

AWD Report

by Jack Holland

FILE	1331_JACK_HOLLAND_AWD_REPORT_162370_334344457.DOCX		
TIME SUBMITTED	18-JAN-2019 02:14AM (UTC+0000)	WORD COUNT	879
SUBMISSION ID	99371912	CHARACTER COUNT	5879

Advanced Website Development

Assignment 1

Student: Jack Holland (P149473)

Lecturer: Mike Dacey

Contents

1. PHP Design Patterns	3
1.1 Factory Design Pattern	3
1.2 Singleton Design Pattern	5
2. Testing	6
3. Bibliography	11
4. Appendices	12

1. PHP Design Patterns

1.1 Factory Design Pattern

The factory design pattern is a commonly used design pattern. There are 3 types of implementations that can be used. The simple factory, the abstract factory and the factory method. The factory design pattern is used because it allows for interchangeable classes at runtime[1].

The factory pattern is often used when:

- A class can't be anticipated before the object is created
- The logic must be encapsulated before instantiating complex objects
- To reduce tight coupling between classes

An example of the factory design pattern is one that creates a car then drives to a town.

The first file created is the Factory.php file, this is where the factory is instantiated.

Example:

```
<?php
namespace CarFactory;

class CarFactory
{
    public function createCar(): Car
    {
        return new Car();
    }
}
?>
```

The next PHP file to be created is the Car.php file, in this file the attributes for the object are created. The constructor is also defined and what attributes are needed to create an object. The properties are also set here.

```
<?php
namespace CarFactory;

class Car
{
    // Attributes
    private $_carColour;
    private $_carModel;
    private $_carManufacturer;

    // Constructor
    public function __construct($carColour, $carModel, $carManufacturer)
    {
        $this->_carColour = $carColour;
        $this->_carModel = $carModel;
        $this->_carManufacturer = $carManufacturer;
    }

    // Methods for Properties
    public function GetCarColour()
    {
        return $this->_carColour;
    }
    public function SetCarColour($value)
    {
        $this->_carColour = $value;
    }

    public function GetCarModel()
    {
        return $this->_carModel;
    }
    public function SetCarModel($value)
    {
        $this->_carModel = $value;
    }

    public function GetCarManufacturer()
    {
        return $this->_carManufacturer;
    }
    public function SetCarManufacturer($value)
    {
        $this->_carManufacturer = $value;
    }

    // Methods
    public function TravelToDestination($destination)
    {
        // Add code here
    }
}

?>
```

To use the Factory design pattern, the following code is used:

```
<?php
    $factory = new CarFactory();
    $car = $factory->CreateCar("Red", "Fiesta", "Ford");
    $car->TravelToDestination("Swansea");
?>
```

A new CarFactory object is created, this is used to create new a Car object with the arguments defined in the constructor. The TravelToDestination method can then be used.

1.2 Singleton Design Pattern

The singleton design pattern is a creational design pattern. The design pattern uses a single class to ensure only one object is created.

The singleton design pattern for database connectors and logging systems. The advantage of using the singleton design pattern is that the class can be accessed without being instantiated.[2]

A basic implementation of the singleton design pattern[3]:

```
<?php

class User
{
    // Hold an instance of the class
    private static $instance;

    // The singleton method
    public static function singleton()
    {
        if (!isset(self::$instance)) {
            self::$instance = new __CLASS__;
        }
        return self::$instance;
    }
}

$user1 = User::singleton();
$user2 = User::singleton();
$user3 = User::singleton();

?>
```

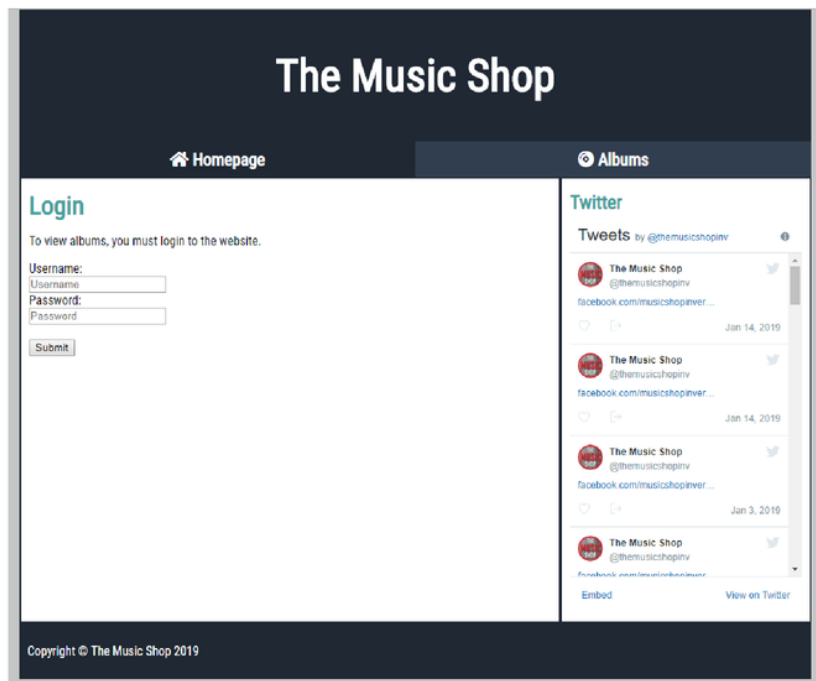
In the example above, the \$user1, \$user2, \$user3 uses the same code to instantiate a new class.

The singleton pattern is considered to be an anti-pattern. This is because it creates a global variable that can be access from anywhere in the code. An anti-pattern is a piece of code or software that is ineffective and bad practice[4].

2. Testing

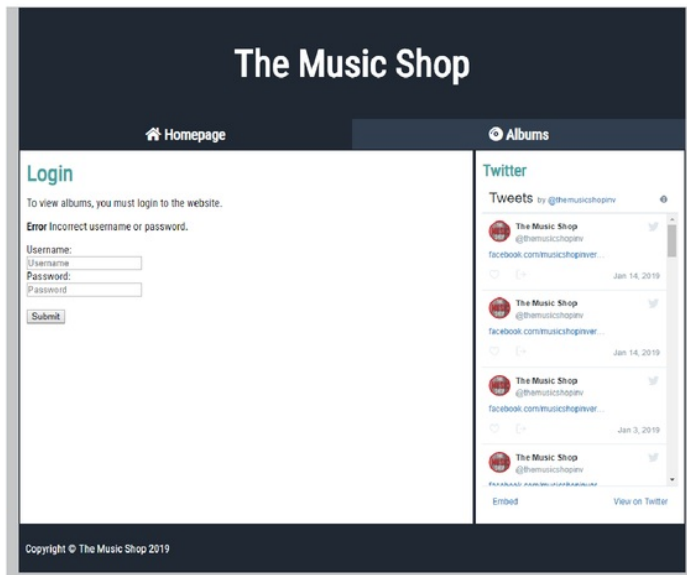
The website scenario is for a website called The Music Shop. It is a small album shop in Swansea which have just expanded to the Internet. They plan to sell a range of popular albums from their shop. To track purchases, users are required to login to the website being able to view their albums and stock.

Testing users having to login before accessing content



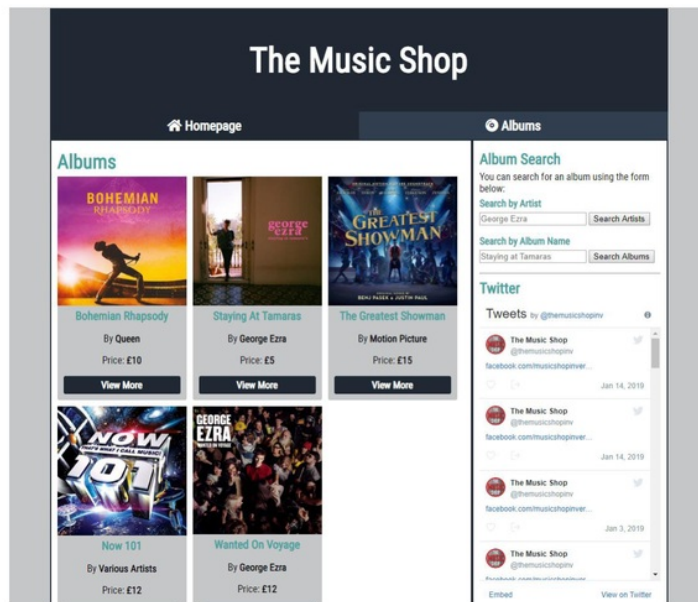
This page is displayed before the user can view the albums page.

Testing incorrect username or password:



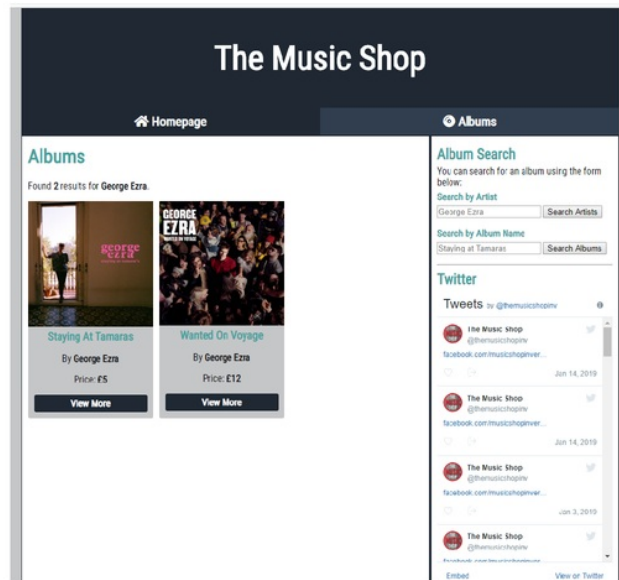
If a user enters an incorrect username or password, they get presented with an error message.

Testing content loading from database



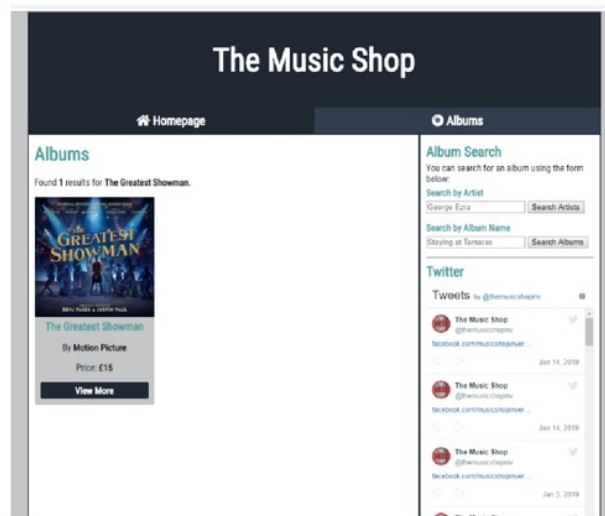
The content is loaded from a MySQL database, the table is called "album" and contains 5 items. The data from the table is added to the Album object.

Searching for an album by artist name



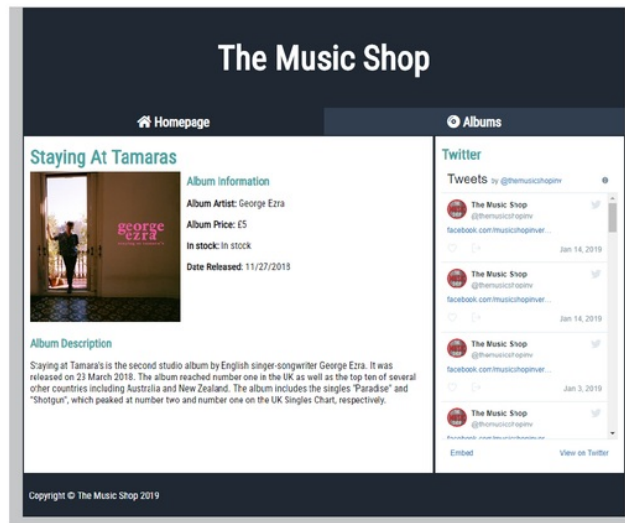
On the sidebar on the right, users can search for albums by artist name. This calls the method `FindAlbumByArtist()`.

Searching for album by album name



Similar to the search above, this calls the method `FindAlbumByArtist()`.

Loading additional information for an album



When the “View more” button is pressed, additional information about the album can be loaded. This is done by using the ID of the album to query the database. The ID is received from the URL using \$_GET.

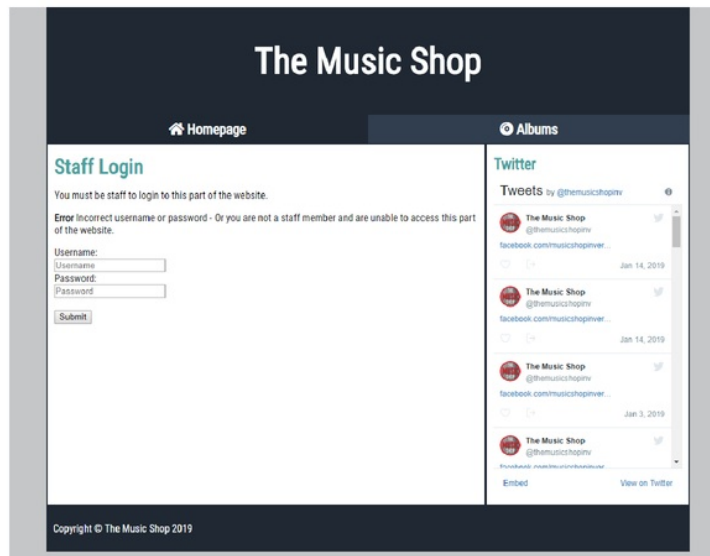
Logging users searches to database

id	page	search_term	username
1	album.php	Now 101	Jack
2	album.php	Staying At Tamaras	Jack
3	album.php	George Ezra	Jack
4	album.php	George Ezra	Jack
5	album.php	George Ezra	Jack
6	album.php	The Greatest Showman	Jack

When a user uses either one of the searches, their search queries are logged in the database. This is done using the method LogToDatabase.

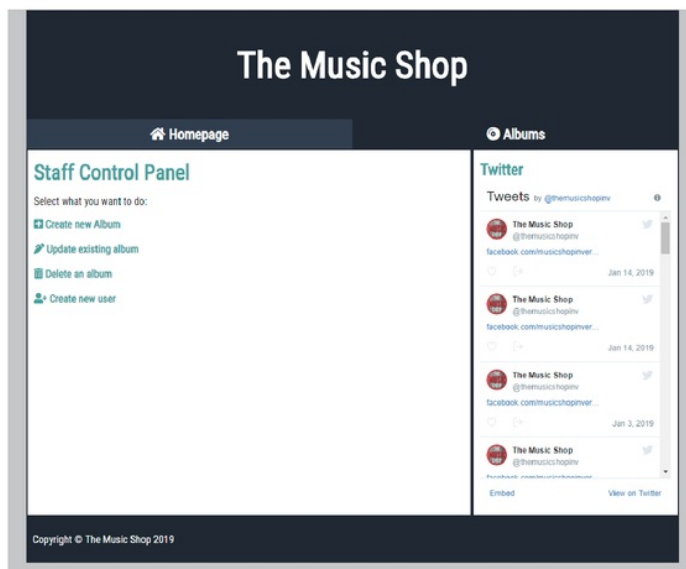
The current page, search term and username are logged.

Attempting to log into staff control panel



The staff control panel is where staff of The Music Shop can edit the website.

Successfully logging into staff control panel



Once logged in, the staff can edit the details of the website. Although this part of the website is unfinished.

3. Bibliography

- [1] "Design Patterns in PHP: Using Factories." [Online]. Available: <https://coderoncode.com/design-patterns/programming/php/coding/development/2014/01/19/design-patterns-php-factories.html>. [Accessed: 17-Jan-2019].
- [2] tutorialspoint.com, "Design Patterns Singleton Pattern," www.tutorialspoint.com. [Online]. Available: https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm. [Accessed: 17-Jan-2019].
- [3] "Design Patterns in PHP: Singletons." [Online]. Available: <https://coderoncode.com/design-patterns/programming/php/development/2014/01/27/design-patterns-php-singletons.html>. [Accessed: 18-Jan-2019].
- [4] "What is an antipattern?," *Agile Alliance*, 31-Mar-2017. .

4. Appendices

PHP Code for Factory Pattern

```
<?php
    namespace CarFactory;

    class CarFactory
    {
        public function createCar(): Car
        {
            return new Car();
        }
    }
?>

<?php
namespace CarFactory;
class Car
{
    // Attributes
    private $_carColour;
    private $_carModel;
    private $_carManufacturer;

    // Constructor
    public function __construct($carColour, $carModel, $carManufacturer)
    {
        $this->_carColour = $carColour;
        $this->_carModel = $carModel;
        $this->_carManufacturer = $carManufacturer;
    }

    // Methods for Properties
    public function GetCarColour()
```

```
{
    return $this->_carColour;
}

public function SetCarColour($value)
{
    $this->_carColour = $value;
}

public function GetCarModel()
{
    return $this->_carModel;
}

public function SetCarModel($value)
{
    $this->_carModel = $value;
}

public function GetCarManufacturer()
{
    return $this->_carManufacturer;
}

public function SetCarManufacturer($value)
{
    $this->_carManufacturer = $value;
}

// Methods
public function TravelToDestination($destination)
{
    // Add code here
}
```

```
}  
?>
```

```
<?php  
    $factory = new CarFactory();  
    $car = $factory->CreateCar("Red", "Fiesta");  
    $car->TravelToDestination("Swansea");  
?>
```

PHP Code for Singleton Pattern

Taken from[3]:

```
<?php  
    class User  
    {  
        // Hold an instance of the class  
        private static $instance;  
  
        // The singleton method  
        public static function singleton()  
        {  
            if (!isset(self::$instance)) {  
                self::$instance = new __CLASS__;  
            }  
            return self::$instance;  
        }  
    }  
  
    $user1 = User::singleton();  
    $user2 = User::singleton();  
    $user3 = User::singleton();  
?>
```

AWD Report

GRADEMARK REPORT

FINAL GRADE

60/100

GENERAL COMMENTS

Instructor

Knowledge & Understanding (50) 26

Quality (20) 14

Login Observer (10) 8

Database Handling (15) 9

Software Testing (5) 3

Description of Factory and Singleton pattern provided but the choice of the Singleton pattern limited the discussion a little.

Basic website can login and search.

No database add, edit or delete.

The database abstraction is a bit odd, the data is copied to a separate class and then searches are conducted against it. This is not really the kind of abstraction that was envisioned.

The observer pattern was not used with login but on the search.

Evidence of testing.

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14