ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA EN SISTEMAS

ASIGNATURA: Construcción y evolución de software **GRUPO:** GR5

FECHA DE ENTREGA: 12/01/2024

INTEGRANTES:

- Daniel Mera
- George Quishpe
- Christian Agila

Documento de Pruebas Unitarias

Contexto: Se busca desarrollar un software que permita a una persona registrar todas las acciones que adquiere, guardando su información como lo es el precio de la acción y la cantidad de acciones compradas.

Para el desarrollo de este proyecto se ha desarrollado una aplicación web utilizando los lenguajes de programación de HTML y PHP, y una base de datos en MySQL que contiene la tabla con el listado de las acciones registradas.

Adicionalmente, hemos aplicado las prácticas de Extreme Programming o XP, por lo cual, hemos desarrollado las siguientes pruebas unitarias para codificar correctamente los métodos que se utilizan para comunicar nuestro sistema con la base de datos.

1. Prueba para obtener la instancia

Al utilizar una base de datos hemos aplicado una clase DAO para realizar la comunicación entre la aplicación web y la base de datos, por lo tanto, iniciamos con una prueba para crear un método que nos permita mantener una sola instancia de nuestra clase DAO y así asegurarnos de que los registros se realizan en una misma instancia.

La prueba creada es la siguiente, donde se crean dos variables de instancias de la clase DAO, que en principio la primera debe de abrir la nueva instancia y para la segunda se debe enviar la misma instancia ya creada. Luego, se verifica con un assert intance off que la primera variable es una instancia de la clase DAO, y después un assert same para comprobar que las dos variables contienen la misma instancia.

```php
public function test_given_twoInstances_when_getIntance_then_same() {
    $instancia1 = AccionDAO::obtenerInstancia();
    $instancia2 = AccionDAO::obtenerInstancia();

    $this->assertInstanceOf(AccionDAO::class, $instancia1);
    $this->assertSame($instancia1, $instancia2);
}
```

Fig. 1. Prueba unitaria 1.

El método creado para que la prueba funcione correctamente es el siguiente, donde primero se comprueba si existe una instancia ya creada en la clase, ya que la instancia es una variable global. Si no existe la instancia se crea una nueva dentro del if, caso contrario no entra al if y simplemente se devuelve la instancia existente.

```php
public static function obtenerInstancia() {
    if (!self::$instancia) {
        self::$instancia = new AccionDAO();
    }
    return self::$instancia;
}
```

Fig. 2. Método DAO 1.

Al ejecutar la prueba podemos comprobar que efectivamente se ha ejecutado correctamente sin ningún error.

```
Usuario@DESKTOP-DTQJLO4 MINGW64 /c/xampp/htdocs/proyecto-1b-cesw (danielBranch)
$ ./vendor/bin/phpunit
 PHPUnit 9.6.15 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 00:00.007, Memory: 6.00 MB

OK (1 test, 2 assertions)
```

Fig. 3. Ejecución de pruebas 1.

2. Prueba para comprobar las acciones

Para poder desplegar las acciones registradas dentro de la base de datos en la aplicación web, es necesario realizar una consulta que nos devuelva el listado de todas las acciones de la tabla en la base de datos. Por lo cual, hemos desarrollado esta prueba en la cual primero se obtiene la instancia utilizando el método creado previamente, y se necesita crear un nuevo método que nos permita obtener todas las acciones de la base de datos. Finalmente se utiliza un assert array para comprobar que efectivamente el resultado obtenido es la lista de las acciones.

```php
public function test_given_anInstance_when_consultAction_then_getArray() {
    $accionDAO = AccionDAO::obtenerInstancia();
    $acciones = $accionDAO->obtenerTodas();

    $this->assertIsArray($acciones);
}
```

Fig. 4. Prueba unitaria 2.

Luego, el método DAO desarrollado para esta prueba unitaria utiliza un try catch para contener los errores que puedan surgir al hacer una consulta a la base de datos, luego utilizamos un método conectarnos a la base de datos y preparamos la consulta sql. Después de ejecutar la consulta retornamos el resultado que obtenemos, y si por algún motivo surge algún error desplegamos el mensaje del error.

```php
public function obtenerTodas(){
    try{
        $result = parent::conectar()->prepare("SELECT * FROM acciones");
        $result->execute();
        return $result->fetchAll();
    }catch(Exception $e){
        die($e->getMessage());
    }
}
```

Fig. 5. Método DAO 2.

A continuación, se muestra también el método para la conexión de la base de datos.

```php
public function conectar(){
    try{
        return new PDO('mysql:host=localhost;dbname=proyecto-1b-cesw','root','');
    }catch (Exception $e){
        die($e->getMessage());
    }
}
```

Fig. 6. Método para conectar a la BD.

Al ejecutar la clase de prueba podemos ver que las dos pruebas creadas hasta el momento se han ejecutado correctamente.

```
Usuario@DESKTOP-DTQJLO4 MINGW64 /c/xampp/htdocs/proyecto-1b-cesw (danielBranch)
$ ./vendor/bin/phpunit
 PHPUnit 9.6.15 by Sebastian Bergmann and contributors.

..                                                                  2 / 2 (100%)

Time: 00:00.013, Memory: 6.00 MB

OK (2 tests, 3 assertions)
```

Fig. 7. Ejecución de pruebas 2.

3. Prueba para registrar una acción

Finalmente, realizamos la prueba para comprobar que las acciones se registrar correctamente en la base de datos. En esta prueba iniciamos obteniendo la instancia de la clase DAO, luego creamos manualmente

una acción a modo de ejemplo para esta prueba, luego tenemos que crear un método que nos permita registrar esta acción y nos devuelva un valor booleano, el cual comprobamos su valor utilizando un assert true.

```php
public function test_given_oneAction_when_register_then_true() {
    $accionDAO = AccionDAO::obtenerInstancia();

    $accion = new Accion("","",0,0);
    $accion->nombre = 'Ejemplo';
    $accion->fechaCompra = '2024-01-11';
    $accion->precioUnitario = 10.5;
    $accion->cantidad = 5;
    $accion->costoTotal = 52.5;

    $resultado = $accionDAO->registrar($accion);

    $this->assertTrue($resultado);
}
```

Fig. 8. Prueba unitaria 3.

Para el método registrar en el DAO utilizamos un try catch para controlar los errores que puedan surgir, luego nos conectamos a la base de datos y preparamos la consulta sql. En esta consulta vamos a enviar los parámetros de la acción que creamos previamente, y finalmente ejecutamos la consulta, la cual nos devolverá un valor booleano con el resultado de la operación.

```php
public function registrar($accion){
    try{
        $result = parent::conectar()->prepare("INSERT INTO acciones (nombre, fechaCompra, precioUnitario, cantidad, costoTotal) VALUES (?,?,?,?,?)");
        $result->bindParam(1, $accion->nombre, PDO::PARAM_STR);
        $result->bindParam(2, $accion->fechaCompra, PDO::PARAM_STR);
        $result->bindParam(3, $accion->precioUnitario, PDO::PARAM_STR);
        $result->bindParam(4, $accion->cantidad, PDO::PARAM_STR);
        $result->bindParam(5, $accion->costoTotal, PDO::PARAM_STR);
        return $result->execute();
    } catch (Exception $e){
       die("Error: No se ha podido registrar la acción " . $e->getMessage());
    }
}
```

Fig. 9. Método DAO 3.

Al ejecutar las pruebas unitarias podemos comprobar que todas pasan con éxito la prueba incluyendo esta última del registro de una acción.

```
Usuario@DESKTOP-DTQJLO4 MINGW64 /c/xampp/htdocs/proyecto-1b-cesw (danielBranch)
$ ./vendor/bin/phpunit
 PHPUnit 9.6.15 by Sebastian Bergmann and contributors.

...                                                                 3 / 3 (100%)

Time: 00:00.052, Memory: 6.00 MB

OK (3 tests, 4 assertions)
```

Fig. 10. Ejecución de pruebas 3.

Adicionalmente, si se desea podemos comprobar en la tabla de la base de datos MySQL creada con el XAMPP que el registro de la acción de ejemplo si se realizó correctamente.

		nombre	fechaCompra	precioUnitario	cantidad	costoTotal
☐	Editar Copiar Borrar	Amazon	2024-01-04	10	5	50
☐	Editar Copiar Borrar	Apple	2024-01-02	50	2	100
☐	Editar Copiar Borrar	Ejemplo	2024-01-11	10.5	5	52.5
☐	Editar Copiar Borrar	Google	2024-01-11	20	10	200
☐	Editar Copiar Borrar	Meta	2024-01-28	140.5	100	14050
☐	Editar Copiar Borrar	Youtube	2024-01-28	1500	2	3000

Fig. 11. Tabla en la base de datos.

4. Pruebas de la API Finnhub

Para poder calcular el valor actual de una acción en el mercado hemos utilizado la API de Finnhub, lo cual se logro utilizando el siguiente método que utiliza un api key y la url del api para poder realizar la búsqueda de la acción solicitada.

```php
private function getPrecioMercadoFromAPI($nombre) {
    $api_key = "cnb4dj9r01qks5iv03pgcnb4dj9r01qks5iv03q0";
    $url = "https://finnhub.io/api/v1/quote?symbol=$nombre&token=$api_key";

    $response = file_get_contents($url);

    $data = json_decode($response, true);

    if ($data && isset($data['c'])) {
        return $data['c'];
    } else {
        return -1;
    }
}
```

Fig. 12. Precio del mercado con la API.

Con esto se modificó también el constructor para que tome el valor que devuelve el api y lo multiplique por la cantidad de acciones compradas. Adicionalmente, se creó el siguiente método para comparar el precio actual con el precio por el que compramos la acción y así obtener el porcentaje de beneficios.

```php
private function calcularCambio() {
    if ($this->gananciaPerdida !== null) {
        $cambioPorcentaje = (($this->gananciaPerdida - $this->costoTotal) / $this->costoTotal) * 100;
        return $cambioPorcentaje;
    } else {
        return null; // Maneja el caso en el que no se pueda obtener la ganancia/pérdida
    }
}
```

Fig. 13. Cálculo del cambio.

Finalmente, basándose en que los dos métodos se utilizan con el constructor de la clase acción, se ha desarrollado una pequeña prueba unitaria para comprobar que se este devolviendo valores correctos tanto para el valor del precio actual y el cambio.

```php
public function test_given_oneAction_when_consultPrice_then_true() {
    $accion = new Accion("AAPL", "2024-02-25", 200, 7);
    $precioMercado =  $accion->getGananciaPerdida();
    $cambio = $accion->getCambio();
    $this->assertNotEquals(-1, $precioMercado);
    $this->assertNotNull($cambio);
}
```

Fig. 14. Prueba unitaria de la API.

Al ejecutar las pruebas usando la consola podemos confirmar que se han ejecutado correctamente todas las pruebas unitarias incluida esta que acabamos de crear.

```
Usuario@DESKTOP-DTQJLO4 MINGW64 /c/xampp/htdocs/proyecto-1b-cesw/codigo (danielBranch)
$ ./vendor/bin/phpunit
 PHPUnit 9.6.17 by Sebastian Bergmann and contributors.

....                                                                4 / 4 (100%)

Time: 00:02.449, Memory: 6.00 MB

OK (4 tests, 6 assertions)
```

Fig. 15. Ejecución de la prueba unitaria.

Y, adicionalmente, podemos observar en la siguiente captura cómo efectivamente se guardan correctamente los valores en la aplicación.

Fig. 16. Evidencia de ejecución de la API.