In_class Assignment 4 -- Forms, Validation, Cookies and Sessions

1. Validate.php - HTML5 provides convenient and user friendly client-side validation features. Client-side validation must usually be double checked on the server. This exercise utilizes HTML5 client-side validation on Validate.php and PHP server-side validation on ValidateConfirm.php. The exercise uses three files:

1. validate.php - contains html input form and client-side validation
2. validateConfirm.php - php server-side validation
3. validationUtilities.php - validation functions used by validateConfirm.php

Steps:

a. Validate.php: You may copy the source code from the sample. The "no validation" button is needed for assignment grading.

b. ValidateConfirm.php: Use the code from the class handout **ValidateConfirm.php.txt** as a template (note: remove the .txt extension on php files). The validation functions require the include file **ValidateConfirm.php.txt**.

c. Test Validate.php to confirm that it works. To validate birthday use the IsValidDate() function in validationUtilities.php.

d. To validate age, add a new function named fIsValidRange in ValidationUtilities.php. The function will have three input parameters: $value, $min, $max. It should use php's is_numeric() function, a greater-than-or-equal test, and a less-than-or-equal test. It will be similar to the fIsValidLength() function.

e. To validate zipcode, add a new function named fIsValidZipcode to ValidationUtilities.php. It should use php's is_numeric() and strlen() functions (length should equal 5).

f. Modify ValidationConfirm.php to utilize the new functions.

2. Order01.php - This three-page order form validates user inputs and uses cookies to store user information between pages. String length validation is used for all three parameters, which must be between 2 and 20 characters in length. Steps:

a. Order01.php contains only client side code. You may copy the html from the sample page and it should work without modification.

b. Order02.php uses both client and server validation to validate the two inputs (fname & model) from checkout01. If the inputs are not valid a message is displayed and the exit() statement is used to terminate execution (so that the form is not displayed). If the inputs are valid they are written to cookies and the form is displayed.

c. The final page, Order03.php, validates the color value from order02.php reads fname and model from the cookies, and displays the information in a message to the user. It also displays an image of the car they have selected.

d. There are nine car-color combinations (3 car models x 3 colors). You should use the images located at: **corvetteBlue.jpg** .

The image names are:

corvetteBlue.jpg
corvetteRed.jpg

corvetteYellow.jpg
mustangBlue.jpg
mustangRed.jpg
mustangYellow.jpg
subaruBlue.jpg
subaruRed.jpg
subaruYellow.jpg

e. The image names are composed of the car model and color. Concatenate the car model and color directly into the image tag to specify the image, as shown below. The curly braces allow interpolation without the need for spaces before or after the variables.

```
echo "<img src='../images/{$model}{$color}.jpg' />";
```

f. Provide a link back to order01.php.

Option: You may use different images. You must have at least three products each with at least three options.

g. The order process assumes that users start at the beginning (order01.php in this case). However users may also land in the middle of the order process via a search engine or a direct link, in which case the back button will not direct them back to the correct page. The solution is to check the referring page (the page where the user arrived from) and automatically redirect the user to the beginning of the order process if they arrived from elsewhere.

The following code snippet shows how this is done on order03. Add this code to order02 and order03 (near the top, before the validation). You will need to modify it slightly for order02.php. You can see how it works by going directly to **Order03.php**.

```
//must arrive from order02.php
$referrer = $_SERVER['HTTP_REFERER'];
if (stripos($referrer, 'order02.php') == false)
header("location:order01.php");
```

3. protected.php - In this assignment you will create a password protected page named protected.php. The secured page checks for a session object to see if the user is authenticated and redirects them to the login page if they are not.

The login page checks the username and password and, if valid, creates a session containing the user's name. The user is then redirected to protected.php.

Note: all php pages that use sessions must include the **session_start()** function at the top of the page before any output is sent to the browser.

Steps for login.php:

1. Create a form with two textboxes, a hidden field named 'postback' (see source code in example) with value 'true, and a button. Passwords should not be passed in the query string so use method = "post" in the form tag.
2. The page will postback when the button is clicked. Retrieve the values for each of three form elements from step 1 and assign to variables.

3. Repopulate the textboxes on page postback. Echo $fname to the value field of the textbox. The syntax is something like:

```
value="<?php echo $fname ?>"
```

4. Validation: the sample page includes HTML5 validation but you need to add server side code as well. To prevent the server-side validation message from displaying when the page originally loads use a hidden filed named 'postback' and set its value to 'true'. Below each textbox add php code similar to:

```
if ($postback && strlen($username) < 1) {
    echo "Please enter your name.";
}
```

5. Validate inputs and create session: At the top of the page add an IF statement that checks that a username has been entered and that password is 'guest' If both conditions are true, assign the user's name to a session object and redirect the user to protected.php. The syntax to check for a valid username and password looks something like:

```
if ($password == 'guest' && strlen($username) > 0) {
```

The syntax to redirect to protected.php:

```
header("location: protected.php"); exit; //stops page execution
```

6. Hyperlink: Add a hyperlink near the bottom of the page that points to protected.php.

7. Finally, check that the page is using HTTPS (secure socket layers). Passwords should always use HTTPS. The first answer in **Redirecting from HTTP to HTTPS with PHP** shows how to redirect users to HTTPS. Cut-and-paste the first answer to the top of your login page.

Steps for protected.php

1. Check if logged in: Add an if statement at the top protected.php checks for a session and if not found redirects the user to login.php. The syntax is something like:

```
if (!isset($_SESSION["username"]))
```

2. Greeting: In the body of the page add a message that greets the user by name.

3. Logout button: Add a form button with the text "logout" and a hidden form field that sets "abandon" to "true."

4. Session_unset(); Add another if statement at the top of the page that checks if the logout button has been clicked. If the posted variable "abandon' is true then use the session_unset(); function to clear the session, then redirect the user back to the login page.