Curso de **Desarrollo Web** en **PHP orientado a objetos** con **MVC**

Eugenia Bahit, Mayo 2015

14

Cookies y Sesiones

Variables de Sesión

Encriptación, *hashing* con MD5, SHA1 y SHA512

1 Definición de los datos de acceso

Elegir un nombre de usuario y contraseña y generar el *hash* de cada uno con el algoritmo deseado:

```
php> $user = hash('sha1', 'admin');
php> print $user;
D033e22ae348aeb5660fc2140aec35850c4da997
php>
php> $pass = hash('sha512', '123456');
Php> print $pass;
Ba3253876aed6bc22d4a6ff53d8406c6ad864195ed144ab5c87621b6c233b548baeae6956df346ec
8c17f5ea10f35ee3cbc514797ed7ddd3145464e2a0bab413
```

Generar el *hash* final de acceso concatenando los dos *hashes* anteriores y encriptándolos con el mismo algoritmo (generalmente, MD5):

```
php> print hash('md5', $user.$pass);
e162837a3f6e0e0521d6807ff50be3cf
```

Copiar el *hash* final generado.

2 Inicialización de constantes
y variables de sesión en el archivo settings.php

Algoritmos con los que serán encriptados usuario, contraseña y cadena de acceso

```
const TOKEN_USER_ALGO = 'sha1';
const TOKEN_PASS_ALGO = 'sha512';
const TOKEN_ACCESS_ALGO = 'md5';
```

Hash de acceso obtenido anteriormente:

```
const TOKEN_ACCESS = "e162837a3f6e0e0521d6807ff50be3cf";
```

Variables de sesión a utilizarse en el proceso de validación:

```
session_start();

if(!isset($_SESSION['active'])) {
    $_SESSION['active'] = False;
};
```

Los algoritmos soportados por PHP pueden conocerse mediante la función `hash_algos()`

```
php> print_r(hash_algos());
Array
(
    [0] => md2
    [1] => md4
    [2] => md5
    [3] => sha1
    [4] => sha224
    [5] => sha256
    [6] => sha384
    [7] => sha512
    [8] => ripemd128
    [9] => ripemd160
    [10] => ripemd256
    [11] => ripemd320
    [12] => whirlpool
    [13] => tiger128,3
    ...
)
php>
```

3 Definir URLs por defecto
(en el archivo settings.php)

Recurso privado por defecto (solo se accede con *password*):

```
const DEFAULT_PRIVATE_URI = "/producto/agregar";
```

Recurso público por defecto (cualquier usuario puede acceder):

```
const DEFAULT_PUBLIC_URI = "/";
```

URL del recurso que mostrará el formulario de *login*:

```
const LOGIN_URI = "/user/login";  # TODO habrá que crear el recurso
```

4 Crear método estático para validación de acceso

Crear la clase `SessionBaseHandler` en el archivo **`core/sessions.php`**

```
Class SessionBaseHandler { }
```

Agregar el método estático para validar los datos ingresados por el usuario:

```
public static function validar() {
    $user_hash = hash(TOKEN_USER_ALGO, $_POST['user']);
    $pass_hash = hash(TOKEN_PASS_ALGO, $_POST['password']);
    $final_hash = hash(TOKEN_ACCESS_ALGO, $user_hash . $pass_hash);
    if($final_hash == TOKEN_ACCESS) {
        self::iniciar_sesion();
    } else {
        self::destruir_sesion();
    }
}
```

Dentro de una **clase "estática"** se utiliza **`self::`** para llamadas internas **en vez de `$this->`**

5 Crear los métodos para iniciar y finalizar sesiones

```
public static function iniciar_sesion() {
    $_SESSION['active'] = True;
    header("Location: " . DEFAULT_PRIVATE_URI);
}

public static function destruir_sesion() {
    $_SESSION['active'] = False;
    header("Location: " . DEFAULT_PUBLIC_URI);
}
```

La palabra clave static se utiliza para definir métodos (o propiedades) que puedan ser llamados de forma “estática”, es decir, sin necesidad de crear un objeto (instancia de clase):

```
Clase::metodo_estatico();    # Llamada a un método estático

$obj = new Clase();
$obj->metodo_no_estatico();  # Llamada a un método “normal”
```

6 Crear método para verificar la sesión

Este método será llamado desde los recursos que se quiera crear como privados:

```
public static function verificar_sesion() {
    if(!$_SESSION['active']) {
        header("Location: " . LOGIN_URI);
    }
}
```

Para restringir un recurso, agregar la llamada a `verificar_sesion()` al comienzo del método a restringir:

```
SessionBaseHandler::verificar_sesion();
```

Los usuarios que no estén *"logueados"* serán redirigidos al formulario de *login*.

7 Crear el formulario de *Login*

Archivo: static/login.html

```
<form class="form-horizontal" action="/user/validar" id="login" method="POST">
  <div class="form-group">
    <label for="user" class="col-sm-3 control-label">Usuario:</label>
    <div class="col-sm-4">
      <input type="text" class="form-control" id="user" name="user">
    </div>
  </div>

  <div class="form-group">
    <label for="user" class="col-sm-3 control-label">Contraseña:</label>
    <div class="col-sm-4">
      <input type="password" class="form-control" id="password" name="password">
    </div>
  </div>

  <div class="form-group">
    <div class="col-sm-offset-3 col-sm-9">
      <button type="submit" class="btn btn-primary">Enviar</button>
    </div>
  </div>
</form>
```

8 Crear el módulo de usuarios

Archivo: user.php

Modelo

```
Class UserModel {}
```

Vista

```
class UserView {

    function login() {
        $titulo = "Ingreso al sistema";
        $contenido = file_get_contents('static/login.html');
        return array($titulo, $contenido);
    }

}
```

(continúa de pantalla anterior)

Controlador

```
class UserController {

    function __construct() {
        $this->model = new UserModel();
        $this->view = new UserView();
    }

    function login() {
        return $this->view->login();
    }

    function logout() {
        SessionBaseHandler::destruir_sesion();
    }

    function validar() {
        SessionBaseHandler::validar();
    }
}
```