Experiment No. 3

Title: Form Handling using PHP

Batch:A3 **Roll No:16010421073** **Experiment No.:4**

Aim: Form Handling using PHP

Resources needed:

Theory:

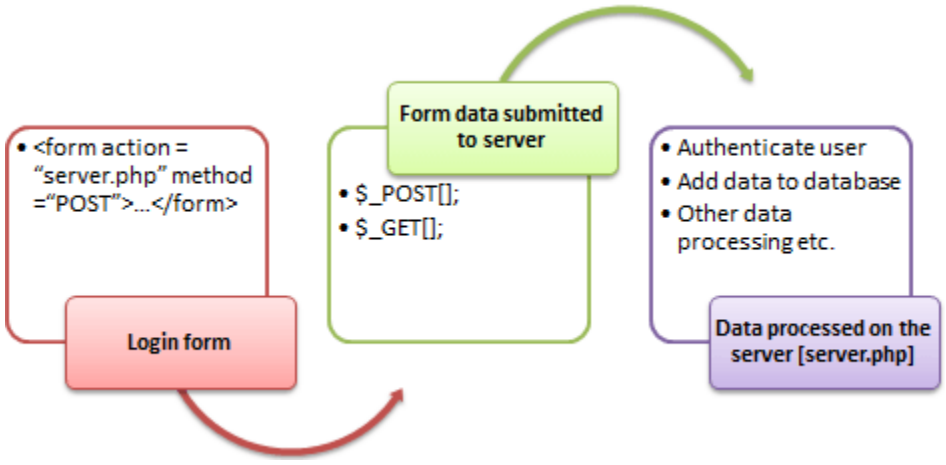
PHP Registration Form using GET, POST Methods with Example

What is Form?

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



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

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

When and why we are 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.

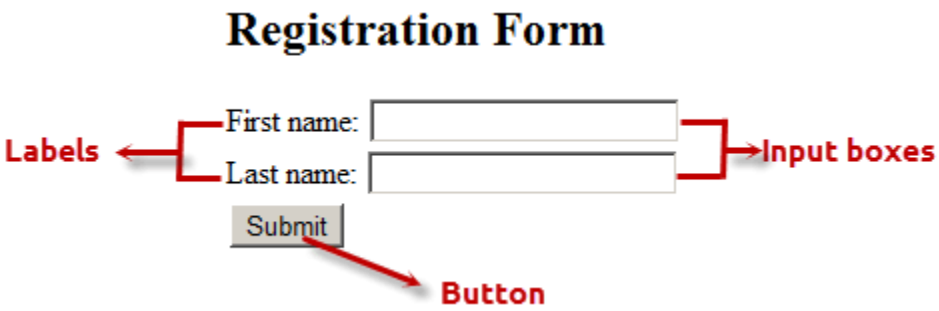
K. J. SOMAIYA COLLEGE OF ENGG.

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.

K. J. SOMAIYA COLLEGE OF ENGG.

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.

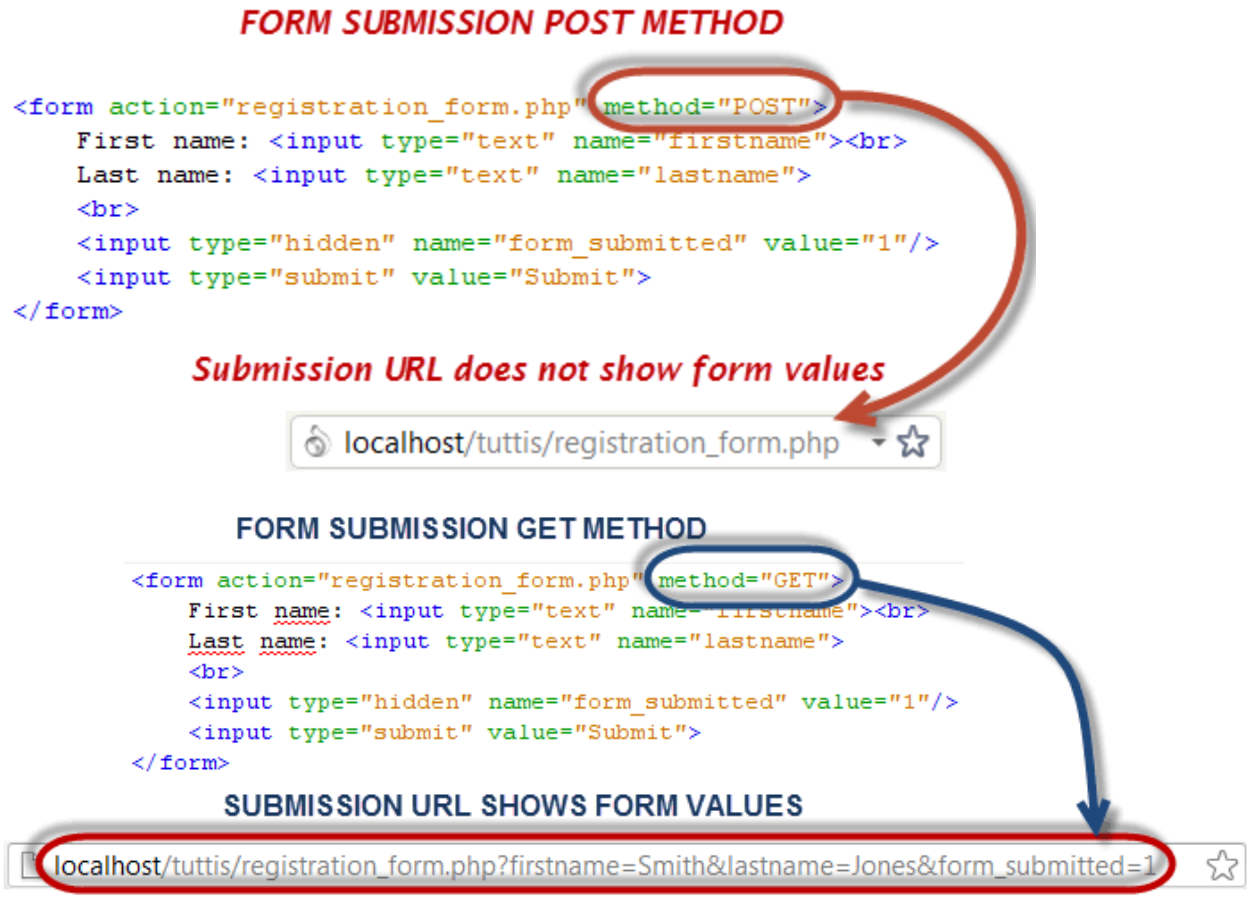
It has the following syntax.

```
<?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 check boxes, radio buttons

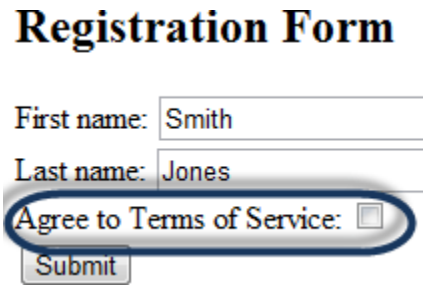
If the user does not select a check box or radio button, no value is submitted, if the user selects a check box or radio button, the value one (1) or true is submitted.

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



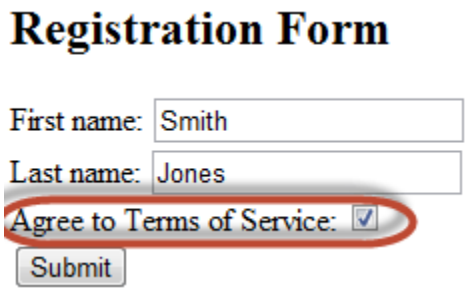
Fill in the first and last names

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

Click on submit button

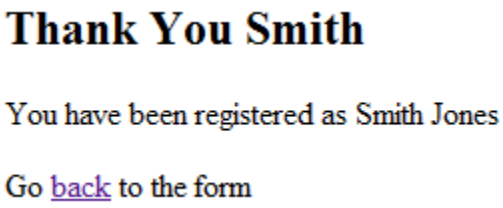
You will get the following results

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



Click on submit button

You will get the following results



Validate Name using expressions

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

Validate E-mail 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')</script>
```

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 of 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/&quot;&gt;&lt;script&gt;alert('h
acked')&lt;/script&gt;">
```

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.

Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hospital Registration Form</title>
</head>
<link rel="stylesheet" href="form.css">
<body>
    <h2>Hospital Registration</h2>
    <form action="process_registration.php" method="post">
        <label for="hospital_name">Hospital Name:</label>
        <input type="text" name="hospital_name" required><br>

        <label for="email">Email:</label>
        <input type="email" name="email" required><br>
```

```
        <label for="password">Password:</label>
        <input type="password" name="password" required><br>

        <label for="department">Department:</label>
        <select name="department" required>
            <option value="">Select Department</option>
            <option value="Cardiology">Cardiology</option>
            <option value="Orthopedics">Orthopedics</option>
            <option value="Pediatrics">Pediatrics</option>
        </select><br>

        <label>Specializations:</label><br>
        <input type="checkbox" name="specializations[]" value="Surgery">Surgery<br>
        <input type="checkbox" name="specializations[]"
value="Radiology">Radiology<br>
        <input type="checkbox" name="specializations[]" value="Oncology">Oncology<br>

        <label for="gender">Gender:</label><br>
        <input type="radio" name="gender" value="Male" required>Male
        <input type="radio" name="gender" value="Female" required>Female<br>

        <input type="submit" value="Register">
    </form>
</body>
</html>
```

Process_registration.php

```
 <?php
if ($_SERVER["REQUEST_METHOD"] == "POST")
{
    $hospital_name = $_POST["hospital_name"];
    $email = $_POST["email"];
    $password = $_POST["password"];
    $department = $_POST["department"];


    if (strlen($hospital_name) < 5)
    {
        echo "Hospital Name must be at least 5 characters long.<br>";
    }


    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "Invalid email format.<br>";
```

```
    }


    if (strlen($password) < 6) {
        echo "Password must be at least 6 characters long.<br>";
    }


    if (empty($department)) {
        echo "Please select a department.<br>";
    }



}
else
{

    echo "Form not submitted.";
}

echo "form submitted";
echo "<br>";
echo "<br>";
echo $hospital_name;
echo "<br>";
echo $email;
echo "<br>";
echo $department;
echo "<br>";


?>
```

form.css

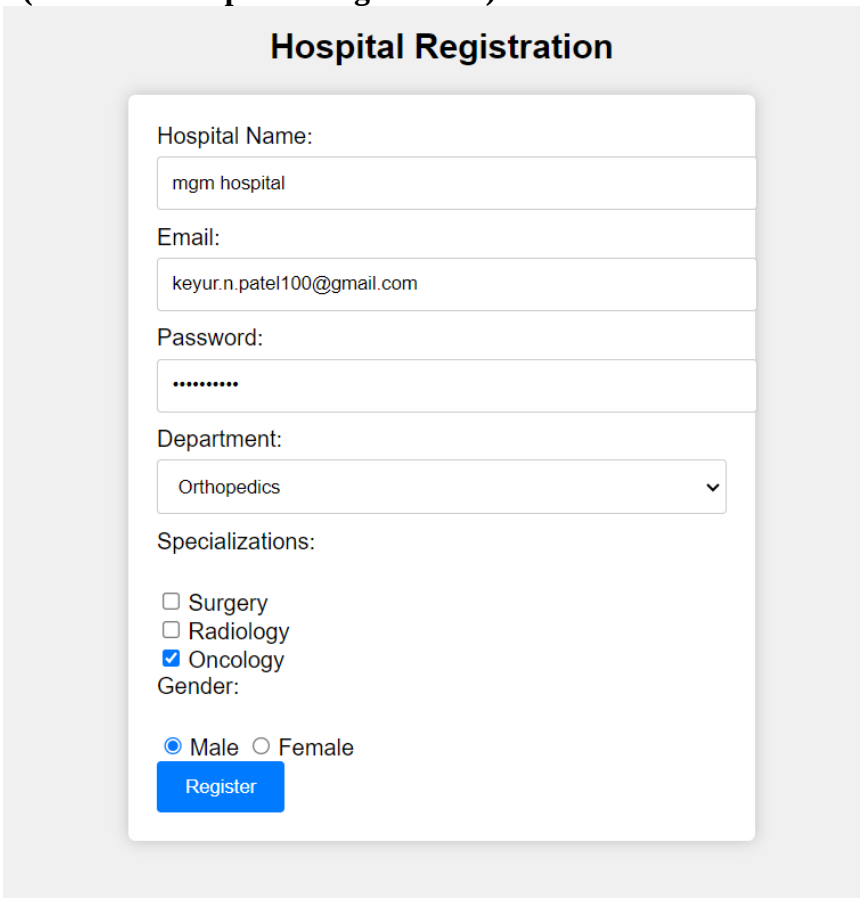
```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
}

h2 {
    text-align: center;
}

form {
    max-width: 400px;
```

```
    margin: 0 auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
}

label {
    display: block;
    margin-bottom: 5px;
}

input[type="text"],
input[type="email"],
input[type="password"],
select {
    width: 100%;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid #ccc;
    border-radius: 3px;
}

input[type="checkbox"] {
    margin-right: 5px;
}

input[type="radio"] {
    margin-right: 5px;
}

input[type="submit"] {
    background-color: #007bff;
    color: #fff;
    padding: 10px 20px;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}

input[type="submit"]:hover {
    background-color: #0056b3;
}

.error {
    color: #d9534f;
    font-size: 14px;
```

```
}
```

Output: Attach Screenshots

(On Correct inputs in registration)

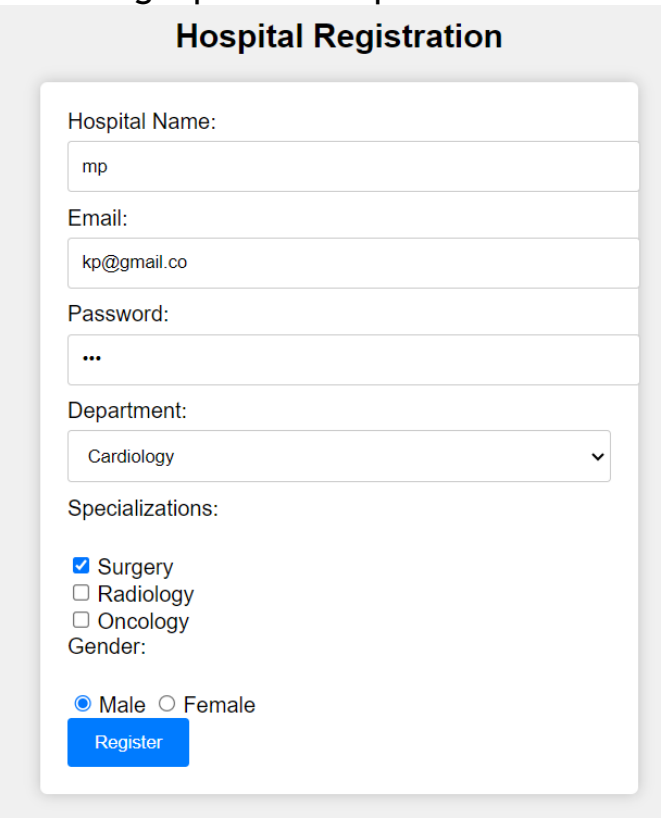


It will show form submitted with no errors

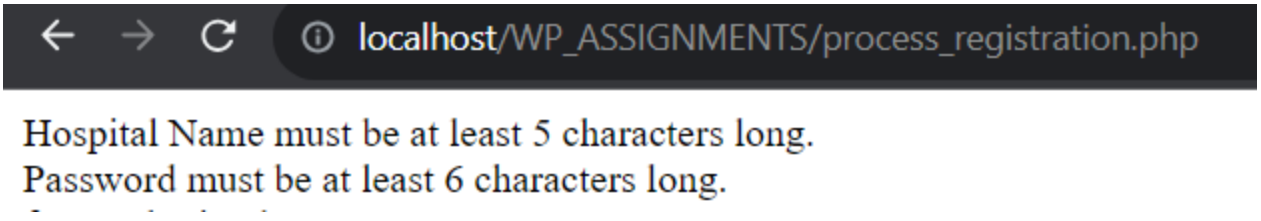
form submitted

mgm hospital
keyur.n.patel100@gmail.com
Orthopedics

On wrong inputs at Hospital name and Password



It will show error at process_registration.php



1) What is the difference between GET and POST method?

Answer:

Aspect	GET	POST
Data Transmission	Appends data to the URL as query parameters	Sends data in the HTTP request body
Security	Less secure as data is visible in the URL	More secure as data is not visible in URL
Data Limit	Limited by URL length (usually around 2048 characters)	Limited by server and PHP configuration
Caching	Data can be cached and bookmarked	Data is not cached or bookmarked
Idempotent	Generally considered idempotent (repeating the request does not change the state)	Not idempotent (repeating the request may change the state)
Browser Back Button	Data is included in the browser's history	Data is not included in the browser's history
Use Cases	Suitable for retrieving data from the server	Suitable for submitting sensitive data or modifying server state
Request Visibility	Data is visible in the URL, can be seen by users	Data is not visible in the URL, more private
Security Consideration	Insecure for sensitive data like passwords	More secure for sensitive data
PHP Access	Accessed using `$_GET` superglobal	Accessed using `$_POST` superglobal

2) Explain the importance of $_Server Method?

Answer: The $_SERVER superglobal in PHP is a critical and powerful variable that provides information about the server environment and the current request. It contains an array of server-related data, including HTTP headers, server information, request method, and more. Understanding and using $_SERVER is important for various reasons:

1. **Request Handling:** $_SERVER['REQUEST_METHOD'] holds the HTTP request method used by the client (e.g., GET, POST, PUT, DELETE). This information is essential for routing requests to the appropriate parts of your application.
2. **Request URI:** $_SERVER['REQUEST_URI'] provides the URL path requested by the client. It's crucial for determining the route or page to serve based on the URL.
3. **Server Information:** $_SERVER['SERVER_SOFTWARE'],

$_SERVER['SERVER_NAME'], and $_SERVER['SERVER_ADDR'] contain information about the web server software, server name, and server's IP address. This can be useful for debugging and logging.

4. **Client Information:** $_SERVER['REMOTE_ADDR'] stores the client's IP address, helping you track or restrict access based on IP.
5. **HTTP Headers:** $_SERVER contains information about incoming HTTP headers (e.g., $_SERVER['HTTP_USER_AGENT'] for the user agent string, $_SERVER['HTTP_REFERER'] for the referring URL). These headers can be used for various purposes, such as browser detection, authentication, and security checks.

In summary, **$_SERVER** is a fundamental tool for PHP developers to gather information about incoming requests, server configurations, and client environments. It plays a crucial role in routing, security, and customization of PHP applications, making it an essential superglobal in PHP programming

3) How Form validation can help to prevent crosssite scripting attack?

Answer: Form validation is an important security measure that can help prevent Cross-Site Scripting (XSS) attacks when properly implemented. XSS attacks occur when an attacker injects malicious scripts (usually JavaScript) into web applications, and these scripts are then executed in the context of a victim's browser. Form validation helps in preventing such attacks by ensuring that user input is safe and does not contain malicious code. Here's how form validation helps prevent XSS attacks:

1. **Sanitizing User Input:** Form validation involves checking user input for potentially dangerous characters and scripts. It can sanitize input by stripping or escaping characters that have special meaning in HTML, JavaScript, or other contexts. For example, it can convert < to < and > to >, preventing the execution of injected scripts.
2. **Rejecting Malicious Input:** Validation can reject input that contains known malicious patterns or scripts. Regular expressions and pattern matching can be used to identify and block input that matches known attack vectors.
3. **Validating Input Types:** Ensuring that input matches the expected data type (e.g., email addresses, phone numbers, or dates) can help prevent attacks. This helps filter out input that doesn't conform to expected patterns, reducing the risk of code injection.
4. **Encoding Output:** While not directly part of form validation, encoding user-generated data when rendering it in HTML can help prevent XSS. Validation should work in tandem with proper output encoding to ensure that even if some malicious input bypasses validation, it doesn't execute when displayed in the browser.
5. **Whitelisting Allowed HTML Tags and Attributes:** In cases where some HTML is allowed (e.g., in user-generated content), form validation can implement a whitelist approach, allowing only specific, safe HTML tags and attributes while rejecting others.

Outcomes: CO2 : Design forms and use session handling mechanism with web applications.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)
I understood the concept of form handling and form validation in php. Implemented form handling and validation for a simple form and on entering wrong inputs user gets an option to rectify the form.

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