LAB 16

Managing State

What You Will Learn

- How to use cookies to manage state
- How to use sessions to manage state
- Some simple page caching that takes account of state.

Approximate Time

The exercises in this lab should take approximately 60 minutes to complete.

Fundamentals of Web Development, 2nd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
http://www.funwebdev.com

Date Last Revised: May 26, 2017

CLIENT-SIDE STATE

PREPARING DIRECTORIES

1 If you haven't done so already, create a folder in your personal drive for all the labs for this book.

2 From the main `labs` folder (either downloaded from the textbook's web site using the code provided with the textbook or in a common location provided by your instructor), copy the folder titled `lab16` to your course folder created in step one.

This lab walks you through several techniques that let you manage state in web applications. Without cookies, sessions, or other mechanisms sent through HTTP headers, a web server couldn't easily differentiate two users (especially behind the same IP address).

This lab makes use of a simple MySQL database in many examples to facilitate working with cookies and sessions. It assumes you have already completed the lab and exercises for chapter 14.

Exercise 16.1 — USING COOKIES

1 Execute the database commands in lab16-exercise01.sql. This will create tables to store user credentials.You may have already created this database in the lab for Chapter 14.

2 Open, examine, and test lab16-exercise01.php. You will see a webpage containing a single login form. The page is also configured to connect to the database you just created. If you try to enter information and submit, the form is posted but no action is taken (aside from saying "Login attempted").

3 Add a function to check the posted credentials against the credentials stored in the database. If the user was successful, output a welcome message, otherwise output an error message with the form using code like:

```
function validLogin(){
   $pdo = new PDO(DBCONNSTRING,DBUSER,DBPASS);
   //very simple (and insecure) check of valuid credentials.
   $sql = "SELECT * FROM Credentials WHERE Username=:user and
Password=:pass";
   $statement = $pdo->prepare($sql);
   $statement->bindValue(':user',$_POST['username']);
   $statement->bindValue(':pass',$_POST['pword']);
   $statement->execute();
   if($statement->rowCount()>0){
     return true;
   }
   return false;
}
```

Now use this newly defined function in our main logic as follows:

```
$loggedIn=false;

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if(validLogin()){
      echo "Welcome ".$_POST['username'];
        $loggedIn=true;
    }
    else{
       echo "login unsuccessful";
    }
}
else{
        echo "No Post detected";
}
```

Finally, where we used to echo the form out, we can conditionally echo it only when no one is logged in

```
if(!$loggedIn){
   echo getLoginForm();
}
else{
   echo "This is some content";
}
```

4 Once you log in successfully, you will see the welcome message, but if you refresh the URL the script will not remember your successful login, and you will be prompted to login again. This is because HTTP has no state,

We will now use cookies, so that when user logs in successfully, their credentials are stored in a cookie, and that cookie is subsequently examined for a valid logged in user. Where the user is authenticated correctly you add code to set the cookie:

```
if(validLogin()){
        echo "Welcome ".$_POST['username'];
        // add 1 day to the current time for expiry time
        $expiryTime = time()+60*60*24;
        setcookie("UserName", $_POST['username'], $expiryTime);
}
```

5 Now, before we even check for a POST we should consider whether a logged in user has a good cookie. Modify your code to check for the presence of a cookie but after checking for the post.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if(validLogin()){
        // add 1 day to the current time for expiry time
        $expiryTime = time()+60*60*24;
        setcookie("Username", $_POST['username'], $expiryTime);
    }
    else{
       echo "login unsuccessful";
    }
   }

   if(isset($_COOKIE['Username'])){
        echo "Welcome ".$_COOKIE['Username'];
   }
   else{
        echo "No Post detected";
   }
```

Similarly, chance your logic to use the cookie to determine whether or not to display the form:

```
if (!isset($_COOKIE['Username'])){
   echo getLoginForm();
}
else{
   echo "This is some content";
}
```

6 Finally we will create a logout file, and include a link to that file from our main page (left as an exercise) . The file, placed in logout.php will look like:

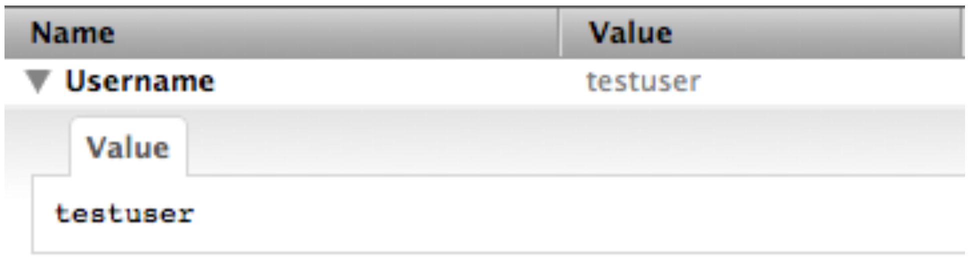
```
<?php

setcookie("Username", "", -1)
header("Location: ".$_SERVER['HTTP_REFERER']);

?>
```

Try logging out, and logging in, see how your cookie will persist until you logout (or the time expires).

Note: The way we store cookies in this exercise is insecure since we transmit plain text, easily modified values (as shown in Figure 16.1). Users could modify the cookie to gain access to other user's accounts. Instead an obfuscated value should be stored in the cookie, so that it can't easily be changed. Sessions do that automatically for us, but it is left as an exercise for the student interested in manual cookie security. *Hint: create and use a hash as the cookie value.*



Name	Value
▼ Username	testuser

Value

testuser

Figure 16.1 – A screenshot of firebug, which shows the cookie being transmitted.

Exercise 16.2 — SERIALIZE YOUR OBJECTS

1 Serialization is the means by which objects and data structures can be stored as files, and then resurrected as objects when the need arises. Before you make use of automatic serialization you will make manual use of it in your own classes. Open lab16-exercise02.php and the referenced Artist.class.php. This file creates several objects and prints them out to the screen.

2 To illustrate how serialization can help you with your own classes, make the Artist class implement the Serializable interface. This requires you to add to the class declaration, and implement two methods serilaize() and unserialize() in Artist.class.php

```
class Artist implements Serializable{
  //...
  public function serialize() {
     return serialize(array("earliest" => self::$earliestDate,
                            "first" => $this->FirstName,
                            "last" => $this->LastName,
                            "bdate" => $this->BirthDate,
                            "ddate" => $this->DeathDate,
                            "bcity" => $this->BirthCity,
                            "works" => $this->artworks
                            ));
  }
  public function unserialize($data) {
     $data = unserialize($data);
     self::$earliestDate = $data['earliest'];
     $this->FirstName = $data['first'];
     $this->LastName = $data['last'];
     $this->BirthDate = $data['bdate'];
     $this->DeathDate = $data['ddate'];
     $this->BirthCity = $data['bcity'];
     $this->artworks = $data['works'];
  }
  //...
}
```

3 Now, since the Artist itself has instances or Art, we must ensure the Art classes and subclasses are also serializable.. In Art.class.php make similar changes to serialize and unserialize those classes.

```
class Art implements Serializable{
  //...

  public function serialize() {
            return serialize(
                array("date" =>$this->dateCreated,
                      "name" => $this->name
                )
         );
 }
 public function unserialize($data) {
    $data = unserialize($data);
     $this->dateCreated = $data['date'];
    $this->name = $data['name'];
 }

 //...
}
```

Note that the Painting and Sculpture subclasses do not need to decalre that they implement Serializable, but they need to override the serialize() and unserialize() methods in order to ensure additional fields are properly stored

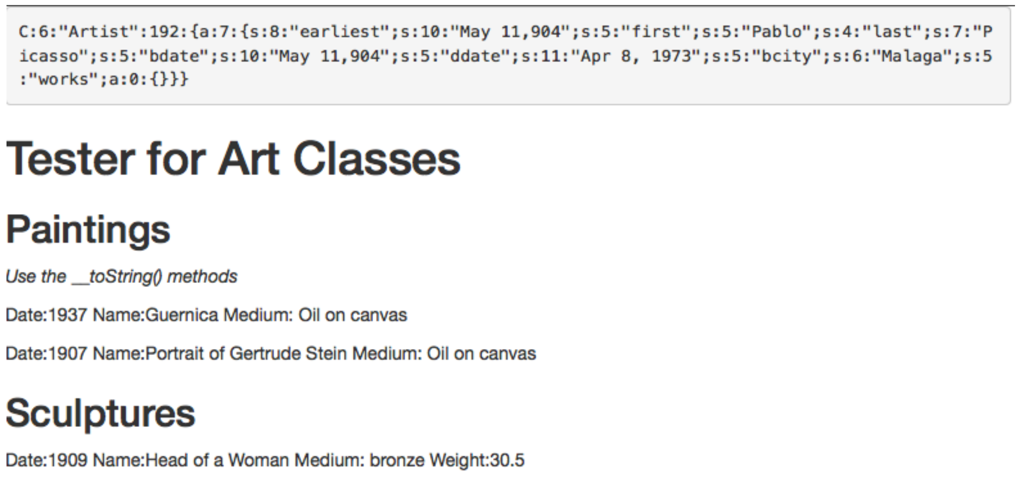
```
class Painting extends Art{
//...
     public function serialize() {
            return serialize(
                array("med" => $this->medium,
                        "artData" => parent::serialize()
                    )
         );
    }
    public function unserialize($data) {
    $data = unserialize($data);
        $this->medium = $data['med'];
        parent::unserialize($data['artData']);
    }
//..
}
class Sculpture extends Painting{
//..
     public function serialize() {
            return serialize(
                array("weight" =>$this->weight,
                        "paintingData" => parent::serialize()
                    )
         );
    }
    public function unserialize($data) {
    $data = unserialize($data);
```

```
        $this->weight = $data['weight'];
        parent::unserialize($data['paintingData']);
       }
//...
}
```

4 Now add the following lines to your lab16-exercise02.php to demonstrate the serialization fo objects, and then their subsequent unserialization.

```
$picassoAsFile = serialize($picasso);
echo "<pre width='100%'>$picassoAsFile</pre>";
$picasso = unserialize($picassoAsFile);
```

The output from the object is unchanged, but inbetween a serialized version of the object is output as shown in Figure 16.2



```
C:6:"Artist":192:{a:7:{s:8:"earliest";s:10:"May 11,904";s:5:"first";s:5:"Pablo";s:4:"last";s:7:"P
icasso";s:5:"bdate";s:10:"May 11,904";s:5:"ddate";s:11:"Apr 8, 1973";s:5:"bcity";s:6:"Malaga";s:5
:"works";a:0:{}}}
```

Tester for Art Classes

Paintings

Use the __toString() methods

Date:1937 Name:Guernica Medium: Oil on canvas

Date:1907 Name:Portrait of Gertrude Stein Medium: Oil on canvas

Sculptures

Date:1909 Name:Head of a Woman Medium: bronze Weight:30.5

Figure 16.2– Exercise 16.2 completed, showing the serialized objects.

A Better Way - Sessions

Exercise 16.3 - Using Sessions

1 This exercise will make use of your Exercise 1, but replace the cookies with sessions (which automatically use cookies).

2 Open your main file and logout.php and at the top add

```
session_start();
```

This will automatically make a session be transmitted as a cookie, and that session can then be associated with values just like a cookie.

3 Next change every where that we used to refernce cookie with the same code, but inside the $_SESSION.

Instead of setting a cookie:

```
setcookie("UserName", $_POST['username'], $expiryTime);
```

set a session variable

```
$_SESSION['Username']=$_POST['username'];
```

As a consequence, everywhere that you check $_COOKIE['username'] should now be $_SESION['Username']. So

```
if(isset($_SESSION['Username'])){
     echo "Welcome ".$_SESSION['Username'];
}

//...
if (!isset($_SESSION['Username'])){
   echo getLoginForm();
}
```

4 In logout.php we log the user out by wiping out the session rather than the cookie.

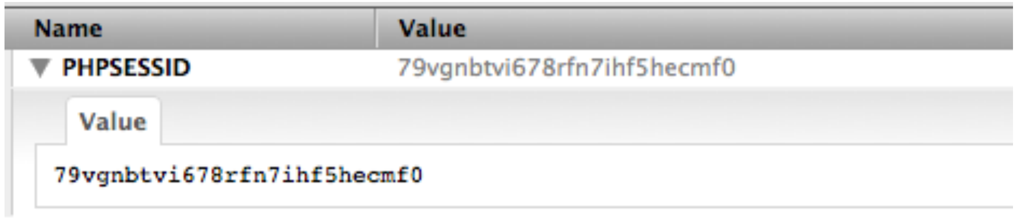
```
setcookie("Username", "", -1);
```

becomes

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

5 That's it. Try logging in as before and your authentication should "stick" just like you made it do with cookies. The benfit is that instead of relying on the value of the cookie,

the session mechanism automatically introduces an obfuscated cookie (as shown in Figure 16.2)



Name	Value
▼ PHPSESSID	79vgnbtvi678rfn7ihf5hecmf0

Value

79vgnbtvi678rfn7ihf5hecmf0

Figure 16.3 – Screenshot of Firebug, showing an auto generated PHP session cookie.

Don't bother with these remaining exercises

Exercise 16.4 — HTML5 Web Storage

1 Open lab16-exercise04.php and notice that it is conducting a 3 question survey using sessions. In order to get the next question the last question must be submitted, requiring multiple requests to the server to conduct the survey. Nothing is done with the results in this example.

2 We will change the page to post all 3 questions and use HTML5 storage to run the survey, collects the answers, and post at the end.

First we will remove the need for sessions to handle which question we are on (easier to debug as well). This means replacing the complicated session logic with the very straightforward:

```
<?php

echo "<h1 id='questionNumber'>Question #1</h1>";
echo "<h2>".getSurveyQuestion(0)."</h2>";


?>
```

Similarly, simplify the form to add a id (for Javascript) and change the input submit firled to a button attached to a JavaScript function.

```
function getSurveyQuestion($i){
  $questions = array("What is your favorite color?", "In what city were you
                      born?", "Your favorite drink is:");
  $form= "
  <form action='' method='get' role='form'>
  <div class ='form-group'>
    <label for='answer' id='question'>".$questions[$i]."</label>
    <input type='text' name='answer' id='answer' class='form-control'/>
```

```
   </div>";
   $form.="<input type='button' value='Next' class='form-control'
onclick='nextQ();'/>";
  $form.="</form>";
  return $form;
}
?>
```

3 Now add JavaScript tags and code to initialize all the questions in the HTML5 sessionStorage:

```
<script language="javascript" type="text/javascript">

if (typeof (localStorage) === "undefined" || typeof (sessionStorage) ===
"undefined") {
 alert("Web Storage is not supported on this browser...");
}
else {
  //gets serialized to a comma separated list of strings.
   sessionStorage.setItem("Questions",
new Array("What is your favorite color?", "In what city were you born?",
"Your favorite drink is:"));
 sessionStorage.setItem("Answers", "");
 sessionStorage.setItem("currentQuestion",0);
 // document.write("web storage modified");
}

</script>
```

4 Finally we will write the function that handles the button p[ress, stores the answer and changes over tot he next question.

```
function nextQ(){
  var currentIndex = sessionStorage.getItem("currentQuestion");
  var answerNode = document.getElementById("answer");
  var answer = answerNode.value;
  var oldAnswers = sessionStorage.getItem("Answers");
  if(oldAnswers!="")
    sessionStorage.setItem("Answers",oldAnswers+","+answer);
  else
    sessionStorage.setItem("Answers",answer);
  //Now increment to Next Question
  currentIndex = parseInt(currentIndex) + 1;
  sessionStorage.setItem("currentQuestion",currentIndex);
  var allQs = sessionStorage.getItem("Questions").split(",");
  if(allQs.length<=currentIndex){
     //echo for   now - survey completed.
      var allAs = sessionStorage.getItem("Answers").split(",");
     for (var i=0;i<allQs.length;i++){
      document.write(allQs[i]+":"+allAs[i]+"</br>");
     }
  }
```

```
    else{
      //Update the questions from SessionStorage
      var questionNode=document.getElementById("questionNumber");
      questionNode.innerHTML=("Question #"+(parseInt(currentIndex)+1));
      var questionNode=document.getElementById("question");
      questionNode.innerHTML=(allQs[currentIndex]);
    }
  }
```

5 Finally you will see that after the last question the values are echoes out to the screen, and still not handled by the server.

After we learn about JQuery and AJAX in Chapter 15, you might want to return to complete this Exercise by posting the final data back to the server.

For now, reflect on how HTML5 storage has allowed the entire survey to be conducted with only 1 request, compared to 3 requests needed for the Session based technique.

Exercise 16.5 — Cache a Page

1 Although page caching is covered in Chapter 23 in depth with Apache, this exercise walks you through a simple writing your own, custom built cache, using a database.

2 Execute the commands in lab16-exercise05.sql. This script builds a table that will store a cached HTML page and timestamp info for better caching.

3 Now open lab16-exercise04.php and notice that it prints a simple multiplication table based on the $_GET['size'] parameter. Visit

lab16-exercise04.php?size=10

To see the 10x10 multiplication table output.

4 Add code at the bottom of the file that writes the output HTML to the database using the GET parameter as the key, and stores the time as well.

```
$multTable =  getMultiplicationTable($_GET['size']);
echo $multTable;

$sql = "INSERT INTO PageCache(ID,HTMl) VALUES(:id, :html) ON DUPLICATE KEY
UPDATE ID=:id, HTML=:html, GenTime=NOW()";
$statement = $pdo->prepare($sql);
$statement->bindValue(':id',$_GET['size']);
$statement->bindValue(':html',$multTable);
$statement->execute();
```

Test to see this works by running the page. And then seeing if a row was added to the database for that key.

5 Now you will add code to try and use the cached file before regenerating the HTML and saying the output. Write a small check to see if the database holds a version that is no older than 1minute. If so echo the contents from that cached file, otherwise regenerate

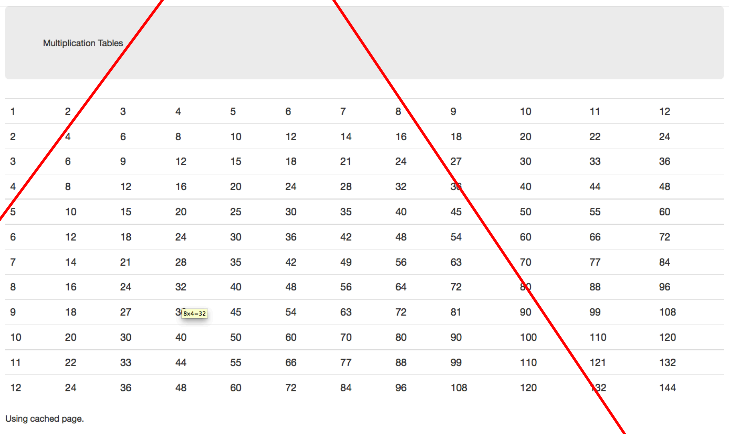
the multiplication table, output and store it for future cache as we are doing.

```
$sql = "SELECT HTML from PageCache WHERE ID=:id AND GenTime > NOW() -
INTERVAL 1 MINUTE";
$statement = $pdo->prepare($sql);
$statement->bindValue(':id',$_GET['size']);
$statement->execute();
if($statement->rowCount()>0){
  $result = $statement->fetch(PDO::FETCH_ASSOC);
  echo $result['HTML'];
  echo "<footer>Using cached page.</footer>";
}
else{

  $multTable =  getMultiplicationTable($_GET['size']);
  echo $multTable;

  $sql = "INSERT INTO PageCache(ID,HTMl) VALUES(:id, :html) ON DUPLICATE
KEY UPDATE ID=:id, HTML=:html, GenTime=NOW()";
  $statement = $pdo->prepare($sql);
  $statement->bindValue(':id',$_GET['size']);
  $statement->bindValue(':html',$multTable);
  $statement->execute();
}
```

Your page will now output the multiplication table the first time, but if you refresh it should pull from the database, and include a note that the page is cached as shown in Figure 16.4. After a minute the page should regenerate the same HTML and save it again to the database, updating the timestamp.



Multiplication Tables

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	14	16	18	20	22	24
3	6	9	12	15	18	21	24	27	30	33	36
4	8	12	16	20	24	28	32	36	40	44	48
5	10	15	20	25	30	35	40	45	50	55	60
6	12	18	24	30	36	42	48	54	60	66	72
7	14	21	28	35	42	49	56	63	70	77	84
8	16	24	32	40	48	56	64	72	80	88	96
9	18	27	3	45	54	63	72	81	90	99	108
10	20	30	40	50	60	70	80	90	100	110	120
11	22	33	44	55	66	77	88	99	110	121	132
12	24	36	48	60	72	84	96	108	120	132	144

Using cached page.

Figure 16.4 Screenshot a of a cached page.