TEMA 5. DESARROLLO DE APLICACIONES WEB UTILIZANDO CÓDIGO EMBEBIDO.

Objetivos

- Conocer la importancia y los métodos para establecer y controlar el mantenimiento del estado de un usuario en una aplicación web.
- Aprender a manejar el estado de un usuario a través de sesiones y de cookies.
- Conocer las listas de control de acceso (ACL) como mecanismos de seguridad para el control de usuarios perfiles y roles.
- Introducir al alumno en las nuevas técnicas de autenticación universl como OpenID u OAuth.
- Comprender el objetivo del uso de un servicio de directorio LDAP y cómo usarlo a través de la web.
- Presentar algunos de los conceptos de pruebas y depuración de aplicaciones web basado en lenguajes embebidos.

5.1.- Mantenimiento del estado en Aplicaciones Web.

El salto de una web a otra implica la pérdida de datos y no preserva los estados de la misma. La única forma de garantizar la permanencia de dichos estados es a través de un mecanismo de sesiones que permita registrar estos estados haciéndolos consistentes y permitiendo que puedan ser gestionados.

El objetivo de la sesión es guardar información específica de cada usuario mientras navega por un sitio web. Cada usuario que entra en un sitio, abre una sesión que es diferente e independiente al del resto de usuarios, evitando que el usuario al pasar de una página a otra pierda la información.

Podemos establecer una sesión entre el servidor y el cliente por medio de URL mediante GET o POST o mediante la creación de cookies.

Cuando un cliente se identifica en el servidor, este envía un identificador único que gestiona la sesión y se llama SID. El SID es un numero asignado por el servidor de forma aleatoria y utilizado para gestionar la sesión. Se almacenará temporalmente hasta cerrar la sesión con el servidor.

Si utilizamos cookies el SID se almacena en un pequeño archivo junto a otros parámetros como la fecha que expira la cookie. Este archivo permite al servidor identificar al cliente. El problema es que muchos navegadores no permiten el uso de cookies o no las soportan. Para evitar esto podemos utilizar sesiones de cookies y por URL a la vez y se debe configurar en el fichero php.ini del servidor.

Control de sesiones con PHP.

El lenguaje PHP cuenta con funciones relacionadas con el manejo de sesiones que permiten guardar datos acerca de la visita y pueden recuperarse en cualquier momento.

Las funciones session_start() debe encabezar la página para permitir crear una sesión o mantener una sesión creada en páginas anteriores y session_destroy() permite destruir una sesión creada si ya no la vamos a utilizar y vaciar sus variables de sesión.

El ID de sesión se obtiene o se asigna mediante session_id(). Por ejemplo podemos asignar un valor al id de sesión utilizando:

```
session_id('misesion');
```

o permitir que PHP asigne un ID aleatorio con:

```
session_id();
```

Podemos crear variables de sesión utilizando:

```
$_SESSION['nombre_variable_sesion'] = valor;
```

Las variables de sesión se almacenan en el servidor en el array **$_SESION[]**.

Ejemplo:

pag_sesion_1.php

```
<?php
	// Damos ID a la sesión que vamos a crear
	session_id('misesion');
	// Creamos sesion
	session_start();
	echo "El ID de sesión es: ".session_id()."<br>";
	// Creamos variable de sesion color
	$_SESSION['color']="azul";
	echo "<a href='pag_sesion_2.php'>Enlace</a>";
?>
```

pag_sesion_2.php

```
<?php
	//Mantenemos la sesion
	session_start();
	echo "El ID de sesión es: ".session_id()."<br>";
	// Mostramos el valor de la variable de sesión color
	if (isset($_SESSION['color'])) {
		echo "variable existe y es: ".$_SESSION['color']."<br>";
	} else {
		echo "variable NO existe<br>";
	}
```

```
      //  destruimos la sesión y borramos las variables de sesión
      session_unset();
      session_destroy();
      if (isset($_SESSION['color'])) {
            echo "variable existe y es: ".$_SESSION['color']."<br>";
      } else {
            echo "variable NO existe<br>";
      }
      echo "<a href='pag_sesion_1.php'>Enlace</a>";
?>
```

Control de cookies en PHP.

Para manipular el envío de cookies desde el servidor disponemos de la función setcookie(). La función crea un pequeño archivo plano en el equipo cliente. La función setcookie dispone de los siguientes parámetros:

setcookie(nombre, valor, seg_tiempo_vida);

Podemos acceder a la cookie creada por medio del array de cookies $_COOKIE[]:

$_COOKIE['nombre'];

Ejemplo:

cookie_pag_1.php

```
<?php
      setcookie("color","azul",time()+60);
      echo "<a href='cookie_pag_2.php'>Enlace</a>";
?>
```

cookie_pag_2.php

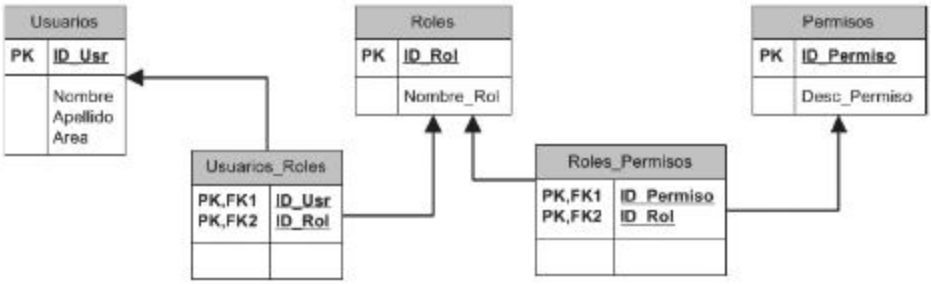
```
<?php
      if (isset($_COOKIE['color'])) {
            echo $_COOKIE['color'];
      } else {
            echo "El tiempo ha expirado";
      }
?>
```

Por razones de seguridad funcional, el archivo cookie no debe superar los 2Kb y como máximo la cantidad de cookies no debe superar los 20 archivos. Otro inconveniente es la posibilidad de que el usuario bloquee las cookies por parte del navegador, lo que implica no poder gestionar las sesiones.

5.2.- Seguridad: Usuarios, Perfiles y Roles.

Un aspecto complementario al manejo del estado en las páginas es el control de usuarios. Por lo general un usuario con distintos niveles de acceso pueden realizar diversas operaciones. Uno de los mecanismos más usuales es a través de Perfiles y Roles.

A continuación se muestra un esquema de datos para implementar una ACL que restringe el acceso a los usuarios a páginas especificas en función de los permisos que tiene el perfil o rol que tenga asignado:



Trabajando con una estructura de datos de arrays esta sería la implementación de la ACL.

```
$usuarios =  array(array(1, "Luis", "Sancho Blas", "lsancho", "pass1"),
       array(2, "Sara", "Solis Romero", "ssolis", "pass2"),
       array(3, "Ana", "Mas Medina", "amas", "pass3"),
       array(4, "Pedro", "Ros Sendra", "pros", "pass4"));

$roles = array (array(1, "Administrador"),
          array(2, "Profesor"),
          array(3, "Alumno"));

$permisos = array (array(1, "Insertar"),
             array(2, "Borrar"),
             array(3, "Modificar"),
             array(4, "Consultar"),
             array(5, "Listar"));

$roles_permisos = array ( array (1, 1),
                     array (1, 2),
                     array (1, 3),
                     array (1, 4),
                     array (1, 5),
                     array (2, 1),
                     array (2, 4),
                     array (2, 5),
                     array (3, 4));

$usuarios_roles = array (array (1, 1),
                     array (2, 3),
                     array (3, 2),
                     array (4, 2));
```

Se pedirán los datos de usuario y contraseña, si la autenticación es correcta el siguiente paso es consultar el rol que posee el usuario. Una vez localizado el rol accederemos al array de permisos para que nos devuelva que permisos tiene asociado el rol del usuario.

Ejemplo:

acceso.php

```
<?php
    include('acl.php');
    include('funciones.php');
    if (isset($_POST['usuario']) && isset($_POST['pass'])){
            $usuario=$_POST['usuario'];
            $pass=$_POST['pass'];
            if (accesoUsuario($usuarios, $usuario, $pass)){
             $id_usu = devuelveId($usuario, $usuarios);
             $id_rol = devuelveRol($id_usu, $usuarios_roles);
             $listaPermisos = devuelvePermisos($id_rol, $roles_permisos);
             echo "Nombre: ".muestraUsuario($id_usu, $usuarios)."<br>";
             echo "Rol: ".muestraRol($id_rol, $roles)."<br>";
             echo "Permisos: ".muestraPermisos($listaPermisos, $permisos);
            } else {
             echo "Usuario o Contraseña incorrectos.";
            }
    } else {
?>
    <meta charset="UTF-8">
    <form action="" method="POST">
        Usuario: <input type="text" name="usuario"><br>
        Contraseña: <input type="password" name="pass"><br>
        <input type="submit" value="Acceso">
    </form>
<?php
    }
?>
```

acl.php

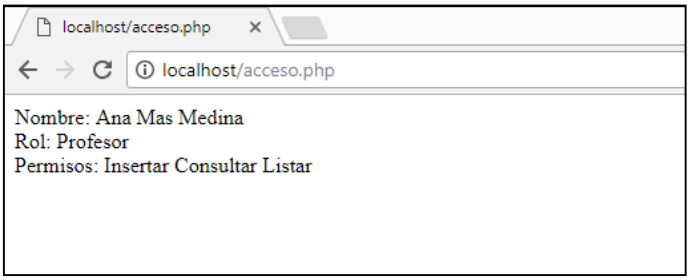
```
<?php
       $usuarios =  array(array(1, "Luis", "Sancho Blas", "lsancho", "pass1"),
              array(2, "Sara", "Solis Romero", "ssolis", "pass2"),
              array(3, "Ana", "Mas Medina", "amas", "pass3"),
              array(4, "Pedro", "Ros Sendra", "pros", "pass4"));

       $roles = array (array(1, "Administrador"),
                 array(2, "Profesor"),
                 array(3, "Alumno"));

       $permisos = array (array(1, "Insertar"),
                    array(2, "Borrar"),
                    array(3, "Modificar"),
                    array(4, "Consultar"),
                    array(5, "Listar"));
```

```
        $roles_permisos = array ( array (1, 1),
                              array (1, 2),
                              array (1, 3),
                              array (1, 4),
                              array (1, 5),
                              array (2, 1),
                              array (2, 4),
                              array (2, 5),
                              array (3, 4));

        $usuarios_roles = array (array (1, 1),
                              array (2, 3),
                              array (3, 2),
                              array (4, 2));
?>
```

funciones.php

```
<?php
        function accesoUsuario($usuarios, $usuario, $pass){
                $existe = false;
                foreach($usuarios as $valor){
                        if ($valor[3]==$usuario && $valor[4]==$pass){
                                $existe=true;
                                break;
                        }
                }
                return $existe;
        }

        function devuelveId($usuario, $usuarios){
                $id_usu = 0;
                foreach($usuarios as $valor){
                        if ($valor[3]==$usuario){
                                $id_usu = $valor[0];
                                break;
                        }
                }
                return $id_usu;
        }

        function devuelveRol($id_usu, $usuarios_roles){
                $id_rol = 0;
                foreach($usuarios_roles as $valor){
                        if ($valor[0]==$id_usu){
                                $id_rol = $valor[1];
                                break;
                        }
                }
                return $id_rol;
        }
```

```
	function devuelvePermisos($id_rol, $roles_permisos){
		$id_permisos = array();
		foreach($roles_permisos as $valor){
			if ($valor[0]==$id_rol){
				array_push($id_permisos, $valor[1]);
			}
		}
		return $id_permisos;
	}

	function muestraUsuario($id_usu, $usuarios){
		$nombreCompleto = "";
		foreach($usuarios as $valor){
			if ($valor[0]==$id_usu){
				$nombreCompleto=$valor[1]." ".$valor[2];
				break;
			}
		}
		return $nombreCompleto;
	}
	function muestraRol($id_rol, $roles){
		$rol = "";
		foreach($roles as $valor){
			if ($valor[0]==$id_rol){
				$rol=$valor[1];
				break;
			}
		}
		return $rol;
	}

	function muestraPermisos($listaPermisos, $permisos){
		$nombrePermisos = "";
		foreach ($listaPermisos as $valor){
			foreach($permisos as $dato){
				if ($dato[0]==$valor){
					$nombrePermisos.=$dato[1]." ";
				}
			}
		}
		return $nombrePermisos;
	}
?>
```



5.3.- Permisos y Upload de ficheros al Servidor.

Permisos de ficheros

Los permisos en PHP consiste en tres componentes numéricos octales que especifican las restricciones de acceso para el propietario, el grupo de usuarios al que pertenece el propietario, y para todos los demás, en este orden. Un componente puede ser computado sumando los permisos. El número 1 significa que se conceden derechos de ejecución, el número 2 significa que se puede escribir en al archivo, el número 4 significa que el archivo se puede leer. Sume estos números para especificar los derechos necesarios. Se deben especificar en sistema Octal poniendo un 0 delante de la codificación de los permisos.

Establecer permisos:

chmod(archivo, permisos);

Ejemplo:

```
<?php
// Lectura y escritura para el propietario, nada para los demás
chmod("/directorio/archivo", 0600);
// Lectura y escritura para el propietario, lectura para los demás
chmod("/directorio/archivo", 0644);
// Todo para el propietario, lectura y ejecución para los otros
chmod("/directorio/archivo", 0755);
// Todo para el propietario, lectura y ejecución para el grupo
chmod("/directorio/archivo", 0750);
?>
```

Mostrar permisos: Muestra los permisos de un archivo o directorio.

fileperms(archivo);

Debemos usar funciones para mostrar los permisos en formato Octal.

(intval($varpermisos,8) y sprintf('%o', fileperms($miarchivo)))

Ejemplo:

```
<form name="formulario" action="">
Permisos <input type="text" name="permisos"><br>
Archivo <input type="text" name="archivo"><br>
<input type="submit" value="Cambiar Permisos">
</form>
<?php
if ((isset($_REQUEST['permisos'])) && (isset($_REQUEST['archivo']))){
      $mispermisos=$_REQUEST['permisos'];
      $miarchivo=$_REQUEST['archivo'];
      chmod($miarchivo, intval($mispermisos, 8));
      echo    "Los    permisos    del    archivo    $miarchivo    son
".substr(sprintf('%o', fileperms($miarchivo)), -4);
}
?>
```

Upload de ficheros al servidor.

Debemos utilizar el control de tipo file en el formulario. El formulario debe tener enctype multipart/form-data y method post.

Además, debemos configurar el servidor apache modificando php.ini para admitir mayor tamaño de archivo:

```
max_execution_time = 120
upload_max_filesize = 50M
post_max_size = 50M
```

Una vez subido el archivo al servidor, podemos accede a los datos mediante la matriz $_FILES. Y con move_upload_file(origen, destino) podemos mover el archivo temporal a la carpeta de destino.

Matriz $_FILES	Significado
$_FILES["archivo"]["type"]	Tipo MIME de archivo
$_FILES["archivo"]["size"]	Tamaño en bytes del archivo
$_FILES["archivo"]["error"]	Código Error al subir el archivo
$_FILES["archivo"]["name"]	Nombre del archivo
$_FILES["archivo"]["tmp_name"]	Nombre temporal del archivo

```
<form action="" method="post" enctype="multipart/form-data">
    <input type="file" name="archivo" />
    <input type="submit" name="boton" value="Subir" />
</form>
<?php
    if(isset($_POST['boton'])){
        if (($_FILES["archivo"]["type"] == "application/pdf") &&
            ($_FILES["archivo"]["size"] < 2000000)) {
            if ($_FILES["archivo"]["error"] > 0) {
                echo $_FILES["archivo"]["error"] . "";
            } else {
                if (file_exists("datos/".$_FILES["archivo"]["name"])) {
                    echo $_FILES["archivo"]["name"] . " ya existe. ";
                } else {
                    move_uploaded_file($_FILES["archivo"]["tmp_name"],
                        "datos/" . $_FILES["archivo"]["name"]);
                    echo "Archivo Subido ";
                }
            }
        } else {
            echo "Archivo no permitido";
        }
    }
?>
```

5.4.- Autenticación de Usuarios: OPENID y OAUTH.

Permiten al usuario entrar en una web pudiendo ser verificado por cualquier otro servidor que soporte estos protocolos. OAuth es un protocolo abierto que permite a través de una API segura disponer de un método estandar y simple de autenticación. Así, si un usuario desea acceder a un servicio, podrá elegir la opción de autenticarse mediante su cuenta de otro servicio como Facebook, Twitter, Google, etc...

En este caso el servicio a consumir en el que no se ha registrado se comunica con el servicio validador () y al que solicita que realice la comprobación del usuario. Por lo general, el servicio validador, envía a una página segura donde se introducirá el usuario y la contraseña que una vez validado comunica al servicio a consumir que el usuario es correcto, activando una sesión autenticada durante un periodo de tiempo limitado y nos envía a la página del servicio a consumir.

Ventajas:

- Con una única cuenta podemos acceder a múltiples servicios.
- Limitamos la información que proporcionamos a cada proveedor de servicios incrementando la privacidad.

5.5.- Protocolo Ligero de Acceso al Servicio de Directorios: LDAP.

Permite el acceso a un servicio de directorio ordenado. Es un protocolo basado en el estandar X.500 para compartir directorios organizando la información de forma jerárquica. Una aplicación cliente LDAP se conecta a u servidor LDAP para consultar el directorio.

Un directorio es un conjunto de objetos con atributos organizados de manera lógica y jerárquica. El estandar LDAP define cuatro modelos:

- **Modelo de información**. Describe la estructura de la información.
- **Modelo de nombres**. Describe cómo se organiza la información.
- **Modelo funcional**. Describe las posibles operaciones.
- **Modelo de seguridad**. Describe como se protege la información.

Active Directory es el nombre que utiliza Windows Server como almacén centralizado de información para sus dominios y está basado en LDAP.

Podemos validar el acceso a una web mediante un servidor LDAP utilizando PHP. Para ello debemos habilitar el soporte LDAP en el fichero php.ini del servidor web. Para ello debemos quitar el ; que comenta la línea:

```
extension = php_ldap.dll
```

y a continuación reiniciar el servidor web para que vuelva a cargar el fichero php.ini

Algunas de las funciones PHP para la conexión con LDAP son:

Función	Acción
ldap_connect()	Establece la conexión con el servidor LDAP.
ldap_bind()	Autentifica al usuario mediante usuario y passwod.
ldap_close()	Cierra la conexión con el servidor LDAP.

5.6.- Prueba y Depuración.

Los problemas habituales en el desarrollo de software vienen por desviaciones en plazos y aparición de fallos durante la implantación y mantenimiento del software.
Las pruebas de software o testing son aquel conjunto de procesos que permite verificar y validar la calidad del producto identificando errores de diseño e implementación (No hace lo que se pedía o lo hace de forma incorrecta).

Suelen comenzar en el momento que comienza el desarrollo y continúan hasta que finaliza el mismo.

El Plan de pruebas permite planificar perfectamente el proceso de testing. Consiste en un conjunto de herramientas, técnicas y métodos que intentan descubrir errores.

Las razones por las que se producen los errores de programación suelen ser:

- Falta de comunicación entre las partes que intervienen en el desarrollo del software.
- Complejidad del desarrollo.
- Exceso de confianza de los programadores.
- Cambios continuos en el desarrollo con rediseños y replanificaciones.
- Plazos de entrega hacen en ocasiones omitir fases de pruebas y control.
- Pobre documentación del código dificulta las modificaciones.

EL tester es la persona que realiza las pruebas y somete al software a diferentes carga de datos de entrada, condiciones del sistema,... con la intención de romper el software y encontrar errores que no se ven a simple vista.

Las pruebas siempre deberán realizarse en un entorno de pruebas y no en un entorno de explotación o real, siendo diferente las personas que prueban el software de las que hicieron el desarrollo.

Existen dos tipos de pruebas:

- **Pruebas de Verificación.** Consiste en demostrar que el programa cumple sus especificaciones. ¿Hace lo que debe?
- **Pruebas de Validación.** Se encargan de comprobar que el programa de la respuesta que espera el usuario. ¿Lo hace bien?

Las pruebas que se suelen realizar son:

- Caja negra.
- Caja blanca.
- Unidad de testeo.
- Integración Incremental.
- Prueba de Integración.
- Prueba de sistema.
- Prueba funcional.
- Prueba de fin a fin.
- Prueba de regresión.
- Prueba de aceptación.
- Prueba de carga.
- Prueba de estrés.
- Prueba de instalación y desinstalación.
- Prueba de seguridad.
- Prueba de recuperación.
- Prueba de compatibilidad.
- Prueba de exploración.
- etc...

Práctica. Indicar en 30 o 40 líneas en qué consiste y las principales características de la prueba de la "caja blanca" y la "caja negra".

Las pruebas pueden ser de dos tipos: manuales y automáticas. En la pruebas manuales se realizan de forma manual por la persona que realiza el testing o por el usuario final si son pruebas de aceptación.
La realización de las pruebas se realizará bajo varios navegadores y verificando la correcta funcionalidad, siendo satisfactorias en caso de mostrar lo esperado o procediendo a corregir las anomalías en caso contrario.

Si los módulos a probar son pequeños podemos utilizar sentencias echo, print_r o var_dump para mostrar valores de la variables y detectar posibbles errores o ir comentando el código poco a poco para detectar el error.

Si el volumen de código es muy grande utilizaremos herramientas como depuradores de código y analizadores de rendimiento, además realizaremos pruebas automatizadas mediante el uso de herramientas para la creación y ejecución de pruebas.

La Herramienta PHPUnit permite realizar pruebas unitarias sobre el código PHP. Los "Casos de Prueba" en PHPUnit deben ser programados para poder ser reutilizados y por convenio el nombre del caso de prueba debe anteponer la palabra Test al nombre de la clase o módulo que se pruebe.

Por ejemplo, si la clase que queremos probar se llama Usuario, el Caso de Prueba de llamará TestUsuario. La clase de prueba extiende de la clase TestCase.de PHPUnit.

Además, los métodos de prueba deben ser públicos y empezar con la palabra “test”.

En PHPUnit, una Aserción es una afirmación que se considera cierta y son muy útiles para la configuración de pruebas. Disponemos de los siguientes tipos de aserciones:

Aserción	Significado
AssertTrue/AssertFalse	Comprueba si la entrada es igual a true/false.
AssertEquals	Comprueba el resultado frente a otra entrada.
AssertGreaterThan	Comprueba el resultado para ver si es mayor que un valor
AssertLessThan	Comprueba el resultado para ver si es menor que un valor
AssertGreaterThanOrEqual	Comprueba si el resultado es mayor o igual que un valor
AssertLessThanOrEqual	Comprueba si el resultado es menor o igual que un valor
AssertContains	Comprueba que la entrada contiene un valor específico.
AssertType	Comprueba que una variable es de un cierto tipo.
AssertNull	Comprueba que una variable es nula.
AssertFileExists	Comprueba que un archivo existe.
AssertRegExp	Comprueba la entrada con una expresión regular.

Ejemplo:

Instalación de PHPUnit.

- Acceder a http://phpunit.de y pulsar el botón Getting Started.
- Pulsar en "Simply download it from **here**...." para descargar el fichero phpunit.phar.
- Crear en el disco C: la carpeta bin y copiar dentro la librería phpunit.
- Añadir la ruta c:\bin de la librería phpunit a las variables de entorno y la ruta al fichero php.exe de nuestro servidor web que normalmente es c:\xampp\php.
- Instalar en la carpeta bin de c: la librería phpunit abriendo una ventana de comandos cmd en la ruta c:\bin y utilizando la sentencia:

```
echo @php "%~dp0phpunit.phar" %* >phpunit.cmd
```

- Se creará en la ruta un fichero phpunit.cmd
- Probar el funcionamiento de phpunit en la ventana de comandos con la sentencia

```
phpunit --version
```

Crear clase Usuario.

fichero: usuario.php

```
<?php
class Usuario {
    public $nombre;

    public function getNombre() {
        return $this->nombre;
    }

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

    public function saludo() {
        return "Buenos dias!";
    }
}
?>
```

Crear clase UsuarioTest para las pruebas con PHPUnit.

fichero: usuarioTest.php

```
<?php
    //Usamos PHPUnit desde el fichero .phar.
    use PHPUnit\Framework\TestCase;

    // incluimos el fichero usuario.php
    require_once "usuario.php";

    // creamos la clase usuarioTest heredada de TestCase
    class usuarioTest extends TestCase{
        // sentencias que se ejecutarán antes de  cualquier prueba
        public function setUp(){
            $this->usuario = new usuario();
            $this->usuario->setNombre("Sara");
        }

        //Prueba 1
        public function testTalk(){
            $saludoEsperado = "Buenos dias!";
            $saludoActual = $this->usuario->saludo();
            $this->assertEquals($saludoEsperado, $saludoActual);
        }

        //Prueba 2
        public function testGetNombre(){
            $this->assertEquals($this->usuario->getNombre(), "Marta");
        }

        // sentencias que se ejecutarán después de  cualquier prueba
        public function tearDown(){
            unset($this->usuario);
        }
    }
?>
```

Ejecución de las Pruebas.

Para ejecutar los casos de pruebas programados debemos abrir en la carpeta donde tenemos las clases php una ventana de comandos y ejecutar la sentencia:

phpunit nombreclase.php

En nuestro ejemplo:

phpunit usuarioTest.php

Se ejecutan los casos de prueba programados obtenidendo un punto por cada una de las pruebas realizadas correctamente y una F por cada una de las pruebas fallidas, obteniendo el siguiente resultado:

```
C:\xampp\htdocs\test>phpunit usuarioTest.php
PHPUnit 6.4.4 by Sebastian Bergmann and contributors.

.F                    2 / 2 (100%)


Time: 304 ms, Memory: 10.00MB

There was 1 failure:

1) usuarioTest::testGetNombre
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'Sara'
+'Marta'

C:\xampp\htdocs\test\usuarioTest.php:24

FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
```