CSE 3101: Internet Computing II

Lab Session 2

Paul Crawford

Semester 1, Week 6 (1st & 2nd October, 2018)

1. Aims

1.1. Further understanding of client-side Web Forms and the server-side PHP language.

1.2. Further increased facility with the ongoing sets of concepts & techniques needed for eventual completion of Assignment 1 (due 12th October).

2. Tasks

2.1. Web Forms {Client-Side; HTML; HTTP Request ('GET', 'POST')}

2.1.1. Practice creating a simple Startup web form, to be *sent back from the server-side*. For instance, assuming a default XAMPP installation, and the presence of a sub-directory 'C:\xampp\htdocs**mywebapp**', you could create a default startup processing-page therein named '**index.php**', with the following content {choose either one of the two variants 'A' or 'B'}:

A). "Static" (Pure HTML)

```
<!DOCTYPE html>
<html>
     <head>
     </head>
     <body>
           <form action="testhandler.php" method="post">
                 First Name: <input type="text" name="fname"><br>
                 Last Name : <input type="text" name="lname"><br>
                 <input type=submit value="Send">
           </form>
     </body>
</html>
```

B). "Dynamic" {Note the concatenation operator ('.') at the end of several lines}

```
<?php
$testForm =
     '<html>' .
     '<head>' .
     '</head>' .
     '<body>' .
     '<form action="testhandler.php" method="post">' .
     'First Name: <input type="text" name="fname"><br>' .
     'Last Name : <input type="text" name="lname"><br>' .
     '<input type=submit value="Send">' .
     '</form>' .
```

```
    '</body>' .
    '</html>';

echo $testForm;
?>
```

Next, you would create another processing-page therein named '**testhandler.php**', to process the data submitted via the above form, as follows:

```
<?php
$fname = $_POST['fname'];
$lname = $_POST['lname'];

echo 'Hello, ' . $fname . ' ' . $lname . "!";
?>
```

Then, in a web browser, navigate to the URL '**localhost/mywebapp/**' {note that the '**index.php**' (or 'index.html') filename is usually default, although it could also be explicitly typed}, and test the above form-submission & processing functionality.

2.1.2. Practice creating a Login web form in a manner similar to the preceding item, with fields for a user-name and password. Upon form-submission, the server side could also send a cookie (denoting that logged-in user); for details, see the § 'Processing Pages / Handlers' below. {Note: In modern web browsers, typically any set cookies would automatically be sent back along with form data re-submitted to the server side (internally via 'Cookie: …=…; …' in the request header).}

2.1.3. Continue to practice creating a Contacts web form in a manner similar to the preceding item, to allow data-entry of record-information per person as shown in Lab Session 1, and submitting that data to the server side (via the '`<form>`' tag, with '`action=…`' & '`method="post"` attributes, and contained '`<input>`' tags, etc.)

2.1.4. Continue to practice creating a Query web form in a manner similar to the preceding item, to allow requesting & receiving lists of those records from the server side (via the usual '`<form>`' tag, etc.) and then displaying the results on the client side, depending on various search criteria — e.g.: *all* records; those where a phone no. begins with "222"; those where the last name starts with "Abc"; etc.

2.2. Processing Pages / Handlers {Server-Side; PHP; HTTP Response}

2.2.1. Continue to practice creating processing-pages, to respectively receive data from the above web forms upon submission.

2.2.1.1. For login-processing, the submitted user-name should be examined to verify that a matching user does exist in in the database. If so, then the submitted password should be checked against that user's (decrypted) one in the database to again verify a match. A failure to match at any point

For cookie-setting, the suitable cookie(s) should be returned to the client side

(internally via ‘Set-Cookie: …=…’ in the HTTP response header). For cookie-getting, the data submitted by the client should be parsed to extract suitable cookie values such as user-name. Assuming a default XAMPP installation, the PHP function ‘`setcookie(…)`’ can be used to tell the client side to store cookies, and the PHP super-global ‘`$_COOKIE`’ array can be used to get any cookies {the function ‘`isset(…)`’ can first be used to check whether a given cookie has been set}.

2.2.1.2. For record-creation, the data should be used to store a new Person record in the database. Also, you could verify that a user-name (or ID) cookie already exists, and use its value to associate that Person data with a specific user. Assuming a default XAMPP installation, either of the two standard PHP interfaces for SQL commands — the older ‘MySQLi’ or the newer ‘PDO’ — can be used to open a connection to the database, issue an appropriate ‘`INSERT INTO …`’ command, and then close the connection.

2.2.1.3. For record-querying, the data should be used to search the database for any matching records, accessible by the current user, to be returned to the client side, in a manner similar to the preceding item.

2.3. Overall Considerations

2.3.1. Do continue striving to logically separate functionality as much as possible into multiple files, and *include* them as necessary. Assuming a default XAMPP installation, the PHP ‘`require`’ and ‘`include`’ statements can be used to pull in code from other files.

2.3.2. For additional practice and code samples, you could also explore any of various useful tutorials available online. For instance, there are several PHP and SQL tutorials available at the **W3Schools** site (<w3schools.com>); pay special attention to the **sections on security and validation, including validated form-processing**.