SERVER-SIDE WEB PROGRAMMING UNIT2: PROGRAMMING BASED ON EMBEDDED LANGUAGE

CFGS - DAW 2

JESUITAS
Sagrado Corazón
Jesuitas · Logroño

Index

- Web Techniques
  - Variables
  - Processing Forms
  - Form Processing with Functions
  - Validating data
  - HTML and JavaScript
  - Displaying Default Values

5. Web Techniques

- The two most common HTTP methods are GET and POST:
  - The ***GET method*** is designed for retrieving information, such as a document, an image, or the results of a database query, from the server.
    - The GET method is what a web browser uses when the user types in a URL or clicks on a link.
  - The ***POST method*** is meant for posting information to the web server, such as a credit card number or information that is to be stored in a database.

5.1. Variables

- Server configuration and request information (including form parameters or cookies) are accessible in different ways from your PHP scripts.
- Collectively, this information is referred to as **EGPCS** (environment, GET, POST, cookies, and server).
- *PHP creates six global arrays that contain the EGPCS information*!!!

5.1. Variables

- The **global arrays** are:
  1. *$_COOKIE*: Contains any cookie values passed as part of the request, where the keys of the array are the names of the cookies
  2. *$_GET*: Contains any parameters that are part of a GET request, where the keys of the array are the names of the form parameters
  3. *$_POST*: Contains any parameters that are part of a POST request, where the keys of the array are the names of the form parameters
  4. *$_FILES*: Contains information about any uploaded files
  5. *$_SERVER*: Contains useful information about the web server
  6. *$_ENV*: Contains the values of any environment variables, where the keys of the array are the names of the environment variables

5.1. Variables

- The *$_REQUEST* array is also created by PHP automatically. It contains the elements of the $_GET, $_POST, and $_COOKIE arrays all in one array variable. Also accesible.

5.2. Processing Forms

- It's easy to process forms with PHP, as the form parameters are available in the $_GET and $_POST arrays.
- Example:
  - http://www.example.com/catalog.php?product_id=21&category=fryingpan puts two values into $_GET:
    1. $_GET['product_id'] is set to 21
    2. $_GET['category'] is set to fryingpan

5.2. Processing Forms

- GET requests: The HTTP specification says they *are idempotent* → one GET request for a particular URL, including form parameters, is the same as two or more requests for that URL.
  - That is why, web browsers can cache the response pages for GET requests because the response page doesn't change regardless of how many times the page is loaded.
- POST requests: they cannot be cached, and the server is re-contacted every time the page is displayed. ("Repost form data?").

5.2. Processing Forms

- Browser caches are generally so poorly implemented, and the Reload button is so easy to hit, that programmers tend to use GET and POST simply based on whether they want the query parameters shown in the URL or not.
- <u>What you need to remember is that GET requests should not be used for any actions that cause a change in the server, such as placing an order or updating a database.</u>

5.2. Processing Forms


```
1   <form method="POST" action="catalog.php">
2       <input type="text" name="product_id">
3       <select name="category">
4       <option value="oven">Oven</option>
5           <option value="fryingpan">Frying Pan</option>
6           <option value="microwave">Microwave</option>
7       </select>
8       <input type="submit" name="submit">
9   </form>
10  Here are the submitted values:
11  product_id: <?php print $_POST['product_id'] ?? '' ?>
12  <br/>
13  category: <?php print $_POST['category'] ?? '' ?>
```

- ??: Null coalescing operator… new in PHP7!

  Check the offitial site.

5.2. Processing Forms


```
1   <form method="POST" action="eat.php">
2       <select name="lunch[]" multiple>
3       <option value="pork">BBQ Pork Pizza</option>
4       <option value="chicken">Chicken Pizza</option>
5       <option value="lotus">Lotus Seed Pizza</option>
6       <option value="hawaian">Hawaian Pizza</option>
7       <option value="carbonara">Carbonara Pizza</option>
8       </select>
9       <input type="submit" name="submit">
10  </form>
11  Selected pizzas:
12  <br/>
13  <?php
14  if (isset($_POST['lunch'])) {
15      foreach ($_POST['lunch'] as $choice) {
16          print "You want a $choice pizza. <br/>";
17      }
18  }
19  ?>
```

- A form element that can have multiple values needs to have a name that ends in [].

5.2. Processing Forms

- The type of method that was used to request a PHP page is available through *$_SERVER['REQUEST_METHOD']*:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    // handle a GET request
}
else {
    die("You may only GET this page.");
}
```

5.3. Form Processing with Functions

form1.php

```php
<?php
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    process_form();
} else {
    show_form();
}

// Do something when the form is submitted
function process_form() {
    print "Hello, ". $_POST['my_name'];
}

// Display the form
function show_form() {
    print<<<_HTML_
    <form method="POST" action="$_SERVER[PHP_SELF]">
        Your name: <input type="text" name="my_name">
        <br/>
        <input type="submit" value="Say Hello">
    </form>
_HTML_;
}
?>
```

5.3. Form Processing with Functions


```php
<?php
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (validate_form()) {
        process_form();
    } else {
        show_form();
    }
} else {
    show_form();
}
// Do something when the form is submitted
function process_form() {
    print "Hello, ". $_POST['my_name'];
}
// Display the form
function show_form() {
    print<<<_HTML_
    <form method="POST" action="$_SERVER[PHP_SELF]">
        Your name: <input type="text" name="my_name">
        <br/>
        <input type="submit" value="Say Hello">
    </form>
_HTML_;
}
// Check the form data
function validate_form() {
// Is my_name at least 3 characters long?
    if (strlen($_POST['my_name']) < 3) {
        return false;
    } else {
        return true;
    }
}
```

Includes validation

form2.php

5.4. Validating data

- Most of the input to your application is probably coming from a *web form*.
- But there are lots of other ways data can flow into your programs as well: *databases that you share with other people or applications, web services and remote servers, even URLs and their parameters*.

5.4. Validating data


```php
<?php
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
// If validate_form() returns errors, pass them to show_form()
    if ($form_errors = validate_form()) {
        show_form($form_errors);
    } else {
        process_form();
    }
} else {
    show_form();
}
// Do something when the form is submitted
function process_form() {
    print "Hello, ". $_POST['my_name'];
}
```

```php
// Display the form
function show_form($errors=[]) {
    // If some errors were passed in, print them out
    if ($errors) {
        print 'Please correct these errors: <ul><li>';
        print implode('</li><li>', $errors);
        print '</li></ul>';
    }
    print<<<_HTML_
    <form method="POST" action="$_SERVER[PHP_SELF]">
        Your name: <input type="text" name="my_name">
        <br/>
        <input type="submit" value="Say Hello">
    </form>
_HTML_;
}
// Check the form data
function validate_form() {
// Start with an empty array of error messages
    $errors = array();
// Add an error message if the name is too short
    /*if (strlen($_POST['my_name']) < 3) {
        $errors[ ] = 'Your name must be at least 3 letters long.';
    }*/
    $ok = filter_input(INPUT_POST, 'my_name', FILTER_VALIDATE_INT);
    if (is_null($ok) || ($ok === false)) {
        $errors[] = 'Please enter a valid age.';
    }
// Return the (possibly empty) array of error messages
    return $errors;
}
```

form3.php

Includes validation & Error displaying

5.4. Validating data

1. <u>Required elements:</u>
   - To make sure something has been entered into a required element, check the element's length with *strlen()*:

```
if (strlen($_POST['email']) == 0) {
    $errors[] = "You must enter an email address.";
}
```

5.4. Validating data

2. <u>Numeric Elements:</u>

- To ensure that a submitted value is an integer or floating-point number, use *filter_input()* function with an appropriate filter (FILTER_VALIDATE_INT and FILTER_VALIDATE_FLOAT).

```
$ok = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
if (is_null($ok) || ($ok === false)) {
    $errors[] = 'Please enter a valid age.';
}
```

A.2.22. Change previous *form3.php* example, in order it to include a new form field for the *age*. The script should also validate the age using the code above and print the error if needed or the age in the greeting.

5.4. Validating data

3. <u>String Elements:</u>

- Combination of trim() and strlen():

```
if (strlen(trim($_POST['name'])) == 0) {
    $errors[] = "Your name is required.";
}
```

- Another enhacement, would be to have an input array with the checked values, instead of working directly with $POST

5.4. Validating data

- Another enhacement, would be to have an input array with the checked values, instead of working directly with $POST:

```
function validate_form() {
    $errors = array();
    $input = array();
    $input['age'] = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
    if (is_null($input['age']) || ($input['age'] === false)) {
        $errors[] = 'Please enter a valid age.';
    }
    // Use the null coalesce operator in case $_POST['name'] isn't set
    $input['name'] = trim($_POST['name'] ?? '');
    if (strlen($input['name']) == 0) {
        $errors[] = "Your name is required.";
    }
    return array($errors, $input);
}
```

A.2.23. Change previous *form3.php* example, in order to use this new $input array. *Clue: It will be a good idea to use* ***list*** *method in order to take out from the return of validate_form the two arrays into two variables.*

5.4. Validating data

4. <u>Number Ranges:</u>

- use the *min_range* and *max_range* options of the *FILTER_VALIDATE_INT* filter:

```
$input['age'] = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT,
    array('options' => array('min_range' => 18,
        'max_range' => 65)));
if (is_null($input['age']) || ($input['age'] === false)) {
    $errors[] = 'Please enter a valid age between 18 and 65.';
}
```

A.2.24. Apply this new filter to our form3.php example.

5.4. Validating data

- Checking a date range (less than 6 months old):

```
// Make a DateTime object for 6 months ago
$range_start = new DateTime('6 months ago');
// Make a DateTime object for right now
$range_end = new DateTime();
// 4-digit year is in $_POST['year']
// 2-digit month is in $_POST['month']
// 2-digit day is is $_POST['day']
$input['year'] = filter_input(INPUT_POST, 'year', FILTER_VALIDATE_INT,
    array('options' => array('min_range' => 1900,
        'max_range' => 2100)));
$input['month'] = filter_input(INPUT_POST, 'month', FILTER_VALIDATE_INT,
    array('options' => array('min_range' => 1,
        'max_range' => 12)));
$input['day'] = filter_input(INPUT_POST, 'day', FILTER_VALIDATE_INT,
    array('options' => array('min_range' => 1,
        'max_range' => 31)));
// No need to use === to compare to false since 0 is not a valid
// choice for year, month, or day. checkdate() makes sure that
// the number of days is valid for the given month and year.
if ($input['year'] && input['month'] && input['day'] &&
    checkdate($input['month'], $input['day'], $input['year'])) {
    $submitted_date = new DateTime(strtotime($input['year'] . '-' .
        $input['month'] . '-' .
        $input['day']));
if (($range_start > $submitted_date) || ($range_end < $submitted_date)) {
    $errors[] = 'Please choose a date less than six months old.';
}
} else {
// This happens if someone omits one of the form parameters or submits
// something like February 31.
    $errors[] = 'Please enter a valid date.';
}
```

5.4. Validating data

5. Email Addresses:

```
$input['email'] = filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL);
if (! $input['email']) {
    $errors[] = 'Please enter a valid email address';
}
```

A.2.25. Apply this new filter to our form3.php example.

5.5. HTML and JavaScript

- Cross-site scripting attack: To prevent them in your programs, never display unmodified external input!
- In most applications, you should use htmlentities() to sanitize external input

```
$comments = htmlentities($_POST['comments']);
// Now it's OK to print $comments
print $comments;
```

A.2.26. Create a little example in order to see how it works.

5.6. Displaying Default Values

- Sometimes, you want to display a form with a value already in a text box or with preselected checkboxes, radio buttons, or <select> menu items.
- Additionally, when you redisplay a form because of an error, it is helpful to preserve any information that a user has already entered (*sticky form*).

5.6. Displaying Default Values

- Create a new php file and follow these steps:

1. Setting a default value in a <select> menu:

```php
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $defaults = $_POST;
} else {
    $defaults = array('delivery' => 'yes',
        'size' => 'medium',
        'ingredients' => array('jam','cheese'),
        'pizza' => 'hawaian');
}
/*Select*/
$pizzas = array('pork' => 'BBQ Pork Pizza',
    'chicken' => 'Chicken Pizza',
    'lotus' => 'Lotus Seed Pizza',
    'hawaian' => 'Hawaian Pizza',
    'carbonara' => 'Carbonara Pizza');

print '<select name="pizza">';

foreach ($pizzas as $option => $label) {
    print '<option value="' .$option .'"';
    if ($option == $defaults['pizza']) {
        print ' selected';
    }
    print "> $label</option>\n";
}
print '</select><br>';
```

form4.php

5.6. Displaying Default Values

2. Setting defaults in a multivalued <select> menu:

```
/*Multiselection*/
$ingredients = array('cheese' => 'Extra cheese',
    'jam' => 'Jam',
    'mushrooms' => 'Mushrooms',
    'bbq' => 'BBQ sauce');

print '<select name="ingredient[]" multiple>';
$selected_options = array();

foreach ($defaults['ingredient'] as $option) {
    $selected_options[$option] = true;
}

foreach ($ingredients as $option => $label) {
    print '<option value="' . htmlentities($option) . '"';
    if (array_key_exists($option, $selected_options)) {
        print ' selected';
    }
    print '>' . htmlentities($label) . '</option>';
    print "\n";
}
print '</select><br>';
```

5.6. Displaying Default Values

3. Setting defaults for checkboxes:

A.2.27. On your own, add the piece of code, where a checkbox will be added to the previous form that represents if the user wants the pizza to be delivered. It will be checked by default.

4. Setting defaults for radio buttons:

A.2.28. Now, you have to add some radio buttons (small, medium and large) where the medium size will be selected by default.

5.7. Putting all toguether

- Now we are going to mix everything we have learn in order to get this:
  - Displaying a form, including default values
  - Validating the submitted data
  - Redisplaying the form with error messages and preserved user input if the submitted data isn't valid
  - Processing the submitted data if it is valid

- Follow next steps…

5.7. Putting all toguether

1. Here you can find a new php file called *FormHelper.php*. This kind of file is very common in PHP frameworks, because they help us to build "Forms" quickly (View layer).
2. Take the last version of *form3.php* you did before. Rename the file into *index.php*. In the same folder you should have *FormHelper.php*.
3. Create another file called form.html.php.
4. Open *index.php* and make these changes. The first thing will be to add a require statement with the *FormHelper.php* file.

```
1  <?php
2
3  // This assumes FormHelper.php is in the same directory as
4  // this file.
5  require 'FormHelper.php';
6
```

5.7. Putting all toguether

5. Inside *index.php*, you are going to change *show_form* method in order to create a new form using *FormHelper* class:

```
// Display the form
function show_form($errors=[],$input=[]) {
    $defaults = array('name' => 'Type your name',
                      'age'  => 'Type your age');
    $form = new FormHelper($defaults);
    include 'complete-form.php';
}
```

form.html.php

5.7. Putting all toguether

6. Finally, copy this code into *form.html.php*:

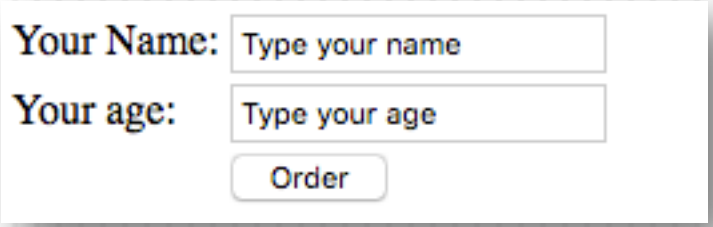
```
<form method="POST" action="<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
    <?php if ($errors) { ?>
        <tr>
            <td>Please correct these errors::</td>
            <td><ul>
                <?php foreach ($errors as $error) { ?>
                    <li><?= $form->encode($error) ?></li>
                <?php } ?>
            </ul></td>
    <?php }  ?>

    <tr><td>Your Name:</td><td><?= $form->input('text', ['name' => 'name']) ?></td></tr>
    <tr><td>Your age:</td><td><?= $form->input('text', ['name' => 'age']) ?></td></tr>

    <tr><td colspan="2" align="center"><?= $form->input('submit', ['value' => 'Order']) ?>
    </td></tr>
</table>
</form>
```

5.7. Putting all toguether

7. You should obtain something like this:



5.7. Putting all toguether

8. The basics from the FormHelper.php:
   A. *Print a select:* make use of this method. You have to pass an array of options and an array of attributes.

```
public function select($options, $attributes = array()) {
```

- *Example:*

```
<tr><td>Pick one sweet item:</td>
    <td><?= $form->select($GLOBALS['sweets'], ['name' => 'sweet']) ?></td>
</tr>
```

5.7. Putting all toguether

B. *Print a input text*: make use of this method, where $type has to be set to 'text':

```
public function input($type, $attributes = array(), $isMultiple = false) {
```

- *Example:*

```
<tr><td>Your Name:</td><td><?= $form->input('text', ['name' => 'name']) ?></td></tr>
```

C. *Print a radio button*: make use of the same method as before, setting the type='radio'

```
<?= $form->input('radio',['name' => 'size', 'value' => 'large']) ?> Large <br/>
```

5.7. Putting all toguether

D. *Print a checkbox*: make use of this method, where $type has to be set to 'checkbox':

```
public function input($type, $attributes = array(), $isMultiple = false) {
```

- *Example:*

```
<?= $form->input('checkbox',['name' => 'delivery', 'value' => 'yes']) ?> Yes
```

E. *Print a text area*: make use of

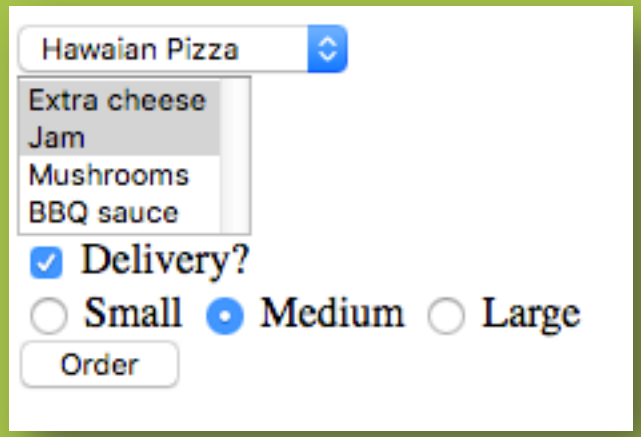
```
public function textarea($attributes = array()) {
```

- *Example:*

```
<?= $form->textarea(['name' => 'comments']) ?>
```

5.7. Putting all toguether

A.2.29. In this exercise you are going to make another version of a previous exercise but making use of *FormHelper* class.



In this case, in order to validate the form you should check that all fields are required (except from Delivery checkbox).

5.7. Putting all toguether

A.2.30. Continuing with previous exercise, change the validation method for the ingredients field, in order to make that at least two of the ingredients are selected.

5.7. Putting all toguether

A.2.31. If you look at the HTML generated fot the radio buttons type in the previous exercise, values are not correctly set, because they are all set to the default value if there is one. Can you find where is the piece of code that makes that assumption? How could you correct it in order the form to have the result expected?