DOCTOR HARISINGH GOUR VISWAVIDALAYA
DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

CSA-CC-3204 WEB APPLICATION DESIGN USING PHP
SESSION: 2024-2025
MID II ASSIGNMENT

Creating HTML Forms with PHP

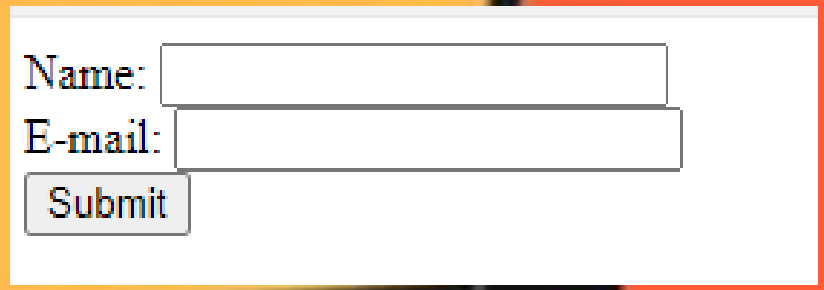
A Guide to Building Dynamic Forms for Web Applications

SUBMITTED BY
SAKSHI PANDEY
Y23271024
MCA III SEMESTER

SUBMITTED TO
ER. KAMAL KANT SIR

TABLE OF CONTENTS

Introduction



What are HTML Forms?

- Used to collect user input.
- Essential for user interaction on websites (e.g., sign-ups, surveys).

Why Use PHP with HTML Forms?

- PHP processes form data on the server.
- Enables dynamic content generation and data handling.

Basic Structure of an HTML Form

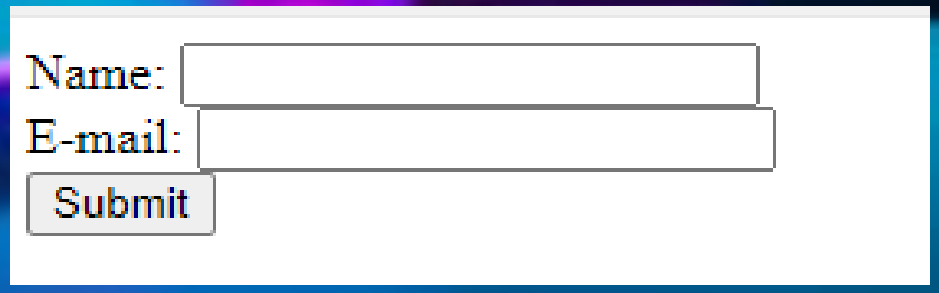
```html
<!DOCTYPE HTML>
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

form_get.php

- **Input Types**:
  - text, email, password, checkbox, radio
- **Key Attributes**:
  - action: The PHP script to process the form data.
  - method: HTTP method (GET or POST) to send data.



GET METHOD VS POST METHOD

	A	B	C
1	Feature	GET Method	POST Method
2	Usage	Used to request data from a server.	Used to send data to a server.
3	Data visibility	Appends data to the URL (visible in URL).	Data is sent in the request body (hidden).
4	Data length	Limited by URL length (generally 2048 characters).	No size limitations, can send large amounts of data.
5	Caching	Can be cached by browsers and servers.	Not cached by default.
6	Bookmarking	Can be bookmarked since data is in the URL.	Cannot be bookmarked.
7	Security	Less secure as data is visible in the URL, especially for sensitive data.	More secure since data is hidden but requires encryption (e.g., HTTPS).
8	Idempotency	Idempotent (repeated requests yield the same result).	Not necessarily idempotent (repeated requests can have different outcomes).
9	Use in forms	Used for form submissions where data retrieval is the goal.	Used in forms where data submission or manipulation is required.
10	Effect on server	Generally does not change server state.	Often changes server state (e.g., updating a database).
11	Performance	Faster due to caching and minimal overhead.	Slightly slower due to data being sent in the body.
12	Data type	Only allows ASCII characters in the URL.	Can send binary, text, or other types of data in the body.
13	Examples	URL with parameters: 'example.com/page?name=John'	Sending a form: data is in the request body, not visible in the URL.
14			

form_post.php

Processing Form Data with PHP

Accessing Form Data:

php

```
<?php if ($_SERVER["REQUEST_METHOD"] == "POST") { $name = htmlspecialchars($_POST['name']); echo "Hello, " . $name; } ?>
```

- **Key Points**:
  - Use $_POST to retrieve data when using the POST method.
  - Use htmlspecialchars() to prevent XSS attacks.

```php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

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

The `htmlspecialchars()` function converts special characters into HTML entities. This means that it will replace HTML characters like `<` and `>` with `<` and `>`. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Validating Form Data

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

form_validation.php

Client-Side Validation:

- Use HTML5 attributes (e.g., required, pattern).

- **Server-Side Validation**:

php

```
if (empty($name)) { echo "Name is required."; } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) { echo "Invalid email format."; }
```

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

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

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

Handling Multiple Form Inputs

* required field

Name: dhsgdhs *

E-mail: vdsahgdhas@gmail.com *

Website: www.fsagfss.com

Comment: dhgfdshgdsjlhfd

Gender: Female Male Other *

Submit

form_complete.php

Your Input:

dhsgdhs
vdsahgdhas@gmail.com
www.fsagfss.com
dhgfdshgdsjlhfd
other

```
Name: <input type="text" name="name" value="<?php echo $name;?>">

E-mail: <input type="text" name="email" value="<?php echo $email;?>">

Website: <input type="text" name="website" value="<?php echo $website;?>">

Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?></textarea>

Gender:
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="other") echo "checked";?>
value="other">Other
```

Storing Form Data in a Database

•**Connecting to a Database**:

form_database.php

php

```
$conn = new mysqli($servername, $username, $password, $dbname);
```

•**Inserting Data**:

php

```
$sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')"; if ($conn->query($sql) === TRUE) { echo "New record created successfully"; }
```

Security Considerations

- **Sanitize User Input**:
  - Use prepared statements to prevent SQL injection.
- **Use HTTPS**:
  - Protect data in transit.
- **Session Management**:
  - Implement proper session handling to maintain user authentication.

Conclusion

- **Summary**:
  - HTML forms are crucial for user interaction.
  - PHP is a powerful tool for processing and validating form data.
  - Proper security measures are essential for protecting user information.

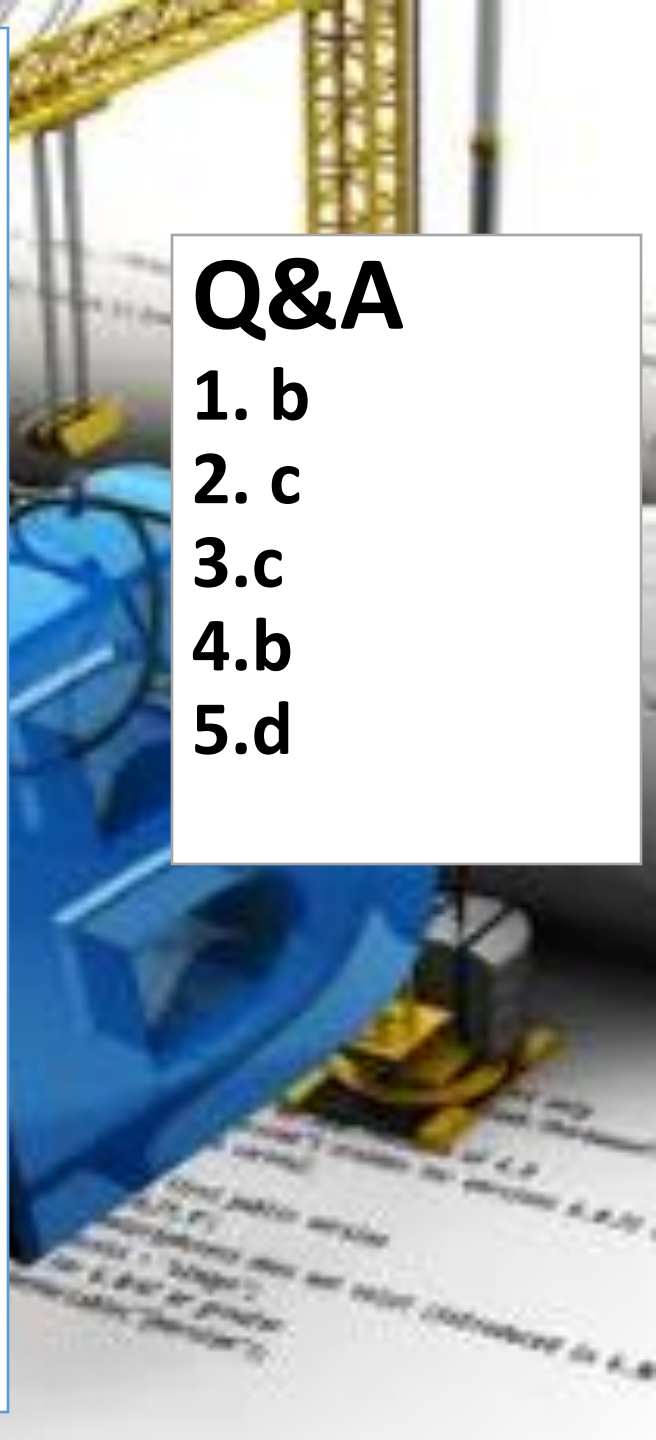
1. Which two predefined variables are used to retrieve information from forms?
   a) $GET & $SET
   b) $_GET & $_SET
   c) $__GET & $__SET
   d) GET & SET

2. The attack which involves the insertion of malicious code into a page frequented by other users is known as _____
   a) basic SQL injection
   b) advanced SQL injection
   c) cross-site scripting
   d) scripting

3. When you use the $_GET variable to collect data, the data is visible to _____
   a) none
   b) only you
   c) everyone
   d) selected few

4. When you use the $_POST variable to collect data, the data is visible to _____
   a) none
   b) only you
   c) everyone
   d) selected few

5. Which variable is used to collect form data sent with both the GET and POST methods?
   a) $BOTH
   b) $_BOTH
   c) $REQUEST
   d) $_REQUEST



Q&A

1. b
2. c
3.c
4.b
5.d

References

- **Further Reading**:
  - PHP Documentation: php.net
  - W3Schools: w3schools.com
  - MDN Web Docs: developer.mozilla.org
  - https://www.sanfoundry.com/1000-php-questions-answers/
  - PHP Black Book by Peter Maudling