Building Forms

Overview

Forms are used to collect user input and send it to the server for processing. HTML forms are created using the `<form>` tag, and PHP is used to process the submitted data.

Key Attributes of a Form

- `action` : Specifies the URL where the form data is sent.
- `method` : Defines how the form data is sent (`GET` or `POST`).
- `enctype` : Specifies how the form data is encoded (e.g., `multipart/form-data` for file uploads).

Example: Building a Form

```
<form action="process_form.php" method="POST">
    Name: <input type="text" name="name"><br>
    Email: <input type="email" name="email"><br>
    Message: <textarea name="message"></textarea><br>
    <input type="submit" value="Submit">
</form>
```

Run HTML

Retrieving Form Data

Overview

PHP provides superglobal arrays (`$_GET` and `$_POST`) to retrieve form data submitted via `GET` or `POST` methods.

Key Superglobals

- `$_GET` : Contains data sent via the `GET` method (visible in the URL).
- `$_POST` : Contains data sent via the `POST` method (not visible in the URL).

Example: Retrieving Form Data

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST['name'];
    $email = $_POST['email'];
    $message = $_POST['message'];
    echo "Name: $name <br> Email: $email <br> Message: $message";
}
?>
```

Processing Forms

Overview

Form processing involves validating and sanitizing user input before using it in your application. This ensures data integrity and security.

Key Steps

1. **Validation**: Check if the input meets specific criteria (e.g., required fields, valid email format).
2. **Sanitization**: Clean the input to prevent security issues (e.g., SQL injection, XSS).

Example: Form Validation and Sanitization

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Validate inputs
    if (empty($_POST['name']) || empty($_POST['email'])) {
        echo "Name and Email are required!";
    } else {
        // Sanitize inputs
        $name = htmlspecialchars($_POST['name']);
        $email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);

        // Process the data
        echo "Name: $name <br> Email: $email";
    }
}
?>
```

Setting Response Headers

Overview

Response headers are used to send additional information from the server to the client. Common use cases include redirecting users, setting cookies, and specifying content types.

Key Functions

- `header()` : Sends a raw HTTP header to the client.

Example: Redirecting Users

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Process form data
    $name = $_POST['name'];
    $email = $_POST['email'];

    // Redirect to a thank-you page
    header("Location: thank_you.php");
    exit();
}
?>
```

Example: Setting Content Type

```
<?php
header("Content-Type: application/json");
echo json_encode(array("message" => "Hello, World!"));
?>
```

Summary of Key Points

TopicDescriptionBuilding Forms Use the `<form>` tag with `action` , `method` , and `enctype` attributes.**Retrieving Form Data**Use `$_GET` or `$_POST` superglobals to access form data.**Processing Forms**Validate and sanitize user input to ensure data integrity and security.**Setting Response Headers**Use the `header()` function to send HTTP headers (e.g., redirects, content type).

Practical Questions

1. Create an HTML form with fields for name, age, and gender. Use the `POST` method to submit the form.
2. Write a PHP script to retrieve and display the form data.
3. Add validation to ensure the name and age fields are not empty.
4. Use the `header()` function to redirect the user to a "thank you" page after form submission.
5. Set the response header to return JSON data instead of HTML.