Server-side
Web Development

Unit 04. Form validation. Guided example.

.sf-sub-indicator
.cart-menu .cart-icon-wr
der-outer.transparent header#top
nav .sf-menu > li.current_page_
nav .sf-menu > li.current-menu-
nav > ul > li > a:hover > .sf-sub
nav ul #search-btn a:hover span,
nav .sf-menu > li.current-menu-ite
:hover .icon-salient-cart,.ascend
!important;color:#ffffff!impo
parent header#top nav>ul>li.butt
-widget-area-toggle a i

IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2023-24

Index

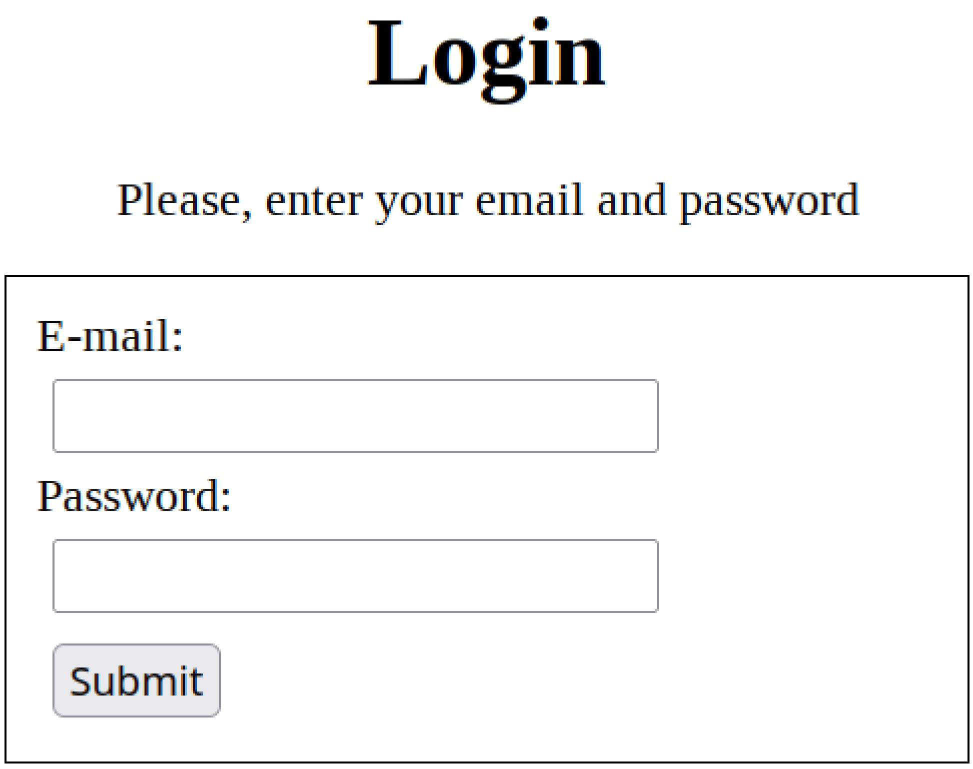
In this example we're going to show how to validate a simple form, using auto-validation techniques.

> **Security is important!**: This guide will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

1 PHP Form validation

The HTML form we will be working at contains 2 input fields, email and password, and a submit button:

```html
<form action="" method="post">
    <label> E-mail: <br>
        <input type="text" name="email">
    </label>
    <label>Password: <br>
        <input type="password" name="pass">
    </label>
    <input type="submit" name="submit" value="Submit">
</form>
```



We will follow the next validation rules:

- **Email**: Required and must contain a valid email address (with @ and .)
- **Password**: Required and must have at least 8 characters.

1.1 Form action

As we want the form to be auto-validated, we have to send the data to itself:

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

> The `$_SERVER["PHP_SELF"]` is a super global variable that returns the filename of the currently executing script.

However this code is vulnerable to some **Cross Site Scripting (XSS)** attacks, because and attacker can add a Javascript code to the url to execute malicious code. Try to add the next string to your form url:

```
...login.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

This can be avoided by using the **`htmlspecialchars()`** function, which converts special characters to HTML entities. The form code should look like this:

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

1.2 Validate form data

In order to get clean data from the inputs, we are going to do 2 actions:

- Remove all the tags with the **`strip_tags()`** function.
- And strip unnecessary characters (extra space, tab, newline), with the PHP **`trim()`** function.

```
$email = trim(strip_tags($POST['email']));
```

At the start of the script, we'll check if 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:

```
<?php
// define variables and set to empty values
$email = $pass = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $email = trim(strip_tags($_POST['email']));
    $pass = trim(strip_tags($_POST['pass']));
}
?>
```

1.3 Required fields

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

First, we add new variables for each error message:

```php
$emailErr = $passErr = "";
```

Also, add 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:

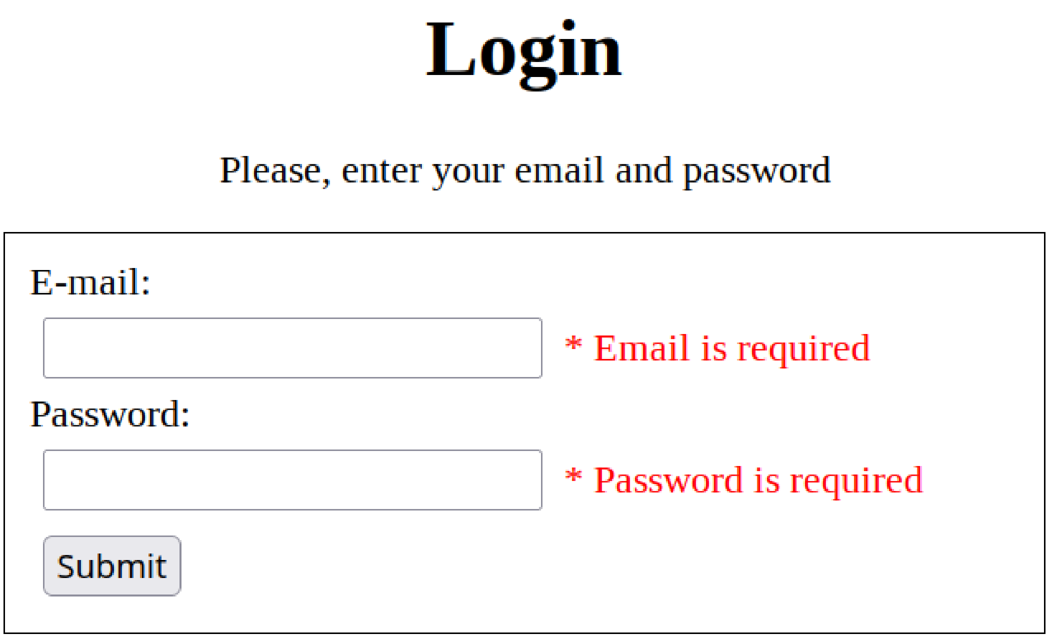
```php
$email = $pass = "";
$emailErr = $passErr = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if(empty($_POST['email'])) {
        $emailErr = "* Email is required";
    } else {
        $email = trim(strip_tags($_POST['email']));
    }

    if(empty($_POST['pass'])){
        $passErr = "* Password is required";
    } else {
        $pass = trim(strip_tags($_POST['pass']));
    }
}
```

1.4 Display the error messages

In the HTML form, we'll add a little script after each required field, which generates the correct error message:

```html
<input type="text" name="email"> <span class="error"> <?= $emailErr; ?>
↪   </span>
 ...
<input type="password" name="pass"> <span class="error"> <?= $passErr; ?>
↪   </span>
```

Try it!:



The next step is to validate the input data.

1.5 Validate inputs

The easiest and safest way to check if an email address is well-formed is to use PHP's **`filter_var()`** function with the **`FILTER_VALIDATE_EMAIL`** filter.

In the code below, if the e-mail address is not well-formed, then store an error message:

```php
if(empty($_POST['email'])) {
    $emailErr = "* Email is required";
} else {
    $email = trim(strip_tags($_POST['email']));
    //Check if the email is well formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $emailErr = "* Invalid email format";
    }
}
```

> Note that we are showing a different message for each error type.

For the password, we simply test if it has a length of 8 characters or more, after cleaning the string:

```
if(empty($_POST['pass'])){
    $passErr = "* Password is required";
} else {
    $pass = trim(strip_tags($_POST['pass']));
    //Check if the password has at least 8 chars
    if (strlen($pass) < 8) {
        $passErr = "* The password must have at least 8 characters";
    }
}
```

1.6 Keep the values in the form

Now, if the user submits the values and the inputs have any errors, the form shows the error messages, but the inputs become blank and the user needs to enter the values again. This is especially frustrating in long forms!

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute using the echo tag:

```
<input type="text" name="email" value="<?= $email;?>"> >
...
<input type="password" name="pass" value="<?= $pass;?>" >
```

1.7 Send the data to another script

If all the checks are ok, it's time to send the data to another script that should do a check against a database to see if the pair email-password exists and is correct. However we will do a simple check against 2 variables for email and password, for example:

- Email: peter@mail.com
- Password: 12345678

The problem now is how to pass this data. Passing passwords via the GET method is a bad idea for security reasons. And our form is passing the data to itself via the POST method. So we will use the **`header()`** statement for redirecting to the check script and session variables to store the variables.

If the check script is named `checkuser.php`, the redirection statement should be:

```php
header("Location:checkuser.php");
```

To create a session, call the function:

```php
session_start();
```

at the start of your scripts.

Each time we store the POST data read by the form in the `$email` and `$pass` variables, we do the same in the session variables:

```php
$email = trim(strip_tags($_POST['email']));
$_SESSION['email'] = $email;
...
$pass = trim(strip_tags($_POST['pass']));
$_SESSION['pass'] = $pass;
```

Another thing we need to do is to check if the inputs have any errors or if they are correct, in that case we will send them to the `checkuser.php` script. For that we create a boolean variable (`$err`) and initialize it to false. Then, if any input is incorrect, we change it to true. Finally, if the variable `$err` is false, we do the redirection with the correct data. Note that we need to set it `true` if we enter the form page directly, to avoid a direct redirection.

The complete code:

```php
<?php
session_start();

// define variables and set to empty values
$email = $pass = "";
$emailErr = $passErr = "";
$err = false; //variable to check if there have been errors

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if(empty($_POST['email'])) {
        $emailErr = "* Email is required";
        $err = true;
    } else {
        $email = trim(strip_tags($_POST['email']));
        $_SESSION['email'] = $email;
        //Check if the email is well-formed
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $emailErr = "* Invalid email format";
            $err = true;
        }
    }

    if(empty($_POST['pass'])){
        $passErr = "* Password is required";
        $err = true;
    } else {
        $pass = trim(strip_tags($_POST['pass']));
        $_SESSION['pass'] = $pass;
        //Check if the password has at least 8 chars
        if (strlen($pass) < 8) {
            $passErr = "* The password must have at least 8 characters";
            $err = true;
        }
    }
} else {
    $err = true;
}

if (!$err) {
    header("Location:checkuser.php");
}

?>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login sample</title>
    <meta name="author" content="Ricardo Sanchez">
    <meta name="description" content="a sample form to show validation
     ↪  techniques">
    <link rel="stylesheet" type="text/css" href="./main.css">
</head>
<body>
<h1>Login</h1>
```

```
<p>Please, enter your email and password</p>

<form method="post" action="<?php echo
↪   htmlspecialchars($_SERVER["PHP_SELF"]);?>">
    <label> E-mail: <br>
        <input type="text" name="email" value="<?= $email;?>"  <span
        ↪   class="error"> <?= $emailErr; ?> </span>
    </label>
    <label>Password: <br>
        <input type="password" name="pass" value="<?= $pass;?>"> <span
        ↪   class="error"> <?= $passErr; ?> </span>
    </label>
    <input type="submit" name="submit" value="Submit">
</form>

</body>
</html>
```

The checkuser.php script simply gets the session variables and compare them with the correct values, showing a message according to the result:

```
<?php
session_start();

$correctEmail = "peter@mail.com";
$correctPass = "12345678";

if($_SESSION['pass']==$correctPass && $_SESSION['email']==$correctEmail) {
    echo "Login correct: <br>";
} else {
    echo "Login incorrect: <br>";
}
echo $_SESSION['email'] . ", " . $_SESSION['pass'] . "<br>";

session_destroy();
```

You can get all the code from the GitHub repository.