COMP519 Web Programming

Lecture 24: PHP (Part 6)

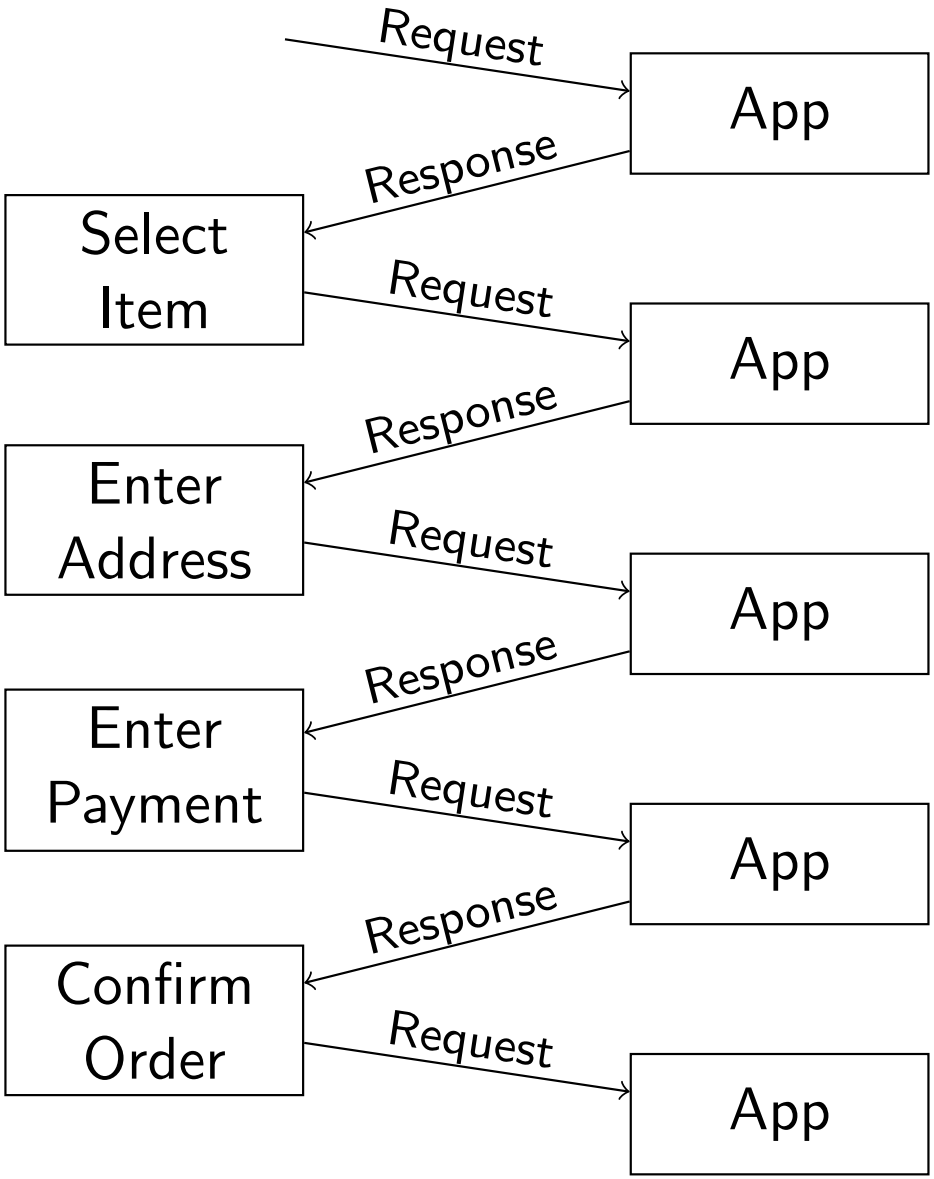
Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Web Applications Revisited



- An interaction between a user and a server-side web application often requires a sequence of requests and responses
- For each request, the application starts from scratch
  - it does not remember any data between consecutive requests
  - it does not know whether the requests come from the same user or different users

  $\rightsquigarrow$ data needs to be transferred from one execution of the application to the next

Transfer of Data: Example

- Assume the user completes a sequence of forms
- By default, a PHP script only has access to the information entered into the last form

form1.php

```
<form action="form2.php" method="post">
  <label>Item: <input type="text" name="item"></label>
</form>
```

form2.php

```
<form action="process.php" method="post">
  <label>Address: <input type="text" name="address"></label>
</form>
```

process.php

```
<?php
 echo $_REQUEST['item'];    echo $_REQUEST['address'];
?>
```

$\rightsquigarrow$ PHP Notice: Undefined index 'item'

Transfer of Data: Hidden Inputs

- Assume for a sequence of requests we do not care whether they come from the same user and whether remembered data has been manipulated
- Then hidden inputs can be used for the transfer of data from one request / page to the next

form1.php

```
<form action="form2.php" method="post">
  <label>Item: <input type="text" name="item"></label>
</form>
```

form2.php

```
<form action="process.php" method="post">
  <label>Address: <input type="text" name="address"></label>
  <input type="hidden" name="item"
         value="<?php echo $_REQUEST['item'] ?>">
</form>
```

process.php

```
<?php
 echo $_REQUEST['item'];    echo $_REQUEST['address'];
?>
```

Sessions

- Assume for a sequence of requests we do care that they come from the same user and that remembered data has not been manipulated
- Sessions help to solve this problem by associating client requests with a specific user and maintaining data over a sequence of requests from that user
- Sessions are often linked to user authentication but are independent of it, for example, eCommerce websites maintain a 'shopping basket' without requiring user authentication first

  However, sessions are the mechanism that is typically used to allow or deny access to web pages based on a user having been authenticated

Sessions

- Servers keep track of a user's sessions by using a session identifier, which
  - is generated by the server when a session starts
  - is remembered by the browser
  - is then send by the browser with every further HTTP request to that server
  - is forgotten by the browser when the session ends or the browser is closed
- In addition, the server can use session variables for storing information that relate to a session (session data), for example, the items of an order
- Sessions variables only store information temporarily

  If one needs to preserve information between visits by the same user, one needs to consider a method such as using a persistent cookie or a database to store such information

Cookies

Browser ⟶ Server

```
GET /index.html HTTP/1.1
Host:  intranet.csc.liv.ac.uk
```

Browser ⟵ Server

```
HTTP/1.0 200 OK
Content-type:  text/html
Set-Cookie:  name1=value1
Set-Cookie:  name2=value2; Expires= Thu, 20 Mar 2014, 14:00 GMT
(content of index.html)
```

Browser ⟶ Server

```
GET /teaching.html HTTP/1.1
Host:  intranet.csc.liv.ac.uk
Cookie:  name1=value1; name2=value2
Accept:  */*
```

Browser ⟵ Server

```
HTTP/1.0 200 OK
Content-type:  text/html
Set-Cookie:  name1=value3
Set-Cookie:  name2=value4; Expires= Fri, 21 Mar 2014, 14:00 GMT
Set-Cookie:  name3=value5; Expires= Fri, 28 Mar 2014, 20:00 GMT
(content of teaching.html)
```

Wikipedia Contributors: HTTP Cookie. Wikipedia, The Free Encyclopedia, 5 March 2014 20:50. `http://en.wikipedia.org/wiki/HTTP_cookie` [accessed 6 Mar 2014]

PHP Sessions

Sesssions proceed as follows

1. Start a PHP session
   - `bool session_start()`
   - `string session_id([id])`
   - `bool session_regenerate_id([delete_old])`
2. Maintain session data
   - `bool session_start()`
   - `$_SESSION` array
   - `bool isset($_SESSION[key])`
   - (interacting with a database)
3. End a PHP session
   - `bool session_destroy()`
   - `void session_unset()`
   - `bool setcookie(name, value, expires, path)`

Start a Session

- `bool session_start()`
  - creates a session
  - creates a session identifier (session id) when a session is created
  - sets up `$_SESSION` array that stores session variables and session data
  - the function must be executed before any other header calls or output is produced
- `string session_id([id])`
  - 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
- `string session_name([name])`
  - returns the name of the current session
  - if a name is given, the current session name will be replaced with the given one and the old name returned

Start a PHP Session

- `bool session_regenerate_id([delete_old])`
  - replaces the current session id with a new one
  - by default keeps the current session information stored in `$_SESSION`
  - if the optional boolean agument is `TRUE`, then the current session information is deleted

  ⇝ regular use of this function alleviates the risk of a session being 'hijacked'

```
<?php
 session_start();
 echo "Session id: ", session_id(), "<br>";
 echo "Session name: ",session_name(),"<br>";

 session_regenerate_id();
 echo "Session id: ", session_id(), "<br>";     // changed
 echo "Session name: ",session_name(),"<br>";   // unchanged
?>
```

Maintain Session Data

- `bool session_start()`
  - resumes the current session based on a session identifier passed via a GET or POST request, or passed via a cookie
  - restores session variables and session data into `$_SESSION`
  - the function must be executed before any other header calls or output is produced
- `$_SESSION` array
  - an associative array containing session variables and session data
  - you are responsible for choosing keys (session variables) and maintaining the associated values (session data)
- `bool isset($_SESSION[key])`

  returns TRUE iff `$_SESSION[key]` has already been assigned a value

Maintain Session Data

- bool session_start()
- $_SESSION array
- bool isset($_SESSION[key])

```
<?php
// Counting the number of page requests in a session
// Each web page contains the following PHP code
session_start();
if (!isset($_SESSION['requests']))
   $_SESSION['requests'] = 1;
else
   $_SESSION['requests']++;
echo "#Requests in this session so far: ",
     $_SESSION['requests'],"<br />\n";
?>
```

End a PHP Session

- `bool session_destroy()`
  - destroys all of the data associated with the current session
  - it does not unset any of the global variables associated with the session, or unset the session cookie
- `void session_unset()`
  - frees all session variables currently registered
- `bool setcookie(name, value, expires, path)`
  - defines a cookie to be sent along with the rest of the HTTP headers
  - must be sent before any output from the script
  - the first argument is the name of the cookie
  - the second argument is the value of the cookie
  - the third argument is time the cookie expires (as a Unix timestamp), and
  - the fourth argument is the parth on the server in which the cookie will be available

End a PHP Session

- `bool session_destroy()`
  - destroys all of the data associated with the current session
- `void session_unset()`
  - frees all session variables currently registered
- `bool setcookie(name, value, expires, path)`
  - defines a cookie to be sent along with the rest of the HTTP headers

```
<?php
session_start();
session_unset();
if (session_id() != "" || isset($_COOKIE[session_name()]))
  // force the cookie to expire
  setcookie(session_name(),session_id(),time()-2592000,'/');
session_destroy();
?>
```

Note: Closing your web browser will also end a session

Transfer of Data: Sessions (Part 1)

- Assume for a sequence of requests we do care whether they come from the same user or different users

form1Session.php (no changes)

```
<form action="form2Session.php" method="post">
  <label>Item: <input type="text" name="item"></label>
</form>
```

Starting/maintaining a session for the first form is optional

Transfer of Data: Sessions (Part 2)

- Assume for a sequence of requests we do care whether they come from the same user or different users

form2Session.php

```
<?php
session_start();
if (isset($_REQUEST['item']))
  $_SESSION['name'] = $_REQUEST['item'];
?>
<!DOCTYPE html>
<html lang='en-GB'>
  <head><title>Form 2</title></head>
  <body>
    <form action="processSession.php" method="post">
      <label>Address: <input type="text" name="address">
      </label>
      <!-- no hidden input required -->
    </form>
  </body>
</html>
```

Transfer of Data: Sessions (Part 3)

- Assume for a sequence of requests we do care whether they come from the same user or different users

processSession.php

```
<?php
session_start();
// not necessary but convenient
if (isset($_REQUEST['address']))
  $_SESSION['address'] = $_REQUEST['address'];
?>
<!DOCTYPE html>
<html lang='en-GB'>
  <head><title>Processing</title></head>
  <body>
<?php
  echo $_SESSION['item'];    echo $_SESSION['address'];
  // Once we do not need the data anymore, get rid of it
  session_unset();  session_destroy();
?>
  </body></html>
```

More on Session Management

The following code tracks whether a session is active and ends the session if there has been no activity for more then 30 minutes

```
if (isset($_SESSION['LAST_ACTIVITY']) &&
   (time() - $_SESSION['LAST_ACTIVITY'] > 1800)) {
   // last request was more than 30 minates ago
   session_destroy(); // destroy session data in storage
   session_unset();   // unset session variables
   if (session_id() != "" || isset($_COOKIE[session_name()]))
      setcookie(session_name(),session_id(),time()-2592000,'/');
} else {
   // update last activity time stamp
   $_SESSION['LAST_ACTIVITY'] = time();
}
```

The following code generates a new session identifier every 30 minutes

```
if (!isset($_SESSION['CREATED'])) {
   $_SESSION['CREATED'] = time();
} else if (time() - $_SESSION['CREATED'] > 1800) {
          // session started more than 30 minates ago
          session_regenerate_id(true);
          $_SESSION['CREATED'] = time();
}
```

http://stackoverflow.com/questions/520237/how-do-i-expire-a-php-session-after-30-minutes

PHP Sessions: Example

mylibrary.php:

```
<?php
session_start();

function destroy_session_and_data() {
 session_unset();
 if (session_id() != "" || isset($_COOKIE[session_name()]))
   setcookie(session_name(),session_id(),time()-2592000,'/');
 session_destroy();
}

function count_requests() {
 if (!isset($_SESSION['requests']))
    $_SESSION['requests'] = 1;
 else $_SESSION['requests']++;
 return $_SESSION['requests'];
}
?>
```

PHP Sessions: Example

page1.php:

```
<?php
 require_once 'mylibrary.php';
 echo "<html lang=\"en-GB\"><head></head><body>\n";
 echo "Hello visitor!<br />This is your page request no ";
 echo count_requests()." from this site.<br />\n";
 echo '<a href="page1.php">Continue</a> |
       <a href="finish.php">Finish</a></body>';
?>
```

finish.php:

```
<?php
 require_once 'mylibrary.php';
 destroy_session_and_data();
 echo "<html lang=\"en-GB\"><head></head><body>\n";
 echo "Goodbye visitor!<br />\n";
 echo '<a href="page1.php">Start again</a></body>';
?>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/page1.php

PHP and Cookies

Cookies can survive a session and transfer information from one session to the next

cmylibrary.php:

```
<?php
session_start();
function destroy_session_and_data() { // unchanged }

function count_requests() {
  if (!isset($_COOKIE['requests'])) {
    setcookie('requests', 1, time()+31536000, '/');
    return 1;
  } else {
    // $_COOKIE['requests']++ would not survive, instead use
    setcookie('requests', $_COOKIE['requests']+1,
              time()+31536000, '/'); // valid for 1 year
    return $_COOKIE['requests']+1;
} }
?>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/cpage1.php

PHP Sessions and Authentication

- Sessions are the mechanism that is typically used to allow or deny access to web pages based on a user having been authenticated
- Outline solution:
  - We want to protect a page content.php from unauthorised use
  - Before being allowed to access content.php, users must first authenticate themselves by providing a username and password on the page login.php
  - The system maintains a list of valid usernames and passwords in a database and checks usernames and passwords entered by the user against that database
    If the check succeeds, a session variable is set
  - The page content.php checks whether this session variable is set
    If the session variable is set, the user will see the content of the page
    If the session variable is not set, the user is redirected to login.php
  - The system also provides a logout.php page to allow the user to log out again

PHP Sessions and Authentication: Example

content.php:

```
<?php
session_start();
if (!isset($_SESSION['user'])) {
    // User is not logged in, redirecting to login page
    header('Location:login.php');
}
?>
<!DOCTYPE html>
<html lang="en-GB">
<head><title>Content that requires login</title></head>
<body>
<h1>Protected Content</h1>
<b>Welcome <i><?php echo $_SESSION['user'] ?></i></b><br />
<b><a href="logout.php">Log Out</a></b>
</body>
</html>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/content.php

PHP Sessions and Authentication: Example

Second part of login.php:

```
<!DOCTYPE html>
<html lang="en-GB">
<head><title>Login</title></head>
<body>
 <h1>Login</h1>
 <form action="" method="post">
  <label>Username:
  <input name="user" placeholder="username" type="text">
  </label>
  <label>
  Password:
  <input name="passwd" placeholder="**" type="password">
  </label>
  <input name="submit" type="submit" value="login ">
  <span><?php echo $error; ?></span>
 </form>
</body>
</html>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/login.php

PHP Sessions and Authentication: Example

First part of login.php:

```
<?php
session_start();

function checkCredentials($user,$passwd) { // Authenticate the user
}
function nextLoc() {                       // Compute next location
}

$error='';
if (isset($_POST['submit'])) {
   if (checkCredentials($_REQUEST['user'],$_REQUEST['passwd'])) {
      $_SESSION['user']=$_REQUEST['user'];
      header("location:".nextLoc()); // Redirecting to content
   } else {
      $error = "Username or Password is invalid. Try Again";
   }
}
if (isset($_SESSION['user'])){
   header("location:".nextLoc());
}
?>
```

PHP Sessions and Authentication: Example

nextLoc():

```
function nextLoc() {
  // Works out where to send the user after they have been authenticated
  if ((basename($_SERVER['HTTP_REFERER']) == 'login.php')  ||
      (basename($_SERVER['HTTP_REFERER']) == 'logout.php')) {
    // If the user came from the login or logout page,
    // send the user to the `default' page.
    return "content.php";
  } else {
    // Otherwise, send the user to where they came from.
    return $_SERVER['HTTP_REFERER'];
  }
}
```

PHP Sessions and Authentication: Example

logout.php:

```
<?php
session_start();
$user = $_SESSION['user'];
session_unset();
session_destroy();
?>
<!DOCTYPE html>
<html lang="en-GB">
<head>
<title>Logout</title>
</head>
<body>
<h1>Logout</h1>
<b>Goodbye <i><?php echo $user ?></i></b><br />
<b><a href="login.php">Login</a></b>
</form>
</body>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/logout.php

Revision and Further Reading

Read

- Chapter 12: Cookies, Sessions, and Authentication

of R. Nixon: Learning PHP, MySQL & JavaScript: with jQuery, CSS & HTML5. O'Reilly, 2018.