Server-side
Web Development

Unit 04. PHP and Forms.

Fidel Oltra, Ricardo Sánchez

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

Index

So far, we have only worked with static pages, without any interaction with the users. Now we want to add dynamism to our web. The main way to do this is by allowing the users to enter information using **HTML Forms**.

1 Reading data from a Form

The data from the HTML form are sent to the script specified in the ***action*** label of the form. In that script, data are available in an associative array called **$_GET** or **$_POST** depending on the method used in the form.

In addition, they are available in the **$_REQUEST** array too, regardless of the method used in the form.

1.1 POST method

If we have this form:

```
<body>
<form action="destination.php" method="POST">
    Name: <input type="text" name="name"><br>
    1st Surname: <input type="text" name="surname1"><br>
    2nd Surname: <input type="text" name="surname2"><br>
    <input type="submit" value="Send">
</form>
</body>
```

In the script **destination.php** we will read the form attributes using the $_POST or the $_RE-QUEST arrays:

```
$name = $_POST['name'];  // or $name=$_REQUEST['name']
$surname1 = $_POST['surname1']; // or $surname1=$_REQUEST['surname1']
$surname2 = $_POST['surname2']; // or $surname2=$_REQUEST['surname2']
```

1.2 GET method

If the GET method is used in the form, everything works in similar ways except that the data sent by the form are attached to the URL.

If we modify the example above replacing POST with GET:

```
<form action="destination.php" method="GET">
```

```
$name = $_GET['name'];
$surname1 = $_GET['surname1'];
$surname2 = $_GET['surname2'];
```

We won't find any difference with one exception. Let's take a look to our address bar when we send the data with POST.

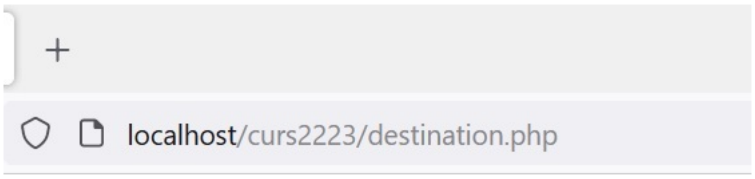

localhost/curs2223/destination.php

Figure 1: FORM sent with POST method

And now, let's compare the URL above with that with the GET method.

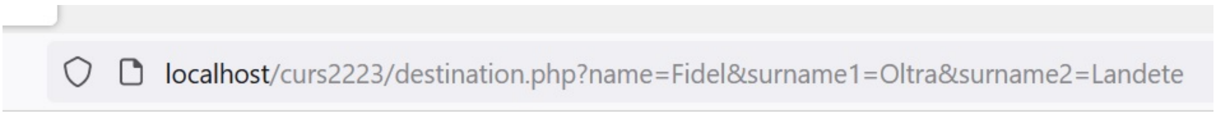

localhost/curs2223/destination.php?name=Fidel&surname1=Oltra&surname2=Landete

Figure 2: FORM sent with GET method

As you can see, the data sent by the form have been attached to the URL of the destination page. That way, the data are visible to the final user. We should decide whether that's important or not.

The advantage of this feature is that we can send data to a script without using a form, just by attaching the variables to the URL in a link.

```
<a href="test.php?name=Marta">link</a>
```

1.3 Reading some special form fields

1.3.1 Reading a checkbox

```html
<fieldset>
    <legend>Please select which sports do you like</legend>
    <label>
        Baseball
        <input type="checkbox" name="baseball"><br>
    </label>
    <label>
        Basketball
        <input type="checkbox" name="basketball"><br>
    </label>
</fieldset>
```

```php
if(isset($_REQUEST['baseball'])) { ... }
if(isset($_REQUEST['basketball'])) { ... }
```

1.3.2 Reading a radio group

A radio group is defined by giving the same name to some radio button elements. Only one radio button in a group can be selected.

```html
<input type="radio" id="html" name="fav_language" value="HTML">
<label for="html">HTML</label><br>
<input type="radio" id="css" name="fav_language" value="CSS">
<label for="css">CSS</label><br>
<input type="radio" id="javascript" name="fav_language"
value="JavaScript">
<label for="javascript">JavaScript</label>
```

```php
$fav_language = $_REQUEST['fav_language'];
echo $fav_language;
```

1.3.3 Simple and multiple SELECT

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
</select>
```

Use the **multiple** attribute to allow the user to select more than one value. The PHP script will receive the data as an **array**.

```
<select id="cars" name="cars[]" size="4" multiple>
```

```
$cars = $_REQUEST['cars'];
foreach ($cars as $car) {
    echo "$car<br>\n";
}
```

We can do the same with checkbox type inputs.

```
<input type="checkbox" name="likes[]" value="Sports"/>Sports<br/>
<input type="checkbox" name="likes[]" value="Music"/>Music<br/>
<input type="checkbox" name="likes[]" value="Reading"/>Reading<br/>
<input type="checkbox" name="likes[]" value="Movies"/>Movies<br/>
```

And then, in the PHP script:

```
if(!empty($_POST['likes'])) {
    echo "You like: ";
    foreach($_POST['likes'] as $like) {
        echo strip_tags($like)." ";
    }
}
```

1.4 Using GET or POST?

It's always better to use the POST method, because GET parameters are passed through the URL and they become more exposed to an attack.

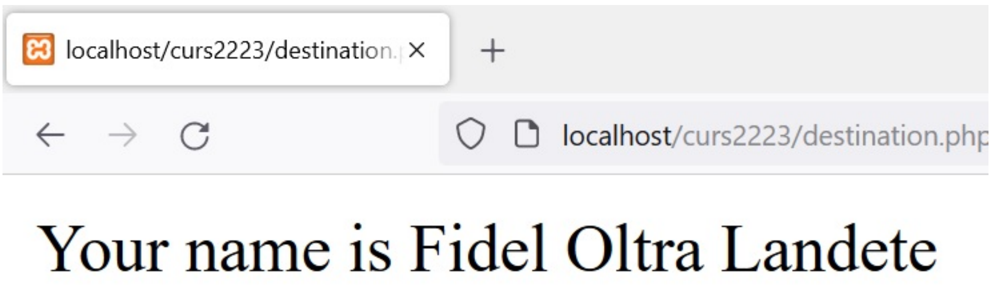
As a rule, we only should use GET in a form when, once the form is submitted, nothing on the server changes. Avoid using GET when the data sent by the form are used to update a database, delete files, include passwords and so on.

2 Validating a Form

We need to validate the received data before using them. It's important to confirm if the data exist and if their content is in the expected format. We have some functions in PHP to check the status of a variable:

- **`isset($variable)`** returns true if the variable exists and is not null
- **`is_null($variable)`** returns true if the variable has a null value
- **`empty($variable)`** returns true if the variable has an empty value (`null`, empty, `0`, `false`) or if the variable doesn't exist.
- **Null coalescing operator**, used to store a value if exists and a default value otherwise.

Let's see an example. In the form above, enter the name and the surnames, press the **Send** button and see what happens.

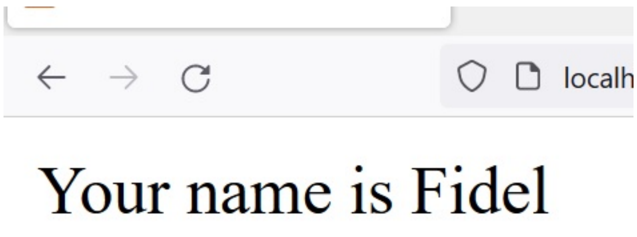


localhost/curs2223/destination.php

Your name is Fidel Oltra Landete

Figure 3: Valid data

Now, return to the form and leave the surnames empty.



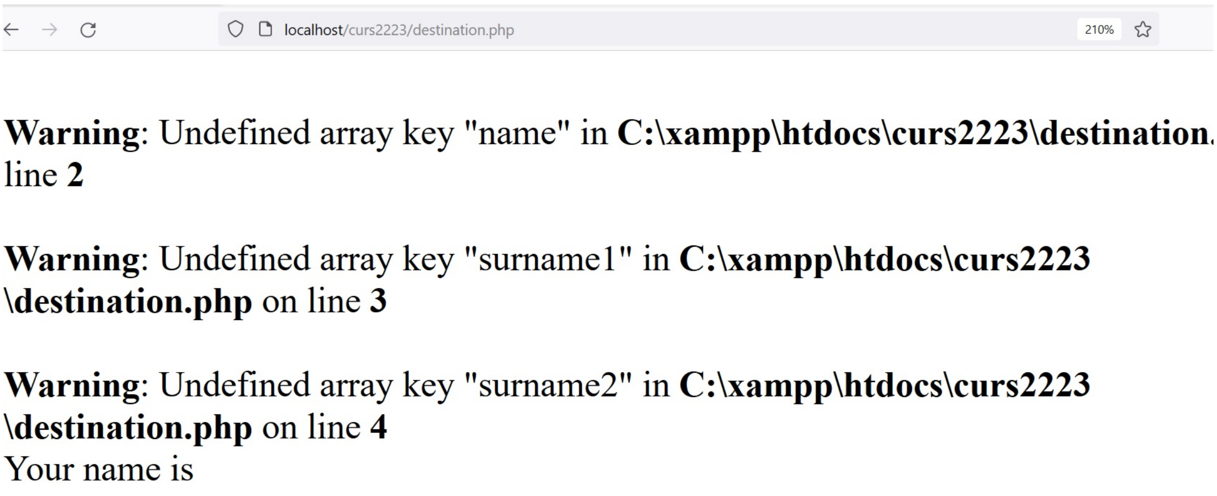
Your name is Fidel

Figure 4: Empty data

As the surnames have been sent empty, they're not displayed. In this case this is not a big problem for us, but in other circumstances it may be dangerous: we can get an error or an unexpected operation.

And what happens if we try to go directly to the destination page without having filled the form? Enter the destination URL directly in your address bar (without the parameters, just in case you are using the

GET method). You'll get some warnings like these:



localhost/curs2223/destination.php

```
Warning: Undefined array key "name" in C:\xampp\htdocs\curs2223\destination.
line 2

Warning: Undefined array key "surname1" in C:\xampp\htdocs\curs2223
\destination.php on line 3

Warning: Undefined array key "surname2" in C:\xampp\htdocs\curs2223
\destination.php on line 4
Your name is
```

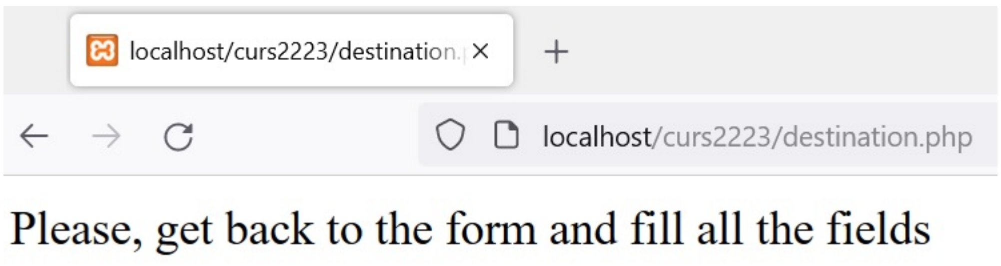
Figure 5: No data received

In this case the data have not been sent empty, they haven't been sent at all.

We can use the function empty to check both if the data exist and if they're not empty. Thus, in the ***destination.php*** page we should do:

```php
if(!empty($_GET['name'])
  && !empty($_GET['surname1'])
  && !empty($_GET['surname2'])) {
    $name = $_GET['name'];
    $surname1 = $_GET['surname1'];
    $surname2 = $_GET['surname2'];
    echo "Your name is $name $surname1 $surname2";
}
else {
    echo "Please, get back to the form and fill all the fields";
}
```

And now, wether the data are empty or not sent, we will get the same result:


localhost/curs2223/destination.
localhost/curs2223/destination.php

Please, get back to the form and fill all the fields

Figure 6: Data empty or not sent

We can add a link to return to the homepage and attach a message to the link. We will be able to read the message using $_GET in the homepage.

In the ***destination.php*** page:

```
$message="Please, fill all the fields";
echo "<a href='form.php?message=$message'>Go back to the form</a>";
```

And in the script where the form is:

```
if(!empty($_GET['message'])) {
    $message=$_GET['message'];
    echo "<h3>$message</h3>";
}
```

3 Security and form handling

We should be aware of the security aspects of our application when there is a exchange of information with the user.

3.1 Cross-Site Scripting (XSS) attacks

XSS occurs when an attacker is capable of injecting a script, often Javascript or SQL code, into the output of a web application.

Example: we have this form:

```
<form action="post.php" method="post">
    <input type="text" name="comment" value="">
    <input type="submit" name="submit" value="Submit">
<form>
```

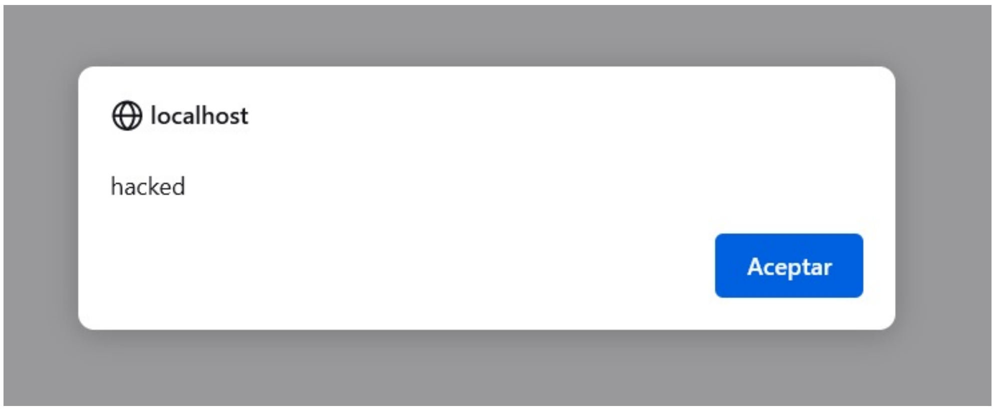
And the page `post.php`:

```
echo $_POST['comment'];
```

If we type in the “comment” field of the form the text:

```
<script>alert("hacked")</script>
```

This is what will happen:


localhost

hacked

Figure 7: Hacked!

This is just an alert, but it could have been a dangerous code!!

To avoid XSS attacks, we can employ some measures:

3.2 Data validation

We must validate that the entered data match the required type.

We can do the validation in the same HTML Form using input features as **type** (type=number), `max` & `min`, `maxlength` and `pattern`.

```
Month (1-12):
<input type="number" name="month" min="1" max="12"/>
DNI: 8 numbers 1 capital letter
<input type="text" pattern="[0-9]{8}[A-Z]{1}" name="dni" />
```

More information about HTML form attributes

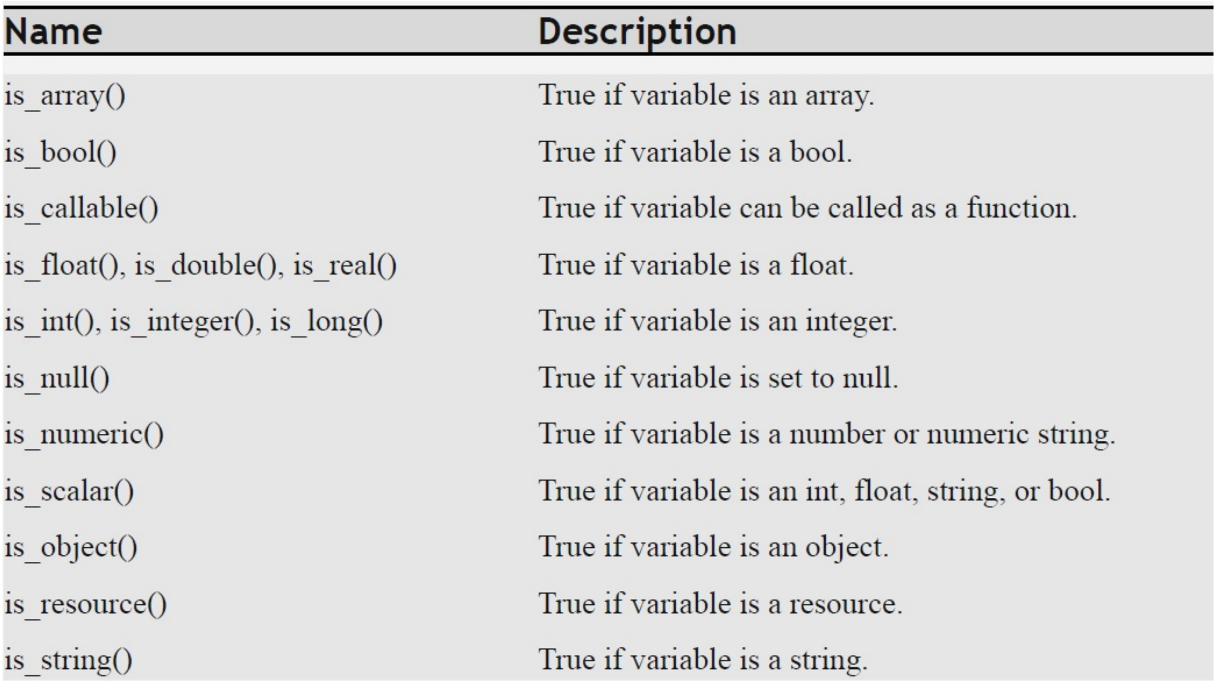
Anyway, as someone can put $_GET parameters directly in a URL, it's a good practice to validate the data in the PHP script too. We can use functions like `is_numeric()`, `is_integer()`, `is_double()` and so on.

We can also use PHP patterns and the `preg_match()` function.

Info about PHP patterns

RegEx101, to test regular expressions

Let's see how to validate a USA phone number and a spanish DNI using patterns.



Name	Description
is_array()	True if variable is an array.
is_bool()	True if variable is a bool.
is_callable()	True if variable can be called as a function.
is_float(), is_double(), is_real()	True if variable is a float.
is_int(), is_integer(), is_long()	True if variable is an integer.
is_null()	True if variable is set to null.
is_numeric()	True if variable is a number or numeric string.
is_scalar()	True if variable is an int, float, string, or bool.
is_object()	True if variable is an object.
is_resource()	True if variable is a resource.
is_string()	True if variable is a string.

Figure 8: Type functions

```php
// phone number USA:
$phone1 = "1-909-466-4344";
// another phone
$phone2 = "992222222";
if (preg_match('/^((1-)?\d{3})-\d{3}-\d{4}/', $phone1)) {
    echo "$phone1: Valid USA phone number<br/>";
} else {
    echo "$phone1: Invalid USA phone number<br/>";
}
if (preg_match('/^((1-)?\d{3})-\d{3}-\d{4}/', $phone2)) {
    echo "$phone2: Valid USA phone number<br/>";
} else {
    echo "$phone2: Invalid USA phone number<br/>";
}
// A Spanish DNI
$dni1="11222333Y";
$dni2="2jlkd9234";
if (preg_match('/^(\d{8})[A-Z]/', $dni1)) {
    echo "$dni1: Valid Spanish DNI<br/>";
} else {
    echo "$dni1: Invalid Spanish DNI<br/>";
}
if (preg_match('/^(\d{8})[A-Z]/', $dni2)) {
    echo "$dni2: Valid Spanish DNI<br/>";
} else {
    echo "$dni2: Invalid Spanish DNI<br/>";
}
```

Another validation method: using **filters** and **filter functions**.

PHP offers some useful predefined filters. You can find them at this link

We can use the `filter_var()` function to check if a expression or variable matches the filter. For example, let's see how to validate an email and an IP address.

```php
if(!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Invalid email address<br/>";
}
if(!filter_var($number, FILTER_VALIDATE_IP)) {
    echo "Invalid IP address<br/>";
}
```

Other filter functions

3.3 Data sanitization

Data sanitization means check if the data are secure and change or remove the bad portions of the data to match our requirements. PHP offers different methods to sanitize data. Basically, all of them remove HTML and PHP tags from a string.

3.3.1 strip_tags() function

The `strip_tags()` function receives two parameters. The first is the string to sanitize. The second, optional, contains the allowed HTML and PHP characters we want to keep in the string.

```php
$comment = strip_tags($_POST["comment"]);
// remove every HTML and PHP tag
$comment = strip_tags($_POST["comment"],'<br>');
// remove every HTML and PHP tag except <br>
```

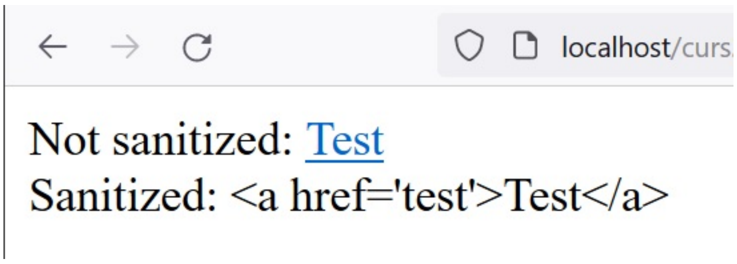
3.3.2 htmlspecialchars() function

The function **`htmlspecialchars()`** prevents code HTML execution, and just shows the HTML characters literally. It replaces < by `&lt;`, > by `&gt;`, etc.

An example:

```
$htmlCode="<a href='test'>Test</a>";
echo "Not sanitized: $htmlCode <br/>";
echo "Sanitized: ".htmlspecialchars($htmlCode, ENT_QUOTES)."<br/>";
```

The output:



```
Not sanitized: Test
Sanitized: <a href='test'>Test</a>
```

Figure 9: Using htmlspecialchars

More info about htmlspecialchars

3.3.3 Sanitization filters

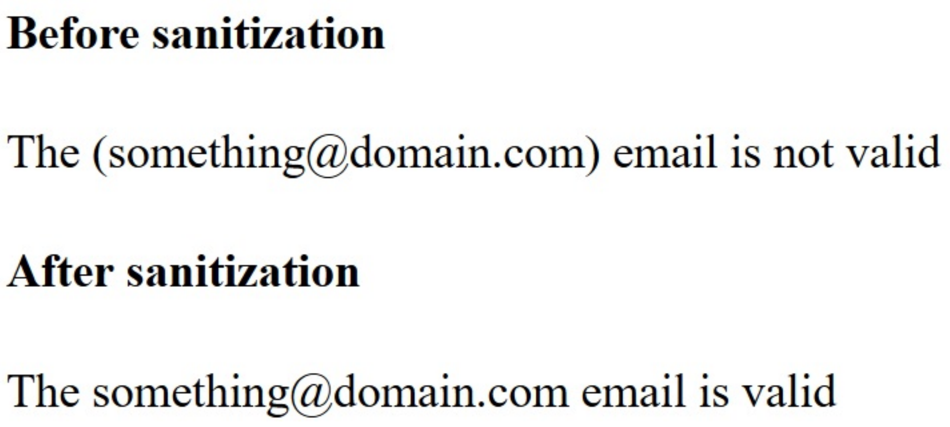
In PHP we have some sanitization filters to be used with the `filter_var()` function seen above. Some of them are `FILTER_SANITIZE_EMAIL`, `FILTER_SANITIZE_URL`, `FILTER_SANITIZE_STRING`, `FILTER_SANITIZE_NUMBER_FLOAT` or `FILTER_SANITIZE_NUMBER_INT`.

When we use `filter_var()` with sanitization filters, the functions returns the sanitized string instead of TRUE/FALSE. An example:

```
$email1="(something@domain.com)";
echo "<h4>Before sanitization</h4>";
if(filter_var($email1,FILTER_VALIDATE_EMAIL)) {
    echo "The $email1 email is valid<br/>";
}
else {
    echo "The $email1 email is not valid<br/>";
}
```

```php
// now we sanitize the email
echo "<h4>After sanitization</h4>";
$email1=filter_var($email1,FILTER_SANITIZE_EMAIL);
// and we try again
if(filter_var($email1,FILTER_VALIDATE_EMAIL)) {
    echo "The $email1 email is valid<br/>";
}
else {
    echo "The $email1 email is not valid<br/>";
}
```

The output:



```
Before sanitization

The (something@domain.com) email is not valid

After sanitization

The something@domain.com email is valid
```

Figure 10: Sanitizing an email

3.4 Header

Sometimes we need to validate the form data before doing anything else. Usually, if the data are no valid we want to redirect the user to the main page.

With the **header** function we can redirect the execution to any other page, and in addition we can send parameters attached to the url that will be read using the `$_GET` array.

The idea is to have a specific script for validation, and then redirect the execution to the corresponding page without the user having to click a link or a button.

> Another way of sending information to a page without user intervention is using the `hidden` fields of a form

`Header` works like this:

```
header("Location:index.html");
```

> It's important that the redirection page (where the `header` function is used) does not show anything on the screen.

Using `header` we can do something like this:

```
<form action="validation.php" method="POST">
<input type="text" name="name"...>
```

and then, in the validation script:

```
// validation.php
if(empty($_POST['name'])) {
    header("Location:error.php"):
}
else {
    header("Location:follow.php");
}
```

If any data is invalid, we can redirect to the form again instead of redirecting to an error page.

```
// validation.php
if(empty($_POST['name']) {
    header("Location:form.html"):
}
else {
    header("Location: follow.php");
}
```

We can attach parameters to the URL so send some information to the destination page.

```
// validation.php
if(empty($_POST['name']) {
    header("Location:form.html?message='Please fill the form'"):
}
else {
    header("Location: follow.php?message='Welcome, user'");
}
```

This information is read using $_GET in the page referred by the URL.

```
if(!empty($_GET['message'])) {
    echo "Message: ".strip_tags($_GET['message']);
}
```

If we jump to a third page from the validation page, the original `$_GET / $_POST` arrays are no longer available. If we need to transfer some form data to a third script from the validation page, we can use two methods. The first one is, as we have seen above, to attach the data to the URL in the `header` function.

```
$name=strip_tags($_POST['name']);
header("Location:thirdpage.php?name=$name");
```

In the third page:

```
if(!empty($_GET['name'])) {
    $name=strip_tags($_GET['name']);
}
```

The second method is to use **sessions**.

4 Sessions

A **session** provides a way to make variables available across multiple pages. In addition, sessions provide control of the user activity.

To begin a session, we will use the function **`session_start()`**. This function must appear before any output is sent to the web page.

```
<?php session_start(); ?>
```

Once the session is started, a **`$_SESSION`** array can be used to store session data. This session data will be available in any other page as long as the session is running.

Let's create a session variable called "username":

```
$_SESSION["password"]=$password;
```

To read the session variable we will do:

```
if(isset($_SESSION["password"])) {
    if($_SESSION["password"]=="...")
```

We can remove all the session variables using the **`session_destroy`** function.

```
session_destroy();
```

The function `session_destroy()` closes the session, too. Therefore, sometimes the session information is stored in cookies. In this case we will need to delete the cookies as well. We can prevent the session information being stored in cookies in the PHP configuration, changing the *session.use_cookies* from 1 to 0.

The session duration can be fixed as well in the PHP configuration (*session.cookie_lifetime* and *session.gc.maxlifetime*) or manually checking the time between interactions.

If we only want to delete one session variable, we can do it with **`unset($_SESSION[...])`**.

Session variables can be used to keep values on forms. That way we can prevent the correct values from being lost if we call again the form because of some invalid data.

```php
<?php session_start();
if(isset($_SESSION['month'])&&isset($_SESSION['year'])){
    $month=$_SESSION['month'];
    $year=$_SESSION['year'];
}
<form action="validation.php" method="POST">
Month<input type="text" name="month" value="<?= $month;?>"/>
Year<input type="text" name="year" value="<?= $year;?>"/>
```

5 Partials

Typically, all pages of a web application have a common part (header, footer, navigation bar, etc.). A good practice consists in extracting these common parts to partial files.

It is convenient to put them inside a partials' folder. Then we will include the partials in the main pages using the `include` function.

In this example we will do 3 parts: for the beginning of all pages, for the header, and for the end.

For instance, if we have the next basic template:

```html
<!DOCTYPE html>
<html>
<head>
    <title> Page Title </title>
    <meta charset="UTF-8">
    <meta name="author" content="">
    <!-- Other head data, as stylesheets -->
</head>

<body>
<header>
    <h1>Page header</h1>
</header>

<!-- Page-content -->

<footer>
    <p>Developed by author's name, year</p>
</footer>

</body>
</html>
```

We can split it in 3 parts: head, header and footer:

```html
<!-- head.part.php -->
<!DOCTYPE html>
<html>
<head>
    <title> Page Title </title>
    <meta charset="UTF-8">
    <meta name="author" content="">
    <!-- Other head data, as stylesheets -->
</head>

<body>
```

```html
<!-- header.part.php -->
<header>
    <h1>Page header</h1>
</header>
```

```html
<!-- footer.part.php -->
<footer>
    <p>Developed by author's name, year</p>
</footer>

</body>
</html>
```

And then include them in each page we make:

```php
<?php
//main.php
include 'parts/head.part.php';
include 'parts/header.part.php';

echo "Main content\n";

include 'parts/footer.part.php';
```

An improvement is to create variables for the title, author, etc. in the main file and use them in the partial files:

```php
<?php
//main.php

//Template variables
$pageTitle = "My Page Title";
$authorName = "My name";
$pageHeader = "My page header";

include 'parts/head.part.php';
include 'parts/header.part.php';

echo "Main content\n";

include 'parts/footer.part.php';
```

```html
<!-- head.part.php -->
<!DOCTYPE html>
<html>
<head>
    <title> <?= $pageTitle ?> </title>
    <meta charset="UTF-8">
    <meta name="author" content="<?= $authorName ?>">
    <!-- Other head data, as stylesheets -->
</head>

<body>
```

```html
<!-- header.part.php -->
<header>
    <h1> <?= $pageHeader ?> </h1>
</header>
```

```html
<!-- footer.part.php -->
<footer>
    <p>Developed by <?= $authorName ?>, year</p>
</footer>

</body>
</html>
```

These variables (or constants) can go on a separate file which we can include in our page.

If we want to add more data to the head, such as stylesheets, we can generate or include them in the main page, assign it to a variable and then use it in the `head.part`.php file. Let's do it with a function:

```php
<?php
//main.php

//Template variables
$pageTitle = "My Page Title";
$authorName = "My name";
$pageHeader = "My page header";
$sheet = "css/main.css";

function linkStylesheet($sheet) {
    return "<link rel='stylesheet' href='$sheet'>\n";
}

include 'parts/head.part.php';
include 'parts/header.part.php';

echo "Main content\n";

include 'parts/footer.part.php';
```

```html
<!-- head.part.php -->
<!DOCTYPE html>
<html>
<head>
    <title> <?= $pageTitle ?> </title>
    <meta charset="UTF-8">
    <meta name="author" content="<?= $authorName ?>">
    <!-- Other head data, as stylesheets -->
    <?= linkStylesheet($sheet) ?>
</head>

<body>
```

We can replace the `linkStylesheet` function with an arrow function, to make it shorter:

```
//main.php
...
$linkStylesheet = fn() => "<link rel='stylesheet' href='$sheet'>\n";
```

```
<!-- head.part.php -->
...
<?= $linkStylesheet(); ?>
```

As in the case of variables, the functions can go on a separate file:

```
<?php
//functions.php
$linkStylesheet = fn() => "<link rel='stylesheet' href='$sheet'>\n";
```

And finally, we can generate automatically the year in the footer:

```
<!-- footer.part.php -->
<footer>
    <p>Developed by <?= $authorName ?>, <?= date("Y");?></p>
</footer>

</body>
</html>
```

With the previous parts and the below main script:

```
<?php
//main.php

//Template variables
$pageTitle = "My Page Title";
$authorName = "My name";
$pageHeader = "My page header";
$sheet = "css/main.css";

include 'parts/functions.php';

include 'parts/head.part.php';
include 'parts/header.part.php';
```

```php
echo "Main content\n";

include 'parts/footer.part.php';
```

the generated code should be:

```html
<!-- head.part.php -->
<!DOCTYPE html>
<html>
<head>
    <title> My Page Title </title>
    <meta charset="UTF-8">
    <meta name="author" content="My name">
    <!-- Other head data, as stylesheets -->
    <link rel='stylesheet' href='css/main.css'>
</head>

<body>
<!-- header.part.php -->
<header>
    <h1> My page header </h1>
</header>
Main content
<!-- footer.part.php -->
<footer>
    <p>Developed by My name, 2022</p>
</footer>

</body>
</html>
```