

**Batch: C2**

**Roll No.: 16010122257**

**Experiment No. 06**

**TITLE:** Cookies and Session in PHP

**AIM: The aim of this experiment is to learn how to create and destroy cookies and sessions in PHP**

**Expected Outcome of Experiment:**

The expected outcome of the experiment is understanding of how to effectively manage cookies and sessions in PHP.

Books/ Journals/ Websites referred:

1. .

**Problem Statement: Description of the application implemented with output**:

1. Create a secure login page with fields for username and password. Implement authentication of user credentials. Upon successful login, establish a session and set cookies, then redirect the user to a welcome page.
2. Upon refreshing the welcome page, ensure the session and cookies are properly destroyed.

Implementation and screen shots of output:

**Expt 6.html:**

<!DOCTYPEhtml>

<htmllang="en">

<head>

<metacharset="UTF-8">

<metaname="viewport"content="width=device-width, initial-scale=1.0">

<title>Personal Information Form</title>

</head>

<body>

<h2>Personal Information Form</h2>



<formaction="submit.php"method="post">

<labelfor="fullname">Full Name:</label><br>

<inputtype="text"id="fullname"name="fullname"required><br>

<labelfor="email">Email:</label><br>

<inputtype="email"id="email"name="email"required><br>

<labelfor="age">Age:</label><br>

<inputtype="number"id="age"name="age"required><br>

<labelfor="country">Country:</label><br>

<inputtype="text"id="country"name="country"required><br>

<inputtype="submit"value="Submit">

</form>

</body>

</html>

**Submit.php:**

<?php session\_start();

// Validate form inputs

$errors = [];

$fullname = $\_POST['fullname'] ??'';

$email = $\_POST['email'] ??'';

$age = $\_POST['age'] ??'';

$country = $\_POST['country'] ??'';

// Check if all fields are filled

if (empty($fullname) || empty($email) || empty($age) || empty($country))

{

$errors[] = "All fields are required";

}

// Check email format

if(!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {

$errors[] = "Invalid email format";

}

// Check if age is a positive number if(!is\_numeric($age) || $age<= 0) {

$errors[] = "Age must be a positive number";

}



// If there are validation errors, redirect back to the form with errors if(!empty($errors)) {

$\_SESSION['errors'] = $errors; header("Location: index.html"); exit;

}

// If no errors, store form data in session

$\_SESSION['fullname'] = $fullname;

$\_SESSION['email'] = $email;

$\_SESSION['age'] = $age;

$\_SESSION['country'] = $country;

// Store form data in cookies

setcookie('fullname', $fullname, time() + (86400 \* 30), "/"); setcookie('email', $email, time() + (86400 \* 30), "/"); setcookie('age', $age, time() + (86400 \* 30), "/"); setcookie('country', $country, time() + (86400 \* 30), "/");

// Redirect to a success page header("Location: success.php"); exit;

?>

**Success.php:**

<?php session\_start();

// Retrieve form data from session

$fullname = $\_SESSION['fullname'] ??'';

$email = $\_SESSION['email'] ??'';

$age = $\_SESSION['age'] ??'';

$country = $\_SESSION['country'] ??'';

// Display form data echo"<h2>Personal Information</h2>"; echo"Full Name: " .$fullname ."<br>"; echo"Email: " .$email ."<br>"; echo"Age: " .$age ."<br>"; echo"Country: " .$country ."<br>";

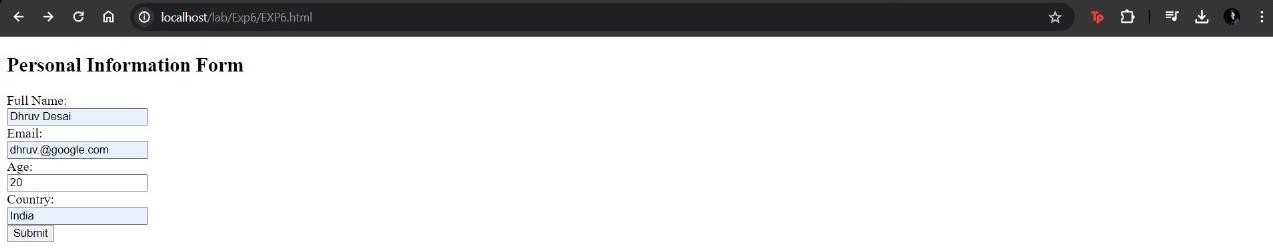
// Destroy session



session\_destroy();

?>

**OUTPUT:**







**Post Lab Objective with Ans :**

1. **Differentiate between sessions and cookies Answer:**

Sessions and cookies are both mechanisms used in web development to maintain state information between HTTP requests, but they serve different purposes and have different characteristics:

* 1. **Sessions:**
     + Server-side storage: Sessions store data on the server side. When a user visits a website, the server creates a unique session identifier (session ID) for that user.
     + Data storage: Sessions can store a wide range of data types including objects, arrays, and strings.
     + Security: Session data is stored on the server, making it more secure compared to cookies which are stored on the client-side.
     + Expiry: Sessions typically have an expiration time, after which the session data is invalidated and removed from the server.
     + Usage: Sessions are commonly used to store sensitive user information such as login credentials, shopping cart contents, and user preferences.
  2. **Cookies:**
     + Client-side storage: Cookies store data on the client side, in the user's browser. Each cookie is associated with a specific domain and path.
     + Data storage: Cookies are limited in the amount of data they can store (usually around 4KB per cookie) and can only store strings.
     + Security: Cookies are less secure compared to sessions because they are stored on the client side and can be manipulated by the user.
     + Expiry: Cookies can have both session cookies (which expire when the browser session ends) and persistent cookies (which have an expiration date set by the server).
     + Usage: Cookies are commonly used for tracking user behavior, personalizing content, and maintaining user preferences across multiple sessions.

In summary, sessions are primarily used for server-side storage of user data, providing better security and flexibility for storing various data types, while cookies are used for client-side storage and are more limited in terms of the data they can store and their security.



1. **Describe the methods used to properly destroy sessions and cookies in a web application.**

**Answer:**

Destroying sessions and cookies properly is important for maintaining security and managing user data in a web application. Here's how you can do it for both sessions and cookies:

Destroying Sessions:

* 1. Server-side Session Management:
     + Most web frameworks provide built-in mechanisms for managing sessions. Ensure that you're using a secure and reliable session management system provided by your framework or library.
     + When a user logs out or their session needs to be destroyed, you should invalidate the session on the server side. This typically involves removing the session data associated with the user's session ID.
  2. Clearing Session Data:
     + Delete all session data associated with the user's session ID. This ensures that any sensitive information stored in the session is no longer accessible.
     + Depending on the framework or language you're using, there are specific methods to clear session data. For example, in PHP, you can use session\_destroy() or $\_SESSION = array().
  3. Session Expiry:
     + Sessions should have a defined expiration time. After this time, the session data should be automatically destroyed on the server side. Ensure that session expiration settings are configured appropriately in your application.

Destroying Cookies:

1. Setting Cookie Expiry:
   * For persistent cookies, set an expiration time when you initially create the cookie. This can be a specific date/time in the future or a duration from the current time.
   * For session cookies, they automatically expire when the user closes their browser. However, you can set an explicit expiration time to ensure the cookie is deleted after a certain period of inactivity.



1. Clearing Cookie Data:
   * To remove a cookie from the client-side, you need to send a new cookie with the same name and an expiry date set to a time in the past. This effectively tells the browser to delete the cookie.
   * In most programming languages and frameworks, there are functions or methods to unset or remove cookies. For example, in JavaScript, you can use document.cookie to modify cookies.
2. Secure and HTTP-Only Flags:
   * When creating cookies, consider setting the secure flag to ensure the cookie is only transmitted over HTTPS connections, enhancing security.
   * Set the HttpOnly flag to prevent client-side scripts from accessing the cookie, mitigating certain types of attacks such as cross-site scripting (XSS).

By following these methods, you can properly destroy sessions and cookies in your web application, ensuring the security and privacy of your users' data.

1. **Discuss strategies for implementing security measures in form sessions Answer:**

Implementing security measures in form sessions is crucial to protect sensitive user data and prevent unauthorized access. Here are some strategies for enhancing security in form sessions:

* 1. Use HTTPS:
     + Always serve your web application over HTTPS to encrypt data transmitted between the client and the server. This prevents attackers from intercepting sensitive information, including session IDs and form data.
  2. Session Expiry:
     + Set a reasonable expiration time for form sessions. Once the session expires, users should be required to log in again to continue accessing sensitive features or submitting forms. This prevents unauthorized access if a session is hijacked.
  3. Secure Session Storage:
     + Store session data securely on the server side. Avoid storing sensitive information in the client's cookies or local storage, as these can be tampered with or accessed by malicious actors. Use server-side session storage mechanisms provided by your web framework or library.



* 1. CSRF Protection:
     + Implement Cross-Site Request Forgery (CSRF) protection mechanisms to prevent attackers from tricking authenticated users into submitting malicious requests on their behalf. Use techniques like CSRF tokens, which are unique tokens generated for each session and embedded in forms to validate the origin of form submissions.
  2. Session Regeneration:
     + Regenerate session IDs after a user authenticates or performs certain sensitive actions. This helps mitigate session fixation attacks and session hijacking attempts by invalidating previously used session IDs.
  3. Limit Session Scope:
     + Limit the scope and permissions of each session based on the user's role and access level. Ensure that users only have access to the resources and functionality they are authorized to use, and restrict sensitive operations to authenticated sessions only.
  4. Session Monitoring and Logging:
     + Implement session monitoring and logging to detect suspicious activity and unauthorized access attempts. Log session-related events such as login attempts, session creations, and session expirations for auditing and forensic purposes.
  5. Session Revocation:
     + Provide users with the ability to revoke active sessions from their account settings. This allows users to terminate any active sessions they no longer trust or recognize, reducing the risk of unauthorized access.
  6. Input Validation and Sanitization:
     + Implement strict input validation and data sanitization measures to prevent common vulnerabilities such as SQL injection, XSS (Cross-Site Scripting), and other injection attacks. Validate and sanitize all form inputs before processing them to prevent malicious input from compromising the integrity of your application.

By implementing these security measures, you can enhance the security of form sessions in your web application and protect sensitive user data from unauthorized access and exploitation.