAMPP

Video 2. "instalación y configuración de Wordpress"
https://bit.ly/2ETKEvw

/ 9. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos analizado teórica y prácticamente el **sistema LDAP**, el cual permite el acceso a un servicio de directorios ordenado y distribuido para localizar la información requerida en un entorno en red. El funcionamiento está basado en un modelo de conexión cliente-servidor, donde un cliente realiza una petición al servidor y este recibe como respuesta siempre la misma vista del directorio, independientemente del servidor LDAP desde el que se conecte.

Cuando hablamos de LDAP, se han de tener presentes cuatro conceptos clave siempre: árboles, entradas, atributos y LDIF. Además, también se han visto la funciones clave para la configuración de sesiones con LDAP, ldap_connect(), ldap_bind() o ldap_close().

Hemos comprobado, también, que el sistema de pruebas es indispensable en cualquier escenario, ya que permite analizar el software creado antes de ser implantado en producción. Siempre resultarán menos costosos los cambios que se deban realizar antes de implantar cualquier herramienta, por eso es conveniente realizar un sistema de prueba los más completo posible. PHPUnit es un framework que permite la implementación de pruebas unitarias en PHP. Finalmente, hemos aprendido que los gestores de contenido desplegados en un entorno de desarrollo permiten modificar e incluir código PHP que personalice casi cualquier tipo de plantillas.

Resolución del caso práctico inicial

Todas las fases son necesarias para optimizar la creación de un sitio web, tanto por tiempo como por coste económico. Si, por ejemplo, omitimos la fase de evaluación, cuando esta suba a producción, podrán aparecer errores que no se habían detectado al no ser evaluada.

Las funciones principales de cada fase son:

1. **Fase de planificación**. En esta se definen los criterios y necesidades básicas del sitio web.
2. **Fase de diseño**. Durante esta fase, se realizará el diseño en base a los criterios definidos en el apartado anterior.
3. **Fase de implementación**. Aquí se escribe y desarrolla el código diseñado en los apartados anteriores.
4. **Fase evaluación**. Fase de pruebas para evaluar el estado del sitio web antes de ser publicado. Al concluir esta fase, será necesario regresar a la fase anterior, en el caso de haber aparecido errores o cambios.
5. **Fase de producción**. En esta fase, se publica el sitio web y los usuarios pueden acceder.
6. **Fase de mantenimiento.** Tras ser publicada, es posible incorporar nuevas funcionalidades o resolver casuísticas que no se habían tenido en cuenta durante su diseño e implementación.

/ 10. Bibliografía

Ganzabal, X. (2019). *Desarrollo web en entorno servidor (1.a.ed.)*. Madrid, España: Síntesis

Vara, J.M. (2012). *Desarrollo en entorno Servidor (1.a ed.)*. Madrid, España: Rama.

10.1. Webgrafía

PHP. Recuperado de: https://www.php.net/manual/es/intro-whatis.php