Informe Refactorización

Sistema de identificación y registro

En este documento se especifican los cambios realizados al código de la aplicación para su mejor entendimiento ademas de las nuevas funcionalidades integradas.

## Fecha: 18 de Febrero de 2015

## Autor: Ricardo Sánchez

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Índice

[Requerimientos de funcionamiento del sistema 1](#_Toc329186826)

[Code Smells encontrados 2](#_Toc329186827)

[Patrones de diseño aplicados 3](#_Toc329186828)

[Problemas Encontrados 3](#_Toc329186828)

[Conclusión 4](#_Toc329186829)

# Funcionalidades Requeridas

## Registro e identificación

* Vía consola de comandos: Se requiere que los usuarios puedan registrarse e identificarse via consola
* Via web: Se requiere que los usuarios puedan registrarse e indetificarse via web mediante un formulario. En este formulario se enviaran los datos a la aplicación por medio de una request POST por temas de seguridad.

## Cifrado de Contraseña

A continuación se especifican dos sistemas de cifrado de contraseña:

* Cifrado mediante el nuevo sistema de HASHING de php 5.5: Se ha implementado esta nueva funcionalidad para el cifrado de las contraseñas de los nuevos usuarios que se registren en la aplicación
* Cifrado mediante el sistema original ya implementado: Se ha mantenido dicha funcionalidad para permitir a los antiguos usurios poder identificarse sin tener que modificar el cifrado de sus actuales contraseñas.

# Code Smells detectados

## Descripcion

En este apartado se describen todas las partes del código que se han considerado como malas prácticas ya que ensucian el código, dificultan su comprensión y dificultan aumentar las funcionalidades de la aplicación.

## Nombres de variables poco descriptivos

Se han detectado nombres de variables con nombres poco descriptivos, lo que conduce a una gran dificultad a la hora de entender el código

Ejemplo:

**if**($check){

**echo** "<div>Access granted.</div>";

}

El nombre de la varible check es poco descriptivo y no sabemos que comprueba,

Una buena solución sería renombrarla como “isValidUser”.

**if**($isValidUser){

**echo** "<div>Access granted.</div>";

}

## Excesivos Comentarios

Los Comentarios En el codigo no termina de ser una mala practica del todo si es para casos que consideramos que es dificil enter una parte del codigo, pero por norma general y debemos utilizar comentarios es por que el nombre de nuestras variables, metodos o clases no son suficientemente descriptivas, ademas de hacer el programador utilizar mas tiempo para tener que leer todos esos comentarios.

La solución aportada ha sido eliminar todos los comentarios y crear un código que sea facil de interpretar.

## Uso excesivo de hardcodeos (Magic numbers y Magic Strings)

En muchas partes del código se dejan texto o números donde los cuales se repiten o si hay que modificarlos es incomodo tener que ir al código, y ver donde se usan y cambiarlos.

Ejemplo:

$host = "localhost";

$db\_name = "practica";

$username = "root";

$password = "root";

En este ejemplo la solución aportada pasa por sacar los parámetros de conexión a un fichero de configuración donde si fuera necesario cambiarlos, de una sola vez podríamos aplicarlos a todas las partes de nuestro programa donde sean necesarios.

Se han dado otras situaciones como texto y números puestos dentro del código directamente, la solución ha pasado por sacarlos a constantes en la parte superior del fichero.

## Bloques de condicionales demasiado grandes

En esta situación nos hemos encontrado con bloques muy grandes de condicionales, eso suele pasar por que se hacen demasiadas cosas dentro de ellos:

Ejemplo:

**if**($num==1){

$row = $stmt->fetch(PDO::FETCH\_ASSOC);

$storedPassword = $row['password'];

$salt = "ilovecodeofaninjabymikedalisay";

$postedPassword = $\_POST['password'];

$saltedPostedPassword = $salt . $postedPassword;

$hasher = **new** PasswordHash(8,**false**);

$check = $hasher->CheckPassword ($saltedPostedPassword,$storedPassword);

**if**($check){

**echo** "<div>Access granted.</div>";

}**else**{

**echo** "<div>Access denied. <a href='login.php'>Back.</a></div>";

}

}

En este ejemplo la solución aportada pasa por sacar poner cláusula de guarda donde comprobar al principio de lo condicionales para asegurarnos que no nos afectaran durante la ejecución del resto del programa, además de separar la lógica en pequeñas funciones fuera del condicional, de esta manera quedaría todo muchas más legible.

## Código Duplicado

En algunas partes del código se puede ver código duplicado.

Ejemplo:

// $check variable is false, access denied.

**else**{

**echo** "<div>Access denied. <a href='login.php'>Back.</a></div>";

}

}

// no rows returned, access denied

**else**{

**echo** "<div>Access denied. <a href='login.php'>Back.</a></div>";

}

Pasa solucionarlo se ha procedido a sacar el código duplicado a métodos

Public function printError(){

**echo** "<div>Access denied. <a href='login.php'>Back.</a></div>";

}

## Concentración de responsabilidades

Las Ficheros de Login y Register contienen casi toda la lógica de la aplicación. Según el principio “SRP” cada una debería tener una única responsabilidad

La solución ha pasado por extraer la lógica en clases donde cada una tiene usa única responsabilidad. Separando de manera individual las siguientes clases e interfaces:

* IRequestMethod-> Esta interfaz se encarga de definir los métodos de las clases que se encargaran de manejar el tipo de request y así saber como tratar los datos recibidos del usuario.
* User -> Clase Abstracta de la cual heredaran tipos de usuarios específicos en función del tipo de persistencia que se quiera tratar, en nuestro caso solo contemplamos la base de datos.
* PasswordValidation -> Clase encargada de tratar las contraseñas de manera aislada, encargándose de la validación y cifrado de estas.
* PdoAdapter -> Clase encargada de manejar la persistencia a la base de datos de manera aislada al resto de la aplicación.

# Patrones de diseño aplicados

## Factory Pattern

En el diseño de la aplicación se ha querido diseñar una clase que tratara las solicitudes del usuario donde se pudiera manejar los datos de entrada y los datos de salida una vez haber comprobado los datos de usuario,

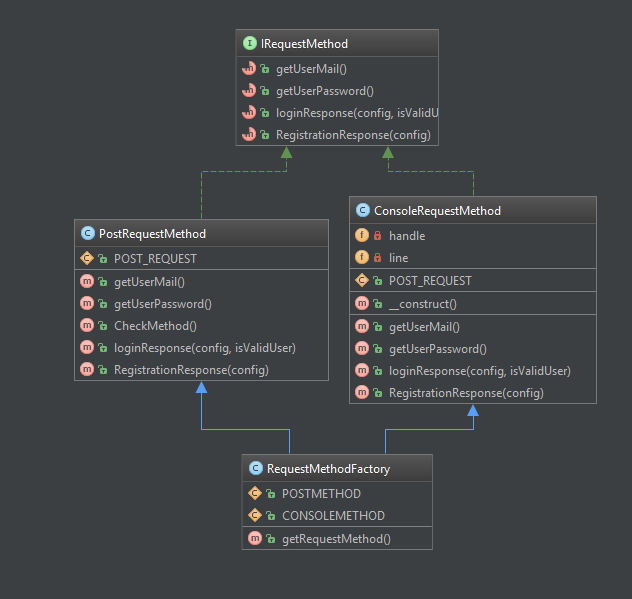
Como uno de los principales requisitos, se detallaba que los usuarios podrían registrarse e identificarse tanto por consola como por web, de esta manera surgía la necesidad de identificar de alguna forma de donde provenían los datos para poder recogerlos de la mejor manera y además de saber cómo poder devolver una respuesta acorde.

Tras estudiar diferentes posibilidades se decidio por usar el patron “factory” ya que a traves del metodo de php “php\_sapi\_name()” se puede identificar de donde proviene el tipo de request y asi saber como solicitar los datos al usuario.

Esta clase Factory se encarga de devolvernos un objeto:

- “PostRequestMethod” (Encaragos de repoger una request por el metodo “POST”)

- ConsoleRequestMethod (Encargado de reoger una request por consola)

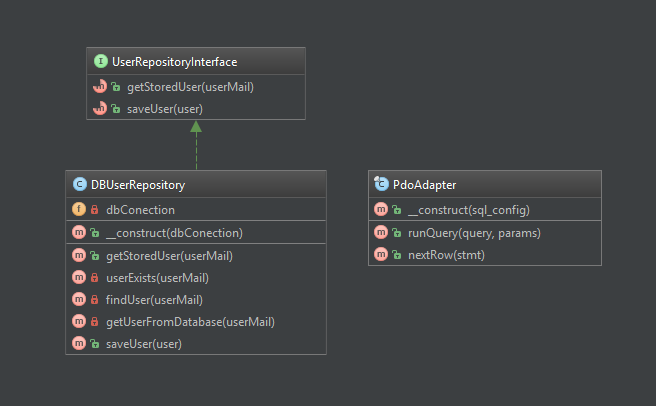
Esta es la estructura diseñada: 

## Repository Pattern

El patrón repositorio está íntimamente relacionado con el acceso a datos y nos permite tener una abstracción de la implementación de acceso a datos en nuestra aplicación, de modo que nuestra lógica de negocio no conozca ni esté acoplada a la fuente de datos.

Esta solución surge de la necesidad de utilizar la Librería PDO para la persistencia de datos en Mysql pero con ampliación de sus funcionalidades y poder definir funciones que cumplieran el contrato con la interfaz del repositorio.

Esta es la estructura diseñada:



# Problemas Encontrados

## Compatibilidad php 5.5 y php 5.4

Según los requerimientos de la aplicación, Se requería utilizar las nuevas funcionalidades de hashing de PHP 5.5 para el cifrado de contraseñas con el inconveniente de que algunos de los servidores que se podrían utilizar aun no dispondrían de esa versión de PHP, y seguirían usando PHP 5.4.

Con esto nos surge el problema de poder compatibilizar las librerías de php 5.5 para hashing de cifrado de contraseñas con servidores con php 5.4

La Solucion elegida para solvertar este problema ha sido incluir ua librería llamada “password\_compat”, descargada de la siguiente dirección:

<https://github.com/ircmaxell/password_compat>

Según su documentación esta librería consigue compatibilizar las nuevas funciones de hashing de contraseñas de PHP 5.5 con versiones a partir de la versión 5.3.7 de PHP.

Para su correcto funcionamiento tan solo es necesario incluir la librería en nuestros ficheros PHP donde vallamos a hacer uso de dichas funciones de cifrado de PHP.

Para comprobar su funcionamiento se han realizado pruebas en un Sistema Operativo CentOs con Apache y PHP 5.4, asegurándonos así de su correcto funcionamiento.

## Uso de Funcionalidad PDO a conexión a Base de Datos

Surgía la necesidad de utilizar una librería de PHP para la persistencia de datos en Mysql, que nos permitiera aislar la parte de persistencia a la del resto de la aplicación.

Se ha debido utilizar la librería PDO que nos aporta PHP a partir de la versión 5.1, pero nos surgía el inconveniente de que cada vez que quisiéramos instanciarla debíamos indicar los datos de conexión a la base de datos además de tener que utilizar las funciones que la misma librería nos proporcionaba la poder lanzar la consultas a las bases de datos, de esta manera si quisiéramos utilizar otra librería para la persistencia no podríamos ya que cada una usa sus propias funciones para lanzar la queries.

La solución ha pasado por extender una clase de la misma librería implementando unos métodos para la inicialización con los parámetros de conexión cargados directamente desde un fichero de configuración además de permitirnos implementar unos métodos personalizados para realizar las propias consultas a la base de datos. Estos métodos estarán definidos por una interfaz que nos ayudaran a asegurarnos que todas las clases que se realicen para la persistencia de datos deban implementar dichas métodos el trato de los datos de usuario.

Las funciones implementadas son:

public function getStoredUser($userMail); //cargar el usuario de la base de datos

public function saveUser(User $user);// guarder un usuario en la base de datos.

# Conclusion