Web Application Development
Workshop 2
Robert Smith

Week 6 - Lab sheet

Contents

Preamble - Sessions for Login Authentication

While most websites offer some information and features to any user viewing pages in a web client, many features of websites involve private and sensitive data and implement some method of securing website data and features. Two related security concepts are authentication and authorization:

> **Authentication:** determining the identity of the person using the computer system (WHO is the user?)
> **Authorization:** deciding whether or not the user is permitted to access a particular part of the computer system or execute some feature (WHAT is the user permitted to do?)

In this lab, we'll learn to implement the username/password login method of authentication to identify the user, and then use data stored in the $_SESSION, combined with access control logic, to authorize which aspects of the web application a user is permitted to use.

Instructions

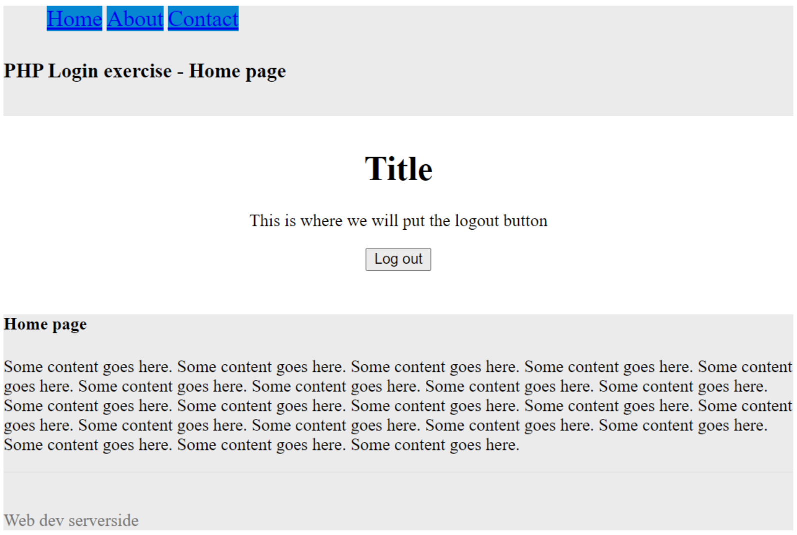
For this lab, students are required to follow the steps to complete each part. Please feel free to copy and paste code where appropriate BUT be sure you understand each part as you will be expected to reproduce similar outcomes in your project.

Submission

Submit files to Moodle link. PHP files should be zipped and in their appropriate directories.

Part 1 – Setting the scene with HTML

I've provided some basic HTML webpages on Moodle. Download and run them on localhost. They should look like the image in Figure 1.



PHP Login exercise - Home page

Title

This is where we will put the logout button

Home page

Some content goes here. Some content goes here.

Web dev serverside

Figure 1 - screenshot of index.php

Open the php scripts in a text editor. Take some time to look around this environment. There are several directories: the *public* directory contains all of the php files that we will be using. The *template* directory contains the header and footer, we will be editing those too.

In the public directory, the *index.php*, *contacts.php*, and *about.php* files are all pretty much the same, with some minor edits. These are the content of our website. The *config.php* file contains a `$username` and `$password` variable. This file pretty much takes the place of a DB, where user data is normally stored. Of course, in the real world, we would use a DB but we can avoid it for now in order to simplify the lesson. Finally, you might have noticed that the *login.php* file is not navigable from the menu on the other pages. This is fine because, in this case, we don't want it to be. Although, if you want to see it, just go to *public/login.php* through localhost in the browser.

Part 2 – Setting up the login logic

Open the *login.php* file in your text editor or IDE.

We are going to use the `$username` and `$password` variables from the *config.php* file. To access that information, we simply use `require_once` at the top of the php file to access the code in *config.php*.

```
<?php
require_once ('config.php'); // This is where the username and
password are currently stored (hardcoded in variables)
?>
```

Next, we will test the user input against the variables we have stored in *config.php*. If the username is ‘Steve’ (uppercase ‘S’) and the password is ‘pass’, then we will print a success message to the screen, if the login credentials don’t match, we will tell the user that the login details are incorrect.

Under the form in the *login.php* file we will add the following PHP:

```
<?php

/* Check if login form has been submitted */
/* isset — Determine if a variable is declared and is different than
NULL*/
if(isset($_POST['Submit']))
{

    /* Check if the form's username and password matches */
    /* these currently check against variable values stored in
config.php but later we will see how these can be checked against
information in a database*/
    if( ($_POST['Username'] == $Username) && ($_POST['Password'] ==
$Password) )
    {
        echo 'Success';
    }
    else
        echo 'Incorrect Username or Password';
}
?>
```

After you have read through this code and understand it, save your progress and run the *login.php* file in the browser. If you enter ‘Steve’ into the Username textbox and ‘pass’ into the Password textbox, then click ‘Sign in’, you should get the word ‘success’ to print under the form. If the login credentials are incorrect, you should get ‘Incorrect Username or Password’.

At this point, you might consider putting the code above into its own function and calling it where required.

Excellent, we now have a form that will test if the inputted username and password matches those that we have stored. However, this doesn’t really do much, we can still go to *public/index.php* in the browser, or any of our other webpages for that matter, i.e., we can check for a username and password, but we cannot block users access to other pages, even if their password is incorrect.

This is where **sessions** come in. See part 3.

Part 3 – Setting up our sessions

Refer to the lecture to understand how sessions work. For now, let's just say that the `$_SESSION` superglobal array allows us to store information between webpages. So, we can save data to a 'session' in *index.php*, let's say a password, and we can grab that later from *contact.php*.

Remember `$_POST` and `$_GET`? These are superglobals, just like `$_SESSION`. So, we can access the information in the `$_SESSION` array, just like we did with those.

And remember, these are all just associative arrays… so we treat them that way.

Important things to remember with sessions:

- If we use sessions in a webpage, we must use `session_start();` at the top of that webpage.
- We can call a session whatever we like, just like variables or arrays, e.g.,
    - `$_SESSION['Username'] = 'Robert'; //this session is called 'Username' and stores the string 'Robert'.`
    - `$_SESSION['Active'] = true;  //this session is called active and stores the Boolean 'true'.`
- We can set session values or grab session values on any page that begins with `session_start()`. For example, if the username session was set in the *index.php* page, we could echo its contents on the *contacts.php* page as follows,
    - `echo $_SESSION['Username'];`
- That's pretty much it: start a session, set a session value, grab the session value.

OK, so lets use some session data: Open *login.php* and setup a session to store the username. To do this, we must first start the session. Remember, we MUST use `session_start()` at the beginning of EVERY page in which we use the session. Luckily, we are using a simple template and we require the *header.php* file at the beginning of every webpage. So, we simply add `session_start()` to the top of the header as follows:

```
<?php session_start(); ?>

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <link rel="stylesheet" type="text/css"
href="../css/stylesheet.css">
```

header.php is 'required' at the beginning of every page in our website. Now, because we've added this code to the *header.php* file, `session_start()` will run in all of our pages too.

Next, we need to create a session called username and set its value to be whatever is stored inside the `$username` variable (which, if you remember, is in *config.php*). We want to do this in *login.php*, where we are checking the user input against the stored values. Add this line to your code:

```
if( ($_POST['Username'] == $Username) && ($_POST['Password'] ==
$Password) )
{
    echo 'Success';
    /* Success: Set session variables and redirect to protected page
*/
    $_SESSION['Username'] = $Username; //store Username to the session
}
else
    echo 'Incorrect Username or Password';
```

Okay, now we have a session and we have stored a value in it. Let us test it by outputting the session value in the *index.php* file. Open the *index.php* file and edit the `<h1>` tag in the 'mainarea' div to `echo` the session value as shown below.

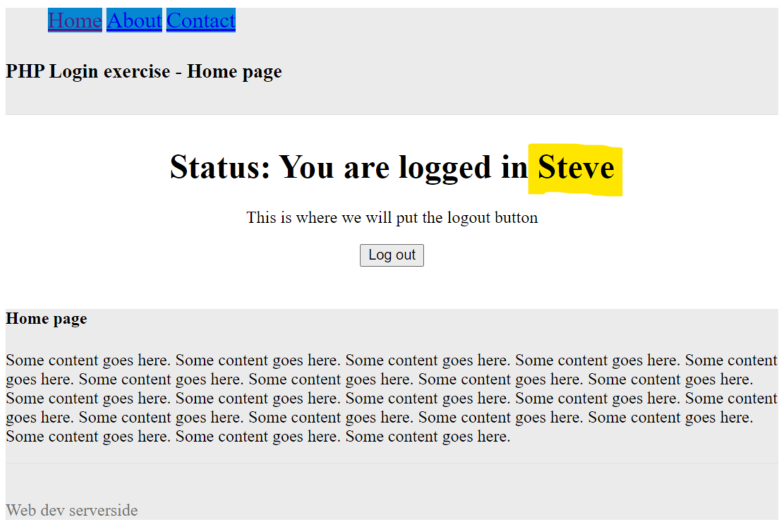
```
<div class="mainarea">
    <h1>Status: You are logged in  <?php echo $_SESSION['Username'];?>
</h1>
    <p class="lead">This is where we will put the logout button</p>

    <form action="" method="post" name="Logout_Form"
class="form-signin">
        <button name="Submit" value="Logout" class="button"
type="submit">Log out</button>
    </form>
</div>
```

Repeat this step for the *about* and *contacts* pages too.

Note that there is no link to the index.php page from the login.php page, you could easily add a link now, or just navigate to public/index.php on the localhost.

Why not try to change the username value in the *config.php* file to see a different output.

Great, now we can see how a simple session works. We created a session in *login.php*, in it we stored information from *config.php*, now we can output the `$username` value 'Steve' on any page we like.


Home About Contact

PHP Login exercise - Home page

Status: You are logged in Steve

This is where we will put the logout button

Home page

Some content goes here. Some content goes here.

Web dev serverside

Figure 2 Screenshot of index.php with $username 'Steve' displayed.

Part 4 – Redirecting the user

At the moment, when the user logs in, they get a simple message indicating success or not. Instead of this, we will redirect them to the *index.php* page. This is the type of interaction that a user expects. This requires use of the **header()** function to redirect the user to the index.php page. In the if statement where we check the user's username and password details, we add our redirect, as below:

```
if( ($_POST['Username'] == $Username) && ($_POST['Password'] ==
$Password) )
{
~~    echo 'Success'; //there's no need for this now, as it will never
be seen~~
    /* Success: Set session variables and redirect to protected page
*/
    $_SESSION['Username'] = $Username;

    header("location:index.php"); /* 'header() is used to redirect
the browser */
    exit; //we've just used header() to redirect to another page but
we must terminate all current code so that it doesn't run when we
redirect
}
else
    echo 'Incorrect Username or Password';
```

This works a treat; our login page now redirects the user to the *index.php* page when the login credentials are correct. We are not done yet, however, as the webpages are still accessible without logging in. We need to write some code to block the user from our web pages unless they are logged in successfully.

Blocking access to web pages

We can do this using **$_SESSION** and a redirect with **header()**. We will create a new session called 'active' that we set to 'true' if the user logs in correctly. Therefore, if the user is not logged in correctly, the value stored in the session will be 'false'. We will use this session value in other pages to block users. OK… let's begin with creating the session. As per the code block below, add a line of code to the if statement in *login.php*. Remember, this session is only created if the username and password are correct.

```
if( ($_POST['Username'] == $Username) && ($_POST['Password'] ==
$Password) )
{
    echo 'Success';
    /* Success: Set session variables and redirect to protected page
*/
    $_SESSION['Username'] = $Username;
    $_SESSION['Active'] = true; //remember we can call a session what
we like e.g. $_SESSION["newsession"]=$value;
    header("location:index.php"); /* 'header() is used to redirect
the browser */
    exit; //weve just used header() to redirect to another page but
we must terminate all current code so that it doesnt reun when we
redirect
}
else
    echo 'Incorrect Username or Password';
```

On the top of all our pages, we will write a small bit of logic that will check to see if the value of `$_SESSION['Active'] == false`. If the value is false, then the user is NOT logged in and our logic will redirect them to the *login.php* page. Remember, we always want to write code using the DRY (Don't Repeat Yourself) principle. As such, the most efficient place to write this piece of logic is in the *header.php* file because this is loaded at the beginning of every page. Edit the top of the header.php file as follows:

```
<?php
session_start();
if($_SESSION['Active'] == false){ /* Redirects user to Login.php if
not logged in. Remember, we set $_SESSION['Active'] == true in
login.php*/
    header("location:login.php");
    exit;
}
/*the code inside these php tags (i.e. the 5 lines of code above) are
required for every page you wish to be accessible only after login*/
?>
```

Excellent, this works.. or does it? If you login and then navigate back to the login page, you will get an error. This is because we are using the same *header.php* file for *login.php* as we are using for all the other pages. Remember, in the header, we redirect to the login page. It's a bad idea to redirect to the *login* page from the *login* page. The result is a type of infinite loop. To avoid this error, just create a new header or put the HTML `<head>` etc.. directly into the *login.php* file. E.g.

```
<?php
require_once ('config.php');
session_start();
?>

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" type="text/css" href="../css/signin.css">
   <link rel="stylesheet" type="text/css" href="../css/stylesheet.css">
    <title>Sign in</title>
</head>

<body>
…
```

Note: there is also a separate css file for the login page called signin.css

Part 5 – Logout

Excellent work, at this point the user can only access the web pages when they are logged in. Now let's allow them to log out. We can do this by destroying the sessions (`session_destroy()`). You will note that there is already a logout button on each page, but it doesn't do anything yet. Let's make it do something. We will write a logout script and then call it when the logout button is pressed.

Create a new file called *logout.php* in the public directory, and add the following lines of code:

```
<?php
  session_start(); /* First we must start the session */
  session_destroy(); /* Destroy started session */

  header("location:login.php");  /* Redirect to login page */
  exit;
```

Next edit the form on the *index.php, contacts.php*, and *about.php* pages to include the form action="logout.php".

```
<form action="logout.php" method="post" name="Logout_Form"
class="form-signin">
    <button name="Submit" value="Logout" class="button"
type="submit">Log out</button>
</form>
```

Now the logout button will call the logout.php script which destroys all sessions and redirects the user to the login.php page. Because the sessions have been destroyed, the user will no longer have access to the *index.php, contacts.php*, and *about.php* pages unless they log back in.

In the lecture we discussed how `session_destroy()` does not unset any of the global variables associated with the session, or unset the session cookie. See more here.

To remove all data associated with the session we can use the `killSession()` and `forgetSession()` functions from the lecture. I have decided to do this with OOPHP to exemplify how you might use Classes and Functions in PHP. First you will need to create a directory called *src* where we will save any classes that we create for our project. We then create a new class file called *session.php*, in which we will create a class called `session`. Paste the below code into session.php

```php
class session
{
    public function killSession()
    {
        //overwrite the current session array with an empty array.
        $_SESSION = [];

        //overwrite the session cookie with empty data too.
        if (ini_get('session.use_cookies')) {
            $params = session_get_cookie_params();
            setcookie(session_name(),
                '', time() - 42000,
                $params['path'], $params['domain'],
                $params['secure'], $params['httponly']
            );
        }
        session_destroy();
    }

    public function forgetSession()
    {
        $this->killSession();
        header("location:login.php");   /* Redirect to login page */
        exit;

    }
}
?>
```

Now that we have a class to use, lets go ahead and use it. Edit the code in *logout.php* as follows:

<?php
  ~~session_start(); /* First we must start the session */~~
~~ session_destroy(); /* Destroy started session */~~

~~ header("location:login.php");   /* Redirect to login page */~~
~~ exit;~~

```php
require_once '../src/session.php';
$session = new session();
$session->forgetSession();
?>
```

It can seem a little strange having to 'require' a file and create an instance of a class as well. Actually, we seem to use require/include a lot. This can be made a lot less onerous by using namespaces and a dependency manager such as Composer. We don't have time to learn about those things this semester, but SymfonyCast.com have a nice short tutorial on name spaces here and Composer here.

Part 6 – What to do next?

You will need to take this a little further for your project so it is a good idea to complete the following:

- Use client and server side form validation to cleanse the username and password input.
- Create some pages that the user can see without logging in (is it the session or the redirect that stop the user accessing the existing pages?).
- Create a registration form – simple form that saves new input data so that, when the user decides to login next time, we can test the login form data against the registration details (as opposed to our hardcoded username and password).
- After you get this to work with simple variables, why not try to save the registration data into a DB. Note, if you have several users registered in your DB table, you will need to search the DB for the specific username and password entered. Take a look back to previous lab work to see how this might be done.