Open Source Week 4 Summary.

Contents

[An Introduction to Stateless and Sessions 3](#_Toc129170828)

[HTTP and State 3](#_Toc129170829)

[Sessions 3](#_Toc129170830)

[Three Soloutions 4](#_Toc129170831)

[Hidden Fields 4](#_Toc129170832)

[Cookies 5](#_Toc129170833)

[Session variables 6](#_Toc129170834)

[Design with UML 8](#_Toc129170835)

[Why design? 8](#_Toc129170836)

[What is being designed? 8](#_Toc129170837)

[Notation 8](#_Toc129170838)

[Client Pages 8](#_Toc129170839)

[Forms 9](#_Toc129170840)

[Control Types 9](#_Toc129170841)

[Radio Buttons within forms 10](#_Toc129170842)

# An Introduction to Stateless and Sessions

## HTTP and State

HTTP is a stateless protocol, which is a drawback of HTTP. Stateless means each command is executed independently with no memory of a previous command. State is needed to react intelligently to user input. Maintaining state is extremely useful when complex applications are being developed.

## Sessions

A session is a sequence of related interactions between a browser and a web server. Often it is necessary to record the data entered into a series of forms and then towards the end of the session, retrieve these values. To achieve this, persistent variables are required. An example of a session can be an insurance quotation, where data is input and then later displayed.

Here is an example:

The browser determines which file to request.

|  |
| --- |
| Input.html |
| <form action="confirm.php" method ="post">  <input type ="text" name="txtaccount" />  The browser generates a request for confirm.php, enclosing the form within the request body  <input type ="submit"/>  </form> |

|  |
| --- |
| confirm.html |
| <? Echo “your account is $\_POST[‘txtaccount’]; ?> |

The scripting engine creates an element with $\_POST for each control within the form that is sent within the request

Here is another example:

The browser generates a request for confirmAccount.php, ConfirmDeposit.php enclosing the form within the request body

The browser determines which file to request.

|  |
| --- |
| InputAccount.html |
| <form action=”confirmAccount.php” method=”post”>  <input type=”text” name=”txtaccount”/>  <input type=”submit”/>  </form> |

|  |
| --- |
| confirmAccount.php |
| <? Echo “your account is $\_POST[txtaccount]”; ?>  <form action=”confirmdeposit.php” method=”post”>  <input type=”text” name-“txtdeposit”/>  <input type=”submit”/>  </form> |

Unlike form elements the $\_POST array is NOT passed within the request, so *txtaccount* element is not recognised. (Stateless=No memory)

|  |
| --- |
| confirmDeposit.php |
| <?  The scripting engine creates an element with $\_POST for each control within the form that is sent within the request  Echo “Your wish to deposit $\_POST[txtdeposit]”;  Echo “Your account is $\_POST[txtaccount]”; |

## Three Soloutions

There are three soloutions to mitigate issues regarding HTTP and its stateless initiative, being:

* ***Hidden HTML fields***

This solution creates hidden input form controls to hold values within the intermediary pages

* ***Cookies***

This solution crates name value pairs

* ***Session Variables***

### Hidden Fields

|  |
| --- |
| <input type=***”hidden”*** name=”name” value=”x”/> |

Hidden fields when used in a form can consist of hidden controls. They work similar to invisible Text controls. They are commonly used to contain data that is not visible on the web page but required by the server-side script. In order to implement a hidden control, simply set its *input type* to *hidden* like so:

Within a PHP file, we are able to assign the value of variable. The *echo* command must be included and the PHP statement must be enclosed within PHP tags. Here is an example:

|  |
| --- |
| <input type=”hidden” name=”hiddenAccount” value=***”php echo $\_POST[‘elementName’] ?>”*** /> |

In order to access the hidden field values, we will incorporate this code into our script, like so:

|  |
| --- |
| <form action=”confirmDeposit.php” method=”post”>  <input type=”text” name=”txtdeposit”/>  <***input type=”hidden” name=”hdaccount” value=”<?php echo $\_POST[‘txtaccount] ?>*** “/>  <input type=”submit”/>  </form |

|  |
| --- |
| confirmDeposit.php |
| <?php echo $\_POST[‘hdaccount’] ?> /> |

Now we understand the concept of Hidden Fields, we are now ready to implement it into our Banking Scenario script addressed earlier on:

***confirmAccount.php*** then displays the submitted value of ***txtaccount*** and also acquires data for ***txtdeposit*** to be submitted in ***confirmDeposit.php***. ***txtaccount*** is then defined as a hidden field under the name ***hdaccount.***

|  |
| --- |
| InputAccount.html |
| <form action=”confirmAccount.php” method=”post”>  InputAccount.html is used to gather information which will be input as ***txtaccount*** into ***confirmAccount.php***  <input type=”text” name=”txtaccount”/>  <input type=”submit”/>  </form> |

|  |
| --- |
| confirmAccount.php |
| <?php echo “your account is $\_POST[txtaccount]”; ?>  <form action=”confirmdeposit.php” method=”post”>  <input type=”text” name-“txtdeposit”/>  <input type=”hidden” name=”hdaccount” value=”<?php echo $\_POST[‘txtaccount’] ?> />  <input type=”submit”/>  ***Confirmdeposit.php*** then echo’s out the submitted data of both ***txtdeposit*** and ***hdaccount***which is the previously named ***txtaccount*** field.  </form> |

|  |
| --- |
| confirmDeposit.php |
| <?php  Echo “your wish to deposit $\_POST[‘txtdeposit’];  Echo “your account is $\_POST[‘hdaccount’];  ?> |

### Cookies

|  |
| --- |
| Setcookie(firstname, “Javed Parvez”); |

Cookies are name value pairs such as *age=21*. They can be created inside PHP script using the ***setcookie()*** function which takes two arguments; the name of the cookie, and cookie value. Here is how you would create a cookie:

This would create the cookie: firstname=Javed Parvez.

|  |
| --- |
| $cookieName = “firstname”  Here, ***$cookieName*** is being set to ‘***firstname’***, with its value named as ***‘name’*** and set to ***‘Javed Parvez’***  $name = “Javed Parvez”;  Setcookie($cookieName, $name); |

It is also possible to create cookies based on the values contained within variables or elements of an array, for example:

|  |
| --- |
| <?php  Echo ***$\_COOKIE[‘firstname’];***  ?> |

Now that the cookies have been set, we can call them by using the PHP $tag of ***$\_COOKIE*** paired with the cookie name, for example:

The general template of ***$\_COOKIE*** is:  
***$\_COOKIE[‘****cookiename(x)’****].***

The ***$\_COOKIE*** array contains all the cookies created within a single Web Application session.

We will now use cookies to develop our application system.

This file is simply setting input types and its name to be submitted to ***confirmAccount.php***

|  |
| --- |
| InputAccount.html |
| <form action=”confirmAccount.php” method=”post”>  <input type=”text” name=”txtaccount”/>  <input type=”submit”/>  </form> |

|  |
| --- |
| confirmAccount.php |
| <? $cookieName = ‘accountNumber’;  ***$cookieName*** is defined with the name ***‘accountNumber’*** and is associated with value of ***txtaccount*** from ***InputAccount.HTML.*** This is then echo’d out. An input is then set-up by the name ***txtdeposit*** and is submitted to ***confirmDeposit.php.***  Setcookie($cookiename, $\_POST[‘txtaccount’]); ?>  <? Echo “Your account is $\_POST[‘txtaccount’]; ?>  <form action =”confirmDeposit.php” method=”post”>  <input type=”text” name =”txtdeposit” />  <input type =”submit”/>  </form> |

|  |
| --- |
| confirmDeposit.php |
| <?  Echo “your wish to deposit $\_POST[txtdeposit]”;  Echo “your account is “. $\_COOKIE[‘accountNumber’];  ?> |

Here, ***txtdeposit*** is echo’d, as well as the ***accountNumber*** cookie defined in ***confirmAccount.php.***

### Session variables

|  |
| --- |
| ***$\_SESSION[***‘firstname’] = “Guy”; |

Session variables are visible to all scripts within the current session. They are stored within the ***$\_SESSION*** associative array, managed by PHP itself. In order to create a new session variable, simply create a new index with the ***$\_SESSION*** array, and assign it a value, for example:

The general template for setting a session variable is   
***$\_SESSION[‘sessionName(x)’] = “value”;***

|  |
| --- |
| *echo* ***$\_SESSION[***‘firstname’]; |

To access the session variable, simply type out its array followed by its set value, so for this scenario we would type:

|  |
| --- |
| *<?php*  ***Session\_start();***  echo $\_SESSION[‘firstname’];  ?> |

In order to use the ***$\_SESSION*** associative array, a session must be started using the ***session\_start()*** function which should be enclosed in the PHP tags, for example:

Now that this is understood, we can now incorporate it into our banking application

|  |
| --- |
| InputAccount.html |
| <form action=”confirmAccount.php” method=”post”>  This file is simply setting input types and its name to be submitted to ***confirmAccount.php***  <input type=”text” name=”txtaccount”/>  <input type=”submit”/>  </form> |

|  |
| --- |
| confirmAccount.php |
| <? session\_start(); ?>  As you can see, ***session\_start()*** is used to initialize a session in this .php file. After the ***txtaccount*** is echo’d, ***$\_SESSION*** is used to set ***accountNumber*** as a session with ***txtaccount*** as its value.  <? Echo “your account is $\_POST[‘txtaccount’];  $\_SESSION[‘accountNumber’] = $\_POST[txtaccount]; ?>  <form action =”confirmDeposit.php” method=”post”>  <input type=”text” name =”txtdeposit” />  <input type =”submit”/>  </form> |

|  |
| --- |
| confirmDeposit.php |
| <?  Session\_start()  Echo “Your wish to deposit $\_POST[‘txtdeposit’]”;  Echo “Your account is “ . $\_SESSION[‘accountNumber’];  ?> |

***Session\_start()*** is used once again in order to allow for the use of the ***$\_SESSION*** array in order to echo both ***txtdeposit*** and ***accountNumber***

# Design with UML

In this section we will be discussing Web Application Design, Unified Modelling Language(UML) and Web Application Extensions(WAE).

## Why design?

Designing before development allows to errors to be recognised and rectified prior to early development. Designing provides a visual representation which can be understood and acknowledge by all contestants of an objective.

## What is being designed?

Regarding PHP, we tend to design based on three elements consisting of their own attributes. These elements are:

* ***HTML***, which consists of HTML files, forms and the controls within these forms
* ***PHP***, which consists of PHP scripts and its dynamically created HTML pages
* ***Navigation***, which consist of general links and form submission

## Notation

In order to provide an effective design, it is important to outline various elements of a design and specify its representation. Designs regarding programming use a notation revolving around boxes, arrows and arrow-types. Here are the element keys:

|  |  |
| --- | --- |
| Notation | Representation |
|  | Boxes represent classes within our system |
|  | Arrows between classes represent associations |
|  | A diamond on one end of the association indicates a contained class |
| **<<stereotype name>>** | Stereotypes within gullets identify specific types of class or associations |

## Client Pages

Client pages represent the HTML page displayed on a client’s browser. They are essentially the page(s) that the customer sees and interacts with. Within the design, they are visualized by a rectangle for example:

***<<Client Page>>***Class Name

The ***Class Name*** for static html pages is the file name, including the .html extension. In situations where a dynamic page is created by a script, it should be a one word description that relates to the page’s contents.

The Client Page stereotype is enclosed in gullets and placed over the filename.

Here is an example of implementation and design notation:

<html>  
 <head>  
 <title>HTMLFILE</title>  
 </head>  
 <body>  
 Hello I am an HTML file  
 </body>  
</html>

***<<Client Page>>***Hello.html

## Forms

The form is represented by its individual box. The Class name is the name applied to the form’s name attribute. Stereotype ***<<Form>>*** is included within the box above the class name like this:

***<<Form>>***Form Name

### Control Types

Within the form are various controls that can also be modelled within the design. The list of valid control types are: ***text, checkbox, submit, file, image, select, password, radio, reset, hidden, button, text area*** and can be represented by using ***controlname:(***type***)*** under the class name in the design notation.

***<<Form>>***Form Name  
*controlname:type*

Here is an example of a Form class w/ control type implementation with its design notation:

***<<Form>>***frmLogin  
*username:text  
pwd:password  
button1:submit*

<form name=”frmLogin” >  
 <input type=”text” name=”username” />  
 <input type=”password” name=”pwd” />  
 <input type=”submit” />  
</form>

### Radio Buttons within forms

Radio buttons can be grouped within a form to allow for only one member of the group to be selected at any given time. This is visualized within the design notation by allocating the grouped radio button the same name but with a different index (ControlName[1],ControlName[2]etc..) Here is an example of both implementation and its design notation:

***<<Form>>***frmSelection  
Card[1]: radio = “visa”  
Card[2]: radio = “access”  
button1: submit

<form name=”frmSelection” >  
 <input type=”radio” name=”card” value=”visa” />  
 <input type=”radio” name=”card” value=”access” />  
 <input type=”submit” />  
</form>

## HTML Files and Forms

A containment association represented by a diamond-ended directional arrow is drawn between the Web page and the form to indicate that the form is part of the web page and cannot exist independently.

***<<Client Page>>  
ClassName***

***<<Form>>  
ClassName***

Here would be the implementation of the design notation above in a detailed form:

***<<Form>>***frmLogin  
user:Text  
button1:submit

***<<Client Page>>***Login.html

<html>  
 <head>  
 <title>HTMLFILE</title>  
 </head>  
 <body>  
 <form name=”frmLogin” >  
 <input type=”text” name=”user”/>  
 <input type=”submit” />  
 </form>  
 </body>  
</html