Dynamic Web Development

Lecture 11 – Cookies and Sessions

HTTP is a STATELESS Protocol

Any data exchanged via HTTP is forgotten when:

- the page has been sent to the client, and
- the connection is closed.

HTTP does not require the server to keep a record of the requests that came before the current one.

This means that the server is not able to preserve the state of the user's interaction.

Therefore extra information has to be:

- transmitted directly to the next page
- stored in the browser, or
- stored on the server

Moving data from one page to another

An HTML form can transmit a set of variables to the next page (via the server)

GET The variables will be sent in the URL of the webpage

www.kevin.com?name=kevin&age=45

POST The variables are enclosed in the HTTP header and are not visible to the user.

They can be accessed on the following page by means of the following global variables:

$_GET['name'] or print_r($_GET)
$_POST['age'] or print_r($_POST)

Cookies

Data is stored in the browser client

Can have a long lifespan

There is a limit on the amount of data they can hold

Good if the transaction isn't always being processed by the same webserver

Can be manipulated by Javascript on the client machine. Not secure.

Sessions

Data is stored on the server

Client sends a session ID (usually stored in a cookie) which enables the server to access the correct data

Only lasts until the user closes their browser window.

There is virtually no limit on the amount of data they can hold.

Requires the user to always use the same server for the entire transaction.

More secure – good for things like online shopping baskets

Using Cookies

Cookies are small files containing information which are saved on the client computer.

Some people have them turned off, or delete them, which means that this won't always work.

Used because they can last longer than a session.

Created by using the setcookie( ) function

Referenced by using the $_COOKIE['cookiename'] command.

Contents of a Cookie

setcookie(name, value, time, path, domain, secure);

Name	all cookies must have a name
Value	the value that you want to store
Time	number of seconds before the cookie expires. time()+60*60*24*365 is 1 year
Path	directory in which the cookie is active. Default: / (entire site).
Domain	Restrict cookie to specified subdomains
Secure	Whether a cookie must have a secure https connection to be set. 0=no, 1=yes

Order is Important

You must use the setcookie( ) function in your PHP page before any HTML code is output.

This is because the HTTP protocol sends

- the header information (including cookies)

before it sends

- the body information (the HTML)

When you use setcookie( ) the server adds an extra header line to your HTTP message.

If you use setcookie( ) in your PHP page after you have started sending HTML you will get an error message.

setcookie.php

```
<?php
setcookie('username', 'This is Kevins cookie', time()+120 );
?>


<html>
<body>
<p>See if you browser will let you look at the cookies that
it has stored inside it.
Then click <a href="lookatcookie.php">here</a> to go to the
next page.</p>
</body>
</html>
```

When you request this page to be sent to your browser:

Will cause this packet to be sent from the server to the client

```
HTTP/1.0 200 OK

Content-Length: 1276
Content-Type: text/html
Date: Sat, 21 Nov 2009 09:00:30 GMT
Set-Cookie: username=This is Kevins cookie; expires=Sat,
21 Nov 2009 09:02:30 GMT


<html>
<body>
<p>See if you browser will let you look at the cookies
that it has stored inside it.
Then click <a href="lookatcookie.php">here</a> to go to
the next page.</p>
</body>
</html>
```

The packet will then cause the browser to:

- create a cookie and store it on the users computer.
- display the webpage.

When the user clicks on the link, to load up the page called:

lookatcookie.php

it will cause this packet to be sent to the server:

HTTP GET request packet

```
GET    /lookatcookie.php    HTTP/1.0
Host: www.myserver.com
Cookie: username=This is Kevins cookie
```

Note that if the browser contains several cookies which were set by the same server, they will all be included in the Cookie header, whenever a request packet is sent to that server, whether they are required or not.

```
GET /lookatcookie.php HTTP/1.0
Host: www.myserver.com
Cookie: username=This is Kevins cookie; cookie2=myvalue
```

lookatcookie.php

```
<html>
<body>
These are the contents of your cookie
<?php
// Print contents of cookie called 'username'
echo $_COOKIE["username"];

// Another way of doing it
print_r($_COOKIE);
?>
</body>
</html>
```

The server will execute this PHP code on the server, before the web page is sent to the client

will cause this packet to be sent from the server to the client

```
HTTP/1.0 200 OK

Content-Length: 1200
Content-Type: text/html
Date: Sat, 21 Nov 2009 09:01:44 GMT

<html>
<body>
These are the contents of your cookie

This is Kevins cookie
This is Kevins cookie
</body>
</html>
```

Note that the server has used the information from the GET request header for this page, to fill in the necessary information in the html of this page before it is transmitted to the client.

More on Cookies

You can test to see if a cookie has been set by using the `isset( )` PHP function.

You can store more than one value in a cookie by using an array.

```php
<?php
// set the cookies
setcookie("mydata[age]", "45");
setcookie("mydata[lastname]", "wilson");
setcookie("mydata[firstname]", "kevin");


// after the page reloads, print them out

print_r($_COOKIE);
echo "<br />";

if (isset($_COOKIE['mydata']))
   {
    foreach ($_COOKIE['mydata'] as $name => $value)
      {
        echo "$name : $value <br />";
      }
  }
?>
```

Using Sessions

A session is a combination of a server-side file containing the data that you wish to store, and a client-side cookie which contains a reference id to that data.

Created and accessed by using the `session_start( )` function.

When you use the `session_start( )` function, the server will check to see if the browser has sent the ID cookie.

It will then:

- Use the ID code to load up the correct session data file,

or

- create a new session data file on the server, and send an ID cookie back to the browser for future use.

Order is important again

As before, the reference cookie will be sent as part of the request headers.

The session_start( ) function must appear in the PHP code before any HTML code does – even before any spaces.

The data is stored in the global array $_SESSION. You can create any variables you want.

```
$_SESSION['age'] = $val;
$_SESSION['FirstName'] = "Kevin Wilson";
```

Putting data into a session variable

setsession.php

```
<?php
session_start( );
$_SESSION["id_code"] = "12345";
$_SESSION["name"] = "Kevin Wilson";
$_SESSION["count"] = "0";
?>
<html>
<body>
<p>This page stores some data in the session
variables.
Click <a href="lookatsession.php">here</a> to go to
the next page.</p>
</body>
</html>
```

Getting data out of a session variable

lookatsession.php

```
<?php
session_start();
echo "<html>";
echo "<body>";
print_r($_SESSION);
echo "<br />";
echo $_SESSION["id_code"];
echo "<br />";
echo $_SESSION["name"];
echo "<br />";
echo $_SESSION["count"];
echo "<br />";
$countvar = $_SESSION["count"];
$_SESSION["count"] = $countvar + 1;
?>
<p>
These are the contents of your session variables.
Navigate to another page or two, then come back.
</p>
Try refreshing the page.
</body>
</html>
```

Testing and Removing a Session Variable

You can test a session variable by using `isset();`

```
if ( isset($_SESSION['name'])  )
                                    //then do something
```

You can remove an item of data from the session array by using the function unset.

```
$_SESSION['name'] = "Kevin";

print $_SESSION['name'];

unset( $_SESSION['name']);
```

Ending a Session

A session lasts until the user closes their browser.

Even if they visit other pages and come back, the session still exists until they close their browser.

If you want to explicitly end a users session and delete their data:

```
<?php
session_start( );
$_SESSION = array( );
session_destroy( );
?>
```

Actually, when people say that a session file is deleted when a browser window is closed, this isn't exactly true.

No message is sent to the server, and so the server has no way of knowing when a browser is closed.

What does happen is that the session cookies are deleted when the browser is closed. This happens to any cookie which is not given an explicit expiry date.

This means that the session file cannot be accessed again – but they are still there on the server.

How long should the server wait before it can assume the session has ended, and so be able to delete the session file?

- Wait for too long and you waste resources on the server.
- Wait not long enough, the user has to log in again and create a new session when they return to the website.

There is usually a default timeout of about 20 minutes.

If a session file hasn't been used after about 20 minutes, it will be deleted during the next garbage collection period.

This timeout can be changed by the server administrator, or by PHP code on an individual webpage.

More on Sessions

There are other useful session handling functions built into PHP – see the documentation.

The session handling system is actually quite basic – storing and retrieving data from flat files.

It is quite common to have a cluster of webservers on a large site, with what the user sees as one website being spread between them.

Which one should be used to hold the session data?

The answer is to store session data on a central database server, which can be accessed by all of the webserver machines.

PHP allows you to write your own functions to do the following jobs:

- session open
- session close
- session read
- session write
- session destroy
- session garbage collect

You can then load them into PHP using the function

- **`session_set_save_handler( )`**

and from then on, PHP will use your custom made functions to save sessions in whatever database package you designed them for.