ISIT307 - WEB SERVER PROGRAMMING

LECTURE 6.1 – MANAGING STATE INFORMATION

LECTURE PLAN

- Learn about state information
- Use hidden form fields to save state information
- Use query strings to save state information
- Use cookies to save state information
- Use sessions to save state information

UNDERSTANDING STATE INFORMATION

- Information about individual visits to a Web site is called **state information**
- HTTP was originally designed to be **stateless** – Web browsers store no persistent data about a visit to a Web site
- **Maintaining state** means to store persistent information about Web site visits

UNDERSTANDING STATE INFORMATION (CONTINUED)

- Customize individual Web pages based on user preferences
- Temporarily store information for a user as a browser navigates within a multipart form
- Allow a user to create bookmarks for returning to specific locations within a Web site
- Provide shopping carts that store order information
- Store user IDs and passwords
- Use counters to keep track of how many times a user has visited a site

UNDERSTANDING STATE INFORMATION (CONTINUED)

- The four tools for maintaining state information with PHP are:
  - Hidden form fields
  - Query strings
  - Cookies
  - Sessions

USING HIDDEN FORM FIELDS TO SAVE STATE INFORMATION

- Hidden form fields are created with the `<input>` element
- **Hidden form fields** temporarily store data that needs to be sent to a server that a user does not need to see
- The syntax for creating hidden form fields is:

```
<input type="hidden">
```

USING HIDDEN FORM FIELDS TO SAVE STATE INFORMATION (CONTINUED)

- Hidden form field attributes are **name** and **value**
- When submitting a form to a PHP script, access to the values submitted from the form is with the `$_GET[]` and `$_POST[]` autoglobals
- To pass form values from one PHP script to another PHP script, the values can be stored in hidden form fields as well
- The disadvantages of hidden boxes
  - Increased transmission times as the state data are sent back and forth between the client and server for each request
  - Requiring requests sent to the server are preferred to be "post" instead of "get"
  - Potential for being altered by the user maliciously

USING HIDDEN FORM FIELDS TO SAVE STATE INFORMATION - EXAMPLE

```
echo "<form method='post' " . " action='AvailableOpportunities.php'>\n";

. . .

echo "<input type='hidden' name='internID' " . " value='$InternID'>\n";

echo "<input type='submit' name='submit' " .

                          " value='View Available Opportunities'>\n";

echo "</form>\n";
```

USING QUERY STRINGS TO SAVE STATE INFORMATION

- A **query string** is a set of name=value pairs appended to a target URL
- Consists of a single text string containing one or more pieces of information
- Add a question mark (?) immediately after the URL followed by the query string that contains the information you want to preserve in name/value pairs

USING QUERY STRINGS TO SAVE STATE INFORMATION - EXAMPLE

- Example

```
<a href="http://www.example.com/TargetPage.php?
                        firstName=Elena&lastName=Vlahu&
                        occupation=lecturer">Link Text</a>
```

  - Accessing query string data

```
echo "{$_GET['firstName']} {$_GET['lastName']} is a
{$_GET['occupation']}. ";
```

USING COOKIES TO SAVE STATE INFORMATION

- Query strings do not permanently maintain state information
- After a Web page that reads a query string or hidden fields closes, the state information is lost
- To store state information beyond the current Web page session, cookies can be created
- **Cookies**, or magic cookies, are small pieces of information about a user/session that are stored by a Web server in text files on the user's computer

USING COOKIES TO SAVE STATE INFORMATION (CONTINUED)

- **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

CREATING COOKIES

- The syntax for the `setcookie()` function is:

  `setcookie(`*`name`* `[,`*`value`* `,`*`expires`*`, `*`path`*`, `*`domain`*`, `*`secure`*`])`

- You must pass each of the arguments in the order specified in the syntax
- To skip the `value`, `path`, and `domain` arguments, an empty string as the argument value can be specified
- To skip the `expires` and `secure` arguments, 0 as the argument value can be specified

CREATING COOKIES (CONTINUED)

- The `setcookie()` function should be called before sending the Web browser any output, including white space, HTML elements, or output from the `echo()` or `print()` statements
- Users can choose whether to accept cookies that a script attempts to write to their system
- A value of `TRUE` is returned even if a user rejects the cookie

CREATING COOKIES (CONTINUED)

- By default, cookies cannot include semicolons or other special characters, such as commas or spaces, that are transmitted between Web browsers and Web servers using HTTP
- Cookies ***can*** include special characters when created with PHP since encoding converts special characters in a text string to their corresponding hexadecimal ASCII value

THE NAME AND VALUE ARGUMENTS

- Cookies created with only the `name` and `value` arguments of the `setcookie()` function are temporary cookies because they are available for only the current browser session

```
<?php
setcookie("firstName", "Elena");
?>
<!DOCTYPE html>
<html>
<head>
<title>caption</title>
...
```

THE NAME AND VALUE ARGUMENTS (CONTINUED)

- The `setcookie()` function can be called multiple times to create additional/update cookies – as long as the `setcookie()` statements come before any other output on a Web page

```
setcookie("firstName", "Elena");
setcookie("lastName", "Vlahu");
setcookie("occupation", "lecturer");
```

THE NAME AND VALUE ARGUMENTS (CONTINUED)

- The following code creates an indexed/associative cookie array named `professional[]` that contains three cookie values:

```
setcookie("professional[0]", "Elena");
setcookie("professional[1]", "Vlahu");
setcookie("professional[2]", "lecturer");
-----
setcookie("professional[firstName]", "Elena");
setcookie("professional[lastName]", "Vlahu");
setcookie("professional[occupation]", "lecturer");
```

THE `EXPIRES` ARGUMENT

- The `expires` argument determines how long a cookie can remain on a client system before it is deleted
- Cookies created without an `expires` argument are available for only the current browser session
- To specify a cookie's expiration time, can be used PHP's `time()` function

```
setcookie("firstName", "Elena", time()+3600);
```

THE `PATH` ARGUMENT

- The `path` argument determines the availability of a cookie to other Web pages on a server
- Using the `path` argument allows cookies to be shared across a server
- A cookie is available to all Web pages in a specified path as well as all subdirectories in the specified path

```
setcookie("firstName", "Elena", time()+3600, "/uow/");
```

THE `DOMAIN` ARGUMENT

- The `domain` argument is used for sharing cookies across multiple servers in the same domain
- Cookies cannot be shared outside of a domain

```
setcookie("firstName", "Elena", time()+3600, "/", ".uow.com");
```

THE `SECURE` ARGUMENT

- The `secure` argument indicates that a cookie can only be transmitted across a secure Internet connection using HTTPS or another security protocol
- To use this argument, assign a value of 1 (for `TRUE`) or 0 (for `FALSE`) as the last argument of the `setcookie()` function

```
setcookie("firstName", "Elena", time()+3600, "/", ".uow.com", 1);
```

READING COOKIES

- Cookies that are available to the current Web page are automatically assigned to the `$_COOKIE` autoglobal
- Each cookie can be accessed by using the cookie name as a key in the associative `$_COOKIE[]` array

```
echo $_COOKIE['firstName'];
```

- Newly created cookies are not available until after the current Web page is reloaded

READING COOKIES (CONTINUED)

- To ensure that a cookie is set before using it, the `isset()` function can be used

```
setcookie("firstName", "Elena");
setcookie("lastName", "Vlahu");
setcookie("occupation", "lecturer");
---
if (isset($_COOKIE['firstName'])
     && isset($_COOKIE['lastName'])
     && isset($_COOKIE['occupation']))
     echo "{$_COOKIE['firstName']} {$_COOKIE['lastName']}
           is a {$_COOKIE['occupation']}.";
```

READING COOKIES (CONTINUED)

- Multidimensional array syntax can be used to read cookies stored as indexed/associative arrays

```
setcookie("professional[0]", "Elena");
setcookie("professional[1]", "Vlahu");
setcookie("professional[2]", "lecturer");
---
if (isset($_COOKIE['professional']))
     echo "{$_COOKIE['professional'][0]}
          {$_COOKIE['professional'][1]} is a
          {$_COOKIE['professional'][2]}.";
```

DELETING COOKIES

- To delete a persistent cookie before the time assigned to the `expires` argument elapses, a new expiration value that is sometime in the past should be assigned
- This can be done by subtracting any number of seconds from the `time()` function

```
setcookie("firstName", "", time()-3600);
setcookie("lastName", "", time()-3600);
setcookie("occupation", "", time()-3600);
```

USING SESSIONS TO SAVE STATE INFORMATION

- A **session** refers to a period of activity when a PHP script stores state information on a Web server
- Sessions allow maintaining state information even when clients disable cookies in their Web browsers

STARTING A SESSION

- The `session_start()` function starts a new session or continues an existing one
- The `session_start()` function generates a unique session ID to identify the session
- A **session ID** is a random alphanumeric string that looks something like:

  `7f39d7dd020773f115d753c71290e11f`
- The `session_start()` function creates a text file on the Web server that is the same name as the session ID, preceded by `sess_`

STARTING A SESSION (CONTINUED)

- Session ID text files are stored in the Web server directory specified by the `session.save_path` directive in the php.ini configuration file
- The `session_start()` function does not accept any arguments, nor does it return a value that can be used in the script

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

STARTING A SESSION (CONTINUED)

- The `session_start()` function must be called before sending the Web browser any output
- If a client's Web browser is configured to accept cookies, the session ID is assigned to a temporary cookie named `PHPSESSID`
- The session ID can be send as a query string or hidden form field to any Web pages that are called as part of the current session

STARTING A SESSION (CONTINUED)

- The `session_id()` function can be used to get or set the session id for the current session
- The constant `SID` can also be used to retrieve the current name and session id as a string suitable for adding to URLs

```
<?php
session_start();
...
?>
<p><a href='<?php echo "Occupation.php?PHPSESSID="
                . session_id(); ?>'>Occupation</a></p>
OR
<p><a href='<?php echo "Occupation.php?"
                . SID; ?>'>Occupation</a></p>
```

WORKING WITH SESSION VARIABLES

- Session state information is stored in the `$_SESSION` autoglobal
- When the `session_start()` function is called, PHP either initializes a new `$_SESSION` autoglobal or retrieves any variables for the current session (based on the session ID) into the `$_SESSION` autoglobal

WORKING WITH SESSION VARIABLES (CONTINUED)

```
<?php
session_start();
$_SESSION['firstName'] = "Elena";
$_SESSION['lastName'] = "Vlahu";
$_SESSION['occupation'] = "lecturer";
?>
---
<p><a href='<?php echo "Occupation.php?"
                . SID ?>'>Occupation</a></p>
```

WORKING WITH SESSION VARIABLES (CONTINUED)

- The `isset()` function can be used to ensure that a session variable is set before you attempt to use it

```
<?php
session_start();
if (isset($_SESSION['firstName']) &&
    isset($_SESSION['lastName'])&&
    isset($_SESSION['occupation']))
      echo "<p>" . $_SESSION['firstName'] . " "
           . $_SESSION['lastName'] . " is a "
           . $_SESSION['occupation'] . "</p>";
?>
```

DELETING A SESSION

- To delete a session manually, should be performed the following steps:

  1. Execute the `session_start()` function
  2. Use the `array()` construct to reinitialize the `$_SESSION` autoglobal
  3. Use the `session_destroy()` function to delete the session

DELETING A SESSION (CONTINUED)

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

MANAGING STATE INFORMATION

- Some *php.ini* settings
  - session.use_strict_mode = 0
  - session.use_cookies = 1
  - session.use_only_cookies = 0
  - session.name = PHPSESSID
  - session.use_trans_sid = 0 / session.use_trans_sid = 1

MANAGING STATE INFORMATION

- Example – College Internship

UNDERSTANDING STATE INFORMATION - EXAMPLE