# My report – Brandon Wright psybw7

Contents

[My report – Brandon Wright psybw7 0](file:///D:\CodeDump\DBICW\psybw7_Report.docx#_Toc100163465)

[HTML 2](#_Toc100163466)

[Form 2](#_Toc100163467)

[Select 2](#_Toc100163468)

[Input 3](#_Toc100163469)

[Name 3](#_Toc100163470)

[Submit 4](#_Toc100163471)

[Div 4](#_Toc100163472)

[Link 5](#_Toc100163473)

[Script 5](#_Toc100163474)

[CSS 5](#_Toc100163475)

[Explained 5](#_Toc100163476)

[JS 6](#_Toc100163477)

[GenerateAllowed 6](#_Toc100163478)

[ShowAll 7](#_Toc100163479)

[HasLetters 7](#_Toc100163480)

[Validate 7](#_Toc100163481)

[PHP 8](#_Toc100163482)

[Sanitising the link 8](#_Toc100163483)

[Custom headers (auto generated) 10](#_Toc100163484)

[Connecting to the database 10](#_Toc100163485)

[SQL 10](#_Toc100163486)

[Code explained 10](#_Toc100163487)

[Problems encountered 11](#_Toc100163488)

# HTML

## Form

I used the form keyword to create the very forms you see on the website. With this, I can easily get the user to input data to process in the php sections of other pages. For example, **MovieSearch.php**, contains a form that redirects the user to **MovieSearchAction.php** when the data is validated. This redirect takes the user to the **MovieSearchAction** page and processes the data as intended.

During development and testing, no errors were encountered.

This keyword can be found in:

**ActorAdd.php**

**ActorDelete.php**

**ActorSearch.php**

**MovieAdd.php**

**MovieSearch.php**

**MovieDelete.php**

**Index.php**

## Select

I used this keyword for the landing page. This is because I don’t need the user to type in anything to help them navigate the site. Why not just let them pick an option and go to another part of the site? This is the exact purpose of the select function. This also helps me keep a clean looking site because I’m not using tonnes of buttons clogging up the screen.

During development and testing, no errors were encountered.

This keyword can be found in:

**Index.php**

## Input

The input keyword is pretty useful. This is because it allows me to pick what kind of input is going in. So I can pick something like <input type=”text”> and it will easily make a text field for the user to input data for processing.

### Name

This is what I used to give each input a difference. This came very useful in the **MovieAdd** page. With this I can give meaning to each of the text input fields for processing in the JS and PHP later.

During development and testing, no errors were encountered.

.

This keyword can be found in:

**ActorAdd.php**

**ActorDelete.php**

**ActorSearch.php**

**MovieAdd.php**

**MovieSearch.php**

**MovieDelete.php**

**Index.php**

## Submit

This keyword Is arguably one of the most important keywords I could have used in HTML. Why? Well, it allows me to easily send data to a JavaScript function that processes the data for further processing in a PHP page. This means that I don’t need to program my own button, there’s a prebuilt one that handles all the hard work for me.

During development, no errors were encountered.

This keyword can be found in:

**ActorAdd.php**

**ActorDelete.php**

**ActorSearch.php**

**MovieAdd.php**

**MovieSearch.php**

**MovieDelete.php**

**Index.php**

## Div

Div was used for dividing sections of the page easily. It also helped me centre elements by setting specific CSS classes to them for the correct layout environment.

During development and testing, no errors were encountered.

This keyword can be found in:

**ALL OF THE PHP FILES**

## Link

This was needed to include all of the CSS in the PHP files.

During development and testing, no errors were encountered.

This keyword can be found in:

**ALL OF THE PHP FILES**

## Script

This was needed to include all of the JS in the PHP files and actually have things like validation happen.

During development and testing, no errors were encountered.

This keyword can be found in:

**ALL OF THE PHP FILES**

# CSS

## Explained

I used CSS to give the website some variation and interactive ability. By interactive ability I mean stuff like buttons changing shape, font getting slightly bigger. The interactive ability is there to give context clues to the user for a better user-friendly environment. Hovering over an interactive element and have it respond can help a user realise that the element has purpose. It can also give some non-interactive elements some extra flair. For example, all headers on the site (given class header, not tag header), glow bright. I have shared most of the code across CSS files, this is so individual files can have small changes to them. I chose this over inline CSS to keep the PHP files clean. I have added some responsive CSS, this makes some elements change on a certain zoom percentage. I chose to do this in case anybody needs to view the site on smaller screen devices like mobile phones and tablets. I do recognise that the responsiveness of the text could have been done a bit better since some of the input field text gets cut off.

# JS

## GenerateAllowed

This function’s purpose is to generate an array at the request of the developer. This is to aid the process of validating user input. With the parameter retype, the developer can decide if only number characters are returned of if number characters and letter characters are returned. I chose to have the differing functionality of the one function to avoid duplicate code. A developer may only want numbers for validating user input that regards real numbers, like a year or a price.

Function parameters: retype

Function return type: array

A semantic error was found during the programming of this function. I had put 27 instead of 26 in the following for loop:

      for(i = 0; i < 27; i++){

        x.push(String.fromCharCode(97 + i)); // Using ascii, we can use 97 as a base number and i as an offset for all the lower case characters in the ascii set

        x.push(String.fromCharCode(65 + i)); // Same premise for this line too

      }

This resulted in 2 extra characters that were not wanted. This has now been fixed and the for loop works correctly and looks like the following

      for(i = 0; i < 26; i++){

        x.push(String.fromCharCode(97 + i)); // Using ascii, we can use 97 as a base number and i as an offset for all the lower case characters in the ascii set

        x.push(String.fromCharCode(65 + i)); // Same premise for this line too

      }

## ShowAll

This is a simple function for redirecting the user to the PHP pages responsible for showing the entire table contents of either the table Movie or table Actor.

Function parameters: path

Function return type: N/A

An error was found during this. In the sections regarding the links, I wrote the wrong file path in relation to the current directory. Due to this error being so simple, I choose not to write it in this file. The function can be found in **validate.js**  at lines 21-27.

## HasLetters

This functions detects if a string has letter characters in it or not. This is to help process user input regarding real numbers. We need to make sure that input fields regarding numbers do not have any letter characters in them or any other symbols for that matter. This function generates the number character array and checks the string characters against it. If the code detects a letter character in the string, it returns true, else it returns false.

Function parameters: string

Function return type: boolean

During development and testing, no errors were encountered.

## Validate

This function is the whole bulk of the JS operation. This function incorporates the helper functions, **generateAllowed and hasLetters.** This function takes the inputs from the form and processes them accordingly. This function was originally written for one input since I wasn’t expecting the user needing to put multiple inputs in (you can tell I did the movie adding page last). The first section of the code uses two for loops, the iterator being x since I added it in later to account for multiple values from the user. The function loops through each input on the outer loop and on the inner loop it processes the characters of the string in question, with this, we can see if a string is valid. And since we are looping through all of the values in the form, we are checking the validity of all inputs. The second part regarding the variable **multi** was created later on to account for the integers that the users can put in. This uses the **hasLetters** function to detect if the numbers, well, have letters in them. If we allowed the letters to stay in the numbers, SQL would have a hissy fit about it not being a real number. At the end of the function, we return the variable **allow**, which keep in mind, has been here since the start of the function. **Allow** is a Boolean to keep track of the whole validity of the inputs, if there is **ANY** point that the data isn’t valid, we will return false for the form, which stops the PHP page in question being called.

# PHP

## Sanitising the link

During this coursework, I thought to myself on how people could break the site, do some malicious things to the database. And the thought that popped to mind was the link in the address bar itself. PHP has a lovely feature of including the values passed to a page within the link, but with that comes the ease of editing said link and changing variables text. So what if somebody changed a variable, escaped the string and called DROP TABLE? Well that would be a headache for me and anyone who wanted to use the site. So how did I combat this? Well PHP processes the link (if you tell it to). So before any actual processing, I process the link itself. Here is the function that does this:

    <?php

        function generateAllowed(){

            $x = [];

            for ($i = 0; $i <= 10; $i++) {

                array\_push($x,(string)$i);

            }

            for($i = 0; $i < 26; $i++){

                $a = chr($i + 97);

                $b = chr($i + 65);

                array\_push($x,(string)$a);

                array\_push($x,(string)$b);

            }

            array\_push($x," ");

            return $x;

        }

        // Movie

        // Actor

        // Price

        // Genre

        $title = $\_GET['Movie'];

        $Actor = $\_GET['Actor'];

        $Price = $\_GET['Price'];

        $Year = $\_GET['Year'];

        $Genre = $\_GET['Genre'];

        echo "<title> Adding Movie: " . $title . "</title>";

        $x = generateAllowed();

        $link\_parts = [$title,$Actor,$Price,$Genre];

        // echo "<h1 class='header'> " . $x . "</h1>";

        function parseLink($link,$x){

            $b = true;

            foreach ($link as $elem){

                $split = str\_split($elem);

                foreach ($split as $char){

                    if(!(in\_array($char, $x))){

                        echo $char;

                        $b = false;

                    }

                }

            }

            return $b;

        }

        $b = parseLink($link\_parts,$x);

        if($b == false){

           echo "<div class='errorDIV'>";

           echo "<img src='../../assets/sadpepe.png' class='errorIMG'>";

           echo "<br/> <p> <h1> ERROR: </h1> <br/> Somehow the link has been altered to allow faulty text. Such text can't be processed in the database. Please return. ";

           echo "<br/> <a href='http://avon.cs.nott.ac.uk/~psybw7'> >>>> Return here <<<< </a>";

           echo "</div>";

           die();

        }

    ?>

This code derives from the validate function found in **validate.js.** I copy pasted my code from that file and changed it to work in PHP. This example is for the add movie page, since theres multiple values to worry about. It does the same string checking techniques as the one in validate.js, it even includes the same generate function!

A lot of errors were encountered during this process, but since its PHP I only had error 500 so the EXACT errors im not sure. I believe it was mostly syntax errors from when I copied over the JS to the PHP file and forgot to change something.

## Custom headers (auto generated)

I admit, I’m a bit lazy. So, what better to have the titles of each page automatically generate for me? Well, that’s what I did! And I did it with the following code:

 <?php $title = basename(\_\_FILE\_\_, '.php');

  echo " <div class='header'> <img src='../assets/cozypepe.png'/> <b> ". $title . "</b>  <img src='../assets/cozypepe.png'/> </div>";

  ?>

This simple snippet gets the file name of the current file we are using, but it doesn’t include the php extension. So the data in title is just the filename. This is then echoed into the body as my special header div which creates the title text!

## Connecting to the database

This was done pretty easily! I used things like **connect\_errno**, **mynewsqli,** and **prepare** to do all of the SQL functionality. There was times that calling **execute** removed all of the content from my page. This was discovered to be an error in the SQL syntax used, killing all the PHP process. Note: All the SQL coding used for this can be found in the [SQL](#_SQL) section.

# SQL

## Code explained

1. In **ActorAddAction**

$sql = "INSERT INTO Actor (actName) VALUES ('$title')";

This inserts data into the Actor table, specifically the actName column. We get this data from the previous pages form with the name **title.** The primary key is already handled by the database itself so we don’t need to worry about it.

1. In **ActorDeleteAction**

$sql = "DELETE FROM Actor WHERE actName='$title';";

This, as the name suggests, deletes the specified actor from the database. This uses the WHERE keyword which is essentially a conditional statement. So, we are deleting data from the actor table where the actors name is what the user specifies.

1. In **ActorSearchAction**

  $sql = "SELECT actID, actName FROM Actor WHERE actName='$title'";

This selects the actors ID and their name from the actor table where the actors name is the one specified by the user in the previous page

1. In **AllActors**

$sql = "SELECT actID, actName FROM Actor";

This grabs all the actors from the actor table. It specifically grabs their ID and name.

1. In **AllMovies**

$sql = "SELECT mvId, mvTitle,mvPrice, mvYear, mvGenre FROM Movie";

This grabs all the movies from the movie table. It specifically grabs; the movie ID, the title, the price of the movie, the year of its release and the genre.

1. In **MovieAddAction**

$sql = "INSERT INTO Movie ( mvTitle,actID,mvPrice,mvYear,mvGenre) VALUES ('$title',$Actor, $Price,$Year,'$Genre');";

This SQL adds movies into the database. I did want to let the user put the actors name in as input and have it resolve the ID but I could never get this to work. I tried it by putting **SELECT actID FROM Actor WHERE actName=’$Actor’** in the section where you just see **$Actor** now.

1. In **MovieDeleteAction**

$sql = "DELETE FROM Movie WHERE mvTitle='$title';";

This line is simple, it deletes the movie from the table Movie where the title is equal to the one the user inputted.

1. In **MovieSearchAction**

$sql = "SELECT mvId, mvTitle,mvPrice, mvYear, mvGenre FROM Movie WHERE mvtitle='$title'";

This is very identical to the **all movies**  page, but it only returns the movie to the table if title is equal to the one the user gave as input.

## Problems encountered

I had a couple of problems with the SQL in this coursework. The first being auto increment. It turns out I made actID auto increment in the Movie table, just like how I did in the Actor table. This was easily changed by using:

*ALTER* *TABLE* Movie MODIFY *COLUMN* mvID INT *NOT* *NULL* AUTO\_INCREMENT;

The other problem I had was just in syntax, the simple mistakes. Things like forgetting the character **`** around data that is meant to be a string. Even things like having different cases in my variables in the SQL than I did in the PHP a few line above. Something like trying to use **$actor** when I declared it as **$Actor.**