Server-side
Web Development

Unit 04. Form validation. Guided example.

Fidel Oltra, Ricardo Sánchez

IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2024-25

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

We are going to do a form to enter providers data. So the data needed are: name, email and CIF.

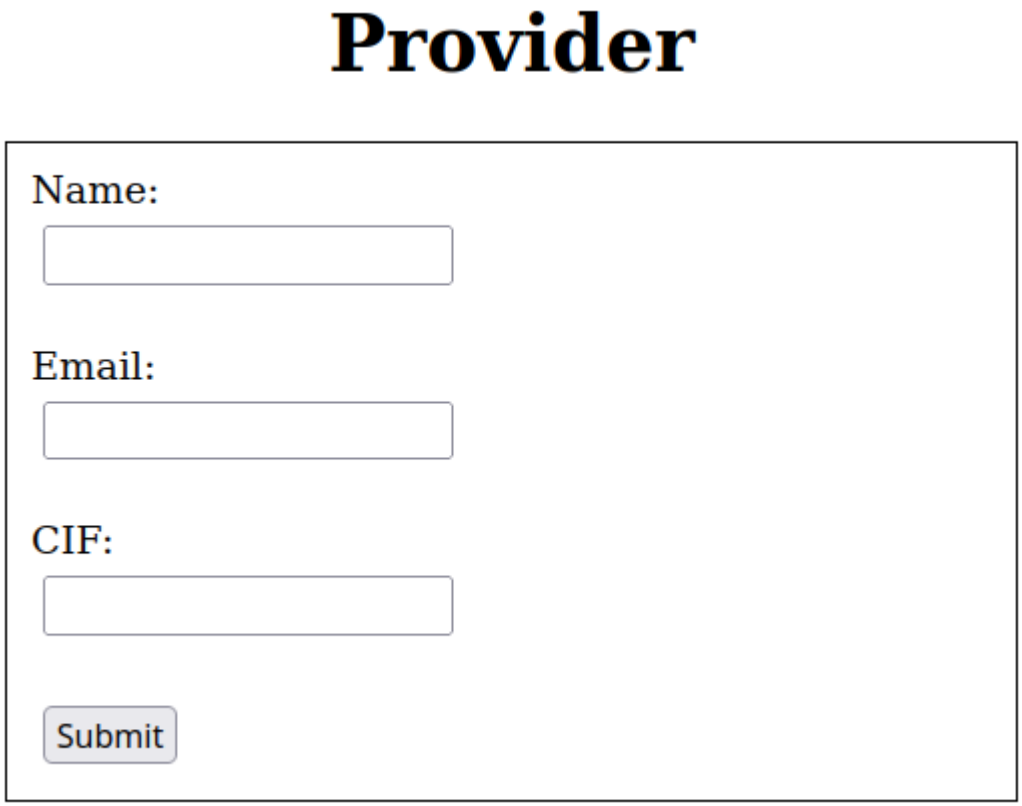
The HTML form we will be working at contains 3 input fields: name, email and CIF, and a submit button:

```
<form action="" method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name"><br><br>

    <label for="email">Email:</label>
    <input type="text" id="email" name="email"><br><br>

    <label for="cif">CIF:</label>
    <input type="text" id="cif" name="cif"><br><br>

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



> Note that we are using `type="text"` instead of `"email"` for the email field. That will ensure that the validation will be done in the PHP code.

We will follow the next validation rules:

- **Name**: Required and must have at least 4 characters.
- **Email**: Required and must contain a valid email address (with @ and .)
- **CIF**: Required and must be a string starting with a letter and 8 digits, ex. C12345678.

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.

```
// define an array to store the provider values
$provider = [];

$provider['name'] = trim(strip_tags($_POST['name']));
$provider['email'] = trim(strip_tags($_POST['email']));
$provider['cif'] = trim(strip_tags($_POST['cif']));
```

At the start of the script, we'll check if the form has been submitted via POST 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 an array to store the provider values
$provider = [];

if ($_SERVER["REQUEST_METHOD"] == "POST") {
```

```php
    $provider['name'] = trim(strip_tags($_POST['name']));
    $provider['email'] = trim(strip_tags($_POST['email']));
    $provider['cif'] = trim(strip_tags($_POST['cif']));
?>
```

1.3 Required fields

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

We are going to validate the data using a function name `validateProvider()` that accepts an array with the provider values and returns an array containing the detected errors. If no errors in the inputs, the array will be empty.

```php
function validateProvider(array $provider): array {
    $errors = [];

    return $errors;
}
```

In the function, 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
function validateProvider(array $provider): array {
    $errors = [];
    if (empty($provider['name'])) {
        $errors['name'] = "* Name is required";
    }

    if(empty($provider['email'])) {
        $errors['email'] = "* Email is required";
    }

    if(empty($provider['cif'])) {
        $errors['cif'] = "* CIF is required";
    }

    return $errors;
}
```

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:

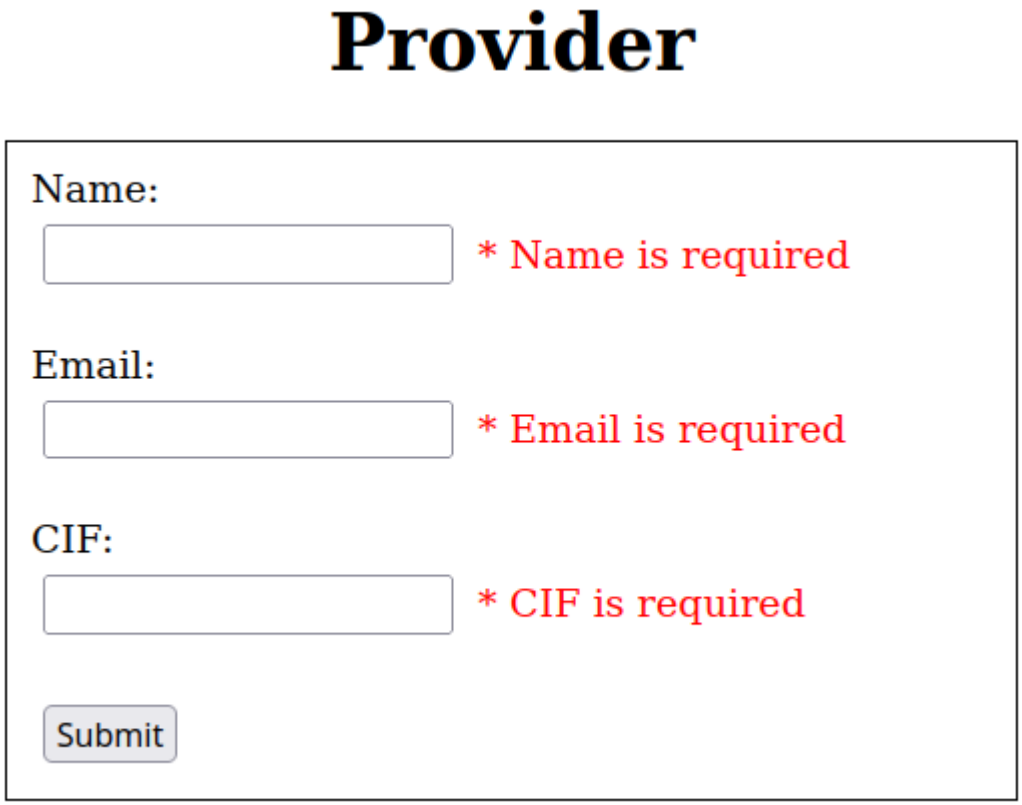
```
<label for="name">Name:</label>
<input type="text" id="name" name="name">
<span class="error"> <?= $errors['name'] ?? '' ?> </span> <br><br>

<label for="email">Email:</label>
<input type="email" id="email" name="email">
<span class="error"> <?= $errors['email'] ?? '' ?> </span> <br><br>

<label for="cif">CIF:</label>
<input type="text" id="cif" name="cif">
<span class="error"> <?= $errors['cif'] ?? '' ?> </span><br><br>
```

Note that we are using the null coalescing operator (??), so, in case an array's key is null, we print an empty string

Try it!:



The next step is to validate the input data.

1.5 Validate inputs

For the provider's name, we only need to check if the string has at least 4 characters:

```php
if (empty($provider['name'])) {
    $errors['name'] = "* Name is required";
} elseif (strlen($provider['name']) < 4) {
    $errors['name'] = "* Name must be at least 4 characters long";
}
```

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

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($provider['email'])) {
        $errors['email'] = "* Email is required";
    } elseif (!filter_var($provider['email'], FILTER_VALIDATE_EMAIL)) {
        $errors['email'] = "* Invalid email format";
    }
```

For the CIF, we need a regular expression to check the validity of the string:

```php
if(empty($provider['cif'])) {
    $errors['cif'] = "* CIF is required";
} elseif (!preg_match("/^[A-Z a-z]{1}[0-9]{8}$/", $provider['cif'])) {
    $errors['cif'] = "* Invalid CIF format";
}
```

1.6 Keep the values in the form

Now, if the user submits the values and the inputs have any error, 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:

```php
<input type="text" id="name" name="name" value="<?= $provider['name'] ?? ''
 ↪  ?>">
...
<input type="text" id="email" name="email" value="<?= $provider['email'] ??
 ↪  '' ?>">
...
<input type="text" id="cif" name="cif" value="<?= $provider['cif'] ?? ''
 ↪  ?>">
```

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 insert the data into a database. Instead of that, we simply will shoe the submitted data.

The problem now is how to pass this data. Passing data 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 another script and session variables to store the variables.

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

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

To create a session, call the function:

```php
session_start();
```

at the start of your scripts.

On the form script we must store the POST data in a session variable before we validate them:

```php
$_SESSION['provider'] = $provider;
$errors = validateProvider($provider);
```

If the validation is passed, the errors array will be empty, so we can redirect to the `show_provider.php`:

```php
if (empty($errors)) {
    header("Location:show_provider.php");
}
```

View the complete code.

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

```php
<?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.