Web Design and Development II

Week 10: PHP Form Handling

1.0 Overview

When the user fills out a web form and clicks the submit button, the form data is sent for processing via PHP file and the form data can be sent with the **HTTP POST, GET or REQUEST** methods.

1.1 PHP GET & POST

Most commonly used methods are **GET & POST** used by web browser to communicates with the server. Communication is through HTTP (Hypertext Transfer Protocol) methods. GET and POST are examples of **superglobal variables.**

- ***What is a superglobal?.***

1.2 PHP Global Variable

SuperGlobals are several predefined variables in PHP. Variables that are always accessible, regardless of scope. Can be accessed from any function, class or file without having to do anything special.

Examples in PHP:

- **$GLOBALS**
- **$_SERVER**
- **$_REQUEST**
- **$_POST**
- **$_GET**
- **$_FILES**
- **$_ENV**
- **$_COOKIE**
- **$_SESSION**

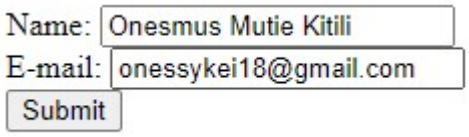
$GLOBALS is a PHP super global variable used o access other global variables from anywhere in the PHP script (methods). PHP stores all global variables in an array called **$GLOBALS[index**]. The index holds the name of the variable. The example below shows how to use the super global variable $GLOBALS:

```
<?php
$x = 75;
$y = 25;
function addition() {
   $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z;
?>
```

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function!

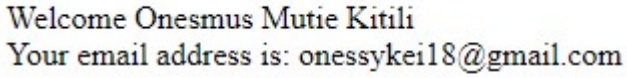
1.2.1 HTTP GET Method

```
<html>
<title > My first Form </title>
<body>
<form action="welcome.php" method="GET">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```



Welcome.php

```
<html>
<body>
Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
</body>
</html>
```



http://localhost/trials/welcome.php?name=Onesmus+Mutie+Kitili&email=onessykei18%40gmail.com

Advantage of GET method

1. Since the data sent by the **GET method** are displayed in the URL, it is possible to bookmark the page with specific query string values.

Disadvantage of GET method

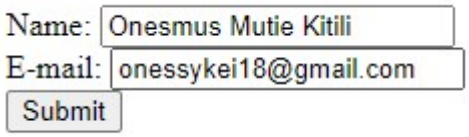
1. The **GET method** is not suitable for passing sensitive information such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.

2. Because the **GET method** assigns data to a server environment variable, the length of the URL is limited. So, there is a limitation for the total data to be sent.

In **GET method,** the data is sent as URL parameters that are usually strings of name and value pairs separated by ampersands (&). In general, a URL with GET data will look like this: http://localhost/trials/welcome.php?**name**=Onesmus+Mutie+Kitili&**email**=onessykei18%40gmail.com. The bold parts in the URL are the GET parameters and the italic parts are the value of those parameters. More than one parameter=value can be embedded in the URL by concatenating with ampersands (&). One can only send simple text data via GET method.

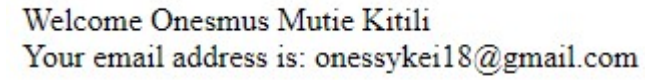
1.2.2 HTTP POST METHOD

```
<html>
<title > My first Form </title>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```



Welcome.php

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
</body>
</html>
```



http://localhost/trials/welcome.php

Advantages of POST method

- It is more secure than GET because user-entered information is **never visible in the URL query string** or in the server logs.
- There is a **much larger limit on the amount of data** that can be passed and one can send text data as well as binary data (uploading a file) using POST.

Disadvantages of POST method

- Since the data sent by the POST method is not visible in the URL, so it is **not possible to bookmark the page** with specific query.

1.2.3 The $_SERVER["PHP_SELF"] variable

What is the $_SERVER["PHP_SELF"] variable?

The $_SERVER["PHP_SELF"] is a **super global** variable that returns the filename of the currently executing script. So, the $_SERVER["PHP_SELF"] **sends the submitted form data to the page itself, instead of jumping to a different page. Form processing is done by a script within the page. Incase of an error,** error messages are displayed on the same page.

1.2.4 PHP REQUEST variable

PHP provides another **superglobal variable** $_REQUEST that contains the values of both the **$_GET** and **$_POST** superglobal variables. The **superglobal variables** $_GET, $_POST and $_REQUEST are built-in variables that are always available in all scopes throughout a script.

1.2.4.1 Capturing inputs by $_REQUEST

```
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>
```

1.2.4.2 Capturing inputs by $_POST

```
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>
```

1.3 PHP Form validation

Can be done by some functions, for example, the ***htmlspecialchars()*** *function.* This function converts special characters to HTML entities. Characters are **harmful**. Entities are **harmless.** It replaces HTML characters like **< and >** with entities like **< and >.** Hackers use such scripts via HTML or **JavaScript. This is called C**ross-site Scripting attacks.

Validation process

The **first thing** we will do is to pass all variables through PHP's **htmlspecialchars() function.** The htmlspecialchars() function converst this code:

<script>location.href('http://www.hacked.com')</script>

To this code:<script>location.href('http://www.hacked.com')</script>

This code has HTML escaped code. The code is now safe to be displayed on a page or inside an e-mail.

Next, we do two more things:

Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP **trim() function).** Remove backslashes (\) from the user input data (with the **PHP stripslashes() function).** This can be done by creating a function that will do all the checking for us (which is much more convenient than writing the same code over and over again). We will name the function **test_input().** The **test_input ()** is then used to check all the **$_POST variables.**

1.3.1 Validating HTML Form

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
```

1.3.2 Checking $_POST variables with test_input() function

```
<?php
// define variables and set to empty values
$name = $email = $gender = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $gender = test_input($_POST["gender"]);
}
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

Notice that at the start of the script, we check whether the form has been submitted using $_SERVER["REQUEST_METHOD"]. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form. **The next step is to make input fields required and create error messages if needed.**

1.3.3 Identifying the Required Fields

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Gender	Required. Must select one

From the validation rules table above, we see that the "**Name", "E-mail", and "Gender**" fields are required. These fields cannot be empty and must be filled out in the HTML form.

1.3.4 Required Field;

In the following code we have added some new variables: **$nameErr, $emailer** and **$genderErr.** These error variables will hold error messages for the required fields. We have also added an if else statement for each **$_POST** variable. This checks if the **$_POST variable** is empty (with the PHP **empty() function).** If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the **test_input() function:**

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr  = "";
$name = $email = $gender  = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }
  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

1.3.5 Displaying Error messages

```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">

Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">

</form>


?>
```

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out etc.

1.3.6 Validating Name

The code below shows a simple way to check if the name field only contains **letters and whitespace**. If the value of the name field is not valid, then store an error message.

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

The **preg_match()** function searches a string for pattern, returning true if the pattern exists, and false otherwise.

1.3.7 Validating Email

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function. In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```