Ejercicio 3:
Estructura del Proyecto

Acerca del ejercicio

En este ejercicio partiremos de una posible solución del ejercicio anterior y organizaremos y mejoraremos el código del proyecto con el objetivo de lograr una estructura adecuada para un proyecto PHP de tamaño medio.

Paso 1: Reorganización de archivos públicos y privados

NOTA PREVIA: Descarga el archivo 01-inicio.zip del campus virtual.

Las convenciones habituales en el desarrollo con PHP (ver PSR-1) establecen que es necesario hacer una distinción en los scripts PHP (al menos) en varios tipos:

- Scripts de vista: aquellos que generan salida para el usuario.
- Scripts de apoyo para vistas: Son scripts que se incluyen exclusivamente en los scripts de vista, normalmente con la finalizar de reutilizar o bien HTML o funciones de apoyo para generar el HTML de las vistas.
- Scripts de lógica / otros scripts: aquellos que exclusivamente contienen definiciones de funciones, clases, etc.

Esta separación, además de permitirnos organizar el código en una primera aproximación, nos permite dar los primeros pasos para poder reutilizar el código entre diferentes proyectos y nos permite restringir el acceso a los usuarios finales a aquellos archivos de nuestra aplicación que no nos interese (e.g. los que tienen datos sensibles como contraseñas de la BD).

Para organizar los scripts también es recomendable crear varias carpetas dentro del proyecto. Al menos deberían de existir las siguientes carpetas:

- includes: Como el propio nombre indica, esta carpeta está pensada para que contenga scripts PHP que se van a incluir (include / require) en otros scripts.
- mysql: En esta carpeta podemos almacenar los diferentes archivos .sql necesarios para crear, borrar y cargar datos de la BD.

Una posible organización para el proyecto podría ser la siguiente.

X:\.....\ejercicio3\

- includes\ : Aquí dejamos los scripts que no son de vista
    - comun\ : Aquí dejamos los scripts de apoyo para las vistas
- mysql\ : Aquí dejamos los .sql
- index.php
- …

El objetivo de este apartado es que:

1. Crees las carpetas necesarias para tener una organización como la anteriormente descrita.
2. Coloques los scripts, atendiendo a su tipo, en las diferentes carpetas.
3. Realices las modificaciones oportunas a los scripts para que la aplicación siga funcionando. Tendrás además que importar la base de datos usando la información incluida en la carpeta mysql.

Adicionalmente, si quieres restringir el acceso a las carpetas a la que no debe tener acceso el usuario final (includes y mysql), puedes crear un archivo ".htaccess" de configuración de Apache y que incluya las directivas de Apache necesarias para restringir el acceso.

Paso 2: Configuración e inicialización de la aplicación

NOTA PREVIA: Puedes seguir con tu propio proyecto, o si te atascas descarga el archivo 02-inicio.zip.

Es habitual que antes de poder procesar la petición del usuario realices algún tipo de inicialización o configuración de la aplicación, por ejemplo, invocar `session_start();` para poder utilizar la sesión del usuario.

Otros aspectos relevantes a configurar son el soporte UTF-8 de PHP para gestionar las peticiones del usuario. El código PHP necesario es:

```
/**
 * Configuración del soporte de UTF-8, localización (idioma y país)
 */
ini_set('default_charset', 'UTF-8');
setLocale(LC_ALL, 'es_ES.UTF.8');
```

Por otro lado, si nuestro proyecto gestiona fechas y horas, es recomendable también indicar la zona horaria de referencia del servidor. El código PHP necesario es:

```
date_default_timezone_set('Europe/Madrid');
```

El objetivo de este apartado es que:

1. Crees el script `includes\config.php` y que incluyas todo el código necesario para inicializar la aplicación antes de procesar la petición del usuario.
2. Modifica los scripts de vista para que como primera instrucción se requiera el archivo includes\config.php y elimina el código que ya no es necesario en estos scripts.

Paso 3: Centralización de la gestión de conexiones a BD y otras operaciones de la aplicación

NOTA PREVIA: Puedes seguir con tu propio proyecto, o si te atascas descarga el archivo 03-inicio.zip.

Si revisas los scripts procesaLogin.php y procesaRegistro.php observarás que hay varias instrucciones PHP que son iguales. Estas instrucciones son las necesarias para crear una conexión a la BD e inicializarla de manera adecuada para que utilice la codificación de caracteres "utf8mb4" necesaria para almacenar cualquier carácter UTF-8 en MySQL.

Por otro lado, hay otro problema y es que los datos de la conexión a la BD (usuario, contraseña, etc.) se incluyen en todos los scripts. Este problema es potencialmente peligroso ya que si existe un error en nuestro script, el usuario final de la aplicación podría llegar a ver los datos de conexión, y si la configuración del servidor no es la adecuada o tenemos un phpMyAdmin instalado el mismo servidor, un potencial atacante podría destrozar nuestra BD.

Finalmente es recomendable ser cuidadosos con la gestión de conexiones a la BD. Las conexiones a la BD requieren bastantes recursos (memoria) en el servidor (tanto en Apache como en MySQL), además el servidor de BD suele tener configurado un límite máximo de conexiones. Finalmente, cabe destacar que el establecer una conexión entre PHP y MySQL requiere de cierto tiempo que hay que añadir al tiempo de ejecución de nuestro código por lo que es interesante no abrir una nueva conexión contra la BD cada vez que queremos realizar una operación contra la misma. Por tanto, es recomendable que por cada petición del usuario se mantenga como mucho 1 conexión por petición.

Una manera de solucionar este problema es crear una clase `Aplicacion` que implemente el patrón Singleton[1] de modo que la única instancia de `Aplicacion` se encargue de gestionar la conexión a la BD.

El objetivo de este apartado es:

1. Crear una clase `Aplicacion` que implemente el patrón Singleton y que su creación sea perezosa.
2. Implementar un método `init($datosBD)` que te permita inicializar la instancia de la aplicación.
    - Se sugiere que `$datosBD` sea un array con todos los datos necesarios para crear una conexión a la BD (e.g. `array('host'=>'xxxx', 'bd'=>'yyyy', 'user'=>'zzzz', 'pass'=>'uuuu')`).

[1] http://www.phptherightway.com/pages/Design-Patterns.html

- Puedes incluir la llamada a session_start() en este método, o dejarla en config.php.

3. Implementar un método `conexionBD()` que se encargue de crear una conexión contra MySQL y devolverla, o si ya existe una devolver la que ya exista.
4. Modificar `procesarLogin.php` y `procesarRegistro.php` para utilizar la clase `Aplicacion` que acabas de crear. Recuerda que debes llamar a `init($datosBD)` en algún momento. Por ejemplo, un buen sitio puede ser en config.php, ya que así nos ahorramos tener que inicializar la aplicación explícitamente en varios scripts diferentes.

Paso 4: Reorganización de la funcionalidad asociada a la gestión de los usuarios

NOTA PREVIA: Puedes seguir con tu propio proyecto, o si te atascas descarga el archivo 04-inicio.zip.

Llegado este apartado del ejercicio, la funcionalidad asociada a la gestión de los usuarios (dejando aparte la gestión de los formularios PHP) en al menos dos scripts `procesarLogin.php` y `procesarRegistro.php`. En una aplicación real esta situación se complicará aún más al añadir otras funcionalidades como la edición del perfil del usuario etc. Además, cambios en los detalles de implementación asociados a los usuarios como un cambio de diseño en la BD o en la gestión de las contraseñas, requeriría tocar varios scripts PHP.

Para evitar esta situación que es poco deseable, la solución consiste en definir la clase Usuario que se encargue de aglutinar toda la funcionalidad propia de la gestión de los usuarios (e.g. acceso a las tablas de la BD de usuarios, gestión de passwords, etc.).

El objetivo de este apartado es:

1. Crear la clase Usuario con la menos los siguientes métodos:
   - `public static function buscaUsuario($nombreUsuario).` Devuelve un objeto Usuario con la información del usuario $nombreUsuario, o false si no lo encuentra.
   - `public function compruebaPassword($password).` Comprueba si la contraseña introducida coincide con la del Usuario.
   - `public static function login($nombreUsuario, $password).` Usando las funciones anteriores, devuelve un objeto Usuario si el usuario existe y coincide su contraseña. En caso contrario, devuelve false.
   - `public static function crea($nombreUsuario, $nombre, $password, $rol).` Crea un nuevo usuario con los datos introducidos por parámetro.
2. Modificar los scripts procesaLogin.php y procesaRegistro.php para que utilicen la clase Usuario

Paso 5: Reorganización de la gestión de formularios

NOTA PREVIA: Puedes seguir con tu propio proyecto, o si te atascas descarga el archivo 05-inicio.zip.

Las etapas principales de gestión de un formulario HTML son muy concretas, de modo que es posible utilizar el patrón TemplateMethod[2] para definir esta lógica una única vez y simplificar la creación y gestión de formularios de toda la aplicación.

Descarga el archivo 05-Form.zip del CV y descomprímelo dentro de \includes para que puedas utilizar la clase Form dentro de tu aplicación. El método plantilla es el método `public function gestiona()` de dicha clase. Es recomendable que analices el código de éste y el resto de métodos (especialmente `generaCamposFormulario()` y `procesaFormulario()`) para que comprendas como funciona, y en particular, como se gestionan los errores del formulario.

Para utilizar esta clase Form dentro de tu aplicación, además de crear una subclase por cada formulario que tengas que gestionar en la aplicación, también es necesario cambiar la metodología de trabajo. Llegado este punto del ejercicio la mayoría de scripts de vista están organizados por pares (login, procesarLogin), (registro, procesarRegistro). Al utilizar las subclases de la clase Form ya no es necesario utilizar los scripts procesarXXXX sino que el mismo script gestiona las peticiones GET (cuando le usuario lo visita por primera vez) o las peticiones POST (cuando el usuario envía el formulario).

El objetivo de este apartado es:

1. Crear la clase FormularioLogin que hereda de Form e incluye toda la lógica de gestión del formulario. La creación de esta clase debería de ser inmediata a partir de los scripts login.php y procesarLogin.php, simplemente debes copiar el código / HTML que sea necesario a la nueva clase.
2. Crear la clase FormularioRegistro que hereda de Form e incluye toda la lógica de gestión del formulario. La creación de esta clase debería de ser inmediata a partir de los scripts registro.php y procesarRegistro.php, simplemente debes copiar el código / HTML que sea necesario a la nueva clase.
3. Modifica los scripts login.php y registro.php para que utilicen las clases que acabas de crear.
4. Elimina los scripts procesarLogin.php y procesarRegistro.php que ya no son necesarios.

[2] https://sourcemaking.com/design_patterns/template_method

Paso 6 (Opcional): Reducción de include / require en los scripts

Como resultado de la reorganización en archivos separados entre los diferentes tipos de script y con mayor énfasis en el caso de las clases, el número de instrucciones require_once es elevado.

Aunque no es demasiado problemático, sí que resulta incómodo a la hora de desarrollar y desplegar la aplicación. Una opción podría ser hacer un require_once por cada clase en el archivo includes\config.php de la aplicación y suponer que no es necesario hacer ningún require_once dentro de los .php asociados a nuestras clases. Esta solución tiene dos problemas principales:

- Es frágil. Si se nos olvida incluir el require_once la aplicación fallará estrepitosamente.
- Es ineficiente. Debemos recordar que PHP es un lenguaje interpretado y, aunque existen optimizaciones, por cada petición que ejecuta debe analizar todo el código PHP de nuestros scripts. Normalmente no es necesario utilizar todas las clases para poder implementar una funcionalidad de nuestra aplicación, por tanto, el motor de PHP tiene que trabajar más de lo necesario.

El segundo problema podría evitarse realizando require_once selectivos, pero el primer problema sólo podríamos mitigarlo parcialmente.

Por suerte PHP tiene un mecanismo que nos permitirá hacer require_once selectivos según hagamos referencias a nuestras clases PHP. Se trata de la función spl_autoload_register. Todavía mejor, existe la convención PSR-4 que nos proporciona unas pautas a la hora de definir nuestras clases de modo que si las seguimos podemos utilizar incluso utilizar una función de carga de clases ya probada[3].

El objetivo de este apartado es:

1. Añade una sentencia namespace a todas tus clases de modo que todas estén dentro del espacio de nombres es\fdi\ucm\aw.
    - Nota: La sentencias `namespace` y `use` tienen una utilidad similar a las sentencias `package` e `import` de Java respectivamente.
2. Incluye dentro de includes\config.php una función de carga de clases, utiliza como referencia el ejemplo proporcionado en PSR-4[4].

[3] https://www.php-fig.org/psr/psr-4/examples/

[4] https://www.php-fig.org/psr/psr-4/examples/#closure-example

- Nota: presta particular atención a las variables $prefix y $base_dir del código.

3. Elimina los require_once que ya no son necesarios en todos los scripts.

Entrega del trabajo

Una vez terminado el ejercicio, comprime todos los archivos y añade un archivo TXT con los nombres de los alumnos que entregan el trabajo (o con tu nombre si has hecho el ejercicio en solitario). Para que una entrega sea calificable debe contener el código resuelto hasta el apartado 5 incluido, y opcionalmente el apartado 6 para subir nota.

Este archivo comprimido debe tener como nombre **Apellido1Nombre1-Apellido2Nombre2.zip** y se debe subir a la entrega habilitada en el campus virtual (fecha límite 18 de abril, 23:55).

Si el ejercicio está hecho en pareja, **sólo uno de los miembros debe subir el ejercicio**.