COSC 360
Lab 10 – State with PHP

Information:

Submit all your files file through GitHub classroom and the paste your repository link in Canvas. This lab will require the use of your web stack.

You can find further practice and info for PHP at https://www.w3schools.com/php/.

Instructions:

In this lab, use your files from lab 9 as starter files. Using the invitation link for lab 10, clone the repository for Lab 10 to your local machine. Make sure that you place your completed files back in the repo folder when committing and pushing your final lab results. To ease development, you may want to check out the lab into your htdocs folder so that XAMPP will be able to host and render the files. If your files are not in the htdocs folder, XAMPP will not be able to process the php files. Make note of the folder name as you will use this in your URL when accessing the files at localhost.

Copy your files from lab 9 files into this location. You will need to create the following pages for this lab. Recall that page redirection in PHP can be accomplished with the `header()` function as discussed in lecture, but this must be called before any code if emitted to the body of the page. See https://www.w3schools.com/php/func_http_header.asp and http://php.net/manual/en/function.header.php for details. Use your database from lab 9 for this lab.

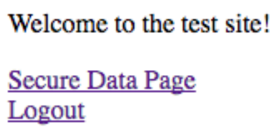
Part 1: Adding State

Create the following pages and add the required functionality to your site:

`login.php`: This is your new login page (Note: this is now different from `login.php` in lab 9). If the user is not currently logged in, display the login form. When the form is submitted, have it `POST` the results to `processlogin.php`. If the user is logged in already and attempts to view this page, redirect then to `home.php`.

`processlogin.php`: This is the file that will check and create a new user in the database. This is based on your `login.php` file from lab 9. When it received valid user credentials (which it will check in the db), it create a new session superglobal for `username` and redirect them to `home.php`. If bad data is entered (ie `GET` or incomplete fields), redirect back to `login.php`. If the user is already logged in when calling `processlogin.php`, redirect them to `home.php`.

`home.php`: This page will display a welcome message and links to `secure.php` and `logout.php` as shown below:


Welcome to the test site!

Secure Data Page
Logout

if the user is logged in. If the user is not logged in (ie. `$_SESSION` superglobal is not set), redirect back to `login.php`.

`secure.php`: This page is only visible if the user is logged in and will display some placeholder text and include a link to `logout.php`. It the user is not logged in, have the page display a message that this content is only available to users and provide a link to `login.php`.

`logout.php`: This page is responsible for clearing the `$_SESSION` superglobal (logging the user out). Once the logout is complete, redirect the user back to the referring page. If a user attempts to view `logout.php` without being logged in, redirect them to `login.php`.

Test your pages to ensure that the state logic functions correctly.

Part 2: Storing Images in your Database

1. Make a copy of the file you created in lab 9 for creating a new user (`lab9-1.html`) and call it `newuser.html`.
2. Modify your form in the new file to add an input field for file types and name the field `userImage`. As the form is now supporting a file update, you will need to set the appropriate encoding type for the form (see https://www.w3schools.com/php/php_file_upload.asp). This form will still submit to `newuser.php`.
3. `newuser.php` will require multiple modifications. Firstly, it must be modified to receive and store uploaded files. Once the file is uploaded successfully, it can be stored in the database. This will be accomplished in two parts.

Part I – File Upload

a. Review the code samples at https://www.w3schools.com/php/php_file_upload.asp and implement the functionality in your `newuser.php` page. Allow only `jpg`, `png` and `gif` files to be uploaded to the server and limit the file size to under 100k. Store the image file type in a variable called `$imageFileType` (this will be important in subsequent steps).
b. In the process of uploading a file, it will need to be moved from a temporary location to the file location on your server (review section 12.4 of the text (9.4 in 1st edition)). You will find a folder called `uploads` relative to your `lab10` folder. This is where your image files will ultimately end up. You may have issues with permissions on the folder as the web server will need access to write to this folder. Although it is not normally advisable, set the read/write permissions to everyone on the `uploads` folder if experiencing issues. Ask your TA if you are encountering issues with permissions.
c. Test your file uploading with the three small images of Zelda available that can be found in the folder `src_images` . Create a new user and upload the file to your server. Check in the `uploads` folder to ensure that all three file formats are able to be uploaded. At this point, you could store the file path to the image in your database for user images which is an efficient method, but you can also store the images directly in your database (if not too large) as done in the following steps.

Part II – Storing BLOBS in the Database

d. A file can be stored in a database as a binary large object. In order to accomplish this, you will need to make some additional changes to your database to support a new table (called userImages) as well as a new attribute for your user table. Run the following sql statement on your database which will create a new attribute called userID that is an auto-incrementing field. This will be used build a relation between the user table and the userImages table. The command is:

```
ALTER TABLE `users` ADD `userID` INT NOT NULL AUTO_INCREMENT , ADD UNIQUE (`userID`);
```

e. Retrieve the userimagesddl.sql from Connect. Import the file to your existing database. This will create a new table for the user images to be stored in, as well as establishing a relationship between the users table and userImages table.
f. The process of inserting a file into a database requires a few steps. The first item you will need to implement is the ability to retrieve the userID (the newly created attribute) for the new user. This can be done by generating a second query after the new users information is entered into the user table using a SELECT statement to access the userID for the username. Implement and test that your code is able to retrieve the correct userID.
g. In order to upload a file into the database, you will need to use a prepared statement. The following code snipped demonstrates how to accomplish this along with comments to help explain the function of each line.

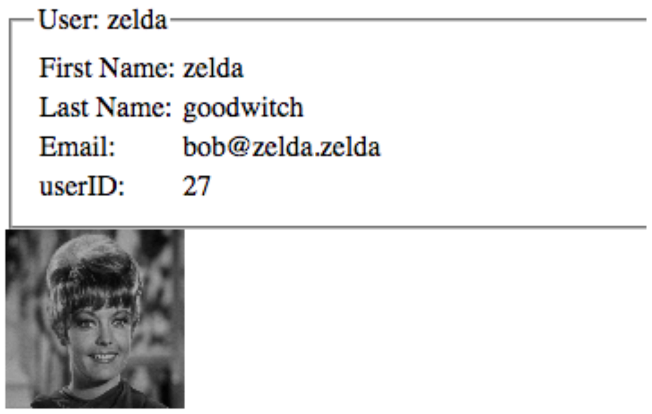
```
$imagedata = file_get_contents($_FILES['userImage']['tmp_name']);
              //store the contents of the files in memory in preparation for upload

$sql = "INSERT INTO userImages (userID, contentType, image) VALUES(?,?,?)";
              // create a new statement to insert the image into the table.  Recall
              // that the ? is a placeholder to variable data.

$stmt = mysqli_stmt_init($connection);      //init prepared statement object

mysqli_stmt_prepare($stmt, $sql);           // register the query

$null = NULL;
mysqli_stmt_bind_param($stmt, "isb", $userID, $imageFileType, $null);
              // bind the variable data into the prepared statement.  You could replace
              // $null with $data here and it also works.  You can review the details
              // of this function on php.net.  The second argument defines the type of
              // data being bound followed by the variable list.  In the case of the
              // blob, you cannot bind it directly so NULL is used as a placeholder.
              // Notice that the parametner $imageFileType (which you created previously)
              // is also stored in the table.  This is important as the file type is
              // needed when the file is retrieved from the database.

mysqli_stmt_send_long_data($stmt, 2, $imagedata);
              // This sends the binary data to the third variable location in the
              // prepared statement (starting from 0).

$result = mysqli_stmt_execute($stmt) or die(mysqli_stmt_error($stmt));
              // run the statement

mysqli_stmt_close($stmt);                   // and dispose of the statement.
```

h. If all has gone well, you should be able to now create a new user and upload the files and have it stored in the database. Using phpmyadmin (or another appropriate tool) check to see ensure data has been entered into the userImage table.

Part II – Retrieving BLOBS from the Database

i. Make a copy of the file you created in lab 9 for finding a user (lab9-4.html) and call in finduser.html. This will be unchanged, but have it send the request to finduser.php.
j. Make a copy of finduser.php from lab 9 and place it in the lab 10 folder.

k. Modify the code so that it also retrieves the `userID` from the user table and display the results as shown below:



User: zelda

First Name: zelda
Last Name: goodwitch
Email: bob@zelda.zelda
userID: 27

l. Once you have the `userID`, you can use this to retrieve the appropriate image from the database. Again, this will be done with a prepared statement using the following code. Please note comments on functionality:

```
$sql = "SELECT contentType, image FROM userImages where userID=?";
               // build the prepared statement SELECTing on the userID for the user
$stmt = mysqli_stmt_init($connection);
               //init prepared statement object
mysqli_stmt_prepare($stmt, $sql);
               // bind the query to the statement
mysqli_stmt_bind_param($stmt, "i", $userID);
               // bind in the variable data (ie userID)
$result = mysqli_stmt_execute($stmt) or die(mysqli_stmt_error($stmt));
               // Run the query.  run spot run!
mysqli_stmt_bind_result($stmt, $type, $image); //bind in results
               // Binds the columns in the resultset to variables
mysqli_stmt_fetch($stmt);
               // Fetches the blob and places it in the variable $image for use as well
               // as the image type (which is stored in $type)
mysqli_stmt_close($stmt);
               // release the statement
```

This will retrieve the blob along with the image type from the database for use in your PHP code.

m. The challenge is that this is no longer a file located on a filepath, but is sitting in memory as a binary object. You can use PHP to emit an image tag and place the binary data inline. Your code will need to be able to tell the web browser what type of data it is so it can be rendered correctly. The following in an example of PHP echoing the image tag so that it can emit the image from the blob:

```
echo '<img src="data:image/'.$type.';base64,'.base64_encode($image).'"/>';
```

Notice that `$type` is now part of the statement which is relate the image type (as stored in the database) to the browser. Binary data cannot be transported raw over the network, so you can use the PHP function `base64_encode()` to ensure that the binary data can survive transport through the transport layers of the network stack. Modify `finduser.php` to display the image of the user on the page as shown in the previous figure.

n. Test the page to ensure proper functionality.

Submission:

Make sure to add your files to your repository before coming. You should have modified the following files:

In part I:

- **login.html**
- **login.php**
- **processlogin.php**
- **home.php**
- **secure.php**
- **logout.php**

In part II:

- **newuser.html**
- **newuser.php**
- **finduser.html**
- **finduser.php**

Commit your HTML and PHP files to your repository and push back to GitHub as well as submit your repository url via canvas.