Working with Forms in PHP

Unit Structure

- Learning Objectives
- Introduction
- Working with forms in PHP
- Validating input data
- using magic quotes
- Storing form data in file
- Saving form data using cookies
- saving form data using sessions

1.1 Learning Objectives

After studying this unit student should be able to:

- understand dynamic web site creation using form
- understand validation controls to provide validation on input
- understand saving data using cookie and session

1.2 Introduction

Dynamic Websites: The Websites provide the functionalities that can use to store, update, retrieve, and delete the data in a database.

What is the Form?

A form is a document that:

- Contains blank fields.
- The user can fill in the data or user can select the data and
- The data will be stored in the database, file, or session.
- GET and POST are the methods that are used to collect data from a form.

Now, it’s high time to apply the knowledge you have obtained so far and put it to real use. A very common application of PHP is to have an HTML form that will collect information from a website's visitor and then use PHP to process that information. In this lesson we will simulate a small business's website that is implementing a very simple order form.

Let's consider a scenario where an educational institution is hosting an event, and they require all attendees to complete a registration process. This involves gathering essential details about the participants, including their personal information, educational background, and contact details. To streamline the process of collecting orders or sign-ups from potential students or participants, a dedicated webpage with an HTML form will be created to efficiently collect the necessary information for the event.

1.3 Working with forms in PHP

We first create an HTML form that will collect students’ basic information, for example, Name, Email and Education. Once the user completes the form and clicks the 'submit' button, the entered data is transmitted for handling to a PHP file named "process_registration.php" via the action attribute. The form data is sent with the HTTP POST method.

The example below displays a simple HTML form with three input fields and a submit button:

```
<html>
<body>
<!–file save as form.html ->
<h1> HTML From demo in PHP </h1><hr>
    <form action="process_registration.php" method="post">
        Name: <input type="text" name="name"><br><br>
        E-mail: <input type="text" name="email"><br><br>
        Education:<select name="edu">
            <option selected>select Education</option>
            <option selected>B.Sc. Computer Science</option>
            <option selected>B.Sc. AI and Data Science</option>
            <option selected>B.Sc. Information Technology</option>
            <option selected>M.Sc. Computer Science</option>
            <option selected>M.Sc. Information Technology</option>
            </select><br><br>
        <input type="submit">
    </form>
</body>
</html>
```

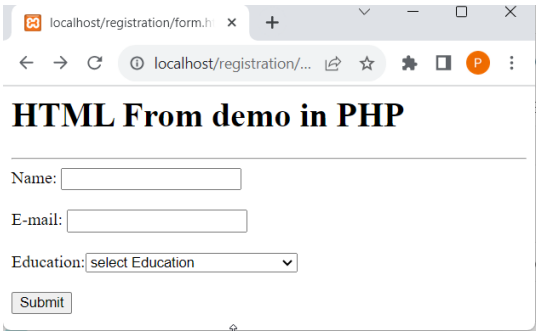

Figure 1: Registration Form

To display the submitted data, you could simply echo all the variables. The " process_registration.php " looks like this:

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

As you probably noticed, the name in $_POST['name'] corresponds to the name that we specified in our HTML form.

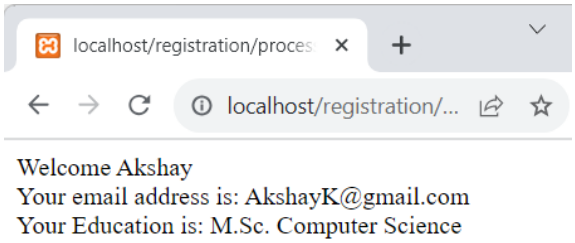

Figure 2: Output after processing the form

1.4 PHP Data Collection method

GET and POST are the methods that are used to collect data from a form. In the above example, the POST method was used.

1.4.1 The GET Method:

GET is used to request data from a specified resource.

The query string (name/value pairs) is sent in the URL of a GET request. For example, form_demo.php?name1=value1&name2=value2

- GET requests can be cached.
- GET requests remain in the browser history.
- GET requests can be bookmarked.
- GET requests should never be used when dealing with sensitive data.
- GET requests have length restrictions (maximum URL length is 2048 characters)
- GET requests is only used to request data (not modify)

1.4.2 The POST Method

Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request).

POST is used to send data to a server to create/update a resource.

The data sent to the server with POST is stored in the request body of the HTTP request.

- POST is one of the most common HTTP methods for parsing large amount of data.
- POST requests are never cached.
- POST requests do not remain in the browser history.
- POST requests cannot be bookmarked.
- POST requests have no restrictions on data length.

Moreover, POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

1.4.3 Comparing GET with POST

The following table compares the two HTTP methods: GET and POST.

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL. Never use GET when sending passwords or other sensitive information.	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

1.5 Validating Input Data

1.5.1 What is Validation?

Validation means checking the input submitted by the user to ensure the accuracy and security of the data. There are two types of validation in PHP:

- ***Client-Side Validation***: Validation is performed on the client machine web browsers.
- ***Server-Side Validation***: After submitting the form data, the latter is sent to a server and validation checks are perform in server machine.

Here are the key differences between them:

	Client-side Validation	Server-side Validation
Location of Validation	This validation occurs on the user's device or web browser. It's implemented using front-end technologies like HTML, CSS, and JavaScript.	This validation occurs on the server where the website or application is hosted. It's implemented using back-end technologies like PHP, Python, Java, etc.
Purpose	Mainly used for enhancing user experience by providing immediate feedback to users without needing to send data to the server.	Primarily used for security and ensuring data integrity. It's the last line of defense against malicious or incorrect data
Security	Can be bypassed by technically savvy users. Therefore, it should always be accompanied by server-side validation to ensure data integrity.	Provides a more secure method of validating data since it occurs on the server and cannot be manipulated by the client.
Response Time	Provides immediate feedback to users, often in real-time as they fill out forms.	Response time depends on the network speed and server load. It typically involves a round trip to the server.
Resource Usage	Utilizes the client's device resources (CPU, memory, etc.) and does not require server resources.	Relies on server resources. If not properly optimized, it can lead to higher server loads.
Languages and Technologies	Implemented using HTML, CSS, and JavaScript/jQuery.	Implemented using server-side programming languages like PHP, Python, Ruby, Java, etc., along with appropriate frameworks.
Fallback Mechanism	If JavaScript is disabled on the client's browser, client-side validation won't work. It's important to have server-side	Functions independently of client-side technologies. It works as long as the server is running.

	validation as a fallback.	
User Experience	Enhances user experience by providing instant feedback, reducing the need for page reloads.	Requires a page reload after submission to process and display validation results.
Complex Validation Rules	Suited for simple validations like checking required fields or email formats.	Capable of handling more complex business logic and database interactions.

You have already done client-side validation (JavaScript, etc.) in the module Client-Side Web Technologies. For this module, we will focus on server-side validation, more specifically, the required fields.

Required field will check whether the field is filled or not in the proper way. Most of cases we will use the * symbol for required field.

1.5.2 Regular Expressions (Regex) in PHP

What is Regular expression in PHP?

PHP Regular Expression also known as regex are powerful ***pattern matching algorithm*** that can be performed in a single expression. Regular expressions use arithmetic operators such as (+,-,^) to create complex expressions. They can help you accomplish tasks such as validating email addresses, IP address etc.

Why use regular expressions

- PHP Regular expressions simplify identifying patterns in string data by calling a single function. This saves us coding time.
- When validating user input such as email address, domain names, telephone numbers, IP addresses,
- Highlighting keywords in search results
- When creating a custom HTML template. Regex in PHP can be used to identify the template tags and replace them with actual data.

Built-in Regular expression Functions in PHP

PHP has built in functions that allow us to work with regular functions. Let's look at the commonly used regular expression functions in PHP.

- preg_match() – this function is used to perform pattern matching in PHP on a string. It

returns true if a match is found and false if a match is not found.

- preg_split() – this function is used to perform a pattern match on a string and then split the results into a numeric array
- preg_replace()– this function is used to perform a pattern match on a string and then replace the match with the specified text.

Syntax:

```
<?php
function_name('/pattern/',subject);
?>
```

Explanations:

- "function_name(…)" is either PHP preg_match(), PHP preg_split() or PHP preg_replace().
- "/…/" The ***forward slashes*** denote the beginning and end of our PHP regex tester function.
- " '/pattern/' " is the pattern that we need to matched
- "subject" is the text string to be matched against

Examples:

Example 1: PHP Preg_match()

The first example uses the preg_match() in PHP function to perform a simple pattern match for the word "**open**" in a given URL. The code below shows the implementation for preg_match() tester function for the above example.

```php
<?php
$my_url = "www.openuniversity.ac.mu";
if (preg_match("/open/", $my_url))
{
    echo "the url $my_url contains open";
}
else
{
    echo "the url $my_url does not contain open";
}
?>
```

The output is as follows:

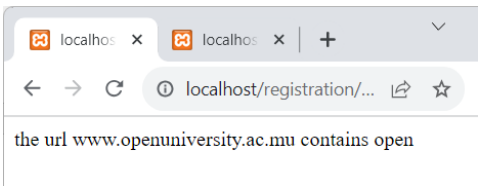

Figure 3: Preg_match output

Example 2: PHP Preg_split()

We will take a string phrase and explode it into an array; the pattern to be matched is a single space. The text string to be used in this example is “I Love Open University of Mauritius”. The code below illustrates the implementation of the above example.

```
<?php
$my_text="I Love Open University of Mauritius";
$my_array = preg_split("/ /", $my_text);
print_r($my_array );
?>
```

The output is as follows:

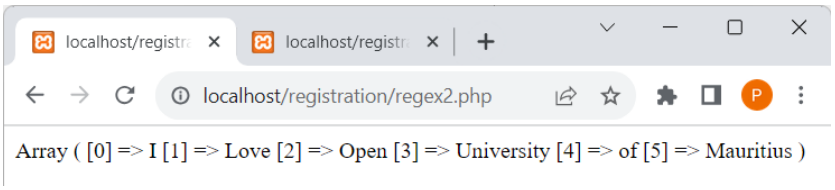

Figure 4: Preg_split output

Example 3: PHP Preg_replace()

The preg_replace() in PHP function performs a pattern match and then replaces the pattern with something else. The code below searches for the word Open in a string. It replaces the word Open with the word Open surrounded by CSS code that highlights the background colour.

```
<?php
$text = "We at Open University strive to make quality education affordable to
the masses. https://www.open.ac.mu/";
$text = preg_replace("/open/", '<span style="background:yellow">Open</span>',
$text);
echo $text;
?>
```

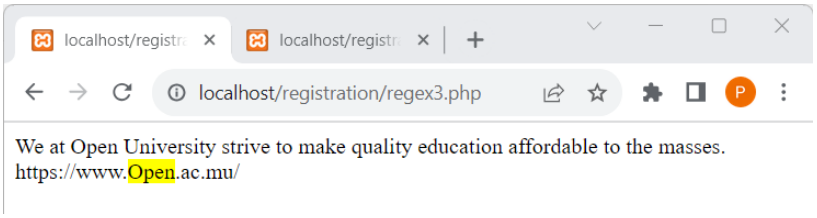

Figure 5: Preg_replace output.

Note on the Preg_replace(): In the above code, the word "open" is used twice. However, only in the URL that the word "open" has been replaced and highlighted. This is to show that the function is case sensitive. The function `preg_replace("/open/"…)` will match only the word `"/open/"` (same case) within the string.

Regular Expression Metacharacters

The above examples used very basic patterns; metacharacters simply allow us to perform more complex pattern matches such as test the validity of an email address. Let's now look at the ***commonly used*** metacharacters**.**

Metacharacter	Description	Example
.	Matches any single character except a new line	/./ matches anything that has a single character
^	Matches the beginning of or string / excludes characters	/^PH/ matches any string that starts with PH
$	Matches pattern at the end of the string	/com$/ matches gmail.com, yahoo.com Etc.
*	Matches any zero (0) or more characters	/com*/ matches computer, communication etc.
+	Requires preceding character(s) appear at least once	/yah+oo/ matches yahoo
\	Used to escape meta characters	/yahoo+\.com/ treats the dot as a literal value
[…]	Character class	/[abc]/ matches abc
a-z	Matches lower case letters	/a-z/ matches cool, happy etc.
A-Z	Matches upper case letters	/A-Z/ matches WHAT, HOW, WHY etc.
0-9	Matches any number between 0 and 9	/0-4/ matches 0,1,2,3,4

Let's now look at a fairly complex example that checks the validity of an email address.

```
<?php
$my_email = "name@company.com";
if (preg_match("/^[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+\.[a-zA-Z.]{2,5}$/", $my_email))
{
      echo "$my_email is a valid email address";
}
else
{
```

```
        echo "$my_email is NOT a valid email address";
}
?>
```

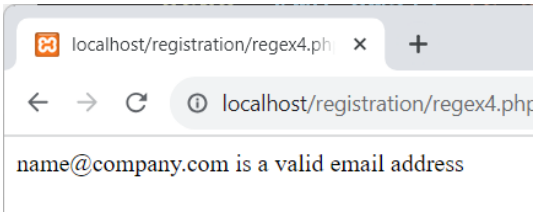

Figure 6: E-mail check output

Explanation of the pattern "[/^[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+\.[a-zA-Z.]{2,5}$/]"
HERE,

- "'/…/'" starts and ends the regular expression
- "^[a-zA-Z0-9._-]" matches any lower or upper case letters, numbers between 0 and 9 and dots, underscores or dashes.
- "+@[a-zA-Z0-9-]" matches the @ symbol followed by lower or upper case letters, numbers between 0 and 9 or dashes.
- "+\.[a-zA-Z.]{2,5}$/" escapes the dot using the backslash then matches any lower or upper case letters with a character length between 2 and 5 at the end of the string.
- $ Matches pattern at the end of the string.

Sanitizing email user inputs

The above example uses hard coded values in the source code for the email address and other details for simplicity. Let's assume you have to create a contact us form for users fill in the details and then submit.

Users can accidently or intentionally inject code in the headers which can result in sending spam mail. To protect your system from such attacks, you can create a custom function that sanitizes and validates the values before the mail is sent.

Filter_var() Function

The filter_var function is used to sanitize and validate the user input data. It has the following basic syntax:

```
<?php
        filter_var($field, SANITIZATION TYPE);
?>
```

HERE,

- "filter_var(…)" is the validation and sanitization function.
- "$field" is the value of the field to be filtered.
- "SANITIZATION TYPE" is the type of sanitization to be performed on the field such as;
    - FILTER_VALIDATE_EMAIL – it returns true for valid email addresses and false for invalid email addresses.
    - FILTER_SANITIZE_EMAIL – it removes illegal characters from email addresses. info\@domain.(com) returns info@domain.com.
    - FILTER_SANITIZE_URL – it removes illegal characters from URLs. http://www.example@.comé returns >http://www.example@.com
    - FILTER_SANITIZE_STRING – it removes tags from string values. am bold becomes am bold.

Example of validating an e-mail using FILTER_VALIDATE_EMAIL: Check if the variable $email is a valid email address.

```php
<?php
$email = "john.doe@example.com";

if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
      echo("$email is a valid email address");
} else {
      echo("$email is not a valid email address");
}
?>
```

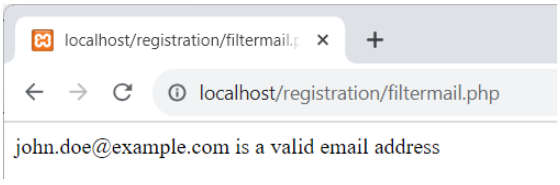

Figure 7: Email input check

First remove all illegal characters from the $email variable, then check if it is a valid email address:

```php
<?php
$email = "john.doe@example.com";

// Remove all illegal characters from email
$email = filter_var($email, FILTER_SANITIZE_EMAIL);
```

```php
// Validate e-mail
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
?>
```

The code below implements uses a custom function to send secure mail.

```php
<?php
function sanitize_my_email($field) {
    $field = filter_var($field, FILTER_SANITIZE_EMAIL);
    if (filter_var($field, FILTER_VALIDATE_EMAIL)) {
        return true;
    } else {
        return false;
    }
}
$to_email = 'name @ company . com';
$subject = 'Testing PHP Mail';
$message = 'This mail is sent using the PHP mail ';
$headers = 'From: noreply @ company. com';
//check if the email address is invalid $secure_check
$secure_check = sanitize_my_email($to_email);
if ($secure_check == false) {
    echo "Invalid input";
} else { //send email
    mail($to_email, $subject, $message, $headers);
    echo "This email is sent using PHP Mail";
}
?>
```

The output is as follows:

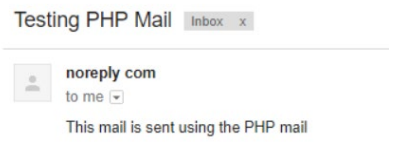

Figure 8: Complete email check

1.5.3 Some of Validation rules for fields

Field	Validation Rules
Name	Should require letters and white-spaces.
Email	Should require @ and .
Website	Should require a valid URL.

Radio	Must be selectable at least once.
Check Box	Must be checkable at least once.
Drop Down menu	Must be selectable at least once.

Validate 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 an error message is sent to the user:

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

Valid URL

Below code shows validation of URL:

```
$website = input($_POST["site"]);
if(!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
        $websiteErr = "Invalid URL";
}
```

Above syntax will verify whether a given URL is valid or not. It should allow some keywords as https, ftp, www, a-z, 0-9,..etc..

FILTER_VALIDATE_URL function:

The best option is to use the FILTER_VALIDATE_URL filter to validate a URL.

Example:

Check if the variable $url is a valid URL:

```
<?php
```

```php
$url = "https://open.ac.mu";

if (filter_var($url, FILTER_VALIDATE_URL)) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
?>
```

The Output is: https://open.ac.mu is a valid URL.

Suppose that I put an invalid URL, as follows:

```php
<?php
$url = "https://open .ac. mu";

if (filter_var($url, FILTER_VALIDATE_URL)) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
?>
```

The output is as follows:

https://open .ac. mu is not a valid URL. Therefore, we need to remove illegal characters from the URL. The following code will first remove all illegal characters from the $url variable, then check if it is a valid URL

```php
<?php
$url = "https://open .ac. mu";

// Remove all illegal characters from a url
$url = filter_var($url, FILTER_SANITIZE_URL);

// Validate url
if (filter_var($url, FILTER_VALIDATE_URL)) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
?>
```

Form fields validation

Let's discuss the example of **form fields validation**:

```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"]);
        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"]);
    // Remove all illegal characters from email
    $email = filter_var($email, FILTER_SANITIZE_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"]);
    }

    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>Absolute classes registration</h2>
<p><span class = "error">* required field.</span></p>
<form method = "post" action = "<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
    <table>
        <tr>
            <td>Name:</td>
            <td><input type = "text" name = "name">
                <span class = "error">* <?php echo $nameErr;?></span>
            </td>
        </tr>
        <tr>
            <td>E-mail: </td>
            <td><input type = "text" name = "email">
                <span class = "error">* <?php echo $emailErr;?></span>
            </td>
        </tr>
        <tr>
            <td>Website:</td>
            <td><input type = "text" name = "website">
                <span class = "error"><?php echo $websiteErr;?></span>
            </td>
        </tr>
        <tr>
            <td>About you:</td>
            <td><textarea name = "comment" rows = "5" cols =
"40"></textarea></td>
        </tr>
        <tr>
            <td>Gender:</td>
            <td><input type = "radio" name = "gender" value = "Female">Female
                <input type = "radio" name = "gender" value = "Male">Male
                <input type = "radio" name = "gender" value = "Others">others
                    <span class = "error">* <?php echo $genderErr;?></span>
            </td>
        </tr>
        <tr>
            <td><input type = "submit" name = "submit" value = "Submit">
            </td>
        </tr>
    </table>
</form>
```

```php
<?php
    echo "<h2>Your entered values are:</h2>";
    echo $name;
    echo "<br>";
    echo $email;
    echo "<br>";
    echo $website;
    echo "<br>";
    echo $comment;
    echo "<br>";
    echo $gender;
?>
</body>
</html>
```

Output of the form (before data input)

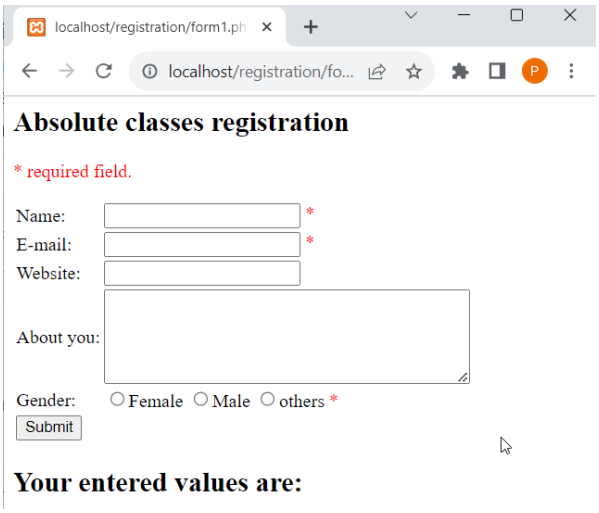

Figure 9: Output of form (When access first time)

When the user clicks on submit button without entering data, it will show the * marks and will inform the user that the field is required.

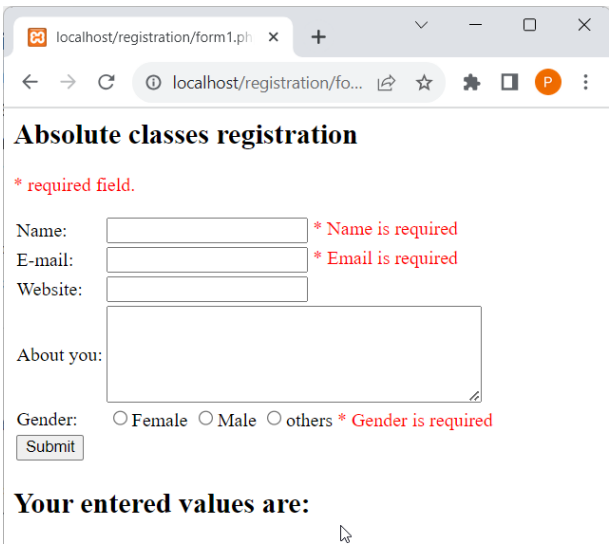

Figure 10: Output when the user clicks on submit button without entering required fields.

Finally, the output will be as follows when the user enter some sample data:

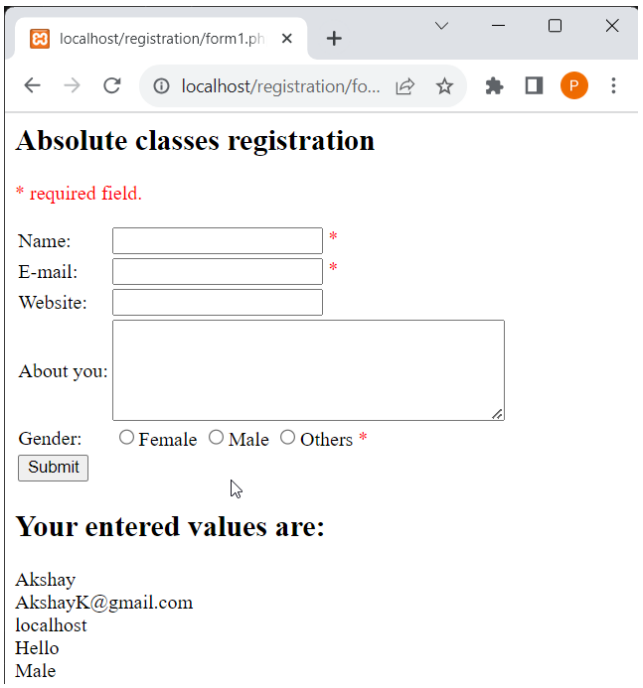

Figure 11: Output when the user clicks on submit button after entering all fields.

Note: Additional validation rules were also applied in the above code such as:

- For name, only letters and white space were allowed.
- Invalid characters such as whitespace is removed to validate the email.

1.6 Storing Form Data in File

In this topic, you will learn how to store form data in a text file or row file. Sometimes it happens that we need to store some data in a local storage file rather than making it complex using the database.

Here is an example, suppose you have an HTML form, and you want to store the data submitted by the user in a text file so that you can easily access it later from that file without opening your database.

1.6.1 PHP Program to store HTML Form data in a .txt file.

Here is a sample code to store the form data in a text file.

```
<!DOCTYPE html>
<html>
<head>
    <title>Store form data in .txt file</title>
</head>
<body>
<form method="post" action = "<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">

Enter Your Text Here:<br>
<textarea name="textdata" rows="4" cols="50"></textarea><br>
<input type="submit" name="submit">
</form>
</body>
</html>

<?php
if(isset($_POST['textdata'])) {
    $data=$_POST['textdata'];
    $fp = fopen('data.txt', 'a');
    fwrite($fp, $data);
    fclose($fp);
}
?>
```

Here ‘textdata’ is the name of the HTML form text field. “data.txt” is the file that will be created for storing the form submission data. “$data” is a PHP variable used to store the form field data entered by the user.

1.7 Saving Form Data using Cookies

Cookies are text files stored on the client computer and they are used for tracking purposes. PHP transparently supports HTTP cookies.

1.7.1 How cookies work.

There are three steps involved in identifying returning users:

- Server script **sends a set of cookies to the browser**. For example: name, age, or identification number etc.
- The **browser stores this information on local machine** (client computer) for future use.
- The next time when the browser sends any request to web server, it also sends those cookies information to the server. The server uses that information to identify the user.

A cookie is a small file with a **maximum size of 4KB** that the web server stores on the client computer. Once a cookie has been set, all page requests that follow return the cookie name and value. A cookie can only be read from the domain that it has been issued from. For example, a cookie set using the domain open.ac.mu cannot be read from the domain support.open.ac.mu.

Most of the websites on the internet display elements from other domains such as advertising. The domains serving these elements can also set their own cookies. These are known as third party cookies. A cookie created by a user can only be visible to them. Other users cannot see its value. Most web browsers have options for disabling cookies, third party cookies or both. If this is the case, then PHP responds by passing the cookie token in the URL.

Cookies are created at server side and saved to client browser. Each time when client sends request to the server, cookie is embedded with request. Such way, cookie can be received at the server side.

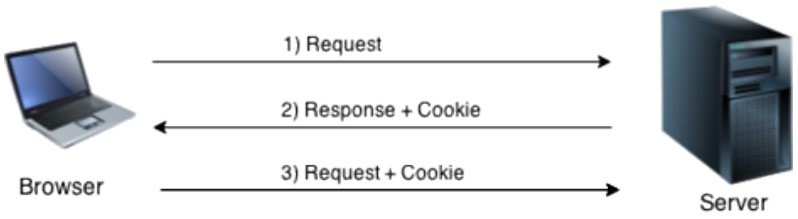

Figure 12: Cookie client-server interaction

In short, cookies can be created, sent, and received at the server end. In this chapter you will learn,

- How to set cookies?
- How to access them?
- how to delete or remove them.

1.7.1 The Anatomy of a Cookie

Cookies are usually **set in the HTTP header** (although JavaScript can also set a cookie directly on a browser). A PHP script that sets a cookie might send headers that look something like this:

HTTP/1.1 200 OK
Date: Thu, 26 Oct 2023 21:03:38 GMT
Server: Apache/2.4 (UNIX) PHP/8.2.4
Set-Cookie: name=xyz; expires=Wednesday, 27-DEC-2023:03:38 GMT; path=/; domain=oum.ac.mu
Connection: close Content-Type: text/html

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this:

GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
*Host: test.demon.co.in:1126 Accept: image/gif, */* Accept-Encoding: gzip Accept-Language: en*
Accept-Charset: iso-8859-1,,utf-8 Cookie: name=xyz*

A PHP script will then have access to the cookie in the environmental variables $_COOKIE or $HTTP_COOKIE_VARS[] which holds all cookie names and values. The above cookie can be accessed using $HTTP_COOKIE_VARS["name"].

1.7.2 Why and when to use Cookies?

- Http is a stateless protocol;
  - cookies allow us to track the state of the application using small files stored on the user's computer.
- The path where the cookies are stored depends on the browser.
  - Internet Explorer usually stores them in Temporary Internet Files folder.
- Personalizing the user experience – this is achieved by allowing users to select their preferences.
  - The page requested that follows are personalized based on the set preferences in the cookies.
- Tracking the pages visited by a user.

1.7.3 Setting Cookies with PHP

PHP provided setcookie() function to set a cookie. This function **should be called before <html> tag**. For each cookie this function has to be called separately.

1.7.4 Syntax

bool setcookie (string $name, string $value, int $expire = 0, string $path, string $domain, bool $secure = false, bool $httponly = false)

Example

- setcookie(name, value, expire, path, domain, security);
- setcookie("CookieName", "CookieValue"); /* defining name and value only*/
- setcookie("CookieName", "CookieValue", time()+1*60*60); //using expiry in 1 hour (1*60*60 seconds or 3600 seconds)
- setcookie("CookieName", "CookieValue", time()+1*60*60, "/mypath/", "mydomain.com", 1);

Here are the details of all the arguments description:

- ***Name:*** This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- ***Value:*** This sets the value of the named variable and is the content that you want to store.
- ***Expiry:*** This specifies a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time, cookies will become inaccessible. If this parameter is not set, then the cookie will

automatically expire when the Web Browser is closed.

- ***Path:*** This specifies the directories for which the cookie is valid. A single forward slash (/) character permits the cookie to be valid for all directories.
- ***Domain:*** This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- ***Security:*** This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which means the cookie can be sent by regular HTTP.

The following example will create two cookies: name and age. These cookies will expire in one hour.

```php
<?php
setcookie("name", "Open University Visit", time()+3600, "/","", 0);
setcookie("age", "25", time()+3600, "/", "", 0);
?>
<html>
<head>
<title>Setting Cookies with PHP</title>
</head>
<body>
<?php echo "Set Cookies"?>
</body>
</html>
```

Output:

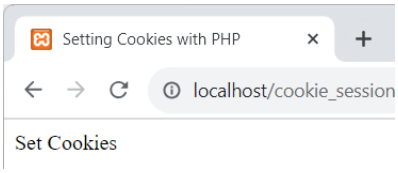

Figure 13: output of set_cookie file

1.7.5 Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. The following example will access all the cookies set in above example.

```php
<?php
echo "Cookie set value: ".$_COOKIE["name"]. "<br />";
/* is equivalent to */
//echo $HTTP_COOKIE_VARS["name"]. "<br />";
echo "Cookie Set value: ".$_COOKIE["age"] . "<br />";
/* is equivalent to */
//echo $HTTP_COOKIE_VARS["age"] . "<br />";
?>
<hr><b>You can use isset() function to check if a cookie is set or not.</b><hr>
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
if( isset($_COOKIE["name"]))
echo "<br>Welcome " . $_COOKIE["name"] . "<br />";
else
echo "<br>Sorry... Not recognized" . "<br />";
?>
</body>
</html>
```

The output is as follows:

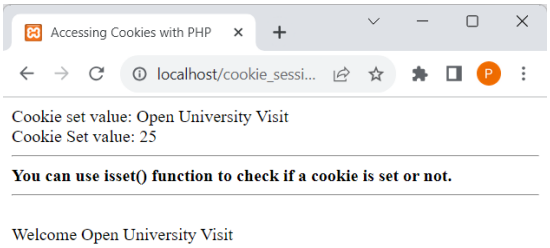

Figure 14: Output of set cookie

1.7.6 Deleting or removing Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired.

```php
<?php
setcookie( "name", "", time()- 60, "/","", 0);
setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
<head>
<title>Deleting Cookies with PHP</title>
</head>
<body>
<?php echo "Deleted Cookies" ?>
</body>
</html>
```

The output will be as follows:

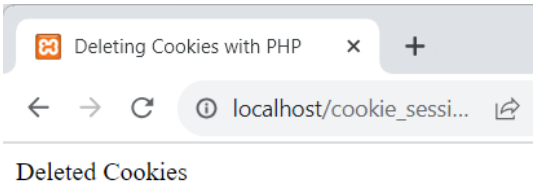

Figure15: Output of delete cookies.

Note: You should pass exactly the same path, domain, and other arguments that you have used when you first created the cookie in order to ensure that the correct cookie is deleted.

1.8 Saving Form Data Using Sessions & Tracking

1.8.1 What is a PHP Session?

When you work with an application, you open it, make some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet, there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g., username, favorite color, etc.). By default, session variables last until the user closes the browser.

Session variables hold information about one single user, and are available to all pages in one application.

- A session is a global variable stored on the server.
- Each session is assigned a unique id which is used to retrieve stored values.
- Whenever a session is created, a cookie containing the unique session id is stored on the user's computer and returned with every request to the server. If the client browser does not support cookies, the unique PHP session id is displayed in the URL.
- Sessions have the capacity to store relatively large data compared to cookies.
- The session values are automatically deleted when the browser is closed. If you want to store the values permanently, then you should store them in the database.
- Just like the $_COOKIE array variable, session variables are stored in the $_SESSION array variable. Just like cookies, the session must be started before any HTML tags.

Note: A session is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the user's computer.

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the php.ini file called session.save_path. Before using any session, variable make sure you have setup this path.

When a session is started following things happen:

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called PHPSESSID is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess.
  i.e. sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

1.8.2 Why and when to use Sessions?

- You want to store important information such as the user id more securely on the server where malicious users cannot temper with them.
- You want to pass values from one page to another.
- You want the alternative to cookies on browsers that do not support cookies.
- You want to store global variables in an efficient and more secure way compared to passing them in the URL.
- You are developing an application such as a shopping cart that has to temporary store information with a capacity larger than 4KB.

1.8.3 Start a PHP Session

A session is started with the session_start() function. Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new PHP page and start a new PHP session and set some session variables.

Start session Example:

```php
<?php
session_start();// Start the session
?>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

Output:

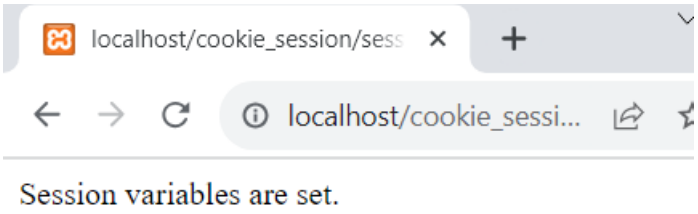

Figure 16: output of start_session

1.8.4 Get PHP Session Variable Values

Next, we create another page from which we will access the session information set on the previous page.

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable: Get PHP Session Variable

```php
<?php
session_start();
?>
<html>
<body>
<?php
// Echo session variables that were set on previous page
echo "Favoritecolor is " . $_SESSION["favcolor"] .".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".<br>";
// or Print all session variable as an array
echo "Print all session variable as an array <br>";
print_r($_SESSION);
?>
</body>
</html>
```

Output is as follows:

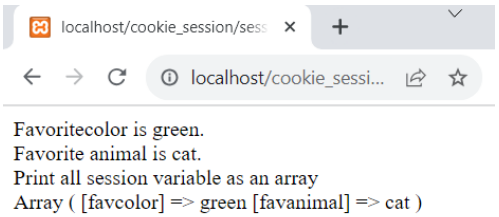

Figure 17: output session variable Individually and as an Array

1.8.5 Destroying Session Variables

The session_destroy() function is used to destroy the whole PHP session variables. If you want to destroy only a session single item, you use the unset() function.

The code below illustrates how to use both methods.

```php
<?php
session_destroy(); //destroy entire session
?>

<?php
unset($_SESSION['product']); //destroy product session item
?>
```

Session_destroy removes all the session data including cookies associated with the session. Unset only frees the individual session variables. Other data remains intact. Every PHP session has a timeout value — a duration, measured in seconds — which determines how long a session should remain alive in the absence of any user activity. You can adjust this timeout duration by changing the value of session.gc_maxlifetime variable in the PHP configuration file (php.ini).