CSE 3101: Internet Computing II

Lab Session 3

Paul Crawford

Semester 1, Week 7 (8th & 9th 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')}, Sent To Web Browser Via Server-Side Processing-Pages

2.1.1. The remainder of this section assumes a default XAMPP installation (where the server-side Web Root repository is located in 'C:\xampp\htdocs'), and the presence of a sub-directory 'C:\xampp\htdocs**mywebapp**'.

2.1.2. Continue to practice creating a startup Login web form, to be sent back from the server-side, 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 §§ '2.2 Processing Pages / Handlers' below. {Note also that, 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).} {Note: The example below does *not* include client-side form validation; for an illustration, please refer to the **W3Schools** site's PHP tutorial, § 'PHP Forms'.}
For instance, you could create a default startup Login processing-page in that above-mentioned sub-directory 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>
            <h1>Welcome to the Address-Book Manager!</h1>
            <h2>Please login, to start:</h2>
            <form action="loginhandler.php" method="post">
                  User Name: <input type="text"
                                    name="UserName"><br>
                  Password : <input type="password"
```

```
                                        name="Password"><br>
                    <input type=submit value="Log in">
              </form>
      </body>
</html>
```

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

```
<?php
$loginForm =
      '<html>' .
      '<head>' .
      '</head>' .
      '<body>' .
      '<h1>Welcome to the Address-Book Manager!</h1><br>' .
      '<h2>Please login, to start:</h2><br>' .
      '<form action="loginhandler.php" method="post">' .
      'User Name: <input type="text" name="UserName"><br>' .
      'Password : <input type="password" name="Password"><br>' .
      '<input type=submit value="Log in">' .
      '</form>' .
      '</body>' .
      '</html>';

echo $loginForm;
?>
```

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.) Buttons could be added to allow choosing a 'Create', 'Update' or 'Delete' task.

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. Assuming a default XAMPP installation, either of the two standard PHP interfaces for database SQL commands — either 'MySQLi' or 'PDO' — could be used. The remainder of this section assumes the existence of a database-processing PHP file '**databasehelpers.php**', with the followng content {choose either one of the two variants 'A' or 'B'}:

A). MySQLi {Older; MySQL only}

```
<?php
function db_open( $server, $db ) {  // => mysqli instance
      $username = "my_username";
      $password = "my_password";

      // Create connection
      $conn = new mysqli( $server, $username, $password, $db );
```

```
      // Check connection
      if ( $conn->connect_error ) {
            die( "Connection failed: " . $conn->connect_error );
      }

      // For debugging only
      //echo "Connected successfully";

      return $conn;
}

// Close an open database
function db_close( mysqli $conn ) {
      $conn->close();
}

// Process the specified credentials
function db_process_login( mysqli $conn, array $account ) {
      $accountTable = 'my_account_table';

      // Build our 'Retrieve' SQL command
      $sql = "SELECT * FROM $accountTable " .
             "WHERE username = '" . $account['UserName'] . "'";

      // Check the result(s)
      $result = $conn->query( $sql );
      if ( $result->num_rows == 1 ) {
            $account_db = $result->fetch_assoc();
            $password_crypted = my_hash( $account['Password'] );
            $password_crypted_db = $account_db['Password'];
            if ( $password_crypted === $password_crypted_db ) {
                  // Password is OK; check if to "remember" User
                  if ( <not the special Admin User> ) {
                        // "Remember" it somehow, e.g., via cookies
                        $userID = $account_db['UserID'];
                        setcookie( 'UserID', $userID );
                  }
             } else {
                  die ( "Login failed: Incorrect password!" );
            }
      } else {
            die ( "Login failed: Unknown or duplicate User!" );
      }

      // NOTE: If it's the special Admin user, need to also send
      // back a web form with a menu of available regular Users;
      // upon submission, we'd re-process that 2nd regular login
      // Otherwise, send back (e.g.) the Contacts web form.
}

// Create the corresponding Person record
function db_create_person( mysqli $conn, array $person ) {
      $personTable = 'my_person_table';
      // … also retrieve the User ID somehow, e.g., via cookies
      $userIDKey = 'UserID';
      if ( ! isset( $_COOKIE[$userIDKey] ) ) {
            die ( "A User-ID cookie is not set!" );
      }
      $userID = $_COOKIE[$userIDKey];
```

```
      // Build our 'Create' SQL command
      $sql = "INSERT INTO $personTable " .
            '( userid, firstname, lastname, … ) ' .
            'VALUES ' .
            "( $userID, {$person['fname']}, {$person['lname']}, … )";

      if ( $conn->query( $sql ) === TRUE ) {
            echo "Person record created successfully!"
      } else {
            echo "Error: " . $sql;
      }
}

// [Other helpers here for the 'Update' and 'Delete' tasks …]

// Search for any matching Person(s) having the specified field(s)
function db_search_for_person( mysqli $conn, array $query ) {
      $personTable = 'my_person_table';

      // Build our 'Query' command, depending on non-empty fields
      $sql = "SELECT * FROM $personTable " .
             "WHERE ( ( some_column_db = '" .
                  $query['some_field_key'] . "' ) AND ( … ) )";

      // Obtain and send back the result(s)
      $result = $conn->query( $sql );
      // Convert result(s) into HTML table, and echo that. :-)
}
?>
```

B). PDO {Newer; Supports several DB types}

```
<?php
// Open the specified database
function db_open( $server, $db ) {  // => PDO instance
      $username = "my_username";
      $password = "my_password";

      try {
            $conn = new PDO( "mysql:host=$server;dbname=$db",
                             $username, $password );
            // set the PDO error mode to exception
            $conn->setAttribute( PDO::ATTR_ERRMODE,
                                 PDO::ERRMODE_EXCEPTION );

            // For debugging only
            //echo "Connected successfully";
      }
      catch( PDOException $e ) {
            echo "Connection failed: " . $e->getMessage();
      }

      return $conn;
}

// Close an open database
function db_close( PDO $conn ) {
      $conn = null;
}

// Process the specified credentials
function db_process_login( PDO $conn, array $account ) {
```

```
      // Code here is similar to that for MySQLi above, except that
      // it should be wrapped within a 'try' / 'catch' block
}

// Create the corresponding Person record
function db_create_person( PDO $conn, array $person ) {
      // Code here is similar to that for MySQLi above, except that
      // it should be wrapped within a 'try' / 'catch' block
}

// [Other helpers …]

// Search for any matching Person(s) having the specified field(s)
function db_search_for_person( PDO $conn, array $query ) {
      // Code here is similar to that for MySQLi above, except that
      // it should be wrapped within a 'try' / 'catch' block
}
?>
```

2.2.2. Continue to practice creating processing-pages, to respectively receive data from the above web forms (in § '2.1 Web Forms') upon submission.

2.2.2.1. 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 at any point should be fatal.

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-ID. 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}. For example, our '**loginhandler.php**' file could have the following content:

```
<?php
// Pull-in our DB helpers
require 'databasehelpers.php';

// Copy-by-value all of the submitted credentials
$accountData = $_POST;
validate_acct( $accountData ); // … and, validate it somehow

// Process these credentials
$conn = db_open( "localhost", "my_db_name" );
db_process_login( $conn, $accountData );
db_close( $conn );
?>
```

2.2.2.2. For record-creation, -updating, and -deletion, the data should be used to respectively store, modify, or remove a 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. You would open a connection to the database, issue the appropriate database command, and then close the connection. For example, a handler for a 'Create' task could be as follows:

```
<?php
// Pull-in our DB helpers
require 'databasehelpers.php';

// Copy-by-value all of the submitted Person data
$personData = $_POST;
validate_person( $personData ); // … and, validate it somehow

// Create the corresponding Person record
$conn = db_open( "localhost", "my_db_name" );
db_create_person( $conn, $personData );
db_close( $conn );
?>
```

2.2.2.3. 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.

```
<?php
// Pull-in our DB helpers
require 'databasehelpers.php';

// Copy-by-value all of the submitted query fields
$queryData = $_POST;  // Copy-by-value all keyed elements
validate_person( $queryData ); // … and, validate it somehow

// Search for any corresponding Person record(s)
$conn = db_open( "localhost", "my_db_name" );
db_search_for_person( $conn, $queryData );
db_close( $conn );
?>
```

2.3. Overall Considerations

2.3.1. After completing all code entry, then in any 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.

2.3.2. Do continue striving to logically separate functionality as much as possible into multiple files, and *require* / *include* them as necessary. Recall that the PHP '`require`' and '`include`' statements can be used to pull in code from other files.

2.3.3. 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**.