Web Engineering LAB

Lab # 10
Form Handling & Form Validation

Instructor: Hurmat Hidayat

Course Code: SL3003

Semester Spring 2023

Department of Computer Science,
National University of Computer and Emerging Sciences FAST
Peshawar Campus

Content

Super Global

Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

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

$_SERVER

It is a PHP super global variable that stores the information about headers, paths and script locations. Some of these elements are used to get the information from the superglobal variable $_SERVER. Below program illustrates the use of $_SERVER in PHP:

```
<?php
	echo $_SERVER['PHP_SELF'];
	echo "<br>";

	echo $_SERVER['SERVER_NAME'];
	echo "<br>";

	echo $_SERVER['HTTP_HOST'];
	echo "<br>";

	echo $_SERVER['HTTP_USER_AGENT'];
	echo "<br>";

	echo $_SERVER['SCRIPT_NAME'];
	echo "<br>"
?>
```

In the above code we used the $_SERVER elements to get some information. We get the current file name which is worked on using 'PHP_SELF' element. Then we get server name used currently using 'SERVER_NAME' element. And then we get the host name through 'HTTP_HOST'.

HTML Form Handling

When you send a letter, it needs to have certain features to be delivered correctly: an address, a stamp, etc. You also use a language that the receiver understands, right? The same applies to the data packets you send to see a website. We use a protocol called HTTP (Hypertext Transfer Protocol).

So when you have a website, you need to have a server (machine) where it lives. When the server receives an incoming request (letter) from someone accessing your website (known as client), it sends back a response (another letter) to the client.

HTTP has several methods to handle such requests and responses. The two most common HTTP methods are: GET and POST.

GET

GET is used to request data from a specified resource. This is what a HTML form which uses the GET method looks like:

```
<html>
<body>

<form method="get" action="process.php">
  <p>
    <label for="username">Name*</label><br>
    <input type="text" name="username">
  </p>

  <p>
    <label for="email">Email*</label><br>
    <input type="email" name="email">
  </p>

  <p><button>Submit</button></p>
</form>
```

Save the file and refresh your browser. The example above will send the name and email address that the user enters to the **process.php** page to be processed. On that page we can store it in a persistent storage system, like a database or a physical csv file.

Create a new file in your directory, called process.php . It can be empty for the moment. Now that the action page exists, we can submit data through the form. Open the page with the form in your browser, fill the fields with some test data and submit it.

When we submit the form with data, we're redirected to the process.php page. The page is blank at the moment but consider the URL in the address bar, specifically everything after the ? symbol.

```
http://localhost:8887/lab_10/process.php?username=test&email=test%40gmail.com
```

Everything after ? is the data we send through the form to the processing page. It shows the field name and the value that was entered.

To get data that's sent from a form, PHP provides us with the $_GET superglobal.
The $_GET superglobal works like an associative array. The field name in the form is the key in the superglobal

Syntax

```
// Form page
<form method="get">
      <input name="fieldname">
</form>

// Process page
$_GET["fieldname"];
```

As an example, let's use the form we built earlier and $_GET the data from the *username* field. To help the demonstration, let's also echo the data to the page.

Process_data.php

```
<?php

echo "Welcome " . $_GET["username"];

echo "<br>";

echo "Welcome " . $_GET["email"];

?>
```

When the form is submitted, the data is sent through the url to the process.php page. At this point, the data is available to the page, but we don't have access to it yet. To get access, we pull the data from the URL with the $_GET superglobal. Once we have the data, we can do what we need to with it, like echo it out to the page.

POST

To implement the POST method, everything in the HTML form remains the same except this line:

```
<form method="post" action="process.php">
```

Sensitive form data, such as a password, should not be sent with the get method. We should instead use method="post" , as the data that's sent is invisible.

```
<html>
<body>

<form method="post" action="process_data.php">
<!-- <form method="get" action="process_data.php"> -->
  <p>
    <label for="username">Name*</label><br>
    <input type="text" name="username">
  </p>
```

```
  <p>
    <label for="email">Email*</label><br>
    <input type="email" name="email">
  </p>

  <p><button>Submit</button></p>
</form>
```

If you submit the form with test data, the data won't show in the URL bar. Accessing **post** data is the same as accessing **get** data, except we use the $_POST superglobal. To demonstrate, let's change our $_GET request on the *username* from earlier to a $_POST request.

```
<?php

echo "Welcome " . $_POST["username"];

echo "<br>";

echo "Welcome " . $_POST["email"];

?>
```

$_GET vs $_POST. When to use which

It really depends on the situation and the type of data.

- $_GET is an array, passed to the action script via URL parameters.
- $_POST is also an array, but passed to the action script via the HTTP POST method.

They are both superglobals, which means they are accessible in any scope, and we can access them from any file, class or function.

- **GET** is used to send non-sensitive data, such as a search term.
- **POST** is used to send sensitive, or potentially sensitive data, such as email addresses and passwords.

Get and Post also have their own advantages, and disadvantages.

- **GET** has a limit of 2000 characters that can be sent. It also allows us to bookmark the action page.
- **POST** has no character limits and supports advanced functionality. Because the data is not in the url, an action page that receives data via post cannot be bookmarked with that specific data.

REQUEST

The **$_REQUEST** variable contains the contents of $_GET, $_POST, and $_COOKIE.

```
<?php
echo "Welcome " . $_REQUEST["username"];
echo "<br>";
echo "Welcome " . $_REQUEST["email"];
?>
```

Submit form to its own page

Instead of jumping to a different page, we can submit the form to the page it's on by using the **$_SERVER["PHP_SELF"]** superglobal.

The interpreter will echo the filename of the current PHP page in the action attribute. So the current page will become the action page.

NOTE *The superglobal can be used by attackers to exploit your application by injecting malicious code. However, we can make the code safer if we convert any special characters in the superglobal to HTML entities.*

PHP allows us to do this easily with the built-in **htmlspecialchars()** method. The method will perform the following conversions.

- & (ampersand) converts to &
- " (double quote) converts to "
- ' (single quote) converts to '
- < (less than) converts to <
- >(greater than) converts to >

To use it, all we have to do is pass the $_SERVER["PHP_SELF"] superglobal to htmlspecialchars() as an argument.

```
<html>
<body>
<form method="post" action="
        <?php
             echo   htmlspecialchars($_SERVER["PHP_SELF"]);
        ?>">
  <p>
    <label for="username">Name*</label><br>
    <input type="text" name="username">
  </p>

  <p>
    <label for="email">Email*</label><br>
    <input type="email" name="email">
  </p>
```

```
  <p><button>Submit</button></p>
</form>
```

Form Validation

When processing a form, it's critical to validate user inputs to ensure that the data is in a valid format.

There are two types of validations:

- client-side
- server-side

The client-side validation is performed in the web browsers of the users. To validate data at the client side, you can use HTML5 validation or JavaScript. The client-side validation aims to assist legitimate users in entering data in the valid format before submitting it to the server. However, client-side validation doesn't prevent malicious users from submitting data that can potentially exploit the application.

The server-side validation validates data in the web server using PHP. To validate data in PHP, you can use the filter_var() and filter_input() functions.

Consider the following example form to understand the concepts of Form validation:

Example:

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male Other *

Submit

Your Input:

Validation Rules:

The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

First we will look at the plain HTML code for the form:

Text Fields:
The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Radio Button:
The gender fields are radio buttons and the HTML code looks like this:

```
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
```

The Form Element

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

When the form is submitted, the form data is sent with method="post".

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's **htmlspecialchars()** function. We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)

2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create 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().**

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}
```

```php
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

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.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

The next step is to make input fields required and create error messages if needed. As we can 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.

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. 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
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

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["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
```

```
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

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

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>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<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, "Does the Website field contain a valid URL?".

PHP - Validate Name, E-mail, and URL

```php
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression
also allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
```

```
}
?>
```

The next step is to show how to prevent the form from emptying all the input fields when the user submits the form.

Complete Form Example

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression
also allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
```

```
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>

<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_S
ELF"]);?>">
  Name: <input type="text" name="name" value="<?php echo $name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?php echo $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?php echo $website
;?>">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comm
ent;?></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="female") echo "checked";?> value="female">Female
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="male") echo "checked";?> value="male">Male
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="other") echo "checked";?> value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
```

```
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

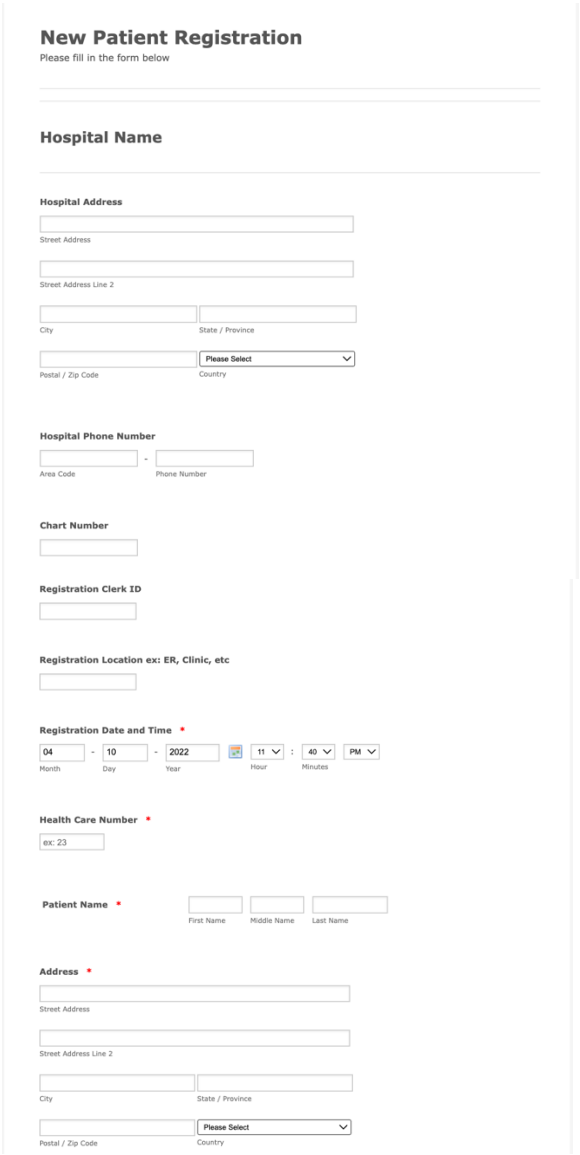
References

1. https://www.koderhq.com/tutorial/php/form-handling/
2. https://www.w3schools.com/php/php_forms.asp
3. https://www.w3schools.in/php/get-post

Lab Tasks:

1. Consider the patient registration form and do the following:
   a. Create and Design the form using HTML, CSS and JS
   b. Define Validation rules
   c. Validate form data with PHP
   d. Display error messages
   e. Finally display the submitted data.

Patient Registration Form



Phone Number - Home *

Area Code - Phone Number

Phone Number - Work (optional)

Area Code - Phone Number

E-mail (optional) ex: myname@example.com

Sex *

Date of Birth * Month Day Year

Marital Status

Province of Residence *

In case of emergency...

Emergency Contact: * First Name Last Name

Relationship *

Contact Number * Area Code - Phone Number

Family Doctor Name

Family Doctor Phone Number

Area Code - Phone Number

Reason for Registration *

Reason for Registration *

Additional Notes

Doctor Name (if applicable)

Taking any medications, currently?

- Yes
- No

If yes, please list it here

Submit Form