Cookies, Sessions and Authentication

Understanding the Basics

- **Cookies:**
  - Small pieces of data stored on the client's browser.
- **Sessions:**
  - Server-side storage of user data during a visit.
- **Authentication:**
  - Verifying the identity of users.

Cookies

- A common method of user identification involves the use of cookies, which are small files that the server implants on the user's computer.
- When the same computer requests a page using a browser, it sends along the associated cookie with each request.

Cookies

- **Definition:**
  - Small pieces of data stored on the client's browser.
- **Purpose:**
  - Tracking user activity, personalization, and authentication.
- **Types:**
  - Session cookies (expire when the session ends) and persistent cookies (stored for a specified duration).
- **Pros:**
  - Lightweight
  - User personalization
  - Persistent data storage
- **Cons:**
  - Security concerns (Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) are both web security vulnerabilities.)
  - Limited storage capacity

Cookies

- **Setting a Cookie in *login.php***
  1. ***// Set a cookie to remember the user for 1 day (86400 seconds)***
  2. ***setcookie('remember_me', $username, time() + 86400, '/');***

- Here's a breakdown of the parameters:
  – 'remember_me': The name of the cookie.
  – $username: The value stored in the cookie, which is the username in this case.
  – time() + 86400: The expiration time of the cookie, calculated as the current time plus 86400 seconds (1 day).
  – '/': The path on the server for which the cookie is available. Using '/' makes the cookie available throughout the entire domain.

Cookies

- **Reading and Refreshing the Cookie in dashboard.php**

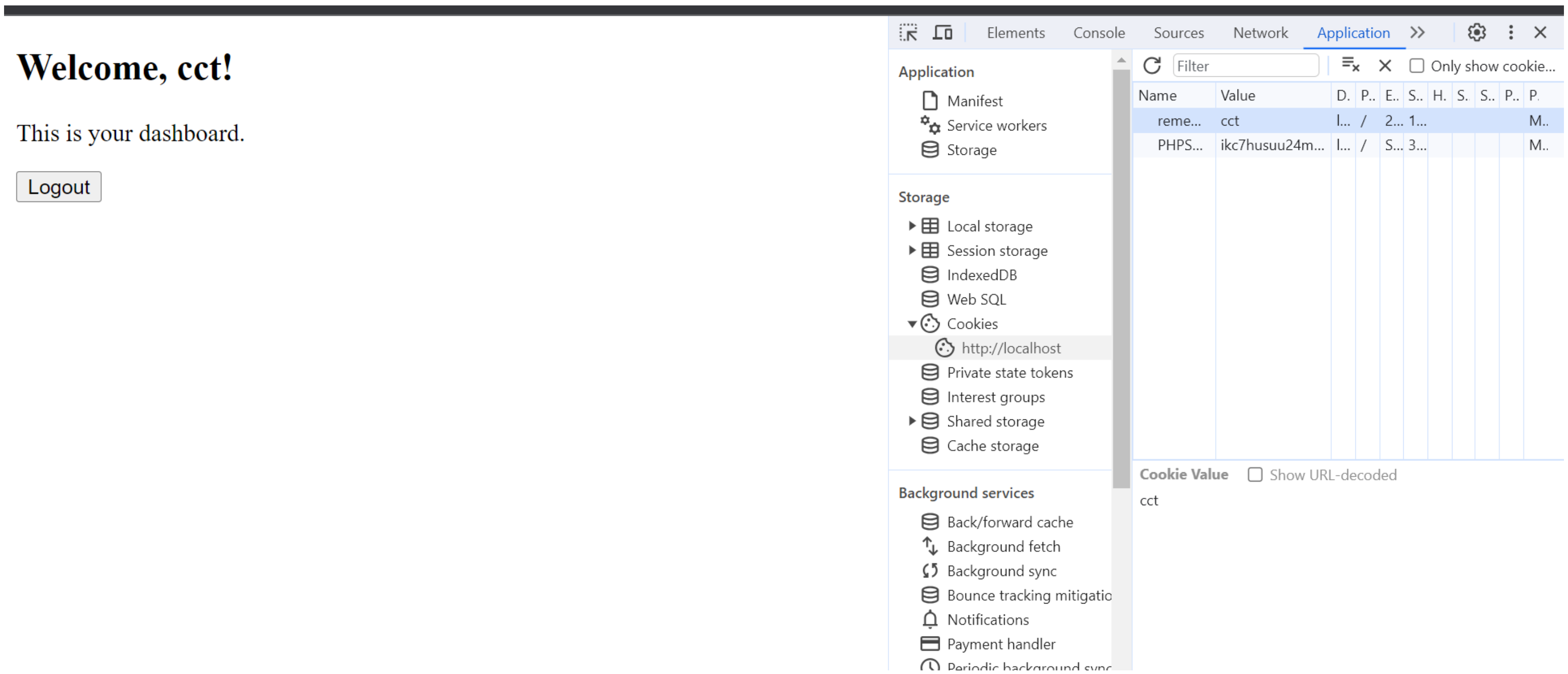
```php
if (isset($_COOKIE['remember_me'])) {
    $username = $_COOKIE['remember_me'];
    $_SESSION['username'] = $username;
    $_SESSION['last_activity'] = time();
    // Refresh the cookie expiration time
    setcookie('remember_me', $username, time() + 86400, '/');
}
```

- In the dashboard.php file, when a user accesses the dashboard, the code checks if the 'remember_me' cookie is set.
- If it is, the username is retrieved from the cookie and used to automatically log in the user.
- Additionally, the session's last activity time is updated, and the cookie's expiration time is refreshed to extend its validity.

Cookies

- **Deleting the Cookie on Logout in logout.php**
  - ***setcookie('remember_me', '', time() - 3600, '/');***
- In the logout.php file, when the user logs out, the 'remember_me' cookie is deleted by setting its expiration time to a past timestamp (time() - 3600).
- This effectively removes the cookie from the user's browser.

Cookies



Welcome, cct!

This is your dashboard.

Logout

Name	Value	D.	P..	E..	S..	H.	S.	S..	P..	P.
reme...	cct	l...	/	2...	1...					M..
PHPS...	ikc7husuu24m...	l...	/	S...	3...					M..

Cookie Value
cct

HTTP Authentication

- **Stateless Protocol (e.g., HTTP):** Each request is independent; the server doesn't retain past interactions. Example: HTTP, where each request contains all necessary information.
- **Stateful Protocol (e.g., FTP):** Maintains a continuous state between client and server across multiple interactions. Example: FTP, which retains session state for file transfers.
- HTTP authentication is a mechanism used to control access to certain parts of a website or web application by requiring users to provide valid credentials.
- There are several types of HTTP authentication, and one common method is Basic Authentication.

HTTP Authentication

- Server side code snippet:

```
<?php                // Username and password for demonstration purposes
$valid_username = 'demo_user';
$valid_password = 'demo_password';
// Check if the user has provided credentials
if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
   header('WWW-Authenticate: Basic realm="Restricted Area"');
   header('HTTP/1.0 401 Unauthorized');
   echo 'Authentication required.';
   exit; }
// Validate the provided credentials
if ($_SERVER['PHP_AUTH_USER'] !== $valid_username || $_SERVER['PHP_AUTH_PW'] !== $valid_password) {
   header('HTTP/1.0 401 Unauthorized');
   echo 'Invalid credentials.';
   exit; }
// Successful authentication
echo 'Welcome, ' . $_SERVER['PHP_AUTH_USER'] . '!';
?>
```

HTTP Authentication

- The server checks if the PHP_AUTH_USER and PHP_AUTH_PW variables are set in the incoming HTTP request headers.
- If not, it sends a 401 Unauthorized response along with a WWW-Authenticate header, prompting the browser to show an authentication dialog.
- If credentials are provided, the server validates them against the expected username and password.
- If the credentials are valid, it allows access; otherwise, it returns an Unauthorized response.

HTTP Authentication

- **Client-side Usage**
- When a user tries to access a resource protected by Basic Authentication, the browser prompts them with a login dialog where they enter the username and password.
- The credentials are then included in the Authorization header of subsequent requests.
  - ***<!-- Example Request Header -->***
  - ***GET /secure/resource HTTP/1.1***
  - ***Host: example.com***
  - ***Authorization: Basic ZGVtb191c2VyOmRlbW9fcGFzc3dvcmQ=***
- In the Authorization header, the word 'Basic' is followed by a base64-encoded string of the form username:password.

HTTP Authentication

- Note: While Basic Authentication is simple, it's generally recommended to use it over HTTPS to encrypt the credentials during transmission, as the base64 encoding alone does not provide security.
- Additionally, more secure authentication methods like Token-based or OAuth are often preferred for production applications.

Session

- **Exploring Server-Side Storage**
  - **Functionality:** Stores user data across multiple requests.
  - **Implementation:** Server generates a unique session identifier.
  - **Security Measures:** Timeout mechanisms and secure session handling.
- A session provides a means to retain information (stored in variables) for use across multiple pages.
- In contrast to a cookie, this information is not saved on the user's computer.

Session

- **Definition:**
  - Server-side storage of user data during a visit.
- **Functionality:**
  - Maintains state across multiple requests.
- **Pros:**
  - More secure than cookies
  - Server controls data
  - Session timeout for security
- **Cons:**
  - Server overhead
  - Requires storage management

Session

- **Session Concept:**
  - Resembles working with an application: open, make changes, and close.
  - Similar to a computer knowing your activity, but web servers lack this awareness due to stateless HTTP.
- **Issue with HTTP:**
  - Web server doesn't recognize users or their actions due to the stateless nature of HTTP.
- **Solution: Session Variables:**
  - Store user information (e.g., username, preferences) for use across pages.
  - Information lasts until the user closes the browser.
- **Functionality:**
  - Session variables are specific to one user.
  - Accessible across all pages within an application.

Session

- A session is started with the session_start() function.
- Session variables are set with the PHP global variable: $_SESSION.
- **Note:** The session_start() function must be the very first thing in your document. Before any HTML tags.
- To remove all global session variables and destroy the session, we use
  - session_unset() and
  - session_destroy():

Cookies vs Session

- **Choosing the Right Tool**
  - **Cookies:**
    - Lightweight, client-side storage.
  - **Sessions:**
    - Server-side storage, more secure.
  - **Decision Factors:**
    - Security requirements, data size, and persistence.

Cookies vs Session

Demo file:

Cookie: https://github.com/neupaneprakash/webTech_II/blob/main/cookies.zip

Auth: https://github.com/neupaneprakash/webTech_II/blob/main/authentication.zip

Session: https://github.com/neupaneprakash/webTech_II/blob/main/session.zip