

## Lab 10 – Managing State and SQL Injection

### Aims:

- To be able to manage state information by passing information from one PHP page to another, using **sessions**.
- To become aware of how to provide redirection from one PHP script to another in PHP.
- To understand how SQL injection works and how to avoid it.

### Getting Started:

Create a new folder '**lab10**' under the unit folder on the mercury server in your */htdocs* folder. Save today's work in this folder.

All delivered web pages should conform to HTML5 and must be validated.

You could also create and link an external stylesheet, to the pages, and this should be valid CSS3.

### Task 1: Up and down counter using PHP sessions

The overall task is to create a simple web application that displays an integer and contains three links to update the integer. One link increments the integer by 1, another link decrements the integer by 1, and the last link sets it to 0.

#### Step 1:

Create a file **number.php** that starts up a session, creates a session variable if it does not exist and displays it on the web page.

```
<?php
    session_start(); // start the session
    if (!isset($_SESSION["number"])) { // check if session variable exists
        $_SESSION["number"] = 0; // create and set the session variable
    }
    $num = $_SESSION["number"]; // copy the value to a variable
?>

<!DOCTYPE html>
<html lang="en" >
    <head>
        <meta charset="utf-8" />
        <meta name="description" content="Playing with PHP Sessions" />
        <title>PHP Sessions Lab</title>
    </head>
    <body>
        <h1>Creating Web Applications - PHP Sessions Lab</h1>
        <?php
            echo "<p>The number is", $num , "</p>"; // displays the number
        ?>
        <p><a href="numberup.php">Up &#x2191;</a></p><!-- links to updating pages -->
        <p><a href="numberdown.php">Down &#x2193;</a></p>
        <p><a href="numberreset.php">Reset</a></p>
    </body>
</html>
```


Test in the browser, and check that the **number.php** page, as delivered to the browser, is valid HTML5.

**Step 2:**

Create a file **numberup.php** that increments the session variable by 1.

**This page does not contain any HTML** and redirects to **number.php** after update.


```
<?php
(1)                                     // start the session
(2)                                     // copy the value to a variable
$num++;                               // increment the value
$_SESSION["number"] = $num;           // update the session variable
header("location:number.php");        // redirect to number.php
?>
```

**Step 3:**

Create a file **numberdown.php** that decrements the session variable by 1.

**This page does not contain any HTML** and redirects to **number.php** after update.

```
<?php
(3)                                     // start the session
(4)                                     // copy the value to a variable
(5)                                     // decrement the value
(6)                                     // update the session variable
(7)                                     // redirect to number.php
?>
```

**Step 4:**

Create a file **numberreset.php** that clears out all session variables and redirects to **number.php** after reset.

**This page does not contain any HTML** and redirects to **number.php** after update.

```
<?php
session_start();                       // start the session
(8)                                    // unset all session variables
(9)                                    // destroy all session data
(10)                                   // redirect to number.php
?>
```



Test in the browser, and check that the page is valid HTML5.

**Note:**

Using `header("location:URL")` function, provides redirection. It is needed as a php script on a server cannot **directly** pass control to another php script on the server. So it does it **indirectly** by passing a http header message back to the client, then the client makes a request to the URL of the php script on the server.

## Task 2: Creating a simple “Guessing Game”

The overall task is to create a simple web application that generates and uses **sessions** to store a random number between 0 and 100.

### Step 1:

Create a file **guessinggame.php** that will be the main page for the game.

In this page, a session variable is randomly set with a value between 1 and 100, if it does not already exist. A user can input a guess, and when the value is returned back to this same page, the page displays whether the number input is higher or lower than the generated number; congratulates them if they correctly guess the number; and displays the number of times the user has guessed. (Always checking that the input data is “in-range” and is numeric).

The page also has:

a 'Give Up' link to `giveup.php`, and

a 'Start Over' link to `startover.php`.

### Guessing Game

Enter a number between 1 and 100,  
then press the Guess button.

**Congratulations! You guessed the hidden number.**

Number of guesses: 5.

[Give Up](#)

[Start Over](#)

### Hint:

The PHP's `rand()` function can be used to generate a random integer.

The `rand()` function accepts two arguments: the first argument specifies the minimum integer to generate; and the second argument specifies the maximum integer to generate.

For example, the statement

`$randNum = rand(10, 20)` generates a random integer between 10 and 20 and assigns the number to the `$randNum` variable.

### Step 2:

Create a file **giveup.php** that displays the random number generated for the current game. The value of the random number is accessed from the session variable.

### Guessing Game

The hidden number was: 40

[Start Over](#)

### Step 3:

Create a file **startover.php** that has no html, it simply destroys the session then **redirects** the user back to `guessinggame.php`

### Step 4:

Test in the browser, and check that the pages `guessinggame.php` and `giveup.php` are valid HTML5.

### Step 5:

Combine `guessinggame.php` and `giveup.php` into one HTML page.

*(At present you can ‘cheat’ by ‘giving up’, then going **back** in your browser and then ‘guessing’ the answer!)*

Hint: Change the ‘Give Up’ link in `guessinggame.php` to a self call with a ‘flag’ in the Query String. e.g. `guessinggame.php?giveup=yes` Then place an ‘if’ control around the form code, so that the form is not displayed, and add an ‘else’ that contains the code from ‘giveup’ to show the hidden number.

## Task 3: VIP member Registration System

The overall task is to create a Web application that manages a VIP Member's details using a MySQL database table created and accessed through PHP scripts.

The following fields are to be created for the '**vipmembers**' table:

- **member\_id** INT AUTO\_INCREMENT PRIMARY KEY
- **fname** VARCHAR(40)
- **lname** VARCHAR(40)
- **gender** VARCHAR(1)
- **email** VARCHAR(40)
- **phone** VARCHAR(20)

### Step 1: Create the VIP Member Home Page vip\_member.html

Create a simple "Home Page" document **vip\_member.html** that contains links to:

- "Add New Member Form", **member\_add\_form.html**,
- "Display All Members Page" **member\_display.php**, and
- "Search Member Page" **member\_search.php**.

### Step 2: Create the add member form member\_add\_form.html

Create a document **member\_add\_form.html** that contains a form with method **post** and action to **member\_add.php**. The form will need inputs for all the data for a new member, except **member\_id**, the "Membership Number".

*Hint: You could adapt any of the previous forms you have created in labs for this purpose.*

### Step 3: Create the database table and a form to insert records member\_add.php

Create the document **member\_add.php** to create a connection to the mysql server and your database, create the '**vipmembers**' table if it does not exist, and insert the details from "Add New Member Form" into the table.

*Hint: You will need to execute two queries in this script: the first to create the table if does not exists and the second to insert the data.*

### Step 4: Retrieve and display records from the database table

#### **member\_display.php**

Create a document "Display all Members Page" **member\_display.php** that selects only **member\_id**, **fname**, **lname** from the database table and displays them neatly in a html table.

### Step 5: Create a Search Page member\_search.php

Create a document "Search Member Page" **member\_search.php** with a form to enable a search for a member(s) based on the last name in the database table, and then display the **member\_id**, **fname**, **lname**, **email** details of all members that matched the last name in a html table.

*Hint: This page will have a form with an action that calls itself ( a self call) as demonstrated in Lecture 9.*

## Step 6: Test and Improve Security

Save `member_search.php` as `member_login.php` and change it to create a “Member Log-in” page with a form to enable a user to input a member’s id (`member_id`) and lastname (`lname`), that will connect to the database and then display the `member_id`, `fname`, `lname`, `email` details of all members *that matched the selection*, in a html table. Again this page will have a form action that calls itself ( a self call)

**Example A:** Initially use a query string something like that shown below. Note that here we do not have quotes around the variables '`$memId`' or '`$lname`'

```
$query = "select member_id, fname, lname, email from $sql_table "
        ."where member_id = $memId and lname= $lname order by lname,
        fname";
```

Test the page with known data, to make sure it is working. Then consider the security issues and solutions discussed in Lecture 10 and Lecture 11, let us try to insert SQL into the inputs in the log-in page.

**Example B:** Firstly try some SQL injection testing on phpMyAdmin:

```
select * from vipmembers where member_id =1 or 1=1 -- and lname=anyname;
```

**Example C:** Then try using SQL injection through your `member_login.php` form:

eg. Input a member id of **1 or 1=1 --** and any value for the lastname.

This injected SQL breaks the script and now shows all the values in the table.

Even worse, if our error message shows the “something wrong with the query string ...” then we have exposed our database variables to the world! A hacker could then try something like:

eg. Member id of **1 OR member\_id LIKE '%'** -- and any value for the lastname.

**Example D:** How can we fix this?

For a start, change the query string to include quotes:

```
$query = "select member_id, fname, lname, email from $sql_table "
        ."where member_id = '$memId' and lname= '$lname' order by lname,
        fname";
```

Test the form again:

eg. Member id of **1 or 1=1 --** and any value for the lastname. What happens?

Add some regular expression checking to ensure that only expected values are permitted.

**[IMPORTANT]** Send your tutor the link to your web page running on the Mercury server to be marked off.