Converted Document

Unit 4

1. Introduction to Cookies

A cookie is a small piece of textual data sent by a web server to a user's browser and stored on the client-side. The browser sends the cookie back to the server with each subsequent request to the same domain.

In PHP, cookies are used to store user-specific data across multiple pages and sessions. Since HTTP is a stateless protocol, cookies help maintain continuity in user interactions.

? 2. Why Are Cookies Needed?

HTTP does not remember previous interactions between a browser and a server. Therefore, to maintain state across multiple page requests (such as remembering a user login), cookies are essential.

They allow developers to:

Track users

Personalize content

Maintain session continuity

? 3. Purpose of Cookies in PHP

Let?s break down the core purposes of cookies in PHP:

? A) User Authentication

Cookies are often used to remember logged-in users.

When a user logs in and checks ?Remember Me?, their login ID or token is stored in a cookie.

On future visits, the server reads the cookie to auto-login the user.

// Set login cookie after authentication

setcookie("user", "john_doe", time() + (86400 * 7), "/"); // Expires in 7 days

? B) User Preferences & Personalization

Cookies can store custom settings such as:

Theme (dark/light)

Language (English, Hindi, etc.)

Font size

Layout preferences

These preferences are loaded each time the user revisits the site.

setcookie("language", "English", time() + 3600 * 24 * 30);

? C) Tracking User Activity

Cookies allow the server to track user navigation behavior, such as:

Last visited pages

Number of visits

Time spent on a page

Useful for analytics, marketing, and improving user experience.

```
if (!isset($_COOKIE["visitCount"])) {
setcookie("visitCount", 1, time() + 3600);
} else {
setcookie("visitCount", $_COOKIE["visitCount"] + 1, time() + 3600);
}
```

? D) Shopping Cart and E-commerce

Cookies can temporarily store cart items if the user is not logged in. This enables the cart to persist across multiple pages or visits.

```
setcookie("cart", json_encode(['item1', 'item2']), time() + 3600);
```

? E) Session Continuity

Even though PHP has session handling ($_SESSION), cookies often work together with sessions:

PHP stores the session ID in a cookie (PHPSESSID)

This ID helps retrieve session data on the server

So technically, even PHP sessions use cookies internally.

? F) Temporary Data Storage

Cookies are useful for storing temporary information such as:

Form values

Temporary tokens

Referral IDs

They reduce server load by keeping some state client-side.

? 4. Security Considerations

Cookies can be manipulated by users. Never store passwords in cookies.

Use the HttpOnly and Secure flags for protection:

```
setcookie("user", "john", time() + 3600, "/", "", true, true);
```

? 5. Summary: Key Benefits of Using Cookies in PHP

? 6. Conclusion

Cookies in PHP are powerful tools for storing client-side data and maintaining state in web applications. They enable personalization, authentication, tracking, and session management. While convenient, they must be handled securely and efficiently to ensure both performance and user privacy.

1. Introduction

Cookies are small text files stored on a user's browser to manage sessions, preferences, and activity tracking. However, over time, many myths and misunderstandings have developed around their role, usage, and security risks. Let's explore and debunk some of these common cookie myths.

?? 2. Common Myths About Cookies (and the Truth)

? Myth 1: Cookies Are Programs That Can Run Code

? False: Cookies are just plain text files.
? Truth: Cookies cannot execute code or perform any operations. They are simply used to store and retrieve data. They pose no threat by themselves unless misused with other vulnerabilities (e.g., cross-site scripting).

? Myth 2: Cookies Are Viruses or Malware

? False: Cookies are not harmful files.
? Truth: Cookies cannot contain viruses or malware because they are not executable. However, they can be used to track users, which raises privacy concerns, not security ones.

? Myth 3: Cookies Can Access Private Data From Your Computer

? False: Cookies have no access to your local files.
? Truth: A cookie can only store the information that a website writes into it and can only be read by that specific domain. Cookies cannot access files, documents, or any other part of your system.

? Myth 4: All Cookies Are Bad for Privacy

? False: Not all cookies are invasive.
? Truth: Some cookies are essential for website functionality, such as session management, login status, and remembering user settings. Privacy concerns mostly arise from third-party tracking cookies used by advertisers.

? Myth 5: Disabling Cookies Makes You Anonymous Online

? False: Many other tracking methods exist.

? Truth: Even if cookies are disabled, websites can track users using techniques like fingerprinting, local storage, IP address, and browser behavior.

? Myth 6: Deleting Cookies Will Log You Out of the Internet

? False: Only sessions on specific websites will end.

? Truth: Deleting cookies logs you out of websites that use cookies for session tracking, but it does not affect your internet connection or ability to browse websites.

? 3. Conclusion

Cookies are often misunderstood. While they can be used for tracking and personalization, they are not inherently dangerous. Most myths around cookies come from a lack of technical understanding. Knowing the truth about how cookies work helps developers use them responsibly and users make informed decisions about privacy settings.

1. What is a Cookie?

A cookie is a small piece of data that a server sends to the user?s web browser. The browser stores it and sends it back with subsequent requests to the same server. Cookies are mainly used to remember user information across different pages and sessions.

? 2. Syntax to Set a Cookie in PHP

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

? Parameters:

? 3. Basic Example of Setting a Cookie

```
<?php
// Set a cookie named "user" with value "John" that expires in 1 hour
setcookie("user", "John", time() + 3600, "/");
?>
```

This cookie will be available on all pages of the website ("/").

It expires after 3600 seconds (1 hour).

On the next page load, the cookie is available in $_COOKIE["user"].

? 4. Accessing the Cookie

```
<?php
if (isset($_COOKIE["user"])) {
echo "Welcome back, " . $_COOKIE["user"];
} else {
echo "Welcome, guest!";
}
?>
```

? 5. Deleting a Cookie

To delete a cookie, set the expiration time to a past time:

```
<?php
setcookie("user", "", time() - 3600, "/"); // Deletes the cookie
?>
```

? 6. Important Notes

Cookies must be set before any output (HTML or echo) is sent to the browser.

They are stored client-side and sent automatically with each HTTP request.

Use HttpOnly and Secure flags for better security.

? 7. Conclusion

Setting cookies in PHP is a simple and powerful way to maintain user data across different pages. Cookies are widely used for session management, user personalization, and login systems. With proper security measures, they enhance the user experience significantly.

1. Introduction

After a cookie is set in PHP using setcookie(), it is stored in the user's browser. The browser then sends the cookie back to the server with each subsequent request. In PHP, cookies are retrieved using the $_COOKIE superglobal.

? 2. Syntax to Retrieve a Cookie

```
$_COOKIE['cookie_name'];
```

$_COOKIE is an associative array.

It holds all the cookie values that have been set and returned by the client.

? 3. Example: Retrieving a Cookie in PHP

Step 1: Set a cookie (in one PHP file)

```
<?php
setcookie("user", "JohnDoe", time() + 3600, "/");
echo "Cookie has been set.";
?>
```

Step 2: Retrieve the cookie (in another page or after reload)

```
<?php
if (isset($_COOKIE["user"])) {
echo "Welcome, " . $_COOKIE["user"];
} else {
echo "User not recognized.";
}
?>
```

? 4. Using isset() for Safety

Before retrieving a cookie, it's important to check if it exists using isset() to avoid errors.

```
if (isset($_COOKIE['theme'])) {
$theme = $_COOKIE['theme'];
} else {
$theme = "light"; // default theme
}
```

? 5. Output Example

If the cookie user = JohnDoe exists, the browser displays:

```
Welcome, JohnDoe
```

If not:

```
User not recognized.
```

? 6. Key Points to Remember

? 7. Conclusion

Retrieving cookies in PHP is simple using the $_COOKIE array. It enables developers to maintain user data across sessions, personalize experiences, and remember user preferences. Always check for the cookie?s existence using isset() to write robust and error-free code.

1. Introduction

Cookies in PHP are stored on the client-side (browser) and are used to hold data such as user preferences, login info, and session identifiers.

Cookies can be expired manually to delete them. Deleting a cookie is essentially the same as setting it with a past expiration time.

? 2. Expiring Cookies

When setting a cookie, you can specify its expiration time using the time() function.

? Syntax:

```
setcookie(name, value, expire, path);
```

expire: Time in UNIX timestamp (seconds since 1970).

Example: time() + 3600 sets the cookie to expire in 1 hour.

? Example: Set a Cookie with 1-hour Expiry

```
<?php
setcookie("user", "John", time() + 3600, "/");
?>
```

This cookie will automatically expire after 1 hour, and the browser will delete it.

? 3. Deleting Cookies in PHP

To delete a cookie manually, you simply reset its value and set the expiration time to a past timestamp.

? Syntax to Delete a Cookie:

```
setcookie("cookie_name", "", time() - 3600, "/");
```

? Example:

```
<?php
// Delete a cookie named "user"
setcookie("user", "", time() - 3600, "/");
echo "Cookie deleted.";
?>
```

time() - 3600 sets the expiration 1 hour in the past.

The browser will remove the cookie immediately.

? 4. Important Notes on Deleting Cookies

? 5. Example: Complete Flow

```
<?php
// Set cookie
setcookie("user", "John", time() + 3600, "/"); // Expires in 1 hour
// Delete cookie
setcookie("user", "", time() - 3600, "/"); // Deletes the cookie
?>
```

? 6. Verifying Cookie Deletion

```
<?php
if (isset($_COOKIE["user"])) {
echo "User: " . $_COOKIE["user"];
} else {
echo "Cookie does not exist.";
}
?>
```

? 7. Conclusion

In PHP, cookies are automatically expired by setting a future expiry time and manually deleted by setting the expiration to a past time. Deleting cookies is an important aspect of user session management, privacy, and security in web applications.

1. Can We Store Arrays Directly in Cookies?

No, PHP cookies can only store strings, not arrays or objects directly. However, you can convert an array to a string format (like JSON or serialized data) and store that in the cookie.

? 2. Methods to Store Arrays in Cookies

? 3. Using serialize() and unserialize()

? Store Array:

```
<?php
$user = ["name" => "Alice", "age" => 25, "city" => "Delhi"];
$cookieData = serialize($user);
setcookie("userInfo", $cookieData, time() + 3600, "/");
?>
```

? Retrieve Array:

```
<?php
if (isset($_COOKIE["userInfo"])) {
$user = unserialize($_COOKIE["userInfo"]);
echo "Name: " . $user["name"];
}
?>
```

? 4. Using json_encode() and json_decode()

? Store Array as JSON:

```
<?php
$cart = ["pen", "notebook", "eraser"];
$cartData = json_encode($cart);
setcookie("cartItems", $cartData, time() + 3600, "/");
?>
```

? Retrieve and Decode:

```
<?php
if (isset($_COOKIE["cartItems"])) {
$cart = json_decode($_COOKIE["cartItems"], true); // 'true' returns array
foreach ($cart as $item) {
echo $item . "<br>";
}
}
?>
```

? 5. Important Notes

? 6. Conclusion

To store arrays in cookies in PHP, you must first convert the array into a string using serialize() or json_encode(). This allows you to take advantage of cookie storage while still preserving structured data like arrays. When retrieving, convert the string back to an array using unserialize() or json_decode().

1. What is a Session in PHP?

A session is a way to store information (in variables) to be used across multiple pages. Unlike cookies, session data is stored on the server, not the user's browser. It helps in maintaining user state between requests.

? 2. Starting a Session

To use sessions in PHP, you must first call:

```
session_start();
```

?? This must be the very first line before any HTML output.

? 3. ? Storing Session Variables

To store a session variable, simply assign a value to the $_SESSION superglobal array:

```
<?php
session_start();
$_SESSION["username"] = "john_doe";
$_SESSION["email"] = "john@example.com";
echo "Session variables are set.";
?>
```

This data will now be available across multiple pages as long as the session is active.

? 4. ? Reading Session Variables

You can access session data just like an associative array:

```
<?php
session_start();
if (isset($_SESSION["username"])) {
echo "Username: " . $_SESSION["username"];
} else {
```

```
echo "No session found.";
}
?>
```

? 5. ? Deleting Specific Session Variables

Use the unset() function to delete a particular session variable:

```
<?php
session_start();
unset($_SESSION["username"]); // Deletes only the 'username' session
?>
```

? 6. ? Destroying All Session Data

To completely remove all session data:

```
<?php
session_start();
session_unset();    // Unsets all session variables
session_destroy();  // Destroys the session itself
?>
```

After session_destroy(), the session ID is no longer valid.

? 7. Example: Full Flow

```
<?php
// Start session
session_start();
// Store session data
$_SESSION["user"] = "Alice";
$_SESSION["role"] = "admin";
// Access session data
echo $_SESSION["user"]; // Output: Alice
// Remove one variable
unset($_SESSION["role"]);
```

// Destroy all session data

session_unset();

session_destroy();

?>

? 8. Why Use Sessions?

? 9. Conclusion

Sessions in PHP allow for secure, server-side storage of user data across different web pages. With session_start(), you can store, read, and delete session variables easily, making it a powerful tool for user login systems, shopping carts, and other web applications that require persistent data.