IERG4210 Web Programming and Security

Forms II - Server-side Implementation

Adonis P.H. FUNG
Dept. of Information Engineering
The Chinese University of Hong Kong
phfung@ie.cuhk.edu.hk

Agenda

- *Client-side* Implementations of Forms (covered in lecture 4)
  - Input Controls -> Validations -> Form Submissions
- *Server-side* Implementations of Forms
  - Recall the Request Methods: GET v.s. POST
  - Server-side Language: PHP
  - Form/Request Handling with PHP:
    - Input - Sanitizations and Validations
    - Process - DB Manipulation
    - Output - HTML v.s. JSON
- Sample Code of Phase 2B - released at CUHK Blackboard

HTTP Request Method: GET v.s. POST

- No matter how grand the client-side is, a server will receive:
- GET Request, or

```
GET /index.php?catid=3 HTTP/1.1
Host: www.shop.ierg4210.org
```

  Parameters are appended as query string at the URL

- POST Request

```
POST /admin-process.php HTTP/1.1
Host: secure.shop.ierg4210.org
Content-Length: 37
Content-Type: application/x-www-form-urlencoded

name=Fresh%20Fruits&action=cat_insert
```

  Parameters are encoded as the request body with 2 additional request headers

Server-side Web Programming Languages

- To pick a server-side language for this course:
  -

Languages	Market Share of Top 1M Most Popular Websites
PHP	78.1%
ASP.NET	20.9%
Java	4.0%
ColdFusion	1.1%
Perl	0.9%
Ruby	0.5%
Python	0.2%

  - PHP is most sought by the job market (JobsDB)
  - Final reason: *I know PHP better :)*

Ref: W3Techs.com, retrieved on October 15, 2012

PHP Basics (1/2)

- PHP is a Server-side Scripting Language
  - Create a file that ends with .php, e.g., test.php
  - Insert PHP code anywhere, e.g. `Today is <?php echo date(); ?>`
  - When the file test.php is visited thru a browser,
    a server can then run it without any precompilation (as demonstrated in Tuto 3)
- C-like syntax with a few syntactic differences:
  - All variables start with the `$` sign, e.g. `$data`, `$array`
  - No need to declare a variable before use
- Dynamic Typing Variables ($a = 1; $a = 'hello';)
- Block-level Scoping for variables (like C but unlike JS)

PHP Basics (2/2)

- Code hidden from client-side; show only processed output
  - Given a helloWorld.php with its content as follows:

```
<h1><?php echo "Hello World"; ?></h1>
```

  - Only the following is visible to the browser when visiting helloWorld.php:

```
<h1>Hello World</h1>
```

  - Hence, dynamic HTML outputs can be mixed with static HTML
- Security: Prevent OWASP A2-Cross-Site Scripting
  - DON'T trust users' input, apply **context-dependent** output sanitizations:

```
<!-- Consider when $name = 'John<script>alert(1)</script>', -->
<h1>Good morning, <?php echo htmlspecialchars($name); ?>.</h1>
```

  Note: htmlspecialchars() escapes < to `&lt;` and > to `&gt;`, etc.
  - AVOID writing JavaScript with PHP as we lack a good santization function!

```
//Improper sanitizations could cause XSS
~~<script>var amount = <?php echo $amount; ?></script>~~
```

PHP String Processing

- Difference between " and ' when quoting a string

PHP code	Output
`echo "Hello\nWorld";`	Hello World
`echo 'Hello\nWorld';`	Hello\nWorld

References: Single-quoted and Double-quoted Strings

- String Concatenation - joined by a dot (v.s. + in JS)

```
<ul><?php   $name = "Apple";    echo "<li>" . $name . "</li>";    ?></ul>
```

- Some Useful Functions

```
<?php
  strlen("hello") == 5            // true
  strpos("hello", "l") == 2       // true
  $a = ''; empty(a)               // true
?>
```

PHP Arrays (1/2)

- Numeric Array (similar to JS array [])

```
$fruits = array("apple", "orange", "pineapple");
```

- Associative Array (similar to JS object {})

```
$ages = array("Niki" => 6, "John" => 30, "Stephen" => 40);
```

- To add/edit an element (dynamic-sized)

```
$fruits[] = "banana";     // yes, a new element is created!! :)
$fruits[1] = "orange2";   // changed orange to orange2
$ages["Peter"] = 10;      // added a new element
$ages["Niki"]++;          // Niki enjoyed her birthday party
```

- To remove an element

```
unset($fruits[1]);        // orange2 is deleted
unset($ages["Stephen"])   // Stephen has rested in peace
```

PHP Arrays (2/2)

- Looping over numeric array

```
for ($i = 0, $len = count($fruits); $i < $len; $i++)
  /* do something with $fruits[$i] */
```

- Looping over associative array

```
foreach ($ages as $key => $val)
  /* do something with $key and $val */
```

- Other Array Functions:
  - array_push() and array_pop() - Using numerica array as a stack
  - implode() - Join array elements with a string (similar to String.join() in JS)
  - explode() - Split a string by string (similar to String.split() in JS)
  - array_map() - Applies the callback to the elements of the given arrays
  - sort() - Sort an array
  - array_diff() - Computes the difference of arrays
  - etc...

Reference: http://php.net/manual/en/control-structures.foreach.php

PHP Functions

- Simple Example

```
// Example Call: hello()
function hello() {
  echo "Hello!";
}
```

- Accepting Function Parameters

```
// Example Call: hello('Niki')
function hello($name) {
  echo "Hello, " . htmlspecialchars($name) . "!";
}
```

Similar to our escapeHTML() in JS, htmlspecialchars() is to sanitize output

- Specifying Default Function Parameters

```
//Example Call: hello('Niki') or hello('Niki', 'F')
function hello($name, $sex = 'M') {}
//Example Call: hello2('Niki') or hello2('Niki', 'F') or hello2('Niki', 'F', 30000)
function hello2($name, $sex = 'M', $income = 10000){}
```

Parameters with default values must be right-aligned

Best Practice: To Include an External File

- Your assignment has main page and product description page, some HTML are actually **shared** among both pages
- Best Practice: Host the common part in a file and load it dynamically across multiple pages to facilitate code reuse
  - Without PHP execution

```
<?php readfile('html/header.html'); ?>
<h1>Product Description:</h1>
<!-- Description goes here -->
<?php readfile('html/footer.html'); ?>
```

  - With PHP execution - good for including PHP libraries

```
<?php include_once('lib/myLib.php'); ?>
```

  - Note: `readfile()` is faster than `include_once()` as no parsing is needed to look for PHP

Form/Request Handling with PHP

- Given an example of HTTP request:

```
POST /admin-process.php?action=cat_insert HTTP/1.1
Host: secure.shop.ierg4210.org
Content-Length: 19
Content-Type: application/x-www-form-urlencoded

name=Fresh%20Fruits
```

- Input parameters are stored in some superglobals arrays:

```
$_POST['name'] == 'Fresh Fruits'    // true; Values are auto-urldecoded, '%20' -> ' '
$_GET['action'] == 'cat_insert'     // true
$_REQUEST['action'] == 'cat_insert' // true
```

Note: $_REQUEST combines $_GET, $_POST and $_COOKIE (default order)

- Finally, a design pattern: Validate before further processing

```
<?php
if ($_REQUEST['action'] == 'cat_insert') {
  inputValidate($_POST['name'], '/^[\w\- ]+$');  // See next slide for details
  DB_insertCategory($_POST['name']);             // DB Manipulation with SQL
}
?>
```

Input - Validation Flaws

- Severity of the problem
  - Ranked High in 2007, 2010 by OWASP Top 10 Application Security Risks
  - In 2010, input validation flaws are ranked: A1 Injection, A2 Cross-site Scripting, A4 Insecure Direct Object References (details in later lecture)
- Root cause: Unexpected users' inputs could lead to the execution of unauthorized actions
- Fundamental Defences: Restrict users' inputs
  - Input Validations - rejecting invalid inputs
    - most effective - whitelisting acceptable data
    - may be insecure - blacklisting malicious characters (hard to exhaust all; can you blacklist unknown exploit?)
  - Input Sanitizations - transforming invalid inputs to be safe
    - Type casting: `parseInt(input='666')` for JS; `$a = (int)$a;` for PHP
    - Escape characters (context-dependent): prevent SQL injection (to be covered)

Input - Server-side and Client-side Validations

- To reiterate once again, apply validations at:
  - *server-side* for security enforcement
  - *client-side* for user experience enhancement

  Reason: Unlike client code that can freely manipulated at browsers, server code and logic is hidden from clients (thus cannot be easily bypassed)
- Security Best Practice: Therefore, maintain both server- and client-side validation code as consistent as possible!!

```
<?php
// Validate an input using the same regular expression as done in JS
if (preg_match('/^[\w=+\-\/][\w=\'+\-\/\.]*@[\w\-]+(\.[\w\-]+)*(\.[\w]{2,6})$/',
         $_POST['email'])) {
  /* TODO: Only validated inputs are used for further processing, e.g. database */
} else {
  /* reject the input */
  exit();
}
?>
```

Process - Database Management

- SQL Languages covered in Tutorial 4
- DB Manipulations w/PDO - examples from sample code

```
function ierg4210_cat_fetchall() {
  // DB manipulation
  global $db;
  $db = ierg4210_DB();
  $q = $db->prepare("SELECT * FROM categories LIMIT 100;");
  if ($q->execute())
    return $q->fetchAll();  // i.e. an array of categories
}
```

```
function ierg4210_cat_insert() {
  // input validation or sanitization
  if (!preg_match('/^[\w\-, ]+$/', $_POST['name']))
    throw new Exception("invalid-name");
  // DB manipulation
  global $db;
  $db = ierg4210_DB();
  $q = $db->prepare("INSERT INTO categories (name) VALUES (?)");
  return $q->execute(array($_POST['name']));  // i.e. T/F - whether it is success
}
```

Note: Prepared statement is to prevent SQL injections (details later)

Process - Design Pattern of Form Handlers

- Maintain a Single Entrance for Form Handlers
  - HTML: All forms send HTTP requests to `admin-process.php`, and associate an unique *action name* as hidden parameter w/each form
  - PHP: In the centralized entrance `admin-process.php`, routes HTTP requests to a corresponding function based on *action name*
- As a result, here's a simplified version of `admin-process.php` from sample code:

```
function ierg4210_cat_fetchall() {/* return an array of categories */}
function ierg4210_cat_insert()   {/* return true or false to indicate success */}
```

```
if (!empty($_REQUEST['action'])) {
  header('Content-Type: application/json');  // JSON to be discussed in next slide
  try {
    // To call the corresponding function based on action name
    if (($returnVal = call_user_func('ierg4210_' . $_REQUEST['action'])) === false)
      echo json_encode(array('failed'=>true));
    else
      echo 'while(1);'.json_encode(array('success' => $returnVal));
  }
  catch(Exception $e) {
    echo 'while(1);'.json_encode(array('failed' => $e->getMessage()));
  }
} else  echo json_encode(array('failed'=>'undefined'));
```

Output - HTML v.s. JSON (1/2)

- Traditionally, HTML output is returned after processing

```
<?php
readfile('html/header.html');
for ($categories=ierg4210_cat_fetchall(), $i=0, $cat; $cat = $categories[$i]; $i++) {
  /* Re-populate the HTML with $cat['catid'] and $cat['name'] */ }
if (ierg4210_cat_insert()) echo '<h2>The category is created successfully.</h2>';
/* Reproduce other HTML snippets here, e.g. forms */
readfile('html/footer.html');     ?>
```

A HTML page let users submit its forms. After each form handling, a browser has to re-download the same HTML page that differs only by a successful notice.

- Nowadays, we use JavaScript Object Notation (JSON) format
  - Encode the output of `ierg4210_cat_fetch_all()` will give:

```
<?php
function ierg4210_cat_fetchall() {/* return an array of categories */}
function ierg4210_cat_insert()   {/* return true or false to indicate success */}
header('Content-Type: application/json');
if (($returnVal = call_user_func('ierg4210_' . $_REQUEST['action'])) === false)
      echo json_encode(array('success' => $returnVal));     ?>
```

```
{"success":[{"catid":"1","name":"Fruits"},{"catid":"2","name":"Candies"}]}
```

Output - HTML v.s. JSON (2/2)

- To decode the JSON output at client-side:
  - Given the JSON result produced by `json_encode()` in PHP

```
{"success":[{"catid":"1","name":"Fruits"},{"catid":"2","name":"Candies"}]}
```

  - Decode the output by `JSON.parse()` in JavaScript

```
<script type="text/javascript">
myLib.ajax({url:'admin-process.php?action=cat_fetchall',success:function(output){
  // to decode the xhr.responseText and turns it to an object
  var json = JSON.parse(output);
  if (json.success) {
    // to print out each record with proper output sanitizations
    for (var i = 0, record; record = json.success[i]; i++) {
      somewhere.innerHTML += 'CatId: ' + parseInt(record.catid) + '<br/>'
                          + 'Name: ' + record.name.escapeHTML();
    }
  } else alert('Error!');
}});
</script>
```

Advantages: 1/Minimize bandwidth needed since no redundant download
2/JSON parsing is stunning fast as the format itself is JS!!
3/Loose coupling: PHP - data-intensive processing; JS - UI handling

Reference: http://www.json.org/js.html

Some Logistics...

- No lecture next week; (Holiday: Chung Yeung Festival)
- Next-lecture Forecast (Oct 30):
  Cookie, Session and Storage Management

- Deadline for Assignment Phase 2B: Oct 23, 2012, 5PM
- Deadline for Quiz 2: Oct 29, 2012, 5PM