COMP284 Scripting Languages

Lecture 12: PHP (Part 4)

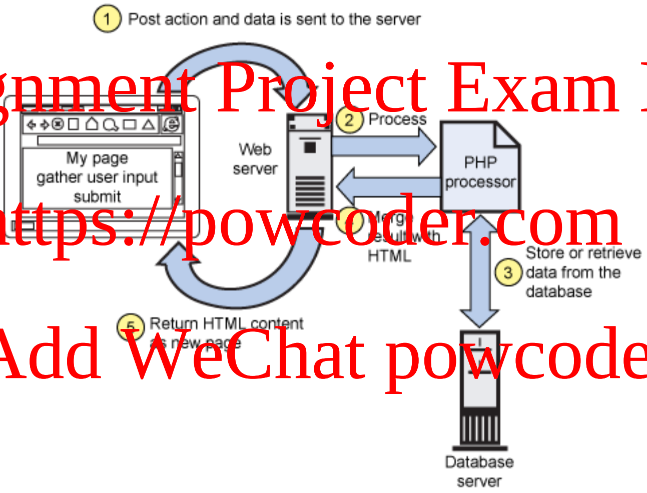
Handouts

Ullrich Hustadt

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

Contents

Web applications using PHP


1 Post action and data is sent to the server
My page
gather user input
submit
Web server
2 Process
PHP processor
4 Merge results with HTML
3 Store or retrieve data from the database
5 Return HTML content as new page
Database server

IBM: Build Ajax-based Web sites with PHP, 2 Sep 2008.
`https://www.ibm.com/developerworks/library/wa-aj-php/` [accessed 6 Mar 2013]

HTML forms

When considering Perl CGI programming we have used HTML forms that generated a client request that was handled by a Perl CGI program:

```
<form action=
 "http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/ullrich/demo"
 method="post">
...
</form>
```

Now we will use a PHP script instead:

```
<form action="http://cgi.csc.liv.ac.uk/~ullrich/demo.php"
 method="post">
...
</form>
```

- The PHP script file must be stored in a directory accessible by the web server, for example `$HOME/public_html`, and be readable by the web server
- The PHP script file name must have the extension `.php`, e.g. `demo.php`

Information available to PHP scripts

- Information about the PHP environment
- Information about the web server and client request
- Information stored in files and datbases
- Form data
- Cookie/Session data
- Miscellaneous
  - `string date(`*format*`)`
    returns the current date/time presented according to *format*
    for example `date('H:i l, j F Y')`
    results in `12:20 Thursday, 8 March 2012`
    (See `http://www.php.net/manual/en/function.date.php`)
  - `int time()`
    returns the current time measured in the number of seconds
    since January 1 1970 00:00:00 GMT

PHP environment

- phpinfo() displays information about the PHP installation and EGPCS data (Environment, GET, POST, Cookie, and Server data) for the current client request
- phpinfo(*part*) displays selected information

```
<html><head></head><body>
<?php
  phpinfo();                   // Show all information
  phpinfo(INFO_VARIABLES);     // Show only info on EGPCS data
?>
</body></html>
```

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

INFO_GENERAL	The configuration, php.ini location, build date, web server
INFO_CONFIGURATION	Local and master values for PHP directives
INFO_MODULES	Loaded modules
INFO_VARIABLES	All EGPCS data

Manipulating the PHP configuration

The following functions can be used to access and change the configuation of PHP from within a PHP script:

- `array ini_get_all()`
  - returns all the registered configuration options

- `string ini_get(option)`
  - returns the value of the configuration option on success
- `string ini_set(option, value)`
  - sets the value of the given configuration option to a new value
  - the configuration option will keep this new value during the script's execution and will be restored afterwards

- `void ini_restore(option)`
  - restores a given configuration option to its original value

Server variables

The $_SERVER array stores information about the web server and the client request

⤳ Similar to %ENV for Perl CGI programs

```
<html><head></head><body>
<?php
echo 'Server software: ',$_SERVER['SERVER_SOFTWARE'],'<br />';
echo 'Remote address: ',$_SERVER['REMOTE_ADDR'],'<br />';
echo 'Client browser: ',$_SERVER['HTTP_USER_AGENT'],'<br />';
echo 'Request method: ',$_SERVER['REQUEST_METHOD'];
?></body></html>
```

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

```
Server software: Apache/2.2.22 (Fedora)
Remote address: 10.128.0.215
Client browser: Mozilla/5.0 ... Chrome/41.0.2272.53 ...
Request method:
```

See http://php.net/manual/en/reserved.variables.server.php for a list of keys

Form data

- Form data is passed to a PHP script via the three arrays:

`$_POST`	Data from POST client requests
`$_GET`	Data from GET client requests
`$_REQUEST`	Combined data from POST and GET client requests (derived from `$_POST` and `$_GET`)

⇝ Accessing `$_REQUEST` is the equivalent in PHP to using the `param` routine in Perl

```
<form action="process.php" method="post">
<label>Enter your user name:
      <input type="text" name="username"></label><br>
<label>Enter your full name:
      <input type="text" name="fullname"></label><br>
<input type="submit" value="Click␣for␣response"></form>
```

`$_REQUEST['username']`	Value entered into field with name 'username'
`$_REQUEST['fullname']`	Value entered into field with name 'fullname'

Forms in PHP: Example (1)

- Create a web-based system that asks the user to enter the URL of a file containing bibliographic information
- Bibliographic information will have the following form:

```
@entry{
 name={Jonas Lehner},
 name={Andreas Schoknecht},
 title={<strong>You only live twice</strong>},
}
@entry{
 name={Andreas Schoknecht},
 name={Eva Eggeling},
 title={No End in Sight?}
}
```

- The system should extract the names, count them, and create a table of names and their frequency, ordered from most frequent to least frequent

Forms in PHP: Example (1)

extract_names.php

```
<!DOCTYPE html>
<html><head><title>Name Extraction</title></head><body>
<?php
 require_once 'extraction.php';
 if (isset($_SERVER['REQUEST_METHOD']) &&
     $_SERVER['REQUEST_METHOD'] == 'POST' &&
     isset($_REQUEST['url'])) {
   $extracted_names = extract_names($_REQUEST['url']);
   echo "<p>The names occurring in <br>", htmlspecialchars($_REQUEST['url']),
        "<br>are</p>$extracted_names\n";
 } else {
   echo <<<FORM
   <form method="post">
     <label>Enter a URL:
       <input type="text" name="url" size="100"
        value="http://cgi.csc.liv.ac.uk/~ullrich/COMP284/tests/a1test1.txt">
     </label><br><br>
     <input type="submit" value="Extract Names">
   </form>
FORM;
 }
?>
</body></html>
```

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

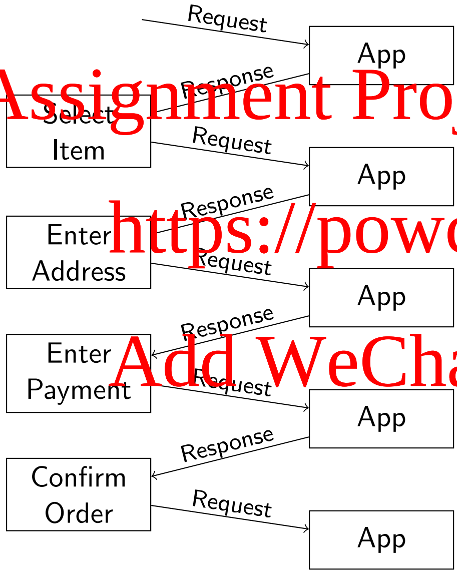
Forms in PHP: Example (1)

extraction.php

```
<?php
function extract_names($url) {
 $text = file_get_contents($url);
 if ($text === false)
   return "ERROR: INVALID URL!";
 else {
   $correct = preg_match_all("/name={([^\}]+)}/",
              $text,$matches,PREG_PATTERN_ORDER);
   if ($correct == 0) return "ERROR: NO NAMES FOUND";
   $count = array_count_values($matches[1]);
   arsort($count);
   foreach ($count as $name => $number) {
     $table .= "<tr><td>$name</td><td>$number</td></tr>";
   }
   $table = "<table><thead><tr><th>Name</th><th>No of occur".
   "rences</th></tr></thead><tbody>".$table."</tbody></table>";
   return $table;
} }
?>
```

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

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 maintain a state 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 for a sequence of requests we do not care whether they come from the same user or different users
- 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>Name: <input type="text" name="name"></label>
</form>
```

form2.php

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

process.php

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

Sessions

- By default, HTML and web servers do not keep track whether several client requests come from the same user or different users

  Thus, a process that spans several pages, for example, placing an order, requires additional mechanisms

- Sessions help solve this problem by associating client requests with specific users and maintaining data during a user's visit

- Sessions are often linked to user authentication but session can be used without user authentication, 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 and
  - is then used by the browser when the user requests a page from the server

  The session identifier can be sent through a cookie or by passing the session identifier in client requests
- In addition, one can use session variables for storing information to relate to a user and her session (session data), for example, the items of an order
- Sessions 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 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

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() - $_SESSION['LAST_ACTIVITY'] > 1800)) {
   // last request was more than 30 minutes ago
   session_destroy(); // destroy session data in storage
   session_unset();   // unset session variables
   if (session_id() != "" || isset($_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><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><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

Second part of login.php:

```
<!DOCTYPE html>
<html>
<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/COMP284/examples/login.php

PHP Sessions and Authentication: Example

First part of login.php:

```
<?php
session_start();

function checkCredentials($user,$passwd) {
  // Check whether $user and $passwd are non-empty
  // and match an entry in the database
}

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

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>
<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/COMP284/examples/content.php

PHP Sessions and Authentication: Example

logout.php:

```
<?php
session_start();
$user = $_SESSION['user'];
session_unset();
session_destroy();
?>
<!DOCTYPE html>
<html>
<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/COMP284/examples/logout.php

Revision

Read

- Chapter 10: Accessing MySQL Using PHP
- Chapter 11: Form Handling
- Chapter 13: Cookies, Sessions, and Authentication

of

R. Nixon:
Learning PHP, MySQL, and JavaScript.
O'Reilly, 2009.