Experiment No. 3

Title: Form Handling using PHP

Batch: B-3 **Roll No: 16010422234** **Experiment No.:3**

Aim: Form Handling using PHP

Resources needed: XAMPP, VS Code text editor, Web browser

Theory:

PHP Registration Form using GET, POST Methods with Example

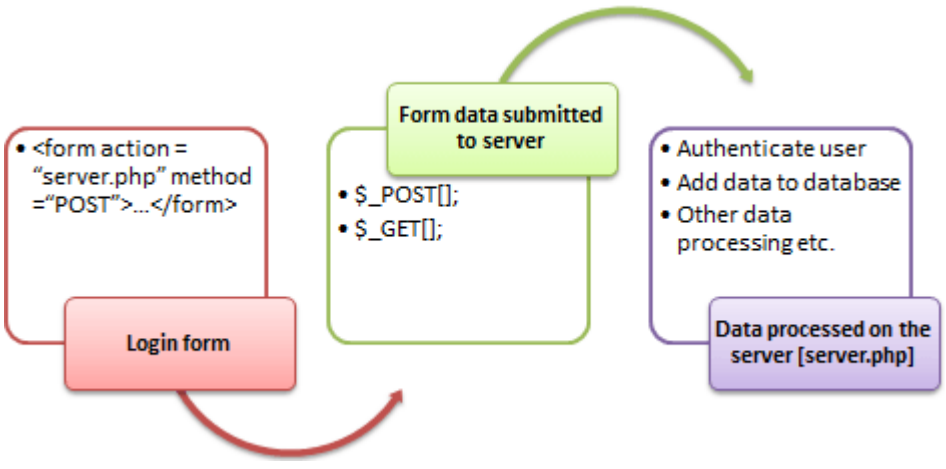
A form is an HTML tag that contains graphical user interface items such as input box, check boxes, radio buttons etc.

When you login into a website or into your mail box, you are interacting with a form.

What is Form?

Forms are used to get input from the user and submit it to the web server for processing.

The diagram below illustrates the form handling process.



The form is defined using the <form>...</form> tags and GUI items are defined using form elements such as input.

When and why are we using forms?

- Forms come in handy when developing flexible and dynamic applications that accept user input.
- Forms can be used to edit already existing data from the database.

Create a form

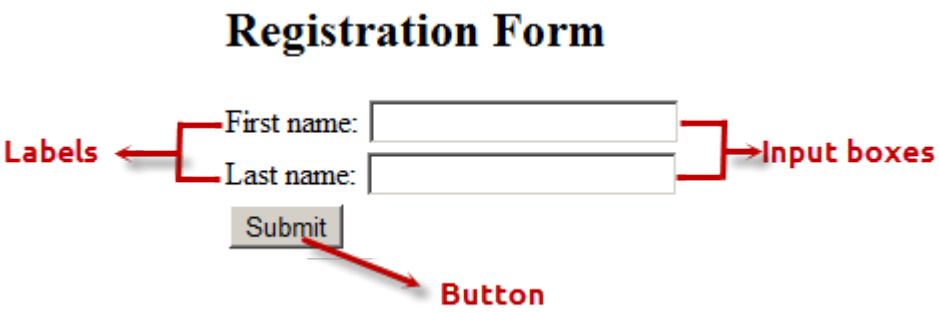
We will use HTML tags to create a form. Below is the minimal list of things you need to create a form.

- Opening and closing form tags <form>…</form>
- Form submission type POST or GET
- Submission URL that will process the submitted data
- Input fields such as input boxes, text areas, buttons,checkboxes etc.

The code below creates a simple registration form

```
<html>
<head>
        <title>Registration Form</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>

    <h2>Registration Form</h2>

    <form action="registration_form.php" method="POST"> First name:

        <input type="text" name="firstname"> <br> Last name:

        <input type="text" name="lastname">

        <input type="hidden" name="form_submitted" value="1" />

        <input type="submit" value="Submit">


    </form>
</body>
</html>
```

Viewing the above code in a web browser displays the following form.



HERE,

- <form…>…</form> are the opening and closing form tags
- action="registration_form.php" method="POST"> specifies the destination URL and the submission type.
- First/Last name: are labels for the input boxes
- <input type=”text”…> are input box tags
-
 is the new line tag
- <input type="hidden" name="form_submitted" value="1"/> is a hidden value that is used to check whether the form has been submitted or not
- <input type="submit" value="Submit"> is the button that when clicked submits the form to the server for processing

Submitting the form data to the server

The action attribute of the form specifies the submission URL that processes the data. The method attribute specifies the submission type.

PHP POST method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP POST method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method is ideal when you do not want to display the form post values in the URL.
- A good example of using post method is when submitting login details to the server.

It has the following syntax.

```
<?php
 $_POST['variable_name'];
?>
```

HERE,

- "$_POST[…]" is the PHP array
- "'variable_name'" is the URL variable name.

PHP GET method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP GET method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method displays the form values in the URL.
- It's ideal for search engine forms as it allows the users to book mark the results.

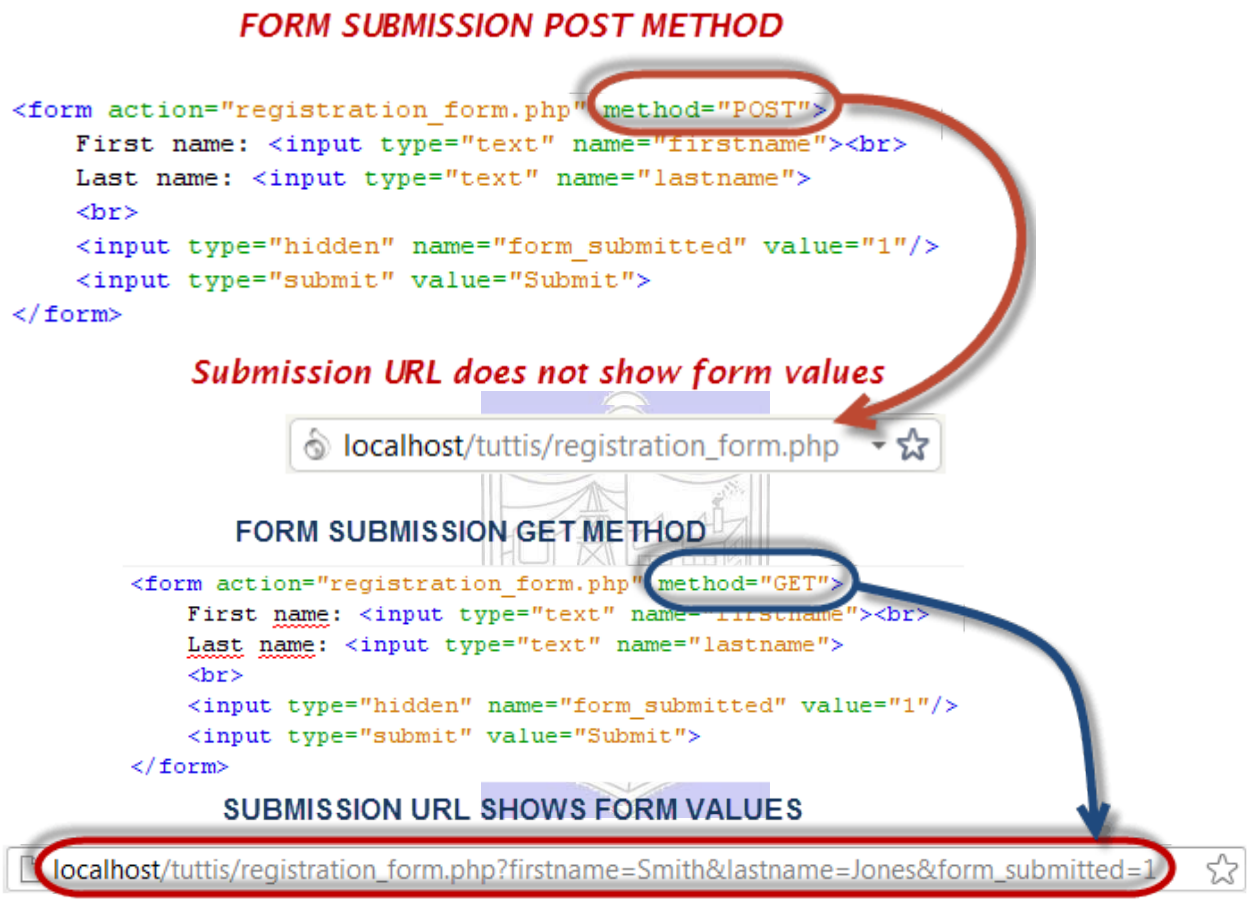
It has the following syntax.

```
<?php
$_GET['variable_name'];
?>
```

HERE,

- "$_GET[…]" is the PHP array
- "'variable_name'" is the URL variable name.

The below diagram shows the difference between get and post



Processing the registration form data

The registration form submits data to itself as specified in the action attribute of the form.

When a form has been submitted, the values are populated in the
$_POST super global array.

We will use the PHP isset function to check if the form values have been filled in the $_POST array and process the data.

We will modify the registration form to include the PHP code that processes the data. Below is the modified code:

```
<html>
<head>
        <title>Registration Form</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

</head>
<body>

    <?php if (isset($_POST['form_submitted'])): ?> //this code is executed when
the f orm is submitted

        <h2>Thank You <?php echo $_POST['firstname']; ?> </h2>

        <p>You have been registered as
            <?php echo $_POST['firstname'] . ' ' . $_POST['lastname']; ?>
        </p>

        <p>Go <a href="/registration_form.php">back</a> to the form</p>

        <?php else: ?>

            <h2>Registration Form</h2>
            <form action="registration_form.php" method="POST">
                 First name:
                <input type="text" name="firstname">

                <br> Last name:
                <input type="text" name="lastname">

                         <input type="hidden" name="form_submitted" value="1" />

                <input type="submit" value="Submit">

            </form>
```

```
      <?php endif; ? >
</body>
</html>
```

HERE,

- <?php if (isset($_POST['form_submitted'])): ?> checks if the form_submitted hidden field has been filled in the $_POST[] array and display a thank you and first name message.

If the form_fobmitted field hasn't been filled in the $_POST[] array, the form is displayed.

Working with checkboxes, radio buttons

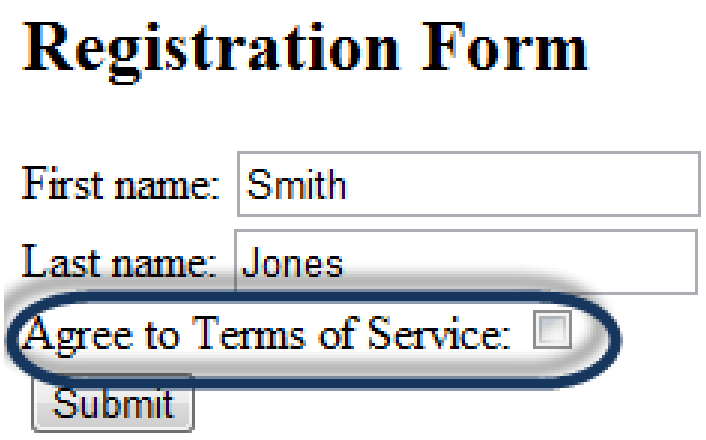
If the user does not select a check box or radio button, no value is submitted, if the user selects a check box or radio button, the value one (1) or true is submitted.

We will modify the registration form code and include a check button that allows the user to agree to the terms of service.

```
<html>
<head>
        <title>Registration Form</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>

    <?php if (isset($_POST['form_submitted'])):
                        ?>

        <?php if (!isset($_POST['agree'])):
                        ?>

            <p>You have not accepted our terms of service</p>

            <?php else: ?>

                <h2>Thank You <?php echo $_POST['firstname']; ?></h2>

                <p>You have been registered as
                    <?php echo $_POST['firstname'] . ' ' . $_POST['lastname']; ?>
                </p>
```

```
            <p> Go <a href="/registration_form2.php">back</a> to the form</p>

        <?php endif; ?>

        <?php else: ?>

                    <h2>Registration Form</h2>

                    <form action="registration_form2.php" method="POST">

                        First name:
                        <input type="text" name="firstname">

                        <br> Last name:
                        <input type="text" name="lastname">

                        <br> Agree to Terms of Service:
                        <input type="checkbox" name="agree">
                        <br>

                        <input type="hidden" name="form_submitted" value="1"
                        />

                        <input type="submit" value="Submit">

                    </form>
    <?php endif; ?>
</body>
</html>
```

View the above form in a browser



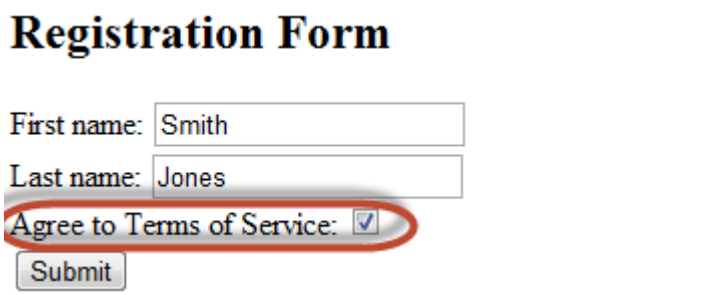
Fill in the first and last names

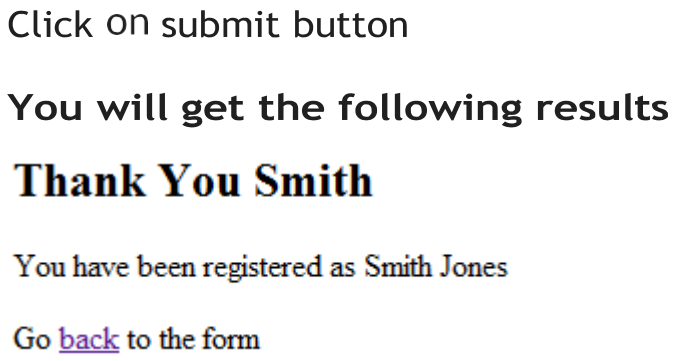
Note the Agree to Terms of Service checkbox has not been selected.

Click on submit button

You will get the following results

Click on back to the form link and then select the checkbox





Validate Name using expressions

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

Validate Email using filter method

```
$email = test_input($_POST["email"]);
```

```
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
$emailErr = "Invalid email format";
}
```

PHP Form Security

The $_SERVER["PHP_SELF"] variable can be used by hackers!

If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Consider

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

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

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

However, consider that a user enters the following URL in the address bar:

http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</scrip t>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can

hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How to avoid exploitation of PHP_SELF and hence avoid cross site scripting?

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

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

> The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:
>
> ```
> <form method="post"
> action="test_form.php/"><script>alert('h
> acked')</script>">
> ```
>
> The exploit attempt fails, and no harm is done!
>
> **Activity: Complete form validation including 2 or more validating criteria for each form input element. The script should include minimum 4 types of Form Input elements and their validation.**

Output:

```
<!DOCTYPE html>
<html>
<head>
    <title>Perfume Feedback Form</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            background-color: #f7ede2;
            color: #333;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            overflow: hidden;
```

```
        }
        .container {
            background: #fff;
            padding: 30px;
            border-radius: 15px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            max-width: 600px;
            width: 100%;
            overflow-y: auto;
            max-height: 90vh;
        }
        h2 {
            text-align: center;
            color: #f28482;
        }
        .error {
            color: #FF0000;
            font-size: 0.9em;
        }
        input[type=text], input[type=password], textarea {
            width: calc(100% - 22px);
            padding: 10px;
            margin: 5px 0 15px 0;
            border: 1px solid #ccc;
            border-radius: 5px;
            box-sizing: border-box;
        }
        input[type=checkbox] {
            margin-right: 10px;
        }
        input[type=submit] {
            background-color: #ffc566;
            color: white;
            border: none;
            padding: 10px 20px;
            border-radius: 5px;
            cursor: pointer;
            width: 100%;
        }
        input[type=submit]:hover {
            background-color: #f28482;
        }
```

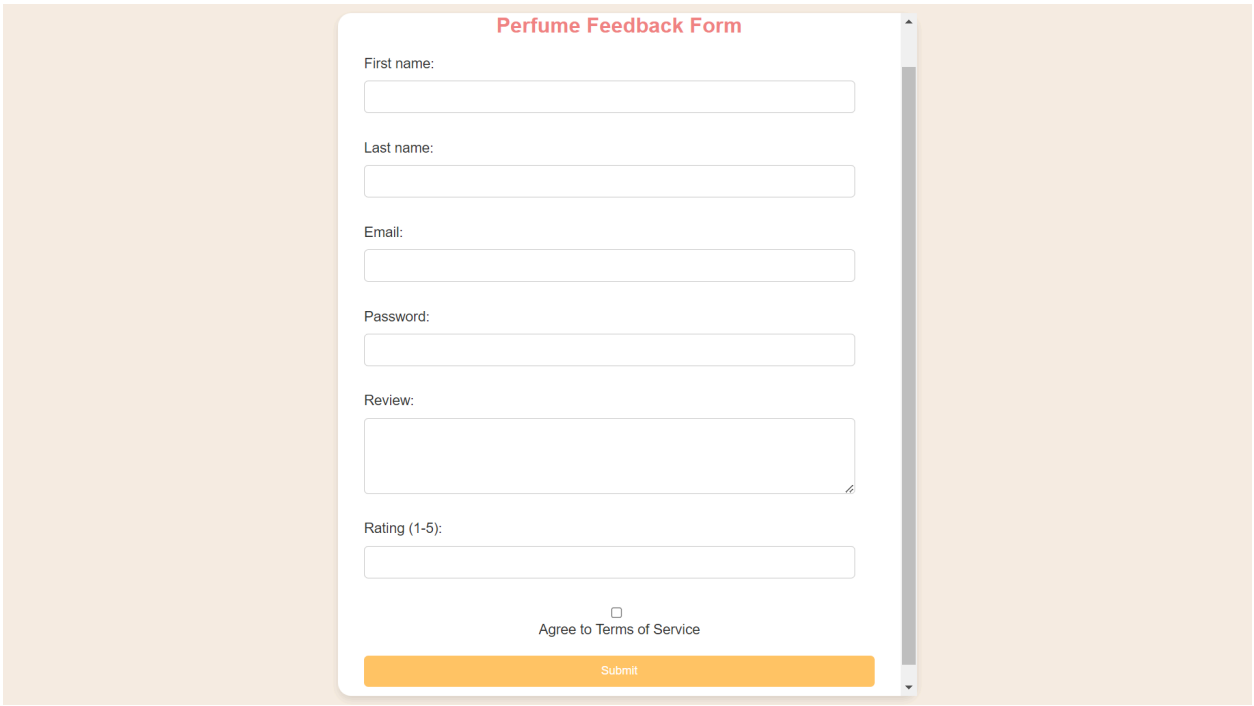
```
        .form-group {
            margin-bottom: 15px;
            display: flex;
            flex-direction: column;
        }
        .form-group label {
            margin-bottom: 5px;
        }
        .form-group .error {
            align-self: flex-end;
            margin-top: -10px;
            margin-bottom: 10px;
        }
        .form-inline {
            display: flex;
            align-items: center;
        }
        .form-inline label {
            margin: 0 10px 0 0;
        }
        .checkbox-group {
            display: flex;
            align-items: center;
        }
        .checkbox-group .error {
            margin-left: 10px;
        }
        .back-link {
            text-align: center;
            margin-top: 20px;
        }
        .back-link a {
            color: #85a79e;
            text-decoration: none;
        }
        .back-link a:hover {
            text-decoration: underline;
        }
    </style>
</head>
<body>
    <div class="container">
```

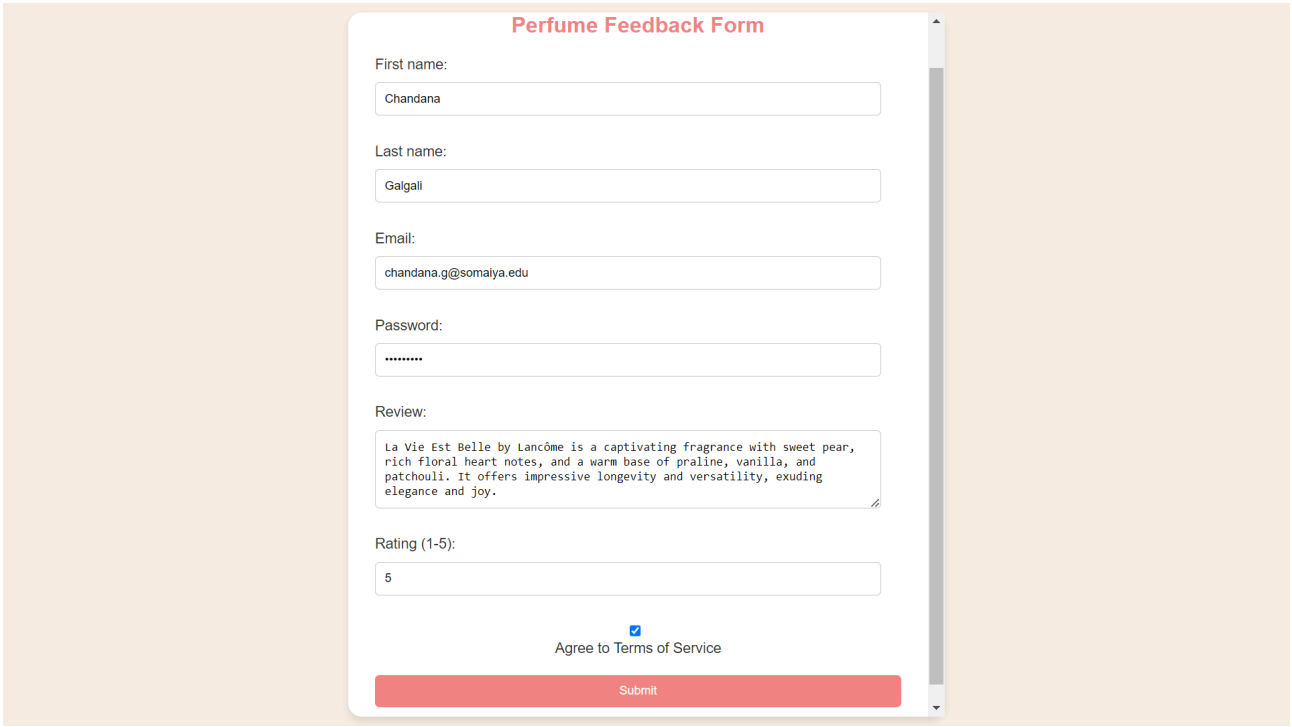
```
        <h2>Perfume Feedback Form</h2>
        <?php
        $firstNameErr = $lastNameErr = $emailErr = $passwordErr =
$agreeErr = $reviewErr = $ratingErr = "";
        $first_name = $last_name = $email = $password = $review = $rating
= "";

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

            if (empty($_POST["last_name"])) {
                $lastNameErr = "Last name is required";
            } else {
                $last_name = test_input($_POST["last_name"]);
                if (!preg_match("/^[a-zA-Z-' ]*$/", $last_name)) {
                    $lastNameErr = "Only letters and white space allowed";
                }
            }

            if (empty($_POST["email"])) {
                $emailErr = "Email is required";
            } else {
                $email = test_input($_POST["email"]);
                if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                    $emailErr = "Invalid email format";
                }
            }

            if (empty($_POST["password"])) {
                $passwordErr = "Password is required";
            } else {
                $password = test_input($_POST["password"]);
                if (strlen($password) < 6) {
                    $passwordErr = "Password must be at least 6 characters
```

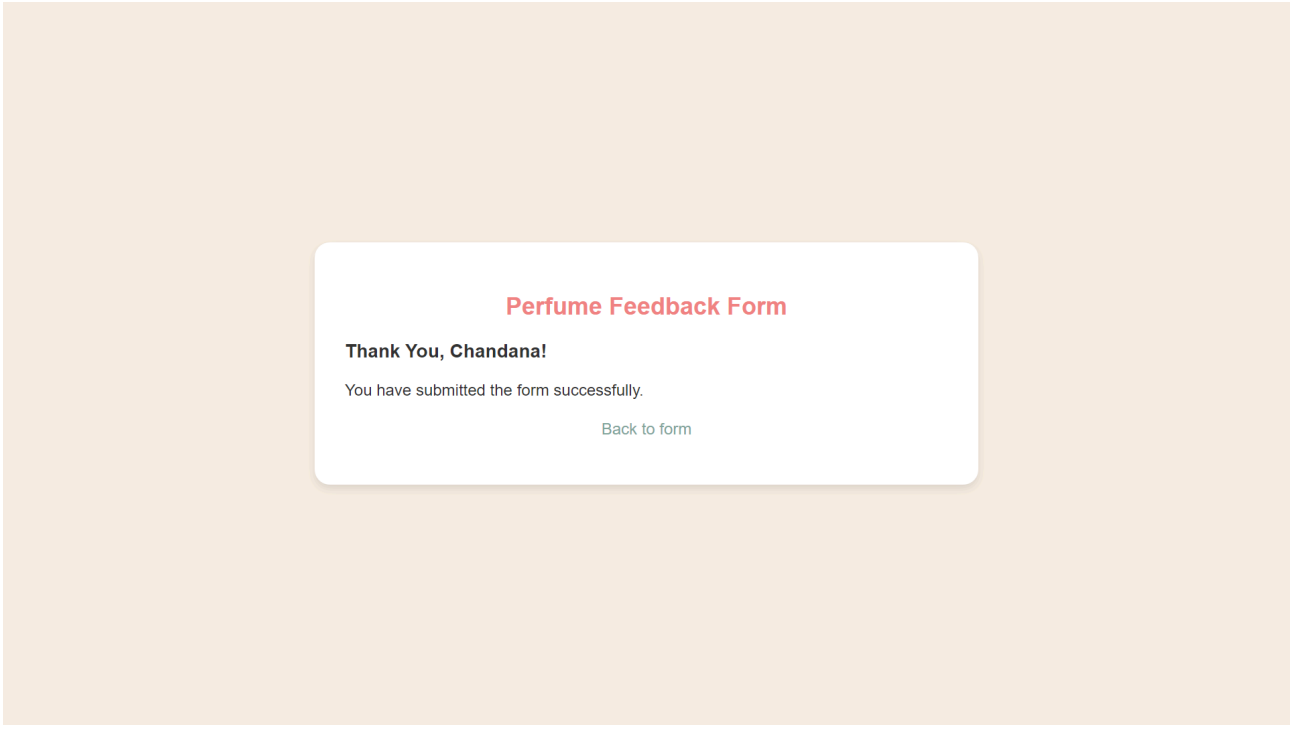
```
long";
                }
                if (!preg_match("/[A-Za-z]/", $password) ||
!preg_match("/[0-9]/", $password)) {
                    $passwordErr = "Password must contain both letters and
numbers";
                }
            }

            if (empty($_POST["review"])) {
                $reviewErr = "Review is required";
            } else {
                $review = test_input($_POST["review"]);
                if (strlen($review) < 20) {
                    $reviewErr = "Review must be at least 20 characters
long";
                }
            }

            if (empty($_POST["rating"])) {
                $ratingErr = "Rating is required";
            } else {
                $rating = test_input($_POST["rating"]);
                if (!is_numeric($rating) || $rating < 1 || $rating > 5) {
                    $ratingErr = "Rating must be a number between 1 and
5";
                }
            }

            if (!isset($_POST['agree'])) {
                $agreeErr = "You must agree to the terms of service";
            }
        }

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

```
        <?php if (isset($_POST['form_submitted']) && $firstNameErr == ""
&& $lastNameErr == "" && $emailErr == "" && $passwordErr == "" &&
$reviewErr == "" && $ratingErr == "" && $agreeErr == ""): ?>
            <h3>Thank You, <?php echo htmlspecialchars($first_name);
?>!</h3>
            <p>You have submitted the form successfully.</p>
            <p class="back-link"><a href="registration_form.php">Back to
form</a></p>
        <?php else: ?>
            <form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
                <div class="form-group">
                    <label for="first_name">First name:</label>
                    <input type="text" name="first_name" value="<?php echo
$first_name; ?>">
                    <span class="error"><?php echo $firstNameErr;
?></span>
                </div>
                <div class="form-group">
                    <label for="last_name">Last name:</label>
                    <input type="text" name="last_name" value="<?php echo
$last_name; ?>">
                    <span class="error"><?php echo $lastNameErr; ?></span>
                </div>
                <div class="form-group">
                    <label for="email">Email:</label>
                    <input type="text" name="email" value="<?php echo
$email; ?>">
                    <span class="error"><?php echo $emailErr; ?></span>
                </div>
                <div class="form-group">
                    <label for="password">Password:</label>
                    <input type="password" name="password" value="<?php
echo $password; ?>">
                    <span class="error"><?php echo $passwordErr; ?></span>
                </div>
                <div class="form-group">
                    <label for="review">Review:</label>
                    <textarea name="review" rows="4"><?php echo $review;
?></textarea>
                    <span class="error"><?php echo $reviewErr; ?></span>
                </div>
```

```
                <div class="form-group">
                    <label for="rating">Rating (1-5):</label>
                    <input type="text" name="rating" value="<?php echo
$rating; ?>">
                    <span class="error"><?php echo $ratingErr; ?></span>
                </div>
                <div class="form-group checkbox-group">
                    <input type="checkbox" name="agree" <?php
if(isset($_POST['agree'])) echo "checked"; ?>>
                    <label for="agree">Agree to Terms of Service</label>
                    <span class="error"><?php echo $agreeErr; ?></span>
                </div>
                <input type="hidden" name="form_submitted" value="1">
                <input type="submit" value="Submit">
            </form>
        <?php endif; ?>
    </div>
</body>
</html>
```







Outcomes: Design forms and use session handling mechanisms with web applications.

Post Lab Questions:

1. **What is the difference between the GET and POST method?**

Ans: GET method:

- Appends form data into the URL in name/value pairs.
- Suitable for short data and non-sensitive information.
- Data is visible in the URL, which can be bookmarked.
- Limited by URL length constraints.

POST method:

- Sends form data as part of the HTTP request body.
- Suitable for larger and sensitive data (e.g., passwords).
- Data is not displayed in the URL, providing more security.
- No size limitations like GET.

2. **Explain the importance of the $_Server Method?**

Ans: The $_SERVER method in PHP is a super global array containing information such as headers, paths, and script locations. It provides server and execution environment information. It helps in obtaining the current script's path, server name, and more, which is essential for dynamic web application development. For example, $_SERVER["PHP_SELF"] returns the filename of the currently executing script, which can be useful for form action attributes but needs to be handled carefully to avoid XSS attacks.

3. **How Form validation can help to prevent cross site scripting attacks?**

Ans: Form validation ensures that the data entered by the user conforms to expected formats and rules, reducing the risk of malicious data submission. By validating and sanitizing inputs, such as checking for valid email formats or disallowing special characters, the likelihood of injecting harmful scripts is minimized. Using functions like htmlspecialchars() to convert special characters into HTML entities prevents malicious code from being executed by the browser, thereby preventing XSS attacks. Regular expressions and built-in PHP filters are often used to validate and sanitize user inputs effectively.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

The objective of the experiment was to understand form handling in PHP using GET and POST methods. Through this experiment, we created a registration form, submitted data to the server, and processed it using PHP. We learned the differences between GET and POST methods, the importance of using the $_SERVER method securely, and how form validation can prevent cross-site scripting attacks. The practical application of these concepts was demonstrated by handling user inputs securely and efficiently, thereby achieving the desired outcomes of the experiment.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books:

1. Thomson PHP and MySQL Web Development Addison-Wesley Professional, 5th Edition 2016.
2. Peter MacIntyre, Kevin Tatroe Programming PHP O'Reilly Media, Inc, 4th Edition 2020
3. Frank M. Kromann Beginning PHP and MySQL: From Novice to Professional, Apress 1st Edition, 2018
4. https://www.w3schools.com/php/php_form_validation.asp
