Autopopulation; Session & Cookies

CGT 356
Web Programming, Development, & Database Integration
Lecture 5
Session array

- Use the Session array to store data that needs to be recalled on later pages
  - \$_SESSION[“foo”]

- Use local variables on data that is ONLY used on that one page that has the variable
  - $foo
Starting the Session

- Session = One user’s continuous visit to the site
- “Global Variables” across the user’s entire visit
- To be able to use the Session array, you must call the session_start method first – at the top of the page

```
session_start();
```
Declaring & Initializing

- \$_SESSION["UserName"] = \$_POST["login"];  
  - This assigns the value from the form input field to the Session array named UserName

- Later, on the same page, or on another page, to recall what was entered in the input field, use:
  - \$_SESSION["UserName"]
  - Example:
    - echo \$_SESSION["UserName"];  
      - Will write the value to the browser window

- "UserName" can be anything you want… it is a variable name
Timeout - PHP

- By default, Session Timeout usually 3 hours, or 180 minutes

- PHP has a global mechanism (if you have access to your own server / installation of PHP) for setting timeout in php.ini
  - `session.cookie_lifetime = 0`
    - If zero (0), then it lasts until the browser is restarted
    - Otherwise the number is in minutes
Timeout - PHP

- `session_cache_expire(30);`
  - By default, this value is 180.
  - The number passed is in minutes
    - This example sets the timeout to 30 minutes

- In Code:
  - `session_cache_expire(15);`
  - `session_start();`
  - ...
Timeout (cont.)

- When does timeout reset?
  - Anytime there is action that requires the server.
    - Click a link, timeout is reset
    - Scroll a page? No
    - Click the back button? …Maybe, depends
    - Basically, any type of navigation that requests another page from the server. That tells the server to reset the timeout period for the current user.
End/Abandon a Session - PHP

- `session_unset();`
  - frees all session variables currently registered

- `session_destroy();`
  - destroys all of the data associated with the current session. It does not unset any of the global variables associated with the session, or unset the session cookie.

- **Example in code:**
  - `session_unset();`
  - `session_destroy();`
Session ID - PHP

- `session_id()`
  - used to get or set the session id for the current session
  
  - `session_id` can only be set before the session starts.
Catching Session Timeout

- First, when a user logs in, store their userID / login into a Session variable:

```php
if (($login == "ron") && ($pass == "php")) {
    $_SESSION["userID"] = $login;
}
```
Catching Session Timeout - PHP

- On all subsequent pages, you can catch a session timeout by checking to make sure a session variable has a value:
  - ...
  - if(empty($SESSION["UserID"]))
  - {
    //don’t display page data
  }
  - ...
- Catches Session(“Login”) in case the session has expired or been abandoned.
empty() - PHP

- empty() is an incredibly useful function.
  - Returns a 1 if true, 0 if false

- Alternate example:
  - if(!empty($_SESSION["UserID"]))
    {
      // display page data
    }

- Notice the !
  - Means “not” – “if session userID is not empty…”
error.php

- Redirecting a user to error.php
  - Whenever you feel a user is trying to hack into your system
  - If a user types in the URL of an admin page instead of navigating to it
  - If a user’s session times out
  - If a user logs out, then tries to go back to their pages
  - Etc.
error.php

- As good practice, you should empty out your own variables anyway:

  Typical contents:

  ```php
  //Clear Session variables
  $_SESSION["Login"] = "";
  $_SESSION["foo"] = "";
  
  //End Session
  session_unset();
  session_destroy();
  
  Then your typical error statements to the user
  ```
logout.php

- The contents of error.php would also be very close to the contents of logout.php – except you do not print an error message

- If a user logs out
  - Want to erase all Session variables
  - Want to unset and destroy the Session
Modification to login.php

- If user has visited the site, their name should be filled into the login form.
- If user has been away for more than $x$ minutes, they have to login again.
- If user has not logged in, they are redirected to the login page.
  - login4.php
Contents

- Want to see all of the contents of the Session array?

```php
session_start();
foreach($_SESSION as $key => $val) {
    echo $key . " : ". $val . "<br />";
}
```
Autopopulation

- The process of putting data into form fields so that the form appears to be filled out when the page loads

- In many cases, this shortens the amount of typing the user has to do – making it more user-friendly
Autopopulation

☐ Short explanation (don’t use this example)

☐ <p>

<label for="userID">Login: </label>
<input type="text" id="userID" name="userID"
value="<?php echo $_SESSION["userID"] ; ??>" />

</p></p>

<label for="comment">Comments: </label>
<textarea id="comment" name="comment">
<?php echo $_SESSION["comment"] ; ?>
</textarea>

</p>

☐ Dynamically write the value into the form element at page load
Autopopulation

In PHP, you need to check session variables first before attempting to use them. Use empty() to determine if a session variable contains anything

```php
if(!empty($_SESSION["userID"]))
{
    echo $_SESSION["userID"];
}
```
Autopopulation

Long Explanation

```php
<p>
<label for="userID">Login: </label>
<input type="text" tabindex="1" id="userID" name="userID"
value="<?php if(!empty($_SESSION["userID"])){ echo $_SESSION["userID"]; } ?>" />
</p>
<p>
<label for="comment">Comments: </label>
<textarea id="comment" name="comment" cols="40" rows="5" tabindex="4">
<?php if(!empty($_SESSION["comment"])){ echo $_SESSION["comment"]; } ?></textarea>
</p>
```
Session vs. Cookie

- What is the difference?
- When do I use one over the other?
Cookies?

- Explain Cookies:
  - Small text chunks left on the user’s machine
  - Restrictions
    - Size at most 4k
    - Only the domain that set it can read it
    - Unless explicitly set, die when user closes browser
    - Explicitly deleted by setting expiration date to “some time in the past”
Session Array

- The session array is essentially a short-term cookie that stores information on the client machine for the duration of a user’s visit to a site.
Cookie

When do I use a cookie?

- When you want to store information on the client for longer than one visit to the site.
Cookie

- What is wrong with that?
  - Users do not like that
    - Do not like having cookies stored on their machines
  - Users may delete your cookie and it will not be there next time
Cookie

- By default, a cookie will expire when a user closes the browser, similar to the Session array.

  - You can set an expiration date so that the cookie stays longer or shorter

    - Just like the Session array.
Cookie Example

$value = "my cookie data";

setcookie("TestCookie", $value);

setcookie("TestCookie", $value, time() + 3600);
/* expire in 1 hour */
Cookie Example

- Setting a key within a cookie

```php
setcookie("Employee[EmpID]", "125775");
setcookie("Employee[FirstName]", "Ron");
setcookie("Employee[LastName]", "Glotzbach");

// after the page reloads, print them out
if (isset($_COOKIE["Employee"])) {
    foreach ($_COOKIE["Employee"] as $name => $value) {
        echo "$name : $value <br />
    }
}

// which would output:
EmpID : 125775
FirstName : Ron
LastName : Glotzbach
```
Cookies

- You get the idea...

- Cookie syntax is similar to any other syntax in PHP when it comes to reading values or comparison in if statements.

```php
echo($_COOKIE["Employee"]);
```