

SWE 363: Web Engineering & Development

Module 7-2

Developing Web Applications with PHP



Objectives

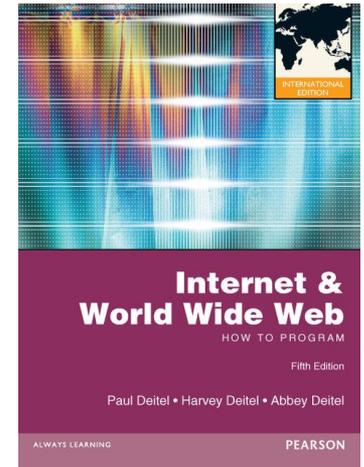
- ❑ Learn how to deal with Superglobal arrays
- ❑ Learn how to manage sessions and cookies

Outline

- Superglobal Arrays
- COOKIEs
- SESSIONs

References

- ❑ “Internet & World Wide Web: How to Program 5th editions”
© Pearson Education
- ❑ “Fundamentals of Web Development” Book by Randy Connolly and Ricardo Hoar, 2015
- ❑ Lots of resources are available at <http://www.php.net>
 - Documentation: manual
 - <http://www.php.net/manual/en/>
 - <http://us2.php.net/manual/en/index.php>
 - Tutorials
 - <http://php.net/manual/en/tutorial.php>
 - Documented PHP functions
 - <http://us2.php.net/quickref.php>
- ❑ W3schools tutorial <http://www.w3schools.com/php/default.asp>
- ❑ Tutorial Public <https://www.tutorialrepublic.com/>



w3schools.com



Superglobal Arrays

Superglobal Variables ?

- ❑ PHP uses special **predefined associative arrays** called **superglobal variables** that allow the programmer to easily access HTTP headers, query string parameters, and other commonly needed information
- ❑ These arrays **always available in all scopes**, ready for the programmer to access or modify them anywhere from any function, class or file (without having to do anything special).

	Variable	Definition
1	GLOBALS	References all variables available in global scope
2	_SERVER	Server and execution environment information
3	_GET	HTTP GET variables
4	_POST	HTTP POST variables
5	_FILES	HTTP File upload variables
6	_COOKIE	HTTP Cookies
7	_SESSION	Session variables
8	_REQUEST	HTTP Request variables
9	_ENV	Environment variables

PHP \$GLOBALS

□ PHP \$GLOBALS

- is used to access global variables **from anywhere in the PHP script** (also from within functions or methods).
- PHP stores all global variables in an array called **\$GLOBALS[index]**.
 - The **index** holds the **name** of the variable.

```
<?php
function myFunc () {
    $foo="Local Scope";
    echo 'This $foo is in the ' . $GLOBALS["foo"] . "\n";
    echo 'This $foo is in the ' . $foo . "\n";
}
$foo="Global Scope";
myFunc ();
?>
```

```
This $foo is in the Global Scope
This $foo is in the Local Scope
```

PHP \$_SERVER

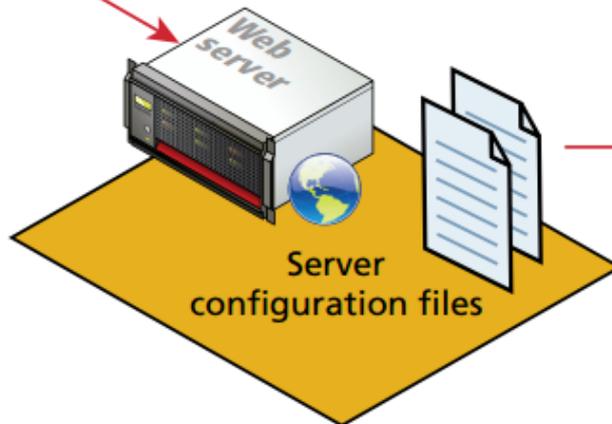
- ❑ PHP `$_SERVER` array contains a variety of information such as **headers**, **paths**, and **script locations**.
- ❑ The entries in this array are **created by the web server**.
 - No every web server will provide all the keys listed by PHP ; servers may omit some, or provide others not listed here.
- ❑ These include:
 - `$_SERVER['PHP_SELF']` ; the filename of the currently executing script
 - `$_SERVER['REQUEST_TIME']`; The timestamp of the start of the request. Available since PHP 5.1.0.
 - `$_SERVER['SERVER_ADDR']`; Returns the IP address of the host server.
 - more

PHP \$_SERVER

`$_SERVER['REQUEST_METHOD']` `$_SERVER['SERVER_PROTOCOL']`



HTTP Request Header



`$_SERVER['SERVER_NAME']`
`$_SERVER['SERVER_ADDR']`
`$_SERVER['SERVER_PORT']`
...

\$_GET and \$_POST superglobal arrays

- ❑ The `$_GET` and `$_POST` arrays are the most important superglobal variables in PHP since they **allow the programmer to access data sent by the client in a query string**.
- ❑ The HTML form allows a client to send data to the server. That data is formatted such that **each value is associated with a name** defined in the form.
- ❑ If the form was submitted using an **HTTP GET** request, then the resulting URL will contain the data in the query string.

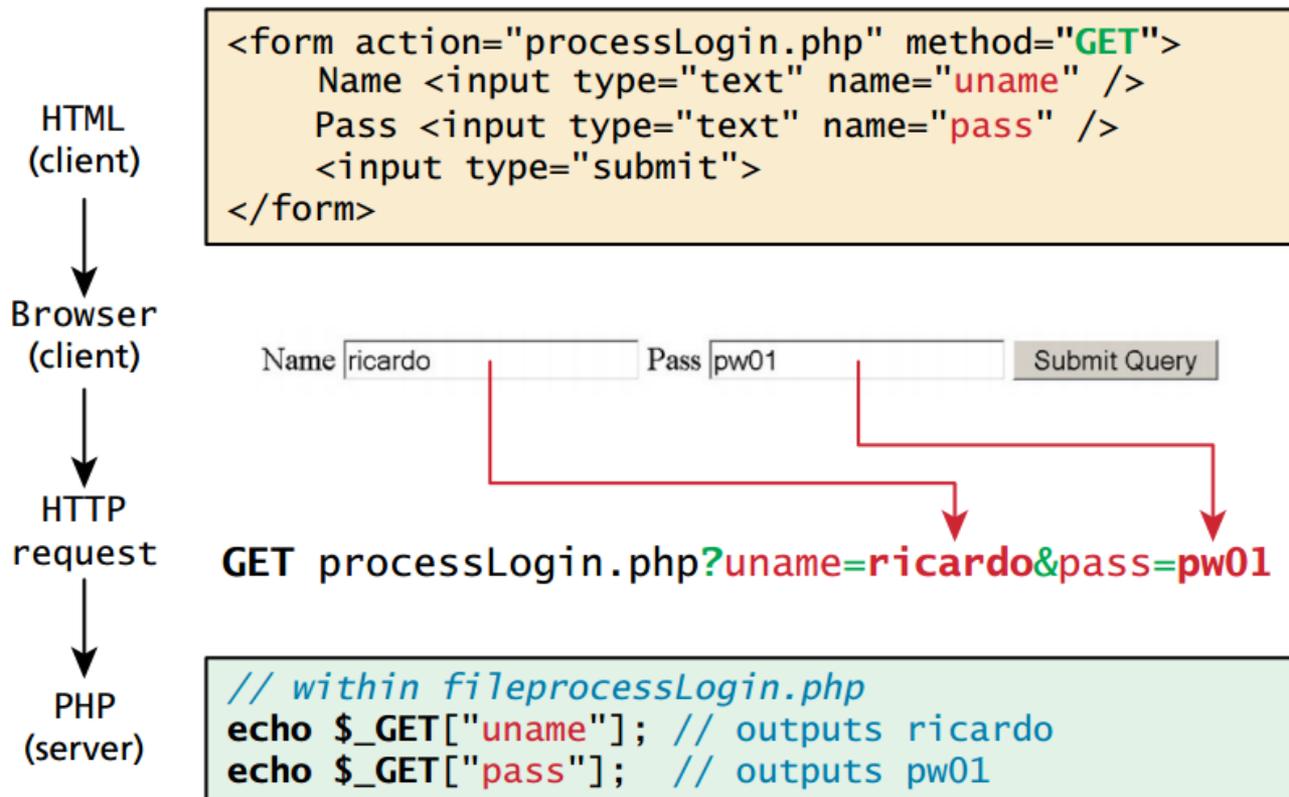
```
http://www.kfupm.edu.sa/ics/courses.php?id=SWE363&term=182
```

- **PHP `$_GET` array** is used to collect form data after submitting an HTML form with `method="get"`.
- ❑ If the form was sent using **HTTP POST**, then the values would not be visible in the URL, but will be sent through HTTP POST request body.
 - **PHP `$_POST` array** is widely used to collect form data after submitting an HTML form with `method="post"`.

From the PHP programmer's perspective, almost nothing changes from a GET data post except that those values and keys are now **stored in the `$_POST` array**.

Using GET method

- ❑ In **GET method** the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands (&).



GET: example

```
<body>
  <form action="action1.php" method="get">
    <p>Full Name: <input type="text" name="fname" /></p>
    <p>Major : <input type="text" name="major" /></p>
    <p><input type="Submit" /></p>
  </form>
</body>
```

Full Name:

Major :

action1.php

```
1 <?php
2 $fname = $_GET['fname'];
3 $major = $_GET['major'];
4
5 echo "Hallo $fname, <br> Your major is $major";
6 ?>
```

Full Name:

Major :

URL

localhost/PHP-SWE363/action1.php?fname=Ahmed+&&major=CS

output

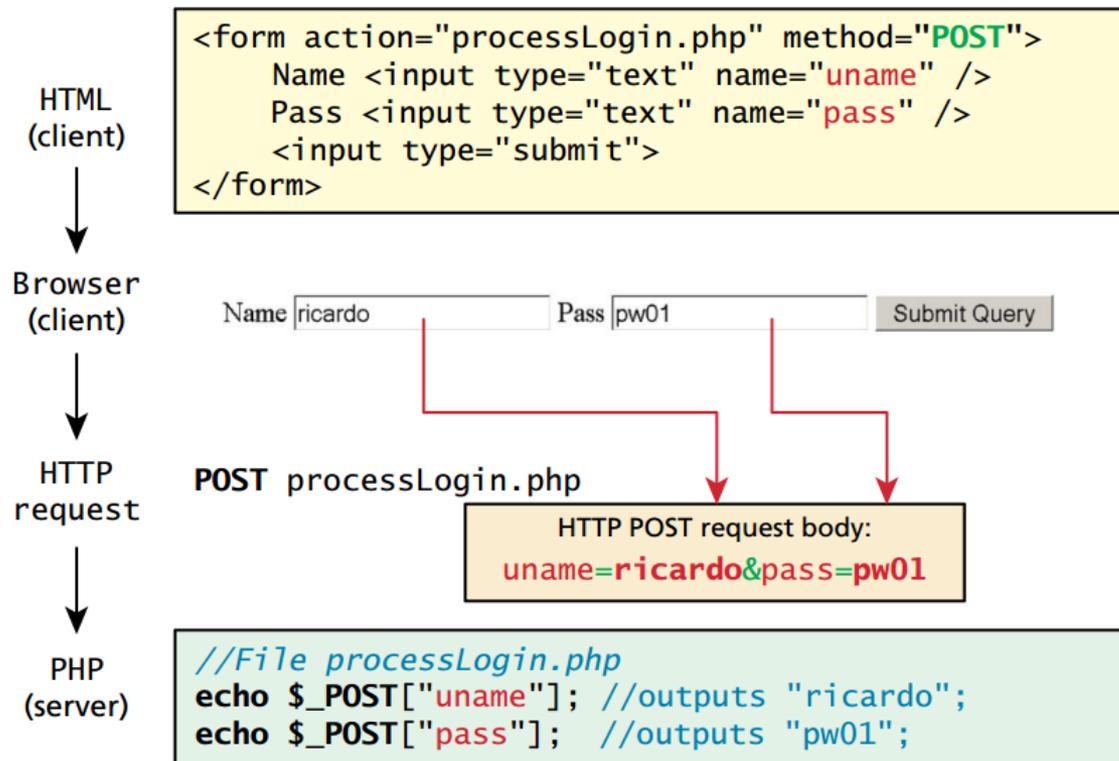
Hallo Ahmed ,
Your major is CS

Source code

Hallo Ahmed ,
 Your major is CS

Using POST method

- ❑ POST Method is **more secure** than GET because user-entered information is never visible in the URL query string or in the server logs.



POST: example

```
<body>
  <form action="action2.php" method="post">
    <p>User Name: <input type="text" name="username" /></p>
    <p>Password: <input type="password" name="password" /></p>
    <p><input type="Submit" /></p>
  </form>
</body>
```

```
1 <?php
2 $name = $_POST['username'];
3 $psw = $_POST['password'];
4
5 if ($name=="Ahmed" and $psw=="abc"){
6   echo "Scuccessful access";
7 }else
8 {
9   echo "Incorrect username or password";
10 }
11 ?>
```

User Name:

Password:



When to use....?

□ GET?

- GET may be used for sending **non-sensitive data**.
- Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL).
- GET also has **limits** on the amount of information to send. The limitation is about **2000 characters**.
- It is possible to **bookmark** the page, because the variables are displayed in the URL. (This can be useful in some cases)
- **Note:** GET should **NEVER** be used for sending passwords or other sensitive information!

□ POST?

- Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request)
- There is a much **larger limit** on the amount of data that can be passed and one can send text data as well as binary data (uploading a file) using POST.
- It is **not possible to bookmark** the page, because the variables are not displayed in the URL

(\$_SERVER['REQUEST_METHOD'])

- ❑ You can determine if you are responding to a **POST** or **GET** by checking the `$_SERVER['REQUEST_METHOD']` variable

```
if($_SERVER['REQUEST_METHOD'] == 'POST')
```

- ❑ Even though you may know that, for instance, a POST request was performed, you may want **to check if any of the fields are set**.
 - To do this you can use the **isset() function** in PHP to see if there is anything set for a particular query string parameter

```
if isset($_POST['myParamName'])
```

- ❑ These two checks may be **true at the same time**, **or they may not be** → you really should check specifically for what information you want to know.

- ❑ `isset` — can be used to determine if a variable is set and is not NULL

```
<form action="action2.php" method="post">
  <p>User Name: <input type="text" name="username" /></p>
  <p>Password: <input type="password" name="password" /></p>
  <p><input type="submit" name="submit" value="login" /></p>
</form>
```

```
<?php
if(isset($_POST['submit'])){
    // code
}
?>
```

Make sure your submit buttons (i.e. `<input type="submit">` etc) have a **"submit" value** assigned to **'name' attribute**.

`<input type="submit" name="submit">` → `isset($_POST["submit"])` returns true.
`<input type="submit" >` → `isset($_POST["submit"])` returns false

From Validation

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["name"])) {
        $nameErr = "Name is required";
    } else { $name = test_input($_POST["name"]); }

    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else { $email = test_input($_POST["email"]); }

    if (empty($_POST["website"])) {
        $website = "";
    } else { $website = test_input($_POST["website"]); }

    if (empty($_POST["comment"])) {
        $comment = "";
    } else { $comment = test_input($_POST["comment"]); }

    if (empty($_POST["gender"])) {
        $genderErr = "Gender is required";
    } else { $gender = test_input($_POST["gender"]); }
}

function test_input($data) {
    $data = trim($data);
    return $data;
}
?>
```

PHP Form Validation Example

* required field.

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male *

From Validation

```
if (empty($_POST["name"])) {
    $nameErr = "Name is required";
} else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/", $name)) {
        $nameErr = "Only letters and white space allowed";
    }
}
```

The [preg_match\(\)](#) function searches a string for pattern, returning true if the pattern exists, and false otherwise.

```
if (empty($_POST["email"])) {
    $emailErr = "Email is required";
} else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $emailErr = "Invalid email format";
    }
}
```

The ID of the filter to apply.
The [Types of filters](#) manual page lists the available filters.

The easiest and safest way to check whether an email address is well-formed is to use PHP's [filter_var\(\)](#) function.

Posting input data to the same page..

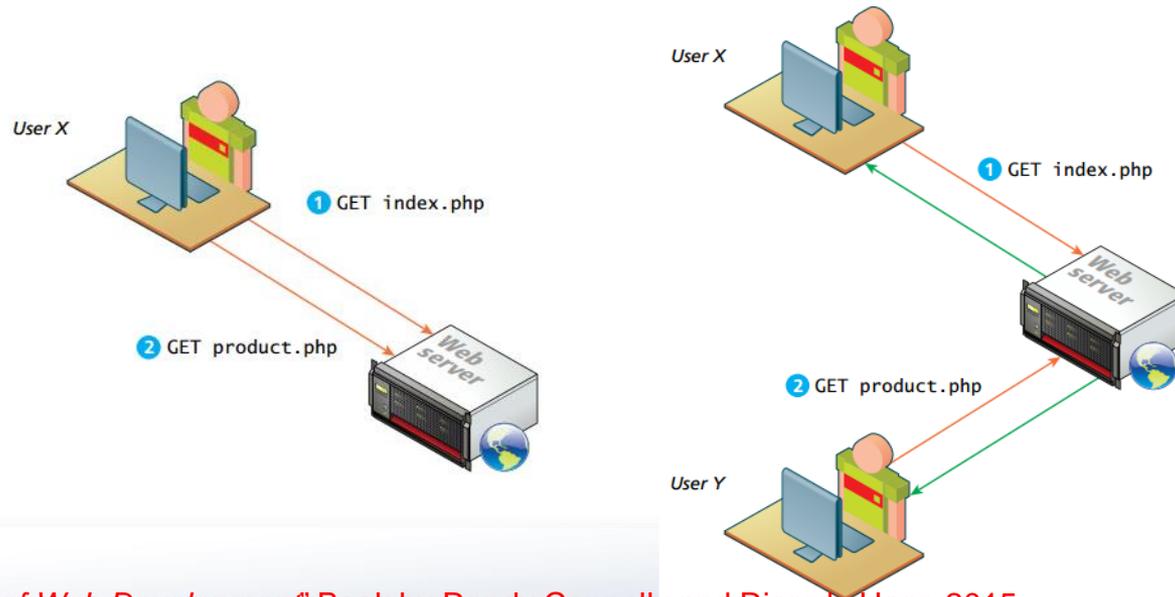
□ In a form on a PHP page, you can use:

- `<form method="post" action="file.php">`
- `<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">`
- `<form method="post" action="">`

Cookies & Sessions

Understanding State Information

- ❑ Information about individual visits to a Web site is called **state information**
- ❑ Maintaining state means to **store persistent information about Web site visits**
- ❑ HTTP was originally designed to be **stateless** – Web browsers store no persistent data about a visit to a Web site
 - The **web server sees only requests**.
 - The HTTP protocol does not (without programming involvement) distinguish two requests by one source from two requests from two different sources



Understanding State Information..

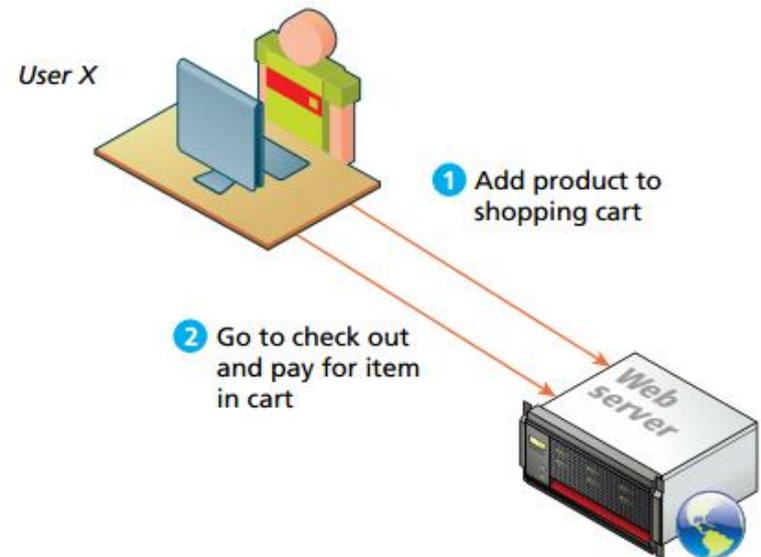
- ❑ Each request for a new web page is processed without any knowledge of previous pages requested or processed.

- ❑ Consider the scenario of a web shopping cart:

>> The user (and the website owner) most certainly wants the server to recognize that

- (1) the request to add an item to the cart
- (2) the subsequent request to check out and pay for the item in the cart

are connected to the same individual !!



Understanding State Information..

- ❑ Most modern applications maintain state, which means that they remember what you were **doing last time** you ran the application, and they **remember all your configuration** settings ..

- ❑ For example:
 - Customize individual Web pages based on user preferences
 - Temporarily store information for a user as a browser navigates within a multipart form
 - Provide shopping carts that store order information
 - Use counters to keep track of how many times a user has visited a site
 - Store user IDs and passwords
 - *A user 'logs in' to a web page. Once logged in, the user can browse the site while maintaining their logged in **state**.*

How this can be done in PHP?

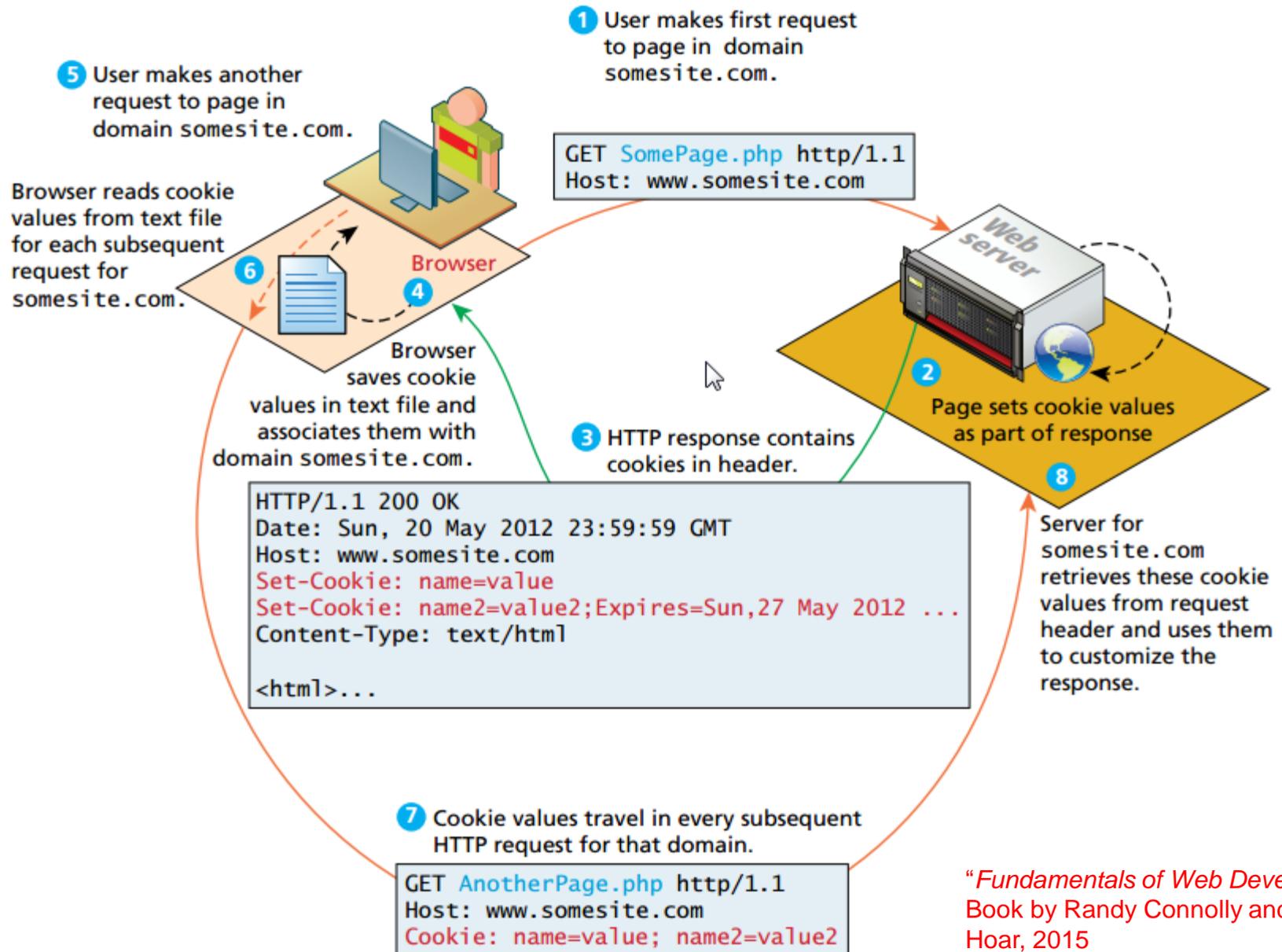
PHP Cookies..

- ❑ **Cookies** are a mechanism for storing data in the remote browser and thus tracking or identifying return users.
- ❑ One typical use of cookies in a website is to “remember” the visitor, so that the server can customize the site for the user.
- ❑ A **cookie** is a small text file that the server embeds on the user's computer and managed by the browser
 - It is often used to identify a user.
 - It is typically used to keeping track of information such as username, preferences that the site can retrieve to personalize the page when user visit the website next time.
 - Each time the browser requests a page to the server, all the data in the **cookie** is automatically **sent to the server** within the request.

PHP Cookies

- ❑ Cookies accompany both server requests and responses within the HTTP header.
- ❑ They are **not associated with a specific page** but with the page's domain,
 - Thus, *the browser and server will exchange cookie information* no matter what page the user requests from the site.
 - The *browser manages the cookies for the different domains* so that one domain's cookies are not transported to a different domain.
- ❑ Cookies can be:
 - **Temporary cookies** remain available only for the current browser session
 - **Persistent cookies** remain available beyond the current browser session and are stored in a text file on a client computer
- ❑ Each individual server or domain can store between **20** and **70** cookies on a user's computer
 - Total cookies per browser cannot exceed **300**
 - The largest cookie size is 4 kilobytes

How Do Cookies Work?



"Fundamentals of Web Development"
Book by Randy Connolly and Ricardo Hoar, 2015

PHP Cookies

- ❑ The `setcookie()` function is used to set a cookie in PHP.

```
setcookie( name, value, expire, path, domain, secure );
```

Parameter	Description
<code>name</code>	The name of the cookie. (<i>required</i>)
<code>value</code>	The value of the cookie. Do not store sensitive information since this value is stored on the user's computer.
<code>expires</code>	The expiry date in UNIX timestamp format. After this time cookie will become inaccessible. (<i>default value is 0</i>)*
<code>path</code>	Specify the <u>path on the server for which the cookie will be available</u> . If set to /, the cookie will be available within the entire domain. (optional)
<code>domain</code>	(optional). Specify the <u>domain for which the cookie is available</u> to e.g www.example.com.
<code>secure</code>	This field, if present, indicates that the cookie should be sent <u>only if a secure HTTPS connection exists</u> .

* If the expiration time of the cookie is set to 0, or omitted, the cookie **will expire at the end of the session** i.e. when the browser closes.

Create a Cookie: example

```
<?php
// Setting a cookie
setcookie("username", "Ahmed", time()+30*24*60*60);
?>
```

- Here's an example that uses `setcookie()` function to create a cookie named `username` and assign the value `Ahmed` to it. It also specifies that the cookie will expire after 30 days (`30 days * 24 hours * 60 min * 60 sec`).
- To skip any argument (except the expire argument), you can replace an argument with an empty string (`""`)
- To skip the expire argument use a zero (0) instead, since it is an integer
- Make sure **you call the `setcookie()` function before any output generated by your script**, otherwise cookie will not set.
 - **Cookies have to be sent before the heading elements** >> Example →

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head><title>PHP Script using Cookies</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
<?php
$strValue = "This is my first cookie";
setcookie ("mycookie", $strValue);
echo "Cookie set<br>";
?>
</body>
</html>

```

Gets an error!:

Warning: Cannot modify header information - headers already sent by (output started at_headers.php:9) in_headers.php on line 11

This is the correct approach!

```

<?php
$strValue = "This is my first cookie";
setcookie ("mycookie", $strValue);
echo "Cookie set<br>";
?>

```

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head><title>PHP Script using Cookies</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
</head>
<body>
  <?php
    echo "<p> A cookie has been set. </p>";
  ?>
</body>
</html>

```

How to Retrieve a Cookie: example

```
<?php
    $cookie_name = "username";
    $cookie_value = "Ahmed";
    setcookie($cookie_name, $cookie_value,
    time()+(86400 * 30), "/"); //86400 = 1 day
?>

<html>
<body>

<?php
    if(!isset($_COOKIE[$cookie_name])) {
        echo "Cookie named '" . $cookie_name . "' is not set!";
    } else {
        echo "Cookie '" . $cookie_name . "' is set!<br>";
        echo "Value is: " . $_COOKIE[$cookie_name];
    }
?>
</body>
</html>
```

The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

How to Delete a Cookie?

- ❑ It will expire !
- ❑ You can delete a cookie by calling the same `setcookie()` function with the cookie name and any value (such as an empty string) however this time you need the **set the expiration date in the past**, as shown in the example below:

```
<?php
// Deleting a cookie
setcookie("username", "", time()-3600);
?>
```

- ❑ You should pass exactly the same path, domain, and other arguments that you have used when you first created the cookie in order to ensure that the correct cookie is deleted.

Where is the cookie stored?

- ❑ Depends on the browser...

- ❑ How do I view and control cookies in my web browser?
 - <https://kb.iu.edu/d/ajfi>

Cookies issues

- ❑ Cookies **can be blocked** as said earlier but some websites become not as effective when we don't accept them.
- ❑ To get the full functionality of websites, we accept the cookies with concerns about privacy and tracking

- ❑ **Security issues:**
 - Since cookies are stored on user's computer it is possible for an attacker to easily modify a cookie content to insert potentially harmful data in your application that might break your application.

- ❑ **Performance issue:**
 - every time the browser requests a URL to the server, all the cookie data for a website is automatically sent to the server within the request.
 - If you have stored 5 cookies on user's system, each having 4KB in size, the browser needs to upload 20KB of data each time the user views a page, which can affect your site's performance.

PHP Sessions

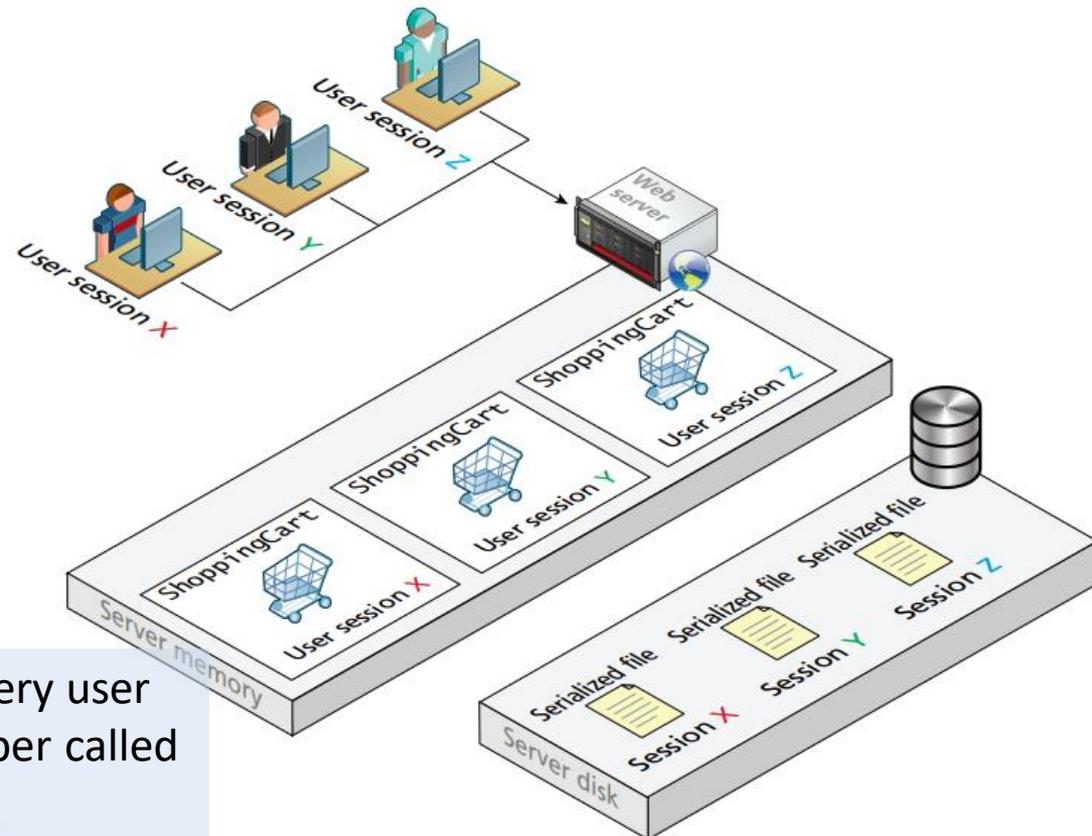
- ❑ **Sessions** are like cookies in which they provide a way for a server to track user's data over a series of pages

Sessions	Cookies
Stores data on the server rather than user's computer .	Stored on the users computer.
More secure (data is not transmitted between server and client)	Easier to create and retrieve
Store more information than a cookies	Require slightly less work from the server
Sessions can work even if user does not accept cookies	Persist over a longer period of time

PHP Sessions ..

❑ **Session** is a **server-based state mechanism** that lets web applications store and retrieve objects of any type for each unique user session.

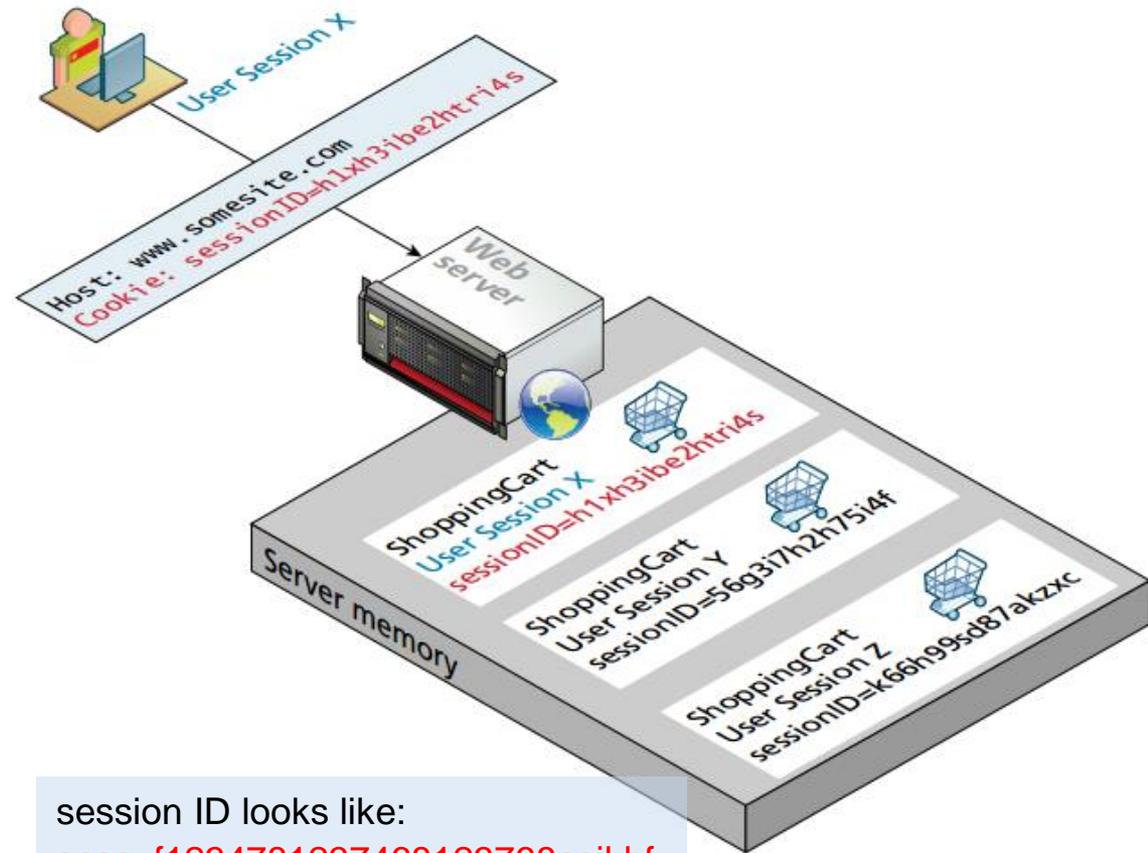
❑ That is, each browser session has its **own session state** stored as a serialized file on the server, which is deserialized and loaded into memory as needed for each request.



In a session based environment, every user is identified through a unique number called **session identifier** or **SID**.

How Does session state Work?

- ❑ The session state works within the **same HTTP context** as any web request.
- ❑ The server needs to be able to identify a given HTTP request with a specific user request.
- ❑ Since **HTTP is stateless**, some type of user/session identification system is needed.
- ❑ Sessions in PHP are identified with a **unique session ID**.
- ❑ In PHP, this is **a unique 32-byte string that is by default transmitted back and forth between the user and the server via a session cookie**.



session ID looks like:

```
sess_f1234781237468123768asjkhf  
a7891234g
```

Starting a PHP Session

- ❑ In PHP, session state is available to the developer as a superglobal associative array, much like the `$_GET`, `$_POST`, and `$_COOKIE` arrays.
- ❑ It can be accessed via the `$_SESSION` variable,
 - but unlike the other superglobals, you have to take additional steps in your own code in order to use the `$_SESSION` superglobal.
- ❑ A session is started with the `session_start()` function (at the beginning of the script)

```
<?php
    // Starting session
    session_start();
?>
```

- This tells PHP that a session is requested.
- A session ID is then allocated at the server end.

The session IDs are randomly generated by the PHP engine which is almost impossible to guess. Furthermore, because the session data is stored on the server, it doesn't have to be sent with every browser request.

Starting a PHP Session..

- ❑ The `session_start()` function first checks to see if a session already exists by looking for the presence of a session ID.
 - If it finds one, i.e. if the session is already started, it sets up the session variables and if doesn't, it starts a new session by creating a new session ID.

>> You must call the `session_start()` function at the beginning of the page i.e. before any output generated by your script in the browser, much like you do while setting the cookies with `setcookie()` function.

- ❑ Session variables store user information to be used across multiple pages (e.g. username, favorite color, etc).
 - By default, session variables last until the user closes the browser.

Storing and Accessing Session Data

- ❑ All session data are **stored as key-value pairs** in the `$_SESSION[]` superglobal array.
- ❑ The stored data can be accessed during lifetime of a session.
 - Consider the following script, which creates a new session and registers two session variables.

```
<?php
    // Starting session
    session_start();
    // Storing session data
    $_SESSION["firstname"] = "Ahmed";
    $_SESSION["lastname"] = "Sami";
?>
```

- ❑ To access the session data, simply recreate the session by calling `session_start()` and then pass the corresponding key to the `$_SESSION` associative array.

```
<?php
    // Starting session
    session_start();
    // Accessing session data
    echo 'Hi, ' . $_SESSION["firstname"] . ' ' .
    $_SESSION["lastname"];
?>
```

Hi, Ahmed Sami

Destroying a Session

- ❑ If you want to **remove certain session** data, simply **unset** the corresponding key of the `$_SESSION` associative array, as shown in the following example:

```
<?php
    // Starting session
    session_start();
    // Removing session data
    if(isset($_SESSION["lastname"])) {
        unset($_SESSION["lastname"]);
    }
?>
```

- ❑ However, to **destroy a session completely**, simply call the `session_destroy()` function. This function does not need any argument and a single call destroys all the session data.

```
<?php
    // Starting session
    session_start();
    // Destroying session
    session_destroy();
?>
```

Now, let's create a new page called "`demo_session1.php`". In this page, we start a new PHP session and set some session variables:

```
<?php
// Start the session
session_start();
?>

<!DOCTYPE html>
<html> <body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
echo "Session variables are
set.";
?>

</body> </html>
```

Next, we create another page called "`demo_session2.php`". From this page, we will access the session information we set on the first page ("`demo_session1.php`").

```
<?php
session_start();
?>

<!DOCTYPE html>
<html> <body>

<?php
// Echo session variables that
were set on previous page
echo "Favorite color is
" . $_SESSION["favcolor"];
?>
</body> </html>
```

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`).

To change a session variable, just overwrite it:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable,
just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

To remove all global session variables and destroy the session

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

```
<form action="login.php" method="post">
  <p>User Name: <input type="text" name="username" /></p>
  <p>Password: <input type="password" name="password" /></p>
  <p><input type="Submit" /></p>
</form>
```

```
1 <?php
2 session_start(); // Starting Session
3 $error=''; // Variable To Store Error Message
4
5 if($_SERVER['REQUEST_METHOD'] == 'POST'){
6     if (empty($_POST['username']) || empty($_POST['password'])) {
7         $error = "Username or Password is invalid";
8     }else
9     {
10         // Define $username and $password
11         $username=$_POST['username'];
12         $password=$_POST['password'];
13
14         if ($username== 'Ahmed' && $password == '123') {
15             $_SESSION['username'] = 'ahmed';
16             header("location: profile.php"); //redirect to another page
17         }else{
18             $error = "Username or Password is invalid";
19         }
20     }
21 } else{
22     $error = "You are not authorized to access this page";
23 }
24 echo $error;
25 ?>
```

```
1 <?php
2 session_start();
3 if (isset($_SESSION['username'])){
4     echo "Hello,". $_SESSION['username'];
5 }else {
6     echo "You are not authorized to access this page";
7 }
8 ?>
9
10 <?php
11     if($_SERVER['REQUEST_METHOD'] == "POST" and isset($_POST['logout']))
12     {
13         logout();
14     }
15     function logout(){
16         unset($_SESSION["username"]);
17         echo 'You have cleaned session';
18         header('Refresh: 2; URL = index.php');
19     }
20 ?>
21
22 <html>
23 <body>
24 <form action="profile.php" method="post">
25     <input type="submit" name="logout" value="Sign out" />
26 </form>
27 </body>
28 </html>
```

